Frameworks in Python for Numeric Computation / ML

```python
import numpy as np
import scipy
import tensorflow
import torch
import torchvision
import torch.backends.cudnn
from PIL import Image
import json
import random
import argparse
import re
import os
```

Why use a framework?

- Why not use the built-in data structures? Why not write our own matrix multiplication function?

```python
def dot(mat1, mat2):
    num_rows = len(mat1)
    assert num_rows > 0

    sum_length = len(mat2)
    assert sum_length > 0

    num_cols = len(mat2[0])
    assert num_cols > 0

    result = []
    for row in range(num_rows):
        new_row = []
        for col in range(num_cols):
            cell_value = 0.0
            for sum_idx in range(sum_length):
                cell_value += mat1[row][sum_idx] * mat2[sum_idx][col]
            new_row.append(cell_value)
        result.append(new_row)

    return result
```

Why use a framework?

- Turns out pure Python is not well suited for high performance numeric computation.

```
In [34]: %timeit recitation2.dot(mat1_python, mat2_python)
177 ms ± 334 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)

In [35]: %timeit np.dot(mat1_numpy, mat2_numpy)
74 µs ± 8.19 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

(Numpy’s matrix multiply function was 2392x faster than ours)

- Frameworks are needed not only because they contain useful library functions, but also because these functions are much more computationally efficient.
- “Vectorizing your code” ≈ “Using library functions instead of loops”

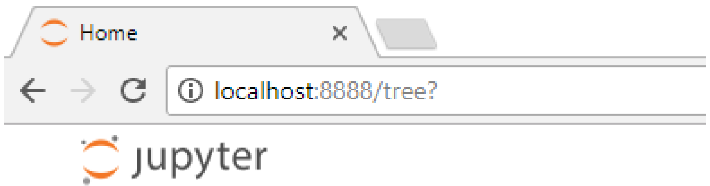
Numpy

- Lets you manipulate and perform computations on arrays of numbers
- Example array operations:
  - "Create an array filled with zeros"
  - "Create an array that contains every second element of another array"
  - "Add one to everything in this array"
  - "Find all locations in the array where the value is negative"
  - … etc.

Numpy and Pytorch Jupyter Notebooks

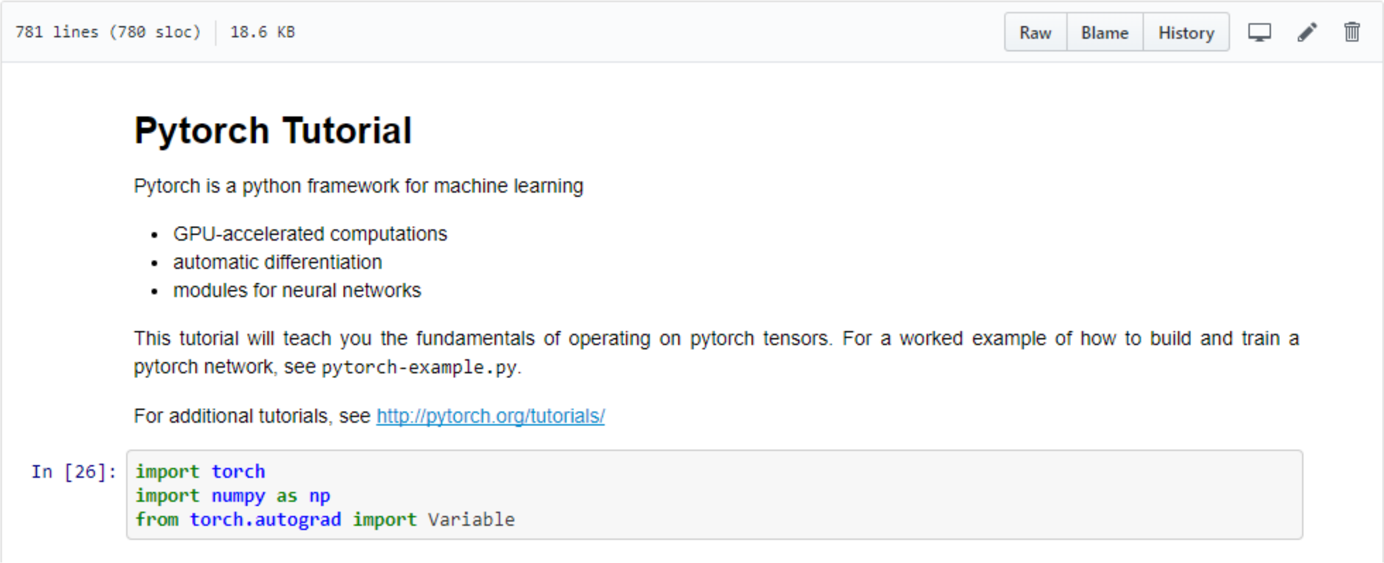
- SSH to your server
- Clone code: `git clone [https://github.com/cmudeeplearning11785/deep-learning-tutorials.git](https://github.com/cmudeeplearning11785/deep-learning-tutorials.git)`
- Activate Pytorch: `source activate pytorch_p36`
- Start Jupyter: `jupyter notebook`
- Browse to Jupyter notebook

