Implementing Framework of Sequential Neural Coding Network

1 Background

Lifelong machine learning is one of the machine learning areas. It focuses on making learned knowledge sustainable, so that one agent can keep learning multiple tasks, adding new training dataset not only without significant interference but also using collaboration with learned experience to improve the performance.

The sequential neural coding network (S-NCN) is introduced [1] as one of the models of lifelong machine learning, which featured as reducing interference with little space requirement increasing in multiple-task learning.

2 work description

In this project, I finished below works:

1. Implementing layer-wise prediction of the network, which is one of the 3 main parts of the framework mentioned.
2. Adjust layer connection to improve the prediction performance
3. Using learned techniques to get a high performance cpu implementation

3 implementation introductions

Layer-wise prediction splits the data training and prediction workload into multiple layers. Each layer handles part of the training data, and be trained on part of the features, just like human-beings: people sometimes using only eyes to classify a coming item and using nose to detect food.

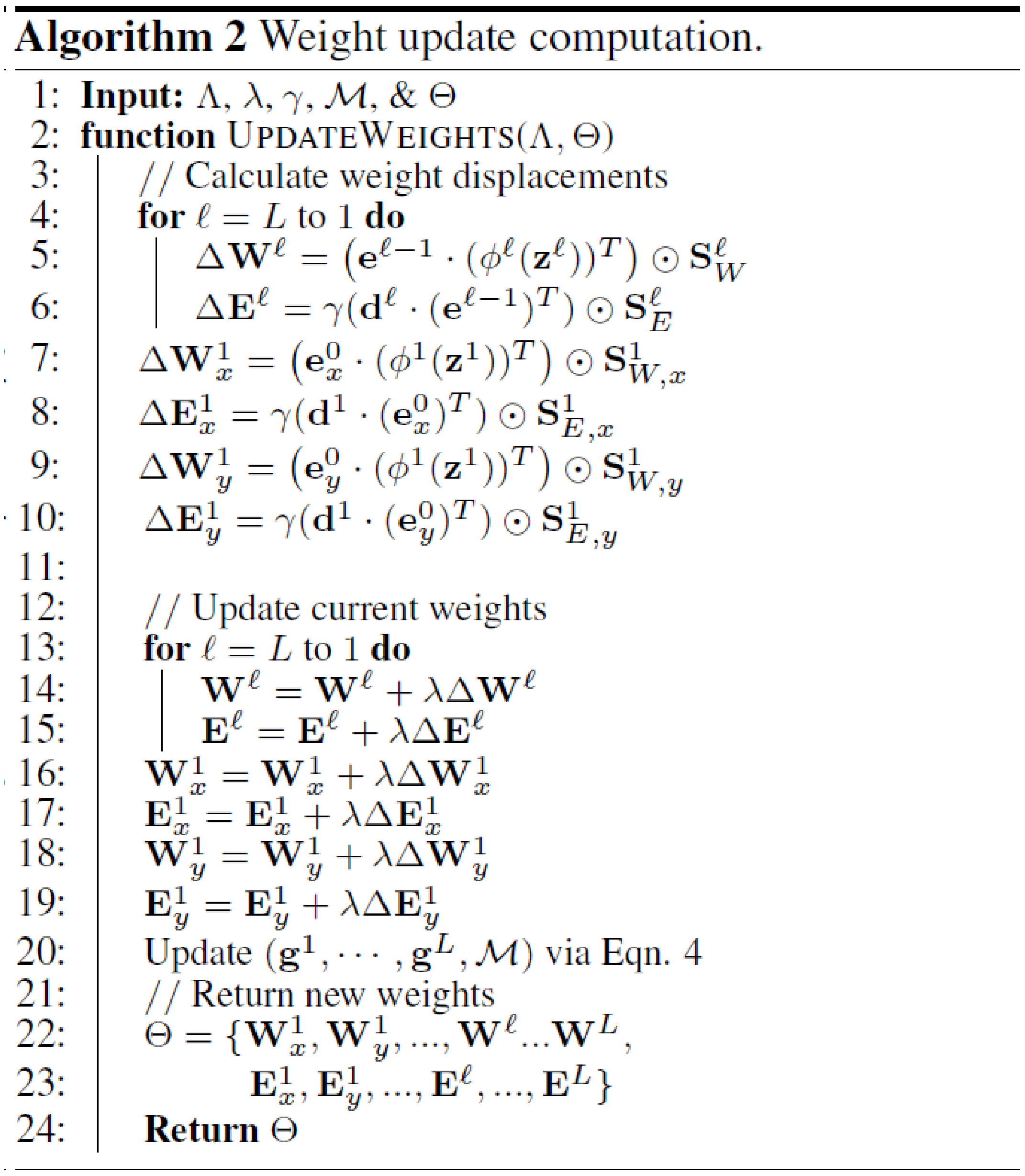
Once a layer has got its trained model in its local dataset, it then uses this model to guess the result it predict on other layers' dataset. After comparison of the prediction by 2 layers on the same piece of dataset, the guessing layer then modify its model using what causes disagreement between two layers.

