Advanced Deep learning tutorial - H2o

R installation instructions are at http://h2o.ai/download For a full tutorial on h2o, please go see [Link] (https://docs.h2o.ai/h2o-tutorials/latest-stable/H2OTutorialsBook.pdf)

```
library(h2o)
```

```
##
## ----------------------------------------------------------------------
##
## Your next step is to start H2O:
##     > h2o.init()
##
## For H2O package documentation, ask for help:
##     > ??h2o
##
## After starting H2O, you can use the Web UI at http://localhost:54321
## For more information visit https://docs.h2o.ai
##
## ----------------------------------------------------------------------
```

```
##
## Attaching package: 'h2o'
```

```
## The following objects are masked from 'package:stats':
##
##     cor, sd, var
```

```
## The following objects are masked from 'package:base':
##
##     %*%, %in%, &&, ||, apply, as.factor, as.numeric, colnames,
##     colnames<-, ifelse, is.character, is.factor, is.numeric, log,
##     log10, log1p, log2, round, signif, trunc
```

Start H2o

Start up a 1-node H2o server on your local machine, and allow it to use all CPU cores and up to 2-8GB of memory:

```
h2o.init(nthreads = -1, max_mem_size = "8G")
```

```
##
## H2O is not running yet, starting it now...
##
## Note:  In case of errors look at the following log files:
##     C:\Users\USER\AppData\Local\Temp\RtmpaEcqza\file34dc56256545/h2o_USER_started_from_r.out
##     C:\Users\USER\AppData\Local\Temp\RtmpaEcqza\file34dc57c7167c/h2o_USER_started_from_r.err
##
##
## Starting H2O JVM and connecting: . Connection successful!
##
```

```
## R is connected to the H2O cluster:
##     H2O cluster uptime:         6 seconds 885 milliseconds
##     H2O cluster timezone:       Europe/Berlin
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.36.0.3
##     H2O cluster version age:    25 days
##     H2O cluster name:           H2O_started_from_R_USER_stx418
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   7.10 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
##     R Version:                  R version 4.0.5 (2021-03-31)
```

```
h2o.removeAll() ## clean slate - just in case the cluster was already running
```

If you would like to explore Deep Learning you can use the following commands `args(h2o.deeplearning)`, `help(h2o.deeplearning)`, `example(h2o.deeplearning)`.

For this tutorial, we will use CoverType Dataset. This can be found in the following [Link] (https://docs.h2o.ai/h2o-tutorials/latest-stable/tutorials/data/).

What is this data? The original ForestCover/Covertype dataset from UCI machine learning repository is a multiclass classification dataset. It is used in predicting forest cover type from cartographic variables only (no remotely sensed data).

```
df <- h2o.importFile(path=normalizePath("../Deeplearning/covtype.full.csv"))
```

```
##   |                                                                              |
```

```
dim(df)
```

```
## [1] 581012     13
```

```
df
```

```
##   Elevation Aspect Slope Horizontal_Distance_To_Hydrology
## 1      3066    124     5                                0
## 2      3136     32    20                              450
## 3      2655     28    14                               42
## 4      3191     45    19                              323
## 5      3217     80    13                               30
## 6      3119    293    13                               30
##   Vertical_Distance_To_Hydrology Horizontal_Distance_To_Roadways Hillshade_9am
## 1                              0                             1533           229
## 2                            -38                             1290           211
## 3                              8                             1890           214
## 4                             88                             3932           221
```

```
## 5                            1                            3901           237
## 6                           10                            4810           182
##   Hillshade_Noon Hillshade_3pm Horizontal_Distance_To_Fire_Points
## 1            236           141                               459
## 2            193           111                              1112
## 3            209           128                              1001
## 4            195           100                              2919
## 5            217           109                              2859
## 6            237           194                              1200
##   Wilderness_Area Soil_Type Cover_Type
## 1          area_0   type_22    class_1
## 2          area_0   type_28    class_1
## 3          area_2    type_9    class_2
## 4          area_0   type_39    class_2
## 5          area_0   type_22    class_7
## 6          area_0   type_21    class_1
##
## [581012 rows x 13 columns]
```

```
splits <- h2o.splitFrame(df, c(0.6,0.2), seed = 1234)
train <- h2o.assign(splits[[1]], "train.hex")
valid <- h2o.assign(splits[[2]], "valid.hex")
test <- h2o.assign(splits[[3]], "test.hex")
```

```
response <- "Cover_Type"
predictors <- setdiff(names(df), response)
predictors
```

```
##  [1] "Elevation"                          "Aspect"
##  [3] "Slope"                              "Horizontal_Distance_To_Hydrology"
##  [5] "Vertical_Distance_To_Hydrology"     "Horizontal_Distance_To_Roadways"
##  [7] "Hillshade_9am"                      "Hillshade_Noon"
##  [9] "Hillshade_3pm"                      "Horizontal_Distance_To_Fire_Points"
## [11] "Wilderness_Area"                    "Soil_Type"
```

