Retaining Skills under Distribution Shifts: Sequential Bayesian Inference, Reinforcement Learning and Applications

Samuel Charles Kessler

New College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Hillary 2023

This thesis is dedicated to
Malu and Charles

Acknowledgements

I would like to thank **Steve** for giving me the freedom to explore my own ideas, for providing stimulating research discussions, and for shielding me from administrative issues and funding politics. I wish the pandemic hadn't taken away all those possible lunchtime discussions we could have had.

I would like to thank my assessors **Prof. Jakob Foerster** and **Prof. Tim Hospedales** for the feedback and probing questions during my thesis viva. I enjoyed our deep discussions and your feedback has made this thesis immeasurably more coherent, technically correct, and clearer to read.

I would like to thank my collaborators **Vu**, **Jack**, **Phil**, **Salah**, **Adam**, **Tim** and **Piotr**. I learned a tremendous amount about machine learning from you guys, from Bayesian non-parametrics to self-supervised learning. You taught me how to do good research, how to write machine learning research papers, and about the paper review process. You made research fun and exciting, and I enjoyed all of our discussions.

Thanks **Stefan** for allowing me to work on the Badam paper, this was a great way to start my PhD, allow me to understand the paper review process, and introduce me to the most important concepts in Bayesian Deep Learning.

On a different note, I wouldn't have been able to get through my PhD with my mental health intact if it wasn't for my fellow rowers at New College Boat Club and the great coaching from the Brookes lads Sam, Barney, Scott, Jake, and last but not least Hugo - "more is more"! If only I had learned that at the start of my PhD...

I would also like to thank the coffee shops on Walton Street: 101 Coffee and Branca for being a constant sink of my PhD funding. I'm also not sure how I would have done it without the supply of Manos and Opera wraps. Last but not least I would like to thank Wharf 24: Giuseppe, Arno, Fede, and Max.

Abstract

Modern machine learning models, such as neural networks which are the focus of this thesis, have been shown to be extremely powerful tools for learning function mappings from inputs to outputs, for example, for image-to-categories or audio-to-text tasks. However, when learning over a set of sequential tasks, one after another, they struggle to recall past tasks. Neural networks, *forget* previous abilities they have learned when learning new tasks. The requirement for neural networks to learn multiple tasks sequentially is an important step toward creating generally capable intelligent agents. In particular, neural networks suffer from a *stability-plasticity* dilemma in which there is a trade-off between learning new tasks and remembering previously learned tasks. Here *plasticity* refers to the ability of neural networks to learn new tasks quickly, and *stability* refers to the desirable quality of recalling and remembering past tasks, so the network is stable in performing previously mastered tasks. This is a trade-off since neural networks can learn new tasks efficiently from new data via gradient-based optimization. However, this can lead to *forgetting* of previously learned abilities from previous tasks, since gradient updates can shift parameters away from regions that are optimal for previous tasks. We can also have a reverse situation where the neural network is unable to learn a new task and new learning is prevented while at the same time, the neural network is able to remember how to solve past tasks, but this means that the neural network isn't plastic to learning the new task.

In pursuit of this goal, we introduce new algorithms anchored in sequential Bayesian inference. These algorithms use the principle of "yesterday's posterior is tomorrow's prior" for learning sequentially and accumulating knowledge. Firstly, the thesis introduces an algorithm that leverages the Bayesian non-parametric Indian Buffet Process prior to enable a Bayesian neural network to expand, adding more neurons as more data and more tasks are seen by the agent, to enable further *plasticity*. A variational approximation of the posterior enables us to learn sequentially using Bayesian inference for posterior learning. Taking the previous task's posterior as a prior and enabling the Bayesian neural network to recall previous tasks. Furthermore, we present work that looks to perform exact sequential Bayesian inference without the need for such a variational approximation. We test whether having access to the true posterior can alleviate forgetting by using Hamiltonian

Monte Carlo, discuss sequential Bayesian inference's limits, and address these by proposing a probabilistic model of the continual learning data-generating process.

Reinforcement learning is a rich area in which to study continual learning. We require the agent to learn a policy that solves multiple environments sequentially while remembering how to solve previous environments and continually explore new environments. We present work in partially observable environments noting that we may observe different tasks that share the same state and action-spaces but have different reward functions or goals. This has important consequences when using experience replay as a continual learning technique. Additionally, we propose using world models in the model-based reinforcement learning paradigm as a straightforward baseline for continual reinforcement learning.

A practical application of continual learning involves speeding up the training of very large transformer-based models by effectively transferring features from an initial task to the learning of new tasks. For instance, training large automatic speech recognition models to transcribe English speech to text, takes days (if not weeks) to train. We show that we can effectively use the features from such a model to speed up the training of a new European language, without losing the ability to transcribe English. This approach can be applied to the updating of any large transformer-based models in light of new data to speed up training for quicker experimentation for new tasks.

Declaration

I, Samuel Charles Kessler, declare that all the work included in this thesis is my own original work.

The chapters in this thesis consist of work that has been peer-reviewed and accepted at various conferences and workshops or both. In this thesis, the chapters encompass different papers and I connect each chapter with the relevant publication and state my contribution and that of my co-authors.

Chapter 4. The work in this chapter contains the following work (Kessler et al., 2021a):

Kessler, S., Nguyen, V., Zohren, S. and Roberts, S.J., 2021, December. "Hierarchical Indian Buffet Neural Networks for Bayesian Continual Learning". In *Uncertainty in Artificial Intelligence (pp. 749-759)*. PMLR.

The IBNN and HIBNN models for Bayesian Continual Learning were developed with Nguyen. I did all the experimental work. The interpretation of results was done with Nguyen and Zohren. Nguyen, Zohren and Roberts helped with writing.

Chapter 5. The material presented in this chapter is based upon the following work (Kessler et al., 2023):

Kessler, S., Cobb, A., Rudner, T.G., Zohren, S. and Roberts, S.J., 2023. "On Sequential Bayesian Inference for Continual Learning". *arXiv preprint arXiv:2301.01828*.

This work has been accepted at Entropy and an earlier version of this work was presented at AABI 2022. I was the main contributor to the ideas, experiments, interpretation, and writing. Roberts helped develop some of the ideas. Cobb helped with the development of the ideas and the implementation of HMC with a density estimator as a prior. Rudner ran the S-FSVI baselines for the experiments. Rudner, Cobb and Roberts helped with paper writing.

Chapter 6. The work in this chapter contains the following publication (Kessler et al., 2021b):

Kessler, S., Parker-Holder, J., Ball, P., Zohren, S. and Roberts, S.J., 2022, June. "Same state, different task: Continual reinforcement learning without interference". In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 36, No. 7, pp. 7143-7151).

The ideas in this paper were generated with Parker-Holder and Ball. I did all the experimental work. The interpretation of the results was done with Parker-Holder and Ball. The writing of the paper was done with Parker-Holder, Ball, and Roberts.

Chapter 7. The work contained in this chapter contains work from the following (Kessler et al., 2022a).

Kessler, S., Miłoś, P., Parker-Holder, J. and Roberts, S.J., 2022. "The Surprising Effectiveness of Latent World Models for Continual Reinforcement Learning". *arXiv preprint arXiv:2211.15944.*

This paper has been accepted at COLLAs 2023 and appeared in the Deep Reinforcement Learning workshop at Neurips 2022. The ideas were developed with Parker-Holder. I did all the experimental work. The interpretation of the results was done with Miłoś. The paper writing was done with Miłoś with small contributions from Parker-Holder and Roberts.

Chapter 8. Contains work from the publication (Kessler et al., 2022c):

Kessler, S., Thomas, B. and Karout, S., 2022, May. "An Adapter Based Pre-Training for Efficient and Scalable Self-Supervised Speech Representation Learning". In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 3179-3183). IEEE.

This paper also appeared at the ICML SSL workshop 2021 entitled "Continual-wav2vec2: an application of continual learning for self-supervised automatic speech recognition". Karout and I developed the ideas for this paper. I did all the experimental work. Thomas and Karout helped on the interpretation of the results. Thomas and Karout helped to write the paper.

Other work. Other papers which I worked on during my DPhil but are not included in my thesis include:

Kessler, S., Salas, A., Tan, V.W., Zohren, S. and Roberts, S., 2018. "Practical Bayesian Learning of Neural Networks via Adaptive Optimisation Methods". *arXiv preprint arXiv:1811.03679*.

This was presented at the ICML 2020 Workshop on Uncertainty and Robustness in Deep Learning. We showed how we can obtain a Bayesian Neural Network with a Laplace approximate posterior using curvature estimates from adaptive optimization methods like Adam with no additional effort (Kessler et al., 2018).

Hergueux, J. and Kessler, S., 2022, April. "Follow the leader: Technical and inspirational leadership in open source software". In *CHI Conference on Human Factors in Computing Systems* (pp. 1-15).

We use an open dataset from `github.com` of open-source-software projects to measure the communication and technical leadership qualities of developers (Hergueux and Kessler, 2022).

Thomas, B., Kessler, S. and Karout, S., 2022, May. "Efficient Adapter Transfer of Self-Supervised Speech Models for Automatic Speech Recognition". In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 7102-7106). IEEE.

Instead of finetuning a large self-supervised automatic speech recognition network we can finetune a small number of new adapter layers which is more efficient and allows transfer to new languages (Thomas et al., 2022).

Wood, K., Kessler, S., Roberts, S. J., & Zohren, S., 2023, October. "Few-Shot Learning Patterns in Financial Time-Series for Trend-Following Strategies". *arXiv preprint arXiv:2310.10500*.

This work has been accepted for publication in the *Journal of Financial Data Science*. To quickly adapt to changing market conditions, we develop an algorithm that can place long and short daily positions and can quickly adapt to changing market conditions by leveraging few-shot learning and change-point detection methods (Wood et al., 2023).

Contents

1 Introduction

There have been many recent high-profile successes of machine learning; in image classification (Krizhevsky et al., 2017), machine translation (Bahdanau et al., 2014), text generation (Devlin et al., 2018; Radford et al., 2019), automatic speech recognition (Baevski et al., 2020), games (Schrittwieser et al., 2019), robotics (Andrychowicz et al., 2020) and in scientific applications (Jumper et al., 2021; Degrave et al., 2022). These tremendous successes showcase methods capable of solving individual tasks, and in some cases, exceeding human performance in these tasks (Mnih et al., 2015; Silver et al., 2016b). However, there is still a large gap between human capabilities and the capabilities of these task-specific AI systems. Looking ahead, it is conjectured that truly scalable intelligent systems will additionally need to master many tasks in a continual manner (Ring, 1994; Hassabis et al., 2017). The field of *continual learning* (CL) aims to develop agents which can solve many tasks, one after the other, whilst still retaining performance on all previously seen tasks (Sodhani et al., 2022; Khetarpal et al., 2020).

One of the key challenges of neural network (NN) based systems today, is the ability to transfer knowledge from previously encountered tasks and to learn new tasks without losing prior task ability. The key issue with gradient-based learning of parametric models such as NNs, is that parameters that were optimal for prior tasks are "overwritten" during gradient-based learning of new tasks Fig. 1.1, which leads

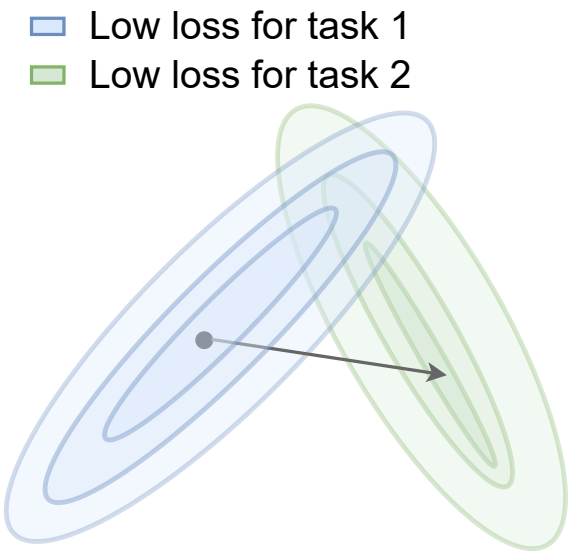


Figure 1.1: Illustrative example of the stochastic gradient descent trajectory in parameter space. Parameter regions of good performance for task 1 in blue and for task 2 in green. After learning on task 1 the parameters are optimal for task 1. Afterward, if we take gradient steps according to task 2 alone, the gray arrow, then the optimal parameters will shift to areas of low loss for task 2, however, there is no guarantee that this will result in an area of low loss for task 1. Hence the forgetting problem.

to *catastrophic forgetting* of previously learned abilities (McCloskey and Cohen, 1989; French, 1999; Mermillod et al., 2013; Goodfellow et al., 2013). On the other hand, we also want to ensure that continual learning systems enable *positive forward transfer* such that previous abilities, which models have learned, enable them to master new tasks better than if they were trained from scratch. This idea of forward transfer is tightly related to *curriculum learning*, where constructing a sequence of tasks of increasing difficulty enables an AI system to solve hard tasks better than learning from scratch (Bengio et al., 2009; Wang et al., 2019). Finally, we desire models to continually learn new abilities, which are scalable rather than having to introduce a new expert for each task (Jordan and Jacobs, 1994).

1.1 Why are Probabilistic Models for Continual Learning Important?

Firstly it is important to answer the question of why probabilistic models of NNs themselves are useful. In the probabilistic or Bayesian approach, we wish to place distributions over the parameters of our models and infer the parameters of these distributions in what is referred to as a *posterior* distribution. We can obtain a distribution over our target prediction y by calculating the posterior predictive distribution of y for some test input point $\boldsymbol{x}$, using the marginal integral:

$$p(y|\boldsymbol{x}, \mathcal{D}) = \int p(y|\boldsymbol{x}, \theta)p(\theta|\mathcal{D})d\theta, \tag{1.1.0.1}$$

where θ are the parameters of the NN model and which themselves have a posterior distribution, $p(\theta|\mathcal{D})$, which we want to marginalize over. This marginalization can be thought of as a summation over many different versions of the NN with different parameters weighted by their probability under their posterior. This approach generalizes more standard maximum likelihood approaches that assign all probability mass to a single NN estimate, $\boldsymbol{\theta}_{\text{ML}}$. By marginalizing over the posterior via Eq. (1.1.0.1) we can improve the accuracy and calibration of our predictions (MacKay, 1992b; Wilson and Izmailov, 2020).

Another important feature of probabilistic or Bayesian models is their ability to update our beliefs about a model and its parameters in the light of new data. This can be achieved by using Bayes' theorem and using the principle of "yesterday's posterior is today's prior" to update our beliefs in the face of new data:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})}. \tag{1.1.0.2}$$

Our beliefs are encompassed by the model $\mathcal{M}$, here a NN. The model $\mathcal{M}$ has a prior probability distribution over its parameters, denoted by $p(\theta)$. The likelihood of the data $\mathcal{D}$ under the model is $p(\mathcal{D}|\theta)$ and evidence $p(\mathcal{D})$ is a normalization constant. We can perform multiple rounds of Bayesian updates and recursively build up our posterior by sequentially observing data (Winther and Solla, 1998; Opper and Winther, 1998; Zeno et al., 2018). This framework, which allows us to

update our beliefs in light of new data, also enables us to incorporate information from new tasks into our model. This is explored as a means for continual learning in Section 2.2. In this thesis, we consider NNs, since they are complex deep function approximator models with a broad hypothesis space and so are a suitably well-specified model $\mathcal{M}$ to consider for continual learning problems. For example, if we are working with images then a suitable model class for $\mathcal{M}$, would be convolutional NNs LeCun et al. (1998), which would be a more suitable model than multi-layer perceptrons (Wilson and Izmailov, 2020).

Another aspect of Bayesian or probabilistic models that is useful is that they are better at knowing what they do not know, or in other terms have better uncertainty quantification. This is due to having a distribution over parameters; there is uncertainty in the value of individual NN parameters. This has been shown to be useful for deep active learning (Filos et al., 2019) and also for calibrating uncertainties even under distributional shift where predictions are inherently less confident (Ovadia et al., 2019a).

1.2 Thesis Organization and Overview

In this thesis, we first consider the motivation for continual learning. This chapter, looks at several example applications, starting with a personalized object recognizer Section 1.3.1, robotic home assistants Section 1.3.2, and finishes by considering how to effectively manage large pre-trained models Section 1.3.3. In Chapter 2, we present some formal background material to better contextualize the research in the main chapters, for example defining continual learning and continual reinforcement learning. In Chapter 3, we provide a literature review of continual learning and continual reinforcement learning.

In Chapter 4 we explore the use of sequential Bayesian inference with a non-parametric prior to enable a Bayesian NN to expand and add new neural resources when learning new tasks. In Chapter 5 we introduce our work which revisits Bayesian continual learning and outlines some important caveats. As a result,

we propose modeling the data-generating process of continual learning as a more effective avenue to develop probabilistic models for continual learning.

In Chapter 6 we explore continual reinforcement learning (CRL) and some interesting pathologies that do not arise in continual supervised learning. We also provide a very simple procedure to mitigate these pathologies. We use a deep ensemble (Lakshminarayanan et al., 2017a) of action-value functions which allows us to use robust uncertainties as feedback to our agent when performing decisions under uncertainty. Deep ensembles of models provide a good approximation of the posterior predictive distribution (Wilson and Izmailov, 2020), and are well calibrated under increasing input data corruption (Lakshminarayanan et al., 2017a; Ovadia et al., 2019a). In Chapter 7, we introduce a simple and effective baseline using world models (Ha and Schmidhuber, 2018) in model-based reinforcement learning for CRL. For this simple yet effective baseline, we also use deep ensembles (Lakshminarayanan et al., 2017a) to obtain a measure of uncertainty for the generated trajectories; this enables us to perform task-agnostic exploration in CRL. In this paradigm, the exploration adapts automatically to each new task presented, hence we do not need to tell the algorithm when the task has changed. Previous methods, in comparison, require additional task information to continually explore new environments. Finally, in Chapter 8, we introduce an application of CL to efficiently update a large self-supervised model in the face of new incoming data.

1.3 Example Applications of Continual Learning

We introduce three different example applications of continual learning which help to contextualize the importance of continual learning research and the work in this thesis.

1.3.1 Personal Object Recognizer

Consider a personal assistant headset device that has cameras and speakers, such technologies are currently used for AR/VR applications with headsets such as the Oculus Quest and Microsoft HoloLens. The cameras are all at the level of one's

Figure 1.2: Video of 10 frames from an egocentric personal object recognizer where the only object being detected with a red bounding box is Pine Sol disinfectant. The video is from the Orbit dataset (Massiceti et al., 2021). The colour of the bounding box indicates the class of the object.

head and so have the same view as a person, this view is called "egocentric". For instance, egocentric action recognition has been used for action recognition of hand gestures (Han et al., 2020). Another application is for helping blind people to detect their own household objects (Massiceti et al., 2021). The headset cameras act as eyes and the audio components are able to provide assistance to which objects are currently in view. Enabling a personal object recognizer to quickly detect new objects with few-shot learning (Vinyals et al., 2016; Snell et al., 2017; Finn et al., 2017) would enable a blind person to start to track new personal objects. More precisely, from video, we require the personal object recognizer to pick out and detect different objects (Redmon et al., 2016) which are in the user's field of view. See Fig. 1.2 for an example of this situation from the ORBIT dataset where the personal object recognizer has detected some Pine Sol cleaner (Massiceti et al., 2021).

Now consider the scenario where a brand new object, which has not been seen before, is required to be tracked by a user. For instance, the video in Fig. 1.3 has keys that are already recognized by the personal object recognizer and can be seen in the first 5 frames. In the final frame, a new object comes into view and the personal object recognizer detects it as not coming from the current closed-set of objects that it has seen before, but from a distinct open-set, in frame number 5 (Scheirer et al., 2012). In Fig. 1.3 the corresponding categorical distribution over

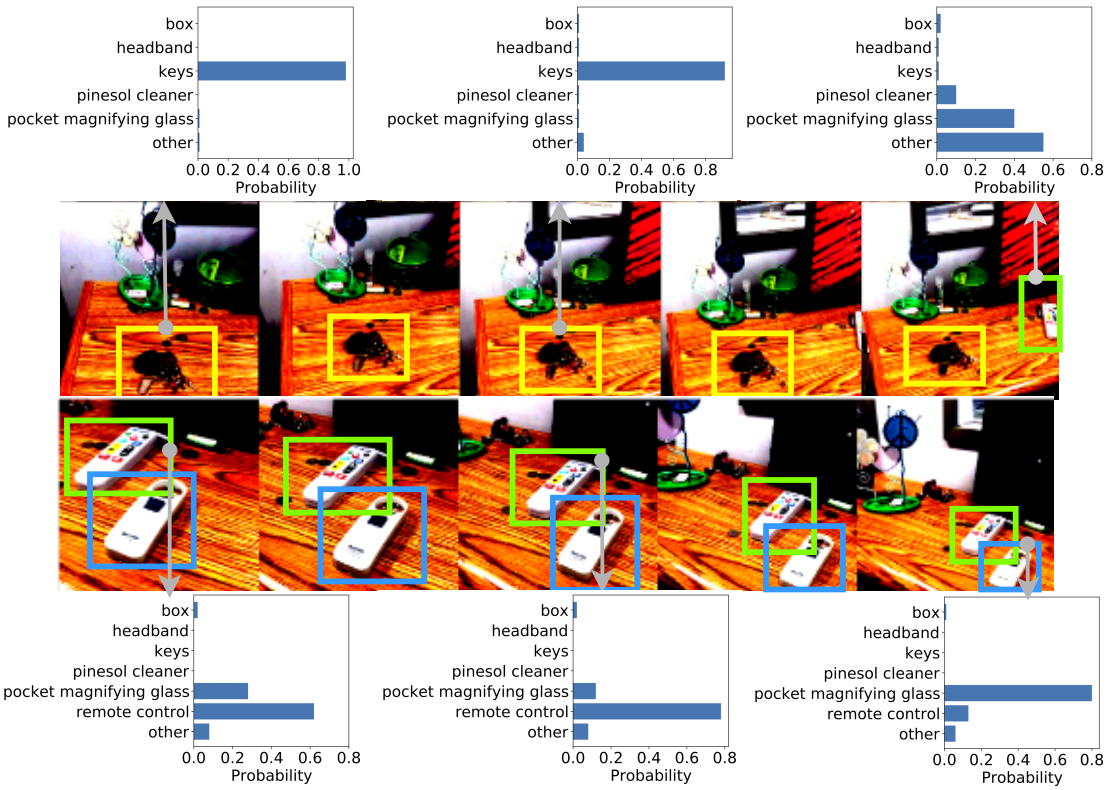


Figure 1.3: Video of 10 frames from an egocentric personal object recognizer where each bounding box denotes an object being detected, each bounding box colour denotes a different object or class. In the first 5 frames keys are detected, in frame 5 a new object the remote control needs to be continually learned. In this scenario, a personal object recognizer needs to start to track a new class using few-shot learning. At the same time, the personal object recognizer needs to retain performance on all the objects seen so far required by the user. Notice how in the final 5 frames we have learned to detect a new class and added a class to the categorical distribution, the user's remote control, and we require that the personal object recognizer retains knowledge of past classes in this update, like the pocket magnifying glasses. The video is from the Orbit dataset (Massiceti et al., 2021).

each bounding box that has been detected using the object detection algorithm, such as YOLO (Redmon et al., 2016), is also visualized in the top and bottom rows. As the user has just bought a new home appliance with a remote control, the user indeed requires the personal object recognizer to track this brand new object, their remote control. In the same frame, a second object that the user has already been tracking, a pocket magnifying glass, has also been detected in the blue bounding box. To start to detect the new remote control, from only a few instances gathered by the user, is a few-shot learning problem, requiring rapid

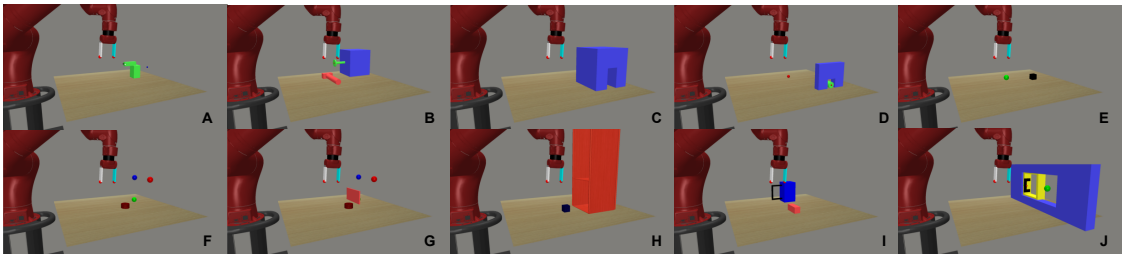


Figure 1.4: Visualization of the `ContinualWorld` tasks (Wolczyk et al., 2021). A robotic agent needs to solve 10 different tasks continually, one after another without forgetting how to solve each task in the sequence. The tasks are **A**, `faucet-close-v1` where the agent needs to close a tap. **B**, `hammer-v1` where the agent needs to hammer a nail. **C**, `handle-press-side-v1` where the agent needs to grip a handle. **D**, `peg-unplug-side-v1` where the agent needs to remove a plug from the big blue box. **E**, `push-back-v1` where the agent needs to hockey puck backward to the green goal. **F**, `push-v1` where needs to push the hockey puck to the green goal. **G**, `push-wall-v1` where the agent needs to push the hockey puck to the wall. **H**, `shelf-place-v1` where the agent needs to place the object on the shelf. **I**, `stick-pull-v1`, where the agent needs to pull a handle. **J**, `window-close-v1` where the agent needs to close a window.

learning of a new object. Furthermore, objects that the personal object recognizer has previously been tracking need to continue to be tracked. That is, learning to track a new remote control object doesn't cause forgetting of the previous objects; in this case, the box, headband, and Pine Sol disinfectant objects, which have not been observed in the past 10 frames Fig. 1.3. So training a personal assistant on new objects is a continual learning problem.

1.3.2 Robotic Home Assistant

One of the major potential applications of reinforcement learning (RL, introduced in Section 2.3) is for robotics (Plappert et al., 2018; Andrychowicz et al., 2020). Let's consider a robotics application which is that of a robotic home assistant that helps with various tasks around the house. We can imagine a robot that is deployed around the house to perform various chores, from loading the dishwasher to mowing the lawn.

Let's say that the owner of the robot would like the robot to perform a new task for which the robot requires task-specific knowledge from the owner and the environment. For instance, consider the seemingly simple task of loading the dishwasher. Even this simple pick-and-place task requires considerable environment-specific information which is unlikely to have been observed during training; from

what the homeowner's dishes look like, to the location of the dishwasher in the kitchen. Another example is gardening: this also requires the robot to learn new environment-specific features that it might not have seen during training before deployment at this home. We can envisage task-specific knowledge that the robot needs to learn, such as the layout of the garden and how to operate the lawnmower. The owner could design a handcrafted reward function for the robot so that it can learn this new task, exploiting the fact that RL agents learn to maximize the expected reward (Sutton et al., 1998). Alternatively, the homeowner could demonstrate to the robot a few expert trajectories to enable the robot to learn from, as is done in imitation learning (Pomerleau, 1991; Ho and Ermon, 2016). Both of these example approaches are valid and what they have in common is that they require learning a new policy for these specific tasks. This learning needs to happen without affecting the policies of currently mastered tasks and without affecting any potential pre-trained representations that allow the agent to adapt quickly to new tasks. If the learning is gradient-based, then learning a new task might push away the optimal parameters of previous tasks when learning new policies. This will involve continual learning considerations to enable learning multiple tasks over time without forgetting any previously obtained skills.

These problems are similar to the continual reinforcement learning (CRL) algorithms, developed for robotic agents to continually learn new skills without forgetting previously mastered skills - such as in the `ContinualWorld` (Yu et al., 2020b; Wolczyk et al., 2021) CRL benchmark Fig. 1.4. In this benchmark, the robot arm is continually required to close taps, hammer nails, and move objects to certain locations. Once it has mastered how to hammer a nail, under a certain budget of environment interactions, it then needs to master a new skill - such as placing an object on a shelf without forgetting how to hammer a nail and without being able to interact with this previous environment.

1.3.3 Managing Large Models

Recent advances in machine learning have shown how results can improve substantially for many different applications by scaling to enormous datasets using larger and larger models. State-of-the-art results have been achieved for natural language processing (NLP), where for instance the recent `PaLM` language model (Chowdhery et al., 2022) is trained on a multilingual language dataset and source code comprising 780B tokens (words and code have a unique identifier called a token with an associated learned embedding vector (Mikolov et al., 2013)). The largest instantiation of the `PaLM` model has 540B parameters and is trained using 6144 TPUs (Jouppi et al., 2017). Similar scaling has been shown in automatic speech recognition (ASR) where models which are able to leverage enormous amounts of data by using larger and larger models also obtain state-of-the-art results when transcribing speech-to-text. For instance, the recent ASR model `Whisper` (Radford et al., 2022) trains on $680,000$ hours of audio and the largest `Whisper` model has 1550M parameters. To put this dataset and model size into context it takes 2 days to train the ASR model, `wav2vec2.0` on 1000 hours of audio on 64 Nvidia V100 GPUs. The `wav2vec2.0` base model has 95M parameters in comparison to `Whisper`'s 1550M. So scaling the number of parameters and the data size by 2 orders of magnitude will probably take much more compute and a lot longer to train.

Despite a large language model's performance on NLP tasks, we note that they suffer from degraded performance when asked to predict future utterances from beyond their training period (Lazaridou et al., 2021). For example, in the year 2022, the prime minister of the UK could have been anyone of Boris Johnson, Liz Truss, or Rishi Sunak. This points to the necessity to efficiently update large models with new data, keeping these large models factually correct in the face of a non-stationary world. These models require excessive resources to train, and hence it is inefficient to simply retrain these models from scratch on the union of past data and current data. Moreover, regulatory restrictions, such as GDPR, mean that one cannot simply store user data indefinitely[1]. One aspect of continual

[1] ec.europa.eu/info/law/law-topic/data-protection

learning which is under-studied, due to computational resource constraints, is how retraining these large models can be achieved in the face of new data from the internet - a highly non-stationary environment.

2
Preliminaries

2.1 Continual Learning

Continual learning (CL) considers a setting whereby a model must sequentially learn to provide predictions over a set of tasks whilst maintaining performance across all previously learned tasks. In CL, the model is sequentially shown T tasks, denoted $\mathcal{T}_t$ for $t = 1, \ldots, T$. Each task, $\mathcal{T}_t$, is comprised of a dataset $\mathcal{D}_t = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N_t}$: from which a model must learn the task solution. More generally, tasks are denoted by distinct tuples comprised of the conditional and marginal data distributions, $\{p_t(y|\mathbf{x}), p_t(\mathbf{x})\}$. After task, $\mathcal{T}_t$ the model loses access to the training dataset, but its performance is still continually evaluated on all tasks $\mathcal{T}_i$ for $i \leq t$. For a thorough overview of the continual learning scenarios and design choices encountered in this thesis, refer to Section 2.5.

As an example of a continual learning problem, we can consider the CIFAR10 dataset. We can split the dataset into 5 two-way classification tasks. This problem is called Split-CIFAR10 (Zenke et al., 2017a), in which the first task involves the classification of airplanes and automobiles, (classes $\{0, 1\}$) followed by a second task comprising the classification of birds and cats, (classes $\{2, 3\}$) and so on Fig. 2.1.

Figure 2.1: The common Split-CIFAR10 continual learning benchmark of 5 consecutive 2-way classification problems. The first task involves classifying airplanes versus automobiles, then birds versus cats, and so on. At the end of training on the data from a particular task the continual learner is tested on test sets from all previous tasks.

2.2 Bayesian Continual Learning

We consider a setting in which task data arrives sequentially at time steps, $t = 1, 2, \ldots, T$. At the first time step, $t = 1$, that is for task $\mathcal{T}_1$ the model receives the first dataset $\mathcal{D}_1$ and learns the conditional distribution $p(y_i|\boldsymbol{x}_i, \theta)$ for all $(\boldsymbol{x}_i, y_i) \in \mathcal{D}_1$ (i indexes a datapoint in $\mathcal{D}_1$). We denote the parameters θ as having a prior distribution $p(\theta)$ for $\mathcal{T}_1$. The posterior predictive distribution for a test point, $\boldsymbol{x}_1^* \in \mathcal{D}_1$ is hence:

$$p(y_1^*|\boldsymbol{x}_1^*, \mathcal{D}_1) = \int p(y_1^*|\boldsymbol{x}_1^*, \theta) p(\theta|\mathcal{D}_1) d\theta. \tag{2.2.0.1}$$

We note that computing this posterior predictive distribution requires $p(\theta|\mathcal{D}_1)$. For $t = 2$, a CL model is required to fit $p(y_i|\boldsymbol{x}_i, \theta)$ for $(\boldsymbol{x}_i, y_i) \in \mathcal{D}_1 \cup \mathcal{D}_2$. The posterior predictive distribution for a new test point $\boldsymbol{x}_2^* \in \mathcal{D}_1 \cup \mathcal{D}_2$ point is:

$$p(y_2^*|\boldsymbol{x}_2^*, \mathcal{D}_1, \mathcal{D}_2) = \int p(y_2^*|\boldsymbol{x}_2^*, \theta) p(\theta|\mathcal{D}_1, \mathcal{D}_2) d\theta. \tag{2.2.0.2}$$

The posterior must thus be updated to reflect this new conditional distribution. We can use repeated application of Bayes' rule to calculate the posterior distributions $p(\theta|\mathcal{D}_1, \ldots, \mathcal{D}_T)$ as:

$$p(\theta|\mathcal{D}_1, \ldots, \mathcal{D}_{T-1}, \mathcal{D}_T) = \frac{p(\mathcal{D}_T|\theta) p(\theta|\mathcal{D}_1, \ldots, \mathcal{D}_{T-1})}{p(\mathcal{D}_T|\mathcal{D}_1, \ldots, \mathcal{D}_{T-1})}. \tag{2.2.0.3}$$

In the CL setting we lose access to previous training datasets: however, using repeated applications of Bayes' rule Eq. (2.2.0.3), allows us to sequentially incorporate information from past tasks in the parameters θ. At $t = 1$, we have access to $\mathcal{D}_1$ and the posterior over parameters is:

$$\log p(\theta|\mathcal{D}_1) = \log p(\mathcal{D}_1|\theta) + \log p(\theta) - \log p(\mathcal{D}_1). \tag{2.2.0.4}$$

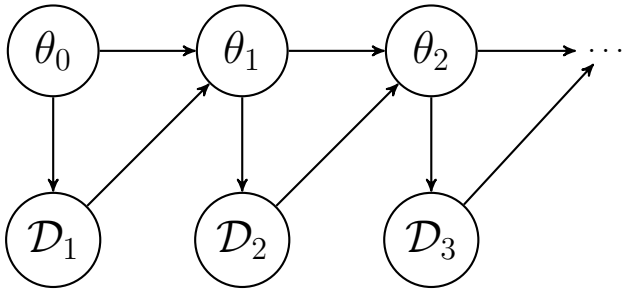


Figure 2.2: Bayesian network of the sequential inference process: the prior over θ_{t-1} generates the data under a likelihood model $p(\mathcal{D}_t|\theta_{t-1})$ and influences the posterior over θ_t. We use an index to explicitly denote the sequential Bayes updates.

At $t = 2$, we require $p(\theta|\mathcal{D}_1, \mathcal{D}_2)$ to calculate the posterior predictive distribution in Eq. (2.2.0.2). However, we have lost access to $\mathcal{D}_1$. According to Bayes' rule, the posterior may be written as:

$$\log p(\theta|\mathcal{D}_1, \mathcal{D}_2) = \log p(\mathcal{D}_2|\theta) + \log p(\theta|\mathcal{D}_1) - \log p(\mathcal{D}_2|\mathcal{D}_1), \tag{2.2.0.5}$$

where we used the conditional independence of $\mathcal{D}_2$ and $\mathcal{D}_1$ given θ. If we consider the following Bayesian Network Fig. 2.2 which represents the sequential Bayesian inference process, then by the rules of conditional independence $\mathcal{D}_1 \perp\!\!\!\perp \mathcal{D}_2|\theta_1$ which is equivalent to $\mathcal{D}_1 \perp\!\!\!\perp \mathcal{D}_2|\theta$, since the index in Fig. 2.2 denotes the Bayesian update, but is the same random variable and the same Bayesian NN parameters.

Also, from Eq. (2.2.0.5) we require that our model with parameters θ is a sufficient statistic of $\mathcal{D}_1$ i.e. the parameters θ "contain all the possible information" of $\mathcal{D}_1$ by being able to make perfect predictions on the dataset $\mathcal{D}_1$ Deisenroth et al. (2020). This observation motivates the use of high-capacity predictors, such as Bayesian neural networks, that are flexible enough to learn from $\mathcal{D}_1$. This ensures the conditional independence of the likelihood for tasks $t > 1$, i.e. $p(\mathcal{D}_2|\theta, \mathcal{D}_1) = p(\mathcal{D}_2|\theta)$, making the likelihood conditionally independent of $\mathcal{D}_1$ given θ.

This makes the likelihood in Eq. (2.2.0.5) $p(\mathcal{D}_2|\theta)$ only dependent upon the current task dataset, $\mathcal{D}_2$, and that the prior $p(\theta|\mathcal{D}_1)$ encodes parameter knowledge from the previous task. Hence, we can use the posterior evaluated at t as a prior for learning a new task at $t + 1$.

2.3 Reinforcement Learning

A Markov Decision Process (MDP, (Bellman, 1957)) is a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ in which $\mathcal{S}$ and $\mathcal{A}$ are respectively the sets of states and actions, such that for $s_t, s_{t+1} \in \mathcal{S}$ and $a_t \in \mathcal{A}$. The transition probability $P(s_{t+1}|s_t, a_t)$ is the probability that the system/agent transitions from s_t to s_{t+1} given action a_t and $R(a_t, s_t)$ is a reward obtained by an agent transitioning from s_t to s_{t+1} via a_t. The discount factor is represented by $\gamma \in (0, 1)$. Actions a_t are chosen using a policy π that maps states to actions $\pi : \mathcal{S} \rightarrow \mathcal{A}$.

The environments we consider in this thesis are partially observable; the agent does not have access to the environment state $s \in \mathcal{S}$. A Partially Observable Markov Decision Process (POMDP (Kaelbling et al., 1998)) is defined via the tuple $(\mathcal{S}, \mathcal{A}, P, R, \Omega, \mathcal{O}, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are respectively the sets of states and actions, such that for $s_t, s_{t+1} \in \mathcal{S}$ and $a_t \in \mathcal{A}$. $P(s_{t+1}|s_t, a_t)$ is the transition distribution and $R(a_t, s_t)$ is the reward function. The discount factor is $\gamma \in (0, 1)$, Ω is the set of observations, and $\mathcal{O} : \mathcal{S} \times \mathcal{A} \rightarrow P(\Omega)$ is an observation function that defines a distribution over observations. Actions a_t are chosen using a policy π that maps observations to actions: $\Omega \rightarrow \mathcal{A}$.

Let us assume we have access to the states s_t. In this thesis, we consider POMDPs with a finite horizon, H. The return from a state is defined as the sum of discounted future rewards $R_t = \sum_{i=t}^{H} \gamma^{(i-t)} R(s_i, a_i)$. In RL, the objective is to maximize $J = \mathbb{E}_{a_i \sim \pi, s_0 \sim p(s_0)}[R_1|s_0]$ given an initial state s_0, sampled from the initial state distribution $s_0 \sim p(s_0)$.

One approach to maximizing expected return is to learn an action-value function for each state-action pair: $Q^{\pi}(s_t, a_t) = \mathbb{E}_{s_t \sim P, r_t \sim R, a_t \sim \pi}\left[\sum_t R_t\right]$. We parameterize the action-value function with a neural network, denoted $Q(s_t, a_t; \boldsymbol{\theta})$, with parameters $\boldsymbol{\theta}$. We optimize $\boldsymbol{\theta}$ to minimize the expected temporal difference error using stochastic gradient descent of the loss function:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{s_t, a_t \sim \rho}[(y - Q(s_t, a_t; \boldsymbol{\theta}))^2]. \tag{2.3.0.1}$$

We use Q-learning (Watkins and Dayan, 1992), where $y = r_t + \gamma \text{max}_a Q(s_{t+1}, a; \bar{\boldsymbol{\theta}})$. The parameters $\bar{\boldsymbol{\theta}}$ are target network parameters, which are not optimized but updated periodically or using an exponentially weighted moving average of $\boldsymbol{\theta}$ (Mnih et al., 2015). The associated update does not require on-policy samples, so learning is off-policy using a replay buffer (Lin, 1992), hence ρ is an empirical distribution that represents samples from the replay buffer[1]. For continuous action settings, we learn a policy $\pi_{\boldsymbol{\phi}} : \mathcal{S} \rightarrow \mathcal{A}$ using a neural network parameterized by $\boldsymbol{\phi}$. For discrete action settings, our policy follows the maximum Q-value: $\pi \coloneqq \text{argmax}_a Q(s, a)$.

One approach to maximizing expected return is to use a *model-free* approach as above; learn a policy $\pi_\phi : \mathcal{S} \rightarrow \mathcal{A}$ with a neural network parameterized by ϕ guided by an action-value function $Q_\theta(s_t, a_t)$ with parameters θ. Instead of learning a policy directly from experience we can employ *model-based* RL (MBRL) and learn an intermediate model $f(\cdot)$, for instance, a transition model $s_{t+1} = f(s_t, a_t)$ from experience and learn our policy with additional experience generated from the model $f(\cdot)$ (Sutton, 1991). Instead of working with the actual state s_t our methods consider the observations o_t and recurrent action-value functions, policies, and models to help better estimate states s_t from observations (Hausknecht and Stone, 2015).

2.4 Continual Reinforcement Learning

In continual RL the agent has a budget of N interactions with each task environment $\mathcal{T}_\tau$. The agent is required to learn a policy to maximize rewards in this environment, before interacting with a new environment and having to learn a new policy. If the environments we are working with are POMDP's, each task is defined as a new POMDP, $\mathcal{T}_\tau = \left(\mathcal{S}_\tau, \mathcal{A}_\tau, P_\tau, R_\tau, \Omega_\tau, \mathcal{O}_\tau, \gamma_\tau\right)$. We can alter this definition straightforwardly if the environments we are working with are MDPs. In general, most real-world environments are POMDPs

The agent is continually evaluated on all past and present tasks so the agent's policy should transfer to new tasks while not forgetting how to perform past

[1] *On-policy* RL algorithms sample actions using the current policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ and optimize the current policy π simultaneously. *Off-policy* RL algorithms can learn a policy π from samples from a different (or older) policy $\rho : \mathcal{S} \rightarrow \mathcal{A}$, so samples can be stored and replayed for Q-learning.

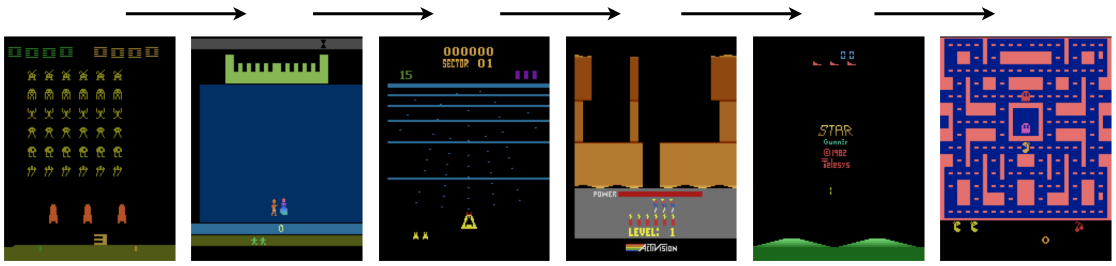


Figure 2.3: Example of a 6 task Atari CRL problem where the agent needs to learn to solve each game sequentially. After a budget of interactions with each environment, the agent needs to learn to solve a new Atari environment. The agent is continuously tested on all previous games to ensure that it can still solve them. This particular suite of tasks is `SpaceInvadersNoFrameskip-v4`, `KrullNoFrameskip-v4`, `BeamRiderNoFrameskip-v4`, `HeroNoFrameskip-v4`, `StarGunnerNoFrameskip-v4` and `MsPacmanNoFrameskip-v4`, in this order from the CORA suite (Powers et al., 2021).

tasks. CRL is not a new problem setting (Thrun and Mitchell, 1995a), however, its definition has evolved and some settings differ from paper to paper. In this thesis, we employ the setting above which is related to previous recent work in CRL (Kirkpatrick et al., 2016; Schwarz et al., 2018a; Rolnick et al., 2019; Kessler et al., 2021b; Powers et al., 2021; Caccia et al., 2022; Wolczyk et al., 2021). An example of a CRL problem is an agent continually solving multiple Atari games (Bellemare et al., 2013) Fig. 2.3.

2.5 Continual Learning Scenarios

In this section, I cover the continual learning scenarios which are the most common from the literature and which are used in the experimental setups in the main chapters of this thesis.

2.5.1 Supervised Continual Learning Experimental Scenarios

To evaluate different continual learning methods, we need to define the commonly used continual learning scenarios that are used in the literature and throughout this thesis. We use Split-CIFAR10 Zenke et al. (2017a) as an example. The CIFAR10 dataset which is commonly used as a computer vision benchmark is comprised of 10 classes with 50k training images The Split-CIFAR10 continual learning benchmark

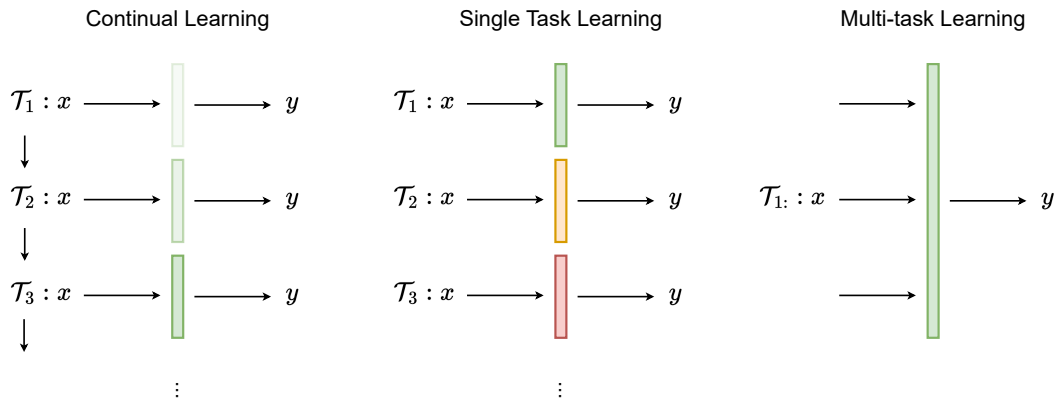


Figure 2.4: Overview highlighting the different between *continual learning*, *single-task learning* and *multi-task learning*. In *single-task learning*, we have a separate predictor per task, denoted by the use of separate colours. In contrast, in *multi-task learning* the learner uses the same predictor for all tasks simultaneously. In *continual learning* the same continual learning model is required to learn each task sequentially.

splits this dataset into 5, 2-way classification tasks with datasets of equal size 10k. The first task dataset is only comprised of images whose classes are in $\{0, 1\}$ while the second task dataset is only comprised of images whose classes are in $\{2, 3\}$ etc.

Previous work has introduced models for continual learning where the NN architectures used a feature extractor shared among all continual learning tasks, but a bespoke feature to output linear layer is trained for each task and then frozen Zenke et al. (2017a). Alternatively, other works have sought methods that do not require this manual selection of different feature to output linear heads and proposed a single feature to output linear layer which is shared among all tasks in continual learning Farquhar and Gal (2018). In this section, we categorize and systematically interpret the different continual learning scenarios, similar to previous important work Hsu et al. (2018a); Van de Ven and Tolias (2019).

In terms of notation a task $\mathcal{T}_\tau$ can be characterized by the conditional and marginal data distributions $p_\tau(y|\boldsymbol{x})$ and $p_\tau(\boldsymbol{x})$ and a task identifier τ (depending on the continual learning scenario, elaborated below) we denote samples from the distributions $y \sim p_\tau(y)$ and $\boldsymbol{x} \sim p_\tau(\boldsymbol{x})$.

Task incremental learning. This first scenario is generally the easiest scenario for continual learning. Each task is a subset of classes in the dataset where the input domains are disjoint: if the input distribution for the first and second tasks are $p_1(\boldsymbol{x})$

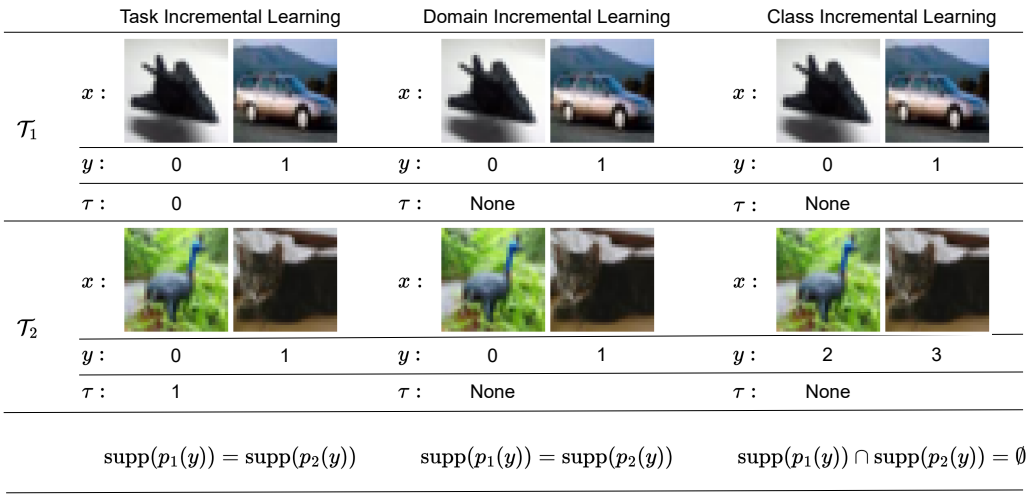


Figure 2.5: Three continual learning scenarios. Example datapoints from 2 tasks from the Split-CIFAR10 benchmark. The first task is a binary classification of airplanes versus automobiles and the second task is a binary classification of birds versus cats Fig. 2.1. In each row we have a different task, in each sub-row in each task the exact class y which needs to be predicted is enumerated, and the task identifier, τ, is shown. The support of the discrete class labels is defined as $\text{supp}(P(y)) = \{y \in \{0, \ldots, 9\} : P(y) > 0\}$.

and $p_2(\boldsymbol{x})$ with supports s_1^x and s_2^x respectively then $s_1^x \cap s_2^x = \emptyset$. At the same time, the output spaces are shared among all tasks, for output distributions $p_1(y)$ and $p_2(y)$ with supports s_1^y and s_2^y respectively then $s_1^y = s_2^y$. A task identifier is also available τ. For example, for Split-CIFAR10 all tasks are binary classification problems, and the classes are all mapped to $\{0, 1\}$ for each task. The task identifier can be used to select different linear layers for each task Kirkpatrick et al. (2016); Nguyen et al. (2017); Zenke et al. (2017a) — described as a multi-head network. The task identifier can be used during training and for evaluation. This is illustrated in Figure 2.5.

Domain incremental learning. In this scenario the domain explicitly increases for each task. The input domains are disjoint: if the input distribution for the first and second tasks are $p_1(\boldsymbol{x})$ and $p_2(\boldsymbol{x})$ with supports s_1^x and s_2^x respectively then $s_1^x \cap s_2^x = \emptyset$. While the output space is shared among all tasks, and no task identifier is given to the learner: for output distributions $p_1(y)$ and $p_2(y)$ with supports s_1^y and s_2^y respectively then $s_1^y = s_2^y$. This is in contrast to task incremental learning where a task identifier is available to the continual learning agent. This is illustrated in Figure 2.5.

Class incremental learning. In this scenario each new task is a new set of classes. The input domain expands as the number of tasks increases, also the number of classes seen also increases as the number of tasks increases. So the support of the output distributions is disjoint: for output distributions $p_1(y)$ and $p_2(y)$ with supports s_1^y and s_2^y respectively then $s_1^y \cap s_2^y = \emptyset$. Additionally, no task identifier is available to the agent. An example is illustrated in Fig. 2.5.

Task-agnostic versus task-aware. In all the above settings we traditionally evaluate methods in a *task-aware* setting. This is not the same as giving the continual learning model access to the task identifier, τ, but it is aware of when tasks change. The continual learning model is aware of task boundaries during training: it knows when a new task arrives. For example, regularization methods Kirkpatrick et al. (2016); Zenke et al. (2017a); Nguyen et al. (2017) (developed in Chapter 3) are able to update the loss function with an additional regularization term to enable retaining knowledge of the previous tasks. In contrast, the *task-agnostic* setting has no knowledge of task boundaries nor has access to a task identifer Zeno et al. (2018); Riemer et al. (2018); Buzzega et al. (2020). The supervised continual learning research in this thesis is all in the task-aware setting.

Multi-head versus single-head networks. A common design choice not exclusively used in continual learning is to have an output linear layer per task, or a linear head per task, h, map to outputs $h : \mathcal{Z} \to \mathcal{Y}$, where $y \in \mathcal{Y}$. Such that a continual learning agent uses a separate head per task $\{h_i\}_{i=1}^T$, these methods are called multi-headed while those that use one head are called single-headed. Note that the multi-head networks are compatible with *task-incremental* learning since they require knowledge of a task identifier, t, to select a new head during training and the head that corresponds to a specific task during evaluation. On the other hand, the single-headed network doesn't require knowledge of the task identifier during training and evaluation, so it is compatible with *domain-incremental* and *class-incremental* learning.

2.5.1.1 Evaluation strategies

Despite the continual learning model losing access to training data from past tasks. The continual learning model is evaluated on all tasks $\mathcal{T}_{<t}$ seen before training on the current task $\mathcal{T}_t$. Hence we define *normal evaluation* as simply iterating through each test set $\mathcal{D}_{<t}^{\text{test}}$ for the continual learner to be evaluated on past tasks according to suitable performance metrics Eq. (2.6.1.1) and Eq. (2.6.1.3). In normal evaluation, we can allow the continual learner access to a task identifier or not give it access to a task identifier.

In contrast *episodic evaluation* allows us to treat each test set $\mathcal{D}_i^{\text{test}}$ for $i < t$ as an episode. We can evaluate predictions or evaluate uncertainties to make a decision and infer which task test set is currently being evaluated. Episodic evaluation is useful when we do not have access to a task identifier. For example, by evaluating uncertainties for each linear head in multi-head network we can infer the task identifier τ and pick a suitable head to solve a particular test set Von Oswald et al. (2020). This is especially useful for the domain and class incremental setting where no task identifier is available during evaluation. Specifically, in Chapter 4 we evaluate the predictive entropy as a measure of uncertainty on the first batch over each head of a multi-head network and pick the head with the lowest predictive entropy to proceed with evaluating the rest of the test set Section 2.5.1.

See Table 2.1 for a summary of the different supervised continual learning evaluation strategies that are used throughout the thesis.

2.5.2 Continual Reinforcement Learning Experimental Scenarios

Task-incremental continual reinforcement learning. Similarly to supervised continual learning, task-incremental continual reinforcement learning is when the task identifier, τ, is supplied to the agent during training and evaluation. This is the most common form of continual reinforcement learning Kirkpatrick et al. (2016); Wolczyk et al. (2021); Caccia et al. (2022).

Table 2.1: Comparison of the continual learning evaluation strategies used throughout this thesis.

	Normal Eval.	**Episodic Eval.**
Task Incremental	• Chapter 4 • Chapter 5 • Chapter 8	
Domain Incremental	• Chapter 4 • Chapter 5	• Chapter 4
Class Incremental	• Chapter 4 • Chapter 5	• Chapter 4

Task-agnostic continual reinforcement learning. More autonomous agents have no knowledge of which task they are currently training on or which task they are being evaluated on Rolnick et al. (2019); Kessler et al. (2022a). This is the most autonomous form of continual reinforcement learning and the most difficult setup.

Task-agnostic at evaluation. There is an intermediate difficulty experimental setup, where an agent has knowledge of the task identifier t during training, however when evaluated on all the tasks and skills it has learned so far it doesn't have the supervision afforded by the task identifier, t. The agent is *task-agnostic at evaluation* and is required to infer the task is being evaluated on to solve it Kessler et al. (2022b). The task-agnostic at evaluation setting encapsulates the robotic home-assistant example Section 1.3.2.

2.5.3 Continual Reinforcement Learning Tasks

Conflicting tasks. In supervised continual learning we can have conflicting tasks if there are tasks where the domains are the same: consider two task input distributions $p_1(\boldsymbol{x})$ and $p_2(\boldsymbol{x})$ and their supports s_1^x and s_2^x are the same $s_1^x = s_2^x$. But the conditional distributions of the outputs are different $p_1(y|\boldsymbol{x}) \neq p_2(y|\boldsymbol{x})$. For example, the conditional distributions could be Gaussians with different means. In continual reinforcement learning, this problem can arise when goals change or the reward function changes between tasks. Specifically, in a task POMDP $\left(\mathcal{S}_\tau, \mathcal{A}_\tau, P_\tau, R_\tau, \Omega_\tau, \mathcal{O}_\tau, \gamma_\tau\right)$ the reward function R_τ changes but all other elements

of the POMDP tuple such as the action space $\mathcal{A}_\tau$ and the state-space $\mathcal{S}_\tau$ remain the same across tasks. This causes tasks to conflict and is explored in Chapter 6.

CRL tasks that are conflicting are compatible with all three CRL scenarios Section 2.5.2. In Chapter 6 we show the pitfalls of using task-agnostic CRL methods with conflicting tasks. Task-incremental CRL methods have knowledge of which task they are in via a task identifier τ. This can be used to avoid conflict during training and evaluation, for example by using the task identifier as an additional input to the policy or as an index to select a linear policy head in a multi-head network. For task-agnostic CRL at evaluation, the task identifier needs to be inferred at evaluation to avoid conflict. We introduce this scenario in Chapter 6.

2.6 Continual Learning Metrics

In this section, I define the different metrics used throughout the chapters of this thesis to measure the performance of a continual learner. The metrics are defined differently for continual supervised learning and continual reinforcement learning but are similar in spirit.

2.6.1 Supervised Continual Learning Metrics

I define the metrics that are used to measure the performance of supervised continual learning agents and which are used in the experimental setups in the main chapters of this thesis. These metrics are commonly used in the literature Lopez-Paz and Ranzato (2017); Chaudhry et al. (2018a).

Average Performance. Let $p_{k,j}$ be the performance of the continual learning agent, higher performance is better. For example, the accuracy on the test set j after training incrementally on tasks $1, \ldots, k$ and $j \leq k$. Then the average performance at task k is defined as:

$$P_k = \frac{1}{k} \sum_{i=1}^{k} p_{k,i}. \tag{2.6.1.1}$$

The higher the average performance P_k over all tasks $1, \ldots, k$ the better the continual learner is at learning all tasks $\mathcal{T}_{1:k}$ seen so far. For classification the performance

$p_{k,j}$ is the accuracy $a_{k,j}$ and for a canonical benchmark like Split CIFAR10 which has 5 tasks in total then we measure the performance at the end of training on the last task i.e. $k = 5$:

$$P_5 = \frac{1}{5}\sum_{i=1}^{5} a_{5,i} \tag{2.6.1.2}$$

which averages the performance over the test sets $i = 1, \ldots, 5$. This is the main metric that is used throughout the thesis as it simultaneously measures how well a continual learner is able to perform a specific task and how well it retains knowledge and prevents catastrophic forgetting.

Average Forgetting. This is defined as the difference between the performance after a task is trained and the current performance. This difference defines the performance gap and therefore how much the model has forgotten how to perform a task. The forgetting for the task j after learning k tasks $1, \ldots, k$ and $j < k$ is defined as:

$$f_j^k = p_{j,j} - p_{k,j}, \quad \forall j < k. \tag{2.6.1.3}$$

The average forgetting can only be defined for the previous $k-1$ tasks as:

$$F_k = \frac{1}{k-1}\sum_{j=1}^{k-1} f_j^k. \tag{2.6.1.4}$$

An F_k close to zero implies little forgetting. A negative forgetting implies that the performance improves throughout continual learning and the learner can transfer knowledge from future tasks when being evaluated on previous tasks. This is a very desirable and yet rare property of a continual learner. A positive F_k indicates forgetting, and the performance on task j degrades after learning new tasks $j+1$. A note of caution: we can get no forgetting while our learner has not learned anything at all: if $p_{k,j} = 0,\ \forall j$ implies no forgetting $F_k = 0$ which is also undesirable.

Performance upper bounds An upper bound to performance is how well the continual learning model can do in comparison to a multi-task model which can learn from all tasks at once. A different upper bound on performance is the single task performance; the performance of training a single different machine

learning model on each task, the individual models won't benefit from transfer available to the multi-task model Fig. 2.4.

Multi-task performance as an upper bound is natural for computer vision continual learning benchmarks like Split CIFAR10 Fig. 2.1 since the tasks are constructed by subsetting classes of the original vision dataset. In sequential Bayesian inference, the upper bound on performance is the multi-task posterior which the sequential posterior builds up through sequential Bayesian updates Eq. (2.2.0.5). As we discuss in Chapter 5 the Bayesian multi-task performance can be sub-optimal for the continual learning problem we want to solve and the continual learning solution can outperform the multi-task solution.

Metrics used in this thesis. Throughout this thesis the main metric that is used is the average performance as it simultaneously measures how well a continual learner can perform on a specific task after learning all tasks and forgetting. The higher the average performance the less forgetting. Comparing different average performance metrics can tell us which methods are better able to mitigate against forgetting relative to each other. We use these metrics in chapters Chapter 4 Chapter 5 Chapter 8.

2.6.2 Continual Reinforcement Learning Metrics

I describe in detail how to calculate the continual reinforcement learning metrics used extensively throughout this thesis.

Average Performance. This measures how well a CRL method performs on all tasks at the end of the task sequence. The task performance is $p_\tau(t) = [-1, 1]$ for all $\tau < T$. Since we have a reward of $+1$ for completing the task and -1 for being killed by a monster or falling into lava. If each task is seen for N environment steps and we have T tasks and the τ-th task is seen over the interval of steps $[(\tau - 1) \times N, \tau \times N]$. The average final performance metric for our continual learning agent is defined as:

$$p(t_f) = \frac{1}{T} \sum_{\tau=1}^{T} p_\tau(t_f), \tag{2.6.2.1}$$

where $t_f = N \times T$ is the final timestep.

Forgetting. The average forgetting is the performance difference after interacting with a task versus the performance at the end of the final task. The average forgetting across all tasks is defined as:

$$F = \frac{1}{T}\sum_{\tau=1}^{T} F_\tau \quad \text{where} \quad F_\tau = p_\tau(\tau \times N) - p_\tau(t_f). \tag{2.6.2.2}$$

The forgetting of the final T-th task is $F_T = 0$. If a CRL agent has better performance at the end of the task sequence compared to after τ-th task at time-step $\tau \times N$ then $F_\tau < 0$. Note that both the average performance and the forgetting metrics are functions of $p_\tau(t_f)$ so we expect anti-correlation between these two metrics.

