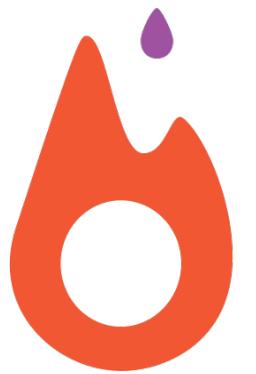


Researcher Edition

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.

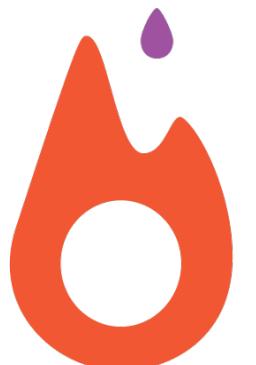
```
from __future__ import print_function  
import torch
```

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:

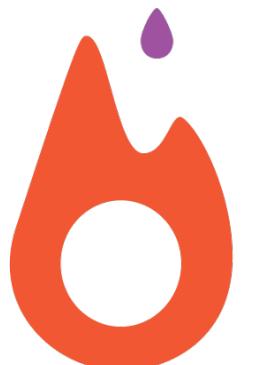
```
0.2598  0.7231  0.8534
0.3928  0.1244  0.5110
0.5476  0.2700  0.5856
0.7288  0.9455  0.8749
0.6663  0.8230  0.2713
[torch.FloatTensor of size 5x3]
```

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)
```

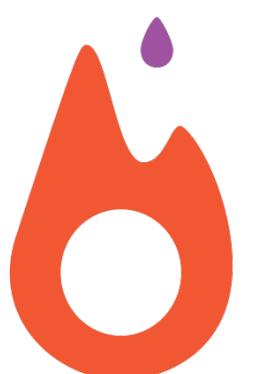
Out:

```
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:

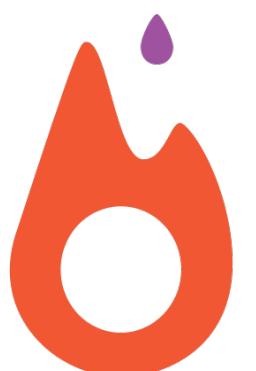
```
1  
1  
1  
1  
1  
[torch.FloatTensor of size 5]
```

**Zero memory-copy
very efficient**

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

Out:

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



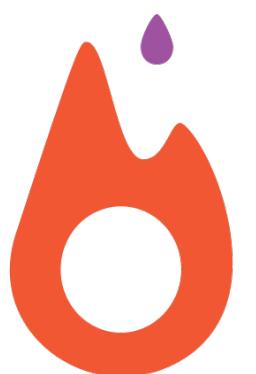
NumPy bridge

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

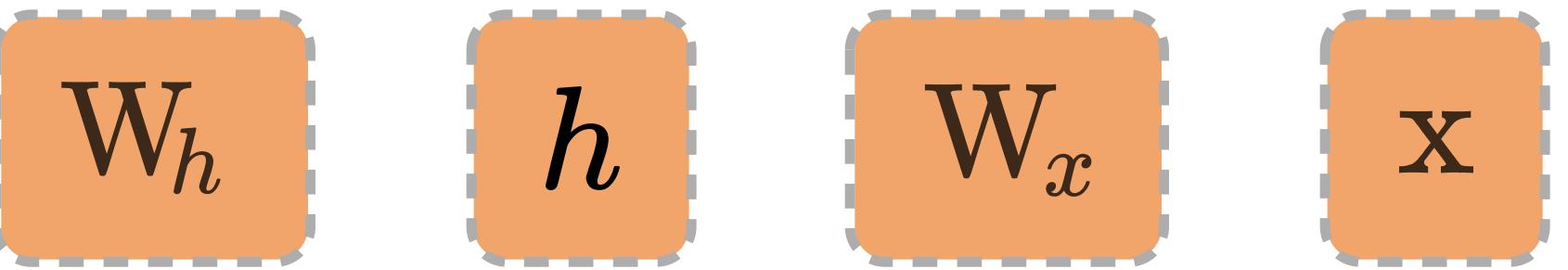
for deep learning and reinforcement learning



PyTorch Autograd

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

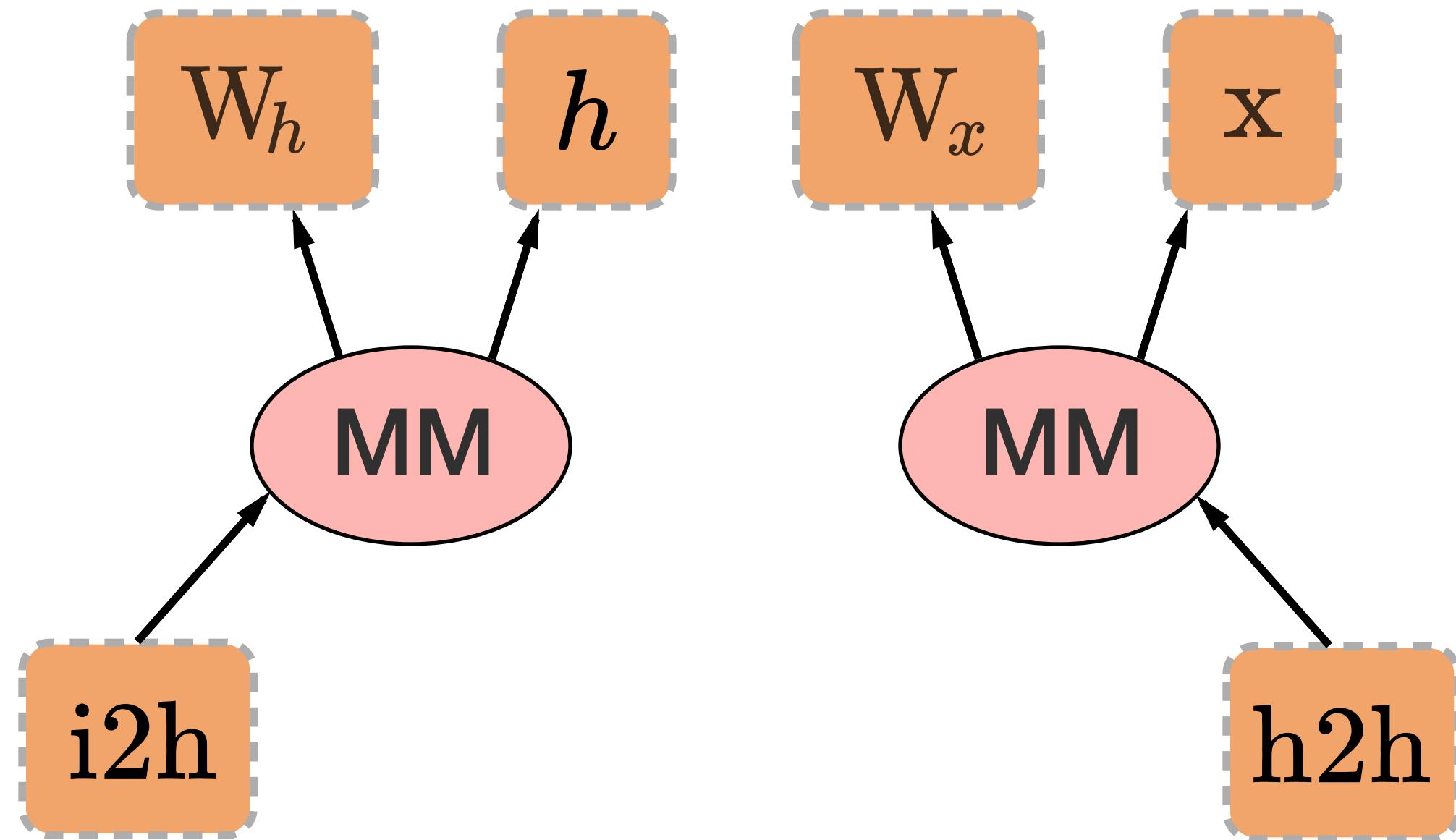
PyTorch Autograd



```
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))
```

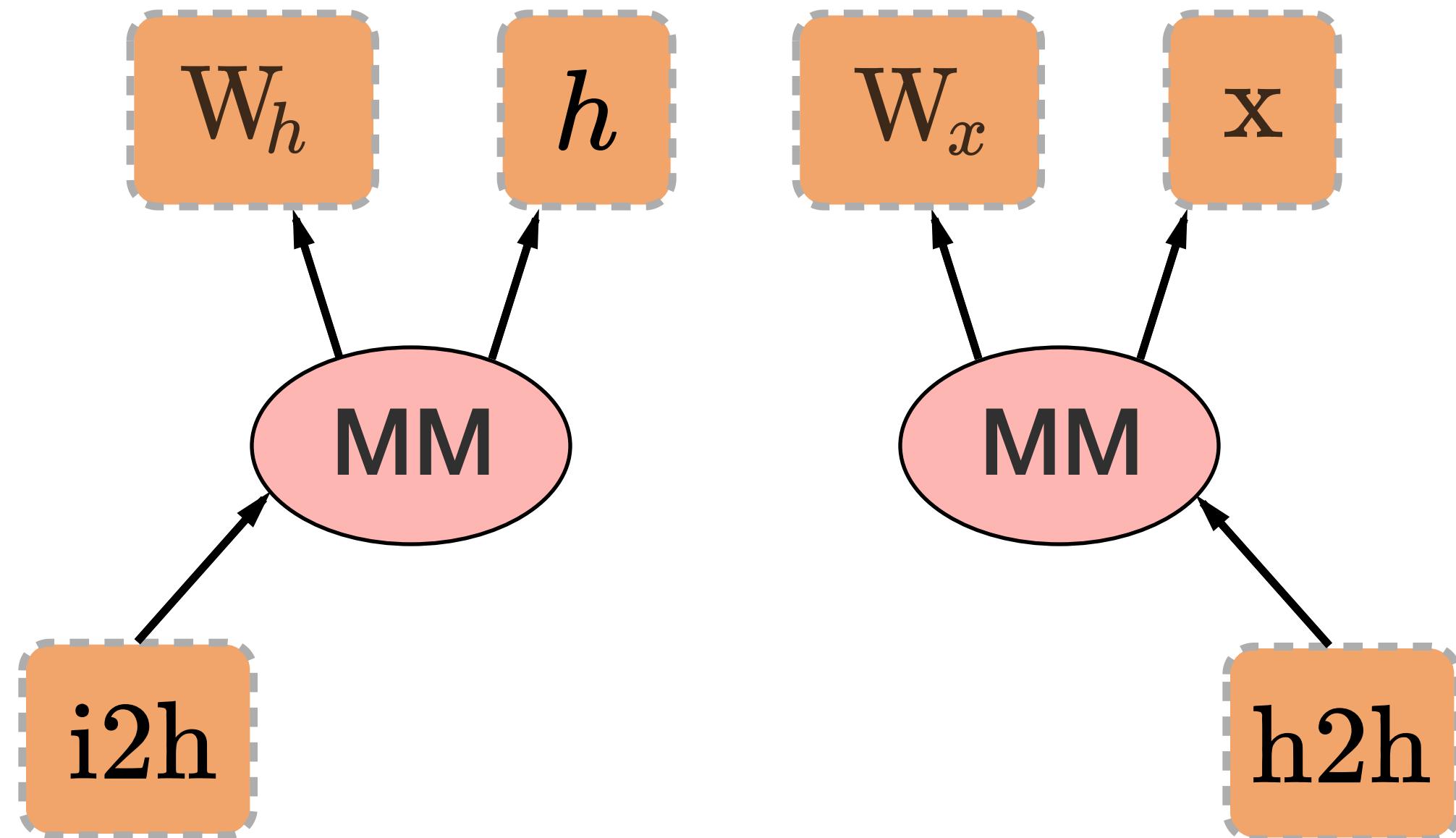
PyTorch Autograd

```
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())
```



