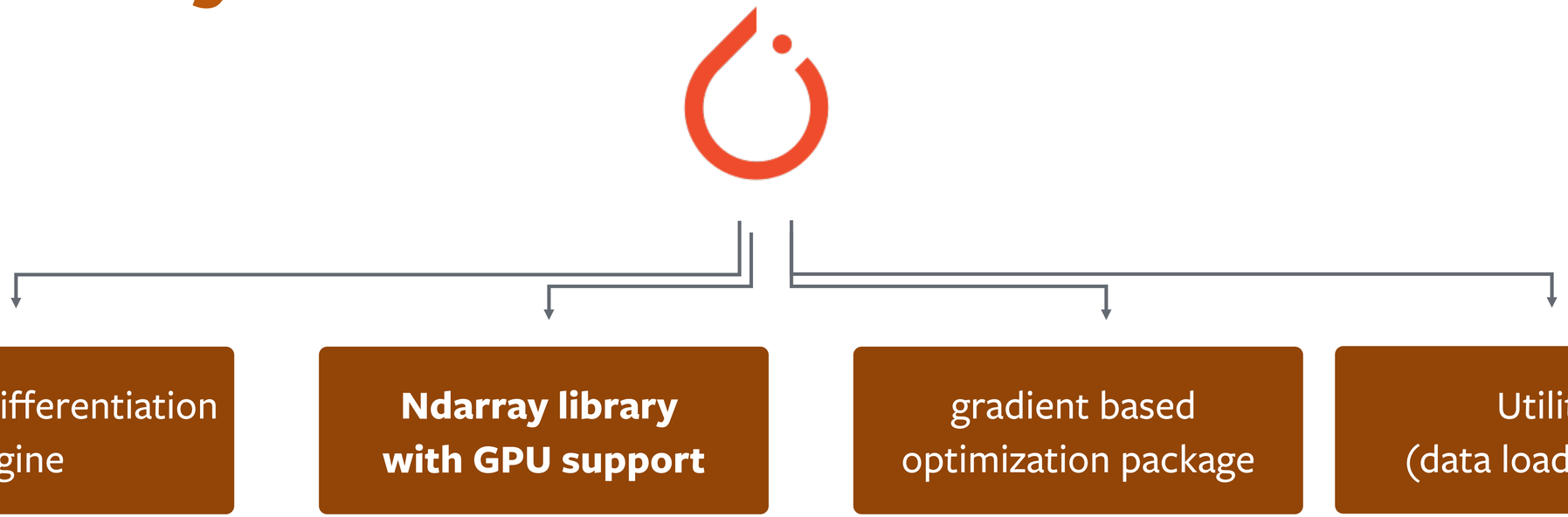

PyTorch

PyTorch

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, Dmytro Dzughalov, Yangqing Jia, Orion Richardson, James Reed & Team

What is PyTorch?



automatic differentiation engine

Ndarray library with GPU support

gradient based optimization package

Utilities (data loading, etc.)

Deep Learning

Reinforcement Learning

Numpy-alternative

ndarray library

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

```numpy
# -*- 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

