Overcoming Catastrophic Forgetting with Orthogonal Update in Continual Learning

Han Zhang
Computer Science, Zhiyuan colledge
Shanghai Jiao Tong University
520020910156
zhhhhahahaha@sjtu.edu.cn

Abstract

This paper proposes a method called **OU** (Orthogonal Updating) to overcome the catastrophic forgetting while the model is learning different task sequentially. The experiments of this paper is focus on MINST dataset using DNN model.

1 Introduction

Continual Learning(CL) is a concept to learn a model for a large number of tasks sequentially without forgetting knowledge obtained from the preceding tasks, where the data in the old tasks are not available any more during training new ones. The goal of CL is achieving two main objectives: *overcoming catastrophic forgetting* (CF) and *encourage knowledge transfer across tasks* (KT). The method OU only focus on overcoming CF in continual learning.

1.1 Mainstream Approach in Continual learning

Regularization Add a regularization in the loss function to consolidate the weights for previous task when learning new task in order to reduce the impact of learning new task on previous learned knowledge. The typical approch of regularization is EWC.

Ensembling When the model is learning new task, explicitly or implicitly add new model to learn the new task. Essentially it makes different model to handle different tasks and then ensembles all the model. This approach seems intuitive for learning different tasks sequentially, but it creates challenge to the learning efficiency and it may cost much to store the whole model. The typical approach of it is the **PathNet** released by google.

Replay The idea of this approch is recording the important data or the gradient of the previous learning process, and then replay these important data or the gradient when learning new task in order to overcome the catastrophic forgetting And typical algorithm MER add the product of the gradient of new task and the old task to the loss function in order to learn the feature that is beneficial to both tasks.

1.2 Different Type of Continual Learning

We can distinguish different type of continual learning based on the relation between different tasks.

input reformatting In this type, the goal of the different task is the same, but the format of the input is different. Take the MINST dataset as example, the old task is identify the MINST dataset, and the new task is disorganize the pixel of the input and then classify the data.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

similar tasks In this type, the goal of the different task is different, but similar. For example, the old task is classifying the MINST data of 0-5, and the new task is classifying the MINST data of 6-9.This is the most intuitive type of continual learning , and the research of this paper is based on this type.

dissimilar tasks In this type, the goal of different task have great difference. For example the old task is classification of the MINST dataset, but the new task is the audio classification.

2 Orthogonal Update

The idea of Orthogonal Update is that we record the input space, and then use the normal gradient descent method to calculate the update of the weights. But when we need to truly update the weights, we need first subtract the component of the calculated update in the direction of the input space, then the update left is orthogonal to the input space. Finally we achieve orthogonal update.

2.1 Mathematical Derivation

We assume that the input vector $\vec{x} \in R^m$, therefore the input space is in R^m
One set of orthogonal unit basis of the input space is $\{\vec{a_1}, \vec{a_2}, ..., \vec{a_n}\}$
The calculated update of the weights using normal gradient descent is ΔW_{origin}
The final update using orthogonal update is ΔW_{final}

$$\Delta W_{final} = \Delta W_{origin} - (\Delta W_{origin} \cdot \vec{a_1})\vec{a_1} - (\Delta W_{origin} \cdot \vec{a_2})\vec{a_2} - ... - (\Delta W_{origin} \cdot \vec{a_n})\vec{a_n}$$

2.2 Implementation Details

In actual implementation, we use a matrix $M \in R^{m \times n}$ to record the set of orthogonal unit basis of the input space.
And the weights $W \in R^{m \times k}$, k is the dimensions of the output vector

$$\Delta W_{final} = \Delta W_{origin} - MM^T \Delta W_{origin}$$

And when we are ready to learn new task, we need to update the matrix M using the input vector x' of the task we have just learned.

$$x'_{unit} = \frac{x' - MM^T x'}{||x' - MM^T x'||}$$

$$M_{new} = [M_{old}, x'_{unit}]$$

In this way, we don't need to calculate matrix everytime we face new input vector, we only need to append the x'_{unit} to the last colomun of the matrix in order to realize the update of the matrix M
But in the actual implementation, we usually have thousands of input vector, we cannot regard every input vector as a new input vector to update the matrix M becasue this will make the matrix M's dimensions to be very high and the high dimensions of M will constraint the learning of the new task to a large extent.
In the actual implementation, we take a batch of input vectors, and we calculate the mean value $\bar{x}$ of them, and we regard the $\bar{x}$ as a new input vectors to update the matrix M.
And we also need a hyper-parameter to determine how many batch we will take to update the matrix M after we have learned a new task.

3 Experiment

3.1 Dataset

We use the MINST dataset in this experiment. The MINST data is of handwritten digits. In the experiment, we spilt the dataset into 10 tasks, respectively are recognize 0 to 9, and we let the model learn these 10 tasks sequentially.

Table 1: Final Experiment Result

Module and Description	Accuracy
DNN with original MINST training set	$\sim$95%
DNN with sequential MINST training set	$\sim$40%
DNN using OU with sequential training set	$\sim$90%

3.2 Model Description

In the experiment we use 2-layer DNN model with [784-800-10] neurons. And we use the **relu** function as the activation function and on the output layer we use **softmax** function and cross-entropy as loss function.

