Deep Transfer Learning with Bayesian Inference

Andrew Gambardella

St Catherine's College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2021

Declaration

This thesis is submitted to the Department of Engineering Science, University of Oxford, in fulfilment of the requirements for the degree of Doctor of Philosophy. This thesis is entirely my own work, and except where otherwise stated, describes my own research.

Andrew Gambardella, St Catherine's College

Acknowledgements

Doing a DPhil in Oxford was an incredible opportunity for me, and I cannot be grateful enough for having the privilege to study here. But I would have never been able to complete this journey alone. I would like to take this space to thank the people who raised me up, and helped me to become the person who I am today.

First and foremost I would like to thank my advisor, the great Phil Torr. Phil's magical talent for always knowing the right thing to say at the right time motivated me when I had lost all hope, and made us laugh and enjoy life even when it seemed that the entire world was collapsing around us. It is often said that an advisor can make or break one's DPhil, and I'm glad to say that Phil made mine.

I should also give special thanks to the two postdocs in Phil's group who acted as second advisors, N. Siddharth and Atılım Güneş Baydin. They had always been there to help me through the experiments and details of my papers, making sure I hit my deadlines and had something to show when Phil came knocking. I certainly couldn't have finished my DPhil without their support.

I would like to thank my other collaborators: Wendelin Böhmer, Ara Darzi, Puneet Dokania, Ana Belen Espinosa-Gonzalez, Maximilian Igl, Naeemullah Khan, Nantas Nardelli, Christian Schroeder de Witt, Bogdan State, Leo Tsourides, Shimon Whiteson, and Rob Zinkov, as well as all of my other labmates who helped me through my DPhil.

I would like to thank my undergrad research advisor at UC Berkeley, Satish Rao, and coworkers at SoftBank and Cogent Labs, who encouraged and developed my early interest in research.

I would like to give thanks to the brave men and women of the NHS, who have worked tirelessly keeping the UK safe during the COVID-19 pandemic.

And lastly, I would like to thank my family. I would like to thank my father, stepmother, and brother, Gary, Bobbi, and Glenn Gambardella, and my mother, Maureen Hunter, for helping me grow into the person I am today.

Abstract

Since the deep learning revolution, a general trend in machine learning literature has been that large, deep models will consistently outperform small, shallow models. This trend, however, comes with the drawback of ever-increasing compute requirements, with many recent state-of-the-art results requiring resources well out of reach of all but the top industry labs. Such issues raise very real concerns with regards to the democratisation of machine learning research, and left unaddressed could ultimately lead to more power and wealth being concentrated in the institutions which are able to invest extremely large sums of money into their AI research programs today.

Transfer learning techniques are a potential solution to these issues, allowing large, general models to be trained once, and then reused in a variety of situations with minimal computation required to adapt them. This work explores novel algorithms and applications of transfer learning in domains as diverse as hierarchical reinforcement learning, generative modeling, and computational social science. Within the hierarchical reinforcement learning domain, we present an algorithm that allows for transfer between options (i.e., temporally abstracted actions) over separate but similar tasks. In the generative modeling domain, we present an algorithm for reusing existing invertible generative models on new data without incurring any extra training cost. Lastly, in the computational social science domain, we show that knowledge can be transferred from human-designed models in order to detect malicious activity targeting a ranking algorithm.

The common thread between all of the algorithms presented in this thesis is that they are inherently Bayesian. We argue that the Bayesian paradigm naturally lends itself to transfer learning applications, in that Bayesian priors can serve as adaptable, general models which can be transformed into task-specific posteriors through the process of inference.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

4.014 The gramophone record, the musical thought, the score, the waves of sound, all stand to one another in that pictorial internal relation, which holds between language and the world. To all of them the logical structure is common.
(Like the two youths, their two horses and their lilies in the story. They are all in a certain sense one.)

4.0141 In the fact that there is a general rule by which the musician is able to read the symphony out of the score, and that there is a rule by which one could reconstruct the symphony from the line on a gramophone record and from this again—by means of the first rule—construct the score, herein lies the internal similarity between these things which at first sight seem to be entirely different. And the rule is the law of projection which projects the symphony into the language of the musical score. It is the rule of translation of this language into the language of the gramophone record.

Tractatus Logico-Philosophicus
Ludwig Josef Johann Wittgenstein [222]

1.1 The Advent and Pitfalls of Deep Learning

In the past decade, deep learning [78, 122] has become a ubiquitous tool throughout the entire machine learning community, helping to achieve state of the art results on a myriad of tasks in sub-fields as diverse as computer vision [118], natural language processing, unsupervised learning [79], and reinforcement learning [142].

More recently, researchers have become more aware of just how powerful

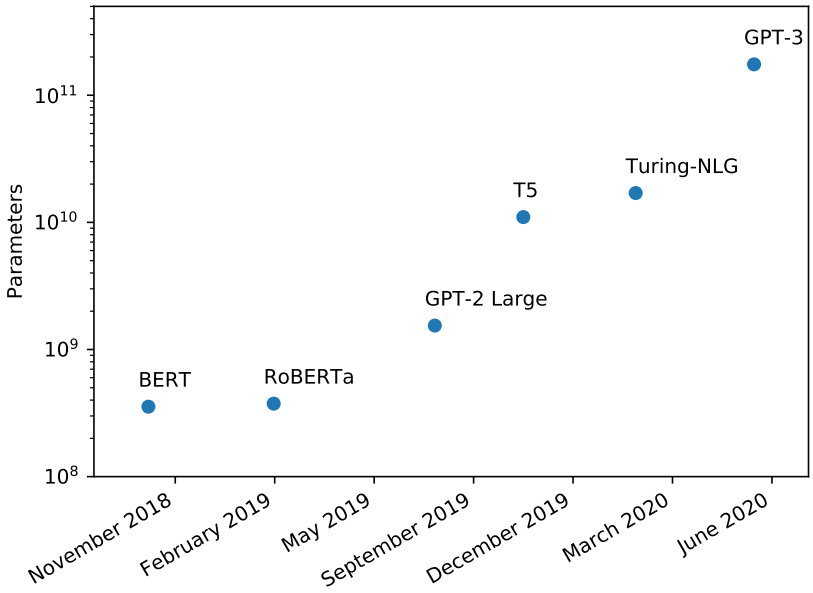


Figure 1.1: Transformer-based architectures [207] have led to remarkable advances in the state of the art in natural language processing. As time goes on, researchers are discovering the benefits of training extremely large models, leading to an explosive, near-exponential growth in the number of parameters in the best-performing models. These highly-parameterised models typically require prohibitively large amounts compute to train as well, necessitating the use of transfer learning techniques to obtain practical benefits from them. Parameter counts from [51, 126, 154, 156, 165, 33].

representations learned from data can be, and more importantly, how best to harness them. It is becoming more and more apparent that not just big data, but also big compute, are key to inducing behaviour that can be deemed 'intelligent'. The entire deep learning movement can be said to have been started once researchers realised that Graphics Processing Units (GPUs) could be used to efficiently implement the tensor operations underlying the backpropagation algorithm [158, 167] on large datasets. Moreover, researchers are slowly awakening to the idea that computational power (or, more directly, model size) may be one of the *only* barriers between us, and truly intelligent models [108]. One prominent example in the area of language modeling would be GPT-3 [33] which has achieved performance leaps and bounds

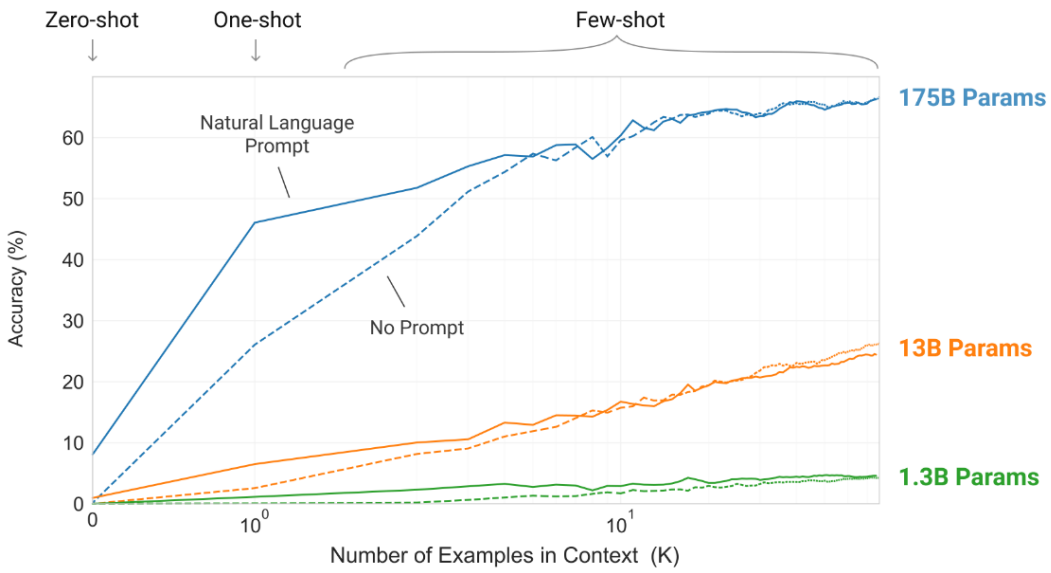


Figure 1.2: GPT-3 leverages transfer learning techniques to gain the advantages of training a large model over a large dataset, while being able to quickly adapt to novel situations for which there may not exist much data. Figure reproduced from [33].

over its predecessor, GPT-2 [154], solely through the use of millions of USD worth of compute, allowing for the training of a 175 billion parameter model compared to the 1.5 billion parameters of GPT-2 [154, 33]. In general, advances in the state of the art in language modeling have been accompanied by a near-exponential growth in the number of parameters used in those models over the last few years, as can be seen in Figure 1.1.

Such progress is encouraging in some respects. After all, we are still using basic algorithms such as backpropagation as we knew them four decades prior. On the other hand, it is clearly infeasible to throw massive amounts of compute at every problem, especially as it is still not clear at what point we will stop getting increases in model performance through further overparameterisation.

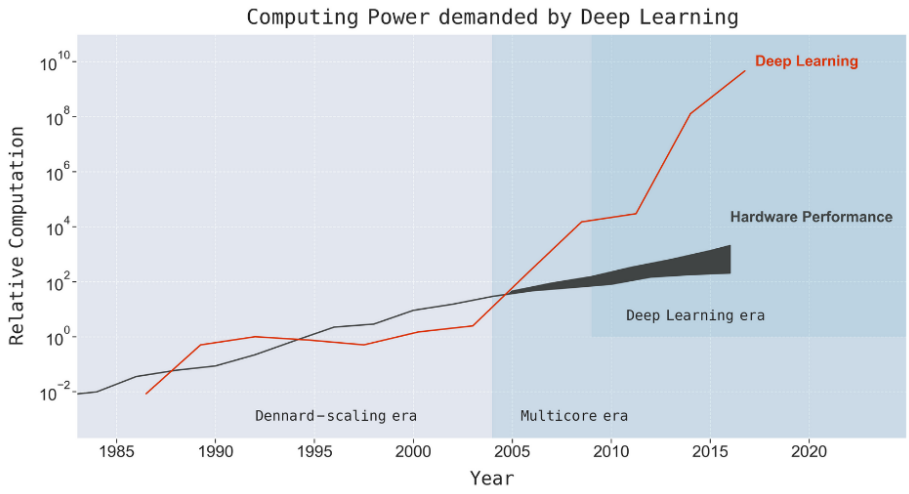


Figure 1.3: Since the deep learning era, computational requirements for machine learning models have strongly outscaled relative improvements in hardware. Figure reproduced from [190].

1.2 Improvise, Adapt, and Overcome

Why would researchers bother training an excessively large autoregressive language model, such as GPT-3? Were GPT-3 simply a language model from which we could unconditionally sample text that is likely to have come from the large and diverse training data, this would be a waste of resources, as it would have very little utility. The utility of GPT-3 comes from its ability to be reused in a multitude of scenarios and situations, and have its output controlled, by being shown few examples of text resembling a subset of the training data, and sampling realistic text from that subset, as shown in Figure 1.2. This paradigm of using previously learned knowledge in order to adapt to a new, but similar situation, goes by many names depending on the specific details: few-shot learning [62, 21, 33], meta-learning [9, 65], and transfer learning [198]. For the purpose of this thesis, we will adopt the terminology of 'transfer learning' in most situations, but these terms are in many ways very similar, as we will explain in Section 1.3.

As our models become more and more compute intensive, it is clear that we

cannot hope for new advances in hardware that allow for us to re-train large models for every conceivable situation. Indeed, we are fast approaching the theoretical limits of Moore's Law, in which heat and quantum mechanical effects impose unavoidable restrictions on chip size [211]. Additionally, even without reaching these limits, computational requirements for state of the art models are growing much faster than improvements in hardware, as depicted in Figure 1.3. In a world in which large models vastly outperform smaller models but we cannot hope for hardware to become progressively more efficient forever, we must rely on amortization of training cost by creating models that can flexibly adapt to new situations with minimal additional cost. Transfer learning, as a paradigm, aims to solve this very issue.

1.3 The Many Faces of Transfer Learning

Transfer learning is a broad term designating many algorithms and scenarios in which one can learn quickly or more robustly, given some sort of previous model or knowledge. This thesis will touch on many of the following scenarios:

1.3.1 Transfer to a Subset of the Training Data

In many examples, such as that of GPT-3 [33] above, one wishes to train a singular, large model once, and then be able to re-use it on any subset of the training data, regardless of how much data we may have for those subsets. While some statistics of the subset may differ from that of the large dataset on which the model was trained (e.g., different vocabulary), more general structural components (e.g., the grammar of the English language) may be held constant throughout any given subset of the training data. In this sense, we can use transfer learning techniques to gain the benefit of big data by using the entire training data to learn these general structural

components thoroughly, even when applying this model to a subset of training data for which we may have very few examples.

1.3.2 Transfer to Similarly Structured Data

Likewise to the previous example, general structural components can be learned across datasets, and then adapted to any similar, specific dataset at hand. Such a scenario frequently occurs in natural language processing applications, for instance, low-resource machine translation [86, 213]. In this specific example, a machine learning algorithm could learn, over many different high-resource linguistic datasets, linguistic universals that hold across all languages [84]. Such universals could be used in order to make assumptions about the grammar of low-resource languages for which we may not have enough data to learn with a small monolingual corpus. Similar experiments have been attempted over differing Markov decision processes with similar dynamics, with limited success [227].

1.3.3 Transfer Between Modalities

There are many situations in which data can be expressed in multiple modalities, and learning about the structure of one modality intrinsically gives one information about the others. Some examples are obvious: the representation of an Atari game in RAM, and the depiction of the game as pixels on a screen, are in bijection, with certain entries in the RAM corresponding to meaningful entities (such as characters or objects) on the screen [17]. Other examples are less obvious: learning about the connection between a piano note represented as spectrogram, and the audio waveform of that note, can induce a relatively simple representation of audio, which could then be transferred to sound produced by other instruments [60]. Even less obvious, the picture theory of language posits that all meaningful utterances in a natural language can be defined or pictured in the real world [222]; thus, learning

facts of the world itself could provide bounds for what meaningful statements could be expressed by a language model or machine translator. More concretely, a language model that transfers information from the real world could give lower likelihood to the grammatically correct sentence, 'colorless green ideas sleep furiously' than a model that does not [36].

1.3.4 Transfer Between Machine Learning Problems

Learning to learn is a paradigm in which part of a machine learning algorithm itself is learned, with the goal being to exploit the similar structure of multiple optimisation problems themselves. In this scenario, we transfer prior knowledge from previous machine learning problems, to a new one. This paradigm is most often referred to as meta-learning [9, 65].

1.3.5 Transfer from a Program

Occasionally, transfer can be accomplished not over multiple learning problems, but from human-created models. As many machine learning-based approaches, particularly deep learning, often struggle to utilise or create highly structured and discrete data structures such as computer programs or neural net architectures themselves, oftentimes humans will wish to convey this structure to a machine learning model, which fills in the easier-to-learn details. Frequently, this structure conveyed by humans will appear in the form of a program, such as partial programs for hierarchical reinforcement learning [136, 137].

While partial programs use machine learning to complete an underspecified, but deterministic program, there are also techniques for inferring the task-specific internals of a program, given its outputs. Given a stochastic simulator (e.g., of high energy physics [15] or epidemiological [82] interactions), generally provided by a human, probabilistic programming techniques [202] can be used to find Bayesian

posteriors over random variables within the simulator. In this sense, the general knowledge with which we are supplied is the simulator itself, and probabilistic programming techniques aim to efficiently adapt the parameters of the random variables in the simulator to fit specific tasks, represented as observed data. Transfer between these 'tasks' can be made efficient by using neural networks to learn proposals for inference algorithms that generalise across many different possible observations [121].

1.4 Approach

There are many approaches to building algorithms that can transfer knowledge from one problem to another; the content of this thesis will centre around approaches which use the machinery of Bayesian inference [31].

Bayesian inference is a natural fit for transfer learning, as the concepts of the prior and posterior naturally map to the ideas of 'general knowledge, independent of task' and 'knowledge conditioned on task-specific data'. The machinery of Bayesian inference is also extremely general, and can be applied anywhere in which one finds themselves with a probability distribution, and data on which we would like to condition it.

Viewing machine learning algorithms under a Bayesian lens confers several advantages over non-Bayesian approaches. Oftentimes, it gives us the ability to know how confident we are when making decisions, where other approaches would not. Bayesian concepts can even be applied to tools as fundamental as the forward pass in a neural network, giving us uncertainty estimates for the network outputs [70].

Bayesian methods can turn single data points, used as conditioning, into a rich posterior from which one can sample. Not only does this provide a principled method

for achieving diverse outputs, it can also confer some algorithmic advantages, such as allowing for estimation of unobserved values through the posterior predictive distribution.

This thesis will demonstrate the above benefits using applications of Bayesian transfer learning and probabilistic prorgramming in sub-fields as diverse as reinforcement learning, generative modeling, and computational social science.

1.5 Thesis Outline

Chapter 2: Literature Review and Background Chapter 2 provides some necessary background and literature review on deep learning, transfer learning, and Bayesian inference in general. We further provide prerequisite information in reinforcement learning, generative modeling, and probabilistic programming that are required to understand the following chapters.

Chapter 3: Transfer Learning in Markov Decision Processes In Chapter 3 we show how to use Bayesian methods to transfer knowledge between options in a hierarchical reinforcement learning setting. We introduce an algorithm which extends Distral [187] to the hierarchical setting, in which we find a shared prior over options that can be conditioned to find options that are specific to certain tasks.

Chapter 4: Transfer Learning in Generative Models Chapter 4 proposes a method for transfer learning within invertible generative models such as normalising flow models [116] that requires no optimisation. We do so by viewing the original, pre-trained flow model as a prior in a Bayesian inference setting, and deriving a posterior which is based on new data. We show that data sampled from this posterior contains a meaningful mixture of components from the original flow model and the new data, and that our found posterior can be used for downstream

tasks.

Chapter 5: Leveraging Human Knowledge with Stochastic Simulators Then in Chapter 5 we show that we can use probabilistic programming techniques with the structure of a human-designed stochastic simulator to efficiently condition random variables on observed simulator outputs, providing a structured and interpretable posterior that is amenable to data. We apply these methods in order to detect and quantify the effects of malicious behaviour in a recommendation algorithm. This problem is extremely under-specified, due to the lack of ground-truth training or test data, and thus traditional methods such as supervised learning cannot be evaluated in a meaningful way. We show that by using probabilistic programming to do Bayesian inference within a simulator, we can identify malicious behaviour within the simulator in a manner that is interpretable and provides uncertainty estimates.

Chapter 6: Conclusions Finally, in Chapter 6 we summarise the contributions of this thesis, and discuss future directions.

1.6 Publications and Contributions

Chapters 3 through 5 are based on the following publications:

Chapter 3

> Based on: Maximilian Igl, Andrew Gambardella, Jinke He, Nantas Nardelli, N. Siddharth, Wendelin Böhmer, Shimon Whiteson. Multitask Soft Option Learning. *Uncertainty in Artificial Intelligence*, 2020.
>
> Contributions: Took part in algorithm development and wrote majority of the experimental framework

Chapter 4

Based on: Andrew Gambardella, Atılım Güneş Baydin, Philip H. S. Torr. Transflow Learning: Repurposing Flow Models Without Retraining. Working Paper, 2019.

Contributions: Devised algorithm, conducted all experiments, and wrote paper

Chapter 5

Based on: Andrew Gambardella, Bogdan State, Naemullah Khan, Leo Tsourides, Philip H. S. Torr, Atılım Güneş Baydin. Detecting and Quantifying Malicious Activity with Simulation-based Inference. Working Paper, 2021.

Contributions: Conducted all experiments and wrote experimental and method sections of paper

Chapter 2

Literature Review and Background

2.0121 It would, so to speak, appear as an accident, when to a thing that could exist alone on its own account, subsequently a state of affairs could be made to fit.
If things can occur in atomic facts, this possibility must already lie in them.
(A logical entity cannot be merely possible. Logic treats of every possibility, and all possibilities are its facts.)
Just as we cannot think of spatial objects at all apart from space, or temporal objects apart from time, so we cannot think of *any* object apart from the possibility of its connexion with other things.
If I can think of an object in the context of an atomic fact, I cannot think of it apart from the *possibility* of this context.

Tractatus Logico-Philosophicus
Ludwig Josef Johann Wittgenstein [222]

This thesis is concerned with augmenting the learning process using prior knowledge, to learn either quickly, more robustly, or with fewer data. This process is known to occur in humans, even over completely separate modalities [200], and as such has been an active area of research for decades, starting in the mid 1990s [192].

Bayesian inference is a natural tool for accomplishing this goal, as it naturally encodes the notion of prior knowledge, and gives rules for updating this prior knowledge in the face of new information. This chapter serves as a review of the preliminaries of and current literature surrounding Bayesian inference for transfer learning as well as transfer learning in general, particularly when applied to

deep reinforcement learning, generative modeling, and probabilistic programming settings.

2.1 Bayesian Inference

2.1.1 Bayesian Inference Preliminaries

Bayesian inference is a principled method for using models of the world in order to understand how certain realizations of observed data took place. Using Bayesian inference necessitates the definition of a *prior*, which is an assumption for how data is distributed absent any observations, and a *likelihood*, which serves as a measure for how well a certain model fits data. Then, given some evidence, the prior and likelihood are combined using Bayes' theorem in order to obtain a *posterior*, a distribution which combines elements of the prior and evidence in accordance with the likelihood to find a distribution from which the evidence could have been sampled [31]. Bayes' theorem gives the relationship between these distributions and data as follows:

$$P(H|E) = \frac{P(E|H) \cdot P(H)}{P(E)} \tag{2.1}$$

where H is a hypothesis, E the observed evidence, $P(H)$ is the prior, $P(E|H)$ is the likelihood, $P(E)$ is the marginal likelihood, and $P(H|E)$ is the posterior [31]. As the marginal likelihood does not affect the relative probabilities of hypotheses, the posterior density is proportional to the likelihood multiplied by the prior.

While in many cases computing the marginal likelihood involves solving an intractable integral, necessitating the use of approximate inference algorithms we will discuss in Section 2.5.1 [130], there are certain classes of likelihood function

for which the appropriate choice of prior leads to a posterior that can be solved analytically. In this case, the prior and posterior are referred to as *conjugate distributions* [157]. We discuss this in detail in Section 4.3.

2.1.2 Bayesian Transfer Learning

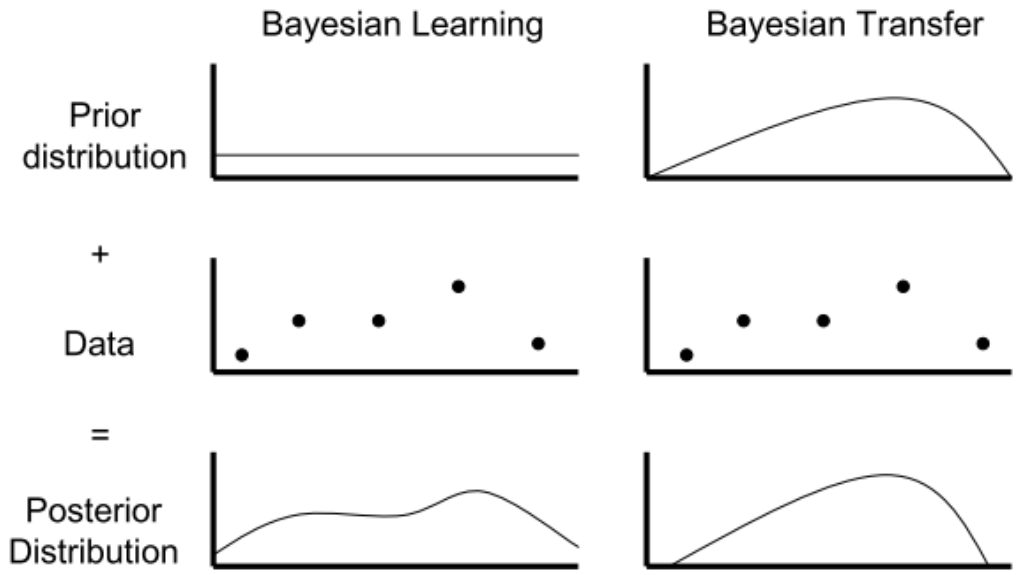


Figure 2.1: Bayesian transfer learning uses a prior distribution based on a source task and modifies it with observed data, to obtain a posterior that incorporates knowledge from both the prior and data. Figure reproduced from [198].

Bayesian methods for transfer learning have a rich history that predates the deep learning era [198]. Bayesian methods are a clear fit for transfer learning applications, as the concepts of prior and posterior distributions map easily to the source and target tasks respectively, as can be seen in Figure 2.1.

The earliest work in this field was in the learning to learn setting, in which it was shown that one can learn a prior over tasks, through Bayesian inference using a non-informative hyperprior [14]. Similar work has found that constructing an informative multivariate Gaussian prior over the parameters of a machine learning

algorithm (in this case, logistic regression) can allow for transfer of information between similar problems in a simple text classification domain [159].

Work has also been done in adapting the Naive Bayes classifier [135] to multi-task scenarios [166]. Placing a Dirichlet prior [117] over the parameters of this classifier will create a generative model for data that would not allow for transfer, as the parameters for each task will be independent of each other. To solve this issue, the Clustered Naive Bayes model [166] is proposed, which uses the Chinese Restaurant Process [4] as the generative model for data; this allows for the clustering of related tasks. While generic approximate inference algorithms such as Markov Chain Monte Carlo [139, 91] would be applicable with this model, Bayesian Hierarchical Clustering [94] is instead used for efficiency purposes. It is found that in low-data scenarios, the Clustered Naive Bayes model outperforms the Dirichlet prior-based model, but is overtaken as the number of samples grows large.

Unlike most deep transfer learning methods, with clever choices of prior and likelihood distributions one can frequently find a closed-form expression for the posterior in a Bayesian setting. This can allow for theoretical results unattainable in other settings, such as those given by the Optimal Bayesian Transfer Learning Classifier [109], which gives a classifier with the Bayesian minimum mean squared error and reduces to an Optimal Bayesian Classifier [43, 44] in the case where there is no information which can be transferred.

2.2 Deep Transfer Learning

Most research since the early-2010s has been focused on algorithms using deep learning. Unlike 'shallow learning' methods such as support vector machines [26] and logistic regression [20], deep learning methods learn a function from input to output through a series of *distributed representations*. These representations are

organized in layers, and typically correspond to different levels of abstraction; with layers representing increasingly complex functions of the input the more transformations have been applied. Visualizing the learned representations is easy for some tasks, such as computer vision tasks, in which one can manually inspect and interpret the learned filters of a convolutional neural network [78].

The idea of using hierarchical, learned representations is both powerful and general, and thus has seen widespread use throughout every sub-field of machine learning. Moreover, the architecture choice of arranging the non-linear transformations of a neural network into stacked layers lends itself naturally to transfer learning, as one can re-use the typically more general features found in layers closer to the input.

There are two 'obvious' transfer learning algorithms that can be applied to deep neural networks which were discovered early on in the deep learning revolution: fine tuning and feature transfer. In the fine-tuning case, one would begin with a network trained on a similar dataset or task as the target dataset or task, and simply retrain starting with the source network weights, and usually a lower learning rate [226].

In the feature transfer case, one would freeze the early layers of a neural network, and retrain the upper layers, with the hope that one would be able to re-use the general concepts learned in lower layers, while adapting the upper layers to the new task. The feature transfer method of transfer learning is surprisingly effective; simply training a support vector machine on convolutional neural network features was enough to obtain state of the art results on many supervised learning tasks in 2014 [160].

2.3 Reinforcement Learning

2.3.1 Reinforcement Learning Preliminaries

Reinforcement Learning [185] refers to a certain set of algorithms that can be used to gain information about, and achieve reward in, a Markov Decision Process (MDP) [18]. An MDP is defined as the tuple (S, A, T, R) where S is a set of states (discrete or continuous), A is a set of actions (also discrete or continuous), $T : S \times A \rightarrow [0, 1]^{|S|}$ is a transition function, giving the probabilities of ending up in any particular state s' when taking action a in state s, and $R : S \times S \rightarrow \mathcal{R}$ is a reward function, which determines how much reward is given by the environment upon making the transition from state s to s'. The MDP is run over multiple timesteps with an agent occupying states in S and executing actions from A. The MDP halts after either reaching a time limit, or when the agent has reached a terminal state [18].

An MDP is commonly referred to in the context of reinforcement learning literature as an *environment*. An MDP is 'solved' when one finds a policy $\pi : S \rightarrow A$ that can obtain optimal *return*, which is simply the sum of reward over all timesteps in the MDP. The set of algorithms that learn to solve an MDP by balancing between *exploration* of trying new (s, a) pairs and *exploitation* of the information already gained in previous trials are referred to as *reinforcement learning algorithms*. Note that algorithms not based on reinforcement learning techniques, such as evolution strategies, can also be used to solve MDPs [170].

Reinforcement learning can be divided into two major approaches, 'classical' reinforcement learning, which solves an MDP using dynamic programming techniques, and 'deep' reinforcement learning, which combines ideas from classical reinforcement learning with learned, non-linear function approximators.

2.3.1.1 Classical Reinforcement Learning

In small environments with discrete state and action spaces, dynamic programming-based algorithms are guaranteed to give the optimal policy, given enough computation and a suitable exploration strategy [103]. One such algorithm is *Q-learning* [215], which learns a function from (s, a) pairs to reward, $Q : S \times A \to R$, using the following dynamic programming update:

$$Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \tag{2.2}$$

where r_t is the reward given at timestep t when transitioning from s_t to s_{t+1}, $\alpha \in (0, 1)$ is a learning rate, and $\gamma \in (0, 1)$ is a discount factor for future rewards [215].

Q-learning does not work unless it is able to visit all (s, a) pairs multiple times, and therefore is completely inapplicable in environments with continuous states and actions, or in discrete environments that are too large to search.

2.3.1.2 Deep Reinforcement Learning

Deep non-linear function approximators such as convolutional neural networks, having already gained popularity in supervised learning, were soon applied to reinforcement learning algorithms to learn in environments with continuous states or actions, or that had state spaces too large to exhaustively search. The first general method to accomplish this was the deep Q-network (DQN) [142], which used a deep neural network to parameterise a Q-function, and quickly became famous for being the first general-purpose reinforcement learning algorithm to be able to play multiple Atari games at human level using the same architecture [17]. The Q-function used in DQN is learned with the loss

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(D)}[(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a, ; \theta))^2] \tag{2.3}$$

where $U(D)$ is a uniform distribution over a dataset of experience D (known as an *experience replay buffer*), γ is a discount factor similar to that found in classical Q-learning, θ is the current set of neural network parameters, and θ^- is a frozen set of neural network parameters from a previous iteration. The use of the experience replay buffer D, as well as frozen weights in the target Q-network, reduced instabilities that would prevent learning using a naive implementation of DQN.

Despite the many tricks employed to stabilise DQN training, it quickly became known as one of the hardest deep reinforcement learning algorithms to use in practice. DQN is incredibly sensitive to the selection of hyperparameters, and training is often unstable, even when using an experience replay buffer. DQN also typically requires more data from the environment than other deep reinforcement learning methods.

While Deep Q-Learning was the impetus for the initial popularity of deep reinforcement learning, researchers quickly discovered that policy gradient algorithms were superior in many ways, often being able to learn using fewer data and with less reliance on the exact specification of hyperparameters. Policy gradient algorithms are all based on the policy gradient theorem, which is a specific case of the score function estimator (the gradient of the likelihood function with respect to the parameters of a model).

The policy gradient theorem can be easily derived:

$$\begin{aligned}
\nabla_\theta \mathbb{E}_a[f(a)] &= \nabla_\theta \textstyle\sum_a p_\theta(a) f(a) \\
&= \textstyle\sum_a \nabla_\theta p_\theta(a) f(a) \\
&= \textstyle\sum_a p_\theta(a) \dfrac{\nabla_\theta p_\theta(a)}{p_\theta(a)} f(a) \\
&= \textstyle\sum_a p_\theta(a) \nabla_\theta \log p_\theta(a) f(a) \\
&= \mathbb{E}_a[f(a) \nabla_\theta \log p_\theta(a)]
\end{aligned}$$

where f is some measure of the quality of an action under the policy, a is an action, and p_θ is the probability of taking a specified action under the policy parameterised by θ [110]. In plain English, the policy gradient theorem tells us that we can solve reinforcement learning tasks by parameterising our policy with a neural network, and then learn the parameters of the policy using gradient ascent on the surrogate objective $\log p(a)$, weighted by scalar reward $f(a)$.

