Mas Rapido! Speedy Automated Machine Learning Models with Parallelized Caret

Machine Learning for Economists

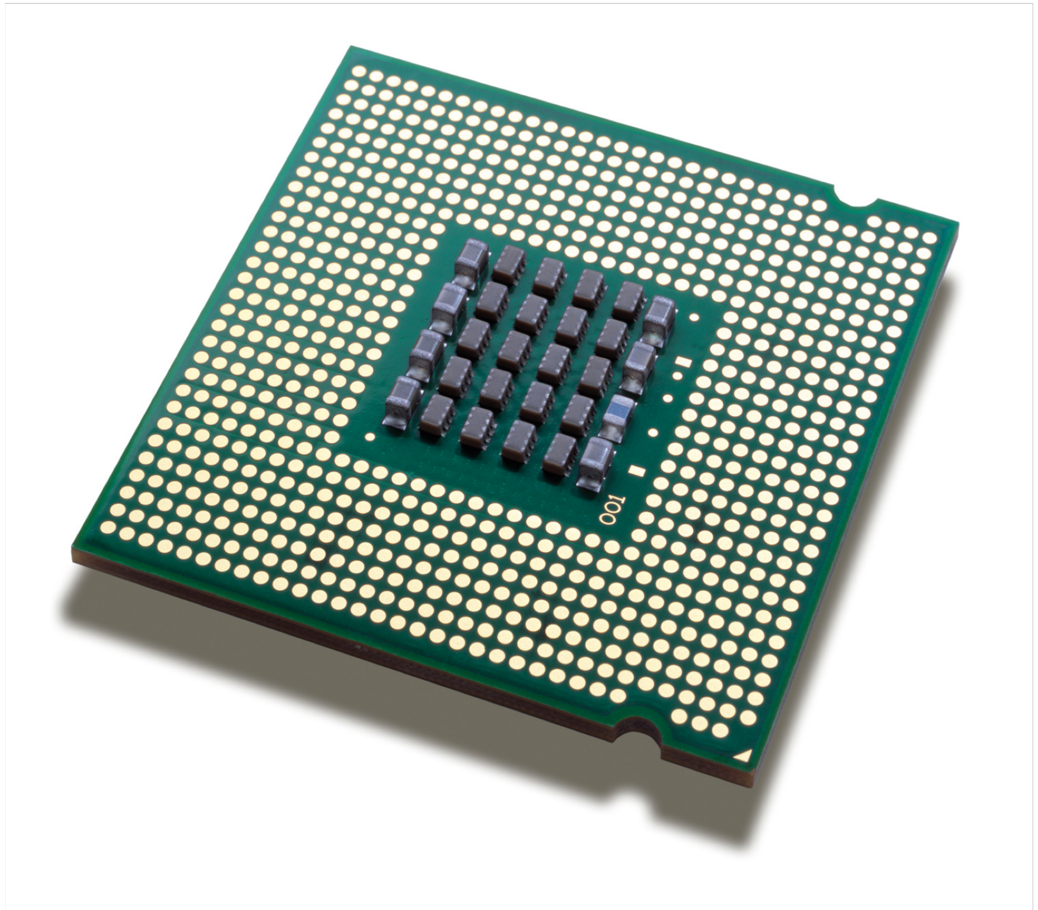
Jonathan Hersh

Plan for Today

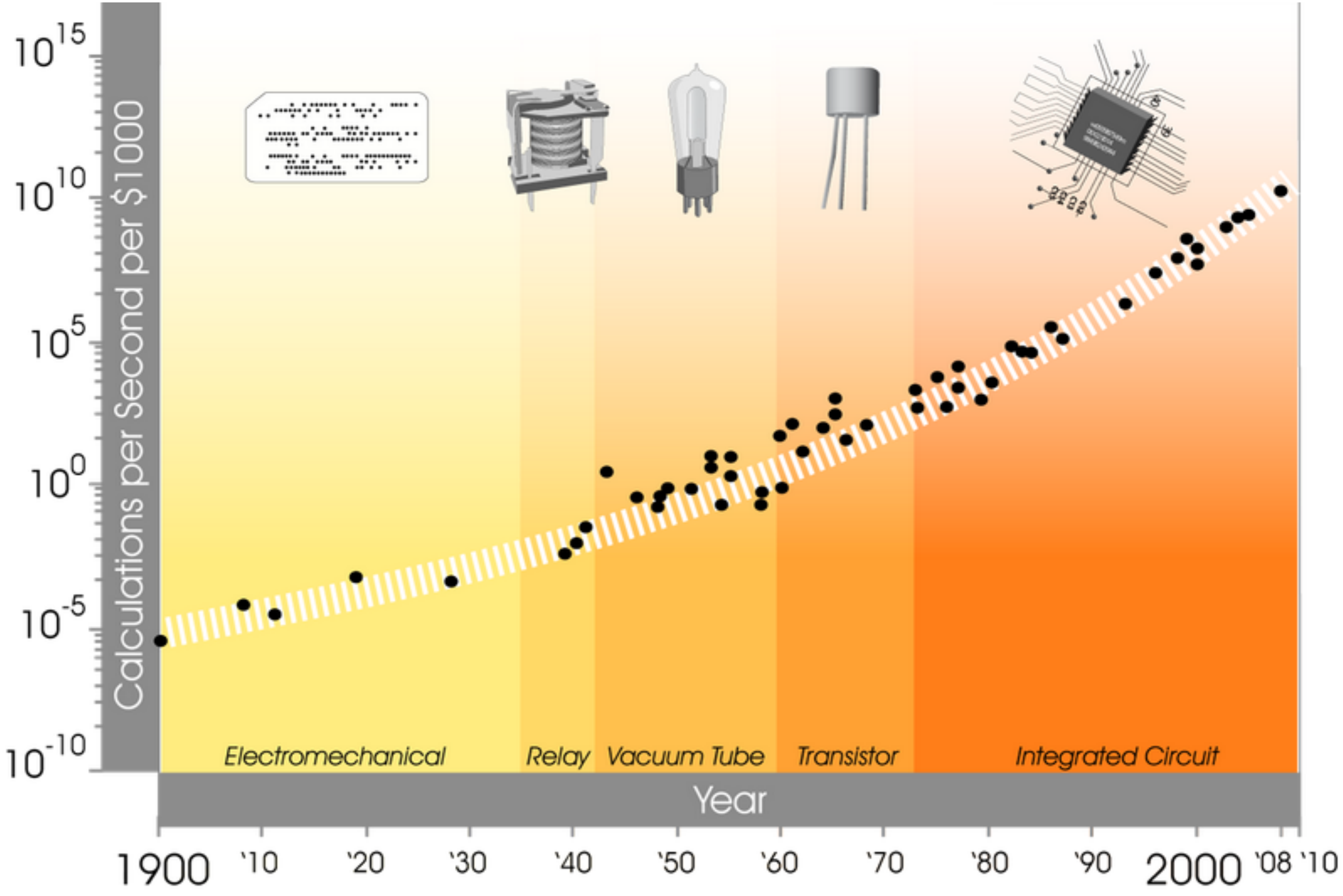
- Parallelizing in R
  - What is a parallel process?
  - Why parallelize?
  - How to do it? R package: doParallel, foreach
- R package caret
  - Automated machine learning software