First run of H2o deep learning

We have learned how to build basic DL model. To keep it fast, lets use epoch = 2 or 1

```
m1 <- h2o.deeplearning(
model_id="dl_model_first",
training_frame=train,
validation_frame=valid, ## validation dataset: used for scoring and early stopping
x=predictors,
y=response,
#activation="Rectifier", ## default
#hidden=c(200,200), ## default: 2 hidden layers with 200 neurons each
epochs=1,
variable_importances=T ## not enabled by default
)
```

```
##   |                                                                              |
```

```
summary(m1)
```

```
## Model Details:
## ==============
##
## H2OMultinomialModel: deeplearning
## Model Key:  dl_model_first
## Status of Neuron Layers: predicting Cover_Type, 7-class classification, multinomial distribution, Cr
##   layer units      type dropout       l1       l2 mean_rate rate_rms momentum
## 1     1    56     Input  0.00 %       NA       NA        NA       NA       NA
## 2     2   200 Rectifier  0.00 % 0.000000 0.000000  0.051101 0.211834 0.000000
## 3     3   200 Rectifier  0.00 % 0.000000 0.000000  0.008100 0.006504 0.000000
## 4     4     7   Softmax      NA 0.000000 0.000000  0.121813 0.302161 0.000000
##   mean_weight weight_rms mean_bias bias_rms
## 1          NA         NA        NA       NA
## 2   -0.010715   0.117495  0.056653 0.126999
## 3   -0.028280   0.112086  0.776293 0.287260
## 4   -0.287863   0.498058 -0.465916 0.110496
##
## H2OMultinomialMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on temporary training frame with 10001 samples **
##
## Training Set Metrics:
## =====================
##
## MSE: (Extract with `h2o.mse`) 0.1417672
## RMSE: (Extract with `h2o.rmse`) 0.3765199
## Logloss: (Extract with `h2o.logloss`) 0.4524532
## Mean Per-Class Error: 0.3271991
## AUC: (Extract with `h2o.auc`) NaN
## AUCPR: (Extract with `h2o.aucpr`) NaN
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,train = TRUE)`)
## =========================================================================
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##         class_1 class_2 class_3 class_4 class_5 class_6 class_7  Error
## class_1    3246     308       4       0       1       4      26 0.0956
## class_2    1062    3778      71       1      19      41       7 0.2412
## class_3       3      17     522       2       0      93       0 0.1805
## class_4       0       0      22      15       0       6       0 0.6512
## class_5      19      66       8       0      56       1       0 0.6267
## class_6       0      11      82       1       0     193       0 0.3275
## class_7      53       0       0       0       0       0     263 0.1677
## Totals     4383    4180     709      19      76     338     296 0.1928
##                     Rate
## class_1 =    343 / 3,589
## class_2 =  1,201 / 4,979
## class_3 =      115 / 637
## class_4 =        28 / 43
## class_5 =       94 / 150
## class_6 =       94 / 287
## class_7 =       53 / 316
## Totals  = 1,928 / 10,001
```

```
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,train = TRUE)`
## =======================================================================
## Top-7 Hit Ratios:
##   k hit_ratio
## 1 1  0.807219
## 2 2  0.983702
## 3 3  0.997100
## 4 4  0.999200
## 5 5  0.999900
## 6 6  1.000000
## 7 7  1.000000
##
##
##
##
## H2OMultinomialMetrics: deeplearning
## ** Reported on validation data. **
## ** Metrics reported on full validation frame **
##
## Validation Set Metrics:
## =====================
##
## Extract validation frame with `h2o.getFrame("valid.hex")`
## MSE: (Extract with `h2o.mse`) 0.1458608
## RMSE: (Extract with `h2o.rmse`) 0.3819173
## Logloss: (Extract with `h2o.logloss`) 0.4627287
## Mean Per-Class Error: 0.3356054
## AUC: (Extract with `h2o.auc`) NaN
## AUCPR: (Extract with `h2o.aucpr`) NaN
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,valid = TRUE)`)
## =========================================================================
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##         class_1 class_2 class_3 class_4 class_5 class_6 class_7  Error
## class_1   38268    3811       3       0       7      44     367 0.0996
## class_2   12428   42009     913       1     179     780      70 0.2549
## class_3       3     168    6000      24       1     947       0 0.1600
## class_4       0       0     303     204       0      55       0 0.6370
## class_5     245     860      81       0     663      21       0 0.6455
## class_6      17     198     918      17       0    2314       0 0.3320
## class_7     892      11       0       0       0       0    3196 0.2203
## Totals    51853   47057    8218     246     850    4161    3633 0.2014
##                       Rate
## class_1 =   4,232 / 42,500
## class_2 =  14,371 / 56,380
## class_3 =    1,143 / 7,143
## class_4 =        358 / 562
## class_5 =    1,207 / 1,870
## class_6 =    1,150 / 3,464
## class_7 =      903 / 4,099
## Totals  = 23,364 / 116,018
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,valid = TRUE)`
## =======================================================================
```

```
## Top-7 Hit Ratios:
##   k hit_ratio
## 1 1  0.798617
## 2 2  0.980986
## 3 3  0.997690
## 4 4  0.999379
## 5 5  0.999983
## 6 6  1.000000
## 7 7  1.000000
##
##
##
##
##
##
## Scoring History:
##             timestamp   duration training_speed  epochs iterations
## 1 2022-03-14 10:43:50  0.000 sec             NA 0.00000          0
## 2 2022-03-14 10:43:53  4.124 sec  14731 obs/sec 0.10063          1
## 3 2022-03-14 10:44:03 14.119 sec  15100 obs/sec 0.50010          5
## 4 2022-03-14 10:44:10 21.683 sec  19089 obs/sec 1.00138         10
##         samples training_rmse training_logloss training_r2
## 1      0.000000            NA               NA          NA
## 2  35120.000000       0.46574          0.69119     0.88100
## 3 174542.000000       0.39006          0.47676     0.91653
## 4 349497.000000       0.37652          0.45245     0.92223
##   training_classification_error training_auc training_pr_auc validation_rmse
## 1                            NA           NA              NA              NA
## 2                       0.29577           NA              NA         0.47214
## 3                       0.20558           NA              NA         0.39473
## 4                       0.19278           NA              NA         0.38192
##   validation_logloss validation_r2 validation_classification_error
## 1                 NA            NA                             NA
## 2            0.71534       0.88573                        0.30520
## 3            0.48693       0.92013                        0.21003
## 4            0.46273       0.92523                        0.20138
##   validation_auc validation_pr_auc
## 1             NA                NA
## 2             NA                NA
## 3             NA                NA
## 4             NA                NA
##
## Variable Importances: (Extract with `h2o.varimp`)
## =================================================
##
## Variable Importances:
##                            variable relative_importance scaled_importance
## 1                         Elevation            1.000000          1.000000
## 2             Wilderness_Area.area_0            0.982811          0.982811
## 3    Horizontal_Distance_To_Roadways            0.980470          0.980470
## 4 Horizontal_Distance_To_Fire_Points            0.951689          0.951689
## 5             Wilderness_Area.area_2            0.860750          0.860750
##   percentage
## 1   0.030243
```

```
## 2   0.029723
## 3   0.029652
## 4   0.028781
## 5   0.026031
##
## ---
##                          variable relative_importance scaled_importance
## 51 Vertical_Distance_To_Hydrology            0.477990          0.477990
## 52              Soil_Type.type_14            0.470552          0.470552
## 53                  Hillshade_3pm            0.428980          0.428980
## 54                         Aspect            0.328295          0.328295
## 55          Soil_Type.missing(NA)            0.000000          0.000000
## 56    Wilderness_Area.missing(NA)            0.000000          0.000000
##    percentage
## 51   0.014456
## 52   0.014231
## 53   0.012973
## 54   0.009928
## 55   0.000000
## 56   0.000000
```

