Assignment 1: Neural Networks (IMDB) - Meister Submission

Contents


```
library(reticulate)
use_virtualenv("r-tensorflow", required = TRUE)

library(tensorflow)
library(keras)
library(dplyr)
library(tidyr)
library(ggplot2)

set.seed(123)
tensorflow::tf$random$set_seed(123)

tf_config()
```

```
## TensorFlow v2.15.1 (~/.virtualenvs/r-tensorflow/lib/python3.11/site-packages/tensorflow)
## Python v3.11 (~/.virtualenvs/r-tensorflow/bin/python)
```

1. Loading and prepare the IMDB data

```
num_words <- 10000
imdb <- dataset_imdb(num_words = num_words)

c(c(train_data, train_labels), c(test_data, test_labels)) %<-% imdb

vectorize_sequences <- function(sequences, dimension = 10000) {
  results <- matrix(0, nrow = length(sequences), ncol = dimension)
  for (i in seq_along(sequences)) {
    idx <- sequences[[i]]
    idx <- idx[idx > 0 & idx <= dimension]
    results[i, idx] <- 1
  }
```

```
  results
}

x_train <- vectorize_sequences(train_data, num_words)
x_test  <- vectorize_sequences(test_data,  num_words)

y_train <- as.numeric(train_labels)
y_test  <- as.numeric(test_labels)

val_n <- 10000
x_val <- x_train[1:val_n, ]
partial_x_train <- x_train[(val_n + 1):nrow(x_train), ]
y_val <- y_train[1:val_n]
partial_y_train <- y_train[(val_n + 1):length(y_train)]
```

2. Helper functions to address hidden layers

```
build_model <- function(num_words = 10000,
                        hidden_layers = 2,
                        units = 16,
                        activation = "relu",
                        loss = "binary_crossentropy",
                        dropout = 0,
                        l2 = 0) {

  model <- keras_model_sequential()

  # First hidden layer
  model %>%
    layer_dense(
      units = units,
      activation = activation,
      input_shape = c(num_words),
      kernel_regularizer = if (l2 > 0) regularizer_l2(l2) else NULL
    )

  if (dropout > 0) {
    model %>% layer_dropout(rate = dropout)
  }

  # Additional hidden layers
  if (hidden_layers >= 2) {
    for (i in 2:hidden_layers) {
      model %>%
        layer_dense(
          units = units,
          activation = activation,
          kernel_regularizer = if (l2 > 0) regularizer_l2(l2) else NULL
        )
      if (dropout > 0) {
        model %>% layer_dropout(rate = dropout)
```

```
      }
    }
  }

  # Output layer for binary classification
  model %>% layer_dense(units = 1, activation = "sigmoid")

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

  model
}

run_experiment <- function(name,
                           hidden_layers,
                           units,
                           activation,
                           loss,
                           dropout = 0,
                           l2 = 0,
                           epochs = 20,
                           batch_size = 512) {

  k_clear_session()

  model <- build_model(
    num_words = num_words,
    hidden_layers = hidden_layers,
    units = units,
    activation = activation,
    loss = loss,
    dropout = dropout,
    l2 = l2
  )

  history <- model %>% fit(
    partial_x_train, partial_y_train,
    epochs = epochs,
    batch_size = batch_size,
    validation_data = list(x_val, y_val),
    verbose = 0
  )

  val_acc <- as.numeric(history$metrics$val_accuracy)
  best_epoch <- which.max(val_acc)

  test_metrics <- model %>% evaluate(x_test, y_test, verbose = 0)

  tibble(
    name = name,
```

```
    hidden_layers = hidden_layers,
    units = units,
    activation = activation,
    loss = loss,
    dropout = dropout,
    l2 = l2,
    best_epoch = best_epoch,
    best_val_acc = max(val_acc),
    test_acc = as.numeric(test_metrics["accuracy"])
  )
}
```

3. Baseline model to build off

```
baseline <- run_experiment(
  name = "Baseline: 2 layers, 16 units, relu, BCE",
  hidden_layers = 2,
  units = 16,
  activation = "relu",
  loss = "binary_crossentropy"
)

baseline
```

```
## # A tibble: 1 x 10
##   name             hidden_layers units activation loss  dropout    l2 best_epoch
##   <chr>                    <dbl> <dbl> <chr>      <chr>   <dbl> <dbl>      <int>
## 1 Baseline: 2 lay~             2    16 relu       bina~       0     0          4
## # i 2 more variables: best_val_acc <dbl>, test_acc <dbl>
```

