2주차 복습 과제

Batch size ?

❓ 입력값 x를 평균 0, 분산 1로 표준화하여 activation function로 전달하여 출력값의 분포를 고르게 한다. x를 표준화 하는 과정에서 batch size 단위로 평균과 분산값을 계산하기 때문에, 어떤 batch size를 선택하는지는 중요하다.

$$\widehat{x} = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}}$$

1) batch size를 크게 하는 경우

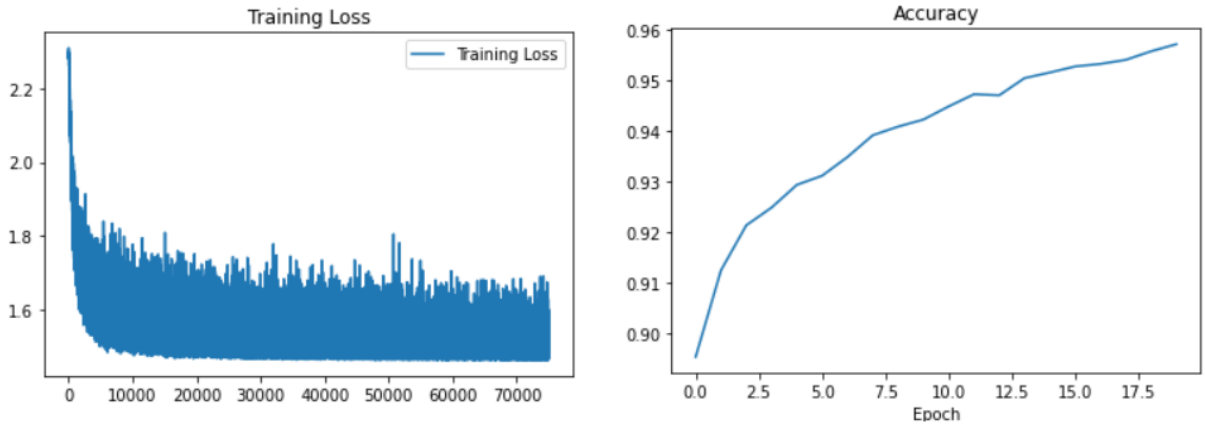
- training set 분포를 좀 더 정확히 추정한다. 이 말은 즉, noise를 감소시켜 모델 수렴을 향상 시킬 수 있다는 것이다. -> 학습 속도 향상

2) batch size를 작게 하는 경우

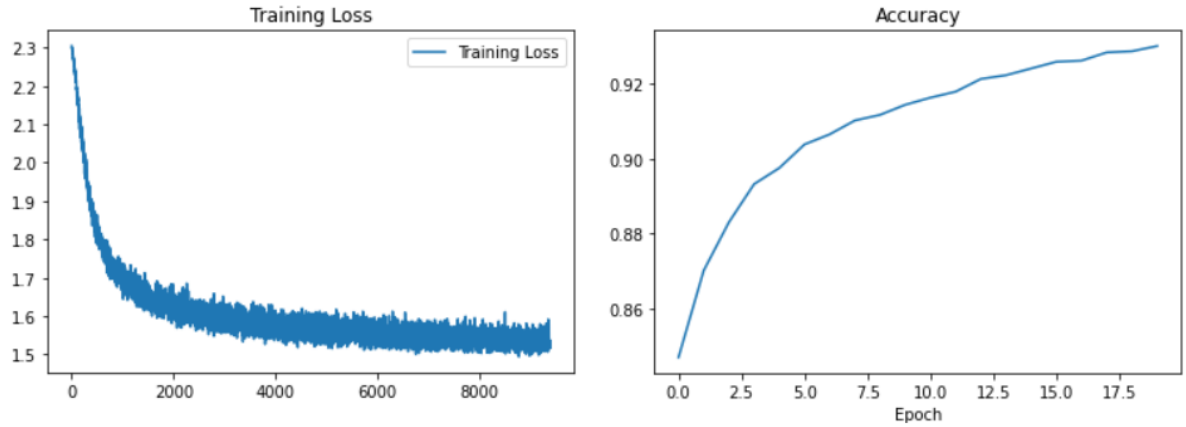
- sample의 수가 적어 training set 통계갓을 부정확하게 계산한다. noise가 많아져서 모델의 수렴이 불안정해진다. 하지만 regularization의 효과는 강해진다.

