31.7.-10.8.

PETNICA SUMMER INSTITUTE

MACHINE LEARNING 6

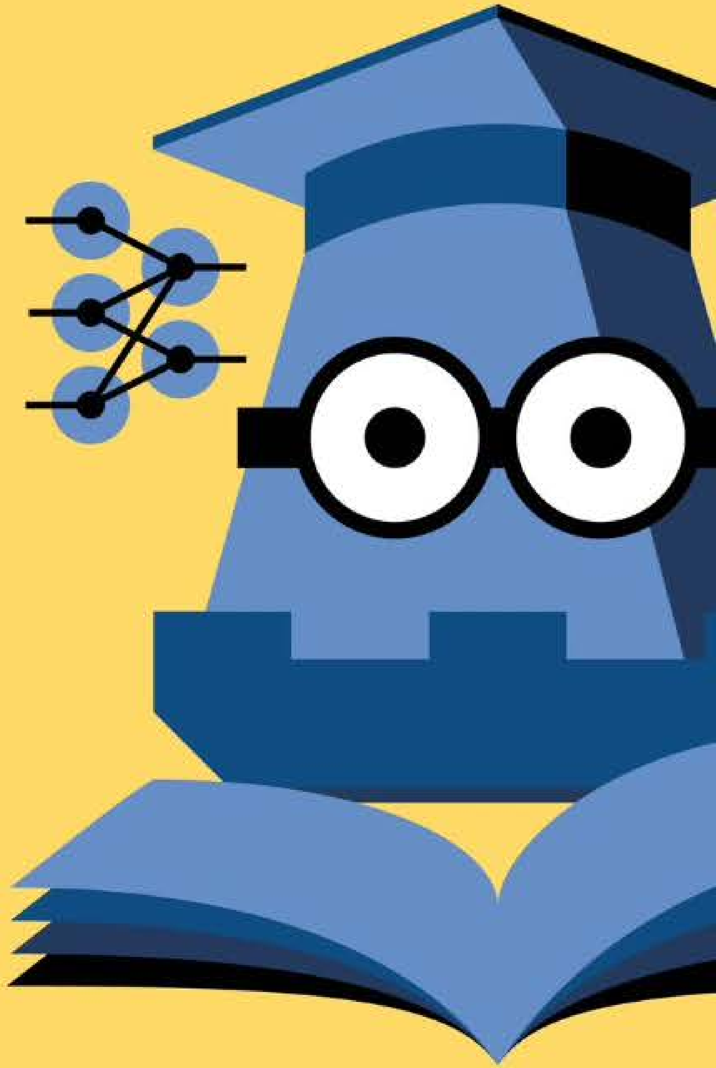
PETNICA SUMMER INSTITUTE MACHINE LEARNING 6

PyTorch Intro

Overview of PyTorch for beginner ML practitioners

Stefan Mojsilovic

Everseen



ML/DL Frameworks

- Could we write all ML code in NumPy?

- Why shouldn't we write all ML code in NumPy?

ML/DL Frameworks

- Can we write all ML/DL code in NumPy?
  - Sure!
- Why shouldn’t we write all ML/DL code in NumPy?
  - Missing a lot of ML/DL specific functions
  - No GPU support
  - No Automatic differentiation

ML/DL Frameworks

- A set of tools allowing you to focus on specific tasks
- A skeleton upon which you build an application
- Takes care of the low level stuff
- Allows you to concentrate on what maters the most

