Character Recognition using a Multilayer Feedforward Neural Network and Backpropagation

Chi Zhang
Department of Electrical Engineering and Computer Science,
University of Tennessee, Knoxville, TN 37996-3450, USA
Email: czhang24@utk.edu

Abstract—**Neural network is widely used for classification and regression problems, and are proved to be effective with high successful rate. In this paper, a two layer feedforward artificial neural network is built based on the back propagation algorithm. The goal of this paper is to apply neural network in a classification problem, to distinguish which character a given input is. By training the neural network with learning set, relation between input and output is represented by weights in the two layers neural network. Then validate set is applied to indicate network convergence in the expectation to avoid overfit. After last, the trained neural network is applied to classify 10 capital letters A, C, D, E, F, G, H, L, P and R, in an given test sets, which is never seen before.**

In this project, neural network achieves an average classification rate of 93.99% in 32.97 seconds. By turning parameters in the network, effects in classification performance and training time are investigated. Small number of hidden nodes results in worse rate and longer training time. The reason is too little hidden nodes lose degrees of freedom in representing the relation between inputs and outputs. However, increasing hidden node number does not always improve identifying accuracy and may take longer time to converge. Secondly, higher learning rate leads to heavy learning error fluctuation, which means global optimal solution may not be found. So classification rate is lower. But meanwhile local optimal solution may be found quickly with shorter training time. The third observation is degrading input generalizations will also degrade identifying rate significantly, because of details are lost in data generalization. At last, k-fold method is used to inspect different validation techniques and is proved to improve successful rate to 96.62%.

I. Introduction

Neural network is one of the most effective learning methods to approximate real-valued, discrete-valued, and vector-valued functions. It has been widely used in classification and regression problems, such as handwritten characters, recognizing spoken words, and face recognition [1].

Given three data sets, the task is to recognize 10 characters in unseen test set, after training the neural network with learning set and validating it by validation set. This problem is formulated as a classification problem. Artificial neural network is known as a good candidate to solve classification problems, so we design a basic neural network and attempt to prove its effectiveness by this character recognition problem. We need three main steps to solve this problem. Firstly, after initializing the neural network, we train it with training set. In each training period, all the training characters, 100 examples of each character in this project, go through the network and get actual network output, By comparing the actual network outputs with the target outputs, we can obtain network errors, which is then applied in updating all network weights. After each training iteration, we also use validation set to validate errors of network at real time. Validating process does not effect weights update, which means it is only used for validation. When validation error increases, we say that neural network is converging, and stop training it. Otherwise, overfit will decrease successful rate. After training and validating processes, we believe that neural network has been well trained and can be used for characters classification. A set of unseen data is inputted into the network and compared with target output to check if network can successfully identify characters [2].

In this project, we design a three layer neural network, with 96 input nodes, which represent 12 by 8 character bits,10 hidden nodes, and 10 output nodes. We also add one bias node in input layer, and one bias node in hidden layer. The value of these two bias nodes are always 1, but related weights are updated in each epoch. The learning rate is set to be a small number of 0.1. Each characters, A, C, D, E, F, G, H, L, P and R, has 100 samples for training, and 250 samples for validating and testing, respectively. Number of weight between input layer and hidden layer is therefore 97 by 11 and number of weights between hidden layer and output layer is 11 by 10. All the weights are initialized to be small random number between -0.01 and 0.01.

II. Experiments

A. Neural network structure

The network structure we use is multi layer feedforward network with backgropapation algorithm. The output from one layer feeds forward into the next layer of neurons, and nodes between two layers are fully connected with various weights. Error is calculated by the difference of outputs from output layer and target outputs. Then errors are feed backward to adjust weights, which are then applied to next iteration to generate new network outputs.

1) Input units: Data sets in this project is formatted 12 by 8 pixels, with 0 indicating an "off" pixel and 1 indicating an "on" pixel. There is an additional 10 digits vector identifying the actual identity of this character.

2) Hidden units: Neural network with only one hidden unit can represent linear functions, and with more than one hidden unit can represent nonlinear functions. In this project, the problem we need to solve is nonlinear so a couple of hidden units are used here. However, choosing the number of hidden units properly is deterministic in a neural network design. Too many hidden units do not guarantee higher network classification performance, and too little hidden units may fail in achieve high identifying rate. Discussion about how to choose hidden units number is in the next section.

