

보고서 및 논문 윤리 서약

1. 나는 보고서 및 논문의 내용을 조작하지 않겠습니다.
2. 나는 다른 사람의 보고서 및 논문의 내용을 내 것처럼 무단으로 복사하지 않겠습니다.
3. 나는 다른 사람의 보고서 및 논문의 내용을 참고하거나 인용할 시 참고 및 인용 형식을 갖추고 출처를 반드시 밝히겠습니다.
4. 나는 보고서 및 논문을 대신하여 작성하도록 청탁하지도 청탁받지도 않겠습니다.

나는 보고서 및 논문 작성 시 위법 행위를 하지 않고, 명지인으로서 또한 공학인으로서 나의 양심과 명예를 지킬 것을 약속합니다.

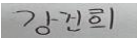


학 과 : 데이터 사이언스 전공

과 목 : 딥러닝

담당교수 : 오 민 식 교수님

학 번 : 60192328

이 름 : 강 건 희 

10000 epoch training 하고 10 epoch 마다 train, test data의 loss를 기록하여 plot으로
그리고 1000 epoch 단위로 train / test accuracy 출

력 하여 코드 / 결과 캡처 (60)

-모델및파라미터설정

```
#LogisticRegression Model
import torch

class LogisticRegression(torch.nn.Module):
    def __init__(self, input_dim, output_dim): #클래스 생성자로, 객체가 생성될 때 자동으로 호출
        super(LogisticRegression, self).__init__()
        self.linear = torch.nn.Linear(input_dim, output_dim)
    def forward(self, x):
        outputs = torch.sigmoid(self.linear(x))
        return outputs

#hyper parameters
epochs = 10000
input_dim = 784
output_dim = 10
lr = 0.01
model = LogisticRegression(input_dim, output_dim)
model = model.to(device)
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=lr)
```

-Accuracy 및 loss 기록

```
for i in range(epochs):
    model.train()
    optimizer.zero_grad()
    output = model(X_train)
    loss = criterion(output, y_train.long())
    l2_lambda = 0.001
    l2_norm = sum(p.pow(2.0).sum() for p in model.parameters())
    loss = loss + l2_lambda * l2_norm
    loss.backward()
    optimizer.step()
    scheduler.step()

    if i % 10 == 0:
        # Train Loss 저장
        loss_save_arr.append(loss.data)
        model.eval()
        with torch.no_grad():
            output_test = model(X_test)
            loss_test = criterion(output_test, y_test.long())
            loss_save_arr2.append(loss_test.data)

    if i % 1000 == 0:
        print("=====")
        print('epoch', i)
        print('loss', loss.data)
        _, pred = torch.max(output.data, axis=1)
        print("train_accuracy {:.0.3f}".format(float((pred == y_train).sum()) / y_train.size(0)))

        _, pred = torch.max(output_test.data, axis=1)
        print("test_accuracy {:.0.3f}".format(float((pred == y_test).sum()) / y_test.size(0)))
```

