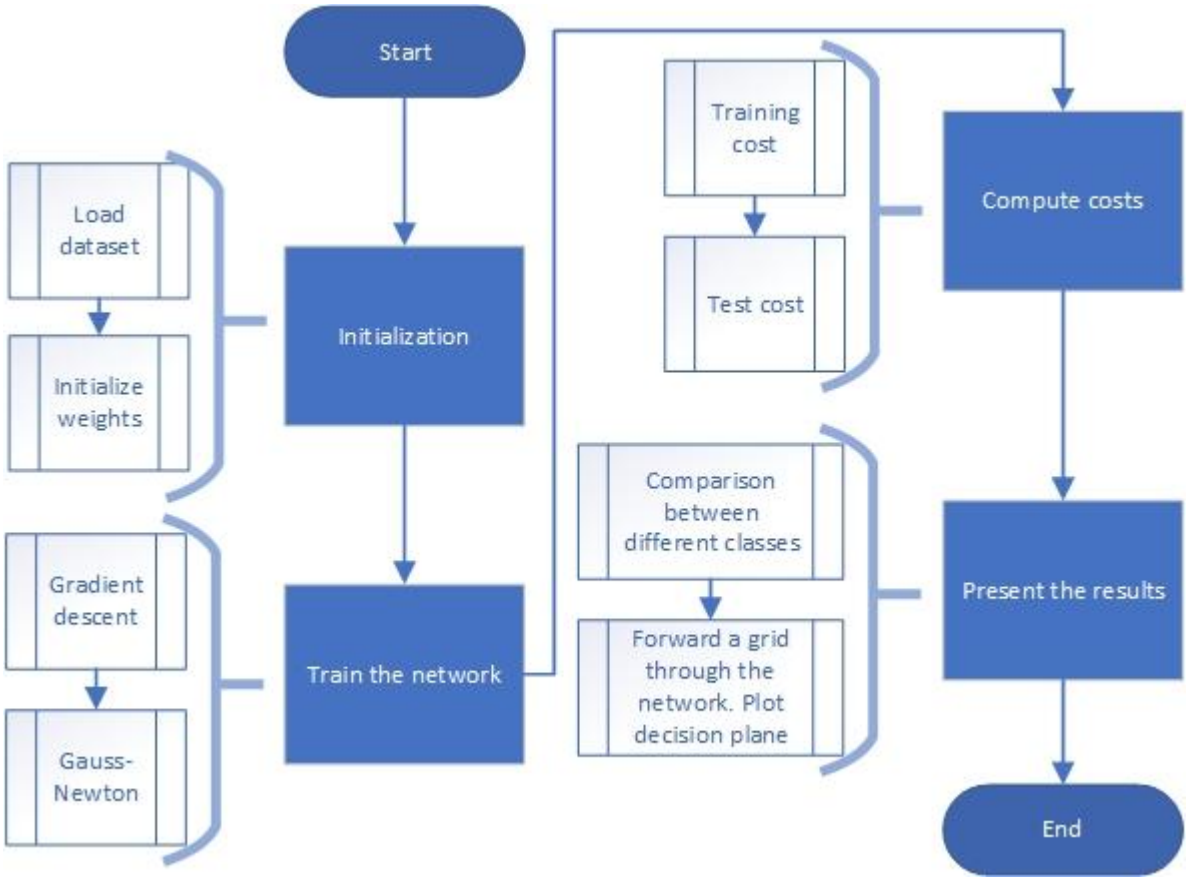
Checkpoint 6.1

Use the program `main6a.m` to optimize a neural network and plot the decision boundary and conditional probabilities. The heavy blue line is for locating the contour line corresponding to the equal probability of the two classes.

Run the network with different weight decays and comment on the difference in the decision surfaces. Create a flow chart of `main6a.m` and its functions.

First, we are going to create a flowchart to understand the given script:



In order to show the results we have to take into count that we are working with seven different dimensions. In order to be able to represent that in two dimensions what we do is we take the two principal components (the coordinate system with the greatest variance) and represent those two instead. Then, in order to represent the decission boundaries what we will do is to do a forward pass with a grid, which will allow us to map the different probabilities.

