PyTorch PyTorch PyTorch PyTorch PyTorch PyTorch
PyTorch PyTorch PyTorch PyTorch PyTorch PyTorch
PyTorch PyTorch PyTorch PyTorch PyTorch PyTorch
PyTorch PyTorch PyTorch PyTorch PyTorch PyTorch
PyTorch PyTorch PyTorch PyTorch PyTorch PyTorch
PyTorch PyTorch PyTorch PyTorch PyTorch PyTorch
PyTorch PyTorch PyTorch PyTorch PyTorch PyTorch
PyTorch PyTorch PyTorch PyTorch PyTorch PyTorch
PyTorch PyTorch PyTorch PyTorch PyTorch PyTorch
PyTorch PyTorch PyTorch PyTorch PyTorch PyTorch
PyTorch PyTorch PyTorch PyTorch PyTorch PyTorch
PyTorch PyTorch PyTorch PyTorch PyTorch PyTorch
PyTorch PyTorch PyTorch PyTorch PyTorch PyTorch

PyTorch

an ecosystem for deep learning

Soumith Chintala
Facebook AI

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

```python
# -*- 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

```python
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
```

Neural Networks

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x)

model = Net()
input = Variable(torch.randn(10, 20))
output = model(input)
```

Neural Networks

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x)

model = Net()
input = Variable(torch.randn(10, 20))
output = model(input)
```

Neural Networks

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x)

model = Net()
input = Variable(torch.randn(10, 20))
output = model(input)
```

Optimization package

SGD, Adagrad, RMSProp, LBFGS, etc.

```python
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()
```

Distributed PyTorch

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

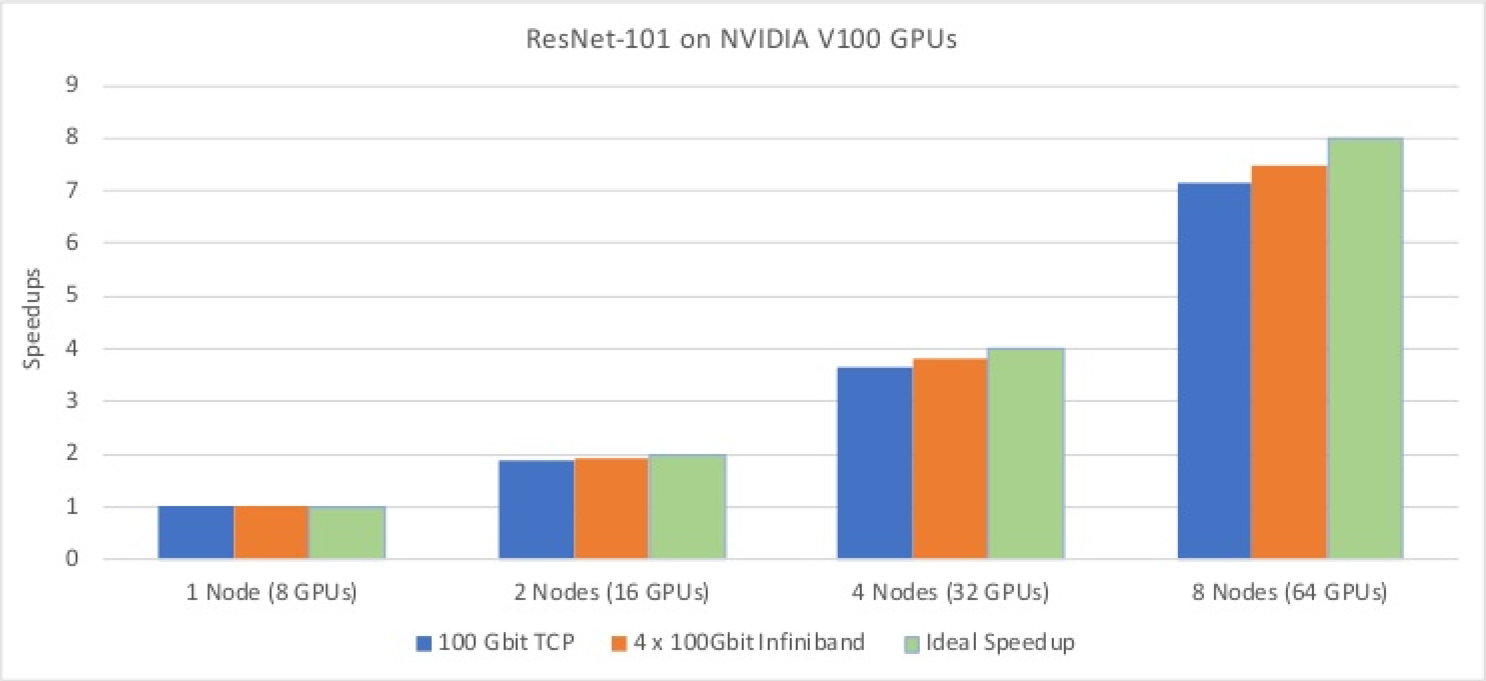
Distributed Data Parallel

```
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()
```

