Homework 1

Adam Shelton

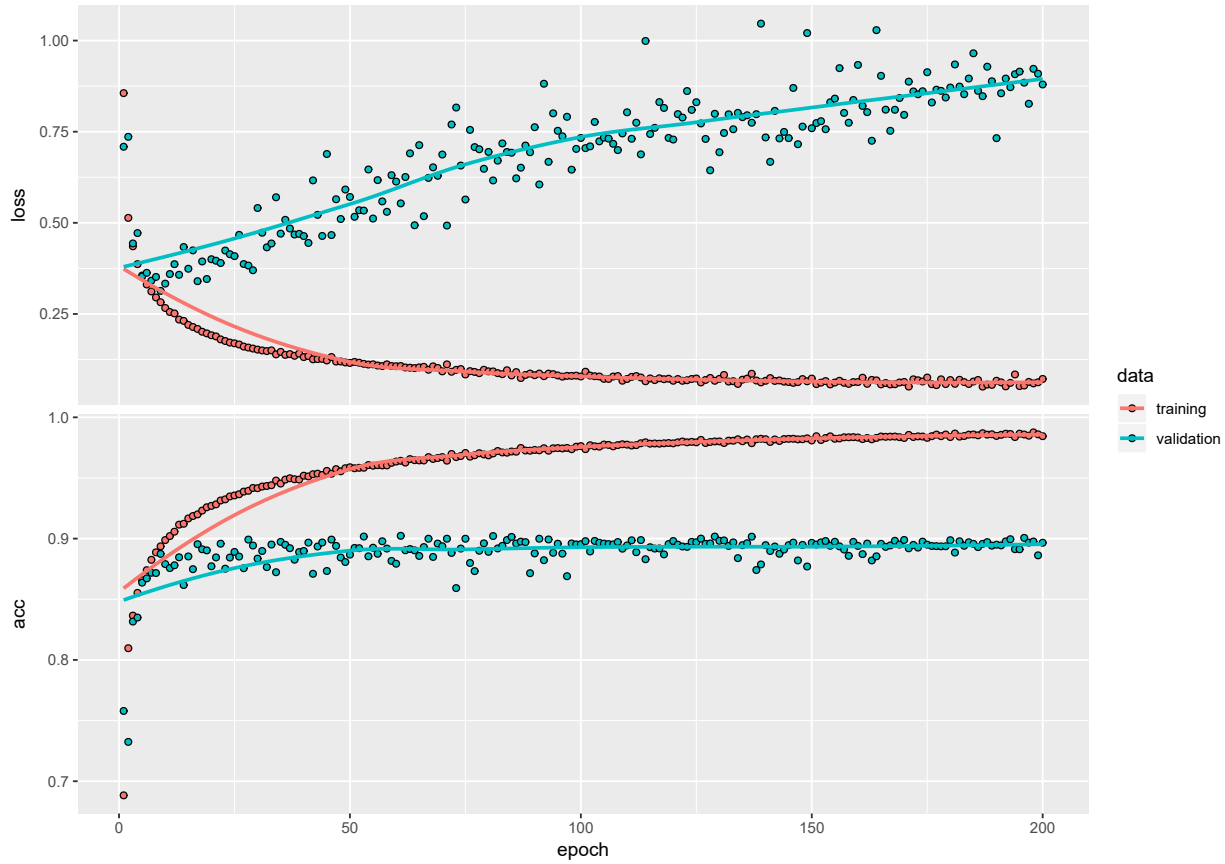
5/8/2019

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)
```