Forward Transfer. The forward transfer is the difference in task performance during continual learning compared to the single task performance. The forward transfer is defined:

$$FT = \frac{1}{T}\sum_{\tau=1}^{T} FT_\tau \quad \text{where} \quad FT_\tau = \frac{\text{AUC}_\tau - \text{AUC}_{\text{ref}_\tau}}{1 - \text{AUC}_\tau}, \tag{2.6.2.3}$$

where AUC denotes the area under the curve and is defined as:

$$\text{AUC}_\tau = \frac{1}{N}\int_{(\tau-1)\times N}^{\tau \times N} p_\tau(t)dt \quad \text{and} \quad \text{AUC}_{\text{ref}_\tau} = \frac{1}{N}\int_{0}^{N} p_{\text{ref}_\tau}(t)dt. \tag{2.6.2.4}$$

$FT_\tau > 0$ means that the CRL agent achieves better performance on task τ during continual learning versus in isolation. So this metric measures how well a CRL agent transfers knowledge from previous tasks when learning a new task.

Performance upper bounds In contrast to supervised continual learning which uses the multi-task performance as an upper bound. In reinforcement learning, multi-task learning Hessel et al. (2019) is more involved and an active area of research. Standard RL algorithms such as PPO Schulman et al. (2017), DQN Mnih et al. (2015) or SAC Haarnoja et al. (2018a) are not designed for multi-task reinforcement learning. Additionally, multi-task RL is susceptible to conflicting gradients between tasks Yu et al. (2020b). It is more straightforward to use the single task performance as an upper bound, this is favoured in the literature and this thesis Wolczyk et al. (2021); Caccia et al. (2022); Kessler et al. (2022a).

Metrics used in this thesis. Throughout this thesis the main metric that is used is the average performance since similarly to the supervised counterpart it tells

us about the underlying performance of the continual learner in solving a particular task well and how well it can prevent forgetting when compared relatively to other methods Chapter 6. All CRL performance metrics are used in Chapter 7.

2.7 Organization of the Thesis

In Chapter 4 we introduce a novel method that leverages Bayesian CL to prevent forgetting. We develop a novel Bayesian NN which uses a non-parametric Indian Buffet Process prior (Griffiths and Ghahramani, 2011) which adds more neurons as more data is seen and so expands as more tasks are seen. We show that our Bayesian NN is sparse and can prevent any underfitting which can plague underspecified Bayesian NNs. In Chapter 5 we revisit Bayesian CL, in the literature, these methods employ tricks such as using a small coreset of data from previous tasks or a head per task (such as our method which we present in Chapter 4). However, performing sequential Bayesian inference using a Bayesian NN without any additional tricks should ensure that we obtain our multitask posterior Eq. (2.2.0.3) and so should obtain good performance and prevent forgetting in the continual learning problems considered. We aim to confirm this by performing sequential Bayesian inference using Hamiltonian Monte Carlo (HMC) which is the gold standard of Bayesian inference and is guaranteed to sample from the true posterior (Neal et al., 2011; Izmailov et al., 2021). We fit a density estimator over HMC samples so that we can use HMC samples as a prior for a new task for continual learning. We show that this approach fails to prevent forgetting since the density estimation doesn't model certain areas of the posterior probability distribution important for learning a new task, despite extensive efforts to ensure that we have an accurate, converged, and multi-modal posterior density estimate. Additionally, sequential Bayesian inference with a misspecified model and task data imbalances causes forgetting despite using sequential Bayesian inference. We thus recommend not performing Bayesian sequential inference over network parameters but instead modeling the data-generating process for CL as a more fruitful avenue.

Our work in CRL is introduced in Chapter 6. We show how in CRL we can have different tasks that share the same state and action-space but have different reward functions, $R_\tau(\cdot)$. This is important since state-of-the-art methods in CL and CRL use experience replay and can be used without giving the agent information as to which task it is in. Experience replay methods work by saving data from previous tasks in a buffer and replaying it to continual learning agents. If we have tasks that share the same state and action-spaces but different reward functions or goals this can cause *interference* of experience in the replay buffer. This interference can cause forgetting. In Chapter 6 we develop a simple solution that leverages existing methods like multiple heads to model different tasks and CL regularization methods to prevent forgetting across encoders shared across tasks. Our method called OWL is task-agnostic during evaluation, by using a multi-armed bandit to choose the optimal policy head which uses the Temporal-Difference error Eq. (2.3.0.1) as feedback. We find that using an ensemble (Lakshminarayanan et al., 2017a) of critics to estimate the log-likelihood of the Temporal-Difference (TD) error helps the bandit to find the correct policy. Having uncertainties in the TD error helped the multi-armed bandit find the correct policy to exploit to solve a task during evaluation. In Chapter 7 we explore CRL in the *model-based reinforcement learning* paradigm by leveraging powerful world models and we show that it is effective for a number of reasons. Since the world model is recurrent and is able to learn a good representation from trajectories of states, actions, and rewards. We do not see interference between tasks using experience replay. Firstly we can leverage experience replay buffers to prevent forgetting in the model. Since the policy is trained on generated experience inside the world model then it is sample efficient and the policy does not forget similarly to generative replay approaches to supervised CL (Shin et al., 2017). Finally, we can leverage task-agnostic exploration strategies by using the uncertainty in the world model's next-state predictions to enable continuous exploration in hard exploration and sparse-reward tasks.

In Chapter 8, we apply ideas from CL to enable rapid training of large self-supervised models to enable quicker experimentation. Self-supervised automatic

speech recognition (ASR) models perform pre-training using self-supervised masked speech modeling and then finetune on a small amount of labeled ASR data. If we have a pre-trained model in English then it makes sense to use this model for a new task in French or Spanish ASR to speed up training by transferring English speech representation. We develop a modular continual learning model based on `wav2vec2.0` (Baevski et al., 2020) which we name `continual-wav2vec2.0`. This enables good performance of new tasks without forgetting the original English ASR. Crucially, we are able to dramatically bring down pre-training times by around 40%, which represents a time saving measured in days.

3
Related Works

3.1 Supervised Continual Learning

Continual Learning (CL) methods can be categorized into three main types (Sodhani et al., 2022; Wang et al., 2023). Firstly, *regularization* methods, which aim to regularize the parameters learned for a new task so that they continue to perform well on previous tasks. Amongst the regularization methods, we can also include Bayesian methods where the prior acts as a regularizer. The second set of methods cover *replay* methods which aim to retain a sample of previous task datasets to ensure that continual training using this sample enables the model to remember solutions for previous tasks. Finally, we distinguish a third category of *expansion* methods for continual learning these add new neural resources, such as new network modules to perform new tasks whilst ensuring that the network doesn't forget previous task solutions by freezing weights (so preventing gradient-based learning on previously trained network modules). We note that these methods are not mutually exclusive and that they can, and have been combined (Yoon et al., 2017; Kessler et al., 2021a).

Regularization methods. The seminal Elastic Weight Consolidation (EWC) paper (Kirkpatrick et al., 2017), motivated by sequential Bayesian inference with a Laplace approximation (MacKay, 1992a) regularizes network parameters when learning a new task, such that they are close in terms of an L^2 norm, with respect

to previous task optimal parameters. In EWC the regularization is applied with an importance weighting for each parameter, which is proportional to the diagonal of the Fisher information matrix. Other methods also use an L^2 regularization around the previous task's optimal parameters but with different L^2 penalty weightings (Schwarz et al., 2018b; Zenke et al., 2017a). Instead of regularizing the weights of the network, we can apply a distillation loss to regularize learning a new function mapping while preserving previously learned function mappings from previous tasks (Li and Hoiem, 2017a; Benjamin et al., 2018; Rebuffi et al., 2017b). Active forgetting of previously learned parameters that conflict with the current task, has also been explored using an additional regularization alongside EWC (Wang et al., 2021). Graph-based Continual Learning embeds data as a random graph (an encoder first embeds images into a low dimensional space). This random graph is regularized using distillation such that there is no forgetting between task graphs (Tang and Matteson, 2020). It has been shown that by regularizing the loss function with a KL-divergence between a classifier and the uniform distribution ensures a flatter loss landscape. A flatter loss landscape ensures that there are more directions for the parameters to move in which enables learning multiple tasks (Cha et al., 2020).

Replay methods. Replay methods use a single predictor for all tasks and tackle forgetting by replaying data from past tasks while training on the current task. This is performed by storing a small memory of data from previous tasks (Lopez-Paz and Ranzato, 2017; Chaudhry et al., 2018b; Guo et al., 2020; Jin et al., 2021). Or using a generative model which is able to sample previous task data and augment the current task dataset for the learner to prevent forgetting past tasks (Shin et al., 2017). Memory methods can also be used by replaying network latent states so that they can be more efficiently stored (van de Ven et al., 2020; Ayub and Wagner, 2021). Selecting which points are important to replay has also been shown to improve performance for replay-based methods (Caccia et al., 2020; Liu et al., 2021). One can ensure that new tasks are learned in orthogonal subspaces to previous tasks. Hence gradient updates minimally influence previous tasks. The key idea is that in high dimensional parameter spaces there are directions that cause large

changes to predictions and others that do not. In particular, moving locally in the direction $\pm\nabla_{\boldsymbol{\theta}} f(x;\boldsymbol{\theta})$ (where $f(\cdot;\boldsymbol{\theta}) : \mathcal{X} \to \mathcal{Y}$ is a network prediction and $\boldsymbol{\theta}$ are the network parameters) leads to the biggest change in the prediction, and moving in the direction orthogonal to $\nabla_{\boldsymbol{\theta}} f(x;\boldsymbol{\theta})$ leads to the least change. So to prevent forgetting one can perform gradient descent orthogonal to the gradients from previous tasks which requires a memory of previous task gradients (Farajtabar et al., 2020). This same idea can be performed, but more efficiently than storing previous task gradients (Chaudhry et al., 2020; Deng et al., 2021; Saha et al., 2021).

Expansion methods. The third category of CL methods we consider is that of expansion methods which add new parameters to a model when learning a new task. In ProgressiveNets a new network predictor is added for each new task, with connections between adjacent layers (Rusu et al., 2016a). This means that the number of network parameters grows super-linearly as the number of tasks increases. Dynamically Expanding Networks uses a sparsity regularization to dynamically select and limit the number of new parameters to add for each new task (Yoon et al., 2017). New modules can be added by casting the expansion process as a reinforcement learning problem (Xu and Zhu, 2018), though this is very expensive to train. Evolution methods and random search have also been applied to select the optimal sub-network in a large NN for each task (Fernando et al., 2017; Rajasegaran et al., 2019). Similarly one can use hypernetworks to generate weights for a task, one is limited by the hypernetworks capacity to generate multiple NN weights and a task embedding used to condition the hypernetwork (Ehret et al., 2020). By finding the task subnetwork that most resembles the previous task and adding modules to this subnetwork, one can then use an exhaustive search to find the subnetwork that performs best on a new task while freezing previous task weights (Veniat et al., 2020). One can learn important visual features in a first task and then freeze and calibrate a CNN's first task features for learning new tasks with new adaptation weights (Singh et al., 2020). Mixture-of-experts approaches which are modeled as an online Dirichlet process, have also been shown to be effective for CL (Lee et al., 2020c). Neural modules can be composed to solve new CL tasks (Ostapenko

et al., 2021), and neural architecture search has also been suggested as a method for CL (Chen et al., 2020). Parameter superposition has also been suggested, in which linear layers project inputs into mutually orthogonal subspaces, for example, rotational superpositions, (Cheung et al., 2019).

Few-shot learning and continual learning. Continual learning agents can use the few-shot learning framework of model-agnostic meta-learning (Finn et al., 2017) to perform few-shot learning of continual learning problems which have not been seen during training (Javed and White, 2019; Banayeeanzade et al., 2021), this is often referred to as *few-shot continual learning*. On the other hand, we can continue to obtain data as part of the continual learning framework and sequentially build up knowledge in the meta-parameters, this is often referred to as *continual few-shot learning* (He et al., 2019; Caccia et al., 2020). One can perform sequential Bayesian inference over meta-parameters using a Laplace approximation, or variational inference, to perform few-shot continual learning (Zhang et al., 2021a; Yap et al., 2021). Encouraging flat-minima around the meta-parameters has also been shown to improve few-shot continual learning (Shi et al., 2021).

Empirical studies. Controlling the largest eigenvalue of the loss; so promoting a flatter loss landscape, can also decrease forgetting. This can come about with different training hyperparameter choices, for instance, dropout regularization, learning rate schedules, and changing batch sizes (Mirzadeh et al., 2020b). Areas of low curvature in the loss landscape of neural networks can hold around 20 tasks, and linear mode connectivity between CL and multi-task solutions can also be leveraged for good continual learning performance (Mirzadeh et al., 2020a). Neural network architecture choice also affects continual learning performance: wide networks forget less through gradient sparsity (Mirzadeh et al., 2022). Task similarity is an important factor that affects forgetting (Bell and Lawrence, 2021; Ramasesh et al., 2020) and so is order (Bell and Lawrence, 2022). One can also leverage self-supervised pre-training to help CL performance (Gallardo et al., 2021).

Theory. Recent theory states that optimal CL requires perfect memory (Knoblauch et al., 2020). There has also been research unifying various regularization meth-

ods (Benzing, 2020). The continual learning process has been modeled as a two-player game where there is a trade-off between generalization and catastrophic forgetting (Raghavan and Balaprakash, 2021). Continual learning theory has also been studied using overparameterized linear models (Lin et al., 2023).

Datasets. Permuted-MNIST is a canonical CL baseline that involves applying a fixed permutation to the pixels in MNIST for each new task (Kirkpatrick et al., 2017). Other datasets such as Split-MNIST and Split-CIFAR10 involve taking these common vision benchmarks and chunking them into distinct binary tasks. Large vision classification problems such as Split-CIFAR100 and Split-MiniImageNet are chunked into 10-way classification problems for instance. Continual ego-centric object recognition can be evaluated with the CORe50 dataset (Lomonaco and Maltoni, 2017). The CLAD benchmark is a continual learning driverless car object detection CL benchmark where all state-of-the-art entries use experience replay (Verwimp et al., 2022).

3.1.1 Bayesian Continual Learning

Methods that approximate sequential Bayesian inference Eq. (2.2.0.3) have been seminal in CL's revival and have used a diagonal Laplace approximation (Kirkpatrick et al., 2017; Schwarz et al., 2018b). The diagonal Laplace approximation has been enhanced by modeling covariances between neural network weights in the same layer (Ritter et al., 2018). Online Laplace for CL can be augmented with an additional gradient projection which is the inverse Hessian of the posterior from all previous tasks. This preconditioning enables the gradient for the current task to take a direction that doesn't affect the posterior of previous tasks thereby preventing forgetting (Kao et al., 2021). Instead of the Laplace approximation, we can use a variational approximation for sequential Bayesian inference (Nguyen et al., 2017; Zeno et al., 2018). Using richer priors has also been explored (Ahn et al., 2019; Farquhar et al., 2020; Kessler et al., 2021a; Mehta et al., 2021; Kumar et al., 2021; Loo et al., 2020). Gaussian processes have also been applied to CL problems leveraging inducing points to retain previous task functions (Titsias et al.,

2020b; Kapoor et al., 2021). Bayesian NN weights can also be generated by a hypernetwork, where the hypernetwork needs only simple CL techniques to prevent forgetting (Henning et al., 2021).

Instead of regularizing high-dimensional weight spaces, regularizing task functions is a more direct approach to combat forgetting (Benjamin et al., 2018). In particular, one can leverage the duality between the Laplace approximation and Gaussian Processes to develop a functional regularization approach to Bayesian CL (Swaroop et al., 2019) or using function-space variational inference (Rudner et al., 2022b). Variational inducing point coresets can be used to summarize data from previous tasks for CL (Manousakas et al., 2022).

3.2 Continual Reinforcement Learning

Seminal work in CRL, EWC (Kirkpatrick et al., 2016) enables a Deep Q-network (Mnih et al., 2015) to be able to continually learn how to play different Atari games with limited forgetting. EWC learns new Q-functions by regularizing the L^2 distance between the new task's optimal weights and the previous task's optimal weights. EWC requires additional supervision informing it of task changes to update its objective, select a specific Q-function head, and select a task-specific ϵ-greedy exploration schedule. Progress and Compress (Schwarz et al., 2018a) applies a regularization to policy and value function feature extractors for an actor-critic approach. Alternatively, LPG-FTW (Mendez et al., 2020) learns an actor-critic that factorizes into task-specific parameters and shared parameters. Both methods require task supervision and make use of task-specific parameters and shared parameters. Task-agnostic methods like CLEAR (Rolnick et al., 2019) do not require task information to perform CRL. CLEAR leverages experience replay buffers (Lin, 1992) to prevent forgetting: by using an actor-critic with V-trace importance sampling (Espeholt et al., 2018) of past experiences from the replay buffer. Model-based RL approaches to CRL have been demonstrated where the model weights are generated from a hypernetwork which itself is conditioned by a task embedding (Huang et al., 2021). Recent work demonstrates that recurrent

policies for POMDPs can obtain good overall performance on continuous control CRL benchmarks (Caccia et al., 2022). Another task-aware solution is to expand a subspace of policies, the number of policies scaling sublinearly with the number of tasks (Gaya et al., 2022). Compositional approaches to CRL have also been used where modules are combined to learn new tasks (Mendez et al., 2022).

A number of previous works have studied transfer in multi-task RL settings where the goals within an environment change (Barreto et al., 2017; Schaul et al., 2015; Barreto et al., 2019). In particular, incorporating the task definition directly into the value function (Schaul et al., 2015) and combining this with off-policy learning allows a CRL agent to solve multiple tasks continually and generalize to new goals (Mankowitz et al., 2018).

Hierarchical RL (Sutton et al., 1999; Bacon et al., 2017) is where a policy controls sub-policies that are temporally extended over multiple sequential transitions. Extensions of hierarchical RL applied to CRL, where sub-policies are continually learned and a master policy adapts to incorporate new sub-policies, have been performed using the graph Laplacian constructed from state, action, and reward samples which are decomposed into eigenvectors called eigen-options in a nod to the RL option literature. The eigen-options can be quickly reused for learning new tasks when learning a new navigational task with a different goal location versus learning from scratch (Klissarov and Machado, 2023).

Environments. Sequences of Atari tasks have been used for CRL (Kirkpatrick et al., 2016). Also, different 2-d mazes have been explored for CRL (Chevalier-Boisvert et al., 2018; Kessler et al., 2022b), as have different 2-d games from Nethack (Küttler et al., 2020; Samvelyan et al., 2021). Different maze configurations, with different textures on walls and objects, have been used for CRL (Lomonaco et al., 2020). The CRL benchmark, ContinualWorld, involves 10 sequential continuous control robotic arm tasks using proprioceptive states (Wolczyk et al., 2021). The CORA suite of tasks (Powers et al., 2021) also explores different suites of tasks including Atari and Procgen (Cobbe et al., 2020).

Continual Adaptation. Instead of focusing on remembering how to perform all past tasks, another line of research investigates quick adaptation to changes in the environment. This can be captured by using a latent variable and off-policy RL (Xie et al., 2020). Alternatively, one can meta-learn a model such that it can then adapt quickly to new changes in the environment (Nagabandi et al., 2018). All these works use small environment changes such as reward function changes or changes in gravity or mass of certain agent limbs for instance. Continual exploration strategies that use curiosity (Pathak et al., 2017) can be added as an intrinsic reward in the face of non-stationary environments in infinite horizon MDPs (Steinparz et al., 2022). One can use Plan2Explore (Sekar et al., 2020) for continual exploration of new tasks in CRL which has been shown to outperform curiosity-based methods (Kessler et al., 2022a). The tasks themselves can be meta-learned using a latent variable world model and task similarities can be exploited when learning a new task (Fu et al., 2022). Continual meta-policy search uses meta-RL: the inner loop task adaptation is performed using PPO and the meta-learning is performed using behavioural cloning on a replay buffer of all previous task experience treating them as expert trajectories (Berseth et al., 2021).

Curriculum Learning. Another related area of research is open-ended learning which aims to build agents that generalize to unseen environments through a curriculum that starts off with easy tasks and then progresses to harder tasks thereby creating agents that can generalize and solve hard tasks (Wang et al., 2019; Team et al., 2021; Parker-Holder et al., 2022).

4

Hierarchical Indian Buffet Neural Networks for Bayesian Continual Learning

On the topic of sequential Bayesian inference for continual learning. We enable a Bayesian neural network to expand as more tasks are seen in continual learning. Thereby mitigating any underfitting which can affect Bayesian neural networks with too few parameters. We place an Indian Buffet process prior over the structure of a Bayesian neural network. The Indian Buffet Process prior is formulated for matrix factorization to select factors, but in this work, we repurpose it to select the neurons of a Bayesian neural network. We further extend this model such that the prior on the structure of each hidden layer is shared globally across all layers, using a Hierarchical Indian Buffet process. We apply this model to the problem of resource allocation in continual learning where new tasks occur and the network requires extra resources. Our model uses sequential Bayesian inference to recursively build up the posterior and remember how to perform all past tasks. In the next chapter Chapter 5, we revisit this assumption and show that sequential Bayesian inference over the weights of a Bayesian neural network is a poor way to retina previous task knowledge. For the Indian Buffet Process prior inference we use online variational inference with reparameterization of the Bernoulli and Beta distributions. As we automatically learn the number of weights in each layer

of the Bayesian neural network, overfitting and underfitting problems are largely overcome. We show empirically that our approach offers a competitive edge over existing methods in CL. We revisit some of the Bayesian continual learning design choices in this chapter in the following chapter Chapter 5.

4.1 Introduction

Humans have the ability to continually learn, consolidate their knowledge and leverage previous experiences when learning a new set of skills. In Continual Learning (CL) an agent must also learn continually, presenting several challenges including learning online, avoiding forgetting, and efficiently allocating resources for learning new tasks. In CL, a neural network model is required to learn a series of tasks, one by one, and remember how to perform each. The model is given a set of M tasks sequentially $\mathcal{T}_t$ for $t = 1, ... M$. Where each task is comprised of a dataset.

The model loses access to the training dataset for task $\mathcal{T}_t$ but will be continually evaluated on the test sets for all previous tasks $\mathcal{T}_i$ for $i \leq t$, we introduce the problem setting more formally in the next section.

The principal challenges to CL are threefold, firstly models need to overcome *catastrophic forgetting* of old tasks; a neural network exhibits forgetting of previous tasks after having learned a few tasks (Goodfellow et al., 2013). Secondly, models need to leverage knowledge transfer from previously learned tasks for learning a new task $\mathcal{T}_t$. Finally, the model needs to have enough neural resources available to learn a new task and adapt to the complexity of the task at hand.

One of the main approaches to CL involves the use of the natural sequential learning approach embedded within Bayesian inference. The prior for task $\mathcal{T}_i$ is the posterior which is obtained from the previous task $\mathcal{T}_{i-1}$. This enables knowledge transfer and offers an approach to overcome catastrophic forgetting. Previous Bayesian CL approaches have leveraged Laplace approximations (Kirkpatrick et al., 2017) and variational inference (Nguyen et al., 2017) to aid computational tractability. Whilst Bayesian methods solve the first and second objectives above, the third objective of ensuring that the BNN has enough neural resources to

adapt its complexity to the task at hand is not necessarily achieved. For instance, additional neural resources can alter performance on MNIST classification (see Table 1 in (Blundell et al., 2015a)). This is a problem as the amount of neural resources required for a current task, may not be enough (or may be redundant) for a future task. Propagating a poor approximate posterior from one task will alter performance for all subsequent tasks.

Non-Bayesian neural networks use additional neurons to learn new tasks and prevent overwriting previous knowledge thus overcoming forgetting. The neural networks that have been trained on previous tasks are frozen and a new neural network is appended to the existing network for learning a new task (Rusu et al., 2016a). The problem with this approach is that of scalability: the number of neural resources increases linearly with the number of tasks. The scalability issue has been tackled with selective retraining and expansion with a group regulariser (Yoon et al., 2017). However, this solution is unable to shrink and so is vulnerable to overfitting if misspecified when starting CL. Moreover, knowledge transfer and prevention of catastrophic forgetting are not solved in a principled manner, unlike approaches couched in a Bayesian framework.

As the resources required are typically unknown in advance, we propose a BNN that adds or withdraws neural resources automatically in response to the data. This is achieved by drawing on Bayesian non-parametrics to learn the structure of each hidden layer of a BNN. Thus, the model size adapts to the amount of data seen and the difficulty of the task. This is achieved by using a binary latent matrix $\mathbf{Z}$, distributed according to an Indian Buffet Process (IBP) prior (Griffiths and Ghahramani, 2011). The IBP prior on an infinite binary matrix, Z, allows inference on which and how many neurons are required for each data point in a task. The weights of the BNN are treated as draws from non-interacting Gaussians (Blundell et al., 2015a). Catastrophic forgetting is overcome by repeated application of the Bayesian update rule, embedded within variational inference (Nguyen et al., 2017). We summarise the contributions as follows. We present a novel BNN using an IBP prior and its hierarchical extension to automatically learn the complexity of each

hidden layer according to the task difficulty. The model's effective use of resources is shown to be useful in CL. We derive a variational inference algorithm for learning the posterior distribution of the proposed models. In addition, our model elegantly bridges two separate CL approaches: expansion methods and Bayesian methods (more commonly referred to as regularization-based methods in CL literature).

4.2 Indian Buffet Neural Networks

We introduce the CL problem setting in Section 4.2.1, variational Bayesian approaches to CL in Section 4.2.2. We present the IBP prior in Section 4.2.3 and the IBP prior on the latent binary matrix Z is then applied to a BNN such that the complexity of each hidden layer can be learned from the data in Section 4.2.4. In Section 4.3, the Hierarchical IBP prior (H-IBP) is introduced and applied to the BNN to encourage a more regular structure. Thus, the use of an IBP and H-IBP prior over the hidden states of the BNN can be readily used together with the Bayesian CL framework presented, and so automatically adapt its complexity according to the task.

4.2.1 Continual Learning

Continual learning (CL) is a setting whereby a model must learn a set of tasks sequentially while maintaining performance across all tasks. In CL, the model is shown a set of M tasks sequentially $\mathcal{T}_t$ for $t = 1, \dots M$. Each task is comprised of a dataset such that $\mathcal{T}_t : \mathcal{D}_t = \{(\boldsymbol{x}_i, y_i)\}$ for $i = 1, \dots, N_t$. The inputs $\boldsymbol{x}_i \in \mathbb{R}^d$ and outputs can be $y_i \in \mathbb{R}$ in the case of regression or a categorical variable for classification. Although the model loses access to the training dataset for task $\mathcal{T}_t$, it is evaluated on all previous tasks $\mathcal{T}_i$ for $i \leq t$. t can be used as a task identifier informing the agent when to start training on a new task or what task to test. For a comprehensive review of CL scenarios see (van de Ven and Tolias, 2018; Hsu et al., 2018b).

4.2.2 Bayesian Continual Learning

The CL process can be decomposed into Bayesian updates where the approximate posterior for $\mathcal{T}_{t-1}$ can be used as a prior for task $\mathcal{T}_t$. Variational CL (VCL) (Nguyen et al., 2017) uses a BNN to perform the prediction tasks where the network weights are independent Gaussians. The variational posterior from previous tasks is used as a prior for new tasks. Consider learning the first task $\mathcal{T}_1$, and $\boldsymbol{\phi}$ are the variational random variables, then the variational posterior is $q_1(\boldsymbol{\phi}|\mathcal{D}_1)$. For the subsequent task, access to $\mathcal{D}_1$ is lost and the prior is $q_1(\boldsymbol{\phi}|\mathcal{D}_1)$, optimization of the ELBO yields the variational posterior $q_2(\boldsymbol{\phi}|\mathcal{D}_2)$. Generalizing, the negative ELBO for the t-th task is:

$$\mathcal{L}(\boldsymbol{\phi}, \mathcal{D}_t) = D_{\mathrm{KL}}\left[q_t(\boldsymbol{\phi})||q_{t-1}(\boldsymbol{\phi}|\mathcal{D}_{t-1})\right] - \mathbb{E}_{q_t}[\log p(\mathcal{D}_t|\boldsymbol{\phi})]. \tag{4.2.2.1}$$

The first term acts to regularise the posterior such that it is close to the previous task's posterior and the second term is the log-likelihood of the data for the current task.

4.2.3 Indian Buffet Process prior

Matrix decomposition aims to represent the data X as a combination of latent features: $X = ZA + \epsilon$ where $X \in \mathbb{R}^{N\times D}$, $\mathbb{Z} \in \mathbb{Z}_2^{N\times K}$, $A \in \mathbb{R}^{K\times D}$ and ϵ is observation noise. Each element in Z corresponds to the presence or absence of a latent feature from A. Specifically, $z_{ik} = 1$ corresponds to the presence of a latent feature A_k in observation X_i and $k \in \{1, \ldots, \infty\}$ all columns in Z with $k > K$ are assumed to be zero. In a scenario where the number of latent features K is to be inferred, then the IBP prior on Z is suitable (Doshi-Velez et al., 2009).[1]

One representation of the IBP prior is the stick-breaking formulation (Teh et al., 2007). The probability π_k is assigned to the column $\boldsymbol{z}_k$ for $k \in \{1, \ldots, \infty\}$, whether a feature has been selected is determined by $z_{nk} \sim \mathrm{Bern}(\pi_k)$. This parameter π_k is generated according to the following stick-breaking process: $v_k \sim \mathrm{Beta}(\alpha, 1)$,

[1] We provide a notebook to demonstrate how the IBP prior can be used for the matrix factorization `https://bit.ly/3asylU5`. In particular, we illustrate how one doesn't need to specify the number of latent dictionary items to infer. This means we do not need to set the hidden state size of a BNN for our model.

and $\pi_k = \prod_{i=1}^{k} v_i$, thus π_k decreases exponentially with k. The Beta concentration parameter α controls how many features one expects to see in the data, the larger α is, the more latent features are present.

4.2.4 Adaptation with the IBP prior

Instead of allowing the IBP prior to select factors for matrix factorization, we apply the IBP prior for neuron selection in a Bayesian neural network. Consider a BNN with k_j neurons for each layer $j \in \{1, \ldots, J\}$ layers. Thence, for an arbitrary activation f, the binary matrix Z is applied elementwise $h_j = f(h_{j-1}W_j) \circ Z_j$ where $h_{j-1} \in \mathbb{R}^{N \times k_{j-1}}$, $W_j \in \mathbb{R}^{k_{j-1} \times k_j}$, $Z_j \in \mathbb{Z}_2^{N \times k_j}$, and where $\circ$ is the elementwise product and N is the number of data points per batch. We have ignored biases for simplicity. Z_j is distributed according to an IBP prior, see Fig. 4.1 for the graphical model of the IBP. The IBP prior has some suitable properties for this application: the number of neurons sampled grows with N and the promotion of a "rich get richer" scheme for neuron selection (Griffiths and Ghahramani, 2011). For convenience, we term the IBP BNN as IBNN for the remainder of this chapter.

The number of neurons selected grows or contracts according to the variational objective; which depends on the complexity of the data. This allows for the efficient use of neural resources which is crucial to a successful CL model. The variational objectives for the IBP prior and BNN are introduced further down the line in Section 4.2.5 and Section 4.3.2. Additionally, the "rich get richer" scheme is useful since the common neurons are selected across tasks enabling knowledge transfer and preventing forgetting.

As a standard practice in variational inference with a Bayesian nonparametric prior, we use a truncation level K, to the maximum number of features in the variational IBP posterior. Doshi et al. (2009) present bounds on the marginal distribution of X in a matrix factorization setting and show that the bound decreases exponentially as K increases. Similar behaviour is expected for our application.

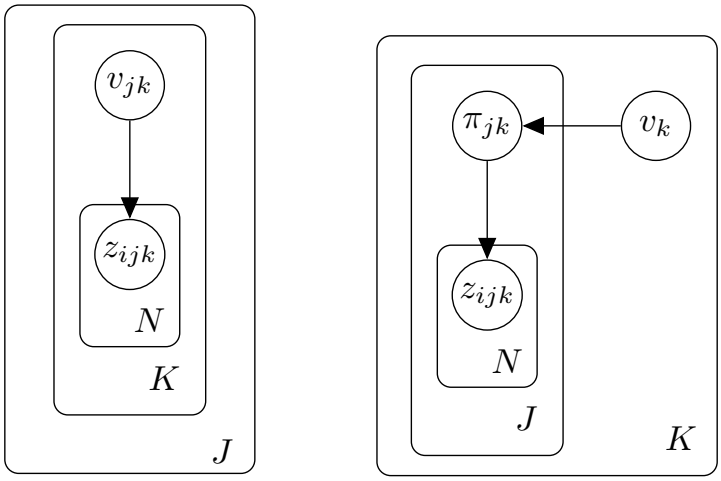


Figure 4.1: The graphical model for the structured variational posterior approximation for **Left**, the IBP and **Right**, the H-IBP^2. Using the language of the eponymous IBP prior metaphor, k (dishes) indicates the neurons, and the number selected K adjusts flexibly. j (restaurants) is the number of layers which is fixed, for the IBNN a separate IBP is applied for each layer. For the HIBNN the probabilities are tied using the H-IBP. i (customers) is a data point.

4.2.5 Structured VI

Structured stochastic VI (SSVI) has been shown to perform better inference of the IBP posterior than mean-field VI in deep latent variable models (Singh et al., 2017). Hence, this inference method has been chosen for learning and presented next.

A separate binary matrix Z_j can be applied to each layer $j \in \{1, \ldots, J\}$ of a BNN. The subscript j is dropped for clarity. The structured variational approximation is: $q(\boldsymbol{\phi}) = \prod_{k=1}^{K} q(v_k) q(\boldsymbol{w}_k) \prod_{i=1}^{N} q(z_{ik}|v_k)$, where the random variables are $\boldsymbol{\phi} = \{v_k, \boldsymbol{w}_k, Z_k\}$ and the variational parameters $\boldsymbol{\varphi} = \{\alpha_k, \beta_k, \boldsymbol{\mu}_k, \boldsymbol{\sigma}_k\}$ for all k. The variational distributions over $\boldsymbol{\phi}$ are defined below and the variational IBP posterior is truncated to K. The constituent distributions are $q(v_k) = \text{Beta}(\alpha_k, \beta_k)$, $\pi_k = \prod_{i=1}^{k} v_i$, $q(z_{ik}) = \text{Bern}(\pi_k)$ and the BNN weights are independent draws from $q(\boldsymbol{w}_k) = \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\sigma}_k^2 \mathbb{1})$. Having defined the structured variational objective, the negative ELBO is:

$$\begin{aligned}\mathcal{L}(\phi, \mathcal{D}) = & D_{\mathrm{KL}}(q(\boldsymbol{v})||p(\boldsymbol{v})) + D_{\mathrm{KL}}(q(\boldsymbol{w})||p(\boldsymbol{w})) \\ & - \sum_{i=1}^{N} \mathbb{E}_{q(\phi)} \left[\log p(y_i|\mathbf{x}_i, \mathbf{z}_i, \boldsymbol{w})\right] \\ & + \sum_{i=1}^{N} D_{\mathrm{KL}}(q(\mathbf{z}_i|\boldsymbol{\pi})||p(\mathbf{z}_i|\boldsymbol{\pi})).\end{aligned} \tag{4.2.5.1}$$

We note that the Bernoulli random variables have an explicit dependence on the stick-breaking probabilities in the structured variational approximation. However, this explicit dependence between parameters is removed in a mean-field approximation. The variational parameter Beta parameters control the expected hidden state size for the BNN and are automatically inferred. For sequential Bayesian updates, the posterior is then used as the next task's prior, thus a prior only needs to be designed for the first task.

4.3 Hierarchical IBNN

In the previous section, we presented the IBNN model to allow a BNN to automatically select the number of neurons for each layer according to the data. For a multi-layer BNN, one can apply the IBP prior independently for each layer. However, we can add an inductive bias to ensure that the inferred number of neurons is similar across all layers of an MLP and ensure that information is shared across layers. To do this, we propose a hierarchical IBP prior (Thibaux and Jordan, 2007) for neuron selection across multiple layers. The number of neurons from all layers is generated from the same global prior, thus discouraging irregular structure in the BNN (a BNN with adjacent wide and narrow layers might be inferred when using independent priors on each hidden layer). Of course, this property might not be desirable for all use cases, however, the majority of BNNs used in the literature have a regular structure. We term our model as HIBNN and present the graphical model that describes the hierarchical IBP prior in Fig. 4.1. This model has the advantage of having fewer learnable parameters as neuron selection is driven by the global prior in comparison to having a single IBP per layer. See Fig. 4.1 for a graphical model of the H-IBP.

4.3.1 Adaptation with the Hierarchical IBP

The global probability of selecting the neuron positioned at the k-th index across all layers is defined according to a stick-breaking process:

$$\pi_k^0 = \prod_{i=1}^{k} v_i, \quad v_k \sim \text{Beta}(\alpha, 1), \quad k = 1, \cdots \infty. \tag{4.3.1.1}$$

Child IBPs are defined over the structure of each individual hidden layer of a BNN which depend on π_k^0 to define the respective Bernoulli probabilities of selecting neuron k in layer j:

$$\pi_{jk} \sim \text{Beta}\left(\alpha_j \pi_k^0, \alpha_j(1 - \pi_k^0)\right), \tag{4.3.1.2}$$

for $j \in \{1, \ldots, J\}$, $k \in \{1, \ldots, \infty\}$, where J is the number of layers in a BNN, α_j are hyperparameters (Thibaux and Jordan, 2007; Gupta et al., 2012). The selection of the k-th neuron in the j-th layer by a particular data point i in the dataset of size N is thus:

$$z_{ijk} \sim \text{Bern}(\pi_{jk}), \quad i = 1, \ldots, N. \tag{4.3.1.3}$$

Notice that if k is small, π_k^0 is close to 1 then the shape parameter of the child Beta distribution will be large. At the same time, the scale parameter will be small. So the Bernoulli probability in Eq. (4.3.1.3) will be close to 1, as k increases π_k^0 and π_{jk} decrease. To infer the posterior $p(\boldsymbol{v}, Z, \boldsymbol{w}|\mathcal{D})$, we perform SSVI.

4.3.2 Structured VI

A structured variational posterior distribution that retains properties of the true posterior is desired such that the global stick-breaking probabilities influence child stick-breaking probabilities of each layer of the BNN. Let us define the variational distributions for our hidden variables as follows, $q(v_k^0) = \text{Beta}(\alpha_k^0, \beta_k^0)$ and $q(\pi_{jk}) = \text{Beta}(\alpha_j \pi_k^0, \alpha_j(1 - \pi_k^0))$, $\pi_k^0 = \prod_{i=1}^{k} v_i^0$, $q(z_{ijk}) = \text{Bern}(\pi_{jk})$ and the weights of the BNN are drawn from $q(\boldsymbol{w}_{jk}) = \mathcal{N}(\boldsymbol{\mu}_{jk}, \boldsymbol{\sigma}_{jk}^2 \mathbb{1})$.

[2]The mean-field approximation removes all edges from these graphical models.

The structured variational distribution is defined as follows

$$q(\boldsymbol{\phi}) = \prod_{k=1}^{K} q(v_k^0) \prod_{j=1}^{J} q(\pi_{jk}|v_k^0) q(\boldsymbol{w}_{jk}) \prod_{i=1}^{N} q(z_{ijk}|\pi_{jk}) \tag{4.3.2.1}$$

where $\boldsymbol{\varphi} = \{\alpha_k^0, \beta_k^0, \boldsymbol{\mu}_{jk}, \boldsymbol{\sigma}_{jk}\}$ for all j and k, up to the variational truncation, K. Having defined the structured variational distribution, the negative ELBO is:

$$\begin{aligned}\mathcal{L}(\boldsymbol{\phi}, \mathcal{D}) = & \mathrm{KL}(q(\boldsymbol{v}^0)||p(\boldsymbol{v}^0)) + \sum_{j=1}^{J} \mathrm{KL}(q(\boldsymbol{\pi}_j|\boldsymbol{v}^0)||p(\boldsymbol{\pi}_j|\boldsymbol{v}^0)) \\ & + \sum_{j=1}^{J} \mathrm{KL}(q(\boldsymbol{w}_j)||p(\boldsymbol{w}_j)) \\ & - \sum_{i=1}^{N} \mathbb{E}_{q(\phi)}[\log p(y_i|\mathbf{x}_i, \mathbf{z}_i, \boldsymbol{w})] \\ & + \sum_{j=1}^{J} \sum_{i=1}^{N} \mathrm{KL}(q(\mathbf{z}_{ij}|\boldsymbol{\pi}_j)||p(\mathbf{z}_{ij}|\boldsymbol{\pi}_j)).\end{aligned} \tag{4.3.2.2}$$

The child stick-breaking variational parameters for each layer are conditioned on the global stick-breaking parameters and the binary masks z_{ijk} for each neuron k in each layer j are conditioned on the child stick-breaking variational parameters. Thus, the variational structured posterior is able to capture dependencies of the prior. The learnable parameters are α_k^0, β_k^0, $\boldsymbol{\mu}_{jk}$ and $\boldsymbol{\sigma}_{jk}$ for all k neurons and for all layers j.

Inference. The variational posterior is obtained by optimising Eq. (4.3.2.2) using structured stochastic VI. For inference to be tractable, we utilize three reparameterizations. The first is for the Gaussian weights (Kingma and Welling, 2014). The second is an implicit reparameterisation of the Beta distribution (Figurnov et al., 2018). The third reparameterization uses a Concrete relaxation to the Bernoulli distribution (Maddison et al., 2016; Jang et al., 2016). Details of these are in the Supplementary material, Section 10.1.1.

4.4 Related Works

IBP priors and model selection in deep learning. An IBP prior has been used in VAEs to automatically learn the number of latent features. Stick-breaking probabilities have been placed directly as the VAE latent state (Nalisnick and Smyth, 2017). The IBP prior has been used to learn the number of features in a VAE hidden

state using mean-field VI (Chatzis, 2018) with black-box VI (Ranganath et al., 2014) and structured VI (Hoffman and Blei, 2015; Singh et al., 2017). As an alternative to truncation, Xu et al. (2019) uses a Russian roulette sampling scheme to sample from the infinite sum in the stick-breaking process for the IBP. Model selection for BNNs has been performed with the Horse-shoe prior over weights (Ghosh et al., 2019). The IBP prior has been employed in BNNs to induce sparsity (Panousis et al., 2019) and simultaneously to our work for CL (Kumar et al., 2019). Both of these approaches apply the IBP prior differently to our work and previous work applying the IBP to VAEs. Also, Kumar et al. (2019) deviates from sequential Bayes by storing masks over NN parameters for each task and uses design choices which means the IBP is not the sole means of selecting weights, we expand on this in Section 10.1.7. Instead of using an IBP prior, Bernoulli distributions (or its Concrete relaxation) are used to select the width and number of layers in a BNN (Dikov et al., 2019). This approach is not non-parametric and so not as desirable for CL. Recently and subsequently to our work, (Mehta et al., 2020) have proposed a CL approach where weights matrices are factorized and the IBP prior applied to the diagonal in the factorization. This work does not use sequential Bayes for CL but rather different neural network weights are used for different tasks and thus alleviate forgetting.

Bayesian continual learning. Repeated application of Bayes' rule can be used to update a model given the arrival of a new task. Previous work has used Laplace approximations (Kirkpatrick et al., 2017) and variational inference (Nguyen et al., 2017). Bayesian methods can also be intuitively thought of as a weight space regularisation. Explicit regularisation in weight space has also proved successful in CL (Zenke et al., 2017a; Schwarz et al., 2018b). Our method builds upon (Nguyen et al., 2017) as the framework for learning continually. None of these works deal with the issue of resource allocation to alleviate potential overfitting or underfitting problems in CL. The model we present adapts its size for CL, and the scenario where BNNs need to adapt to a changing data distribution or to *concept drift* in CL has been studied too (Kurle et al., 2019). Bayesian CL can be interpreted

as a regularisation in weight space. However, intuitively it is more sensible to regularise NN function outputs between tasks (Benjamin et al., 2018; Titsias et al., 2020a; Rudner et al., 2022b).

Adaptive models in continual learning. Non-Bayesian CL approaches use additional neural resources to learn new tasks and remember previous tasks. One approach boils down to learning individual networks for each task (Rusu et al., 2016a). More efficient use of resources can be done by selective retraining of neurons and expansion with a group sparsity regulariser (Yoon et al., 2017). However, this approach is unable to shrink and continues to expand if it overfits the first task. Another approach uses reinforcement learning by adding neural resources by penalizing the complexity of the task network in the reward function (Xu and Zhu, 2018). Recently Rao et al. (2019) proposed an unsupervised CL model (CURL) in scenarios that lack a task identifier. CURL is adaptive, insofar that if a new task is detected then a new component is added to the mixture of Gaussians in the model.

Task learning in CL can be modeled as a mixture of expert models. The experts are distributed according to a Dirichlet prior (Lee et al., 2020b); new experts can be added to the mixture automatically.

4.5 Experimental Results

To demonstrate the effectiveness of the IBP and H-IBP priors on determining the size of the BNN, we perform weight pruning to see whether the pruned weights coincide with the weights dropped by the IBP and H-IBP priors in Section 4.5.1. Furthermore, we then use the IBNN and HIBNN in a CL setting in Section 4.5.2. Also in the supplementary material, further continual learning results and all experimental details are outlined. Unless explicitly stated, all curves are an average of 5 independent runs $\pm$ one standard error. By test error, we refer to $(1 -$ accuracy$) \times 100$.[3] Note that the Z matrices are sampled per data point and they are sampled from the Concrete distribution so the values of $Z_{ij} \in [0, 1]$ Jang et al.

[3]Our code is available at `https://github.com/skezle/IBP_BNN`.

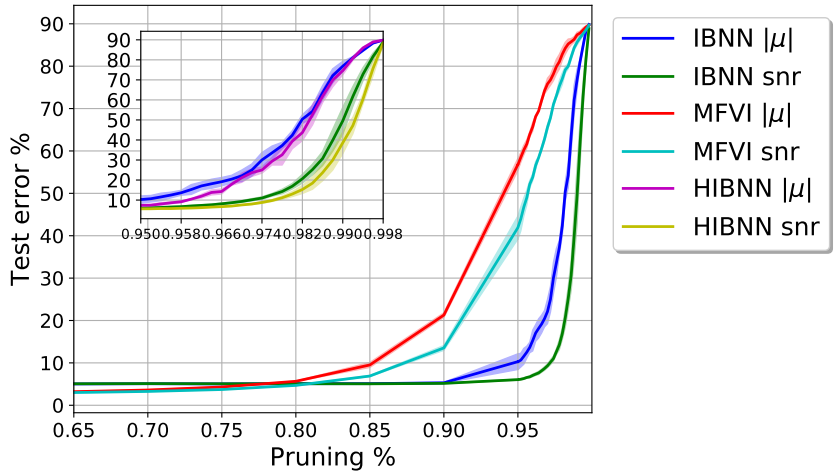


Figure 4.2: The weight pruning curves show test error versus the percentage of weights that have been zeroed out according to the magnitude of the variational mean and SNR ($|\boldsymbol{\mu}|/\boldsymbol{\sigma}$). The HIBNN and IBNN are much sparser than an MFVI BNN and the HIBNN is more robust to pruning and therefore sparser than the IBNN.

(2016); Maddison et al. (2016) and this Concrete variable is applied to each neuron. This doesn't follow the strict binary formulation of the IBP since we are using a Concrete relaxation of the Bernoulli distribution. We tried thresholding the IBP values, Z by some value $\tau \in [0, 1]$ such that $\bar{Z}_{ij} = 1$ if $Z_{ij} > \tau$ and 0 otherwise, but this produced worse results and adds an additional hyperparameter.

4.5.1 IBP induces sparsity

We perform weight pruning to see whether the IBP posterior sensibly selects neurons through the binary matrix Z. Weights are pruned in two ways. The first is pruning according to $|\boldsymbol{\mu}|$: zeroing out weights according to the magnitude of their mean. Important weights will be large in absolute value and so pruned last. Secondly, according to signal-to-noise ratio: $|\boldsymbol{\mu}|/\boldsymbol{\sigma}$ (SNR). Weights with high uncertainty will also be zeroed out first. Weight pruning is performed on MNIST and compared to a mean-field BNN (Blundell et al., 2015a) (denoted MFVI in plots). The pruning accuracies in Fig. 4.2 demonstrate that the HIBNN is indeed much sparser than a BNN and that pruning according to SNR is more robust, as expected. The HIBNN is more robust to pruning than the IBNN due to its inductive bias leading to the more regular structure Fig. 4.3. The baseline BNN has two layers with hidden

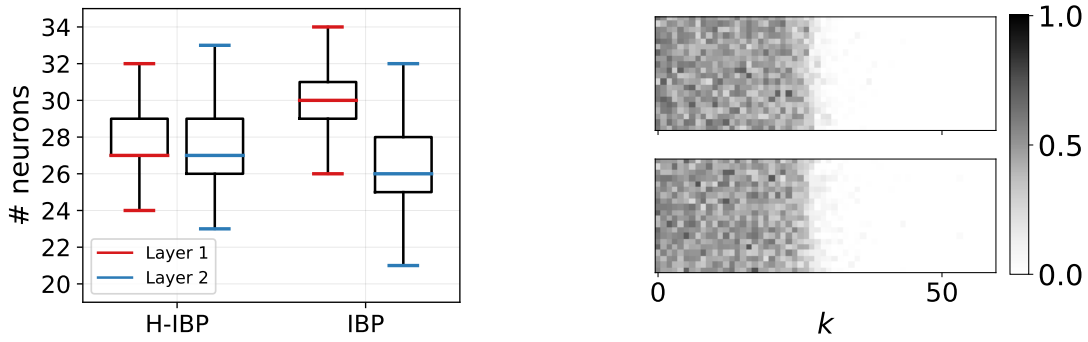


Figure 4.3: The number of active neurons in each layer of the IBNN and HIBNN. The HIBNN introduces an inductive bias that encourages and enables a regular structure. The Z matrices for the HIBNN show the neurons that are being learned on the left[4]. The Z matrix is discretely sampled and contains values in the range $[0, 1]$; it is not a one-hot vector since the Concrete distribution contains a temperature parameter that smooths the discrete distribution Jang et al. (2016).

state sizes of 200, the HIBNN and IBNN use a variational truncation of $K = 200$ for a fair comparison. The HIBNN and IBNN achieve an accuracy of 0.95 before pruning while MFVI achieves 0.98 before pruning. This gap in performance is due to the approximate inference of the H-IBP and IBP posteriors and the various reparameterizations used, in particular the Concrete reparameterization which applies a 'soft' mask on the hidden layers of the HIBNN. The IBNN and HIBNN are slightly less sparse compared to Sparse Variational Dropout which is specifically designed to be sparse (Molchanov et al., 2017), see Section 10.1.2.

Varying depth. As we increase the depth we see that the HIBNN and IBNN remain sparse while the MFVI's sparsity decreases, Fig. 4.4. We measure the sparsity as the pruning percentage at which the accuracy drops over 10% (the kinks in Fig. 4.4). There is little variation of the accuracy with depth for all models before pruning, however after pruning 95% of the weights with the SNR our models retain their performance while the MFVI BNN performs worse with depth since it becomes less sparse with depth Fig. 4.4. For the same analysis on fashion-MNIST, see Section 10.1.3.2.

[4] Z is usually shown in *left-ordered form*, however since the inference procedure is based on the stick-breaking construction, the order is meaningful and sorted according to k (Xu et al., 2019).

4.5.2 Continual learning experiments

Adaptive complexity. Approximate inference of the IBP and H-IBP posteriors is challenging in a stationary setting making the performance attenuated in comparison to a BNN with only independent Gaussian weights. Despite this, the approximate IBP and H-IBP posteriors are useful in non-stationary CL settings, where the amount of resources is unknown beforehand. In Fig. 4.5, one can see that the average accuracies across all CL tasks for permuted MNIST vary considerably with the hidden state size for VCL hence the benefit of our model which automatically infers the hidden state size for each task, see Fig. 4.6. The details of the experiment are introduced below.

Continual learning scenarios. Three different CL scenarios are used for evaluation (van de Ven and Tolias, 2018). The first is *task incremental learning* (`CL1`) where the task identifier is given during evaluation. The second is *domain incremental learning* (`CL2`), the task identifier needs to be inferred at test time. The domain increases with each new task and the models are required to perform binary classification. The third is *incremental class learning* (`CL3`), the task identifier and specific class need to be inferred at test time. The models are required to do multi-class classification for each task. During training, the task identifiers are given for all scenarios.[5] Using a multi-head architecture and the predictive entropy we can infer the task; the head with the lowest predictive entropy is chosen for `CL2` and `CL3` (Von Oswald et al., 2020). For Permuted MNIST `CL2` a single-head architecture is used (van de Ven and Tolias, 2018).

Baseline models. We compare our models to VCL (Nguyen et al., 2017), since the IBNN and HIBNN models build on top of it. We also compare to EWC (Kirkpatrick et al., 2017) and SI (Zenke et al., 2017a). We compare to DEN (Yoon et al., 2017) which is an expansion method that expands a neural network by a fixed number of weights for each new task, uses regularisation to mitigate overfitting, and freezes

[5] *Task-free CL* is a more challenging scenario and requires the model the infer new tasks during training.

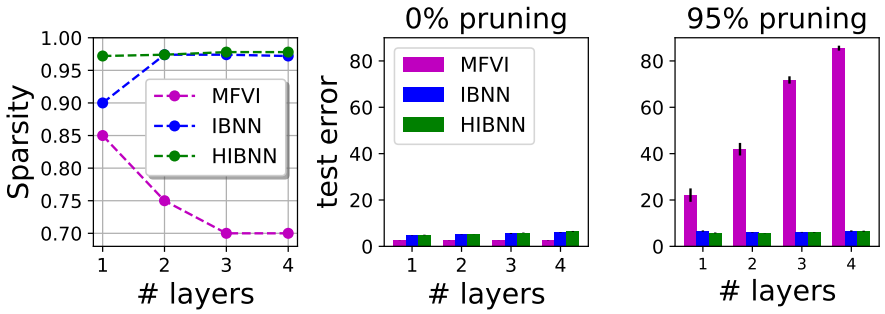


Figure 4.4: Left, as the depth of the IBNN and HIBNN increases the networks tend to remain very sparse while the MFVI BNN becomes less sparse. **Middle & Right**, the HIBNN and IBNN remain robust to pruning with increasing depth.

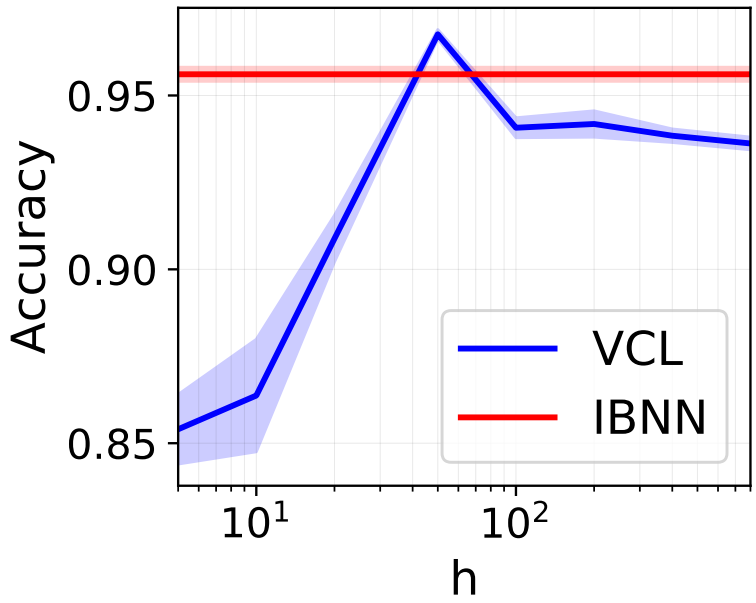


Figure 4.5: Test accuracies for Permuted MNIST `CL1`, for different VCL widths. Our model yields good results without having to specify a width, only a small range of values outperforms our model.

weights from previous tasks. For DEN the predictive entropy is used for inferring the correct head for `CL2` and `CL3`. Another baseline used is GEM (Lopez-Paz and Ranzato, 2017), which uses replay as its primary mechanism for alleviating forgetting. We hypothesize that GEM will be insensitive to model size. It also achieves strong results (Hsu et al., 2018b). VCL is the fairest comparison for our model and also uses uncertainties for `CL2` and `CL3`. Our objective is to demonstrate that limited or excessive neural resources can cause problems in CL in comparison to our adaptive model.

MNIST Experimental details. All models use a single layer with varying hidden state sizes. The use of a single layer is enough as MNIST is a simple task. The results for EWC in Table 4.1 on Split MNIST outperform those presented in (Hsu et al., 2018b; van de Ven and Tolias, 2018) which use larger models. We report accuracies for our non-adaptive baselines (EWC, SI, GEM, and VCL) over a set of hidden state sizes $\mathcal{H} = \{10, 50, 100, 400\}$. A hidden state of 10 might seem small but we also set the IBP prior parameter $\alpha = 5$ for task 1. This corresponds to only 5 neurons being selected by each data point in expectation. The initial hidden state size for DEN is set to 50.

Permuted MNIST benchmarks. The permuted MNIST benchmark involves performing multiclass classification on MNIST where in each task, the pixels have been shuffled by a fixed permutation. Our model is able to overcome overfitting and underfitting, these can cause increased forgetting which affects VCL. See Fig. 10.8 in the supplement, for a per-task accuracy breakdown. In contrast, the IBNN expands continuously Fig. 4.6. There is a small gap in performance between our model and VCL for $h = 50$ due to the approximations used for inference of the variational IBP posterior. Our method outperforms all regularization-based methods and DEN[6] on all CL scenarios and provides comparable results to GEM, see Table 4.1.

Split MNIST benchmarks. The split MNIST benchmark for CL involves a sequence of classification tasks of MNIST and more difficult variants with background noise and background images denoted S+ϵ and S+img. For EWC, SI, and VCL notice a considerable difference in performance with hidden state size in Table 4.1. GEM is sensitive for `CL3` only. EWC and SI perform well for `CL1` only. The IBNN outperforms VCL as it is not susceptible to overfitting or underfitting and thus propagating a subsequent poor posterior for a new task resulting in forgetting.

Split MNIST is a simple task that doesn't show overfitting, hence the use of the MNIST variant datasets where the IBNN outperforms all VCL models of different sizes as it is not susceptible to overfitting or underfitting. Indeed our method

[6]It is not clear how to reconcile single-head networks and DEN, thus `CL2` for DEN is omitted.

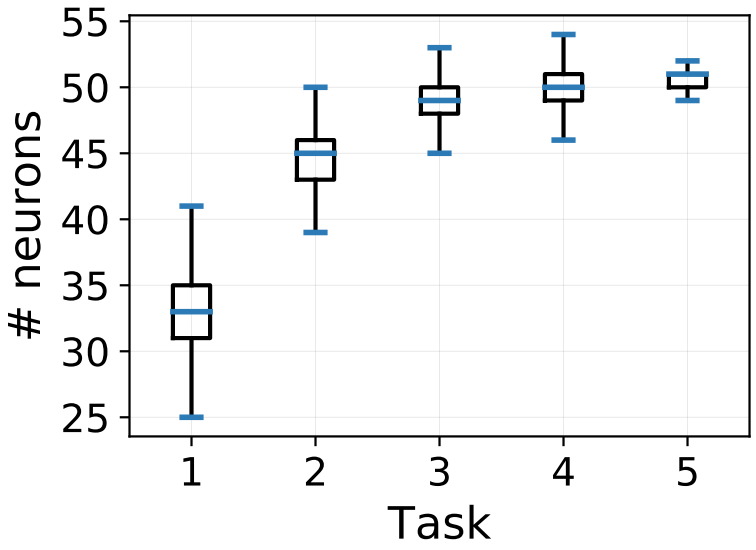


Figure 4.6: Number of active neurons selected by the IBP variational posterior for each task, for permuted MNIST `CL1`[7]. Our model is able to adapt and manage resources effectively.

outperforms not only VCL but EWC and SI and performs comparably to GEM. When analyzing the performance of the VCL baselines, we notice they have a tendency to overfit on the second task and propagate a poor approximate posterior and hence underperform in comparison to the IBNN model, Fig. 10.6 and Fig. 10.9 in the supplement. The IBNN increases its capacity over the course of CL, Fig. 10.7 in the supplement. The standard errors for VCL and our method on `CL3` are largely due to the severity of making mistakes for multiclass classification.

DEN performs well on all Split MNIST tasks and variants due to its "time-stamped inference" which segregates parts of the network per task and so uncertainties over seen tasks are well defined thus the good results for `CL2` and `CL3`. Indeed removing it renders performance comparable to IBNN, see Section 10.1.4 in the supplement. Statistical processes that mimic this could be an interesting direction for Bayesian expansion methods in CL.

Increasing task complexity. To test the expansion capabilities of our models we devise a set of tasks of increasing difficulty: two tasks from MNIST, followed by

[7]We define a neuron as active by aggregating all neurons where $z_{ik} > 0.1$ for data point $\mathbf{x_i}$ and neuron k.

Table 4.1: Average test accuracies on MNIST and variants over 5 runs. For EWC, SI, GEM, and VCL the median accuracy is taken from hidden state sizes, $\mathcal{H} = \{10, 50, 100, 400\}$. We also show the range between max and min average accuracies in $\mathcal{H}$. The models with the best median or mean accuracy are highlighted. If the IBNN mean accuracy lies within the min-max range then our model is also highlighted. Our IBNN achieves good performance overall compared to the baselines which can underfit/overfit. DEN performs very poorly on permuted MNIST.

	EWC (`max`, `min`)	SI (`max`, `min`)	GEM (`max`, `min`)	DEN	VCL (`max`, `min`)	IBNN
P `CL1`	90.0 (94.0, 79.1)	91.8 (95.1, 82.9)	94.8 (95.9, 88.4)	91.4±0.5	93.9 (96.8, 86.9)	**95.6±0.2**
P `CL2`	88.2 (93.0, 78.2)	89.2 (93.3, 84.1)	**95.6** (**96.7**, **88.1**)	-	88.7 (95.1, 75.2)	**93.7±0.6**
P `CL3`	59.3 (66.9, 24.0)	47.5 (55.4, 18.1)	**94.6** (**95.8**, **87.3**)	63.9±19.2	84.0 (94.1, 40.5)	**93.8±0.3**
S `CL1`	98.9 (99.0, 94.8)	98.1 (99.2, 95.5)	98.1 (98.2, 98.0)	**99.1±0.1**	96.6 (98.0, 93.7)	95.3±2.0
S `CL2`	63.7 (74.8, 63.3)	78.0 (80.1, 72.9)	94.0 (94.8, 92.2)	**98.9±0.1**	87.4 (94.9, 81.9)	91.0±2.2
S `CL3`	19.9 (21.8, 19.8)	18.8 (19.6, 15.4)	89.2 (89.6, 87.8)	**99.1±0.1**	69.0 (78.9, 66.3)	85.5±3.2
S+ϵ `CL1`	94.6 (96.5, 81.8)	88.4 (91.1, 72.7)	95.6 (95.7, 95.1)	**97.2±0.2**	89.2 (90.3, 86.3)	95.1±1.1
S+ϵ `CL2`	72.7 (74.6, 68.7)	66.1 (72.1, 61.8)	78.2 (78.7, 77.2)	84.8±16.7	69.1 (70.2, 61.7)	**89.7±3.8**
S+ϵ `CL3`	18.8 (19.0, 13.7)	13.7 (16.7, 10.4)	74.1 (74.4, 70.7)	**90.9±12.8**	32.5 (37.8, 30.0)	78.7±11.7
S+img `CL1`	88.2 (90.0, 78.1)	80.8 (87.4, 67.8)	91.7 (91.8, 91.1)	**93.8±0.8**	87.1 (87.9, 85.7)	91.6±1.2
S+img `CL2`	67.7 (73.9, 62.9)	63.3 (67.0, 58.5)	**74.1** (**76.4**, **74.0**)	**79.1±13.1**	70.4 (75.2, 65.3)	**80.5±7.8**
S+img `CL3`	17.0 (18.0, 12.0)	13.4 (16.4, 10.3)	63.9 (64.5, 56.3)	**91.6±5.1**	54.3 (66.1, 39.9)	66.2±13.4

Table 4.2: Accuracies for tasks of increasing difficulty. Accuracies are an average over 5 runs. VCL uses different hidden state sizes, $\mathcal{H} = \{50, 100, 400\}$, we show the range between the max and min average accuracies centered at the median. Our models are able to overcome underfitting.

