Uncertainty-based Continual Learning with Adaptive Regularization

Hongjoon Ahn[*], **Sungmin Cha**[*], **Donggyu Lee and Taesup Moon**
Department of Electrical and Computer Engineering
Sungkyunkwan University, Suwon, Korea 16419
{hong0805, csm9493, ldk308, tsmoon}@skku.edu

Abstract

We introduce a new regularization-based continual learning algorithm, dubbed as Uncertainty-regularized Continual Learning (UCL), which builds on traditional online Bayesian learning framework with variational inference. Recently proposed regularization-based methods for preventing catastrophic forgetting have good performance with fixed network capacity. However, these methods have two significant drawbacks: a) considerable memory cost of additional parameters for determining regularization strength and b) the absence of task adaptive gracefully forgetting method which can prevent performance degradation in training new tasks. In this paper, we show UCL can solve these two problems by introducing a fresh interpretation of the traditional Gaussian mean-field approximation in variational inference. Instead of giving uncertainty on individual weights, we introduce the uncertainty of nodes, which drastically decrease the additional parameters for regularization terms. Moreover, we add additional regularization term which gives flexibility to control the actively learning parameters for a new task by gracefully forgetting what was learned before. In results, UCL outperforms most of recent state-of-the-art baselines on both supervised learning and reinforcement learning benchmarks, and by using Bayesian learning framework with variational inference, UCL can carry out both exploration and lifelong learning in reinforcement learning.

1 Introduction

Continual learning, also called as lifelong learning, is a long-standing open problem in machine learning in which data from multiple tasks continuously arrive and the learning algorithm should constantly adapt to new tasks as well as not forget what it has learned in the past. The main challenge is to resolve the so-called *stability-plasticity dilemma* [2, 19]. Namely, a learning agent should be able to preserve what it has learned, but focusing too much on the stability may hinder it from quickly learning a new task. On the other hand, when the agent focuses too much on the plasticity, it tends to quickly forget what it has learned. Particularly, for the artificial neural network (ANN)-based models, which became the mainstream of the machine learning methods, it is well-known that they are prone to such *catastrophic forgetting* phenomenon [18, 4]. As opposed to the ANNs, humans are able to maintain the obtained knowledge while learning a new task, and the forgetting in human brain happens gradually rather than drastically. This difference motivates active research in developing neural network based continual learning algorithms.

As given in a comprehensive survey [21] on this topic, approaches for tackling the catastrophic forgetting in neural network based continual learning can be roughly grouped into three categories: regularization-based [15, 13, 31, 20], dynamic network architecture-based [24, 30], and dual memory system-based [23, 16, 28, 10]. While each category has its own merit, of particular interest are the

[*]Equal contribution.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

regularization-based methods, since they pursue to maximally utilize the limited network capacity by imposing constraints on the update of the network given a new task. Computationally, they typically are realized by adding regularization terms that penalize the changes in the network parameters when learning a new task. This approach makes sense since it is well-known that neural network models are highly over-parametrized, and once successful, it can be also complementary to other approaches since it can lead to the efficient usage of network capacity as the number of tasks grows, as in [26].

The recent state-of-the-art regularization-based methods typically implement the per-parameter regularization parameters based on several different principles inferring the importance of each parameter for given tasks; e.g., diagonal Fisher information matrix for EWC [13], variance term associated with each weight parameter for VCL [20], and the path integrals of the gradient vector fields for SI [31]. While these methods are shown to be very effective in several continual learning benchmarks, one common caveat is that the amount of the memory required to store the model is twice the original neural network parameters, since they need to store the individual regularization parameters. We note that this could be a limiting factor for being deployed with large network size.

In this paper, we propose a new regularization-based continual learning algorithm, dubbed as Uncertainty-regularized Continual Learning (UCL), that stores much smaller number of additional parameters for regularization terms than the recent state-of-the-arts, but achieves much better performance in several benchmark datasets. Our approach builds upon the Bayesian learning framework, but makes a fresh interpretation of the variational approximation based regularization term and defines a notion of "uncertainty" for each hidden node in the network. Namely, when sampling the weight parameters from the Gaussian distributions, we constrain the (learnable) variances of the weights of a node to have the *same* value and define the uncertainty of the node as the common variance used to sample the incoming weights. The regularization parameter of each weight is then set to be large when the uncertainty of either of the node that the weight connects is small, since the weights connected to a certain node should be less updated when a new task comes. As the uncertainty is only associated with nodes, we can easily see that the additional memory required for UCL to store the regularization parameters is much smaller than other methods. Moreover, we add two additional regularization terms to the objective function; one that promotes *freezing* the weights that are identified to be important (i.e., certain) for past tasks, and the other that gives flexibility to control the *actively* learning parameters for a new task by gracefully forgetting what was learned before. In results, we show our UCL outperforms recent state-of-the-art baselines on both supervised and reinforcement learning benchmarks, with much less number of parameters. We also visualize the learned variances of nodes in the network and show that the network capacity is adaptively controlled by learning the right amount of uncertainties as the number of tasks increases.

2 Related Work

Continual learning There are numerous approaches in continual learning and we refer the readers to [21] for an extensive review. We only list work relevant to our method. The main approach of regularization-based methods in continual learning is to identify the important weights for the learned tasks and penalize the large updates on those weights when learning a new task. LwF [15] contains task-specific layers, and keeps the similar outputs for the old tasks by knowledge distillation [9]. In EWC [13], the diagonal of the Fisher information matrix at the learned parameter of the given task is used for giving the relative regularization strength. An extended version of EWC, IMM [14], merged the posteriors based on the mean and the mode of the old and new parameters. SI [31] computes the parameter importance considering a path integral of gradient vector fields during the parameter updates . Additionally, VCL [20] adopts online Bayesian learning framework.

Some work approached continual learning differently than the regularization-based method for the limited network capacity case. PackNet [17] picks out task-specific weights based on the weight pruning method, which requires saving the binary masks of the weights. Instead of storing the heuristic masks, they can be learned from the training procedure. HAT [27] employs node-wise attention mechanism per layer using task identifier embedding. However, a critical limitation of HAT is that it needs to know the number of tasks *a priori*.

Variational inference In standard Bayesian learning, the main idea of learning is efficiently approximating the posterior distribution on the models. [6] introduces a practical variational inference technique for neural network, which suggested that variational parameters can be learned using back-propagation. Another approach in variational inference is [12] which introduces the approximated lower bound of likelihood, and learn variational parameter using re-parameterization trick. In [1], they

introduce Unbiased Monte Carlo, which also uses back-propagation, but many kinds of priors can be used in the Unbiased Monte Carlo. In addition, there are several practical methods for variational inference in neural networks, such as using dropout [5] or Expectation-Propagation [8].

