





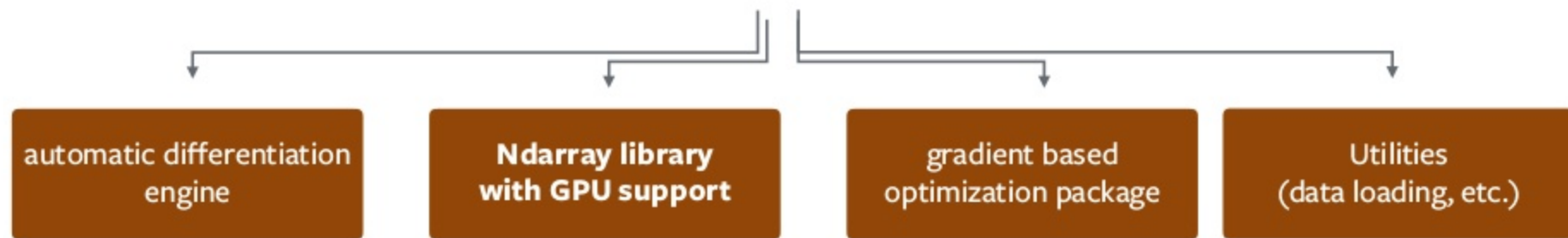
an ecosystem for deep learning

**Soumith Chintala**

Facebook AI



# What is PyTorch?



Deep Learning

Numpy-alternative

Reinforcement Learning



# ndarray library

- `np.ndarray`  $\leftrightarrow$  `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
```

## 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
```

# 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:

```
1.000000e-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

```
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(1, 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(1, 1, 1, 1))
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(1, 20))
21 output = model(input)
```

# Optimization package

SGD, Adagrad, RMSProp, LBFGS, etc.

```
1 net = Net()
2 optimizer = torch.optim.SGD(net.parameters(), lr=0.01, momentum=0.9)
3
4 for input, target in dataset:
5     optimizer.zero_grad()
6     output = model(input)
7     loss = F.cross_entropy(output, target)
8     loss.backward()
9     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

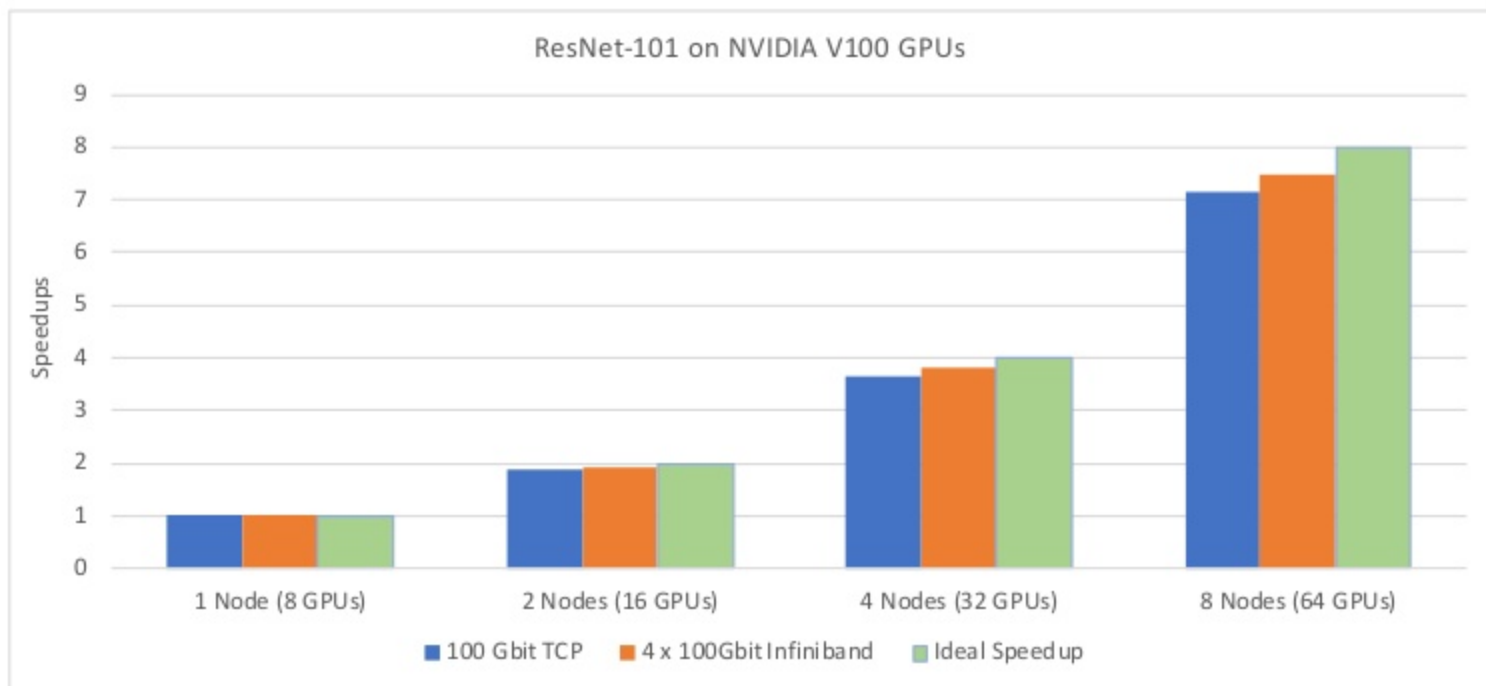
```
for epoch in range(max_epochs):  
    for data, target in enumerate(training_data):  
        output = model(data)  
        model = nn.DistributedDataParallel(model)  
        loss = F.nll_loss(output, target)  
        loss.backward()  
        optimizer.step()
```





