Homework 1

Adam Shelton

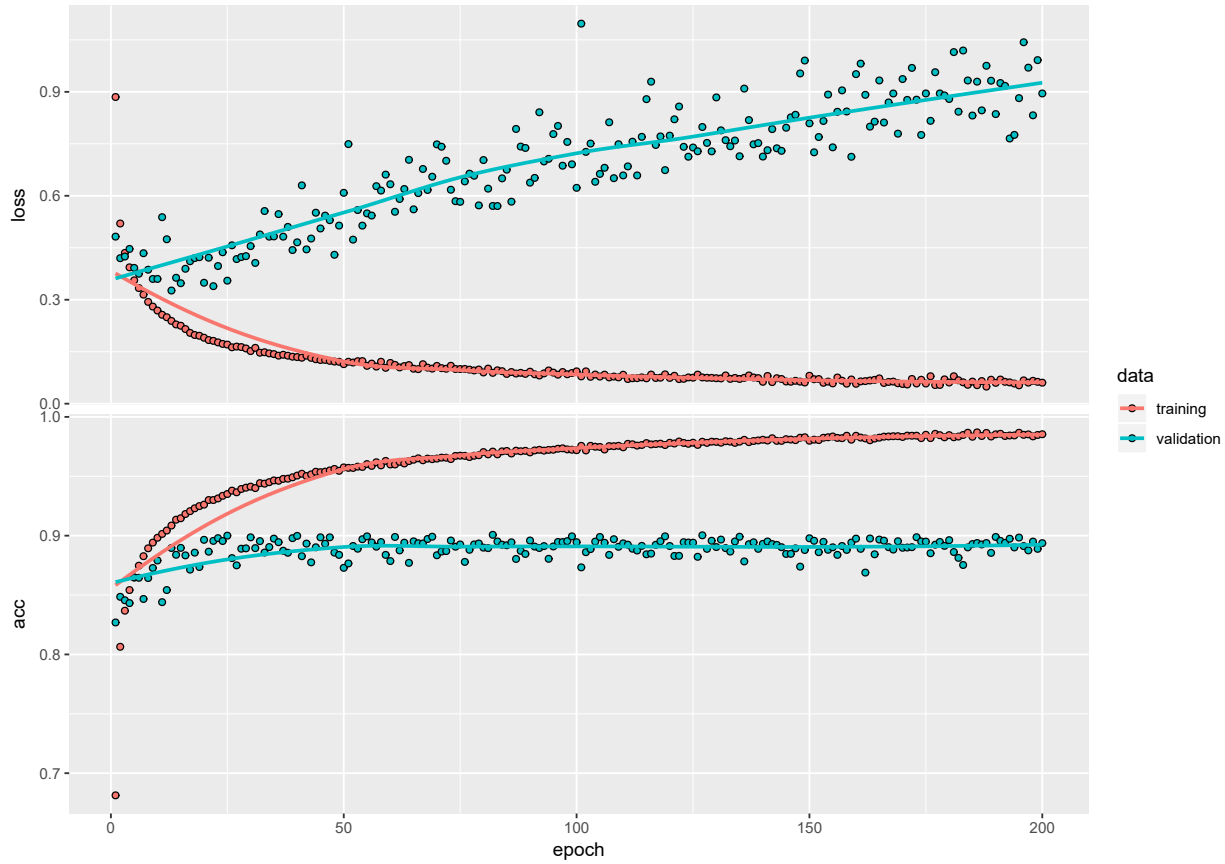
5/8/2019

Getting Data

```
fashion_mnist = keras::dataset_fashion_mnist()

x_train = fashion_mnist$train$x
y_train = fashion_mnist$train$y

x_test = fashion_mnist$test$x
y_test = fashion_mnist$test$y

rm(fashion_mnist)

x_train = array_reshape(x_train, c(nrow(x_train), 28 * 28))/255
x_test = array_reshape(x_test, c(nrow(x_test), 28 * 28))/255

y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

Initial test

