*COURSE 02457*

# Non-Linear Signal Processing: Answers to Exercise 6

In this exercise we are going to apply a neural network for binary classification purposes.

### Weight decay controls linearity of output

The network we are training in this assignment has a *tanh* as activation function for the hidden units and a $softmax$ as activation function for the output units. These are both functions that can behave both as linear function and as a step function depending on the size of the argument. The amount of weight decay that controls the size of the weights therefore control the linearity of the output of the network.

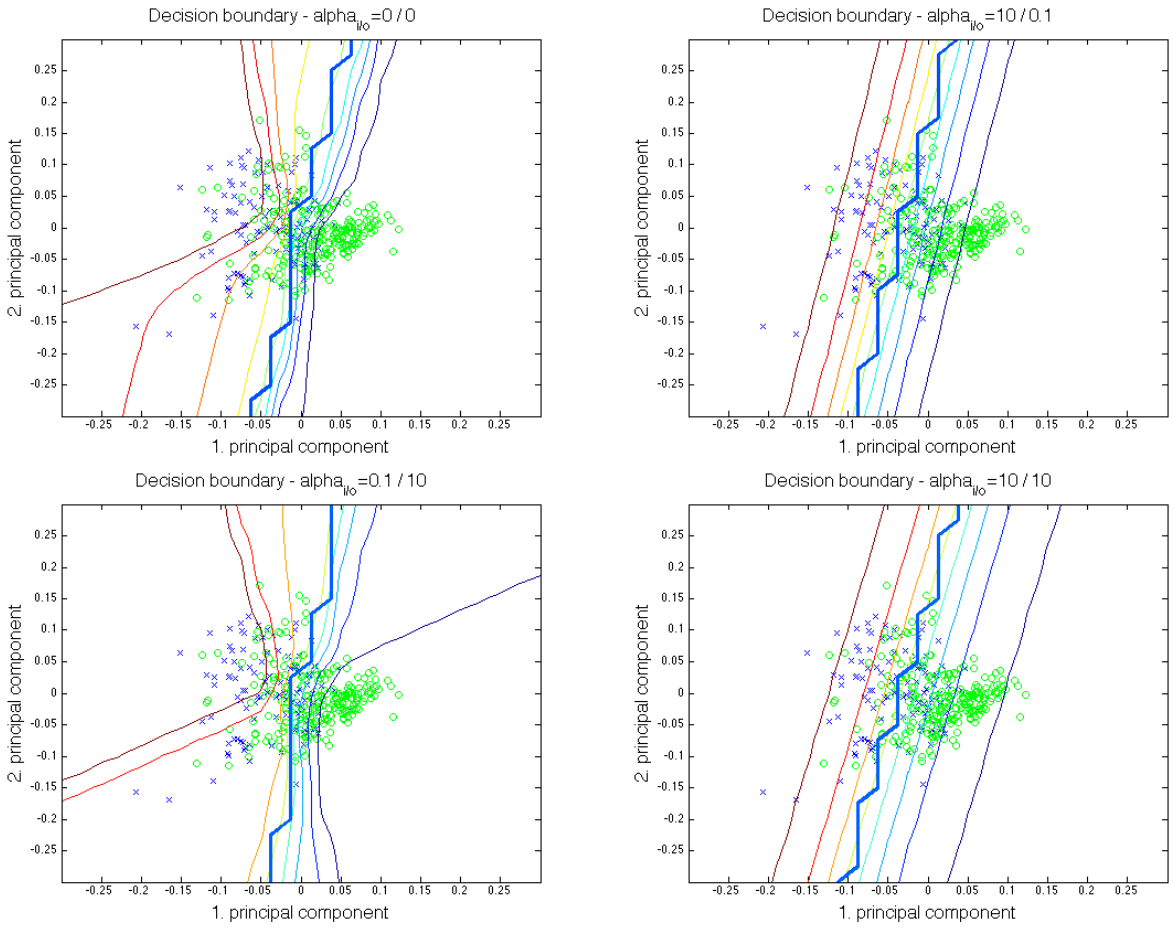


Figure 1: In all these figure, the initialization range is set to 0.3.

