Session 1

Preliminaries & PyTorch

Deep learning - Fall 2020

Deep Learning Frameworks

What is a deep learning framework?

An interface, library or a tool which allows us to build deep learning models more easily and quickly, without getting into the details of underlying algorithms.

1. Optimized for performance
2. Easy to understand and code
3. Good community support
4. Parallelize the processes to reduce computations
5. Automatically compute gradients

Why we need a framework?

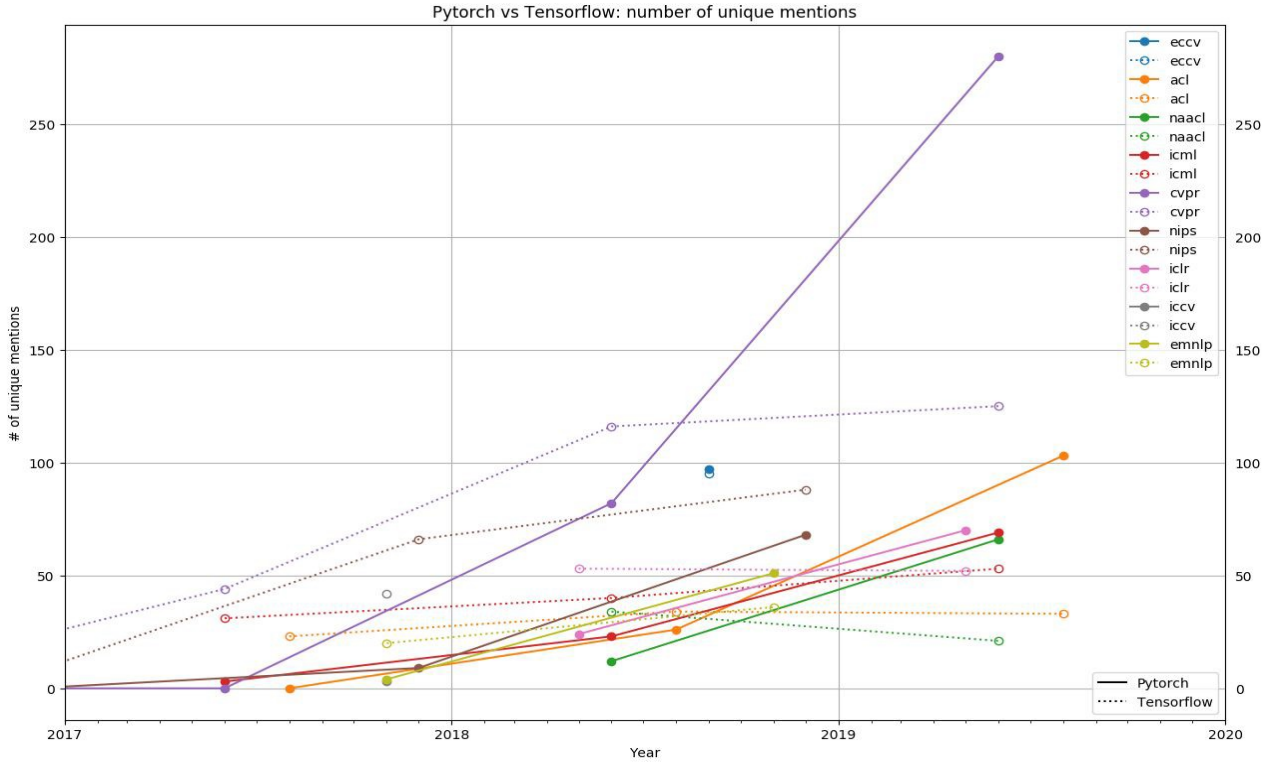
Due to,

- Computational complexity of forward and backward pass in a network
- Need to use hardware efficiently

