Computation Graph using PyTorch

August 11, 2021

[1]:
```
#Importing the packages

import torch
import torch.optim as optim
import torch.nn as nn
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
#Defining the Hyperparameters

input_size = 1
output_size = 1
num_epochs = 10000
learning_rate = 0.001
```

[3]:
```
# Defining a Toy Dataset

x_train = np.array([[3.3], [4.4], [5.5], [6.71], [6.93], [4.168],[9.779], [6.
 ↪182], [7.59], [2.167], [7.042], [10.791], [5.313], [7.997], [3.1]], dtype=np.
 ↪float32)

y_train = np.array([[1.7], [2.76], [2.09], [3.19], [1.694], [1.573], [3.366],␣
 ↪[2.596], [2.53], [1.221], [2.827], [3.465], [1.65], [2.904], [1.3]],␣
 ↪dtype=np.float32)
```

[4]:
```
# Linear Regression Model

model = nn.Linear(input_size, output_size)

# Loss Function:
criterion = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=learning_rate)
```

[5]:
```
# Training the Model

for epoch in range(num_epochs):
    inputs = torch.from_numpy(x_train)
    targets = torch.from_numpy(y_train)

    outputs = model(inputs)
    loss = criterion(outputs, targets)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if (epoch + 1) % 1000 == 0:
        print("Epoch: {}/{}; \tLoss: {}".format(epoch + 1, num_epochs, loss.
↪item()))
```

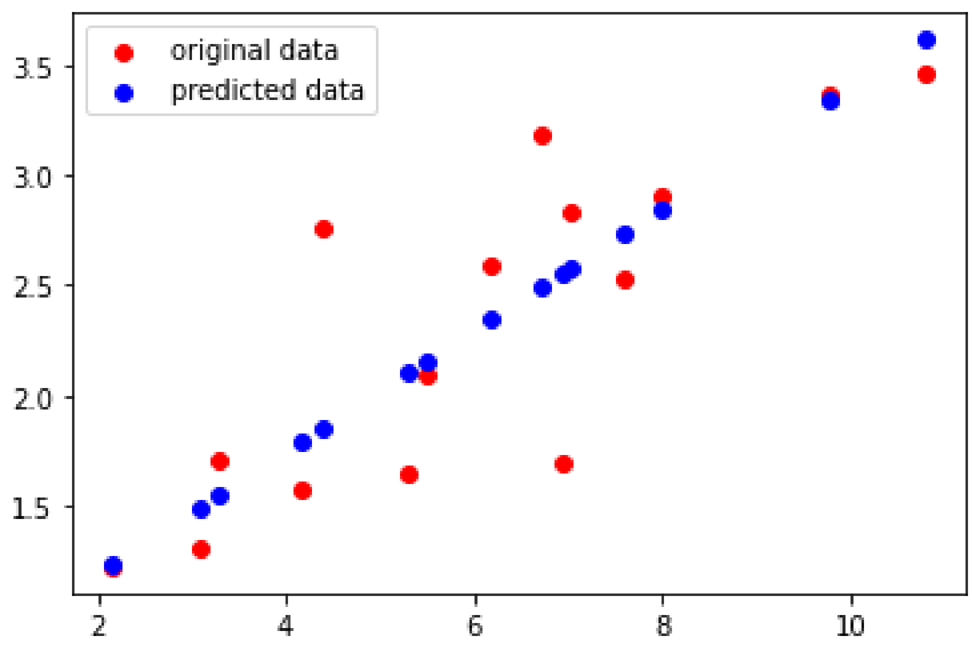
```
Epoch: 1000/10000;      Loss: 0.35465386509895325
Epoch: 2000/10000;      Loss: 0.2800377905368805
Epoch: 3000/10000;      Loss: 0.23539696633815765
Epoch: 4000/10000;      Loss: 0.20868952572345734
Epoch: 5000/10000;      Loss: 0.19271118938922882
Epoch: 6000/10000;      Loss: 0.1831517517566681
Epoch: 7000/10000;      Loss: 0.1774325966835022
Epoch: 8000/10000;      Loss: 0.1740109771490097
Epoch: 9000/10000;      Loss: 0.1719639003276825
Epoch: 10000/10000;     Loss: 0.17073921859264374
```

[6]:

```
# Plotting the outputs

predicted = model(torch.from_numpy(x_train)).detach().numpy()

plt.scatter(x_train, y_train, label='original data', color='r')
plt.scatter(x_train, predicted, label='predicted data', color='b')
plt.legend()
plt.show()
```



