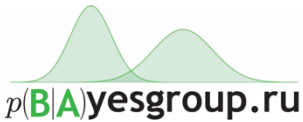
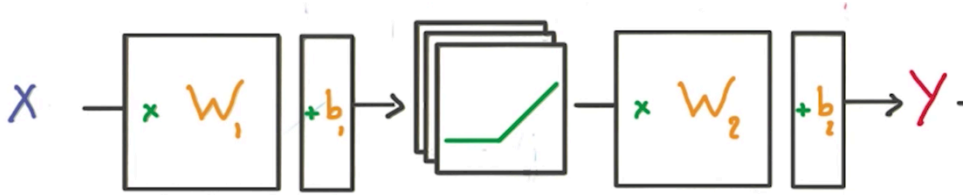
Deep Learning Software

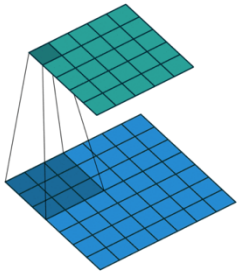
Arsenii Ashukha

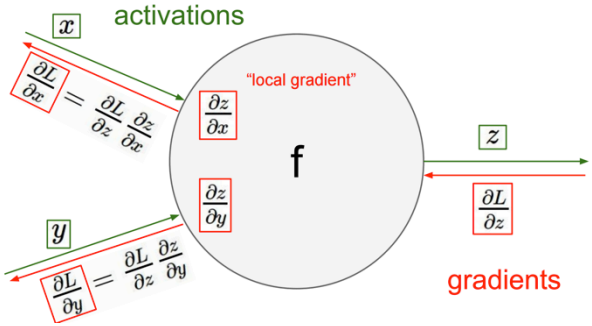


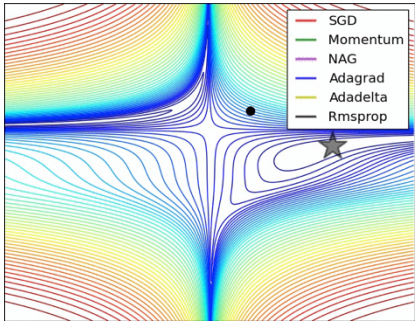
Aug 17th, 2017

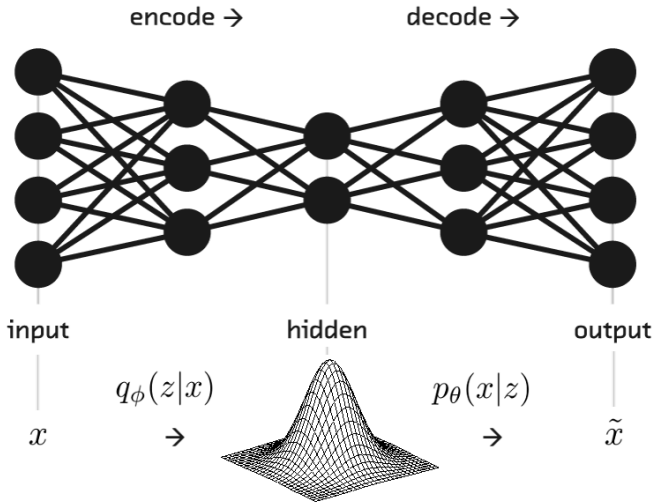
Earlier


X
W1
b1
W2
b2
Y




activations
"local gradient"
f
gradients


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop


encode →
decode →
input
hidden
output
$q_\phi(z|x)$
$p_\theta(x|z)$
x
$\tilde{x}$

The point of deep learning frameworks

- Easily build big computational graphs
- Easily compute gradients in computational graphs
- Run it all efficiently on GPU (wrap cuDNN, cuBLAS, etc)

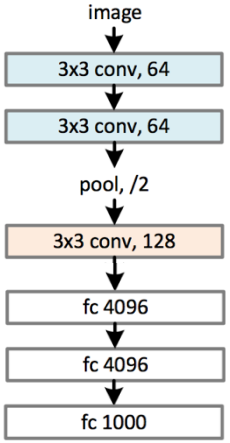
All frameworks do the same thing

but in different ways!

