Fine-tuning in Dynamic Neural Networks

Jingyang You
Research School of Computer Science, Australian National University
u6830576@anu.edu.au

Abstract. Fine-tuning is a well-known transfer learning technique which is widely used in various fixed-topology networks, while there is little research on combining fine-tuning with dynamic neural networks. In this paper, self-to-self fine-tuning, with slight modification in implementation, is applied on a modified version of the Casper neural network to test its performance on dynamic networks using a two-feature two-class dataset. Applying cross validation, self-to-self fine-tuning can achieve a compact target network (11.6 layers) with 84% test set accuracy on the average.

Keywords: Cascading Correlation, Dynamic Neural Network, Fine-tuning

1 Introduction

Fine-tuning is proven to improve generalization on various fixed-topology deep neural networks [1], however there is little research on its performance in dynamic neural networks like Cascade-Correlation (Cascor) [2] and Cascade network employing Progressive RPROP (Casper) [3, 4, 5]. It is hard to determine the appropriate training epochs in these dynamic networks. Compared to fixed-topology networks, in which large training epochs will not compromise much behavior of the network [1], a large training epoch in dynamic networks will cause significant overfitting problem, for the network will try to achieve a better train set accuracy by adding more layers to itself even if the network is over-complicated and the accuracy is high. Fine-tuning converges fast [1], which intensifies this problem. A small training epoch, however, will cause underfitting problems.

In this paper, I have applied the fine-tuning method on a modified version of Casper (mCasper) [6] and aimed to find whether it can improve generalization on dynamic networks such as mCasper. A self-to-self transfer will be applied and different chop-off layers of the fine-tuning method, different training epochs will be experimented to conclude what are the best hyperparameters for fine-tuning used in dynamic networks.

2 Related Works

In this paper, I mainly talk about the fine-tuning implementation on mCasper networks. Details related to them are presented below to provide context for this research.

2.1 Cascor, Casper and mCasper

Cascade-Correlation [2] is a dynamic network structure that starts with no hidden neurons. It then automatically trains unless there is no improvement. At this time, a new hidden neuron is added to the network and training starts again, while all input-side weights are frozen to improve training efficiency. Our research uses Cascor structure to realize automatic network construction.

Cascor is powerful, however, weak on generalization [3]. Thus, Cascade network employing Progressive RPROP [4, 5] is raised. Instead of using techniques like weight freezing, Casper trains the whole network with different learning rates (new added weights with higher learning rate) and uses Progressive RPROP as optimizer. A non-freezing network can reduce the loss by learning previous parameters, which produces more compact neural networks with better generalization.

mCasper is proved to have similar effects as Casper but with a much simpler and coherent architecture [6]. I tested fine-tuning on mCasper for it is more general and representative in the domain of dynamic neural networks.

2.2 Fine-tuning

A promising alternative to training a neural network from scratch is to apply fine-tuning [7]. If there is a well-trained neural network (source network) and its dataset has close relation with the dataset we want to train (target network), we can use fine-tuning to improve generalization [1] and save the time. In fine-tuning, a chop-off layer of the well-trained model is set and the layers after the chop-off layer are retrained with larger learning rate, while the transferred parameters are fine-tuned with smaller learning rates.

3 Data Inspection

3.1 Data Introduction

[8]'s work has shown people's pupillary responses can distinguish whether the angry face presented in videos is real or acted. 20 videos with 10 real anger and 10 acted anger are preprocessed and watched by 22 participants (and only 20 of them gave valid responses), and their pupillary responses are recorded. So, in total, there are 200 responses for "Genuine" anger and 200 responses for "Posed" anger, which is balanced and sufficient.

The dataset is already preprocessed and two PCA features are generated. The goal is to find whether a given pupillary response corresponds to a "Genuine" anger or a "Posed" anger (i.e., a 2-class classification problem).

3.2 Data Analysis

The inputs of the dataset are mean, std. dev in pupillary responses and the difference of pupillary size after watching the videos in both eyes. It is hard to manually define different ranges for these features, so keeping them as real values is a good choice. The output of the dataset is labeled as "Genuine" and "Posed". They can be encoded as 1 and 0, respectively.

The given dataset is in order, while the mixed and uniform dataset is needed. A possible solution is to shuffle the dataset, so each split contains approximately equal number of "Genuine" and "Posed". Figure 1 shows an example of shuffling.

