Lab 3: Exploring The Neural Net Package

Tshiamo Phaahla

19/11/2021

Please Note The Following.

The content contained in this document is from the third lab session in a series of lectures prepared for a two-week introductory course in Machine Learning at the University of Cape Town, South Africa. The course is aimed at students with some background in statistical modelling, computing, and linear algebra. It is recommended that you watch the third Key-Point Lecture in this series before tackling the lab session.

Link to the the relevant lab https://youtu.be/YfUUR6lj58Y

==========

Goal: Analyse the iris data using the neuralnet package (no validation, just use all the data) and evaluate the response curve under a neural network.

Task 0: Prepare the data by scaling the inputs of the iris data.

```
library(neuralnet)

data(iris)
attach(iris)

newData = iris
newData$Sepal.Length = scale(newData$Sepal.Length)
newData$Sepal.Width = scale(newData$Sepal.Width)
newData$Petal.Length = scale(newData$Petal.Length)
newData$Petal.Width = scale(newData$Petal.Width)
```

Task 1: Fit a (5)-network to the iris data using the Petal predictors. Choose an appropriate specification for the activation and objective functions, and set the parameters for the backprop algorithm. Plot the resulting model object.

Few things to note:

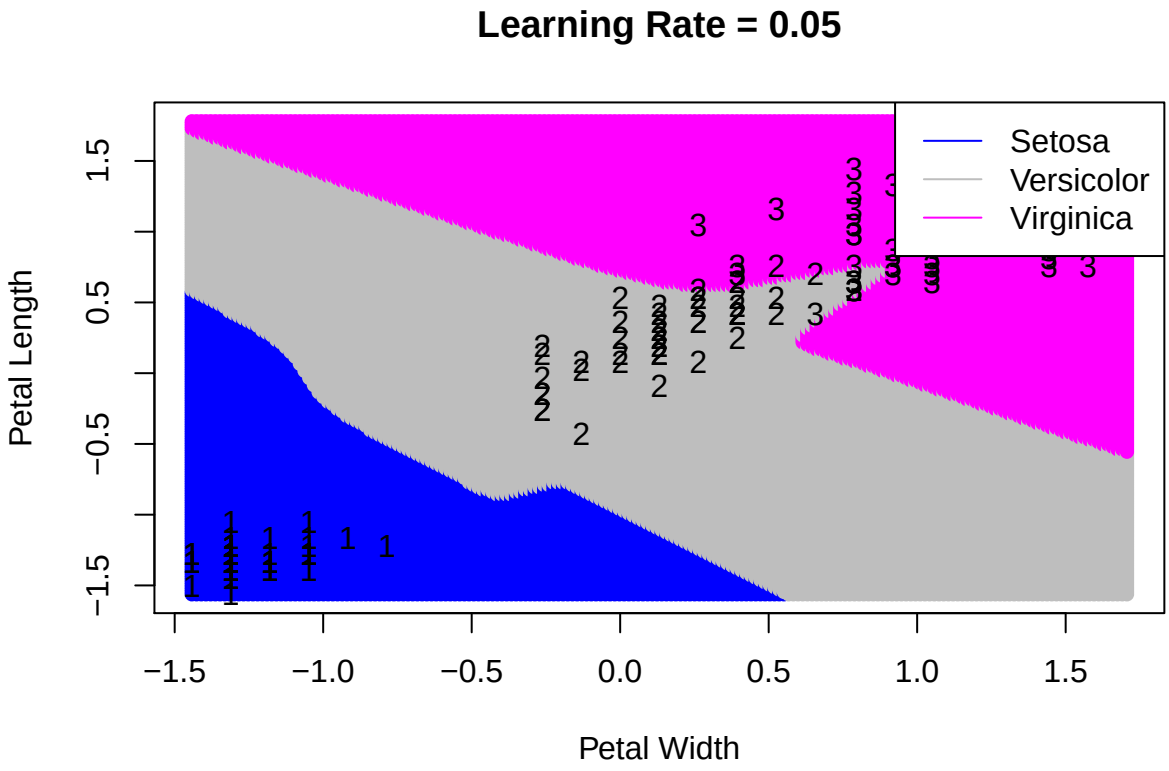
- Threshold is topping criteria. We need all gradient to fall below this threshold before we can say that our gradient descent algorithm has converged.
- stepmax: number of steps for training fo neural network.
- Can only set learning rate if we have specified we are using back propagation.

