Deep Learning in R

Обзор фреймворков с примерами

metya

2018-12-08

Disclaimer

Цель доклада не дать понимаение что такое глубокое обучение и детально разобрать как работать с ним и обучать современные модели, а скорее показать как просто можно начать тем, кто давно хотел и чесались руки, но все было никак не взяться

Deep Learning

Что это?

Deep Learning

Что это?

- Когда у нас есть исскуственная нейронная сеть

Deep Learning

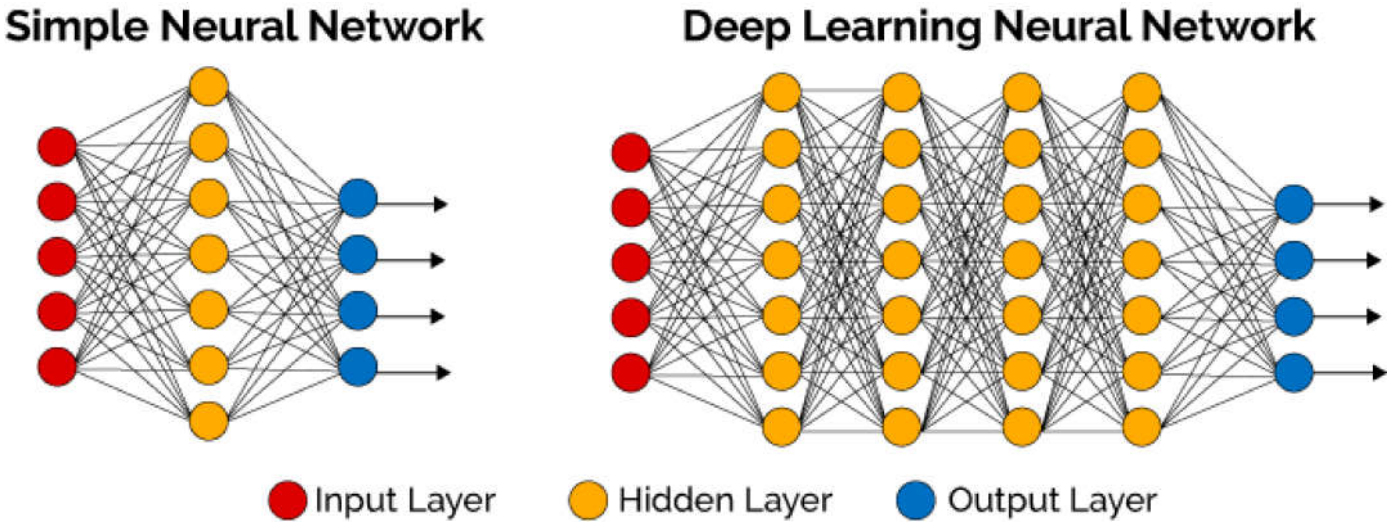
Что это?

- Когда у нас есть исскуственная нейронная сеть
- Когда скрытых слоев в этой сети больше чем два

Deep Learning

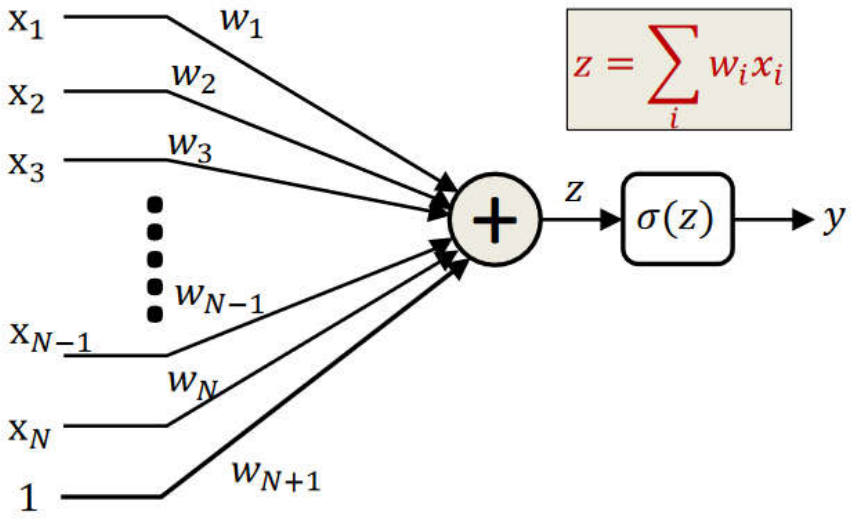
Что это?

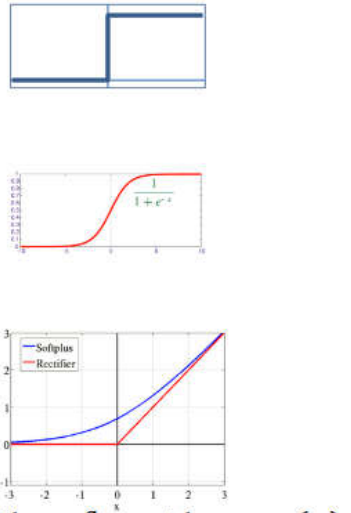
- Когда у нас есть исскуственная нейронная сеть
- Когда скрытых слоев в этой сети больше чем два



