Predict the type of electroencephalography (EEG) activity using DNN

Clementine Surya & 22200226

2023-07-03

```
# Load necessary package
library(dplyr)
library(keras)
library(ggplot2)
```

Introduction

In this report, I deploy a neural network with 2 hidden layers and a neural network with 3 hidden layers and compare the performances of these two networks at predicting the type of condition (and area of the brain) an EEG signal belongs to.

First, I load and prepare the data with below code:

```
# Load the data
load("data_epileptic.RData")

# Data preparation
data <- cbind(x,y)
data <- as.matrix(data)
dimnames(data) <- NULL
data_test <- cbind(x_test, y_test)
data_test <- as.matrix(data_test)
dimnames(data_test) <- NULL

# Scale the data
data[,1:178] <- scale(data[,1:178])
data_test[,1:178] <- scale(data_test[,1:178])

# Change the target variable as numeric
data[,179] <- as.numeric(data[,179]) - 1
data_test[,179] <- as.numeric(data_test[,179]) - 1

# Data Partition
set.seed(1234) # for reproducibility
ind <- sample(2, nrow(data), replace = T, prob = c(0.7,0.3))
training <- data[ind == 1, 1:178]
trainingtarget <- data[ind==1,179]
val <- data[ind == 2, 1:178]
valtarget <- data[ind==2,179]
testing <- data_test[,1:178]
testtarget <- data_test[,179]

# One hot encoding the target variable
trainLabels <- to_categorical(trainingtarget)
valLabels <- to_categorical(valtarget)
testLabels <- to_categorical(testtarget)

V <- ncol(x) # Compute number of column = 178
```

Then, these are the function and code to be used for plotting that will be used later:

```
# To add a smooth line to points (will be used later)
smooth_line <- function(y) {
  x <- 1:length(y)
  out <- predict( loess(y ~ x) )
  return(out)
}

# Some colors will be used later
cols <- c("black", "dodgerblue3", "gray50", "deepskyblue2")
```

I start to build the Neural Network:
The model has 178 input units to accommodate the 178 features in the dataset. The objective is to perform multiclass classification with 5 classes, and therefore, a softmax activation function is employed in the output layer, along with categorical cross-entropy as the loss function. ReLU activation is chosen for each hidden layer. RMSProp optimization method is utilized, with an adaptive learning rate. Early stopping is employed as a regularization technique to prevent overfitting. Despite using these techniques, the results indicate the presence of overfitting patterns.

