Continual Learning by Asymmetric Loss Approximation with Single-Side Overestimation

Dongmin Park\textsuperscript{2}, Seokil Hong\textsuperscript{1}, Bohyung Han\textsuperscript{1}, and Kyoung Mu Lee\textsuperscript{1}

\textsuperscript{1}Seoul National University, Seoul, Korea
\textsuperscript{2}Samsung Electronics, Suwon, Korea
\{dmpark04\}@gmail.com, \{hongceo96, bhhan, kyoungmu\}@snu.ac.kr

Abstract

Catastrophic forgetting is a critical challenge in training deep neural networks. Although continual learning has been investigated as a countermeasure to the problem, it often suffers from requirements of additional network components and weak scalability to a large number of tasks. We propose a novel approach to continual learning by approximating a true loss function based on an asymmetric quadratic function with one of its sides overestimated. Our algorithm is motivated by the empirical observation that updates of network parameters affect target loss functions asymmetrically. In the proposed continual learning framework, we estimate an asymmetric loss function for the tasks considered in the past through a proper overestimation of its unobserved side in training new tasks, while deriving the accurate model parameter for the observed side. In contrast to existing approaches, our method is free from side effects and achieves the state-of-the-art results that are even close to the upper-bound performance on several challenging benchmark datasets.

1. Introduction

It is common to learn machine learning models for multiple tasks in an incremental manner when new tasks are given one by one, not in a batch. Continual learning is a concept to learn a large number of tasks sequentially without forgetting knowledge obtained from the preceding tasks, where data in the old tasks are not available any more during training new ones.

Catastrophic forgetting is a crucial but challenging problem in training deep neural networks. With a naive stochastic gradient descent (SGD) method, deep neural networks easily forget knowledge obtained from earlier tasks while adapting to new information quickly from incoming tasks \cite{25, 7}. This is mainly because, without any countermeasure, the simple optimization based on the losses of the new tasks is generally undesirable to retain the knowledge about the tasks learned in the past. The catastrophic forgetting problem limits the capability and potential of deep neural networks to be applied to dynamic real-world problems that require continuous adaptation to new environments. Continual learning or life-long learning is a well-known framework to handle the catastrophic forgetting problems. It can be categorized into one of the following three groups: architectural approach, functional ap-
proach, and structural regularization approach. Architectural and functional approaches typically need additional network components and/or batch processing. Structural regularization methods work well for a limited number of tasks, but often have scalability issues to many tasks.

This paper presents a novel continual learning framework, referred to as asymmetric loss approximation (ALASSO), which is effective to adapting a large number of tasks. ALASSO approximates true loss functions corresponding to all the previously considered tasks asymmetrically, where it overestimates the unobserved sides of the loss functions with respect to the previous tasks while deriving the optimal parameters for the quadratic approximation of the observed sides. Figure 1 illustrates the main concept of this paper; our algorithm computes the optimal parameter for quadratic approximation in the observed side (left) while using a steep surrogate quadratic function in the unobservable side (right). Our algorithm also decouple hyperparameters of the current surrogate loss approximation and the past loss difference reestimation. This approach is motivated by our observation that model parameter updates of deep neural networks affect the target losses asymmetrically, and that the use of overestimated loss functions is relatively safe for optimization without the training data of previous tasks.

In contrast to existing approaches, the proposed technique is free from the additional memory requirement for the previous tasks and the overhead of batch processing. Our algorithm achieves the state-of-the-art performance and is even close to the upper-bounds of several challenging benchmark datasets, including the permuted MNIST, the split CIFAR-10/CIFAR-100, and the split Tiny ImageNet. In particular, we demonstrate promising results in terms of scalability and robustness to a large number of tasks.

The contribution of our work is summarized as follows:

- We propose a novel continual learning framework by overestimating the unobserved side of a loss function in the current task and approximating the loss using an asymmetric quadratic function. This strategy facilitates a reliable loss estimation even without the training data of the previous tasks.

- Our algorithm provides the optimal solution of the quadratic loss approximation for the previous tasks, which allows to derive the best surrogate loss function in the observed side using the currently available training data only.

- The proposed technique achieves the state-of-the-art performance on several challenging benchmark datasets by large margins, including the permuted MNIST, the split CIFAR-10/CIFAR-100, and the split Tiny ImageNet.

The rest of the paper is organized as follows. We first discuss related works in Section 2, and present technical details of our method, ALASSO, in Section 4. Section 5 demonstrates our experimental results with analysis and Section 6 concludes this paper.

2. Related Work

Continual learning algorithms can be categorized into three groups [35]: architectural, functional, and structural regularization approaches. This section discusses the examples and characteristics of individual categories.

2.1. Architectural Approaches

The approaches in this category realize continual learning by freezing model parameters learned from previous tasks and/or providing limited architectural variations for learning new tasks [28, 21, 4, 19]. This framework often requires additional network components for new tasks, and the size of network gradually increases. This drawback limits the applicability to large-scale problems and is not appropriate for configurations with limited resources such as embedded systems. Some existing methods [23, 32, 14, 33, 12] use network compression techniques. Instead of physically adding new network components, it employs network compression algorithms to find unused or rarely used space in the target network and makes the free space to new tasks. It alleviates the drawbacks of the methods discussed earlier, but requires substantial overhead of batch processing for network compression.