3 Uncertainty-regularized Continual Learning (UCL)

3.1 Notations and a review on online Bayesian learning

Consider a discriminative neural network model, $p(\boldsymbol{y}|\mathbf{x}, \boldsymbol{\mathcal{W}})$, that returns a probability distribution over the output $\boldsymbol{y}$ given an input $\mathbf{x}$ and parameters $\boldsymbol{\mathcal{W}}$. In standard Bayesian learning, $\boldsymbol{\mathcal{W}}$ is assumed to be sampled from some prior distribution $p(\boldsymbol{\mathcal{W}}|\boldsymbol{\alpha})$ that depends on some parameter $\boldsymbol{\alpha}$, and after observing some data $\mathcal{D} = \{(\mathbf{x}_i, \boldsymbol{y}_i)\}_{i=1}^n$, obtaining the posterior $p(\boldsymbol{\mathcal{W}}|\boldsymbol{\alpha}, \mathcal{D})$ becomes the central problem to learn the model parameters. Since exactly obtaining the posterior becomes intractable, variational inference [1, 3, 6] instead tries to approximate this posterior with a more tractable distribution $q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta})$. The approximation is done by minimizing (over $\boldsymbol{\theta}$) the so-called *variational free energy*, which can be written as

$$\mathcal{F}(D, \boldsymbol{\theta}) = \mathbb{E}_{q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta})}[-\log p(D|\boldsymbol{\mathcal{W}})] + D_{KL}(q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta})||p(\boldsymbol{\mathcal{W}}|\boldsymbol{\alpha})), \tag{1}$$

in which $\log p(D|\boldsymbol{\mathcal{W}})$ is the log-likelihood of the data D determined by the model $p(\boldsymbol{y}|\mathbf{x}, \boldsymbol{\mathcal{W}})$, and $D_{KL}(\cdot)$ stands for the Kullback-Leibler divergence. Moreover, the commonly used $q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta})$ is the so-called Gaussian mean-field approximation, $q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta}) = \prod_i \mathcal{N}(w_i|\mu_i, \sigma_i)$, that is defined by the parameter $\boldsymbol{\theta} = (\boldsymbol{\mu}, \boldsymbol{\sigma})$, and $\boldsymbol{\theta}$ can be learned via so-called reparametrization trick [12] and the standard back-propagation.

In Bayesian learning framework, standard variational inference method can be applied to the continual learning setting. Namely, when a dataset for task t, $\mathcal{D}_t$ arrives, VCL solves to minimize

$$\mathcal{F}(D_t, \boldsymbol{\theta}_t) = \mathbb{E}_{q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta}_t)}[-\log p(D_t|\boldsymbol{\mathcal{W}})] + D_{KL}(q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta}_t)||q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta}_{t-1})) \tag{2}$$

over $\boldsymbol{\theta}_t = (\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t)$, in which $q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta}_{t-1})$ stands for the posterior learned after observing $\mathcal{D}_{t-1}$ acting as a prior for learning $q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta}_t)$. Note in (2), we can observe that the KL-divergence term naturally acts as a regularization term. In [20], they showed that the network learned by sequentially solving (2) for each task t can successfully combat the catastrophic forgetting problem to some extent.

However, we argue that this Bayesian approach has a couple of drawbacks as well. First, due to the Monte-Carlo sampling of the model weights for computing the likelihood term in (2), the time and space complexity for learning grows with the sample size. Second, since the variance term is defined for every weight parameter, the number of parameters to maintain becomes exactly twice the actual neural network weights. This becomes problematic when deploying a large-sized network, as is the case in modern deep learning. In this paper, we present a novel approach which can resolve above problems. Our key idea is derived from a fresh interpretation of the closed form of KL divergence term in (2) and the Bayesian neural network based model pruning [6, 1].

3.2 Interpreting KL-divergence and motivation of UCL

While the KL divergence in (2) acts as a generic regularization term, we give a closer look at it, particularly for the Gaussian mean-field approximation model. Namely, after some algebra and evaluating the Gaussian integral, the closed-form of $D_{KL}(q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta}_t)\|q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta}_{t-1}))$ becomes:

$$\frac{1}{2}\sum_{l=1}^{L}\Big[\underbrace{\Big\|\frac{\boldsymbol{\mu}_t^{(l)} - \boldsymbol{\mu}_{t-1}^{(l)}}{\boldsymbol{\sigma}_{t-1}^{(l)}}\Big\|_2^2}_{(a)} + \underbrace{\mathbf{1}^\top\Big\{\Big(\frac{\boldsymbol{\sigma}_t^{(l)}}{\boldsymbol{\sigma}_{t-1}^{(l)}}\Big)^2 - \log\Big(\frac{\boldsymbol{\sigma}_t^{(l)}}{\boldsymbol{\sigma}_{t-1}^{(l)}}\Big)^2\Big\}}_{(b)}\Big], \tag{3}$$

in which L is the number of layers in the network, $(\boldsymbol{\mu}_t^{(l)}, \boldsymbol{\sigma}_t^{(l)})$ are the mean and standard deviation of the weight matrix for layer l that are subject to learning for task t, $(\boldsymbol{\mu}_{t-1}^{(l)}, \boldsymbol{\sigma}_{t-1}^{(l)})$ are the same quantity that are learned up to the previous task, the fraction notation means the element-wise division between tensors, and $\|\cdot\|_2^2$ stands for the Frobenius norm of a matrix. Term (a) in (3) can be interpreted as a square of the Mahalanobis distance between vectorized $\boldsymbol{\mu}_t^{(l)}$ and $\boldsymbol{\mu}_{t-1}^{(l)}$, in which the covariance matrix is $\text{diag}\{(\boldsymbol{\sigma}_{t-1}^{(l)})^2\}$, and it acts as a regularization term for $\boldsymbol{\mu}_t^{(l)}$. Namely, when minimizing (3)

over $\boldsymbol{\theta}_t^{(l)} = (\boldsymbol{\mu}_t^{(l)}, \boldsymbol{\sigma}_t^{(l)})$, the inverse of the variance learned up to task $(t-1)$ is acting as per-weight regularization strengths for $\boldsymbol{\mu}_t^{(l)}$ deviating from $\boldsymbol{\mu}_{t-1}^{(l)}$. This makes sense since each element of $(\boldsymbol{\sigma}_{t-1}^{(l)})^2$ can be regarded as an *uncertainty* measure for the corresponding mean weight of $\boldsymbol{\mu}_{t-1}^{(l)}$, and a weight with small uncertainty should be treated as *important* such that high penalty is imposed when significantly getting updated for a new task t. Moreover, term (b) in (3), which is convex in $(\boldsymbol{\sigma}_t^{(l)})^2$ and is minimized when $\boldsymbol{\sigma}_t^{(l)} = \boldsymbol{\sigma}_{t-1}^{(l)}$, is acting as a regularization term for $\boldsymbol{\sigma}_t^2$. Note it promotes to preserve the learned uncertainty measure when updating for a new task. This also makes sense for preventing catastrophic forgetting since the weights identified as important in previous tasks should be kept as important for future tasks as well such that the weight does not get updated too much by term (a). Based on this interpretation, we now modify each term and devise our UCL.

3.3 Modifying term (a)

We modify term (a) in (3) based on the following three intuitions. First, instead of maintaining the uncertainty measure for each mean weight parameter of $\boldsymbol{\mu}_t$, we devise a notion of uncertainty for each *node* of the network. Second, based on the node uncertainty, we set high regularization strength for a weight when either of nodes it connects has low uncertainty. Third, we add additional ℓ_1-regularizer such that a weight gets even more stringent penalty for getting updated when the weight has large magnitude or low uncertainty, inspired by [1, 6]. We elaborate each of these intuitions below.

While maintaining the weight-level importance as in other work [13, 20, 31] makes sense, we believe maintaining the importance (or uncertainty in our case) at the level of node makes more sense, not only for the purpose of reducing the model parameters, but also because the node value (or the activation) is the basic unit for representing the learned information from task. A similar intuition of working at node-level also appears in HAT [27], which devised a hard attention mechanism for important *nodes*, or dropout [29], which randomly drops *nodes* while training. In our variational setting, we define the uncertainty of a node as following; first constrain the incoming weights to the node to have the *same* variance parameters, then set that variance as the uncertainty of the node. For Gaussian mean-field approximation case, this constraint corresponds to adding zero-mean i.i.d Gaussian noise (with difference variances for different nodes) to the incoming weights when sampling for the variational learning.