[7]:

```
# Logistic Regression Model

import torchvision
import torchvision.transforms as transforms
```

[8]:

```
# Defining Hyperparameters

input_size = 784
num_classes = 10
num_epochs = 20
batch_size = 100
learning_rate = 0.001
```

[9]:

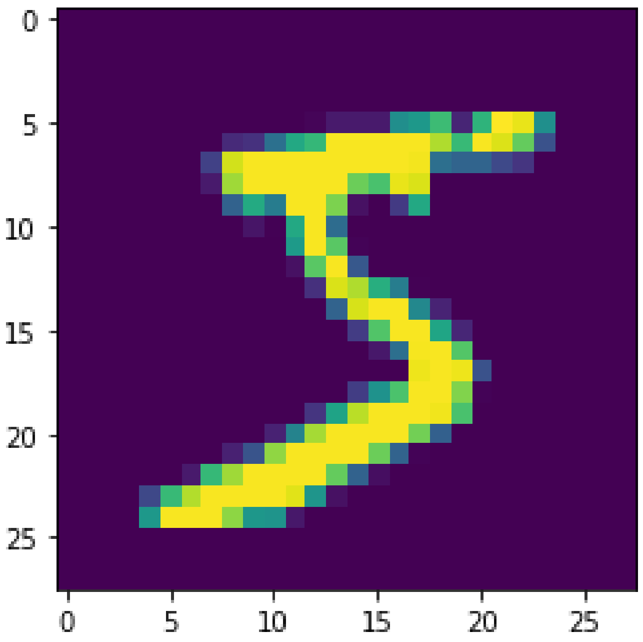
```
# loding the Dataset

train_dataset = torchvision.datasets.MNIST(root="./data", train=True,␣
 ↪transform=transforms.ToTensor(), download=True)
test_dataset = torchvision.datasets.MNIST(root="./data", train = False,␣
 ↪transform=transforms.ToTensor(), download=True)
```

[10]:

```
plt.imshow(train_dataset.train_data[0])
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\torchvision\datasets\mnist.py:55:
UserWarning: train_data has been renamed data
  warnings.warn("train_data has been renamed data")
```



[11]:
```
# Create DataLoader objects

trainloader = torch.utils.data.DataLoader(dataset=train_dataset,␣
 ↪batch_size=batch_size, shuffle=True)
testloader = torch.utils.data.DataLoader(dataset=train_dataset,␣
 ↪batch_size=batch_size, shuffle=True)
```

[12]:
```
model = nn.Linear(input_size, num_classes)

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=learning_rate)
```

