Using TensorFlow and R

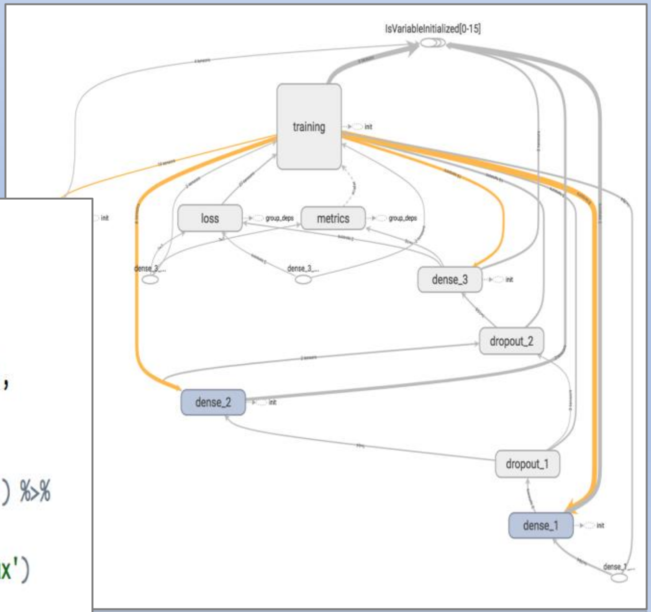
LONDONR

2018-03-27

Andrie de Vries

Solutions Engineer, RStudio

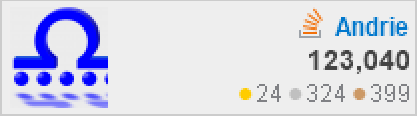
@RevoAndrie



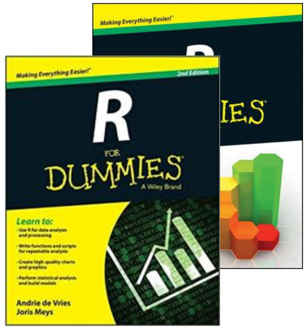

```r

library(keras)

model <- keras_model_sequential() %>%
  layer_dense(units = 128, 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 = 10, activation = 'softmax')

```

Overview

- TensorFlow using R
- Worked example of `keras` in R
- Demo
- Supporting tools
- Learning more



[StackOverflow: andrie](#)
[Twitter: @RevoAndrie](#)
[GitHub: andrie](#)



Slides at https://speakerdeck.com/andrie/londonr-tensorflow

What is TensorFlow

What is TensorFlow

- Originally developed by researchers and engineers working on the Google Brain Team for the purposes of conducting machine learning and deep neural networks research.
- Open source software (Apache v2.0 license)
- Hardware independent
  - CPU (via Eigen and BLAS)
  - GPU (via CUDA and cuDNN)
  - TPU (Tensor Processing Unit)
- Supports automatic differentiation
- Distributed execution and large datasets

What is a tensor?

- Spoiler alert: it's an array

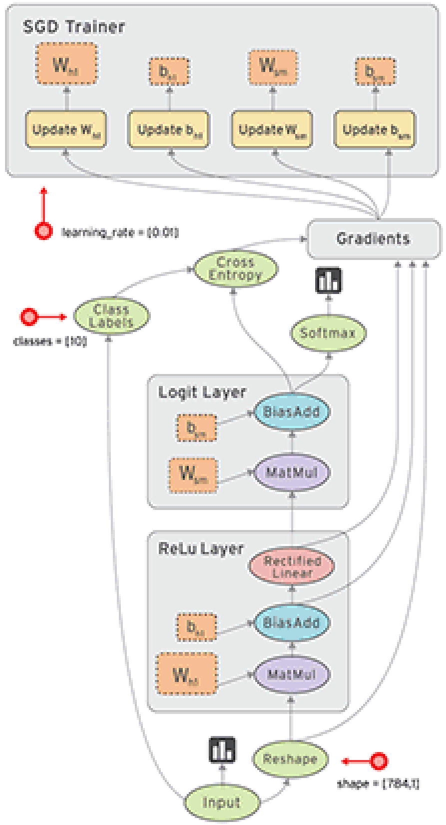
Tensor dimensionality	R object class	Example
0	Vector of length one	Point value
1	Vector	Weights
2	Matrix	Time series
3	Array	Grey scale image
4	Array	Colour images
5	Array	Video

Note that the first dimension is always used for the observations, thus "adding" a dimension

RStudio

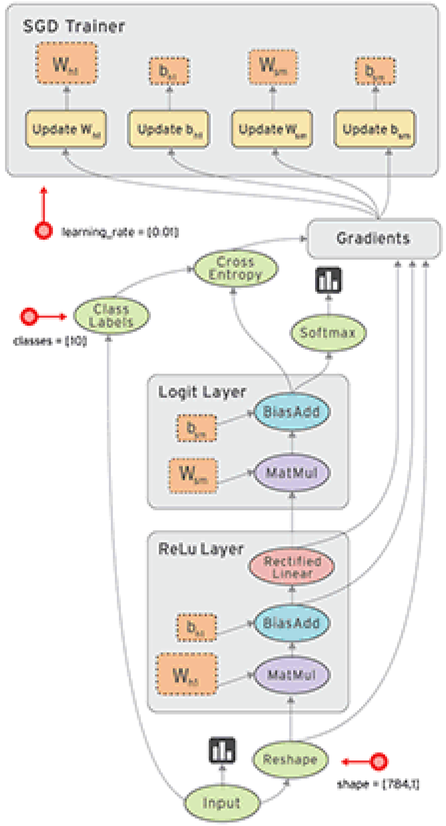
What is tensor flow?

- You define the graph in R
- Graph is compiled and optimized
- Graph is executed on devices
- Nodes represent computations
- Data (tensors) flows between them



Why a dataflow graph?

