Stochastic linear dynamics in parameters to deal with Neural Networks plasticity loss

Alexandre Galashov
Google DeepMind, Gatsby Unit, UCL
agalashov@google.com

Michalis Titsias
Google DeepMind
mtitsias@google.com

Razvan Pascanu
Google DeepMind
razp@google.com

Yee Whye Teh
Google DeepMind
ywteh@google.com

Maneesh Sahani
Gatsby Unit, UCL
maneesh@gatsby.ucl.ac.uk

Abstract

Plasticity loss has become an active topic of interest in the continual learning community. Over time, when faced with non-stationary data, standard gradient descent loses its ability to learn. It comes in two forms, the inability of the network to generalize and its inability to fit the training data. Several causes have been proposed including ill-conditioning or the saturation of activation functions. In this work we focus on the inability of neural networks to optimize due to saturating activations, which particularly affects online reinforcement learning settings, where the learning process itself creates a non-stationary setting even if the environment is kept fixed. Recent works have proposed to answer this problem by relying on dynamically resetting units that seem inactive, allowing them to be tuned further. We explore an alternative approach to this based on stochastic linear dynamics in parameters which allows to model non-stationarity and provides a mechanism to adaptively and stochastically drift the parameters towards the prior, implementing a mechanism of soft parameters reset.

1 Plasticity loss in Neural Networks

Learning algorithms for Neural Networks (NNs) such as Stochastic Gradient Descent (SGD) rely on the iid training data assumption. This assumption, however, is violated in many scenarios such as continual learning [13], reinforcement learning, non-stationary contextual bandits, supervised learning with distribution shift, to name a few. When trained on non-stationary data, Neural Networks tend to lose the ability to learn and adapt to new data, a phenomenon known as *loss of plasticity* [2, 8].

Addressing the loss of plasticity is an active area of research and many methods already have been proposed to tackle it. We note that the phenomenon can take different forms. For example, [2, 4] describe how in non-stationary settings, neural networks lose plasticity by becoming unable to generalize. In contrast [8] highlights a slightly different phenomenon, which is more commonly studied in follow up works, where the network is unable to minimize the training objective, the causes of which could range from ill-conditioning to the prevalence of dead units [17]. To address this, regularization based methods provide regularization techniques which mitigate plasticity loss, some works include [2, 15, 17]. Architectural based methods are based on changing the architecture to make it less pron to the plasticity loss such as [18, 3, 20, 1]. Methods based on changing the optimization algorithm have also been proposed which essentially show that the optimizer and its hyperparameters could have an impact on the plasticity loss, see [18]. Finally, a large body of work focuses on the idea that some parameters of the Neural Network (NN) needs to be occasionally reset when dealing with non-stationarity. A notable algorithm is Continual Backpropagation [8], a

Workshop on Distribution Shifts, 37th Conference on Neural Information Processing Systems (NeurIPS 2023).

modification of the original backpropagation [19] which tracks the utility measure for each neuron and resets the ones with utility below a certain threshold. Shrink and Perturb [2] is a method which multiplies each parameter by a scalar $\lambda \in [0, 1]$ and adds Gaussian noise with some variance $\sigma \geq 0$. This allows the network to maintain the plasticity and deal with [2] problem. Recently, a method called *Regenerative Regularization* [16] was proposed which regularizes the parameters to their initial values and could be interpreted as a form of a parameters reset.

Most of the approaches, however, do not explicitly model the non-stationarity in the data. Recently, a few methods were proposed [6, 21] which explicitly take into account the non-stationarity by *a priori* assuming that model parameters could drift. These methods are designed for non-stationary online learning and are different implementations of Bayesian filtering principle. However, in [21] the authors rely on strong assumption that the observational model is linear in the non-stationary parameters, and in [6], the authors use Extended Kalman Filter (EKF) which scales quadratically with the number of classes making it impractical for many large-scale classification settings.

In this work, we take inspiration from [6, 21] to explicitly model the drift in parameters and propose a simple yet effective strategy which stochastically moves each Neural Network parameter back to the prior (initialization), controlled by resetting parameters. These resetting parameters are learned online using the same objective as in [21]. The overall method could be interpreted as adaptive Shrink&Perturb [2] and a mechanism for adaptive parameter resetting, where the amount of reset is captured from the non-stationarity in the data.

2 Our approach

Consider a stream of labelled data $\mathcal{S}_T = (\mathcal{D}_t)_{t=1}^T$, where $\mathcal{D}_t = (x_t^n, y_t^n)_{n=1}^{N_t}$ and N_t is a size of each chunk of data. Without loss of generality and to simplify the presentation, let us assume that $N_t = 1$. The objective is to fit a model $y = f(x|\theta)$ with parameters $\theta \in \mathcal{R}^D$ in online fashion, assuming that the data distribution **may** change over time. Most of the works on plasticity loss operate in this setting. A stationary case when the data is i.i.d. is a special case of this framework.

A simple approach to this problem is *Online Stochastic Gradient Descent* (Online-SGD) described as follows. Let θ_t be the parameter of the model after being trained on the data $\mathcal{S}_t$ up to the time t. Let $p(y_{t+1}|x_{t+1}, \theta)$ be the likelihood of y_{t+1} given the parameters θ and x_{t+1}. At time $t + 1$, when we observe new data (x_{t+1}, y_{t+1}), we update parameters θ_t as:

$$\theta_{t+1} = \theta_t - \alpha \nabla_{\theta_t} \left[-\log p(y_{t+1}, f(x_{t+1}, \theta_t)) \right] \tag{1}$$

In practice, instead of using SGD optimizer with the rule from eqn. 1, one might use a different optimizer such as Adam [12]. Neural networks trained in this way may exhibit loss of plasticity [2, 17].

