Riemannian Walk for Incremental Learning: Understanding Forgetting and Intransigence

Arslan Chaudhry*, Puneet K. Dokania*, Thalaiyasingam Ajanthan*, Philip H. S. Torr

University of Oxford, United Kingdom
{firstname.lastname}@eng.ox.ac.uk

Abstract. Incremental learning (IL) has received a lot of attention recently, however, the literature lacks a precise problem definition, proper evaluation settings, and metrics tailored specifically for the IL problem. One of the main objectives of this work is to fill these gaps so as to provide a common ground for better understanding of IL. The main challenge for an IL algorithm is to update the classifier whilst preserving existing knowledge. We observe that, in addition to *forgetting*, a known issue while preserving knowledge, IL also suffers from a problem we call *intransigence*, its inability to update knowledge. We introduce two metrics to quantify *forgetting* and *intransigence* that allow us to understand, analyse, and gain better insights into the behaviour of IL algorithms. Furthermore, we present RWalk, a generalization of EWC++ (our efficient version of EWC [6]) and Path Integral [25] with a theoretically grounded KL-divergence based perspective. We provide a thorough analysis of various IL algorithms on MNIST and CIFAR-100 datasets. In these experiments, RWalk obtains superior results in terms of accuracy, and also provides a better trade-off for forgetting and intransigence.

1 Introduction

Realizing human-level intelligence requires developing systems capable of learning new tasks continually while preserving *knowledge* about the old ones. This is precisely the objective underlying incremental learning (IL) algorithms. By definition, IL has ever-expanding output space, and no or limited access to data from the previous tasks while learning a new one. This makes it more challenging and fundamentally different from the classical learning paradigm where the entire dataset is available and the output space is fixed. Recently, there have been several works in IL [6,14,19,25] with varying evaluation settings and metrics making it difficult to establish fair comparisons. The first objective of this work is to rectify these issues by providing precise definitions, evaluation settings, and metrics for IL for the classification task.

Let us now discuss the key points to consider while designing IL algorithms. The first question is *'how to define knowledge: factors that quantify what the model has learned'*. Usually, knowledge is defined either using the input-output behaviour of the network [4,19] or the network parameters [6,25]. Once the knowledge is defined, the objective then is to *preserve* and *update* it to counteract two inherent issues with IL algorithms: (1) *forgetting*: catastrophically forgetting knowledge of previous tasks; and

* Joint first authors

(2) *intransigence*: inability to update the knowledge to learn the new task. Both of these problems require contradicting solutions and pose a trade-off for any IL algorithm.

To capture this trade-off, we advocate the use of measures that evaluate an IL algorithm based on its performance on the *past* and the *present* tasks in the hope that this will reflect in its behaviour in the *future* unseen tasks. Taking this into account we introduce two metrics to evaluate *forgetting* and *intransigence*. These metrics together with the standard multi-class average accuracy allow us to understand, analyse, and gain better insights into the behaviour of various IL algorithms.

In addition, we present a generalization of two recently proposed incremental learning algorithms, Elastic Weight Consolidation (EWC) [6], and Path Integral (PI) [25]. In particular, first we show that in EWC, while learning a new task, the model's likelihood distribution is regularized using a well known second-order approximation of the KL-divergence [1,17], which is equivalent to computing distance in a Riemannian manifold induced by the Fisher Information Matrix [1]. To compute and update the Fisher matrix, we use an efficient (in terms of memory) and online (in terms of computation) approach, leading to a faster and online version of EWC which we call EWC++. Note that, a similar extension to EWC, called online-EWC, is concurrently proposed by Schwarz *et al*. [21]. Next, we modify the PI [25] where instead of computing the change in the loss per unit distance in the Euclidean space between the parameters as the measure of sensitivity, we use the approximate KL divergence (distance in the Riemannian manifold) between the output distributions as the distance to compute the sensitivity. This gives us the *parameter importance score* which is accumulated over the optimization trajectory encoding information about the previous tasks as well. Finally, RWalk is obtained by combining EWC++ and the modified PI.

Furthermore, in order to counteract *intransigence*, we study different *sampling* strategies that store a small representative subset ($\leq 5\%$) of the dataset from the previous tasks. This not only allows the network to *recall* information about the previous tasks but also helps in learning to *discriminate* current and previous tasks. Finally, we present a thorough analysis to better understand the behaviour of IL algorithms on MNIST [10] and CIFAR-100 [7] datasets. To summarize, our main contributions are:

1. New evaluation metrics - *Forgetting* and *Intransigence* - to better understand the behaviour and performance of an incremental learning algorithm.
2. EWC++: An efficient and online version of EWC.
3. RWalk: A generalization of EWC++ and PI with theoretically grounded KL-divergence based perspective providing new insights.
4. An analysis of different methods in terms of accuracy, forgetting, and intransigence.

2 Problem Set-up and Preliminaries

Here we define the IL problem and discuss the practicality of two different evaluation settings: (a) single-head; and (b) multi-head. In addition, we review the probabilistic interpretation of neural networks and the connection of KL-divergence with the distance in the Riemannian manifold, both of which are crucial to our approach.

2.1 Single-head vs Multi-head Evaluations

We consider a stream of tasks, each corresponding to a set of labels. For the k-th task, let $\mathcal{D}_k = \{(\mathbf{x}_i^k, y_i^k)\}_{i=1}^{n_k}$ be the dataset, where $\mathbf{x}_i^k \in \mathcal{X}$ is the input and $y_i^k \in \mathbf{y}^k$ the ground truth label, and $\mathbf{y}^k$ is the set of labels specific to the task. The main distinction between the single-head and the multi-head evaluations is that, at test time, in *single-head*, the task identifier (k) is unknown, whereas in *multi-head*, it is given. Therefore, for the single-head evaluation, the objective at the k-th task is to learn a function $f_\theta : \mathcal{X} \rightarrow \mathcal{Y}^k$, where $\mathcal{Y}^k = \cup_{j=1}^{k} \mathbf{y}^j$ corresponds to all the known labels. For multi-head, as the task identifier is known, $\mathcal{Y}^k = \mathbf{y}^k$. For example, consider MNIST with 5 tasks: $\{\{0,1\}, \cdots, \{8,9\}\}$; trained in an incremental manner. Then, at the 5-th task, for a given image, the *multi-head* evaluation is to predict a class out of two labels $\{8,9\}$ for which the 5-th task was trained. However, the *single-head* evaluation at 5-th task is to predict a label out of all the ten classes $\{0, \cdots, 9\}$ that the model has seen thus far.