-Loss 시각화

```
import matplotlib.pyplot as plt
%matplotlib inline

plt.plot(range(1000), [e.to("cpu") for e in loss_save_arr], label='train_Loss')

plt.plot(range(1000), [e.to("cpu") for e in loss_save_arr2], label='test_loss')

plt.title('Loss Comparison')
plt.xlabel('Epochs')
plt.ylabel('Loss Value')

plt.legend()

plt.grid(True)

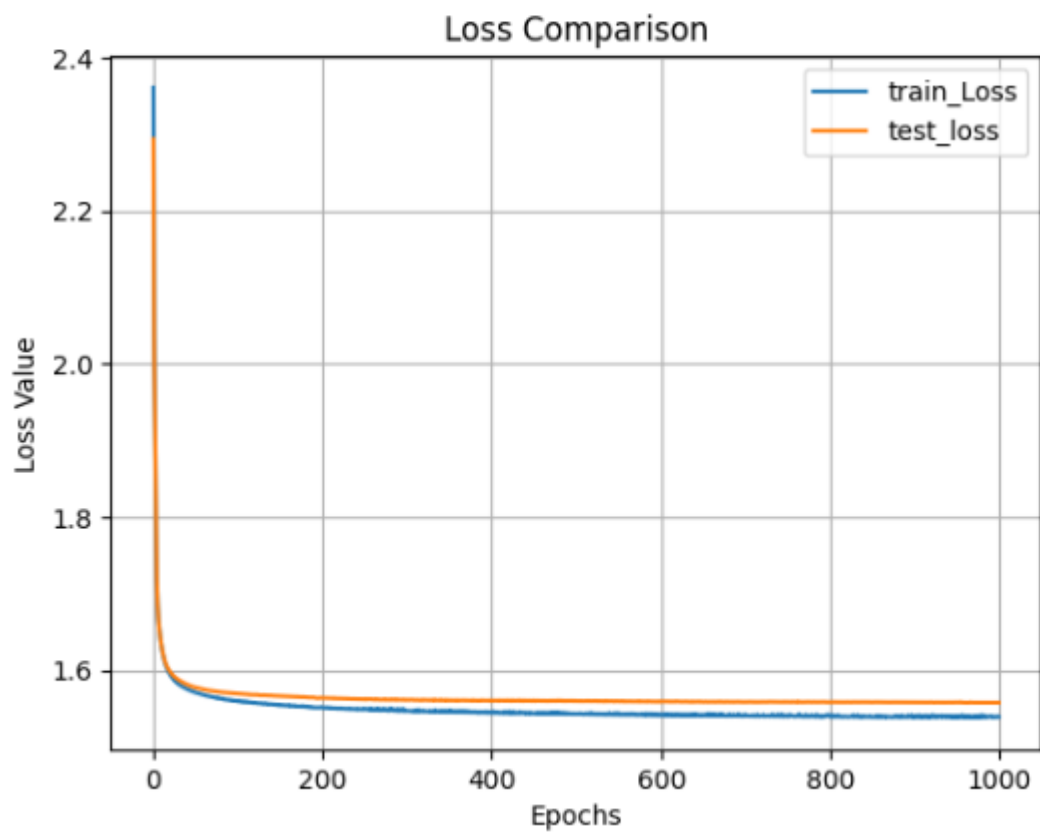
plt.show()
```

-결과



```
=====
epoch 0
loss tensor(2.3606, device='cuda:0')
train_accuracy 0.099
test_accuracy 0.114
=====
epoch 1000
loss tensor(1.5593, device='cuda:0')
train_accuracy 0.890
test_accuracy 0.889
=====
epoch 2000
loss tensor(1.5510, device='cuda:0')
train_accuracy 0.898
test_accuracy 0.896
=====
epoch 3000
loss tensor(1.5470, device='cuda:0')
train_accuracy 0.901
test_accuracy 0.900
=====
epoch 4000
loss tensor(1.5442, device='cuda:0')
train_accuracy 0.902
test_accuracy 0.900
=====
epoch 5000
loss tensor(1.5428, device='cuda:0')
train_accuracy 0.903
test_accuracy 0.900
=====
epoch 6000
loss tensor(1.5426, device='cuda:0')
train_accuracy 0.904
test_accuracy 0.901
=====
epoch 7000
loss tensor(1.5403, device='cuda:0')
train_accuracy 0.905
test_accuracy 0.901
=====
```

```
epoch 7000  
loss tensor(1.5403, device='cuda:0')  
train_accuracy 0.905  
test_accuracy 0.901  
=====  
epoch 8000  
loss tensor(1.5397, device='cuda:0')  
train_accuracy 0.906  
test_accuracy 0.901  
=====  
epoch 9000  
loss tensor(1.5389, device='cuda:0')  
train_accuracy 0.906  
test_accuracy 0.902
```



Accuracy를 올리기 위해서 무엇을 추가 할 수 있을지 한줄로 의견과 근거를 작성하고 실제로 코드에 반영 했을 때 결과가 어떨지 확인 (40)

CrossEntropyLoss

```
CLASS torch.nn.CrossEntropyLoss(weight=None, size_average=None, ignore_index=-100, reduce=None,
reduction='mean', label_smoothing=0.0) [SOURCE]
```

This criterion computes the cross entropy loss between input logits and target.

It is useful when training a classification problem with C classes. If provided, the optional argument `weight` should be a 1D `Tensor` assigning weight to each of the classes. This is particularly useful when you have an unbalanced training set.