- Major gains in performance, scalability, and portability
  - Parallelism
    - System runs operations in parallel.
  - Distributed execution
    - Graph is partitioned across multiple devices.
  - Compilation
    - Use the information in your dataflow graph to generate faster code (e.g. fusing operations)
  - Portability
    - Dataflow graph is a language-independent representation of the code in your model (deploy



Uses of TensorFlow

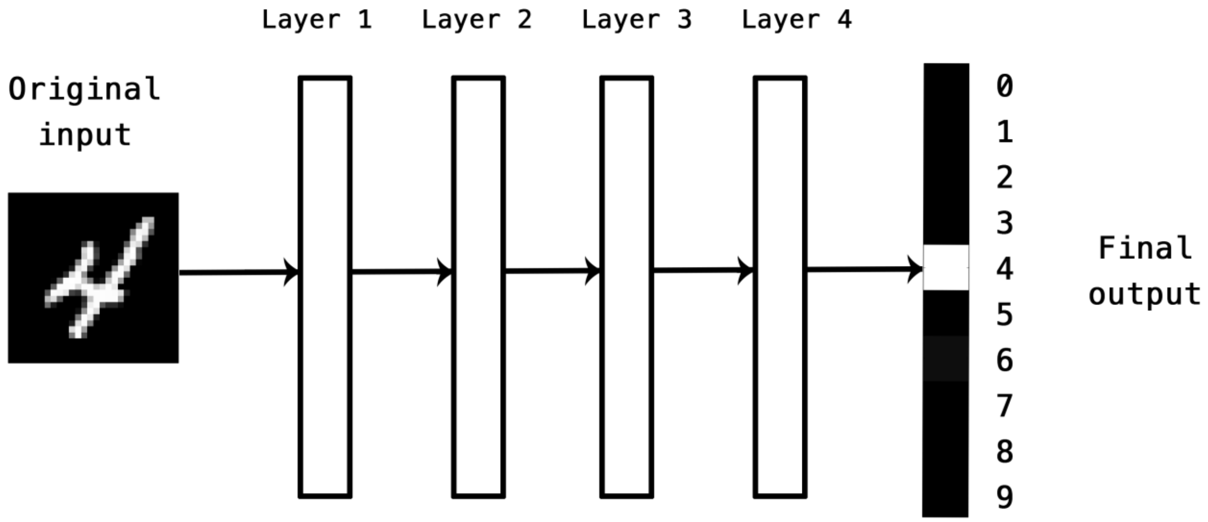
- Image classification
- Time series forecasting
- Classifying peptides for cancer immunotherapy
- Credit card fraud detection using an autoencoder
- Classifying duplicate questions from Quora
- Predicting customer churn
- Learning word embeddings for Amazon reviews

https://tensorflow.rstudio.com/gallery/

What is deep learning

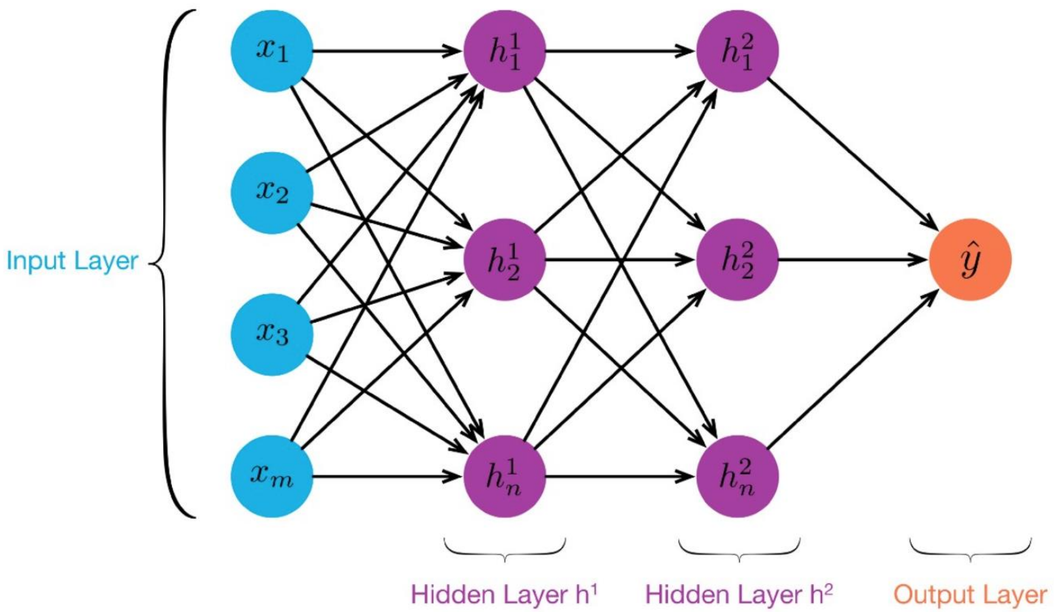
What is deep learning?

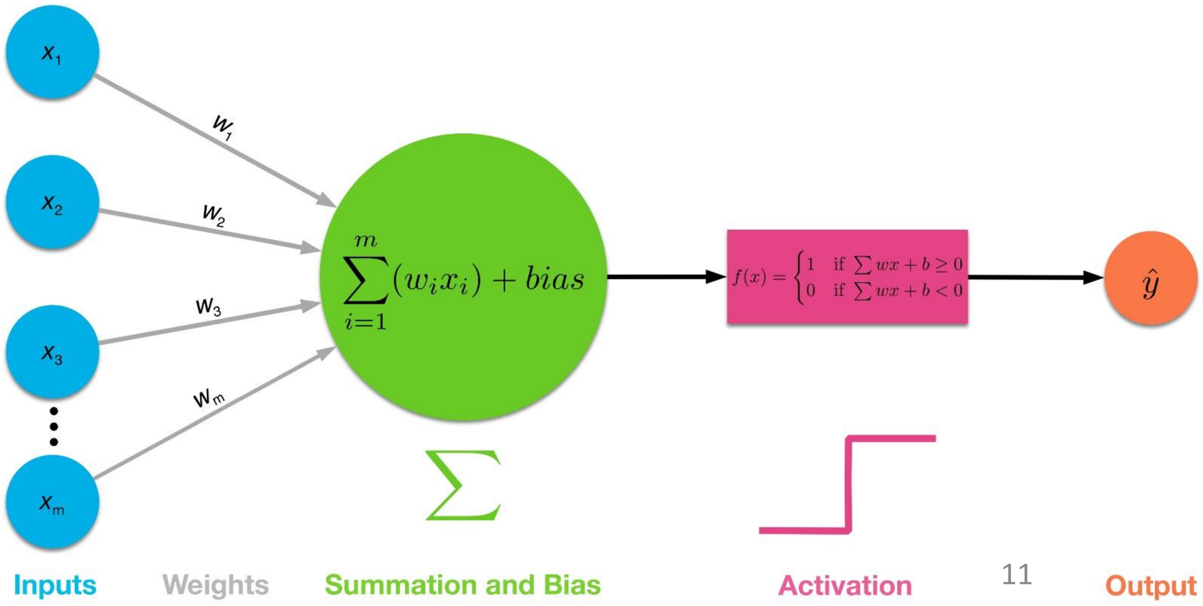
- Input to output via layers of representation



What are layers?