```
model = keras_model_sequential()

model %>% layer_dense(units = 512, activation = "relu", input_shape = c(784)) %>%
    layer_dense(units = 512, activation = "relu") %>% layer_dense(units = 512,
    activation = "relu") %>% layer_dense(units = 512, activation = "relu") %>%
    layer_dense(units = 10, activation = "softmax")

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

history_init = model %>% fit(x_train, y_train, epochs = 200,
    batch_size = 512, validation_split = 0.166666666666667)

plot(history_init)
```




```
model %>% evaluate(x_test, y_test)
```

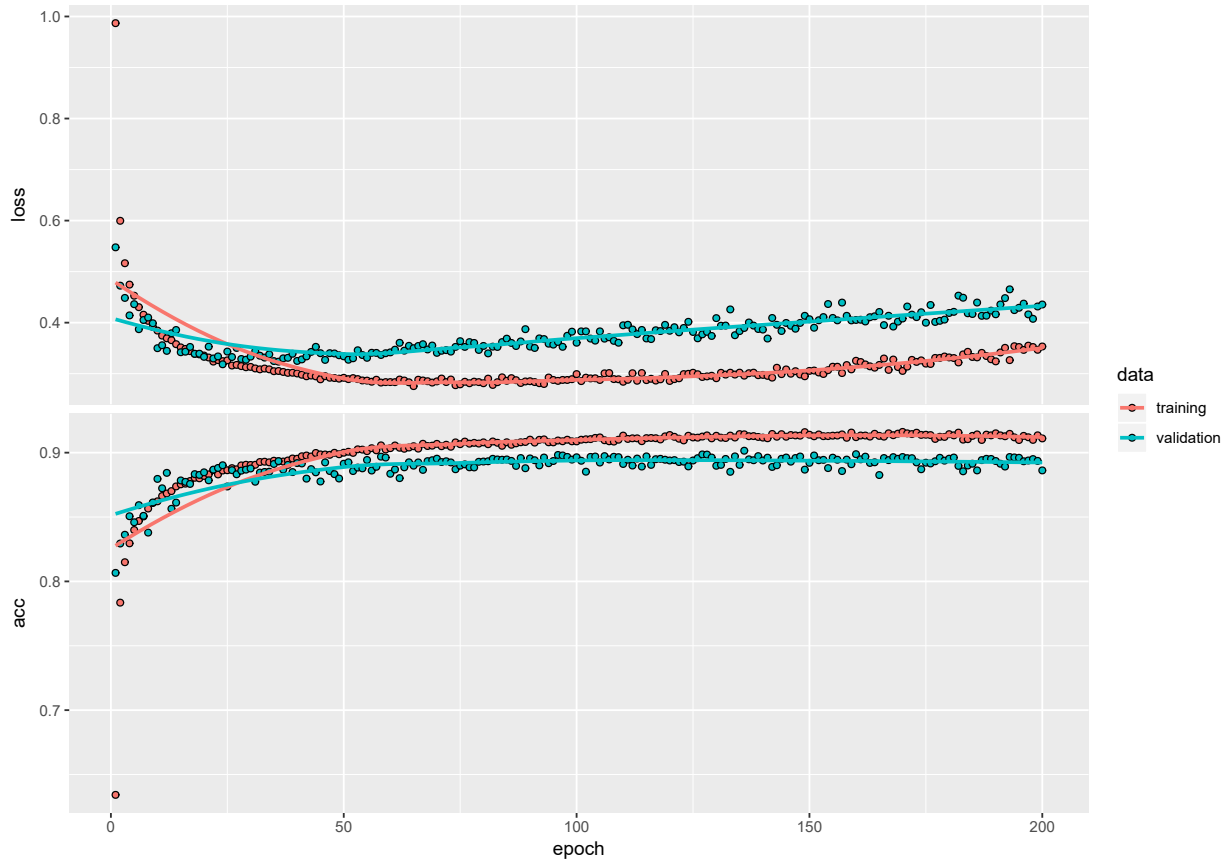
```
## $loss
## [1] 0.9614517
##
## $acc
## [1] 0.8927
```

The validation performance does not improve any further after 50 epochs.

Implementing Dropout