Why is single-head the right evaluation for IL? In the case of *single-head*, used by [12,19], the output space consists of all the known labels. This requires the classifier to learn to distinguish labels from different tasks as well. Since, the tasks are supplied in a sequence in IL, while learning a task, the classifier must also learn the inter-task discrimination with no or limited access[1] to the previous data. This is a much harder problem compared to multi-head where the output space contains labels of the current task only. Furthermore, single-head is more practical as knowing a priori the subset of labels to look at is a big assumption. For instance, if the task contains only one label, multi-head evaluation would be equivalent to knowing the ground truth label itself.

2.2 Probabilistic Interpretation of Neural Network Output

If the final layer of a neural network is a soft-max layer and the network is trained using cross entropy loss, then the output may be interpreted as a probability distribution over the categorical variables. Thus, at a given θ, the conditional likelihood distribution learned by a neural network is actually a conditional multinoulli distribution defined as $p_\theta(\mathbf{y}|\mathbf{x}) = \prod_{j=1}^{K} p_{\theta,j}^{[y=j]}$, where $p_{\theta,j}$ is the soft-max probability of the j-th class, K are the total number of classes, $\mathbf{y}$ is the one-hot encoding of length K, and $[\cdot]$ is Iverson bracket. A prediction can then be obtained from the likelihood distribution $p_\theta(\mathbf{y}|\mathbf{x})$. Typically, instead of sampling, a label with the highest soft-max probability is chosen as the network's prediction. Note that, if $\mathbf{y}$ corresponds to the ground-truth label then the log-likelihood is exactly the same as the negative of the cross-entropy loss, *i.e.*, if the ground-truth corresponds to the t-th index of the one-hot representation of $\mathbf{y}$, then $\log p_\theta(\mathbf{y}|\mathbf{x}) = \log p_{\theta,t}$. More insights can be found in the supplementary material.

2.3 KL-divergence as the Distance in the Riemannian Manifold

Let $D_{KL}(p_\theta \| p_{\theta+\Delta\theta})$ be the KL-divergence [8] between the conditional likelihoods of a neural network at θ and $\theta + \Delta\theta$, respectively. Then, assuming $\Delta\theta \rightarrow 0$, the second-

[1] Since the number of tasks are potentially unlimited in IL, it is impossible to store all the previous data in a scalable manner.

order Taylor approximation of the KL-divergence can be written as $D_{KL}(p_\theta \| p_{\theta+\Delta\theta}) \approx \frac{1}{2}\Delta\theta^\top F_\theta \Delta\theta = \frac{1}{2}\|\Delta\theta\|^2_{F_\theta}$ [2], where F_θ, known as the *empirical Fisher Information Matrix* [1,17] at θ, is defined as:

$$F_\theta = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}}\left[\left(\frac{\partial \log p_\theta(\mathbf{y}|\mathbf{x})}{\partial\theta}\right)\left(\frac{\partial \log p_\theta(\mathbf{y}|\mathbf{x})}{\partial\theta}\right)^\top\right], \tag{1}$$

where $\mathcal{D}$ is the dataset. Note that, as mentioned earlier, the log-likelihood $\log p_\theta(\mathbf{y}|\mathbf{x})$ is the same as the negative of the cross-entropy loss function, thus, F_θ can be seen as the *expected loss-gradient covariance matrix*. By construction (outer product of gradients), F_θ is positive semi-definite (PSD) which makes it highly attractive for second-order optimization techniques [1,17,2,9,15]. Thus, when $\Delta\theta \to 0$, computing KL-divergence $\frac{1}{2}\|\Delta\theta\|^2_{F_\theta}$ is equivalent to computing the *distance* in a Riemannian manifold[3] [11] induced by the Fisher information matrix at θ. Since $F_\theta \in \mathbb{R}^{P\times P}$ and P is usually in the order of millions for neural networks, it is practically infeasible to store F_θ. To handle this, similar to [6], we assume parameters to be independent of each other (diagonal F_θ) which results in the following approximation of the KL-divergence:

$$D_{KL}(p_\theta \| p_{\theta+\Delta\theta}) \approx \frac{1}{2}\sum_{i=1}^{P} F_{\theta_i}\,\Delta\theta_i^2\,, \tag{2}$$

where θ_i is the i-th entry of θ. Notice, the diagonal entries of F_θ are the expected square of the gradients, where the expectation is over the entire dataset. Thus, F_θ is expensive to compute as it requires a full forward-backward pass over the dataset.

3 Forgetting and Intransigence

Since the objective is to continually learn new tasks while preserving knowledge about the previous ones, an IL algorithm should be evaluated based on its performance both on the *past* and the *present* tasks in the hope that this will reflect in its behaviour on the *future* unseen tasks. To achieve this, along with average accuracy, there are two crucial components that must be quantified (1) *forgetting*: how much an algorithm forgets what it learned in the past; and (2) *intransigence*: inability of an algorithm to learn new tasks. Intuitively, if a model is heavily regularized over previous tasks to preserve knowledge, it will forget less but have high intransigence. If, in contrast, the regularization is too weak, while the intransigence will be small, the model will suffer from catastrophic forgetting. Ideally, we want a model that suffers less from both, thus efficiently utilizing the finite model capacity. In contrast, if one observes high negative correlation between forgetting and intransigence, which is usually the case, then, it suggests that either the model capacity is saturated or the method does not effectively utilize it. Before defining metrics for quantifying forgetting and intransigence, we first define the multi-class average accuracy which will be the basis for defining the other two metrics. Note, some other task specific measure of correctness (*e.g.*, IoU for object segmentation) can also be used while the definitions of forgetting and intransigence remain the same.

