PyTorch Tutorial

Ahmad Diab
ahd23@pitt.edu
Mingda Zhang

Roadmap

- PyTorch - general intro
- PyTorch vs Numpy
- PyTorch vs TensorFlow
- Power of GPU (cuda)
- Automatic Differentiation
- Neural Network Models
  - Linear Regression
  - Multi-Layer Perceptrons (MLP)
  - Convolutional Neural Nets (CNN)
  - Off-the-shelf models
- TensorBoards

General Intro

- What is PyTorch?
  - Free, open-source Machine Learning library.
  - Applications in Computer Vision and Natural Language Processing.
  - Development led by Facebook AI Research.
- What makes it special?
  - Array computations with GPU compatibility.
  - Deep Learning Networks with Automatic Differentiation.
- Quick facts:
  - Python and C++ interfaces.
  - Core of many DL softwares such as Autopilot (Tesla), Pyro (Uber), Transformers (HuggingFace).
  - It is called Torch (not PyTorch) when used/imported, was born from Lua Torch package.

PyTorch vs. Numpy

- PyTorch introduced a new class called Tensor.
- Tensor is an multi dimensional array.
- Numpy also uses multi dimensional arrays supported by fast-implemented methods.
- Where is the catch?
    - Tensors can run on GPUs, utilizing powerful parallel processing.
    - Numpy arrays can only use CPUs.
- Good news! They are interchangable.

PyTorch is not alone!

K Keras

PYTORCH

Caffe

TensorFlow

mxnet

PyTorch vs. TensorFlow

- Popularity: PyTorch is the most common library in academics (based on paper implementations), where TensorFlow is still dominating industry (based on job listing). (source: https://www.stateof.ai/)
- Easy to learn: Subjective.
- Computation Graphs: PyTorch utilizes a dynamic perspective where TensorFlow uses a static view.
- Debugging: Harder for TensorFlow, needs external tools.

GPU and cuda

- Graphics Processing Unit (GPU) performs operations in parallel/fast way.
- CPUs use MIMD, GPUs use SIMD
    - A = B * C (think about types and sizes)
- How to use it?
- CUDA = API.
- A software layer that gives access to GPU virtual instruction set.
- Created by NVIDIA, 2007.

```
a = torch.tensor(5)
b = torch.tensor(5).cuda()
a = a ** 2
b = b ** 2
print(a)
print(b)
```

```
tensor(25)
tensor(25, device='cuda:0')
```

Automatic Differentiation

- Simple Example: $$X = [1,2,3]$$ $$Y = [4,5,6]$$ $$Z = 5X + 2Y$$ $$m = sum(Z) = z_0 + z_1 + z_2$$
- Calculate the derivative using the chain rule.

$$\frac{\partial m}{\partial x_i} = \frac{\partial m}{\partial z_i} \cdot \frac{\partial z_i}{\partial x_i} = 1 \times 5 = 5$$

$$\frac{\partial m}{\partial y_i} = \frac{\partial m}{\partial z_i} \cdot \frac{\partial z_i}{\partial y_i} = 1 \times 5 = 2$$

$$\frac{\partial m}{\partial X} = [5,5,5]$$

$$\frac{\partial m}{\partial Y} = [2,2,2]$$

Automatic Differentiation - more example

- What changes if:
  - X=[1,2,3,4] and Y=[5,6,7,8]
  - Z = 5X^2 + 2Y^3

$$\frac{\partial m}{\partial x_i} = \frac{\partial m}{\partial z_i} \cdot \frac{\partial z_i}{\partial x_i} =$$

$$\frac{\partial m}{\partial y_i} = \frac{\partial m}{\partial z_i} \cdot \frac{\partial z_i}{\partial y_i} =$$

$$\frac{\partial m}{\partial X} =$$

$$\frac{\partial m}{\partial Y} =$$

Automatic Differentiation - PyTorch

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

m = z.sum()
m.backward()
print(x.grad, '\n', y.grad)
```

```
tensor([5., 5., 5.])
 tensor([2., 2., 2.])
```

```
x = torch.tensor([1., 2., 3., 4.], requires_grad=True)
y = torch.tensor([5., 6., 7., 8.], requires_grad=True)
z = 5 * x + 2 * y