```pytorch
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

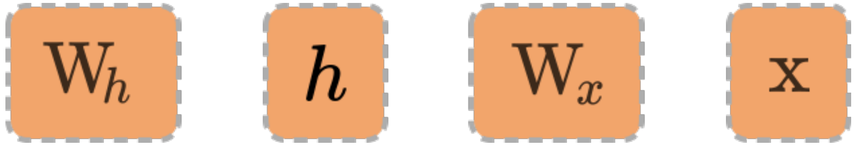
Tensors can be moved onto GPU using the `.cuda` function.

```
# 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

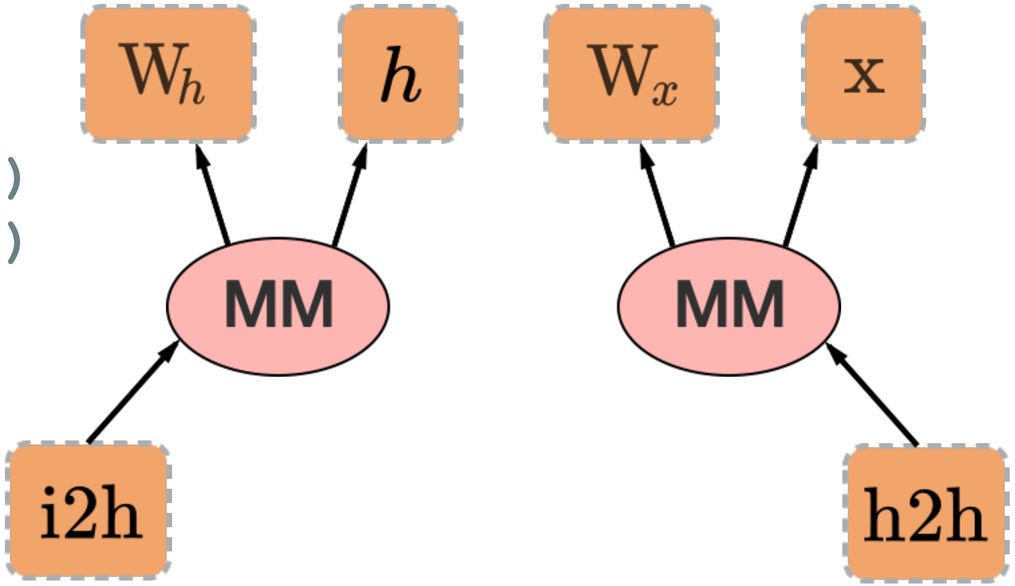
PyTorch Autograd




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

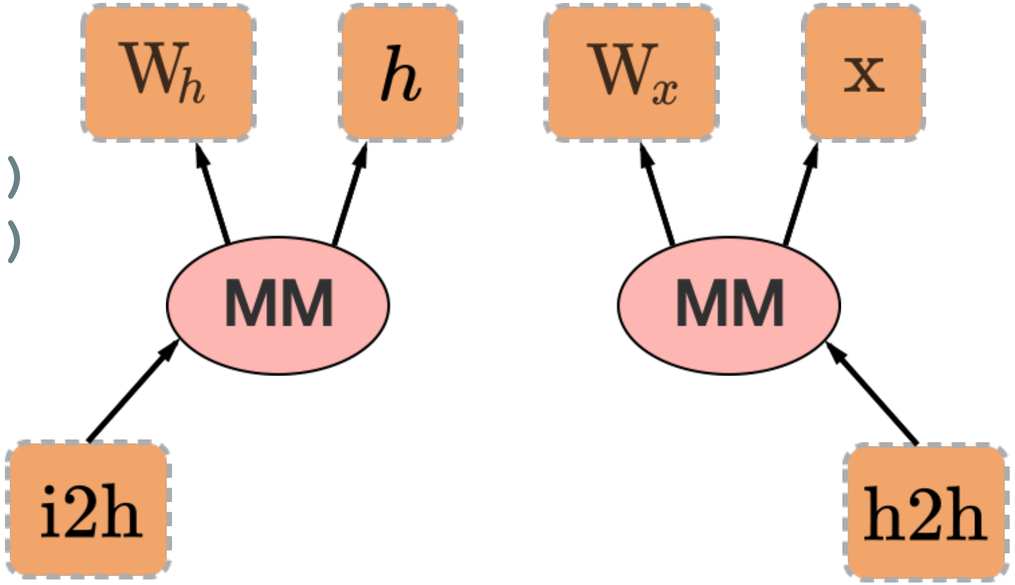
PyTorch Autograd

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


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



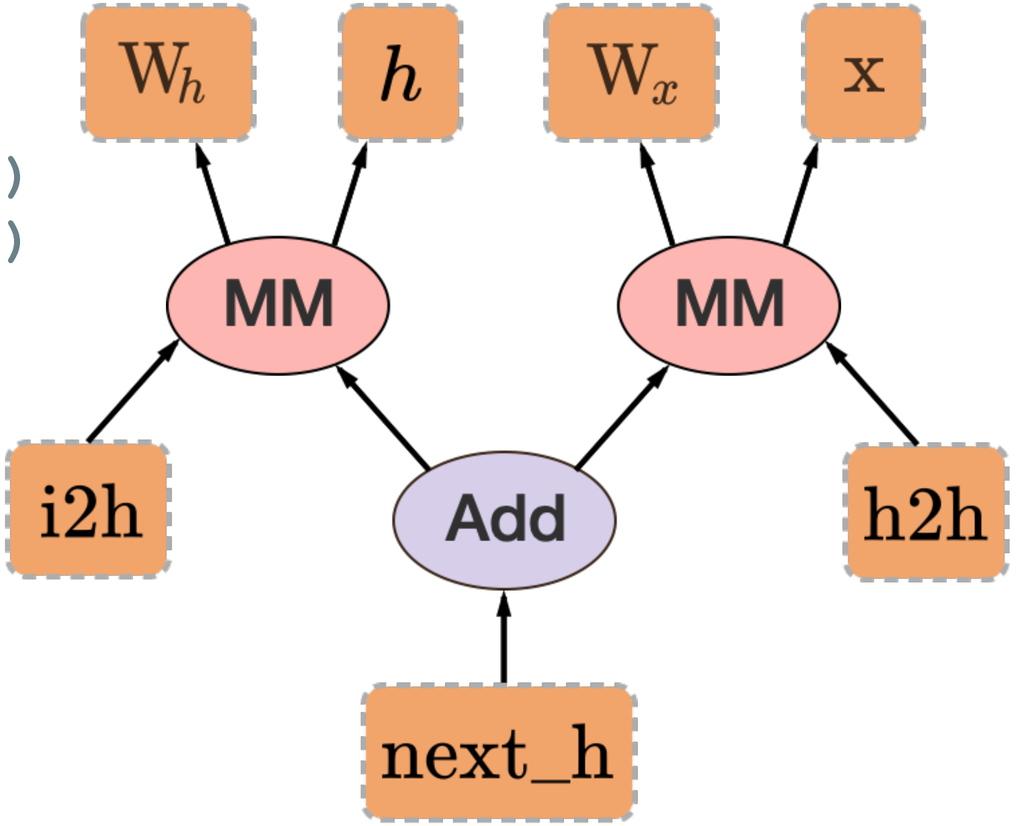
PyTorch Autograd

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


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



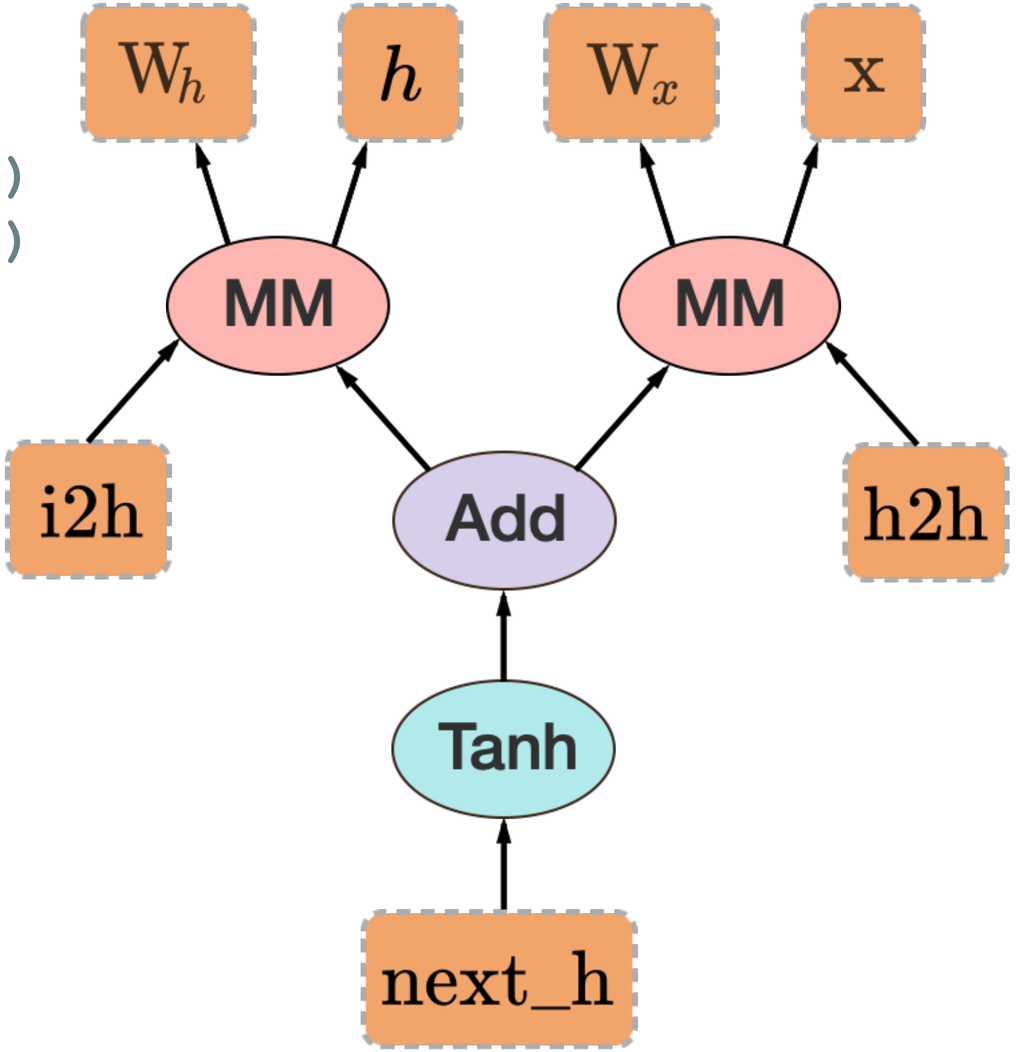
PyTorch Autograd

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


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



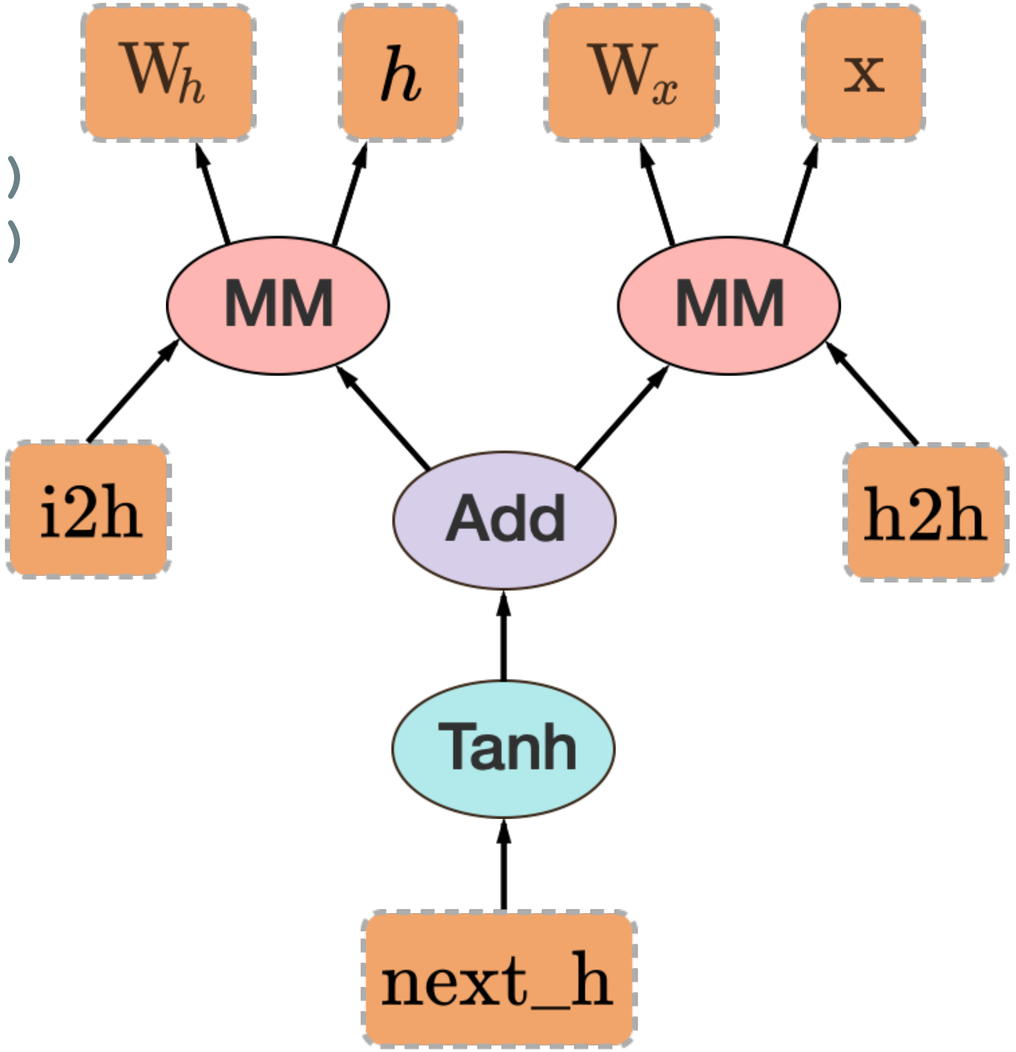
PyTorch Autograd

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


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

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


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.

