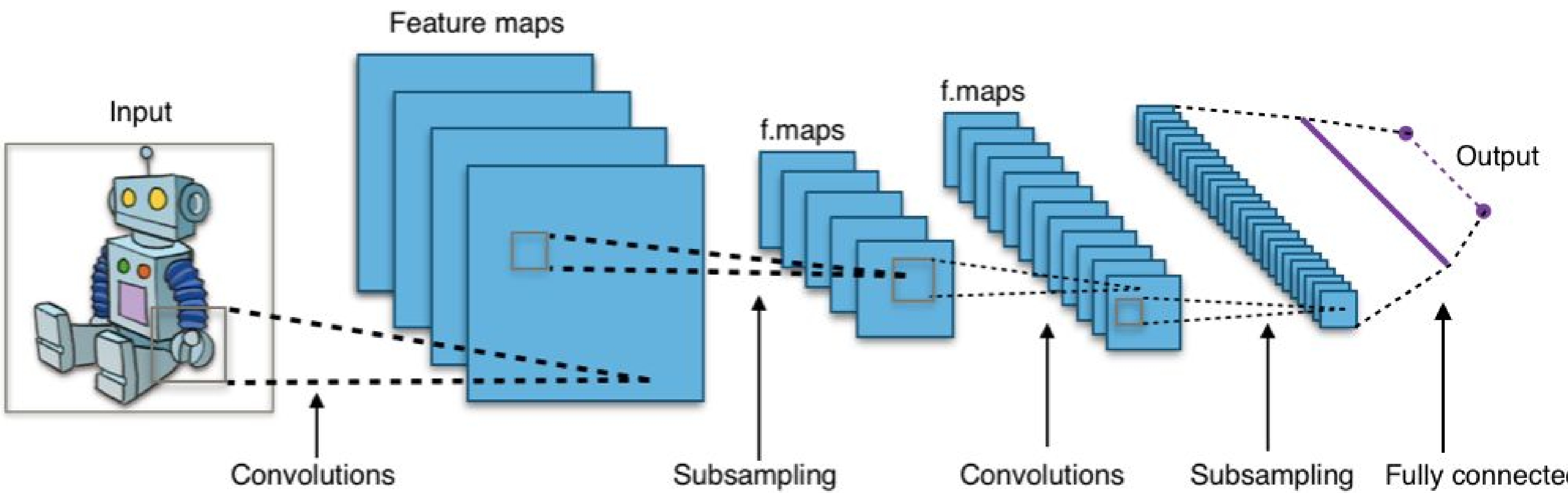
High-Level APIs in TensorFlow

"How to Data Science Good and Do Other Stuff Good, Too."

Silicon Valley Data Science - 7 December 2016

Andrew Zaldivar, Ph.D.
Senior Strategist, Trust & Safety

For this talk...



Demonstrate how to build a small neural network model in TensorFlow—then show how it can be done even easier using high level APIs built on top of TensorFlow

Image Source: https://en.wikipedia.org/wiki/Convolutional_neural_network

TensorFlow

- **Open source** Machine Learning library
- Especially useful for **Deep Learning**
- For research **and** production
- **Apache 2.0** license

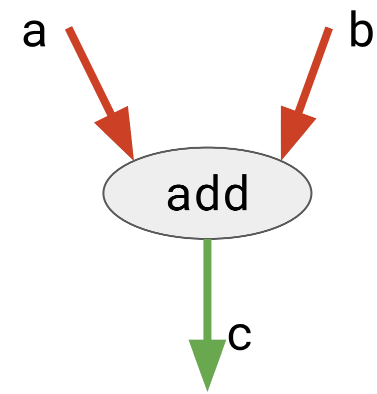
A multidimensional array.

TensorFlow

A graph of operations.

