FC 과제

- STL10 데이터셋을 이용하여 MLP로 이미지 클래스 분류 모델 구축하기

MI LAB

한승민

어떤 관점으로 접근했는가?

- hyperparameter를 구하기 위해 한 것들
- 학습과 모델의 성능에 영향을 주는 Activation Function / learning rate / Loss function 값들을 각각 지정해봄
- 학습시키고, evaluation을 하면서 Loss를 찍어봄.
- loss가 낮아지는 추세가 계속 될 때까지 학습
- loss가 낮아지다가 다시 올라가는 구간에서 다른 parameter설정과의 비교 및 원인 분석
- 다른 parameter설정과의 비교

• # 1차 버전 및 1차 버전에서 반영한 피드백

Train Loss가 점점 줄어들었지만, val loss는 늘어났음. STL에서 validation 데이터를 제공하지 않아서 학습이 잘 진행되고 있는지 확인하기 위해서 각 epoch마다 train loss, validation loss를 찍어보았음.

epoch가 반복함에 따라 모델이 학습되지 않은 데이터를 평가할 때는 loss가 올라가고 있는 것을 알 수 있었음.
Train_loss가 떨어지길래 성능이 마냥 좋아지는 중이라고 생각하였으나, 조우성님이 오해하고 있는 부분을 정정해주었음. Train_loss는 학습한 데이터에 대하여 예측 오차이기 때문에 학습을 지속할수록 무조건 내려갈 거라고 하였음. 맞는 말인 게, 우리가 실제로 사용하고자 하는 것은 학습하지 않았던 새로운 데이터에 대한 성능 평가였기에, val_loss가 낮아져야 했음. 이를 통해 Train_data의 일부에서 validation 데이터셋을 따로 두어(split) 학습하면 val_loss를 줄일 수 있을 거라 생각하였음.

그래서 train 데이터 중 일부를 validation데이터로 분리(데이터 비중은 최소 0.1~ 최대 0.2)하고 각각의 학습마다 validation 거쳐서 오버피팅 막아보고자 했음.(비중에 따른 cross validation을 적용하여 각각의 값을 비교하고 우수한 것을 고르는 비교하여 최적의 k값을 고를 수 있겠는데 이 과정은 하더라도 다음 step에서 진행하는 게 맞지 않을까라는 생각을 했음. 다른 파라미터에 비해 우선순위가 밀려날 거라고 생각했고, 실험 초기에서는 영향력의 비중이 그만큼 적다고 생각했음)
우선 위 과정만 해도 val_loss에서 오버피팅은 줄어들 것 같았음

Accuracy : about 36%

- # 2차 버전 및 2차 버전에서 반영한 피드백
- 1. 데이터 수의 증가를 위하여 이미지를 가로로 뒤집거나 회전시킴

```
transforms.RandomHorizontalFlip(),  # 50% 확률로 가로 뒤집기
transforms.RandomRotation(10),      # ±10도 회전
```

- 2. 섣부른 일반화 및 오버피팅 방지를 위해 노드 dropout 적용해봄(0.3)

```
# Hidden layers 생성
for hidden_size in hidden_sizes:
    layers.append(nn.Linear(previous_size, hidden_size))  # Linear layer
    layers.append(nn.BatchNorm1d(hidden_size))  # 배치 정규화
    layers.append(activation_function())  # Activation function

    layers.append(nn.Dropout(0.3))  # 드롭아웃

    previous_size = hidden_size
```

- 3. 이전의 방법에서 validation_loss가 점점 증가하던 것을 막고자 training data의 일부(0.1~0.2)를 validation data로 사용하였음.
- 4. (몇번 학습해보니 loss가 감소하는 추세를 보이길래)hidden_unit의 수를 늘렸음 / hidden_layer의 수를 4개로 늘렸음

설정 값 : hidden_size = 256 / hidden layer = 4 / lr = 0.0005 / 100 epoch

Accuracy: 36% → 43.01%로 향상

3차 버젼 및 3차 버젼에서 반영한 피드백

1. hidden layer 3으로 조정 Hidden unit : 256 --> 128 / lr=0.001
2. 학습 중 Learning Rate Scheduler를 사용해 점진적으로 감소시킴
3. Activation function을 ReLU말고 leakyReLU를 사용해 봄. ReLU에 비해 음수의 값도 반영하는 특성이, 오버피팅을 방지해주지 않을까?라는 단순한 접근에서 출발하여 시도해보았고, accuracy가 올랐음.
4. train loss는 에포크 늘어나면 무조건 줄으니까 train loss가 줄어든다고 해서 좋은 게 아니고 val loss를 잘 보자. 그리고 early stopping 추가하여 이상한 길로 가는 것 같으면 학습을 중단하자.
5. 우성님 지원님 피드백

파라미터가 너무 적음 27,648(96x96x3)에서 256(hiddenUnit 설정 값)로 추출하고 있는데, 이는 시작부터 고된 추출일수도 있음을 인지하기. 그래서 모델의 파라미터 규모를 키우는 방법도 생각해 보라고 함.