	VCL (`max`, `min`)	IBNN	HIBNN
`CL1`	**82.3** (**86.1**, **81.2**)	80.8±2.0	**81.3±1.8**
`CL2`	79.9 (83.1, 76.7)	78.7±1.5	**81.5±1.4**

two from fashion MNIST, followed by two from CIFAR10. We compare the HIBNN and IBNN models to VCL with two layers and widths in $\mathcal{H} = \{50, 100, 400\}$. Our models have two layers with $K = 200$. The larger width VCL networks perform well but smaller ones exhibit forgetting due to underfitting. For the HIBNN we allow the hyperparameter α_j to increase for each new dataset seen i.e. every two tasks. We perform a random search over the IBP and H-IBP parameters for the IBNN and HIBNN models Section 10.1.6.1. The HIBNN performs better than the IBNN, additionally, we can see Fig. 4.7 that both of our models can have very different structures after learning on a task. Notice that since there is a sharing of parameters at a global level, the widths of the HIBNN match across different layers, unlike the IBNN.

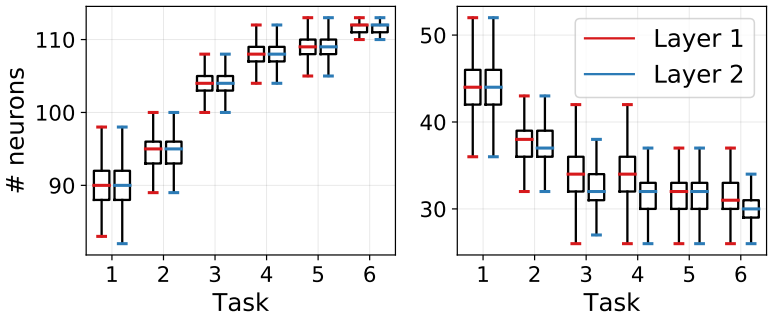


Figure 4.7: Inferred structure for the HIBNN and IBNN respectively for the experiments of tasks of increasing task complexity, with results in Table 4.2. Our models can infer varied structures, expand, and contract.

4.6 Conclusion and Discussion

Model size is an important contributing factor to CL performance. Most CL methods assume a perfectly selected model. Our novel Bayesian CL framework nonparametrically adapts the complexity of a BNN to the task difficulty. Our model is based on the IBP prior for selecting the number of neurons for each task and uses stochastic variational inference. The models presented reconcile two different approaches to CL: Bayesian or regularization-based approaches and dynamic architecture approaches through the use of an IBP and H-IBP prior. We have demonstrated our model on MNIST-like datasets. Future work will focus on showing that our method can scale to larger vision datasets that are used in CL. This will involve applying the IBP and H-IBP priors to Convolutional Neural Networks (CNN). Also ensuring that inference can be performed efficiently will be another challenge since inference in the IBNN and HIBNN is more expensive than for VCL (see Section 10.1.6.3) and VCL has been shown to produce poor results when scaling to these larger models (Pan et al., 2020). This could be performed with other inference methods such as using natural gradients (Osawa et al., 2019) or only training certain parts of the convolutional block of the CNN (Ovadia et al., 2019b). This is left for future work.

4.7 Limitations and Future Work

We perform inference on the IBP variational distribution, $q(Z)$ instance-wise by doing 1 step of gradient descent per data point. More specifically, we sample a single Z per data point and the Z's do not persist over batches. As a result, this requires more training iterations and epochs for the ELBO to converge than VCL Nguyen et al. (2017). A promising future direction of work is to infer a Z per task instead of per instance. This should allow for quicker convergence and make the optimization easier.

In addition to inferring the network width of a Bayesian neural network and allowing it to expand over the course of continual learning. We could also extend our work and allow the discrete selection over the number of layers of a deep network.

5

On Sequential Bayesian Inference for Continual Learning

Sequential Bayesian inference can be used for *continual learning* to prevent catastrophic forgetting of past tasks and provide an informative prior when learning new tasks. We revisit sequential Bayesian inference and assess whether using the previous task's posterior as a prior for a new task can prevent catastrophic forgetting in Bayesian neural networks. Our first contribution is to perform sequential Bayesian inference using Hamiltonian Monte Carlo. We propagate the posterior as a prior for new tasks by approximating the posterior via fitting a density estimator on Hamiltonian Monte Carlo samples. We find that this approach fails to prevent catastrophic forgetting demonstrating the difficulty in performing sequential Bayesian inference in neural networks. Furthermore, we study simple analytical examples of sequential Bayesian inference and CL and highlight the issue of model misspecification which can lead to sub-optimal continual learning performance despite exact inference. Furthermore, we discuss how task data imbalances can cause forgetting. From these limitations, we argue that we need probabilistic models of the continual learning generative process rather than relying on sequential Bayesian inference over Bayesian neural network weights. Our final contribution is to propose a simple baseline called *Prototypical Bayesian Continual Learning*,

which is competitive with the best performing Bayesian continual learning methods on class incremental continual learning computer vision benchmarks.

5.1 Introduction

The goal of continual learning (CL) is to find a predictor that learns to solve a sequence of new tasks without losing the ability to solve previously learned tasks. One key challenge of CL with neural networks (NNs) is that model parameters from previously learned tasks are "overwritten" during gradient-based learning of new tasks, which leads to *catastrophic forgetting* of previously learned abilities (McCloskey and Cohen, 1989; French, 1999). One approach to CL hinges on using recursive applications of Bayes' Theorem; using the weight posterior in a Bayesian neural network (BNN) as the prior for a new task (Kirkpatrick et al., 2017). However, obtaining a full posterior over NN weights is computationally demanding and we often need to resort to approximations, such as the Laplace method (MacKay, 1992a) or variational inference (Graves, 2011; Blundell et al., 2015a) to obtain a neural network weight posterior.

When performing Bayesian CL, sequential Bayesian inference is performed with an approximate BNN posterior, not the true posterior (Schwarz et al., 2018b; Ritter et al., 2018; Nguyen et al., 2017; Ebrahimi et al., 2019; Kessler et al., 2021a; Loo et al., 2020). If we consider the performance of sequential Bayesian inference with a variational approximation over a BNN weight posterior then we barely observe an improvement over simply learning new tasks with stochastic gradient descent (SGD). We develop this statement further in Section 5.2.2. So if we had access to the true BNN weight posterior, would this be enough to prevent forgetting by sequential Bayesian inference?

Our contributions in this chapter are to revisit Bayesian CL. 1) Experimentally, we perform sequential Bayesian inference using the true Bayesian NN weight posterior. We do this by using the gold standard of Bayesian inference methods, Hamiltonian Monte Carlo (HMC) (Neal et al., 2011). We use density estimation over HMC samples and use this approximate posterior density as a prior for the

next task within the HMC sampling process. Surprisingly our HMC method for CL yields no noticeable benefits over an approximate inference method (VCL (Nguyen et al., 2017)) despite using samples from the true posterior. 2) As a result we consider a simple analytical example and highlight that exact inference with a misspecified model can still cause forgetting. 3) We show mathematically that under certain assumptions task data imbalances cause forgetting in Bayesian NNs. 4) We propose a new probabilistic model for CL and show that by explicitly modeling the generative process of the data, we can achieve good performance, avoiding the need to rely on recursive Bayesian inference over NN weights to prevent forgetting. Our proposed model, *Prototypical Bayesian Continual Learning* (ProtoCL), is conceptually simple, scalable, and competitive with state-of-the-art Bayesian CL methods in the class-incremental learning setting.

5.2 Background

5.2.1 The Continual Learning Problem

Continual learning (CL) is a learning setting whereby a model must learn to make predictions over a set of tasks sequentially while maintaining performance across all previously learned tasks. In CL, the model is sequentially shown T tasks, denoted $\mathcal{T}_t$ for $t = 1, \ldots, T$. Each task, $\mathcal{T}_t$, is comprised of a dataset $\mathcal{D}_t = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N_t}$ which a model needs to learn to make predictions with. More generally, tasks are denoted by distinct tuples comprised of the conditional and marginal data distributions, $\{p_t(y|\mathbf{x}), p_t(\mathbf{x})\}$. After task, $\mathcal{T}_t$ the model loses access to the training dataset but its performance is evaluated on all tasks $\mathcal{T}_i$ for $i \leq t$. See Section 2.5 for a full description of different continual learning scenarios.

5.2.2 Bayesian Continual Learning

We consider a setting in which task data arrives sequentially at time steps, $t = 1, 2, \ldots, T$. At the first time step, $t = 1$, the model parameterized by $\boldsymbol{\theta}$ receives the dataset $\mathcal{D}_1$ and learns the conditional distribution $p(y_i|\boldsymbol{x}_i, \theta)$ for all $(\boldsymbol{x}_i, y_i) \in \mathcal{D}_1$ (i

indexes a datapoint), the parameters θ have a prior distribution $p(\theta)$. The posterior predictive distribution for a test point, $\boldsymbol{x}_1^*$ is:

$$p(y_1^*|\boldsymbol{x}_1^*, \mathcal{D}_1) = \int p(y_1^*|\boldsymbol{x}_1^*, \theta)p(\theta|\mathcal{D}_1)d\theta. \tag{5.2.2.1}$$

Computing this posterior predictive distribution above requires $p(\theta|\mathcal{D}_1)$. For $t = 2$, a CL model is required to fit $p(y_i|\boldsymbol{x}_i, \theta)$ for $\mathcal{D}_1 \cup \mathcal{D}_2$. The posterior predictive distribution for a new test point $\boldsymbol{x}_2^*$ point is:

$$p(y_2^*|\boldsymbol{x}_2^*, \mathcal{D}_1, \mathcal{D}_2) = \int p(y_2^*|\boldsymbol{x}_2^*, \theta)p(\theta|\mathcal{D}_1, \mathcal{D}_2)d\theta. \tag{5.2.2.2}$$

The posterior must thus be updated to reflect this new conditional distribution. We can use repeated application of Bayes' rule to calculate the posterior distributions $p(\theta|\mathcal{D}_1, \ldots, \mathcal{D}_T)$ as:

$$p(\theta|\mathcal{D}_1, \ldots, \mathcal{D}_{T-1}, \mathcal{D}_T) = \frac{p(\mathcal{D}_T|\theta)p(\theta|\mathcal{D}_1, \ldots, \mathcal{D}_{T-1})}{p(\mathcal{D}_T|\mathcal{D}_1, \ldots, \mathcal{D}_{T-1})}. \tag{5.2.2.3}$$

In the CL setting we lose access to previous training datasets: however, using repeated applications of Bayes' rule Eq. (5.2.2.3), allows us to sequentially incorporate information from past tasks in the parameters θ. At $t = 1$, we have access to $\mathcal{D}_1$ and the posterior over weights is:

$$\log p(\theta|\mathcal{D}_1) = \log p(\mathcal{D}_1|\theta) + \log p(\theta) - \log p(\mathcal{D}_1). \tag{5.2.2.4}$$

At $t = 2$, we require a posterior $p(\theta|\mathcal{D}_1, \mathcal{D}_2)$ to calculate the posterior predictive distribution in Eq. (5.2.2.2). However, we have lost access to $\mathcal{D}_1$. According to Bayes' rule, the posterior may be written as:

$$\log p(\theta|\mathcal{D}_1, \mathcal{D}_2) = \log p(\mathcal{D}_2|\theta) + \log p(\theta|\mathcal{D}_1) - \log p(\mathcal{D}_2|\mathcal{D}_1), \tag{5.2.2.5}$$

where we used the conditional independence of $\mathcal{D}_2$ and $\mathcal{D}_1$ given θ. We note that the likelihood is only dependent upon the current task dataset, $\mathcal{D}_2$, and that the prior encodes parameter knowledge from the previous task. Hence, we can use the posterior at t as a prior for learning a new task at $t + 1$. From Eq. (5.2.2.5) we require that our model with parameters θ is a sufficient statistic of $\mathcal{D}_1$, making the likelihood conditionally independent of $\mathcal{D}_1$ given θ. This observation motivates the use of high-capacity predictors, such as Bayesian neural networks, that are flexible enough to learn $\mathcal{D}_1$.

5.2.2.1 Continual Learning Example: Split-MNIST

For the MNIST dataset (LeCun et al., 1998) we know that if we were to train a BNN we would achieve good performance by inferring the posterior $p(\theta|\mathcal{D})$ Section 2.6 and integrating out the posterior to infer the posterior predictive distribution over a test point Eq. (5.2.2.1). So if we were to split the dataset MNIST into 5 two-class classification tasks then we should be able to recursively recover the multi-task posterior $p(\theta|\mathcal{D}) = p(\theta|\mathcal{D}_1 \ldots, \mathcal{D}_5)$ using Eq. (5.2.2.5). This problem is called Split-MNIST (Zenke et al., 2017a), where the first task involves the classification of the digits $\{0, 1\}$ then the second task classification of the digits $\{2, 3\}$ and so on.

We can define 3 different CL settings (Hsu et al., 2018a; Van de Ven and Tolias, 2019; van de Ven et al., 2022). When we allow the CL agent to make predictions with a task identifier τ the scenario is referred to as *task-incremental*. The identifier τ could be used to select different heads Section 5.2.1, for instance. This scenario is not compatible with sequential Bayesian inference outlined in Eq. (5.2.2.5) since no task identifier is required for making predictions. *Domain-incremental* learning is another scenario that doesn't have access to τ during evaluation and requires the CL agent to perform classification to the same output space for each task, for example for Split-MNIST the output space is $\{0, 1\}$ for all tasks, so this amounts to classifying between even and odd digits. Domain incremental learning is compatible with sequential Bayesian inference with a Bernoulli likelihood. The third scenario is *class-incremental* learning which also doesn't have access to τ but the agent needs to classify each example to its corresponding class. For Split-MNIST, for example, the output space is $\{0, \ldots, 9\}$ for each task. Class-incremental learning is compatible with sequential Bayesian inference with a categorical likelihood.

5.2.3 Variational Continual Learning

Variational CL (VCL; (Nguyen et al., 2017)) simplifies the Bayesian inference problem in Eq. (5.2.2.5) into a sequence of approximate Bayesian updates on the distribution over random neural network weights $\boldsymbol{\theta}$. To do so, VCL uses the variational posterior from previous tasks as a prior for new tasks. In this way,

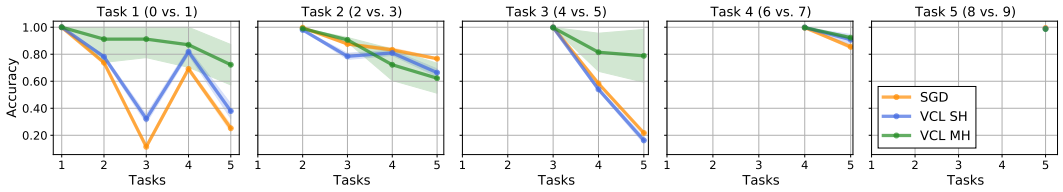


Figure 5.1: Accuracy on Split-MNIST for various CL methods with a two-layer BNN, all accuracies are an average and standard deviation over 10 runs with different random seeds. We compare an NN trained with SGD (single-headed) with VCL. We consider single-headed (SH) and multi-head (MH) VCL variants, i.e. domain and task incremental learning respectively.

learning to solve the first task entails finding a variational distribution $q_1(\theta|\mathcal{D}_1)$ that maximizes a corresponding variational objective. For the subsequent task, the prior is chosen to be $q_1(\theta|\mathcal{D}_1)$, and the goal becomes to learn a variational distribution $q_2(\theta|\mathcal{D}_2)$ that maximizes a corresponding variational objective under this prior. Denoting the recursive posterior inferred from multiple datasets by $q_t(\theta|\mathcal{D}_{1:t})$, we can express the variational CL objective for the t-th task as:

$$\mathcal{L}(\theta, \mathcal{D}_t) = D_{\mathrm{KL}}\left[q_t(\theta)||q_{t-1}(\theta|\mathcal{D}_{1:t-1})\right] - \mathbb{E}_{q_t}[\log p(\mathcal{D}_t|\theta)]. \tag{5.2.3.1}$$

When applying VCL to the problem of Split-MNIST Fig. 5.1, we can see that single-headed VCL barely performs better than SGD when remembering past tasks. Multi-headed VCL performs better, despite not being a requirement from sequential Bayesian inference Eq. (5.2.2.5). So why does single-head VCL not improve over SGD if we can recursively build up an approximate posterior using Eq. (5.2.2.5)? We hypothesize that it could be due to using a variational approximation of the posterior and so we are not actually strictly performing the Bayesian CL process described in Section 5.2.2. We test this hypothesis in the next section by propagating the true BNN posterior to verify whether we can recursively obtain the true multi-task posterior and so improve on single-head VCL and prevent catastrophic forgetting.

5.3 Bayesian Continual Learning with Hamiltonian Monte Carlo

To perform inference over BNN weights we use the HMC algorithm (Neal et al., 2011). We then use these samples and learn a density estimator that can be used as a prior for a new task[1]. HMC is considered the gold standard in approximate inference and is guaranteed to asymptotically produce samples from the true posterior[2]. We use posterior samples of $\boldsymbol{\theta}$ from HMC and then fit a density estimator over these samples, to use as a prior for a new task. This allows us to use a multi-modal posterior distribution over $\boldsymbol{\theta}$. In contrast, to a diagonal Gaussian variational posterior like in VCL. More concretely, to propagate the posterior $p(\theta|\mathcal{D}_1)$ we use a density estimator, defined $\hat{p}(\theta|\mathcal{D}_1)$, to fit a probability density on HMC samples as a posterior. For the next task $\mathcal{T}_2$ we can use $\hat{p}(\theta|\mathcal{D}_1)$ as a prior for a new HMC sampling chain and so on (see Fig. 5.2). The density estimator priors need to satisfy two key conditions for use within HMC sampling. Firstly, that they are a probability density function. Secondly, that they are differentiable with respect to the input samples.

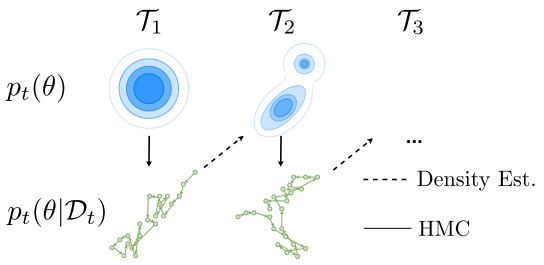


Figure 5.2: Illustration of the posterior propagation process with HMC; priors in blue are in the top row and posterior samples on the bottom row. This is a two-step process where we first perform HMC with an isotropic Gaussian prior for $\mathcal{T}_1$ then perform density estimation on the HMC samples from the posterior to obtain $\hat{p}_1(\theta|\mathcal{D}_1)$. This posterior can then be used as a prior for the new task $\mathcal{T}_2$ and so on.

We use a toy dataset (Fig. 5.3) with two classes and inputs $\boldsymbol{x} \in \mathbb{R}^2$ (Pan et al., 2020). Each task is a binary classification problem where the decision boundary extends from left to right for each new task. We train a two-layer BNN, with a hidden state size of 10. We use a Gaussian Mixture Model (GMM) as a density estimator for approximating the posterior with HMC samples. We also tried Normalizing Flows which should be more flexible (Dinh et al., 2016) however these did not work

[1]We considered Sequential Monte Carlo, but it is unable to scale to the dimensions required for the NNs we consider (Chopin et al., 2020). HMC on the other hand has recently been successfully scaled to relatively small BNNs of the size considered in this chapter (Cobb and Jalaian, 2021) and ResNet models but at large computational cost (Izmailov et al., 2021).

[2]In the NeurIPS 2021 Bayesian Deep Learning Competition, the goal was to find an approximate inference method that is as "close" as possible to the posterior samples from HMC.

robustly for HMC sampling[3]. To the best of our knowledge, we are the first to incorporate flexible priors into sampling methods like HMC.

Training a BNN with HMC on the same multi-task dataset gets a test accuracy of 1.0. Thus, the final posterior is suitable for continual learning under Eq. (5.2.2.3) we can recursively arrive at the multi-task posterior with recursive Bayesian inference. To be clear, since the test accuracy for the multi-task posterior is 1.0 this doesn't mean that HMC has estimated the true posterior. It only means that if we were to sequentially build up the posterior Eq. (5.2.2.3) then we should expect an accuracy of 1.0 as well since the multi-task posterior predictive performance is an upper bound to the performance of the sequential Bayesian inference for continual learning Section 2.6. To show that we recover the true posterior we would need some kind of oracle posterior distribution which we would have to show that we can obtain sequentially.

The results from Fig. 5.3 demonstrate that using HMC with an approximate multi-modal posterior fails to prevent forgetting and is less effective than using multi-head VCL. In fact, multi-head VCL clearly outperforms HMC indicating that the source of the knowledge retention is not through the propagation of the posterior but through the task-specific heads. For $\mathcal{T}_2$ we use $\hat{p}(\theta|\mathcal{D}_1)$ instead of $p(\theta|\mathcal{D}_1)$ as a prior and this will bias the HMC sampling for all subsequent tasks. In the next paragraph, we detail the measures taken to ensure that our HMC chains have converged, and so can assume that we are sampling from the true posterior. Also, we assess the fidelity of the GMM density estimator with respect to the HMC samples. We also repeated these experiments with another toy dataset of five binary classification tasks where we observed similar results (Fig. 10.13).

For HMC we ensure that we are sampling from the posterior by assessing chain convergence and effective sample sizes (Fig. 10.17). The effective sample size measures the autocorrelation in the chain. The effective sample sizes for the HMC chains for our BNNs are similar to the literature (Cobb and Jalaian, 2021).

[3]RealNVP was very sensitive to the choice of random seed, the samples from the learned distribution did not give accurate predictions for the current task and led to numerical instabilities when used as a prior within HMC sampling.

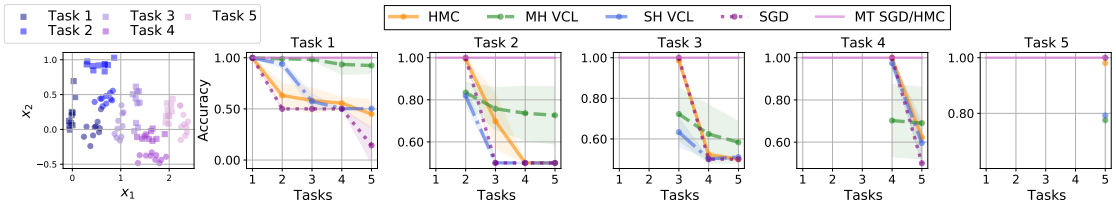


Figure 5.3: On the left is the toy dataset of 5 distinct 2-way classification tasks which involve classifying circles and squares (Pan et al., 2020). Also, continual learning binary classification test accuracies over 10 seeds. The pink solid line is a multi-task (MT) baseline accuracy using SGD/HMC with the same model as for the CL experiments. This is a domain incremental learning scenario, entirely consistent with sequential Bayesian inference Section 2.6.

Also, we ensure that the GMM approximate posterior is multi-modal and so has a more complex posterior in comparison to VCL and that the GMM samples produce equivalent results to HMC samples for the current task (Fig. 10.16). That our approximate GMM posterior obtains the same results as HMC samples doesn't mean we have estimated the true posterior, but that the GMM is included within the support of the true posterior. See Section 10.2.2 for details.

The 2-d benchmarks we consider in this section are from previous works and are domain-incremental continual learning problems. The domain incremental setting is also simpler (van de Ven et al., 2022) than the class-incremental setting and thus a good starting point when attempting to perform exact sequential Bayesian inference. Despite this, we are not able to perform sequential Bayesian inference in BNNs despite using HMC which is considered the gold standard of Bayesian deep learning. HMC and density estimation with a GMM produces richer, more accurate, and multi-modal posteriors. Despite this, we are still not able to sequentially build up the multi-task posterior or get much better results than an isotropic Gaussian posterior like single-head VCL. The weak point of this method is the density estimation, the GMM removes probability mass over areas of the BNN weight space posterior which is important for the new task. This demonstrates just how difficult a task it is to model BNN weight posteriors. In the next section, we study a different analytical example of sequential Bayesian inference and look at how model misspecification and task data imbalances can cause forgetting in Bayesian CL.

5.4 Bayesian Continual Learning and Model Misspecification

We now consider a simple analytical example where we can perform the sequential Bayesian inference Eq. (5.2.2.3) in closed form using conjugacy. We consider a Bayesian linear regression model with Gaussian likelihoods for 2 continual learning tasks. This simple example highlights that forgetting Equation (2.6.1.3) may occur under certain conditions despite correct inference. We also introduce the metric of *Bayesian forgetting* which contains the notion of multi-task performance as a comparison which is what the sequential posterior aims to infer. Bayesian forgetting also allows us to identify continual learning settings where our model is misspecified and achieves sub-optimal continual learning performance.

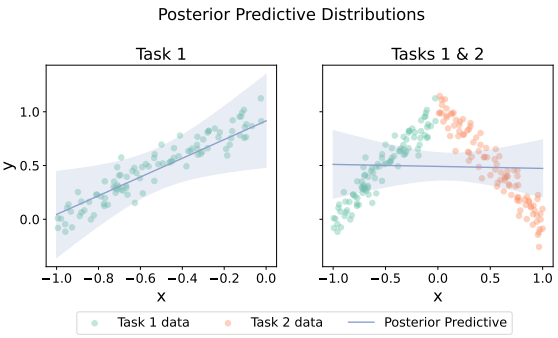


Figure 5.4: Posterior predictive distributions for a Bayesian linear regression model. Left, data comes from a single task that the Bayesian linear model can fit well. Right, a new dataset is obtained from a different part of the domain, and our sequentially updated Bayesian linear regression models correctly a global solution to both of these datasets which is sub-optimal for both.

5.4.1 Bayesian Sequential Inference can Forget

We use a Gaussian likelihood of the form $p(\mathcal{D}|\boldsymbol{\theta}) = \mathcal{N}(y; f(X;\boldsymbol{\theta}), \beta^{-1})$ such that $y = f(X;\boldsymbol{\theta}) + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \beta^{-1})$ and $f(X;\boldsymbol{\theta}) = \boldsymbol{\theta}X^\top$. We put a Gaussian prior over the parameters $\boldsymbol{\theta}$ such that $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{m}_0, \Sigma_0)$ for our first task. Via conjugacy of the Gaussian prior and likelihood, the posterior is also Gaussian, $p(\boldsymbol{\theta}|\mathcal{D}) = \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{m}, \Sigma)$ where $\boldsymbol{m} = \Sigma(\Sigma_0^{-1}\boldsymbol{m}_0 + \beta X^\top \boldsymbol{y})$ and $\Sigma^{-1} = \Sigma_0^{-1} + \beta X^\top X$ for task 2 and onwards. By using sequential Bayesian inference we can have closed-form update equations for our parameters.

From the form of the linear regression posterior, our model can only model linear data. So, if we have data that is linear and drawn from a second task, from a distinct part of the domain, then the model correctly models a linear model, which

is the Bayes solution of the multi-task problem Section 2.6. For example, the task 1 dataset is generated according to $y = x + 1 + \epsilon$, where $\epsilon = \mathcal{N}(0, \beta^{-1})$ for $x \in [-1, 0)$. From Fig. 5.4 we can see that our Bayesian linear regression accurately models this first dataset. Now, if we sequentially model a second dataset, with data drawn from $y = -x + 1 + \epsilon$, where $\epsilon = \mathcal{N}(0, \beta^{-1})$ for $x \in [0, 1]$. The model regresses to both of these datasets: the continual learning regression from using the posterior from task 1 as a prior for task 2 is the same as if we were to regress to the multi-task dataset of both tasks 1 and 2 (see Fig. 5.4 on the right)[4].

However, as we can see from Fig. 5.4 (right), this is a suboptimal continual learning solution, since we want the average performance Eq. (2.6.1.1) to be high for all tasks. More specifically the performance after learning task 2 is $P_2 = p_{2,2} + p_{2,1}$ where $p_{i,j}$ is the performance for task j after learning task i and $j \leq i$. Higher is better for the performance measure $p_{i,j}$, for regression $p_{i,j}$ could be the log-likelihood. We see however, that performance is low for the exact sequential Bayesian linear regression continual learning model Fig. 5.4. As with all continual learning benchmarks, we require our model to perform equally well on both tasks. In this case, we can specify a better model which is not just a Bayesian linear regression model, but a mixture of linear regressors Bishop (2006). *Despite performing exact inference a misspecified model can forget.* We get forgetting of task 1 after learning task 2, since the forgetting metric Eq. (2.6.1.3), is $f_1^2 = p_{1,1} - p_{2,1} > 0$ since $p_{1,1} > p_{2,1}$ which indicates forgetting, as can be seen from Fig. 5.4.

In the case of HMC, we verified that our Bayesian neural network had perfect performance on all tasks *beforehand*. In Section 5.3 we had a well-specified model but struggled with exact sequential Bayesian inference Eq. (5.2.2.3). With this Bayesian linear regressions scenario we are performing exact inference, however, we have a misspecified model and so unable to obtain good performance on both tasks Section 2.6. It is important to disentangle model misspecification and exact

[4]A previous version of this section had a similar example where data for both tasks was generated from the same domains $p_1(\boldsymbol{x}) = p_2(\boldsymbol{x})$, but the conditional distributions mapped to different outputs $p_1(y|\boldsymbol{x}) \neq p_2(y|\boldsymbol{x})$. In this previous example, the tasks were conflicting and so the Bayesian multitask posterior was also sub-optimal for continual learning. Now, in this 2 task linear regression setup, the domains for both tasks are distinct and so the tasks are not conflicting.

inference and highlight that model misspecification is a caveat that has not been highlighted in the CL literature as far as we are aware. Furthermore, we can only ensure that our models are well specified if we have access to data from all tasks a priori. So in the scenario of *online continual learning* (Aljundi et al., 2019c,b; De Lange et al., 2019) we cannot know if our model will perform well on all past and future tasks without making assumptions on the task distributions.

5.4.2 Bayesian Forgetting

The objective of sequential Bayesian inference is to recover the multi-task posterior. The current definition of forgetting Eq. (2.6.1.3) doesn't have a notion of multi-task performance as a comparison. Additionally as highlighted in Section 5.4.1, we can have forgetting but it is unclear if this is due to an inappropriate model for the continual learning problem or issues with the (approximate) inference.

We, therefore, put forward the metric of *Bayesian forgetting* for task i after having seen j tasks in total where $i \in \{1, \dots, j-1\}$ (we can only calculate the forgetting for tasks up until $j-1$):

$$\varphi_i^j = p_{1:j}^i - p_{ij}, \quad j > 1 \quad \text{and} \quad j > i, \tag{5.4.2.1}$$

where $p_{1:j}^i$ is the multi-task model performance on task i having inferred the posterior using multi-task data from tasks 1 to j. The term p denotes the performance (higher is better), and for example, could be the test log-likelihood or test accuracy using the posterior. Comparing this to forgetting Eq. (2.6.1.3) which we repeat for clarity:

$$f_i^j = p_{ii} - p_{ij}, \quad j > 1 \quad \text{and} \quad j > i, \tag{5.4.2.2}$$

which compares the performance on task i after learning task j with the original performance after learning task i, p_{ii}. The difference is that Bayes forgetting compares to the multi-task posterior.

Bayes forgetting requires accurate inference of posteriors, so it's applicability is limited to small models and requires state-of-the-art inference techniques like HMC. Also, another issue it is expensive to calculate since we need to infer a multi-task

posterior for each prefix of tasks. If we have tasks $\mathcal{T}_i$ for $i \in \{1, \ldots, T\}$ we need to obtain multi-task models $\mathcal{M}_{1:i}$ for $i \in \{2, \ldots, T\}$.

We can interpret $\varphi_i^j > 0$ as displaying Bayesian forgetting, our sequential inference scheme is sub-optimal since we do not arrive at a model with the same performance as our multi-task posterior. If $\varphi_i^j = 0$ then we have no Bayesian forgetting and the posterior predictive performance from the continual learning model is equivalent to the multi-task model's performance. The case where $\varphi_i^j < 0$ is not possible under sequential Bayesian inference since the multi-task performance is an upper bound to the continual learning performance (and we assume we do not have conflicting tasks Lin et al. (2019); Kessler et al. (2021b) which will negatively affect multi-task performance).

If we compare Bayesian forgetting Equation (5.4.2.1) with the traditional definition of forgetting Equation (2.6.1.3) then if forgetting is high and the Bayesian forgetting is zero: $\varphi_i^j = 0$ and $f_i^j > 0$ this is an indication of model misspecification Fig. 5.5. In the case where $\varphi_i^j > 0$ and $f_i^j > 0$ then both forgetting metrics are exhibited and this is the situation in Section 5.3 where we have a difficult sequential Bayesian inference problem. In the case where $\varphi_i^j = 0$ and $f_i^j = 0$ we have no forgetting and no Bayes forgetting, this suggests the multi-task posterior and the continual learning posterior have similar predictive performance and performance hasn't degraded after learning task j. In the case where $\varphi_i^j > 0$ and $f_i^j = 0$ we have no forgetting but we have Bayes forgetting, this suggests that there is some performance degradation with respect to the multi-task posterior and some transfer of knowledge is happening for tasks $\mathcal{T}_k$ for $k \in \{i+1, \ldots, j\}$ which boosts

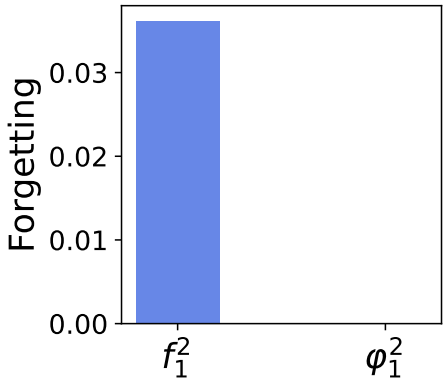


Figure 5.5: Forgetting Equation (5.4.2.2) and Bayes forgetting Equation (5.4.2.1) for the Bayesian linear regression problem in Section 5.4.1, we use the log-likelihood as a performance metric to calculate forgetting metrics. This shows no Bayesian forgetting since the sequential inference is equal to the multi-task posterior inference, in contrast, we observe forgetting. The values for both of these metrics indicates the model class is not suitable for the continual learning problem.

the performance of the multi-task posterior vis-a-vis p_{ii} used in the forgetting metric Equation (5.4.2.2). With both of these forgetting metrics, we can identify whether we have good continual performance within a model-class and can identify if the model class is suitable for the continual learning problem posed.

5.5 Sequential Bayesian Inference and Imbalanced Task Data

Neural Networks are complex models with a broad hypothesis space and hence are a suitably well-specified model when tackling continual learning problems (Wilson and Izmailov, 2020). However, we struggle to fit the posterior samples from HMC to perform sequential Bayesian inference in Section 5.3.

We continue to use Bayesian filtering and assume a Bayesian NN where the posterior is Gaussian with a full covariance. By modeling the entire covariance we enable modeling of how each individual weight varies with respect to all others. We do this by interpreting online learning in Bayesian NNs as filtering (Ciftcioglu and Türkcan, 1995). Our treatment is similar to (Aitchison, 2020) who derives an optimizer by leveraging Bayesian filtering. We consider inference in the graphical model depicted in Fig. 5.6. The aim is to infer the optimal BNN weights, θ_t^* at t given a single observation and the BNN weight prior. The previous BNN weights are used as a prior for inferring the posterior BNN parameters. We consider the online setting where a single data point $(\boldsymbol{x}_t, y_t)$ is observed at a time.

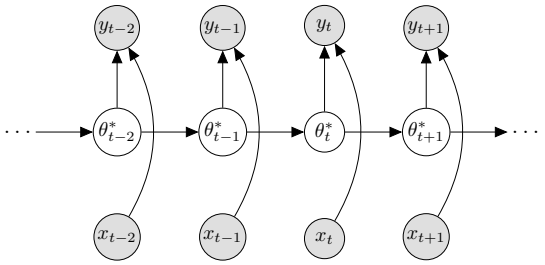


Figure 5.6: Graphical model for filtering. Grey and white nodes are observed and latent variables respectively.

Instead of modeling the full covariance we instead consider each parameter θ_i as a function of all the other parameters θ_{-it}. We also assume that the values of the weights are close to those of the previous timestep (Jacot et al., 2018). To

obtain the update equations for BNN parameters given a new observation and prior we make two simplifying assumptions as follows.

Assumption 5.5.1. *For a Bayesian neural network with output $f(\boldsymbol{x}_t;\theta)$ and likelihood $\mathcal{L}(\boldsymbol{x}_t, y_t;\theta)$, the derivative evaluated at θ_t is $\mathbf{z}_t = \partial\mathcal{L}(\boldsymbol{x}_t, y_t;\theta)/\partial\theta|_{\theta=\theta_t}$ and the Hessian is $\boldsymbol{H}$. We assume a quadratic loss for a data point $(\boldsymbol{x}_t, y_t)$ of the form:*

$$\mathcal{L}(\boldsymbol{x}_t, y_t;\theta) = \mathcal{L}_t(\theta) = -\frac{1}{2}\theta^\top \boldsymbol{H}\theta + \mathbf{z}_t^\top\theta, \tag{5.5.0.1}$$

the result of a second-order Taylor expansion. The Hessian is assumed to be constant with respect to $(\boldsymbol{x}_t, y_t)$ (but not with respect to $\boldsymbol{\theta}$).

To construct the dynamical equation for θ, consider the gradient for the i-th weight while all other parameters are set to their current estimate at the optimal value for the θ^*_{it}:

$$\theta^*_{it} = -\frac{1}{H_{ii}}\boldsymbol{H}^\top_{-ii}\theta_{-it}, \tag{5.5.0.2}$$

since $z_{it} = 0$ at a mode. The equation above shows us that the dynamics of the optimal weight θ^*_{it} is dependent on all the other current values of the parameters θ_{-it}. The dynamics of θ_{-it} are a complex stochastic process dependent on many different variables: such as the dataset, model architecture, learning rate schedule, etc.

Assumption 5.5.2. *Since reasoning about the dynamics of θ_{-it} are intractable, we assume that at the next time-step the optimal weights are close to the previous timesteps with a discretized Ornstein-Uhlenbeck process for the weights θ_{-it} with reversion speed $\vartheta \in \mathbb{R}_+$ and noise variance η^2_{-i}:*

$$p(\theta_{-i,t+1}|\theta_{-i,t}) = \mathcal{N}((1-\vartheta)\theta_{-it}, \eta^2_{-i}), \tag{5.5.0.3}$$

this implies that the dynamics for the optimal weight is defined by

$$p(\theta^*_{i,t+1}|\theta^*_{i,t}) = \mathcal{N}((1-\vartheta)\theta^*_{it}, \eta^2), \tag{5.5.0.4}$$

where $\eta^2 = \eta^2_{-i}\boldsymbol{H}^\top_{-ii}\boldsymbol{H}_{-ii}$.

In simple terms, in Assumption 5.5.2 we assume a parsimonious model of the dynamics. That the next value of $\boldsymbol{\theta}_{-i,t}$ is close to their previous value according to a Gaussian, similarly to Aitchison (2020).

Lemma 5.5.3. *Under Assumptions 5.5.1 and 5.5.2 the dynamics and likelihood are Gaussian. Thus we are able to infer the posterior distribution over the optimal weights using Bayesian updates and by linearizing the BNN the update equations for the posterior of the mean and variance of the BNN for a new data point are:*

$$\mu_{t,post} = \sigma^2_{t,post}\left(\frac{\mu_{t,prior}}{\sigma^2_{t,prior}(\eta^2)} + \frac{y_t}{\sigma^2}g(\boldsymbol{x}_t)\right) \qquad \textit{and} \qquad \frac{1}{\sigma^2_{t,post}} = \frac{g(\boldsymbol{x}_t)^2}{\sigma^2} + \frac{1}{\sigma^2_{t,prior}(\eta^2)}, \tag{5.5.0.5}$$

where we drop the notation for the i-th parameter, the posterior is $\mathcal{N}(\theta^*_t; \mu_{t,post}, \sigma^2_{t,post})$ and $g(\boldsymbol{x}_t) = \frac{\partial f(\boldsymbol{x}_t;\theta^*_{it})}{\partial \theta^*_{it}}$ and $\sigma^2_{t,prior}$ is a function of η^2.

See Section 10.2.5 for the derivation of Lemma 5.5.3. From Eq. (5.5.0.5) we can notice that the posterior mean depends linearly on the prior and a data-dependent term and so behaves similarly to our previous example in Section 5.4. Under Assumption 5.5.1 and Assumption 5.5.2 then if there is a data imbalance between tasks in Eq. (5.5.0.5), then the data-dependent term dominates the prior term if there is more data for the current task.

In Section 5.3 we showed that it is very difficult with current machine learning tools to perform sequential Bayesian inference for simple CL problems with small Bayesian NNs. When we disentangle Bayesian inference and model misspecification we showed that misspecified models can forget despite exact Bayesian inference. The only way to ensure that our model is well specified is to show that the multi-task posterior produces reasonable posterior predictive distributions $p(y|\boldsymbol{x}, \mathcal{D}) = \int p(y|\boldsymbol{x}, \mathcal{D}, \boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{D})d\boldsymbol{\theta}$ for one's application. Additionally, in this section, we have shown that if there is a task dataset size imbalance then we can get forgetting under certain assumptions.

5.6 Related Work

There has been a recent resurgence in the field of CL (Thrun and Mitchell, 1995b) given the advent of deep learning. Methods that approximate sequential Bayesian inference Eq. (5.2.2.5) have been seminal in CL's revival and have used a diagonal Laplace approximation (Kirkpatrick et al., 2017; Schwarz et al., 2018b). The diagonal Laplace approximation has been enhanced by modeling covariances of between neural network weights in the same layer (Ritter et al., 2018). Instead of the Laplace approximation, we can use a variational approximation for sequential Bayesian inference (Nguyen et al., 2017; Zeno et al., 2018). Using richer priors has also been explored (Ahn et al., 2019; Farquhar et al., 2020; Kessler et al., 2021a; Mehta et al., 2021; Kumar et al., 2021; Loo et al., 2020). Gaussian processes have also been applied to CL problems leveraging inducing points to retain previous task functions (Titsias et al., 2020b; Kapoor et al., 2021).

Bayesian methods that regularize weights have not matched up to the performance of experience replay-based CL methods (Buzzega et al., 2020) in terms of accuracy on CL image classification benchmarks. Instead of regularizing high-dimensional weight spaces, regularizing task functions is a more direct approach to combat forgetting (Benjamin et al., 2018). Bayesian NN weights can also be generated by a hypernetwork, where the hypernetwork needs only simple CL techniques to prevent forgetting (Henning et al., 2021). In particular, one can leverage the duality between the Laplace approximation and Gaussian Processes to develop a functional regularization approach to Bayesian CL (Swaroop et al., 2019) or using function-space variational inference (Rudner et al., 2022a,b).

In the next section, we propose a simple Bayesian continual learning baseline that models the data-generating continual learning process and performs exact sequential Bayesian inference in a low-dimensional embedding space. Previous work has explored modeling the data-generating process by inferring the joint distribution of inputs and targets $p(\mathbf{x}, \mathrm{y})$ and learning a generative model to replay data to prevent forgetting (Lavda et al., 2018), and by learning a generative model per

class and evaluating the likelihood of the inputs given each class $p(\mathbf{x}|\mathrm{y})$ (van de Ven et al., 2021).

5.7 Prototypical Bayesian Continual Learning

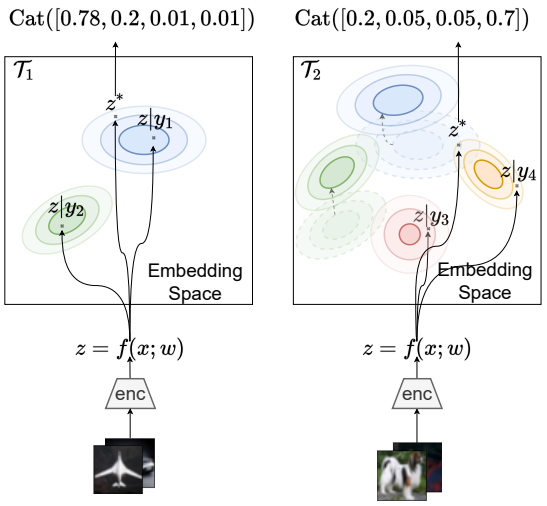


Figure 5.7: Overview of ProtoCL.

We have shown that sequential Bayes over NN parameters is very difficult (Section 5.3), and is only suitable for situations where the multi-task posterior is suitable for all tasks. We now show that a more fruitful approach is to model the generative CL problem with a generative classifier where each class is represented by a prototype and classification is distance-based to the prototype. This is simple and scalable. In particular, we represent classes by prototypes (Snell et al., 2017; Rebuffi et al., 2017b) and maintain prototypes with a replay buffer to prevent catastrophic forgetting. We refer to this framework as Prototypical Bayesian Continual Learning, or ProtoCL for short. This approach can be viewed as a probabilistic variant of iCarl (Rebuffi et al., 2017b), which creates embedding functions for different classes which are simply class means and predictions are made by nearest neighbors. ProtoCL also bears similarities to the few-shot learning model Probabilistic Clustering for Online Classification (Harrison et al., 2020) and MetaQDA Zhang et al. (2021b), developed for few-shot image classification. MetaQDA uses Normal-inverse Wishart conjugate priors for Gaussian quadratic discriminant analysis (QDA), ProtoCL has different design choices as follows.

Model. ProtoCL models the generative CL process. We consider classes $j \in \{1, \ldots, J\}$, generated from a categorical distribution with a Dirichlet prior:

$$y_{i,t} \sim \text{Cat}(p_{1:J}), \quad p_{1:J} \sim \text{Dir}(\alpha_t). \tag{5.7.0.1}$$

Images are embedded into an embedding space by an encoder, $\boldsymbol{z} = f(\boldsymbol{x}; \boldsymbol{w})$ with parameters $\boldsymbol{w}$. The per-class embeddings are Gaussian whose with isotropic variance.

The prototype mean has a prior which is also Gaussian:

$$\boldsymbol{z}_{it}|y_{it} \sim \mathcal{N}(\bar{\boldsymbol{z}}_{yt}, \Sigma_\epsilon), \quad \bar{\boldsymbol{z}}_{yt} \sim \mathcal{N}(\boldsymbol{\mu}_{yt}, \Lambda_{yt}^{-1}). \tag{5.7.0.2}$$

See Fig. 5.7 for an overview of the model. To alleviate forgetting in CL, ProtoCL uses a coreset of past task data to continue to embed past classes distinctly as prototypes. The posterior distribution over class probabilities $\{p_j\}_{j=1}^J$ and class embeddings $\{\bar{z}_{y_j}\}_{j=1}^J$ is denoted in short hand as $p(\boldsymbol{\theta})$ with parameters $\eta_t = \{\alpha_t, \boldsymbol{\mu}_{1:J,t}, \Lambda_{1:J,t}^{-1}\}$. We model the Gaussians with a diagonal covariance. ProtoCL models each class prototype but does not use task-specific NN parameters or modules like multi-head VCL. ProtoCL uses a probabilistic model over an embedding space which allows it to use powerful embedding functions $f(\cdot\,; \boldsymbol{w})$ without having to parameterize them probabilistically and so this approach is more scalable than VCL, for instance.

Inference. As the Dirichlet prior is conjugate with the Categorical distribution and likewise the Gaussian over prototypes with a Gaussian prior over the prototype mean, we can calculate posteriors in closed form and update the parameters η_t as new data is observed without using gradient-based updates. We optimize the model by maximizing the posterior predictive distribution and use a softmax over class probabilities to perform predictions. We perform gradient-based learning of the NN embedding function $f(\cdot\,; \boldsymbol{w})$ and update the parameters, η_t at each iteration of gradient descent as well, see Algorithm 1.

Sequential updates. We can obtain our parameter updates for the Dirichlet posterior by Categorical-Dirichlet conjugacy:

$$\alpha_{t+1,j} = \alpha_{t,j} + \sum_{i=1}^{N_t} \mathbb{I}(y_t^i = j), \tag{5.7.0.3}$$

where N_t are the number of points seen during the update at time step t. Also, due to Gaussian-Gaussian conjugacy, the posterior for the Gaussian prototypes is governed by:

$$\Lambda_{y_{t+1}} = \Lambda_{y_t} + N_y \Sigma_\epsilon^{-1} \tag{5.7.0.4}$$

$$\Lambda_{y_{t+1}} \boldsymbol{\mu}_{y_{t+1}} = N_y \Sigma_\epsilon^{-1} \bar{\boldsymbol{z}}_{y_t} + \Lambda_{y_t} \boldsymbol{\mu}_{y_t}, \; \forall y_t \in C_t, \tag{5.7.0.5}$$

Algorithm 1 ProtoCL continual learning

Input: task datasets $\mathcal{T}_{1:T}$, initialize embedding function: $f(\,\cdot\,;\boldsymbol{w})$, coreset: $\mathcal{M} = \emptyset$.
for $\mathcal{T}_1$ **to** $\mathcal{T}_T$ **do**
  for each batch in $\mathcal{T}_i \cup \mathcal{M}$ **do**
    Optimize $f(\cdot;\boldsymbol{w})$ by maximizing the posterior predictive $p(\mathbf{z}, y)$ Eq. (5.7.0.6)
    Obtain posterior over θ by updating η, Eqs. (5.7.0.3) to (5.7.0.5).
  end for
  Add random subset from $\mathcal{T}_i$ to $\mathcal{M}$.
end for

where N_y are the number of samples of class y and $\bar{\boldsymbol{z}}_{y_t} = (1/N_y)\sum_{i=1}^{N_y} z_{yi}$, see Section 10.2.4.2 for the detailed derivation.

Objective. We optimize the posterior predictive distribution of the prototypes and classes:

$$p(\mathbf{z}, y) = \int p(\mathbf{z}, y|\boldsymbol{\theta}_t;\eta_t)p(\boldsymbol{\theta}_t;\eta_t)d\boldsymbol{\theta}_t = p(y)\prod_{i=1}^{N_t}\mathcal{N}(\mathbf{z}_{it}|y_{it};\boldsymbol{\mu}_{y_t,t},\Sigma_\epsilon + \Lambda_{y_t,t}^{-1}). \quad (5.7.0.6)$$

Where the $p(y) = \alpha_y/\sum_{j=1}^{J}\alpha_j$, see Section 10.2.4.3 for the detailed derivation. This objective can then be optimized using gradient-based optimization for learning the prototype embedding function $\boldsymbol{z} = f(\boldsymbol{x};\boldsymbol{w})$.

Predictions. To make a prediction for a test point $\boldsymbol{x}^*$ the class with the maximum (log)-posterior predictive is chosen, where the posterior predictive is:

$$p(y^* = j|\boldsymbol{x}^*, \boldsymbol{x}_{1:t}, y_{1:t}) = p(y^* = j|\boldsymbol{z}^*, \boldsymbol{\theta}_t) = \frac{p(y^* = j, \boldsymbol{z}^*|\boldsymbol{\theta}_t)}{\sum_i p(y = i, \boldsymbol{z}^*|\boldsymbol{\theta}_t)}, \quad (5.7.0.7)$$

see Section 10.2.4.4 for further details.

Preventing forgetting. We use coresets to retain the class prototypes. Coresets are randomly sampled data from previous tasks which are then stored together in a replay buffer and added to the next task training set. At the end of learning a task $\mathcal{T}_t$, we retain a subset $\mathcal{M}_t \subset \mathcal{D}_t$ and augment each new task dataset to ensure that posterior parameters η_t and prototypes are able to retain previous task information. With no coreset the average accuracy over Split-MNIST is 33.25 ± 0.15, Fig. 5.8 dramatically below 93.73 ± 1.05 from Table 5.1. So the main mechanism for preventing forgetting is the replay buffer which enables the network

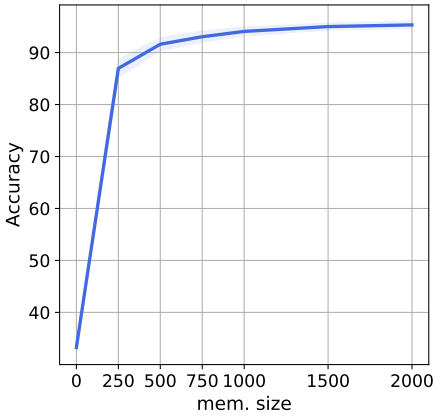


Figure 5.8: Split-MNIST average test accuracy over 5 tasks for different memory sizes. On the x-axis we show the size of the entire memory buffer shared by all 5 tasks. Accuracies are over a mean and standard deviation over 5 different runs with different random seeds.

to maintain a prototype per class, rather than the sequential Bayesian inference in the prototype embedding space similarly to functional Bayesian regularization methods Titsias et al. (2020a).

Class-incremental learning. In this CL setting we do not tell the CL agent which task it is being evaluated on with a task identifier τ. So we cannot use the task identifier to select a specific head to use for classifying a test point, for example. Also, we require the CL agent to identify each class, $\{0, \ldots, 9\}$ for Split-MNIST and Split-CIFAR10 for example, and not just $\{0, 1\}$ as in domain-incremental learning. Class-incremental learning is more general, realistic, and harder a problem setting and thus important to focus on rather than other settings, despite domain-incremental learning also being compatible with sequential Bayesian inference as described in Eq. (5.2.2.5).

Implementation. For Split-MNIST and Split-FMNIST the baselines and ProtoCL all use two-layer NNs with a hidden state size of 200. For Split-CIFAR10 and Split-

Table 5.1: Mean accuracies across all tasks over CL vision benchmarks for *class incremental learning* (Van de Ven and Tolias, 2019). All results are averages and standard errors over 10 seeds. *Uses the predictive entropy to make a decision about which head for class incremental learning.

Method	Coreset	Split-MNIST	Split-FMNIST
VCL (Nguyen et al., 2017)	✗	33.01 ± 0.08	32.77 ± 1.25
+ coreset	✓	52.98 ± 18.56	61.12 ± 16.96
HIBNN* (Kessler et al., 2021a)	✗	85.50 ± 3.20	43.70 ± 20.21
FROMP (Pan et al., 2020)	✓	84.40 ± 0.00	68.54 ± 0.00
S-FSVI (Rudner et al., 2022b)	✓	$\mathbf{92.94 \pm 0.17}$	80.55 ± 0.41
ProtoCL (**ours**)	✓	$\mathbf{93.73 \pm 1.05}$	$\mathbf{82.73 \pm 1.70}$

Table 5.2: Mean accuracies across all tasks over CL vision benchmarks for *class incremental learning* (Van de Ven and Tolias, 2019). All results are averages and standard errors over 10 seeds. *Uses the predictive entropy to make a decision about which head for class incremental learning. Training times have been benchmarked using an Nvidia RTX3090 GPU.

Method	**Training time (sec)** (↓)	**Split CIFAR-10 (acc)** (↑)
FROMP (Pan et al., 2020)	1425 ± 28	48.92 ± 10.86
S-FSVI (Rudner et al., 2022b)	44434 ± 91	50.85 ± 3.87
ProtoCL (**ours**)	$\mathbf{384 \pm 6}$	$\mathbf{55.81 \pm 2.10}$
		Split CIFAR-100 (acc)
S-FSVI (Rudner et al., 2022b)	37355 ± 1135	20.04 ± 2.37
ProtoCL (**ours**)	$\mathbf{1425 \pm 28}$	$\mathbf{23.96 \pm 1.34}$

CIFAR100, the baselines and ProtoCL use a four-layer convolution neural network with two fully connected layers of size 512 similarly to Pan et al. (2020). For ProtoCL and all baselines that rely on replay, we fix the size of the coreset to 200 points per task. For all ProtoCL models, we allow the prior Dirichlet parameters to be learned and set their initial value to 0.7 found by a random search over MNIST with ProtoCL. An important hyperparameter for ProtoCL is the embedding dimension of the Gaussian prototypes for Split-MNIST and Split-FMNIST this was set to 128 while for the larger vision datasets, this was set to 32 found using grid-search.

Results. ProtoCL produces good results on CL benchmarks on par or better than S-FSVI (Rudner et al., 2022b) which is state-of-the-art among Bayesian CL methods while being a lot more efficient to train and without requiring expensive variational inference. ProtoCL can flexibly scale to larger CL vision benchmarks producing better results than S-FSVI. Code to reproduce all experiments can be found here https://github.com/skezle/bayes_cl_posterior. All our experiments are in the more realistic class incremental learning setting, which is a harder setting than those reported in most CL papers, so the results in Table 5.1 are lower for certain baselines than in the respective papers. We use 200 data points per task, see Fig. 10.18 for a sensitivity analysis of the performance over the Split-MNIST benchmark as a function of core size for ProtoCL.

The stated aim of ProtoCL is not to provide a novel state-of-the-art method for CL, but rather to propose a simple baseline that takes an alternative route

than weight-space sequential Bayesian inference. We can achieve strong results that mitigate forgetting, namely by modeling the generative CL process and using sequential Bayesian inference over a few parameters in the class prototype embedding space. We argue that modeling the generative CL process is a fruitful direction for further research rather than attempting sequential Bayesian inference over the weights of a BNN. ProtoCL scales to 10 tasks of Split-CIFAR100 which to the best of our knowledge, is the largest number of tasks and classes which has been considered by previous Bayesian continual learning methods.

5.8 Discussion & Conclusion

In this chapter, we have revisited the use of sequential Bayesian inference for CL. We can use sequential Bayes to recursively build up the multi-task posterior Eq. (5.2.2.5). Previous methods have relied on approximate inference and see little benefit over SGD. We test the hypothesis of whether this poor performance is due to the approximate inference scheme by using HMC in two simple CL problems. HMC asymptotically samples from the true posterior and we use a density estimator over HMC samples to use as a prior for a new task within the HMC sampling process. This density is multi-modal and accurate with respect to the current task but is not able to improve over using an approximate posterior. This demonstrates just how challenging it is to work with BNN weight posteriors. The source of error comes from the density estimation step. We then look at an analytical example of sequential Bayesian inference where we perform exact inference however due to model misspecification, we observe forgetting. The only way to ensure a well-specified model is to assess the multi-task performance over all tasks a priori. This might not be possible in online CL settings. We also introduce the metric of *Bayesian forgetting* which enables us to identify if model misspecification in Bayesian CL models is occurring. The Bayesian forgetting metric can be used along side the traditional forgetting metric from the literature. We then model an analytical example over Bayesian NNs and under certain assumptions show that if there are task data imbalances then this causes forgetting. Because of these results,

we argue against performing weight space sequential Bayesian inference and instead model the generative CL problem. We introduce a simple baseline called ProtoCL. ProtoCL doesn't require complex variational optimization and achieves competitive results to the state-of-the-art in the realistic setting of class incremental learning.

This conclusion should not be a surprise since the latest Bayesian CL papers have all relied upon multi-head architectures or inducing points/coresets to prevent forgetting, rather than better weight-space inference schemes. Our observations are in line with recent theory from (Knoblauch et al., 2020) which states that optimal CL requires perfect memory. Although the results were shown with deterministic NNs the same results follow for BNN with a single set of parameters. Future research directions include enabling coresets of task data to efficiently and accurately approximate the posterior of a BNN to remember previous tasks.

6

Same State Different Task: Continual Reinforcement Learning without Interference

Continual learning considers the problem of training an agent sequentially on a set of tasks while seeking to retain performance on all previous tasks. A key challenge in continual learning is catastrophic forgetting, which arises when performance on a previously mastered task is reduced when learning a new task. While a variety of methods exist to combat forgetting, in some cases tasks are fundamentally incompatible with each other and thus cannot be learned by a single policy. This can occur, in reinforcement learning when an agent may be rewarded for achieving *different goals* from the *same observation*. In this chapter, we formalize this "interference" as distinct from the problem of forgetting. We show that existing continual learning methods based on single neural network predictors with shared replay buffers fail in the presence of interference. Instead, we propose a simple method, OWL, to address this challenge. OWL learns a factorized policy, using *shared* feature extraction layers, but *separate* heads, each specializing on a new task. The separate heads in OWL are used to prevent interference. At test time, we formulate policy selection as a multi-armed bandit problem and show it is possible to select the best policy for an *unknown task* seen during training using the

temporal-difference error as feedback. We find that incorporating uncertainties into the temporal-difference feedback helps the bandit to select the correct policy to solve an environment. We use deep ensembles of action-value functions to add a notion of uncertainty into the bandit algorithm feedback. The use of bandit algorithms allows the OWL agent to constructively re-use different continually learned policies at different times during an episode. We show in multiple reinforcement learning environments that existing replay-based continual learning methods fail, while OWL is able to achieve close to optimal performance when training sequentially.

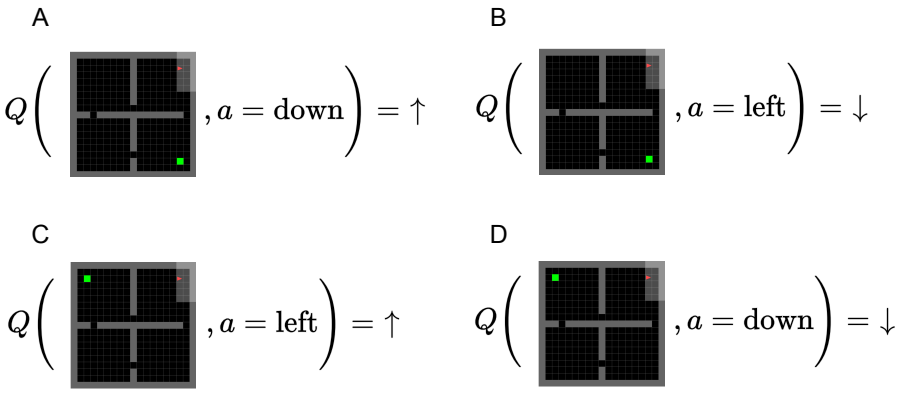


Figure 6.1: A and B a simple four rooms environment task where the goal (the green square) is located at the bottom right. C and D a different four rooms environment where the goal is at a different location, the optimal Q-function should give different values for the same action for the same starting state. The symbol ↑ denotes a high Q-value and ↓ a low Q-value. Task agnostic, single predictor continual RL methods are sub-optimal compared to methods that use the task identifier to condition their policies like OWL.

6.1 Introduction

Reinforcement Learning (RL (Sutton et al., 1998)) considers the problem of an agent taking sequential actions in an environment to maximize some notion of reward. In recent times there has been tremendous success in RL, with impressive results in Games (Silver et al., 2016a) and Robotics (OpenAI et al., 2018), and even real-world settings (Bellemare et al., 2020). However, these successes have predominantly focused on learning a *single* task, with agents often brittle to changes in the dynamics or rewards (or even the seed (Henderson et al., 2018)).