m = z.sum()
m.backward()
print(x.grad, '\n', y.grad)
```

```
tensor([5., 5., 5., 5.])
 tensor([2., 2., 2., 2.])
```

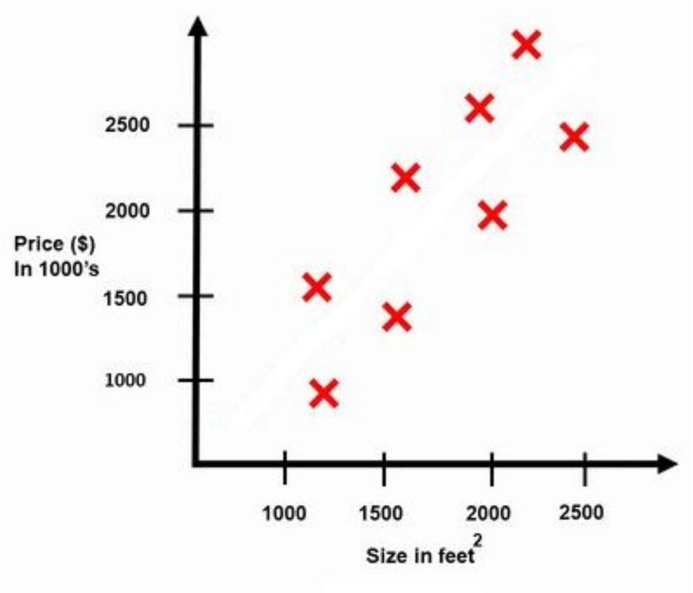
```
x = torch.tensor([1., 2., 3.], requires_grad=True)
y = torch.tensor([4., 5., 6.], requires_grad=True)
z = 5 * x ** 2 + 2 * y ** 3

m = z.sum()
m.backward()
print(x.grad, '\n', y.grad)
```

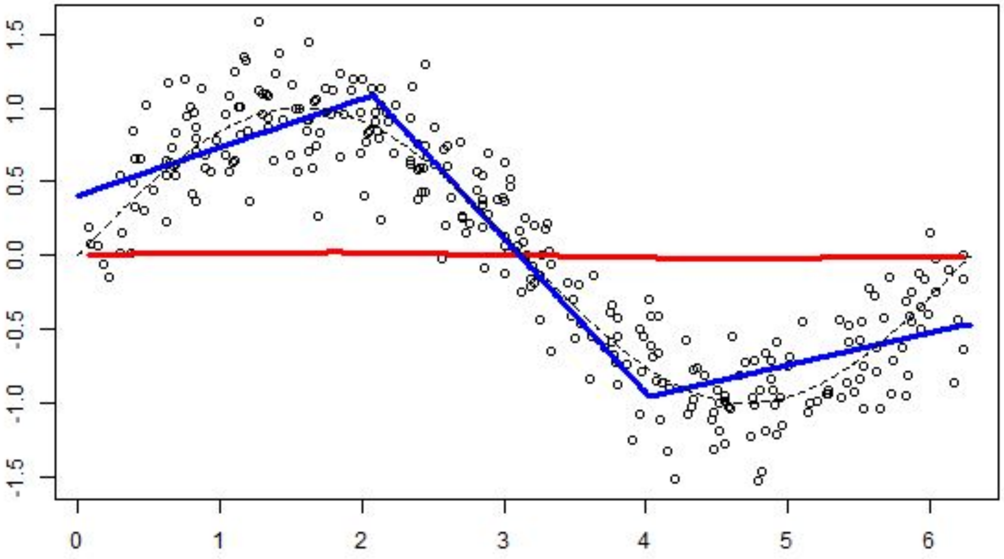
```
tensor([10., 20., 30.])
 tensor([ 96., 150., 216.])
```

Linear Regression

- Entry-level technique in Machine Learning Realm.
- Example: *House Prices*

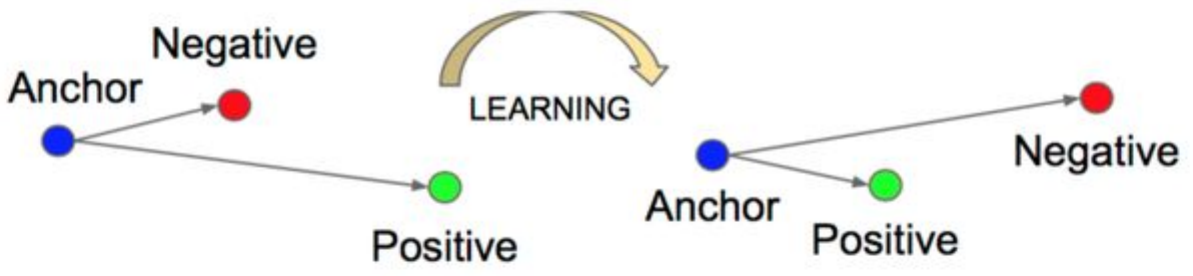


Linear Regression - in action



Loss Function

- Loss is critical, it serves as the supervision for training a neural network.
- Different types of loss functions, such as:
  - Mean Squared Error Loss (Regression)
  - Cross-Entropy Loss (Classification)
  - Triplet Loss (Representation Learning)



- All implemented off-the-shelf with unified API, which is easy to use.

Optimizers

- Lower the Loss function by updating model parameters for future iterations based on loss value of current iteration.
- *torch.optim* is a package implementing various optimization algorithms.
- It supports common methods, and provides interface for future integrations.

Example:

```
for input, target in dataset:
    optimizer.zero_grad()
    output = model(input)
    loss = loss_fn(output, target)
    loss.backward()
    optimizer.step()
```

Linear Regression - Module

```python
import torch.nn as nn

class ManualLinearRegression(nn.Module):
  def __init__(self):
    super().__init__()

    self.weight = nn.Parameter(torch.randn(1, requires_grad=True))
    self.bias = nn.Parameter(torch.randn(1, requires_grad=True))

  def forward(self, x):
    return self.weight * x + self.bias
```

```python
# Build a Model
model = ManualLinearRegression()
loss_fn = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

Linear Regression - manual implementation

```
weight = torch.rand(1, requires_grad=True)   # Assuming y = x * w + b
bias = torch.rand(1, requires_grad=True)

learning_rate = 0.1 # Hyperparameter

total_steps = tqdm_notebook(range(10000))

for step in total_steps:
  prediction = weight * data + bias
  loss = ((prediction - label) ** 2).mean()   # Mean squared error
  if (step + 1) % 100 == 0:
    total_steps.set_description("Loss: %.4f" % loss.detach().item())

  loss.backward()
  # plt.plot(x, weight * x + bias)

  with torch.no_grad():   # Stops tracking variable
    weight -= learning_rate * weight.grad
    bias -= learning_rate * bias.grad

    weight.grad.zero_()   # Avoid gradients being accumulated
    bias.grad.zero_()
```

PyTorch built-in module: *torch.nn*

torch.nn

Parameters

+ Containers
+ Convolution layers
+ Pooling layers
+ Padding layers
+ Non-linear activations (weighted sum, nonline:
+ Non-linear activations (other)
+ Normalization layers
+ Recurrent layers
+ Transformer layers
+ Linear layers
+ Dropout layers
+ Sparse layers
+ Distance functions
+ Loss functions
+ Vision layers
+ DataParallel layers (multi-GPU, distributed)
+ Utilities

Quantized Functions

– Linear layers

Identity

Linear

Bilinear

Linear

CLASS `torch.nn.Linear(in_features, out_features, bias=True)` [SOURCE]

Applies a linear transformation to the incoming data: $y = xA^T + b$

Parameters

- **in_features** – size of each input sample
- **out_features** – size of each output sample
- **bias** – If set to `False`, the layer will not learn an additive bias. Default: `True`

Shape:

- Input: $(N, *, H_{in})$ where $*$ means any number of additional dimensions and $H_{in} = \text{in\_features}$
- Output: $(N, *, H_{out})$ where all but the last dimension are the same shape as the input and $H_{out} = \text{out\_features}$.

