ML_and_AI-Assignment3

Nisarga G-19203753

20/04/2020

Preparation of data: x_train and x_test are converted into matrix and then normalized.we must perform hot encoding of the target variable using function to_categorical. Using one hot encoding, we have converted the 10 categories (0-9) of the target variable into 10 dummy variables captured by 0s and 1s.

```
library(keras)
library(tensorflow)
library(tfruns)
library(jsonlite)
set.seed(19203753)

#Loading the dataset
load("data_usps_digits.RData")

#Change independent variables to matrix
x_test <- as.matrix(x_test)
x_train <- as.matrix(x_train)

#Range normalize the input variables
#Input train data
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)
range(x_train)

## [1] 0 1

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

## [1] 0 1

dim(x_train)

## [1] 7291  256

#Convert the target variable (dependent) to one hot encoded vectors
y_train <- to_categorical(y_train, num_classes = 10)
y_test <- to_categorical(y_test, num_classes = 10)
dim(y_train)
```

```
## [1] 7291   10

#Calculating rows and columns of x_train
N <- nrow(x_train)
V <- ncol(x_train)

#Splitting the test data in half for validation and model tuning
#and the other half for actual testing and evaluation of the generalized
#predictive performance
val <- sample(1:nrow(x_test), 1003)
test <- setdiff(1:nrow(x_test), val)
x_val <- x_test[val,]
y_val <- y_test[val,]
x_test <- x_test[test,]
y_test <- y_test[test,]
dim(x_val)

## [1] 1003  256

dim(y_val)

## [1] 1003   10

dim(x_test)

## [1] 1004  256

dim(y_test)

## [1] 1004   10
```

"tfruns" package is installed and loaded to effectively tune the model configuration.

config_file.R:A separate R script file is used where we have the model instantiation and declaration of the flags that control the number of nodes within each layer and dropout rate. We have deployed a multilayer neural network using 2 hidden layers with ReLU (Rectified Linear Units) activation function and the output layer with softmax output activation.

(Configuration file is provided at the end of this report ,also as a separate r file.)

Grid Search: We have set a grid of values for both the hyperparameters we are interested in tuning, i.e. "number of nodes within each layer" and "dropout rate", setting 4 values each for each hyperparameter for each layer, resulting in 4 X 4 X 4 X 4 = 256 possible model configurations to be evaluated.

"tuning_run" function is used to perform an exhaustive grid search, running all combinations of the specified training flags defined in a pre-specified grid. 256 possible combinations will take hours of time to evaluate. So we have considered a random sub-sample of 20% of the total possible configurations, thus running only 51 models. We have

also specified the directory name as “runs_example” where all the runs for 51 models will be stored.

```
#Setting a grid of values for the flags/hyperparameters
dense_units1_set <- c(128, 64, 32, 16)
dropout1_set <- c(0.0, 0.1, 0.2, 0.4)
dense_units2_set <- c(64, 32, 16, 8)
dropout2_set <- c(0.0, 0.1, 0.2, 0.4)

#Hyperparameter tuning
runs <- tuning_run("config_file.R",
                   runs_dir = "runs_example",
                   flags = list(
                     dense_units1 = dense_units1_set,
                     dropout1 = dropout1_set,
                     dense_units2 = dense_units2_set,
                     dropout2 = dropout2_set),
                   sample = 0.2)
```

NOTE : The output has been deleted in order to keep the report concise.

Determine Optimal configuration: Now we have a folder named “runs_example” in our working directory which has all the results and information about all the 51 runs. “jsonlite” package is installed and loaded to get the scores stored in .json format in all the runs folders in “runs_example” directory. Two functions are used to explore the results - read_metrics (to extract the learning scores) and plot_learning curve (to plot the corresponding validation learning curves for all the sampled model configurations).

read_metrics function:Learning scores for 51 runs are extracted using the function read_metrics. The output infers the different parameters of the learning scores, which includes loss during each epoch before convergence, accuracy, validation loss, validation accuracy, flags for each hyperparameter for each hidden layer, evaluation loss and evaluation accuracy. We have also calculated the validation accuracy for each epoch for each run having 51 combinations of tuned hyperparameters. It can be observed that the maximum validation accuracy for any run (out of 51 sampled runs) is 0.9053.

