REPORT

DNN Using MNIST Assignment

과목명	딥러닝
담당교수	정우환 교수님
학생이름	박준우
학과	인공지능학과
학번	2021006253
제출일	2023.10.10

HANYANG UNIVERSITY

Source code for dataload

```
batch_size=12                                                                                   //배치 사이즈 설정

train_data=datasets.MNIST('dataset', train=True, download=True, transform=transforms.ToTensor()) //학습 data 다운

test_data=datasets.MNIST('dataset', train=False, download=True,transform=transforms.ToTensor())   //test data 다운


train_loader = torch.utils.data.DataLoader(train_data,batch_size=batch_size,shuffle=True)        //학습 data 로드

test_loader = torch.utils.data.DataLoader(test_data,batch_size=batch_size)                       //test data 로드
```

Source code for model

```
class MLP_h(nn.Module):                                          //nn.Module 상속
    def __init__(self,hidden_units):                             //부모 클래스 nn.Module 호출
        super(MLP_h, self).__init__()

        self.in_dim=28*28                                        //입력과 출력의 차원
        self.out_dim=10                                          //클래스의 수

        layers = []                                              //레이어 추가할 리스트
        layers.append(nn.Linear(self.in_dim,hidden_units[0]))    //선형 레이어와 ReLU 초기화
        layers.append(nn.ReLU())
        for i in range(len(hidden_units)-1):                     //은닉층과 활성화 함수 순차적 추가
            layers.append(nn.Linear(hidden_units[i],hidden_units[i+1]))
            layers.append(nn.ReLU())
        layers.append(nn.Linear(hidden_units[-1],self.out_dim))  //마지막 선형 레이어 추가

        self.l_layers = nn.ModuleList(layers) # Use nn.ModuleList  //layers 를 모듈리스트로 변환

    def forward(self,x):
        a=x.view(-1,self.in_dim)
        for layer in self.l_layers:                              //모든 결과를 a 에 저장
            a = layer(a)
        return a
```

Source code for training

```
models = {                                                    //모델별 은닉층 개수, 뉴런 수
    '2layers': MLP_h([512]),                                  // layer2 개 1hidden+1output
    '3layers': MLP_h([512, 256]),                             // layer3 개 2hidden+1output
    '4layers': MLP_h([512, 256, 128]),                        // layer4 개 3hidden+1output
    '5layers': MLP_h([512, 256, 128, 64])                     // layer5 개 4hidden+1output
}
accuracies = {}                                               //정확도 저장용 딕셔너리

criterion=nn.CrossEntropyLoss()                               //크로스엔트로피 손실 함수

for model_name, model in models.items():
    print(f"Training {model_name} model...")

    optimizer=optim.SGD(model.parameters(),lr=0.01)           //모델별 최적함수  초기화
    for epoch in range(10):
        running_loss=0.0                                      //epoch 별 누적 손실값 초기화
        for i, data in enumerate(train_loader,0):             //train data 훈련
            inputs, labels=data                               //입력 data 와 label 분리

            optimizer.zero_grad()                             #backward//옵티마이저 기울기 초기화

            outputs=model(inputs)                             #forward
            loss=criterion(outputs,labels)                    #backward
            loss.backward()                                   #backward
            optimizer.step()                                  #backward

            running_loss+=loss.item()                         //누적 손실값 업데이트
            if(i+1)%2000==0:                                  //2000 개 마다 손실값 출력
                print('[%d,%5d] loss: %.3f'% (epoch + 1, i+1,running_loss/2000))
                running_loss=0.0

    print(f"Testing {model_name} model...")

    n_predict = 0                                             //예측 횟수
```

```
    n_correct = 0                                              //맞힌 횟수

    with torch.no_grad():                                      //모델 평가 시 기울기 계산 X
        for data in test_loader:                               //test data 로 테스트 진행
            inputs, labels = data
            outputs = model(inputs)
            _, predicted = torch.max(outputs, 1)               //최대값 클래스를 예측값으로
            n_predict += len(predicted)                        //예측 데이터 수와 맞힌 개수
            n_correct += (labels == predicted).sum().item()

    accuracy = n_correct / n_predict                           //각 모델별 정확도 계산
    accuracies[model_name] = accuracy                          //모델별 정확도 저장
    print(f"{model_name} Accuracy: {accuracy:.3f}₩n")          //모델별 정확도 출력

print("Finsih the Test")
print("Accuracies:", accuracies)
```