Variable Importances

How do we check variable importances for DL? Be aware that variable importances for DL is complex and you should be aware of potential pitfalls. Further information can be found here: https://arxiv.org/pdf/1901.09839.pdf

```
head(as.data.frame(h2o.varimp(m1)))
```

```
##                            variable relative_importance scaled_importance
## 1                         Elevation           1.0000000         1.0000000
## 2             Wilderness_Area.area_0           0.9828110         0.9828110
## 3    Horizontal_Distance_To_Roadways           0.9804703         0.9804703
## 4 Horizontal_Distance_To_Fire_Points           0.9516892         0.9516892
## 5             Wilderness_Area.area_2           0.8607497         0.8607497
## 6             Wilderness_Area.area_3           0.8167333         0.8167333
##   percentage
## 1 0.03024252
## 2 0.02972268
## 3 0.02965189
## 4 0.02878148
## 5 0.02603124
## 6 0.02470007
```

Adaptive Learning Rate

By default, H2O Deep learning uses an adaptive learning rate (ADADELTA) for its stochastic gradient descent optimization. There are oly two tuning parameters for this model: `rho` and `epsilon`. `rho` is the similarity to prior weight updates (similar to momentum), and `epsilon` is a parametter that prevents the optimization to get stuck in local optima.

Hyper-parameter Tuning with Gridsearch

As we know, there are a lot of parameters that can impact model accuracy.

For speed, we will only train on the first 10,000 rows of the training dataset.

```
sampled_train = train[1:10000,]
```

first we need to set our grid.

```
hyper_params <- list(
hidden=list(c(32,32,32),c(64,64)),
input_dropout_ratio=c(0,0.05),
rate=c(0.01,0.02),
rate_annealing=c(1e-8,1e-7,1e-6)
)
hyper_params
```

```
## $hidden
## $hidden[[1]]
## [1] 32 32 32
##
## $hidden[[2]]
## [1] 64 64
##
##
## $input_dropout_ratio
## [1] 0.00 0.05
##
## $rate
## [1] 0.01 0.02
##
## $rate_annealing
## [1] 1e-08 1e-07 1e-06
```

```
grid <- h2o.grid(
algorithm="deeplearning",
grid_id="dl_grid",
training_frame=sampled_train,
validation_frame=valid,
x=predictors,
y=response,
epochs=10,
stopping_metric="misclassification",
stopping_tolerance=1e-2, ## stop when misclassification does not improve by >=1% for 2 scoring events
stopping_rounds=2,
score_validation_samples=10000, ## downsample validation set for faster scoring
score_duty_cycle=0.025, ## don't score more than 2.5% of the wall time
adaptive_rate=F, ## manually tuned learning rate
momentum_start=0.5, ## manually tuned momentum
momentum_stable=0.9,
momentum_ramp=1e7,
l1=1e-5,
```

```
l2=1e-5,
activation=c("Rectifier"),
max_w2=10, ## can help improve stability for Rectifier
hyper_params=hyper_params
)
```

```
##   |                                                                      |
```

```
grid
```

```
## H2O Grid Details
## ================
##
## Grid ID: dl_grid
## Used hyper parameters:
##   -  hidden
##   -  input_dropout_ratio
##   -  rate
##   -  rate_annealing
## Number of models: 24
## Number of failed models: 0
##
## Hyper-Parameter Search Summary: ordered by increasing logloss
##         hidden input_dropout_ratio    rate rate_annealing        model_ids
## 1     [64, 64]             0.00000 0.01000        0.00000 dl_grid_model_18
## 2 [32, 32, 32]             0.00000 0.01000        0.00000  dl_grid_model_1
## 3     [64, 64]             0.00000 0.02000        0.00000  dl_grid_model_6
## 4     [64, 64]             0.00000 0.01000        0.00000  dl_grid_model_2
## 5     [64, 64]             0.05000 0.01000        0.00000  dl_grid_model_4
##   logloss
## 1 0.56860
## 2 0.57892
## 3 0.58494
## 4 0.58758
## 5 0.59320
##
## ---
##          hidden input_dropout_ratio    rate rate_annealing        model_ids
## 19 [32, 32, 32]             0.05000 0.01000        0.00000 dl_grid_model_19
## 20 [32, 32, 32]             0.05000 0.02000        0.00000  dl_grid_model_7
## 21     [64, 64]             0.05000 0.01000        0.00000 dl_grid_model_20
## 22 [32, 32, 32]             0.00000 0.02000        0.00000 dl_grid_model_21
## 23 [32, 32, 32]             0.05000 0.02000        0.00000 dl_grid_model_23
## 24 [32, 32, 32]             0.05000 0.02000        0.00000 dl_grid_model_15
##    logloss
## 19 0.62806
## 20 0.62843
## 21 0.63125
## 22 0.63433
## 23 0.64934
## 24 0.67498
```

