Intro

- In this video we will overview basic principles of TensorFlow

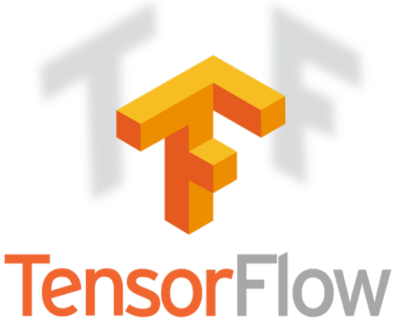
TensorFlow DL framework

- We will use it in Jupyter Notebook with Python 3 kernel

```
import numpy as np
import tensorflow as tf
```

- We will overview Python API for TensorFlow 1.2+

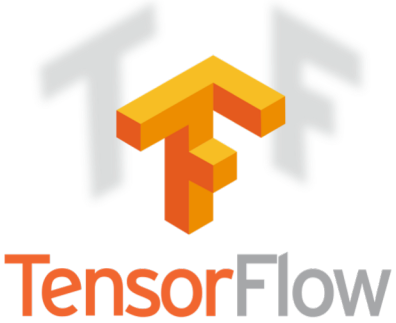
- APIs in other languages exist: Java, C++, Go
  - Python API is at present the most complete and the easiest to use
  - https://www.tensorflow.org/api_docs/



What is TensorFlow?

1. A tool to describe computational graphs
   - The foundation of computation in TensorFlow is the **Graph** object. This holds a network of nodes, each representing one **operation**, connected to each other as inputs and outputs.

2. A runtime for execution of these graphs
   - On CPU, GPU, TPU, …
   - On one node or in distributed mode



Why this name

- **Input** to any operation will be a collection of **tensors** (multi-dimensional arrays)

- **Output** will be a collection of tensors as well.

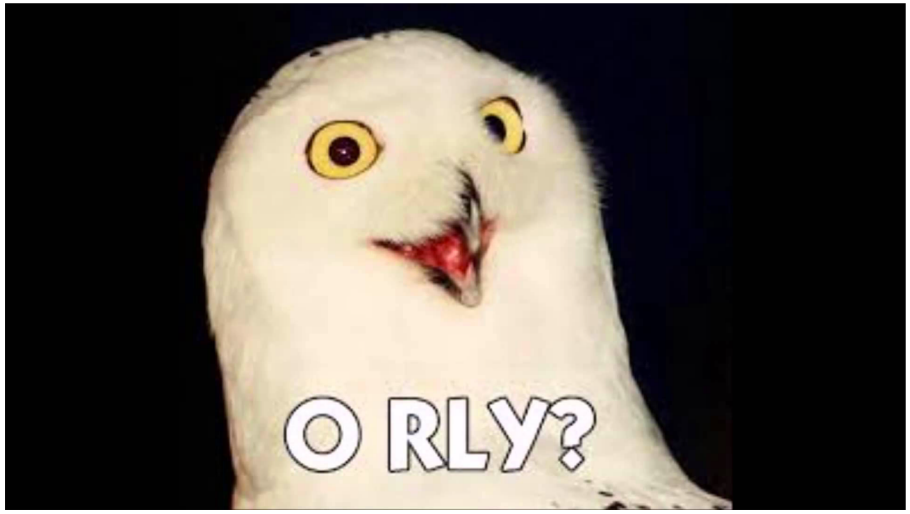
- We will have a graph of operations, each of which transforms tensors into another tensors, so it’s a kind of a flow of tensors

Why this name

- **Input** to any operation will be a collection of **tensors** (multi-dimensional arrays)

- **Output** will be a collection of tensors as well.

- We will have a graph of operations, each of which transforms tensors into another tensors, so it's a kind of a flow of tensors



How the input looks like

- **Placeholder**
  - This is placeholder for a tensor, which will be fed during graph execution (e.g. input features)
  - x = tf.placeholder(tf.float32, (None, 10))
- **Variable**
  - This is a tensor with some value that is updated during execution (e.g. weights matrix in MLP)
  - w = tf.get_variable("w", shape=(10, 20), dtype=tf.float32)
  - w = tf.Variable(tf.random_uniform((10, 20)), name="w")
- **Constant**
  - This is a tensor with constant value, that cannot be changed
  - c = tf.constant(np.ones((4, 4)))

Operation example

- Matrix product:

```
x = tf.placeholder(tf.float32, (None, 10))
w = tf.Variable(tf.random_uniform((10, 20)), name="w")
z = x @ w
# z = tf.matmul(x, w)
print(z)
```

- Output:

```
Tensor("matmul:0", shape=(?, 20), dtype=float32)
```

- We don’t do any computations here, we just **define** the graph!