4. Experiments

For each experiment I changed one main thing at a time so it is easier to compare results. I did use MSE loss function and tanh once.

```
# Layers: 1 layer vs 3 layers
exp_layers_1 <- run_experiment(
  name = "1 layer, 16 units, relu, BCE",
  hidden_layers = 1, units = 16, activation = "relu", loss = "binary_crossentropy"
)

exp_layers_3 <- run_experiment(
  name = "3 layers, 16 units, relu, BCE",
  hidden_layers = 3, units = 16, activation = "relu", loss = "binary_crossentropy"
)

# Units: 32 and 64 (keeping 2 layers, relu, BCE)
exp_units_32 <- run_experiment(
  name = "2 layers, 32 units, relu, BCE",
  hidden_layers = 2, units = 32, activation = "relu", loss = "binary_crossentropy"
```

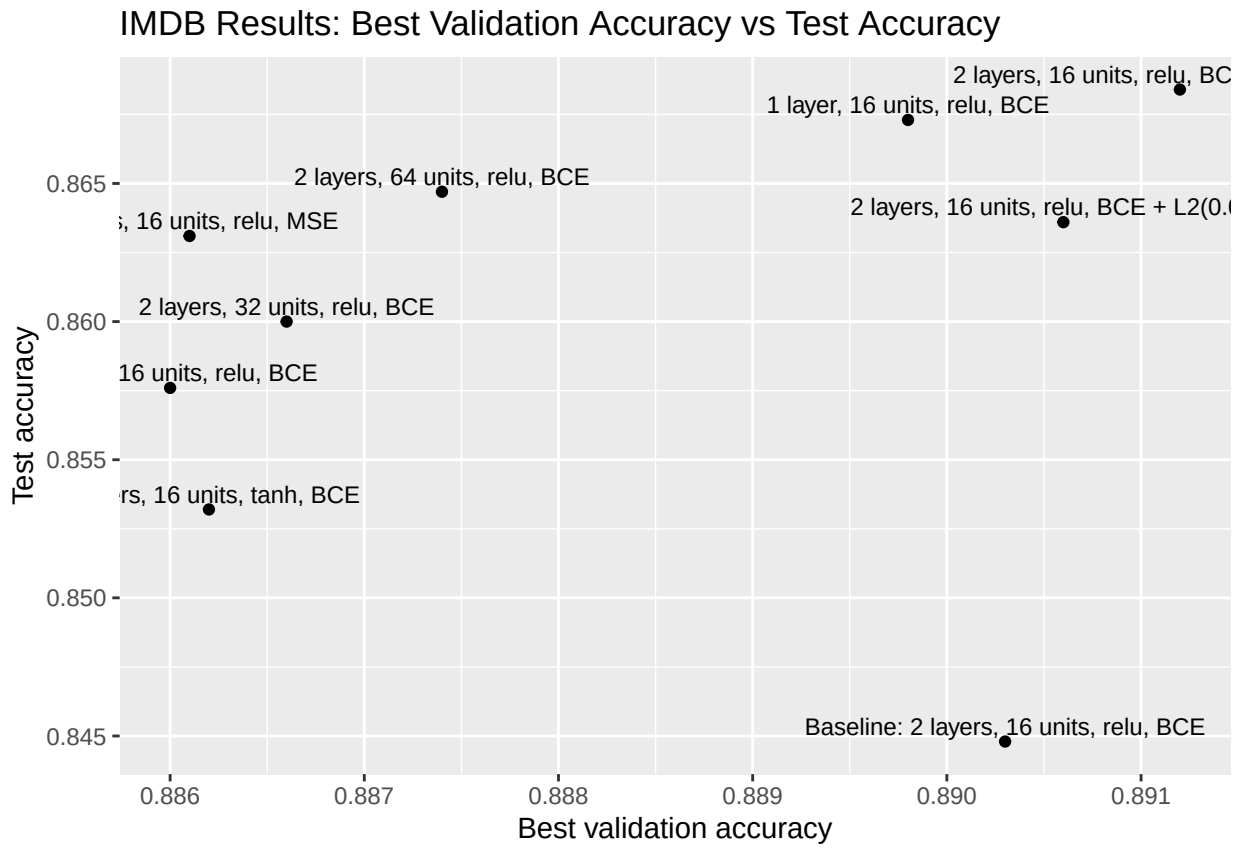
```
)

exp_units_64 <- run_experiment(
  name = "2 layers, 64 units, relu, BCE",
  hidden_layers = 2, units = 64, activation = "relu", loss = "binary_crossentropy"
)

# Activation: tanh
exp_tanh <- run_experiment(
  name = "2 layers, 16 units, tanh, BCE",
  hidden_layers = 2, units = 16, activation = "tanh", loss = "binary_crossentropy"
)

# Loss: MSE
exp_mse <- run_experiment(
  name = "2 layers, 16 units, relu, MSE",
  hidden_layers = 2, units = 16, activation = "relu", loss = "mse"
)

exp_dropout <- run_experiment(
  name = "2 layers, 16 units, relu, BCE + dropout(0.3)",
  hidden_layers = 2, units = 16, activation = "relu",
  loss = "binary_crossentropy", dropout = 0.3
)

exp_l2 <- run_experiment(
  name = "2 layers, 16 units, relu, BCE + L2(0.001)",
  hidden_layers = 2, units = 16, activation = "relu",
  loss = "binary_crossentropy", l2 = 0.001
)

results <- bind_rows(
  baseline,
  exp_layers_1, exp_layers_3,
  exp_units_32, exp_units_64,
  exp_tanh, exp_mse,
  exp_dropout, exp_l2
) %>%
  mutate(
    best_val_acc = round(best_val_acc, 4),
    test_acc = round(test_acc, 4)
  ) %>%
  arrange(desc(best_val_acc))

results
```

```
