



Adam Paszke, Sam Gross, **Soumith Chintala**, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Alban Desmaison, Andreas Kopf, Edward Yang, Zach Devito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy & Team



What is PyTorch?



What is PyTorch?



automatic differentiation
engine

**Ndarray library
with GPU support**

gradient based
optimization package

Utilities
(data loading, etc.)

Deep Learning

Numpy-alternative

Reinforcement Learning



ndarray library

- np.ndarray <-> torch.Tensor
- 200+ operations, similar to numpy
- very fast acceleration on NVIDIA GPUs



```

# -*- coding: utf-8 -*-
import numpy as np

# N is batch size; D_in is input dimension;
# H is hidden dimension; D_out is output dimension.
N, D_in, H, D_out = 64, 1000, 100, 10

# Create random input and output data
x = np.random.randn(N, D_in)
y = np.random.randn(N, D_out)

# Randomly initialize weights
w1 = np.random.randn(D_in, H)
w2 = np.random.randn(H, D_out)

learning_rate = 1e-6
for t in range(500):
    # Forward pass: compute predicted y
    h = x.dot(w1)
    h_relu = np.maximum(h, 0)
    y_pred = h_relu.dot(w2)

    # Compute and print loss
    loss = np.square(y_pred - y).sum()
    print(t, loss)

    # Backprop to compute gradients of w1 and w2 with respect to loss
    grad_y_pred = 2.0 * (y_pred - y)
    grad_w2 = h_relu.T.dot(grad_y_pred)
    grad_h_relu = grad_y_pred.dot(w2.T)
    grad_h = grad_h_relu.copy()
    grad_h[h < 0] = 0
    grad_w1 = x.T.dot(grad_h)

    # Update weights
    w1 -= learning_rate * grad_w1
    w2 -= learning_rate * grad_w2

```

Numpy

```

import torch
dtype = torch.FloatTensor
# dtype = torch.cuda.FloatTensor # Uncomment this to run on GPU

# N is batch size; D_in is input dimension;
# H is hidden dimension; D_out is output dimension.
N, D_in, H, D_out = 64, 1000, 100, 10

# Create random input and output data
x = torch.randn(N, D_in).type(dtype)
y = torch.randn(N, D_out).type(dtype)

# Randomly initialize weights
w1 = torch.randn(D_in, H).type(dtype)
w2 = torch.randn(H, D_out).type(dtype)

learning_rate = 1e-6
for t in range(500):
    # Forward pass: compute predicted y
    h = x.mm(w1)
    h_relu = h.clamp(min=0)
    y_pred = h_relu.mm(w2)

    # Compute and print loss
    loss = (y_pred - y).pow(2).sum()
    print(t, loss)

    # Backprop to compute gradients of w1 and w2 with respect to loss
    grad_y_pred = 2.0 * (y_pred - y)
    grad_w2 = h_relu.t().mm(grad_y_pred)
    grad_h_relu = grad_y_pred.mm(w2.t())
    grad_h = grad_h_relu.clone()
    grad_h[h < 0] = 0
    grad_w1 = x.t().mm(grad_h)

    # Update weights using gradient descent
    w1 -= learning_rate * grad_w1
    w2 -= learning_rate * grad_w2

```

PyTorch

ndarray / Tensor library

Tensors are similar to numpy's ndarrays, with the addition being that Tensors can also be used on a GPU to accelerate computing.

```
from __future__ import print_function  
import torch
```

Construct a 5x3 matrix, uninitialized:

```
x = torch.Tensor(5, 3)  
print(x)
```

Out:

```
1.00000e-25 *  
 0.4136  0.0000  0.0000  
 0.0000  1.6519  0.0000  
 1.6518  0.0000  1.6519  
 0.0000  1.6518  0.0000  
 1.6520  0.0000  1.6519  
[torch.FloatTensor of size 5x3]
```



ndarray / Tensor library

Construct a randomly initialized matrix

```
x = torch.rand(5, 3)
print(x)
```

Out:

```
0.2598  0.7231  0.8534
0.3928  0.1244  0.5110
0.5476  0.2700  0.5856
0.7288  0.9455  0.8749
0.6663  0.8230  0.2713
[torch.FloatTensor of size 5x3]
```

Get its size

```
print(x.size())
```

Out:

```
torch.Size([5, 3])
```



ndarray / Tensor library

You can use standard numpy-like indexing with all bells and whistles!

```
print(x[:, 1])
```

Out:

```
0.7231
0.1244
0.2700
0.9455
0.8230
[torch.FloatTensor of size 5]
```



ndarray / Tensor library

```
y = torch.rand(5, 3)
print(x + y)
```

Out:

```
0.7931  1.1872  1.6143
1.1946  0.4669  0.9639
0.7576  0.8136  1.1897
0.7431  1.8579  1.3400
0.8188  1.1041  0.8914
[torch.FloatTensor of size 5x3]
```



NumPy bridge

Converting torch Tensor to numpy Array

```
a = torch.ones(5)  
print(a)
```

Out:

```
1  
1  
1  
1  
1  
[torch.FloatTensor of size 5]
```

```
b = a.numpy()  
print(b)
```

Out:

```
[ 1.  1.  1.  1.  1.]
```



NumPy bridge

Converting torch Tensor to numpy Array

```
a = torch.ones(5)  
print(a)
```

Out:

```
1  
1  
1  
1  
1  
[torch.FloatTensor of size 5]
```

**Zero memory-copy
very efficient**

```
b = a.numpy()  
print(b)
```

Out:

```
[ 1.  1.  1.  1.  1.]
```



NumPy bridge

See how the numpy array changed in value.

```
a.add_(1)  
print(a)  
print(b)
```

Out:

```
2  
2  
2  
2  
2  
[torch.FloatTensor of size 5]  
[ 2.  2.  2.  2.  2.]
```



NumPy bridge

Converting numpy Array to torch Tensor

See how changing the np array changed the torch Tensor automatically

```
import numpy as np
a = np.ones(5)
b = torch.from_numpy(a)
np.add(a, 1, out=a)
print(a)
print(b)
```

Out:

```
[ 2.  2.  2.  2.  2.]
2
2
2
2
2
[torch.DoubleTensor of size 5]
```

All the Tensors on the CPU except a CharTensor support converting to NumPy and back.



Seamless GPU Tensors

CUDA Tensors

Tensors can be moved onto GPU using the `.cuda` function.

```
# let us run this cell only if CUDA is available
if torch.cuda.is_available():
    x = x.cuda()
    y = y.cuda()
    x + y
```



automatic differentiation engine

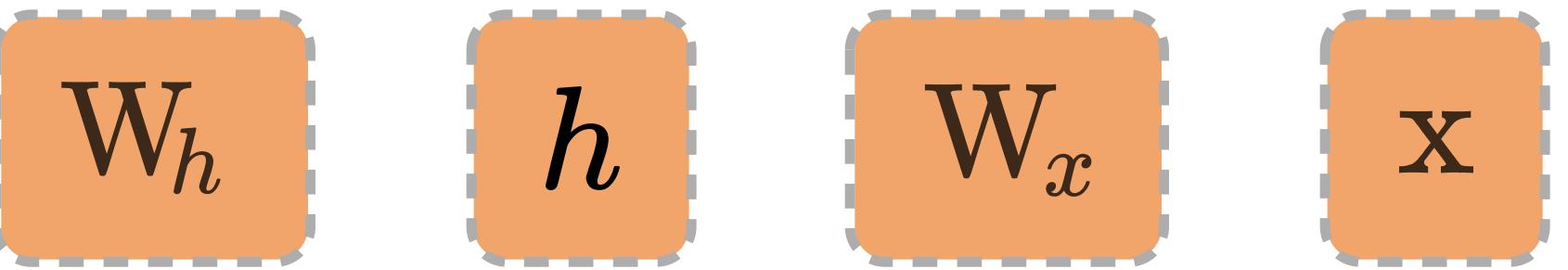
for deep learning and reinforcement learning



PyTorch Autograd

```
from torch.autograd import Variable
```

PyTorch Autograd

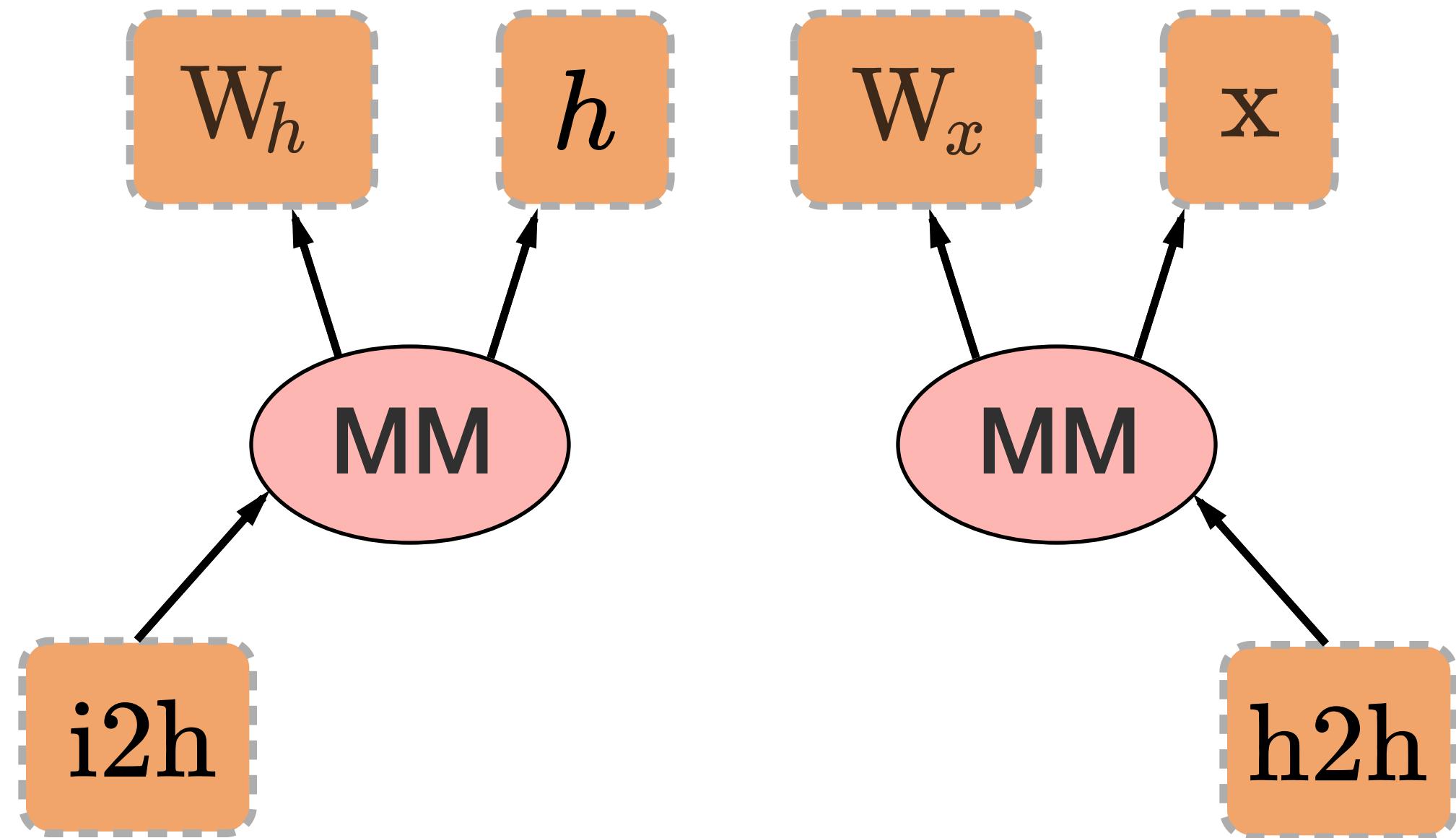


```
from torch.autograd import Variable
```

```
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))
```

