Deep learning in R using MXNet

Qiang Kou

qkou@umail.iu.edu

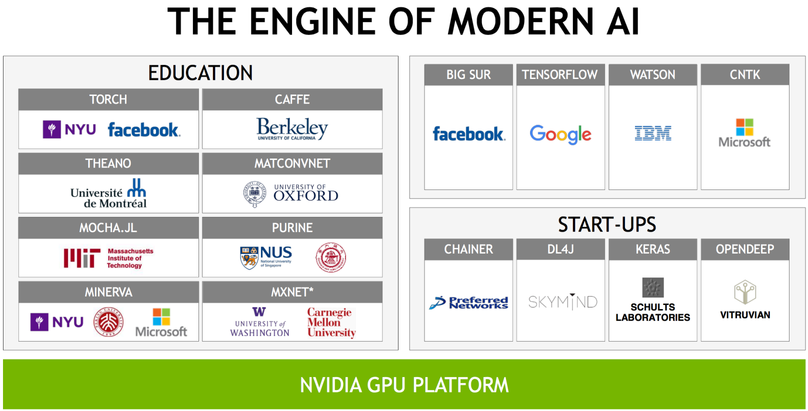
R/Finance 2016, Chicago

INDIANA UNIVERSITY

- Qiang Kou, KK, 寇强
- PhD student in bioinformatics from Indiana University
- Rcpp team member

What is MXNet?

Deep learning platforms



THE ENGINE OF MODERN AI

EDUCATION

- TORCH: NYU, facebook
- CAFFE: Berkeley University of California
- THEANO: Université de Montréal
- MATCONVNET: University of Oxford
- MOCHA.JL: Massachusetts Institute of Technology
- PURINE: NUS National University of Singapore
- MINERVA: NYU, Microsoft
- MXNET*: University of Washington, Carnegie Mellon University

- BIG SUR: facebook
- TENSORFLOW: Google
- WATSON: IBM
- CNTK: Microsoft

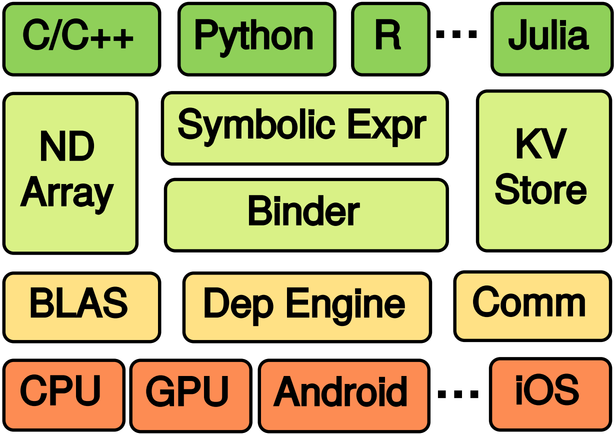
START-UPS

- CHAINER: Preferred Networks
- DL4J: SKYMIND
- KERAS: SCHULTS LABORATORIES
- OPENDEEP: VITRUVIAN

NVIDIA GPU PLATFORM

http://www.slideshare.net/NVIDIA/nvidia-ces-2016-press-conference

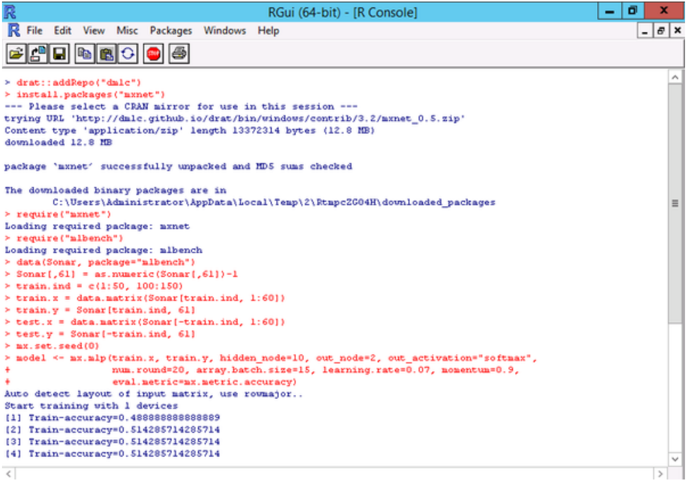
MXNet



MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems

Why MXNet?