ML/DL Frameworks

PyTorch vs Tensorflow

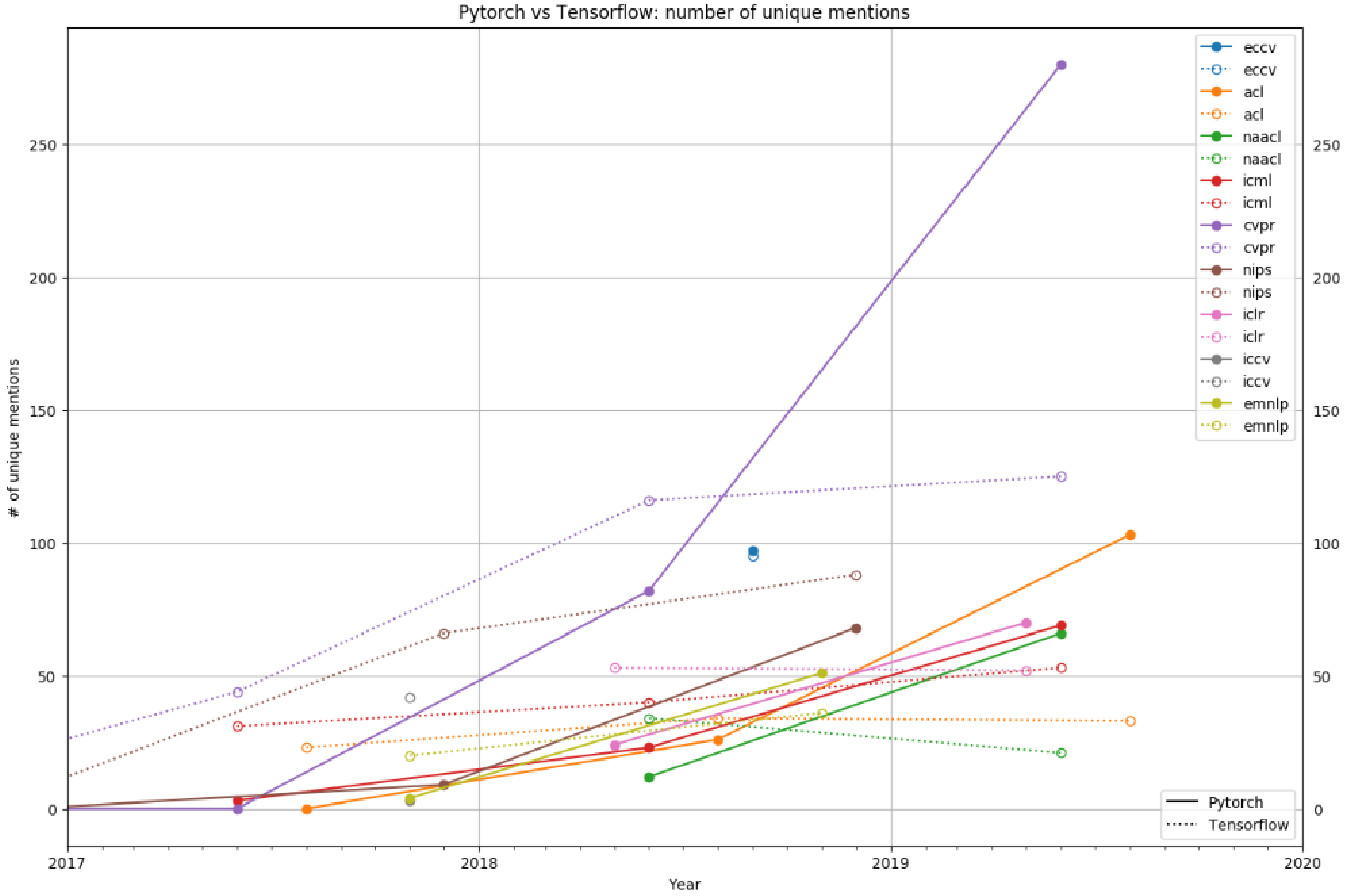


Image source: https://thegradient.pub/

PyTorch vs other frameworks

- **PyTorch** (Facebook, IBM, Yandex, OpenAI) dominates research applications
- **Tensorflow** (Google, Uber, ABnB) and **MXNET** (Amazon, Microsoft, Intel) dominate industry applications
  - Exception is caffe2 (now part of pytorch) which is Facebook's default production environment

- Useful links:
  - https://www.netguru.com/blog/deep-learning-frameworks-comparison
  - https://thegradient.pub/state-of-ml-frameworks-2019-pytorch-dominates-research-tensorflow-dominates-industry/
  - https://executecommands.com/machine-learning-frameworks-2020/

What is PyTorch?

Python-based package for scientific computing with two goals:

- A replacement for NumPy to use the power of GPUs
- A deep learning research platform w/ maximum flexibility and speed

Why is it good?

- Intuitive, low-level, highly modular, easy to extend, open source
- High growth – numerous libraries and support by the devs
- Massively adopted by the research community

PyTorch as Numpy replacement - torch.Tensor

Numpy	**PyTorch**
x = numpy.array**(***object***,** *dtype=None, …***)**	x = torch.tensor(*data*, *dtype=None*, *device=None, requires_grad=False*)
.ones() .zeros()	.ones() .zeros()
.max() .mean() .sum()	.max() .mean() .sum()
.random.rand() .random.normal() .random.randint()	.rand() .normal() .randint()
x.shape x.ndim len(x) x.size x.dtype x[0, :, 1:3]	x.shape x.ndim len(x) **x.size()** x.dtype x[0, :, 1:3]
+ - * /	+ - * /
.matmul(x, y)	**.mm(x, y)**
x.reshape() x.T x.copy()	**x.reshape() x.view() x.t() x.clone()**
x.squeeze() x.expand_dims()	x.squeeze() **x.unsqueeze**
np.save('file.npy', x) np.load('file.npy')	torch.save(x, 'file.pth') torch.load('file.pth‘)

From CPU to GPU and back

1. x = x.to(device="cuda")
2. x = x.to(device="cpu")

Literally as simple as that ☺

- Arithmetic and torch functions on GPU tensors will be done on GPU
- Note: mixing operands with different devices raises a RuntimeError!

