Riemannian Walk for Incremental Learning: Understanding Forgetting and Intransigence

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

Department of Engineering Science,
University of Oxford
arslan.chaudhry@eng.ox.ac.uk, {puneet,ajanthan}@robots.ox.ac.uk,
philip.torr@eng.ox.ac.uk

Abstract. We study incremental learning for the classification task, a key component for life-long learning systems. For an incremental learning algorithm, the main challenges are to update the classifier whilst preserving previous knowledge. In addition to *forgetting*, a well-known issue while preserving knowledge, we observe that incremental learning algorithms also suffer from a crucial problem of *intransigence*, its inability to update knowledge. First, we introduce two metrics to quantify *forgetting* and *intransigence* that allow us to understand, analyse, and gain better insights into the behaviour of an incremental learning algorithm. Second, we present a generalization of EWC [7] and Path Integral [25], with a theoretically grounded KL-divergence based perspective. We thoroughly analyse and compare the behaviour of different incremental learning algorithms on MNIST and CIFAR-100 datasets. We obtain superior results for our method in terms of accuracy, and provide better trade-off for forgetting and intransigence.

Keywords: Incremental Learning, Continual Learning, Catastrophic Forgetting
[1]

1 Introduction

One challenging problem in realizing human-level intelligence is developing systems that can learn new tasks continually while preserving existing knowledge. However, the current AI systems are built to perform well in one particular task (*e.g.*, classification or segmentation of a *fixed* number of objects), and have minimal flexibility to adapt to new scenarios. In such a typical offline learning paradigm, once a model is learned for hundreds of tasks, adding a new task requires retraining the model for all the tasks. This would not only require tremendous amount of storage but also a huge training time. For incremental learning, on the other hand, the objective is to train the model just for the new task while preserving existing knowledge.

To accomplish this, the focus is on developing algorithms flexible enough to update their parameters to accommodate new tasks online, while still performing well on the previous tasks. While designing such algorithms, knowledge in a neural network is usually defined either using the output activations [5,21] or the network parameters [7,25]. Once the knowledge is defined, the task of preserving and updating the knowledge has two inherent problems: (1) *catastrophic forgetting*: forgetting previous knowledge; and (2) *intransigence*: inability to update previous knowledge in order to learn the current

[1] * Equal contribution

task. Both of these problems require differing solutions and pose a trade-off for any incremental learning algorithm.

Consequently, we advocate that an incremental learning algorithm must be evaluated based on its performance in the *past* and the *present* tasks in order to be confident about its behaviour in the *future* unseen tasks. In fact such measures would accurately reflect whether the algorithm utilizes the model capacity efficiently or it learns on some tasks while not learning on others. To this end, we introduce two metrics; *forgetting*: how much the algorithm has forgotten what it has learned, and *intransigence*: how inflexible it is to learn the task at present. These metrics together with the standard multi-class average accuracy allow us to understand, analyse, and gain better insights about the behaviour of various incremental learning algorithms.

In addition to this, we present a generalization of two recently proposed incremental learning algorithms, Elastic Weight Consolidation (EWC) [7], and Path Integral (PI) [25]. In particular, first we show that in EWC, while learning for the new task, the model's likelihood distribution is regularized using the well known second-order approximation of KL-divergence [1,19], which is equivalent to computing distance in a Riemannian manifold induced by the Fisher Information Matrix [1]. Next, inspired by the PI approach, we introduce a *parameter importance score* that is proportional to the sensitivity of the classification loss with respect to the KL-divergence (distance in the Riemannian manifold) between the corresponding output probability distributions. By accumulating the score over the optimization trajectory, information about previous tasks is effectively retained. Then, we combine the KL-divergence-based regularization with the parameter importance-based regularization, while modifying Fisher and score computations such that the combined model is efficient to train (no additional overhead) and less sensitive to the regularization hyperparameter.

Furthermore, in a more practical setting, where the prediction space consists of all the classes encountered during learning, along with *forgetting*, incremental learning methods suffer from high *intransigence*. To tackle this, we study different *sampling* strategies that store a small representative subset ($\leq 5\%$) of the dataset from 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. We present a thorough analysis to better understand the behaviour of incremental learning algorithms on MNIST [11] and CIFAR-100 [8] datasets in which our method yields superior results. Our main contributions in the paper are:

1. New evaluation metrics - *Forgetting* and *Intransigence* - to better understand the behaviour and performance of an incremental learning algorithm.
2. Generalization of recent methods, namely, EWC [7] and Path Integral [25], with theoretically grounded KL-divergence based perspective, providing new insights.
3. Analysis of different methods in terms accuracy, forgetting, and intransigence.

2 Problem Set-up and Preliminaries

Here we explain the problem set-up 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 distance in the Riemannian manifold, both of which are crucial to our approach.

2.1 Single-head vs Multi-head Evaluations

In an incremental learning problem, a stream of tasks is received where every new task consists of data with corresponding labels. Let $\mathcal{D}_k = \{(\mathbf{x}_i^k, y_i^k)\}_{i=1}^{n_k}$ be the dataset corresponding to the k-th task, where $\mathbf{x}_i^k \in \mathcal{X}$ is the input sample and $y_i^k \in \mathbf{y}^k$ the ground truth label. For example, $\mathbf{x}_i^k$ could be an image and $\mathbf{y}^k$ a set of labels specific to the k-th task. Under this setting, at the k-th task, the objective is to learn a prediction function $f_\theta : \mathcal{X} \rightarrow \mathcal{Y}^k$, where $\theta \in \mathbb{R}^P$ are the parameters of f (a neural network) and $\mathcal{Y}^k$ is either $\mathbf{y}^k$ (multi-head) or $\cup_{j=1}^k \mathbf{y}^j$ (single-head).

Note that, in the case of *single-head* [14,21], even though only $\mathcal{D}_k$ is available for the training of the k-th task, the output space consists of all the labels seen previously. This is in contrast to the standard *multi-head* evaluation criteria [7,25] where $\mathcal{Y}^k = \mathbf{y}^k$, *i.e.*, the output space consists of only the labels corresponding to the k-th task. Multi-head evaluation is much easier and less practical than single-head. For example, if we divide the MNIST dataset into five subsets (for 5 tasks) as follows: $\{\{0,1\}, \cdots, \{8,9\}\}$. Then, at the 5-th task, for a given image, the *multi-head* would predict a class out of two labels $\{8,9\}$, however, the *single-head* would have to predict a class out of eight classes $\{0, \cdots, 9\}$. Thus, in single-head, the classifier must learn to distinguish labels from different tasks as well. Also, single-head is more practical as knowing a priori the subset of labels to look at is a big assumption. For example, if the new task contains only one label, multi-head setting would be equivalent to knowing the ground truth label a priori.

2.2 Probabilistic Interpretation of Neural Networks

In probabilistic sense, for a multi-class classification task, each index of the soft-max output of a neural network can be interpreted as the success probability corresponding to that index (or label). 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 defined as the sample obtained from the likelihood distribution $p_\theta(\mathbf{y}|\mathbf{x})$. Typically, instead of doing such sampling, a label with the highest soft-max probability is chosen as the network 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 in the supplementary.

