Introduction to Keras and TensorFlow

Qing Wan and **Yoonsuck Choe**

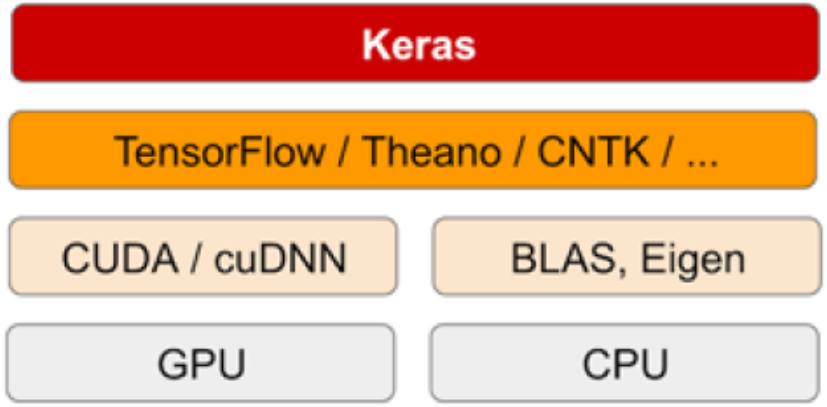
Texas A&M University

Outline

- Background → NVIDIA CUDA GPU
- Installation
- Basic skills to write a machine learning model
- A specific case: XOR gate

Keras

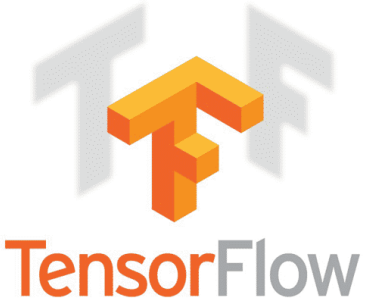
- A python package (Python 2.7-3.6)
- Sits on top of TensorFlow or Theano (Stopped)
- High-level neural network API
- Runs seamlessly on CPU and GPU
- Open source with user manual (https://keras.io/)
- Less coding lines required to build/run a model



TensorFlow

- Inherit from Theano (data flow graph)
- A python(3.5-3.7) package/C++ library
- Running on CPU or NVIDIA CUDA GPU
- End-2-End platform for machine/deep learning
- Multi platform (desktop, web by TF.js, mobile by TF Lite)
- Open source with user manual (https://www.tensorflow.org/)

- More coding lines required to build/run a model



Install Tensorflow GPU
with CUDA 10.0

NVIDIA CUDA Toolkit

- C/C++ library
- A parallel computing platform for NVIDIA GPU
- Most deep learning researchers rely on
- GPU-accelerated computing/applications
- Not open source (https://developer.nvidia.com/cuda-zone)
- CPU vs GPU: TensorFlow training CNN model on CIFAR10 images

Device	Speed of training, examples/sec
2 x AMD Opteron 6168	440
i7-7500U	415
GeForce 940MX	1190
GeForce 1070	6500

