Chem 277B Spring 2024 Tutorial 4

Outline

- Installation & Introduction
- PyTorch Tensors and auto grads
- Building up neural network
- Optimizers

Installation & Introduction

- Official Web site
- Installation

PyTorch is an open-source machine learning library widely used for deep learning applications. Developed by Facebook's AI Research lab (FAIR), it provides a flexible and intuitive framework for building and training neural networks. PyTorch is known for its ease of use, computational efficiency, and dynamic computational graph, making it a favorite among researchers and developers for both academic and industrial applications.

Key Features of PyTorch

- **Dynamic Computational Graph**: PyTorch uses a dynamic computation graph (also known as a define-by-run paradigm), meaning the graph is built on the fly as operations are performed. This makes it more intuitive and flexible, allowing for easy changes and debugging.
- **Eager Execution**: Operations in PyTorch are executed eagerly, meaning they are computed immediately without waiting for a compiled graph of operations. This allows for more interactive and dynamic development.
- **Pythonic Nature**: PyTorch is deeply integrated with Python, making it easy to use and familiar to those with Python experience. It leverages Python's features and libraries, allowing for seamless integration with the Python data science stack (e.g., NumPy, SciPy, Pandas).
- **Extensive Library Support**: PyTorch provides a wide range of libraries and tools for various tasks in deep learning, including computer vision (TorchVision), natural

language processing (TorchText), and more. This ecosystem supports a vast array of algorithms, pre-trained models, and datasets to facilitate development and experimentation.
- **GPU Acceleration**: It supports CUDA, enabling it to leverage Nvidia GPUs for accelerated tensor computations. This makes training deep neural networks significantly faster compared to CPU-based training.
- **Community and Support**: PyTorch has a large and active community, contributing to a growing ecosystem of tools, libraries, and resources. It also enjoys robust support from major tech companies, ensuring continuous development and improvement.

Tensors

Tensors are data structure in PyTorch to manipulate data. It is very similar to numpy.ndarray, but with support for automatic differentiation and hardware acceleration (Nvidia GPU, Apple silicon)

In [ ]:
```
import torch
```

In [ ]:
```
a = torch.tensor([[1, 2], [3, 4]], dtype=torch.float)
print(type(a))
a
```

```
<class 'torch.Tensor'>
```

Out[ ]:
```
tensor([[1., 2.],
        [3., 4.]])
```

Bridge with NumPy

In [ ]:
```
import numpy as np

arr = np.array([[1., 2.], [3., 4.]])
arr_torch = torch.from_numpy(arr)
arr_torch
```

Out[ ]:
```
tensor([[1., 2.],
        [3., 4.]], dtype=torch.float64)
```

In [ ]:
```
# detach() stops a tensor from tracking history in automatic differentiatior
arr_np = arr_torch.detach().numpy()
```

Generate random numbers

In [ ]:
```
# normal distribution
torch.randn(4, 4)
```

```
Out[ ]: tensor([[ 1.5061,  1.5521, -1.0470,  0.9524],
                [ 0.1724, -0.8170, -0.4362, -1.7570],
                [-0.9271,  0.5948,  0.5426,  0.6211],
                [ 2.5317, -0.0726,  1.2755, -0.9024]])
```

```
In [ ]: # uniform distribution
        torch.rand(4, 4)
```

```
Out[ ]: tensor([[0.4538, 0.1356, 0.3053, 0.3966],
                [0.7942, 0.2810, 0.5078, 0.0802],
                [0.2853, 0.4896, 0.2016, 0.0923],
                [0.2763, 0.8430, 0.4971, 0.7215]])
```

Others

```
In [ ]: # arange
        torch.arange(5)
```

```
Out[ ]: tensor([0, 1, 2, 3, 4])
```

```
In [ ]: # linspace
        torch.linspace(-4, 4, 10)
```

```
Out[ ]: tensor([-4.0000, -3.1111, -2.2222, -1.3333, -0.4444,  0.4444,  1.3333,  2.2
        222,
                 3.1111,  4.0000])
```

```
In [ ]: # ones & zeros
        torch.ones(6)
```

```
Out[ ]: tensor([1., 1., 1., 1., 1., 1.])
```

```
In [ ]: torch.zeros(6)
```

```
Out[ ]: tensor([0., 0., 0., 0., 0., 0.])
```