```
model = keras_model_sequential()

model %>% layer_dense(units = 512, activation = "relu", input_shape = c(784)) %>%
    layer_dropout(rate = 0.5) %>% layer_dense(units = 512, activation = "relu") %>%
    layer_dropout(rate = 0.5) %>% layer_dense(units = 512, activation = "relu") %>%
    layer_dropout(rate = 0.5) %>% layer_dense(units = 512, activation = "relu") %>%
    layer_dropout(rate = 0.5) %>% layer_dense(units = 10, activation = "softmax")

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

history_do = model %>% fit(x_train, y_train, epochs = 200, batch_size = 512,
    validation_split = 0.166666666666667)
```

```
plot(history_do)
```




```
model %>% evaluate(x_test, y_test)
```

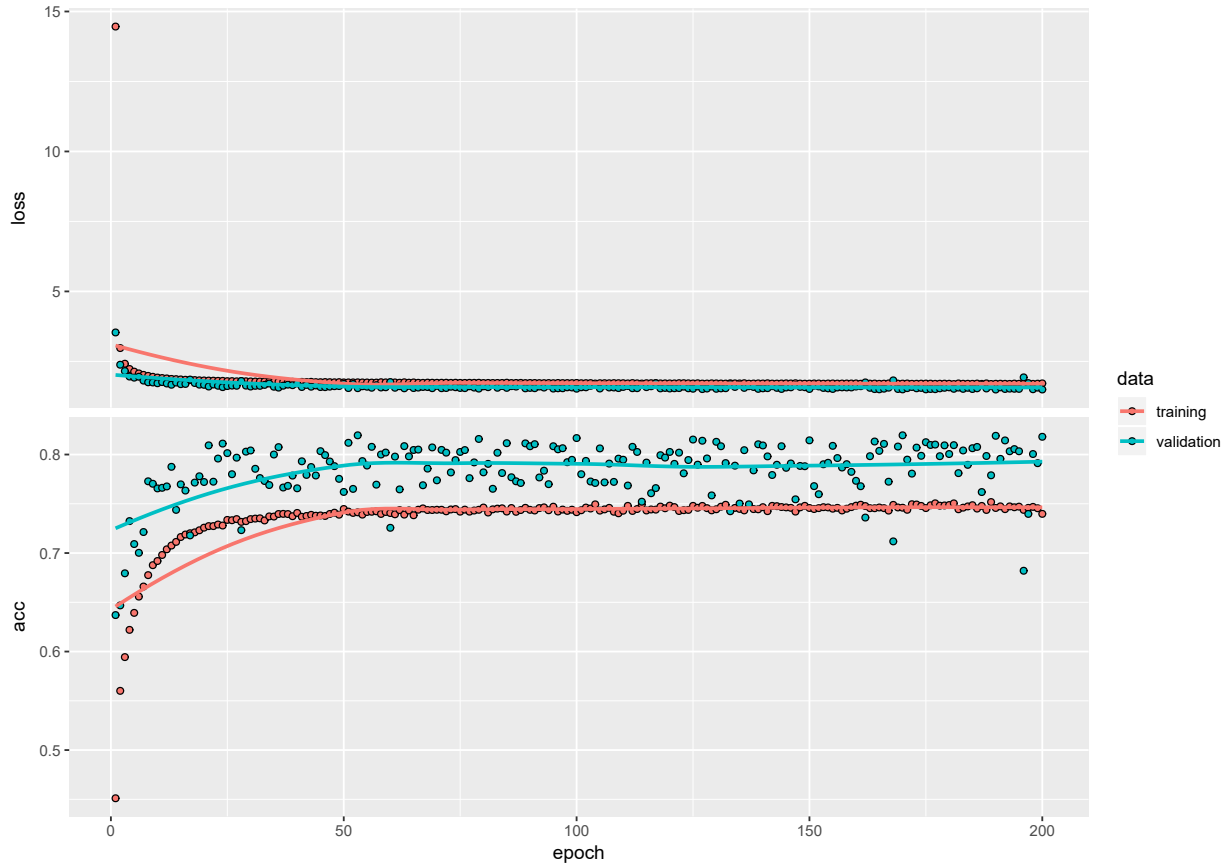
```
## $loss
## [1] 0.4862465
##
## $acc
## [1] 0.8835
```

The validation accuracy, while still leveling off around epoch 50, is considerably higher, with validation losses being less as well.

Weight Regularization

