facebook

Artificial Intelligence Research

人工智能研究部门

人工智能公开课 GMIC 2018

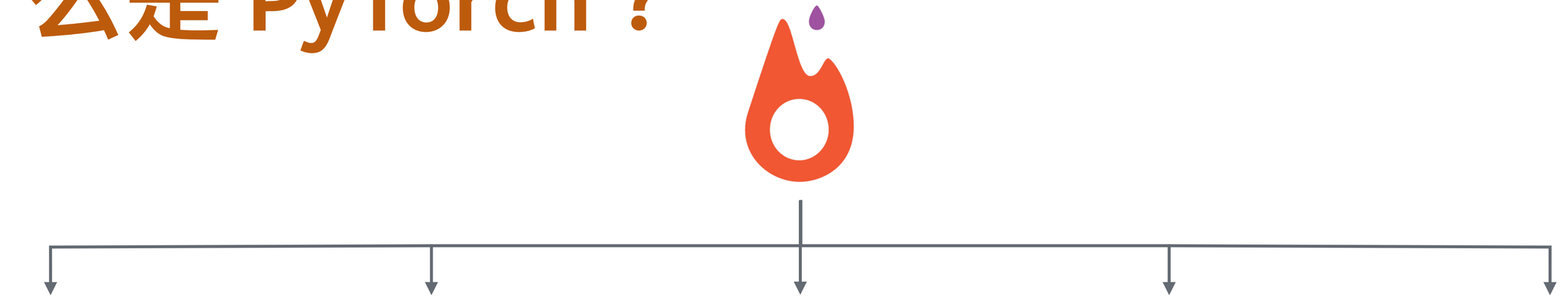
Tongzhou Wang, Will Feng, Ailing Zhang, Teng Li

Facebook AI Research

人工智能研究部门

什么是 PyTorch？

什么是 PyTorch？



高效的多维矩阵库
GPU支持

自动微分引擎

快速的机器学习模块

基于梯度的优化器

机器学习生态环境

CUDA

类似于NumPy的接口

概率模型

深度学习

强化学习

⋮

数据加载

可视化

图像和文字数据处理库

⋮

Tensor 张量 (多维矩阵) 库

Tensor 张量 (多维矩阵) 库

- 类似于NumPy 的 API (np.ndarray <-> torch.Tensor)

- 支持在NVIDIA GPU上的高效运算

Tensor 张量 (多维矩阵) 库

- 类似于NumPy 的 API (np.ndarray <-> torch.Tensor)
  - 轻易在不同的设备（CPU/GPU）创建各种数据类型的Tensor
  - 科学计算函数（例如线性代数）
  - 和 np.ndarray 之间的快速转换

Tensor 张量 (多维矩阵) 库

- 类似于NumPy 的 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)
```

Tensor 张量 (多维矩阵) 库

- 支持在NVIDIA GPU上的高效运算
  - GPU上的快速科学计算函数
  - 简洁的设备管理
  - 和CPU Tensor的无缝转换

Tensor 张量 (多维矩阵) 库

- 支持在NVIDIA GPU上的高效运算

-

```
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]])
```

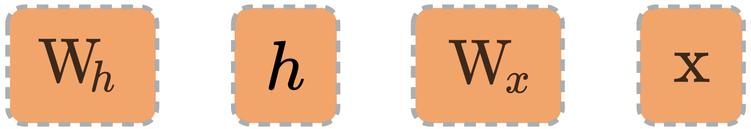
自动微分 (autograd) 引擎

自动微分引擎

每一步运算都在逐步构建一个图

自动微分引擎

每一步运算都在逐步构建一个图



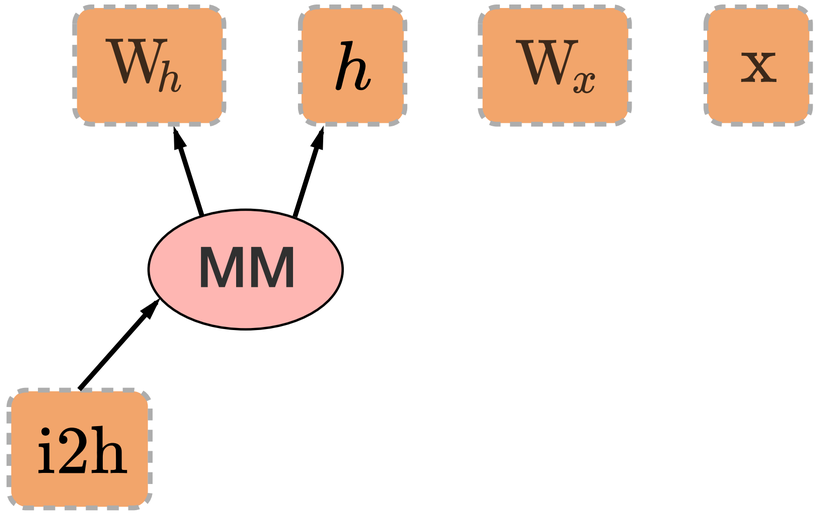

```
x = torch.randn(1, 10)
prev_h = torch.randn(1, 20)
W_h = torch.randn(20, 20)
W_x = torch.randn(20, 10)
```

自动微分引擎

每一步运算都在逐步构建一个图

```
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())
```

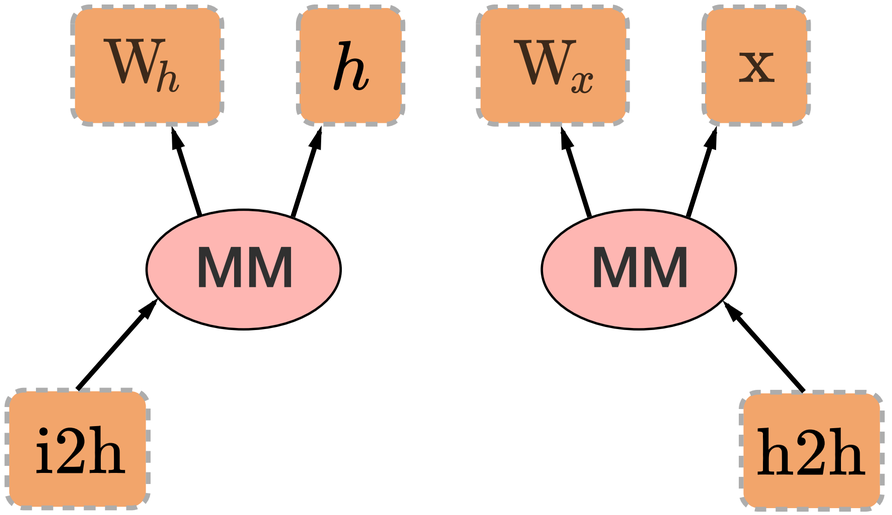


自动微分引擎

每一步运算都在逐步构建一个图

