CS573 Lab 2

Urminder Singh

March 2, 2018

Data

I downloaded the data from https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits. I used the optdigits.tra as training set and optdigits.tes as test set. Further, I divided training data into two non-overlapping parts having 80% and 20% data respectively. Then I used the partition with 80% data as training set to train the models and 20% data as a validation set for the models.

Experiment 1a

For Experiment 1 I chose the hidden layers to be ReLU and I changed other parameters. I iterated over combinations of following parameters and buit models and tested them.

Parameters

- Error Function: Quadratic, CrossEntropy
- Hidden Layers: 1, 2, 3
- Hidden Units: 100, 200, 300
- Learning Rate: 0.005, 0.01
- Momentum Start: 0, 0.5
- Input Scaling: True, False

I wrote the attached R code for simulation (using h2o). The results are described in Table 1.

Experiment 1a R code

```
library("magrittr", lib.loc = "~/R/x86_64-pc-linux-gnu-library/3.4")
library(h2o)
library(data.table)
train <- fread("optdigits.tra", stringsAsFactors = T, colClasses = c(rep("numeric",
    64), "character"))
test <- fread("optdigits.tes", stringsAsFactors = T, colClasses = c(rep("numeric",
    64), "character"))
h2o.init(nthreads = -1, enable_assertions = FALSE)
```

```
##
## H2O is not running yet, starting it now...
##
## Note:  In case of errors look at the following log files:
##     /tmp/RtmpO2JTiy/h2o_usingh_started_from_r.out
##     /tmp/RtmpO2JTiy/h2o_usingh_started_from_r.err
##
##
## Starting H2O JVM and connecting: . Connection successful!
```

```
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         1 seconds 353 milliseconds
##     H2O cluster version:        3.16.0.2
##     H2O cluster version age:    3 months and 8 days
##     H2O cluster name:           H2O_started_from_R_usingh_hwg248
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   3.48 GB
##     H2O cluster total cores:    8
##     H2O cluster allowed cores:  8
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     H2O API Extensions:         XGBoost, Algos, AutoML, Core V3, Core V4
##     R Version:                  R version 3.4.1 (2017-06-30)
```

```r
h2o.no_progress()
# Divide into train and test/validation
c.train <- train[1:(nrow(train) * 0.8), ]
c.validation <- train[(nrow(train) * 0.2):nrow(train), ]
# data to h2o cluster
train.h2o <- as.h2o(c.train)
validation.h2o <- as.h2o(c.validation)
test.h2o <- as.h2o(test)
# last variable is the class category
y.dep <- "V65"
predictors <- setdiff(names(c.train), y.dep)
###### Set model hyper-parameters
hiddenLayers <- c(1, 2, 3)
hiddenUnits <- c(100, 200, 300)
learningRates <- c(0.005, 0.01)
momentumStart <- c(0, 0.5)
inputScaling <- c(T, F)
errorFunc <- c("Quadratic", "CrossEntropy")
# hiddenLayers<-c(1) hiddenUnits<-c(100,200) learningRates<-c(0.01)
# momentumStart<-c(0) inputScaling<-c(T) errorFunc<-c('Quadratic') Table to
# write results
header1 <- c("ErrorFunction", "layers", "hiddenUnits", "learnRate", "momentumStart",
    "Scale", "Acc_val", "Acc_test", "Time in min")
resultsTab <- data.frame(matrix(ncol = 9, nrow = 0))
colnames(resultsTab) <- header1
# set seed for reproducible results
set.seed(1263)
# save the best model i.e. highest accuraccy on test set
bestModel <- NULL
bestAcc <- 0
# make all comniations of the parameters
for (errF in errorFunc) {
    for (hL in hiddenLayers) {
        for (hU in hiddenUnits) {
            for (lR in learningRates) {
                for (mS in momentumStart) {
```

```r
                  for (iS in inputScaling) {

                    # build model and calculate accuraccy
                    s <- proc.time()  #start time
                    model1 <- h2o.deeplearning(x = predictors, y = y.dep, training_frame = train.h2o,
                      validation_frame = validation.h2o, hidden = c(rep(hU),
                        hL), activation = "Rectifier", epochs = 150, loss = errF,
                      rate = lR, momentum_start = mS, standardize = iS, adaptive_rate = F)
                    d <- proc.time() - s  #end time
                    # print('Model training metrics') model1@model$training_metrics print('Model
                    # validation metrics') model1@model$validation_metrics
                    # model1@model$training_metrics@metrics$model_category
                    # h2o.confusionMatrix(model1) test on testdata cat('Performance on test
                    # data:') perf<-h2o.performance(model1,test.h2o) perf compute accuraccy on
                    # validation
                    valResult <- h2o.predict(model1, validation.h2o, y = y.dep)
                    predictions <- as.data.frame(valResult[, 1])
                    trueLabels <- c.validation$V65
                    correct <- 0
                    for (i in 1:dim(predictions)[1]) {
                      if (as.numeric(predictions$predict[i]) == as.numeric(c.validation$V65[i])) {
                        correct <- correct + 1
                      }
                    }
                    acc_V <- format(correct/dim(predictions)[1], digits = 4)
                    # cat('Accuraccy on validation set:',acc_V) compute accuraccy on test
                    testResult <- h2o.predict(model1, test.h2o, y = y.dep)
                    predictions <- as.data.frame(testResult[, 1])
                    trueLabels <- test$V65
                    correct <- 0
                    for (i in 1:dim(predictions)[1]) {
                      if (as.numeric(predictions$predict[i]) == as.numeric(test$V65[i])) {
                        correct <- correct + 1
                      }
                    }
                    acc <- format(correct/dim(predictions)[1], digits = 4)
                    cat("Accuraccy on test set:", acc)
                    resultsTab[nrow(resultsTab) + 1, ] <- c(errF, hL, hU, lR,
                      mS, iS, acc_V, acc, format(as.numeric(d)[3]/60, digits = 2))

                    if (acc > bestAcc) {
                      bestModel <- model1
                      bestAcc <- acc
                    }
                  }
                }
            }
        }
    }
}
```

```
## Accuraccy on test set: 0.1196Accuraccy on test set: 0.09905Accuraccy on test set: 0.09293Accuraccy o
```

```r
resultsTab %>% knitr::kable(caption = "Experiment 1 outcomes. Hidden units were ReLU.")
```

Table 1: Experiment 1 outcomes. Hidden units were ReLU.

ErrorFunction	layers	hiddenUnits	learnRate	momentumStart	Scale	Acc_val	Acc_test	Time in min
Quadratic	1	100	0.005	0	TRUE	0.1314	0.1196	0.076
Quadratic	1	100	0.005	0	FALSE	0.09905	0.09905	0.088
Quadratic	1	100	0.005	0.5	TRUE	0.1	0.09293	0.056
Quadratic	1	100	0.005	0.5	FALSE	0.09905	0.09905	0.041
Quadratic	1	100	0.01	0	TRUE	0.2037	0.2126	0.039
Quadratic	1	100	0.01	0	FALSE	0.09905	0.09905	0.039
Quadratic	1	100	0.01	0.5	TRUE	0.1824	0.1669	0.039
Quadratic	1	100	0.01	0.5	FALSE	0.09971	0.09905	0.04
Quadratic	1	200	0.005	0	TRUE	0.1085	0.1091	0.073
Quadratic	1	200	0.005	0	FALSE	0.09905	0.09905	0.057
Quadratic	1	200	0.005	0.5	TRUE	0.1978	0.1987	0.074
Quadratic	1	200	0.005	0.5	FALSE	0.09905	0.09905	0.055
Quadratic	1	200	0.01	0	TRUE	0.09905	0.09905	0.039
Quadratic	1	200	0.01	0	FALSE	0.09905	0.09905	0.056
Quadratic	1	200	0.01	0.5	TRUE	0.09905	0.1013	0.056
Quadratic	1	200	0.01	0.5	FALSE	0.1095	0.1046	0.057
Quadratic	1	300	0.005	0	TRUE	0.1213	0.1263	0.09
Quadratic	1	300	0.005	0	FALSE	0.09905	0.1013	0.056
Quadratic	1	300	0.005	0.5	TRUE	0.09938	0.0946	0.12
Quadratic	1	300	0.005	0.5	FALSE	0.09905	0.09905	0.074
Quadratic	1	300	0.01	0	TRUE	0.08434	0.08403	0.091
Quadratic	1	300	0.01	0	FALSE	0.09905	0.09905	0.056
Quadratic	1	300	0.01	0.5	TRUE	0.09905	0.09905	0.056
Quadratic	1	300	0.01	0.5	FALSE	0.09905	0.09905	0.056
Quadratic	2	100	0.005	0	TRUE	0.2037	0.1942	0.056
Quadratic	2	100	0.005	0	FALSE	0.09905	0.09905	0.039
Quadratic	2	100	0.005	0.5	TRUE	0.3789	0.3484	0.074
Quadratic	2	100	0.005	0.5	FALSE	0.09905	0.09905	0.04
Quadratic	2	100	0.01	0	TRUE	0.5783	0.5659	0.056
Quadratic	2	100	0.01	0	FALSE	0.09905	0.09905	0.039
Quadratic	2	100	0.01	0.5	TRUE	0.2945	0.2844	0.074
Quadratic	2	100	0.01	0.5	FALSE	0.09905	0.09905	0.039
Quadratic	2	200	0.005	0	TRUE	0.006211	0.007791	0.11
Quadratic	2	200	0.005	0	FALSE	0.203	0.1981	0.057
Quadratic	2	200	0.005	0.5	TRUE	0.4904	0.468	0.14
Quadratic	2	200	0.005	0.5	FALSE	0.1978	0.1931	0.056
Quadratic	2	200	0.01	0	TRUE	0.003923	0.006121	0.11
Quadratic	2	200	0.01	0	FALSE	0.09905	0.09905	0.039
Quadratic	2	200	0.01	0.5	TRUE	0.1994	0.2031	0.14
Quadratic	2	200	0.01	0.5	FALSE	0.09905	0.09905	0.04
Quadratic	2	300	0.005	0	TRUE	0.09546	0.1714	0.16
Quadratic	2	300	0.005	0	FALSE	0.09905	0.09905	0.056
Quadratic	2	300	0.005	0.5	TRUE	0.5835	0.5609	0.21
Quadratic	2	300	0.005	0.5	FALSE	0.09905	0.09905	0.056
Quadratic	2	300	0.01	0	TRUE	0.1965	0.1976	0.12
Quadratic	2	300	0.01	0	FALSE	0.09905	0.09905	0.056
Quadratic	2	300	0.01	0.5	TRUE	0.09742	0.1046	0.16
Quadratic	2	300	0.01	0.5	FALSE	0.09905	0.09905	0.056