2.3 KL-divergence as Distance in the Riemannian Manifold

Let $D_{KL}(p_\theta \| p_{\theta+\Delta\theta})$ be the KL-divergence [9] between the conditional likelihood of a neural network (could be any probability distribution) at θ and $\theta + \Delta\theta$, respectively. Then, assuming $\Delta\theta \rightarrow 0$, the second-order Taylor approximation of KL-divergence can be written as $D_{KL}(p_\theta \| p_{\theta+\Delta\theta}) \approx \frac{1}{2}\Delta\theta^\top F_\theta \Delta\theta = \frac{1}{2}\|\Delta\theta\|_{F_\theta}^2$ [2], where F_θ, known as the

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

empirical Fisher Information Matrix [1,19] 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 also makes it highly attractive for second-order optimization techniques [1,19,2,10,17]. Thus, when $\Delta\theta \to 0$, computing KL-divergence $\frac{1}{2}\|\Delta\theta\|^2_{F_\theta}$ is equivalent to computing *distance* in a Riemannian manifold[3] [13] 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 [7], 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, it is expensive to compute F_θ as it requires a full forward-backward pass over the dataset.

3 Forgetting and Intransigence

We now develop tools to better understand an incremental learning algorithm. Since the objective of an incremental learning algorithm is to keep on learning new tasks, it must be evaluated based on the model's performance on the *past* and the *present* tasks in order to be confident about its behaviour in the *future* unseen tasks. Along with average accuracy (normally enough to evaluate standard classification task), there are two additional crucial components that we must quantify – (1) *forgetting*: how much an algorithm forgets what it has 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, it will have less forgetting but much higher intransigence. If, in contrast, the regularization is too weak (*e.g.*, *vanilla*), while the intransigence will be very low, the model will suffer from catastrophic forgetting. Ideally, we would want a model that suffers less in both, thus efficiently utilizing the finite model capacity. In contrast, if one observes high negative correlation between forgetting and intransigence (usually the case) then, it suggests that the model is saturated (in terms of model capacity) or the method does not effectively utilize the model capacity. Before defining metrics for quantifying forgetting and intransigence, we first give the multi-class average accuracy which will be the basis for defining these metrics.

Average Accuracy (A) Let $a_{k,j} \in [0,1]$ be the accuracy (fraction of correctly classified images) evaluated on the held-out set of the j-th task ($j \le k$) after training the network incrementally from tasks 1 to k. Note that, to compute $a_{k,j}$, the output space

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

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 defined as:

$$A_k = \frac{1}{k}\sum_{j=1}^{k} a_{k,j} \ . \tag{3}$$

The higher the A_k the better the classifier, but this does not provide any information about forgetting or intransigence profile of the learning algorithm which would be crucial to judge the behaviour of the algorithm. In what follows, we define these two metrics and provide insights about them.

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

$$f_j^k = \max_{l\in\{1,\cdots,k-1\}} a_{l,j} - a_{k,j} \ , \quad \forall j < k \ . \tag{4}$$

Here we quantify knowledge using accuracy, however, depending on the task, other metrics could be used (*e.g.*, IoU for object segmentation). Note, $f_j^k \in [-1, 1]$ is defined for $j < k$ as we are interested in quantifying forgetting for *previous* tasks.

Positive/Negative backward transfer (P/N)BT: $f_j^k < 0$ implies PBT, which means that learning task k helped improve the knowledge about previous task j. Similarly, $f_j^k > 0$ (NBT) implies learning task k forced the model to forget some information about task j. Thus, f_j^k helps us to understand how incrementally training the model impacts knowledge about the past tasks.

By normalizing against the number of tasks seen previously, the average forgetting at k-th task is written as:

$$F_k = \frac{1}{k-1}\sum_{j=1}^{k-1} f_j^k \ . \tag{5}$$

Lower F_k implies less forgetting on previous tasks. If $\max_{l\in\{1,\cdots,k-1\}} a_{l,j}$ is replaced with $a_{j,j}$, then F_k is similar to the negative of Backward Transfer (BT) [16], but F_k is more generic than BT as it captures both PBT and NBT.

Intransigence Measure ($\boldsymbol{I}$) We define intransigence as the *inability* or *rigidity* to learn new tasks or update knowledge. In incremental learning, there can be two potential reasons for intransigence: (1) too rigid a regularization for the old tasks; and (2) unavailability of data from the previous tasks. The rigidity of the regularization can be handled by adjusting the attached hyperparameter. However, storing few samples, depending on the memory budget, requires judiciously selecting representative samples (refer Sec. 4.2 for more detail). In the single-head setting, we notice that the unavailability of the samples from the previous tasks dramatically increases the confusion among the current and the previous task labels. This is due to the fact that, while training for the current task, the model has no means to learn to differentiate it from the previous tasks. This is

less observed in the multi-head setting as the output label of each task is independent of each other. Experimentally we show that storing just a few representative samples from the previous tasks not only improves intransigence but also improves forgetting.

Since we wish to quantify the *inability* to learn, we need a reference to what could be learned had we trained our model in an *ideal* setting. We assume that *ideal* setting to be the standard classification with access to all the datasets at all times. Thus, 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:

$$I_k = a_k^* - a_{k,k} \ , \tag{6}$$

where $a_{k,k}$ denotes the accuracy on the k-th task when trained up to task k. Similar to forgetting, $I_k \in [-1, 1]$, and lower the I_k the better the model is. Notice that, the reference/target model must be defined depending on the feasibility to obtain it. In situations where it is highly expensive to obtain, an approximation can be used.

Positive/Negative forward transfer (P/N)FT: $I_k < 0$ implies PFT, which means learning incrementally up to task k positively influences model's knowledge about it. This is not to be confused with the Forward Transfer (FT) reported in [16]. Similarly, $I_k > 0$ implies NFT, which means incrementally learning had negative influence on task k.

4 Riemannian Walk for Incremental Learning

We now describe our generalization to EWC [7] and PI [25]. Briefly, our approach has three key components: (1) KL-divergence-based regularization over the conditional likelihood $p_\theta(\mathbf{y}|\mathbf{x})$; (2) parameter importance score based on the sensitivity of loss over the movement in the Riemannian manifold; and (3) sampling strategies to handle intransigence. The first two components mitigate the effects of catastrophic forgetting, whereas the third component handles intransigence.

