Tutorial 1

Pytorch Introduction

About the exercises

1) It is not required to submit by email

   However, if you want feedback, feel free to email

2) Next week presenter (?)

Last week's exercise

Make a program that counts the frequency of words in a file

create a dictionary *counts* create a map to hold counts

open a file

for each *line* **in** the file
 split *line* **into** *words*

 for *w* **in** *words*
  if *w* exists in *counts*, **add** 1 to *counts*[*w*]
  else set *counts*[*w*] = 1

print key, value of *counts*

```
counts = {}

filename = "data/00-input.txt"

f = open(filename)

for line in f:
    words = line.split()

    for w in words:
        if w in counts:
            counts[w] += 1
        else:
            counts[w] = 1

for k, v in sorted(counts.items()):
    print(k,v)
```

Question

What is the vocabulary size considering the following sentence?

The book is on the table

Question

What is the vocabulary size considering the following sentence?

The book is on the table → The, the ⇒ the → Vocabulary: {the, book, is, on, table}

When processing texts, we usually need to do some **normalization**.

Last week’s exercise

Make a program that counts the frequency of words in a file

create a dictionary *counts* create a map to hold counts

open a file

for each *line* **in** the file
  split *line* **into** *words*

  for *w* **in** *words*
    if *w* exists in *counts*, **add** 1 to *counts*[*w*]
    else set *counts*[*w*] = 1

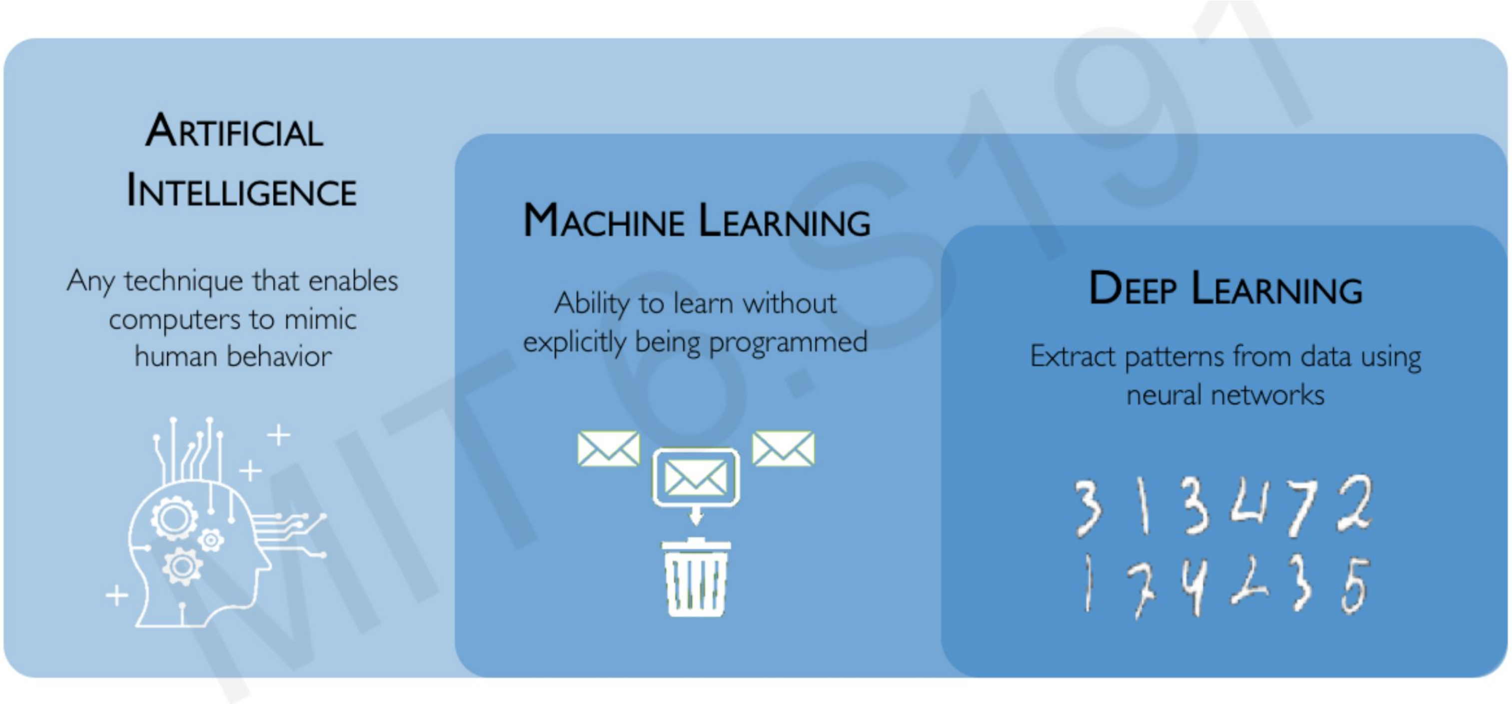
print key, value of *counts*

```python
counts = {}

filename = "data/00-input.txt"

f = open(filename)

for line in f:
    words = line.lower().split()

    for w in words:
        if w in counts:
            counts[w] += 1
        else:
            counts[w] = 1

for k, v in sorted(counts.items()):
    print(k,v)
```