```
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())
```

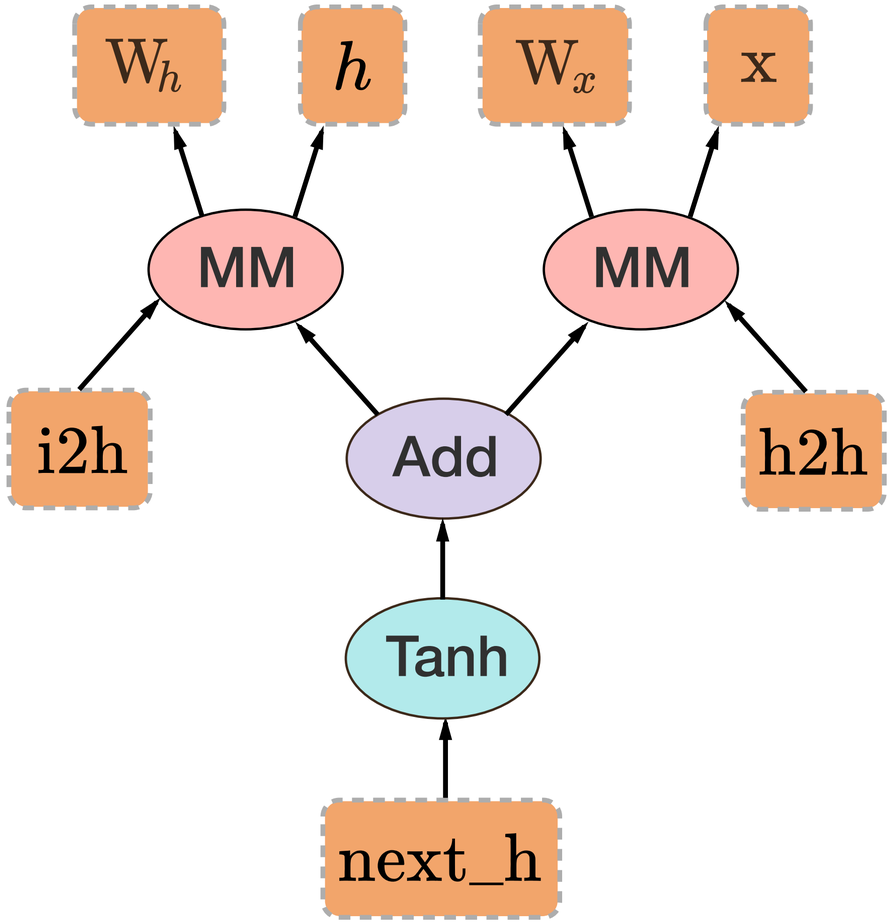


自动微分引擎

每一步运算都在逐步构建一个图

```
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()
```



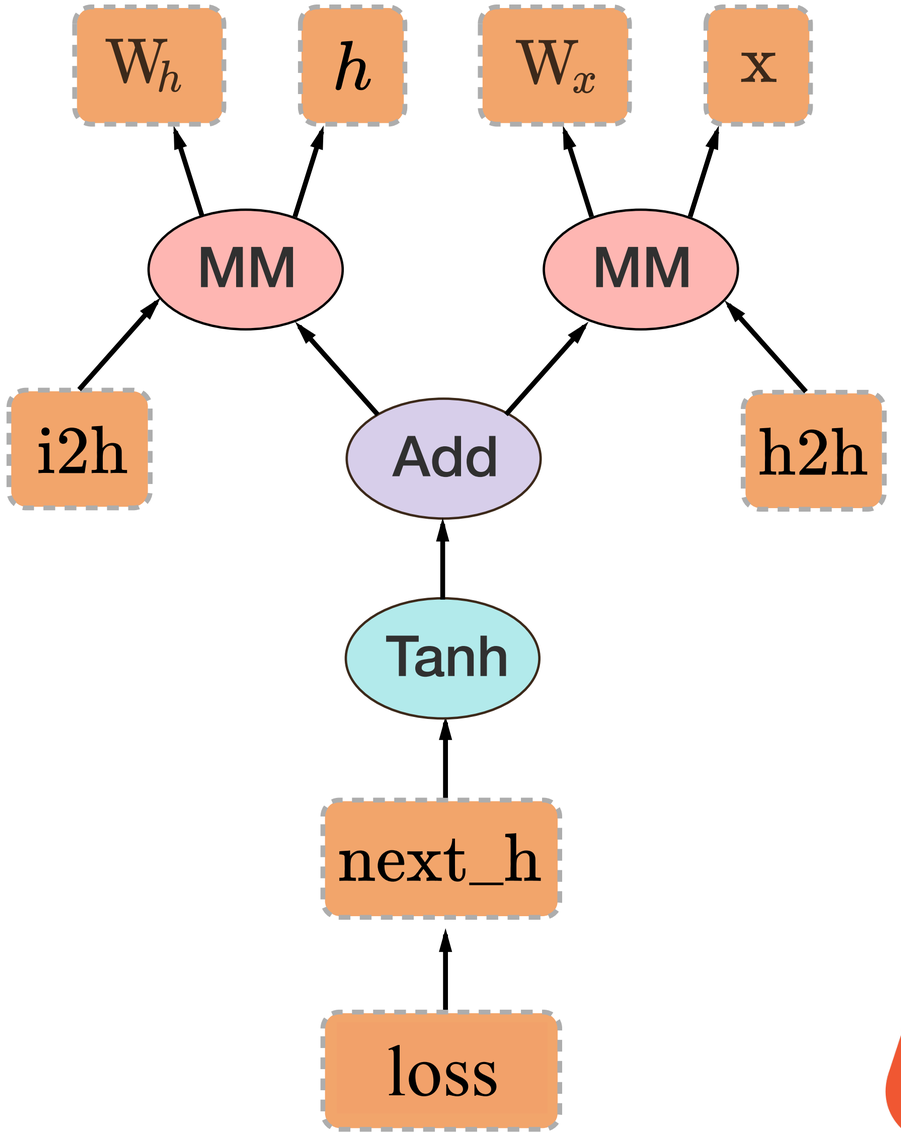
自动微分引擎

每一步运算都在逐步构建一个图