Validation learning curve: A validation learning curve is a plot of model learning performance over number of iterations with different combinations of hyperparameters, i.e. it shows the curve of validation accuracy over number of epochs at convergence. In our case, we can see 4 runs colored in blue. These runs are the top 4 runs that have the highest validation accuracy at convergence and the convergence is mostly before 80 epochs. The curves show that there are many model configurations that led to a validation accuracy larger than 0.80, with few runs having even more validation accuracy.

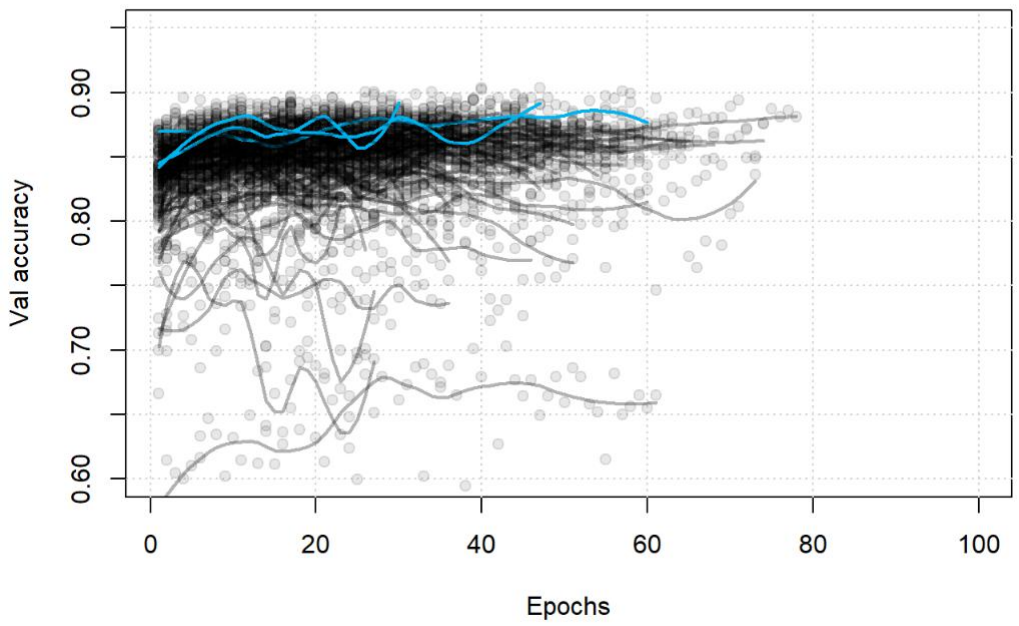
```
#Extracting values from the stored runs
read_metrics <- function(path, files =NULL)
{
  path <- paste0(path, "/")
```

```r
  if(is.null(files)) files <- list.files(path)
  n <- length(files)
  out <- vector("list", n)
  for(i in 1:n) {
    dir <- paste0(path, files[i], "/tfruns.d/")
    out[[i]] <- jsonlite::fromJSON(paste0(dir, "metrics.json"))
    out[[i]]$flags <- jsonlite::fromJSON(paste0(dir, "flags.json"))
    out[[i]]$evaluation <- jsonlite::fromJSON(paste0(dir,"evaluation.json"))
  }
  return(out)
}

#Plotting the corresponding validation learning curves
plot_learning_curve <- function(x, ylab = NULL, cols = NULL, top = 3,
                                span = 0.4, ...)
  {
  smooth_line <- function(y) {
    x <- 1:length(y)
    out <- predict(loess(y~x, span = span))
    return(out)
    }
  matplot(x, ylab = ylab, xlab = "Epochs", type = "n", ...)
  grid()
  matplot(x, pch = 19, col = adjustcolor(cols, 0.3), add = TRUE)
  tmp <- apply(x, 2, smooth_line)
  tmp <- sapply(tmp, "length<-", max(lengths(tmp)))
  set <- order(apply(tmp, 2, max, na.rm = TRUE), decreasing = TRUE)[1:top]
  cl <- rep(cols, ncol(tmp))
  cl[set] <- "deepskyblue2"
  matlines(tmp, lty = 1, col = cl, lwd = 2)
}

#Extracting results
out <- read_metrics("runs_example")

#Extracting validation accuracy and plotting learning curve
acc <- sapply(out, "[[", "val_accuracy")
plot_learning_curve(acc, col = adjustcolor("black", 0.3),
                    ylim = c(0.60, 0.95), ylab = "Val accuracy", top = 4)
```