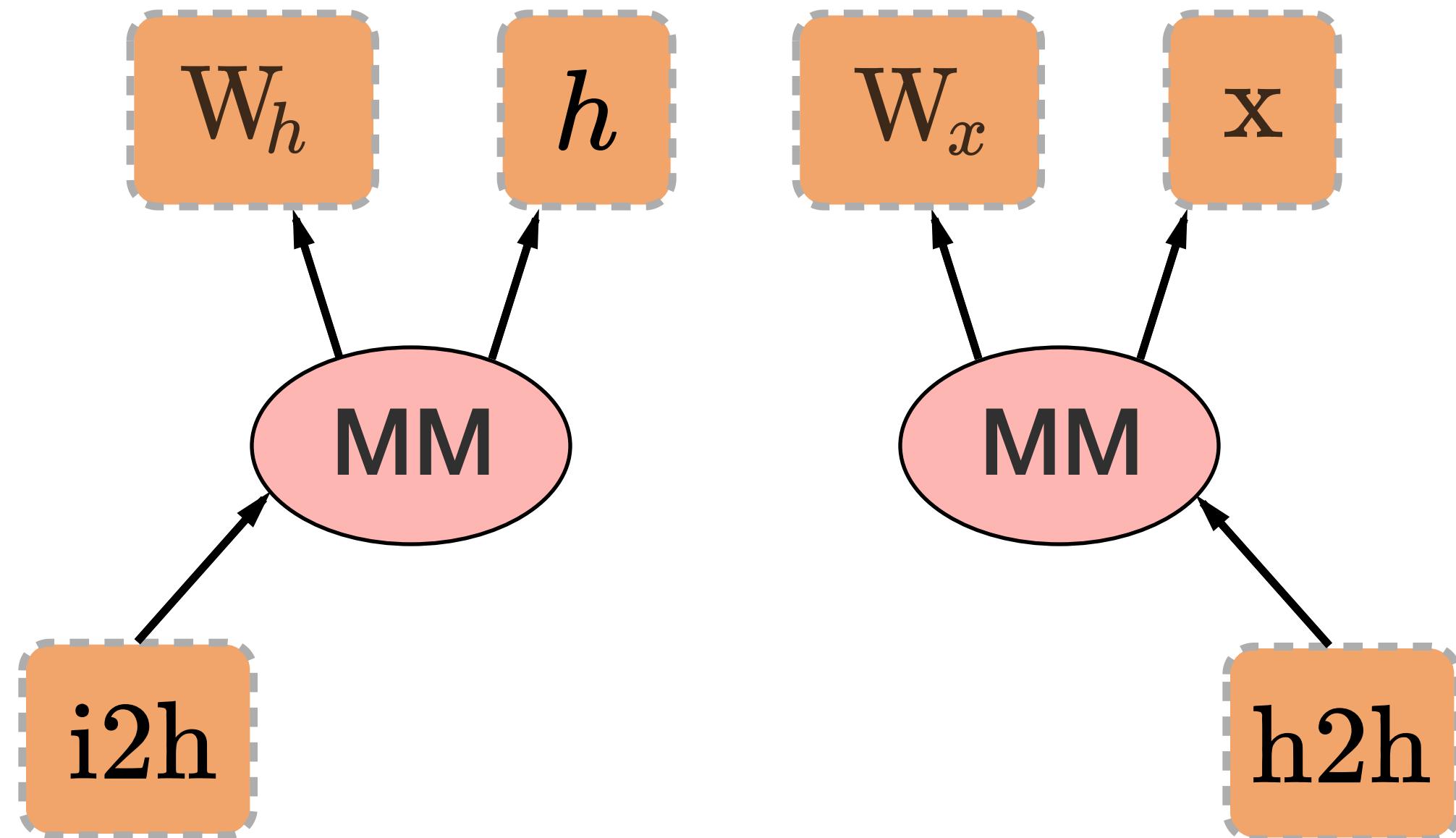
PyTorch Autograd

```
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))  
  
i2h = torch.mm(W_x, x.t())  
h2h = torch.mm(W_h, prev_h.t())
```



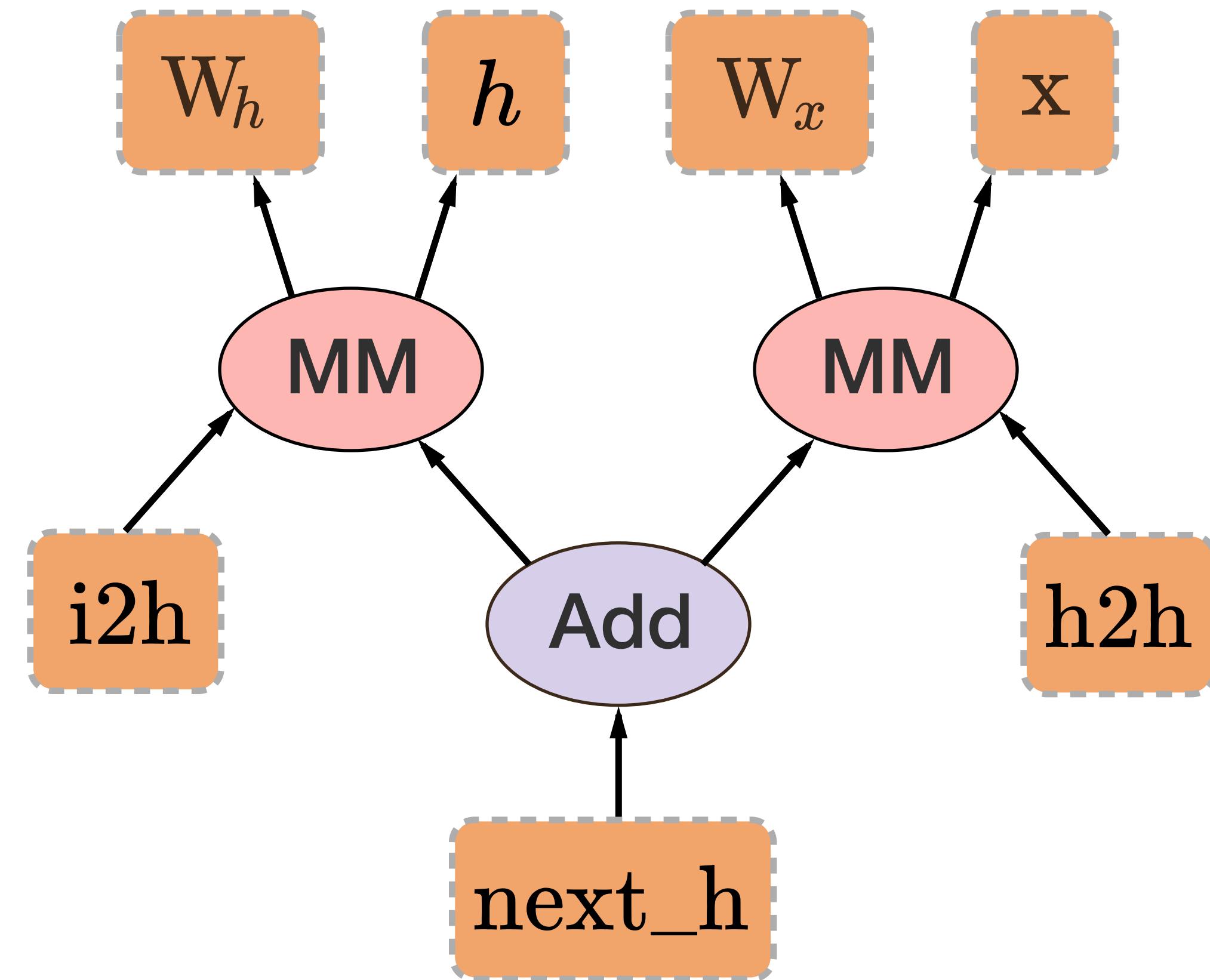
PyTorch Autograd

```
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))  
  
i2h = torch.mm(W_x, x.t())  
h2h = torch.mm(W_h, prev_h.t())  
next_h = i2h + h2h
```



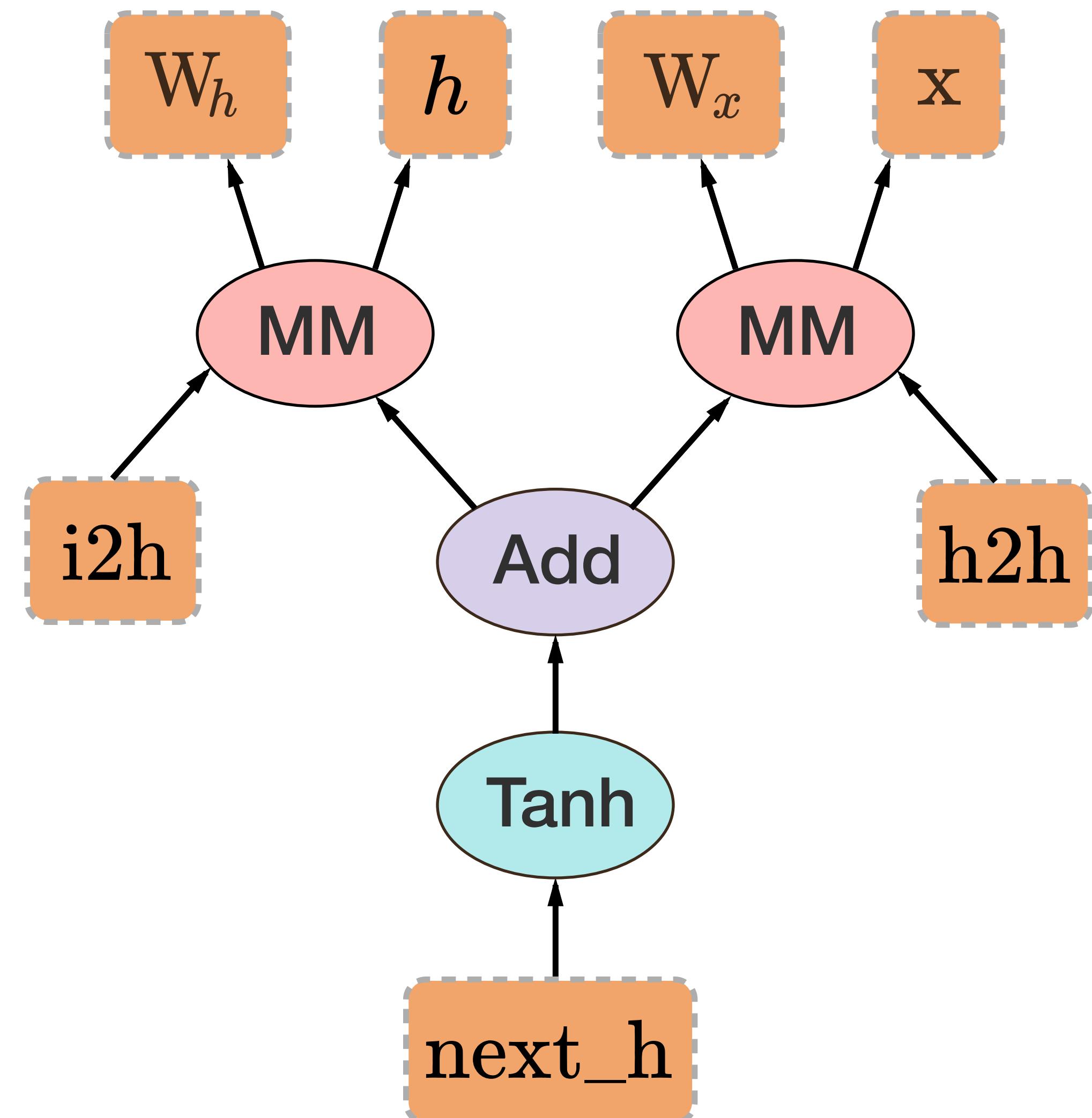
PyTorch Autograd

```
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))  
  
i2h = torch.mm(W_x, x.t())  
h2h = torch.mm(W_h, prev_h.t())  
next_h = i2h + h2h
```