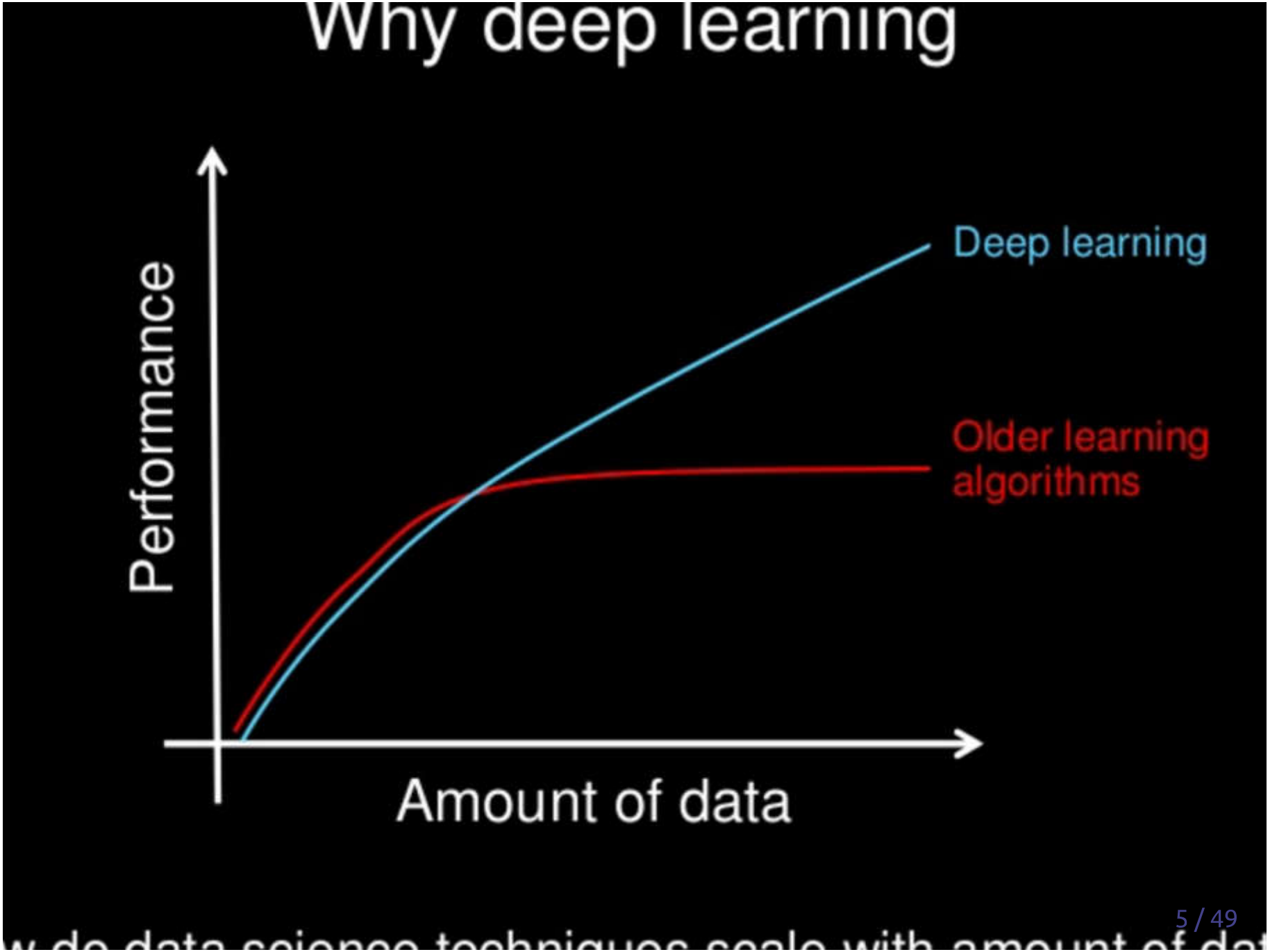
[1] https://machinelearningmastery.com/what-is-deep-learning/

