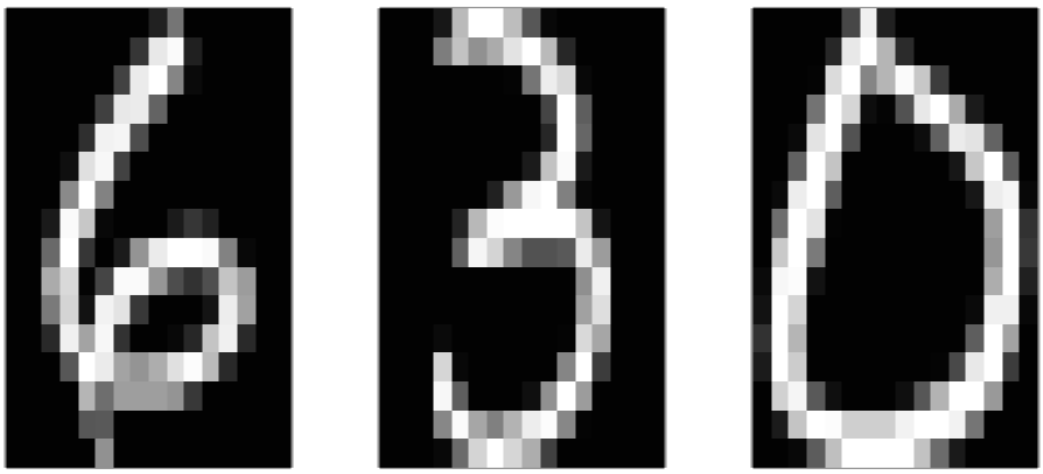
ML and AI ASSIGNMENT 02

Nisarga G-19203753

29/03/2020

Loading and preparing data: We must perform one hot encoding of target varaible.One hot encoding is a vector representation where all the elements of the vector are 0 except one, which has 1 as its value.Also normalise x_train and x_test such that all the values lie between 0-1.

```
#install.packages("keras")
#install.packages("tensorflow")
library(tensorflow)
library(keras)
load("data_usps_digits.RData")

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")
}

#Plotting few digits in 16 X 16 grid:
par(mfrow = c(1,3), mar = rep(1.5, 4))
plot_digit(100, x_train)
plot_digit(7, x_train)
plot_digit(9, x_train)
```



```
#one hot encoding of target variable.
y_train<-to_categorical(y_train-1)
```

```
y_test<-to_categorical(y_test-1)

#normalize x_train and x_test.
range_norm <- function(x, a = 0, b = 1) {
( (x - min(x)) / (max(x) - min(x)) )*(b - a) + a }

x_test <- apply(x_test, 2, range_norm)
x_train <- apply(x_train, 2, range_norm)

range(x_test)

## [1] 0 1

range(x_train)

## [1] 0 1
```