```
net = Net()
optimizer = torch.optim.SGD(net.parameters(), lr=0.01, momentum=0.9)

for input, target in dataset:
    optimizer.zero_grad()
    output = model(input)
    loss = F.cross_entropy(output, target)
    loss.backward()
    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

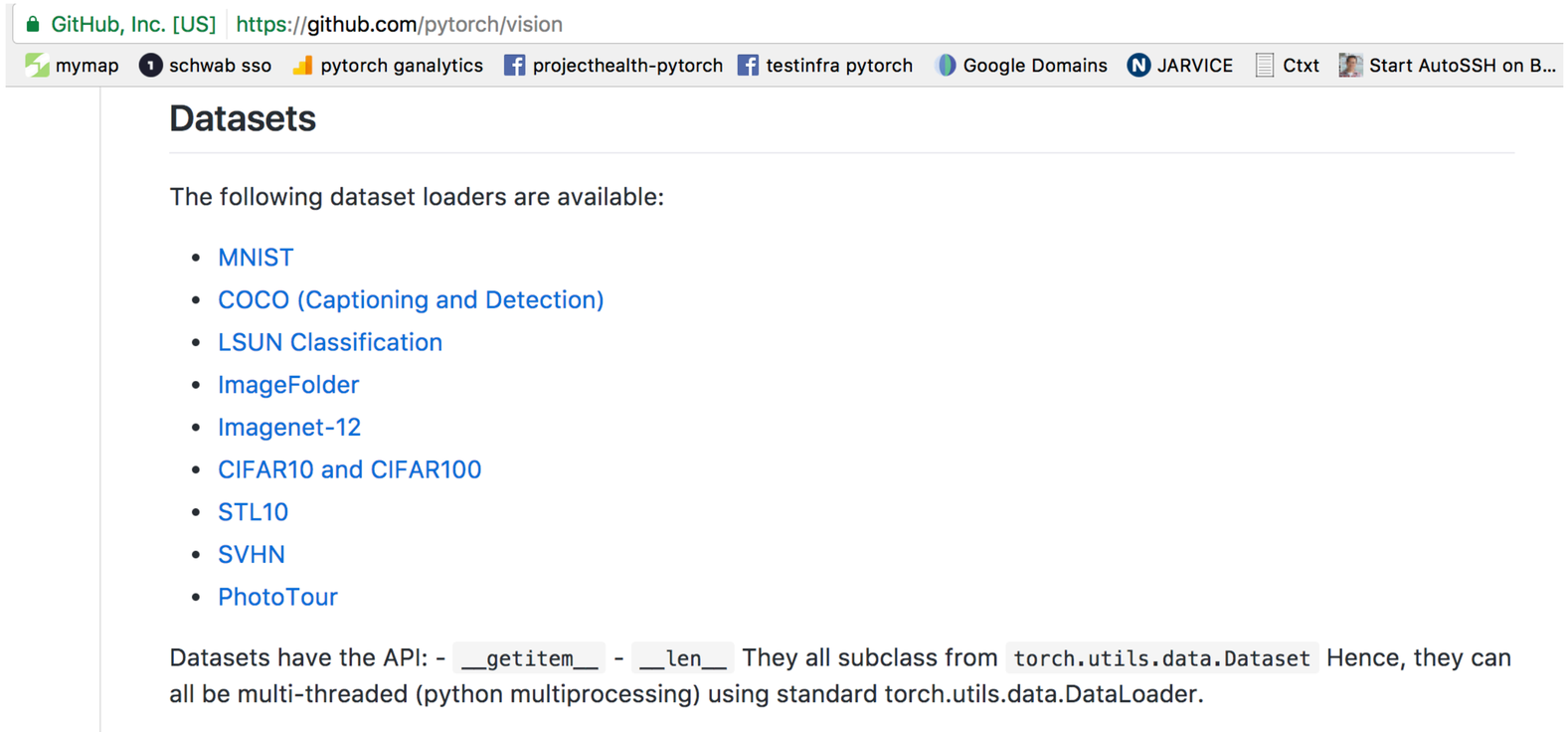
Writing Data Loaders

- 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:

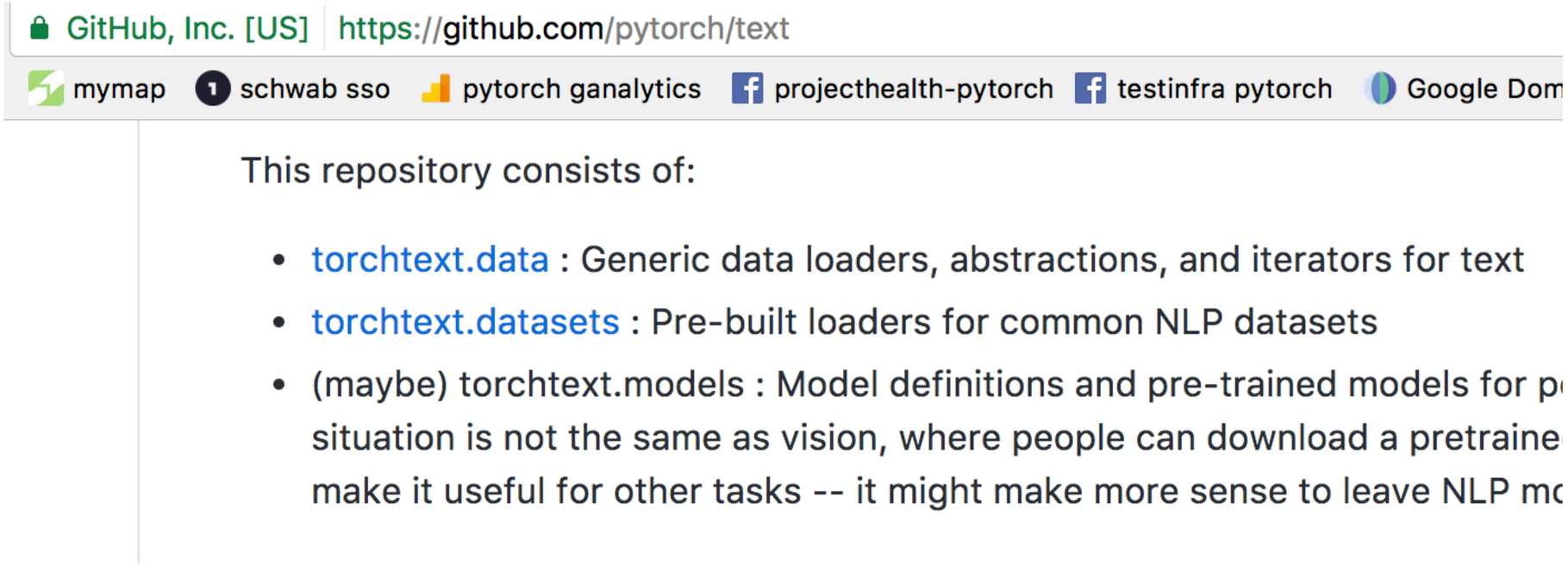
- share data loaders across the community!



Writing Data Loaders

PyTorch solution:

- share data loaders across the community!



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

Writin

PyTorch

- Code in p

```
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 Da

PyTorch soluti

- Code in practic

```
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)
```

Rese
Work

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
- 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

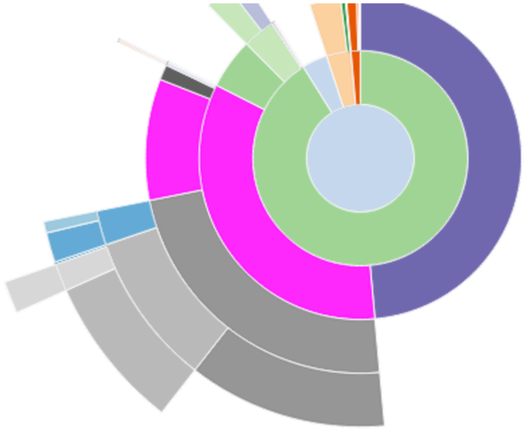
← PREVIOUS NEXT → G

SnakeViz
Installation
Starting SnakeViz
Generating Profiles
Interpreting Results
Controls
Notes
Contact

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/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      IntLoc = IntParIO + 10
   152     50000        63162      1.3     10.4      while 1:
   153     50000        69065      1.4     11.4          if Char1Glob == 'A':
   154     50000        66354      1.3     10.9              IntLoc = IntLoc - 1
   155     50000        67263      1.3     11.1              IntParIO = IntLoc - IntGlob
   156     50000        65494      1.3     10.8              EnumLoc = Ident1
   157     50000        68001      1.4     11.2          if EnumLoc == Ident1:
   158     50000        63739      1.3     10.5              break
   159     50000        61575      1.2     10.1      return IntParIO
```

Identifying bottlenecks

- torch.autograd.profiler

```
>>> x = torch.randn((1, 1), requires_grad=True)
>>> with torch.autograd.profiler.profile() as prof:
...     y = x ** 2
...     y.backward()
>>> # NOTE: some columns were removed for brevity
... print(prof)
-----------------------------------  --------------  --------------
Name                                        CPU time        CUDA time
-----------------------------------  --------------  --------------
PowConstant                               142.036us         0.000us
N5torch8autograd9GraphRootE                63.524us         0.000us
PowConstantBackward                       184.228us         0.000us
MulConstant                                50.288us         0.000us
PowConstant                                28.439us         0.000us
Mul                                        20.154us         0.000us
N5torch8autograd14AccumulateGradE          13.790us         0.000us
N5torch8autograd5CloneE                     4.088us         0.000us
```

Distributed PyTorch

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

torch.distributed

SYNC MODE

```
# Backward compatible synchronous collective op
torch.distributed.all_reduce(tensor, op, group, async_op=False)
```

ASYNC MODE

```
# New asynchronous collective op
work = torch.distributed.all_reduce(tensor, op, group, async_op=True)
work.wait()
```

TURN KEY SOLUTION

torch.nn.DistributedDataParallel

JUST WRAP YOUR MODEL

```
torch.distributed.init_process_group(world_size=4, init_method='...')
model = torch.nn.DistributedDataParallel(model)

for epoch in range(max_epochs):
    for data, target in enumerate(training_data):
      output = model(data)
      loss = F.nll_loss(output, target)
      loss.backward()
      optimizer.step()
```

