ONECLICK AI

이미지 처리와 CNN

연준모

목차

1. 텐서와 이미지 데이터 텐서가 무엇일까?

텐서란?

데이터를 표현하는 단위

1차원 텐서(벡터):
[1, 2, 3, 4]

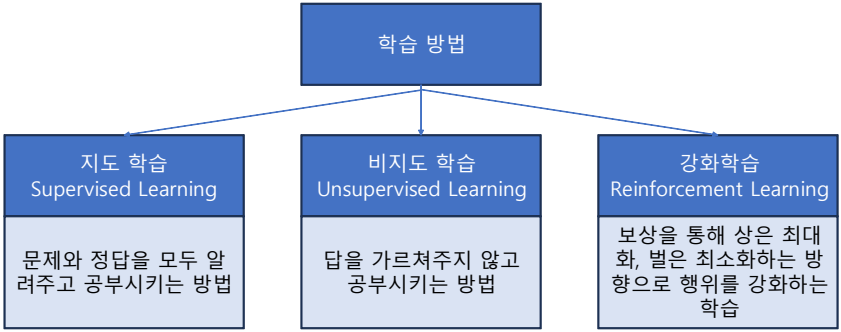
2차원 텐서(행렬):
가로, 세로가 있는 흑백 사진 같은거

3차원 텐서(행렬이 여러개 쌓임):
컬러 사진 (가로, 세로, RGB)

4차원 텐서(3차원 텐서가 여러개 쌓임):
컬러 사진 여러 장을 한 번에 처리하기 위해 묶어 놓는 것

2. 학습의 종류와 현재 시점 학습

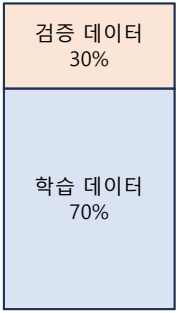
학습방법 3가지



2. 학습의 종류와 현재 시점 학습

지도학습

✓분류랑 회귀가 특징



	분류	회귀
목표	카테고리, 그룹 예측	연속적인 숫자값 예측
출력값	스팸/정상, 개/고양이	15.7억, 255도 등
질문 형태	어느 그룹에 속하는가	얼마나 될 것인가

✓입력 데이터의 일부분을 검증용 데이터로 사용

✓저번주에 한 딥러닝이 지도학습이다

✓예를들어, {x = 1, y = 2}, {x = 26, y = 27}, … 이런 식의 데이터가 10 개 있을 때, 7개로 딥러닝 돌리고, 남은 3개로 점검해서 학습 데이터 제외하고 정확도가 얼마나 되는지 판단한다.

✓이는 과적합이 얼마나 일어났는지 확인할 수 있는 중요한 지표가 된다.

2. 학습의 종류와 현재 시점 학습

비지도학습

✓군집화 (비슷한 것 들을 찾아서 그룹지게 한다)

✓고양이와 강아지 사진을 보여주되, 뭐가 고양이, 강아지 인지는 안 알려 준다

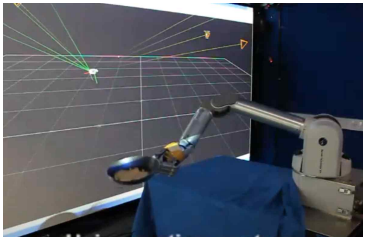
✓만일, 학습을 성공해서 고양이와 강아지 구분에 성공한다면,

✓우리가 생각하는 특징 외의 방법으로(새로운 규칙) 고양이와 강아지를 구분하는 새로운 방법을 알게 될 수도 있다.

=> 새로운 규칙을 얻기 위함

2. 학습의 종류와 현재 시점 학습

강화학습



✓보상을 주는 함수 여러 개를 사용(올바른 동작을 하면 더 높은 점수 준다)

✓모델은 더 높은 점수를 얻기 위해 다양한 시도를 한다

✓처음에, 뒤집는 답을 알려주지 않았다면 저래 안된다

✓지도학습을 약간 시도 했음에도 불구하고 결과까지 나오는데 엄청 오래 걸린다

https://www.youtube.com/watch?v=W_gxLKSsSIE&ab_channel=PetarKormushev

현실은, 지도학습 빼고 전부 사장되었다

3. Overfitting

과적합

훈련 데이터에 존재하는 아주 작은 노이즈나 특이점까지 모두 학습하려는 경향

입력 데이터에 대한 loss 값은 0에 가깝지만, 예상하지 못한 입력 데이터가 들어오면 망가진 결과가 나오는 것 ex) 연습문제만 암기하여 새 유형에 대응하지 못함

모델의 복잡도가 주어진 데이터에 비해 너무 높을 때 발생한다 ex) 파라미터 개수가 너무 많을 경우

+ 은닉층이 많아질 수록 모델의 성능은 무조건 올라가지만, 모델이 학습되는 속도보다 과적합이 발생하는 속도가 더 빨라져서 의미 없다

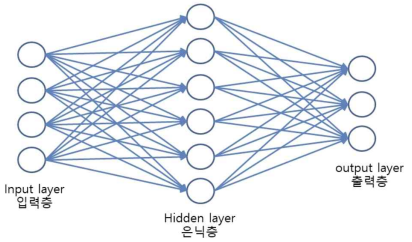
https://gihak111.github.io/ai/2025/09/23/overfitting_upload.html

https://arxiv.org/abs/1706.05394

https://arxiv.org/abs/1912.02292

4. CNN 아키텍쳐 따라잡기 어떻게 생겼을까?

✓ **여러개의 레이어의 조합**



여러 개의 히든 레이어로 구성되어 있다.