ErrorFunction	layers	hiddenUnits	learnRate	momentumStart	Scale	Acc_val	Acc_test	Time in min
Quadratic	3	100	0.005	0	TRUE	0.4521	0.4396	0.073
Quadratic	3	100	0.005	0	FALSE	0.09905	0.09905	0.039
Quadratic	3	100	0.005	0.5	TRUE	0.9742	0.9149	0.09
Quadratic	3	100	0.005	0.5	FALSE	0.09905	0.09905	0.039
Quadratic	3	100	0.01	0	TRUE	0.8696	0.7924	0.074
Quadratic	3	100	0.01	0	FALSE	0.09905	0.09905	0.039
Quadratic	3	100	0.01	0.5	TRUE	0.7028	0.6806	0.37
Quadratic	3	100	0.01	0.5	FALSE	0.8758	0.8347	0.04
Quadratic	3	200	0.005	0	TRUE	0.9676	0.8993	0.16
Quadratic	3	200	0.005	0	FALSE	0.8823	0.8358	0.055
Quadratic	3	200	0.005	0.5	TRUE	0.9657	0.9043	0.18
Quadratic	3	200	0.005	0.5	FALSE	0.09905	0.1013	0.057
Quadratic	3	200	0.01	0	TRUE	0.6653	0.6177	0.11
Quadratic	3	200	0.01	0	FALSE	0.09971	0.1013	0.039
Quadratic	3	200	0.01	0.5	TRUE	0.9807	0.9065	0.16
Quadratic	3	200	0.01	0.5	FALSE	0.09905	0.09905	0.056
Quadratic	3	300	0.005	0	TRUE	0.8993	0.8175	0.19
Quadratic	3	300	0.005	0	FALSE	0.09905	0.09905	0.057
Quadratic	3	300	0.005	0.5	TRUE	0.9837	0.9065	0.28
Quadratic	3	300	0.005	0.5	FALSE	0.6966	0.6644	0.091
Quadratic	3	300	0.01	0	TRUE	0.964	0.8965	0.19
Quadratic	3	300	0.01	0	FALSE	0.09905	0.09905	0.057
Quadratic	3	300	0.01	0.5	TRUE	0.9791	0.9093	0.28
Quadratic	3	300	0.01	0.5	FALSE	0.09905	0.09905	0.057
CrossEntropy	1	100	0.005	0	TRUE	0.09251	0.09238	0.056
CrossEntropy	1	100	0.005	0	FALSE	0.09905	0.09905	0.039
CrossEntropy	1	100	0.005	0.5	TRUE	0.1036	0.1018	0.056
CrossEntropy	1	100	0.005	0.5	FALSE	0.09905	0.09905	0.038
CrossEntropy	1	100	0.01	0	TRUE	0.09905	0.09905	0.04
CrossEntropy	1	100	0.01	0	FALSE	0.09905	0.09905	0.039
CrossEntropy	1	100	0.01	0.5	TRUE	0.09905	0.09905	0.04
CrossEntropy	1	100	0.01	0.5	FALSE	0.09905	0.09905	0.039
CrossEntropy	1	200	0.005	0	TRUE	0.09905	0.09905	0.056
CrossEntropy	1	200	0.005	0	FALSE	0.09905	0.09905	0.039
CrossEntropy	1	200	0.005	0.5	TRUE	0.1007	0.1024	0.091
CrossEntropy	1	200	0.005	0.5	FALSE	0.09905	0.09905	0.056
CrossEntropy	1	200	0.01	0	TRUE	0.09905	0.09905	0.056
CrossEntropy	1	200	0.01	0	FALSE	0.09905	0.09905	0.039
CrossEntropy	1	200	0.01	0.5	TRUE	0.287	0.2693	0.074
CrossEntropy	1	200	0.01	0.5	FALSE	0.09905	0.09905	0.039
CrossEntropy	1	300	0.005	0	TRUE	0.201	0.2037	0.12
CrossEntropy	1	300	0.005	0	FALSE	0.09905	0.09905	0.074
CrossEntropy	1	300	0.005	0.5	TRUE	0.003923	0.01336	0.14
CrossEntropy	1	300	0.005	0.5	FALSE	0.09905	0.09905	0.056
CrossEntropy	1	300	0.01	0	TRUE	0.09905	0.09905	0.072
CrossEntropy	1	300	0.01	0	FALSE	0.09905	0.09905	0.056
CrossEntropy	1	300	0.01	0.5	TRUE	0.09938	0.1002	0.11
CrossEntropy	1	300	0.01	0.5	FALSE	0.09905	0.09905	0.055
CrossEntropy	2	100	0.005	0	TRUE	0.2945	0.2844	0.055
CrossEntropy	2	100	0.005	0	FALSE	0.09905	0.09905	0.039
CrossEntropy	2	100	0.005	0.5	TRUE	0.1007	0.1068	0.073
CrossEntropy	2	100	0.005	0.5	FALSE	0.09905	0.09905	0.039

ErrorFunction	layers	hiddenUnits	learnRate	momentumStart	Scale	Acc_val	Acc_test	Time in min
CrossEntropy	2	100	0.01	0	TRUE	0.4887	0.4597	0.058
CrossEntropy	2	100	0.01	0	FALSE	0.09905	0.09905	0.038
CrossEntropy	2	100	0.01	0.5	TRUE	0.09873	0.1013	0.056
CrossEntropy	2	100	0.01	0.5	FALSE	0.09905	0.09905	0.04
CrossEntropy	2	200	0.005	0	TRUE	0.09905	0.09905	0.056
CrossEntropy	2	200	0.005	0	FALSE	0.09905	0.09905	0.057
CrossEntropy	2	200	0.005	0.5	TRUE	0.9647	0.8614	0.16
CrossEntropy	2	200	0.005	0.5	FALSE	0.09905	0.09905	0.039
CrossEntropy	2	200	0.01	0	TRUE	0.09905	0.09905	0.039
CrossEntropy	2	200	0.01	0	FALSE	0.09905	0.09905	0.04
CrossEntropy	2	200	0.01	0.5	TRUE	0.1	0.4808	0.09
CrossEntropy	2	200	0.01	0.5	FALSE	0.09905	0.09905	0.039
CrossEntropy	2	300	0.005	0	TRUE	0.4989	0.468	0.14
CrossEntropy	2	300	0.005	0	FALSE	0.09905	0.09905	0.057
CrossEntropy	2	300	0.005	0.5	TRUE	0.3099	0.2849	0.091
CrossEntropy	2	300	0.005	0.5	FALSE	0.09905	0.09905	0.057
CrossEntropy	2	300	0.01	0	TRUE	0.09905	0.09905	0.056
CrossEntropy	2	300	0.01	0	FALSE	0.09905	0.09905	0.057
CrossEntropy	2	300	0.01	0.5	TRUE	0.09905	0.09905	0.056
CrossEntropy	2	300	0.01	0.5	FALSE	0.09905	0.09905	0.056
CrossEntropy	3	100	0.005	0	TRUE	0.9784	0.8948	0.073
CrossEntropy	3	100	0.005	0	FALSE	0.09905	0.09905	0.04
CrossEntropy	3	100	0.005	0.5	TRUE	0.981	0.9243	0.092
CrossEntropy	3	100	0.005	0.5	FALSE	0.09905	0.09905	0.04
CrossEntropy	3	100	0.01	0	TRUE	0.6911	0.8386	0.057
CrossEntropy	3	100	0.01	0	FALSE	0.09905	0.09905	0.04
CrossEntropy	3	100	0.01	0.5	TRUE	0.4956	0.7396	0.073
CrossEntropy	3	100	0.01	0.5	FALSE	0.09905	0.09905	0.039
CrossEntropy	3	200	0.005	0	TRUE	0.5891	0.5364	0.13
CrossEntropy	3	200	0.005	0	FALSE	0.09905	0.09905	0.039
CrossEntropy	3	200	0.005	0.5	TRUE	0.001635	0.008347	0.12
CrossEntropy	3	200	0.005	0.5	FALSE	0.09905	0.09905	0.04
CrossEntropy	3	200	0.01	0	TRUE	0.5969	0.5815	0.11
CrossEntropy	3	200	0.01	0	FALSE	0.09905	0.09905	0.039
CrossEntropy	3	200	0.01	0.5	TRUE	0.8866	0.8197	0.11
CrossEntropy	3	200	0.01	0.5	FALSE	0.09905	0.09905	0.039
CrossEntropy	3	300	0.005	0	TRUE	0.4969	0.4702	0.16
CrossEntropy	3	300	0.005	0	FALSE	0.09905	0.09905	0.055
CrossEntropy	3	300	0.005	0.5	TRUE	0.9876	0.9282	0.23
CrossEntropy	3	300	0.005	0.5	FALSE	0.09905	0.09905	0.056
CrossEntropy	3	300	0.01	0	TRUE	0.9889	0.9388	0.18
CrossEntropy	3	300	0.01	0	FALSE	0.09905	0.09905	0.057
CrossEntropy	3	300	0.01	0.5	TRUE	0.09905	0.09905	0.057
CrossEntropy	3	300	0.01	0.5	FALSE	0.09905	0.09905	0.057

