TensorFlow

Prof. Gheith Abandah

Reference: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by Aurélien Géron (O'Reilly). Copyright 2017 Aurélien Géron, 978-1-491-96229-9.

Introduction

- YouTube Video: *Deep Learning with TensorFlow - Introduction to TensorFlow* from Cognitive Class

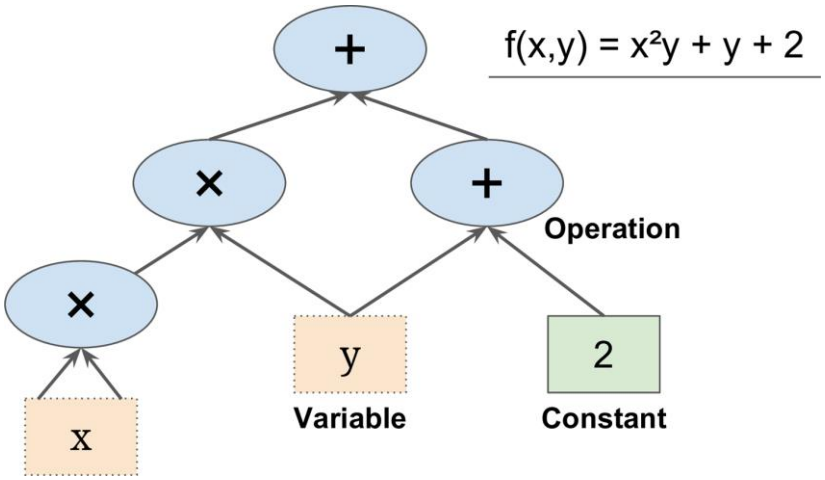
https://youtu.be/MotG3XI2qSs

Outline

1. Introduction
2. Installation
3. Example
4. Lifecycle of a node value
5. Linear regression with TensorFlow
6. Implementing gradient descent
7. Saving and restoring models
8. Exercises

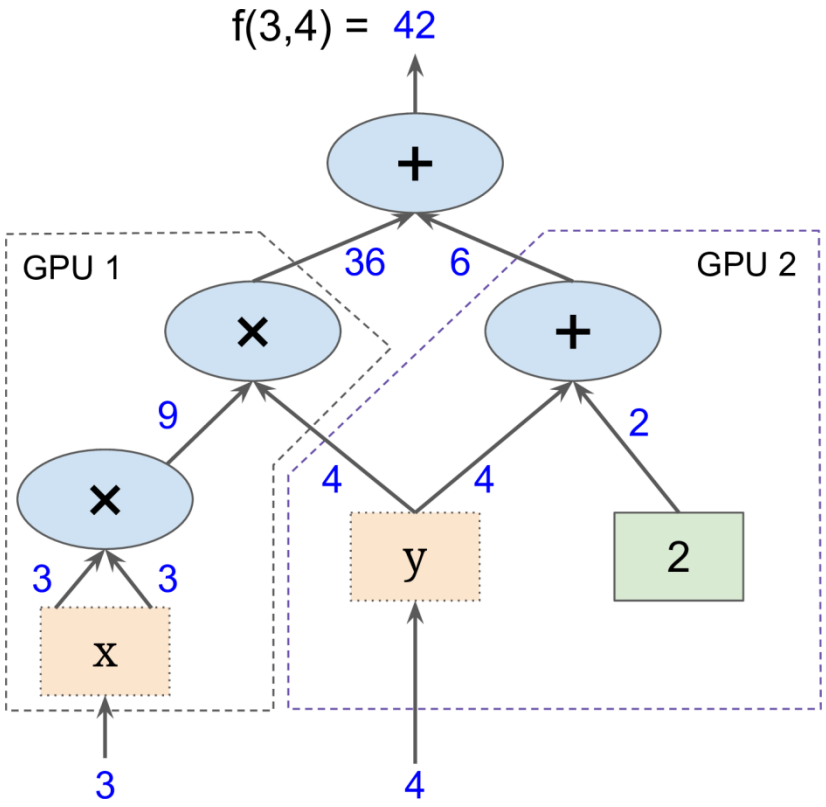
1. Introduction

- *TensorFlow* is a powerful open source, well supported, software library for numerical computation, particularly for large-scale Machine Learning.
- Its basic principle is simple:
  - you first define in Python a graph of computations to perform,
  - and then TensorFlow takes that graph and runs it efficiently using optimized C++ code.



1. Introduction

- It is possible to break up the graph into several chunks and run them in parallel across multiple CPUs or GPUs.
- TensorFlow also supports distributed computing, so you can train colossal neural networks on humongous training sets in a reasonable amount of time by splitting the computations across hundreds of servers.



