Hidden Neurons Reduction Practice on Classification Tasks

Yufeng Xu

Research School of Computer Science,
Australian National University

u6089801@anu.edu.au

Abstract. In this paper, we applied a neural network reduction technique based on the distinctiveness of hidden units to two classification tasks: 1) a binary classification task with a 3-layer small neural network and 2) a 10-class classification task with a convolutional neural network (CNN). In the binary classcification task, we found a small number of units in the hidden layer have been pruned (2 out of 9) with a litte decrease in network accuracy (from 97.22% to 93.67% in average). This is compared to another classfication model with SVM on the same dataset, which has an accuracy of 96.67% or 97.78%. In the CNN, pruning is applied to the last fully connected layer. We achieved an 68.61% accuracy of the CNN model before pruning and got an accuracy of 68.51%, 68.63% after pruning 10 and 8 units from the layer (originally has 600 units). Compared to other models applied on the same dataset (error rates range from 8.6% to 15.13%), the accuracy of our CNN model is not desirable but it shows the introduced pruning method is workable in both simple neural network and CNN to some extent.

Keywords: neural network reduction, hidden unit, distinctiveness, back-propagation neural network, convolutional neural network

1 Introduction

Reducing the size of a neural network is an important and meaningful topic to discuss and research. Usually, the complexity of the network would be increased to achieve better performance. However, the large number of units in a network could occupy too much computational resources and increase the time for training. Also, there would be some undesirable hidden units in the network, which either produce constant effects or be similar/complementary to other units. These undesirable units contribute little to the network performance. Consequently, it is worthwhile to design some methods to identify these hidden units in a neural network and prune them. In this study, we referred to Gedeon and Harris's paper about network reduction techniques and partially implemented the pruning technique based on distinctiveness of hidden units (1991). After that, we applied this technique to two different models: a 3-layer neural network with 9 units in the hidden layer and a CNN with a last fully connected layer containing 600 units. It is shown from the tests that the partially implemented pruning technique works on both kinds of neural networks, although a small accuracy decrease occurred in the 3-layer network and insignificant accuracy decrease occurred in some tests of the CNN. Also, the number of the pruned units in both models are not that large. In Section 3, these results will be discussed in detail.

To make the results in this paper comparable, here we use the two networks on two different data sets.

The 3-layer network (a 18-node input layer, a 9-node hidden layer, and a 2-node output layer) worked on a dataset of climate model simulation crashes, with 540 instances in total. Each instance has 21 properties: ID of the study which this simulation belongs to, simulation ID, values of 18 climate model parameters and the simulation outcome. The 3-layer network is used to predict the climate model simulation outcome (success or failure). The motivations of using this dataset are: firstly, it is a real-world data set; secondly, predicting simulation results based on this data set is meaningful, since climate models are always complex and with large volume of code which makes debugging much harder while using model prediction can help in locating bugs and problems; thirdly, this data set has a suitable complexity for the study on a small size neural network, compared to the CNN.

The CNN (3 channels, two convolution layers with 32 and 64 filters all in 3*3 respectively, two average pooling layer with 2*2 window size without overlapping, a fully connected layer with 600 units and a 10-unit output layer) worked on the CIFAR-10 data set (Krizhevsky, 2009). This data set contains 60000 32*32 color images in 10 classes with 6000 images in each class. There are 50000 images for training and 10000 for testing. The test images are randomly selected from each class and contains exactly 1000 images for each class. CNN is used to classify these images. Motivation for choosing this data set is that the CNN is a proper model for processing images and the size of the data set is large enough, compared to the previous one, so that the variety of problems we look at in this paper is enriched.

2 Methods

2.1 Preprocessing on Data Set

To better model the problem, the data set for the 3-layer network is pre-processed. The first two properties, study ID and simulation ID, are removed since they don't affect the simulation outcome. Additionally, every instance in the data set is originally a text. In order to take the data into a neural network, text data is split into columns. Since in the original data set, the values of the 18 model parameters have been scaled into [0,1] and the outcome is represented by 0 (failure) and 1 (success), no further encoding is needed. The CIFAR-10 data set is not preprocessed and used directly through the given class/methods in torchvision.

