成绩

北京航空航天大學
BEIHANG UNIVERSITY

Experiments for “Pattern Recognition and Machine Learning”

Experiment 4

Neural Networks and Back Propagation

院（系）名称 自动化科学与电气工程学院

专 业 名 称 自动化

学 生 学 号 15071135

学 生 姓 名 刘雨鑫

2018 年 5 月 21 日

4.1 Introduction

The learning model of Artificial Neural Networks (ANN) (or just a neural network(NN)) is an approach inspired by biological neural systems that perform extraordinarily complex computations in the real world without recourse to explicit quantitative operations. The original inspiration for the technique was from examination of bioelectrical networks in the brain formed by neurons and their synapses. In a neural network model, simple nodes (called variously "neurons" or "units") are connected together to form a network of nodes, hence the term "neural network".

Each node has a set of input lines which are analogous to input synapses in a biological neuron. Each node also has an "activation function" that tells the node when to fire, similar to a biological neuron. In its simplest form, this activation function can just be to generate a '1' if the summed input is greater than some value, or a '0' otherwise. Activation functions, however, do not have to be this simple - in fact to create networks that can do useful things, they almost always have to be more complex, for at least some of the nodes in the network. Typically there are at least three layers to a feed-forward network - an input layer, a hidden layer, and an output layer. The input layer does no processing - it is simply where the data vector is fed into the network. The input layer then feeds into the hidden layer. The hidden layer, in turn, feeds into the output layer. The actual processing in the network occurs in the nodes of the hidden layer and the output layer.

4.2 Principle and Theory

The goal of any supervised learning algorithm is to find a function that best maps a set of inputs to its correct output. An example would be a simple classification task, where the input is an image of an animal, and the correct output would be the name of the animal. For an intuitive example, the first layer of a Neural Network may be responsible for learning the orientations of lines using the inputs from the individual pixels in the image. The second layer may combine the features learned in the first layer and learn to identify simple shapes such as circles. Each higher layer learns more and more abstract features such as those mentioned above that can be used to classify the image. Each layer finds patterns in the layer below it and it is this ability to create internal representations that are independent of outside input that gives multi-layered networks their power. The goal and motivation for developing the back-propagation algorithm was to find a way to train a multi-layered neural network such that it can learn the appropriate internal representations to allow it to learn any arbitrary mapping of input to output.

Mathematically, a neuron's network function $f(x)$ is defined as a composition of other functions $g_i(x)$ which can further be defined as a composition of other functions. This can be conveniently represented as a network structure, with arrows depicting the dependencies between variables. A widely used type of composition is the nonlinear weighted sum, where:

$$f(x) = (\sum_i w_i g_i(x))$$

where K (commonly referred to as the activation function) is some predefined function, such as the hyperbolic tangent. It will be convenient for the following to refer to a collection of

functions g_i as simply a vector $g = (g_1, g_2, ..., g_n)$ .Back-propagation requires a known, desired output for each input value in order to calculate the loss function gradient. It is therefore usually considered to be a supervised learning method.

The squared error function is:

$$E = \frac{1}{2}(t - y)^2$$

Where E is the squared error, t is the target output for a training sample, and y is the actual output of the output neuron. For each neuron j, its output o_j is defined as

$$o_j = \varphi(net_j) = \varphi(\sum_{k=1}^{n} w_{kj} x_k)$$

The input net to a neuron is the weighted sum of outputs o_k of previous neurons. If the neuron is in the first layer after the input layer, the o_k of the input layer are simply the inputs x_k to the network. The number of input units to the neuron is n. The variable w_{ij} denotes the weight between neurons i and j.

The activation function φ is in general non-linear and differentiable. A commonly used activation function is the logistic function, e.g.:

$$\varphi(z) = \frac{1}{1 + e^{-z}}$$

which has a nice derivative of:

$$\frac{\partial \varphi}{\partial z} = \varphi(1 - \varphi)$$

Calculating the partial derivative of the error with respect to a weight w_{ij} is done using the chain rule twice:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$$

We can finally yield:

$$\frac{\partial E}{\partial w_{ij}} = \delta_j x_i$$

with

$$\delta_j = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} = \begin{cases} (o_j - t_i)\varphi(net_j)(1 - \varphi(net_j)) & \text{if } j \text{ is an output neuron} \\ (\sum_{l \in L} \delta_l w_{jl})\varphi(net_j)(1 - \varphi(net_j)) & \text{if } j \text{ is an inner neuron} \end{cases}$$

4.3 Objective

The goals of the experiment are as follows:

(1) To understand how to build a neural network for a classification problem.