One of the most appealing qualities of RL agents is their ability to continue to learn and thus improve throughout their lifetime. As such, there has recently been an increase in interest in *Continual Reinforcement Learning* (CRL), (Ring, 1994; Khetarpal et al., 2020), a paradigm where agents train on tasks sequentially, while seeking to maintain performance on previously mastered tasks (Kirkpatrick et al., 2016; Schwarz et al., 2018a; Rolnick et al., 2019). A key issue when training on sequential tasks is *catastrophic forgetting*, a consequence of gradient-based learning, as training on a new task overwrites parameters that were important for previous tasks (Robert M. French, 1999). While existing methods address this, they typically only consider the setup where each "task" is an entirely different RL environment, for example, different games from the Arcade Learning Environment (Bellemare et al., 2012) or different simulated robotics environments (Ahn et al., 2019).

In this paper, we focus on a more challenging problem. Instead of distinct environments as different tasks, we consider learning to solve different tasks with the *same state space*. Since these tasks may not only have different but even opposite optimal actions for the same observation, training on them sequentially causes what we call "interference" which can in turn *induce* forgetting, as the agent directly optimizes for an opposing policy. Interference can also lead to reduced performance for future tasks, as the agent cannot fully learn to solve a new task given its retained knowledge of previous opposing objectives. This problem regularly arises in real-world settings, whereby there is a multitude of tasks with reward functions that all share the same state space (e.g. a visual observation of the world). Since previous CRL methods used different environments as different tasks then the agents can learn that the different state spaces correspond to different optimal behaviors and so interference is rarely exhibited.

We begin by showing a simple supervised setting where it is impossible to solve interfering tasks, continually with a single predictor despite using strong CL techniques to prevent forgetting Figure 6.2. This setting can occur in RL where we have different goals but the same observation for different tasks, see Figure 6.1. We, therefore, introduce a new approach to CRL, which we call C**O**ntinual RL

Without Conf**L**ict or OWL. OWL makes use of *shared feature extraction layers*, while acting based on *separate independent policy heads*. The use of the shared layers means that the size of the model scales gracefully with the number of tasks. The separate heads act to model multi-modal objectives and not as a means to retain task-specific knowledge, as they are implicitly used in CL. Task agnostic, single predictor CRL methods that use experience replay (Rolnick et al., 2019) thus suffer from this interference. But have the advantage of not having to infer the task the agent is in to then solve it. Experience replay has been shown to be a strong CL strategy (Balaji et al., 2020). The use of factorized policies is not novel Ammar et al. (2014); Kirkpatrick et al. (2016); Mendez et al. (2020) however their usefulness as a means to prevent interference is parsimonious and has not been studied for CRL.

In the presence of interference, task inference is now necessary and at test time, OWL adaptively selects the policy using a method inspired by multi-armed bandits. This scenario, described in Section 2.5.2, might seem contrived but giving the task identifier at training time and requiring the agent to infer the task it is being evaluated at test time is the same setting as the example of the Home Robot described in Section 1.3.2. We demonstrate in a series of simple yet challenging RL problems that OWL can successfully deal with interference, where existing methods fail. To alleviate forgetting we consider simple weight space (Kirkpatrick et al., 2016) and functional regularizations (Hinton et al., 2015) as these have been repeatedly shown to be effective in CRL (Nekoei et al., 2021).

Our core contribution is to identify a challenging new setting for CRL and propose a simple approach to solving it. As far as we are aware, we are the first to consider a bandit approach for selecting policies for deployment and inferring unknown tasks. The power of this method is further demonstrated in a set of generalization experiments, where OWL is able to solve tasks it has never seen before, up to 6x more successfully than the experience replay baseline (Rolnick et al., 2019).

6.2 Related Work

Continual Learning in Supervised Learning: Continual Learning (CL) is a sequential learning problem. One approach to CL, referred to as *regularization approaches* regularizes an NN predictor's weights to ensure that new learning produces weights that are similar to previous tasks (Kirkpatrick et al., 2016; Nguyen et al., 2017; Zenke et al., 2017b). Alternatively, previous task functions can be regularized to ensure that the functions mapping inputs to outputs are remembered (Li and Hoiem, 2017b). By contrast, *expansion approaches* add new neural resources to enable learning new tasks while preserving components for specific tasks (Rusu et al., 2016b; Lee et al., 2020a). *Memory approaches* replay data from previous tasks when learning the current task. This can be performed with a generative model (Shin et al., 2017). Or small samples from previous tasks (*memories*) (Lopez-Paz and Ranzato, 2017; Aljundi et al., 2019d; Chaudhry et al., 2019). Meta-learning pre-training has been explored to augment continual learning (and *context-dependent targets* can allow interference but is not explored) (Caccia et al., 2020).

Continual Learning in Reinforcement Learning: The CL regularization method EWC has been applied to DQN (Mnih et al., 2015) to learn over a series of Atari games (Kirkpatrick et al., 2016). Both Progressive Networks (Rusu et al., 2016b) and Progress and Compress (Schwarz et al., 2018a) are applied to policy and value function feature extractors for an actor-critic approach. These methods are told when the task changes.

CLEAR leverages experience replay buffers (Lin, 1992) only to prevent forgetting: by using an actor-critic with V-trace importance sampling (Espeholt et al., 2018) of past experiences from the replay buffer catastrophic forgetting can be overcome (Rolnick et al., 2019). CLEAR uses a single predictor NNs and all experience is stored in the replay buffer thus CLEAR is not required to know when the task changes. However, interference is ignored as multi-task performance of all environments is similar to the sum of performances of individual environments. Different selective experience replay strategies can be used for

preserving performance on past tasks (Isele and Cosgun, 2018a). Alternatively, Mendez et al. (2020) learns a policy gradient model which factorizes into task-specific parameters and shared parameters. OWL is more general as it can wrap around any RL algorithm and be used for discrete action spaces and continuous control settings and achieve better results.

A number of previous works have studied transfer in multi-task RL settings where the goals within an environment change (Barreto et al., 2016; Schaul et al., 2015). Our work is related to MetaWorld Yu et al. (2020a) which considers interference between gradients in a multi-task setting. ContinualWorld adapts the tasks from MetaWorld for continual RL Wolczyk et al. (2021) however in the experimental setup no interference is possible since a one-hot encoding of the task is appended to the states.

Interference: this problem is discussed in Rolnick et al. (2019) and has been studied in multi-task (for example (Bishop and Svensén, 2012; Lin et al., 2019)) and meta-learning (for example, (Rajendran et al., 2020)). However, we believe we are the first to consider it in CRL, and that the current state-of-the-art methods lack an approach to tackle it. In particular, we note that existing replay-based methods such as Rolnick et al. (2019) fail to address this issue, as the experience replay buffer contains tuples of the same state-action pairs but different rewards for different tasks. Thus, the agent does not converge, as we show later in our experiments.

6.3 Background

6.3.1 Reinforcement Learning

A Markov Decision Process (MDP, (Bellman, 1957)) is a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$. Here $\mathcal{S}$ and $\mathcal{A}$ are the sets of states and actions respectively, such that for $s_t, s_{t+1} \in \mathcal{S}$ and $a_t \in \mathcal{A}$. $P(s_{t+1}|s_t, a_t)$ is the probability that the system/agent transitions from s_t to s_{t+1} given action a_t and $R(a_t, s_t, s_{t+1})$ is a reward obtained by an agent transitioning from s_t to s_{t+1} via a_t. The discount factor is represented by $\gamma \in (0, 1)$. Actions a_t are chosen using a policy π that maps states to actions: $\mathcal{S} \to \mathcal{A}$. In this chapter, we consider MDPs with finite horizons H. The return from a state is defined as the

sum of discounted future rewards $R_t = \sum_{i=t}^{H} \gamma^{(i-t)} r(s_i, a_i)$. In RL the objective is to maximize $J = \mathbb{E}_{a_i \sim \pi}[R_1 | s_0]$ given an initial state s_0, sampled from the environment.

One approach to maximizing expected return is to learn an action-value function for each state-action pair: $Q^{\pi}(s_t, a_t) = \mathbb{E}_{s_t \sim P, r_t \sim R, a_t \sim \pi}[\sum_t R_t]$. We parameterize the action-value function with a neural network, denoted $Q(s_t, a_t; \boldsymbol{\theta})$, with parameters $\boldsymbol{\theta}$. We optimize $\boldsymbol{\theta}$ to minimize the expected temporal difference error using gradient descent:

$$\mathcal{L}_i(\boldsymbol{\theta}) = \mathbb{E}_{s_t, a_t \sim \rho}[(y_i - Q(s_t, a_t; \boldsymbol{\theta}))^2] \tag{6.3.1.1}$$

We use Q-learning (Watkins and Dayan, 1992), giving $y_i = r_t + \gamma \text{max}_a Q(s_{t+1}, a; \bar{\boldsymbol{\theta}}_i)$, and the parameters $\bar{\boldsymbol{\theta}}$ are target network parameters. The associated update does not require on-policy samples, so learning is off-policy using a replay buffer (Lin, 1992), hence ρ is an empirical distribution that represents samples from the buffer. For continuous action settings, we learn a policy $\pi_{\boldsymbol{\phi}} : \mathcal{S} \rightarrow \mathcal{A}$ using a neural network parameterized by $\boldsymbol{\phi}$. For discrete action settings, our policy follows the maximum Q-value: $\pi \coloneqq \text{argmax}_a Q(s, a)$.

6.3.2 Continual Learning

Continual learning (CL) is a paradigm whereby an agent must learn a set of tasks sequentially while maintaining performance across all tasks. This presents several challenges, in particular avoiding forgetting and efficiently allocating resources for learning new tasks. In CL, the model is shown M tasks $\mathcal{T}_\tau$ sequentially, where $\tau = 1, \ldots M$. A task is defined as $\mathcal{T}_\tau = \left\{p(X), p(Y|X), \tau\right\}$ where X and Y are input and output random variables.

In practice a task is comprised of $i = 1 \ldots N_\tau$ inputs $x_i \in \mathbb{R}^d$ and outputs $y_i \in \mathbb{R}$ ($\mathbb{N} \subset \mathbb{R}$). The model loses access to the training dataset for task $\mathcal{T}_\tau$, and it will be continually evaluated on all previous tasks $\mathcal{T}_j$ for $j \leq \tau$. τ can be used as a task identifier informing the agent when to start training on a new task. For a comprehensive review of CL scenarios see (van de Ven and Tolias, 2018). For RL the definition of a task is simply an MDP and task identifier:

$\mathcal{T}_\tau = \left\{\mathcal{S}_\tau, \mathcal{A}_\tau, p_\tau(s_1), p_\tau(s_{t+1}|s_t, a_t), R_\tau(a_t, s_t, s_{t+1}), \tau\right\}$, and so the agent will no longer be able to interact with previous environments, but must ensure that it can remember how to solve all past tasks/environments.

6.4 Catastrophic Forgetting vs. Interference

Forgetting occurs when performance on old tasks is reduced while learning new tasks. On the other hand, interference occurs when two or more tasks are incompatible for the same model. We re-use these definitions from CLEAR (Rolnick et al., 2019). We observe this when the multi-task objectives are multi-modal and tasks share the same observation space but have different goals/objectives.

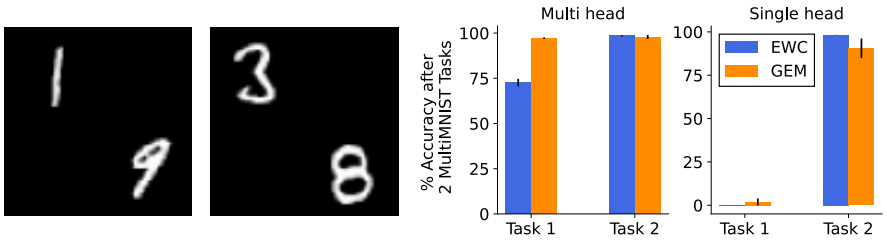


Figure 6.2: Left, two samples from the MultiMNIST dataset (Sabour et al., 2017). The first task $\mathcal{T}_1$ requires classifying the top left digit and the second task $\mathcal{T}_2$ requires classifying the bottom right. Right, results from continually learning on both MultiMNIST tasks. We compare two different CL methods EWC (Kirkpatrick et al., 2016) and GEM (Lopez-Paz and Ranzato, 2017). Single-head task agnostic setups suffer from catastrophic forgetting of the first task due to interference. The multi-head setup is a simple remedy to the problem of interference.

We demonstrate interference using the MultiMNIST dataset (Sabour et al., 2017) Fig. 6.2. Each image is composed of two different MNIST digits and for $\mathcal{T}_1$ we are required to classify the top image and in $\mathcal{T}_2$ we are required to classify the bottom image. For both of these tasks, the only difference is the objective. When we perform CL with a single predictor network or single-headed network with different CL strategies to alleviate forgetting we see that the interference between tasks causes almost 100% forgetting of the first task, despite using established CL strategies. On the other hand, using multi-headed networks allows us to model both objectives in MultiMNIST. We get interference in the following.

Observation 6.4.1. *Consider two tasks $\mathcal{T}_i$ and $\mathcal{T}_j$. Let both tasks' input distributions $p_k(X)$ share the same support but have different conditional distributions $p_k(Y|X) = \mathcal{N}(f^k(X), \beta^{-1})$, where f^k is a mean function with $f^i \neq f^j$ and β^{-1} is data noise. Then the multi-task distribution is bi-modal and using a Gaussian likelihood will result in interference.*

This may seem contrived in the supervised setting, however, it is common throughout reinforcement learning. Consider a partially observable MDP (POMDP) where we receive an initial observation but do not know the goal location or reward function then an agent might require different policies for each task. We see an example of this in Fig. 6.1 where in one task the goal is in the room below the agent and in the other the goal is in the room to the left. The most efficient policies guide the agent in different directions depending on the task the agent is in. This observation has important consequences: methods that are task agnostic and do not condition on the task or do not use task-specific parameters are susceptible to interference. Some CL methods use a single predictor and aim to approximate the multi-task setting by using storing samples in a buffer (Aljundi et al., 2019a; Rolnick et al., 2019). Furthermore, single-headed networks are often used as more difficult CL scenarios when studying methods to mitigate forgetting (van de Ven and Tolias, 2018; Farquhar and Gal, 2018).

Our solution is to model the multi-modality by learning a mixture of linear regressions (see: §14.5.1 (Bishop, 2006)). The same applies to CRL: the Q-function needs to have separate weights for each task or needs to condition the task to solve CRL environments with interference. In practice, we use multi-headed network predictors. *Whereas these are commonly employed to preserve previous task knowledge and prevent forgetting in CL, we are employing them as a means to prevent interference.* So our motivation for the use of multi-head networks is wholly different. Most supervised CL settings are benchmarked on vision tasks that use different distinct classes as tasks. Thus $p_\tau(X)$ and $p_\tau(Y|X)$ both change with task τ; a single NN predictor can model this. However in RL only the reward function, $R_\tau(s_t, a_t, s_{t+1})$, need change with task τ.

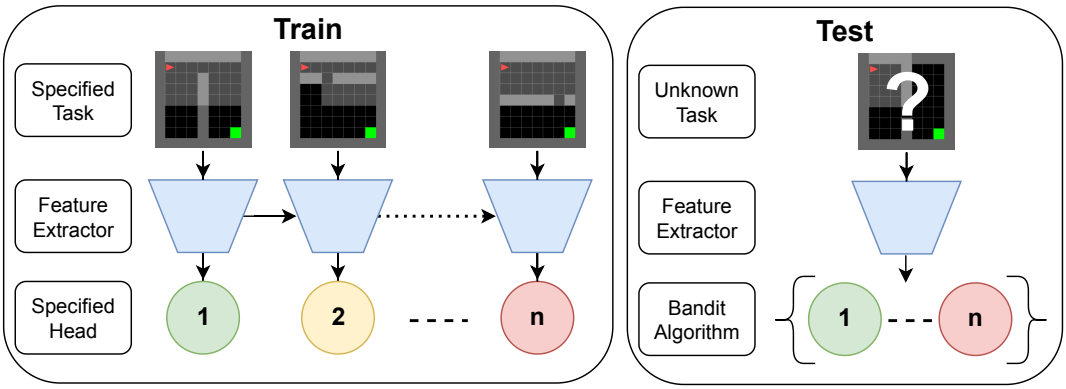


Figure 6.3: An overview of our approach. On the left, we show the training setup. When training sequentially on different tasks we use the same feature extractor (blue), regularized with EWC, but maintain a set of distinct heads (green, yellow, ..., pink) corresponding to different tasks. On the right, we show the test time adaptation. Note that the task is not known, and must be inferred through interaction with the environment.

6.5 Continual RL without Conflict

At a high level, OWL uses an off-policy RL algorithm to train a Q-function across tasks sequentially. To prevent interference our key insight is that: 1.) we can use a single network with a shared feature extractor but multiple heads, parameterized by linear layers to fit individual tasks; 2.) we flush the experience replay buffer when starting to learn in a new task. At test time, we frame policy/head selection as a multi-armed bandit problem, to adaptively select the best policy. In this section we provide additional details on each component, describing first the structure of the Q-function, before moving to our test time adaptation.

6.5.1 Factorized Q-Functions

Multi-head networks are commonly used in CL (Li and Hoiem, 2017b; Nguyen et al., 2017), and they enable learning task-specific output mapping with a shared feature extractor. Multi-head networks are effective in that allow learning of task-specific parameters that can be recalled and so help to alleviate forgetting. Single-head networks are commonly used as a more difficult baseline for CL benchmarks (Farquhar and Gal, 2018; van de Ven and Tolias, 2018). In our work, multi-head networks are used as they prevent interference.

Algorithm 2 OWL: Training

Input: Tasks $\mathcal{T} = \{\mathcal{T}_i\}_{i=1}^{M}$.
Initialize: $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$, $\Omega^Q = \Omega^\pi = \emptyset$.
for $t = 1, 2, \ldots, M$ **do**
  See Task $\mathcal{T}_t$
  Train Q-function with parameters $\{\boldsymbol{\theta}_z, \boldsymbol{\theta}_i\}$ and regularization Ω^Q.
  if $\mathcal{A}$ is continuous **then**
    Train policy with parameters $\{\boldsymbol{\phi}_z, \boldsymbol{\phi}_i\}$ with regularization Ω^π.
  end if
  Calculate Q-function EWC regularization and $\Omega^Q := \{\mathcal{L}^Q_{\text{EWC}}, \Omega^Q\}$.
  if $\mathcal{A}$ is continuous **then**
    Calculate policy EWC regularization and $\Omega^\pi := \{\mathcal{L}^\pi_{\text{EWC}}, \Omega^\pi\}$.
  end if
  Empty the experience reply buffer $\mathcal{D} = \emptyset$.
  Evaluate according to Algorithm Algorithm 3.
end for

Alleviating forgetting: We represent a factorized Q-function as having parameters $\theta = \{\boldsymbol{\theta}_z, \boldsymbol{\theta}_{1:M}\}$, where θ_z are feature extractor parameters and $\boldsymbol{\theta}_{1:M}$ the heads. For discrete problems, one can follow the maximum Q-values to obtain the next action. For parameterized policies, we can equally construct our policy similarly with parameters $\phi = \{\boldsymbol{\phi}_z, \boldsymbol{\phi}_{1:M}\}$ where ϕ_z are the neural network feature extraction layers, and $\phi_{1:M}$ are linear policy heads. To address forgetting in the shared neural network feature extractors we use regularization methods. In particular, we found EWC to work well and is a simple approach to prevent forgetting (Kirkpatrick et al., 2017), (we also tried a functional regularization (Hinton et al., 2015; Li and Hoiem, 2017b) but found it underperformed; see Section 6.6.3). We train our agent according to Algorithm 2. As we see more and more tasks new heads can easily be added so we do not need to prespecify the number of tasks or policy heads $M \in \{1, \ldots, \infty\}$.

6.5.2 Selecting Policies as a Multi-Armed Bandit Problem

At test time we do not tell OWL which task it is being evaluated on. We consider the set of arms M to be the set of policies that can be chosen to act at each timestep of the test task. The aim is to find the policy which achieves the highest reward on a given test task. We use a modified version of the Exponentially Weighted Average

Forecaster algorithm (Cesa-Bianchi and Lugosi, 2006), as has been shown to be successful in adapting components of RL algorithms (Ball et al., 2020). In this setup, we consider M experts making recommendations at the beginning of each round. After sampling a decision $i_t \in \{1, \cdots, M\}$ from a distribution $\mathbf{p}^t \in \Delta_M$ with the form $\mathbf{p}^t(i) \propto \exp(\ell_t(i))$ the learner experiences a loss $l_{i_t}^t \in \mathbb{R}$. The distribution $\mathbf{p}^t$ is updated by changing ℓ_t as follows:

$$\ell_{t+1}(i) = \begin{cases} \ell_t(i) + \eta \frac{l_i^t}{\mathbf{p}^t(i)} & \text{if } i = i_t \\ \ell_t(i) & \text{otherwise,} \end{cases} \tag{6.5.2.1}$$

for some step size parameter η. We consider the case where the selection of $\boldsymbol{\phi}_i$ is thought of as choosing among M experts which we identify as the different policies $\{\boldsymbol{\phi}_i\}_{i=1}^M$, trained on the corresponding Q-functions $\{\boldsymbol{\theta}_i\}_{i=1}^M$. The loss we consider is of the form $l_{i_t} = 1/\hat{G}_{\boldsymbol{\phi}_i}(\boldsymbol{\theta}_i)$, where $G_{\boldsymbol{\phi}_i}(\theta_i)$ is the TD error or the log-likelihood of the observed reward from the test task, r_t given the predicted Q-values. If required, we can then perform a normalization of G, hence $\hat{G}$. Henceforth we denote by $\mathbf{p}_{\boldsymbol{\phi}}^t$ the exponential weights distribution over $\boldsymbol{\phi}$ values at time t. The pseudocode for our test-time procedure is shown in Algorithm 3.

Algorithm 3 OWL: Testing

1: **Input:** tasks seen so far $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_\tau\}$, Q-functions $\{\boldsymbol{\phi}_i\}_{i=1}^M$, step size η, maximum number of timesteps T.
2: **Initialize:** $\mathbf{p}_{\boldsymbol{\phi}}^1$ as a uniform distribution, s_1 as the initial state of the test task.
3: **for** $\mathcal{T}_j \in \mathcal{T}$ **do**
4: **for** $t = 1, \ldots, T-1$ **do**
5: Select $i_t \sim \mathbf{p}_{\boldsymbol{\phi}}^t$, and set $\pi_{\text{test}} = \pi_{\boldsymbol{\phi}_{i_t}}$.
6: Take action $a_t \sim \pi_{\text{test}}(s_t)$, and receive reward r_t and the next state s_{t+1} from $\mathcal{T}_j$.
7: Use Eq. (6.5.2.1) to update $\mathbf{p}_{\boldsymbol{\phi}}^t$ with $l_{i_t}^t = \hat{G}_{\boldsymbol{\phi}_t}(\boldsymbol{\theta}_{t+1})$
8: **end for**
9: **end for**

6.6 Experiments

To test our approach, we consider challenging CRL problems where tasks have similar or identical state and action spaces but distinct goals/rewards. Our main

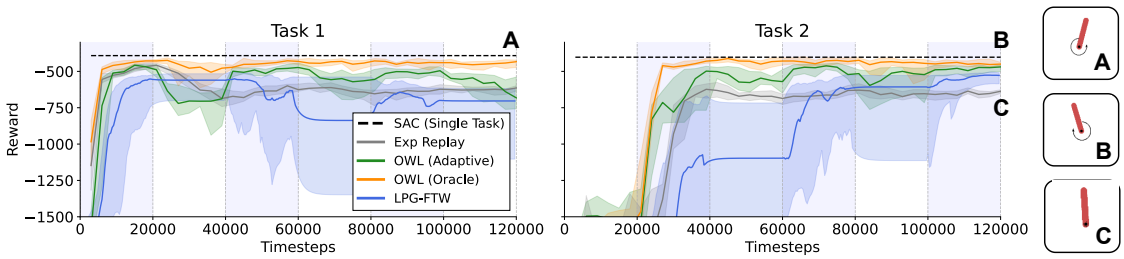


Figure 6.4: Median performance (with inter-quartile range) across 10 seeds. Pale blue shaded regions correspond to the timesteps when the task in question is being trained on. A, Task 1 arm position from the OWL agent (oracle) B, Task 2 arm position from the OWL agent (oracle) C, Optimal arm position for Exp Replay for Tasks 1 and 2 (note that the reward function also has angular velocity terms thus the OWL/SAC agent isn't able to place the arms exactly at $\pm 90°$ without obtaining a sub-optimal reward). Exp Replay clearly displays interference.

hypothesis is that these tasks cannot be solved continually with a single policy, using a shared replay buffer. Our primary baseline, which we call Experience Replay (Exp Replay in figures) corresponds to this case (Rolnick et al., 2019) and has been shown to be a very effective baseline in CL (Balaji et al., 2020). In each setting, we test two versions of our algorithm, which we refer to as Oracle and Adaptive. With the Oracle, the OWL agent is told at test time which task is being evaluated. Finally, we consider the multi-arm bandit (MAB) approach, denoted Adaptive. Code is available at https://github.com/skezle/owl.

6.6.1 Pendulums with Interfering Goals

The first setting we consider is a simple yet challenging take on the well-known Pendulum-v0 environment (Brockman et al., 2016). Typically, the policy is rewarded for placing the pendulum at 0°. Instead, we amend the reward function to produce two interfering tasks, with optimal positions: $\{+90°, -90°\}$. We have continuous actions and train a multi-head policy and Q-function using Soft Actor-Critic (Haarnoja et al., 2018c). We train on each task three times, switching every $20,000$ environment steps. For more details on our implementation see Section 10.3.3.1. Results are shown in Fig. 6.4.

First, we see evidence confirming our first hypothesis that training with a shared replay buffer over all tasks leads to suboptimal performance on both tasks due to

the interfering nature of the tasks (Exp Replay). The Exp Replay agent (grey) learns to place the pendulum at 0°. As we see on the bottom right, this balances the conflicting goals but is suboptimal for both individual tasks (see Fig. 6.4C). Secondly, the Oracle version of OWL (orange), which knows the task under evaluation at test time, performs well since two separate policies are trained on the individual tasks (black, dashed) and display minimal forgetting. Encouragingly we can almost achieve this same performance without informing the agent of the task index, using our adaptive mechanism (green). Our method also outperforms LPG-FTW (Mendez et al., 2020) which uses 2.5x more gradient steps as our method builds on top of SAC which yields state-of-the-art results in continuous control. Regarding feedback to the algorithm, we explore the importance of the probabilistic networks Section 10.3.4.

6.6.2 MiniGrid Environments

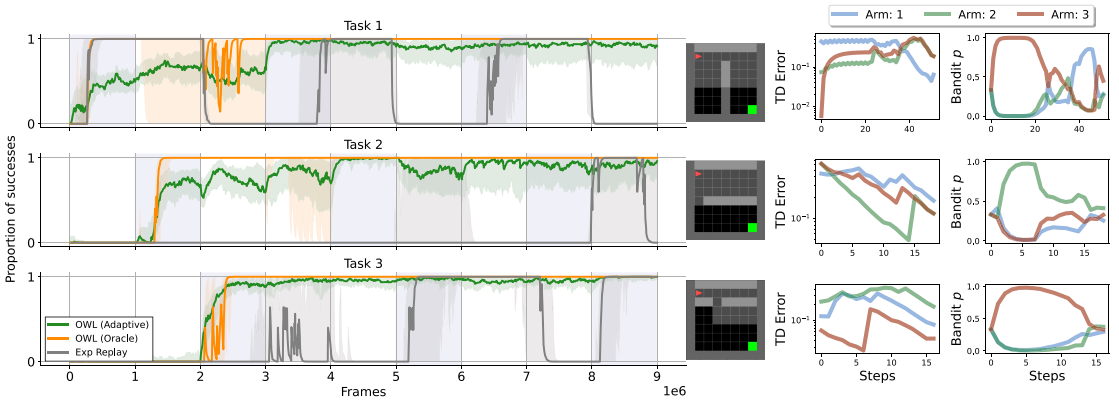


Figure 6.5: Left, Median performance across 10 seeds for three different MiniGrid environments trained continually. Shaded envelopes correspond to the interquartile range. Shaded pale blue regions correspond to the current training task. OWL Adaptive and Oracle are able to prevent forgetting and interference while Exp. Replay fails. Right, Bandit arm probabilities over the course of a roll-out to demonstrate how the TD error feedback is used to select the right arm/policy to solve the task (note these have been smoothed for visualization purposes).

MiniGrid is a challenging set of procedurally generated maze environments (Chevalier-Boisvert et al., 2018). Each environment is partially observable, with the agent only "seeing" a small region of visual input out of a larger state. Additionally, each state is an image, and rewards are sparse (the agent only receives a reward for navigating

to the green tile in Fig. 6.3), which makes it harder for agents to find learning signals. We use the SimpleCrossing environment, which has a single wall and gap. The environment seed corresponds to different wall positions, orientations, and gaps. The initial observation can look identical (or very similar) for two different environments, and the agent has to explore to discover the wall location and door position.

We employ DQN to handle the discrete action space (Mnih et al., 2015), see Section 10.3.3.2 for implementation details. We train the same methods as the previous experiment on three distinct MiniGrid grid worlds continually, repeating every three times for 1M steps. We use the TD error in Eq. (6.3.1.1) as feedback to the MAB. OWL (Oracle) is able to consistently solve all environments after one round Fig. 6.5. OWL (Adaptive) can dynamically select the correct policy most of the time after seeing each task once and continuously improves, with the final performance almost matching OWL (oracle)[1]. Exp Replay exhibits significant interference between tasks.

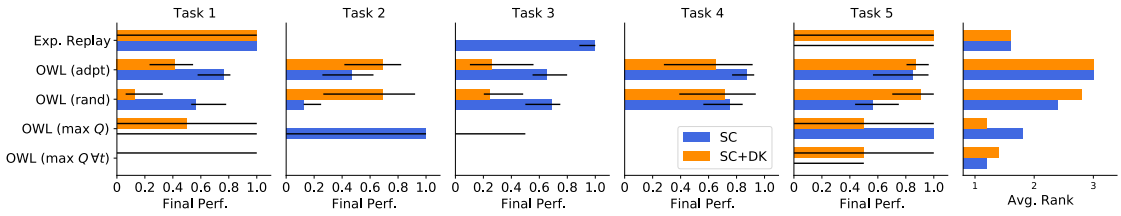


Figure 6.6: Final performance for different OWL policy selection strategies and Exp Replay.

Scaling to more tasks. We now scale to 5 SimpleCrossing tasks (denoted as SC in plots) and another set of 5 tasks with 3 SimpleCrossing and 2 DoorKey environments (denoted as SC+DK). The set of tasks is repeated 3 times each task is seen for 0.75M environment steps. For Exp Replay we adjust the buffer size to 4M to ensure that data from all tasks are in the buffer over the course of training. We note that Exp. Replay again suffers from interference while OWL can overcome it (and overcome forgetting) see Fig. 10.23 and Fig. 10.24 in the appendix.

We explore different arm selection strategies in Fig. 6.6. We compare the multi-armed bandit (MAB) versus random policy head selection at each step of the

[1]For videos of OWL in action, see: https://sites.google.com/view/crlwithoutconflict/

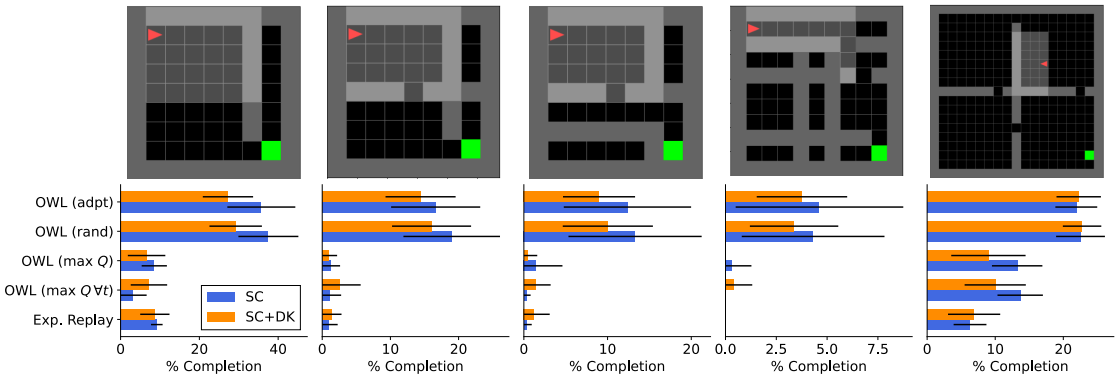


Figure 6.7: Mean and std error proportion of successes for 100 different environments which have not been seen during training for 10 seeds. OWL is able to generalize to unseen environments while Exp Replay fails.

roll-out (OWL (rand)), versus selecting the policy head with the largest expected reward (OWL (max Q)) and versus selecting the policy with the largest expected reward at each step of the roll-out (max Q $\forall t$). We see that approaches that use the Q-value to select the policy fail, as does Exp Replay. Random policy selection performs well, but the MAB performs significantly better than random policy selection, a t-test with unequal variances has a p-value of 0.06 for the SC policies, and there isn't a significant difference for the SC+DK policies. Thus there are statistically significant benefits to using the MAB. The MAB approach can also behave similarly to a random policy head selection and can perform well (Bergstra and Bengio, 2012; Mania et al., 2018).

Generalization Results. Many works have focused on the generalization properties of RL agents in the MiniGrid environment (Goyal et al., 2020), training on hundreds of levels. Instead, we train on just 5 levels sequentially, which produces a significant risk of overfitting. We take the final SC and SC+DK policies and evaluate them on 100 different, unseen tasks Fig. 6.7. We find that our OWL agent is able to transfer effectively to these unseen tasks, solving up to 40% of unseen levels single-walled levels, around 4× more than Exp Replay. OWL can even solve harder environments where Exp Replay totally fails. This exciting result demonstrates that the OWL agent is able to re-use each of the base policies learned sequentially for solving totally different tasks. Randomly selecting policies in the roll-out is a

	SC	SC+DK
Exp Replay	0.01 (0.61, 0.00)	0.00 (0.52, 0.00)
OWL (orcl)	**0.85** (0.97, 0.72)	**0.60 (0.98, 0.44)**
OWL (adpt)	0.59 (0.75, 0.48)	**0.63 (0.79, 0.45)**
OWL - EWC (orcl)	0.45 (0.53, 0.39)	0.40 (0.48, 0.30)
OWL - EWC (adpt)	0.49 (0.60, 0.39)	**0.50 (0.62, 0.37)**
OWL - EWC + DL (orcl)	0.45 (0.53, 0.36)	0.34 (0.40, 0.29)
OWL - EWC + DL (bndt)	0.53 (0.61, 0.38)	0.39 (0.45, 0.33)
Full Rehearsal	0.99 (0.99, 0.97)	0.99 (1.00, 0.98)

Table 6.1: Comparisons and ablations for OWL evaluating on the 5 SC and SC+DK tasks for 10 seeds.

strong strategy that the MAB can emulate. This demonstrates the potential for our approach in a hierarchical RL setting, with links to options (Bacon et al., 2017).

6.6.3 Ablations

OWL decreases in performance when we remove the EWC Table 6.1. Replacing EWC with a distillation regularization which ensures that the outputs from the previous task's Q-function remain similar to the previous task's Q-function for the current task (Hinton et al., 2015; Li and Hoiem, 2017b), also decreases performance, see Section 10.3.4.2 for more details. Distillation regularization works well for classification problems, however, we are performing regression, which could explain the drop in performance compared to EWC. We also compare to a Full Rehearsal (FR) which is an upper bound to OWL performance. FR has a buffer for each task and a separate policy head for each task, as such it does not scale gracefully as the number of tasks increases in comparison to OWL, see Section 10.3.7 in the appendix for implementation details.

6.7 Conclusion and Future Work

We consider a challenging CRL setting where *different tasks* have the *same observation*. We showed that established experience replay methods which are task agnostic with a single predictor network fail due to interference. Our main contribution is to highlight this interference problem and introduce a simple yet effective approach

for this paradigm, which we call OWL. OWL is able to limit forgetting while training on tasks sequentially by using a Q-function with a shared feature extractor and a population of linear heads for each task. OWL does not require knowledge of the task at test time but is still able to achieve close to optimal performance using a multi-armed bandits algorithm. We evaluated OWL on challenging RL environments such as MiniGrid, where we were able to solve five different tasks with similar observations. Finally, we showed it is possible to transfer our learned policies to unseen and more difficult environments.

There are a variety of exciting future directions for this work. For instance, exploring change detection methods using more robust probabilistic models in RL, to detect shifts in reward and state-action distributions.

In this work, we study the use of memoryless policies to solve POMDPs, where partial observability can be attributed in part to the observation not having a task-id at test time. The use of an MAB adds memory at test time to enable adaptation to a specific task. Another way to solve this test-time adaptation which is an interesting direction for future work is to use an explicit form of memory in the policy such as an RNN and to study the test-time adaptation of the RNN policy.

7

The Effectiveness of World Models for Continual Reinforcement Learning

World models power some of the most efficient reinforcement learning algorithms. In this work, we showcase that they can be harnessed for continual learning – a situation when the agent faces changing environments. World models typically employ a replay buffer for training, which can be naturally extended to continual learning. We systematically study how different selective experience replay methods affect performance, forgetting, and transfer. We also provide recommendations regarding various modeling options for using world models. The best set of choices is called Continual-Dreamer, it is task-agnostic and utilizes the world model for continual exploration. We use the uncertainty in the next state predictions as an additional intrinsic reward to continually explore without any additional supervision. Continual-Dreamer is sample efficient and outperforms state-of-the-art task-agnostic continual reinforcement learning methods on Minigrid and Minihack continual reinforcement learning benchmarks.

7.1 Introduction

There have been many recent successes in reinforcement learning (RL), such as in games (Schrittwieser et al., 2019), robotics (OpenAI et al., 2018) and in scientific applications (Nguyen et al., 2021; Degrave et al., 2022). However, these successes showcase methods for solving individual tasks. Looking beyond, the field of continual reinforcement learning (CRL) aims to develop agents that can solve many tasks, one after another, continually while retaining performance on all previously seen ones (Khetarpal et al., 2020). Such capabilities are conjectured to be essential for truly scalable intelligent systems, Ring (1994) and Hassabis et al. (2017) conjecture that truly scalable intelligent systems will additionally need to master many tasks in a continual manner. The field of continual reinforcement learning (CRL) aims to develop agents which can solve many tasks, one after another, continually while retaining performance on all previously seen ones (Khetarpal et al., 2020).

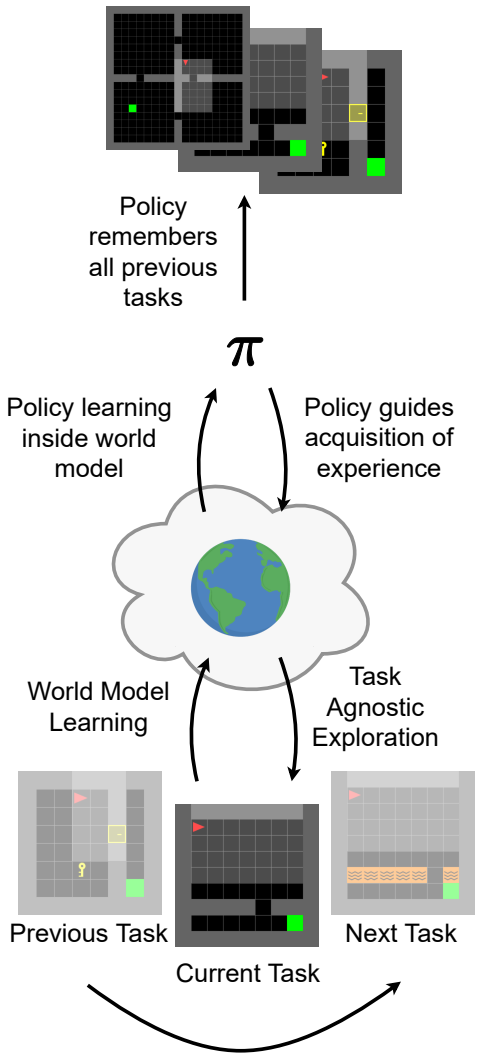


Figure 7.1: CRL with world models.

World models combine generative models with RL and have become one of the most successful paradigms in single-task RL (Ha and Schmidhuber, 2018). World models are typically trained iteratively, first, the policy interacts with the environment, second, the collected experience is used to train the world model and then the policy is trained using hallucinated experience generated by the world model (Kaiser et al., 2019). Such approaches are typically sample efficient,

offloading the most data-hungry trial-and-error RL training to the imagined world. Further, generative models can be used to create compact representations that facilitate policy training and so obtain very good results for challenging pixel-based environments Hafner et al. (2023).

This chapter explores using world models for learning tasks sequentially. We showcase a method that satisfies the traditional CL desiderata: avoids catastrophic forgetting, achieves transfer, high average performance, and is scalable. The proposed method *Continual-Dreamer* is built on top of DreamerV2 (Hafner et al., 2020) – an RL algorithm, which achieves state-of-the-art on a number of benchmarks. We define the Continual-Dreamer configuration in Section 7.6. DreamerV2 uses a replay buffer (Lin, 1992), which we extend across many tasks to mitigate forgetting akin to (Isele and Cosgun, 2018b; Rolnick et al., 2019). Additionally, we demonstrate that world models are capable of operating without explicit task identification. This is an important requirement in many CRL scenarios and enables further capabilities. In particular, we implement an adaptive exploration method similar to (Steinparz et al., 2022), which implicitly adapts to task changes. Based on our empirical results we argue that world-models offer a potent approach for CRL and should attract more research attention.

Our contributions are as follows:

- We present the first approach to task-agnostic model-based CRL. We use DreamerV2 as a backbone, and our work is transferable to other world models.
- We evaluate our method performance on two challenging CRL benchmarks, Minigrid and Minihack, and show that it outperforms state-of-the-art task-agnostic methods, demonstrating that the model-based paradigm is a viable solution to CRL.
- We thoroughly explore different experience replay strategies, which address how we sample from or populate the experience replay buffer to balance preventing forgetting of previously learned skills while enabling learning new skills from new tasks.

7.2 Preliminaries

7.2.1 Reinforcement Learning

A Partially Observable Markov Decision Process (POMDP (Kaelbling et al., 1998)) is the following tuple $(\mathcal{S}, \mathcal{A}, P, R, \Omega, \mathcal{O}, \gamma)$. Here, $\mathcal{S}$ and $\mathcal{A}$ are the sets of states and actions respectively, such that for $s_t, s_{t+1} \in \mathcal{S}$ and $a_t \in \mathcal{A}$. $P(s_{t+1}|s_t, a_t)$ is the transition distribution and $R(a_t, s_t, s_{t+1})$ is the reward function. Additionally, $\gamma \in (0, 1)$ is the discount factor. Since the environments we consider are partially observable, the agent does not have access to the environment state $s \in \mathcal{S}$, but only the observations $o \in \Omega$, where Ω is the set of observations and $\mathcal{O} : \mathcal{S} \times \mathcal{A} \to P(\Omega)$ is an observation function that defines a distribution over observations. Actions a_t are chosen using a policy π that maps observations to actions: $\Omega \to \mathcal{A}$. For the purposes of this introduction, let us assume we have access to the states s_t and we are working with a finite horizon H. Then the return from a state is $R_t = \sum_{i=t}^{H} \gamma^{(i-t)} r(s_i, a_i)$. In RL the objective is to maximize the expected return $J = \mathbb{E}_{a_i \sim \pi, s_0 \sim \rho}[R_1|s_0]$ where $s_0 \sim \rho(s_0)$ and $\rho(\cdot)$ is the initial state distribution.

One approach to maximizing expected return is to use a *model-free* approach, to learn a policy $\pi_{\boldsymbol{\phi}} : \mathcal{S} \to \mathcal{A}$ with a parametric model such as a neural network with parameters $\boldsymbol{\phi}$ guided by an action-value function $Q_{\boldsymbol{\theta}}(s_t, a_t)$ with parameters $\boldsymbol{\theta}$. Alternatively, instead of learning a policy directly from experience we can employ *model-based* RL (MBRL) and learn an intermediate model f, for instance, a transition model $s_{t+1} = f(s_t, a_t)$ from experience and learn our policy with additional experience generated from the model f (Sutton, 1991). Instead of working with the actual state s_t, our methods consider the observations o_t, we employ recurrent policies, action-value functions, and models to help better estimate states s_t (Hausknecht and Stone, 2015).

7.2.2 Continual Supervised Learning

Continual Learning (CL) is a setting whereby a model must master a set of tasks sequentially while maintaining performance across all previously learned tasks.

Other important objectives are to develop scalable CL methods that are able to transfer knowledge from previous tasks to aid the learning of new tasks, known as forward transfer. Traditionally, in supervised CL, the model is sequentially shown T tasks, denoted $\mathcal{T}_\tau$ for $\tau = 1, \ldots, T$. Each task, $\mathcal{T}_\tau$, is comprised of a dataset $\mathcal{D}_\tau = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N_\tau}$ which a neural network is required to learn from. More generally, a task is denoted by a tuple comprised of the conditional and marginal distributions, $\{p_\tau(y|\mathbf{x}), p_\tau(\mathbf{x})\}$. After task τ, the model loses access to the training dataset for $\mathcal{T}_\tau$, however, its performance is continually evaluated on all tasks $\mathcal{T}_i$ for $i \leq \tau$. For a comprehensive review of CL scenarios see (Hsu et al., 2018a; Van de Ven and Tolias, 2019).

7.2.3 Continual Reinforcement Learning

In continual RL the agent has a budget of N interactions with each task environment $\mathcal{T}_\tau$ The agent is then required to learn a policy to maximize rewards in this environment, before interacting with a new environment and having to learn a new policy. Each task is defined as a new POMDP, $\mathcal{T}_\tau = \left(\mathcal{S}_\tau, \mathcal{A}_\tau, P_\tau, R_\tau, \Omega_\tau, \mathcal{O}_\tau, \gamma_\tau\right)$.

The agent is continually evaluated on all past and present tasks so it is desirable for the agent's policy to transfer to new tasks while not forgetting how to perform past tasks. CRL is not a new problem setting (Thrun and Mitchell, 1995a). Its definition has evolved and some settings differ from paper to paper, we employ the setting above which is related to previous recent work in CRL (Kirkpatrick et al., 2016; Schwarz et al., 2018a; Rolnick et al., 2019; Wolczyk et al., 2021; Kessler et al., 2021b; Powers et al., 2021; Caccia et al., 2022).

7.3 Related Work

7.3.1 Continual Supervised Learning

Here, we briefly describe CL methods. One approach referred to as *regularization methods* regularizes an NN's weights to ensure that optimizing for a new task finds a solution that is "close" to the previous task's (Kirkpatrick et al., 2016; Nguyen et al., 2017; Zenke et al., 2017b). Working with functions can be easier than with NN

weights and so task functions can be regularized to ensure that learning new function mappings are "close" across tasks (Li and Hoiem, 2017b; Benjamin et al., 2019; Buzzega et al., 2020). By contrast, *expansion methods* add new NN components to enable learning new tasks while preserving components for specific tasks (Rusu et al., 2016b; Lee et al., 2020c). *Memory methods* replay data from previous tasks when learning the current task. This can be performed with a generative model (Shin et al., 2017). Or samples from previous tasks (*memories*) (Lopez-Paz and Ranzato, 2017; Aljundi et al., 2019d; Chaudhry et al., 2019).

7.3.2 Continual Reinforcement Learning

Seminal work in CRL, EWC (Kirkpatrick et al., 2016) enables DQN (Mnih et al., 2015) to continually learn to play different Atari games with limited forgetting. EWC learns new Q-functions by regularizing the parameter-weighted L2 distance between the new task's current weights and the previous task's optimal weights. EWC requires additional supervision informing it of task changes to update its objective, select a specific Q-function head, and select a task-specific ϵ-greedy exploration schedule. Progress and Compress (Schwarz et al., 2018a) applies a regularization to policy and value function feature extractors for an actor-critic approach. Alternatively, LPG-FTW (Mendez et al., 2020) learns an actor-critic that factorizes into task-specific parameters and shared parameters. Both methods require task supervision and make use of task-specific parameters and shared parameters. Task-agnostic methods like CLEAR (Rolnick et al., 2019) do not require task information to perform CRL. CLEAR leverages experience replay buffers (Lin, 1992) to prevent forgetting: by using an actor-critic with V-trace importance sampling (Espeholt et al., 2018) of past experiences from the replay buffer. Model-based RL approaches to CRL have been demonstrated where the model weights are generated from a hypernetwork which itself is conditioned by a task embedding (Huang et al., 2021). Recent work demonstrates that recurrent policies for POMDPs can obtain good overall performance on continuous control CRL benchmarks (Caccia et al., 2022). Another task-aware solution is to expand

a subspace of policies, the number of policies scaling sublinearly with the number of tasks (Gaya et al., 2022). Of the related works presented, only CLEAR is task-agnostic and so is the primary baseline under consideration when comparing task-agnostic world model approaches to CRL.

A number of previous works have studied transfer in multi-task RL settings where the goals within an environment change (Barreto et al., 2017; Schaul et al., 2015; Barreto et al., 2019). In particular, incorporating the task definition directly into the value function (Schaul et al., 2015) and combining this with off-policy learning allows a CRL agent to solve multiple tasks continually, and generalize to new goals (Mankowitz et al., 2018).

7.3.3 Continual Adaptation

Instead of focusing on remembering how to perform all past tasks, another line of research investigates quick adaptation to changes in the environment. This can be captured by using a latent variable and off-policy RL (Xie et al., 2020). Alternatively, one can meta-learn a model such that it can then adapt quickly to new changes in the environment (Nagabandi et al., 2018). All these works use small environment changes such as modification of the reward function or variations in gravity or mass of certain agent limbs as new tasks. The tasks that we consider in this work contain substantially different $\mathcal{A}$, $\mathcal{S}$ from one task to the next. For example, skills such as opening doors with keys, avoiding lava, or crossing a river which are quite different in comparison. Continual exploration strategies that use curiosity (Pathak et al., 2017) can be added as an intrinsic reward in the face of non-stationary environments in infinite horizon MDPs (Steinparz et al., 2022). Our proposed model uses Plan2Explore which has been shown to outperform curiosity-based methods (Sekar et al., 2020). The tasks themselves can be meta-learned using a latent variable world model and task similarities can be exploited when learning a new task (Fu et al., 2022).

Another related area of research is open-ended learning which aims to build agents that generalize to unseen environments through a curriculum that starts off

with easy tasks and then progresses to harder tasks thereby creating agents that can generalize (Wang et al., 2019; Team et al., 2021; Parker-Holder et al., 2022).

7.4 Recurrent State-Space World Models

We use DreamerV2 (Hafner et al., 2020) as a backbone for our CRL agent. DreamerV2 introduces a discrete stochastic and recurrent world model that is state-of-the-art on numerous RL benchmarks. Training DreamerV2 involves two steps: firstly training the world model to reconstruct state-action-reward trajectories from experience. Secondly, model-based planning; learning of the policy cheaply and sample-efficiently inside the world model using generated imagination from the world model. We describe these two processes in more detail in this background section.

7.4.1 Learning the World Model

DreamerV2 learns a recurrent (latent) state-space world model (RSSM) which predicts the forward dynamics of the environment. At each time step t the world model receives an observation o_t and is required to reconstruct the observations, o_t conditioned on the previous actions $a_{<t}$ (in addition to reconstructing rewards and discounts). The forward dynamics are modeled using an RNN, $h_t = \text{GRU}(h_{t-1}, z_t, a_t)$ (Chung et al., 2014) where h_t is the hidden state z_t are the discrete probabilistic latent states (Van Den Oord et al., 2017) which condition the observation predictions $p(o_t|z_t, h_t)$. Trajectories are sampled from an experience replay buffer and so persisting the replay buffer across different tasks should alleviate forgetting in the world model (Rolnick et al., 2019).

7.4.2 Policy Learning inside the World Model

The policy π is learned inside the world model by using an actor-critic (Sutton and Barto, 2018) while freezing the weights of the RSSM world model. At each step t of the dream inside the RSSM world model a latent state z_t is sampled, z_t, and the RNN hidden state condition the actor $\hat{a}_t \sim \pi(\cdot\,|z_t, h_t)$. The reward $\hat{r}_{t+1}$ is generated by the world model. The policy, π is then used to obtain new

trajectories in the real environment. These trajectories are added to the experience replay buffer. An initial observation o_1 is used to start generating rollouts for policy learning. This training regime ensures that the policy generalizes to previously seen environments through the world model.

7.5 World Models for Continual Reinforcement Learning

We leverage world models for learning tasks sequentially without forgetting. This is a good choice for CRL since the world model is trained by reconstructing state, action, and reward trajectories from experience, we can thus leverage experience replay buffers which persist across tasks to prevent *forgetting* in the world model. Additionally, we can train a policy in the imagination or in the generated trajectories of the world model, similar to generative experience replay methods in supervised CL which remember previous tasks by replaying generated data (Shin et al., 2017). Thus, using a world model is also sample efficient. Also, world models are *task-agnostic* and do not require external supervision about task changes, without signaling to the agent that it is interacting with a new task. Additionally, by generating rollouts in the world model's imagination the uncertainty in the world model's predictions, more specifically the disagreement between predictions can be used as a task-agnostic exploration bonus. To summarize, we propose using model-based RL with recurrent world models as a viable method for CRL, see Algorithm 4 for an overview, with DreamerV2 as the world model. Recently, world models have been shown to collect diamonds in Minecraft, a very hard skill to achieve which requires the composition of many other skills with

Algorithm 4 CRL with World Models

1: **Input:** Tasks (environments) $\mathcal{T}_{1:T}$, world model M, policy π, experience replay buffer $\mathcal{D}$.
2: **for** $\mathcal{T}_1$ **to** $\mathcal{T}_T$ **do**
3: Train world model M on $\mathcal{D}$.
4: Train π inside world model M.
5: Execute π in task $\mathcal{T}_\tau$ to gather episodes and append to $\mathcal{D}$.
6: **end for**

DreamerV3 (Hafner et al., 2023), the ideas introduced in this manuscript are directly applicable to newer world model methods as well.

7.5.1 Task-agnostic Exploration

The policy learns using the imagined trajectories from the RSSM world model and the world model's predicted rewards are used as a signal for the agent's policy and critic. The policy is also used to gain experience inside the real environment. For exploration, the policy prioritizes regions of the state and action space where the world model produces uncertain predictions. Hence, the uncertainty in the world model's trajectory prediction can be used as an additional intrinsic reward. This idea underpins Plan2Explore (Sekar et al., 2020) which naturally fits with DreamerV2.

The world model quantifies the uncertainty in the next latent state prediction by using a deep ensemble; multiple neural networks with independent weights. Deep ensembles are a surprisingly robust baseline for uncertainty quantification (Lakshminarayanan et al., 2017a) and the ensemble's variance is used as an intrinsic reward. The exploration neural networks in the ensemble are trained to predict the next RSSM latent features $[z_{t+1}, h_{t+1}]$. The world model is frozen while the ensemble is trained.

The policy π observes the reward $r = \alpha_i r_i + \alpha_e r_e$, where r_e is the extrinsic reward predicted by the world model, r_i is the intrinsic reward, the latent disagreement between the next latent state predictions. The coefficients α_i and α_e are $\in [0, 1]$. Hence the policy π can be trained inside the world model to seek regions in the state-action space that the world model struggles to predict and hence when the policy is deployed in the environment it will seek these same regions in the state-action space to obtain new trajectories to train the RSSM world model. The exploration strategy is significant for CRL since it is not task dependent unlike using DQN where each task needs an ϵ-greedy schedule (Kirkpatrick et al., 2016; Kessler et al., 2021b) or SAC (Haarnoja et al., 2018a) which needs an entropy regularizer per task (Wolczyk et al., 2021).

7.5.2 Selective experience replay methods

To enable Continual-Dreamer to remember how to solve all the previous tasks it has learned with a limited replay buffer size requires us to select important trajectories to fill the experience replay buffer and selectively choose trajectories to train the world model. DreamerV2 uses first-in-first-out (FIFO) replay buffer. It also randomly samples trajectories from the replay buffer to train the world model on and also randomly samples a trajectory so that the world model can start dreaming and allow the policy can learn inside the dream. In such a scenario, catastrophic forgetting can occur due to the loss of experience from previous tasks since the replay buffer is FIFO. There is prior work on managing experience replay buffers from supervised CL (Caccia et al., 2019) and off-policy RL (Isele and Cosgun, 2018b) which we can systematically study for the application of CRL with world models. To ensure that we have a uniform sample of experience from all tasks in the replay buffer to sample from we explore the following methods:

- **Reservoir Sampling** (rs) (Vitter, 1985; Isele and Cosgun, 2018b): enables a uniform distribution over all task experience seen so far. This is achieved by storing new examples in the replay buffer with a decreasing probability of $\min(n/t, 1)$, where t is the number of trajectories seen so far and n is the size of the replay buffer. This can be paired with any method of sampling from the replay buffer. By default, this is using random sampling.

- **Coverage Maximization** (cm) (Isele and Cosgun, 2018b): also attempts to create a uniform distribution of experience seen so far. Experience is added to the replay buffer by checking how close it is to trajectories already stored in the replay buffer. Trajectories are embedded using a fixed convolutional LSTM architecture (Shi et al., 2015) and we can calculate distances using an L^2 distance between the LSTM hidden state with respect to 1000 randomly selected trajectories from the replay buffer. The median distance determines the priority for the sample to be added to the replay buffer.

In addition to methods that populate the experience replay buffer, we can also consider how we should construct the mini-batch for world model and policy learning. For instance, we can prioritize more informative samples to help remember and help learn new tasks to aid stability and plasticity. We consider 3 approaches:

- **Uncertainty sampling** (*us*): we construct a minibatch of experience where the probability of sampling an episode corresponds to the trajectory's uncertainty or intrinsic reward from Plan2Explore. Next state uncertainties are generated for each transition and summed and normalized per trajectory before it is added to the replay buffer. We only calculate the uncertainty once before it is added to the replay buffer. This is similar to sampling the replay buffer according to the size of the temporal-difference error, known as sampling via "surprise" (Isele and Cosgun, 2018b). The temporal difference error is also only calculated once when transitions are added to the experience replay buffer for DQN.

- **Reward sampling** (*rwd*) (Isele and Cosgun, 2018b): we construct a mini-batch of experience for world model learning where the probability that an episode is sampled, corresponds to the reward from the environment.

- 50**:**50 **sampling**, of past and recent experience. We construct a mini-batch for world model learning based on a 50:50 ratio of uniform random sampling from the replay buffer and sampling from a triangular distribution that favors the most recent experience added so far to help learning more recent tasks. This idea is similar to the on-policy off-policy ratio of learning in CLEAR (Rolnick et al., 2019) which also aims to balance stability and plasticity.

7.5.3 Task-aware baseline

All replay buffer management techniques presented above are task-agnostic, i.e. operate without explicit task identification. We also consider a task-aware baseline for comparison, we use L^2 weight regularization with respect to the weights from the previous task, which is a simple regularization-based approach to CRL. In this

scenario, after the first task, we add to each loss function, an additional objective that minimizes the distance between the current model and policy weights and the optimal weights from the previous task.

7.6 Experiments

Our results indicate that DreamerV2 and DreamerV2 + Plan2Explore obtain good out-of-the-box performance for CRL on 3 Minigrid tasks (Chevalier-Boisvert et al., 2018). On a harder Minihack (Samvelyan et al., 2021) tasks from the CORA suite (Powers et al., 2021), we find that DreamerV2 and DreamerV2 + Plan2Explore exhibit forgetting. To address forgetting we systematically study various selective experience replay methods. The best configuration uses reservoir sampling (Vitter, 1985) which we name Continual-Dreamer and Continual-Dreamer + Plan2Explore.

We use two primary baselines. First, Impala which is a powerful deep RL method not designed for CRL (Espeholt et al., 2018). Second, we consider CLEAR (Rolnick et al., 2019) which uses Impala as a base RL algorithm and leverages experience replay buffers to prevent forgetting and is task-agnostic.

Throughout our experiments, we use 3 different metrics to assess the effectiveness of each method (Wolczyk et al., 2021) in addition to qualitatively inspecting the learning curves.

Average Performance. This measures how well a CRL method performs on all tasks at the end of the task sequence. The task performance is $p_\tau(t) = [-1, 1]$ for all $\tau < T$. Since we have a reward of $+1$ for completing the task and -1 for being killed by a monster or falling into lava. If each task is seen for N environment steps and we have T tasks and the τ-th task is seen over the interval of steps $[(\tau - 1) \times N, \tau \times N]$. The average performance metric for our continual learning agent is defined as:

$$p(t_f) = \frac{1}{T} \sum_{\tau=1}^{T} p_\tau(t_f), \tag{7.6.0.1}$$

where $t_f = N \times T$ is the final timestep.

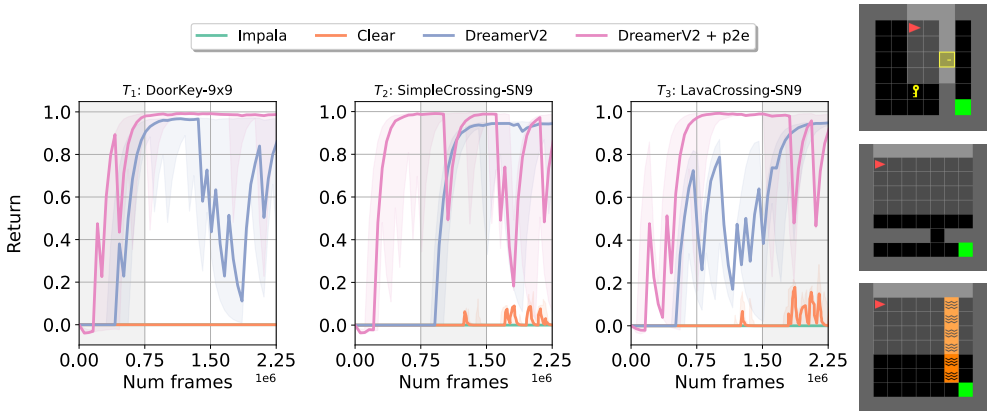


Figure 7.2: Performance of CRL agents on 3 Minigrid tasks. Grey-shaded regions indicate the environment which the agent is currently interacting with. All learning curves are a median and inter-quartile range across 10 seeds. On the right, we pick a random instantiation of the Minigrid environments that are being evaluated.

Forgetting. The average forgetting is the performance difference after interacting with a task versus the performance at the end of the final task. The average forgetting across all tasks is defined as:

$$F = \frac{1}{T}\sum_{\tau=1}^{T} F_\tau \quad \text{where} \quad F_\tau = p_\tau(\tau \times N) - p_\tau(t_f). \tag{7.6.0.2}$$

The forgetting of the final T-th task is $F_T = 0$. If a CRL agent has better performance at the end of the task sequence compared to after τ-th task at time-step $\tau \times N$ then $F_\tau < 0$.

Forward Transfer. The forward transfer is the difference in task performance during continual learning compared to the single task performance. The forward transfer is defined:

$$FT = \frac{1}{T}\sum_{\tau=1}^{T} FT_\tau \quad \text{where} \quad FT_\tau = \frac{\text{AUC}_\tau - \text{AUC}_{\text{ref}_\tau}}{1 - \text{AUC}_\tau}, \tag{7.6.0.3}$$

where AUC denotes the area under the curve and is defined as:

$$\text{AUC}_\tau = \frac{1}{N}\int_{(\tau-1)\times N}^{\tau \times N} p_\tau(t)dt \quad \text{and} \quad \text{AUC}_{\text{ref}_\tau} = \frac{1}{N}\int_{0}^{N} p_{\text{ref}_\tau}(t)dt. \tag{7.6.0.4}$$

$FT_\tau > 0$ means that the CRL agent achieves better performance on task τ during continual learning versus in isolation. So this metric measures how well a CRL agent transfers knowledge from previous tasks when learning a new task.