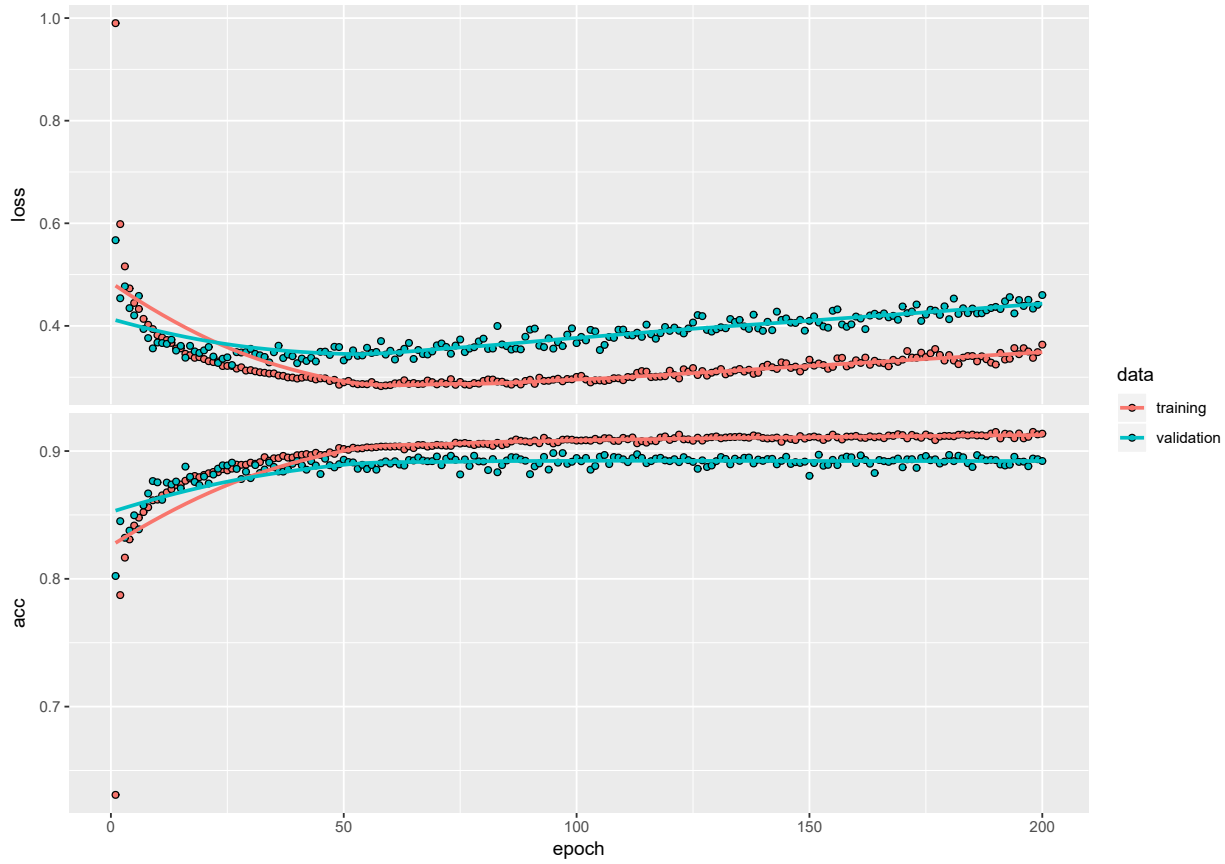
```
## $loss
## [1] 0.9158287
##
## $acc
## [1] 0.8962
```

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)
```