3) Output units: According to the requirement of character classification target, we have 10 output units sequence representing each character. For example, the first character "A" is represented by [1 0 0 0 0 0 0 0 0 0], with the first digit valued 1 and the rest to be 0. This is also consistent with the way the given data sets do to represent characters.

B. Backpropagation algorithm

Outlining the structure of our target neural network, we give pseudo code of backpropagation algorithm (Algorithm 1) used in this project. One specific details in updating weights is that after training stopping, last iteration's weights should be stored as neural network weights, because last iteration's weights having the lowest error.

Algorithm 1 Online Backpropagation Pseudo Code for the characters classification problem

```
Initialize all $v_{ih}$ and $w_{hj}$ to rand (-0.01, 0.01)
$RUN = 1$
While $(RUN)$
DO{
$store$ all $w$ and $v$
/* Training */
  $epoch = epoch + 1$
  for $all$ $(x^t, r^t) \in X_{training}$ in random order
      for $all$ hidden nodes $z_h$
            $z_h = sigmoid\left(\sum_{j=0}^{d} w_{hj} x_j^t\right)$
      for $all$ output nodes $y_i$
            $y_i = softmax(\sum_{h=0}^{H} v_{ih} z_h)$
  for $all$ weights $v_{ih}$
      $\Delta v_{ih} = \eta\left(r_i^t - y_i^t\right) z_h$
      $v_{ih} = v_{ih} + \Delta v_{ih}$
  for $all$ weights $w_{ih}$
      $\Delta w_{hj} = \eta\left(\sum_{i=1}^{K}\left(r_i^t - y_i^t\right) v_{ih}\right) z_h\left(1 - z_h\right) x_j^t$
      $w_{ih} = w_{ih} + \Delta w_{hj}$
/* Validating */
  for $all$ $(x^t, r^t) \in X_{validating}$
      for $all$ hidden nodes $z_h$
            $z_h = sigmoid\left(\sum_{j=0}^{d} w_{hj} x_j^t\right)$
      for $all$ output nodes $y_i$
            $y_i = softmax(\sum_{h=0}^{H} v_{ih} z_h)$
  $err\left(epoch\right) = \frac{1}{2}\sum_{x \in X_{validation}}\left(\sum_{i=1}^{K}\left(r_i^t - y_i^t\right)^2\right)$
  if $err\left(epoch\right) > err\left(epoch - 1\right)$
      $RUN = 0$
}
/* Testing */
for $all$ $(x^t, r^t) \in X_{testing}$
    for $all$ hidden nodes $z_h$
          $z_h = sigmoid\left(\sum_{j=0}^{d} w_{hj} x_j^t\right)$
    for $all$ output nodes $y_i$
          $y_i = softmax(\sum_{h=0}^{H} v_{ih} z_h)$
    if $y == r$
          $"Success\,Classification"$
    else
          $"Failed\,Classification"$
```

III. Results

A. Training with large iterations

Firstly, we do an experiment with large iteration numbers, saying 5000 training epochs, to have an general sense about error rate. Fig 1 shows results of both validation set error and training set error with 5000 iterations. The successful rate is 93% and training time is 13433.34 seconds. The two curves illustrate that training set error is always decreasing while validation set error increase after certain training epoch. Obviously, from epoch within 200, minimal error point appears.

To be more specific, we do the experiment with 100 epochs and get more details about error information from validation and training sets. The successful rate is 93.44%, and training time is 102.0485 seconds. Rate is higher than training for 5000 iterations, and training time is sharply decreasing. This observation inspires us that after the turning point is reached, more training iteration cannot help improving performance. In fact, although error of training set is always decreasing with more training iterations, the successful rate will decrease. This is called overfitting. To avoid overfitting, we use validating set to constrain the training process. We also collect sum of squared errors for each output unit in the 100 iterations, as Figure 2 shows. These ten lines in Figure 3 decreases in the whole training period, which reflect the square error of training set always increases as epoch number increases.

