Assignment 2 (Machine Learning and AI)

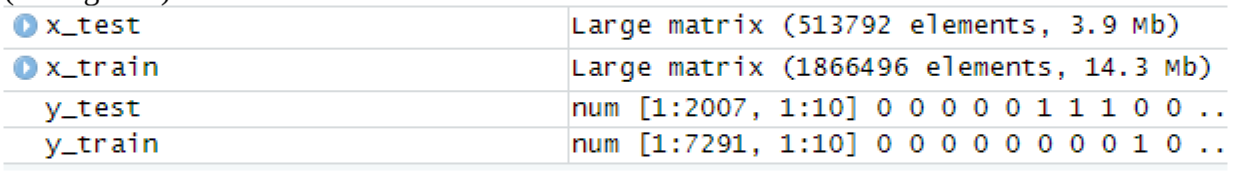
- Shubhang Periwal (19201104)

Abstract

The file data_usps_digits.RData contains data recording handwritten digits from the United States Postal Service (USPS). More in details, the data consists of grayscale (16x16) grid representations of image scans of the digits “0” through “9” (10 digits). We need to use a multilayer neural network with 2 hidden layers to predict the type of digit. We then need to perform analysis of the output. This would be followed by an analysis for 3 hidden layers.

Methodology and Theory

1. Loading data
   - Set a seed so that output remains consistent.
   - Data has 2 main components. i.e. x_train, y_train (training data) and x_test, y_test (testing data).

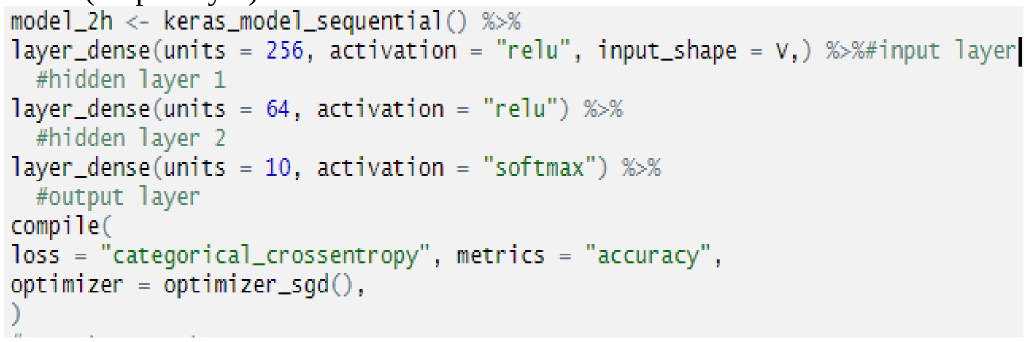


```
x_test     Large matrix (513792 elements, 3.9 Mb)
x_train    Large matrix (1866496 elements, 14.3 Mb)
y_test     num [1:2007, 1:10] 0 0 0 0 0 1 1 1 0 0 ..
y_train    num [1:7291, 1:10] 0 0 0 0 0 0 0 0 1 0 ..
```

   - Helper plot function to visualize this data.

2. Preprocessing
   - Conversion of y to categorical using one-hot encoding.
   - Normalization of data between 0 and 1.
   - Converting x into matrix format from a data frame.

