STAT40970 – Machine Learning & A.I.

Assignment2

21201588 - Denis O'Riordan

Exercise 1

Question 1

The model in this question has a depth of two corresponding to the number of hidden layers. The layers have a width of three and two units for the first and second layers respectively.

It has two input nodes, three bias terms, one for each hidden layer and the output layer. There is one node in the output layer.

The total number of parameters of the neural network with $V+1$ input units (including intercept), G hidden layers, H_l+1 hidden units in layer l (including intercept) and K output units is given by the following formula:

$$VH_1 + \sum_{l=1}^{G-1} H_l H_{(l+1)} + H_G K + \sum_{l=1}^{G} H_l + K$$

Substituting the values for the model in question into the above question giving us `2*3 + 3*2 + 2*1 + 3 + 2 + 1` $= 20$. 20 is the total number of parameters of the network.

Question 2

Let h_j^i denote the j^{th} node of the i^{th} layer of the corresponding multiple layer neural network.

The forward generating process for a multilayer neural network is given by the following formulas.

$$h_k^1 = f\left(\sum_{k=0}^{V} w_{kj}^{(1)} x_k\right)$$

$$h_k^{l+1} = f\left(\sum_{k=0}^{H_l} w_{kj}^{(l+1)} h_j^l\right)$$

$$y = g\left(\sum_{k=0}^{H_G} w_k^{(G)} h_k^G\right)$$

Where f(), g() are the RELU and sigmoid activation functions and w_{jk}^i is the weight coefficient for the j^{th} node of the i^{th} layer for the k^{th} input variable.

```
relu = function(x=0){
  if (x > 0){return(x)}
  else{return(0)}}

sigmoid = function(x=0){
  return(exp(x)/(1 + exp(x)))
}
```

h_1^1
Input observation vector = (0.1, -0.7), corresponding weight terms = (0.6, 0.3), bias term = 1.2
Linear combination of bias, input vector and weights = `1.2 + (0.1*0.6) + (-0.7*0.3)` = 1.05
Activation Function: `relu(1.05)` = 1.05

h_2^1
Input observation vector = (0.1, -0.7), corresponding weight terms = (-0.4, -0.1), bias term = -0.8
Linear combination of bias, input vector and weights = `-0.8  + (0.1*-0.4) +(-0.7*-0.1)` = -0.77
Activation Function: `relu(-0.77)` = 0

h_3^1
Input observation vector = (0.1, -0.7), corresponding weight terms = (0.2, 0.2), bias term = 2.3
Linear combination of bias, input vector and weights = `2.3 + (0.1*0.2) + (-0.7*0.2)` = 2.18
Activation Function: `relu(2.18)` = 2.18

h_1^2
Input observation vector = (1.05, 0 , 2.18), corresponding weight terms = (-0.3,1.3 -0.6), bias term = 0.9
Linear combination of bias, input vector and weights = `0.9 + 1.05*-0.3 + 0*1.3 + 2.18*-0.6` = -0.723
Activation Function: `relu(-0.723)` = 0

h_2^2
Input observation vector = (1.05, 0 , 2.18), corresponding weight terms = (0.7, 0.9, 1.1), bias term = -0.8
Linear combination of bias, input vector and weights = `-0.8 +  1.05*0.7 + 0*0.9 + 2.18*1.1` = 2.333
Activation Function: `relu(2.333)` = 2.333

Output Layer
Input observation vector = (0, 2.333), corresponding weight terms = (0.5, 0.4), bias term = 0.3
Linear combination of bias, input vector and weights = `0.3 +  0*0.5 + 2.333*0.4` = 1.2332
Activation Function: `sigmoid(1.2332)` = 0.7743782

The value of the output unit corresponding to this input vector (0.1, -0.7) is 0.7743782.

Question 3

In a binary classification task,the error function is the cross-entropy:

$$E(w) = -\sum_{i=1}^{N}[y_i \times log(o_i) + (1 - y_i) \times log(1 - o_i)]$$

In this training instance the value of the target variable $y_i = 1$ and the value of the output $o_i = 0.7743782$

Error associated with this training instance = `- (1*log(0.7743782) + (1-1)*log(1- 0.7743782))` = 0.2556949

Exercise 2

1. Number of input units

1024

2. Number of batches processed in each epoch

Batch size = 345 and N = 34500. `34500/345` = 100. There will be 100 batches processed in each epoch.

3. Type of task for which the network is employed

Multi-class classification

4. Type of regularization used.

Early stopping.The accuracy on the validation data is the criterion to be monitored with a patience of 20.