We propose an approach which explicitly takes into account the non-stationarity in the data. Let $p(\theta_0)$ be the prior (initialization) distribution of the parameters θ, which in practice is often a Gaussian distribution $p(\theta_0) = \mathcal{N}(0, \sigma_\theta^2 \circ I_D)$ with some variance **vector** $\sigma_\theta^2 = (\sigma_{\theta^0}^2, \ldots, \sigma_{\theta^D}^2)$, where $\circ$ is the element-wise multiplication. In Neural Network training, a Gaussian distribution is typically used for the weights whereas a zero initialisation is used for the biases. Throughout this presentation to keep the notation consistent, we assume that the biases are initialised from a Gaussian with variance 0.

Our objective is to find new parameters θ_{t+1} such that it fits well the new data by maximizing the likelihood $p(y_{t+1}|x_{t+1}, \theta_{t+1})$ in one gradient update. To account for non-stationarity, following [21], at time $t + 1$, **before** observing new data, we assume the model parameters drift as:

$$\hat{\theta}_{t+1} = \gamma \circ \theta_t + \left(\sqrt{1 - \gamma^2} \right) \circ \theta_0, \theta_0 \sim \mathcal{N}(0, \sigma_\theta^2 \circ I_D) \tag{2}$$

Here, $p(\hat{\theta}_{t+1}|\theta_t, \gamma) = \mathcal{N}(\hat{\theta}_{t+1}; \gamma \circ \theta_t, (1 - \gamma^2) \circ \sigma_\theta^2)$ represents the prior probability about a range of possible values of the true parameter θ_{t+1} could take at next time $t + 1$ before observing the data. Here, $\gamma = (\gamma^1, \ldots, \gamma^D)$ is a per-parameter vector such that $\gamma^i \in [0, 1]$. Each γ^i can be interpreted as *forgetting factor* (see [21]) or *resetting parameter* which controls the amount of drift to the prior. If $\gamma^i = 1$ there is no drift and $\hat{\theta}_{t+1}^i = \theta_t^i$. If $\gamma^i = 0$ then the corresponding parameter is re-initialized from the prior $\hat{\theta}_{t+1}^i = \theta_0^i \sim \mathcal{N}(\theta_0^i; 0, \sigma_{\theta^i}^2)$. Using a value of $\gamma^i \in [0, 1)$ could be viewed as a mechanism for a soft parameter reset.

Let $\tilde{\theta}_{t+1} = \gamma \circ \theta_t$ be the mean of $p(\hat{\theta}_{t+1}|\theta_t, \gamma)$. The likelihood of new data under $\tilde{\theta}_{t+1}$ is given as:

$$p(y_{t+1}|x_{t+1}, \tilde{\theta}_{t+1}) = \int p(y_{t+1}|x_{t+1}, \theta_{t+1}) p(\theta_{t+1}|\tilde{\theta}_{t+1}) d\theta_{t+1},$$

which can be written as following using reparameterisation trick:

$$p(y_{t+1}|x_{t+1}, \tilde{\theta}_{t+1}) = \int p(y_{t+1}|x_{t+1}, \tilde{\theta}_{t+1} + \sqrt{1-\gamma^2} \circ \sigma_\theta \epsilon) \mathcal{N}(\epsilon|0, I_D) d\epsilon \tag{3}$$

This gives us an objective function for learning new value of parameter θ_{t+1} fitting new data as:

$$\theta_{t+1} = \tilde{\theta}_{t+1} - \alpha \nabla_{\tilde{\theta}_{t+1}} \left(-\log \int p(y_{t+1}|x_{t+1}, \tilde{\theta}_{t+1} + \sqrt{1-\gamma^2} \circ \sigma_\theta \epsilon) \mathcal{N}(\epsilon|0, I_D) d\epsilon \right) \tag{4}$$

The eqn. 4 could be interpreted as follows. At time $t+1$, before observing new data, we assume that the parameter could drift accoring to the eqn. 2 and we take the mean of the underlying distribution as the initial guess. After that we incorporate the uncertainty induced by the eqn. 2 about the parameter drift, in the Online SGD style parameters update. When $\gamma^i = 1, \forall i$, this is equivalent to Online SGD.

The above procedure is somewhat similar to Shrink&Perturb method [2], which applies the following transformation to the parameters before the gradient updates:

$$\hat{\theta}_{t+1} = \lambda \theta_t + \sigma \epsilon, \epsilon \sim \mathcal{N}(0, I), \tag{5}$$

where $\lambda \in (0, 1]$ is the shrinking parameter and $\sigma > 0$ the perturbation parameter. After the transformation from eqn. 18, the method proceeds via Online SGD applied to the perturbed parameter $\hat{\theta}_{t+1}$. The authors of [2] argued that this allows parameters to escape the warmstarting problem and gain plasticity. In our method, if we assume $\lambda = \gamma$ and $\sigma = \sqrt{1-\gamma^2}\sigma_\theta$, where γ and σ_θ are scalars, we recover a method very similar to Shrink&Perturb with an exception that the random perturbation is only used in the gradient update in eqn. 4 and not in the initial guess $\tilde{\theta}_{t+1}$. The interpretation, however, is a quite different from Shrink&Perturb. In our method, γ implements a trade-off between previous parameter value and the prior, acting as a way of doing soft parameter reset. In Appendix 4.4 we provide an empirical comparison of our method with fixed γ and Shrink&Perturb.

There are a few alternatives to the objective in eqn. 3. One alternative is to first apply perturbation $\tilde{\theta}_{t+1} = \gamma \circ \theta_t + \sqrt{1-\gamma^2} \circ \sigma_\theta \epsilon$ and then optimize noise-free Online-SGD objective $\log p(y_{t+1}|x_{t+1}, \tilde{\theta}_{t+1})$. This objective is closer to the Shrink&Perturb method and could be interpreted as one iteration of SGLD algorithm [22] with no additional noise. Second alternative is to initialize $\tilde{\theta}_{t+1} = \gamma \circ \theta_t$ and then similarly, optimize noise-free Online-SGD objective $\log p(y_{t+1}|x_{t+1}, \tilde{\theta}_{t+1})$. This would correspond to finding MAP on the data (x_{t+1}, y_{t+1}) assuming a prior $\mathcal{N}(\tilde{\theta}_{t+1}; \gamma \circ \theta_t; \lambda^2)$ on parameters $\hat{\theta}_{t+1}$ for some $\lambda > 0$.