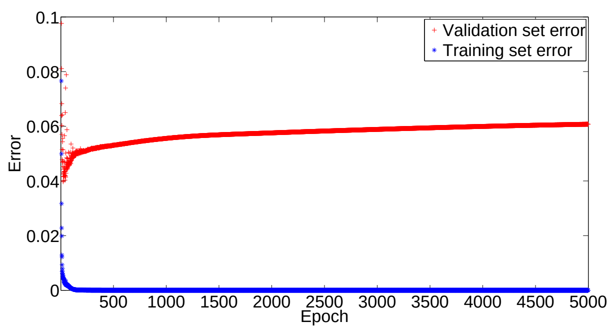

Figure 1. Error versus weight updates(epoch)

B. Training with validation

According to the above observation, we add the restraint condition, that if error of validation is lower than 0.05, any increasing after will terminate training process. With this constrain, hopefully higher successful rate will be obtained, along with training time saving. We run the experiments for 10 times, showing data in Table I. The average classification

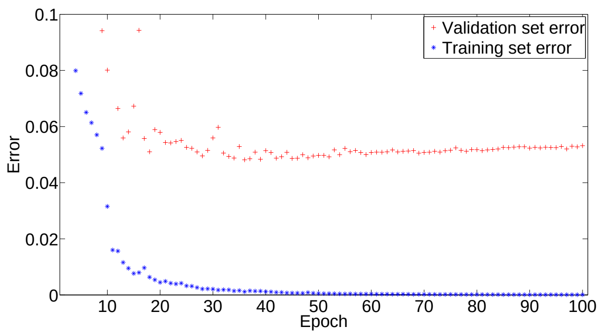

Figure 2. Error versus weight updates(epoch)

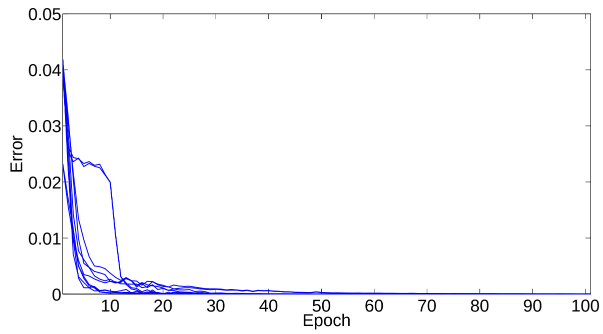

Figure 3. Sum of squared errors for each output unit

rate is 93.988%, with average training time 32.97 seconds. Average training epoch is less than 50 iterations. Validation affords us better network successful rate as well as optimal training time.

For each character, we also calculate its classification rate, as Table II shows. From the table, we can see that some characters have identifying rate as high as 98% ~ 99%, but the worst case character F can just achieve 85.86%.

Experiment No.	1	2	3	4	5
Rate (%)	93.64	93.52	93.56	94.44	94.64
Time (s)	31.16	35.98	31.66	31.11	30.97
Experiment No.	6	7	8	9	10
Rate (%)	94.00	94.40	93.56	93.72	94.40
Time (s)	32.82	33.12	35.45	32.92	34.51

Table I
SUCCESSFUL CLASSIFICATION RATE AND TRAINING TIME

Character	A	C	D	E	F
Rate (%)	88.04	98.64	97.88	95.72	85.86
Character	G	H	L	P	R
Rate (%)	95.80	92.04	99.72	91.24	95.12

Table II
SUCCESSFUL RATE FOR EACH CHARACTER USING TRAINED NEURAL NETWORK

No. of hidden units	4	10	15	20
Rate (%)	89.98	93.99	94.18	93.77
Time (s)	44.85	32.97	44.58	55.26

Table III
EFFECT OF HIDDEN UNITS NUMBER IN TERMS OF SUCCESSFUL RATE AND TRAINING TIME

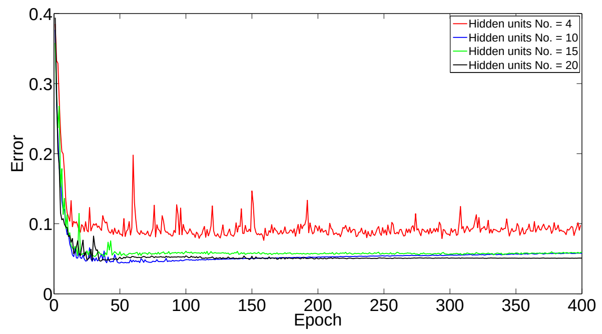

Figure 4. Effect of hidden units number in terms of square error

IV. Discussion

A. *Effect of hidden units number*

Number of hidden units affect neural network performance greatly in terms of degree of freedom a neural network has. Hidden units represent input of neural network by sequence coding. For example, 10 inputs can be represented by at least 4 digits because $2^3 < 10 < 2^4$. But coding with 4 digits also means complex coding rule, which may lead to longer converging time. At very beginning of the experiments, only 4 hidden nodes are used in the hidden layer, and is proved to take relatively long training time. But too many hidden units will not help improve successful identifying rate. We do experiments with 4, 10, 15, and 20 hidden units.