```
model = keras_model_sequential()

model %>% layer_dense(units = 512, activation = "relu", input_shape = c(784),
    kernel_regularizer = regularizer_l1(0.001)) %>% layer_dropout(rate = 0.5) %>%
    layer_dense(units = 512, activation = "relu", kernel_regularizer = regularizer_l1(0.001)) %>%
    layer_dropout(rate = 0.5) %>% layer_dense(units = 512, activation = "relu",
    kernel_regularizer = regularizer_l1(0.001)) %>% layer_dropout(rate = 0.5) %>%
    layer_dense(units = 512, activation = "relu", kernel_regularizer = regularizer_l1(0.001)) %>%
    layer_dropout(rate = 0.5) %>% layer_dense(units = 10, activation = "softmax")

model %>% compile(loss = "categorical_crossentropy", optimizer = optimizer_rmsprop(),
```

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

history_l1 = model %>% fit(x_train, y_train, epochs = 200, batch_size = 512,
    validation_split = 0.166666666666667)

plot(history_l1)
```



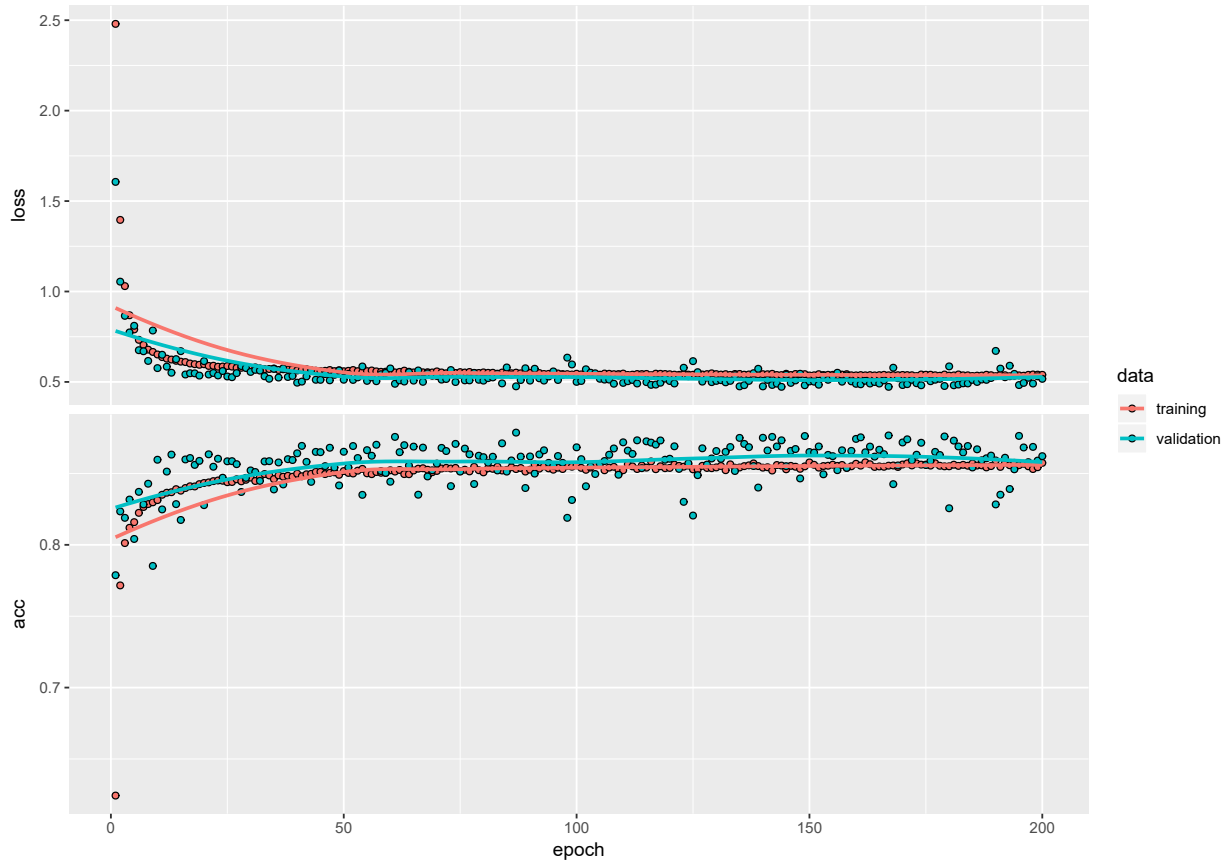

```
model %>% evaluate(x_test, y_test)
```

```
## $loss
## [1] 1.517587
##
## $acc
## [1] 0.8071
```