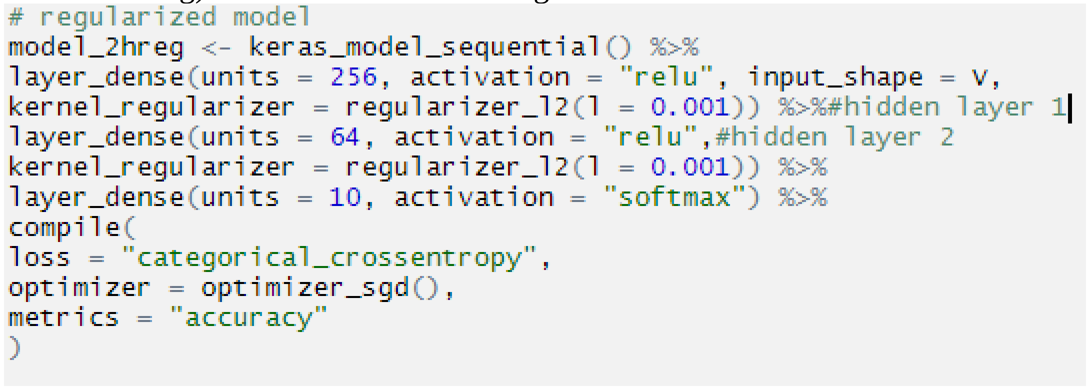
3. Model Design
   - Using keras library with tensorflow backend we can create a model with architecture of input of 256 units as each image is 16x16 (first hidden layer).
   - Second layer with 64 units (second hidden layer) followed by a third layer of 10(output layer).



```
model_2h <- keras_model_sequential() %>%
layer_dense(units = 256, activation = "relu", input_shape = V,) %>%#input layer
  #hidden layer 1
layer_dense(units = 64, activation = "relu") %>%
  #hidden layer 2
layer_dense(units = 10, activation = "softmax") %>%
  #output layer
compile(
loss = "categorical_crossentropy", metrics = "accuracy",
optimizer = optimizer_sgd(),
)
```

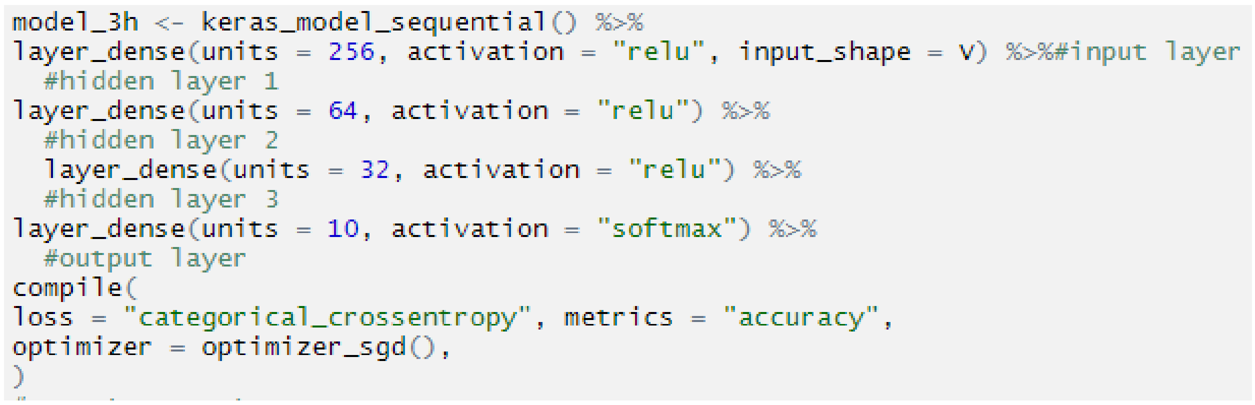
   - We then save this model into a variable for using this to train and validate data. We then need to produce a plot of testing and training data error to visualize how the model is performing.

- We repeat the first 3 steps in case of regularization (adding information to prevent over fitting). We must add a kernel regularizer to do the same.