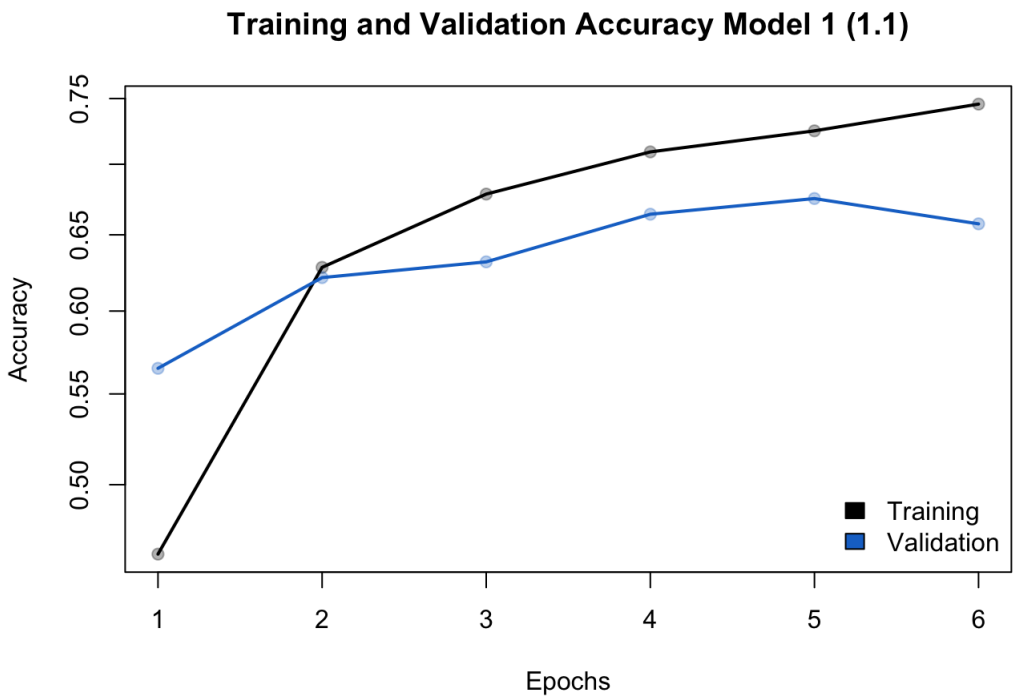
Model 1: 2 Hidden Layers Neural Network

For Model 1, I selected 150 as the number of hidden units in the first hidden layer based on a tuning process that I conducted separately (see source code in the appendix). To prevent overfitting, I decreased the number of units in the subsequent hidden layer by half. The resulting model with the optimized number of neurons in the hidden layer is presented below.

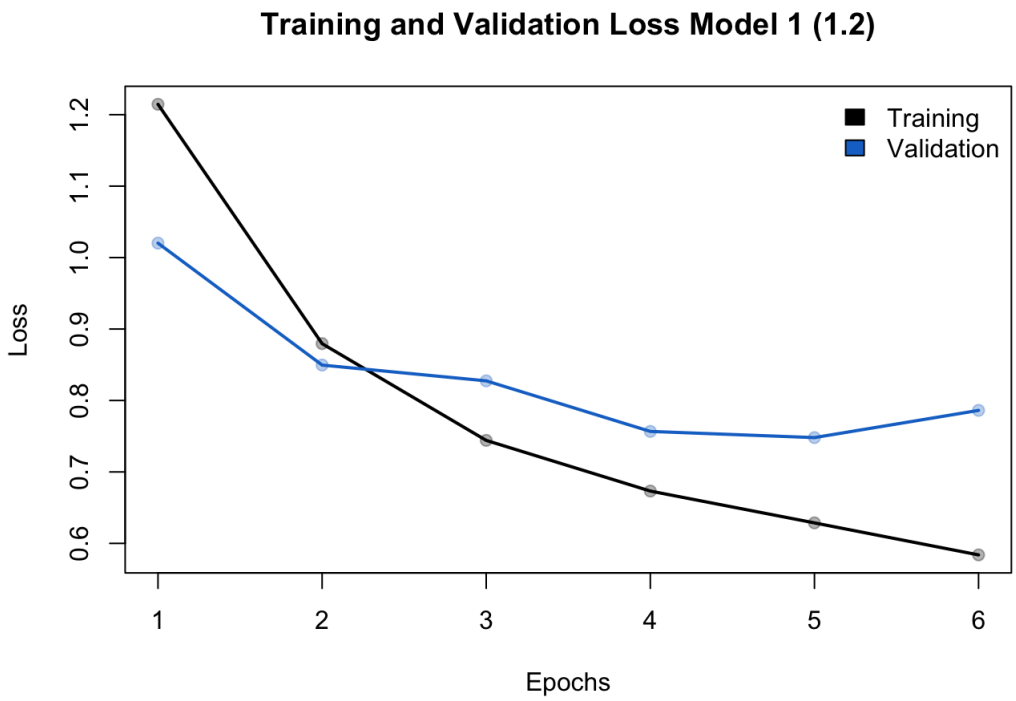
```
# Create sequential model
model_1 <- keras_model_sequential() %>%
  layer_dense(units = 150, activation = "relu", input_shape = V) %>%
  layer_dense(units = 150/2, activation = "relu") %>%
  layer_dense(units = 5, activation = "softmax") %>%
  compile(loss = "categorical_crossentropy", metrics = "accuracy",
          optimizer = optimizer_rmsprop())

# Fit the model on the training data and evaluate on val data at each epoch
fit_1 <- model_1 %>% fit(
  x = training, y = trainLabels,
  validation_data = list(val, valLabels),
  epochs = 100,
  batch_size = 32,
  verbose = 1,
  callbacks = list(
    callback_early_stopping(monitor = "val_accuracy", patience = 1)))
```