From Table 1 we can see that the model with highest accuraccy had following parameters: CrossEntropy, 3, 300, 0.01, 0, TRUE, 0.9889, 0.9388, 0.18.

Experiment 1a. Best model confusion matrix and model summary

```
bestModel
```

```
## Model Details:
## ==============
##
## H2OMultinomialModel: deeplearning
## Model ID:  DeepLearning_model_R_1520628378438_141
## Status of Neuron Layers: predicting V65, 10-class classification, multinomial distribution, CrossEnt
##   layer units      type dropout       l1       l2 mean_rate rate_rms
## 1     1    61     Input  0.00 %
## 2     2   300 Rectifier  0.00 % 0.000000 0.000000  0.006855 0.000000
## 3     3     3 Rectifier  0.00 % 0.000000 0.000000  0.006855 0.000000
## 4     4    10   Softmax         0.000000 0.000000  0.006855 0.000000
##   momentum mean_weight weight_rms mean_bias bias_rms
## 1
## 2 0.000000   -0.001117   0.167060 -0.036282 0.326305
## 3 0.000000    0.018890   0.295198  1.432974 0.212181
## 4 0.000000    0.365853   1.715177 -0.000412 4.948715
##
##
## H2OMultinomialMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on full training frame **
##
## Training Set Metrics:
## =====================
##
## Extract training frame with `h2o.getFrame("c.train")`
## MSE: (Extract with `h2o.mse`) 0.002665826
## RMSE: (Extract with `h2o.rmse`) 0.05163164
## Logloss: (Extract with `h2o.logloss`) 0.01419965
## Mean Per-Class Error: 0.002954603
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,train = TRUE)`)
## =========================================================================
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##          0   1   2   3   4   5   6   7   8   9  Error        Rate
## 0      293   0   0   0   0   0   0   0   0   0 0.0000 =   0 / 293
## 1        1 312   0   0   0   0   0   0   0   0 0.0032 =   1 / 313
## 2        0   0 306   0   0   0   0   0   0   0 0.0000 =   0 / 306
## 3        0   0   0 307   0   0   0   0   0   0 0.0000 =   0 / 307
## 4        0   0   0   0 306   0   5   0   0   0 0.0161 =   5 / 311
## 5        0   0   0   0   0 311   0   0   0   0 0.0000 =   0 / 311
## 6        0   0   0   0   0   0 306   0   0   0 0.0000 =   0 / 306
## 7        0   0   0   0   0   0   0 314   0   0 0.0000 =   0 / 314
## 8        0   0   0   1   0   1   0   1 289   0 0.0103 =   3 / 292
## 9        0   0   0   0   0   0   0   0   0 305 0.0000 =   0 / 305
## Totals 294 312 306 308 306 312 311 315 289 305 0.0029 = 9 / 3,058
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,train = TRUE)`
## =======================================================================
## Top-10 Hit Ratios:
##     k hit_ratio
```

```
## 1   1  0.997057
## 2   2  0.999346
## 3   3  0.999346
## 4   4  0.999673
## 5   5  0.999673
## 6   6  0.999673
## 7   7  0.999673
## 8   8  1.000000
## 9   9  1.000000
## 10 10  1.000000
##
##
## H2OMultinomialMetrics: deeplearning
## ** Reported on validation data. **
## ** Metrics reported on full validation frame **
##
## Validation Set Metrics:
## =====================
##
## Extract validation frame with `h2o.getFrame("c.validation")`
## MSE: (Extract with `h2o.mse`) 0.009576173
## RMSE: (Extract with `h2o.rmse`) 0.09785792
## Logloss: (Extract with `h2o.logloss`) 0.07998924
## Mean Per-Class Error: 0.01101842
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,valid = TRUE)`)
## =========================================================================
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##          0   1   2   3   4   5   6   7   8   9  Error         Rate
## 0      303   0   0   0   0   0   0   0   0   0 0.0000 =    0 / 303
## 1        1 302   0   0   0   1   0   0   0   1 0.0098 =    3 / 305
## 2        0   0 302   0   0   2   0   1   0   0 0.0098 =    3 / 305
## 3        0   0   0 313   0   2   1   0   0   0 0.0095 =    3 / 316
## 4        0   0   0   0 302   0   7   0   0   1 0.0258 =    8 / 310
## 5        0   0   1   0   0 300   0   1   0   0 0.0066 =    2 / 302
## 6        0   0   0   1   0   0 298   0   0   0 0.0033 =    1 / 299
## 7        1   0   0   1   0   0   0 301   0   1 0.0099 =    3 / 304
## 8        0   3   2   1   0   1   0   0 304   2 0.0288 =    9 / 313
## 9        0   0   0   1   0   0   0   0   1 300 0.0066 =    2 / 302
## Totals 305 305 305 317 302 306 306 303 305 305 0.0111 = 34 / 3,059
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,valid = TRUE)`
## =======================================================================
## Top-10 Hit Ratios:
##     k hit_ratio
## 1   1  0.988885
## 2   2  0.995423
## 3   3  0.996731
## 4   4  0.997712
## 5   5  0.998366
## 6   6  0.999346
## 7   7  0.999346
## 8   8  0.999346
## 9   9  0.999673
## 10 10  1.000000
```

Experiment 1a. Best model confusion matrix on test set

```r
h2o.confusionMatrix(bestModel, test.h2o)
```

```
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##          0   1   2   3   4   5   6   7   8   9  Error          Rate
## 0      176   0   0   0   0   0   0   2   0   0 0.0112 =     2 / 178
## 1        1 174   4   0   0   1   0   0   0   2 0.0440 =     8 / 182
## 2        0   0 164   1   0   6   1   4   1   0 0.0734 =    13 / 177
## 3        0   0   2 171   0   5   2   0   1   2 0.0656 =    12 / 183
## 4        1   1   0   0 172   0   1   0   0   6 0.0497 =     9 / 181
## 5        0   0   0   4   0 172   0   2   0   4 0.0549 =    10 / 182
## 6        0   0   0   1   2   0 175   0   0   3 0.0331 =     6 / 181
## 7        0   0   0   1   0   7   4 160   1   6 0.1061 =    19 / 179
## 8        0   3   6   1   0   6   0   0 154   4 0.1149 =    20 / 174
## 9        0   0   0   3   2   2   0   0   4 169 0.0611 =    11 / 180
## Totals 178 178 176 182 176 199 183 168 161 196 0.0612 = 110 / 1,797
```

Experiment 1a. Plots showing vaiability of test accuraccy of best model with respect to hyperparameters

```r
# for plotting
resultsTab$ErrorFunction <- factor(resultsTab$ErrorFunction, levels = errorFunc)
par(mfrow = c(2, 2))
boxplot(as.numeric(resultsTab$Acc_test) ~ as.numeric(resultsTab$layers), data = resultsTab,
    main = "Acc vs Layers", ylab = "Accuraccy", xlab = "Num layers")
boxplot(as.numeric(resultsTab$Acc_test) ~ as.numeric(resultsTab$hiddenUnits),
    data = resultsTab, main = "Acc vs Units", xlab = "Hidden units", ylab = "Accuraccy")
boxplot(as.numeric(resultsTab$Acc_test) ~ as.numeric(resultsTab$ErrorFunction),
    data = resultsTab, main = "Acc vs Error func", xlab = "Error fun (1.SumofSq, 2.CrossEntropy)",
    ylab = "Accuraccy")
boxplot(as.numeric(resultsTab$Acc_test) ~ as.numeric(resultsTab$learnRate),
    data = resultsTab, main = "Acc vs Rate", ylab = "Accuraccy", xlab = "Learning rate")
```

```r
par(mfrow = c(1, 1))
```

Experiment 1b

For Experiment 1b I chose the error function to be cross entropy and I changed other parameters. I iterated over combinations of following parameters and buit models and tested them.