Computational graph

- TensorFlow creates a **default graph** after importing
  - All the operations will go there by default
  - You can get it with tf.get_default_graph(), which returns an instance of tf.Graph.

- You can **create your own** graph variable and define operations there:

```
g = tf.Graph()
with g.as_default():
    pass
```

- You can **clear** the default graph like this:

```
tf.reset_default_graph()
```

Jupyter Notebook cells

- If you run this cell **3 times**:

```
x = tf.placeholder(tf.float32, (None, 10))
```

- This is what you get in your **default graph**:
  - Using tf.get_default_graph().get_operations()

```
[<tf.Operation 'Placeholder' type=Placeholder>,
 <tf.Operation 'Placeholder_1' type=Placeholder>,
 <tf.Operation 'Placeholder_2' type=Placeholder>]
```

- **Your graph is cluttered**!
  - Clear your graph with tf.reset_default_graph()

Operations and tensors

- Every **node** in our graph is an **operation**:

```
x = tf.placeholder(tf.float32, (None, 10), name="x")
```

- Listing nodes with tf.get_default_graph().get_operations():

```
[<tf.Operation 'x' type=Placeholder>]
```

- How to get **output tensors** of operation:
  - tf.get_default_graph().get_operations()[0].outputs
  - Output: [<tf.Tensor **'x:0'** shape=(?, 10) dtype=float32>]

Running a graph

- A **tf.Session** object encapsulates the environment in which tf.Operation objects are executed, and tf.Tensor objects are evaluated.
- Create a session:

```
s = tf.InteractiveSession()
```

- Defining a graph:

```
a = tf.constant(5.0)
b = tf.constant(6.0)
c = a * b
```

- Running a graph:

```
print(c)  # here just looking at the type
print(s.run(c))  # that's how you run the graph
```

- Output:

```
Tensor("mul:0", shape=(), dtype=float32)
30.0
```

Running a graph

- Operations are written in C++ and executed on CPU or GPU.

- tf.Session owns necessary resources to execute your graph, such as tf.Variable, that occupy RAM.

- It is important to release these resources when they are no longer required with tf.Session.close()

Initialization of variables

- A variable has an initial value:
  - Tensor: tf.Variable(tf.random_uniform((10, 20)), name="w")
  - Initializer: tf.get_variable("w", shape=(10, 20), dtype=tf.float32)

- You need to run some code to **compute that initial value** in graph execution environment

- This is done with a call in your session s:

```
s.run(tf.global_variables_initializer())
```

- Without it you will get “Attempting to use uninitialized value” errors

Example

- Definition:

```
tf.reset_default_graph()
a = tf.constant(np.ones((2, 2), dtype=np.float32))
b = tf.Variable(tf.ones((2, 2)))
c = a @ b
```

- Running attempt:

```
s = tf.InteractiveSession()
s.run(c)
```

Output: **“Attempting to use uninitialized value”** error

- Running properly:

```
s.run(tf.global_variables_initializer())
s.run(c)
```

Output: array([[2.,2.],[2.,2.]], dtype=float32)

Feeding placeholder values

- Definition:

```
tf.reset_default_graph()
a = tf.placeholder(np.float32, (2, 2))
b = tf.Variable(tf.ones((2, 2)))
c = a @ b
```

- Running attempt:

```
s = tf.InteractiveSession()
s.run(tf.global_variables_initializer())
s.run(c)
```

Output: "**You must feed a value for placeholder tensor**" error

- Running properly:

```
s.run(tf.global_variables_initializer())
s.run(c, feed_dict={a: np.ones((2, 2))})
```

Output: array([[2.,2.],[2.,2.]], dtype=float32)

Summary

- TensorFlow: defining and running computational graphs
- Nodes of a graph are operations, that convert a collection of tensors into another collection of tensors
- In Python API you **define** the graph, you **don't execute** it along the way
  - In 1.5+ the latter mode is supported: eager execution
- You create a **session** to execute your graph (fast C++ code on CPU or GPU)
- Session **owns** all the resources (tensors eat RAM)