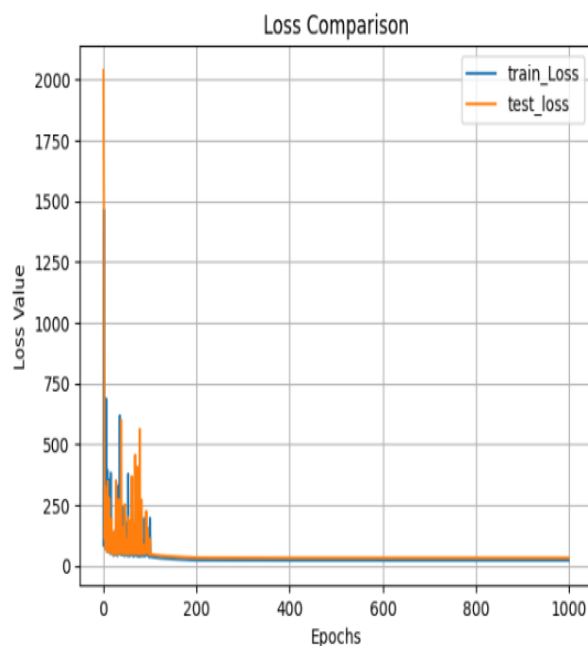
The input is expected to contain the **unnormalized logits for each class** (which do not need to be positive or sum to 1, in

처음 코드는 criterion으로 `torch.nn.CrossEntropyLoss`를 사용하는데 `CrossEntropyLoss`는 위 사진의 문장처럼 정규화 되지 않은 logit값을 input으로 받는 것이 좋다.

따라서 우리가 모델설계 시 `outputs = sigmoid(self.linea(x))` 부분에 `sigmoid`를 빼는 것이 스케일링 하지 않은 logit값을 줄 수 있기 때문에 이를 코드에 적용 해서 아래와 같은 결과를 얻었다. -> 0.01에 accuracy가 오름

```
epoch 0
loss tensor(84.1990, device='cuda:0')
train_accuracy 0.093
test_accuracy 0.658
=====
epoch 1000
loss tensor(195.4857, device='cuda:0')
train_accuracy 0.710
test_accuracy 0.805
=====
epoch 2000
loss tensor(22.6272, device='cuda:0')
train_accuracy 0.927
test_accuracy 0.911
=====
epoch 3000
loss tensor(21.8913, device='cuda:0')
train_accuracy 0.926
test_accuracy 0.911
=====
epoch 4000
loss tensor(21.8200, device='cuda:0')
train_accuracy 0.926
test_accuracy 0.911
=====
epoch 5000
loss tensor(21.8129, device='cuda:0')
train_accuracy 0.926
test_accuracy 0.911
=====
epoch 6000
loss tensor(21.8126, device='cuda:0')
train_accuracy 0.926
test_accuracy 0.911
=====
```

```
epoch 7000
loss tensor(21.8126, device='cuda:0')
train_accuracy 0.926
test_accuracy 0.911
=====
epoch 8000
loss tensor(21.8126, device='cuda:0')
train_accuracy 0.926
test_accuracy 0.911
=====
epoch 9000
loss tensor(21.8126, device='cuda:0')
train_accuracy 0.926
test_accuracy 0.911
```



Appendix에서 model을 분리 했을 때 sigmoid가 없는데 위 코드가 맞는 코드인가 틀린 코드인가? 에 대한 답을 작성 (추가 점수: 20)

CrossEntropyLoss

```
CLASS torch.nn.CrossEntropyLoss(weight=None, size_average=None, ignore_index=-100, reduce=None,
    reduction='mean', label_smoothing=0.0) [SOURCE]
```

This criterion computes the cross entropy loss between input logits and target.

It is useful when training a classification problem with C classes. If provided, the optional argument `weight` should be a 1D *Tensor* assigning weight to each of the classes. This is particularly useful when you have an unbalanced training set.

The *input* is expected to contain the **unnormalized logits for each class** (which do *not* need to be positive or sum to 1, in

2번에서 언급된 사진속 문장처럼 정규화 되지 않은 logit값을 input으로 넣는게 맞으므로 틀린 코드에 속한다고 생각한다.