

**facebook**

Artificial Intelligence Research



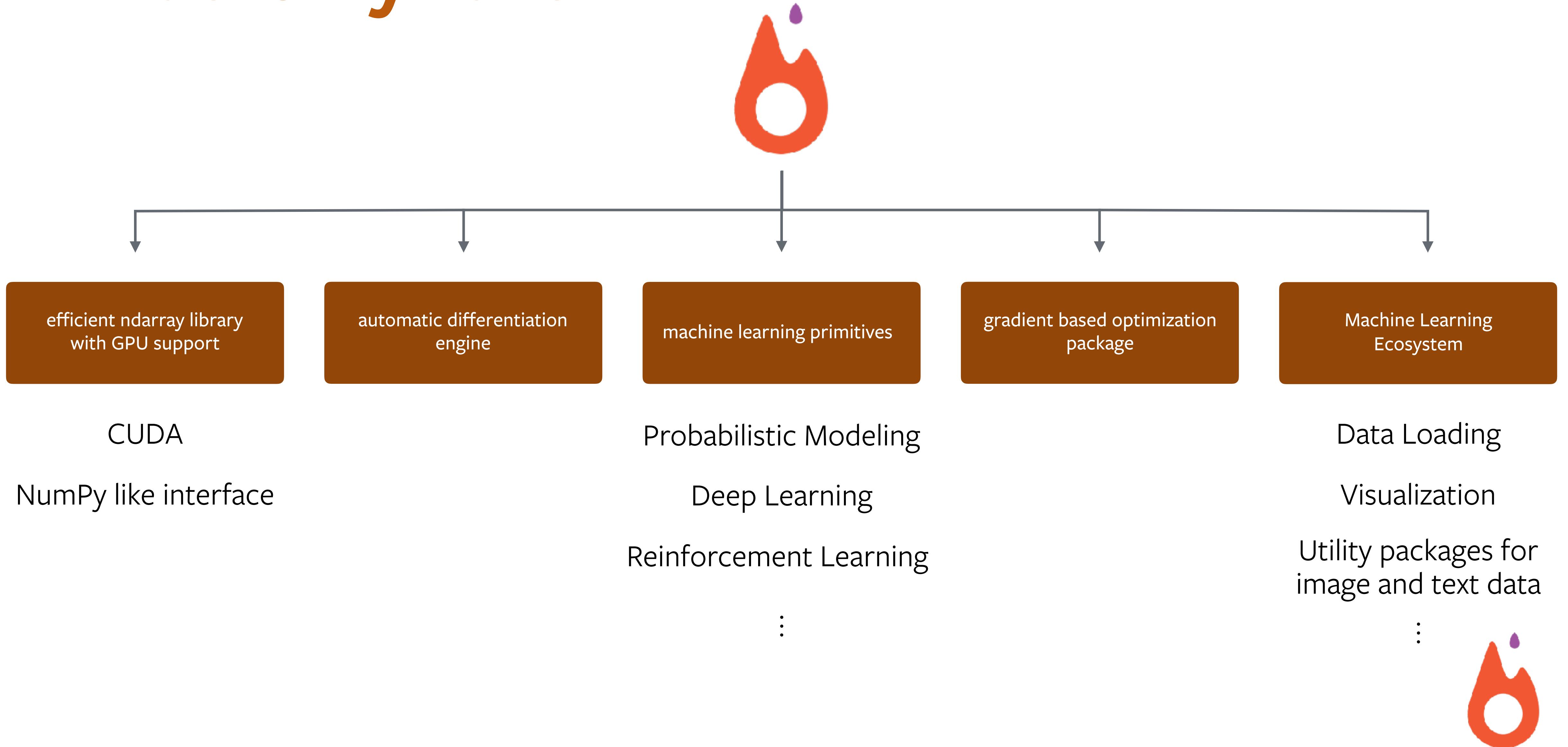
Adam Paszke, Sam Gross, Soumith Chintala, **Francisco Massa**, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Alban Desmaison, Andreas Kopf, Edward Yang, Zach Devito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy & Team



# What is PyTorch?



# What is PyTorch



# Tensor (ndarray) library



# Tensor (ndarray) library

- NumPy-like API ( np.ndarray <-> torch.Tensor )
- Very fast acceleration on NVIDIA GPUs



# Tensor (ndarray) library

- NumPy-like API ( `np.ndarray <-> torch.Tensor` )
  - Easy creation of Tensors with various dtypes and on different devices
  - Scientific computing methods (linear algebra, reduction, etc.)
  - Fast conversion from and to `np.ndarray`



# Tensor (ndarray) library

- NumPy-like API ( np.ndarray <-> torch.Tensor )

```
In [1]: import torch  
import numpy as np
```

```
In [2]: a = torch.rand(2, 2) # a random Tensor  
a
```

```
Out[2]: tensor([[ 0.8836,  0.9006],  
                 [ 0.1145,  0.8328]])
```

```
In [3]: a + 1
```

```
Out[3]: tensor([[ 1.8836,  1.9006],  
                 [ 1.1145,  1.8328]])
```

```
In [4]: torch.svd(a) # linear algebra operations! (SVD)
```

```
Out[4]: (tensor([[-0.8548, -0.5190],  
                  [-0.5190,  0.8548]]),  
        tensor([ 1.4521,  0.4358]),  
        tensor([[[-0.5611, -0.8278],  
                  [-0.8278,  0.5611]]]))
```

```
In [5]: a.numpy() # convert to a NumPy ndarray
```

```
Out[5]: array([[ 0.88362992,  0.90055138],  
                 [ 0.11449063,  0.83281231]], dtype=float32)
```

```
In [6]: torch.from_numpy(np.zeros((2, 3))) # convert a NumPy ndarray to a Tensor
```

```
Out[6]: tensor([[ 0.,  0.,  0.],  
                 [ 0.,  0.,  0.]], dtype=torch.float64)
```



```

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 = np.matmul(x, w1)
    h_relu = np.maximum(h, 0)
    y_pred = np.matmul(h_relu, w2)

    # Compute and print loss
    loss = ((y_pred - y) ** 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 = np.matmul(h_relu.T, grad_y_pred)
    grad_h_relu = np.matmul(grad_y_pred, w2.T)
    grad_h = grad_h_relu.copy()
    grad_h[h < 0] = 0
    grad_w1 = np.matmul(x.T, grad_h)

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

```

# Numpy

```

import torch

# 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)
y = torch.randn(N, D_out)

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

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

    # Compute and print loss
    loss = ((y_pred - y) ** 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 = torch.matmul(h_relu.t(), grad_y_pred)
    grad_h_relu = torch.matmul(grad_y_pred, w2.t())
    grad_h = grad_h_relu.clone()
    grad_h[h < 0] = 0
    grad_w1 = torch.matmul(x.t(), grad_h)

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

```

# PyTorch

# Tensor (ndarray) library

- Very fast acceleration on NVIDIA GPUs

- Efficient scientific computing on GPU
- Clean device management
- Seamless conversion to and from CPU



# Tensor (ndarray) library

- Very fast acceleration on NVIDIA GPUs

```
In [1]: import torch

In [2]: cuda = torch.device("cuda") # cuda device object

In [3]: a = torch.rand(3, 3, device=cuda) # create a random Tensor on GPU
a

Out[3]: tensor([[ 0.7402,  0.1515,  0.7794],
               [ 0.3259,  0.3071,  0.4555],
               [ 0.3657,  0.1409,  0.5663]], device='cuda:0')

In [4]: a + 1 # operations also work on GPU

Out[4]: tensor([[ 1.7402,  1.1515,  1.7794],
               [ 1.3259,  1.3071,  1.4555],
               [ 1.3657,  1.1409,  1.5663]], device='cuda:0')

In [5]: a.svd() # linear algebra operations on GPU! (SVD)

Out[5]: (tensor([[-0.7615,  0.5344, -0.3668],
               [-0.4326, -0.8405, -0.3263],
               [-0.4826, -0.0898,  0.8712]], device='cuda:0'),
         tensor([ 1.4171,  0.2102,  0.0898], device='cuda:0'),
         tensor([[-0.6218,  0.4226, -0.6594],
               [-0.2232, -0.9026, -0.3681],
               [-0.7507, -0.0817,  0.6555]], device='cuda:0'))

In [6]: cpu = torch.device("cpu") # cpu device object
a.to(cpu) # convert to a CPU tensor

Out[6]: tensor([[ 0.7402,  0.1515,  0.7794],
               [ 0.3259,  0.3071,  0.4555],
               [ 0.3657,  0.1409,  0.5663]])
```



# Automatic Differentiation Engine



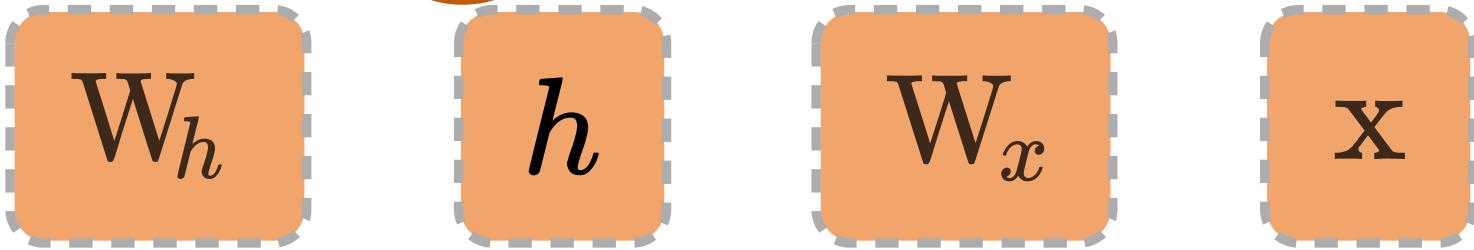
# Automatic Differentiation Engine

Computation as a graph built on-the-fly



# Automatic Differentiation Engine

Computation as a graph built on-the-fly



```
x = torch.randn(1, 10)
prev_h = torch.randn(1, 20)
w_h = torch.randn(20, 20)
w_x = torch.randn(20, 10)
```

