과제 4: Training Neural Networks

김의찬
국민대학교 전자공학부
uichan8@naver.com

요 약

기본적인 함수를 이용하여 DNN 의 기본 구조와 작동 방식을 익히고 모델을 만듭니다. 모델의 하이퍼 파라미터를 조정하고 인풋을 조정해보며 좋은 성능이 나오도록 조정해봅니다.

1. 서론

기존 과제까지는 고전적인 방식으로 인간이 직접 프로그램을 모델링해서, (bag of words, Harr like) 분류를 했습니다. 이번에는 DNN 기법으로 모델을 학습시켜 목표를 달성하기 위한 복잡한 함수를 찾아내 사진들을 분류해보고자 합니다.

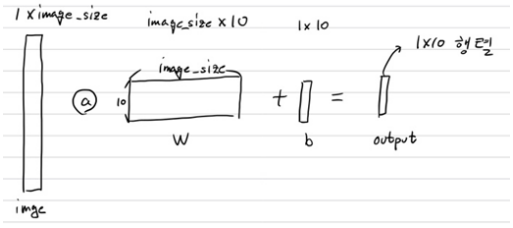
2. 과제 수행 내용

#가산점 수행 내용 :part1 하이퍼 파라미터 튜닝+ part2 모델 다양하게 구성

2.1 model.py 구현

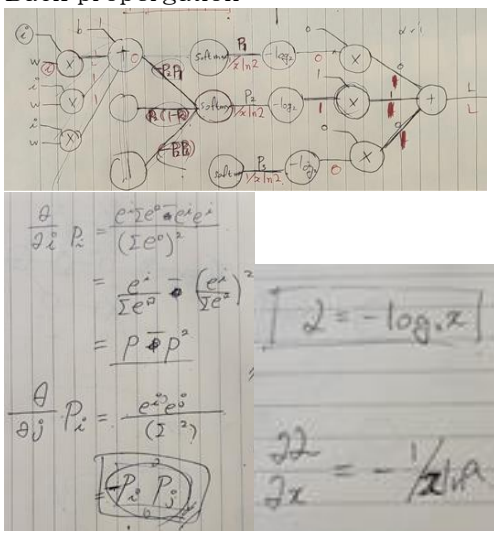
Model class 내의 두 파라미터 객체, W 와 b(weights, biases)를 학습을 통해 결정할 것 입니다.

1. 학습에 사용될 이미지 한장을 랜덤으로 가져옵니다.(이미 구현되어있음)
2. Feedforward



output 으로 나온행렬을 지수에 올려 준 후 soft max 기법으로 각각의 확률을 구해줍니다.

3. Loss function
우리는 해당되는 항이 1 에 가깝도록 하고자 합니다. 즉 해당 클래스 값이 0 에 가까울수록 큰 패널티를, 1 에 가까울수록 작은 패널티를 줘야합니다.
크로스 엔트로피 기법을 사용하여, 이를 표현해 줍니다.
4. Back propergation



그래프를 그리고 각각을 미분하여 코드에 계산식을 표현해줍니다. 좌측 이미지가 soft max 에 대한 미분 우측이 cross entropy 에 대한 미분입니다.

5. 파라미터값 업데이트
원래 있던 값에서 기울기와 lr 을 곱해서 빼줍니다.

2.2 model_pytorch.py 구현

Lenet5 를 유사하게 구현합니다. 실제 논문에서는 컨볼루션 하고 샘플링하고 거기에 wx+b 한 후 sigmod 에 넣어주고 이런식으로 현재와 다른 방향으로 하기 때문에 기존모델에서 조정하였습니다.