Distributed Data Parallel

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

Distributed Training Performance – ResNet101



Use via DataBricks MLFlow

- mlflow.pytorch
  - saves and loads models
- More resources:
  - https://docs.databricks.com/spark/latest/mllib/mlflow-pytorch.html
  - https://www.mlflow.org/docs/latest/models.html

Ecosystem

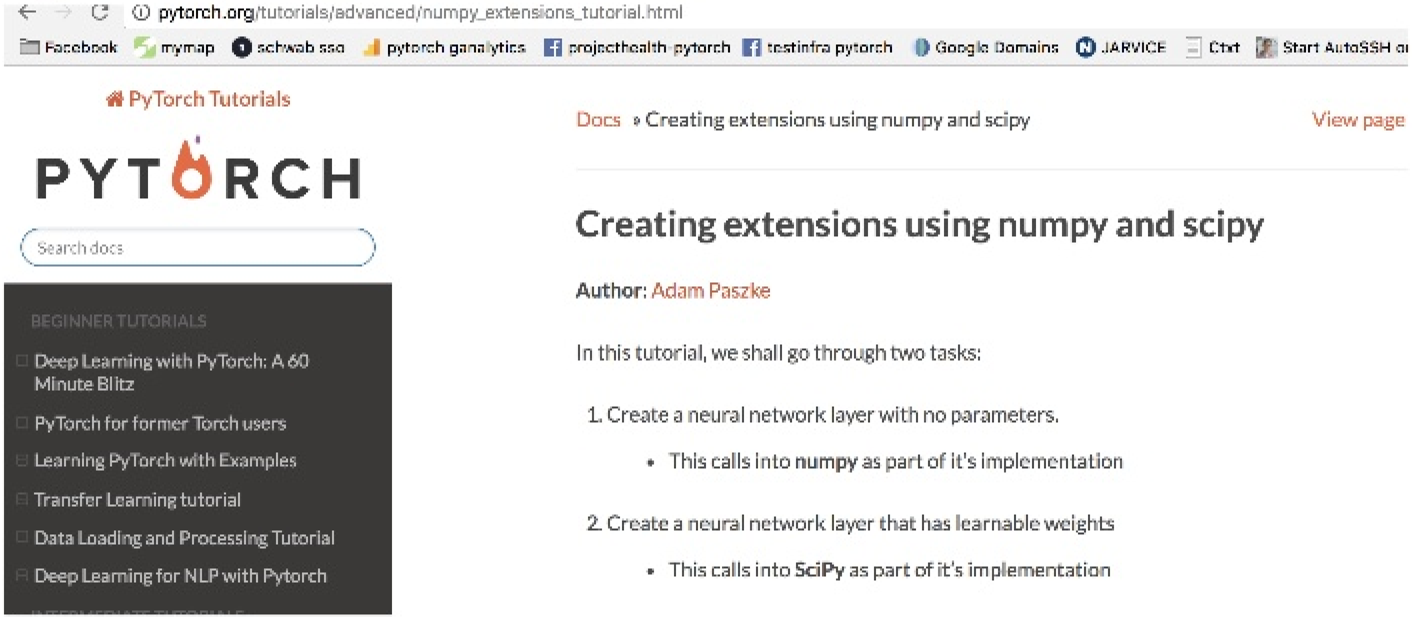
- Use the entire Python ecosystem at your will

