Deep Learning 실습

원중호

서울대학교 통계학과

2019년 6월 21일

목차

- Tensorflow and Keras
- Using Keras on R
- Example - MNIST Data and CNN
- Example - Generate Nietzsche's writing with LSTM

Section 1

Tensorflow and Keras

Tensorflow 소개

- an open source software library for numerical computation using data flow graphs, more than deep learning. TensorFlow actually has tools to support reinforcement learning and other algos.
- Google Brain Team within Google's Machine Intelligence research organization
- 언어 : a Python API over a C/C++ engine
  - Deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API
- Deep Learning Course at Udacity

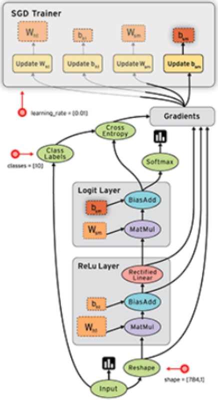
Tensorflow 특징

- Deep Flexibility
  - TensorFlow isn't a rigid neural networks library
  - If you can express your computation as a data flow graph, you can use TensorFlow
  - You construct the graph, and you write the inner loop that drives computation
- True Portability
  - TensorFlow runs on CPUs or GPUs, and on desktop, server, or mobile computing platforms
  - Scale-up and train that model faster on GPUs with no code changes
  - Deploy that trained model on mobile
  - Run the model as a service in the cloud

Tensorflow 특징

- Connect Research and Production
  - Google research scientists experiment with new algorithms in TensorFlow
  - Google Product teams use TensorFlow to train and serve models live to real customers
- Auto-Differentiation
  - Can define the computational architecture of predictive model, combine that with objective function, and just add data
  - TensorFlow handles computing the derivatives
  - Computing the derivative of some values w.r.t. other values in the model just extends the graph

Computational Graph



https://www.tensorflow.org/images/tensors_flowing.gif

Computational Graph

- TensforFlow generates a computational graph (e.g. a series of matrix operations such as $z = \text{simoid}(x)$ where x and z are matrices) and performs automatic differentiation.
- Automatic differentiation is important because you don't want to have to hand-code a new variation of backpropagation every time you're experimenting with a new arrangement of neural networks.

Mathematical computation with a directed graph

Nodes

- represent mathematical operations
- can also represent endpoints to feed in data, push out results, or read/write persistent variables
- assigned to computational devices
    - execute asynchronously and in parallel once all the tensors on their incoming edges becomes available

Edges

- represent the input/output relationships between nodes
- the multidimensional data arrays (tensors) communicated between them

Tensorflow Reference

- TensorFlow Tutorials
- Additional Resources
- Zoo : TensorFlow enables researchers to build machine learning models. We collect such models in our Zoo.

Keras 소개

- a minimalist, highly modular neural networks library, written in Python and capable of running on top of either TensorFlow or Theano.
- https://github.com/fchollet/keras
- 언어 : a Python API on top of TensorFlow or Theano
- The MIT License (MIT)
- Examples

Keras 특징

특징

- **Modularity.** A model is understood as a sequence or a graph of standalone, fully-configurable modules that can be plugged together with as little restrictions as possible. In particular, neural layers, cost functions, optimizers, initialization schemes, activation functions, regularization schemes are all standalone modules that you can combine to create new models.
- **Minimalism.** Each module should be kept short and simple. Every piece of code should be transparent upon first reading. No black magic: it hurts iteration speed and ability to innovate.
- **Easy extensibility.** New modules are dead simple to add (as new classes and functions), and existing modules provide ample examples. To be able to easily create new modules allows for total expressiveness, making Keras suitable for advanced research.
- **Work with Python**. No separate models configuration files in a declarative format. Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility.

Keras backends

- http://keras.io/backend/

TensorFlow

- an open-source symbolic tensor manipulation framework developed by Google, Inc.