Here, I check the performance of Model 1 by showing the training and validation accuracy and loss.

```
# Create accuracy plot for training and validation over Epochs for Model 1
acc_1 <- cbind(fit_1$metrics$accuracy,
               fit_1$metrics$val_accuracy)
matplot(acc_1 , pch = 19, ylab = "Accuracy", xlab = "Epochs",
        main = "Training and Validation Accuracy Model 1 (1.1)",
        col = adjustcolor(cols[1:2], 0.3), log = "y") # on log scale
matlines(apply(acc_1, 2, smooth_line), lty = 1, col = cols[1:2], lwd = 2)
legend("bottomright", legend = c("Training", "Validation"),
       fill = cols[1:2], bty = "n")
```



```
# Create loss plot for training and validation over Epochs for Model 2
loss_1 <- cbind(fit_1$metrics$loss, fit_1$metrics$val_loss)
matplot(loss_1, pch = 19, ylab = "Loss", xlab = "Epochs",
        main = "Training and Validation Loss Model 1 (1.2)",
        col = adjustcolor(cols[1:2], 0.3))
matlines(apply(loss_1, 2, smooth_line), lty = 1, col = cols[1:2], lwd = 2)
legend("topright", legend = c("Training", "Validation"),
       fill = cols[1:2], bty = "n")
```



Model evaluation for training and validation for Model 1:

```
# Evaluate model 1 with training data
model_1 %>% evaluate(training, trainLabels, verbose = 0)
```

```
##      loss  accuracy
## 0.5548250 0.7639961
```

```
# Evaluate model 1 with validation data
model_1 %>% evaluate(val, valLabels, verbose = 0)
```

```
##      loss  accuracy
## 0.7861530 0.6575342
```

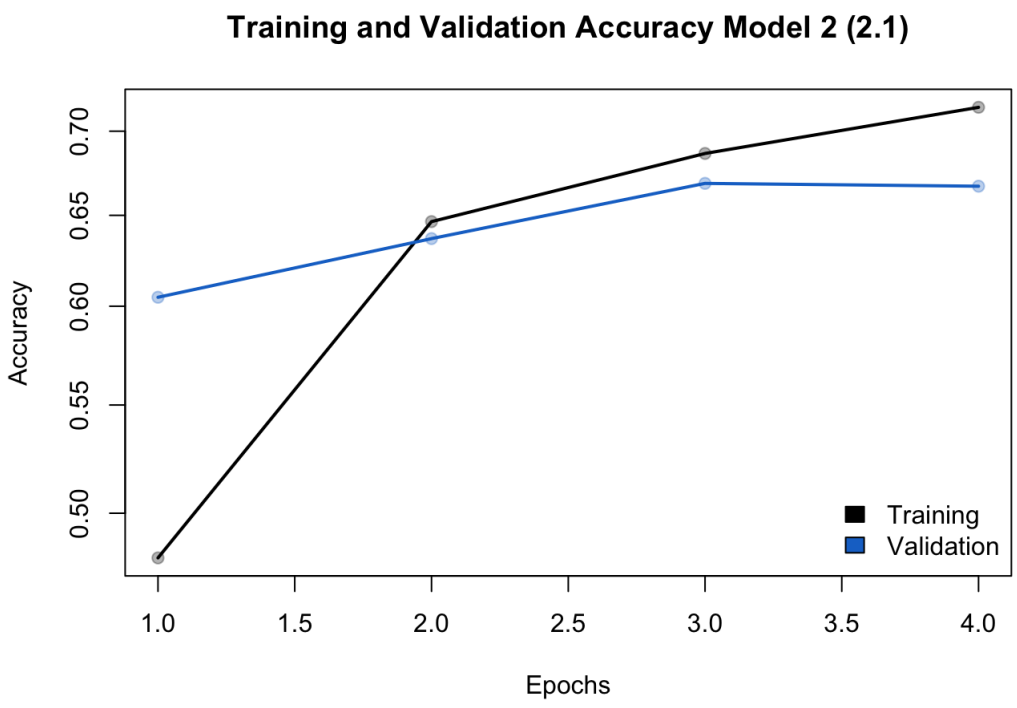
Model 2: 3 Hidden Layers Neural Network

Similar to Model 1, I chose 150 as the number of hidden units in the first hidden layer for Model 2, which was determined through a tuning process detailed in the appendix. To mitigate overfitting, I gradually reduced the number of units in the subsequent hidden layer by half. The resulting model architecture is presented below.

