Keras

Prof. Gheith Abandah

References:

- François Chollet, *Deep Learning with Python*, Manning Pub. 2018
- *Introduction to Keras by Francois Chollet*, March 9th, 2018 (pdf)
- TensorFlow, *Keras*, https://www.tensorflow.org/guide/keras

Introduction

- YouTube Video: *Keras Explained* from Siraj Raval

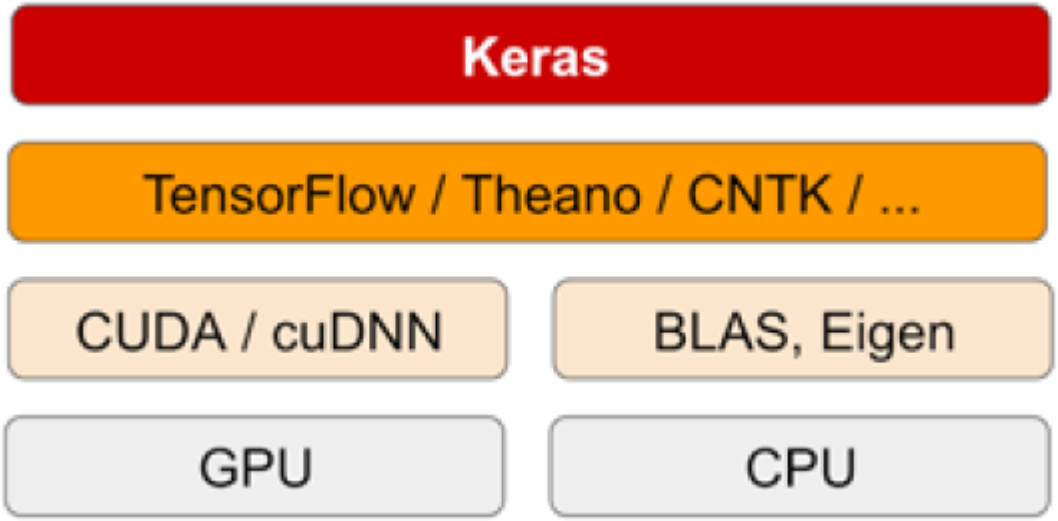
https://youtu.be/j_pJmXJwMLA

Outline

1. Introduction
2. Keras API Styles
3. TensorFlow Keras
4. The Sequential Model
5. Example – MNIST
6. Fashion-MNIST Example
7. Tutorials

1. Introduction

- **Keras** is a high-level API to build and train deep learning models.