	Avg. Performance (↑)	Avg. Forgetting (↓)	Avg. Forward Transfer (↑)
Impala	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
CLEAR	0.04 ± 0.06	0.01 ± 0.02	0.00 ± 0.01
Impala×10	0.16 ± 0.16	0.06 ± 0.13	-
CLEAR×10	0.64 ± 0.20	0.00 ± 0.00	-
DreamerV2	0.72 ± 0.15	-0.04 ± 0.16	0.17 ± 0.61
DreamerV2 + p2e	0.74 ± 0.02	-0.02 ± 0.03	1.06 ± 0.83

Table 7.1: Results on 3 Minigrid tasks. All metrics are an average and standard deviation over 10 seeds. We use 0.75M interactions for each task and 7.5M in methods marked with ×10. ↑ indicates better performance with higher numbers, and ↓ the opposite.

7.6.1 Minigrid

We test the out-of-the-box performance of DreamerV2 and DreamerV2 + Plan2Explore as a CRL baseline on 3 sequential Minigrid tasks. Minigrid is a challenging image-based, partially observable, and sparse reward environment. The agent, in red, will get a reward of +1 when it gets to the green goal Fig. 7.2. The agent sees a small region of the Minigrid environment as observation, o_t. We use 3 different tasks from Minigrid: `DoorKey-9x9`, `SimpleCrossing-SN9` and `LavaCrossing-9x9`. Each environment has a different skill and so the tasks are diverse. Each method interacts with each task for 0.75M environment interactions, as previously proposed in Kessler et al. (2021b).

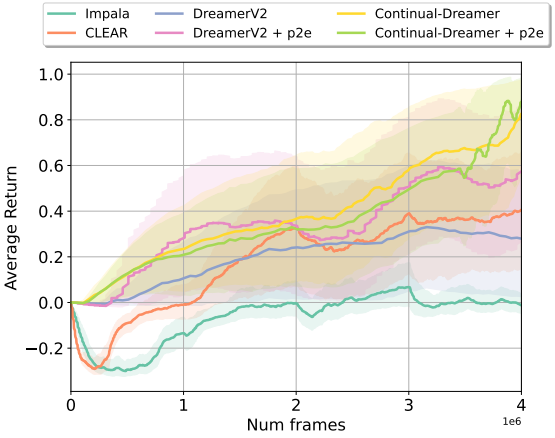


Figure 7.3: Return averaged over all tasks for various CRL agents on 4 Minihack tasks. All learning curves are IQM from the `rliable` package across 10 seeds and 1000 bootstrap samples (Agarwal et al., 2021).

We evaluate CRL agents on all tasks, see Fig. 7.2. The results indicate that DreamerV2 is able to solve difficult exploration tasks like the `DoorKey-9x9`. Additionally, since DreamerV2 trains its policy inside the world model it is more sample efficient than CLEAR which needs ×10 more environment interactions to be able to solve the easier Minigrid tasks `SimpleCrossing-SN9` and `LavaCrossing-9x9`,

Table 7.1. The addition of Plan2Explore enables DreamerV2 to solve these environments even more quickly, see Fig. 7.2. DreamerV2 does exhibit some forgetting of the `DoorKey-9x9` task and this indicates that additional mechanisms to prevent forgetting might be needed.

From the metrics in Table 7.1 we can see that DreamerV2 has strong forward transfer. From the learning curves for individual tasks Fig. 10.26 we can see that DreamerV2 struggles with independent task learning over the course of 1M environment steps. In contrast, when learning continually DreamerV2 is able to solve all tasks indicating that it transfers knowledge from previous tasks. This is not entirely surprising since the levels look similar and so the world model is able to reconstruct observations of a new task more quickly compared to reconstruction from scratch.

For DreamerV2 we use the model and hyperparameters from Hafner et al. (2020) with an experience replay buffer for world model learning of size 2M. For DreamerV2 + Plan2Explore we set the reward coefficients to $\alpha_i = \alpha_e = 0.9$ which was found by grid search of various single task Minihack environments over $\{0.1, 0.5, 0.9\}$ we use the same policy for exploration and evaluation and learn the world model by observation reconstruction only, rather than observation, reward, and discount reconstruction. We explore these design decisions using the Minihack benchmark in Section 10.4.3.4. For CLEAR we also use an experience replay buffer size of 2M like DreamerV2 and DreamerV2 + Plan2Explore.

7.6.2 Minihack

We test DreamerV2 and DreamerV2 with Plan2Explore on a set of harder Minihack tasks (Samvelyan et al., 2021). Minihack is a set of diverse, image-based, and sparse reward tasks based on the game of Nethack (Küttler et al., 2020) which have larger state spaces than MiniGrid and require learning more difficult skills such as crossing rivers by pushing multiple rocks into the river for instance. This tests the task-agnostic exploration mechanism from Plan2Explore further. We use 4 tasks Minihack tasks: in particular, we consider the following tasks `Room-Random-15x15-v0`,

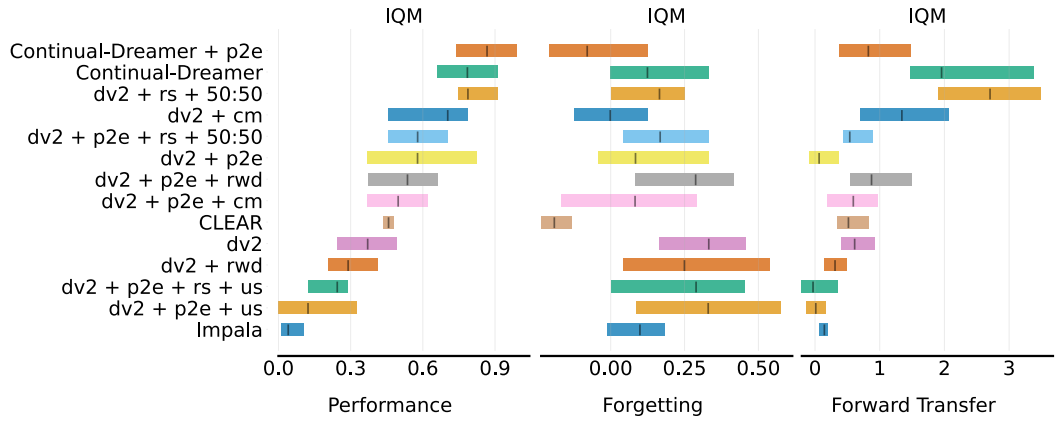


Figure 7.4: Metrics on 4 Minihack tasks using interquartile mean with 10 runs with different seeds and 1000 bootstrap samples from the `rliable` package (Agarwal et al., 2021).

`Room-Trap-15x15-v0`, `River-Narrow-v0`, and `River-Monster-v0` which are a subset of the 12 Minihack tasks from the CORA CRL benchmark (Powers et al., 2021). Each task is seen once and has a budget of 1M environment interactions. We use the same experimental setup as in Section 7.6.1, however, we keep the sizes of the experience replay buffer fixed to 1M for both DreamerV2, its variants, and CLEAR.

We perform a comprehensive evaluation of various replay buffer management and mini-batch selection strategies. We find that using reservoir sampling together with DreamerV2 and DreamerV2 + Plan2Explore is the best configuration which we name Continual-Dreamer, see detailed analysis in Section 7.6.2.1. The results of the Minihack experiment, as shown in Fig. 7.3, demonstrate that Continual-Dreamer and Continual-Dreamer + Plan2Explore perform better than the baselines regarding the average return over all tasks and 10 seeds. In particular, Continual-Dreamer and Continual-Dreamer + Plan2Explore exhibit faster learning on new tasks. Neither task-agnostic continual learning baselines, Impala and CLEAR, can learn the most difficult task `River-Monster-v0`, Fig. 10.28. These results suggest that world models are effective for consolidating knowledge across environments and changing task objectives. We also compare to a task-aware baseline: an L^2 regularization of the world model and actor-critic about the previous task's optimal parameters Fig. 10.29. We find that this performs poorly.

7.6.2.1 Evaluation of Different Selective Replay Methods

Fig. 7.4 presents results for different replay strategies. If we consider DreamerV2 and DreamerV2 + Plan2Explore we can see that there is some forgetting of the first Room tasks, see Fig. 10.28. Our main finding is that reservoir sampling robustly mitigates forgetting, which gives rise to Continual-Dreamer.

We can increase plasticity by biasing the sampling toward recent experiences. We can see that if we add 50:50 sampling of the minibatch construction together with reservoir sampling, this causes inconsistent effects. For Continual-Dreamer, 50:50 sampling increases forgetting while the performance remains constant, indicating better learning of harder tasks and transfer. On the other hand, when 50:50 sampling is applied to Continual-Dreamer + Plan2Explore performance and forgetting worsen.

We tested DreamerV2's performance in other variants, including uncertainty sampling (*us*), coverage maximization (*cm*), and reward sampling (*rwd*). The results we obtained are consistent with prior works (Isele and Cosgun, 2018b). It can be seen that *us* performed closely to Impala with low performance, forward transfer, and high forgetting. Using *rwd* sampling results in performance that does not improve over random sampling of the minibatch. Using *cm* with DreamerV2 and DreamerV2 + Plan2Explore results in less forgetting. However, it behaves inconsistently when observing performance; performance increases when applied to DreamerV2 and it decreases when applied to DreamerV2 + Plan2Explore.

As a baseline, we also tested a naive approach, which is to increase the size of the replay buffer; this indeed prevents forgetting and increases performance, see Section 10.4.3.5. However, this is at the cost of making it harder to learn new tasks: the harder exploration Minihack tasks are not solved and forward transfer decreases as we increase the size of the replay buffer.

7.6.3 Scaling to More Tasks

We scale to 8 Minihack tasks from the CORA suite: `Room-Random-15x15-v0`, `Room-Monster-15x15-v0`, `Room-Trap-15x15-v0`, `Room-Ultimate-15x15-v0`,

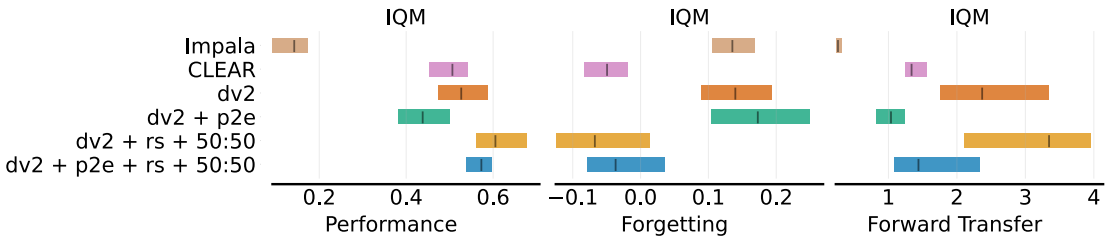


Figure 7.5: Metrics on 8 Minihack tasks using interquartile mean with 10 runs with different seeds and 1000 bootstrap samples. The results for DreamerV2 + rs + 50:50 with and without Plan2Explore are an interquartile mean with 5 runs with different seeds and 1000 bootstrap samples.

`River-Narrow-v0`, `River-v0`, `River-Monster-v0` and `HideNSeek-v0`. For DreamerV2 and its variants and for CLEAR we set the size of the experience replay buffer to 2M. We can see that by using reservoir sampling with 50:50 sampling prevents forgetting and slightly improves performance in DreamerV2 and DreamerV2 + Plan2Explore Fig. 7.5. Performance over CLEAR is also improved by introducing reservoir sampling and 50:50 sampling. The difficult Minihack tasks, such as `River-v0`, `River-Monster-v0` and `HideNSeek-v0` are solved by DreamerV2 and its variants but not by CLEAR Fig. 10.30. We conjecture that warm-starting the world model by solving easier related tasks enables DreamerV2 to improve its downstream performance. For instance the single-task DreamerV2, see Fig. 10.27, does not solve `River-Monster-v0` as often as continual agent Fig. 10.30. Arguably, the skills learned in `Room-Monster-15x15-v0` and `River-v0` enable DreamerV2 to solve `River-Monster-v0`.

7.7 Discussion and Future Works

We have explored the use of world models as a *task-agnostic* CRL baseline. World models can be powerful CRL agents as they train the policy inside the world model and can thus be sample-efficient. World models are trained by using experience replay buffers and so we can prevent *forgetting* of past tasks by persisting the replay buffer across from the current task to new tasks. Importantly, the world model's prediction uncertainty can be used as an additional intrinsic task-agnostic reward to help exploration and solve difficult tasks in a task-agnostic fashion (Sekar

et al., 2020). Previous CRL exploration strategies in the literature all require an indication of when the agent stops interacting with a previous task and starts interacting with a new task to reset an exploration schedule. Our implementation uses DreamerV2 as the world model (Hafner et al., 2020), and we demonstrate a selective experience replay setting which we call Continual-Dreamer which is a powerful CRL method on two difficult CRL benchmarks.

We show empirically that world models can be a strong task-agnostic baseline for CRL problems compared to state-of-the-art task-agnostic methods (Rolnick et al., 2019). DreamerV2 with Plan2Explore outperforms CLEAR on Minigrid. Our experiments on Minihack test the limits of using world models for CRL and require us to introduce experience replay buffer management methods to aid in retaining skills in addition to enabling the learning of new skills. We show that reservoir sampling enables an even coverage of experience in the replay buffer to mitigate forgetting. We call this configuration of DreamerV2 with reservoir sampling Continual-Dreamer. Future work will explore continuous control CRL benchmarks, such as ContinualWorld (Wolczyk et al., 2021), and explore other world-models.

8

Continual-wav2vec2.0: Efficient adapter transfer of Self-Supervised speech models for Automatic Speech Recognition

Whereas previous chapters study novel continual learning methodologies on benchmark problems. In this chapter, we explore the application of continual learning. Production NLP and speech traffic are non-stationary by their very nature, new utterances enter into common everyday parlance. As a result speech and language models need to be continually updated in the face of this non-stationary data Lazaridou et al. (2021); Baby et al. (2022). However, language and speech production models are enormous and so are the datasets on which they are trained. If they are only trained on the most recent data from production systems, the models will forget Baby et al. (2022). On the other hand training, on all new and old data is expensive and data protection laws mean that data cannot be stored indefinitely. Hence production systems require continual learning mechanisms for efficient re-training in the face of new data Section 1.3.3.

In this chapter, we explore continual pre-training of large self-supervised speech models to enable them to absorb new datasets without forgetting the initial datasets on which they were trained. This enables the models to retain previous task

information without forgetting while being able to quickly train on a new task and decrease training times when training a new task.

With this in mind, we present a method for transferring pre-trained self-supervised (SSL) speech representations to multiple languages. In this chapter, we show how we can transfer knowledge from large self-supervised pre-trained models to speed up self-supervised pre-training of a new model in a subsequent step. There is an abundance of unannotated speech, so creating self-supervised representations from raw audio and fine-tuning on small annotated datasets is a promising direction to build speech recognition systems. State-of-the-art SSL models generally perform SSL on raw audio in a pre-training phase and then fine-tune on a small fraction of annotated data. However, these models are very expensive to pre-train. We use an existing wav2vec2 model and tackle the problem of learning new language representations while utilizing existing model knowledge. Crucially we do so without catastrophic forgetting of the existing language representation. We use adapter modules to speed up pre-training for a new language task. Our model can decrease pre-training times by 32% when learning a new language task, and learn this new audio-language representation without forgetting previous language representations.

8.1 Introduction

Neural networks require large labeled datasets to train for applications such as image recognition or neural machine translation. Automatic speech recognition (ASR) labeled datasets are expensive to obtain and speech recognition systems generally need thousands of hours of speech annotated with text for good performance. Furthermore, there are thousands of different languages spoken in the world and only a select few have large annotated datasets. Self-supervised learning (SSL) has recently garnered a lot of attention in machine learning since it can learn representations from unlabeled data alone and achieve extremely competitive results in comparison to fully supervised methods while only training on a small amount of labeled data. SSL for ASR has been successfully shown to produce very good performance when pre-training on an unlabeled raw audio dataset, and

then subsequently fine-tuning on a small labeled dataset. Self-supervised speech recognition has been proposed with many different approaches (Oord et al., 2018; Pascual et al., 2019; Chung et al., 2019; Schneider et al., 2019), the most successful and pre-dominant approaches so far have been wav2vec2 (Baevski et al., 2020), and HuBERT (Hsu et al., 2021).

SSL for ASR models do two things simultaneously, firstly they learn how to map raw audio into a vector representation and secondly, they learn a language-specific representation of the extracted speech features. This chapter focuses on the wav2vec2 architecture, but our proposed approach is also applicable to the more recent HuBERT model. wav2vec2 uses state-of-the-art architectures for sequence learning based on multi-head self-attention (Vaswani et al., 2017). This means that these models are extremely large with $O(10^8)$ parameters for the wav2vec2 base model. Training this model takes in the order of two weeks to train on an 8 GPU node[1]. Going one step further, if we want to learn a representation for a second language then obtaining a multi-lingual representation from the union of two unlabeled datasets will take even longer to train.

We are interested in leveraging ideas from the Continual Learning (CL) paradigm to be more economical when learning a new language representation and to make SSL for audio more experimentation-friendly. SSL models for speech encode generic speech information as well as language-specific representations. We aim to utilize those generic speech features to speed up pre-training on a new language. One major issue for any kind of continual learning is catastrophic forgetting (French, 1999), where the previous task parameters are overwritten while learning a new task. We aim to retain performance on the original task while learning our new language representations Fig. 8.1. We also want to do this in a parameter-efficient manner that scales sub-linearly compared to training an independent model for a new task.

Our core contributions are to demonstrate a method of efficient transfer learning for pre-trained, self-supervised ASR models. This learns a new language speech representation by utilizing knowledge from an existing speech representation model.

[1]Simulating a 64 GPU cluster with 8 gradient accumulation steps on 960 hours of audio.

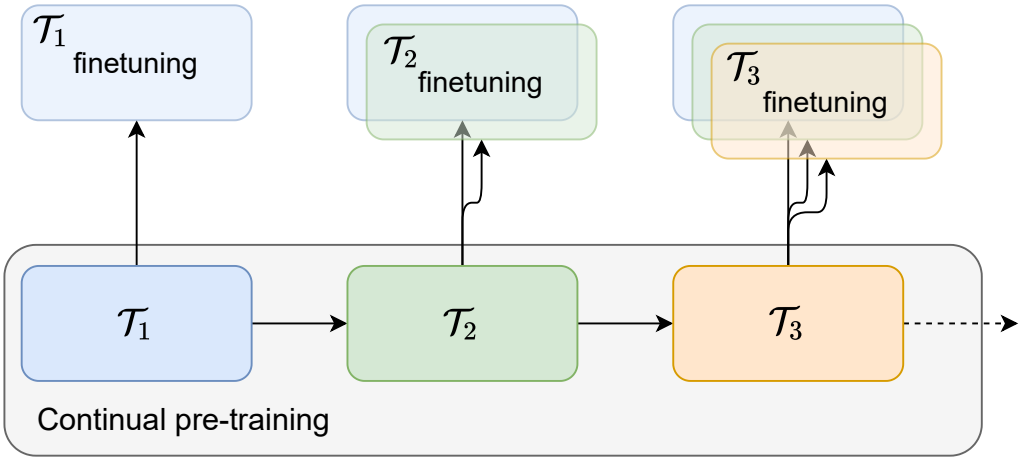


Figure 8.1: The task $\mathcal{T}_i$ corresponds to self-supervised learning of a representation of a language i. Different colors are different languages. After pre-training on a new language we assess the performance on all languages seen so far by fine-tuning.

Our model can retain performance on old tasks and crucially transfer knowledge from a previous task to be more computationally efficient when training a new task. To do this we use a modular approach where we add new parameters as language adapters (Houlsby et al., 2019) for learning new languages. The power of this method is demonstrated in that we are able to prevent forgetting completely and speed up training a new task by reusing model components, see the results in Section 8.4. We name our method continual-wav2vec2.0 (cwav2vec2) and refer to it as such throughout.

8.2 Preliminaries

8.2.1 wav2vec 2.0

The wav2vec2 model in Fig. 8.2 takes in raw waveform as input, this is encoded by a convolutional neural network (CNN) into a representation which is then quantized and contrasted against a token output by a multi-head self-attention (MHSA) encoder (Vaswani et al., 2017). Inputs to the transformer encoder are randomly masked and compared to the true tokens of the quantizer (Baevski et al., 2020) and so $g(\cdot)$ has to perform BERT style masked prediction to learn a representation from audio (pre-training) (Devlin et al., 2018).

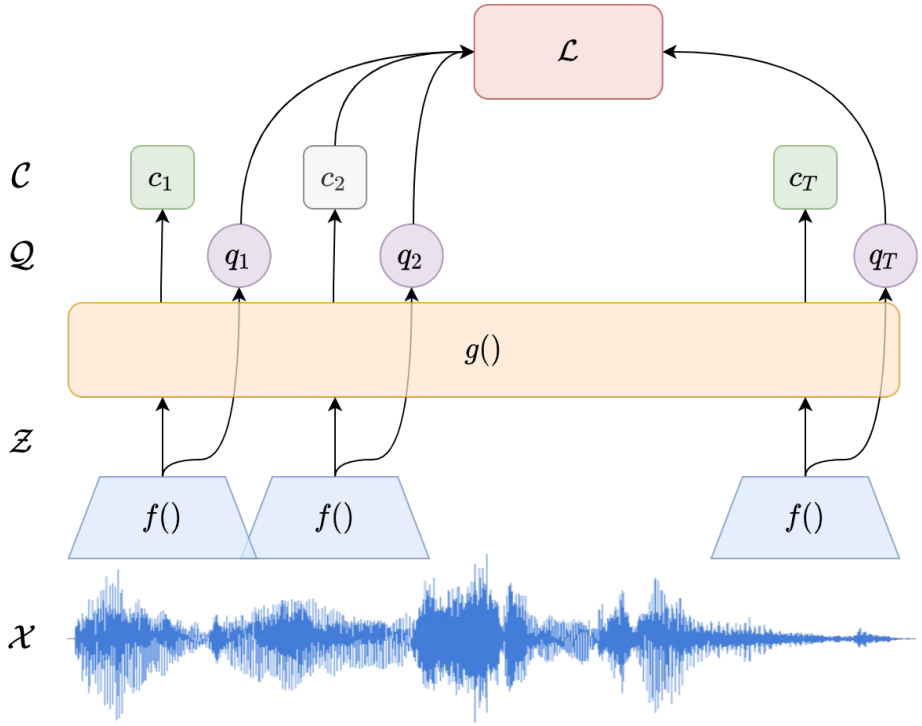


Figure 8.2: Diagram of wav2vec2 model which takes in as input raw waveform $\mathcal{X}$ and uses a self-supervised loss $\mathcal{L}$ to learn a representation of speech. $f(\cdot)$ is a convolutional encoder and $g(\cdot)$ is a masked transformer encoder the grey time-step c_2 is masked out and is predicted by $g(\cdot)$ and contrasted against the discrete tokens q_t for $t \in \{1, \ldots, T\}$.

Formally, the model takes as input a raw waveform $\mathcal{X}$, a CNN extracts features $f : \mathcal{X} \rightarrow \mathcal{Z}$. The extracted sequence is passed to an MHSA context network, $g : \mathcal{Z} \rightarrow \mathcal{C}$. The wav2vec2 model also discretizes the output of the feature extractor $\mathcal{Z} \rightarrow \mathcal{Q}$. Quantization module is comprised of G codebooks (Gumbel-Softmax distributions (Jang et al., 2016; Maddison et al., 2016)), with V entries. The outputs of the quantizer are then contrasted against the outputs of the MHSA encoder.

Objective. The contrastive loss $\mathcal{L}_m$ identifies the true quantized latent speech representation from masked time steps. The MHSA encoder output $\boldsymbol{c}_t = g(\cdot)$ at a masked time step t is contrasted against the true quantized latent speech representation $\boldsymbol{q}_t$ from a set of $K+1$ representations Q_t which include K distractors. Distractors are uniformly sampled from other masked time steps of the same utterance. The loss is:

$$\mathcal{L}_m = -\log \frac{\exp(\text{sim}(\boldsymbol{c}_t, \boldsymbol{q}_t)/\kappa)}{\sum_{\tilde{\boldsymbol{q}} \sim Q_t} \exp(\text{sim}(\boldsymbol{c}_t, \tilde{\boldsymbol{q}})/\kappa)}, \tag{8.2.1.1}$$

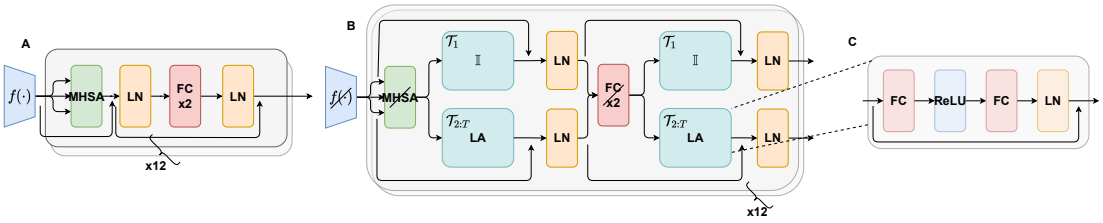


Figure 8.3: Diagram of main components of the cwav2vec2 architecture. **A** MHSA layers in the original wav2vec2 encoder $g(\cdot)$. **B** the cwav2vec2 additional layers in the encoder $g(\cdot)$: additional adapters and layer norms per pre-training task. **C** shows the adapter architecture. Modules with a diagonal line through denote layers that are frozen for tasks $t > 1$. II is an identity mapping for the first task only.

where $\text{sim}(\boldsymbol{a}, \boldsymbol{b})$ is the cosine similarity, defined as $\text{sim}(\boldsymbol{a}, \boldsymbol{b}) = -\boldsymbol{a}^\top \boldsymbol{b} / ||\boldsymbol{a}|| \, ||\boldsymbol{b}||$.

Masking and pre-training. The objective is to identify the correct quantized latent audio representation by comparing it to the masked prediction. The feature encoder outputs are masked before being fed into the context network. Inputs into the quantization module are not masked. We follow the same approach as in wav2vec2 (Baevski et al., 2020).

8.2.2 Adapters

Adapters are intermediate layers that are inserted into a deep neural network to allow adaptation to a new domain. The intermediate layers can be convolutional layers that allow adaptation to a new vision dataset after having trained the network on ImageNet to obtain domain-agnostic parameters (Rebuffi et al., 2017a). Similarly in NLP adapters in the form of fully-connected bottleneck layers can be inserted in each layer of BERT to allow parameter efficient fine-tuning on a wide range of text classification tasks (Houlsby et al., 2019). We use these adapters for cwav2vec2 Fig. 8.3.

8.3 Continual-wav2vec 2.0

The wav2vec2 model does two things: a) it learns a representation from raw waveform to a vector and b) learns a language representation of speech, both by self-supervision. The objective of our cwav2vec2 model is to transfer as much knowledge from the first task's language representation to 1) enable quicker learning

of a second language representation, 2) prevent catastrophic forgetting of our first task's language representation.

We take a modular approach to pre-training transfer by using language adapters (LAs) (Houlsby et al., 2019; Pfeiffer et al., 2020a,b). LAs have been used before for fine-tuning large language models such as BERT (Devlin et al., 2018) on different tasks after pre-training. The cwav2vec2 model aims to speed up pre-training on a new language speech task ($\mathcal{T}_{i>1}$) by freezing learned parameters of the first task. The feature extractor $f(\cdot)$ and MHSA layers in $g(\cdot)$ are frozen. We assume that the wave-form to vector function mapping $f(\cdot)$ doesn't need further training for tasks $\mathcal{T}_{i>1}$. Two LAs are added in each layer of $g(\cdot)$ to learn a language-specific representation during the self-supervised pre-training (Houlsby et al., 2019), in addition to a language task-specific layer norm (Ba et al., 2016). The LA architecture is simply two fully connected layers followed by a layer norm with a skip connection (see Fig. 8.3). Despite restricting the number of parameters used for pre-training a new language/task representation, recent work has shown that BERT style pre-training learns universal representations which are widely transferable to a wide variety of tasks (Lu et al., 2021). Thus, our hypothesis is that even though we are training a first task on a *single* language audio dataset, the features can transfer well for a second audio pre-training task and enable us to meet our criteria 1). Since we are freezing parameters from our first task then we should also meet our criteria 2).

We also work with *multi-head* architectures which are a common method used in CL to retain task-specific knowledge. These work by learning task-specific mappings from the feature extractor $f(\cdot)$ and from the encoder $g(\cdot)$ to the objective $\mathcal{L}$ (Li and Hoiem, 2017a).

Fine-tuning. We follow the procedure in (Baevski et al., 2020) by freezing the feature extractor and appending a linear classification layer to the output of the wav2vec2 architecture. However, we additionally freeze the MHSA encoder layers and append a new task-specific LA on top of the pre-trained LA. This is partially inspired by (Pfeiffer et al., 2020b) which uses both language and task-based

adapters for NLP. We only optimize the task-specific LA and the layer norms in the MHSA encoder using a standard CTC loss.

This approach of adapter fine-tuning significantly reduces the number of trainable parameters per downstream task. This enables us to efficiently test ASR performance on multiple languages.

8.4 Experiments

We first pre-train wav2vec2 on English and fine-tune on English as a baseline. We use the `fairseq` framework for all experiments (Ott et al., 2019). We then continue pre-training on either French or Spanish and fine-tune the model throughout training on all languages seen so far. See Fig. 8.1. We use methods inspired by CL as baselines.

Data. For English experiments we pre-train using the full 960h LibriSpeech dataset (Panayotov et al., 2015a), and fine-tune using the 10-hour supervised LibriLight subset (Kahn et al., 2020). Results are reported on Librispeech dev-clean. For French and Spanish we use the Common Voice dataset (Ardila et al., 2019a), approximately 1000h for pre-training and 10h for fine-tuning. Results are reported on the standard Common Voice test sets.

Baselines. We compare our cwav2vec2 model to the following baselines. Warm-starting: simply continuing training with wav2vec2 on a new dataset. Multi-head wav2vec2 (MH wav2vec2) is a simpler continual learning model which leverages task-specific neural modules Kirkpatrick et al. (2016); Nguyen et al. (2017); Rusu et al. (2016a) between 1. the convolutional encoder and the MHSA layers and 2. between the MHSA layers and the output layer. In addition, MH wav2vec2 has a frozen convolutional encoder $f(\cdot)$, speech representation learning can be re-used from one language to another, also we have tuned the learning rate for learning a second task. We also compare to MH wav2vec2 with an L2 regularization of the MHSA layers about the previous task's optimum (MH wav2vec2 + L2). We add the term $\eta||\boldsymbol{\theta} - \boldsymbol{\theta}^*_{t-1}||_2$ to the loss function, where $\boldsymbol{\theta}^*_{t-1}$ is the previous task's optimal parameters, η is a hyperparameter. This is a simpler mechanism than other CL regularization methods (Kirkpatrick et al., 2017; Zenke et al., 2017a).

Results. In Fig. 8.4 and Fig. 8.5 we can see that simply warm-starting the wav2vec2 model when learning a new language representation for French or Spanish after English in $\mathcal{T}_1$ yields an unstable model with 100% word-error rate (WER). With MH wav2vec2 we can learn a more stable representation however this model is susceptible to forgetting the original English language representation; we see a degradation in the English WER when fine-tuning. Thirdly we validate our hypothesis that by using LAs we can entirely stop forgetting and enable learning a new language representation. However, the new language representation is not as powerful as MH wav2vec2 since the number of training parameters is smaller when learning a second task. We see that using an L2 regularization on the MHSA layers (MH wav2vec2 + L2) can help to alleviate forgetting. However, it is not as robust as using adapters, as can be seen in a small drop in performance when pre-training on Spanish. Additionally, we do not get the same efficiency benefits of cwav2vec2 Fig. 8.6. Training time is reduced with cwav2vec2 and the number of trainable parameters is much smaller allowing much greater scalability to multiple tasks.

Our objective 1), to transfer knowledge from our first task to learn a new language representation or audio and decrease the amount of time required to train a new task is achieved using our cwav2vec2 model Fig. 8.6. The reason for this is due to parameters that are frozen. Objective 2) is also achieved as LAs preserve the previous task parameters thus preventing forgetting.

8.5 Discussion and Conclusion

We introduce cwav2vec2, a simple and effective modular continual learning approach to build up self-supervised language representations from raw audio. cwav2vec2 builds on the successful wav2vec2 model, it is able to limit catastrophic forgetting of an initial audio representation while learning a new task by freezing parameters and training LAs which are used to specialize to a new task. By freezing parameters and training new LA modules, we are able to significantly speed up learning a new self-supervised language representation on audio. By using cwav2vec2 we can train

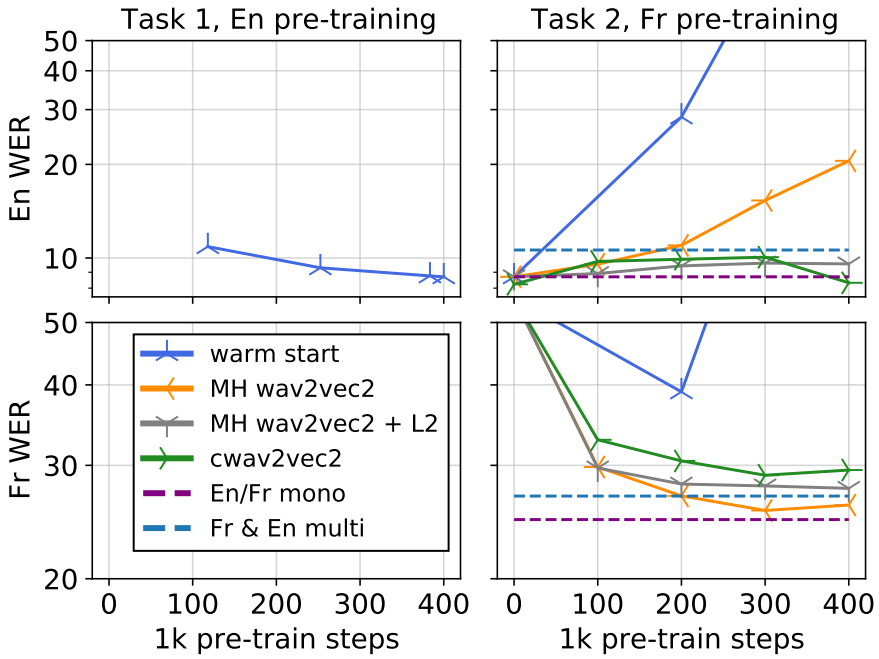


Figure 8.4: Word-error rate (WER) learning curves after fine-tuning on 10 hours of data having performed $n \times 1000$ steps of self-supervision. The purple dashed lines denote monolingual pre-training and then finetuning performance on En and Fr. The blue dashed line denotes the multi-lingual performance of pre-training on En and Fr together in a multi-task fashion and then finetuning on respective languages. By using LAs we protect against forgetting and allow learning new languages.

a new representation in 10 days compared to nearly 15 using wav2vec2, enabling us to be more economical with computation resources and making self-supervision with audio more experimentation friendly. Additionally, we can achieve close to the mono-lingual performance on a new language despite having frozen a significant fraction of the model's parameters and training LAs only. We are able to utilize generic speech information from the initial pre-training without needing to retrain on the original data and achieve good results in multiple languages through purely monolingual training. Future directions include allowing cwav2vec2 to scale to further tasks and allowing different LAs to combine.

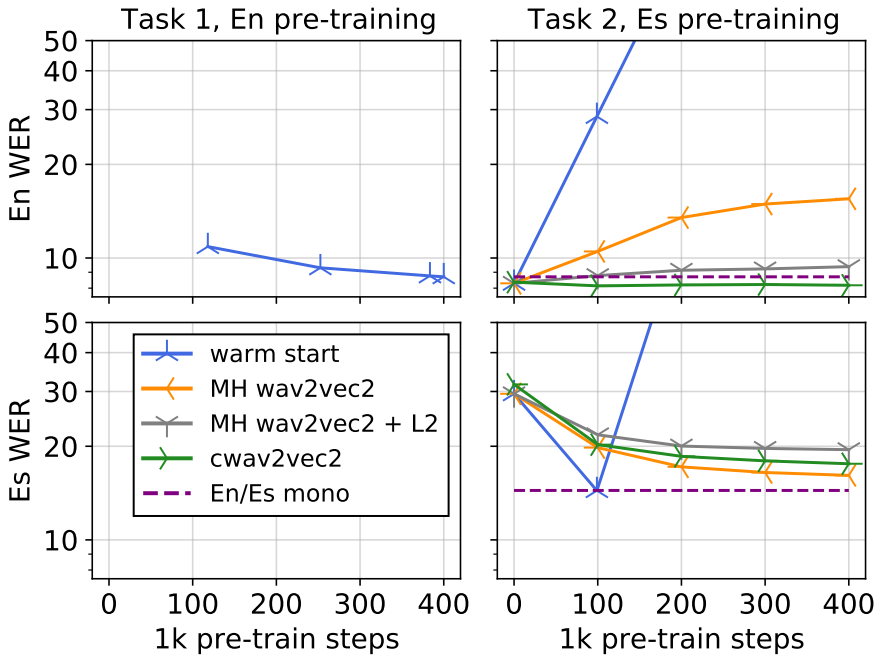


Figure 8.5: Word-error rate (WER) learning curves after fine-tuning on 10 hours of data having performed $n \times 1000$ steps of self-supervision of first En then Es. We yield very similar results when using Spanish as to the learning curves for French in Fig. 8.4. The purple dashed lines denote monolingual pre-training and then finetuning performance on En and Es.

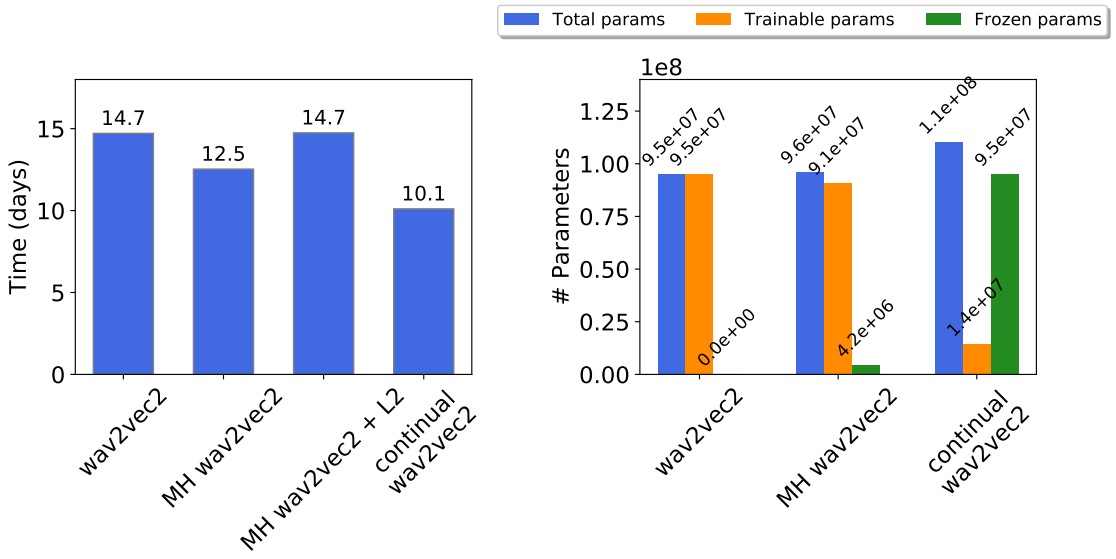


Figure 8.6: **Left** Time required to pre-train our models on a second task. **Right** The total number of parameters in the models, total number of trainable and frozen parameters. Pre-training on a second task is more efficient using cwav2vec2.

9 Discussion and Conclusion

In this thesis, we emphasize the ability of Bayesian neural networks to incorporate new data naturally by using sequential Bayesian inference. This enables us to sequentially build up a posterior over the network parameters which prevents forgetting of past tasks, enables forward transfer by using previous similar task posteriors as a prior, and has calibrated uncertainties. We presented a novel Bayesian neural network with a non-parametric Indian Buffet process prior Chapter 4. This enables the network to add more neurons as more tasks are encountered, alleviating underfitting and overfitting issues affecting BNNs. We see benefits over variational continual learning (Nguyen et al., 2017) which can underfit or overfit in certain continual learning problems.

Despite this novel contribution to Bayesian continual learning, we also revisit Bayesian continual learning and remove all the common tricks that are used in the literature and in Chapter 4. We study whether solely relying on sequential Bayesian inference enables us to prevent forgetting and sequentially build up the multitask posterior as dictated by theory. In Chapter 4 we typically rely on tricks such as task-specific heads. One exception to the need for task-specific heads is when we have conflicting tasks Chapter 6 and the task identifier is required to disambiguate between tasks. Other Bayesian continual learning approaches, also employ coresets of data from previous tasks (Nguyen et al., 2017; Pan et al., 2020) to help prevent

forgetting, which is again not required by theory. Indeed it is commonly understood in the community and we show that solely relying on a variational approximation of the posterior results in very poor performance and we see substantial forgetting.

This motivates us to test whether having access to the true posterior (and not a variational approximation) enables the model to sequentially build up the multi-task posterior representation of all tasks and thus prevent forgetting. We use Hamiltonian Monte Carlo (HMC) which is the gold standard of Bayesian inference. We use a density estimator over HMC samples to represent the posterior as a prior for a new HMC chain for a new task. We find that we are barely able to improve over a variational approximation, demonstrating just how difficult it is to perform Bayesian sequential inference in parameter space despite checks to ensure a multi-modal and accurate density estimator over posterior samples. The source of error in this approach is likely the density estimation which removes probability mass over parts of parameter space that are important for new tasks, but not so important for the current task. We perform many standard checks for HMC convergence.[1] From here we study exact sequential inference in linear models to show that, despite exact inference, we can still observe forgetting if there is model misspecification. If there is model misspecification, the multi-task posterior might actually be suboptimal for the continual learning criteria which we aim to optimize for. We also highlight the problem of dataset imbalances which can lead to forgetting past tasks; data imbalances are a common problem in many areas of machine learning in addition to continual learning Chrysakis and Moens (2020). Sequential Bayesian inference is also not exempt from the effects of task data imbalances. We propose a simple baseline called ProtoCL which relies on a functional regularization for learning new tasks by maintaining previous task prototypes using a random coreset of past data. A more promising direction of research for probabilistic continual learning methods is to functionally regularize the learning of new task functions using for instance the inducing point literature Titsias et al. (2020a); Rudner et al. (2022a).

[1]That said it is not clear how HMC inference in a Bayesian neural network can infer the true posterior of a BNN and account for permutation invariance in the weights, this is left for future work.

We apply continual learning to reinforcement learning where we show that including prediction uncertainties can aid various aspects of learning in changing environments. In Chapter 6 we show how we can obtain different tasks by simply changing the reward functions of a POMDP, or if the state spaces are similar between different tasks. In this setting, we show that powerful task-agnostic experience replay methods can fail since experiences can conflict. We find that it is important to have task information available to enable the agent to condition their policies. This can be performed easily with separate heads per task. To enable automatic task inference at test time we introduce our method, OWL, which automatically selects the correct head using a multi-armed bandit. We use the temporal difference (TD) error Eq. (2.3.0.1) as feedback to the multi-armed bandit and show that providing uncertainties, using an ensemble (Lakshminarayanan et al., 2017a) helps the bandit select the correct head for solving a task it has seen before without task information. We prevent forgetting via simple regularization techniques of the feature extractor which is shared among all tasks.

We explore using the model-based reinforcement learning paradigm for CRL in Chapter 7, and propose a simple baseline for CRL problems using world models. We show that latent recurrent world models are particularly effective since they naturally leverage experience replay buffers for learning the world model, which we can persist across tasks to prevent *forgetting* of the world model. Further, they are sample efficient since they train a policy inside the world model, limiting the amount of experience required from the environment. Finally, and most importantly in the context of this thesis, is that by leveraging ensembles (Lakshminarayanan et al., 2017a) the world model can generate trajectories and the ensemble uncertainty in the next state predictions can be used as an intrinsic reward and for continual exploration (Steinparz et al., 2022). This allows the world model policy to explore areas where there is a high level of disagreement between the ensemble trajectories when the policy is rolled out in the environment. We note that this form of exploration in terms of trajectory prediction uncertainty, is completely *task-agnostic* a rarity within the family of CRL methods. In general, continual learning has not

been fully studied in the field of RL, and there are lots of interesting open problems. For instance, how to leverage offline datasets for CRL (Levine et al., 2020).

We consider an application of continual learning for rapid learning from new datasets in large self-supervised models that take days if not weeks to train Chapter 8. By inserting small neural network modules, adapters (Houlsby et al., 2019) for learning a new task, we can effectively leverage the representations learned for initial tasks to learn a new task, decreasing training times while continuing to be able to solve early tasks. This technique can reduce training times by over 40% making experimentation quicker for large self-supervised automatic speech recognition models without having to rely on retraining on all data seen thus far. This is important since production speech and NLP models degrade in performance if they are not updated Lazaridou et al. (2021), so efficiently updating pre-training representations is an important problem for reducing already long training times.

The effectiveness of Bayesian continual learning. We show in Chapter 5 the difficulties of relying on sequential weight-space Bayesian inference to recursively build up the multi-task posterior to continual learning problems. Even with HMC (Neal et al., 2011) the problem is very challenging, despite HMC being a gold standard for Bayesian deep learning (Izmailov et al., 2021). Sequential Bayesian inference over neural network parameters cannot be relied upon for continual learning. Rather, research efforts should focus on more functional approaches to sequential Bayesian inference, in which previous task functions are remembered (Titsias et al., 2020a; Pan et al., 2020). This shifts the problem of remembering previous task functions to a coreset similar to sparse variational-inference Gaussian Processes (Titsias, 2009; Hensman et al., 2013). Thus the question becomes one of effectively choosing a coreset, under a potential budget constraint, to ensure a scalable continual learning solution in which the distribution of the coreset matches that of the empirical distribution of the task data as closely as possible (Manousakas et al., 2022).

There is little continual learning research assessing the uncertainty calibration of Bayesian continual learning methods. In Bayesian deep learning, the community has

measured the robustness of uncertainties produced by the methods under question using metrics such as expected calibration error (ECE) or out-of-distribution detection. Probabilistic continual learning models are currently only assessed like non-probabilistic methods, using posterior expectation accuracies only. We know that Bayesian methods provide additional benefits beyond more accurate predictions and current continual learning benchmarks do not measure these uncertainty metrics. There has been little work on whether in addition to catastrophic forgetting, there is also forgetting of calibrated uncertainties.

Reconciling continual reinforcement learning and meta-reinforcement learning. Supervised continual learning can be approached using few-shot learning, either using meta-learning with continual learning tasks and evaluation on unseen continual learning problems (Javed and White, 2019), or as new tasks are observed learning is incorporated in meta-parameters (Riemer et al., 2018; Caccia et al., 2020; Zhang et al., 2021b; Tao et al., 2020). There has been little work in enabling meta-RL (Kirk et al., 2021) to work effectively on CRL problems.

Neural networks beyond gradient descent. Looking further afield, the phenomenon of catastrophic forgetting in neural networks can occur when learning a new task whereby the gradient-based parameter optimization pushes parameters away from areas of low loss for previous tasks. This phenomenon is the result of using stochastic gradient descent on non-stationary problems. It is without question that stochastic gradient descent, defining a loss function, and parameterizing a large model result in the current best results for many problems. However, moving away from this machine learning paradigm to using more biologically inspired neural networks, potentially inspired by further insights into how the human brain works, could lead to methods that can transfer knowledge and retain skills under distribution shifts and so overcome the issue of catastrophic forgetting (Kaplanis et al., 2018).

Plasticity and capacity loss. Research in capacity loss, considers non-stationary RL environments and demonstrates that RL policy networks tend to lose the capacity to fit new functions as learning proceeds. Throughout the course of learning the policy, the policy becomes less able to learn new function mappings

from states to actions. To maintain plasticity, one can regularize the policy with respect to a randomly initialized network to ensure that the network retains capacity and is plastic to learning under standard non-stationary RL conditions (Lyle et al., 2022, 2023). This notion of maintaining plasticity to enable rapidly learning new tasks has not been explored in the context of continual reinforcement learning, where in addition to the standard notion of non-stationarity in a single RL environment there is an additional non-stationarity as new tasks are seen. On the other hand, maintaining stable policies that are able to retain skills has been the focus of CRL research up until now. Enabling rapid learning of new tasks, similar to meta-RL is an interesting future direction of investigation.

Adaptation to new financial markets and regimes. When financial market conditions change, forecasting models that are used to take positions in the markets can perform very poorly. The recent success of deep learning for automatically learning representations from data to make accurate predictions has translated into better models for financial forecasting (Lim et al., 2019). However deep learning models also rely on large datasets for automatic representation learning. When there is a *change of regime* in financial markets one might only have a handful of data points available to then produce an accurate forecast. Recent work shows that deconstructing financial timeseries into different regimes using change-point detection techniques using a deep learning model to make forecasts using inter-regime data substantially improves returns (Wood et al., 2022). There is an apparent benefit to training deep learning models which have additional supervision regarding which regime data comes from, and only training and making predictions on data from a particular regime. This is not a continual learning problem, but more a continual adaptation and few-shot learning problem (Finn et al., 2017; Garnelo et al., 2018). To develop an agent that can very quickly adapt to new market regimes, with just a handful of datapoints. One can use few-shot learning applied to timeseries data to enable quick adaptation to new regimes by transferring knowledge from past regimes and other markets to quickly learn a current regime or new market. Meta-learning

has been used in finance to construct a partial index portfolio to track a benchmark index where the asset allocation is meta-learned Yang and Hospedales (2023).

Continually updating large models. In the era of large language and speech models the distribution of utterances that are commonly used by the population is ever-evolving and models need to be continually updated in the face of this non-stationary data Lazaridou et al. (2021). This is a continual learning challenge, current solutions involve simple sampling of new and old data at a particular ratio Baby et al. (2022). Current continual learning benchmarks do not reflect this kind of gradual non-stationarity in the data. So firstly there are no set benchmarks and definitions of these continual learning problems let alone solutions. In the future, this is an interesting and impactful area of research.

Bibliography

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A. C., and Bellemare, M. (2021). Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems*, 34:29304–29320.

Ahn, H., Cha, S., Lee, D., and Moon, T. (2019). Uncertainty-based continual learning with adaptive regularization. *Advances in neural information processing systems*, 32.

Aitchison, L. (2020). Bayesian filtering unifies adaptive and non-adaptive neural network optimization methods. *Advances in Neural Information Processing Systems*, 33:18173–18182.

Aljundi, R., Caccia, L., Belilovsky, E., Caccia, M., Lin, M., Charlin, L., and Tuytelaars, T. (2019a). Online continual learning with maximally interfered retrieval. *arXiv preprint arXiv:1908.04742*.

Aljundi, R., Kelchtermans, K., and Tuytelaars, T. (2019b). Task-free continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11254–11263.

Aljundi, R., Lin, M., Goujaud, B., and Bengio, Y. (2019c). Gradient based sample selection for online continual learning. *Advances in neural information processing systems*, 32.

Aljundi, R., Lin, M., Goujaud, B., and Bengio, Y. (2019d). Gradient based sample selection for online continual learning. In *Advances in Neural Information Processing Systems*.

Ammar, H. B., Eaton, E., Ruvolo, P., and Taylor, M. (2014). Online multi-task learning for policy gradient methods. In *International conference on machine learning*, pages 1206–1214. PMLR.

Andrychowicz, O. M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., et al. (2020). Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20.

Ardila, R., Branson, M., Davis, K., Henretty, M., Kohler, M., Meyer, J., Morais, R., Saunders, L., Tyers, F. M., and Weber, G. (2019a). Common voice: A massively-multilingual speech corpus. *arXiv preprint arXiv:1912.06670*.

Ardila, R., Branson, M., Davis, K., Henretty, M., Kohler, M., Meyer, J., Morais, R., Saunders, L., Tyers, F. M., and Weber, G. (2019b). Common voice: A massively-multilingual speech corpus. *arXiv preprint arXiv:1912.06670*.

Ayub, A. and Wagner, A. R. (2021). Eec: Learning to encode and regenerate images for continual learning. *arXiv preprint arXiv:2101.04904*.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Baby, D., D'Alterio, P., and Mendelev, V. (2022). Incremental learning for rnn-transducer based speech recognition models.

Bacon, P.-L., Harb, J., and Precup, D. (2017). The option-critic architecture. In *AAAI*, pages 1726–1734.

Baevski, A., Zhou, H., Mohamed, A., and Auli, M. (2020). wav2vec 2.0: A framework for self-supervised learning of speech representations. *arXiv preprint arXiv:2006.11477*.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Balaji, Y., Farajtabar, M., Yin, D., Mott, A., and Li, A. (2020). The effectiveness of memory replay in large scale continual learning. *arXiv preprint arXiv:2010.02418*.

Ball, P., Parker-Holder, J., Pacchiano, A., Choromanski, K., and Roberts, S. (2020). Ready policy one: World building through active learning. *International Conference on Machine Learning*.

Banayeeanzade, M., Mirzaiezadeh, R., Hasani, H., and Soleymani, M. (2021). Generative vs. discriminative: Rethinking the meta-continual learning. *Advances in Neural Information Processing Systems*, 34:21592–21604.

Barreto, A., Borsa, D., Hou, S., Comanici, G., Aygün, E., Hamel, P., Toyama, D. K., Hunt, J. J., Mourad, S., Silver, D., et al. (2019). The option keyboard: Combining skills in reinforcement learning.

Barreto, A., Dabney, W., Munos, R., Hunt, J. J., Schaul, T., Van Hasselt, H., and Silver, D. (2016). Successor features for transfer in reinforcement learning. *arXiv preprint arXiv:1606.05312*.

Barreto, A., Dabney, W., Munos, R., Hunt, J. J., Schaul, T., van Hasselt, H. P., and Silver, D. (2017). Successor features for transfer in reinforcement learning. *Advances in neural information processing systems*, 30.

Bell, S. J. and Lawrence, N. D. (2021). Behavioral experiments for understanding catastrophic forgetting. *arXiv preprint arXiv:2110.10570*.

Bell, S. J. and Lawrence, N. D. (2022). The effect of task ordering in continual learning. *arXiv preprint arXiv:2205.13323*.

Bellemare, M., Candido, S., Castro, P., Gong, J., Machado, M., Moitra, S., Ponda, S., and Wang, Z. (2020). Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588:77–82.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2012). The Arcade Learning Environment: An Evaluation Platform for General Agents. *CoRR*, abs/1207.4708.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.

Bellman, R. (1957). A Markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48.

Benjamin, A. S., Rolnick, D., and Kording, K. (2018). Measuring and regularizing networks in function space. *arXiv preprint arXiv:1805.08289*.

Benjamin, A. S., Rolnick, D., and Kording, K. P. (2019). Measuring and Regularizing Networks in Function Space. In *International Conference on Learning Representations*.

Benzing, F. (2020). Unifying regularisation methods for continual learning. *arXiv preprint arXiv:2006.06357*.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. In *Journal of Machine Learning Research*.

Berseth, G., Zhang, Z., Zhang, G., Finn, C., and Levine, S. (2021). Comps: Continual meta policy search. *arXiv preprint arXiv:2112.04467*.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

Bishop, C. M. and Svensén, M. (2012). Bayesian hierarchical mixtures of experts. *arXiv preprint arXiv:1212.2447*.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015a). Weight uncertainty in neural network. In *International Conference on Machine Learning*, pages 1613–1622. PMLR.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015b). Weight Uncertainty in Neural Networks. In *International Conference on Machine Learning*.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). OpenAI Gym.

Buzzega, P., Boschini, M., Porrello, A., Abati, D., and Calderara, S. (2020). Dark experience for general continual learning: a strong, simple baseline. *Advances in neural information processing systems*, 33:15920–15930.

Caccia, L., Aljundi, R., Belilovsky, E., Caccia, M., Charlin, L., and Tuytelaars, T. (2019). Online continual learning with maximally interfered retrieval. *Advances in Neural Information Processing (NeurIPS)*.

Caccia, M., Mueller, J., Kim, T., Charlin, L., and Fakoor, R. (2022). Task-agnostic continual reinforcement learning: In praise of a simple baseline. *arXiv preprint arXiv:2205.14495*.

Caccia, M., Rodriguez, P., Ostapenko, O., Normandin, F., Lin, M., Page-Caccia, L., Laradji, I. H., Rish, I., Lacoste, A., Vázquez, D., et al. (2020). Online fast adaptation and knowledge accumulation (osaka): a new approach to continual learning. *Advances in Neural Information Processing Systems*, 33.

Cesa-Bianchi, N. and Lugosi, G. (2006). *Prediction, learning, and games*. Cambridge university press.

Cha, S., Hsu, H., Hwang, T., Calmon, F. P., and Moon, T. (2020). Cpr: classifier-projection regularization for continual learning. *arXiv preprint arXiv:2006.07326*.

Chatzis, S. P. (2018). Indian Buffet Process deep generative models for semi-supervised classification. Technical report.

Chaudhry, A., Dokania, P. K., Ajanthan, T., and Torr, P. H. (2018a). Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 532–547.

Chaudhry, A., Facebook, M. R., Research, A. I., Elhoseiny, M., Ajanthan, T., Dokania, P. K., Torr, P. H. S., and Ranzato, M. . A. (2019). On Tiny Episodic Memories in Continual Learning. *arxiv.org:1902.10486*.

Chaudhry, A., Khan, N., Dokania, P., and Torr, P. (2020). Continual learning in low-rank orthogonal subspaces. *Advances in Neural Information Processing Systems*, 33:9900–9911.

Chaudhry, A., Ranzato, M., Rohrbach, M., and Elhoseiny, M. (2018b). Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*.

Chen, H.-J., Cheng, A.-C., Juan, D.-C., Wei, W., and Sun, M. (2020). Mitigating forgetting in online continual learning via instance-aware parameterization. *Advances in Neural Information Processing Systems*, 33:17466–17477.

Cheung, B., Terekhov, A., Chen, Y., Agrawal, P., and Olshausen, B. (2019). Superposition of many models into one. *Advances in neural information processing systems*, 32.

Chevalier-Boisvert, M., Willems, L., and Pal, S. (2018). Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`.

Chopin, N., Papaspiliopoulos, O., et al. (2020). *An introduction to sequential Monte Carlo*, volume 4. Springer.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. (2022). Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

Chrysakis, A. and Moens, M.-F. (2020). Online continual learning from imbalanced data. In *International Conference on Machine Learning*, pages 1952–1961. PMLR.

Chua, K., Calandra, R., McAllister, R., and Levine, S. (2018). Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems 31*, pages 4754–4765.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Chung, Y.-A., Hsu, W.-N., Tang, H., and Glass, J. (2019). An unsupervised autoregressive model for speech representation learning. *arXiv preprint arXiv:1904.03240*.

Ciftcioglu, Ö. and Türkcan, E. (1995). Adaptive training of feedforward neural networks by Kalman filtering.

Cobb, A. D. and Jalaian, B. (2021). Scaling Hamiltonian Monte Carlo Inference for Bayesian Neural Networks with Symmetric Splitting. *Uncertainty in Artificial Intelligence*.

Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. (2020). Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR.

Conneau, A., Baevski, A., Collobert, R., Mohamed, A., and Auli, M. (2020). Unsupervised cross-lingual representation learning for speech recognition. *arXiv preprint arXiv:2006.13979*.

De Lange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., Slabaugh, G., and Tuytelaars, T. (2019). A continual learning survey: Defying forgetting in classification tasks. *arXiv preprint arXiv:1909.08383*.

Degrave, J., Felici, F., Buchli, J., Neunert, M., Tracey, B., Carpanese, F., Ewalds, T., Hafner, R., Abdolmaleki, A., de Las Casas, D., et al. (2022). Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419.

Deisenroth, M. P., Faisal, A. A., and Ong, C. S. (2020). *Mathematics for machine learning*. Cambridge University Press.

Deng, D., Chen, G., Hao, J., Wang, Q., and Heng, P.-A. (2021). Flattening sharpness for dynamic gradient projection memory benefits continual learning. *Advances in Neural Information Processing Systems*, 34:18710–18721.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dikov, G., van der Smagt, P., and Bayer, J. (2019). Bayesian Learning of Neural Network Architectures. In *AISTATS*.

Dillon, J. V., Langmore, I., Tran, D., Brevdo, E., Vasudevan, S., Moore, D., Patton, B., Alemi, A., Hoffman, M., and Saurous, R. A. (2017). TensorFlow Distributions.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2016). Density estimation using real NVP. *arXiv preprint arXiv:1605.08803*.

Doshi, F., Miller, K., Van Gael, J., and Teh, Y. W. (2009). Variational inference for the Indian Buffet Process. In *Artificial Intelligence and Statistics*, pages 137–144.

Doshi-Velez, F., Miller, K. T., Van Gael, J., Teh, Y. W., Van, J., Yee, G. ., and Teh, W. (2009). Variational Inference for the Indian Buffet Process. Technical report.

Ebrahimi, S., Elhoseiny, M., Darrell, T., and Rohrbach, M. (2019). Uncertainty-guided continual learning with Bayesian neural networks. *arXiv preprint arXiv:1906.02425*.

Ehret, B., Henning, C., Cervera, M. R., Meulemans, A., Von Oswald, J., and Grewe, B. F. (2020). Continual learning in recurrent neural networks. *arXiv preprint arXiv:2006.12109*.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. (2018). IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. In *International Conference on Machine Learning*.

Eysenbach, B. and Levine, S. (2019). If maxent rl is the answer, what is the question?

Farajtabar, M., Azizan, N., Mott, A., and Li, A. (2020). Orthogonal gradient descent for continual learning. In *International Conference on Artificial Intelligence and Statistics*, pages 3762–3773. PMLR.

Farquhar, S. and Gal, Y. (2018). Towards robust evaluations of continual learning. *arXiv preprint arXiv:1805.09733*.

Farquhar, S., Osborne, M. A., and Gal, Y. (2020). Radial bayesian neural networks: Beyond discrete support in large-scale bayesian deep learning. In *International Conference on Artificial Intelligence and Statistics*, pages 1352–1362. PMLR.

Fernando, C., Banarse, D., Blundell, C., Zwols, Y., Ha, D., Rusu, A. A., Pritzel, A., and Wierstra, D. (2017). Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*.

Figurnov, M., Mohamed, S., and Mnih, A. (2018). Implicit Reparameterization Gradients. In *Neural Information Processing Systems*.

Filos, A., Farquhar, S., Gomez, A. N., Rudner, T. G., Kenton, Z., Smith, L., Alizadeh, M., de Kroon, A., and Gal, Y. (2019). A systematic comparison of bayesian deep learning robustness in diabetic retinopathy tasks. *arXiv preprint arXiv:1912.10481*.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR.

French, R. M. (1999). Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135.

Fu, H., Yu, S., Littman, M., and Konidaris, G. (2022). Model-based lifelong reinforcement learning with bayesian exploration. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*.

Gal, Y. and Ghahramani, Z. (2016). Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In *International Conference on Machine Learning*.

