Deep Neural Networks with Keras in R

A. Sanchez, F. Reverter and E. Vegas

Deep Learning with R

- We assume familiarity with:
  - Statistical learning basics
    - Classification/Regression problems,
    - Model building, Model evaluation
  - Deep Neural Networks (even more basic)
    - Artificial neural networks, layers, activation function, gradient methods.
- And we focus on **how to build and use deep neural networks using R**

Outline

- Which software (and Hw) for Deep Learning
  - Python vs R
  - Tensorflow, Keras, Pytorch and many more
- Deep learning, vectorization and Tensors
  - Vectorization for efficient computation
  - Tensors for data representation and manipulation
- The Machine learning (Dl?) Workflow
  - The general ML workflow
  - A Keras pipeline

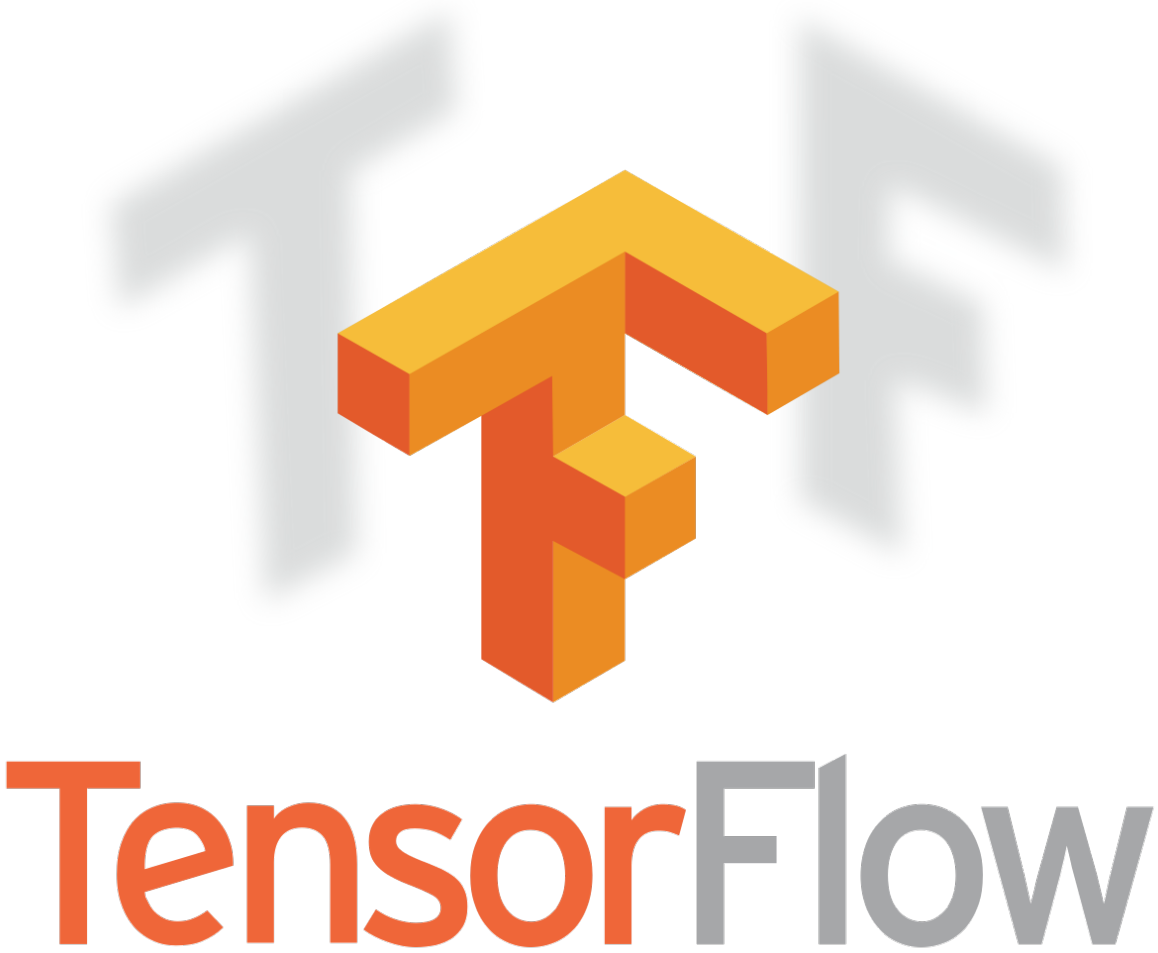
Software for Deep Learning

1. TensorFlow
2. PyTorch
3. Keras
4. MXNet
5. Caffe
6. Theano
7. Microsoft Cognitive Toolkit (CNTK)

TensorFlow

- An open-source deep learning framework
- Developed by Google
- with a comprehensive ecosystem of tools and resources for building and deploying machine learning models.

link to TensorFlow



Pytorch

- Open-source deep learning framework
- known for its dynamic computational graph feature
- and user-friendly interface
- developed by Facebook's AI Research lab.
- link to PyTorch

Keras

- High-level neural networks API
- written in Python
- runs on top of TensorFlow, CNTK, or Theano,
- emphasizes simplicity and ease of use.
- link to Keras

Keras

Deep learning with R

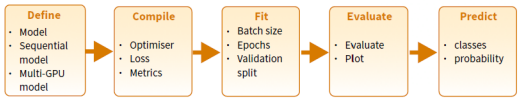
- As of 2023, common approach is
  - Tensorflow + Keras from within R
  - Using python in the background
- Multiple possible installations

- Possibly, the simplest is go to:
  - TensorFlow for R site

Keras

The Keras pipeline

- Training and using a model in keras is intended to be done through the usual steps of a ML worfflow



A keras cheat sheet

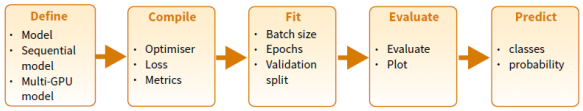
Deep Learning with Keras : : CHEAT SHEET