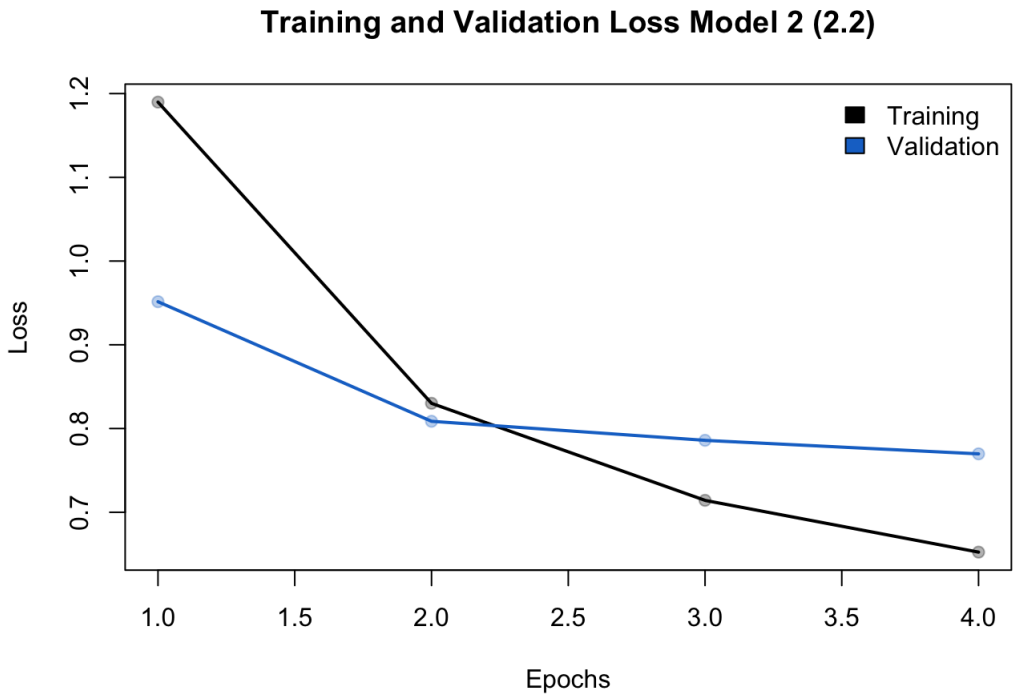
```
# Create sequential model
model_2 <- keras_model_sequential() %>%
  layer_dense(units = 150, activation = "relu", input_shape = V) %>%
  layer_dense(units = 150/2, activation = "relu") %>%
  layer_dense(units = 150/4, activation = "relu") %>%
  layer_dense(units = 5, activation = "softmax") %>%
  compile(loss = "categorical_crossentropy", metrics = "accuracy",
          optimizer = optimizer_rmsprop())

# Fit the model on the training data and evaluate on val data at each epoch
fit_2 <- model_2 %>%
  fit(x = training, y = trainLabels,
      validation_data = list(val, valLabels),
      epochs = 100,
      batch_size = 32,
      verbose = 1,
      callbacks = list(
        callback_early_stopping(monitor = "val_accuracy", patience = 1)))
```

