Enabling Continual Learning in Neural Networks with Meta Learning

Anonymous Authors[1]

Abstract

Catastrophic forgetting in neural networks is one of the most well-known problems in continual learning. Previous attempts on addressing the problem focus on preventing important weights from changing (Kirkpatrick et al., 2017; Zenke et al., 2017). Such methods often require task boundaries to learn effectively and do not support backward transfer learning. In this paper, we propose a meta-learning algorithm which learns to reconstruct the gradients of old tasks w.r.t. the current parameters and combines these reconstructed gradients with the current gradient to enable continual learning and backward transfer learning from the current task to previous tasks. Experiments on standard continual learning benchmarks show that our algorithm can effectively prevent catastrophic forgetting and supports backward transfer learning.

1. Introduction

The ability to learn continually without forgetting previously learned skills is crucial to artificial general intelligence (AGI) (Legg & Hutter, 2007). Addressing catastrophic forgetting in artificial neural networks (ANNs) has been the top priority of continual learning research. Notable attempts on solving the problem include Elastic Weight Consolidation (EWC) by Kirkpatrick et al. (2017) and the follow up work on Synaptic Intelligence (SI) by Zenke et al. (2017), and Memory Aware Synapse (MAS) by Aljundi et al. (2018). These algorithms share the same core idea: preventing important parameters from deviating from their old (presumably better) values. In order to achieve that, EWC-like algorithms compute the importance of each parameter w.r.t. each task in the sequence and for each old task, a regularization term is added to the loss of the new task to prevent that task from being catastrophically forgotten. The regularization term for task $\mathcal{T}^{(i)}$ in EWC-like algorithms takes the following form:

$$\frac{\lambda^{(i)}}{2}\sum_j \omega_j^{(i)}(\theta_j - \theta_j^{(i)*})^2 \tag{1}$$

where $\lambda^{(i)}$ controls the relative importance of task i to the current task, $\boldsymbol{\theta}$ is the current parameters, $\boldsymbol{\theta}^{(i)*}$ is the parameters found at the end of the training of $\mathcal{T}^{(i)}$, and $\omega_j^{(i)}$ is the importance of parameter $\theta_j^{(i)*}$ w.r.t. $\mathcal{T}^{(i)}$. Intuitively, the regularizer prevents θ_j with large $\omega_j^{(i)}$ from deviating too far from $\theta_j^{(i)*}$ while allowing θ_j with small $\omega_j^{(i)}$ to change more freely.

EWC-like algorithms have several weaknesses:

1. The regularizer in Eqn. 1 prevent changes to important parameters regardless of the effect of these changes. Unless $\theta_j^{(i)*}$ is the optimal value for the j-th parameter, either increasing or decreasing its value will result in better performance on task i. Keeping $\boldsymbol{\theta}$ close to $\boldsymbol{\theta}^{(i)*}$ only prevent the network from catastrophically forgetting $\mathcal{T}^{(i)}$ but cannot help the network to leverage the information from the current task $\mathcal{T}^{(k)}, k > i$ to improve its performance on $\mathcal{T}^{(i)}$ and other previous tasks. In other words, regularizers of the form in Eqn. 1 do not support backward transfer learning.
2. The number of old parameter and importance vectors, $\boldsymbol{\theta}^*$ and $\boldsymbol{\omega}$, grows linearly with the number of tasks, making EWC-like algorithms not scalable to a large number of tasks. Schwarz et al. (2018) proposed the online EWC algorithm which maintains only one copy of $\boldsymbol{\theta}^*$ and $\boldsymbol{\omega}$. The sizes of $\boldsymbol{\theta}^*$ and $\boldsymbol{\omega}$ are equal to that of the network. Therefore, the memory requirement of online EWC is still considerably large for large networks.

To address these limitations of EWC-like algorithms, we propose a meta learning algorithm which:

1. Learns to approximate the gradient of a task w.r.t. the current parameters from the current parameters
2. Combines the approximated gradients of old tasks w.r.t. the current parameters and the current task's gradient

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

to result in an update that improves the performance of the network on all tasks.

By combining the gradients, our algorithm exploits the similarity between the current task and previous tasks to enable backward transfer learning. As described in section 2.2 and 5.2, the size of a meta-network is typically orders of magnitude smaller than that of the main network and meta-networks for different tasks can be distilled into a single meta-network in an online manner. That significantly reduces the memory requirement of our method.

In the next section, we introduce our learning to learn algorithm for continual learning. Experiments are presented in section 3. Conclusions and future work are located in section 4 and 5, respectively.