1. Introduction

- Runs on Windows, Linux, and macOS, iOS, and Android.
- It provides a very simple Python APIs:
  - TF.Learn (tensorflow.contrib.learn), compatible with Scikit-Learn
  - TF-slim (tensorflow.contrib.slim)
  - Keras
  - Pretty Tensor
- Its main Python API offers much more flexibility (at the cost of higher complexity) to create all sorts of computations, including various neural network architectures.
- Includes a visualization tool called *TensorBoard*

2. Installation

- On a cmd window, execute:

```
pip3 install --upgrade tensorflow
```

- To verify, enter and run the following in Python:

```
import tensorflow as tf
hello = tf.constant('Hello, TensorFlow!')
sess = tf.Session()
print(sess.run(hello))
```

- If the system outputs the following, then you are ready to begin writing TensorFlow programs:

```
Hello, TensorFlow!
```

3. Example

- The following code creates the graph represented in Slide 4.

```
import tensorflow as tf

x = tf.Variable(3, name="x")
y = tf.Variable(4, name="y")
f = x*x*y + y + 2
```

- Create session, initialize vars, evaluate, and close.

```
>>> sess = tf.Session()
>>> sess.run(x.initializer)
>>> sess.run(y.initializer)
>>> result = sess.run(f)
>>> print(result)
42
>>> sess.close()
```

3. Example

- A better way in a Python program.

```
with tf.Session() as sess:
    x.initializer.run()
    y.initializer.run()
    result = f.eval()
```

- Alternatively, you can create and run an initialization node for all variables.

```
init = tf.global_variables_initializer()  # prepare an init node

with tf.Session() as sess:
    init.run()  # actually initialize all the variables
    result = f.eval()
```

4. Lifecycle of a Node Value

- When you evaluate a node, TensorFlow automatically determines the set of nodes that it depends on and it evaluates these nodes first.

```
w = tf.constant(3)
x = w + 2
y = x + 5
z = x * 3

with tf.Session() as sess:
    print(y.eval())  # 10
    print(z.eval())  # 15
```

Note: To evaluate y and z efficiently, without evaluating w and x twice, code, use:

```
with tf.Session() as sess:
    y_val, z_val = sess.run([y, z])
    print(y_val)  # 10
    print(z_val)  # 15
```

5. Linear Regression with TensorFlow

$$\mathbf{y} = X\boldsymbol{\beta} + \boldsymbol{\epsilon},$$

- The inputs and outputs are multidimensional arrays (NumPy ndarrays), called *tensors*.

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

```
import numpy as np
from sklearn.datasets import fetch_california_housing

housing = fetch_california_housing()
m, n = housing.data.shape
housing_data_plus_bias = np.c_[np.ones((m, 1)), housing.data]

X = tf.constant(housing_data_plus_bias, dtype=tf.float32, name="X")
y = tf.constant(housing.target.reshape(-1, 1), dtype=tf.float32, name="y")
XT = tf.transpose(X)
theta = tf.matmul(tf.matmul(tf.matrix_inverse(tf.matmul(XT, X)), XT), y)

with tf.Session() as sess:
    theta_value = theta.eval()
```

6. Implementing Gradient Descent

- Manually or using an optimizer (replace last two lines with the two lines in the box):

$$\theta^{(\text{next step})} = \theta - \eta \nabla_\theta \text{MSE}(\theta)$$

```
n_epochs = 1000
learning_rate = 0.01

X = tf.constant(scaled_housing_data_plus_bias, dtype=tf.float32, name="X")
y = tf.constant(housing.target.reshape(-1, 1), dtype=tf.float32, name="y")
theta = tf.Variable(tf.random_uniform([n + 1, 1], -1.0, 1.0), name="theta")
y_pred = tf.matmul(X, theta, name="predictions")
error = y_pred - y
mse = tf.reduce_mean(tf.square(error), name="mse")
gradients = 2/m * tf.matmul(tf.transpose(X), error)
training_op = tf.assign(theta, theta - learning_rate * gradients)

optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(mse)
```

6. Implementing Gradient Descent

```
init = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(init)

    for epoch in range(n_epochs):
        if epoch % 100 == 0:
            print("Epoch", epoch, "MSE =", mse.eval())
        sess.run(training_op)

    best_theta = theta.eval()
```

7. Saving and Restoring Models

```
[...]
theta = tf.Variable(tf.random_uniform([n + 1, 1], -1.0, 1.0), name="theta")
[...]
init = tf.global_variables_initializer()
saver = tf.train.Saver()

with tf.Session() as sess:
    sess.run(init)

    for epoch in range(n_epochs):
        if epoch % 100 == 0:  # checkpoint every 100 epochs
            save_path = saver.save(sess, "/tmp/my_model.ckpt")

        sess.run(training_op)

    best_theta = theta.eval()
    save_path = saver.save(sess, "/tmp/my_model_final.ckpt")
```

7. Saving and Restoring Models

```
with tf.Session() as sess:
    saver.restore(sess, "/tmp/my_model_final.ckpt")
    [...]
```

Summary

1. Introduction
2. Installation
3. Example
4. Lifecycle of a node value
5. Linear regression with TensorFlow
6. Implementing gradient descent
7. Saving and restoring models
8. Exercises

Exercises

- Check TensorFlow exercises on: https://github.com/Kyubyong/tensorflow-exercises