Assignment1

2016025196

김동규

실행 환경

Google colab, pytorch, python

코드 설명

Assignment1 코드는 기존 assignment1.ipynb파일에서 Classifier와 main실행코드를 수정하여 만들어졌습니다.

해당 코드에는 Pytorch의 Dropout, BatchNormal, Adam API를 통하여 구현되었습니다.

```python
class Classifier(nn.Module):
    # 모델의 코드는 여기서 작성해주세요


    def __init__(self):
        super(Classifier, self).__init__()
        self.linear1=nn.Linear(32*32*3,32*32)
        self.linear2=nn.Linear(32*32,512)
        self.linear3=nn.Linear(512,256)
        self.linear4=nn.Linear(256,10)

        self.bn1=nn.BatchNorm1d(32*32)
        self.bn2=nn.BatchNorm1d(512)
        self.bn3=nn.BatchNorm1d(256)
        self.activation=nn.ReLU()
        self.acti2=nn.Sigmoid()
        self.dropout=nn.Dropout(0.35)

    def forward(self, x):
        z1=self.linear1(x)
        z1=self.bn1(z1)
        a1=self.activation(z1)
        a1=self.dropout(a1)

        z2=self.linear2(a1)
        z2=self.bn2(z2)
        a2=self.activation(z2)
        a2=self.dropout(a2)

        z3=self.linear3(a2)
        z3=self.bn3(z3)
        a3=self.activation(z3)

        output=self.linear4(a3)
        return output
```

Classifier는 총 4개의 layer로 되어있으며, Layer1(32*32*3,32*32), Layer2(32*32,512), Layer3(512,256), Layer4(256,10)으로 구성되었습니다.

이에 따라 총 세개의 BatchNormal이 선언되어 있으며, 이미지 분류라는 특성상 1,2,3 Layer는 ReLU activation을 사용합니다. 실험결과 layer 3에도 ReLU를 사용하는 것이 가장 정확도가 높았습니다.

Dropout의 확률은 여러 실험 결과 35%가 가장 적절한 수치임을 알 수 있었습니다.

```python
if __name__ == "__main__":
    # 학습코드는 모두 여기서 작성해주세요
    transform=transforms.Compose([transforms.ToTensor(),transforms.Normalize((0.4914,0.4822,0.4465),(0.247,0.243,0.261))])

    train_dataset = torchvision.datasets.CIFAR10(root="CIFAR10/",
                                                 train=True,
                                                 transform=transform,
                                                 download=True)
    test_dataset = torchvision.datasets.CIFAR10(root="CIFAR10/",
                                                train=False,
                                                transform=transform,
                                                download=True)



    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    model = Classifier()
    model=model.to(device)
    model.train()
    optimizer=torch.optim.Adam(params=model.parameters(),lr=0.001,betas=(0.9,0.999))
    criterion=nn.CrossEntropyLoss()



    batch_size=64
    train_dataloader=torch.utils.data.DataLoader(train_dataset,batch_size=batch_size,shuffle=True)
    total_batch_num=len(train_dataloader)
    epochs=28
    for epoch in range(epochs):
        avg_cost=0
        for b_x,b_y in train_dataloader:
          b_x=b_x.view(-1,32*32*3).to(device)
          logits=model(b_x)
          loss=criterion(logits , b_y.to(device))

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

          avg_cost+=loss/total_batch_num
        print("Epoch : {} / {} , Cost : {}".format(epoch+1,epochs,avg_cost))

    #===================================
    #           train code
    #===================================

    torch.save(model.state_dict(), 'model.pt')  # 학습된 모델을 저장하는 코드입니다.
```

학습 코드는 main문 내에서 구현되었으며, 이전 수업에서 나온 예시 코드를 활용하여 작성하였습니다. gpu가 사용가능한지 여부를 확인하여 사용하는 코드와, train_dataloader를 추가한 뒤, batch size와 epochs에 따라 모델의 train을 진행합니다. Optimizer로는 Adam을 사용하였으며, lr=0.001, betas=(0.9,0.999)일 때 가장 성능이 높은 것을 확인하여, 이 수치는 조정하지 않았습니다. 진행상황과 cost값을 보기 위해 매 epoch마다 진행상황을 출력합니다.