Variables

- **~Linear.weight** – the learnable weights of the module of shape $(\text{out\_features}, \text{in\_features})$. The values are initialized from $\mathcal{U}(-\sqrt{k}, \sqrt{k})$, where $k = \frac{1}{\text{in\_features}}$
- **~Linear.bias** – the learnable bias of the module of shape (out_features). If `bias` is `True`, the values are initialized from $\mathcal{U}(-\sqrt{k}, \sqrt{k})$ where $k = \frac{1}{\text{in\_features}}$

Examples:

```
>>> m = nn.Linear(20, 30)
>>> input = torch.randn(128, 20)
>>> output = m(input)
>>> print(output.size())
torch.Size([128, 30])
```

MNIST (dataset)

- Dataset of handwritten digits.
- 60k training images, 10k testing images.
- Why?

  Good for learning ML/DL techniques on real-world data *while* spending minimal efforts on preprocessing and formatting.

- From *torchvision*; package consists of popular datasets, models, and image transformations for computer vision.

Modularized Networks: MLP and CNN

- All the modules can be stacked together arbitrarily, which makes it convenient to build complicated network architecture using simpler ones.
  - Here the Linear module is stacked to build multi-layer perceptrons, together with ReLU (a non-linear activation module).

```
# Define our model (3-layer MLP)
class MultilayerPerceptron(nn.Module):
  def __init__(self, input_size, hidden_size, num_classes):
    super().__init__()
    self.fc1 = nn.Linear(input_size, hidden_size)
    self.relu = nn.ReLU()
    self.fc2 = nn.Linear(hidden_size, num_classes)

  def forward(self, x):
    out = self.fc1(x.reshape(-1, 28 * 28))
    out = self.relu(out)
    out = self.fc2(out)
    return out
```

Modularized Networks: MLP and CNN

```python
class ConvNet(nn.Module):
  def __init__(self, num_classes=10):
    super().__init__()

    # We can further simplify the code if a data will be sequentially processed
    # by multiple modules: just use `nn.Sqeuential` to link them all.
    self.layer1 = nn.Sequential(
        nn.Conv2d(1, 16, kernel_size=5, stride=1, padding=2),
        nn.BatchNorm2d(16),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=2))

    self.layer2 = nn.Sequential(
        nn.Conv2d(16, 32, kernel_size=5, stride=1, padding=2),
        nn.BatchNorm2d(32),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=2))

    self.fc = nn.Linear(7*7*32, num_classes)

  def forward(self, x):
    out = self.layer1(x)
    out = self.layer2(out)
    out = out.reshape(out.size(0), -1)
    out = self.fc(out)

    return out
```