For our second intuition, we derive the weight-level regularization scheme based on the following arguments. We believe there are two major sources that can cause the catastrophic forgetting of the past tasks when an weight update for a new task happens; 1) the negative transfer happening at the incoming weights of a node, and 2) the information loss happening at the outgoing weights of a node. Namely, suppose a node is identified as important, or as having low uncertainty, for the past task and the learning of a new task is taking place. From the perspective of that node, it is clear that when any of the incoming weights are significantly updated during the learning of the new task, the node's (important) representation for the past task will significantly get altered as the node will differently combine information from the lower layer, and hurt the past task accuracy. On the other hand, when the outgoing weights of the important node are significantly updated, the information of that node will get washed out during forward propagation, hence, it may not play important role in computing the prediction, causing the accuracy drop for the past task.

From above argument, we devise the weight-level regularization such that as weight gets high regularization strength when either of the node it connects has low uncertainty. This is realized by replacing term (a) of (3) with the following:

$$\frac{1}{2}\Big(\sum_{l=1}^{L}\Big\|\boldsymbol{\Lambda}^{(l)}\odot(\boldsymbol{\mu}_t^{(l)}-\boldsymbol{\mu}_{t-1}^{(l)})\Big\|_2^2\Big),\ \boldsymbol{\Lambda}_{ij}^{(l)}\triangleq\max\Big(\frac{\boldsymbol{\sigma}_{\text{init}}^{(l)}}{\sigma_{t-1,i}^{(l)}},\frac{\boldsymbol{\sigma}_{\text{init}}^{(l-1)}}{\sigma_{t-1,j}^{(l-1)}}\Big), \tag{4}$$

in which $\boldsymbol{\sigma}_{\text{init}}^{(l)}$ is the initial standard deviation hyperparameter for all weights on l-th layer, L is the number of layers in the network, $\boldsymbol{\mu}_t^{(l)}$ is the mean weight matrix for layer l and task t, $\odot$ is the element-wise multiplication between matrices, and $\boldsymbol{\Lambda}^{(l)}$ is a matrix that defines the regularization strength for the weight $\mu_{t,ij}^{(l)}$.

While (4) is a sensible replacement of term (a) in (3), our third intuition above is based on the observation that (4) does not take into account of the magnitude of the learned weights, i.e., $\boldsymbol{\mu}_{t-1}^{(l)}$. In [1, 6], they applied a heuristic for *pruning* network weights learned by variational inference; a weight

is important if the magnitude of the ratio μ/σ is large. Inspired by the pruning heuristic, we devise an additional ℓ_1-norm based regularizer

$$\sum_{l=1}^{L}(\boldsymbol{\sigma}_{\text{init}}^{(l)})^2\left\|\left(\frac{\boldsymbol{\mu}_{t-1}^{(l)}}{\boldsymbol{\sigma}_{t-1}^{(l)}}\right)^2\odot(\boldsymbol{\mu}_t^{(l)}-\boldsymbol{\mu}_{t-1}^{(l)})\right\|_1, \tag{5}$$

in which the division and square inside the ℓ_1-norm should be understood as the element-wise operation. Note $\boldsymbol{\sigma}_{t-1}^{(l)}$ has the same dimension as $\boldsymbol{\mu}_{t-1}^{(l)}$, and the i-th row of $\boldsymbol{\sigma}_{t-1}^{(l)}$ has the same variance value associated with the i-th node in layer l. Thus, in (5), if the ratio $(\mu_{t-1,ij}^{(l)}/\sigma_{t-1,i}^{(l)})^2$ is large, the ℓ_1-norm will promote sparsity and $\mu_{t,ij}^{(l)}$ will tend to *freeze* to $\mu_{t-1,ij}^{(l)}$.

3.4 Modifying term (b)

Regarding term (b) in (3), we can also devise a similar loss on the uncertainties associated with nodes. As mentioned in Section 3.2, the loss will promote $\boldsymbol{\sigma}_t^{(l)}=\boldsymbol{\sigma}_{t-1}^{(l)}$, meaning that once a node becomes important at task $(t-1)$, it tends to stay important for a new task as well. While this makes sense for preventing the catastrophic forgetting as it may induce high regularization parameters for penalties in (4) and (5), one caveat is that the network capacity can quickly fill up when the number of tasks grows. Therefore, we choose to add one more regularization term to term (b) in (3),

$$\sum_{l=1}^{L}\frac{1}{2}\mathbf{1}^\top\left(\left(\frac{\boldsymbol{\sigma}_t^{(l)}}{\boldsymbol{\sigma}_{t-1}^{(l)}}\right)^2-\log\left(\frac{\boldsymbol{\sigma}_t^{(l)}}{\boldsymbol{\sigma}_{t-1}^{(l)}}\right)^2+(\boldsymbol{\sigma}_t^{(l)})^2-\log(\boldsymbol{\sigma}_t^{(l)})^2\right), \tag{6}$$

which forces $\boldsymbol{\sigma}_t^{(l)}$ to get close to $\sqrt{2}\boldsymbol{\sigma}_t^{(l)}$ when minimized. Therefore, if a node becomes uncertain when training current task, the regularization strength becomes smaller. The detailed derivation of optimal point of (6) is in Supplementary material. Since our initial standard deviation $\boldsymbol{\sigma}_{\text{init}}^{(l)}$ is usually set to be small, the additional term in (6) compared to term (b) in (3) will tend to increase the number of "actively" learning nodes that have incoming weights with sufficiently large variance values for *exploration*. Moreover, when a new task arrives while most of the nodes have low uncertainty, (6) will force some of them to increase the uncertainty level to learn the new task, resulting in *gracefully* forgetting the past tasks.

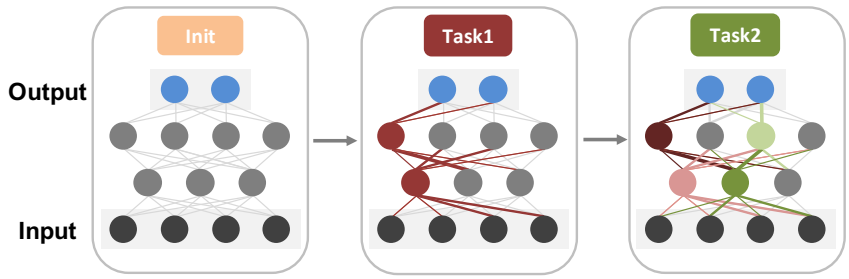


Figure 1: Colored hidden nodes and edges denote important nodes and highly regularized weights due to (4), respectively. The width of colored edge denotes the regularization strength of (5). Note as new task comes the uncertainty level of a node can vary due to (6), respresented with color changes.

3.5 Final loss function for UCL

Combining (4), (5), and (6), the final loss function for our UCL for task t becomes

$$-\log p(D_t|\boldsymbol{\mathcal{W}})+\sum_{l=1}^{L}\Bigg[\left(\frac{1}{2}\left\|\boldsymbol{\Lambda}^{(l)}\odot(\boldsymbol{\mu}_t^{(l)}-\boldsymbol{\mu}_{t-1}^{(l)})\right\|_2^2+(\boldsymbol{\sigma}_{\text{init}}^{(l)})^2\left\|\left(\frac{\boldsymbol{\mu}_{t-1}^{(l)}}{\boldsymbol{\sigma}_{t-1}^{(l)}}\right)^2\odot(\boldsymbol{\mu}_t^{(l)}-\boldsymbol{\mu}_{t-1}^{(l)})\right\|_1\right)$$
$$+\frac{\beta}{2}\mathbf{1}^\top\left\{\left(\frac{\boldsymbol{\sigma}_t^{(l)}}{\boldsymbol{\sigma}_{t-1}^{(l)}}\right)^2-\log\left(\frac{\boldsymbol{\sigma}_t^{(l)}}{\boldsymbol{\sigma}_{t-1}^{(l)}}\right)^2+(\boldsymbol{\sigma}_t^{(l)})^2-\log(\boldsymbol{\sigma}_t^{(l)})^2\right\}\Bigg], \tag{7}$$

which is minimized over $\{\boldsymbol{\mu}_t^{(l)}, \boldsymbol{\sigma}_t^{(l)}\}_{l=1}^{L}$ and has two hyperparameters, $\boldsymbol{\sigma}_{\text{init}}^{(l)}$ and β. The former serves as a pivot determining the degree of uncertainty of a node, and the latter controls the increasing or decreasing speed of $\boldsymbol{\sigma}_t^{(l)}$. As elaborated in above sections, it is clear that the *uncertainty* of a node plays a critical role in setting the regularization parameters, hence. justifies the name UCL. Moreover, unlike in Monte Carlo sampling, we sample the model parameters every iteration, and the number of sampling is 1 for each iteration. At the beginning epoch of task t, we sample from $q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta}_t)$ with $\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1}$, then continue to update $\boldsymbol{\theta}_t$ in the subsequent iterations. Illustration of the regularization mechanism of UCL is given in Figure 1.

