AMLAssignment 1

Deepak Machkuri

Preparing data

```
library(keras)
imdb <- dataset_imdb(num_words = 10000)
c(c(train_data, train_labels), c(test_data, test_labels)) %<-% imdb
```

Data vectorization

```
vectorize_sequences <- function(sequences, dimension = 10000) {
results <- matrix(0, nrow = length(sequences), ncol = dimension)
for (i in 1:length(sequences))
results[i, sequences[[i]]] <- 1
results}
x_train <- vectorize_sequences(train_data)
x_test <- vectorize_sequences(test_data)
y_train <- as.numeric(train_labels)
y_test <- as.numeric(test_labels)
```

Building network - 2 layers, units = 16, loss function: “binary_crossentropy”, activation = “relu”

```
model <- 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")
model %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy"))
```

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]
model %>% fit(x_train, y_train, epochs = 4, batch_size = 512)
results <- model %>% evaluate(x_test, y_test)
results
## $loss
## [1] 0.2840919
##
## $accuracy
## [1] 0.88716
```

Using 3 hidden layers

Building network - 3 layers, units = 16, loss function: “binary_crossentropy”, activation = “relu”

```
model1 <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = "relu", input_shape = c(10000)) %>%
  layer_dense(units = 16, activation = "relu") %>% layer_dense(units = 16, activati
on = "relu") %>% layer_dense(units = 1, activation = "sigmoid")
model1 %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy"))
```

Validation

```
val_indices1 <- 1:10000
x_val1 <- x_train[val_indices,]
partial_x_train1 <- x_train[-val_indices,]
y_val1 <- y_train[val_indices]
partial_y_train1 <- y_train[-val_indices]
model1 %>% fit(x_train, y_train, epochs = 4, batch_size = 512)
results1 <- model1 %>% evaluate(x_test, y_test)
results1
## $loss
## [1] 0.3016961
##
## $accuracy
```

```
## [1] 0.88284
```

Using layers with more hidden units

Building network - 2 layers, units = 32, loss function: “binary_crossentropy”, activation = “relu”

```
model3 <- keras_model_sequential() %>%
  layer_dense(units = 32, activation = "relu", input_shape = c(10000)) %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")
model3 %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy"))
```

Validation

```
val_indices <- 1:10000
x_val3 <- x_train[val_indices,]
partial_x_train3 <- x_train[-val_indices,]
y_val3 <- y_train[val_indices]
partial_y_train3 <- y_train[-val_indices]
model3 %>% fit(x_train, y_train, epochs = 4, batch_size = 512)
results3 <- model3 %>% evaluate(x_test, y_test)
results3
## $loss
## [1] 0.3278894
##
## $accuracy
## [1] 0.875
```

Building network - 2 layers, units = 16, loss function: “mean_squared_error”, activation = “relu”

```
model5 <- 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")
```

```
model5 %>% compile(
  optimizer = "rmsprop",
  loss = "mean_squared_error",
  metrics = c("accuracy"))
```

Validation

```
val_indices <- 1:10000
x_val5 <- x_train[val_indices,]
partial_x_train5 <- x_train[-val_indices,]
y_val5 <- y_train[val_indices]
partial_y_train5 <- y_train[-val_indices]
model5 %>% fit(x_train, y_train, epochs = 4, batch_size = 512)
results5 <- model5 %>% evaluate(x_test, y_test)
results5
## $loss
## [1] 0.08579193
##
## $accuracy
## [1] 0.8826
```

Building network - 2 layers, units = 16, loss function: “binary_crossentropy”, activation = “tanh”

```
model6 <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = "tanh", input_shape = c(10000)) %>%
  layer_dense(units = 16, activation = "tanh") %>%
  layer_dense(units = 1, activation = "sigmoid")
model6 %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy"))
```

Validation

```
val_indices <- 1:10000
x_val6 <- x_train[val_indices,]
partial_x_train6 <- x_train[-val_indices,]
y_val6 <- y_train[val_indices]
```

```
partial_y_train6 <- y_train[-val_indices]

model6 %>% fit(x_train, y_train, epochs = 4, batch_size = 512)

results6 <- model6 %>% evaluate(x_test, y_test)

results6

## $loss

## [1] 0.3214437

##

## $accuracy

## [1] 0.87756
```

After adjusting different parameters and methods and comparing, the model of 2 layers, units = 16, loss function: “binary_crossentropy”, activation = “relu” has the highest accuracy.

Units of layer: Accuracy reduced from 16 to 32 units, and accuracy increased from 32 to 64 units.

Loss function = “mean_squared_error”: It reduces the loss, but it also reduces the accuracy.

Activation = “tanh”: It reduces both loss and accuracy.

Building network with Regularization - 2 layers, units = 16, loss function: “binary_crossentropy”, activation = “relu”

```
set.seed(123)

model7 <- keras_model_sequential() %>%

  layer_dense(units = 16, kernel_regularizer = regularizer_l2(0.001), activation =
"relu", input_shape = c(10000)) %>%

  layer_dense(units = 16, kernel_regularizer = regularizer_l2(0.001), activation =
"relu") %>%

  layer_dense(units = 1, activation = "sigmoid")

model7 %>% compile(

  optimizer = "rmsprop",

  loss = "binary_crossentropy",

  metrics = c("accuracy"))
```

Validation

```
val_indices <- 1:10000

x_val7 <- x_train[val_indices,]

partial_x_train7 <- x_train[-val_indices,]
```

```
y_val7 <- y_train[val_indices]
partial_y_train7 <- y_train[-val_indices]
model7 %>% fit(x_train, y_train, epochs = 4, batch_size = 512)
results7 <- model7 %>% evaluate(x_test, y_test)
results7
## $loss
## [1] 0.3369988
##
## $accuracy
## [1] 0.88536
```