The choice of f in the above derivation is crucial, and different choices lead to different algorithms. The generally accepted best formulation as of the writing of this thesis is Advantage Actor-Critic [141, 174], which takes f to be the advantage function, $Q(s, a) - V(s)$. Both Q and V are learned, in practice typically using one single neural network with two heads. When the advantage function is trained asynchronously over multiple servers, the algorithm is known as Asynchronous Advantage Actor-Critic (A3C) [141] and when trained synchronously in mini-batches on a GPU it is known simply as Advantage Actor-Critic (A2C).

Unlike supervised learning, the choice of optimiser in policy gradient settings is also critical, with many advances in reinforcement learning having come simply from having better optimisers. Naively using optimisers frequently used in supervised learning, such as Adam [111] often does not work in practice. There are generally two main reasons for the poor performance of these optimisers:

1. The set of states and dynamics of the environment seen by the policy change as the policy changes. This non-stationary behaviour causes optimisers with momentum terms to perform more poorly than would be expected.

2. Sudden, large changes in policy are undesirable, as a policy network would not be able to operate correctly when seeing entirely new states. Combined with the 'catastrophic forgetting' problem associated with neural network training, taking large steps in policy could mean destroying all knowledge that was previously learnt.

These two issues encourage the use of optimisers that make small changes to the policy after each gradient update and optimisers specifically designed for policy gradient-based reinforcement learning operate on this principle.

In order to make small changes in policy per gradient update, it is generally also a pre-requisite to be able to take small steps in parameter space. This is an issue, however, as neural network parameter spaces in general have a Riemannian metric structure. The non-Euclidean nature of neural network parameter spaces means that as training progresses, the concept of 'distance' in parameter space will also change. This issue can be circumvented by taking steps in the direction of the natural gradient [7], which gives the direction of steepest descent in a neural network parameter space, factoring in this non-Euclidean nature. The natural gradient is given by

$$\widetilde{\nabla} L(\theta) = G^{-1} \nabla L(\theta) \tag{2.4}$$

where G is the Fisher information matrix [6] and $\nabla L(\theta)$ is the normal gradient of the loss L with respect to parameters θ [7].

Given a neural network with n parameters, the Fisher information matrix is

an $n \times n$ matrix, making storing or inverting this matrix impossible in most cases. Many natural gradient-based optimisers such as Trust-Region Policy Optimization (TRPO) [173] and Actor Critic using Kronecker-Factored Trust Region (ACKTR) [224] therefore instead approximate this gradient without computing or storing the full Fisher information matrix.

2.3.2 Deep Transfer Reinforcement Learning

Results on transfer learning in the deep reinforcement learning setting have, to this point, been fairly discouraging. Markov decision processes suffer from the issue that the set of states, the dynamics, and the reward functions of environments may differ, so transfer learning is much less straightforward than in the supervised learning case.

Indeed, in many cases, it may even be difficult to define what is desired by transfer reinforcement learning. Reinforcement learning algorithms do not generalise in the same way that supervised learning algorithms do; whereas one would expect an image classification algorithm to generalise beyond images found in the training set to those in the same distribution, but not contained in the training data, it is in many cases difficult to train reinforcement learning algorithms to correctly explore and solve even a singular task.

There has, nevertheless, been limited work on obtaining some sort of limited transfer in the reinforcement learning setting. In many instances, these algorithms focus on some narrow aspect of the problem, for instance, transfer between environments in which only the dynamics differ by some small amount. As transfer learning in reinforcement learning problems is still, for the most part, an unsolved problem, many of these techniques are disparate, with the community not having chosen any single consistent direction for attacking the transfer learning problem. Furthermore, many of these techniques only work in simple domains, such as 2D

discrete navigation problems or very simple continuous control problems.

2.3.2.1 Adaptations of Earlier Methods for Deep Transfer Reinforcement Learning

One of the earliest examples of transfer deep reinforcement learning was across modalities. Researchers discovered that, in order to learn how to play the computer game Civilization II, they could search a game manual for relevant passages and condition a neural network parameterising a Q-function on an encoding of the aforementioned text from the manual, along with an encoding of its relevancy [28]. This Q-function was then used to augment Monte-Carlo Search [188], similarly to the use of the Q-function trained on Monte-Carlo rollouts used in AlphaGo [180], which was necessary because similarly to computer Go, full rollouts are infeasible in Civilization II due to the long horizon. It was found that the agent which augmented its playing abilities through information found in the game manuals gave a 34% absolute improvement over an agent that learned simply through playing the game.

Using textual information in order to inform game playing reinforcement learning agents has also been extended with the LanguagE-Action Reward Network (LEARN) framework [81]. LEARN contained a two-stage approach, first training a neural network that attempts to predict if given language data describes the actions in a trajectory. Next, the trained LEARN network is used to shape the reward in the MDP [146], with LEARN providing reward modifications depending on whether the agent's current trajectory seems to match the textual instructions. Unfortunately, results with this framework were weak, with normal reinforcement learning baselines frequently outperforming or performing similarly to LEARN, depending on the task. On average, however, it was found that using LEARN on the difficult Atari game Montezuma's Revenge [17], a naive RL baseline could solve less than 1000 episodes after 500,000 timesteps, whereas providing language-based

rewards allowed more than 1500 episodes to be solved in the same amount of time.

These methods for augmenting game-playing AI have roots in transfer reinforcement learning pre-dating the widespread use of deep learning methods. Rather than learning from textual information in game manuals, human-designed knowledge in the form of computer programs known as *partial programs* [136, 137] were proposed to enable hierarchical reinforcement learning for RTS gameplay. These partial programs did so by allowing a Q-learning [215] agent to optimise choices in an under-specified, human-designed program that gives general instructions on how the structure of an agent for playing the game successfully may look. This work touches on many areas relevant to this thesis; we will discuss transfer learning in the hierarchical reinforcement learning scenario in Chapter 3, and discuss how humans can design programs that can influence learning in Chapter 5. Our work, however, also differs greatly from the work in partial programs, as our hierarchical reinforcement learning methods are trained end-to-end, with very little structure imposed from the human researchers. Our work in probabilistic programming differs from partial programs as in probabilistic programming, one provides a fully-specified, and yet stochastic, computer program and wishes to learn the ways in which the stochasticity may have presented itself to realise a certain output of that program, whereas partial programs wish to learn part of a deterministic program itself.

Another early example of deep transfer reinforcement learning was an adaptation of a known method for supervised transfer learning. It was known that one could re-encode the functions learned by an ensemble of neural networks into one, smaller neural network through a technique known as distillation, which works by training a new neural network to copy the softmax outputs of each member of the ensemble [96]. Policy distillation [168] is the extension of this method to the reinforcement learning setting. Naïvely attempting to copy the Q-values of a Deep Q-Network

[142] does not work as it is difficult to learn the Q-values for every action at once, and one cannot predict the single best action because there may be multiple actions with similar Q-values. Instead, it is found that one can distill a teacher Deep Q-Network into a student by minimising the KL-divergence between the teacher and student's Q-values. The connection to transfer learning comes from the case in which we have multiple teachers from different environments, as we can simply construct a combined Q-value dataset from each of these environments, and use them as targets for the student network. Policy distillation allowed for the creation of a single agent that can play ten different Atari games, and achieved, on average, 89.3% of the score of the individually trained agents [168]. The formulation of policy distillation given in [168] was later theoretically justified and found to be the most reliable when compared against competing formulations [41].

Another method for distillation of multiple task-specific policies, and transfer learning can be found in Distral [187]. Unlike the original formulation of policy distillation [168], which involved distilling several teacher policies into a student policy, Distral distills each teacher policy into a single centroid policy, which is then used to regularise both the teachers and any new policy that is to be learned. Like policy distillation [168], regularisation happens through a KL-divergence penalty, that discourages task-specific policies from straying too far from the centroid. Furthermore, there is an added entropy regularisation term, which prevents both the centroid policies and the task-specific policies from becoming too peaked, thus encouraging exploration. The entropy regularisation term is especially important in the multi-task training setting, as otherwise the centroid policy and all of the task-specific policies could collapse to simply the easiest of all task-specific policies. Distral was able to learn to solve navigation tasks in complex 3D environments, providing more stable training compared to naive RL baselines. In Chapter 3, we will describe an extension of Distral to the hierarchical reinforcement learning case.

Successor Features [12] provide a generalisation of the concept of the successor representation [47], by modeling the reward function of a Markov decision process as the dot product of some features of each transition $\phi(s, a, s')$ where $\phi : S \times A \times S \rightarrow \mathbb{R}^d$ and a learned weight vector $\mathbf{w} \in \mathbb{R}^d$. This formalisation allows for the ideas in [47] to be used in environments with continuous states and actions, as well as allowing for the use of non-linear function approximators in learning $\mathbf{w}$. Successor features give way to a natural framework for transfer learning over environments in which the states, actions, and transition function are held constant, with the reward function being allowed to vary. Suppose that we have learned the optimal value function over several environments. As the reward function is factorized into a ϕ term that is constant across all environments and a $\mathbf{w}$ which differs across all environments, learning the value function in a new environment can simply be reduced to the supervised problem of approximating $\mathbf{w}$ in the new environment. Improvements in multitask learning over DQN are shown in 2D navigation tasks and a continuous control task.

These results were extended with universal successor features [128], which combined elements of successor features [12] with universal value function approximators [172]. While the original formulation of successor features [12] learned features of a transition, this work assumes that there is only one transition that provides reward (this situation occurs naturally in many domains, such as tasks in which one must navigate to a goal at which point they receive reward) so that they may learn features of states rather than transitions. Their algorithm lead to a factorization of a goal-specific Q-function into successor features and goal-specific features, with the successor feature function being shared across all goals, and parameterised by a neural network so that it can be used with any reinforcement learning algorithm. The goal-specific features were also parameterised by a separate neural network which took a goal as input. The authors show improvements in

learning speed after pre-training on a set of disjoint tasks (in this case, goal states) over DQN on a simple 2D navigation task, and over Deep Deterministic Policy Gradient [125] on both a simple continuous control problem and on a realistic robotic arm simulator.

2.3.2.2 Architectures for Deep Transfer Reinforcement Learning

In the following years, not only algorithms, but also many architectures specifically designed to allow for transfer learning were created. One such architecture was the progressive neural network [169]. Given a neural network with parameters $\Theta^{(1)}$ which is trained to solve a task, its features can be transferred to a target neural network with a separate task and parameters $\Theta^{(2)}$ by letting each layer of the target neural network take inputs from the previous layers in both $\Theta^{(1)}$ and $\Theta^{(2)}$. As the features of $\Theta^{(1)}$ may not be directly applicable to the target task, the features coming from $\Theta^{(1)}$ are also passed through a learned adapter layer, which both reduces the dimensionality of the incoming features and adapts them to the new task. As the adapter layers are designed to take an arbitrary number of feature layers as input, transfer from multiple tasks can be achieved by passing multiple feature layers into the adapter. Progressive neural networks were demonstrated to be able to transfer between many variants of the Atari game Pong, between random Atari games [17], and between different tasks in a 3D maze successfully.

The typical scenario for training reinforcement learning algorithms is to get all supervision simply from the reward. When training on a Markov decision process that makes use of visual features (e.g., training a policy to play Atari games from pixels, with the policy parameterised as a convolutional neural network) this setup is wasteful, as every state of the MDP should contain information that can be used to train convolutional filters, even if it is not clear how to use this information until reward is achieved. Moreover, the mapping from pixels to actions will most

likely overfit the domain on which it was trained, and not generalize to new reward functions. DARLA [95] aims to solve this by training a network that first learns how to see, then how to act, and then is able to transfer to new domains without additional retraining. The training pipeline is as follows: first, a disentangled generative model learns how to see in a visual pre-training MDP, in which frames are collected using a random policy. Then, using the previously learned disentangled representations of the states, an agent can be learned with a standard reinforcement learning algorithm. The learned policy can then transfer to a new, similar environment in a zero-shot fashion, as its state representation was chosen to be disentangled by design. DARLA was able to achieve significantly better zero-shot performance than vanilla RL agents on several complex continuous control tasks, and outperformed ablations in which the generative model is entangled.

Another architecture specifically designed for transfer is Metalearning Shared Hierarchies (MLSH) [67]. MLSH aims to learn over multiple tasks a master policy, which can take observations in a specific task, and choose one of several sub-policies which are shared amongst all tasks. At training time the master policy and sub-policies are jointly learned. The sub-policies can afterward be transferred to tasks outside of the training distribution by re-learning a master policy specific to that task. MLSH was able to learn policies that achieved high reward quickly compared to a naive RL algorithm, even on very difficult hierarchical tasks such as 3D navigation in a complex physics simulator.

2.4 Generative Models

2.4.1 Generative Model Preliminaries

Generative modeling, and in particular the task of image generation, has seen enormous amounts of attention and development since the deep learning revolution.

Whereas supervised learning tasks such as classification are concerned with learning a map between some data X and labels Y, generative modeling tasks are concerned with learning a density over data given only unlabelled data X. In doing so, one may sample data that could have plausibly come from this distribution. Before discussing advances in transfer learning with these models, we first summarise the main approaches currently in use.

2.4.1.1 Variational Auto-Encoders

Variational Auto-Encoders (VAEs) [114, 113] were one of the earliest deep generative models to be discovered. VAEs have a structure similar to that of autoencoders, but are trained with the goal of creating a latent space that is disentangled and amenable to interpolation. VAEs work by parameterizing a neural network encoder, $q_\phi(\mathbf{z}|\mathbf{x})$, which takes an input $\mathbf{x}$ and outputs the mean and covariance for a latent $\mathbf{z}$. Likewise, there is also a neural network decoder, $p_\theta(\mathbf{x}|\mathbf{z})$, taking $\mathbf{z}$ and outputting a reconstruction of $\mathbf{x}$. The networks are trained by maximizing the variational lower bound [107, 210]:

$$\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \mathbb{D}_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\theta(\mathbf{z})) \tag{2.5}$$

where we can differentiate through sampling operations with the *reparameterization trick* [114]. Maximizing $\log p_\theta(\mathbf{x}|\mathbf{z})$ trains the networks to reconstruct the data and minimizing $\mathbb{D}_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p_\theta(\mathbf{z}))$ forces $q_\phi(\mathbf{z}|\mathbf{x})$ to be close to the prior $p_\theta(\mathbf{z})$ [114, 113].

2.4.1.2 Generative Adversarial Networks

Generative Adversarial Networks (GANs) [79] were also discovered early into the deep learning revolution, and very quickly became one of the most popular

models. GANs learn to generate data through the adversarial training of two neural networks: a generator G which deterministically turns a latent variable $\mathbf{z}$ from some distribution $p_{\mathbf{z}}(\mathbf{z})$ into some data, and a discriminator D which, given some data, classifies it as having come from G or from the true dataset. The objective to be optimised in GANs is

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))] \tag{2.6}$$

where the term on the left is training the discriminator to become better at differentiating between real and fake samples, and the term on the right is training the generator to fool the discriminator [79].

Using the basic formulation of GANs described here, obtaining a latent corresponding to a datapoint (and as a consequence, computing the likelihood of such a latent under $p_{\mathbf{z}}(\mathbf{z})$) is non-trivial. Evaluating the likelihood of a datapoint in this fashion is necessary for many real-world applications that make use of generative modeling (such as those found in physics applications [72, 30]) and therefore GANs may seem like an unattractive option. The sample quality produced by GANs in the image generation domain as well as their novel adversarial training setup, however, has fueled sustained attention from the machine learning community for years, often at the expense of other methods that have more attractive theoretical properties but produce lower quality samples.

2.4.1.3 Autoregressive Generative Models

Autoregressive models such as PixelRNN [204], PixelCNN [205], WaveNet [203], and GPT-3 [33] are a class of generative model in which any part of the output is generated conditioned on all of the previous parts. For instance, an image would be

generated pixel-by-pixel, being conditioned on each of the already generated pixels, or a waveform would be generated timestep-by-timestep, conditioned on the parts of the waveform already generated.

In mathematical terms, for some datapoint $\mathbf{x}$, its probability is modeled as

$$p(\mathbf{x}) = \prod_{i=1}^{n} p(x_i|x_1, ..., x_{i-1}) \tag{2.7}$$

where there are n 'components' (timesteps, pixels, etc) to the datapoint. With this setup, training is straightforward: one only needs to maximise the likelihood of the data using a standard loss.

Autoregressive models, despite having tractable likelihoods and being relatively easy to train, come with several drawbacks. Both training and inference are slow, requiring many forward passes through a neural network in order to make a single generation (due to the nature of having to generate a single datapoint component-by-component). Sample quality is also typically lower than that of a GAN, but greater than that of flow models or VAEs for image generation.

2.4.1.4 Normalizing Flow Models

Normalizing flow models [54, 161, 55, 112] are another class of generative models with tractable likelihoods. Flow models are formed out of the composition of a series of invertible functions $\{f_1, ..., f_k\}$ such that, for any random variable $\mathbf{z_0}$, we can obtain a corresponding sample $\mathbf{z_k}$ from our data distribution simply by passing it through the series of functions making up the flow model:

$$\mathbf{z}_K = f_K \circ ... \circ f_1(\mathbf{z}_0) = f(\mathbf{z}_0) \ . \tag{2.8}$$

The density of a datapoint $\mathbf{z_k}$ is then calculated with the formula

$$q_K(\mathbf{z}_K) = q_0(\mathbf{z}_0) \prod_{k=1}^{K} \left| \det \frac{\partial f_k}{\partial \mathbf{z}_{k-1}} \right|^{-1} . \tag{2.9}$$

The prior probability density for a flow model is usually taken to be a very simple model, such as a multivariate Gaussian, and must have the same dimensionality as the data itself. Despite being among the worst of deep generative models in terms of sample quality, the simplicity of both generation and density evaluation makes flow models attractive, especially for scientific applications which frequently require accurate and efficient density estimation [72, 30].

In the context of this thesis, normalizing flow models are also especially attractive due to their ability to model a full density. In Bayesian settings, this allows them to be used as surrogates for a likelihood [152] or a posterior [83]. In Chapter 4 we will show how this property of flow models can be exploited in order to efficiently adapt a pre-trained generative model to new situations.

2.4.1.5 Score-Based Generative Models

Score-based generative models [181] are a very recent class of generative models that can generate high-quality samples with many of the advantages given by flow models. Score-based generative models are sampled using Langevin dynamics [218], an iterative method that computes

$$\tilde{\mathbf{x}}_t = \tilde{\mathbf{x}}_{t-1} + \frac{\epsilon}{2} \nabla_{\mathbf{x}} \log p(\tilde{\mathbf{x}}_{t-1}) + \sqrt{\epsilon}\, \mathbf{z}_t \tag{2.10}$$

where we want to sample from $p(\mathbf{x})$, we are given an initial value $\tilde{\mathbf{x}}_0$ from some prior distribution $\pi(\mathbf{x})$, and $\mathbf{z}_t \sim \mathcal{N}(0, 1)$ [181].

As sampling with this method only requires $\nabla_{\mathbf{x}} \log p(\tilde{\mathbf{x}})$, one can train a score network [101], $\mathbf{s}_\theta(\mathbf{x})$, that matches $\nabla_{\mathbf{x}} \log p(\tilde{\mathbf{x}})$ by minimising the objective

$$\frac{1}{2}\mathbb{E}_{p_{\text{data}}}[\|\mathbf{s}_\theta(\mathbf{x}) - \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})\|_2^2] \tag{2.11}$$

which is equivalent to minimising

$$\mathbb{E}_{p_{\text{data}}(\mathbf{x})}\left[\operatorname{tr}(\nabla_{\mathbf{x}}\mathbf{s}_\theta(\mathbf{x})) + \frac{1}{2}\|\mathbf{s}_\theta(\mathbf{x})\|_2^2\right] \tag{2.12}$$

where $\nabla_{\mathbf{x}}\mathbf{s}_\theta(\mathbf{x})$ denotes the Jacobian of $\mathbf{s}_\theta(\mathbf{x})$ [181].

Score-based generative models are capable of producing samples that match, or exceed the quality of those produced by Generative Adversarial Networks, with the added benefit of being able to compute the likelihood of a datapoint under the learned density. In addition, unlike flow models which must be invertible and have strict conditions on the dimensionality of the input and network components, the learnt score network used in a score-based generative model can be arbitrary. These properties make score-based generative models an extremely attractive avenue for future research. The current main drawback to score-based models is that the iterative process through which they are sampled is slow, requiring a forward pass through the neural network $\mathbf{s}_\theta(\mathbf{x})$ at each timestep.

2.4.2 Transfer Learning in Generative Models

While generative modeling has become extremely popular since the deep learning revolution, with architectures such as Variational Auto-Encoders (VAEs) [114], Generative Adversarial Networks (GANs) [79], autoregressive models [205], and normalizing flow models [161] having all been discovered and used extensively in

the past decade, transfer learning results with these models has lagged significantly.

As would be expected, research community interest has consistently been centred around the models that have seemed to have been achieving the best performance. Despite not allowing for tractable likelihoods, this has placed GANs at the forefront of the generative modeling community since its discovery. Likewise, the limited transfer learning results in the generative modeling community have largely been centred around transfer learning in GANs as well.

There has been some success in applying transfer learning methods from the supervised learning domain to generative models. The fine-tuning method discussed in Section 2.2 has been shown to be applicable in the GAN setting, and it has been somewhat surprisingly shown that transferring the discriminator is more important than transferring the generator [214]. These results have been recently extended by researchers who observed that freezing the first few layers in the discriminator before transfer is effective [143].

It has also been found that transfer learning in GANs can be achieved through the re-training of batch normalization parameters [102] in the generator of a GAN [147]. Unlike normal GAN training, however, the GAN generator in this scenario is updated using supervised learning, changing the network's batch normalization parameters in order to minimize the L1 and perceptual losses between an image from a transfer set, and the image[1] of its corresponding latent vector under the generator. As GANs are not invertible by nature, this necessitates a costly optimisation step, similar to that found in Generative Latent Optimisation [25].

A more recent method for transfer learning in GANs called Few-Shot GAN applies a singular value decomposition to both the generator and discriminator of a GAN, and then adapted the singular values to a transfer set while leaving the singular vectors fixed [164]. This method was able to achieve more effective

[1] In both of its senses

out-of-distribution transfer, with a lower number of learned parameters compared to competing GAN transfer methods.

Another result called Latent Constraints [59] aimed to turn a pre-trained, unconditional VAE into a conditional VAE. Latent Constraints works by simultaneously training two discriminators in an actor-critic setting, one to distinguish between reconstructions of class-specific data and samples from the model and another to ensure that reconstructions are realistic. These discriminators are trained with the goal of optimizing the latent vectors so that the originally unconditional VAE can naturally sample from a specific class. This method comes with the downside, however, of having to retrain the discriminators for every new class from which one wants to sample.

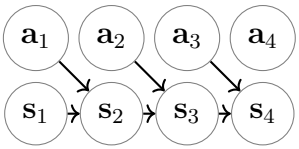
2.5 Probabilistic Programming

While to this point we have solely discussed transfer learning as an avenue to inject knowledge or structure into a novel machine learning task, from a separate machine learning task, there are also ways for researchers themselves to guide the learning process. Human-based guidance of learning has a long history in the artificial intelligence community, with memorable examples being the ELIZA chatbot [217] and the proliferation of Prolog-based expert systems [29].

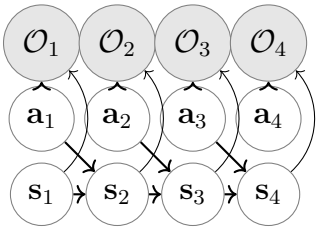
Over the past few decades, expert systems have fallen out of favor with the artificial intelligence community, in strong favor of machine learning-based methods. That is not to say, however, that human-based guidance of learning has no place in modern artificial intelligence. Since the deep learning revolution, researchers have striven to combine powerful, but uninterpretable non-linear functions learned by neural networks, with the sort of disentangled and interpretable representations such as those found in expert systems [179].

One such method that has become popular since the 2010s is probabilistic programming [80]. Probabilistic programming techniques can be used in several different ways, but one of the most popular and exciting is the use case of finding posteriors over latent variables in a stochastic simulator. Such simulators may already exist and be in use, such as those found in high-energy physics simulations [15, 16] or epidemiological simulations [82, 223, 48]. Otherwise, researchers may write a model that attempts to fit or explain given data, and still find posteriors over latent variables in that model, even when observed data was not generated from that process. This method has also been used to find interpretable computer programs that explain phenomena which could not possibly be generated from a statistical model, such as the behaviour of African lions [52] or the structure of human faces [119]. Probabilistic programs are extremely simple to implement, requiring one to only write a stochastic simulator, or make minor adjustments to an existing simulator, so that it can interface with a probabilistic programming library such as Anglican [197], Pyro [24], PyProb [16], Stan [34], Edward [199], or PyMC3 [171].

Probabilistic programming has advantages over competing methods for discovering the latent structure in a stochastic simulator, such as supervised learning. Probabilistic programming methods, being inherently Bayesian, provide uncertainty estimates for their predictions. While it is known that the outputs of a Softmax function used for classification cannot be naïvely interpreted as probabilities (in that higher numbers from a Softmax classifier do not indicate more confidence) [87], there are ways to achieve uncertainty estimates for classifiers, such as through variational methods [97] or by exploiting properties of neural networks trained with dropout [182, 70, 69]. Unfortunately these methods, known as Bayesian Neural Networks [77], are known to work poorly in practice. Probabilistic programming methods, on the other hand, are guaranteed to converge to the correct posterior with



(a) graphical model with states and actions



(b) graphical model with optimality variables

Figure 2.2: The graphical models for planning as inference with states and actions (a), and with optimality variables for reward information (b). Figure reproduced from [123].

an inference engine based on Markov Chain Monte Carlo [139, 91], given enough compute, with established methods being used to detect whether convergence has been achieved or not [74].

Probabilistic programming techniques often scale poorly, as they require the execution of often computationally expensive simulators thousands of times in order to find a posterior given a single observation. Amortised inference methods such as inference compilation [121], discussed in detail in Section 2.5.1.4 can increase the scalability and efficiency of probabilistic programming methods by using non-linear function approximators to propose locations in the latent variable space to sample.

Through the planning as inference framework [123], probabilistic programming methods can also be used to find a policy that maximises reward in a Markov Decision Process [61]. This formulation involves rewriting the MDP as the graphical model in Figure 2.2, in which each state is conditioned on the previous states and actions. In order to incorporate reward information as well, binary 'optimality variables' $\mathcal{O}_t$, which are taken to be 1 when the action in a time step is optimal and 0 when it is not, are added for each time step. These optimality variables are used as observations, with the graphical model conditioned on $\mathcal{O}_t = 1$ for all t, giving us the inference problem that will solve the MDP. We elaborate on this formulation in Section 3.3.1 and use it to transfer between tasks in a hierarchical reinforcement

learning setting in Chapter 3.

When probabilistic programming methods are used in the context of simulation-based inference [38], some special terminology is adopted. Each forward run of the simulator is referred to as a *trace*, with an *empirical posterior distribution* being formed out of several traces which represent samples from the posterior. When collecting data for posterior inference, many traces will not be particularly useful for the empirical posterior distribution, due to having low weight (when using importance sampling), or being too correlated with other samples (when using MCMC). The approximate number of useful samples for algorithms such as importance sampling [115] is known as the *effective sample size* [183].

2.5.1 Approximate Inference Algorithms

Besides the programming language in which the library is implemented, core to a researcher's choice of probabilistic programming library are the methods of probabilistic inference implemented by that library. As exact inference within these simulators is often intractable, there are a wealth of approximate inference algorithms available, with most probabilistic programming libraries providing several. Each comes with their own tradeoffs in terms of required compute and accuracy, and with only some of these methods providing strict guarantees with regards to whether one has sampled enough to ensure that they have found a correct posterior. We here list some of the most common ones.

2.5.1.1 Markov Chain Monte Carlo

The most basic of algorithms for finding a posterior given a stochastic simulator and an observation is Markov Chain Monte Carlo (MCMC) [139, 91]. MCMC-based algorithms work by creating a Markov Chain that has the posterior as its equilibrium distribution, so that running the Markov Chain will give latent variables in the

Algorithm 1 Metropolis-Hastings Markov Chain Monte Carlo

Input: Target distribution π, initial parameters θ_0, number of iterations n
Define proposal distribution $Q(y|x)$
for i in $0...n$ **do**
    Propose new parameters $\theta_p \sim Q(y|x)$
    Compute acceptance probability $A = min(1, \frac{\pi(\theta_p)Q(\theta_i|\theta_p)}{\pi(\theta_i)Q(\theta_p|\theta_i)})$
    With probability A let $\theta_{i+1} = \theta_p$, else let $\theta_{i+1} = \theta_i$
return θ_n

simulator that would result in the observed data. The basic algorithm is given in Algorithm 1.

An English-language description of the algorithm is as follows: we start with some random setting of the latent variables in the simulator, θ_0. At each timestep i, we propose a new set of latent variables θ_p, and measure how well they match the observed data compared to the latent variables. We then update our current set of latent variables with probability equal to that ratio.

MCMC has many drawbacks compared to other methods. For one, MCMC is notorious for being relatively difficult to parallelise, as the dependence of the setting of latent variable at each timestep depending on the last renders it to be an inherently sequential algorithm. Another drawback is its compute requirements, often needing to be run for days or weeks (depending on the speed of the simulator and difficulty of the inference problem) in order to converge. MCMC is used despite these drawbacks, however, because unlike other methods it is guaranteed to converge to the correct posterior given enough time and compute.

One popular method used to check for convergence is the Gelman-Rubin convergence diagnostic [37], which compares the between-chain and within-chain variances using multiple independent MCMC chains [74, 32]. Given M chains of length N, and given model parameter θ with sample posterior mean and variance in the m^{th} chain being $\hat{\theta}_m$ and $\hat{\sigma}_m^2$ respectively and overall sample posterior mean being $\hat{\theta}_m$, the between-chain and within-chain variances are given by

$$B = \frac{N}{M-1}\sum_{m=1}^{M}(\hat{\theta}_m - \hat{\theta})^2 \tag{2.13}$$

$$W = \frac{1}{M}\sum_{m=1}^{M}\hat{\sigma}_m^2 \tag{2.14}$$

where we note we can obtain a single W as at convergence, all m within-chain variances should be equal. Then the pooled variance

$$\widehat{V} = \frac{N-1}{N}W + \frac{M+1}{MN}B \tag{2.15}$$

is an unbiased estimator of the variance of the parameter of interest θ. Taking the ratio between $\widehat{V}$ and σ^2 would give the scale reduction factor R, which would tell us whether the Markov chain had converged. As σ^2 is unknown, however, and must be estimated by data, we can derive an overestimate of R by underestimating σ^2 using W. This gives the potential scale reduction factor $\widehat{R}$

$$\widehat{R} = \frac{\widehat{V}}{W} \tag{2.16}$$

which can be corrected for sampling variablity with $\widehat{R}_c$

$$\widehat{R}_c = \frac{d+3}{d+1}\widehat{R} \tag{2.17}$$

where d is the estimated degrees of freedom for a Student-t approximation to posterior inference, and can be estimated with $d \approx \frac{2\widehat{V}}{\widehat{var}(\widehat{V})}$. $\widehat{R}_c$ will go to 1 for each parameter of interest θ upon convergence [74, 32].

As MCMC chains start from random locations sampled from the prior and only represent posterior samples after $\widehat{R}_c$ nears 1, the part of the MCMC chain which

Algorithm 2 Sequential Importance Sampling

Input: Prior distribution p, proposal distribution π number of traces N, number of timesteps k, observations $y_{0:k}$
Output: Empirical posterior weights $w_1, ..., w_N$
for i in $1...N$ **do**
    Sample $x_k^i \sim \pi(x_k|x_{0:k-1}^i, y_{0:k})$
for i in $1...N$ **do**
    Compute unnormalized weights $\hat{w}_k^i = w_{k-1}^i \frac{p(y_k|x_k^i)p(x_k^i|x_{k-1}^i)}{\pi(x_k^i|x_{0:k-1}^i, y_{0:k})}$
for i in $1...N$ **do**
    Compute normalized weights $w_k^i = \frac{\hat{w}_k^i}{\sum_{i=1}^N (w_k^i)^2}$
return $\{(x_k^1, w_k^1)...(x_k^N, w_k^N)\}$

was sampled prior to convergence (known as the *burn-in*) must be removed in order to avoid contaminating the empirical posterior with samples near the prior.

Due to the nature of MCMC, in which the parameters for each sample are based on those from previous timesteps, samples from consecutive traces will be *correlated*. In order to accurately sample from the posterior, one would desire uncorrelated samples, meaning one would need to determine the *lag* t for which every t^{th} trace is sufficiently uncorrelated with the trace generated t MCMC iterations previous. Given N traces, the autocorrelation ρ_t, at lag t is given by the formula

$$\rho_t = \frac{\sum_{i=0}^{N-t}(\theta_i - \mu)(\theta_{i+t} - \mu)}{\sum_{i=0}^{N}(\theta_i - \mu)^2} \tag{2.18}$$

where θ is any parameter of interest and μ is the mean of $p(\theta)$ [75]. In practice, one would find a value of t for which ρ_t is sufficiently low (close to 0) for every relevant parameter, and then save every t^{th} draw from the Markov Chain, ensuring uncorrelated samples while also reducing disk space requirements by orders of magnitude.

2.5.1.2 Importance Sampling

Importance sampling [148] is a method for estimating integrals of functions from which one cannot sample, which is also amenable to finding and sampling from posterior distributions. In the Bayesian inference scenario, it is commonly referred to as the *Sequential Importance Resampling Particle Filter* [115], as particles (traces) are sampled from another distribution, known as the *proposal distribution* (which can be as simple as the prior distribution defined by the simulator) and then reweighted in accordance with the likelihood function, which determines how closely traces match with the observed ground truth. It can clearly be seen that this algorithm will be more sample efficient the closer the proposal distribution is to the posterior distribution, as samples from the proposal will be assigned high weights with higher probability the closer the proposal distribution is to the posterior.