[2] Proof and insights are provided in the supplementary material.

[3] Since F_θ is PSD, this makes it a pseudo-manifold.

Average Accuracy (A) Let $a_{k,j} \in [0,1]$ be the accuracy (fraction of correctly classified images) evaluated on the held-out test set of the j-th task ($j \leq k$) after training the network incrementally from tasks 1 to k. Note that, to compute $a_{k,j}$, the output space consists of either $\mathbf{y}^j$ or $\cup_{j=1}^{k}\mathbf{y}^j$ depending on whether the evaluation is multi-head or single-head (refer Sec. 2.1). The average accuracy at task k is then defined as $A_k = \frac{1}{k}\sum_{j=1}^{k} a_{k,j}$. The higher the A_k the better the classifier, but this does not provide any information about forgetting or intransigence profile of the IL algorithm which would be crucial to judge its behaviour.

Forgetting Measure (F) We define forgetting for a particular task (or label) as the difference between the *maximum* knowledge gained about the task throughout the learning process in the past and the knowledge the model currently has about it. This, in turn, gives an estimate of how much the model forgot about the task given its current state. Following this, for a classification problem, we quantify forgetting for the j-th task after the model has been incrementally trained up to task $k > j$ as:

$$f_j^k = \max_{l \in \{1,\cdots,k-1\}} a_{l,j} - a_{k,j}\ , \quad \forall j < k\ . \tag{3}$$

Note, $f_j^k \in [-1,1]$ is defined for $j < k$ as we are interested in quantifying forgetting for *previous* tasks. Moreover, by normalizing against the number of tasks seen previously, the average forgetting at k-th task is written as $F_k = \frac{1}{k-1}\sum_{j=1}^{k-1} f_j^k$. Lower F_k implies less forgetting on previous tasks. Here, instead of $\max$ one could use *expectation* or $a_{j,j}$ in order to quantify the knowledge about a task in the past. However, taking $\max$ allows us to estimate forgetting along the learning process as explained below.

Positive/Negative Backward Transfer ((P/N)BT): Backward transfer (BT) is defined in [14] as the influence that learning a task k has on the performance on a previous task $j < k$. Since our objective is to measure forgetting, negative forgetting ($f_j^k < 0$) implies positive influence on the previous task or positive backward transfer (PBT) and the opposite for NBT. Furthermore, in [14], $a_{j,j}$ is used in place of $\max_{l \in \{1,\cdots,k-1\}} a_{l,j}$ (refer Eq. (3)) which makes the measure agnostic to the IL process and does not effectively capture forgetting. To understand this, let us consider an example with 4 tasks trained in an incremental manner and we are interested in measuring forgetting of task 1 after training up to task 4. Let the accuracies be $\{a_{1,1}, a_{1,2}, a_{1,3}, a_{1,4}\} = \{0.7, 0.8, 0.6, 0.5\}$. Here, forgetting measured based on Eq. (3) is $f_1^4 = 0.3$, whereas [14] would measure it as 0.2 (irrespective of the variations in $a_{1,2}$ and $a_{1,3}$). Hence, it does not capture the fact that there was a PBT in the learning process and, we believe, it is vital that an evaluation metric of an IL algorithm considers such behaviour along the learning process.

Intransigence Measure (I) We define *intransigence* as the inability of a model to learn new tasks. The effect of intransigence is more prominent in the single-head setting especially in the absence of previous data, as the model is expected to learn to differentiate the current task from the previous ones. Experimentally we show that storing just a few representative samples (refer Sec. 4.2) from the previous tasks improves intransigence

significantly. Since we wish to quantify the *inability* to learn, we compare to the standard classification model which has access to all the datasets at all times. We train a reference/target model with dataset $\bigcup_{l=1}^{k} \mathcal{D}_l$ and measure its accuracy on the held-out set of the k-th task, denoted as a_k^*. We then define the intransigence for the k-th task as:

$$I_k = a_k^* - a_{k,k} \ , \tag{4}$$

where $a_{k,k}$ denotes the accuracy on the k-th task when trained up to task k in an incremental manner. Note, $I_k \in [-1, 1]$, and lower the I_k the better the model. A reasonable reference/target model can be defined depending on the feasibility to obtain it. In situations where it is highly expensive, an approximation can be proposed.

Positive/Negative Forward Transfer ((P/N)FT): Since intransigence is defined as the gap between the accuracy of an IL algorithm and the reference model, negative intransigence ($I_k < 0$) implies learning incrementally up to task k positively influences model's knowledge about it, *i.e*., positive forward transfer (PFT). Similarly, $I_k > 0$ implies NFT. However, in [14], FT is quantified as the gain in accuracy compared to the random guess (not a measure of intransigence) which is complementary to our approach.

4 Riemannian Walk for Incremental Learning

We first describe EWC++, an efficient version of the well known EWC [6], and then RWalk which is a generalization of EWC++ and PI [25]. Briefly, RWalk has three key components: (1) a KL-divergence-based regularization over the conditional likelihood $p_\theta(\mathbf{y}|\mathbf{x})$ (EWC++); (2) a parameter importance score based on the sensitivity of the loss over the movement on the Riemannian manifold (similar to PI); and (3) strategies to obtain a few representative samples from the previous tasks. The first two components mitigate the effects of catastrophic forgetting, whereas the third handles intransigence.