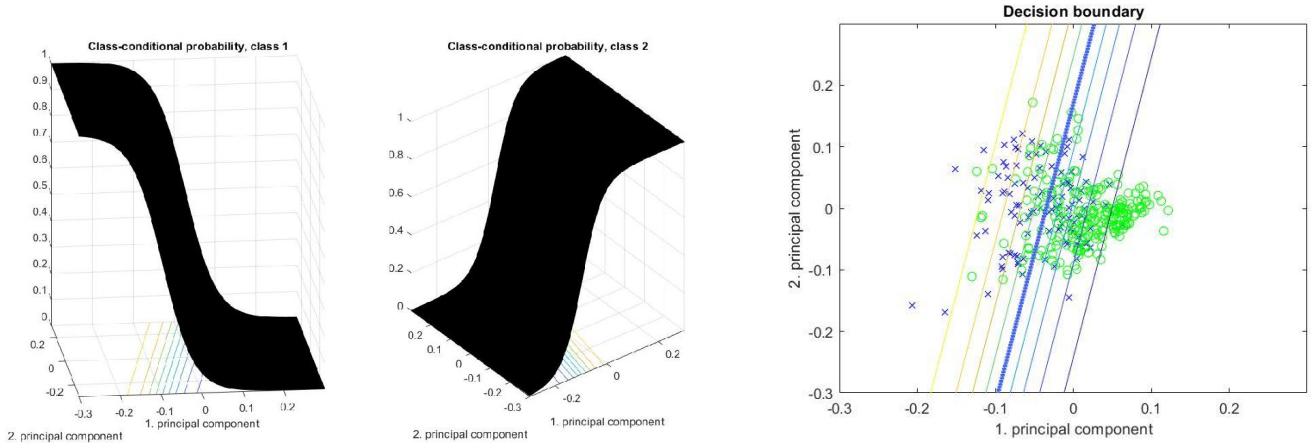
After understanding the main elements of the program, let's take a look at the influence of weight decay in the performance of the network. For that matter we have two parameters, the weight decay of the input and of the output layers, refering to the weights between input and hidden layer on the one side and the hidden and output layer respectfully. Weight decay penalizes big weights, both negative and positive, but what is the influence of each in the network?

First, let's take a look at the ecuations that gobern the gradients:

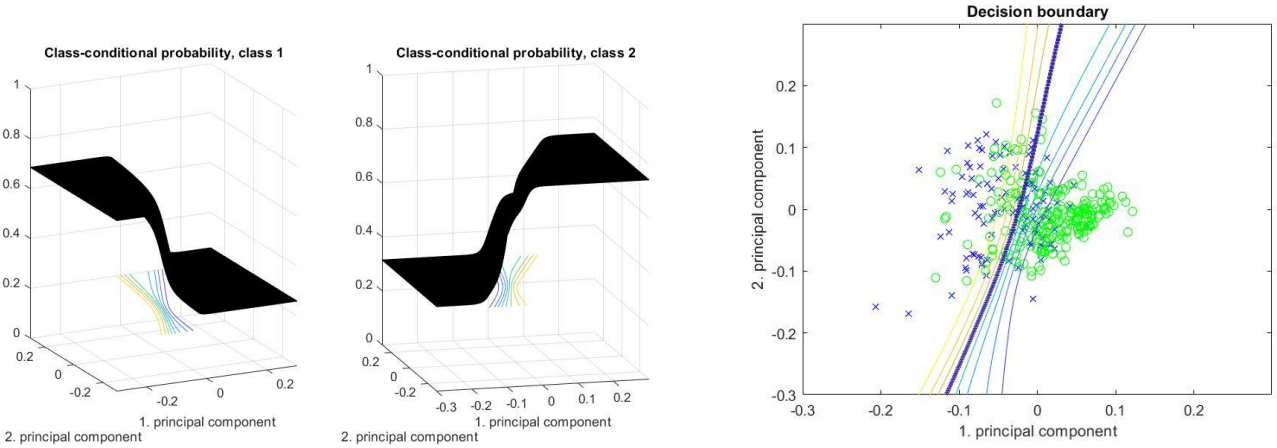
$$\triangle w_{kj}^{(2)} = -\eta \left(\sum_{n=1}^{N} (y_{nk} - t_{nk}) z_{nj} + \alpha w_{kj}^{(2)} \right)$$

$$\triangle w_{ji}^{(1)} = -\eta \left(\sum_{n=1}^{N} \left((1 - z_{nj}^2) \sum_{k=1}^{c} w_{kj}^{(2)} (y_{nk} - t_{nk}) \right) x_{ni} + \alpha w_{ji}^{(1)} \right)$$

When the alpha_in is too big the weights converges to 0. If the weights(1) are really small, due to the nature of the activation function the model will stop to behave as a non-linear model, and start to work as a linear one instead, which explains the straight line in the decision boundary.

