A Tale of Three Deep Learning Frameworks: TensorFlow, Keras, and PyTorch

Brooke Wenig
Jules S. Damji

Spark + AI Summit, London 4 October 2018

databricks

About Us . . .

Jules S. Damji

Apache Spark Developer & Community Advocate @Databricks

Program Chair Spark + AI Summit

Software engineering @Sun Microsystems, Netscape, @Home, VeriSign, Scalix, Centrify, LoudCloud/Opsware, ProQuest

https://www.linkedin.com/in/dmatrix

@2twitme

Brooke Wenig

Machine Learning Practice Lead @ Databricks

Data Science @ Splunk & MyFitnessPal

MS Machine Learning (UCLA)

Fluent in Chinese

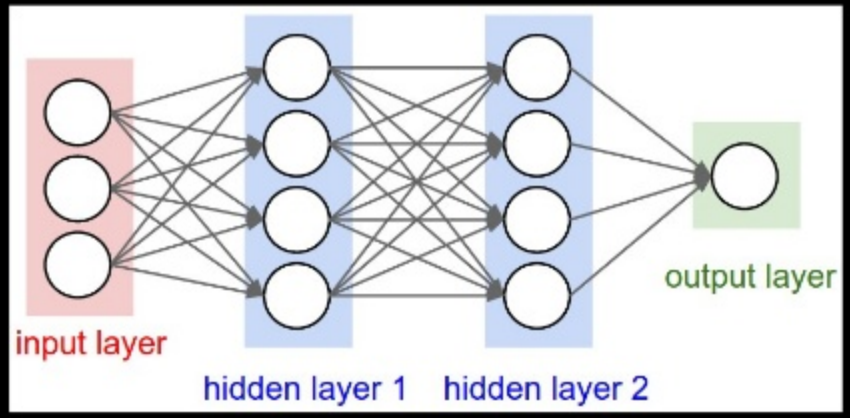
https://www.linkedin.com/in/brookewenig/

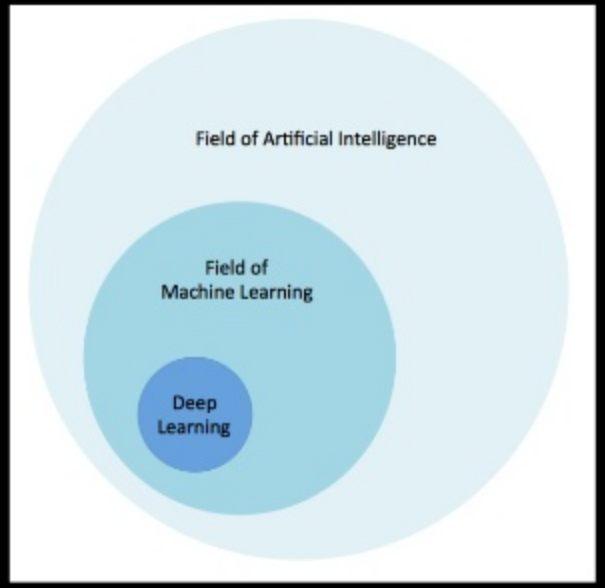
Agenda for Today's Talk

- What's Deep Learning and Why?
- Short Survey of 3 DL Frameworks
  - TensorFlow
  - Keras
  - PyTorch
- Training Options
  - Single Node
  - Distributed
- Q&A

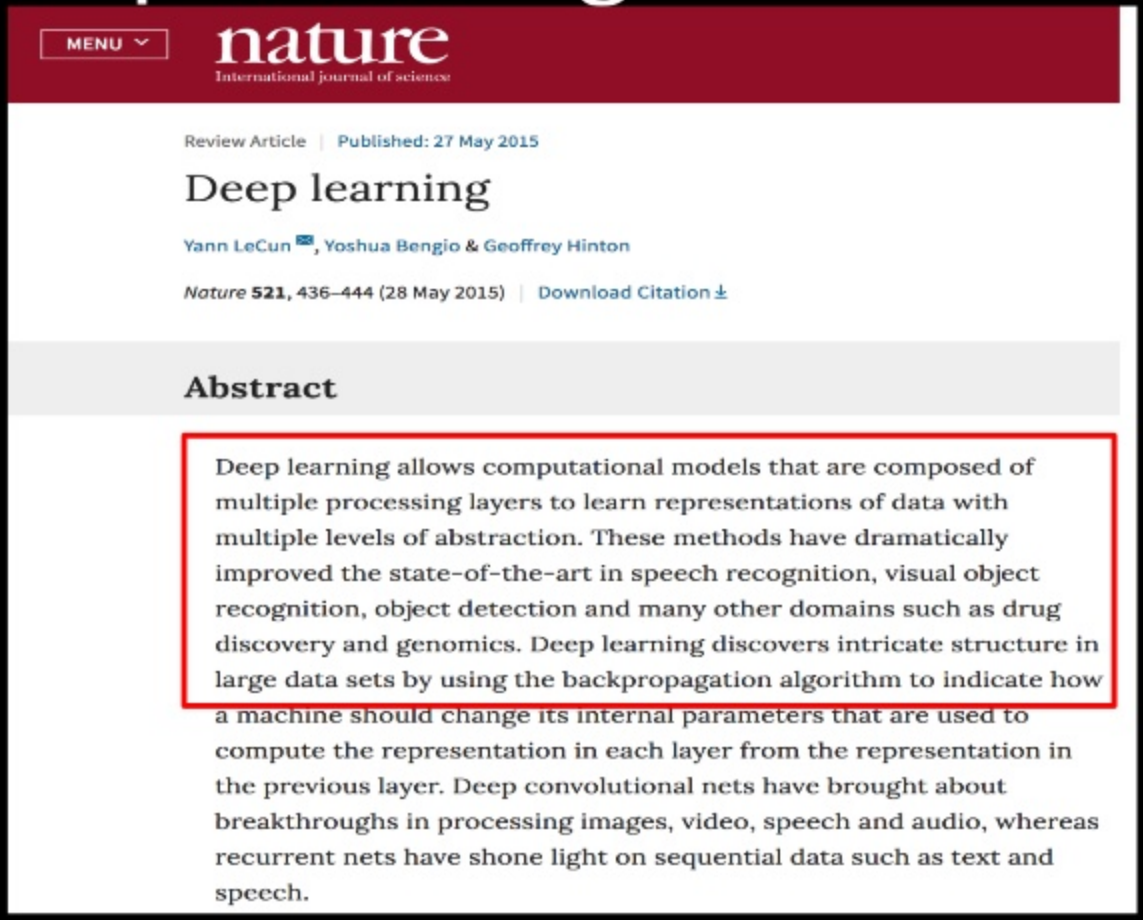
What is Deep Learning?

"Composing representations of data in a hierarchical manner"





Why Deep Learning?



MENU

nature
International journal of science

Review Article | Published: 27 May 2015

Deep learning

Yann LeCun, Yoshua Bengio & Geoffrey Hinton

Nature **521**, 436–444 (28 May 2015) | Download Citation

Abstract

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics. Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer. Deep convolutional nets have brought about breakthroughs in processing images, video, speech and audio, whereas recurrent nets have shone light on sequential data such as text and speech.

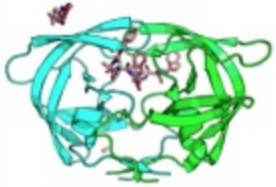
Applications

Machine Learning is Everywhere?