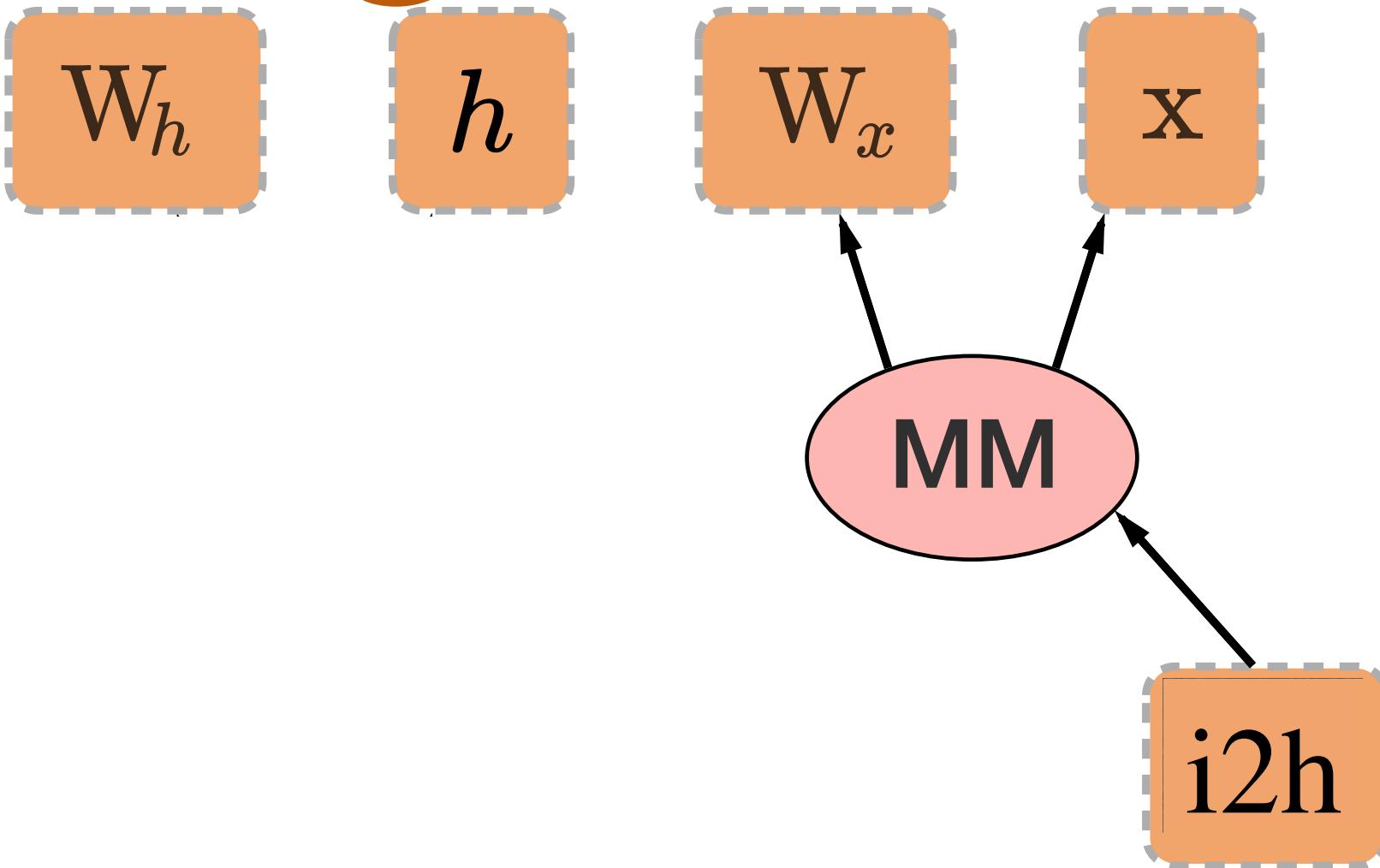


# Automatic Differentiation Engine

Computation as a graph built on-the-fly

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

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

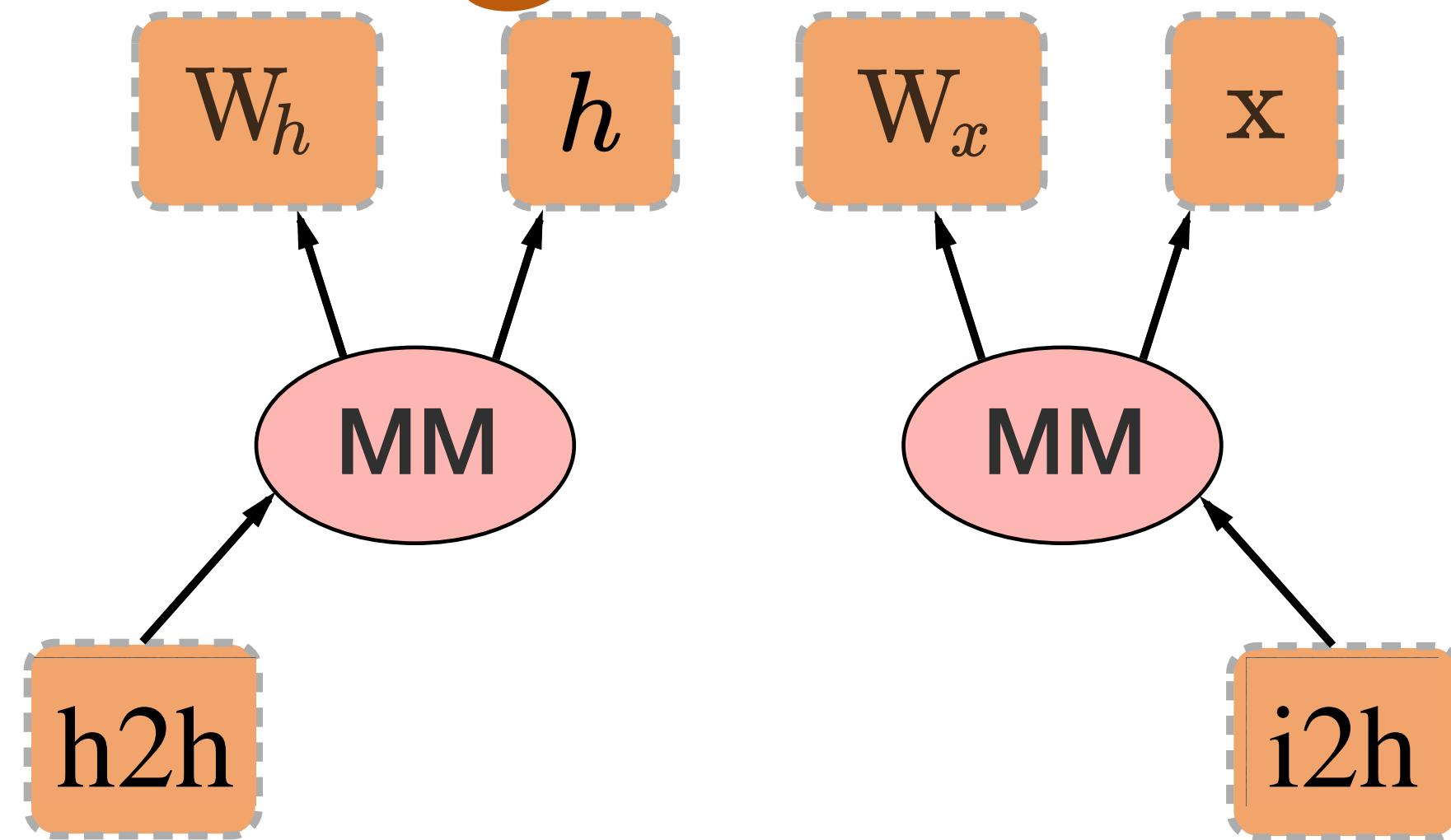


# Automatic Differentiation Engine

Computation as a graph built on-the-fly

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

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

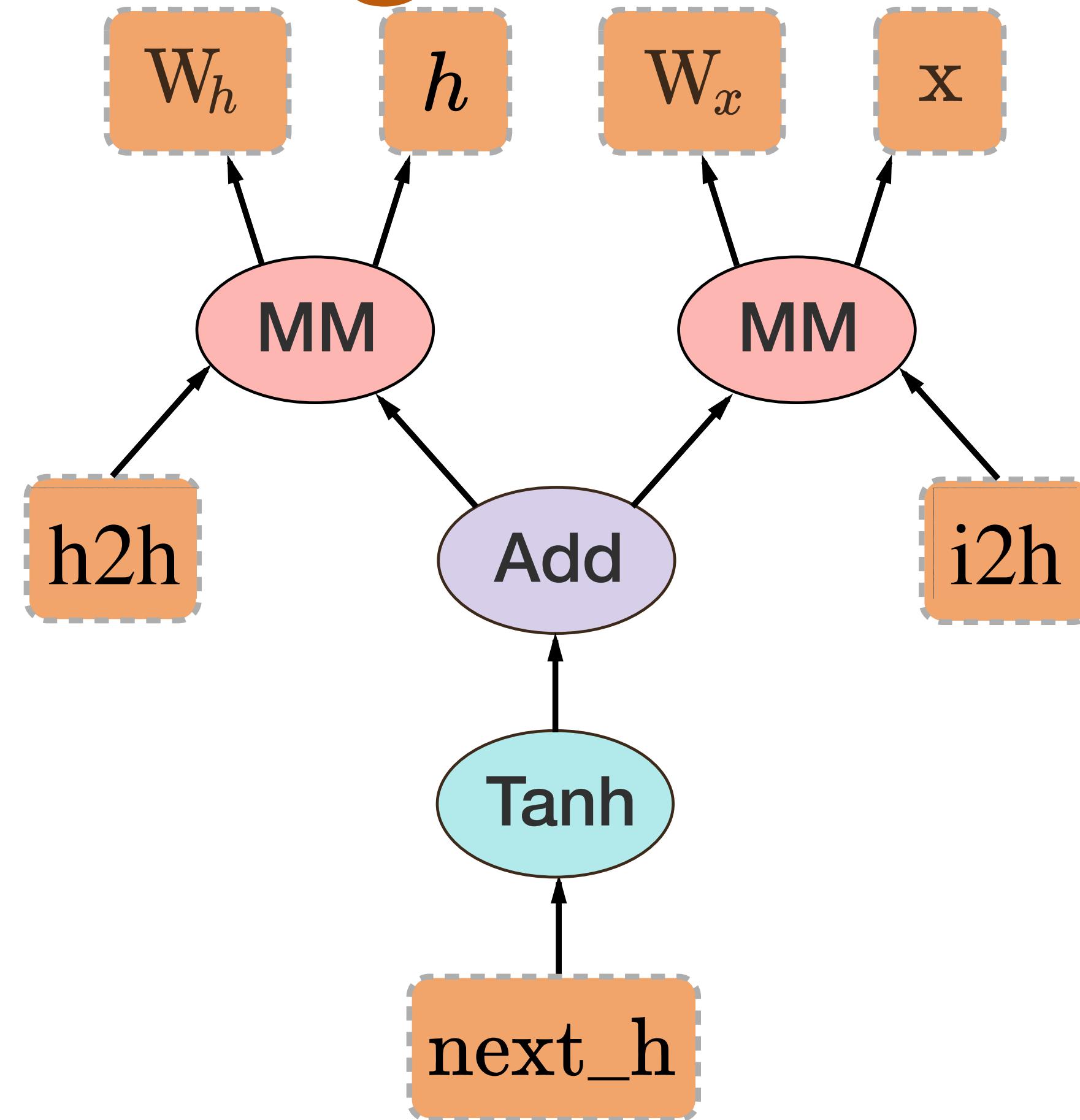