Standard Pipeline to Train your DNN

Before training:

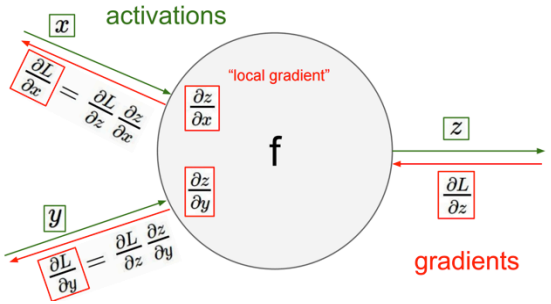
- Get your training/validation data
- Define the model

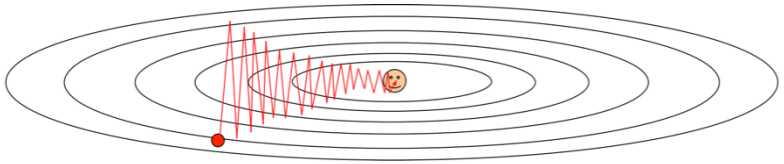


A lot of times:

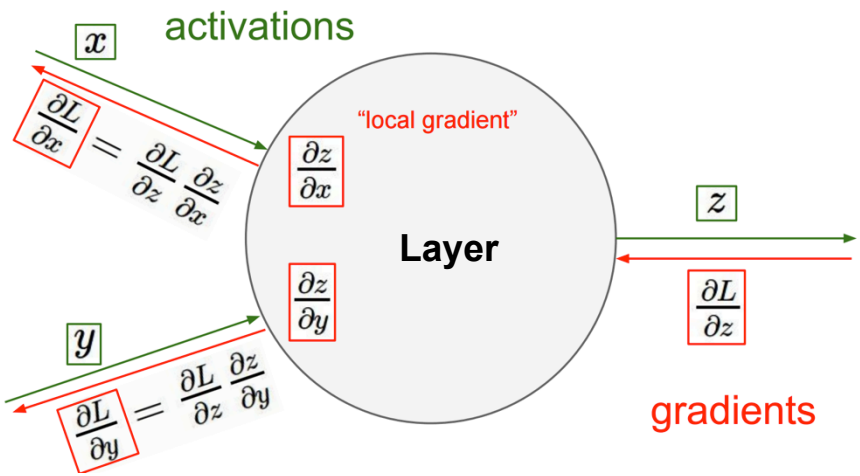
- Sample a batch (subset) of images/labels
- Evaluate the Gradient of the Loss function
- Update the parameters using the Gradient

$$\frac{\partial L}{\partial w_i} \qquad L = \underbrace{\frac{1}{N}\sum_i L_i}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularization loss}}$$





What is the simplest way to create a DL Library?



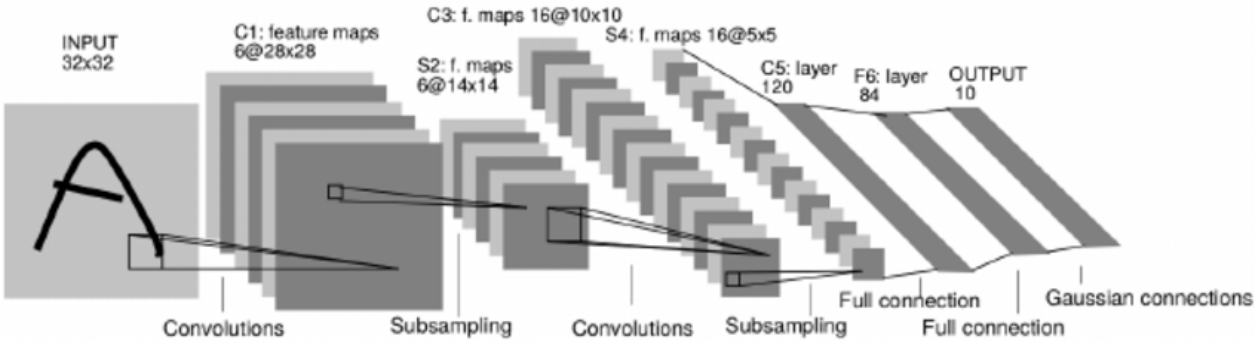

```
void Layer<Dtype>::Forward_gpu(
const vector<Blob<Dtype>*>& bottom,
const vector<Blob<Dtype>*>& top) {
        ...
        % Compute Layer Output efficiently
        % on your own with C++ and CUDA
}
```

```
void Layer<Dtype>::Backward_gpu(
const vector<Blob<Dtype>*>& top,
const vector<bool>& propagate_down,
const vector<Blob<Dtype>*>& bottom) {
        ...
        % Compute Layer Gradients
}
```

Caffe

- \- You have to write C++ code if you want to define a new layer
- \+ Standard models with no coding at all
- \+ A lot of computer vision things are made with Caffe