Gallardo, J., Hayes, T. L., and Kanan, C. (2021). Self-supervised training enhances online continual learning. *arXiv preprint arXiv:2103.14010*.

Garnelo, M., Rosenbaum, D., Maddison, C., Ramalho, T., Saxton, D., Shanahan, M., Teh, Y. W., Rezende, D., and Eslami, S. A. (2018). Conditional neural processes. In *International Conference on Machine Learning*, pages 1704–1713. PMLR.

Gaya, J.-B., Doan, T., Caccia, L., Soulier, L., Denoyer, L., and Raileanu, R. (2022). Building a subspace of policies for scalable continual learning. *arXiv preprint arXiv:2211.10445*.

Ghosh, S., Yao, J., and Doshi-Velez, F. (2019). Model Selection in Bayesian Neural Networks via Horseshoe Priors. *Journal of Machine Learning Research*, 20(182):1–46.

Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A., and Bengio, Y. (2013). An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*.

Goyal, A., Sodhani, S., Binas, J., Peng, X. B., Levine, S., and Bengio, Y. (2020). Reinforcement learning with competitive ensembles of information-constrained primitives. In *International Conference on Learning Representations*.

Graves, A. (2011). Practical variational inference for neural networks. *Advances in neural information processing systems*, 24.

Graves, A., Fernández, S., Gomez, F., and Schmidhuber, J. (2006). Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376.

Griffiths, T. L. and Ghahramani, Z. (2011). The Indian Buffet Process: An Introduction and Review. *Journal of Machine Learning Research*, 12:1185–1224.

Guo, Y., Liu, M., Yang, T., and Rosing, T. (2020). Improved schemes for episodic memory-based lifelong learning. *Advances in Neural Information Processing Systems*, 33:1023–1035.

Gupta, S. K., Phung, D., and Venkatesh, S. (2012). A Slice Sampler for Restricted Hierarchical Beta Process with Applications to Shared Subspace Learning. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, pages 316–325.

Ha, D. and Schmidhuber, J. (2018). World models. *arXiv preprint arXiv:1803.10122*.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018a). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018b). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1861–1870.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S. (2018c). Soft actor-critic algorithms and applications. *CoRR*, abs/1812.05905.

Hafner, D., Lillicrap, T., Norouzi, M., and Ba, J. (2020). Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*.

Hafner, D., Pasukonis, J., Ba, J., and Lillicrap, T. (2023). Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*.

Han, S., Liu, B., Cabezas, R., Twigg, C. D., Zhang, P., Petkau, J., Yu, T.-H., Tai, C.-J., Akbay, M., Wang, Z., et al. (2020). Megatrack: monochrome egocentric articulated hand-tracking for virtual reality. *ACM Transactions on Graphics (ToG)*, 39(4):87–1.

Harrison, J., Sharma, A., Finn, C., and Pavone, M. (2020). Continuous meta-learning without tasks. *Advances in neural information processing systems*, 33:17571–17581.

Hassabis, D., Kumaran, D., Summerfield, C., and Botvinick, M. (2017). Neuroscience-inspired artificial intelligence. *Neuron*, 95(2):245 – 258.

Hausknecht, M. and Stone, P. (2015). Deep recurrent q-learning for partially observable mdps. In *2015 aaai fall symposium series*.

He, X., Sygnowski, J., Galashov, A., Rusu, A. A., Teh, Y. W., and Pascanu, R. (2019). Task agnostic continual learning via meta learning. *arXiv preprint arXiv:1906.05201*.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018). Deep reinforcement learning that matters. *AAAI*.

Hendrycks, D. and Gimpel, K. (2016). Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*.

Henning, C., Cervera, M., D'Angelo, F., Von Oswald, J., Traber, R., Ehret, B., Kobayashi, S., Grewe, B. F., and Sacramento, J. (2021). Posterior meta-replay for continual learning. *Advances in Neural Information Processing Systems*, 34:14135–14149.

Hensman, J., Fusi, N., and Lawrence, N. D. (2013). Gaussian processes for big data. *arXiv preprint arXiv:1309.6835*.

Hergueux, J. and Kessler, S. (2022). Follow the leader: Technical and inspirational leadership in open source software. In *CHI Conference on Human Factors in Computing Systems*, pages 1–15.

Hernandez-Mena, C. D. (2019). TEDx Spanish Corpus. Audio and transcripts in Spanish taken from the TEDx Talks; shared under the CC BY-NC-ND 4.0 license. Web Download.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Deepmind, S. (2017). Rainbow: Combining Improvements in Deep Reinforcement Learning. In *AAAI*.

Hessel, M., Soyer, H., Espeholt, L., Czarnecki, W., Schmitt, S., and Van Hasselt, H. (2019). Multi-task deep reinforcement learning with popart. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3796–3803.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Ho, J. and Ermon, S. (2016). Generative adversarial imitation learning. *Advances in neural information processing systems*, 29.

Hoffman, M. D. and Blei, D. M. (2015). Structured Stochastic Variational Inference. In *International Conference on Artificial Intelligence and Statistics*.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. (2019). Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR.

Hsu, W.-N., Bolte, B., Tsai, Y.-H. H., Lakhotia, K., Salakhutdinov, R., and Mohamed, A. (2021). Hubert: Self-supervised speech representation learning by masked prediction of hidden units. *arXiv preprint arXiv:2106.07447*.

Hsu, Y.-C., Liu, Y.-C., Ramasamy, A., and Kira, Z. (2018a). Re-evaluating continual learning scenarios: A categorization and case for strong baselines. *arXiv preprint arXiv:1810.12488*.

Hsu, Y.-C., Liu, Y.-C., Ramasamy, A., and Kira, Z. (2018b). Re-evaluating Continual Learning Scenarios: A Categorization and Case for Strong Baselines. In *Continual Learning Workshop, 32nd Conference on Neural Information Processing Systems*.

Huang, Y., Xie, K., Bharadhwaj, H., and Shkurti, F. (2021). Continual model-based reinforcement learning with hypernetworks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 799–805. IEEE.

Isele, D. and Cosgun, A. (2018a). Selective Experience Replay for Lifelong Learning. In *AAAI*.

Isele, D. and Cosgun, A. (2018b). Selective experience replay for lifelong learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Izmailov, P., Vikram, S., Hoffman, M. D., and Wilson, A. G. (2021). What Are Bayesian Neural Network Posteriors Really Like? *arXiv preprint arXiv:2104.14421*.

Jacot, A., Gabriel, F., and Hongler, C. (2018). Neural tangent kernel: Convergence and generalization in neural networks. *arXiv preprint arXiv:1806.07572*.

Jang, E., Gu, S., and Poole, B. (2016). Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Jang, E., Gu, S., and Poole, B. (2017). Categorical Reparametrization with Gumbel-Softmax. In *International Conference on Learning Representations*.

Jankowiak, M. and Obermeyer, F. (2018). Pathwise derivatives beyond the reparameterization trick. In *International Conference on Machine Learning*, pages 2235–2244.

Javed, K. and White, M. (2019). Meta-learning representations for continual learning. *Advances in neural information processing systems*, 32.

Jin, X., Sadhu, A., Du, J., and Ren, X. (2021). Gradient-based editing of memory examples for online task-free continual learning. *Advances in Neural Information Processing Systems*, 34.

Jordan, M. I. and Jacobs, R. A. (1994). Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214.

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., Boyle, R., Cantin, P.-l., Chao, C., Clark, C., Coriell, J., Daley, M., Dau, M., Dean, J., Gelb, B., Ghaemmaghami, T. V., Gottipati, R., Gulland, W., Hagmann, R., Ho, C. R., Hogberg, D., Hu, J., Hundt, R., Hurt, D., Ibarz, J., Jaffey, A., Jaworski, A., Kaplan, A., Khaitan, H., Killebrew, D., Koch, A., Kumar, N., Lacy, S., Laudon, J., Law, J., Le, D., Leary, C., Liu, Z., Lucke, K., Lundin, A., MacKean, G., Maggiore, A., Mahony, M., Miller, K., Nagarajan, R., Narayanaswami, R., Ni, R., Nix, K., Norrie, T., Omernick, M., Penukonda, N., Phelps, A., Ross, J., Ross, M., Salek, A., Samadiani, E., Severn, C., Sizikov, G., Snelham, M., Souter, J., Steinberg, D., Swing, A., Tan, M., Thorson, G., Tian, B., Toma, H., Tuttle, E., Vasudevan, V., Walter, R., Wang, W., Wilcox, E., and Yoon, D. H. (2017). In-datacenter performance analysis of a tensor processing unit. *SIGARCH Comput. Archit. News*, 45(2):1–12.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., et al. (2021). Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134.

Kahn, J., Rivière, M., Zheng, W., Kharitonov, E., Xu, Q., Mazaré, P.-E., Karadayi, J., Liptchinsky, V., Collobert, R., Fuegen, C., et al. (2020). Libri-light: A benchmark for asr with limited or no supervision. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7669–7673. IEEE.

Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., et al. (2019). Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems.

Kao, T.-C., Jensen, K. T., van de Ven, G. M., Bernacchia, A., and Hennequin, G. (2021). Natural continual learning: success is a journey, not (just) a destination. In *Thirty-Fifth Conference on Neural Information Processing Systems*.

Kaplanis, C., Shanahan, M., and Clopath, C. (2018). Continual reinforcement learning with complex synapses. In *International Conference on Machine Learning*, pages 2497–2506. PMLR.

Kapoor, S., Karaletsos, T., and Bui, T. D. (2021). Variational auto-regressive Gaussian processes for continual learning. In *International Conference on Machine Learning*, pages 5290–5300. PMLR.

Kessler, S., Cobb, A., Rudner, T. G., Zohren, S., and Roberts, S. J. (2023). On sequential bayesian inference for continual learning. *arXiv preprint arXiv:2301.01828*.

Kessler, S., Miłoś, P., Parker-Holder, J., and Roberts, S. J. (2022a). The surprising effectiveness of latent world models for continual reinforcement learning. *arXiv preprint arXiv:2211.15944*.

Kessler, S., Nguyen, V., Zohren, S., and Roberts, S. J. (2021a). Hierarchical indian buffet neural networks for bayesian continual learning. In *Uncertainty in Artificial Intelligence*, pages 749–759. PMLR.

Kessler, S., Parker-Holder, J., Ball, P., Zohren, S., and Roberts, S. J. (2021b). Same state, different task: Continual reinforcement learning without interference. *arXiv preprint arXiv:2106.02940*.

Kessler, S., Parker-Holder, J., Ball, P., Zohren, S., and Roberts, S. J. (2022b). Same state, different task: Continual reinforcement learning without interference. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 7143–7151.

Kessler, S., Salas, A., Tan, V. W., Zohren, S., and Roberts, S. (2018). Practical bayesian learning of neural networks via adaptive optimisation methods. *arXiv preprint arXiv:1811.03679*.

Kessler, S., Thomas, B., and Karout, S. (2022c). An adapter based pre-training for efficient and scalable self-supervised speech representation learning. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3179–3183. IEEE.

Khetarpal, K., Riemer, M., Rish, I., and Precup, D. (2020). Towards continual reinforcement learning: A review and perspectives. *arXiv preprint arXiv:2012.13490*.

Kingma, D. P. and Lei Ba, J. (2015). ADAM: A Method for Stochastic Optimization. In *International Conference on Learning Representations*.

Kingma, D. P., Salimans, T., and Welling, M. (2015). Variational Dropout and the Local Reparameterization Trick. In *Neural Information Processing Systems*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational Bayes. *CoRR*, abs/1312.6114.

Kingma, D. P. and Welling, M. (2014). Auto-Encoding Variational Bayes. In *International Conference on Learning Representations*.

Kirk, R., Zhang, A., Grefenstette, E., and Rocktäschel, T. (2021). A survey of generalisation in deep reinforcement learning. *arXiv preprint arXiv:2111.09794*.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N. C., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. (2016). Overcoming catastrophic forgetting in neural networks. *CoRR*, abs/1612.00796.

Klissarov, M. and Machado, M. C. (2023). Deep laplacian-based options for temporally-extended exploration.

Knoblauch, J., Husain, H., and Diethe, T. (2020). Optimal continual learning has perfect memory and is NP-hard. In *International Conference on Machine Learning*, pages 5327–5337. PMLR.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90.

Kumar, A., Chatterjee, S., and Rai, P. (2019). Bayesian structure adaptation for continual learning. *arXiv preprint arXiv:1912.03624*.

Kumar, A., Chatterjee, S., and Rai, P. (2021). Bayesian structural adaptation for continual learning. In *International Conference on Machine Learning*, pages 5850–5860. PMLR.

Kurle, R., Cseke, B., Klushyn, A., Van Der Smagt, P., and Günnemann, S. (2019). Continual learning with bayesian neural networks for non-stationary data. In *International Conference on Learning Representations*.

Küttler, H., Nardelli, N., Miller, A. H., Raileanu, R., Selvatici, M., Grefenstette, E., and Rocktäschel, T. (2020). The NetHack Learning Environment. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017a). Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017b). Simple and scalable predictive uncertainty estimation using deep ensembles. In *Neural Information Processing Systems*.

Lavda, F., Ramapuram, J., Gregorova, M., and Kalousis, A. (2018). Continual classification learning using generative models. *arXiv preprint arXiv:1810.10612*.

Lazaridou, A., Kuncoro, A., Gribovskaya, E., Agrawal, D., Liska, A., Terzi, T., Gimenez, M., de Masson d'Autume, C., Kocisky, T., Ruder, S., et al. (2021). Mind the gap: Assessing temporal generalization in neural language models. *Advances in Neural Information Processing Systems*, 34:29348–29363.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lee, S., Ha, J., Zhang, D., and Kim, G. (2020a). A Neural Dirichlet Process Misture Model for Task-Free Continaul Learning. In *International Conference on Learning Representations*.

Lee, S., Ha, J., Zhang, D., and Kim, G. (2020b). A Neural Dirichlet Process Mixture Model for Task-Free Continual Learning. In *International Conference on Learning Representations*.

Lee, S., Ha, J., Zhang, D., and Kim, G. (2020c). A neural dirichlet process mixture model for task-free continual learning. *arXiv preprint arXiv:2001.00689*.

Levine, S., Kumar, A., Tucker, G., and Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*.

Li, Z. and Hoiem, D. (2017a). Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947.

Li, Z. and Hoiem, D. (2017b). Learning without Forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Lim, B., Zohren, S., and Roberts, S. (2019). Enhancing time-series momentum strategies using deep neural networks. *The Journal of Financial Data Science*, 1(4):19–38.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Mach. Learn.*, 8(3–4):293–321.

Lin, S., Ju, P., Liang, Y., and Shroff, N. (2023). Theory on forgetting and generalization of continual learning. *arXiv preprint arXiv:2302.05836*.

Lin, X., Zhen, H.-L., Li, Z., Zhang, Q., and Kwong, S. (2019). Pareto multi-task learning. *arXiv preprint arXiv:1912.12854*.

Liu, Y., Schiele, B., and Sun, Q. (2021). Rmm: Reinforced memory management for class-incremental learning. *Advances in Neural Information Processing Systems*, 34:3478–3490.

Lomonaco, V., Desai, K., Culurciello, E., and Maltoni, D. (2020). Continual reinforcement learning in 3d non-stationary environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 248–249.

Lomonaco, V. and Maltoni, D. (2017). Core50: a new dataset and benchmark for continuous object recognition. In *Conference on Robot Learning*, pages 17–26. PMLR.

Loo, N., Swaroop, S., and Turner, R. E. (2020). Generalized variational continual learning. *arXiv preprint arXiv:2011.12328*.

Lopez-Paz, D. and Ranzato, M. (2017). Gradient episodic memory for continual learning. *arXiv preprint arXiv:1706.08840*.

Louizos, C., Ullrich, K., and Welling, M. (2017). Bayesian Compression for Deep Learning. In *Neural Information Processing Systems*.

Lu, K., Grover, A., Abbeel, P., and Mordatch, I. (2021). Pretrained transformers as universal computation engines. *arXiv preprint arXiv:2103.05247*.

Lyle, C., Rowland, M., and Dabney, W. (2022). Understanding and preventing capacity loss in reinforcement learning. *arXiv preprint arXiv:2204.09560*.

Lyle, C., Zheng, Z., Nikishin, E., Pires, B. A., Pascanu, R., and Dabney, W. (2023). Understanding plasticity in neural networks. *arXiv preprint arXiv:2303.01486*.

MacKay, D. J. (1992a). A practical Bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472.

MacKay, D. J. (1992b). The evidence framework applied to classification networks. *Neural computation*, 4(5):720–736.

Maddison, C. J., Mnih, A., and Teh, Y. W. (2016). The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*.

Maddison, C. J., Mnih, A., and Teh, Y. W. (2017). The Concrete Distribution: a Continual Relaxation of Discrete Random Variables. In *International Conference on Learning Representations*.

Mania, H., Guy, A., and Recht, B. (2018). Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*.

Mankowitz, D. J., Žídek, A., Barreto, A., Horgan, D., Hessel, M., Quan, J., Oh, J., van Hasselt, H., Silver, D., and Schaul, T. (2018). Unicorn: Continual learning with a universal, off-policy agent. *arXiv preprint arXiv:1802.08294*.

Manousakas, D., Ritter, H., and Karaletsos, T. (2022). Black-box coreset variational inference. *arXiv preprint arXiv:2211.02377*.

Massiceti, D., Zintgraf, L., Bronskill, J., Theodorou, L., Harris, M. T., Cutrell, E., Morrison, C., Hofmann, K., and Stumpf, S. (2021). Orbit: A real-world few-shot dataset for teachable object recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10818–10828.

McCloskey, M. and Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier.

Mehta, N., Liang, K., Verma, V. K., and Carin, L. (2021). Continual learning using a bayesian nonparametric dictionary of weight factors. In *International Conference on Artificial Intelligence and Statistics*, pages 100–108. PMLR.

Mehta, N., Liang, K. J., and Carin, L. (2020). Bayesian Nonparametric Weight Factorization for Continual Learning. Technical report.

Mendez, J., Wang, B., and Eaton, E. (2020). Lifelong policy gradient learning of factored policies for faster training without forgetting. *Advances in Neural Information Processing Systems*, 33:14398–14409.

Mendez, J. A., van Seijen, H., and Eaton, E. (2022). Modular lifelong reinforcement learning via neural composition. *arXiv preprint arXiv:2207.00429*.

Mermillod, M., Bugaiska, A., and Bonin, P. (2013). The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mirzadeh, S. I., Chaudhry, A., Yin, D., Hu, H., Pascanu, R., Gorur, D., and Farajtabar, M. (2022). Wide neural networks forget less catastrophically. In *International Conference on Machine Learning*, pages 15699–15717. PMLR.

Mirzadeh, S. I., Farajtabar, M., Gorur, D., Pascanu, R., and Ghasemzadeh, H. (2020a). Linear mode connectivity in multitask and continual learning. *arXiv preprint arXiv:2010.04495*.

Mirzadeh, S. I., Farajtabar, M., Pascanu, R., and Ghasemzadeh, H. (2020b). Understanding the role of training regimes in continual learning. *Advances in Neural Information Processing Systems*, 33:7308–7320.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*.

Mohamed, A., Okhonko, D., and Zettlemoyer, L. (2019a). Transformers with convolutional context for asr. *arXiv preprint arXiv:1904.11660*.

Mohamed, S., Rosca, M., Figurnov, M., and Mnih, A. (2019b). Monte carlo gradient estimation in machine learning. *ArXiv*, abs/1906.10652.

Molchanov, D., Ashukha, A., and Vetrov, D. (2017). Variational Dropout Sparsifies Deep Neural Networks. In *International Conference on Machine Learning*.

Nagabandi, A., Finn, C., and Levine, S. (2018). Deep online learning via meta-learning: Continual adaptation for model-based rl. *arXiv preprint arXiv:1812.07671*.

Nalisnick, E. and Smyth, P. (2017). Stick-Breaking Variational Autoencoders. In *International Conference on Learning Representations*.

Neal, R. M. et al. (2011). MCMC using Hamiltonian dynamics. *Handbook of Markov chain Monte Carlo*, 2(11):2.

Nekoei, H., Badrinaaraayanan, A., Courville, A., and Chandar, S. (2021). Continuous coordination as a realistic scenario for lifelong learning.

Nguyen, C. V., Li, Y., Bui, T. D., and Turner, R. E. (2017). Variational continual learning. *arXiv preprint arXiv:1710.10628*.

Nguyen, V., Orbell, S., Lennon, D. T., Moon, H., Vigneau, F., Camenzind, L. C., Yu, L., Zumbühl, D. M., Briggs, G. A. D., Osborne, M. A., et al. (2021). Deep reinforcement learning for efficient measurement of quantum devices. *npj Quantum Information*, 7(1):1–9.

Nix, D. A. and Weigend, A. S. (1994). Estimating the mean and variance of the target probability distribution. In *IEEE International Conference on Neural Networks - Conference Proceedings*, volume 1, pages 55–60.

Oord, A. v. d., Li, Y., and Vinyals, O. (2018). Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*.

OpenAI, Andrychowicz, M., Baker, B., Chociej, M., Józefowicz, R., McGrew, B., Pachocki, J. W., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L., and Zaremba, W. (2018). Learning dexterous in-hand manipulation. *CoRR*, abs/1808.00177.

Opper, M. and Winther, O. (1998). A Bayesian approach to on-line learning. *On-line learning in neural networks*, pages 363–378.

Osawa, K., Swaroop, S., Jain, A., Eschenhagen, R., Turner, R. E., Yokota, R., and Emtiyaz Khan, M. (2019). Practical Deep Learning with Bayesian Principles. In *Neural Information Processing Systems*.

Ostapenko, O., Rodriguez, P., Caccia, M., and Charlin, L. (2021). Continual learning via local module composition. *Advances in Neural Information Processing Systems*, 34.

Ott, M., Edunov, S., Baevski, A., Fan, A., Gross, S., Ng, N., Grangier, D., and Auli, M. (2019). fairseq: A fast, extensible toolkit for sequence modeling. *arXiv preprint arXiv:1904.01038*.

Ovadia, Y., Fertig, E., Ren, J., Nado, Z., Sculley, D., Nowozin, S., Dillon, J., Lakshminarayanan, B., and Snoek, J. (2019a). Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. *Advances in neural information processing systems*, 32.

Ovadia, Y., Fertig, E., Research, G., Ren, J., Nado Google Research, Z., Nowozin Google Research, S., Dillon Google Research, J. V., Lakshminarayanan, B., and Snoek, J. (2019b). Can You Trust Your Model's Uncertainty? Evaluating Predictive Uncertainty Under Dataset Shift. In *Neurips*.

Pan, P., Swaroop, S., Immer, A., Eschenhagen, R., Turner, R. E., and Khan, M. E. (2020). Continual deep learning by functional regularisation of memorable past. *arXiv preprint arXiv:2004.14070*.

Panayotov, V., Chen, G., Povey, D., and Khudanpur, S. (2015a). Librispeech: an asr corpus based on public domain audio books. In *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5206–5210. IEEE.

Panayotov, V., Chen, G., Povey, D., and Khudanpur, S. (2015b). Librispeech: an asr corpus based on public domain audio books. In *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5206–5210. IEEE.

Panousis, K. P., Chatzis, S., and Theodoridis, S. (2019). Nonparametric Bayesian deep networks with local competition. *36th International Conference on Machine Learning, ICML 2019*, 2019-June(May):8772–8783.

Parker-Holder, J., Jiang, M., Dennis, M., Samvelyan, M., Foerster, J., Grefenstette, E., and Rocktäschel, T. (2022). Evolving curricula with regret-based environment design. *arXiv preprint arXiv:2203.01302*.

Pascual, S., Ravanelli, M., Serra, J., Bonafonte, A., and Bengio, Y. (2019). Learning problem-agnostic speech representations from multiple self-supervised tasks. *arXiv preprint arXiv:1904.03416*.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR.

Petersen, K. B., Pedersen, M. S., et al. (2008). The matrix cookbook. *Technical University of Denmark*, 7(15):510.

Pfeiffer, J., Kamath, A., Rücklé, A., Cho, K., and Gurevych, I. (2020a). Adapterfusion: Non-destructive task composition for transfer learning. *arXiv preprint arXiv:2005.00247*.

Pfeiffer, J., Vulić, I., Gurevych, I., and Ruder, S. (2020b). Mad-x: An adapter-based framework for multi-task cross-lingual transfer. *arXiv preprint arXiv:2005.00052*.

Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., Schneider, J., Tobin, J., Chociej, M., Welinder, P., et al. (2018). Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*.

Pomerleau, D. A. (1991). Efficient training of artificial neural networks for autonomous navigation. *Neural computation*, 3(1):88–97.

Powers, S., Xing, E., Kolve, E., Mottaghi, R., and Gupta, A. (2021). Cora: Benchmarks, baselines, and metrics as a platform for continual reinforcement learning agents. *arXiv preprint arXiv:2110.10067*.

Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C., and Sutskever, I. (2022). Robust speech recognition via large-scale weak supervision. *arXiv preprint arXiv:2212.04356*.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Raghavan, K. and Balaprakash, P. (2021). Formalizing the generalization-forgetting trade-off in continual learning. *Advances in Neural Information Processing Systems*, 34:17284–17297.

Rajasegaran, J., Hayat, M., Khan, S., Khan, F. S., Shao, L., and Yang, M.-H. (2019). An adaptive random path selection approach for incremental learning. *arXiv preprint arXiv:1906.01120*.

Rajendran, J., Irpan, A., and Jang, E. (2020). Meta-learning requires meta-augmentation. In *Advances in Neural Information Processing Systems 33*.

Ramasesh, V. V., Dyer, E., and Raghu, M. (2020). Anatomy of catastrophic forgetting: Hidden representations and task semantics. *arXiv preprint arXiv:2007.07400*.

Ranganath, R., Gerrish, S., and Blei, D. M. (2014). Black Box Variational Inference. In *Artificial Intelligence and Statistics*.

Rao, D., Visin, F., Rusu, A. A., Teh, Y. W., Pascanu, R., and Hadsell, R. (2019). Continual Unsupervised Representation Learning. In *Neural Information Processing Systems*.

Rebuffi, S.-A., Bilen, H., and Vedaldi, A. (2017a). Learning multiple visual domains with residual adapters. *arXiv preprint arXiv:1705.08045*.

Rebuffi, S.-A., Kolesnikov, A., Sperl, G., and Lampert, C. H. (2017b). ICARL: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788.

Riemer, M., Liu, M., and Tesauro, G. (2018). Learning abstract options. *arXiv preprint arXiv:1810.11583*.

Ring, M. B. (1994). *Continual learning in reinforcement environments*. PhD thesis, University of Texas at Austin.

Ritter, H., Botev, A., and Barber, D. (2018). Online structured Laplace approximations for overcoming catastrophic forgetting. *arXiv preprint arXiv:1805.07810*.

Robert M. French (1999). Catastrophic forgetting in connectionists networks. *Trends in Cognitive Sciences*, 3(4):128–135.

Rolnick, D., Ahuja, A., Schwarz, J., Lillicrap, T., and Wayne, G. (2019). Experience replay for continual learning. In *Advances in Neural Information Processing Systems 32*, pages 350–360.

Rudner, T. G. J., Chen, Z., Teh, Y. W., and Gal, Y. (2022a). Tractabe Function-Space Variational Inference in Bayesian Neural Networks.

Rudner, T. G. J., Smith, F. B., Feng, Q., Teh, Y. W., and Gal, Y. (2022b). Continual Learning via Sequential Function-Space Variational Inference. In *Proceedings of the 38th International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016a). Progressive neural networks. *arXiv preprint arXiv:1606.04671*.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016b). Progressive neural networks. *CoRR*, abs/1606.04671.

Sabour, S., Frosst, N., and Hinton, G. E. (2017). Dynamic routing between capsules. *arXiv preprint arXiv:1710.09829*.

Saha, G., Garg, I., and Roy, K. (2021). Gradient projection memory for continual learning. *arXiv preprint arXiv:2103.09762*.

Samvelyan, M., Kirk, R., Kurin, V., Parker-Holder, J., Jiang, M., Hambro, E., Petroni, F., Kuttler, H., Grefenstette, E., and Rocktäschel, T. (2021). Minihack the planet: A sandbox for open-ended reinforcement learning research. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. (2015). Universal value function approximators. In *International conference on machine learning*, pages 1312–1320. PMLR.

Scheirer, W. J., de Rezende Rocha, A., Sapkota, A., and Boult, T. E. (2012). Toward open set recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(7):1757–1772.

Schneider, S., Baevski, A., Collobert, R., and Auli, M. (2019). wav2vec: Unsupervised pre-training for speech recognition. *arXiv preprint arXiv:1904.05862*.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T. P., and Silver, D. (2019). Mastering atari, go, chess and shogi by planning with a learned model. *CoRR*, abs/1911.08265.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms.

Schwarz, J., Czarnecki, W., Luketina, J., Grabska-Barwinska, A., Teh, Y. W., Pascanu, R., and Hadsell, R. (2018a). Progress & compress: A scalable framework for continual learning. In Dy, J. G. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4535–4544. PMLR.

Schwarz, J., Czarnecki, W., Luketina, J., Grabska-Barwinska, A., Teh, Y. W., Pascanu, R., and Hadsell, R. (2018b). Progress & compress: A scalable framework for continual learning. In *International Conference on Machine Learning*, pages 4528–4537. PMLR.

Sekar, R., Rybkin, O., Daniilidis, K., Abbeel, P., Hafner, D., and Pathak, D. (2020). Planning to explore via self-supervised world models. In *International Conference on Machine Learning*, pages 8583–8592. PMLR.

Shi, G., Chen, J., Zhang, W., Zhan, L.-M., and Wu, X.-M. (2021). Overcoming catastrophic forgetting in incremental few-shot learning by finding flat minima. *Advances in Neural Information Processing Systems*, 34:6747–6761.

Shi, X., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-K., and Woo, W.-c. (2015). Convolutional lstm network: A machine learning approach for precipitation nowcasting. *Advances in neural information processing systems*, 28.

Shin, H., Lee, J. K., Kim, J., and Kim, J. (2017). Continual learning with deep generative replay. *Advances in neural information processing systems*, 30.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T. P., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016a). Mastering the game of go with deep neural networks and tree search. *Nat.*, 529(7587):484–489.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016b). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

Singh, P., Verma, V. K., Mazumder, P., Carin, L., and Rai, P. (2020). Calibrating cnns for lifelong learning. *Advances in Neural Information Processing Systems*, 33:15579–15590.

Singh, R., Ling, J., and Doshi-Velez, F. (2017). Structured Variational Autoencoders for the Beta-Bernoulli Process. In *Workshop on Advances in Approximate Bayesian Inference, Neural Information Processing Systems*.

Snell, J., Swersky, K., and Zemel, R. S. (2017). Prototypical networks for few-shot learning. *arXiv preprint arXiv:1703.05175*.

Sodhani, S., Faramarzi, M., Mehta, S. V., Malviya, P., Abdelsalam, M., Janarthanan, J., and Chandar, S. (2022). An introduction to lifelong supervised learning. *arXiv preprint arXiv:2207.04354*.

Steinparz, C., Schmied, T., Paischer, F., Dinu, M.-C., Patil, V., Bitto-Nemling, A., Eghbal-zadeh, H., and Hochreiter, S. (2022). Reactive exploration to cope with non-stationarity in lifelong reinforcement learning. *arXiv preprint arXiv:2207.05742*.

Sutton, R. S. (1991). Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., Barto, A. G., et al. (1998). *Introduction to reinforcement learning*, volume 135. MIT press Cambridge.

Sutton, R. S., Precup, D., and Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211.

Swaroop, S., Nguyen, C. V., Bui, T. D., and Turner, R. E. (2019). Improving and understanding variational continual learning. *arXiv preprint arXiv:1905.02099*.

Tang, B. and Matteson, D. S. (2020). Graph-based continual learning. *arXiv preprint arXiv:2007.04813*.

Tao, X., Hong, X., Chang, X., Dong, S., Wei, X., and Gong, Y. (2020). Few-shot class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12183–12192.

Team, O. E. L., Stooke, A., Mahajan, A., Barros, C., Deck, C., Bauer, J., Sygnowski, J., Trebacz, M., Jaderberg, M., Mathieu, M., et al. (2021). Open-ended learning leads to generally capable agents. *arXiv preprint arXiv:2107.12808*.

Teh, Y. W., Grür, D., and Ghahramani, Z. (2007). Stick-breaking Construction for the Indian Buffet Process. In *International Conference on Artificial Intelligence and Statistics*.

Thibaux, R. and Jordan, M. I. (2007). Hierarchical Beta processes and the Indian buffet process. In *Artificial Intelligence and Statistics*, pages 564–571.

Thomas, B., Kessler, S., and Karout, S. (2022). Efficient adapter transfer of self-supervised speech models for automatic speech recognition. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7102–7106. IEEE.

Thrun, S. and Mitchell, T. M. (1995a). Lifelong robot learning. *Robotics and Autonomous Systems*, 15(1):25–46. The Biology and Technology of Intelligent Autonomous Agents.

Thrun, S. and Mitchell, T. M. (1995b). Lifelong robot learning. *Robotics and autonomous systems*, 15(1-2):25–46.

Titsias, M. (2009). Variational learning of inducing variables in sparse gaussian processes. In *Artificial intelligence and statistics*, pages 567–574. PMLR.

Titsias, M. K., Schwarz, J., Matthews, A. G. d. G., Pascanu, R., and Teh, Y. W. (2020a). Functional regularisation for continual learning. *International Conference on Learning Representations*.

Titsias, M. K., Schwarz, J., Matthews, A. G. d. G., Pascanu, R., and Teh, Y. W. (2020b). Functional regularisation for continual learning with gaussian processes. In *ICLR*.

van de Ven, G. M., Li, Z., and Tolias, A. S. (2021). Class-incremental learning with generative classifiers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3611–3620.

van de Ven, G. M., Siegelmann, H. T., and Tolias, A. S. (2020). Brain-inspired replay for continual learning with artificial neural networks. *Nature communications*, 11(1):1–14.

van de Ven, G. M. and Tolias, A. S. (2018). Three scenarios for continual learning. In *NeurIPS Continual Learning workshop*.

Van de Ven, G. M. and Tolias, A. S. (2019). Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*.

van de Ven, G. M., Tuytelaars, T., and Tolias, A. S. (2022). Three types of incremental learning. *Nature Machine Intelligence*, pages 1–13.

Van Den Oord, A., Vinyals, O., et al. (2017). Neural discrete representation learning. *Advances in neural information processing systems*, 30.

Van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(11).

Van Hasselt, H., Guez, A., and Silver, D. (2015). Deep Reinforcement Learning with Double Q-learning. In *AAAI*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *arXiv preprint arXiv:1706.03762*.

Veniat, T., Denoyer, L., and Ranzato, M. (2020). Efficient continual learning with modular networks and task-driven priors. *arXiv preprint arXiv:2012.12631*.

Verwimp, E., Yang, K., Parisot, S., Lanqing, H., McDonagh, S., Pérez-Pellitero, E., De Lange, M., and Tuytelaars, T. (2022). Clad: A realistic continual learning benchmark for autonomous driving. *arXiv preprint arXiv:2210.03482*.

Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al. (2016). Matching networks for one shot learning. *Advances in neural information processing systems*, 29.

Vitter, J. S. (1985). Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11:37–57.

Von Oswald, J., Henning, C., Sacramento, J., and Grewe, B. F. (2020). Continual Learning with Hypernetworks. In *International Conference on Learning Representations*.

Wang, L., Zhang, M., Jia, Z., Li, Q., Bao, C., Ma, K., Zhu, J., and Zhong, Y. (2021). Afec: Active forgetting of negative transfer in continual learning. *Advances in Neural Information Processing Systems*, 34:22379–22391.

Wang, L., Zhang, X., Su, H., and Zhu, J. (2023). A comprehensive survey of continual learning: Theory, method and application. *arXiv preprint arXiv:2302.00487*.

Wang, R., Lehman, J., Clune, J., and Stanley, K. O. (2019). Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv preprint arXiv:1901.01753*.

Wang, Z., Schaul, T., Hessel, M., Lanctot, M., and de Freitas, N. (2016). Dueling Network Architectures for Deep Reinforcement Learning Hado van Hasselt. In *ICML*.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3):279–292.

Wilson, A. G. and Izmailov, P. (2020). Bayesian deep learning and a probabilistic perspective of generalization. *Advances in neural information processing systems*, 33:4697–4708.

Winther, O. and Solla, S. A. (1998). Optimal Bayesian online learning. *Theoretical Aspects of Neural Computation (TANC-97), KYM Wong, I. King and D.-Y. Yeung eds., Springer Verlag, Singapore*.

Wolczyk, M., Zajac, M., Pascanu, R., Kucinski, L., and Milos, P. (2021). Continual world: A robotic benchmark for continual reinforcement learning. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 28496–28510.

Wołczyk, M., Zając, M., Pascanu, R., Kuciński, Ł., and Miłoś, P. (2022). Disentangling transfer in continual reinforcement learning. *arXiv preprint arXiv:2209.13900*.

Wood, K., Kessler, S., Roberts, S. J., and Zohren, S. (2023). Few-shot learning patterns in financial time-series for trend-following strategies. *arXiv preprint arXiv:2310.10500*.

Wood, K., Roberts, S., and Zohren, S. (2022). Slow momentum with fast reversion: A trading strategy using deep learning and changepoint detection. *The Journal of Financial Data Science*, 4(1):111–129.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

Xie, A., Harrison, J., and Finn, C. (2020). Deep reinforcement learning amidst lifelong non-stationarity. *arXiv preprint arXiv:2006.10701*.

Xu, J. and Zhu, Z. (2018). Reinforced continual learning. *arXiv preprint arXiv:1805.12369*.

Xu, K., Srivastava, A., and Sutton, C. (2019). Variational Russian Roulette for Deep Bayesian Nonparametrics. In *Proceedings of the 36th International Conference on Machine Learning*.

Yang, Y. and Hospedales, T. (2023). Partial index tracking: A meta-learning approach. In *Conference on Lifelong Learning Agents*, pages 415–436. PMLR.

Yap, P., Ritter, H., and Barber, D. (2021). Addressing catastrophic forgetting in few-shot problems. In *International Conference on Machine Learning*, pages 11909–11919. PMLR.

Yoon, J., Yang, E., Lee, J., and Hwang, S. J. (2017). Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*.

Yu, T., Kumar, S., Gupta, A., Levine, S., Hausman, K., Finn, C., University, S., Berkeley, U. C., and At Google, R. (2020a). Gradient Surgery for Multi-Task Learning. In *Advances in Neural Information Processing Systems*.

Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. (2020b). Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, pages 1094–1100. PMLR.

Zenke, F., Poole, B., and Ganguli, S. (2017a). Continual learning through synaptic intelligence. In *International conference on machine learning*, pages 3987–3995. PMLR.

Zenke, F., Poole, B., and Ganguli, S. (2017b). Continual Learning Through Synaptic Intelligence. In *International Conference on Machine Learning*.

Zeno, C., Golan, I., Hoffer, E., and Soudry, D. (2018). Task agnostic continual learning using online variational bayes. *arXiv preprint arXiv:1803.10123*.

Zhang, Q., Fang, J., Meng, Z., Liang, S., and Yilmaz, E. (2021a). Variational continual bayesian meta-learning. *Advances in Neural Information Processing Systems*, 34:24556–24568.

Zhang, X., Meng, D., Gouk, H., and Hospedales, T. M. (2021b). Shallow bayesian meta learning for real-world few-shot recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 651–660.

10
Supplementary Material

Table of Contents

10.1 Hierarchical Indian Buffet Neural Networks for Bayesian Continual Learning

10.1.1 Model and Inference Details

In this section, we present the variational BNN with the structure of the hidden layer determined by the H-IBP variational posterior (a.k.a. HIBNN). It is straightforward to apply the following methodology for a simpler model with independent IBP variational posteriors determining the structure of each hidden layer (referred to as the IBNN in the main paper).

We derive a structured variational posterior where dependencies are established between global parameters and local parameters (Hoffman and Blei, 2015). Once the variational posterior is obtained, we can follow the VCL framework (Nguyen et al., 2017) for CL. The following set of equations governs the stick-breaking H-IBP model for an arbitrary layer $j \in \{1, ..., J\}$ with k neurons in each layer of a BNN:

$$v_k \sim \text{Beta}\left(\alpha, 1\right), \qquad \text{for } k = 1, \ldots, \infty, \tag{10.1.1.1}$$

$$\pi_k^0 = \prod_{i=1}^{k} v_i, \qquad \forall\, k, \tag{10.1.1.2}$$

$$\pi_{jk} \sim \text{Beta}(\alpha_j \pi_k^0, \alpha_j(1 - \pi_k^0)), \qquad \forall\, k, j = 1, \ldots, J, \tag{10.1.1.3}$$

$$z_{ijk} \sim \text{Bern}(\pi_{jk}), \qquad \forall\, j, k, i = 1, \ldots, N, \tag{10.1.1.4}$$

$$\boldsymbol{w}_{jk} \sim \mathcal{N}(\boldsymbol{\mu}_{jk}, \boldsymbol{\sigma}_{jk}^2 \mathbb{1}), \qquad \forall\, j, k, \tag{10.1.1.5}$$

$$h_{jk} = f(\boldsymbol{h}_{j-1}\boldsymbol{w}_{jk}) \circ z_{ijk} \qquad \forall\, i, j, k. \tag{10.1.1.6}$$

The index k denotes a particular neuron, j denotes a particular layer, and $\boldsymbol{w}_{jk}$ denotes a column (of W_j) from the weight matrix mapping hidden states from layer $j-1$ to layer j, such that $h_j = f(h_{j-1}W_j) \circ Z_j$. We denote $\circ$ as the elementwise multiplication operation. The binary matrix Z_j controls the inclusion of a particular neuron in layer j. The dimensionality of our variables are as follows $\boldsymbol{w}_{jk} \in \mathbb{R}^{k_{j-1}}$, $h_{j-1} \in \mathbb{R}^{k_{j-1}}$, $h_j \in \mathbb{R}^{k_j}$ and $z_{ijk} \in \mathbb{Z}_2 = \{0, 1\}$.

The closed-form solution to the true posterior of the H-IBP parameters and BNN weights involves integrating over the joint distribution of the data and hidden variables, $\phi = \{\boldsymbol{Z}, \boldsymbol{\pi}, \boldsymbol{\pi^0}, \boldsymbol{w}\}$. Since it is not possible to obtain a closed-form

solution to this integral, variational inference together with reparameterizations of the variational distributions are used (Kingma and Welling, 2013; Figurnov et al., 2018) to employ gradient-based methods. The variational approximation used is

$$q(\phi) = \prod_{k=1}^{K} q(v_k^0; \alpha_k^0, \beta_k^0) \prod_{j=1}^{J} q(\pi_{jk}|v_k^0) q(\boldsymbol{w}_{jk}; \boldsymbol{\mu}_{jk}, \boldsymbol{\sigma}_{jk}) \prod_{i=1}^{N} q(z_{ijk}|v_k^0), \tag{10.1.1.7}$$

where the variational posterior is truncated up to K, the prior is still infinite (Nalisnick and Smyth, 2017). The set of variational parameters which we optimise over are $\boldsymbol{\varphi} = \{\boldsymbol{\alpha^0}, \boldsymbol{\beta^0}, \boldsymbol{\mu}, \boldsymbol{\sigma}\}$ and $q(\phi)$ is the variational distribution. Each term in Eq. (10.1.1.7) is specified as follows

$$q(v_k^0; \alpha_k^0, \beta_k^0) = \text{Beta}(v_k; \alpha_k^0, \beta_k^0), \tag{10.1.1.8}$$

$$\pi_k^0 = \prod_{i=1}^{k} v_i^0, \tag{10.1.1.9}$$

$$q(\pi_{jk}|v_k^0) = \text{Beta}(\pi_{jk}; \alpha_j \pi_k^0, \alpha_j(1 - \pi_k^0)) \tag{10.1.1.10}$$

$$q(z_{ijk}|v_k^0) = \text{Bern}(z_{ijk}; \pi_{jk}), \tag{10.1.1.11}$$

$$q(\boldsymbol{w}_{jk}; \boldsymbol{\mu}_{jk}, \boldsymbol{\sigma}_{jk}) = \mathcal{N}(\boldsymbol{w}_{jk}; \boldsymbol{\mu}_{jk}, \boldsymbol{\sigma}_{jk}^2 \mathbb{1}), \tag{10.1.1.12}$$

where α_j are hyperparameters (Gupta et al., 2012). Now that we have defined our structured variational approximation in Eq. (10.1.1.7) we can write down the objective

$$\arg\min D_{\text{KL}}\left(q(\boldsymbol{\phi})||p(\boldsymbol{\phi}|\mathcal{D})\right) = \arg\min D_{\text{KL}}\left(q(\boldsymbol{\phi})||p(\boldsymbol{\phi})\right) - \mathbb{E}_{q(\phi)}[\log p(\mathcal{D}|\boldsymbol{\phi})]. \tag{10.1.1.13}$$

In the above formula, $q(\boldsymbol{\phi})$ is the approximate posterior, and $p(\boldsymbol{\phi})$ is the prior. By substituting Eq. (10.1.1.7), we obtain the negative ELBO objective:

$$\begin{aligned}\mathcal{L}(\boldsymbol{\phi}, \mathcal{D}) = \text{KL}(q(\boldsymbol{v^0})||p(\boldsymbol{v^0})) + \sum_{j=1}^{J} \text{KL}(q(\boldsymbol{\pi_j}|\boldsymbol{v}^0)||p(\boldsymbol{\pi_j}|\boldsymbol{v}^0)) + \text{KL}(q(\boldsymbol{w_j})||p(\boldsymbol{w_j})) \\ - \sum_{j=1}^{J}\sum_{i=1}^{N} \mathbb{E}_{q(\phi)}[\log p(y_i|\boldsymbol{x_i}, \boldsymbol{z_{ij}}, \boldsymbol{w_j})] + \sum_{j=1}^{J}\sum_{i=1}^{N} \text{KL}(q(\boldsymbol{z_{ij}}|\boldsymbol{\pi_j})||p(\boldsymbol{z_{ij}}|\boldsymbol{\pi_j})).\end{aligned} \tag{10.1.1.14}$$

Estimating the gradient of the Bernoulli and Beta variational parameters requires a suitable reparameterization. Samples from the Bernoulli distribution in

Equation Eq. (10.1.1.11) arise after taking an `argmax` over the Bernoulli parameters. The `argmax` is discontinuous and a gradient is undefined. The Bernoulli is reparameterized as a Concrete distribution (Maddison et al., 2016; Jang et al., 2016). Additionally, the Beta is reparameterized implicitly (Figurnov et al., 2018) to separate sampling nodes and parameter nodes in the computation graph and allow the use of stochastic gradient methods to learn the variational parameters $\boldsymbol{\varphi}$ of the approximate H-IBP posterior. The mean-field approximation for the Gaussian weights of the BNN is used, $\boldsymbol{w}$ in Eq. (10.1.1.12) (Blundell et al., 2015a; Nguyen et al., 2017). In the next sections, we detail the reparameterizations used to optimise Eq. (10.1.1.14).

10.1.1.1 The variational Gaussian weight distribution reparameterization

The variational posterior over the weights of the BNN is diagonal Gaussian $\boldsymbol{w}_{jk} \sim \mathcal{N}(\boldsymbol{w}_{jk}|\boldsymbol{\mu}_{jk}, \boldsymbol{\sigma}_{jk}^2\mathbb{1})$. By using a reparameterization, one can represent the BNN weights using a deterministic function $\boldsymbol{w}_{jk} = g(\epsilon; \boldsymbol{\varphi})$, where $\epsilon \sim \mathcal{N}(0, \mathbb{1})$ is an auxiliary variable and $g(\epsilon; \boldsymbol{\varphi})$ a deterministic function parameterized by $\boldsymbol{\varphi} = (\boldsymbol{\mu}_{jk}, \boldsymbol{\sigma}_{jk})$, such that:

$$\nabla_{\varphi}\mathbb{E}_{q(\boldsymbol{w}_{jk};\varphi)}\left[f(\boldsymbol{w}_{jk})\right] = \mathbb{E}_{q(\epsilon)}\left[\nabla_{\varphi} f(\boldsymbol{w}_{jk})\right]|_{\boldsymbol{w}_{jk}=g(\epsilon;\varphi)}, \tag{10.1.1.15}$$

where f is an objective function, for instance Eq. (10.1.1.14). The BNN weights can be sampled directly through the reparameterization: $\boldsymbol{w}_{jk} = \boldsymbol{\mu}_{jk} + \boldsymbol{\sigma}_{jk}\epsilon$. By using this simple reparameterization the weight samples are now deterministic functions of the variational parameters $\boldsymbol{\mu}_{jk}$ and $\boldsymbol{\sigma}_{jk}$ and the noise comes from the independent auxiliary variable ϵ (Kingma and Welling, 2014). Taking a gradient of our ELBO objective in Eq. (10.1.1.14) the expectation of the log-likelihood may be rewritten by integrating over ϵ so that the gradient with respect to $\boldsymbol{\mu}_{jk}$ and $\boldsymbol{\sigma}_{jk}$ can move into the expectation according to Eq. (10.1.1.15) (the backward pass). In the forward pass, $\epsilon \sim q(\epsilon)$ is sampled to compute $\boldsymbol{w}_{jk} = \boldsymbol{\mu}_{jk} + \boldsymbol{\sigma}_{jk}\epsilon$.

10.1.1.2 The implicit Beta distribution reparameterization

Implicit reparameterization gradients (Figurnov et al., 2018; Mohamed et al., 2019b) are used to learn the variational parameters for Beta distribution. Nalisnick and Smyth (2017) proposes to use a Kumaraswamy reparameterization. However, in CL a Beta distribution rather than an approximation is desirable for repeated Bayesian updates.

There is no simple inverse of the reparameterization for the Beta distribution like the Gaussian distribution presented earlier. Hence, the idea of implicit reparameterization gradients is to differentiate a standardization function rather than have to perform its inverse. The standardization function is given by the Beta distribution's CDF.

Following Mohamed et al. (2019b), the derivative required for general stochastic VI is:

$$\begin{aligned}\nabla_{\varphi}\mathbb{E}_{q(v^0;\varphi)}\left[f(v^0)\right] &= \mathbb{E}_{q(\epsilon)}\left[\nabla_{\varphi}f(v^0)\right]|_{v^0=g(\epsilon;\varphi)} \\ &= \mathbb{E}_{q(v^0;\varphi)}[\nabla_{v^0}f(v^0)\nabla_{\varphi}v^0],\end{aligned} \tag{10.1.1.16}$$

where f is an objective function, i.e. Eq. (10.1.1.14), $\varphi = \{\alpha_k^0, \beta_k^0\}$ are the Beta parameters. The implicit reparameterisation gradient solves for $\nabla_{\varphi}v^0$, above, using implicit differentiation (Figurnov et al., 2018):

$$\nabla_{\varphi}v^0 = -(\nabla_z g_{\varphi}^{-1}(v^0;\varphi))^{-1}\nabla_{\varphi}g_{\varphi}^{-1}(v^0;\varphi), \tag{10.1.1.17}$$

where $v^0 = g(\epsilon;\varphi)$, $g(.)$ is the inverse CDF of the Beta distribution and $\epsilon \sim$ Unif$[0,1]$ and so $\epsilon = g^{-1}(v^0;\varphi)$ is the standardization path which removes the dependence of the distribution parameters on the sample. The key idea in implicit reparameterization is that this expression for the gradient in Eq. (10.1.1.17) only requires differentiating the standardization function $g_{\varphi}^{-1}(v^0;\varphi)$ and not inverting it. Given $v^0 = g(\epsilon;\varphi) = F^{-1}(v^0;\varphi)$ then using Eq. (10.1.1.17) the implicit gradient is:

$$\begin{aligned}\nabla_{\varphi}v^0 &= \nabla_{\varphi}F^{-1}(v^0;\varphi) \\ &= \frac{\nabla_{\varphi}F(v^0;\varphi)}{p_{\varphi}(v^0;\varphi)},\end{aligned} \tag{10.1.1.18}$$

where $p_\varphi(v^0;\varphi)$ is the Beta PDF. The CDF of the Beta distribution is given by

$$F(v^0;\varphi) = \frac{B(v^0;\varphi)}{B(\varphi)} \tag{10.1.1.19}$$

where $B(v^0;\varphi)$ and $B(\varphi)$ are the incomplete Beta function and Beta function, respectively. The derivatives of $\nabla_\varphi F(v^0;\varphi)$ do not admit simple analytic expressions. Thus, numerical approximations have to be made, for instance, by using the Taylor series for $B(v^0;\varphi)$ (Jankowiak and Obermeyer, 2018; Figurnov et al., 2018).

In the forward pass, $v^0 \sim q_\varphi(v^0)$ is sampled from a Beta distribution or alternatively from two Gamma distributions. That is, if $v_1^0 \sim \text{Gamma}(\alpha_k^0, 1)$ and $v_2^0 \sim \text{Gamma}(\beta_k^0, 1)$, then $\frac{v_1^0}{v_1^0+v_2^0} \sim \text{Beta}(\alpha_k^0, \beta_k^0)$.

Then, in the backward pass, we estimate the partial derivatives w.r.t. $\varphi = \{\alpha_k^0, \beta_k^0\}$ as can be taken by using Eq. (10.1.1.18) and previous equations. Implicit reparameterization of the Beta distribution is readily implemented in `TensorFlow Distributions` (Dillon et al., 2017).

10.1.1.3 The variational Bernoulli distribution reparameterization

The Bernoulli distribution can be parameterized using a continuous relaxation to the discrete distribution and so Eq. (10.1.1.15) can be used.

Consider a discrete distribution $(\alpha_1, \cdots \alpha_K)$ where $\alpha_j \in \{0, \infty\}$ and $D \sim \text{Discrete}(\alpha) \in \{0, 1\}$, then $P(D_j = 1) = \frac{\alpha_j}{\sum_k \alpha_k}$. Sampling from this distribution requires performing an `argmax` operation, the crux of the problem is that the `argmax` operation doesn't have a well-defined derivative.

To address the derivative issue above, the Concrete distribution (Maddison et al., 2017) or Gumbel-Softmax distribution (Jang et al., 2017) is used as an approximation to the Bernoulli distribution. The idea is that instead of returning a state on the vertex of the probability simplex like `argmax` does, these relaxations return states inside the probability simplex (see Figure 2 in Maddison et al. (2016)). The Concrete formulation and notation from Maddison et al. (2016) is used. We sample from the probability simplex using

$$X_j = \frac{\exp((\log \alpha_j + G_j)/\lambda)}{\sum_{i=1}^n \exp((\log \alpha_i + G_i)/\lambda)}, \tag{10.1.1.20}$$

with temperature hyperparameter $\lambda \in (0, \infty)$, parameters $\alpha_j \in (0, \infty)$ and i.i.d. Gumbel noise $G_j \sim \text{Gumbel}(0, 1)$. This equation resembles a `softmax` with a Gumbel perturbation. As $\lambda \to 0$ the `softmax` computation approaches the `argmax` computation. This can be used as a relaxation of the variational Bernoulli distribution and can be used to reparameterize Bernoulli random variables to allow gradient-based learning of the variational Beta parameters downstream in our model.

When performing variational inference using the Concrete reparameterization for the posterior, a Concrete reparameterization of the Bernoulli prior is required to properly lower bound the ELBO in Eq. (10.1.1.14). If $q(z_{ijk}; \pi_{jk}|v_k^0)$ is the variational Bernoulli posterior over sparse binary masks z_{ijk} for weights $\boldsymbol{w}_{jk}$ and $p(z_{ijk}; \pi_{jk}|v_k^0)$ is the Bernoulli prior. To guarantee a lower bound on the ELBO, both Bernoulli distributions require replacing with Concrete densities, i.e.,

$$\text{KL}\left[q(z_{ijk}; \pi_{jk}|v_k^0)||p(z_{ijk}; \pi_{jk}|v_k^0)\right] \geq \text{KL}\left[q(z_{ijk}; \pi_{jk}, \lambda_1|v_k^0)||p(z_{ijk}; \pi_{jk}, \lambda_2|v_k^0)\right], \tag{10.1.1.21}$$

where $q(z_{ijk}; \pi_{jk}, \lambda_1|v_k^0)$ is a Concrete density for the variational posterior with parameter π_{jk}, temperature parameter λ_1 given global parameters v_k^0. The Concrete prior is $p(z_{ijk}; \pi_{jk}, \lambda_2|v_k^0)$. Eq. (10.1.1.21) is evaluated numerically by sampling from the variational posterior (we take a single Monte Carlo sample (Kingma and Welling, 2013)). At test time one can sample from a Bernoulli using the learned variational parameters of the Concrete distribution (Maddison et al., 2016).

In practice, the log transformation is used to alleviate underflow problems when working with Concrete probabilities. One can instead work with $\exp(Y_{ijk}) \sim \text{Concrete}(\pi_{jk}, \lambda_1|v_k^0)$, as the KL divergence is invariant under this invertible transformation and Eq. (10.1.1.21) is valid for optimizing our Concrete parameters (Maddison et al., 2017). For binary Concrete variables, one can sample from $y_{ijk} = (\log \pi_{jk} + \log \epsilon - \log(1-\epsilon))/\lambda_1$ where $\epsilon \sim \text{Unif}[0, 1]$ and the log-density (before applying the sigmoid activation) is $\log q(y_{ijk}; \pi_{jk}, \lambda_1|v_{jk}) = \log \lambda_1 - \lambda_1 y_{ijk} + \log \pi_{jk} - 2\log(1 + \exp(-\lambda_1 y_{ijk} + \log \pi_{jk}))$ (Maddison et al., 2017). The reparameterization $\sigma(y_{ijk}) = g(\epsilon; \pi_{jk})$, where σ is the sigmoid function, enables us to differentiate through the Concrete and use a similar formula to Eq. (10.1.1.15).

10.1.2 Comparison of HIBNN and Sparse Variational Dropout

Using the IBP and H-IBP priors for model selection induces sparsity as a side effect. Other priors such as Sparse Variational Dropout (SVD) (Molchanov et al., 2017) or a horse-shoe prior on weights (Louizos et al., 2017) are specifically employed for sparsity. By comparing the effect of weight pruning between a HIBNN and a BNN employing SVD, we can see that the HIBNN is not as sparse as SVD, although SVD is specifically designed to be sparse, unlike our method which employs a non-parametric prior for CL, sparsity is a side effect. The accuracies obtained from SVD before pruning are 98.1 ± 0.1 compared to 95 ± 0.0. The performance gap is due to the difficulty in inference and with the variational approximations and reparameterizations, as noted when compared to a BNN with no special prior in Section 4.5.1. In terms of pruning, SVD is slightly more robust; performance starts to degrade after 99% of weights are pruned in comparison to the HIBNN's 98%, see Fig. 10.1.

The SVD BNN uses a two-layer BNN with 200 neurons and the HIBNN with a variational truncation of $K = 200$ for a fair comparison. The H-IBP prior parameters and the initialization of the variational posterior are $\alpha_k^0 = 4.2$ and $\beta_k^0 = 1.0$ for all k, the hyperparameter $\alpha_j = 4$ for all layers j. The Concrete temperatures used are $\lambda_1 = 0.7$ for the variational posterior and $\lambda_2 = 0.7$ for the Concrete prior. The prior on the Gaussian weights were set to $\mathcal{N}(0.0, 0.7)$, these parameters were found to work well on split MNIST and were assumed to also work well for multiclass MNIST.

Both networks are optimized using Adam (Kingma and Lei Ba, 2015) with a decaying learning rate schedule starting at 10^{-3} at a rate of 0.87 every 1000 steps, for 200 epochs and using a batch size of 128. Weight means are initialized with their ML parameters found after training for 100 epochs and $\log \sigma^2 = -6$. Local reparameterization (Kingma et al., 2015) is employed. SVD is trained for 100 epochs while our method is trained for 200 epochs, as it requires more epochs to converge.

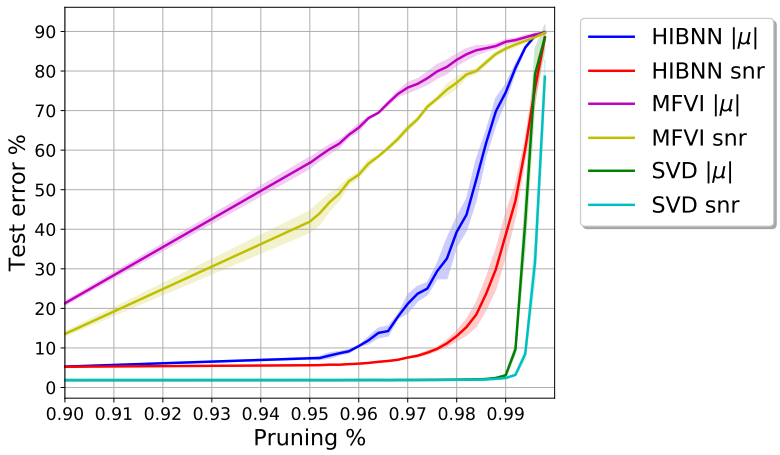


Figure 10.1: Test errors for different pruning cut-offs for the HIBNN and with a BNN trained with Sparse Variational Dropout (SVD) on MNIST. SVD is more sparse than the HIBNN.

10.1.3 Weight Pruning Further Results

10.1.3.1 MNIST

To investigate the behaviour of the novel BNN introduced in this paper we perform simple and intuitive weight pruning experiments. The Gaussian weights of the BNN are zeroed out in two particular orders. The first is by the magnitude of the variational mean: $|\mu|$ the second is by the variational signal-to-noise ratio (SNR): $|\mu|/\sigma$. For the IBNN and HIBNN, weights are pruned according to these two metrics at the same time as having the sparse matrix Z acting on each layer. As Z is sparse by design we expect our models to be sparse in comparison to a BNN where the weights are modeled by mean-field variational inference (MFVI) (Blundell et al., 2015b). Indeed the IBNN and HIBNN are sparse in comparison to MFVI and become sparser with increased depth. In Fig. 10.2 the weights are pruned and the accuracy on the test set is measured for networks of different sizes. MFVI is less sparse as the depth increases. Accuracy does not vary for all models of depth 1 to 4. The MFVI models have a hidden state size of 200 and the IBNN and HIBNN use a variational truncation of $K = 200$ for a fair comparison.

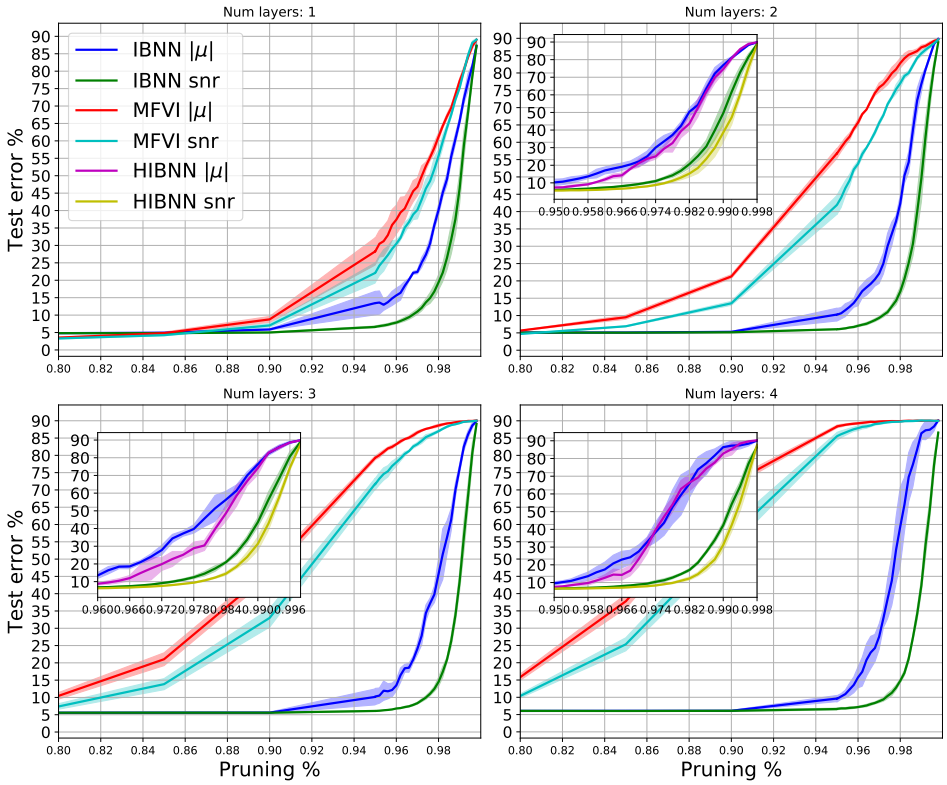


Figure 10.2: Weight pruning experiments for the IBNN and HIBNN on MNIST for models of different depths. As the depth increases the IBNN and HIBNN become more sparse.