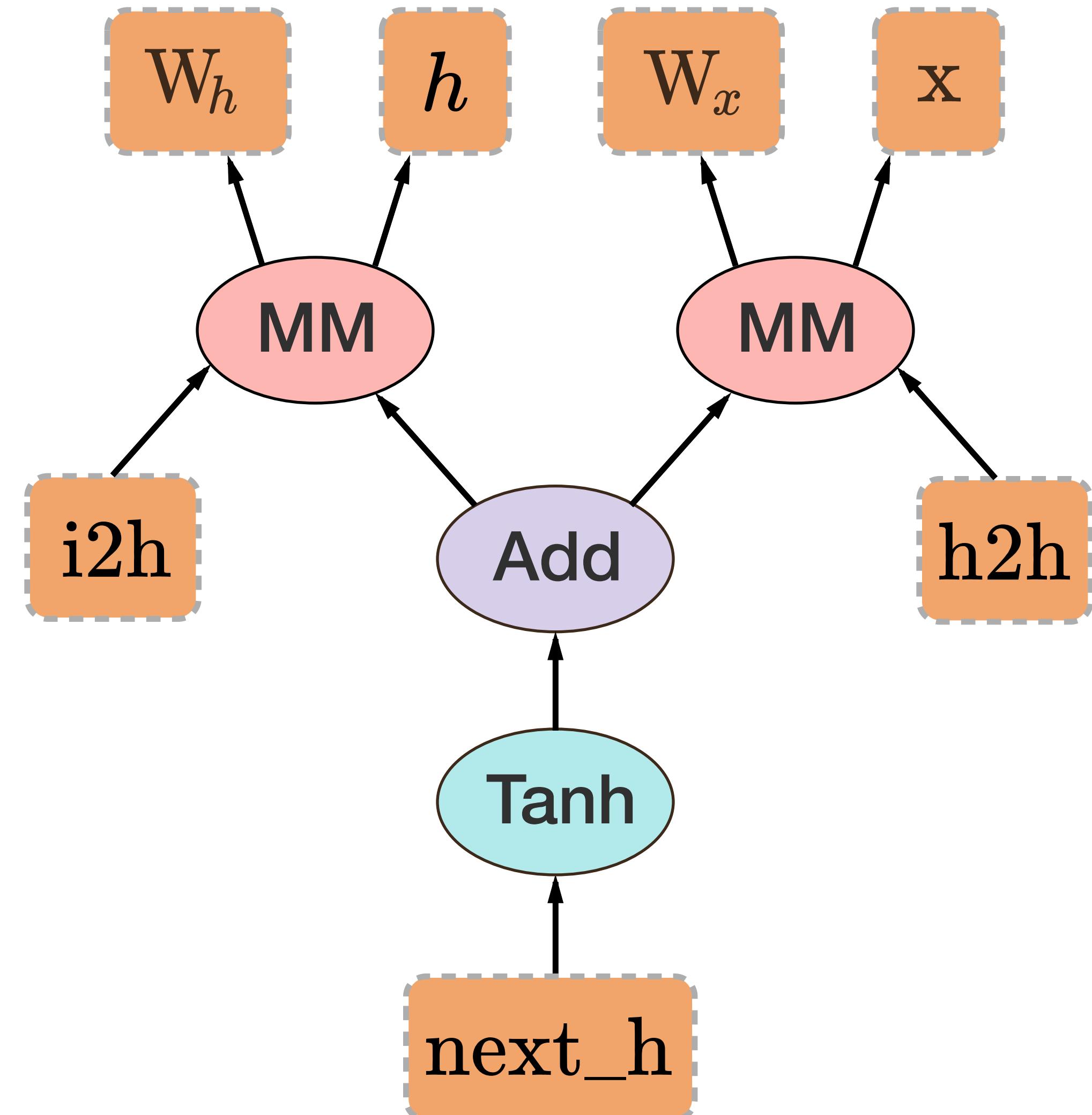
PyTorch Autograd

```
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))  
  
i2h = torch.mm(W_x, x.t())  
h2h = torch.mm(W_h, prev_h.t())  
next_h = i2h + h2h  
next_h = next_h.tanh()
```



PyTorch Autograd

```
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))  
  
i2h = torch.mm(W_x, x.t())  
h2h = torch.mm(W_h, prev_h.t())  
next_h = i2h + h2h  
next_h = next_h.tanh()  
  
next_h.backward(torch.ones(1, 20))
```



Neural Networks

```
1  class Net(nn.Module):
2      def __init__(self):
3          super(Net, self).__init__()
4          self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
5          self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
6          self.conv2_drop = nn.Dropout2d()
7          self.fc1 = nn.Linear(320, 50)
8          self.fc2 = nn.Linear(50, 10)
9
10     def forward(self, x):
11         x = F.relu(F.max_pool2d(self.conv1(x), 2))
12         x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
13         x = x.view(-1, 320)
14         x = F.relu(self.fc1(x))
15         x = F.dropout(x, training=self.training)
16         x = self.fc2(x)
17         return F.log_softmax(x)
18
19 model = Net()
20 input = Variable(torch.randn(10, 20))
21 output = model(input)
```

Neural Networks

```
1  class Net(nn.Module):
2      def __init__(self):
3          super(Net, self).__init__()
4          self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
5          self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
6          self.conv2_drop = nn.Dropout2d()
7          self.fc1 = nn.Linear(320, 50)
8          self.fc2 = nn.Linear(50, 10)
9
10     def forward(self, x):
11         x = F.relu(F.max_pool2d(self.conv1(x), 2))
12         x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
13         x = x.view(-1, 320)
14         x = F.relu(self.fc1(x))
15         x = F.dropout(x, training=self.training)
16         x = self.fc2(x)
17         return F.log_softmax(x)
18
19 model = Net()
20 input = Variable(torch.randn(10, 20))
21 output = model(input)
```

Neural Networks

```
1  class Net(nn.Module):
2      def __init__(self):
3          super(Net, self).__init__()
4          self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
5          self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
6          self.conv2_drop = nn.Dropout2d()
7          self.fc1 = nn.Linear(320, 50)
8          self.fc2 = nn.Linear(50, 10)
9
10     def forward(self, x):
11         x = F.relu(F.max_pool2d(self.conv1(x), 2))
12         x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
13         x = x.view(-1, 320)
14         x = F.relu(self.fc1(x))
15         x = F.dropout(x, training=self.training)
16         x = self.fc2(x)
17         return F.log_softmax(x)
18
19 model = Net()
20 input = Variable(torch.randn(10, 20))
21 output = model(input)
```

Optimization package

SGD, Adagrad, RMSProp, LBFGS, etc.

```
1 net = Net()
2 optimizer = torch.optim.SGD(net.parameters(), lr=0.01, momentum=0.9)
3
4 for input, target in dataset:
5     optimizer.zero_grad()
6     output = model(input)
7     loss = F.cross_entropy(output, target)
8     loss.backward()
9     optimizer.step()
```

Work items in practice

Writing
Dataset loaders

Building models

Implementing
Training loop

Checkpointing
models

Interfacing with
environments

Building optimizers

Dealing with
GPUs

Building
Baselines



Work items in practice

Writing
Dataset loaders

Building models

Implementing
Training loop

Checkpointing
models

Python + PyTorch - an environment to do all of this

Interfacing with
environments

Building optimizers

Dealing with
GPUs

Building
Baselines



Writing Data Loaders

- every dataset is slightly differently formatted



Writing Data Loaders

- every dataset is slightly differently formatted
- have to be preprocessed and normalized differently



Writing Data Loaders

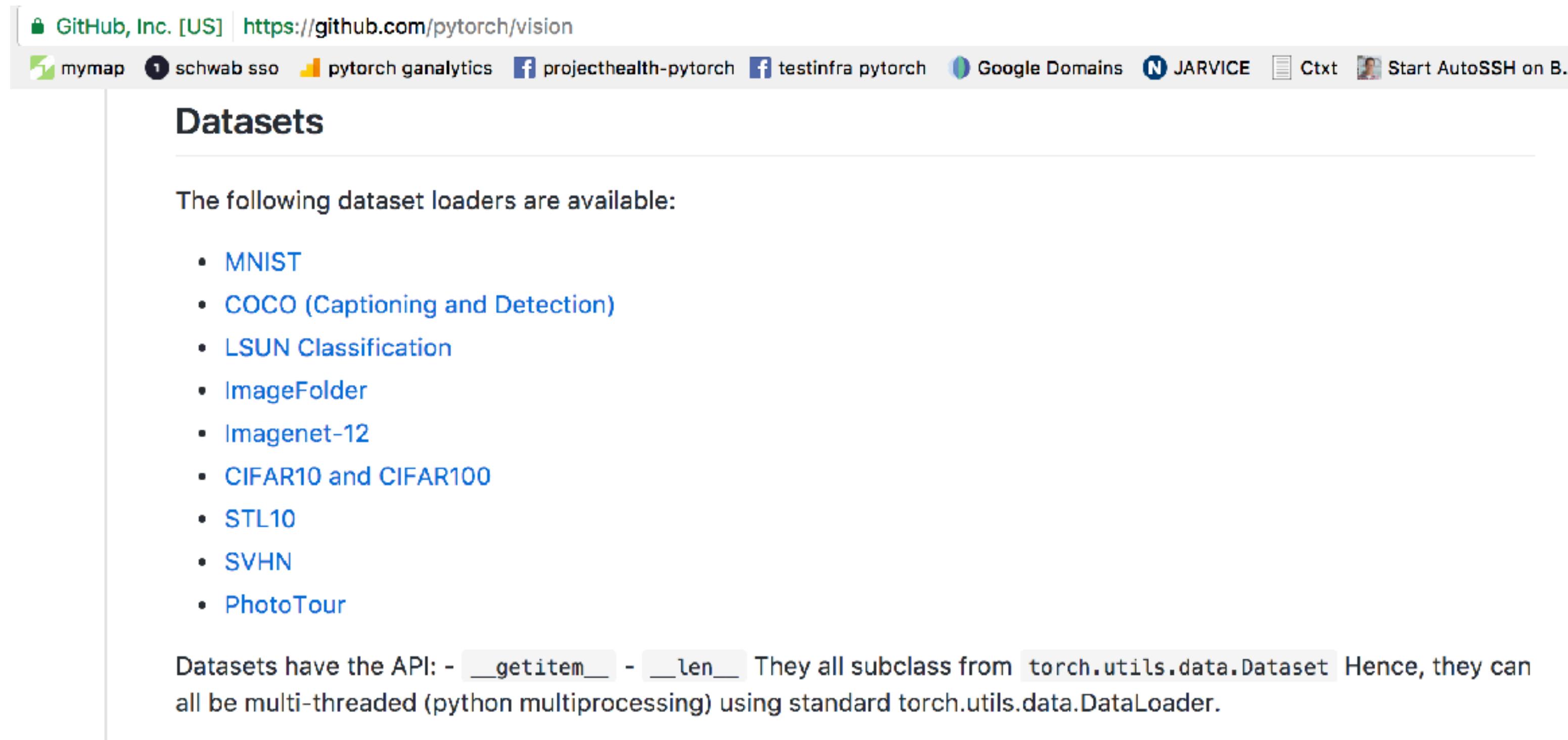
- every dataset is slightly differently formatted
- have to be preprocessed and normalized differently
- need a multithreaded Data loader to feed GPUs fast enough



Writing Data Loaders

PyTorch solution:

- share data loaders across the community!