Caffe Example

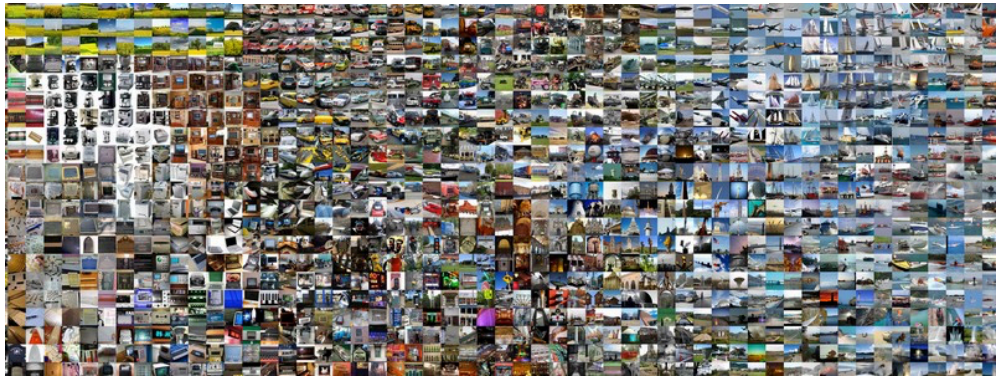



```
name: "LeNet"
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {lr_mult: 1}
  param {lr_mult: 2}
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }}}
....
```

130 lines

Caffe Example

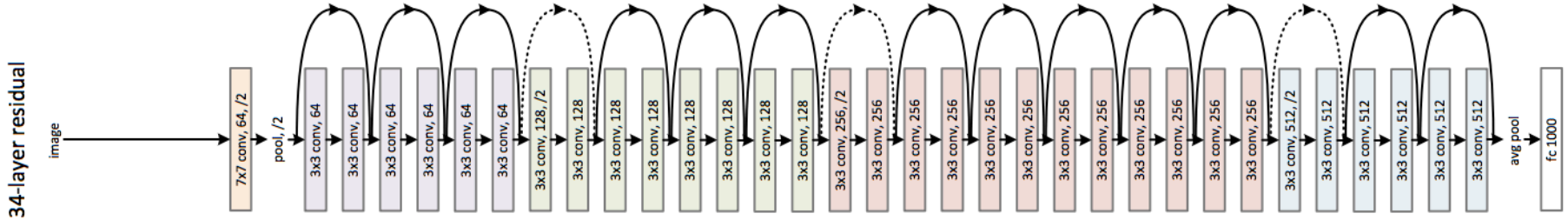
IMAGENET



6776 lines (6104 sloc) | 95.7 KB

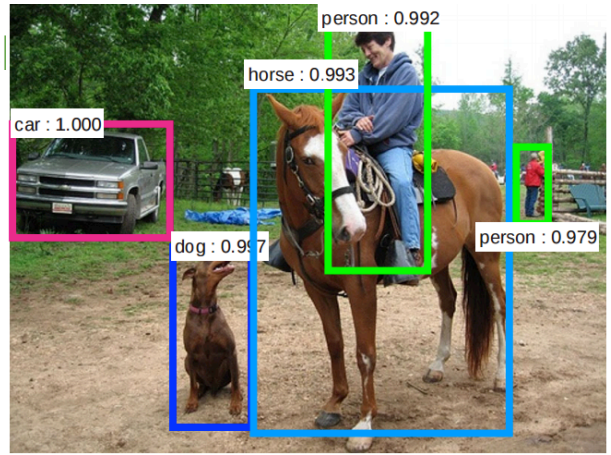
```
1   name: "ResNet-152"
2   input: "data"
3   input_dim: 1
4   input_dim: 3
5   input_dim: 224
6   input_dim: 224
7
8   layer {
9           bottom: "data"
10          top: "conv1"
```