4.1 Avoiding Catastrophic Forgetting

KL-divergence-based Regularization We regularize over the conditional likelihood distributions $p_\theta(\mathbf{y}|\mathbf{x})$ using the approximate KL-divergence, Eq. (2), as the distance measure. Intuitively, the use of KL-divergence would allow the network to learn parameters such that the conditional likelihood distributions of the new and previous tasks are close to each other. This would preserve the inherent properties of the model about previous tasks as the learning progresses. More precisely, given parameters θ^{k-1} trained sequentially from task 1 to $k-1$, and $\mathcal{D}_k$ for the k-th task, our objective is:

$$\underset{\theta}{\operatorname{argmin}} \, \tilde{L}^k(\theta) := L^k(\theta) + \lambda \, D_{KL}\left(p_{\theta^{k-1}}(\mathbf{y}|\mathbf{x}) \| p_\theta(\mathbf{y}|\mathbf{x})\right) \ , \tag{7}$$

where, λ is the 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 this regularization is the same as of EWC [7]. *However, we present it from the KL-divergence-based perspective*. Recall, the Fisher component at θ^{k-1} corresponding to the i-th parameter, $F_{\theta_i^{k-1}}$, is the *expected square of the loss gradient* w.r.t. the i-th parameter (see Eq. (1)).

Furthermore, when θ^{k-1} is at a local minimum, gradients would be nearly zero, making Fisher very small. Theoretically, the regularization is negligible, which would result in catastrophic forgetting. However, experimentally we observed that this can be circumvented by using a very high value ($\approx 10^6$) for the hyperparameter λ. However, to completely avoid this issue in situations where gradients are very small (increased robustness) and to capture model's behaviour during training, we add another optimization-path-based parameter importance score (positive) to each component of Fisher, which we explain next.

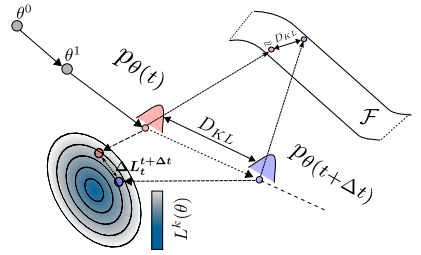


Fig. 1: *Parameter importance accumulated over the optimization trajectory where the importance is the sensitivity of the classification loss with respect to the KL-divergence approximated using the distance in the Riemannian manifold.*

Optimization Path-based Parameter Importance In KL-divergence-based regularization, the Fisher component $F_{\theta_i^{k-1}}$ can be seen as the importance score for the i-th parameter at θ^{k-1}. Since Fisher captures the intrinsic properties of the model only *at the minimum*, it is blinded towards the influence of parameters over the optimization path. Motivated by [25], we accumulate task-specific parameter importance over the entire training trajectory. In this case, the parameter importance 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.

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:

$$L(\theta(t+\Delta t)) - L(\theta(t)) \approx -\sum_{i=1}^{P}\sum_{t=t}^{t+\Delta t} \frac{\partial L}{\partial \theta_i} (\theta_i(t+1) - \theta_i(t)) = -\sum_{i=1}^{P} \Delta L_t^{t+\Delta t}(\theta_i), \quad (8)$$

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 $s_{t_1}^{t_2}(\theta_i) = \sum_{t=t_1}^{t_2} \frac{\Delta L_t^{t+\Delta t}(\theta_i)}{F_{\theta_i}^t \Delta\theta_i(t)^2 + \epsilon}$, where $\Delta\theta_i(t) = \theta_i(t+\Delta t) - \theta_i(t)$ and $\epsilon > 0$ is to prevent division by zero. The denominator is computed at every discrete intervals of $\Delta t \geq 1$ and $F_{\theta_i}^t$ is computed at every t-th step as described towards the end of this section. The computation of this importance score is illustrated in Fig. 1. Since we care about positive influence of the parameters, negative scores are set to zero. Note that, if the Euclidean distance is used in place of the KL-divergence-based distance, the score $s_{t_1}^{t_2}(\theta_i)$ will be similar to the one obtained using the path integral approach [25].

Final Objective Function Combining the Fisher information matrix and the optimization-path based importance scores we define our final objective function as:

$$\tilde{L}^k(\theta) = L^k(\theta) + \lambda \sum_{i=1}^{P} (F_{\theta_i^{k-1}} + s_{t_0}^{t_{k-1}}(\theta_i))(\theta_i - \theta_i^{k-1})^2 \ . \tag{9}$$

Here, $s_{t_0}^{t_{k-1}}(\theta_i)$ is the importance accumulated from the first training iteration t_0 until the last training iteration t_{k-1} corresponding to task $k-1$. Note that, since the scores are accumulated over time, the regularization becomes 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, which would retain the influence of both the terms. This can be ensured by individually normalizing then 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 [7] and PI [25] are highly sensitive to λ, making them relatively less reliable for incremental learning. Note that, at any point, the additional space complexity is $\mathcal{O}(P)$ as P-dimensional vectors of θ^{k-1}, $s_{t_0}^{t_{k-1}}$ and $F_{\theta^{k-1}}$ are stored. In what follows, we show how Fisher can be approximately computed highly efficiently, instead of computing it using additional forward-backward pass over the entire dataset at the optimum.

Efficiently Updating Fisher Information Matrix At a given θ, estimating empirical Fisher F_θ requires us to compute the expectation over the entire dataset, which is expensive. Instead, we use *exponential moving average* to keep on updating the Fisher matrix similar to [17]. Thus, given F_θ^{t-1} at $t-1$, the Fisher is updated as follows:

$$F_\theta^t = \alpha F_\theta^t + (1-\alpha) F_\theta^{t-1} \ , \tag{10}$$

where F_θ^t is the Fisher matrix obtained using the current batch and $\alpha \in [0, 1]$ is a hyperparameter. Computing Fisher in this manner contains information about previous tasks, and also eliminates the additional forward-backward pass, contrary to computing Fisher at the minimum of a task.

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 the *confusion* at the 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 [21]. 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 [21].

Uniform Sampling A naive 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 we are regularizing over the parameters, 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 soft-max output 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 [21] 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 both 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 [12,20,22,24]. 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 [4]. This entails regularizing either over the activations (output) [21,15] or over the network parameters [7,25]. 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 both EWC [7] and path integral [25]. Similar in spirit to regularization over the parameters, Lee *et al*. [14] use moment matching to obtain network weights as the combination of the weights of all the tasks, and Nguyen *et al*. [18] enforce the distribution over the model parameters to be close via a Bayesian framework. Different from the above approaches, Lopez-Paz *et al*. [16] update gradients such that the losses of the previous tasks do not increase, while Shin *et al*. [23] resort to a retraining strategy where the samples of the previous tasks are generated using a learned generative model.

6 Experiments

We will now describe several experiments to analyse the performance of different incremental learning algorithms.

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 [7]	75000	**99.3**	0.001	0.01	3×10^6	72.8	0.001	0.07
PI [25]	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 [7]	75000	55.8	0.08	0.77	3×10^6	23.1	0.03	0.17
PI [25]	0.1	57.6	0.11	0.8	10	22.8	0.04	0.2
iCaRL-hb1 [21]	-	36.6	0.68	-0.01	-	7.4	0.40	0.06
iCaRL [21]	-	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

Datasets We evaluate baselines and our proposed model named, Riemannian Walk (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 in our experiments are similar to the ones used in [25]. In particular, for MNIST, we use a fully-connected neural network with two hidden layers. Each layer has 256 units with ReLU nonlinearites. For CIFAR-100, we use a convolutional neural network (CNN) with four convolutional layers followed by a single dense layer. Further details of the convolutional network, such as, kernel size, number of filters, *etc*., are given in the supplementary material. All the baselines and proposed model are trained with Adam optimizer [6] (learning rate = 1×10^{-3}, $\beta_1 = 0.9$, $\beta_2 = 0.999$) with a fixed batch size of 64. Note that, having relatively smaller CNN for CIFAR allowed us to do extensive experiments covering different aspects of incremental learning algorithms.