```
ben@unity:/mnt/data/projects/deep-learning-11785$ jupyter notebook
[I 00:51:56.397 NotebookApp] Serving notebooks from local directory: /mnt/data/projects/deep-learning-11785
[I 00:51:56.397 NotebookApp] 0 active kernels
[I 00:51:56.397 NotebookApp] The Jupyter Notebook is running at:
[I 00:51:56.397 NotebookApp] http://localhost:8888/?token=d40d35ea5245285f6d179fd26c002ab497ced3a66ee93916
[I 00:51:56.398 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[W 00:51:56.398 NotebookApp] No web browser found: could not locate runnable browser.
[C 00:51:56.398 NotebookApp]

    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://localhost:8888/?token=d40d35ea5245285f6d179fd26c002ab497ced3a66ee93916
```



Numpy and Pytorch Jupyter Notebooks

- If you cannot access AWS right now, view the notebook on Github
  - https://github.com/cmudeeplearning11785/deep-learning-tutorials
  - Go to folder ‘recitation-2’
  - You cannot run the code but you can follow along



781 lines (780 sloc) | 18.6 KB

Raw | Blame | History

Pytorch Tutorial

Pytorch is a python framework for machine learning

- GPU-accelerated computations
- automatic differentiation
- modules for neural networks

This tutorial will teach you the fundamentals of operating on pytorch tensors. For a worked example of how to build and train a pytorch network, see `pytorch-example.py`.

For additional tutorials, see http://pytorch.org/tutorials/

In [26]:

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

Numpy

- Try the Numpy tutorial in Jupyter Notebook

Numpy Tutorial

`numpy` is the core python package for CPU computations on arrays. It is the package for managing arrays that is used by other packages such as `pytorch`, `tensorflow` or `scipy`.

In [1]:
```
import numpy as np  # standard practice is to name package `np`
```

Creating Numpy Arrays

Create arrays from an n-d list using `np.array`

In [2]:
```
x = np.array([4,5,6,7])
print(x)
```

```
[4 5 6 7]
```

In [3]:
```
x = np.array([[3,4,2], [5,6,7]])
print(x)
```

```
[[3 4 2]
 [5 6 7]]
```

Generate random data using functions in `np.random`. It is best practice to seed numpy for consistent results.

In [4]:
```
np.random.seed(123)
print(np.random.random((9,))) # uniform random 0-1
print(np.random.randint(0,3,(10,))) # random integers [0,1,2]
```

```
[ 0.69646919  0.28613933  0.22685145  0.55131477  0.71946897  0.42310646
  0.9807642   0.68482974  0.4809319 ]
[1 0 2 0 1 2 1 0 0 0]
```

PyTorch

- The framework we will be using in this course to design deep neural networks
- The interface is similar to Numpy, though some function names will be different
  - Instead of np.array, you have torch.Tensor
  - Tensors can be moved to the GPU
  - torch.Variable can be used for automatic differentiation

Pytorch

- Brief look at the Pytorch tutorial

Pytorch Tutorial

Pytorch is a python framework for machine learning

- GPU-accelerated computations
- automatic differentiation
- modules for neural networks

This tutorial will teach you the fundamentals of operating on pytorch tensors. For a worked example of how to build and train a pytorch network, see `pytorch-example.py`.

In [1]:
```
import torch
import numpy as np
from torch.autograd import Variable
```

Tensors

Tensors are the fundamental object for array data. The most common types you will use are `IntTensor` and `FloatTensor`.

In [2]:
```
# Create uninitialized tensor
x = torch.FloatTensor(2,3)
print(x)
# Initialize to zeros
x.zero_()
print(x)
```