Как это математически

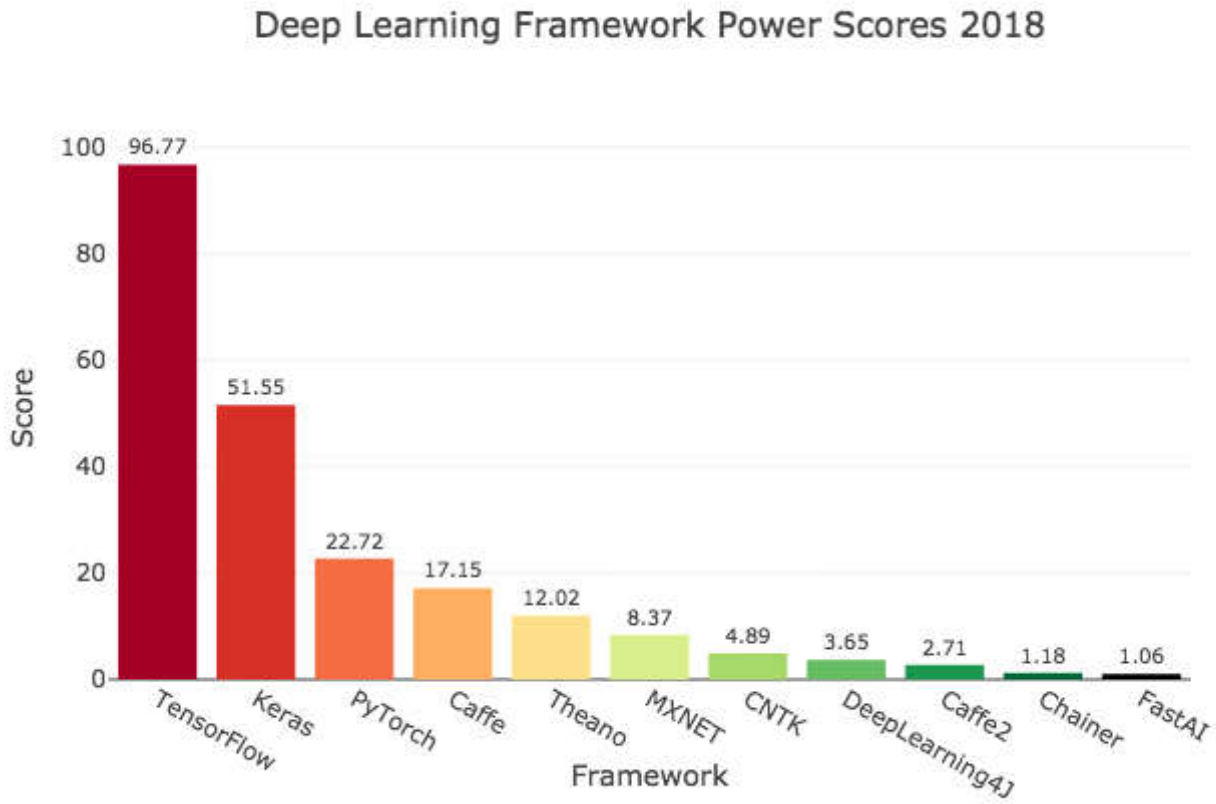




Activation functions $\sigma(z)$


Why deep learning
Performance
Deep learning
Older learning algorithms
Amount of data
5 / 49

Frameworks



[1] https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a

Нас интересуют только те, что есть в R через API

Нас интересуют только те, что есть в R через API

- TensorFlow

Нас интересуют только те, что есть в R через API

- TensorFlow
- theano

Нас интересуют только те, что есть в R через API

- TensorFlow
- theano
- Keras

Нас интересуют только те, что есть в R через API

- TensorFlow
- theano
- Keras
- CNTK

Нас интересуют только те, что есть в R через API

- TensorFlow
- theano
- Keras
- CNTK
- MXNet

Нас интересуют только те, что есть в R через API

- TensorFlow
- theano
- Keras
- CNTK
- MXNet
- ONNX

Есть еше несколько пакетов

- darch (removed from cran)
- deepnet
- deepr
- H2O (interface) (Tutorial)

https://www.tensorflow.org/

https://tensorflow.rstudio.com/

- Делает Google
- Самый популярный, имеет тучу туториалов и книг
- Имеет самый большой спрос у продакшн систем
- Имеет API во множестве языков
- Имеет статический граф вычислений, что бывает неудобно, зато оптимизированно
- Примерно с лета имеет фичу **eager execution**, который почти нивелирует это неудобство. Но почти не считается
- Доступен в R как самостоятельно, так и как бэкэнд Keras

http://www.deeplearning.net/software/theano/

- Делался силами университета Монреаль с 2007
- Один из самый старых фреймворков, но почти почил в забытьи
- Придумали идею абстракции вычислительных графов (статических) для оптимизации и вычисления нейронных сетей
- В R доступен как бэкенд через Keras

https://cntk.ai/

- Делается силами Майкрософт
- Имеет половинчатые динамические вычислительные графы (на самом деле динамические тензоры скорее)
- Доступен как бэкенд Keras так и как самостоятельный бэкенд с биндингами в R через reticulate package, что значит нужно иметь python версию фреймворка

https://keras.io/

https://keras.rstudio.com/

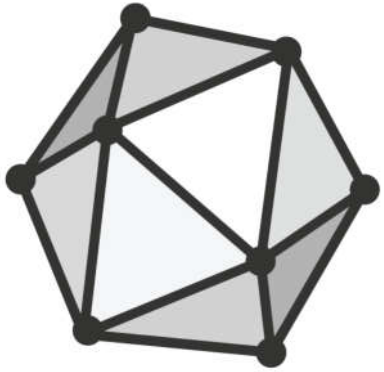
https://tensorflow.rstudio.com/keras/

- Высокоуровневый фреймворк над другими такими бэкэндами как Theano, CNTK, Tensorflow, и еще некоторые на подходе
- Делается Франсуа Шолле, который написал книгу Deep Learning in R
- Очень простой код
- Один и тот же код рабоает на разных бэкендах, что теоретически может быть полезно (нет)
- Есть очень много блоков нейросетей из современных SOTA работ
- Нивелирует боль статических вычислительных графов
- Уже давно дефолтом поставляется вместе с TensorFlow как его часть, но можно использовать и отдельно

https://mxnet.apache.org/

https://github.com/apache/incubator-mxnet/tree/master/R-package

- Является проектом Apache
- Сочетает в себе динамические и статические графы
- Тоже имеет зоопарк предобученных моделей
- Как и TensorFlow поддерживается многими языками, что может быть очень полезно
- Довольно разумный и хороший фреймворк, непонятно, почему не пользуется популярностью



ONNX

https://onnx.ai/

https://onnx.ai/onnx-r/

- Предоставляет открытый формат представления вычислительных графов, чтобы можно было обмениваться запускать одни и теже, экспортированные в этот формат, модели с помощью разных фреймворков и своего рантайма
- Можно работать из R
- Изначально делался Microsoft вместе с Facebook
- Поддерживает кучу фреймворков нативно и конвертацию в ML и TF, Keras