2.2. Functional Approaches

In functional approaches [22, 13, 30], knowledge distillation is often incorporated to realize continual learning. The previously learned networks are fixed and used for feature computation in training new tasks. The new networks are encouraged to learn representations that are coherent to the features computed in the previous networks. However, the feature coherency of new input data with respect to old ones does not always ensure the output similarity of all possible input data. Existing functional approaches also require the previous networks to be stored and evaluated, which incurs additional computational overhead.
3. Synaptic Intelligence (SI)

The proposed approach, called ALASSO, is closely related to Synaptic Intelligence (SI) [35]. We first discuss the main idea of SI and then point out its critical problems before presenting our new algorithm.

3.1. Quadratic Approximation of Loss

SI is categorized as a structural regularization method, which employs a static network architecture and does not use additional memory throughout continual learning process. To prevent the catastrophic forgetting problem and maintain the performance with respect to previous tasks while adapting to a new task \( n \), this technique introduces a surrogate loss function \( \bar{L}_n^{-1} \), which approximates the loss of the previous tasks. Assuming that the surrogate loss is a quadratic function, the total loss function to learn the \( n \)th task, \( \bar{L}_n \), in terms of the model parameter \( \theta_k \) is approximated by

\[
\bar{L}_n = L^n + c \bar{L}_k^{-1} = L^n + c \sum_k \hat{\Omega}_k^{n-1} (\theta_k - \hat{\theta}_k^{n-1})^2,
\]

where \( L^n \) denotes a loss for the current task \( n \), \( \hat{\theta}_k \) is the weight in the \( k \)th dimension of the estimated parameter for the previous tasks, and \( \hat{\Omega}_k^{n-1} \) is a coefficient for the corresponding model parameter. \( c \) is a hyperparameter for the regularizer.

We minimize the total loss, \( \bar{L}_n \), with respect to \( \theta_k \), and obtain the optimized parameters, \( \hat{\theta}_k^n \). Assuming that the surrogate loss function for each parameter up to the \( n \)th task is defined by a quadratic function as

\[
L^n_s(\theta_k) = \hat{\Omega}_k^n \left( \theta_k - \hat{\theta}_k^n \right)^2,
\]

\( \hat{\Omega}_k^n \) is derived approximately by

\[
\hat{\Omega}_k^n \approx \frac{\omega_k^n}{\left( \hat{\theta}_k^n - \hat{\theta}_k^{n-1} \right)^2} + \hat{\Omega}_k^{n-1},
\]

where \( \omega_k^n \) denotes the difference between the losses for the task \( n \) before and after training the task, \( i.e., \omega_k^n = L^n(\hat{\theta}_k^{n-1}) - L^n(\hat{\theta}_k^n) \). Refer to [35] for more details about SI.

3.2. Underestimation of Loss

Since SI employs a symmetric loss function for approximation with respect to previous tasks, it can observe only a single side of the symmetric function along the trajectory of parameter update from \( \hat{\theta}_k^{n-1} \) to \( \hat{\theta}_k^n \) during the optimization process for the \( n \)th task. In other words, the surrogate loss

![Figure 2](image.png)

Figure 2. Illustration of asymmetric characteristics of a loss function in terms of each model parameter \( \theta_k \), where \( k \) is the index of model parameter. Each graph shows how much loss \( L^1 \) changes by updating each parameter \( \hat{\theta}_k^1 \) after learning the first task in the Permuted MNIST dataset.

2.3. Structural Regularization Approaches

Structural regularization approaches typically augment a penalty term to the original loss functions and discourage the updates of weights important to previously learned tasks. Elastic weight consolidation (EWC) [16] and synaptic intelligence (SI) [35] employ surrogate quadratic losses as an approximation of the real loss functions of previous tasks. Although they are simple and effective for a small number of tasks, their performances drop drastically as a larger number of tasks are involved. Memory aware synapses (MAS) [2] estimates the importance of the weights when a larger number of tasks are involved. Incremental moment matching (IMM) [20] additionally performs a separate model-merging step after learning a new task. Variational continual learning (VCL) [26] combines online variational inference [8, 29, 5] with Monte Carlo sampling [4] using neural networks, but it requires a relatively large amount of computational cost to infer an approximate posterior distribution.

On the other hand, [31] proposes a task-specific hard attention mechanism to achieve outstanding performance on a small number of tasks, but, as in [28, 6, 22], the requirement of multi-head outputs—a separate output per task—limits the number of target domains considered concurrently.

Instead of using architectural and functional techniques (or their combinations), our framework employs a structural regularization approach because it does not incur extra computational overhead such as additional network components and the need for occasional batch processing.
function is assumed to be symmetric as shown in Eq. (1) and fails to model its unobserved side accurately unless the true loss function is also symmetric.