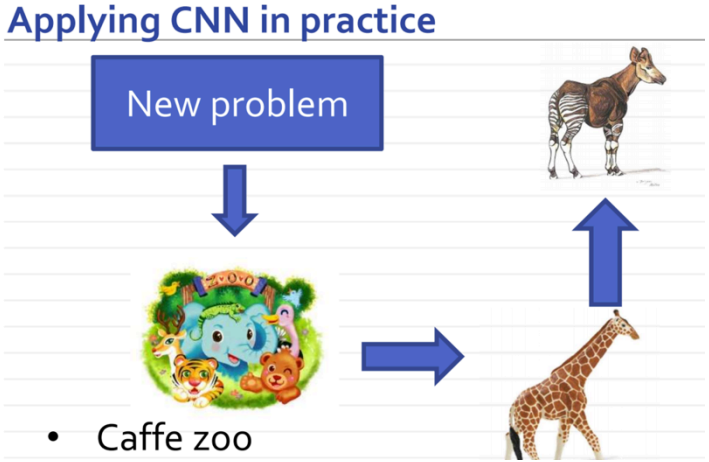
6776 lines



Caffe - Protobuf language

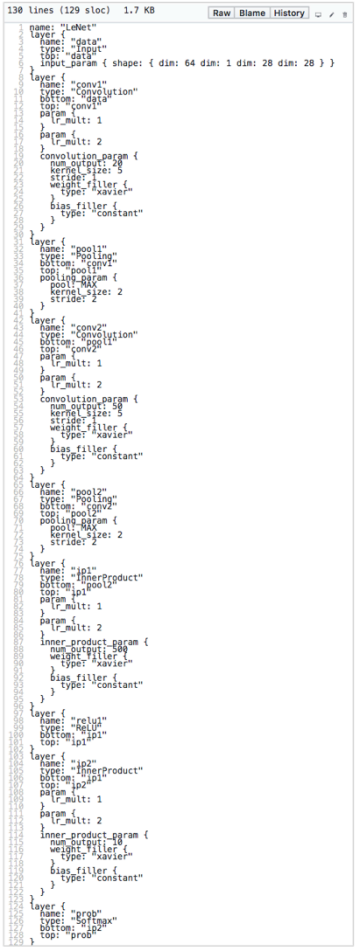
- Hard to create your own layers (C++, CUDA, Recompile, ...)
- Hard to debug
- Easy to deploy, you can convert caffe models to pure C++
- De facto standard in Computer Vision tas
- Ric





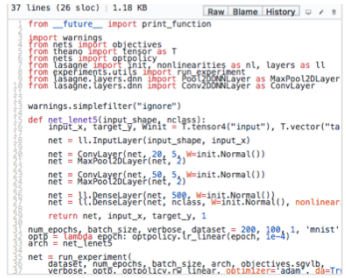
https://github.com/BVLC/caffe/wiki/Model-Zoo

Lenet Caffe vs Theano



Caffe

[130 lines]

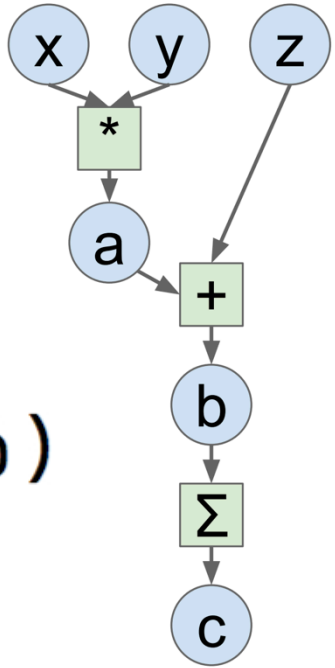


Theano

[37 lines]

Static Computational Graph

```
a = x * y
b = a + z
c = np.sum(b)
```



+ Easy to build nets and eval gradients
+ We can optimize the Graph
- Graph is static during training
- Need time to compile/optimize
- Hard to debug

theano **and** TensorFlow™

Wrappers

From basic operation **sum, mul, log** to **layers**!