Now, let's see which model had the lowest validation error:

```
grid <- h2o.getGrid("dl_grid",sort_by="err",decreasing=FALSE)
grid
```

```
## H2O Grid Details
## ================
##
## Grid ID: dl_grid
## Used hyper parameters:
##   -  hidden
##   -  input_dropout_ratio
##   -  rate
##   -  rate_annealing
## Number of models: 24
## Number of failed models: 0
##
## Hyper-Parameter Search Summary: ordered by increasing err
##         hidden input_dropout_ratio    rate rate_annealing        model_ids
## 1     [64, 64]             0.00000 0.01000        0.00000 dl_grid_model_18
## 2     [64, 64]             0.00000 0.02000        0.00000  dl_grid_model_6
## 3     [64, 64]             0.00000 0.01000        0.00000  dl_grid_model_2
## 4     [64, 64]             0.05000 0.01000        0.00000 dl_grid_model_12
## 5 [32, 32, 32]             0.00000 0.01000        0.00000  dl_grid_model_1
##       err
## 1 0.24832
## 2 0.24995
## 3 0.25204
## 4 0.25309
## 5 0.25327
##
## ---
##          hidden input_dropout_ratio    rate rate_annealing        model_ids
## 19 [32, 32, 32]             0.00000 0.02000        0.00000 dl_grid_model_13
## 20 [32, 32, 32]             0.05000 0.02000        0.00000  dl_grid_model_7
## 21     [64, 64]             0.05000 0.01000        0.00000 dl_grid_model_20
## 22 [32, 32, 32]             0.05000 0.02000        0.00000 dl_grid_model_23
## 23     [64, 64]             0.05000 0.02000        0.00000  dl_grid_model_8
## 24 [32, 32, 32]             0.05000 0.02000        0.00000 dl_grid_model_15
##        err
## 19 0.26921
## 20 0.26929
## 21 0.27457
## 22 0.27555
## 23 0.28076
## 24 0.28459
```

```
## To see what other "sort_by" criteria are allowed
#grid <- h2o.getGrid("dl_grid",sort_by="wrong_thing",decreasing=FALSE)

## Sort by logloss
h2o.getGrid("dl_grid",sort_by="logloss",decreasing=FALSE)
```

```
## H2O Grid Details
```

```
## ================
##
## Grid ID: dl_grid
## Used hyper parameters:
##   -  hidden
##   -  input_dropout_ratio
##   -  rate
##   -  rate_annealing
## Number of models: 24
## Number of failed models: 0
##
## Hyper-Parameter Search Summary: ordered by increasing logloss
##         hidden input_dropout_ratio    rate rate_annealing        model_ids
## 1     [64, 64]             0.00000 0.01000        0.00000 dl_grid_model_18
## 2 [32, 32, 32]             0.00000 0.01000        0.00000  dl_grid_model_1
## 3     [64, 64]             0.00000 0.02000        0.00000  dl_grid_model_6
## 4     [64, 64]             0.00000 0.01000        0.00000  dl_grid_model_2
## 5     [64, 64]             0.05000 0.01000        0.00000  dl_grid_model_4
##   logloss
## 1 0.56860
## 2 0.57892
## 3 0.58494
## 4 0.58758
## 5 0.59320
##
## ---
##          hidden input_dropout_ratio    rate rate_annealing        model_ids
## 19 [32, 32, 32]             0.05000 0.01000        0.00000 dl_grid_model_19
## 20 [32, 32, 32]             0.05000 0.02000        0.00000  dl_grid_model_7
## 21     [64, 64]             0.05000 0.01000        0.00000 dl_grid_model_20
## 22 [32, 32, 32]             0.00000 0.02000        0.00000 dl_grid_model_21
## 23 [32, 32, 32]             0.05000 0.02000        0.00000 dl_grid_model_23
## 24 [32, 32, 32]             0.05000 0.02000        0.00000 dl_grid_model_15
##    logloss
## 19 0.62806
## 20 0.62843
## 21 0.63125
## 22 0.63433
## 23 0.64934
## 24 0.67498
```

```
## Find the best model and its full set of parameters
grid@summary_table[1,]
```

```
## Hyper-Parameter Search Summary: ordered by increasing err
##     hidden input_dropout_ratio    rate rate_annealing        model_ids     err
## 1 [64, 64]             0.00000 0.01000        0.00000 dl_grid_model_18 0.24832
```

```
best_model <- h2o.getModel(grid@model_ids[[1]])
best_model
```

```
## Model Details:
## ==============
```

```
##
## H2OMultinomialModel: deeplearning
## Model ID:  dl_grid_model_18
## Status of Neuron Layers: predicting Cover_Type, 7-class classification, multinomial distribution, Cr
##   layer units      type dropout       l1       l2 mean_rate rate_rms momentum
## 1     1    56     Input  0.00 %       NA       NA        NA       NA       NA
## 2     2    64 Rectifier  0.00 % 0.000010 0.000010  0.009091 0.000000 0.504000
## 3     3    64 Rectifier  0.00 % 0.000010 0.000010  0.009091 0.000000 0.504000
## 4     4     7   Softmax      NA 0.000010 0.000010  0.009091 0.000000 0.504000
##   mean_weight weight_rms mean_bias bias_rms
## 1          NA         NA        NA       NA
## 2   -0.012098   0.209746  0.228521 0.140636
## 3   -0.057936   0.187991  0.869040 0.151229
## 4    0.006092   0.393333 -0.025930 0.576858
##
##
## H2OMultinomialMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on full training frame **
##
## Training Set Metrics:
## =====================
##
## Extract training frame with `h2o.getFrame("RTMP_sid_b932_6")`
## MSE: (Extract with `h2o.mse`) 0.1651635
## RMSE: (Extract with `h2o.rmse`) 0.4064031
## Logloss: (Extract with `h2o.logloss`) 0.5106568
## Mean Per-Class Error: 0.3836606
## AUC: (Extract with `h2o.auc`) NaN
## AUCPR: (Extract with `h2o.aucpr`) NaN
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,train = TRUE)`)
## ========================================================================
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##         class_1 class_2 class_3 class_4 class_5 class_6 class_7  Error
## class_1    2763     820       1       0       4       0     100 0.2508
## class_2     671    4045      74       0      17      15      13 0.1634
## class_3       0      34     586       6       0       4       0 0.0698
## class_4       0       0      20      24       0       0       0 0.4545
## class_5       3      90       6       0      57       0       0 0.6346
## class_6       0      78     199       0       0      32       0 0.8964
## class_7      72       1       0       0       0       0     265 0.2160
## Totals     3509    5068     886      30      78      51     378 0.2228
##                     Rate
## class_1 =    925 / 3,688
## class_2 =    790 / 4,835
## class_3 =       44 / 630
## class_4 =        20 / 44
## class_5 =       99 / 156
## class_6 =      277 / 309
## class_7 =       73 / 338
## Totals  = 2,228 / 10,000
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,train = TRUE)`
## ========================================================================
```

```
## Top-7 Hit Ratios:
##   k hit_ratio
## 1 1  0.777200
## 2 2  0.979600
## 3 3  0.997400
## 4 4  0.999100
## 5 5  0.999700
## 6 6  1.000000
## 7 7  1.000000
##
##
##
##
## H2OMultinomialMetrics: deeplearning
## ** Reported on validation data. **
## ** Metrics reported on temporary validation frame with 9943 samples **
##
## Validation Set Metrics:
## =====================
##
## MSE: (Extract with `h2o.mse`) 0.1822705
## RMSE: (Extract with `h2o.rmse`) 0.4269315
## Logloss: (Extract with `h2o.logloss`) 0.5686029
## Mean Per-Class Error: 0.4437962
## AUC: (Extract with `h2o.auc`) NaN
## AUCPR: (Extract with `h2o.aucpr`) NaN
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,valid = TRUE)`)