10.1.3.2 Fashion-MNIST

We repeat this analysis for the fashion MNIST (fMNIST) (Xiao et al., 2017). Similarly to MNIST, the IBNN and HIBNN networks become sparser with increasing depth since the drop-off in accuracies arises for higher pruning percentages. There is little difference in the performance or sparsity between the IBNN and HIBNN.

10.1.4 Dynamically Expanding Networks and Time-Stamping

DEN (Yoon et al., 2017) "time-stamps" parts of its network allowing specific parts of the network to be used by specific tasks thereby achieving good performance and good uncertainties over tasks which have been seen for `CL2` and `CL3`. This is at the cost of not allowing backward transfer from newly added task weights. Removing the time-stamping feature (and making DEN more in line with the IBNN) we see that it achieves similar performance for the more difficult MNIST variants,

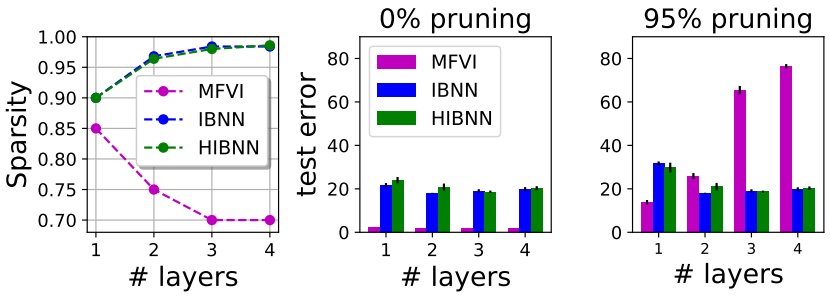


Figure 10.3: fMNIST network depth ablation. **Left**, as the depth of the IBNN and HIBNN increases the networks tend to remain very sparse while the MFVI BNN becomes less sparse. **Middle & Right**, the HIBNN and IBNN remain robust to pruning with increasing depth.

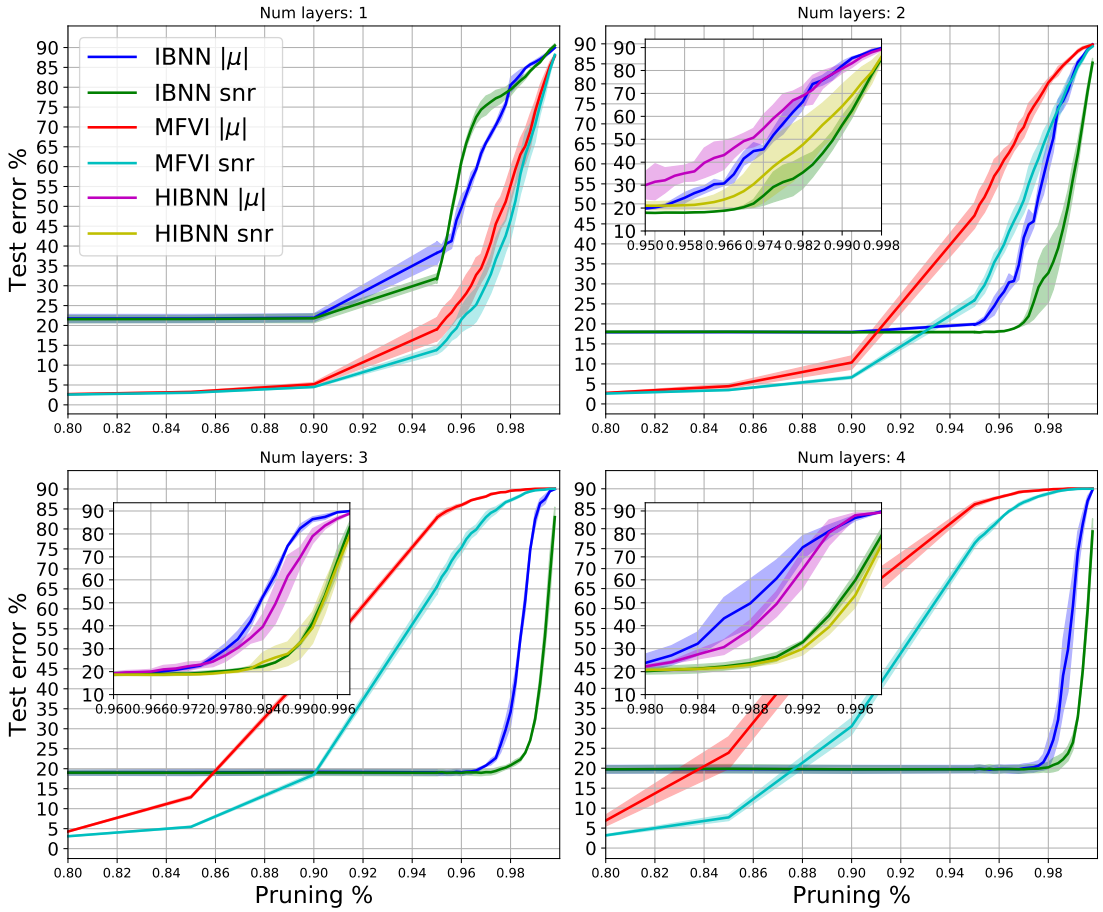


Figure 10.4: Weight pruning for the IBNN and HIBNN in comparison to MFVI on fashion-MNIST for models of different depths. As the depth increases the IBNN and HIBNN become more sparse.

Table 10.1: Average test accuracy on MNIST CL experiments over 5 runs. After removing the time-stamping feature from DEN the IBNN achieves comparable performance. The best model between the IBNN and DEN with no time-stamping is highlighted.

	DEN	DEN - no ts	IBNN
P `CL1`	91.4±0.5	91.6±0.5	**95.6±0.2**
P `CL3`	63.9±19.2	75.1±21.0	**93.8±0.3**
S `CL1`	99.1±0.1	**98.8±0.1**	95.3±2.0
S `CL2`	98.9±0.1	**98.9±0.1**	91.0±2.2
S `CL3`	99.1±0.1	**93.6±10.4**	**85.5±3.2**
S+ϵ `CL1`	97.2±0.2	**95.1±1.9**	**95.1±1.1**
S+ϵ `CL2`	84.8±16.7	**91.1±7.0**	**89.7±3.8**
S+ϵ `CL3`	90.9±12.8	54.1±24.1	**78.7±11.7**
S+img `CL1`	93.8±0.8	**91.7±2.1**	**91.6±1.2**
S+img `CL2`	79.1±13.1	**98.9±0.1**	80.5±7.8
S+img `CL3`	91.6±5.1	46.7±14.3	**66.2±13.4**

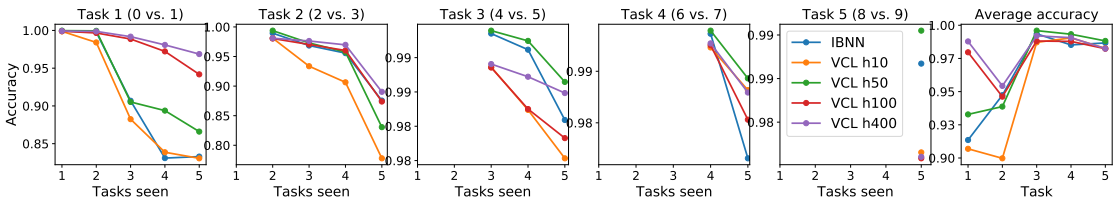


Figure 10.5: Split MNIST `CL1`, task accuracies versus the number of tasks the model has performed a Bayesian update. Our model is compared to VCL benchmarks with different numbers of hidden states denoted hx, $x \in \{10, 50, 100, 400\}$. For Split MNIST the tasks are simple and no overfitting is observed in VCL.

see Table 10.1. This motivates segregating parts of a BNN to perform a specific task so as to mimic time-stamping for future research.

10.1.5 Overfitting and Underfitting in VCL

Preliminaries. All task accuracies are an average of 5 runs and by action neurons we mean that $z_{ijk} > 0.1$ for a datapoint $\boldsymbol{x}_i$ in layer j and for neuron k.

For CL on split MNIST, the IBNN is able to outperform the 10-neuron VCL baseline as it underfits. On the other hand, the larger VCL networks slightly outperform the IBNN, see Fig. 10.5. The IBP prior enables the BNN to expand from a median of 11 neurons for the first task to 14 neurons for later tasks, see Fig. 10.7.

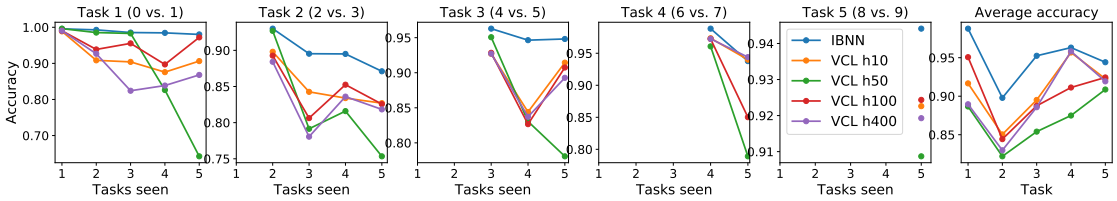


Figure 10.6: Split MNIST with random background noise for `CL1`, task accuracies versus the number of tasks the model has performed a Bayesian update. Our model is compared to VCL benchmarks with different numbers of hidden states denoted hx, $x \in \{10, 50, 100, 400\}$. Our model is able to counter overfitting issues and outperform VCL.

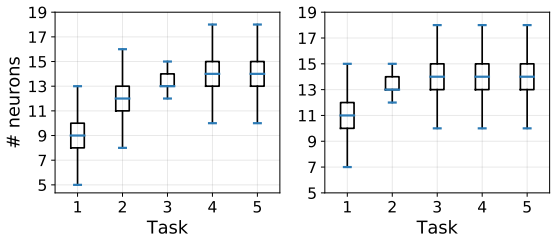


Figure 10.7: Box plots of the number of active neurons selected by the IBP variational posterior for each task, for **Left** Split MNIST and **Right** Split MNIST background noise variant for `CL1`. For both datasets our model expands its capacity over the course of CL.

Regarding CL on MNIST with random background noise, it is clear that the IBNN is able to outperform VCL for all widths considered. The baseline models overfit on the second task and propagate a poor approximate posterior which affects subsequent task performance, see Fig. 10.6. The IBNN expands over the course of the 5 tasks from a median of 11 neurons to 14 neurons in Fig. 10.7.

VCL networks with a small width of 10 neurons on Permuted MNIST display underfitting due to a lack of capacity which translates into forgetting in CL. On the other hand, a VCL model that is overparameterized with a width of 100 displays overfitting and will subsequently propagate a poor posterior and results in forgetting for all future tasks Fig. 10.8. The same phenomena of forgetting due to underfitting due to restricted model size and overfitting due to overparameterization is observed for the Split MNIST with random background images dataset in Fig. 10.9.

10.1.6 Experimental Details

We summarise all dataset sizes in Table 10.2.

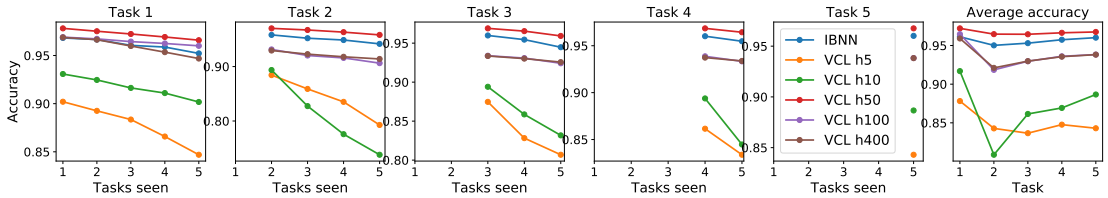


Figure 10.8: Permuted MNIST task accuracies versus the number of tasks the model has seen and performed a Bayesian update. Our model is compared to VCL with different numbers of hidden state sizes. Our model has a very good performance versus most models for Permuted MNIST.

Figure 10.9: Split MNIST with background images, task accuracies versus the number of tasks the model has seen and performed a Bayesian update. Our model is compared to VCL with different hidden state sizes. Our model adapts its complexity and overcomes forgetting better than VCL.

Permuted MNIST. No preprocessing of the data is performed like Nguyen et al. (2017). For the permuted MNIST experiments, the BNNs used for the baselines and the IBNN consist of a single layer with ReLU activations. For the IBNN, the variational truncation parameter is set to $K = 100$. For the first task, the parameters of the Beta prior and the variational Beta distribution are initialized to $\alpha_k = 5$ and $\beta_k = 1$ for all k. The temperature parameters of the Concrete distributions for the variational posterior and priors are set to $\lambda_1 = 1$ and $\lambda_2 = 1$ respectively. For each batch, 10 samples are averaged from the IBP priors as this makes training stable. The Gaussian weights of the BNNs have their means and variances initialized with the maximum likelihood estimators found after training for 100 epochs with Adam and the variances initialized to $\log \sigma^2 = -6$ for the first task. For CL Adam is also used for 200 epochs and an initial learning rate of 0.001 which decays exponentially with a rate of 0.87 every 1000 iterations for each task. The CL experiments follow the implementation of VCL (Nguyen et al., 2017).

Split MNIST and variants. The BNNs used for the baselines and the IBNN consist of a single layer with ReLU activations. The variational truncation parameter is set to $K = 100$. For `CL1` and for the first task, the parameters of the Beta prior and the variational Beta distribution are initialized to $\alpha_k = 5$ and $\beta_k = 1$ for all k. The temperature parameters of the Concrete distributions for the variational posterior and priors are set to $\lambda_1 = 0.7$ and $\lambda_2 = 0.7$, respectively. The prior for the Gaussian weights is $\mathcal{N}(0, 0.7)$. The Gaussian weights of the BNNs have their means initialized with the maximum likelihood estimators, found after training a NN for 100 epochs with Adam. The variances are initialized to $\log \sigma^2 = -6$. The Adam optimizer is used for 600 epochs, the default in the original VCL paper (Nguyen et al., 2017) for Split MNIST. To ensure convergence of the IBNN, the first task needs to be trained for 20% more epochs. An initial learning rate of 0.001 which decays exponentially with a rate of 0.87 every 1000 iterations is employed.

10.1.6.1 Hyperparameter optimisation details

Random search is performed on some of the experiments presented in the main paper by sampling over a discrete set of values or over a range for some hyperparameters listed below. We denote curly brackets $\{\ldots\}$ as a discrete set and square brackets $[\ldots]$ as a range.

HIBNN `CL1` and `CL2` for CL experiments with increasing task difficulty Table 4.2.

- Concrete posterior temperature: $\{1/2, 2/3, 3/4, 1, 5/4, 3/2, 7/4, 2, 9/4, 5/2, 11/4, 3\}$.
- Concrete prior temperature: $\{1/2, 2/3, 3/4, 1, 5/4, 3/2, 7/4, 2, 9/4, 5/2, 11/4, 3\}$.
- Child IBP base α: $\{1, 2, 3, 4, 5\}$.
- Child IBP multiple of α after each new dataset seen in during CL: $\{1, 2, 3\}$
- Global IBP prior, α^0: $[5, 25]$

IBNN `CL1` and `CL2` for CL experiments with increasing task difficulty Table 4.2 and IBNN `CL3` for Split MNIST with background noise and `CL2` and `CL3` for Split MNIST with background images Table 4.1.

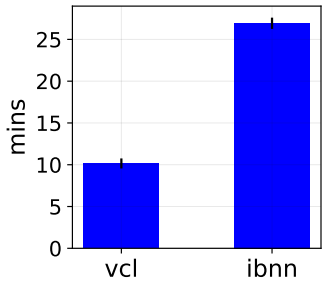


Figure 10.10: Time to run one permuted MNIST task for our model and VCL. Our model takes longer due to the inference of the IBP variational posterior.

- Concrete prior temperature: $\{1/2, 2/3, 3/4, 1, 5/4, 3/2, 7/4, 2, 9/4, 5/2, 11/4, 3\}$
- Concrete posterior temperature: $\{1/2, 2/3, 3/4, 1, 5/4, 3/2, 7/4, 2, 9/4, 5/2, 11/4, 3\}$
- IBP prior, α: $[5, 25]$

10.1.6.2 Baselines Implementations

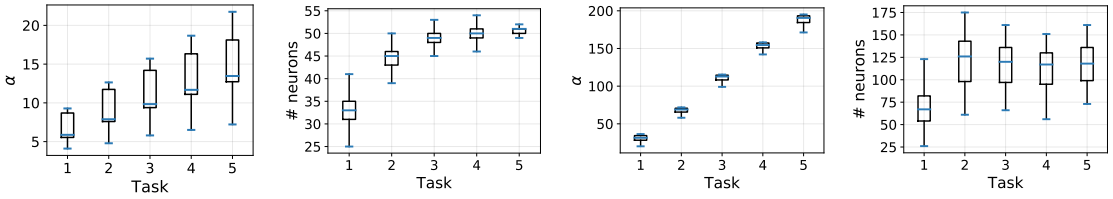
The implementations for EWC, SI, and GEM are from Hsu et al. (2018b) and use default hyperparameters. For DEN we use the implementation provided by the authors with default hyperparameters (Yoon et al., 2017). For VCL we likewise use the implementation provided by the authors with default hyperparameters (Nguyen et al., 2017).

10.1.6.3 Training time

Comparing the training time of one Permuted MNIST task for 200 epochs can be seen in Fig. 10.10. The IBNN takes longer to train. Stochastic variational inference of the IBP posterior involves taking multiple samples from it in a forward pass. Then in the backward pass, gradients of samples from the Concrete distribution and implicit gradients from samples from the Beta distribution need to be calculated; this explains the longer training times in comparison to VCL. Improving the inference scheme is a good direction for further work.

Table 10.2: Dataset sizes used for experiments. Split MNIST's variants is obtained from `https://sites.google.com/a/lisa.iro.umontreal.ca/public_static_twiki/variations-on-the-mnist-digits`.

Dataset	Train set size	Test set size
Permuted MNIST	$50,000$	$10,000$
Split MNIST	$60,000$	$10,000$
Split MNIST + noise	$52,000$	$10,000$
Split MNIST + images	$52,000$	$10,000$
CIFAR 10	$50,000$	$10,000$



(a) IBNN α. **(b)** IBNN Z. **(c)** BSCL α. **(d)** BSCL Z.

Figure 10.11: Comparing our model to BSCL (Kumar et al., 2019) for a simple 5 task CL experiment on Permuted MNIST. In our model, the variational parameters α which control the number of neurons, increase for each task Fig. 10.11a, as does the resulting number of neurons Fig. 10.11b. While for BSCL as α increases Fig. 10.11c, this doesn't translate into additional weights becoming activated Fig. 10.11d. Since the additional parameter p_k for all k is influencing the number of neurons that are active in addition to the IBP prior term item Item 1. We use the outputs of the Concrete distribution as Z and define a neuron as active if $z_k > 0.1$ for neuron k.

10.1.7 Detailed Comparison with Related Works

The related papers (Panousis et al., 2019) and (Kumar et al., 2019) apply the IBP prior to a BNN like in our model, however, they apply it in a very different fashion. In these works, the IBP prior is applied to each layer such that $Z \in \mathbb{Z}_2^{d \times k}$, where d is the input dimension or input hidden layer size and k is the output dimension. Crucially, Z is not sampled for each data point. Our model applies $Z \in \mathbb{Z}_2^{n \times k}$, such that each point selects neurons according to the IBP, this remains closer to the original formulation of the IBP prior for matrix factorization Section 4.2.3 and to related works applying the IBP prior to VAEs.

Despite this important difference with Bayesian Structure Adaptation for CL (BSCL) (Kumar et al., 2019), the results for BSCL are strong and suggest some interesting design choices and ideas that can be leveraged to use the IBP prior for CL

with BNNs. It is promising to see similar ideas being put forward in this area. There are some design choices that are quite different from ours in three main regards.

1. An additional variational parameter for each Gaussian neural network weight is introduced. The IBP prior model is $z_k \sim \text{Ber}(\pi_k)$ where $\pi_k = \prod_{i=1}^{k} v_i$ and where $v_i \sim \text{Beta}(\alpha, \beta)$ for each layer, for a neuron k. However in the BSCL implementation, the Bernoulli probability is $\tilde{\pi}_k = p_k + \prod_{i=1}^{k} v_i$, where p_k is an additional learnable parameter. Thus the Bernoulli parameters are being directly learned, similarly to Dikov et al. (2019). The parameter α directly controls the number of active items in the IBP prior (Griffiths and Ghahramani, 2011) and so this additional parameter p_k can potentially dominate α in controlling the expansion of Z see Fig. 10.11.

2. The implementation of BSCL does not strictly follow sequential Bayes, in particular the Beta distribution (which is part of the IBP and controls expansion). In sequential Bayes, like VCL, the variational posterior of the Beta distribution (the implementation uses a relaxation, the Kumaraswamy distribution), is used as the prior for the forthcoming task. The previous task's variational Beta posterior parameters are $\alpha \in \mathbb{R}_+^K$ and $\beta \in \mathbb{R}_+^K$ respectively, the new task's Beta prior parameters are then set to $\max(\alpha) \in \mathbb{R}_+$ and 1 respectively where K is the variational truncation[1]. This explains the large increase in the values of α seen in Fig. 10.11 and so allows a lot of expansion in the model.

3. BSCL requires more memory in storing Z's from each task. The Z's are used as a mask to then finetune the network which can result in better performance. For each task, Z's for all weights are stored and recalled thus alleviating forgetting, rather than relying on sequential variational Bayes like VCL and our method. We apply sequential Bayes on the IBP and so generate Z's for each new task without having to store any parameters from previous tasks.

Table 10.3: Average test accuracy on MNIST variants for *task incremental learning* (`CL1`) experiments over 5 runs. Despite the many design choices BSCL (Kumar et al., 2019), performs similarly to our method which is simpler and more correct.

	BSCL	IBNN
S+ϵ `CL1`	**95.2**±1.5	**95.1**±1.1
S+img `CL1`	**92.9**±1.0	91.6±1.2

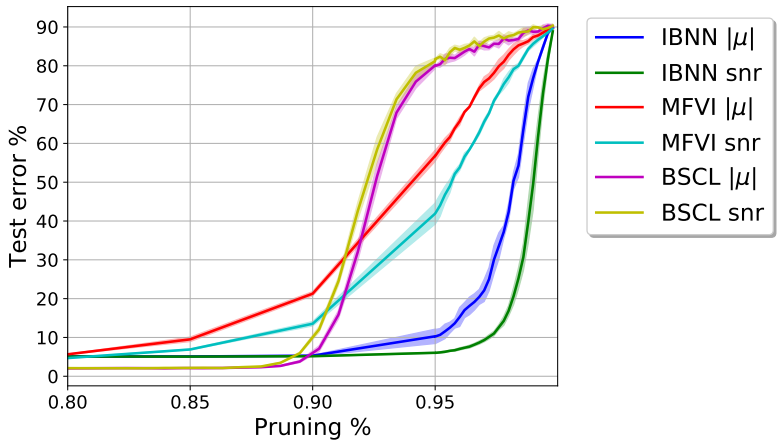


Figure 10.12: Weight pruning on MNIST for BSCL compared to our model and a BNN with independent Gaussian weights (MFVI). BSCL is less robust to pruning than our model and MFVI, also it is less robust when pruning with signal-to-noise ratio: $|\boldsymbol{\mu}|/\boldsymbol{\sigma}$, thus the variational variances are not being learned properly.

Despite these design choices, BSCL performs similarly to our model for the more difficult MNIST variants, see Table 10.3. We use the implementation provided by the authors with default parameters used for Split MNIST CL experiments. We also perform weight pruning on MNIST using a 2 layer BSCL network with variational truncation $K = 200$ using the default parameters for Split MNIST CL experiments. We see from Fig. 10.12 that despite having a strong 0% pruning accuracy of 98 ± 0.1 in comparison to the IBNN's 95 ± 0.0. As we prune weights by the signal-to-noise ratio (snr), $|\boldsymbol{\mu}|/\boldsymbol{\sigma}$, this is less robust than pruning by $|\boldsymbol{\mu}|$ which shows that BSCL has not learned proper variational variances.

[1] `https://github.com/scakc/NPBCL/blob/master/ibpbnn.py#L939`

10.2 On Sequential Bayesian Inference for Continual Learning

10.2.1 The Toy Gaussians Dataset

See Fig. 10.13 for a visualization of the toy Gaussians dataset which we use as a simple CL problem. This is used for evaluating our method for propagating the true posterior by using HMC for posterior inference and then using a density estimator on HMC samples as a prior for a new task. We construct 5, 2-way classification problems for CL. Each 2-way task involves classifying adjacent circles and squares Fig. 10.13. With a 2 layer network with 10 neurons, we get a test accuracy of 1.0 for the multi-task learning of all 5 tasks together. Hence according to Eq. (5.2.2.3) a BNN with the same size should be able to learn all 5 binary classification tasks continually by sequentially building up the posterior.

10.2.2 HMC implementation details

We set the prior for $\mathcal{T}_1$, to $p_1(\theta) = \mathcal{N}(0, \tau^{-1}\mathbb{I})$ with $\tau = 10$. We burn in the HMC chain for 1000 steps and sample for 10000 more steps and run 20 different chains to obtain samples from our posterior, which we then pass to our density estimator. We use a step size of 0.001 and trajectory length of $L = 20$, see Section 10.2.3 for further implementation details of the density estimation procedure. For the GMM we optimize for the number of components by using a holdout set of HMC samples.

10.2.3 Density Estimation Diagnostics

We provide plots to show that the HMC chains indeed sample from the posterior have converged in Fig. 10.15 and Fig. 10.17. We run 20 HMC sampling chains and randomly select one chain to plot for each seed (of 10). We run HMC over 10 seeds and aggregate the results Fig. 5.3 and Fig. 10.13. The posteriors $p(\theta|\mathcal{D}_1), \ldots$ are approximated with a GMM and used as a prior for the second task and so forth.

We provide empirical evidence to show that the density estimators have fit to HMC samples of the posterior in Fig. 10.14 and Fig. 10.16. Where we show the number of components of the GMM density estimator which we use as a prior

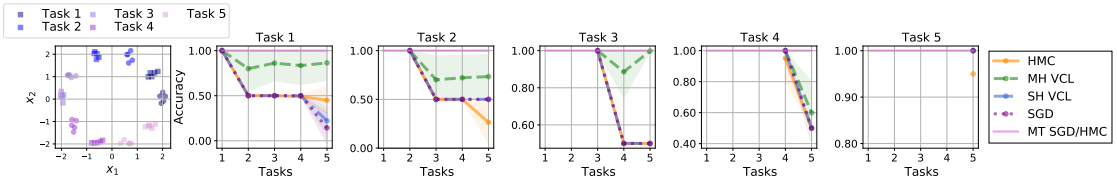


Figure 10.13: Continual learning binary classification accuracies from the toy Gaussian dataset similar to (Henning et al., 2021) using 10 random seeds. The pink solid line is a multi-task (MT) baseline test accuracy using SGD/HMC.

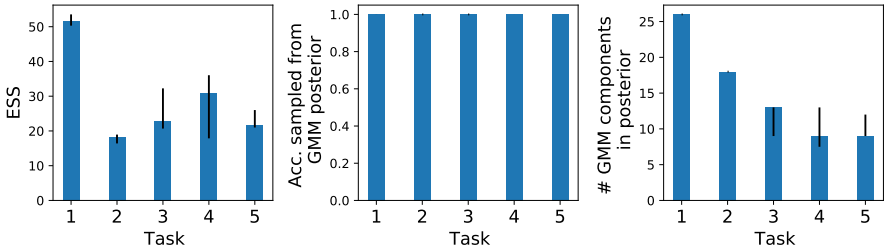


Figure 10.14: Diagnostics from using a GMM prior, to fit samples of the posterior generated from HMC, all results are for 10 random seeds. **Left,** effective sample sizes (ESS) of the resulting HMC chains of the posterior, all are greater than those reported in other works using HMC for BNNs (Cobb and Jalaian, 2021). **Middle,** the accuracy of the BNN when using samples from the GMM density estimator instead of the samples from HMC. **Right,** The optimal number of components of each GMM posterior fitted with a holdout set of HMC samples by maximizing the likelihood.

for a new task are all multi-modal posteriors. We show the BNN accuracy when sampling BNN weights from our GMM all recover the accuracy of the converged HMC samples. The effective sample size (ESS) of the 20 chains is a measure of how correlated the samples are (higher is better). The reported ESS values for our experiments are in line with previous work which uses HMC for BNN inference (Cobb and Jalaian, 2021).

10.2.4 Prototypical Bayesian Continual Learning

ProtoCL models the generative process of CL where new tasks are comprised of new classes $j \in \{1, \ldots, J\}$ of a total of J and can be modeled by using a categorical distribution with a Dirichlet prior:

$$y_{i,t} \sim \text{Cat}(p_{1:J}), \quad p_{1:J} \sim \text{Dir}(\alpha_t). \tag{10.2.4.1}$$

We learn a joint embedding space for our data with a NN, $\boldsymbol{z} = f(\boldsymbol{x}; \boldsymbol{w})$ with parameters $\boldsymbol{w}$. The embedding space for each class is Gaussian whose mean has

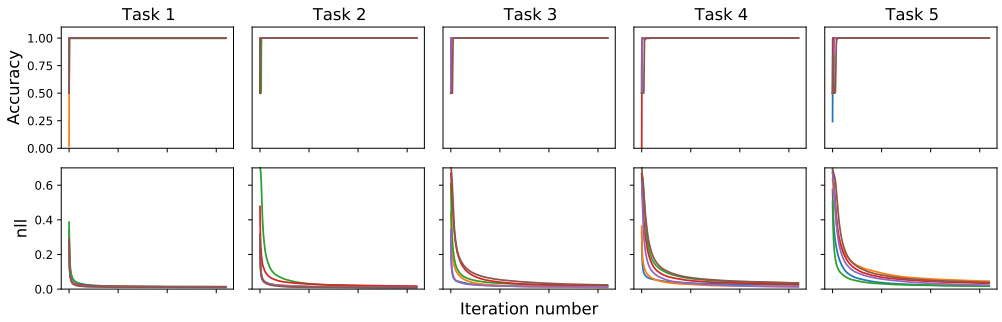


Figure 10.15: Convergence plots from a one randomly sampled HMC chain (of 20) for each task over 10 different runs (seeds) for 5 tasks from the toy Gaussians dataset similar to Henning et al. (2021) (visualized in Fig. 10.13). We use a GMM density estimator as the prior conditioned on previous task data.

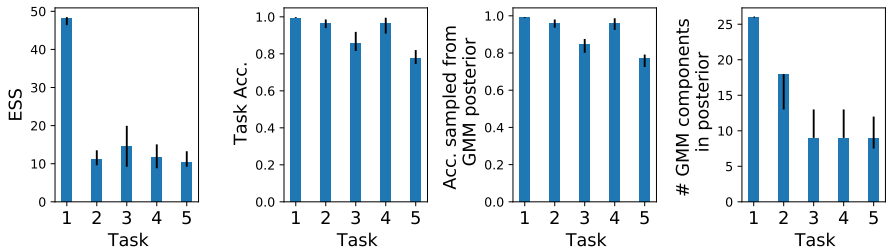


Figure 10.16: Diagnostics from using a GMM to fit samples of the posterior HMC samples, all results are for 10 random seeds on the toy dataset from Pan et al. (2020) (and visualized in Fig. 5.3). **Left,** effective sample sizes (ESS) of the resulting HMC chains of the posterior, all are greater than those reported in other works using HMC for BNNs (Cobb and Jalaian, 2021). **Middle left.** the current task accuracy from HMC sampling. **Middle right,** the accuracy of the BNN when using samples from the GMM density estimator instead of the converged HMC samples. **Right,** The optimal number of components of each GMM posterior fitted with a holdout set of HMC samples by maximizing the likelihood.

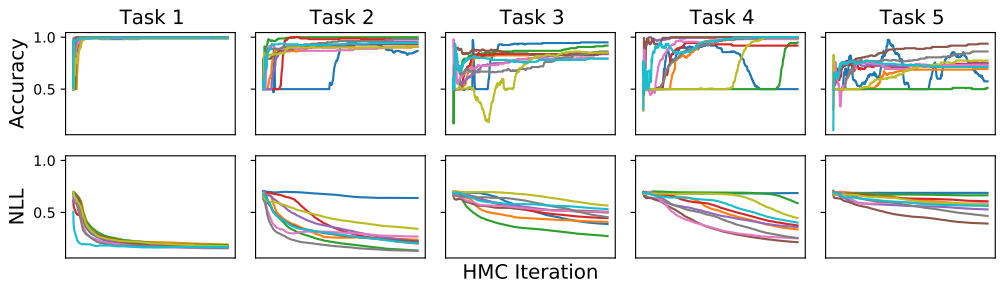


Figure 10.17: Convergence plots from a randomly sampled HMC chain (of 20) for each task over 10 different seeds for 5 tasks from the toy dataset from (Pan et al., 2020) (see Fig. 5.3 for a visualization of the data). We use a GMM density estimator as a prior.

a prior which is also Gaussian:

$$\boldsymbol{z}_{it}|y_{it} \sim \mathcal{N}(\bar{\mathbf{z}}_{yt}, \Sigma_\epsilon), \quad \bar{\mathbf{z}}_{yt} \sim \mathcal{N}(\boldsymbol{\mu}_{yt}, \Lambda_{yt}^{-1}). \tag{10.2.4.2}$$

By ensuring that we have an embedding per class and using a memory of past data, we ensure that the embedding does not drift. The posterior parameters are $\eta_t = \{\alpha_t, \boldsymbol{\mu}_{1:J,t}, \Lambda_{1:J,t}^{-1}\}$.

10.2.4.1 Inference

As the Dirichlet prior is conjugate with the Categorical distribution and so is the Gaussian distribution with a Gaussian prior over the mean of the embedding, then we can calculate posteriors in closed form and update our parameters as we see new data online without using gradient-based updates. We perform gradient-based learning of the NN embedding function $f(\cdot\,; \boldsymbol{w})$ with parameters $\boldsymbol{w}$. We optimize the model by maximizing the log-predictive posterior of the data and use the softmax over class probabilities to perform predictions. The posterior over class probabilities $\{p_j\}_{j=1}^J$ and class embeddings $\{\bar{\boldsymbol{z}}_{y_j}\}_{j=1}^J$ is denoted as $p(\theta)$ for shorthand and has parameters are $\eta_t = \{\alpha_t, \boldsymbol{\mu}_{1:J,t}, \Lambda_{1:J,t}^{-1}\}$ are updated in closed form at each iteration of gradient descent.

10.2.4.2 Sequential updates

We can obtain our posterior:

$$p(\boldsymbol{\theta}_t|\mathcal{D}_t) \propto p(\mathcal{D}_t|\boldsymbol{\theta}_t)p(\boldsymbol{\theta}_t) \tag{10.2.4.3}$$

$$= \prod_{i=1}^{N_t} p(\boldsymbol{z}_t^i|y_t^i; \bar{\boldsymbol{z}}_{y_t}, \Sigma_{\epsilon,y_t}) p(y_t^i|p_{1:J}) p(p_{i:J}; \alpha_t) p(\bar{\boldsymbol{z}}_{y_t}; \boldsymbol{\mu}_{y_t,t}, \Lambda_{y_t,t}^{-1}) \tag{10.2.4.4}$$

$$= \mathcal{N}(\mu_{t+1}, \Sigma_{t+1}) \mathrm{Dir}(\alpha_{t+1}), \tag{10.2.4.5}$$

where N_t is the number of data points seen during update t. Concentrating on the Categorical-Dirichlet conjugacy:

$$\text{Dir}(\alpha_{t+1}) \propto p(p_{1:J}; \alpha_t) \prod_{i=1}^{N_t} p(y_t^i; p_{i:J}) \tag{10.2.4.6}$$

$$\propto \prod_{j=1}^{J} p_j^{\alpha_j - 1} \prod_{i=1}^{N_t} \prod_{j=1}^{J} p_j^{\mathbb{I}(y_t^i = j)} \tag{10.2.4.7}$$

$$= \prod_{j=1}^{J} p_j^{\alpha_j - 1 + \sum_{i=1}^{N_t} \mathbb{I}(y_t^i = j)}. \tag{10.2.4.8}$$

Thus:

$$\alpha_{t+1,j} = \alpha_{t,j} + \sum_{i=1}^{N_t} \mathbb{I}(y_t^i = j). \tag{10.2.4.9}$$

Also, due to Gaussian-Gaussian conjugacy, then the posterior for the Gaussian prototype of the embedding for each class is:

$$\mathcal{N}(\boldsymbol{\mu}_{t+1}, \Lambda_{t+1}) \propto \prod_{i=1}^{N_t} \mathcal{N}(\boldsymbol{z}_t^i | y_t^i; \bar{\boldsymbol{z}}_{y_t}, \Sigma_\epsilon) \mathcal{N}(\bar{\boldsymbol{z}}_{y_t}; \boldsymbol{\mu}_{y_t,t}, \Lambda_{y_t}^{-1}) \tag{10.2.4.10}$$

$$= \prod_{y_t \in \{1,\ldots,J\}} \mathcal{N}(\boldsymbol{z}_{y_t} | y_t; \bar{\boldsymbol{z}}_{y_t}, \frac{1}{N_{y_t}} \Sigma_\epsilon) \mathcal{N}(\bar{\boldsymbol{z}}_{y_t}; \boldsymbol{\mu}_{y_{t+1}}, \Lambda_{y_t}^{-1}) \tag{10.2.4.11}$$

$$= \prod_{y_t \in \{1,\ldots,J\}} \mathcal{N}(\bar{\boldsymbol{z}}_{y_t}; \boldsymbol{\mu}_{t+1}, \Lambda_{y_{t+1}}^{-1}), \tag{10.2.4.12}$$

where N_{y_t} is the number of points of class y_t from the set of all classes $C = \{1, \ldots, J\}$. The update equations for the mean and variance of the posterior are:

$$\Lambda_{y_{t+1}} = \Lambda_{y_t} + N_{y_t} \Sigma_\epsilon^{-1}, \quad \forall y_t \in C_t \tag{10.2.4.13}$$

$$\Lambda_{y_{t+1}} \boldsymbol{\mu}_{y_{t+1}} = N_{y_t} \Sigma_\epsilon^{-1} \bar{\boldsymbol{z}}_{y_t} + \Lambda_{y_t} \boldsymbol{\mu}_{y_t}, \quad \forall y_t \in C_t. \tag{10.2.4.14}$$

10.2.4.3 ProtoCL Objective

The posterior predictive distribution we want to optimize is:

$$p(\boldsymbol{z}, y) = \int p(\boldsymbol{z}, y | \theta; \eta) p(\theta; \eta) d\theta, \tag{10.2.4.15}$$

where $p(\theta)$ denotes the distributions over class probabilities $\{p_j\}_{j=1}^J$ and mean embeddings $\{\bar{\boldsymbol{z}}_{y_j}\}_{j=1}^J$,

$$p(\boldsymbol{z}, y) = \int \prod_{i=1}^{N_t} p(\boldsymbol{z}_{it}|y_{it}; \bar{\boldsymbol{z}}_{y_t}, \Sigma_\epsilon) p(y_{it}|p_{1:J}) p(p_{1:J}; \alpha_t) p(\bar{\boldsymbol{z}}_{y_t}; \boldsymbol{\mu}_{y_t,t}, \Lambda_{y_t,t}^{-1}) dp_{1:J} d\bar{\boldsymbol{z}}_{y_t} \tag{10.2.4.16}$$

$$= \int \prod_{i=1}^{N_t} p(\boldsymbol{z}_{it}|y_{it}; \boldsymbol{z}_{y_t}, \Sigma_\epsilon) p(\bar{\boldsymbol{z}}_{y_t}; \boldsymbol{\mu}_{y_t,t}, \Lambda_{y_t,t}^{-1}) d\bar{\boldsymbol{z}}_{y_t} \underbrace{\int \prod_{i=1}^{N_t} p(y_{it}|p_{1:J}) p(p_{1:J}; \alpha_t) dp_{1:J}}_{\prod_i p(y_i)=p(y)} \tag{10.2.4.17}$$

$$= p(y) \prod_{i=1}^{N_t} Z_i^{-1} \int \mathcal{N}(\bar{\boldsymbol{z}}_{y_{it}}; \boldsymbol{c}, C) d\bar{\boldsymbol{z}}_{y_t} \tag{10.2.4.18}$$

$$= p(y) \prod_{i=1}^{N_t} \mathcal{N}(\boldsymbol{z}_{it}|y_{it}; \boldsymbol{\mu}_{y_t,t}, \Sigma_\epsilon + \Lambda_{y_t,t}^{-1}). \tag{10.2.4.19}$$

Where in Eq. (10.2.4.18) we use §8.1.8 in (Petersen et al., 2008). The term $p(y)$ is:

$$p(y) = \int p(y|p_{1:J}) p(p_{1:J}; \alpha_t) dp_{1:J} \tag{10.2.4.20}$$

$$= \int p_y \frac{\Gamma(\sum_{j=1}^J \alpha_j)}{\prod_{j=1}^J \Gamma(\alpha_j)} \prod_{j=1}^J p_j^{\alpha_j - 1} dp_{1:J} \tag{10.2.4.21}$$

$$= \frac{\Gamma(\sum_{j=1}^J \alpha_j)}{\prod_{j=1}^J \Gamma(\alpha_j)} \int \prod_{j=1}^J p_j^{\mathbb{I}(y=j)+\alpha_j - 1} dp_{1:J} \tag{10.2.4.22}$$

$$= \frac{\Gamma(\sum_{j=1}^J \alpha_j)}{\prod_{j=1}^J \Gamma(\alpha_j)} \frac{\prod_{j=1}^J \Gamma(\mathbb{I}(y=j) + \alpha_j)}{\Gamma(1 + \sum_{j=1}^J \alpha_j)} \tag{10.2.4.23}$$

$$= \frac{\cancel{\Gamma(\sum_{j=1}^J \alpha_j)}}{\prod_{j=1}^J \Gamma(\alpha_j)} \frac{\prod_{j=1}^J \Gamma(\mathbb{I}(y=j) + \alpha_j)}{\sum_{j=1}^J \alpha_j \cancel{\Gamma(\sum_{j=1}^J \alpha_j)}} \tag{10.2.4.24}$$

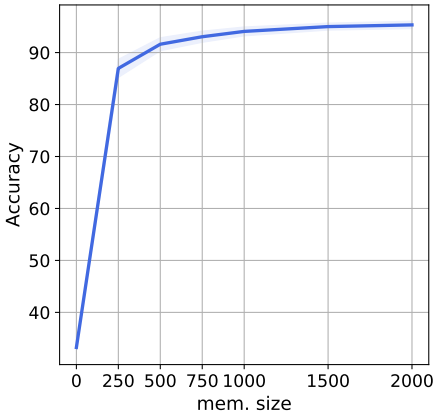
$$= \frac{\prod_{j=1, j\neq y}^J \Gamma(\alpha_j)}{\prod_{j=1}^J \Gamma(\alpha_j)} \frac{\Gamma(1 + \alpha_y)}{\sum_{j=1}^J \alpha_j} \tag{10.2.4.25}$$

$$= \frac{\prod_{j=1, j\neq y}^J \Gamma(\alpha_j)}{\prod_{j=1}^J \Gamma(\alpha_j)} \frac{\alpha_y \Gamma(\alpha_y)}{\sum_{j=1}^J \alpha_j} \tag{10.2.4.26}$$

$$= \frac{\alpha_y}{\sum_{j=1}^J \alpha_j}, \tag{10.2.4.27}$$

where we use the identity $\Gamma(n+1) = n\Gamma(n)$.

Figure 10.18: Split-MNIST average test accuracy over 5 tasks for different memory sizes. On the x-axis we show the size of the entire memory buffer shared by all 5 tasks. Accuracies are over a mean and standard deviation over 5 different runs with different random seeds.



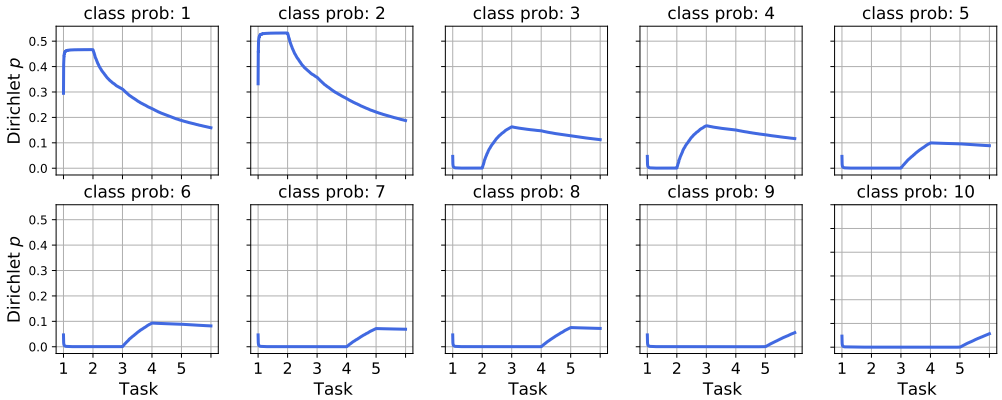


Figure 10.19: The evolution of the Dirichlet parameters α_t for each class in Split-MNIST tasks for ProtoCL. All α_j are shown over 10 seeds with ± 1 standard error. By the end of training, all classes are roughly equally likely, as we have trained on equal amounts of all classes.

10.2.4.4 Predictions

To make a prediction for a test point $\boldsymbol{x}^*$:

$$p(y^* = j|\boldsymbol{x}^*, \boldsymbol{x}_{1:t}, y_{1:t}) = p(y^* = j|\boldsymbol{z}^*, \theta_t) \tag{10.2.4.28}$$

$$= \frac{p(\boldsymbol{z}^*|y^* = j, \theta_t)p(y^* = j|\theta_t)}{\sum_i p(\boldsymbol{z}^*|y^* = i, \eta_t)p(y^* = i|\theta_t)} \tag{10.2.4.29}$$

$$= \frac{p(y^* = j, \boldsymbol{z}^*|\theta_t)}{\sum_i p(y = i, \boldsymbol{z}^*|\theta_t)}, \tag{10.2.4.30}$$

where θ_t are sufficient statistics for $(\boldsymbol{x}_{1:t}, y_{1:t})$.

Preventing forgetting. As we wish to retain the task-specific prototypes, at the end of learning a task $\mathcal{T}_t$ we take a small subset of the data as a memory to ensure that posterior parameters and prototypes do not drift, see Algorithm 1.

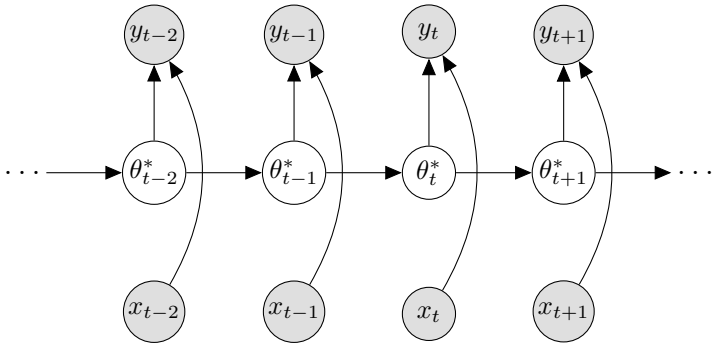


Figure 10.20: Graphical model of under which we perform inference in Section 5.5. Grey nodes are observed and white are latent variables.

10.2.4.5 Experimental Setup

The prototype variance, Σ_ϵ is set to a diagonal matrix with the variances of each prototype set to 0.05. The prototype prior precisions, Λ_{yt} are also diagonals and initialized randomly and exponentiated to ensure a positive semi-definite covariance for the sequential updates. The parameters $\alpha_j \, \forall j$ are set to 0.78 which was found by random search over the validation set on MNIST. We also allow α_j to be learned in the gradient update step in addition to the sequential update step (lines 4 and 5 Algorithm 1), see Fig. 10.19 to see the evolution of the α_j or all classes j over the course of learning Split-MNIST.

For the Split-MNIST and Split-FMNIST benchmarks, we use an NN with 2 layers of size 200 and trained for 50 epochs with an Adam optimizer. We perform a grid search over learning rates, dropout rates, and weight decay coefficients. The embedding dimension is set to 128. For the Split-CIFAR10 and Split-CIFAR100 benchmarks, we use the same network as Pan et al. (2020) which consists of 4 convolution layers and 2 linear layers. We train the networks for 80 epochs for each task with the Adam optimizer with a learning rate of $1e-3$. The embedding dimension is set to 32. All experiments are run on a single GPU NVIDIA RTX 3090.

10.2.5 Sequential Bayesian Estimation as Bayesian Neural Network optimization

We shall consider inference in the graphical model depicted in Fig. 10.20. The aim is to infer the optimal BNN weights, θ^*_t at time t given observations and

the previous BNN weights. We assume a Gaussian posterior over parameters with full covariance hence we model interactions between all weights. We shall consider the online setting where we see one data point $(\boldsymbol{x}_t, y_t)$ at a time and we make no assumption as to whether the data comes from the same task or different tasks over the course of learning.

We set up the problem of sequential Bayesian inference as a filtering problem and we leverage the work of Aitchison (2020) which casts NN optimization as Bayesian sequential inference. We make the reasonable assumption that the distribution over weights is a Gaussian with full covariance. Since reasoning about the full covariance matrix of a BNN is intractable we instead consider the i-th parameter and reason about the dynamics of the optimal estimates θ_{it}^* as a function of all the other parameters θ_{-it}. Each weight is functionally dependent on all others. If we had access to the full covariance of the parameters then we could reason about the unknown optimal weights θ_{it}^* given the values of all the other weights θ_{-it}. However, since we do not have access to the full covariance another approach is to reason about the dynamics of θ_{it}^* given the dynamics of θ_{-it} and assume that the values of the weights are close to those of the previous time-step (Jacot et al., 2018) and so we cast the problem as a dynamical system.

Consider a quadratic loss of the form:

$$\mathcal{L}(\boldsymbol{x}_t, y_t; \theta) = \mathcal{L}_t(\theta) = -\frac{1}{2}\theta^\top \boldsymbol{H}\theta + \mathbf{z}_t^\top \theta, \tag{10.2.5.1}$$

which we can arrive by simple Taylor expansion where $\boldsymbol{H}$ is the Hessian which is assumed to be constant across data points but not across the parameters θ. If the BNN output takes the form $f(\boldsymbol{x}_t; \theta)$, then the derivative evaluated at θ_t is $\boldsymbol{z}_t = \frac{\partial \mathcal{L}(\boldsymbol{x}_t, y_t; \theta)}{\partial \theta}|_{\theta=\theta_t}$. To construct the dynamical equations for our weights, consider the gradient for a single datapoint:

$$\frac{\partial \mathcal{L}_t(\theta)}{\partial \theta} = -\boldsymbol{H}\theta + \mathbf{z}_t. \tag{10.2.5.2}$$

If we consider the gradient for the i-th weight while all other parameters are set to their current estimate:

$$\frac{\partial \mathcal{L}(\theta_i, \theta_{-i})}{\partial \theta_i} = -H_{ii}\theta_{it} - \boldsymbol{H}_{-ii}^{\top}\theta_{-it} + z_{ti}. \tag{10.2.5.3}$$

When the gradient is set to zero we recover the optimal value for θ_{it}, denoted as θ_{it}^*:

$$\theta_{it}^* = -\frac{1}{H_{ii}}\boldsymbol{H}_{-ii}^{\top}\theta_{-it}. \tag{10.2.5.4}$$

since $z_{ti} = 0$ at the modes. The equation above shows us that the dynamics of the optimal weight θ_{it}^* is dependent on all the other current values of the parameters θ_{-it}. That is, the dynamics of θ_{it}^* are governed by the dynamics of the weights θ_{-it}. The dynamics of θ_{-it} are a complex stochastic process dependent on many different variables. Since reasoning about the dynamics is intractable we instead assume a discretized Ornstein-Uhlenbeck process for the weights θ_{-it} with reversion speed $\vartheta \in \mathbb{R}_+$ and noise variance η_{-i}^2:

$$p(\theta_{-i,t+1}|\theta_{-i,t}) = \mathcal{N}((1-\vartheta)\theta_{-it}, \eta_{-i}^2), \tag{10.2.5.5}$$

this implies that the dynamics for the optimal weight is defined by

$$p(\theta_{i,t+1}^*|\theta_{i,t}^*) = \mathcal{N}((1-\vartheta)\theta_{it}^*, \eta^2), \tag{10.2.5.6}$$

where $\eta^2 = \eta_{-i}^2 \boldsymbol{H}_{-ii}^{\top}\boldsymbol{H}_{-ii}$. This same assumption is made in Aitchison (2020). This assumes a parsimonious model of the dynamics. Together with our likelihood:

$$p(y_t|\boldsymbol{x}_t; \theta_t^*) = \mathcal{N}(y_t; f(\boldsymbol{x}_t; \theta_t^*), \sigma^2) \tag{10.2.5.7}$$

where $f(\,\cdot\,, \theta)$ is a neural network prediction with weights θ, we can now define a linear dynamical system for the optimal weight θ_i^* by linearizing the Bayesian NN (Jacot et al., 2018) and by using the transition dynamics in Eq. (10.2.5.6). Thus we are able to infer the posterior distribution over the optimal weights using Kalman filter-like updates (Kalman, 1960). As the dynamics and likelihood are

Gaussian, then the prior and posterior are also Gaussian, for ease of notation we drop the index i such that $\theta^*_{it} = \theta^*_t$:

$$p(\theta^*_t|(\boldsymbol{x}, y)_{t-1}, \ldots, (\boldsymbol{x}, y)_1) = \mathcal{N}(\mu_{t,\text{prior}}, \sigma^2_{t,\text{prior}}) \tag{10.2.5.8}$$

$$p(\theta^*_t|(\boldsymbol{x}, y)_t, \ldots, (\boldsymbol{x}, y)_1) = \mathcal{N}(\mu_{t,\text{post}}, \sigma^2_{t,\text{post}}) \tag{10.2.5.9}$$

By using the transition dynamics and the prior we can obtain closed-form updates:

$$p(\theta^*_t|(\boldsymbol{x}, y)_{t-1}, \ldots, (\boldsymbol{x}, y)_1) = \int p(\theta^*_t|\theta^*_{t-1}) p(\theta^*_{t-1}|(\boldsymbol{x}, y)_{t-1}, \ldots, (\boldsymbol{x}, y)_1) d\theta^*_{t-1} \tag{10.2.5.10}$$

$$\mathcal{N}(\theta^*_t; \mu_{t,\text{prior}}, \sigma^2_{t,\text{prior}}) = \int \mathcal{N}(\theta^*_t; (1-\vartheta)\theta^*_{t-1}, \eta^2) \mathcal{N}(\theta^*_{t-1}; \mu_{t-1,\text{post}}, \sigma^2_{t-1,\text{post}}) d\theta^*_{t-1}. \tag{10.2.5.11}$$

Integrating out θ^*_{t-1} we can get updates for the prior for the next timestep as follows:

$$\mu_{t,\text{prior}} = (1-\vartheta)\mu_{t-1,post} \tag{10.2.5.12}$$

$$\sigma^2_{t,\text{prior}} = \eta^2 + (1-\vartheta)^{-2}\sigma^2_{t-1,\text{post}}. \tag{10.2.5.13}$$

The updates for obtaining our posterior parameters: $\mu_{t,\text{post}}$ and $\sigma^2_{t,\text{post}}$, comes from applying Bayes' theorem:

$$\log \mathcal{N}(\theta^*_t; \mu_{t,\text{post}}, \sigma^2_{t,\text{post}}) \propto \log \mathcal{N}(y_t; f(\boldsymbol{x}_t; \theta^*_t), \sigma^2) + \log \mathcal{N}(\theta^*_t; \mu_{t,\text{prior}}, \sigma^2_{t,\text{prior}}), \tag{10.2.5.14}$$

by linearizing our Bayesian NN such that $f(\boldsymbol{x}_t, \theta_0) \approx f(\boldsymbol{x}_t, \theta_0) + \frac{\partial f(\boldsymbol{x}_t; \theta^*_t)}{\partial \theta^*_t}(\theta^*_t - \theta_0)$ and by substituting into Eq. (10.2.5.14) we obtain our update equation for the posterior of the mean of our BNN parameters:

$$-\frac{1}{2\sigma^2_{t,\text{post}}}(\theta^*_t - \mu_{t,\text{post}})^2 = -\frac{1}{2\sigma^2}(y - g(\boldsymbol{x}_t)\theta^*_t)^2 - \frac{1}{2\sigma^2_{t,\text{prior}}}(\theta^*_t - \mu_{t,\text{prior}})^2 \tag{10.2.5.15}$$

$$\mu_{t,\text{post}} = \sigma^2_{t,\text{post}} \left(\frac{\mu_{t,\text{prior}}}{\sigma^2_{t,\text{prior}}} + \frac{y}{\sigma^2} g(\boldsymbol{x}_t) \right), \tag{10.2.5.16}$$

where $g(\boldsymbol{x}_t) = \frac{\partial f(\boldsymbol{x}_t; \theta^*_t)}{\partial \theta^*_t}$, and the update equation for the variance of the Gaussian posterior is:

$$\frac{1}{\sigma^2_{t,\text{post}}} = \frac{g(\boldsymbol{x}_t)^2}{\sigma^2} + \frac{1}{\sigma^2_{t,\text{prior}}}. \tag{10.2.5.17}$$

From our update equations Eq. (10.2.5.16) and Eq. (10.2.5.17) we can notice that the posterior mean depends linearly on the prior and an additional data dependent term. These equations are similar to the filtering example in Section 5.4. So under certain assumptions above, a BNN should behave similarly. If there exists a task data imbalance then the data term will dominate the prior term in Eq. (10.2.5.16) and could lead to forgetting of previous tasks.

10.3 Same State Different Task: Continual Reinforcement Learning without Interference

10.3.1 Limitations

In Q-learning it is the ranking of the Q-values conditioned on a state that is important rather than the accuracy of the Q-values themselves; indeed we observe empirically that TD error increases with training as we encounter higher return episodes. With very diverse tasks (e.g., sparse *and* dense reward) this could result in a diverse range of TD errors which will require normalization when trying to facilitate MAB policy selection; for instance, policy heads corresponding to sparser reward tasks may be over-selected simply because they have intrinsically lower TD error. We have chosen similar tasks to demonstrate interference where TD error is comparable across tasks but accept that tasks with diverse TD error may need different feedback to the MAB.

There is a statistically significant benefit to using the MAB for in-distribution tasks that have been seen during continual training. However, the benefits do not translate for tasks that are unseen. The default behaviour of the MAB is to behave randomly and assign a uniform distribution/prior on all policy heads; this is intended. It seems that using DQN as a base learner for 5 tasks is not enough for our policy to truly generalize to the distribution of tasks. Thus the feedback to MAB ensures that MAB probabilities are uniform. This is shown in Fig. 10.21, on the left we show the TD errors of the seen task where the TD error for arm 3 is the lowest; the MAB is correctly picking the arm/policy to solve the task. On the right, an unseen 5 wall task where all TD errors are high: there isn't a signal for the MAB to latch onto. There isn't enough generalization in the 3 policies of this agent for the agent to assign a non-uniform distribution for the MAB arms/policies hence the random behaviour of the MAB. If we could endow DQN with invariances through for instance self-supervision such that the policies can generalize then the feedback to the MAB would be stronger and outperform randomly picking the policy head.

We have on purpose picked the Minigrid sparse reward environments as they are difficult to solve. If the MAB has a dense reward like in the Pendulum then it can quite easily settle on the correct task just by using the reward as feedback.

A strength of OWL is its simplicity. However, there is still a performance gap between the optimal performance achievable: FR versus OWL. There is still some forgetting as we scale to more tasks and train for less time. Thus the effectiveness of EWC as a method to prevent forgetting is a further limitation. Our paper focuses on preventing interference and task inference at test time but the use of stronger methods to prevent forgetting is an interesting extension.

10.3.2 MultiMNIST Experiments

In terms of dataset construction: two different classes from the MNIST dataset are placed on the top left and bottom right of each image and each randomly shifted 4 pixels in each direction. The resulting images are of size 64×64. The first task $\mathcal{T}_1$ requires classifying the top right image and the second task $\mathcal{T}_2$ requires classifying the bottom right: these two tasks are conflicting. We use EWC and GEM continual learning strategies (Kirkpatrick et al., 2016; Lopez-Paz and Ranzato, 2017) with LeNet backbones.

10.3.3 Implementation Details

In this section, we detail the implementation of OWL applied for SAC which was used in the continuous control setting with the Pendulum-v0 environments. Also, we detail the implementation of OWL applied to DQN for solving the MiniGrid environments.

All experiments were run on a single NVIDIA RTX 3090 GPU, with multiple seeds fitting onto the GPU.

10.3.3.1 Conflicting Pendulums: Soft Actor-Critic

We implement Soft Actor-Critic (SAC) in PyTorch, following the learned reward temperature approach of (Haarnoja et al., 2018c). Briefly, Soft Actor-Critic aims

to maximize the sum of the reward and the entropy of the policy over the task horizon; this results in behaviour that can be summarized as "maximizing reward while acting as randomly as possible", and can be shown as being optimal in a meta-POMDP setting (Eysenbach and Levine, 2019). What is difficult about such a dual objective is that ultimate task performance is sensitive to the trade-off between randomness/entropy and reward (Haarnoja et al., 2018b). In the continual setting that we present here, different tasks over the agent's lifetime may require different degrees of reward/entropy scaling, which would require specific tuning of this parameter (usually denoted as α) per new task. It is possible however to learn this parameter α from the task by introducing a 'target entropy' ($\bar{\mathcal{H}}$), which actively scales reward against policy entropy such that the expected entropy matches the target $\bar{\mathcal{H}} = -\dim(\mathcal{A})$, where $\mathcal{A}$ is the action space of an environment. This results in α that adapts to its current policy and experiences it receives from the environment, as well as an appropriate degree of randomness for each environment.

For Exp Replay and SAC we use a buffer of size 1M, OWL uses an EWC regularization strength of $\lambda = 100$. In this setting, we use an ensemble of probabilistic networks and thus we are able to provide a negative log-likelihood of the Q-values for each policy as feedback to the bandit algorithm $l_{i_t}^t$ for deciding which Q-function and policy to use to solve the task which is being evaluated on.