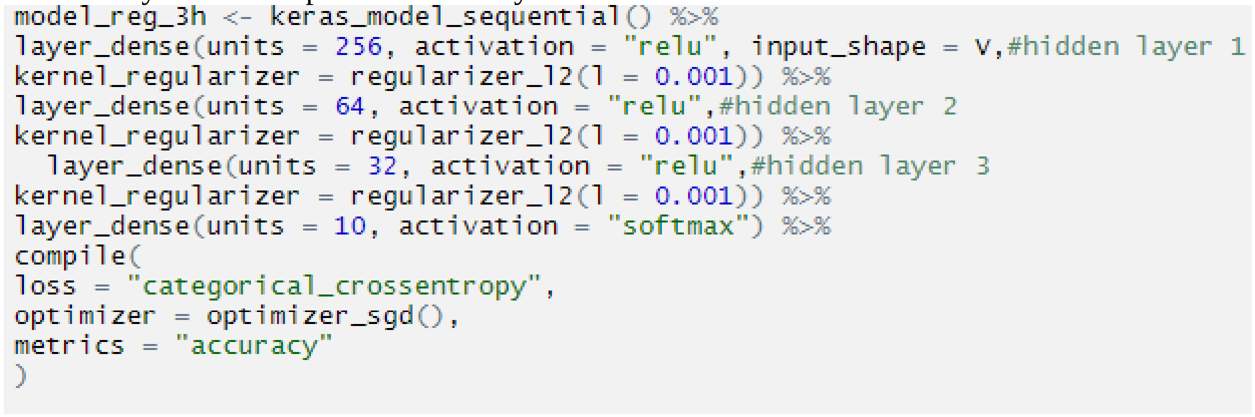
```
# regularized model
model_2hreg <- keras_model_sequential() %>%
layer_dense(units = 256, activation = "relu", input_shape = v,
kernel_regularizer = regularizer_l2(l = 0.001)) %>%#hidden layer 1
layer_dense(units = 64, activation = "relu",#hidden layer 2
kernel_regularizer = regularizer_l2(l = 0.001)) %>%
layer_dense(units = 10, activation = "softmax") %>%
compile(
loss = "categorical_crossentropy",
optimizer = optimizer_sgd(),
metrics = "accuracy"
)
```

- We now compare both the models by plotting errors along with epochs in a graph.
- We again repeat the first 3 steps, but instead we add one more layer in case of the 3 hidden layer problem. We then add another layer of 32 units within the architecture between 64 and 10 units.



```
model_3h <- keras_model_sequential() %>%
layer_dense(units = 256, activation = "relu", input_shape = v) %>%#input layer
  #hidden layer 1
layer_dense(units = 64, activation = "relu") %>%
  #hidden layer 2
  layer_dense(units = 32, activation = "relu") %>%
  #hidden layer 3
layer_dense(units = 10, activation = "softmax") %>%
  #output layer
compile(
loss = "categorical_crossentropy", metrics = "accuracy",
optimizer = optimizer_sgd(),
)
```

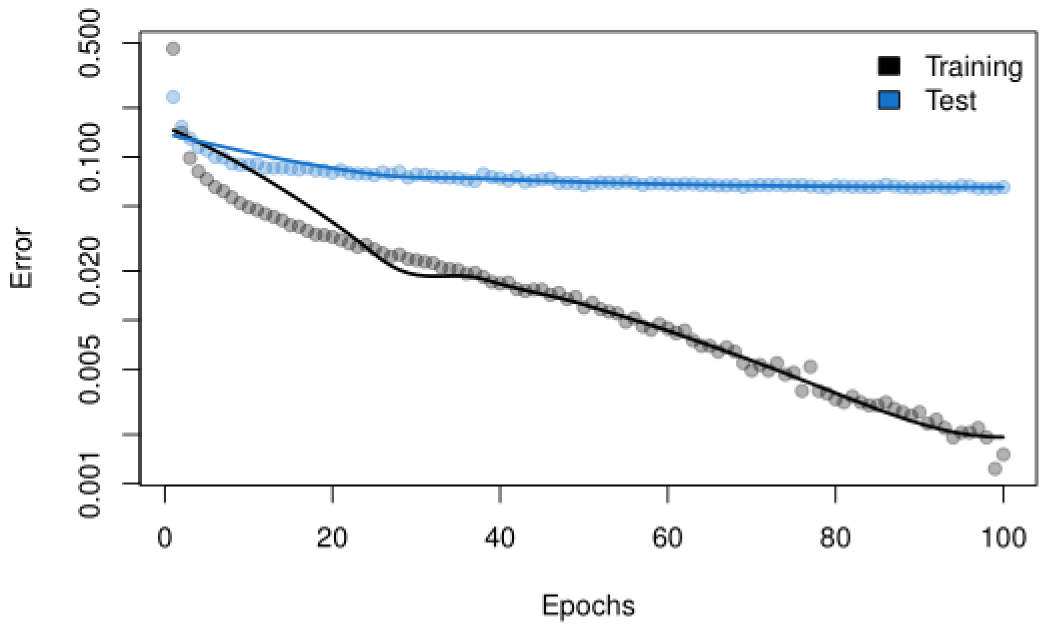
- To solve the hidden layer with regularization, we add a kernel regularizer within each layer. We compare the accuracy of all 4 models.