We see that when both weight decay parameters are 0, the weights are free to adapt as they please, and the resulting decision line and probability densities shows non-linear characteristics (top left corner). When the output weight decay parameter is large compared to the other one,

the weights in the output layer are small. In this case weights scaling the input to the $softmax$ function are small, but the weight from the input layer are large, so the $tanh$ function causes the output to be non linear. When the input weight decay are large, the input weights become small, and the output becomes linear. The same happen when both weight decay parameters are large. The main difference between the output when both weight decay parameters are large and when only the input weight decay parameter is large, is the behavior of the $softmax$ function. When the output decay parameter is small, the output weights become large, and the $softmax$ resembles a step function, making the slopes of the conditional densities steep, so that the contour lines appear closer to each other.

## Checkpoint 6.2

**Error functions for multi class classification problems.**

In this checkpoint we investigate the properties of two different error cost functions as a function of the difference between the output and the target, for a multi-class problem in which the input is assigned to one of $K$ mutually exclusive classes.

The two different cost function are the squared error given by

$$E_{sq} = \frac{1}{2}\sum_n \sum_k (y_k^n - t_k^n)^2 \tag{1}$$

and the cross entropy cost error function is given by

$$E_{ce} = -\sum_n \sum_k t_k^n \, ln(y_k^n). \tag{2}$$

where $0 < y_k < 1$ is the output of the network activated by the $softmax$ function,

$$y_k = \frac{exp(\phi_k)}{\sum_j exp(\phi_j)} \tag{3}$$

and can be interpreted as the probability of class $k$ given the data $p(C_k|x)$. Since we only have two classes and they are mutually exclusive, we can write the prabablitites as

$$y_1 = \frac{exp(\phi_1)}{exp(\phi_1) + 1} \tag{4}$$

$$y_2 = 1 - y_1 \tag{5}$$

We create a sequence of outputs from an imaginary neural network ranging from $\phi_{out} = [-10, 10]$ and a target value $t_1 = 1$. We want to inspect the behavior of the cost functions when the imaginary neural network outputs $y_out$ when the true class vector is $[t_1 t_2]^T = [10]^T$.

The cost for the different outputs are calculated both before and after the $softmax$ is applied, so there is one plot where the x-axis is the $\phi_{out}$ vector and one where the x-axis is the $y_1$ vector.

In order to maximize the probability of classifying the input as class one, which is the true class, we want the output $\phi_{out}$ to be large, so that after the $softmax$ has been applied, the probability is close to one.

From Fig. 2, it is easily seen that this is best done by using the entropic cost function, because it penalizes small $\phi$'s and small probabilities much more than the squared error.