## =========================================================================
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##         class_1 class_2 class_3 class_4 class_5 class_6 class_7  Error
## class_1    2568     921       0       0       4       0     100 0.2853
## class_2     739    4045      71       1      33      12      14 0.1770
## class_3       0      51     524      10       0       4       0 0.1104
## class_4       0       0      28      17       0       1       0 0.6304
## class_5       4     104       4       0      40       1       0 0.7386
## class_6       1      76     203       5       0      24       0 0.9223
## class_7      80       2       0       0       0       0     256 0.2426
## Totals     3392    5199     830      33      77      42     370 0.2483
##                    Rate
## class_1 = 1,025 / 3,593
## class_2 =   870 / 4,915
## class_3 =      65 / 589
## class_4 =       29 / 46
## class_5 =     113 / 153
## class_6 =     285 / 309
## class_7 =      82 / 338
## Totals  = 2,469 / 9,943
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,valid = TRUE)`
## =======================================================================
## Top-7 Hit Ratios:
##   k hit_ratio
## 1 1  0.751685
## 2 2  0.973046
```

```
## 3 3  0.995776
## 4 4  0.998994
## 5 5  0.999799
## 6 6  0.999899
## 7 7  1.000000
```

```
print(best_model@allparameters)
```

```
## $model_id
## [1] "dl_grid_model_18"
##
## $training_frame
## [1] "RTMP_sid_b932_6"
##
## $validation_frame
## [1] "valid.hex"
##
## $nfolds
## [1] 0
##
## $keep_cross_validation_models
## [1] TRUE
##
## $keep_cross_validation_predictions
## [1] FALSE
##
## $keep_cross_validation_fold_assignment
## [1] FALSE
##
## $ignore_const_cols
## [1] TRUE
##
## $score_each_iteration
## [1] FALSE
##
## $balance_classes
## [1] FALSE
##
## $max_after_balance_size
## [1] 5
##
## $max_confusion_matrix_size
## [1] 20
##
## $overwrite_with_best_model
## [1] TRUE
##
## $use_all_factor_levels
## [1] TRUE
##
## $standardize
## [1] TRUE
##
## $activation
```

```
## [1] "Rectifier"
##
## $hidden
## [1] 64 64
##
## $epochs
## [1] 10
##
## $train_samples_per_iteration
## [1] -2
##
## $target_ratio_comm_to_comp
## [1] 0.05
##
## $seed
## [1] "-3026565506544836290"
##
## $adaptive_rate
## [1] FALSE
##
## $rho
## [1] 0.99
##
## $epsilon
## [1] 1e-08
##
## $rate
## [1] 0.01
##
## $rate_annealing
## [1] 1e-06
##
## $rate_decay
## [1] 1
##
## $momentum_start
## [1] 0.5
##
## $momentum_ramp
## [1] 1e+07
##
## $momentum_stable
## [1] 0.9
##
## $nesterov_accelerated_gradient
## [1] TRUE
##
## $input_dropout_ratio
## [1] 0
##
## $l1
## [1] 1e-05
##
## $l2
```

```
## [1] 1e-05
##
## $max_w2
## [1] 10
##
## $initial_weight_distribution
## [1] "UniformAdaptive"
##
## $initial_weight_scale
## [1] 1
##
## $loss
## [1] "Automatic"
##
## $distribution
## [1] "multinomial"
##
## $quantile_alpha
## [1] 0.5
##
## $tweedie_power
## [1] 1.5
##
## $huber_alpha
## [1] 0.9
##
## $score_interval
## [1] 5
##
## $score_training_samples
## [1] 10000
##
## $score_validation_samples
## [1] 10000
##
## $score_duty_cycle
## [1] 0.025
##
## $classification_stop
## [1] 0
##
## $regression_stop
## [1] 1e-06
##
## $stopping_rounds
## [1] 2
##
## $stopping_metric
## [1] "misclassification"
##
## $stopping_tolerance
## [1] 0.01
##
## $max_runtime_secs
```

```
## [1] 1.797693e+308
##
## $score_validation_sampling
## [1] "Uniform"
##
## $diagnostics
## [1] TRUE
##
## $fast_mode
## [1] TRUE
##
## $force_load_balance
## [1] TRUE
##
## $variable_importances
## [1] TRUE
##
## $replicate_training_data
## [1] TRUE
##
## $single_node_mode
## [1] FALSE
##
## $shuffle_training_data
## [1] FALSE
##
## $missing_values_handling
## [1] "MeanImputation"
##
## $quiet_mode
## [1] FALSE
##
## $autoencoder
## [1] FALSE
##
## $sparse
## [1] FALSE
##
## $col_major
## [1] FALSE
##
## $average_activation
## [1] 0
##
## $sparsity_beta
## [1] 0
##
## $max_categorical_features
## [1] 2147483647
##
## $reproducible
## [1] FALSE
##
## $export_weights_and_biases
```

```
## [1] FALSE
##
## $mini_batch_size
## [1] 1
##
## $categorical_encoding
## [1] "OneHotInternal"
##
## $elastic_averaging
## [1] FALSE
##
## $elastic_averaging_moving_rate
## [1] 0.9
##
## $elastic_averaging_regularization
## [1] 0.001
##
## $auc_type
## [1] "AUTO"
##
## $x
##  [1] "Soil_Type"                          "Wilderness_Area"
##  [3] "Elevation"                          "Aspect"
##  [5] "Slope"                              "Horizontal_Distance_To_Hydrology"
##  [7] "Vertical_Distance_To_Hydrology"     "Horizontal_Distance_To_Roadways"
##  [9] "Hillshade_9am"                      "Hillshade_Noon"
## [11] "Hillshade_3pm"                      "Horizontal_Distance_To_Fire_Points"
##
## $y
## [1] "Cover_Type"
```

```
print(h2o.performance(best_model, valid=T))
```

```
## H2OMultinomialMetrics: deeplearning
## ** Reported on validation data. **
## ** Metrics reported on temporary validation frame with 9943 samples **
##
## Validation Set Metrics:
## =====================
##
## MSE: (Extract with `h2o.mse`) 0.1822705
## RMSE: (Extract with `h2o.rmse`) 0.4269315
## Logloss: (Extract with `h2o.logloss`) 0.5686029
## Mean Per-Class Error: 0.4437962
## AUC: (Extract with `h2o.auc`) NaN
## AUCPR: (Extract with `h2o.aucpr`) NaN
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,valid = TRUE)`)
## =========================================================================
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##         class_1 class_2 class_3 class_4 class_5 class_6 class_7  Error
## class_1    2568     921       0       0       4       0     100 0.2853
## class_2     739    4045      71       1      33      12      14 0.1770
## class_3       0      51     524      10       0       4       0 0.1104
## class_4       0       0      28      17       0       1       0 0.6304
```

```
## class_5       4     104       4       0      40       1       0 0.7386
## class_6       1      76     203       5       0      24       0 0.9223
## class_7      80       2       0       0       0       0     256 0.2426
## Totals     3392    5199     830      33      77      42     370 0.2483
##                    Rate
## class_1 = 1,025 / 3,593
## class_2 =   870 / 4,915
## class_3 =      65 / 589
## class_4 =       29 / 46
## class_5 =     113 / 153
## class_6 =     285 / 309
## class_7 =      82 / 338
## Totals  = 2,469 / 9,943
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,valid = TRUE)`
## =======================================================================
## Top-7 Hit Ratios:
##   k hit_ratio
## 1 1  0.751685
## 2 2  0.973046
## 3 3  0.995776
## 4 4  0.998994
## 5 5  0.999799
## 6 6  0.999899
## 7 7  1.000000
```

```
print(h2o.logloss(best_model, valid=T))
```

```
## [1] 0.5686029
```