What is parallelization?

- The CPU, or central processing unit, is the bit in the computer that actually calculates things.
- These are fast
  - Intel Core i7 7500U ~ 50 billion instructions per second
  - 50,000 MIPS
- GFLOPS (1 billion floating points per second)
  - Floating point = 7 digits (32 bit)
  - “double” 16 digits (64 bit)



Processor calculation speed over time



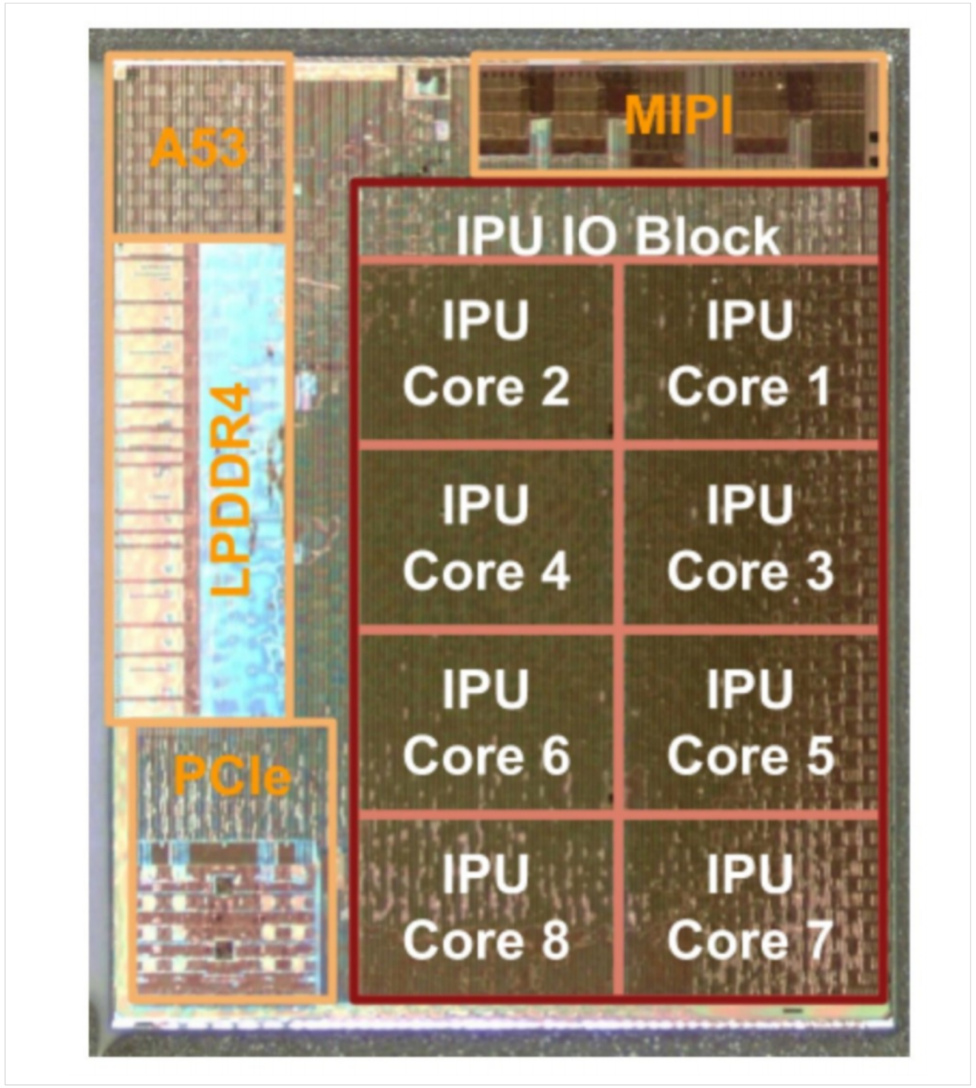
What’s the problem?