# Automatic Differentiation Engine

Computation as a graph built on-the-fly

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



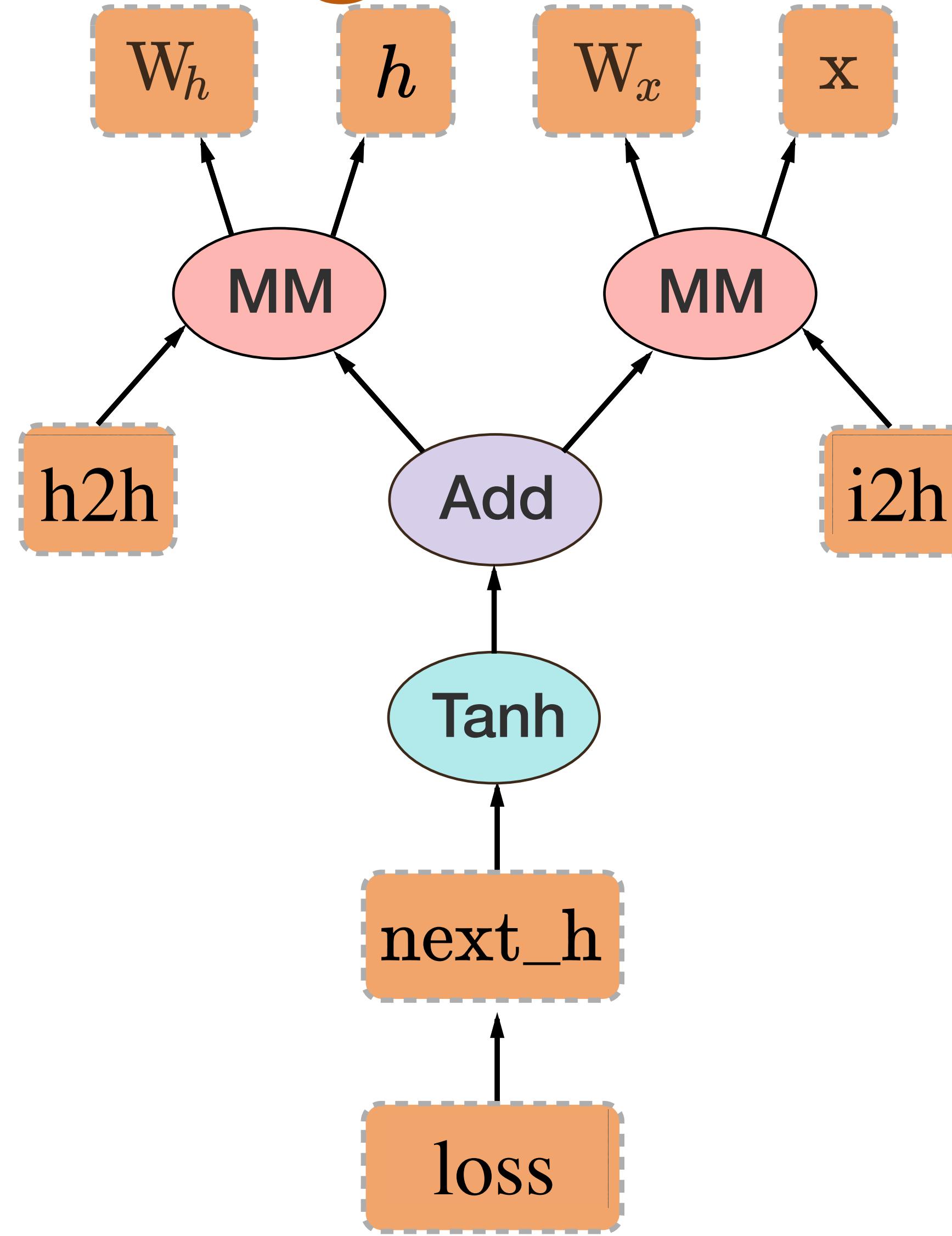
# Automatic Differentiation Engine

Computation as a graph built on-the-fly

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

loss = next_h.sum()
```



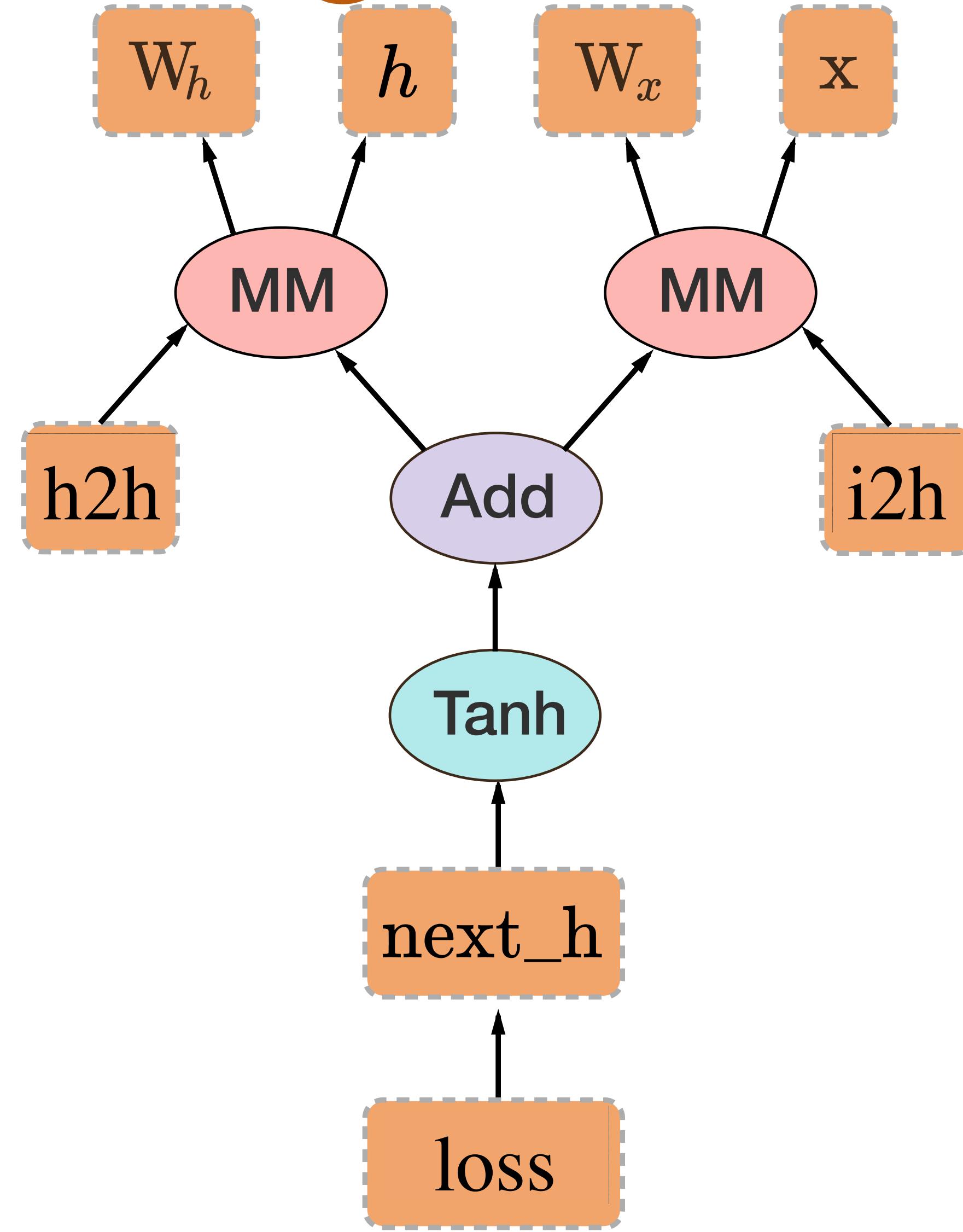
# Automatic Differentiation Engine

Computation as a graph built on-the-fly

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

loss = next_h.sum()
loss.backward()
```