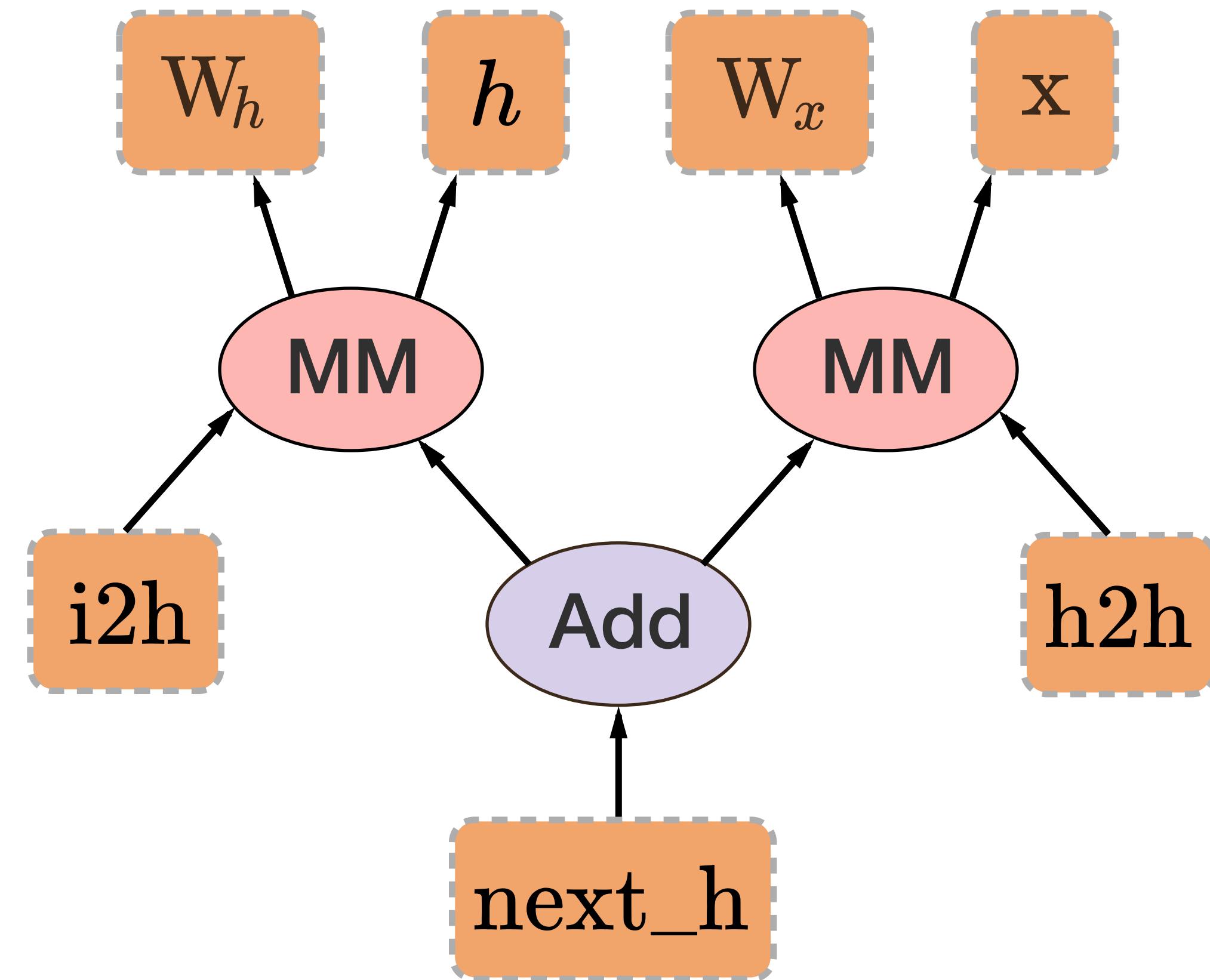
PyTorch Autograd

```
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
```



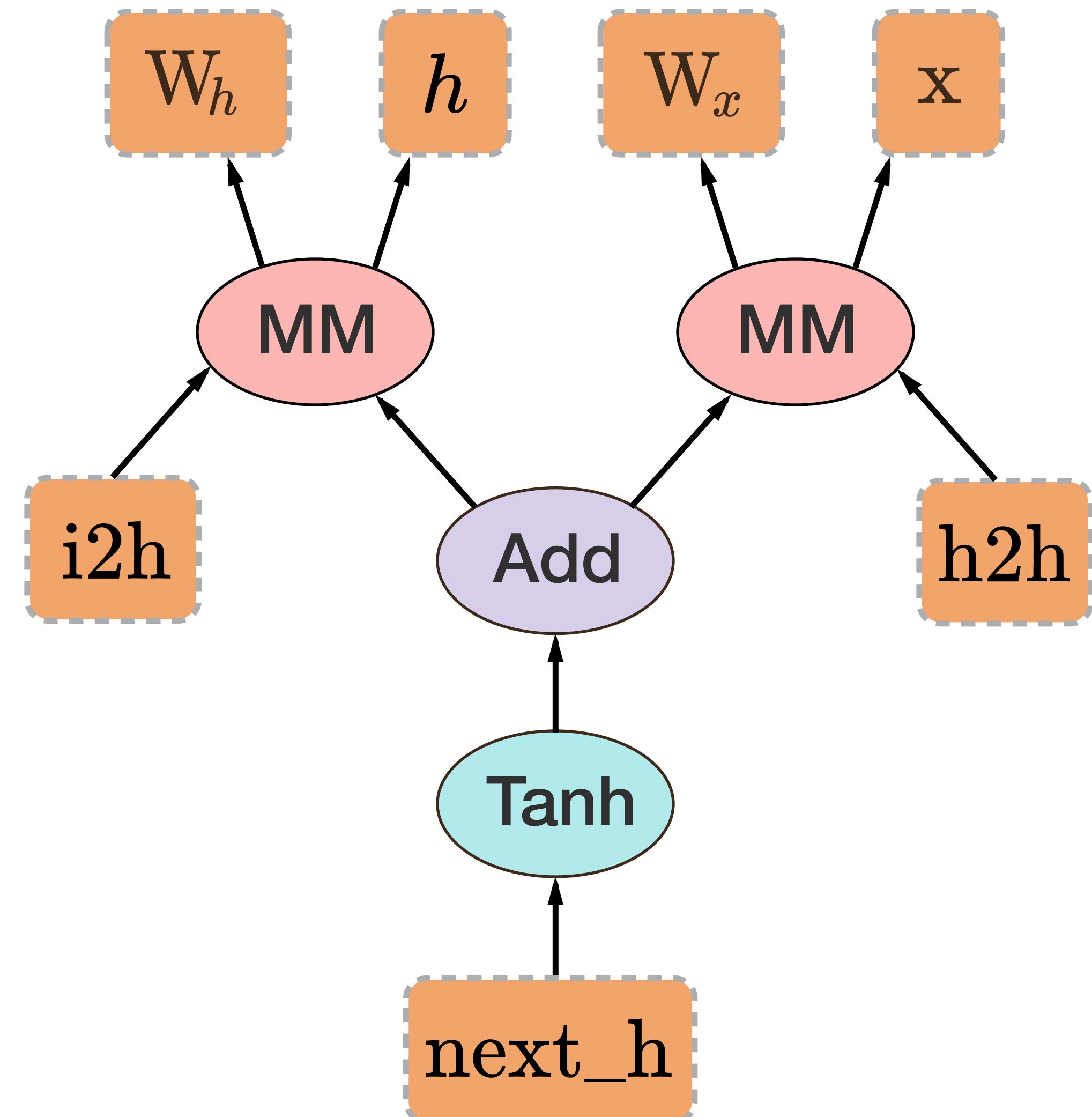
PyTorch Autograd

```
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
```



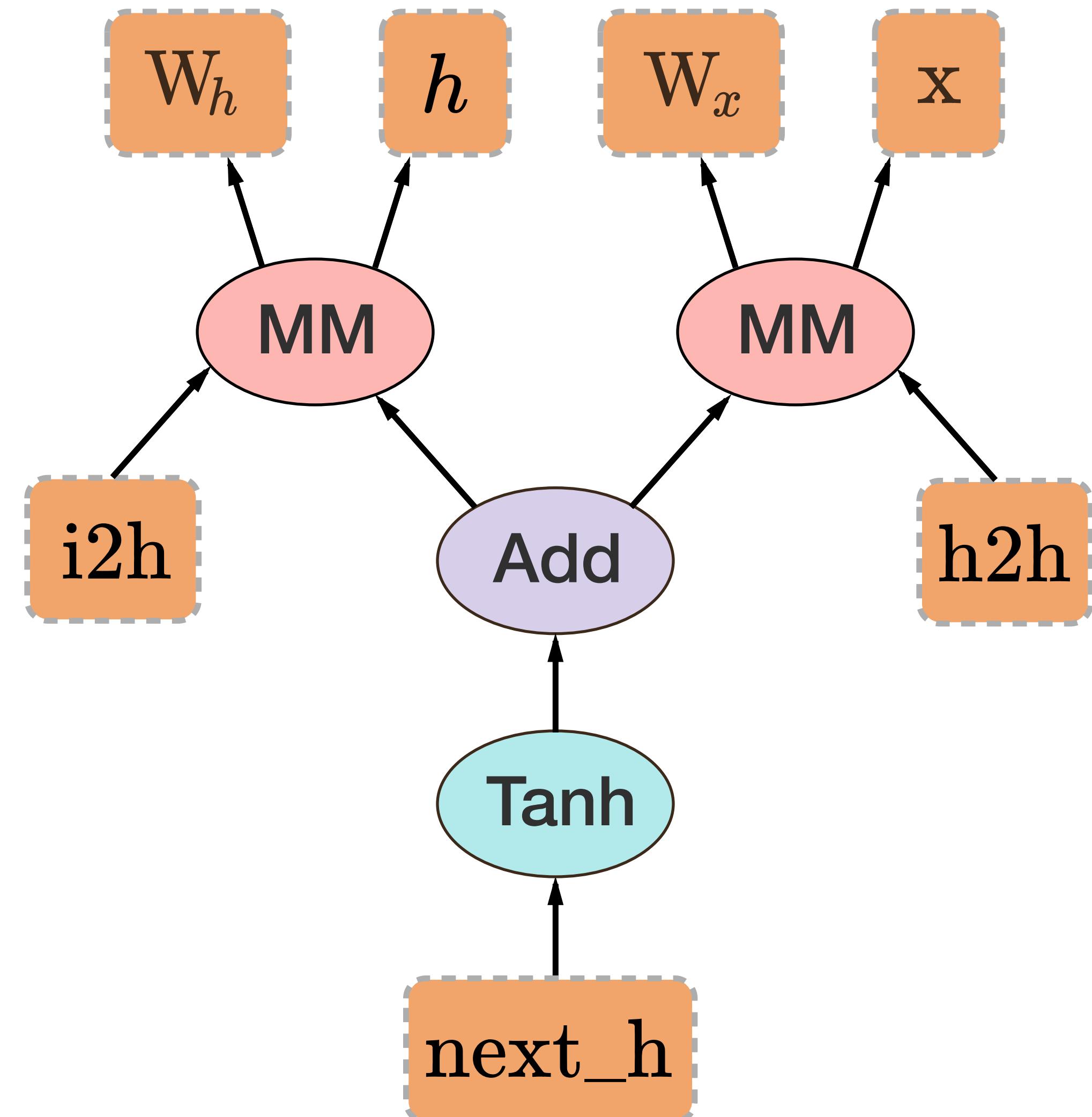
PyTorch Autograd

```
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

```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 5x5 square convolution
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square you can only specify a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

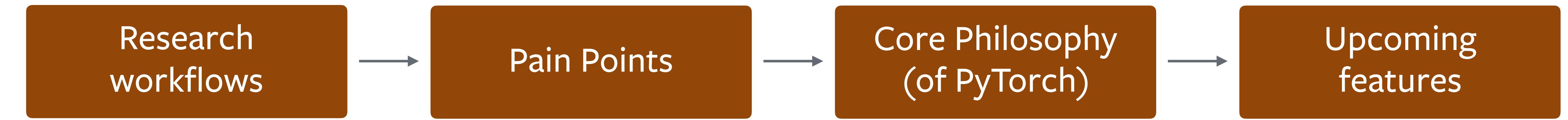
    def num_flat_features(self, x):
        size = x.size()[1:]  # all dimensions except the batch dimension
        num_features = 1
        for s in size:
            num_features *= s
        return num_features

net = Net()
```

Optimization package

SGD, Adagrad, RMSProp, LBFGS, etc.