- err.fct: the objection we need to set to calculate the errors. Can use default or create our own.
- In regression we use SSE in classification CE - cross entropy.

```
set.seed(2021)
neuralMod = neuralnet(Species~ Petal.Length + Petal.Width, data = newData,
                      hidden = c(5),
                      algorithm = 'backprop',
                      err.fct = 'ce',
                      act.fct = 'logistic',
                      learningrate = 0.05,
                      stepmax = 10^6,
                      linear.output = F, threshold=0.01)
                      #lifesign = 'full') # lifesign full means it will keep reporting as it's fitting

plot(neuralMod)
```

Task 2: Construct a response curve for the model fitted in Task 1. Remember that the data are now scaled!

```
M=200

x1_dummy = seq(min(newData$Petal.Length), max(newData$Petal.Length), length=M)
x2_dummy = seq(min(newData$Petal.Width), max(newData$Petal.Width), length=M)

x1 = rep(x1_dummy, M)
x2 = rep(x2_dummy, each=M)

Lat = data.frame(Petal.Width = x2, Petal.Length = x1)

pred = predict(neuralMod, Lat)

clss = apply(pred, 1, which.max)

cols = c('blue', 'gray', 'magenta')

plot(Lat$Petal.Length~Lat$Petal.Width, pch=16, col=cols[clss],
     xlab="Petal Width", ylab = "Petal Length", main = "Learning Rate = 0.05")

text(newData$Petal.Length~newData$Petal.Width, labels= as.numeric(newData$Species))

legend('topright',lty=1, legend = c('Setosa', 'Versicolor', 'Virginica'),
       col = c('blue', 'gray', 'magenta'), bty='o', bg='white')
```




```
#text(newData$Petal.Length~newData$Petal.Width, labels= as.numeric(newData$Species))
```

A Few Comments

- A neural network is just a model. Yes we can fit a model to the data but that doesn't mean that any learning has taken place.
- We have no complexity controls applied so we cannot guarantee that it has actually learned.
- Validation analysis is very very very very important.
- NeuralNet package lacks regularisation. This is strange in the world of machine learning because obviously we need complexity controls and the complexity controls for neural networks is regularisation.