The screenshot shows a GitHub page for the PyTorch vision datasets. The URL is <https://github.com/pytorch/vision>. The page title is "Datasets". It lists several dataset loaders available:

- MNIST
- COCO (Captioning and Detection)
- LSUN Classification
- ImageFolder
- Imagenet-12
- CIFAR10 and CIFAR100
- STL10
- SVHN
- PhotoTour

At the bottom, there is a note: "Datasets have the API: - `__getitem__` - `__len__` They all subclass from `torch.utils.data.Dataset` Hence, they can all be multi-threaded (python multiprocessing) using standard `torch.utils.data.DataLoader`".

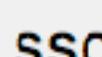
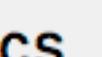


Writing Data Loaders

PyTorch solution:

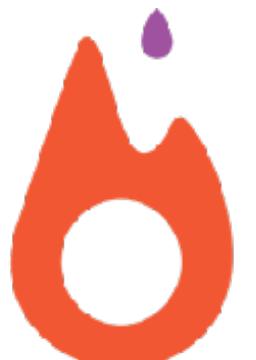
- share data loaders across the community!

 GitHub, Inc. [US] | <https://github.com/pytorch/text>

 mymap  schwab sso  pytorch ganalytics  projecthealth-pytorch  testinfra pytorch  Google Domains

This repository consists of:

- `torchtext.data` : Generic data loaders, abstractions, and iterators for text
- `torchtext.datasets` : Pre-built loaders for common NLP datasets
- (maybe) `torchtext.models` : Model definitions and pre-trained models for particular situations. The situation is not the same as vision, where people can download a pre-trained model and make it useful for other tasks -- it might make more sense to leave NLP models as separate components.



Writing Data Loaders

PyTorch solution:

- use regular Python to write Datasets:
leverage existing Python code



Writing Data Loaders

PyTorch solution:

- use regular Python to write Datasets:
leverage existing Python code

Example: ParlAI



ParlAI

ParlAI (pronounced "par-lay") is a framework for dialog AI research, implemented in Python.

Its goal is to provide researchers:

- a unified framework for training and testing dialog models
- multi-task training over many datasets at once
- seamless integration of [Amazon Mechanical Turk](#) for data collection and human evaluation

Over 20 tasks are supported in the first release, including popular datasets such as [SQuAD](#), [bAbI tasks](#), [MCTest](#), [WikiQA](#), [WebQuestions](#), [SimpleQuestions](#), [WikiMovies](#), [QACNN & QADailyMail](#), [CBT](#), [BookTest](#), [bAbI Dialog tasks](#), [Ubuntu Dialog](#), [OpenSubtitles](#), [Cornell Movie](#) and [VQA-COCO2014](#).



Writing Data Loaders

PyTorch solution:

- Code in practice



Writing PyTorch Code in Python

```
57 if opt.dataset in ['imagenet', 'folder', 'lfw']:
58     # folder dataset
59     dataset = dset.ImageFolder(root=opt.dataroot,
60                               transform=transforms.Compose([
61                                 transforms.Scale(opt.imageSize),
62                                 transforms.CenterCrop(opt.imageSize),
63                                 transforms.ToTensor(),
64                                 transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
65                               ]))
66 elif opt.dataset == 'lsun':
67     dataset = dset.LSUN(db_path=opt.dataroot, classes=['bedroom_train'],
68                         transform=transforms.Compose([
69                           transforms.Scale(opt.imageSize),
70                           transforms.CenterCrop(opt.imageSize),
71                           transforms.ToTensor(),
72                           transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
73                         ]))
74 elif opt.dataset == 'cifar10':
75     dataset = dset.CIFAR10(root=opt.dataroot, download=True,
76                           transform=transforms.Compose([
77                             transforms.Scale(opt.imageSize),
78                             transforms.ToTensor(),
79                             transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
80                           ]))
81 )
82 assert dataset
83 dataloader = torch.utils.data.DataLoader(dataset, batch_size=opt.batchSize,
84                                         shuffle=True, num_workers=int(opt.workers))
```



Writing Data PyTorch solution

- Code in practice

```
def __init__(self, root, annFile, transform=None, target_transform=None):
    from pycocotools.coco import COCO
    self.root = os.path.expanduser(root)
    self.coco = COCO(annFile)
    self.ids = list(self.coco.imgs.keys())
    self.transform = transform
    self.target_transform = target_transform

def __getitem__(self, index):
    """
    Args:
        index (int): Index

    Returns:
        tuple: Tuple (image, target). target is a list of captions for the image.
    """
    coco = self.coco
    img_id = self.ids[index]
    ann_ids = coco.getAnnIds(imgIds=img_id)
    anns = coco.loadAnns(ann_ids)
    target = [ann['caption'] for ann in anns]

    path = coco.loadImgs(img_id)[0]['file_name']

    img = Image.open(os.path.join(self.root, path)).convert('RGB')
    if self.transform is not None:
        img = self.transform(img)

    if self.target_transform is not None:
        target = self.target_transform(target)

    return img, target

def __len__(self):
    return len(self.ids)
```



Interfacing with environments



Cars



Video games



 **UNIVERSE**
Measurement and training for
artificial intelligence.

Internet



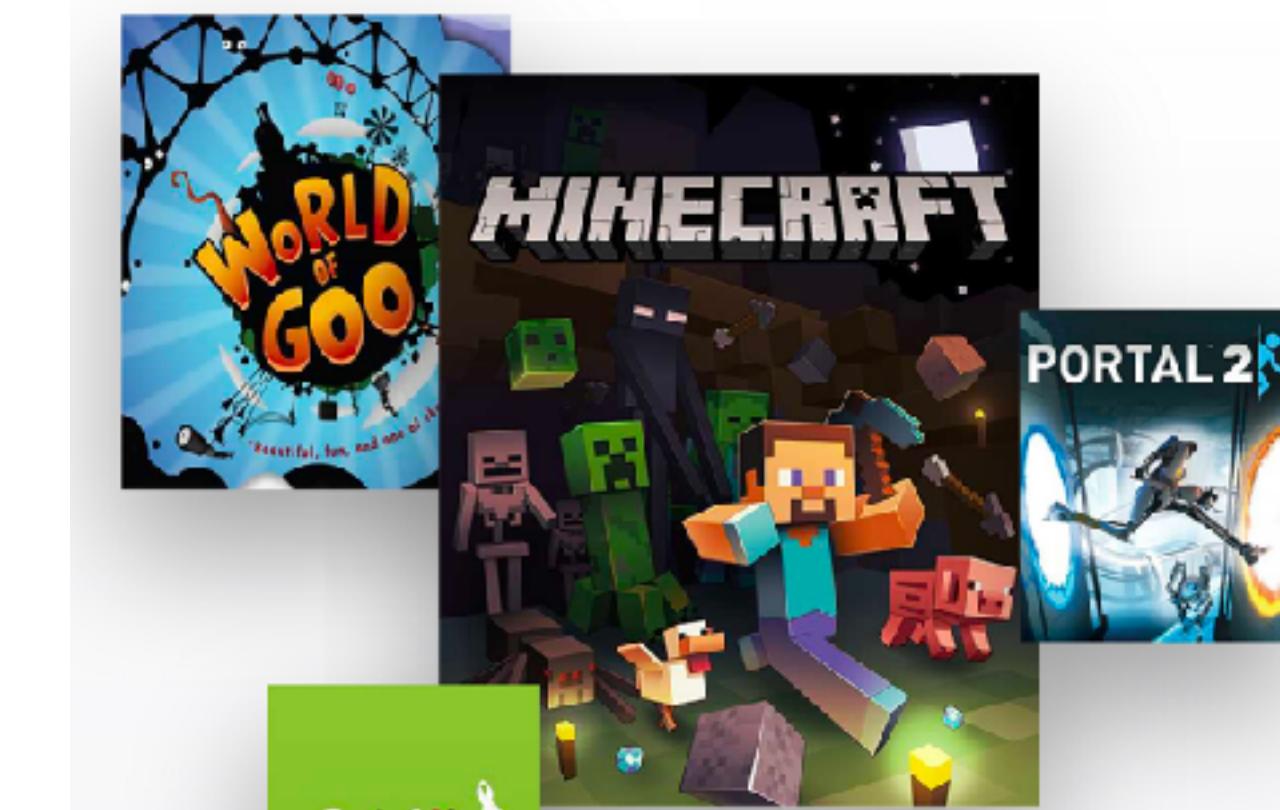
Interfacing with environments



Cars



Video games



 **UNIVERSE**
Measurement and training for
artificial intelligence.

Pretty much every environment provides a Python API



Interfacing with environments



Cars



Video games



 **UNIVERSE**
Measurement and training for
artificial intelligence.

Natively interact with the environment directly



Debugging

- PyTorch is a Python extension



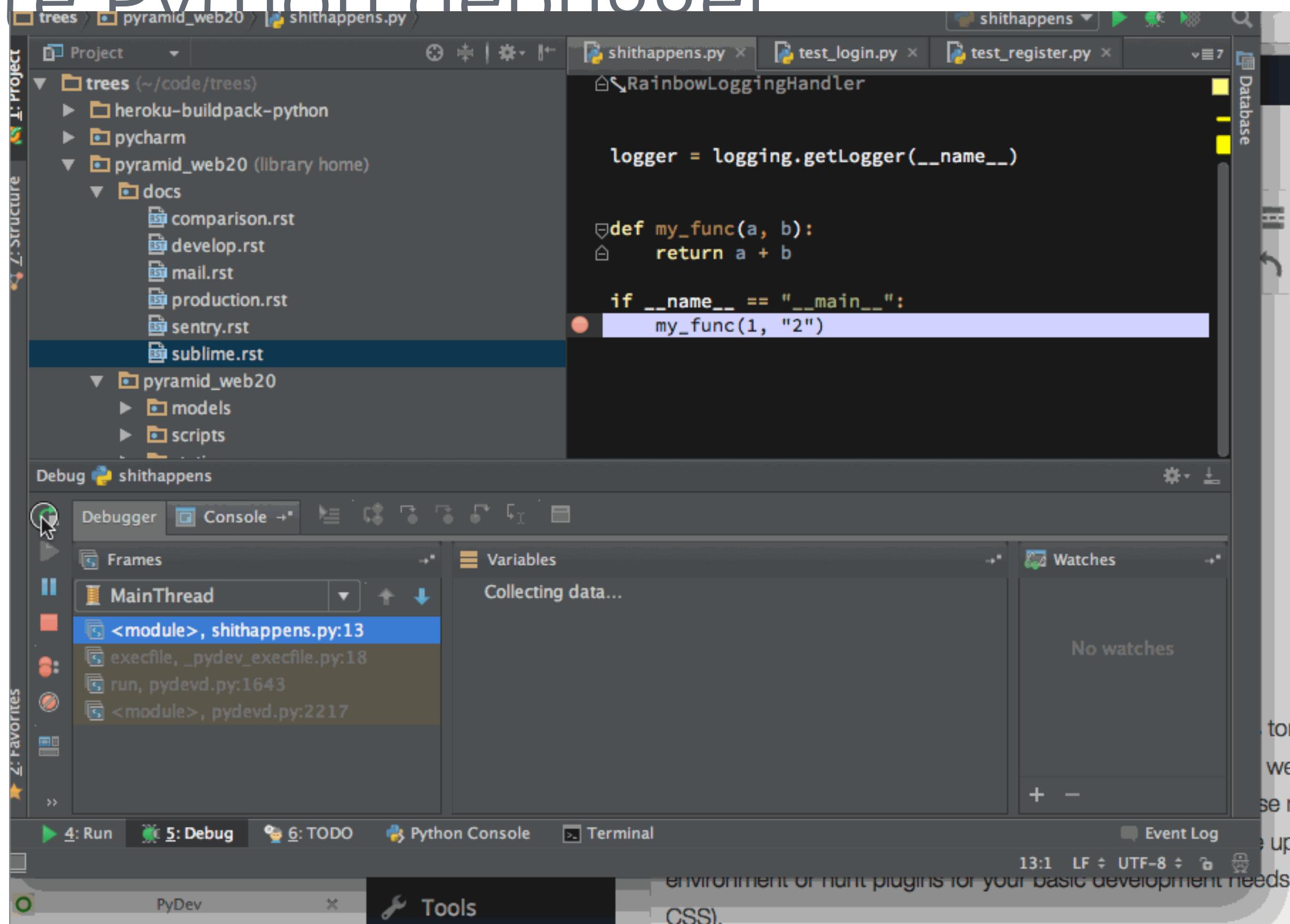
Debugging

- PyTorch is a Python extension
- Use your favorite Python debugger



Debugging

- PyTorch is a Python extension
- Use your favorite Python debugger



Debugging

- PyTorch is a Python extension
- Use your favorite Python debugger
- Use the most popular debugger:



Debugging

- PyTorch is a Python extension
- Use your favorite Python debugger
- Use the most popular debugger:

print (foo)

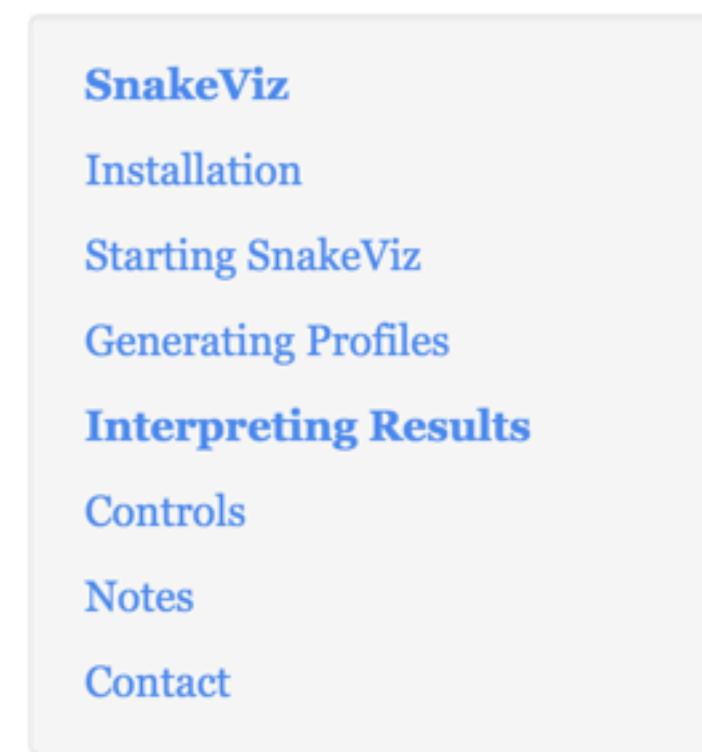


Identifying bottlenecks

- PyTorch is a Python extension
- Use your favorite Python profiler

SNAKEVIZ

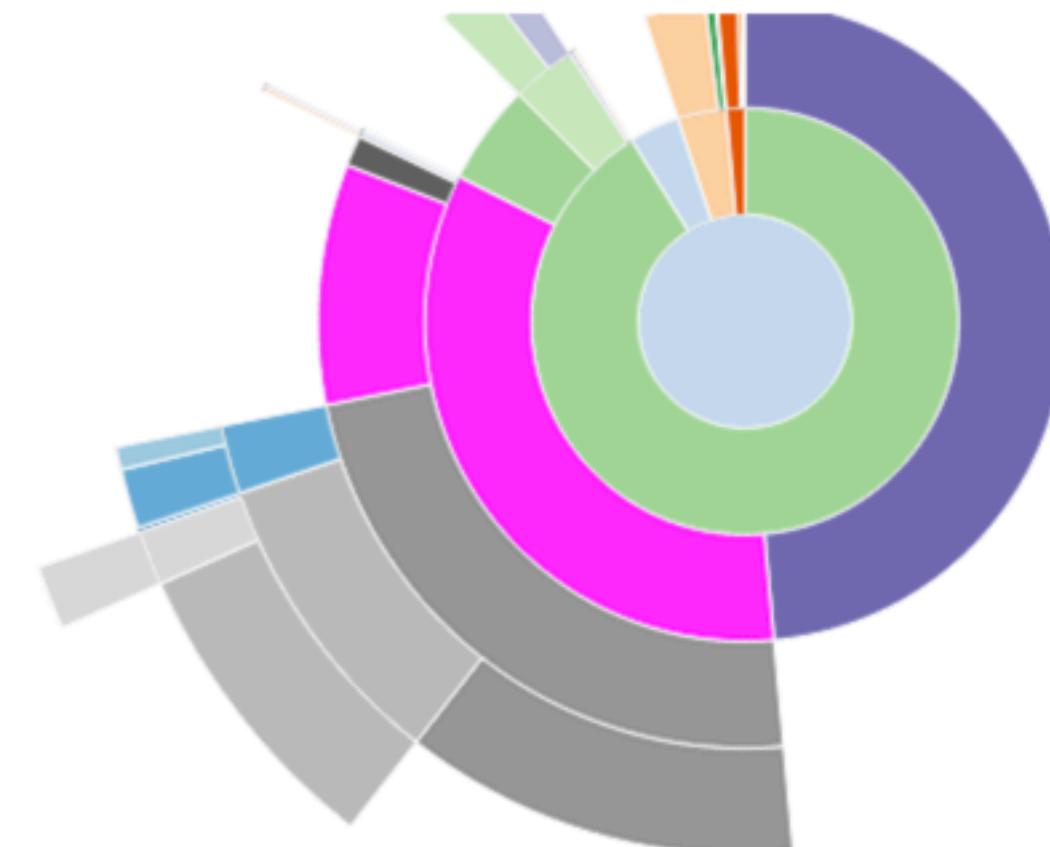
← PREVIOUS NEXT → G



FUNCTION INFO

Placing your cursor over an arc will highlight that arc and any other visible instances of the same function call. It also displays a list of information to the left of the sunburst.

Name:
filter
Cumulative Time:
0.000294 s (31.78 %)
File:
fnmatch.py
Line:
48
Directory:
/Users/jiffyclub/miniconda3/envs/snakevizdev/lib/python3.4/



Identifying bottlenecks

- PyTorch is a Python extension
- Use your favorite Python profiler: Line_Profiler

```
File: pystone.py
Function: Proc2 at line 149
Total time: 0.606656 s

Line #      Hits       Time  Per Hit   % Time  Line Contents
=====
149                      @profile
150                      def Proc2(IntParIO):
151    50000     82003     1.6    13.5
152    50000     63162     1.3    10.4
153    50000     69065     1.4    11.4
154    50000     66354     1.3    10.9
155    50000     67263     1.3    11.1
156    50000     65494     1.3    10.8
157    50000     68001     1.4    11.2
158    50000     63739     1.3    10.5
159    50000     61575     1.2    10.1
                                         return IntParIO
```



Compilation Time

- PyTorch is written for the impatient



Compilation Time

- PyTorch is written for the impatient
- Absolutely no compilation time when writing your scripts
whatsoever



Compilation Time

- PyTorch is written for the impatient
- Absolutely no compilation time when writing your scripts whatsoever
- All core kernels are pre-compiled



Distributed PyTorch

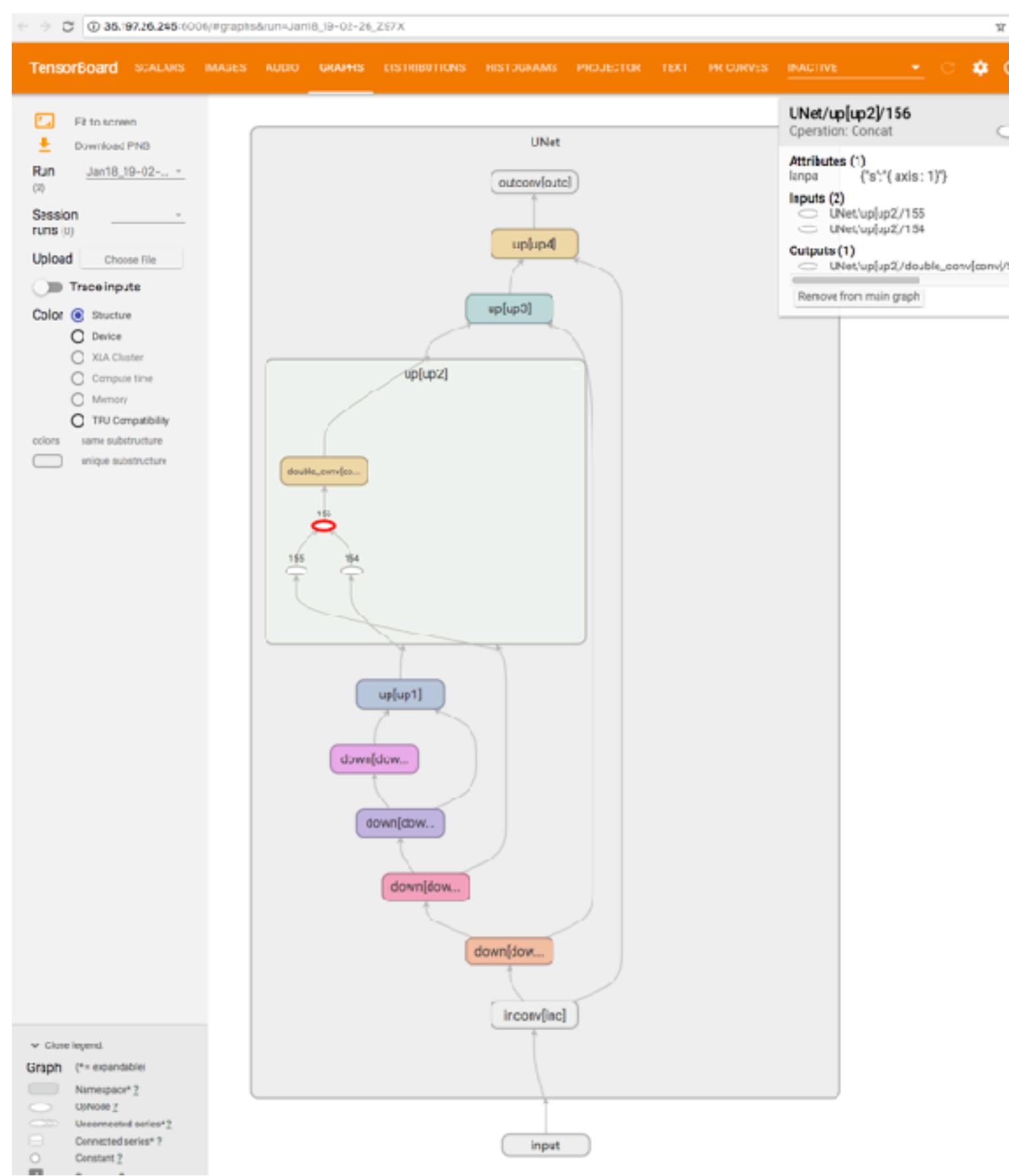
- MPI style distributed communication
- Broadcast Tensors to other nodes
- Reduce Tensors among nodes
 - for example: sum gradients among all nodes