## # A tibble: 9 x 10
##   name             hidden_layers units activation loss  dropout    l2 best_epoch
##   <chr>                    <dbl> <dbl> <chr>      <chr>   <dbl> <dbl>      <int>
## 1 2 layers, 16 un~             2    16 relu       bina~     0.3 0              4
## 2 2 layers, 16 un~             2    16 relu       bina~     0   0.001          3
## 3 Baseline: 2 lay~             2    16 relu       bina~     0   0              4
## 4 1 layer, 16 uni~             1    16 relu       bina~     0   0              4
```

```
## 5 2 layers, 64 un~             2    64 relu       bina~     0   0             4
## 6 2 layers, 32 un~             2    32 relu       bina~     0   0             3
## 7 2 layers, 16 un~             2    16 tanh       bina~     0   0             4
## 8 2 layers, 16 un~             2    16 relu       mse       0   0             5
## 9 3 layers, 16 un~             3    16 relu       bina~     0   0             5
## # i 2 more variables: best_val_acc <dbl>, test_acc <dbl>
```

5. Plot of results (validation vs. test)

```
ggplot(results, aes(x = best_val_acc, y = test_acc)) +
  geom_point() +
  geom_text(aes(label = name), vjust = -0.4, size = 3) +
  labs(
    title = "IMDB Results: Best Validation Accuracy vs Test Accuracy",
    x = "Best validation accuracy",
    y = "Test accuracy"
  )
```



6. Conclusions

Based on my runs, the baseline model performed well without needing a very complex structure. -Adding more layers did not automatically improve test accuracy. -More units seemed to increase validation accuracy, but it did not always increase test accuracy, which suggests overfitting can still happen. -MSE worked, but

binary crossentropy is a better match for sigmoid-based binary classification and usually performed as well or better, though I did only use MSE once since it did not seem to improve anything. -ReLU generally performed better than tanh in this setup, though I did only use it once, again, because it didn't really improve anything. -I tried dropout and L2 to improve generalization. In my results they sometimes helped a little, but the best settings depend on the model and training. -The best validation accuracy I received was .8903 with 2 layers, 64 units, ReLU, Binary-Crossentropy. While the MSE was exactly the same in validation, it's testing performed slightly worse.