```
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()
```



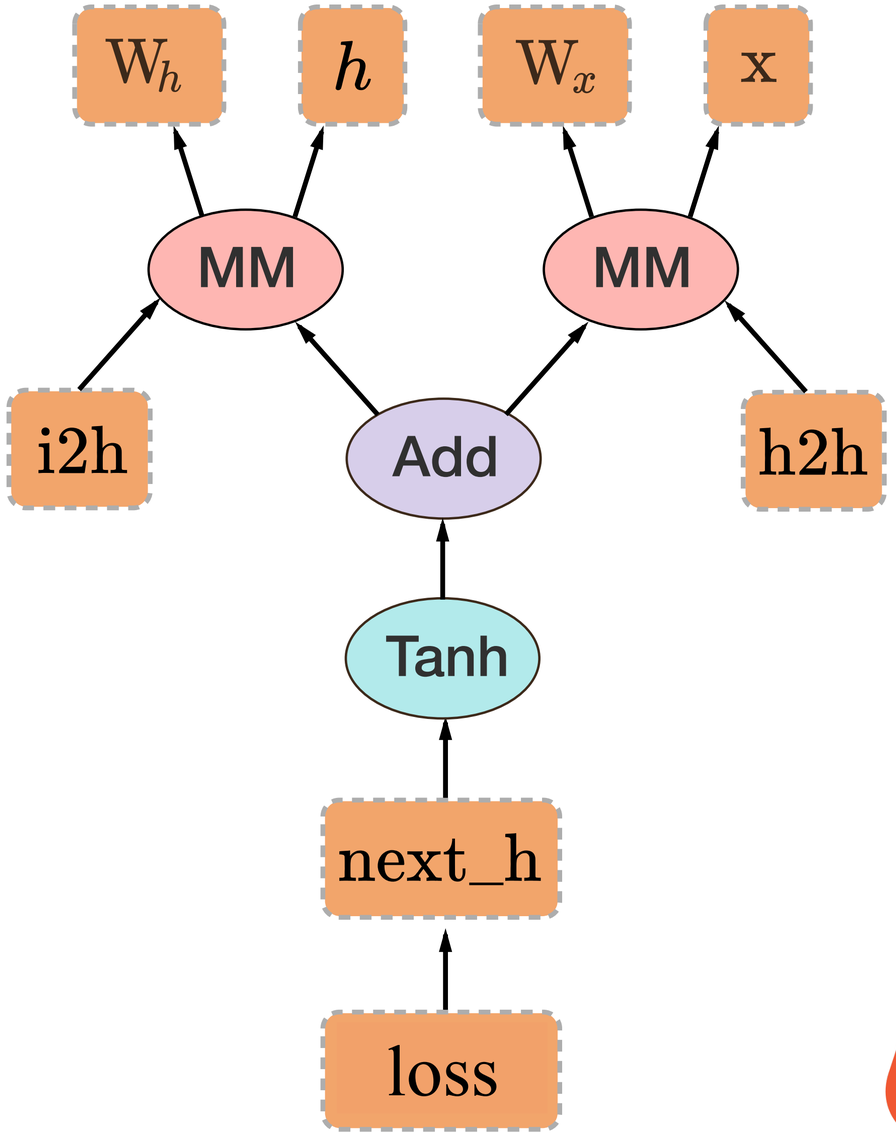
自动微分引擎

每一步运算都在逐步构建一个图

```
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!
```



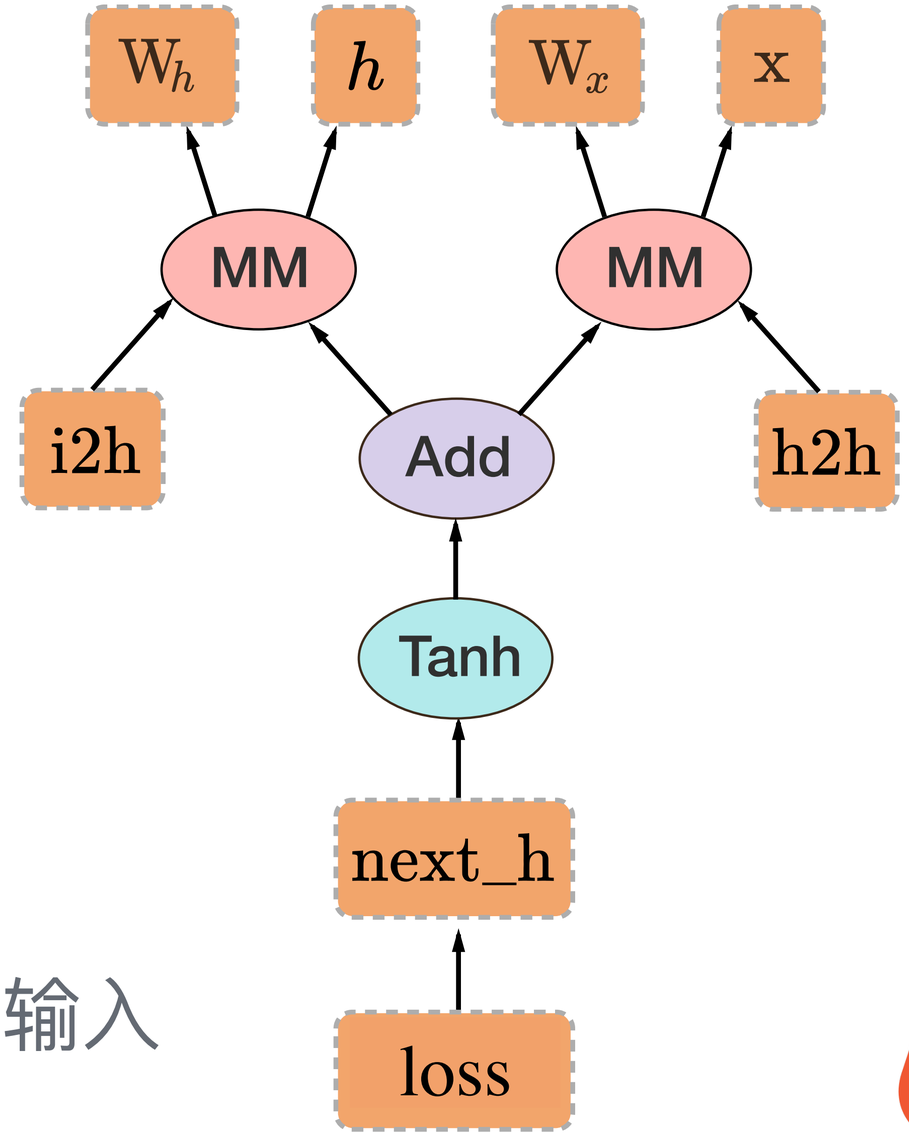
自动微分引擎

每一步运算都在逐步构建一个图