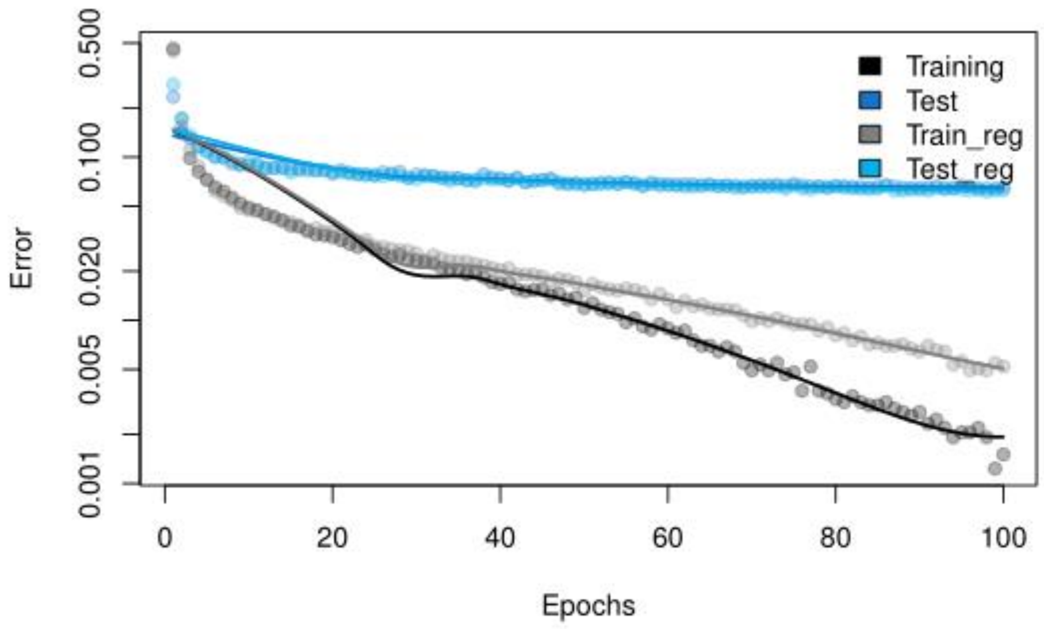
```
model_reg_3h <- keras_model_sequential() %>%
layer_dense(units = 256, activation = "relu", input_shape = v,#hidden layer 1
kernel_regularizer = regularizer_l2(l = 0.001)) %>%
layer_dense(units = 64, activation = "relu",#hidden layer 2
kernel_regularizer = regularizer_l2(l = 0.001)) %>%
  layer_dense(units = 32, activation = "relu",#hidden layer 3
kernel_regularizer = regularizer_l2(l = 0.001)) %>%
layer_dense(units = 10, activation = "softmax") %>%
compile(
loss = "categorical_crossentropy",
optimizer = optimizer_sgd(),
metrics = "accuracy"
)
```