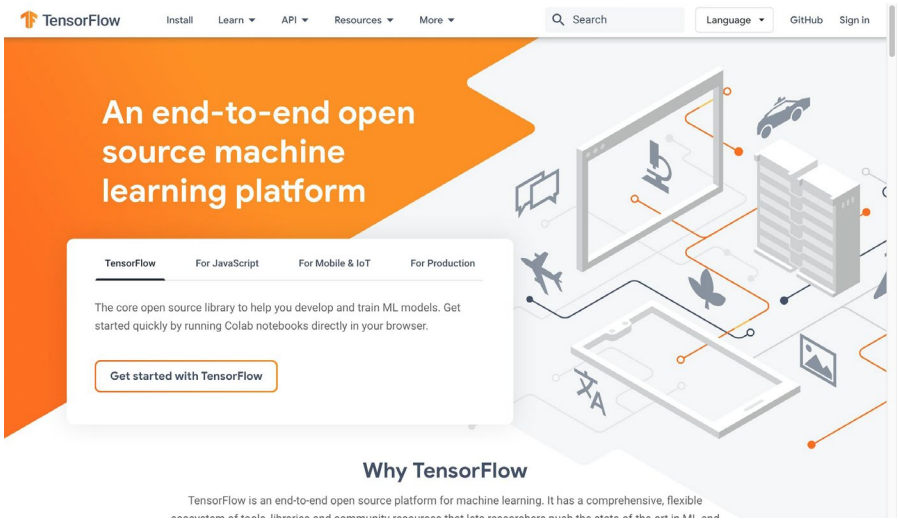
without frameworks,

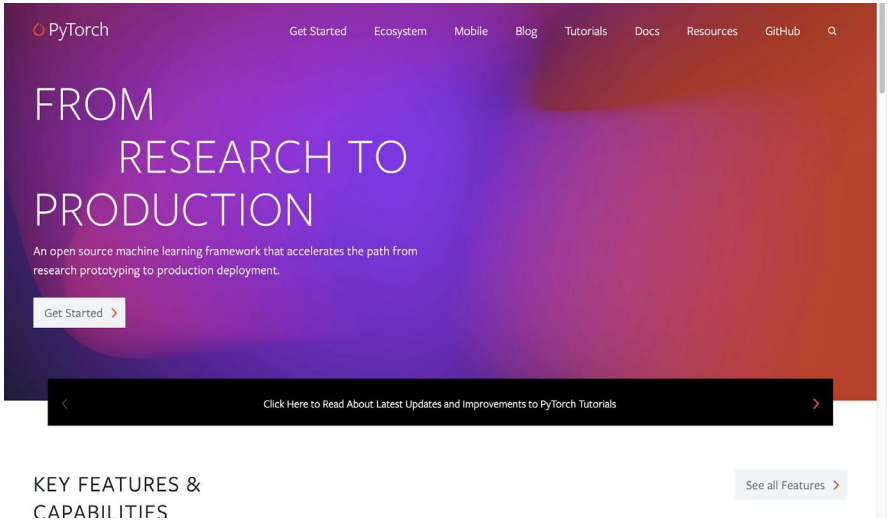
- Having really deep networks would be impossible!
- No place for complex architectures and new ideas
- Minimum effect on science and technology
- No standard way of implementation

Tensorflow vs. Pytorch



Tensorflow vs. Pytorch (cont)





Tensorflow

Pytorch

Tensorflow vs. Pytorch (cont)

1. Tensorflow is based on Theano and has been developed by Google.
   PyTorch is based on Torch and has been developed by Facebook.

2. Tensorflow creates a static graph.
   PyTorch believes in a dynamic graph.

3. Learning pytorch seems to be easier than tensorflow due to its structure.

4. Tensorflow is older than pytorch so it has a much bigger community behind it than PyTorch.

5. Tensorflow is better for production models and scalability.
   PyTorch is relatively better for passion projects and building rapid prototypes.

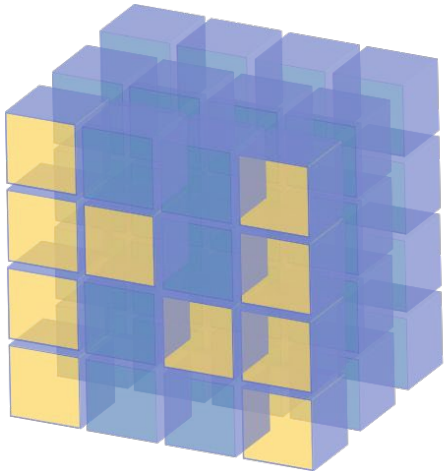
Preliminaries (Python, NumPy)

Why Python?

- Easy-to-use language
- Great community participation
- Decent library availability (especially machine learning libraries)