3.3 Results of the Experiment

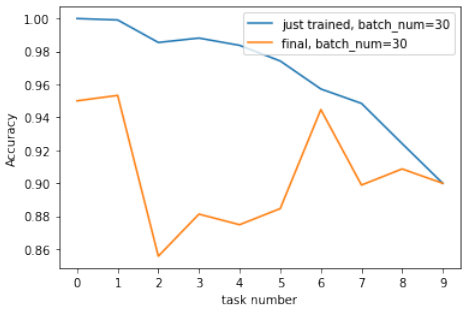
Final result In the experiment, we test the model's nearly the best performance in three conditions(table 1). First, we just use the DNN model and the input is the original MINST training set. The model realize a great performance on classify the handwritten digits, the accuracy reach 95%. But after we spilt the MINST dataset and let the model learn the classification of handwritten digits 0-9 sequentially, the performance of the model face a catastrophic drop. The accuracy of the model has only 40%, and this is nearly the best performance under this condition. But after we use the Orthogonal Update, the performance of the model improved greatly, the accuracy reach 90%

Just Trained Accuracy and Final Accuracy The just trained accuray on one task means the accuracy of the model's prediction after just learning this task, the final accuray on one task means the accuracy of the model's prediction after learning all the task, and we compare this two accuracy(figure 1 (a)). The idea of the Orthogonal Update is let the subsequent weights changing do not interfere the previous input, but in actual implementation we only select a few batch of the input vectors calculating the mean value to update the matrix M, so the subsequent weights changing will inevitably change the performance of the model on the previous input. In the figure we see that nearly all the tasks have decrease on final accuracy compared to just trained accuracy. This shows that we cannot achieve the ideal situation.

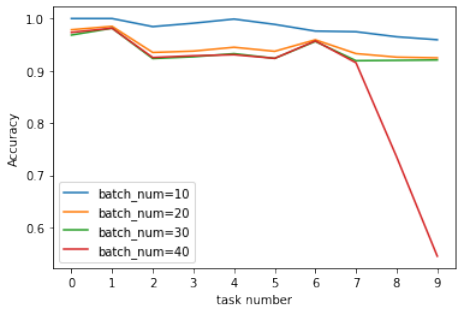
Performance of Orthogonal Update using different hyper-parameter As we mentioned before, Orthogonal Update need a hyper-parameter *batch_num* to determine how many batch we will take to update the matrix M after we have learned a new task. In the experiment we try different hyper-parameter and test the just trained accuracy of the model (figure 1 (b)). We can see that the overall accuracy of the just trained accuracy decreases as the batch number increases. This is in line with our expectations beacause as the batch number increase, the rank of the M increases, so it creates more constraints on learning new task. Due to the same reason, we can see from the figure that the just trained accuracy have downtrend as the task number increases.
We also test the final accuracy actually is the model's performance using different hyper-parameter (figure 1 (c)). We can see from the graph that the accuracy of the model slowly increase at first but decrease rapidly after that as the batch number increase. And this is also in line with our expectations. When the batch number is too small, the matrix M's rank is too small to maintain the weight trained by previous jobs, the performance of the model must be relatively low. And when the batch number is too big, the matrix M's rank is too big so that it creates great constraint to the subsequent tasks, the performance of the model must be relatively low either.

4 Conclusion

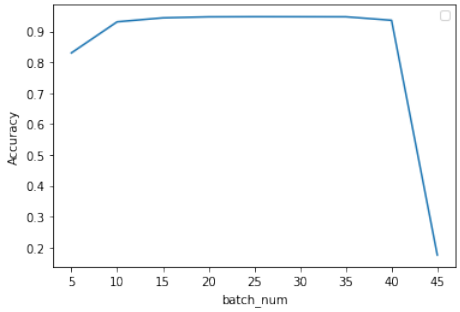
The experiment shows that Orthogonal Update can greatly overcome the catastrophic forgetting, and the only hyper-parameter *batch_num* will affect the performance of the model and they are not monotonic relationship, so every time we need to select the best *batch_num* in order to reach best performance.



(a) "just trained" means that the accuray of the model's predction about one task just after learning it. "final" means the the accuray of the model's prediction about one task after learning all the task



(b) The accuracy of this figure is the "just trained" accuracy, meaning the accuracy about the model's prediction about one task after just learning it



(c) The accuracy of this figure is the "final" accuracy, meaning the accuracy about the model's prediction about one task after learning all the tasks

Figure 1: Experiment Result

Weakness of Orthogonal Update The weakness of the Orthogonal Update is that when the input vector's dimensions is determined, the rank of the matrix M is at most the dimensions of the input vector. So as the task number increase, the subsequent task have more and more constraints, they are more and more hard to be learned. Take the MINST dataset as example, the input vector is 785 dimensions, and each batch used to update the matrix M at least increase one rank of the matrix M, so we at most use 785 batch to update matrix M. And in actual implementation, as the rank of the matrix M increase to some extent, the performance of the model will drop rapidly, so we cannot jut use Orthogonal Update to learn a large number of tasks.

References

[1] Zixuan Ke, Bing Liu, Nianzu Ma, Hu Xu and Lei Shu. Achieving Forgetting Prevention and Knowledge Transfer in Continual Learning. In *NeurIPS* 2021.

[2] Ian J. Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, Yoshua Bengio. An Empirical Investigation of Catastrophic Forgetting in Gradient-Based Neural Networks. *arXiv:1312.6211*

[3] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler Hayes, Christopher Kanan. Measuring Catastrophic Forgetting in Neural Networks. In *AAAI* 2018.