PyTorch and AI Basics

Dhruv Makwana

What is PyTorch and why learn PyTorch?

- PyTorch is an optimized Deep Learning tensor library based on Python and Torch.
- The main use of PyTorch is mainly for applications using GPUs and CPUs.
- PyTorch is favored over other Deep Learning frameworks like TensorFlow and Keras since it uses dynamic computation graphs.

Code and Presentation: https://github.com/Ignitarium-AI/PyTorch-Tutorial

Installation

Requirements:

- Python

Step-1: Visit following website, Select the preferences and run the install command.

Web: https://pytorch.org/get-started/locally/

Optional: create seperate virtual environment for installation of pytorch

```
python3 -m venv pytorch
source pytorch/bin/activate
```

Step-2: Verify installation with following sequence of commands in terminal.

```
python
>>> import torch
>>> print(torch.__version__)
>>> exit()
```

Tensor Basics

Creation of simple Tensor

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

We can also specify data type while creating tensor

```
>>> tensor_a = torch.tensor([[1, 2, 3], [4, 5, 6]], dtype=torch.float16)
>>> print(tensor_a)
```

Above command will create tensor_a in float16 data type

References: https://pytorch.org/docs/stable/tensors.html#data-types

Tensor Basics

We can also specify device type while creating tensor

```
>>> tensor_a = torch.tensor([[1, 2, 3], [4, 5, 6]], dtype=torch.float32,
    device="cpu")
>>> print(tensor_a)
```

Above comand will create two rows and three columns tensor with float32 values on specified device.

using cuda instead:

```
>>> tensor_a = torch.tensor([[1, 2, 3], [4, 5, 6]], dtype=torch.float32,
    device="cuda:0")
```

(Q). how to check if torch is cuda enabled

```
>>> torch.cuda.is_available() # returns True or False
```

Tensor Basics

Usual practise is that we decide device in starting

```
>>> device = "cuda:0" if torch.cuda.is_available() else "cpu" # here 0 signifies the device id
>>> tensor_a = torch.tensor([[1, 2, 3], [4, 5, 6]], dtype=torch.float32, device=device) # two rows and three columns tensor with float32 values on specified device
>>> print(tensor_a)
```

References:

https://pytorch.org/docs/stable/tensor_attributes.html#torch-device

Tensor Basics

Common Initialization methods:

- Uninitialized data:

```
>>> x = torch.empty(size=(10, 10))
```

reference:
https://pytorch.org/docs/stable/generated/torch.empty.html

- All zero data

```
>>> x = torch.zeros(size=(10, 10))
```

References:
https://pytorch.org/docs/stable/generated/torch.zeros.html

Tensor Basics

Common Initialization methods:

- All ones data:

```
>>> x = torch.ones(size=(10, 10))
```

reference:
https://pytorch.org/docs/stable/generated/torch.ones.html

- All random data

```
>>> x = torch.rand(size=(5, 5))
```

References:
https://pytorch.org/docs/stable/generated/torch.rand.html

Tensor Basics

Common Initialization methods:

- All random numbers with uniform distribution between with given mean and std dev

```
>>> x = torch.empty(size=(5, 5)).normal_(mean=0, std=2)
```

NOTE: "_" after normal means that it is an inplace operation

References:

https://pytorch.org/docs/stable/generated/torch.rand.html

- Create evenly spaced values between given range

```
>>> x = torch.linspace(start=1, end=54, steps=13)
```

References:

https://pytorch.org/docs/stable/generated/torch.linspace.html

Tensor Basics

Common Initialization methods:

- Identity matrix:

```
>>> x = torch.eye(5, 5)
```

reference:
https://pytorch.org/docs/stable/generated/torch.eye.html

- Create list of values

```
>>> x = torch.arange(start=0, end=100, step=20)
```

References:
https://pytorch.org/docs/stable/generated/torch.arange.html

Tensor Basics

Changing tensor types

```
>>> tensor = torch.tensor([-1, 0, 1, 2, 3]) # int64
>>> print(tensor.bool()) # [True, false, True, True, True]
>>> print(tensor.short()) # int16
>>> print(tensor.long()) # int64
>>> print(tensor.half()) # float16
>>> print(tensor.float()) # float32
>>> print(tensor.double()) # float64
```