2. Method

2.1. Motivation

Let us consider a continual learning problem with a learner $f(\boldsymbol{x};\boldsymbol{\theta}) : \mathbb{R}^D \rightarrow \mathbb{R}^d$, T tasks $\left\{\mathcal{T}^{(i)}\right\}_{i=1}^{T}$ with T corresponding training datasets $\left\{\mathcal{D}^{(i)}\right\}_{i=1}^{T}$, $\mathcal{D}^{(i)} = \left\{\left(\boldsymbol{x}_j^{(i)}, \boldsymbol{y}_j^{(i)}\right)\right\}_{j=1}^{n^{(i)}}$, and T loss functions $\left\{\mathcal{L}^{(i)}\right\}_{i=1}^{T}$, $\mathcal{L}^{(i)}\left(\hat{\boldsymbol{y}}^{(i)}, \boldsymbol{y}^{(i)}\right) : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$. To avoid clutter, we remove the input of the loss function $\mathcal{L}$, and $\nabla^{(i)}$ and $\nabla_{\boldsymbol{\theta}}\mathcal{L}^{(i)}$ are short for $\nabla_{\boldsymbol{\theta}}\mathcal{L}^{(i)}\left(\left\{\hat{\boldsymbol{y}}_j^{(i)}\right\}_{j=1}^{n^{(i)}}, \left\{\boldsymbol{y}_j^{(i)}\right\}_{j=1}^{n^{(i)}}\right)$ [1].

In joint learning settings, data from all tasks is available to the learner. The parameter $\boldsymbol{\theta}$ is updated using the average of gradients from all tasks:

$$\boldsymbol{\theta}' = \boldsymbol{\theta} - \alpha\boldsymbol{\delta},$$

where α is the learning rate, and

$$\boldsymbol{\delta} = \frac{1}{T}\nabla_{\boldsymbol{\theta}}\sum_{i=1}^{T}\mathcal{L}^{(i)} = \frac{1}{T}\sum_{i=1}^{T}\nabla_{\boldsymbol{\theta}}\mathcal{L}^{(i)} \tag{2}$$

Generally, updating $\boldsymbol{\theta}$ with $\boldsymbol{\delta}$ will improve the performance of f on all T tasks.

In continual learning settings, at task $t+1$, the learner cannot access to $\mathcal{D}^{(i)}, i = 1, ..., t$ and cannot compute $\nabla_{\boldsymbol{\theta}}\mathcal{L}^{(i)}, i = 1, ..., t$. The update at task $t+1$ is computed from $\nabla_{\boldsymbol{\theta}}\mathcal{L}^{(t+1)}$ only. When $\boldsymbol{\theta}$ is updated with this gradient, f's performance on $\mathcal{T}^{(t+1)}$ will be improved while f's performance on tasks $\mathcal{T}^{(i)}, i = 1, ..., t$ might be catastrophically damaged.

[1] The analysis here still applies to the case where mini-batches are used because the expectation of $\mathcal{L}^{(i)}\left(\hat{\boldsymbol{y}}^{(i)}, \boldsymbol{y}^{(i)}\right)$ converges to $\mathcal{L}^{(i)}\left(\left\{\hat{\boldsymbol{y}}_j^{(i)}\right\}_{j=1}^{n^{(i)}}, \left\{\boldsymbol{y}_j^{(i)}\right\}_{j=1}^{n^{(i)}}\right)$.

To address this problem, we propose the following meta learning algorithm. During task i, we train meta-network $h^{(i)}$ to reconstruct $\nabla_{\boldsymbol{\theta}}\mathcal{L}^{(i)}$ from $\boldsymbol{\theta}$. In subsequent tasks, $h^{(i)}$ is used to reconstruct the gradient of task i w.r.t. the current parameters without having to access to $\mathcal{D}^{(i)}$. More concretely, $h^{(i)}$ learns to map the parameter to the corresponding gradient:

$$h^{(i)}(\boldsymbol{\theta};\boldsymbol{\phi}^{(i)}) \approx \nabla_{\boldsymbol{\theta}}\mathcal{L}^{(i)} \tag{3}$$

When the main network f is trained on a new task $\mathcal{T}^{(k)}, k > i$, $h^{(i)}$ is used to produce $\hat{\nabla}_{\boldsymbol{\theta}}\mathcal{L}^{(i)} = h^{(i)}(\boldsymbol{\theta};\boldsymbol{\phi}^{(i)})$, an approximate of $\nabla_{\boldsymbol{\theta}}\mathcal{L}^{(i)}$. $\hat{\nabla}_{\boldsymbol{\theta}}\mathcal{L}^{(i)}$ is used in the place of $\nabla_{\boldsymbol{\theta}}\mathcal{L}^{(i)}$ in Eqn. 2.

Section 2.3 introduces several ways to combine predicted gradients with the current gradient to prevent catastrophic forgetting and enable backward transfer learning. For our method to work when optimizers other than SGD is used to train the main network, $\nabla_{\boldsymbol{\theta}}\mathcal{L}^{(i)}$ should be replaced with the update vector produced by the optimizer.