```
for input, target in dataset:  
    optimizer.zero_grad()  
    output = model(input)  
    loss = loss_fn(output, target)  
    loss.backward()  
    optimizer.step()
```



what do they look
like in
the deep learning space

how does PyTorch
deal with them?



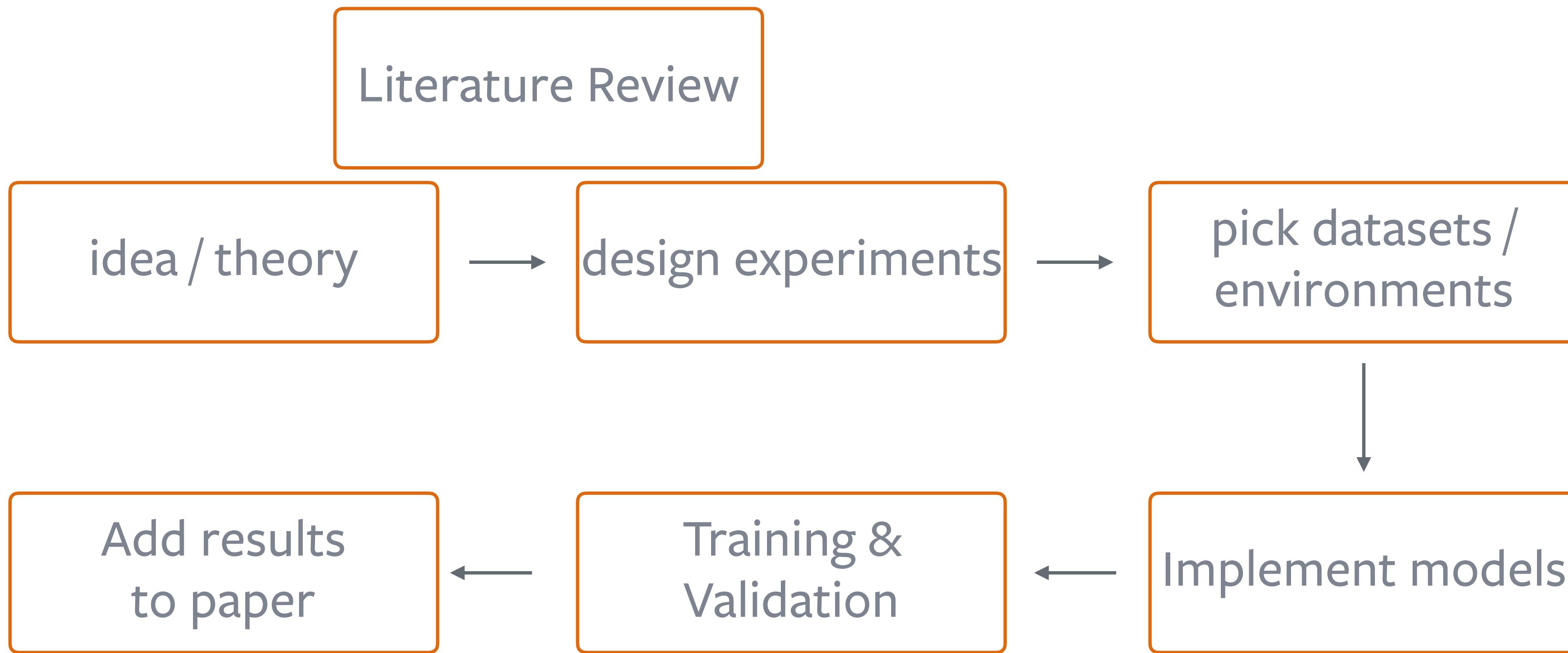
Research Workflows



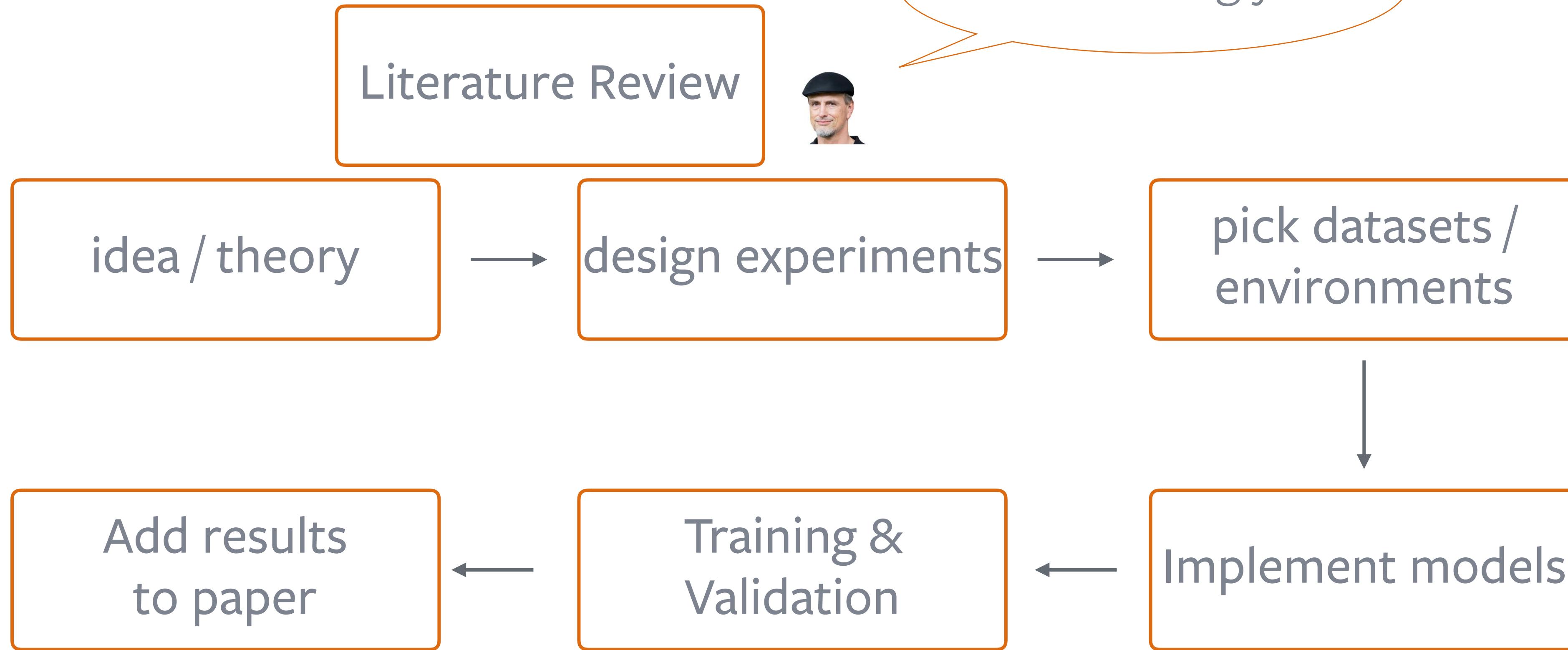
Research Workflow



Research Workflow



Research Workflow



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

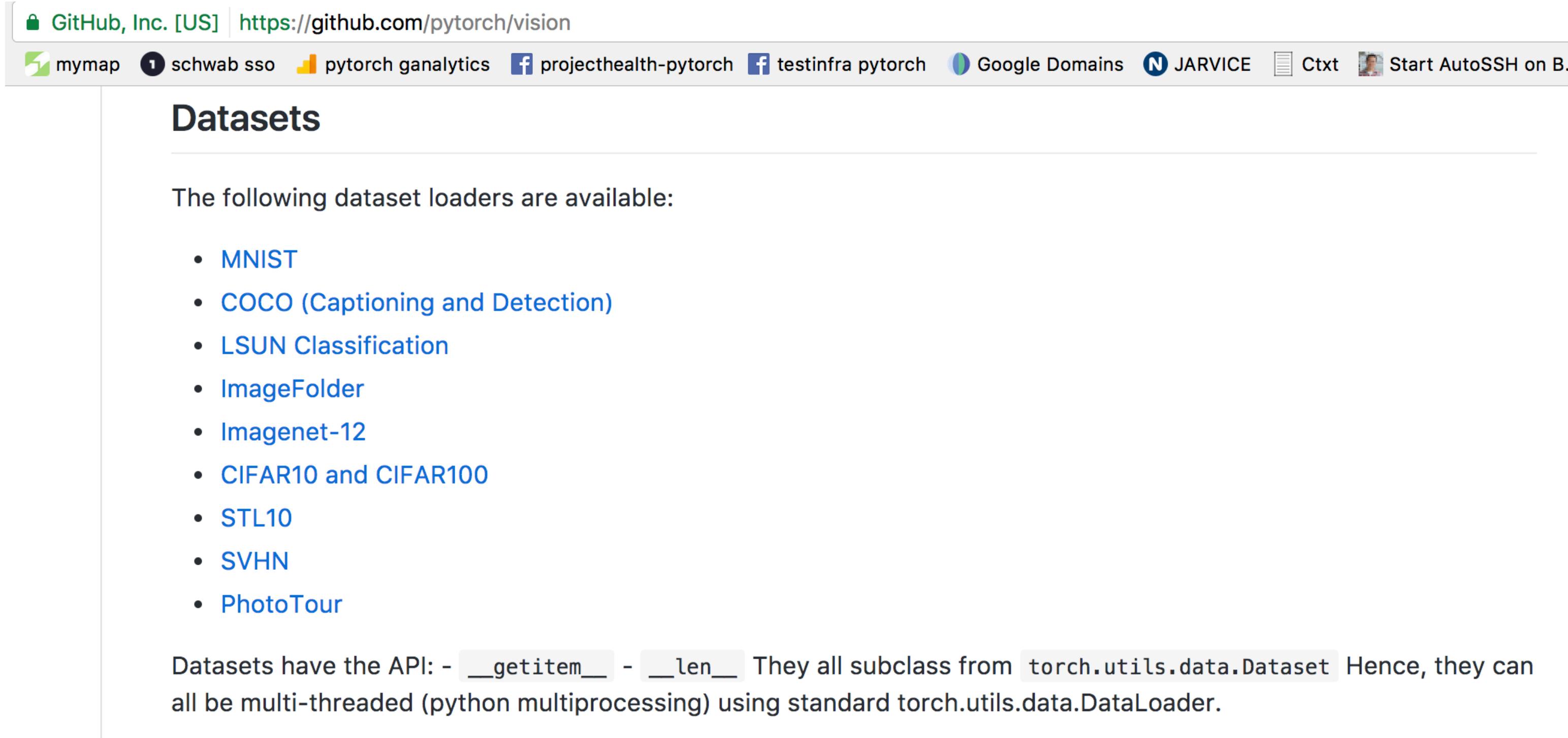
- every dataset is slightly differently formatted
- have to be preprocessed and normalized differently
- need a multithreaded Data loader to feed GPUs fast enough



Writing Data Loaders

PyTorch solution:

- share data loaders across the community!



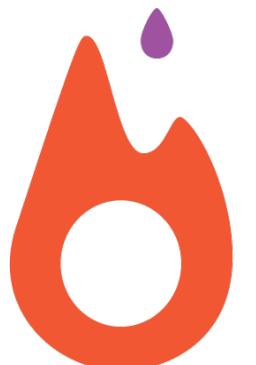
The screenshot shows a GitHub repository page for "pytorch/vision". The title bar includes the GitHub logo, URL (<https://github.com/pytorch/vision>), and various GitHub team and organization icons. The main content area is titled "Datasets" and contains a list of available dataset loaders. Below the list, a note explains the API for datasets.

Datasets

The following dataset loaders are available:

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

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

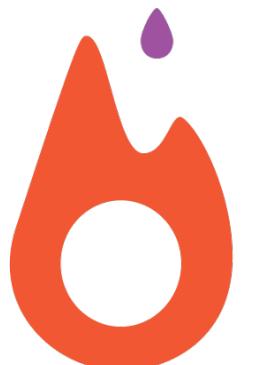
PyTorch solution:

- share data loaders across the community!

The screenshot shows the GitHub repository page for `pytorch/text`. The top navigation bar includes links for GitHub, Inc. [US], the repository URL, and various organization members and projects. Below the header, a section titled "This repository consists of:" lists three main components: `torchtext.data`, `torchtext.datasets`, and `torchtext.models`.

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 p
situation is not the same as vision, where people can download a pretrained
make it useful for other tasks -- it might make more sense to leave NLP m



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



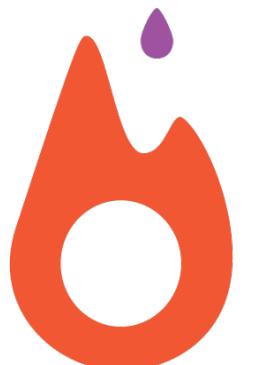
ParlAI

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]

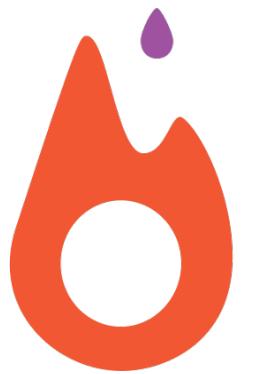
    path = coco.loadImgs(img_id)[0]['file_name']

    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)
```