- Data transformation functions parameterized by weights
  - A layer is a geometric transformation function on the data that goes through it (transformations must be differentiable for stochastic gradient descent)
  - Weights determine the data transformation behavior of a layer

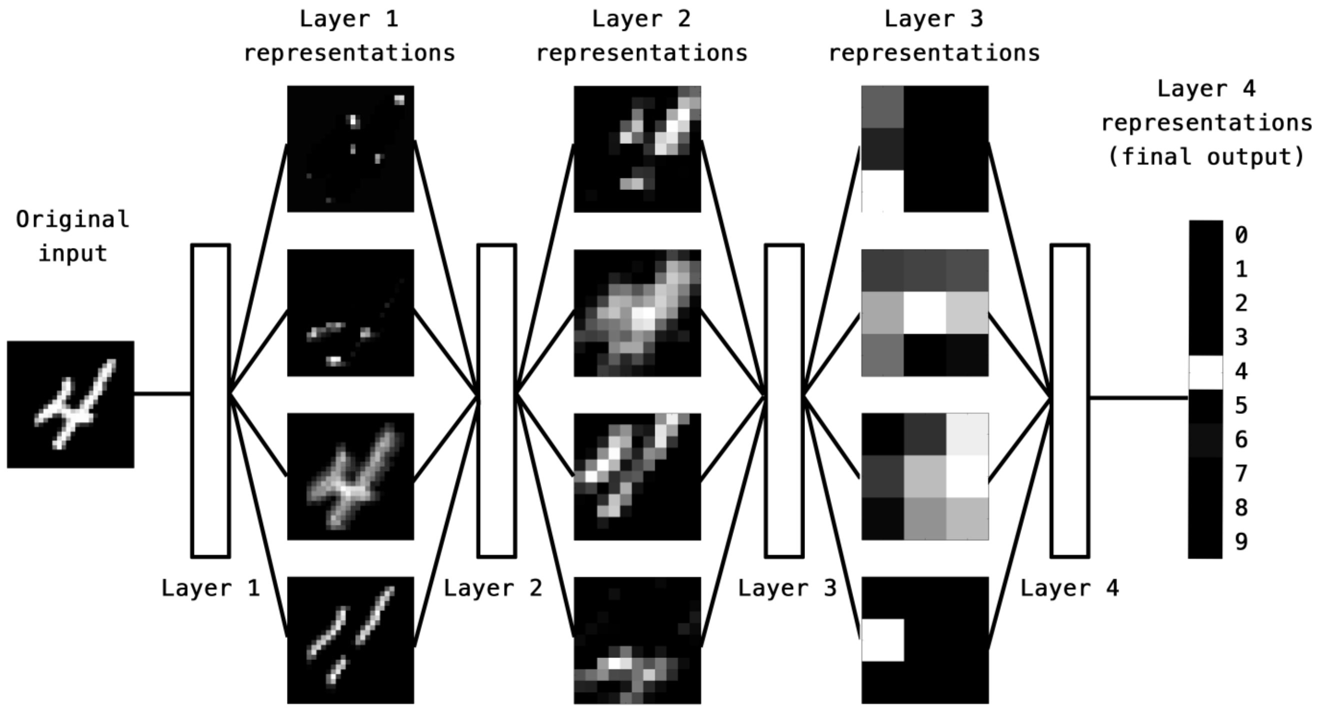




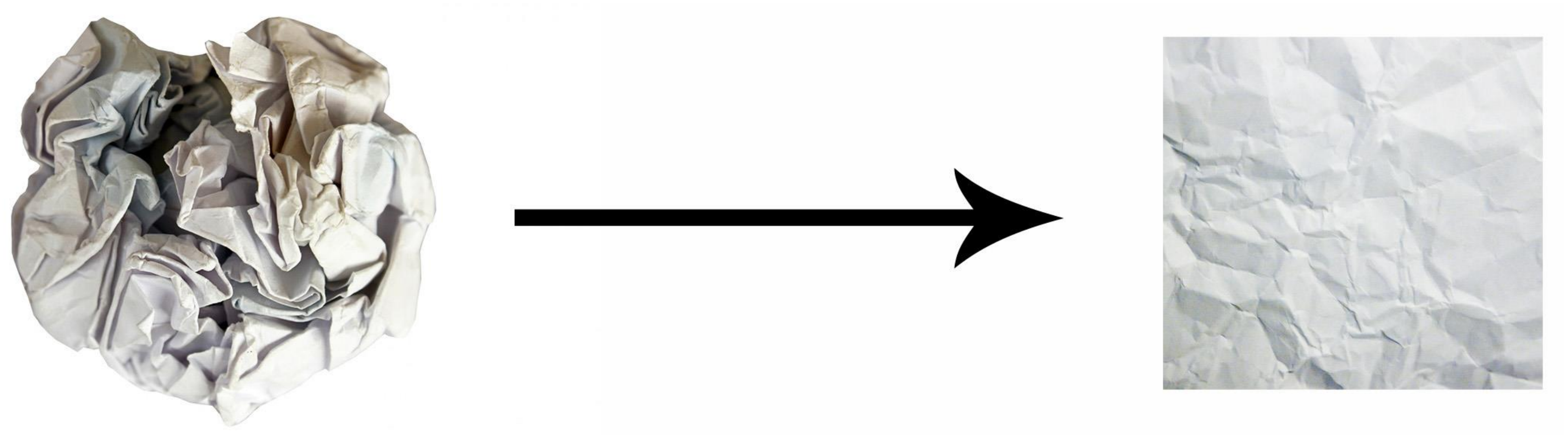
MNIST layers in R

```
library(keras)
model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = 'relu',
                input_shape = c(28,28,1)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_flatten() %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dense(units = 10, activation = 'softmax')
```