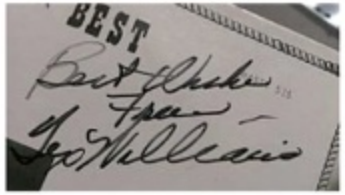
AlphaGo

NETFLIX

Recommendation systems



Drug discovery



Character recognition

2σ TWO SIGMA

Hedge fund stock predictions

Voice assistants

Assisted driving

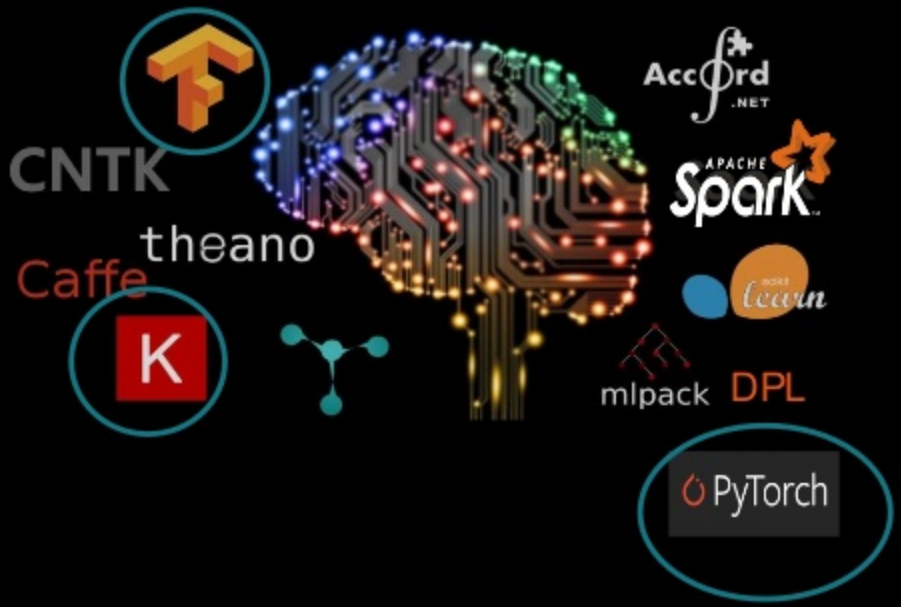
Face detection/recognition

Cancer diagnosis

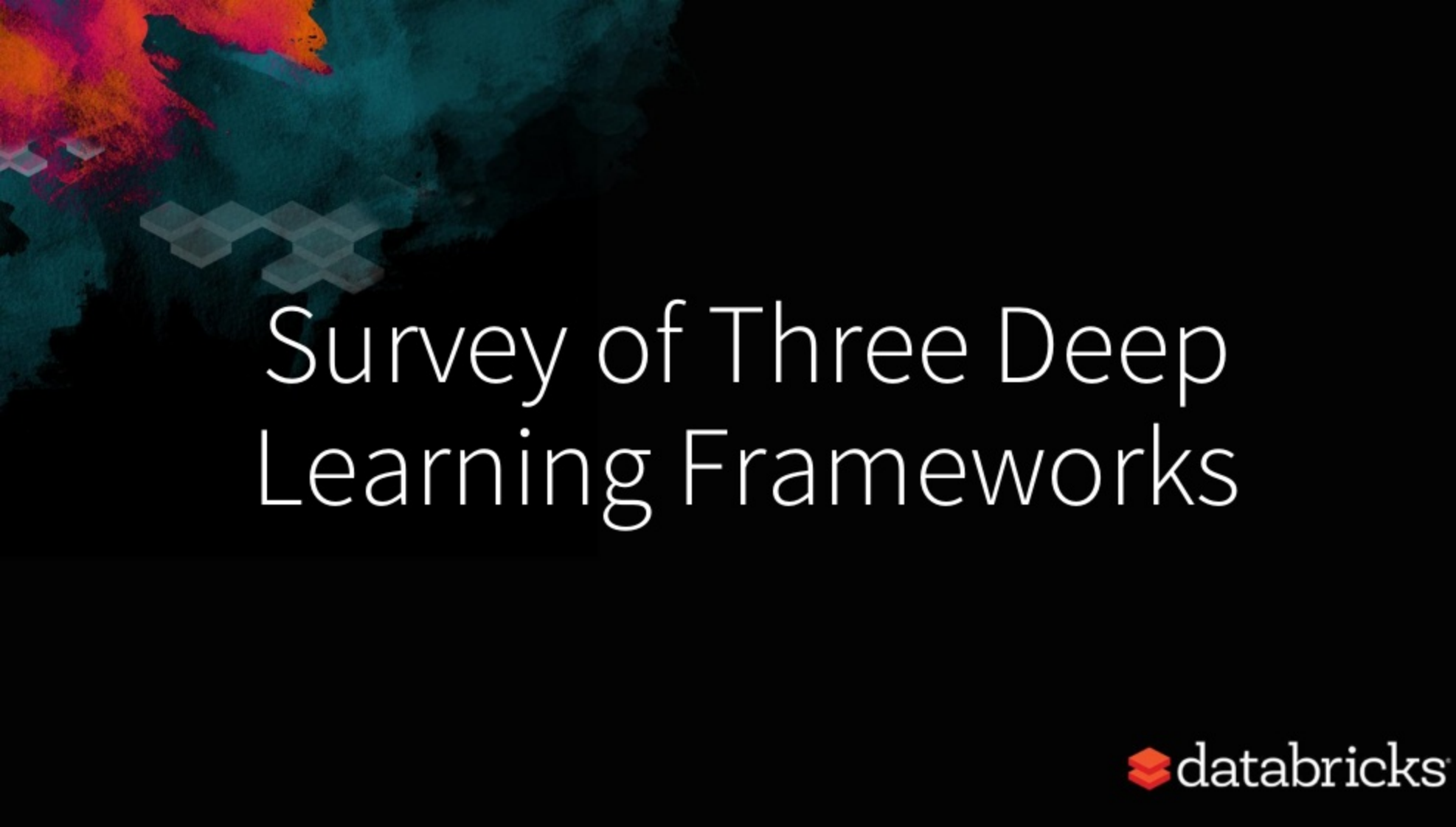
Source : MIT

databricks

Zoo of DL Frameworks: Which One?

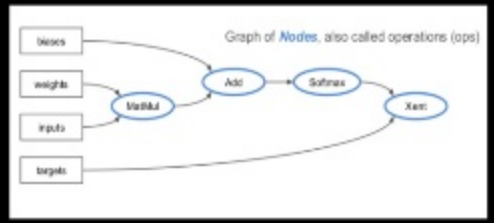


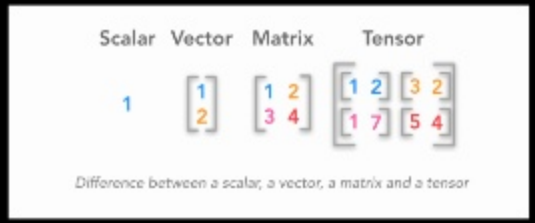
Survey of Three Deep Learning Frameworks



What's TensorFlow?

- Open source from Google, 2015
  - Current v1.12 API
  - 2.0 Coming Soon... :)
  - Declarative Toolkit
- Fast: Backend C/C++
- Data flow graphs
  - Nodes are functions/operators
  - Edges are input or data (tensors)
  - Lazy execution
  - Eager execution (1.7)





TensorFlow Key API Concepts

- Constants
- Variables
- Placeholders
- Operations
- Sessions
- Tensors
- Graphs

```
x = tf.constants (42, name= 'x')
w = tf.Variable(1.34, name='w')
input= tf.Placeholder("float")
c = tf.add(x , w); m = tf.matmul(a, b) ...
with tf.Session([URI]) as sess:
 1, [1, 2], [[2, 3], [4, 5]] ...
g = tf.Graph("my_graph")
 with g.as_default():
        c = tf.add(x,w)
         m = tf.matmul(a, b)
```

TensorFlow Code


```
import tensorflow as tf

a = tf.placeholder(tf.float32, shape=(2,1))
b = tf.placeholder(tf.float32, shape=(1,2))

c = tf.matmul(a, b)

sess = tf.Session()
print(sess.run(c, {a: [[1],[2]], b:[[3,4]]}))
```

Create TF placeholder types, a & b
Define their input shape as tensors

TF matmul matrix operation

Create a TF Session
Run the session, with input parameters for place holders ‘a’ & ’b’

‘c’ as an operation won’t to run until sess.run() Lazily evaluated.

```
[[3. 4.] [6. 8.]]
```

TF session output

TensorFlow Code: *MNIST*


```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.matmul(x, W) + b
y_ = tf.placeholder(tf.int64, [None])
...
# Define loss and optimizer
cross_entropy = tf.losses.sparse_softmax_cross_entropy(labels=y_, logits=y)
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
#create session, train, and evaluate
sess = tf.InteractiveSession()
tf.global_variables_initializer().run()
# Train
for _ in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
# Test trained model
correct_prediction = tf.equal(tf.argmax(y, 1), y_)
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
...
print(sess.run(accuracy, feed_dict={
        x: mnist.test.images,
        y_: mnist.test.labels
    }))
```

Use tf input_data modules for MNIST

TF placeholders & variables

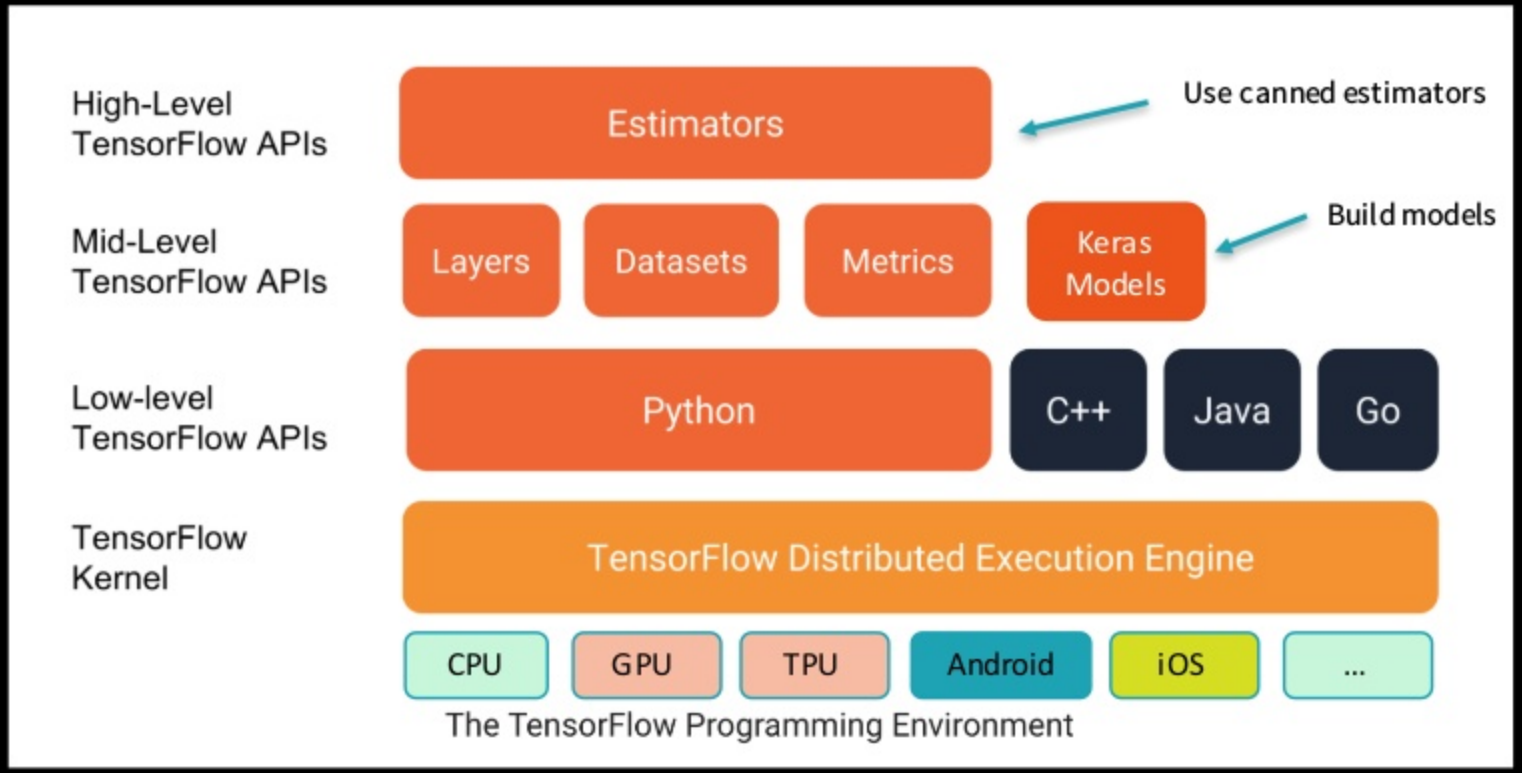
Define our model
TF variable for predicted value y'

Define our loss function: cross_entropy
Use Gradient Descent Optimizer

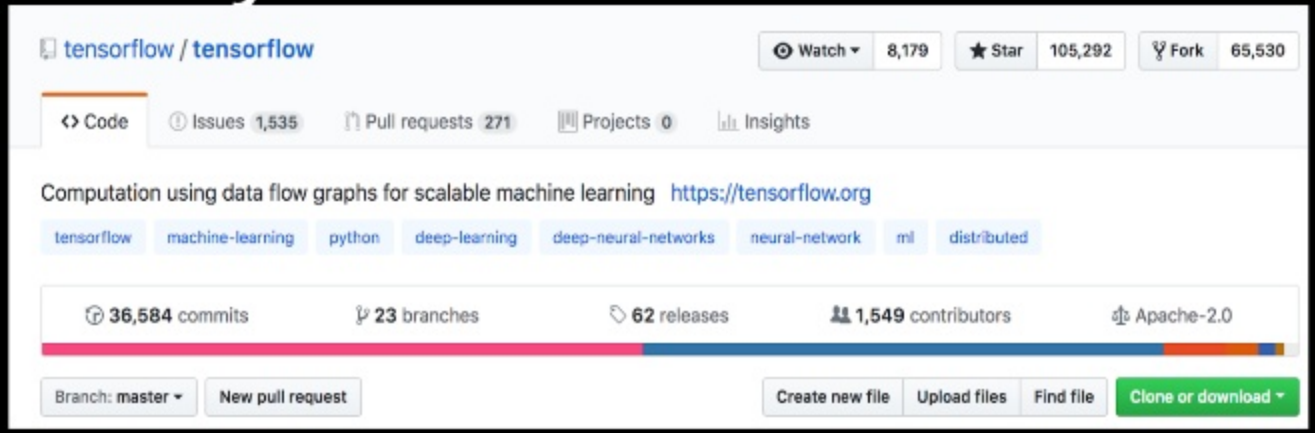
Train or evaluate the model

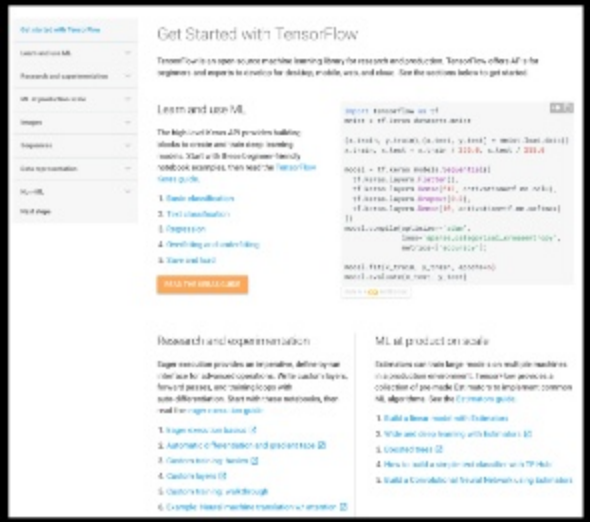
https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/tutorials/mnist/mnist_softmax.py

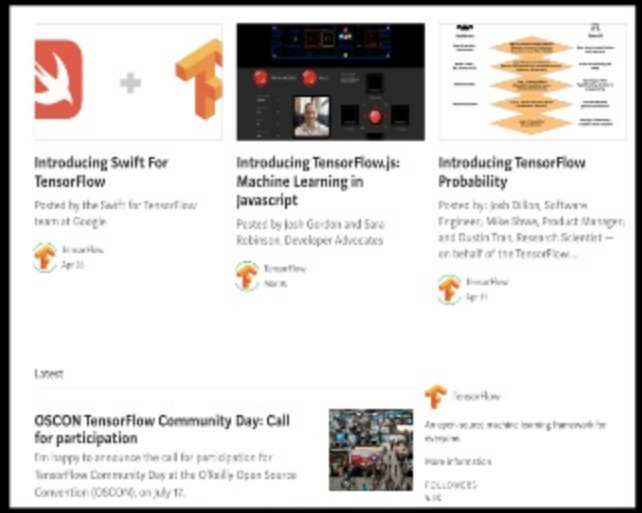
TensorFlow Programming Stack



Why TensorFlow: Community



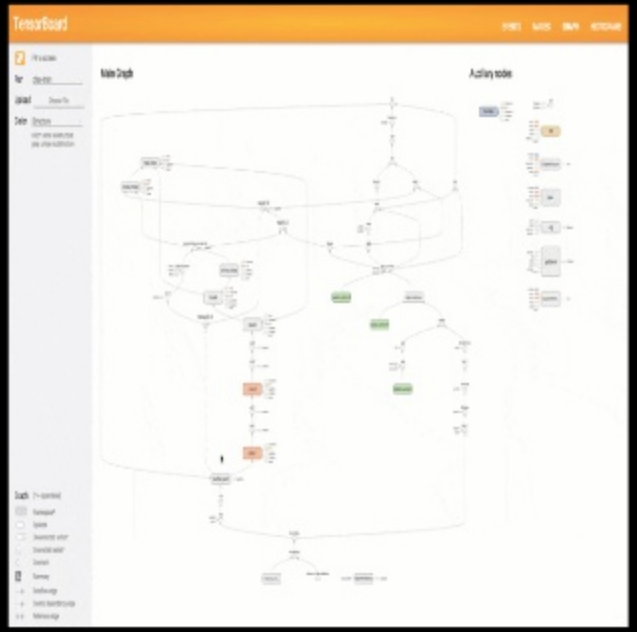


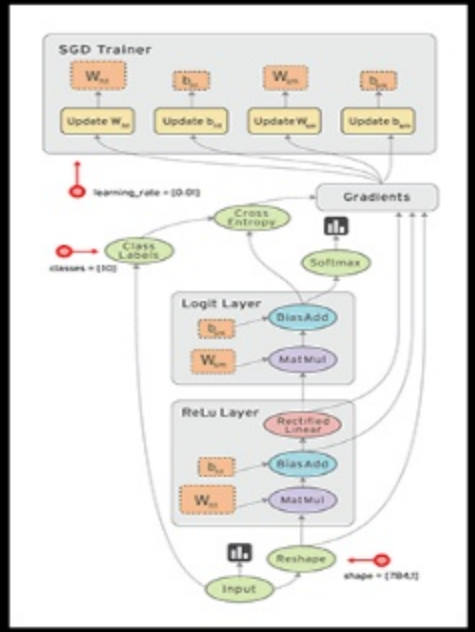


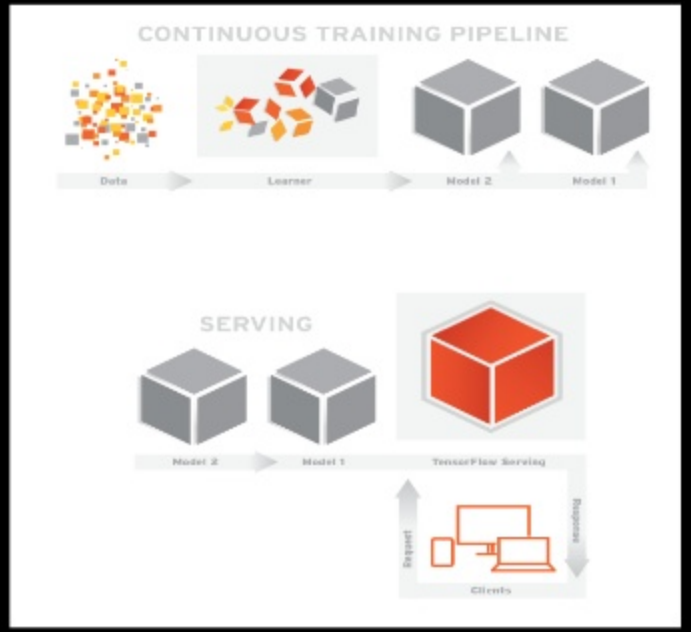
- 105K+ stars!
- 11+M downloads
- Popular open-source code
- TensorFlow Hub & Blog
  - Code Examples & Tutorials!
  - Learn + share from others

databricks

Why TensorFlow: Tools





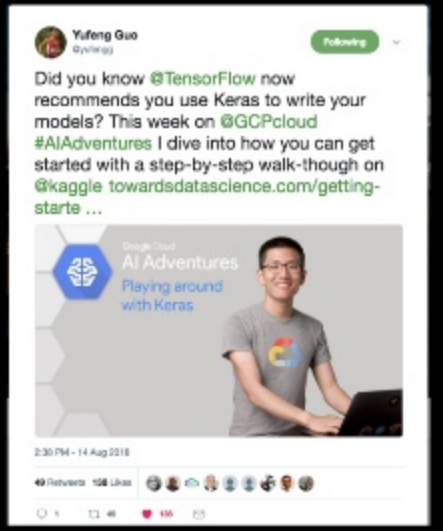


[Visualize Tensors flow]

[Deploy + Serve Models TFX]

TensorFlow: We Get it … So What?

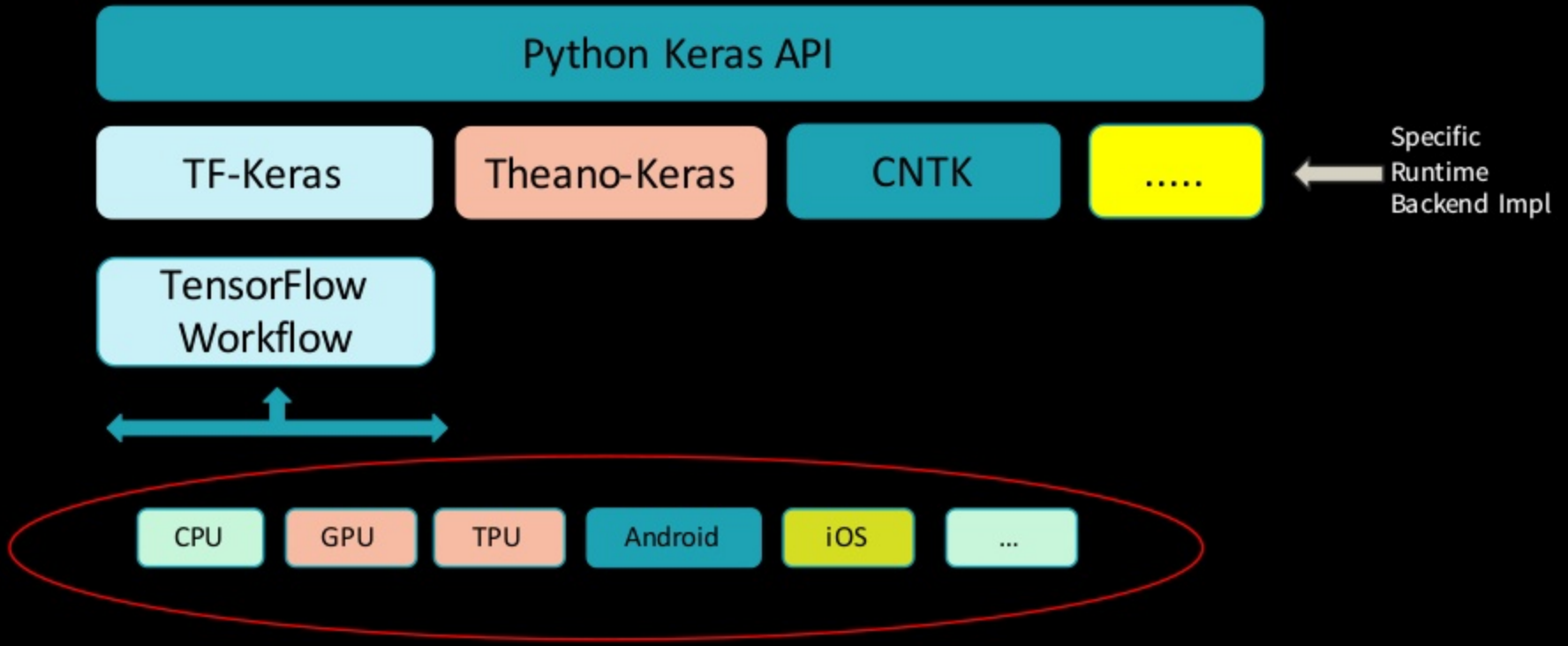
- Steep learning curve, *but powerful!!*
- Low-level APIs, *but offers control!!*
- Expert in Machine Learning, *just learn!!*
- Yet, high-level Estimators help, *you bet!!*
- Yeah, TensorFlow 2.0, *ease-of-use, eager execution!*
- Better, Keras integration helps, *indeed!!*



What's Keras?

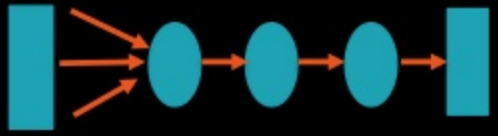
- Open source Python Library APIs for Deep Learning
  - Current v2.2.2 APIs François Chollet (Google)
- APIs : with TensorFlow, CNTK and Theano Backends
- *Easy to Use High-Level Declarative APIs!*
  - Build layers
    - Great for Neural Network Applications
      - CNN
      - RNN & LSTM
- *Fast Experimentation, Modular & Extensible!*

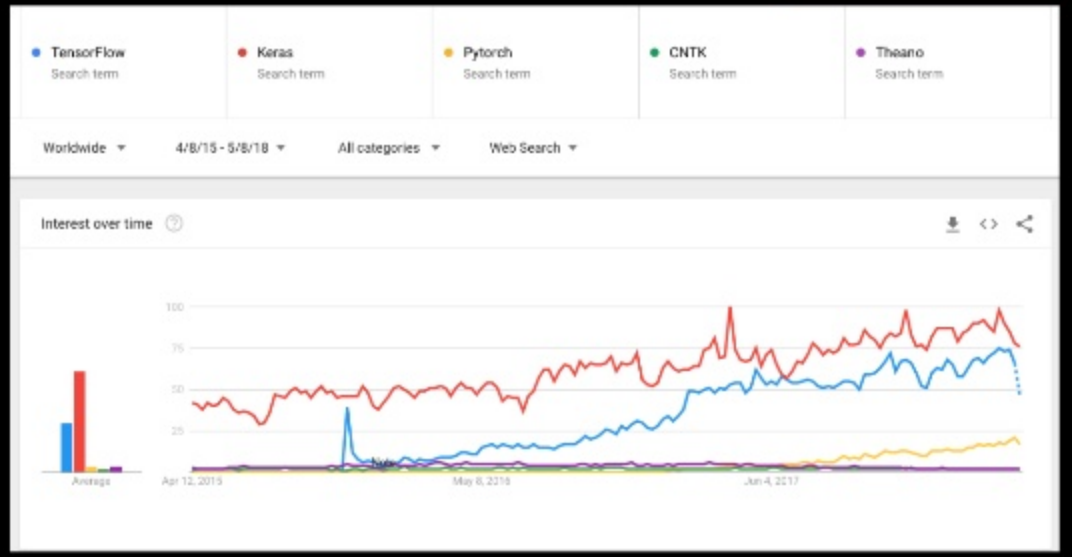
Keras Programming Stack



Why Keras?

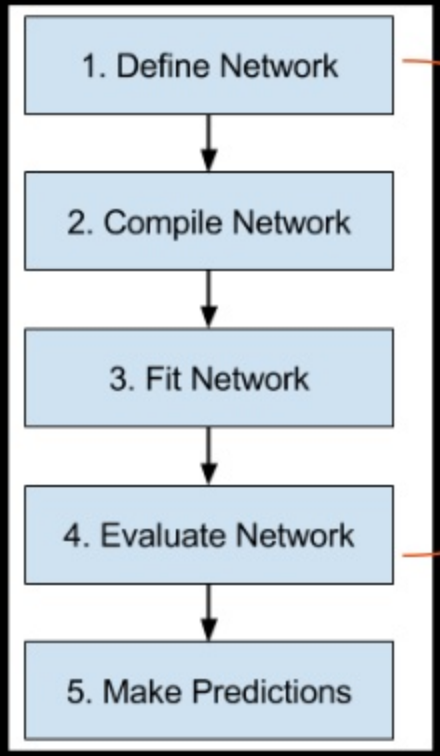
- *Focuses on Developer Experience*
- Popular & Broader Community
- Supports multiple backends
- Modularity
  - Sequential
  - Functional






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

Keras Code: *MNIST*




```
from keras import models
from keras import layers

mnist = tf.keras.datasets.mnist
(train_images, train_labels),(test_images, test_labels)=
   prepare_data(mnist.load_data())

network= models.Sequential()
network.add(layers.Dense(512, activation='relu',
                         input_shape(28 * 28,)))
 network.add(layers.Dense(10, activation='softmax'))


 network.compile(optimizer='rmsprop',
                 loss='categorical_crossentropy',
                 metrics=['accuracy'])


 network.fit(train_images, train_labels, epochs=5,
                  batch_size=128 )


 results = network.evaluate(test_images, test_labels)


 predictions = network.predict(new_images)
```

Set up code & use dataset

Define Network

Compile Network

Fit Network

Evaluate Network

Make Predictions

databricks

Another a Simple Network Model: Which code is easier to read?

Python: The Language of Deep Learning?

```
with tf.variable_scope('conv1') as scope:
  kernel = _variable_with_weight_decay('weights',
                                       shape=[5, 5, 3, 64],
                                       stddev=5e-2,
                                       wd=None)
  conv = tf.nn.conv2d(images, kernel, [1, 1, 1, 1], padding='SAME')
  biases = _variable_on_cpu('biases', [64], tf.constant_initializer(0.0))
  pre_activation = tf.nn.bias_add(conv, biases)
  conv1 = tf.nn.relu(pre_activation, name=scope.name)
  _activation_summary(conv1)

# pool1
pool1 = tf.nn.max_pool(conv1, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1],
                       padding='SAME', name='pool1')
# norm1
norm1 = tf.nn.lrn(pool1, 4, bias=1.0, alpha=0.001 / 9.0, beta=0.75,
                  name='norm1')

# conv2
with tf.variable_scope('conv2') as scope:
  kernel = _variable_with_weight_decay('weights',
                                       shape=[5, 5, 64, 64],
                                       stddev=5e-2,
                                       wd=None)
  conv = tf.nn.conv2d(norm1, kernel, [1, 1, 1, 1], padding='SAME')
  biases = _variable_on_cpu('biases', [64], tf.constant_initializer(0.1))
  pre_activation = tf.nn.bias_add(conv, biases)
  conv2 = tf.nn.relu(pre_activation, name=scope.name)
  _activation_summary(conv2)

# norm2
```

TensorFlow

```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

Keras

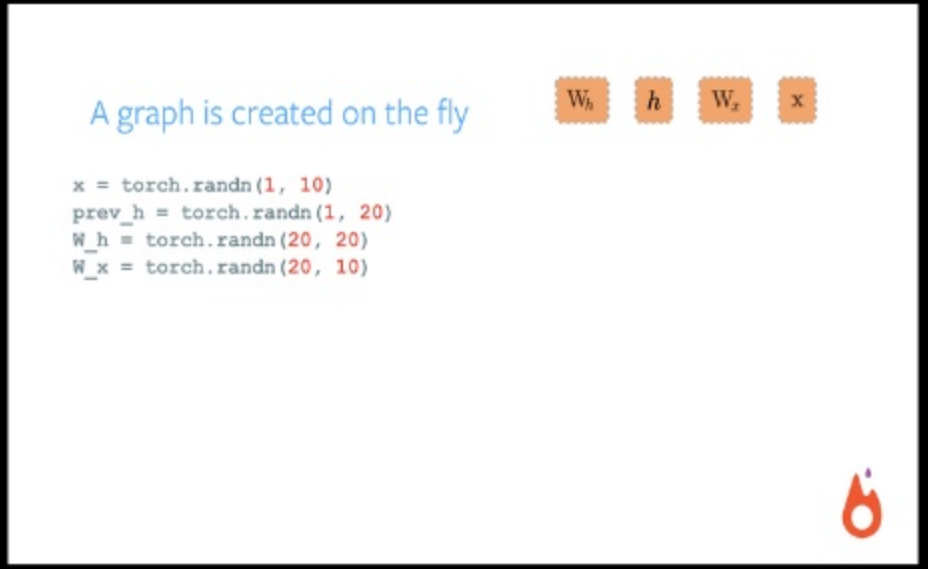
```
from torch.autograd import Variable
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

PyTorch

What is PyTorch?

PyTorch

- Open source from Facebook, 2017
  - v1.0 dev release
  - Primarily a Python Package
  - Tensor Computations
    - Torch.tensor -> CPU, GPU/CUDA
  - *Dynamic* NN: Tape-based Autograd
  - Graph Based *Dynamic* Computations
  - Imperative Toolkit



PyTorch Programming Stack

Python API

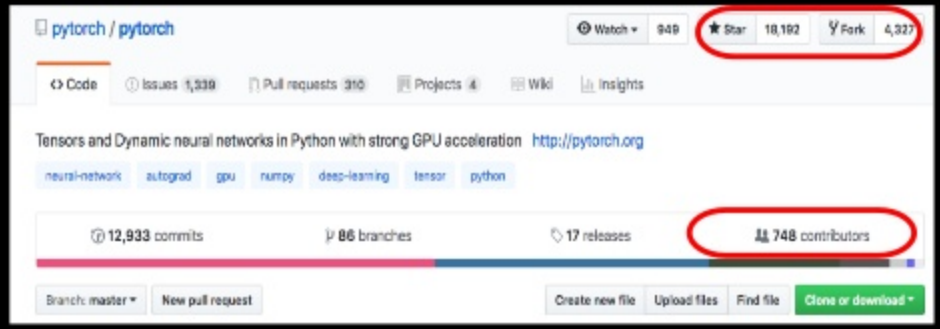
Auto Differentiation Engine	Numpy Library (GPU Support)	Gradient Based Optimized Package	Utilities (Data, Text, Video..)
Deep Learning & Reinforcement Learning	Numpy Alternative Ndarray <->torch.Tensor	Optimized: Adam, Adamax, RMSProp, SGD, LBFGS ...	Data loading: SciPy, SpaCy, OpenCV, PIL, torchvision

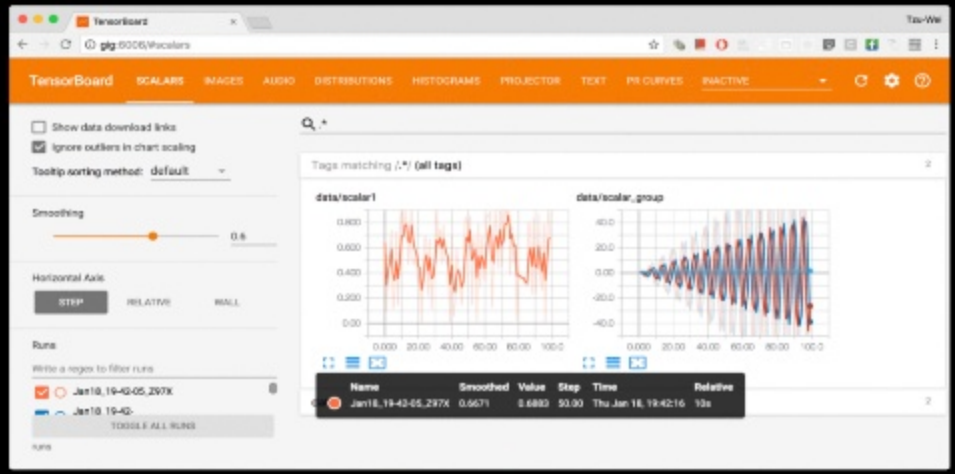
CPU CPU GPU ... GPU

Why PyTorch?

PyTorch

- Imperative Experience
  - Rapid Prototyping for Research
  - Easy Debugging & TBX
- Quick Ramp-up time
- Decent Docs & Community
  - 275K Downloads
  - 1900 Community Repos
  - 13+K Blogs Posts
- *Pythonic*!





databricks

PyTorch Key API Concepts

PyTorch

- Variables & Autograd
- Torch Tensors
- Operations

```
from torch import Variable
x = Variable(torch.Tensor([2]),requires_grad=True)
y = 5*x**4 + 3*x**3 + 7*x**2 + 9*x - 5
y.backward() #compute gradient and backpropagate
x.grad
```

```
#different kinds of Torch Tensors
x = torch.rand(5, 3);
y = torch.rand(5, 3)
t = torch.tensor([5.5, 3])
n = torch.tensor(np.array([[1, 2, 3], [4, 5, 6]]))
```

```
#operations or element-wise operations
a = (x + y)
m = (x * y)
if torch.cuda.is_available():
      x = x.cuda()
      y = y.cuda()
torch.add(x, y, out=result)
print(a, m, out)
```

databricks

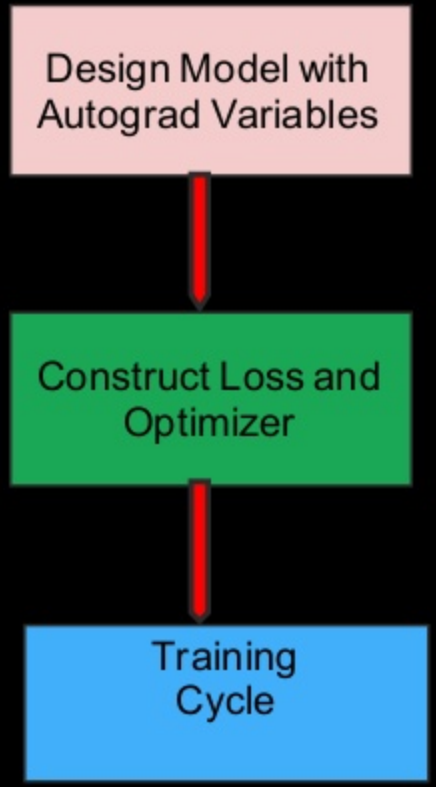
PyTorch Rhythm ...

PyTorch

1. Design a model using PyTorch Autograd Variables

2. Construct a loss function and optimizer with PyTorch APIs

3. Train your model: forward, backward, and update steps



databricks

PyTorch Rhythm : Linear Regression

PyTorch

```
x_data = Variable(torch.Tensor([[1.0], [2.0], [3.0], [4.0]]))
y_data = Variable(torch.Tensor([[0.], [0.], [1.], [1.]]))

class Model(torch.nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.linear = torch.nn.Linear(1, 1)  # One in and one out

    def forward(self, x):
        y_pred = F.sigmoid(self.linear(x))
        return y_pred

# our model
model = Model()
```

```
criterion = torch.nn.BCELoss(size_average=True)
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
```

```
# Training loop
for epoch in range(1000):
    # Forward pass: Compute predicted y by passing x to the model
    y_pred = model(x_data)

    # Compute and print loss
    loss = criterion(y_pred, y_data)
    print(epoch, loss.data[0])

    # Zero gradients, perform a backward pass, and update the weights.
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

# After training
hour_var = Variable(torch.Tensor([[1.0]]))
print("predict 1 hour ", 1.0, model(hour_var).data[0][0] > 0.5)
hour_var = Variable(torch.Tensor([[7.0]]))
print("predict 7 hours", 7.0, model(hour_var).data[0][0] > 0.5)
```

Logistic regression

1 **Design your model using class**

2 **Construct loss and optimizer (select from PyTorch API)**

3 **Training cycle (forward, backward, update)**

https://www.youtube.com/watch?v=113b7O3mabY (Sung Kim)

databricks

Training Options

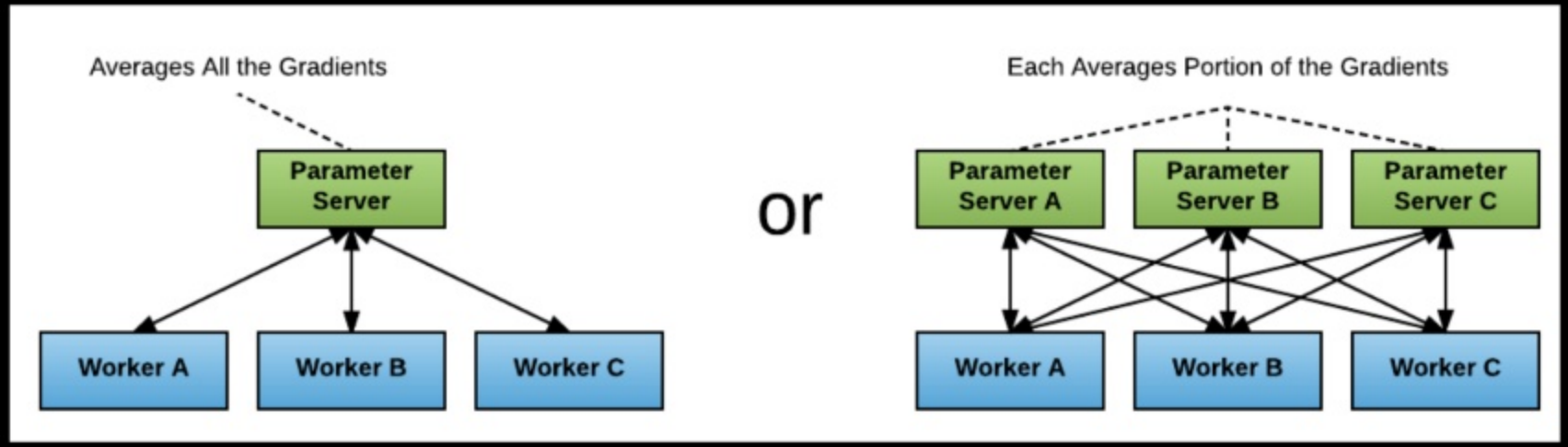
Options

1) Train on single node
2) Train on single node, distributed inference
3) Distributed training

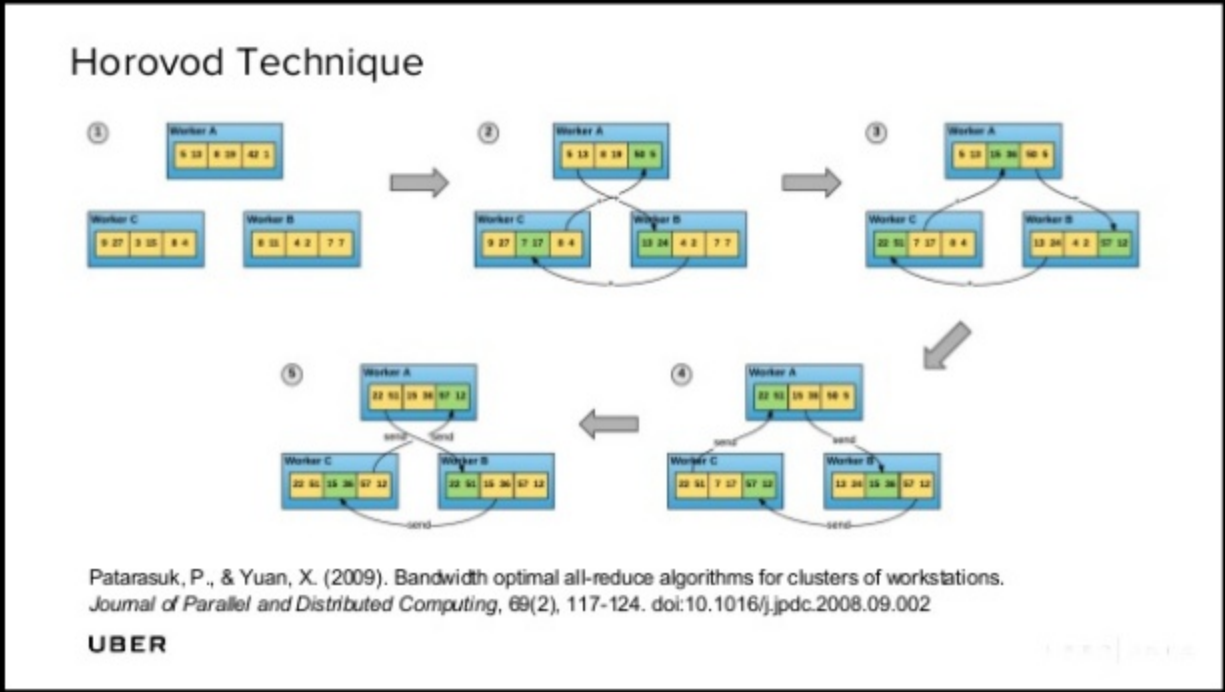
Horovod

- Created by Alexander Sergeev of Uber, open-sourced in 2017
- Simplifies distributed neural network training
- Supports TensorFlow, Keras, and PyTorch