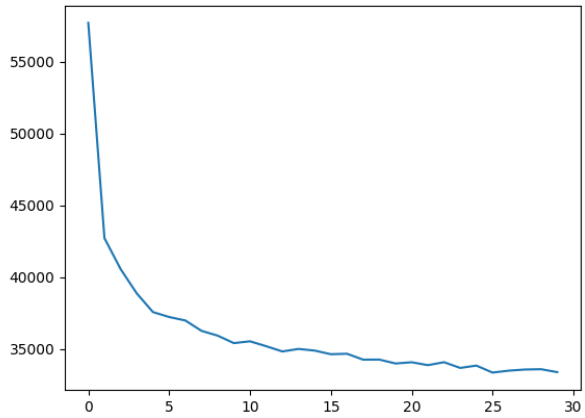
5*5 -> ReLU -> Maxpool (2*2)-> 5*5 -> ReLU -> Maxpool (2*2) -> fc -> linear(120) -> ReLU -> linear(84) -> ReLU -> linear(class num)
으로 구성하였습니다. 또한 resnet 비슷하게 구현하여 레이어를 깊게 쌓아봤습니다.

3. 실험 결과 및 분석

과제 1

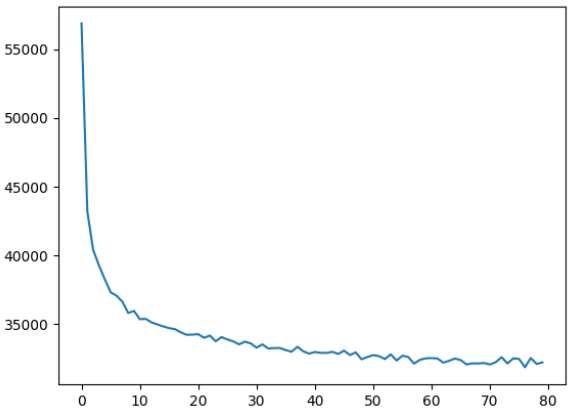
기본셋팅 -> lr : 0.0005 epoch : 30 정확도 87.5%

```
Epoch 29: Total loss: [33483.9331376]
nn model training accuracy: 88%
Epoch 29: Total loss: [33551.20882115]
nn model training accuracy: 87%
```



변형 1 -> epoch : 80 정확도 88%

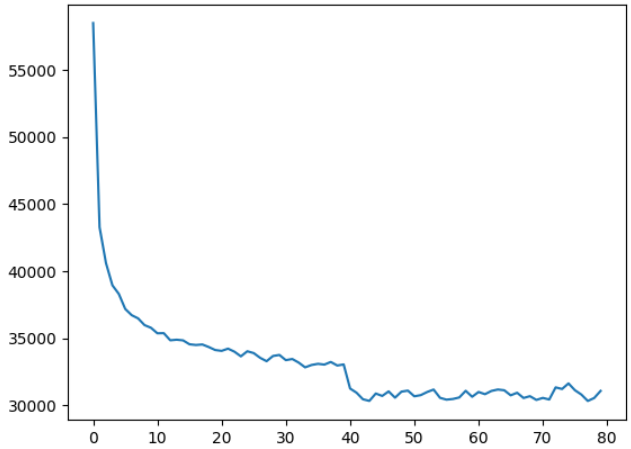
```
Epoch 79: Total loss: [32221.71428494]
nn model training accuracy: 89%
Epoch 79: Total loss: [32108.69979837]
nn model training accuracy: 87%
```



변형 2 -> 40 epoch 에 leaning late * 0.1 정확도 90%

```
Epoch 99: Total loss: [30222.7210015]
nn model training accuracy: 90%
Epoch 99: Total loss: [30969.21680783]
nn model training accuracy: 90%
```

정확도의 표준편차가 생각보다 높아서 정확도로 모델을 판단하기가 어렵다고 판단됩니다. 하지만 loss 의 분포는 확실 히 차이가 났습니다.