From Table 3, we can see that large hidden units number leads to long training time, but did not improve identifying performance significantly. Small hidden units number also extend the training period, and degrade successful rate as well. 4 hidden units takes more than 70 iterations to converge, while the other situations having less than 50 iterations.

As Figure 4 shows, low hidden units number results in higher error (around 0.08), so that when there are only 4 hidden nodes, the successful identifying rate is lower than 10 hidden units. This is caused by the lose of degree of freedom of small hidden node numbers. Any inaccurate representing can cause error increasing. Large number of nodes enable better error tolerance. But increasing hidden nodes did not improve performance significantly, 15 and 20 hidden units have very similar rate as 10 hidden units but takes more time to train. It is easy to understand that, more hidden units need more weights in between each two layers. So updating each weights will take more time. As a result, 10 -15 hidden units may have both good performance and good converging time.

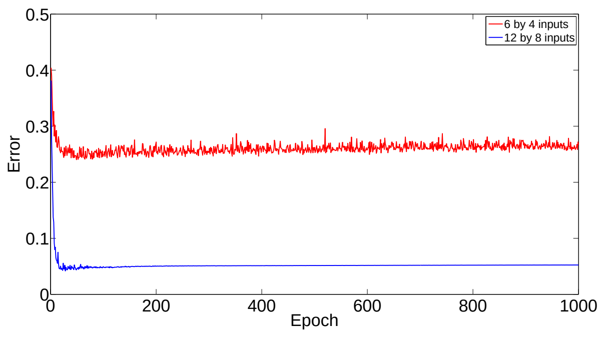

Figure 5. Effect of different input generalizations

Learning rate	0.01	0.05	0.1	0.5	0.8
Rate (%)	93.97	93.13	93.99	91.34	90.58
Time (s)	179	32.30	32.97	92.34	141.6

Table IV
EFFECT OF LEARNING RATE IN TERMS OF SUCCESSFUL RATE AND TRAINING TIME

B. *Effect of different input generalizations*

To have a general sense about the effect of different input generalizations, we do experiments with 1000 iterations for 6 by 4 inputs and 12 by 8 inputs separately. Results in Figure 5 shows that lowering input generalizations will do harm to successful rate significantly. In 6 by 4 inputs case, minimal error of validation set is around 0.25, so we set 0.25 as the converge threshold. The successful rate is only 61.48%, and training time is 20.94 seconds, which takes around 30 iterations to validate convergence. From this experiment, we can draw the conclusion that lowering input generalization will lose character graphic details, therefore decrease overall network classification performance, but can improve training time.

C. *Effect of learning rate*

Learning rate represents learning step size, so that large learning rate implys large delta weights. Typically we choose 0.1 as the default value. And we try learning value range from 0.05, 0.1, 0.2, 0.5 and 1.0 to evaluate its influence in classification rate and training time. After run this experiment for several times, we find that the minimal error of validation set is larger than 0.05, so 0.05 cannot guarantee convergence. In this case, we use 0.07 as the new threshold. For leaning rate of 0.8, threshold is 0.08 to fulfill convergence requirement, this threshold is also obtained by pre-run the experiment for a very large iteration (more than 10000 times).

As Table IV shows, smaller learning rate 0.05 has very similar performance with 0.1, but large learning rate 0.5 and 1.0 will increase training time significantly and decrease successful classification rate. Intuitively, smaller learning rate should achieve the same or a little better successful rate as 0.1, but takes longer time (more than 100 iterations) to converge since each update step is small. Larger learning rate may

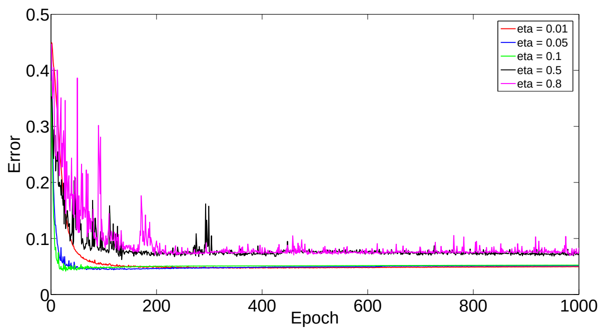

Figure 6. Effect of learning rate in terms of square error

result in coarse-grained weight updating and taking much longer time to converge. The curve in Figure 6 shows heavy fluctuation, which is caused by large learning rate. When eta equals 0.01, 0.05 and 0.1, the minimal error values are very close, so that in Table 4, the successful classification rates of those three are also close. But curves representing 0.5 and 0.8 eta values are much higher, which mean the minimal errors are larger. Global optimal solution is escaped from large scale weights updating. So that identifying rates are lower, which is also told by Table 4. Therefore, 0.1 is a proper rate for this data sets and neural network training.