!! Rethinking ‘Batch’ in BatchNorm 논문에서는 보편적으로 32~128 사이의 batch size가 적절하다고 말한다.

Batch size 16 / ephoc 20 / lr 1e-4



Batch size 128 / ephoc 20 / lr 1e-4

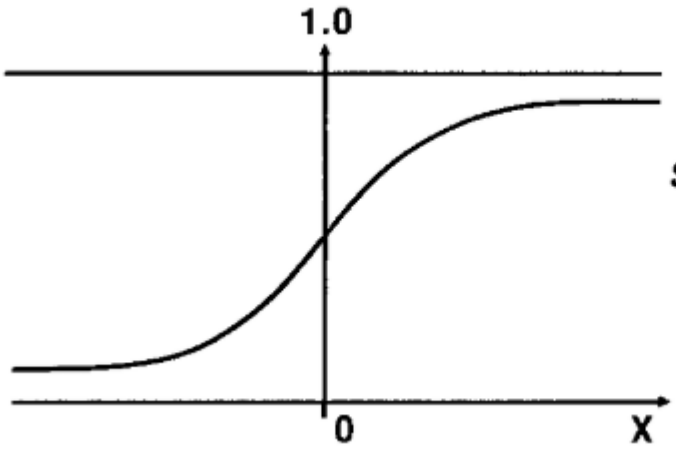


Batch size를 늘리니, training loss의 수렴 속도가 증가하는 것을 확실히 볼 수 있다.

Activation Function ?

활성화 함수의 종류는 무척 많다. 몇 가지만 얘기해보자.