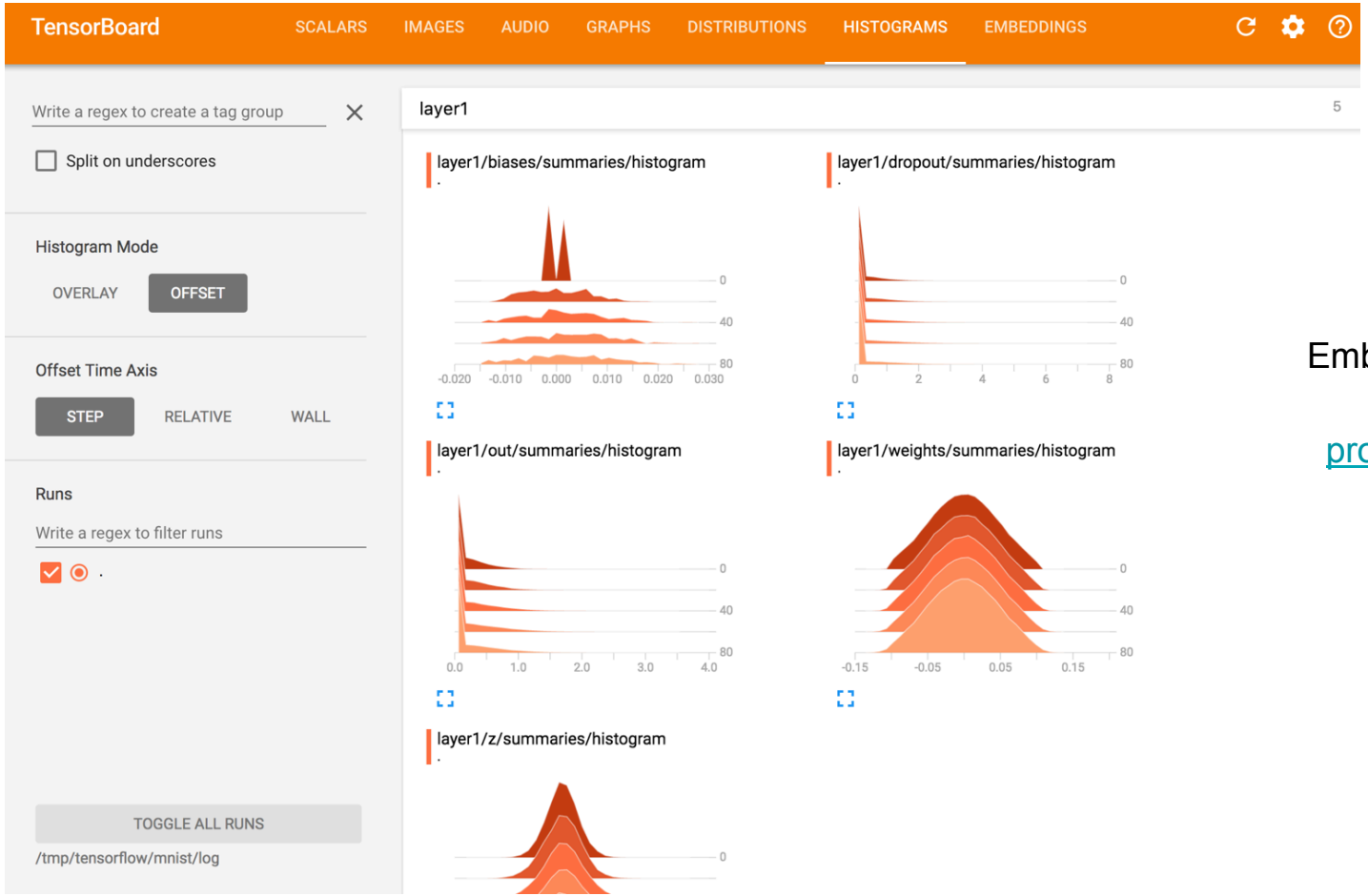
Lasagne

Keras

TensorFlow-Slim

TF-Slim is a lightweight library for defining, training and evaluating complex models in TensorFlow. Components of tf-slim can be freely mixed with native tensorflow, as well as other frameworks, such as tf.contrib.learn.

Tensor Board



Embeddings Visualization

projector.tensorflow.org

PyTorch

Andrej Karpathy
@karpathy

Following

I've been using PyTorch a few months now and I've never felt better. I have more energy. My skin is clearer. My eye sight has improved.

Dynamic Computational Graph

PYTORCH

A graph is created on the fly


```
from torch.autograd import Variable

x = Variable(torch.randn(1, 10))
prev_h = Variable(torch.randn(1, 20))
W_h = Variable(torch.randn(20, 20))
W_x = Variable(torch.randn(20, 10))
```

- \- Dynamic graph is hard to optimize
- \- Very young and still a bit raw
- \+ Step-by-step debugging
- \+ Changing architecture on the fly