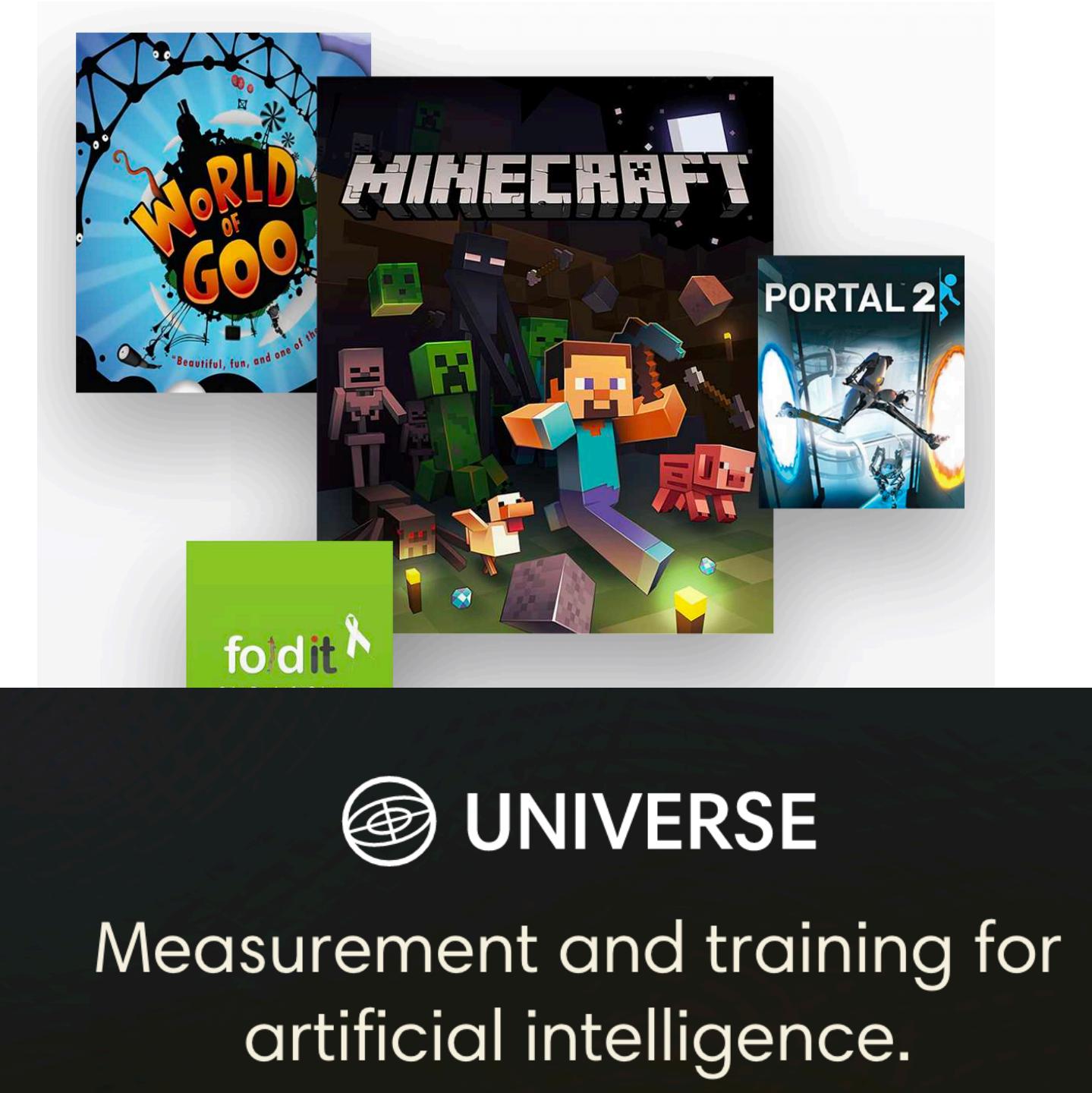
Interfacing with environments



Cars



Video games



Internet



Interfacing with environments



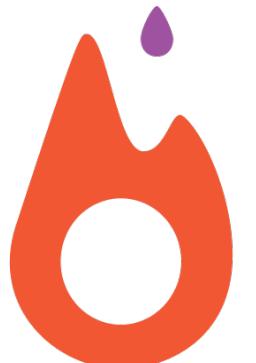
Cars



Video games

The collage includes covers for "World of Goo" (a puzzle game), "Minecraft" (a block-based沙盒游戏), and "Portal 2" (a physics-based puzzle game). Below the collage is a green "foldit" logo with a ribbon icon. A dark rectangular banner at the bottom right contains the "UNIVERSE" logo (a stylized globe icon) and the text "Measurement and training for artificial intelligence."

Pretty much every environment provides a Python API



Interfacing with environments



Cars



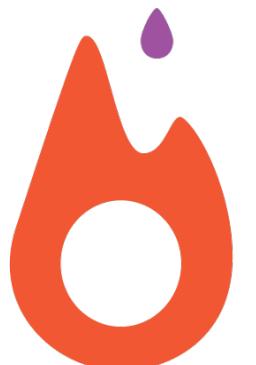
Video games

foldit

UNIVERSE

Measurement and training for
artificial intelligence.

Natively interact with the environment directly



Debugging

- PyTorch is a Python extension



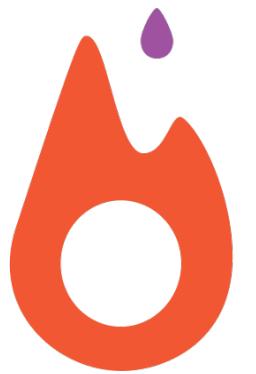
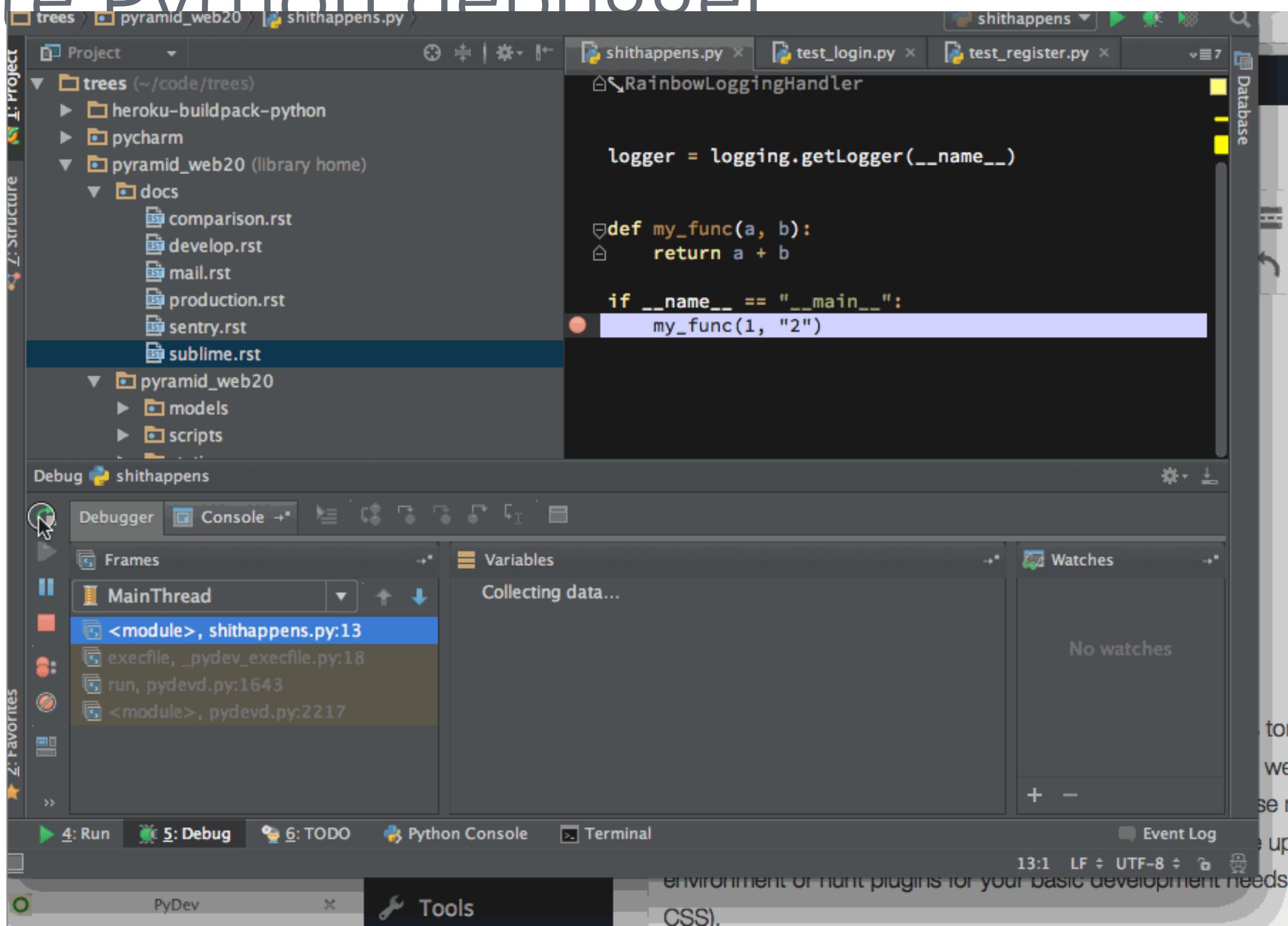
Debugging

- 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)

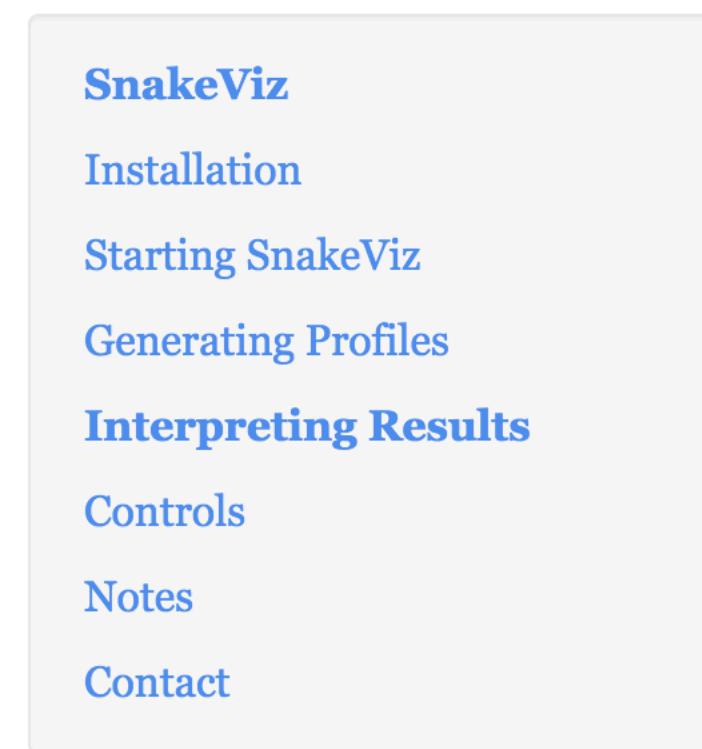


Identifying bottlenecks

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

SNAKEVIZ

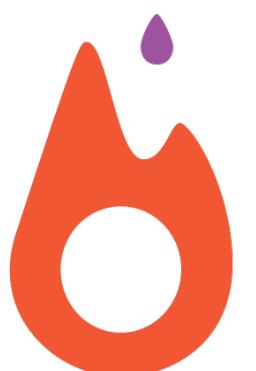
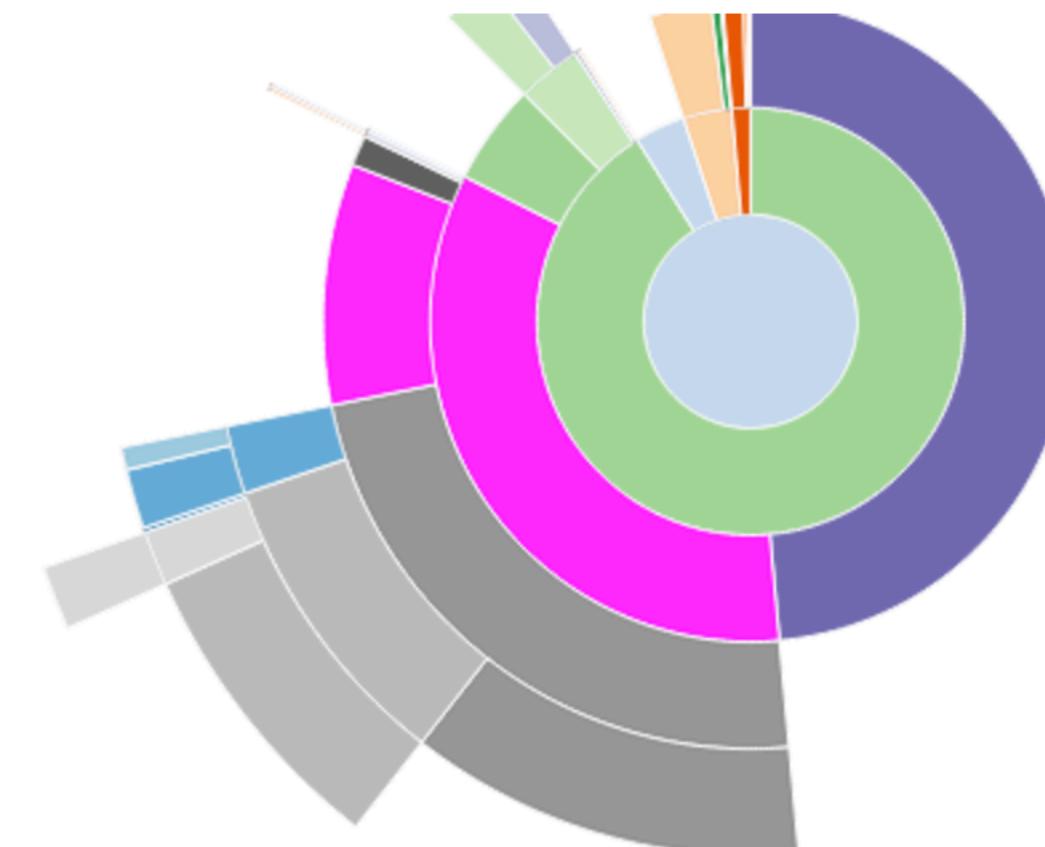
← PREVIOUS NEXT → G