Figure 1. An example of data shuffling. The shuffled data can be divided so each split contains similar number of 1 and 0.

PCAd1	PCAd2	label
0.1	0.1	1
0.2	0.2	1
0.3	0.3	0
0.4	0.4	0

PCAd1	PCAd2	label
0.1	0.1	1
0.3	0.3	0
0.2	0.2	1
0.4	0.4	0

Note in Figure 1, I only used these two PCA features for training, so I only kept them and the "label" column of the dataset as a new preprocessed and encoded dataset.

3.3 Data Utilization

To use the self-to-self transfer, an 8:2 train / transfer split is set. To improve the persuasiveness of the model, cross validation is applied to the train set. The whole dataset is divided into 10 splits. During 10 trials, each trial will take 8 splits as the train set and the left 2 splits as the transfer set (which ensures transfer data do not overlap train data).

After a model for a train set is trained, the parameters of it are transferred to the model of the corresponding transfer set. I also set up an 8:2 train / test split for transfer sets to retrain the transferred models and test their performance. Note that cross validation is not used in transfer sets, for it is already applied in making train / transfer splits and doing it again in transfer sets is redundant.

4 Approach

In this section, we elaborate on the implementation of fine-tuning in the condition of mCasper.

4.1 Fine-tuning in dynamic networks

Now fine-tuning is applied on a dynamic network, its implementation should be different from that on fixed-topology networks.

First, the step “reinitialize parameters in not transferred layers” [1] is unnecessary. In fixed-topology networks, the number of layers as well as the number of parameters in each layer are fixed. After loading the parameters from the source model and filling them to the target model, the remaining parameters of the target model, which are not transferred from the source model, should be reinitialized to ensure the retrain process can be conducted. While in dynamic networks like mCasper, neither the number of layers nor the number of parameters in each layer are known. If they are manually bounded, the target network is not dynamic anymore. The only way to apply fine-tuning in dynamic networks is to initialize the whole target network as the transferred partial source network. Then “retrain” is done by training the target network starting from the initial parameters, so the structure of the dynamic network can vary according to the train set of the target domain.

Second, the retrain epochs should be different. As mentioned above, the retrain epochs should be of great importance. A small epoch may cause underfitting while a large epoch can lead to severe overfitting problems, since the structure of dynamic networks can grow indefinitely, and these extra layers are memorizing the training data. To solve this problem, the number of transferred parameters and not transferred parameters in the source model are recorded. The retrain epoch is given as:

$$epoch = \alpha \times n_{transfer} + \beta \times n_{not-transfer}$$

where α, β are two hyperparameters, and $n_{transfer}$, $n_{not-transfer}$ stand for the number of transferred parameters and not transferred parameters, respectively. Intuitively, $n_{transfer}$ is weighed much less than $n_{not-transfer}$, because the transferred parameters are fine-tuned, so small number of epochs is sufficient, while $n_{not-transfer}$ records the potential number of parameters that requires completely retrain, and this corresponds to a large number of epochs. That is, $\alpha \ll \beta$. The best choice for them will be described in detail in section5.

Finally, since the number of the layers in a dynamic network is not fixed, when applying fine-tuning, it is better to set a chop-off percentage instead of a fixed number. For example, we can set 25%, 50% and 75% as chop-off percentages, instead of setting the 4th, 10th and 12th chop-off layers.

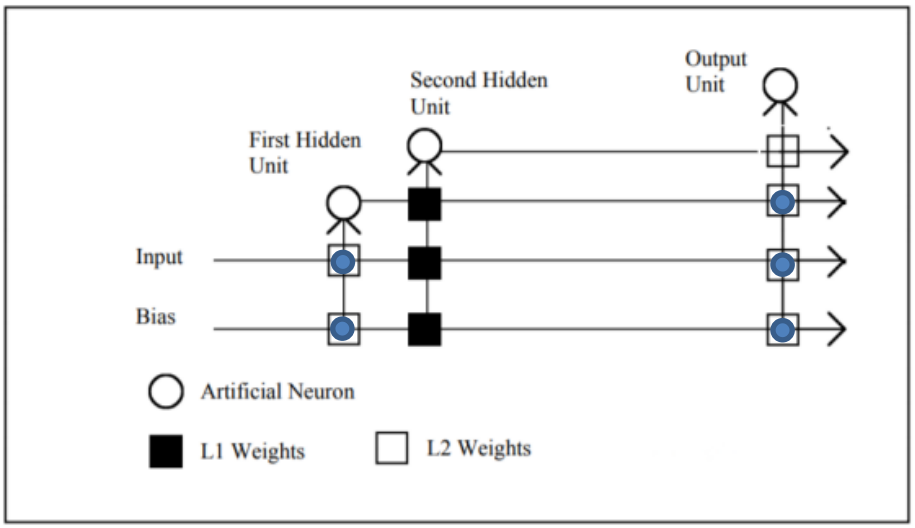
4.2 Fine-tuning with mCasper

There are some details worth mentioning when combining fine-tuning with mCasper.