(2) To understand how the back-propagation algorithm is used for training a given a neural network.

(3) To understand the limitation of the neural network model (e.g., the local minimum).

(4) To understand how to use back-propagation in Autoencoder.

4.4 Contents and Procedure

Stage 1 ：

（1）build a neural network(NN) and train the network using the BP algorithm.

First of all, I get the data named breast cancer Wisconsin from the UCI ML Repository. Each sample has ten attributes and one class attribute. The attribution is shown in the table 1 (class attribute has been moved to last column).

Table 1 Attribute Information

#	Attribute	domain
1	Sample code number	id number
2	Clump Thickness	1 - 10
3	Uniformity of Cell Size	1 - 10
4	Uniformity of Cell Shape	1 - 10
5	Marginal Adhesion	1 - 10
6	Single Epithelial Cell Size	1 - 10
7	Bare Nuclei	1 - 10
8	Bland Chromatin	1 - 10
9	Normal Nucleoli	1 - 10
10	Mitoses	1 - 10
11	Class	2 for benign, 4 for malignant

Then I use the data without the attribute named Sample code number. Because of the activation function φ I chose and the domain of the data, I take the data multiplied by 0.1 as the pre-processing data, which ranges from 0.1 to 1.

Next I write the code BP(q,h).m with one hidden layer, which needs to be input the amount of neurons in the hidden layer q and the learning rate h, and outputs the weight matrix. After finishing the code, the net is like the fig.1.

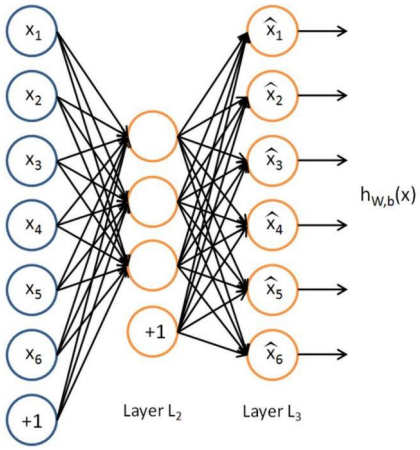

Fig.1 General structure of the net

I run it with 439 samples pre-processed and then get the weights. Then I write the text code BP.m, run it with 244 samples pre-processed and the results are as follows.

Fig.2 shows the variance of the sample text, the maximum variance is below 0.008. Fig.3 shows the figure difference between the target output and the actual output. If we see that it is classed correctly when the variance is less than 0.0025, the number of erroneous classification is 5, and the proportion of erroneous classification shown in the fig.3 is 0.016393. So we can see that most of the data are correctly classified.