- Deep learning and machine learning often need many more compute cycles to train

Model	Type	Parameters	Model Size (MB)	GFLOPs (forward pass)
ResNet152	CNN	60,344,387	230MB	11.3

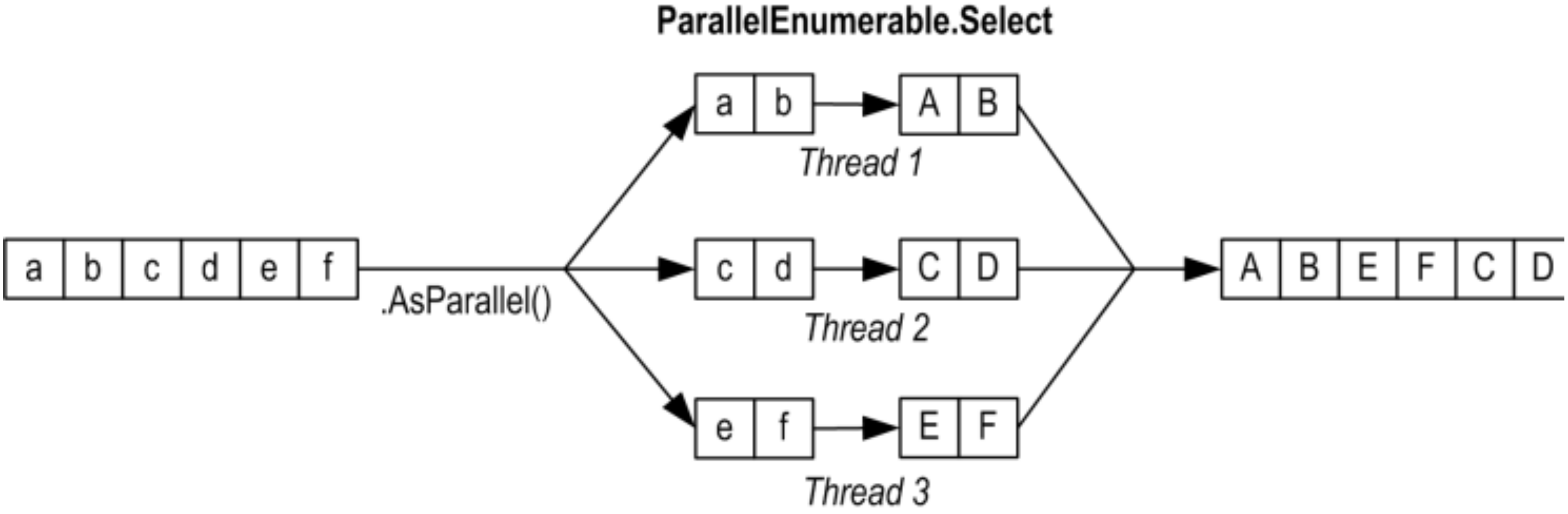
- Training data: 14M images (ImageNet)
- FLOPs per pass of data: 3*11.3*10^9 * 14 * 10^6
- Training time for 1 epoch 140 TFLOPS (trillion floating points)

Multiple Cores to the Rescue



- Core i7 4770K 4 cores @ 3.5 GHz
  - 182 GFLOPS
  - 45.5 GFLOPS per core
- Core i7 5960X: 8 Core @ 3GHz
  - 354 GFLOPS
  - 44.25 per core

