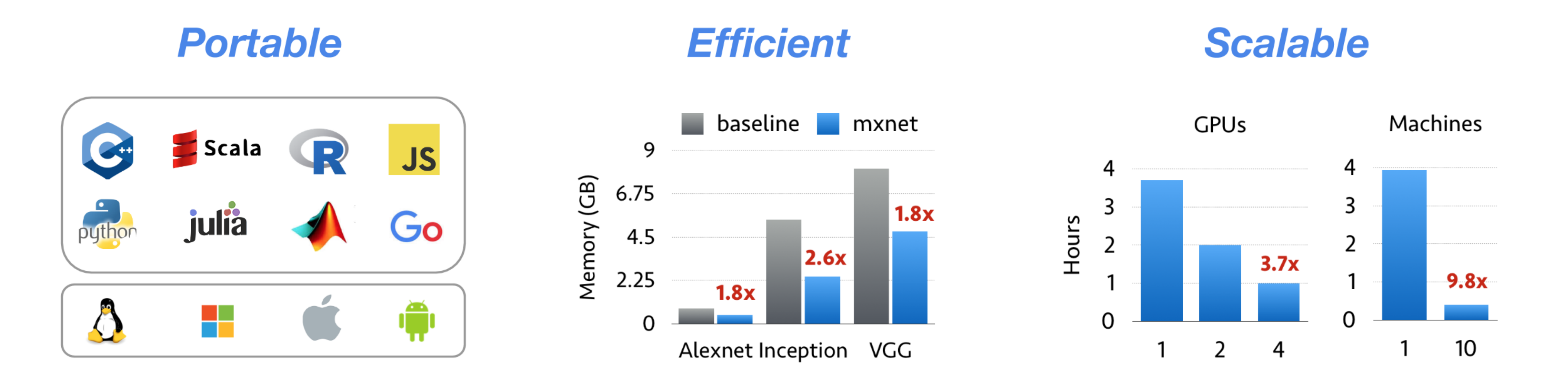
Deep Learning with MXNet and R

Dmitry Larko
October 26th, 2016

What is MXNet?



MXNet is a deep learning framework designed for both *efficiency* and *flexibility*.

The library is portable and lightweight, and it scales to multiple GPUs and multiple machines.

Project's GitHub: https://github.com/dmlc/mxnet

This presentation code: https://github.com/IzuiT/H2O_Dallas_MXNet

H2O.ai

Why MXNet?

R packages

dmlc mxnet

H2O.ai

TensorFlow

Python packages

Lasagne

TensorFlow

Caffe

theano

H2O.ai

Chainer

dmlc mxnet

Keras

Our pipeline

Get data → Process data → Train Neural Net(s)

Dataset

File: “default of credit card clients.csv”
From: UCI Machine Learning repository dataset. Link
Goal: Predict default of credit card clients
Size: 30K rows, 23 features (3 categorical, 20 numeric)

default payment next month: binary variable (Yes = 1, No = 0).
LIMIT_BAL: Amount of the given credit (NT dollar)
SEX : Gender (1 = male; 2 = female).
EDUCATION : Education (1 = graduate school; 2 = university; 3 = high school; 4 = others).
MARRIAGE : Marital status (1 = married; 2 = single; 3 = others).
AGE : Age (in years).
PAY_0 - PAY_6: History of past payment (from April to September, 2005) in months, so -1 = pay duly, 1 = payment delay for one month;. . .; 9 = payment delay for nine months and above
BILL_AMT1 - BILL_AMT6 : Amount of bill statement (from April to September, 2005)
PAY_AMT1 - PAY_AMT6 : Amount of previous payment (from April to September, 2005)

Our pipeline

Get data → Process data → Train Neural Net(s)

Data processing: vtreat to the rescue!

vtreat is an R data.frame processor/conditioner that prepares real-world data for predictive modeling in a statistically sound manner.

The library is designed to produce a 'data.frame' that is entirely numeric and takes common precautions to guard against the following real world data issues:

- Categorical variables with very many levels.
- Rare categorical levels.
- Novel categorical levels.
- Missing/invalid values NA, NaN, +/- Inf.
- Extreme values.
- Constant and near-constant variables.
- Need for estimated single-variable model effect sizes and significances.

In short: vtreat is AWESOME!!!