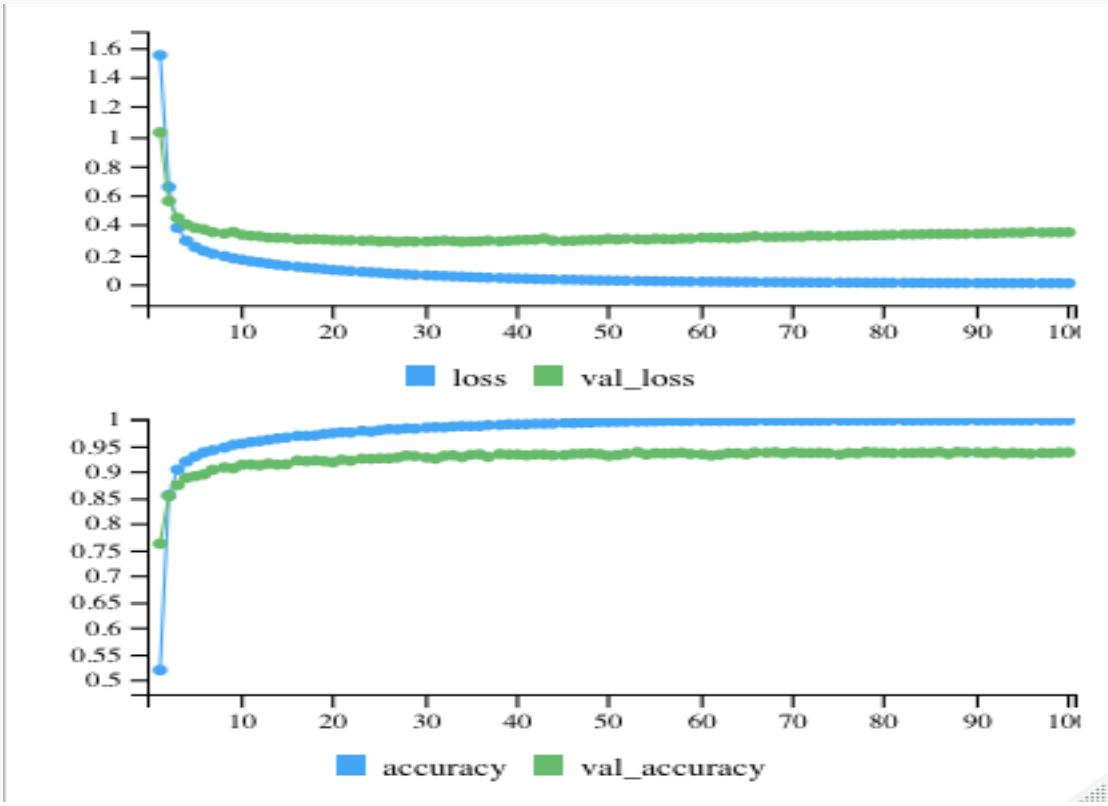
Multilayer neural network: Using the functionalities of keras we deploy a neural network with 2 hidden layer with ReLU activation and one output layer with softmax activation.In the first layer we consider 256 units, as the input units, then in the next layer we set the number of neurons by halving the number of units of the previous level i.e 128(total 2 layers).To compare predictive performance improvement obtained by adding one extra hidden layer to the network architecture, we create model1 with 3 hidden layers.

```
V<-ncol(x_train)
#model with 2 hidden layers
model <- keras_model_sequential() %>%
 layer_dense(units = 256, activation = "relu", input_shape = V) %>%
 layer_dense(units = 128, activation = "relu") %>%
 layer_dense(units = 9, activation = "softmax") %>%
 compile(
 loss = "categorical_crossentropy", metrics = "accuracy",
 optimizer = optimizer_sgd(),
)
#model with 3 hidden layers
model1 <- keras_model_sequential() %>%
 layer_dense(units = 256, activation = "relu", input_shape = V) %>%
 layer_dense(units = 128, activation = "relu") %>%
 layer_dense(units= 64,activation = "relu") %>%
 layer_dense(units = 9, activation = "softmax") %>%
 compile(
 loss = "categorical_crossentropy", metrics = "accuracy",
 optimizer = optimizer_sgd(),
)

count_params(model)

## [1] 99849

count_params(model1)
```

```
## [1] 107529
```

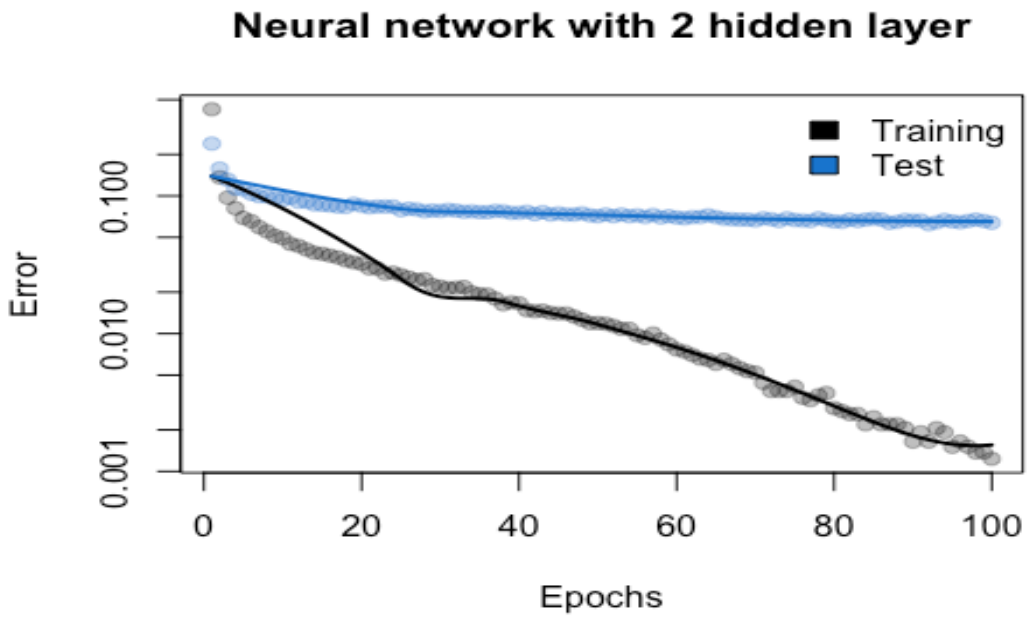
Fit the model : The function fit is used to train both model and model1.Here we consider 100 epochs. The function allows also to evaluate the performance of the model on a separate test set at each epoch, to monitor the progress during the training process.Argument verbose = 1 produces an interactive visualization of the training procedure. The output of fit is a list which contains the configuration parameters of the model (in params) and performance metrics (in metrics).The performance metrics are the accuracy and value of error function at each epoch for training and test set. We can look at the error progress over the epochs.

```
#fit model1
fit1 <- model %>% fit(
 x = x_train, y = y_train,
 validation_data = list(x_test, y_test),
 epochs = 100,
 verbose = 1
)
#fit model2
fit2 <- model1 %>% fit(
 x = x_train, y = y_train,
 validation_data = list(x_test, y_test),
 epochs = 100,
 verbose = 1
)
```

