Pattern Recognition and Machine Learning

Experiment Report

院（系）名称	自动化科学与电气工程学院
专 业 名 称	模式识别与智能系统
学 生 姓 名	邓颖
学 号	12031156
任 课 老 师	秦曾昌

2015 年 6 月 14 日

Neural Networks and Back Propagation

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.

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.

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:

$$\mathrm{f(x)} = \sum_i w_i g_i(x) \tag{1}$$

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 $g = (g_1, g_2, \cdots g_n)$. 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:

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

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 *oj* is defined as

$$o_j = \varphi(net_j) = \varphi\left(\sum_{k=1}^{n} w_{kj} x_k\right) \tag{3}$$

The input net to a neuron is the weighted sum of outputs *ok* 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.:

$$\varphi(\mathrm{z}) = \frac{1}{1+e^{-z}} \tag{4}$$

which has a nice derivative of:

$$\frac{\partial\varphi}{\partial z} = \varphi(1-\varphi) \tag{5}$$

Calculating the partial derivative of the error with respect to a weight *wij* is done using the chain rule twice:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}} \tag{6}$$

We can finally yield:

$$\frac{\partial}{\partial w_{ij}} = \delta_j x_i \tag{7}$$

with

$$\delta_j = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j} = \begin{cases} (o_j - t_j)\varphi(net_j)\left(1-\varphi(net_j)\right) & if\ j\ is\ an\ output\ neuron \\ \sum_{l\in L}\delta_l w_{jl}\varphi(net_j)\left(1-\varphi(net_j)\right) & if\ j\ is\ an\ inner\ neuron \end{cases} \tag{8}$$

3. Objective

Stage 1:

(1) Utilize the Iris dataset to build a neural network that clusters iris flowers into three natural classes, so that similar classes are grouped together. Each iris is described by four features: sepal length, sepal width, petal length and petal width. The four flower attributes will act as inputs to the NN with one hidden layer, which will map them onto three outputs.

Set the parameters as follows:

learning rate: 0.1　　largest times of training:2000　　error threshold:0.1

then run the program codes to train the neural network, and the get the result:

correct ratio of the test data: 97.33%

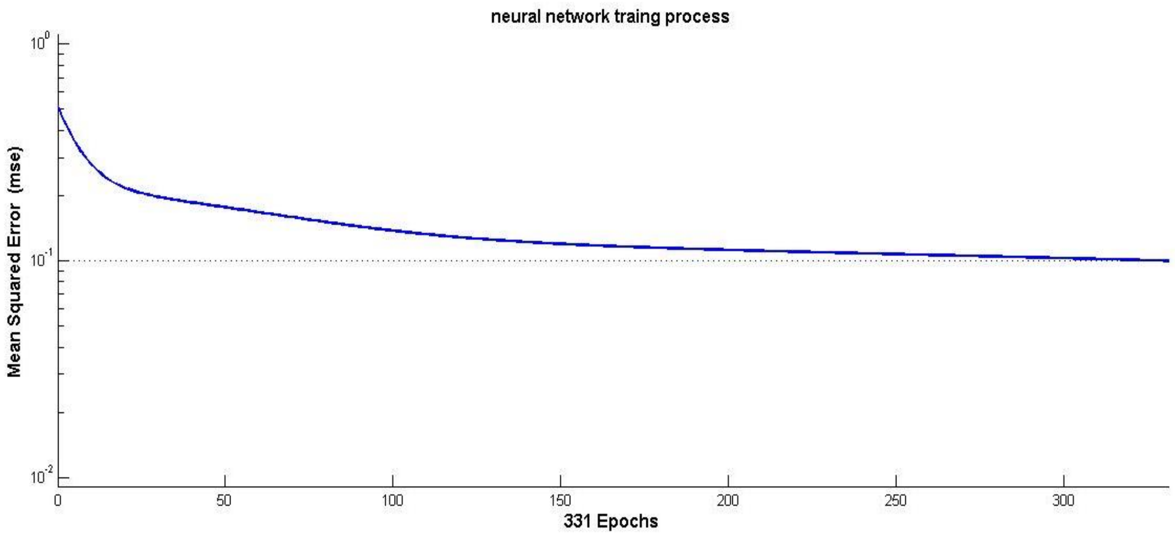


Figure 1 Neural network process

(2) The result of the perceptron model: 721 epochs, correct ratio: 80.67%

By comparing the results of the two models, we can come to the conclusion that the NN is superior to the Perceptron in both efficiency and accuracy.

Analysis:

The perceptron is an algorithm for binary classifiers, with the function to decide whether an input belongs to one class or another. Hence I have to execute the perceptron twice to classify the iris to three classes. While the NN is much faster as it enables multiple outputs, which improves efficiency markedly. Besides, perceptron is a type of linear classifier. But NN, which in fact is a multilayer perceptron, works well in linear, non-linear and more complicated problems, hence ensures better accuracy.

(3) The influence different parameters make to the performance of the model:

1) The original weight:

Change the variable repeatedly to see the results. The neural networks all converge and the training time changes irregularly.