Classical Parameter Server



All-Reduce

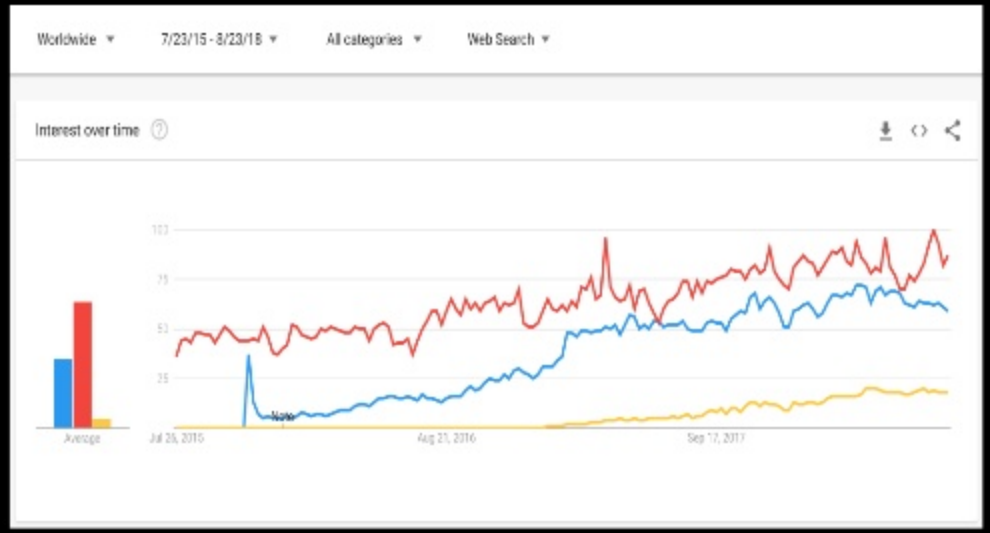


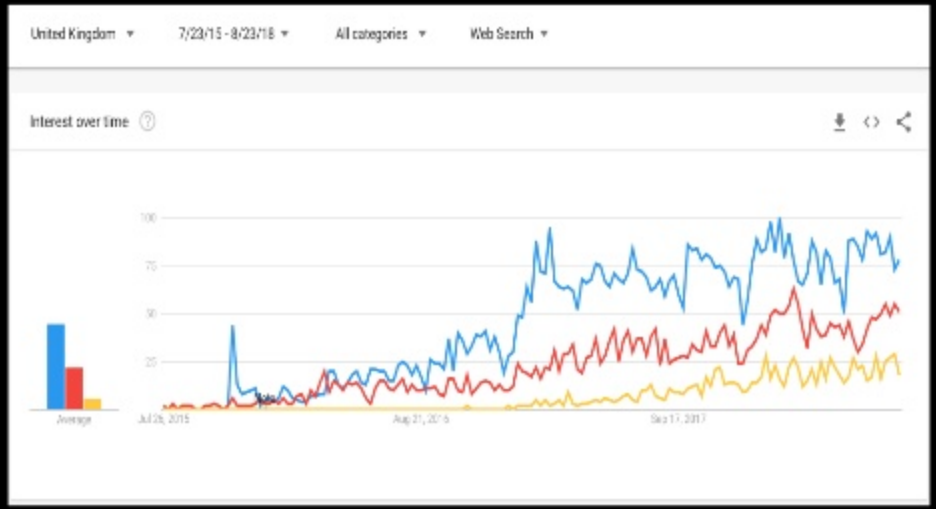
Minimal Code Change

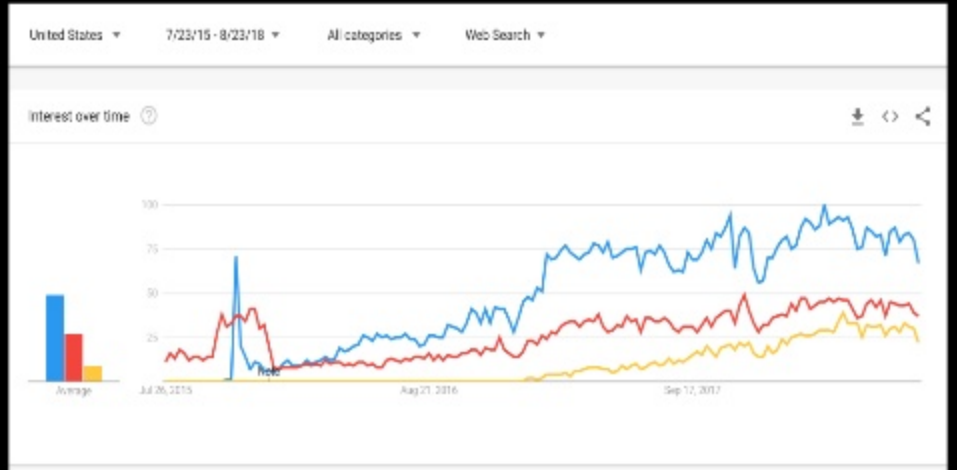
```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

# Horovod: adjust learning rate based on number of GPUs.
opt = keras.optimizers.Adadelta(1.0 * hvd.size())

# Horovod: add Horovod Distributed Optimizer.
opt = hvd.DistributedOptimizer(opt)

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=opt,
              metrics=['accuracy'])
```

TensorFlow, Keras, or PyTorch?

Takeaways: Gaining Momentum...







Keras
TensorFlow
PyTorch

Takeaways: When to Use TF, Keras or PyTorch

- Low-level APIs & Control
- Model Serving
- Supports multiple languages

TensorFlow

- High-level APIs
- Multiple Backends
- Love Python
- Rapid Experimentation

Keras

- *Pythonic!*
- Imperative Programming
- Rapid Experimentation

PyTorch

Databricks Runtime for Machine Learning

Ready to use clusters with built-in ML Frameworks

including TensorFlow, Keras, Horovod, and more

Horovod Estimator

for simplified distributed training on TensorFlow with Horovod using Apache Spark on Databricks

GPU support

on AWS (P2/P3) and Azure (NC/NC-v3) instances now supported!

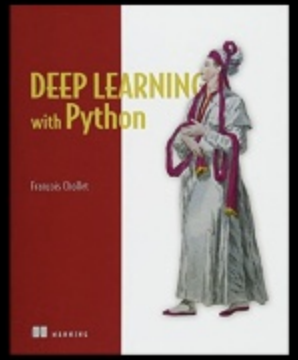
Resources & Books

Blog posts Talk, & webinars (http://databricks.com/blog)

- GPU acceleration in Databricks
- Deep Learning and Apache Spark
- fast.ai
- TensorFlow Tutorials
- TensorFlow Dev Summit
- Keras/TensorFlow Tutorials
- PyTorch Docs & Tutorials
    - Talk-1 from Soumith Chintala
    - Talk-2 from Soumith Chintala
- MLflow.org

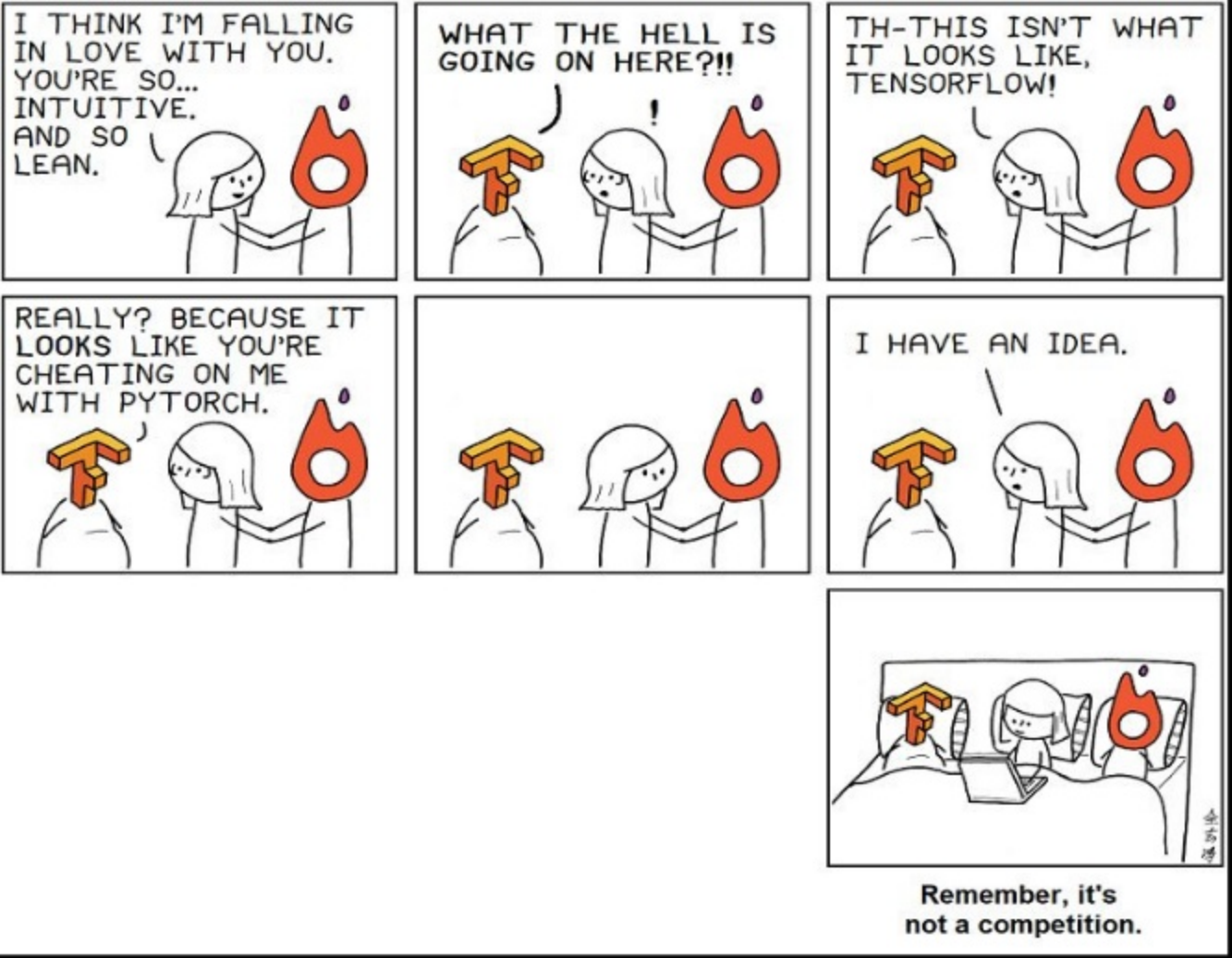
Docs for Deep Learning on Databricks (http://docs.databricks.com)

- Databricks Runtime ML
- HorovodEstimator



fast.ai

Making neural nets uncool again



Thank You! Questions?

brooke@databricks.com
jules@databricks.com (@2twitme)

databricks