Here, I check the performance of Model 2 by showing the training and validation accuracy and loss.

```
# Create accuracy plot for training and validation over Epochs for Model 2
acc_2 <- cbind(fit_2$metrics$accuracy,
               fit_2$metrics$val_accuracy)
matplot(acc_2 , pch = 19, ylab = "Accuracy", xlab = "Epochs",
         main = "Training and Validation Accuracy Model 2 (2.1)",
        col = adjustcolor(cols[1:2], 0.3), log = "y") # on log scale
matlines(apply(acc_2, 2, smooth_line), lty = 1, col = cols[1:2], lwd = 2)
legend("bottomright", legend = c("Training", "Validation"),
       fill = cols[1:2], bty = "n")
```




```
# create loss plot for training and validation over Epochs for Model 2
loss_2 <- cbind(fit_2$metrics$loss, fit_2$metrics$val_loss)
matplot(loss_2, pch = 19, ylab = "Loss", xlab = "Epochs",
        main = "Training and Validation Loss Model 2 (2.2)",
        col = adjustcolor(cols[1:2], 0.3))
matlines(apply(loss_2, 2, smooth_line), lty = 1, col = cols[1:2], lwd = 2)
legend("topright", legend = c("Training", "Validation"),
       fill = cols[1:2], bty = "n")
```



Model evaluation for training and validation for Model 2:

```
# Evaluate model 2 with training data
model_2 %>% evaluate(training, trainLabels, verbose = 0)
```

```
##      loss  accuracy
## 0.6173733 0.7321454
```

```
# Evaluate model 2 with validation data
model_2 %>% evaluate(val, valLabels, verbose = 0)
```

```
##      loss  accuracy
## 0.7697545 0.6668894
```

Comparison between 2 models

Observing the training versus validation accuracy and loss plots for both Model 1 and Model 2 above, it is apparent that there is evidence of overfitting. This is evident as the validation accuracy does not improve while the training accuracy continues to increase. The loss curves also display some overfitting, as the gap between validation and training loss seems to increase with the number of epochs.

Moreover, the evaluation results for both models show much higher accuracy in the training set than what was observed on the validation set, further indicating overfitting.

Regarding the accuracy of predictive performance on the validation set, Model 1 and Model 2 exhibit similar levels of accuracy when compared to each other.

Here, I compare the performances of these two networks at predicting the type of condition (and area of the brain) an EEG signal belongs to based on the performance on validation data:

```
# Check predictive performance of model 1 to the validation data
prob_val_1 <- model_1 %>% predict(val) # generate predicted values for the validation data using model 1
pred_val_1 <- model_1 %>% predict(val) %>% max.col() # convert the predicted probabilities to class labels
(according to the standard maximum a posteriori rule)
tab_val_1  <- table(valtarget+1, pred_val_1) # create confusion matrix to compare validation predicted class
labels to the actual class labels
tab_val_1 <- cbind(tab_val_1, cl_sens = diag(tab_val_1)/rowSums(tab_val_1)) # compute class sensitivity
names( dimnames(tab_val_1) ) <- c("class_test", "class_test_hat") # for readability
tab_val_1 # show the confusion matrix
```

```
##           class_test_hat
## class_test   1   2   3   4   5   cl_sens
##          1 534  31   4  30   2 0.8885191
##          2  22 192 347   6  41 0.3157895
##          3  13 109 395  14  68 0.6594324
##          4   4  11  10 485  82 0.8192568
##          5   0  40  80 111 362 0.6104553
```

```
# Check predictive performance of model 2 to the validation data
prob_val_2 <- model_2 %>% predict(val) # generate predicted values for the validation data using model 2
pred_val_2 <- model_2 %>% predict(val) %>% max.col() # convert the predicted probabilities to class labels (according to the standard maximum a posteriori rule)
tab_val_2  <- table(valtarget+1, pred_val_2) # create confusion matrix to compare validation predicted class labels to the actual class labels
tab_val_2 <- cbind(tab_val_2, cl_sens = diag(tab_val_2)/rowSums(tab_val_2)) # compute sensitivity
names( dimnames(tab_val_2) ) <- c("class_test", "class_test_hat") # for readability
tab_val_2 # show the confusion matrix
```

```
##           class_test_hat
## class_test   1   2   3   4   5   cl_sens
##          1 571  23   1   6   0 0.9500832
##          2  15 429 112   9  43 0.7055921
##          3  15 326 187  23  48 0.3121870
##          4   7  26   5 473  81 0.7989865
##          5   0 103  33 121 336 0.5666105
```

The performance of the models varies across the different classes. We can see from the confusion matrix above for the sensitivity measures for each classes for both models with:
1 - Epileptic seizure activity, 2 - Patient with tumor formation, EEG activity recorded on healthy area during epilepsy-free interval, 3 - Patient with tumor formation, EEG activity recorded on tumor location area during epilepsy-free interval, 4 - EEG activity of healthy subject with eyes closed, 5 - EEG activity of healthy subject with eyes open.