히든 레이어
- ✓ ConvLayer : 합성곱 레이어
- ✓ FlattenLayer : 평탄화 레이어
- ✓ DenseLayer : 완전연결 레이어
- ✓ MaxPoolingLayer : 텐서 축소 레이어

활성화함수
- ✓ RelULayer : ReLU 활성화 함수
- ✓ Softmax : 소프트맥스 함수

손실함수
- ✓ Cross_entropy_loss : 교차 엔트로피 손실

4. CNN 아키텍쳐 따라잡기 ConvLayer?

✓ **ConvLayer**

입력 데이터에서 특징을 추출하기 위해 필터를 적용하는 방식

필터(커널) 크기 : F * F => 입력데이터와 합성곱 연산을 수행하는 크기

보폭 : S * S => 필터가 입력 데이터를 순회할 때 한번 이동하는 칸의 크기

패딩 p => 입력데이터의 외곽에 특정값(주로 0)을 추가하는것 모서리 부분 정보 손실을 줄이는 역할

입력 텐서의 크기가 W_in * H_in 이라 할 때, 출력텐서 B는 다음과 같아진다

$$H_{out} = \lfloor \frac{H_{in} - F + 2P}{S} \rfloor + 1$$

$$W_{out} = \lfloor \frac{W_{in} - F + 2P}{S} \rfloor + 1$$

4. CNN 아키텍쳐 따라잡기 ConvLayer?

출력텐서 B의 특정원소 I, j는 다음과 같이 정의된다.

$$B_{i,j} = \sum_{h=0}^{F-1} \sum_{w=0}^{F-1} K_{h,w} \cdot A_{i \cdot S+h, j \cdot S+w}$$

B_i, j : 출력 텐서의 (i, j) 위치의 원소

K_h, w : 필터(커널)의 (h, w) 위치의 가중치

A_x, y : 입력 텐서의 (x, y) 위치의 원소

S : 스트라이드 (Stride)

4. CNN 아키텍쳐 따라잡기 ConvLayer?

뭔 소린지 모르겠으니, 예시를 통해 이해해 보자

입력텐서, 필터를 다음과 같이 하고, S = 1, P = 0 이라 하자

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix} \qquad K = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

1. 첫 번째 위치 연산

$$\begin{pmatrix} 1 & 2 & 3 \\ 5 & 6 & 7 \\ 9 & 10 & 11 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} = (1 \cdot 1) + (2 \cdot 0) + (3 \cdot 1) + (5 \cdot 0) + (6 \cdot 1) \quad \text{... = 30}$$

스트라이크가 1 이므로, 한 칸 이동해서 연산

$$\begin{pmatrix} 2 & 3 & 4 \\ 6 & 7 & 8 \\ 10 & 11 & 12 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} = (2 \cdot 1) + (3 \cdot 0) + (4 \cdot 1) + (6 \cdot 0) \quad \text{... = 35}$$

반복해서 얻은 최종 결과물

$$B = \begin{matrix} [[30,\ 35], \\ [50,\ 55]] \end{matrix}$$

4. CNN 아키텍쳐 따라잡기 MaxPoolingLayer?

✓ MaxPoolingLayer

입력 텐서의 특정 영역에서 가장 큰 값만 추출하는 방식

윈도우를 이용해서 특정 맵을 순회한다.
필터는 크기(pool_size)와 보폭(strides)을 가진다.
이 윈도우의 영역 안의 모든 값에서 가장 큰 값만 선택하여 출력으로 보낸다.
이를 입력 텐서의 모든 영역에 반복적으로 적용, 최종적으로 다운샘플링된 출력 특징맵을 생성한다.

윈도우 크기 : F * F

보폭 : S * S

라 하면, 출력텐서 B의 크기는 =>

$$H_{out} = \lfloor \frac{H_{in}-F}{S} \rfloor + 1$$
$$W_{out} = \lfloor \frac{W_{in}-F}{S} \rfloor + 1$$

출력텐서 B의 특정원소는 다음과 같은 수식으로 정의 된다

=>

$$B_{i,j} = \max_{h,w}(A_{S\cdot i+h, S\cdot j+w})$$

i는 출력 텐서의 행 인덱스 $(0 \leq i < H_{out})$

j는 출력 텐서의 열 인덱스 $(0 \leq j < W_{out})$

h는 필터 영역 내의 행 인덱스 $(0 \leq h < F)$

w는 필터 영역 내의 열 인덱스 $(0 \leq w < F)$

✓ **MaxPoolingLayer**

예시를 통해 알아보자

4 * 4 인 입력텐서 A $A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix}$

최종출력텐서 => $B = \begin{pmatrix} 6 & 8 \\ 14 & 16 \end{pmatrix}$

F = 2

S = 2

이면, 다음과 같은 4 범위가 된다

1 => $A_{0:2,0:2} = \begin{pmatrix} 1 & 2 \\ 5 & 6 \end{pmatrix}$

2 => $A_{0:2,2:4} = \begin{pmatrix} 3 & 4 \\ 7 & 8 \end{pmatrix}$

3 => $A_{2:4,0:2} = \begin{pmatrix} 9 & 10 \\ 13 & 14 \end{pmatrix}$

4 => $A_{2:4,2:4} = \begin{pmatrix} 11 & 12 \\ 15 & 16 \end{pmatrix}$

4. CNN 아키텍쳐 따라잡기 FlattenLayer?

✓ **FlattenLayer**

다차원 텐서를 1차원 벡터로 변환하는데 사용.

학습 가능한 매개면수(가중치, 편향)이 없어 별도의 연산을 수행하지 않는다. 그냥 1차원 벡터로 변형하는게 끝

만일, 입력 텐서의 형태가 (C, H, W) 라면

C : 채널의 수

H : 높이

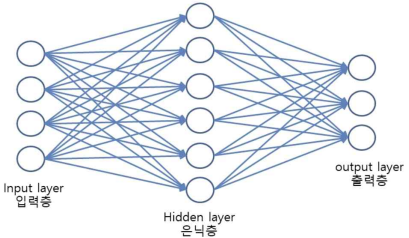
W : 너비

출력 => (C * H * W)

즉, FlattenLayer를 통화한 후의 출력 텐서의 원소 개수는 입력 텐서의 모든 원소 개수의 곱과 같다

4. CNN 아키텍쳐 따라잡기 DenseLayer?

✓ **DenseLayer**



저번 주에 설명한 완전연결 레이어
모든 노드에 대한 연결이 되어 있다.

