Lab Session on Feedforward Neural Networks

Temujin Gautama & Karl Pauwels

October 30, 2003

1 Introduction

The three-hours lab session on feedforward neural networks is an AE (7.1) about feedforward neural networks which takes about three hours. The Matlab-code for this AE is written in Matlab and is available from `http://134.58.34.50/lab_session/ernie.tar.gz`. For the poor brave souls among you who want to become ANN experts in three hours, there is an alternative version of this AE, which will be made available in approximately April 2, 2009. Until this date, they will have to make do with this humble effort, which tries to familiarise you with the common pitfalls of basic feedforward neural networks.

You don't need to hand in a report at the end of the AE, since it is not mandatory to attend anyway. So, yes, if you want to leave now to go home and do whatever, please feel free to go home and do whatever. You might even want to do this AE at home on your own, but I would like to note that in that case, possible questions that might arise, will only be answered by your fellow students (try to locate those that wanted to become ANN experts in three hours). You can find a copy of this text (in a more practical format for printing) here `http://134.58.34.50/lab_session/lab_text.pdf`.

Anyway, what are we going to do? There are two basic applications of feedforward neural networks that are included in this AE (7.1). namely regression and classification. For each of these, there are several data sets to try out using all sorts of architectures of EEEs (7.1). Furthermore, several learning schemes are provided, so that you can see which one works where and when and hopefully you'll know why.

2 Where Is the Start Button?

Ernie doesn't *have* a start button. In fact, Ernie doesn't, as one would intuitively think, *start* at all. But, most importantly, Ernie isn't self-contained and needs Matlab. So, open up a matlab window, and you might want to download Ernie, who is proud to be among the selected few EEEs (7.1) who is publicly available from `http://134.58.34.50/lab_session/ernie.tar.gz`. Unzip him (GENTLY) to whatever directory and cd to that directory from the Matlab window.

Now, to start up Ernie, all you have to do is type "`please_go`", and he'll show up and off, ready for whichever of the two applications you want him to do.

3 How the Dodo Got its Name

This section will get you all set to start the simulations. It introduces Ernie, the friendly neural network, and shows you how to interact with him. Basically, Ernie, as was his father and his father's father before him, is a pure-blood EEE (7.1). Contrary to his ancestors (which were also Matlab-generated, but were coded in a fairly ad hoc manner), Ernie can have as many hidden layers as he wants with however many neurons per layer. Although people have told him that more than two hidden layers makes him unnecessary slow and sluggish, he sometimes experiments

with this idea if only to establish his true identity (and especially, and more importantly, only when nobody else is looking).

So, when configuring the EEE (7.1) architecture, *e.g.*, when starting a new training session, you'll see something like this:

```
Set number of neurons in layer to zero => layer = output
Number of neurons in hidden layer 1 [0]:
```

Evidently, you have to input the number of neurons in hidden layer 1. If it is set to 0 (which is the default value between square brackets), this layer is used as the output layer. If a positive number is given, the number of neurons of the next hidden layer will be asked. If a positive number is given, the number of neurons of the next hidden layer will be asked. If a positive number is given, the number of neurons of the next hidden layer will be asked. And this until the end of times, or until you just press enter or zero. The number of neurons in input and output layer are always defined by the application at hand. All neurons have the usual sigmoid activation function.

Next, you will be asked for the type of learning algorithm:

```
Learning algorithm:
        1. Backpropagation
        2. Backpropagation with momentum
```

Uh-uh! Pitfall number one. One of the most important reasons why EEEs (7.1) such as Ernie fail to do the job, is ignorance. Yes, I know, you are studying the Art of Artificial Intelligence, but one of the main lessons to be learned about English-speaking EEEs is that "ils ne sont pas vraiment intelligents", which I will quickly translate for Ernie: that "they are very very intelligent and sensible creatures". So, anyway, if you don't know what either of the two choices mean, maybe it is time to open your course notes (you might also want to check out section 7.2).

Okay, next a number of learning parameters:

```
Number of epochs to train [10]:
```

One epoch means providing all data points once. Default values are shown between square brackets and can always be selected by just pressing enter. Basically, I have included these as practical guidelines on how not to choose your parameters.