Keras TensorFlow

Intro

Keras is a high-level neural networks API developed with a focus on enabling fast experimentation. It supports multiple back-ends, including TensorFlow, CNTK and Theano.

TensorFlow is a lower level mathematical library for building deep neural network architectures. The `keras` R package makes it easy to use Keras and TensorFlow in R.



https://tensorflow.rstudio.com

https://www.manning.com/books/deep-learning-with-r-second-edition

The "Hello, World!" of deep learning

INSTALLATION

The `keras` R package uses the Python keras library. You can install all the prerequisites directly from R.

https://keras.rstudio.com/reference/install_keras.html

```
library(keras)
install_keras()
```

See ?install_keras for GPU instructions

This installs the required libraries in an Anaconda environment or virtual environment 'r-tensorflow'.

Working with keras models

DEFINE A MODEL

keras_model() Keras Model

keras_model_sequential() Keras Model composed of a linear stack of layers

multi_gpu_model() Replicates a model on different GPUs

COMPILE A MODEL

compile(object, optimizer, loss, metrics = NULL) Configure a Keras model for training

FIT A MODEL

fit(object, x = NULL, y = NULL, batch_size = NULL, epochs = 10, verbose = 1, callbacks = NULL, ...) Train a Keras model for a fixed number of epochs (iterations)

fit_generator() Fits the model on data yielded batch-by-batch by a generator

train_on_batch() test_on_batch() Single gradient update or model evaluation over one batch of samples

EVALUATE A MODEL

evaluate(object, x = NULL, y = NULL, batch_size = NULL) Evaluate a Keras model

evaluate_generator() Evaluates the model on a data generator

PREDICT

predict() Generate predictions from a Keras model

predict_proba() and **predict_classes()** Generates probability or class probability predictions for the input samples

predict_on_batch() Returns predictions for a single batch of samples

predict_generator() Generates predictions for the input samples from a data generator