MNIST layers of representation

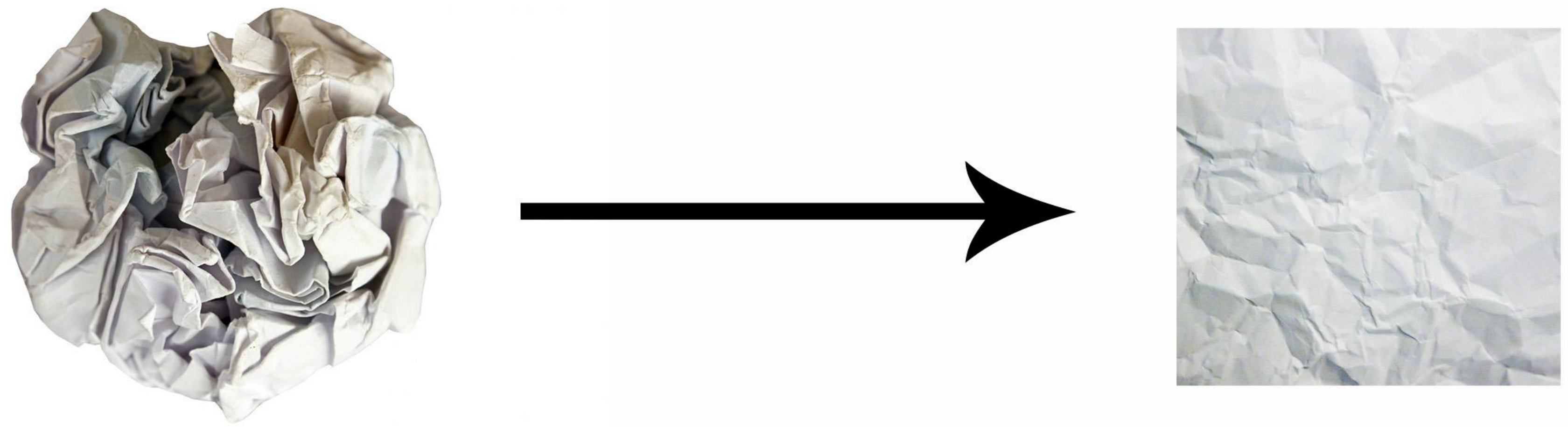


Geometric interpretation



- Deep-learning models are mathematical machines for uncrumpling complicated manifolds of high-dimensional data.
- Deep learning is turning meaning into vectors, into geometric spaces, and then incrementally learning complex geometric transformations that map one space to another.

How can we do this?



- How can we do this with simple parametric models trained with gradient descent?
- We just need
  - **Sufficiently large parametric models**,
  - trained with gradient descent on
  - **sufficiently many examples**

Sufficiently large parametric models

- Simple grayscale digit recognizer model has > 1 million parameters

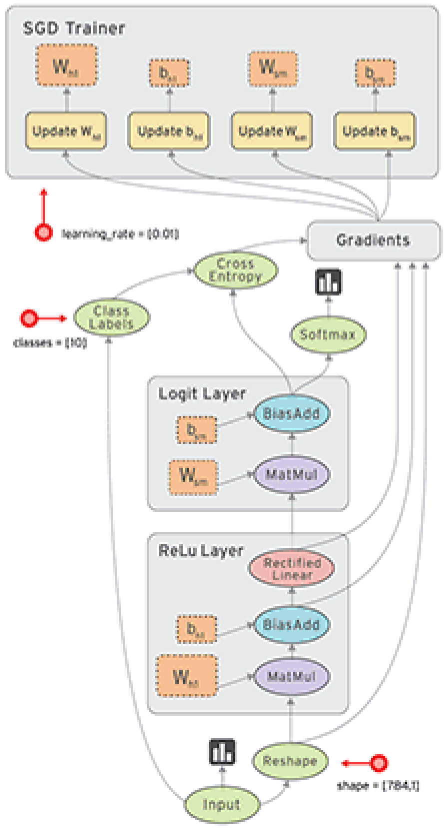
```
Summary(model)
```

```
Layer (type)                     Output Shape                  Param #
=======================================================================
conv2d_3 (Conv2D)                (None, 26, 26, 32)            320
_______________________________________________________________________
conv2d_4 (Conv2D)                (None, 24, 24, 64)            18496
_______________________________________________________________________
max_pooling2d_2 (MaxPooling2D)   (None, 12, 12, 64)            0
_______________________________________________________________________
flatten_2 (Flatten)              (None, 9216)                  0
_______________________________________________________________________
dense_3 (Dense)                  (None, 128)                   1179776
_______________________________________________________________________
dense_4 (Dense)                  (None, 10)                    1290
=======================================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
_______________________________________________________________________
```

TensorFlow using R

Why should R users care about TensorFlow?

- A new **general purpose** numerical computing library
    - Hardware independent
    - Distributed execution
    - Large datasets
    - Automatic differentiation
- **Not all data has to be in RAM**
    - Highly general optimization, e.g. SGD, Adam
- **Robust foundation** for machine and deep learning
- TensorFlow models can be **deployed with C++ runtime**
- R has a lot to offer as an **interface language**



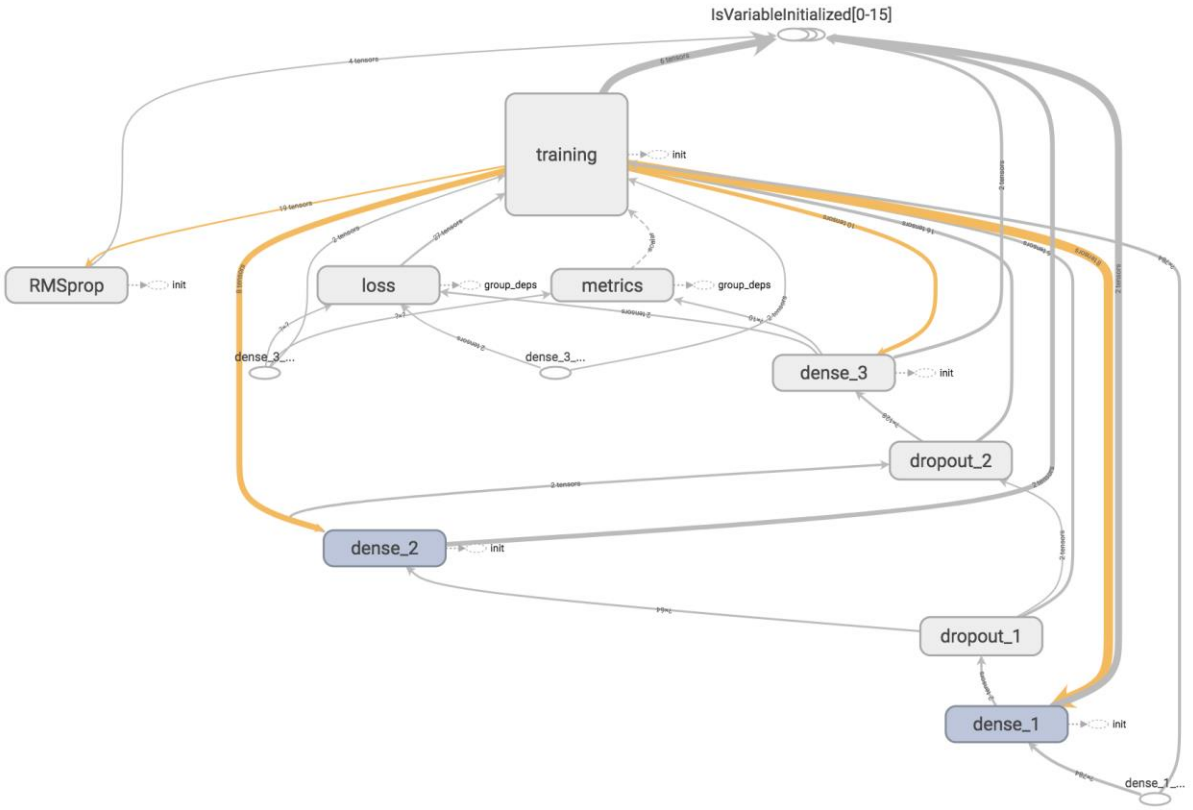
R interface to Tensorflow

- https://tensorflow.rstudio.com

- High-level R interfaces for neural nets and traditional models
- Low-level interface to enable new applications (e.g. Greta)
- Tools to facilitate productive workflow / experiment management
- Straightforward access to GPUs for training models
- Breadth and depth of educational resources

Graph is generated automatically from R

```r
library(keras)

model <- keras_model_sequential() %>%
  layer_dense(units = 128, 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 = 10, activation = 'softmax')
```



TensorFlow APIs

- Distinct interfaces for various tasks and levels of abstraction

Keras API

The Keras API for TensorFlow provides a high-level interface for neural networks, with a focus on enabling fast experimentation.