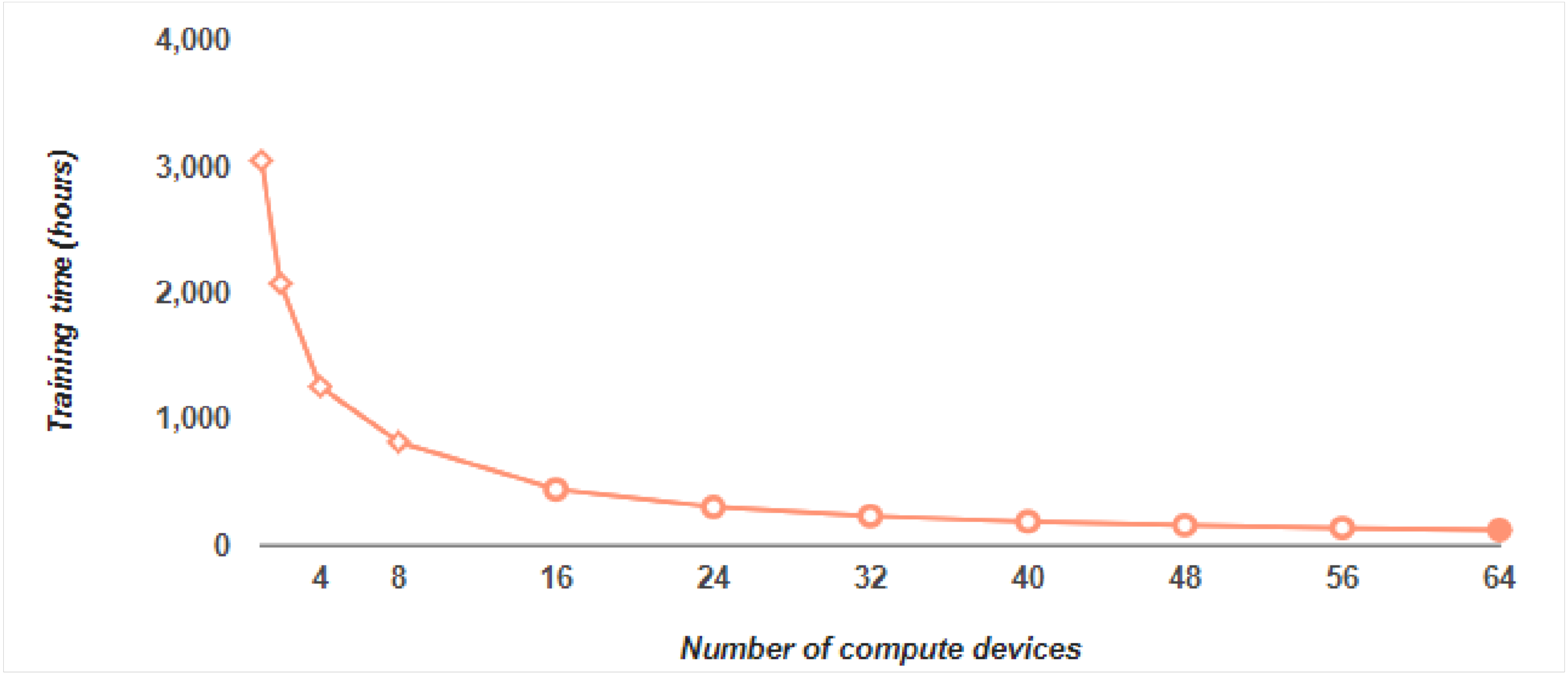
Idea of Parallelization



GPU – Graphics Processor Units

	TITAN X PASCAL
CUDA cores	3584
Boost Clock	1.53 GHZ
Memory	12GB G5X
Memory Speed (Gb/s)	10
Memory Bandwidth (GB/s)	480
Texture Rate (GT/s)	343
TFLOPS (INT8)	44
TFLOPS (FP32)	11

Multiple Cores to the Rescue



Parallelizing in R w/ doParallel

doParallel: Foreach Parallel Adaptor for the 'parallel' Package

Provides a parallel backend for the %dopar% function using the parallel package.

Version:	1.0.14
Depends:	R (≥ 2.14.0), foreach (≥ 1.2.0), iterators (≥ 1.0.0), parallel, utils
Suggests:	caret, mlbench, rpart, RUnit
Enhances:	compiler
Published:	2018-09-24
Author:	Rich Calaway [cre], Microsoft Corporation [aut, cph], Steve Weston [aut], Dan Tenenbaum [ctb]
Maintainer:	Rich Calaway <richcala at microsoft.com>
License:	GPL-2
NeedsCompilation:	no
Materials:	NEWS
CRAN checks:	doParallel results

Downloads:

Reference manual:	doParallel.pdf
Vignettes:	Getting Started with doParallel and foreach
Package source:	doParallel_1.0.14.tar.gz
Windows binaries:	r-devel: doParallel_1.0.14.zip, r-release: doParallel_1.0.14.zip, r-oldrel: doParallel_1.0.14.zip
OS X binaries:	r-release: doParallel_1.0.14.tgz, r-oldrel: doParallel_1.0.14.tgz
Old sources:	doParallel archive

Reverse dependencies:

Detecting your cores using detectCores()

```r
## doParallel package
library(doParallel)
numCores <- detectCores()
numCores
```

```
> ## doParallel package
> library(doParallel)
> numCores <- detectCores()
> numCores
[1] 4
```

Set number of cores with registerDoParallel()

```r
# register your cores to doParallel
# note on windows machines can only set = 1 :(
registerDoParallel(cores = numCores)
```

registerDoParallel {doParallel} R Documentation

registerDoParallel

Description

The `registerDoParallel` function is used to register the parallel backend with the `foreach` package.

Usage

foreach and %dopar% function

```r
# regular for loop
for(i in 1:3){
  print(sqrt(i))
}
```

```
> for(i in 1:3){
+   print(sqrt(i))
+ }
[1] 1
[1] 1.414214
[1] 1.732051
```

```r
# paralellized for loop!
foreach(i=1:3) %dopar% sqrt(i)
```

```
> foreach(i=1:3) %dopar% sqrt(i)
[[1]]
[1] 1

[[2]]
[1] 1.414214

[[3]]
[1] 1.732051
```

A real example: Let's Parallelize a Prime Number Finder Function

```r
# a real example with bootstrapping
getPrimeNumbers <- function(n) {
  n <- as.integer(n)
  if(n > 1e6) stop("n too large")
  primes <- rep(TRUE, n)
  primes[1] <- FALSE
  last.prime <- 2L
  for(i in last.prime:floor(sqrt(n)))
  {
    primes[seq.int(2L*last.prime, n, last.prime)] <- FALSE
    last.prime <- last.prime + min(which(primes[(last.prime+1):n]))
  }
  which(primes)
}
```

Find primes up to 100

```
> getPrimeNumbers(100)
 [1]  2  3  5  7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83
[24] 89 97
```

Let's do this 10,000 times using a boring for loop

```r
index <- 10:10000
library(tictoc)
tic()
results <- c()
for(i in index){
  results[[i]] <- getPrimeNumbers(i)
}
toc()
```

```
> tic()
> results <- c()
> for(i in index){
+   results[[i]] <- getPrimeNumbers(i)
+ }
> toc()
28.942 sec elapsed
```

Let’s do this 10,000 times the smart way

```r
# let's try now with doParallel
library(doParallel)
numCores <- detectCores()
registerDoParallel(cores = numCores)
tic()
results <- foreach(i = 10:1000) %dopar% getPrimeNumbers(i)
toc()
```

```
> toc()
0.555 sec elapsed
```

What if we want to store the results from each for loop?