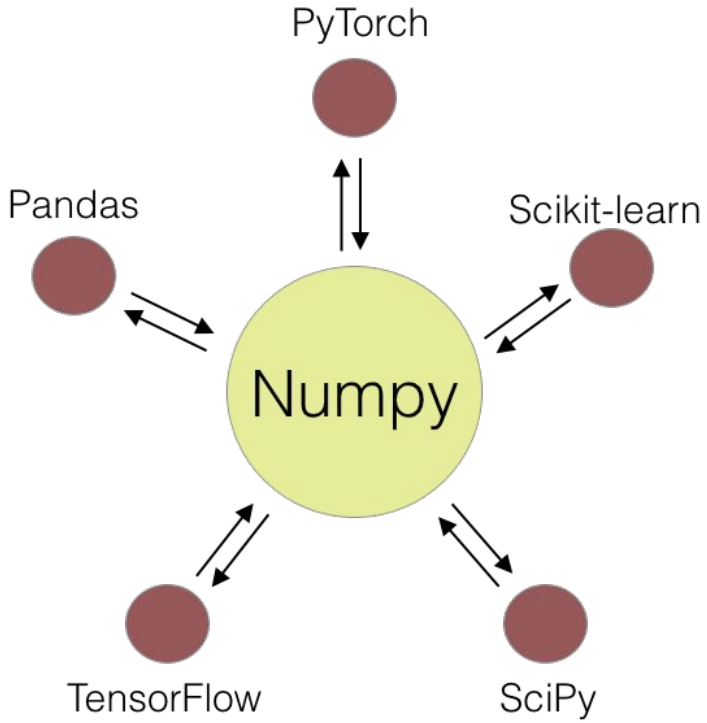
Numpy

- Core library for scientific computing in Python
- Appropriate for processing homogeneous multidimensional arrays and matrices



NumPy

Numpy



Numpy

Array

- A numpy array is a grid of values, all of the same type
- number of dimensions is the *ndim (rank)* of the array
- The *shape* of an array is a tuple of integers giving the size of array along each dimension
- We can initialize numpy arrays from nested Python lists, and access elements using square brackets

Numpy

- Numpy is Fast
- Numpy arrays are densely packed arrays of homogeneous type (locality of reference)
- Many Numpy operations are implemented in C
- e.g. if you are summing up two arrays the addition will be performed with the specialized CPU vector operations

> [See the example for this part in the notebook]

Numpy

- Uses much less memory to store data (compared to Python lists)

```
np.array([1, 2, 3, 4], dtype=np.int8)
```

- Simple and beautiful API
- **Indexing**
- **Broadcasting**: Working with arrays of different shapes

> [See the examples for this part in the notebook]

Numpy

- Ex. Linear regression in Numpy

model → $h(x) = \sum_{i=0}^{d} \theta_i x_i = \theta^T x$

loss → $J(\theta) = \frac{1}{2} \sum_{i=1}^{n} (h_\theta(x^{(i)}) - y^{(i)})^2$

  - LMS algorithm

$$\text{Repeat until convergence } \{$$

$$\theta_j := \theta_j + \alpha \sum_{i=1}^{n} \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)} \qquad \text{(for every } j\text{)}$$

$$\}$$

  - The Normal Equation

$$X^T X \theta = X^T \vec{y} \longrightarrow \theta = (X^T X)^{-1} X^T \vec{y}$$

PyTorch

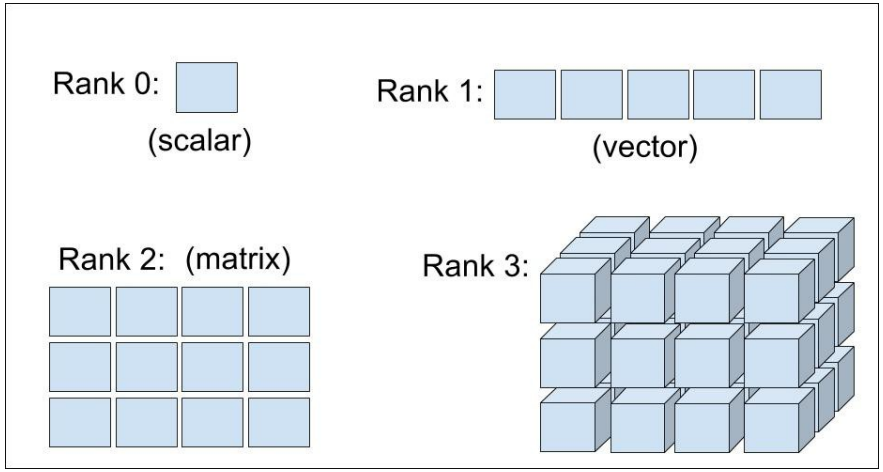
PyTorch

- Initial release: October 2016
- Written in C++, Python, CUDA
- **Can use power of GPUs**
- **Using tape-based automatic differentiation**
- Python-Approach (When you execute a line of code, it gets executed. There isn't an asynchronous view of the world)
- Easier To Learn And Simpler To Code (than TensorFlow)
- Dynamic computational graph

PyTorch

Tensor

- multi-dimensional matrix containing elements of a single data type
- Torch defines nine CPU tensor types and nine GPU tensor types
- PyTorch Tensor API looks almost exactly like Numpy!



