tmd2_1

2023 年 10 月 30 日

1 FDU PRML 2023 Fall Assignment 2.1

Name: 田沐钊 Student ID: 21307140069

Deadline: 2023-10-30 23:59 Overall score weight: 30/100

In this semester, we are going to complete 3 assignments, each may contain **2-3 parts**. This is the first part of the second assignment, in which we will get to know about Pytorch. This assignment contains **little** explorations but **a bunch of** basic operations. So most of us will get full score in this assignment.

Installing torch:

`pip install torch`

[22]:
```
# Basic imports

import torch
```

Please follow the instructions below and write your torch implementation in the following:

1.1 1. Basic Operations of Tensors

[23]:
```
my_first_tensor =  torch.zeros(size=(3, 4), dtype=torch.float32,␣
 ↪requires_grad=True)

my_second_tensor = torch.randn(size=(3, 4), dtype=torch.float32,␣
 ↪requires_grad=True)

their_matrix_product = my_first_tensor @ torch.transpose(my_second_tensor, 0, 1)
```

```
some_meaningless_concatenation = torch.cat([my_first_tensor, my_second_tensor],␣
 ↪dim=0)

some_meaningless_stack = torch.stack([my_first_tensor]*5, dim=0)
# What is the shape of some_meaningless_stack? Can you imagine the geometric␣
 ↪interpretation of stacking 5 matrices of shape (3, 4) along the first␣
 ↪dimension?
```

1.2 2. A simple logistic regression

There are 4 core components in Pytorch training process: **model**, **loss function**, **optimizer** and **data loader**. In this part, we will implement a simple logistic regression model to illustrate them.

1.2.1 2.1 Model and Loss Function

[24]:
```
# Define a linear layer for logitstic regression

import torch.nn.functional as F

class Linear(torch.nn.Module):
        def __init__(self, input_dim, output_dim):
                super().__init__()
                self.linear = torch.nn.Linear(input_dim, output_dim)
                self.sigmoid = torch.nn.Sigmoid()


        def forward(self, x):
                output = self.linear(x)
                output = self.sigmoid(output)
                return output


def loss_function(y_pred, y):
    return F.binary_cross_entropy(y_pred, y)
```

1.2.2 Synthetic Data

In real world, we usually have to deal with large-scale datasets. However, in this assignment, we will use synthetic data to illustrate the training process. The synthetic data is generated by the following function:

[25]:
```
# Generate some random data for binary classification

num_samples = 100
num_features = 2

x_0 = torch.randn(num_samples, num_features) + torch.tensor([2.0, 2.0])
y_0 = torch.zeros(num_samples)

x_1 = torch.randn(num_samples, num_features) + torch.tensor([-2.0, -2.0])
y_1 = torch.ones(num_samples)

x = torch.cat([x_0, x_1], dim=0)
y = torch.cat([y_0, y_1], dim=0)
```

1.2.3 2.2 Dataloader

[26]:
```
# Define a dataset to feed into the model

class MyDataset(torch.utils.data.Dataset):
        def __init__(self, x, y):
                super().__init__()
                self.x = x
                self.y = y

        def __getitem__(self, index):
                return self.x[index], self.y[index]

        def __len__(self):
                return len(self.x)

dataset = MyDataset(x, y)
dataloder = torch.utils.data.DataLoader(dataset, batch_size=16, shuffle=True)
```

1.2.4 2.3 Optimizer

[27]:
```
import torch.optim as optim

my_model = Linear(num_features, 1)

optimizer = optim.SGD(my_model.parameters(), lr=0.01)
```

1.2.5 Putting all together

Since this is just a toy experiment, we do not need validation.

In the following code, we expect to see the training loss decreasing to 0.001 or lower.