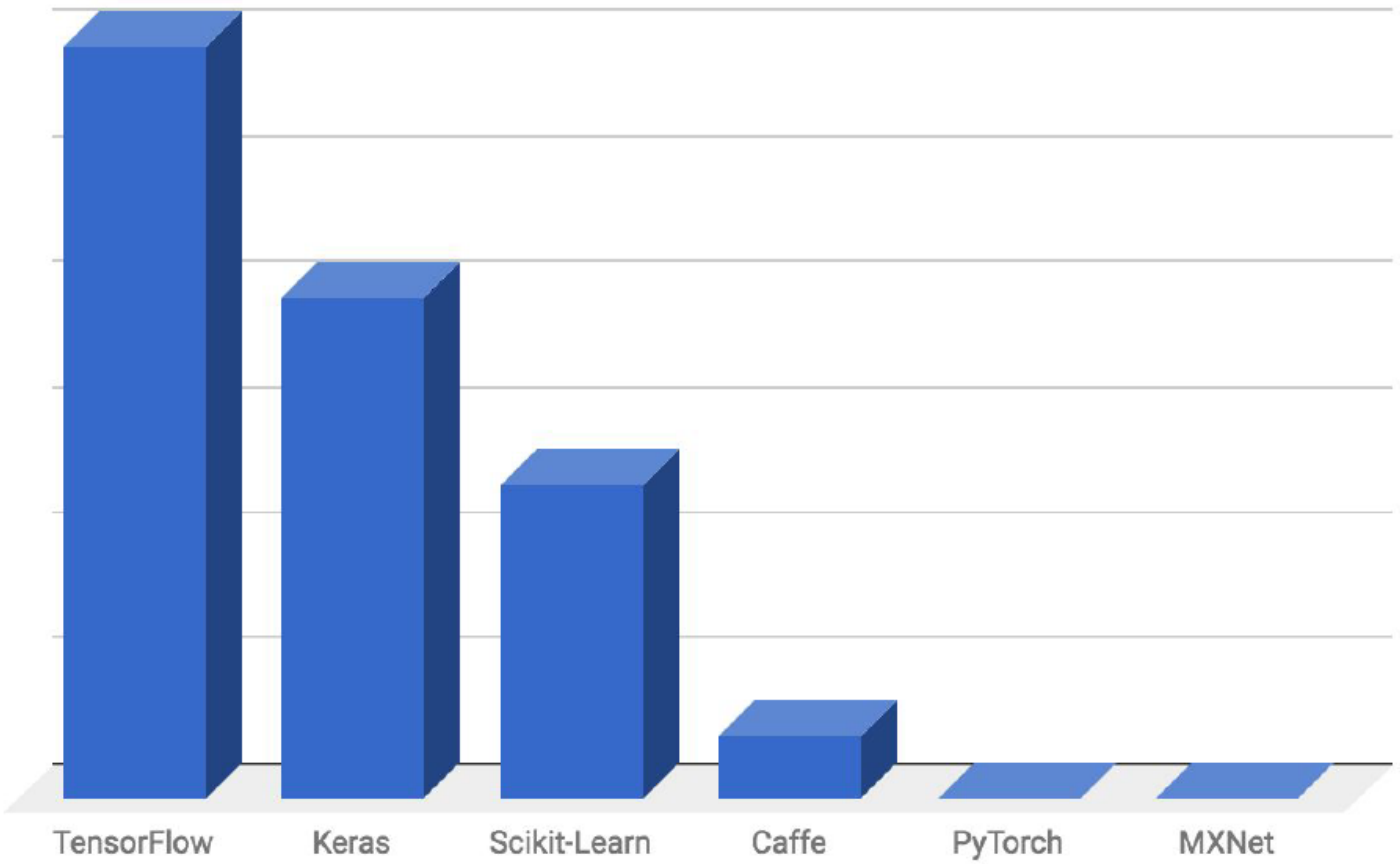


1. Introduction – Advantages

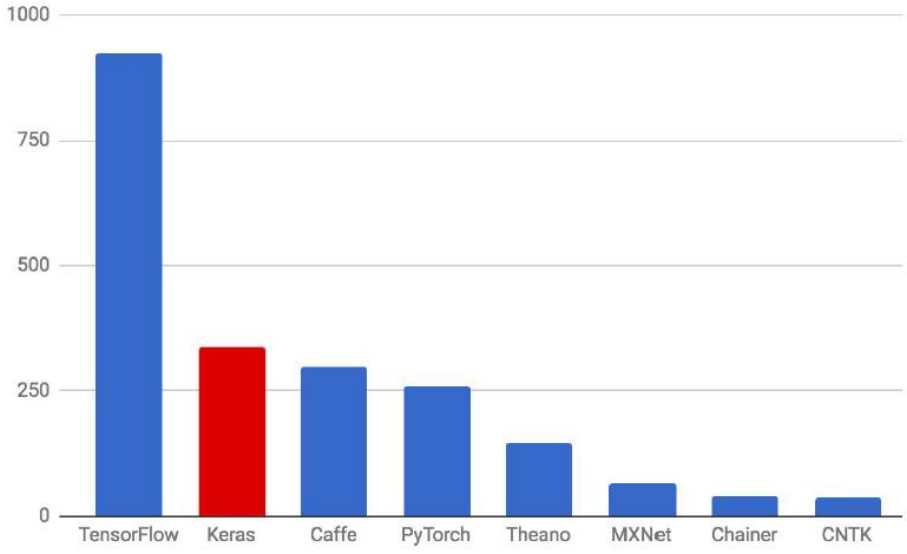
- *User friendly*: Keras has a simple, consistent interface optimized for common use cases. It provides clear and actionable feedback for user errors.
- *Modular and composable*: Keras models are made by connecting configurable building blocks together, with few restrictions.
- *Easy to extend*: Write custom building blocks to express new ideas for research. Create new layers, loss functions, and develop state-of-the-art models.

1. Introduction – Traction

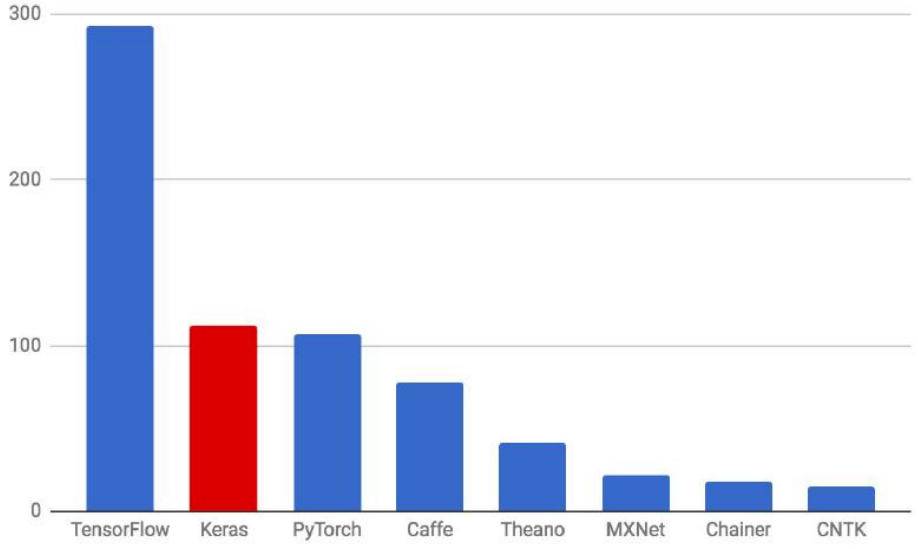
Hacker News jobs board mentions - out of 964 job postings