2.2 Technique to Identify Undesirable Units

According to Gedeon and Harris (1991), the units which are not necessary for a neural network can be identified as four classes. The first class of undesirable unit is the unit which is not functional, including the unit with zero-weight or very low weight and the unit which is always on/off. The second class is the similar units which have the identical output for every input pattern. Two or more units can be similar. The third class is the group of units which have no function when being in effect together. This kind of group can contain two or more units and when it contains two units, it represents these units are complementary to each other. Last class is the group of units producing a constant effect together for all input patterns.

The term, distinctiveness, is introduced in this context to help identifying the above classes of undesirable units according to Gedeon and Harris (1991). To determine the distinctiveness of a hidden unit, a vector is constructed to record the unit output activation for all input patterns, which means the vector has a length equal to the number of input patterns. Then the vectors of different hidden units can be compared to see the functionality of these hidden units. For example, if the vectors of two hidden units are identical, it means these two units are similar, belonging to the second class above.

Identifying the undesirable hidden units is not our ultimate goal. The actions taken after that are the most important. As Gedeon and Harris (1991) described in their paper, for similar units, one unit is kept and others should be removed after adding their weights to the remaining unit; for units which produce no function or constant function, they should all be removed from the network although corresponding biases need to be adjusted if a constant effect is produced.

2.3 Implementation

In my study, I implemented part of above technique and applied it to the trained model. I used the vector introduced in section 2.1 to identify three types of undesirable hidden units: the single unit producing a constant effect, the units which have similar functionalities and the pairs of units which are complementary to each other.

Functionality Vector of Hidden Unit. As mentioned before, this vector records the unit output activation values for all input patterns. To get these values, when constructing the forward function of the neural network, I chose to not only return the prediction result but also the output activation values of the hidden layer. Here is the implemented forward function:

```
def forward(self, x):
    """
       In the forward function we define the process of performing
       forward pass, that is to accept a Variable of input
       data, x, and return a Variable of output data, y_pred,
       and a Variable of hidden layer activation output, h_output.
    """
    # get hidden layer input
    h_input = self.hidden(x)
    # define activation function for hidden layer
    #h_output = F.sigmoid(h_input)
    h_output = F.leaky_relu(h_input)
    # get output layer output
    y_pred = self.out(h_output)

    return y_pred, h_output
```

With this forward function, we can get a matrix which contains the functionality vectors of all hidden units. We can then use it to compare the vectors.

Single Unit Producing A Constant Effect. To identify this type of unit, we can check the functionality vector of every hidden unit. If the elements in the vector of a unit are all identical, it means the unit produces same output activation for all patterns. Consequently, this unit can be identified as this type of undesirable units. The code for identifying is shown below:

```
# collect hidden units output activation vectors
_, H_out = net(X) # H_out should be 360*9
out_matrix = H_out.data

constE = []  # store constant effect unit
for i in range(hidden_neurons):
    v = out_matrix[:, i].numpy()
        if len(set(v)) == 1:
            constE.append(i)
        print('constant effect unit found.')
```

After identifying, we remove these units for network reduction. Also, corresponding output biases are updated according to the value of the constant output activation. Here is the code for handling the units:

```
removed = []  # store indices of units to be removed

for unit in constE:
    # bias is updated to (bias - const effect * weight)
    const = out_matrix[:, unit][0]
    b_out = torch.add(b_out, -1, const * w_out[:, unit].numpy())
    # remove the unit
    removed.append(unit)
```