2.2. Learning to predict gradients

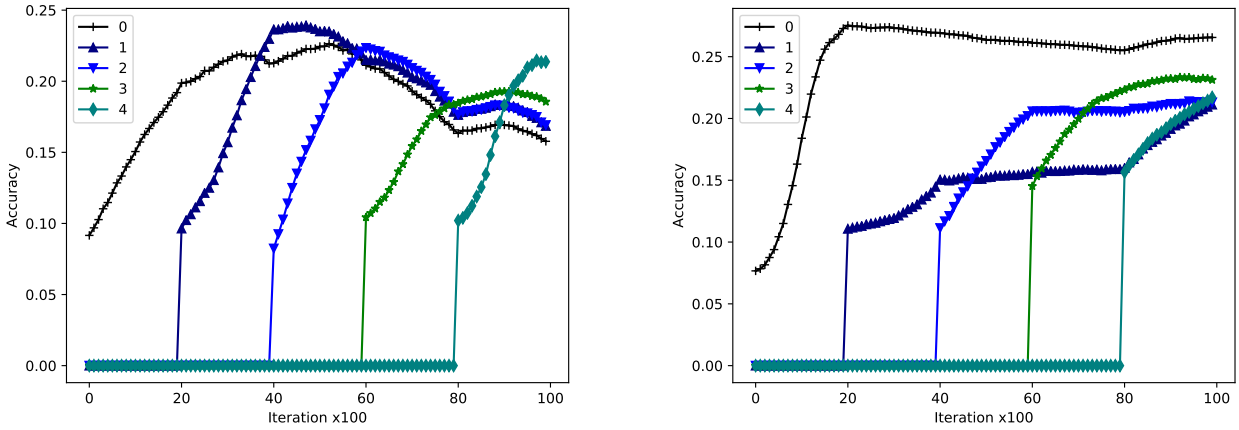
Because a real world neural network typically contains tens of thousands to billions of parameters, the naive way of training h would require an astronomically large number of samples of $\boldsymbol{\theta}$ and ∇ to cover a very high dimensional space. A fully connected meta-network h also need to be extremely large to receive a very high dimensional input and produce a very high dimensional output. To circumvent the problem, we follow the coordinate-wise approach proposed by Andrychowicz et al. (2016) where each coordinate is processed independently. h is a neural network that takes in a 1-dimensional input and produces 1-dimensional output

$$h(\theta_j;\boldsymbol{\phi}) \approx \nabla_j \tag{4}$$

The procedure is applied to all coordinates in $\boldsymbol{\theta}$. In our experiments, hs are MLPs and are trained to minimize the Euclidean distance between $h(\theta_j;\boldsymbol{\phi})$ and ∇_j for all θ_j in $\boldsymbol{\theta}$. h could be modified to process more inputs such as the position of parameter in the network or the previous values of θ_j. It is also possible for h to process a small set of related parameters simultaneously, e.g. parameters in the same filter of a CNN. However, we leave these variations for future work.

2.3. Preventing catastrophic forgetting with predicted gradients

Let us consider a pair of gradients $\nabla^{(k)}$ and $\nabla^{(i)}$. If $\nabla_j^{(k)}$ and $\nabla_j^{(i)}$ have different signs and α is small enough, then updating f with $\nabla_j^{(k)}$ will improve the network's performance on task k and damage its performance on task i. If they have the same sign then the update will improve the performance



(a) (b)

Figure 1. Accuracy of a 3 hidden layer MLP with 784 inputs, 10 outputs, and 256 hidden neurons on 5 tasks of the Permuted MNIST dataset. Each task contains 2000 iterations. The batch size is 64, the learning rate is $\alpha = 0.003$ for both experiments. (a) The accuracy when plain SGD is applied. (b) The accuracy when our algorithm is applied. The gradient predictors are 3 hidden layer MLPs with 1 input, 1 output, and 256 hidden neurons.

on both tasks. That intuition leads to the following rule to create an update vector from a pair of gradients:

$$\delta_j = \begin{cases} 0, \text{ if } \nabla_j^{(k)} \cdot \nabla_j^{(i)} \leq 0 \\ \nabla_j^{(k)}, \text{ if } \nabla_j^{(k)} \cdot \nabla_j^{(i)} > 0 \end{cases} \tag{5}$$

At task $t+1$, an update vector $\boldsymbol{\delta}$ can be produced by applying the above rule the pair between $\nabla^{(t+1)}$ and all other gradients $\hat{\nabla}^{(i)}, i = 1, ..., t$. When t is large, that method usually results in a sparse update vector. In practice, we apply the rule to the pair $\left\{\nabla^{(t+1)}, \bar{\nabla}^{1:t}\right\}$ where $\bar{\nabla}^{1:t} = \frac{1}{t}\sum_{i=1}^{t}\hat{\nabla}^{(i)}$. Updating the main network with $\boldsymbol{\delta}$ will improve the performance on task $t+1$ and will likely to improve the performance on tasks $1, ..., t$.