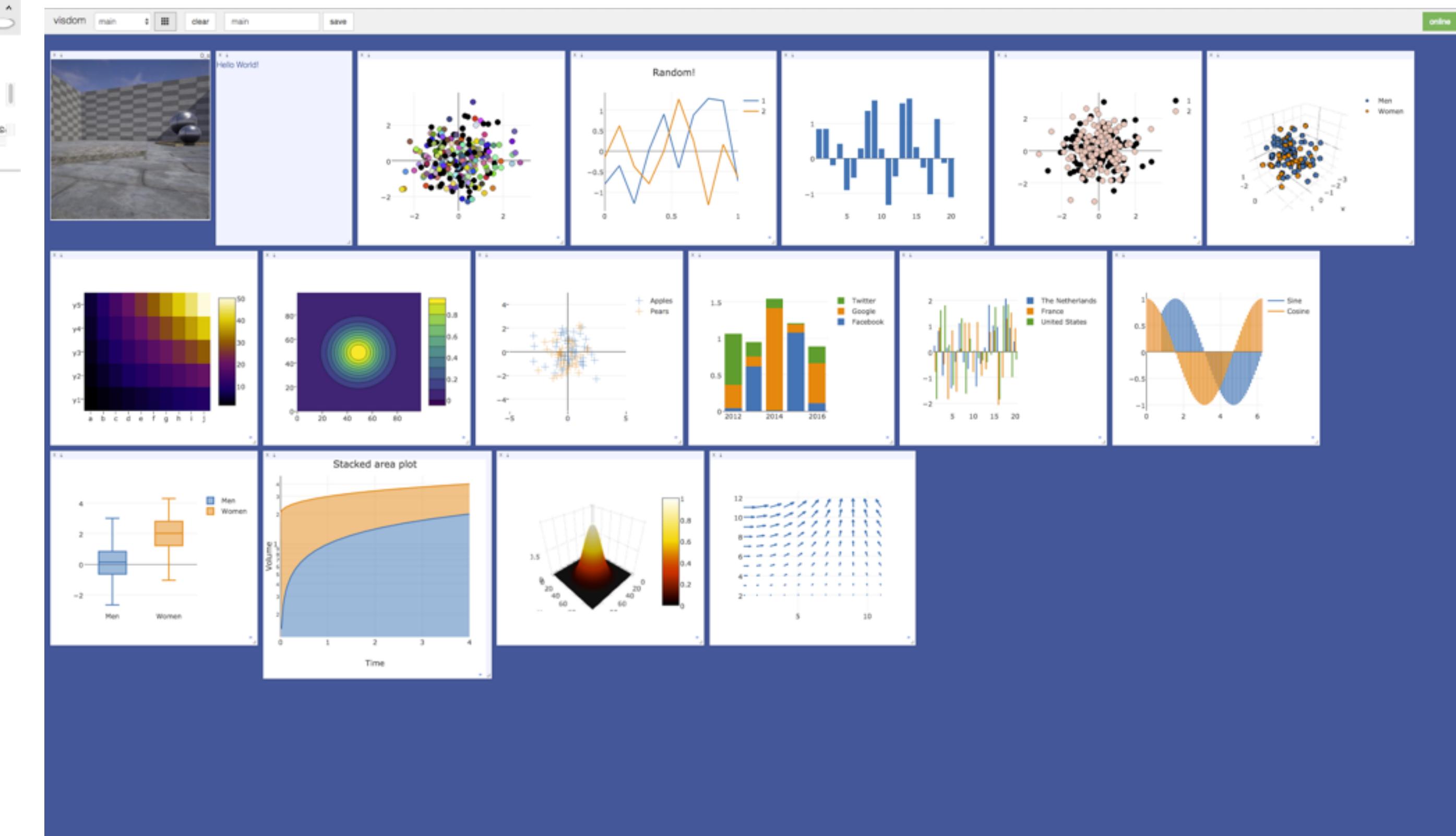
Visualization

TensorBoard-PyTorch

<https://github.com/lanpa/tensorboard-pytorch>



<https://github.com/facebookresearch/visdom>



Ecosystem

- Use the entire Python ecosystem at your will



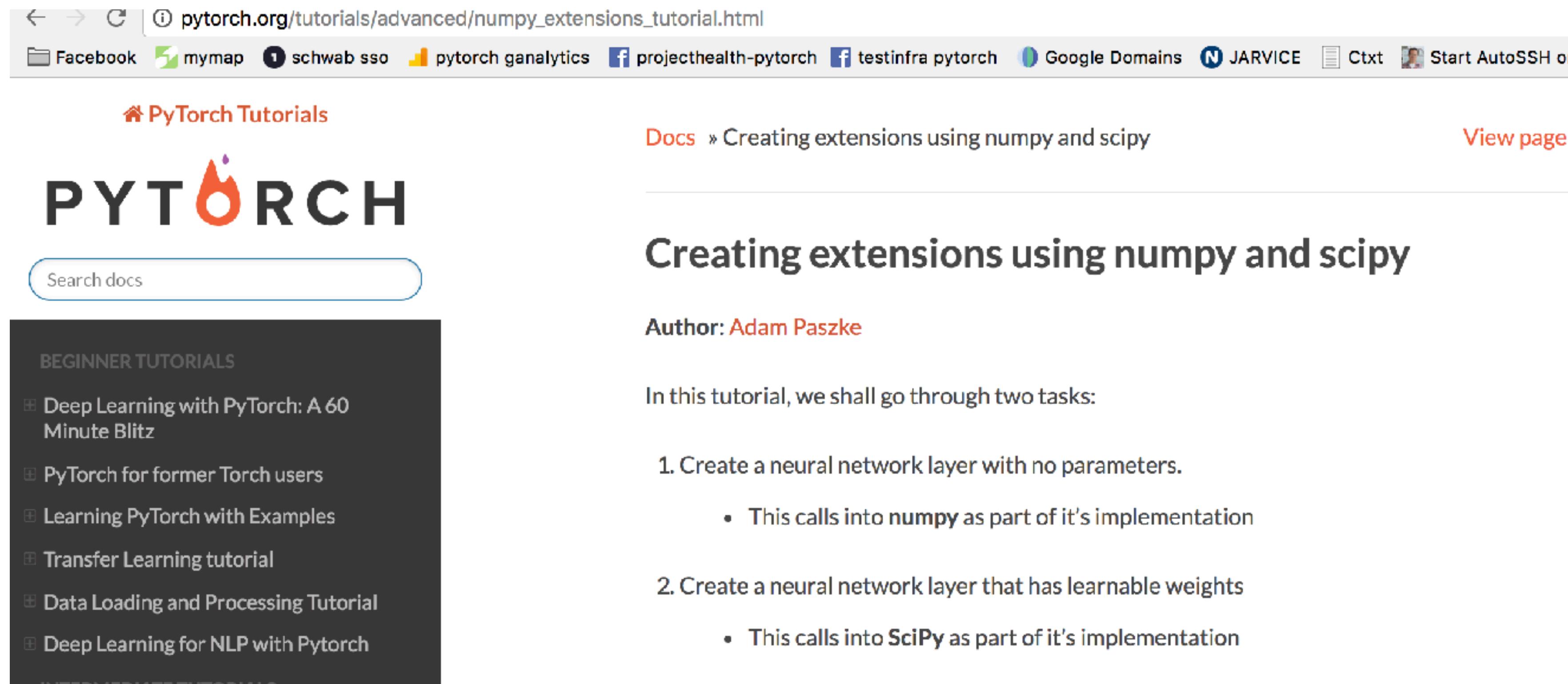
Ecosystem

- Use the entire Python ecosystem at your will
- Including SciPy, Scikit-Learn, etc.



Ecosystem

- Use the entire Python ecosystem at your will
- Including SciPy, Scikit-Learn, etc.



The screenshot shows a browser window displaying the PyTorch documentation. The URL in the address bar is `pytorch.org/tutorials/advanced/numpy_extensions_tutorial.html`. The page title is "Creating extensions using numpy and scipy". The author is listed as "Adam Paszke". The content starts with a brief introduction: "In this tutorial, we shall go through two tasks:". It then lists two numbered steps: 1. Create a neural network layer with no parameters. (This calls into numpy as part of its implementation) and 2. Create a neural network layer that has learnable weights. (This calls into SciPy as part of its implementation). On the left side of the page, there is a sidebar titled "BEGINNER TUTORIALS" which includes links to "Deep Learning with PyTorch: A 60 Minute Blitz", "PyTorch for former Torch users", "Learning PyTorch with Examples", "Transfer Learning tutorial", "Data Loading and Processing Tutorial", and "Deep Learning for NLP with Pytorch". There is also a "SEARCH DOCS" input field. The PyTorch logo is visible in the bottom right corner of the page.



Ecosystem

- A shared model-zoo:

We provide pre-trained models for the ResNet variants and AlexNet, using the PyTorch `torch.utils.model_zoo`. These can be constructed by passing `pretrained=True`:

```
import torchvision.models as models  
resnet18 = models.resnet18(pretrained=True)  
alexnet = models.alexnet(pretrained=True)
```



Ecosystem

- A shared

GitHub, Inc. [US] | https://github.com/aaron-xichen/pytorch-playground

mymap schwab sso pytorch ganalytics projecthealth-pytorch testinfra pytorch Google Domains JARVICE Ctxt Start Auto

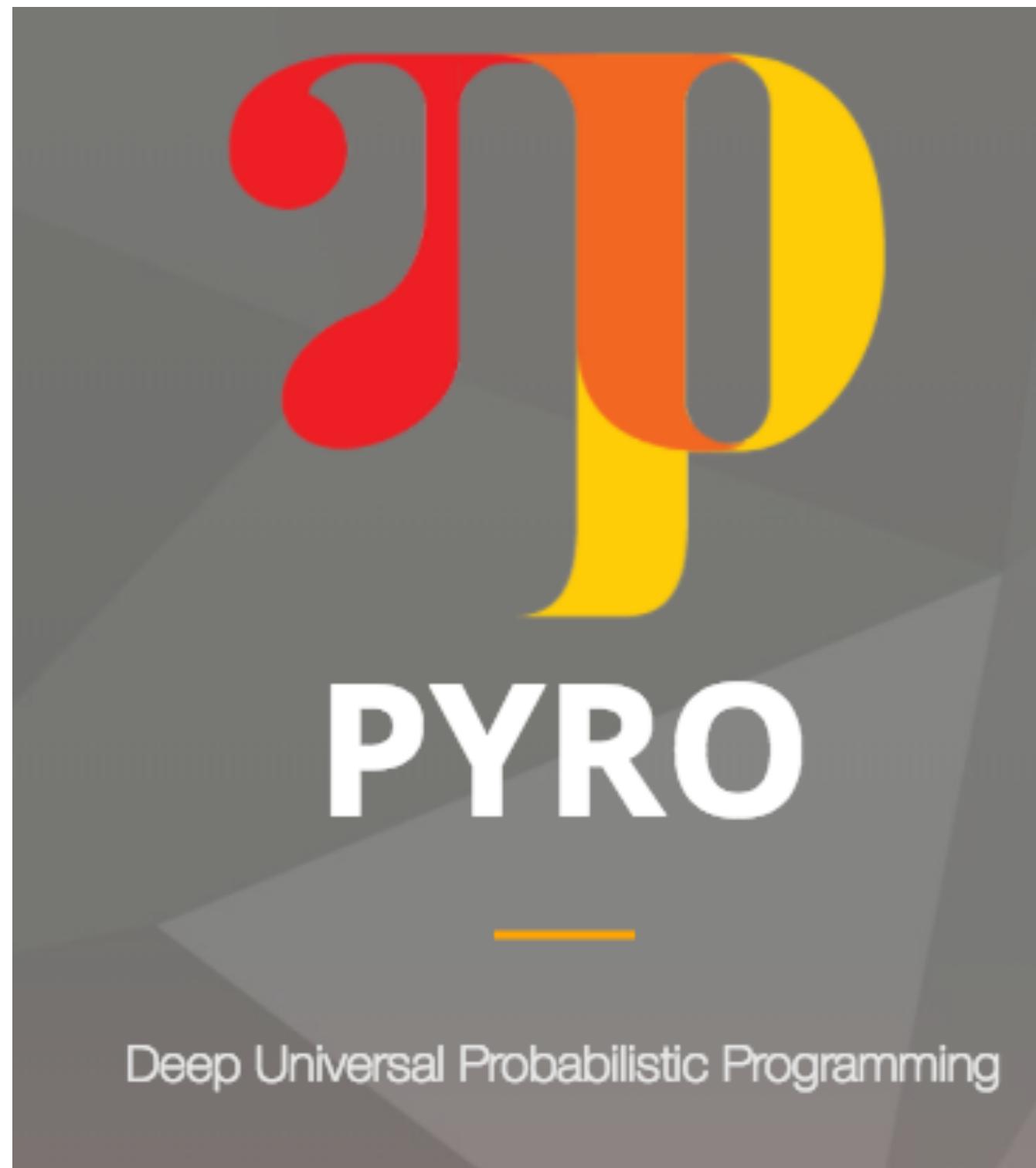
We evaluate the performance of popular dataset and models with linear quantized method. The bit-width of running mean and running variance in BN are 10 bits for all results. (except for 32-float)