4 Experiments

4.1 Initializing standard deviation

To initialize $\boldsymbol{\sigma}_{\text{init}}^{(l)}$, we adopted two approaches. First, we set $\boldsymbol{\sigma}_{\text{init}}^{(l)}$ to constant value for all layers, which means $\boldsymbol{\sigma}_{\text{init}}^{(l)} = \boldsymbol{\sigma}_{\text{init}}^{(l-1)} = c$. In experiments using only fully connected layers, we used this approach. However, in case of using convolutional neural network, since we use much deeper network, the variance of network output is exponentially increasing as the model architecture is going deeper which may cause unstable training. Therefore, to make the entire training process stable while adjusting the variance of output, we initialize $\boldsymbol{\sigma}_{\text{init}}^{(l)}$ adaptive to the network architecture. The detailed explanation of initializing $\boldsymbol{\sigma}_{\text{init}}^{(l)}$ on convolutional neural network is in Supplementary material.

4.2 Supervised learning

We evaluate the performance of UCL together with EWC [13], SI [31], VCL [20], and HAT [27]. We also make a comparison with Coreset VCL as in original paper[20]. The number of sampling weights was 10 for VCL, and 1 for UCL. All of the results are averaged over 5 different seeds. The initial standard deviation of UCL, $\boldsymbol{\sigma}_{\text{init}}^{(l)}$, was set to 0.06. We fairly did the hyperparamter selection for all methods for fair comparison, and refer to Supplementary Materials for more details.

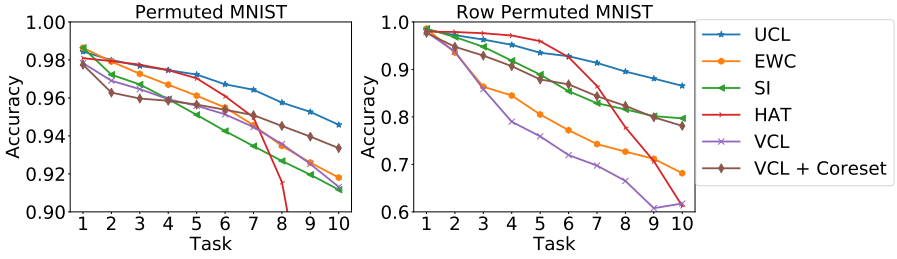


Figure 2: Experimental results on various Permuted MNIST with single head.

Permuted / Row Permuted MNIST We first test on the popular Permuted MNIST benchmark. We used fully connected single head networks of two hidden layers with 400 units followed by ReLU nonlinearities for all methods. We compare the average test accuracy over the learned tasks in Figure 2(left). After training on 10 tasks sequentially, EWC, SI, and VCL show little difference of performance among them achieving 91.8%, 91.1%, and 91.3% respectively. Although VCL with the coreset size of 200 makes an improvement of 2%, UCL outperforms all other baselines achieving 94.5%. Interestingly, HAT keeps almost the same average accuracy over first 5 tasks as UCL, however it starts to make a significant drop in accuracy after training task 7. This points out the limit of applying HAT in a single-headed network. As a variation of Permuted MNIST, we shuffled only rows of MNIST images pixels instead of shuffling all the image pixels, which we denote as Row Permuted MNIST. We empirically find that all algorithms are prone to forgetting more in Row Permuted MNIST. Looking at the accuracy scale, all the methods show severe degradation of performance compared to Permuted MNIST in Figure 2(right). This may be due to permuting of the correlated row blocks causing more weight changes in the network. After 10 tasks, UCL again achieved the highest average accuracy 86.5% in this experiment as well.

For a better understanding of our model, we visualize the standard deviations in all layers. Figure 3 shows the change of standard deviation as the training proceeds. After the model trained on task 1, we find that just a few of them become smaller than $\boldsymbol{\sigma}_{\text{init}}^{(l)}$, and most of them becomes much larger

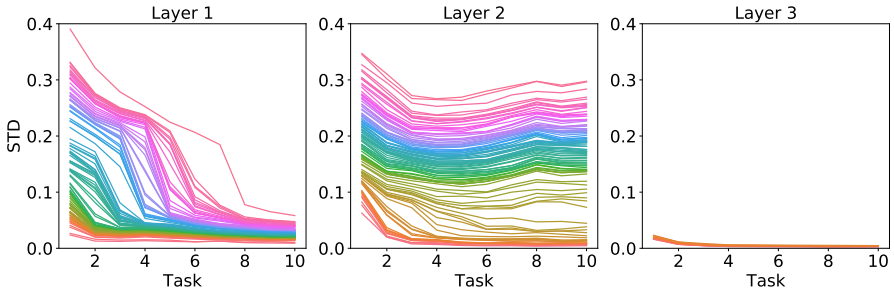


Figure 3: Standard deviation histogram in Permuted MNIST experiment. We randomly selected 100 standard deviations for layer 1 and 2. In layer 3, all 10 nodes are shown.

in the first hidden layer. Interestingly, the uncertain nodes in layer 1 show a drastic decline of their standard deviations at a specific task as the learning progresses, which means the model had to make them certain for adapting to the new task. On the other hand, all the nodes in the output layer had to reduce their uncertainty as early as possible considering even a tiny randomness can lead to a totally different prediction. Most of the nodes in layer 2, in addition, do not show a monotonic tendency. This can be interpreted as many of them need not belong to a particular task. As a result, this gives the plasticity and gracefully forgetting in our model.

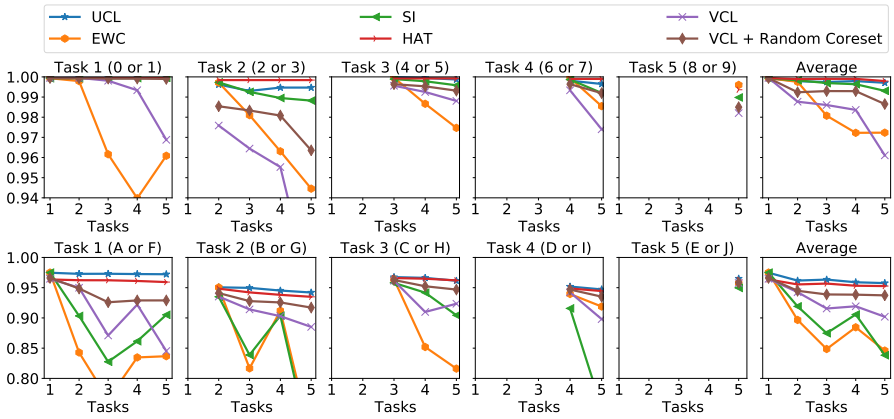


Figure 4: Experimental results on Split MNIST(top) and Split notMNIST(bottom)

Split MNIST We test also in the splitted dataset setting that each task consists of 2 consecutive classes of MNIST dataset. This benchmark was used in [31, 20]. Two successive digits starting from 0 in MNIST data compose a task, resulting in total 5 tasks. For this experiment, we trained the algorithms on fully connected networks with multi-head outputs, which have 2 hidden layers with 256 units followed by ReLU nonlinearities. At the top row of Figure 4, we compare the test accuracy of each task together with the average accuracy over all observed tasks at the right end. UCL accomplishes the same 5 tasks average accuracy as HAT; 99.7%, which is slightly better than the results of SI and VCL with coreset, 99.0%, and 98.7%, respectively, and significantly outperforms EWC and VCL. Note that HAT needs to know the number of tasks a priori while UCL need not.