```
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!
```



从 loss 一步步逆序方向传播，得到关于输入Tensor的梯度。

自动微分引擎

- 每一步运算都在逐步构建一个图
  - 可运用Python的控制流来构建图（例如 for-循环）
- 拥有高效 backward 实现的运算
  - `torch.matmul`, `torch.fft`, `torch.svd`, `torch.trtrs`, 等等。
- 自动反向遍历运算图来求微分
  - 高阶导数 (backward 遍历也是一个图)
  - 涵盖多个设备的图（例如 多GPU）

高效的机器学习模块

高效的机器学习模块

- 深度学习
  - `torch.nn.*`
- 强化学习、概率模型
  - `torch.distributions.*`
- 和更多...
  - 科学计算函数 + `autograd`

神经网络

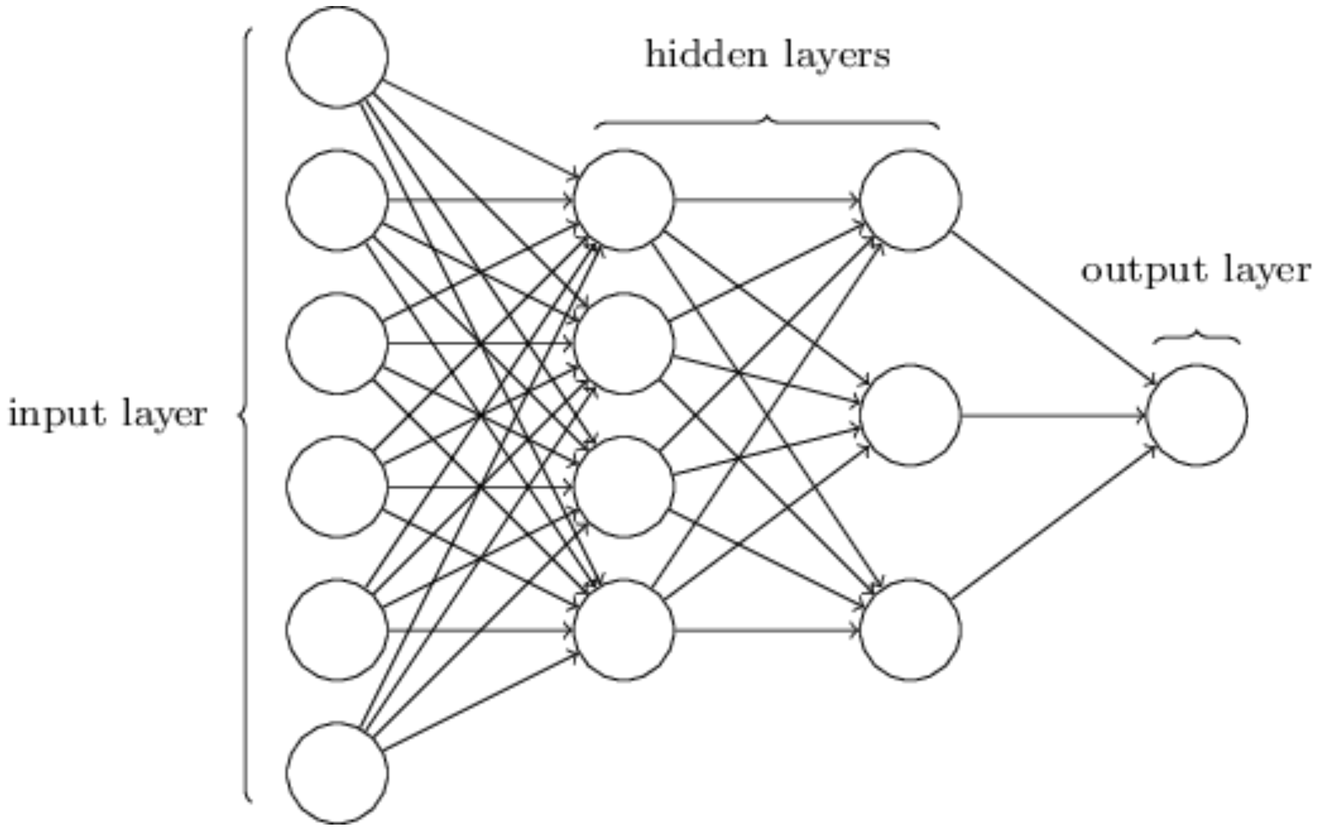
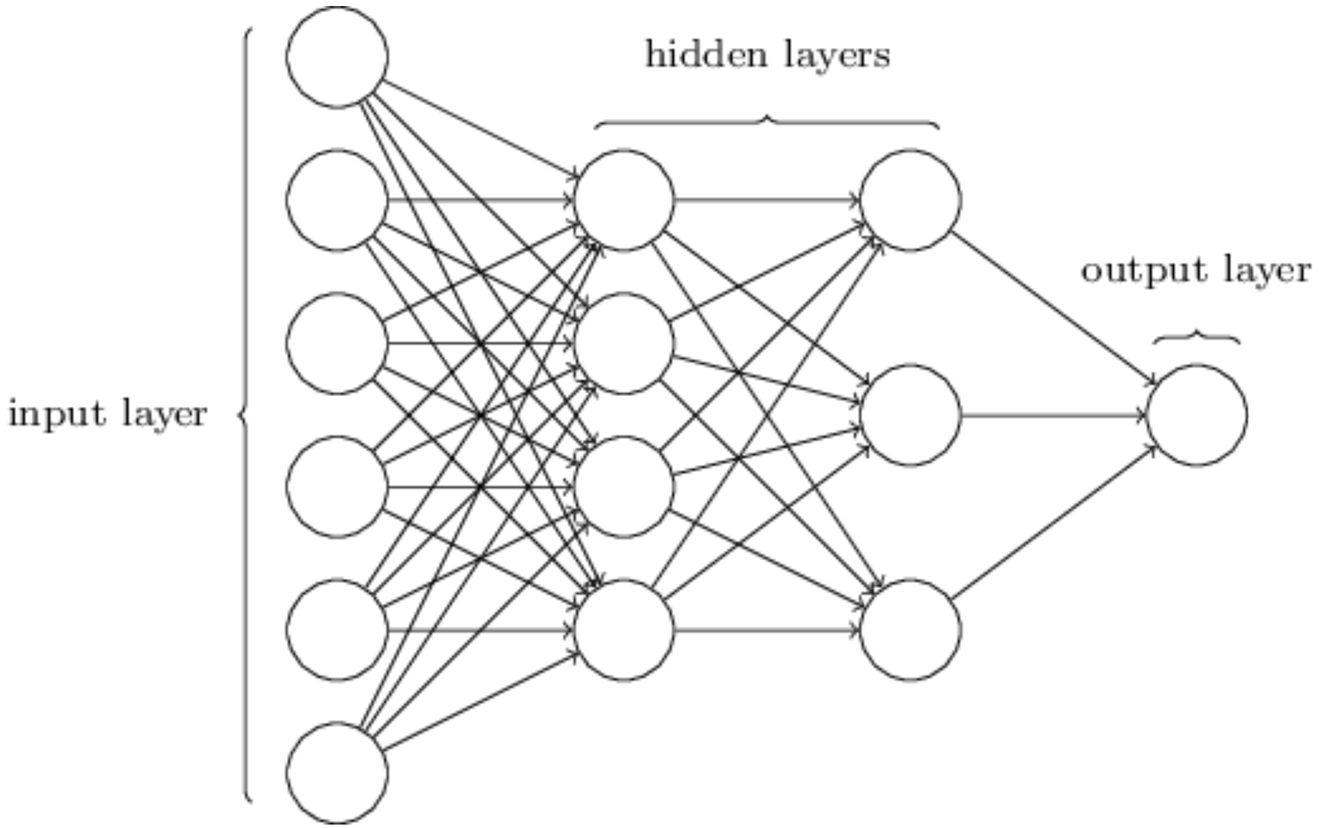


Figure by Md. Rezaul Karim

神经网络