1. Introduction – Traction



arXiv mentions as of 2018/03/07 (past 3 months)



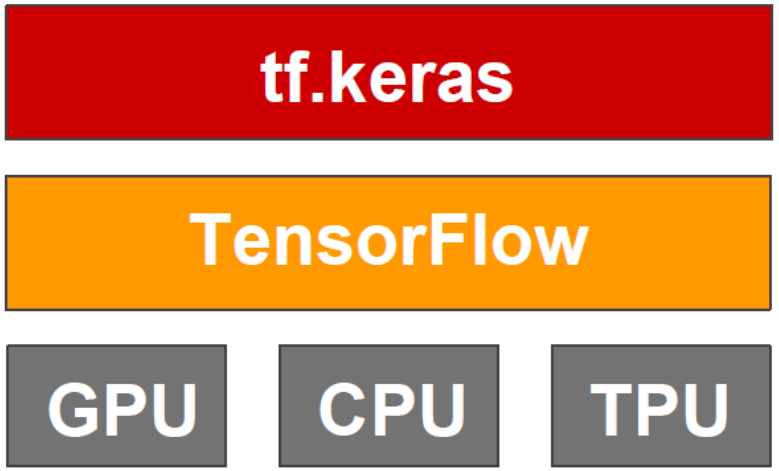
arXiv mentions as of 2018/03/07 (past 1 month)

2. Keras API Styles

1. The Sequential Model
   - Dead simple
   - Only for single-input, single-output, sequential layer stacks
   - Good for 70+% of use cases
2. The functional API
   - Like playing with Lego bricks
   - Multi-input, multi-output, arbitrary static graph topologies
   - Good for 95% of use cases
3. Model subclassing
   - Maximum flexibility
   - Larger potential error surface

3. TensorFlow Keras

- Keras is the official high-level API of TensorFlow
- tensorflow.keras (tf.keras) module
- Part of core TensorFlow since v1.4
- Full Keras API
- Better optimized for TF
- Better integration with TF-specific



3. TensorFlow Keras

- To import Keras from TensorFlow

```
import tensorflow as tf
from tensorflow.keras import layers

print(tf.VERSION)
print(tf.keras.__version__)
```

```
1.12.0
2.1.6-tf
```

- layers has:
  - Dense
  - Activations
  - Dropout
  - Conv1D, 2D, 3D
  - Polling
  - RNN, LSTM, GRU
  - …

4. The Sequential Model

- In Keras, you assemble layers to build models. The most common type of model is a stack of layers: the tf.keras.Sequential model.
- To build a simple, fully-connected network (i.e. multi-layer perceptron):

```
model = tf.keras.Sequential()
# Adds a densely-connected layer with 64 units to the model
model.add(layers.Dense(64, activation='relu'))
# Add another:
model.add(layers.Dense(64, activation='relu'))
# Add a softmax layer with 10 output units:
model.add(layers.Dense(10, activation='softmax'))
```

4.1 Configuring the layers

- Configure by passing the name of a built-in function or a callable object.

```
# Create a sigmoid layer:
layers.Dense(64, activation='sigmoid')
# Or:
layers.Dense(64, activation=tf.sigmoid)

# A linear layer with L1 regularization of factor 0.01 applied to the kernel mat
layers.Dense(64, kernel_regularizer=tf.keras.regularizers.l1(0.01))

# A linear layer with L2 regularization of factor 0.01 applied to the bias vecto
layers.Dense(64, bias_regularizer=tf.keras.regularizers.l2(0.01))

# A linear layer with a kernel initialized to a random orthogonal matrix:
layers.Dense(64, kernel_initializer='orthogonal')

# A linear layer with a bias vector initialized to 2.0s:
layers.Dense(64, bias_initializer=tf.keras.initializers.constant(2.0))
```

The default is no activation function, i.e., linear layer.