Split notMNIST In this experiment, we make an assessment on another splitted dataset tasks with notMNIST dataset, which has 10 character classes. We split the characters of notMNIST into 5 groups same as VCL[20]: A/F, B/G, C/H, D/I, and E/J. Here we employ fully connected networks with four hidden layers with 150 units followed by ReLU. Unlike the previous experiment, SI shows similar result to EWC around 84% average accuracy, and VCL attains a better result of 90.1%(bottom row of Figure 4). Our UCL again makes an outstanding result of 95.7%, that is higher than HAT and VCL with coreset: 95.2% and 93.7%, respectively.

Comparison of model parameters Table 3 shows the number of model parameters in each experiment. Vanilla stands for the base network architecture of all methods. It is shown that our algorithm, UCL, has fewer parameters than other regularization based approaches. Especially, UCL has almost

Table 1: The number of parameters used for each benchmark.

Dataset\Methods	Vanilla	UCL	EWC	SI	HAT	VCL
Permuted MNIST	478K	960K	1435K	1435K	486K	1914K
Split MNIST	270K	538K	808K	808K	272K	1077K
Split notMNIST	187K	375K	559K	559K	190K	749K

half the number of VCL parameters, based on similar variational framework. Although HAT shows the least number of parameters, it has the drawback of requiring to know the number of task *a priori*.

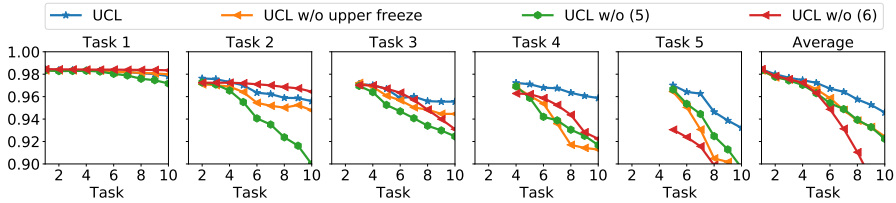
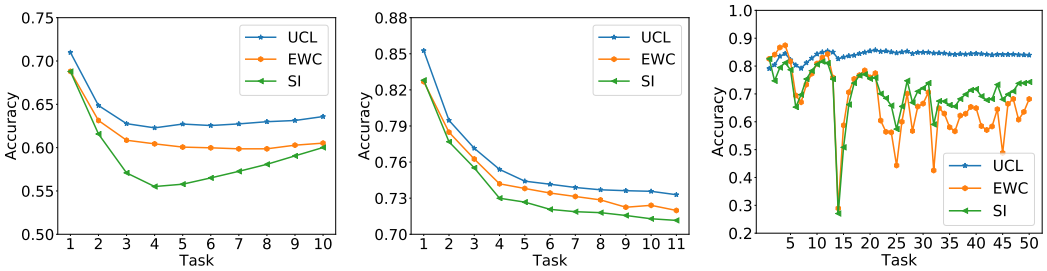


Figure 5: Ablation study in Permuted MNIST. Each line denote the test accuracy.

Ablation study Figure 5 shows an ablation study on UCL's regularization. We tested on three cases with Permuted MNIST and all experimental settings are same as Figure 2. "UCL w/o upper freeze" stands for using $\mathbf{\Lambda}_{ij}^{(l)} = \boldsymbol{\sigma}_{\text{init}}^{(l)}/\sigma_{t-1,i}^{(l)}$ only in (4), and we observe regularizing the outgoing weights of an important node in UCL very important. "UCL w/o (5)" stands for the removing (5) from (7), and we clearly see the pruning heuristic based weight freezing is also very important. "UCL w/o (6)" stands for not using (6) and it shows that while the accuracy of Task 1 & 2 are even higher than UCL, but the accuracy drastically decreases after Task 3. This is because the capacity of the model decreases rapidly due to lack of "actively" learning weights.



(a) Split CIFAR-100 (b) Split CIFAR-10/100 (c) Omniglot

Figure 6: Experiments on supervised learning using convolutional neural network

Convolutional neural network To check the effectiveness of UCL beyond the MNIST tasks, we experimented our UCL on three additional datasets, Split CIFAR-100, Split CIFAR10/100 and Omniglot. For Split CIFAR-100, each task consists of 10 consecutive classes of CIFAR-100, for Split CIFAR-10/100, we combined CIFAR-10 and Split CIFAR-100, and for Omniglot, each alphabet is treated as a single task, and we used all 50 alphabets. For Omniglot, as in [26], we rescaled all images to 28×28 and augmented the dataset by including 20 random permutations (rotations and shifting) for each image. For these datasets, unlike the experiments in fully connected layer, we used deeper CNN architectures, for which the notion of *uncertainty* in the convolution layer is defined for each *channel* (i.e., filter). We used multi-head outputs for all experiments, and 8 different random seed runs are averages for all dataset. The details of experiment on convolution neural network are in Supplementary material. In Figure 6, we compared with EWC and SI and carried out extensive hyperparameter search for fair comparison. We did not compare with VCL since it did not have any results on vision datasets with CNN architecture.

In Split CIFAR-100, EWC and SI achieves 60.5% and 60.0% respectively. However, UCL outperforms SI and EWC achieving 63.4%. In a slightly different task, Split CIFAR-10/100, which prevents overfitting on Split CIFAR-100 using pre-training on CIFAR-10, UCL also outperforms baselines which achieves 73.2%. In Omniglot task, though UCL is unstable in learning first task, it eventually

achieves 83.9% average accuracy on all 50 tasks. However, EWC and SI only achieves 68.1% and 74.2% respectively, which are much lower than UCL. From the results, we clearly observe that UCL outperforms the baselines for all tasks as well, stressing the effectiveness of UCL on diverse datasets.

4.3 Reinforcement learning

To verify our method in various aspects, we also conducted an experiment on reinforcement learning. Roboschool [25] consists of 12 tasks and each task has a different shape of state and continuous action space, and goal. From these tasks, we randomly chose the eight tasks and sequentially train a model by following order, {*Walker-HumanoidFlagrun-Hooper-Ant-InvertedDoublePendulum-Cheetah-Humanoid-InvertedPendulum*}. We also trained the model using PPO [25] as a training algorithm and selected EWC and Fine-tuning as baselines. We trained and tested all baselines in exactly the same condition and had a parameter searching to find a optimal setting for baselines. More experimental details are described in Supplementary Materials. Figure 7 shows the accumulated sum of normalized rewards up to the learned task and Figure 8 shows the normalized rewards for each task, where the normalization was done for each task with the reward obtained by EWC (λ=10). Thus, the high cumulative sum corresponds to effectively combating the catastrophic forgetting (CF), and fine-tuning, which is known to suffer from CF, hovering around 1 makes sense. In Figure 7, we observe UCL significantly outperforms EWC and different σ values have little effect on the final reward. We believe the reason why EWC does not excel as in Figure 4B of the original EWC paper[13] is because we consider pure continual learning setting, while the original EWC paper allows learning tasks multiple times in recurring fashion. A possible reason why UCL works so well in RL setting may be due to the by-product of our weight sampling procedure; namely, it enables effective exploration as in [22]. Figure 8 shows UCL overwhelmingly surpasses EWC in Task1 ans Task3, these make a big difference with EWC in the accumulated sum of normalized rewards in Figure 7. We experimentally checked that, the role of β for *gracefully* forgetting. The result of $\beta = 5\text{x}10^{-6}$ shows the better result of overcoming CF, from almost tasks, however, had a difficulty learning new tasks than $\beta = 5\text{x}10^{-5}$, from the result of Task 7 and 8. We stress that there are few algorithms in the literature that work well on both SL and RL continual learning setting, and our UCL is very competitive in that sense.