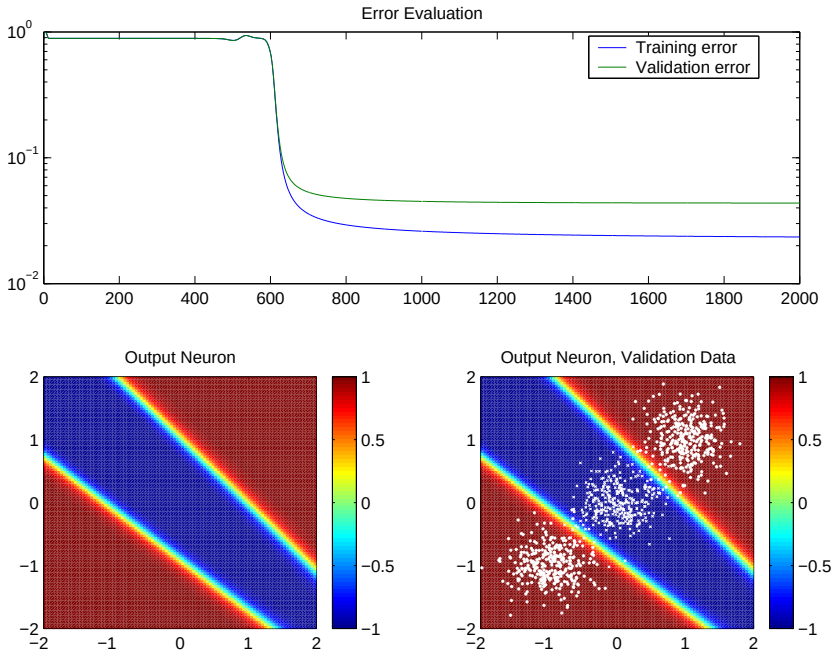
```
Learning rate [.2]:
```

The learning rate determines the step size when doing gradient descent (7.2).

```
Stop criterion:
        1. No stop criterion
        2. Increase in validation error (early stopping)
```

Hmmm, stop criterion. Validation error, cross-validation, early stopping ... confusing isn't it. Check the course notes or look here (7.3). If you have, I'll just say that the test or validation set contain the same number of data points as the training set. Maybe you'll see that cross-validation doesn't always get you the results you want.

After training, you'll see Ernie's interpretation of the world and he'll provide you with a number of pictures taken during his long hours of study (training). It might look something like this: Top figure is the training and validation error during training. Basically, Ernie and you want these curves to go down as quickly and deeply as possible. Bottom left (or sometimes just the bottom pic) is a picture of what Ernie does when you pinch him in certain places: the EEE (7.1) output. Bottom right picture shows the same thing, but also visualises the validation set (labelled "+" for the data points for which the desired output is +1, and "-" for the points for which the desired output is -1). Finally, Ernie will be honest enough to share his innermost thoughts with you, namely the mean-square-error (MSE) obtained on the training and validation set.



After training, you can always *resume* the previous training session, either with the same or with different parameters. Needless to say that this option can save you **a lot** of time. Furthermore, you can confront Ernie personally (`Query network`) and see what he thinks of a given data point:

```
Enter input data point between square brackets
Input [0 0]:
```

The input data has to be enclosed in square brackets, *e.g.*, `[1 0]` for the input data (1,0). As a result, you will get the EEE output.

But anyway, to come back to the original topic of this section: `http://jls-web.pausd.palo-alto.ca.us/~mwillis/willis202/project/dodo/`.

4 Regression

What is regression? Consider a scalar input (meaning an input consisting of simple numbers), and a scalar output. Supposing there is a relationship (function) between the two, we want Ernie to find and mimic it.

There are a number of different data sets that you should try. Generate some data, decide on the network architecture and train the network (3). You might want to take a look at this (6).

Hmm, take one type of regression problem, and train identical networks for different levels of noise. Do the training and validation errors increase as a function of the noise level? Does this mean that Ernie does a better or worse job?

Can you get some overfitting? No? Try a very complex network on a very simple data set with very few (10?) noisy data points. Try training (for a very, very long time) with and without early stopping (using the resume training with same parameters would be helpful here). Try to figure out what Ernie is doing.

5 Classification

Okay, now. We will just consider classification problems for which the desired output is either +1 or -1 (so there are, let's see, one ... two ... yes, two classes to distinguish between). There are a number of different data sets that you should try, each of which has a different type of classification boundary. You will of course start wondering how you are supposed to know how many neurons and hidden layers you need. Well, there are guidelines to choose the number of layers and the number of neurons per layer. I think you also know where to find those guidelines. Try every data set for different network architectures. Furthermore, when generating the data, you can specify a certain 'overlap' between the two classes. A zero overlap means that the two classes are perfectly separable (not necessarily linearly, mind) and, thus, that there is a hard decision boundary between the two classes. A non-zero overlap means that the data points of the two classes overlap somewhat. Try to find out what happens when there is an overlap between the classes.

Does cross-validation (early stopping) always work? Are you sure? What happens when there is overlap between the classes? Can you still get crisp classification boundaries?

Have a look at the training curves (top figures). What do they tell you about the energy landscape? Check the course notes on gradient descent (or, 7.2). Are there plateaus? Hmm, plateau, momentum, ... Sometimes your network can get really stuck in a wrong solution. Check the section on network saturation (7.4).

A high training error in the case of overlapping classes does not necessarily mean that Ernie is not doing a very good job. Why?

Can you get some overfitting? Try a very complex network on *e.g* the linear boundary thingy with an overlap of 1 with very few (10?) data points.

6 Some Important Thingies to Look At

Always look for the minimal network architecture. But then again, what happens when the networks are too big or too small?

Have you noticed that, when you use no stopping criterion, Ernie starts oscillating from time to time? Any idea why? Any remedies?

What is a plateau? How can you see from the error evaluation graphs that you're probably in a plateau? What can you do to speed up learning in these cases?

For the classification problems, what happens to the EEE (7.1) output at the classification boundaries?