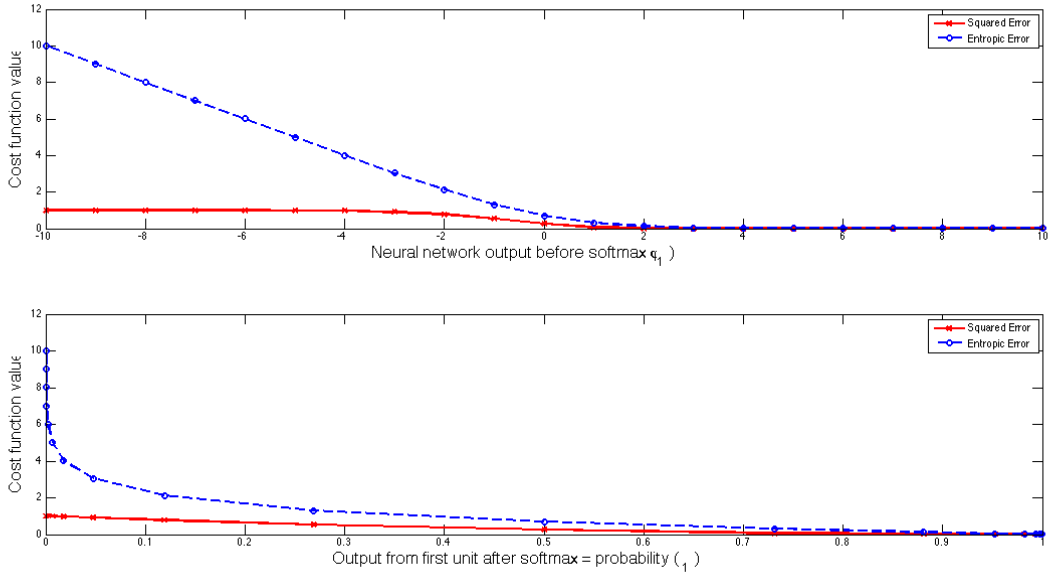


Figure 2: The squared and entropic error cost functions vs. the output from the network.

As is also mentioned in the exercise formulation, the use of the squared loss function is based on the assumption of additive gaussian distributed noise of the output, which is not the case here, since the output is binary. In this context, the noise would manifest itself as a misclassification, and would be strictly non-negative.