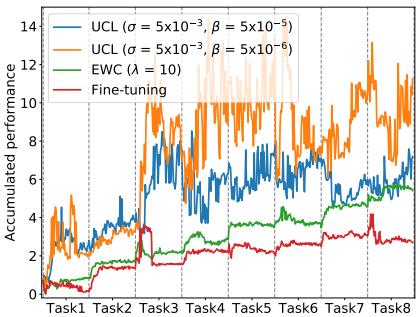


Figure 7: Accum. rewards for 8 RL tasks.

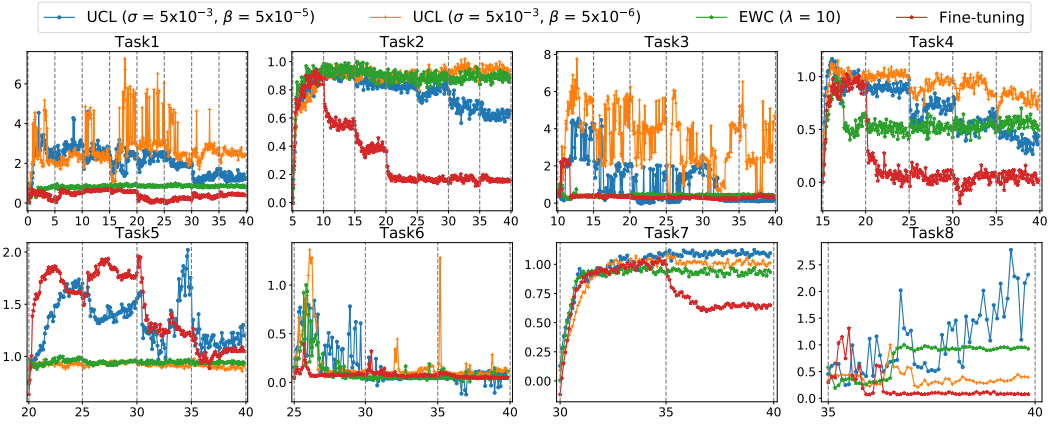


Figure 8: Experimental results on reinforcement learning.

5 Conclusion

We proposed a new uncertainty-based regularization method for overcoming catastrophic forgetting named UCL. Our method constrains the flexibility of weights by L1 and L2 regularization combined with variance regularization based on node-wise uncertainty. As a result, UCL showed impressive

experimental results that outperforms other state-of-the-art baselines in both supervised and reinforcement learning. UCL requires fewer parameters than others and also can employ gracefully forgetting past tasks. For future work, we plan to extend UCL to the convolutional neural networks.

6 Acknowledgement

We are supported by IITP.

References

[1] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1613–1622, Lille, France, 07–09 Jul 2015. PMLR.

[2] Gail A Carpenter and Stephen Grossberg. Art 2: Self-organization of stable category recognition codes for analog input patterns. *Applied optics*, 26(23):4919–4930, 1987.

[3] Geoffrey E. Hinton and Drew Van Camp. Keeping neural networks simple by minimizing the description length of the weights. *Proceedings of COLT-93*, 07 1999.

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

[5] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[6] Alex Graves. Practical variational inference for neural networks. In *Advances in neural information processing systems*, pages 2348–2356, 2011.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[8] José Miguel Hernández-Lobato and Ryan Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning*, pages 1861–1869, 2015.

[9] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[10] Ronald Kemker and Christopher Kanan. Fearnet: Brain-inspired model for incremental learning. *arXiv preprint arXiv:1711.10563*, 2017.

[11] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[12] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

[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, 2017.

[14] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4652–4662. Curran Associates, Inc., 2017.

[15] Zhizhong Li and Derek Hoiem. Learning without forgetting. In *ECCV*, 2016.

[16] David Lopez-Paz and Marc Aurelio Ranzato. Gradient episodic memory for continual learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6467–6476. Curran Associates, Inc., 2017.

[17] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[18] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[19] Martial Mermillod, Aurélia Bugaiska, and Patrick Bonin. The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects. *Frontiers in psychology*, 4:504, 2013.

[20] Cuong V. Nguyen, Yingzhen Li, Thang D. Bui, and Richard E. Turner. Variational continual learning. In *International Conference on Learning Representations*, 2018.

[21] German Ignacio Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *CoRR*, abs/1802.07569, 2018.

[22] Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y. Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. Parameter space noise for exploration. In *International Conference on Learning Representations*, 2018.

[23] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017.

[24] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *CoRR*, abs/1606.04671, 2016.

[25] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[26] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4528–4537, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[27] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4548–4557, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[28] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2990–2999. Curran Associates, Inc., 2017.

[29] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal Of Machine Learning Research*, 15:1929–1958, 2014.

[30] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. In *International Conference on Learning Representations*, 2018.

[31] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3987–3995, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

7 Derivation of (3)

Let assume $q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta}_t)$ and $q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta}_{t-1})$ as below.

$$\begin{aligned} q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta}_t) &= N(\boldsymbol{\mathcal{W}}|\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t^2) \\ &= \prod_{d=1}^{D} \frac{1}{\sqrt{2\pi\sigma_{t,d}^2}} \exp\Big(-\frac{1}{2}\frac{(\mathcal{W}_d - \mu_{t,d})^2}{\sigma_{t,d}^2}\Big) \\ q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta}_t) &= N(\boldsymbol{\mathcal{W}}|\boldsymbol{\mu}_{t-1}, \boldsymbol{\sigma}_{t-1}^2) \\ &= \prod_{d=1}^{D} \frac{1}{\sqrt{2\pi\sigma_{t-1,d}^2}} \exp\Big(-\frac{1}{2}\frac{(\mathcal{W}_d - \mu_{t-1,d})^2}{\sigma_{t-1,d}^2}\Big) \end{aligned}$$

Then, $D_{KL}(q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta}_t)||q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta}_{t-1}))$ is as below.