Windows support


RGui (64-bit) - [R Console]
File Edit View Misc Packages Windows Help

```
> drat::addRepo("dmlc")
> install.packages("mxnet")
--- Please select a CRAN mirror for use in this session ---
trying URL 'http://dmlc.github.io/drat/bin/windows/contrib/3.2/mxnet_0.5.zip'
Content type 'application/zip' length 13372314 bytes (12.8 MB)
downloaded 12.8 MB

package 'mxnet' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\Administrator\AppData\Local\Temp\2\RtmpcZG04H\downloaded_packages
> require("mxnet")
Loading required package: mxnet
> require("mlbench")
Loading required package: mlbench
> data(Sonar, package="mlbench")
> Sonar[,61] = as.numeric(Sonar[,61])-1
> train.ind = c(1:50, 100:150)
> train.x = data.matrix(Sonar[train.ind, 1:60])
> train.y = Sonar[train.ind, 61]
> test.x = data.matrix(Sonar[-train.ind, 1:60])
> test.y = Sonar[-train.ind, 61]
> mx.set.seed(0)
> model <- mx.mlp(train.x, train.y, hidden_node=10, out_node=2, out_activation="softmax",
+                  num.round=20, array.batch.size=15, learning.rate=0.07, momentum=0.9,
+                  eval.metric=mx.metric.accuracy)
Auto detect layout of input matrix, use rowmajor..
Start training with 1 devices
[1] Train-accuracy=0.488888888888889
[2] Train-accuracy=0.514285714285714
[3] Train-accuracy=0.514285714285714
[4] Train-accuracy=0.514285714285714
```

Installation

DRAT repo

For Windows and Mac users

```
install.packages("drat", repos="https://cran.rstudio.com")
drat:::addRepo("dmlc")
install.packages("mxnet")
```

MNIST demo

MNIST demo

```r
library(mxnet)
train <- read.csv("train.csv", header=TRUE)
test <- read.csv("test.csv", header=TRUE)
train <- data.matrix(train)
test <- data.matrix(test)
train.x <- train[,-1]
train.y <- train[,1]
train.x <- t(train.x/255)
test <- t(test/255)
```

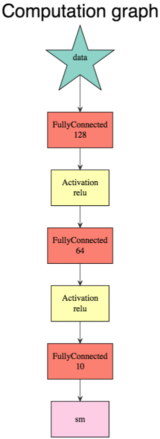
The dataset: https://www.kaggle.com/c/digit-recognizer/data

MNIST demo

```
data <- mx.symbol.Variable("data")
fc1 <- mx.symbol.FullyConnected(data, name="fc1", num_hidden=128)
act1 <- mx.symbol.Activation(fc1, name="relu1", act_type="relu")
fc2 <- mx.symbol.FullyConnected(act1, name="fc2", num_hidden=64)
act2 <- mx.symbol.Activation(fc2, name="relu2", act_type="relu")
fc3 <- mx.symbol.FullyConnected(act2, name="fc3", num_hidden=10)
softmax <- mx.symbol.SoftmaxOutput(fc3, name="sm")
```