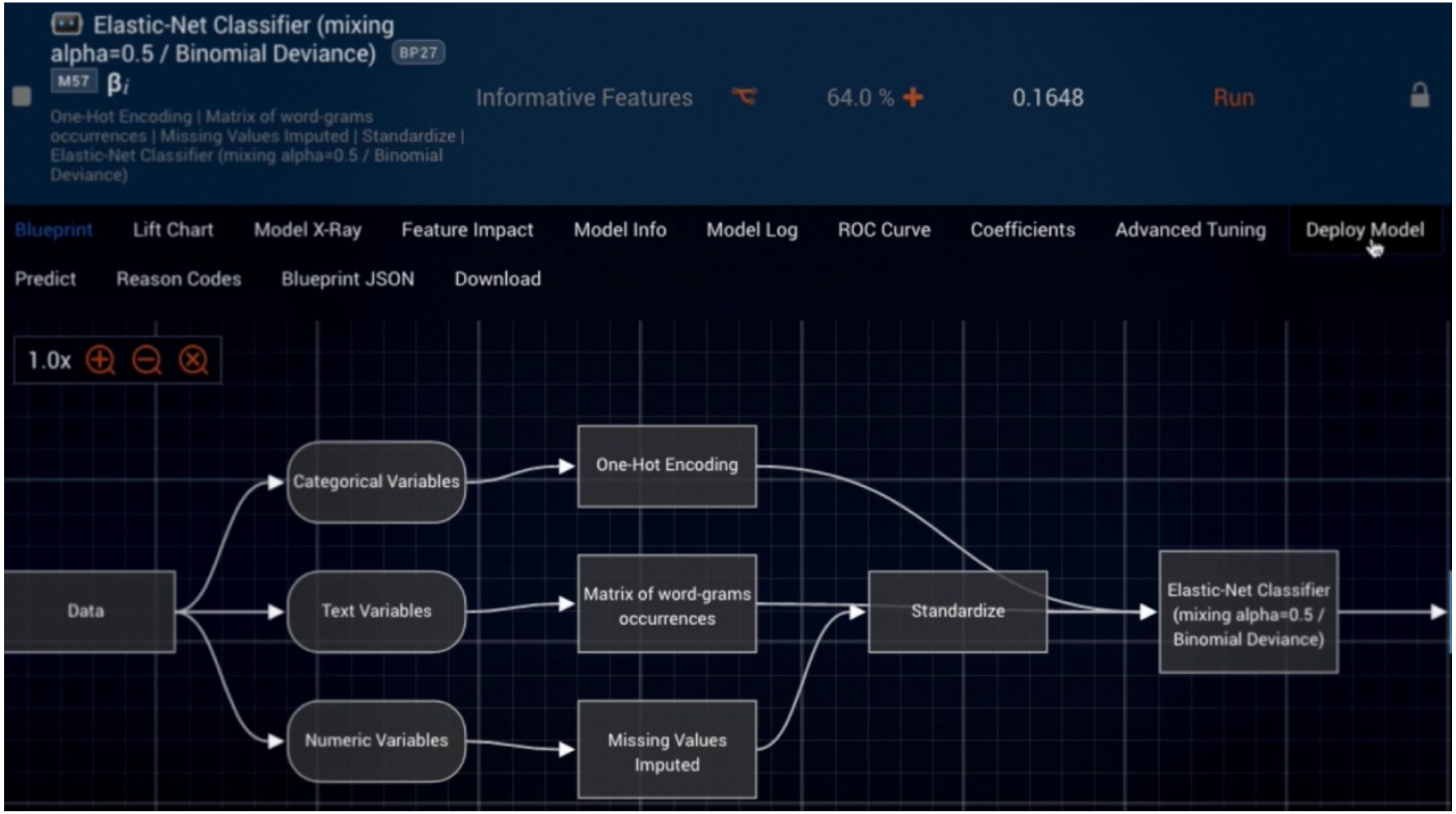
```r
results <- foreach(1:100, .combine = data.frame) %dopar% {
  # do something
}
```

On your own…generate 1000 random numbers of length 1000

On your own…generate 1000 random numbers of length 1000

```
tic()
results <- foreach(i=1:1000, .combine = data.frame) %dopar% {
  data.frame(rands = rnorm(1000))
}
toc()
```