Our approach could be viewed as the noisy version of this MAP procedure. Adding noise to the gradient updates is a technique used in *continuation methods* [7, 10, 9] or in *graduated optimization* [11]. These approaches are designed for non-convex optimization and inject noise at every iteration of gradient update. Empirically, we found that our variant works the best. Comparison to other variants is given in Appendix 4.3.

2.1 Learning dynamics parameter γ online

Following [21], at time $t+1$ before using perturbations from the eqn. 2, we update γ as follows:

$$\gamma \leftarrow \gamma - \beta \nabla_\gamma \left(-\log \int p(y_{t+1}|x_{t+1}, \gamma \circ \theta_t + \sqrt{1-\gamma^2} \circ \sigma_\theta \epsilon) \mathcal{N}(\epsilon|0, I_D) d\epsilon \right) \tag{6}$$

The interpretation of the eqn. 6 is as following. Given current parameter θ_t and new data, we find new γ such that it minimizes the loss on new data, assuming the drift from eqn. 2. This is corresponds to an online model selection procedure which adapts γ to the non-stationarity of the data. In practice, as in [21], we parameterise $\gamma = \exp\{-0.5\delta\}$ for some $\delta \geq 0$ for numerical stability and we optimize the objective from eqn. 6 wrt δ.

Algorithm 1 Learned weight perturbation

Input: Data-stream $\mathcal{S}_T = \{(x_t, y_t)\})_{t=1}^T$, hyperparameters $\delta_0, \sigma_\theta^2$, learning rates α, β
$acc(y, \hat{y})$ - accuracy function between ground truth y and prediction $\hat{y}$
$p(\theta_0) = \mathcal{N}(0, \sigma_\theta^2 I_D)(\theta_0)$ - prior distribution over NN parameters
N - number of NN parameters, T - number of iterations, K - number of Monte-Carlo samples
$E = 0$ - cumulative error,
Initialise: NN parameters $\theta_0 \sim \mathcal{N}(0, \sigma_\theta^2 I_D)$, model dynamics parameters $\delta^i = \delta_0, \forall i$
for step $t = 0, 1, 2, \ldots, T$ **do**
  (x_{t+1}, y_{t+1}), predict $\hat{y}_{t+1} = f(x_{t+1}|\theta_t)$, compute the model error $e_t = 1 - acc(y_{t+1}, \hat{y}_{t+1})$
  Update cumulative error $E = E + e_t$
  Sample NN parameters from the prior $\theta_0 \sim p_0(\theta_0|\sigma_\theta^2)$
  Let $\gamma = \exp\{-0.5\delta\}$ and use it to update δ with eqn. 8
  Use this new γ to find new parameters θ_{t+1} using eqn. 7
end for

2.2 Monte-Carlo approximation

In practice the integrals from eqn. 4 and eqn. 6 cannot be computed in the closed form if the likelihood $p(y_{t+1}|x_{t+1}, \theta)$ does not have a simple form (i.e., a linear function of θ as in [21]). Thus, we employ a Monte-Carlo (MC) approximation. This would provide a modified update rule from eqn. 4:

$$\theta_{t+1} = \tilde{\theta}_{t+1} - \alpha \nabla_{\tilde{\theta}_{t+1}} \left(-\log \frac{1}{K} \sum_{k=1}^{K} p(y_{t+1}|x_{t+1}, \tilde{\theta}_{t+1} + \sqrt{1-\gamma^2} \circ \sigma_\theta \epsilon_k) \right), \tag{7}$$

where $\epsilon_k \sim \mathcal{N}(0, I_D)$. Similarly, the MC approximation for eqn. 6 is given as:

$$\gamma \leftarrow \gamma - \beta \nabla_\gamma \left(-\log \frac{1}{K} \sum_{k=1}^{K} p(y_{t+1}|x_{t+1}, \gamma \circ \theta_t + \sqrt{1-\gamma^2} \circ \sigma_\theta \epsilon_k) \right) \tag{8}$$

The full algorithm is described in Algorithm 1. In practice we found that $K = 1$ works very well. The appropriate ablation is provided in Appendix 4.1.

3 Experiments

In this section, we provide empirical evidence of the effectiveness of this approach to deal with plasticity loss. Similar to [16], we construct a sequence of tasks based on CIFAR-10 [14], where each task corresponds to a subset of data where for each example the labels are randomly permuted. For more details, see [18] and [16]. The task identity or the task change points are unknown to the method. We consider two settings, **easy** and **hard**, where in the **easy** setting we subsample 1200 data points from CIFAR-10 [14], and construct 50 tasks, each having a duration of 1875 steps with batch size of 16, which corresponds to 400 epochs. This is equivalent to "*Random-label CIFAR-10*" from [16]. In the **hard** variant, we consider the full set of CIFAR-10 [14], and construct 10 tasks, each having a duration of 7500 steps with batch size 128, which corresponds to 19.2 epochs. The main difference between two settings consists in the amount of data a method sees within each of the task and in the amount of epochs per task. Arguably, the **hard** variant of the task would highlight the data-efficiency aspect of addressing plasticity loss for each of the methods. On top of that, we also report results on *Random Label MNIST*, which is the same task as the **easy** variant of *Random Label CIFAR-10*. For more details, see [18] and [16]. Moreover, we added more details on the benchmarks and model architectures in Appendix 4.