Estimator API

The Estimator API for TensorFlow provides high-level implementations of common model types such as regressors and classifiers.

Core API

The Core TensorFlow API is a lower-level interface that provides full access to the TensorFlow computational graph.

tensorflow

- Low level access to TensorFlow graph operations
  https://tensorflow.rstudio.com/tensorflow

```
library(tensorflow)

W <- tf$Variable(tf$random_uniform(shape(1L), -1.0, 1.0))
b <- tf$Variable(tf$zeros(shape(1L)))
y <- W * x_data + b

loss <- tf$reduce_mean((y - y_data) ^ 2)
optimizer <- tf$train$GradientDescentOptimizer(0.5)
train <- optimizer$minimize(loss)

sess = tf$Session()
sess$run(tf$global_variables_initializer())

for (step in 1:200)
  sess$run(train)
```

tfestimators

- High level API for TensorFlow models (https://tensorflow.rstudio.com/tfestimators/)

```
library(tfestimators)

linear_regressor()
linear_classifier()
dnn_regressor()
dnn_classifier()
dnn_linear_combined_regressor()
dnn_linear_combined_classifier()
```

keras

- High level API for neural networks (https://tensorflow.rstudio.com/keras/)

```
library(keras)

model <- keras_model_sequential() %>%
  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 = 10, activation = 'softmax')
```

Worked example using keras

Steps in building a keras model

Define

- Model
- Sequential model
- Multi-GPU model

Compile

- Optimiser
- Loss
- Metrics

Fit

- Batch size
- Epochs
- Validation split

Evaluate

- Evaluate
- Plot

Predict

- classes
- probability

Cheat sheet: https://github.com/rstudio/cheatsheets/raw/master/keras.pdf

Keras data pre-processing

- Transform input data into tensors

```
library(keras)

# Load MNIST images datasets (built-in to Keras)
c(c(x_train, y_train), c(x_test, y_test)) %<-% dataset_mnist()

# Flatten images and transform RGB values into [0,1] range
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

# Convert class vectors to binary class matrices
y_train <- to_categorical(y_train, 10)
y_test <- to_categorical(y_test, 10)
```

Datasets are downloaded from S3 buckets and cached locally

Use %<-% to assign to multiple objects

TensorFlow expects row-primary tensors. Use array_reshape() to convert from (column-primary) R arrays

Normalize to [-1; 1] range for best results

Ensure your data is numeric only, e.g. by using one-hot encoding

RStudio

Model definition

Sequential models are very common, but you can have multiple inputs – use keras_model()

```
model <- keras_model_sequential()  %>%
  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 = 10, activation = 'softmax')

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

Many different layers and activation types are available. You can also define your own.

Compilation modifies in place. Do not re-assign result to object.

Note: Models are modified in-place

- Object semantics are not by-value! (as is conventional in R)
  - Keras models are directed acyclic graphs of layers whose state is updated during training.
  - Keras layers can be shared by multiple parts of a Keras model.

```
# Modify model object in place (note that it is not assigned back to)

model %>% compile(
  optimizer = 'rmsprop',
  loss = 'binary_crossentropy',
  metrics = c('accuracy')
)
```

In the compile() step, do not assign the result, i.e. modify in place

Keras: Model training

- Feeding mini-batches of data to the model thousands of times

```
history <- model %>% fit(
  x_train, y_train,
  batch_size = 128,
  epochs = 10,
  validation_split = 0.2
)
```

- Feed 128 samples at a time to the model (`batch_size = 128`)
- Traverse the input dataset 10 times (`epochs = 10`)
- Hold out 20% of the data for validation (`validation_split = 0.2`)

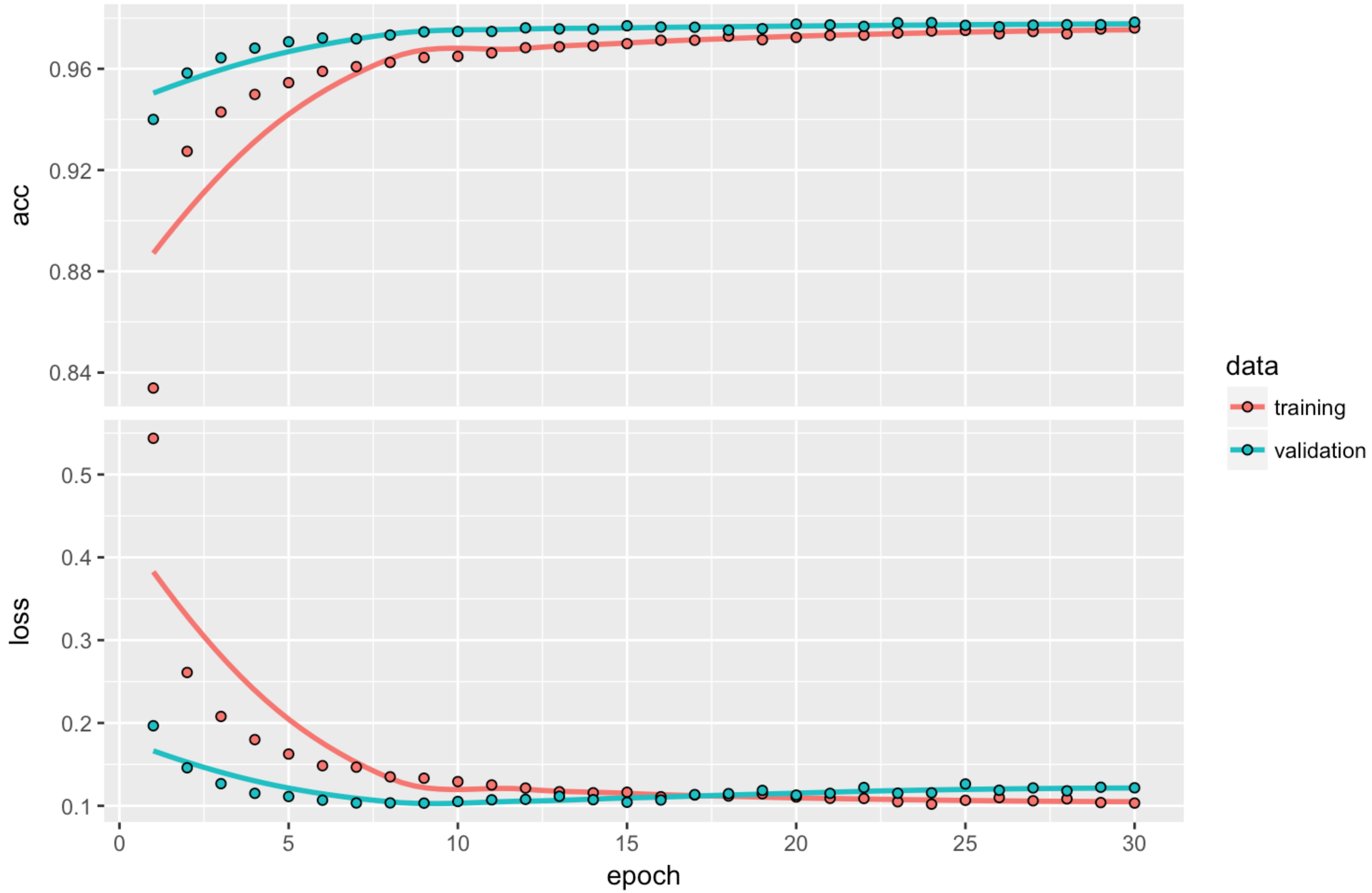
Evaluation and prediction

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

$loss
[1] 0.1078904

$acc
[1] 0.9815
```

```
model %>% predict_classes(x_test[1:100,])

  [1] 7 2 1 0 4 1 4 9 5 9 0 6 9 0 1 5 9 7 3 4 9 6 6 5 4 0 7 4 0 1 3 1 3 4 7
 [36] 2 7 1 2 1 1 7 4 2 3 5 1 2 4 4 6 3 5 5 6 0 4 1 9 5 7 8 9 3 7 4 6 4 3 0
 [71] 7 0 2 9 1 7 3 2 9 7 7 6 2 7 8 4 7 3 6 1 3 6 9 3 1 4 1 7 6 9
```