Exercise 3

Question 1

Choice of Parameters
In order to compare like with like, the two neural networks deployed will be identical apart from the addition of a third layer for the second network. The parameters used were found by attempting a range of values and trial-and-error. The following parameters resulted in the optimum multilayer neural network for a neural network with two hidden layers while keeping run-time under 3 minutes. In this scenario, the optimum set of parameters is one which minimizes the error of the validation data and prevent overfitting.

Preprocessing: Input variables normalized (variable mean = 0 and standard deviation = 1)
Batch Size & Epochs: 150 (N*0.0033) batches. 150 Epochs (w/ Early Stopping)
Regularization: Weight Decay on Hidden Layers 1 and 2 with a $\lambda = 0.01$ to shrink the magnitude of the connections
Early Stopping: Early Stopping on validation accuracy with a patience of 20 to prevent overfitting
Number of Nodes: Hidden Layer 1: 72. Hidden Layer 2: 36. Hidden Layer 3 (**only for model with three hidden layers**): 18. Output Layer: 5
Activation functions: Hidden Layers 1,2 & 3: RELU. Output Layer: Softmax
Optimization method: Adam algorithm (adds momentum and with an additional decaying term) with default learning rate parameters

```
# load data/library
load("C:/Users/denis/Documents/STAT40970 - Machine Learning & AI/Assignment2/data_bats_call.Rdata")
library(keras)

# y
# subset target variable
y <- data[, 1]
# numerically code target variable Emballonuridae=0, Molossidae = 1 etc.
y <- ifelse(y == "Molossidae",1,
            ifelse(y == "Mormoopidae",2,
                   ifelse(y == "Phyllostomidae",3,
                          ifelse(y == "Vespertilionidae",4,0))))
# convert to categorical variable
y <- to_categorical(y)
```

```
## Loaded Tensorflow version 2.8.0
```

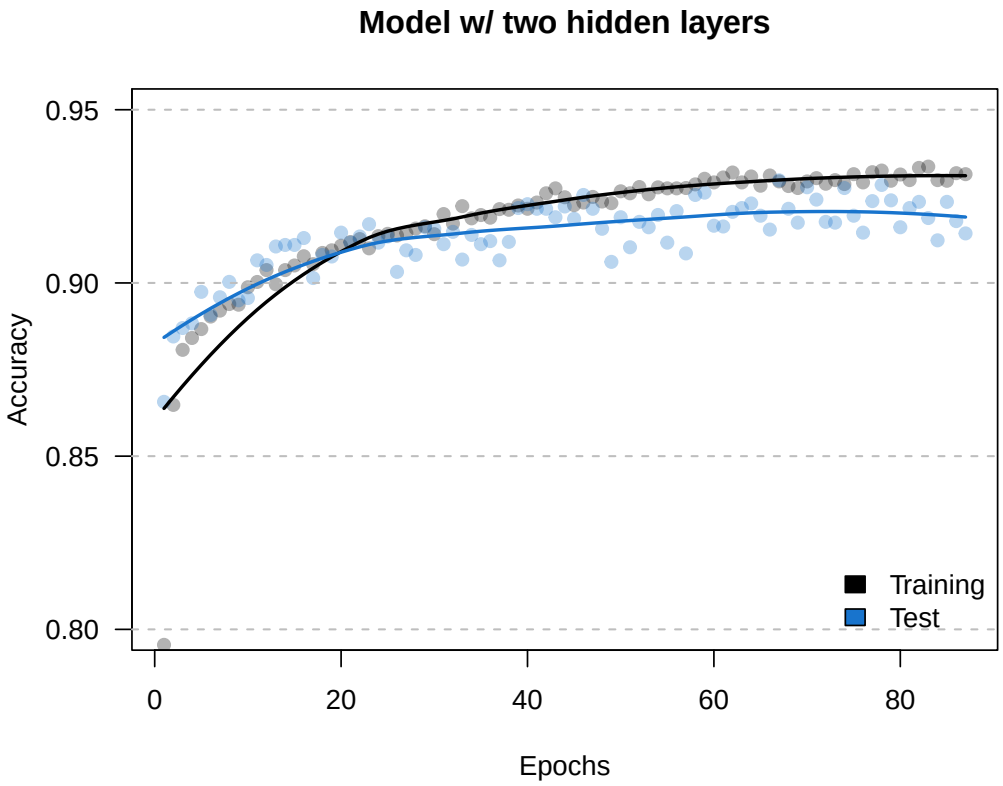
```
#repeat above for test data
y_test <- data_test[,1]
y_test <- ifelse(y_test == "Molossidae",1,
                 ifelse(y_test == "Mormoopidae",2,
                        ifelse(y_test == "Phyllostomidae",3,
                               ifelse(y_test == "Vespertilionidae",4,0))))
y_test <- to_categorical(y_test)

# x
# subset predictor variable
x <- data[,-1]
x_test <- data_test[,-1]

# range normalaization ~N(0,1)
```