## Checkpoint 6.3

### Training Set Size

The amount of training data is of course important to the performance of

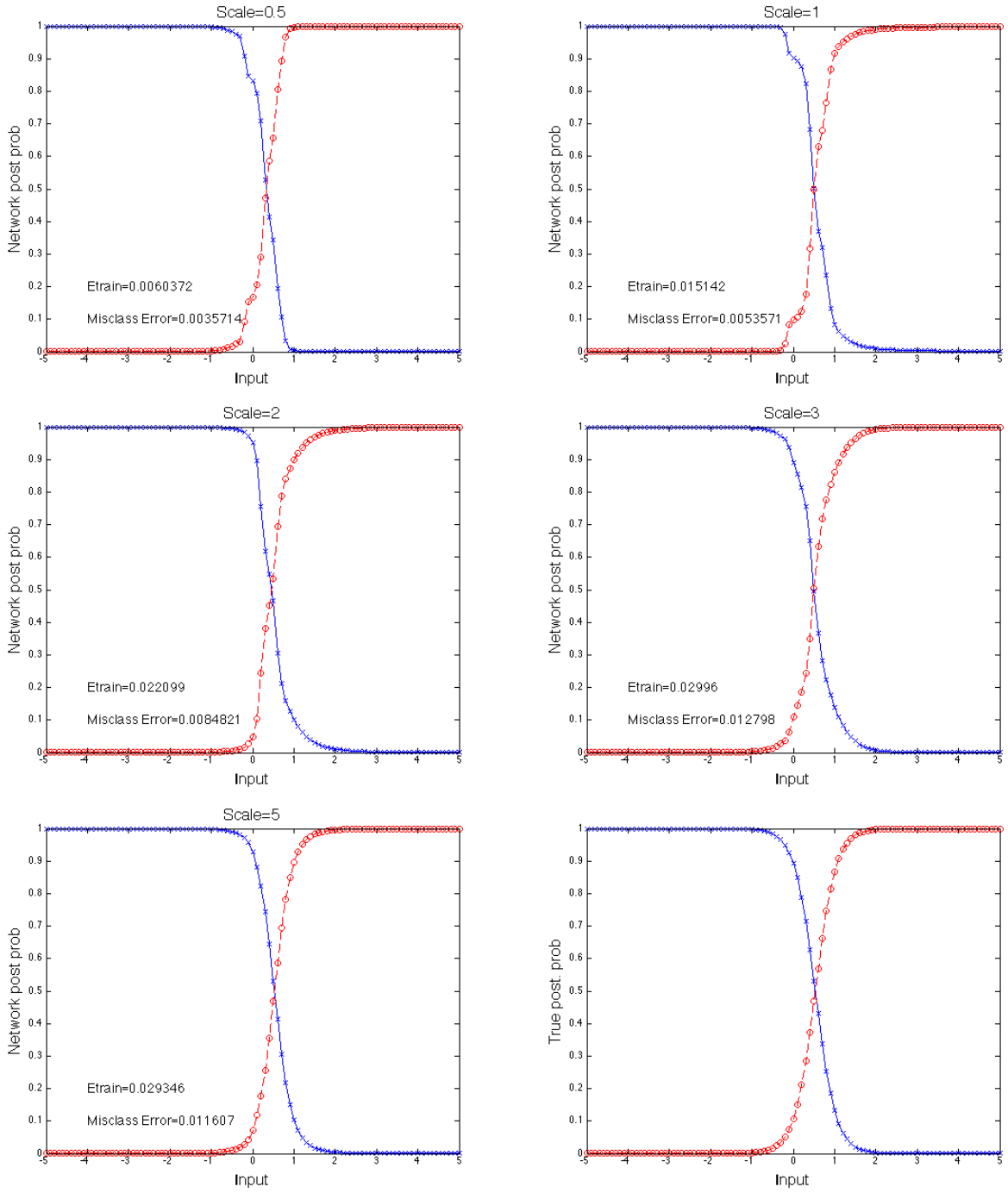


Figure 3: The posterior probabilities estimated by a neural network for a gaussian mixture mddl with two gaussian components. The number of training points from each cluster is (100,12) *scale* The posterior distribution is calculated 10 times for each training set size and averaged.

In Fig. 3 it seems that if there is very few data points, there is a low training error and misclassification error even though the estimate of the posterior distribution is poor. This could be because there are very few points to misclassify. As the number of points grow, the estimated posterior distribution becomes more accurate and the training and misclassification errors settles to fluctuate around a fixed level.

## Checkpoint 6.4

### Pruning

In this checkpoint we are going to investigate a method called **pruning** that can be used to limit the model complexity of a neural network.

This is done by first training a fully connected network, and then calculating the approximate change in the error cost function that would incur if we removed one of the weights (set it to zero).This is called the saliency. We do this for all the weights, and the weight that prove to be un-important are pruned (set to zero). We can gain more insight about the **saliency** by deriving the expression for it.

First we make a taylor's expansion of the error cost function $E(w)$ around its minimum,

$$E(w) = E(w^*) + \frac{\partial E}{\partial w}(w - w^*) + \frac{1}{2}\frac{\partial^2 E}{\partial w^2}(w - w^*)^2 + \mathcal{O}(w^3) \tag{6}$$

$$\Delta E(w) \approx \frac{1}{2}\frac{\partial^2 E}{\partial w^2}(w - w^*)^2 = S \tag{7}$$

For the deletion of the $j^{th}$ weight, we define the saliency $S_j$ of that weight to be

$$S_j = \frac{1}{2}\frac{\partial^2 E}{\partial w^2} w_j^2 \tag{8}$$

The saliency therefore says something about the sensitivity of the error function with respect to a particular weight, and wether it is important or not for the error function. Weights with small saliencies are therefore pruned away to obtain a more sparse network.

In script **main6d.m** a neural network was trained on the pima indian data set to illustrate the effect of pruning. This neural network takes 7 inputs and delivers 1 output.

The inputs are

1. Number of pregnancies
2. Plasma glucose concentration
3. Diastolic blood pressure
4. Triceps skin fold thickness
5. Body mass index (weight/height$^2$)
6. Diabetes pedigree function
7. Age

When running the script several times, it becomes clear that there are two configuration of weights after the pruning have have a low error cost, one where only two weights survive plus bias, and one where five weights survive plus bias.