Random Hyper-Parameter Search

Often, hyper-parameter search for more than 4 parameters can be done more efficiently with random parameter search than with grid search. Please read: https://eranraviv.com/hyper-parameter-optimization-using-random-search/

We simply build up to `max_models` models with parameters drawn randomly from user-specific distributions. For this example, we use the adaptive learning rate and focus on tuning the network architecture and the regularization parameters. We also let he grid search stop automatically once the performance at the top of the leaderboard doesn't change much anymore (i.e., convergence).

```
hyper_params <- list(
activation=c("Rectifier","Tanh","Maxout","RectifierWithDropout","TanhWithDropout","MaxoutWithDropout"),
hidden=list(c(20,20),c(50,50),c(30,30,30),c(25,25,25,25)),
input_dropout_ratio=c(0,0.05),
l1=seq(0,1e-4,1e-6),
l2=seq(0,1e-4,1e-6)
)
hyper_params
```

```
## $activation
## [1] "Rectifier"           "Tanh"                "Maxout"
```

```
## [4] "RectifierWithDropout" "TanhWithDropout"      "MaxoutWithDropout"
##
## $hidden
## $hidden[[1]]
## [1] 20 20
##
## $hidden[[2]]
## [1] 50 50
##
## $hidden[[3]]
## [1] 30 30 30
##
## $hidden[[4]]
## [1] 25 25 25 25
##
##
## $input_dropout_ratio
## [1] 0.00 0.05
##
## $l1
##   [1] 0.0e+00 1.0e-06 2.0e-06 3.0e-06 4.0e-06 5.0e-06 6.0e-06 7.0e-06 8.0e-06
##  [10] 9.0e-06 1.0e-05 1.1e-05 1.2e-05 1.3e-05 1.4e-05 1.5e-05 1.6e-05 1.7e-05
##  [19] 1.8e-05 1.9e-05 2.0e-05 2.1e-05 2.2e-05 2.3e-05 2.4e-05 2.5e-05 2.6e-05
##  [28] 2.7e-05 2.8e-05 2.9e-05 3.0e-05 3.1e-05 3.2e-05 3.3e-05 3.4e-05 3.5e-05
##  [37] 3.6e-05 3.7e-05 3.8e-05 3.9e-05 4.0e-05 4.1e-05 4.2e-05 4.3e-05 4.4e-05
##  [46] 4.5e-05 4.6e-05 4.7e-05 4.8e-05 4.9e-05 5.0e-05 5.1e-05 5.2e-05 5.3e-05
##  [55] 5.4e-05 5.5e-05 5.6e-05 5.7e-05 5.8e-05 5.9e-05 6.0e-05 6.1e-05 6.2e-05
##  [64] 6.3e-05 6.4e-05 6.5e-05 6.6e-05 6.7e-05 6.8e-05 6.9e-05 7.0e-05 7.1e-05
##  [73] 7.2e-05 7.3e-05 7.4e-05 7.5e-05 7.6e-05 7.7e-05 7.8e-05 7.9e-05 8.0e-05
##  [82] 8.1e-05 8.2e-05 8.3e-05 8.4e-05 8.5e-05 8.6e-05 8.7e-05 8.8e-05 8.9e-05
##  [91] 9.0e-05 9.1e-05 9.2e-05 9.3e-05 9.4e-05 9.5e-05 9.6e-05 9.7e-05 9.8e-05
## [100] 9.9e-05 1.0e-04
##
## $l2
##   [1] 0.0e+00 1.0e-06 2.0e-06 3.0e-06 4.0e-06 5.0e-06 6.0e-06 7.0e-06 8.0e-06
##  [10] 9.0e-06 1.0e-05 1.1e-05 1.2e-05 1.3e-05 1.4e-05 1.5e-05 1.6e-05 1.7e-05
##  [19] 1.8e-05 1.9e-05 2.0e-05 2.1e-05 2.2e-05 2.3e-05 2.4e-05 2.5e-05 2.6e-05
##  [28] 2.7e-05 2.8e-05 2.9e-05 3.0e-05 3.1e-05 3.2e-05 3.3e-05 3.4e-05 3.5e-05
##  [37] 3.6e-05 3.7e-05 3.8e-05 3.9e-05 4.0e-05 4.1e-05 4.2e-05 4.3e-05 4.4e-05
##  [46] 4.5e-05 4.6e-05 4.7e-05 4.8e-05 4.9e-05 5.0e-05 5.1e-05 5.2e-05 5.3e-05
##  [55] 5.4e-05 5.5e-05 5.6e-05 5.7e-05 5.8e-05 5.9e-05 6.0e-05 6.1e-05 6.2e-05
##  [64] 6.3e-05 6.4e-05 6.5e-05 6.6e-05 6.7e-05 6.8e-05 6.9e-05 7.0e-05 7.1e-05
##  [73] 7.2e-05 7.3e-05 7.4e-05 7.5e-05 7.6e-05 7.7e-05 7.8e-05 7.9e-05 8.0e-05
##  [82] 8.1e-05 8.2e-05 8.3e-05 8.4e-05 8.5e-05 8.6e-05 8.7e-05 8.8e-05 8.9e-05
##  [91] 9.0e-05 9.1e-05 9.2e-05 9.3e-05 9.4e-05 9.5e-05 9.6e-05 9.7e-05 9.8e-05
## [100] 9.9e-05 1.0e-04
```

```
## Stop once the top 5 models are within 1% of each other (i.e., the windowed average varies less than
search_criteria = list(strategy = "RandomDiscrete", max_runtime_secs = 360, max_models = 100, seed=1234

dl_random_grid <- h2o.grid(
algorithm="deeplearning",
grid_id = "dl_grid_random",
training_frame=sampled_train,
```

```
validation_frame=valid,
x=predictors,
y=response,
epochs=1,
stopping_metric="logloss",
stopping_tolerance=1e-2, ## stop when logloss does not improve by >=1% for 2 scoring events
stopping_rounds=2,
score_validation_samples=10000, ## downsample validation set for faster scoring
score_duty_cycle=0.025, ## don't score more than 2.5% of the wall time
max_w2=10, ## can help improve stability for Rectifier
hyper_params = hyper_params,
search_criteria = search_criteria
)
```

```
##   |                                                                      |
```

```
grid <- h2o.getGrid("dl_grid_random",sort_by="logloss",decreasing=FALSE)
grid
```

```
## H2O Grid Details
## ================
##
## Grid ID: dl_grid_random
## Used hyper parameters:
##   -  activation
##   -  hidden
##   -  input_dropout_ratio
##   -  l1
##   -  l2
## Number of models: 100
## Number of failed models: 0
##
## Hyper-Parameter Search Summary: ordered by increasing logloss
##   activation           hidden input_dropout_ratio      l1      l2
## 1       Tanh         [50, 50]             0.00000 0.00005 0.00009
## 2       Tanh     [30, 30, 30]             0.00000 0.00004 0.00002
## 3       Tanh         [50, 50]             0.00000 0.00009 0.00002
## 4       Tanh         [50, 50]             0.00000 0.00008 0.00007
## 5       Tanh [25, 25, 25, 25]             0.00000 0.00000 0.00008
##                 model_ids logloss
## 1 dl_grid_random_model_38 0.66390
## 2 dl_grid_random_model_23 0.66819
## 3  dl_grid_random_model_6 0.67080
## 4 dl_grid_random_model_93 0.67467
## 5 dl_grid_random_model_42 0.67801
##
## ---
##              activation           hidden input_dropout_ratio      l1      l2
## 95  RectifierWithDropout [25, 25, 25, 25]             0.05000 0.00008 0.00003
## 96  RectifierWithDropout [25, 25, 25, 25]             0.05000 0.00001 0.00010
## 97     MaxoutWithDropout     [30, 30, 30]             0.05000 0.00008 0.00009
## 98     MaxoutWithDropout [25, 25, 25, 25]             0.00000 0.00001 0.00001
## 99     MaxoutWithDropout [25, 25, 25, 25]             0.05000 0.00006 0.00001
```