Anaconda3 Installation

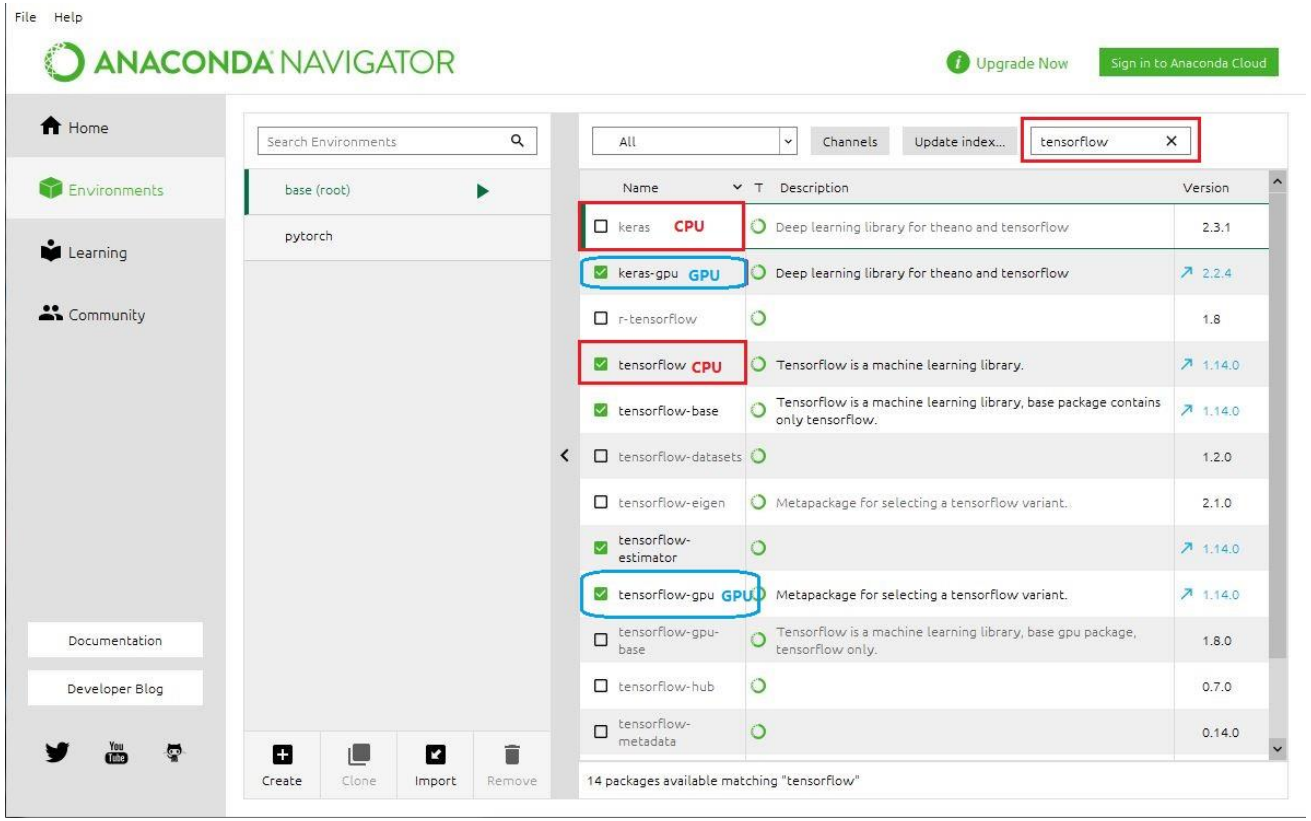
- Anaconda3
  - Download (https://www.anaconda.com/distribution/)
  - Installation (https://docs.anaconda.com/anaconda/install/)
  - Restart required

TensorFlow/Keras Installation

- Start the anaconda navigator
  - Windows: Start->All program->Anaconda3->Anaconda Navigator
  - Linux: type “anaconda-navigator” under the linux terminal
- Install TensorFlow and Keras
  - Environments->choose All
  - type “tensorflow”
  - CPU based:

    tensorflow (choose 1.14)

    keras (2.2.4) apply
  - GPU based:
    - CUDA Compute Capability >= 3.0, better >= 3.7 (check more)
    - tensorflow-gpu (choose 1.14) and keras-gpu (2.2.4), then apply



Installation Confirmed

- TensorFlow test code:

```
import tensorflow as tf
sess = tf.compat.v1.Session()
a = tf.compat.v1.constant(1)
b = tf.compat.v1.constant(2)
print(sess.run(a+b))
```

- Expect to have answer 3

```
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)] :: Ana
conda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
>>> sess = tf.compat.v1.Session()
2020-03-10 17:14:06.299800: I tensorflow/stream_executor/platform/default/dso_lo
ader.cc:42] Successfully opened dynamic library nvcuda.dll
2020-03-10 17:14:06.517813: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1
640] Found device 0 with properties:
name: Quadro K1000M major: 3 minor: 0 memoryClockRate(GHz): 0.8505
pciBusID: 0000:01:00.0
2020-03-10 17:14:06.528813: I tensorflow/stream_executor/platform/default/dlopen
_checker_stub.cc:25] GPU libraries are statically linked, skip dlopen check.
2020-03-10 17:14:06.546814: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1
763] Adding visible gpu devices: 0
2020-03-10 17:14:06.553815: I tensorflow/core/platform/cpu_feature_guard.cc:142]
 Your CPU supports instructions that this TensorFlow binary was not compiled to
use: AVX
2020-03-10 17:14:06.570816: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1
640] Found device 0 with properties:
name: Quadro K1000M major: 3 minor: 0 memoryClockRate(GHz): 0.8505
pciBusID: 0000:01:00.0
2020-03-10 17:14:06.583816: I tensorflow/stream_executor/platform/default/dlopen
_checker_stub.cc:25] GPU libraries are statically linked, skip dlopen check.
2020-03-10 17:14:06.601817: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1
763] Adding visible gpu devices: 0
2020-03-10 17:14:07.301857: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1
181] Device interconnect StreamExecutor with strength 1 edge matrix:
2020-03-10 17:14:07.309858: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1
187]      0
2020-03-10 17:14:07.314858: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1
200] 0:   N
2020-03-10 17:14:07.331859: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1
326] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 wit
h 1691 MB memory) -> physical GPU (device: 0, name: Quadro K1000M, pci bus id: 0
000:01:00.0, compute capability: 3.0)
>>> a = tf.compat.v1.constant(1)
>>> b = tf.compat.v1.constant(2)
>>> print(sess.run(a+b))
3
>>>
```