However, it turns out that the true loss functions are typically asymmetric and the symmetric loss functions identified in SI are prone to underestimate the true losses in practice. Figure 2 illustrates how loss changes are represented in terms of each model parameter, where x-axis denotes the variation from the optimal model parameter and y-axis is the difference of cross-entropy losses, which are given by learning the first task of the permuted MNIST dataset. Therefore, for a more accurate and stable surrogate loss function parametrized by \( \hat{\Omega}_k^n \), we need to consider an asymmetric loss function formulation.

4. Proposed Algorithm

We propose a novel structure regularizer, ALASSO, which mitigates the limitations of SI. The main contributions of our algorithm include the introduction of the asymmetric loss function with single-side overestimation and the more accurate quadratic approximation of loss function, which lead to achieve remarkable performance improvements over various datasets. This section presents the details about ALASSO, especially in comparison to the existing approach, SI.

4.1. Overview

Our algorithm overestimates the unobserved sides of the approximate loss function and make models learn in harsher conditions. We also derive the optimal parameters of the quadratic loss functions for the approximation in their observed sides. To accelerate the optimization procedure and handle the conflicts between the loss computation of the current task and the loss approximation of the previous tasks, we introduce a hyper-parameter decoupling technique although the values of the decoupled parameters should be identical conceptually. The proposed algorithm inherits the merits of the standard structural regularization approaches such as online learning and local updates while providing the capability to maintain the crucial knowledge about the prior tasks by making the models adapt to harsher conditions. We now discuss our technical contribution in ALASSO and its characteristics in the continual learning scenario.

4.2. Asymmetric Loss Approximation

One possible option for a better structural regularizer in continual learning is asymmetric loss approximation of previous tasks. We believe that the symmetric regularizer as in SI overly simplifies the true loss function and have a critical limitation in maintaining the knowledge obtained from the previous tasks. Figure 3 demonstrates why asymmetric loss approximation is effective. The approximate loss function may be sufficiently accurate in the area, where the true loss is observable along the model parameter updates during training. However, it may incur substantial error in the unobserved side of the loss function, so it is dangerous to assume that the true loss function is symmetric, as also illustrated in Figure 2.

Based on this motivation, we now propose a simple but effective approximation approach for the true loss function. Basically, we believe that the coefficient, \( \hat{\Omega}_k^n \), is reliable if the learnable parameter \( \hat{\theta}_k^n \) and the fixed parameter \( \hat{\theta}_k^{n-1} \) are in the same side from \( \hat{\theta}_k^n \) because \( \hat{\theta}_k^n \) can observe the true loss function during optimization process. Conversely, if \( \hat{\theta}_k^n \) and \( \hat{\theta}_k^{n-1} \) are in the opposite sides from \( \hat{\theta}_k^n \), then \( \hat{\Omega}_k^n \) is unreliable. The details of the mapping on the graph are shown in Figure 4. To model the relationship between the variables, we introduce the following function:

\[
\alpha(\theta_k) \equiv \left( \hat{\theta}_k - \hat{\theta}_k^n \right) \left( \hat{\theta}_k^{n-1} - \hat{\theta}_k^n \right).
\] (4)

In our framework, if the true loss function cannot be observed, we overestimate the quadratic approximation parameter \( \hat{\Omega}_k^n \), which results in the definition of a new loss function. A generic asymmetric loss function in a quadratic form is given by

\[
\mathcal{L}_s^\alpha(\theta_k, \alpha) = \begin{cases} 
\hat{\Omega}_k^n \left( \theta_k - \hat{\theta}_k^n \right)^2 & \text{if } \alpha(\theta_k) > 0 \\
\left( a\hat{\Omega}_k^n + \epsilon \right) \left( \theta_k - \hat{\theta}_k^n \right)^2 & \text{if } \alpha(\theta_k) \leq 0 
\end{cases}
\] (5)

\( \alpha(>1) \) is a hyperparameter to control the degree of overestimation and \( \epsilon \) is a small number to make the loss overesti-
mated even when \( \hat{\Omega}^n_k = 0 \). Note that, since \( \hat{\Omega}^n_k \) is supposed to be positive, \( (a\hat{\Omega}^n_k + \epsilon) \) is always larger than \( \Omega^n_k \). From now, we omit the hyperparameter \( a \) in \( L^n_s(\theta_k, a) \) for notational simplicity, i.e., \( L^n_s(\theta_k) \equiv L^n_s(\theta_k) \).

The asymmetric loss function in Eq. (5) is used to define the total loss \( \hat{L}^n \), which is given by

\[
\hat{L}^n = L^n + cL^{n-1}_s
\]

(6)

\[
= L^n + c \sum_k L^{n-1}_s(\theta_k).
\]

Note that \( L^{n-1}_s \) can also be interpreted as a regularizer for the \( n^{th} \) task. One remaining concern is how to compute \( \hat{\Omega}^n_k \) in Eq. (5), which is discussed next.

