ML Theory Week 1

BY JACK YANSONG LI
University of Illinois Chicago
Email: yli340@uic.edu

1 A tutorial on PyTorch

```
>>> import torch
>>> import torch.nn as nn
>>> import torch.optim as optim
>>>
```

The goal is to approximate the XOR gate defined as follows:

```
>>> def xor(a,b):
        return a ^ b
>>> inputs = [(0, 0), (0, 1), (1, 0), (1, 1)]
    outputs = [xor(x[0], x[1]) for x in inputs]
>>>
```

The XOR gate is approximated by a single layer neural net defined as:

```
>>> class XORNet(nn.Module):
        def __init__(self):
            super(XORNet, self).__init__()
            self.layer1 = nn.Linear(2,4)
            self.layer2 = nn.Linear(4,1)
        def forward(self, x):
            x = torch.relu(self.layer1(x))
            x = self.layer2(x)
            return x
>>> model = XORNet()
>>> # Define the loss function and optimizer
    loss_func = nn.MSELoss()  # Mean Squared Error Loss
    optimizer = optim.SGD(model.parameters(), lr=0.01)  # Stochastic Gradient Descent
>>> def weights_init(model):
        for m in model.modules():
            if isinstance(m, nn.Linear):
                # initialize the weight tensor, here we use a normal distribution
                m.weight.data.normal_(0, 1)

    weights_init(model)
>>>
```

Convert inputs and outputs data to torch tensors for training

```
>>> X = torch.tensor(inputs, dtype=torch.float32)
>>> Y = torch.tensor([y for y in outputs], dtype=torch.float32).view(-1,1)
    print("Inputs:", X)
    print("Outputs:", Y)
    #print(model(X))
```

```
Inputs: tensor([[0., 0.],
        [0., 1.],
        [1., 0.],
        [1., 1.]])
Outputs: tensor([[0.],
        [1.],
        [1.],
        [0.]])
```

```
>>> epochs = 2000  # Increased epochs
    for epoch in range(epochs):
        Y_pred = model(X)
        loss = loss_func(Y_pred, Y)
        if epoch % 500 == 0:
            print(f'Epoch␣{epoch}␣Loss:␣{loss.item()}')

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
  Epoch 0 Loss: 3.7070677280426025
  Epoch 500 Loss: 0.014563385397195816
  Epoch 1000 Loss: 0.001690817647613585
  Epoch 1500 Loss: 0.00018612013082019985
>>> # Test the model
    with torch.no_grad():
        test_pred = model(X)
        print("Predicted␣outputs:")
        print(test_pred.round())
  Predicted outputs:
  tensor([[0.],
          [1.],
          [1.],
          [0.]])
>>>
```

However, if we only use a single layer network defined below, it cannot approximate the `XOR` gate:

```
>>> class XORNetSingleLayer(nn.Module):
        def __init__(self):
            super(XORNetSingleLayer, self).__init__()
            self.layer1 = nn.Linear(2,1)
        def forward(self, x):
            x = torch.relu(self.layer1(x))
            return x
>>> model_single_layer = XORNetSingleLayer()
>>> # Define the loss function and optimizer
    loss_func = nn.MSELoss()  # Mean Squared Error Loss
    optimizer = optim.SGD(model.parameters(), lr=0.01)  # Stochastic Gradient Descent
>>> weights_init(model)
>>> epochs = 2000  # Increased epochs
    for epoch in range(epochs):
        Y_pred = model_single_layer(X)
        loss = loss_func(Y_pred, Y)
        if epoch % 500 == 0:
            print(f'Epoch␣{epoch}␣Loss:␣{loss.item()}')

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
  Epoch 0 Loss: 0.2725851237773895
  Epoch 500 Loss: 0.2725851237773895
  Epoch 1000 Loss: 0.2725851237773895
  Epoch 1500 Loss: 0.2725851237773895
>>> # Test the model
    with torch.no_grad():
        test_pred = model_single_layer(X)
        print("Predicted␣outputs:")
        print(test_pred.round())
  Predicted outputs:
  tensor([[0.],
          [0.],
          [1.],
```

```
        [0.]])
>>>
```

2 Homework

Problem 1. Formally state and prove that a single layer neural network (also known as perceptron) cannot approximate the `XOR` gate. Verify your result empirically. *Hint: derive a lower bound of the approximation error. Verify your bound by drawing the approximation error w.r.t. number of iterations.*

Problem 2. Formally state and prove that a two-layer neural network with more than 2 neurons in the hidden layer can approximate the `XOR` gate. *Hint: Manually construct a neural network that gives the same outputs as* `XOR` *gate and computes its parameters by hand.*