Deep Learning with MXNet

Установка

В Windows и MacOS в R

```
# Windows and MacOs
cran ← getOption("repos")
cran["dmlc"] ← "https://apache-mxnet.s3-accelerate.dualstack.amazona
options(repos = cran)
install.packages("mxnet")
```

Linux bash

```
# On linux
git clone --recursive https://github.com/apache/incubator-mxnet.git m
cd mxnet/docs/install
./install_mxnet_ubuntu_python.sh
./install_mxnet_ubuntu_r.sh

cd incubator-mxnet
make rpkg
```

Загрузка и обработка данных

```
df ← read_csv("data.csv")
set.seed(100)
```

```
#transform and split train on x and y
train_ind ← sample(1:77, 60)
x_train ← as.matrix(df[train_ind, 2:8])
y_train ← unlist(df[train_ind, 9])
x_val ← as.matrix(df[-train_ind, 2:8])
y_val ← unlist(df[-train_ind, 9])
```

Задания архитектуры сети

```
require(mxnet)
# define graph
data ← mx.symbol.Variable("data")

fc1 ← mx.symbol.FullyConnected(data, num_hidden = 1)

linreg ← mx.symbol.LinearRegressionOutput(fc1)

# define learing parameters
initializer ← mx.init.normal(sd = 0.1)

optimizer ← mx.opt.create("sgd",
                          learning.rate = 1e-6,
                          momentum = 0.9)
# define logger

logger ← mx.metric.logger()
epoch.end.callback ← mx.callback.log.train.metric(
  period = 4, # число батчей, после которого оценивается метрика
  logger = logger)

# num of epoch
n_epoch ← 20
```

Построим граф модели

```
# plot our model
graph.viz(linreg)
```

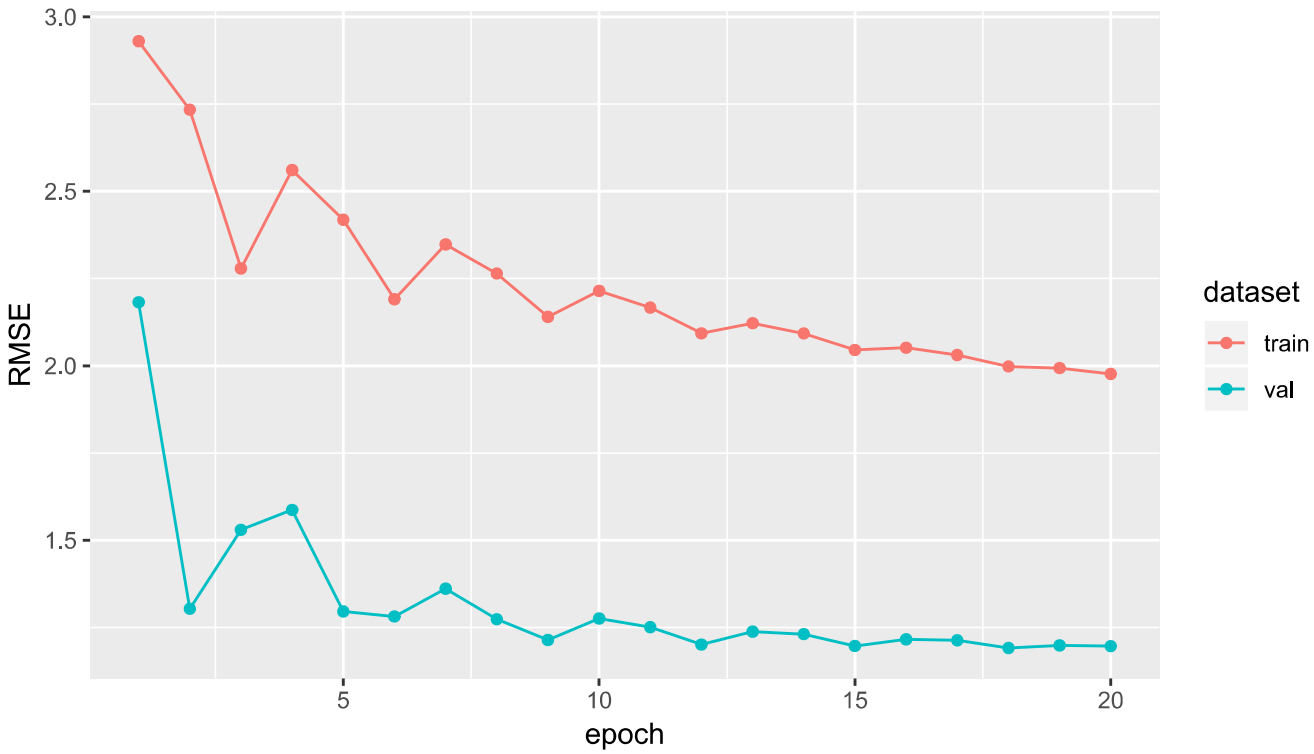
Обучим

```
model ← mx.model.FeedForward.create(
  symbol = linreg, # our model
  X = x_train, # our data
  y = y_train, # our label
  ctx = mx.cpu(), # engine
  num.round = n_epoch,
  initializer = initializer, # inizialize weigths
  optimizer = optimizer, # sgd optimizer
  eval.data = list(data = x_val, label = y_val), # evaluation on eve
  eval.metric = mx.metric.rmse, # metric
  array.batch.size = 15,
  epoch.end.callback = epoch.end.callback) # logger
```

```
## Warning in mx.model.select.layout.train(X, y): Auto detect layout of input

## Start training with 1 devices

## [1] Train-rmse=2.93010157346725

## [1] Validation-rmse=2.1820957660675

## [2] Train-rmse=2.7339181303978
```

