Building a Feedforward Neural Network using Pytorch NN Module

August 11, 2021

[1]:
```
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
```

[2]:
```
# Device configuration
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

[3]:
```
# Hyper-parameters
input_size = 784
hidden_size = 500
num_classes = 10
num_epochs = 5
batch_size = 100
learning_rate = 0.001
```

[4]:
```
# MNIST 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())
```

[5]:
```
# Data loader
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)

test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=batch_size,
                                          shuffle=False)
```

[6]:
```
# Fully connected neural network with one hidden layer
class NeuralNet(nn.Module):
```

```
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out

model = NeuralNet(input_size, hidden_size, num_classes).to(device)
```

[7]:
```
# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

[8]:
```
# Train the model
total_step = len(train_loader)
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        # Move tensors to the configured device
        images = images.reshape(-1, 28*28).to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if (i+1) % 100 == 0:
            print ('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'
                   .format(epoch+1, num_epochs, i+1, total_step, loss.item()))
```

```
Epoch [1/5], Step [100/600], Loss: 0.3857
Epoch [1/5], Step [200/600], Loss: 0.2867
Epoch [1/5], Step [300/600], Loss: 0.3168
Epoch [1/5], Step [400/600], Loss: 0.2138
Epoch [1/5], Step [500/600], Loss: 0.1319
Epoch [1/5], Step [600/600], Loss: 0.1362
Epoch [2/5], Step [100/600], Loss: 0.0794
Epoch [2/5], Step [200/600], Loss: 0.1434
```

```
Epoch [2/5], Step [300/600], Loss: 0.1334
Epoch [2/5], Step [400/600], Loss: 0.1534
Epoch [2/5], Step [500/600], Loss: 0.1131
Epoch [2/5], Step [600/600], Loss: 0.0341
Epoch [3/5], Step [100/600], Loss: 0.1792
Epoch [3/5], Step [200/600], Loss: 0.0709
Epoch [3/5], Step [300/600], Loss: 0.0900
Epoch [3/5], Step [400/600], Loss: 0.0329
Epoch [3/5], Step [500/600], Loss: 0.0979
Epoch [3/5], Step [600/600], Loss: 0.0661
Epoch [4/5], Step [100/600], Loss: 0.1074
Epoch [4/5], Step [200/600], Loss: 0.0342
Epoch [4/5], Step [300/600], Loss: 0.0892
Epoch [4/5], Step [400/600], Loss: 0.0731
Epoch [4/5], Step [500/600], Loss: 0.0371
Epoch [4/5], Step [600/600], Loss: 0.0266
Epoch [5/5], Step [100/600], Loss: 0.0224
Epoch [5/5], Step [200/600], Loss: 0.0190
Epoch [5/5], Step [300/600], Loss: 0.0177
Epoch [5/5], Step [400/600], Loss: 0.0409
Epoch [5/5], Step [500/600], Loss: 0.0435
Epoch [5/5], Step [600/600], Loss: 0.0304
```

[9]:
```python
# Test the model
# In test phase, we don't need to compute gradients (for memory efficiency)
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in test_loader:
        images = images.reshape(-1, 28*28).to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    print('Accuracy of the network on the 10000 test images: {} %'.format(100 *␣
 ↪correct / total))
```

```
Accuracy of the network on the 10000 test images: 98.01 %
```

[10]:
```python
# Save the model checkpoint
torch.save(model.state_dict(), 'model.ckpt')
```