Parameters

- Error Function: CrossEntropy
- Activation Function: TanH, ReLU
- Hidden Layers: 1, 2, 3
- Hidden Units: 100, 200, 300
- Learning Rate: 0.005, 0.01
- Momentum Start: 0, 0.5
- Input Scaling: True, False

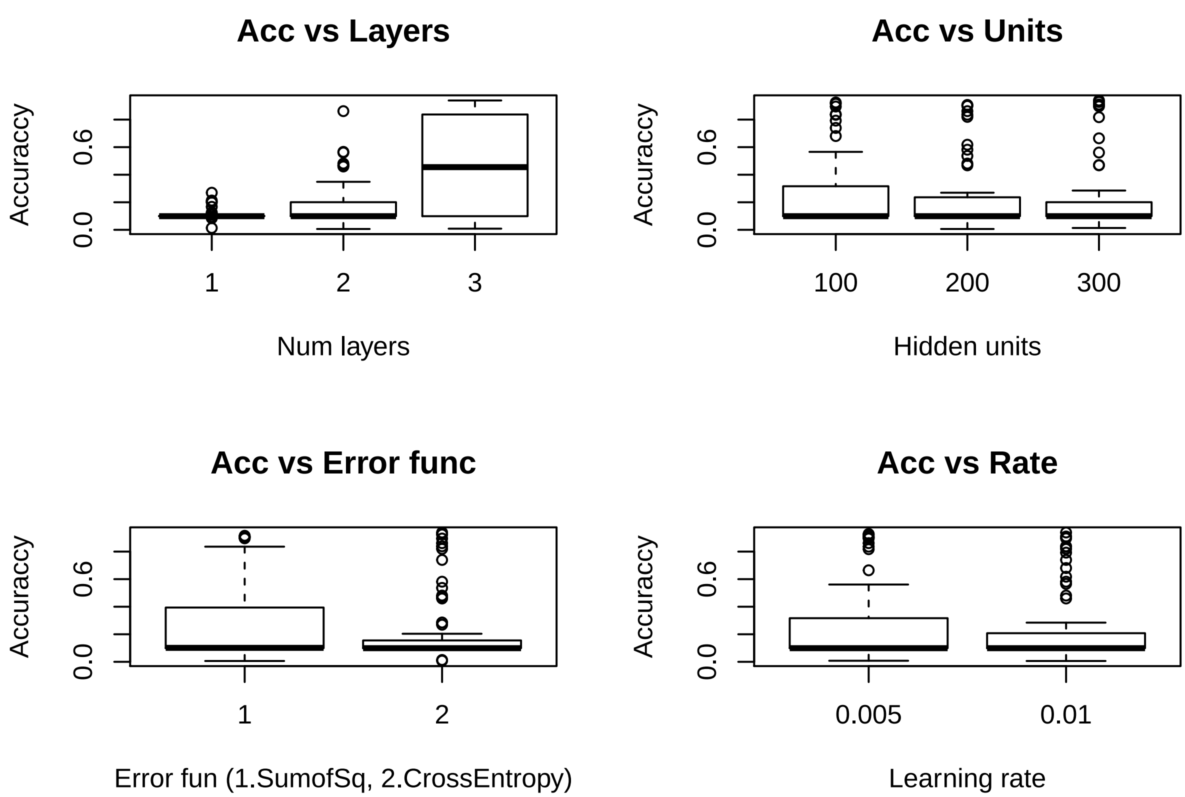

Figure 1: Experiment 1a. Plots showing vaiability of test accuraccy of best model with respect to hyperparameters

I wrote the attached R code for simulation (using h2o). The results are described in Table 2.

Experiment 1b R code

```
## ALL data is already loaded

###### Set model hyper-parameters
hiddenLayers <- c(1, 2, 3)
hiddenUnits <- c(100, 200, 300)
learningRates <- c(0.005, 0.01)
momentumStart <- c(0, 0.5)
inputScaling <- c(T, F)
errorFunc <- c("CrossEntropy")
act <- c("Rectifier", "Tanh")
# hiddenLayers<-c(1) hiddenUnits<-c(100,200) learningRates<-c(0.01)
# momentumStart<-c(0) inputScaling<-c(T) errorFunc<-c('Quadratic') Table to
# write results
header2 <- c("activation", "layers", "hiddenUnits", "learnRate", "momentumStart",
    "Scale", "Acc_val", "Acc_test", "Time in min")
resultsTab2 <- data.frame(matrix(ncol = 9, nrow = 0))
colnames(resultsTab2) <- header2
# set seed for reproducible results
set.seed(1654)
# save the best model i.e. highest accuraccy on test set
bestModelb <- NULL
bestAcc <- 0
# make all comniations of the parameters
for (a in act) {
    for (hL in hiddenLayers) {
        for (hU in hiddenUnits) {
            for (lR in learningRates) {
                for (mS in momentumStart) {
                  for (iS in inputScaling) {

                    # build model and calculate accuraccy
                    s <- proc.time()  #start time
                    model1 <- h2o.deeplearning(x = predictors, y = y.dep, training_frame = train.h2o,
                      validation_frame = validation.h2o, hidden = c(rep(hU),
                        hL), activation = a, epochs = 150, loss = errorFunc,
                      rate = lR, momentum_start = mS, standardize = iS, adaptive_rate = F)
                    d <- proc.time() - s  #end time

                    # test on testdata cat('Performance on test data:')
                    # perf<-h2o.performance(model1,test.h2o) perf compute accuraccy on
                    # validation
                    valResult <- h2o.predict(model1, validation.h2o, y = y.dep)
                    predictions <- as.data.frame(valResult[, 1])
                    trueLabels <- c.validation$V65
                    correct <- 0
                    for (i in 1:dim(predictions)[1]) {
                      if (as.numeric(predictions$predict[i]) == as.numeric(c.validation$V65[i])) {
                        correct <- correct + 1
                      }
```

```
                    }
                    acc_V <- format(correct/dim(predictions)[1], digits = 4)
                    # cat('Accuraccy on validation set:',acc_V) compute accuraccy on test
                    testResult <- h2o.predict(model1, test.h2o, y = y.dep)
                    predictions <- as.data.frame(testResult[, 1])
                    trueLabels <- test$V65
                    correct <- 0
                    for (i in 1:dim(predictions)[1]) {
                      if (as.numeric(predictions$predict[i]) == as.numeric(test$V65[i])) {
                        correct <- correct + 1
                      }
                    }
                    acc <- format(correct/dim(predictions)[1], digits = 4)
                    # cat('Accuraccy on test set:',acc)
                    resultsTab2[nrow(resultsTab2) + 1, ] <- c(a, hL, hU, lR,
                      mS, iS, acc_V, acc, format(as.numeric(d)[3]/60, digits = 2))

                    if (acc > bestAcc) {
                      bestModelb <- model1
                      bestAcc <- acc
                    }
                  }
                }
            }
        }
    }
}
resultsTab2 %>% knitr::kable(caption = "Experiment 1b outcomes. Error function was cross-entropy.")
```

Table 2: Experiment 1b outcomes. Error function was cross-entropy.

activation	layers	hiddenUnits	learnRate	momentumStart	Scale	Acc_val	Acc_test	Time in min
Rectifier	1	100	0.005	0	TRUE	0.004904	0.01391	0.039
Rectifier	1	100	0.005	0	FALSE	0.09905	0.09905	0.039
Rectifier	1	100	0.005	0.5	TRUE	0.0984	0.09905	0.057
Rectifier	1	100	0.005	0.5	FALSE	0.09905	0.09905	0.039
Rectifier	1	100	0.01	0	TRUE	0.09905	0.09905	0.04
Rectifier	1	100	0.01	0	FALSE	0.09905	0.09905	0.039
Rectifier	1	100	0.01	0.5	TRUE	0.09971	0.09794	0.04
Rectifier	1	100	0.01	0.5	FALSE	0.09905	0.09905	0.039
Rectifier	1	200	0.005	0	TRUE	0.201	0.1976	0.056
Rectifier	1	200	0.005	0	FALSE	0.09905	0.09905	0.038
Rectifier	1	200	0.005	0.5	TRUE	0.01177	0.02115	0.13
Rectifier	1	200	0.005	0.5	FALSE	0.09905	0.09905	0.039
Rectifier	1	200	0.01	0	TRUE	0.09938	0.0985	0.073
Rectifier	1	200	0.01	0	FALSE	0.09905	0.09905	0.039
Rectifier	1	200	0.01	0.5	TRUE	0.09905	0.09905	0.04
Rectifier	1	200	0.01	0.5	FALSE	0.09905	0.09905	0.039
Rectifier	1	300	0.005	0	TRUE	0.09905	0.09905	0.096
Rectifier	1	300	0.005	0	FALSE	0.09905	0.09905	0.056
Rectifier	1	300	0.005	0.5	TRUE	0.09905	0.09905	0.056
Rectifier	1	300	0.005	0.5	FALSE	0.09905	0.09905	0.058
Rectifier	1	300	0.01	0	TRUE	0.09905	0.09905	0.059