How did we get over the head

We were happy with theano
Until we started working on ImageNet

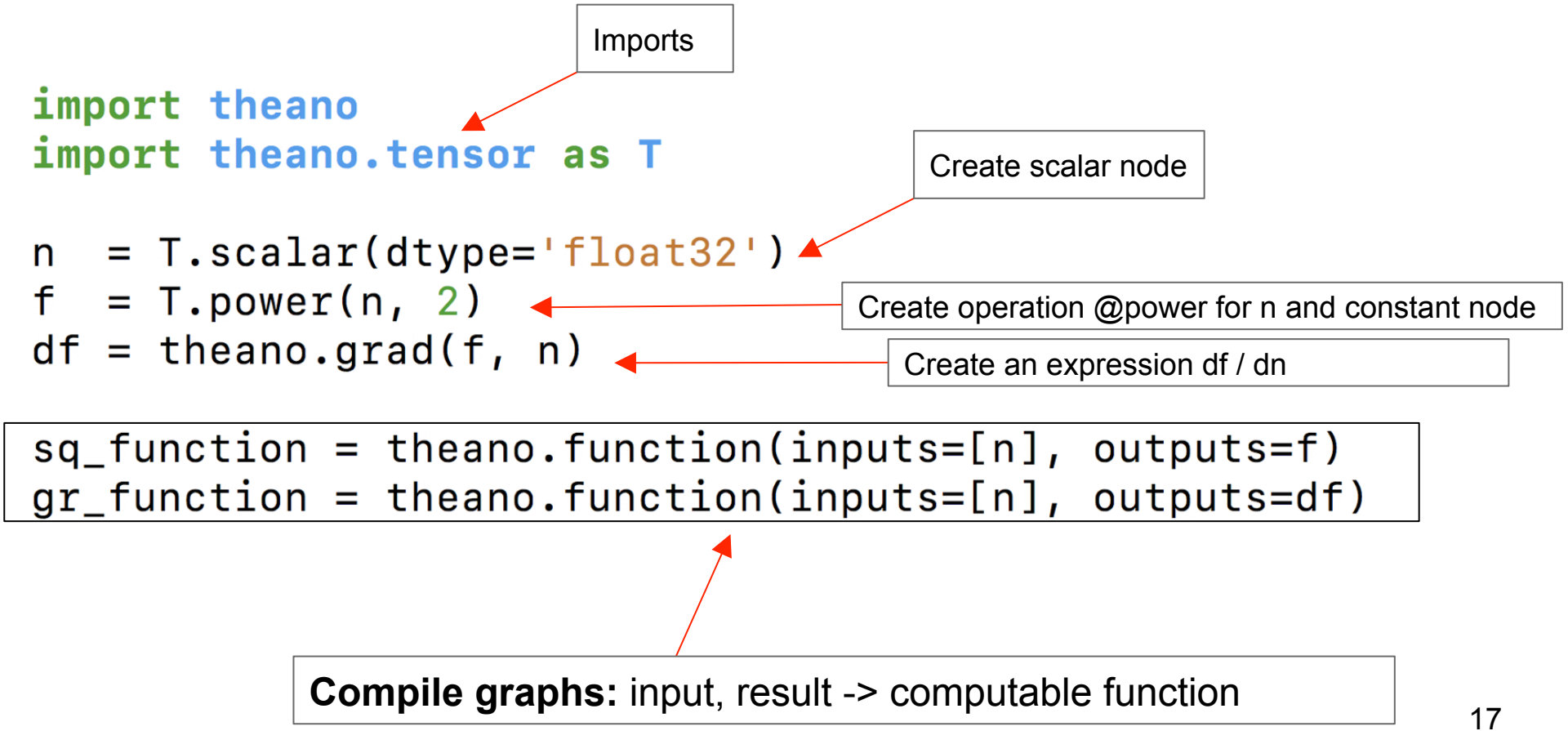
We were happy with pytorch
Until we started working with Numerical Unusable code

Still trying to work with TF

Comparison

	Caffe	theano	TensorFlow™	PYTORCH
Paradigm	Declarative	Declarative	Declarative	Imperative
Language	Protobuf, C++	Python, C	Python, C++	Python, C++
Deployment	Easy to deploy		Easy to deploy	
Model size		Good for small models	Good for large models	Good for large models
CV inside	Good for CV		Good for CV	
Stability		Yes	Not yet	Not yet

Theano


Imports
import theano
import theano.tensor as T
Create scalar node
n = T.scalar(dtype='float32')
f = T.power(n, 2)
Create operation @power for n and constant node
df = theano.grad(f, n)
Create an expression df / dn
sq_function = theano.function(inputs=[n], outputs=f)
gr_function = theano.function(inputs=[n], outputs=df)
Compile graphs: input, result -> computable function

Log Reg Theano

```
import theano
import numpy as np
import theano.tensor as T
from sklearn.datasets import load_digits
```

```
Xtr, Ytr = load_digits(2, return_X_y=True)
Ytr = 2 * (Ytr - 0.5) # [0, 1] -> [-1, 1]
```

```
X = T.matrix()
y = T.vector()
W = theano.shared(np.zeros(64))
```