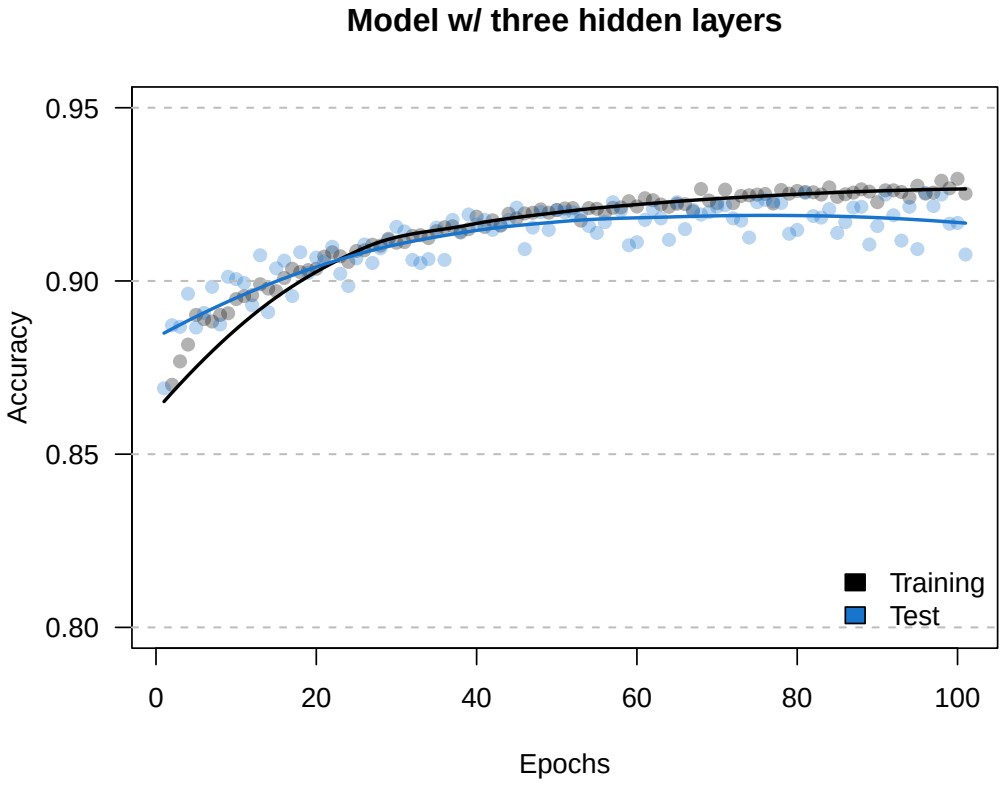
```r
x <- scale(x)
x_test <- scale(x_test)

# split to training and validation (70:30)
N <- nrow(x)
t <- round(N*0.7)
v <- round(N*0.3)
# set seed as year for purpose of replicating results upon different RMD evaluations
set.seed(2022)
train <- sample(1:N, t)
val <- setdiff(1:N, train)

x_train = x[train,]
x_val = x[val,]
y_train = y[train,]
y_val = y[val,]

# input shape
V <- ncol(x_train)

#batch size
N <- nrow(x_train)
bs = round(N*0.0033)

# function to model results
smooth_line <- function(y) {
  x <- 1:length(y)
  out <- predict( loess(y ~ x) )
  return(out)
}
# some colours will be used later
cols <- c("black", "dodgerblue3", "gray50", "deepskyblue2")


######################################################################
####################  model - two hidden layers ####################
######################################################################

model2 <- keras_model_sequential() %>%
  # hidden layer 1 - 72 units w/ weight decay
  layer_dense(units = 72, activation = "relu", input_shape = V,
              kernel_regularizer = regularizer_l2(l = 0.01)) %>%
  # hidden layer 2  - 36 units w/ weight decay
  layer_dense(units = 36, activation = "relu",
              kernel_regularizer = regularizer_l2(l = 0.01)) %>%
  # output layer - 5 units
  layer_dense(units = 5, activation = "softmax") %>%
  # set optimizer
  compile(
    loss = "categorical_crossentropy", metrics = "accuracy",
    optimizer = optimizer_adam(),
  )
```

```r
# fit the model on the training data
# and evaluate on test data at each epoch
fit2 <- model2 %>% fit(
  x = x_train, y = y_train,
  validation_data = list(x_val, y_val),
  # epoch/batch size
  epochs = 150,
  batch_size = bs,
  # dont graph progress
  verbose = 0,
  # early stopping
  callbacks = list(
    callback_early_stopping(monitor = "val_accuracy", patience = 20)
  )
)

# store relevant results
out2 <- cbind(fit2$metrics$accuracy, fit2$metrics$val_accuracy)


######################################################################
##################  model - three hidden layers ######################
######################################################################

model3 <- keras_model_sequential() %>%