PyTorch

Models are Python programs

- Simple
- Debuggable — `print` and `pdb`
- Hackable — use any Python library

- Needs Python to run
- Difficult to optimize and parallelize

PyTorch *Eager Mode*

Models are Python programs

- Simple
- Debuggable — `print` and `pdb`
- Hackable — use any Python library

- Needs Python to run
- Difficult to optimize and parallelize

PyTorch *Eager Mode*

Models are Python programs

- Simple
- Debuggable — `print` and `pdb`
- Hackable — use any Python library

- Needs Python to run
- Difficult to optimize and parallelize

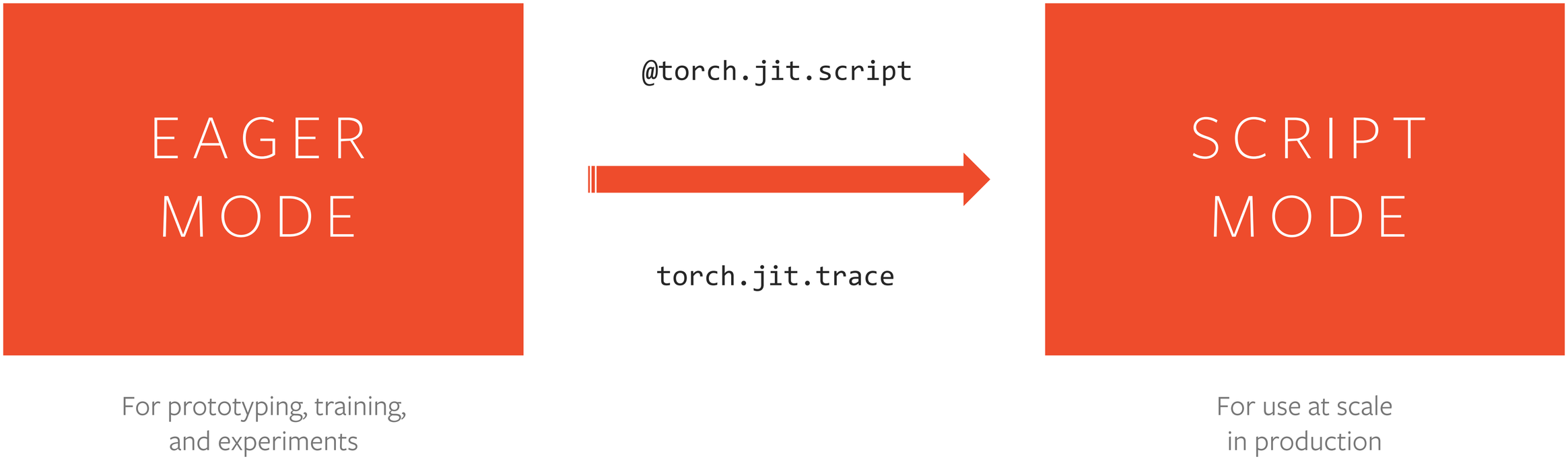
PyTorch *Script Mode*

Models are programs written in an optimizable subset of Python

- Production deployment
- No Python dependency
- Optimizable

PYTORCH JIT

Tools to transition eager code into script mode



Transitioning a model with `torch.jit.trace`

Take an existing eager model, and provide example inputs.

The tracer runs the function, recording the tensor operations performed.

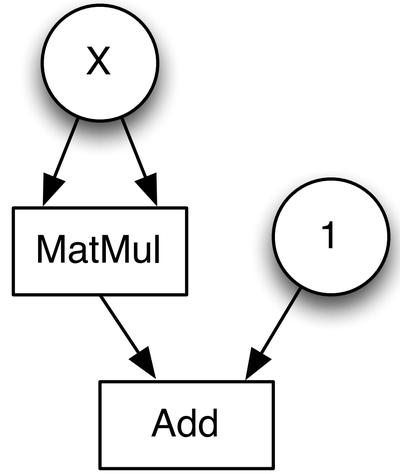
We turn the recording into a Torch Script module.

- Can reuse existing eager model code
- ⚠️ Control-flow is ignored

```
import torch
import torchvision


def foo(x, y):
  return 2*x + y


# trace a model by providing example inputs
traced_foo = torch.jit.trace(foo,
                             (torch.rand(3), torch.rand(3)))


traced_resnet = torch.jit.trace(torchvision.models.resnet18(),
                                torch.rand(1, 3, 224, 224))
```

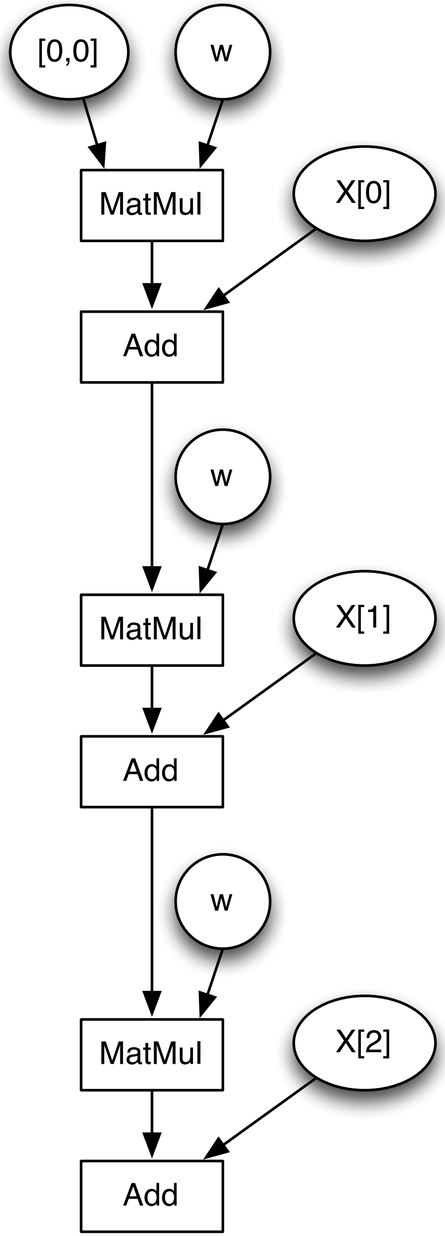
Tracing

```
def foo(x, t):
  y = x.mm(x)
  print(y) # still works!
  return y + t

x = torch.Tensor([[1,2],[3,4]])
foo(x, 1)

trace = torch.jit.trace(foo, (x, 1))
trace.save(“serialized.pt”)
```



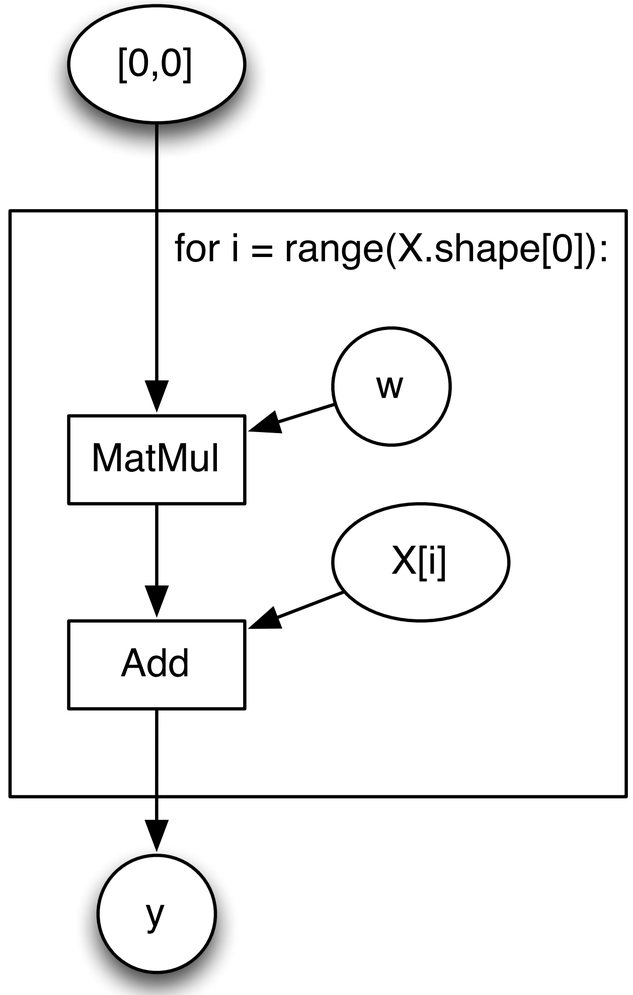
Tracing

```
def foo(x, t):
  y = x.mm(x)
  print(y) # still works!
  return y + t

def bar(x, w):
    y = torch.zeros(1, 2)
    for t in x:
        y = foo(y, w, t)
    return y

trace = torch.jit.trace(foo, (x, 1))
trace.save(“serialized.pt”)
```



Script

```
def foo(x, t):
  y = x.mm(x)
  print(y) # still works!
  return y + t

@script
def bar(x, w):
    y = torch.zeros(1, 2)
    for t in x:
        y = foo(y, w, t)
    return y

trace = torch.jit.trace(foo, (x, 1))
trace.save("serialized.pt")
```



Transitioning a model with `@torch.jit.script`

Write model directly in a subset of Python, annotated with `@torch.jit.script` or `@torch.jit.scri`

- Control-flow is pre
- `print` statement debugging
- Remove the annot standard Python tools.

```
class RNN(torch.jit.ScriptModule):
  def __init__(self, W_h, U_h, W_y, b_h, b_y):
    def forward(self, x, h):
      y = []
      for t in range(x.size(0)):
        h = torch.tanh(x[t] @ self.W_h + h @ self.U_h + self.b_h)
        y += [torch.tanh(h @ self.W_y + self.b_y)]
        if t % 10 == 0:
          print("stats: ", h.mean(), h.var())
      return torch.stack(y), h
```

You can mix both trace and script
in a single model.