As baselines, we consider *Online SGD*, *Shrink&Perturb* [2], *L2 init* [16] and *Hard reset* at task boundaries. *L2 init* adds L2-regularisation towards initial parameters $||\theta - \theta_0||^2$ to the loss and as argued in [16], acts as a way of resetting parameters. *Hard reset* re-initialises the parameters at task boundaries, which is a strong baseline but requires the knowledge of these boundaries. As optimizer for all methods we use Adam [12]. For our method, we use $K = 1$ number of Monte-Carlo samples. We select hyperparameters which minimize the cumulative error at the end of the training. For more details, see Appendix 4.

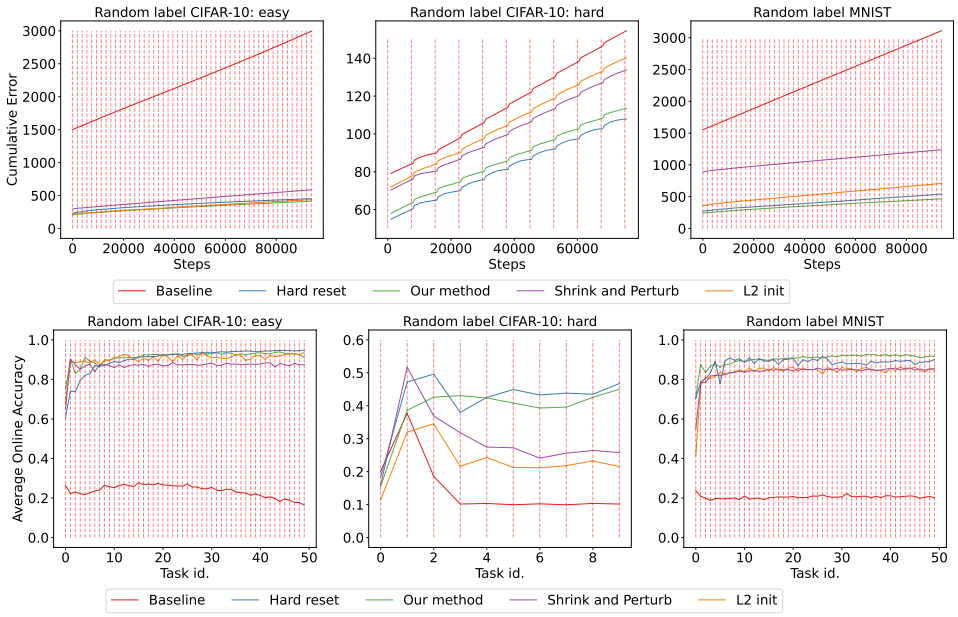

Figure 1: Learning on random label CIFAR-10 and MNIST. The red lines correspond to the task changes. The plot on the top shows cumulative error metric with X-axis corresponding to training steps. The plot on the bottom shows the average per-task online accuracy and the X-axis indicates the task id. Columns correspond to the task variant.

For evaluation, we use two metrics. First, we use cumulative (across training) error computed for each training step. Second, we use average per-task online accuracy. These metrics highlight different aspects of the problem. Cumulative error highlights the data efficiency of the method, whereas the average per-task online accuracy highlights the final performance per task of each of the method. See Appendix 4 for more details about each of the metrics.

The results are given in Figure 1. We observe that our method is able to achieve similar final performance to the *Hard Reset* but also is more data-efficient. On **easy** task variant of *Random label CIFAR-10*, Figure 1, left, we observe that the method performs very closely to *L2 init*, and both methods are very close to hard reset and slightly outperform it. On Figure 1, center, however, for the **hard** task variant of *Random label CIFAR-10*, we see that our method is very close to the hard reset, whereas *L2 init* is much worse. Finally, we also observe that our seems to even slightly outperform the *Hard Reset* on *Random label MNIST*, Figure 1, right, whereas *L2 init* method has slightly worse performance. This last result is interesting, because the only reason one could outperform *Hard Reset* on this benchmark would be in implementing a form of knowledge transfer. This highlights the **soft-reset** property of our method. Since, the parameters γ are learned per-parameter, in principle, it could have $\gamma \sim 1$ for some parameters and $\gamma << 1$ for others. This would allow to in one hand, adapt to the non-stationarity, and on the other hand, transfer knowledge. See Appendix 4.6 for visualizations of γ learning dynamics. Overall, the experiments suggest that our method maintains plasticity better than baseline variants. In Appendix 4 we provide more ablations for our method.

4 Conclusion

We have presented a simple method for mitigating the plasticity loss which explicitly takes into account the non-stationarity in the data. The method implements a mechanism of adaptively drifting parameters towards the prior therefore implementing a soft resetting mechanim. Empirically, we found this method performs well on the studied benchmarks compared to the baselines. In the future work, we will validate this method in the general non-stationary learning setting and will study in-depth the connection to other online learning approaches.

References

[1] Zaheer Abbas, Rosie Zhao, Joseph Modayil, Adam White, and Marlos C. Machado. Loss of plasticity in continual deep reinforcement learning, 2023.

[2] Jordan T. Ash and Ryan P. Adams. On warm-starting neural network training, 2020.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[4] Tudor Berariu, Wojciech Czarnecki, Soham De, Jorg Bornschein, Samuel Smith, Razvan Pascanu, and Claudia Clopath. A study on the plasticity of neural networks, 2021.

[5] Zhipeng Cai, Ozan Sener, and Vladlen Koltun. Online continual learning with natural distribution shifts: An empirical study with visual data, 2021.

[6] Peter G. Chang, Gerardo Durán-Martín, Alexander Y Shestopaloff, Matt Jones, and Kevin Murphy. Low-rank extended kalman filtering for online learning of neural networks from streaming data, 2023.

[7] Pratik Chaudhari, Adam Oberman, Stanley Osher, Stefano Soatto, and Guillaume Carlier. Deep relaxation: partial differential equations for optimizing deep neural networks, 2017.

[8] Shibhansh Dohare, Richard S. Sutton, and A. Rupam Mahmood. Continual backprop: Stochastic gradient descent with persistent randomness, 2022.