Easy plotting of fitting history

```
plot(history)
```



Demo

File Edit Code View Plots Session Build Debug Profile Tools Help

Go to file/function | Addins | 2018-02 IBM Index TensorFlow

mnist_mlp.R

Source on Save | Run | Source

```r

library(keras)

# Load MNIST images datasets (built in to Keras)
c(c(x_train, y_train), c(x_test, y_test)) %<-% dataset_mnist()

# Flatten images and transform RGB values into [0,1] range
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

# Convert class vectors to binary class matrices
y_train <- to_categorical(y_train, 10)
y_test <- to_categorical(y_test, 10)

# Define the model
model <- keras_model_sequential()  %>%
  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 = 10, activation = 'softmax')

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

# Print a summary
summary(model)

# Fit the model
history <- model %>% fit(
  x_train, y_train,
  batch_size = 128,
  epochs = 10,
  validation_split = 0.2
)

# Plot the training history
```

2:1 (Top Level) | R Script

Console | Terminal

C:/Users/apdev/OneDrive/Conferences/2018-02 IBM Index TensorFlow/

```
Restarting R session...

>
```

Environment | History | Connections

Files | Plots | Packages | Help | Viewer

Supporting tools

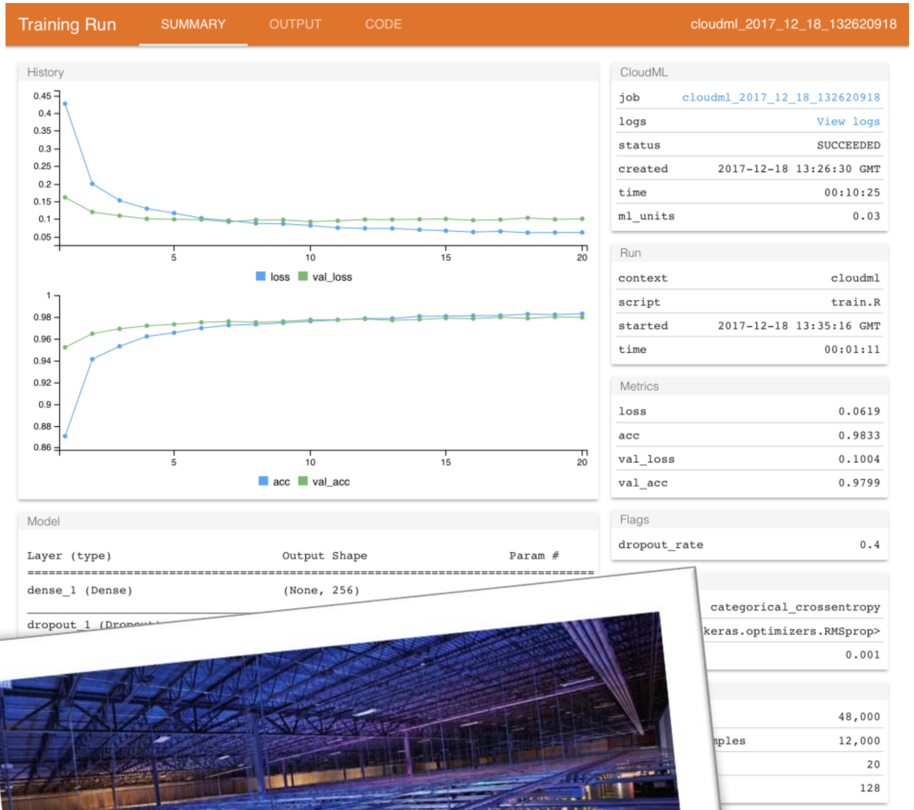
tfruns

- https://tensorflow.rstudio.com/tools/tfruns/
- Successful deep learning requires a huge amount of experimentation.
- This requires a systematic approach to conducting and tracking the results of experiments.
- The `training_run()` function is like the `source()` function, but it automatically tracks and records output and metadata for the execution of the script:


```
library(tfruns)
training_run("mnist_mlp.R")
```