Under the hood of `@torch.jit.script`

```
class MyModule(torch.jit.ScriptModule):
    def __init__(self, N, M):
        super(MyModule, self).__init__()
        self.weight = torch.nn.Parameter(torch.rand(N, M))

    @torch.jit.script_method
    def forward(self, input):
        if bool(input.sum() > 0):
          output = self.weight.mv(input)
        else:
          output = self.weight + input
        return output

ms = MyModule(3, 4)
```

```
ms.weight[1,2]
```

```
tensor(0.5476, grad_fn=<SelectBackward>)
```

```
ms(torch.rand(4))
```

```
tensor([1.2406, 1.4690, 1.8646], grad_fn=<MvBackward>)
```

```
ms.graph
```

```
graph(%input : Dynamic
      %5 : Dynamic) {
  %7 : int = prim::Constant[value=1]()
  %2 : int = prim::Constant[value=0]()
  %1 : Dynamic = aten::sum(%input)
  %3 : Dynamic = aten::gt(%1, %2)
  %4 : bool = prim::TensorToBool(%3)
  %output : Dynamic = prim::If(%4)
    block0() {
      %output.1 : Dynamic = aten::mv(%5, %input)
      -> (%output.1)
    }
    block1() {
      %output.2 : Dynamic = aten::add(%5, %input, %7)
      -> (%output.2)
    }
  return (%output);
}
```

```
print(ms.graph.pretty_print())
```

```
def graph(self,
    input: Tensor,
    _0: Tensor) -> Tensor:
  if bool(aten.gt(aten.sum(input), 0)):
    output = aten.mv(_0, input)
  else:
    output = aten.add(_0, input, alpha=1)
  return output
```

```
ms.save('mymodel.pt')
```

Predictable error messages `@torch.jit.script`

PARSE TIME

```
  @torch.jit.script_method
  def forward(self, input):
      if bool(input.sum() > 0):
        output = self.weight.mv(input_foo)
      else:
        output = self.weight + input
      return output
```

```
RuntimeError:
undefined value inpu:
@torch.jit.script_method
def forward(self, input):
    if bool(input.sum() > 0):
      output = self.weight.mv(inpu)
                              ~~~~ <--- HERE
    else:
      output = self.weight + input
    return output
```

RUNTIME

```
ms(torch.rand(111))
```

```
RuntimeError:
size mismatch, [3 x 4], [111] at caffe2/
operation failed in interpreter:
@torch.jit.script_method
def forward(self, input):
    if bool(input.sum() > 0):
      output = self.weight.mv(input)
               ~~~~~~~~~~~~~~~ <--- HERE
    else:
      output = self.weight + input
    return output
```

Loading a model without Python

Torch Script models can be saved to a model archive, and loaded in a python-free executable using a C++ API.

Our C++ Tensor API is the same as our Python API, so you can do preprocessing and post processing before calling the model.

```
# Python: save model
traced_resnet = torch.jit.trace(torchvision.models.resnet18(),
                                torch.rand(1, 3, 224, 224))
traced_resnet.save("serialized_resnet.pt")
```

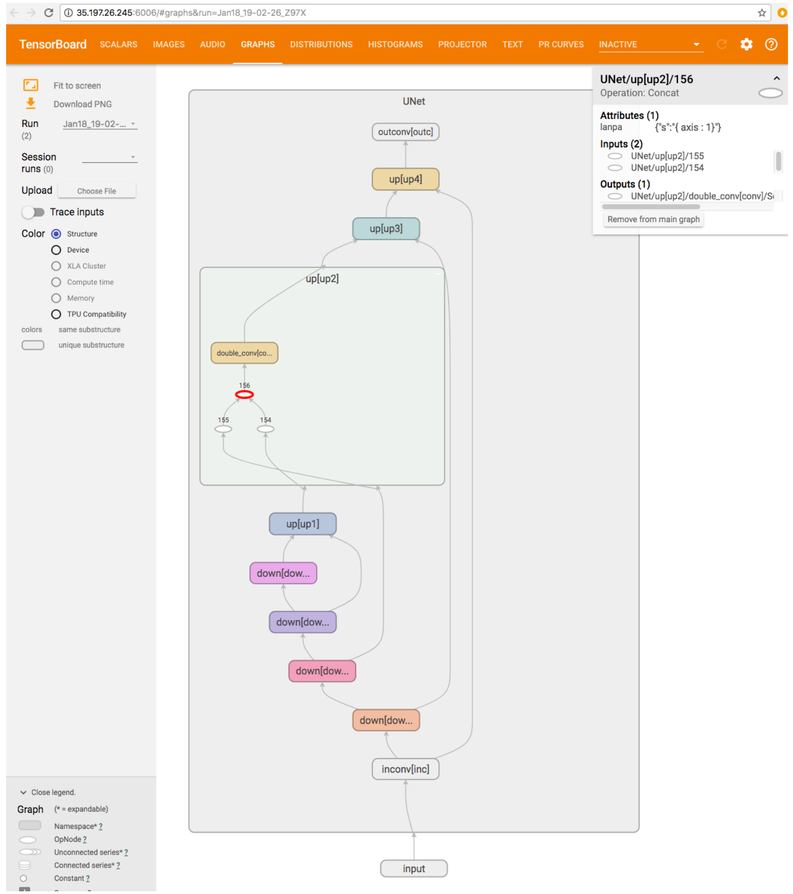
```
// C++: load and run model
auto module = torch::jit::load("serialized_resnet.pt");
auto example = torch::rand({1, 3, 224, 224});
auto output = module->forward({example}).toTensor();
std::cout << output.slice(1, 0, 5) << '\n';
```

Visualization

TensorBoard-PyTorch

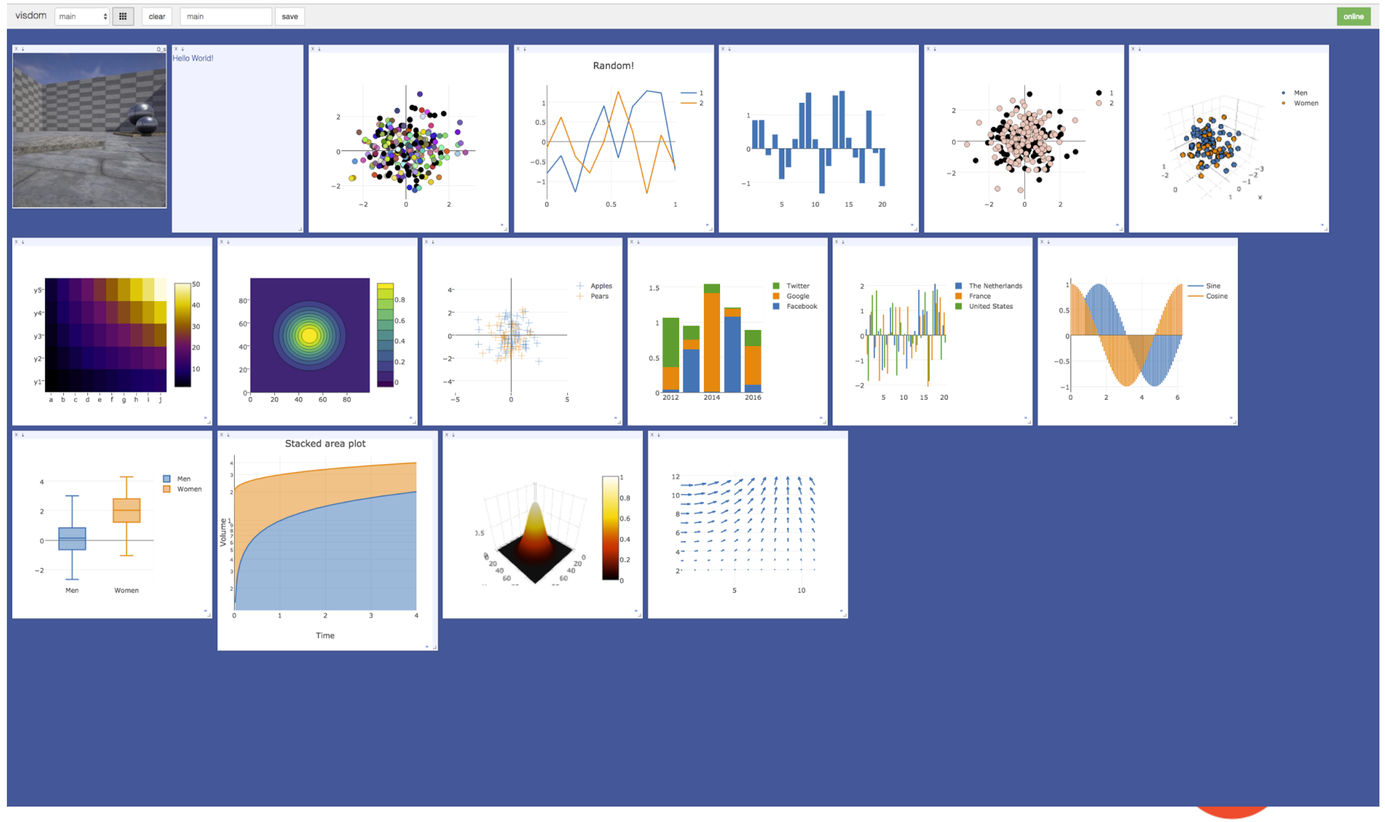
https://github.com/lanpa/tensorboard-pytorch

torch.utils.tensorboard (as of v1.1.0)