4.1 Avoiding Catastrophic Forgetting

KL-divergence based Regularization (EWC++) We learn parameters for the current task such that the new conditional likelihood is close (in terms of KL) to the one learned until previous tasks. To achieve this, we regularize over the conditional likelihood distributions $p_\theta(\mathbf{y}|\mathbf{x})$ using the approximate KL-divergence, Eq. (2), as the distance measure. This would preserve the inherent properties of the model about previous tasks as the learning progresses. Thus, given parameters θ^{k-1} trained sequentially from task 1 to $k-1$, and dataset $\mathcal{D}_k$ for the k-th task, our objective is:

$$\underset{\theta}{\operatorname{argmin}}\, \tilde{L}^k(\theta) := L^k(\theta) + \lambda\, D_{KL}\left(p_{\theta^{k-1}}(\mathbf{y}|\mathbf{x}) \| p_\theta(\mathbf{y}|\mathbf{x})\right) \ , \tag{5}$$

where, λ is a hyperparameter. Substituting Eq. (2), the KL-divergence component can be written as $D_{KL}\left(p_{\theta^{k-1}} \| p_\theta\right) \approx \frac{1}{2} \sum_{i=1}^{P} F_{\theta_i^{k-1}} (\theta_i - \theta_i^{k-1})^2$. Note that, for two tasks, the above regularization is exactly the same as that of EWC [6]. Here we presented it

from the KL-divergence based perspective. Another way to look at it would be to consider Fisher[4] for each parameter to be its importance score. The intuitive explanation for this is as follows; since Fisher captures the local curvature of the KL-divergence surface of the likelihood distribution (as it is the second-derivative component of the Taylor approximation, refer Sec. 2.3), higher Fisher implies higher curvature, thus suggests to move less in that direction in order to preserve the likelihood.

In the case of multiple tasks, EWC requires storing Fisher for each task independently ($\mathcal{O}(kP)$ parameters), and regularizing over all of them jointly. This is practically infeasible if there are many tasks and the network has millions of parameters. Moreover, to estimate the empirical Fisher, EWC requires an additional pass over the dataset of each task (see Eq. (1)). To address these two issues, we propose EWC++ that (1) maintains single diagonal Fisher matrix as the training over tasks progresses, and (2) uses moving average for its efficient update similar to [15]. Given F_θ^{t-1} at $t-1$, Fisher in EWC++ is updated as:

$$F_\theta^t = \alpha F_\theta^t + (1-\alpha) F_\theta^{t-1} \ , \tag{6}$$

where F_θ^t is the Fisher matrix obtained using the *current batch* and $\alpha \in [0, 1]$ is a hyperparameter. Note, t represents the training iterations, thus, computing Fisher in this manner contains information about previous tasks, and also eliminates the additional forward-backward pass over the dataset. At the end of each task, we simply store F_θ^t as $F_{\theta^{k-1}}$ and use it to regularize the next task, thus storing only two sets of Fisher at any instant during training, irrespective of the number of tasks. Similar to EWC++, an efficient version of EWC referred to as online-EWC is concurrently developed in [21].

In EWC, Fisher is computed at a local minimum of $\tilde{L}^k$ using the gradients of L^k, which is nearly zero whenever $\tilde{L}^k \approx L^k$ (*e.g.*, smaller λ or when $k = 1$). This results in negligible regularization leading to catastrophic forgetting. This issue is partially addressed in EWC++ using moving average. However, to improve it further and to capture model's behaviour not just at the minimum but also during the entire training process, we augment each element of the diagonal Fisher with a positive scalar as described below. This also ensures that the augmented Fisher is always positive-definite.

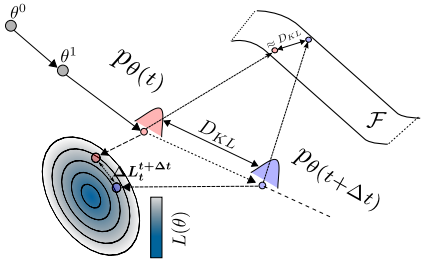


Fig. 1: *Parameter importance accumulated over the optimization trajectory.*

Optimization-path based Parameter Importance Since Fisher captures the intrinsic properties of the model and it only depends on L^k, it is blinded towards the influence of parameters over the optimization path on the loss surface of $\tilde{L}^k$. Similar to [25], we accumulate parameter importance based on $\tilde{L}^k$ over the entire training trajectory. This score is defined as the ratio of the change in the loss function to the distance between the conditional likelihood distributions per step in the parameter space.

[4] By Fisher we always mean the empirical Fisher information matrix.

More precisely, for a change of parameter from $\theta_i(t)$ to $\theta_i(t+1)$ (where t is the time step or training iteration), we define parameter importance as the ratio of the change in the loss to its influence in $D_{KL}(p_{\theta(t)} \| p_{\theta(t+1)})$. Intuitively, importance will be higher if a small change in the distribution causes large improvement over the loss. Formally, using the first-order Taylor approximation, the change in loss L can be written as:

$$L(\theta(t+\Delta t)) - L(\theta(t)) \approx -\sum_{i=1}^{P} \sum_{t=t}^{t+\Delta t} \frac{\partial L}{\partial \theta_i} \left(\theta_i(t+1) - \theta_i(t)\right) = -\sum_{i=1}^{P} \Delta L_t^{t+\Delta t}(\theta_i), \quad (7)$$

where $\frac{\partial L}{\partial \theta_i}$ is the gradient at t, and $\Delta L_t^{t+\Delta t}(\theta_i)$ represents the accumulated change in the loss caused by the change in the parameter θ_i from time step t to $t+\Delta t$. This change in parameter would cause a corresponding change in the model distribution which can be computed using the approximate KL-divergence (Eq. (2)). Thus, the importance of the parameter θ_i from training iteration t_1 to t_2 can be computed as $s_{t_1}^{t_2}(\theta_i) = \sum_{t=t_1}^{t_2} \frac{\Delta L_t^{t+\Delta t}(\theta_i)}{\frac{1}{2} F_{\theta_i}^t \Delta \theta_i(t)^2 + \epsilon}$, where $\Delta \theta_i(t) = \theta_i(t+\Delta t) - \theta_i(t)$ and $\epsilon > 0$. The denominator is computed at every discrete intervals of $\Delta t \geq 1$ and $F_{\theta_i}^t$ is computed efficiently at every t-th step using moving average as described while explaining EWC++. The computation of this importance score is illustrated in Fig. 1. Since we care about the positive influence of the parameters, negative scores are set to zero. Note that, if the Euclidean distance is used instead, the score $s_{t_1}^{t_2}(\theta_i)$ would be similar to that of PI [25].

