Advanced Machine Learning
Assignment 1
Reginald Mordi
3/7/2021

```r
library(ggplot2)
library(tidyverse)
library(tensorflow)
library(cowplot)
library(keras)

# Importing IMDB movie reviews dataset and we only keep the top 10,000 most frequently occurring words in
 the training data.
imdb <- dataset_imdb(num_words = 10000)
c(c(train_data, train_labels), c(test_data, test_labels)) %<-% imdb

# Converting my data to binary data
vectorize_sequences <- function(sequences, dimention = 10000) {
    # Create an all-zero matrix of shape (len(sequences), dimension)
  results <- matrix(0, nrow = length(sequences), ncol= dimention)
  for(i in 1:length(sequences))
        # Sets specific indices of results[i] to 1s
  results[i, sequences[[i]]] <- 1
  results
}
x_train <- vectorize_sequences(train_data)  # vectorized training data
x_test <- vectorize_sequences(test_data)    # vectorized test data
# vectoring the labels
y_train <- as.numeric(train_labels)
y_test <- as.numeric(test_labels)
# structure of the vectorized samples
str(x_train[1,])

# Validating my approach
## Setting apart 10,000 samples from the original training data for validation
val_indices <- 1:10000
x_val <- x_train[val_indices,]
partial_x_train <- x_train[-val_indices,]
y_val <- y_train[val_indices]
partial_y_train <- y_train[-val_indices]
```

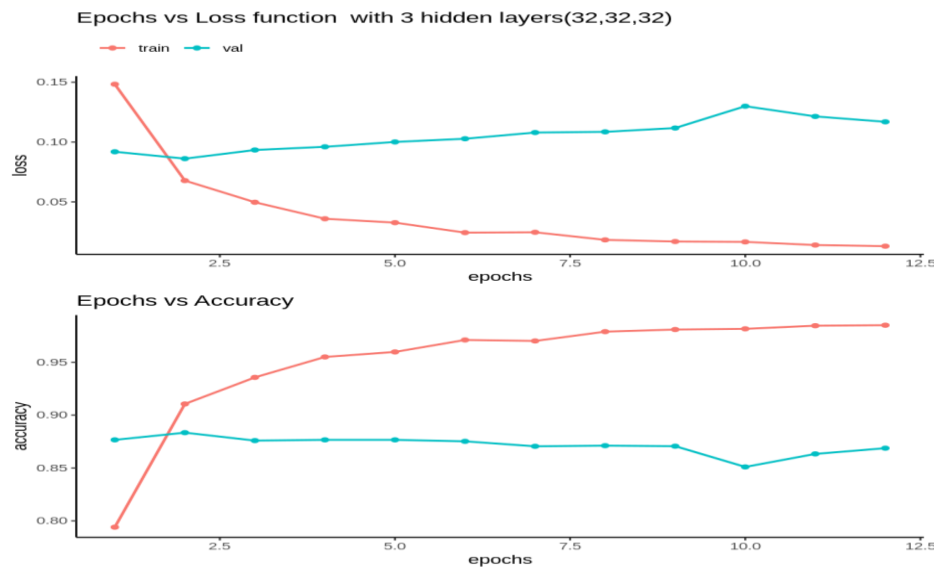
```
num [1:10000] 1 1 0 1 1 1 1 1 1 0 ...
```

```r
model1 <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = "relu", input_shape = c(10000)) %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")
model1 %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy"))
history <- model1 %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 12,
  batch_size = 512,
  validation_data = list(x_val, y_val))
# Building the network
# Using 3 hidden layers with 32, 32, 32 units, tanh activation function, batch size 512, 20 epoch.
model333 <- keras_model_sequential() %>%
  layer_dense(units = 32, activation = "tanh", input_shape = c(10000)) %>%
  layer_dense(units = 32, activation = "tanh") %>%
  layer_dense(units = 32, activation = "tanh") %>%
  layer_dense(units = 1, activation = "sigmoid")
model333 %>% compile(
  optimizer = "rmsprop",
  loss = "mse",
  metrics = c("accuracy"))
history_3_layer32 <- model333 %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 12,
  batch_size = 512,
  validation_data = list(x_val, y_val))

# Let us train a new network from scratch for 4 epochs and then evaluate it on the test data.
model3331 <- keras_model_sequential() %>%
  layer_dense(units = 32, activation = "tanh", input_shape = c(10000)) %>%
  layer_dense(units = 32, activation = "tanh") %>%
  layer_dense(units = 32, activation = "tanh") %>%
  layer_dense(units = 1, activation = "sigmoid")

model3331 %>% compile(
  optimizer = "rmsprop",
  loss = "mse",
  metrics = c("accuracy"))

model3331 %>% fit(x_train, y_train, epochs = 5, batch_size = 512)
```

```
results1 <- model3331 %>% evaluate(x_test, y_test)
results1

loss        0.105736047029495
accuracy    0.867999970912933
```