# Automatic Differentiation Engine

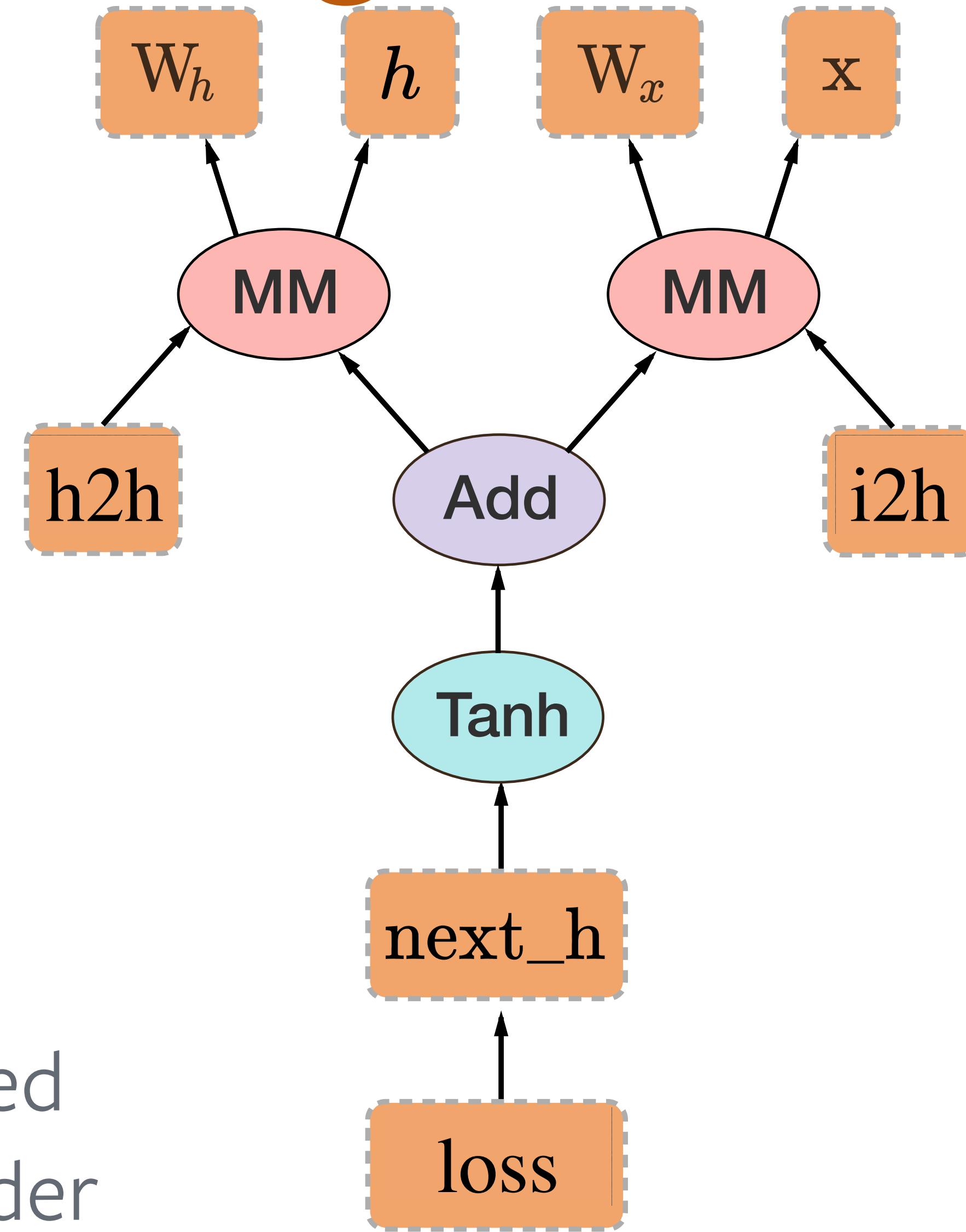
Computation as a graph built on-the-fly

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

loss = next_h.sum()
loss.backward() # compute gradients!
```

Gradient w.r.t. the input Tensors is computed  
step-by-step from `loss` to top in reverse order



# Automatic Differentiation Engine

- Computation as a graph built on-the-fly
  - Can use Python primitives to build the graph (e.g. for-loop)
- Functions with efficient backward implementations
  - `torch.matmul`, `torch.fft`, `torch.svd`, `torch.trtrs`, etc.
- Gradients by automatic backpropagation through the graph
  - Higher-order gradients (backward traversal is also a graph)
  - Multi-device graphs



# Efficient Machine Learning Primitives



# Machine Learning primitives

- Deep Learning
  - torch.nn
- Reinforcement Learning, Probabilistic Modeling
  - torch.distributions
- and more...
  - scientific computing methods + autograd



# Neural Networks

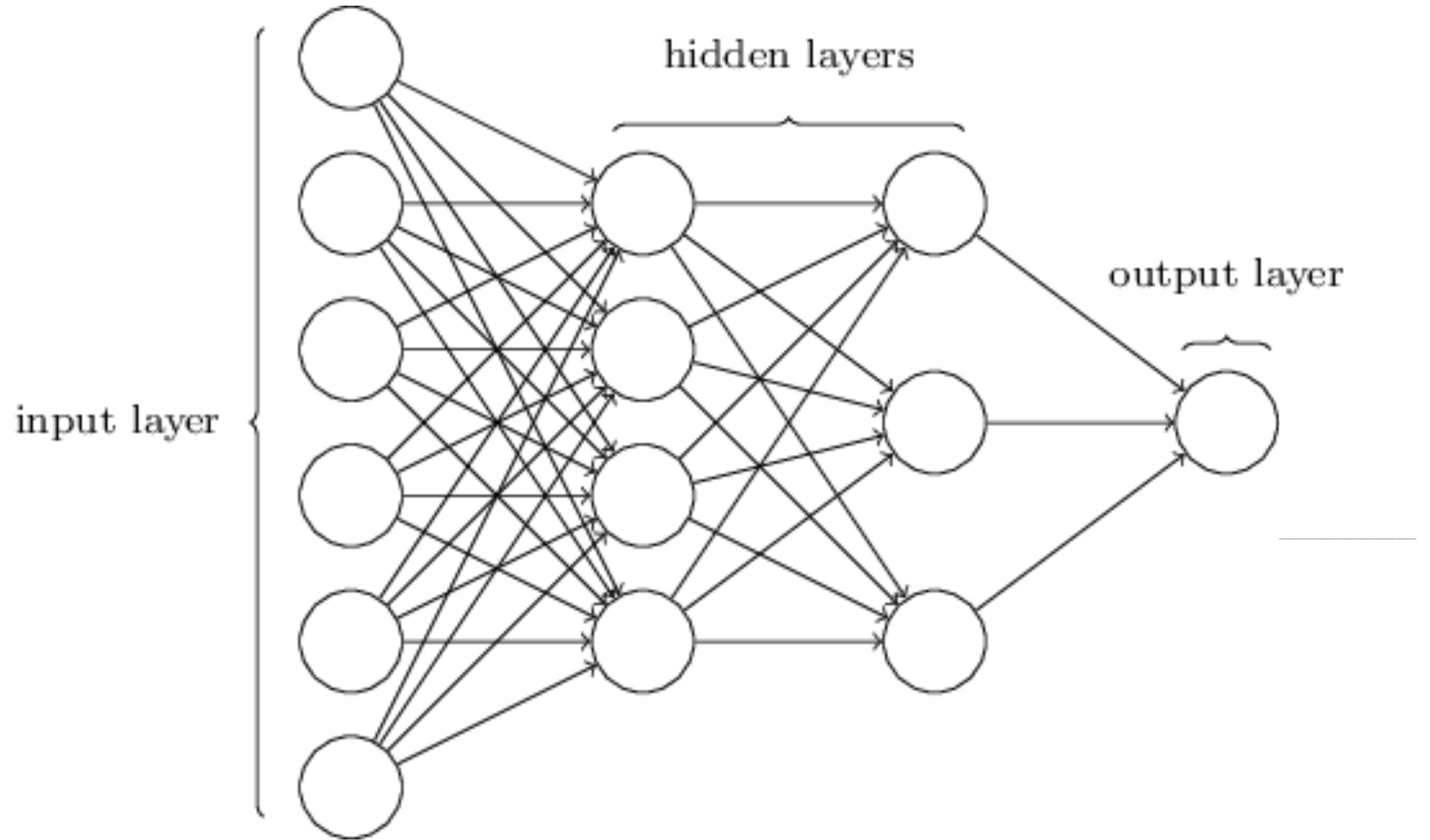
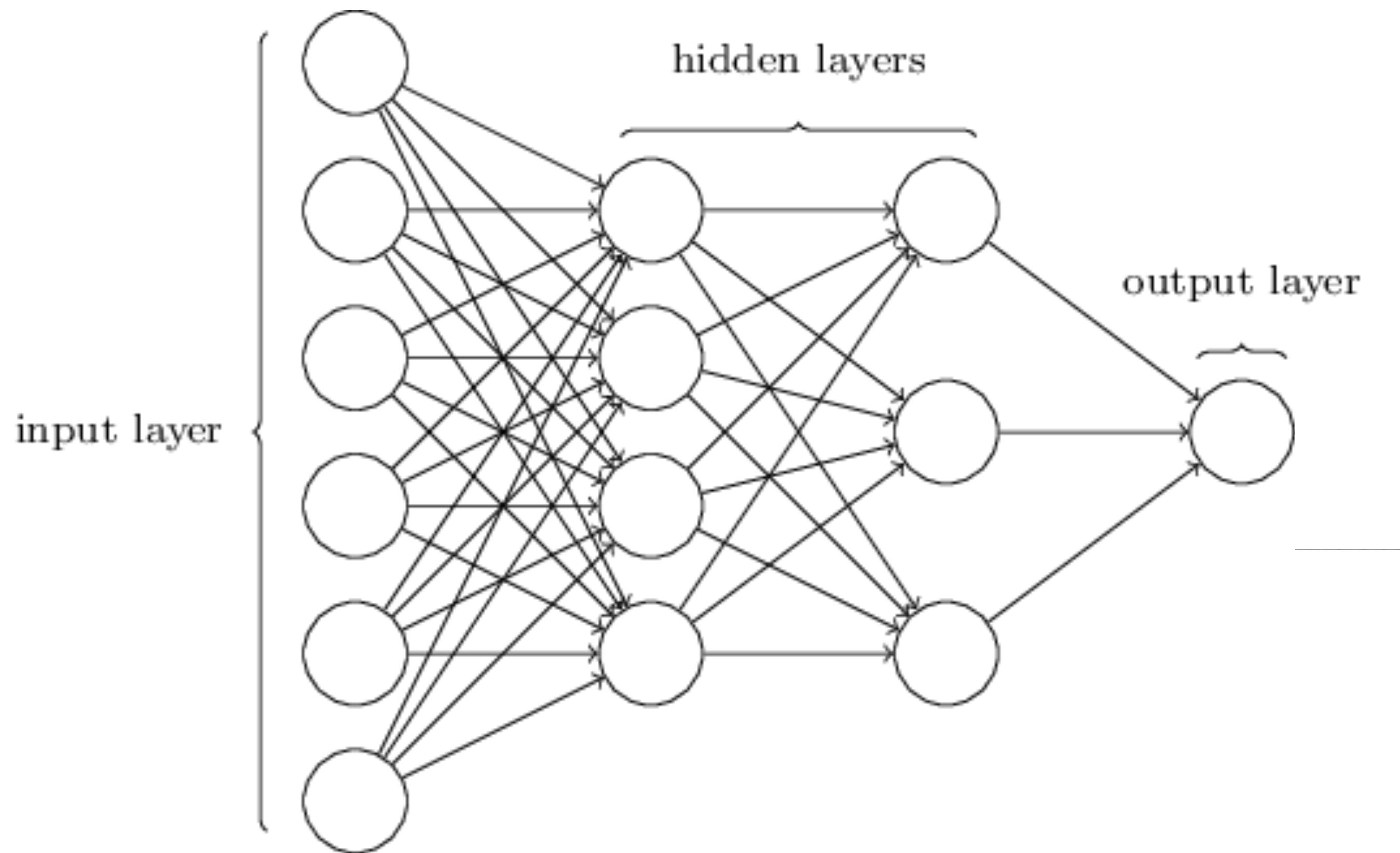


Figure by Md. Rezaul Karim

# Neural Networks



```
1 import torch.nn as nn  
2  
3 my_network = nn.Sequential(  
4     nn.Linear(6, 4),  
5     nn.Sigmoid(),  
6     nn.Linear(4, 3),  
7     nn.Sigmoid(),  
8     nn.Linear(3, 1),  
9 )
```



# Convolutional Neural Networks (CNN)

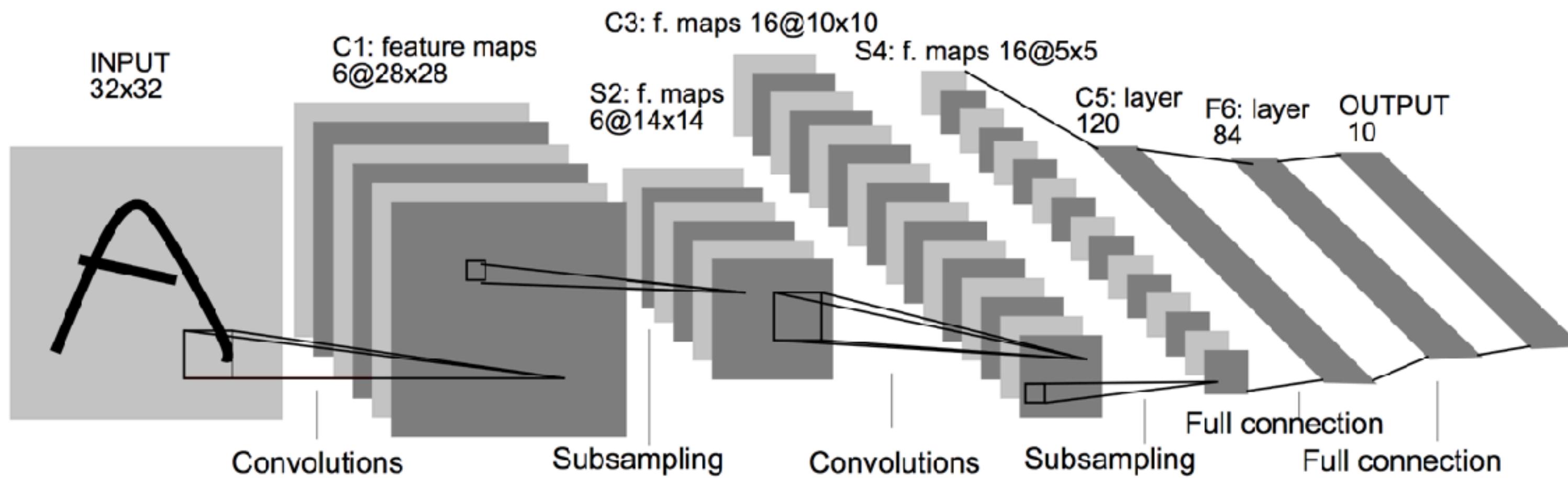
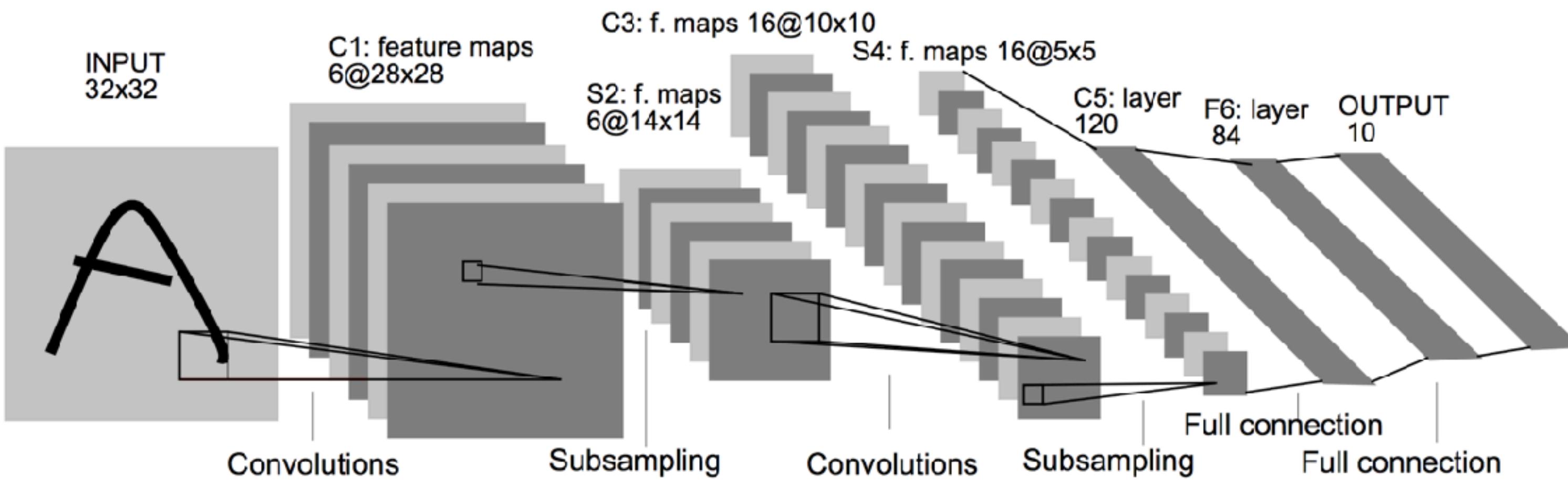


Figure by Yann LeCun et al.



# Convolutional Neural Networks (CNN)