Final Objective Function (RWalk) We now combine Fisher information matrix based importance and the optimization-path based importance scores as follows:

$$\tilde{L}^k(\theta) = L^k(\theta) + \lambda \sum_{i=1}^{P} (F_{\theta_i^{k-1}} + s_{t_0}^{t_{k-1}}(\theta_i))(\theta_i - \theta_i^{k-1})^2 \,. \quad (8)$$

Here, $s_{t_0}^{t_{k-1}}(\theta_i)$ is the score accumulated from the first training iteration t_0 until the last training iteration t_{k-1}, corresponding to task $k-1$. Since the scores are accumulated over time, the regularization gets increasingly rigid. To alleviate this and enable continual learning, after each task the scores are averaged: $s_{t_0}^{t_{k-1}}(\theta_i) = \frac{1}{2}\left(s_{t_0}^{t_{k-2}}(\theta_i) + s_{t_{k-2}}^{t_{k-1}}(\theta_i)\right)$. This continual averaging makes the tasks learned far in the past less influential than the tasks learned recently. Furthermore, while adding, it is important to make sure that the scales of both $F_{\theta_i^{k-1}}$ and $s_{t_0}^{t_{k-1}}(\theta_i)$ are in the same order, so that the influence of both the terms is retained. This can be ensured by individually normalizing them to be in the interval $[0, 1]$. *This, together with score averaging, have a positive side-effect of the regularization hyperparameter λ being less sensitive to the number of tasks.* However, EWC [6] and PI [25] are highly sensitive to λ, making them relatively less reliable for IL. Note, during training, the space complexity for RWalk is $\mathcal{O}(P)$, independent of the number of tasks.

4.2 Handling Intransigence

Experimentally, we observed that training k-th task with $\mathcal{D}_k$ leads to a poor test accuracy for the current task compared to previous tasks in the *single-head* evaluation setting

(refer Sec. 2.1). This happens because during training the model has access to $\mathcal{D}_k$ which contains labels only for the k-th task, $\mathbf{y}^k$. However, at test time the label space is over all the tasks seen so far $\mathcal{Y}^k = \cup_{j=1}^{k} \mathbf{y}^j$, which is much larger than $\mathbf{y}^k$. This in turn increases *confusion* at test time as the predictor function has no means to differentiate the samples of the current task from the ones of previous tasks. An intuitive solution to this problem is to store a small subset of representative samples from the previous tasks and use it while training the current task [19]. Below we discuss different strategies to obtain such a subset. Note that we store m points from each task-specific dataset as the training progresses, however, it is trivial to have a fixed total number of samples for all the tasks similar to iCaRL [19].

Uniform Sampling A naïve yet highly effective (shown experimentally) approach is to sample uniformly at random from the previous datasets.

Plane Distance-based Sampling In this case, we assume that samples closer to the decision boundary are more representative than the ones far away. For a given sample $\{\mathbf{x}_i, y_i\}$, we compute the pseudo-distance from the decision boundary $d(\mathbf{x}_i) = \phi(\mathbf{x}_i)^\top w^{y_i}$, where $\phi(\cdot)$ is the feature mapping learned by the neural network and w^{y_i} are the last fully connected layer parameters for class y_i. Then, we sample points based on $q(\mathbf{x}_i) \propto \frac{1}{d(\mathbf{x}_i)}$. Here, the intuition is, since the change in parameters is regularized, the feature space and the decision boundaries do not vary much. Hence, the samples that lie close to the boundary would act as *boundary defining samples*.

Entropy-based Sampling Given a sample, the entropy of the output soft-max distribution measures the uncertainty of the sample which we used to sample points. The higher the entropy the more likely is that the sample would be picked.

Mean of Features (MoF) iCaRL [19] proposes a method to find samples based on the feature space $\phi(\cdot)$. For each class y, m number of points are found whose mean in the feature space closely approximate the mean of the entire dataset for that class. However, this subset selection strategy is inefficient compared to the above sampling methods. In fact, the time complexity is $\mathcal{O}(nfm)$ where n is dataset size, f is the feature dimension and m is the number of required samples.

5 Related Work

One way to address catastrophic forgetting is by dynamically expanding the network for each new task [24,18,20,23]. Though intuitive and simple, these approaches are not scalable as the size of the network increases with the number of tasks. A better strategy would be to exploit the over-parametrization of neural networks [3]. This entails regularizing either over the activations (output) [19,13] or over the network parameters [6,25]. Even though activation-based approach allows more flexibility in parameter updates, it is memory inefficient if the activations are in millions, *e.g.*, semantic segmentation. On the contrary, methods that regularize over the parameters - weighting

the parameters based on their individual *importance* - are suitable for such tasks. Our method falls under the latter category and we show that our method is a generalization of EWC++ and PI [25], where EWC++ is our efficient version of EWC [6], very similar to the concurrently developed online-EWC [21]. Similar in spirit to regularization over the parameters, Lee *et al*. [12] use moment matching to obtain network weights as the combination of the weights of all the tasks, and Nguyen *et al*. [16] enforce the distribution over the model parameters to be close via a Bayesian framework. Different from the above approaches, Lopez-Paz *et al*. [14] update gradients such that the losses of the previous tasks do not increase, while Shin *et al*. [22] resort to a retraining strategy where the samples of the previous tasks are generated using a learned generative model.

6 Experiments

Datasets We evaluate baselines and our proposed model - RWalk - on two datasets:

1. *Incremental MNIST*: The standard MNIST dataset is split into five disjoint subsets (tasks) of two consecutive digits, *i.e*., $\cup_k \mathbf{y}^k = \{\{0,1\},\ldots,\{8,9\}\}$.
2. *Incremental CIFAR*: To show that our approach scales to bigger datasets, we use incremental CIFAR where CIFAR-100 dataset is split into ten disjoint subsets such that $\cup_k \mathbf{y}^k = \{\{0-9\},\ldots,\{90-99\}\}$.