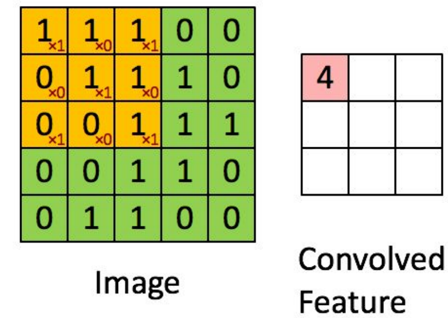
Build a Graph; Then Run It

```
...
c = tf.add(a, b)

...

session = tf.Session()
value_of_c = session.run(c, {a=1, b=2})
```



CNN in TensorFlow

```python
# Create functions that initializes our weights and biases
def weight_variable(shape):
  initial = tf.truncated_normal(shape, stddev=0.1)
  return tf.Variable(initial)

def bias_variable(shape):
  initial = tf.constant(0.1, shape=shape)
  return tf.Variable(initial)
```




```python
# Create functions that performs convolution and pooling operations
def conv2d(x, W):
  return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_2x2(x):
  return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                        strides=[1, 2, 2, 1], padding='SAME')
```

https://www.tensorflow.org/tutorials/mnist/pros/#build_a_multilayer_convolutional_network
https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

CNN in TensorFlow

```
# Implement the first convolutional layer
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])

x_image = tf.reshape(x, [-1,28,28,1])

h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)


# Implement the second convolutional layer
W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])

h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)
```

https://www.tensorflow.org/tutorials/mnist/pros/#build_a_multilayer_convolutional_network
https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

CNN in TensorFlow

```
# Add a fully-connected layer
W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])

h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)


# Apply dropout and add a softmax (readout) layer
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])

y_conv=tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)
```

https://www.tensorflow.org/tutorials/mnist/pros/#build_a_multilayer_convolutional_network
https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

CNN in TensorFlow

```
# Train and evaluate the model
cross_entropy = tf.reduce_mean(-tf.reduce_sum(
    y_ * tf.log(y_conv), reduction_indices=[1]))
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
correct_prediction = tf.equal(tf.argmax(y_conv,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(
    correct_prediction, tf.float32))
sess.run(tf.initialize_all_variables())
for i in range(20000):
  batch = mnist.train.next_batch(50)
  if i%100 == 0:
    train_accuracy = accuracy.eval(feed_dict={
        x:batch[0], y_: batch[1], keep_prob: 1.0})
    print("step %d, training accuracy %g"%(i, train_accuracy))
  train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})

print("test accuracy %g"%accuracy.eval(feed_dict={
    x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0}))
```

http://sebastianruder.com/optimizing-gradient-descent/

That Seemed Complicated... Can We Simplify?

```
import tensorflow as tf

# Specify that all features have real-value data
feature_columns = tf.contrib.learn.infer_real_valued_columns_from_input(data)

# Construct a deep neural network with 3 layers of 10, 20, & 10 neurons.
classifier = tf.contrib.learn.DNNClassifier(
    feature_columns=feature_columns, hidden_units=[10, 20, 10], n_classes=10)

# Fit model.
classifier.fit(data, labels, batch_size=100, steps=10000)

# Evaluate accuracy.
accuracy_score = classifier.evaluate(x=test_set.data,
                                     y=test_set.labels)["accuracy"]
print('Accuracy: {0:f}'.format(accuracy_score))
```

https://www.tensorflow.org/tutorials/tflearn/

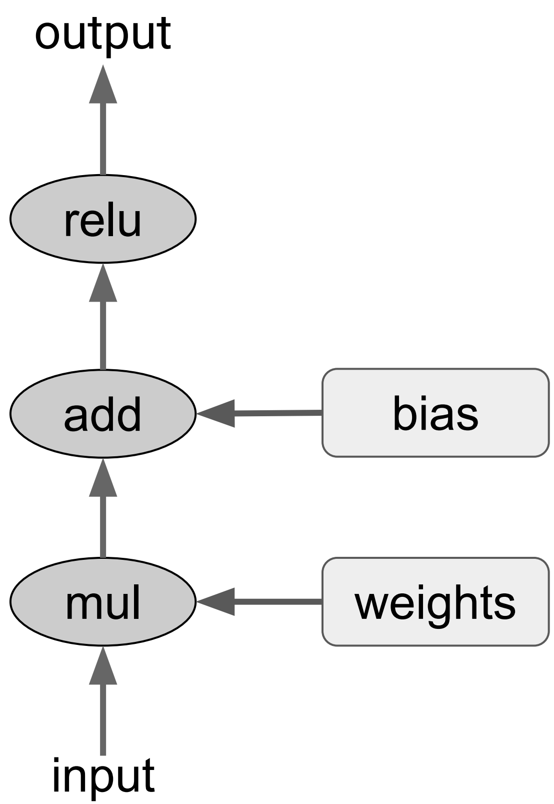
That Seemed Complicated... Can We Simplify?

```
import tensorflow as tf
slim = tf.contrib.slim

# Define a simple CNN
def my_cnn(images, num_classes, is_training):
    with slim.arg_scope([slim.max_pool2d], kernel_size=[3, 3], stride=2):
        net = slim.conv2d(images, 64, [5, 5])
        net = slim.max_pool2d(net)
        net = slim.conv2d(net, 64, [5, 5])
        net = slim.max_pool2d(net)
        net = slim.flatten(net)
        net = slim.fully_connected(net, 192)
        net = slim.fully_connected(net, num_classes, activation_fn=None)
        return net

# Create the model
num_classes = 10
logits = my_cnn(images, num_classes, is_training=True)
probabilities = tf.nn.softmax(logits)
```