[9] Caglar Gulcehre, Marcin Moczulski, Misha Denil, and Yoshua Bengio. Noisy activation functions, 2016.

[10] Caglar Gulcehre, Marcin Moczulski, Francesco Visin, and Yoshua Bengio. Mollifying networks, 2016.

[11] Elad Hazan, Kfir Y. Levy, and Shai Shalev-Shwartz. On graduated optimization for stochastic non-convex problems, 2015.

[12] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[13] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, mar 2017.

[14] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[15] Aviral Kumar, Rishabh Agarwal, Dibya Ghosh, and Sergey Levine. Implicit under-parameterization inhibits data-efficient deep reinforcement learning, 2021.

[16] Saurabh Kumar, Henrik Marklund, and Benjamin Van Roy. Maintaining plasticity via regenerative regularization, 2023.

[17] Clare Lyle, Mark Rowland, and Will Dabney. Understanding and preventing capacity loss in reinforcement learning, 2022.

[18] Clare Lyle, Zeyu Zheng, Evgenii Nikishin, Bernardo Avila Pires, Razvan Pascanu, and Will Dabney. Understanding plasticity in neural networks, 2023.

[19] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[20] Wenling Shang, Kihyuk Sohn, Diogo Almeida, and Honglak Lee. Understanding and improving convolutional neural networks via concatenated rectified linear units, 2016.

[21] Michalis K. Titsias, Alexandre Galashov, Amal Rannen-Triki, Razvan Pascanu, Yee Whye Teh, and Jorg Bornschein. Kalman filter for online classification of non-stationary data, 2023.

[22] Max Welling and Yee Whye Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 681–688, Madison, WI, USA, 2011. Omnipress.

Benchmarks and model architectures

We define a task as learning on CIFAR-10 [14] such that the labels are randomly permuted. The task identity nor the task change point are unknown to the method. We consider two settings, **easy** and **hard**. The **easy** setting is exactly the one considered in [16], where we initially subsample CIFAR-10 [14] to 1200 data points and each task corresponds to 30000 timesteps (400 epochs with batch size of 16). The total number of tasks is 50. As a model $f(y|x, \theta)$, we take a simple convolutional neural network with 2 convolutional layer with kernel size 5 for the first layer and kernel size 3 of the second one. Number of channels is equal to 64. The convolutional layers are then followed by 1 fully connected layer with size 256 and one output layer. The **hard** setting corresponds to a full CIFAR-10 [14], where each task involves training on it for 7500 timesteps (19.2 epochs with batch size of 128). The backbone model is 4 layers CNN, with the first layer having kernel size of 5 and subsequent layers have a kernel size of 3. The number of channels is equal to 64. The convolutional layers are then followed by 1 fully connected layer with size 256 and one output layer. On top of that, we consider a random label variant of MNIST task, which we call *Random label MNIST*, similarly to the one considered in [16]. The setting on MNIST task is the same as the **easy** variant of *Random label CIFAR-10* in terms number of data points, number of tasks, number of steps per task, batch size and the architecture. Note that unlike [16], we use CNN for MNIST.

Metrics

When evaluating the performance of the methods, we used two different metrics. The *average online accuracy* metric was introduced in [5] and is defined as:

$$\mathcal{A} = \frac{1}{T}\sum_{t=1}^{T} acc(y_t, \hat{y}_t), \tag{9}$$

where T is the number of steps per task, y_t is the ground truth at time t and $\hat{y}_t$ is the prediction at time t. This metric measures how well a method could adapt to the changes in the distribution. Following the presentation from [16], we use the *per-task average online accuracy*, which is defined as follows for the task T_i:

$$\mathcal{A}_{T_i} = \frac{1}{M}\sum_{j=t_i}^{t_i+M-1} acc(y_t, \hat{y}_t), \tag{10}$$

where t_i is the starting time step of task T_i, M is the number of steps in the task T_i. This metric captures how quickly we are able to learn to do well on the task, which is a measure of its plasticity. If this accuracy goes down over time, we say that there is a plasticity loss, assuming that all tasks are of equal difficulty.

Other metric we use is *training cumulative error*, which at time T is defined as:

$$\mathcal{E}_T = \frac{1}{T}\sum_{t=1}^{T}(1 - acc(y_t, \hat{y}_t)) \tag{11}$$

Higher the training cumulative error is, worse the model trains. Compared to the per-task average online accuracy, this metric also highlights each method's data efficiency, i.e., how fast each method can adapt to a non-stationarity. We report the results in Figure 1. We see that these metrics indeed highlight different aspects of the problem.

Hyperparameters

For every hyperparameter for every method, we ran experiment with 3 different seeds. The best hyperparameter for each method was selected by minimizing the cumulative error at the end of the training.

For all the methods we did a sweep over learning rates α for the θ parameters which included the following values: $1e-3, 5e-4, 1e-4, 5e-5, 1e-5$.

For **L2 init**, we considered the following values for the regularisation parameter $\lambda \in \{10.0, 1.0, 1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10\}$.

For **Shrink&Perturb**, we considered the following values for the shirnkage parameter $\lambda \in \{1.0, 0.99999, 0.9999, 0.999, 0.99, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1\}$. We have considered the following values for the perturbation parameter $\sigma \in \{0.5, 1.0, 0.5, 1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10\}$.

For **Our method**, we had 2 hyperparameters, the variance σ_θ and the learning rate β for updating γ. For the variance we have considered the following values $\sigma_\theta \in \{0.01, 0.1, 0.2, 0.5, 1.0, 10.0\}$. For the learning rate β we have considered the following values $\beta \in \{1e-6, 5e-6, 1e-5, 5e-5, 5e-4, 1e-4\}$. We use $K = 1$ number of Monte-Carlo samples.

Additional ablations

In this section we provide ablations which motivate the design decisions behind our method.