```python
import torch.nn as nn

my_network = nn.Sequential(
    nn.Linear(6, 4),
    nn.Sigmoid(),
    nn.Linear(4, 3),
    nn.Sigmoid(),
    nn.Linear(3, 1),
)
```

Figure by Md. Rezaul Karim

卷积神经网络 (CNN)

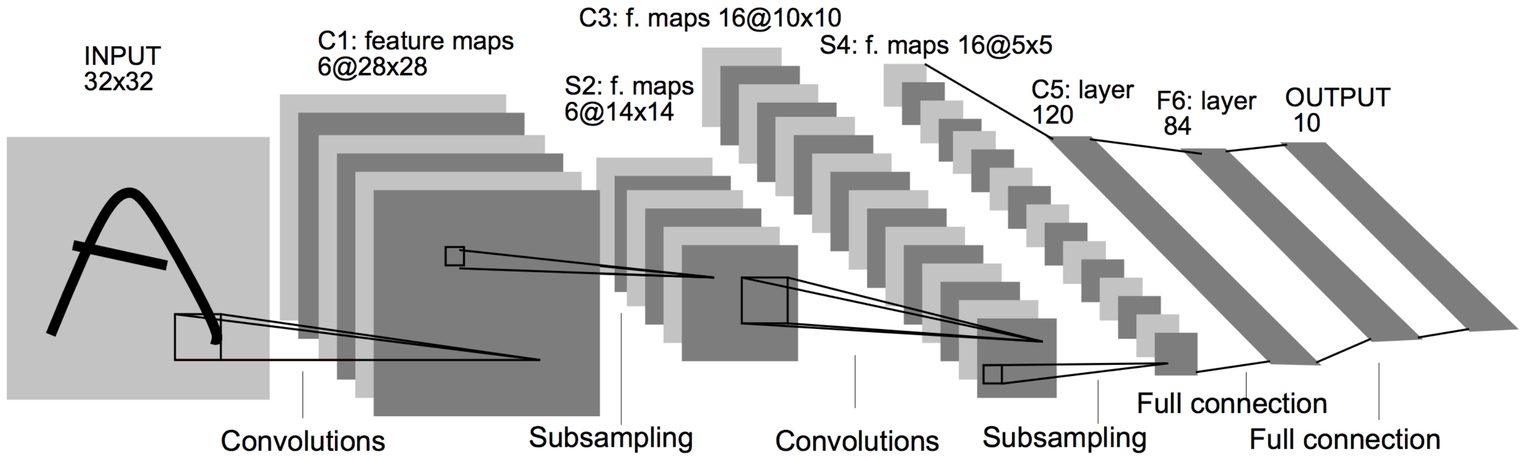
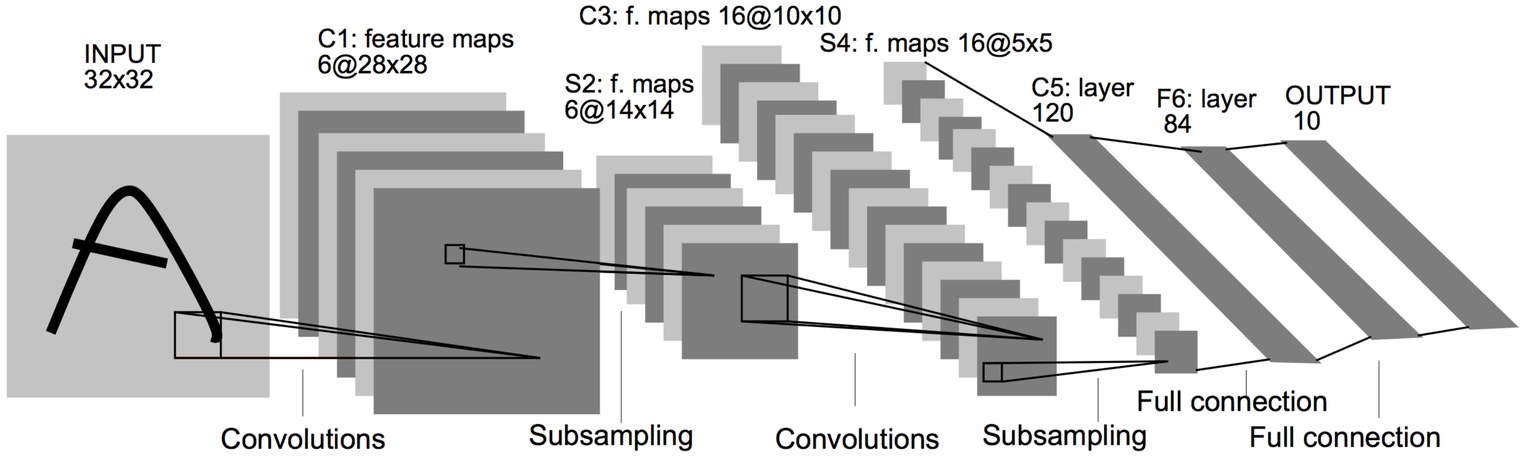


Figure by Yann LeCun et al.

卷积神经网络 (CNN)