Installation Confirmed

- Keras requires backend setting for Windows users:
  - https://keras.io/backend/
  - Setting in keras.json:

    “backend”: “tensorflow”
- Keras test code:

  import keras

```
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)] :
conda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import keras
Using TensorFlow backend.
>>>
```

- Expect to see

  Using TensorFlow backend 

Keras Models

- Two main types of models available
  - The Sequential model (easy to learn, high-level API)
    - A linear stack of layers
    - Need to specify what input shape it should expect (input dimension)
    - https://keras.io/getting-started/sequential-model-guide/
  - The Model class used with the functional API (similar to tensorflow2.0)
    - https://keras.io/models/about-keras-models/
    - https://keras.io/getting-started/functional-api-guide/

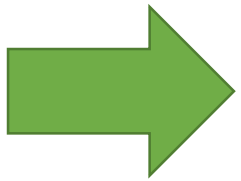
Keras Sequential Model

- Define a sequential model

```
model = Sequential()
mode.add(Dense(32, input_dim=784))
model.add(Activation('relu'))
model.add(Dense(10))
model.add(Activation('softmax'))
```

- Compilation

```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuray'])
```



- Training

```
model = model.fit(data, one_hot_labels,
epoch=10, batch_size=32)
```

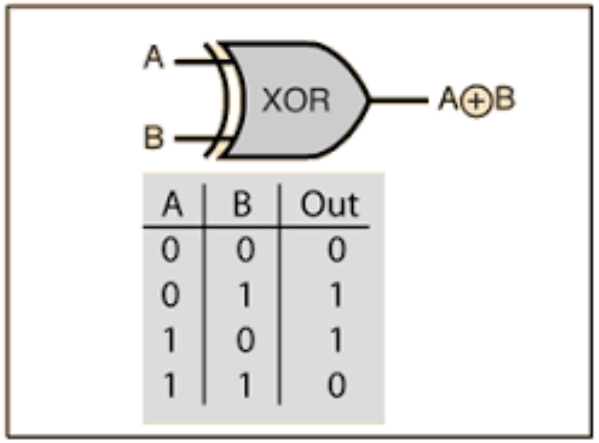
- Predition

```
Y = model.predict(X)
```

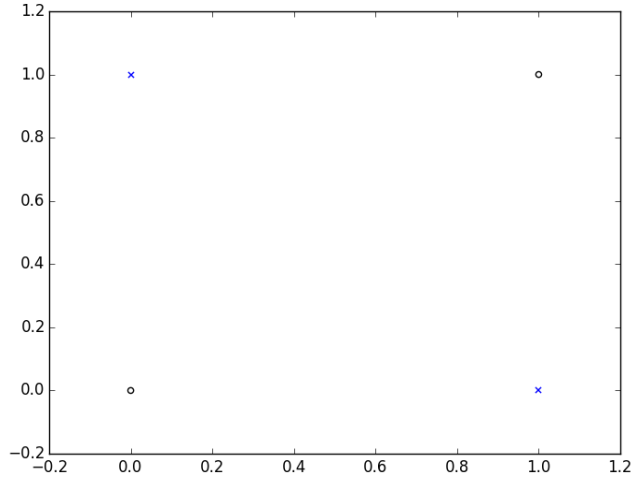
Case Study: XOR gate

Keras

```
import numpy as np
# import necessary packages or APIs from keras
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD
from keras.initializers import RandomUniform
# Prepare data and labels
X = np.asarray([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
Y = np.asarray([[0], [1], [1], [0]], dtype=np.float32)
# Build a model
model = Sequential()
model.add(Dense(units=2, activation='tanh', use_bias=True,
                kernel_initializer=RandomUniform(minval=-1, maxval=1, seed=None),
                input_dim=2))
model.add(Dense(units=1, activation='sigmoid', use_bias=True,
                kernel_initializer=RandomUniform(minval=-1, maxval=1, seed=None)))
# Build optimizer and do model compilation
op = SGD(lr=0.01, momentum=0.0)
model.compile(optimizer=op,
              loss='mse',
              metrics=['accuracy'])
# Start to train
model.fit(x=X, y=Y, epochs=20000, batch_size=4, shuffle=True)
# Prediction
Y = model.predict(X)
print("Y = ", Y)
```