4.1 Number of Monte Carlo samples and stochasticity

In this section, we ablate the number of Monte-Carlo samples we use in eqn. 7 and in eqn. 8. The results are given in Figure 2. We observe that it is beneficial for this method to keep the number of samples K to be low. We believe it is due to the fact that having too many samples K would smooth out the perturbation introduced by the linear dynamics from eqn. 2, and the method would revert closer to Online SGD. The experiment is run on *Random Label CIFAR-10: easy* task.

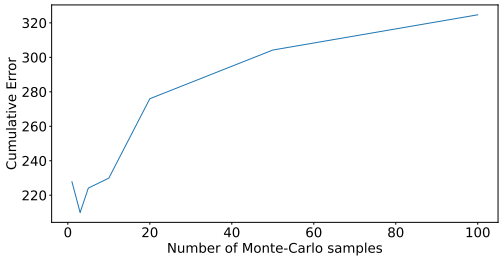

Figure 2: The impact of number of Monte-Carlo samples on the performance. The experiment is run on *Random Label CIFAR-10: easy* task.

On top of that, we check whether it is important to use the same Monte-Carlo samples ϵ_k for eqn. 7 and eqn. 8, or it is beneficial to resample ϵ_k for each of the objective. The Figure 3 shows that it is slightly more beneficial to keep the same noise among two objectives. The experiment is run on *Random Label CIFAR-10: easy* task.

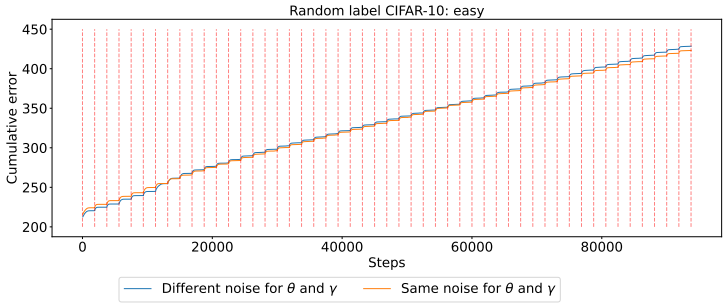

Figure 3: The impact of noise for Monte-Carlo approximations, on the performance. The experiment is run on *Random Label CIFAR-10: easy* task.

4.2 Order of learning γ

In this section we ablate whether the order of whether we learn γ before or after learning θ is important. Figure 4. We see that learning γ before learning θ is more important. We believe that the opposite order would lead to more overfitting.

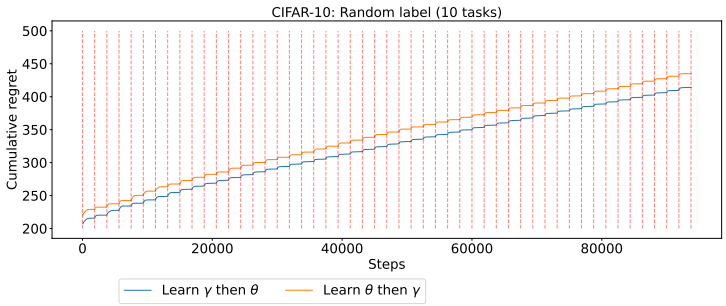

Figure 4: Impact of the order of gamma learning on the performance. The experiment is run on *Random Label CIFAR-10: easy* task.

4.3 Alternatives of our method

As discussed in Section 2, our method could have multiple alternatives. The alternatives only define the learning rule for the parameters θ_{t+1} and modify the objective function in eqn. 4. Below, we list possible alternatives.

Stochastic transition + noise-free SGD alternative operates as following. First, it does the stochastic transition:

$$\tilde{\theta}_{t+1} = \gamma\theta_t + \sqrt{1-\gamma^2}\sigma_\theta\epsilon, \epsilon \sim \mathcal{N}(0, I_D) \tag{12}$$

then it applies one SGD update on the likelihood without the noise:

$$\theta_{t+1} = \tilde{\theta}_{t+1} - \alpha\nabla_{\tilde{\theta}_{t+1}}\left(-\log p(y_{t+1}|x_{t+1}, \tilde{\theta}_{t+1})\right) \tag{13}$$

This variant is essentially a version of Shrink&Perturb with specific parameterisation of $\lambda = \gamma$ and $\sigma = \sqrt{1-\gamma^2}\sigma_\theta$ in the eqn. 18. It is a bit more expressive since we have γ per parameter in our model, and we learn these γ online. The procedure described by eqn. 12 and eqn. 13 could also be interpreted as one step of the SGLD [22] algorithm without additional noise.

Deterministic transition + noise-free SGD alternative does the deterministic transition (as our method):

$$\tilde{\theta}_{t+1} = \gamma\theta_t \tag{14}$$

and does the gradient update noise-free:

$$\theta_{t+1} = \tilde{\theta}_{t+1} - \alpha\nabla_{\tilde{\theta}_{t+1}}\left(-\log p(y_{t+1}|x_{t+1}, \tilde{\theta}_{t+1})\right) \tag{15}$$

This procedure could be interpreted as doing MAP inference on the following distribution:

$$\theta_{t+1} = \arg\max_\theta p(y_{t+1}|x_{t+1}, \theta)\mathcal{N}(\theta; \gamma\theta_t, \lambda), \tag{16}$$

with some $\lambda > 0$ constant depending on α. This comes from the implicit parameterisation of SGD due to the initialization.

Deterministic transition + noisy SGD is our method and we described it in detail in Section 2.

Detereministic transition + noisy SGD + perturbation at the end is a variant of our method where we add an additional perturbation step after the gradient update in eqn. 4:

$$\theta_{t+1} \leftarrow \theta_{t+1} + \sqrt{1-\gamma^2}\sigma_\theta\epsilon, \epsilon \sim \mathcal{N}(0, I_D) \tag{17}$$

The ablation results are provided in Figure 5. First of all, we see that the stochastic transition before noise-free SGD is much worse than variants with deterministic transition. Second, we see that deterministic transition + noisy SGD + pertubation at the end is worse than other variants. Interestingly, we see that determinisitc transition with noisy SGD performs similarly well as the deterministic transition without the noise.

