딥러닝및응용
Assignment 1

컴퓨터소프트웨어학부 심승현

목차

- 코드 설명
- 실험 결과

코드 설명

```
import torch
import torch.nn as nn

import torchvision
import torchvision.transforms as transforms

import matplotlib.pyplot as plt
import numpy as np

# 미리 작성된 코드들은 수정할 수 없으며, 이외의 코드를 작성하시면 됩니다.

torch.manual_seed(0)
torch.cuda.manual_seed(0)
torch.cuda.manual_seed_all(0)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

사용할 Library를 import 하였습니다.

성능 비교를 위하여 Random Seed 를 고정하였습니다.

빠른 빌드를 위하여 그래픽카드를 사용하도록 하는 부분입니다.

코드 설명

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

    def __init__(self):
        super(Classifier, self).__init__()
        self.linear1 = nn.Linear(32*32*3, 512)
        self.linear2 = nn.Linear(512, 256)
        self.linear3 = nn.Linear(256, 64)
        self.linear4 = nn.Linear(64, 10)

        self.dropout = nn.Dropout(0.15) # dropout

        self.activation = nn.ReLU()

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

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

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

        z4 = self.linear4(a3)

        return z4
```

강의에서 사용한 5.Regularization.pdf 에서 제시된 코드에서 Linear layer 의 개수를 조정하고, in_features, out_features 를 조정하였습니다.

Dropout을 적용하였습니다.
0.5~0.1 사이의 값으로 실험하였으며
0.15 일 때 가장 높은 정확도를 보였습니다.

Activation function 을 sigmoid 에서 ReLU 로 변경하였습니다.

코드 설명

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

    batch_size = 512
    train_dataloader = torch.utils.data.DataLoader(train_dataset, batch_size, shuffle = True )
    test_dataloader = torch.utils.data.DataLoader(test_dataset, batch_size)

    model = Classifier().to(device)
    optimizer = torch.optim.Adam(model.parameters(), lr=0.00005) # optim?
    criterion = nn.CrossEntropyLoss()
```

Normalization 을 진행하였습니다

Shuffle을 통해 학습이 치우치지 않도록 하였습니다.

Optimizer를 ADAM 으로 변경하였습니다.

코드 설명

Epochs 와 lambda 값을 조정하였습니다.

L2 Regularization 을 진행하였습니다.

```
epochs = 160
lmbd = 0.1
train_avg_costs = []
test_avg_costs = []

test_total_batch = len(test_dataloader)
total_batch_num = len(train_dataloader)

for epoch in range(epochs):
    avg_cost = 0
    model.train()
    for b_x, b_y in train_dataloader:
      b_x = b_x.view(-1, 32*32*3).to(device)
      logits = model(b_x) # forward propagation
      loss = criterion(logits, b_y.to(device)) # get cost

      reg = model.linear1.weight.pow(2.0).sum()
      reg += model.linear2.weight.pow(2.0).sum()
      reg += model.linear3.weight.pow(2.0).sum()
      reg += model.linear4.weight.pow(2.0).sum()

      loss += lmbd*reg/len(b_x)/2.

      optimizer.zero_grad()
      loss.backward() # back propagation
      optimizer.step() # update parameters

      avg_cost += loss / total_batch_num
    train_avg_costs.append(avg_cost.detach()) # point
    print('Epoch : {} / {}, cost : {}'.format(epoch+1, epochs, avg_cost))

    test_avg_cost = 0
    model.eval()
    for b_x, b_y in test_dataloader:
      b_x = b_x.view(-1, 32*32*3).to(device)
      with torch.no_grad():
        logits = model(b_x)
        test_loss = criterion(logits, b_y.to(device))
      test_avg_cost += test_loss / test_total_batch

    test_avg_costs.append(test_avg_cost.detach()) # point
```

실험 결과

실험 요소

- activation function
- optimizer
- hyperparameters (lr, lambda …)

실험 결과

Activation function

Sigmoid, Softmax(d=1), Tanh, Relu 함수를 비교하였고,
ReLU() 를 사용하였을 때 가장 높은 정확도를 보였습니다

Optimizer

SGD, Adam, AdaGrad 를 비교하였고,
Adam 을 사용하였을 때 가장 높은 정확도를 보였습니다.

실험 결과

Hyperparameters

- Learning rate : 0.05 이상의 값에서는 20% 미만의 정확도를 보였으며,
  제출한 파일에서는 0.00005로 설정하였습니다.

- Layer : Linear layer 와 in_features, out_features 를 모두 늘렸습니다.
  값이 너무 커지는 경우 빌드 시간이 급격히 늘어나는 현상이 있어
  빌드 시간을 고려하여 값을 선택하였습니다.

- Lmbd : 실습 슬라이드에 제시된 0.003을 적용했을 시에 정확도가 낮은 현상이 있었습니다.
  제출한 파일에서는 0.1로 설정되어 있으며
  0.3까지 증가시켰을 때는 오히려 정확도가 낮아지는 현상이 있었습니다.

- Epoch : 70, 100, 130, 150, 160, 200 을 테스트해 본 결과
  Epoch 이 늘어날수록 학습 정확도가 높아지는 경향이 있었습니다.
  160과 200의 차이가 크지 않아, 빌드 시간이 짧은 160을 선택하였습니다.

실험 결과

```
# 학습된 모델의 성능을 평가하는 코드입니다.
# 아래의 코드로 평가를 진행할 예정이므로 아래의 코드가 정상 동작 해야하며, 제출전 모델의 성능을 확인하시면 됩니다.

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

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


test_dataloader = torch.utils.data.DataLoader(test_dataset, batch_size=10000)

classifier = Classifier().to(device)
classifier.load_state_dict(torch.load('model.pt'))
classifier.eval()


for data, label in test_dataloader:
    data = data.view(-1, 32 * 32 * 3).to(device)

    with torch.no_grad():
        logits = classifier(data)

        pred = torch.argmax(logits, dim=1)

        total = len(label)
        correct = torch.eq(pred, label.to(device)).sum()

        print("Accuracy on test set : {:.4f}%".format(100 * correct / total))
```

```
Files already downloaded and verified
Accuracy on test set : 56.1200%
```

앞서 기재한 hyperparameter들을 적용한 결과,
56.12%의 정확도를 보였습니다.