  # hidden layer 2  - 72 units w/ weight decay
  layer_dense(units = 72, activation = "relu", input_shape = V,
              kernel_regularizer = regularizer_l2(l = 0.01)) %>%
  # hidden layer 2  - 36 units w/ weight decay
  layer_dense(units = 36, activation = "relu",
              kernel_regularizer = regularizer_l2(l = 0.01)) %>%
  # hidden layer 2  - 18 units w/ weight decay
  layer_dense(units = 18, activation = "relu",
              kernel_regularizer = regularizer_l2(l = 0.01)) %>%
  # output layer - 5 units
  layer_dense(units = 5, activation = "softmax") %>%
  # set optimizer
  compile(
    loss = "categorical_crossentropy", metrics = "accuracy",
    optimizer = optimizer_adam(),
  )

# fit the model on the training data
# and evaluate on test data at each epoch
fit3 <- model3 %>% fit(
  x = x_train, y = y_train,
  validation_data = list(x_val, y_val),
  # epoch/batch size
  epochs = 150,
  batch_size = bs,
  # dont graph progress
  verbose = 0,
  # early stopping
```

```
  callbacks = list(
    callback_early_stopping(monitor = "val_accuracy", patience = 20)
  )
)

#store results
out3 <- cbind(fit3$metrics$accuracy, fit3$metrics$val_accuracy)
```

```
# plot results
par(mfrow=c(2,1))
# two layer neural network
matplot(out2, pch = 19, ylab = "Accuracy", xlab = "Epochs",
        col = adjustcolor(cols[1:2], 0.3), main = "Model w/ two hidden layers",ylim = c(0.8, 0.95), yax
# custom axis and gridlines
axis(2, at = seq(0.8, 1, by=0.05), las=1)
grid(nx = NA, ny = NULL, lty = 2, col = "gray",  lwd = 1.2)
# smooth line
matlines(apply(out2, 2, smooth_line), lty = 1, col = cols[1:2], lwd = 2)
legend("bottomright", legend = c("Training", "Test"),
       fill = cols[1:2], bty = "n")

# three layer neural network
matplot(out3, pch = 19, ylab = "Accuracy", xlab = "Epochs",
        col = adjustcolor(cols[1:2], 0.3), main = "Model w/ three hidden layers",
        ylim = c(0.8, 0.95), yaxt='n' )
# custom axis and gridlines
axis(2, at = seq(0.8, 1, by=0.05), las=1)
grid(nx = NA, ny = NULL, lty = 2, col = "gray",  lwd = 1.2)
# smooth line
matlines(apply(out3, 2, smooth_line), lty = 1, col = cols[1:2], lwd = 2)
legend("bottomright", legend = c("Training", "Test"),
       fill = cols[1:2], bty = "n")
```