Baselines We compare RWalk against the following baselines in both multi-head and single-head settings:

– Vanilla: Network trained with standard multi-class cross-entropy loss without any regularization.

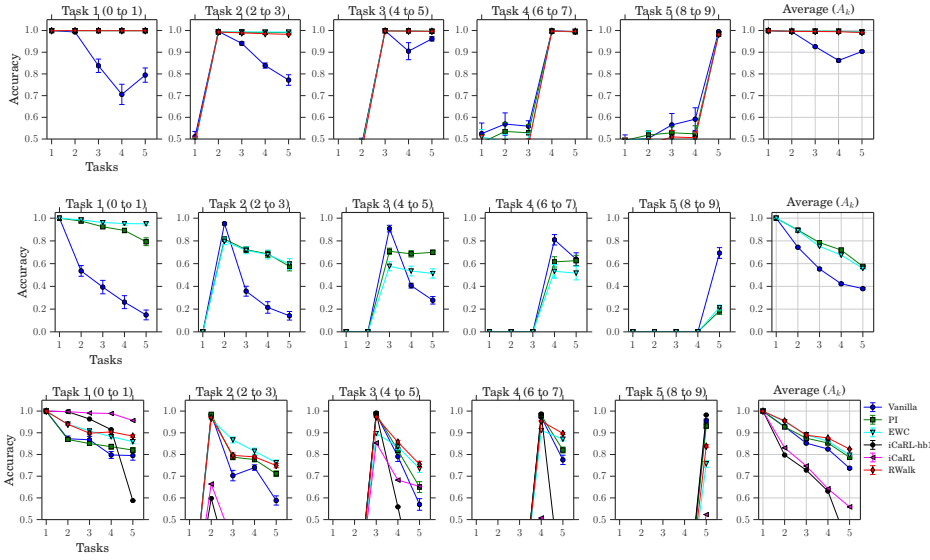


Fig. 2: *Accuracy measure in incremental MNIST with multi-head evaluation (**top**), and the single-head evaluation without (**middle**) and with samples (**bottom**). The first five columns show how the performance of different tasks vary as the model is trained for new tasks,* e.g.*, the first plot depicts the variation in performance on Task 1 when the network is sequentially trained for the five tasks in an incremental manner. The last column shows the average accuracy measure (A_k, by varying k) as defined in Eq.* (3)*. Mean of features (MoF) sampling is used. (best viewed in color)*

- EWC [7] and PI [25]: Use parameter regularization to avoid catastrophic forgetting.
- iCaRL [21]: Uses regularization over the activations and a nearest-exempler-based classifier. Here, iCaRL-hb1 refers to the *hybrid1* version of iCaRL, which uses the standard neural network classifier. The model makes explicit use of samples from previous tasks.

To consolidate our baselines further, we make use of samples from previous tasks and show that even a small amount of such samples is sufficient to considerably improve the baselines in the single-head setting. Note that, we do not use samples in case of a multi-head setting.

6.1 Results

We report the results of all the baselines in both the multi- and single-head settings in Tab. 1. In the multi-head evaluation setting, as used by [25,16], all the regularized methods perform almost the same (see the top row of Fig. 2) with state-of-the-art accuracy, and *almost zero* forgetting and intransigence. *This gives an impression that incremental learning problem is solved.* However, as discussed in Sec. 2.1, this is an easier evaluation setting and has a less practical value in an incremental learning set-up where the subset of labels to look at for a sample is not given a priori.

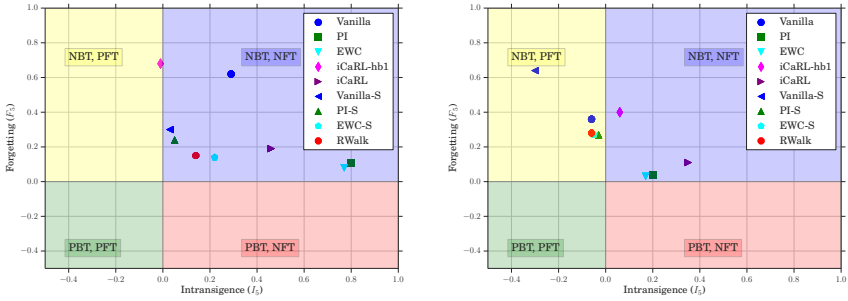
If, however, we switch to the single-head evaluation setting, the inability of the network to differentiate among the tasks results in a substantial increase in *forgetting* and

intransigence. Due to this, it can be seen in Tab. 1 and the middle row of Fig. 2, the performance of all the methods drops significantly. For instance, on MNIST in Vanilla, the forgetting and intransigence 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 [7] and PI [25], designed to counter catastrophic forgetting, result in less degradation of forgetting, their accuracy is still significantly less - compare 99.3% of PI [25] in the multi-head setting with 57.6% in the single-head setting on MNIST. In Tab. 1, a similar performance decrease can be observed on CIFAR-100 as well. This shows that it is not only important to just forget less about the previous knowledge (quantified by forgetting and achieved through regularization) but also to update the knowledge (captured by intransigence) to achieve better performance (quantified by average accuracy). Task-level analysis for CIFAR dataset, similar to Fig. 2 for MNIST, is presented in the supplementary material.

Now, we show that, with access to a small amount of samples from previous classes the network can update its knowledge about the current task thereby alleviating the intransigence resulting in improved performance. On MNIST, with only 10 samples for each class (almost 0.2% of the total examples of the class), the average accuracy increases from 38.0% to 73.7% for Vanilla and 57.6% to 78.7% for PI [25]. Similarly, on CIFAR-100 we see a $+10.5\%$ and $+10.8\%$ increase in the average accuracy by using only 5% of the samples in EWC [7] and PI [25] respectively. From this it can be concluded that, in the practical single-head setting, where the task information for a sample is not available at test time, the algorithm needs to have access to at least a few examples from the past tasks so that it can differentiate among the current and previous tasks. The efficacy of Intransigence measure (I_k) is evidenced in Tab. 1, where it decreases (improves) sharply with sampling, thus adequately capturing the rigidity of an incremental learner to adapt to new tasks.

In Tab. 1 we also provide the results of our proposed approach - RWalk. As can be seen that our method outperforms even the *stronger baselines* (with samples) on both MNIST and CIFAR-100. Empirically, we conclude that, RWalk, which captures the importance of parameters both along the training trajectory and at the minimum of each task is a strict generalization of both EWC [7] and PI [25]. We also note that the performance of iCaRL [21] in our experiments (smaller CNN for CIFAR-100) is significantly worse than what has been reported by the authors on a comparatively larger ResNet32 network. We believe this is due to the dependence of iCaRL on a highly expressive feature space, as both the regularization and classifier of the iCaRL depends on the feature map. This reduced expressivity of the feature space due to the smaller network, resulted in the loss of performance of iCaRL. However, on MNIST where the feature space of the fully-connected network is good enough for the dataset, iCaRL performs reasonably well, but still considerably less than the other baselines with access to the same amount of samples.