```
 5.9826e+24  4.5630e-41  8.8594e-37
 0.0000e+00  4.4842e-44  0.0000e+00
[torch.FloatTensor of size 2x3]
```

Pytorch Example Model

- Open `pytorch-example.py`
- This example:
  - Creates a model
  - Creates a dataset
  - Iteratively trains the model on the dataset

Simple Task

- Illustrative example: making an MLP to count the number of 1’s in an n-bit binary input.
  - To make things a bit more interesting, we’ll require the output to be a binary number instead of the usual one-hot representation.

For example:

X: [1,0,1,1,0,0,0] (input has 3 ones)

Y: [0,1,1] (output is binary encoding of ‘3’)

Create a module

- Create a class that extends torch.nn.Module
- Add components and specify how they will be used in the forward pass

```python
class OnesCounter(torch.nn.Module):
    def __init__(self, input_size):
        super().__init__()
        self.input_size = input_size

        count_bitwidth = int(np.ceil(np.log2(input_size + 1)))
        self.to_hidden1 = torch.nn.Linear(input_size, 2 * input_size)
        self.hidden_sigmoid1 = torch.nn.Sigmoid()
        self.to_hidden2 = torch.nn.Linear(2 * input_size, 2 * input_size)
        self.hidden_sigmoid2 = torch.nn.Sigmoid()
        self.to_binary = torch.nn.Linear(2 * input_size, count_bitwidth)

    def forward(self, input_val):
        hidden1 = self.hidden_sigmoid1(self.to_hidden1(input_val))
        hidden2 = self.hidden_sigmoid2(self.to_hidden2(hidden1))
        return self.to_binary(hidden2)
```

Load data

- Load the training data (or create it in our case)

```python
def load_data():
    # We'll just make our data on the spot here, but
    # we usually load real data sets from a file

    # Create 10000 random 7-bit inputs
    data = np.random.binomial(1, 0.5, size=(10000, 7))

    # Count the number of 1's in each input
    labels = data.sum(axis=1)

    # Create the binary encoding of the ground truth labels
    # As a bit of practice using Numpy, we're going to do this
    # without using a Python loop.
    labels_binary = np.unpackbits(labels.astype(np.uint8)).reshape((-1,8))
    labels_binary = labels_binary[:,-3:]

    return (data, labels_binary)
```

Training the network

- Neural networks in PyTorch take in `torch.autograd.Variable` objects (as opposed to `torch.Tensor` or `np.array` objects)
  - The Variable type allows Pytorch to automatically compute gradients
- So we actually have to convert our Numpy data to Tensors, and then to Variables.
- We also want to move all our data to the GPU if we have access to one.
- To make this less of a headache, it’s recommended that you create several helper functions to do this

The helper functions

```python
def to_tensor(numpy_array):
    # Numpy array -> Tensor
    return torch.from_numpy(numpy_array).float()


def to_variable(tensor):
    # Tensor -> Variable (on GPU if possible)
    if torch.cuda.is_available():
        # Tensor -> GPU Tensor
        tensor = tensor.cuda()
    return torch.autograd.Variable(tensor)
```

Training the network

- Choose the loss function and the optimizer
  - We’ll use binary cross entropy loss and the stochastic gradient descent optimizer
- Create a PyTorch data set from our Numpy data, and a data loader which will load minibatches from the data set.
- For each minibatch, we must:
  - Reset the gradients (a kind of boilerplate step)
  - Run a feed forward pass
  - Compute the losses
  - Run a backpropagation pass
  - Update the network
  - Record metrics, log progress, bookkeeping, etc.

The training routine

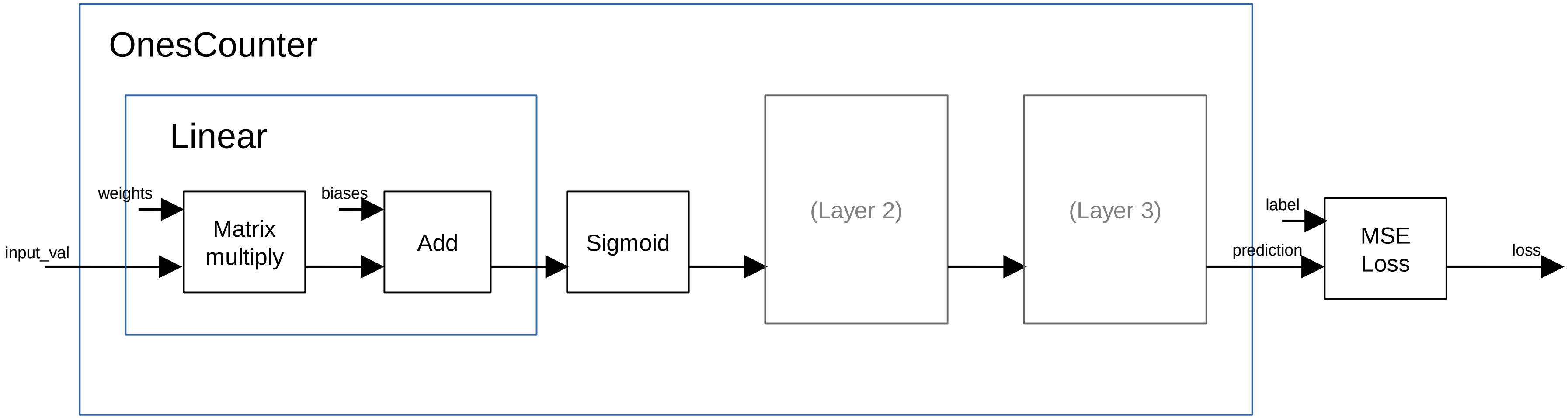
```
def training_routine(num_epochs, minibatch_size, learn_rate):
    (data, labels_binary) = load_data()

    my_net = OnesCounter(7)  # Create the network,
    loss_fn = torch.nn.MSELoss()  # and choose the loss function / optimizer
    optim = torch.optim.SGD(my_net.parameters(), lr=learn_rate)

    if torch.cuda.is_available():
        # Move the network and the optimizer to the GPU
        my_net = my_net.cuda()
        loss_fn = loss_fn.cuda()

    dataset = torch.utils.data.TensorDataset(
        to_tensor(data), to_tensor(labels_binary))
    data_loader = torch.utils.data.DataLoader(
        dataset, batch_size=minibatch_size, shuffle=True)

    for _ in range(num_epochs):
        for (input_val, label) in data_loader:
            optim.zero_grad()  # Reset the gradients

            prediction = my_net(to_variable(input_val))  # Feed forward
            loss = loss_fn(prediction, to_variable(label))  # Compute losses
            loss.backward()  # Backpropagate the gradients

            optim.step()  # Update the network

    return my_net
```

Testing it out

- Now run the training routine for 100 epochs with batch size 50 and learning rate 2.0
- Verify that the network has indeed learned to count the number of 1’s by feeding it some custom inputs

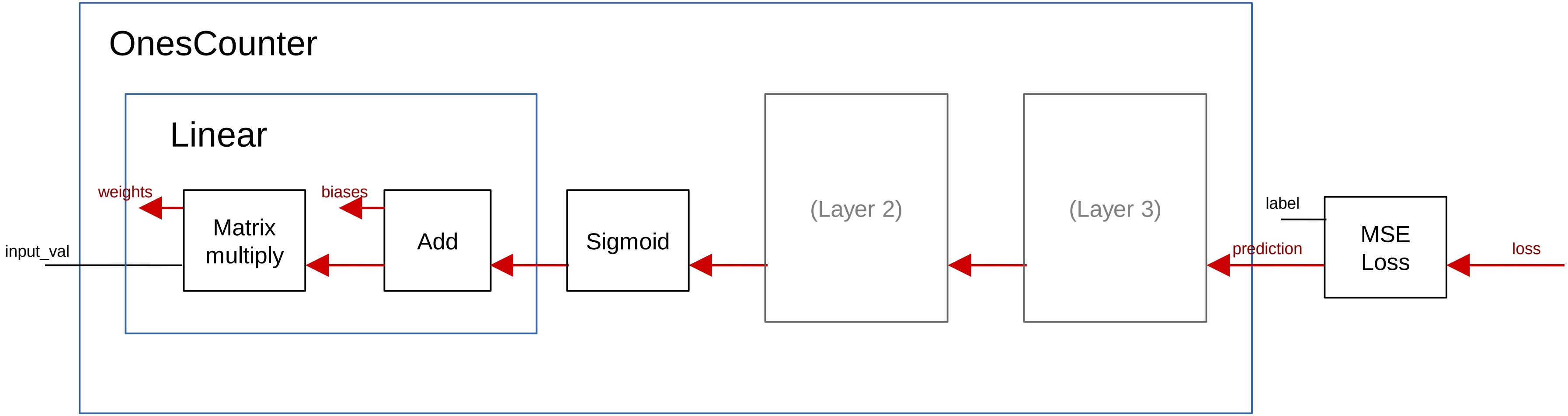
What happens behind the hood