```python
import torch.nn as nn

class LeNet5(nn.Module):
    def __init__(self):
        super(LeNet5, self).__init__()
        self.feature = nn.Sequential(
            nn.Conv2d(1, 6, kernel_size=5),
            nn.Tanh(),
            nn.MaxPool2d(2),
            nn.Conv2d(6, 16, kernel_size=5),
            nn.Tanh(),
            nn.MaxPool2d(2),
        )
        self.fc = nn.Sequential(
            nn.Linear(400, 120),
            nn.Tanh(),
            nn.Linear(120, 84),
            nn.Tanh(),
            nn.Linear(84, 10),
        )

    def forward(self, input):
        features = self.feature(input)
        features = features.view(-1, 400)  # flatten
        return self.fc(features)

my_cnn = LeNet5()
```

Figure by Yann LeCun et al.

递归神经网络 (RNN)

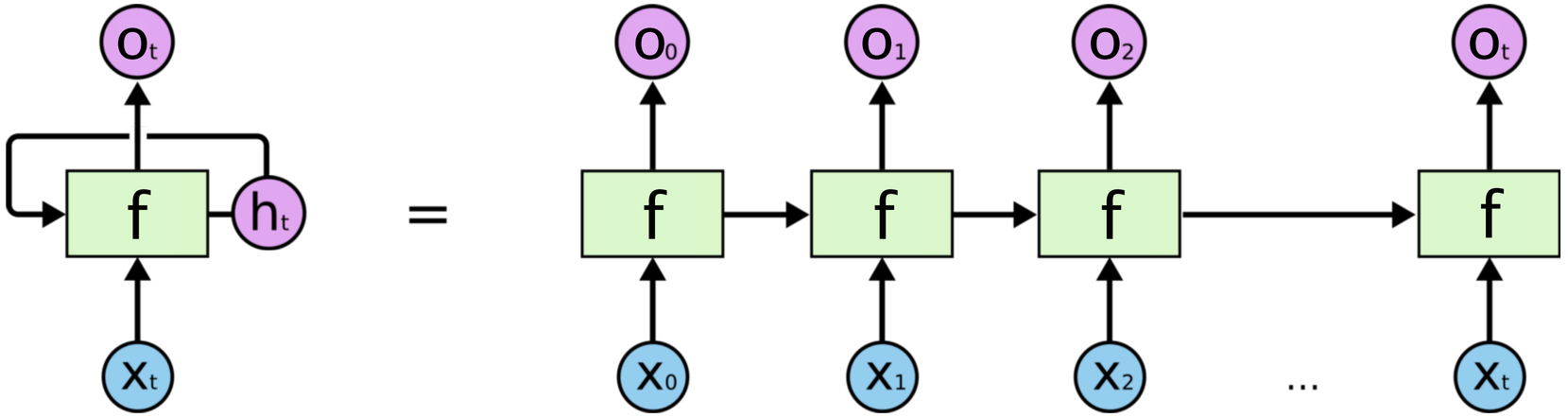
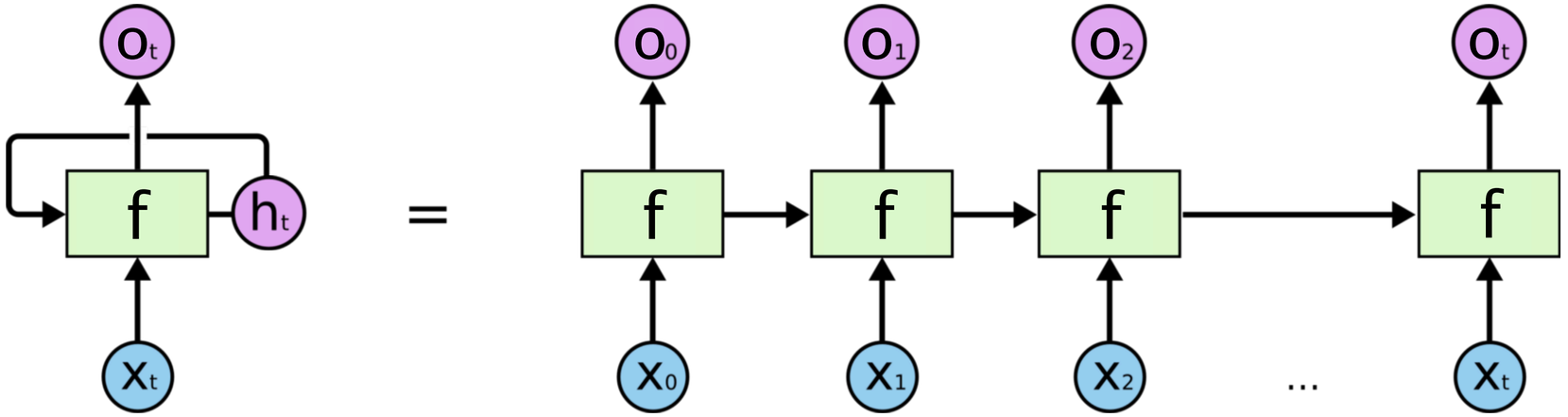


Figure originally from Pranoy Radhakrishnan

递归神经网络 (RNN)




```python
import torch.nn as nn

my_rnn = nn.LSTM(input_size=10, hidden_size=10)
```

Figure originally from Pranoy Radhakrishnan

基于梯度的优化器库

基于梯度的优化器库

- 最先进和最广泛使用的优化算法
  - `torch.optim.*`
  - SGD, Adam, RMSProp, L-BFGS 等等。
- Learning Rate 规划器
  - `torch.optim.lr_scheduler.*`
- 易于扩展的 API

```
optimizer = torch.optim.Adam(model.parameters())
criterion = nn.CrossEntropyLoss()  # 损失函数

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

机器学习生态系统

机器学习流程

加载数据

建立模型

训练循环

存储模型

Python + PyTorch - 一个可以满足这些所有需求的环境

与外界环境交互

优化

GPU训练

建立baseline

数据载入

- 每个数据集都有所不同
- 需要用不同的方式来处理和归一化
- 需要多线程的数据载入来满足GPU的速度

数据载入

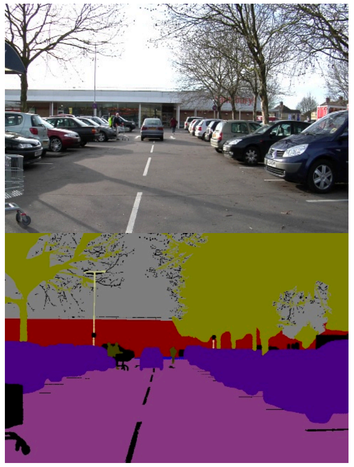
PyTorch 解决方案:

- 在社区内分享格式处理和预处理的代码
  - torchvision（计算机视觉）、torchtext（自然语言处理）、ParlAI (对话) 等等。

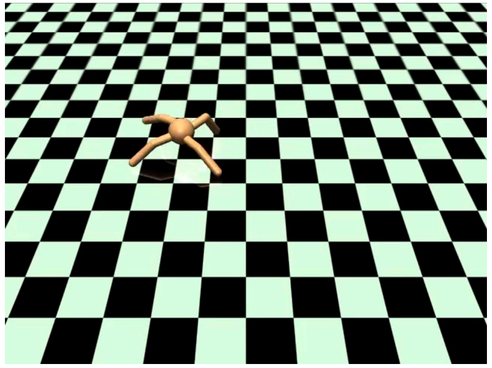
ParlAI

- `torch.utils.data.DataLoader` 实现多进程载入
  - 可以应用在任何数据集上

与外界环境交互



汽车

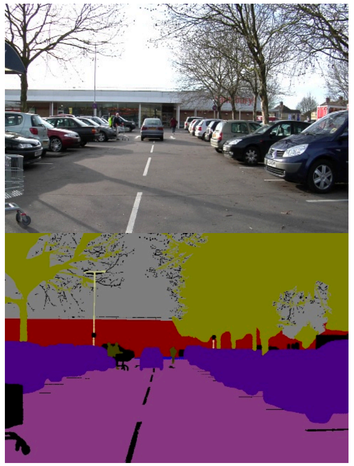


模拟

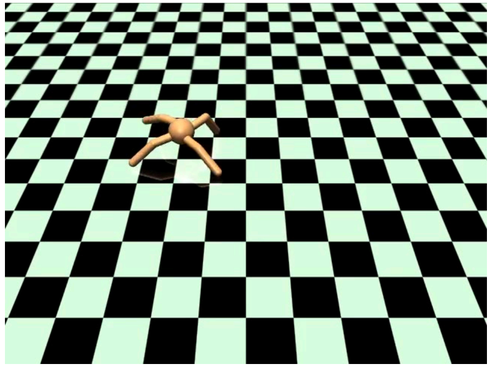
游戏

网络

与外界环境交互



汽车



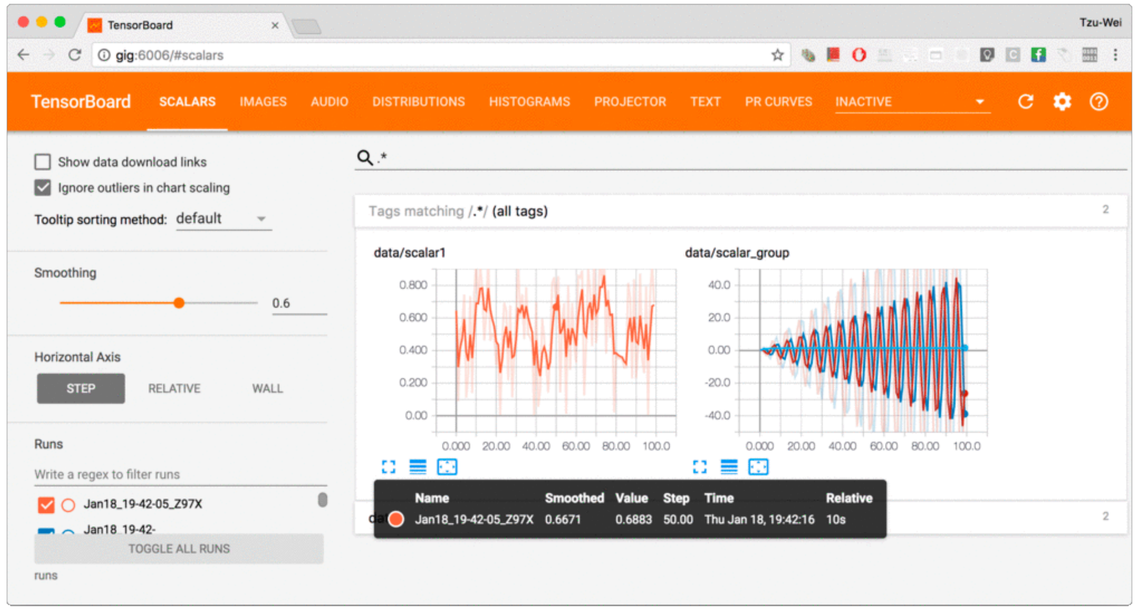
模拟

游戏

网络

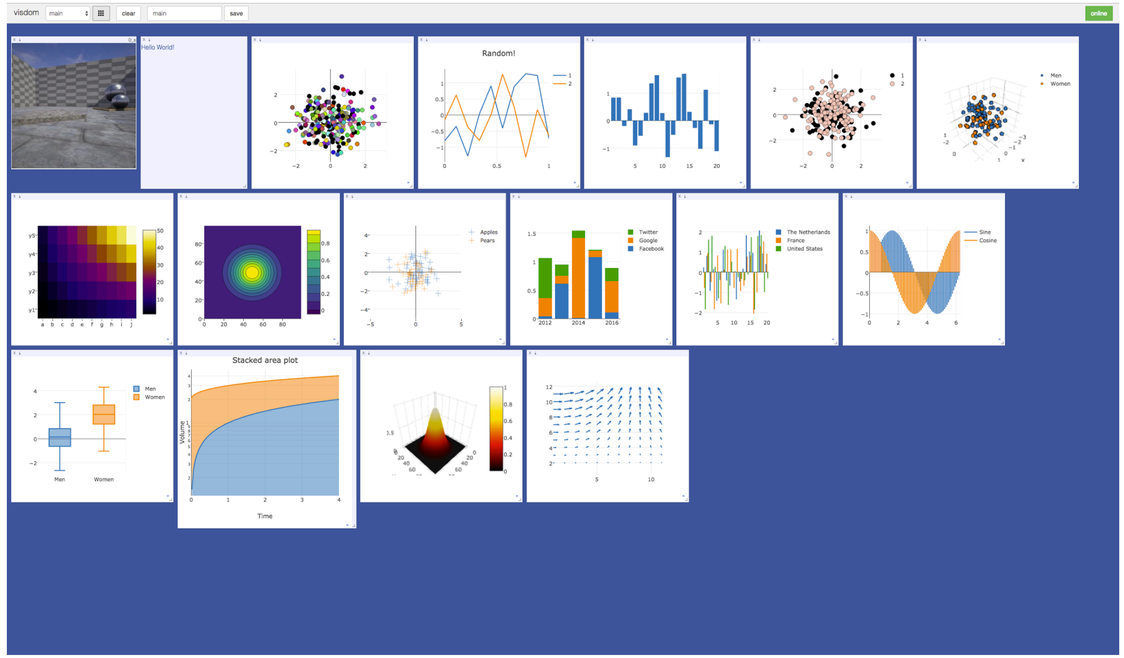
几乎所有环境都有Python API

可视化



Tensorboard-PyTorch

github.com/lanpa/tensorboard-pytorch



Visdom

github.com/facebookresearch/visdom

Debug

- PyTorch 是一个 Python 扩展，每一步语句都真正执行了Tensor运算
- 作为脚本运行、不需要编译
- 使用任何 Python debugger
- 在代码中任何地方都可以暂停
- 可以print任何你想print的东西

```
38  hidden = my_cnn(input)
39  output = hidden.mul(3).sigmoid()  # BUG here? :(
40  print(output)
41  import pdb; pdb.set_trace()
42  loss = output.sum()
43  ...
```

