Intro to PyTorch

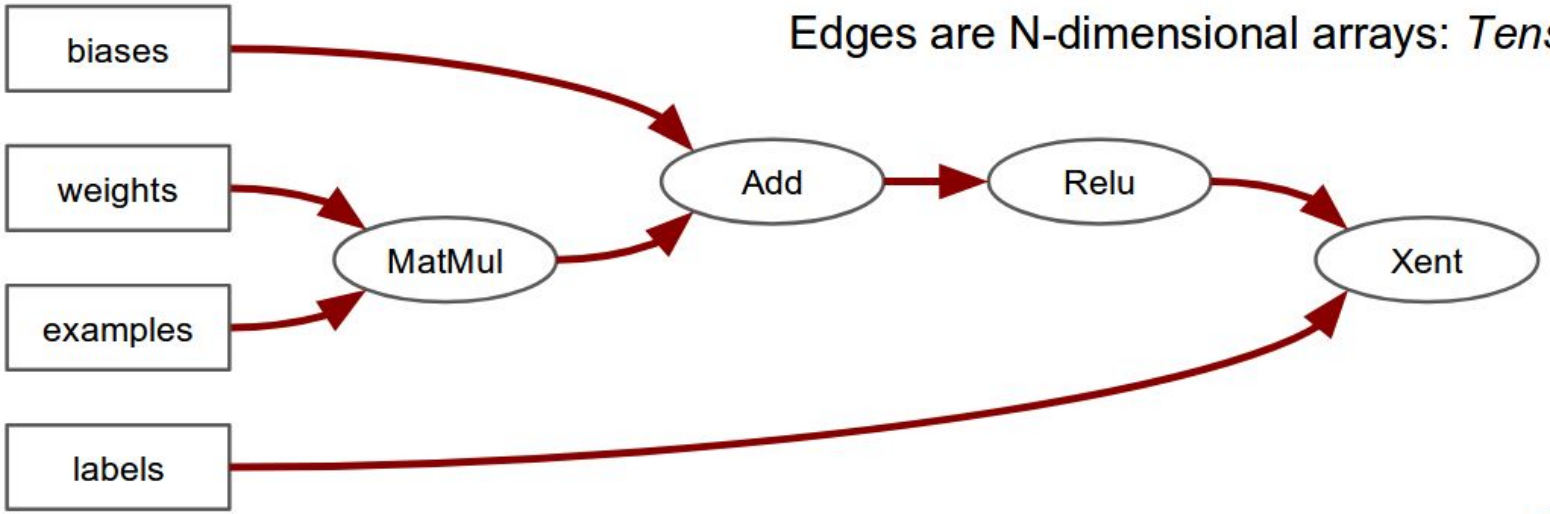
Erik Reppel

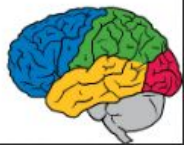
A brief history of DL Frameworks

High(er) level	Both!	Low(er) level	???
- Keras	- PyTorch	- Theano	- Nnabla (Sony)
- Lasagne	- TensorFlow	- Caffe	- CNTK (MS)
- Neupy	- MxNet		
- Blocks	- Deeplearning4j?		
	- Torch		
	- Caffe2		
	- DyNet		
	- Neon		

Computation is a dataflow graph

with tensors





From a talk by Jeff Dean

PyTorch vs TensorFlow

- Facebook w/ Nvidia, CMU, ...
- Full re-write of Torch (lua)
- Dynamic graph construction
- Targets: researchers

- Google Brain
- Inspired by DistBelief, Theano
- Targets: engineers

Performance: Very similar, PyTorch typically wins on RNNs and ResNets, overall not the biggest concern as both use CUDNN

What I don't like about TensorFlow

- Fractured eco-system

(Keras, contrib, tflearn, layers, Sonnet, slim, tensorpack, ...)

- Hard to debug

- Hard to "play around with"

What I like about TensorFlow

- Scalable to multi-gpu, multi-cluster
- Good story around productionizing models [link]
- Massive community
- Tons of tutorials

What I don't like about PyTorch

- Young (still in beta v0.2.0)
- Story about deploying models is weak (fixed now!) ONNX
- Ceremony around training/testing not built in