FUNCTION INFO

Placing your cursor over an arc will highlight that arc and any other visible instances of the same function call. It also display 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/en
vs/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



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 ecosystem
- Including SciPy



Brandon Amos @brandondamos · Jan 18

Why PyTorch's layer creation is powerful: Here's my layer that solves an optimization problem with a primal-dual interior point method.

```
def forward_single(input_i, Q, G, A, b, h, U_Q, U_S, R):
    nineq, nz = G.size()
    neq = A.size(0) if A.ndimension() > 0 else 0

    d = torch.ones(nineq).type_as(Q)
    nb = -b if b is not None else None
    factor_kkt(U_S, R, d, neq)
    x, s, z, y = solve_kkt(U_Q, d, G, A, U_S, input_i, torch.zeros(nineq).type_as(Q),
                           -h, nb, neq, nz)

    if torch.min(s) < 0:
        s -= torch.min(s) - 1
    if torch.min(z) < 0:
        z -= torch.min(z) - 1

    for i in range(20):
        rx = (torch.mv(A.t(),y) if neq > 0 else 0.) + \
             torch.mv(G.t(), z) + torch.mv(Q,x) + input_i
        rs = z
        ry = torch.mv(G,x) + s - h
        rz = torch.mv(A,x) - b if neq > 0 else torch.Tensor([0.])
        mu = torch.dot(s,z)/nineq
        pri_resid = torch.norm(ry) + torch.norm(rz)
        dual_resid = torch.norm(rx)
        d = z/s

        factor_kkt(U_S, R, d, neq)
        dx_aff, ds_aff, dz_aff, dy_aff = solve_kkt(U_Q, d, G, A, U_S,
                                                     rx, rs, ry, rz, neq, nz)

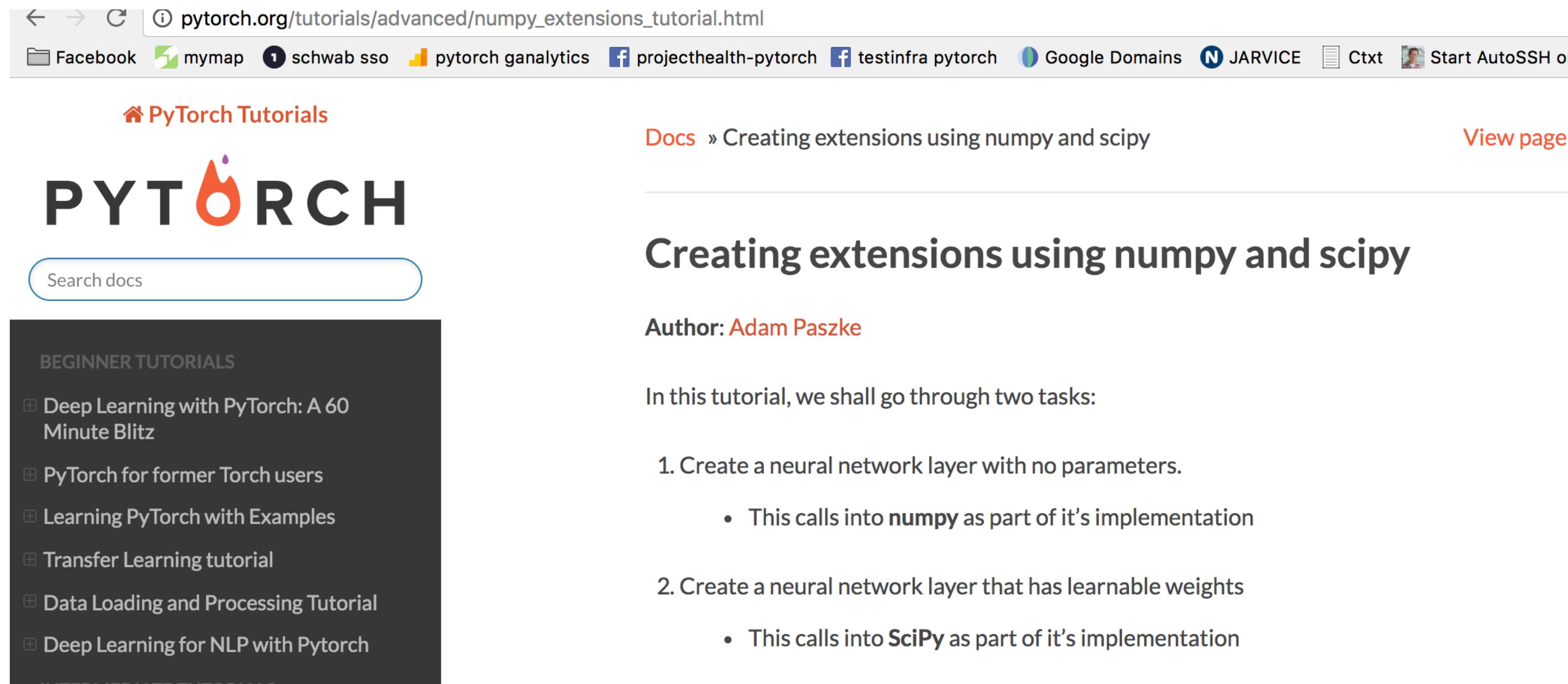
        # compute centering directions
        alpha = min(min(get_step(z,dz_aff), get_step(s, ds_aff)), 1.0)
        sig = (torch.dot(s + alpha*ds_aff, z + alpha*dz_aff)/(torch.dot(s,z)))**3
        dx_cor, ds_cor, dz_cor, dy_cor = solve_kkt(
            U_Q, d, G, A, U_S, torch.zeros(nz).type_as(Q),
            (-mu*sig*torch.ones(nineq).type_as(Q) + ds_aff*dz_aff)/s,
            torch.zeros(nineq).type_as(Q), torch.zeros(neq).type_as(Q), neq, nz)

        dx = dx_aff + dx_cor
        ds = ds_aff + ds_cor
        dz = dz_aff + dz_cor
        dy = dy_aff + dy_cor if neq > 0 else None
```



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 "Author: Adam Paszke". The main content describes two tasks: creating a neural network layer with no parameters (calling into numpy) and creating one with learnable weights (calling into SciPy). The left sidebar lists "BEGINNER TUTORIALS" including "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".

← → ⌂ ⓘ pytorch.org/tutorials/advanced/numpy_extensions_tutorial.html

Facebook mymap schwab sso pytorch ganalytics projecthealth-pytorch testinfra pytorch Google Domains JARVICE Ctxt Start AutoSSH or

PyTorch Tutorials Docs » Creating extensions using numpy and scipy View page

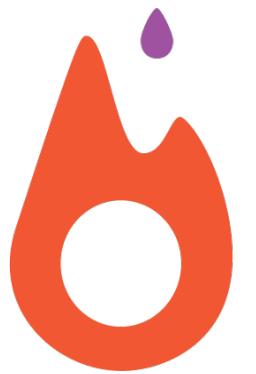
PYTORCH

Search docs

BEGINNER TUTORIALS

- 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
- Deep Learning for NLP with Pytorch

INTERMEDIATE TUTORIALS



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



Linear style of programming

- PyTorch is an imperative / eager computational toolkit



Linear style of programming

- PyTorch is an imperative / eager computing style

```
1 import torch
2 from torch.autograd import Variable
3
4 trX = torch.linspace(-1, 1, 101)
5 trY = 2 * trX + torch.randn(*trX.size()) * 0.33
6
7 w = Variable(trX.new([0.0]), requires_grad=True)
8
9 for i in range(100):
10     for (x, y) in zip(trX, trY):
11         X = Variable(x)
12         Y = Variable(y)
13         print(Y)
14
15         y_model = X * w.expand_as(X)
16         cost = (Y - Y_model) ** 2
17         cost.backward(torch.ones(*cost.size()))
18
19         w.data = w.data + 0.01 * w.grad.data
20
21     print(w)
```



Linear style of programming

- PyTorch is an imperative / eager computational toolkit
 - Not unique to PyTorch



Linear style of programming

- PyTorch is an imperative / eager computational toolkit
 - Not unique to PyTorch
 - Chainer, Dynet, MXNet-Imperative, TensorFlow-imperative, TensorFlow-eager, etc.



Linear style of programming

- PyTorch is an imperative / eager computational toolkit
 - Not unique to PyTorch
 - Chainer, Dynet, MXNet-Imperative, TensorFlow-imperative, TensorFlow-eager, etc.
 - Least overhead, designed with this in mind
 - 20 to 30 microseconds overhead per node creation
 - vs several milliseconds / seconds in other options



Go Through an example



The Philosophy of PyTorch



The Philosophy of PyTorch

- Stay out of the way
- Cater to the impatient
- Promote linear code-flow
- Full interop with the Python ecosystem
- Be as fast as anything else



Upcoming features



Distributed PyTorch

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



Higher order derivatives

- `grad(grad(grad(grad(grad(torch.norm(x)))))))`



Higher order derivatives

- `grad(grad(grad(grad(grad(torch.norm(x)))))))`
- Useful to implement crazy ideas