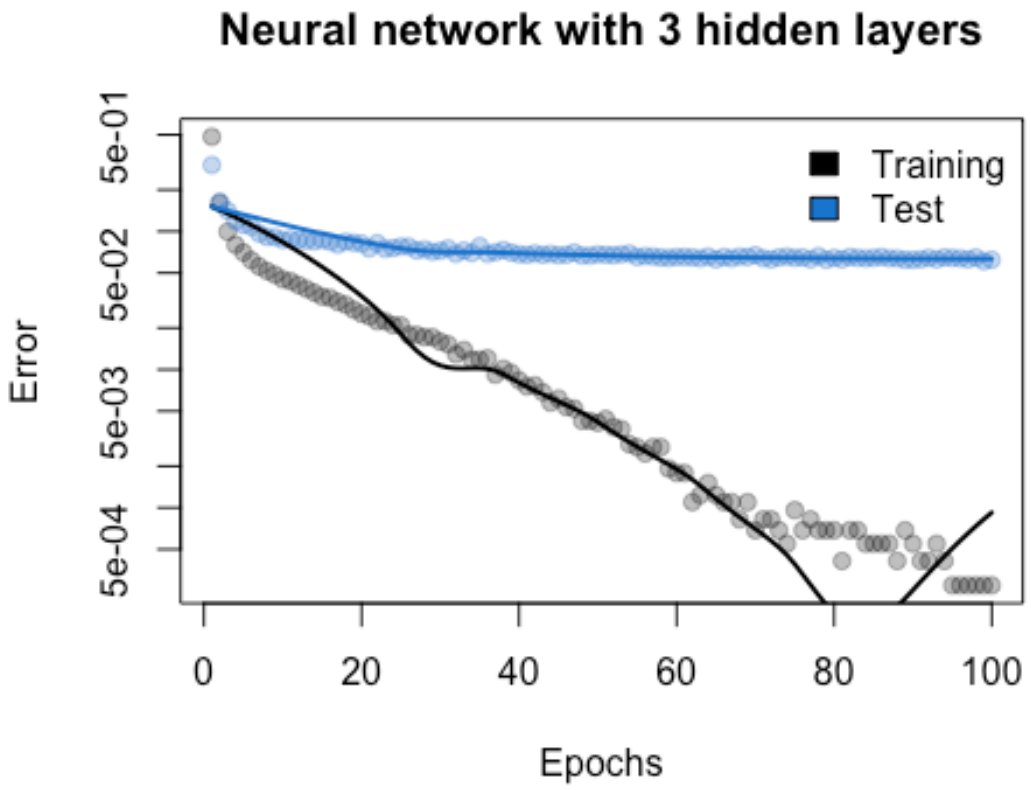


Performance of model with 2 hidden layers: The classification error is going to be 0.0015 after a certain number of epochs, while the test error is around 0.065. This could be a sign of overfitting. Moreover, it seems that after a certain number of epochs training the model does not improve performance. Performance of model1 with 3 hidden layers: The classification error is 0.00027 after a certain number of epochs and rises after certain number of epochs, while the test error is approximately 0.061. This is the difference between model and model1.In model with three layers , classification error is lesser and test error is also lesser ,hence performance increases by adding one layer on top of 2 layer model.

```
smooth_line <- function(y) {
 x <- 1:length(y)
 out <- predict( loess(y ~ x) )
 return(out)
}
cols <- c("black", "dodgerblue3", "gray50", "deepskyblue2")