Extra Problems

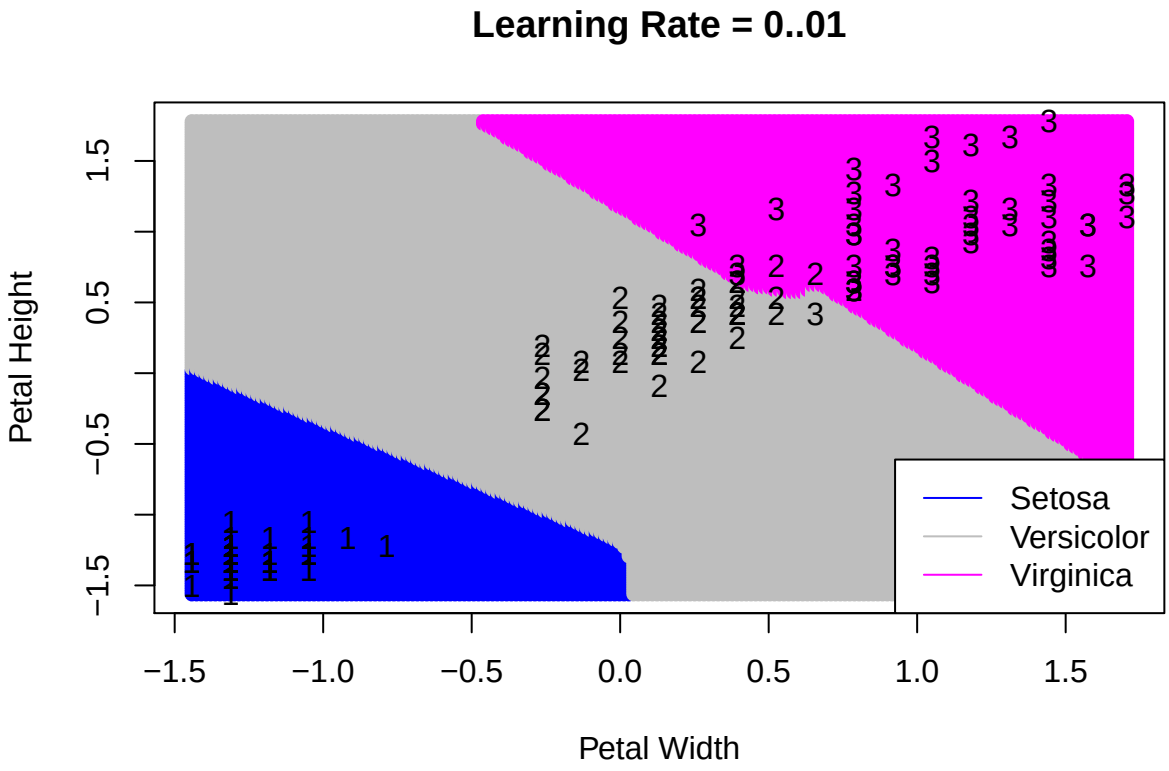
Problem 1

For the iris data, what activation function would you specify for the output layer that would be more appropriate?

ReLU. When training a network with many layers, the gradient can diminish dramatically as the network is propagated through backwards. The error may be so small that it has little effect by the time it reaches layers close to the input. This can result in the algorithm converging too early without propagating useful gradient information from the output layer to the layers closest to the input layer.

Problem 2: Re-run the analysis (all else equal) using 'learning rate=0.01'.

```
model2 = neuralnet(Species~ Petal.Length + Petal.Width, data = newData,
                   hidden = c(5),
                   stepmax = 10^6,
                   learningrate = 0.01,
                   threshold= 0.01,
                   algorithm = 'backprop',
                   err.fct = 'ce',
                   act.fct = 'logistic',
                   linear.output = F)#,
                   #lifesign = 'full'
                   #)

#plot(model2)

pred1 = predict(model2, Lat)
clss1 = apply(pred1, 1, which.max)

plot(x1~x2, pch=16, col=cols[clss1], xlab="Petal Width",
     ylab = "Petal Height", main = "Learning Rate = 0..01")

text(newData$Petal.Length~newData$Petal.Width, labels= as.numeric(newData$Species))

legend('bottomright',lty=1, legend = c('Setosa', 'Versicolor', 'Virginica'),
       col = c('blue', 'gray', 'magenta'), bty='o', bg='white')
```



How does the behaviour of the learning algorithm changes? Also, why has the response curve changed?