Observations and Conclusions

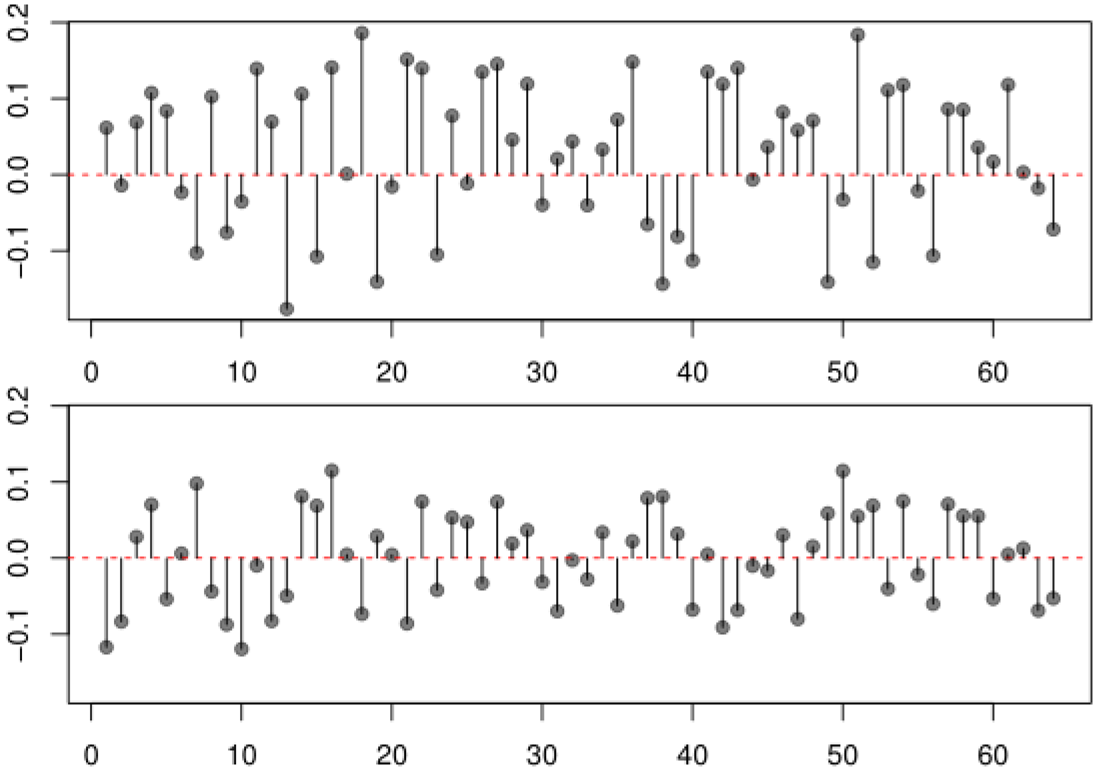
- Incase of the first model, as we increase the number of epochs, the model’s accuracy keeps on improving for the training data which makes sense as we are feeding in the same data again and again, but for testing data, it reaches a saturation level soon after which there is not much improvement.



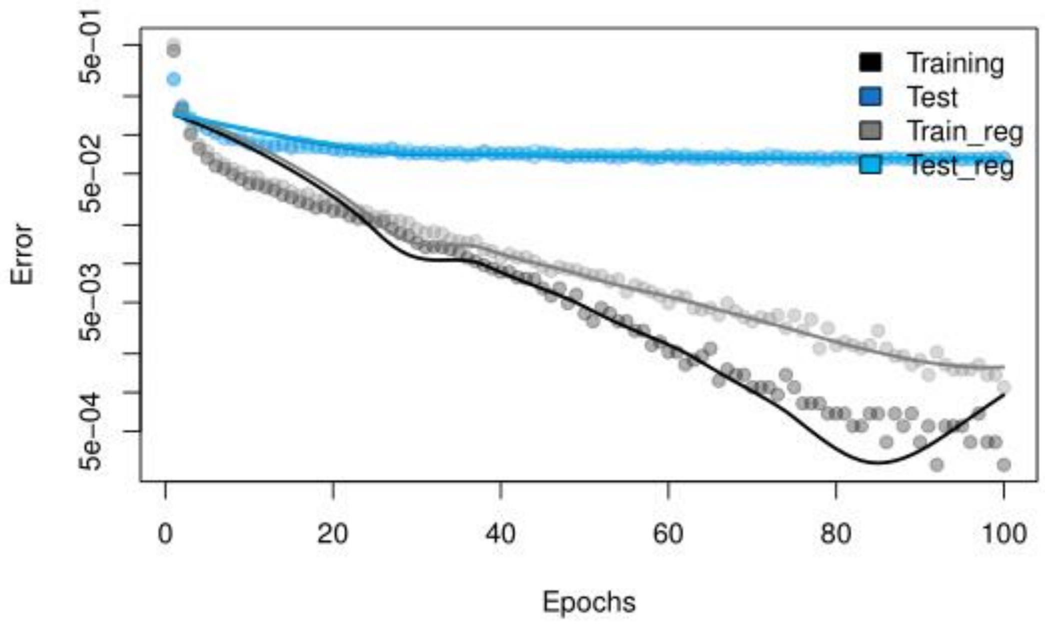
- When we compare this data to regularized model, the regularized model has almost same accuracy as our regular model, but a lower accuracy for training data. This is because random information is induced in this data.