Best model: We have extracted runs with validation accuracy larger than 0.87 ("res" object). The result shows a data frame with the hyperparameter combinations of number of nodes and the dropout rate for both the hidden layers. It also consists of validation and test accuracy associated to each run which have validation accuracy greater than 0.87. We have also extracted the model configuration and performance of the top 10 runs so as to check what accuracy or loss they behold with what combination of number of nodes and dropout rate for both hidden layers.

It can be observed that the best model (within the random sub-sample of the grid) is the one which has the highest validation accuracy (0.8943), i.e. 89.43% chances are there that the type of digit is predicted correctly on the validation set and eval_accuracy 0.9084, i.e. 90.84% accuracy is there that predictions on the type of digit will be correct when tested on the test data. Generalization performance is quite good with almost 90% accuracy in predictions and after tuning, the best model should have 64 nodes in the first hidden layer and 64 nodes/units in the second hidden layer with dropout rate of 0.0 in both the hidden layers.

NOTE : The output has been deleted in order to keep the report below 5 pages.

```
#Extracting runs with validation accuraAcy larger than 0.87
res <- ls_runs(metric_val_accuracy > 0.87,
               runs_dir = "runs_example", order = metric_val_accuracy)
res

#Best hyperparameter values
res <- res[,c(2,4,8:11)]
res[1:10,]
```

```r
#Extracting run with maximum validation accuracy
res[which.max(res$metric_val_accuracy),]
```

##Configuration file

```r
#Setting default flags
FLAGS <- flags(flag_integer("dense_units1", 64),
               flag_integer("dense_units2", 32),
               flag_numeric("dropout1", 0.1),
               flag_numeric("dropout2", 0.1))

#Model configuration
model <- keras_model_sequential() %>%
  layer_dense(units = FLAGS$dense_units1, input_shape = ncol(x_train),
              activation = "relu", name = "layer_1",
              kernel_regularizer = regularizer_l2(0.01)) %>%
  layer_dropout(rate = FLAGS$dropout1) %>%
  layer_dense(units=FLAGS$dense_units2, activation="relu",name="layer_2",
              kernel_regularizer = regularizer_l2(0.01)) %>%
  layer_dropout(rate = FLAGS$dropout2) %>%
  layer_dense(units = ncol(y_train), activation = "softmax",
              name = "layer_out") %>%
  compile(loss = "categorical_crossentropy", metrics = "accuracy",
          optimizer = optimizer_adam(lr = 0.01),)

#Training and evaluation
fit <- model %>% fit(
  x = x_train, y = y_train,
```

```
  validation_data = list(x_val, y_val),
  epochs = 100,
  batch_size = 32,
  verbose = 1,
  callbacks = callback_early_stopping(
    monitor = "val_accuracy", patience = 20))

#Storing accuracy on test set for each run
score <- model %>%
  evaluate(x_test, y_test, verbose = 0)
```