OTHER MODEL OPERATIONS

summary() Print a summary of a Keras model

export_savedmodel() Export a saved model

get_layer() Retrieves a layer based on either its name (unique) or index

pop_layer() Remove the last layer in a model

save_model_hdf5(); load_model_hdf5() Save/ Load models using HDF5 files

serialize_model(); unserialize_model() Serialize a model to an R object

clone_model() Clone a model instance

freeze_weights(); unfreeze_weights() Freeze and unfreeze weights

CORE LAYERS

layer_input() Input layer

layer_dense() Add a densely-connected NN layer to an output

layer_activation() Apply an activation function to an output

layer_dropout() Applies Dropout to the input

layer_reshape() Reshapes an output to a certain shape

layer_permute() Permute the dimensions of an input according to a given pattern

layer_repeat_vector() Repeats the input n times

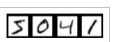
layer_lambda(object, f) Wraps arbitrary expression as a layer

layer_activity_regularization() Layer that applies an update to the cost function based input activity

layer_masking() Masks a sequence by using a mask value to skip timesteps

layer_flatten() Flattens an input

TRAINING AN IMAGE RECOGNIZER ON MNIST DATA



```
# input layer: use MNIST images
mnist <- dataset_mnist()
x_train <- mnist$train$x; y_train <- mnist$train$y
x_test <- mnist$test$x; y_test <- mnist$test$y

# reshape and rescale
x_train <- array_reshape(x_train, c(nrow(x_train), 784))
x_test <- array_reshape(x_test, c(nrow(x_test), 784))
x_train <- x_train / 255; x_test <- x_test / 255

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

# defining the model and layers
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_dense(units = 10, activation = 'softmax')

# compile (define loss and optimizer)
model %>% compile(
 loss = 'categorical_crossentropy',
 optimizer = optimizer_rmsprop(),
 metrics = c('accuracy')
)

# train (fit)
model %>% fit(
 x_train, y_train,
 epochs = 30, batch_size = 128,
 validation_split = 0.2
)
model %>% evaluate(x_test, y_test)
model %>% predict_classes(x_test)
```

posit

Available at rstudio github repo

Hello world of deep learning (1)

```
# load packages
library(keras)
# input layer: use MNIST images
mnist <- dataset_mnist()
x_train <- mnist$train$x; y_train <- mnist$train$y
x_test <- mnist$test$x; y_test <- mnist$test$y
```

Hello world of deep learning (2)

```
# reshape and rescale
x_train <- array_reshape(x_train, c(nrow(x_train), 784))
x_test <- array_reshape(x_test, c(nrow(x_test), 784))
x_train <- x_train / 255; x_test <- x_test / 255
y_train <- to_categorical(y_train, 10)
y_test <- to_categorical(y_test, 10)
```

Hello world of deep learning (3)

```
# defining the model and layers
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_dense(units = 10, activation = 'softmax')
```

Hello world of deep learning (4)

```
# compile (define loss and optimizer)
model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)
```

Hello world of deep learning (5)

```
# train (fit)
model %>% fit(
x_train, y_train,
epochs = 30, batch_size = 128,
validation_split = 0.2
)
```

Hello world of deep learning (6)

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

One must-do digression: Tensors

- Deep learning is filled with the word "tensor",
  - Not to talk of *TensorFlow*
- What are Tensors any way?
  - R users: familiar with vectors (1-d arrays) and matrices (2-d arrays).
  - Tensors extend this concept to higher dimensions.
  - Can be seen as multi-dimensional arrays that generalize matrices.