[13]:
```
# Training the Model

total_step = 0
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(trainloader):
        images = images.reshape(-1, 28 * 28)
        outputs = model(images)
        loss = criterion(outputs, labels)

        optimizer.zero_grad()
        loss.backward()

        optimizer.step()
        if (i + 1) % 200 == 0:
            print("Epoch: {}/{}, \tIteration: {}/{}, \tLoss: {}".format(epoch +␣
↪1, num_epochs, i + 1, len(trainloader), loss.item()))
```

```
Epoch: 1/20,    Iteration: 200/600,     Loss: 2.11348295211792
Epoch: 1/20,    Iteration: 400/600,     Loss: 1.9768154621124268
Epoch: 1/20,    Iteration: 600/600,     Loss: 1.8140112161636353
Epoch: 2/20,    Iteration: 200/600,     Loss: 1.6307744979858398
Epoch: 2/20,    Iteration: 400/600,     Loss: 1.5595742464065552
Epoch: 2/20,    Iteration: 600/600,     Loss: 1.416924238204956
Epoch: 3/20,    Iteration: 200/600,     Loss: 1.4517042636871338
Epoch: 3/20,    Iteration: 400/600,     Loss: 1.373637080192566
Epoch: 3/20,    Iteration: 600/600,     Loss: 1.3432384729385376
Epoch: 4/20,    Iteration: 200/600,     Loss: 1.187993049621582
Epoch: 4/20,    Iteration: 400/600,     Loss: 1.15120267868042
Epoch: 4/20,    Iteration: 600/600,     Loss: 1.287697196006775
Epoch: 5/20,    Iteration: 200/600,     Loss: 1.023123860359192
Epoch: 5/20,    Iteration: 400/600,     Loss: 1.0275392532348633
Epoch: 5/20,    Iteration: 600/600,     Loss: 0.9594303369522095
Epoch: 6/20,    Iteration: 200/600,     Loss: 1.0727664232254028
Epoch: 6/20,    Iteration: 400/600,     Loss: 1.0496281385421753
Epoch: 6/20,    Iteration: 600/600,     Loss: 0.9817613959312439
Epoch: 7/20,    Iteration: 200/600,     Loss: 0.8590309619903564
Epoch: 7/20,    Iteration: 400/600,     Loss: 0.9260696172714233
Epoch: 7/20,    Iteration: 600/600,     Loss: 0.8033376932144165
Epoch: 8/20,    Iteration: 200/600,     Loss: 0.809929370880127
Epoch: 8/20,    Iteration: 400/600,     Loss: 0.9332719445228577
Epoch: 8/20,    Iteration: 600/600,     Loss: 0.8235397338867188
Epoch: 9/20,    Iteration: 200/600,     Loss: 0.7932674288749695
Epoch: 9/20,    Iteration: 400/600,     Loss: 0.7713871002197266
Epoch: 9/20,    Iteration: 600/600,     Loss: 0.6913479566574097
Epoch: 10/20,   Iteration: 200/600,     Loss: 0.683384120464325
Epoch: 10/20,   Iteration: 400/600,     Loss: 0.7840595841407776
Epoch: 10/20,   Iteration: 600/600,     Loss: 0.798953115940094
Epoch: 11/20,   Iteration: 200/600,     Loss: 0.7547604441642761
Epoch: 11/20,   Iteration: 400/600,     Loss: 0.7991681694984436
Epoch: 11/20,   Iteration: 600/600,     Loss: 0.7662707567214966
Epoch: 12/20,   Iteration: 200/600,     Loss: 0.6646339297294617
Epoch: 12/20,   Iteration: 400/600,     Loss: 0.8230312466621399
Epoch: 12/20,   Iteration: 600/600,     Loss: 0.5955288410186768
Epoch: 13/20,   Iteration: 200/600,     Loss: 0.6695789098739624
Epoch: 13/20,   Iteration: 400/600,     Loss: 0.6369901895523071
Epoch: 13/20,   Iteration: 600/600,     Loss: 0.7367023229598999
Epoch: 14/20,   Iteration: 200/600,     Loss: 0.788683295249939
Epoch: 14/20,   Iteration: 400/600,     Loss: 0.6644474864006042
Epoch: 14/20,   Iteration: 600/600,     Loss: 0.5794206261634827
Epoch: 15/20,   Iteration: 200/600,     Loss: 0.5658922791481018
```

```
Epoch: 15/20,   Iteration: 400/600,     Loss: 0.6202057003974915
Epoch: 15/20,   Iteration: 600/600,     Loss: 0.6490528583526611
Epoch: 16/20,   Iteration: 200/600,     Loss: 0.589232861995697
Epoch: 16/20,   Iteration: 400/600,     Loss: 0.653016984462738
Epoch: 16/20,   Iteration: 600/600,     Loss: 0.5731772780418396
Epoch: 17/20,   Iteration: 200/600,     Loss: 0.6609398126602173
Epoch: 17/20,   Iteration: 400/600,     Loss: 0.6446942090988159
Epoch: 17/20,   Iteration: 600/600,     Loss: 0.5710018277168274
Epoch: 18/20,   Iteration: 200/600,     Loss: 0.6565715074539185
Epoch: 18/20,   Iteration: 400/600,     Loss: 0.5492011904716492
Epoch: 18/20,   Iteration: 600/600,     Loss: 0.6849066019058228
Epoch: 19/20,   Iteration: 200/600,     Loss: 0.5802387595176697
Epoch: 19/20,   Iteration: 400/600,     Loss: 0.5950618386268616
Epoch: 19/20,   Iteration: 600/600,     Loss: 0.5446805357933044
Epoch: 20/20,   Iteration: 200/600,     Loss: 0.6342569589614868
Epoch: 20/20,   Iteration: 400/600,     Loss: 0.5714309215545654
Epoch: 20/20,   Iteration: 600/600,     Loss: 0.6127232313156128
```

[14]:

```python
# Testing the model

with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in testloader:
        images = images.reshape(-1, 28 * 28)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        correct += (predicted == labels).sum()
        total += labels.size(0)
    print("Accuracy of the model: {}".format(float(correct) / float(total)))
```

```
Accuracy of the model: 0.8650666666666667
```