TensorFlow & Keras

An Introduction

(Some of the contents on these slides, along with the template, have been adopted from William Guss (ex TA) and CS 224 and CS20 at Stanford)

Deep Learning Frameworks

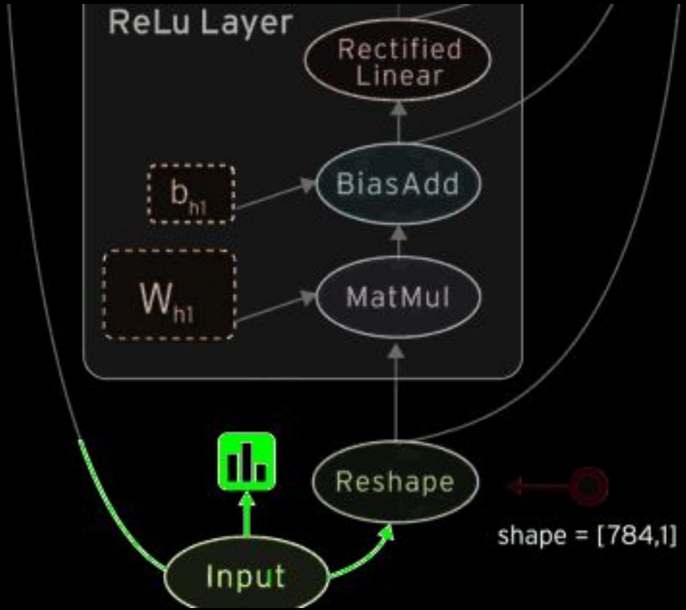
- Scale ML code
- Compute Gradients!
- Standardize ML applications for sharing
- Interface with GPUs for parallel processing

TensorFlow

What is TensorFlow?

TensorFlow is a graph computation framework for deep learning.

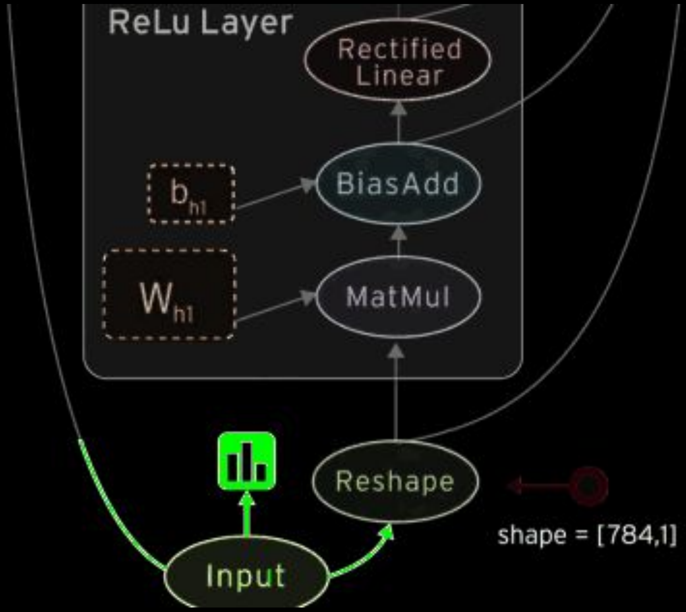
Originally developed by Google Brain Team to conduct ML research



What is TensorFlow?

TensorFlow allows for the specification and optimization of complex feed-forward models.