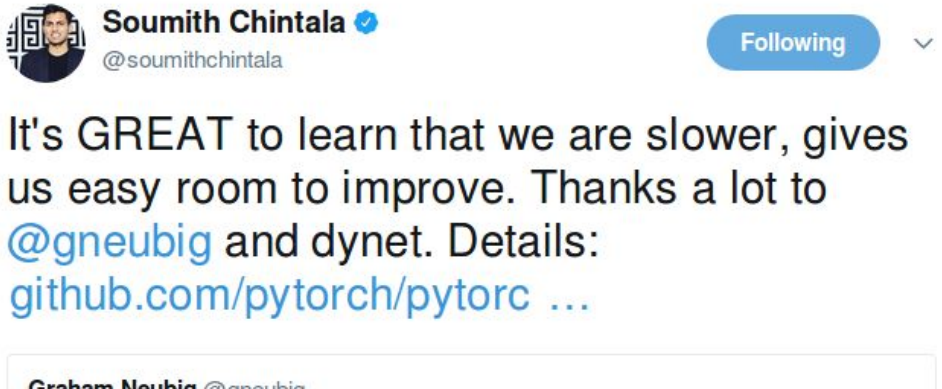
What I like about PyTorch

- Dynamic graph means any debugger works (PDB, etc)

- Networks and layers are modular (!!)

- Listening to feedback and actively developing

- Tons of control
- Multi-gpu, distributed
- More fun!



```python
import torch
from torch.autograd import Variable
import numpy as np

def rmse(y, y_hat):
    """Compute root mean squared error"""
    return torch.sqrt(torch.mean((y - y_hat).pow(2).sum()))

def forward(x, e):
    """Forward pass for our fuction"""
    return x.pow(e.repeat(x.size(0)))

# Let's define some settings
n = 100 # number of examples
learning_rate = 5e-6

# Model definition
x = Variable(torch.rand(n) * 10, requires_grad=False)

# Model parameter and it's true value
exp = Variable(torch.FloatTensor([2.0]), requires_grad=False)
exp_hat = Variable(torch.FloatTensor([4]), requires_grad=True)
y = forward(x, exp)

loss_history = []
exp_history = []

# Training loop
for i in range(0, 200):
    print("Iteration %d" % i)

    # Compute current estimate
    y_hat = forward(x, exp_hat)

    # Calculate loss function
    loss = rmse(y, y_hat)

    # Do some recordings for plots
    loss_history.append(loss.data[0])
    exp_history.append(y_hat.data[0])

    # Compute gradients
    loss.backward()

    print("loss = %s" % loss.data[0])
    print("exp = %s" % exp_hat.data[0])

    # Update model parameters
    exp_hat.data -= learning_rate * exp_hat.grad.data
    exp_hat.grad.data.zero_()
```

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

def rmse(y, y_hat):
    """Compute root mean squared error"""
    return tf.sqrt(tf.reduce_mean(tf.square((y - y_hat))))

def forward(x, e):
    """Forward pass for our fuction"""
    # tensorflow has automatic broadcasting
    # so we do not need to reshape e manually
    return tf.pow(x, e)

n = 100 # number of examples
learning_rate = 5e-6

# Placeholders for data
x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)

# Model parameters
exp = tf.constant(2.0)
exp_hat = tf.Variable(4.0, name='exp_hat')

# Model definition
y_hat = forward(x, exp_hat)

# Optimizer
loss = rmse(y, y_hat)
opt = tf.train.GradientDescentOptimizer(learning_rate)

# Summaries (NEW)
loss_summary = tf.summary.scalar("loss", loss)
exp_summary = tf.summary.scalar("exp", exp_hat)
all_summaries = tf.summary.merge_all()

# We will run this operation to perform a single training step,
# e.g. opt.step() in Pytorch.
# Execution of this operation will also update model parameters
train_op = opt.minimize(loss)

# Let's generate some training data
x_train = np.random.rand(n) + 10
y_train = x_train ** 2

loss_history = []
exp_history = []

# First, we need to create a Tensorflow session object
with tf.Session() as sess:

    # Initialize all defined variables
    tf.global_variables_initializer().run()

    summary_writer = tf.summary.FileWriter('./tensorboard', sess.graph)

    # Training loop
    for i in range(0, 500):
        print("Iteration %d" % i)
        # Run a single trainig step
        summaries, curr_loss, curr_exp, _ = sess.run([all_summaries, loss, exp_hat, train_op], fe
ed_dict={x: x_train, y: y_train})

        print("loss = %s" % curr_loss)
        print("exp = %s" % curr_exp)

        # Do some recordings for plots
        loss_history.append(curr_loss)
        exp_history.append(curr_exp)

        summary_writer.add_summary(summaries, i)
```

Lets see some code!

Content

-
- 1_Derivatives_optimizers
- 2_creating_models
- 3_handling_data
- 4_using_gpus
- 5_all_together
- 6_reducing_boilerplate