Attributes of tensors

```
In [ ]: tensor = torch.rand(3,4)

        # shape, dtype, device
        print(tensor.shape)
        print(tensor.dtype)
        print(tensor.device)
```

```
torch.Size([3, 4])
torch.float32
cpu
```

Single-element tensor can use `.item()` method to get a Python float object

```
In [ ]: a = torch.tensor([4.])
        print(type(a.item()))
```

```
<class 'float'>
```

PyTorch can work on different hardwares

In [ ]:
```
device = (
    "cuda"
    if torch.cuda.is_available()
    else "mps"
    if torch.backends.mps.is_available()
    else "cpu"
)

# send the tensor to device
tensor_device = tensor.to(device)

# send the tensor back to cpu
tensor_cpu = tensor.cpu()
```

Autograd

In [ ]:
```
x = torch.tensor([[1, 2], [3, 4]], dtype=torch.float, requires_grad=True)
y = torch.sum(x ** 2)
# backward
y.backward()
# get grad
x.grad
```

Out[ ]:
```
tensor([[2., 4.],
        [6., 8.]])
```

Build Neural Network with PyTorch

In [ ]:
```
import torch.nn as nn
```

Activation Functions

In [ ]:
```
tensor = 5 * (torch.rand(3, 2) * 2 - 1)
print(tensor)

# ReLU
relu = nn.ReLU()
print("ReLU:", relu(tensor))

# Tanh
tanh = nn.Tanh()
print("Tanh:", tanh(tensor))

# Sigmoid
sigmoid = nn.Sigmoid()
print("Sigmoid:", sigmoid(tensor))

# Softmax
```

```
softmax = nn.Softmax(dim=1)
print('Softmax:', softmax(tensor))
```

```
tensor([[-4.4536, -3.6808],
        [ 4.2285,  0.9234],
        [-2.8912,  0.2049]])
ReLU: tensor([[0.0000, 0.0000],
        [4.2285, 0.9234],
        [0.0000, 0.2049]])
Tanh: tensor([[-0.9997, -0.9987],
        [ 0.9996,  0.7275],
        [-0.9939,  0.2021]])
Sigmoid: tensor([[0.0115, 0.0246],
        [0.9856, 0.7157],
        [0.0526, 0.5510]])
Softmax: tensor([[0.3159, 0.6841],
        [0.9646, 0.0354],
        [0.0433, 0.9567]])
```

Loss functions

In [ ]:

```
# mse
mse = nn.MSELoss()
a, b = torch.rand(5, 2), torch.rand(5, 2)
print(mse(a, b))

# cross-entropy
cross_entropy = nn.CrossEntropyLoss()
a = torch.rand(10, 2)
b = torch.randint(2, (10,))
print(cross_entropy(a, b))
```

```
tensor(0.0510)
tensor(0.8622)
```

Neural Network

In [ ]:

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        # create a net with one hidden layer
        # input_dim 13, hidden_dim 3, output_dim 3
        # use ReLU and softmax activation func
        self.layers = nn.Sequential(
            nn.Linear(13, 3),
            nn.ReLU(),
            nn.Linear(3, 3),
            nn.Softmax(dim=1)
        )

    def forward(self, X):
        return self.layers(X)
```

```python
model = Net()
model
```

Out[ ]:
```
Net(
  (layers): Sequential(
    (0): Linear(in_features=13, out_features=3, bias=True)
    (1): ReLU()
    (2): Linear(in_features=3, out_features=3, bias=True)
    (3): Softmax(dim=1)
  )
)
```

In [ ]:
```python
for name, param in model.named_parameters():
    print(f"Layer: {name} | Size: {param.size()} | Values : {param.data} \n"
```