With L1 regularization at 0.001, training accuracy drops to be lower than validation accuracy. Both accuracies are lower than previous models.

```
model = keras_model_sequential()

model %>% layer_dense(units = 512, activation = "relu", input_shape = c(784),
    kernel_regularizer = regularizer_l2(0.001)) %>% layer_dropout(rate = 0.5) %>%
    layer_dense(units = 512, activation = "relu", kernel_regularizer = regularizer_l2(0.001)) %>%
    layer_dropout(rate = 0.5) %>% layer_dense(units = 512, activation = "relu",
    kernel_regularizer = regularizer_l2(0.001)) %>% layer_dropout(rate = 0.5) %>%
    layer_dense(units = 512, activation = "relu", kernel_regularizer = regularizer_l2(0.001)) %>%
    layer_dropout(rate = 0.5) %>% layer_dense(units = 10, activation = "softmax")
```

```
model %>% compile(loss = "categorical_crossentropy", optimizer = optimizer_rmsprop(),
    metrics = c("accuracy"))

history_l2 = model %>% fit(x_train, y_train, epochs = 200, batch_size = 512,
    validation_split = 0.166666666666667)

plot(history_l2)
```




```
model %>% evaluate(x_test, y_test)
```

```
## $loss
## [1] 0.5407895
##
## $acc
## [1] 0.8559
```

The L2 regularization results in training and validation accuracy which are much more similar than with L1 regularization. The accuracy is also higher at about 0.9.

Other Options

```
tune_grid = expand.grid(num_units = c(256, 512), bat_size = c(512),
    epochs = c(50, 200), do_ratio = c(0.25, 0.5), weight_pen = c(0.001),
    weight_method = c(1, 2)) %>% as_tibble()

hyper_param_tune = function(index, t_grid, model) {
```

```
    params = t_grid[index, ]

    model = keras_model_sequential()

    if (params$weight_method == 2) {
        model %>% layer_dense(units = params$num_units, activation = "relu",
            input_shape = c(784), kernel_regularizer = regularizer_l2(params$weight_pen)) %>%
            layer_dropout(rate = params$do_ratio) %>% layer_dense(units = params$num_units,
            activation = "relu", kernel_regularizer = regularizer_l2(params$weight_pen)) %>%
            layer_dropout(rate = params$do_ratio) %>% layer_dense(units = params$num_units,
            activation = "relu", kernel_regularizer = regularizer_l2(params$weight_pen)) %>%
            layer_dropout(rate = params$do_ratio) %>% layer_dense(units = params$num_units,
            activation = "relu", kernel_regularizer = regularizer_l2(params$weight_pen)) %>%
            layer_dropout(rate = params$do_ratio) %>% layer_dense(units = 10,
            activation = "softmax")
    } else {
        model %>% layer_dense(units = params$num_units, activation = "relu",
            input_shape = c(784), kernel_regularizer = regularizer_l2(params$weight_pen)) %>%
            layer_dropout(rate = params$do_ratio) %>% layer_dense(units = params$num_units,
            activation = "relu", kernel_regularizer = regularizer_l1(params$weight_pen)) %>%
            layer_dropout(rate = params$do_ratio) %>% layer_dense(units = params$num_units,
            activation = "relu", kernel_regularizer = regularizer_l1(params$weight_pen)) %>%
            layer_dropout(rate = params$do_ratio) %>% layer_dense(units = params$num_units,
            activation = "relu", kernel_regularizer = regularizer_l1(params$weight_pen)) %>%
            layer_dropout(rate = params$do_ratio) %>% layer_dense(units = 10,
            activation = "softmax")
    }


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

    history_l2 = model %>% fit(x_train, y_train, epochs = params$epochs,
        batch_size = params$bat_size, validation_split = 0.166666666666667)

    model %>% evaluate(x_test, y_test) %>% as_tibble() %>% bind_cols(params) %>%
        return()
}

results = sapply(1:length(tune_grid$num_units), hyper_param_tune,
    tune_grid, model)
```

```
results %>% as.matrix() %>% t() %>% as_tibble() %>% kable()
```