Interplay of Forgetting and Intransigence In Fig. 3 we study the interplay of forgetting and intransigence among different models with and without sampling in the single-head setting. As discussed in Sec. 3, ideally we would like the model to be in the quadrant marked as *PBT, PFT* (*i.e.*, positive backward transfer and positive forward transfer). In our experiments, none of the models exhibited positive backward trans-



(a) MNIST (b) CIFAR

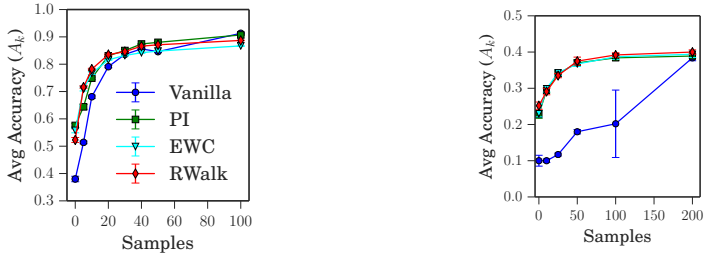
Fig. 3: *Interplay of forgetting and intransigence for different models. Ideal model will be in the bottom-left quadrant (PBT, PFT). In our experimental setup, the closer the model is to origin* $(0, 0)$ *the better. (best viewed in color)*

fer that could have occurred had there been a significant overlap among the tasks. On MNIST, since all the methods, except iCaRL-hb1, lie in the top-right quadrant, hence for a fixed accuracy, a model which has the smallest distance from $(0, 0)$ would be better. As observed in Fig. 3, RWalk is closest to $(0, 0)$, and hence finds a *better trade-off in forgetting and intransigence* among all the other methds. In Fig. 3, it can be seen that on CIFAR-100 the models lie in both the top quadrants and *with the introduction of samples, all the regularized methods show positive forward transfer.* Now, since the models lie in different quadrants, their comparison of forgetting and intransigence becomes application specific, meaning, whether learning the present task is more important or not forgetting what the model has already learned is vital for desired performance in the application. Note that, even on CIFAR-100, RWalk maintains comparable performance to other baselines while yielding higher average accuracy.

Effect of Increasing the Number of Samples We now discuss the effect of increasing the number of samples from the previous tasks on different methods (refer Fig. 4). As expected, for smaller number of samples, regularized methods perform far superior to Vanilla. 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 model has access to enough samples of the previous tasks to relearn them at each step, thereby obviating the need of regularized models. But since in an incremental learning system a fixed small-sized memory budget is assumed, therefore, we cannot afford to store large number of samples from previous tasks.

Additionally, note that, 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. It is obvious to not to compare the regularized models once Vanilla crosses them. As can be seen in Fig. 4, when the performance of Vanilla is significantly less than the regularized methods, RWalk is consistently better than the baselines.

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



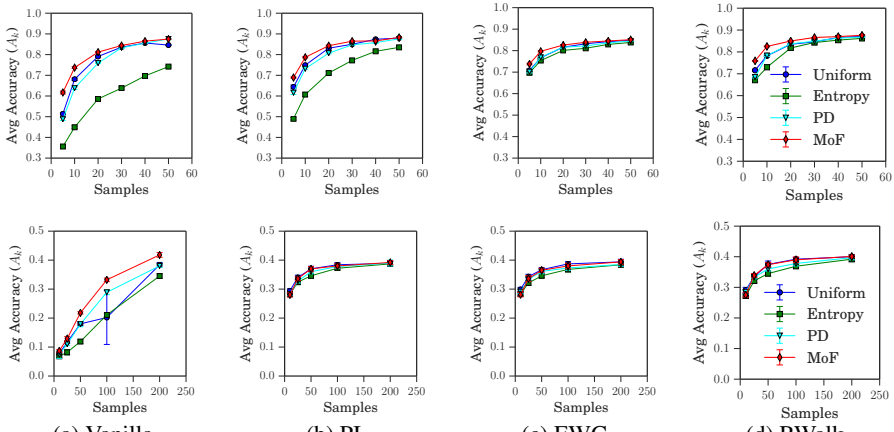
(a) MNIST (b) CIFAR

Fig. 4: *Performance of different methods as the number of samples are increased. On MNIST and CIFAR each class has around 5000 and 500 total samples, respectively. As the number of samples are increased Vanilla becomes the best performing method. In the samples range where performance of Vanilla is significantly less than the regularized methods, RWalk consistently performs the best. Uniform sampling is used. (best viewed in color)*

Mean-of-Features (MoF) subset selection procedure, introduced in iCaRL [21], performs better than other strategies. Surprisingly, *uniform* sampling, despite being simple, is as good as more complex MoF, Plane Distance (PD) and entropy-based sampling strategies. Furthermore, in Fig. 5 it can be seen that regularized methods remain insensitive to different sampling strategies, whereas in Vanilla method, 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 in Vanilla model. Since the subset size is small, the weight updates becomes sensitive to the sample we pick. In PI, EWC and RWalk, this problem is avoided by regularizing over the previous task weights.

7 Discussion

In this work, we have analyzed the challenges in an incremental learning problem, namely, catastrophic forgetting and intransigence; and introduced metrics to quantify them. Such metrics, as demonstrated by our experiments, accurately reflect the interplay between *forgetting* and *intransigence* and provide means to get an idea about whether the incremental learning algorithm utilizes the finite model capacity efficiently. This, we believe, encourage future research in the direction of exploiting model capacity, such as, sparsity enforcing regularization, and exploration-based methods for incremental learning. In addition, we have presented a generalization of recently proposed EWC [7] and PI [25] with a KL-divergence-based perspective to EWC. 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 [21], future research in this direction would enable the possibility of incremental learning on segmentation tasks.



(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) introduced by [21] consistently performs better than others in all the methods for varying number of samples. (best viewed in color)*

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.

Supplementary Material

For the sake of completeness, we first give more details on the KL-divergence approximation using Fisher information matrix (Sec. 2.3). In particular, we give the proof of KL approximation, $D_{KL}(p_\theta \| p_{\theta+\Delta\theta}) \approx \frac{1}{2}\Delta\theta^\top F_\theta \Delta\theta$, discuss the difference between the true Fisher and the empirical Fisher[4], and explain why the Fisher goes to zero at a minimum. Later, in Sec. B, we provide a comparison with GEM [16] - recently proposed continual learning method based on gradient projections that uses samples from previous tasks to avoid forgetting. Note that, GEM comparison is not available in the main paper. We observe that we obtain better average accuracy (A_k) on CIFAR-100 with a simpler CNN (**79.6**%) and with less number of samples compared to GEM (63.3%) *which uses ResNet-18*. Additionally, we discuss the sensitivity of different models to the regularization hyperparameter (λ) in Sec. C. Finally, we conclude in Sec. D with the details of the architecture and task-based analysis of the network used for CIFAR-100 dataset. *We note that with additional experiments and further analysis in this supplementary the conclusions of the main paper hold.*

[4] By Fisher, we always mean the empirical Fisher.