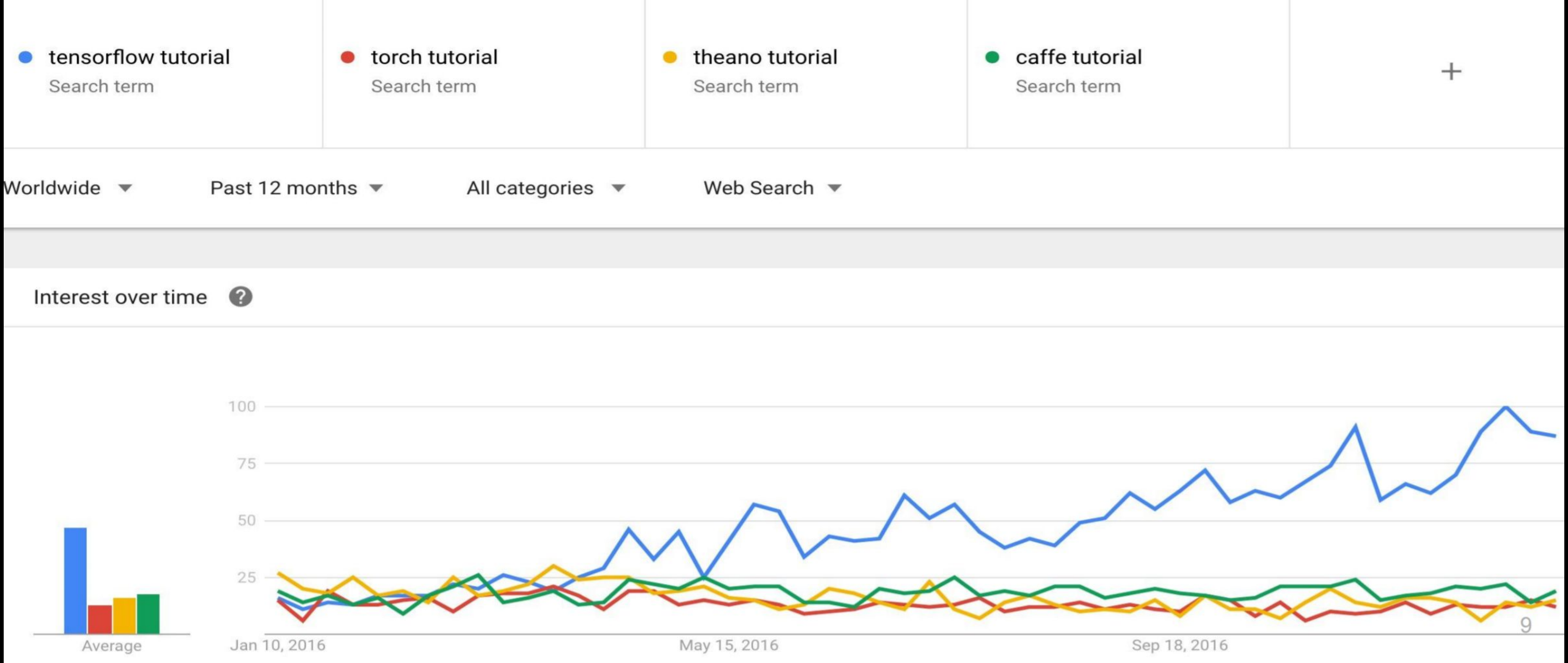
In particular, TensorFlow automatically differentiates a specified model.



Why TensorFlow?

- Python API
- Portability: deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API
- Visualization: TensorBoard
- Auto-differentiation
- Large community (> 10,000 commits and > 3000 TF-related repos in 1 year)
- Awesome projects already using TensorFlow

Why TensorFlow?



A motivating example

The NumPy approach

```
net1 = W1 @ x + b1
h = s(net1)
net2 = W2 @ h + b2
output = s(net2)

# Manually compute derivative for W2
```

$$\frac{dL}{d\text{output}} = (\texttt{output} - \texttt{label})$$

$$\frac{dL}{dW_2} = \frac{dL}{d\text{output}} \; @ \; \frac{ds}{d\text{net}}(\texttt{net}) \; @ \; \texttt{h}$$

$$\texttt{W2 -= learning\_rate} * \frac{dL}{dW_2}$$

```
# Repeat for all other variables :\
```

The TensorFlow approach

```
net1 = W1 @ x + b1
h = tf.nn.sigmoid(net1)
net2 = W2 @ h + b2
output = tf.nn.sigmoid(net2)

# Let tensorflow do the heavy lifting
optimizer = tf.train.AdamOptimizer(learning_rate)
train = optimizer.minimize(L)

# Done :)
```