The basic algorithm is given in Algorithm 2 [57]. Using this method, the effective sample size $\hat{N}$ is simple to compute:

$$\hat{N} = \frac{1}{\sum_{i=1}^{N} (w_k^i)^2} \,. \tag{2.19}$$

If the effective sample size does not reach a certain threshold, it is common to resample the particles $x_k^1 ... x_k^N$ with probabilities proportional to their weights, making the new distribution obtained by the resampling procedure unweighted [50].

2.5.1.3 Stochastic Variational Inference

Stochastic Variational Inference (SVI) [100] is a method by which one can scalably perform posterior inference over a large dataset, using variational inference techniques [107, 210]. Unlike MCMC [139, 91] which would need to use the entire dataset

at each timestep, SVI performs posterior inference through an optimisation problem which can use minibatches of data, similar to modern supervised learning training setups. This optimisation problem is formulated by maximising the evidence lower bound (ELBO) similarly to the equation to train a VAE given in Section 2.4.1.1 using the mean-field approximation, which assumes that the latent variables in the model are independent from each other, so that their joint distribution can be written as the product of each latent distribution. The natural gradient [7], described in Section 2.3.1.2, is used to solve this optimisation problem.

2.5.1.4 Inference Compilation

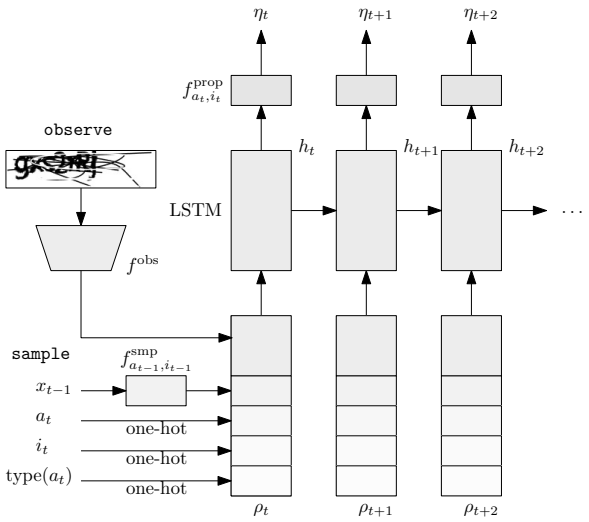


Figure 2.3: The LSTM [99] architecture used in the original formulation of inference compilation, with the left-hand side variables being the LSTM inputs and $\{\eta_t, ..., \eta_{t+n}\}$ being the proposal distribution parameters. Figure reproduced from [121].

Inference compilation [121] is a method used to amortize the cost of inference in probabilistic programming settings over many different observations using deep neural networks, which parameterise a proposal distribution. In its original form described in [121], this neural network is an LSTM [99] which takes as inputs observation embeddings; sample embeddings; previous sample values; and address,

instance, and proposal type embeddings. The LSTM outputs proposal distribution parameters and is trained to minimize the expected Kullback-Leibler divergence between the posterior and the learned proposal distribution over the possible distribution of observations. In its original incarnation, the learned proposal distribution was then used in a sequential importance sampling [115] inference engine.

These ideas were extended to the Metropolis-Hastings MCMC case [139, 91] through a framework known as Lightweight Inference Compilation (LIC) [124]. LIC seeks to amortize inference in Lightweight Metropolis Hastings (LMH) [221, 163] by using neural networks to parameterise a proposal distribution which approximates the single-site Gibbs sampler used in LMH inference. Similarly to the objective in [121], LIC minimises the expected KL divergence between the learned proposal distribution and the posterior, with the expectation taken over the marginal distribution of observations. Their architecture, however, differs in that LIC forgoes the LSTM used in [121] and instead finds a neural embedding of each latent variable and its Markov Blanket [153], combining each variable of the Markov Blanket into a fixed-size vector with a graph convolutional network [49]. The concatenation of these vectors then serves as the parameters of the proposal distribution.

2.5.2 Implementation of Probabilistic Programming Languages

While early probabilistic programming languages such as BLOG [140] and Probabilistic Scheme [155] required either the creation of an entire programming language from scratch or inefficiently implemented Bayesian inference through exhaustive search, modern probabilistic programming languages are able to be seamlessly embedded into an existing programming language, and perform inference within those languages efficiently. The current paradigm under which probabilistic programming languages are implemented is through a method known as transformational compi-

lation [221]. Transformational compilation allows the probabilsitic programming language to control execution traces by rewriting the program itself and then using the native compiler or interpreter to run this new program which tracks information necessary to use inference algorithms such as MCMC [221].

Transformational compilation works by assigning a name to each call to a stochastic primitive (such as samples from a uniform or normal distribution) in an execution trace. These names are used to store the values sampled from these stochastic primitives (along with other necessary information) in a database, such that the program could be deterministically re-executed by looking up these values. This property makes MCMC in trace space possible, as one can deterministically replay existing traces and then propose new parameters for a random call midway (as described on line 4 of Algorithm 1). Should that affect the dynamics of the rest of the program, new random calls can be made, with their names and values stored in the database [221].

In order to ensure high reuse of the values stored in the database, variables are typically named with reference to their position in the code, i.e., their positions in the stack trace of the program when they are called. This naming scheme keeps similar random calls aligned across traces, reducing misalignments that would result from a naive naming scheme, which would cause unnecessary computation resulting from needing to re-sample already computed values from the prior [221].

2.6 Discussion

While transfer learning has been widely studied since the advent of machine learning, there are still many unexplored and under-explored areas. On the generative model front, very few attempts have been made to adapt transfer learning methods to architectures other than GANs. This seems especially important, as state-of-the-art

language models such as GPT-3 [33], audio models such as WaveNet [203], and likelihood-based generative models such as normalising flow models [161] require significant amounts of computation to train, thus benefiting from advances in transfer learning enormously.

On the reinforcement learning front, progress has been disparate and slow, mostly due to our currently poor understanding of deep reinforcement learning in general. The features learned by policy gradient methods [141] or deep Q-networks [142] are qualitatively different from those learned by backpropagation [167] in a supervised learning setting, and thus transfer learning methods that were developed for supervised learning techniques generally do not apply to the reinforcement learning setting at all.

Probabilistic programming is an exciting new technique for combining human knowledge with machine learning techniques. While it is computationally expensive, recent advances have been made that allow for speed and scalability without sacrificing the benefits given by probabilistic programming frameworks, such as its inherent interpretability and allowance for uncertainty estimates.

Chapter 3

Transfer Learning in Markov Decision Processes

6.373 The world is independent of my will.

6.374 Even if everything we wished were to happen, this would only be, so to speak, a favour of fate, for there is no logical connexion between will and world, which would guarantee this, and the assumed physical connexion itself we could not again will.

Tractatus Logico-Philosophicus
Ludwig Josef Johann Wittgenstein [222]

3.1 Overview

Reinforcement learning, being one of the most computationally expensive subfields of machine learning, stands to benefit greatly from transfer learning methods. Even more so are hierarchical reinforcement learning methods, which are orders of magnitude more difficult to train, both in terms of hyperparameter searching and compute required to train a model.

One major goal of the reinforcement learning community is the creation of reinforcement learning algorithms that can generalise across or adapt quickly to new tasks by exploiting similarities in reward, dynamics, or other types of Markov

decision process structure. One such algorithm is Distral [187], which utilises a prior, centroid policy into which policies over multiple tasks are distilled. This centroid policy can then be used as a regulariser which guides learning of policies on new tasks.

In this chapter, we introduce Multi-task Soft Option Learning (MSOL), an extension of Distral to the hierarchical reinforcement learning setting. MSOL allows for the learning of temporally extended actions (known as options) that are shared amongst tasks, similarly to the centroid policy of Distral [187]. We demonstrate that our method produces sensible options that allow for fast transfer of policies to new tasks in several benchmark environments that benefit from hierarchical learning.

3.2 Introduction

A key challenge in deep reinforcement learning is to scale current approaches to complex tasks without requiring a prohibitive number of environmental interactions. One promising approach is to construct or learn efficient exploration priors to focus on more relevant parts of the state-action space, reducing the number of required interactions. This includes, for example, reward shaping [146], curriculum learning [19], meta-learning [212], and transfer learning [187].

In particular, transfer learning does not require human designed rewards or curricula, instead allowing the network to learn what and how to transfer knowledge between tasks. One promising way to capture such knowledge is to decompose policies into a hierarchy of sub-policies (or skills) that can be reused and combined in novel ways to solve new tasks [186]. This idea of hierarchical reinforcement learning is also supported by findings that humans appear to employ a hierarchical mental structure when solving tasks [27]. In such a hierarchical policy, lower-level, *temporally extended* skills yield directed behaviour over multiple time steps. This has

two advantages:

1. it allows efficient exploration, as the target states of skills can be reached without having to explore much of the state space in between, and

2. directed behaviour also reduces the variance of the future reward, which accelerates convergence of estimates thereof.

On the other hand, while a hierarchical approach can significantly speed up exploration and training, it can also severely limit the expressiveness of the final policy and lead to suboptimal performance when the temporally extended skills are not able to express the required policy for the task at hand.

Many methods exist for learning such hierarchical skills, e.g. [186, 11, 85]. The key challenge is to learn skills which are diverse, and relevant for future tasks. One widely used approach is to rely on additional human-designed input, often in the form of manually specified subgoals [208, 145] or a fixed temporal extension of learned skills [67]. While this can lead to impressive results, it is only applicable in situations where relevant subgoals or temporal extension can be easily identified a priori.

We propose Multi-task Soft Option Learning (MSOL), an algorithm to learn hierarchical skills from a given distribution of tasks without any additional human specified knowledge. MSOL trains simultaneously on multiple tasks from this distribution and autonomously extracts sub-policies which are reusable across them.

Importantly, unlike prior work [67], our proposed *soft option* framework avoids several pitfalls of learning options from multiple tasks, which arise when skills are jointly optimised with a higher-level policy that determines when each skill is used. Generally, as each skill must be used for similar purposes across all tasks, to learn consistent behaviour, a complex training schedules is required to assure a nearly converged higher-level policy before skills can be updated [67]. However, once

a skill has converged it can be hard to change its behaviour without hurting the performance of higher-level policies that rely it. Training is therefore prone to end up in local optima: even if changing a skill on *one* task could increase the return, it would likely lead to lower returns on *other* tasks in which it is currently used. This is particularly an issue when multiple skills have learned similar behaviour, preventing the learning of a diverse set of skills.

MSOL alleviates both difficulties. The core idea is to learn a 'prototypical' – or *prior* – behaviour for each skill, while allowing the actually-executed skill on each task – the *posterior* – to deviate from it if the specific task rewards require it. Penalizing deviations between the prior and posteriors from different tasks gives rise to skills that are consistent across tasks, and can be elegantly formulated in the PAI framework [123]. This distinction between prior and task-dependent posterior obviates the need for complex training schedules: every task can change their posterior independently of each other and discover new skills without direct interference in other tasks. Nevertheless, the penalization term encourages skills to be similar across tasks *and* rewards higher-level policies for preferring such more specialised skills. We discuss in more detail in Section 3.4.5 how this helps to prevent the aforementioned local optima.

In addition to these optimisation pitfalls, the idea of soft options also alleviates the restrictiveness of hierarchical policies. New tasks can make use of learned skills, by initialising their posterior skills from the priors, but are not restricted by them. The penalization term between prior and posterior acts here as learned shaping reward, guiding the exploration on new tasks towards previously relevant behaviour, without requiring the new policy to exactly match previous behaviour. In difference to prior work, MSOL can thus even learn tasks that are not solvable with previously learned skills alone. Finally, we show how the *soft option* framework gives rise to a natural solution to the challenging task of learning option-termination

policies.

Our experiments demonstrate that MSOL outperforms previous hierarchical and transfer-learning algorithms during transfer tasks in a multitask setting. Unlike prior work, MSOL only modifies the regularised reward and loss function and does not require specialised architectures, or artificial restrictions on the expressiveness of either the higher-level or intra-option policies.

3.3 Preliminaries

An agent's task is formalised as a MDP $(\mathcal{S}, \mathcal{A}, \rho, P, r, \gamma)$, consisting of the state space $\mathcal{S}$, action space $\mathcal{A}$, initial state distribution ρ, transition probability $P(s_{t+1}|s_t, a_t)$ of reaching state s_{t+1} by executing action a_t in state s_t, reward $r(s_t, a_t) \in \mathbb{R}$ that an agent receives for this transition, and discount factor $\gamma \in [0, 1]$. An optimal policy π chooses actions that maximise the expected return $R_t(s_t) = \mathbb{E}_\pi[\sum_k \gamma^k r_{t+k}]$ consisting of discounted future rewards.

3.3.1 Planning as Inference

Planning as inference (PAI) [195, 123], frames reinforcement learning as a probabilistic inference problem. The agent learns a distribution $q_\phi(a|s)$ over actions a given states s, i.e., a policy, parameterised by ϕ, which induces a distribution over trajectories τ of length T, i.e., $\tau = (s_1, a_1, s_2, \ldots, a_T, s_{T+1})$:

$$q_\phi(\tau) = \rho(s_1) \prod_{t=1}^{T} q_\phi(a_t|s_t)\, P(s_{t+1}|s_t, a_t)\,. \tag{3.1}$$

This can be seen as a structured variational approximation of the optimal trajectory distribution. Note that the true initial state probability $\rho(s_1)$ and transition probability $P(s_{t+1}|s_t, a_t)$ are used in the variational posterior, as we can only control the policy, not the environment.

A significant advantage of this formulation is that it is straightforward to incorporate information both from prior knowledge, in the form of a prior policy distribution, and the task at hand through a likelihood function that is defined in terms of the achieved reward. The prior policy $p(a_t|s_t)$ can be specified by hand or, as in our case, learned (see Section 3.4). To incorporate the reward, we introduce a binary *optimality variable* $\mathcal{O}_t$ [123], whose likelihood is highest along the optimal trajectory that maximises return: $p(\mathcal{O}_t = 1|s_t, a_t) = \exp\big(r(s_t, a_t)/\beta\big)$, where for $\beta \to 0$ we recover the original reinforcement learning problem. The constraint $r \in (-\infty, 0]$ can be relaxed without changing the inference procedure [123]. For brevity, we denote $\mathcal{O}_t = 1$ as $\mathcal{O}_t \equiv (\mathcal{O}_t = 1)$. If a given prior policy $p(a_t|s_t)$ explores the state-action space sufficiently, then $p(\tau, \mathcal{O}_{1:T})$ is the distribution of desirable trajectories. PAI aims to find a policy such that the variational posterior in Equation 3.1 approximates this distribution by minimizing the KL divergence:

$$\begin{aligned} \mathcal{L}(\phi) &= \mathbb{D}_{\mathrm{KL}}(q_\phi(\tau) \parallel p(\tau, \mathcal{O}_{1:T}))\,, \text{ where} \\ p(\tau, \mathcal{O}_{1:T}) &= \rho(s_1)\prod_{t=1}^{T} p(a_t|s_t)P(s_{t+1}|s_t, a_t)p(\mathcal{O}_t|s_t, a_t). \end{aligned} \tag{3.2}$$

3.3.2 Multi-task Learning

In a multi-task setting, we have a set of different tasks $i \in \mathcal{T}$, drawn from a task distribution with probability $\xi(i)$. All tasks share state space $\mathcal{S}$ and action space $\mathcal{A}$, but each task has its own initial-state distribution ρ_i, transition probability $P_i(s_{t+1}|s_t, a_t)$, and reward function r_i. Our goal is to learn n tasks concurrently, distilling common information that can be leveraged to learn faster on new tasks from $\mathcal{T} \in \{1, ..., n\}$. In this setting, the prior policy $p_\theta(a_t|s_t)$ can be learned jointly with the task-specific posterior policies $q_{\phi_i}(a_t|s_t)$ [187]. To do so, we simply extend Equation 3.2 to

$$\begin{aligned}\mathcal{L}(\{\phi_i\}, \theta) &= \mathbb{E}_{i\sim\xi}\big[\, \mathbb{D}_{\mathrm{KL}}(q_{\phi_i}(\tau) \,\|\, p_\theta(\tau, \mathcal{O}_{1:T}))\,\big] \\ &= -\frac{1}{\beta}\, \mathbb{E}_{i\sim\xi,\tau\sim q}\left[\sum_{t=1}^{T} R_{i,t}^{\mathrm{reg}}\right],\end{aligned} \tag{3.3}$$

where $R_{i,t}^{\mathrm{reg}} := r_i(s_t, a_t) - \beta \ln \frac{q_{\phi_i}(a_t|s_t)}{p_\theta(a_t|s_t)}$ is a regularised reward. Minimizing the loss in Equation 3.3 is equivalent to maximising the regularised reward $R_{i,t}^{\mathrm{reg}}$. Moreover, minimizing the term $\mathbb{E}_{\tau\sim q}\big[\ln \frac{q_{\phi_i}(a_t|s_t)}{p_\theta(a_t|s_t)}\big]$ implicitly minimises the expected KL-divergence $\mathbb{E}_{s_t\sim q}\big[\mathbb{D}_{\mathrm{KL}}[q_{\phi_i}(\cdot|s_t)\|p_\theta(\cdot|s_t)]\big]$. In practice (see Section 3.7.1) we will also make use of a discount factor $\gamma \in [0, 1]$. For details on how γ arises in the PAI framework we refer to [123].

3.3.3 Options

Options [186] are skills (abstract, temporally-extended actions) that generalise primitive actions and consist of three components:

1. an intra-option policy $p(a_t|s_t, z_t)$ from which primitive actions are drawn according to the currently active option z_t,
2. a probability $p(b_t|s_t, z_{t-1})$ of terminating the *previously* active option z_{t-1}, and
3. an initiation set $\mathcal{I} \subseteq \mathcal{S}$, which we simply assume to be $\mathcal{S}$.

Note that by construction, the higher-level (or master-) policy $p(z_t|z_{t-1}, s_t, b_t)$ can only select a new option z_t if the previous option z_{t-1} has terminated.

3.4 Algorithm

We aim to learn a reusable set of options that allow for faster training on new tasks from a given distribution. To differentiate ourselves from classical 'hard' options,

Algorithm 3 Multi-task Soft Option Learning

Input: Number m of options to learn, n_i different training tasks env_i, *fixed* termination prior $p^T(b_t) = (1-\alpha)^{b_t}\alpha^{1-b_t}$ and *fixed* master prior $p^H(z_t|z_{t-1}, b_t) = (1-b_t)\,\delta(z_t - z_{t-1}) + b_t \frac{1}{m}$
Initialise once: *Learnable* termination prior $p_\theta^T(b_t|s_t, z_{t-1})$, intra-option prior $p_\theta^L(a_t|s_t, z_t)$
Initialise for each task i**:** Learnable termination posterior $q_{\phi_i}^T(b_t|s_t, z_{t-1})$, master policy $q_{\phi_i}^H(z_t|s_t, z_{t-1}, b_t)$, intra-option policy $q_{\phi_i}^L(a_t|s_t, z_t)$
// Note that this leads to a total of m intra option priors for the m different values of $z \in \{1 \dots m\}$
// and a total of $m \times n_i$ intra option posteriors.

while not converged **do**
  // Collect data
  for each task i **do**
    if beginning of episode **then**
      $s_0 \leftarrow \mathrm{env.reset}()$
      $b_0 \leftarrow 1$ *// This allows q^H to sample a new option z_0.*
    else
      $b_t \sim q_{\phi_i}^T(b_t|s_t, z_{t-1})$
    $z_t \sim q_{\phi_i}^H(z_t|s_t, z_{t-1}, b_t)$
    $a_t \sim q_{\phi_i}^L(a_t|s_t, z_t)$
    $s_{t+1}, r_t \sim \mathrm{env}_i(a_t)$
    // Compute regularised reward (eq. Equation 3.7); note that we use the fixed priors here
    $R_t^{\mathrm{reg}} \leftarrow r_i(s_t, a_t) - \beta \ln \frac{q_{\phi_i}^H(z_t|s_t, z_{t-1}, b_t)}{p^H(z_t|z_{t-1}, b_t)} - \beta \ln \frac{q_{\phi_i}^L(a_t|s_t, z_t)}{p_\theta^L(a_t|s_t, z_t)} - \beta \ln \frac{q_{\phi_i}^T(b_t|s_t, z_{t-1})}{p^T(b_t)}$
    Add $s_t, s_{t+1}, r_t, R_t^{\mathrm{reg}}$ to $\mathcal{D}_i$

  // Update parameters ϕ_i
  for each task i **do**
    Update ϕ_i using A2C or PPO on $\mathcal{D}_i$ as described in Section 3.7.1.
    Note that for PPO, R_t^{reg} needs to be re-computed and updated between gradient updates to ϕ_i as the regularisation terms change.

  // Update parameters θ
  Update θ to minimise
  $\sum_i \mathbb{E}_{\mathcal{D}_i}\Big[\mathbb{D}_{\mathrm{KL}}\Big(q_{\phi_i}^L(a_t|s_t, z_t) \,\|\, p_\theta^L(a_t|s_t, z_t)\Big) + \mathbb{D}_{\mathrm{KL}}\big(\hat{q}_{\phi_i}(b=1|s_t, z_{t-1}) \,\|\, p_\theta^T(b_t|s_t, z_{t-1})\big)\Big]$
  Here, $\hat{q}_{\phi_i}(b=1|s_t, z_{t-1}) = \sum_{z_t \neq z_{t-1}} q_{\phi_i}^H(z_t|s_t, z_{t-1}, b_t = 1)$, i.e. instead of distilling the average of $q_{\phi_i}^T$ into p_θ^T, we distill whether the *master policy* $q_{\phi_i}^H$ would have changed the option z_t if it had the chance (i.e. if $b_t = 1$). Since $q_{\phi_i}^T$ is regularised to be similar to the *fixed* p^T, this approach allows us to learn a termination prior p_θ^T which is less influenced by our manually specified prior p^T, and more by what is needed for the task.

which, once learned, do not change during new tasks, we call our novel approach *soft-options*. Each soft-option consists of an option *prior*, denoted by p_θ, which is shared across all tasks, and a task-specific option *posterior*, denoted by q_{ϕ_i} for task i. Unlike most previous work, e.g. [67], we learn both intra-option and termination policies. The priors of both the intra-option policy p_θ^L and the termination policy p_θ^T capture how an option typically behaves and remain fixed once they are fully learned. At the beginning of training on a new task, they are used to initialise the task-specific posterior distributions $q_{\phi_i}^L$ and $q_{\phi_i}^T$. During training, the posterior is then regularised against the prior to prevent inadvertent unlearning. However, if

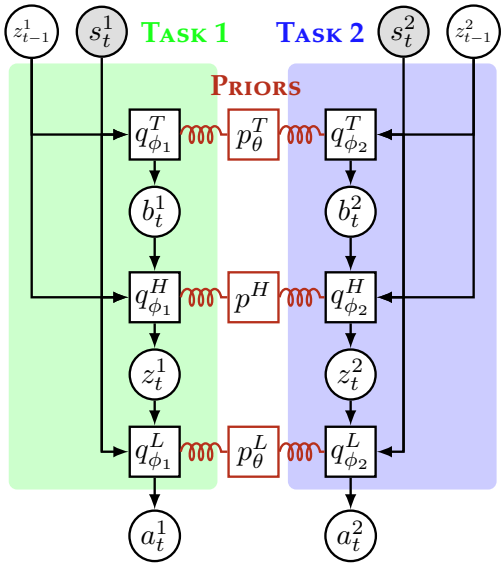


Figure 3.1: Two hierarchical posterior policies (left and right) with common priors (middle). For each task i, the policy conditions on the current state s^i_t and the last selected option z^i_{t-1}. It samples, in order, whether to terminate the last option (b^i_t), which option to execute next (z^i_t) and what primitive action (a^i_t) to execute in the environment.

maximising the reward on certain tasks is not achievable with the prior policy, the posterior is free to deviate from it. We can thus speed up training using options, while remaining flexible enough to solve more tasks. Additionally, this soft option framework also allows for learning good *priors* in a multitask setting while avoiding complex training schedules and local optima (see Section 3.4.5). In this work, we also learn the higher-level posterior $q^H_{\phi_i}$ within the framework of PAI, but assume a fixed, uniform prior distribution p^H, i.e. we assume there is no shared higher-level structure between tasks. Figure 3.1 shows an overview over this architecture which we explain further below.

3.4.1 Hierarchical Posterior Policies

To express options in the PAI framework, we introduce two additional variables at each time step t: *option selections* z_t, representing the currently selected option, and decisions b_t to *terminate* them and allow the higher-level (master) policy to choose a new option. The agent's behaviour depends on the currently selected option z_t, by

drawing actions a_t from the *intra-option posterior policy* $q^L_{\phi_i}(a_t|s_t, z_t)$. The selection z_t itself is drawn from a *master policy* $q^H_{\phi_i}(z_t|s_t, z_{t-1}, b_t) = (1-b_t)\,\delta(z_t - z_{t-1}) + b_t\, q^H_{\phi_i}(z_t|s_t)$, which conditions on $b_t \in \{0, 1\}$, drawn by the *termination posterior policy* $q^T_{\phi_i}(b_t|s_t, z_{t-1})$. The master policy either continues with the previous z_{t-1} or draws a new option, where we set $b_1 = 1$ at the beginning of each episode. We slightly abuse notation by referring by $\delta(z_t - z_{t-1})$ to the Kronecker delta $\delta_{z_t, z_{t-1}}$ for discrete and the Dirac delta distribution for continuous z_t. The joint posterior policy is

$$\begin{aligned} &q_{\phi_i}(a_t, z_t, b_t|s_t, z_{t-1}) = \\ &q^T_{\phi_i}(b_t|s_t, z_{t-1})\, q^H_{\phi_i}(z_t|s_t, z_{t-1}, b_t)\, q^L_{\phi_i}(a_t|s_t, z_t)\,. \end{aligned} \tag{3.4}$$

While z_t can be a continuous variable, we consider only $z_t \in \{1 \dots m\}$, where m is the number of available options. The induced distribution $q_{\phi_i}(\tau)$ over trajectories of task i, $\tau = (s_1, b_1, z_1, a_1, s_2, \dots, s_T, b_T, z_T, a_T, s_{T+1})$, is then

$$q_{\phi_i}(\tau) = \rho_i(s_1) \prod_{t=1}^{T} q_{\phi_i}(a_t, z_t, b_t|s_t, z_{t-1}) P_i(s_{t+1}|s_t, a_t). \tag{3.5}$$

3.4.2 Hierarchical Prior Policy

Our framework transfers knowledge between tasks by a shared prior $p_\theta(a_t, z_t, b_t|s_t, z_{t-1})$ over all joint policies (3.4):

$$\begin{aligned} &p_\theta(a_t, z_t, b_t|s_t, z_{t-1}) = \\ &p^T_\theta(b_t|s_t, z_{t-1})\, p^H(z_t|z_{t-1}, b_t)\, p^L_\theta(a_t|s_t, z_t). \end{aligned} \tag{3.6}$$

By choosing p^T_θ, p^H, and p^L_θ correctly, we can learn useful temporally extended options. The parameterised priors $p^T_\theta(b_t|s_t, z_{t-1})$ and $p^L_\theta(a_t|s_t, z_t)$ are structurally

equivalent to the posterior policies $q^T_{\phi_i}$ and $q^L_{\phi_i}$ so that they can be used as initialisation for the latter on new tasks. Optimizing the regularised return (see next section) w.r.t. θ distills the common behaviour into the prior policy and softly enforces similarity across posterior distributions of each option amongst all tasks i.

The prior $p^H(z_t|z_{t-1}, b_t) = (1 - b_t)\,\delta(z_t - z_{t-1}) + b_t \frac{1}{m}$ selects the previous option z_{t-1} if $b_t = 0$, and otherwise draws options uniformly to ensure exploration. Because the posterior master policy is different on each task, there is no need to distill common behaviour into a joint prior.

3.4.3 Objective

We extend the multitask objective in (3.3) by substituting $p_\theta(\tau, \mathcal{O}_{1:T})$ and $p_{\phi_i}(\tau)$ with those induced by our hierarchical posterior policy in (3.4) and the corresponding prior. The resulting objective has the same form but with a new regularised reward that is maximised:

$$\begin{aligned} R^{\text{reg}}_{i,t} =& r_i(s_t, a_t) - \underbrace{\beta \ln \frac{q^H_{\phi_i}(z_t|s_t, z_{t-1}, b_t)}{p^H(z_t|z_{t-1}, b_t)}}_{①} \\ & - \underbrace{\beta \ln \frac{q^L_{\phi_i}(a_t|s_t, z_t)}{p^L_\theta(a_t|s_t, z_t)}}_{②} - \underbrace{\beta \ln \frac{q^T_{\phi_i}(b_t|s_t, z_{t-1})}{p^T_\theta(b_t|s_t, z_{t-1})}}_{③} . \end{aligned} \tag{3.7}$$

As we maximise $\mathbb{E}_q[R^{\text{reg}}_{i,t}]$, this corresponds to maximising the expectation over

$$r_i(s_t, a_t) - \beta \big[\mathbb{D}_{\text{KL}}(q^H_{\phi_i} \| p^H) + \mathbb{D}_{\text{KL}}(q^L_{\phi_i} \| p^L_\theta) + \mathbb{D}_{\text{KL}}(q^T_{\phi_i} \| p^T_\theta)\big], \tag{3.8}$$

along the on-policy trajectories drawn from $q_{\phi_i}(\tau)$. In the following, we will discuss the effects of all three regularisation terms on the optimisation.

Term ① of the regularisation encourages exploration in the space of options since we chose a uniform prior for p^H when the previous option was terminated. It can

also be seen as a form of deliberation cost [89] as it is only nonzero whenever we terminate an option and the master policy needs to select another to execute: if the option is not terminated, we have $z_t = z_{t-1}$ with probability 1 for both prior and posterior by construction and $\mathbb{D}_{\text{KL}}(q^H_{\phi_i} \| p^H) = 0$.

Because Equation 3.7 is optimised across *all* tasks i, term ② updates the prior towards the 'average' posterior. It also regularises each posterior towards this prior. This enforces similarity between option posteriors across tasks. Importantly, it also encourages the *master* policy to pick the most *specialised* option that still maximises the return, i.e the option for which the posteriors $q^L_{\phi_i}$ are most similar across tasks as this will minimise term ②. Consequently, if multiple options have learned the desired behaviour, the master policy will only pick the most specialised option consistently. As discussed in Section 3.4.5, this allows us to escape the local optima to which hard options are prone in multitask learning, while still having fully specialised options after training.

Lastly, we can use ③ to also encourage temporal abstraction of options. To do so, *during option learning*, we fix the termination prior p^T to a Bernoulli distribution $p^T(b) = (1-\alpha)^b \alpha^{1-b}$. Choosing a large α encourages prolonged execution of one option, but allows switching whenever necessary. This is similar to deliberation costs [89] but with a more flexible cost model.

We can still distill a termination prior p^T_θ which can be used on future tasks. Instead of learning p^T_θ by minimizing the KL against the posterior termination policies, we can get more decisive terminations by minimizing

$$\min_\theta \sum_{i=1}^{n} \mathbb{E}_{\tau \sim q^i} \left[\mathbb{D}_{\text{KL}} \left(\hat{q}_{\phi_i}(\cdot | s_t, z_{t-1}) \| \, p^T_\theta(\cdot | s_t, z_{t-1}) \right) \right], \tag{3.9}$$

and $\hat{q}_{\phi_i}(b{=}1 | s_t, z_{t-1}) {=} \sum_{z_t \neq z_{t-1}} q^H_{\phi_i}(z_t | s_t, z_{t-1}, b_t {=} 1)$ i.e., the learned termination prior distills the probability that the tasks' master policies would change the active option if they had the opportunity. Details on how we optimised the MSOL objective are

given in Section 3.7.

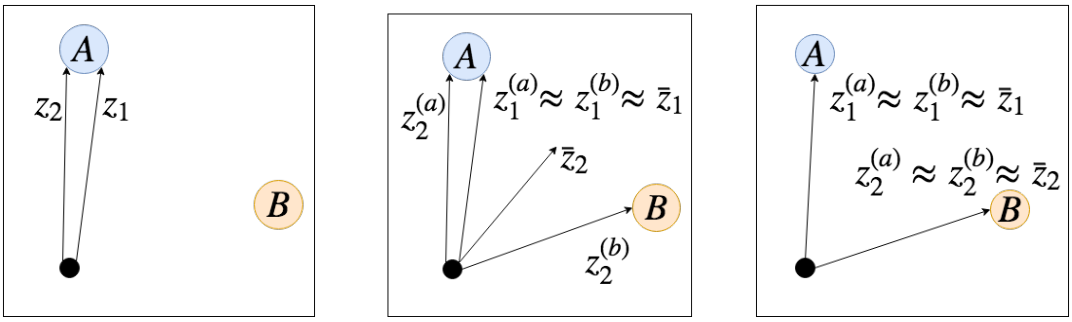
3.4.4 MSOL vs Classical Options

Assume we are faced with a new task and are given some prior knowledge in the form of a set of skills that we can use. Using the skills' policies and termination probabilities as prior policies p^T and p^L in the soft option framework, we can interpret β as a temperature parameter determining how closely we are required to follow them. For $\beta \to \infty$ we recover the classical 'hard' option case and our posterior option policies are restricted to the prior.[1] For $\beta = 0$ the priors only initialise the otherwise unconstrained policy, quickly unlearning behaviour that may be useful down the line. Only for $0 < \beta < \infty$ MSOL can keep prior information to guide long-term exploration but can also explore policies 'close' to them.