So the original weight has little to do with the performance of neural network.

2) Learning rate

Invariant parameters: weight--zero matrix, hidden layer--5 nodes

α	0.1	0.5	0.9	1	1.5	2	3	4
best epoch	331	259	107	60	67	56	31	diverge
correct ratio	96.67%	97.33%	92.00%	95.33%	97.00%	92.33%	90.00%	

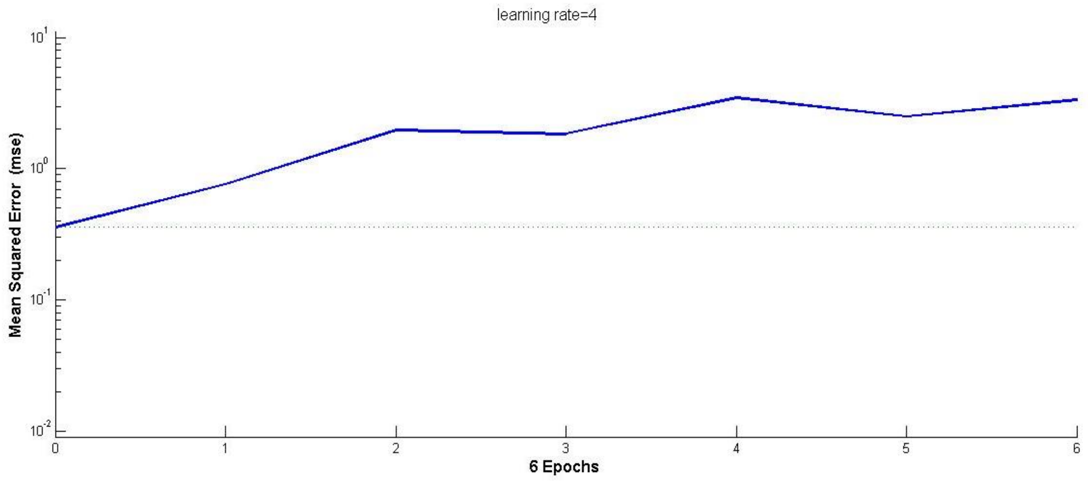


Figure 2 Neural network when learning rate is 4

As the learning rate grows larger, the speed of the training processing accelerates but the quality of the classification declines. Once the learning rate becomes too large the neural network will diverge.

3) Number of the nodes in hidden layer

Invariant parameters: weight—zero matrix, learning rate—1

nodes	3	5	7	9	11
best epoch	125	101	123	108	112
correct ratio	98%	92%	97.33%	98%	97%

Number of nodes in hidden layer seems to have no direct influence on neural network. But too much nodes will increase the amount of calculation.

4) Activation functions
Invariant parameters:
weight--zero matrix, hidden layer--5 nodes, learning rate-1

function	purelin	logsig	tansig
best epoch	103	106	86
correct ratio	89%	98%	98%

As is shown above, the linear function generates the lowest correct ratio, and the tansig function is the best with high efficiency and accuracy.

Stage 2:

(1) Once the structure of the network and training sample are settled, the mean square error is mainly determined by the activation function, thus influence the results. Specific problems call for a set of specific activation functions because of their different characters. But generally, the sigma function is widely used as it can produce output values in the range of [-1,1] and minimize the computation capacity for training. It provides good efficiency and accuracy through the contrast in stage 1.

The more complicated a problem is, the more hidden layers it needs. And within certain limits growing number of layers often makes the network easier to converge and raise the accuracy. But too much layers will augment the burden of calculation and lead to over learning. So it`s better to find the appropriate number of hidden layers by several tests.

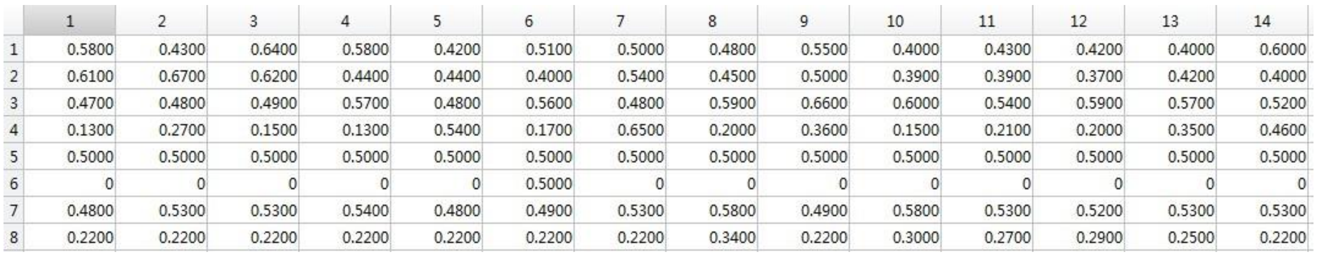
(2) The neural networks work quite similar to our NN model in some ways:

i. They are made up of many very simple units, each having a local memory to store and manage information.

ii. The units are connected by unidirectional channels which carry numeric data and each unit operates only on their local data and the inputs through the channels.