Theano

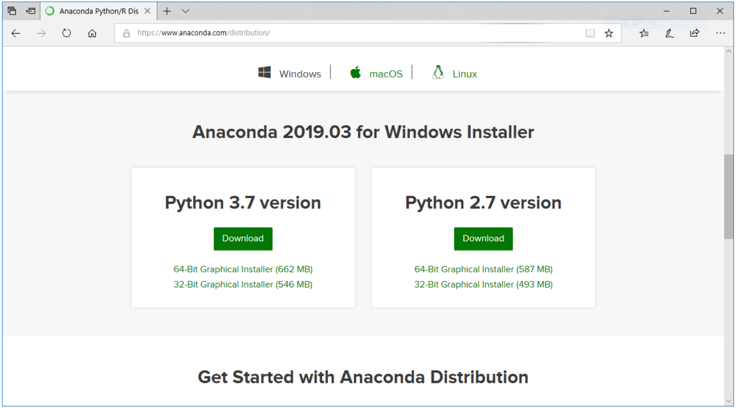
- an open-source symbolic tensor manipulation framework developed by LISA Lab at Université de Montréal.

CNTK

- an open-source toolkit for deep learning developed by Microsoft.

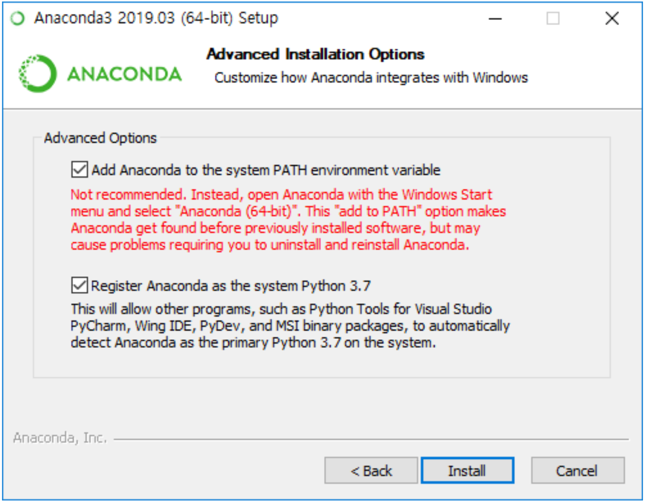
Using Keras on R

- Install Anaconda for Windows (Python 3.7) at https://www.anaconda.com/distribution/



Using Keras on R

- Be sure to add anaconda on your system path.



Using Keras on R

- keras_setting.R
- Setup Keras with Tensorflow backend on Rstudio

The following R code will setup conda environment for keras automatically.

```
install.packages("keras")

library(keras)
keras_install()
```

Section 2

Keras Example - MNIST Data and CNN

Keras Example - MNIST Data

- MNIST_dense.R
- Load library and set hyper-parameter

```r
library(keras)

batch_size <- 128
num_classes <- 10
epochs <- 30
```

Keras Example - MNIST Data

- Get MNIST data

```
mnist <- dataset_mnist()
x_train <- mnist$train$x
y_train <- mnist$train$y
x_test <- mnist$test$x
y_test <- mnist$test$y

dim(x_train)
```

```
## [1] 60000    28    28
```

```
dim(y_train)
```

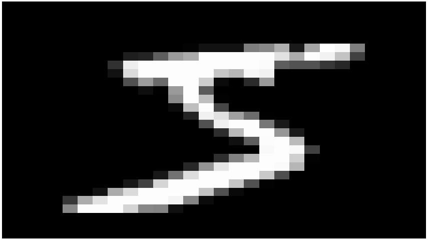
```
## [1] 60000
```

```
y_train[1]
```

```
## [1] 5
```

Keras Example - MNIST Data

```
image(t(x_train[1, 28:1,]), useRaster=FALSE, axes=FALSE,
      col=grey(seq(0, 1, length = 256)))
```



Keras Example - MNIST Data

- Prepare dataset

```r
x_train <- array_reshape(x_train, c(nrow(x_train), 784))
x_test <- array_reshape(x_test, c(nrow(x_test), 784))
dim(x_train)
```

```
## [1] 60000   784
```

```r
x_train <- x_train / 255
x_test <- x_test / 255

y_train <- to_categorical(y_train, 10)
y_test <- to_categorical(y_test, 10)
dim(y_train)
```

```
## [1] 60000    10
```

```r
y_train[1,]
```