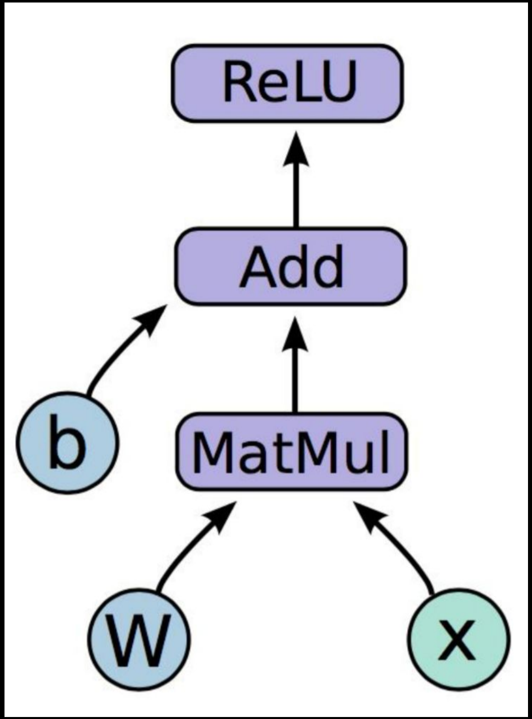
Programming Model

Big idea: express a numeric computation as a graph

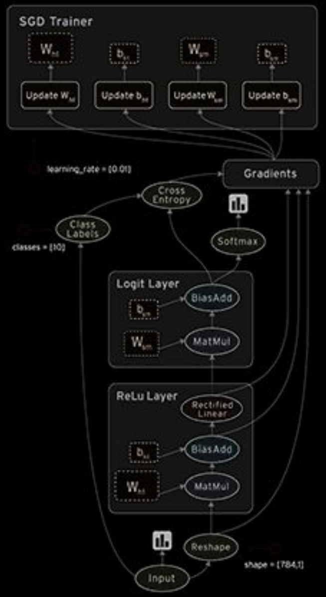
- Graph nodes are operations which have any number of inputs and outputs
- Graph edges are tensors which flow between nodes

Programming Model

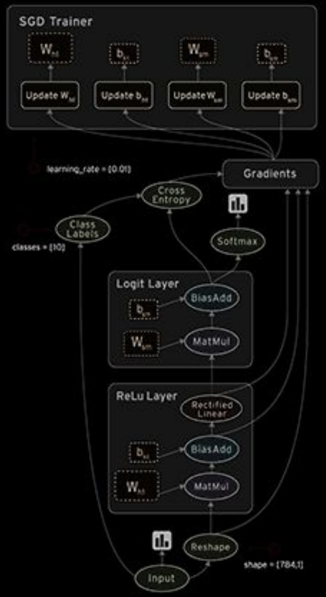
$h = ReLU\,(Wx + b)$



TensorFlow Basics



Construction



Execution

TensorFlow Basics: Construction

Placeholders (`tf.Placeholder`)

- Allow data to be fed into the computation graph at execution time (e.g. features, labels)

```
x = tf.Placeholder
(float)
```

```
y = tf.constant(5.0)
```

Variables (`tf.Variable`)

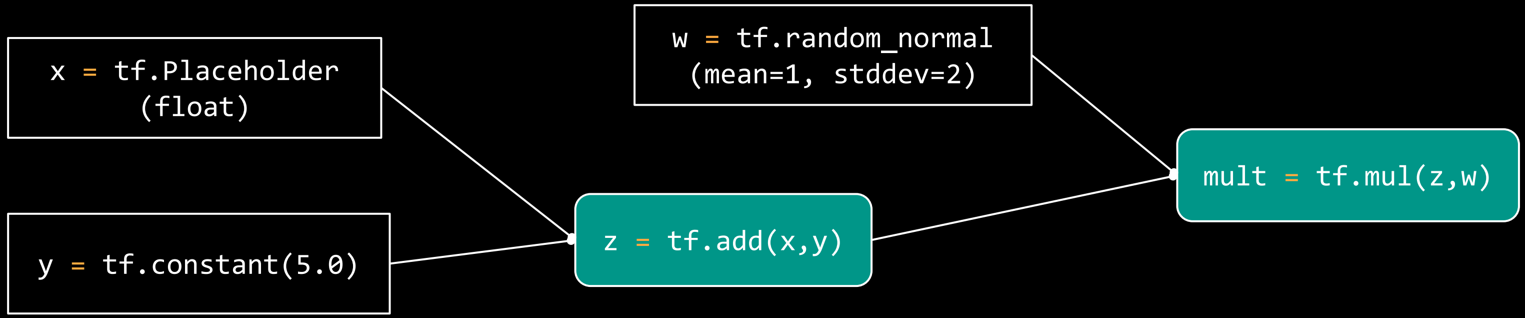
- Store parameters in graph
- Can be trainable (optimized during backprop) or untrainable
- Variety of initializers (e.g. constant, normal)

```
w = tf.random_normal
(mean=1, stddev=2)
```

TensorFlow Basics: Construction

Operations (`tf.Operation`)

- Takes in variable and/or outputs from other operations.
- Can be fed into other operations and linked in the graph.
- This includes linear algebraic operations and optimizers.