lowercase

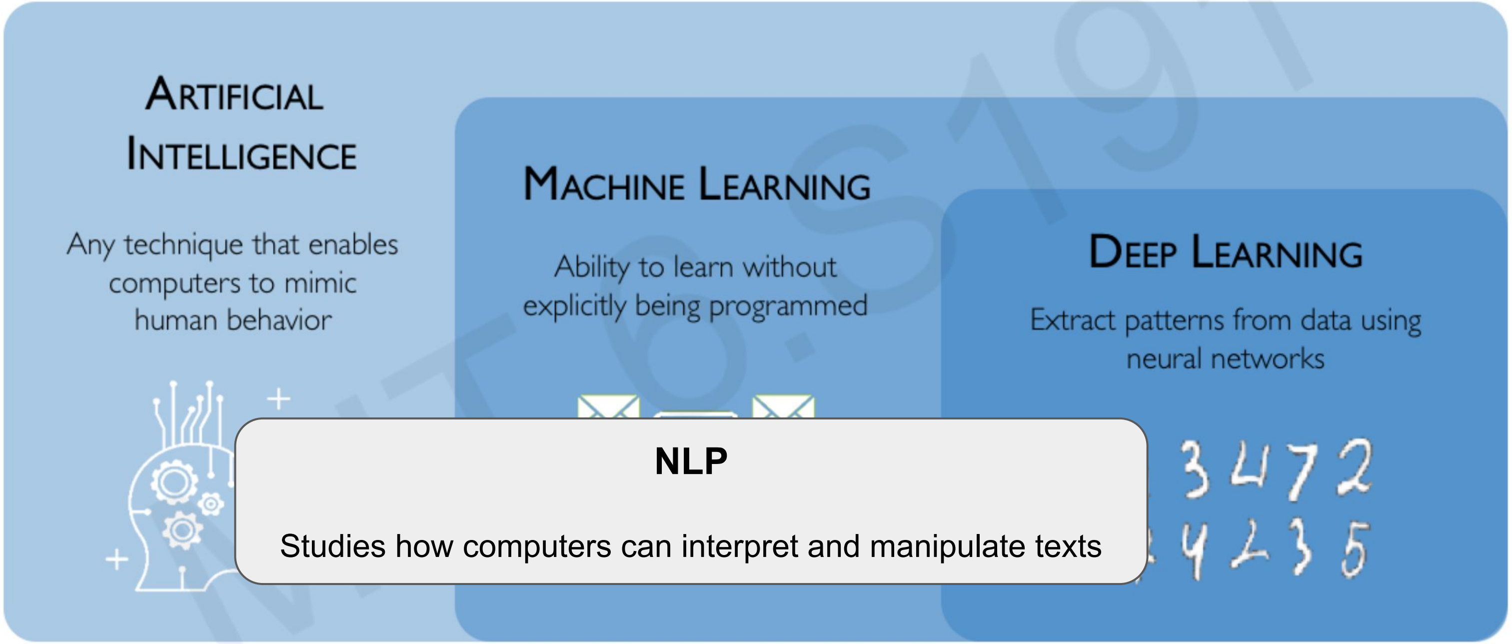
Pytorch Introduction

Introduction



From: http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Introduction



NLP

Studies how computers can interpret and manipulate texts

Supervised Learning

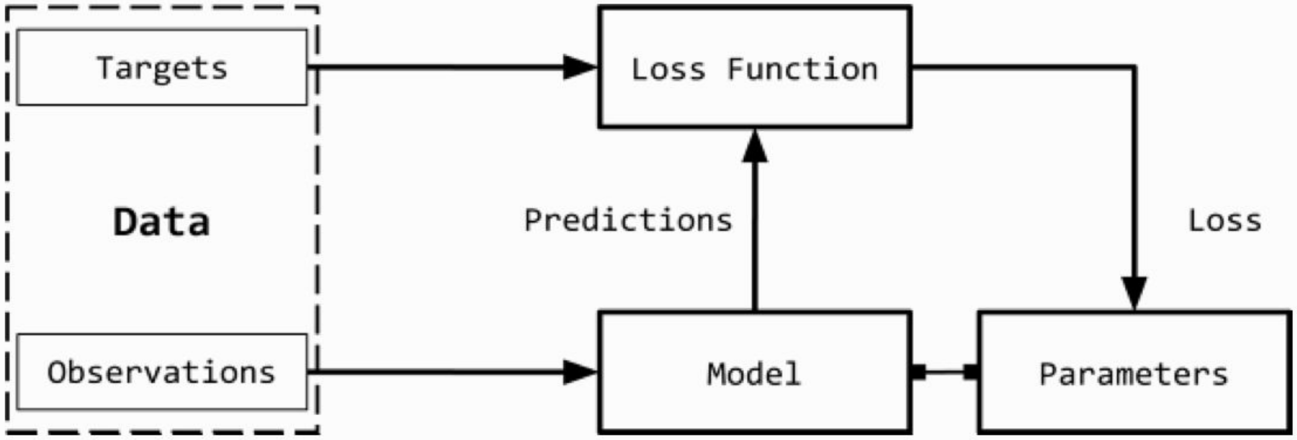