https://github.com/tensorflow/models/blob/master/slim/slim_walkthough.ipynb

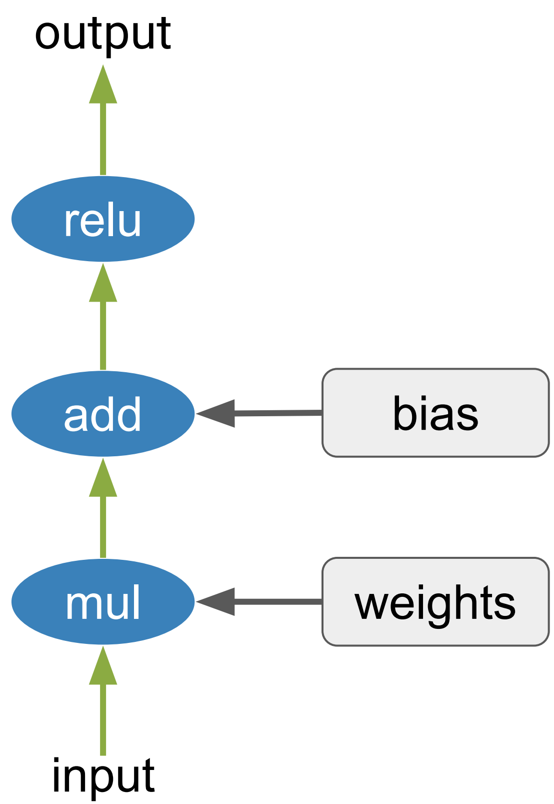
Google

Awesome! What's Going On?



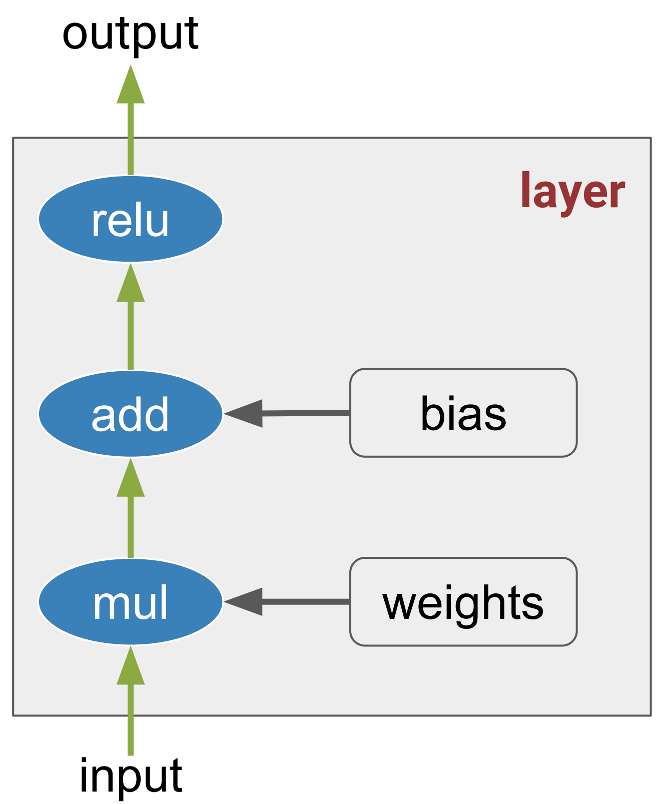
Ops Are What Makes TensorFlow Flexible

Nodes in the graph are called **Ops** (short for **operations**). An op takes zero or more **Tensors**, performs some computation, and produces zero or more **Tensors.**



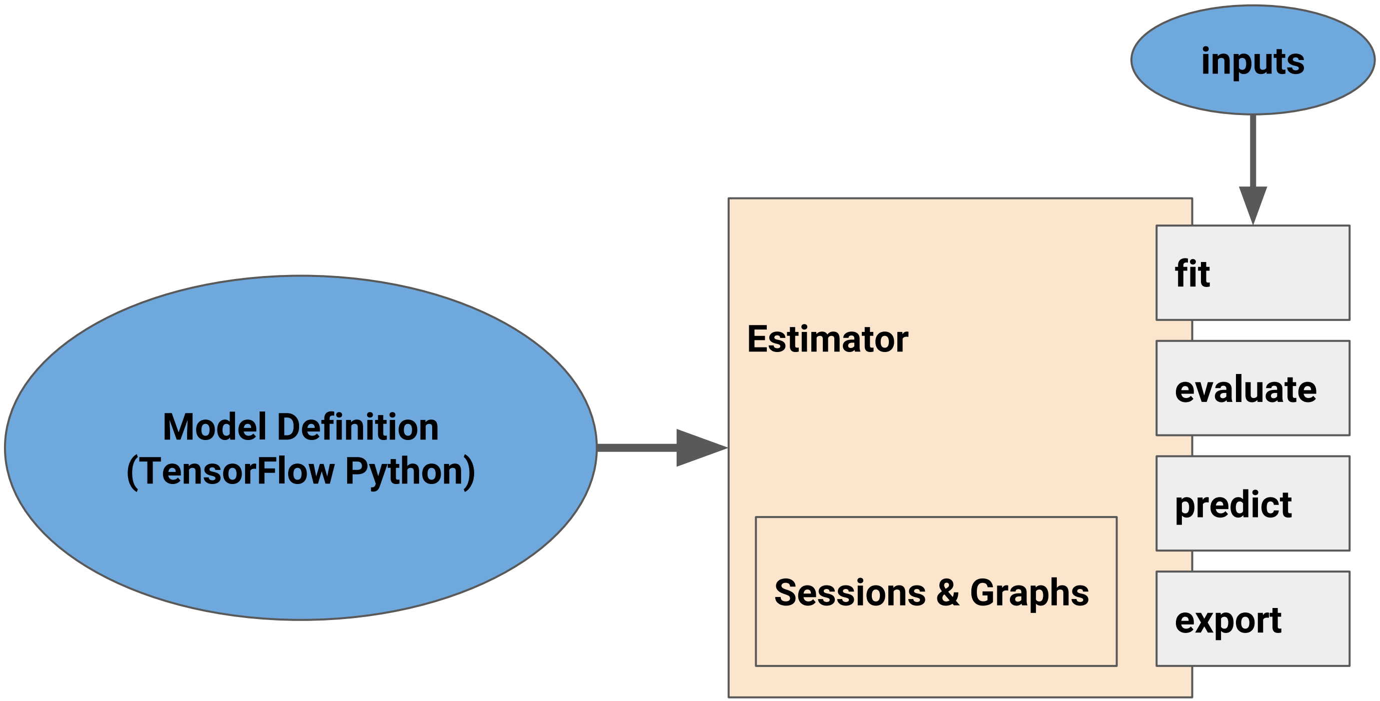
Creating Bigger Ops Adds Usability

Nodes in the graph are called **Ops** (short for **operations**). An op takes zero or more **Tensors**, performs some computation, and produces zero or more **Tensors.**



High-level APIs offer bigger **Ops** that make it easier to build models. In this example, this ReLU operation is encapsulated inside a **layer**.

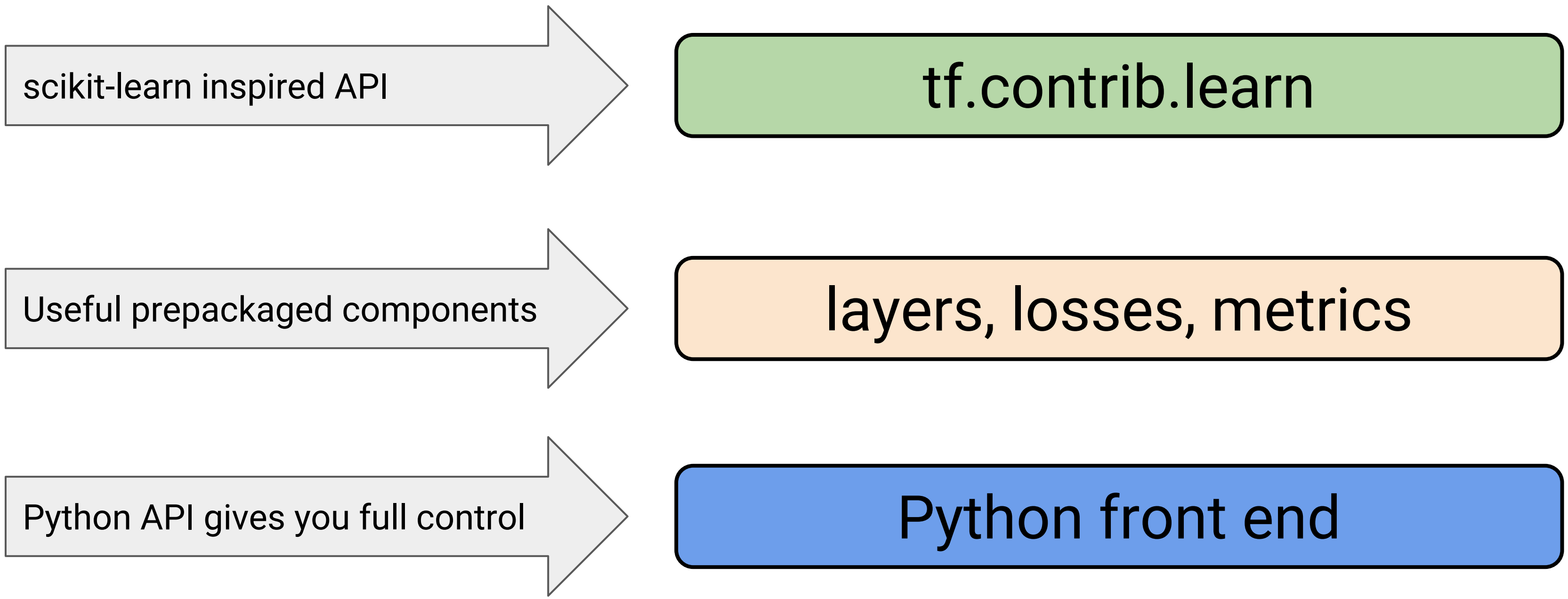
TF Learn Estimator Focuses on the Model



Estimators Modeled After Scikit-Learn Interface

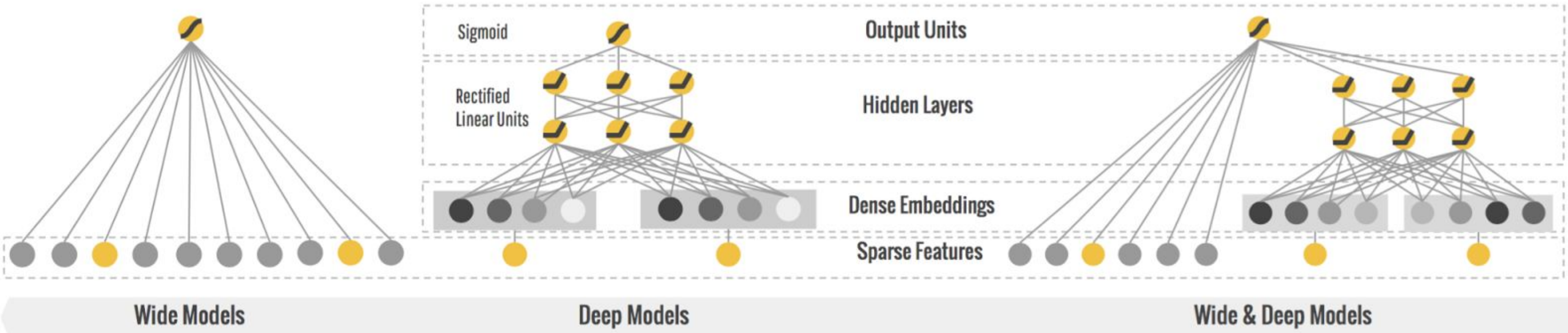
```
# In TF Learn
from tensorflow.contrib import learn
estimator = learn.LinearRegressor(feature_columns, model_dir)
estimator.fit(input_fn=input_fn_train)
estimator.evaluate(input_fn=input_fn_eval)
estimator.predict(x=x)
estimator.export(export_dir)

# In Scikit-Learn
from sklearn import linear_model
import pickle
regression = linear_model.LinearRegression()
regression.fit(X_train, Y_train)
regression.predict(X_test)
pickle.dump(regression)
```

Toolkit Hierarchy



Wide and Deep in TF Learn

```
estimator = tf.contrib.learn.DNNLinearCombinedClassifier(
    model_dir=YOUR_MODEL_DIR,
    linear_feature_columns=wide_columns,
    dnn_feature_columns=deep_columns,
    dnn_hidden_units=[100, 50])
```

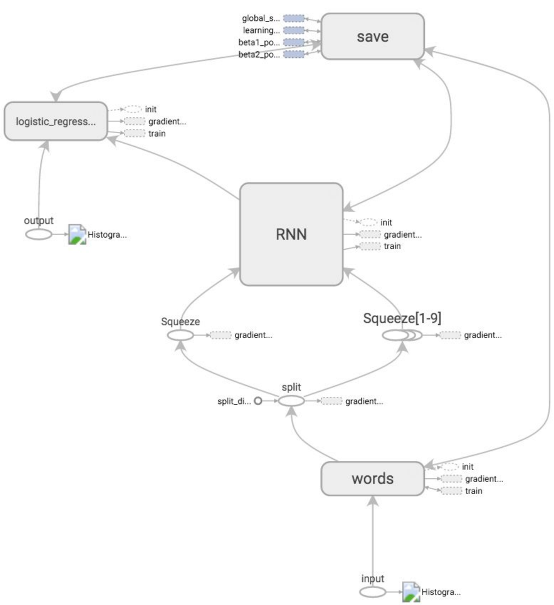


https://www.tensorflow.org/tutorials/wide_and_deep/
https://arxiv.org/abs/1606.07792

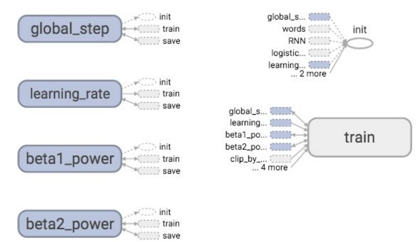
Logging and Monitoring with TF Learn

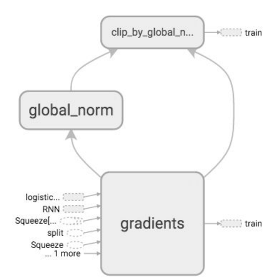
```
est = learn.DNNClassifier(..., model_dir="/tmp/my_model")
In Terminal: tensorboard --logdir=/tmp/tf_examples/my_model_1
```

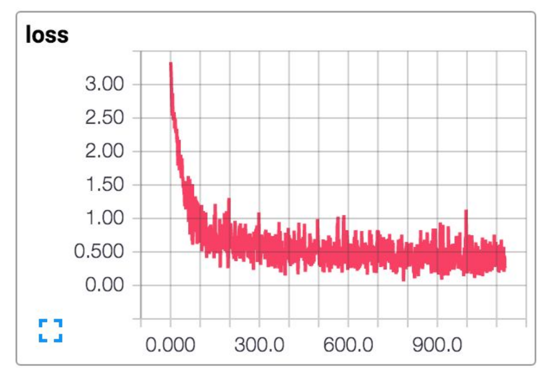
Main Graph



Auxiliary nodes







Available TF Learn Estimators

Linear{Regressor, Classifier}

DNN{Regressor, Classifier}

DNNLinearCombined{Regressor, Classifier}

KMeans

SVM

TensorForest

...and more coming very soon!

TF Slim Model Definition

TF Slim is composed of several parts which were design to exist independently:

- arg_scope: let's you define arguments for specific operations within a scope
- data: dataset definition, data providers, parallel readers and decoding utilities
- evaluation: routines for evaluating models
- layers: high level layers for building models
- learning: routines for training models
- losses: commonly used loss functions
- metrics: popular evaluation metrics
- nets: popular network definitions such as VGG or AlexNet
- queues: context manager for easily/safely starting/closing QueueRunners
- regularizers: weight regularizers
- variables: convenience wrappers for variable creation and manipulation

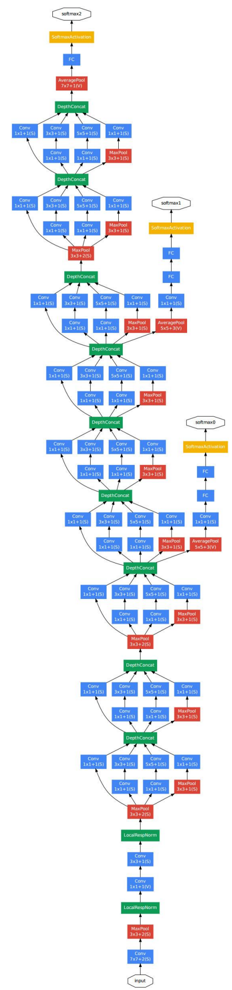
Displaying Data in TF Slim

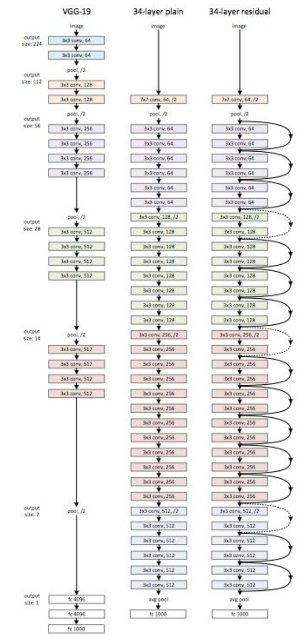
```
import tensorflow as tf
from datasets import flowers

slim = tf.contrib.slim

with tf.Graph().as_default():
    dataset = flowers.get_split('train', flowers_data_dir)
    data_provider = slim.dataset_data_provider.DatasetDataProvider(
        dataset, common_queue_capacity=32, common_queue_min=1)
    image, label = data_provider.get(['image', 'label'])
```

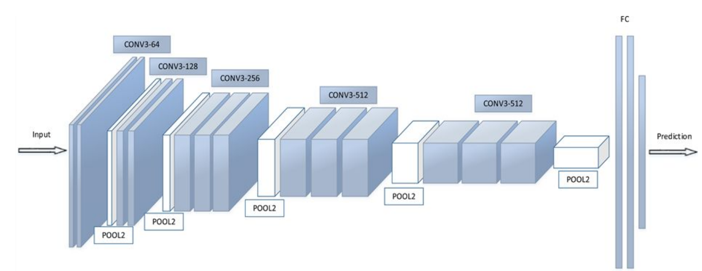
Popular Network Architectures in TF Slim

You can fine-tune a **pre-trained model** on an entirely new dataset or even a new task.

- Inception V1-V4
- Inception-ResNet-v2
- ResNet 50/101/152
- VGG 16/19







TensorFlow High-Level APIs: Takeaways

- Makes it easier to build models
- Brings TensorFlow to more users
- Make model development faster
- Promote best practices
- Try TF Slim if you want something that's easy to use, extensible, and let's you mix and match with TF
- Try TF Learn if you're looking to quickly configure common model types

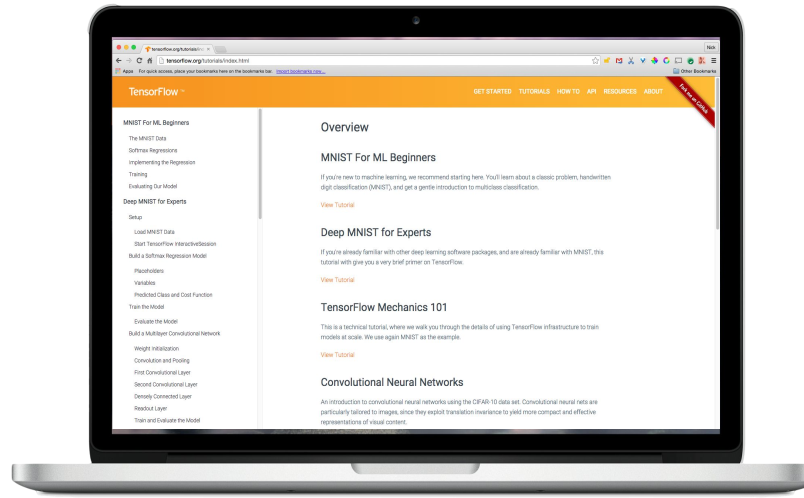
For More Information

Tutorials and code

www.tensorflow.org

TF Learn

- TF Learn README
- TF Learn Quickstart
- TF Learn Jupyter Notebook
- Google Research Blog on Wide and Deep Learning with TF Learn API



TF Slim

- TF Slim README
- TF Slim Jupyter Notebook
- Google Research Blog on TF Slim

Thanks & Have Fun!