Tensor Basics

Convert between numpy and torch tensor

```
>>> import numpy as np
>>> numpy_array = np.random.rand(5, 5)
>>> print(numpy_array)
>>> torch_array = torch.from_numpy(numpy_array)
>>> print(torch_array)
>>> numpy_array_recon = torch_array.numpy()
>>> print(numpy_array_recon)
```

References:

https://pytorch.org/docs/stable/generated/torch.from_numpy.html

https://pytorch.org/docs/stable/generated/torch.Tensor.numpy.html

Tensor Basics

Common Mathematic operations

- Addition

Method-1:

```
>>> x = torch.tensor([1, 2, 3])
>>> y = torch.tensor([4, 5, 6])
>>> z = torch.empty(3)
>>> torch.add(x, y, out=z)
```

Method-2:

```
>>> x = torch.tensor([1, 2, 3])
>>> y = torch.tensor([4, 5, 6])
>>> z = torch.add(x, y)
```

Tensor Basics

Common Mathematic operations

- Addition

Method-3:

```
>>> x = torch.tensor([1, 2, 3])
>>> y = torch.tensor([4, 5, 6])
>>> z = x + y
```

Inplace addition

```
>>> x = torch.tensor([1, 2, 3])
>>> y = 2
>>> x.add_(y)
```

References:

https://pytorch.org/docs/stable/generated/torch.add.html

https://pytorch.org/docs/stable/generated/torch.Tensor.add_.html

Tensor Basics

Common Mathematatic operations

- Subtraction

```
>>> x = torch.tensor([1, 2, 3])
>>> y = torch.tensor([4, 5, 6])
>>> z = x - y
```

- Inplace subtraction

```
>>> x = torch.tensor([1, 2, 3])
>>> y = 2
>>> x.sub_(y)
```

References:

https://pytorch.org/docs/stable/generated/torch.sub.html

https://pytorch.org/docs/stable/generated/torch.Tensor.sub_.html

Tensor Basics

Common Mathematatic operations

- Multiplication

Element wise multiplication if both are tensor and of same shape

```
>>> x = torch.tensor([1, 2, 3])
>>> y = torch.tensor([4, 5, 6])
>>> z = torch.mul(x, y) # can be written as z = x * y also
```

Scalar multiplication of elements from first array with integer provided as second element

```
>>> x = torch.tensor([1, 2, 3])
>>> y = 2
>>> z = torch.mul(x, y) # can be written as z = x * y also
```

References:

https://pytorch.org/docs/stable/generated/torch.mul.html

Tensor Basics

Common Mathematatic operations

- Division

Element wise division if both are tensor and of same shape

```
>>> x = torch.tensor([1, 2, 3])
>>> y = torch.tensor([4, 5, 6])
>>> z = torch.div(x, y) # can be written as z = x / y also
```

Scalar division of elements from first array with integer provided as second element

```
>>> x = torch.tensor([1, 2, 3])
>>> y = 2
>>> z = torch.div(x, y) # can be written as z = x / y also
```

References:

https://pytorch.org/docs/stable/generated/torch.div.html#torch.div

Tensor Basics

Common Mathematic operations

- Power

Element wise power of tensor

```
>>> x = torch.tensor([1, 2, 3])
>>> z = x.pow(2)
>>> z = x**2
```

- Dot

Computes the dot product of two 1D tensors.

```
>>> x = torch.tensor([2, 3])
>>> y = torch.tensor([2, 1])
>>> z = torch.dot(x, y)
```