Figure 1-1. The supervised learning paradigm, a conceptual framework for learning from labeled input data.

Supervised Learning

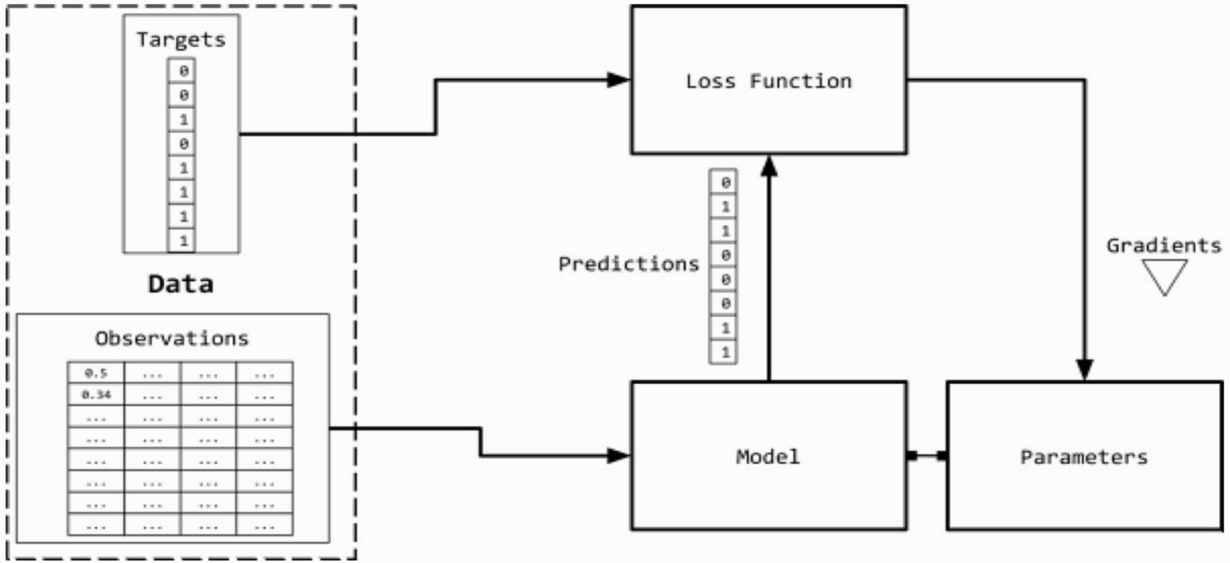


Figure 1-2. Observation and target encoding: The targets and observations from Figure 1-1 are represented numerically as vectors, or tensors. This is collectively known as input "encoding."

One-Hot Representation

Means one **1**, and rest **0**s

Words can be represented as **one-hot** vectors

fruit = [0,0,0,0,0,1,0,0]
like = [0,0,0,0,0,0,1,0]

Vector dimension: number of words in the vocabulary (e.g. 8)

One-hot representation

Time flies like an arrow.
Fruit flies like a banana.

One-hot representation

Time flies like an arrow.
Fruit flies like a banana.

Vocabulary: {an, arrow,banana, flies, fruit, like, time}

One-hot representation

Time flies like an arrow.
Fruit flies like a banana.

Vocabulary: {an, arrow,banana, flies, fruit, like, time}

	time	fruit	flies	like	a	an	arrow	banana
1_{time}	1	0	0	0	0	0	0	0
1_{fruit}	0	1	0	0	0	0	0	0
1_{flies}	0	0	1	0	0	0	0	0
1_{like}	0	0	0	1	0	0	0	0
1_{a}	0	0	0	0	1	0	0	0
1_{an}	0	0	0	0	0	1	0	0
1_{arrow}	0	0	0	0	0	0	1	0
1_{banana}	0	0	0	0	0	0	0	1

One-hot representation

like a banana

Vocabulary: {an, arrow,banana, flies, fruit, like, time}

	time	fruit	flies	like	a	an	arrow	banana
1_{like}	0	0	0	1	0	0	0	0
1_{a}	0	0	0	0	1	0	0	0
1_{banana}	0	0	0	0	0	0	0	1

"Collapsed" or Binary One-hot representation

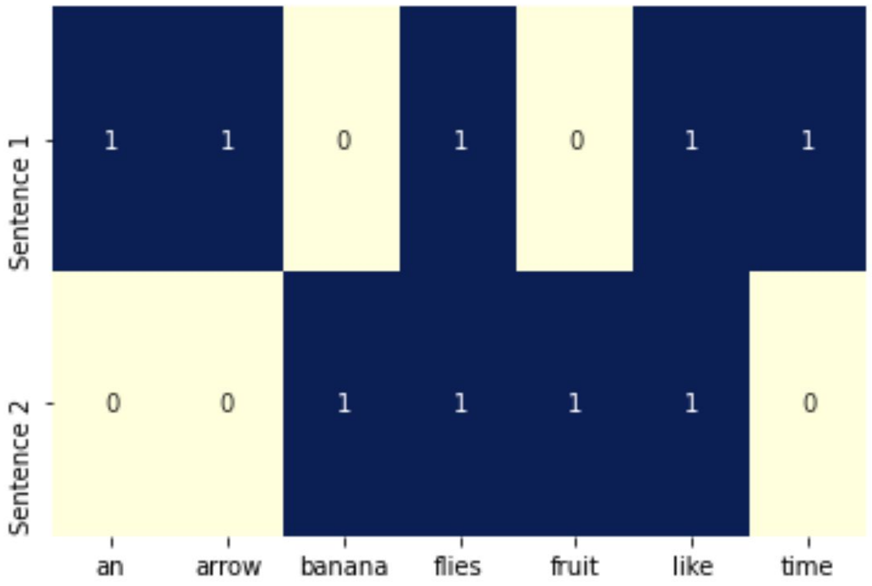
like a banana

Vocabulary: {an, arrow,banana, flies, fruit, like, time}

	time	fruit	flies	like	a	an	arrow	banana
$1_{like}, 1_{a}, 1_{banana}$	0	0	0	1	1	0	0	1

“Collapsed” or Binary One-hot representation

```python
from sklearn.feature_extraction.text import CountVectorizer
import seaborn as sns

corpus = ['Time flies flies like an arrow.',
          'Fruit flies like a banana.'
          ]
one_hot_vectorizer = CountVectorizer(binary = True)

one_hot = one_hot_vectorizer.fit_transform(corpus).toarray()
vocab = one_hot_vectorizer.get_feature_names()

sns.heatmap(one_hot, annot=True,
            cbar=False, xticklabels=vocab,
            yticklabels=['Sentence 1','Sentence 2'],
            cmap="YlGnBu")
```



TF Representation

- TF: **T**erm **F**requency
- TF representation of sentence or document:
  - Sum of the one-hot representations of its constituent words

fruit fruit like a banana

Vocabulary: {an, arrow,banana, flies, fruit, like, time}

time	fruit	flies	like	a	an	arrow	banana
0	2	0	1	1	0	0	1

TF-IDF Representation

IDF: **I**nverse **D**ocument **F**requency

$$\underbrace{IDF(w) = \log \frac{N}{n_w}}$$

Total number of documents

Number of documents containing the word **w**

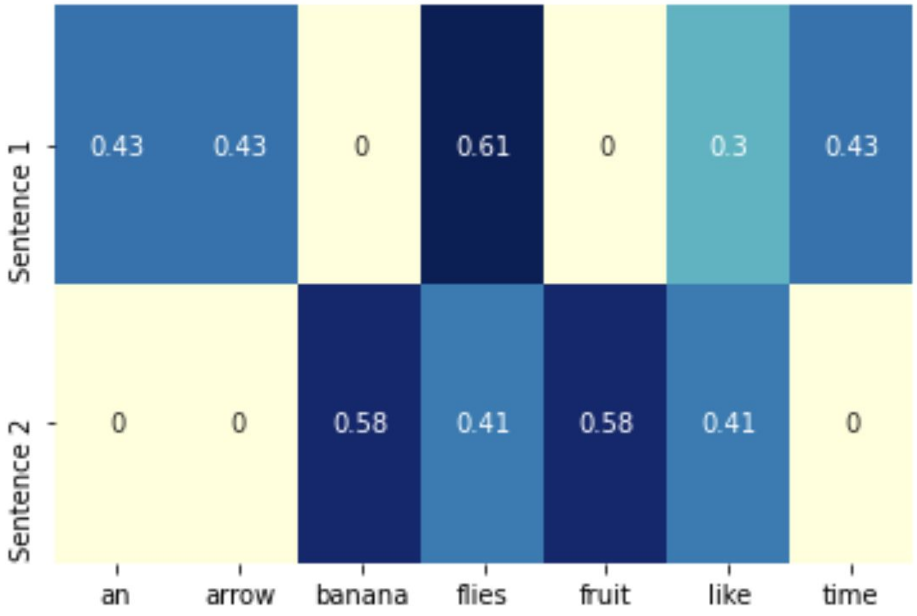
Common tokens get **lower** score
Rare tokens get **higher** score

TF-IDF score:

TF(w)*IDF(w)

TF-IDF

```python
from sklearn.feature_extraction.text import TfidfVectorizer

corpus = ['Time flies flies like an arrow.',
          'Fruit flies like a banana.'
          ]

tfidf_vectorizer = TfidfVectorizer()

tdidf = tfidf_vectorizer.fit_transform(corpus).toarray()
vocab = one_hot_vectorizer.get_feature_names()

sns.heatmap(tdidf, annot=True,
            cbar=False, xticklabels=vocab,
            yticklabels=['Sentence 1','Sentence 2'],
            cmap="YlGnBu")
```



TF-IDF

Common baseline (can be done in unsupervised way)

Sentence similarity, document similarity

Summarization, etc.

Actively used today in production systems

More info: https://web.stanford.edu/~jurafsky/slp3/6.pdf

Computational Graph

Abstraction that models mathematical expressions

E.g.

$y = wx + b$

Can be divided into:

$z = wx$ and $y = z + b$

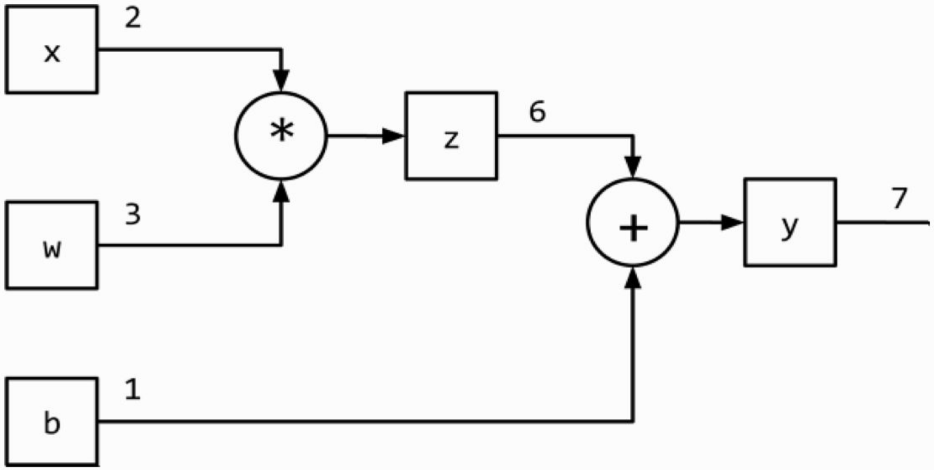


Figure 1-6. Representing $y = wx + b$ using a computational graph.

In Pytorch: implement automatic differentiation for computing gradients during training

Dynamic versus Static Computational Graphs

Static Frameworks

theano

Caffe

mxnet

TensorFlow

Dynamic Frameworks (Recommended!)

dy/net

Chainer

PYTORCH

+Gluon

+Eager

From: http://phontron.com/class/nn4nlp2020/assets/slides/nn4nlp-01-intro.pdf

Pytorch Basics

Core data structure: **Tensor**

Tensor: mathematical object holding some multidimensional data

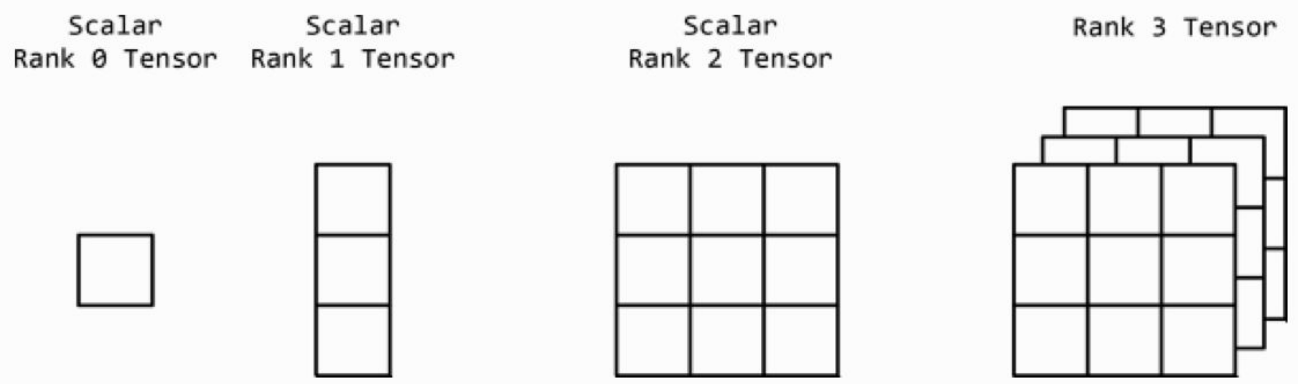


Figure 1-7. Tensors as a generalization of multidimensional arrays.

Creating Tensors

```
import torch

def describe(x):
  print("Type: {}".format(x.type()))
  print("Shape/size {}:".format(x.shape))
  print("Values: \n{}".format(x))

describe(torch.Tensor(2,3))
```

```
Type: torch.FloatTensor
Shape/size torch.Size([2, 3]):
Values:
tensor([[1.6285e-34, 0.0000e+00, 3.7835e-44],
        [0.0000e+00,        nan, 0.0000e+00]])
```

Creating randomly initialized tensors

```
describe(torch.rand(2,3))   #uniform random dist.
print()
describe(torch.randn(2,3))  #random normal dist. (mean=0, variance=1)
```

```
Type: torch.FloatTensor
Shape/size torch.Size([2, 3]):
Values:
tensor([[0.1024, 0.7524, 0.4075],
        [0.7064, 0.8535, 0.9619]])

Type: torch.FloatTensor
Shape/size torch.Size([2, 3]):
Values:
tensor([[ 0.3324,  0.8341,  0.7928],
        [ 1.2872, -0.4749, -1.4327]])
```