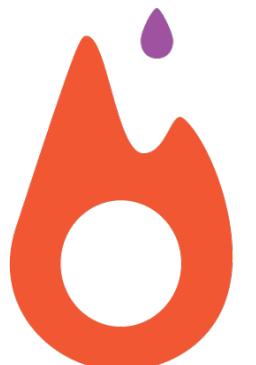
Broadcasting and Advanced Indexing

- Numpy-style broadcasting
- Numpy-style indexing that covers advanced cases



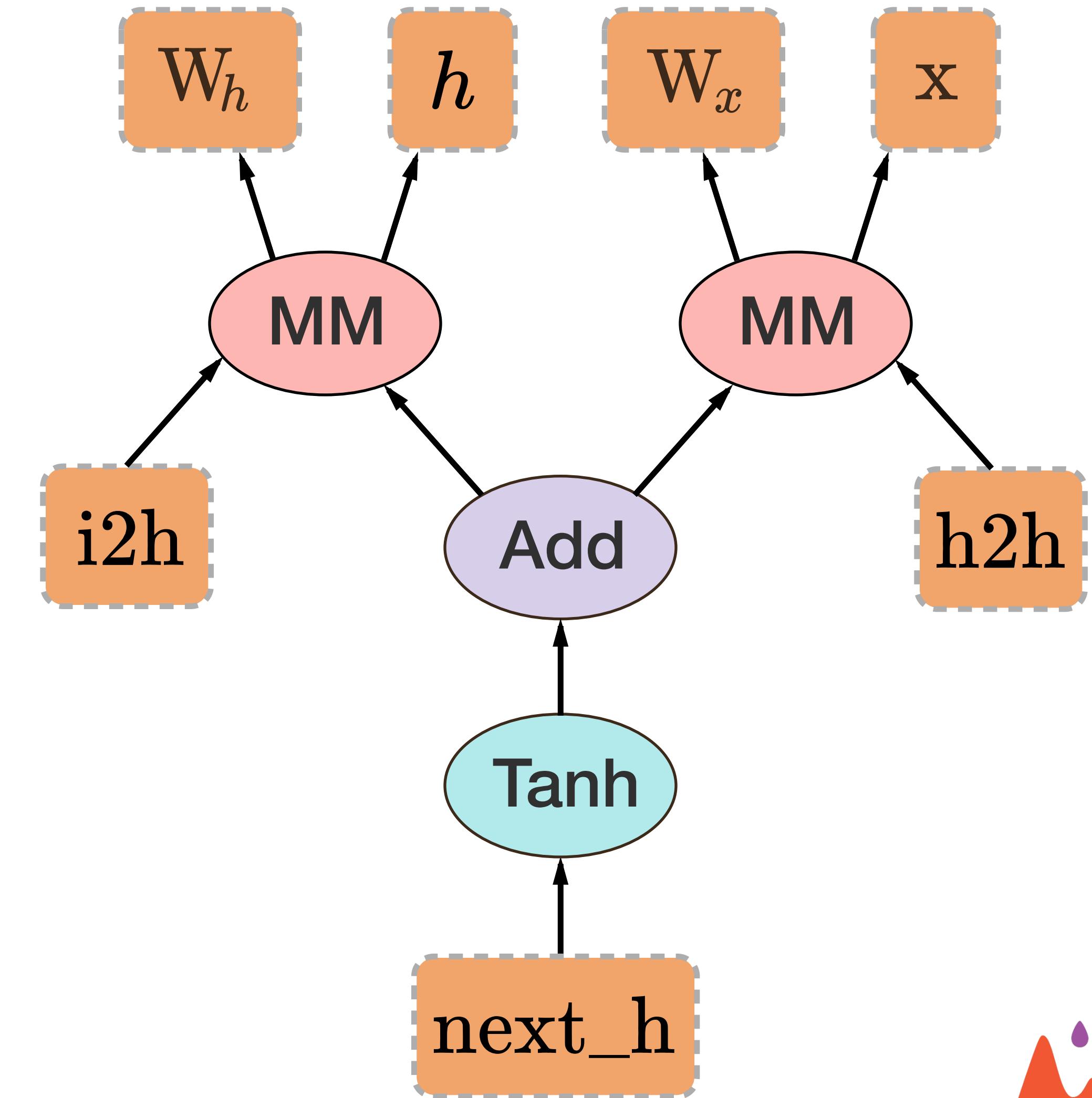
JIT Compilation

- Possible in imperative frameworks
- The key idea is deferred or lazy evaluation
 - $y = x + 2$
 - $z = y * y$
 - # nothing is executed yet, but the graph is being constructed
 - `print(z)` # now the entire graph is executed: $z = (x+2) * (x+2)$
- We can do just in time compilation on the graph before execution



Lazy Evaluation

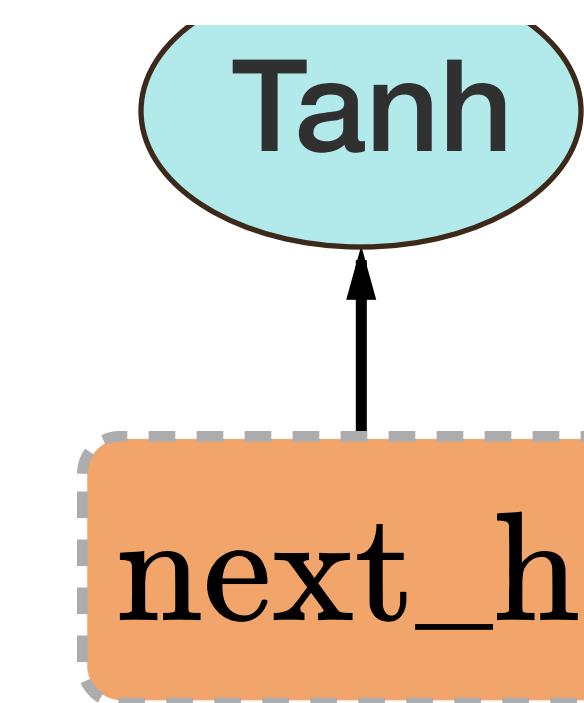
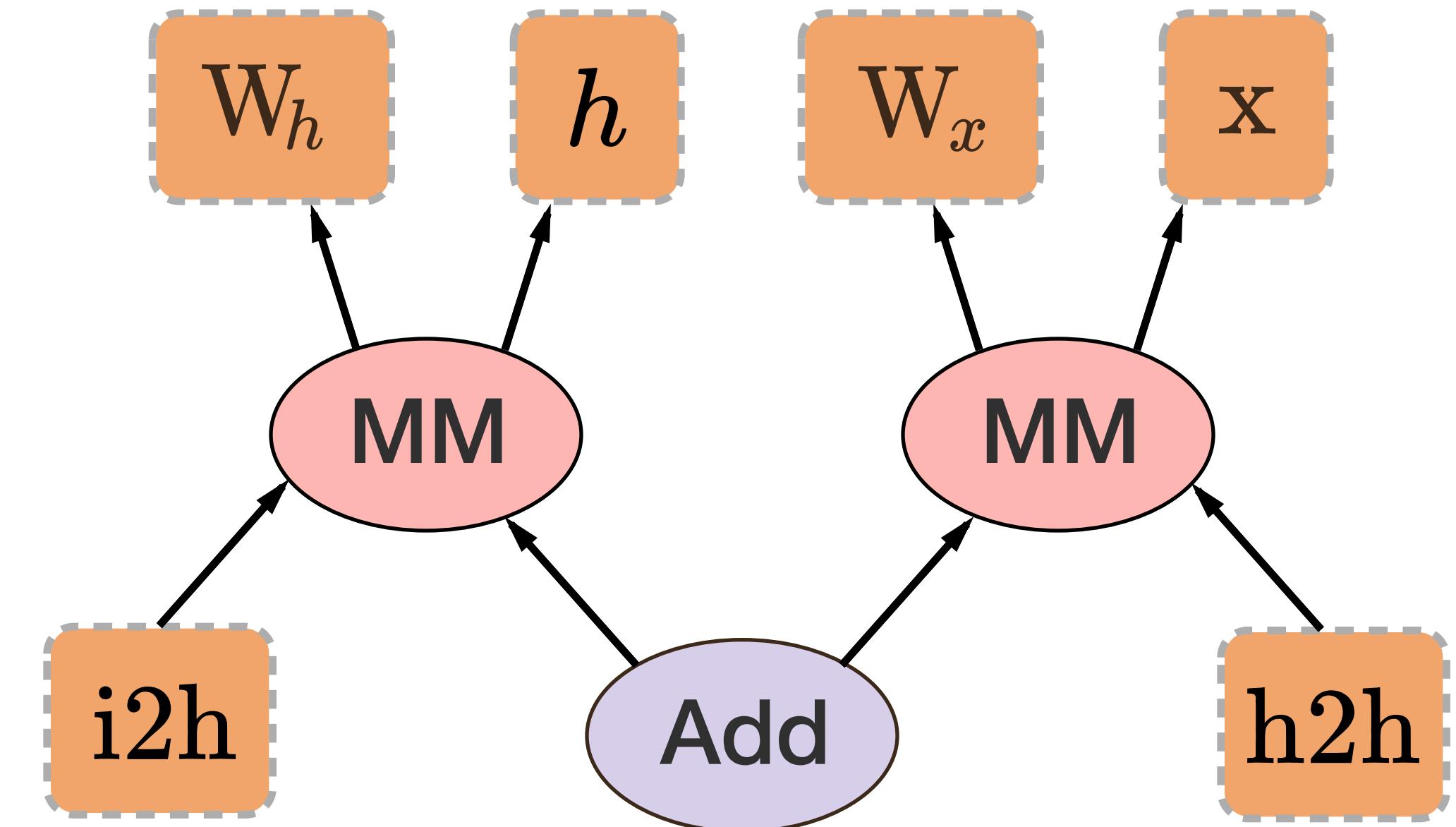
```
from torch.autograd import Variable  
  
x = Variable(torch.randn(1, 10))  
prev_h = Variable(torch.randn(1, 20))  
W_h = Variable(torch.randn(20, 20))  
W_x = Variable(torch.randn(20, 10))  
  
i2h = torch.mm(W_x, x.t())  
h2h = torch.mm(W_h, prev_h.t())  
next_h = i2h + h2h  
next_h = next_h.tanh()
```



Lazy Evaluation

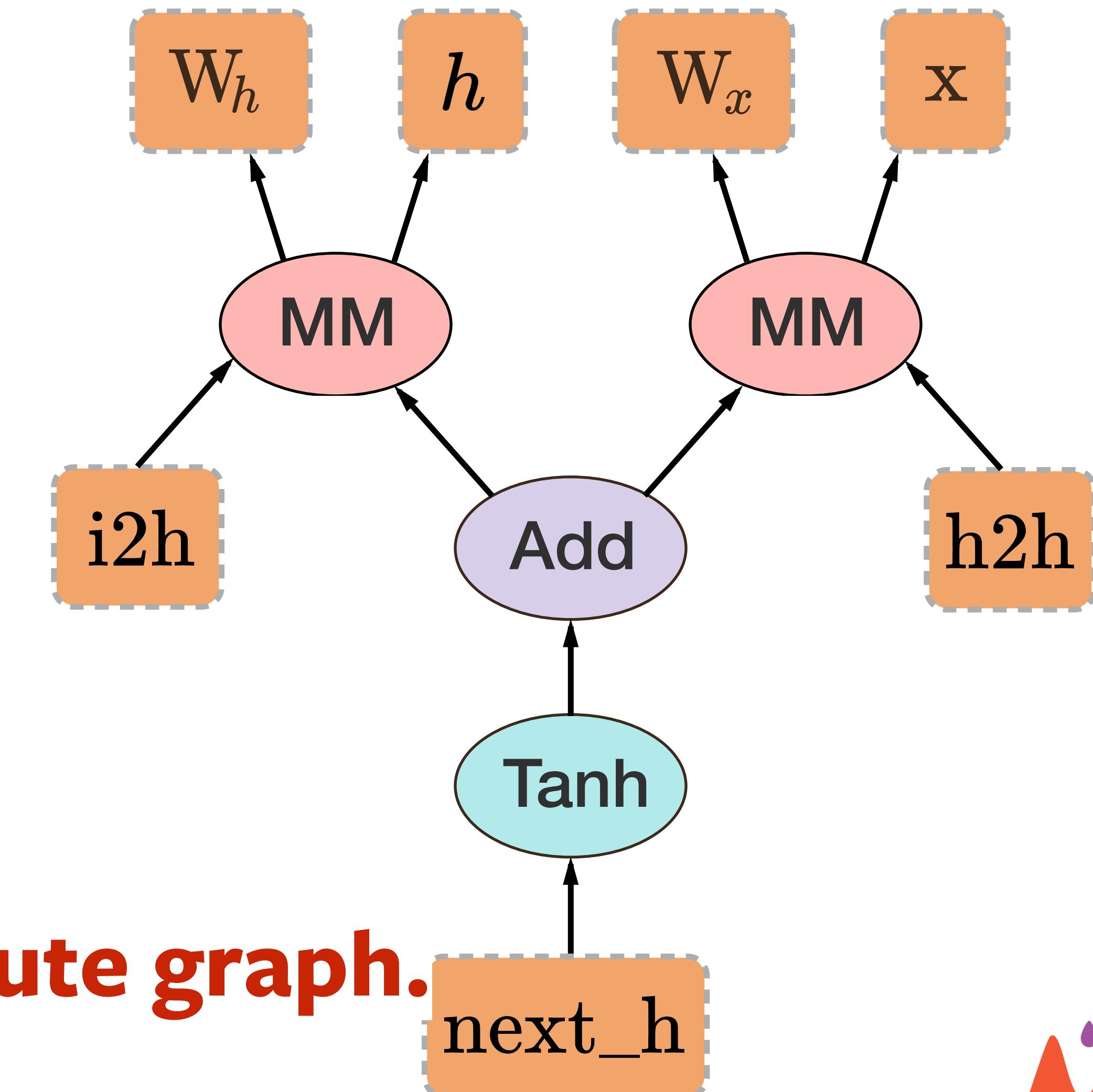
```
from torch.autograd import Variable  
  
x = Variable(torch.randn(1, 10))  
prev_h = Variable(torch.randn(1, 20))  
W_h = Variable(torch.randn(20, 20))  
W_x = Variable(torch.randn(20, 10))  
  
i2h = torch.mm(W_x, x.t())  
h2h = torch.mm(W_h, prev_h.t())  
next_h = i2h + h2h  
next_h = next_h
```

Graph built but not actually executed

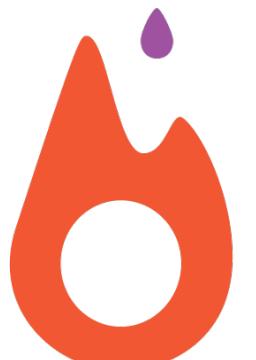


Lazy Evaluation

```
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()  
  
print(next_h)
```