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

Keras Example - MNIST Data

- Architecture Design

```
model <- keras_model_sequential()
model %>%
  layer_dense(units = 256, activation = 'relu',
              input_shape = c(784)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = num_classes, activation = 'softmax')
summary(model)
```

Keras Example - MNIST Data

- Architecture Design

```
> summary(model)
________________________________________________________________________________
Layer (type)                        Output Shape                    Param #
================================================================================
dense (Dense)                       (None, 256)                     200960
________________________________________________________________________________
dropout (Dropout)                   (None, 256)                     0
________________________________________________________________________________
dense_1 (Dense)                     (None, 128)                     32896
________________________________________________________________________________
dropout_1 (Dropout)                 (None, 128)                     0
________________________________________________________________________________
dense_2 (Dense)                     (None, 10)                      1290
================================================================================
Total params: 235,146
Trainable params: 235,146
Non-trainable params: 0
________________________________________________________________________________
```

Keras Example - MNIST Data

- Architecture design example : click Tinker With a Neural Network in Your Browser

Architecture Components

- 2D Convolution, 2D Maxpooling, Dense(MLP) layers. . .
- ReLU, Softmax activations
- Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Keras Example - MNIST Data

- Model compile

```r
model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_adam(),
  metrics = c('accuracy')
)
```

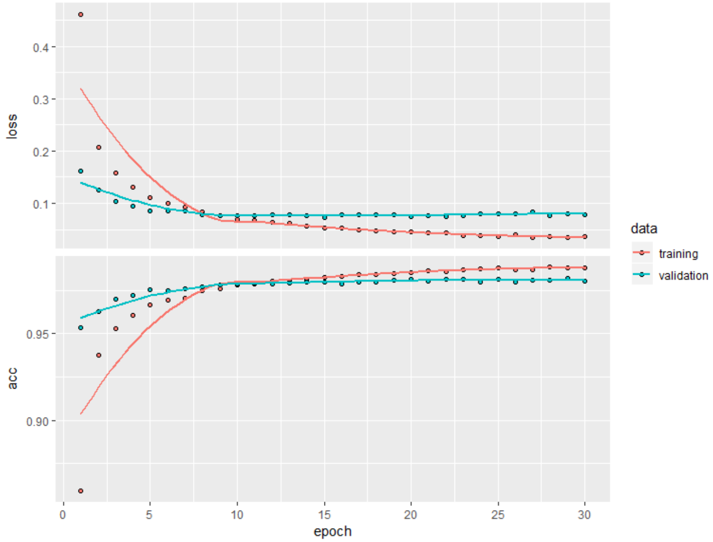
- Model training

```r
history <- model %>% fit(
  x_train, y_train,
  epochs = epochs, batch_size = batch_size,
  validation_split = 0.2
)
```

Keras Example - MNIST Data

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/30
48000/48000 [==============================] - 3s 66us/sample - loss: 0.4637 - acc: 0.8575 - val_loss: 0.1667 - val_acc: 0.9488
Epoch 2/30
48000/48000 [==============================] - 3s 56us/sample - loss: 0.2061 - acc: 0.9389 - val_loss: 0.1144 - val_acc: 0.9651
Epoch 3/30
48000/48000 [==============================] - 3s 55us/sample - loss: 0.1530 - acc: 0.9536 - val_loss: 0.1049 - val_acc: 0.9693
Epoch 4/30
48000/48000 [==============================] - 3s 56us/sample - loss: 0.1268 - acc: 0.9613 - val_loss: 0.0921 - val_acc: 0.9712
Epoch 5/30
48000/48000 [==============================] - 3s 56us/sample - loss: 0.1108 - acc: 0.9668 - val_loss: 0.0862 - val_acc: 0.9744
Epoch 6/30
48000/48000 [==============================] - 3s 55us/sample - loss: 0.0984 - acc: 0.9702 - val_loss: 0.0831 - val_acc: 0.9759
Epoch 7/30
48000/48000 [==============================] - 3s 59us/sample - loss: 0.0906 - acc: 0.9719 - val_loss: 0.0836 - val_acc: 0.9773
Epoch 8/30
15616/48000 [========>.....................] - ETA: 1s - loss: 0.0777 - acc: 0.9754
```

Keras Example - MNIST Data



Keras Example - MNIST Data

- Evaluation and Prediction