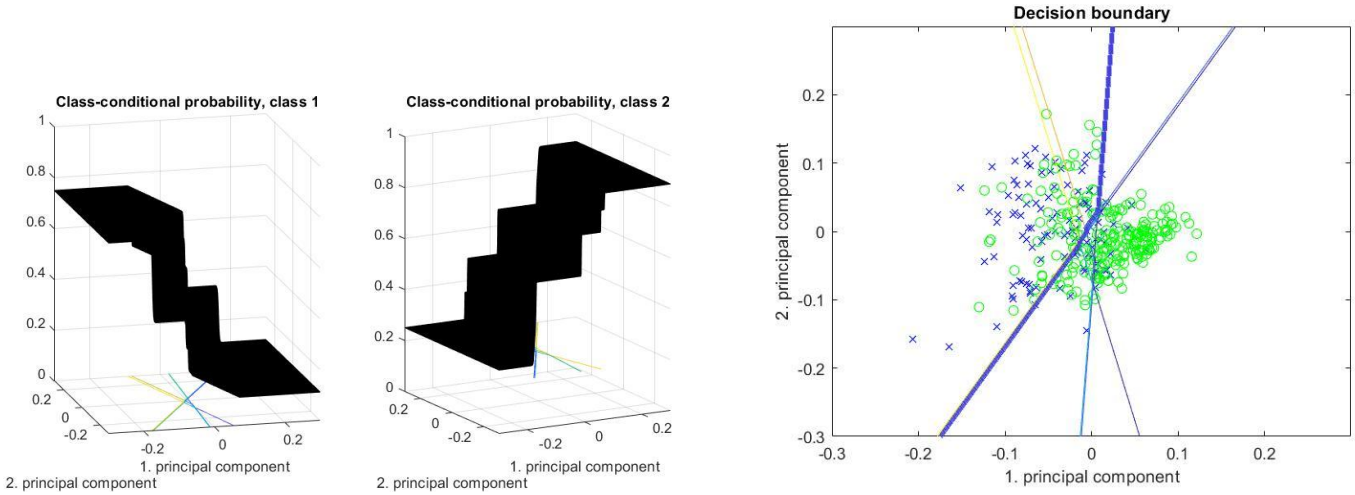


When alpha_out is too big it becomes more difficult to distinguish between classes. As the weights become really small the difference in the output, which will be fed to the softmax function, becomes also really small and the influence of the previous layer is reduced. The NN becomes less confident about its predictions, as we can note from the following results:



But what about we use large weight decays for both input and outputs? In this case both effects are combined, and the decision boundaries become linear at the same time that the confidence is reduced.

Now, if we take the previous example when the weight decay on the output was quite big and we reduced the one in the output we obtain the following result:



In this case the boundary planes are not that smooth, due to overfitting. The first layer does not suffer regularization anymore, and the weights we obtain are really big, allowing for this sudden changes on the boundaries.

Checkpoint 6.2

The program `main6b.m` calculates the cost function value associated with one example for a two-class problem. It sweeps the output of the neural network ϕ_1 from -10 to 10.

What was the target t_1? How much do the two cost functions penalize a "wrong" decision.