[28]:
```
# Train the model
for epoch in range(1000):
        for batch_x, batch_y in dataloder:
                optimizer.zero_grad()
                pred = my_model(batch_x)
                batch_y = batch_y.view(-1, 1)
                loss = loss_function(pred, batch_y)
                loss.backward()
                optimizer.step()

        if epoch % 10 == 0:
                print('epoch: {}, loss: {}'.format(epoch, loss.item()))

# save the model

torch.save(my_model.state_dict(), 'my_model.pt')
```

```
epoch: 0, loss: 0.40014904737472534
epoch: 10, loss: 0.14905522763729095
epoch: 20, loss: 0.05595482140779495
epoch: 30, loss: 0.06289394944906235
epoch: 40, loss: 0.04849345609545708
epoch: 50, loss: 0.026742951944470406
epoch: 60, loss: 0.02393188327550888
epoch: 70, loss: 0.013506157323718071
```

```
epoch: 80, loss: 0.016785962507128716
epoch: 90, loss: 0.012103969231247902
epoch: 100, loss: 0.04488692805171013
epoch: 110, loss: 0.012101143598556519
epoch: 120, loss: 0.010843245312571526
epoch: 130, loss: 0.017391851171851158
epoch: 140, loss: 0.0181632898747921
epoch: 150, loss: 0.021699152886867523
epoch: 160, loss: 0.01942072995007038
epoch: 170, loss: 0.010556275025010109
epoch: 180, loss: 0.008864075876772404
epoch: 190, loss: 0.0011879559606313705
epoch: 200, loss: 0.006390029098838568
epoch: 210, loss: 0.006628625560551882
epoch: 220, loss: 0.008606494404375553
epoch: 230, loss: 0.007750355172902346
epoch: 240, loss: 0.0065074497833848
epoch: 250, loss: 0.012020064517855644
epoch: 260, loss: 0.012469933368265629
epoch: 270, loss: 0.0034214681945741177
epoch: 280, loss: 0.0054931724444031715
epoch: 290, loss: 0.011135067790746689
epoch: 300, loss: 0.0013491115532815456
epoch: 310, loss: 0.011860196478664875
epoch: 320, loss: 0.01797356829047203
epoch: 330, loss: 0.0037162592634558678
epoch: 340, loss: 0.004001731518656015
epoch: 350, loss: 0.005673537962138653
epoch: 360, loss: 0.004153629764914513
epoch: 370, loss: 0.001254882081411779
epoch: 380, loss: 0.00651506707072258
epoch: 390, loss: 0.006151625420898199
epoch: 400, loss: 0.00549811776727438
epoch: 410, loss: 0.006783232092857361
epoch: 420, loss: 0.006614106707274914
epoch: 430, loss: 0.0021149085368961096
epoch: 440, loss: 0.009055135771632195
```

```
epoch: 450, loss: 0.0049789524637162685
epoch: 460, loss: 0.0030827075242996216
epoch: 470, loss: 0.0028490957338362932
epoch: 480, loss: 0.001131778466515243
epoch: 490, loss: 0.004738299176096916
epoch: 500, loss: 0.001480437582358718
epoch: 510, loss: 0.002555401297286153
epoch: 520, loss: 0.0014903664123266935
epoch: 530, loss: 0.010564470663666725
epoch: 540, loss: 0.006091583054512739
epoch: 550, loss: 0.0014177185948938131
epoch: 560, loss: 0.011617898941040039
epoch: 570, loss: 0.0027381370309740305
epoch: 580, loss: 0.001662570284679532
epoch: 590, loss: 0.0038287879433482885
epoch: 600, loss: 0.000702335441019386
epoch: 610, loss: 0.00856531597673893
epoch: 620, loss: 0.0024316778872162104
epoch: 630, loss: 0.01305563747882843
epoch: 640, loss: 0.0037987008690834045
epoch: 650, loss: 0.001760658691637218
epoch: 660, loss: 0.0067083146423101425
epoch: 670, loss: 0.0002473318891134113
epoch: 680, loss: 0.0017146541504189372
epoch: 690, loss: 0.0042451671324670315
epoch: 700, loss: 0.0032528003212064505
epoch: 710, loss: 0.0009643085650168359
epoch: 720, loss: 0.000714918423909694
epoch: 730, loss: 0.00518075842410326
epoch: 740, loss: 0.0005195606499910355
epoch: 750, loss: 0.0004733542737085372
epoch: 760, loss: 0.0016774852992966771
epoch: 770, loss: 0.0016612509498372674
epoch: 780, loss: 0.0007314930553548038
epoch: 790, loss: 0.0015786091098561883
epoch: 800, loss: 0.0010057931067422032
epoch: 810, loss: 0.004655364900827408
```

```
epoch: 820, loss: 0.004052665550261736
epoch: 830, loss: 0.0168419498950243
epoch: 840, loss: 0.0029639306012541056
epoch: 850, loss: 0.004947346169501543
epoch: 860, loss: 0.0021405464503914118
epoch: 870, loss: 0.006192948669195175
epoch: 880, loss: 0.00020130320626776665
epoch: 890, loss: 0.00984048843383789
epoch: 900, loss: 0.002248404547572136
epoch: 910, loss: 0.0019002301851287484
epoch: 920, loss: 0.0014852361055091023
epoch: 930, loss: 0.004002372268587351
epoch: 940, loss: 0.006227927748113871
epoch: 950, loss: 0.004738884046673775
epoch: 960, loss: 0.001745815621688962
epoch: 970, loss: 0.001280197873711586
epoch: 980, loss: 0.00023427626001648605
epoch: 990, loss: 0.0028342956211417913
```