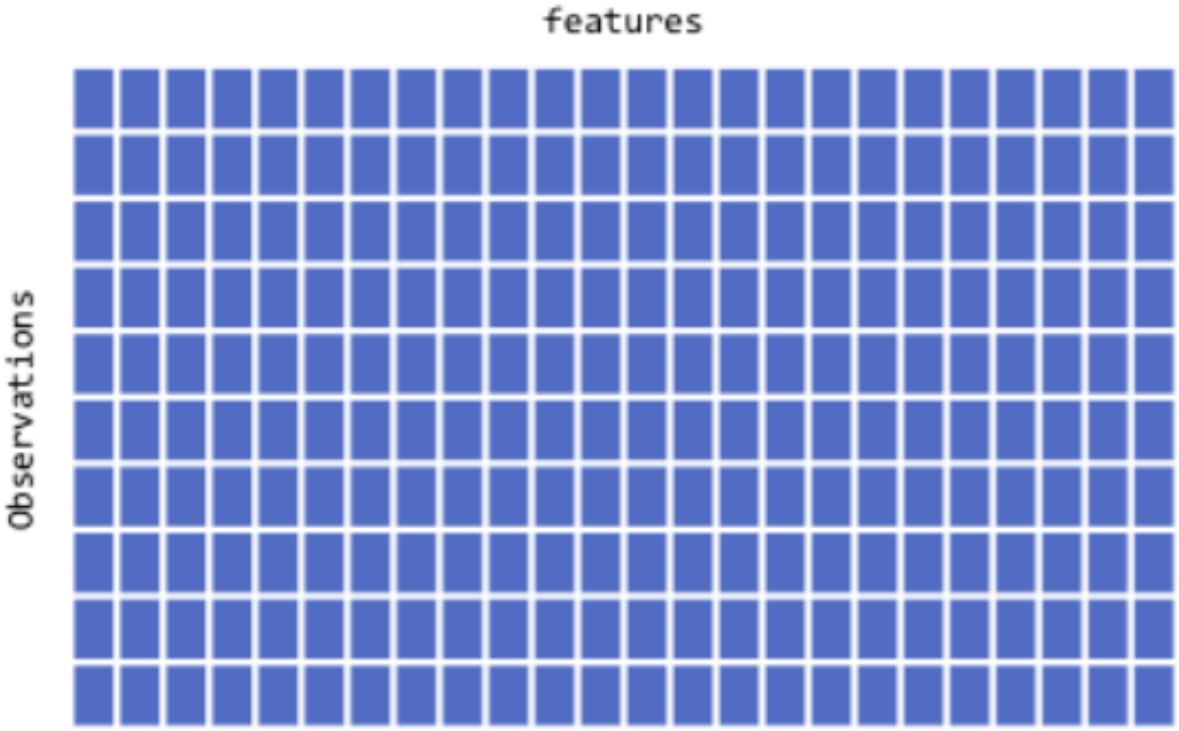
Why tensors?

- Working with tensors has many benefits:
  - **Generalization**: Tensors generalize vectors and matrices to an arbitary number of dimensions,
  - **Flexibility**: can hold a wide range of data types.
  - **Speed**: Use of tensors facilitates fast, parallel processing computations.

One and two dimensional tensors

Vectors:rank-1 tensors.