Model	32-float	12-bit	10-bit	8-bit	6-bit
MNIST	98.42	98.43	98.44	98.44	98.32
SVHN	96.03	96.03	96.04	96.02	95.46
CIFAR10	93.78	93.79	93.80	93.58	90.86
CIFAR100	74.27	74.21	74.19	73.70	66.32
STL10	77.59	77.65	77.70	77.59	73.40
AlexNet	55.70/78.42	55.66/78.41	55.54/78.39	54.17/77.29	18.19/36.25
VGG16	70.44/89.43	70.45/89.43	70.44/89.33	69.99/89.17	53.33/76.32
VGG19	71.36/89.94	71.35/89.93	71.34/89.88	70.88/89.62	56.00/78.62
ResNet18	68.63/88.31	68.62/88.33	68.49/88.25	66.80/87.20	19.14/36.49
ResNet34	72.50/90.86	72.46/90.82	72.45/90.85	71.47/90.00	32.25/55.71
ResNet50	74.98/92.17	74.94/92.12	74.91/92.09	72.54/90.44	2.43/5.36
ResNet101	76.69/93.30	76.66/93.25	76.22/92.90	65.69/79.54	1.41/1.18
ResNet152	77.55/93.59	77.51/93.62	77.40/93.54	74.95/92.46	9.29/16.75
SqueezeNetV0	56.73/79.39	56.75/79.40	56.70/79.27	53.93/77.04	14.21/29.74
SqueezeNetV1	56.52/79.13	56.52/79.15	56.24/79.03	54.56/77.33	17.10/32.46
InceptionV3	76.41/92.78	76.43/92.71	76.44/92.73	73.67/91.34	1.50/4.82



Ecosystem

- Probabilistic Programming



github.com/probtorch/probtorch

<http://pyro.ai/>



Ecosystem

• Gaussian Processes

GPyTorch (Alpha Release)

[build](#) passing

GPyTorch is a Gaussian Process library, implemented using PyTorch. It is designed for creating flexible and modular Gaussian Process models with ease, so that you don't have to be an expert to use GPs.

This package is currently under development, and is likely to change. Some things you can do right now:

- Simple GP regression ([example here](#))
- Simple GP classification ([example here](#))
- Multitask GP regression ([example here](#))
- Scalable GP regression using kernel interpolation ([example here](#))
- Scalable GP classification using kernel interpolation ([example here](#))
- Deep kernel learning ([example here](#))
- And ([more!](#))

<https://github.com/cornellius-gp/gpytorch>



Ecosystem

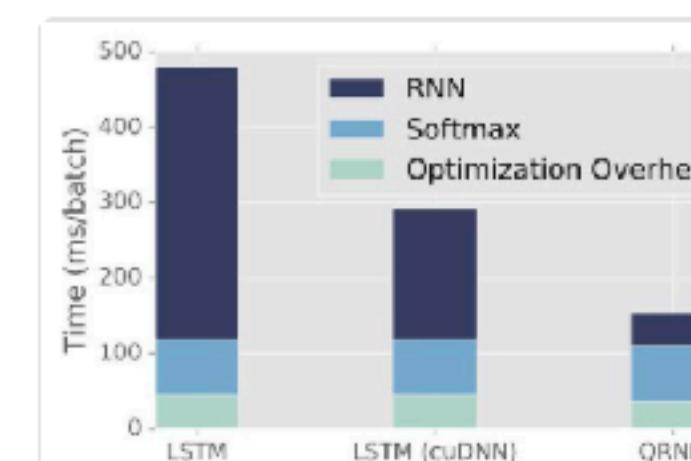
- QRNN - 2x to 17x faster than LSTM

<https://github.com/salesforce/pytorch-qrnn>

 **Smerity**
@Smerity

Following

We're releasing [@PyTorch](#)-QRNN, 2-17x faster than NVIDIA's cuDNN LSTM. Speed thanks to 50 lines of CUDA via CuPy.
github.com/salesforce/pyt...



	Sequence length	32	64	128	256	512
Batch size	8	5.5x	8.8x	11.0x	12.4x	16.9x
	16	5.5x	6.7x	7.8x	8.3x	10.8x
	32	4.2x	4.5x	4.9x	4.9x	6.4x
	64	3.0x	3.0x	3.0x	3.0x	3.7x
	128	2.1x	1.9x	2.0x	2.0x	2.4x
	256	1.4x	1.4x	1.3x	1.3x	1.3x

2:30 PM - 9 Oct 2017

316 Retweets 848 Likes

Richard, James Bradbury, Nitish and Caiming Xiong

15 316 848



Ecosystem

- CycleGAN, pix2pix

<https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix>



Ecosystem

• Machine Translation

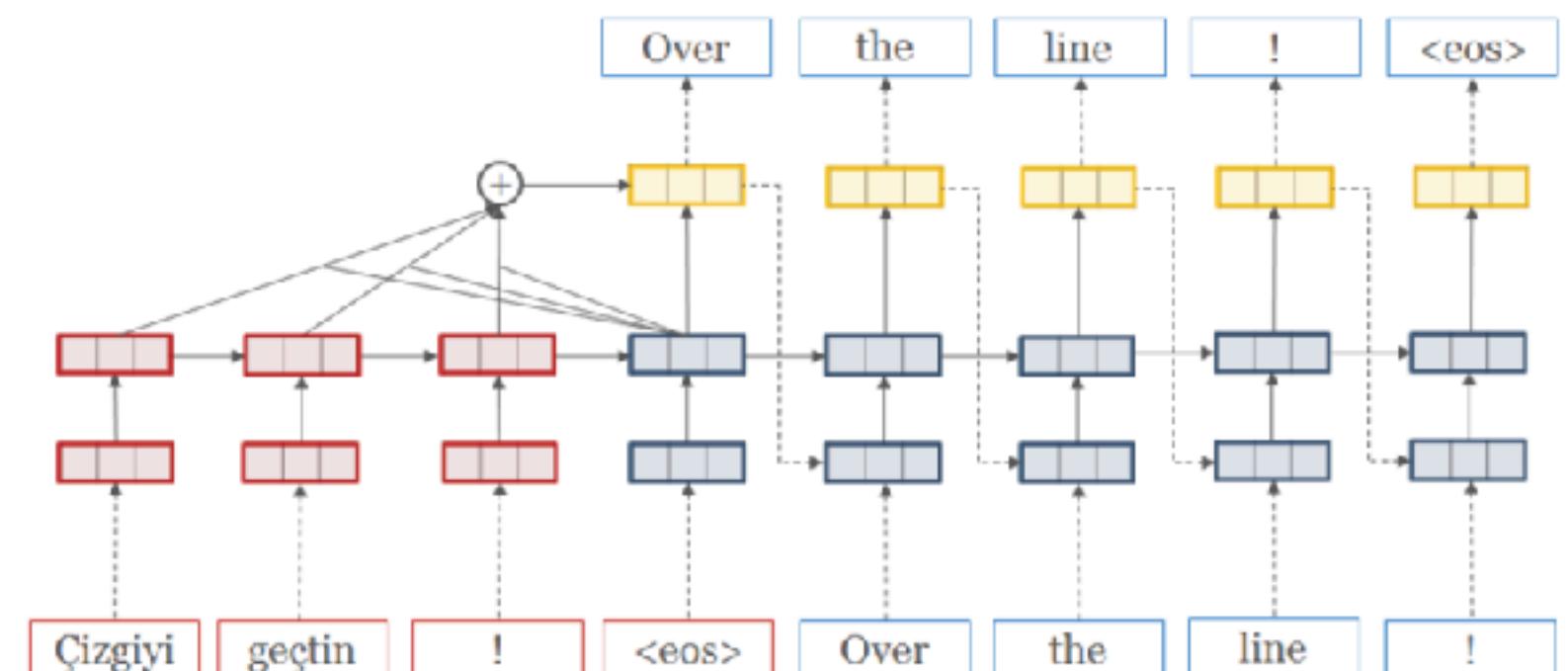
OpenNMT-py: Open-Source Neural Machine Translation

build passing

This is a Pytorch port of [OpenNMT](#), an open-source (MIT) neural machine translation system. It is designed to be research friendly to try out new ideas in translation, summary, image-to-text, morphology, and many other domains.

Codebase is relatively stable, but PyTorch is still evolving. We currently recommend forking if you need to have stable code.

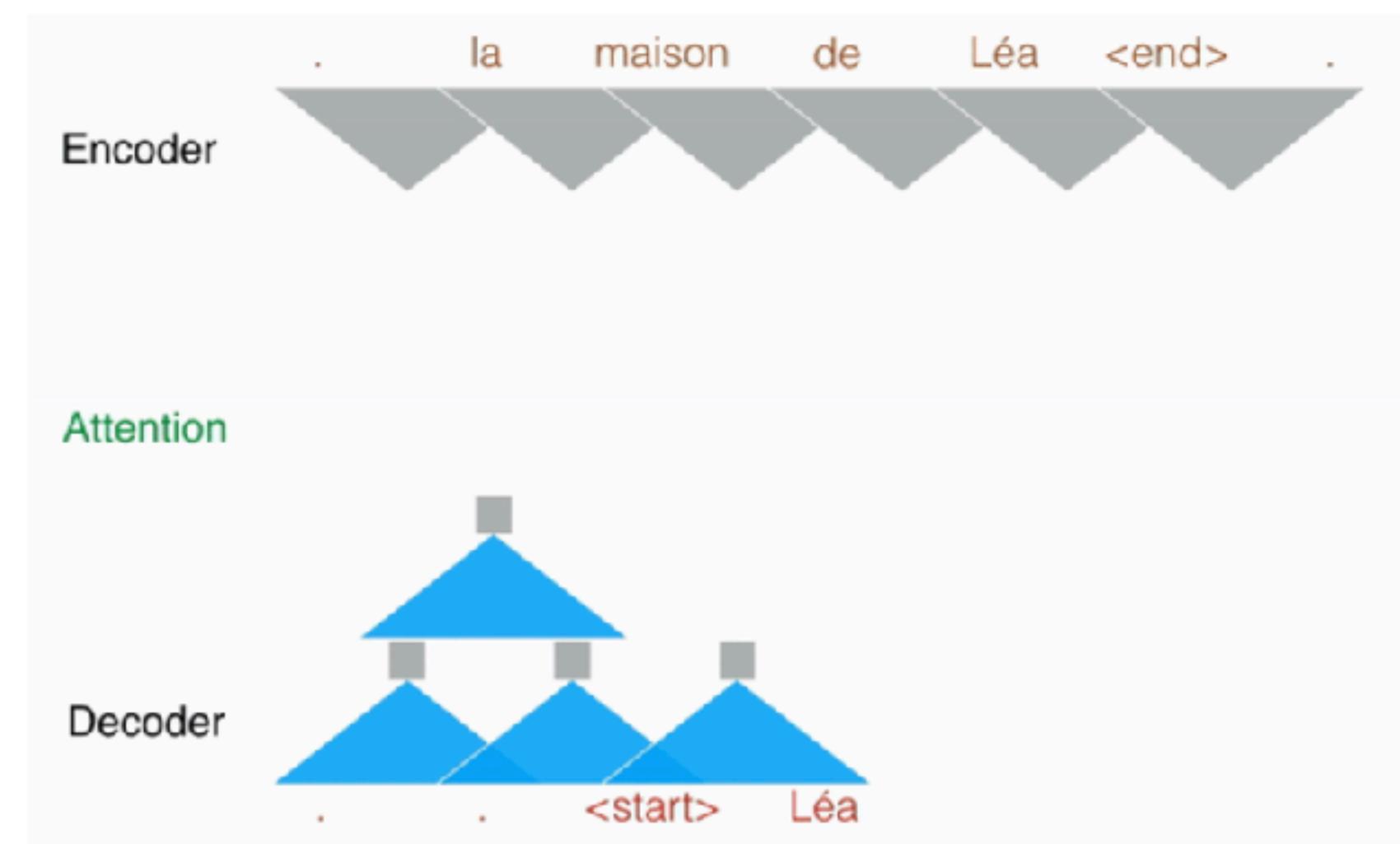
OpenNMT-py is run as a collaborative open-source project. It is maintained by [Sasha Rush](#) (Cambridge, MA), [Ben Peters](#) (Saarbrücken), and [Jianyu Zhan](#) (Shenzhen). The original code was written by [Adam Lerer](#) (NYC). We love contributions. Please consult the Issues page for any [Contributions Welcome](#) tagged post.



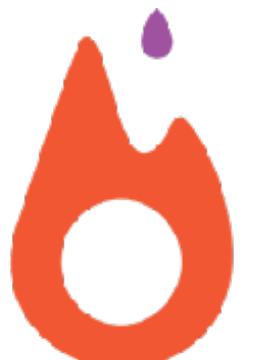
<https://github.com/OpenNMT/OpenNMT-py>

FAIR Sequence-to-Sequence Toolkit (PyTorch)

This is a PyTorch version of [fairseq](#), a sequence-to-sequence learning toolkit from Facebook AI Research. The original authors of this reimplementation are (in no particular order) Sergey Edunov, Myle Ott, and Sam Gross. The toolkit implements the fully convolutional model described in [Convolutional Sequence to Sequence Learning](#) and features multi-GPU training on a single machine as well as fast beam search generation on both CPU and GPU. We provide pre-trained models for English to French and English to German translation.



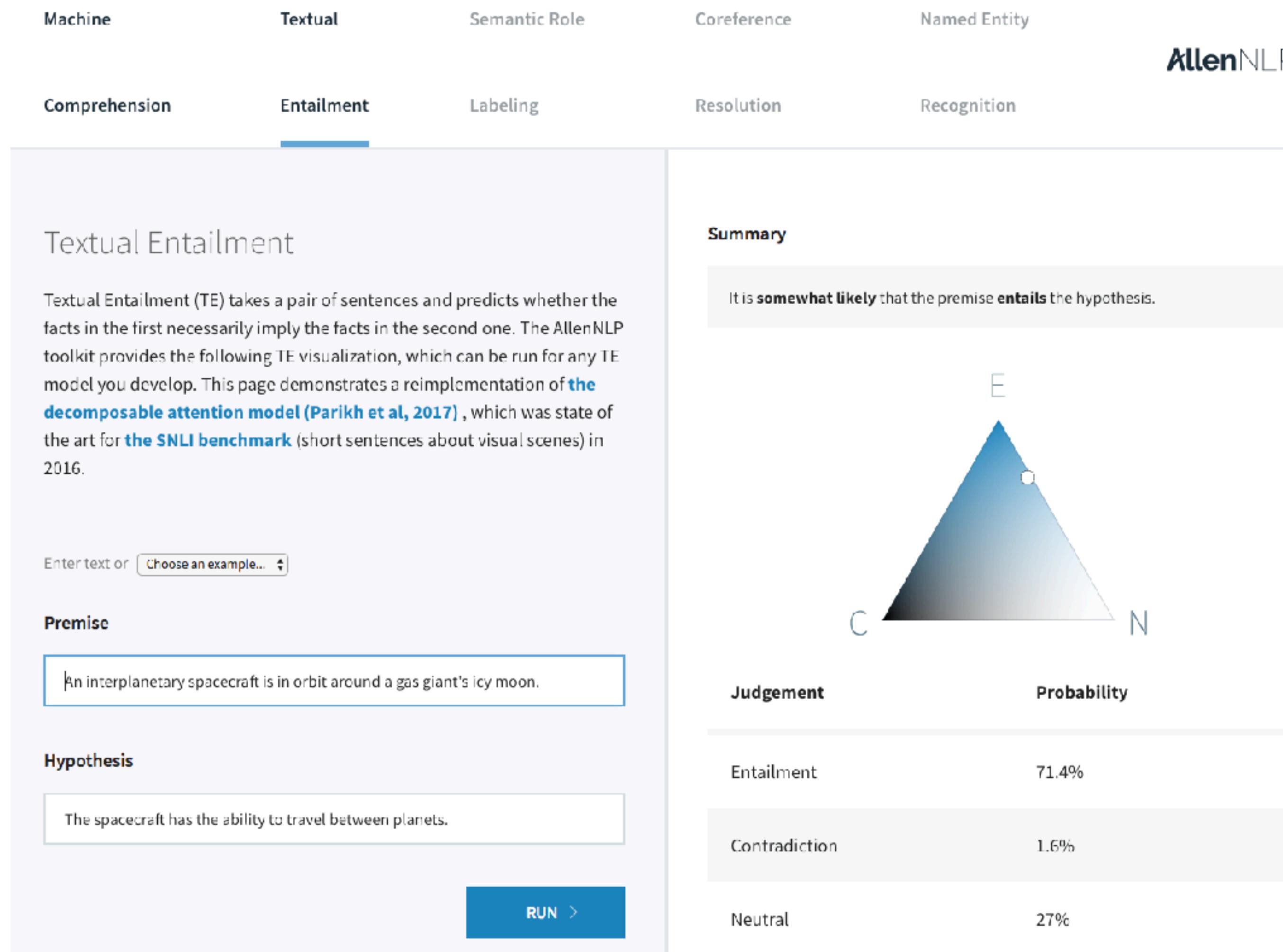
<https://github.com/facebookresearch/fairseq-py>



Ecosystem

- AllenNLP

<http://allennlp.org/>

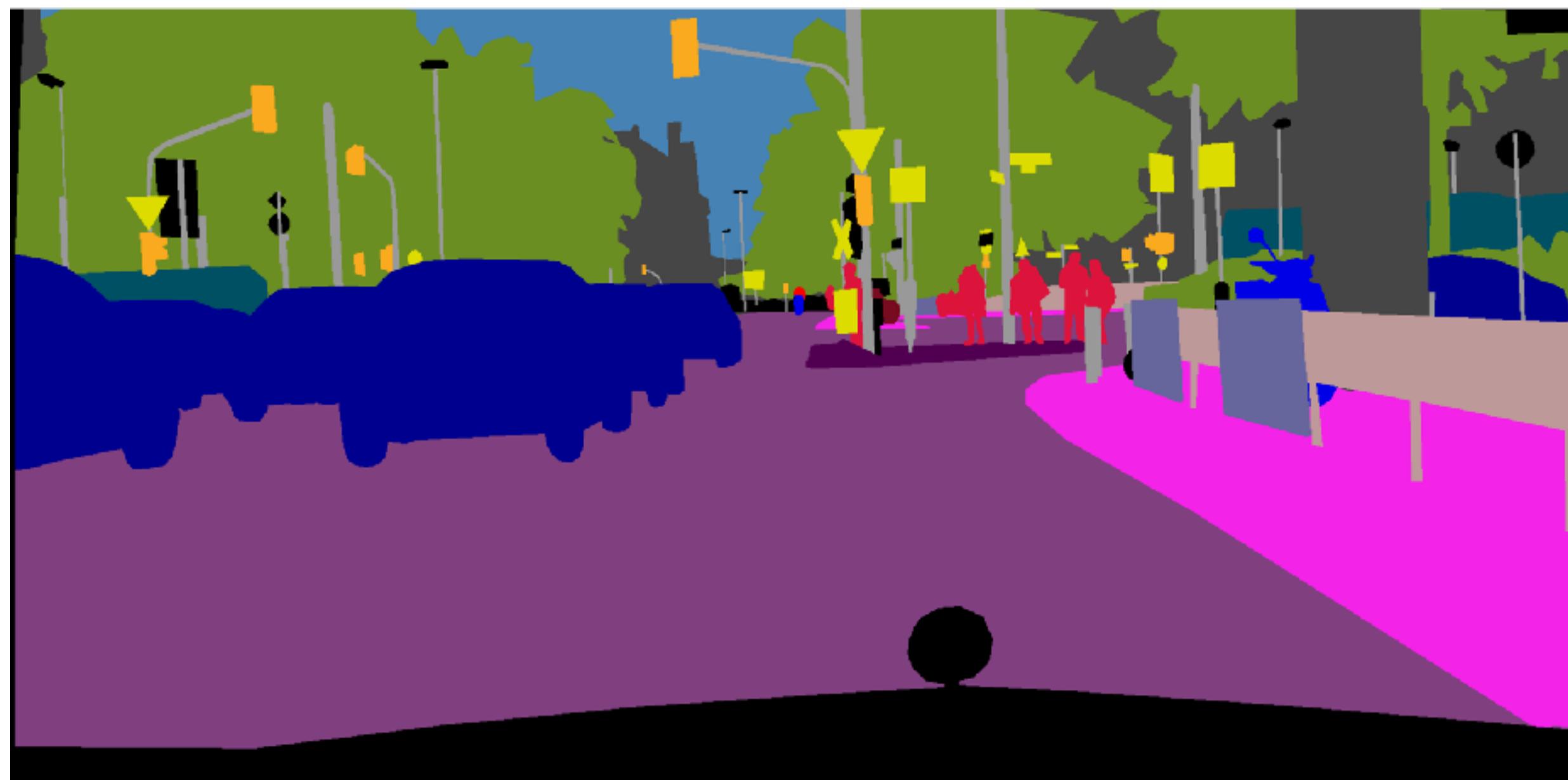


Ecosystem

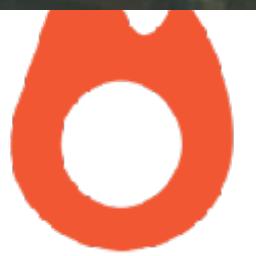
- Pix2PixHD

<https://github.com/NVIDIA/pix2pixHD>

Input labels



Synthesized image



Ecosystem

- Sentiment Discovery

<https://github.com/NVIDIA/sentiment-discovery>

Exquisitely acted and masterfully if preciously interwoven... (the film) addresses in a fascinating, intelligent manner the intermingling of race, politics and local commerce.

Thrilling, provocative and darkly funny, this timely sci-fi mystery works on so many different levels that it not only invites, it demands repeated viewings.

What could and should have been biting and droll is instead a tepid waste of time and talent.

A dreary, incoherent, self-indulgent mess of a movie in which a bunch of pompous windbags drone on inanely for two hours... a cacophony of pretentious, meaningless prattle.



Ecosystem

- FlowNet2: Optical Flow Estimation with Deep Networks

<https://github.com/NVIDIA/flownet2-pytorch>



With ❤ from



<http://pytorch.org>