3.4.5 Local Optima Option Learning

In this section our aim is to provide an intuitive explanation of why learning hard options in a multitask setting can lead to local optima and how soft options can overcome this. In this local optimum, multiple options have learned the same behaviour and are unable to change it, even if doing so would ultimately lead to a higher reward. We use the Moving Bandits experiment schematically depicted in Figure 3.2 as an example. The agent (black dot) observes two target locations A and B but does not know which one is the correct one that has to be reached in order to generate a reward. The state- and action-spaces are continuous, requiring multiple actions to reach either A or B from the starting position. Consequently, having access to two options, one for each location, can accelerate learning. Experimental results comparing MSOL against a recently proposed 'hard option' method (MLSH, [67]) are discussed in Section 3.6.1.

[1] However, in this limiting case optimisation using the regularised reward is not possible.



(a) Hard options (b) Soft options (c) After training

Figure 3.2: Hierarchical learning of two concurrent tasks (task a = reaching target A, and task b = reaching target B) using two options (z_1 and z_2).
Figure 3.2a: Local optimum when simply sharing options across tasks.
Figure 3.2b: Soft options allow for the task-specific posterior $z_2^{(b)}$ (the second option for task b) to deviate from the local optimum through regularisation against the task-agnostic option priors $\bar{z}_i$.
Figure 3.2c: After training, the prior for option 1, as well as its task-specific posteriors for both tasks a and b lead to target A. Likewise, the prior for option 2, as well as its task-specific posteriors for both tasks a and b lead to target B. When presented with task a, the master policy can use option 1, and when presented with task b, the master policy can use option 2. These options allow for optimal return over both tasks. Details are given in Section 3.4.5.

Let us denote the options we are learning as z_1 and z_2 and further assume that due to random initialisation or late discovery of target B, both skills currently reach A. In this situation, the master policies on tasks in which the correct goal is A are indifferent between using z_1 and z_2 and will consequently use *both* with equal probability.

In the case of hard options, changing one skill, e.g. z_2, towards B in order to solve tasks in which B is the correct target, decreases the performance on all tasks that currently use z_2 to reach target A, because for hard options the skills are shared exactly across tasks. Averaged across all tasks, this would at first *decrease* the overall average return, preventing any option from changing away from A, leaving B unreachable and training stuck in a local optimum.

To 'free up' z_2 and learn a new skill reaching B, all master policies need to refrain from using z_2 to reach A and instead use the equally useful skill z_1 exclusively. Importantly, using soft options makes this possible. In Figure 3.2b and Figure 3.2c

we depict this schematically. The key difference is that in MSOL we have separate *task-specific posteriors* $z_i^{(a)}$ and $z_i^{(b)}$ for tasks a and b and soft options $i \in \{1, 2\}$ (for simplicity, we assume that the correct target is A for task a and B for task b). This allows us, in a first step, to solve *all* tasks (Figure 3.2b); despite master policies on tasks a still using posterior $z_2^{(a)}$ to reach A, the other posterior $z_2^{(b)}$ can learn to reach B. However, this now makes option z_2 less specialised across tasks, i.e. the prior $\bar{z}_2$ does not agree with either posterior $z_2^{(a)/(b)}$. Consequently, for tasks a, the master policies will now strictly prefer option z_1 to reach A, allowing option z_2 to specialise on only reaching B, leading to the situation shown in Figure 3.2c in which both options specialise to reach different targets.

3.5 Related Work

Most hierarchical approaches rely on proxy rewards to train the lower level components and their terminations. Some of them aim to reach pre-specified subgoals [186], which are often found by analyzing the structure of the MDP [144], previously learned policies [189] or predictability [90]. Those methods typically require knowledge, or a sufficient approximation, of the transition model, both of which are often infeasible.

Recently, several authors have proposed unsupervised training objectives for learning diverse skills based on their distinctiveness [85]. However, those approaches don't learn termination functions and cannot guarantee that the required behaviour on the downstream task is included in the set of learned skills. [92] also incorporate reward information, but do not learn termination policies and are therefore restricted to learning multiple solutions to the provided task instead of learning a *decomposition* of the task solutions which can be re-composed to solve new tasks.

A third usage of proxy rewards is by training lower level policies to move towards

goals defined by the higher levels. When those goals are set in the original state space [145], this approach has difficulty scaling to high dimensional state spaces like images. Setting the goals in a learned embedding space [208] can be difficult to train, though. In both cases, the temporal extension of the learned skills are set manually. On the other hand, [81] also learn a hierarchical agent, but not to transfer skills, but to find decisions states based on how much information is encoded in the latent layer.

HiREPS [45] also take an inference motivated approach to learning options. In particular [46] propose a similarly structured hierarchical policy, albeit in a single task setting. However, they do not utilise learned prior *and* posterior distributions, but instead use expectation maximisation to iteratively infer a hierarchical policy to explain the current reward-weighted trajectory distribution.

Several previous works try to overcome the restrictive nature of options that can lead to sub-optimal solutions by allowing the higher-level actions to modulate the behaviour of the lower-level policies [93, 88]. However, this significantly increases the required complexity of the higher-level policy and therefore the learning time.

The multitask- and transfer-learning setup used in this work is inspired by [191] who suggests extracting options by using commonalities between solutions to multiple tasks. Prior multitask approaches often rely on additional human supervision like policy sketches [8] or desirable sub-goals [189] in order to learn skills which transfer well between tasks. In contrast, our work aims at finding good termination states without such supervision. [193] investigate the use of different priors for the higher-level policy while we are focussing on learning transferrable option priors. Closest to our work is MLSH [67] which, however, shares the lower-level policies across all tasks without distinguishing between prior and posterior and does not learn termination policies. As discussed, this leads to local minima and insufficient diversity in the learned options. Similarly to us, [66] differentiate

between prior and posterior policies on multiple tasks and utilise a KL-divergence between them for training. However, they do not consider termination probabilities and instead only choose one option per task.

Our approach is closely related to Distral [187] with which we share the multitask learning of prior and posterior policies. However, Distral has no hierarchical structure and applies the same prior distribution over primitive actions, independent of the task. As a necessary hierarchical heuristic, the authors propose to also condition on the last primitive action taken. This works well when the last action is indicative of future behaviour; however, in Section 3.6 we show several failure cases where a *learned* hierarchy is needed.

3.6 Experiments

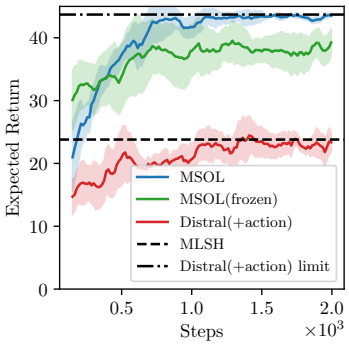


Figure 3.3: Moving Bandits

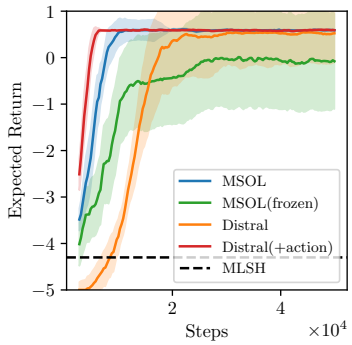


Figure 3.4: Taxi

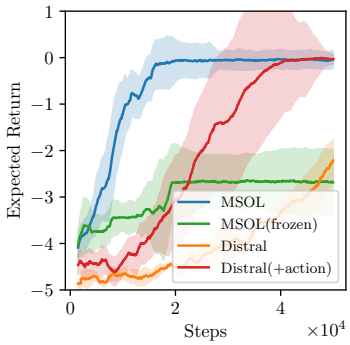


Figure 3.5: Directional Taxi

Figure 3.6: Performance of applying the learned options and exploration priors to new tasks. Each line is the median over 5 random seeds (2 for MLSH) and shaded areas indicated standard deviations. Performance during the training phase is shown in Figure 3.15. *Moving Bandits* (a) is a simple environment capturing the effects described in Section 3.4.5. The results show that MLSH, which uses hard options, struggles with local minima during the learning phase, whereas MSOL is able to learn a diverse set of options. *Taxi* (b) and *Directional Taxi* (c) additionally require good termination policies, which MLSH cannot learn as it uses a fixed option duration. See Figure 3.7 for a visualization of the options and terminations learned by MSOL. Distral(+action) is a strong non-hierarchical baseline which uses the last action as option-heuristic, but suffers when that action is not very informative, for example in (c).

We conduct a series of experiments to show: i) MSOL trains successfully without complex training schedules like in MLSH [67], ii) MSOL can learn useful termination policies, iii) when learning hierarchies in a multitask setting, unlike other methods, MSOL successfully overcomes the local minimum of insufficient option diversity, as described in Section 3.4.5, iv) using soft options yields fast transfer learning while still reaching optimal performance, even on new, out-of-distribution tasks.

Subsequently, we test how quickly we can learn new tasks from $\mathcal{T}$ (or another distribution $\mathcal{T}'$).

We compare the following algorithms: MSOL is our proposed method that utilises soft options both during option learning and transfer. MSOL(frozen) uses the soft options framework during learning to find more diverse skills, but does not allow fine-tuning the posterior sub-policies after transfer. Distral [187] is a strong non-hierarchical transfer learning algorithm that also utilises prior and posterior distributions. Distral(+action) utilises the last action as option-heuristic, that is, as additional input to the policy and prior, which works well in some tasks but fails when the last action is not sufficiently informative. Conditioning on an *informative* last action allows the Distral prior to learn temporally correlated exploration strategies. MLSH [67] is a multitask option learning algorithm like MSOL, but utilises 'hard' options for both learning and transfer, i.e., sub-policies that are shared exactly across tasks. It relies on fixed option durations and requires a complex training schedule between master and intra-option policies to stabilise training. We use the author's MLSH implementation. We also compare against OC [11], which takes the task-id as additional input in order to apply it to multiple tasks.

Note that, during test time, MLSH and MSOL(frozen) can be fairly compared as each uses one fixed policy per skill. On the other hand, Distral, Distral(+action) and MSOL use adaptive posterior policies for each task and are consequently more expressive.

3.6.1 Moving Bandits

We start with the 2D Moving Bandits environment proposed and implemented by [67], which is similar to the example in Section 3.4.5. There are two randomly sampled, distinguishable, marked positions in the environment. In each episode, the agent receives a reward of 1 for each time step it is sufficiently close to the correct one of both positions, and 0 otherwise. Which location is rewarded is not signaled in the observation. The agent can take actions that move it in one of the four cardinal directions. Each episode lasts 50 steps.

We compare against MLSH and Distral to highlight challenges that arise in multitask training. We allow MLSH and MSOL to learn two options. During transfer, optimal performance can only be achieved with diverse options that have successfully learned to reach *different* marked locations. In Figure 3.3 we can see that MSOL is able to do so but the hard options learned by MLSH both learned to reach the *same* goal location, resulting in only approximately half the optimal return during transfer. This is exactly the situation outlined in Section 3.4.5 in which learning hard options can lead to local optima.

Distral, even with the last action provided as additional input, is not able to quickly utilise the prior knowledge. The last action only conveys meaningful information when taking the goal locations into account: Distral agents need to *infer* the intention based on the last action and the relative goal positions. While this is possible, in practice the agent was not able to do so, even with a much larger network. Much longer training ultimately allows Distral to perform as well as MSOL, denoted by 'Distral(+action) limit'. This is not surprising since its posterior is flexible and will therefore eventually be able to learn any task. However, it is not able to learn transferrable prior knowledge which allows *fast* training on the new task. Lastly, MSOL(frozen) also outperforms Distral(+action) and MLSH, but performs worse that MSOL. This highlights the utility of making options soft, i.e.

adaptable, during transfer to new tasks. It also shows that the advantage of MSOL over the other methods lies not only in its flexibility during transfer, but also during the original learning phase.

3.6.2 Taxi

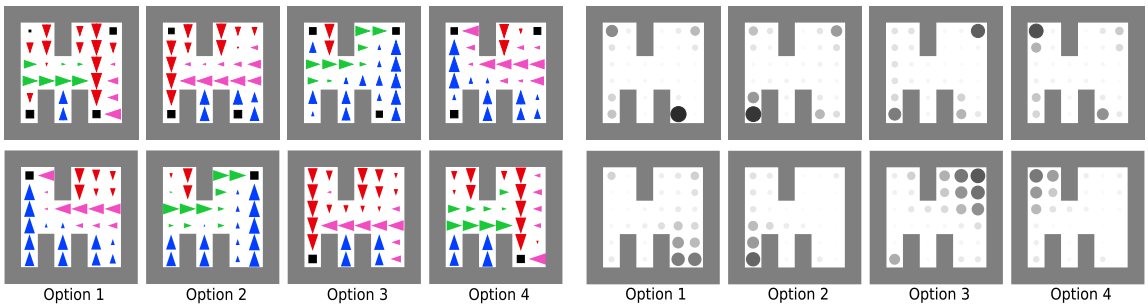


Figure 3.7: Four options learned with MSOL on the taxi domain, before (top) and after pickup (bottom). The light gray area indicates walls. The left plots show the intra-option policies: arrows and colors indicated direction of most likely action, the size indicates its probability. A square indicates the pickup/dropoff action. The right plots show the termination policies: intensity and size of the circles indicate termination probability.
For instance, given a pickup task in the lower right-hand corner and a dropoff task in the lower left-hand corner, MSOL could use option 1 to reach the pickup location and then execute the pickup action. At this point option 1 would terminate with high probability and option 3 could be used to solve the MDP by navigating the agent to the lower left-hand corner and executing the dropoff action. It is encouraging that the four options learned by MSOL correspond to the eight possible tasks in the Taxi environment (picking up or dropping off in any corner), as this indicates that MSOL is learning a diverse set of options that can obtain optimal reward.

Next, we use a slightly modified version of the original Taxi domain [53] to show learning of termination functions as well as transfer- and generalisation capabilities. To solve the task, the agent must pick up a passenger on one of four possible locations by moving to their location and executing a special 'pickup/drop-off' action. Then, the passenger must be dropped off at one of the other three locations, again using the same action executed at the corresponding location. The domain has a discrete state space with 30 locations arranged on a grid and a flag indicating whether the passenger was already picked up. The observation is a one-hot encoding of

the discrete state, excluding passenger- and goal location. This introduces an information-asymmetry between the task-specific master policy, and the shared options, allowing them to generalise well [71]. Walls (see Figure 3.7) limit the movement of the agent and invalid actions.

We investigate two versions of Taxi. In the original, just called *Taxi*, the action space consists of one no-op, one 'pickup/drop-off' action and four actions to move in all cardinal directions. In *Directional Taxi*, we extend this setup: the agent faces in one of the cardinal directions and the available movements are to move forward or rotate either clockwise or counter-clockwise. In both environments the set of tasks $\mathcal{T}$ are the 12 different combinations of pickup/drop-off locations. Episodes last at most 50 steps and there is a reward of 2 for delivering the passenger to its goal and a penalty of -0.1 for each time step. During training, the agent is initialised to any valid state. During testing, the agent is always initialised without the passenger on board.

We allow four learnable options in MLSH and MSOL. This necessitates the options to be diverse, i.e., one option to reach each of the four pickup/drop-off locations. Importantly, it also requires the options to learn to terminate when a passenger is picked up. As one can see in Figure 3.4, MLSH struggles both with option-diversity and due to its fixed option duration: because the starting position is random, the duration until the option needs to terminate is different between episodes and cannot be captured by one hyperparameter. Furthermore, even without correct terminations, one could still learn to solve (at least) four out of the twelve tasks, leading to an average reward of approximately -3.2[2]. However, MLSH is not able to learn diverse enough policies, resulting in worse performance.

Distral(+action) performs well in the original *Taxi* environment, as seen in Figure 3.4. This is expected since here the last action, moving in a compass direction,

[2]The optimal policy for a task achieves approximate a return of 0.5 on average whereas the worst possible return is -5.

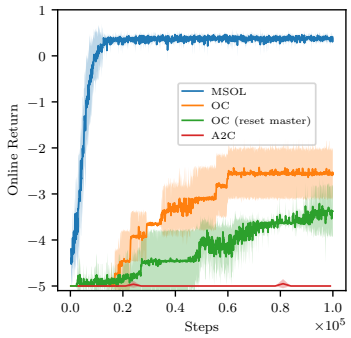


Figure 3.8: Taxi: Generalization

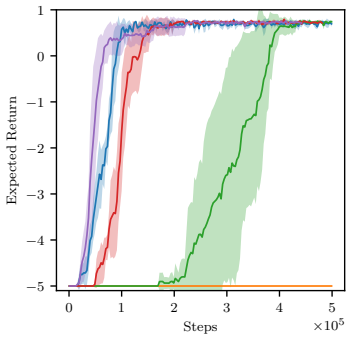


Figure 3.9: Taxi (small): Adaptation

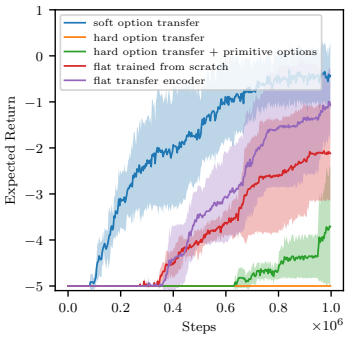


Figure 3.10: Taxi (large): Adaptation

Figure 3.11: We compare MSOL against Option-Critic, hard options and flat policies trained from scratch or with a pre-trained encoder. For a fair comparison, the soft option prior is identical to the hard option in these experiments. *Left:* Since the options in OC are not task-agnostic, they fail to generalise to previously unseen tasks. *Middle and right:* Transfer performance of options to environments in which the pickup and dropoff locations where shifted, making the options misspecified. Only soft options provide utility over flat policies in this setting. The middle figure shows results on a small grid in which exploration is simple, whereas the right figure shows that transfer learning can accelerate exploration especially on larger tasks.

is a good indicator for the agent's intention, effectively acting as an optimal 'option' and inducing temporally extended exploration. However, in the directional case shown in Figure 3.5, actions rarely indicate intentions, which makes it much harder for Distral(+action) to use prior knowledge. By contrast, MSOL performs well in both taxi environments. In the directional case, learned MSOL options capture temporally correlated behaviour much better than the last action in Distral.

Figure 3.7 demonstrates that the options learned by MSOL learn movement and termination policies that make intuitive sense. Note that the same soft option represents different behaviour depending on whether it already picked up the passenger, as this behaviour does not need to terminate the current option on three of the 12 tasks.

3.6.3 Out-of-distribution Tasks

In this section, we show how learning soft options can help with transfer to unseen tasks. In Figure 3.8 we show learning on four tasks from $\mathcal{T}$ using options that were trained on the remaining eight, comparing against A2C [141] and OC [11]. Note that in OC, there is no information-asymmetry: the same networks are shared across all tasks and provided with a task-id as additional input, including to the option-policies. This prevents OC from generalising well to unseen tasks. On the other hand, withholding the task-information would be similar to MLSH, which we already showed to struggle with local minima. The strong performance of MSOL shows that information-asymmetric options help to generalise to previously unseen tasks.

We also investigate the utility of flexible soft options under a shift of the task distribution: in Figure 3.9 and Figure 3.10 we show learning performance on twelve *modified* tasks in which the pickup/dropoff locations where moved by one cell while the options were trained with the original locations. While the results in Figure 3.9 use a smaller grid, Figure 3.10 shows the results for a larger grid in which exploration is more difficult. As expected, hard options are not able to solve this task for either grid-size. Moreover, while combining hard options with primitive actions allows the tasks to be solved eventually, it performs worse than training a new, flat policy from scratch. The finding that access to misspecified, hard options can actually *hurt* exploration is consistent with previous literature [106]. On the other hand, MSOL is able to quickly learn on this new task by adapting the previously learned options.

Note that on the small grid in which exploration is easy, our hierarchical method performs similar to a flat policy. On the larger grid exploration becomes more challenging and MSOL learns significantly faster, highlighting how transfer learning can improve exploration.

In Figure 3.16 we show the performance on *further modified* environments, for which the goal locations were moved by a second block from the original location for which the options were trained. We only compare soft options with flat policies trained from scratch, as we already showed that hard options are unable to cope well with goal modifications.

As expected, the flat policy trained from scratch performs similarly as before, as the moved goal location does not impact it much. On the other hand, using a pre-trained encoder performs slightly worse. On the smaller task (left figure) the options are too misspecified to be competitive, despite being soft. For the larger grid (right figure) and for a sufficiently small value of β ('KLC'), the options, despite misspecification, are still competitive.

Consequently, while there is no hard limitation of our approach for appropriately chosen β, if the target task is too different from the source task, it will be faster to learn a new policy from scratch. Which algorithm trains faster depends mainly on the difficulty of exploration in the target task. Hard exploration makes options more useful compared to a new, flat policy, even if the options are misspecified. However, the more misspecified the options are, the smaller the advantage. If target and source task are too different, very little positive transfer can be expected, and learning a new (flat) policy becomes more efficient.

3.7 Training Details

3.7.1 Optimisation

Even though R_i^{reg} depends on ϕ_i, its gradient w.r.t. ϕ_i vanishes.[3] Consequently, we can treat the regularised reward as a classical reinforcement learning reward and use any reinforcement learning algorithm to find the optimal hierarchical policy

[3] $\int p(x) \nabla \ln p(x)\, dx = \int \nabla p(x)\, dx = \nabla \int p(x)\, dx = 0.$

parameters ϕ_i. In the following, we explain how to adapt A2C [141] to soft options. The extension to PPO [175] is straightforward.[4]

The joint posterior policy in (3.4) depends on the current state s_t and the previously selected option z_{t-1}. The expected sum of regularised future rewards of task i, the value function V_i, must therefore also condition on this pair:

$$V_i(s_t, z_{t-1}) \;:=\; \mathbb{E}_{\tau\sim q}\Big[\sum_{t'=t}^{T}\gamma^{t'-t}R^{\text{reg}}_{i,t'}\;\Big|\; s_t, z_{t-1}\Big]\,. \tag{3.10}$$

As $V_i(s_t, z_{t-1})$ cannot be directly observed, we approximate it with a parameterised model $V_{\phi_i}(s_t, z_{t-1})$. The k-step advantage estimation at time t of trajectory τ is given by

$$A_{\phi_i}(\tau_{t:(t+k)}) \quad := \quad \sum_{j=0}^{k-1}\gamma^j R^{\text{reg}}_{t+j} + \gamma^k V^{-}_{\phi_i}(s_{t+k}, z_{t+k-1}) - V_{\phi_i}(s_t, z_{t-1})\,, \tag{3.11}$$

where the superscript '$-$' indicates treating the term as a constant. The approximate value function V_{ϕ_i} can be optimised towards its bootstrapped k-step target by minimizing $\mathcal{L}_V(\phi_i, \tau_{1:T}) := \sum_{t=1}^{T}(A_{\phi_i}(\tau_{t:(t+k)}))^2$. As per A2C, $k \in [1 \dots n_s]$ depending on the state [141]. The corresponding policy gradient loss is

$$\mathcal{L}_A(\phi_i, \tau_{1:T}) := \sum_{t=1}^{T} A^{-}_{\phi_i}(\tau_{t:(t+k)}) \ln q_{\phi_i}(a_t, z_t, b_t | s_t, z_{t-1})\,.$$

The gradient w.r.t. the prior parameters θ is[5]

$$\nabla_\theta \mathcal{L}_P(\theta, \tau_{1:T}, \tilde{b}_{1:T}) \quad := \quad -\sum_{t=1}^{T}\Big(\nabla_\theta \ln p^L_\theta(a_t|s_t, z_t) + \nabla_\theta \ln p^T_\theta(\tilde{b}_t|s_t, z_{t-1})\Big)\,, \tag{3.12}$$

where $\tilde{b}_t = \delta_{z_{t-1}}(z'_t)$ and $z'_t \sim q^H(z'_t|s_t, z_{t-1}, b_t = 1)$. To encourage exploration in all

[4] However, for PAI frameworks like ours, unlike in the original PPO implementation, the advantage function must be updated after each epoch.

[5] Here we ignore β as it is folded into λ_P later.

policies of the hierarchy, we also include an entropy maximisation loss:

$$\mathcal{L}_H(\phi_i, \tau_{1:T}) \quad := \quad \sum_{t=1}^{T} \Big(\ln q^H_{\phi_i}(z_t|s_t, z_{t-1}, b_t) + \ln q^L_{\phi_i}(a_t|s_t, z_t) + \ln q^T_{\phi_i}(b_t|s_t, z_{t-1}) \Big) . \tag{3.13}$$

Note that term ① in (3.7) already encourages maximising $\mathcal{L}_H(\phi_i, \tau)$ for the master policy, since we chose a uniform prior $p^H(z_t|b_t = 1)$. As both terms serve the same purpose, we are free to drop either one of them. In our experiments, we chose to drop the term for q^H in R^{reg}_t, which proved slightly more stable to optimise that the alternative.

We can optimise all parameters jointly with a combined loss over all tasks i, based on sampled trajectories $\tau^i := \tau^i_{1:T} \sim q_{\phi_i}$ and corresponding sampled values of $\tilde{b}^i := \tilde{b}^i_{1:T}$:

$$\mathcal{L}(\{\phi_i\}, \theta, \{\tau^i\}, \{\tilde{b}^i\}) = \sum_{i=1}^{n} \Big(\mathcal{L}_A(\phi_i, \tau^i) + \lambda_V \mathcal{L}_V(\phi_i, \tau^i) + \lambda_P \mathcal{L}_P(\theta, \tau^i, \tilde{b}^i) + \lambda_H \mathcal{L}_H(\phi_i, \tau^i) \Big) .$$

3.7.2 Training Schedule

For faster training, it is important to prevent the master policies q^H from converging too quickly to allow sufficient updating of all options. On the other hand, a lower exploration rate leads to more clearly defined options. We consequently anneal the exploration bonus λ_H with a linear schedule during training.

Similarly, a high value of β leads to better options but can prevent finding the extrinsic reward $r_i(s_t, a_t)$ early on in training. Consequently, we increase β over the course of training, also using a linear schedule.

3.8 Discussion

Multi-task Soft Option Learning (MSOL) proposes reformulating options using the perspective of prior and posterior distributions. This offers several key advantages.

First, during transfer, it allows us to distinguish between fixed, and therefore knowledge-preserving option *priors*, and flexible option *posteriors* that can adjust to the reward structure of the task at hand. This effects a similar speed-up in learning as the original options framework, while avoiding sub-optimal performance when the available options are not perfectly aligned to the task. Second, utilising this 'soft' version of options in a multitask learning setup increases optimisation stability and removes the need for complex training schedules. Furthermore, this framework naturally allows master policies to coordinate across tasks and avoid local minima of insufficient option diversity. It also allows for autonomously learning option-termination policies, a very challenging task which is often avoided by fixing option durations manually.

Lastly, using this formulation also allows inclusion of prior information in a principled manner without imposing too rigid a structure on the resulting hierarchy. We utilise this advantage to explicitly incorporate the bias that good options should be temporally extended.

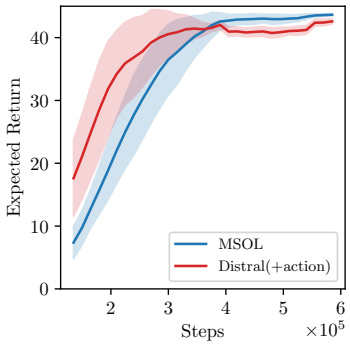


Figure 3.12: Moving Bandits

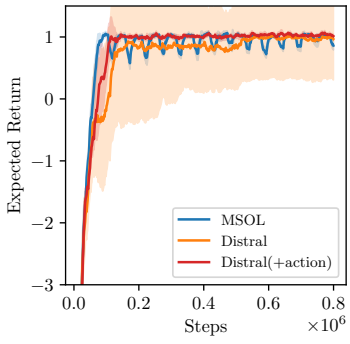


Figure 3.13: Taxi

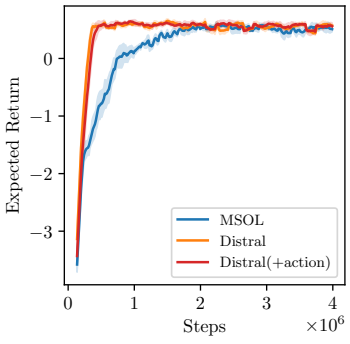


Figure 3.14: Directional Taxi

Figure 3.15: Performance during training phase. Note that MSOL and MSOL(frozen) share the same training as they only differ during testing. Further, note that the highest achievable performance for Taxi and Directional Taxi is higher during training as they can be initialised closer to the final goal (i.e. with the passenger on board).

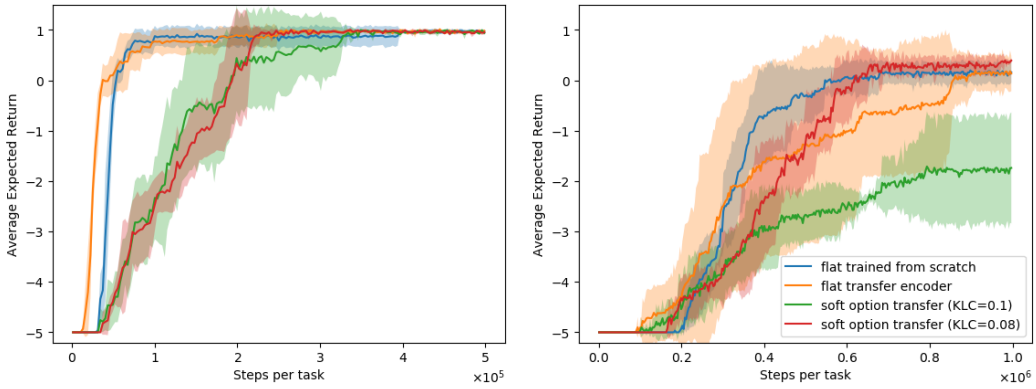


Figure 3.16: Results on a 'further modified' taxi environment in which the goal locations at test time were shifted compared to training, making the learned options misspecified, similar to Figure 3.9 and Figure 3.10. Here, the goal locations were shifted further, making the options more misspecified. *Left:* Results on a 'small' 8x8 grid. *Right:* Results on a 'large' 10x10 grid.

Chapter 4

Transfer Learning in Generative Models

5.135 In no way can an inference be made from the existence of one state of affairs to the existence of another entirely different from it.

5.136 There is no causal nexus which justifies such an inference.

Tractatus Logico-Philosophicus
Ludwig Josef Johann Wittgenstein [222]

4.1 Overview

The ultimate goal of transfer learning is to provide methods that can allow the transfer of knowledge with minimal computation required. While our method introduced in Chapter 3 accomplished the goal of allowing transfer between hierarchical reinforcement learning tasks, it did so in a very computationally intensive fashion, both during training and when attempting transfer to new tasks.

Within the generative modeling literature, recently several architectures have been proposed that combine deep generative models with data in order to learn modifications to the new data, such as making an image look as though it were from a different class or painted in a certain style. These methods, while being

able to transfer these styles effectively, also typically require large amounts of training in order to learn a single class of manipulations. Much less common is the idea of combining generative models with data in order to output a modified generative model. In this chapter we present Transflow Learning, a method for transforming a pre-trained, invertible generative model into a conditional generative model, so that its outputs more closely resemble data that we provide afterwards. TransFlow Learning provides an efficient and easy to use method for transfer learning within these generative models which can be used to sample from the conditional distribution, or solve downstream tasks.

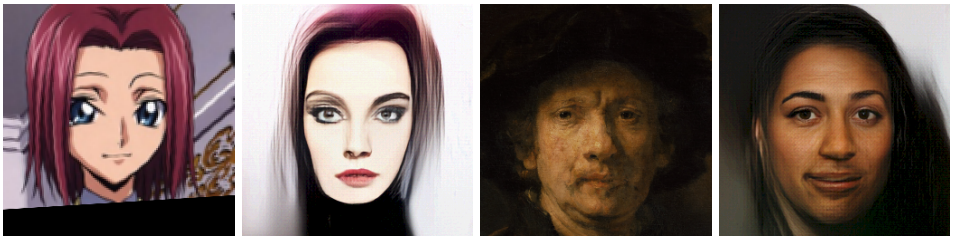


Figure 4.1: Transflow Learning allows us to warp the latent distribution of any trained invertible generative model, so that we can instead sample data similar to that we provide post-hoc. This works by treating the latent distribution as the prior in a Bayesian posterior inference setting where we condition on the provided data. In this example, given only 18–24 instances of images in the art domains on the left of each pair, a flow model trained on CelebA [127] is able to generate human faces with matching attributes, even though these attributes, such as pink hair or monotone skin colour, are not contained in the CelebA dataset.

4.2 Introduction

One important way in which transfer learning techniques are applied is in the subfield of few-shot learning [206, 120]. Whereas a human is capable of learning a task such as handwritten digit recognition after only having seen a few samples of each digit, even simple machine learning classifiers require training multiple epochs over a relatively large dataset. When it comes to more complicated tasks,

machine learning algorithms become even more data-hungry—whereas humans can learn to play Atari games in a matter of minutes, even the most sample-efficient of reinforcement learning algorithms take hundreds of hours of gameplay [200].

What advantages do humans have over machines that allows us to consistently beat them with regards to sample efficiency? One might argue that humans are constantly utilizing their experiences in other domains in order to draw parallels between tasks. Even when it comes to very disparate sets of tasks, such as 'natural language understanding' and 'video game playing,' studies have shown that it is possible to transfer knowledge between these domains for major sample efficiency gains in both machine learning algorithms and human learning [200, 28]. The idea of taking knowledge from one domain or task, and using that knowledge in another domain, is known as 'transfer learning' [151].