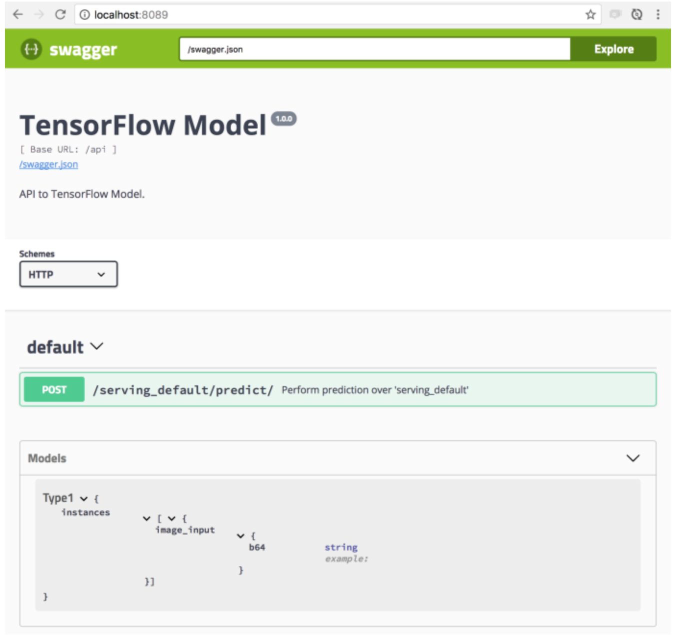
cloudml

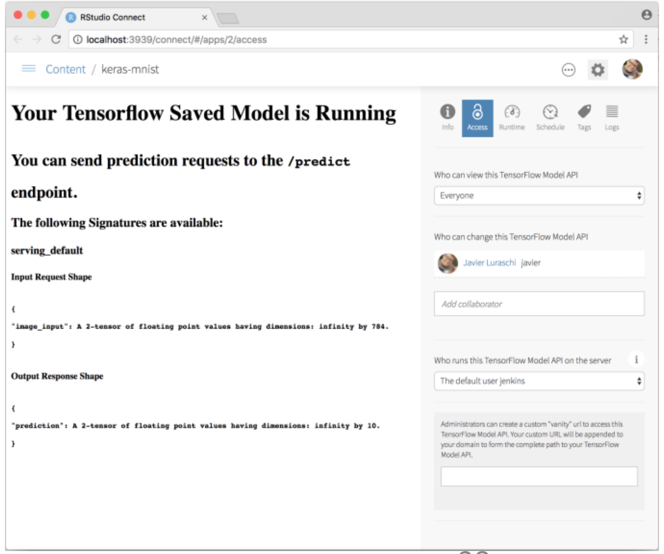
- https://tensorflow.rstudio.com/tools/cloudml/
- Scalable training of models built with the `keras`, `tfestimators`, and `tensorflow` R packages.
- On-demand access to training on GPUs, including Tesla P100 GPUs from NVIDIA®.
- Hyperparameter tuning to optimize key attributes of model architectures in order to maximize predictive accuracy.



tfdeploy

- https://tensorflow.rstudio.com/tools/tfdeploy/
- TensorFlow was built from the ground up to enable deployment using a low-latency C++ runtime.
- Deploying TensorFlow models requires no runtime R or Python code.
- Key enabler for this is the TensorFlow SavedModel format:
  - *a **language-neutral format***
  - *enables higher-level tools to **produce, consume and transform** models.*
- TensorFlow models can be deployed to servers, embedded devices, mobile phones, and even to a web browser!