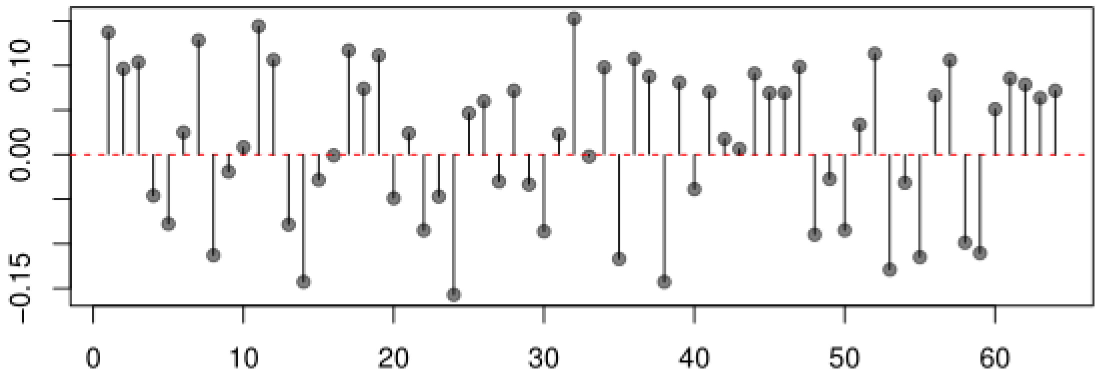


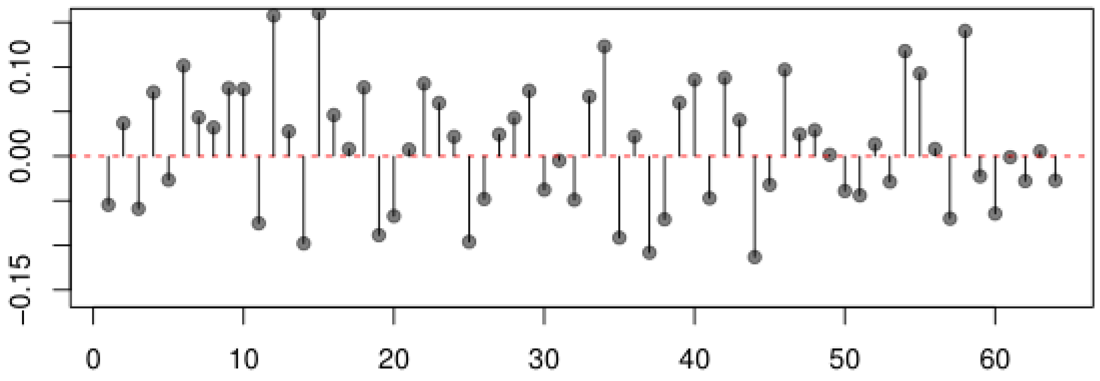
- We can compare the parameters of the 2 models, we can see that our initial model has relatively higher absolute values than our regularized model.



- Incase of the 3 hidden layer architecture, the accuracy for the testing data remains almost the same, but there is a significant improvement incase of training data. The accuracy of training data improves.







Model	Training Data Error	Testing Data Error
256 x 64 x 10 (normal)	0.001234412	0.064275026
256 x 64 x 10 (regularized)	0.004937589	0.061285496
256 x 64 x 32 x 10 (normal)	0.0002743006	0.0642750263
256 x 64 x 32 x 10 (regularized)	0.0010972619	0.0627802610

The accuracy of testing data is almost the same in all 4 cases, but we can see that regularized models have lower accuracy on training data which is due to adding information to data. It helps in avoiding overfitting, but in this case there is not much improvement.