References:

https://pytorch.org/docs/stable/generated/torch.pow.html

https://pytorch.org/docs/stable/generated/torch.dot.html

Tensor Basics

Common Mathematic operations

- Matrix multiplication

```
>>> x = torch.randn(2, 3)
>>> y = torch.randn(3, 5)
>>> z = torch.matmul(x, y)
```

- Batch matrix multiplication

```
>>> x = torch.randn(10, 2, 3)
>>> y = torch.randn(10, 3, 5)
>>> z = torch.matmul(x, y)
```

References:

https://pytorch.org/docs/stable/generated/torch.matmul.html#torch.matmul

Tensor Basics

Useful Mathematic operations

- Sum operation

```
>>> x = torch.tensor([[1, 2, 3], [4, 5, 6]])
>>> z = torch.sum(x, dim=0) # [5, 7, 9]
>>> z = torch.sum(x, dim=1) # [6, 15]
>>> z = torch.sum(x, dim=(0,1)) # 21
```

- Mean operation

```
>>> x = torch.tensor([[1, 2, 3], [4, 5, 6]], dtype=torch.float32)
>>> z = torch.mean(x, dim=0) # [2.5000, 3.5000, 4.5000]
>>> z = torch.mean(x, dim=1) # [2., 5.]
>>> z = torch.mean(x, dim=(0,1)) # 3.5
```

References:

https://pytorch.org/docs/stable/generated/torch.sum.html

https://pytorch.org/docs/stable/generated/torch.mean.html

Tensor Basics

Useful Mathematic operations

- Min operation

```
>>> x = torch.tensor([[1, 2, 3], [4, 5, 6]])
>>> z_v, z_i = torch.min(x, dim=0) # [1, 2, 3]
>>> z_v, z_i = torch.min(x, dim=1) # [1, 4]
>>> z = torch.min(x) # 1
```

- Max operation

```
>>> x = torch.tensor([[1, 2, 3], [4, 5, 6]])
>>> z_v, z_i = torch.max(x, dim=0) # [4, 5, 6]
>>> z_v, z_i = torch.max(x, dim=1) # [3, 6]
>>> z = torch.max(x) # 6
```

References:

https://pytorch.org/docs/stable/generated/torch.min.html

https://pytorch.org/docs/stable/generated/torch.max.html

Tensor Basics

Useful Mathematic operations

Clamp operation :

Clamps all elements in input into the range [min, max].

```
>>> min = -128
>>> max = 365
>>> x = (max-min)*torch.rand((2, 5)) + min
>>> print(x.min(), x.max())

>>> z = torch.clamp(x, min=0, max=255)
>>> print(z.min(), z.max())
```

References:

https://pytorch.org/docs/stable/generated/torch.clamp.html

Tensor Basics

Indexing

```
>>> x = torch.rand((10, 64))
>>> print(x[0].shape)
>>> print(x[:, 0].shape)

>>> x = torch.tensor([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
>>> z = x[[2, 5, 8]]
>>> print(z)

>>> x = torch.tensor([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
>>> z = x[(x<3) | (x>8)]
>>> print(z)
```

Tensor Basics

Reshaping

```
>>> x = torch.tensor([1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> print(x.size())
>>> z = x.view(3, 3)
>>> print(z.size())
>>> z = x.view(3, -1) # if other dimension is not known
>>> print(z.size())
```

References:

https://pytorch.org/docs/stable/generated/torch.Tensor.view.html

Tensor Basics

Transpose

```
>>> x = torch.rand(size=(3,3))
>>> print(x)
>>> z = x.t()
>>> print(z)
```

Flatten

```
>>> x = torch.rand(size=(3,3))
>>> z = torch.flatten(x)
>>> print(z)
or
>>> z = x.view(-1)
>>> print(z)
```