activation	layers	hiddenUnits	learnRate	momentumStart	Scale	Acc_val	Acc_test	Time in min
Rectifier	1	300	0.01	0	FALSE	0.09905	0.09905	0.057
Rectifier	1	300	0.01	0.5	TRUE	0.09905	0.09905	0.056
Rectifier	1	300	0.01	0.5	FALSE	0.09905	0.09905	0.057
Rectifier	2	100	0.005	0	TRUE	0.8751	0.7963	0.057
Rectifier	2	100	0.005	0	FALSE	0.09905	0.09905	0.04
Rectifier	2	100	0.005	0.5	TRUE	0.4966	0.4697	0.074
Rectifier	2	100	0.005	0.5	FALSE	0.09905	0.09905	0.04
Rectifier	2	100	0.01	0	TRUE	0.09905	0.09905	0.04
Rectifier	2	100	0.01	0	FALSE	0.09905	0.09905	0.039
Rectifier	2	100	0.01	0.5	TRUE	0.001635	0.09293	0.057
Rectifier	2	100	0.01	0.5	FALSE	0.09905	0.09905	0.039
Rectifier	2	200	0.005	0	TRUE	0.09807	0.09738	0.075
Rectifier	2	200	0.005	0	FALSE	0.09905	0.09905	0.038
Rectifier	2	200	0.005	0.5	TRUE	0.3969	0.3801	0.14
Rectifier	2	200	0.005	0.5	FALSE	0.09905	0.09905	0.04
Rectifier	2	200	0.01	0	TRUE	0.4936	0.4736	0.091
Rectifier	2	200	0.01	0	FALSE	0.09905	0.09905	0.04
Rectifier	2	200	0.01	0.5	TRUE	0.09905	0.09905	0.039
Rectifier	2	200	0.01	0.5	FALSE	0.09905	0.09905	0.04
Rectifier	2	300	0.005	0	TRUE	0.001961	0.01057	0.12
Rectifier	2	300	0.005	0	FALSE	0.09905	0.09905	0.057
Rectifier	2	300	0.005	0.5	TRUE	0.09905	0.09905	0.057
Rectifier	2	300	0.005	0.5	FALSE	0.09905	0.09905	0.057
Rectifier	2	300	0.01	0	TRUE	0.1909	0.1764	0.12
Rectifier	2	300	0.01	0	FALSE	0.09905	0.09905	0.057
Rectifier	2	300	0.01	0.5	TRUE	0.1033	0.1029	0.09
Rectifier	2	300	0.01	0.5	FALSE	0.09905	0.09905	0.057
Rectifier	3	100	0.005	0	TRUE	0.9823	0.9132	0.073
Rectifier	3	100	0.005	0	FALSE	0.09905	0.09905	0.039
Rectifier	3	100	0.005	0.5	TRUE	0.001308	0.01614	0.075
Rectifier	3	100	0.005	0.5	FALSE	0.09905	0.09905	0.039
Rectifier	3	100	0.01	0	TRUE	0.09905	0.09905	0.04
Rectifier	3	100	0.01	0	FALSE	0.09905	0.09905	0.04
Rectifier	3	100	0.01	0.5	TRUE	0.8862	0.8264	0.074
Rectifier	3	100	0.01	0.5	FALSE	0.09905	0.09905	0.04
Rectifier	3	200	0.005	0	TRUE	0.9873	0.9327	0.12
Rectifier	3	200	0.005	0	FALSE	0.09905	0.09905	0.04
Rectifier	3	200	0.005	0.5	TRUE	0.9882	0.916	0.16
Rectifier	3	200	0.005	0.5	FALSE	0.09905	0.09905	0.039
Rectifier	3	200	0.01	0	TRUE	0.1919	0.1848	0.091
Rectifier	3	200	0.01	0	FALSE	0.09905	0.09905	0.039
Rectifier	3	200	0.01	0.5	TRUE	0.49	0.4708	0.13
Rectifier	3	200	0.01	0.5	FALSE	0.09905	0.09905	0.039
Rectifier	3	300	0.005	0	TRUE	0.8807	0.8125	0.16
Rectifier	3	300	0.005	0	FALSE	0.09905	0.09905	0.059
Rectifier	3	300	0.005	0.5	TRUE	0.9572	0.8815	0.14
Rectifier	3	300	0.005	0.5	FALSE	0.09905	0.09905	0.056
Rectifier	3	300	0.01	0	TRUE	0.09905	0.09905	0.056
Rectifier	3	300	0.01	0	FALSE	0.09905	0.09905	0.056
Rectifier	3	300	0.01	0.5	TRUE	0.09905	0.09905	0.056
Rectifier	3	300	0.01	0.5	FALSE	0.09905	0.09905	0.057
Tanh	1	100	0.005	0	TRUE	0.2148	0.1981	0.14

activation	layers	hiddenUnits	learnRate	momentumStart	Scale	Acc_val	Acc_test	Time in min
Tanh	1	100	0.005	0	FALSE	0.2043	0.2137	0.058
Tanh	1	100	0.005	0.5	TRUE	0.4309	0.3461	0.18
Tanh	1	100	0.005	0.5	FALSE	0.1007	0.09794	0.057
Tanh	1	100	0.01	0	TRUE	0.03204	0.03506	0.14
Tanh	1	100	0.01	0	FALSE	0.2743	0.2721	0.056
Tanh	1	100	0.01	0.5	TRUE	0.1912	0.1981	0.18
Tanh	1	100	0.01	0.5	FALSE	0.1465	0.1452	0.056
Tanh	1	200	0.005	0	TRUE	0.1533	0.1519	0.23
Tanh	1	200	0.005	0	FALSE	0.118	0.1269	0.11
Tanh	1	200	0.005	0.5	TRUE	0.2635	0.2359	0.3
Tanh	1	200	0.005	0.5	FALSE	0.3011	0.2938	0.13
Tanh	1	200	0.01	0	TRUE	0.09774	0.0985	0.23
Tanh	1	200	0.01	0	FALSE	0.08859	0.09794	0.11
Tanh	1	200	0.01	0.5	TRUE	0.1007	0.1002	0.3
Tanh	1	200	0.01	0.5	FALSE	0.004904	0.007234	0.11
Tanh	1	300	0.005	0	TRUE	0.3576	0.3344	0.33
Tanh	1	300	0.005	0	FALSE	0.1948	0.2009	0.16
Tanh	1	300	0.005	0.5	TRUE	0.0003269	0.005565	0.44
Tanh	1	300	0.005	0.5	FALSE	0.09971	0.1029	0.26
Tanh	1	300	0.01	0	TRUE	0.167	0.1658	0.33
Tanh	1	300	0.01	0	FALSE	0.007192	0.008347	0.14
Tanh	1	300	0.01	0.5	TRUE	0.05917	0.07179	0.42
Tanh	1	300	0.01	0.5	FALSE	0.0009807	0.001669	0.16
Tanh	2	100	0.005	0	TRUE	0.9689	0.8559	0.14
Tanh	2	100	0.005	0	FALSE	0.0134	0.02838	0.056
Tanh	2	100	0.005	0.5	TRUE	0.9568	0.8253	0.16
Tanh	2	100	0.005	0.5	FALSE	0.5528	0.5292	0.074
Tanh	2	100	0.01	0	TRUE	0.9683	0.8559	0.13
Tanh	2	100	0.01	0	FALSE	0.2409	0.1119	0.057
Tanh	2	100	0.01	0.5	TRUE	0.879	0.7952	0.16
Tanh	2	100	0.01	0.5	FALSE	0.2798	0.2682	0.073
Tanh	2	200	0.005	0	TRUE	0.9745	0.8614	0.25
Tanh	2	200	0.005	0	FALSE	0.4534	0.4396	0.11
Tanh	2	200	0.005	0.5	TRUE	0.9693	0.8742	0.3
Tanh	2	200	0.005	0.5	FALSE	0.3488	0.345	0.13
Tanh	2	200	0.01	0	TRUE	0.9689	0.8798	0.23
Tanh	2	200	0.01	0	FALSE	0.2913	0.2832	0.11
Tanh	2	200	0.01	0.5	TRUE	0.9467	0.8275	0.3
Tanh	2	200	0.01	0.5	FALSE	0.219	0.2938	0.11
Tanh	2	300	0.005	0	TRUE	0.9755	0.8709	0.33
Tanh	2	300	0.005	0	FALSE	0.1471	0.1475	0.16
Tanh	2	300	0.005	0.5	TRUE	0.9742	0.8765	0.42
Tanh	2	300	0.005	0.5	FALSE	0.003923	0.005008	0.18
Tanh	2	300	0.01	0	TRUE	0.9474	0.8408	0.33
Tanh	2	300	0.01	0	FALSE	0.2118	0.2254	0.16
Tanh	2	300	0.01	0.5	TRUE	0.6221	0.6572	0.41
Tanh	2	300	0.01	0.5	FALSE	0.09905	0.09905	0.3
Tanh	3	100	0.005	0	TRUE	0.9846	0.9032	0.14
Tanh	3	100	0.005	0	FALSE	0.8562	0.8002	0.057
Tanh	3	100	0.005	0.5	TRUE	0.9833	0.8976	0.18
Tanh	3	100	0.005	0.5	FALSE	0.9369	0.8848	0.074
Tanh	3	100	0.01	0	TRUE	0.983	0.9137	0.14