First, notice that in mCasper, the output layer has dimension (num_input + num_hidden, num_out), and if this layer is transferred directly, problems will be raised. For example, if num_input = 2, num_hidden = 10 and num_out = 1 and we want to transfer the first 3 hidden layers. To ensure the transferred partial source network is a valid mCasper network, the output of the transferred network should be (2 + 3, 1) = (5, 1) instead of (12, 1). Thus, when applying the transfer, the first several parameters of the output layer are taken as the output layer of the transferred network.

Second, the smaller learning rate of parameters to be fine-tuned is not explicitly implemented, due to the structure of mCasper. When feeding the trained parameters in mCasper, the total number of layers is added by 1, thus the parameters to be fine-tuned are taken as parameters in previous layers of mCasper. [6] has shown that parameters in previous layers are trained using L2 initial learning rate (0.002, compared to 0.2 in the greatest layer). Thus, after transferring, the parameters to be fine-tuned are automatically trained using a small training rate (Figure 2).

Figure 2. The structure of target mCasper after transferring the first hidden layer of the source domain [6]. The transferred parameters are labelled with blue circles. We can find all of them lie within the field of L2 weights, which means they are automatically trained using a small learning rate after transferring.



5 Experiments

To test the performance of fine-tuning in mCasper, different values of α, β as well as chop-off percentages are tried on a well-trained mCasper (using hyperparameters in [6] with 6000 epochs). The number of hidden neurons and test set accuracy are shown in Table 1.

Table 1 the number of hidden layer and test accuracy with different set of hyperparameters. The rows are pairs of α, β and the columns are different chop-off percentages. The results are the contents of the table of the form (hidden layer, accuracy). The results are the average of 100 independent runs. Specifically, 100% means take all parameters from the source model and retrain.

	25%	**50%**	**75%**	**100%**
(0.1, 5)	(8.4, 0.68)	(9.8, 0.78)	(14.0, 0.76)	(18.4, 0.76)
(0.1, 6)	(7.7, 0.75)	(11.6, 0.83)	(15.0, 0.76)	(19.7, 0.79)
(0.1, 7)	(11.8, 0.74)	(16.4, 0.81)	(18.2, 0.76)	(22.2, 0.80)
(0.5, 5)	(7.2, 0.73)	(13.0, 0.71)	(15.6, 0.81)	(20.2, 0.74)
(0.5, 6)	(10.8, 0.74)	(14.2, 0.74)	(16.2, 0.84)	(21.0, 0.76)
(0.5, 7)	(11.2, 0.70)	(15.4, 0.73)	(16.4, 0.80)	(21.8, 0.78)
(1, 5)	(10.4, 0.69)	(15.4, 0.69)	(16.6, 0.65)	(22.2, 0.65)
(1, 6)	(11.2, 0.79)	(15.8, 0.78)	(19.0, 0.74)	(23.2, 0.74)
(1, 7)	(12.2, 0.79)	(16.2, 0.79)	(21.2, 0.78)	(23.5, 0.75)

We can find when $\alpha, \beta = (0.5, 6)$ and the chop-off percentage = 75%, the best accuracy is achieved, while $\alpha, \beta = (0.1, 6)$ and the chop-off percentage = 50%, much compact (11.6 layers, compared to 16.2 in the case with best accuracy) mCasper networks with little compromise (1%) on the accuracy are achieved.

From these results we know that medium or small choices of α are better. If α is large, the number of the transferred parameters are heavily weighed, but in fact they are not needed. This will cause the network to overfit, because fine-tuning converges fast [1], and even a slightly larger epoch can cause severe problems in the dynamic neural networks, because they can grow indefinitely, and more layers means more likely for a model (for all models, dynamic and fixed-topology) to overfit. The overall layer is increased, and the accuracy is decreased when $\alpha = 1$ compared to other cases. Similar to α, a large β will overestimate the number of parameters to be retrained thus causes overfitting while a small β will underestimate the number of parameters to be retrained, leading to underfitting problems (that is why $\beta = 6$ is generally better compared to $\beta = 5$ and $\beta = 7$).