使用pdb的例子

机器学习生态系统

概率编程

http://pyro.ai/

github.com/probtorch/probtorch

机器学习生态系统

共享 model zoo（训练好的模型）

torchvision 提供了很多最先进的计算机视觉模型。只需要声明 `pretrained=True` 即可以得到训练好的模型，方便做分析、迁移学习等。

```
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

机器学习生态系统

共享 model zoo（训练好的模型）

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

机器学习生态系统

语言分析

AllenNLP

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

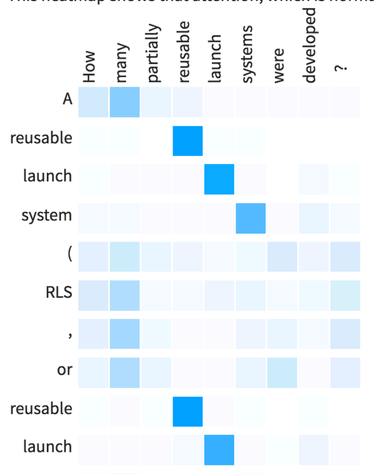
How many partially reusable launch systems were developed?

external tank was discarded after each flight.

Model internals (beta)

Passage to Question attention

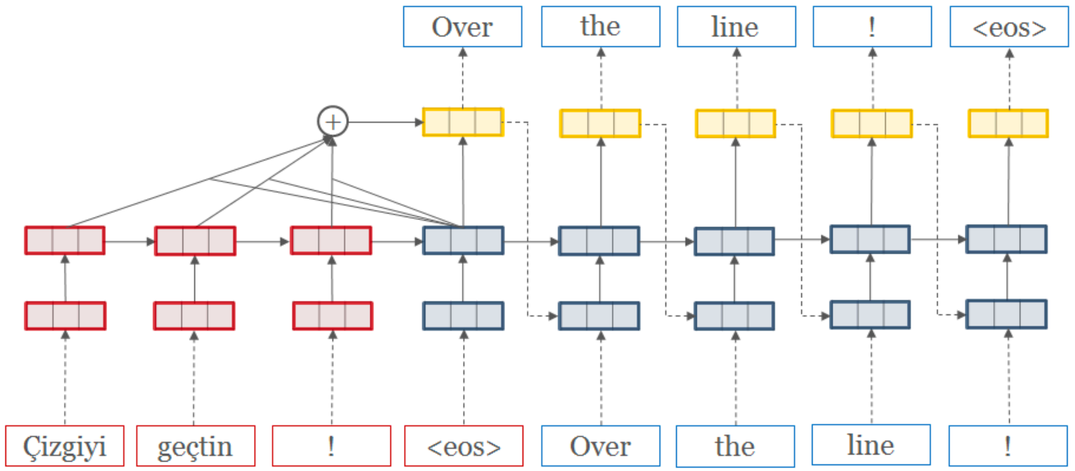
For every passage word, the model computes an attention over the question words. This heatmap shows that attention, which is normalized for every row in the matrix.



http://allennlp.org/

机器学习生态系统

机器翻译



OpenNMT-py: Open-Source Neural Machine Translation

OpenNMT-py: 开源神经网络机器翻译

github.com/OpenNMT/OpenNMT-py

FAIR Sequence-to-Sequence Toolkit

FAIR 序列到序列工具箱

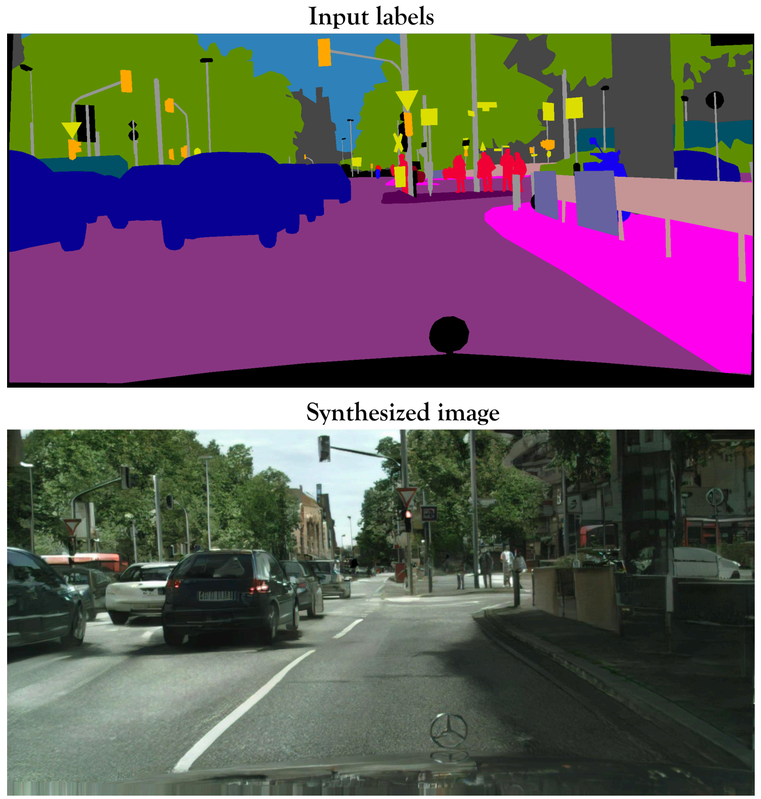
github.com/facebookresearch/fairseq-py

机器学习生态系统

图片到图片

pix2pix & CycleGAN

github.com/junyanz/pytorch-CycleGAN-and-pix2pix



pix2pixHD

github.com/NVIDIA/pix2pixHD

机器学习生态系统

光流（Optical Flow）估计

FlowNet 2.0

github.com/NVIDIA/flownet2-pytorch

和更多...

- 分布式训练
- 代码分析（Profiling）
- 扩展autograd
- C++ 接口 (ATen) and C++ 扩展
- 开放式神经网络交换（ONNX）
- 即时编译（JIT）
- ...


PYTORCH