Architectures The architectures used are similar to [25]. For MNIST, we use an MLP with two hidden layers each having 256 units with ReLU nonlinearites. For CIFAR-100, we use a CNN with four convolutional layers followed by a single dense layer (see supplementary for more details). In all experiments, we use Adam optimizer [5] (learning rate = 1×10^{-3}, $\beta_1 = 0.9$, $\beta_2 = 0.999$) with a fixed batch size of 64.

Baselines We compare RWalk against the following baselines:

- Vanilla: Network trained without any regularization over past tasks.
- EWC [6] and PI [25]: Both use parameter based regularization. Note, we observed that EWC++ performed at least as good as EWC and therefore, in all the experiments, by EWC we mean the stronger baseline EWC++.
- iCaRL [19]: Uses regularization over the activations and a nearest-exempler-based classifier. Here, iCaRL-hb1 refers to the *hybrid1* version, which uses the standard neural network classifier. Both the versions use previous samples.

Note, we use a few samples from the previous tasks to consolidate our baselines further in the single-head setting.

6.1 Results

We report the results in Tab. 1 where RWalk outperforms all the baselines in terms of average accuracy and provides better trade-off between forgetting and intransigence. We now discuss the results in detail.

Table 1: *Comparison with different baselines on MNIST and CIFAR in both multi-head and single-head evaluation settings. Baselines where samples are used are appended with '-S'. For MNIST and CIFAR, 10 (0.2%) and 25(5%) samples are used from the previous tasks using mean of features (MoF) based sampling strategy (refer Sec. 4.2).*

Methods	**MNIST**				**CIFAR**			
	Multi-head Evaluation							
	λ	$A_5(\%)$	F_5	I_5	λ	$A_{10}(\%)$	F_{10}	I_{10}
Vanilla	0	90.3	0.12	6.6×10^{-4}	0	44.4	0.36	0.02
EWC	75000	**99.3**	0.001	0.01	3×10^6	72.8	0.001	0.07
PI	0.1	**99.3**	0.002	0.01	10	73.2	0	0.06
RWalk (Ours)	1000	**99.3**	0.003	0.01	1000	**74.2**	0.004	0.04
	Single-head Evaluation							
Vanilla	0	38.0	0.62	0.29	0	10.2	0.36	-0.06
EWC	75000	55.8	0.08	0.77	3×10^6	23.1	0.03	0.17
PI	0.1	57.6	0.11	0.8	10	22.8	0.04	0.2
iCaRL-hb1	-	36.6	0.68	-0.01	-	7.4	0.40	0.06
iCaRL	-	55.8	0.19	0.46	-	9.5	0.11	0.35
Vanilla-S	0	73.7	0.30	0.03	0	12.9	0.64	-0.3
EWC-S	75000	79.7	0.14	0.22	15×10^5	33.6	0.27	-0.05
PI-S	0.1	78.7	0.24	0.05	10	33.6	0.27	-0.03
RWalk (Ours)	1000	**82.5**	0.15	0.14	500	**34.0**	0.28	-0.06

In the multi-head evaluation setting [25,14], except Vanilla, all the methods provide state-of-the-art accuracy with *almost zero* forgetting and intransigence (top row of Fig. 2). *This gives an impression that* IL *problem is solved.* However, as discussed in Sec. 2.1, this is an easier evaluation setting and does not capture the essence of IL.

However, in the single-head evaluation, *forgetting* and *intransigence* increase substantially due to the the inability of the network to differentiate among tasks. Hence, the performance significantly drops for all the methods (refer Tab. 1 and the middle row of Fig. 2). For instance, on MNIST, forgetting and intransigence of Vanilla deteriorates from 0.12 to 0.62, and 6.6×10^{-4} to 0.29, respectively, causing the average accuracy to drop from 90.3% to 38.0%. Although, regularized methods, EWC and PI, designed to counter catastrophic forgetting, result in less degradation of forgetting, their accuracy is still significantly worse - compare 99.3% of PI in multi-head against 57.6% in single-head. In Tab. 1, a similar performance decrease is observed on CIFAR-100 as well. Such a degradation in accuracy even with less forgetting shows that it is not only important to preserve knowledge (quantified by forgetting) but also to update knowledge (captured by intransigence) to achieve better performance. Task-level analysis for CIFAR dataset, similar to Fig. 2, is presented in the supplementary material.

We now show that even with a few representative samples intransigence can be mitigated. For example, in the case of PI on MNIST with only 10 ($\approx 0.2\%$) samples for each previous class, the intransigence drops from 0.8 to 0.05 which results in improving the average accuracy from 57.6% to 78.7%. Similar improvements can be seen for other