2. Does adding one extra hidden layer to the network architecture lead to an improvement in the predictive performance? Discuss briefly.

As we can see from the model evaluation results above, adding one extra hidden layer to the network architecture **does not** lead to an improvement in the predictive performance and it may even lead to overfitting. This is because the more complex the model, the more parameters it has to learn from the data, which can increase its capacity to fit the training data too closely. Furthemore, with more layers, the network can learn more complex representations of the data, which may capture both the relevant and irrelevant patterns in the training set. This can cause the model to become too specialized to the training data, and it may not generalize well to new unseen data.

3. Evaluate the test accuracy of the two deep neural networks and comment briefly on their performance. Which of the two networks provide the best accuracy at predicting EEG signals of patients with tumor formations on the test data? Motivate.

In this part, I evaluate the test accuracy of both models implemented above:

```
# Evaluate model with testing data
model_1 %>% evaluate(testing, testLabels, verbose = 0) # test
```

```
##      loss  accuracy
## 0.8324274 0.6478261
```

```
# Check predictive performance of model 1 to the test data
prob_test_1 <- model_1 %>% predict(testing) # generate predicted values for the test data using model 1
pred_test_1 <- model_1 %>% predict(testing) %>% max.col() # convert the predicted probabilities to class labels (according to the standard maximum a posteriori rule)
tab_test_1  <- table(testtarget+1, pred_test_1) # create confusion matrix to compare test predicted class labels to the actual class labels
tab_test_1 <- cbind(tab_test_1, cl_acc = diag(tab_test_1)/rowSums(tab_test_1)) # compute class sensitivity
names( dimnames(tab_test_1) ) <- c("class_test", "class_test_hat") # for readability
tab_test_1 # show the confusion matrix
```

```
##           class_test_hat
## class_test   1  2   3   4   5    cl_acc
##          1 240 18   3  11   0 0.8823529
##          2  11 88 151   7  23 0.3142857
##          3   5 50 186   9  26 0.6739130
##          4   0  4   6 229  37 0.8297101
##          5   0 14  32  79 151 0.5471014
```

```
# Evaluate model with testing data
model_2 %>% evaluate(testing, testLabels, verbose = 0)
```

```
##      loss  accuracy
## 0.7832972 0.6384058
```

```
# Check predictive performance of model 2 to the test data
prob_test_2 <- model_2 %>% predict(testing) # generate predicted values for the test data using model 2
pred_test_2 <- model_2 %>% predict(testing) %>% max.col() # convert the predicted probabilities to class lab
els (according to the standard maximum a posteriori rule)
tab_test_2  <- table(testtarget+1, pred_test_2) # create confusion matrix to compare test predicted class la
bels to the actual class labels
tab_test_2 <- cbind(tab_test_2, cl_sens = diag(tab_test_2 )/rowSums(tab_test_2 )) # compute class sensitivit
y
names( dimnames(tab_test_2) ) <- c("class_test", "class_test_hat") # for readability
tab_test_2 # show the confusion matrix
```

```
##           class_test_hat
## class_test   1   2  3   4   5   cl_sens
##          1 253  15  1   3   0 0.9301471
##          2  14 174 55   6  31 0.6214286
##          3   4 168 73  15  16 0.2644928
##          4   1  10  1 231  33 0.8369565
##          5   0  37 13  76 150 0.5434783
```

From the result, we can see that both models obtain similar result for prediction accuracy in the test set

Overall, the accuracies achieved in predicting the test set are quite low for both models, indicating that they are not effective in accurately detecting EEG activities related to different subjects, conditions, and areas of the brain.