activation	layers	hiddenUnits	learnRate	momentumStart	Scale	Acc_val	Acc_test	Time in min
Tanh	3	100	0.01	0	FALSE	0.6411	0.6439	0.056
Tanh	3	100	0.01	0.5	TRUE	0.9889	0.9149	0.18
Tanh	3	100	0.01	0.5	FALSE	0.252	0.2482	0.074
Tanh	3	200	0.005	0	TRUE	0.9905	0.9165	0.21
Tanh	3	200	0.005	0	FALSE	0.7702	0.754	0.11
Tanh	3	200	0.005	0.5	TRUE	0.9915	0.9282	0.12
Tanh	3	200	0.005	0.5	FALSE	0.8424	0.7974	0.12
Tanh	3	200	0.01	0	TRUE	0.7934	0.7351	0.23
Tanh	3	200	0.01	0	FALSE	0.5198	0.4864	0.14
Tanh	3	200	0.01	0.5	TRUE	0.4982	0.4836	0.31
Tanh	3	200	0.01	0.5	FALSE	0.5436	0.4485	0.12
Tanh	3	300	0.005	0	TRUE	0.9876	0.9137	0.33
Tanh	3	300	0.005	0	FALSE	0.815	0.7952	0.16
Tanh	3	300	0.005	0.5	TRUE	0.9876	0.9165	0.43
Tanh	3	300	0.005	0.5	FALSE	0.7555	0.7129	0.18
Tanh	3	300	0.01	0	TRUE	0.4959	0.4741	0.35
Tanh	3	300	0.01	0	FALSE	0.09905	0.09905	0.26
Tanh	3	300	0.01	0.5	TRUE	0.1981	0.1976	0.43
Tanh	3	300	0.01	0.5	FALSE	0.04021	0.04174	0.19

From Table 2 we can see that the model with highest accuraccy in experiment 1b had following parameters: Rectifier, 3, 200, 0.005, 0, TRUE, 0.9873, 0.9327, 0.12.

Experiment 1b. Best model confusion matrix and model summary

```
bestModelb
```

```
## Model Details:
## ==============
##
## H2OMultinomialModel: deeplearning
## Model ID:  DeepLearning_model_R_1520628378438_201
## Status of Neuron Layers: predicting V65, 10-class classification, multinomial distribution, CrossEnt
##   layer units      type dropout       l1       l2 mean_rate rate_rms
## 1     1    61     Input  0.00 %
## 2     2   200 Rectifier  0.00 % 0.000000 0.000000  0.003428 0.000000
## 3     3     3 Rectifier  0.00 % 0.000000 0.000000  0.003428 0.000000
## 4     4    10   Softmax         0.000000 0.000000  0.003428 0.000000
##   momentum mean_weight weight_rms mean_bias bias_rms
## 1
## 2 0.000000    0.000410   0.151602  0.286098 0.210754
## 3 0.000000   -0.007039   0.370658  1.098224 0.181903
## 4 0.000000    0.338403   2.005357 -0.001935 6.168470
##
##
## H2OMultinomialMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on full training frame **
##
## Training Set Metrics:
## =====================
```

```
##
## Extract training frame with `h2o.getFrame("c.train")`
## MSE: (Extract with `h2o.mse`) 0.002949724
## RMSE: (Extract with `h2o.rmse`) 0.05431136
## Logloss: (Extract with `h2o.logloss`) 0.02978086
## Mean Per-Class Error: 0.001993095
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,train = TRUE)`)
## =========================================================================
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##          0   1   2   3   4   5   6   7   8   9  Error        Rate
## 0      292   0   0   0   0   0   1   0   0   0 0.0034 =   1 / 293
## 1        0 313   0   0   0   0   0   0   0   0 0.0000 =   0 / 313
## 2        0   0 304   0   0   2   0   0   0   0 0.0065 =   2 / 306
## 3        0   0   0 307   0   0   0   0   0   0 0.0000 =   0 / 307
## 4        0   0   0   0 311   0   0   0   0   0 0.0000 =   0 / 311
## 5        0   0   0   0   0 311   0   0   0   0 0.0000 =   0 / 311
## 6        0   0   0   0   0   0 306   0   0   0 0.0000 =   0 / 306
## 7        0   0   0   0   0   0   0 314   0   0 0.0000 =   0 / 314
## 8        0   0   1   0   0   0   0   0 291   0 0.0034 =   1 / 292
## 9        0   0   0   1   0   0   1   0   0 303 0.0066 =   2 / 305
## Totals 292 313 305 308 311 313 308 314 291 303 0.0020 = 6 / 3,058
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,train = TRUE)`
## =======================================================================
## Top-10 Hit Ratios:
##     k hit_ratio
## 1   1  0.998038
## 2   2  0.998692
## 3   3  0.999346
## 4   4  0.999673
## 5   5  0.999673
## 6   6  0.999673
## 7   7  0.999673
## 8   8  1.000000
## 9   9  1.000000
## 10 10  1.000000
##
##
## H2OMultinomialMetrics: deeplearning
## ** Reported on validation data. **
## ** Metrics reported on full validation frame **
##
## Validation Set Metrics:
## =====================
##
## Extract validation frame with `h2o.getFrame("c.validation")`
## MSE: (Extract with `h2o.mse`) 0.01321373
## RMSE: (Extract with `h2o.rmse`) 0.114951
## Logloss: (Extract with `h2o.logloss`) 0.09441424
## Mean Per-Class Error: 0.01275885
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,valid = TRUE)`)
## =========================================================================
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##          0   1   2   3   4   5   6   7   8   9  Error         Rate
```

```
## 0      297   0   0   0   0   4   1   0   0   1 0.0198 =    6 / 303
## 1        0 302   0   0   0   0   0   0   3   0 0.0098 =    3 / 305
## 2        1   0 300   0   0   4   0   0   0   0 0.0164 =    5 / 305
## 3        0   0   0 312   0   3   0   0   1   0 0.0127 =    4 / 316
## 4        0   0   0   0 307   0   2   0   0   1 0.0097 =    3 / 310
## 5        0   0   0   3   0 299   0   0   0   0 0.0099 =    3 / 302
## 6        0   0   0   0   0   0 298   0   0   1 0.0033 =    1 / 299
## 7        0   0   0   0   0   0   2 301   1   0 0.0099 =    3 / 304
## 8        0   2   0   1   0   0   0   0 310   0 0.0096 =    3 / 313
## 9        0   0   0   6   0   0   1   0   1 294 0.0265 =    8 / 302
## Totals 298 304 300 322 307 310 304 301 316 297 0.0127 = 39 / 3,059
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,valid = TRUE)`
## =======================================================================
## Top-10 Hit Ratios:
##     k hit_ratio
## 1   1  0.987251
## 2   2  0.993462
## 3   3  0.998039
## 4   4  0.999346
## 5   5  1.000000
## 6   6  1.000000
## 7   7  1.000000
## 8   8  1.000000
## 9   9  1.000000
## 10 10  1.000000
```

Experiment 1b. Best model confusion matrix on test set

```r
h2o.confusionMatrix(bestModelb, test.h2o)
```

```
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##          0   1   2   3   4   5   6   7   8   9  Error           Rate
## 0      168   0   0   0   0   8   2   0   0   0 0.0562 =    10 / 178
## 1        0 175   0   0   0   0   0   0   3   4 0.0385 =     7 / 182
## 2        1   1 161   2   0   5   1   1   3   2 0.0904 =    16 / 177
## 3        1   0   0 168   0   7   0   1   2   4 0.0820 =    15 / 183
## 4        0   2   0   0 174   0   0   1   0   4 0.0387 =     7 / 181
## 5        1   0   2   2   0 170   4   0   0   3 0.0659 =    12 / 182
## 6        3   0   0   0   1   0 177   0   0   0 0.0221 =     4 / 181
## 7        2   0   1   0   0   1   5 163   5   2 0.0894 =    16 / 179
## 8        0   4   0   9   0   0   1   0 159   1 0.0862 =    15 / 174
## 9        0   2   0   6   5   0   4   0   2 161 0.1056 =    19 / 180
## Totals 176 184 164 187 180 191 194 166 174 181 0.0673 = 121 / 1,797
```