MNIST demo

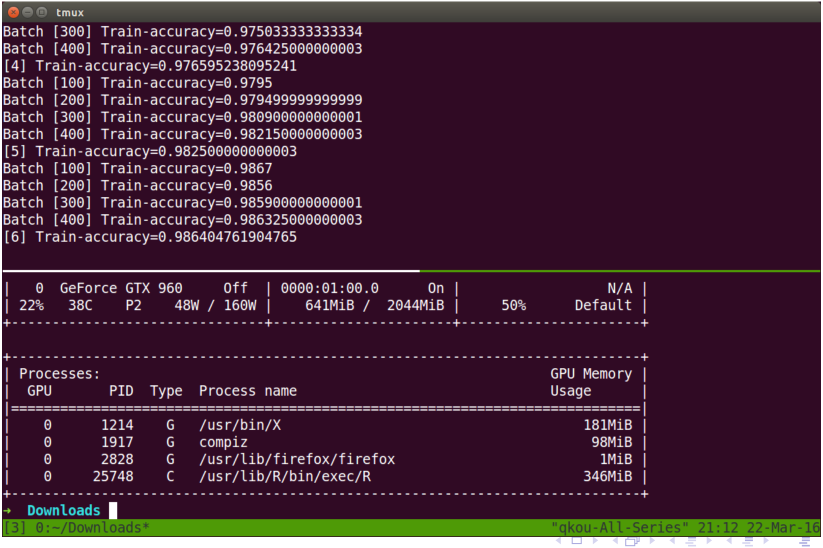
```
graph.viz(softmax$as.json())
```



MNIST demo

```
mx.set.seed(0)
model <- mx.model.FeedForward.create(softmax, X=train.x,
              y=train.y, ctx=mx.gpu(), num.round=10,
              array.batch.size=100,
              learning.rate=0.07, momentum=0.9,
              eval.metric=mx.metric.accuracy,
              initializer=mx.init.uniform(0.07),
              batch.end.callback
                   = mx.callback.log.train.metric(100))
```

MNIST demo



```
Batch [300] Train-accuracy=0.975033333333334
Batch [400] Train-accuracy=0.976425000000003
[4] Train-accuracy=0.976595238095241
Batch [100] Train-accuracy=0.9795
Batch [200] Train-accuracy=0.979499999999999
Batch [300] Train-accuracy=0.980900000000001
Batch [400] Train-accuracy=0.982150000000003
[5] Train-accuracy=0.982500000000003
Batch [100] Train-accuracy=0.9867
Batch [200] Train-accuracy=0.9856
Batch [300] Train-accuracy=0.985900000000001
Batch [400] Train-accuracy=0.986325000000003
[6] Train-accuracy=0.986404761904765

|   0  GeForce GTX 960     Off  | 0000:01:00.0      On |                  N/A |
| 22%   38C    P2    48W / 160W |    641MiB /  2044MiB |     50%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID  Type  Process name                               Usage      |
|=============================================================================|
|    0      1214    G   /usr/bin/X                                     181MiB |
|    0      1917    G   compiz                                          98MiB |
|    0      2828    G   /usr/lib/firefox/firefox                         1MiB |
|    0     25748    C   /usr/lib/R/bin/exec/R                          346MiB |
+-----------------------------------------------------------------------------+
→  Downloads
[3] 0:~/Downloads*                                   "qkou-All-Series" 21:12 22-Mar-16
```

Convolutional Neural Network

```r
# input
data <- mx.symbol.Variable('data')
# first conv
conv1 <- mx.symbol.Convolution(data=data, kernel=c(5,5), num_filter=20)
tanh1 <- mx.symbol.Activation(data=conv1, act_type="tanh")
pool1 <- mx.symbol.Pooling(data=tanh1, pool_type="max", kernel=c(2,2), stride=c(2,2))
# second conv
conv2 <- mx.symbol.Convolution(data=pool1, kernel=c(5,5), num_filter=50)
tanh2 <- mx.symbol.Activation(data=conv2, act_type="tanh")
pool2 <- mx.symbol.Pooling(data=tanh2, pool_type="max", kernel=c(2,2), stride=c(2,2))
# first fullc
flatten <- mx.symbol.Flatten(data=pool2)
fc1 <- mx.symbol.FullyConnected(data=flatten, num_hidden=500)
tanh3 <- mx.symbol.Activation(data=fc1, act_type="tanh")
# second fullc
fc2 <- mx.symbol.FullyConnected(data=tanh3, num_hidden=10)
# loss
lenet <- mx.symbol.SoftmaxOutput(data=fc2)
```