Automated machine learning software: DataRobot



AutoML: TensorFlow & Keras

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

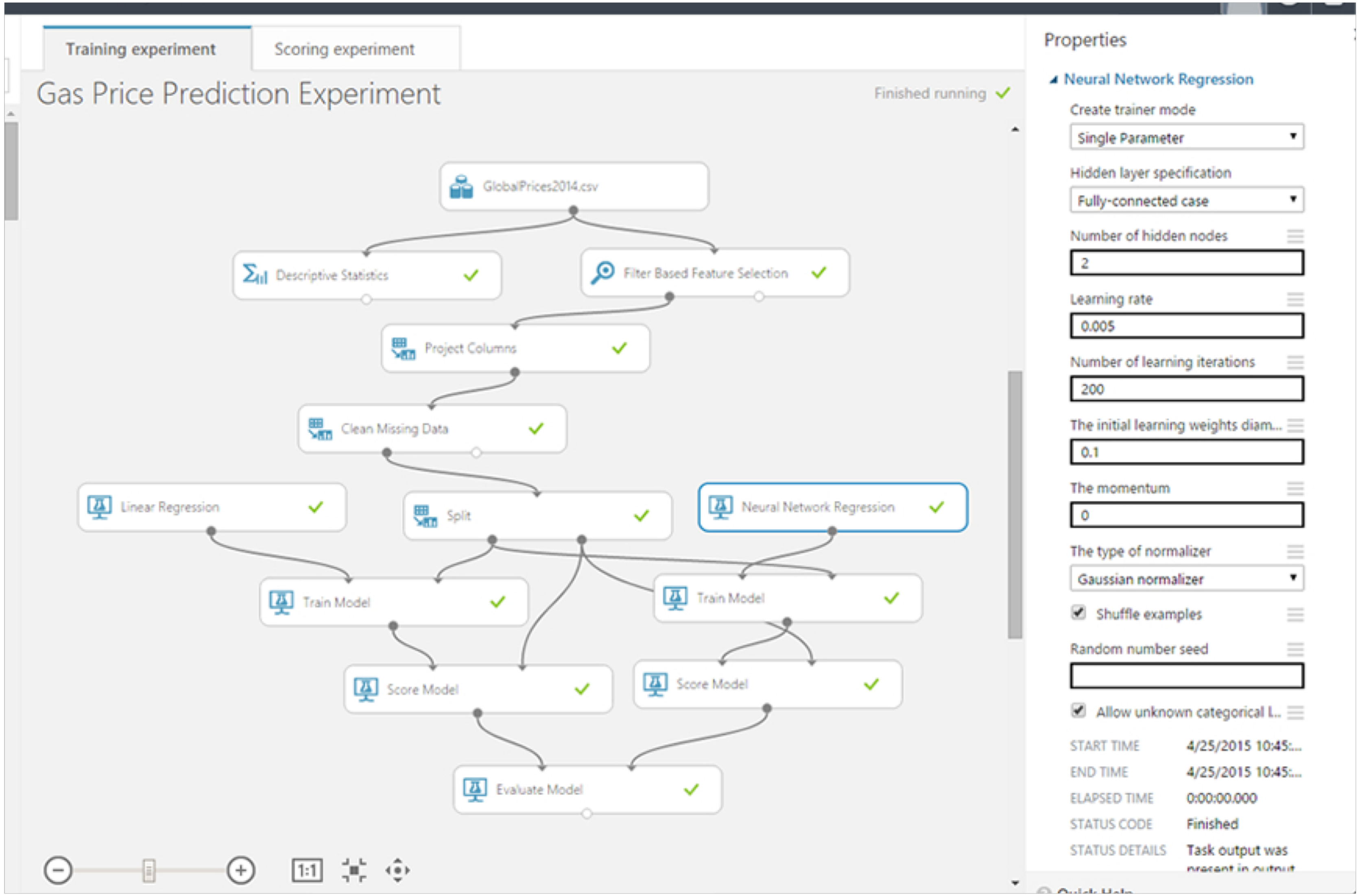
K Keras

R interface to Keras

Keras is a high-level neural networks API developed with a focus on enabling fast experimentation. *Being able to go from idea to result with the least possible delay is key to doing good research.* Keras has the following key features:

- Allows the same code to run on CPU or on GPU, seamlessly.
- User-friendly API which makes it easy to quickly prototype deep learning models.
- Built-in support for convolutional networks (for computer vision), recurrent networks (for sequence processing), and any combination of both.
- Supports arbitrary network architectures: multi-input or multi-output models, layer sharing, model sharing, etc. This means that Keras is appropriate for building essentially any deep learning model, from a memory network to a neural Turing machine.
- Is capable of running on top of multiple back-ends including TensorFlow, CNTK, or Theano.

Auto ML: Azure Studio



A Short Introduction to the caret Package

The **caret** package (short for Classification And REgression Training) contains functions to streamline the model training process for complex regression and classification problems. The package utilizes a number of R packages but tries not to load them all at package start-up (by removing formal package dependencies, the package startup time can be greatly decreased). The package "suggests" field includes 30 packages. **caret** loads packages as needed and assumes that they are installed. If a modeling package is missing, there is a prompt to install it.

Install **caret** using

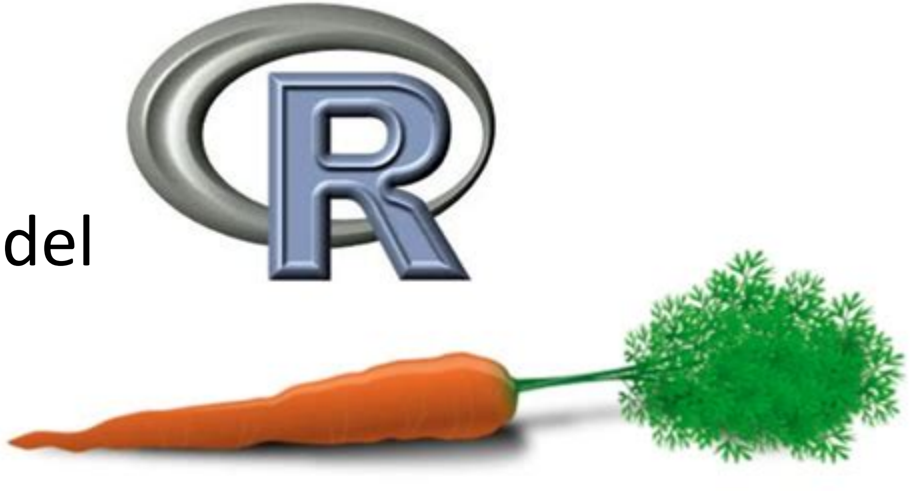
```
install.packages("caret", dependencies = c("Depends", "Suggests"))
```

to ensure that all the needed packages are installed.

The **main help pages** for the package are at https://topepo.github.io/caret/ Here, there are extended examples and a large amount of information that previously found in the package vignettes.

Why caret?