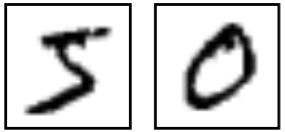
```
logit = X.dot(W)
ppred = 1.0 / (1.0 + np.exp(logit))
```

```
datL = T.log(1 + T.exp(y * ppred)).sum()
regL = T.power(W, 2).sum()

L = datL + 1e-4 * regL
```

```
dW = T.grad(L, W)
up = {W: W - 1e-3*dW}
```

```
train_func = theano.function([X, y], L, updates=up)
predi_func = theano.function([X], ppred)
```



$$f(x_i, W, b) = Wx_i + b$$

$$L = \underbrace{\frac{1}{N}\sum_i L_i}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularization loss}}$$

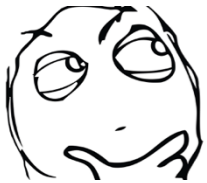
$$\nabla_W L$$

Log Reg Theano

```
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(Xtr, Ytr)

for i in range(31):
    loss_i = train_func(X_train,y_train)
    if i % 5 == 0:
        print ' loss at iter %i: %.4f' % (i, loss_i),
        print ' tr auc: %.2f' % roc_auc_score(y_train, predi_func(X_train)),
        print ' te auc: %.2f' % roc_auc_score(y_test, predi_func(X_test))
```



What goes wrong?

```
loss at iter 0: 193.0345  tr auc: 0.10  te auc: 0.10
loss at iter 5: 135.5271  tr auc: 0.00  te auc: 0.00
loss at iter 10: 135.3691  tr auc: 0.00  te auc: 0.00
loss at iter 15: 135.2518  tr auc: 0.00  te auc: 0.00
loss at iter 20: 135.2101  tr auc: 0.00  te auc: 0.00
loss at iter 25: 135.1875  tr auc: 0.00  te auc: 0.00
loss at iter 30: 135.1733  tr auc: 0.00  te auc: 0.00
```

Lasagne

theano

```
W = T.matrix()
b = T.vector()
T.max(0, T.dot(W, X) + b))
```

Lasagne

latest

```
lasagne.DenseLayer(X)
```

$$
\begin{aligned}
i_t &= \sigma_i(x_t W_{xi} + h_{t-1} W_{hi} + w_{ci} \odot c_{t-1} + b_i) \\
f_t &= \sigma_f(x_t W_{xf} + h_{t-1} W_{hf} + w_{cf} \odot c_{t-1} + b_f) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \sigma_c(x_t W_{xc} + h_{t-1} W_{hc} + b_c) \\
o_t &= \sigma_o(x_t W_{xo} + h_{t-1} W_{ho} + w_{co} \odot c_t + b_o) \\
h_t &= o_t \odot \sigma_h(c_t)
\end{aligned}
$$

```
lasagne.layers

lasagne.updates

lasagne.init

lasagne.nonlinearities

lasagne.objectives

lasagne.regularization

lasagne.random

lasagne.utils
```

Lasagne: Simplest Network

```
from mnist import load_dataset
X_train, y_train, X_val, y_val, X_test, y_test = load_dataset()
```

← Read Dataset

```
input_X, target_y = T.tensor4('X'), T.vector('y', dtype='int32')
```

← Create ***X, y*** variables

```
inputl = ll.InputLayer((None, 1, 28, 28), input_X)

net = ll.Conv2DLayer(inputl, 20, 5)
net = ll.Pool2DLayer(net, 2)
net = ll.DenseLayer(net,  100)

outputl = ll.DenseLayer(net, num_units=10, nonlinearity=nl.softmax)
```

← Define your DNN

```
predicted = ll.get_output(outputl)
weights = ll.get_all_params(outputl)
```

```
loss = lo.categorical_crossentropy(predicted, target_y).mean()
opt = lasagne.updates.rmsprop(loss, weights, learning_rate=0.01)
```

← Define loss and opt

```
train_fun = theano.function([input_X, target_y], [loss], updates=opt)
```

← Compile the Graph

```
num_epochs, batch_size = 10, 50
for epoch in range(num_epochs):
    for inputs, targets in iterate_minibatches(X_train, y_train, batch_size):
        train_err_batch = train_fun(inputs, targets)
        print '%.2f' % train_err_batch[0]
```

Iterate over batches

Theano Install and Run

```
pip install numpy
pip install --upgrade https://github.com/Theano/Theano/archive/rel-0.9.0.zip
pip install --upgrade https://github.com/Lasagne/Lasagne/archive/master.zip
```