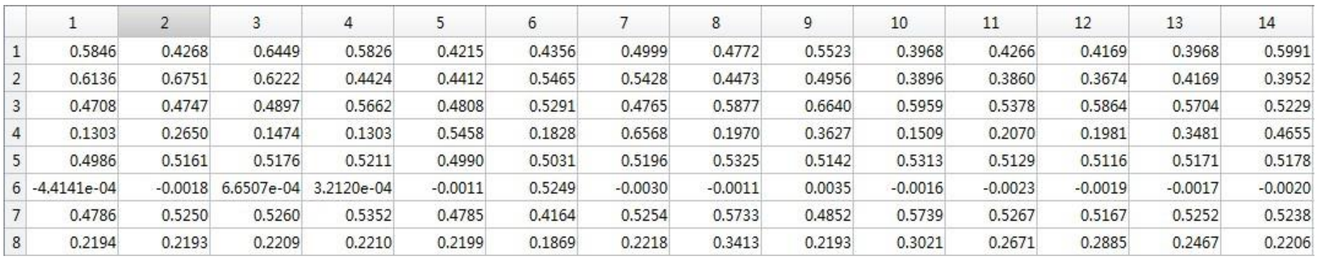
iii. They all have some sort of feedback regulation. The neural networks receive the feedback adjustment from all kinds of hormone, while the NN model have backpropagation algorithm.

(3) For this experiment I choose an 8-dimension dataset, thus the input layer as well as the output one of this antoencoder has 8 nodes. Construct an antoencoder with only one hidden layer of 4 nodes. Compile the program, a part of the input and output data are as follows.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0.5800	0.4300	0.6400	0.5800	0.4200	0.5100	0.5000	0.4800	0.5500	0.4000	0.4300	0.4200	0.4000	0.6000
2	0.6100	0.6700	0.6200	0.4400	0.4400	0.4000	0.5400	0.4500	0.5000	0.3900	0.3900	0.3700	0.4200	0.4000
3	0.4700	0.4800	0.4900	0.5700	0.4800	0.5600	0.4800	0.5900	0.6600	0.6000	0.5400	0.5900	0.5700	0.5200
4	0.1300	0.2700	0.1500	0.1300	0.5400	0.1700	0.6500	0.2000	0.3600	0.1500	0.2100	0.2000	0.3500	0.4600
5	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000
6	0	0	0	0	0	0.5000	0	0	0	0	0	0	0	0
7	0.4800	0.5300	0.5300	0.5400	0.4800	0.4900	0.5300	0.5800	0.4900	0.5800	0.5300	0.5200	0.5300	0.5300
8	0.2200	0.2200	0.2200	0.2200	0.2200	0.2200	0.2200	0.3400	0.2200	0.3000	0.2700	0.2900	0.2500	0.2200

Figure 3 Input data



	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0.5846	0.4268	0.6449	0.5826	0.4215	0.4356	0.4999	0.4772	0.5523	0.3968	0.4266	0.4169	0.3968	0.5991
2	0.6136	0.6751	0.6222	0.4424	0.4412	0.5465	0.5428	0.4473	0.4956	0.3896	0.3860	0.3674	0.4169	0.3952
3	0.4708	0.4747	0.4897	0.5662	0.4808	0.5291	0.4765	0.5877	0.6640	0.5959	0.5378	0.5864	0.5704	0.5229
4	0.1303	0.2650	0.1474	0.1303	0.5458	0.1828	0.6568	0.1970	0.3627	0.1509	0.2070	0.1981	0.3481	0.4655
5	0.4986	0.5161	0.5176	0.5211	0.4990	0.5031	0.5196	0.5325	0.5142	0.5313	0.5129	0.5116	0.5171	0.5178
6	-4.4141e-04	-0.0018	6.6507e-04	3.2120e-04	-0.0011	0.5249	-0.0030	-0.0011	0.0035	-0.0016	-0.0023	-0.0019	-0.0017	-0.0020
7	0.4786	0.5250	0.5260	0.5352	0.4785	0.4164	0.5254	0.5733	0.4852	0.5739	0.5267	0.5167	0.5252	0.5238
8	0.2194	0.2193	0.2209	0.2210	0.2199	0.1869	0.2218	0.3413	0.2193	0.3021	0.2671	0.2885	0.2467	0.2206

Figure 4 Output data

(4) Similar to the NN model, reduction of the number of nodes can increase computational speeds, but lead to the data loss in some degree. Enough nodes are beneficial to the consistency of input and output data.

4. Experiences

Through this experiment I get a deeper hold of the theory and implementation of the NN model, at the same time make a contrast with biological neural networks. In the process of experiment, I witnessed the change that various parameters do on the results and try to explain the phenomena. Those help get my thought clear and make me really appreciate the efficiency of NN model. But I also realize the defects of this model, such as the lack of methods to decide the proper number of hidden layers. All in all, I get a better knowledge of neural network from this experiment.