***** Having a higher-dimensional representation space (more hidden units) allows the network to learn more complex representations, but it makes the network more computationally expensive and may lead to learning unwanted patterns, patterns that will improve performance on the training data but not on the test data.**

```
# Visualizing the model331 output of loss function and accuracy
model331.df <- as.data.frame(history_3_layer32$metrics)
names(model331.df) <- c("train_loss","train_accuracy","val_loss","val_accuracy")
model331.df <- model331.df %>% mutate(epochs=1:n()) %>% gather("split","values",-
epochs) %>% separate(split,c("split","metric")) %>% spread(metric,values)
p1<-
ggplot(model331.df) + geom_line(aes(x=epochs,y=loss,color=split),size=0.8)+geom_point(aes(x=epochs,y=lo
ss,color=factor(split)),size=1.5)+ggtitle("Epochs vs Loss function  with 3 hidden layers(32,32,32)")+theme(pa
nel.grid = element_blank(),panel.background = element_blank())+theme_classic()+theme(legend.position = 'to
p',legend.justification = 'left',legend.title = element_blank())
p2<-
ggplot(model331.df) + geom_line(aes(x=epochs,y=accuracy,color=split),size=0.8,show.legend = F)+geom_po
int(aes(x=epochs,y=accuracy,color=split),size=1.5,show.legend = F)+ggtitle("Epochs vs Accuracy")+theme(pa
nel.grid = element_blank(),panel.background = element_blank())+theme_classic()
plot_grid(p1,p2,nrow = 2)
```



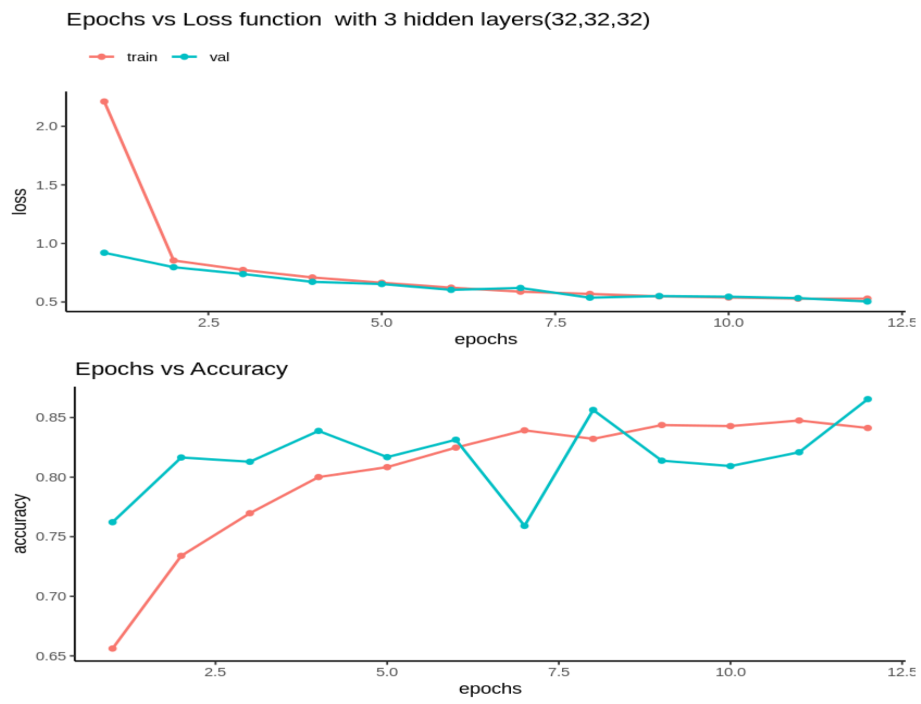
```
# Using 3 hidden layers with 64, 32, 16 units, tanh activation function, batch size 512, 20 epoch.
model632 <- keras_model_sequential() %>%
  layer_dense(units = 64,kernel_regularizer = regularizer_l1(0.001), activation = "tanh",input_shape = c(10000
)) %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 32,kernel_regularizer = regularizer_l1(0.001), activation = "tanh") %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 16,kernel_regularizer = regularizer_l1(0.001), activation = "tanh") %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 1, activation = "sigmoid")
model632 %>% compile(
  optimizer = "rmsprop",
  loss = "mse",
  metrics = c("accuracy"))
history_3 <- model632 %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 12,
  batch_size = 512,
  validation_data = list(x_val, y_val))
```