- Sigmoid 함수

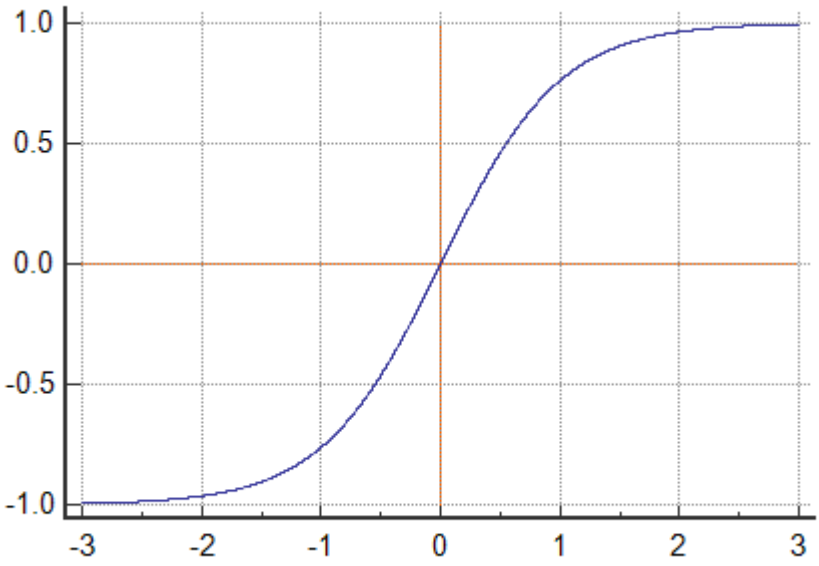


Sigmoid

$$f(x) = \frac{1}{1+e^{-x}}$$

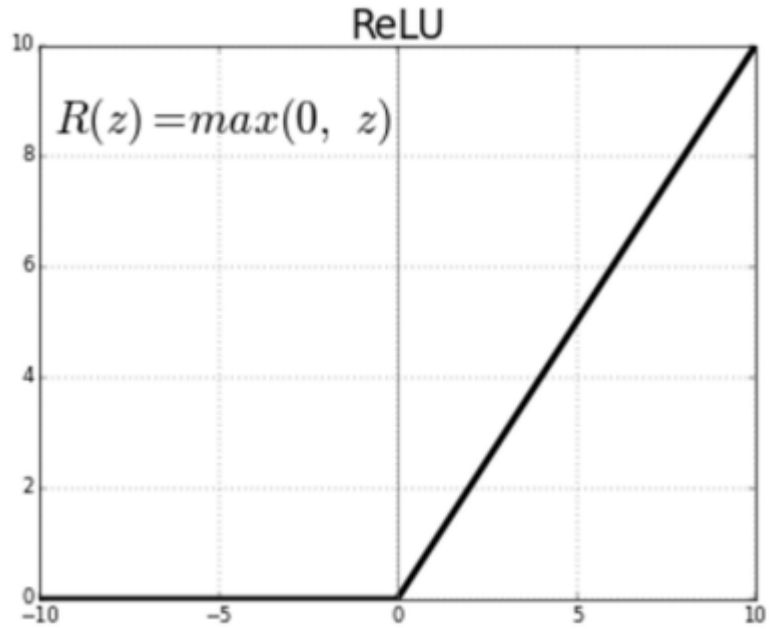
엄청 큰 데이터도 0과 1 사이로 변환시킨다. 딥러닝은 gradient를 이용해서 값을 찾아나가는데 층이 많아지면 점점 값이 0에 수렴되는 문제가 발생해서 성능이 떨어진다. 이를 기울기 손실(vanishing gradient problem)문제라고 하며, 이것을 발생시키는 것이 sigmoid 함수의 대표적인 단점이다.

- Tanh 함수



-1~1 사이의 값으로 계산 / Sigmoid function과 한계점은 동일

- ReLU 함수



딥러닝에서 가장 많이 사용되는 활성화 함수라고 할 수 있다. f(x) = max(0, x) → 매우 간단한 수식을 가니다. 0 이하는 0으로, 0값을 초과하면 그대로. 단순한 구조이다.

0과 1사이로 데이터를 꾸겨넣지 않기 때문에 위 함수들에서 발생한 vanishing gradient problem이 발생하지 않는다. 그래서 레이어가 많아져도 원활한 학습이 가능하고, 학습 속도도 매우 뛰어나다. 또한 ReLU function은 세개의 명령으로 충분하지만, sigmoid는 넘 많음.

- Softmax 함수

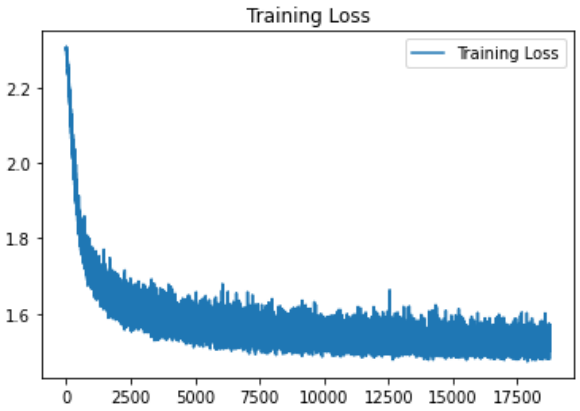
그래프가 존재하지 않는 함수이다. 확률값으로 변경하는 함수. / 다 더해서 1

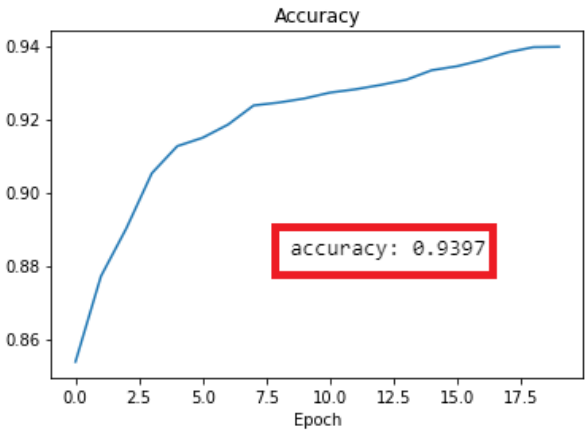
Layer 쌓아보기 (batch size : 32 / epoch:20 고정)

```
# 기본 Layer 구조
class net(nn.Module):
    def __init__(self):
        super(net, self).__init__()

        self.fc = nn.Sequential(
            nn.Linear(28 * 28, 64),
            nn.ReLU(),
            nn.Linear(64, 10),
            nn.Sigmoid()
           )
```

