MACS 30200 HW1

Ling Dai

```r
#import libraries
library(keras)
library(tensorflow)
use_python("/Users/lingdai/anaconda3/bin/python3")

#import data
fmnist <- dataset_fashion_mnist()

#set seed
set.seed(1234)
```

```r
train_images <- fmnist$train$x
train_labels <- fmnist$train$y
test_images <- fmnist$test$x
test_labels <- fmnist$test$y

train_labels <- to_categorical(train_labels)
test_labels <- to_categorical(test_labels)

#Preprocess the data by converting the data to a 2D tensor with individual values between 0 and 1
img_rows <- img_cols <- 28
train_images <- array_reshape(train_images, c(60000, 28*28))
train_images <- train_images / 255
str(train_images)
```

```
##  num [1:60000, 1:784] 0 0 0 0 0 0 0 0 0 0 ...
```

```r
test_images <- array_reshape(test_images, c(10000, 28*28))
test_images <- test_images / 255
str(test_images)
```

```
##  num [1:10000, 1:784] 0 0 0 0 0 0 0 0 0 0 ...
```

```r
#Randomly split the training data into 50,000 training observations and 10,000 validation observations
training_ind <- sample(60000, size = 50000)
training_images <- train_images[training_ind, ]
training_labels <- train_labels[training_ind, ]
validation_images <- train_images[-training_ind, ]
validation_labels <- train_labels[-training_ind, ]
```

Final Model (the lowest validation loss score)

Among all the models tested above, alternative model 8 achieved the best performance, with a lowest validation loss of 0.26 at epoch 33. Therefore, it is selected as the final model for the subsequent test.

When reestimating the model using all of the training data with the same hyperparameter, the calculated test set loss is 0.37, and the test set accuracy is 0.88. While the test set loss is quite higher than the validation loss during the first found of model testing, 0.37 is still an acceptable value. Moreover, the 0.87 test set accuracy is also decent. Overall, the model generalizes decently when applied onto the test set.

Compared to other models tested, this model configuration has several characteristics: (1) it has only 2 layers (including the output layer); (2) the first layer has 1024, which is greater than the 512 units used in the initial model; (3) the model doesn't use any drop-out or weight-regularized configurations. The fact that the model has only 2 layers but still performs relatively well compared to other models with more layers may indicate that one hidden layer is sufficient in solving the problem in this case. Moreover, the relatively large of the first layer may have allowed the model to perform better. Last but not least, while drop-out and regularized configurations may prevent models from overfitting when the number of epochs is large, they do not necessarily always improve upon the minimum loss or the maximum accuracy a model can achieve. Because of that, although this model doesn't have a drop-out or regularized configuration, it may still attain better performance than drop-out and regularized models when the correct number of epochs is chosen.

```
network <- keras_model_sequential() %>%
  layer_dense(units = 1024, activation = "relu", input_shape = c(28 * 28)) %>%
  layer_dense(units = 10, activation = "softmax")

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

network %>% fit(train_images, train_labels,
                epochs = 30, batch_size = 1024)

metrics <- network %>% evaluate(test_images, test_labels, verbose = 0)
metrics
```

```
## $loss
## [1] 0.3212416
##
## $acc
## [1] 0.8915
```