Network Reduction Techniques for the Optical Recognition of Handwritten Digits Data Set

Longfei Zhao
College of Engineering & Computer Science
Australian National University
Canberra ACT 0200
u5976992@anu.edu.au

Abstract. In this paper, I implement Network reduction techniques [1] in the Optical Recognition of Handwritten Digits Data Set [2]. Based on Network reduction techniques [1], I developed a two-layer neural network with some my own ideas. I will focus on some technical details and some different ideas about network reduction. Furthermore, I analyzed the result and compare them with other paper. Finally, I will give some my thought about advantages and disadvantages of this method and how to improve it.

Keywords: network reduction techniques, neural network pruning

1 Introduction

Gedeon's paper gives us some guidance and intuition about how to prune redundant units in a neural network. In this paper, I will talk about technical details with the Optical Recognition of Handwritten Digits Data Set [2]. I will implement network reduction techniques [1] in this certain problem. We should focus on how to remove some redundant units and get a smaller network instead of improving the performance of this network. Since the whole work just wants to give us some simple intuition, we don't want to build a large network. A simple two-layer network is fine.

2 Data set

This data set is from a total of 43 people, 30 of them contributed to the training set and different 13 to the test set. There are 3823 instances in train set and 1797 instance in the test set. For each image, it is a 32x32 bitmap, which is divided into non-overlapping 4x4 blocks. Therefore, The number of features will be 64 and each element is an integer in the range 0 to 16. The output class is an integer in the range 0 to 9.

Table 1. Class distribution of this data set

Class	No. of examples in training set	No. of examples in testing set
0	376	177
1	389	182
2	380	177
3	389	183
4	387	181
5	376	182
6	377	181
7	387	179
8	380	174
9	382	190

2.1 Reasons for choosing this data set

Firstly, handwritten recognition is a classic classification problem for machine learning. Secondly, I think network reduction techniques just somehow improve overfitting problem. Obviously, we cannot significantly improve neural network through pruning redundant hidden units. Instead, we try to figure out the minimum neural network for certain problem, which should not reduce performance. Therefore, a mature data set seems to be a good choice, which can easily get a good model using a simple two-layer neural network.

3 Model design

3.1 Neural network structure

In this paper, I use PyTorch to build a simple feed-forward two-layer network. the number of input feature is 64 and the number of output class is 10. Hidden layer size will be a hyper-parameter. Some details are as follows.

1. I use `Tanh` function for activation function instead of the `sigmoid` function from the original paper. Since the output's range of `Tanh` function is from -1 to 1, we don't need to normalize the result. According to Deep Learning Specialization [3], `Tanh` is always better than `sigmoid`.
2. I use Adam method [4] for optimization since it would converge fast.
3. `CrossEntropyLoss` function for cost function.
4. I build a new function `hidden_layer` for getting the result of hidden layer units.

3.2 Performance of the neural network

For the baseline, I use 100 hidden units and 1000 epochs as parameters to function `train(net, num_epochs, X, Y)`. As we can see in Figure 1, the neural network converges very fast.

