Introduction to PyTorch
Deep learning frameworks

- Modern tools make it easy to implement neural networks
- Often used components
  - Linear, convolution, recurrent layers, transformers, etc.
PyTorch

- Fast tensor computation (like numpy) with strong **GPU support**
- Deep learning framework with maximum **flexibility** and speed
- Dynamic graphs and **automatic differentiation**
Outlines

- Basic concepts
- Write a model
- Classification of MNIST dataset
Basic Concepts

torch.Tensor - similar to numpy.array, auto-differentiation

autograd.Function - operate on Variables, implements forward and backward pass

nn.Module - contain Parameters and define functions on input Variables (base-class for neural network)
Basic Concepts

```python
import torch
x = torch.rand(10, 5)
y = torch.rand(10, 5)
z = x * y
```
Basic Concepts

```python
import torch
x = torch.rand(10, 5)
y = torch.rand(10, 5)
z = x * y
```

```python
import torch
x = torch.ones(5, 5)
b = x.numpy()
```
import torch
x = torch.rand(10, 5)
y = torch.rand(10, 5)
z = x * y

import torch
x = torch.ones(5, 5)
b = x.numpy()
a = np.ones(5)
x = torch.from_numpy(a)
import torch
x = torch.rand(100, 100)
x = x.cuda()
Gradients

- Used for automatic differentiation

```python
import torch
w1 = torch.tensor([[1.0, -3.0],
                   [-2.0, 4.0]],
                   requires_grad=True)
out = w1 * w1
loss = out.sum()
print(w1.grad)  # None
loss.backward() # execute backward-pass of loss
print(w1.grad)  # d(loss)/d(w1) = tensor([[ 2., -6.],
                                         [-4.,  8.]])
print(loss.grad_fn) # <SumBackward0 object at 0x0000020CA7892BC8>
```
Network definition

class TwoLayerNet(torch.nn.Module):
    def __init__(self, D_in, H, D_out):
        super(TwoLayerNet, self).__init__()
        self.linear1 = torch.nn.Linear(D_in, H)
        self.linear2 = torch.nn.Linear(H, D_out)
        self.relu = torch.nn.ReLU()

    def forward(self, x):
        h_relu = self.linear1(x)
        h_relu = self.relu(h_relu)
        y_pred = self.linear2(h_relu)
        return y_pred
Optimize the Network

model = TwoLayerNet(3, 2, 1)
loss_fn = torch.nn.MSELoss(size_average=False)
optimizer = torch.optim.SGD(model.parameters(), lr=1e-4)

for t in range(500):
    # x = ... , y = ...
    # Zero gradients
    optimizer.zero_grad()
    # Forward pass: Compute predicted y by passing x to the model
    y_pred = model(x)
    # calculate loss
    loss = loss_fn(y_pred, y)
    # perform a backward pass, and update the weights.
    loss.backward()
    optimizer.step()
Classifying Handwritten Digits

Input Image

Conv 5x5, 1/10

MaxPool 2x2 + RELU

Conv 5x5, 10/20

MaxPool 2x2 + RELU

Conv 5x5, 10/20

Dropout , p = 0.5

Linear, 320/50

Linear, 50/10

Softmax
Classifying Handwritten Digits

Sample code in Docker container:

/home/freicar/freicar_deps/pytorch_tutorial/train_mnist.py
General tips

- **Know your data**: Visualize the input to your model (images) and the predictions of your model + the ground truth data (labels) for your model. Make sure that they all make sense.
- Don't forget to **normalize your input data**.
- Don't forget `optimizer.zero_grad()`
- Train on **small subset** of dataset and see if you can overfit your model to it. Then, train on full training split and evaluate on evaluation split.
Further reading

- PyTorch 60 min blitz tutorial
  https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html
Thank you!