out <- 1 - cbind(fit1$metrics$accuracy,
                 fit1$metrics$val_accuracy)

matplot(out, pch = 19, ylab = "Error", xlab = "Epochs",main = "Neural network
with 2 hidden layer",
        col = adjustcolor(cols[1:2], 0.3),
        log = "y")

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")
```



```
out1 <- 1 - cbind(fit2$metrics$accuracy,
        fit2$metrics$val_accuracy)

matplot(out1, pch = 19, ylab = "Error", xlab = "Epochs",main="Neural network
with 3 hidden layers",
col = adjustcolor(cols[1:2], 0.3),
log = "y")

matlines(apply(out1, 2, smooth_line), lty = 1, col = cols[1:2], lwd = 2)
legend("topright", legend = c("Training", "Test"),
fill = cols[1:2], bty = "n")
```

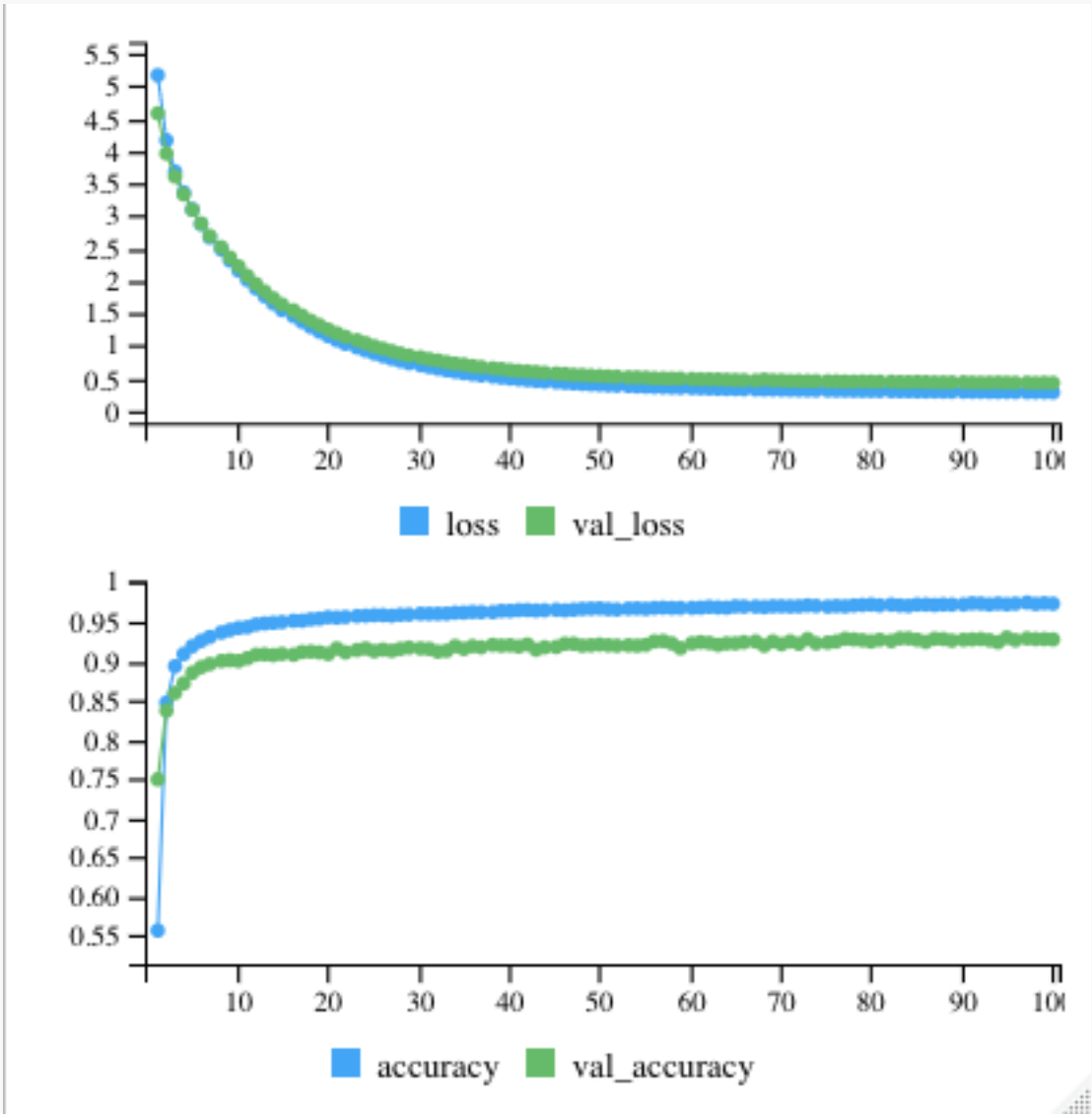


PENALTY BASED REGULARIZATION:

The argument used to set a regularization term applied to the weights of each layer is kernel_regularizer.

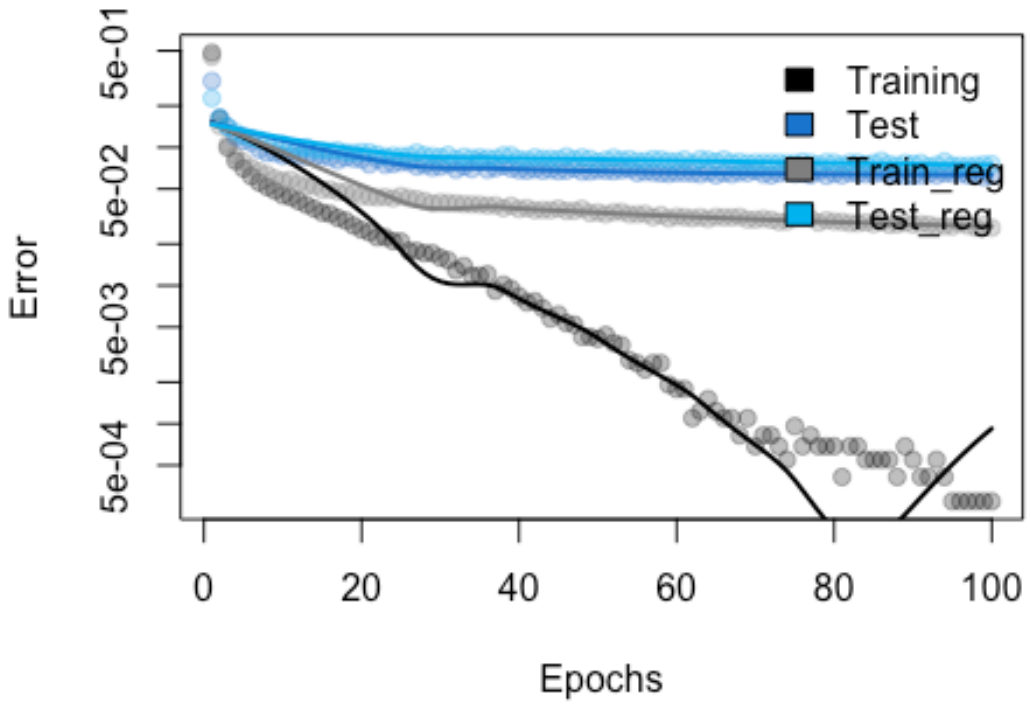
```
model_reg <- keras_model_sequential() %>%
        layer_dense(units = 256, activation = "relu", input_shape = V,
                kernel_regularizer = regularizer_l2(l = 0.009)) %>%
        layer_dense(units = 128, activation = "relu",
                kernel_regularizer = regularizer_l2(l = 0.009)) %>%
        layer_dense(units = 9, activation = "softmax") %>%
```

```
compile(
loss = "categorical_crossentropy", optimizer = optimizer_sgd(),
metrics = "accuracy" )
# train and evaluate on test data at each epoch
fit_reg <- model_reg %>% fit(
        x = x_train, y = y_train,
        validation_data = list(x_test, y_test),
        epochs = 100,
        verbose = 1
)
```



Below we can compare the performance of the regularized model against the performance of the unregularized one.The test error of the regularized model is slightly higher than the test error of the unregularized one. This indicates that regularization can have adverse effects. Rather there was no overfitting previously.

```
out <- 1 - cbind(fit2$metrics$accuracy,
                 fit2$metrics$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")
```




```
apply(out, 2, min)
```

```
## [1] 0.0002743006 0.0607872605 0.0261966586 0.0727453828
```

We can inspect the weights of the two models. We consider one instance of the weights of the first hidden layer. The weights of the regularized model are lower in magnitude.

```
# get all weights
w_all <- get_weights(model)
w_all_reg <- get_weights(model_reg)
# weights of first hidden layer # one input --> 64 units
w <- w_all[[3]][1,]
```

```r
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)
```