설정 값 :
hidden_size = 256 / hidden layer = 3 / lr = 0.001 / 100
epoch / Act functino : LeakyReLU / Dropout : 0.3 /
scheduler = StepLR(optimizer, step_size=20, gamma=0.5)
Accuracy: 43.01% → 48.89% 향상

- # 4차 버전 및 4차 버전에서 반영한 피드백

1. Hidden unit : 2048 -> 512 -> 256(hidden layer== 3) (우성님 피드백 반영)

```
hidden_sizes = [2048, 512, 256]  # 점진적으로 크기 줄이도록 수정 / hidden layer의 개수는 hiddensize.length와 같음
```

2. Val_loss가 지그재그로 형태로, 수렴하지 않는 것을 보고 lr를 낮춰보는 것 시도

3. Color Jitter , Random Crop 추가하여 **데이터 증강** (우성님 피드백 반영)

```
transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),  # 밝기, 대비, 채도, 색조 +- 20%로 조정
```

4. LeakyReLU → ReLU 변경(변경 이유는 설명 못 하겠음. 그냥 시도해봄.....)

5. Grid Search 방법으로 구간 내에서 여러 조합 중 가장 높은 accuracy를 보이는 최적의 조합을 고르는 방법을 적용해볼 수 있을 것 같긴 함.('시도 가능성'이라는 뉘앙스로 말하는 이유는 과제 기한과 모델 학습 소요시간으로 인해 trial이 즉각즉각 나오지 않기 때문에 피드백 또한 늦어졌다는 것임)

6. 48.89% → 41.76%로 감소함. 최적의 결과가 나오진 않았음(과제 제출 시점까지 원인 규명을 하지 못 함)

```
1 # 모델 평가
2 accuracy = evaluate_model(model, test_loader)
```

```
Accuracy: 41.76%
```

최적의 성능을 기록한 결과의 셋팅값 및 결과

버젼 : 3

셋팅 값

- 데이터 증강
  - 50% 확률로 가로 뒤집기
  - 플마 10도 회전
- Training data의 0.2 비중을 Validation Set으로 나누어 모델 성능을 지속적으로 평가
- hidden_size = 256
- n_layers = 3
- lr = 0.001 - 매 20 에포크마다 학습률을 반으로 줄임
- Loss Function : Cross Entropy
- Optimizer = Adam

```
Epoch [1/100], Train Loss: 1.7892, Val Loss: 1.7173
Epoch [2/100], Train Loss: 1.7106, Val Loss: 1.6819
Epoch [3/100], Train Loss: 1.6651, Val Loss: 1.6801
Epoch [4/100], Train Loss: 1.6203, Val Loss: 1.6594
Epoch [5/100], Train Loss: 1.5772, Val Loss: 1.6559
Epoch [6/100], Train Loss: 1.5592, Val Loss: 1.6596
Epoch [7/100], Train Loss: 1.5455, Val Loss: 1.6197
Epoch [8/100], Train Loss: 1.4927, Val Loss: 1.5988
Epoch [9/100], Train Loss: 1.4681, Val Loss: 1.5952
Epoch [10/100], Train Loss: 1.4493, Val Loss: 1.5816
...
Epoch [40/100], Train Loss: 1.0146, Val Loss: 1.5172
Epoch [41/100], Train Loss: 0.9620, Val Loss: 1.5493
Epoch [42/100], Train Loss: 0.9529, Val Loss: 1.5786
Epoch [43/100], Train Loss: 0.9278, Val Loss: 1.5909
Epoch [44/100], Train Loss: 0.9121, Val Loss: 1.5595
Epoch [45/100], Train Loss: 0.9215, Val Loss: 1.5751
Epoch [46/100], Train Loss: 0.9029, Val Loss: 1.5931
Epoch [47/100], Train Loss: 0.9142, Val Loss: 1.5945
Epoch [48/100], Train Loss: 0.9132, Val Loss: 1.5539
Epoch [49/100], Train Loss: 0.8919, Val Loss: 1.5958
Epoch [50/100], Train Loss: 0.8889, Val Loss: 1.5792
...
Epoch [90/100], Train Loss: 0.7147, Val Loss: 1.6714
Epoch [91/100], Train Loss: 0.7352, Val Loss: 1.6740
Epoch [92/100], Train Loss: 0.7110, Val Loss: 1.6892
Epoch [93/100], Train Loss: 0.6957, Val Loss: 1.6910
Epoch [94/100], Train Loss: 0.7100, Val Loss: 1.6760
Epoch [95/100], Train Loss: 0.6853, Val Loss: 1.7078
Epoch [96/100], Train Loss: 0.7025, Val Loss: 1.6840
Epoch [97/100], Train Loss: 0.7022, Val Loss: 1.7023
Epoch [98/100], Train Loss: 0.7206, Val Loss: 1.6722
Epoch [99/100], Train Loss: 0.7012, Val Loss: 1.6817
Epoch [100/100], Train Loss: 0.7084, Val Loss: 1.6786
```

```
[ ]  1 # 모델 평가
     2 accuracy = evaluate_model(model, test_loader)
```

```
Accuracy: 48.89%
```

느낀점

- 성능이 좋아졌다 한들 왜 / 어떻게 좋아졌는지 설명하기가 참 어려웠다.(그저 설정값의 변화만 나열할 수 있을 뿐..)
- 모델 학습에 관한 내부 동작의 이해가 중요하다는 것을 느꼈다.
  - 아무리 블랙박스의 방식이라해도 Val_loss가 올라가는 것을 보고 학습 과정에서 오버피팅이 계속 되고 있다는 것은 알 수 있었으나, 무엇 때문인지, 코드 어디에서 발생하는 건지 모르니 참 답답했다.
- 학습이 지속되는 것을 방지하기 위하여 Early Stopping은 참 효율적인 것 같다.
- N-fold cross validation과 같은 방법을 배운 적이 있으나 막상 적용에 시도하려고 하니 쉽지 않았다.
- Colab 한 계정에서 여러 세션으로 GPU를 사용하고 싶었다.
- 실험에서 좋은 성과를 내기엔 아직은 이른 것 같다. 개념에 대한 이해가 매우 절실했다.