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum \|w\|$$

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum \|w\|^2$$

4.2 Set up training

- After the model is constructed, configure its learning process by calling the compile method:

```
model.compile(optimizer=tf.train.AdamOptimizer(0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

- Arguments
  - optimizer: The training procedure: AdamOptimizer, RMSPropOptimizer, or GradientDescentOptimizer.
  - loss: The function to minimize: mean square error (mse), categorical_crossentropy, and binary_crossentropy.
  - metrics: Used to monitor training.

4.3 Training

1. Input NumPy data

```
import numpy as np

data = np.random.random((1000, 32))
labels = np.random.random((1000, 10))

model.fit(data, labels, epochs=10, batch_size=32)
```

```
Epoch 1/10
1000/1000 [==============================] - 0s 254us/step - loss: 11.5936 - categorical_accuracy: 0
Epoch 2/10
1000/1000 [==============================] - 0s 64us/step - loss: 11.5301 - categorical_accuracy: 0.
```

Can also monitor progress on validation data

```
model.fit(data, labels, epochs=10, batch_size=32,
          validation_data=(val_data, val_labels))
```

4.3 Training

2. Use the Datasets API to scale to large datasets or multi-device training.

```
# Instantiates a toy dataset instance:
dataset = tf.data.Dataset.from_tensor_slices((data, labels))
dataset = dataset.batch(32)
dataset = dataset.repeat()

# Don't forget to specify `steps_per_epoch` when calling `fit`
model.fit(dataset, epochs=10, steps_per_epoch=30)
```

```
model.fit(dataset, epochs=10, steps_per_epoch=30,
          validation_data=val_dataset,
          validation_steps=3)
```

4.4 Evaluate

```
data = np.random.random((1000, 32))
labels = np.random.random((1000, 10))

model.evaluate(data, labels, batch_size=32)

model.evaluate(dataset, steps=30)
```

```
1000/1000 [==============================] - 0s 74us/step
30/30 [==============================] - 0s 3ms/step

[11.492062791188557, 0.178125]
```

4.5 Predict

```
result = model.predict(data, batch_size=32)
print(result.shape)
```

```
(1000, 10)
```

5. Example - MNIST

1. Define your training data: input tensors and target tensors.
2. Define a network of layers (or *model*) that maps your inputs to your targets.
3. Configure the learning process by choosing a loss function, an optimizer, and some metrics to monitor.
4. Iterate on your training data by calling the fit() method of your model.

5. Example – Prepare the data

```
from keras.datasets import mnist
(train_images, train_labels), (test_images,
      test_labels) = mnist.load_data()
#(60000, 28, 28), (6000), #(10000, 28, 28), (1000)
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255

from keras.utils import to_categorical    #one hot
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

5. Example – Define and configure the network

```
from keras import models
from keras import layers

network = models.Sequential()
network.add(layers.Dense(512, activation='relu',
        input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))

network.compile(optimizer='rmsprop',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

5. Example – Training and evaluation

```
network.fit(train_images, train_labels, epochs=5,
        batch_size=128)
```

```
Epoch 1/5
60000/60000 [==============================] - 2s - loss: 0.2577 - acc: 0.9245
Epoch 2/5
60000/60000 [==============================] - 1s - loss: 0.1042 - acc: 0.9690
Epoch 3/5
60000/60000 [==============================] - 1s - loss: 0.0687 - acc: 0.9793
Epoch 4/5
60000/60000 [==============================] - 1s - loss: 0.0508 - acc: 0.9848
Epoch 5/5
60000/60000 [==============================] - 1s - loss: 0.0382 - acc: 0.9890
```

```
test_loss, test_acc = network.evaluate(test_images,
        test_labels)
```

```
9536/10000 [============================>..] - ETA: 0s
```

```
test_acc: 0.9777
```

6. Fashion-MNIST Example

- Fashion-MNIST with tf.keras by Yufeng Guo

- YouTube Video: *Getting Started with Keras (AI Adventures)* from Google Cloud Platform: https://youtu.be/J6Ok8p463C4

- Code: https://www.kaggle.com/yufengg/fashion-mnist-with-keras

7. Tutorials

- https://keras.io/
- https://www.tensorflow.org/guide/keras
- Keras Tutorial: Deep Learning in Python from DataCamp, https://www.datacamp.com/community/tutorials/deep-learning-python
- Keras Tutorial: The Ultimate Beginner's Guide to Deep Learning in Python, from EliteDataScience, https://elitedatascience.com/keras-tutorial-deep-learning-in-python

Summary

1. Introduction
2. Keras API Styles
3. TensorFlow Keras
4. The Sequential Model
5. Example – MNIST
6. Fashion-MNIST Example
7. Tutorials