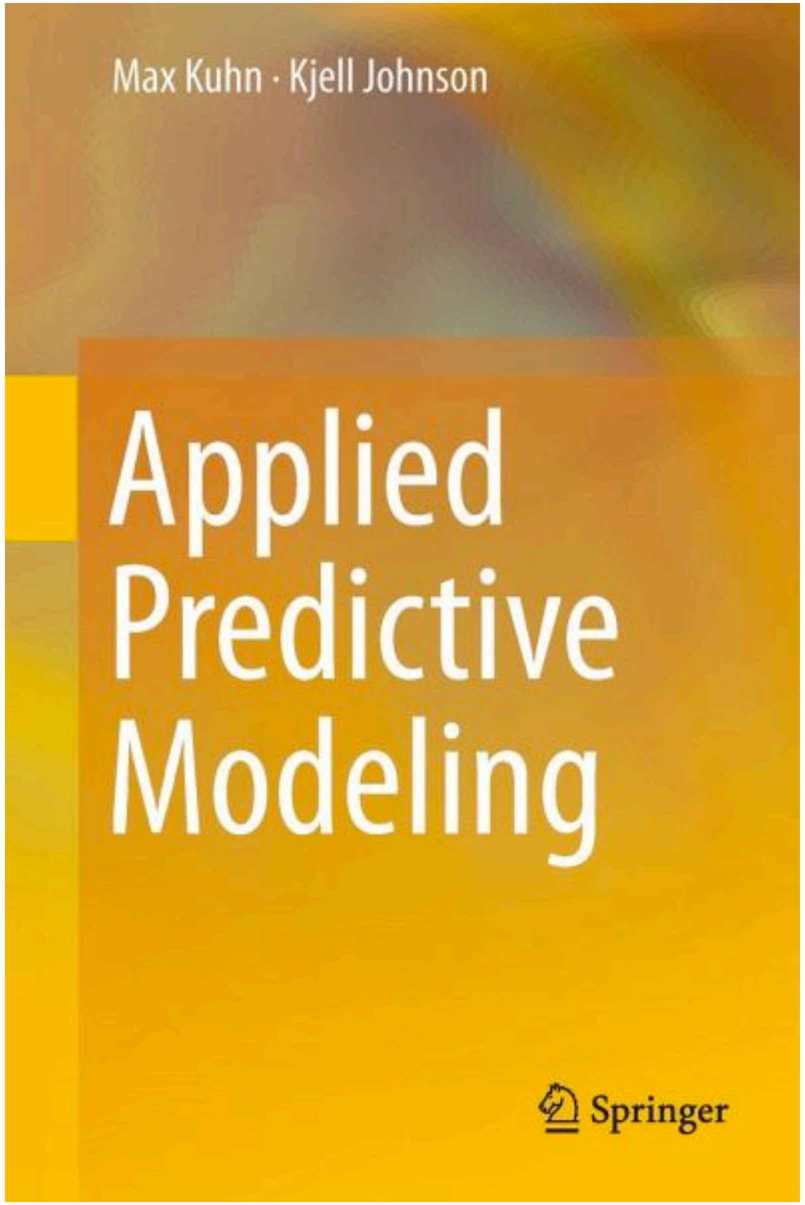
- In short, caret streamlines the model building process and
- Provides a unified formula interface
- Evaluates, using resampling, the effect of model parameters on validation performance
- Chooses optimal parameters on OOS performance
- HUGE list of models available



Caret Has Tools For

- Data Splitting
- Data Pre-Processing (center, scaling)
- Feature selection
- Model tuning and resampling
- Variable importance and other post-estimation diagnostics

Comprehensive guide at:
https://topepo.github.io/caret/index.html



Partial Model List

- Glmnet
- K-nearest neighbors
- Logistic regression
- Neural networks
- Random forests
- Parallel random forests
- Partial Least Squares
- Ridge Regression
- SVMs
- Weighted Least Squares
- Extreme Gradient Boosted Trees
- Bayesian Additive Regression Trees (BART)
- Bayesian GLM
- Multivariate Adaptive Regression Splines (MARS)
- Partial Least Squares
- Least Squares Support Vector Machine with Radial Basis Function Kernel
- Fuzzy Rules Using the Structural Learning Algorithm on Vague Environment

Caret Model Building Process

```
Define sets of model parameter values to evaluate
for each parameter set do
    for each resampling iteration do
        Hold–out specific samples
        [Optional] Pre–process the data
        Fit the model on the remainder
        Predict the hold–out samples
    end
    Calculate the average performance across hold–out predictions
end
Determine the optimal parameter set
Fit the final model to all the training data using the optimal parameter set
```

Key functions: train() and trainControl()

trainControl()

- Specifies range of parameters over which you want to estimate your model, nuances of training process

Train()

- Function that estimates your model against your data given a formula using options specified by trainControl()

Example Wage Data

Griliches {Ecdat}

Wage Datas

Description

a cross-section from 1980

number of observations : 758

observation : individuals

country : United States

Usage

```
data(Griliches)
```

rns
: residency in the southern states (first observation) ?

rns80
: same variable for 1980

mrt
: married (first observation) ?

mrt80
: same variable for 1980

smsa
: residency in metropolitan areas (first observation) ?

smsa80
: same variable for 1980

med
: mother's education in years

iq
: IQ score

kww
: score on the “knowledge of the world of work” test

year
: year of the observation

age
: age (first observation)

age80
: same variable for 1980

school
: completed years of schooling (firs

school80
: same variable for 1980

expr
: experience in years (first observat

expr80
: same variable for 1980

tenure
: tenure in years (first observation)

tenure80
: same variable for 1980

lw
: log wage (first observation)

lw80
: same variable for 1980

Download Wage DF and create a formula to predict log wage in 1980

```
install.packages('Ecdat')
data(Griliches, package = "Ecdat")
wages <- Griliches
wageFormula <- lw80 ~ age80 + school80 + expr80 + iq + rns80 + mrt80 +
  smsa80 + tenure80 + med + kww
```

Example trainControl()

```
# example trainControl function
ctrl1 <- trianControl(method = "repeatedcv", repeats = 5,
                                allowParallel = TRUE)
```

Example trainControl()

```
# example trainControl function
ctrl1 <- trianControl(method = "repeatedcv", repeats = 5,
                      allowParallel = TRUE)
```

train() function – parameters needed? – modelLookup('model')

```
> modelLookup('rf')
  model parameter                         label forReg forClass probModel
1    rf      mtry #Randomly Selected Predictors   TRUE     TRUE      TRUE
```

```
> modelLookup('xgbTree')
    model        parameter                         label forReg forClass probModel
1 xgbTree          nrounds           # Boosting Iterations   TRUE     TRUE      TRUE
2 xgbTree        max_depth                  Max Tree Depth   TRUE     TRUE      TRUE
3 xgbTree              eta                       Shrinkage   TRUE     TRUE      TRUE
4 xgbTree            gamma          Minimum Loss Reduction   TRUE     TRUE      TRUE
5 xgbTree colsample_bytree      Subsample Ratio of Columns   TRUE     TRUE      TRUE
6 xgbTree min_child_weight Minimum Sum of Instance Weight   TRUE     TRUE      TRUE
7 xgbTree        subsample           Subsample Percentage   TRUE     TRUE      TRUE
```

