MACS 30200 HW1 (pt2)

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, ]
```

Weight-Regularized Models

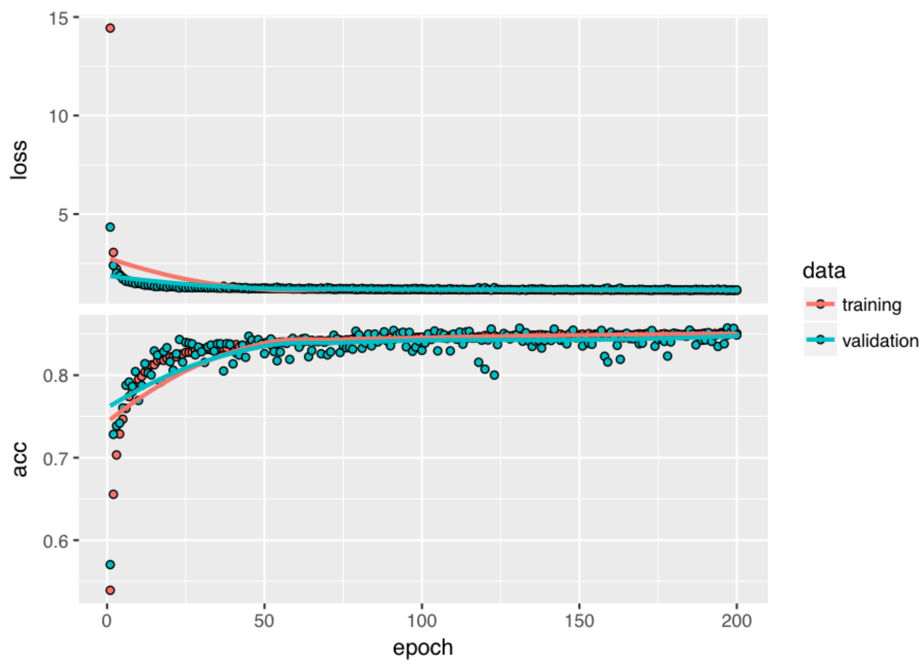
The l1 weight-regularized model attained a minimum validation loss at epoch 171. The l2 weight-regularized model attained a minimum validation loss at epoch 105. Also, according to the graphical comparison below, l2 weight-regularized model performed better than l1 weight-regularized model in this case.

```r
network <- keras_model_sequential() %>%
  layer_dense(units = 512, activation = "relu", input_shape = c(28 * 28),
              kernel_regularizer = regularizer_l1(l=0.001)) %>%
```

```r
  layer_dense(units = 512, activation = "relu", input_shape = c(28 * 28),
              kernel_regularizer = regularizer_l1(l=0.001)) %>%
  layer_dense(units = 512, activation = "relu", input_shape = c(28 * 28),
              kernel_regularizer = regularizer_l1(l=0.001)) %>%
  layer_dense(units = 512, activation = "relu", input_shape = c(28 * 28),
              kernel_regularizer = regularizer_l1(l=0.001)) %>%
  layer_dense(units = 10, activation = "softmax")

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

history3 <- network %>% fit(training_images, training_labels,
                epochs = 200, batch_size = 512,
                validation_data = list(validation_images, validation_labels))
plot(history3)
```




```r
min(history3$metrics$val_loss)
```

```
## [1] 1.126875
```

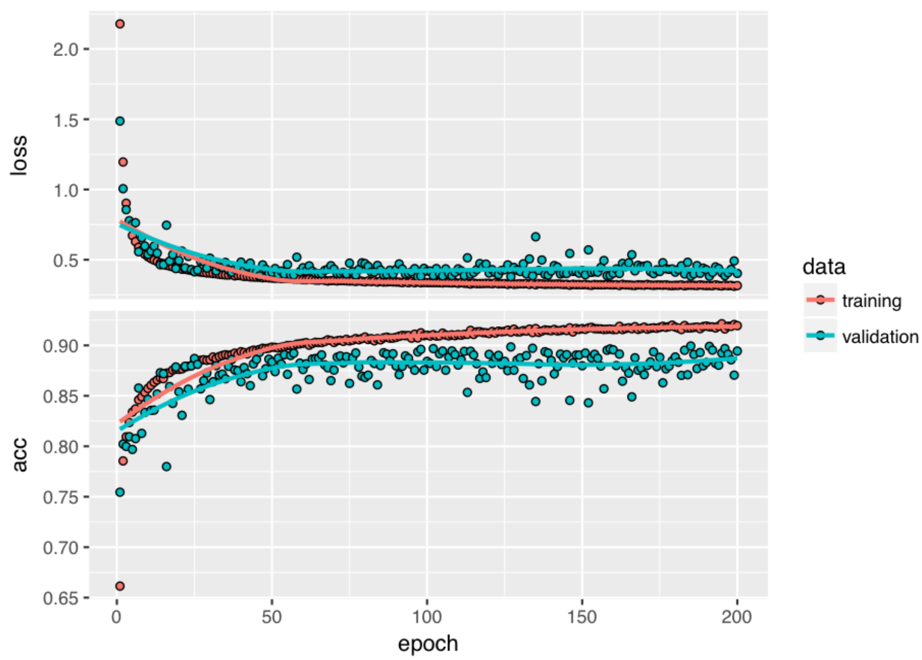
```r
which(history3$metrics$val_loss==min(history3$metrics$val_loss))
```

```
## [1] 197
```

```r
network <- keras_model_sequential() %>%
  layer_dense(units = 512, activation = "relu", input_shape = c(28 * 28),
              kernel_regularizer = regularizer_l2(l=0.001)) %>%
  layer_dense(units = 512, activation = "relu", input_shape = c(28 * 28),
              kernel_regularizer = regularizer_l2(l=0.001)) %>%
  layer_dense(units = 512, activation = "relu", input_shape = c(28 * 28),
              kernel_regularizer = regularizer_l2(l=0.001)) %>%
  layer_dense(units = 512, activation = "relu", input_shape = c(28 * 28),
              kernel_regularizer = regularizer_l2(l=0.001)) %>%
  layer_dense(units = 10, activation = "softmax")

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

history4 <- network %>% fit(training_images, training_labels,
                epochs = 200, batch_size = 512,
                validation_data = list(validation_images, validation_labels))
plot(history4)
```




```r
min(history4$metrics$val_loss)
```

```
## [1] 0.3794742
```

```r
which(history4$metrics$val_loss==min(history4$metrics$val_loss))
```

```
## [1] 97
```

```r
plot(seq(1,200), history3$metrics$val_loss, type='l', col="blue", ylim=c(0,5))
lines(seq(1,200), history4$metrics$val_loss, col="red", ylim=c(0,5))
```