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

- Probabilistic Programming

PYRO

Deep Universal Probabilistic Programming

PROB TORCH

github.com/probtorch/probtorch

http://pyro.ai/

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

- Machine Translation

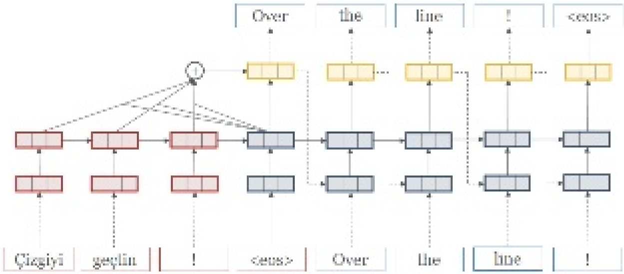
OpenNMT-py: Open-Source Neural Machine Translation

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.

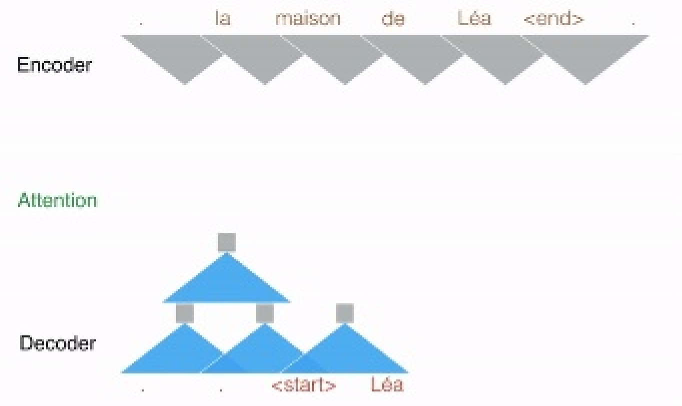
OpenNMT-py is run as a collaborative open-source project. It is maintained by Sasha Rush (Cambridge, MA), Ben Peters (Saarbrücken), and Jianyu Zhan (Shenzhen). The original code was written by Adam Lerer (NYC). We love contributions. Please consult the Issues page for any Contributions Welcome tagged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



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

FAIR Sequence-to-Sequence Toolkit (PyTorch)