LSTM

```
# lstm cell symbol
lstm <- function(num.hidden, indata, prev.state, param, seqidx, layeridx, dropout=0) {
  if (dropout > 0)
    indata <- mx.symbol.Dropout(data=indata, p=dropout)
  i2h <- mx.symbol.FullyConnected(data=indata,
                                  weight=param$i2h.weight,
                                  bias=param$i2h.bias,
                                  num.hidden=num.hidden * 4,
                                  name=paste0("t", seqidx, ".l", layeridx, ".i2h"))
  h2h <- mx.symbol.FullyConnected(data=prev.state$h,
                                  weight=param$h2h.weight,
                                  bias=param$h2h.bias,
                                  num.hidden=num.hidden * 4,
                                  name=paste0("t", seqidx, ".l", layeridx, ".h2h"))
  gates <- i2h + h2h
  slice.gates <- mx.symbol.SliceChannel(gates, num.outputs=4,
                                        name=paste0("t", seqidx, ".l", layeridx, ".slice"))

  in.gate <- mx.symbol.Activation(slice.gates[[1]], act.type="sigmoid")
  in.transform <- mx.symbol.Activation(slice.gates[[2]], act.type="tanh")
  forget.gate <- mx.symbol.Activation(slice.gates[[3]], act.type="sigmoid")
  out.gate <- mx.symbol.Activation(slice.gates[[4]], act.type="sigmoid")
  next.c <- (forget.gate * prev.state$c) + (in.gate * in.transform)
  next.h <- out.gate * mx.symbol.Activation(next.c, act.type="tanh")

  return (list(c=next.c, h=next.h))
}
```

A shiny app

A shiny app

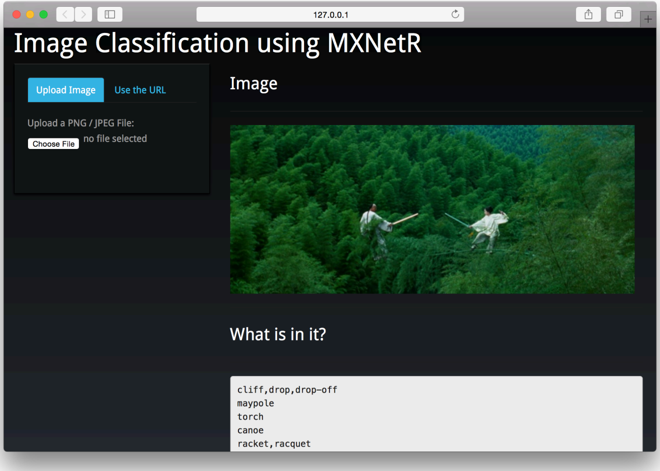


Image Classification using MXNetR

Upload a PNG / JPEG File:

no file selected

Image

What is in it?

```
cliff,drop,drop-off
maypole
torch
canoe
racket,racquet
```

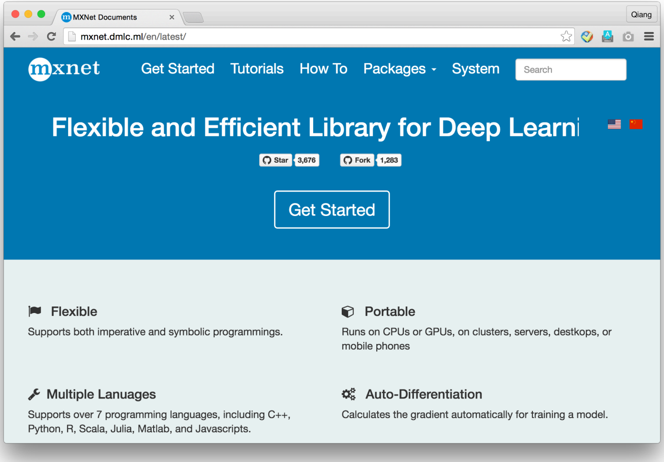
A shiny app

```r
model <- mx.model.load("Inception/Inception_BN", iteration = 39)
synsets <- readLines("Inception/synset.txt")
mean.img <-
  as.array(mx.nd.load("Inception/mean_224.nd")[["mean_img"]])
im <- load.image(src)
normed <- preproc.image(im, mean.img)
prob <- predict(model, X = normed)
```

Acknowledgment

Bai du 百度
Carnegie Mellon University
intel
NYU
Microsoft
UNIVERSITY OF ALBERTA
W
UNIVERSITY of WASHINGTON
McGill

Go to http://mxnet.dmlc.ml/ to get started!



Flexible and Efficient Library for Deep Learni

Flexible
Supports both imperative and symbolic programmings.

Portable
Runs on CPUs or GPUs, on clusters, servers, destkops, or mobile phones

Multiple Lanuages
Supports over 7 programming languages, including C++, Python, R, Scala, Julia, Matlab, and Javascripts.

Auto-Differentiation
Calculates the gradient automatically for training a model.

Thank you for the time!