D. *k-fold cross validation*

In addition to training and validating with given sets, we also do k-fold data sets experiments. In k-fold approach, each character has 100 samples from learning set and 250 samples from validating set, so it is 350 samples for each character. Those 350 samples are equally and randomly divided to 10 folds. At every iteration, 9 folds are picked as training sets and the left one is validating set. Ten iterations compose one epoch, in which each fold will play the role of validating set for one time and training set for nine times. Then we use the average validating error to decide converging point. After stopping training, all data are used as training samples for the last time weights updating. According to the experiments, k-fold approach has successful rate as high as 96.62%. Because one epoch in k-fold actually trained for 10 times, it is not reasonable to compare its epoch-error curve with the regular method. After running this experiment for 5 times, we use the mean value as the final results for k-fold method, as Figure 7 shows. The reason for better performance of k-fold is our training set are so small, that regular method only use 100 for training and 250 for validating, but k-fold can use 315 samples for training and 35 for validating. More samples in training means more information are obtained by neural network. At the mean time, training / validating sets rotation avoid over fit problem since at each iteration different training / validating data are used. Although k-fold method takes longer training time, it is worthy the successful classification rate.

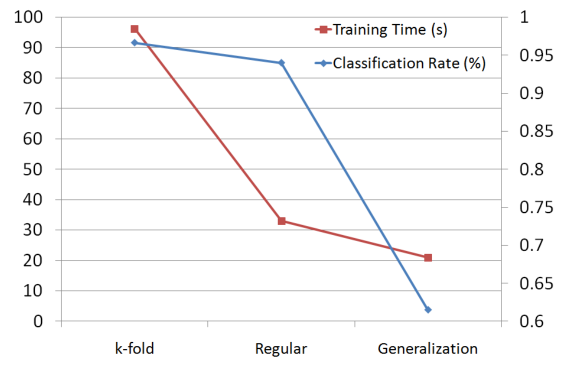

Figure 7. Classification rate and training time of k-fold and regular approach

V. Conclusion

In this project, an feedforward backpropagation algorithm artificial neural network is implemented to identify 10 uppercase characters. Three separate data sets are used as training, validating, and test data. As discussed above, successful classification rate is about 94%, and by using k-fold approach it can reach as high as 96.7%. Overfitting will make classification rate decrease, as we have discussed in Section 3.1.

Additional, several phenomenon are observed in the experiments. Firstly, choosing proper hidden units number is very important in network training. At the very beginning, only 4 hidden units are used, and the converging time is quite long, with low successful rate. The reason is too little hidden nodes lose degrees of freedom in representing the relation between inputs and outputs. However, increasing hidden node number does not always improve identifying accuracy and on the other hand, may take longer time to converge. After trying for several times, we found 10 hidden nodes can represent the 10 characters in reasonable time and successful rate. So properly selecting number of hidden nodes is important for neural network design. Secondly, for speed up the converging process, enlarge η value properly will make converging period shorter, which facilitate observing general converging threshold. But too large η will extend training time and skip global optimal value. Thirdly, because fluctuations in different network environment are quite different, and sometimes are very large at the beginning of training periods, it is hard to determine when to stop training. So that we run each experiment for 1000 iterations and plot all the error data to have a general sense about the error value, then set an approximate minimal value as converging threshold. This method can guarantee an acceptable classification performance, but is a little time consuming.

At last, k-fold method is regarded as an improvement of regular neural network. Because our data sets are small, lacking of enough data makes the input-output relation cannot be fully represented. With k-fold approach, we can utilize more data for training, so that more relation information are grasped by neural network, then higher identifying rate is achieved.

References

[1] Tom Mitchell, McGraw Hill, *Machine Learning*. Mcgraw-Hill International Edit, 1997

[2] Lynne. E. Parker, *Handout: Notes on Neural Networks*. http://web.eecs.utk.edu/~parker/Courses/CS425-528-fall10/Handouts/Neural-net-notes.pdf.