```
model %>% evaluate(x_test, y_test)
model %>% predict_classes(x_test) %>% head()
```

```
> model %>% evaluate(x_test, y_test)
10000/10000 [==============================] - 0s 31us/sample - loss: 0.0741 - acc: 0.9828
$loss
[1] 0.07407283

$acc
[1] 0.9828

>
> model %>% predict_classes(x_test) %>% head()
[1] 7 2 1 0 4 1
```

Keras Example - MNIST Data and CNN

- MNIST_cnn.R

```r
# CNN Data Preparation
# Input image dimensions
img_rows <- 28
img_cols <- 28

# The data, shuffled and split between train and test sets
mnist <- dataset_mnist()
x_train <- mnist$train$x
y_train <- mnist$train$y
x_test <- mnist$test$x
y_test <- mnist$test$y
```

Keras Example - MNIST Data and CNN

- Data Preparation

```
# Input image dimensions
img_rows <- 28
img_cols <- 28

# The data, shuffled and split between train and test sets
mnist <- dataset_mnist()
x_train <- mnist$train$x
y_train <- mnist$train$y
x_test <- mnist$test$x
y_test <- mnist$test$y
```

Keras Example - MNIST Data and CNN

```r
# Redefine  dimension of train/test inputs
x_train <- array_reshape(x_train, c(nrow(x_train),
                                    img_rows, img_cols, 1))
x_test <- array_reshape(x_test, c(nrow(x_test),
                                  img_rows, img_cols, 1))
input_shape <- c(img_rows, img_cols, 1)

# Transform RGB values into [0,1] range
x_train <- x_train / 255
x_test <- x_test / 255

y_train <- to_categorical(y_train, 10)
y_test <- to_categorical(y_test, 10)
```

Keras Example - MNIST Data and CNN

```r
model <- keras_model_sequential()
model %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3),
                activation = 'relu', input_shape = input_shape) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3),
                activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_dropout(rate = 0.25) %>%
  layer_flatten() %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = num_classes, activation = 'softmax')
```

Keras Example - MNIST Data and CNN

```
> summary(model)
________________________________________________________________________________
Layer (type)                        Output Shape                    Param #
================================================================================
conv2d (Conv2D)                     (None, 26, 26, 32)              320
________________________________________________________________________________
conv2d_1 (Conv2D)                   (None, 24, 24, 64)              18496
________________________________________________________________________________
max_pooling2d (MaxPooling2D)        (None, 12, 12, 64)              0
________________________________________________________________________________
dropout (Dropout)                   (None, 12, 12, 64)              0
________________________________________________________________________________
flatten (Flatten)                   (None, 9216)                    0
________________________________________________________________________________
dense (Dense)                       (None, 128)                     1179776
________________________________________________________________________________
dropout_1 (Dropout)                 (None, 128)                     0
________________________________________________________________________________
dense_1 (Dense)                     (None, 10)                      1290
================================================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
```

Keras Example - MNIST Data and CNN

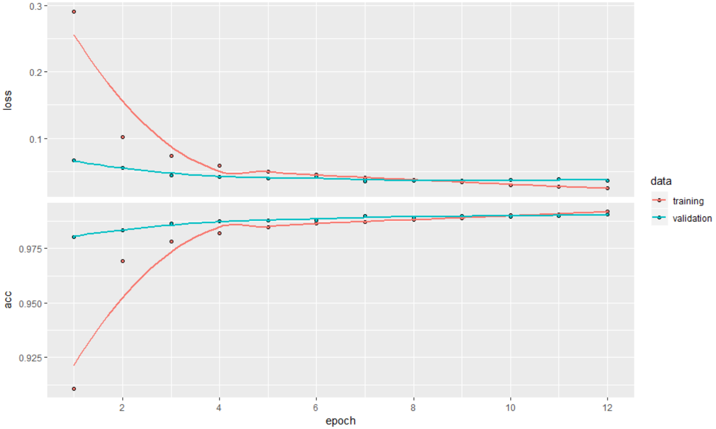
```
# Model config
model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_adam(),
  metrics = c('accuracy')
)
summary(model)