4.3. Perfect Quadratic Approximation

In addition to the overestimation of the true loss in the unobserved side for a new task \( n \), our algorithm estimates the optimal coefficient \( \hat{\Omega}^n_k \) for the quadratic approximation of the loss function in its observed side. Note that, as presented in Eq. (5), \( \hat{\Omega}^n_k \) actually affects the approximation of the loss function in both sides; it is critical to derive \( \hat{\Omega}^n_k \) accurately for the performance of our algorithm.

The quadratic surrogate loss function in SI is determined by approximating its parameters \( \hat{\Omega}^n_k \) as in Eq. (3), which is derived from the Eq. (2). Instead of using the equation, we present the derivation that leads to the perfect quadratic approximation, which is given by

\[
\hat{\Omega}^n_k = \frac{L^n_s(\theta_k)}{L^n_s(\hat{\theta}^n_k - \hat{\theta}^n_k)^2} = \frac{L^n_s(\hat{\theta}^{n-1}_k)}{L^n_s(\hat{\theta}^{n-1}_k - \hat{\theta}^{n-1}_k)^2} = \frac{L^n_s(\hat{\theta}^{n-1}_k - \hat{\theta}^{n-1}_k)}{L^n_s(\hat{\theta}^{n-1}_k - \hat{\theta}^{n-1}_k)^2} = \omega^n_k + \omega^{1:(n-1)}_k.
\]

(7)

Note that \( \hat{\Omega}^n_k \) means the change of the surrogate loss \( L^n_s \) when model parameter changes from \( \hat{\theta}^{n-1}_k \) to \( \hat{\theta}^n_k \) in the process of learning the \( n^{th} \) task. The following properties and definitions are required to derive Eq. (7):

\[
L^n_s(\hat{\theta}^n_k) \propto (\hat{\theta}^n_k - \hat{\theta}^n_k)^2 = 0
\]

(8)

\[
L^n_s(\hat{\theta}^n_k) = L^n(\hat{\theta}^n_k) + cL^{n-1}_s(\hat{\theta}^n_k)
\]

(9)

and

\[
\omega^n_k \equiv L^n(\hat{\theta}^{n-1}_k) - L^n(\hat{\theta}^n_k)
\]

(10)

\[
\omega^{1:(n-1)}_k \equiv c(L^{n-1}_s(\hat{\theta}^{n-1}_k) - L^{n-1}_s(\hat{\theta}^n_k))
\]

(11)

Intuitively, the second equality in Eq. (7) is simply by plugging in \( \hat{\theta}^{n-1}_k(\theta_k) \) to \( L^n_s(\theta_k) \).

4.4. Parameter Decoupling

The overestimated loss approximation is effective to reduce errors that happen inevitably in the previous tasks, but optimization with this technique may suffer from slow convergence due to its inherent limitation. Specifically, some hyperparameters, such as \( a \) and \( c \), affect the objective function in one way or another depending on where they occur in our formulation. This is because we approximate the real loss for all the previous tasks using a single asymmetric function and it is practically impossible to consider all possible combinations of approximate functions estimated in the past.

In our formulation, there are two different occurrences of hyperparameters; one is related to the gradient computation in SGD and the other is in calculating \( \hat{\Omega}^n_k \) for loss approximation. For example, increasing \( c \) in Eq. (6) makes the model consider the previous tasks more while increasing \( c \) in Eq. (11) results in more weight on the current task by underestimating \( \hat{\Omega}^n_k \).

To handle the inconsistent impact of identical hyperparameters on the optimization process, we decouple the parameters into two sets; a set of parameters used for compu-
Figure 5. Test accuracy on the permuted MNIST dataset with 30 (left) and 100 (right) tasks. The results from several continual learning techniques including ours (ALASSO) are presented together with two upper-bound methods, single_task and multi_task. Note that x-axis denotes the index of a task given by the order of training. ALASSO achieves the state-of-the-art performance for both cases. Each method uses the hyper-parameters optimized for itself.

4.5. Discussion

Figure 4 illustrates the promising results of the proposed algorithm in comparison to SI, which is one of the state-of-the-art methods. The figure presents how the accuracy of each task that is learned earlier changes over time as new tasks are added one by one. We notice that the amount of degradation in ALASSO (ours) is much smaller than SI and the accuracy differences of the two algorithms in the same are getting larger as time goes by. These results clearly show the potential of our algorithm.

We claim that the proposed algorithm is practically good because it does not involve side effects such as architectural modification, network size increase, additional memory requirements, batch processing, multiple execution of networks, and inference on multi-head networks. We only need to store several variables such as $\Omega^n_k$, $\hat{\theta}^n_k$, and $\omega^n_k$ during training, and perform additional operations to compute the surrogate losses.

5. Experiments

This section demonstrates performance of our algorithm compared to existing approaches on the standard datasets. The code and trained model will be released.

5.1. Datasets and Algorithms

We use three standard benchmark datasets to evaluate our continual learning framework, which include the permuted MNIST, the split CIFAR-10/CIFAR-100 and the split Tiny ImageNet. The permuted MNIST is a synthetic dataset based on MNIST [18], where all pixels of an image in MNIST are permuted differently in each task but coherently in a single task. This dataset contains a large number of tasks and is widely used for evaluation of continual learning algorithms [34, 10, 16, 35]. The split CIFAR-10/CIFAR-100 dataset is generated from CIAFR-10 and CIFAR-100 [17] while the split Tiny ImageNet is derived from Tiny ImageNet [1]. These datasets divide all classes into multiple subsets, which correspond to individual tasks.