Creating a filled tensor

```
describe(torch.zeros(2,3)) #tensor of zeros
x = torch.ones(2,3)  #tensor of ones
describe(x)
x.fill_(5) #filling witg value 5
describe(x)
```

```
Type: torch.FloatTensor
Shape/size torch.Size([2, 3]):
Values:
tensor([[0., 0., 0.],
        [0., 0., 0.]])

Type: torch.FloatTensor
Shape/size torch.Size([2, 3]):
Values:
tensor([[1., 1., 1.],
        [1., 1., 1.]])

Type: torch.FloatTensor
Shape/size torch.Size([2, 3]):
Values:
tensor([[5., 5., 5.],
        [5., 5., 5.]])
```

Creating and initializing a tensor from lists

```
x = torch.Tensor([[1,2,3], #inializing a tensor from lists
                  [4,5,6]])
describe(x)
```

```
Type: torch.FloatTensor
Shape/size torch.Size([2, 3]):
Values:
tensor([[1., 2., 3.],
        [4., 5., 6.]])
```

Creating and initializing a tensor from numpy

```
import numpy as np

npy = np.random.rand(2,3)
describe(torch.from_numpy(npy))
```

```
Type: torch.DoubleTensor
Shape/size torch.Size([2, 3]):
Values:
tensor([[0.4260, 0.1411, 0.9704],
        [0.0330, 0.0276, 0.7927]], dtype=torch.float64)
```

Tensor Types and Size

```
x = torch.FloatTensor([[1,2,3],
                       [4,5,6]])
describe(x)
```

```
Type: torch.FloatTensor
Shape/size torch.Size([2, 3]):
Values:
tensor([[1., 2., 3.],
        [4., 5., 6.]])
```

```
x = x.long()
describe(x)
```

Converts a tensor to long type

```
Type: torch.LongTensor
Shape/size torch.Size([2, 3]):
Values:
tensor([[1, 2, 3],
        [4, 5, 6]])
```

Tensor operations: addition

```
x = torch.randn(2,3)
describe(x)
describe(torch.add(x,x))
describe(x+x)
```

Two ways to do addition

```
Type: torch.FloatTensor
Shape/size torch.Size([2, 3]):
Values:
tensor([[ 0.7481,  1.2792, -0.8766],
        [ 0.6250,  0.9984, -0.2227]])

Type: torch.FloatTensor
Shape/size torch.Size([2, 3]):
Values:
tensor([[ 1.4963,  2.5583, -1.7532],
        [ 1.2500,  1.9967, -0.4454]])

Type: torch.FloatTensor
Shape/size torch.Size([2, 3]):
Values:
tensor([[ 1.4963,  2.5583, -1.7532],
        [ 1.2500,  1.9967, -0.4454]])
```

Dimension-based tensor operations

```
x = torch.arange(6.)
describe(x)

x = x.view(2,3)
describe(x)
```

Same data, but different shape

```
Type: torch.FloatTensor
Shape/size torch.Size([6]):
Values:
tensor([0., 1., 2., 3., 4., 5.])

Type: torch.FloatTensor
Shape/size torch.Size([2, 3]):
Values:
tensor([[0., 1., 2.],
        [3., 4., 5.]])
```

Dimension-based tensor operations

```
x = torch.arange(6.)
x = x.view(2,3)
describe(x)
describe(torch.sum(x, dim=0))
describe(torch.sum(x, dim=1))
```

Sum elements on dimension 0 (row)

```
Type: torch.FloatTensor
Shape/size torch.Size([2, 3]):
Values:
tensor([[0., 1., 2.],
        [3., 4., 5.]])

Type: torch.FloatTensor
Shape/size torch.Size([3]):
Values:
tensor([3., 5., 7.])

Type: torch.FloatTensor
Shape/size torch.Size([2]):
Values:
tensor([ 3., 12.])
```

Sum elements on dimension 1 (column)

Dimension-based tensor operations

```
x = torch.arange(6.)
x = x.view(2,3)
describe(x)
describe(torch.transpose(x,0,1))
```

Dimensions 0 and 1 are swapped

```
Type: torch.FloatTensor
Shape/size torch.Size([2, 3]):
Values:
tensor([[0., 1., 2.],
        [3., 4., 5.]])

Type: torch.FloatTensor
Shape/size torch.Size([3, 2]):
Values:
tensor([[0., 3.],
        [1., 4.],
        [2., 5.]])
```