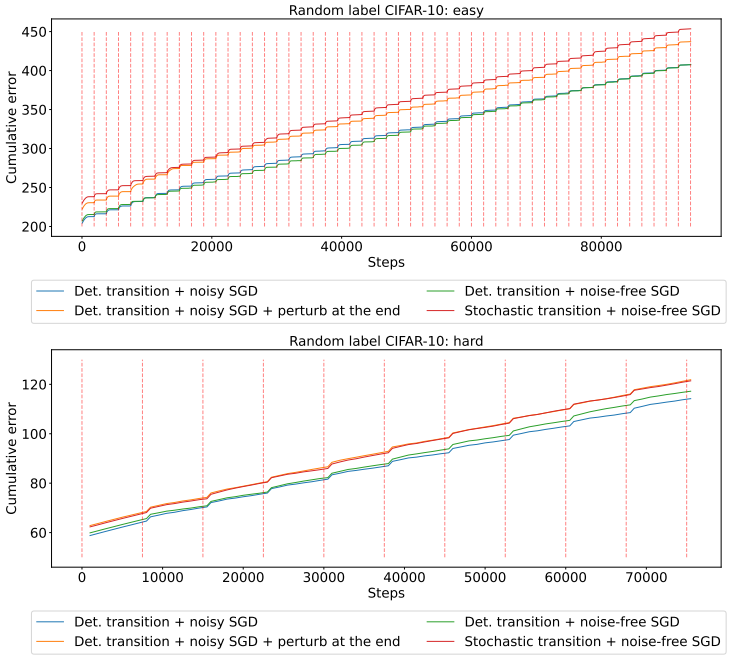

Figure 5: Algorithm variants ablation. The top row shows results on **easy** variant of *Random label CIFAR-10*, whereas the bottom row shows results on **hard** variant of this task.

4.4 Connection to Shrink and Perturb

As discussed in the Section 2, our method could is somewhat similar to Shrink&Perturb [2]. In Shrink&Perturb method, we change the parameters as follows before the gradient updates:

$$\hat{\theta}_{t+1} = \lambda\theta_t + \sigma\epsilon, \epsilon \sim \mathcal{N}(0, I), \tag{18}$$

where $\lambda \in (0, 1]$ is the shrinking parameter and $\sigma > 0$ the perturbation parameter. After the transformation from eqn. 18, the method proceeds via Online SGD applied to the perturbed parameter $\hat{\theta}_{t+1}$. The authors of [2] argued that this allows parameters to escape the warmstarting problem and gain plasticity. In our method, if we assume $\lambda = \gamma$ and $\sigma = \sqrt{1-\gamma^2}\sigma_\theta$, where γ and σ_θ are scalars, we recover a method very similar to Shrink&Perturb with an exception that the random perturbation is only used in the gradient update in eqn. 4 and not in the initial guess $\tilde{\theta}_{t+1}$. Moreover, in our method we learn γ per parameter and therefore the overall method is much more expressive.

Nevertheless, we ran an ablation of our method, where we fix γ to be a scalar over which we sweep. We also do the sweep over the parameters for the Shrink&Perturb. The results are given in Figure . We see that even with a fixed value of γ, our method achieves better performance than Shrink&Perturb.

4.5 Impact of noise in learning γ

In this section we study the impact of noise in eqn. 6 together with eqn. 4. In particular, we study whether it is important to use the noise ϵ in these updates, or we can put $\epsilon = 0$ and use noise-free update. The noise-free updates for θ are given by:

$$\theta_{t+1} = \tilde{\theta}_{t+1} - \alpha\nabla_{\tilde{\theta}_{t+1}}(-\log p(y_{t+1}|x_{t+1}, \tilde{\theta}_{t+1})) \tag{19}$$

The noise-free update in γ is given by:

$$\gamma \leftarrow \gamma - \beta\nabla_\gamma\left(-\log p(y_{t+1}|x_{t+1}, \gamma \circ \theta_t)\right) \tag{20}$$

Compare these updates to eqn. 6 for γ and to eqn. 4 for θ.

The results of this ablation are given in Figure 7. We see clearly that using noise in γ updates is very important, however using noise in θ update is marginally important.

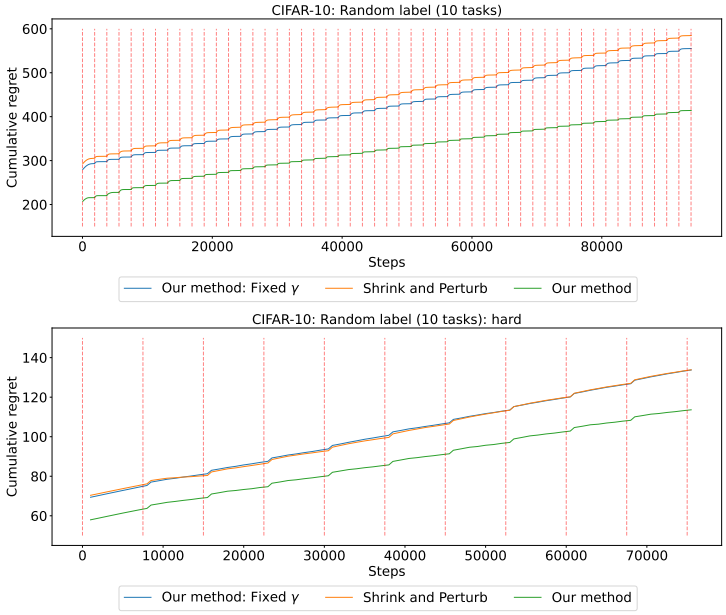

Figure 6: Comparison to Shrink&Perturb.