In Code

1. Create weights, including initialization

   W ~ Uniform(-1, 1); b = 0

2. Create input placeholder x

   m * 784 input matrix

3. Build flow graph

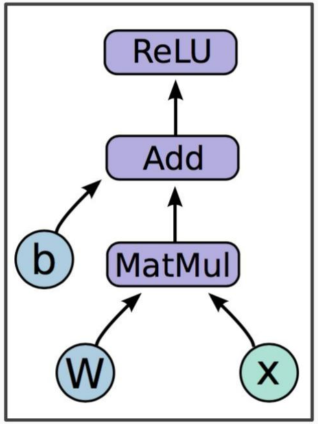
```python
import tensorflow as tf

b = tf.Variable(tf.zeros((100,)))
W = tf.Variable(tf.random_uniform((784, 100), -1, 1))

x = tf.placeholder(tf.float32, (100, 784))

h = tf.nn.relu(tf.matmul(x, W) + b)
```

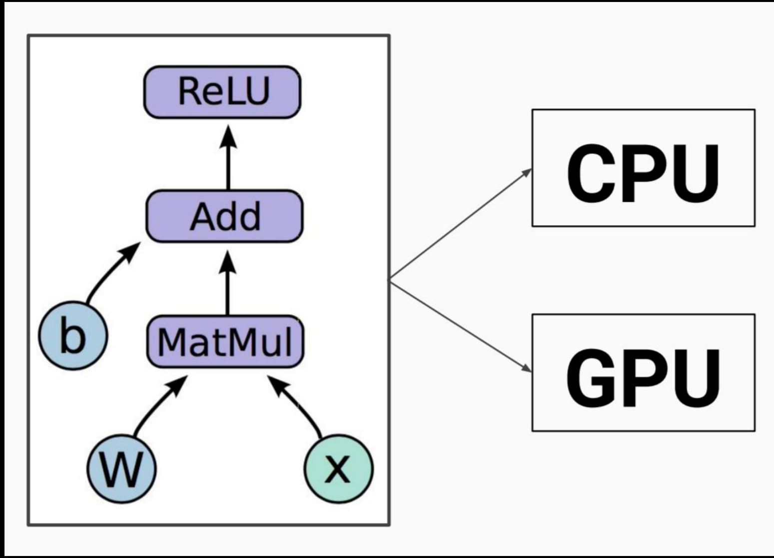
$$h = ReLU(Wx + b)$$



How do we run it?

So far we only talked about defining a graph

We can deploy this graph with a session - a binding to a particular execution context (e.g. CPU, GPU)

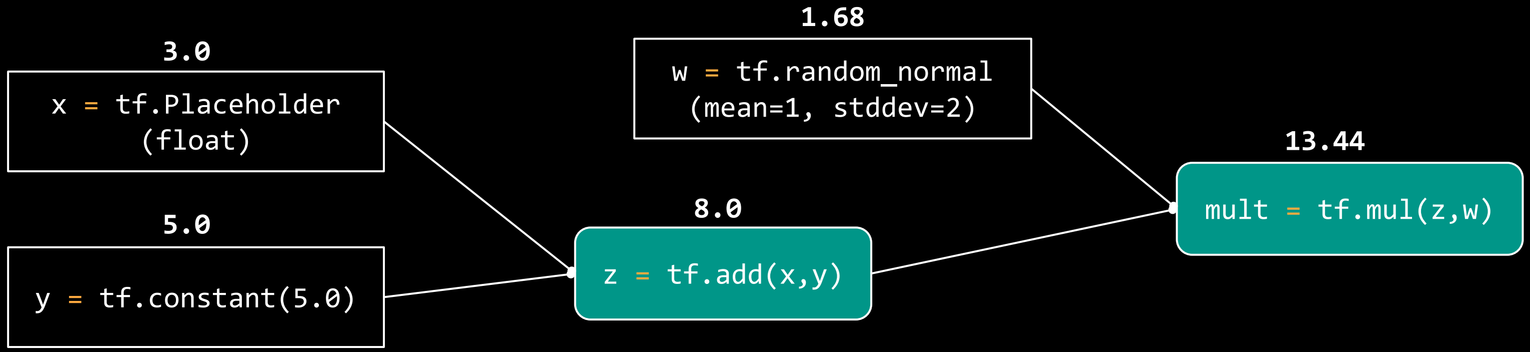


TensorFlow Basics: Execution

Sessions (tf.Session)