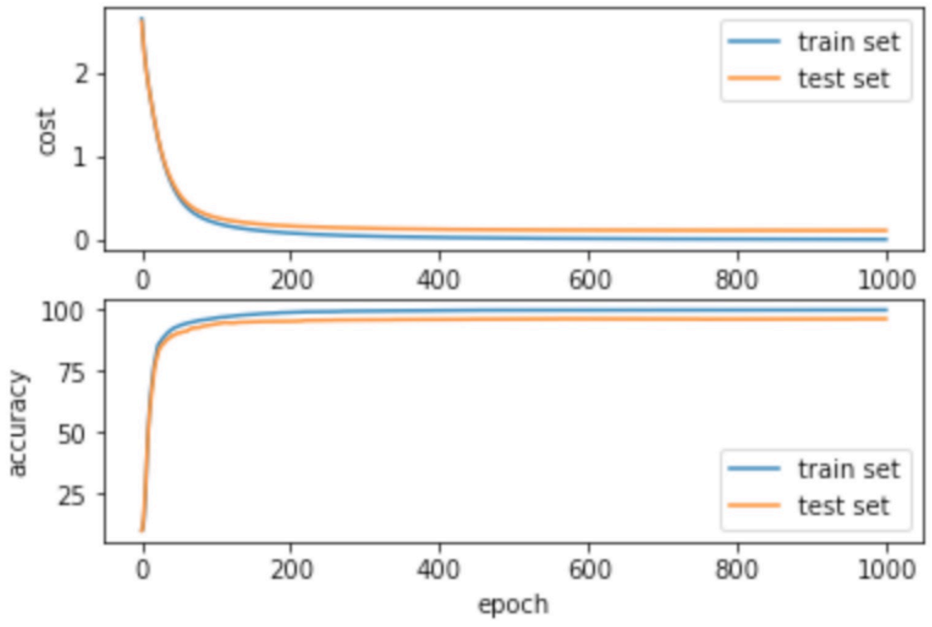

Fig. 1. Train the neural network with 100 hidden units and 1000 epochs. Final train set accuracy is 100.0% and final test set accuracy is 96.3829%

4 Remove method

I build a function `remove_redundant_units` to deal with hidden units and find out which units are redundant.

Code for function `remove_redundant_units`

```
def remove_redundant_units(net, hidden_size, X, threshold=15):
    hidden_units = net.hidden_layer(X).data.numpy()
    weight2 = net.fc2.weight.data.numpy()
```

```
removed = set()
num_removed = 0
for i in range(hidden_size):
    a = hidden_units[:, i].reshape(-1, 1)
    if np.linalg.norm(a) < num_train * 0.01:
        print('hidden unit: {0:2d}   norm: {1:.2f}'
              .format(i, np.linalg.norm(a)))
        removed.add(i)
        weight2[:, i].fill(0)
        num_removed += 1
else:
    print('There is no unit close to 0')

print('   units     angle')
for i in range(hidden_size):
    if i in removed:
        continue
    a = hidden_units[:, i].reshape(-1, 1)
    for j in range(i+1, hidden_size):
        if j in removed:
            continue
        b = hidden_units[:, j].reshape(-1, 1)
        cos = np.asscalar(a.T @ b / (np.linalg.norm(a) * np.linalg.norm(b)))
        cos = min(cos, 1)
        cos = max(cos, -1)
        angle = np.arccos * 180 / np.pi
        if angle < threshold:
            print('{0:4d} {1:4d}    {2:.4f}'.format(i, j, angle))
            removed.add(j)
            weight2[:, i] += weight2[:, j]
            weight2[:, j].fill(0)
            num_removed += 1
        if angle > 180 - threshold:
            print('{0:4d} {1:4d}    {2:.4f}'.format(i, j, angle))
            removed.add(j)
            weight2[:, i] -= weight2[:, j]
            weight2[:, j].fill(0)
            num_removed += 1
            break
return weight2, num_removed
```

1. Instead of use 15 and 165 as threshold in the original paper, I set a parameter `threshold` as threshold. Therefore, we just need to compare angle with `threshold` and `180-threshold`.
2. For each unit, I build a vector, which contains this unit's output for all train example.
3. Firstly, I tried to find which units output are close to 0. However, through my observation, their outputs norm always similar. Also, the paper doesn't define how is close to 0. I just use the number of the train set by 0.01 for intuition. In fact, this step does not change anything.
4. Secondly, I calculate the angle of any two units just like the original paper. Notice that if a unit is removed, I will tag it and don't use this unit in later.
5. If the angle is smaller than the `threshold`, we remove one of the units and add its weight which is from hidden layer to output layer to another one. For simplicity, I just change those redundant units' weight to 0 instead of truly remove them from the whole neural network. Since we think those two units have similar structure or behavior from input layer to hidden layer, we don't need to deal with weight from input layer to hidden layer, otherwise, we should focus on weight from hidden layer to output weight.
6. If the angle is larger than `180-threshold`, instead of removing both of units as the original paper, I choose to remove one of the units and minus its weight which is from hidden layer to output layer to another one. I think this method makes more sense since those units should have a different effect for output.
7. Also, I will count the number of units removed. Finally, we will get the new weight matrix and the number of units removed.

5 Results and Discussion

I set the number of epochs always be 1000. Use hidden units' size as 50, 100, 150, 200 and threshold as 15, 30, 45. The result as below.