H2O.ai

Data processing: code

```r
dataset <- read.csv("default of credit card clients.csv")
#convert to categical
dataset$SEX <- as.factor(dataset$SEX)
dataset$EDUCATION <- as.factor(dataset$EDUCATION)
dataset$MARRIAGE <- as.factor(dataset$MARRIAGE)
#rename target variable
names(dataset)[names(dataset) == 'default.payment.next.month'] <- 'target'
#remove ID from data
dataset[,"ID"] <-NULL
```

Split data into train and test using caret package

```r
set.seed(3456)
testIndex <- createDataPartition(as.factor(dataset$target), times = 1, p = 0.2, list = F)
dTrain <- dataset[-testIndex,]
dTest <- dataset[testIndex,]
```

Use vtreat magic

```r
  yName <- 'target' # define target variable
  yTarget <- 1 # define level to be considered "success"
  varNames <- setdiff(names(dTrain),yName) # get variable names for vtreat

  treatmentsC <- designTreatmentsC(dTrain,varNames,yName,yTarget, verbose=FALSE)
  dTrainTreated <- prepare(treatmentsC,dTrain,pruneSig=c(),scale=TRUE)
  dTestTreated <- prepare(treatmentsC,dTest,pruneSig=c(),scale=TRUE)
```

Our pipeline

Get data → Process data → Train Neural Net(s)

H2O.ai

MXNet: mx.mlp

```
varNames <- setdiff(names(dTestTreated),yName) #get variables name for mx.mlp
mx.set.seed(1234) # set random seed, MXNet has its own!!!

model <- mx.mlp(data.matrix(dTrainTreated[,varNames]), # data
                dTrainTreated[,"target"], # target variable
                hidden_node=c(128,128,128,128), # number of hidden nodes per layer
                activation = "relu", # activation function,
                                     # can be a vector as well,
                                     # like c("relu","tanh", "tanh")
                out_node=1, # output node, 1 in our case (binary classification)
                out_activation="logistic", # can be "rmse" for regression
                                           # or "softmax" for multiclassification
                optimizer = "adam", # "sgd" by default,
                                    # can be "rmsprop", "adagrad", "adadelta"
                num.round=20, # number of epochs
                array.batch.size=256, # batch size
                learning.rate=0.01, # learning rate
                device = mx.gpu(1), # uses mx.cpu() by default
                eval.metric=mx.metric.mlogloss, # LogLoss as evaluation metric
                verbose = T, array.layout = "rowmajor")
```

MXNet: mx.mlp cont'd

MXNet doesn't have logloss metric in R, so I added one

```
mx.metric.mlogloss <- mx.metric.custom("mlogloss", function(label, pred){
  require(Metrics)
  return(logLoss(label, pred))
})
```

Now we can run a prediction using our trained model

```
preds = t(predict(model, data.matrix(dTestTreated[,varNames]),
                  array.layout = "rowmajor"))
```

As a baseline model I used random forest with 500 trees from **ranger** library

```
rf <- ranger(as.factor(target) ~., data = dTrain, probability = T)
rf_preds <- predict(rf, dTest)$predictions
```

Model	AUC	LogLoss
MXNet MLP	**0.7857**	0.4347
Random Forest	0.7768	**0.4316**

Building your own net

We start with defining the very same net we build using mx.mlp

```
data <- mx.symbol.Variable('data')
fc1  <- mx.symbol.FullyConnected(data = data, name = 'fc1', num_hidden = 128)
act1 <- mx.symbol.Activation(data = fc1, name = 'relu1', act_type = "relu")

fc2  <- mx.symbol.FullyConnected(data = act1, name = 'fc2', num_hidden = 128)
act2 <- mx.symbol.Activation(data = fc2, name = 'relu2', act_type = "relu")

fc3 <- mx.symbol.FullyConnected(data = act2, name = 'fc3', num_hidden = 128)
act3 <- mx.symbol.Activation(data = fc3, name = 'relu3', act_type = "relu")

fc4  <- mx.symbol.FullyConnected(data = act3, name = 'fc4', num_hidden = 128)
act4 <- mx.symbol.Activation(data = fc4, name = 'relu4', act_type = "relu")

fc5  <- mx.symbol.FullyConnected(data = act4, name = 'fc5', num_hidden = 1)
mlp  <- mx.symbol.LogisticRegressionOutput(data = fc5, name = 'logistic')
```

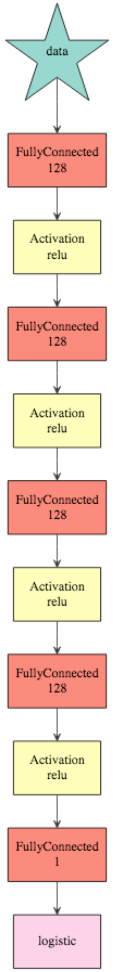
After that we can train it

```
model <- mx.model.FeedForward.create(
  X                 = data.matrix(dTrainTreated[,varNames]),
  y                 = dTrainTreated[,"target"],
  optimizer         = "adam",
  ctx               = mx.gpu(1),
  symbol            = mlp,
  eval.metric       = mx.metric.mlogloss,
  num.round         = 20,
  learning.rate     = 0.01,
  array.batch.size  = 256
)
```