PyTorch

Tensor

- `torch.Tensor()`

```
>>> import torch
>>> t = torch.Tensor()
>>> t
tensor([])
>>> print(type(t))
<class 'torch.Tensor'>
>>> print(t.dtype)
torch.float32
>>> print(t.device)
cpu
```

PyTorch

Tensor

- A tensor can be constructed from a Python list or sequence using the `torch.tensor()` constructor:

```
>>> torch.tensor([[1., -1.], [1., -1.]])
tensor([[ 1., -1.],
        [ 1., -1.]])
>>> torch.tensor(np.array([[1, 2, 3], [4, 5, 6]]))
tensor([[1, 2, 3],
        [4, 5, 6]])
```

PyTorch

Tensor

- Four important parameters:
  - **data** (array_like) → `numpy.ndarray` or `list`
  - **dtype** (`torch.dtype`) → `torch.float32, torch.int32, ...`
  - **device** (`torch.device`) → `torch.device('cpu')` or `torch.device('cuda:0')`
  - **requires_grad** (bool) → `True` or `False`

PyTorch

Tensor

- Important Tensor creation Ops
  - **`torch.zeros`** & **`torch.zeors_like`**
  - `torch.ones` & `torch.ones_like`
  - **`torch.empty`** & `torch.empty_like`
  - `torch.eye`
  - `torch.full` & `torch.full_like`
  - **`torch.rand`**, **`torch.randn`**, **`torch.randint`**, ...
  - `torch.arange`

PyTorch

Tensor

- Constructing a tensor

```
>>> torch.zeros([2, 4], dtype=torch.int32)
tensor([[0, 0, 0, 0],
        [0, 0, 0, 0]], dtype=torch.int32)
>>> cuda0 = torch.device('cuda:0')
>>> torch.ones([2, 4], dtype=torch.float64, device=cuda0)
```

PyTorch

Tensor

- Accessing the contents of a Tensor

```
>>> x = torch.tensor([[1, 2, 3], [4, 5, 6]])
>>> print(x[1, 2])
tensor(6)
>>> x[0][1] = 8
>>> print(x)
tensor([[1, 8, 3],
        [4, 5, 6]])
```

PyTorch

Tensor

- Use `Tensor.item()` to get a Python number from a tensor containing a single value:

```
>>> x = torch.tensor([[1]])
>>> x
tensor([[1]])
>>> x.item()
1
>>> x = torch.tensor(2.5)
>>> x
tensor(2.5000)
>>> x.item()
2.5
```

PyTorch

Tensor

- Creating Tensors from Numpy arrays
  - `from_numpy()`
  - `as_tensor()`

```
>>> a = np.array([1, 2, 3])
>>> t = torch.as_tensor(a)
>>> t
tensor([1, 2, 3])
>>> t[0] = -1
>>> a
array([-1,  2,  3])
```

```
>>> a = np.array([1, 2, 3])
>>> t = torch.from_numpy(a)
>>> t
tensor([1, 2, 3])
>>> t[0] = -1
>>> a
array([-1,  2,  3])
```

PyTorch

Tensor

- Important operations on Tensors (`te` is a Tensor)
  - `te.cat()` & `te.stack()`
  - `te.reshape()`
  - **`te.squeeze()` & `te.unsqueeze()`**
  - `te.t()`
  - `te.permute()`
  - `te.max()` & `te.min()`
  - `te.argmax()` & `te.argmin()`
  - `te.sum()`
  - `te.clone()`

PyTorch

Tensor

- Important operations on Tensors (`te` is a Tensor)
  - **`te.to(torch.device)`** -> ex. `te.to('cuda:0')`
  - `te.cpu()` & `te.cuda()`
  - **`te.float(), te.long(), te.double(), ...`**
  - **`te.view(*shape)`**
  - `te.numpy()`
  - `te.dot()`
  - `te.mm()` & `te.matmul()`
  - `te.bmm()`
  - `te.topk()`

PyTorch

Tensor

- Important operations on Tensors

  Note: Methods which mutate a tensor are marked with an underscore suffix.

  - `te.empty_()`
  - `te.random_()`
  - `te.add_()`
  - `te.sub_()`
  - `te.clamp_()`

Pytorch

Tensor

- Serialization
  - `Torch.save()`

```
>>> import torch
>>> x = torch.tensor([0, 1, 2 , 3, 4])
>>> torch.save(x, 'tensor.pt')
```

  - `Torch.load()`

```
>>> x = torch.load('tensor.pt')
```

Links and references

- You can find a great tutorial on Python and Numpy here.
- A great IPynotebook tutorial on Numpy is here.
- Read the Tensor documentation here.
- The documentation of different Tensor Creation operations can be found here.
- PyTorch Github Readme is here.

Acknowledgements

- This material is partially taken from deep learning workshops and TA sessions of CE719 Fall 2019 and Spring 2020 and CE550 Spring 2020 at Sharif University of Technology.