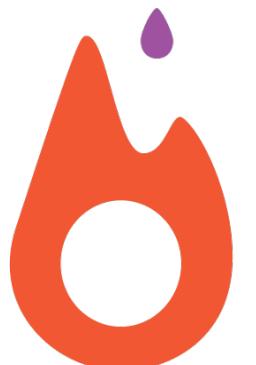
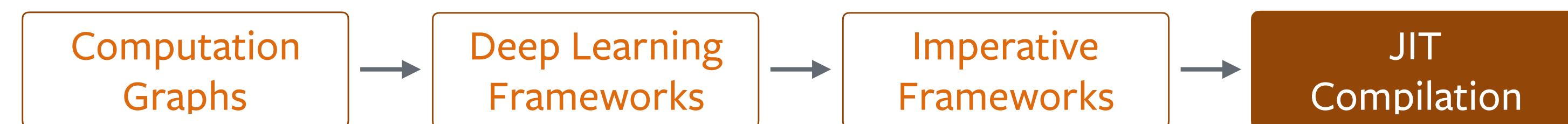
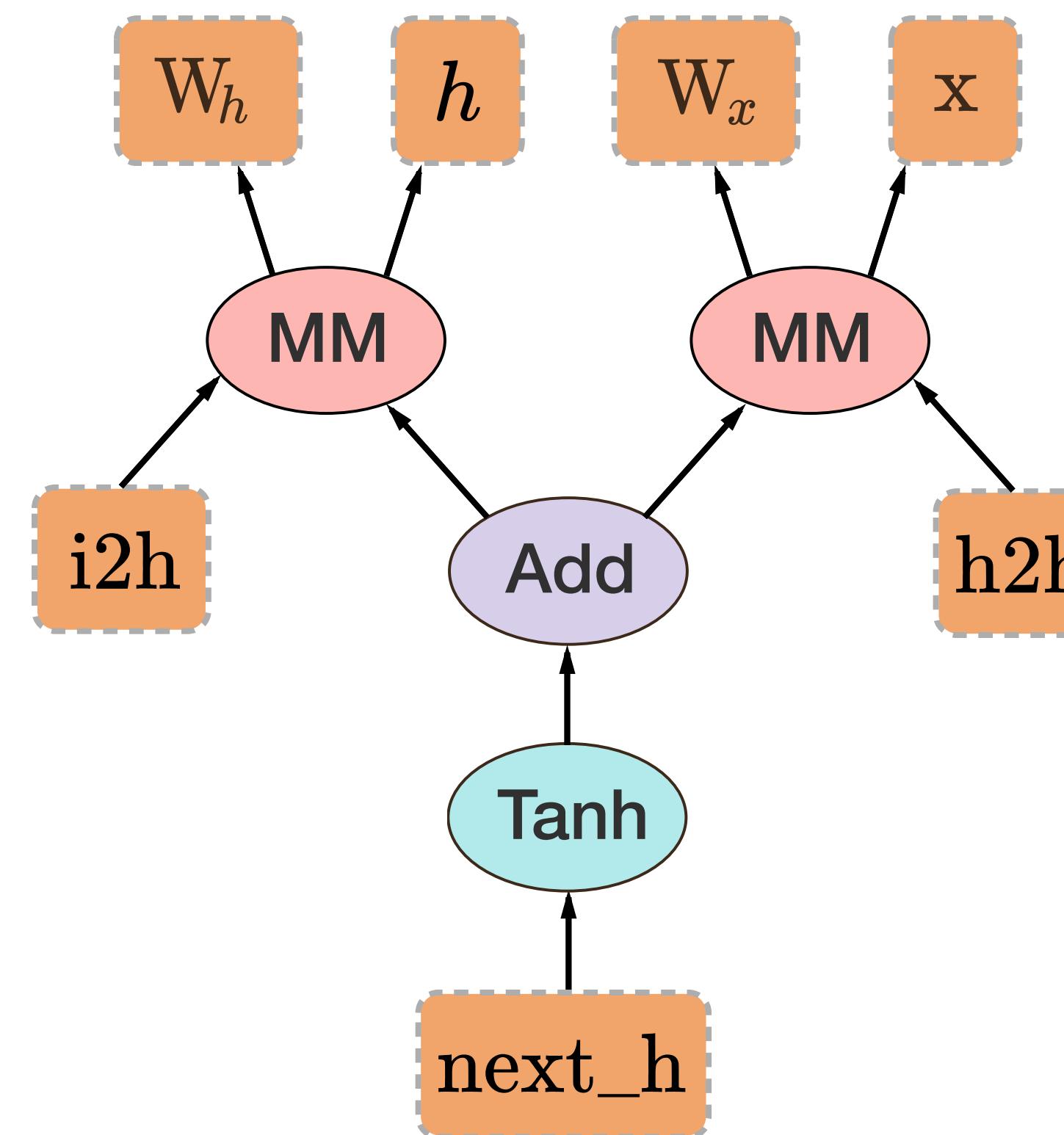


Data accessed. Execute graph.



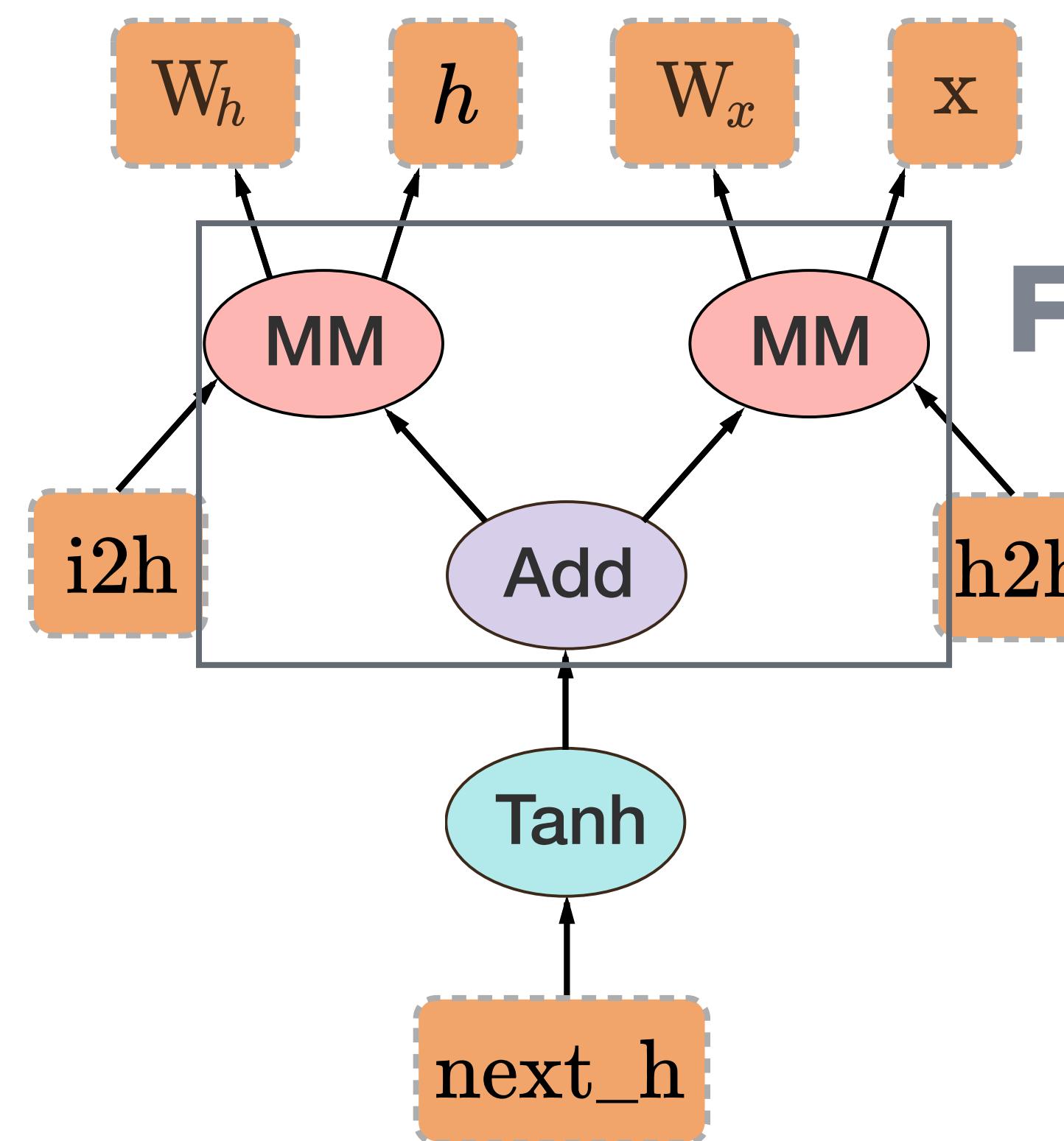
Lazy Evaluation

- A little bit of time between building and executing graph
 - Use it to compile the graph just-in-time



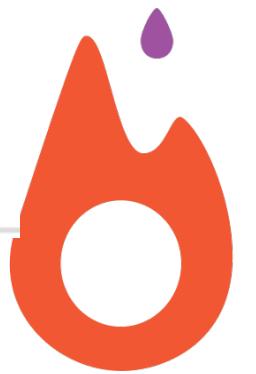
JIT Compilation

- Fuse and optimize operations



Fuse operations. Ex:

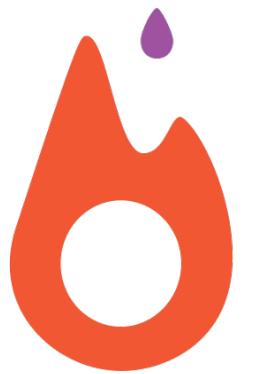
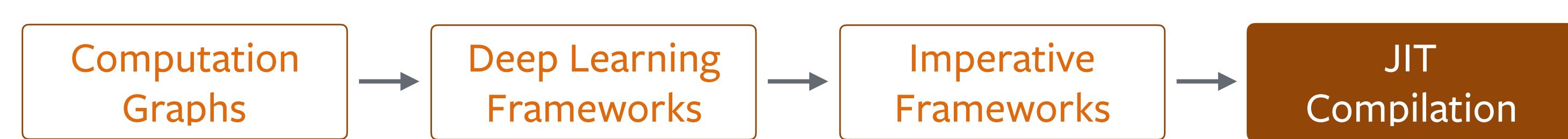
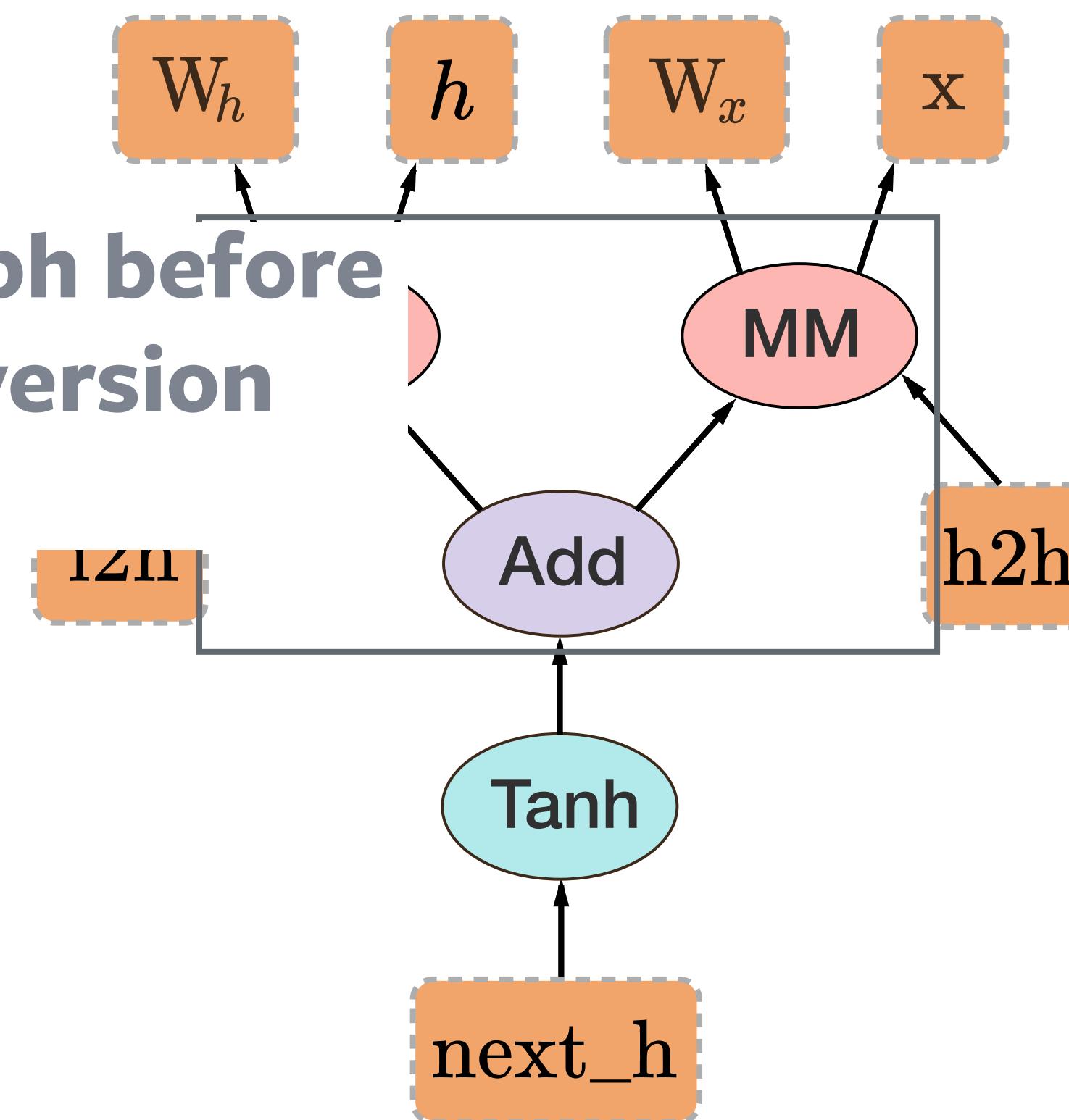
```
1  x = [0, 1, 2, 3, 4]
2  for i in range(len(x)):
3      x[i] = x[i] + 1
4
5  for i in range(len(x)):
6      x[i] = x[i] * 2
7
8
9  # Fused
10 for i in range(len(x)):
11     x[i] = (x[i] + 1) * 2
```