```
## 100    MaxoutWithDropout [25, 25, 25, 25]             0.00000 0.00006 0.00009
##                   model_ids logloss
## 95  dl_grid_random_model_91 1.16168
## 96  dl_grid_random_model_64 1.22733
## 97  dl_grid_random_model_54 1.42682
## 98  dl_grid_random_model_68 1.67043
## 99  dl_grid_random_model_19 1.69278
## 100 dl_grid_random_model_25 1.81905
```

```
grid@summary_table[1,]
```

```
## Hyper-Parameter Search Summary: ordered by increasing logloss
##   activation   hidden input_dropout_ratio      l1      l2
## 1       Tanh [50, 50]             0.00000 0.00005 0.00009
##                 model_ids logloss
## 1 dl_grid_random_model_38 0.66390
```

```
best_model <- h2o.getModel(grid@model_ids[[1]]) ## model with lowest logloss
best_model
```

```
## Model Details:
## ==============
##
## H2OMultinomialModel: deeplearning
## Model ID:  dl_grid_random_model_38
## Status of Neuron Layers: predicting Cover_Type, 7-class classification, multinomial distribution, Cr
##   layer units    type dropout       l1       l2 mean_rate rate_rms momentum
## 1     1    56   Input  0.00 %       NA       NA        NA       NA       NA
## 2     2    50    Tanh  0.00 % 0.000053 0.000087  0.041632 0.190277 0.000000
## 3     3    50    Tanh  0.00 % 0.000053 0.000087  0.005216 0.002297 0.000000
## 4     4     7 Softmax      NA 0.000053 0.000087  0.004056 0.004507 0.000000
##   mean_weight weight_rms mean_bias bias_rms
## 1          NA         NA        NA       NA
## 2   -0.001031   0.135855 -0.009264 0.108107
## 3    0.001275   0.141971 -0.009207 0.122153
## 4   -0.045411   0.429073 -0.117586 0.165039
##
##
## H2OMultinomialMetrics: deeplearning
## ** Reported on training data. **
## ** Metrics reported on full training frame **
##
## Training Set Metrics:
## =====================
##
## Extract training frame with `h2o.getFrame("RTMP_sid_b932_6")`
## MSE: (Extract with `h2o.mse`) 0.2188036
## RMSE: (Extract with `h2o.rmse`) 0.4677645
## Logloss: (Extract with `h2o.logloss`) 0.674118
## Mean Per-Class Error: 0.5743572
## AUC: (Extract with `h2o.auc`) NaN
## AUCPR: (Extract with `h2o.aucpr`) NaN
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,train = TRUE)`)
```

```
## ================================================================================
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##         class_1 class_2 class_3 class_4 class_5 class_6 class_7  Error
## class_1    2807     828       1       0       0       0      52 0.2389
## class_2    1142    3605      74       0       0       0      14 0.2544
## class_3       0      69     560       0       0       1       0 0.1111
## class_4       0       0      44       0       0       0       0 1.0000
## class_5       1     149       5       0       1       0       0 0.9936
## class_6       0     118     189       0       0       2       0 0.9935
## class_7     144       1       0       0       0       0     193 0.4290
## Totals     4094    4770     873       0       1       3     259 0.2832
##                     Rate
## class_1 =    881 / 3,688
## class_2 =  1,230 / 4,835
## class_3 =       70 / 630
## class_4 =        44 / 44
## class_5 =      155 / 156
## class_6 =      307 / 309
## class_7 =      145 / 338
## Totals  = 2,832 / 10,000
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,train = TRUE)`
## ================================================================================
## Top-7 Hit Ratios:
##   k hit_ratio
## 1 1  0.716800
## 2 2  0.958000
## 3 3  0.991800
## 4 4  0.998100
## 5 5  0.999800
## 6 6  0.999900
## 7 7  1.000000
##
##
##
##
## H2OMultinomialMetrics: deeplearning
## ** Reported on validation data. **
## ** Metrics reported on temporary validation frame with 10003 samples **
##
## Validation Set Metrics:
## =====================
##
## MSE: (Extract with `h2o.mse`) 0.2144664
## RMSE: (Extract with `h2o.rmse`) 0.4631051
## Logloss: (Extract with `h2o.logloss`) 0.663898
## Mean Per-Class Error: 0.5738685
## AUC: (Extract with `h2o.auc`) NaN
## AUCPR: (Extract with `h2o.aucpr`) NaN
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,valid = TRUE)`)
## ================================================================================
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##         class_1 class_2 class_3 class_4 class_5 class_6 class_7  Error
## class_1    2895     803       0       0       0       0      64 0.2305
```

```
## class_2    1083    3629      62       0       0       0       4 0.2405
## class_3       0      77     530       0       0       1       0 0.1283
## class_4       0       0      36       0       0       1       0 1.0000
## class_5       0     154       3       0       0       0       0 1.0000
## class_6       0     103     190       0       0       4       0 0.9865
## class_7     156       1       0       0       0       0     207 0.4313
## Totals     4134    4767     821       0       0       6     275 0.2737
##                     Rate
## class_1 =    867 / 3,762
## class_2 =  1,149 / 4,778
## class_3 =       78 / 608
## class_4 =        37 / 37
## class_5 =      157 / 157
## class_6 =      293 / 297
## class_7 =      157 / 364
## Totals  = 2,738 / 10,003
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,valid = TRUE)`
## =======================================================================
## Top-7 Hit Ratios:
##   k hit_ratio
## 1 1  0.726282
## 2 2  0.961812
## 3 3  0.991303
## 4 4  0.998201
## 5 5  0.999700
## 6 6  0.999900
## 7 7  1.000000
```