실험 결과

아래는 첫 작성 코드입니다.

```
 def __init__(self):
     super(Classifier, self).__init__()
     self.linear1=nn.Linear(32*32*3,256)
     self.linear2=nn.Linear(256,128)
     self.linear3=nn.Linear(128,10)

     self.bn1=nn.BatchNorm1d(256)
     self.bn2=nn.BatchNorm1d(128)
     self.activation=nn.Sigmoid()

 def forward(self, x):
     z1=self.linear1(x)
     z1=self.bn1(z1)
     a1=self.activation(z1)

     z2=self.linear2(a1)
     z2=self.bn2(z2)
     a2=self.activation(z2)

     output=self.linear3(a2)
     return output
```

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model = Classifier()
model=model.to(device)
model.train()
optimizer=torch.optim.Adam(params=model.parameters(),lr=0.001,betas=(0.9,0.999))
criterion=nn.CrossEntropyLoss()


batch_size=128
train_dataloader=torch.utils.data.DataLoader(train_dataset,batch_size=batch_size,shuffle=True)

epochs=50
for epoch in range(epochs):

    for b_x,b_y in train_dataloader:
      b_x=b_x.view(-1,32*32*3).to(device)
      logits=model(b_x)
      loss=criterion(logits , b_y.to(device))

      optimizer.zero_grad()
      loss.backward()
      optimizer.step()
    print("Epoch : {} / {}".format(epoch+1,epochs))

#==================================
#           train code
#==================================

torch.save(model.state_dict(), 'model.pt')  # 학습된 모델을 저장하는 코드입니다.
```

```
Files already downloaded and verified
Accuracy on test set : 44.0400%
```

이러한 결과를 얻은 뒤, 여러 수치들을 조정해가며 결과값이 좋아지던 결과만 작성했습니다.

```
Files already downloaded and verified
Accuracy on test set : 47.4500%
```
epochs=30

```
Files already downloaded and verified
Accuracy on test set : 48.3100%
```
ReLU사용

```
Files already downloaded and verified
Accuracy on test set : 51.4900%
```
batchsize=64

54%(캡처본 사라짐) 드랍아웃 적용(30%)

```
Files already downloaded and verified
Accuracy on test set : 55.3200%
```
linear2 256,256로 수정

```
Files already downloaded and verified
Accuracy on test set : 56.7300%
```
레이어3 256,128 추가

```
Files already downloaded and verified
Accuracy on test set : 56.9300%
```
epoch 25

```
Files already downloaded and verified
Accuracy on test set : 57.2300%
```
1레이어 32*32*3, 32*32 수정

```
Files already downloaded and verified
Accuracy on test set : 58.2300%
```
2레이어 32*32, 512 3레이어 512,256 수정

```
Files already downloaded and verified
Accuracy on test set : 58.5000%
```
drop30% , epoch 29 및 layer3에 sigmoid사용

```
Files already downloaded and verified
Accuracy on test set : 59.2200%
```
input nomalization적용, dropout 32%

```
Files already downloaded and verified
Accuracy on test set : 59.3500%
```
dropout 35%

최종 모델 : epochs=29, Dropout Probability=35%, batch size=64, Layer={(32*32*3, 32*32), (32*32,256), (256,128), (128,10)}

여러 차례의 실험 결과 59.35%의 정확도를 갖는 최적화된 모델을 생성하였습니

다.

결론

1. epochs는 30을 기준으로 점차 overfitting으로 인해 정확도가 낮아진다.

2. 이미지 분류기 특성상 activation함수로 ReLU함수가 성능이 높다.

3. batch size는 32~256등 여러 결과를 사용해본 결과 64가 가장 성능이 좋다.

4. dropout은 25~35%가 성능이 가장 좋고, 최종 layer의 크기 기준으로 35%가 가장 성능이 높다.

5. Layer의 크기는 input size 32*32*3을 기준으로 Layer1(32*32*3, 32*32), Layer2 (32*32, 256), Layer3(256,128), Layer4(128,10)이 성능과 학습시간상으로 가장 성능이 좋다.