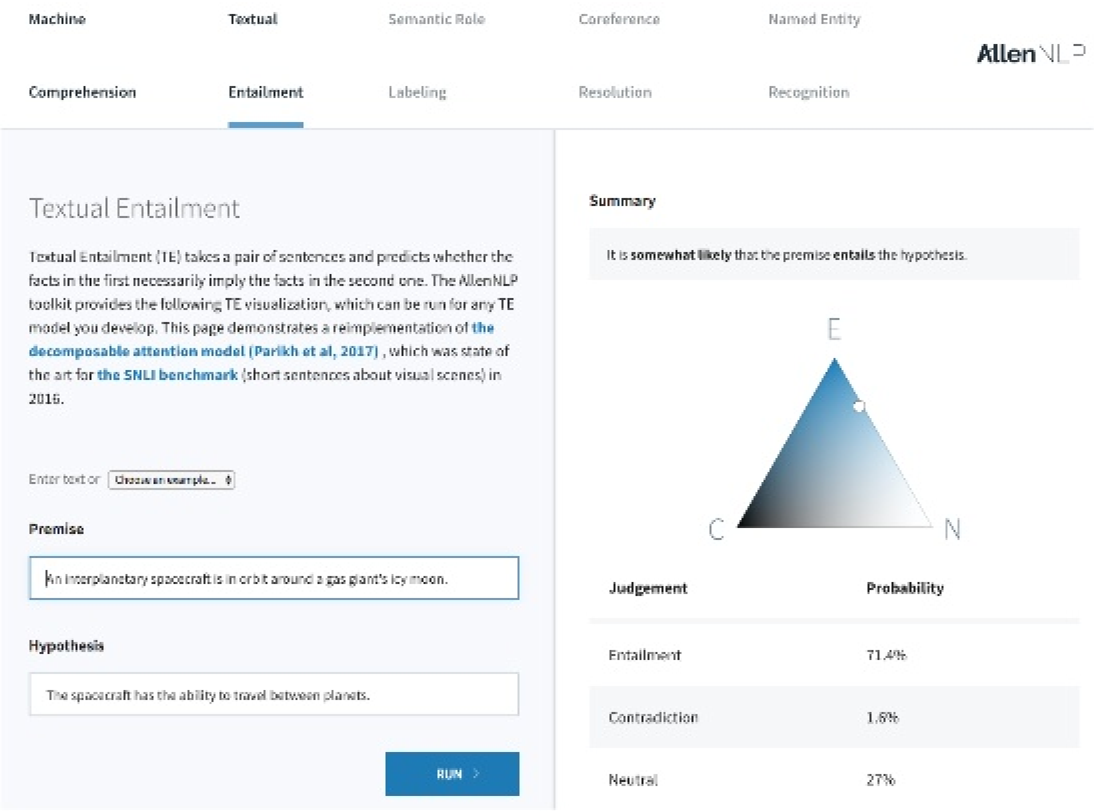
This is a PyTorch version of fairseq, a sequence-to-sequence learning toolkit from Facebook AI Research. The original authors of this reimplementation are (in no particular order) Sergey Edunov, Myle Ott, and Sam Gross. The toolkit implements the fully convolutional model described in Convolutional Sequence 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/

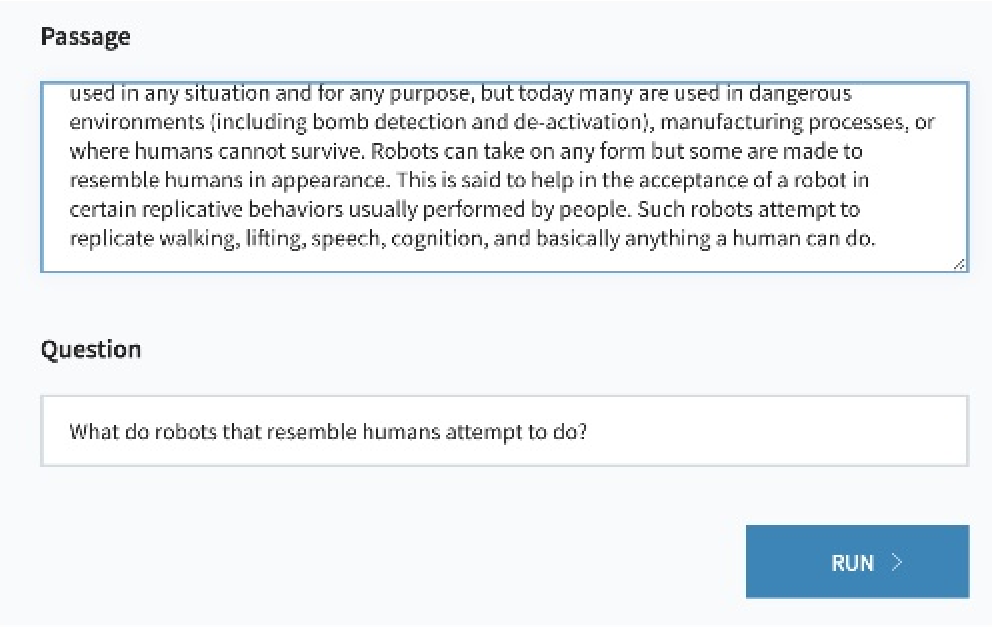


Ecosystem

- AllenNLP http://allennlp.org/
- State-of-the-art models for comprehension, Q&A, various other NLP tasks