When attempting to transfer knowledge about one task to another, two broad questions must be asked: how is the prior knowledge from other tasks stored, and how can it be used? We propose an answer to these questions in the context of generative models.

Modern generative models, such as Generative Adversarial Networks (GANs) [79], normalizing flow models [161], and autoregressive models such as Conditional PixelCNN [205] differ in their learning mechanisms, but all share a common thread: they learn a function $f_{\boldsymbol{\theta}} : \mathbb{R}^d \to \mathbb{R}^d$ which transforms samples of a latent random variable $\mathbf{z} \sim q(\mathbf{z})$, where $\mathbf{z} \in \mathbb{R}^d$ and $q(\mathbf{z})$ is a known distribution, to a data point (e.g., an image) $\mathbf{x} \in \mathbb{R}^d$. The latent distribution $q(\mathbf{z})$ is usually a simple distribution such as the multivariate Gaussian, $\mathcal{N}(\vec{0}, I)$. Typical generative models keep $q(\mathbf{z})$ fixed, and concentrate efforts on optimizing the parameters $\boldsymbol{\theta}$.

In the context of above discussion on transfer learning, however, we take a different view. We wish to transfer knowledge from a trained generative model f, to some new task. To be concrete, assume we have a generative model that will

output an arbitrary celebrity face when given a latent vector $\mathbf{z} \sim \mathcal{N}(\vec{0}, I)$ as input. Can we use the same generative model to only output celebrities with red hair? Can we output a celebrity which looks like an anime character? Can we even classify handwritten digits?

In this chapter we will demonstrate that the above is possible, with the condition that our generative model f is *invertible*. That is, we require an inverse function $\mathbf{z} = f^{-1}(\mathbf{x})$ which takes a data point (e.g., an image) as input, and outputs the corresponding latent vector. Normalizing flow models [161] are the most natural class of such generative models, as they are by nature invertible, unlike other architectures such as GANs which can be inverted only under specific circumstances [56].

In order to achieve transfer learning without retraining the given flow model, our method treats the flow model and its parameters as fixed, and instead modifies the parameters of the latent distribution. Specifically, we treat the latent distribution $q(\mathbf{z})$ as a prior distribution in a Bayesian inference setting where we update it to a posterior $q(\mathbf{z}|\boldsymbol{\zeta}_{1:m})$ conditioned on some observed data samples $\mathbf{x}_{1:m}$ mapped to corresponding latent vectors using $\boldsymbol{\zeta}_i = f^{-1}(\mathbf{x}_i), i = 1, \ldots, m$. We call our method Transflow Learning, as it uses flow models to perform tasks for which they were not originally trained.

In Section 4.3 we cover some essential concepts, followed by Section 4.4 where we describe the mechanism by which we warp $q(\mathbf{z})$: Bayesian inference in the latent space. Section 4.5 covers related work. In Section 4.6, we provide example use cases in which knowledge can be transferred, specifically modeling distributions other than the training data, and solving downstream tasks. Finally we discuss future work and implications of our current work in Section 4.7.

4.3 Preliminaries

4.3.1 Normalizing Flow Models

We discuss flow models in detail in Section 2.4.1.4. To recap, a normalizing flow is a series of learned invertible transformations which can transform one probability distribution into another. If random variable $\mathbf{z}_0$ with associated probability density $q_0(\mathbf{z}_0)$ is put through a series of invertible transformations $\{f_1, ..., f_K\}$ so that

$$\mathbf{z}_K = f_K \circ ... \circ f_1(\mathbf{z}_0) = f(\mathbf{z}_0) \tag{4.1}$$

then we have

$$q_K(\mathbf{z}_K) = q_0(\mathbf{z}_0) \prod_{k=1}^{K} \left| \det \frac{\partial f_k}{\partial \mathbf{z}_{k-1}} \right|^{-1} . \tag{4.2}$$

Each f_k contains learnable parameters, which are typically learned by maximum likelihood. The flow is 'normalizing' because each $q_k(\mathbf{z}_k)$ defines a valid probability distribution [161]. Here we designate $\mathbf{z} = \mathbf{z}_0$ as the latent vector and $\mathbf{x} = \mathbf{z}_K$ as the data point produced by the transformation $\mathbf{x} = f(\mathbf{z})$.

4.3.2 Posterior Inference

Bayesian inference is a powerful tool for reasoning about probability distributions [31]. Core to the idea of Bayesian inference are the concepts of the *prior*, which is a probability distribution $p(z)$ that we assume exists before having seen any data, and the *posterior*, which is a probability distribution $p(z|x)$ that we obtain after having observed some data (or evidence) x with a *likelihood* $p(x|z)$. These are related with the relationship $p(z|x) = p(x|z)\, p(z)/p(x)$. The likelihood is essentially a weighting that tells us to what degree the prior must be moved after having observed some

Algorithm 4 Transflow Learning

1: **Input:** Trained flow model $\mathbf{x} = f(\mathbf{z})$ with inverse $\mathbf{z} = f^{-1}(\mathbf{x})$, where $\mathbf{x}$ are data and $\mathbf{z}$ are latents
2: For each set of evidence data $\mathbf{x}_{1:m}$, find corresponding latents $\{\boldsymbol{\zeta}_1, ..., \boldsymbol{\zeta}_m\} = \{f^{-1}(\mathbf{x}_1), ..., f^{-1}(\mathbf{x}_m)\}$
3: Construct posterior $q(\mathbf{z}|\boldsymbol{\zeta}_{1:m}) = \mathcal{N}(\mu_p, \boldsymbol{\Sigma}_p)$ by computing $\boldsymbol{\mu}_p$ and $\boldsymbol{\Sigma}_p$ analytically
4: Obtain samples from the data posterior $p(\mathbf{x}|\mathbf{x}_{1:m})$ by computing $\mathbf{x} = f(\mathbf{z})$ where $\mathbf{z} \sim q(\mathbf{z}|\boldsymbol{\zeta}_{1:m})$

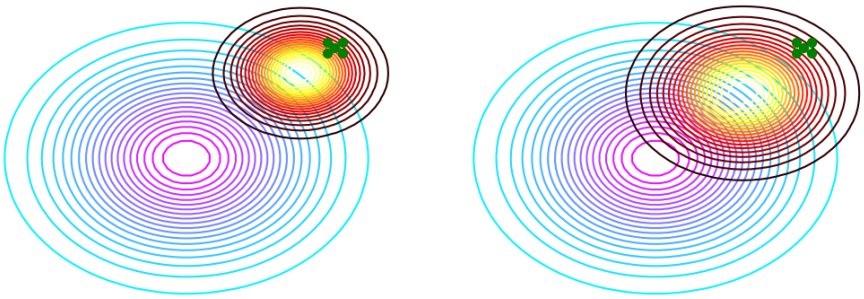
evidence.

Also important is the concept of a *conjugate prior* [157], which means that with certain choices of prior and likelihood distributions, we can ensure that we have a closed-form analytical expression for the posterior distribution. In particular, we will need to use the fact that if we have a prior which is a multivariate Gaussian and a likelihood function which is a multivariate Gaussian with a known covariance matrix, then the posterior is also guaranteed to be a multivariate Gaussian.

Using knowledge of conjugate distributions is particularly attractive, as it will allow us to solve for the posterior parameters analytically. Without a certain specification of likelihood function, we would need to resort to sampling-based methods such as stochastic variational inference [100] in order to obtain an approximation to the posterior.

4.4 Algorithm

Our algorithm is detailed in Algorithm 4. The key insight is to treat the underlying latent distribution $q(\mathbf{z})$ of a given flow model as a prior, where usually $q(\mathbf{z}) = \mathcal{N}(\vec{0}, I)$. We are then interested in obtaining a posterior over the latent space of the flow model $q(\mathbf{z}|\boldsymbol{\zeta}_{1:m})$, conditioned on $\boldsymbol{\zeta}_i$, which are some data observations $\mathbf{x}_i$ mapped to the latent space using $\boldsymbol{\zeta}_i = f^{-1}(\mathbf{x}_i), i = 1, \ldots, m$.
In other words, we provide evidence in the form of new latent vectors and, con-



$\lambda = 0.3m$ $\lambda = 0.7m$

Figure 4.2: Transflow Learning finds a posterior (warm colours) in between the prior (cool colours) and the evidence (green points). We see that as λ becomes larger, the mean of the posterior becomes closer to the mean of the prior, and the covariance of the posterior becomes larger, but in both cases the evidence can be sampled with relatively high probability. As the found posterior is as close to the centre of the prior as possible while also allowing the evidence to be sampled with high probability, samples from the posterior will appear to contain elements of both the prior and the evidence.

ditioned on this evidence, we find a posterior distribution over the flow model's latent variables. This effectively gives us a new generative model from which we can sample data resembling the evidence. In order to accomplish this, we require our generative model f to be invertible.

4.4.1 Computing the Posterior over Latent Vectors

As most implementations of normalizing flow models use a multivariate Gaussian to model $q(\mathbf{z})$, with an appropriate choice of likelihood function, we can compute the posterior analytically. Assume we are given a trained flow model, and that during training the model uses the prior $\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) = \mathcal{N}(\vec{0}, I)$. We assume a multivariate Gaussian likelihood function $p(\boldsymbol{\zeta}|\boldsymbol{z}) \sim \mathcal{N}(\mathbf{z}, \boldsymbol{\Sigma})$ for observations $\{i, ..., m\}$, which gives the effect of taking repeated noisy measurements of the observed latent variables $\boldsymbol{z}$. Under this setup, the posterior over the latent vectors is also a multivariate Gaussian, so that $q(\mathbf{z}|\boldsymbol{\zeta}_{1:m}) = p(\boldsymbol{\zeta}_{1:m}|\boldsymbol{z})p(\boldsymbol{z})/p(\boldsymbol{\zeta}_{1:m}) = \mathcal{N}(\boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p)$.

The parameters of this distribution can be solved for analytically and are given by the formulae:

$$\boldsymbol{\mu}_p = \left(\boldsymbol{\Sigma}_0^{-1} + m\boldsymbol{\Sigma}^{-1}\right)^{-1}\left(\boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu}_0 + m\boldsymbol{\Sigma}^{-1}\bar{\boldsymbol{\zeta}}\right) \tag{4.3}$$

$$\boldsymbol{\Sigma}_p = \left(\boldsymbol{\Sigma}_0^{-1} + m\boldsymbol{\Sigma}^{-1}\right)^{-1} \tag{4.4}$$

where m is the number of observed data points, $\bar{\boldsymbol{\zeta}}$ is the mean of observed latent vectors $\boldsymbol{\zeta}_{1:m}$, $\boldsymbol{\mu}_0$ is the prior mean and $\boldsymbol{\Sigma}_0$ is the prior covariance matrix. As we know that $\boldsymbol{\mu}_0$ is equal to $\vec{0}$ and $\boldsymbol{\Sigma}_0$ is the identity matrix, we can further simplify these formulae:

$$\boldsymbol{\mu}_p = \left(I + m\boldsymbol{\Sigma}^{-1}\right)^{-1}\left(m\boldsymbol{\Sigma}^{-1}\bar{\boldsymbol{\zeta}}\right) \tag{4.5}$$

$$\boldsymbol{\Sigma}_p = \left(I + m\boldsymbol{\Sigma}^{-1}\right)^{-1} \tag{4.6}$$

The choice of $\boldsymbol{\Sigma}$ here serves as a hyperparameter. A full covariance matrix in a latent space with dimension d would have d^2 entries, therefore using a non-sparse covariance matrix would cause significant computational difficulty. For a flow model trained on full-colour, 256 by 256 images, a full covariance matrix would contain more than 38 billion entries.

A natural choice for the Σ hyperparameter would be a scalar matrix, which implies that we would like to keep the latent dimensions uncorrelated and weighted identically. With likelihood covariance $\boldsymbol{\Sigma}$ being set to λI, with λ a scaling hyperparameter, the posterior parameters become simple to compute:

$$\boldsymbol{\mu}_p = \left(I + \frac{m}{\lambda}I\right)^{-1}\left(\frac{m}{\lambda}I\bar{\boldsymbol{\zeta}}\right) = \frac{\frac{m}{\lambda}\bar{\boldsymbol{\zeta}}}{\frac{m}{\lambda}+1} \tag{4.7}$$

$$\boldsymbol{\Sigma}_p = \left(I + \frac{m}{\lambda}I\right)^{-1} = \frac{1}{\frac{m}{\lambda}+1}I \tag{4.8}$$

There are three special cases that we can examine:

1. If we let $\lambda = c$, where $c \in \mathbb{R}$ is a small constant relative to $m \in \mathbb{N}$, then the posterior mean will be close to the sample mean, and the posterior covariance will be very small.

2. If we let $\lambda = m$, then the posterior mean will be locked onto $0.5\bar{\boldsymbol{\zeta}}$, and the posterior covariance will remain constant at $0.5I$, regardless the value of m.

3. If we let λ become arbitrarily large, then the posterior mean will be close to $\vec{0}$ and the posterior covariance will be close to I, i.e., all conditioning is completely ignored and we get back the original flow model.

In other words, low values of λ relative to m will give a posterior that is close to the sample mean, and with very low covariance, whereas high values of λ will become more and more like the original flow model.

4.4.2 Properly Setting λ

The hyperparameter λ determines the variance of the likelihood that is used in the conditioning on observed data points. This is similar to the use of approximate Bayesian computation [134, 219] likelihoods in Bayesian inverse graphics [133], where the variance of the likelihood plays the role of a 'tolerance' in judging how closely an image generated by the generative model matches an observed image. High tolerances admit generation of images that do not closely match the observation, whereas low tolerances push inference towards closely mimicking the observation while reducing the sample efficiency in complex image settings.

From the perspective of the analytical posteriors introduced in Section 4.4.1, while setting λ to a high value may seem like a mistake due to the behaviour of the posterior as λ grows larger, we argue that low values of λ will also produce undesirable behaviour. As flow models learn invertible maps, the dimensionality of the latent vectors must be equal to that of the output. For example, if we wish to

output full-colour, 256 by 256 images, then the dimensionality of the latent space is $3 \times 256 \times 256 = 196,608$. In contrast, the dimensionality of the latent space for a typical GAN [79] or VAE [114], which do not have this restriction, is around 100.

The high dimensionality of flow model latent vectors implies that vectors which should be 'close' in that they share similar features in image space will be very far in the L_2 sense. This has implications for the sample mean of latent vectors, $\bar{\boldsymbol{\zeta}}$, which will have a smaller L_2 norm as more observed data points are averaged, due to the curse of dimensionality spreading supposedly 'similar' vectors in different directions relative to the origin. As vectors with smaller norm are closer to the mean of Gaussian on which the flow model was trained, $\vec{0}$, this has the effect that conditioning on many data points will give a posterior mean which is very 'generic,' as it has unreasonably high likelihood under the original model (i.e., much higher likelihood than a vector randomly sampled from $\mathcal{N}(\vec{0}, I)$). While this is not bad in and of itself, this issue is further compounded by the covariance of the posterior shrinking as more data points are added, making it so that if λ is set too low, every sample from the distribution is too close to the mode. Even though the mean of the posterior distribution has even smaller norm with larger values of λ, the larger covariance makes up for it. The effect of setting λ to be too small or too large can be seen in Figures 4.6 and 4.7.

In practice we find that a large range of settings of λ work, depending on the nature of the conditioning, but in general values which are in the range $[0.3m, 0.7m]$ are preferable. We explore the consequences of different choices of λ in Section 4.6.

4.5 Related Work

Image2StyleGAN [1] explored interpolations and embeddings of real images into the latent space of a GAN [79]. They found that while able to embed natural

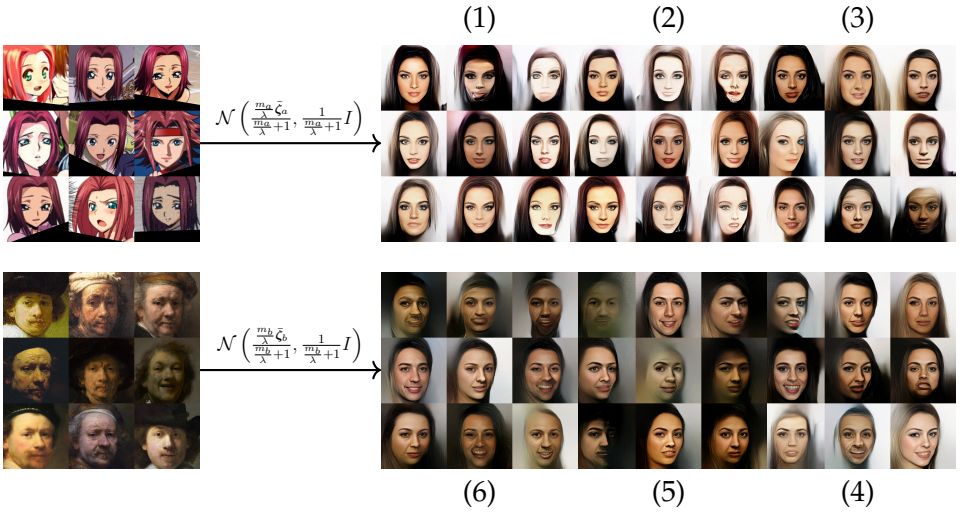


Figure 4.3: Interpolation between two sets of images far outside the training distribution, by first projecting onto the manifold of human faces, and then interpolating the parameters of the posterior distributions. Note that as the distribution gets closer to that of Rembrandt's self-portraits, the colours in the image get darker, men are sampled much more frequently, the hair is often gone from the samples (as Rembrandt often wore a hat which blended in with the background), and the sampled faces are more tilted towards the right. (View in numerical or reverse numerical order)

images almost perfectly, including those out of the distribution on which the GAN was trained, they were unable to do sensible latent space interpolations. Our method is able to do sensible interpolations between out-of-distribution datasets by interpolating the mean and covariance of their posterior distributions as we show in Figure 4.3. This method can be thought of as first projecting the out-of-distribution images onto the flow model manifold before interpolating.

Neural Style Transfer [73] is a method for re-rendering images with a different style, while also keeping the content similar. Our method can be seen as similar to Neural Style Transfer methods, with the 'content' being provided by the flow model and the 'style' being provided by the evidence. Unlike previous Neural Style Transfer methods which work on a single content image, we learn an entire distribution from which we can sample.

CycleGAN [228] and Few-Shot Unsupervised Image-to-Image Translation [126] are also methods for blending two unpaired datasets, but the aim is slightly different. Whereas these methods are capable of turning one specific image into that resembling a different class, we are able to generate many diverse samples of the class given as evidence. At the same time, our method would be unable to modify a single image in a meaningful way.

Glow [112] explored the use of manipulation vectors in order to induce specific attributes in images. Whereas their simple algebraic method required both positive and negative examples of the attribute they wished to express, we require only positive examples. We also obtain a full posterior distribution from which we can sample many diverse images, unlike their method which can only transform a single image.

Latent Constraints [59] explored how to sample conditionally from an unconditional generative model, similarly to our work. The main difference is in the method employed: whereas they learn value functions in order to find regions of the latent space with both desired conditional attributes and high likelihood, we exploit the invertibility of flow models in order to find a posterior with these qualities without resorting to any gradient-based training whatsoever.

The common thread between the contents of this chapter and previous works is that while many previous works showed manipulations of individual images using trained generative models, we aim to combine pre-trained generative models with new data to create an entirely new generative model.

4.6 Experiments

While there are many different types of flow models such as NICE [54], Real NVP [55], and Flow++ [98], we chose to use Glow [112] for all of our experiments. This

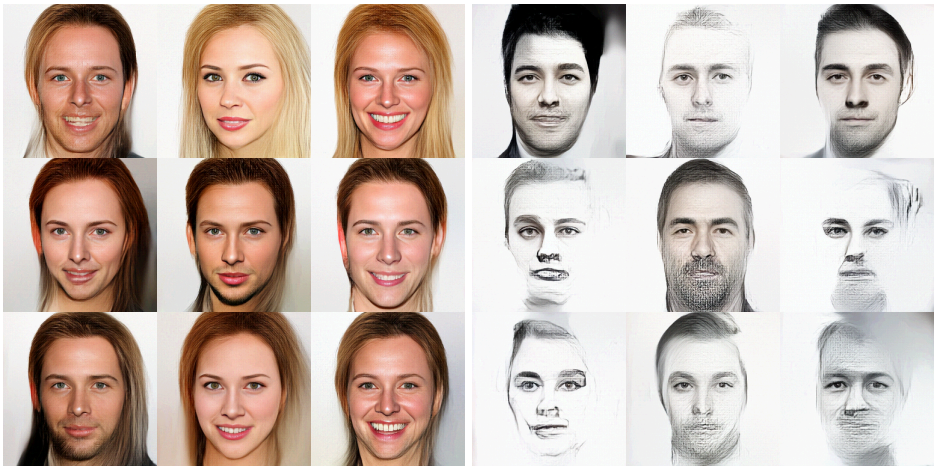
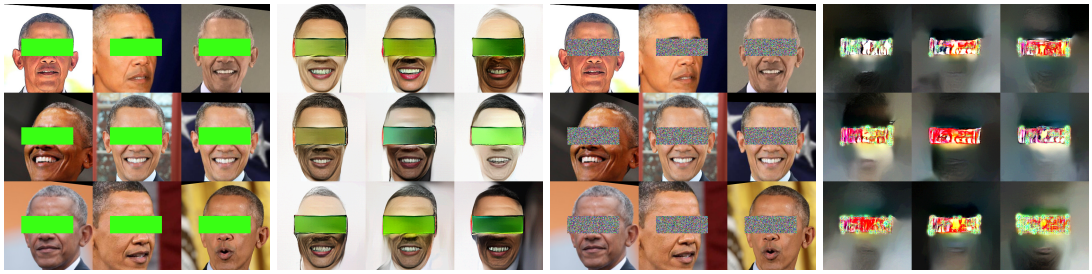


Figure 4.4: A flow model trained on CelebA, conditioned on natural images. The images of people with red hair (left) are sampled from a posterior using only five images as evidence, showing that our model is very sample-efficient. Greyscale images (right), despite not appearing in the CelebA training set, were also successfully captured by a Transflow Learning posterior.

is an arbitrary choice mainly influenced by the public availability of a pre-trained model for Glow, trained on the CelebA dataset [127]. Flow models are currently prohibitively difficult to train, both in terms of time and compute requirements, and this study is solely interested in exploring transfer learning using existing models.

4.6.1 In-distribution Conditioning

A simple experiment to demonstrate the capabilities of Transflow Learning is to sample from some coherent subset of data within the CelebA dataset, such as people with red hair, people with glasses, or individual people. We found that for categories which are strict subsets of the training data, such as people with red hair, we could create a posterior distribution which accurately captures the given attributes with both a small amount of data and a wide range of λ. In Figure 4.4 we show results from Transflow Learning given 5 images of people with red hair, a distribution which is wholly a subset of CelebA, and 21 images of greyscale human faces, a distribution which is not represented in the CelebA training set, but is also



(a) (b) (c) (d)

Figure 4.5: Even when providing Transflow Learning with evidence that is far outside of the distribution on which the flow model is originally trained (a), we are able to learn a sensible posterior distribution (b). Evidence that is so unlikely that it could not have come from a natural image (c), however, causes the posterior mean to be too far from the mean of the original distribution, and output samples (d) are no longer meaningful, even for high values of λ.

not too far off.

We also attempted to condition on natural faces with a large occlusion, and were surprised by the results. Figure 4.5 shows results when attemping to condition Glow on 25 images of President Obama with a large occlusion over his eyes. Transflow Learning was able to generate images of men with a neon-green occlusion over their eyes, despite similar images clearly not being located in the CelebA training set. It is important to reemphasize at this point that Transflow Learning in no way modifies the flow model—there was simply a region in the Glow latent space in which latent vectors corresponding to these images exist, and Transflow Learning was able to find a Gaussian covering this space. As the posterior contains elements of both the prior and the evidence, we expected the posterior to perform similar to inpainting, and were surprised to learn that the latent space of Glow was rich enough to be able to generate these images which were far outside of the training set. We only observed an inpainting-like effect for relatively high values of λ (i.e., values close to m), but at that point the model had forgotten to also generate President Obama, and was generating seemingly random samples with a faint, translucent occlusion around the eyes.

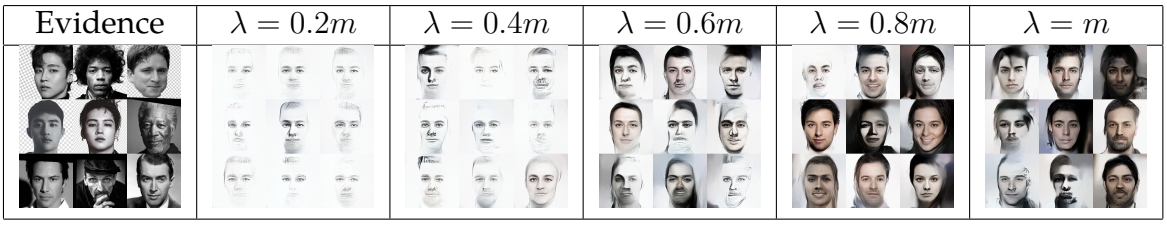


Figure 4.6: Varying the λ hyperparameter for a greyscale dataset. λ that is low creates images resembling pencil sketches, whereas λ that is high creates images with very subdued colors.

Even more surprisingly, when we changed the occlusion so that it would be made up of random pixels as opposed to one solid colour, Transflow Learning was no longer able to generate human faces, even for relatively high values of λ. This phenomenon is due to the effect of input complexity on likelihood in generative models [176]. We found that latent vectors corresponding to a real image of President Obama, the same image of President Obama with a monochromatic occlusion, and an image of an anime character have the log-likelihoods of -284,462, -281,377, and -285,610 respectively. Notably, these are all contained in roughly the same range, and the image of President Obama with a monochromatic occlusion was actually more likely than the image without the occlusion. Conversely, the latent vector corresponding to an image of President Obama with a noisy occlusion has a log-likelihood of -333,436, a number which is completely off the charts. This effect pushes the posterior too far out, to the point that samples around the posterior mean no longer correspond to meaningful images. Indeed, the pattern around the eyes in the posterior samples also resemble patterns that appear for any image corresponding to a latent vector with extremely high magnitude, which are guaranteed to be unlikely.

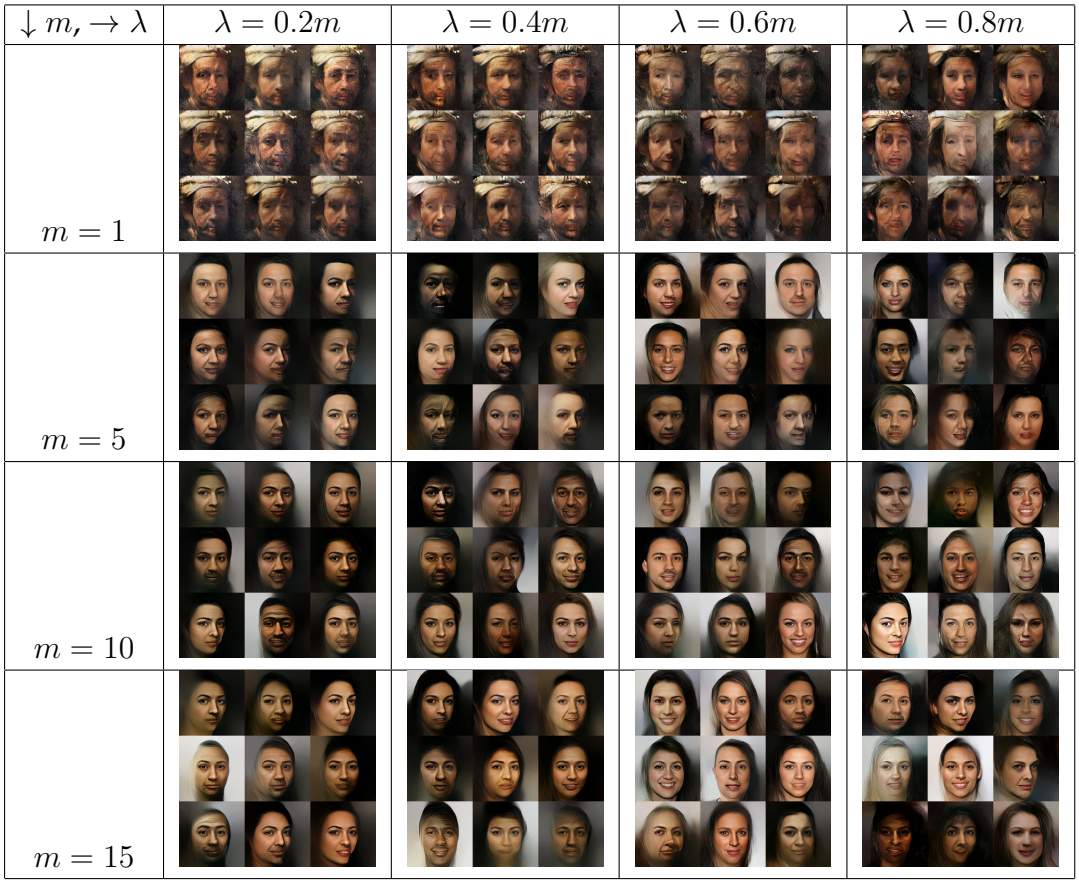


Figure 4.7: Varying λ and m simultaneously, for a Glow model trained on CelebA and then conditioned on self-portraits of Rembrandt. For very low m, samples always look like the provided evidence. As m increases, the output looks more and more like CelebA faces, painted in the style of Rembrandt's portraits. Our method is very data efficient, achieving reasonable results with as few as 5 data points. Best viewed zoomed in, as there are many fine details captured by the Transflow Learning posterior.

4.6.2 Out-of-distribution Conditioning

We also found that Transflow Learning could generate samples of many types of images which are not strictly human faces. While generated images were often nonsensical when conditioned on images which could not be interpreted in any way as a face, we found that a wide variety of images, such as cartoon faces or paintings of faces, gave interesting results. In Figure 4.7 we show the effects of varying both the amount of data, m and the hyperparameter λ given a dataset of Rembrandt's

self-portraits.

We found that the setting of λ was much more difficult in out-of-distribution scenarios. While with in-distribution conditioning we could freely set λ to any reasonable value and achieve sensible (although different) results, many settings of λ for out-of-distribution conditioning created distributions that were either too narrow or too much like the original flow model.

The CelebA dataset [127] is also strongly aligned, which created difficulty in conditioning on out-of-distribution datasets. We found that even for datasets that could be interpreted as human faces, sample quality decreased sharply in the presence of poorly aligned inputs. This posed particular difficulty when conditioning on anime faces, as the facial keypoint detector trained on cartoon faces frequently mistook anime mouths for noses and chins for mouths, or more often failed to find a face at all.

While samples from the flow model are visually meaningless when evidence cannot be interpreted as a human face, the learned posteriors can still be used for downstream tasks. In the next section, we will show that Transflow Learning can use a flow model trained on the CelebA dataset to do MNIST classification in a low-shot setting.

4.6.3 MNIST Classification

TransFlow Learning can be used for generative classification through the posterior predictive distribution. While generative classifiers are typically less accurate than discriminative classifiers (as they must solve an intermediate problem, rather than simply learning how to classify), they are oftentimes more explainable. For instance, in cases when both a discriminative and generative classifier gives equal likelihoods to two classifications, a generative classifier can demonstrate whether it is uncertain because the input strongly agrees with both classes (high probability to be in either

class), or strongly disagrees with both classes (low probability to be in either class). A discriminative classifer would be unable to give this information [131].

The posterior predictive distribution evaluates the probability of a possible unobserved value $\boldsymbol{\zeta}$ of latent vectors conditioned on the observed values $\boldsymbol{\zeta}_{1:m}$. It is obtained by marginalizing the distribution of an observation over the posterior $q(\mathbf{z}|\boldsymbol{\zeta}_{1:m})$:

$$q(\boldsymbol{\zeta}|\boldsymbol{\zeta}_{1:m}) = \int p(\boldsymbol{\zeta}|\mathbf{z})\, q(\mathbf{z}|\boldsymbol{\zeta}_{1:m})\, \mathrm{d}z \tag{4.9}$$

In the setting of multivariate Gaussian conjugate priors that we described in Section 4.4.1, its form is known and easily calculated:

$$\boldsymbol{\mu}_{pp} = \boldsymbol{\mu}_p = \frac{\frac{m}{\lambda}\bar{\boldsymbol{\zeta}}}{\frac{m}{\lambda}+1} \tag{4.10}$$

$$\boldsymbol{\Sigma}_{pp} = \boldsymbol{\Sigma}_p + \boldsymbol{\Sigma} = \frac{1}{\frac{m}{\lambda}+1} I + \lambda I \tag{4.11}$$

Its mean is exactly the same as that of the posterior, and its covariance is also identical, save for the extra λI term, which accounts for uncertainty in the parameters. We can use the posterior predictive distribution for a wide variety of tasks unrelated to sampling, such as classification tasks. In such a scenario, our chosen workflow looks as follows:

1. Take a flow model pre-trained on any dataset
2. Compute posterior predictive distributions conditioned on a number of observations, m, from each class in MNIST, obtaining ten separate distributions
3. When given an image of a new digit $\mathbf{x}$, compute the probability of $f^{-1}(\mathbf{x})$ under each of the ten posterior predictive distributions
4. The new image $\mathbf{x}$ is classified as having come from the posterior predictive

distribution under which it was the most likely

We compared Transflow Learning to k-nearest neighbors in both pixel and the flow model latent space on the task of m-shot MNIST classification. Our results are located in Table 4.1. We wish to emphasize that unlike previous methods using generative models for few-shot learning, we did *not* pre-train our flow model on the MNIST training set. For each experiment, we used the same implementation of Glow trained on CelebA and then showed each algorithm only m labeled images from each class in the MNIST training set.