Visdom

https://github.com/facebookresearch/visdom



Ecosystem

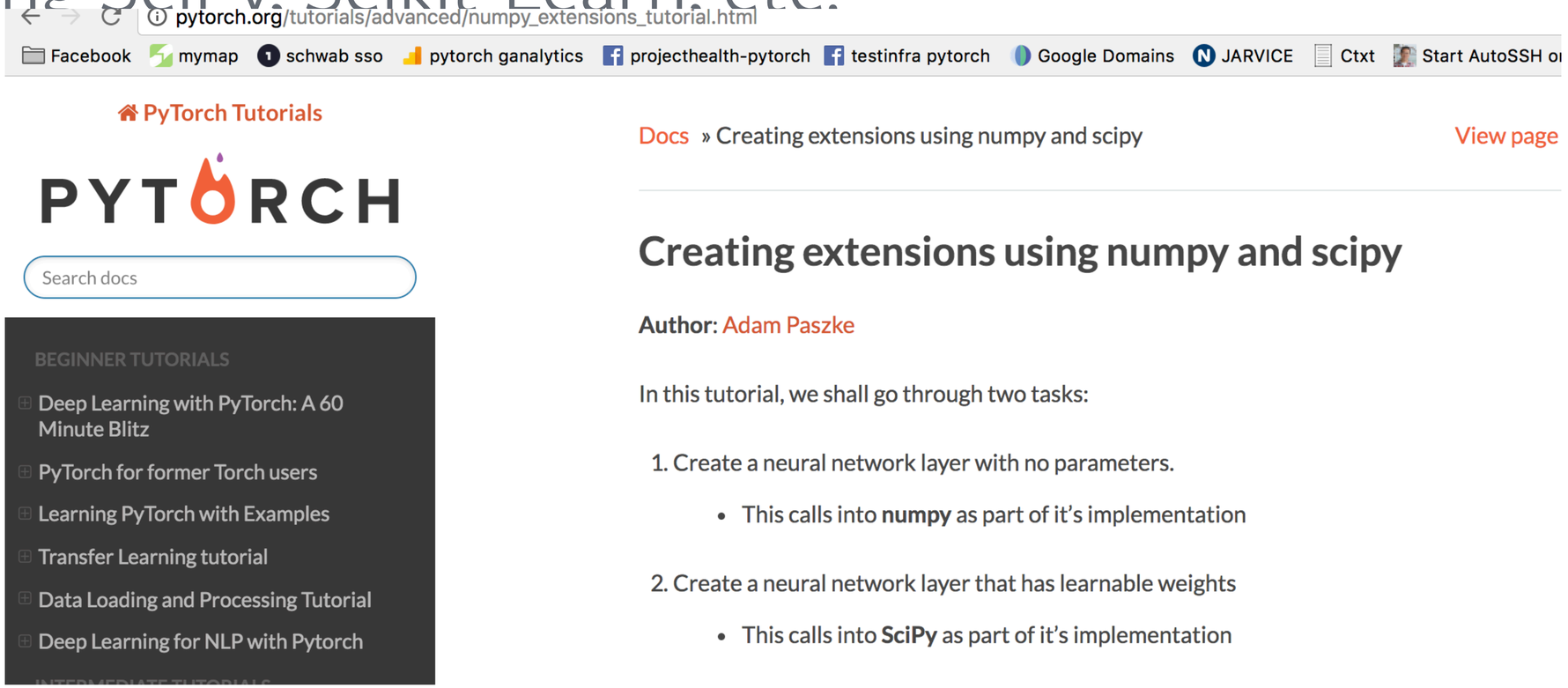
- Use the entire Python ecosystem at your will

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.



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 constructed by passing `pretrained=True`:

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

Ecosystem

https://github.com/skorch-dev/skorch

Scikit + PyTorch

Ecosyste

SKOF

korch

```python
import numpy as np
from sklearn.datasets import make_classification
from torch import nn
import torch.nn.functional as F

from skorch import NeuralNetClassifier


X, y = make_classification(1000, 20, n_informative=10, random_state=0)
X = X.astype(np.float32)
y = y.astype(np.int64)

class MyModule(nn.Module):
    def __init__(self, num_units=10, nonlin=F.relu):
        super(MyModule, self).__init__()

        self.dense0 = nn.Linear(20, num_units)
        self.nonlin = nonlin
        self.dropout = nn.Dropout(0.5)
        self.dense1 = nn.Linear(num_units, 10)
        self.output = nn.Linear(10, 2)

    def forward(self, X, **kwargs):
        X = self.nonlin(self.dense0(X))
        X = self.dropout(X)
        X = F.relu(self.dense1(X))
        X = F.softmax(self.output(X), dim=-1)
        return X


net = NeuralNetClassifier(
    MyModule,
    max_epochs=10,
    lr=0.1,
    # Shuffle training data on each epoch
    iterator_train__shuffle=True,
)

net.fit(X, y)
y_proba = net.predict_proba(X)
```

Ecosystem

SKORCH

https://github.com/skorch-dev/skorch

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler


pipe = Pipeline([
    ('scale', StandardScaler()),
    ('net', net),
])

pipe.fit(X, y)
y_proba = pipe.predict_proba(X)
```

Ecosystem

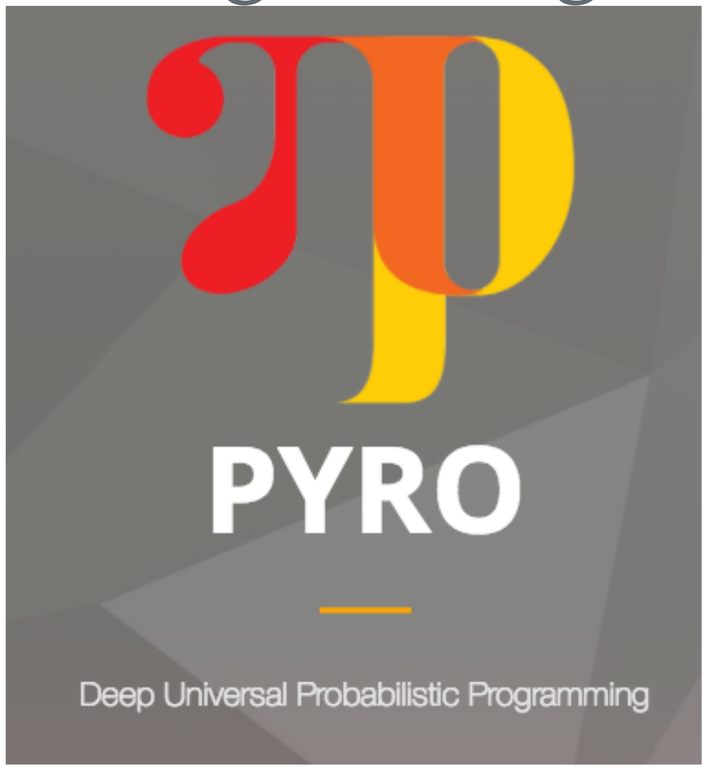
SKORCH

https://github.com/skorch-dev/skorch