Source code for plot

```
layers = list(accuracies.keys())

acc_values = list(accuracies.values())


plt.figure(figsize=(10, 6))

plt.plot(layers, acc_values, marker='o', color='b', label='Accuracy')

plt.title("Acc per Layers")

plt.xlabel("Num of Layers")

plt.ylabel("Accuracy")

plt.grid(True)

plt.legend()

plt.show()
```

Plot accuracy varying the number of layers

```
Training 2layers model...
[1, 2000] loss: 0.804
[1, 4000] loss: 0.370
[2, 2000] loss: 0.296
[2, 4000] loss: 0.274
[3, 2000] loss: 0.231
[3, 4000] loss: 0.222
[4, 2000] loss: 0.196
[4, 4000] loss: 0.185
[5, 2000] loss: 0.164
[5, 4000] loss: 0.159
[6, 2000] loss: 0.146
[6, 4000] loss: 0.137
[7, 2000] loss: 0.125
[7, 4000] loss: 0.120
[8, 2000] loss: 0.111
[8, 4000] loss: 0.110
[9, 2000] loss: 0.098
[9, 4000] loss: 0.101
[10, 2000] loss: 0.087
[10, 4000] loss: 0.089
Testing 2layers model...
2layers Accuracy: 0.972
```

2layers

```
Training 3layers model...
[1, 2000] loss: 1.017
[1, 4000] loss: 0.367
[2, 2000] loss: 0.283
[2, 4000] loss: 0.239
[3, 2000] loss: 0.194
[3, 4000] loss: 0.177
[4, 2000] loss: 0.150
[4, 4000] loss: 0.133
[5, 2000] loss: 0.114
[5, 4000] loss: 0.109
[6, 2000] loss: 0.092
[6, 4000] loss: 0.093
[7, 2000] loss: 0.078
[7, 4000] loss: 0.081
[8, 2000] loss: 0.066
[8, 4000] loss: 0.067
[9, 2000] loss: 0.060
[9, 4000] loss: 0.056
[10, 2000] loss: 0.051
[10, 4000] loss: 0.047
Testing 3layers model...
3layers Accuracy: 0.978
```

3layers

```
Training 4layers model...
[1, 2000] loss: 1.511
[1, 4000] loss: 0.417
[2, 2000] loss: 0.282
[2, 4000] loss: 0.228
[3, 2000] loss: 0.175
[3, 4000] loss: 0.151
[4, 2000] loss: 0.119
[4, 4000] loss: 0.114
[5, 2000] loss: 0.093
[5, 4000] loss: 0.086
[6, 2000] loss: 0.075
[6, 4000] loss: 0.069
[7, 2000] loss: 0.055
[7, 4000] loss: 0.061
[8, 2000] loss: 0.049
[8, 4000] loss: 0.048
[9, 2000] loss: 0.042
[9, 4000] loss: 0.039
[10, 2000] loss: 0.033
[10, 4000] loss: 0.034
Testing 4layers model...
4layers Accuracy: 0.979
```

4layers

```
Training 5layers model...
[1, 2000] loss: 2.179
[1, 4000] loss: 0.662
[2, 2000] loss: 0.299
[2, 4000] loss: 0.222
[3, 2000] loss: 0.153
[3, 4000] loss: 0.140
[4, 2000] loss: 0.104
[4, 4000] loss: 0.098
[5, 2000] loss: 0.074
[5, 4000] loss: 0.080
[6, 2000] loss: 0.060
[6, 4000] loss: 0.061
[7, 2000] loss: 0.048
[7, 4000] loss: 0.048
[8, 2000] loss: 0.037
[8, 4000] loss: 0.038
[9, 2000] loss: 0.030
[9, 4000] loss: 0.028
[10, 2000] loss: 0.026
[10, 4000] loss: 0.024
Testing 5layers model...
5layers Accuracy: 0.976
```

5layers

Result of each model

```
Finsih the Test
Accuracies: {'2layers': 0.9719, '3layers': 0.9776, '4layers': 0.9786, '5layers': 0.9756}
```

Figure