# Start Fitting
system.time({
  history <- model %>% fit(
    x_train, y_train,
    batch_size = batch_size,
    epochs = epochs,
    validation_split = 0.2
  )
})
```

Keras Example - MNIST Data and CNN

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/12
48000/48000 [==============================] - 83s 2ms/sample - loss: 0.2910 - acc: 0.9106 - val_loss: 0.0663
 - val_acc: 0.9805
Epoch 2/12
48000/48000 [==============================] - 83s 2ms/sample - loss: 0.1014 - acc: 0.9693 - val_loss: 0.0558
 - val_acc: 0.9835
Epoch 3/12
48000/48000 [==============================] - 83s 2ms/sample - loss: 0.0734 - acc: 0.9781 - val_loss: 0.0446
 - val_acc: 0.9867
Epoch 4/12
48000/48000 [==============================] - 83s 2ms/sample - loss: 0.0587 - acc: 0.9821 - val_loss: 0.0413
 - val_acc: 0.9874
Epoch 5/12
48000/48000 [==============================] - 83s 2ms/sample - loss: 0.0498 - acc: 0.9849 - val_loss: 0.0399
 - val_acc: 0.9880
Epoch 6/12
48000/48000 [==============================] - 83s 2ms/sample - loss: 0.0448 - acc: 0.9864 - val_loss: 0.0424
 - val_acc: 0.9878
Epoch 7/12
48000/48000 [==============================] - 83s 2ms/sample - loss: 0.0410 - acc: 0.9874 - val_loss: 0.0353
 - val_acc: 0.9899
```

Keras Example - MNIST Data and CNN



Keras Example - MNIST Data and CNN

- Model evaluation

```
> scores <- model %>% evaluate(
+   x_test, y_test, verbose = 0
+ )
> # Output metrics
> cat('Test loss:', scores[[1]], '\n')
Test loss: 0.03092995
> cat('Test accuracy:', scores[[2]], '\n')
Test accuracy: 0.9923
> 
```

Keras Example - MNIST Data and CNN

- Save whole model

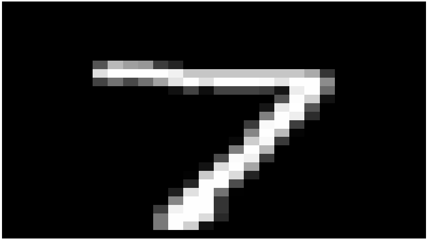
```
model %>% save_model_hdf5("MNIST_cnn.h5")
new_model <- load_model_hdf5("MNIST_cnn.h5")
new_model %>% summary()
```

- Save weights in model

```
model %>% save_model_weights_hdf5("MNIST_cnn_weights.h5")
model %>% load_model_weights_hdf5("MNIST_cnn_weights.h5")
model %>% predict_classes(x_test) %>% head()
```

Keras Example - MNIST Data and CNN

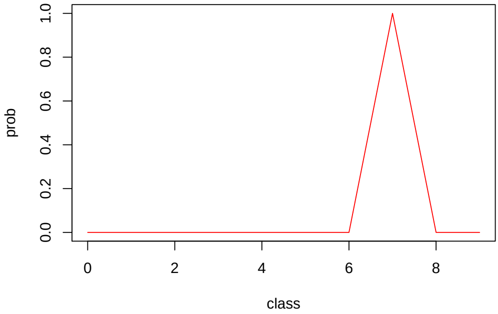
```
probe <- array_reshape(x_test[1,], c(1, 28, 28, 1))
image(t(probe[1, 28:1, ,1]*255), useRaster=FALSE,
      axes=FALSE, col=grey(seq(0, 1, length = 256)))
```



Keras Example - MNIST Data and CNN

- Predicted probability for each class

```
model %>% predict(probe) %>% plot(x=0:9, y = ., type = "l",
                                  col = "red", xlab = "class", ylab
```