What is the effect of the stopping criterion on Ernie's understanding of the world?

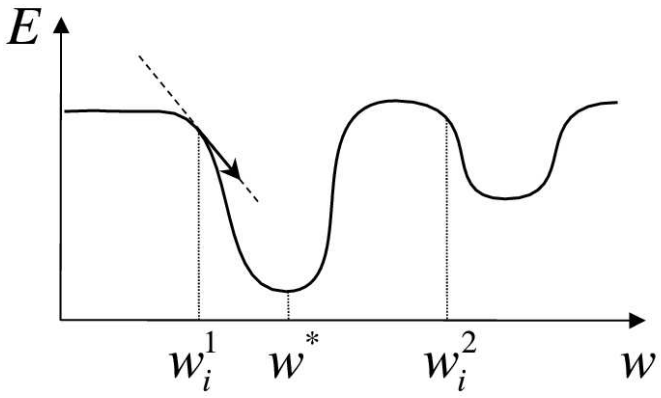
7 The Duh Section

7.1 List of Abbreviations

AE	lAb sEssion
EEE	fEedforward nEural nEtwork
MSE	Mean-Square-Error

7.2 Gradient Descent

Training an EEE involves finding the minimum of a complicated function (called "error function"). This function describes the error the neural network makes in approximating or classifying the training data, as a function of the weights of the network. A simple example of such an error function is shown below. Suppose the initial (randomly chosen) weight(s) are w_i^1. We want the error to become as small as possible and should thus try to move towards w^*. Gradient descent simply means going downhill in small steps until you (hopefully) reach the bottom. This is the



learning technique used in backpropagation. The backpropagation weight update is equal to the slope of the energy function scaled by a *learning rate*, η (thus, the steeper the slope, the bigger the update). To see that this simple procedure doesn't always work, imagine we start learning from w_i^2. If we do gradient descent here we will more than likely end up in a local minimum.

Learning is faster and more stable when you use "backpropagation with momentum". This algorithm moves faster over large plateaus (regions where the error function stays almost constant) and has less problems with oscillations of the MSE during training. So if you use this, learning will be easier.

7.3 Cross Validation

As you probably know by now, EEEs are amazing creatures that can learn to classify or regress very complicated datasets. This can be dangerous sometimes and can result in a problem known far and wide as "overfitting". This happens when your network architecture is too complicated for the problem you're trying to solve. It is then possible that the EEE starts to learn the data by heart. It will give excellent answers to the data used for training, but will respond hopelessly naive when presented with new data, related to the same problem. Here's an example of overfitting (overtraining).
Ernie, A means 1, B means 2, D means 4. A means 1, B means 2, D means 4. A means 1, B means 2, D means 4. A means 1, B means 2, D means 4.
Ernie, A? 1!
Ernie, B? 2!
Ernie, D? 4!
Ernie, C? Ehm, Zimbabwe!
In classication problems, this means that Ernie would start drawing very complex decision boundaries **exactly** around the training set. Thus, data points from the validation set lying just next to those of the training set could result in completely different network outputs.

To counter overfitting, you can use a technique called "early stopping". Here, all the data you have on your problem is split into two groups: a training set and a validation set. In learning the proper weights, the EEE is only allowed to use the training set. After each epoch however, the error is also computed on the validation set. When the network starts memorizing the training data, this will typically result in an increasing error on the validation set (while the training error

keeps on decreasing). If this validation error stays increasing for a while it may be a good idea to stop training (even though the training error is still going down).

7.4 Network Saturation

On many occassions, you will be confronted with a saturated network that is unable to make any more sense (symptoms: bad performance, very spiky training error which doesn't go down as it is supposed to do). You can typically detect this situation in classification problems by looking at the network output: the decision boundaries will be very sharp (sharp transition from red to blue) as compared to the smooth boundaries you observe when the network is still training. If this boundary coincides with the decision boundary you were looking for, this is an ideal situation. However, if it occurs at weird locations, this probably means that Ernie is saturated: the output is either +1 or -1 (the saturated regions of the sigmoid activation function).

When does Ernie saturate? Well, basically, the weights are too big. Just think about it: the weights are too big, so the net activation (weighted sum of inputs) is too big. The net activation is fed into the sigmoids, which for large positive and negative values, yield only +1 and -1.

Why does Ernie saturate? Well, basically, the weight update were too big. Maybe he started with normal (random) weights, but his first update was simply too big. Aha ... if the update is too big, the weights will be completely wrong after the update, so the error is also going to be big next time. Aha, big error, bigger update ... Boom!!!

How can we get Ernie out of saturation? Well, basically, you can't. Remember gradient descent (7.2)? The weight update scales proportionally to the slope of the energy function, right? The slope of the energy function is defined as the change in energy when the weights are changed a tiny little bit. When saturated, changing the weights (and, thus, the net activation which is to be fed into the sigmoid) slightly, doesn't change much. Therefore, the slope and the weight update is going to be veeeery small, and it will take a veeeery long time to get out of this mess.

How can we keep Ernie from saturating? Well, basically, you will have to find that one out for yourself.