- Handles post-construction interactions with the graph
- Call the run method to evaluate tensors

```
sess = tf.Session()
sess.run(tf.global_variables_initializer())
sess.run(mult, feed_dict={
    x: 3.0}) # 13.44
```



Getting Output

sess.run(fetches, feeds)

Fetches: List of graph nodes. Return the outputs of these nodes.

Feeds: Dictionary mapping from graph nodes to concrete values. Specifies the value of each graph node given in the dictionary.

```
import numpy as np
import tensorflow as tf

b = tf.Variable(tf.zeros((100,)))
W = tf.Variable(tf.random_uniform((784, 100),
                -1, 1))

x = tf.placeholder(tf.float32, (100, 784))
h = tf.nn.relu(tf.matmul(x, W) + b)

sess = tf.Session()
sess.run(tf.initialize_all_variables())
sess.run(h, {x: np.random.random(100, 784)})
```

So far

We first built a graph using variables and placeholders

We then deployed the graph onto a session, which is the execution environment

Next we will see how to train a model

Define Loss?

Use placeholder for labels

Build loss node using labels and prediction

```
prediction = tf.nn.softmax(...)  #Output of neural network
label = tf.placeholder(tf.float32, [100, 10])

cross_entropy = -tf.reduce_sum(label * tf.log(prediction), axis=1)
```

Compute Gradients?

```
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
```

- **tf.train.GradientDescentOptimizer** is an Optimizer object
- **tf.train.GradientDescentOptimizer(lr).minimize(cross_entropy)** adds optimization operation to computation graph

Compute Gradients?

```
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
```

TensorFlow graph nodes have attached gradient operations

Gradient with respect to parameters computed with backpropagation

… automatically

So far

```
prediction = tf.nn.softmax(...)
label = tf.placeholder(tf.float32, [None, 10])

cross_entropy = tf.reduce_mean(-tf.reduce_sum(label * tf.log(prediction),
reduction_indices=[1]))

train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
```

Training the Model

1. Create session

2. Create training schedule

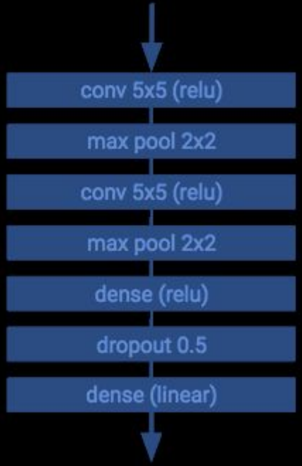
3. Run train_step

```
sess = tf.Session()
sess.run(tf.initialize_all_variables())

for i in range(1000):
    batch_x, batch_label = data.next_batch()
    sess.run(train_step, feed_dict={x: batch_x,
                                    label: batch_label}
```

TensorFlow for Deep Learning

TensorFlow has first class support for high and low-level deep learning `tf.Operations`.




```
x = tf.layers.conv2d(x, kernel_size=[5,5], ...)
x = tf.layers.max_pooling2d(x, kernel_size=[2,2], ...)
x = tf.layers.conv2d(x, kernel_size=[5,5], ...)
x = tf.layers.max_pooling2d(x, kernel_size=[2,2], ...)
x = tf.layers.dense(x, activation_fn=tf.nn.relu)
x = tf.layers.dropout(x, 0.5)
x = tf.layers.dense(x)
```

In Summary

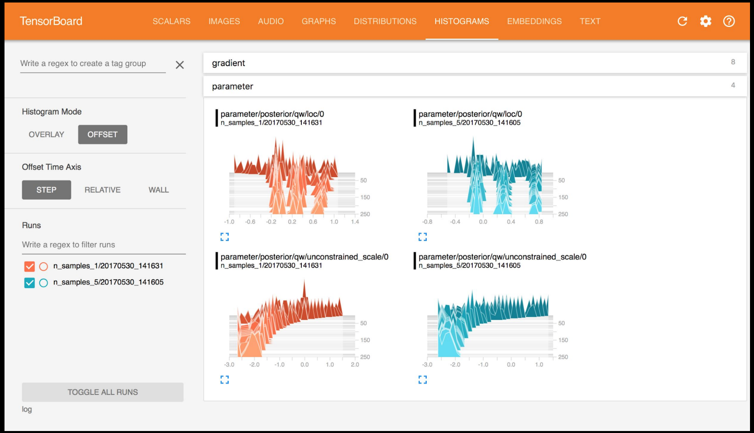
1. Build a graph
   a. Feedforward / Prediction
   b. Optimization (gradients and train_step operation)