A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0



Case Study: XOR gate

- Training log

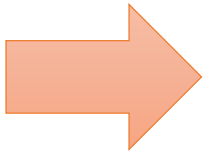
```
4/4 [==============================] - 2s 454ms/step - loss: 0.2538 - acc: 0.7500
Epoch 2/20000
...
4/4 [==============================] - 0s 2ms/step - loss: 0.2531 - acc: 0.5000
Epoch 100/20000
...
4/4 [==============================] - 0s 1ms/step - loss: 0.2511 - acc: 0.5000
Epoch 1000/20000
...
4/4 [==============================] - 0s 2ms/step - loss: 0.2291 - acc: 0.7500
Epoch 10000/20000
...
4/4 [==============================] - 0s 1ms/step - loss: 0.0254 - acc: 1.0000
Epoch 20000/20000

4/4 [==============================] - 0s 1ms/step - loss: 0.0254 - acc: 1.0000
```

- Prediction

```
Y =  [[0.1462788 ]
 [0.8177988 ]
 [0.8254448 ]
 [0.12802514]]
```

TensorFlow Models

- What is Tensor?
  - An object (constant, scalar, vector, matrix, …)
  - Allow to define ops (+, -, *, /, sum, max, concatenate, …) on
- TensorFlow Model
  - A function with learnable parameters
  - Maps input to an output by ops
  - Parameters are all defined by yourself
    - Model itself
    - Loss
    - Optimizer
    - Whether a parameter is learnable
    - Data operation



More flexible than Keras
More complex than Keras

Build a TensorFlow Model

- Two ways to build a machine learning model
  - Using the layers API where you build a model using layers
    e.g. tf.keras.layers.Dense, tf.layers.Conv2D, …
  - Using the Core API with lower-level ops
    e.g. tf.math.add, tf.math.abs, tf.concat, …

Case Study: XOR gate


```
import numpy as np
import tensorflow as tf

# Prepare data and labels
data = np.asarray([[0, 0, 0], [0, 1, 1], [1, 0, 1], [1, 1, 0]], dtype=np.float32)
# Build a model
x = tf.compat.v1.placeholder(tf.float32, shape=[None, 2], name='x_in') # input
label = tf.compat.v1.placeholder(tf.float32, shape=[None, 1], name='label') # label
hidden = tf.keras.layers.Dense(units=2, activation=tf.nn.tanh, use_bias=True,
                               kernel_initializer=tf.keras.initializers.RandomUniform(minval=-1, maxval=1))(x)
pred = tf.compat.v1.keras.layers.Dense(units=1, activation=tf.nn.sigmoid, use_bias=True,
                                       kernel_initializer=tf.keras.initializers.RandomUniform(minval=-1, maxval=1))(hidden)
cost = tf.compat.v1.norm(tf.math.subtract(label, pred), ord=2, name='cost')
op = tf.compat.v1.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

training_epochs = 20000
# Set GRAM allocation
config = tf.compat.v1.ConfigProto(device_count={'GPU': 1})
config.gpu_options.per_process_gpu_memory_fraction = 0.5
# Start a Session to train and test
with tf.compat.v1.Session(config=config) as sess:
    sess.run(tf.compat.v1.global_variables_initializer())
    # Train
    for epoch in range(training_epochs):
        loss = 0
        for d in data:
            training_X, training_Y = np.asarray(d[0:2], dtype=np.float32), np.asarray(d[2], dtype=np.float32)
            training_X, training_Y = np.expand_dims(training_X, axis=0), np.expand_dims([training_Y], axis=0)
            sess.run(op, feed_dict={x: training_X, label: training_Y})
            loss += sess.run(cost, feed_dict={x: training_X, label: training_Y})
        if epoch % 100 == 0:
            print("epoch: %d, loss = %f" % (epoch, loss))
    # Test
    for d in data:
        Y = sess.run(pred, feed_dict={x: np.expand_dims(np.asarray(d[0:2], dtype=np.float32), axis=0)})
        print("d = ", d, "output = ", Y)
```

Case Study: XOR gate

- Training log

  epoch: 0, loss = 2.025534
  epoch: 100, loss = 1.963005
  …
  epoch: 1000, loss = 0.051374
  …
  epoch: 10000, loss = 0.003195
  …
  epoch: 19800, loss = 0.001572
  epoch: 19900, loss = 0.001564

- Prediction

  d = [0. 0. 0.] output = [[0.00028096]]
  d = [0. 1. 1.] output = [[0.9994849]]
  d = [1. 0. 1.] output = [[0.99948573]]
  d = [1. 1. 0.] output = [[0.0002458]]

Case Study: XOR gate

- What happen if we remove kernel_initialization in both Keras model and TensorFlow model?
- Try if you are interested

- Do we really get the right answer?
- Are these results stable?
- What’s a potential cause to this?

Summary

As two popular deep learning packages

- Keras
  - User friendly with high-level APIs
  - Quick to get started
  - Coding less lines for machine learning model construction/training/testing
  - Sometimes training convergence is not stable.

- TensorFlow
  - Flexible for developing new machine learning model
  - Multi platform
  - Community support
  - Not friendly for new learner due to many low level APIs

Thanks