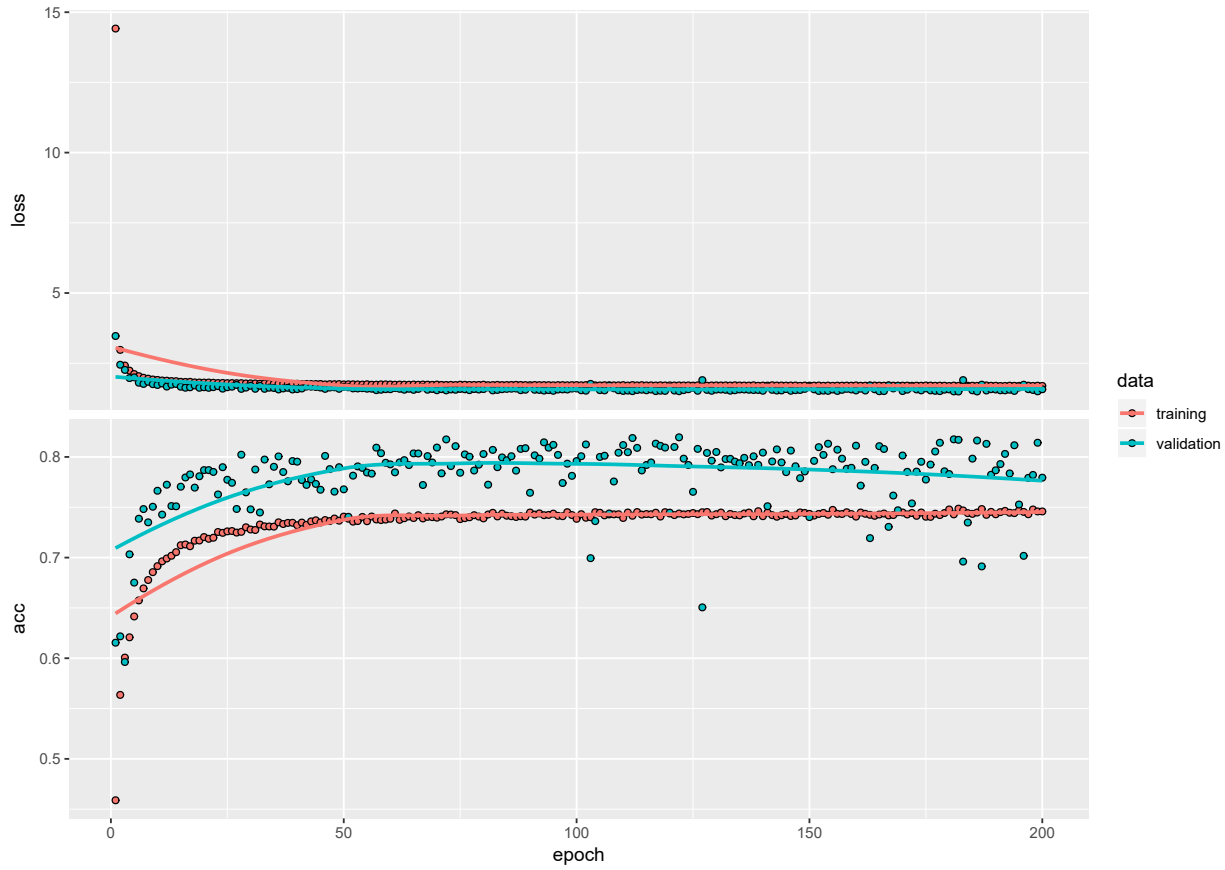
```
## $loss
## [1] 0.5190937
##
## $acc
## [1] 0.8895
```