PyTorch as a DL Research Platform

- torch.cuda -> tools for enabling GPU computation
- torch.autograd -> tools for automatic differentiation
- torch.data -> tools for loading your data efficiently
- torch.nn -> tools for building your ML models
- torch.optim -> tools for optimizing your models
- Libraries -> torchaudio, torchtext, torchvision…
- And many, many, many more:
  - https://pytorch.org/ecosystem/
  - https://pytorch.org/hub/

torch.cuda

- Info on CUDA-enabled devices
  - .is_available()
  - .device_count()
  - .current_device()
  - .get_device_name()
- Selecting a device
  - device = torch.device(“cuda:n”), where n=0,1,2…
  - .set_device(device)
- Advanced: Explicit control over streams, memory, events, sync…

```python
print(torch.cuda.is_available())
print(torch.cuda.device_count())
print(torch.cuda.current_device())
print(torch.cuda.get_device_name())
```

```
True
1
0
GeForce GTX 1060
```

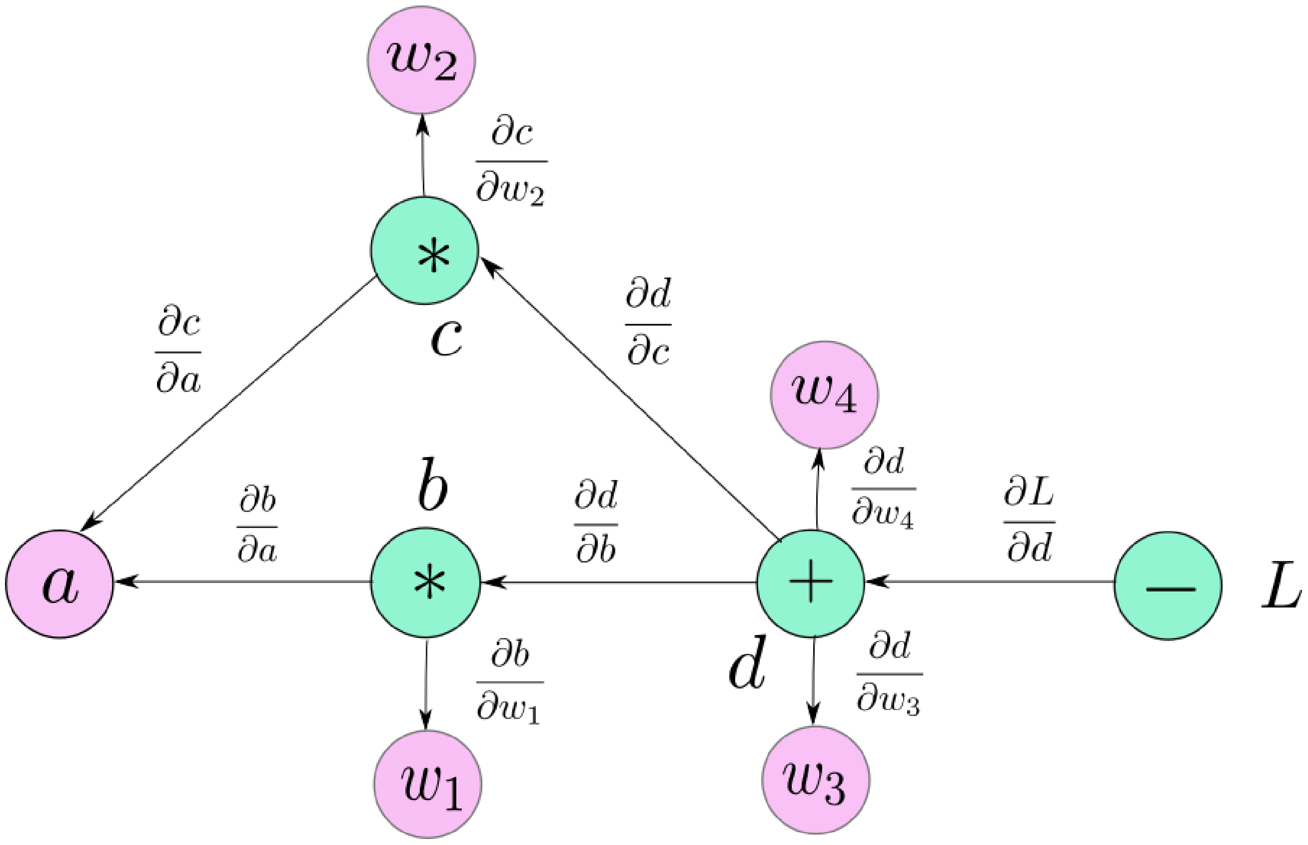
torch.autograd
Dynamic Computational Graph

