Machine learning II
Master Data Science
Winter Semester 2019/20

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

Prof. Tim Downie

Workshop 8
Coding Artificial Neural Networks

A source file `Workshop8.R` is provided in Moodle which you should work through for this workshop. Please read all of this worksheet before working through the R Code.

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,
`x1test, x2test, ytest`: an equivalent validation data set,
`xcl1, xcl2, ycl, xcl1test, xcl2test, ycltest`: similar variables for classifier NN (lengths 100 and 25 respectively).

The script file gives the code to scale the training and test data. Notice that the same values are used to transform both the training and validation data sets.

Hints:

- In R-Studio to load/update the an R-Function use *Ctrl*+*Alt*+*f* when the cursor is in that function.
- With the cursor on a `for(........)` command type *Ctrl*+*Enter* to run the whole loop, without needing to select the whole loop.

In each exercise the function `NN`, `NN2`, etc. applies the NN to all the observations in one sweep. This makes use of the R-Vector arithmetic, for example in the code line
`z1<-whl11*x1+whl12*x2+bhl1`
`x1` and `x2` are vectors and so the object `z1` is also a vector. This is much faster than using a for loop for each observation. The parameter values and the observation values are passed into the function as arguments.

The **minimisation routine** used in the code is a rejection routine. It tries out new parameter value and rejects 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 then 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 test 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 1000, and then 10000.
There is a commented out command which slowly reduces the window. Uncomment this line of code. Reducing the window size seems to reduce the loss a little bit quicker.

Re-run the code using the starting coefficients again. This time compare the SSE for the test data after every 1000 iterations.

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

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

It seems that the SSE improvement is faster than for the NN with one node.

Exercise 3 Neural Net 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 test data set after every 1000 iterations.