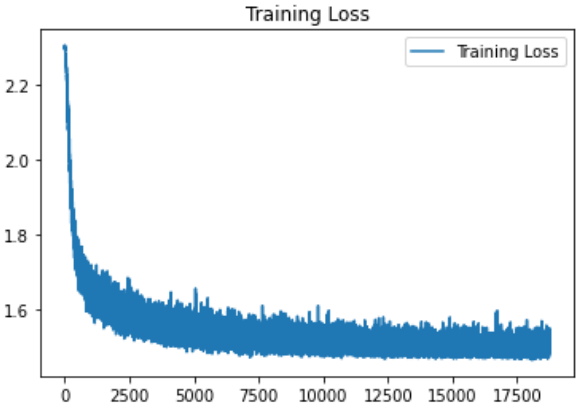
```
    def forward(self, x):
        x = x.view(-1, 28 * 28)
        return self.fc(x)
```

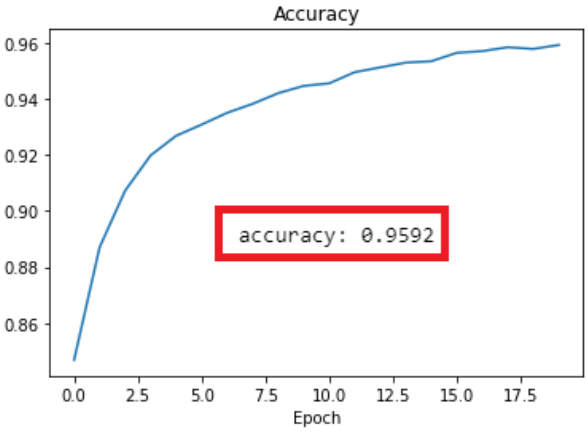




한층 쌓기

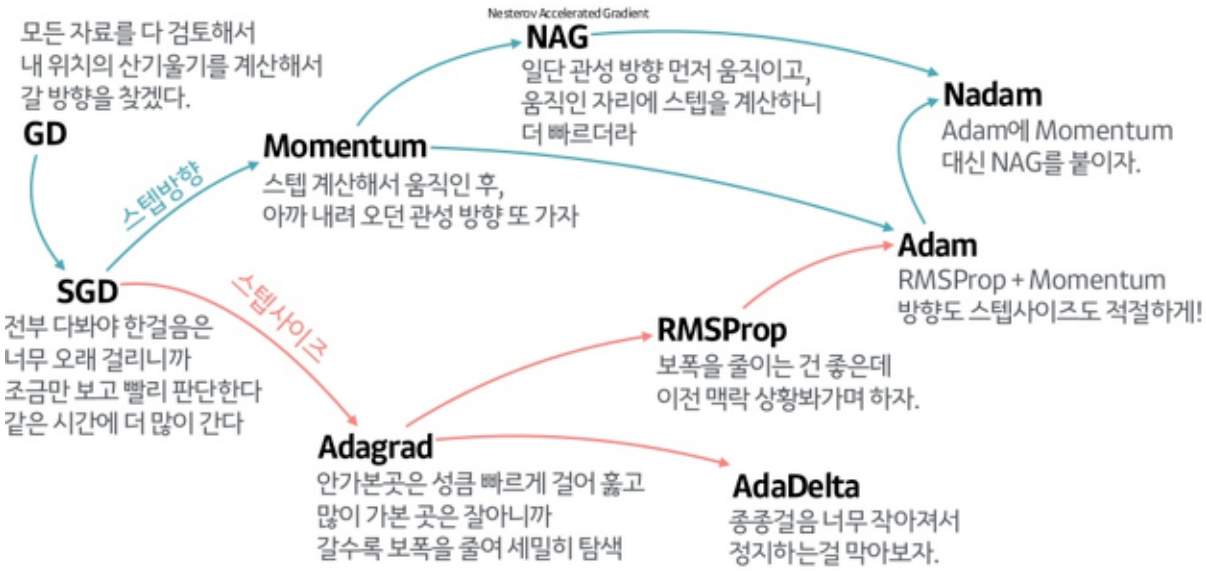
```
self.fc = nn.Sequential(
            nn.Linear(28 * 28, 128),
            nn.ReLU(),
            nn.Linear(128, 64)
            nn.Linear(64, 10),
            nn.Sigmoid()
           )
```





여기서 epoch을 더 주면 accuracy는 더 증가할 것이다.

Optimizer List



- Gradient Descent
- SGD
- Momentum
- NAG
- Adam
- AdaGrad
- RMSProp
- AdaMax
- Nadam

각각의 optimizer에 대한 구글링 자료가 무척 많다. 틈틈히 참고해서 상기시키면 좋을 것 같다.