```
from sklearn.model_selection import GridSearchCV


params = {
    'lr': [0.01, 0.02],
    'max_epochs': [10, 20],
    'module__num_units': [10, 20],
}
gs = GridSearchCV(net, params, refit=False, cv=3, scoring='accuracy')

gs.fit(X, y)
print(gs.best_score_, gs.best_params_)
```

Ecosystem

- Probabilistic Programming



http://pyro.ai/

github.com/probtorch/probtorch

Ecosystem

- Gaussian Processes

GPyTorch (Alpha Relase)

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

- CycleGAN, pix2pix

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

Ecosystem

Machine Translation

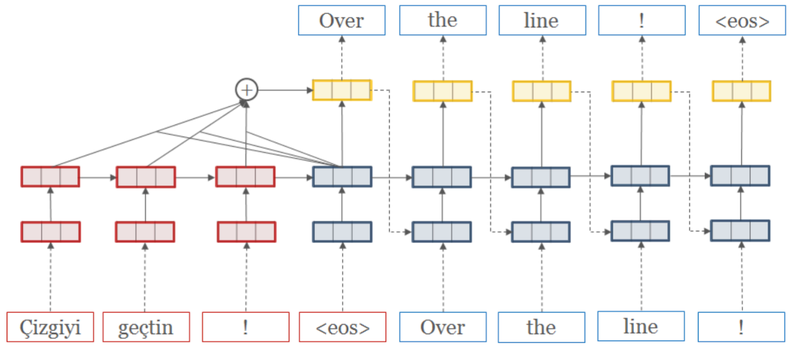
OpenNMT-py: Open-Source Neural Machine Translation

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.

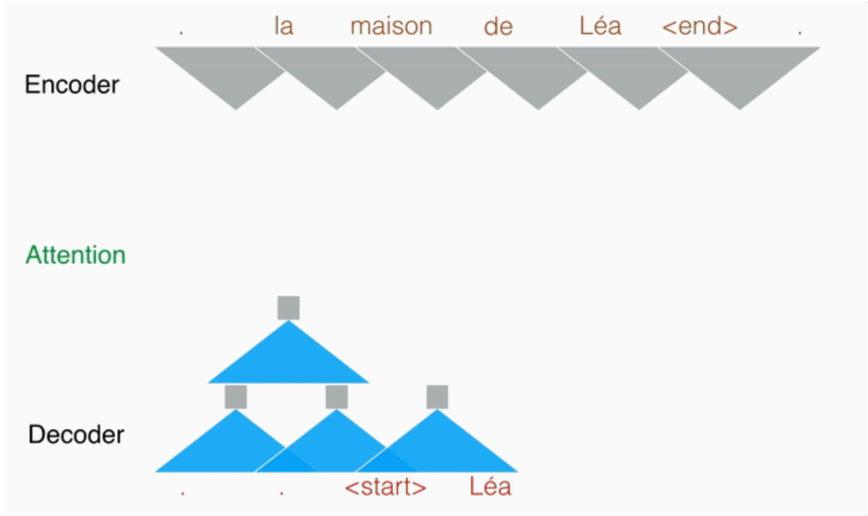
OpenNMT-py is run as a collaborative open-source project. It is maintained by Sasha Rush (Cambridge, MA), Ben Peters (Saarbrücken), and Jianyu Zhan (Shenzhen). The original code was written by Adam Lerer (NYC). We love contributions. Please consult the Issues page for any Contributions Welcome tagged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

•Machine Translation

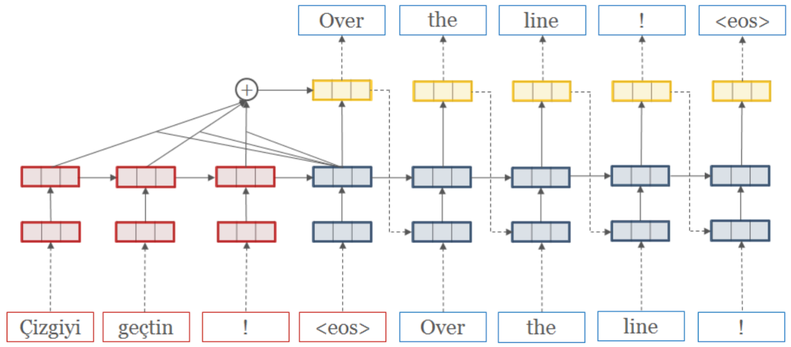
OpenNMT-py: Open-Source Neural Machine Translation

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.