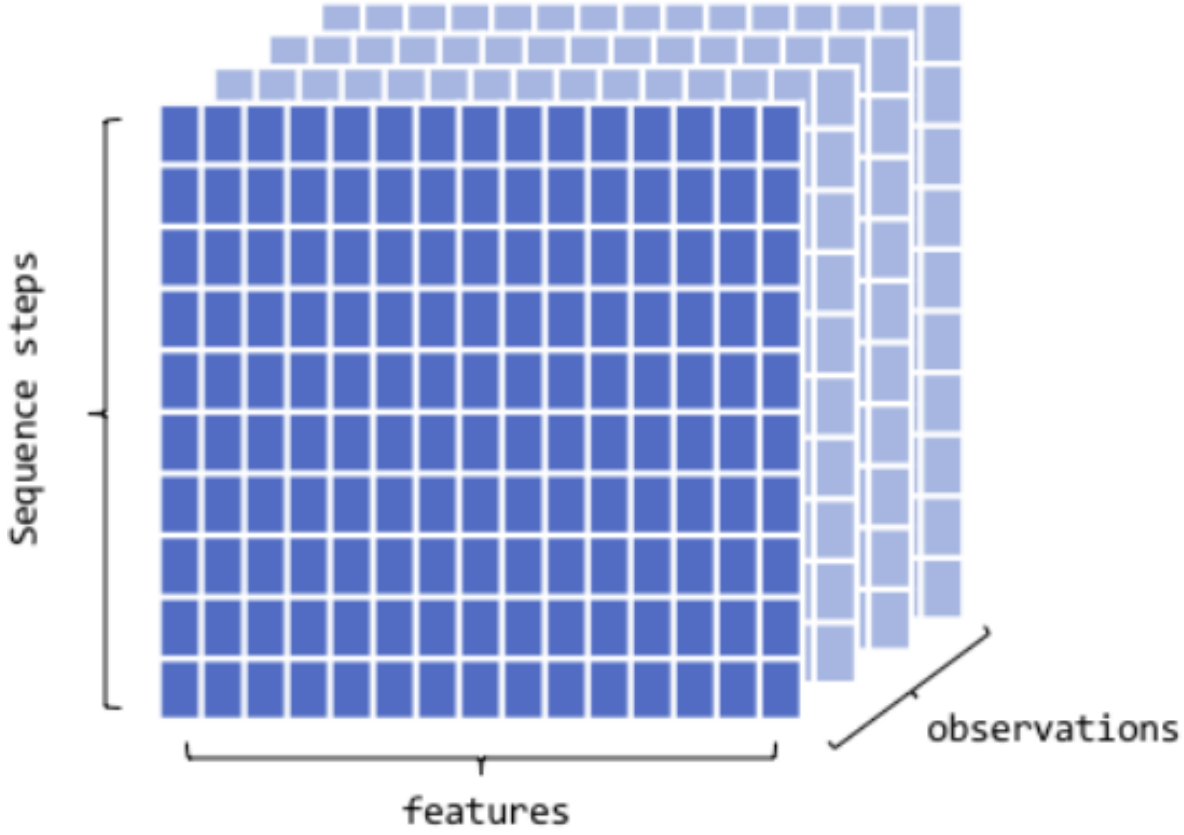
Matrices: rank-2 tensors.