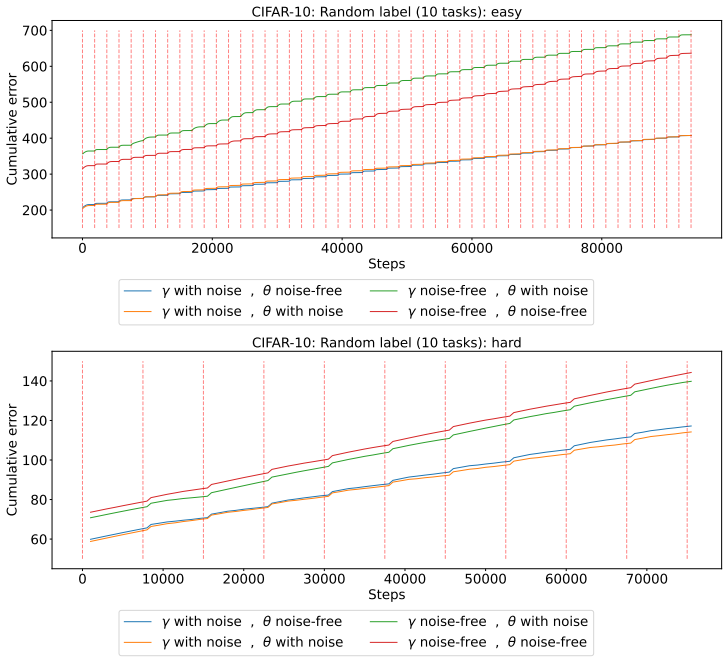

Figure 7: Impact of noise in learning γ and θ. The top row shows results on **easy** variant of *Random label CIFAR-10*, whereas the bottom row shows results on **hard** variant of this task.

4.6 Expressivity of γ

In this section we study the impact of different expressivity of γ on the performance. A normal version of our method is phrased in a way that γ is learned for each parameter. Nevertheless, we study the performance of our method where γ could be learned per layer or as a global scalar. The results are given in Figure 8. We see generally that as expressivity of γ increases, the cumulative error decreases. This makes sense since the neural networks are complex non-linear mappings and finding one scalar might not perform very well.

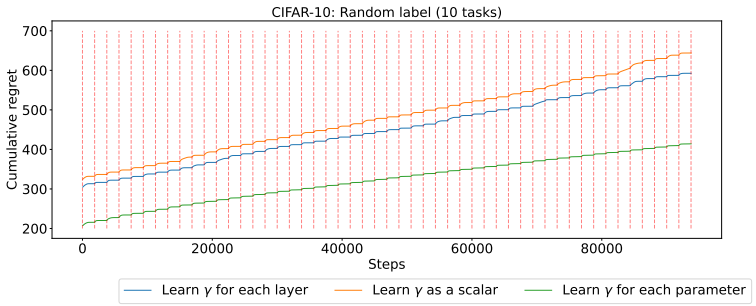

Figure 8: Impact of the how expressive the γ on the performance. The experiment is run on *Random Label CIFAR-10: easy* task.

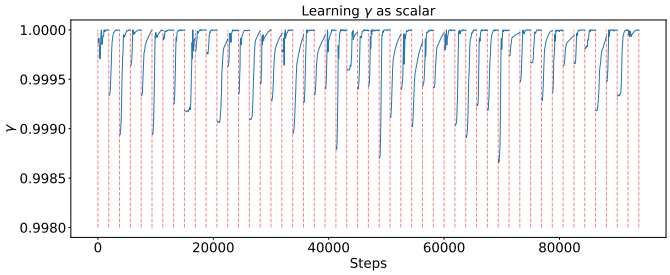

Figure 9: Learning of γ when it is learned as a scalar. The experiment is run on *Random Label CIFAR-10: easy* task.

Moreover, in Figure 9 we show the dynamics of learning γ parameter when it is learned as a scalar. On top of that, in Figure 10 and in Figure 11 we show the different γ for different layers of the neural network. Overall, we see that learning parameter γ captures the non-stationarity of the data, we see that γ generally decreases at task boundaries and increases withing the task. This is exactly the behaviour we would ancitipate. Moreover, when we look at the dynamics of γ for different layers, we see that it is much more complex, especially for biases. This suggests that this mechanism could be interesting when applied to complex non-stationary problems.

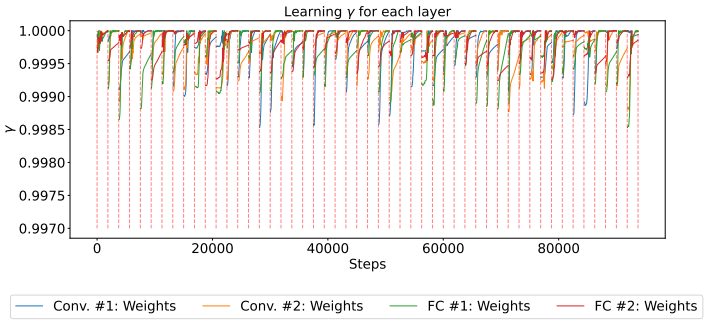

Figure 10: Learning of γ when it is learned per layer. Here we show the γ corresponding to the weight matrices. The experiment is run on *Random Label CIFAR-10: easy* task.

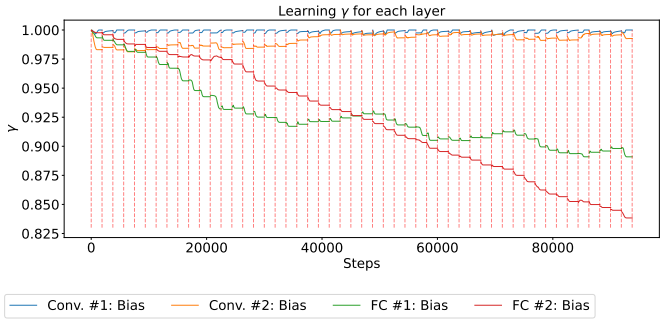

Figure 11: Learning of γ when it is learned per layer. Here we show the γ corresponding to the biases. The experiment is run on *Random Label CIFAR-10: easy* task.