과제 2

기본 -> lr : 0.0005 epoch : 30 batch 정확도: 11%

```
Epoch [30/30], Step [90/93], Loss: 2.6886
pytorch cnn model training accuracy: 11%
```

변형 1 -> lr : 0.005 epoch : 30 정확도: 30%

```
Epoch [30/30], Step [90/93], Loss: 0.0341
pytorch cnn model training accuracy: 30%
```

변형 2 -> lr : 0.005 epoch : 100 정확도: 32%

```
Epoch [100/100], Step [90/93], Loss: 0.0000
pytorch cnn model training accuracy: 32%
```

Resnet -> lr : 0.00005 epoch : 30 정확도: 36%

```
resnet Epoch [30/30], Step [90/93], Loss: 0.0039
accuracy : 36.07%
pytorch cnn model training accuracy: 36%
```

4. 결론

직접 딥러닝 모델을 만들어 작동시켰고, pytorch 를 이용하여 여러가지 모델을 만들었습니다.

학습률을 잘못 잡으면 성능에 큰 영향을 미칠 수 있습니다. 또한 학습을 돌리면서 학습률을 줄여나가는 것도 성능에 큰 좋은 영향을 줄 수 있습니다.

epoch 을 많이 돌릴수록 어느 정도 까지는 더 좋은 성능이 나오는 것을 확인 할 수 있었습니다.

레이어를 깊게 쌓을수록, 항상 좋은 성능의 모델이 나오지 않는다는 것을 확인 할 수 있었습니다.

참고문헌

[1] Y.LeCun ,“Gradient-Based Learning Applied to Document Recongnition”,1998 http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf //Lenet5 구성 참고

[2] https://blog.naver.com/jjunsss/222522565616 //Resnet pytorch 구성 참고

5 월 28 일 업데이트 내용

1. model.py 의 nan 문제 해결(softmax 함수 변경)

$softmax[0] = \frac{e^a}{e^a+e^b+e^c}$의 꼴을 가지는데 e^a 에서 a 가 커져 버릴 경우 e^a 값자체가 엄청나게 커진다. 이로 인해 오버플로우 현상이 발생해서 nan 가 뜨게 된다.

$$softmax[0] = \frac{e^a}{e^a+e^b+e^c} = \frac{e^a}{e^a+e^b+e^c} * \frac{e^{val}}{e^{val}}$$
$$= \frac{e^{a+val}}{e^{a+val}+e^{b+val}+e^{c+val}}$$

다음과 같은 관계식에서 val 값을 a,b,c 중에서 가장 큰 값(max) 로 잡아주면 모든 e^a 꼴의 지수함수가 1 보다 작은 값으로 나오게 된다. 이를 이용하여 overflow 현상을 방지할 수 있다.

scenerec 데이터 input_size : 128 epoch : 80

```
Epoch 79: Total loss: [137.75557019]
nn model training accuracy: 20%
```

2. model_pytorch.py 입력 이미지에 대한 유연성 적용

모델 클래스 인풋에 이미지 크기를 추가하여 모든 이미지에 대해서 학습 가능 하도록 수정

mnist 데이터 batchsize :1 epoch:1

```
Lenet5
cuda : True
Epoch [1/1], Step [375/3750], Loss: 2.3119
Epoch [1/1], Step [750/3750], Loss: 2.2788
Epoch [1/1], Step [6000/60000], Loss: 0.8718
Epoch [1/1], Step [12000/60000], Loss: 0.0031
Epoch [1/1], Step [18000/60000], Loss: 0.1484
Epoch [1/1], Step [24000/60000], Loss: 0.0001
Epoch [1/1], Step [30000/60000], Loss: 0.0170
Epoch [1/1], Step [36000/60000], Loss: 0.0076
Epoch [1/1], Step [42000/60000], Loss: 0.0002
Epoch [1/1], Step [48000/60000], Loss: 0.0003
Epoch [1/1], Step [54000/60000], Loss: 0.0044
Epoch [1/1], Step [60000/60000], Loss: 0.0171
pytorch cnn model training accuracy: 97%
```

scenerec 데이터 input_size : 128 epoch:150

```
Epoch [150/150], Step [1350/1500], Loss: 0.0000
Epoch [150/150], Step [1500/1500], Loss: 0.0001
pytorch cnn model training accuracy: 35%
```