$$b = w_1 * a$$

$$c = w_2 * a$$

$$d = w_3 * b + w_4 * c$$

$$L = 10 - d$$



torch.autograd

- .Function
  - .forward()
  - .backward()
- Tensor attributes and methods
  - .requires_grad
  - .is_leaf
  - .grad_fn
  - .grad
  - .detach()
- Backprop with .backward()
  - On scalar valued tensors only
  - Frees the non-leaf buffers and destroys the graph
  - Remember to zero the gradients between .backward() calls, otherwise they will be accumulated

Note:
- To turn off requires_grad globally: torch.set_grad_enabled(False)
- Or as context manager - with torch.no_grad():

torch.utils.data

- .Dataset -> feat, label = dataset[idx]
  - .__init__()
  - .__len__()
  - .__get_item__(idx)
- .Dataloader -> for feat, label in dl …
  - Shuffling
  - Batching
  - Prefetching
  - Multiprocessing

```python
class ImageDataset(Dataset):
    """Image dataset for classification."""

    def __init__(self, root_dir, csv_file, transform=None):
        """
        Args:
            root_dir (string): Directory with all the images.
            csv_file (string): Path to the csv file with paths to images
                               relative to the root_dir, and their class labels.
            transform (callable): Optional transform to be applied on a sample.
        """
        self.root_dir = root_dir
        self.image_dataframe = pd.read_csv(csv_file)
        self.transform = transform

    def __len__(self):
        return len(self.image_dataframe)

    def __getitem__(self, idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()

        img_filepath = os.path.join(self.root_dir,
                                    self.image_dataframe.iloc[idx, 0])
        image = cv2.imread(img_filepath)
        class_label = self.image_dataframe.iloc[idx, 1]

        if self.transform:
            image = self.transform(image)

        return image, label
```

torch.nn.Module

- Used for building NNs and their components
  - Layers (fc, conv, pooling, normalization, dropout…)
  - Activations (sigmoid, softmax, ReLU…)
  - Losses (BCE, CE, MSE, triplet)
- Or as a container for already built components
  - Manually adding them to the Module
  - Or using nn.Sequential

```
model = nn.Sequential(
            nn.Linear(100, 10),
            nn.ReLU(),
            nn.Linear(10, 1),
            nn.Sigmoid() )
```

torch.nn.Module

- When defining a model, subclass Module and implement:
  - .__init__() – define the parameters of the object
  - .forward() – define the computational graph

```python
import torch.nn as nn
import torch.nn.functional as F
class Model(torch.nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.layer1 = torch.nn.Linear(in_features=100, out_features=10, bias=True)
        self.layer2 = torch.nn.Linear(in_features=10, out_features=1, bias=True)

    def forward(self, x):
        x = F.relu(self.layer1(x))
        x = self.layer2(x)
        return x
```

- However, when calling the instance of your model class
  - USE: output = model(input)
  - NOT: output = model.forward(input), as it will disregard hooks

torch.optim

- Optimizers – optimization algorithm (SGD, Adam…)
  - optimizer=optim.NAME(model.parameters(), **kwargs)
  - optimizer.zero_grad()
  - … model output, loss, loss.backward()
  - optimizer.step() – uses gradients computed by .backward() to update model.parameters()
- LR Schedulers – update optimizer hyper-parameters
  - lr_scheduler=optim.lr_scheduler.NAME(optimizer, **kwargs)
  - lr_scheduler.step() – updates the optimizer hyper-parameters

Congratulations!

PyTorch Intro

By Stefan Mojsilovic

stefan.mjs+psiml@gmail.com

Working in Everseen Ltd. on deep learning applications in computer vision and large-scale retail. Interested in cognitive models and augmented intelligence.