Using this kind of implementation, layers are almost parallel. As they seldom communicate with each other, except for the several "guessing and comparison" action, which is more efficient to find critical points. Thus, parallel implementation becomes more accessible.

4 technique details

My layer-wise prediction implementation consists of three main parts:

1. Build multi-layer structure, then each layer performs local training. I used stochastic gradient descent with relatively small batch, which can speed up the training.
2. Different layers using their own trained weights to guess other layers prediction. Data with different prediction will be collected.
3. Using the collected data of disagreement as a new batch of dataset, to further train the data.



Algorithm 2 Weight update computation.

```
Input: $\Lambda$, $\lambda$, $\gamma$, $\mathcal{M}$, & $\Theta$
function UPDATEWEIGHTS($\Lambda, \Theta$)
    // Calculate weight displacements
    for $\ell = L$ to 1 do
        $\Delta\mathbf{W}^\ell = \left(\mathbf{e}^{\ell-1} \cdot (\phi^\ell(\mathbf{z}^\ell))^T\right) \odot \mathbf{S}_W^\ell$
        $\Delta\mathbf{E}^\ell = \gamma(\mathbf{d}^\ell \cdot (\mathbf{e}^{\ell-1})^T) \odot \mathbf{S}_E^\ell$
    $\Delta\mathbf{W}_x^1 = \left(\mathbf{e}_x^0 \cdot (\phi^1(\mathbf{z}^1))^T\right) \odot \mathbf{S}_{W,x}^1$
    $\Delta\mathbf{E}_x^1 = \gamma(\mathbf{d}^1 \cdot (\mathbf{e}_x^0)^T) \odot \mathbf{S}_{E,x}^1$
    $\Delta\mathbf{W}_y^1 = \left(\mathbf{e}_y^0 \cdot (\phi^1(\mathbf{z}^1))^T\right) \odot \mathbf{S}_{W,y}^1$
    $\Delta\mathbf{E}_y^1 = \gamma(\mathbf{d}^1 \cdot (\mathbf{e}_y^0)^T) \odot \mathbf{S}_{E,y}^1$

    // Update current weights
    for $\ell = L$ to 1 do
        $\mathbf{W}^\ell = \mathbf{W}^\ell + \lambda\Delta\mathbf{W}^\ell$
        $\mathbf{E}^\ell = \mathbf{E}^\ell + \lambda\Delta\mathbf{E}^\ell$
    $\mathbf{W}_x^1 = \mathbf{W}_x^1 + \lambda\Delta\mathbf{W}_x^1$
    $\mathbf{E}_x^1 = \mathbf{E}_x^1 + \lambda\Delta\mathbf{E}_x^1$
    $\mathbf{W}_y^1 = \mathbf{W}_y^1 + \lambda\Delta\mathbf{W}_y^1$
    $\mathbf{E}_y^1 = \mathbf{E}_y^1 + \lambda\Delta\mathbf{E}_y^1$
    Update $(\mathbf{g}^1, \cdots, \mathbf{g}^L, \mathcal{M})$ via Eqn. 4
    // Return new weights
    $\Theta = \{\mathbf{W}_x^1, \mathbf{W}_y^1, ..., \mathbf{W}^\ell ... \mathbf{W}^L,$
            $\mathbf{E}_x^1, \mathbf{E}_y^1, ..., \mathbf{E}^\ell, ..., \mathbf{E}^L\}$
    Return $\Theta$
```

Figure 1. update weight algorithm

Epoch = 10	Layer-wise	Single-layer
Data size = 1000	Acc = 0.734	Acc = 0.54
Data size =10000	Acc = 0.707	Acc = 0.542
Data size = 50000	Acc = 0.712	Acc = 0.544

5 conclusions

Layer-wise prediction is reasonably implement-friendly and believed to have advantage in performance as it meets the basic of parallel. Also, it looks reasonable in comparison with our known human behavior mechanism. The more essential part of the framework contains on the next section of model selection and context correction, which simulates action of human-being switching the most suitable knowledge and updating knowledge after encountering multiple tasks. However due to limitation of time and my capacity, these are not reproduced.

6 Thoughts on the topic

Layer-wise prediction is done with the error units which can be distinguished by training with some tasks with different “concept”, but can also be folded by training with similar or even irrelevant tasks. Will it be a more proficient way that can quickly adjust a “ridiculous” prediction to a normal one we want?

Predictions are often done among 2 layers. When comes to 3 or more layers, things go more complicated. Are there ways to go through the limitations that predictions be done in small range of layers.

References

[1] ORORBIA, A., MALI, A., KIFER, D., AND GILES, C. L. Lifelong neural predictive coding: Learning cumulatively online without forgetting. arXiv preprint arXiv:1905.10696 (2019).