- During the forward pass, pyTorch creates a *computational graph* that describes how the input data is passed between modules and how each module modifies the data.



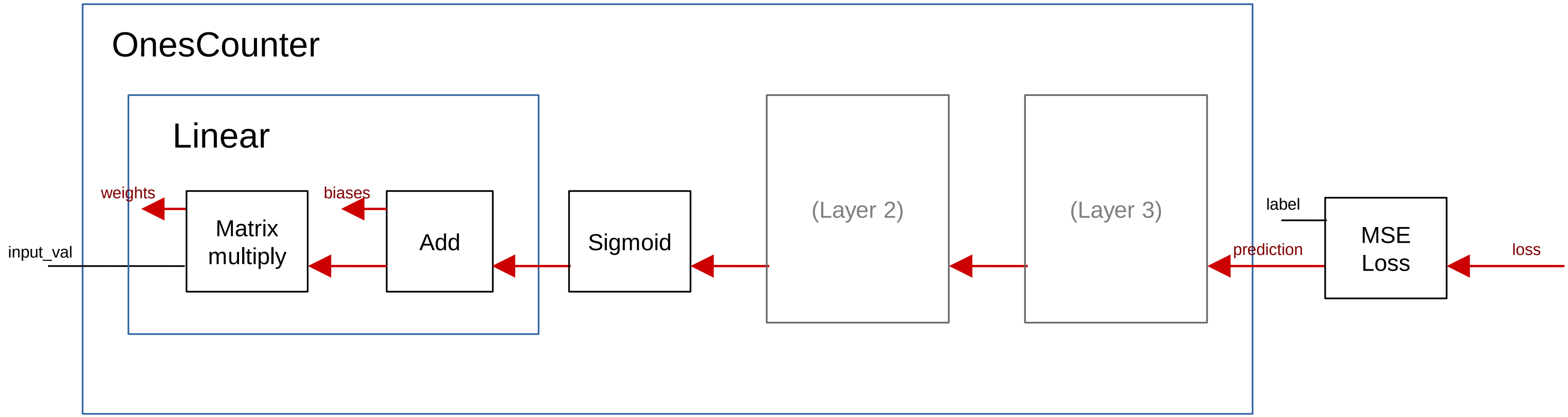
What happens behind the hood

- When `loss.backward()` is called, pyTorch computes the gradient of some variables with respect to the loss. The gradients are temporarily stored inside the Variable objects.



What happens behind the hood

- The optimizer uses these gradients to update the parameters of the network.



Adding complexity

- We only made a bare-bones system where we can train a simple network and test it on some hand-picked inputs.
- Features that we probably should add
  - Take in command line arguments for the batch size, number of epochs, learning rate... etc.
  - Saving *checkpoints* (a file containing the network parameters, which can be loaded when needed)
  - Performing evaluation on a test set
  - Performing cross validation, and creating a backup of the network when the validation score starts to become worse.
- There are many features in PyTorch we didn’t cover. Thankfully, you can find detailed documentation and tutorials online.

Utility Libraries

- Pytorch does not include (much) plumbing for common tasks
  - Saving, loading, logging, validating
- Utility libraries like inferno can take care of the boilerplate
  - You can just focus on the model itself