A Approximate KL divergence using Fisher Information Matrix

A.1 Proof of Approximate KL divergence

Lemma 1. *Assuming $\Delta\theta \to 0$, the second-order Taylor approximation of KL-divergence can be written [1,19] as:*

$$D_{KL}(p_\theta \| p_{\theta+\Delta\theta}) \approx \frac{1}{2}\Delta\theta^\top F_\theta \Delta\theta \,, \tag{11}$$

where F_θ is the empirical Fisher at θ.

The proof is similar to the one presented in [19].

Proof. The KL divergence is defined as:

$$D_{KL}(p_\theta(\mathbf{z}) \| p_{\theta+\Delta\theta}(\mathbf{z})) = \mathbb{E}_{\mathbf{z}}\left[\log p_\theta(\mathbf{z}) - \log p_{\theta+\Delta\theta}(\mathbf{z})\right] \,. \tag{12}$$

Note that we use the shorthands $p_\theta(\mathbf{z}) = p_\theta(\mathbf{y}|\mathbf{x})$ and $\mathbb{E}_{\mathbf{z}}[\cdot] = \mathbb{E}_{\mathbf{x}\sim\mathcal{D},\mathbf{y}\sim p_\theta(\mathbf{y}|\mathbf{x})}[\cdot]$. We denote partial derivatives as column vectors. Let us first write the second order Taylor series expansion of $\log p_{\theta+\Delta\theta}(\mathbf{z})$ at θ:

$$\log p_{\theta+\Delta\theta} \approx \log p_\theta + \Delta\theta^\top \frac{\partial \log p_\theta}{\partial \theta} + \frac{1}{2}\Delta\theta^\top \frac{\partial^2 \log p_\theta}{\partial \theta^2}\Delta\theta \,. \tag{13}$$

Now, by substituting this in Eq. (12), the KL divergence can be approximated as:

$$\begin{aligned} D_{KL}(p_\theta \| p_{\theta+\Delta\theta}) &\approx \mathbb{E}_{\mathbf{z}}[\log p_\theta] - \mathbb{E}_{\mathbf{z}}[\log p_\theta] \qquad\qquad (14a)\\ &\quad - \Delta\theta^\top \mathbb{E}_{\mathbf{z}}\left[\frac{\partial \log p_\theta}{\partial \theta}\right] - \frac{1}{2}\Delta\theta^\top \mathbb{E}_{\mathbf{z}}\left[\frac{\partial^2 \log p_\theta}{\partial \theta^2}\right]\Delta\theta \,, \\ &= \frac{1}{2}\Delta\theta^\top \mathbb{E}_{\mathbf{z}}\left[-\frac{\partial^2 \log p_\theta}{\partial \theta^2}\right]\Delta\theta \quad \text{see Eq. (15)} \,, \\ &= \frac{1}{2}\Delta\theta^\top \bar{H}\Delta\theta \quad \text{see Eq. (16b)} \,. \qquad\qquad (14b) \end{aligned}$$

In Eq. (14a), since the expectation is taken such that, $\mathbf{x} \sim \mathcal{D}, \mathbf{y} \sim p_\theta(\mathbf{y}|\mathbf{x})$, the first order partial derivatives cancel out, *i.e.*,

$$\begin{aligned} \mathbb{E}_{\mathbf{z}}\left[\frac{\partial \log p_\theta(\mathbf{z})}{\partial \theta}\right] &= \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\left[\sum_{\mathbf{y}} p_\theta(\mathbf{y}|\mathbf{x})\frac{\partial \log p_\theta(\mathbf{y}|\mathbf{x})}{\partial \theta}\right] \,, \qquad (15)\\ &= \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\left[\sum_{\mathbf{y}} p_\theta(\mathbf{y}|\mathbf{x})\frac{1}{p_\theta(\mathbf{y}|\mathbf{x})}\frac{\partial p_\theta(\mathbf{y}|\mathbf{x})}{\partial \theta}\right] \,, \\ &= \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\left[\frac{\partial}{\partial \theta}\sum_{\mathbf{y}} p_\theta(\mathbf{y}|\mathbf{x})\right] \,, \\ &= \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}[0] = 0 \,. \end{aligned}$$

Note that this holds for the continuous case as well, where assuming sufficient smoothness and the fact that limits of integration are constants (0 to 1), the Leibniz's rule would allow us to interchange the differentiation and integration operators.

Additionally, in Eq. (14b), the expected value of negative of the Hessian can be shown to be equal to the true Fisher matrix ($\tilde{F}$) by using Information Matrix Equality.

$$\mathbb{E}_{\mathbf{z}}\left[-\frac{\partial^2 \log p_\theta(\mathbf{z})}{\partial \theta^2}\right] = -\,\mathbb{E}_{\mathbf{z}}\left[\frac{1}{p_\theta(\mathbf{z})}\frac{\partial^2 p_\theta(\mathbf{z})}{\partial \theta^2}\right] + \mathbb{E}_{\mathbf{z}}\left[\left(\frac{\partial \log p_\theta(\mathbf{z})}{\partial \theta}\right)\left(\frac{\partial \log p_\theta(\mathbf{z})}{\partial \theta}\right)^\top\right], \tag{16a}$$

$$= -\,\mathbb{E}_{\mathbf{z}}\left[\frac{1}{p_\theta(\mathbf{z})}\frac{\partial^2 p_\theta(\mathbf{z})}{\partial \theta^2}\right] + \tilde{F}_\theta\,. \tag{16b}$$

- By the definition of KL-divergence, the expectation in the above equation is taken such that, $\mathbf{x} \sim \mathcal{D}, \mathbf{y} \sim p_\theta(\mathbf{y}|\mathbf{x})$. This cancels out the first term by following a similar argument as in Eq. (15). Hence, in this case, the expected value of negative of the Hessian equals true Fisher matrix ($\tilde{F}$).
- However, if in Eq. (16b), the expectation is taken such that, $(\mathbf{x}, \mathbf{y}) \sim \mathcal{D}$, the first term does not go to zero, and $\tilde{F}_\theta$ becomes the *empirical Fisher matrix* (F_θ).
- Additionally, at the optimum, since the model distribution approaches the true data distribution, hence even sampling from dataset *i.e.*, $(\mathbf{x}, \mathbf{y}) \sim \mathcal{D}$ will make the first term to approach zero, and $\bar{H} \approx F_\theta$.

With the approximation that $\bar{H} \approx \tilde{F}_\theta \approx F_\theta$, the proof is complete.

Note that, as we will argue in Section A.2 the true Fisher matrix is expensive to compute as it requires multiple backward passes, hence, instead, as widely used in literature [1,19], we employ empirical Fisher to approximate the KL-divergence.