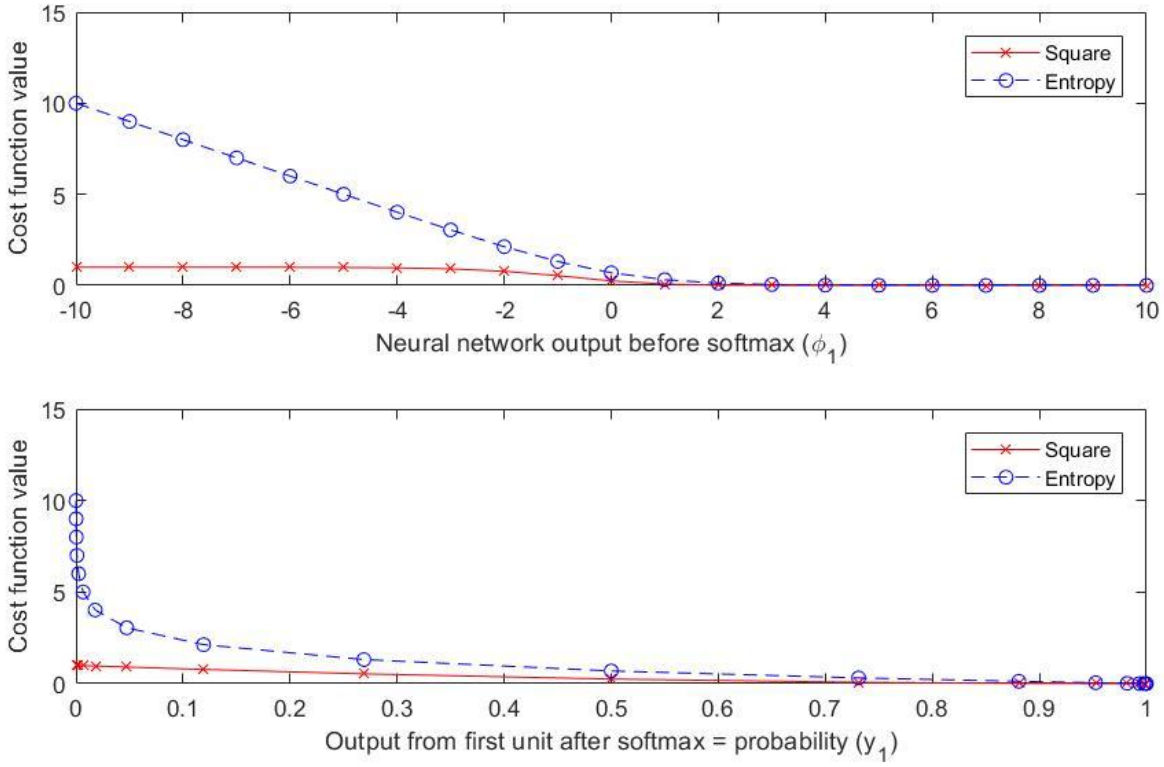
In this problem we are trying to see the evolution of the cross-entropy error and the mean square error of a given artificial output of a NN that ranges between -10 and 10. This values allows us to see how the error function changes when the output of the network is 0, moment when both classes obtain the same probabilities. This result is independent of the other outputs of the network, as the values of softmax are independent of the other results on the network. This can be proben from the softmax equation:

$$y_k = \frac{\exp(\phi_k)}{\sum_j^{c-1} \exp(\phi_j) + 1}, \qquad k = 1, 2, \ldots, c-1$$

$$y_c = 1 - \sum_k^{c-1} y_k \,.$$

If we introduce then our output of 0, we can see that the y_k will be equal 0.5.

Now let's take a look at the comparison between cross-entropy error and mean square error using a target of 1:

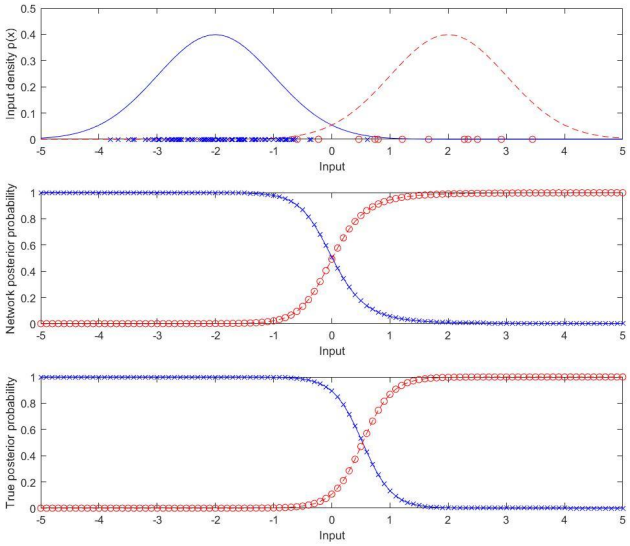


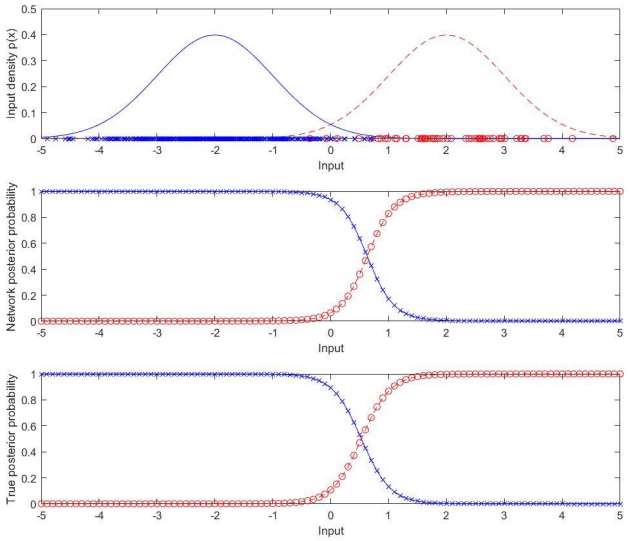
We can determine at first glance that the cross entropy cost function penalizes more the outputs that are far away of the target value in comparison with the square mean error. This allows the network to learn faster, as the error that is backpropagated is bigger when it is really far of.