The Q-function used for OWL is identical to the architecture from PETS which has been shown to be successful in Model-Based RL (Chua et al., 2018). OWL uses a multi-head architecture where the number of heads equals the number of tasks. Each head has a mean and variance mapping (Nix and Weigend, 1994). SAC uses Q-functions with four layers with ReLU activations. An ensemble of two Q-functions is used for compensation for an optimistic Q-value as is standard for SAC. The target Q-function $Q(\cdot, \cdot; \theta_{i-1})$ is updated using an exponentially weighted moving average with $\gamma = 0.99$.

The Gaussian policy is parameterized by a NN which uses mean and variance heads that share three NN layers with ReLU activations, it is trained with a learning

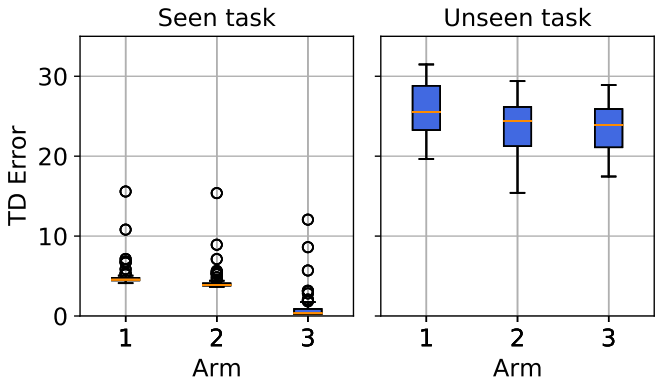


Figure 10.21: TD errors for each head of a 3 head/policy OWL agent for a task which OWL has seen before **Left** task 3 in Fig. 6.5. **Right**, TD errors for an unseen 5 walled SimpleCrossing task.

rate of 3×10^{-4}. Gradients are propagated through the Gaussian policy using the reparameterization trick (Kingma and Welling, 2013).

Elastic Weight Consolidation. The EWC implementation for both the Q-function and policy uses $\lambda = 100$, we didn't attempt to optimize λ. The weighting of the L^2 regularization is performed with the empirical Fisher Information $F(\theta) = \sum_n \nabla_\theta \log p_\theta(y_n|x_n) \nabla_\theta \log p_\theta(y_n|x_n)^\top$ which is calculated using all samples from the experience replay buffer up to a maximum of $60,000$ samples. EWC is applied on the feature extraction layers of the Q-function and policy and to the task-specific mean and variance heads for when they are trained again. No EWC regularization is directly applied to the target Q-function.

Online Learning. Two Q-values are estimated by SAC and OWL. In OWL the Q-function outputs a mean and variance, these estimates can be combined by considering the Q-values as a mixture of Gaussians. The Gaussian mixture of L Q-functions has mean and variance $\mu_* = \frac{1}{L}\sum_{l=1}^{L} \mu_l(x)$ and $\sigma_*^2 = \sum_{l=1}^{L}(\sigma_l^2(x) - \mu_l^2(x)) - \mu_*^2(x)$ (Lakshminarayanan et al., 2017b). These estimates are then used as the basis for MSE and negative log-likelihood feedback to the Exponentially Weighted Average Forecaster Bandit algorithm (ExpWeights). More specifically the feedback is $G_{\phi_i}(\theta_i) = \log \sigma_*^2(s_t, a_t; \theta_i) + \frac{(\mu_*(s_t,a_t;\theta_i)) - \hat{\mu}_*(s_t,a_t;\theta_i)}{\sigma_*^2(s_t,a_t;\theta_i)}$ for head θ_i and $\hat{\mu}(\cdot)$ is the target mean Q-function, the feedback to the MAB is thus $l_{i_t} = 1/G_{\phi_i}(\theta_i)$.

The losses supplied to ExpWeights are sometimes very large, especially for feedback in the form of an MSE. Hence we set a threshold to $l_{i_t} = \min(50, l_{i_t})$ where l is a loss, those considered are inverse negative log-likelihood and inverse MSE, i_t denotes the index of the policy chosen at time t in the rollout. The step size is set to $\eta = 0.98$. We haven't attempted to optimize these. The MAB algorithm that we use for OWL is summarized in Algorithm 3.

10.3.3.2 Minigrid: DQN

We use a DQN implementation to solve the MiniGrid environments and adapt our OWL method accordingly. For the Q-function we use Duelling Networks (Wang et al., 2016) and use Double-DQN for estimating the Q-value for a state-action (Van Hasselt et al., 2015). For the Minigrid environment, the inputs are images of size $7\times7\times3$. We focus on tasks generated from the SimpleCrossingS9N1 environment. We use a CNN to as a function approximator for learning Q-values, see Table 10.5 for a description of the architecture used. To enable exploration of the Minigrid state-space we use an ϵ-greedy exploration strategy which is annealed over the course of training. In addition, we use a count-based reward bonus for visiting different places on the grid and if the agent reaches the goal during training the reward is multiplied by 100 to boost the signal and differentiate the reward signal from the state-based exploration bonus. We limit the action space to $\mathcal{A} = \{\text{left}, \text{right}, \text{forward}\}$ for the SC agent.

We use an experience replay buffer of size 1M frames which is standard in DQN implementations. Crucially this buffer is emptied when learning a new task with OWL. For our Exp. Replay baseline experiences persist across different tasks. The target Q-function network is assigned the parameters from the Q-function every 80 optimization steps. The Adam optimizer (Kingma and Lei Ba, 2015) is used for the Q-function with a Huber loss. See Table 10.4 for the list of hyperparameters used and their values, most values are set to best practice values (Hessel et al., 2017).

OWL. To enable learning multiple tasks sequentially without interference we use a shared feature extractor and a different linear head per task. Additionally, we clear the experience replay buffer before learning a new task so that experiences

Table 10.4: DQN hyperparameters used for Exp. Replay and OWL methods.

Parameter	Value
Min number of random experiences to start learning	10K frames
Discount	0.99
Adam learning rate	0.0000625
Batch size	32
Start exploration ϵ	0.9
Min exploration ϵ	0.01
Exploration decay rate	250k steps
Target network update frequency	Every 80 opt. steps
Q-function update frequency	Every 4 env. steps
Experience replay size	1M frames
Maximum evaluation steps in env.	100
Number of episodes for evaluation	16
Frames per task	1M

do not interfere. To alleviate forgetting we use EWC regularization of the shared feature extractor. We found $\lambda = 500$ worked well, see Section 10.3.4.

Online Learning. We compute the feedback to the bandit algorithm similarly to the Pendulum-v0 experiment. However, we use the TD error as feedback $G_{\phi_i}(\theta_i) = (Q(s_t, a_t; \theta_i) - \hat{Q}(s_t, a_t; \theta_i))^2$ for Q-function head θ_i, ϕ_i is the policy head which is the same as θ_i in DQN since there is no separate policy like in SAC and $\hat{Q}(\cdot)$ is the target. We found that the TD error worked well as feedback. We optimized the bandit algorithm's hyperparameters and found $\eta = 0.88$ worked well.

10.3.4 Ablation Studies

10.3.4.1 Conflicting Pendulums

Do we need uncertainty-aware models? The feedback for our online learning mechanism for OWL adapted to SAC incorporates both the mean and variance predictions, thus considering aleatoric uncertainty. To assess the impact of this, we also ran OWL with a simple MSE feedback, ignoring the variance head. In Fig. 10.22 we show both the performance of this approach, as well as the effectiveness of the online learning mechanism.

Table 10.5: DQN Q-function architecture used for Exp. Replay and OWL methods.

Layer	Channel	Kernel	Stride	Padding
Input 7×7	3	-	-	-
Conv 1	16	(2×2)	1	0
ReLU	-	-	-	-
Max Pool 2-d	16	(2×2)	2	0
Conv 2	32	(2×2)	1	0
ReLU	-	-	-	-
Conv 3	64	(2×2)	1	0
ReLU	-	-	-	-
Flatten	-	-	-	-
Linear	200	-	-	-
ReLU	-	-	-	-
Value Stream	1	-	-	-
Advantage Stream	$\lvert\mathcal{A}\rvert$	-	-	-

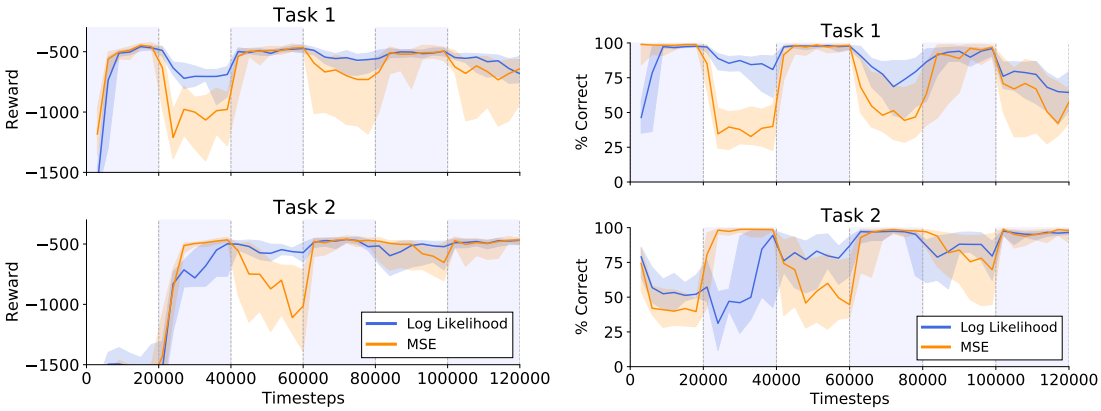


Figure 10.22: Left: Median performance across 10 seeds. The shaded region enveloping the lines correspond to the interquartile range. Right: The performance of the online learning mechanism for both tasks. Pale blue alternating tiles correspond to the task that is currently being trained on.

It is clear to see here that the negative log-likelihood is significantly more effective. The linkage between selecting the correct policy and final performance is evident and justifies our use of the probabilistic models.

10.3.4.2 Minigrid

We analyze the performance of the different OWL variants by looking at performance on 2 Minigrid tasks repeated twice.

EWC regularization strength: We can vary the strength of the EWC

regularization, λ. The larger λ the less forgetting of previous tasks, however, the additional regularizations can inhibit learning new tasks. We do not want λ to be too large or too small. We found $\lambda = 500$ to work well, in general performance isn't very sensitive to λ, see Table 10.6.

Functional regularization: Different parameterizations of a NN can yield the same output function. Instead of regularizing the weights of an NN it is more desirable to regularize the output function of an NN such that past outputs look the same as current outputs (Hinton et al., 2015; Li and Hoiem, 2017b; Benjamin et al., 2019). Since we use a shared feature extractor f_{θ_z} and different outputs heads $\theta_{1:M}$ to parameterize the Q-function. At the end of training on $\mathcal{T}_\tau$ we can cache the parameters of the model and use this old model to regularize the new model. While training the new task we can ensure that the shared feature extractor's outputs do not deviate from old values by regularizing the outputs with respect to the previous task's outputs. The functional regularization for task $\mathcal{T}_{\tau+1}$ is

$$\mathcal{L}_{\text{func}}^{\tau+1} = \mu \, \text{KL}(Q_{\tau+1}((\cdot,\cdot), Q_\tau(\cdot,\cdot)) \tag{10.3.4.1}$$

$$= \mu \left(g_{\theta_\tau} \circ f_{\theta_{z,\tau+1}}(\cdot,\cdot) - g_{\theta_\tau} \circ f_{\theta_{z,\tau}}(\cdot,\cdot) \right)^2. \tag{10.3.4.2}$$

Where we assume that the variances of the Q-functions are equal and constant. θ_τ are the output head parameters of the previous task and g_{θ_τ} is the linear head, $\circ$ denotes a composition of functions. $\theta_{z,\tau+1}$ are the parameters of the shared feature extractor for the current task $\mathcal{T}_{\tau+1}$ and μ is a hyperparameter which controls the strength of the regularization. On 2 Minigrid tasks repeated twice, we found that the functional regularization doesn't perform as well as EWC regularization for different values of the regularization strength $\mu \in \{0.1, 1, 10, 100\}$. The best value of μ we found was 100.

Warm starting the replay buffer: DQN initializes the experience replay buffer by drawing 10^5 random actions before using the DQN policy to select actions for the first task. In OWL we flush the experience replay buffer when starting to train in a new task to avoid interference, however, this means that our experience replay buffer is empty for a new task and we would start training with no experiences.

Table 10.6: Median proportion of successes for 5 seeds after 4M frames for 2 Minigrid tasks (75-th quartile, 25-th quartile). Performance is stable to different EWC regularization strengths, λ.

	Oracle		Adaptive		
	Final Perf.	Cumulative Perf.	Final Perf.	Cumulative Perf.	Avg. Rank
Exp Replay	0.00 (0.09, 0.00)	0.38 (0.87, 0.01)	-	-	4.0
OWL $\lambda = 200$	1.00 (1.00, 1.00)	0.84 (0.91, 0.76)	1.00 (1.00, 0.79)	0.73 (0.85, 0.61)	2.0
OWL $\lambda = 500$	1.00 (1.00, 1.00)	0.92 (0.95, 0.89)	1.00 (1.00, 0.64)	0.75 (0.88, 0.61)	1.25
OWL $\lambda = 1000$	1.00 (1.00, 1.00)	0.91 (0.92, 0.75)	1.00 (1.00, 1.00)	0.86 (0.89, 0.84)	1.25

Table 10.7: Median proportion of successes for 5 seeds after 4M frames and cumulative performance over 4M frames and (75-th quartile, 25-th quartile). All ablations have EWC regularization with $\lambda = 500$.

	Oracle		Adaptive		
	Final Perf.	Cumulative Perf.	Final Perf.	Cumulative Perf.	Avg. Rank
Exp Replay	0.00 (0.09, 0.00)	0.38 (0.87, 0.01)	-	-	6.0
OWL warm start 50k	1.00 (1.00, 1.00)	0.92 (0.93, 0.90)	1.00 (1.00, 0.88)	0.88 (0.89, 0.81)	1.0
OWL warm start 100k	1.00 (1.00, 1.00)	0.92 (0.95, 0.87)	1.00 (1.00, 0.88)	0.81 (0.87, 0.67)	1.5
OWL ϵ-greedy warm start $1 \times \tau_\epsilon$	1.00 (1.00, 1.00)	0.78 (0.85, 0.68)	0.97 (1.00, 0.80)	0.71 (0.75, 0.64)	3.5
OWL ϵ-greedy warm start $2 \times \tau_\epsilon$	1.00 (1.00, 1.00)	0.82 (0.87, 0.69)	0.80 (0.98, 0.62)	0.64 (0.78, 0.60)	3.5
OWL MLP head	1.00 (1.00, 1.00)	0.88 (0.92, 0.88)	1.00 (1.00, 1.00)	0.85 (0.89, 0.75)	1.75

Alternatively, we could sample some experiences from each task and use these to warm start the experience replay buffer when revisiting these tasks. We found this increased the performance of OWL when warm starting the experience replay buffer with $B = \{5 \times 10^4, 10^5\}$ experiences, see Table 10.7. Our results in Table 10.9 include this technique with $B = 5 \times 10^4$.

Restarting the ϵ greedy exploration strategy: We keep a separate ϵ-greedy exploration schedule for each task for OWL. When we revisit a task we simply look up the schedule we were using the last time we were training in the task and continue where the ϵ-greedy schedule left off, this approach is used in Kirkpatrick et al. (2016). However, when revisiting a task it makes sense to add a bit of exploration as a warm start for the experience replay buffer, random actions are added to the experience replay buffer when training on DQN. We try resetting the ϵ-greedy schedule when revisiting a task and annealing at the same rate ($1 \times \tau_\epsilon$) or annealing ϵ at twice the rate when revisiting the task ($2 \times \tau_\epsilon$). We found poor performance for both of these scenarios. It is best to keep an ϵ-greedy schedule for each task and restart the schedule where the DQN agent left off when revisiting a task, see Table 10.7.

Table 10.8: Median proportion of successes for 5 seeds after 4M frames and cumulative performance over 4M frames of 2 different MiniGrid tasks repeated twice (75-th quartile, 25-th quartile). Enabling dropout regularization and using a negative log-likelihood feedback for the bandit algorithm hurts performance. Average ranks computed for warm-start and no warm-start OWL methods separately with Experience Replay.

	Oracle		Adaptive		
	Final Perf.	Cumulative Perf.	Final Perf.	Cumulative Perf.	Avg. Rank
Exp Replay	0.00 (0.09, 0.00)	0.38 (0.87, 0.01)	-	-	4.0
OWL $\lambda = 500$	1.00 (1.00, 1.00)	0.92 (0.95, 0.89)	1.00 (1.00, 0.64)	0.75 (0.88, 0.61)	1.25
OWL $d = 0.05$	1.00 (1.00, 1.00)	0.85 (0.93, 0.61)	1.00 (1.00, 0.87)	0.66 (0.74, 0.60)	1.75
OWL $d = 0.1$	1.00 (1.00, 0.25)	0.69 (0.76, 0.52)	0.88 (0.99, 0.20)	0.59 (0.74, 0.52)	2.75
OWL warm start 50k $d = 0.0$	1.00 (1.00, 1.00)	0.92 (0.93, 0.90)	1.00 (1.00, 0.88)	0.88 (0.89, 0.81)	1.0
OWL warm start 50k $d = 0.05$	1.00 (1.00, 1.00)	0.88 (0.93, 0.63)	1.00 (1.00, 0.88)	0.70 (0.82, 0.54)	1.5
OWL warm start 50k $d = 0.1$	1.00 (1.00, 1.00)	0.80 (0.85, 0.78)	0.95 (1.00, 0.40)	0.73 (0.80, 0.59)	2.25
OWL warm start 50k $d = 0.2$	0.99 (1.00, 0.24)	0.46 (0.62, 0.29)	0.77 (0.96, 0.17)	0.44 (0.57, 0.29)	3.75

MLP head: Multi-head models are a very useful tool and widely used in CL (Nguyen et al., 2017) a feature extractor $z = f_{\theta_z}(x)$ is shared for all tasks and a task-specific linear head $g_{\theta_i}(z)$ is appended for each individual task. We can add additional flexibility by making the head $g_{\theta_i}(z)$ a 2 layer MLP. This wasn't shown to help Table 10.7.

MC dropout: Can we obtain uncertainties for our Q-function and can this be a better guide/feedback for the bandit algorithm? We showed that providing the negative log-likelihood feedback improved results for SAC in Pendulum-v0. Can we obtain uncertainties for DQN? Having a probabilistic network with a variance head didn't work; the agent wasn't able to learn anything by optimizing a negative log-likelihood as an objective function instead of a Huber loss. We turn our attention to using MC dropout which is also very easy to implement for our current setup: we need to 1. train our Q-function with dropout and at evaluation 2. use dropout to make predictions (Gal and Ghahramani, 2016). We found that performance degrades considerably when enabling dropout at different rates Table 10.8. The Oracle performance decreases, hence training with dropout provides poor policies. Additionally using the resulting negative log-likelihood of the TD error as feedback also doesn't help the performance of the bandit algorithm. The main results Table 10.9 already show that using a TD error for feedback to the bandit algorithm is sufficient feedback for OWL solve for an environment.

Table 10.9: Median proportion of successes for three different MiniGrid environments (tasks) and inter-quartile range for 10 seeds after 9M frames and 3 task repeats and cumulative performance over the same period. Our methods are able to solve all three tasks, without knowledge of the test task, while Experience Replay fails due to interference.

	Final Perf.	**Cumulative Perf.**
Exp Replay	0.00 (1.00, 0.00)	0.37 (0.49, 0.12)
OWL (Orcl.)	1.00 (1.00, 1.00)	0.83 (0.94, 0.71)
OWL + WS (Orcl.)	1.00 (1.00, 1.00)	0.95 (0.97, 0.89)
OWL (Adpt.)	0.86 (1.00, 0.70)	0.72 (0.77, 0.62)
OWL + WS (Adpt.)	0.99 (1.00, 0.83)	0.79 (0.91, 0.73)

10.3.5 Scaling to 3 SimpleCrossing tasks.

Above, we considered a variety of design choices for OWL. In particular, we found that warm starting the experience replay buffer, denoted as OWL+WS in Table 10.9 with a small sample of experiences from the same task upon revisiting it worked well. Finally, we used $\lambda = 500$ for the EWC regularization strength. The results on 3 Minigrid levels which accompany the learning curves in Fig. 6.5 are summarized in Table 10.9.

10.3.6 Scaling to 5 Minigrid tasks

We scale to 5 `SimpleCrossing` tasks (denoted as SC in plots) and another set of 5 tasks where 3 are from `SimpleCrossing` and 2 from `DoorKey` (denoted as SC+DK). The set of tasks is repeated 3 times each task is seen for 0.75M environment steps. For Exp Replay we adjust the buffer size to 4M to ensure that data from all tasks are in the buffer over the course of training. Again we see that agents trained with Exp Replay suffer from interference as some tasks are never solved over the course of training Fig. 10.23 and Fig. 10.24. In contrast, our OWL agents are able to solve all tasks in the face of interference, the OWL agents only see each task 20% of the time but they can solve the tasks for a larger proportion of the time showing that the EWC regularization is mitigating forgetting in the feature extractor.

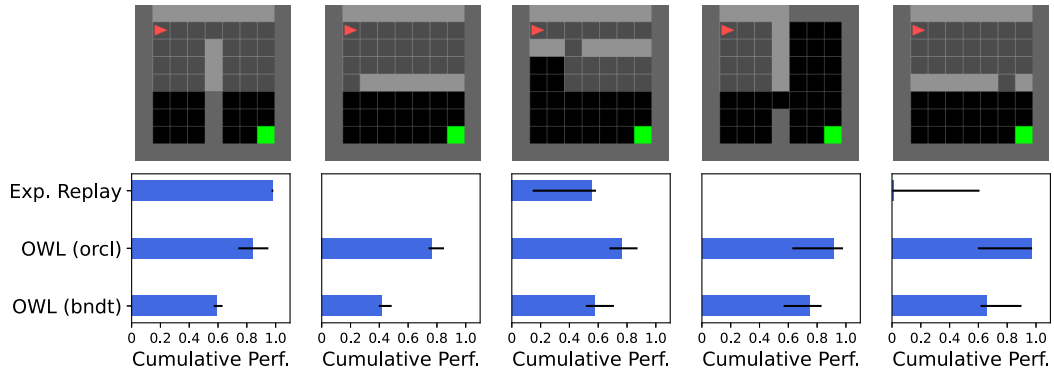


Figure 10.23: Cumulative level completions for each different level for different CL strategies. Experience replay suffers from interference and cannot complete some tasks altogether.

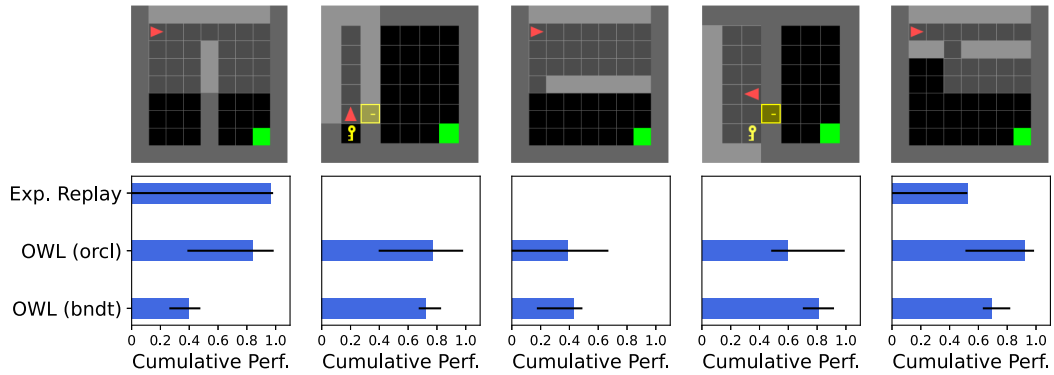


Figure 10.24: Cumulative level completions for each different level for different CL strategies. Experience replay suffers from interference and cannot complete some tasks altogether.

10.3.7 Full Rehearsal

Algorithm 5 Full rehearsal: training

Input: Current task $\tau \in \mathcal{T} = \{1, \ldots, M\}$, experience replay buffers per task $\{\mathcal{D}[\tau], \forall \tau \in \mathcal{T}\}$.
Initialize: $\mathcal{D}[\tau] =, \forall \tau \in \mathcal{T}$
Sample $r \sim \mathrm{U}[0, 1]$.
if $r > 0.25$ **then**
    Use head $\tilde{\tau} = \tau$.
else
    Use head $\tilde{\tau} \sim \mathrm{U}\left[\mathcal{T} \setminus \{\tau\}\right]$.
end if
Sample experiences from replay buffer $\mathcal{D}[\tilde{\tau}]$.
Optimize Q-function with head $\tilde{\tau}$.

We use a multi-head network with separate replay buffers as an upper bound

to OWL, named Full Rehearsal (FR). FR does not scale to many tasks but can achieve good performance as it essentially mimics performing DQN on all tasks simultaneously while learning continuously. While training on the current task τ we need to sample a different task every so often to ensure that past tasks are not forgotten by the multi-headed DQN agent. Thus we devise the following simple algorithm which randomly selects a past task with probability 0.25 to train on past tasks. This is summarized in Algorithm 5. This achieves close to optimal performance, see Table 6.1.

10.3.8 Bandit algorithm visualization

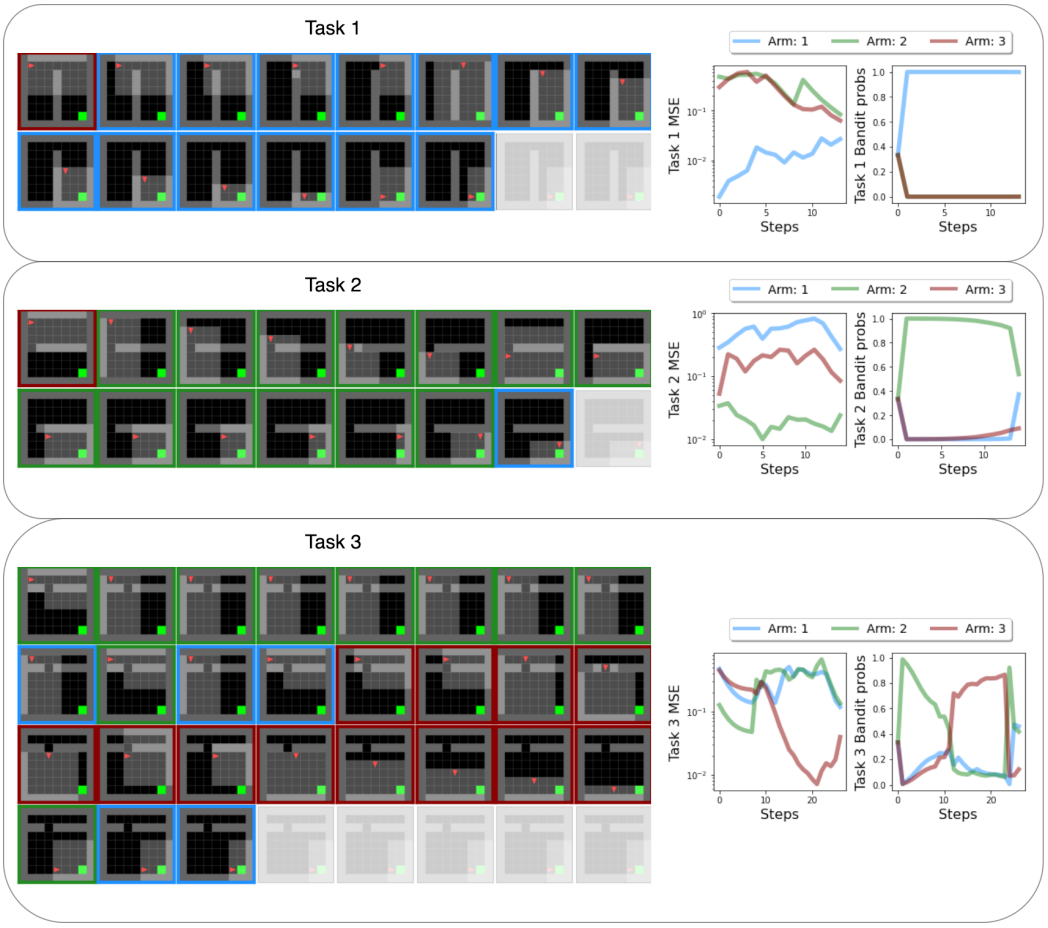


Figure 10.25: Visualization of the OWL agent using the bandit algorithm to decide which policy to use at test time (best viewed in color). **Left**, visualization of environment and agent. The agent proceeds from left to right over a rollout. The bandit explores the different policies, to begin with but then settles on the correct policy to exploit before finding the goal. **Right**, TD error (referred to as MSE in the plots) for each policy - which is fed back as the loss to the bandit algorithm and the bandit algorithm arm (policy) probabilities over the course of the rollout. For all test environments, the bandit algorithm is able to find the correct policy and exploit it to find the goal. The color around the image indicates which policy/bandit arm that has been pulled. We find that close to the goal all policies give a good estimate of their Q-value and hence similar TD errors and hence similar bandit arm probabilities. This behaviour is expected as all policies should be able to get to the goal when close by. Hence at the end of the rollout, the bandit algorithm can choose a different policy head to the one that has been trained for the task under evaluation, and navigate to the goal successfully.

10.4 The Effectiveness of World Models for Continual Reinforcement

10.4.1 Continual Reinforcement Learning Metrics

We describe in detail how to calculate the continual reinforcement learning metrics used extensively throughout this manuscript.

10.4.1.1 Average Performance

This measures how well a CRL method performs on all tasks at the end of the task sequence. The task performance is $p_\tau(t) = [-1, 1]$ for all $\tau < T$. Since we have a reward of $+1$ for completing the task and -1 for being killed by a monster or falling into lava. If each task is seen for N environment steps and we have T tasks and the τ-th task is seen over the interval of steps $[(\tau - 1) \times N, \tau \times N]$. The average final performance metric for our continual learning agent is defined as:

$$p(t_f) = \frac{1}{T} \sum_{\tau=1}^{T} p_\tau(t_f), \tag{10.4.1.1}$$

where $t_f = N \times T$ is the final timestep.

10.4.1.2 Forgetting

The average forgetting is the performance difference after interacting with a task versus the performance at the end of the final task. The average forgetting across all tasks is defined as:

$$F = \frac{1}{T} \sum_{\tau=1}^{T} F_\tau \quad \text{where} \quad F_\tau = p_\tau(\tau \times N) - p_\tau(t_f). \tag{10.4.1.2}$$

The forgetting of the final T-th task is $F_T = 0$. If a CRL agent has better performance at the end of the task sequence compared to after τ-th task at time-step $\tau \times N$ then $F_\tau < 0$.

10.4.1.3 Forward Transfer

The forward transfer is the difference in task performance during continual learning compared to the single task performance. The forward transfer is defined:

$$FT = \frac{1}{T}\sum_{\tau=1}^{T} FT_\tau \quad \text{where} \quad FT_\tau = \frac{\text{AUC}_\tau - \text{AUC}_{\text{ref}_\tau}}{1 - \text{AUC}_\tau}, \tag{10.4.1.3}$$

where AUC denotes the area under the curve and is defined as:

$$\text{AUC}_\tau = \frac{1}{N}\int_{(\tau-1)\times N}^{\tau\times N} p_\tau(t)dt \quad \text{and} \quad \text{AUC}_{\text{ref}_\tau} = \frac{1}{N}\int_{0}^{N} p_{\text{ref}_\tau}(t)dt. \tag{10.4.1.4}$$

$FT_\tau > 0$ means that the CRL agent achieves better performance on task τ during continual learning versus in isolation. So this metric measures how well a CRL agent transfers knowledge from previous tasks when learning a new task.

10.4.2 Single Task experiments

To assess the forward transfer of DreamerV2 for CRL we need the performance of each task as a reference Eq. (10.4.1.3). Single task learning curves for Minigrid are shown in Fig. 10.26 and single task learning curves for all Minihack tasks are shown in Fig. 10.27.

10.4.3 Further Experiments

We introduce further experiments which are referenced in the main paper. In Section 10.4.3.1 we show learning curves for each individual task for 4 task Minihack. In Section 10.4.3.2 we show the results of the regularization-based task-aware world model baseline. In Section 10.4.3.3 we show learning curves for each individual task from 8 task Minihack experimental setup. In Section 10.4.3.4 we explore various design choices required for DreamerV2 with Plan2Explore to get the best performance for CRL. In Section 10.4.3.5 we explore how increasing the size of the experience replay buffer size affects performance in the Minihack CRL benchmark. These experiments are on 8 tasks of Minihack. The 8 tasks are a subset of those introduced in the CORA Minihack suite (Powers et al., 2021) and are a superset of

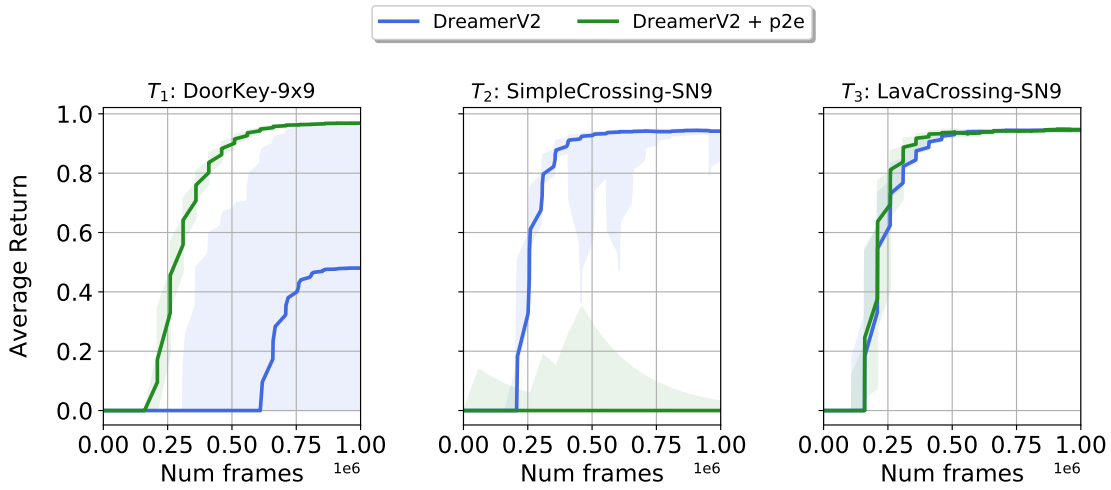


Figure 10.26: Single task performance of individual tasks from the Minigrid CRL benchmark. All curves are a median and inter-quartile range over 5 seeds.

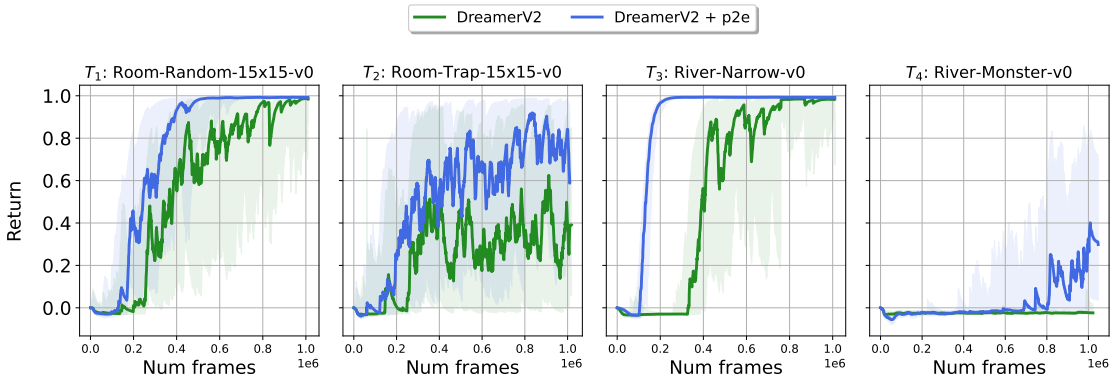


Figure 10.27: Single task performance of individual tasks from the Minihack CRL benchmark. All curves are a median and inter-quartile range over 10 seeds.

the 4 tasks in the main paper. The 8 tasks, in order, are: `Room-Random-15x15-v0`, `Room-Monster-15x15-v0`, `Room-Trap-15x15-v0`, `Room-Ultimate-15x15-v0`, `River-Narrow-v0`, `River-v0`, `River-Monster-v0`, and `HideNSeek-v0`.

10.4.3.1 4 task Minihack

Fig. 10.28 shows the success rates of the baseline methods and DreamerV2 variants for each of the four tasks in the Minihack experiment Section 7.6.2. Continual Dreamer successfully balances retaining knowledge from previous tasks and learning new ones. On the other hand, the CLEAR baseline can only learn the first two tasks with a significant delay, while Impala struggles on all tasks. DreamerV2

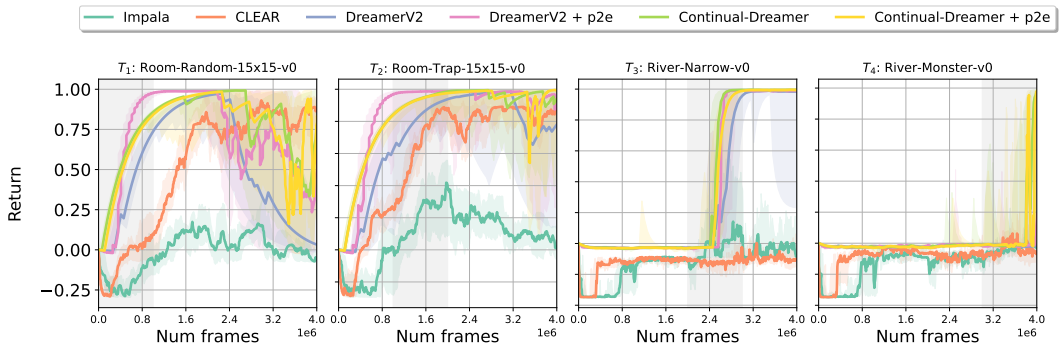


Figure 10.28: Detailed, per task, comparison of baselines and DreamerV2 variants that are presented in Fig. 7.3 with average return. All curves are a median and inter-quartile range over 10 seeds.

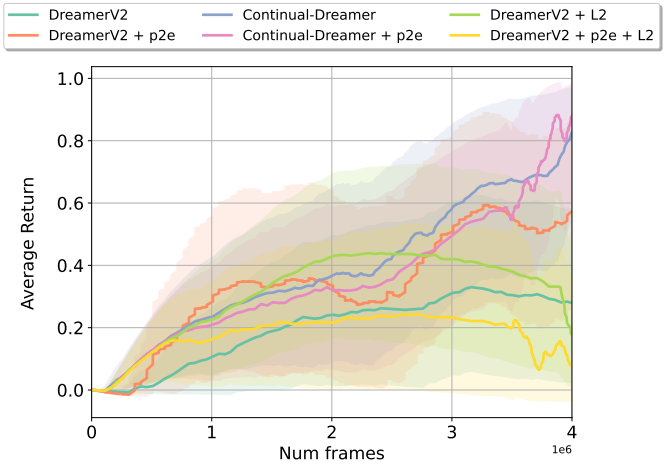


Figure 10.29: Comparison of the average return over all tasks between task-aware and task-agnostic approaches based on DreamerV2, on 4 Minihack tasks. All curves are IQM from `rliable` package across 10 seeds and 1000 bootstrap samples.

and DreamerV2 + Plan2Explore perform poorly on the last task and exhibit more forgetting than Continual-Dreamer.

10.4.3.2 Task-aware world-model baseline

Differences for task-agnostic and task-aware variants of DreamerV2 are shown in Fig. 10.29. The task-aware variant based on L^2 regularization underperforms in comparison to task-agnostic methods. The plausible explanation is that L^2 regularization makes the method too rigid to efficiently learn the two last tasks because of the excessive influence of the first tasks. We optimize the size of the L^2 scaling from the set $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 100\}$.

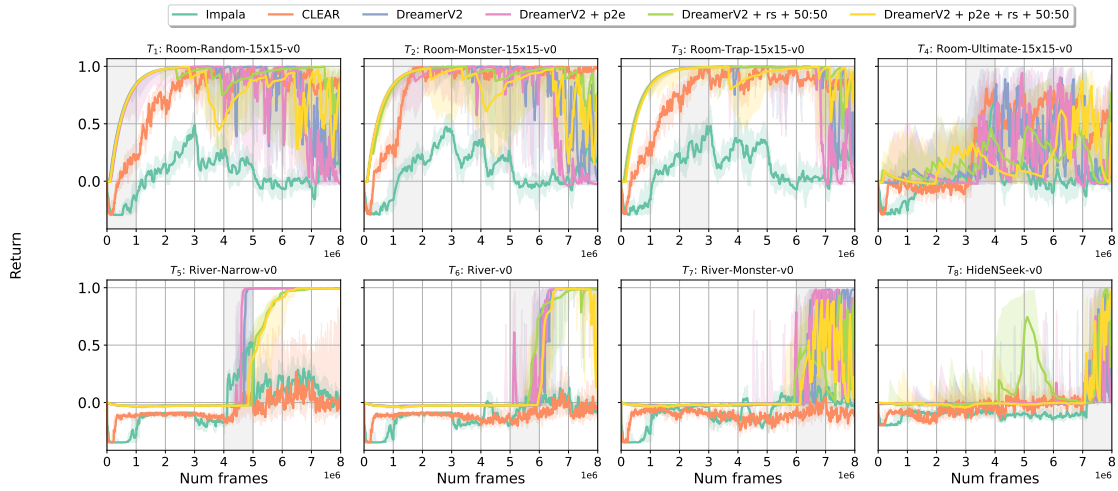


Figure 10.30: Learning curves for 8 Minihack tasks for DreamerV2 and variants and Impala and CLEAR baselines. All curves are a median and interquartile range of 10 seeds.

10.4.3.3 Scaling to 8 tasks

The results for 8 Minihack tasks are shown in Fig. 10.30. DreamerV2 variants display strong knowledge retention and effective learning on almost every task compared to baseline methods. CLEAR method struggles with the last 4 tasks, whereas Impala's performance is poor on every task. DreamerV2 and its variants display forgetting of the initial tasks, for which CLEAR retains the highest performance. However, CLEAR, in contrast to DreamerV2 variants, struggles to learn novel tasks.

10.4.3.4 DreamerV2 Ablation Experiments

We explore various design choices that come from the implementations of DreamerV2 (Hafner et al., 2020) and Plan2Explore (Sekar et al., 2020).

1. The use of Plan2Explore as an intrinsic reward.

2. World model learning by reconstructing the observations $\hat{o}_t$ only and not the observations, rewards, and discounts altogether.

3. The use of the exploration policy to evaluate the performance on all current and past tasks rather than having a separate exploration and evaluation policy.

Plan2Explore	$\hat{o}$ reconstruction only	$\pi_{exp} = \pi_{eval}$	Avg. Performance (↑)	Avg. Forgetting (↓)	Avg. Forward Transfer (↑)
-	-	-	0.09 ± 0.07	0.37 ± 0.07	0.56 ± 0.86
✔	-	-	0.28 ± 0.13	0.13 ± 0.08	0.11 ± 0.15
✔	✔	-	0.39 ± 0.13	0.19 ± 0.16	0.87 ± 0.95
✔	✔	✔	0.38 ± 0.03	0.22 ± 0.05	0.76 ± 0.25

Table 10.10: CRL metrics for different design decisions on DreamerV2 for the Minihack CRL benchmark of 8 tasks. All metrics are an average and standard deviation over 5 seeds. ↑ indicates better performance with higher numbers, and ↓ the opposite.

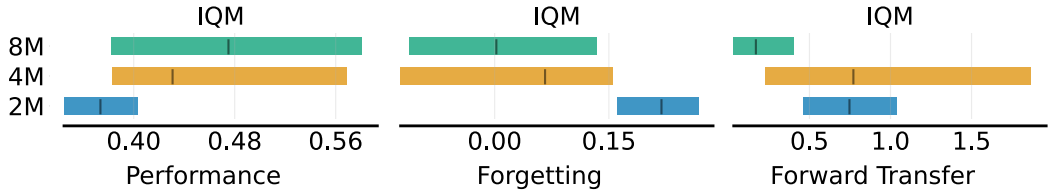


Figure 10.31: CRL metric package for DreamerV2 + Plan2Explore for the Minihack benchmark of 8 tasks versus the experience replay buffer size of the world model for DreamerV2 + Plan2Explore. All metrics are an interquartile mean (IQM) over 5 seeds with 1000 bootstrap samples from the `rliable`.

The results are shown in Table 10.10. We decided to pick the model in the final line in Table 10.10 to report the results in the main paper as they produce good results on Minihack with a relatively small standard deviation.

10.4.3.5 Stability versus Plasticity: Increasing the Size of the Replay Buffer

By increasing the replay buffer size for world model learning for DreamerV2 + Plan2Explore we see that forgetting and average performance increase. However, the forward transfer simultaneously decreases, Fig. 10.31. Additionally, by inspecting the learning curves we notice that the harder exploration tasks are not learned as the replay buffer size increases. This is an instance of the stability-plasticity trade-off in continual learning the larger buffer size enables better remembering but simultaneously prevents quickly learning the new tasks.

10.5 Continual-wav2vec2: an Application of Continual Learning for Self-Supervised Automatic Speech Recognition

10.5.1 Datasets

10.5.1.1 English: LibriSpeech

We use the LibriSpeech dataset which is an English ASR dataset derived from audiobooks and contains 1000 hours of speech (Panayotov et al., 2015b). The corpus is divided into "clean" and "other" partitions, lower-WER speakers are placed into the clean dataset and the rest into the other dataset. We validate our models on the `dev-clean` dataset, the size of the datasets are described in Table 10.11.

Dataset	Hours
`dev-clean`	5.4
`dev-other`	5.3
`test-clean`	5.4
`test-other`	5.1
`train-clean-100`	100.6
`train-clean-360`	363.6
`train-other-500`	596.7

Table 10.11: LibriSpeech datasets and their lengths.

10.5.1.2 French

We used the following training datasets for performing self-supervision of French audio, in Table 10.12. We use 1000 hours of raw audio to perform self-supervision with 5% for validation. For finetuning we use 10 hours from the `common_voice` training set and we validate with the 5 hours from the `common_voice` dev set.

[1] `https://librivox.org/api/info`
[2] `https://github.com/kyubyong/css10`
[3] `http://www.voxforge.org/fr/Downloads`
[4] `https://librivox.org/api/info`
[5] `https://datashare.ed.ac.uk/handle/10283/2353`
[6] `https://www.openslr.org/57/`

Dataset	train (hours)	dev (hours)	test (hours)
`common_voice` (Ardila et al., 2019b)	962.4	23.8	25.0
`fr FR HQ` [2]	142.8	-	-
`french single speaker` [3]	19.1	-	-
`Voxforge` [4]	37.2	-	-
`LibriFrench` [5]	45.6	-	-
`Siwis` [6]	10.7	-	-
`African Accented` [7]	0	1.0	0.3

Table 10.12: French datasets used for pretraining and finetuning.

10.5.1.3 Spanish

We used the following training datasets for performing self-supervision of Spanish audio in Table 10.13. We use 839.89 hours of raw audio to perform self-supervision with 5% for validation. For finetuning we use 10 hours from the `common_voice` training set and we validate with the 5 hours from the `common_voice` dev set, similarly to French.

Dataset	train (hours)	dev (hours)	test (hours)
`common_voice` (Ardila et al., 2019b)	825.19	24.8	25.5
`tedX` (Hernandez-Mena, 2019)	14.7	-	-

Table 10.13: Spanish datasets used for pretraining and finetuning.

10.5.2 Implementations

We use the `fairseq` framework (Ott et al., 2019) to implement our continual-wav2vec2 models. The pre-training and finetuning proceeds similarly to (Baevski et al., 2020) apart from the model changes in MH wav2vec2.0 and cwav2vec2.0. We provide a recap for completeness.

Feature extractor. The feature extractor $f(\cdot)$ contains several temporal convolutional layers with layer norms (Ba et al., 2016) and GeLU activation functions (Hendrycks and Gimpel, 2016).

Quantization module. The Quantization discretizes the output of the feature extractor $\mathcal{Z} \rightarrow \mathcal{Q}$. Quantization module is comprised of G codebooks, with V entries $\boldsymbol{e} \in \mathbb{R}^{V \times d/G}$. One entry from each codebook is chosen and concatenated: $[\boldsymbol{e}_1, \ldots \boldsymbol{e}_G] \in \mathbb{R}^{V \times d}$. To construct the quantizer codebook, the outputs of the feature

encoder $\boldsymbol{z} = f(x)$ are mapped to logits $\boldsymbol{l} \in \mathbb{R}^{G \times V}$. In the wav2vec2.0 BASE model used, $G = 2$ the outputs from the encoder $\boldsymbol{z}_t \in \mathbb{R}^{512}$ and each codebook is of dimensionality $V = 320$. Then a linear transformation is applied to the vectors $\boldsymbol{e}_i, \ldots \boldsymbol{e}_G$ such that $\mathbb{R}^d \to \mathbb{R}^f$ where $d = 640$ and $f = 256$.

Objective. The contrastive loss $\mathcal{L}_m$ identifies the true quantized latent speech representation from masked time steps as:

$$\mathcal{L}_m = -\log \frac{\exp(\text{sim}(\boldsymbol{c}_t, \boldsymbol{q}_t)/\kappa)}{\sum_{\tilde{\boldsymbol{q}} \sim Q_t} \exp(\text{sim}(\boldsymbol{c}_t, \tilde{\boldsymbol{q}})/\kappa)}, \tag{10.5.2.1}$$

where $\text{sim}(\boldsymbol{a}, \boldsymbol{b})$ is the cosine similarity, defined as $\text{sim}(\boldsymbol{a}, \boldsymbol{b}) = -\boldsymbol{a}^\top \boldsymbol{b}/||\boldsymbol{a}||\,||\boldsymbol{b}||$. A diversity regularization is also added to encourage the model to use the entire codebook of V entries in each of the G codebooks by maximizing the entropy of the averaged softmax distribution l over the codebooks:

$$\mathcal{L}_d = \frac{1}{GV} \sum_{g=1}^{G} -H(\bar{p}_g) = \frac{1}{GV} \sum_{g=1}^{G} \sum_{v=1}^{V} \bar{p}_{gv} \log \bar{p}_{gv}. \tag{10.5.2.2}$$

Thus the objective is $\mathcal{L} = \mathcal{L}_m + \alpha \mathcal{L}_d$ where α is a scalar that controls the strength of the codebook diversity regularization.

Masking. The training objective is to identify the correct quantized latent audio representation by comparing to the masked prediction. The feature encoder outputs are masked before being fed into the context network. To mask, we randomly sample a with probability $p = 0.065$ of all time steps to be starting indices and then mask the subsequent $M = 10$ consecutive time steps from every sampled index; spans can overlap. This masking procedure is identical to that from wav2vec2.0 (Baevski et al., 2020).

Pre-training. We performed pre-training on 8 Tesla V100 GPUs. We simulated 64 GPUs which are used by wav2vec2.0, by using 8 gradient accumulation steps before updating gradients. We follow the pre-training procedure of wav2vec2.0 (Baevski et al., 2020). For mask starting timesteps with $p = 0.065$ and mask the next $M = 10$ timesteps. The feature extractor $f(\cdot)$ has seven blocks with 512 channels with strides $(5, 2, 2, 2, 2, 2, 2)$ and kernel widths $(10, 3, 3, 3, 3, 2, 2)$. A 1-d convolutional layer models relative positional embeddings and has size 128 and

16 groups (Mohamed et al., 2019a). We use the BASE model configuration fro the encoder $g(\cdot)$ (Baevski et al., 2020): 12 transformer blocks with dimension 768, inner dimension 3072 (FC$\times$ 2 in Fig. 8.3) and 8 attention heads. Audio crops do not exceed 1.4m samples per GPU, the batch size is 1.6 hours. We use Adam and warm up the learning rate for the first 8% of updates to a maximum of 5×10^{-4}, we use the same learning rate scheduling procedure for all our models and the learning rate quoted is always the maximum. We train for 400k steps, similarly to the wav2vec2.0 BASE model. The penalty for the diversity loss is $\alpha = 0.1$. The Gumbel-Softmax temperature τ is annealed from a maximum of 2 to a minimum of 0.5 by a factor of 0.999995 every update (Jang et al., 2016). The temperature of the contrastive loss $\kappa = 0.1$ and use $K = 100$ distractors. An L2 penalty is added to the activations of the final layer of the feature encoder with a size of 10.

Fine-tuning. After pretraining on each unlabelled task dataset we finetune the learned representations on 10 hours of labeled data following the same procedure as in (Baevski et al., 2020). A new classification layer is placed on the final layer of the network, the feature extractor is frozen during finetuning. The network is optimized with Adam and the CTC loss (Graves et al., 2006) using a tri-stage learning rate: the learning rate is warmed up for the first 10% of updates, then held constant at a maximum learning for the next 40% of steps and then linearly decayed. We do not use a Language model for fine-tuning.

10.5.2.1 Warm-starting wav2vec2.0

If we use the default parameters for wav2vec2.0 we find that the model yields unstable results when we simply warm-start the network when learning a new audio representation for a new language for a second task. The learning rate is 5×10^{-4}. We can lower the learning rate for the new task though, to prevent instability. By lowering the learning rate for a new task we are essentially balancing how much we can learn a new task versus remaining stable with respect to forgetting the previous task. What we find is that by simply lowering the learning rate of wav2vec2.0 when learning a new task we can restrict forgetting but also enable

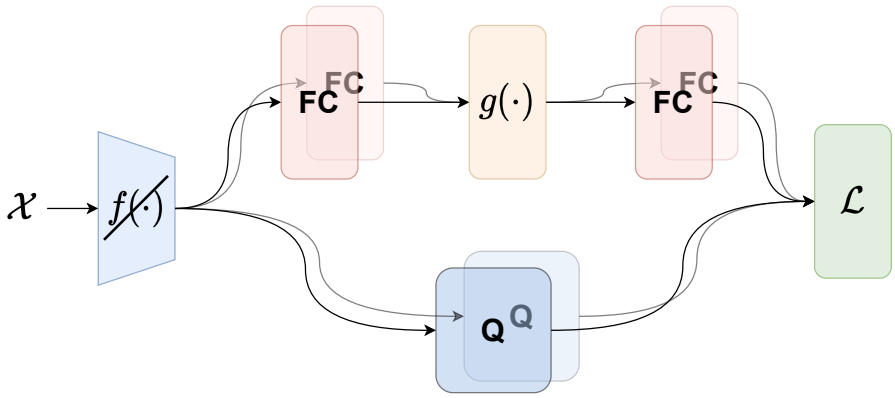


Figure 10.32: Diagram of cwav2vec2.0 v1 architecture, the strike-through means that the module it frozen and no gradient is taken when training with SGD.

learning a new audio task, see Fig. 8.4 and Fig. 8.5. However, by simply warm-starting the model we are not going to get the performance benefits for a second or third task that we would get for cwav2vec2.0.

10.5.2.2 Multi-headed wav2vec2.0

We propose a very simple baseline which is to use multi-head architectures for wav2vec2.0. Multi-head architectures are commonly used in CL to retain task-specific knowledge by learning task-specific mappings from hidden to output layers (Li and Hoiem, 2017a; Nguyen et al., 2017). We are using heads to learn task-specific mappings from the feature extractor to the encoder and from the encoder to the objective function. After having trained on the first task, the feature extractor $f(\cdot)$ has learned how to extract features from the audio, so we can freeze the feature extractor and add a language-specific head before into the MHSA encoder $g(\cdot)$. We also tune the learning rate by pretraining for 50k steps from the set $\{1 \times 1 \times 10^{-5}, 5 \times 10^{-5}, 1 \times 10^{-4}, 5 \times 10^{-4}\}$ and found 5×10^{-5} to work well.

Additionally, we enable learning new language representation in the quantization module by adding a language-specific quantizer. This enables learning a new language without interference from a previous language. This is in contrast to previous multi-task wav2vec2.0 models which share a quantizer for all languages (Conneau et al., 2020). Instead of warm-starting the quantizer from the previous task, we train a new quantizer from randomly initialized weights. In the CTC finetuning

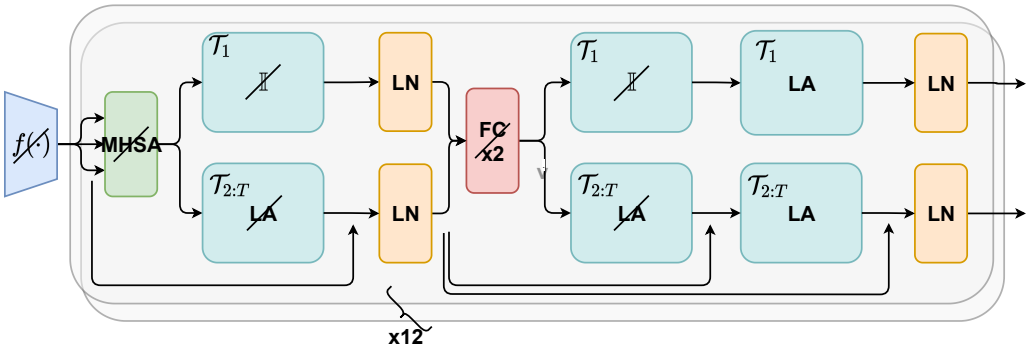


Figure 10.33: Diagram of cwav2vec2.0 finetuning architecture: the pretraining adapters are frozen and the new finetuning adapters are placed on top of the pretraining adapters and trained together with the layer norms during the fine-tuning stage.

step we train the entire encoder and an additional fully connected projection layer mapping into the C classes representing the vocabulary of the ASR task.

Finetuning proceeds in the same way as wav2vec2.0.

10.5.2.3 Multi-headed wav2vec2.0 and L2 regularization

To combat forgetting regularization methods like EWC (Kirkpatrick et al., 2017) and SI (Zenke et al., 2017a) add a regularization penalty around the previous task's optimal parameters with different weightings for each parameter. We do a simple regularization in a similar vein: add a regularization to each parameter in the MHSA layers. We add the regularization for task $\mathcal{T}_t$: $||\boldsymbol{\theta} - \boldsymbol{\theta}^*_{t-1}||_2$, where $\boldsymbol{\theta}^*_{t-1}$ are the optimal parameters for the previous task and η is a hyperparameter which controls the regularization strength. We experiment with $\eta \in \{0.1, 1, 10, 100\}$ and find that $\eta = 0.1$ works the best after training for $50k$ steps of pretraining, we also experiment with regularization of the top $\{6, 12\}$ layers of the MHSA encoder $g(\cdot)$ and find little difference in performance after $50k$ steps of pretraining, hence we regularize the top 6 layers to make the optimization simpler and ensure we are not constraining learning the new language representation too much.

10.5.2.4 Continual-wav2vec2.0

We place LAs in each layer of the MHSA encoder $g(\cdot)$, and the model dimension is 768. For French, every adapter has a bottleneck dimension 512, which we tune by

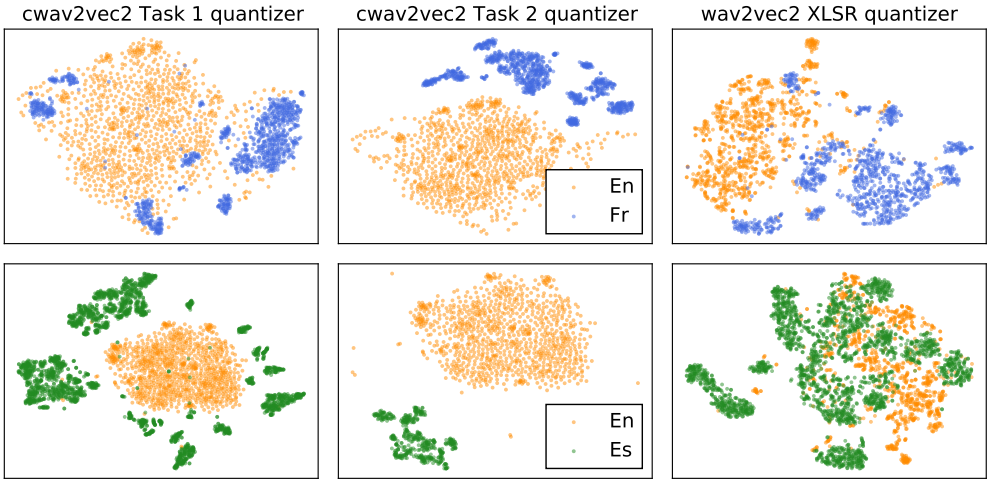


Figure 10.34: t-SNE embeddings (Van der Maaten and Hinton, 2008). We compare the cwav2vec2 embeddings from different languages from different quantizers; after $\mathcal{T}_1$ learning only English (column 1), then after learning a second task $\mathcal{T}_2$ French (row 1) or Spanish (row 2). The third columns are the multilingual embeddings from XLSR (Conneau et al., 2020).

pretraining for only $50k$ from the set $\{256, 384, 512, 768, 1024\}$. We use a learning rate of 1×10^{-4} which is tuned from by pretraining for only 50k steps from the set $\{1 \times 1 \times 10^{-5}, 5 \times 10^{-5}, 1 \times 10^{-4}, 5 \times 10^{-4}\}$. For Spanish, every adapter has a bottleneck dimension 640, which we tune by pretraining for only $50k$ from the set $\{256, 384, 512, 640, 768, 1024\}$. We use a learning rate of 2×10^{-4} which is tuned from by pretraining for only 50k steps from the set $\{1 \times 1 \times 10^{-5}, 5 \times 10^{-5}, 1 \times 10^{-4}, 2 \times 10^{-4}, 5 \times 10^{-4}\}$. Performance is very dependent on adapter size and learning rate, and further tuning could lead to better performance.

Finetuning. In wav2vec2 the encoder and a new classification layer are finetuned. Alternatively, for cwav2vec2.0 we place LAs on top of the pretraining adapters and train only the newly placed adapters and the corresponding layers norms Fig. 10.33. The finetuning LA bottleneck dimension is 256 and we use a tri-stage learning rate where the maximum learning rate is 0.0008.

10.5.3 Qualitative analysis of cwav2vec2 versus wav2vec2

In Fig. 10.34 we show embeddings from the quantizer from cwav2vec2.0 in columns 1 and 2 and compare to the multi-task/multi-lingual representation from the XLSR

model (Conneau et al., 2020), column 3. We visualize the features from the quantizers: we are able to get an understanding of the language representation from $g(\cdot)$ as both are learned jointly. We notice from Fig. 10.34 that when training on $\mathcal{T}_1$ which is English only (column 1) we can achieve a good 0-shot embedding for French and Spanish, reflected in the fact that we can get a WER $< 100\%$ when we perform finetuning in these languages without any pretraining. More importantly, we show in the second column that there is a separation between languages that have been learned by cwav2vec2.0. cwav2vec2.0 is task aware and learns to represent a new language as a different task thus the separation between language embeddings in column 2, note how the representation for English in $\mathcal{T}_2$ has been preserved from column 1. This is in contrast to the final column which has been trained multi-lingually on many different languages and is not task-aware, as a result, the language embeddings overlap.

We can see from Fig. 10.34 how the separability of the embeddings is produced by constraining the pretraining process for $\mathcal{T}_2$ to focus specifically on learning a new language representation, distinct from $\mathcal{T}_1$.