When α is large ($\alpha = 1$), the best accuracy is achieved when the chop-off percentage 25%. When α is medium or α is small (0.5, 0.1 respectively), the best accuracies are achieved when chop-off percentage = 75% or 50%. This is a similar result compared to [9]. In [9], a self-to-self transfer also achieved the best accuracy when the chop-off layer is around 50% and 75% of the total number of layers.

6 Conclusion

In this paper, a self-to-self fine-tuning is applied on the anger dataset [8]. The fine-tuning method is slightly modified to be integrated with dynamic neural networks such as mCasper. The result shows good generalization in the transferred network with 83% test set accuracy and 11.6 hidden layers on the average. Compared to the 85% accuracy and 20.3 hidden layers in [6], we can find the network structure is significantly compressed with little compromise on the accuracy. This shows that fine-tuning can also improve the performance of in dynamic neural networks, at least the mCasper architecture.

7 Discussion and Future work

In this section, I will talk about the limitations of this research.

First, the values of α, β are guessed and tested randomly with the only knowledge that $\alpha \ll \beta$, which may cause problems like low accuracy. It is better if they can be directly calculated from the model, such as training epochs, hidden neurons, and the number of transferred parameters. Another neural network, with inputs as different training epochs, hidden neurons, and the number of transferred parameters and outputs as the test set accuracy, can be constructed if we want to find an explicit formula for the α, β.

Second, I used the formula $epoch = \alpha \times n_{transfer} + \beta \times n_{not-transfer}$ to calculate the retrain epochs. Other than α, β, $n_{transfer}$ and $n_{not-transfer}$ can possibly be modified. Basis functions can be applied to them, since the relationship among $epoch$, $n_{transfer}$ and $n_{not-transfer}$ is likely to be non-linear.

Finally, it is not enough to say fine-tuning fits all dynamic neural networks well if it is only conducted on mCasper. One possible extension can be testing fine-tuning on more complex dynamic networks like [10].

References

1. Edna Chebet Too, Li Yujian, Sam Njuki, Liu Yingchun, A comparative study of fine-tuning deep learning models for plant disease identification, Computers and Electronics in Agriculture, Volume 161, 2019, Pages 272-279, ISSN 0168-1699, https://doi.org/10.1016/j.compag.2018.03.032.S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In NIPS, 1989
2. S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In NIPS, 1989
3. Treadgold N.K., Gedeon T.D. (1997) A cascade network algorithm employing Progressive RPROP. In: Mira J., Moreno-Díaz R., Cabestany J. (eds) Biological and Artificial Computation: From Neuroscience to Technology. IWANN 1997. Lecture Notes in Computer Science, vol 1240. Springer, Berlin, Heidelberg. https://doi.org/10.1007/BFb0032532
4. Riedmiller, M. and Braun, H. (1993) A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. In:Ruspini, H., (Ed.) Proc. of the ICNN 93, San Francisco, pp. 586–591.
5. Riedmiller, M. (1994) Rprop — Description and Implementation Details, Technical Report, University of Karlsruhe.
6. Jingyang You, “Small Number of PCA Features Classification with Cascading Neural Network”, 4th ANU Bioinspired Computing Conference
7. N. Tajbakhsh et al., "Convolutional Neural Networks for Medical Image Analysis: Full Training or Fine Tuning?," in IEEE Transactions on Medical Imaging, vol. 35, no. 5, pp. 1299-1312, May 2016, doi: 10.1109/TMI.2016.2535302.
8. Lu Chen, Tom Gedeon, Md Zakir Hossain, and Sabrina Caldwell, 2017, Are you really angry? Detecting emotion veracity as a proposed tool for interaction. In Proceedings of the 29th Australian Conference on Human-Computer Interaction (OzCHI ’17), Brisbane, QLD, Australia, November 2017, 5 pages. https://doi.org/10.1145/3152771.3156147
9. Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks? In Advances in neural informationprocessing systems, pages 3320–3328.
10. Yanwei Li, Lin Song, Yukang Chen, Zeming Li, Xiangyu Zhang, Xingang Wang, Jian Sun; Learning Dynamic Routing for Semantic Segmentation, Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 8553-8562