Let us add a dropout layer in our IMDB network to see how well they do at reducing overfitting using 32 units.

```
# Training a new network from scratch for 4 epochs and then evaluate it on the test data.
model6313 <- keras_model_sequential() %>%
  layer_dense(units = 64, activation = "tanh",input_shape = c(10000)) %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 32, activation = "tanh") %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 16, activation = "tanh") %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 1, activation = "sigmoid")
model6313 %>% compile(
  optimizer = "rmsprop",
  loss = "mse",
  metrics = c("accuracy"))
model6313 %>% fit(x_train, y_train, epochs = 4, batch_size = 512)
results4 <- model6313 %>% evaluate(x_test, y_test)
results4
```

```
loss        0.0939082130789757
accuracy    0.883520007133484
```

```
# Visualizing the model6313 output of loss function and accuracy
model6313.df <- as.data.frame(history_3$metrics)
names(model6313.df) <- c("train_loss","train_accuracy","val_loss","val_accuracy")
model6313.df <- model6313.df %>% mutate(epochs=1:n()) %>% gather("split","values",-
epochs) %>% separate(split,c("split","metric")) %>% spread(metric,values)
p1<-
ggplot(model6313.df) + geom_line(aes(x=epochs,y=loss,color=split),size=0.8)+geom_point(aes(x=epochs,y=l
oss,color=factor(split)),size=1.5)+ggtitle("Epochs vs Loss function  with 3 hidden layers(32,32,32)")+theme(p
anel.grid = element_blank(),panel.background = element_blank())+theme_classic()+theme(legend.position = 't
op',legend.justification = 'left',legend.title = element_blank())
p2<-
ggplot(model6313.df) + geom_line(aes(x=epochs,y=accuracy,color=split),size=0.8,show.legend = F)+geom_p
oint(aes(x=epochs,y=accuracy,color=split),size=1.5,show.legend = F)+ggtitle("Epochs vs Accuracy")+theme(
panel.grid = element_blank(),panel.background = element_blank())+theme_classic()
plot_grid(p1,p2,nrow = 2)
```



Conclusion:

We notice that almost all models reach a high validation accuracy at epoch 2, 3, or 4, then the val. Acc. It starts to drop down as the model starts to overfit for the training data.
The more extensive network (3 layers) starts overfitting almost right away, after just one epoch, and overfits much more severely. Its validation loss is also noisier.

As you can see, the more extensive network gets its training loss near zero very quickly. The more capacity the network has, the quicker it will model the training data (resulting in a low training loss). The more susceptible it is to overfitting (resulting in a significant difference between the training and validation loss).

The simplest way to prevent overfitting is to reduce the size of the model, i.e., the number of learnable parameters in the model (determined by the number of layers and the number of units per layer). Intuitively, a model with more parameters will have more "memorization capacity" and will be able to learn a perfect dictionary easily.

We Observed that the model with three layers (64, 32, 16) units and applying the dropout function tends to perform well in both the validation accuracy consistency and has a high accuracy on test data 88.3 %. Hence this model is my best model.