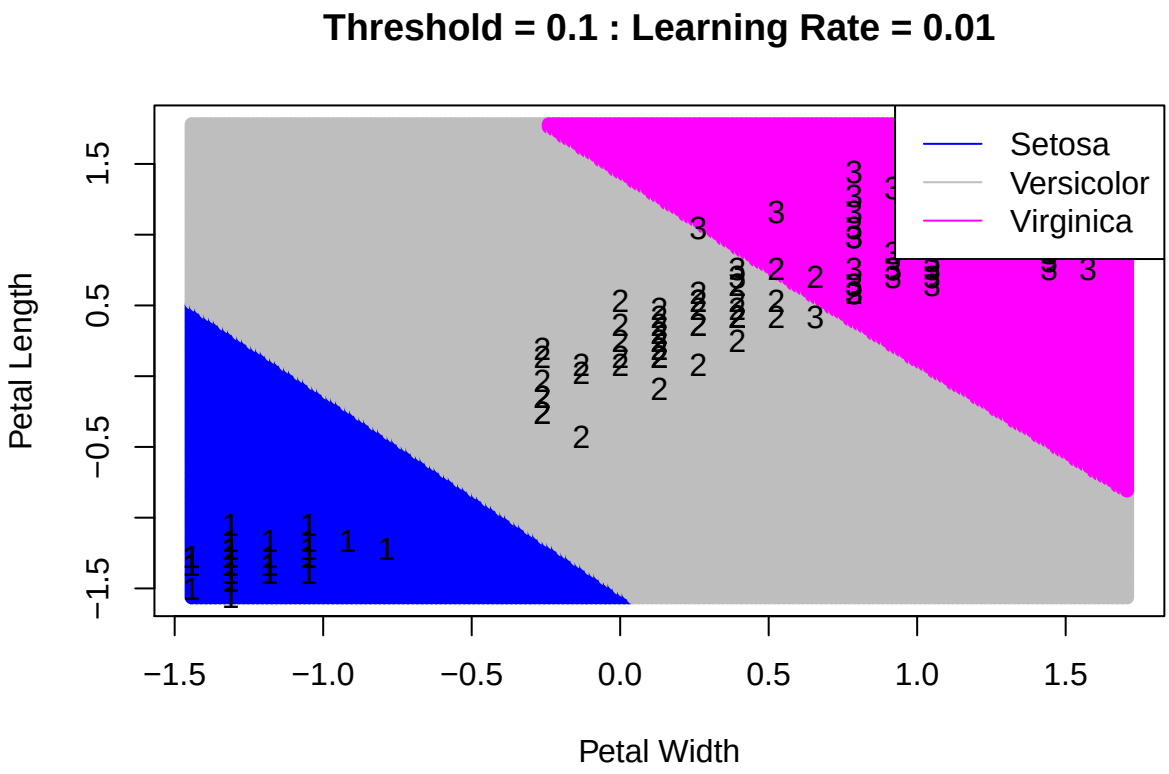
Multiple possible reasons:

- Are the starting parameters the same or random? If random, it's very likely they are just converging to different points.
- Could also be that there is no Convergence at all, but this is unlikely since there is a clear cut.
  - i.e Same minima with different magnitudes of closeness. This seems most likely because the results are still extremely similar.
- Could also be that the convergence criteria is met at different points (just just converged vs very close to complete convergence).

Gradient descent seeks a coordinate in the parameter space which the objective (penalized obj. if regularization is present) reaches a minimum. Gradient descent steps discretely during this search so depending on both the initial values (where you start to search), the step-size pars, and the stopping criteria, the coordinaate at which the search terminates may be different. In consequence, the fitted model will be different. It may produce similar objective values but the 'solution' corresponds to a different configuration of the model, hence the discrepancy between the response curves.

Problem 3: `Early stopping` is said to be a form of regularisation. Re-run the analysis (all else equal) using `learning-rate = 0.01` and `threshold = 0.1`. Do you concur that the resulting fitted model is simpler than what you would've observed under less stringent stopping criterion?

```
model3 = neuralnet(Species~ Petal.Length + Petal.Width, data=newData,
                   hidden = c(5),
                   threshold = 0.1,
                   stepmax = 10^6,
                   learningrate = 0.01,
                   algorithm = 'backprop',
                   act.fct = 'logistic',
                   err.fct = 'ce',
                   linear.output = FALSE)

pred2 = predict(model3, Lat)
clss2 = apply(pred2, 1, which.max)

plot(x1~x2, pch=16, col=cols[clss2], xlab = "Petal Width",
     ylab = "Petal Length", main = "Threshold = 0.1 : Learning Rate = 0.01")

text(newData$Petal.Length~newData$Petal.Width, labels= as.numeric(newData$Species))

legend('topright', lty=1, legend = c("Setosa", "Versicolor", "Virginica"),
       col=c('blue', 'gray', 'magenta'), bg='white')
```



We can concur that the resulted fitted model is much simpler just by looking at the response curve of the model with `learningrate = 0.01` and `threshold = 0.1` compared to all the others above.