PYTORCH 1.0

## 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





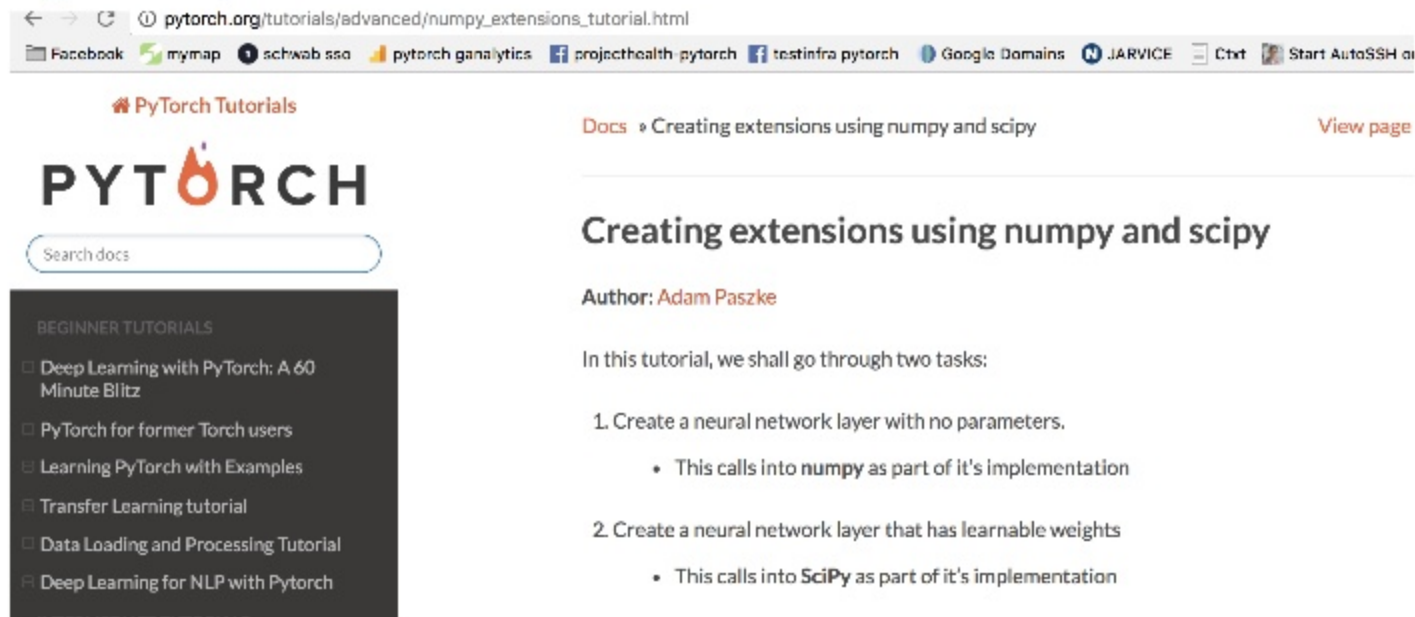
# Ecosystem

- 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.



The screenshot shows a web browser window with the URL `pytorch.org/tutorials/advanced/numpy_extensions_tutorial.html`. The browser's address bar and tabs are visible at the top. The page content includes the PyTorch logo, a search bar, and a sidebar with a list of tutorials. The main content area displays the title 'Creating extensions using numpy and scipy' and the author 'Adam Paszke'. Below this, the text states 'In this tutorial, we shall go through two tasks:' followed by a numbered list of two tasks, each with a bullet point describing the library used for implementation.

PyTorch Tutorials

PYTORCH

Search docs

BEGINNER TUTORIALS

- Deep Learning with PyTorch: A 60 Minute Blitz
- PyTorch for former Torch users
- Learning PyTorch with Examples
- Transfer Learning tutorial
- Data Loading and Processing Tutorial
- Deep Learning for NLP with Pytorch

Docs » Creating extensions using numpy and scipy [View page](#)

## Creating extensions using numpy and scipy

Author: [Adam Paszke](#)

In this tutorial, we shall go through two tasks:

1. Create a neural network layer with no parameters.
  - This calls into **numpy** as part of it's implementation
2. Create a neural network layer that has learnable weights
  - This calls into **SciPy** as part of it's implementation



# Ecosystem

- A shared model-zoo:

We provide pre-trained models for the ResNet variants and AlexNet, using the PyTorch