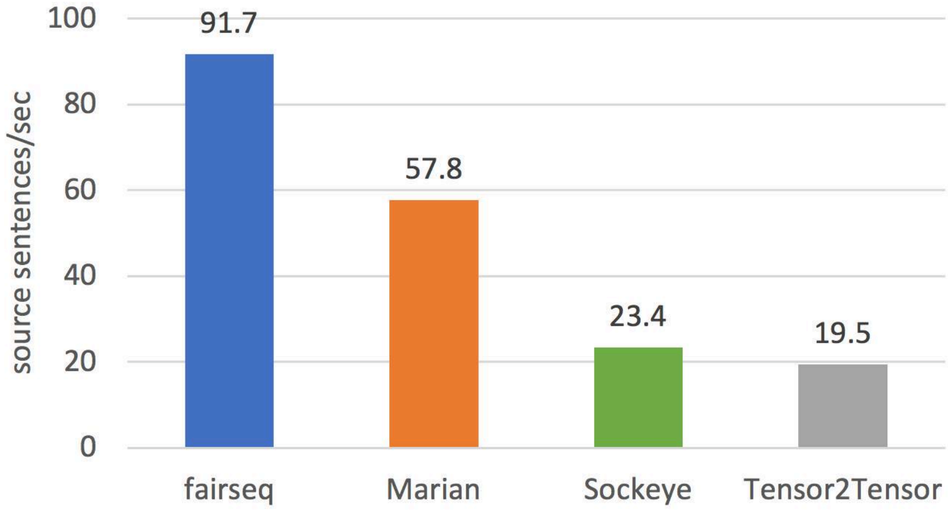
OpenNMT-py is run as a collaborative open-source project. It is maintained by Sasha Rush (Cambridge, MA), Ben Peters (Saarbrücken), and Jianyu Zhan (Shenzhen). The original code was written by Adam Lerer (NYC). We love contributions. Please consult the Issues page for any Contributions Welcome tagged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

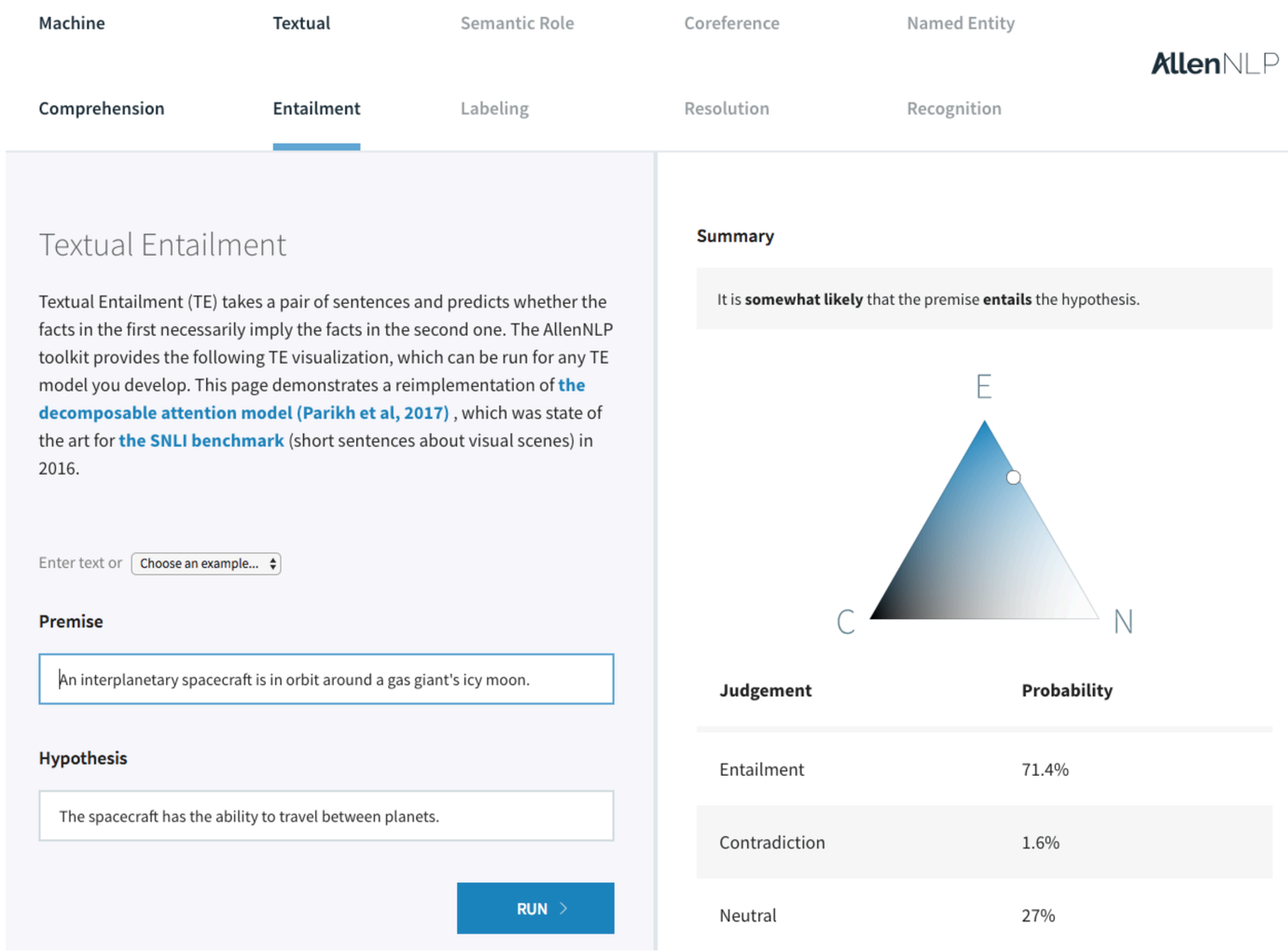


https://github.com/facebookresearch/fairseq-py

Ecosystem

- AllenNLP

http://allennlp.org/

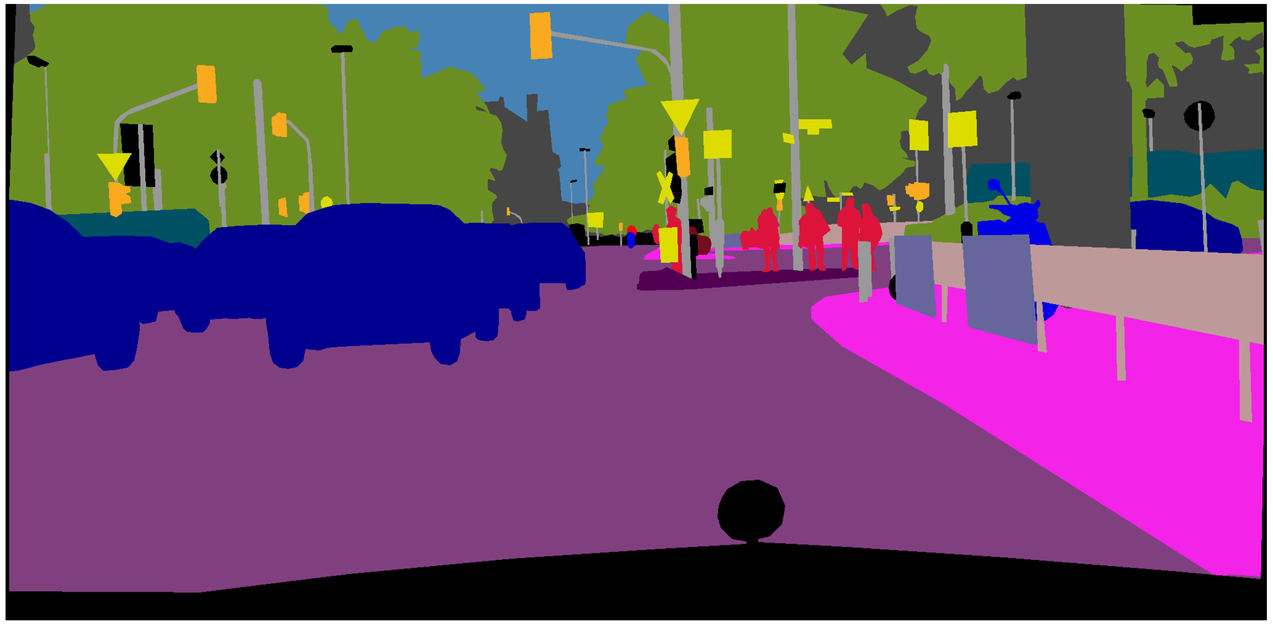


Ecosystem

- Pix2PixHD

https://github.com/NVIDIA/pix2pixHD

Input labels

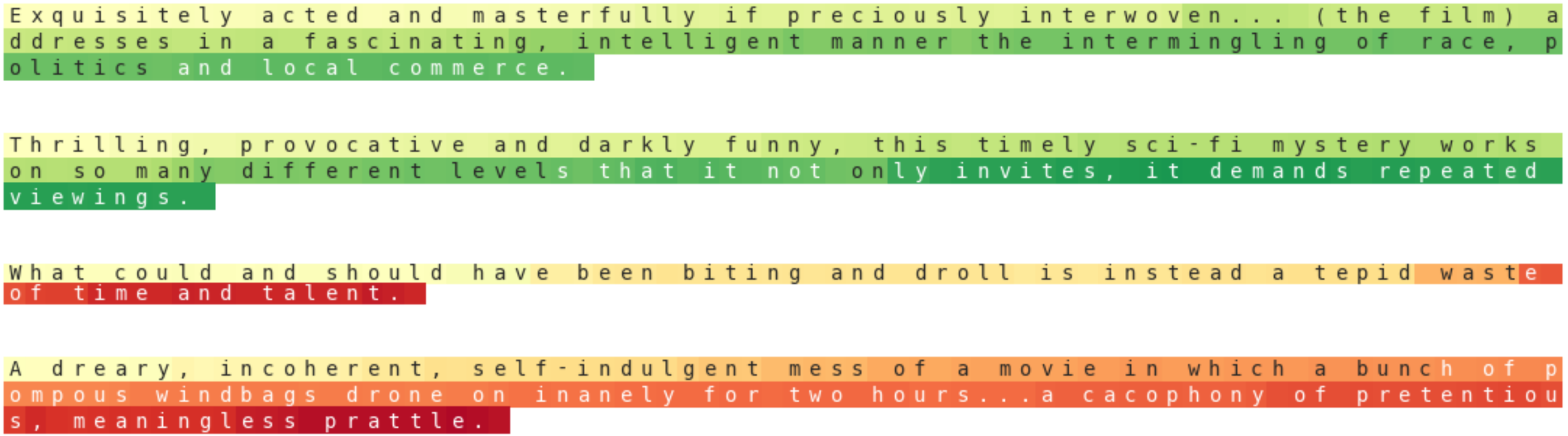


Synthesized image

Ecosystem

- Sentiment Discovery

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



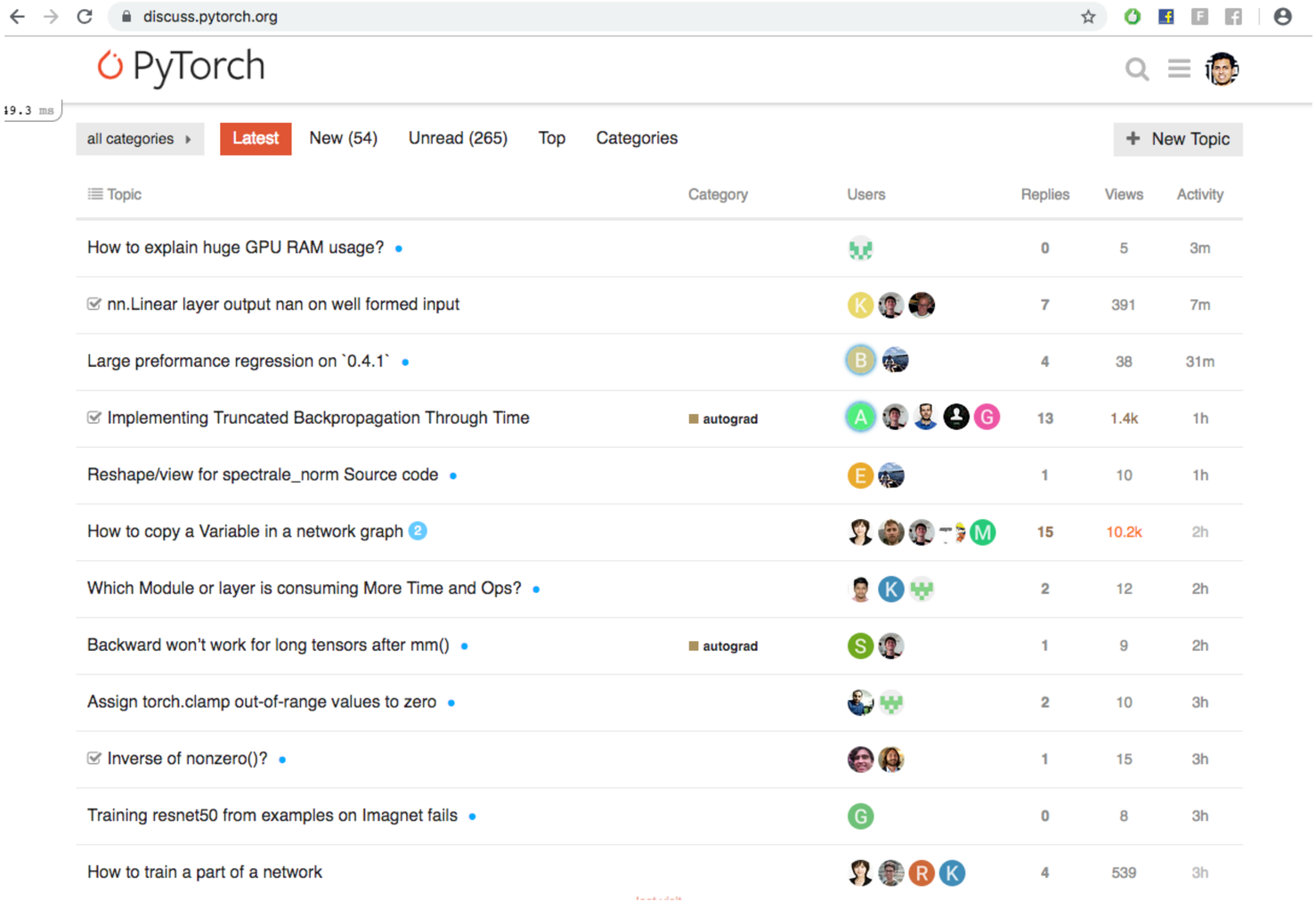
Ecosystem

- FlowNet2: Optical Flow Estimation with Deep Networks

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

Ecosystem

- A strong support system



https://pytorch.org

With from

facebook

UBER

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

Tutorial Time - https://bit.ly/pytorch-tutorials

- **Go to the google drive folder: link above**
- **Add it to My Drive**
- **Go go https://colab.research.google.com**
- **Connect app to drive**
- **Open basic_tutorials/**