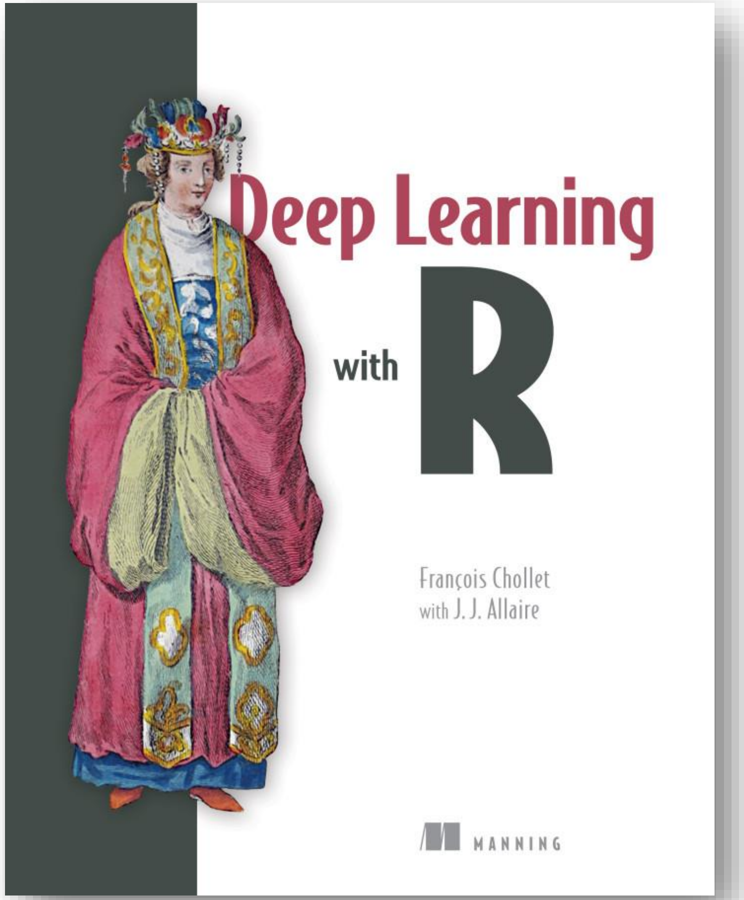




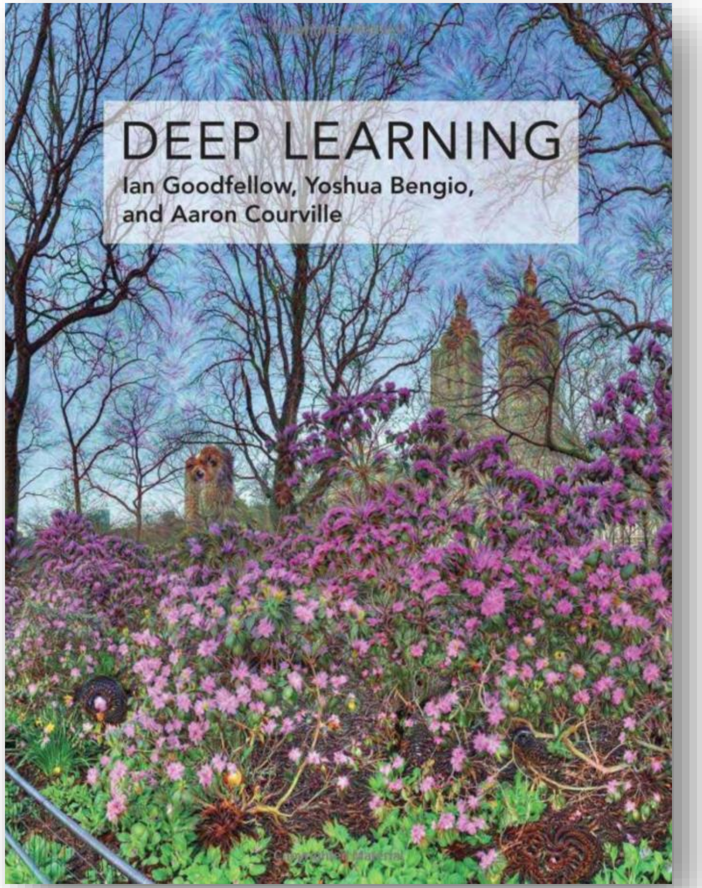
Resources

Recommended reading

Chollet and Allaire



Goodfellow, Bengio & Courville



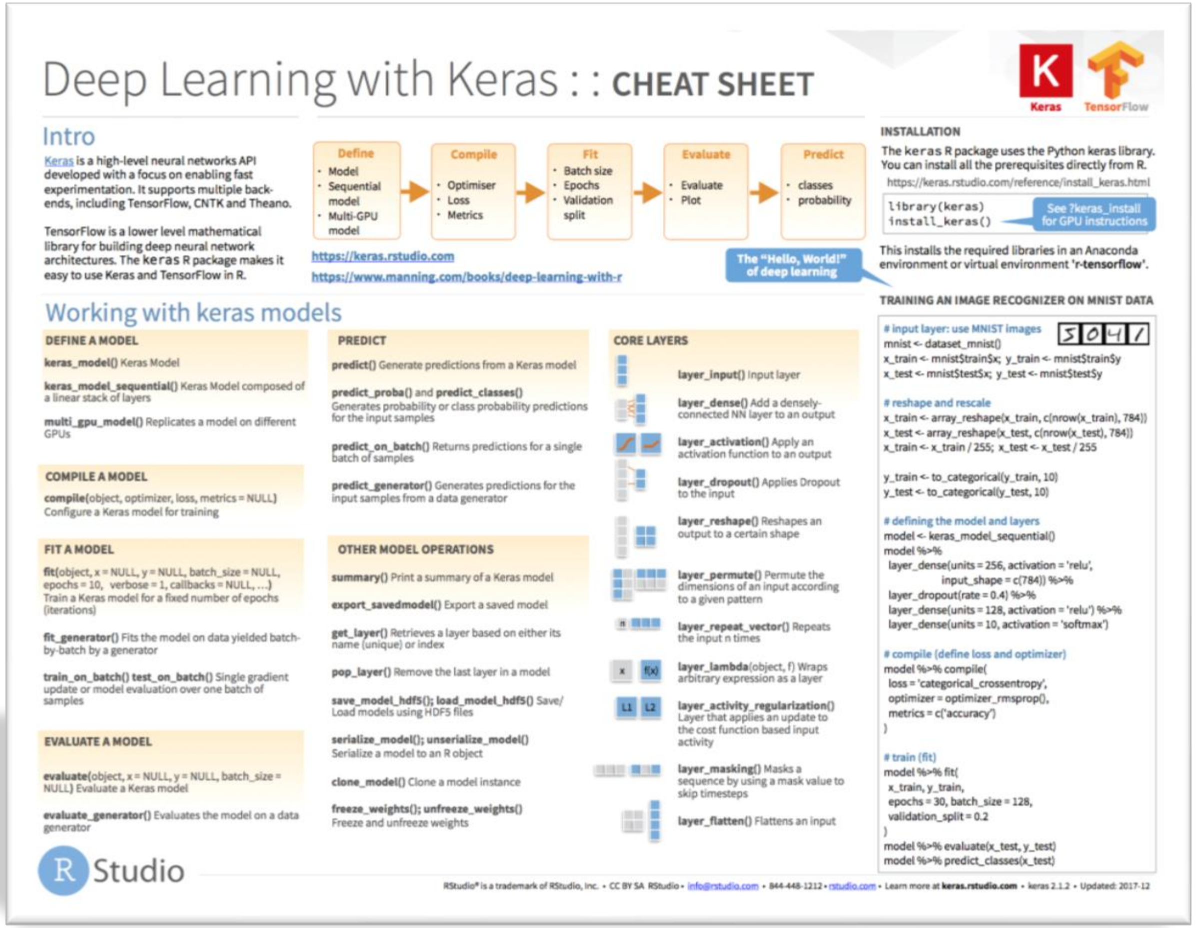
R examples in the gallery

- https://tensorflow.rstudio.com/gallery/

  - Image classification on small datasets
  - Time series forecasting with recurrent networks
  - Deep learning for cancer immunotherapy
  - Credit card fraud detection using an autoencoder
  - Classifying duplicate questions from Quora
  - Deep learning to predict customer churn
  - Learning word embeddings for Amazon reviews
  - Work on explainability of predictions

Keras for R cheat sheet

https://github.com/rstudio/cheatsheets/raw/master/keras.pdf



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.

Define
- Model
- Sequential model
- Multi-GPU model

Compile
- Optimiser
- Loss
- Metrics

Fit
- Batch size
- Epochs
- Validation split

Evaluate
- Evaluate
- Plot

Predict
- classes
- probability

https://keras.rstudio.com
https://www.manning.com/books/deep-learning-with-r

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 ?keras_install 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)
```

R Studio

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at keras.rstudio.com • keras 2.1.2 • Updated: 2017-12

rstudio::conf videos

- Keynote: Machine Learning with TensorFlow and R
  - https://www.rstudio.com/resources/videos/machine-learning-with-tensorflow-and-r/

About the Speaker

J.J. Allaire
Founder and CEO, RStudio

J.J. is the maintainer of the R interfaces to TensorFlow and Keras.

Summary

Summary

TensorFlow APIs

Package	Description
keras	Interface for neural networks, focus on fast experimentation.
tfestimators	Implementations of common model types, e.g. regressors and classifiers.
tensorflow	Low-level interface to the TensorFlow computational graph.

Supporting tools

Package	Description
tfdatasets	Scalable input pipelines for TensorFlow models.
tfruns	Track, visualize, and manage TensorFlow training runs and experiments.
tfdeploy	Tools designed to make exporting and serving TensorFlow models easy.
cloudml	R interface to Google Cloud Machine Learning Engine.

Summary

- TensorFlow is a new **general purpose numerical computing** library with lots to offer the R community.

- Deep learning has made **great progress** and will likely **increase in importance** in various fields in the coming years.

- R now has a **great set of APIs** and **supporting tools** for using TensorFlow and doing deep learning.

Slides at https://speakerdeck.com/andrie/londonr-tensorflow