`torch.utils.model_zoo`. These can be constructed by passing `pretrained=True`:

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



# Ecosystem

- Probabilistic Programming



<http://pyro.ai/>



**PROB  
TORCH**

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



# 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>



# Ecosystem

## •Machine Translation

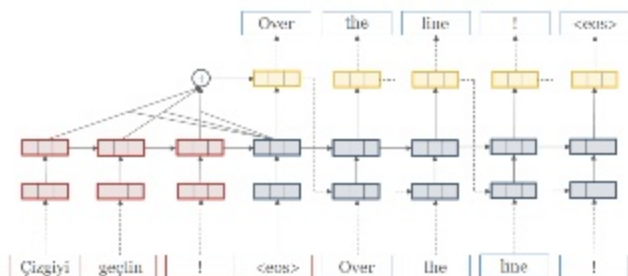
### OpenNMT-py: Open-Source Neural Machine Translation

build `newling`

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.

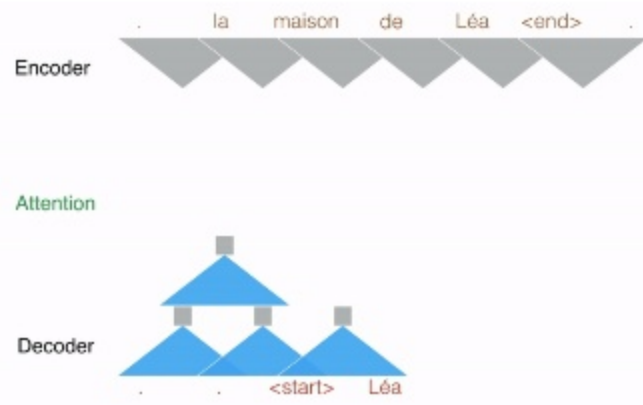
OpenNMT-py is run as a collaborative open-source project. It is maintained by [Sasha Rush](#) (Cambridge, MA), [Ben Peters](#) (Searbrücken), and [Jianyu Zhou](#) (Shenzhen). The original code was written by [Adam Lerer](#) (NYC). We love contributions. Please consult the [Issues](#) page for any [Contributions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

Machine

Textual

Semantic Role

Coreference

Named Entity

AllenNLP

Comprehension

Entailment

Labeling

Resolution

Recognition

## Textual Entailment

Textual Entailment (TE) takes a pair of sentences and predicts whether the facts in the first necessarily imply the facts in the second one. The AllenNLP toolkit provides the following TE visualization, which can be run for any TE model you develop. This page demonstrates a reimplementation of the [decomposable attention model \(Parikh et al, 2017\)](#), which was state of the art for the [SNLI benchmark](#) (short sentences about visual scenes) in 2016.

Enter text or [Use an example...](#)

Premise

An interplanetary spacecraft is in orbit around a gas giant's icy moon.

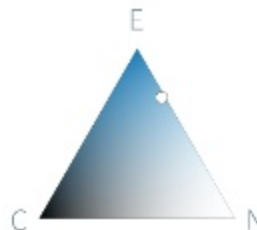
Hypothesis

The spacecraft has the ability to travel between planets.

RUN >

## Summary

It is **somewhat likely** that the premise **entails** the hypothesis.



Judgement

Probability

Entailment

71.4%

Contradiction

1.8%

Neutral

27%



# 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

## Passage

used in any situation and for any purpose, but today many are used in dangerous environments (including bomb detection and de-activation), manufacturing processes, or where humans cannot survive. Robots can take on any form but some are made to resemble humans in appearance. This is said to help in the acceptance of a robot in certain replicative behaviors usually performed by people. Such robots attempt to replicate walking, lifting, speech, cognition, and basically anything a human can do.

## Question

What do robots that resemble humans attempt to do?

RUN >



# Ecosystem

- AllenNLP
- State-of-the-art n
- various other NLP

<http://allennlp.org/>

Answer

replicate walking, lifting, speech, cognition

, Q&A,

Passage Context

Robotics is an interdisciplinary branch of engineering and science that includes mechanical engineering, electrical engineering, computer science, and others. Robotics deals with the design, construction, operation, and use of robots, as well as computer systems for their control, sensory feedback, and information processing. These technologies are used to develop machines that can substitute for humans. Robots can be used in any situation and for any purpose, but today many are used in dangerous environments [including bomb detection and de-activation], manufacturing processes, or where humans cannot survive. Robots can take on any form but some are made to resemble humans in appearance. This is said to help in the acceptance of a robot in certain replicative behaviors usually performed by people. Such robots attempt to replicate walking, lifting, speech, cognition , and basically anything a human can do.



# 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, tvn.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 \[source\]
```





<https://pytorch.org>

With ❤️ from

facebook



ParisTech  
INSTITUT DES SCIENCES ET TECHNOLOGIE  
PARIS INSTITUTE OF TECHNOLOGY

Carnegie  
Mellon  
University



Digital  
Reasoning

Stanford  
University



Inria



Berkeley  
UNIVERSITY OF CALIFORNIA