The update vector $\boldsymbol{\delta}$ contains information that are common between task $t+1$ and previous tasks. Updating f with $\boldsymbol{\delta}$ transfers information from the current task to previous tasks. $\boldsymbol{\delta}$ is the medium for backward transfer learning in our algorithm.

3. Experiments

We tested our algorithm on the Permuted MNIST dataset (Kirkpatrick et al., 2017). To better demonstrate the effect of backward transfer learning, we train each task for only 2000 iterations to prevent the main network from reaching its maximum performance. The result is shown in Fig. 1.

The network in Fig. 1(a) suffers from catastrophic forgetting problem: the performance on old tasks decrease rapidly when new tasks are trained. The network trained with our algorithm (Fig. 1(b)) does not suffer from catastrophic forgetting: the performance on old tasks is maintained or even improved when new tasks are trained. The performance improvement on old tasks suggests that our algorithm has backward transfer learning capability. We also note the forward transfer learning phenomenon in Fig. 1(b): the starting accuracy of a later task is higher than that of former ones.

4. Conclusions

In this paper, we present a meta learning algorithm for continual learning. Experiments on Permuted MNIST dataset show that our algorithm is effective in preventing catastrophic forgetting and is capable of supporting backward transfer learning.

5. Future work

5.1. Learning to learn continually without task boundaries

To make our algorithm works without task boundaries, we need to detect boundaries automatically. The simplest way to detect task boundaries is to look at the loss of the main network. That way, however, does not work well when different tasks uses different loss functions or have different input, output scales. We propose to use detect task boundaries using the loss of the meta-networks. Different tasks

must have different gradient patterns [2], the change in task will result in higher error in gradient reconstruction and can be detected. When a new task is detected, a new meta-network is created to learn the gradient pattern of that task. The next section present a method for distilling knowledge from all previous meta-networks into a single one.

5.2. Distilling knowledge from meta-networks

As presented in section 2.3, we only need $\bar{\nabla}^{1:t}$ and $\nabla^{(t+1)}$ to produce the update for task $t+1$, other gradients $\hat{\nabla}^{(i)}, i = 1, ..., t$ are not needed for computing $\boldsymbol{\delta}$. We can thus reduce the number of meta-networks by creating a meta network $g(\boldsymbol{\theta}; \boldsymbol{\mu})$ which approximates $\bar{\nabla}^{1:t}$. The knowledge from $h^{(i)}, i = 1, ..., t$ is distilled into g. Because

$$\bar{\nabla}^{1:t+1} = \frac{t}{t+1}\bar{\nabla}^{1:t} + \frac{1}{t+1}\hat{\nabla}^{(t+1)} \quad (6)$$

$$\approx \frac{t}{t+1}g(\boldsymbol{\theta}; \boldsymbol{\mu}) + \frac{1}{t+1}\hat{\nabla}^{(t+1)} \quad (7)$$

g can be trained to approximate $\bar{\nabla}^{1:t+1}$ in an online manner: at task 1, g is initialized to be $h^{(1)}$; at task $t+1$, g is trained to approximate $\bar{\nabla}^{1:t+1}$ using the above equations. At any task $t+1$, we only need to keep g and $h^{(t+1)}$.

References

Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M., and Tuytelaars, T. Memory aware synapses: Learning what (not) to forget. In Ferrari, V., Hebert, M., Sminchisescu, C., and Weiss, Y. (eds.), *Computer Vision – ECCV 2018*, pp. 144–161, Cham, 2018. Springer International Publishing. ISBN 978-3-030-01219-9.

Andrychowicz, M., Denil, M., Gómez, S., Hoffman, M. W., Pfau, D., Schaul, T., and de Freitas, N. Learning to learn by gradient descent by gradient descent. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 3981–3989. Curran Associates, Inc., 2016.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017. ISSN 0027-8424. doi: 10.1073/pnas.1611835114.

Legg, S. and Hutter, M. Universal intelligence: A definition of machine intelligence. *CoRR*, abs/0712.3329, 2007. URL http://arxiv.org/abs/0712.3329.

[2] If two tasks have similar gradient patterns then they are similar and can be considered as a single task.

Schwarz, J., Czarnecki, W., Luketina, J., Grabska-Barwinska, A., Teh, Y. W., Pascanu, R., and Hadsell, R. Progress & compress: A scalable framework for continual learning. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4528–4537, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

Zenke, F., Poole, B., and Ganguli, S. Continual learning through synaptic intelligence. In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 3987–3995, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.