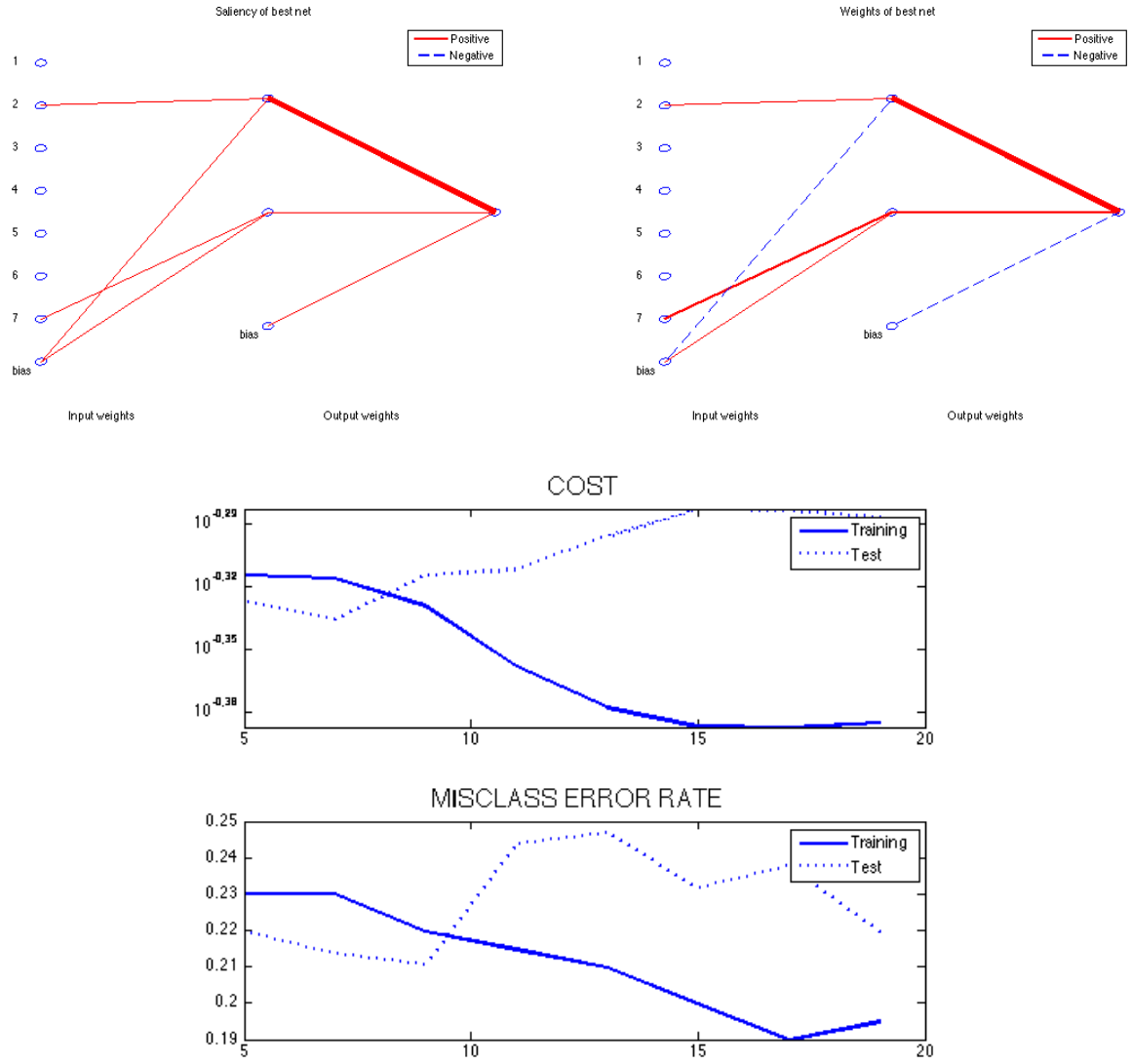


Figure 4: In this network the most important inputs are the the plasma glucose concentration and age plus bias (negative).