JIT Compilation

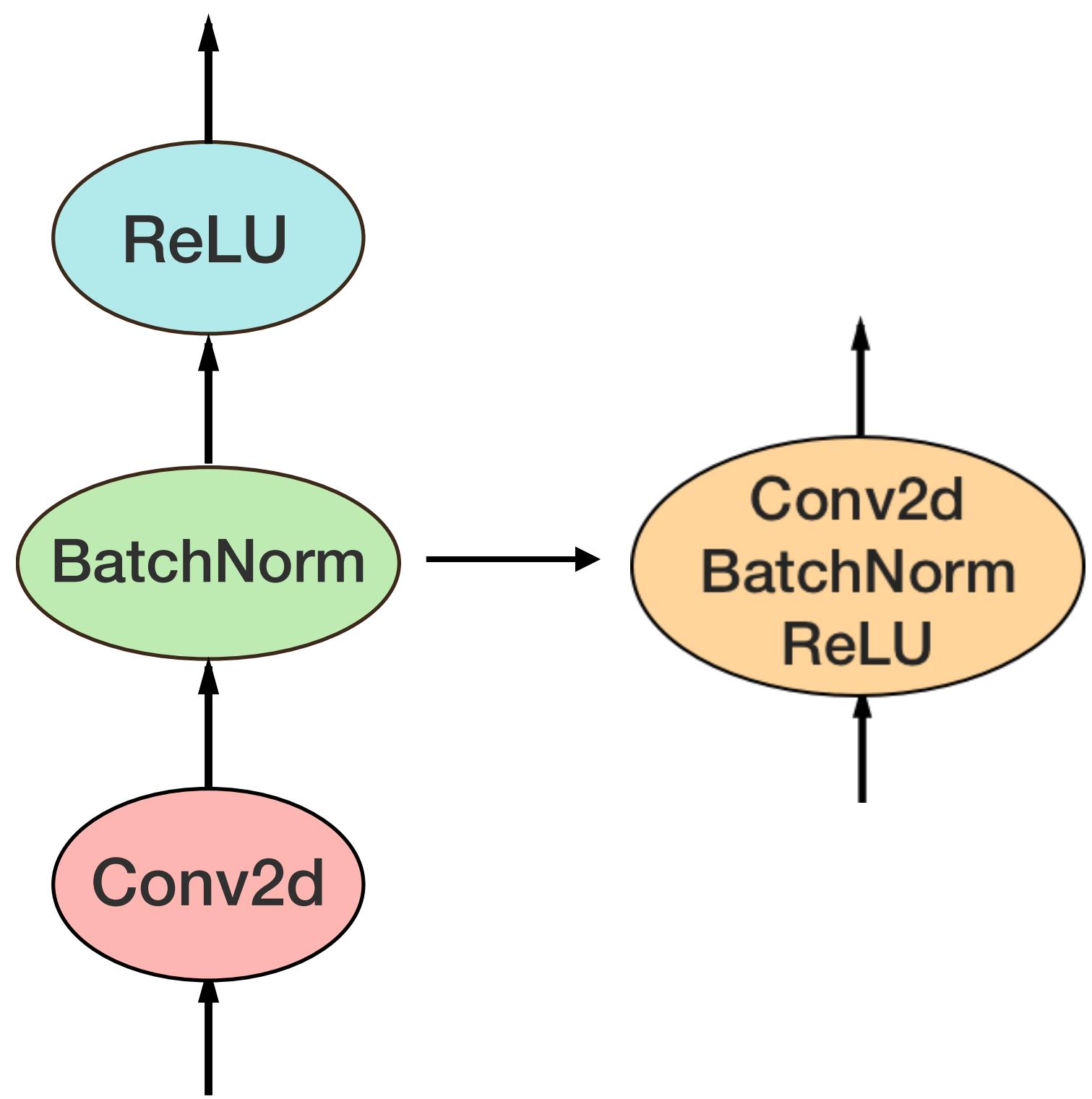
- Cache subgraphs

I've seen this part of the graph before
let me pull up the compiled version
from cache

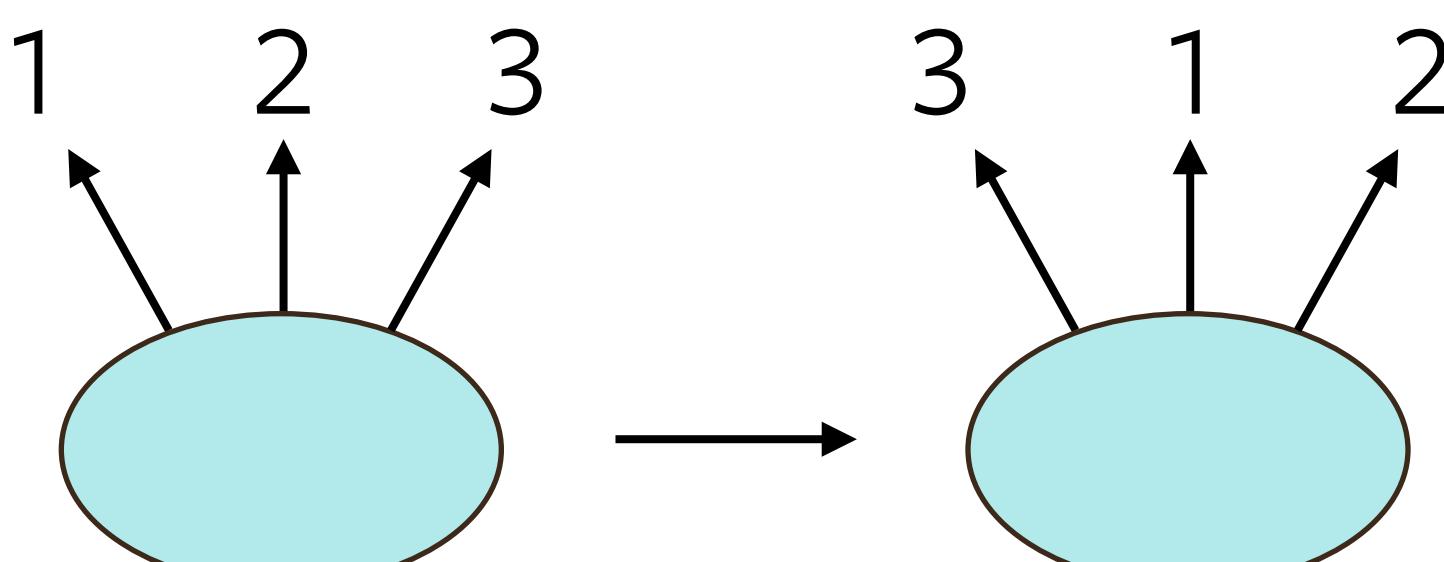


Compilation benefits

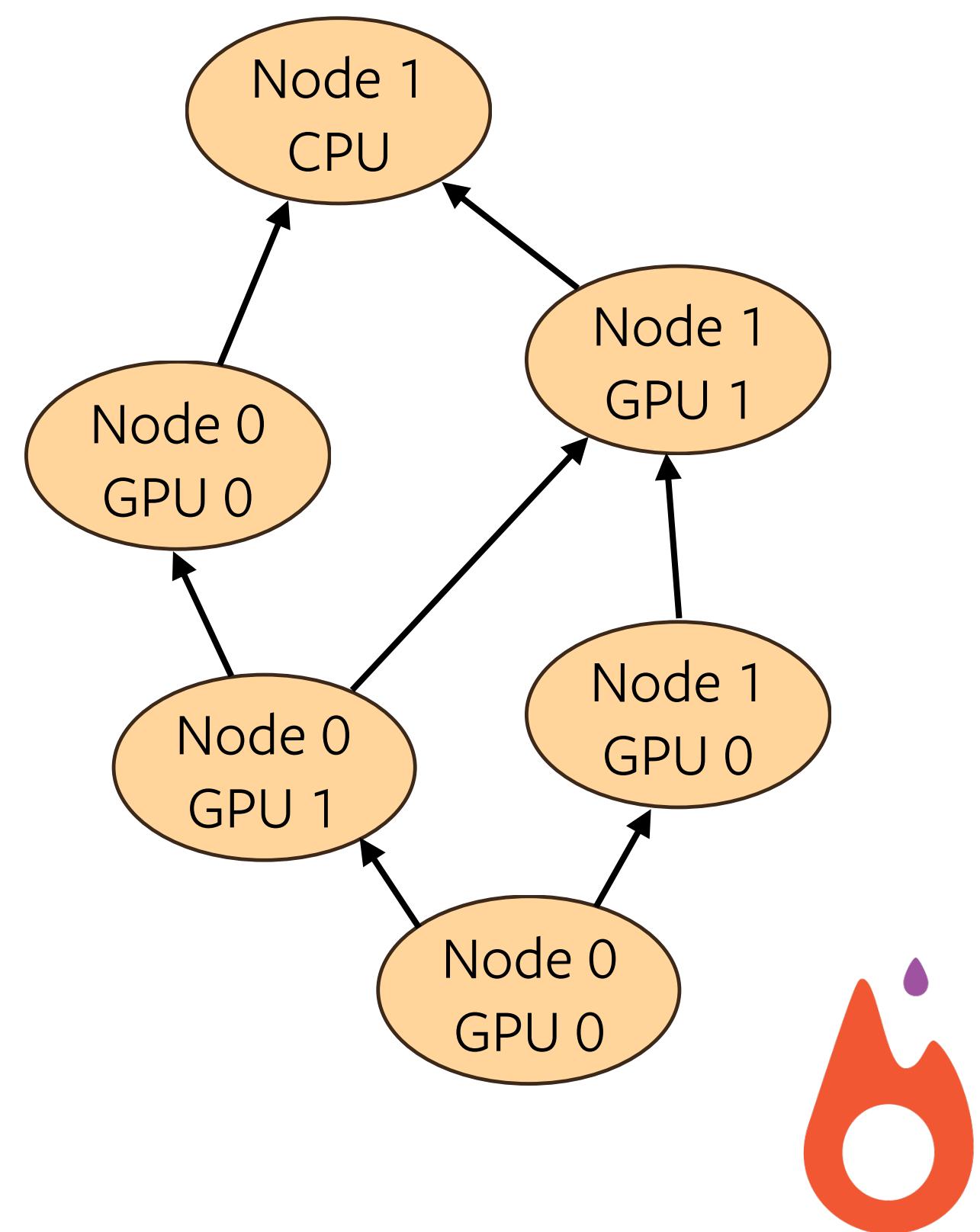
Kernel fusion



Out-of-order
execution



Automatic
work placement





<http://pytorch.org>

Released Jan 18th

100,000+ downloads

950+ community repos

8200+ user posts

245 contributors

With ❤ from

facebook



NVIDIA

salesforce

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIE
PARIS INSTITUTE OF TECHNOLOGY

Carnegie
Mellon
University

UNIVERSITE
PIERRE & MARIE CURIE
LA SCIENCE A PARIS

Digital
Reasoning

Stanford
University

UNIVERSITY OF
OXFORD

NYU

Inria

ENS
ÉCOLE NORMALE
SUPÉRIEURE

EPFL
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Berkeley
UNIVERSITY OF CALIFORNIA