Visualizing your own net

We also can visualize model's graph

```
graph.viz(model$symbol$as.json())
```

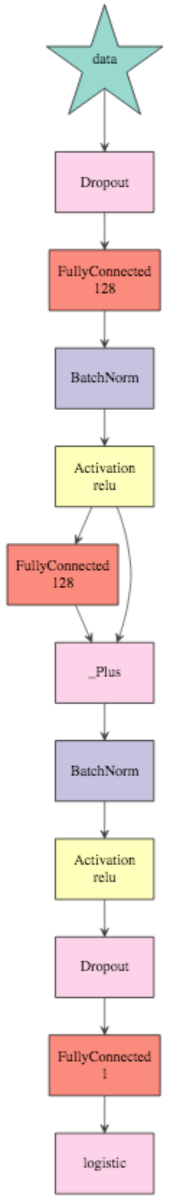


Your own net, things getting interesting

We can build neural net using some well-known techniques, like dropout, batch normalization and a trick from residual nets

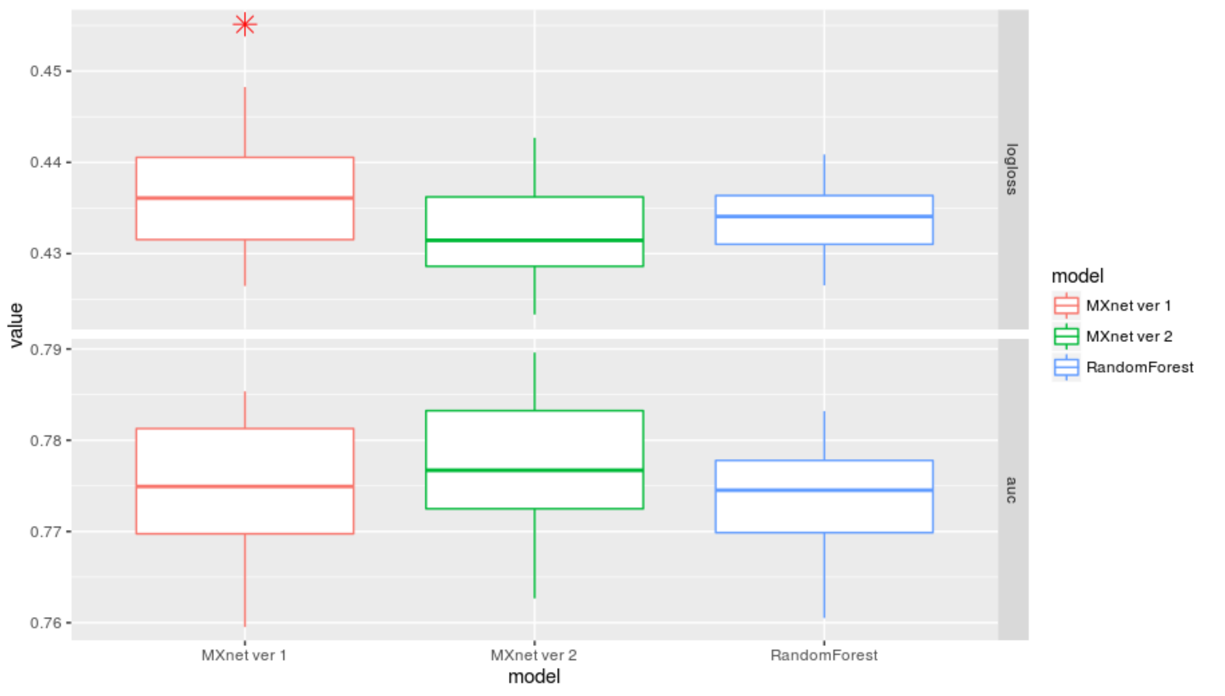
```
data <- mx.symbol.Variable('data')
dr0 = mx.symbol.Dropout(data = data, p = 0.1)
fc1  <- mx.symbol.FullyConnected(data = dr0, name = 'fc1', num_hidden = 128)
bn1 <- mx.symbol.BatchNorm(data = fc1, name = 'bn1')
act1 <- mx.symbol.Activation(data = bn1, name = 'relu1', act_type = "relu")

fc2 <- mx.symbol.FullyConnected(data = act1, name = 'fc2', num_hidden = 128)
resid <- act1 + fc2
bn2 <- mx.symbol.BatchNorm(data = resid, name = 'bn2')
act2 <- mx.symbol.Activation(data = bn2, name = 'relu2', act_type = "relu")

dr1 = mx.symbol.Dropout(data = act2, p = 0.5)
fc3  <- mx.symbol.FullyConnected(data = dr1, name = 'fc3', num_hidden = 1)
mlp  <- mx.symbol.LogisticRegressionOutput(data = fc3, name = 'logistic')
```

Visualization



Models performance

Model	AUC	LogLoss
MXNet MLP	0.7749 +/- 0.0089	0.4369 +/- 0.0087
MXNet Net #2	**0.7774 +/- 0.0091**	**0.4322 +/- 0.0067**
Random Forest	0.7729 +/- 0.0078	0.4340 +/-0.0053



H2O.ai

Thank you!

Q&A

This presentation code: https://github.com/IzuiT/H2O_Dallas_MXNet