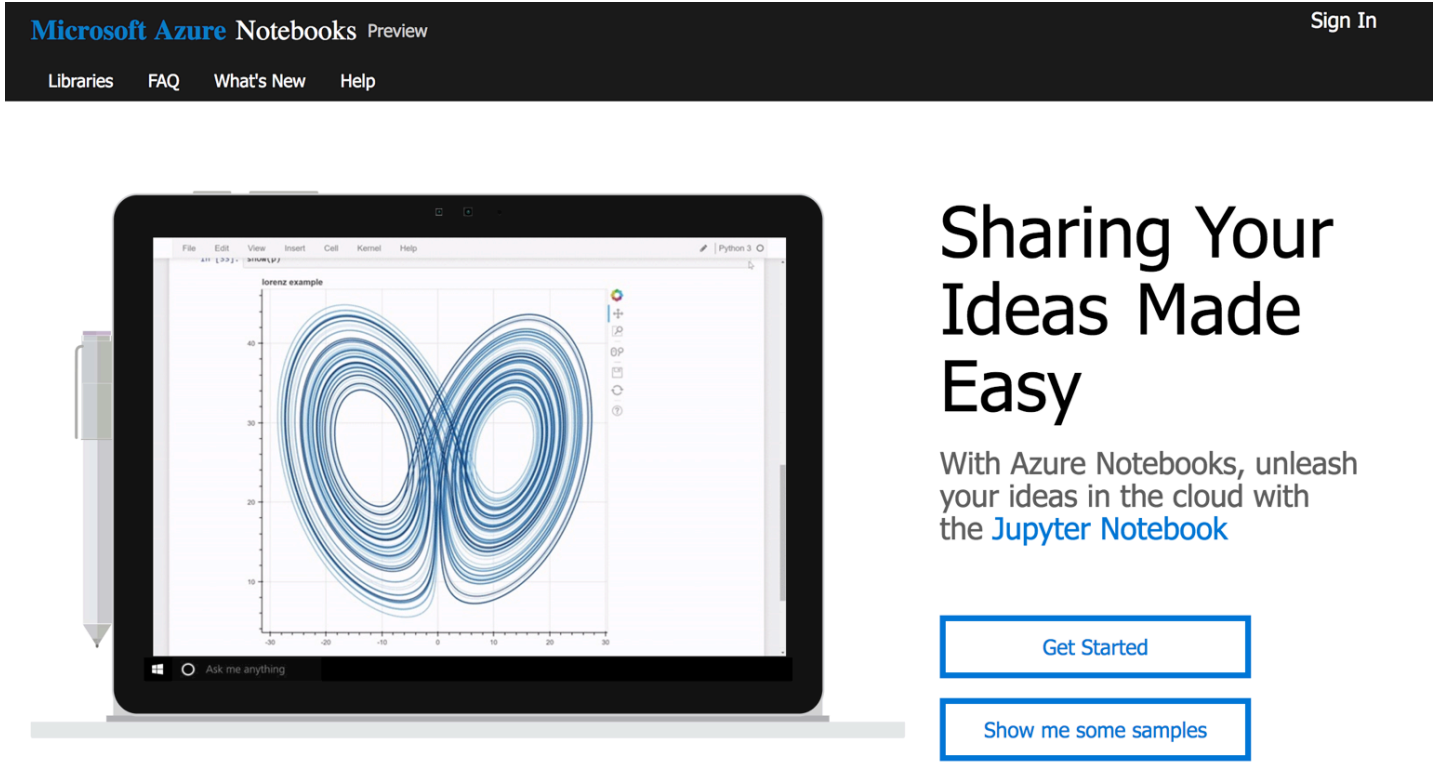
```
export THEANO_FLAGS='floatX=float32,device=gpu3,lib.cnmem=0.5'
ipython2 notebook
```

```
import theano
import lasagne
import theano.tensor as T
from lasagne.layers import *
```

```
WARNING (theano.sandbox.cuda): The cuda backend is deprecated and will be removed in the next release (v0.10).  Pleas
e switch to the gpuarray backend. You can get more information about how to switch at this URL:
 https://github.com/Theano/Theano/wiki/Converting-to-the-new-gpu-back-end%28gpuarray%29

Using gpu device 3: Tesla K40m (CNMeM is enabled with initial size: 80.0% of memory, cuDNN 5105)
```

Azure Notebooks



https://notebooks.azure.com

Summary

- All frameworks do the same thing but in different ways!
- Good idea is to choose framework for your requirements
  - Caffe seems good only if you are in Computer Vision
  - TF or PyTorch are good default choice, but they still have a couple of troubles (optimization, rawness, ...)
  - Theano is good only for small nets and easy prototy| a power graph optimiza