Checkpoint 6.3

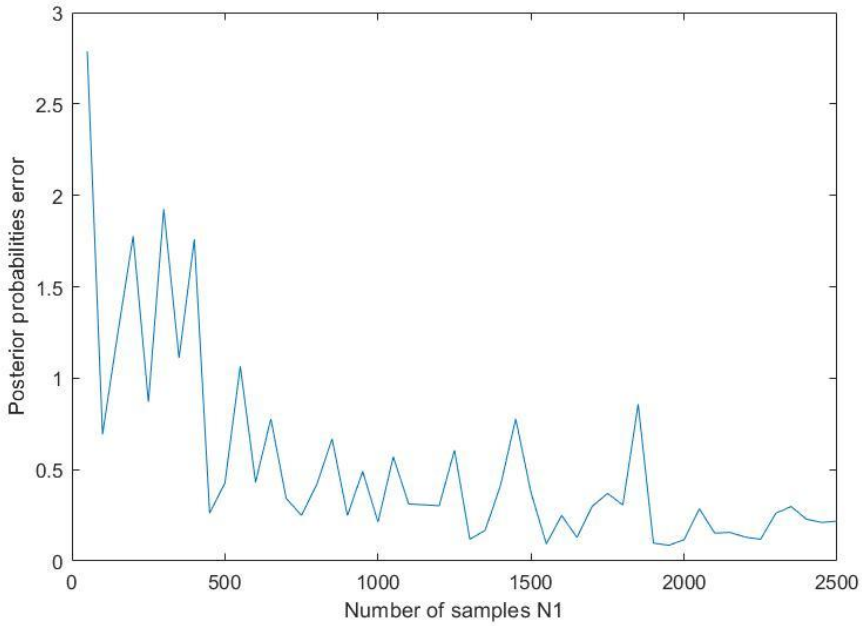
Use the program `main6c.m` to plot posterior probabilities. The data is very simple with one dimensional x and two classes each Gaussian distributed with $\sigma_{\mathcal{A}}^2 = \sigma_{\mathcal{B}}^2 = 1$, $\mu_{\mathcal{A}} = -2$ and $\mu_{\mathcal{B}} = 2$, thus $p(\mathrm{x}|\mathcal{C}_{\mathcal{A}}) \sim \mathcal{N}(-2, 1)$ and $p(\mathrm{x}|\mathcal{C}_{\mathcal{B}}) \sim \mathcal{N}(2, 1)$. There are 100 samples in $\mathcal{A}$ and 12 samples in $\mathcal{B}$, initially, change this to larger values (e.g. 500 and 60, to keep the prior probabilities the same) and observe how the posterior approaches the true posterior.

In this exercise we are working with artificially generated data, that we choose the prior probability from. After constructing ground truth data we are going to train our network based on that dataset. Once it is trained we can compare the posterior probabilities out from our neural network (obtained by making a forward pass with a grid) and the true posterior, obtained by using bayes theorem.





From the results when N1=100 we can see that the posterior probabilities out of the NN are not perfect, due to the lack of points to map the boundary of the distributiions. We can see how it is solved when N=500. We observe this way that with a larger dataset we are able to map the true posterior probabilities better, but let's do an experiment to probe so. In the following graph we have mapped the error between posterior probabilities for different sizes of the dataset:



We can see that the error reduces with the number of samples, improving significantly the results.

Look at artificially generated data. We choose prior probability, from either of the two classes. We draw as gaussian distribution. We construct ground truth data. If we know prior probabilities and densities we can also prlot the probability for each of the classes and compare to the one that we got from.... .

We can do overfitting if we run it varius times, because of the smooth transition between the different classes.

Checkpoint 6.4

Derive the saliency expression as the approximate cost of removing a single weight while keeping the remaining weights fixed at their 'optimal' values.

The best net is selected as the net with the lowest cost function value on the test set.

Note that not all inputs are used, is the number of pregnancies important for the diagnosis?

The target output is 1 for positive diagnosis (diabetes), and 0 for negative diagnosis (healthy).

What does the neural network say about the relation between age and diabetes (positive/negative correlation)?