A.2 Empirical vs True Fisher

Loss gradient Let q be any reference distribution and p (parametrized by θ) be the model distribution obtained after applying softmax on the class scores (s). The cross-entropy loss between q and p can be written as: $\ell(\theta) = -\sum_j q_j \log p_j$. The gradients of the loss with respect to the class scores are:

$$\frac{\partial \ell(\theta)}{\partial s_j} = p_j - q_j\,. \tag{17}$$

By chain rule, the loss gradients w.r.t. the model parameters are $\frac{\partial \ell(\theta)}{\partial \theta} = \frac{\partial \ell(\theta)}{\partial \mathbf{s}}\frac{\partial \mathbf{s}}{\partial \theta}$.

Empirical Fisher In case of an empirical Fisher, the expectation is taken such that $(\mathbf{x}, \mathbf{y}) \sim \mathcal{D}$. Since every input $\mathbf{x}$ has only one ground truth label, this makes q a Dirac delta distribution . Then, Eq. (17) becomes:

$$\frac{\partial \ell(\theta)}{\partial s_j} = \begin{cases} p_j - 1, & \text{if '}j\text{' is the ground truth label}\,, \\ p_j, & \text{otherwise}\,. \end{cases}$$

Since *at any optimum* the loss-gradient approaches to zero, thus, Fisher being the expected loss-gradient covariance matrix would also approach to a zero matrix.

True Fisher In case of true Fisher, given $\mathbf{x}$, the expectation is taken such that $\mathbf{y}$ is sampled from the model distributions $p_\theta(\mathbf{y}|\mathbf{x})$. In case of multi-class classification, at a given θ, the model 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. At a good optimum, the model distribution $p_\theta(\mathbf{y}|\mathbf{x})$ becomes peaky around the ground truth label, implying $p_{\theta,t} \gg p_{\theta,j}, \forall j \neq t$ where t is the ground-truth label. Thus, given input $\mathbf{x}$, the model distribution $p_\theta(\mathbf{y}|\mathbf{x})$ approach the ground-truth output distribution. This makes the true and empirical Fisher behave in a very similar manner. Note, in order to compute the expectation over the model distribution, the true Fisher requires multiple backward passes. This makes it prohibitively expensive to compute and the standard solution is to resort to the empirical Fisher instead [1,19].

B Further Experiments

In Tab. 2 we compare proposed RWalk with recently published Gradient Episodic Memory (GEM) [16]. GEM explicitly makes uses of episodic memory (samples from previous classes). Additionally, GEM assumes an integer task-id at test time that makes the evaluation setting *multi-head* as discussed in Sec. 2.1 of the main paper. For CIFAR-100, GEM splits the dataset in 20 tasks, with each task containing 5 consecutive classes. We use the same set-up for our proposed approach (RWalk) and show the results in Tab. 2. We observe that, even if we use a simpler CNN, RWalk performs better than GEM by a margin of **16.3%** *even without using any samples from the previous task*. This, again, substantiates the claim we made in the main paper (Sec. 2.1) that multi-head setting is too easy a setting for practical scenarios, and for a GEM-like set-up it completely obviates the need for samples/ memory from previous tasks in regularized models such as EWC [7], PI [25], and RWalk (Ours).

Additionally, we note that the performance of our method decreases slightly if we use *smaller memory*. This is expected, since when we use samples from previous classes, we update the last layer weights of the corresponding classes as well. As we use a small subset from previous classes, this weight update is sub-optimal, resulting in a slightly decreased performance. Note, however, that even if we use samples, we still observe better average accuracy (78.5%) than GEM (63.3%).

C Insensitivity to Lambda (λ)

In Tab. 3 we analyse the sensitivity of different methods to the regularization hyper-parameter (λ). As argued in Sec. 4.1 of the main paper, and can be seen in the table, RWalk remains insensitive to λ compared to EWC [7] and PI [25]. For example, as we vary λ by a factor of 1×10^5 on MNIST, the *forgetting* and *intransigence* measures

Table 2: *Comparison with GEM [16] on CIFAR-100 dataset. GEM uses an integer task-descriptor at test time, that makes the evaluation setting* ***multi-head****. Additionally, GEM uses ResNet18 [3] whereas we use a smaller CNN as explained in Sec. D. Following GEM setup, we increase the number of tasks from 10 to 20, meaning, each task now contains images from 5 consecutive classes;* $\cup_k \mathbf{y}^k = \{\{0-4\},\{4-9\},\ldots,\{95-99\}\}$. *Additionally, note that, the total number of samples are from all the tasks combined.*

Methods	Total Number of Samples	A_k (%)
GEM [16]	2560	63.3
RWalk (Ours)	0	**79.6**
	2500	**78.5**

Table 3: *Comparison of different methods on MNIST and CIFAR-100 as the regularization strength* (λ) *is varied. With Forgetting and Intransigence we also provide - in brackets '()' - the change* (Δ) *in the corresponding measures, where the first row in each method is taken as the reference. As discussed in Sec. 4.1 in the main paper, our method (****RWalk****) is less sensitive to* λ *compared to EWC [7] and PI [25], making it more appealing for an incremental learning setting.*

Methods	MNIST				CIFAR			
	λ	$A_5(\%)$	$F_5(\Delta)$	$I_5(\Delta)$	λ	$A_{10}(\%)$	$F_{10}(\Delta)$	$I_{10}(\Delta)$
EWC [7]	75	80.3	0.19 (0)	0.1 (0)	3.0	28.9	0.38 (0)	-0.17 (0)
	75×10^3	79.2	0.15 (-0.04)	0.21 (0.11)	300	34.1	0.28 (-0.1)	-0.07 (0.1)
	75×10^5	79.1	0.13 (-0.06)	0.24 (0.14)	3×10^5	33.7	0.27 (-0.11)	-0.03 (0.14)
PI [25]	0.1	79.3	0.23 (0)	0.05 (0)	0.1	34.7	0.27 (0)	-0.07 (0)
	100	80.3	0.15 (-0.08)	0.22 (0.17)	10	34.3	0.26 (-0.01)	-0.04 (0.03)
	10000	78.5	0.16 (-0.07)	0.18 (0.13)	1×10^4	33.7	0.27 (0)	-0.06 (0.01)
RWalk (Ours)	0.1	82.6	0.16 (0)	0.12 (0)	0.1	34.5	0.28 (0)	-0.06 (0)
	100	81.6	0.16 (0)	0.14 (0.02)	10	33.2	0.28 (0)	-0.06 (0)
	10000	81.6	0.16 (0)	0.12 (0)	1×10^4	34.2	0.28 (0)	-0.05 (0.01)

changed by -0.06 and 0.14 on EWC [7], and -0.07 and 0.13 on PI [25], respectively. On the other hand, the change in RWalk, as can be seen in the Tab. 3, is 0 for both the measures. On CIFAR-100 a similar trend is observed in Tab. 3.