```
1 import torch.nn as nn
2
3 class LeNet5(nn.Module):
4     def __init__(self):
5         super(LeNet5, self).__init__()
6         self.feature = nn.Sequential(
7             nn.Conv2d(1, 6, kernel_size=5),
8             nn.Tanh(),
9             nn.MaxPool2d(2),
10            nn.Conv2d(6, 16, kernel_size=5),
11            nn.Tanh(),
12            nn.MaxPool2d(2),
13        )
14         self.fc = nn.Sequential(
15             nn.Linear(400, 120),
16             nn.Tanh(),
17             nn.Linear(120, 84),
18             nn.Tanh(),
19             nn.Linear(84, 10),
20         )
21
22     def forward(self, input):
23         features = self.feature(input)
24         features = features.view(-1, 400) # flatten
25         return self.fc(features)
26
27 my_cnn = LeNet5()
```



# Recurrent Neural Networks (RNN)

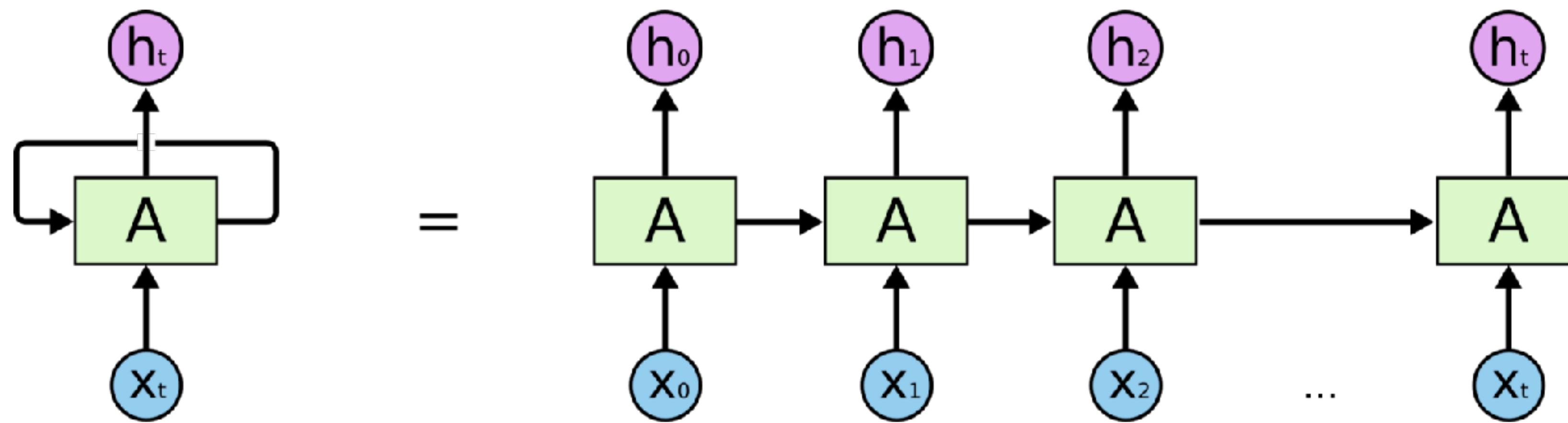
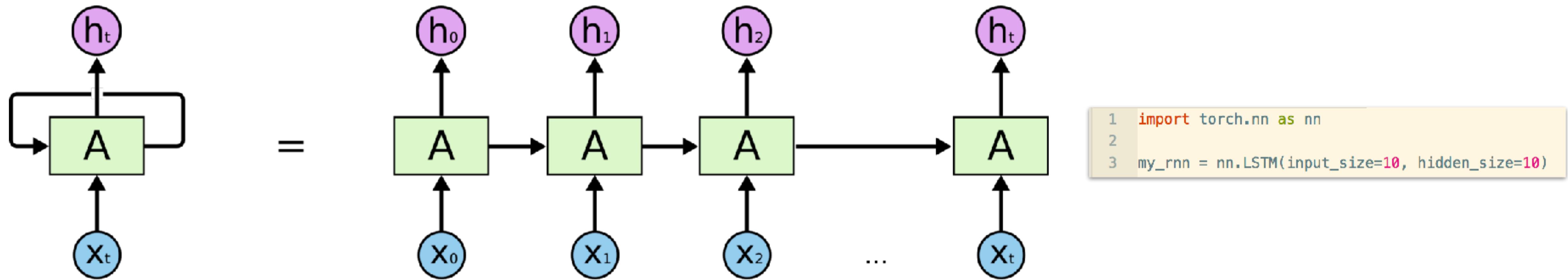


Figure by Pranoy Radhakrishnan

# Recurrent Neural Networks (RNN)



# Gradient based optimization package



# Gradient based optimization package

- State-of-the-art optimization algorithms

- `torch.optim.*`
  - SGD, Adam, RMSProp, L-BFGS, etc.

- Learning Rate scheduler

- `torch.optim.lr_scheduler.*`

- Extensible API

```
1 optimizer = torch.optim.Adam(model.parameters())
2
3 for input, target in data:
4     optimizer.zero_grad()
5     output = model(input)
6     loss = F.cross_entropy_loss(output, target)
7     loss.backward()
8     optimizer.step()
```



# Machine Learning Ecosystem



# 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

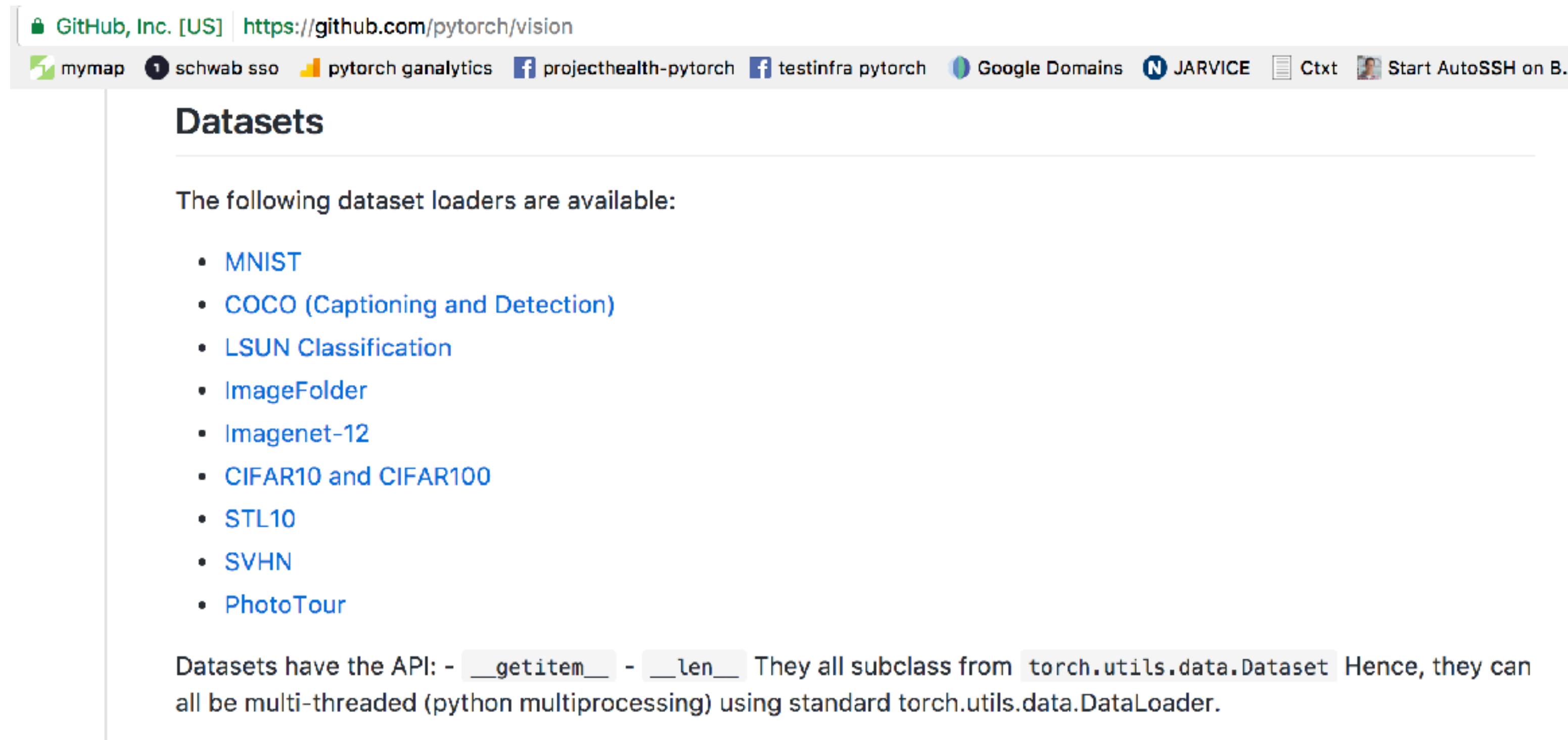
- every dataset is slightly differently formatted
- have to be preprocessed and normalized differently
- need a multithreaded Data loader to feed GPUs fast enough



# Writing Data Loaders

## PyTorch solution:

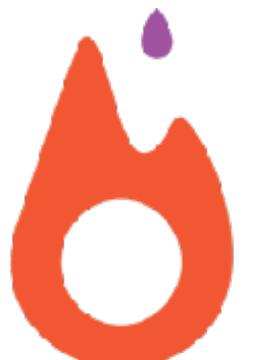
- share data loaders across the community!



The screenshot shows a GitHub repository page for 'pytorch/vision'. The title is 'Datasets'. Below it, a section titled 'The following dataset loaders are available:' lists ten datasets:

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

At the bottom, a note states: '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`'.

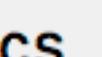


# Writing Data Loaders

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 black boxes.



# 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

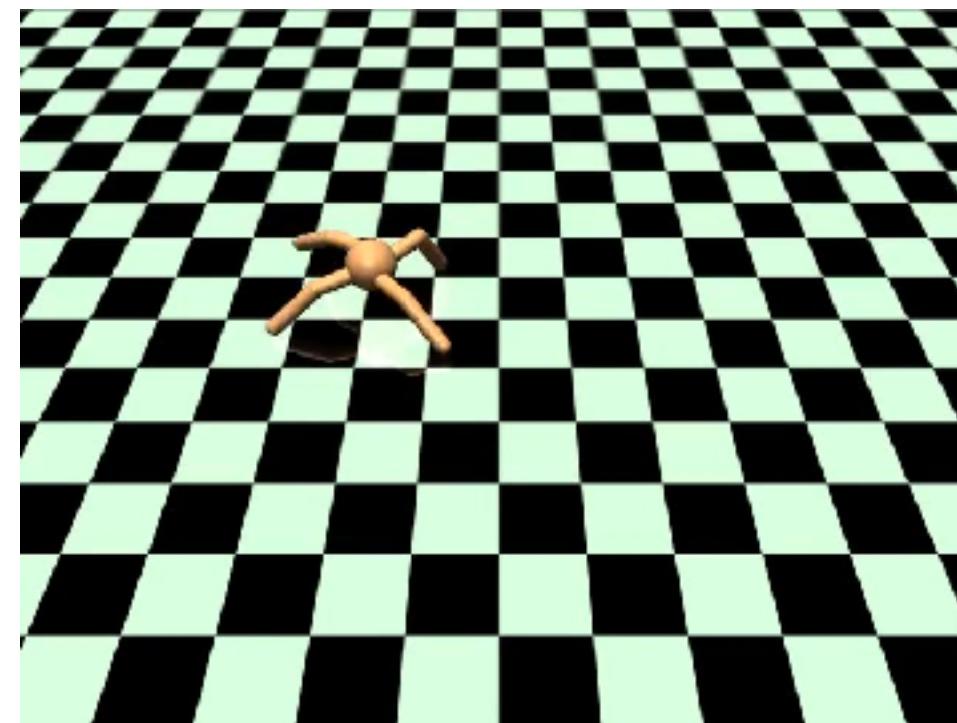
def __len__(self):
    return len(self.ids)
```