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)
```



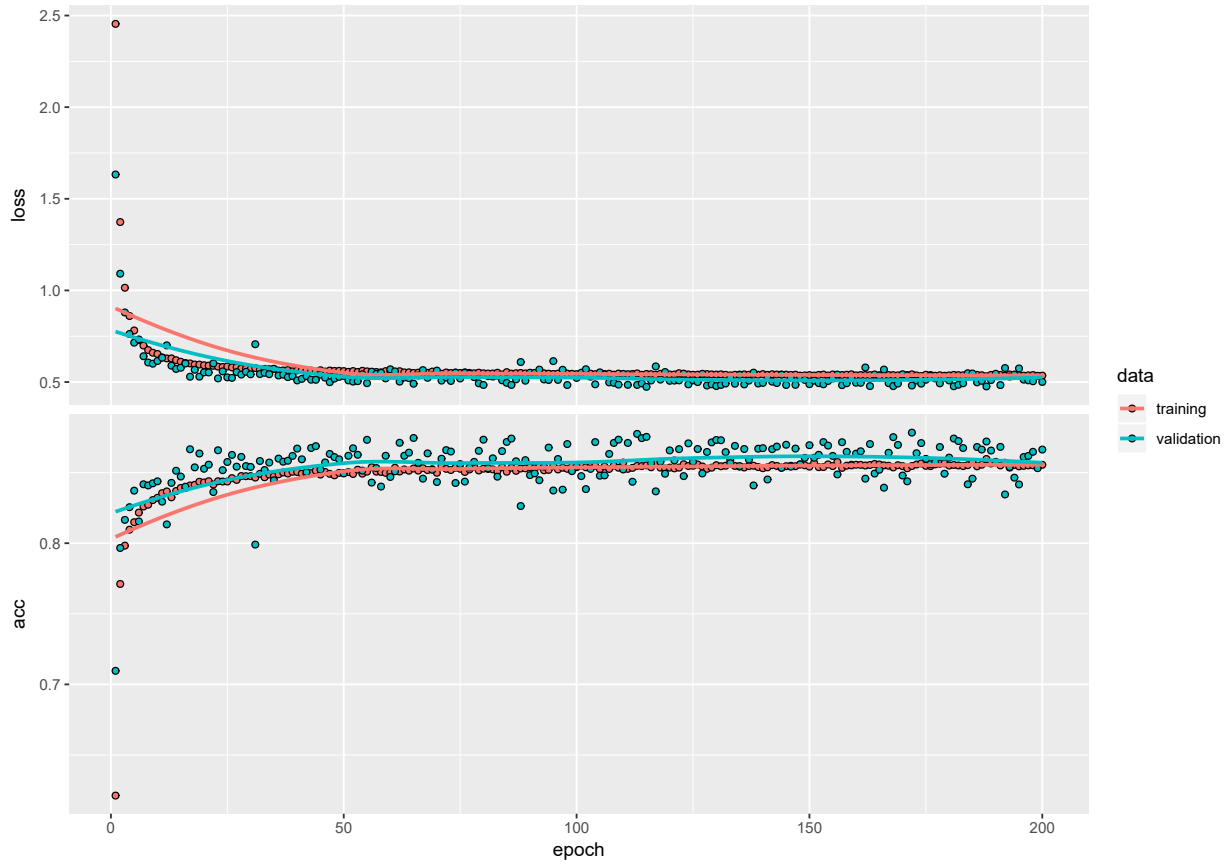

```
model %>% evaluate(x_test, y_test)
```

```
## $loss
## [1] 1.589599
##
## $acc
## [1] 0.7706
```

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] 0.5317465
##
## $acc
## [1] 0.851
```