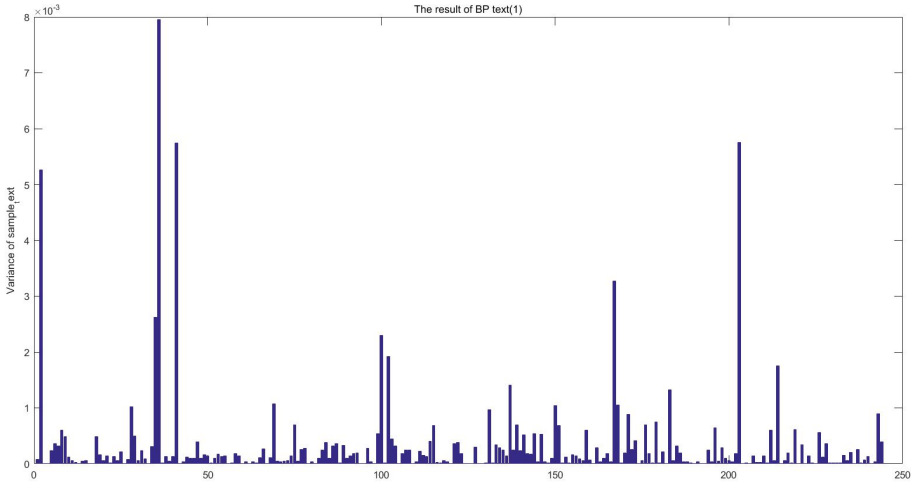

Fig.2 Variance of sample text

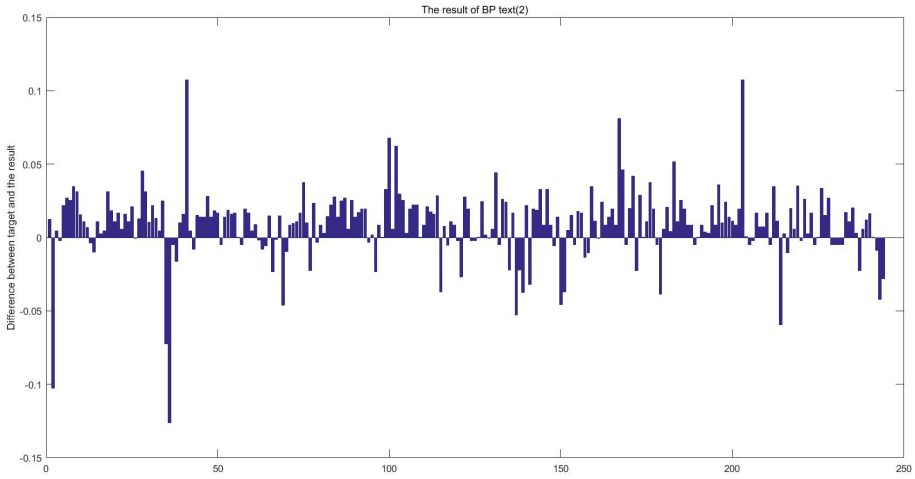

Fig.3 Difference between the target output and the actual output

(2) Compare the NN results to the results of Perceptron

Then I use the samples used in the experiment one for Linear Classifier, which has two attributes, and one class attribute, and compare the NN results to the results of Perceptron. I find that the number of iterations of the NN results is much less than Perceptron. For the data which has 200 samples, the number of iterations of the NN is about 3600, while the number of iterations of the Perception is about 8600, when both of them use the same learning rate. But the accuracy of the NN is lower than the Perceptron. I think if the NN model has more hidden layers, and more training samples, the accuracy of the result may be promoted.

(3) Whether the performance of the model is heavily influenced by different parameters settings

I set the q, the amount of neurons in the hidden layer, at 20, and assign 0.1, 0.5, 1, 1.5, 2 to the learning rate in turn and the results are shown in table 1. The iterations shown in table 2 is 1/10000 of the real.

Table 2 results of different learning rate h

Learning rate	0.1	0.5	1	1.5	2
iterations	40.46	8.77	4.97	4.51	55.31

From the table above, we can see the iterations decreases first and then increases as the learning increases. The smaller the learning rate, the greater the number of iterations. And the number of erroneous classification ranges from 2 to 4.

Then I set the learning rate h at 0.8, and assign 5, 10, 15, 20 to the amount of neurons in the hidden layer in turn and the results are shown in table 3. The iterations shown in table 3 is 1/10000 of the real.

Table 3 results of different amount of neurons in the hidden layer q

the number of nodes	5	10	15	20
iterations	3.97	4.94	5.58	6.59

From the table above, we can see that the iterations increases as the amount of neurons increases, and And the number of erroneous classification ranges from 3 to 4.

Stage 2 ：

(1) How the efficiency and accuracy will be influenced by different activation functions and more hidden layers.

In my opinion, the overabundance of the hidden layer not only increases the training time, but also leads to the problem that it requires long learning time, and the relative error is not the best, and the fault tolerance is poor and may cause the overfitting. If it has few hidden layers, the result error may be large. So the it requires the proper number of hidden layers.

From the ability to express the network, no activation function can make the network only a linear model, and the ability is limited, so it is necessary to make the network have a strong ability to express the function as a nonlinear function. And in the process of backpropagation, the gradient updated by the previous parameter is generally the product of the derivative, which also contains the derivative of the activation function. If the derivative value such as thanh or sigmod is related to the value of the function, the derivative product will also have the value of the activation function. So we also hope that the range of the activation function and the derivative of the activation function can meet certain requirements, if not too large, it will lead to the gradient explosion. Must not be too small, will cause the gradient dispersion.

(2) Biological neural networks work in the same way as our NN model?