$$\begin{aligned} &D_{KL}(q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta}_t)||q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta}_{t-1})) \\ &= \int q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta}_t) \log \frac{q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta}_t)}{q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta}_{t-1})} d\boldsymbol{\mathcal{W}} \\ &= \int N(\boldsymbol{\mathcal{W}}|\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t) \log \frac{N(\boldsymbol{\mathcal{W}}|\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t^2)}{N(\boldsymbol{\mathcal{W}}|\boldsymbol{\mu}_{t-1}, \boldsymbol{\sigma}_{t-1}^2)} d\boldsymbol{\mathcal{W}} \\ &= \sum_{d=1}^{D} \int N(\boldsymbol{\mathcal{W}}|\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t^2) \log \frac{N(\mathcal{W}_d|\mu_t, \sigma_t^2)}{N(\mathcal{W}_d|\mu_{t-1,d}, \sigma_{t-1,d}^2)} d\boldsymbol{\mathcal{W}} \\ &= \sum_{d=1}^{D} \int \prod_{i=1}^{D} N(\mathcal{W}_i|\mu_{t,i}, \sigma_{t,i}^2) \log \frac{N(\mathcal{W}_d|\mu_{t,d}, \sigma_{t,d}^2)}{N(\mathcal{W}_d|\mu_{t-1,d}, \sigma_{t-1,d}^2)} d\mathcal{W}_1 ... \mathcal{W}_d .. \mathcal{W}_D \\ &= \sum_{d=1}^{D} \int N(\mathcal{W}_d|\mu_{t,d}, \sigma_{t,d}^2) \log \frac{N(\mathcal{W}_d|\mu_{t,d}, \sigma_{t,d}^2)}{N(\mathcal{W}_d|\mu_{t-1,d}, \sigma_{t-1,d}^2)} d\mathcal{W}_d \Big(\int \prod_{i \neq d}^{D} N(\mathcal{W}_i|\mu_{t,i}, \sigma_{t,i}^2) d\mathcal{W}_i \Big) \\ &= \sum_{d=1}^{D} \int N(\mathcal{W}_d|\mu_{t,d}, \sigma_{t,d}) \log \frac{N(\mathcal{W}_d|\mu_{t,d}, \sigma_{t,d})}{N(\mathcal{W}_d|\mu_{t-1,d}, \sigma_{t-1,d})} d\mathcal{W}_d \\ &= \sum_{d=1}^{D} \int N(\mathcal{W}_d|\mu_{t,d}, \sigma_{t,d}^2) \log N(\mathcal{W}_d|\mu_{t,d}, \sigma_{t,d}^2) d\mathcal{W}_d \qquad (8) \\ &- \int N(\mathcal{W}_d|\mu_{t,d}, \sigma_{t,d}^2) \log N(\mathcal{W}_d|\mu_{t-1,d}, \sigma_{t,d}^2) d\mathcal{W}_d \qquad (9) \end{aligned}$$

For simplicity, we decompose $D_{KL}(q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta}_t)||q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta}_{t-1}))$ as (8) and (9). At first, the closed form of (8) is as below.

$$\begin{aligned} &\int N(\mathcal{W}|\mu_{t,d}, \sigma_{t,d}^2) \log N(\mathcal{W}|\mu_{t,d}, \sigma_{t,d}^2) d\mathcal{W}_d \\ &= \int N(\mathcal{W}|\mu_{t,d}, \sigma_{t,d}^2) \log \frac{1}{\sqrt{2\pi\sigma_{t,d}^2}} \exp\Big(-\frac{1}{2}\frac{(\mathcal{W}_d - \mu_{t,d})^2}{\sigma_{t,d}^2}\Big) d\mathcal{W}_d \\ &= \int N(\mathcal{W}_d|\mu_{t,d}, \sigma_{t,d}^2) \log \frac{1}{\sqrt{2\pi\sigma_{t,d}^2}} \exp\Big(-\frac{1}{2}\frac{(\mathcal{W}_d - \mu_{t,d})^2}{\sigma_{t,d}^2}\Big) d\mathcal{W}_d \\ &= \log \frac{1}{\sqrt{2\pi\sigma_{t,d}^2}} + \int N(\mathcal{W}_d|\mu_{t,d}, \sigma_{t,d}^2) \Big(-\frac{1}{2}\frac{(\mathcal{W}_d - \mu_{t,d})^2}{\sigma_{t,d}^2}\Big) d\mathcal{W}_d \\ &= \log \frac{1}{\sqrt{2\pi\sigma_{t,d}^2}} - \frac{1}{2\sigma_{t,d}^2} Var[\mathcal{W}_d]_{N(\mu_{t,d}, \sigma_{t,d}^2)} \\ &= -\frac{1}{2}\log 2\pi\sigma_{t,d}^2 - \frac{1}{2} \qquad (10) \end{aligned}$$

And then, (9) is as below.

$$
\begin{aligned}
&\int N(\mathcal{W}|\mu_{t,d},\sigma_{t,d}^2)\log N(\mathcal{W}|\mu_{t-1,d},\sigma_{t-1,d}^2)d\mathcal{W}_d \\
&=\int N(\mathcal{W}|\mu_{t,d},\sigma_{t,d}^2)\log\frac{1}{\sqrt{2\pi\sigma_{t-1,d}^2}}\exp\Big(-\frac{1}{2}\frac{(\mathcal{W}_d-\mu_{t-1,d})^2}{\sigma_{t-1,d}^2}\Big)d\mathcal{W}_d \\
&=\int N(\mathcal{W}_d|\mu_{t,d},\sigma_{t,d}^2)\log\frac{1}{\sqrt{2\pi\sigma_{t-1,d}^2}}\exp\Big(-\frac{1}{2}\frac{(\mathcal{W}_d-\mu_{t-1,d})^2}{\sigma_{t-1,d}^2}\Big)d\mathcal{W}_d \\
&=\log\frac{1}{\sqrt{2\pi\sigma_{t-1,d}^2}}+\int N(\mathcal{W}_d|\mu_{t,d},\sigma_{t,d}^2)\Big(-\frac{1}{2}\frac{(\mathcal{W}_d-\mu_{t-1,d})^2}{\sigma_{t-1,d}^2}\Big)d\mathcal{W}_d \\
&=\log\frac{1}{\sqrt{2\pi\sigma_{t-1,d}^2}}-\frac{1}{2\sigma_{t-1,d}^2}E[(\mathcal{W}_d-\mu_{t-1,d})^2]_{N(\mathcal{W}_d|\mu_{t,d},\sigma_{t,d}^2)} \\
&=-\frac{1}{2}\log 2\pi\sigma_{t-1,d}^2-\frac{1}{2\sigma_{t-1,d}^2}\Big(\sigma_{t,d}^2+(\mu_{t,d}-\mu_{t-1,d})^2\Big)(\because E[(X-a)^2]=Var[X]+(E[X]-a)^2)
\end{aligned}
\tag{11}
$$

Therefore, combining (10) and (11), $D_{KL}(q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta}_t)||q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta}_{t-1}))$ can be as

$$
\begin{aligned}
&D_{KL}(q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta}_t)||q(\boldsymbol{\mathcal{W}}|\boldsymbol{\theta}_{t-1})) \\
&=\sum_{d=1}^{D}\frac{1}{2\sigma_{t-1,d}^2}(\mu_{t,d}-\mu_{t-1,d})^2+\frac{1}{2}\Big(\frac{\sigma_{t,d}^2}{\sigma_{t-1,d}^2}-\log\frac{\sigma_{t,d}^2}{\sigma_{t-1,d}^2}\Big)+C \\
&=\frac{1}{2}\Big\|\frac{\boldsymbol{\mu}_t-\boldsymbol{\mu}_{t-1}}{\boldsymbol{\sigma}_t}\Big\|_2^2+\frac{1}{2}\mathbf{1}^\top\Big(\frac{\boldsymbol{\sigma}_t^2}{\boldsymbol{\sigma}_{t-1}^2}-\log\frac{\boldsymbol{\sigma}_t^2}{\boldsymbol{\sigma}_{t-1}^2}\Big)+C
\end{aligned}
\tag{12}
$$

where $\mathbf{1}$ stands for all-1 vector with D dimensions. Therefore, the form of (12) is same as (3) in main paper.

8 Detailed explanation on initializing standard deviation

As mentioned in section 4.1, due to using much deeper architecture in convolutional neural networks, we used another initialization technique which is adaptive to model architecture. Our initialization method is highly motivated by [7]. As discussed in [7], we mainly considered ReLU activation for deriving the initialization method.