If we assume that the error is nearly quadratic we can perform a second order expansion on the cost error funtion we obtain the following:

$$E(\mathbf{w}) \approx E(\mathbf{w}^*) + \frac{\partial E}{\partial \mathbf{w}}(\mathbf{w}-\mathbf{w}^*) + \frac{1}{2}(\mathbf{w}-\mathbf{w}^*)^T \mathbf{H}(\mathbf{w}-\mathbf{w}^*)$$

with w* being the new weigths after the pruning. Thus, we can define the change on the cost error as:

$$\delta E = E(w+\delta w) - E(w).$$

Merging both equations we obtain that the error change is defined as follows

$$\delta E \approx \delta w \frac{\partial E}{\partial w} + \frac{1}{2}\delta w^T H \delta w$$

In a well trained network we can assume that we have reached a minimum of the cost error, so the first equation is equal to zero. Let's take a look now at the Hessian matrix:

$$H = \begin{pmatrix} \frac{\partial E^2}{\partial w_1 \partial w_1} & \cdots & \frac{\partial E^2}{\partial w_1 \partial w_K} \\ \cdots & \cdots & \cdots \\ \frac{\partial E^2}{\partial w_K \partial w_1} & \cdots & \frac{\partial E^2}{\partial w_K \partial w_K} \end{pmatrix}.$$

Now let's assume that the change of the error contributed by "perturbing" several weigths is the sum of the error of perturbing each weight individually. i.e. there is no correlation between the "perturbations" of multiple weights. In this case this matrix becomes diagonal, simplifying the error change to the expresion:

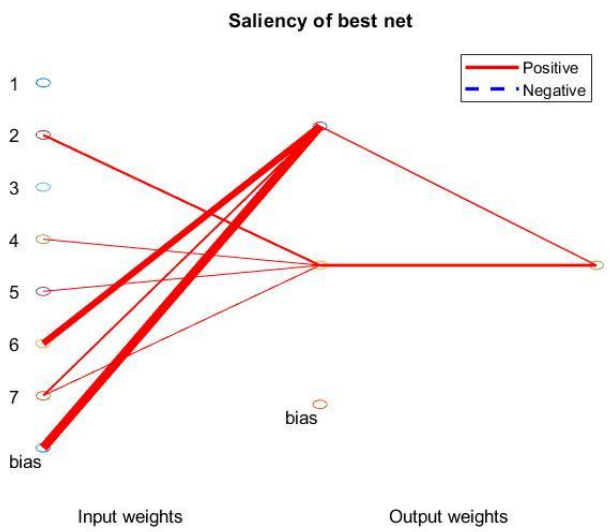
$$\delta E \approx \frac{1}{2}\sum_{i=1}^{K} h_{i,i}\delta w_i{}^2$$

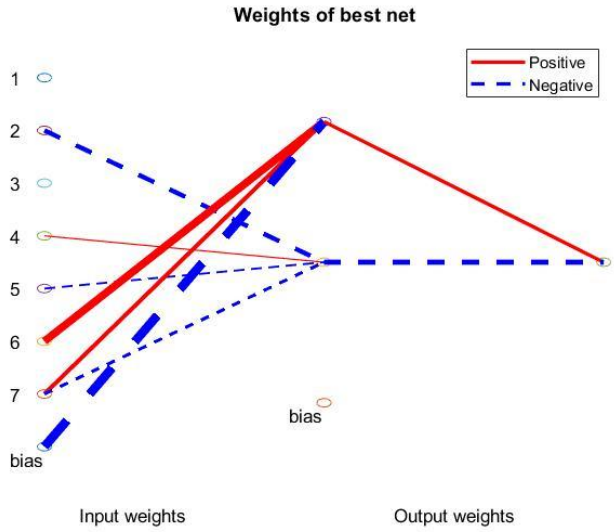
Finally, as we are not adding a perturbation but setting the value of the weight to 0, the error change by deleting a weight will be equal to:

$$\delta E_i \approx \frac{1}{2} h_{i,i} w_i{}^2.$$

Now we can calculate the saliency of all the weights and remove the ones that affect the least to the NN. Afterwards, we will retrain the network and repeat the process.

Based on this method we have obtained the following results:





This results, apart from giving us the NN with a reduced error, give us a way to interpret the model. In this case we can see that the number of pregnancies (1) and the Diastolic blood presure (3) does not have any important effect on the output, so they are removed from the model. On the other hand, we have values like age (7) which has a direct influence to getting a positive diagnosis when (6) is also big but a negative relation when other data (2) (5) are big.