Keras Example - Image recognition with advanced CNN

- ResNet.R
- Download Pre-trained Model

```
model <- application_resnet50(weights = 'imagenet')

img_path <- "elephant.jpg"
img <- image_load(img_path, target_size = c(224,224))
x <- image_to_array(img)
x <- array_reshape(x, c(1, dim(x)))
x <- imagenet_preprocess_input(x)

preds <- model %>% predict(x)
imagenet_decode_predictions(preds, top = 3)[[1]]
```

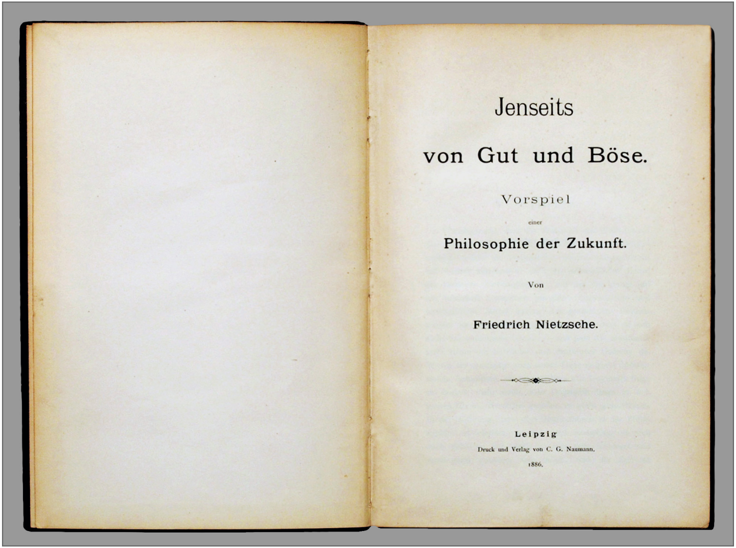
```
##   class_name class_description      score
## 1  n02504458  African_elephant 0.81822115
## 2  n01871265            tusker 0.15097509
## 3  n02504013   Indian_elephant 0.02352115
```

Section 3

Keras Example - Generate Nietzsche's writing with LSTM

Keras Example - Generate Nietzsche's writing with LSTM

- Generate_Nietzsche.R



Keras Example - Generate Nietzsche's writing with LSTM

- Data preparation

```r
library(keras)
library(readr)
library(stringr)
library(purrr)
library(tokenizers)

# Parameters ---------------------------------------------------------

maxlen <- 40

# Data Preparation ---------------------------------------------------

# Retrieve text
path <- get_file(
  'nietzsche.txt',
  origin='https://s3.amazonaws.com/text-datasets/nietzsche.txt'
  )
```

Keras Example - Generate Nietzsche's writing with LSTM

```
> read_lines(path) %>% head()
[1] "PREFACE"
[2] ""
[3] ""
[4] "SUPPOSING that Truth is a woman--what then? Is there not ground"
[5] "for suspecting that all philosophers, in so far as they have been"
[6] "dogmatists, have failed to understand women--that the terrible"
```

```r
text <- read_lines(path) %>%
  str_to_lower() %>%
  str_c(collapse = "\n") %>%
  tokenize_characters(strip_non_alphanum = FALSE, simplify = TRUE)

print(sprintf("corpus length: %d", length(text)))
```

```
## [1] "corpus length: 600893"
```

Keras Example - Generate Nietzsche's writing with LSTM

```r
chars <- text %>%
  unique() %>%
  sort()

print(sprintf("total chars: %d", length(chars)))
```

```
## [1] "total chars: 57"
```

```
> chars
 [1] "'"  "-"  " "  "\n" "!"  "\"" "("  ")"  ","  "."  ":"  ";"  "?"  "["  "]"
[16] "_"  "="  "0"  "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "a"  "ä"  "æ"
[31] "b"  "c"  "d"  "ë"  "e"  "é"  "f"  "g"  "h"  "i"  "j"  "k"  "l"  "m"  "n"
[46] "o"  "p"  "q"  "r"  "s"  "t"  "u"  "v"  "w"  "x"  "y"  "z"
```