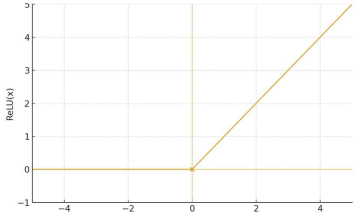
4. CNN 아키텍쳐 따라잡기 ReLULayer?

✓ **ReLULayer** **독립적인 계층으로 모델에 추가될 수 있기 때문에 활성화 함수임에도 불구하고 레이어 이다.**

활성화 함수 중 하나

음수값은 0으로, 양수값은 그대로 유지

역전파가 존재



두 개의 선형이 합쳐져 비선형성을 가진다

비 선형성

여러 개의 ReLU 레이어를 쌓으면

복잡한 패턴을 학습 가능

계산 효율성

복잡한 뎃셈, 뺄셈 없고 단순히 0과 입력값을 비교하는게 전부라 매우 빠르다

기울기 소실 문제 완화

시그모이드는 입력값이 너무 크거나 작을 때 미분값이 0에 가까워 지는 기울기 소실이 발생. 하지만, ReLU는 양수 영역에서 항상 미분값이 1 이므로, 역전파에서 매우 안정

4. CNN 아키텍쳐 따라잡기 RelULayer?

✓ **RelULayer** **역전파에서 미분시**

역전파를 통해 기울기를 전달해야 한다

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{cases}$$

X = 0일 때는 미분 불가능. 따라서, 이런 경우엔 미분값을 0 또는 1로 정의해서 사용한다.

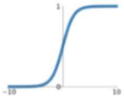
단, 입력값이 0보다 작으면 기울기가 0이 되어, 해당 뉴런이 더 이상 학습되지 않는 죽은 ReLU 문제가 발생할 수 있다.

4. CNN 아키텍쳐 따라잡기 RelULayer?

✓ **왜 시그모이드 안 쓰고 렐루 쓰는 걸까?**

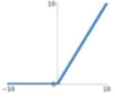
Sigmoid

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



ReLU

$$\max(0, x)$$



시그모이드는 w와 b가 1을 넘지 않는 경우 값이 계속 작아진다

➔이로 인해 기울기 소실이 일어나 기울기가 0이 되는 문제가 발생한다

때문에, CNN처럼 레이어가 5개만 되도 시그모이드 사용하면 기울기 다 0 된다

이래서 ReLU 쓰는 것.

하지만, 렐루도 레이어 6개 되면 또 0 때문에 학습이 안된다 그래서 LeakyReLU 같은거 쓴다

https://gihak111.github.io/ai/2025/09/24/Activation_function_upload.html

4. CNN 아키텍쳐 따라잡기 Softmax?

✓ **Softmax** **신경망의 출력값을 확률분포로 (0과 1 사이) 변환하는데 사용(출력층)**

입력백터가 다음과 같을 때, $z = (z_1, z_2, z_3, \dots)$

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

e : 자연상수

z_i : 입력백터 z 의 i 번째 원소

$\sum_{j=1}^{K} e^{z_j}$: 입력백터의 모든 원소에 대해 e의 거듭제곱을 취한 값들의 총합

모든 출력값의 합이 1이 되도록 하는 정규화 역할 한다

4. CNN 아키텍쳐 따라잡기 Softmax?

✓ Softmax

3개의 클래스를 분류하는 모델의 마지막 층이 다음과 같다고 해 보자

$$z = (1.5,\ 2.5,\ 0.5)$$

1. 각 원소에 e의 거듭제곱을 취한다

$$e^{1.5} \approx 4.48, \quad e^{2.5} \approx 12.18, \quad e^{0.5} \approx 1.65$$

2. 모든 지수함수의 결과값을 더한다

$$4.48 + 12.18 + 1.65 = 18.31$$

3. 각 지수함수 결과를 함계로 나눈다

4.18/18.31 = 0.24

12.18/18.31 = 0.66

1.65/18.31 = 0.09 이렇게, 최종 출력을 전부 다 더하면 1이 된다

4. CNN 아키텍쳐 따라잡기 one hot encoding?

데이터 전처리 시 정답 레이블을 숫자로 변환하는데 사용

이를 위한 one – hot encoding

범주형 데이터를 컴퓨터가 이해할 수 있는 숫자 백터로 변환

포인트가, 각 카테고리는 서로 연관이 없고 독립적임을 강조 (동등하며, 순서나 크기가 없다)

예를 들어, 사과 = 1, 배 = 2, 수박 = 3 이라 하면, 수박 > 배 > 사과 로 이해할 수 도 있다

따라서, 사과 = [1, 0, 0], 배 = [0, 1, 0], 수박 = [0, 0, 1] 로 한다

방법은, 원하는 카테고리만큼 정의한 후 속하는 카테고리를 1로, 그 외는 전부 0으로 하면 된다

✓ **Cross_entropy_loss**

두 확률분포 사이의 차이를 측정하는 방법

예측분포(Q)가 실제분포(P)와 얼마나 다른지를 나타낸다.

2진분류인 경우와, 다중 클래스 분류에서 사용하는 수식이 다르다는 특징이 있다.

2진일 경우

$$\text{Loss} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$$

N: 전체 데이터의 수

y_i: i번째 데이터의 실제 레이블 (0 또는 1)

$\hat{y}\_i$: i번째 데이터에 대해 모델이 예측한 확률 (클래스 1일 확률)

4. CNN 아키텍쳐 따라잡기 Cross_entropy_loss?

M진일 경우

$$\text{Loss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log(\hat{y}_{ij})$$

N: 전체 데이터의 수

C: 전체 클래스의 수

y_ij: i번째 데이터가 클래스 j에 속하면 1, 아니면 0 (원-핫 인코딩된 실제 값)

$\hat{y}\_ij$: i번째 데이터가 클래스 j에 속할 것이라고 모델이 예측한 확률

4. CNN 아키텍쳐 따라잡기 Cross_entropy_loss?

예시를 통해서 알아보자

개, 고양이, 새를 다중 클래스 분류하는 문제가 있다고 하자. 정답이 고양이면

실제 값은 y = [0, 1, 0] 이 되어야 한다

모델이 출력한 값이 y = [0.2, 0.7, 0.1] 이라 했을 때, (출력 레이어 sofrmax)
각 클래스 별로 실제값 * log(예측 확률)을 계산해서 모두 더한 뒤 마지막에 –를 붙이는 것이다

따라서, 개 : 0 * log(0.2) = 0

고양이 : 1 * log(0.7) = -0.3567

새 : 0 * log(0.1) = 0

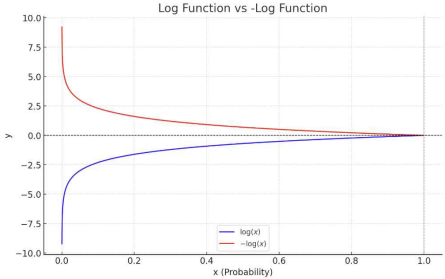
Loss = -1 * -0.3567 = 0.3567 이다

4. CNN 아키텍쳐 따라잡기 Cross_entropy_loss?

이해를 더 돕기 위해

왜 ? Log를 씌우는 걸까?

더 큰 벌을 주기 위함이다



정답을 거의 맞추어 1이 들어오면 loss는 0이 됨

완전히 틀려버려 입력이 이상하면 y 는 무한대로 발산

=> 애매하게 맞히는 것 보다 확실하게 맞히도록 강력하게 유도할 수 있다.

마지막에 – 곱하는 것도 양수로 하여 차이의 량을 보여주기 위함

5. CNN 수식으로 열어보기

1. 다음과 같이 생각하고 시작하자

Op	Depth	Height	Width	filter Height	filter Width
input	1	4	4		
Convolution	1	2	2	3	3
ReLU Layer	1	2	2		
Maxpooling	1	1	1	2	2
Flatten	1	1	1		
Dense	2	1	1		
Softmax	2	1	1		

입력층에서의 Depth는 흑백이라는 의미
그 이후에는 적용된 필터의 개수
Dense 이후에는 출력 뉴런의 개수

마찬가지로 순전파, 역전파, 가중치 업데이트 과정 따른다

원래는, 하나의 ConV레이어가 아닌, ConV -> Maxpool -> ConV -> Maxpool
이런식으로 여러번 반복되지만, 쉬운 계산을 위해 한 번만 넣었다

5. CNN 수식으로 열어보기

1. 다음과 같이 생각하고 시작하자

Op	Depth	Height	Width	filter Height	filter Width
input	1	4	4		
Convolution	1	2	2	3	3
ReLU Layer	1	2	2		
Maxpooling	1	1	1	2	2
Flatten	1	1	1		
Dense	2	1	1		
Softmax	2	1	1		

? 근데 Dense layer에 대한 활성화 함수는 없나요?
전엔 있어야 한다매

⇒Conv Layer에 대한 활성화 함수는 ReLU
이미지 특징 추출 -> ReLU로 비선형성 추가
⇒Dense Layer에 대한 활성화 함수는 Softmax
최종 분류를 위한 점수 계산 -> softmax로 확률로 변환

1. 다음과 같이 생각하고 시작하자

파라미터 설정

Conv 필터(W_conv), 편향(b_conv)

w_{conv} = [[1, 0, 1], [0, 1, 0], [1, 0, 1]]

b_{conv} = 1

Dense 가중치(W_dense), 편향(d_dense)

Flatten 후 입력 크기가 1 이므로 w_{dense} = [[0.5, -0.5]]

b_{dense} = [0.1, 0.2]

실제 정답 레이블 (y_true) : [1, 0]

학습율 : 0.1 Maxpolling 필터 크기: 2 * 2

입력 데이터

[[1, 2, 0, 1],
[3, 1, 1, 5],
[0, 2, 2, 0],
[1, 0, 3, 1]]

5. CNN 수식으로 열어보기 1. 순전파

✓ **입력 데이터가 신경망의 각 층을 통과하며 예측값을 계산하는 과정**

순전파 1단계. Convolution 레이어 계산 입력으로 4 * 4 이미지 들어온다

3 * 3 필터가 4 * 4 이미지를 순회하며 합성곱 연산을 수행한다

$$Z_{conv} = \sum (X_{region} \odot W_{conv}) + b_{conv}$$

솔직히 식만 놓고보면 내가 봐도 뭔 소린지 모르겠다

5. CNN 수식으로 열어보기 1. 순전파

✓ 수식으로 열어보자

순전파 1단계. Convolution 레이어 계산

[[1, 0, 1],
[0, 1, 0],
[1, 0, 1]]

[[1, 2, 0, 1],
[3, 1, 1, 5],
[0, 2, 2, 0],
[1, 0, 3, 1]]

Top - Left = ((1 * 1 + 2 * 0 + 0 * 1) + (3 * 0 + 1 * 1 + 1 * 0) + (0 * 1 + 2 * 0 + 2 * 1)) + 1 = (1 + 1 + 2) + 1 = 5

Top – Right = ((2 * 1 + 0 * 0 + 1 * 1) + (1 * 0 + 1 * 1 + 5 * 0) + (2 * 1 + 2 * 0 + 0 * 1)) + 1 = (2 + 1 + 1 + 2) + 1 = 7

Bottom – Left = ((3 * 1 + 1 * 0 + 1 * 1) + (0 * 0 + 2 * 1 + 2 * 0) + (1 * 1 + 0 * 0 + 3 * 1)) + 1 = (3 + 1 + 2 + 1 + 3) + 1 = 11

Bottom – Right = ((1 * 1 + 1 * 0 + 5 * 1) + (2 * 0 + 2 * 1 + 0 * 0) + (0 * 1 + 3 * 0 + 1 * 1)) + 1 = (1 + 5 + 2 + 1) + 1 = 10

따라서, Conv 출력은

[[5, 7],
[11, 10]]

5. CNN 수식으로 열어보기 1. 순전파

✓ **입력 데이터가 신경망의 각 층을 통과하며 예측값을 계산하는 과정**

순전파 2단계. ReIU 레이어 계산 0보다 작은 값은 0으로, 0보다 큰 값은 그대로

$$A_{relu} = \max(0, Z_{conv})$$

✓ **숫자를 통해 따라가보자**

이미 모든 값이 0보다 크므로, 변경점은 없다

[[5, 7],
 [11, 10]]

5. CNN 수식으로 열어보기 1. 순전파

✓ **입력 데이터가 신경망의 각 층을 통과하며 예측값을 계산하는 과정**

순전파 3단계. MaxPooling 레이어 계산

$$A_{pool} = \max(A_{relu})$$

✓ **숫자를 통해 따라가보자**

필터의 크기가 ReLU의 전체 크기와 같으므로 ReLU 전체에서 가장 큰 값을 선택한다

[[5, 7],
[11, 10]]

여기서, 제일 큰 숫자가 11 이므로,
MaxPooling 의 출력은 11이 된다

[[11]]

5. CNN 수식으로 열어보기 1. 순전파

✓ **입력 데이터가 신경망의 각 층을 통과하며 예측값을 계산하는 과정**

순전파 4단계. Flatten 레이어 계산

입력값을 1차원 백터로 펼쳐준다

$$A_{flat} = \text{reshape}(A_{pool})$$

✓ **숫자를 통해 따라가보자**

이미 1 * 1 행렬이므로, 1차원 백터로 풀어도 생긴게 거의 비슷하다

[[11]] → [11]

만일, 행렬이 더 크다면?

https://gihak111.github.io/ai/2025/09/23/Flatten_layer_upload.html

5. CNN 수식으로 열어보기 1. 순전파

✓ **입력 데이터가 신경망의 각 층을 통과하며 예측값을 계산하는 과정**

순전파 5단계. Dense 레이어 계산

Flatten된 벡터를 가중치 행렬과 곱하고 편향을 더해준다

$$Z_{dense} = W_{dense} \cdot A_{flat} + b_{dense}$$

✓ **숫자를 통해 따라가보자**

계산을 진행하면,

[11] @ [[0.5, -0.5]] + [0.1, 0.2]

[11 * 0.5, 11 * -0.5] + [0.1, 0.2]

[5.5, -5.5] + [0.1, 0.2] = [5.6, -5.3]

따라서, Z_dense = [5.6, -5.3]

✓ 입력 데이터가 신경망의 각 층을 통과하며 예측값을 계산하는 과정

순전파 6단계. Softmax 레이어 계산

Dense 출력을 확률로 변환, 정답과의 오차를 계산한다

Softmax 계산 $\hat{y}_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$ **Cross – Entropy 계산** $L = -\sum_i y_i \log(\hat{y}_i)$

✓ 숫자를 통해 따라가보자

Softmax 계산

$e^{5.6} \approx 270.43$

$e^{-5.3} \approx 0.005$

최종 예측값 (*y_pred*): `[0.99998, 0.00002]`

Cross – Entropy 계산

$Loss = -(1 \cdot \log(0.99998) + 0 \cdot \log(0.00002))$

최종 손실 (Loss): $\approx \mathbf{0.00002}$

$\delta_{dense} = \hat{y} - y = [0.99998 - 1, 0.00002 - 0] = [-0.00002, 0.00002]$

5. CNN 수식으로 열어보기 2. 역전파

✓ **입력 데이터가 신경망의 각 층을 통과하며 예측값을 계산하는 과정**

역전파 1단계. Sofrmax + Cross Entroppt 기울기 계산

기울기 $\frac{\partial L}{\partial Z_i} = \hat{y}_i - y_i$

[0.99998, 0.00002] – [1, 0] = [-0.00002, 0.00002]

결과 $\frac{\partial L}{\partial Z_{\text{dense}}}$ = [-0.00002, 0.00002]

✓ **입력 데이터가 신경망의 각 층을 통과하며 예측값을 계산하는 과정**

역전파 2단계. Dense Layer 역전파

가중치 기울기 $\frac{\partial L}{\partial W_{dense}} = A_{flat}^{T} \cdot \frac{\partial L}{\partial Z_{dense}}$

[[11]] @ [[-0.00002, 0.00002]] = [[0.00022, 0.00022]]

편향 기울기 $\frac{\partial L}{\partial b_{dense}} = \frac{\partial L}{\partial Z_{dense}}$

[-0.00002, 0.00002]

입력 기울기

$\frac{\partial L}{\partial A_{flat}} = \frac{\partial L}{\partial Z_{dense}} \cdot W_{dense}^{T}$

[-.00002, 0.00002] @ [[0.5], [-0.5]]

= -0.00001 – 0.00001

= -0.00002

5. CNN 수식으로 열어보기 2. 역전파

✓ **입력 데이터가 신경망의 각 층을 통과하며 예측값을 계산하는 과정**

역전파 3단계. Flatten Layer 역전파

1D 백터 형태의 기울기를 다시 1 * 1 행렬로 복원한다

$$\frac{\partial L}{\partial A_{pool}} = \text{reshape}^{-1\backslash Big}\left(\frac{\partial L}{\partial A_{flat}} \backslash Big\right)$$

$$\frac{\partial L}{\partial A_{pool}} = [[-0.00002]]$$

5. CNN 수식으로 열어보기 2. 역전파

✓ **입력 데이터가 신경망의 각 층을 통과하며 예측값을 계산하는 과정**

역전파 4단계. MaxPooling Layer 역전파

$$\frac{\partial L}{\partial A_{relu}} = \mathbf{1}(A_{relu} = \max(A_{relu})) \cdot \frac{\partial L}{\partial A_{pool}}$$

결과 = [[0, 0],
[-0.00002, 0]]

순전파 Maxpooling때의 위치가 Bottom - Left

[[5, 7],
[11, 10]]

여기서, 제일 큰 숫자가 11 이므로,
MaxPooling 의 출력은 11이 된다

[[11]]

✓ **입력 데이터가 신경망의 각 층을 통과하며 예측값을 계산하는 과정**

역전파 5단계. ReLU Layer 역전파

순전파 시 입력이 모두 0보다 컸으므로, 기울기는 그대로 전달

$$\frac{\partial L}{\partial Z_{conv}} = \mathbf{1}(Z_{conv} > 0) \cdot \frac{\partial L}{\partial A_{relu}}$$

결과 = [[0, 0],
[-0.00002, 0]]

✓ **입력 데이터가 신경망의 각 층을 통과하며 예측값을 계산하는 과정**

역전파 6단계. Convolution Layer 역전파

필터 가중치 기울기

출력 기울기에서 0이 아닌 값은 Bottom – Left 위치의 -0.00002 뿐이다.
따라서 이 값을 만드는 데 사용된 입력 이미지의 해당 영역과 곱해준다

$$\frac{\partial L}{\partial W_{conv}} = X * \frac{\partial L}{\partial Z_{conv}}$$

$$\frac{\partial L}{\partial X} = W_{conv} * \frac{\partial L}{\partial Z_{conv}}$$

5. CNN 수식으로 열어보기 2. 역전파

✓ **입력 데이터가 신경망의 각 층을 통과하며 예측값을 계산하는 과정**

역전파 6단계. Convolution Layer 역전파

필터 가중치 기울기

계산 해 보면,

```
[[3, 1, 1],
 [0, 2, 2],
 [1, 0, 3]] * -0.00002
```

따라서, 결과는 다음과 같다

```
[[-0.00006, -0.00002, -0.00002],
 [-0.     , -0.00004, -0.00004],
 [-0.00002, -0.     , -0.00006]]
```

```
[[1, 2, 0, 1],
 [3, 1, 1, 5],
 [0, 2, 2, 0],
 [1, 0, 3, 1]]
```

입력 데이터

✓ **입력 데이터가 신경망의 각 층을 통과하며 예측값을 계산하는 과정**

역전파 6단계. Convolution Layer 역전파

편향 기울기

$$\frac{\partial L}{\partial b_{conv}} = \sum \frac{\partial L}{\partial Z_{conv}}$$

계산 = 0 + 0 – 0.00002 + 0 = -0.00002

✓ **입력 데이터가 신경망의 각 층을 통과하며 예측값을 계산하는 과정**

가중치 업데이트. Dense Layer 업데이트

가중치 업데이트 수식

$$W_{new} = W_{old} - \eta \cdot \frac{\partial L}{\partial W}$$

Dense 레이어 업데이트

$W_{dense,\,new}$ = [[0.5, -0.5]] – 0.1 * [[-0.00022, 0.00022]]
= [[0.500022, -0.500022]]

$b_{dense,\,new}$ = [0.1, 0.2] – 0.1 * [-0.00002, 0.00002]
=[0.100002, 0.199998]

5. CNN 수식으로 열어보기 3. 가중치 업데이트

✓ **입력 데이터가 신경망의 각 층을 통과하며 예측값을 계산하는 과정**

가중치 업데이트. Conv Layer 업데이트

$$W_{new} = W_{old} - \eta \cdot \frac{\partial L}{\partial W}$$

[[1, 0, 1], [[-0.00006, -0.00002, -0.00002],
[0, 1, 0], - 0.1 * [-0. , -0.00004, -0.00004],
[1, 0, 1]] [-0.00002, -0. , -0.00006]]

[[1.00006, 0.00002, 1.00002],
= [0.00002, 1.00004, 0.00004],
[1.00002, 0.00002, 1.00006]]

5. CNN 수식으로 열어보기 3. 가중치 업데이트

✓ **입력 데이터가 신경망의 각 층을 통과하며 예측값을 계산하는 과정**

가중치 업데이트. Conv Layer 업데이트

$$b_{\text{conv, new}}$$

$b_{conv,\,\text{new}}$ = 1 (– 0.1 * -0.00002) = 1.00002

5. CNN 수식으로 열어보기 4. 최종 비교

학습 전

Conv 필터(W_conv), 편향(b_conv)

$w_{conv} =$ [[1, 0, 1],
[0, 1, 0],
[1, 0, 1]]

$b_{conv} =$ 1

Dense 가중치(W_dense), 편향(d_dense)

$w_{\mathrm{dense}} =$ [[0.5, -0.5]]

$b_{\mathrm{dense}} =$ [0.1, 0.2]

학습 후

Conv 필터(W_conv), 편향(b_conv)

$w_{conv} =$ [[1.00006, 0.00002, 1.00002],
[0.00002, 1.00004, 0.00004],
[1.00002, 0.00002, 1.00006]]

$b_{conv} =$ 1.00002

Dense 가중치(W_dense), 편향(d_dense)

$w_{\mathrm{dense}} =$ [[0.500022, -0.500022]]

$b_{\mathrm{dense}} =$ [0.100002, 0.199998]

6. Numpy로 CNN 구현하기

https://huggingface.co/gihakkk/CNN_test

```python
import numpy as np
# 데이터 로딩을 위해서만 tensorflow.keras.datasets를 사용합니다.
# 실제 모델 아키텍처와 학습 로직에는 전혀 사용되지 않습니다.
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# ============================================================
# --- 1. 데이터 준비 ---
# ============================================================

def load_mnist_data():
    """MNIST 데이터셋을 불러오고 전처리합니다."""
    (x_train, y_train), (x_test, y_test) = mnist.load_data()

    # 픽셀 값을 0과 1 사이로 정규화
    x_train = x_train.astype('float32') / 255.0
    x_test = x_test.astype('float32') / 255.0

    # 채널 차원 추가 (N, 28, 28) -> (N, 28, 28, 1)
    x_train = np.expand_dims(x_train, -1)
    x_test = np.expand_dims(x_test, -1)

    # 레이블을 원-핫 인코딩
    y_train = to_categorical(y_train, 10)
    y_test = to_categorical(y_test, 10)

    return x_train, y_train, x_test, y_test
```

Numpy 임포트, 입력 데이터 전처리

픽셀값 정규화 하는 이유는,
딥러닝 계속 해 보니까 값이 0 ~ 1 사이일 경우
가장 잘 되기 때문에

자세한건
왜 입력값이 0 ~ 1 로 정규화 되어야 할까?

6. Numpy로 CNN 구현하기


```python
class ConvLayer:
    """합성곱 계층"""
    def __init__(self, num_filters, filter_size, input_channels):
        self.num_filters = num_filters
        self.filter_size = filter_size
        self.input_channels = input_channels
        # Xavier/Glorot 초기화 (더 좋은 학습 안정성)
        self.filters = np.random.randn(filter_size, filter_size, input_channels, num_filters) * np.sqrt(2. / (filter_size * filter_size * input_channels))
        self.biases = np.zeros(self.num_filters)
        self.last_input = None

    def forward(self, input_image):
        self.last_input = input_image
        batch_size, input_height, input_width, _ = input_image.shape
        output_height = input_height - self.filter_size + 1
        output_width = input_width - self.filter_size + 1

        output = np.zeros((batch_size, output_height, output_width, self.num_filters))

        for b in range(batch_size):
            for i in range(output_height):
                for j in range(output_width):
                    region = input_image[b, i:(i + self.filter_size), j:(j + self.filter_size), :]
                    for f in range(self.num_filters):
                        output[b, i, j, f] = np.sum(region * self.filters[:, :, :, f]) + self.biases[f]
        return output
```

Conv layer 구현

6. Numpy로 CNN 구현하기


```python
def backward(self, d_loss_d_output, learning_rate):
    """역전파 메서드 (입력 그래디언트 계산 추가)"""
    batch_size, _, _, _ = self.last_input.shape
    d_loss_d_filters = np.zeros_like(self.filters)
    d_loss_d_input = np.zeros_like(self.last_input)

    # 필터와 입력에 대한 그래디언트 계산
    for b in range(batch_size):
        for i in range(d_loss_d_output.shape[1]): # output height
            for j in range(d_loss_d_output.shape[2]): # output width
                region = self.last_input[b, i:(i + self.filter_size), j:(j + self.filter_size), :]
                for f in range(self.num_filters):
                    # 필터 그래디언트 누적
                    d_loss_d_filters[:, :, :, f] += d_loss_d_output[b, i, j, f] * region
                    # 입력 그래디언트 누적 (Chain Rule)
                    d_loss_d_input[b, i:i+self.filter_size, j:j+self.filter_size, :] += d_loss_d_output[b, i, j, f] * self.filters[:, :, :, f]

    # 편향의 그래디언트
    d_loss_d_biases = np.sum(d_loss_d_output, axis=(0, 1, 2))

    # 가중치 및 편향 업데이트
    self.filters -= learning_rate * d_loss_d_filters / batch_size
    self.biases -= learning_rate * d_loss_d_biases / batch_size

    # 이전 계층으로 전달할 그래디언트 반환
    return d_loss_d_input
```

Conv layer 구현

그래디언트 누적이란? 링크 달자

6. Numpy로 CNN 구현하기


```python
class ReLULayer:
    """ReLU 활성화 함수"""
    def __init__(self):
        self.last_input = None

    def forward(self, input_data):
        self.last_input = input_data
        return np.maximum(0, input_data)

    def backward(self, d_loss_d_output):
        d_relu = (self.last_input > 0).astype(int)
        return d_loss_d_output * d_relu
```

간단한 ReLU 레이어

6. Numpy로 CNN 구현하기


```python
class MaxPoolingLayer:
    """맥스 폴링 계층"""
    def __init__(self, pool_size):
        self.pool_size = pool_size
        self.last_input = None

    def forward(self, input_image):
        self.last_input = input_image
        batch_size, input_height, input_width, num_filters = input_image.shape
        output_height = input_height // self.pool_size
        output_width = input_width // self.pool_size

        output = np.zeros((batch_size, output_height, output_width, num_filters))

        for b in range(batch_size):
            for i in range(output_height):
                for j in range(output_width):
                    for f in range(num_filters):
                        region = input_image[b, (i*self.pool_size):(i*self.pool_size + self.pool_size),
                                                (j*self.pool_size):(j*self.pool_size + self.pool_size), f]
                        output[b, i, j, f] = np.max(region)
        return output
```

앞선 식과 똑같은 구조

6. Numpy로 CNN 구현하기


```
def backward(self, d_loss_d_output):
    d_loss_d_input = np.zeros_like(self.last_input)

    for b in range(d_loss_d_output.shape[0]):
        for i in range(d_loss_d_output.shape[1]):
            for j in range(d_loss_d_output.shape[2]):
                for f in range(d_loss_d_output.shape[3]):
                    region = self.last_input[b, (i*self.pool_size):(i*self.pool_size + self.pool_size),
                                                (j*self.pool_size):(j*self.pool_size + self.pool_size), f]
                    max_val = np.max(region)
                    mask = (region == max_val)
                    d_loss_d_input[b, (i*self.pool_size):(i*self.pool_size + self.pool_size),
                                      (j*self.pool_size):(j*self.pool_size + self.pool_size), f] += \
                                      mask * d_loss_d_output[b, i, j, f]
    return d_loss_d_input
```

6. Numpy로 CNN 구현하기


```python
class FlattenLayer:
    """평탄화 계층"""
    def __init__(self):
        self.last_input_shape = None

    def forward(self, input_data):
        self.last_input_shape = input_data.shape
        batch_size = input_data.shape[0]
        return input_data.reshape(batch_size, -1)

    def backward(self, d_loss_d_output):
        return d_loss_d_output.reshape(self.last_input_shape)

class DenseLayer:
    """완전 연결 계층"""
    def __init__(self, input_size, output_size):
        self.weights = np.random.randn(input_size, output_size) * np.sqrt(2. / input_size)
        self.biases = np.zeros(output_size)
        self.last_input = None
        self.last_input_shape = None

    def forward(self, input_data):
        self.last_input_shape = input_data.shape
        self.last_input = input_data
        return np.dot(input_data, self.weights) + self.biases

    def backward(self, d_loss_d_output, learning_rate):
        batch_size = self.last_input.shape[0]
        d_loss_d_input = np.dot(d_loss_d_output, self.weights.T)
        d_loss_d_weights = np.dot(self.last_input.T, d_loss_d_output)
        d_loss_d_biases = np.sum(d_loss_d_output, axis=0)
        self.weights -= learning_rate * d_loss_d_weights / batch_size
        self.biases -= learning_rate * d_loss_d_biases / batch_size
        return d_loss_d_input
```

마찬가지로
식을 그대로 코드로 구현한 모습

6. Numpy로 CNN 구현하기


```python
# ================================================================
# --- 3. Softmax 및 손실 함수 ---
# ================================================================

def softmax(logits):
    exps = np.exp(logits - np.max(logits, axis=1, keepdims=True))
    return exps / np.sum(exps, axis=1, keepdims=True)

def cross_entropy_loss(y_pred, y_true):
    samples = y_true.shape[0]
    epsilon = 1e-12
    y_pred_clipped = np.clip(y_pred, epsilon, 1. - epsilon)
    loss = -np.sum(y_true * np.log(y_pred_clipped)) / samples
    return loss
```

활성화 함수들 구현

6. Numpy로 CNN 구현하기


```
# ==============================================================
# --- 4. CNN 모델 클래스 ---
# ==============================================================
class SimpleCNN:
    def __init__(self):
        # 입력: 28x28x1
        self.conv1 = ConvLayer(num_filters=8, filter_size=3, input_channels=1) # -> 26x26x8
        self.relu1 = ReLULayer()
        self.pool1 = MaxPoolingLayer(pool_size=2)                              # -> 13x13x8

        self.conv2 = ConvLayer(num_filters=16, filter_size=3, input_channels=8)# -> 11x11x16
        self.relu2 = ReLULayer()
        self.pool2 = MaxPoolingLayer(pool_size=2)                               # -> 5x5x16

        self.flatten = FlattenLayer()                                           # -> 400
        self.dense = DenseLayer(5 * 5 * 16, 10)                                 # -> 10

    def forward(self, image):
        out = self.conv1.forward(image)
        out = self.relu1.forward(out)
        out = self.pool1.forward(out)
        out = self.conv2.forward(out)
        out = self.relu2.forward(out)
        out = self.pool2.forward(out)
        out = self.flatten.forward(out)
        out = self.dense.forward(out)
        return out
```

6. Numpy로 CNN 구현하기


```
def backward(self, d_loss_d_out, learning_rate):
    """역전파 메서드 (그래디언트 흐름 수정)"""
    # 역전파는 순전파의 역순으로 진행
    gradient = self.dense.backward(d_loss_d_out, learning_rate)
    gradient = self.flatten.backward(gradient)

    # 2단계 역전파
    gradient = self.pool2.backward(gradient)
    gradient = self.relu2.backward(gradient)
    gradient = self.conv2.backward(gradient, learning_rate)

    # 1단계 역전파
    gradient = self.pool1.backward(gradient)
    gradient = self.relu1.backward(gradient)
    self.conv1.backward(gradient, learning_rate)
```

6. Numpy로 CNN 구현하기

ONECLICK AI

```
# ============================================================
# --- 5. 학습 루프 ---
# ============================================================

if __name__ == '__main__':
    x_train, y_train, x_test, y_test = load_mnist_data()
    x_train_small, y_train_small = x_train[:1000], y_train[:1000]
    x_test_small, y_test_small = x_test[:500], y_test[:500]

    model = SimpleCNN()

    learning_rate = 0.01
    epochs = 5
    batch_size = 32

    print("학습 시작...")
    for epoch in range(epochs):
        epoch_loss = 0

        for i in range(0, x_train_small.shape[0], batch_size):
            x_batch = x_train_small[i:i+batch_size]
            y_batch = y_train_small[i:i+batch_size]

            logits = model.forward(x_batch)
            predictions = softmax(logits)

            loss = cross_entropy_loss(predictions, y_batch)
            epoch_loss += loss

            d_loss_d_out = (predictions - y_batch)
            model.backward(d_loss_d_out, learning_rate)

        avg_loss = epoch_loss / (len(x_train_small) / batch_size)
        print(f"Epoch {epoch + 1}/{epochs}, Loss: {avg_loss:.4f}")
```

6. Numpy로 CNN 구현하기


```
print("\n테스트 시작...")
test_logits = model.forward(x_test_small)
test_predictions = softmax(test_logits)

predicted_labels = np.argmax(test_predictions, axis=1)
true_labels = np.argmax(y_test_small, axis=1)
accuracy = np.mean(predicted_labels == true_labels)

print(f"테스트 정확도: {accuracy * 100:.2f}%")
```

오늘 한 것

- 텐서란?
- 학습의 종류
- Overfitting
- CNN 아키텍쳐

감사합니다