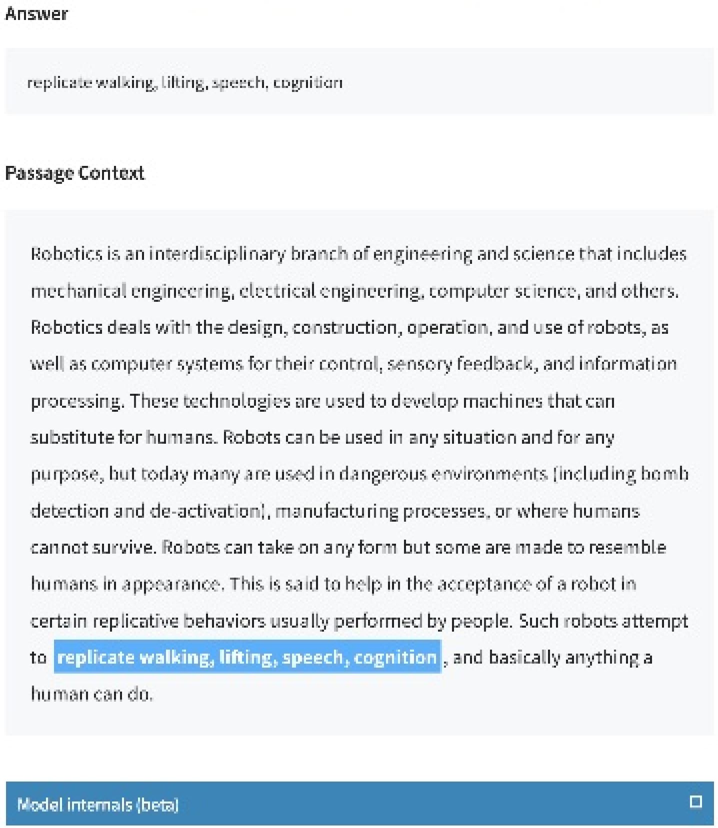
Ecosystem

- AllenNLP http://allennlp.org/
- State-of-the-art models for comprehension, Q&A, various other NLP tasks



Ecosystem

- AllenNLP http://allennlp.org/
- State-of-the-art m , Q&A, various other NLP



fast.ai 1.0

- High-level library on PyTorch: http://docs.fast.ai

fast.ai 1.0

- High-level library on PyTorch: http://docs.fast.ai
- Built by Jeremy Howard, Rachel Thomas and many community members

fast.ai 1.0

- High-level library on PyTorch: http://docs.fast.ai
- Built by Jeremy Howard, Rachel Thomas and many community members
- an online course accompanies the library

fast.ai 1.0

- High-level library on PyTorch: http://docs.fast.ai
- Built by Jeremy Howard, Rachel Thomas and many community members
- an online course accompanies the library
- Read more at http://www.fast.ai/2018/10/02/fastai-ai/

fast.ai 1.0

- state-of-the-art models in few lines

fast.ai 1.0

- state-of-the-art models in few lines
- fine-tune on your own data

fast.ai 1.0

- state-of-the-art models in few lines
- fine-tune on your own data

Near State-of-the-art Image Classifiers

```
data = data_from_imagefolder(Path('data/dogscats'),
    ds_tfms=get_transforms(), tfms=imagenet_norm, size=224)
learn = ConvLearner(data, tvm.resnet34, metrics=accuracy)
learn.fit_one_cycle(6)
learn.unfreeze()
learn.fit_one_cycle(4, slice(1e-5,3e-4))
```

fast.ai 1.0

- state-of-the-art models in few lines
- fine-tune on your own data

Models and Transforms for Tabular Data

class TabularModel

```
TabularModel ( emb_szs : ListSizes ,  n_cont : int ,  out_sz : int ,
layers : Collection [ int ],  ps : Collection [ float ]=None ,  emb_drop : float = 0.0 ,
y_range : OptRange = None ,  use_bn : bool = True ) ::  Module  [source]
```

PyTorch

https://pytorch.org

With ❤ from