Построим кривую обучения

```
rmse_log ← data.frame(RMSE = c(logger$train, logger$eval),dataset =
library(ggplot2)
ggplot(rmse_log, aes(epoch, RMSE, group = dataset, colour = dataset))
```



Deep Learning with Keras

Установка

```
install.packages("keras")
keras::install_keras(tensorflow = 'gpu')
```

Загрузка нужных нам пакетов

```
require(keras)     # Neural Networks
require(tidyverse) # Data cleaning / Visualization
require(knitr)     # Table printing
require(rmarkdown) # Misc. output utilities
require(ggridges)  # Visualization
```

Загрузка данных

```
activityLabels ← read.table("Deep_Learning_in_R_files/HAPT Data Set/
                            col.names = c("number", "label"))
activityLabels %>% kable(align = c("c", "l"))
```

number label

---

1 WALKING
2 WALKING_UPSTAIRS
3 WALKING_DOWNSTAIRS 4 SITTING
5 STANDING
6 LAYING
7 STAND_TO_SIT
8 SIT_TO_STAND
9 SIT_TO_LIE
10 LIE_TO_SIT
11 STAND_TO_LIE
12 LIE_TO_STAND

```
labels ← read.table("Deep_Learning_in_R_files/HAPT Data Set/RawData/
                    col.names = c("experiment", "userId", "activity"
dataFiles ← list.files("Deep_Learning_in_R_files/HAPT Data Set/RawDa
labels %>%
  head(50) %>%
  paged_table()
```

	experiment <int>	userId <int>	activity <int>	startPos <int>	endPos <int>
1	1	1	5	250	1232
2	1	1	7	1233	1392
3	1	1	4	1393	2194
4	1	1	8	2195	2359
5	1	1	5	2360	3374
6	1	1	11	3375	3662
7	1	1	6	3663	4538
8	1	1	10	4539	4735
9	1	1	4	4736	5667
10	1	1	9	5668	5859

1-10 of 50 rows Previous 1 2 3 4 5 Next

TLDR

```
allObservations ← read_rds("allObservations.rds")
allObservations %>% dim()
```

Посмотрим на данные

```
allObservations %>%
  mutate(recording_length = map_int(data,nrow)) %>%
  ggplot(aes(x = recording_length, y = activityName)) +
  geom_density_ridges(alpha = 0.8)
```

```
## Picking joint bandwidth of 32.5
```

Отфильтруем

```
desiredActivities ← c("STAND_TO_SIT", "SIT_TO_STAND", "SIT_TO_LIE",
filteredObservations ← allObservations %>%
  filter(activityName %in% desiredActivities) %>%
  mutate(observationId = 1:n())
filteredObservations %>% paged_table()
```

experiment <int>	userId <int>	activity <int>	data <list>
1	1	7	<data.frame [160 <U+00D7> 6]>
2	1	7	<data.frame [206 <U+00D7> 6]>
3	2	7	<data.frame [157 <U+00D7> 6]>
4	2	7	<data.frame [160 <U+00D7> 6]>
5	3	7	<data.frame [142 <U+00D7> 6]>
6	3	7	<data.frame [190 <U+00D7> 6]>
7	4	7	<data.frame [236 <U+00D7> 6]>
8	4	7	<data.frame [178 <U+00D7> 6]>
9	5	7	<data.frame [165 <U+00D7> 6]>
10	5	7	<data.frame [235 <U+00D7> 6]>

1-10 of 358 rows | 1-4 of 6 columns Previous 1 2 3 4 5 6 … 36 Next

Разделим на трейн тест

```
set.seed(100) # seed for reproducibility

## get all users
userIds ← allObservations$userId %>% unique()

## randomly choose 24 (80% of 30 individuals) for training
trainIds ← sample(userIds, size = 24)

## set the rest of the users to the testing set
testIds ← setdiff(userIds,trainIds)

## filter data.
trainData ← filteredObservations %>%
  filter(userId %in% trainIds)

testData ← filteredObservations %>%
  filter(userId %in% testIds)
```