Similar Units. To identify similar units, which can be a group of two or more units, we can compare the functionality vectors in pairs and go through all pairs of the hidden units. The metric used for comparison is calculating the angle between two vectors. If the angle is no larger than 15°, we identify this pair of units to be similar units (Gedeon & Harris, 1991). In the technique paper, the origin of the vectors is moved to 0.5, 0.5 considering the situation that after using the sigmoid function, elements in the vectors are constrained in the range from 0 to 1. Moving the origin can broaden the angular range from 0-90° to 0-180° in this situation. However, in my neural network, LeakyReLu is used instead of sigmoid and there is no such constraint on the output values. Under this circumstance, with 0, 0 origin, we can still get an angular range of 0-180°. So, in my implementation, I didn't move the origin of the functionality vectors. Here is the code for calculating angles and identify similar units:

```
similar = []  # store similar pairs or groups
```

```
for u1 in range(hidden_neurons - 1):
    for u2 in range(u1+1, hidden_neurons):
        v1 = out_matrix[:, u1].numpy()
        v2 = out_matrix[:, u2].numpy()
        cos = v1.dot(v2)/(np.sqrt(v1.dot(v1))*np.sqrt(v2.dot(v2)))
        angle = np.arccos(cos)*180/np.pi
        print('Vector angle between (%d , %d): %f' % (u1, u2, angle))

        # angle <= 15, similar pair/group
    if angle <= 15:
         similar.append([u1, u2])
         print('similar pair found!')
```

Originally, I attempted to group the pairs with a common unit into a group, which I identify them as all similar to each other. However, further consideration told me that this is not correct since 'A is similar to B', 'B is similar to C' don't imply 'A is similar to C'. So, I still recognized similar pairs to be processed.

After identification, for each similar group (with 2 units, as mentioned above), one unit in the group is kept and weight vectors of other units are added to the remaining unit before removing those units. The corresponding code is shown below. It is worth mentioning that for each group, I checked how many units in the group are already removed due to previous reduction (since a unit can belong to more than one undesirable type) before performing the reduction on the group. If only one unit remains, no further reduction is needed; if more than one unit remains, apply reduction and weight update on those units. The aim of this checking is to avoid repeated weight addition.

```
for group in similar:
    diff = list(set(group).difference(set(removed)))

    if len(diff) > 1:
        group_exist = diff
        # keep one, remove others
        # update weights
        for unit in group_exist[1:]:
            w_out[:, group_exist[0]] = torch.add(w_out[:, group_exist[0]], 1,
w_out[:, unit])

        removed.extend(group_exist[1:])
```

Complementary Pairs. We use angles between functionality vectors to identify units in this type as well. If the angle is not smaller than 165°, we identify this pair of units to be complementary units (Gedeon & Harris, 1991). The code for identification is:

```
comp = []  # store complementary pairs

# angle >= 165, complementary pair
# remove both units
if angle >= 165:
    comp.append([u1, u2])
    print('complimentary pair found!')
```

For a complementary pair, since they produce no effect together, both of the units should be removed.

```
for pair in comp:
    if pair[0] not in removed and pair[1] not in removed:
        # remove both units
        removed.extend(pair)
```

3 Results and Discussion

For the 3-layer neural network, before pruning, I trained the network with the Study 1 and Study 2 data in the dataset and tested on Study 3 data, which is aligned with the study using SVM (Lucas, et al., 2013). According to the result, it is found that the model doesn't totally fit the training set while it has an accuracy of 97.22% on the testing set. 97.22% is the highest accuracy value I got after many trials. Here, the confusion matrix Fig.1 is used as another way to show the performance of the network. In the study using SVM (Lucas, et al., 2013), researchers also use the confusion matrix to

show prediction results. This is one of the reasons why I use confusion matrix to analyse the results. It will be more intuitive to show the comparison when the same analysis method is applied. Also, since this is a binary classification task, using confusion matrix is simple but useful in evaluating the results. Fig.2 is the testing result in the mentioned study. It can be identified that the accuracy using SVM with different decision criteria (avg, sum, snr) is either 96.67% or 97.78%. Compared with the performance of the built network model, it is shown that the network performed at the same accuracy level.