loss	acc	num_units	bat_size	epochs	do_ratio	weight_pen	weight_method
0.725365580463409	0.85860002040863	256	512	50	0.25	0.001	1
1.04473942184448	0.849200010299683	512	512	50	0.25	0.001	1
0.698598344993591	0.853600025177002	256	512	200	0.25	0.001	1
0.988663054084778	0.860899984836578	512	512	200	0.25	0.001	1
0.871824821281433	0.831300020217896	256	512	50	0.5	0.001	1
1.16336769018173	0.839399993419647	512	512	50	0.5	0.001	1
0.843302340316772	0.831799983978271	256	512	200	0.5	0.001	1
1.11139794340134	0.83050000667572	512	512	200	0.5	0.001	1

loss	acc	num_units	bat_size	epochs	do_ratio	weight_pen	weight_method
0.594045415210724	0.834699988365173	256	512	50	0.25	0.001	2
0.553289919757843	0.842899978160858	512	512	50	0.25	0.001	2
0.489982923412323	0.867200016975403	256	512	200	0.25	0.001	2
0.488590700721741	0.863499999046326	512	512	200	0.25	0.001	2
0.532943052577972	0.852299988269806	256	512	50	0.5	0.001	2
0.633132949542999	0.819199979305267	512	512	50	0.5	0.001	2
0.52707149374485	0.857900023460388	256	512	200	0.5	0.001	2
0.517564965677261	0.860400021076202	512	512	200	0.5	0.001	2