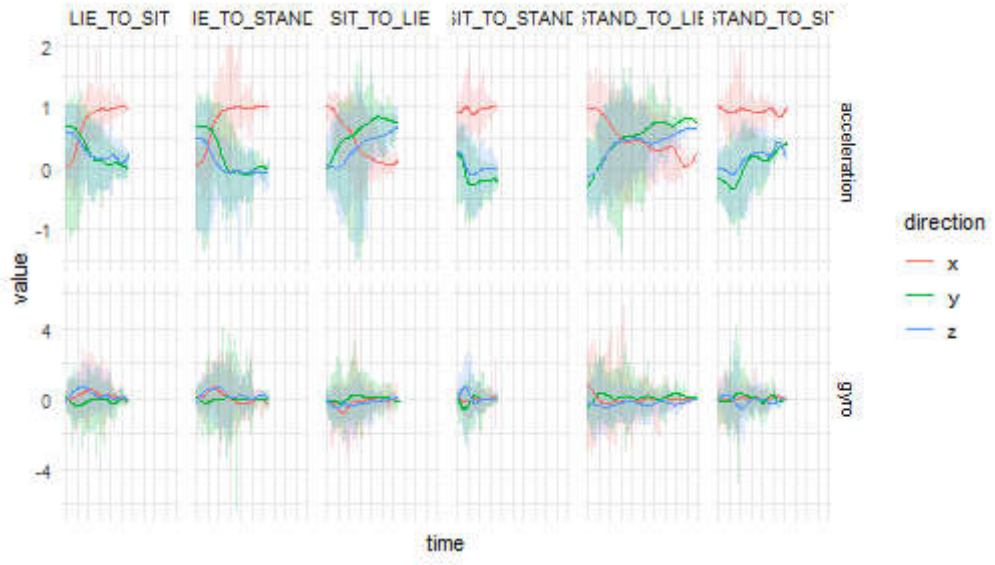
Посмотрим собственно на активности по классам

```
unpackedObs ← 1:nrow(trainData) %>%
  map_df(function(rowNum){
    dataRow ← trainData[rowNum, ]
    dataRow$data[[1]] %>%
      mutate(
        activityName = dataRow$activityName,
        observationId = dataRow$observationId,
        time = 1:n() )
  }) %>%
  gather(reading, value, -time, -activityName, -observationId) %>%
  separate(reading, into = c("type", "direction"), sep = "_") %>%
  mutate(type = ifelse(type == "a", "acceleration", "gyro"))
```

Посмотрим собственно на активности по классам

```
unpackedObs %>%
  ggplot(aes(x = time, y = value, color = direction)) +
  geom_line(alpha = 0.2) +
  geom_smooth(se = FALSE, alpha = 0.7, size = 0.5) +
  facet_grid(type ~ activityName, scales = "free_y") +
  theme_minimal() +
  theme( axis.text.x = element_blank() )
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



Подготовка данных к обучению

```
padSize ← trainData$data %>%
  map_int(nrow) %>%
  quantile(p = 0.98) %>%
  ceiling()
padSize
```

```
## 98%
## 334
```

```
convertToTensor ← . %>%
  map(as.matrix) %>%
  pad_sequences(maxlen = padSize)

trainObs ← trainData$data %>% convertToTensor()
testObs ← testData$data %>% convertToTensor()

dim(trainObs)
```

```
## [1] 286 334   6
```

Подготовка данных к обучению

```
# one hot encoding
oneHotClasses ← . %>%
{• - 7} %>%         # bring integers down to 0-6 from 7-12
  to_categorical() # One-hot encode

trainY ← trainData$activity %>% oneHotClasses()
testY ← testData$activity %>% oneHotClasses()
```

Наконец то сетка!

```
input_shape ← dim(trainObs)[-1]
num_classes ← dim(trainY)[2]

filters ← 24     # number of convolutional filters to learn
kernel_size ← 8  # how many time-steps each conv layer sees.
dense_size ← 48  # size of our penultimate dense layer.
```

Наконец то сетка!

```
model ← keras_model_sequential()
model %>% layer_conv_1d(
    filters = filters,
    kernel_size = kernel_size,
    input_shape = input_shape,
    padding = "valid",
    activation = "relu") %>%
  layer_batch_normalization() %>%
  layer_spatial_dropout_1d(0.15) %>%
  layer_conv_1d(filters = filters/2,
    kernel_size = kernel_size,
    activation = "relu") %>%
  layer_global_average_pooling_1d() %>%
  layer_batch_normalization() %>%
  layer_dropout(0.2) %>%
  layer_dense(dense_size,
    activation = "relu") %>%
  layer_batch_normalization() %>%
  layer_dropout(0.25) %>%
  layer_dense(num_classes,
    activation = "softmax",
    name = "dense_output")
```

Наконец то сетка!

Выведем описание нашей сетки

```
summary(model)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_7 (Conv1D)            (None, 327, 24)           1176
_________________________________________________________________
batch_normalization_10 (BatchNormalization) (None, 327, 24)      96
_________________________________________________________________
spatial_dropout1d_4 (SpatialDropout1D) (None, 327, 24)           0
_________________________________________________________________
conv1d_8 (Conv1D)            (None, 320, 12)           2316
_________________________________________________________________
global_average_pooling1d_4 (GlobalAveragePooling (None, 12)      0
_________________________________________________________________
batch_normalization_11 (BatchNormalization) (None, 12)           48
_________________________________________________________________
dropout_7 (Dropout)          (None, 12)                0
_________________________________________________________________
dense_4 (Dense)              (None, 48)                624
_________________________________________________________________
batch_normalization_12 (BatchNormalization) (None, 48)           192
_________________________________________________________________
dropout_8 (Dropout)          (None, 48)                0
_________________________________________________________________
dense_output (Dense)         (None, 6)                 294
=================================================================
Total params: 4,746
Trainable params: 4,578
Non-trainable params: 168
_________________________________________________________________
```

Обучим же наконец

Компиляция графа