Model w/ two hidden layers
0.95
0.90
0.85
0.80
Accuracy
0
20
40
60
80
Epochs
Training
Test


Model w/ three hidden layers
0.95
0.90
0.85
0.80
Accuracy
0
20
40
60
80
100
Epochs
Training
Test

```r
# summary validation accuracy for model w/ two hidden layers
summary(out2[,2])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.8657  0.9099  0.9156  0.9133  0.9211  0.9296
```

```r
# summary validation accuracy for model w/ three hidden layers
summary(out3[,2])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.8690  0.9065  0.9150  0.9120  0.9198  0.9254
```

Results vary slightly between each time the models are run (using RMarkdown).

In each of the six summary statistics, the model with two hidden layers (model2) has recorded a better validation performance than the corresponding figure for the model with three hidden layers (model3). In each instance the difference is less than a percent so it would be unwise to say that model2 is a far better model than model3. However, this it is indicative of a scenario where model3 is certainly not greater than model2, despite it's extra hidden layer.

Both models perform reasonably well at predicting the validation data. Over 75% of the epochs for each model recorded a validation accuracy of 0.9 or greater. Validation accuracies for an epoch of over 0.92 was recorded for both models also. Mean validation accuracies of 0.9133312 and 0.9119748 , for model2 and model3 respectively, further add to their impressive predictive performances.

Both plots of the validation accuracy over the epochs are similar. Models quickly reach validation accuracy over 0.85 and continue increasing more gradually until they level off between 0.90-0.95. The gap between test data accuracy and training data accuracy is slightly narrower for model3 than model2. This would indicate a better generalisability of the model compared to model2 which is an important feature of any neural network. However the difference between training and test accuracy for model2 is not significant enough to display signs of significant overfitting.

Question 2

As seen in the plots and figures above, there is little immediately discernible difference between the predictive performance of the model with two layers (model2) and the model with three layers (model3) on the validation data. These results indicate that adding one extra hidden layer to the network architecture does not lead to an improvement in the predictive performance.

This is likely due to the parameters and hyperparameters being tuned to optimize performance. If both models were not tuned and misspecified it is more likely the model with three layers would lead to an improvement in the predictive performance.

It is seen that model2 stopped training before model3. This can be seen as model2 learning faster than model3, given the similarity in results and that model3 was also stopped before the 150^{th} epoch. This would further contribute to the argument that a third layer of the neural network is not needed.

Empirical results show that deeper networks generalize better but it is possible in this scenario that two layers is sufficient for producing acceptable result for predicting the type of bat family a call belongs to and that three layers will not produce significantly improved results and will only increase computational resources and increase run-time.

The number of hidden layers is only one parameter of multi-layer neural networks. There are many, many others which could be altered in countless ways to increase predictive performance. Solely and arbitrarily increasing the number of layers in a network is not a certain way to improve predictive performance.

Question 3

```
# convert target variable back to categorical
y_raw <- max.col(y_test) - 1
# network with two layers
# predicted classes
class_2 <- model2 %>% predict(x_test) %>% max.col()
tab2 <- table(y_raw, class_2)
# table for model with two layers
tab2
```

```
##      class_2
## y_raw    1    2    3    4    5
##     0  644    5   28    0    3
##     1   63  470    0    0  187
##     2    5    0  532    6    1
##     3    1    0   14  930   89
##     4    5   21    5  144 3291
```

```
# network with three layers
# predicted classes
class_3 <- model3 %>% predict(x_test) %>% max.col()
tab3 <- table(y_raw, class_3)
# table for model with three layers
tab3
```

```
##      class_3
## y_raw    1    2    3    4    5
##     0  567   68   22    1   22
##     1   10  476    0    0  234
##     2   14    0  521    5    4
##     3    0    0   16  921   97
##     4    0   16    0  122 3328
```

The network with two hidden layers has an overall accuracy for the test data of `sum( diag(tab2) ) / sum(tab2)` = 0.9104593. The network with three hidden layers has an overall accuracy for the test data of `sum( diag(tab3))/sum(tab3)` = 0.9020795.
Empirical results show that deeper networks generalize better but in this scenario a well-tuned two layer neural network performs slightly better than a three layered neural network with identical parameters.

Referring back to the numerical encoding deployed in Q1, the family Mormoopidae were assigned the value 2 i.e. third row, third column (**note:** the predicted classes were assigned labels 1-5 and due to time constraints I was unable to alter the range to 0-4 to match the original labels, apologies for that)

The network with two hidden layers predicted bats from the family Mormoopidae with an accuracy of `tab2[3,3]/sum(tab2[3,])` = 0.9779412. Of all the predictions of the family Mormoopidae using the test data, `tab2[3,3]/sum(tab2[,3])` = 0.9188256 were found to be correct.

The network with three hidden layers predicted bats from the family Mormoopidae with an accuracy of `tab3[3,3]/sum(tab3[3,])` = 0.9577206. Of all the predictions of the family Mormoopidae using the test data, `tab3[3,3]/sum(tab3[,3])` = 0.9320215 were found to be correct.

Curiously, the two statistics above don't necessarily agree over which is the better model. model2 identified more of the the Mormoopidae bats but model2 also predicted more bats to be Mormoopidae than model3,

with model2's predictions less likely to be correct than model3. model2 predicted 579 Mormoopidae bats while model3 only predicted 559 Mormoopidae bats.

However, the magnitudes of the differences between the two models for the above statistics would indicate model2 marginally provides the best accuracy at predicting calls from the family Mormoopidae on the test data.