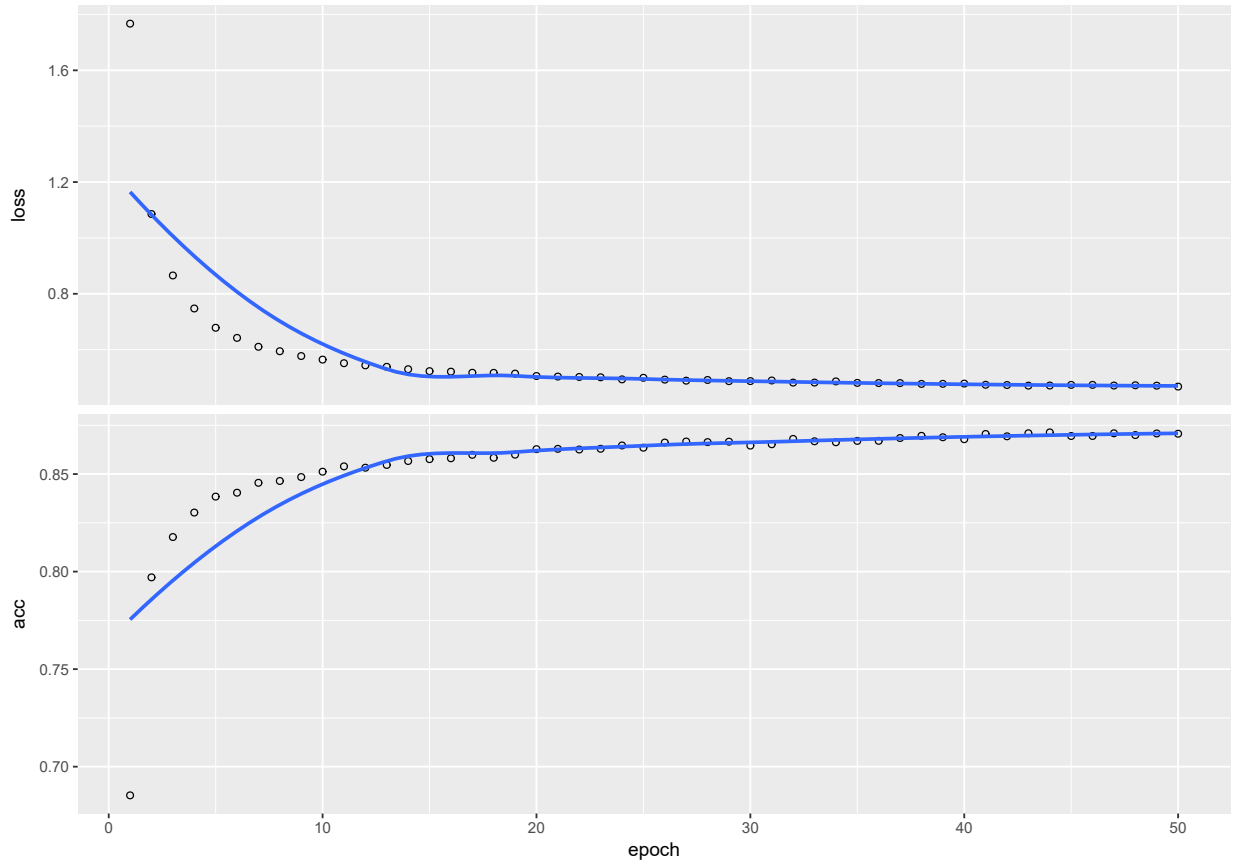
The best model appears to be a model using L2 weight regularization, with 256 units, a batch size of 512, a dropout ratio of 0.25, running at 200 epochs. However, while the gains in accuracy are minimal from a model that runs for 50 epochs, a model running for 200 epochs runs for significantly more time. As the previous models all improved very little past 50 epochs, I will use a model with 50 epochs instead to sacrifice a little accuracy to gain back a significant amount of time.

Final Model

```
final_model = keras_model_sequential()

final_model %>% layer_dense(units = 256, activation = "relu",
    input_shape = c(784), kernel_regularizer = regularizer_l2(0.001)) %>%
    layer_dropout(rate = 0.25) %>% layer_dense(units = 256, activation = "relu",
    kernel_regularizer = regularizer_l2(0.001)) %>% layer_dropout(rate = 0.25) %>%
    layer_dense(units = 512, activation = "relu", kernel_regularizer = regularizer_l2(0.001)) %>%
    layer_dropout(rate = 0.25) %>% layer_dense(units = 256, activation = "relu",
    kernel_regularizer = regularizer_l2(0.001)) %>% layer_dropout(rate = 0.25) %>%
    layer_dense(units = 10, activation = "softmax")
4
```

```
## [1] 4
```

```
final_model %>% compile(loss = "categorical_crossentropy", optimizer = optimizer_rmsprop(),
    metrics = c("accuracy"))
history_final = final_model %>% fit(x_train, y_train, epochs = 50,
    batch_size = 512, validation_split = 0)

plot(history_final)
```




```
yhat_test = predict(final_model, x_test)

correct_vals = yhat_test == y_test

final_accuracy = correct_vals %>% as.numeric() %>% sum()/length(correct_vals)

final_loss = loss_categorical_crossentropy(y_test, yhat_test)
```