Experiment 1b. Plots showing vaiability of test accuraccy of best model with respect to hyperparameters

```r
# for plotting
resultsTab2$activation <- factor(resultsTab2$activation, levels = act)
par(mfrow = c(2, 2))
boxplot(as.numeric(resultsTab2$Acc_test) ~ as.numeric(resultsTab2$layers), data = resultsTab2,
```

```r
    main = "Acc vs Layers", ylab = "Accuraccy", xlab = "Num layers")
boxplot(as.numeric(resultsTab2$Acc_test) ~ as.numeric(resultsTab2$hiddenUnits),
    data = resultsTab2, main = "Acc vs Units", xlab = "Hidden units", ylab = "Accuraccy")
boxplot(as.numeric(resultsTab2$Acc_test) ~ as.numeric(resultsTab2$activation),
    data = resultsTab2, main = "Acc vs Activation", xlab = "Activation fun (1.ReLU, 2.tanh)",
    ylab = "Accuraccy")
boxplot(as.numeric(resultsTab2$Acc_test) ~ as.numeric(resultsTab2$learnRate),
    data = resultsTab2, main = "Acc vs Rate", ylab = "Accuraccy", xlab = "Learning rate")
```

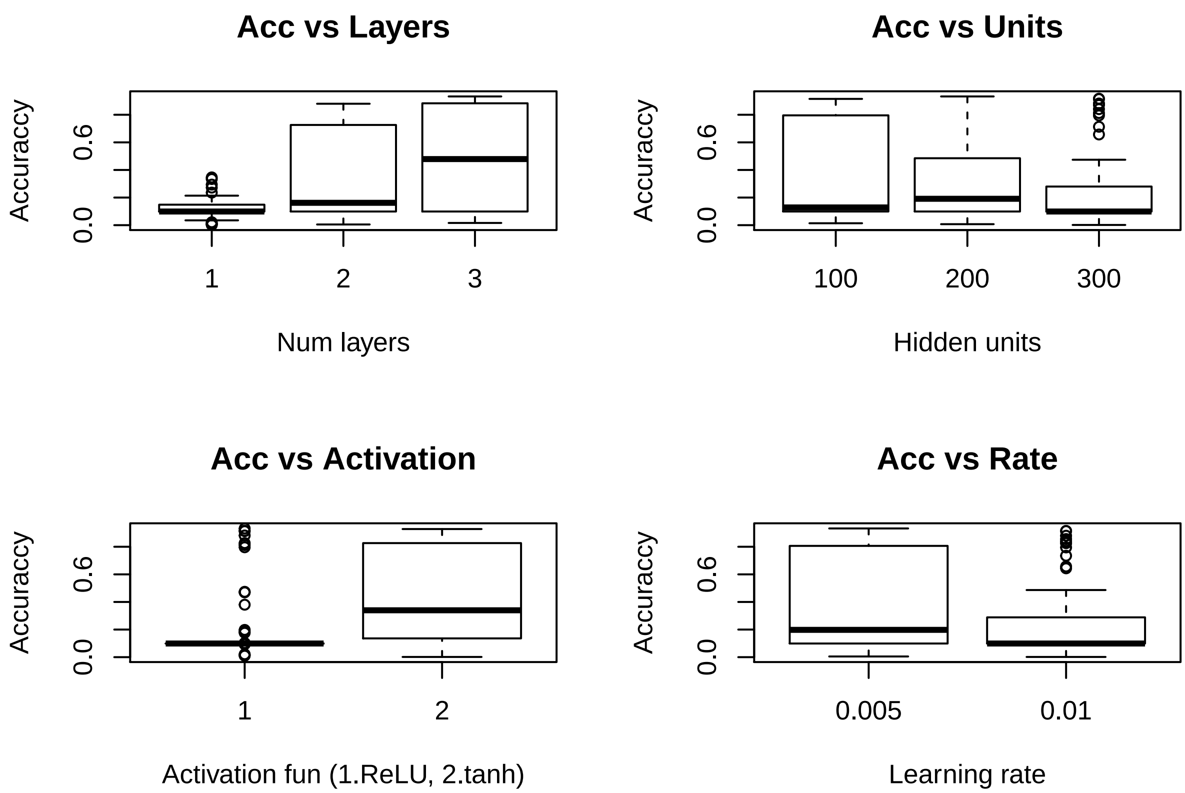

Figure 2: Experiment 1b. Plots showing vaiability of test accuraccy of best model with respect to hyperparameters

```r
par(mfrow = c(1, 1))
```

```r
# shutdown h2o
h2o.shutdown(prompt = FALSE)
```

```
## [1] TRUE
```

Experiment 1: Discussion

Experiment 1a

In Experiment 1a I fixed the activation function for hidden layers to be ReLU and I built models by iterating over hyperparameter space which I generated arbitarily. I found that the best model has following parameters:

- Error function: Quadratic

- Hidden layers: 3
- Hidden units: 200
- learning rate: 0.005
- momentum start: 0
- Input Scaling: TRUE
- Accuracy (validation): 0.9676
- Accuraccy (test): 0.8993

I expected the model to have maximum number of hidden units and layers but this is not always the result I found. Infact the model with 3 hidden layers with 300 units and other hyperparameters same as the best model gave an accuraccy of only 45% on the test set.

In Fig 1 we can see how the hyperparameters i.e. hidden layers, hidden units, error function and learning rate contributes to accuraccy. These plots show overall variability of test set accuraccy over these hyperparameters. We see that hidden layers 2 and 3 have can cause higher variability in accuraccy. When hidden units are lesser variation is high although the range of accuraccy over different number of units looks same. Clearly, the median values of accuracy over cross-entropy and sum of squares looks same but cross entropy has higher max value. With learning rate lower i.e. 0.005 we see that accuracy has higher maximum value as compared to learning rate 0.01.

Experiment 1b

In Experiment 1b I fixed the error function to be cross entropy and I built models by iterating over the hyperparameter space which I generated arbitarily. I found that the best model has following parameters: * Error function: CrossEntropy * Error function: Rectifier * Hidden layers: 3 * Hidden units: 200 * learning rate: 0.005 * momentum start: 0 * Input Scaling: TRUE * Accuracy (validation): 0.9873 * Accuraccy (test): 0.9327

In Fig 2 we can see how the hyperparameters i.e. hidden layers, hidden units, activation function and learning rate contributes to accuraccy. Just as in Fig1, these plots show overall variability of test set accuraccy over these hyperparameters. We see that,similar to Fig1, hidden layers 2 and 3 have can cause higher variability in accuraccy, with the median value for 3 layers to be much higher. Variation of accuracy accross different hidden units also look similar to Fig1. When hidden units are lesser variation is high although the range of accuraccy over different number of units looks same. When activation function is tanh the variation is high with median value higher than ReLU. As in experiment 1a. with learning rate lower i.e. 0.005 we see that accuracy has higher maximum value as compared to learning rate 0.01.

The above experiments reveals that while training neural networks one must be very careful while setting the hyperparameters. It is a good practice to iterate over a space of hyperparameters and choose the best as choice of best parameters may not always be intuitive.

Experiment 2

For experiment 2 I implemented convolutional networks with 2 convolutional layers. I set the error function to be cross entropy and the activation function was ReLU. Then, I trained models with different hyperparameters and found the best model i.e. model with highest accuraccy on test set. I iterated over the following hyperparameters: * Hidden units in layer1 * Hidden units in layer2 * Kernel size * Number of filters * Learning rate

To train convolutional network first I converted the given data into 3D data of 8x8x1 ,where 8x8 was the image size 1 is the filter i.e. grayscale. I wrote the attached R code for simulation of convolutional nets (using mxnet). The results are described in Table 3.