References:

https://pytorch.org/docs/stable/generated/torch.t.html

https://pytorch.org/docs/stable/generated/torch.flatten.html

Tensor Basics

Concat operation

```
>>> x = torch.rand(size=(10, 3, 128, 128))
>>> y = torch.rand(size=(10, 3, 128, 128))
>>> print(x.size())
>>> print(y.size())

>>> z = torch.cat([x, y], dim=0)
>>> print(z.size())

>>> z = torch.cat([x, y], dim=1)
>>> print(z.size())
```

References:

https://pytorch.org/docs/stable/generated/torch.cat.html

Tensor Basics

Dimension switch

```
>>> x = torch.rand(size=(10, 3, 128, 128))
>>> z = x.permute(0, 2, 3, 1)
>>> print(z.size())
```

Adding or Removing extra dimension

```
>>> x = torch.rand(size=(3, 128, 128))
>>> z = x.unsqueeze(0)
>>> print(z.size())
>>> y = z.squeeze(0)
>>> print(y.size())
```

References:

https://pytorch.org/docs/stable/generated/torch.permute.html

https://pytorch.org/docs/stable/generated/torch.squeeze.html

https://pytorch.org/docs/stable/generated/torch.unsqueeze.html

Perceptron Training for LINE problem

```
>>> # imports
>>> import torch
>>> import torch.nn as nn
>>> import torch.nn.functional as F
>>> import torch.optim as optim

>>> # line equation: y = w*x + c
>>> w = 3
>>> c = 5
>>> X = torch.FloatTensor([[0], [1], [2], [3], [4], [5], [6], [7], [8], [9],
    [10], [11]]).to(device)
>>> Y = torch.FloatTensor([[w*x+c] for x in range(12)]).to(device)
>>> # Y = torch.FloatTensor([[5], [8], [11], [14], [17], [20], [23], [26], [29],
    [32], [35], [38]]).to(device) # 3x+5
```

Perceptron Training for LINE problem

Creation of model

```
>>> # Perceptron model
>>> model = nn.Sequential(
>>>      nn.Linear(1, 1, bias=True),
>>> ).to(device)
```

Observe prior weights and biases initialized:

```
>>> print("Starting weights: {}".format(model[0].weight))
>>> print("Starting bias: {}".format(model[0].bias))
```

Loss and optimizer

```
>>> criterion = nn.MSELoss()
>>> optimizer = optim.SGD(model.parameters(), lr=0.01)
```

Perceptron Training for LINE problem

Training loop

```
>>> for step in range(5000):
>>>     pred = model(X)
>>>     loss = criterion(pred, Y)
>>>     optimizer.zero_grad()
>>>     loss.backward()
>>>     optimizer.step()
>>>     if step % 200==0:
>>>         print('step:', step, " loss:", loss.item())
```

Observe learned weights and biases initialized:

```
>>> print("Learned weights: {}".format(model[0].weight))
>>> print("Learned bias: {}".format(model[0].bias))
```

For Logical OR, Logical AND, and Logical XOR Training using perceptron code, Please visit the git link specified in second slide.

The Ignitarium logo represents a stylized Delta - the classical symbol for fire. The Delta logo is created from the amalgamation of smaller deltas signifying the stages of transition from spark to ember to flame to fire.

THANK YOU

email: info@ignitarium.com

Ignitarium Technology Solutions

Bangalore, India
#2615, 3rd Floor,
27th Main Road, Sector 1,
HSR Layout,
Bangalore – 560102
Phone: +91-80-42054217

Kochi, India
Office No. 1A-1,
Jyothirmaya IT Building P.O,
Infopark Phase 2, SEZ, Kochi,
Kerala 682303
Phone: +91-484-4876089

San Jose, CA
2570 N. First Street,
Suite 200 San Jose,
CA 95131
Phone: +1 512 640 3488

Austin, TX
555 Round Rock West Drive,
Suite E217,
Round Rock, TX 78681
Phone: +1-512-669-9214

Yokohama, Japan
7-25-22 Okamura, Isogo-ku,
Yokohama 235-0021, Japan
Phone: +81-901-707-4661
Tel/Fax: +81-45-350-1757