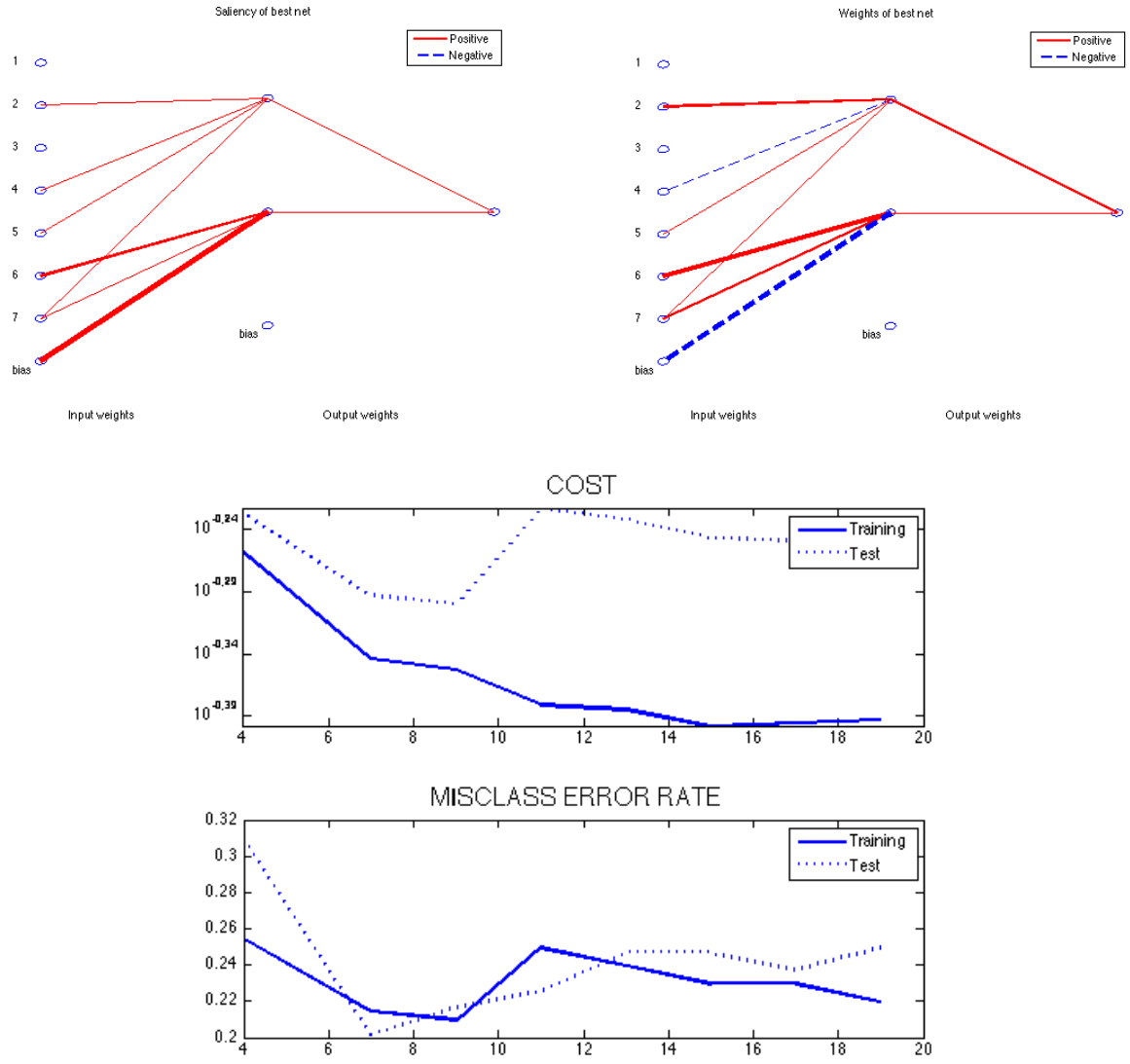


Figure 5: In this network the most important inputs are the plasma glucose concentration, triceps skin fold thickness, the body mass index, the diabetes pedigree function and the age plus bias (negative). of these five (excluding the bias), the glucose concentration and the pedigree seems to be the most salient.