Experiment 2 R Code

```r
# clear workspace
rm(list = ls())
# Load MXNet
require(mxnet)
library("magrittr", lib.loc = "~/R/x86_64-pc-linux-gnu-library/3.4")
library(data.table)
# load data files
train <- fread("optdigits.tra", stringsAsFactors = T, colClasses = c(rep("numeric",
    64), "character"))
test <- fread("optdigits.tes", stringsAsFactors = T, colClasses = c(rep("numeric",
    64), "character"))
# set train and test data
train_cn <- data.matrix(train)
train_x <- t(train_cn[, 1:64])
train_y <- train_cn[, 65]
train_array <- train_x

test_cn <- data.matrix(test)
test_x <- t(test_cn[, 1:64])
test_y <- test[, 65]
test_array <- test_x
## missed steps resize to 8x8 image
dim(train_array) <- c(8, 8, 1, ncol(train_x))
dim(test_array) <- c(8, 8, 1, ncol(test_x))

data <- mx.symbol.Variable("data")

# define hyperparameter space
K <- c(2)
numF <- c(20, 40)
hid1 <- c(200, 500)
hid2 <- c(10, 40)
learnRate <- c(0.005, 0.1)
## test K<-c(2) numF<-c(20) hid1<-c(500) hid2<-c(40) learnRate<-c(0.1)

# create table for results Table to write results
header3 <- c("Num.Convlayers", "Units in conv1", "Units in conv2", "kernel",
    "NumFilter", "Rate", "Acc_train", "Acc_test", "Time in min")
resultsTab3 <- data.frame(matrix(ncol = 9, nrow = 0))
colnames(resultsTab3) <- header3

bestModelc <- NULL
bestAcc <- 0
# error func cross entropy, hidden activation ReLU
for (h1 in hid1) {
    for (h2 in hid2) {
        for (ks in K) {
            for (f in numF) {
                for (r in learnRate) {
                  cat(h1, h2, ks, f, r)
                  s <- proc.time()  #start time
```

```
# 1st convolutional layer
conv_1 <- mx.symbol.Convolution(data = data, kernel = c(ks,
  ks), num_filter = f)
relu_1 <- mx.symbol.Activation(data = conv_1, act_type = "relu")
pool_1 <- mx.symbol.Pooling(data = relu_1, pool_type = "max",
  kernel = c(ks, ks))
# 2nd convolutional layer
conv_2 <- mx.symbol.Convolution(data = pool_1, kernel = c(ks,
  ks), num_filter = f)
relu_2 <- mx.symbol.Activation(data = conv_2, act_type = "relu")
pool_2 <- mx.symbol.Pooling(data = relu_2, pool_type = "max",
  kernel = c(ks, ks))
# 1st fully connected layer
flat <- mx.symbol.Flatten(data = pool_2)
fcl_1 <- mx.symbol.FullyConnected(data = flat, num_hidden = h1)
relu_3 <- mx.symbol.Activation(data = fcl_1, act_type = "relu")
# 2nd fully connected layer
fcl_2 <- mx.symbol.FullyConnected(data = relu_3, num_hidden = h2)
# Output
NN_model <- mx.symbol.SoftmaxOutput(data = fcl_2, name = "softmax")
# Set seed for reproducibility
mx.set.seed(100)
# use CPU
device <- mx.cpu()
# Train whole training data
model <- mx.model.FeedForward.create(NN_model, X = train_array,
  y = train_y, ctx = device, num.round = 10, array.batch.size = 100,
  learning.rate = r, eval.metric = mx.metric.accuracy, epoch.end.callback = mx.callba
  verbose = F)
d <- proc.time() - s  #end time
# accuraccy on train set
predict_probs <- predict(model, train_array)
predicted_labels <- max.col(t(predict_probs)) - 1
correct <- 0
for (i in 1:length(predicted_labels)) {
  if (as.numeric(predicted_labels[i]) == as.numeric(train$V65[i])) {
    correct <- correct + 1
  }
}
acc_tr <- format(correct/length(predicted_labels), digits = 4)
# cat('ConvNet Accuraccy on train set:', acc) accuraccy on test set
predict_probs <- predict(model, test_array)
predicted_labels <- max.col(t(predict_probs)) - 1
correct <- 0
for (i in 1:length(predicted_labels)) {
  if (as.numeric(predicted_labels[i]) == as.numeric(test$V65[i])) {
    correct <- correct + 1
  }
}
acc <- format(correct/length(predicted_labels), digits = 4)
# cat('ConvNet Accuraccy on test set:', acc)

# add to table
```

```r
                    resultsTab3[nrow(resultsTab3) + 1, ] <- c(2, h1, h2, paste("(",
                      ks, ",", ks, ")", sep = ""), f, r, acc_tr, acc, format(as.numeric(d)[3]/60,
                      digits = 2))

                    # choose best model
                    if (acc > bestAcc) {
                      bestModelc <- model
                      bestAcc <- acc
                    }

                  }
              }
          }
      }
}
```

```
## 200 10 2 20 0.005200 10 2 20 0.1200 10 2 40 0.005200 10 2 40 0.1200 40 2 20 0.005200 40 2 20 0.1200 4
```

```r
confusion_matrix <- table(predicted_labels, t(test_y))

resultsTab3 %>% knitr::kable(caption = "Experiment 2 outcomes. Error function was cross-entropy and hid
```

Table 3: Experiment 2 outcomes. Error function was cross-entropy and hidden units were ReLU.

Num.Convlayers	Units in conv1	Units in conv2	kernel	NumFilter	Rate	Acc_train	Acc_test	Time in min
2	200	10	(2,2)	20	0.005	0.1012	0.1007	0.56
2	200	10	(2,2)	20	0.1	0.1025	0.1018	0.59
2	200	10	(2,2)	40	0.005	0.1012	0.1007	0.36
2	200	10	(2,2)	40	0.1	0.1018	0.1018	0.48
2	200	40	(2,2)	20	0.005	0.1012	0.1007	0.59
2	200	40	(2,2)	20	0.1	0.5075	0.4402	0.58
2	200	40	(2,2)	40	0.005	0.1012	0.1007	0.62
2	200	40	(2,2)	40	0.1	0.9584	0.9354	0.64
2	500	10	(2,2)	20	0.005	0.1012	0.1007	0.57
2	500	10	(2,2)	20	0.1	0.2004	0.1925	0.56
2	500	10	(2,2)	40	0.005	0.1012	0.1007	0.51
2	500	10	(2,2)	40	0.1	0.3584	0.3534	0.53
2	500	40	(2,2)	20	0.005	0.1012	0.1007	0.57
2	500	40	(2,2)	20	0.1	0.8397	0.798	0.53
2	500	40	(2,2)	40	0.005	0.1012	0.1007	0.61
2	500	40	(2,2)	40	0.1	0.9393	0.8971	0.6

From Table 3 we can see that the convolutional net model with highest accuraccy in experiment 2 had following parameters: 2, 200, 40, (2,2), 40, 0.1, 0.9584, 0.9354, 0.64.

Experiment 2: Best model confusion matrix on test set

```r
confusion_matrix
```

```
##
## predicted_labels    0    1    2    3    4    5    6    7    8    9
```

```
##              1  175    0    0    0    1    2    3    0    1    0
##              2    0  160    0   12   13    2    1    1   25    5
##              3    0    0  170    2    0    0    0    2    1    0
##              4    0    5    0  155    0    0    0    0    0    2
##              5    2    0    0    0  162    1    1    5    0    0
##              6    0   10    0    1    0  172    4    0    1    3
##              7    1    0    0    0    1    2  171    0    0    0
##              8    0    0    5    1    0    0    0  151    2    3
##              9    0    0    0    2    3    0    1    4  129    0
##              10   0    7    2   10    1    3    0   16   15  167
```

Experiment 2: Discussion

Convolutional networks take much more time to build and given sufficient iteration time I saw convolutional networks can achevie 100% accuraccy on training data and still perform better on the test set. Overall, I found that training convolutional networks one must be careful to set the hyperparameters. If parameters are set icorrectly the convolutional network may not learn true model and will give poor results. E.g. when hidden units in second layers is less than 50 the accuraccy is only 10%. On the other hand if parameters are set to learn and iterate over data slowly, the convolutional network will take a lot of time to converge. E.g. set the learn rate 0.005 with 500 hidden units in each layer, and num.iterations = 500 the model will acheive accuraccy 100% on training set but will take a lot of time to build.

Compared to feedforward networks, convolutional networks are much better for image/pattern recognition.

Appendix A

System information

```r
sessionInfo()
```

```
## R version 3.4.1 (2017-06-30)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Debian GNU/Linux 8 (jessie)
##
## Matrix products: default
## BLAS: /usr/lib/openblas-base/libblas.so.3
## LAPACK: /usr/lib/libopenblasp-r0.2.12.so
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] mxnet_1.2.0         data.table_1.10.4-3 h2o_3.16.0.2
```

```
## [4] magrittr_1.5
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.15       plyr_1.8.4         compiler_3.4.1
##  [4] pillar_1.1.0       formatR_1.5        RColorBrewer_1.1-2
##  [7] influenceR_0.1.0   highr_0.6          bindr_0.1
## [10] viridis_0.5.0      bitops_1.0-6       tools_3.4.1
## [13] digest_0.6.15      viridisLite_0.3.0  gtable_0.2.0
## [16] jsonlite_1.5       evaluate_0.10.1    tibble_1.4.2
## [19] rgexf_0.15.3       pkgconfig_2.0.1    rlang_0.2.0
## [22] igraph_1.1.2       rstudioapi_0.7     yaml_2.1.17
## [25] bindrcpp_0.2       gridExtra_2.3      downloader_0.4
## [28] DiagrammeR_1.0.0   dplyr_0.7.4        stringr_1.3.0
## [31] knitr_1.20         htmlwidgets_1.0    hms_0.4.1
## [34] grid_3.4.1         rprojroot_1.3-2    glue_1.2.0
## [37] R6_2.2.2           Rook_1.1-1         XML_3.98-1.10
## [40] rmarkdown_1.9      ggplot2_2.2.1      tidyr_0.8.0
## [43] purrr_0.2.4        readr_1.1.1        codetools_0.2-15
## [46] scales_0.5.0       backports_1.1.2    htmltools_0.3.6
## [49] assertthat_0.2.0   colorspace_1.3-2   brew_1.0-6
## [52] stringi_1.1.6      visNetwork_2.0.3   lazyeval_0.2.1
## [55] munsell_0.4.3      RCurl_1.95-4.10
```