Keras Example - Generate Nietzsche's writing with LSTM

```
> text[1:100]
  [1] "p"  "r"  "e"  "f"  "a"  "c"  "e"  "\n" "\n" "\n" "s"  "u"  "p"  "p"
 [15] "o"  "s"  "i"  "n"  "g"  " "  "t"  "h"  "a"  "t"  " "  "t"  "r"  "u"
 [29] "t"  "h"  " "  "i"  "s"  " "  "a"  " "  "w"  "o"  "m"  "a"  "n"  "-"
 [43] "-"  "w"  "h"  "a"  "t"  " "  "t"  "h"  "e"  "n"  "?"  " "  "i"  "s"
 [57] " "  "t"  "h"  "e"  "r"  "e"  " "  "n"  "o"  "t"  " "  "g"  "r"  "o"
 [71] "u"  "n"  "d"  "\n" "f"  "o"  "r"  " "  "s"  "u"  "s"  "p"  "e"  "c"
 [85] "t"  "i"  "n"  "g"  " "  "t"  "h"  "a"  "t"  " "  "a"  "l"  "l"  " "
 [99] "p"  "h"
```

Keras Example - Generate Nietzsche's writing with LSTM

- Cutting texts in semi-redundant sequences of maxlen characters

```
dataset <- map(
  seq(1, length(text) - maxlen - 1, by = 3),
  ~list(sentece = text[.x:(.x + maxlen - 1)],
        next_char = text[.x + maxlen])
  )

dataset <- transpose(dataset)
```

Keras Example - Generate Nietzsche's writing with LSTM

```
> dataset$sentece[[1]]
 [1] "p"  "r"  "e"  "f"  "a"  "c"  "e"  "\n" "\n" "\n" "s"  "u"  "p"  "p"  "o"
[16] "s"  "i"  "n"  "g"  " "  "t"  "h"  "a"  "t"  " "  "t"  "r"  "u"  "t"  "h"
[31] " "  "i"  "s"  " "  "a"  " "  "w"  "o"  "m"  "a"
> dataset$sentece[[2]]
 [1] "f"  "a"  "c"  "e"  "\n" "\n" "\n" "s"  "u"  "p"  "p"  "o"  "s"  "i"  "n"
[16] "g"  " "  "t"  "h"  "a"  "t"  " "  "t"  "r"  "u"  "t"  "h"  " "  "i"  "s"
[31] " "  "a"  " "  "w"  "o"  "m"  "a"  "n"  "-"  "-"
> dataset$next_char[[1]]
[1] "n"
> dataset$next_char[[2]]
[1] "w"
```

Keras Example - Generate Nietzsche's writing with LSTM

- Vectorization (Warning: array X needs lots of memory)

```
X <- array(0, dim = c(length(dataset$sentece), maxlen, length(chars
y <- array(0, dim = c(length(dataset$sentece), length(chars)))

for(i in 1:length(dataset$sentece)){

  X[i, , ] <- sapply(chars, function(x){
    as.integer(x == dataset$sentece[[i]])
  })

  y[i, ] <- as.integer(chars == dataset$next_char[[i]])

}
```

Keras Example - Generate Nietzsche's writing with LSTM

- Model Definition

```
model <- keras_model_sequential()

model %>%
  layer_lstm(units = 128, input_shape = c(maxlen, length(chars))) %>
  layer_dense(length(chars)) %>%
  layer_activation("softmax")

optimizer <- optimizer_adam(lr = 0.01)

model %>% compile(
  loss = "categorical_crossentropy",
  optimizer = optimizer
)
summary(model)
```

Keras Example - Generate Nietzsche's writing with LSTM

```
> summary(model)
________________________________________________________________________________
Layer (type)                        Output Shape                    Param #
================================================================================
lstm (LSTM)                         (None, 128)                     95232
________________________________________________________________________________
dense (Dense)                       (None, 57)                      7353
________________________________________________________________________________
activation (Activation)             (None, 57)                      0
================================================================================
Total params: 102,585
Trainable params: 102,585
Non-trainable params: 0
```