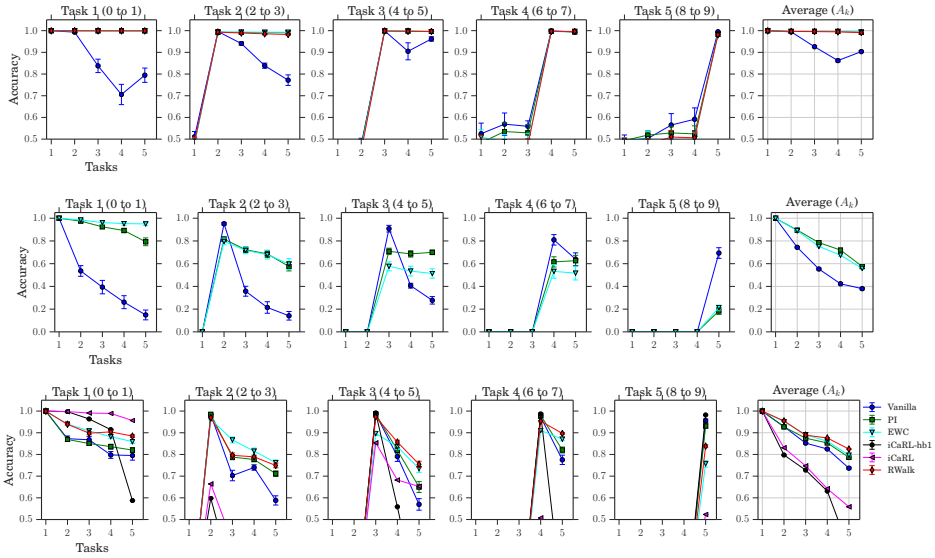
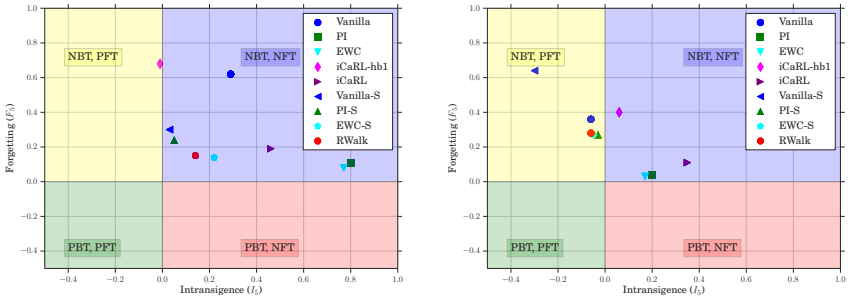


Fig. 2: *Accuracy on incremental MNIST with multi-head evaluation (**top**), and single-head evaluation without (**middle**) and with samples (**bottom**). First five columns show the variation in performance for different tasks,* e.g.*, the first plot depicts the performance variation on Task 1 when trained incrementally over five tasks. The last column shows the accuracy (A_k, refer Sec. 3). Mean of features (MoF) sampling is used.*

methods as well. On CIFAR-100, with only 5% representative samples, almost identical behaviour is observed.

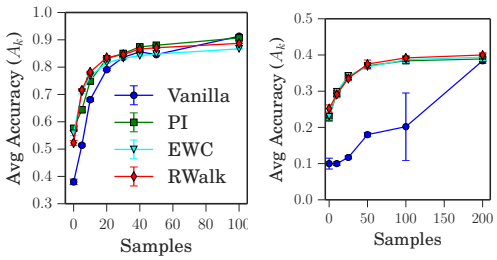
In our CIFAR-100 experiments (CNN instead of ResNet32), we note that the performance of iCaRL [19] is significantly worse than what has been reported by the authors. We believe this is due to the dependence of iCaRL on a highly expressive feature space, as both the regularization and the classifier depend on it. Perhaps, this reduced expressivity of the feature space due to the smaller network resulted in the performance loss.

Interplay of Forgetting and Intransigence In Fig. 3 we study the interplay of forgetting and intransigence in the single-head setting. Ideally we would like a model to be in the quadrant marked as *PBT, PFT* (*i.e.*, positive backward transfer and positive forward transfer). On MNIST, since all the methods, except iCaRL-hb1, lie on the top-right quadrant, hence for models with comparable accuracy, a model which has the smallest distance from $(0, 0)$ would be better. As evident, RWalk is closest to $(0, 0)$, providing a better trade-off between forgetting and intransigence compared to all the other methods. On CIFAR-100, the models lie on both the top quadrants and with the introduction of samples, all the regularized methods show positive forward transfer. Since the models lie on different quadrants, their comparison of forgetting and intransigence becomes application specific. In some cases, we might prefer a model that performs well on new tasks (better intransigence), irrespective of its performance on the old ones



(a) MNIST (b) CIFAR

Fig. 3: *Interplay between forgetting and intransigence.*



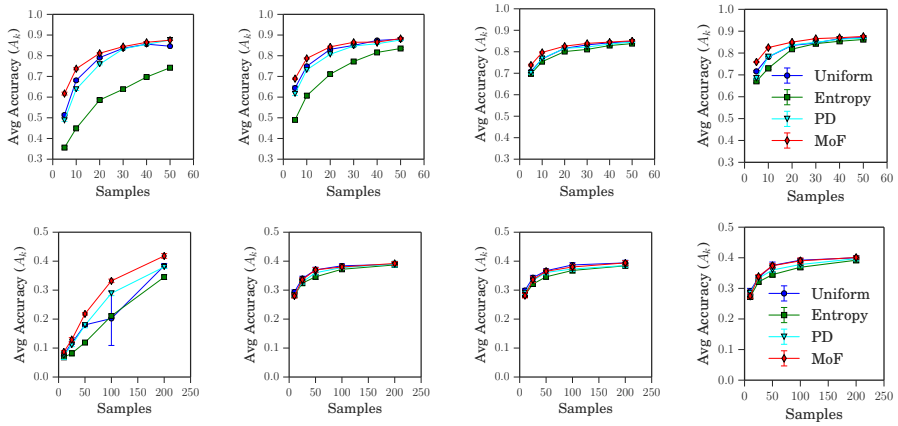
(a) MNIST (b) CIFAR

Fig. 4: *Comparison by increasing the number of samples. On MNIST and CIFAR each class has around 5000 and 500 samples, respectively. With increasing number of samples, the performance of Vanilla improved, but in the range where Vanilla is poor, RWalk consistently performs the best. Uniform sampling is used.*

(can compromise forgetting), and vice versa. Note that, RWalk maintains comparable performance to other baselines while yielding higher average accuracy on CIFAR-100.

Effect of Increasing the Number of Samples As expected, for smaller number of samples, regularized methods perform far superior compared to Vanilla (refer Fig. 4). However, once the number of samples are sufficiently large, Vanilla starts to perform better or equivalent to the regularized models. The reason is simple because now the Vanilla has access to enough samples of the previous tasks to relearn them at each step, thereby obviating the need of regularized models. However, in an IL problem, a fixed small-sized memory budget is usually assumed. Therefore, one cannot afford to store large number of samples from previous tasks. Additionally, for a simpler dataset like MNIST, Vanilla quickly catches up to the regularized models with small number of samples (20, 0.4% of total samples) but on a more challenging dataset like CIFAR it takes considerable amount of samples (200, 40% of total samples) of previous tasks for Vanilla to match the performance of the regularized models.