Many trials and modifications were made during network construction process, to achieve a better result with a smaller size of the network. I first chose the sigmoid function as activation function for hidden layer and then changed to LeakyReLU. During the tests, I found when using sigmoid, the network hardly predicted the outcome as 'failure', which resulted in low accuracy. With LeakyReLU, accuracy is improved. This might result from the limitation of sigmoid function in error backpropagation process. Also, I used a heuristic for choosing the number of hidden layer units, which is 'the number of hidden nodes is equal to log(T), where T is the number of training samples' (Wanas, Auda, Kamel, & Karray, 1998, pp.918). As mentioned before, Study 1 and 2 are used to train the network and there are 360 training instances in total. I tried 7, 8 and 9 for the number of hidden units and got the best result with 9. There are other heuristics (Stathakis, 2009) as well. But since the number of hidden units grows when using them, I finally chose the log(T) heuristic.

```
Training process:
Epoch [1/5000] Loss: 0.5371  Accuracy: 91.11 %
Epoch [501/5000] Loss: 0.1282  Accuracy: 96.11 %
Epoch [1001/5000] Loss: 0.1017  Accuracy: 96.94 %
Epoch [1501/5000] Loss: 0.0930  Accuracy: 96.94 %
Epoch [2001/5000] Loss: 0.0881  Accuracy: 97.22 %
Epoch [2501/5000] Loss: 0.0849  Accuracy: 97.22 %
Epoch [3001/5000] Loss: 0.0820  Accuracy: 97.22 %
Epoch [3501/5000] Loss: 0.0795  Accuracy: 97.22 %
Epoch [4001/5000] Loss: 0.0762  Accuracy: 97.78 %
Epoch [4501/5000] Loss: 0.0728  Accuracy: 97.78 %

Confusion matrix on training set:

  26    6
   1  327
[torch.FloatTensor of size 2x2]

Testing Accuracy: 97.22 %

Confusion matrix on testing set:

  11    3
   2  164
[torch.FloatTensor of size 2x2]
```

Fig. 1. 1-hidden layer network training process and testing result

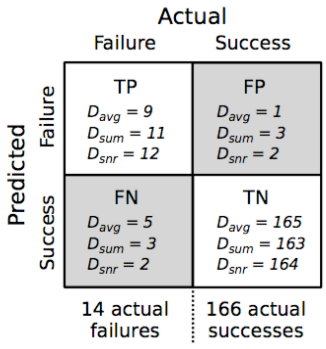

Fig. 2. Testing result in the study using SVM (Lucas, et al., 2013)

After pruning the network, from the result in Fig. 3, we can see that 2 units are pruned and the accuracy decreases a little compared to the original model. The confusion matrix for the test set is also shown. Aside form this result, I performed another 4 tests which shows accuracy of 92.78%, 93.33%, 92.22% and 94.44%.

```
Vector angle between (4 , 5): 43.910521
Vector angle between (4 , 6): 29.121498
Vector angle between (4 , 7): 40.319016
Vector angle between (4 , 8): 31.734862
Vector angle between (5 , 6): 34.771062
Vector angle between (5 , 7): 42.977090
Vector angle between (5 , 8): 32.868732
Vector angle between (6 , 7): 30.354935
Vector angle between (6 , 8): 9.788480
similar pair found!
Vector angle between (7 , 8): 33.285178

2 hidden unit(s) removed.

Performance of the pruned network:
Testing Accuracy: 95.00 %

Confusion matrix for pruned model on testing set:

  12    2
   7  159
[torch.FloatTensor of size 2x2]
```

Fig. 3. Pruned 3-layer network performance

Ideally, we expect the pruned network to perform as good as the original network. The worse performance of the pruned network most likely results from the partially implemented reduction technique. Only three types of undesirable units have been detected in my implementation, so there might be other types of undesirable units remaining in the network and these units might have correlations with the removed units. For example, there might be remaining units which produce constant effect/no effect together with the removed units. If so, as they have not been identified and handled, the total performance might be affected. Also, it seems that the accuracy doesn't decrease much for the network I built in this study, though I only implemented part of the technique. What is worth mentioning is that the topology of the original network should be considered as well. The original network is not complex with only three layers and 9 hidden nodes. If a more complex network with more hidden layers is tested here, the decrease of accuracy might be more apparent.