```
Layer: layers.0.weight | Size: torch.Size([3, 13]) | Values : tensor([[ 1.49
70e-01,  1.9379e-01, -9.0148e-03, -1.7468e-01, -7.0514e-03,
          2.5876e-01, -1.6496e-01,  3.2200e-02,  2.7713e-01, -7.9044e-02,
          2.1192e-01, -1.1687e-01, -9.4473e-02],
        [-1.1988e-01,  2.6266e-01, -2.7326e-01,  3.7626e-02,  6.1010e-03,
          1.1791e-01,  1.8273e-01,  2.6510e-02,  2.3690e-01,  1.5921e-01,
         -1.0735e-01,  1.0431e-01, -1.3671e-01],
        [-8.7143e-02,  6.0707e-02,  2.5314e-01, -1.8690e-04, -1.8122e-01,
         -2.6005e-01,  8.0267e-03,  9.1560e-02,  3.6584e-02, -1.1889e-01,
          2.4024e-01, -7.0544e-03, -6.0306e-02]])

Layer: layers.0.bias | Size: torch.Size([3]) | Values : tensor([0.1372, 0.12
76, 0.1171])

Layer: layers.2.weight | Size: torch.Size([3, 3]) | Values : tensor([[ 0.297
3,  0.2046,  0.2851],
        [-0.1045, -0.3047, -0.0849],
        [ 0.4997,  0.4337,  0.5680]])

Layer: layers.2.bias | Size: torch.Size([3]) | Values : tensor([0.0160, 0.36
79, 0.2519])
```

In [ ]:
```python
X = torch.rand(3, 13)
y = model(X)
print(y)
```

```
tensor([[0.2821, 0.2517, 0.4662],
        [0.2887, 0.2284, 0.4829],
        [0.2880, 0.2503, 0.4617]], grad_fn=<SoftmaxBackward0>)
```

Optimization

In [ ]:
```python
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

df = pd.read_csv("wines.csv")
df.head()
```

Out[ ]:

	Alcohol %	Malic Acid	Ash	Alkalinity	Mg	Phenols	Flavanoids	Phenols.1	Proantho-cyanins	in
0	14.23	1.71	2.43	15.6	127	2.8	3.06	0.28	2.29	
1	13.24	2.59	2.87	21.0	118	2.8	2.69	0.39	1.82	
2	14.83	1.64	2.17	14.0	97	2.8	2.98	0.29	1.98	
3	14.12	1.48	2.32	16.8	95	2.2	2.43	0.26	1.57	
4	13.75	1.73	2.41	16.0	89	2.6	2.76	0.29	1.81	

In [ ]:

```
features = df.drop(['Start assignment', 'ranking'], axis=1).values
X = StandardScaler().fit_transform(features)
X = torch.tensor(X, dtype=torch.float32)
y = torch.tensor(df['ranking'].values - 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# define loss
loss_func = nn.CrossEntropyLoss()

# define optimizer
optimizer = torch.optim.Adam(model.parameters(), lr=1E-3)

epochs = 50
for _ in range(epochs):
    y_pred = model(X_train)
    loss = loss_func(y_pred, y_train)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    with torch.no_grad():
        test_loss = loss_func(model(X_test), y_test)
        print(test_loss.item())
```

```
1.1370505094528198
1.1359128952026367
1.1347821950912476
1.1336588859558105
1.1325392723083496
1.1313978433609009
1.1302647590637207
1.1291391849517822
1.1280276775360107
1.1269327402114868
1.1258574724197388
1.1247683763504028
1.123687744140625
1.1226146221160889
1.1215488910675049
1.120490550994873
1.119441032409668
1.1184029579162598
1.1173787117004395
1.1163653135299683
1.115361213684082
1.114367127418518
1.113383412361145
1.1123921871185303
1.111432433128357
1.1104927062988281
1.1095753908157349
1.108666181564331
1.1077618598937988
1.1068756580352783
1.1060177087783813
1.1051641702651978
1.1042935848236084
1.1034016609191895
1.102513313293457
1.1016294956207275
1.1007494926452637
1.09987211227417
1.0990054607391357
1.0981426239013672
1.0972955226898193
1.09645414352417
1.0956145524978638
1.0947521924972534
1.0938698053359985
1.0929876565933228
1.0921039581298828
1.0912169218063354
1.0903290510177612
1.0894485712051392
```

In [ ]: