DNN: 뉴럴 네트워크를 깊게 쌓는 것.

하나의 Linear regression unit을 여러 개 모아서 여러 레이어를 만들 수 있다.

= 레이어를 깊게, 넓게 쌓기.

어떻게 좋은 성능을 낼 수 있는 네트워크를 설계할 수 있을까?

깊게 쌓는 방법에 대해서 배운다.

DNN 모델 학습을 위한 팁

1. 활성 함수를 잘 선택한다. • ReLU가 가장 널리 사용된다.
2. 가중치 초기화 방법을 잘 선택한다. • Xavier가 가장 널리 사용된다
3. 드랍 아웃을 잘 적용한다. • “NN-ReLU-Dropout”을 하나의 블락으로 쌓는다.
4. BN을 잘 적용한다. • “NN-ReLU-BN”을 하나의 블락으로 쌓는다.

- 2개 이상의 레이어를 쌓아서 XOR문제를 해결하였다.

- 이후 DNN을 이용해서 너무 깊게 쌓으니 학습이 잘 이루어지지 않는 문제가 발생하였다.

Backpropagation의 gradient vanishing 문제가 발생.

Backpropagation은 Error를 가중치 업데이트에 사용하게 되는데

Layer가 너무 깊으니까 gradient가 마지막 layer까지 가지 못하고 중간에서 0이 되어버렸다.

그래서 w와 b를 더 이상 갱신할 수 없고,

Weight가 갱신되지 않으므로 학습이 이루어지지 않게 되었다.

3p 학습이 잘 되지 않는 문제와 그에 대한 해결법

1. 잘못된 비선형함수(활성화함수)를 사용하고 있었다.
2. 좋지 않은 weight 초기화함수를 사용하였다.

# Activation function

미분이 가능해야 활성함수로 사용 가능하다.

weight갱신 시, 각 노드 위치에서의 편미분 값을 사용하므로

1. ~~sigmoid함수~~

: activation function의 대표적 함수

- 입력을 0 or 1사이로 정규화 한다.

0.5 이상이면 1 다음 뉴런으로 신호를 보낸다.

0.5 이하이면 0 신호를 보내지 않는다.

Boundary가 넓어질 수록 미분 값이 0에 가까워진다. 미분 값이 너무 작다.

그래서 backpropagation을 하면 작은 미분값으로 인해 weight값이 작아져서 vanishing을 야기한다. 여러 개의 layer를 쌓으면 신경망 학습이 잘 되지 않는다.

🡪 3개 이상의 layer(DNN)에서는 활성함수로 사용하지 않는다.

1. ~~Tanh~~

Sigmoid를 보완하고자 제안된 활성 함수

성능이 개선될 확률이 적음

1. ReLU

ReLU가 전반적으로 성능이 좋다고 알려져 있으며, 가장 인기가 많다.

1. Leaky ReLU

ReLU함수의 dying ReLU를 해결하기 위해 기울기를 0이 아닌 0.01을 사용한다.

dying ReLU: x<0 이면 모두 0을 출력

1. ELU

dying ReLU 문제를 해결함

출력값이 거의 zero-centered에 가깝다.

= Layer를 통과하는 데이터들이 정규화가 잘 되어서 좋은 성능을 보인다.

1. Maxout3.

# 2. Weight Initialization "Initialize weights in a smart way”

\* Random initialization이 다르기 때문에 매번 ReLU를 사용할 때마다 수렴하는 정도가 다르다.

\* W=0이면 Backpropagation 시 gradient 값이 0되어 Gradient Vanishing 현상이 발생하므로

절대 w=0으로 초기화 해서는 안된다.

\* Softmax 같은 알고리즘에서는 -1 ~ 1의 난수를 Weight로 사용

나쁜 방법은 아니지만 깊은 신경망을 설계할 수록 학습이 잘 되지 않는다.

초기화를 위한 방법론

**🡪 노드의 입출력 수에 비례해서 w와 b 초기값을 결정짓는 방법 제안**

Pytorch 초기화 방법: <https://pytorch.org/docs/stable/nn.init.html>

Torch.nn.init 매뉴얼에 다양한 초기화 방법이 있다.

1. Xavie initialization

사용방법 example

w를 3\*5로 생성 후,

w를 넣어 주기 🡪 입력과 출력에 따라 초기화를 해준다.

1. He initialization 26p

🡺 위의 두 방법론을 사용해서 w, b를 초기화 하면 된다.

27p 동일한 모델이지만 초기화를 어떻게 하는지에 따라 성능이 바뀔 수 있다.

수렴하는 값 minimum이 달라질 수 있다.

# Dropout and model Ensemble

## 10주차 복습 **overfitting**

\* 좋은 모델은 학습을 통해 얻은 수학함수가 데이터의 distribution을 잘 표현하고 있는 모델이다.

Overfitting 모델은 Loss=0이 될 수 있지만 새로운 데이터에 대한 대응력이 없어 모델 학습 의미가 상실된다.

\* overfitting을 판단하는 방법

보통 데이터를 랜덤하게 train, test로 나누기 때문에 두 데이터의 경향성은 매우 유사하다

그런데 Train data에 대해 학습이 진행되면서 loss값이 감소하는 반면,

Test data의 에러 값은 감소하지 않고, 어느 지점부터 상승한다.

이는 overfitting이 발생했음을 보여준다.

## **regularization(-dropout)**

overfitting을 피하는 방법 중 하나이다.

Dropout: 신경망에서 유닛을 제거하는 것

모든 노드가 서로 연결 되어있다.

학습 시, 노드의 연결을 끊고 일부 노드의 w만 갱신한다.

🡪 DNN의 과적합 문제가 해결된다.

\* 연결 edge를 얼마나 끊어야 할까? 30%~50%

34p 성능향상이 가능한 이유

정해진 노드를 모든 학습동안 계속 끊는 것이 아니다.

연결을 끊는 정도 p=0.3으로 설정하면

매 학습때마다 랜덤하게 p만큼의 노드를 선택하여 연결을 끊는다.

Torch.nn.Dropout(p=0.5, inplace=False)

P의 default는 0.5