Keras Examples - Generate Nietzsche's writing with LSTM

- Training and Evaluation

```
# large temperature means more uniform from character set
sample_mod <- function(preds, temperature = 1) {
  preds <- log(preds) / temperature
  exp_preds <- exp(preds)
  preds <- exp_preds / sum(exp(preds))

  rmultinom(1, 1, preds) %>%
    as.integer() %>%
    which.max()
}
```

Keras Examples - Generate Nietzsche's writing with LSTM

```
set.seed(123)
diversity <- 0.5

for(iteration in 1:20){
    cat(sprintf("iteration: %02d ---------------\n\n", iteration))
    model %>% fit(
        X, y,
        batch_size = 128,
        epochs = 1
    )

    start_index <- sample(1:(length(text) - maxlen), size = 1)
    sentence <- text[start_index:(start_index + maxlen - 1)]
    generated <- ""
```

Keras Examples - Generate Nietzsche's writing with LSTM

```
  # Generate 400 chars with randomly chosen sequence from train da
  for(i in 1:400){

    x <- sapply(chars, function(x){
      as.integer(x == sentence)
    })
    x <- array_reshape(x, c(1, dim(x)))

    preds <- predict(model, x)
    next_index <- sample_mod(preds, diversity)
    next_char <- chars[next_index]

    generated <- str_c(generated, next_char, collapse = "")
    sentence <- c(sentence[-1], next_char)

  }

  cat(generated)
  cat("\n\n")
}
```

Keras Example - Generate Nietzsche's writing with LSTM

```
iteration: 01 ---------------

200284/200284 [==============================] - 131s 654us/sample - loss: 2.0020
l in the stands himself his the most to it be the something great and discoper and men to the doment of the master of all
the worth of the strenght of a be man to shilitate to generations of the may not greated men which the such the been which a
can present of the confictary in and a conality, man were is us the mon reancessity to pleasent, which the command of the
haman the the will the are the

iteration: 02 ---------------

200284/200284 [==============================] - 124s 617us/sample - loss: 1.6348
y been the otherly, as the superson of conscience and often the subject of all as the sense of a presenticty, the fing the
worker thonges--the the morality is the subjection has the self-canterable conception of the out of in the makes a distained
to readay tante of the our philosopher and presertion, and the stands the thing in the believed the presently and the really
all science of the parting

iteration: 03 ---------------

200284/200284 [==============================] - 124s 620us/sample - loss: 1.5450
, in a strong and early, and the origination of the means of the sense of his supprisive from the long sensation--and
purpose of the part and the serseed and the discovers of the sersons of the forms and the best to forms of the existence,
and endire and without the possible in a truth and without the spirit living certain simply, of sensition, that so for the
thing the sense of the life in a phil
```

Keras Example - Generate Nietzsche's writing with LSTM

```
iteration: 18 ---------------

200284/200284 [==============================] - 123s 616us/sample - loss: 1.3570
ht of mespolling who makes a cause of the ideal the pointing attained and sense--ow. it is not hore, in the same sympathy
and all the same man of the proper out
"the great of the solution and all his faith of the conditions are a seem the world to them a famained to the artificith and
factation of his classible and should under substated
are the end. but is always its seem and one constitution tha

iteration: 19 ---------------

200284/200284 [==============================] - 124s 619us/sample - loss: 1.3532
 certain constitution of the value and solitude of the good" and in the soll enough to even in the distrust is an and
instrumnes of the consequently the relations, my will seem that is their enougle of easy, for instance with all the
consequently not the devire the little read and and the resolution of the remotting of the contempt and things of strives to
have to be proceation, and do not be our

iteration: 20 ---------------

200284/200284 [==============================] - 125s 626us/sample - loss: 1.3525
 themselves has the best and the morals to the strong to the physical thing of the relations of the surprise and the
individual more delicate have
something and god is "not become free soul to the german standarician has the particulan instinct the contemplation to the
world, in the morality of the prospiring to have as it cannot from the stake of the means of the one are not be
contemplation of t
```

Reference

- https://keras.rstudio.com/ (R interface to Keras)
- He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.
- https://tensorflow.rstudio.com/keras/articles/examples/lstm_text_generation.html
- https://de.wikipedia.org/wiki/Jenseits_von_Gut_und_B%C3%B6se_(Nietzsche)