It is also important to emphasize that no 'training' in the traditional sense is done here whatsoever. In the Transflow Learning experiments, the labeled MNIST images are simply used to warp the latent distribution of the CelebA flow model (Figure 4.8). This has implications for transfer learning using large datasets as conditioning, as for a dataset of size n, we would only require n evaluations of the function from data to latent variables, f^{-1}, in order to obtain a new classifier. Compared to the common practice of gradient-based fine-tuning of models with new training data, which requires several epochs of both costly forward and backward propagations, our method is exceptionally cheap in terms of number of function evaluations required.

It may seem surprising that using a flow model trained on human faces could improve MNIST classification over naïve k-nearest neighbors at all, as these datasets have very little in common. At the same time, our experiments in Figures 4.4 and 4.6 show that there are regions of the Glow latent space dedicated to greyscale images, despite there being no greyscale images contained in the CelebA dataset. While the experiments in Figure 4.8 show that there are not regions in the latent space dedicated to handwritten digits that could be covered by a Gaussian, k-nearest neighbors performed in the Glow model latent space outperforms k-nearest neighbors in pixel space by a large margin. This implies that projections of digits into the latent space are frequently close to projections of the same digit, leading us

to believe that the Glow latent space is rich enough that the posteriors conditioned on each digit are fairly distinct.

Figure 4.8: Samples from the posterior of a CelebA flow model conditioned on MNIST, for m equal to 1, 5, and 30 respectively. For m equal to 1, the samples look very similar to the evidence. As m is increased, sample quality decreases due to the sample means becoming closer to $\vec{0}$ (and therefore becoming more 'human-like'), but prediction accuracy increases greatly.

m	Ours	Pixel k-NN	Latent k-NN
1	**30.39%**	14.99%	27.78%
5	**46.39%**	32.99%	35.10%
10	**58.73%**	19.50%	40.40%
20	**65.35%**	22.83%	44.12%
30	**69.52%**	20.55%	47.58%

Table 4.1: Accuracy on single-shot and few-shot MNIST classification, given a flow model trained on CelebA. In all k-nearest neighbors experiments, k is set to 3 when possible, otherwise 1.

4.7 Discussion

While flow models are the most natural choice to study invertible generative models, it is also possible, albeit more unwieldy, to find a mapping from data to latent vectors in other generative models. One such example is the BiGAN [56], which adds an extra term to the GAN objective in order to learn this mapping. As our methods are not specific to flow models in particular and only require the model to be invertible, it would also be possible to do posterior inference in the BiGAN latent space. As this latent space is several orders of magnitude smaller than the flow model latent space, it is very likely that posterior inference in a GAN's latent space would allow for a lower setting of the λ hyperparameter and more finely-grained results. For instance, as the sample mean would be much closer to that of a natural image than the sample mean under a flow model, perhaps it would be possible to give multiple images of a specific person as conditioning, and generate new images

of that person. At the same time, perhaps the size of the flow model latent space is contributing to the richness of samples that we are able to generate, which is a question we would like to investigate in future work.

Training generative models on multimodal datasets, such as videos with sound, is currently not feasible. If, however, it were, we could use partial data (e.g., only sound) as conditioning and then perform posterior inference in the latent space. In this scenario, the flow model would then possibly be able to generate plausible video that goes with the sound given as conditioning. Given our experiments with occluded faces, however, making this work may not be trivial.

This thesis is motivated by the increasing importance of transfer learning in the face of ever-increasing compute requirements for state-of-the-art models. Flow models are no exception, currently being prohibitively difficult to train even for simple tasks, often requiring multiple GPUs in parallel and sometimes trained for weeks on end. We therefore believe that Transflow Learning will be an extremely useful tool for scientists using flow models in the future, especially when applied to downstream tasks.

Chapter 5

Leveraging Human Knowledge with Stochastic Simulators

5.156 Probability is a generalization.
It involves a general description of a propositional form. Only in default of certainty do we need probability.
If we are not completely acquainted with a fact, but know *something* about its form.
(A proposition can, indeed, be an incomplete picture of a certain state of affairs, but it is always *a* complete picture.)
The probability proposition is, as it were, an extract from other propositions.

Tractatus Logico-Philosophicus
Ludwig Josef Johann Wittgenstein [222]

5.1 Overview

The methods introduced in Chapters 3 and 4 both involved using deep neural networks in order to transfer knowledge learned over previous tasks. In both of these scenarios, a costly training phase had to be incurred before transfer in order to learn the Bayesian priors from which one transferred knowledge.

Rather than transferring pre-learned knowledge from a machine learning model, however, it is also possible to use human knowledge about a specific problem in

order to induce structure or other known preconceptions about how the model behaves in the absence of task-specific information, similarly to the learned models introduced in previous chapters.

Probabilistic programming, as discussed in Section 2.5, is a principled method for adapting human knowledge, encoded as a stochastic computer program expressing a Bayesian prior, into interpretable posteriors over unobserved variables in the model that explain how observed data could have come to be realized. In this chapter, we create a stochastic computer program that describes a ranking algorithm, along with malicious agents who are attempting to influence its outcomes. We show that probabilistic programming techniques are effective at discovering these agents in an interpretable fashion, and provide a novel simulation-based measure of influence to quantify the effect that malicious actors produced on the dynamics of the ranking algorithm.

5.2 Introduction

In 1993 a famous New Yorker cartoon of a computer-browsing canine dryly proclaimed, 'on the Internet nobody knows you're a dog.' With hindsight this ur-meme has proven prescient with respect to the problem of authenticity on the Internet. That any one Internet user can have identities that are both multitudinous and mutable formed an important part of the network's promise as a medium for communication, self-expression and empowerment. But flexible identities also carry with them the risk of deception, with Internet-facilitated fraud coming to cast a dark shadow over the luminous future originally envisaged by techno-optimists [68].

Stakes increase dramatically when the problem of authenticity meets the power of ranking algorithms, which are responsible for fulfilling and defining information retrieval needs. Tricking an algorithm into honoring at face value those features

coming from a certain set of (malicious) users can result in disaster, with unsuspecting users being recommended content, the consumption of which serves purely to enrich a set of attackers rather than to fulfill users' information needs.

The vulnerabilities of recommendation algorithms have long been recognized. Starting with the early 2000s an expansive literature has developed around 'shilling' attacks, in particular against collaborative filtering algorithms for product recommendations [178]. Another, related concern emerged a decade later when researchers described the effects of bots and automated systems on content recommendation systems, leading to the artificial inflation of distribution of purported news content that falls far short of journalistic standards. This low-quality content – a new generation of spam – often features misinformation or at least sensationalises events to help drive traffic from unsuspecting consumers to attackers' websites.

As the Internet becomes ever more embedded in the world economy, the potential rewards to successful attacks against recommendation algorithms become ever larger, mobilizing a wide number of actors – on a wide spectrum ranging from over-eager social media promoters to organized crime networks – to try and take advantage of any ranking vulnerabilities that can enable the promotion or sinking of an item.

The adversaries of recommendation algorithms are both intelligent and adaptive, which is why many consumer Internet companies employ large teams of software engineers and scientists focused on the integrity of algorithms that recommend products or content – the defenders of infrastructure against complex attacks. The evolution of real-world algorithms occurs in the endless tug-of-war between attackers and defenders who pursue conflicting interests with varying degrees of success. In this environment there are no definitive victories – only progressive innovation which swings the balance of powers between defenders and attackers.

There are daily battles fought in the cloud between Silicon Valley data centers and

shadowy botnets. The outcomes of these confrontations present both attackers and integrity teams with the opportunity to learn and improve their craft. This makes for an important observation: while designing ranking algorithms impervious to *any* attacks seems like an impossible task, understanding how attackers may adapt in response to certain algorithmic features can help integrity teams stay ahead of their adversaries.

Ideally speaking, a good recommendations system should be able to identify and remove malicious users before they can disrupt the ranking system by a significant margin. However, to eliminate the risk of false positives a resilient ranking system can use as much data as possible. So we have to adjust the tradeoff between false positives and the damage a set of malicious users can cause to a ranking system.

Bearing these limitations in mind, as a first approximation, it seems reasonable, for the sake of greater analytical clarity, to divide the user base of an online social network into the vast majority of organic users and a minority of attack profiles. Seen in this way, discussions of inauthentic amplification and coordinated inauthentic behavior fit in with earlier analyses of 'shilling' in recommender systems [178]. Here, we study a popular class of shilling attacks known as profile injection attacks [220], in which attackers add bogus accounts to a recommendation system, and attempt to push the ratings of a certain subset of products upward and others downward, while obfuscating their intentions [162].

An important but under-appreciated aspect of the struggle between attackers and defenders of online recommendation systems is the evolutionary nature of spammers' strategies [39]. In their constant search for the best new features, spam-fighters create evolutionary pressures on the attacker population, who are forced to update their strategies or exit the spam business – this shift explains the decline in effectiveness of supervised bot-detection approaches and the development of new, unsupervised techniques that emphasize the identification of rare patterns among

groups of accounts [40, 39, 104]. The latest step in this evolution is that of 'deepfakes,' social bots whose identity is generated and reinforced through sophisticated artificial neural networks which ostensibly render them indistinguishable from non-malicious accounts [35, 63].

Tautologically, the best defense against algorithmically-amplified spam is stopping spam's algorithmic amplification – i.e., changing the incentives which make it possible for attacks to succeed in the first place. A key, and often overlooked issue is that in an adversarial environment, the very quantification of attacker success is a challenging task. This measurement difficulty is inherent in the complexity of the universe created by the interactions between ranking algorithm, content producers, benign users and malicious attackers. This issue with quantification of a very difficult counter-factual limits the effectiveness of potential safeguards against attacks on recommender systems.

In our case we are interested in asking the deceptively simple question, *how would ranking outcomes differ in the absence of malicious users*. We define malicious users here as those users misrepresenting either their motives or their identity for strategic gain involving the promotion or demotion of units of content. The question is difficult to answer because of the multitudinous feedback loops potentially at work in recommender systems, which mean that simply counting the effects directly attributable to known malicious users is insufficient for giving an accurate picture of ranking outcomes in the putative counterfactual universe in which no malicious users existed.

Probabilistic programming [202] has emerged as a principled means of dealing with complex causal scenarios not unlike the issue discussed here, being used in domains as diverse as lion behavior interpretation [52], spacecraft trajectories [2] and high-energy physics [15, 16]. It is our contention that probabilistic programming, and simulation-based inference [38] in general, can be used credibly to estimate

the difference between realized outcomes and the counter-factual scenario which excludes malicious behavior. We support our assertion by providing:

1. A proof-of-concept detection algorithm, validating that malicious user identification using simulation-based inference techniques is possible using data from the model. This is a necessary but insufficient condition for the computation of a counterfactual scenario.

2. A simulation-based counterfactual measure of influence in a ranking algorithm, grounded in an information theoretical view of the joint probability distributions of ranking outcomes in the presence, and absence, of malicious users.

3. An illustration of how the measure could be applied – we show that malicious users acting in coordination have greater impact on social network dynamics than those acting independently.

5.3 Related Work

The misuse of recommendation algorithms for adversarial gain dates back to the Internet's first growth spurt as a communication platform, and is inherently intertwined with the history of digital spam, defined as:

> 'the attempt to abuse of, or manipulate, a techno-social system by producing and injecting unsolicited, and/or undesired content aimed at steering the behavior of humans or the system itself, at the direct or indirect, immediate or long-term advantage of the spammer(s).' [63]

Recommendation algorithms have been a key vector for the amplification of spam since the 1990s, when automated – rather than curated – information retrieval first became feasible in a consumer setting, thanks to Page, Brin, Motwani,

and Winograd's (1999) now-famous development of the PageRank algorithm. Compared to earlier proposals, PageRank notably provided a mechanism which enforced algorithmic resiliency – a recursive definition of popularity which protected against simplistic attempts at faking site popularity for monetary gain through the construction of hyperlinking rings ('spamdexing').

PageRank became the algorithmic foundation for Google, the dominant search engine of the past two decades. Nonetheless, in what would become a common pattern in the Internet industry, its original mechanisms proved only partially effective against adaptive adversaries. The rise of ranking also gave birth to an entire industry, search engine optimization (SEO), dedicated to improving results against the ranking algorithm, sometimes using adversarial 'black hat' methods [132]. This evolution, in turn, led to subsequent changes to Google's algorithms to improve their resiliency against adversaries [138].

With a progressively larger proportion of the world's population coming online, recommendation algorithms play an increasingly important role in the economic, political and cultural life of societies. The rise of online social networks, in particular provides the backdrop against which the more recent development of algorithmically-amplified spam can be seen [63].

The importance of social media recommendation algorithms for the dissemination of news arguably first came to the fore during the rapidly-evolving 'Arab Spring' popular uprisings taking place across the Middle East and North Africa during the early 2010s. This historical moment revealed a new potential for the coordination of organically-driven social movements, while at the same time creating a new, expansive battleground for disinformation campaigns, for instance those run through automated accounts against the White Helmets paramilitary organization in Syria [149, 184].

Adversarial attacks against recommendation algorithms have become increas-

ingly prominent recently, given the importance of social media in shaping contentious news cycles rife with misinformation, in particular during the course of events such as the 2016 U.S. general elections [5], or the 2018 Brazilian general elections [129]. Automated posting and engagement, via 'social bots' has been recognized as a particularly important factor in the spread of disinformation on social media [64, 10, 177, 39]. The issue of automation is intertwined with that of inauthenticity, with 'coordinated inauthentic behavior' (CIB) emerging as a distinct concept both among academics [76] and industry practitioners [216].

The concept of CIB relies on the existence of 'inauthentic accounts,' distinguishable from authentic users. The notion of authenticity carries with it a great deal of complexity on the Internet. Our analysis as a result is scoped to those platforms (such as Facebook or Twitter) which rely on the explicit expectation of online identities consistent with offline personas. Even CIB itself should not be seen simply through the binary view of malicious attackers and organic users, as attackers may act to catalyze existing grievances and ideologies present among audiences ripe for manipulation [184].

To fight vote spam in user–item interactions, [23] proposed training ranking models using a method based on simulated voting spam at training time. Alternatively, [22] have proposed creating incentives for power users (*'connoisseurs'*) to counter-act the influence of spammers. More recently, [13] have proposed introducing noise into the ranking function to account for distorted incentives leading to the production of low-quality content (e.g., through link farming).

Computational social scientists such as [209] have noted that classifiers for detecting abusive content are frequently miscalibrated in that they do not line up with human evaluations and have noted that Bayesian approaches such as probabilistic programming [202] are able to not only provide interpretable predictions, but can also be used to recalibrate existing classifiers. In a more general sense, probabilistic

programming methods have been shown by [225] to be effective at anomaly detection, a generalization of social network malicious activity detection, using detection of attacks on a computer system as their testbed.

5.4 Methods

Our examination is meant to provide a minimal example of a recommender system. We choose movie ranking as our setting, given the canonical nature of the task, e.g., IMDb[1] data having a long history of use in the study of recommender systems. This is admittedly a 'toy' model, which does not account for more complex designs (i.e., personalization), or for the many issues that intervene in the deployment of recommender systems in the real world (model update cycles, A/B testing, site outages, etc.). Formulating and studying such a model allows us to focus on the derivation of core concepts such as the influence metric (Section 5.5.2) in the framework of probabilistic programming.

Our broad goal is to discover and measure the impact of malicious users attacking a recommendation algorithm. While methods such as supervised learning could plausibly solve some limited aspects of this problem, especially when these can be formulated as classification or regression tasks, we turn to simulation-based inference [38] techniques in order to produce interpretable explanations of activity in a recommendation algorithm that also provide principled uncertainty estimates in all predictions. Probabilistic programming necessitates the definition of a probabilistic generative model from which observed data could have plausibly been sampled [202]. Given this model and observed data, we will be able to find a posterior distribution over the values of latent variables (random variables that are not directly observable, such as the identities of the malicious users) in the model that could

[1] https://www.imdb.com/

Algorithm 5 Probabilistic generative model of movie ratings based on the movie ranking setting, defining the joint distribution $p(\boldsymbol{\mu}, \boldsymbol{\upsilon}, \boldsymbol{\beta}, \boldsymbol{\tau}, \boldsymbol{R}) = p(\boldsymbol{R}|\boldsymbol{\mu}, \boldsymbol{\upsilon}, \boldsymbol{\beta}, \boldsymbol{\tau})\, p(\boldsymbol{\mu}, \boldsymbol{\upsilon}, \boldsymbol{\beta}, \boldsymbol{\tau})$, where $p(\boldsymbol{R}|\cdot)$ is the likelihood and $p(\boldsymbol{\mu}, \boldsymbol{\upsilon}, \boldsymbol{\beta}, \boldsymbol{\tau})$ is the prior. Given an observed rating matrix $\boldsymbol{R}_{\text{obs}}$, we would like to infer the posterior $p(\boldsymbol{\beta}, \boldsymbol{\tau}|\boldsymbol{R}_{\text{obs}})$.

$N_\mu \in \mathbb{N}$: Number of movies
$N_\upsilon \in \mathbb{N}$: Number of users
$p_\beta \in [0, 1]$: Probability of maliciousness
$\rho_\beta \in [0, 1]$: Mean malicious rating
$\rho_\sigma \in \mathbb{R}^+$: Rating standard deviation
$\tau_\mu \in [0, N_\mu]$: Mean malicious target
$\tau_\sigma \in \mathbb{R}^+$: Malicious target standard deviation
$T \in \mathbb{N}$: Time steps
$\alpha \in [0, 1]$: Difficulty

procedure RatingModel
Sample latents:
  $\boldsymbol{\mu} \leftarrow$ fixed vector, components sampled only once as $\{\mu_i \sim \text{Uniform}(0, 1)\}_{i=1}^{N_\mu}$ ▷ Movies and taste features
  $\boldsymbol{\upsilon} \leftarrow \{\upsilon_i \sim \text{Uniform}(0, 1)\}_{i=1}^{N_\upsilon}$ ▷ Users and taste features
  $\boldsymbol{\beta} \leftarrow \{\beta_i \sim \text{Bernoulli}(p_\beta)\}_{i=1}^{N_\upsilon}$ ▷ Maliciousness of users
  $\boldsymbol{\tau} \leftarrow \{\tau_i \sim \text{TruncatedNormal}(\tau_\mu, \tau_\sigma, 0, N_\mu)\}_{i=1}^{N_\upsilon}$ ▷ Maliciousness targets (ignored for organic users)
Simulate:
  $\boldsymbol{R} \leftarrow \{\rho_{i,j} = 0\}_{i=1,j=1}^{N_\upsilon, N_\mu}$ ▷ Rating matrix (N_υ rows, N_μ cols)
  for $t = 1, \ldots, T$ **do**
    for $i = 1, \ldots, N_\upsilon$ **do**
      if β_i **then** ▷ Malicious user
        if τ_i is unrated **then**
          $j \leftarrow \tau_i$ ▷ Pick malicious target
          $\rho \leftarrow (1 - \alpha) * \rho_\beta + \alpha * \text{Rate}(\upsilon_i, \mu_j)$ ▷ Rate to boost τ_i
        else
          $j \leftarrow \text{Pick}(\boldsymbol{R})$ ▷ Pick movie based on ranking
          $\rho \leftarrow \alpha * \text{Rate}(\upsilon_i, \mu_j)$ ▷ Rate lower than or equal to taste feature match
      else ▷ Organic user
        $j \leftarrow \text{Pick}(\boldsymbol{R})$ ▷ Pick movie based on ranking
        $\rho \leftarrow \text{Rate}(\upsilon_i, \mu_j)$ ▷ Rate according to taste feature match
      $\rho_{i,j} \sim \text{TruncatedNormal}(\rho, \rho_\sigma, 0, 1)$ ▷ Sample rating by user i for movie j
  return $\boldsymbol{R}$ ▷ Return rating matrix
procedure Pick($\boldsymbol{R}$)
  $\boldsymbol{m} \leftarrow \{m_j = \frac{1}{N_\upsilon} \sum_{i=1}^{N_\upsilon} \rho_{i,j}\}_{j=1}^{N_\mu}$ ▷ Mean movie ratings
  $j \sim \text{Categorical}(N_\mu, \boldsymbol{m})$ ▷ Sample movie index based on mean ratings
  return j ▷ Return picked movie index j/
procedure Rate(υ_i, μ_j)
  return $1 - |\upsilon_i - \mu_j|$ ▷ Return rating

have resulted in the observed data.

While methods other than probabilistic programming could in theory be used to do inference in our model, we found that probabilistic programming techniques were crucial in order to allow us to rapidly iterate while designing and testing the model, so that inference would be computationally feasible while ensuring that the model reflects real-world behaviour.

A model that represents several malicious users attempting to game a recommendation algorithm modeled as a simple ranking algorithm (without loss of generality, users rating movies and being recommended new movies to watch based on the current mean rating) is given in Algorithm 5. In this model, multiple users (some malicious and some organic) are rating items, which are then ranked and suggested to other users based on their ranking. User tastes are modeled by real-valued variables $\nu_i \in [0, 1]$, which determine which movies they would naturally like. Similarly, movies have taste features $\mu_j \in [0, 1]$ which denote something akin to their genre and in our model are left fixed. We model the rating function $\mathrm{Rate}(\upsilon_i, \mu_j)$ so that user i will rate movie j higher the closer user taste ν_i is to movie taste μ_j. The resulting ratings $\rho_{i,j}$ in each user–movie pair constitute the elements of the global rating matrix $\boldsymbol{R}$.

Users select which movie they will watch next through the $\mathrm{Pick}(\boldsymbol{R})$ function, which draws a movie from a categorical distribution weighted so that higher-rated movies are drawn with higher probability and any movie previously seen by the user has zero probability of being picked. Organic users will pick movies based on the ranking, whereas malicious users will pick the movie they would like to boost (the malicious target) first and obfuscate this rating as a mixture between a very high malicious rating, and what they would rate the movie were they an organic user. Likewise, malicious users rate the other movies using the rating they would give were they an organic user, but scaled down. This behavior models the difficulty

of distinguishing malicious and organic users, and is governed by the difficulty parameter α.

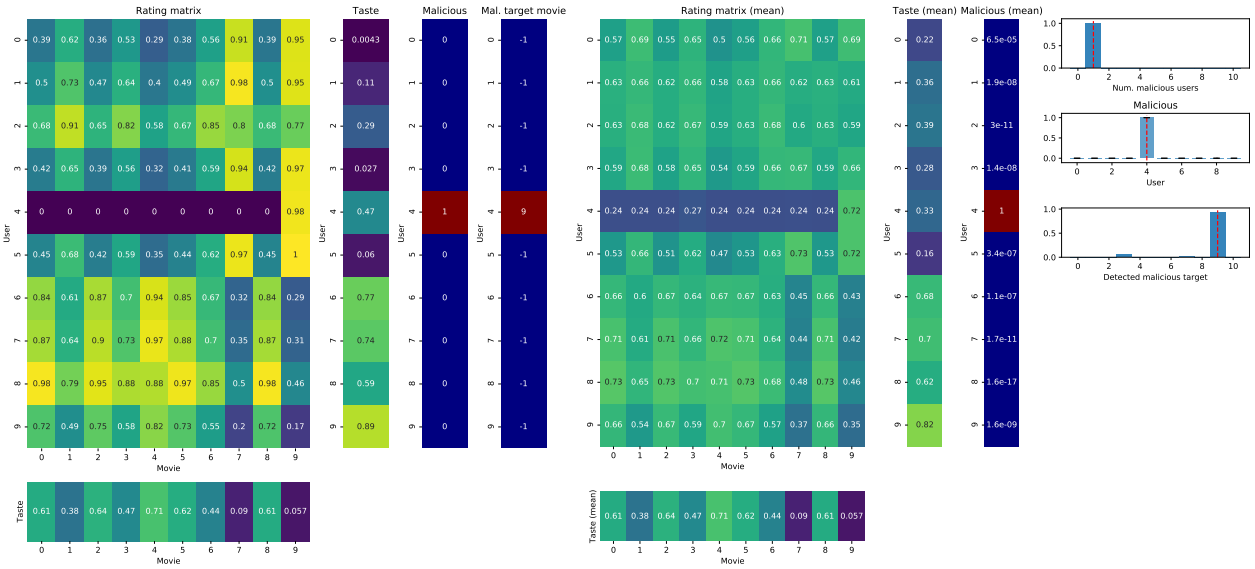
In this model the main latent variables we would like to infer are the binary variables β_i, denoting whether a given user i is malicious or not, and τ_i, denoting the target movie which user i would like to boost, if user i is malicious, i.e., if $\beta_i = 1$. Probabilistic programming will allow us to condition this model on a given rating matrix (for instance, one that represents real-world movie ratings), and then find empirical distributions over the latent variables in the simulator $(\boldsymbol{\mu}, \boldsymbol{\nu}, \boldsymbol{\beta}, \boldsymbol{\tau})$ consistent with the given rating matrix $\boldsymbol{R}_{\text{obs}}$. In summary, for the purposes of identifying malicious users and what they are trying to boost, we will obtain the posterior distribution $p(\boldsymbol{\beta}, \boldsymbol{\tau} | \boldsymbol{R}_{\text{obs}})$, while leaving $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ as nuisance variables.

We implemented Algorithm 5 in PyProb [16], a lightweight probabilistic programming library for stochastic simulators. We obtain our posteriors using weighted importance sampling [115] which gives a posterior in the form of weighted traces drawn from a proposal distribution, which is in our case the unmodified stochastic simulator. As our posterior is given to us in the form of simulations conditioned on observed data, it is by nature completely disentangled and interpretable, and will tell us the goals of each of the malicious users (τ_i) in addition to their identities (β_i). Crucially, this Bayesian approach also gives us principled uncertainty estimates associated with all our predictions.

5.5 Experiments

5.5.1 Obtaining the Posterior and Identifying Malicious Users

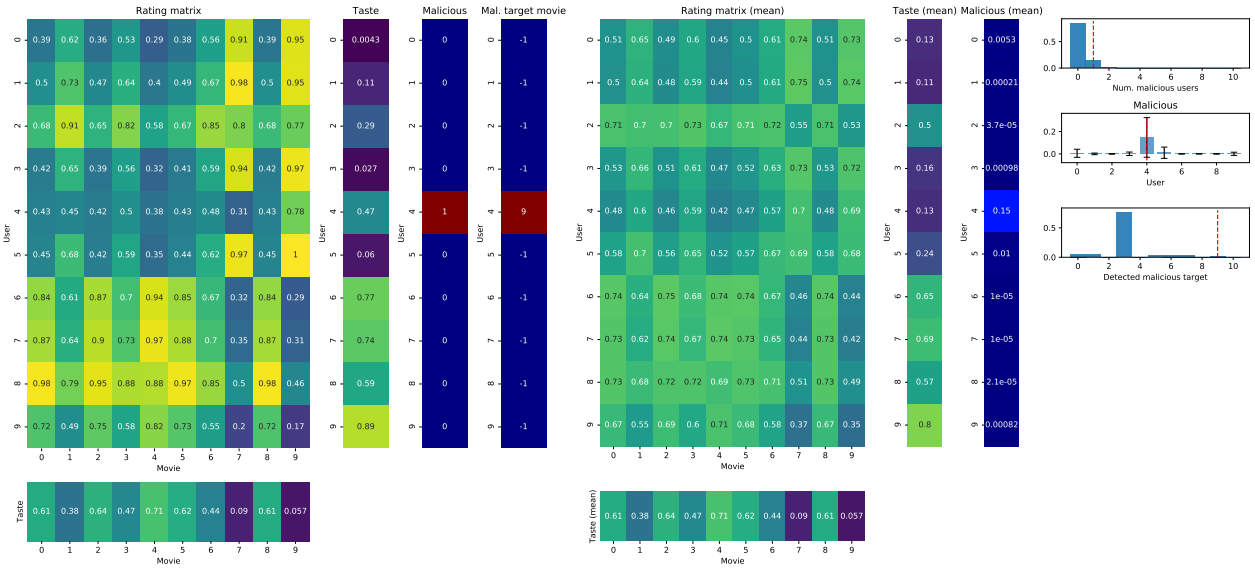
We found that obtaining a posterior over the identities of malicious users, as well as their targets, was non-trivial, with different inference engine families behaving considerably differently. Markov-chain Monte Carlo (MCMC) [139, 91, 221], despite



(a) Observed rating matrix $\boldsymbol{R}_{\text{obs}}$ (b) Empirical posterior $p(\boldsymbol{\beta}, \boldsymbol{\tau}, \boldsymbol{R}|\boldsymbol{R}_{\text{obs}})$

Figure 5.1: (a) Rating matrix $\boldsymbol{R}_{\text{obs}}$ that we observe as the input, and the corresponding ground truth values for user maliciousness $\boldsymbol{\beta}$ and targets $\boldsymbol{\tau}$. (b) Empirical posterior distribution found by weighted IS in a scenario in which malicious users do not attempt to disguise their activities ($\alpha = 0$). We can see that the posterior predictive rating matrix, $p(\boldsymbol{R}|\boldsymbol{R}_{\text{obs}})$ matches fairly closely with the ground truth $\boldsymbol{R}_{\text{obs}}$, and that the posterior malicious users $p(\boldsymbol{\beta}, \boldsymbol{\tau}|\boldsymbol{R}_{\text{obs}})$ match up with the ground truth in all posterior simulations. IS is able to discover both the malicious user and its target with little uncertainty. Dashed vertical lines show ground truth values.

being the gold standard to converge to the correct posterior given enough samples, performed extremely poorly. We suspect that this is due to the variables of interest (the malicious users and their targets) making up only a small portion of the latent variables in the model, as well as being discrete whereas the others are continuous. We observed that the model as it is currently formulated was not a good fit for the single-site MCMC [221, 202] inference engine implemented in PyProb, mainly because generating proposals where a single maliciousness latent β_i is flipped lead to very low acceptance probabilities due to the very abrupt nature of the resulting change in the rating matrix (which needs to be compensated by corresponding changes in some user taste latents υ_i that cannot be achieved in a single-site MCMC algorithm), leading to slow mixing and poor sample efficiency.



(a) Observed rating matrix $\boldsymbol{R}_{\text{obs}}$ (b) Empirical posterior $p(\boldsymbol{\beta}, \boldsymbol{\tau}, \boldsymbol{R}|\boldsymbol{R}_{\text{obs}})$

Figure 5.2: (a) Rating matrix $\boldsymbol{R}_{\text{obs}}$ that we observe as the input, sampled from a scenario in which malicious imitate non-malicious users ($\alpha = 0.3$). (b) Empirical posterior found by weighted IS. IS is able to discover this malicious user in about 15% of simulations making up the empirical posterior, but cannot discover their target accurately. Dashed vertical lines show ground truth values.

We found that weighted importance sampling (IS) [115] performed better in practice than MCMC, and used it to obtain posteriors conditioned on observed ratings matrices. We show results using IS with 100,000 executions of our model in which malicious users do not attempt to disguise their activities (i.e., difficulty $\alpha = 0$) in Figure 5.1. As can be seen in the figure, the mean of rating matrices in the posterior, i.e., the posterior predictive $p(\boldsymbol{R}|\boldsymbol{R}_{\text{obs}})$, appears to be a noisy version of the observed ground truth rating matrix $\boldsymbol{R}_{\text{obs}}$, showing that the inference scheme sampled a posterior distribution over simulation runs in which the observed rating matrix is likely.

Much more interesting are scenarios in which malicious users attempt to disguise their activities through obfuscated attacks. We model this obfuscation by setting the difficulty hyperparameter $\alpha = 0.3$ and give the result of one such experiment in Figure 5.2. This obfuscation introduces a significant amount of uncertainty into

our results, which is reflected in the detection of malicious users. Whereas in the unambiguous case IS produced an empirical posterior over malicious users and malicious target that matched with the ground truth nearly exactly, in the ambiguous case we see significant uncertainty in the empirical posterior. The results show that the most probable (0.85) explanation of the observed rating matrix is that there are no malicious users, although we also see non-negligible (0.15) probability attached to the scenario where there is one malicious user (which we know to be the ground truth for the observed rating matrix). We also see that the mode of the posterior distribution for the malicious users matches the ground truth (user 4), albeit with much lower probability (0.15) compared with the unambiguous case in Figure 5.1 (1.0). We also see that while there is probability mass associated with the ground truth value of the malicious target (movie 9), this probability is quite low, showing that the IS inference scheme has significant uncertainty in the identification of the malicious target, given the experimental setup and the number of traces (100,000) that we ran during inference.

5.5.2 Using Counterfactuals to Quantify the Damage Done by Malicious Users

In addition to giving disentangled and interpretable explanations of the behaviour of users interacting with a ranking algorithm, simulation-based inference also gives us the ability to measure the effects of users on the model's dynamics. We can quantify the amount of impact that users imposed on the dynamics of an observed rating matrix by using a slightly modified model, as shown in Algorithm 6, which allows us to disarm users by nullifying the effect of their ratings. Comparing the dynamics reflected in the distributions both with and without disarmed users gives us a counterfactual-based method of quantifying their impact on the dynamics of the simulator. Given a set of disarmed users γ, we find the distance between the

Algorithm 6 Variation of Algorithm 5, in which we can disarm a subset of users $\boldsymbol{\gamma}$ and quantify the effect (influence) of one or more users in the rating distribution, e.g., by computing a distance between predictive distributions $p(\boldsymbol{R})$ and $p(\boldsymbol{R}|\boldsymbol{\gamma})$ for the prior, or $p(\boldsymbol{R}|\boldsymbol{R}_{\text{obs}})$ and $p(\boldsymbol{R}|\boldsymbol{R}_{\text{obs}}, \boldsymbol{\gamma})$ for the posterior conditioned on a given rating matrix observation $\boldsymbol{R}_{\text{obs}}$.