# Interfacing with environments



Cars



Simulations



Video games



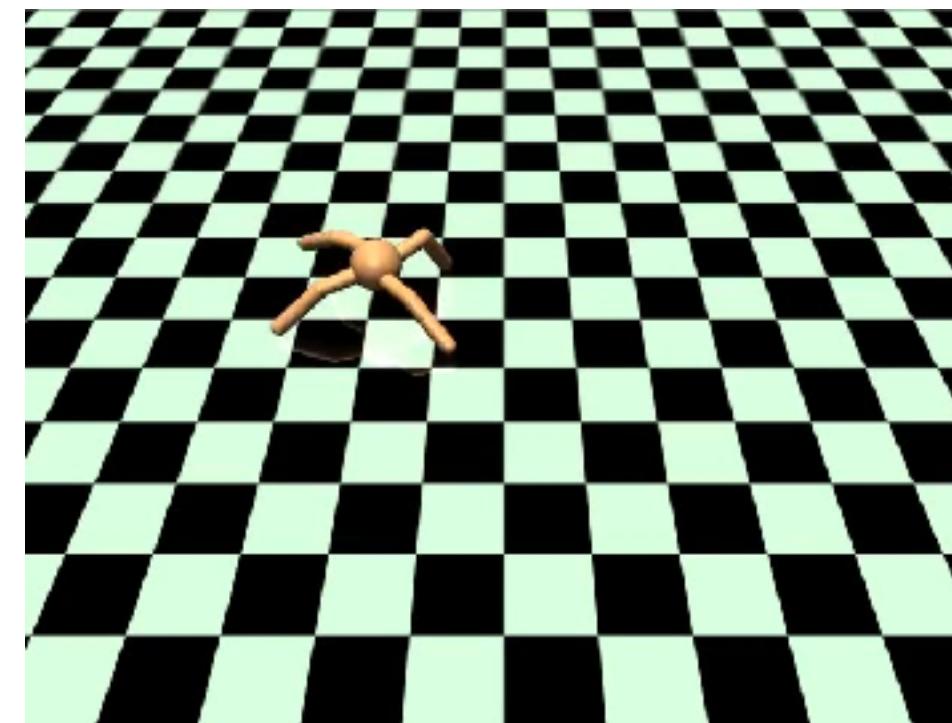
Internet



# Interfacing with environments



Cars



Simulations



Video games



Internet

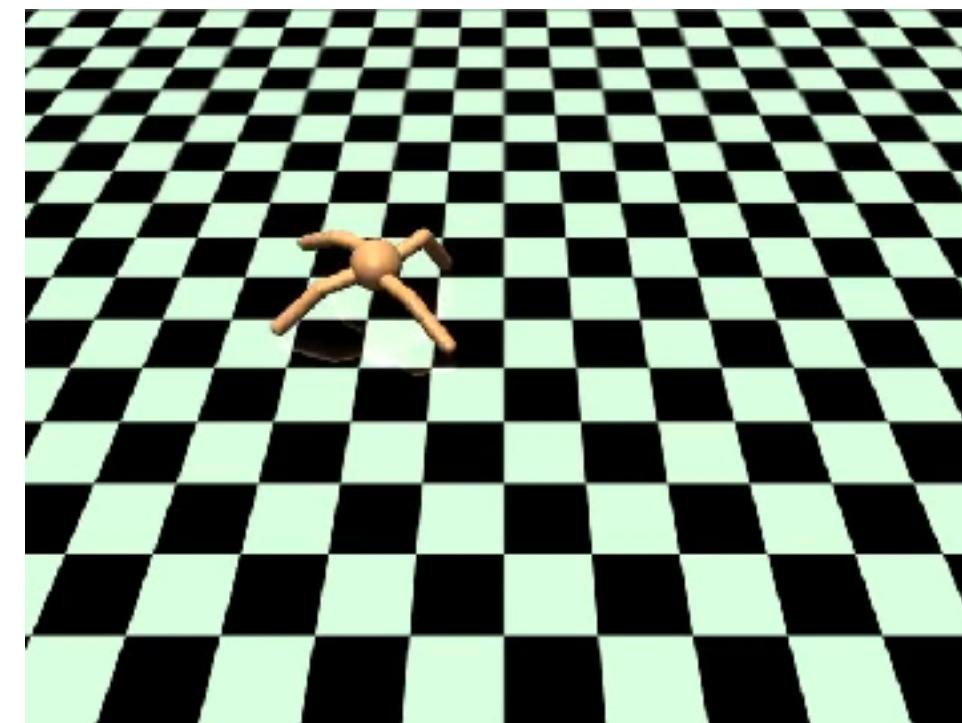
Pretty much every environment provides a Python API



# Interfacing with environments



Cars



Simulations



Video games

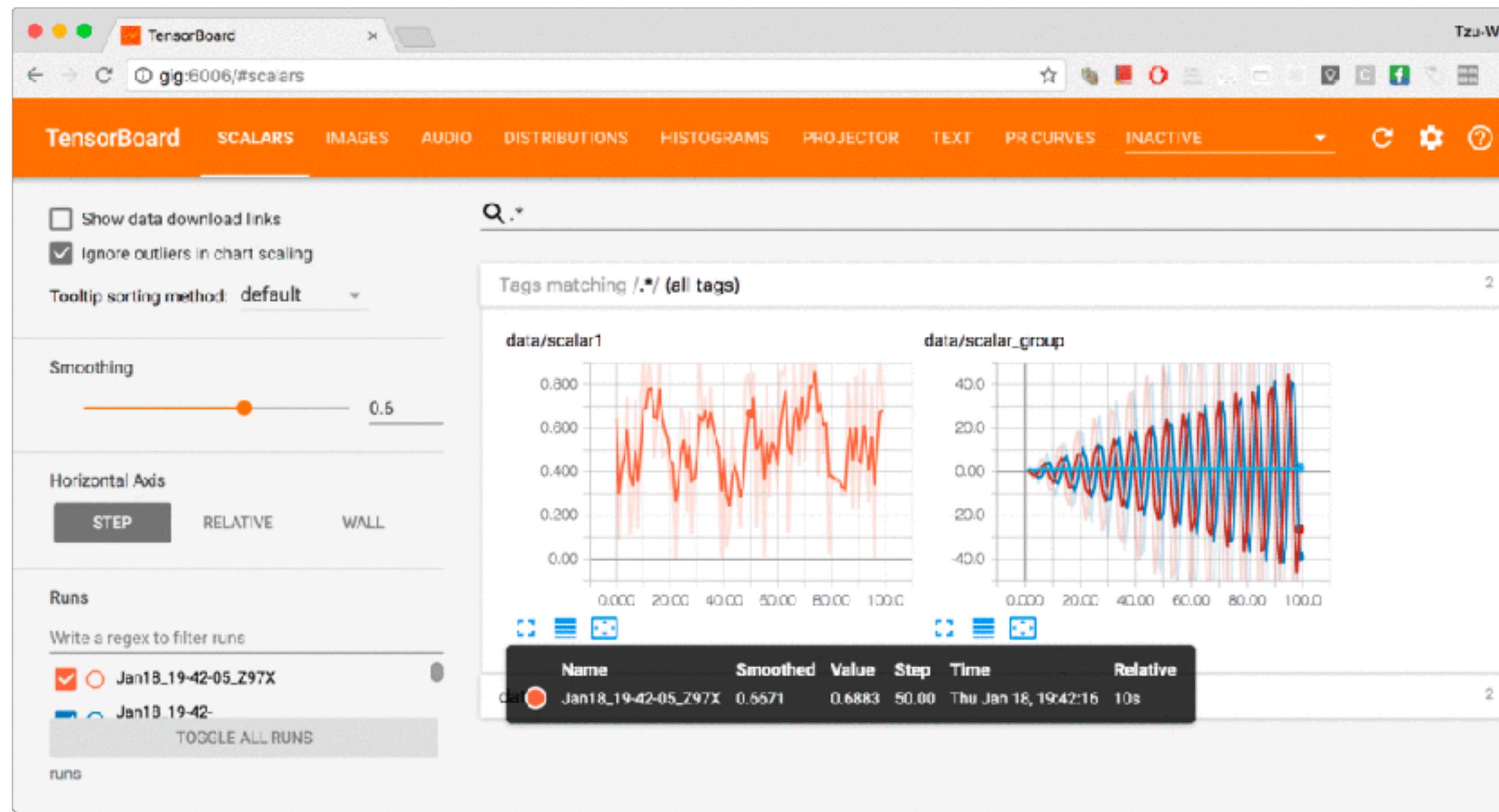


Internet

**Natively interact with the environment directly**

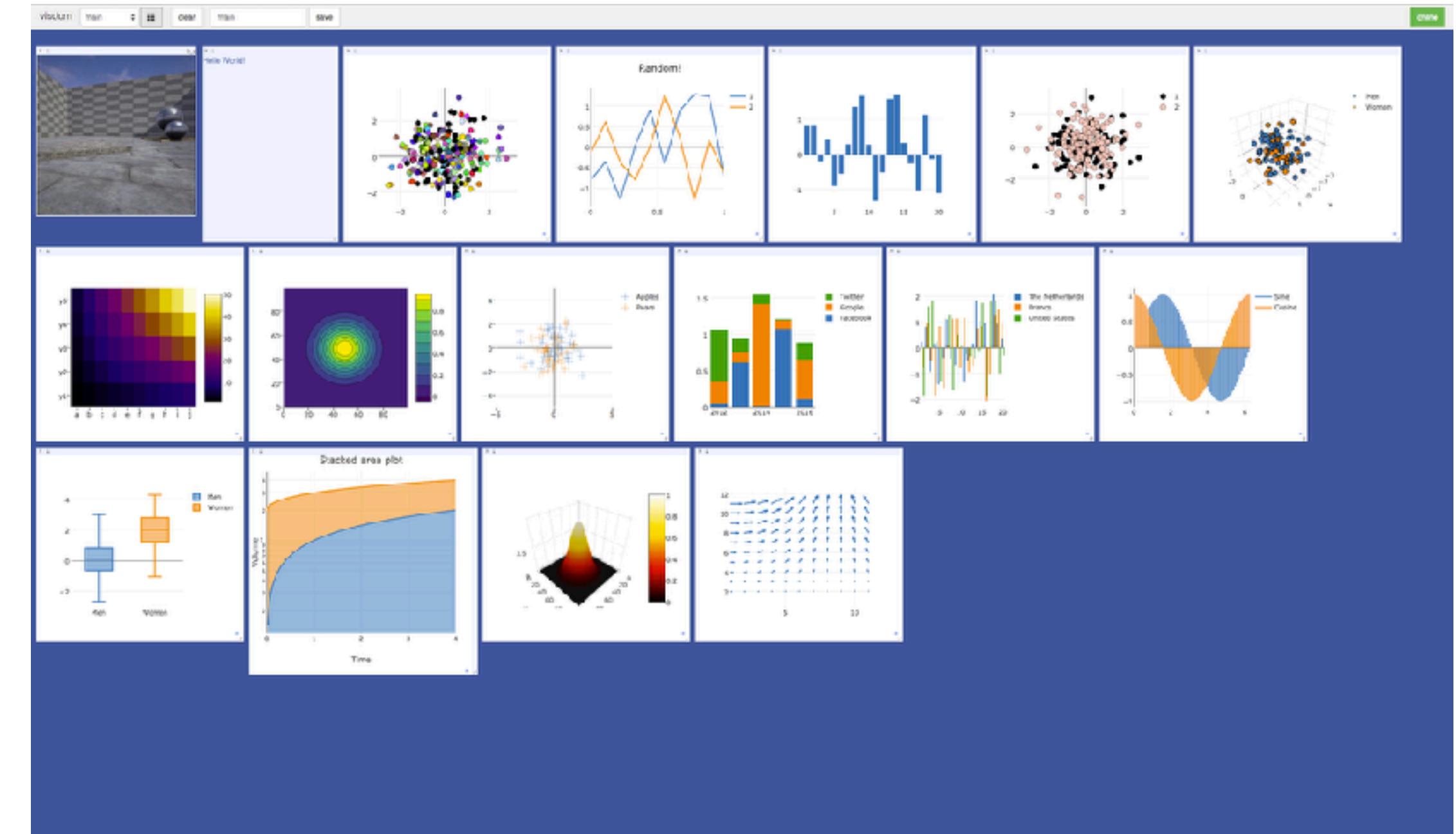


# Visualization



Tensorboard-PyTorch

[github.com/lanpa/tensorboard-pytorch](https://github.com/lanpa/tensorboard-pytorch)



Visdom

[github.com/facebookresearch/visdom](https://github.com/facebookresearch/visdom)