Off-the-shelf Model and Checkpoints - example

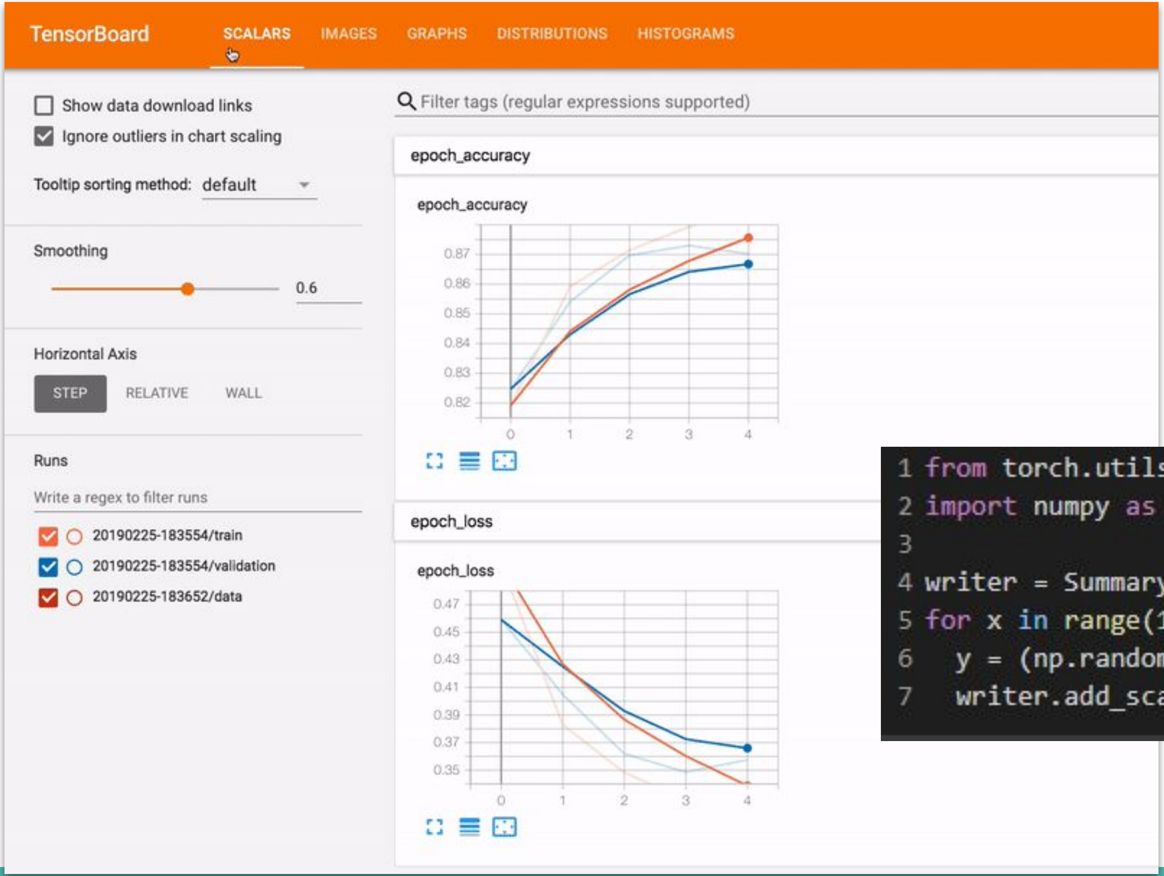
```python
import torchvision
import torchvision.models as models

print(models.mobilenet_v2())
```

```
MobileNetV2(
  (features): Sequential(
    (0): ConvBNReLU(
      (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU6(inplace=True)
    )
    (1): InvertedResidual(
      (conv): Sequential(
        (0): ConvBNReLU(
          (0): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32, bias=False)
          (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (1): Conv2d(32, 16, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (2): InvertedResidual(
      (conv): Sequential(
        (0): ConvBNReLU(
          (0): Conv2d(16, 96, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (1): ConvBNReLU(
          (0): Conv2d(96, 96, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), groups=96, bias=False)
          (1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          (2): ReLU6(inplace=True)
        )
        (2): Conv2d(96, 24, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (3): BatchNorm2d(24, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (3): InvertedResidual(
      (conv): Sequential(
```

MobileNet is a light-weight network designed for mobile devices, which has 17 *InvertedResidual* Blocks each with multiple Convolution and Normalization layers.

TensorBoards: Experiment Visualization.



Tracks some key metrics during model training such as loss, accuracy and histogram of gradients, which is super useful for development.

```python
from torch.utils.tensorboard import SummaryWriter
import numpy as np

writer = SummaryWriter()
for x in range(100):
  y = (np.random.random() + x) / 100
  writer.add_scalar("Random Values", y, x)
```

PyTorch Resources

- Step-by-step live demo:

  https://colab.research.google.com/drive/1L2US3PF6j2kzoOemkeedAb_E_hbYjVMP

- Official Documentation:

  https://pytorch.org/docs/stable/index.html

- PyTorch Tutorial from its developers:

  https://pytorch.org/tutorials/

- Plenty of text/visual resources.