D CIFAR Architecture and Task-Level Analysis

In Tab. 4 we report the detailed architecture of the convolutional network used in the incremental CIFAR-100 experiments (Sec. 6). Note that, in comparison to [25], we use only one fully-connected layer (denoted as 'FC' in the table). For each task k, the weights in the last layer of the network can be dynamically added. Additionally, in Fig 6, we present a similar task-level analysis on CIFAR-100 as done for MNIST (Fig. 2 in the main paper). Note that, for all the experiments 'α' in Eq. (10) is set to 0.9 and 'Δt' in Eq. (8) is 10 and 50 for MNIST and CIFAR, respectively.

Table 4: *CNN architecture for incremental CIFAR-100 used for Vanilla, EWC, PI, iCaRl, RWalk in the main paper. Here, 'n' denotes the number of classes in each task.*

Operation	**Kernel**	**Stride**	**Filters**	**Dropout**	**Nonlin.**
3x32x32 input					
Conv	3×3	1×1	32		ReLU
Conv	3×3	1×1	32		ReLU
MaxPool		2×2		0.5	
Conv	3×3	1×1	64		ReLU
Conv	3×3	1×1	64		ReLU
MaxPool		2×2		0.5	
Task 1: FC			n		
$\cdots$: FC			n		
Task k: FC			n		

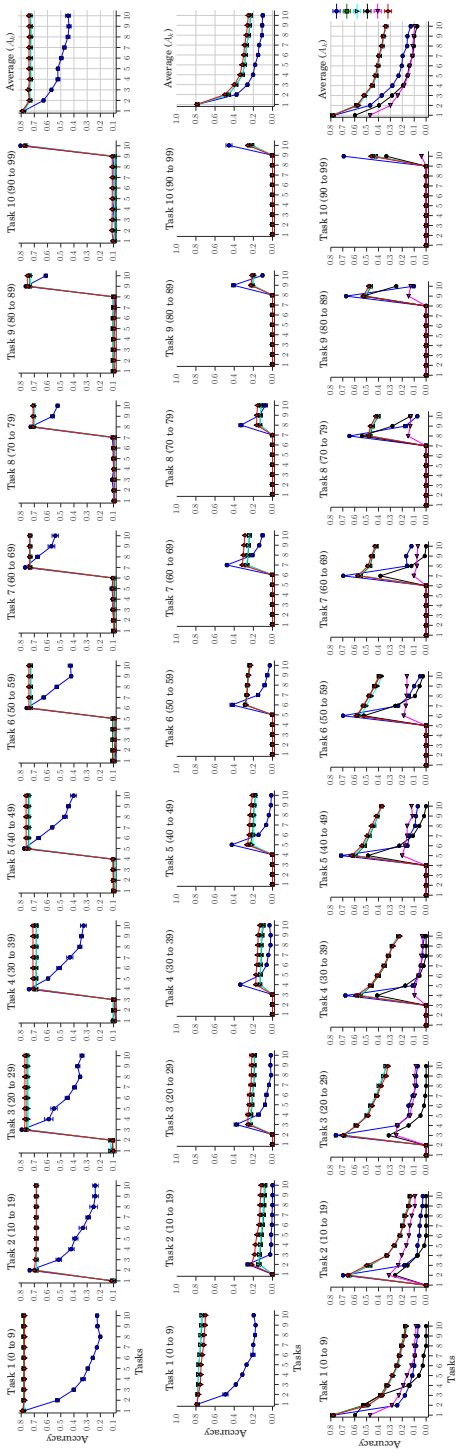


Fig. 6: *Accuracy measure in incremental CIFAR-100 with multi-head evaluation (**top**), and the single-head evaluation without (**middle**) and with samples (**bottom**). The first ten columns show how the performance of different tasks vary as the model is trained for new tasks, e.g., the first plot depicts the variation in performance on Task 1 when the network is sequentially trained for the ten tasks in an incremental manner. The last column shows the average accuracy measure (A_k, by varying k) as defined in Eq. (3). Mean of features (MoF) sampling is used. (best viewed in color)*

References

1. S.-I. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 1998. 2, 4, 16, 17, 18
2. R. Grosse and J. Martens. A kronecker-factored approximate fisher matrix for convolution layers. In *ICML*, 2016. 4
3. K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 19
4. R. Hecht-Nielsen et al. Theory of the backpropagation neural network. *Neural Networks*, 1(Supplement-1):445–448, 1988. 9
5. G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. In *NIPS*, 2014. 1
6. D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 10
7. J. Kirkpatrick, R. Pascanu, N. C. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences of the United States of America (PNAS)*, 2016. 1, 2, 3, 4, 6, 8, 9, 10, 11, 12, 14, 18, 19
8. A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *https://www.cs.toronto.edu/ kriz/cifar.html*, 2009. 2
9. S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 1951. 3
10. N. Le Roux, M. Pierre-Antoine, and Y. Bengio. Topmoumoute online natural gradient algorithm. In *NIPS*, 2007. 4
11. Y. LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998. 2
12. J. Lee, J. Yun, S. Hwang, and E. Yang. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*, 2017. 9
13. J. M. Lee. *Riemannian manifolds: an introduction to curvature*, volume 176. Springer Science & Business Media, 2006. 4
14. S.-W. Lee, J.-H. Kim, J.-W. Ha, and B.-T. Zhang. Overcoming catastrophic forgetting by incremental moment matching. In *NIPS*, 2017. 3, 9
15. Z. Li and D. Hoiem. Learning without forgetting. In *European Conference on Computer Vision*, pages 614–629, 2016. 9
16. D. Lopez-Paz and M. Ranzato. Gradient episodic memory for continuum learning. In *NIPS*, 2017. 5, 6, 9, 11, 15, 18, 19
17. J. Martens and R. Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *ICML*, 2015. 4, 8
18. C. V. Nguyen, Y. Li, T. D. Bui, and R. E. Turner. Variational continual learning. *ICLR*, 2018. 9
19. R. Pascanu and Y. Bengio. Revisiting natural gradient for deep networks. In *ICLR*, 2014. 2, 4, 16, 17, 18
20. S.-A. Rebuffi, H. Bilen, and A. Vedaldi. Learning multiple visual domains with residual adapters. In *NIPS*, 2017. 9
21. S.-V. Rebuffi, A. Kolesnikov, and C. H. Lampert. iCaRL: Incremental classifier and representation learning. In *CVPR*, 2017. 1, 3, 8, 9, 10, 11, 12, 14, 15
22. A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016. 9

23. H. Shin, J. K. Lee, J. Kim, and J. Kim. Continual learning with deep generative replay. In *NIPS*, 2017. 9
24. A. V. Terekhov, G. Montone, and J. K. ORegan. Knowledge transfer in deep block-modular neural networks. In *Conference on Biomimetic and Biohybrid Systems*, pages 268–279, 2015. 9
25. F. Zenke, B. Poole, and S. Ganguli. Continual learning through synaptic intelligence. In *ICML*, 2017. 1, 2, 3, 6, 7, 8, 9, 10, 11, 12, 14, 18, 19