We compared our algorithm with the existing state-of-the-art methods including na"ıve stochastic gradient descent (SGD) [27, 15], SGD with dropout (SGD+dropout) [10], VCL [26], EWC [16], SI [35], IMM [20] and MAS [2]. To estimate the upper-bound performance of our algorithm, we present results from the models learned for individual tasks and all the tasks in a batch, which are denoted by single_task and multi_task, respectively.

5.2. Permuted MNIST

We present the results on 30 tasks and 100 tasks of the permuted MNIST dataset from all compared algorithms. For better visualization, we omitted results from SGD, SGD+dropout, IMM and MAS because their accuracy is significantly lower than the presented ones. We will show all the results in the supplementary document. We will also present more detailed configuration of all networks.
with additional ReLU and softmax output layers. Dropout is not used except for SGD+dropout.

According to our experiment shown in Figure 5, our algorithm achieves the best performance among all compared methods with large margins (about 15% point at least) with respect to the rest of approaches for both 30 and 100 tasks. Note that all other methods undergo large performance drops as the number of tasks increases. In contrast, our ALASSO is more robust than others; it presents only moderate performance loss even after learning 100 tasks.

To investigate the benefit of our main contribution—perfect estimation of a quadratic approximation parameter $\hat{\Omega}_n^k$ and asymmetric overestimation of the loss function in the unobserved side of the true loss, we perform without the perfect $\hat{\Omega}_n^k$ or asymmetric loss overestimation. These two ablation study results are also presented in Figure 5, which illustrates clear benefit of the two components.

Figure 6 illustrates that the results of our algorithm are stable in the sense that the performance variation across tasks are small, compared to all other methods, even though the learning process is incremental and we cannot access the data of the previous tasks$^2$. The overall accuracy of ALASSO is not affected by parameter decoupling as illustrated in Figure 7 while it is helpful to make the models converge fast. Note that parameter decoupling has been proposed to figure out the inconsistency impact of some hyperparameters on the performance.

### 5.3. Split CIFAR-10/CIFAR-100

We evaluate the performance of our algorithm in a more realistic scenario using the split CIFAR-10/CIFAR-100 dataset. In this experiment, we employ a CNN with 4

$$\text{VCL, SI, EWC, Single_task (upper bound), Multi_task (upper bound), ALASSO (ours)}$$

 convolutional layers followed by 2 fully connected layers with dropouts; ReLU and 2x2 max pooling are also used to add nonlinearity in the model. (Detailed configuration information can be found in the supplementary materials.) Figure 8 demonstrates the results from SI, single_task, and ALASSO on the split CIFAR-10/CIFAR-100 dataset. Our method outperforms SI consistently even on this more realistic dataset. It achieves 5.7% point better than SI in average while SI loses accuracy by 13.9% point; 41% of the performance drop of SI is recovered by ALASSO. Note that ALASSO even shows the comparable performance to single_task and multi_task methods in some tasks.

### 5.4. Split Tiny ImageNet

We performed an experiment with a larger CNN on the split Tiny ImageNet dataset, which has more classes. The
Figure 8. Accuracy at every 5 task on split CIFAR-10/CIFAR-100 with a CNN after training 30 tasks in an online manner. ALASSO works well compared to SI even in this more realistic dataset.

Figure 9. Per-task accuracy of every 5 tasks on split Tiny ImageNet with larger CNNs after training 30 tasks in an online manner. Even in this more realistic dataset with more classes, ALASSO achieves the average validation accuracy comparable to single_task.

The primary hyperparameter in our algorithm is $a$, which is introduced to overestimate the unobserved side of the loss function. Since $a$ determines the factor of overestimation, it is reasonable to set its value to a moderately larger number than 1. We determine these values empirically in the permuted MNIST dataset; we choose the similar values for the other datasets, and they are fixed within each dataset. According to our experience, the overall performance is not sensitive to the wide range of the value as shown in Table 1 if $a$ is 1, loss function is approximated by symmetric function. We make additional observation that the performance of our algorithm is degraded with $a < 1$. Another hyperparameter $c$ balances between the losses in the current and the past task, and it is set to 1 throughout the experiments.

### 5.5. Robustness

<table>
<thead>
<tr>
<th>$a$</th>
<th>0.8</th>
<th>1.0</th>
<th>2.0</th>
<th>3.0</th>
<th>4.0</th>
<th>5.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>30 tasks</td>
<td>0.91</td>
<td>0.92</td>
<td>0.94</td>
<td>0.94</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>100 tasks</td>
<td>0.59</td>
<td>0.62</td>
<td>0.79</td>
<td>0.79</td>
<td>0.78</td>
<td>0.75</td>
</tr>
</tbody>
</table>

Table 1. Sensitivity to $a$ on permuted MNIST

The CNN has 5 convolutional layers followed by 2 fully connected layers with dropout while 2x2 max pooling and ReLU activation function are employed as well to add nonlinearity of the network. (Detailed configuration information can be found in the supplementary materials.)

Figure 9 illustrates the results of SI, ALASSO and single_task (an upper bound). Note that the overall accuracy of ALASSO is as competitive as single_task, which should be the upper-bound performance conceptually. In reality, the average validation performance of our method is 59.4% and even higher than 58.9% of single_task accuracy. ALASSO also outperforms SI by $\sim 10\%$ point on average.

### 6. Conclusion

We presented a novel continual learning framework based on overestimated asymmetric loss approximation with the better parametrization for the quadratic approximation, which is a carefully designed generalized version of SI. Our algorithm alleviates the catastrophic forgetting issue, which is common in deep neural networks, and is particularly helpful for the scenario with a large number of tasks. The proposed solution is motivated by physical observation that network parameter updates does not affect target loss function symmetrically, and does not involve substantial side effects. It achieves the state-of-the-art performance on several challenging standard benchmark datasets.

### References


SUPPLEMENTARY MATERIAL

This document discusses intuitive interpretation of $L^n(\theta_k)$ in Section G. Section H presents the derivation of $\hat{\Omega}_n^k$ which is quadratic approximation parameter in SI [35] and ALASSO(ours). Section I presents the results on 30 tasks of permuted MNIST dataset from all state-of-the-art comparable algorithms. Section J provides the detailed configuration of the experimental network architectures.

G. Additional comment of $L^n(\theta_k)$

This section presents loss for the current task $L^n(\theta_k)$, which is a component dependent on the parameter $\theta_k$ in loss $L^n(\theta)$. The loss $L^n(\theta)$ can be expressed by the sum of $L^n(\theta_k)$:

$$L^n(\theta) = \sum_k L^n(\theta_k).$$  \hspace{1cm} (A.12)

$L^n(\theta_k)$ can be interpreted as the parameter specific contribution to the total loss $L^n(\theta)$.

H. Derivation of $\hat{\Omega}_n^k$ in Eq. (3) and (7) in the main paper

Assuming that the surrogate loss function for each parameter up to the $n^{th}$ task, $L^{n}_{s}(\theta_k)$, is defined by a quadratic function, which is further decomposed of the two terms as

$$L^{n}_{s}(\theta_k) = \hat{\Omega}_n^k (\theta_k - \hat{\theta}_n^k)^2 = L^n(\theta_k) + cL^{n-1}_{s}(\theta_k),$$  \hspace{1cm} (A.13)

where $L^n(\theta_k)$ is the loss for the current task, $L^{n-1}_{s}(\theta_k)$ is the surrogate loss function up to $(n - 1)^{th}$ task, and $c$ is a hyperparameter to balance between the two terms.

After completing learning of the $n^{th}$ task, we can further assume that we already have $n^{th}$ loss function $L^n(\theta_k)$ and the new model parameter $\hat{\theta}_n^k$ given $\hat{\Omega}^{n-1}_k$ and $\hat{\theta}^{n-1}_k$ from the previous iteration. Then, we can derive the value of $\hat{\Omega}_n^k$ satisfying Eq. (A.13) in SI and ALASSO based on the following procedures.
H.1. Synaptic Intelligence [35]

Eq. (A.14) shows that the $\hat{\Omega}_n^k$ obtained by the SI [35] method is not accurate approximation for quadratic surrogate loss function.

\[
\hat{\Omega}_n^k = \frac{\mathcal{L}^n (\hat{\theta}_k) + c \mathcal{L}_{n-1}^n (\hat{\theta}_k)}{(\hat{\theta}_k - \hat{\theta}_n^{k-1})^2} \quad \text{from Eq. (A.13)} \\
= \frac{\mathcal{L}^n (\hat{\theta}_k) + c \hat{\Omega}_{n-1}^k (\theta_k - \hat{\theta}_n^{k-1})^2}{(\hat{\theta}_k - \hat{\theta}_n^{k-1})^2}
\]

by plugging in $\hat{\theta}_n^{k-1}$

\[
= \frac{\mathcal{L}^n (\hat{\theta}_n^{k-1}) - \mathcal{L}^n (\hat{\theta}_n^k)}{(\hat{\theta}_n^{k-1} - \hat{\theta}_n^k)^2} + \frac{\mathcal{L}^n (\hat{\theta}_n^k)}{(\hat{\theta}_n^{k-1} - \hat{\theta}_n^k)^2} \\
= \frac{\mathcal{L}^n (\hat{\theta}_n^{k-1}) - \mathcal{L}^n (\hat{\theta}_n^k)}{(\hat{\theta}_n^{k-1} - \hat{\theta}_n^k)^2} + \frac{\mathcal{L}^n (\hat{\theta}_n^k) \cdot \hat{\Omega}_{n-1}^k}{(\hat{\theta}_n^{k-1} - \hat{\theta}_n^k)^2} \\
= \frac{\mathcal{L}^n (\hat{\theta}_n^{k-1}) - \mathcal{L}^n (\hat{\theta}_n^k)}{(\hat{\theta}_n^{k-1} - \hat{\theta}_n^k)^2} + \frac{\mathcal{L}^n (\hat{\theta}_n^k) \cdot \hat{\Omega}_{n-1}^k}{\mathcal{L}_{n-1}^n (\hat{\theta}_n^k)} \\
= \frac{\omega_n^k}{(\hat{\theta}_n^{k-1} - \hat{\theta}_n^k)^2} + \frac{\mathcal{L}^n (\hat{\theta}_n^k) \cdot \hat{\Omega}_{n-1}^k}{\mathcal{L}_{n-1}^n (\hat{\theta}_n^k)}
\]

by introducing a new variable $\omega_n^k$

\[
\approx \frac{\omega_n^k}{(\hat{\theta}_n^{k-1} - \hat{\theta}_n^k)^2} + \hat{\Omega}_{n-1}^k, \quad \text{(A.14)}
\]

where $\omega_n^k$ denotes the difference between the losses for the task $n$ before and after training the task, $\omega_n^k = \mathcal{L}^n (\hat{\theta}_n^{k-1}) - \mathcal{L}^n (\hat{\theta}_n^k)$
H.2. ALASSO

Eq. (A.15) shows that the \( \hat{\Omega}^n_k \) obtained by ALASSO provides the perfect quadratic approximation.

\[
\hat{\Omega}^n_k = \frac{L^n (\theta_k)}{(\theta_k - \hat{\theta}^n_k)^2}
\]

from Eq. (A.13)

\[
= \frac{L^n (\hat{\theta}^{n-1}_k)}{\left( \hat{\theta}^{n-1}_k - \hat{\theta}^n_k \right)^2}
\]

by plugging in \( \hat{\theta}^{n-1}_k \)

\[
= \frac{L^n (\hat{\theta}^{n-1}_k) - \hat{\Omega}^n_k (\hat{\theta}^n_k - \hat{\theta}^n_k)^2}{\left( \hat{\theta}^{n-1}_k - \hat{\theta}^n_k \right)^2}
\]

\[
= \frac{L_n (\hat{\theta}^{n-1}_k) - L_n (\hat{\theta}^n_k)}{\left( \hat{\theta}^{n-1}_k - \hat{\theta}^n_k \right)^2}
\]

\[
= \frac{\left( L^n (\hat{\theta}^{n-1}_k) + c L^{n-1} (\hat{\theta}^{n-1}_k) \right) - \left( L^n (\hat{\theta}^n_k) + c L^{n-1} (\hat{\theta}^n_k) \right)}{\left( \hat{\theta}^{n-1}_k - \hat{\theta}^n_k \right)^2}
\]

from Eq. (A.13)

\[
= \frac{\left( L^n (\hat{\theta}^{n-1}_k) - L^n (\hat{\theta}^n_k) \right) + c \left( L^{n-1} (\hat{\theta}^{n-1}_k) - L^{n-1} (\hat{\theta}^n_k) \right)}{\left( \hat{\theta}^{n-1}_k - \hat{\theta}^n_k \right)^2}
\]

by introducing a new variable \( \omega^n_k, \omega^{1:(n-1)}_k \)

(A.15)

where \( \omega^n_k \) denotes the difference between the losses for the task \( n \) before and after training the task, \( \omega^n_k = L^n (\hat{\theta}^{n-1}_k) - L^n (\hat{\theta}^n_k) \). \( \omega^{1:(n-1)}_k \) denotes the difference between the surrogate losses of previous tasks for the task \( n \) before and after training the task, \( \omega^{1:(n-1)}_k = c \left( L^{n-1} (\hat{\theta}^{n-1}_k) - L^{n-1} (\hat{\theta}^n_k) \right) \).
I. Additional comparison with other methods

In this subsection we show the comparative experimental results with more algorithms on the permuted MNIST 30 tasks. The additionally compared with the existing state-of-the-art algorithms include SI [35], EWC [16], VCL [26], MAS [2], IMM [20], naïve stochastic gradient descent (SGD) [27, 15], SGD with dropout (SGD+dropout) [10]. Figure J and Figure K show the accuracy and standard deviation of each algorithm over the number of tasks. The Multi task as another upper bound means a simple multi-task learning with a mini-batch mixing data of all the previous tasks and the current task. According to experiment in Figure J our algorithm achieves the best performance among all compared methods with large margins (about 15% point at least). Besides, Figure K shows that our algorithm is stable in the sense that the performance variation across tasks are small, compared to all other methods, even though the learning process is incremental.

![Figure J. Total test accuracy of all previously and currently learned tasks on permuted MNIST (one of the most commonly used benchmarks) over time with different continual learning methods (including our method, ALASSO), and single task and multi task (as upper bounds) (x-axis: the index of the lastly trained task. Our method achieves state-of-the-art performance for 30 tasks (even with a near-upper-bound result)](image1)

![Figure K. Standard deviation of per-task test accuracy of all previously and currently learned tasks on permuted MNIST over time with different continual learning methods (including our method, ALASSO) and single task and multi task (as upper bounds) (x-axis: the index of the lastly trained task. Our method achieves near-0 standard deviation similar to the upper bound)](image2)
**J. Configuration of the experimental architectures**

The detailed network architecture configuration information for our experiments on the Permuted MNIST (Table. B) / Split CIFAR-10, CIFAR-100 (Table. C) / Tiny ImageNet (Table. D) are given below.

**Table B. Network Architecture for Permuted MNIST Experiment**

<table>
<thead>
<tr>
<th>Layer Type</th>
<th>Layer Size(or value)</th>
<th>input size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense + ReLU</td>
<td>2000</td>
<td>1x1x784</td>
</tr>
<tr>
<td>Dense + ReLU</td>
<td>2000</td>
<td>1x1x2000</td>
</tr>
<tr>
<td>Dense</td>
<td>10</td>
<td>1x1x2000</td>
</tr>
<tr>
<td>Softmax</td>
<td>-</td>
<td>1x1x10</td>
</tr>
</tbody>
</table>

**Table C. Network Architecture for Split CIFAR-10/CIFAR-100 Experiment**

<table>
<thead>
<tr>
<th>Layer Type</th>
<th>Layer Size(or value)</th>
<th>input size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv + ReLU</td>
<td>3x3x4</td>
<td>32x32x3</td>
</tr>
<tr>
<td>Conv + ReLU</td>
<td>3x3x4</td>
<td>32x32x4</td>
</tr>
<tr>
<td>Max Pooling</td>
<td>2x2</td>
<td>32x32x4</td>
</tr>
<tr>
<td>Dropout</td>
<td>0.25</td>
<td>16x16x4</td>
</tr>
<tr>
<td>Conv + ReLU</td>
<td>3x3x8</td>
<td>16x16x4</td>
</tr>
<tr>
<td>Conv + ReLU</td>
<td>3x3x8</td>
<td>16x16x8</td>
</tr>
<tr>
<td>Max Pooling</td>
<td>2x2</td>
<td>16x16x8</td>
</tr>
<tr>
<td>Dropout</td>
<td>0.25</td>
<td>8x8x8</td>
</tr>
<tr>
<td>Dense + ReLU</td>
<td>64</td>
<td>1x1x512</td>
</tr>
<tr>
<td>Dropout</td>
<td>0.5</td>
<td>1x1x64</td>
</tr>
<tr>
<td>Dense</td>
<td>90</td>
<td>1x1x64</td>
</tr>
<tr>
<td>Softmax (Per-task)</td>
<td>-</td>
<td>1x1x90</td>
</tr>
</tbody>
</table>

**Table D. Network Architecture for Tiny ImageNet Experiment**

<table>
<thead>
<tr>
<th>Layer Type</th>
<th>Layer Size(or value)</th>
<th>input size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv + ReLU</td>
<td>3x3x32</td>
<td>224x224x3</td>
</tr>
<tr>
<td>Max Pooling</td>
<td>2x2</td>
<td>224x224x32</td>
</tr>
<tr>
<td>Dropout</td>
<td>0.25</td>
<td>112x112x32</td>
</tr>
<tr>
<td>Conv + ReLU</td>
<td>3x3x32</td>
<td>112x112x32</td>
</tr>
<tr>
<td>Max Pooling</td>
<td>2x2</td>
<td>112x112x32</td>
</tr>
<tr>
<td>Dropout</td>
<td>0.25</td>
<td>56x56x32</td>
</tr>
<tr>
<td>Conv + ReLU</td>
<td>3x3x64</td>
<td>56x56x32</td>
</tr>
<tr>
<td>Max Pooling</td>
<td>2x2</td>
<td>56x56x64</td>
</tr>
<tr>
<td>Dropout</td>
<td>0.25</td>
<td>28x28x64</td>
</tr>
<tr>
<td>Conv + ReLU</td>
<td>3x3x64</td>
<td>28x28x64</td>
</tr>
<tr>
<td>Max Pooling</td>
<td>2x2</td>
<td>28x28x64</td>
</tr>
<tr>
<td>Dropout</td>
<td>0.25</td>
<td>14x14x64</td>
</tr>
<tr>
<td>Conv + ReLU</td>
<td>3x3x64</td>
<td>14x14x64</td>
</tr>
<tr>
<td>Max Pooling</td>
<td>2x2</td>
<td>14x14x64</td>
</tr>
<tr>
<td>Dropout</td>
<td>0.25</td>
<td>7x7x64</td>
</tr>
<tr>
<td>Dense + ReLU</td>
<td>2048</td>
<td>1x1x3136</td>
</tr>
<tr>
<td>Dropout</td>
<td>0.5</td>
<td>1x1x2048</td>
</tr>
<tr>
<td>Dense</td>
<td>180</td>
<td>1x1x2048</td>
</tr>
<tr>
<td>Softmax (Per-task)</td>
<td>-</td>
<td>1x1x180</td>
</tr>
</tbody>
</table>