Comparison of Different Sampling Strategies In Fig. 5 we compare different subset selection strategies discussed in Sec. 4.2. It can be observed that for all the methods Mean-of-Features (MoF) subset selection procedure, introduced in iCaRL [19], performs the best. Surprisingly, *uniform* sampling, despite being simple, is as good as



(a) Vanilla (b) PI (c) EWC (d) RWalk

Fig. 5: *Comparison of different sampling strategies discussed in Sec. 4.2 on MNIST (**top**) and CIFAR-100 (**bottom**). Mean of features (MoF) outperforms others.*

more complex MoF, Plane Distance (PD) and entropy-based sampling strategies. Furthermore, the regularized methods remain insensitive to different sampling strategies, whereas in Vanilla, performance varies a lot against different strategies. We believe this is due to the unconstrained change in the last layer weights of the previous tasks.

7 Discussion

In this work, we analyzed the challenges in the incremental learning problem, namely, catastrophic forgetting and intransigence, and introduced metrics to quantify them. Such metrics reflect the interplay between *forgetting* and *intransigence*, which we believe will encourage future research for exploiting model capacity, such as, sparsity enforcing regularization, and exploration-based methods for incremental learning. In addition, we have presented an efficient version of EWC referred to as EWC++, and a generalization of EWC++ and PI with a KL-divergence-based perspective. Experimentally, we observed that these parameter regularization methods suffer from high intransigence in the practical *single-head* setting and showed that this can be alleviated with a small subset of representative samples. Since these methods are memory efficient compared to knowledge distillation-based algorithms such as iCaRL, future research in this direction would enable the possibility of incremental learning on segmentation tasks.

Acknowledgements

This work was supported by The Rhodes Trust, EPSRC, ERC grant ERC-2012-AdG 321162-HELIOS, EPSRC grant Seebibyte EP/M013774/1 and EPSRC/MURI grant EP/N019474/1.

References

1. Amari, S.I.: Natural gradient works efficiently in learning. Neural Computation (1998) 2, 4
2. Grosse, R., Martens, J.: A kronecker-factored approximate fisher matrix for convolution layers. In: ICML (2016) 4
3. Hecht-Nielsen, R., et al.: Theory of the backpropagation neural network. Neural Networks **1**(Supplement-1), 445–448 (1988) 9
4. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. In: NIPS (2014) 1
5. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. In: ICLR (2015) 10
6. Kirkpatrick, J., Pascanu, R., Rabinowitz, N.C., Veness, J., Desjardins, G., Rusu, A.A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., Hadsell, R.: Overcoming catastrophic forgetting in neural networks. Proceedings of the National Academy of Sciences of the United States of America (PNAS) (2016) 1, 2, 4, 6, 8, 9, 10
7. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. https://www.cs.toronto.edu/ kriz/cifar.html (2009) 2
8. Kullback, S., Leibler, R.A.: On information and sufficiency. The Annals of Mathematical Statistics (1951) 3
9. Le Roux, N., Pierre-Antoine, M., Bengio, Y.: Topmoumoute online natural gradient algorithm. In: NIPS (2007) 4
10. LeCun, Y.: The mnist database of handwritten digits. http://yann. lecun. com/exdb/mnist/ (1998) 2
11. Lee, J.M.: Riemannian manifolds: an introduction to curvature, vol. 176. Springer Science & Business Media (2006) 4
12. Lee, S.W., Kim, J.H., Ha, J.W., Zhang, B.T.: Overcoming catastrophic forgetting by incremental moment matching. In: NIPS (2017) 3, 10
13. Li, Z., Hoiem, D.: Learning without forgetting. In: ECCV (2016) 9
14. Lopez-Paz, D., Ranzato, M.: Gradient episodic memory for continuum learning. In: NIPS (2017) 1, 5, 6, 10, 11
15. Martens, J., Grosse, R.: Optimizing neural networks with kronecker-factored approximate curvature. In: ICML (2015) 4, 7
16. Nguyen, C.V., Li, Y., Bui, T.D., Turner, R.E.: Variational continual learning. ICLR (2018) 10
17. Pascanu, R., Bengio, Y.: Revisiting natural gradient for deep networks. In: ICLR (2014) 2, 4
18. Rebuffi, S.A., Bilen, H., Vedaldi, A.: Learning multiple visual domains with residual adapters. In: NIPS (2017) 9
19. Rebuffi, S.V., Kolesnikov, A., Lampert, C.H.: iCaRL: Incremental classifier and representation learning. In: CVPR (2017) 1, 3, 9, 10, 12, 13
20. Rusu, A.A., Rabinowitz, N.C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., Hadsell, R.: Progressive neural networks. arXiv preprint arXiv:1606.04671 (2016) 9
21. Schwarz, J., Luketina, J., Czarnecki, W.M., Grabska-Barwinska, A., Teh, Y.W., Pascanu, R., Hadsell, R.: Progress & compress: A scalable framework for continual learning. In: ICML (2018) 2, 7, 10
22. Shin, H., Lee, J.K., Kim, J., Kim, J.: Continual learning with deep generative replay. In: NIPS (2017) 10
23. Terekhov, A.V., Montone, G., ORegan, J.K.: Knowledge transfer in deep block-modular neural networks. In: Conference on Biomimetic and Biohybrid Systems. pp. 268–279 (2015) 9

24. Yoon, J., Yang, E., Lee, J., Hwang, S.J.: Lifelong learning with dynamically expandable networks. In: ICLR (2018) 9
25. Zenke, F., Poole, B., Ganguli, S.: Continual learning through synaptic intelligence. In: ICML (2017) 1, 2, 6, 7, 8, 9, 10, 11