$N_\mu \in \mathbb{N}$: Number of movies
$N_\upsilon \in \mathbb{N}$: Number of users
$p_\beta \in [0, 1]$: Probability of maliciousness
$\rho_\beta \in [0, 1]$: Mean malicious rating
$\rho_\sigma \in \mathbb{R}^+$: Rating standard deviation
$\tau_\mu \in [0, N_\mu]$: Mean malicious target
$\tau_\sigma \in \mathbb{R}^+$: Malicious target standard deviation
$T \in \mathbb{N}$: Time steps
$\alpha \in [0, 1]$: Difficulty
$\boldsymbol{\gamma} : \{\gamma_i \in 0, 1\}_{i=1}^{N_\upsilon}$: Disarmed users

procedure RATINGMODEL
Sample latents:
  $\boldsymbol{\mu} \leftarrow$ fixed vector, components sampled only once as $\{\mu_i \sim \text{Uniform}(0, 1)\}_{i=1}^{N_\mu}$ ▷ Movies and taste features
  $\boldsymbol{\upsilon} \leftarrow \{\upsilon_i \sim \text{Uniform}(0, 1)\}_{i=1}^{N_\upsilon}$ ▷ Users and taste features
  $\boldsymbol{\beta} \leftarrow \{\beta_i \sim \text{Bernoulli}(p_\beta)\}_{i=1}^{N_\upsilon}$ ▷ Maliciousness of users
  $\boldsymbol{\tau} \leftarrow \{\tau_i \sim \text{TruncatedNormal}(\tau_\mu, \tau_\sigma, 0, N_\mu)\}_{i=1}^{N_\upsilon}$ ▷ Maliciousness targets (ignored for organic users)
Simulate:
  $\boldsymbol{R} \leftarrow \{\rho_{i,j} = 0\}_{i=1,j=1}^{N_\upsilon, N_\mu}$ ▷ Rating matrix (N_υ rows, N_μ cols)
  for $t = 1, \ldots, T$ **do**
    for $i = 1, \ldots, N_\upsilon$ **do**
      if not γ_i **then** ▷ Do not let user rate if disarmed
        if β_i **then** ▷ Malicious user
          if τ_i is unrated **then**
            $j \leftarrow \tau_i$ ▷ Pick malicious target
            $\rho \leftarrow (1 - \alpha) * \rho_\beta + \alpha * \text{Rate}(\upsilon_i, \mu_j)$ ▷ Rate to boost τ_i
          else
            $j \leftarrow \text{Pick}(\boldsymbol{R})$ ▷ Pick movie based on ranking
            $\rho \leftarrow \alpha * \text{Rate}(\upsilon_i, \mu_j)$ ▷ Rate lower than or equal to taste feature match
        else ▷ Organic user
          $j \leftarrow \text{Pick}(\boldsymbol{R})$ ▷ Pick movie based on ranking
          $\rho \leftarrow \text{Rate}(\upsilon_i, \mu_j)$ ▷ Rate according to taste feature match
        $\rho_{i,j} \sim \text{TruncatedNormal}(\rho, \rho_\sigma, 0, 1)$ ▷ Sample rating by user i for movie j
  return $\boldsymbol{R}$ ▷ Return rating matrix
procedure PICK($\boldsymbol{R}$)
  $\boldsymbol{m} \leftarrow \{m_j = \frac{1}{N_\upsilon} \sum_{i=1}^{N_\upsilon} \rho_{i,j}\}_{j=1}^{N_\mu}$ ▷ Mean movie ratings
  $j \sim \text{Categorical}(N_\mu, \boldsymbol{m})$ ▷ Sample movie index based on mean ratings
  return j ▷ Return picked movie index j/
procedure RATE(υ_i, μ_j)
  return $1 - |\upsilon_i - \mu_j|$ ▷ Return rating

posterior-predictive distributions $p(\boldsymbol{R}|\boldsymbol{R}_{\text{obs}})$ and $p(\boldsymbol{R}|\boldsymbol{R}_{\text{obs}}, \boldsymbol{\gamma})$ given observed data $\boldsymbol{R}_{\text{obs}}$, or between the prior-predictive $p(\boldsymbol{R})$ and $p(\boldsymbol{R}|\boldsymbol{\gamma})$ in the generic case without an observed $\boldsymbol{R}_{\text{obs}}$.

Our chosen metric for measuring the impact that a disarmed user or group of users has caused is the average Jensen-Shannon (JS) distance [58], the square

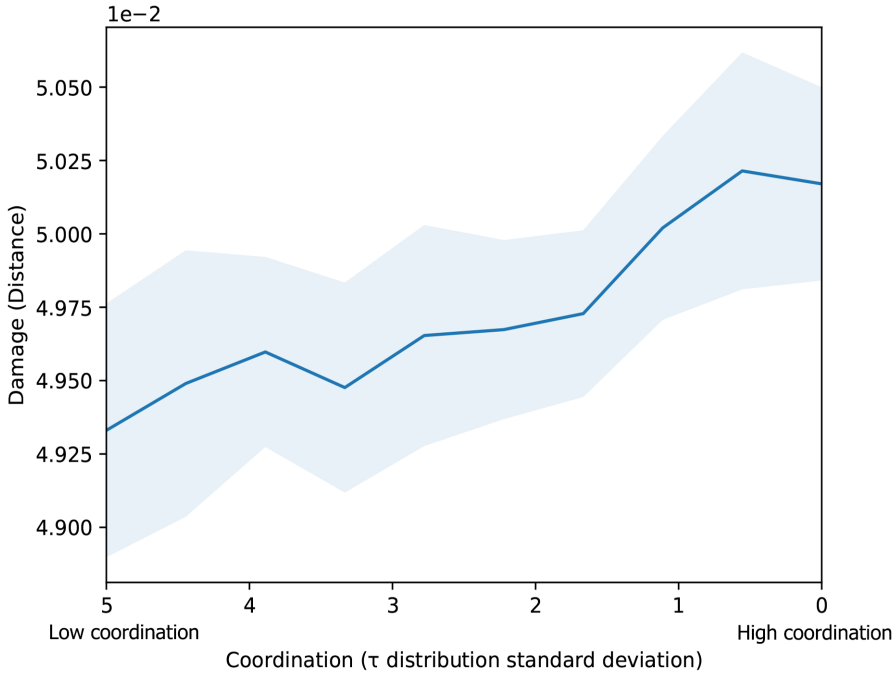


Figure 5.3: Measuring the effect on the rating matrix as the standard deviation of the distribution from which we draw τ is increased. Lower values of standard deviation (i.e., more coordination between malicious actors) results in a higher impact to the dynamics of the ratings in the matrix. Results averaged over 10 different seeds, with one standard deviation bounds shown.

root of the symmetric JS divergence [42], computed as the average of JS distances between the counterfactual and real probability distributions over ratings, per entry in the rating matrix. In a distribution over rating matrices $p(\boldsymbol{R}|\cdot)$, each entry in the matrix is a probability distribution (in the empirical case, a histogram) over ratings for each user–movie pair over a large number of simulator executions. (Note that in the posterior distributions in Figures 5.1 and 5.2 we show only the mean of these distributions for each rating entry in the rating matrix.) Our measure for impact is then the average JS distance for each of these histograms, between the realized and counterfactual rating matrices. The JS distance is a valid metric between probability distributions and always normalised to $[0, 1]$, making it an attractive choice to measure distances between distributions over matrix entries.

We show results using our influence measure between counterfactual and realised

ratings matrices while varying the amount of coordination between malicious users in Figure 5.3. When malicious targets are drawn from a distribution with a low standard deviation, malicious actors act in a more coordinated fashion (as they are more likely to target the same movie), which leads to a higher average distance between the counterfactual and realized ratings distributions. As the malicious target standard deviation increases, malicious users act against each others' interests, leading to a lower overall impact on the dynamics of the model.

5.6 Discussion

We have suggested the use of probabilistic programming techniques to both discover and measure the influence of malicious users interacting with a ranking algorithm. We base our choice of this method on its conceptual advantages in modeling the full universe of possibilities deriving from the complex interactions between users, content and ranking algorithms. The use of simulation-based approaches in this setting has been limited by practical concerns, until recent advances in numerical computation – coupled with the emergence of high-quality libraries for probabilistic programming.

Probabilistic programming techniques also have some important additional advantages over other methods. Importantly, they provide for interpretable explanations as to, e.g., why a user would be classified as malicious, and provide measures of confidence in their predictions. Furthermore, generative modeling of the entities and processes involved in this setting allows us to make precise definitions of the core concepts and to quantify key aspects such as user influence and malicious user damage.

Our work shows that simulation-based inference is minimally feasible in trying to answer fundamental questions for the future of recommender systems on the

web and beyond. We do not contest, however, that more work is needed to make this method scale to the extremely large universes that characterize real-world recommender systems.

Despite the limitations mentioned, we want to emphasize the promise held by simulation-based inference for the evaluation of recommender systems. We have provided a novel measure of a user's influence over the dynamics of a ranking algorithm using a simulation-based distance between realized and counterfactual observations. Using our measure, we have shown that within our simulation, malicious users acting in coordination will have a greater impact on the dynamics of the simulation than malicious users acting independently. With hindsight this result is not unexpected, but the fact that we are able to measure effects due to coordination suggests our method effectively quantifies the feedback effects with which classic attribution methods struggle.

Chapter 6

Conclusions

4.116 Everything that can be thought at all can be thought clearly. Everything that can be said can be said clearly.

Tractatus Logico-Philosophicus
Ludwig Josef Johann Wittgenstein [222]

While the use of industry-scale computational resources have fueled significant advances in deep learning and deep reinforcement learning throughout the past few years, such advances are frequently unscalable or unreproducible due to the financial and environmental costs of training such large models on large datasets. While re-training such models is infeasible, in this thesis we proposed several novel methods that use Bayesian inference techniques to learn efficiently based off of a pre-trained neural network, or a human-designed stochastic model. Such advancements could produce real-life impacts, allowing for the re-use of large, singular and general models in specific domains without requiring large amounts of data or retraining, thus increasing equity in the field. We considered applications of these methods in deep reinforcement learning, deep generative modeling, and computational social science settings, demonstrating the broad range of subfields in which these methods can be applied.

Reinforcement learning problems, with their extreme compute requirements

and optimisation difficulty, stand to benefit greatly from advances in transfer learning. Hierarchical reinforcement learning in particular complicates matters further, necessitating the training of policies on multiple timescales, increasing compute requirements by a significant factor. In order to amortise this training cost through transfer learning, in Chapter 3 we introduced Multi-task Soft Option Learning (MSOL), a method which brings Distral [187] to the hierarchical reinforcement learning setting. MSOL extends Distral by introducing task-specific posterior latent variables corresponding to temporally extended actions ('options') which are regularised to be close to prior options shared amongst all tasks. Applying these shared policies as a regulariser allows for faster and more robust learning in new tasks.

MSOL required the training of policy networks in order to be compatible with our transfer learning setup. In reality, training MSOL in order to enable transfer learning in our hierarchical setup was much more difficult than training several independent networks, thus severely limiting its practicality. In more realistic cases, it is beneficial to be able to apply transfer learning to networks that were trained on data without transfer learning in mind, and in particular it is beneficial to do so with minimal additional training cost. In order to solve these issues, in Chapter 4 we introduced TransFlow Learning, a method by which one can repurpose a pre-trained flow model to fit new data, without having to retrain its parameters, by revealing an area in the latent space of the flow model from which one can sample the observed data with high probability. TransFlow Learning works on both subsets of the prior data distribution and on out-of-distribution data, allowing for sampling from the new distribution as well as evaluations of the probability of data under its likelihood. The latter property allows for the use of TransFlow Learning in downstream applications such as classification, where TransFlow Learning outperforms other algorithms that do not use gradient-based training such as k-nearest neighbors.

While our previously mentioned methods worked in cases where there was

a reward from an environment or a density in the form of a dataset from which one could derive learning signal, such signal will not always exist in practice. For many real-life problems, there may not even be a clearly defined ground truth, forcing us to rely on human-designed models of the behaviour which we would like to guide our understanding of novel data. To tackle issues such as these, in Chapter 5 we created a simple stochastic simulator for a recommendation algorithm, the dynamics of which several malicious users are trying to influence. This serves as a simple reduction of many real-life problems that exist with content on the Internet today, such as troll farms spreading 'fake news' [184]. We showed that probabilistic programming is effective at finding the malicious users and their aims within the simulation and additionally provided a novel counterfactual-based method for quantifying their impact on the dynamics of the simulation.

6.1 Future Work

The work done in this thesis only scratches the surface of what is possible in transfer learning. We list here a few ways in which the work described in this thesis can be extended.

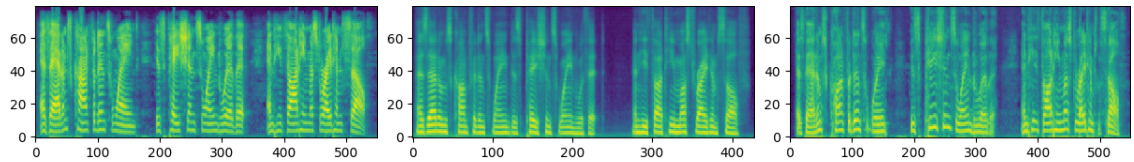
6.1.1 Extensions of MSOL to Domains Requiring Both Transfer and Hierarchy

Due to difficulties in training reinforcement learning algorithms in general, and in particular hierarchical reinforcement learning algorithms, Multi-task Soft Option Learning (MSOL) has so far only been evaluated on relatively simple tasks such as the taxi domain, a very small and discrete domain that could be solved with classical reinforcement learning algorithms easily. While these domains do make limited use of hierarchy, they are so simple to solve as to not warrant the use of

transfer learning in any practical setting. There are, however, many problems of practical importance that could make use of a hierarchical reinforcement learning algorithm that allows for efficient transfer to new tasks.

Hierarchical reinforcement learning algorithms are frequently motivated by applications in robotics. For instance, one could break down the option 'move a robotic arm to a certain position' into the low-level actions in the form of motor torques that cause that option. MSOL would find practical use in scenarios such as this, where there are many opportunities for transfer. A simple example would be to transfer learned options quickly to locations that were out of distribution for what the robot had seen previously, but more complex examples, such as transfer to different robotic arm lengths, could also be conceived and find practical use. These experiments could be run in simulation relatively easily by making simple modifications to the MuJoCo reacher environment [196].

6.1.2 Applications of TransFlow Learning Outside of Image Modeling



(a) Female (b) Male (c) Transfer

Figure 6.1: Using TransFlow Learning, researchers were able to transfer acoustic characteristics from a female speaker to a male speaker while keeping the utterance fixed and without necessitating additional flow model training. Figure reproduced from [201].

TransFlow Learning allows for the conditioning of an unconditional flow model on new data, and generation from an approximation to the new data's density. While we only evaluated our method on image data, flow models are extensively used

in areas such as high-energy physics, where they are used to sample points from complicated, high-dimensional distributions and perform numerical integration [72]. Being able to easily repurpose these flow models, which take many GPU-days to train, could lead to real-world impacts, as the problems in high-energy physics to which these integrators are applied come from a family of similar, but slightly different, functions which would normally require the training of many different flow models [72].

TransFlow Learning has also seen use in speech synthesis, where models trained on certain speakers can be transferred to others [201]. It has been shown that TransFlow Learning was successful in transferring speech across gender boundaries and across different speech styles in the same speaker [201]. Both of these examples are quite out of the distribution on which the flow models were trained, demonstrating that TransFlow Learning is still widely applicable and practical even outside of image modeling.

Normalizing flow models have seen wide application over the last few years in many other domains. Normalizing flow models have been shown to be able to learn state-of-the-art Probabilistic Context-Free Grammars (PCFGs) for natural languages [105]. While the researchers in this instance trained individual flow models for each language, one could imagine a scenario in which a flow model is instead trained on multiple languages at once. Using a TransFlow Learning posterior, it is possible that one would be able to derive PCFGs for languages on which the flow model was not trained, taking advantage of Greenberg's linguistic universals [84]. Such a setup would not only save time and compute in retraining flow models for every language, but would also provide benefits in modeling low-resource languages.

6.1.3 Making the TransFlow Learning Posterior More General

For computational efficiency, TransFlow Learning assumes a Gaussian likelihood in latent space with a scalar diagonal covariance matrix. This setup allows for practitioners to easily modify the posterior through a single hyperparameter (the scaling of the likelihood) but constrains the posterior to be an isotropic Gaussian. While this method produces good results in practice, in general the distribution of the data on which the model is conditioned could be of any form, even potentially being multimodal or entirely disconnected. Even simple, but more computationally expensive, operations such as learning the scalars on the diagonal of the TransFlow Learning likelihood covariance, could provide for a more principled posterior. Learning the parameters of the likelihood in this fashion is similar to the optimization process used to find latent vectors corresponding to images in uninvertible models such as GANs [25], but with the goal of maximizing the likelihood of a set of data, rather than just matching outputs with an image-based loss on a single image. As an even more principled but computational costly approach, one could also find a sampling-based posterior using the approximate inference techniques described in Section 2.5.1.

6.1.4 Identifying Real Malicious Activity in a Social Network

In Chapter 5, we defined a stochastic simulator, the outputs of which formed a prior distribution for how movies would be rated by malicious and non-malicious users, and then showed that we could recover the identities of the malicious users by conditioning on an observed ratings matrix. Notably, we had only explored conditioning our simulator on outputs from the prior itself, ensuring that it would be possible to obtain a sensible posterior using our model. While we could condition our simulation on any arbitrary ratings matrix, the sim-to-real gap in probabilistic programming makes conditioning a model on data that was not generated from it

difficult, albeit with some successes [52, 119]. It would be useful to practitioners who wish to use these methods in real-life settings to identify how applicable they are to real data, and if any techniques, such as domain randomization [194], can be used in order to bridge the sim-to-real gap.

Likewise, our simulation-based measure of influence that a set of actors had on the dynamics of the ranking process was shown to work on data from the model, but could also be conditioned on real data using the posterior predictive distribution described in Algorithm 6. Showing that confirmed, real coordinated malicious users cause large divergences in the model of malicious behaviour would serve not only to verify the model, but could also be used to provide more evidence for the identification of other real malicious users who had yet to be discovered.

6.2 Concluding Remarks

Since the deep learning revolution, the barrier to entry in terms of knowledge required in order to implement state of the art machine learning models has been lowered considerably, but barriers to entry in terms of compute requirements have increased by many orders of magnitude. These barriers to entry have had real affects on the field itself, with machine learning research now being dominated by large industry labs and elite universities [3]. The work in this thesis represents a step towards democratising AI, through techniques that allow the re-use of pre-trained or human-supplied models in novel situations with minimal additional learning required. As state-of-the-art machine learning models become larger and more expensive to train by the year, we look forward to future advances in transfer learning so that the benefits of these models can be enjoyed by all.

Bibliography

[1] R. Abdal, Y. Qin, and P. Wonka. Image2StyleGAN: How to embed images into the StyleGAN latent space? In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2019-Octob, pages 4431–4440, 2019. ISBN: 9781728148038. DOI: 10.1109/ICCV.2019.00453 (cited on page 84).

[2] G. Acciarini, F. Pinto, S. Metz, S. Boufelja, S. Kaczmarek, et al. Spacecraft Collision Risk Assessment with Probabilistic Programming. In *Third Workshop on Machine Learning and the Physical Sciences (NeurIPS 2020), Vancouver, Canada*, 2020 (cited on page 100).

[3] N. Ahmed and M. Wahed. The De-democratization of AI: Deep learning and the compute divide in artificial intelligence research. *arXiv preprint*, 2020. ISSN: 23318422. arXiv: 2010.15581 (cited on page 121).

[4] D. J. Aldous. Exchangeability and related topics. In *École d'Été de Probabilités de Saint-Flour XIII—1983*, pages 1–198. 1985. DOI: 10.1007/bfb0099421 (cited on page 15).

[5] H. Allcott and M. Gentzkow. Social media and fake news in the 2016 election. *Journal of Economic Perspectives*, 31(2):211–236, 2017. ISSN: 08953309. DOI: 10.1257/jep.31.2.211 (cited on page 103).

[6] S.-i. Amari. *Differential-Geometrical Methods in Statistics*. 1985. ISBN: 9780387960562 (cited on page 21).

[7] S. I. Amari. Natural Gradient Works Efficiently in Learning. *Neural Computation*, 10(2):251–276, 1998. ISSN: 08997667. DOI: 10.1162/089976698300017746 (cited on pages 21, 43).

[8] J. Andreas, D. Klein, and S. Levine. Modular multitask reinforcement learning with policy sketches. In *34th International Conference on Machine Learning*, volume 1, pages 229–239, 2017. ISBN: 9781510855144 (cited on page 62).

[9] M. Andrychowicz, M. Denil, S. G. Colmenarejo, M. W. Hoffman, D. Pfau, et al. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3988–3996, 2016 (cited on pages 4, 7).

[10] D. Arnaudo. Computational Propaganda in Brazil: Social Bots during Elections. *Computational Propaganda Research Project*, 8:1–39, 2017 (cited on page 103).

[11] P. L. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *31st AAAI Conference on Artificial Intelligence*, pages 1726–1734, 2017 (cited on pages 49, 64, 69).

[12] A. Barreto, W. Dabney, R. Munos, J. J. Hunt, T. Schaul, et al. Successor features for transfer in reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4056–4066, 2017 (cited on page 26).

[13] R. B. Basat, M. Tennenholtz, and O. Kurland. A game theoretic analysis of the adversarial retrieval setting. *Journal of Artificial Intelligence Research*, 60:1127–1164, 2017. ISSN: 10769757. DOI: 10.1613/jair.5547 (cited on page 103).

[14] J. Baxter. A Bayesian/Information Theoretic Model of Learning to Learn via Multiple Task Sampling. *Machine Learning*, 28(1):7–39, 1997. ISSN: 08856125. DOI: 10.1023/A:1007327622663 (cited on page 14).

[15] A. G. Baydin, L. Heinrich, W. Bhimji, B. Gram-Hansen, G. Louppe, et al. Efficient probabilistic inference in the quest for physics beyond the standard model. In *Advances in Neural Information Processing Systems*, volume 33, 2019 (cited on pages 7, 36, 100).

[16] A. G. Baydin, L. Shao, W. Bhimji, L. Heinrich, L. Meadows, et al. Etalumis: Bringing probabilistic programming to scientific simulators at scale. In *International Conference for High Performance Computing, Networking, Storage and Analysis, SC*, 2019. ISBN: 9781450362290. DOI: 10.1145/3295500.3356180 (cited on pages 36, 100, 107).

[17] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The Arcade Learning Environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013. ISSN: 10769757. DOI: 10.1613/jair.3912 (cited on pages 6, 18, 23, 27).

[18] R. Bellman. A Markovian Decision Process. *Indiana University Mathematics Journal*, 6(4):679–684, 1957. ISSN: 0022-2518. DOI: 10.1512/iumj.1957.6.56038 (cited on page 17).

[19] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *ACM International Conference Proceeding Series*, volume 382, 2009. ISBN: 9781605585161. DOI: 10.1145/1553374.1553380 (cited on page 48).

[20] J. Berkson. Application of the Logistic Function to Bio-Assay. *Journal of the American Statistical Association*, 39(227):357–365, 1944. ISSN: 1537274X. DOI: 10.1080/01621459.1944.10500699 (cited on page 15).

[21] L. Bertinetto, J. F. Henriques, J. Valmadre, P. H. Torr, and A. Vedaldi. Learning feed-forward one-shot learners. In *Advances in Neural Information Processing Systems*, 2016 (cited on page 4).

[22] R. Bhattacharjee and A. Goel. Algorithms and incentives for robust ranking. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, volume 07-09-Janu, pages 425–433, 2007. ISBN: 9780898716245 (cited on page 103).

[23] J. Bian, E. Agichtein, Y. Liu, and H. Zha. A few bad votes too many? Towards robust ranking in social media. In *AIRWeb 2008 - Proceedings of the 4th International Workshop on Adversarial Information Retrieval on the Web*, pages 53–60, 2008. ISBN: 9781605581590. DOI: 10.1145/1451983.1451997 (cited on page 103).

[24] E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, et al. Pyro: Deep universal probabilistic programming. *Journal of Machine Learning Research*, 20(1):973–978, 2019 (cited on page 36).

[25] P. Bojanowski, A. Joulin, D. L. Paz, and A. Szlam. Optimizing the latent space of generative networks. *35th International Conference on Machine Learning*, 2:960–972, 2018 (cited on pages 34, 120).

[26] B. E. Boser, I. M. Guyon, and V. N. Vapnik. Training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 144–152, 1992. ISBN: 089791497X. DOI: 10.1145/130385.130401 (cited on page 15).

[27] M. M. Botvinick, Y. Niv, and A. C. Barto. Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective. *Cognition*, 113(3):262–280, 2009. ISSN: 00100277. DOI: 10.1016/j.cognition.2008.08.011 (cited on page 48).

[28] S. Branavan, D. Silver, and R. Barzilay. Learning to Win by Reading Manuals in a Monte-Carlo Framework. *Journal of Artificial Intelligence Research*, 43:661–704, 2012. ISSN: 10769757. DOI: 10.1613/jair.3484 (cited on pages 23, 77).

[29] I. Bratko. PROLOG Programming for Artificial Intelligence. *Information and Software Technology*, 29(6):339–340, 1987. ISSN: 09505849. DOI: 10.1016/0950-5849(87)90035-8 (cited on page 35).

[30] J. Brehmer and K. Cranmer. Flows for simultaneous manifold learning and density estimation. In *Advances in Neural Information Processing Systems*, volume 33, 2020 (cited on pages 30, 32).

[31] S. Brooks, A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin. *Bayesian Data Analysis.* Volume 45 of number 2. Chapman and Hall/CRC, 1996, page 266. DOI: 10.2307/2988417 (cited on pages 8, 13, 79).

[32] S. P. Brooks and A. Gelman. General Methods for Monitoring Convergence of Iterative Simulations. *Journal of Computational and Graphical Statistics*, 7(4):434, 1998. ISSN: 10618600. DOI: 10.2307/1390675 (cited on pages 39, 40).

[33] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, et al. Language models are few-shot learners. *arXiv preprint*, 2020. arXiv: 2005.14165 (cited on pages 2–5, 30, 46).

[34] B. Carpenter, A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, et al. Stan: A probabilistic programming language. *Journal of Statistical Software*, 76(1), 2017. ISSN: 15487660. DOI: 10.18637/jss.v076.i01 (cited on page 36).

[35] B. Chesney and D. Citron. Deep fakes: A looming challenge for privacy, democracy, and national security. *California Law Review*, 107(6):1753–1820, 2019. ISSN: 00081221. DOI: 10.15779/Z38RV0D15J (cited on page 100).

[36] N. Chomsky. *Syntactic Structures*, volume 33 of number 3. Mouton & Co., 1957, page 375. DOI: 10.2307/411160 (cited on page 7).

[37] M. K. Cowles and B. P. Carlin. Markov Chain Monte Carlo Convergence Diagnostics: A Comparative Review. *Journal of the American Statistical Association*, 91(434), 1996. ISSN: 1537274X. DOI: 10.1080/01621459.1996.10476956 (cited on page 39).

[38] K. Cranmer, J. Brehmer, and G. Louppe. The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*, 117(48):30055–30062, 2020. ISSN: 0027-8424. DOI: 10.1073/pnas.1912789117 (cited on pages 38, 100, 104).

[39] S. Cresci. Detecting malicious social bots: Story of a never-ending clash. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 12021 LNCS, pages 77–88. Springer, 2020. ISBN: 9783030396268. DOI: 10.1007/978-3-030-39627-5_7 (cited on pages 99, 100, 103).

[40] S. Cresci, R. D. Pietro, M. Petrocchi, A. Spognardi, and M. Tesconi. The paradigm-shift of social spambots: Evidence, theories, and tools for the arms race. In *Proceedings of the 26th international conference on world wide web companion*, pages 963–972, 2017 (cited on page 100).

[41] W. M. Czarnecki, R. Pascanu, S. Osindero, S. M. Jayakumar, G. Świrszcz, et al. Distilling policy distillation. In *AISTATS 2019 - 22nd International Conference on Artificial Intelligence and Statistics*, 2019 (cited on page 25).

[42] I. Dagan, L. Lee, and F. Pereira. Similarity-based methods for word sense disambiguation. In *Proceedings of the Thirty-Fifth Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pages 56–63, 1997. DOI: 10.3115/979617.979625 (cited on page 112).

[43] L. A. Dalton and E. R. Dougherty. Optimal classifiers with minimum expected error within a Bayesian framework - Part I: Discrete and Gaussian models. *Pattern Recognition*, 2013. ISSN: 00313203. DOI: 10.1016/j.patcog.2012.10.018 (cited on page 15).

[44] L. A. Dalton and E. R. Dougherty. Optimal classifiers with minimum expected error within a Bayesian framework - Part II: Properties and performance analysis. *Pattern Recognition*, 2013. ISSN: 00313203. DOI: 10.1016/j.patcog.2012.10.019 (cited on page 15).

[45] C. Daniel, G. Neumann, O. Kroemer, and J. Peters. Hierarchical relative entropy policy search. *Journal of Machine Learning Research*, 17:1–50, 2016. ISSN: 15337928 (cited on page 62).

[46] C. Daniel, H. van Hoof, J. Peters, and G. Neumann. Probabilistic inference for determining options in reinforcement learning. *Machine Learning*, 104(2-3):337–357, 2016. ISSN: 15730565. DOI: 10.1007/s10994-016-5580-x (cited on page 62).

[47] P. Dayan. Improving Generalization for Temporal Difference Learning: The Successor Representation. *Neural Computation*, 5(4):613–624, 1993. ISSN: 0899-7667. DOI: 10.1162/neco.1993.5.4.613 (cited on page 26).

[48] C. S. de Witt, B. Gram-Hansen, N. Nardelli, A. Gambardella, R. Zinkov, et al. Simulation-Based inference for global health decisions. In *ICML Workshop on Machine Learning for Global Health, Thirty-Seventh International Conference on Machine Learning*, 2020 (cited on page 36).

[49] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, 2016 (cited on page 44).

[50] P. Del Moral, A. Doucet, and A. Jasra. On adaptive resampling strategies for sequential Monte Carlo methods. *Bernoulli*, 2012. ISSN: 13507265. DOI: 10.3150/10-BEJ335 (cited on page 42).

[51] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186, 2019 (cited on page 2).

[52] N. Dhir, F. Wood, M. Vákár, A. Markham, M. Wijers, et al. Interpreting lion behaviour with nonparametric probabilistic programs. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2017 (cited on pages 36, 100, 121).

[53] T. G. Dietterich. The MAXQ method for hierarchical reinforcement learning. In *Fifteenth International Conference on Machine Learning*, pages 118–126, 1998 (cited on page 66).

[54] L. Dinh, D. Krueger, and Y. Bengio. NICE: Non-linear independent components estimation. *3rd International Conference on Learning Representations, ICLR 2015 - Workshop Track Proceedings*, 2015 (cited on pages 31, 86).

[55] L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real NVP. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 2017. arXiv: 1605.08803 (cited on pages 31, 86).

[56] J. Donahue, P. Krähenbühl, and T. Darrell. Adversarial Feature Learning. *5th International Conference on Learning Representations*, 2017 (cited on pages 78, 94).

[57] A. Doucet and A. M. Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of Nonlinear Filtering*, 2009 (cited on page 42).

[58] D. M. Endres and J. E. Schindelin. A new metric for probability distributions. *IEEE Transactions on Information Theory*, 49(7):1858–1860, 2003. ISSN: 00189448. DOI: 10.1109/TIT.2003.813506 (cited on page 111).

[59] J. Engel, M. D. Hoffman, and A. Roberts. Latent constraints: Learning to generate conditionally from unconditional generative models. In *International Conference on Learning Representations*, 2018 (cited on pages 35, 86).

[60] J. Engel, C. Resnick, A. Roberts, S. Dieleman, M. Norouzi, et al. Neural audio synthesis of musical notes with WaveNet autoencoders. In *34th International Conference on Machine Learning, ICML 2017*, volume 3, pages 1771–1780, 2017. ISBN: 9781510855144 (cited on page 6).

[61] O. Evans, A. Stuhlmüller, J. Salvatier, and D. Filan. Modeling Agents with Probabilistic Programs, 2017. URL: https://agentmodels.org/ (cited on page 37).

[62] L. Fei-Fei, R. Fergus, and P. Perona. A Bayesian approach to unsupervised one-shot learning of object categories. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2, 2003. DOI: 10.1109/iccv.2003.1238476 (cited on page 4).

[63] E. Ferrara. The History of Digital Spam. *Communications of the ACM*, 62(8):82–91, 2019 (cited on pages 100–102).

[64] E. Ferrara, O. Varol, C. Davis, F. Menczer, and A. Flammini. The rise of social bots. *Communications of the ACM*, 59(7):96–104, 2016. ISSN: 15577317. DOI: 10.1145/2818717 (cited on page 103).

[65] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *34th International Conference on Machine Learning, ICML 2017*, 3:1856–1868, 2017 (cited on pages 4, 7).

[66] R. Fox, M. Moshkovitz, and N. Tishby. Principled Option Learning in Markov Decision Processes. *13th European Workshop on Reinforcement Learning (EWRL)*, 2016 (cited on page 62).