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()
```

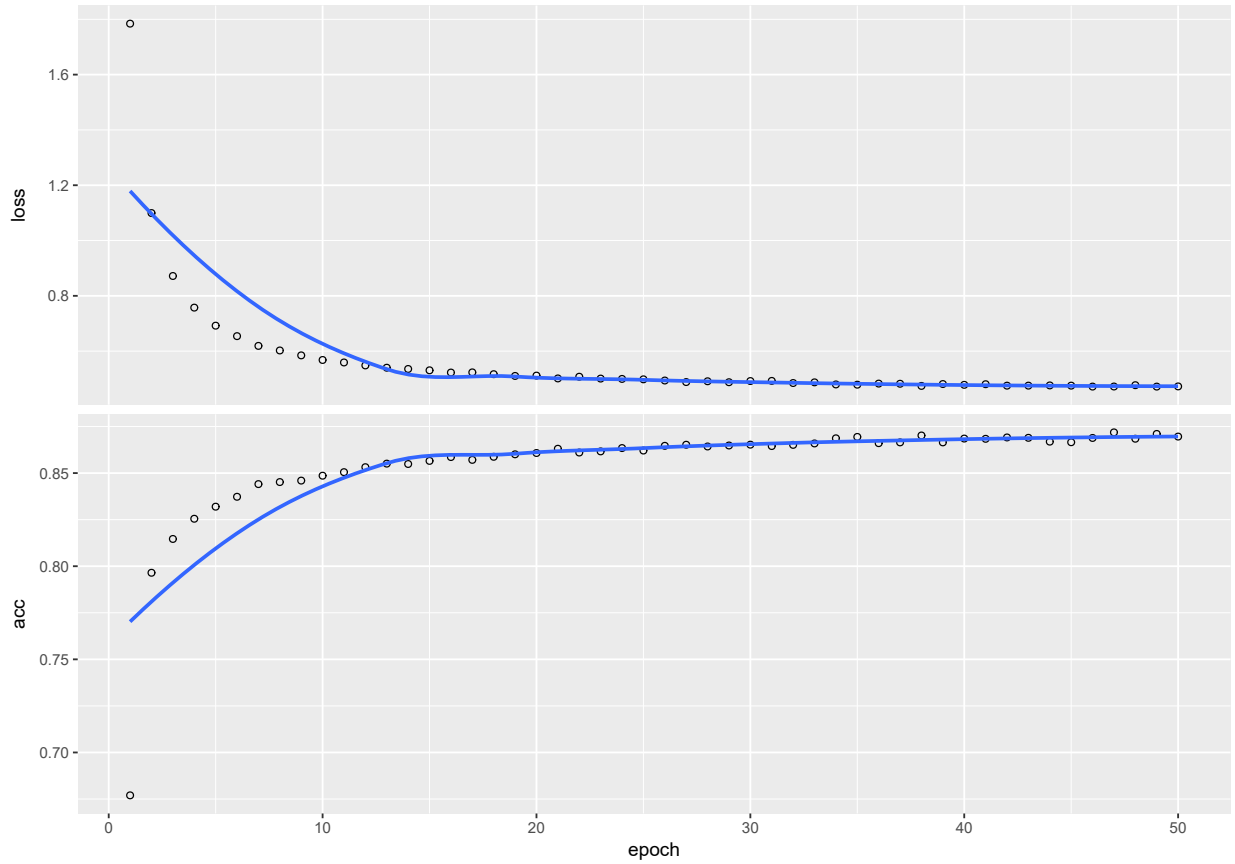
loss	acc	num_units	bat_size	epochs	do_ratio	weight_pen	weight_method
0.724125845861435	0.852699995040894	256	512	50	0.25	0.001	1
1.02142885847092	0.85699999332428	512	512	50	0.25	0.001	1
0.689043481826782	0.86379998922348	256	512	200	0.25	0.001	1
1.00140358867645	0.849200010299683	512	512	200	0.25	0.001	1
0.864067422676086	0.833299994468689	256	512	50	0.5	0.001	1
1.15087582521439	0.83270001411438	512	512	50	0.5	0.001	1
0.876796465110779	0.810599982738495	256	512	200	0.5	0.001	1
1.16995921669006	0.816699981689453	512	512	200	0.5	0.001	1

loss	acc	num_units	bat_size	epochs	do_ratio	weight_pen	weight_method
0.511347409939766	0.851599991321564	256	512	50	0.25	0.001	2
0.50847452955246	0.857699990272522	512	512	50	0.25	0.001	2
0.502453733158112	0.86080002784729	256	512	200	0.25	0.001	2
0.512077448415756	0.85180002450943	512	512	200	0.25	0.001	2
0.508739902925491	0.867500007152557	256	512	50	0.5	0.001	2
0.564224062633514	0.840799987316132	512	512	50	0.5	0.001	2
0.500197556734085	0.866299986839294	256	512	200	0.5	0.001	2
0.529413218832016	0.853900015354156	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")

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_raw = predict(final_model, x_test)
yhat_test_tib = yhat_test_raw %>% as_tibble() %>% round()

reverse_encode = function(ohe_tib) {
    ohe_tib = yhat_test_tib
    rev_func = function(index, ohe_tib, categories) {
        return(categories[ohe_tib[index, ] == 1])
    }
    len = 1:length(as.vector(unlist(ohe_tib[, 1])))
    cats = 0:(length(ohe_tib[1, ]) - 1)
    sapply(len, rev_func, ohe_tib, cats) %>% unlist() %>% return()
}

yhat_test = reverse_encode(yhat_test_tib)

y_test_vec = reverse_encode(y_test)

correct_vals = yhat_test == y_test_vec

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

final_loss = loss_categorical_crossentropy(y_test, predict(final_model,
    x_test))$value_index
```

The accuracy of the final model is 1 and the loss is 0. This indicates that the final model predicts the testing data better than the training data. This is perhaps due to the combination of dropout layers and regularizers

that make the model sufficiently general to fit to other data well.