# Debugging

- PyTorch is a Python extension



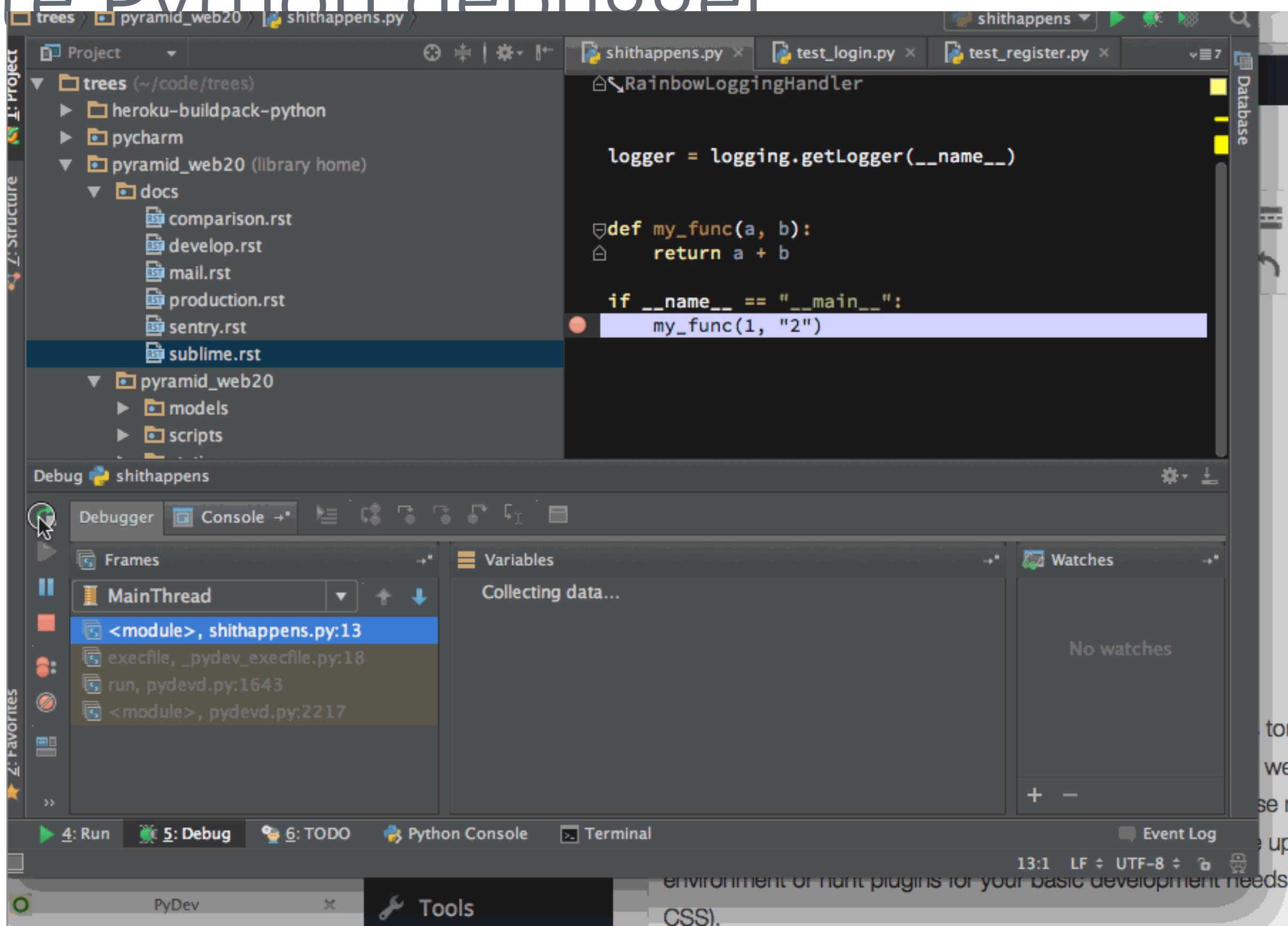
# Debugging

- 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)*

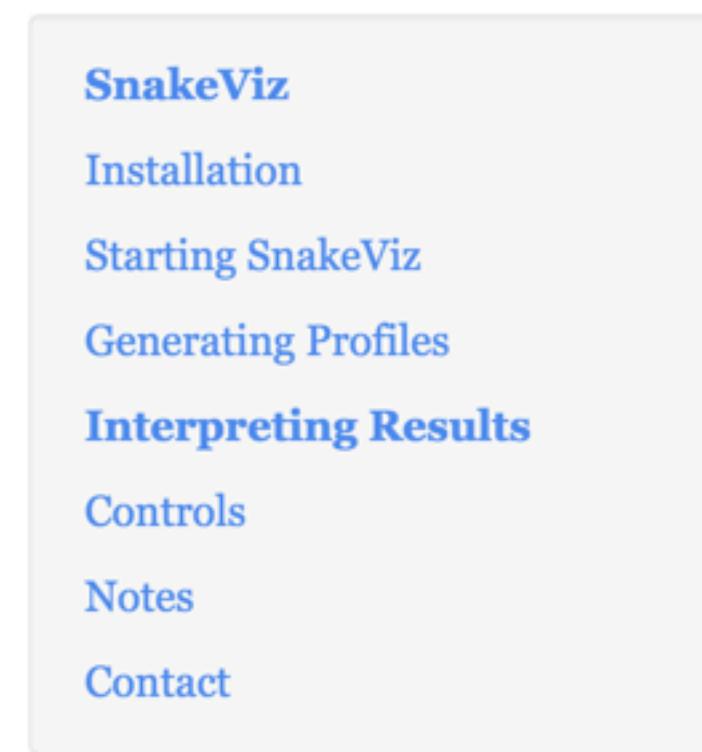


# Identifying bottlenecks

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

SNAKEVIZ

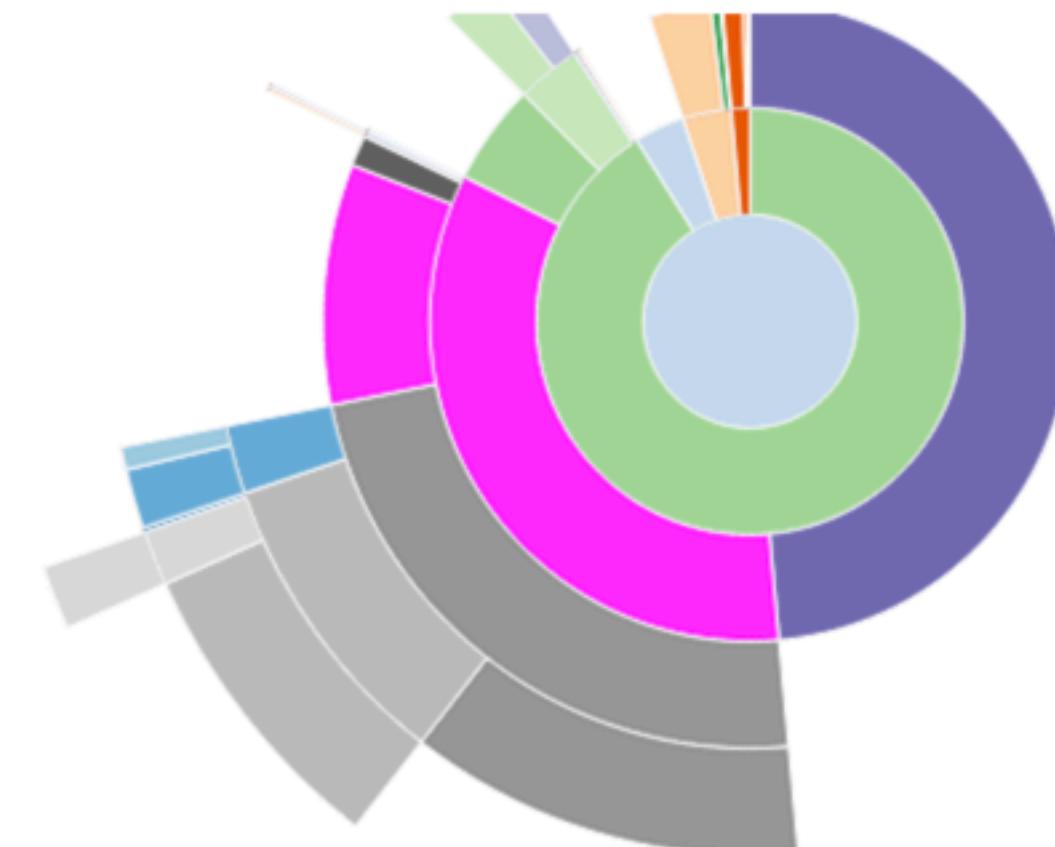
← PREVIOUS    NEXT →    G



## 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



# Ecosystem

- Use the entire Python ecosystem at your will



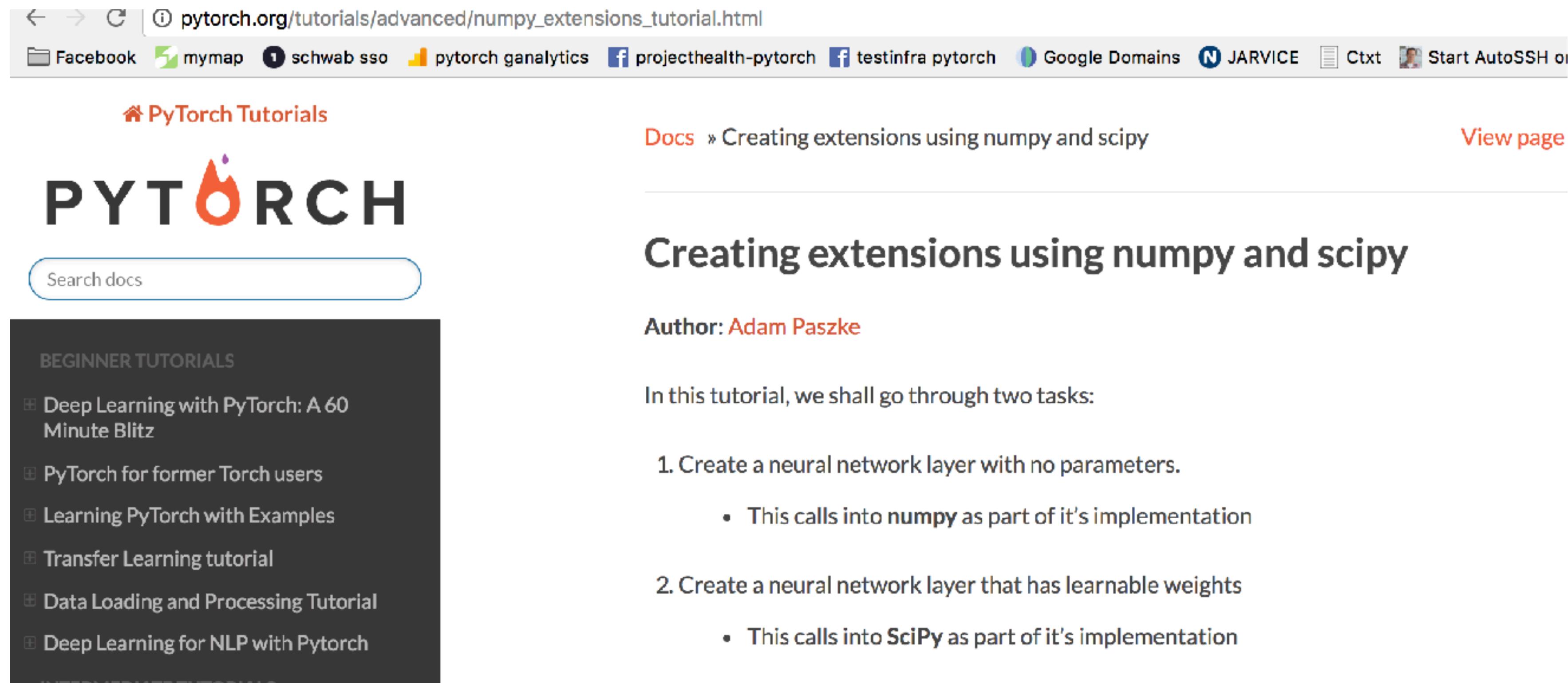
# Ecosystem

- 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.