Appendix

R-Code

```
---
title: "ML and AI 2"
author: "Shubhang Periwal 19201104"
date: "3/29/2020"
output:
  pdf_document: default
  word_document: default
---
```

~~~
```{r}
library(keras)
tensorflow::tf$random$set_seed(0)
load("D:/data_usps_digits.RData")
```
~~~

~~~
```{r}
# helper function
# to plot the digit
plot_digit <- function(index, data) {
tmp <- (-data + 1) / 2 # to convert back to original
z <- matrix( data = as.numeric(data[index, 256:1]), 16, 16 )
image(z[16:1,1:16], col = gray((1:100)/100),
xaxt = "n", yaxt = "n")
}

# plot few example digits
par(mfrow = c(1,2), mar = rep(1.5, 4))
plot_digit(14, x_train)
plot_digit(900, x_train)

```
~~~

~~~
```{r}
#preprocessing
# Convert y to categorical using one-hot encoding
y_train <- to_categorical(y_train, num_classes = 10)
y_test <- to_categorical(y_test , num_classes = 10)
# convert x_train and x_test from data frame to matrix for valid network input
x_train <- as.matrix(x_train)
x_test <- as.matrix(x_test)

# normalizen x(input) to 0-1
range_norm <- function(x, a = 0, b = 1) {
( (x - min(x)) / (max(x) - min(x)) )*(b - a) + a }

x_train <- apply(x_train, 2, range_norm)
x_test <- apply(x_test, 2, range_norm)
range(x_train)
range(x_test)
```
~~~

~~~
```{r}
V <- ncol(x_train) # 256
model_2h <- keras_model_sequential() %>%
layer_dense(units = 256, activation = "relu", input_shape = V,) %>%#input layer
  #hidden layer 1
layer_dense(units = 64, activation = "relu") %>%
  #hidden layer 2
layer_dense(units = 10, activation = "softmax") %>%
  #output layer
compile(
loss = "categorical_crossentropy", metrics = "accuracy",
optimizer = optimizer_sgd(),
)
# count parameters
count_params(model_2h)
```



```{r}
# fit the model on the training data
# and evaluate on test data at each epoch
fit <- model_2h %>% fit(
x = x_train, y = y_train,
validation_data = list(x_test, y_test),
epochs = 100,
verbose = 0
)
```

```{r, echo=FALSE}
# to add a smooth line to points
smooth_line <- function(y) {
x <- 1:length(y)
out <- predict( loess(y ~ x) )
return(out)
}
# some colors will be used later
cols <- c("black", "dodgerblue3", "gray50", "deepskyblue2")
# check performance ---> error
out <- 1 - cbind(fit$metrics$accuracy,
fit$metrics$val_accuracy)
matplot(out, pch = 19, ylab = "Error", xlab = "Epochs",
col = adjustcolor(cols[1:2], 0.3),
log = "y") # on log scale to visualize better differences
matlines(apply(out, 2, smooth_line), lty = 1, col = cols[1:2], lwd = 2)
legend("topright", legend = c("Training", "Test"),
~~~

```
fill = cols[1:2], bty = "n")

```

```{r, echo=FALSE}
# regularized model
model_2hreg <- keras_model_sequential() %>%
layer_dense(units = 256, activation = "relu", input_shape = V,
kernel_regularizer = regularizer_l2(l = 0.001)) %>%#hidden layer 1
layer_dense(units = 64, activation = "relu",#hidden layer 2
kernel_regularizer = regularizer_l2(l = 0.001)) %>%
layer_dense(units = 10, activation = "softmax") %>%
compile(
loss = "categorical_crossentropy",
optimizer = optimizer_sgd(),
metrics = "accuracy"
)

# count parameters
count_params(model_2hreg)
```

```{r, echo=FALSE}
# train and evaluate on test data at each epoch
fit_reg <- model_2hreg %>% fit(
x = x_train, y = y_train,
validation_data = list(x_test, y_test),
epochs = 100,
verbose = 1
)
```

We compare the performance of the regularized model against the performance of the unregularized one.