Rank three tensors

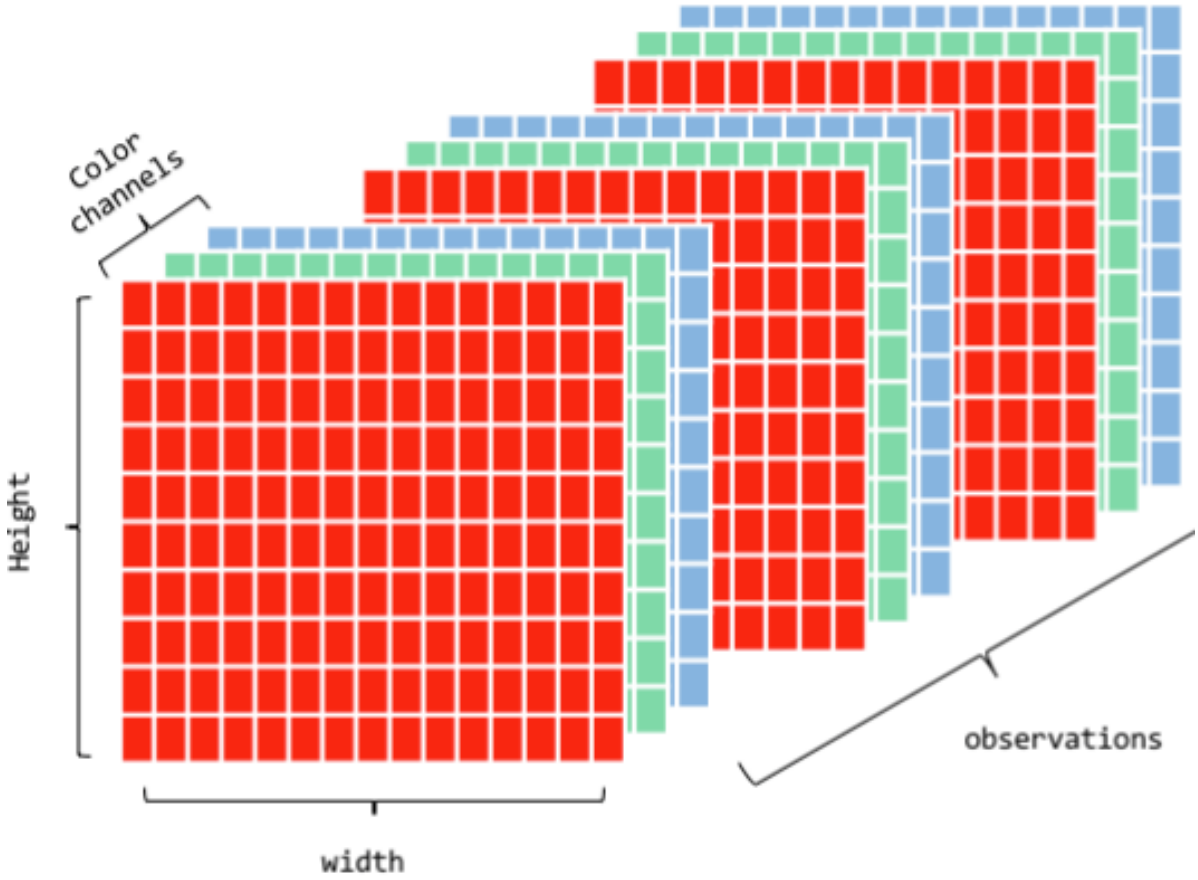
- Arrays in layers.
- Typic use: Sequence data
  - time series, text
  - dim = (observations, seq steps, features)
- Examples

  - 250 days of high, low, and current stock price for 390 minutes of trading in a day; dim = c(250, 390, 3)
  - 1M tweets that can be 140 characters long and include 128 unique characters; dim = c(1M, 140, 128)



Rank four tensors

- Layers of groups of arrays
- Typic use: Image data
  - RGB channels
  - dim = (observations, height, width, color_depth)
  - MNIST data could be seen as a 4D tensor where color_depth = 1

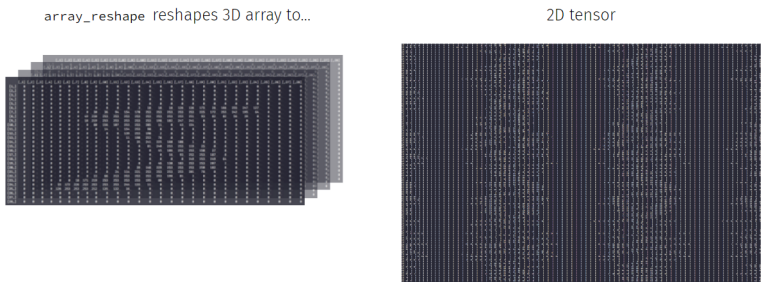


Rank five tensors

- Typic use: Video data
  - samples: 4 (each video is 1 minute long)
  - frames: 240 (4 frames/second)
  - width: 256 (pixels)
  - height: 144 (pixels)
  - channels: 3 (red, green, blue)
- Tensor shape (4, 240, 256, 144, 3)

One can always *reshape*

- Each DNN model has a given architecture which usually requires 2D/3D tensors.
- If data is not in the expected form it can be *reshaped*.



See Deep learning with R for more.

References and Resources

Workshops

- Deep learning with R *Summer course*
- Deep learning with keras and Tensorflow in R (Rstudio conf. 2020)

Books

- Deep learning with R, 2nd edition. F. Chollet

Documents

- 7 Best Deep Learning Frameworks To Watch Out For in 2022