Table 2. Experiment Result

Experiment Id	1	2	3	4	5	6
Hidden units	50	50	50	100	100	100
Threshold	15	30	45	15	30	45
Train set accuracy	99.9738%	99.8954%	99.9215%	100%	100%	100%
Test set accuracy	95.8820%	95.8820%	96.4942%	96.6055%	96.6611%	96.3272%
Remove units	1	4	11	4	8	23
New Test set accuracy	95.8820%	96.0490%	95.7151%	96.8055%	96.6055%	96.2159%

Experiment Id	7	8	9	10	11	12
Hidden units	150	150	50	200	200	200
Threshold	15	30	45	15	30	45
Train set accuracy	100%	100%	100%	100%	100%	100%
Test set accuracy	97.0505%	96.3829%	96.5498%	96.4942%	97.1063%	96.5498%
Remove units	5	14	54	4	22	59
New Test set accuracy	97.0506%	96.4942%	96.0490%	96.4942%	96.9950%	96.3272%

Compare to Methods of Combining Multiple Classifiers and Their Applications to Handwriting Recognition [5] which used multiple classifiers and their best result is 95% recognition and 5% rejection. We can see even simple neural network have a strong learning ability. Deep learning has a great advantage in those tasks comparing some traditional machine learning techniques.

When we set threshold from 15 to 45, removed units increases. Notice that when the threshold is 30, sometimes the new neural network which removes redundant units will become better. It gives us a tuition that 30 degrees maybe a good choice for threshold if we want to better the model.

Also, by observing the results, we can see that we cannot get the minimum neural network by this method. In fact, the neural network is still far away from the minimum neural network.

It's interesting to notice that even if we set the threshold to like 60 or 80 degrees, it will remove much more units but the accuracy is still acceptable. I did those experiment in the code but not put those result here. If you are interested, you would check the jupyter notebook. I think it gives us intuition that threshold should be a hyper-parameter which need to carefully set.

6 Conclusion and Future Work

The benefit of this method is that we can use it to prune some units and get a smaller neural network without retraining. It can give us intuition about how to prune some of the redundant units. However, there are, I think, still some disadvantages and problems need to figure out.

Firstly, since the hidden layer behavior is much more complex and hard to be predicted, the whole thing seems like lack of mathematical deduction. It is more like followed some basic idea. For example, we don't consider the distribution of input. However, obviously the units' behavior and threshold are closely related to the distribution of input. Secondly, the threshold is more like a hyper-parameter, that means we need to figure out a good threshold for certain task. This step will cost extra time. At the same time, It cannot give us the minimum neural network. Thirdly, I think it should be related to the number of train examples, which means we should use the number of train example as a parameter to use this method. Finally, it's also related to which activation function we choose. For example, in this paper I choose function `tanh` instead of function `sigmoid`. Different activation function will give us a different result.

For improving this method, I have some thought. Firstly, instead of using hidden layer's output, I think maybe using output layer's result makes more sense. For each unit, we can compare their contribution for output. If some of them are similar, which means angle is close to 0, we can combine them together and if some of them are opposite which means angle is close to 180 we can somehow remove them together. Notice that in this situation since we have two layers' weight, how to combine them or remove them together is still a problem. Secondly, I think that using train set and test set together to prune redundant units maybe make more sense because we cannot assume the distribution of train set and test set are same. Therefore, just using the train set to do this step may cause some bias.

For future work, I think how to use this method in a large network and other types of networks are very attractive. It could be a fixed step when we train a network to give us some intuition about this network's redundancy.

References

1. Gedeon, T., Harris, D.: Network reduction techniques. In: Proceedings International Conference on Neural Networks Methodologies and Applications, pp. 119-126 (1991)
2. Alpaydin, E., Kaynak, C.: Optical recognition of handwritten digits data set. Department of Computer Engineering, Bogazici University, 80815 Istanbul Turkey (1994). http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits
3. Ng, A., Katanforoosh, K., Mourri, Y.B.: Deep learning specialization. deeplearning.ai (2018). https://www.coursera.org/specializations/deep-learning
4. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Proceedings International Conference on Learning Representations (ICLR). arXiv preprint arXiv:1412.6980, pp 1-13 (2015)
5. Xu, L., Krzyzak, A., Suen, C.Y.: Methods of combining multiple classifiers and their applications to handwriting recognition. IEEE Transactions on Systems, Man, and Cybernetics 22, 418-435 (1992)