# Machine Learning Ecosystem

## Shared model zoo (pretrained models)

We provide pre-trained models, 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)
squeezenet = models.squeezeNet1_0(pretrained=True)
vgg16 = models.vgg16(pretrained=True)
densenet = models.densenet161(pretrained=True)
inception = models.inception_v3(pretrained=True)
```

torchvision



# Machine Learning Ecosystem

Shared model zoo (pretrained models)

Top1 Accuracy

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

[github.com/aaron-xichen/pytorch-playground](https://github.com/aaron-xichen/pytorch-playground)



# Machine Learning Ecosystem

Probabilistic programming



<http://pyro.ai/>



[github.com/probtorch/probtorch](https://github.com/probtorch/probtorch)



# Machine Learning 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>



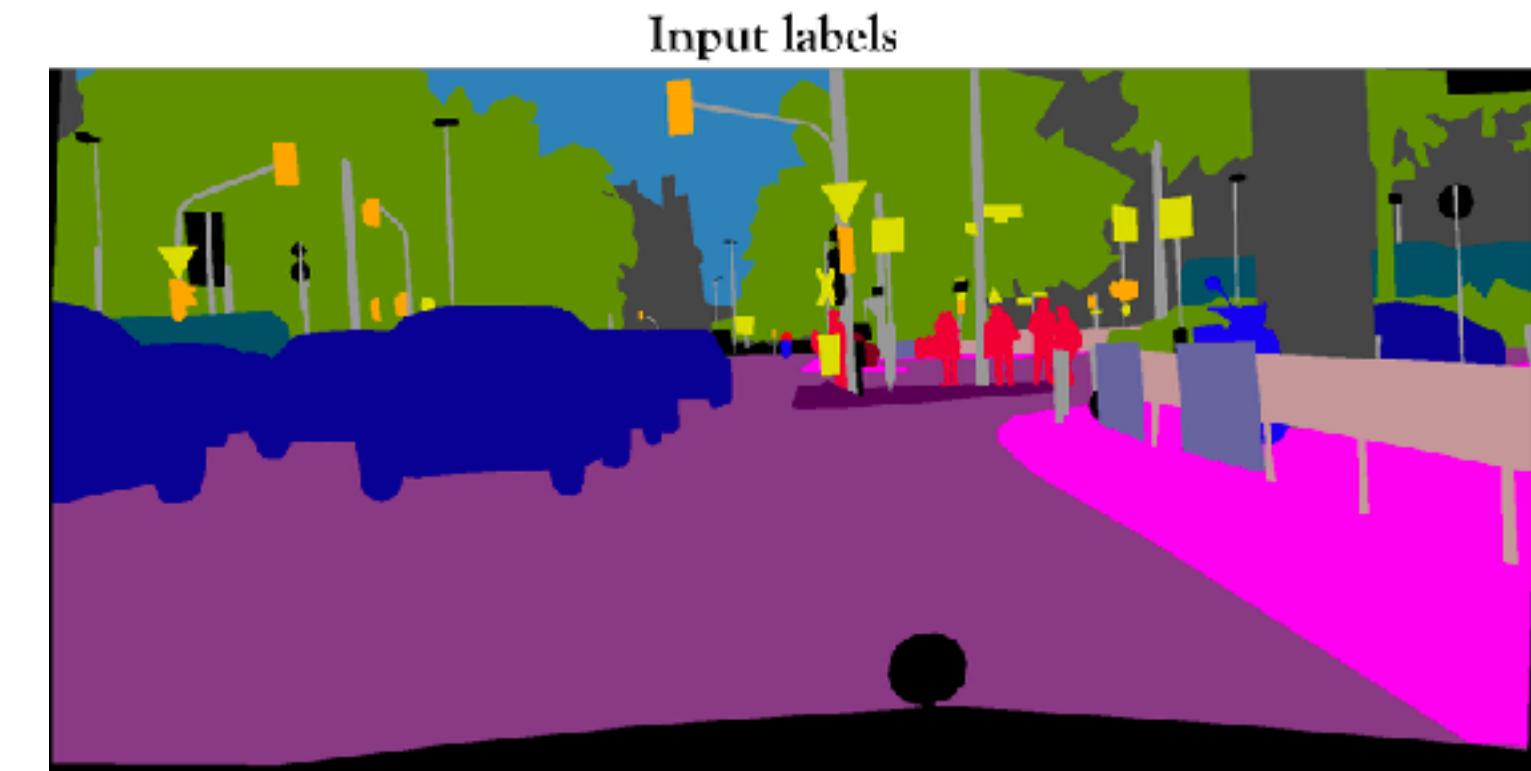
# Machine Learning Ecosystem

## Image-to-Image Translation



pix2pix & CycleGAN

[github.com/junyanz/pytorch-CycleGAN-and-pix2pix](https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix)



Synthesized image



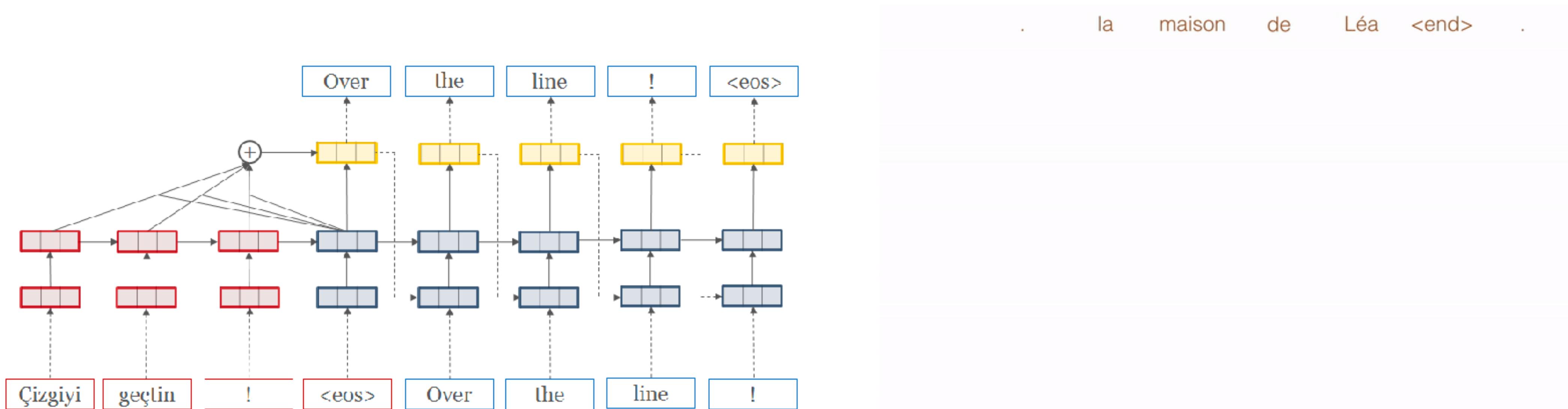
pix2pixHD

[github.com/NVIDIA/pix2pixHD](https://github.com/NVIDIA/pix2pixHD)



# Machine Learning Ecosystem

## Machine Translation



**OpenNMT-py: Open-Source Neural Machine Translation**

[github.com/OpenNMT/OpenNMT-py](https://github.com/OpenNMT/OpenNMT-py)

**FAIR Sequence-to-Sequence Toolkit**

[github.com/facebookresearch/fairseq-py](https://github.com/facebookresearch/fairseq-py)



# Machine Learning Ecosystem

## General Linguistics Tasks

SRL Model   MC Model   TE Model

### Semantic Role Labeling



Enter text or

Choose an example... ▾

#### Sentence

E.g. "John likes and Bill hates ice cream."

RUN >

### Machine Comprehension

Machine Comprehension (MC) answers natural language questions by selecting an answer span within an evidence text. The AllenNLP toolkit provides the following MC visualization, which can be used for any MC model in AllenNLP. This page demonstrates a reimplementation of BiDAF (Seo et al., 2017), or Bi-Directional Attention Flow, a widely used MC baseline that achieved state-of-the-art accuracies on the SQuAD dataset (Wikipedia sentences) in early 2017.

Enter text or Choose an example... ▾

#### Passage

discarded. No completely reusable orbital launch system has ever been created. Two partially reusable launch systems were developed, the Space Shuttle and Falcon 9. The Space Shuttle was partially reusable: the orbiter (which included the Space Shuttle main engines and the Orbital Maneuvering System engines), and the two solid rocket boosters were reused after several months of refitting work for each launch. The external tank was discarded after each flight.

#### Question

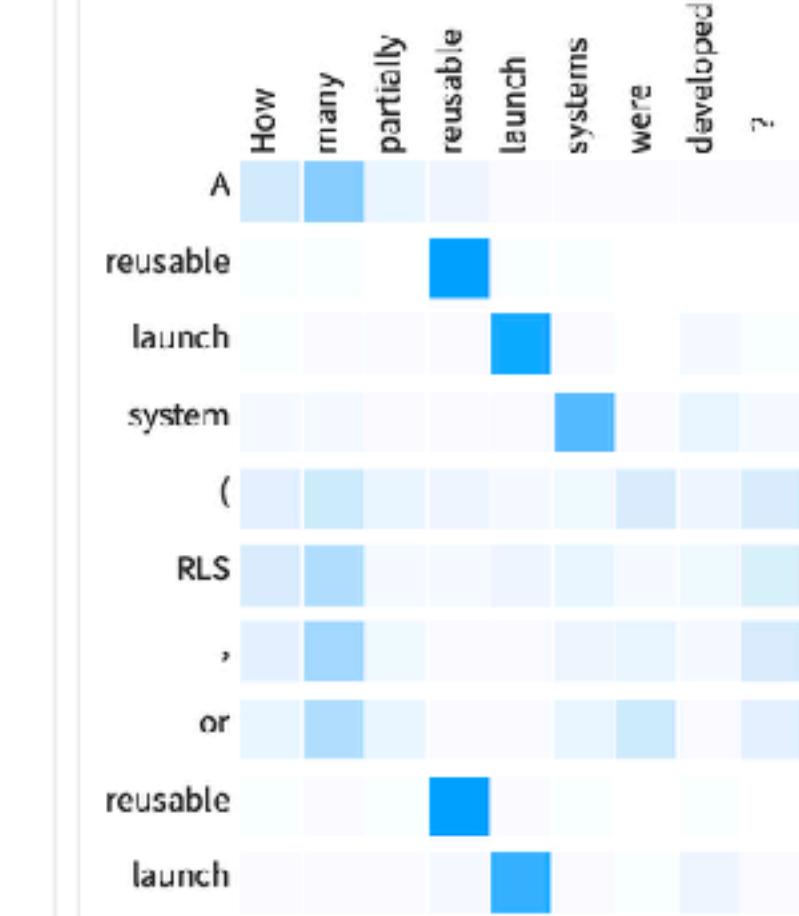
How many partially reusable launch systems were developed?

external tank was discarded after each flight.

#### Model internals (beta)

##### Passage to Question attention

For every passage word, the model computes an attention over the question words. This heatmap shows that attention, which is normalized for every row in the matrix.



<http://allennlp.org/>

AllenNLP



# Machine Learning Ecosystem

## 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.

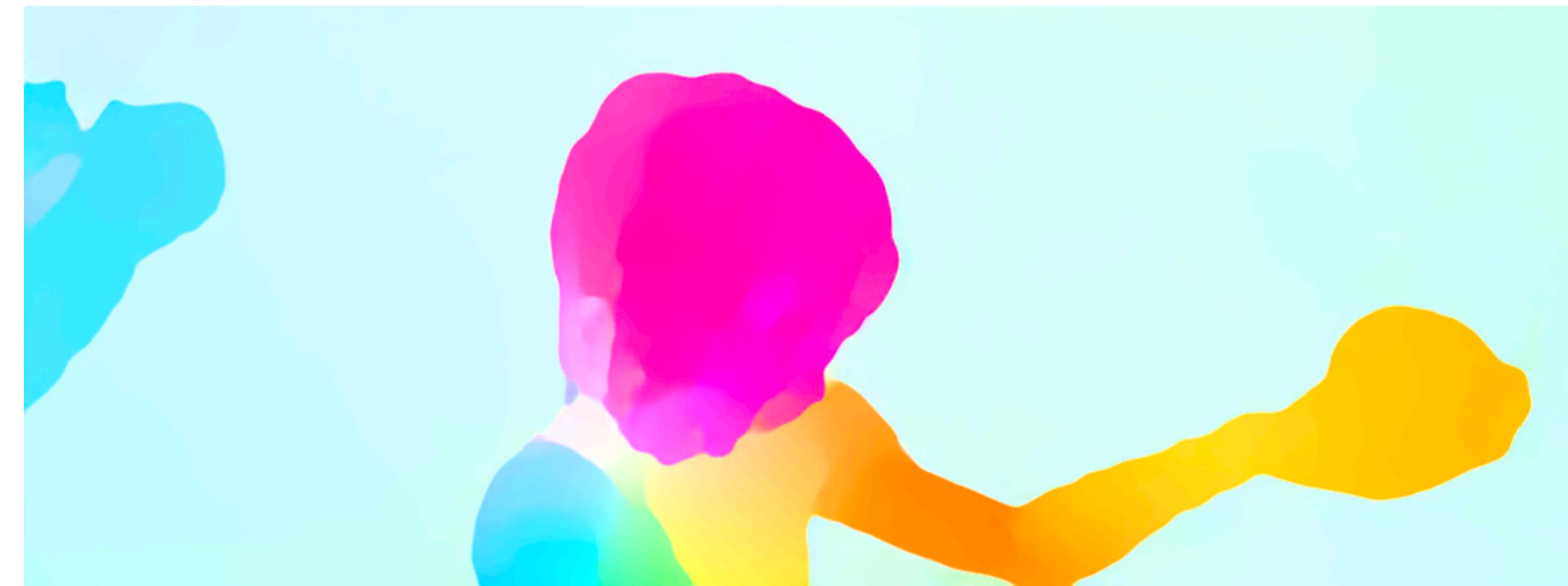
A dreary, incoherent, self-indulgent mess of a movie in which a bunch of pompous windbags drone on inanely for two hours... a cacophony of pretentious, meaningless prattle.

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



# Machine Learning Ecosystem

Optical Flow Estimation



FlowNet 2.0

[github.com/NVIDIA/flownet2-pytorch](https://github.com/NVIDIA/flownet2-pytorch)



# and many more...

- Distributed training
- Profiling
- Extending autograd
- C++ interface (ATen) and C++ extensions
- ONNX
- JIT
- ...



With ❤ from



<http://pytorch.org>