```
model %>% compile(
  loss = "categorical_crossentropy",
  optimizer = "rmsprop",
  metrics = "accuracy"
)
```

Обучим же наконец

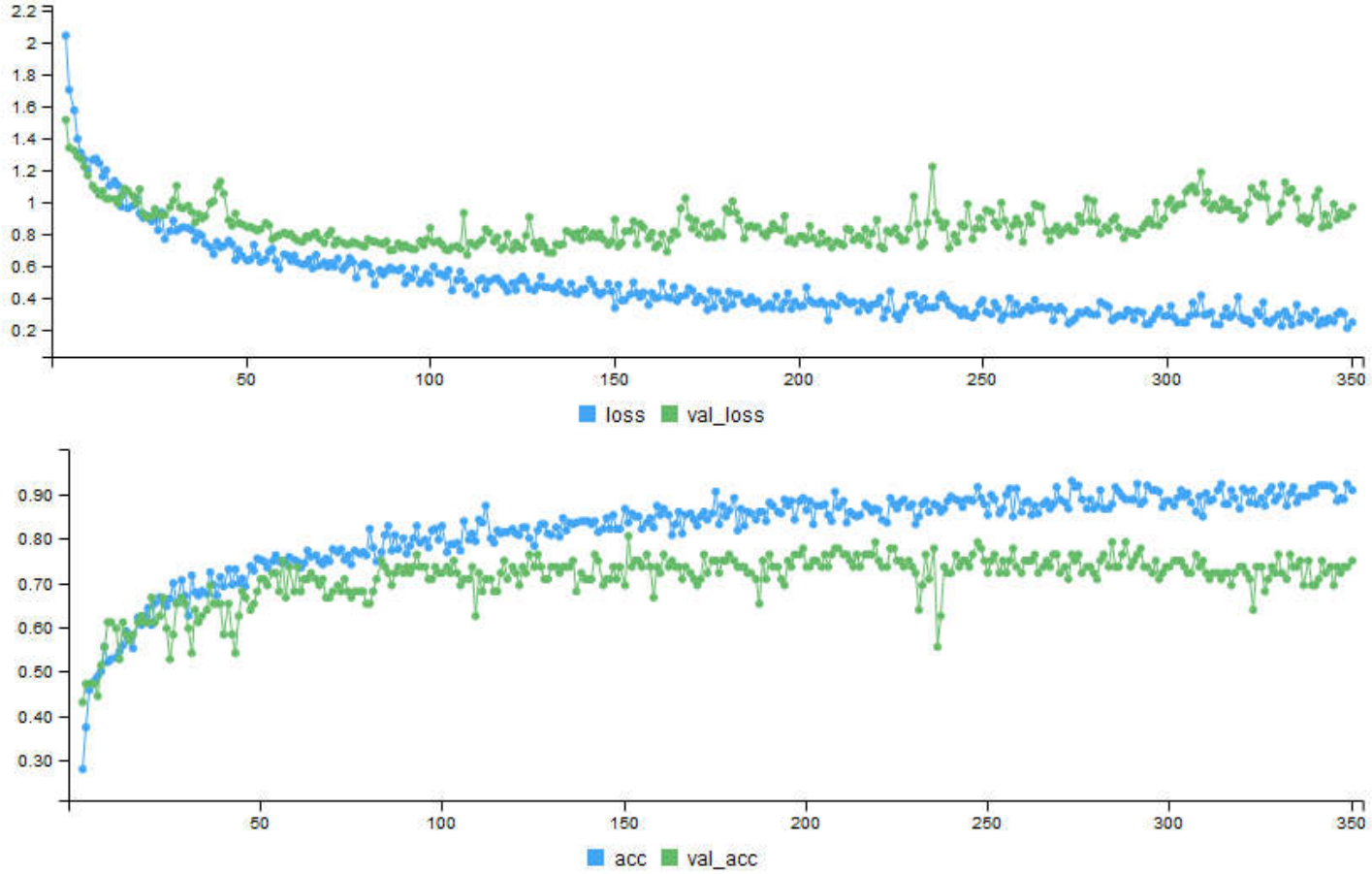
train

```
trainHistory ← model %>%
  fit(
    x = trainObs, y = trainY, # data
    epochs = 350, # num epoch
    validation_data = list(testObs, testY), # validation tests on eac
    callbacks = list(
      callback_model_checkpoint("best_model.h5",
                                save_best_only = TRUE))) # update tra
```

Обучим же наконец

```
Train on 286 samples, validate on 72 samples
Epoch 1/350
286/286 [==============================] - 3s 10ms/step - loss: 2.0388 - acc: 0.2797 - val_loss: 1.5122 - val_acc: 0.4306
Epoch 2/350
286/286 [==============================] - 0s 483us/step - loss: 1.6998 - acc: 0.3741 - val_loss: 1.3388 - val_acc: 0.4722
Epoch 3/350
286/286 [==============================] - 0s 500us/step - loss: 1.5719 - acc: 0.4580 - val_loss: 1.3223 - val_acc: 0.4722
Epoch 4/350
286/286 [==============================] - 0s 531us/step - loss: 1.3942 - acc: 0.4790 - val_loss: 1.2848 - val_acc: 0.4722
Epoch 5/350
286/286 [==============================] - 0s 610us/step - loss: 1.3068 - acc: 0.4895 - val_loss: 1.2697 - val_acc: 0.4444
Epoch 6/350
286/286 [==============================] - 0s 486us/step - loss: 1.2666 - acc: 0.5000 - val_loss: 1.2210 - val_acc: 0.5139
Epoch 7/350
286/286 [==============================] - 0s 580us/step - loss: 1.2000 - acc: 0.5559 - val_loss: 1.1661 - val_acc: 0.5556
Epoch 8/350
286/286 [==============================] - 0s 503us/step - loss: 1.2626 - acc: 0.5210 - val_loss: 1.1003 - val_acc: 0.6111
Epoch 9/350
286/286 [==============================] - 0s 514us/step - loss: 1.2714 - acc: 0.5280 - val_loss: 1.0738 - val_acc: 0.6111
```

Обучим же наконец



Предсказание

Подготовка теста