[67] K. Frans, J. Ho, X. Chen, P. Abbeel, and J. Schulman. Meta learning shared hierarchies. In *International Conference on Learning Representations*, 2018 (cited on pages 28, 49, 54, 59, 62, 64, 65).

[68] E. J. Friedman and P. Resnick. The social cost of cheap pseudonyms. *Journal of Economics and Management Strategy*, 10(2):173–199, 2001. ISSN: 10586407. DOI: 10.1162/105864001300122476 (cited on page 97).

[69] Y. Gal and Z. Ghahramani. Bayesian Convolutional Neural Networks with Bernoulli Approximate Variational Inference. *4th International Conference on Learning Representations (ICLR) workshop track*, 2016 (cited on page 36).

[70] Y. Gal and Z. Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *33rd International Conference on Machine Learning, ICML 2016*, volume 3, pages 1651–1660, 2016. ISBN: 9781510829008 (cited on pages 8, 36).

[71] A. Galashov, S. M. Jayakumar, L. Hasenclever, D. Tirumala, J. Schwarz, et al. Information asymmetry in KL-regularized RL. In *International Conference on Learning Representations*, 2018 (cited on page 67).

[72] C. Gao, J. Isaacson, and C. Krause. i-flow: High-dimensional Integration and Sampling with Normalizing Flows. *Machine Learning: Science and Technology*, 1(4), 2020. ISSN: 2632-2153. DOI: 10.1088/2632-2153/abab62 (cited on pages 30, 32, 119).

[73] L. Gatys, A. Ecker, and M. Bethge. A Neural Algorithm of Artistic Style. *Journal of Vision*, 16(12):326, 2016. ISSN: 1534-7362. DOI: 10.1167/16.12.326 (cited on page 85).

[74] A. Gelman and D. B. Rubin. Inference from iterative simulation using multiple sequences. *Statistical Science*, 7(4):457–472, 1992. ISSN: 08834237. DOI: 10.1214/ss/1177011136 (cited on pages 37, 39, 40).

[75] C. J. Geyer. Introduction to Markov Chain Monte Carlo, 2011. DOI: 10.1201/b10905-2 (cited on page 41).

[76] F. Giglietto, N. Righetti, L. Rossi, and G. Marino. It takes a village to manipulate the media: coordinated link sharing behavior during 2018 and 2019 Italian elections. *Information Communication and Society*, 23(6):867–891, 2020. ISSN: 14684462. DOI: 10.1080/1369118X.2020.1739732 (cited on page 103).

[77] E. Goan and C. Fookes. Bayesian Neural Networks: An Introduction and Survey. In *Case Studies in Applied Bayesian Data Science*, pages 45–87. 2018 (cited on page 36).

[78] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning Book*. 2015 (cited on pages 1, 16).

[79] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, et al. Generative adversarial networks. In *Advances in Neural Information Processing Systems*, 2014. DOI: 10.1145/3422622 (cited on pages 1, 29, 30, 33, 77, 84).

[80] A. D. Gordon, T. A. Henzinger, A. V. Nori, and S. K. Rajamani. Probabilistic programming. In *Future of Software Engineering, FOSE 2014 - Proceedings*, pages 167–181, 2014. ISBN: 9781450328654. DOI: 10.1145/2593882.2593900 (cited on page 36).

[81] P. Goyal, S. Niekum, and R. J. Mooney. Using natural language for reward shaping in reinforcement learning. In *IJCAI International Joint Conference on Artificial Intelligence*, volume 2019-Augus, pages 2385–2391, 2019. ISBN: 9780999241141. DOI: 10.24963/ijcai.2019/331 (cited on pages 23, 62).

[82] B. J. Gram-Hansen, C. S. De Witt, T. Rainforth, P. H. Torr, Y. W. Teh, et al. Hijacking malaria simulators with probabilistic programming. In *International Conference on Machine Learning AI for Social Good workshop*, 2019 (cited on pages 7, 36).

[83] D. S. Greenberg, M. Nonnenmacher, and J. H. Macke. Automatic posterior transformation for likelihood-free inference. In *36th International Conference on Machine Learning, ICML 2019*, 2019. ISBN: 9781510886988 (cited on page 32).

[84] Greenberg J. H. Some universals of grammar with particular reference to the order of meaningful elements. In *Universals of Language*. 1963 (cited on pages 6, 119).

[85] K. Gregor, D. J. Rezende, and D. Wierstra. Variational intrinsic control. In *5th International Conference on Learning Representations - Workshop Track Proceedings*, 2017 (cited on pages 49, 61).

[86] J. Gu, Y. Wang, Y. Chen, K. Cho, and V. O. Li. Meta-learning for low-resource neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018. DOI: 10.18653/v1/d18-1398 (cited on page 6).

[87] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. On calibration of modern neural networks. In *34th International Conference on Machine Learning*, volume 3, 2017 (cited on page 36).

[88] R. Haarnoja, K. Hartikainen, P. Abbeel, and S. Levine. Latent space policies for hierarchical reinforcement learning. In *35th International Conference on Machine Learning, ICML 2018*, volume 4, pages 2965–2975, 2018. ISBN: 9781510867963 (cited on page 62).

[89] J. Harb, P. L. Bacon, M. Klissarov, and D. Precup. When waiting is not an option: Learning options with a deliberation cost. In *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pages 3165–3172, 2018. ISBN: 9781577358008 (cited on page 58).

[90] A. Harutyunyan, W. Dabney, D. Borsa, N. Heess, R. Munos, et al. The termination critic. In *AISTATS 2019 - 22nd International Conference on Artificial Intelligence and Statistics*, 2019 (cited on page 61).

[91] W. K. Hastings. Monte Carlo Sampling Methods Using Markov Chains and Their Applications. *Biometrika*, 57(1):97, 1970. ISSN: 00063444. DOI: 10.2307/2334940 (cited on pages 15, 37, 38, 42, 44, 107).

[92] K. Hausman, J. T. Springenberg, Z. Wang, N. Heess, and M. Riedmiller. Learning an embedding space for transferable robot skills. In *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 2018 (cited on page 61).

[93] N. Heess, G. Wayne, Y. Tassa, T. Lillicrap, M. Riedmiller, et al. Learning and Transfer of Modulated Locomotor Controllers. *arXiv preprint*, 2016. arXiv: 1610.05182 (cited on page 62).

[94] K. A. Heller and Z. Ghahramani. Bayesian hierarchical clustering. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 297–304, 2005. ISBN: 1595931805. DOI: 10.1145/1102351.1102389 (cited on page 15).

[95] I. Higgins, A. Pal, A. Rusu, L. Matthey, C. Burgess, et al. DARLA: Improving zero-shot transfer in reinforcement learning. *34th International Conference on Machine Learning, ICML 2017*, 3:2335–2350, 2017 (cited on page 28).

[96] G. Hinton, O. Vinyals, and J. Dean. Distilling the Knowledge in a Neural Network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015 (cited on page 24).

[97] G. E. Hinton and D. van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory - COLT '93*, pages 5–13, 1993. ISBN: 0897916115. DOI: 10.1145/168304.168306 (cited on page 36).

[98] J. Ho, X. Chen, A. Srinivas, Y. Duan, and P. Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. In *36th International Conference on Machine Learning, ICML 2019*, volume 2019-June, pages 4827–4842, 2019. ISBN: 9781510886988 (cited on page 86).

[99] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735 (cited on page 43).

[100] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14(1):1303–1347, 2013. ISSN: 15324435 (cited on pages 42, 80).

[101] A. Hyvärinen. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6, 2005. ISSN: 15337928 (cited on page 33).

[102] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *32nd International Conference on Machine Learning, ICML 2015*, 1:448–456, 2015 (cited on page 34).

[103] T. Jaakkola, M. I. Jordan, and S. P. Singh. On the Convergence of Stochastic Iterative Dynamic Programming Algorithms. *Neural Computation*, 6(6):1185–1201, 1994. ISSN: 0899-7667. DOI: 10.1162/neco.1994.6.6.1185 (cited on page 18).

[104] M. Jiang, A. Beutel, P. Cui, B. Hooi, S. Yang, et al. Spotting Suspicious Behaviors in Multimodal Data: A General Metric and Algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 28(8):2187–2200, 2016. ISSN: 10414347. DOI: 10.1109/TKDE.2016.2555310 (cited on page 100).

[105] L. Jin, F. Doshi-Velez, T. Miller, W. Schuler, and L. Schwartz. Unsupervised learning of PCFGs with normalizing flow. In *ACL 2019 - 57th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, 2019. ISBN: 9781950737482. DOI: 10.18653/v1/p19-1234 (cited on page 119).

[106] N. K. Jong, T. Hester, and P. Stone. The utility of temporal abstraction in reinforcement learning. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, volume 1, pages 294–301, 2008. ISBN: 9781605604701 (cited on page 69).

[107] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. Introduction to variational methods for graphical models. *Machine Learning*, 1999. ISSN: 08856125. DOI: 10.1023/A:1007665907178 (cited on pages 29, 42).

[108] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, et al. Scaling Laws for Neural Language Models. *arXiv preprint*, 2020. arXiv: 2001.08361 (cited on page 2).

[109] A. Karbalayghareh, X. Qian, and E. R. Dougherty. Optimal Bayesian Transfer Learning. *IEEE Transactions on Signal Processing*, 66(14):3724–3739, 2018. ISSN: 1053587X. DOI: 10.1109/TSP.2018.2839583 (cited on page 15).

[110] A. Karpathy. Deep Reinforcement Learning: Pong from Pixels, 2016. URL: https://karpathy.github.io/2016/05/31/rl/ (cited on page 20).

[111] D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015 (cited on page 20).

[112] D. P. Kingma and P. Dhariwal. Glow: Generative flow with invertible 1×1 convolutions. In *Advances in Neural Information Processing Systems*, volume 2018-Decem, pages 10215–10224, 2018 (cited on pages 31, 86).

[113] D. P. Kingma and M. Welling. An Introduction to Variational Autoencoders. In *Foundations and Trends® in Machine Learning: Vol. 12: No. 4*, pages 307–392. 2019. DOI: 10.1561/2200000056. arXiv: 1906.02691. URL: http://arxiv.org/abs/1906.02691http://dx.doi.org/10.1561/2200000056 (cited on page 29).

[114] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, 2014 (cited on pages 29, 33, 84).

[115] G. Kitagawa. Monte Carlo Filter and Smoother for Non-Gaussian Nonlinear State Space Models. *Journal of Computational and Graphical Statistics*, 1996. ISSN: 10618600. DOI: 10.2307/1390750 (cited on pages 38, 42, 44, 107, 109).

[116] I. Kobyzev, S. Prince, and M. Brubaker. Normalizing Flows: An Introduction and Review of Current Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*:1–1, 2020. ISSN: 0162-8828. DOI: 10.1109/tpami.2020.2992934 (cited on page 9).

[117] S. Kotz, N. Balakrishnan, and N. L. Johnson. *Continuous Multivariate Distributions, Models and Applications: Second Edition*, volume 1. 2005, pages 1–722. ISBN: 9780471722069. DOI: 10.1002/9780471722069 (cited on page 15).

[118] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6), 2017. ISSN: 15577317. DOI: 10.1145/3065386 (cited on page 1).

[119] T. D. Kulkarni, P. Kohli, J. B. Tenenbaum, and V. Mansinghka. Picture: A probabilistic programming language for scene perception. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2015. ISBN: 9781467369640. DOI: 10.1109/CVPR.2015.7299068 (cited on pages 36, 121).

[120] B. M. Lake, R. Salakhutdinov, J. Gross, and J. B. Tenenbaum. One shot learning of simple visual concepts. In *33rd Annual Conference of the Cognitive Science Society*, 2011 (cited on page 76).

[121] T. A. Le, A. G. Baydin, and F. Wood. Inference compilation and universal probabilistic programming. *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017*, 2017 (cited on pages 8, 37, 43, 44).

[122] Y. Lecun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553), 2015. ISSN: 14764687. DOI: 10.1038/nature14539 (cited on page 1).

[123] S. Levine. Reinforcement Learning and Control as Probabilistic Inference: Tutorial and Review. *arXiv preprint*, 2018. arXiv: 1805.00909 (cited on pages 37, 50–53).

[124] F. Liang, N. Arora, N. Tehrani, Y. Li, M. Tingley, et al. Accelerating Metropolis-Hastings with Lightweight Inference Compilation. In *International Conference on Artificial Intelligence and Statistics*, 2021 (cited on page 44).

[125] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, et al. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations*, 2015 (cited on page 27).

[126] M. Y. Liu, X. Huang, A. Mallya, T. Karras, T. Aila, et al. Few-shot unsupervised image-to-image translation. *Proceedings of the IEEE International Conference on Computer Vision*, 2019-Octob:10550–10559, 2019. ISSN: 15505499. DOI: 10.1109/ICCV.2019.01065 (cited on pages 2, 86).

[127] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2015 Inter, pages 3730–3738, 2015. ISBN: 9781467383912. DOI: 10.1109/ICCV.2015.425 (cited on pages 76, 87, 91).

[128] C. Ma, D. R. Ashley, J. Wen, and Y. Bengio. Universal successor features for transfer Reinforcement Learning. *arXiv preprint*, 2020. arXiv: 2001.04025 (cited on page 26).

[129] C. Machado, B. Kira, V. Narayanan, B. Kollanyi, and P. N. Howard. A study of misinformation in whatsapp groups with a focus on the brazilian presidential elections. In *The Web Conference 2019 - Companion of the World Wide Web Conference, WWW 2019*, pages 1013–1019, 2019. ISBN: 9781450366755. DOI: 10.1145/3308560.3316738 (cited on page 103).

[130] D. MacKay. Information Theory, Inference, and Learning Algorithms. *IEEE Transactions on Information Theory*, 2004. ISSN: 15579654. DOI: 10.1109/TIT.2004.834752 (cited on page 13).

[131] R. Mackowiak, L. Ardizzone, U. Köthe, and C. Rother. Generative Classifiers as a Basis for Trustworthy Image Classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2971–2981, 2021 (cited on page 92).

[132] R. A. Malaga. Search Engine Optimization—Black and White Hat Approaches. In *Advances in Computers*. Volume 78, pages 1–39. Elsevier, 2010. DOI: 10.1016/s0065-2458(10)78001-3 (cited on page 102).

[133] V. K. Mansinghka, T. D. Kulkarni, Y. N. Perov, and J. B. Tenenbaum. Approximate Bayesian image interpretation using generative probabilistic graphics programs. In *Advances in Neural Information Processing Systems*, pages 1520–1528, 2013 (cited on page 83).

[134] P. Marjoram, J. Molitor, V. Plagnol, and S. Tavaré. Markov chain Monte Carlo without likelihoods. *Proceedings of the National Academy of Sciences of the United States of America*, 100(26):15324–15328, 2003. ISSN: 00278424. DOI: 10.1073/pnas.0306899100 (cited on page 83).

[135] M. E. Maron. Automatic Indexing: An Experimental Inquiry. *Journal of the ACM (JACM)*, 8(3):404–417, 1961. ISSN: 1557735X. DOI: 10.1145/321075.321084 (cited on page 15).

[136] B. Marthi, S. J. Russell, and D. Latham. Writing Stratagus-playing Agents in Concurrent ALisp. *Proceedings of the IJCAI-05 Workshop on Reasoning, Representation, and Learning in Computer Games*:67, 2005 (cited on pages 7, 24).

[137] B. Marthi, S. J. Russell, D. Latham, and C. Guestrin. Concurrent hierarchical reinforcement learning. *Proceedings of the 20th national conference on Artificial intelligence*, 20(Section 2):1652, 2005 (cited on pages 7, 24).

[138] D. McCullagh. Testing Google's Panda Algorithm: CNET Analysis. https://www.cnet.com/news/testing-googles-panda-algorithm-cnet-analysis/, 2011. URL: http://news.cnet.com/8301-31921_3-20054797-281.html (cited on page 102).

[139] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6):1087–1092, 1953. ISSN: 00219606. DOI: 10.1063/1.1699114 (cited on pages 15, 37, 38, 42, 44, 107).

[140] B. Milch, B. Marthi, S. Russell, D. Sontag, D. L. Ong, et al. BLOG: Probabilistic models with unknown objects. In *IJCAI International Joint Conference on Artificial Intelligence*, 2005 (cited on page 44).

[141] V. Mnih, A. P. Badia, L. Mirza, A. Graves, T. Harley, et al. Asynchronous methods for deep reinforcement learning. *33rd International Conference on Machine Learning, ICML 2016*, 4:2850–2869, 2016 (cited on pages 20, 46, 69, 71).

[142] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. ISSN: 14764687. DOI: 10.1038/nature14236 (cited on pages 1, 18, 25, 46).

[143] S. Mo, M. Cho, and J. Shin. Freeze the Discriminator: a Simple Baseline for Fine-Tuning GANs. *Conference on Computer Vision and Pattern Recognition Workshops*, 2020 (cited on page 34).

[144] P. Moradi, A. AJdari Rad, A. Khadivi, and M. Hasler. Automatic Discovery of Subgoals in Reinforcement Learning using Bridgeness Centrality Measures. In *International Conference on Machine Learning*, 2001. ISBN: 1595931805 (cited on page 61).

[145] O. Nachum, H. Lee, S. Gu, and S. Levine. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 2018-Decem, pages 3303–3313, 2018 (cited on pages 49, 62).

[146] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations : Theory and application to reward shaping. *Sixteenth International Conference on Machine Learning*, 1999. ISSN: 1098-6596 (cited on pages 23, 48).

[147] A. Noguchi and T. Harada. Image generation from small datasets via batch statistics adaptation. *Proceedings of the IEEE International Conference on Computer Vision*, 2019-Octob:2750–2758, 2019. ISSN: 15505499. DOI: 10.1109/ICCV.2019.00284 (cited on page 34).

[148] A. B. Owen. Monte Carlo theory, methods and examples, 2013 (cited on page 42).

[149] D. Pacheco, A. Flammini, and F. Menczer. Unveiling Coordinated Groups Behind White Helmets Disinformation. In *Companion Proceedings of the Web Conference 2020*, pages 611–616, 2020 (cited on page 102).

[150] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. *World Wide Web Internet And Web Information Systems*, 1998. ISSN: 1752-0509. DOI: 10.1.1.31.1768 (cited on page 101).

[151] S. Panigrahi, A. Nanda, and T. Swarnkar. A Survey on Transfer Learning. *Smart Innovation, Systems and Technologies*, 194(10):781–789, 2021. ISSN: 21903026. DOI: 10.1007/978-981-15-5971-6_83 (cited on page 77).

[152] G. Papamakarios, D. C. Sterratt, and I. Murray. Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows. In *AISTATS 2019 - 22nd International Conference on Artificial Intelligence and Statistics*, 2019 (cited on page 32).

[153] J. Pearl. Evidential reasoning using stochastic simulation of causal models. *Artificial Intelligence*, 1987. ISSN: 00043702. DOI: 10.1016/0004-3702(87)90012-9 (cited on page 44).

[154] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, et al. Language Models are Unsupervised Multitask Learners. *OpenAI Blog*, 2019 (cited on pages 2, 3).

[155] A. Radul. Report on the probabilistic language scheme. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA*, 2007. ISBN: 9781595938688. DOI: 10.1145/1297081.1297085 (cited on page 44).

[156] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 2020. ISSN: 15337928 (cited on page 2).

[157] H. Raiffa and R. Schlaifer. *Applied Statistical Decision Theory*. 1961. DOI: 10.1080/00401706.1969.10490676 (cited on pages 14, 80).

[158] R. Raina, A. Madhavan, and A. Y. Ng. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th International Conference On Machine Learning, ICML 2009*, pages 873–880, 2009. ISBN: 9781605585161 (cited on page 2).

[159] R. Raina, A. Y. Ng, and D. Koller. Constructing informative priors using transfer learning. In *ACM International Conference Proceeding Series*, volume 148, pages 713–720, 2006. ISBN: 1595933832. DOI: 10.1145/1143844.1143934 (cited on page 15).

[160] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN features off-the-shelf: An astounding baseline for recognition. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 512–519, 2014. ISBN: 9781479943098. DOI: 10.1109/CVPRW.2014.131 (cited on page 16).

[161] D. J. Rezende and S. Mohamed. Variational inference with normalizing flows. In *32nd International Conference on Machine Learning, ICML 2015*, volume 2, pages 1530–1538, 2015. ISBN: 9781510810587 (cited on pages 31, 33, 46, 77–79).

[162] F. Ricci, B. Shapira, and L. Rokach. *Recommender systems handbook, Second edition*. 2015. ISBN: 9781489976376. DOI: 10.1007/978-1-4899-7637-6 (cited on page 99).

[163] D. Ritchie, A. Stuhlmüller, and N. D. Goodman. C3: Lightweight incrementalized MCMC for probabilistic programs using continuations and callsite caching. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016*, 2016 (cited on page 44).

[164] E. Robb, W.-S. Chu, A. Kumar, and J.-B. Huang. Few-Shot Adaptation of Generative Adversarial Networks. *arXiv preprint*, 2020. arXiv: 2010.11943 (cited on page 34).

[165] C. Rosset. Turing-NLG: A 17-billion-parameter language model by Microsoft, 2020. URL: https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/ (cited on page 2).

[166] D. M. Roy and L. P. Kaelbling. Efficient Bayesian task-level transfer learning. In *IJCAI International Joint Conference on Artificial Intelligence*, pages 2599–2604, 2007 (cited on page 15).

[167] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. ISSN: 00280836. DOI: 10.1038/323533a0 (cited on pages 2, 46).

[168] A. A. Rusu, S. G. Colmenarejo, Ç. Gülçehre, G. Desjardins, J. Kirkpatrick, et al. Policy distillation. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016 (cited on pages 24, 25).

[169] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, et al. Progressive Neural Networks. *arXiv preprint*, 2016. arXiv: 1606.04671 (cited on page 27).

[170] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *arXiv preprint*, 2017. arXiv: 1703.03864. URL: http://arxiv.org/abs/1703.03864 (cited on page 17).

[171] J. Salvatier, T. V. Wiecki, and C. Fonnesbeck. Probabilistic programming in Python using PyMC3. *PeerJ Computer Science*, 2016(4), 2016. ISSN: 23765992. DOI: 10.7717/peerj-cs.55 (cited on page 36).

[172] T. Schaul, D. Horgan, K. Gregor, and D. Silver. Universal value function approximators. In *32nd International Conference on Machine Learning, ICML 2015*, volume 2, pages 1312–1320, 2015. ISBN: 9781510810587 (cited on page 26).

[173] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel. Trust region policy optimization. *32nd International Conference on Machine Learning, ICML 2015*, 3:1889–1897, 2015 (cited on page 22).

[174] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016 (cited on page 20).

[175] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint*, 2017. arXiv: 1707.06347 (cited on page 71).

[176] J. Serrà, D. Álvarez, V. Gómez, O. Slizovskaia, J. F. Núñez, et al. Input complexity and out-of-distribution detection with likelihood-based generative models. In *International Conference on Learning Representations*, volume 8, 2020. arXiv: 1909.11480 (cited on page 89).

[177] C. Shao, G. L. Ciampaglia, O. Varol, K. Yang, A. Flammini, et al. The spread of low-credibility content by social bots. *Nature communications*, 9(1):1–9, 2017 (cited on page 103).

[178] M. Si and Q. Li. Shilling attacks against collaborative recommender systems: a review. *Artificial Intelligence Review*, 53(1):291–319, 2020. ISSN: 15737462. DOI: 10.1007/s10462-018-9655-x (cited on pages 98, 99).

[179] N. Siddharth, B. Paige, J. W. Van De Meent, A. Desmaison, N. D. Goodman, et al. Learning disentangled representations with semi-supervised deep generative models. In *Advances in Neural Information Processing Systems*, volume 2017-Decem, pages 5926–5936, 2017 (cited on page 35).

[180] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. ISSN: 14764687. DOI: 10.1038/nature16961 (cited on page 23).

[181] Y. Song and S. Ermon. Generative Modeling by Estimating Gradients of the Data Distribution. In *Proceedings of the 33rd Annual Conference on Neural Information Processing Systems*, 2019 (cited on pages 32, 33).

[182] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. ISSN: 15337928 (cited on page 36).

[183] Stan Development Team. Stan Reference Manual: Effective Sample Size. URL: https://mc-stan.org/docs/2_26/reference-manual/effective-sample-size-section.html (cited on page 38).

[184] K. Starbird, A. Arif, and T. Wilson. Disinformation as collaborative work: Surfacing the participatory nature of strategic information operations. *Proceedings of the ACM on Human-Computer Interaction*, 3(CSCW):1–26, 2019. ISSN: 25730142. DOI: 10.1145/3359229 (cited on pages 102, 103, 117).

[185] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. 2018 (cited on page 17).

[186] R. S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, 1999. ISSN: 00043702. DOI: 10.1016/S0004-3702(99)00052-1 (cited on pages 48, 49, 53, 61).

[187] Y. W. Teh, V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, et al. Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 2017-Decem, pages 4497–4507, 2017 (cited on pages 9, 25, 48, 52, 63, 64, 116).

[188] G. Tesauro and G. R. Galperin. On-line policy improvement using Monte-Carlo search. In *Advances in Neural Information Processing Systems*, pages 1068–1074, 1997. ISBN: 0262100657 (cited on page 23).

[189] C. Tessler, S. Givony, T. Zahavy, D. J. Mankowitz, and S. Mannor. A deep hierarchical approach to lifelong learning in minecraft. In *31st AAAI Conference on Artificial Intelligence, AAAI 2017*, pages 1553–1561, 2017. arXiv: 1604.07255 (cited on pages 61, 62).

[190] N. C. Thompson, K. Greenewald, K. Lee, and G. F. Manso. The Computational Limits of Deep Learning. *arXiv preprint*, 2020. ISSN: 23318422. arXiv: 2007.05558 (cited on page 4).

[191] S Thrun and A Schwartz. Finding structure in reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 385–392, 1995 (cited on page 62).

[192] S. Thrun. Is Learning The n-th Thing Any Easier Than Learning The First? In *Advances in Neural Information Processing Systems*, page 7, 1996. DOI: 10.1.1.44.2898 (cited on page 12).

[193] D. Tirumala, H. Noh, A. Galashov, L. Hasenclever, A. Ahuja, et al. Exploiting Hierarchy for Learning and Transfer in KL-regularized RL. *arXiv preprint*, 2019. arXiv: 1903.07438 (cited on page 62).

[194] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, et al. Domain randomization for transferring deep neural networks from simulation to the real world. In *IEEE International Conference on Intelligent Robots and Systems*, volume 2017-Septe, 2017. DOI: 10.1109/IROS.2017.8202133 (cited on page 121).

[195] E. Todorov. General duality between optimal control and estimation. In *Proceedings of the IEEE Conference on Decision and Control*, pages 4286–4292, 2008. ISBN: 9781424431243. DOI: 10.1109/CDC.2008.4739438 (cited on page 51).

[196] E. Todorov, T. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. In *IEEE International Conference on Intelligent Robots and Systems*, 2012. ISBN: 9781467317375. DOI: 10.1109/IROS.2012.6386109 (cited on page 118).

[197] D. Tolpin, J. W. Van De Meent, H. Yang, and F. Wood. Design and implementation of probabilistic programming language anglican. In *ACM International Conference Proceeding Series*, 2016. ISBN: 9781450347679. DOI: 10.1145/3064899.3064910 (cited on page 36).

[198] L. Torrey and J. Shavlik. Transfer Learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI Global, 2010. ISBN: 9781466586758. DOI: 10.1201/b17320 (cited on pages 4, 14).

[199] D. Tran, M. D. Hoffman, R. A. Saurous, E. Brevdo, K. Murphy, et al. Deep Probabilistic Programming. In *Fifth International Conference on Learning Representations*, 2017. arXiv: 1701.03757 (cited on page 36).

[200] P. A. Tsividis, T. Pouncy, J. L. Xu, J. B. Tenenbaum, and S. J. Gershman. Human Learning in Atari. *AAAI Spring Symposium - Technical Report*, SS-17-01 -:643–646, 2017 (cited on pages 12, 77).

[201] R. Valle, K. Shih, R. Prenger, and B. Catanzaro. Flowtron: an Autoregressive Flow-based Generative Network for Text-to-Speech Synthesis. *arXiv preprint*, 2020. arXiv: 2005.05957 (cited on pages 118, 119).

[202] J.-W. van de Meent, B. Paige, H. Yang, and F. Wood. An Introduction to Probabilistic Programming. *arXiv preprint*, 2018. arXiv: 1809.10756 (cited on pages 7, 100, 103, 104, 108).

[203] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, et al. WaveNet: A Generative Model for Raw Audio. *arXiv preprint*, 2016. arXiv: 1609.03499 (cited on pages 30, 46).

[204] A. Van Den Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. In *33rd International Conference on Machine Learning, ICML 2016*, volume 4, pages 2611–2620, 2016. ISBN: 9781510829008 (cited on page 30).

[205] A. Van Den Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, et al. Conditional image generation with PixelCNN decoders. In *Advances in Neural Information Processing Systems*, pages 4797–4805, 2016 (cited on pages 30, 33, 77).

[206] E. van der Spoel, M. P. Rozing, J. J. Houwing-Duistermaat, P. Eline Slagboom, M. Beekman, et al. Siamese Neural Networks for One-Shot Image Recognition. In *ICML - Deep Learning Workshop*, volume 7 of number 11, pages 956–963. University of Toronto, 2015. ISBN: 9788578110796 (cited on page 76).

[207] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, et al. Attention Is All You Need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 6000–6010, 2017 (cited on page 2).

[208] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, et al. FeUdal networks for hierarchical reinforcement learning. In *34th International Conference on Machine Learning, ICML 2017*, volume 7, pages 5409–5418, 2017. ISBN: 9781510855144 (cited on pages 49, 62).

[209] B. Vidgen, S. Hale, S. Staton, T. Melham, H. Margetts, et al. Recalibrating classifiers for interpretable abusive content detection. In *Proceedings of the Fourth Workshop on Natural Language Processing and Computational Social Science*, pages 132–138, 2020. DOI: 10.18653/v1/2020.nlpcss-1.14 (cited on page 103).

[210] M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 2008. ISSN: 19358237. DOI: 10.1561/2200000001 (cited on pages 29, 42).

[211] M. M. Waldrop. More Than Moore. *Nature*, 530(7589):144–147, 2016. ISSN: 14764687. DOI: 10.1038/530144a (cited on page 5).

[212] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, et al. Learning to Reinforcement Learn. *CogSci*, 2017 (cited on page 48).

[213] X. Wang and G. Neubig. Target conditioned sampling: Optimizing data selection for multilingual neural machine translation. In *ACL 2019 - 57th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, 2020. DOI: 10.18653/v1/p19-1583 (cited on page 6).

[214] Y. Wang, C. Wu, L. Herranz, J. van de Weijer, A. Gonzalez-Garcia, et al. Transferring GANs: Generating images from limited data. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 218–234, 2018. ISBN: 9783030012304. DOI: 10.1007/978-3-030-01231-1_14 (cited on page 34).

[215] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992. ISSN: 0885-6125. DOI: 10.1007/bf00992698 (cited on pages 18, 24).

[216] J. Weedon, W. Nuland, and A. Stamos. Information Operations and Facebook, 2017 (cited on page 103).

[217] J. Weizenbaum. ELIZA—A Computer Program For the Study of Natural Language Communication Between Man And Machine. *Communications of the ACM*, 26(1):23–28, 1983. ISSN: 15577317. DOI: 10.1145/357980.357991 (cited on page 35).

[218] M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011*, pages 681–688, 2011. ISBN: 9781450306195 (cited on page 32).

[219] R. D. Wilkinson. Approximate Bayesian computation (ABC) gives exact results under the assumption of model error. *Statistical Applications in Genetics and Molecular Biology*, 12(2):129–141, 2013. ISSN: 15446115. DOI: 10.1515/sagmb-2013-0010 (cited on page 83).

[220] C. A. Williams, B. Mobasher, and R. Burke. Defending recommender systems: Detection of profile injection attacks. *Service Oriented Computing and Applications*, 2007. ISSN: 18632386. DOI: 10.1007/s11761-007-0013-0 (cited on page 99).

[221] D. Wingate, A. Stuhlmüller, and N. D. Goodman. Lightweight implementations of probabilistic programming languages via transformational compilation. In *Journal of Machine Learning Research*, volume 15, pages 770–778, 2011 (cited on pages 44, 45, 107, 108).

[222] L. Wittgenstein. Tractatus Logico-Philosophicus. *Annalen der Naturphilosophie*, 1921 (cited on pages 1, 6, 12, 47, 75, 96, 115).

[223] F. Wood, A. Warrington, S. Naderiparizi, C. Weilbach, V. Masrani, et al. Planning as Inference in Epidemiological Models. *arXiv preprint*, 2020. arXiv: 2003.13221 (cited on page 36).

[224] Y. Wu, E. Mansimov, S. Liao, R. Grosse, and J. Ba. Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. In *Advances in Neural Information Processing Systems*, volume 2017-Decem, pages 5280–5289, 2017 (cited on page 22).

[225] K. Xu, D. Yao, B. G. Ryder, and K. Tian. Probabilistic Program Modeling for High-Precision Anomaly Classification. In *Proceedings of the Computer Security Foundations Workshop*, 2015. ISBN: 9781467375382. DOI: 10.1109/CSF.2015.37 (cited on page 104).

[226] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems*, volume 4 of number January, pages 3320–3328, 2014 (cited on page 16).

[227] A. Zhang, H. Satija, and J. Pineau. Decoupling Dynamics and Reward for Transfer Learning. *ICLR 2018 Workshop Track*, 2018 (cited on page 6).

[228] J. Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2017-Octob, pages 2242–2251, 2017. ISBN: 9781538610329. DOI: 10.1109/ICCV.2017.244 (cited on page 86).