See list of models
https://topepo.github.io/caret/available-models.html

Training Grid

```r
# grid of mtry values to try
rfGrid <- expand.grid(mtry = seq(1, 10, 1))
```

```
> rfGrid
   mtry
1     1
2     2
3     3
4     4
5     5
6     6
7     7
8     8
9     9
10   10
```

Finally..estimate train() model

```r
# grid of mtry values to try
rfGrid <- expand.grid(mtry = seq(1, 10, 1))
```

```
> rfGrid
   mtry
1     1
2     2
3     3
4     4
5     5
6     6
7     7
8     8
9     9
10   10
```

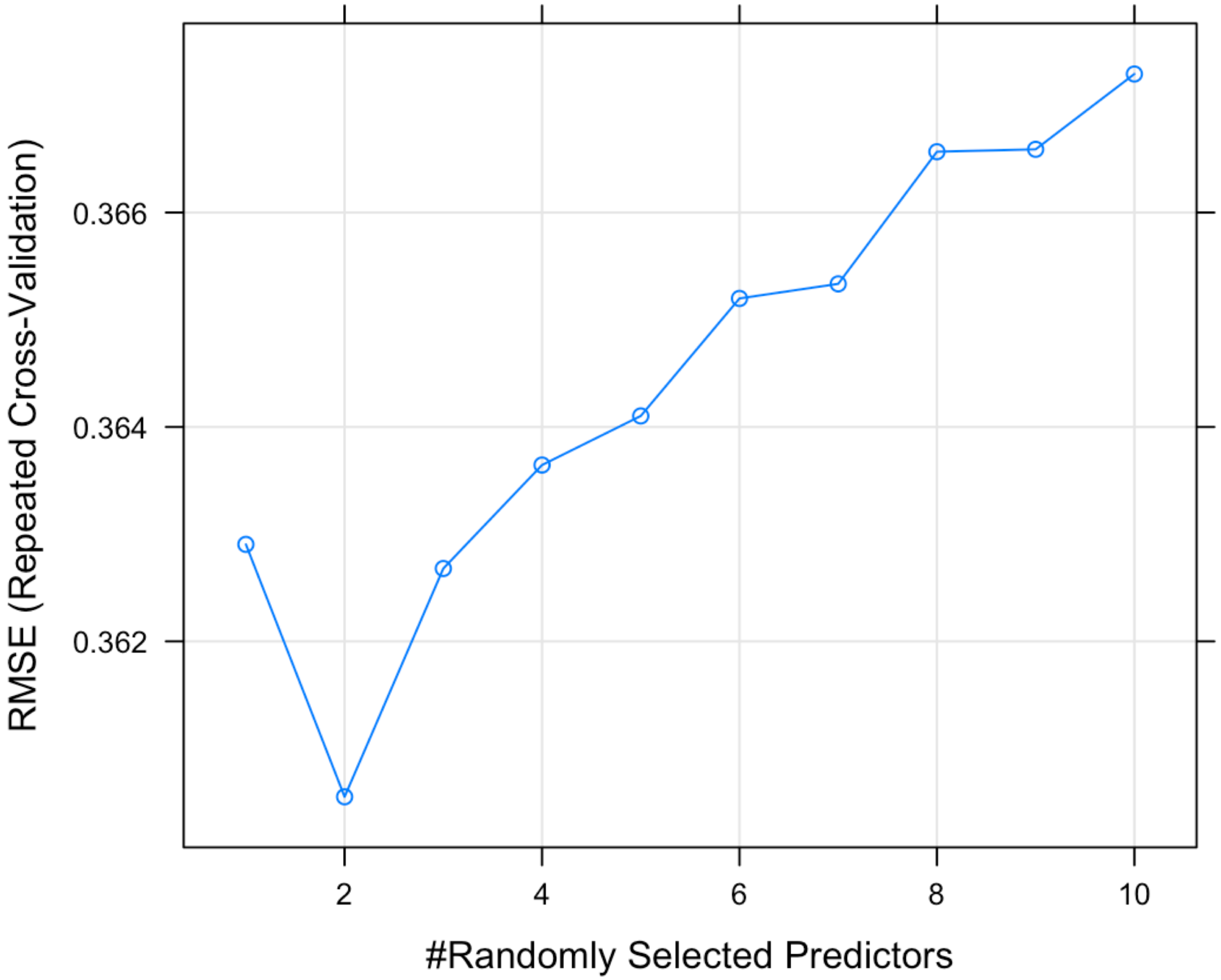
```
> rfTrain
Random Forest

758 samples
 10 predictor

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 682, 682, 682, 683, 682, 682, ...
Resampling results across tuning parameters:

  mtry  RMSE       Rsquared   MAE        Cost
   1    0.3629045  0.2403263  0.2787721  1
   2    0.3605500  0.2282054  0.2751021  1
   3    0.3626793  0.2191935  0.2763876  1
   4    0.3636449  0.2163870  0.2772056  1
   5    0.3641026  0.2149631  0.2775443  1
   6    0.3651989  0.2112161  0.2780479  1
   7    0.3653347  0.2110051  0.2784213  1
   8    0.3665673  0.2071727  0.2793231  1
   9    0.3665899  0.2075322  0.2793810  1
  10    0.3672926  0.2052027  0.2796850  1

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was mtry = 2.
```



RMSE (Repeated Cross-Validation)
0.366
0.364
0.362
2
4
6
8
10
#Randomly Selected Predictors

> plot(rfTrain)
>

Okay, your turn

- Use the Cracker dataset in the package Ecdat to build an xgboost model using caret to predict purchase of cracker type.
- Call modelLookup("xgbT ree") to intelligently choose your grid of values over which to optimize.
- Use "Accuracy" as your metric and plot the results comparing performance across parameter options.