Let's look at the model with the lowest validation misclassification rate:

```
grid <- h2o.getGrid("dl_grid",sort_by="err",decreasing=FALSE)
best_model <- h2o.getModel(grid@model_ids[[1]]) ## model with lowest classification error (on validatio
h2o.confusionMatrix(best_model,valid=T)
```

```
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
##         class_1 class_2 class_3 class_4 class_5 class_6 class_7  Error
## class_1    2568     921       0       0       4       0     100 0.2853
## class_2     739    4045      71       1      33      12      14 0.1770
## class_3       0      51     524      10       0       4       0 0.1104
## class_4       0       0      28      17       0       1       0 0.6304
## class_5       4     104       4       0      40       1       0 0.7386
## class_6       1      76     203       5       0      24       0 0.9223
## class_7      80       2       0       0       0       0     256 0.2426
## Totals     3392    5199     830      33      77      42     370 0.2483
##                     Rate
## class_1 = 1,025 / 3,593
## class_2 =   870 / 4,915
## class_3 =      65 / 589
## class_4 =       29 / 46
## class_5 =     113 / 153
## class_6 =     285 / 309
## class_7 =      82 / 338
## Totals  = 2,469 / 9,943
```

```
best_params <- best_model@allparameters
best_params$activation
```

```
## [1] "Rectifier"
```

```
best_params$hidden
```

```
## [1] 64 64
```

```
best_params$input_dropout_ratio
```

```
## [1] 0
```

```
best_params$l1
```

```
## [1] 1e-05
```

```
best_params$l2
```

```
## [1] 1e-05
```