Additionally, during the implementation, the order of handling different types of undesirable units was considered. The single unit producing constant effect was handled first, then the complementary pairs were removed and lastly, I turned to the similar groups. Indeed, the constant-effect unit can also be handled lastly based on the tests. The order between the other two types is important. Assuming we identify two similar groups (A, B, C) and (D, E) and we also identify that D is complementary to all units in the first group, so as E, then different order in handling similar groups and complementary pairs will give different reduction result: if firstly reduce the similar units, then one of each group remains, and both of them will be removed when we handle the complementary pairs; if firstly process complementary pairs, only one unit in the first group will be kept and the second group are all removed. Considering this assumed case, I chose to handle complementary pairs first in my implementation.

For the CNN, before pruning, several attempts have been made on the number of filters in each convolution layer, the pooling method, the size of the FC layer, the learning rate and the kernel size as well. The results show that in this data set, a kernel of 3*3 is better that 5*5 since using 3*3 balances the emphasis on details and background of the input images. The results on accuracy also show that average pooling performs better that max pooling here. The size of the 1st convolution layer has been increased from 10 to 32 to achieve better accuracy. Similar conditions applied to the 2nd convolution layer as well as the FC layer. The best achieved accuracy before pruning is 68.61% on the test set.

After pruning, the results are when 10 units are pruned, the accuracy is 68.51% and when 8 units are pruned, accuracy is 68.63%. It is worth mentioning here that different from the previous model, we use 55 degree here to identify the similar pairs. It has been tested that when using degrees less than 40, almost no units are pruned; when it comes to 45 degree, small number of units are pruned with a tiny increase on the accuracy; 50 and 55 degree help in pruning more units but result in a tiny decrease in the accuracy. It can be seen from here that the original technique is need some modification to be compatible with a hidden layer with large number of units and large amount of input patterns. The pruned units in the network are all similar units, which tells that consideration is need on the degree used to identify complementary pairs. We also discovered that pruning performance based on the activation outputs from training set and testing set are almost identical. So maybe pruning can be usually applied based on the test set to reduce the time cost.

4 Conclusion and Future Work

In this study, I constructed a 3-layer neural network to solve a binary classification task and a CNN to classify 10 classes images. Several heuristics have been tried to determine the number of units in the hidden layer and the one which produces the best result is log(T) where T is the number of training samples. The performance of my neural network is compared with another study on the same data set using SVM to classify and it is found that the two models perform at the same level. I also implemented part of the network reduction technique by identifying three classes of undesirable units and handling them. The pruned network performed a little bit worse than the original model, which is most likely due to the partially identified undesirable classes. It is also found that the order of handling different types of undesirable units is important. More research can be done in the future to figure out whether the reduction order matters when more undesirable types are identified.

Works Cited

Climate Model Simulation Crashes Data Set. (n.d.). Retrieved May 1, 2018, from UCI Machine Learning Repository: http://archive.ics.uci.edu/ml/datasets/Climate+Model+Simulation+Crashes#

Gedeon, T., & Harris, D. (1991). Network reduction techniques. *Proceedings International Conference on Neural Networks Methodologies and Applications, 1*, 119-126.

Krizhevsky, A. (2009). *Learning Multiple Layers of Features from Tiny Images*.

Lucas, D., Klein, R., Tannahill, J., Ivanova, D., Brandon, S., Domyancic, D., & Zhang, Y. (2013). Failure analysis of parameter-induced simulation crashes in climate models. *Geoscientific Model Development, 6*(4), 1157-1171.

Stathakis, D. (2009). How many hidden layers and nodes? *International Journal of Remote Sensing, 30*(8), 2133-2147.

Wanas, N., Auda, G., Kamel, M. S., & Karray, F. (1998). On the optimal number of hidden nodes in a neural network. *Electrical and Computer Engineering, 1998. IEEE Canadian Conference*, *2*, pp. 918-921.