```
trainer = Trainer(model) \
  .build_criterion('CrossEntropyLoss') \
  .build_metric('CategoricalError') \
  .build_optimizer('Adam') \
  .validate_every((2, 'epochs')) \
  .save_every((5, 'epochs')) \
  .save_to_directory(SAVE_DIRECTORY) \
  .set_max_num_epochs(10) \
  .build_logger(TensorboardLogger(log_scalars_every=(1, 'iteration'),
                                  log_images_every='never'),
                log_directory=LOG_DIRECTORY)
```

Other Frameworks/Libraries to Know

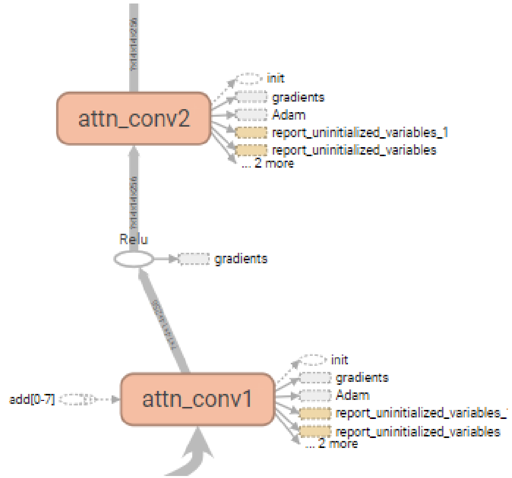
TensorFlow

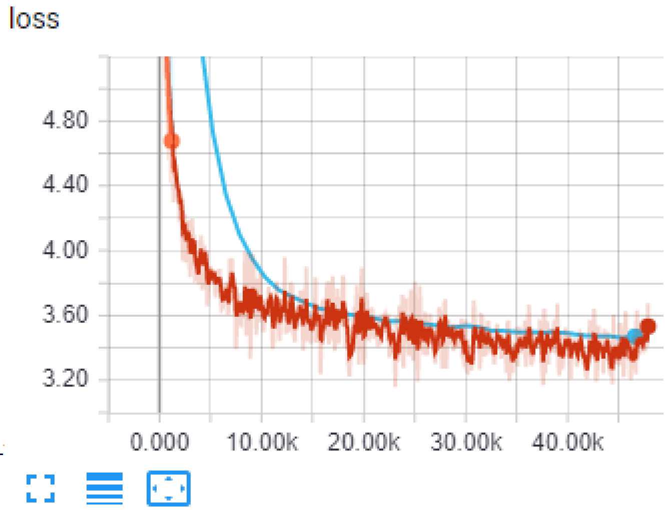
- TensorFlow is another commonly used deep learning framework.
- Two stages
  - Build a static computational graph with placeholders
  - Execute the graph, feeding placeholder values

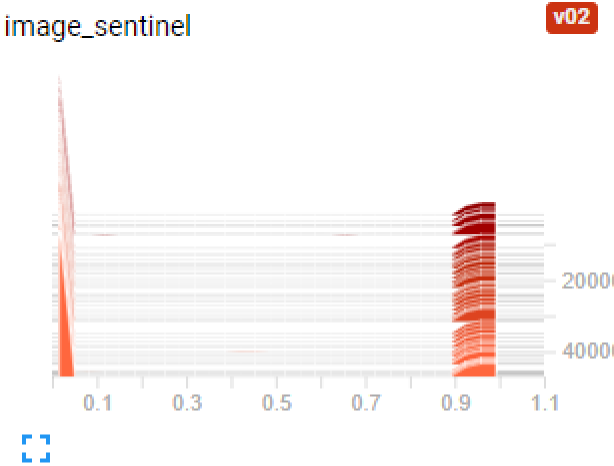
```
# Build a graph
x = tf.placeholder(tf.float32, [None])
y = x*2
# Run graph
With tf.Session() as s:
  result = s.run(y, feed_dict={x:5}
```

Tensorboard

- Component of TensorFlow for visualization, but can be used with pytorch
  - Visualize computation graph of tensors and computations
  - Live graphs of arbitrary scalars (loss, accuracy, etc.)
  - Histograms (useful for visualizing weights)







Scipy

- Scipy contains advanced library functions for numeric computation, some of which are useful for machine learning (e.g. sparse matrices, clustering)

Scikit-learn

- Scikit-learn implements many ML algorithms such as SVM, PCA, decision trees, approximate nearest neighbor search etc.

OpenCV

- OpenCV provides many image-related functions in python
- Useful for reading, rotating, re-scaling, and other image-preprocessing tasks

NLTK

- NLTK is a toolkit for natural language
- Easiest way to get and tokenize a corpus in python
- Includes
  - Corpus downloaders
  - Tokenizers
  - Parsers

Sympy

- Somewhere between numpy and pytorch
- Provides symbolic differentiation
  - Not GPU accelerated or useful for ML
  - BUT, you can symbolically calculate integrals and derivatives
- Useful if you want to double-check your calculus

Other frameworks

- Many other deep learning frameworks exist, such as Caffe, Theano, Keras, DL4J etc.
- They differ in what components they offer, how easy it is to create custom network architectures, whether they let you make performance tweaks, whether they use static or dynamic graphs, and so on.