```{r}

out <- 1 - cbind(fit$metrics$accuracy,
fit$metrics$val_accuracy,
fit_reg$metrics$accuracy,
fit_reg$metrics$val_accuracy)
# check performance
matplot(out, pch = 19, ylab = "Error", xlab = "Epochs",
col = adjustcolor(cols, 0.3),
log = "y")
```

```
matlines(apply(out, 2, smooth_line), lty = 1, col = cols, lwd = 2)
legend("topright", legend = c("Training", "Test", "Train_reg", "Test_reg"),
fill = cols, bty = "n")

```

```{r}
apply(out, 2, min)
```

```{r, echo=FALSE}
# get all weights
w_all <- get_weights(model_2h)
w_all_reg <- get_weights(model_2hreg)
# weights of first hidden layer
# one input --> 64 units
w <- w_all[[3]][1,]
w_reg <- w_all_reg[[3]][1,]
# compare visually the magnitudes
par(mfrow = c(2,1), mar = c(2,2,0.5,0.5))
r <- range(w)
n <- length(w)
plot(w, ylim = r, pch = 19, col = adjustcolor(1, 0.5))
abline(h = 0, lty = 2, col = "red")
segments(1:n, 0, 1:n, w)
#
plot(w_reg, ylim = r, pch = 19, col = adjustcolor(1, 0.5))
abline(h = 0, lty = 2, col = "red")
segments(1:n, 0, 1:n, w_reg)
```

```{r, echo=FALSE}
model_3h <- keras_model_sequential() %>%
layer_dense(units = 256, activation = "relu", input_shape = V) %>%#input layer
  #hidden layer 1
layer_dense(units = 64, activation = "relu") %>%
  #hidden layer 2
  layer_dense(units = 32, activation = "relu") %>%
  #hidden layer 3
layer_dense(units = 10, activation = "softmax") %>%
  #output layer
compile(
loss = "categorical_crossentropy", metrics = "accuracy",
optimizer = optimizer_sgd(),
)
```

~~~
# count parameters
count_params(model_3h)
# fit the model on the training data
# and evaluate on test data at each epoch
fit_3h <- model_3h %>% fit(
x = x_train, y = y_train,
validation_data = list(x_test, y_test),
epochs = 100,
verbose = 0
)
```



```{r}
model_reg_3h <- keras_model_sequential() %>%
layer_dense(units = 256, activation = "relu", input_shape = V,#hidden layer 1
kernel_regularizer = regularizer_l2(l = 0.001)) %>%
layer_dense(units = 64, activation = "relu",#hidden layer 2
kernel_regularizer = regularizer_l2(l = 0.001)) %>%
  layer_dense(units = 32, activation = "relu",#hidden layer 3
kernel_regularizer = regularizer_l2(l = 0.001)) %>%
layer_dense(units = 10, activation = "softmax") %>%
compile(
loss = "categorical_crossentropy",
optimizer = optimizer_sgd(),
metrics = "accuracy"
)

# count parameters
count_params(model_reg_3h)

# fit the model on the training data
# and evaluate on test data at each epoch
fit_3hreg <- model_reg_3h %>% fit(
x = x_train, y = y_train,
validation_data = list(x_test, y_test),
epochs = 100,
verbose = 0
)
```

```{r}


out <- 1 - cbind(fit_3h$metrics$accuracy,
fit_3h$metrics$val_accuracy,
~~~

~~~
fit_3hreg$metrics$accuracy,
fit_3hreg$metrics$val_accuracy)
# check performance
matplot(out, pch = 19, ylab = "Error", xlab = "Epochs",
col = adjustcolor(cols, 0.3),
log = "y")

matlines(apply(out, 2, smooth_line), lty = 1, col = cols, lwd = 2)
legend("topright", legend = c("Training", "Test", "Train_reg", "Test_reg"),
fill = cols, bty = "n")

```


```{r}
apply(out, 2, min)
```



```{r, echo=FALSE}
# get all weights
w_all <- get_weights(model_3h)
w_all_reg <- get_weights(model_reg_3h)
# weights of first hidden layer
# one input --> 64 units
w <- w_all[[3]][1,]
w_reg <- w_all_reg[[3]][1,]
# compare visually the magnitudes
par(mfrow = c(2,1), mar = c(2,2,0.5,0.5))
r <- range(w)
n <- length(w)
plot(w, ylim = r, pch = 19, col = adjustcolor(1, 0.5))
abline(h = 0, lty = 2, col = "red")
segments(1:n, 0, 1:n, w)
#
plot(w_reg, ylim = r, pch = 19, col = adjustcolor(1, 0.5))
abline(h = 0, lty = 2, col = "red")
segments(1:n, 0, 1:n, w_reg)
```
~~~