Complex indexing: noncontiguous indexing of a tensor

```
x = torch.arange(6.).view(2,3)
describe(x)
indices = torch.LongTensor([0,2])
describe(indices)
describe(torch.index_select(x,dim=1,index=indices))
```

```
Type: torch.FloatTensor
Shape/size torch.Size([2, 3]):
Values:
tensor([[0., 1., 2.],
        [3., 4., 5.]])

Type: torch.LongTensor
Shape/size torch.Size([2]):
Values:
tensor([0, 2])

Type: torch.FloatTensor
Shape/size torch.Size([2, 2]):
Values:
tensor([[0., 2.],
        [3., 5.]])
```

Concatenating tensors

```
x = torch.arange(6.).view(2,3)
describe(x)
describe(torch.cat([x,x], dim=0))
describe(torch.cat([x,x], dim=1))
describe(torch.stack([x,x]))
```

```
Type: torch.FloatTensor
Shape/size torch.Size([2, 3]):
Values:
tensor([[0., 1., 2.],
        [3., 4., 5.]])

Type: torch.FloatTensor
Shape/size torch.Size([4, 3]):
Values:
tensor([[0., 1., 2.],
        [3., 4., 5.],
        [0., 1., 2.],
        [3., 4., 5.]])

Type: torch.FloatTensor
Shape/size torch.Size([2, 6]):
Values:
tensor([[0., 1., 2., 0., 1., 2.],
        [3., 4., 5., 3., 4., 5.]])

Type: torch.FloatTensor
Shape/size torch.Size([2, 2, 3]):
Values:
tensor([[[0., 1., 2.],
         [3., 4., 5.]],

        [[0., 1., 2.],
         [3., 4., 5.]]])
```

Tensor multiplication

```
x1 = torch.arange(6.).view(2,3)
describe(x1)
x2 = torch.ones(3,2)
x2[:,1] += 1
describe(x2)
describe(torch.mm(x1,x2))
```

```
Type: torch.FloatTensor
Shape/size torch.Size([2, 3]):
Values:
tensor([[0., 1., 2.],
        [3., 4., 5.]])

Type: torch.FloatTensor
Shape/size torch.Size([3, 2]):
Values:
tensor([[1., 2.],
        [1., 2.],
        [1., 2.]])

Type: torch.FloatTensor
Shape/size torch.Size([2, 2]):
Values:
tensor([[ 3.,  6.],
        [12., 24.]])
```

Creating tensors for gradient bookkeeping

```
x = torch.ones(2,2, requires_grad=True)
describe(x)
print(x.grad is None)
print()

y = (x+2)*(x+5) + 3
describe(y)
print(x.grad is None)
print()

z = y.mean()
describe(z)
z.backward()
print(x.grad is None)
```

```
Type: torch.FloatTensor
Shape/size torch.Size([2, 2]):
Values:
tensor([[1., 1.],
        [1., 1.]], requires_grad=True)

True

Type: torch.FloatTensor
Shape/size torch.Size([2, 2]):
Values:
tensor([[21., 21.],
        [21., 21.]], grad_fn=<AddBackward0>)

True

Type: torch.FloatTensor
Shape/size torch.Size([]):
Values:
21.0
```

CUDA Tensors

```python
print(torch.cuda.is_available())

#device agnostic tensor instantiation
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)
print()
x = torch.rand(3,3).to(device)
describe(x)
```

```
True
cuda

Type: torch.cuda.FloatTensor
Shape/size torch.Size([3, 3]):
Values:
tensor([[0.3205, 0.6094, 0.1437],
        [0.9105, 0.3578, 0.6457],
        [0.3558, 0.8847, 0.1556]], device='cuda:0')
```

Mixing CUDA tensors with CPU-bound errors

```
y = torch.rand(3,3)
x + y
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-52-2de5f9fcdd39> in <module>()
      1 y = torch.rand(3,3)
----> 2 x + y
      3 
      4 cpu_device = torch.device("cpu")

RuntimeError: expected device cuda:0 but got device cpu
```

SEARCH STACK OVERFLOW

```
cpu_device = torch.device("cpu")
y = y.to(cpu_device)
x = x.to(cpu_device)
x + y
```

```
tensor([[0.9282, 0.6427, 0.9240],
        [1.2185, 0.4185, 1.3111],
        [1.2352, 1.0999, 0.5164]])
```

For next week

Try to practice the examples

Try to practice the exercises

Install spacy.io on your machine

Is there anyone who wants to present next week?