```
oneHotToLabel ← activityLabels %>%
  mutate(number = number - 7) %>%
  filter(number ⩾ 0) %>%
  mutate(class = paste0("V",number + 1)) %>%
  select(-number)
```

Выбор лучшей модели

```
bestModel ← load_model_hdf5("best_model.h5")
```

Предсказание

Еще немного кода

```
tidyPredictionProbs ← bestModel %>%
  predict(testObs) %>%
  as_data_frame() %>%
  mutate(obs = 1:n()) %>%
  gather(class, prob, -obs) %>%
  right_join(oneHotToLabel, by = "class")

predictionPerformance ← tidyPredictionProbs %>%
  group_by(obs) %>%
  summarise(
    highestProb = max(prob),
    predicted = label[prob == highestProb]
  ) %>%
  mutate(
    truth = testData$activityName,
    correct = truth == predicted
  )
```

Предсказание

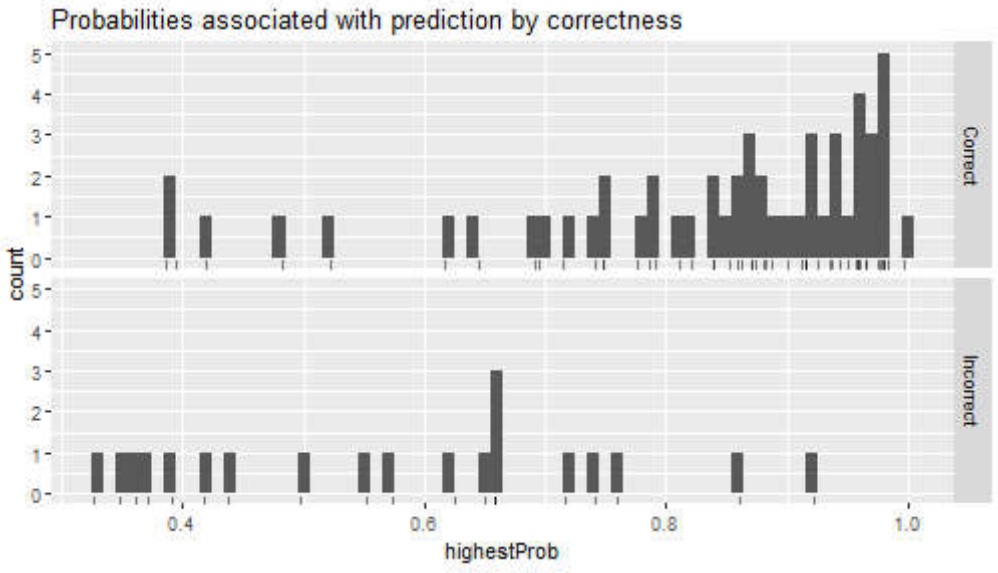
```
predictionPerformance %>% paged_table()
```

obs <int>	highestProb <dbl>	predicted <fctr>	truth <fctr>	correct <lgl>
1	0.6953406	STAND_TO_SIT	STAND_TO_SIT	TRUE
2	0.7918580	STAND_TO_SIT	STAND_TO_SIT	TRUE
3	0.8739656	STAND_TO_SIT	STAND_TO_SIT	TRUE
4	0.6162640	STAND_TO_SIT	STAND_TO_SIT	TRUE
5	0.9656017	STAND_TO_SIT	STAND_TO_SIT	TRUE
6	0.9810318	STAND_TO_SIT	STAND_TO_SIT	TRUE
7	0.4375742	SIT_TO_STAND	STAND_TO_SIT	FALSE
8	0.7479992	STAND_TO_SIT	STAND_TO_SIT	TRUE
9	0.6449857	STAND_TO_SIT	STAND_TO_SIT	TRUE
10	0.3713405	STAND_TO_LIE	STAND_TO_SIT	FALSE

1-10 of 72 rows Previous 1 2 3 4 5 6 … 8 Next

Визуализация ошибок

```
predictionPerformance %>%
  mutate(result = ifelse(correct, 'Correct', 'Incorrect')) %>%
  ggplot(aes(highestProb)) +
  geom_histogram(binwidth = 0.01) +
  geom_rug(alpha = 0.5) +
  facet_grid(result~.) +
  ggtitle("Probabilities associated with prediction by correctness")
```



Визуализация ошибок

```
predictionPerformance %>%
  group_by(truth, predicted) %>%
  summarise(count = n()) %>%
  mutate(good = truth == predicted) %>%
  ggplot(aes(x = truth,  y = predicted)) +
  geom_point(aes(size = count, color = good)) +
  geom_text(aes(label = count),
            hjust = 0, vjust = 0,
            nudge_x = 0.1, nudge_y = 0.1) +
  guides(color = FALSE, size = FALSE) +
  theme_minimal()
```

Заключение

Deep Learning with R

François Chollet
with J. J. Allaire

Спасибо!

Слайды сделаны с помощью R package **xaringan**.

Веб версию слайдов можно найти на
https://metya.github.io/DeepLearning_n_R/

Код можно посмотреть здесь https://github.com/metya/DeepLearning_n_R/