2. Initialize a session

3. Train with session.run(train_step, feed_dict)

Demo

Visualizing Learning: TensorBoard

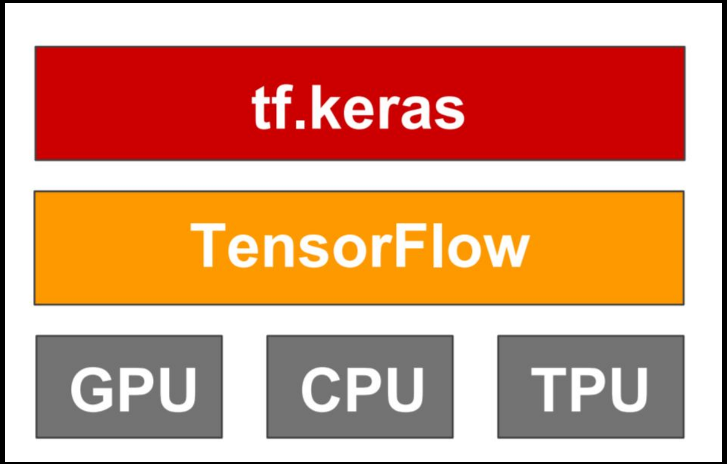


TensorBoard provides a visual representation of the graph and the performance of optimizers.

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 features



What’s special about Keras?

- A focus on user experience.
- Large adoption in the industry and research community.
- Multi-backend, multi-platform.
- Easy productization of models.

633 contributors

Microsoft

Google

NVIDIA

aws

250,000

Keras developers

> 2x

Year-on-year growth

Industry Use

NETFLIX
UBER
Google
instacart
HUAWEI
NVIDIA
Square
Expedia
Zocdoc
yelp

etc...

User Experience

Keras is an API designed for human beings, not machines. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.

This makes Keras easy to learn and easy to use. As a Keras user, you are more productive, allowing you to try more ideas than your competition, faster -- which in turn helps you win machine learning competitions.

This ease of use does not come at the cost of reduced flexibility: because Keras integrates with lower-level deep learning languages (in particular TensorFlow), it enables you to implement anything you could have built in the base language. In particular, as tf.keras, the Keras API integrates seamlessly with your TensorFlow workflows.

Using Keras

Three API Styles

- The Sequential Model
  - Dead simple
  - For single-input, single-output, sequential layer stacks
  - Good for 70+% of use cases

- The functional API
  - Like playing with Lego bricks
  - Multi-input, multi-output, arbitrary static graph topologies
  - Good for 95% of use cases

- Model Subclassing
  - Maximum flexibility
  - Larger potential error surface

The Sequential API

```python
import keras
from keras import layers

model = keras.Sequential()
model.add(layers.Dense(20, activation='relu', input_shape=(10,)))
model.add(layers.Dense(20, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

model.fit(x, y, epochs=10, batch_size=32)
```

The functional API

```python
import keras
from keras import layers

inputs = keras.Input(shape=(10,))
x = layers.Dense(20, activation='relu')(x)
x = layers.Dense(20, activation='relu')(x)
outputs = layers.Dense(10, activation='softmax')(x)

model = keras.Model(inputs, outputs)
model.fit(x, y, epochs=10, batch_size=32)
```

Model Subclassing

```
import keras
from keras import layers

class MyModel(keras.Model):

    def __init__(self):
        super(MyModel, self).__init__()
        self.dense1 = layers.Dense(20, activation='relu')
        self.dense2 = layers.Dense(20, activation='relu')
        self.dense3 = layers.Dense(10, activation='softmax')

    def call(self, inputs):
        x = self.dense1(x)
        x = self.dense2(x)
        return self.dense3(x)

model = MyModel()
model.fit(x, y, epochs=10, batch_size=32)
```

Demo