Machine learning II
Master Data Science
Winter Semester 2023/24 **Prof. Tim Downie**

Workshop 9
Basic artificial neural networks

In this workshop you will code a basic neural network from scratch, using the basic definitions of a NN. The purpose of the workshop is to learn the fundamental principles a NN rather than writing code for proper data analysis.

An R script file, `Workshop9.R`, is in Moodle for you to work through. Please read all of this worksheet before working through the script file.

Several simulated data sets have been provided in one file and can be loaded using
`> load("NNdatasets.Rda").`
The variables provided in this file are:
`x1`, `x2`: two input variables (training data) of length 20,
`y`: the corresponding outcome variable,
`x1val, x2val, yval`: an equivalent validation data set,
`xcl1, xcl2, ycl, xcl1val, xcl2val, yclval`: similar variables for classifier NN (with length 100 and 25 respectively).

The script file gives the code to scale the training and validation data.

Tips:

- A user defined function in your script file, needs to loaded into R after every time it is modified. To do this put the cursor somewhere in the function and type *Ctrl*+*Alt*+*f*.
- To run a whole block of code, that is defined using `{ }`, put the cursor on the same line as the `{` or `}`, it doesn't mater which, and type *Ctrl*+*Enter*. This is useful to re run the NN with a different number of iterations.

In each exercise the function `NN`, `NN2` and `NN3` run the neural network for a given set of weights and biasses. These parameter values and the observation values are passed into the function as arguments.

`NN` and `NN3` make use of the R-Vector arithmetic to fit the current set of parameters to the all the observations in one sweep. Take, for example, the line in the function `NN`

```
z1<-whl11*x1+whl12*x2+bhl1
```

`x1` and `x2` are vectors, so the resulting object `z1` is also a vector. This is much faster than using a `for` loop on each observation. In `NN2` it is more difficult to do this and the coefficients are calculated for each observation using a `for` loop.

The **minimisation routine** used in the code is a rejection routine. It tries out a new parameter value, if the loss function is lower then that new parameter value is accepted but is rejected it if it is worse.
An outline of the routine is:

- Let $\boldsymbol{\theta}^{\text{cur}}$ be the current vector of parameters with length Q.
- Simulate Δ, one normally distributed random number with standard deviation `window`.
- Perturb only the q-th element of $\boldsymbol{\theta}^{\text{cur}}$ by a random amount and set $\theta_s^{\text{new}} = \begin{cases} \theta_s^{\text{cur}} + \Delta & \text{for } s = q \\ \theta_s^{\text{cur}} & \text{for } s \neq q \end{cases}$
- Run the NN using this new parameter vector $\boldsymbol{\theta}^{\text{new}}$ and calculate the loss function.
- If the new loss function is smaller than before, update $\boldsymbol{\theta}^{\text{cur}} = \boldsymbol{\theta}^{\text{new}}$ otherwise reject this attempt.
- Loop q over each element $1, \ldots, Q$ and repeat ad infinitum!

This minimisation algorithm is very easy to code but it is slow and requires many thousands of iterations.

Exercise 1

As in the lecture we will start with a neural network with one hidden layer and just one node in the hidden layer. The first step is to scale the training and validation data and define the sigmoid function.

The parameters are

`whl11`	w coefficient in hidden layer node 1 for x_1
`whl12`	w coefficient in hidden layer node 1 for x_2
`bhl1`	bias coefficient in hidden layer node 1
`wol11`	w coefficient in output layer for activation a_1
`bol`	bias coefficient in output layer

The initial number of iterations is 10, so each parameter will be updated at most twice. Run the for loop. Look at the"best so far" SSE and plot the current predicted values against the true values.

Increase the number of iterations to 10, 1000, 10^4 and 10^5.

After each run write down the training and validation SSE.

There is a commented out command which slowly reduces the window. Uncomment this line of code. Reducing the window with the number of iterations size seems to reduce the loss a little bit quicker.

Re-run the code using several times notice that the validation MSE value varies more than the training MSE.

Exercise 2

Now add a second node to the hidden layer.

The *new* parameters are

`whl21`	w coefficient in hidden layer node 2 for x_1
`whl22`	w coefficient in hidden layer node 2 for x_2
`bhl2`	bias coefficient in hidden layer node 2
`wol2`	w coefficient in output layer for activation $a_2^{(1)}$

You will need to define the formulae for the new components in the function `NN2`.

Again play around with the number of iterations and the window size.

It seems that the SSE improves faster than in Exercise 1.

Exercise 3 Neural network as a classifier

For classification the output layer requires as many nodes as there are classes. the data `ycl` has $K = 3$ classes (labelled 1, 2 and 3).

The function `NN3` has 3 output activations (don't forget to apply the sigmoid function). The activations then need to be normalised to give a predicted probability for each class. This last step uses a couple of "R-Tricks" to be able to do this without using a for loop, so this step has been provided for you.

Run the for loop and investigate the output, including the resulting classification matrix.

Re-run obtaining the loss function for the training and validation data set after every 1000 iterations. So that you can follow the progress of the training, a dot plot of the probabilities of the true category is drawn.