Forward Propagation Case Let assume a forward propagation in lth layer is

$$
\mathbf{y}_l=\boldsymbol{\mathcal{W}}_l\mathbf{h}_{l-1}+\mathbf{b}_l,
\tag{13}
$$

in which $\boldsymbol{\mathcal{W}}_l$, is sampled weights from some prior distribution $p(\boldsymbol{\mathcal{W}}|\alpha)$, $\mathbf{h}_{l-1}$ is the activated value of previous layer, and $\mathbf{b}_l$ is bias.

$$
\begin{aligned}
\mathbf{y}_l &= \mathcal{W}_l \mathbf{h}_{l-1} + \mathbf{b}_l \\
\mathbf{Var}[y_l] &= n_l \cdot \mathbf{Var}[w_l \cdot h_{l-1}] \\
\mathbf{Var}[y_l] &= n_l \cdot \mathbf{Var}[w_l] \cdot \mathbf{E}[h_{l-1}^2] \\
\mathbf{Var}[y_l] &= \frac{1}{2} n_l \cdot \mathbf{Var}[w_l] \cdot \mathbf{Var}[y_{l-1}] \\
\mathbf{Var}[y_L] &= \mathbf{Var}[y_1] \cdot \Big(\prod_{l=2}^{L} \frac{1}{2} n_l \cdot \mathbf{Var}[w_l] \Big) \\
\mathbf{Var}[w_l] &= \frac{2}{n_l} \\
\mathbf{Var}[w_l] &= \mathbf{Var}[\mu_l + \sigma_l \epsilon] = \mathbf{Var}[\mu_l] + \sigma_l^2
\end{aligned} \tag{14}
$$

8.1 Regularization on initial network

9 Additional experimental results

10 Implementation details

10.1 Supervised learning

10.1.1 Training details

Permuted MNIST / Row Permuted MNIST

We trained all of our baselines with mini batch size of 256 for 100 epochs other than VCL; 200 epochs for VCL. We also optimized them with learning rate 0.001 by Adam optimizer[11] But for HAT, we updated it by stochastic gradient descent(SGD) with learning rate 0.05. Fisher information matrix were computed using all training samples of a task. Regularization hyperparameters are compared as below :

- UCL - {0.0001, 0.001, 0.01, **0.02(row permuted)**, **0.03(permuted)**}
- EWC - {40, **400(best)**, 4000, 40000}
- SI - {0.01, **0.03(best)**, 0.1, 0.3, 0.5, 0.7, 1.0}
- HAT - {0.25, 0.5, **0.75(best)**, 1.0}
- VCL - not needed

Split MNIST

We use the whole training dataset of a task for batch size of VCL, and trained it for 120 epochs. The others are trained in the same way as previous experiment. Regularization hyperparameters are compared as below :

- UCL - {**0.0001(best)**, 0.001, 0.01, 0.02, 0.03(best)}
- EWC - {40, 400, **4000(best)**, 40000}
- SI - {0.01, 0.03, 0.1, 0.3, 0.5, 0.7, **1.0(best)**}
- HAT - {0.25, 0.5, **0.75(best)**, 1.0}
- VCL - not needed

Split notMNIST

The training settings are equal to those in Split MNIST. Hyperparameters are compared as below :

- UCL - {0.0001, **0.001**, 0.01, 0.02, 0.03(best)}
- EWC - {40, 400, **4000(best)**, 40000}

- SI - {0.01, 0.03, 0.1, **0.3(best)**, 0.5, 0.7, 1.0}
- HAT - {0.25, 0.5, **0.75(best)**, 1.0}
- VCL - not needed

10.2 Reinforcement learning

10.2.1 Information on selected eight tasks and a model

Table 2: Details on environments.

Task number	Task name	Observation shape	Action shape	Goal
1	Walker2d	(22,)	(3,)	Make robot run as fast as possible
2	HumanoidFlagrun	(44,)	(17,)	Make a 3D humanoid robot run towards a target
3	Hopper	(15)	(3,)	Make the hopper hop as fast as possible
4	Ant	(28)	(8,)	Make the creature walk as fast as possible
5	InvertedDoublePendulum	(3,)	(1,)	Keep a pole upright by moving the 1-D cart
6	HalfCheetah	(28,)	(8,)	Make the creature walk as fast as possible
7	Humanoid	(22,)	(6,)	Make robot run as fast as possible
8	InvertedPendulum	(3,)	(1,)	Keep a pole upright by moving the 1-D cart

Figure 2 shows details on environments that we are used in experimental section on reinforcement learning. We trained each model of baselines and UCL using these eight tasks. We used two fully connected hidden layers with 16 nodes, with one input layer and multiple output layers and set σ = 0.005 for an initialization of UCL, used *He init* for other baselines. The number of nodes in input layer is 44, the maximum size of the state space in selected eight tasks. Depending on the tasks, unused areas are filled with zeros. Each output layer is equal to the size of the action space of each task, and we used DiagGaussian as a output layer for training a continuous action.

10.2.2 Hyperparameters of PPO

We used PPO (Proximal Policy Optimization) to train a model for reinforcement learning. Figure 3 shows hyperparameters we used and these hyperparameters are applied to the all cases in reinforcement learning experiments equally.

Table 3: Details on environments.

Hyperparameters	Value
# of steps of each task	5m
# of processes	128
# of steps per iteration	64
PPO epochs	10
entropy coefficient	0
value loss coefficient	0.5
γ for accumulated rewards	0.99
λ for GAE	0.95
mini-batch size	64

10.3 Additional experimental results of σ = $1\mathbf{x}10^{-3}$

Figure 10 and 9 shows the experimental result with σ = $1\text{x}10^{-3}$ for UCL. The results of other baselines are same with the manuscript and each task of all experimental results is normalized by the maximum and minimum value of EWC. Comparing with the result of σ = $1\text{x}10^{-3}$ on the manuscript, we clearly see that the importance of choosing σ for UCL.

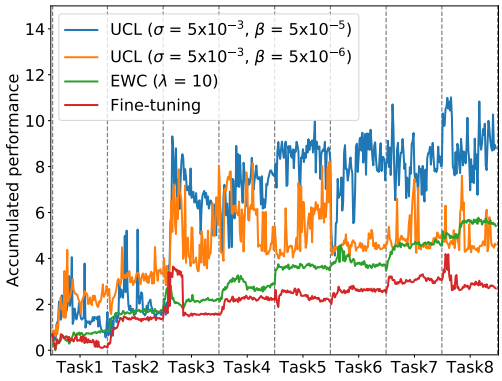


Figure 9: Experimental results on reinforcement learning.

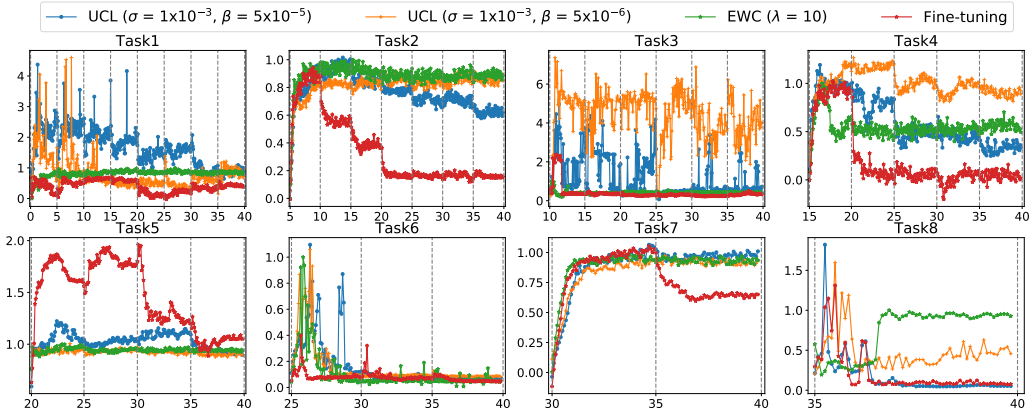


Figure 10: Experimental results on reinforcement learning.