I think that the biological neural network is conceptually similar, or the same as the NN model. The idea of "calculating nonlinear functions" is no different from the idea of the nature of artificial neural networks: when electrical signals reach the end of an axon of a neuron, they excite the cell membrane of the axon to release neurotransmitters. These transmitters react with the receptor on the dendrites or on the surface of the cell membrane of the receptor neurons and stimulate / inhibit the release of the receptor neurons (producing pulse signals). The stimuli or restraints in this process can correspond to the positive and negative weights in artificial neural networks, while sigmoid, the activation function, is only an oversimplification of this nonlinear process in human (biological) neurons.

(3) Using the BP algorithm to train an autoencoder.

I get the same data named breast cancer Wisconsin from the UCI ML Repository, and use the BP algorithm to train an autoencoder. So the inputs x is ten dimensional, and the output y is also ten dimensional.

Because of the activation function φ we choose and the domain of the data, I take the data multiplied by 0.1 as the pre-processing data, which ranges from 0.1 to 1.

Next I write the code BP_autoencoder(q,h,e).m with one hidden layer, which needs to be input the amount of neurons in the hidden layer q, the learning rate h and the condition of convergence e, and outputs the weight matrix. After finishing the code, I set q at 8, h at 0.5, e at 0.0025, then run it with 440 samples pre-processed and then get the weights. Then I write the text code BP_autoencoder_text.m, run it with 240 samples pre-processed and the results are as follows.

Fig.4 shows the MSE(mean square error) of each sample. We see that it's correct if the mean square error is less than 0.0025, the amount of correct samples is 174, the proportion of correct results is 0.725. If the amount of training samples gets larger, I think the accuracy will improve.

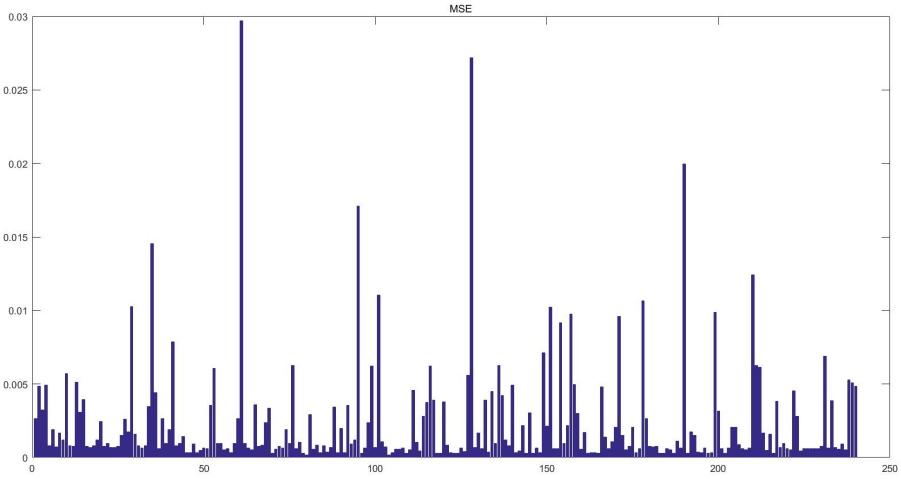

Fig.4 mean square error of the autoencoder results

(4) How the number of nodes in the hidden layer influence the model performance?

I set the learning rate h at 0.5, the condition of convergence e at 0.0025, assign 1, 3, 5, 7, 9, 11, 13, 15 to the number of nodes in turn, and the results are shown in table 4.

Table 4 results of different number of nodes q

the number of nodes	1	3	5	7	9	11	13	15
Proportion of correct	0.038	0.425	0.521	0.671	0.738	0.738	0.671	0.508

From the table above, we can see the proportion of correct increases first and then decreases as the number of nodes increases. The boundary value is about 10, which is the same as the number of nodes in the hidden layer.

4.5 Experience

Through this experiment, I have better understood the conception of Neural Networks and the principle of back-propagation algorithm, and try to program the code to achieve the function. This experience is just a step for me to get familiar to the Neural Networks, there are lots of works need to do if I want to work it out. What's more important is that my interest and enthusiasm for machine learning has much increased and I want to learn it more deeply from my heart.

4.6 Code

The code can be download at

https://github.com/Jackboomer/Experiments-for-Pattern-Recognition-and-Machine-Learning.git