협성대학교 소프트웨어공학과

인공지능기초

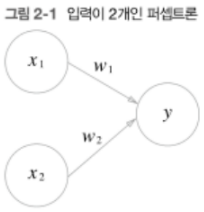
중간고사 대비 1~7 주차 강의 요약.
(<밑바닥부터 시작하는 딥러닝 1> 1~8 장 요약)

이시헌

2024-4-22

1주차 강의 요약 시작

1. 퍼셉트론(perceptron)
   A. 정의: 다수의 신호를 받아서 하나의 신호를 출력하는 알고리즘.
   B. 구조: $x_1 \cdot w_1 + x_2 \cdot w_2 = y$
   C. 흐른다: $x_1 \cdot w_1 + x_2 \cdot w_2 > \theta : 1$
   D. 흐르지 않는다: $x_1 \cdot w_1 + x_2 \cdot w_2 \leq \theta : 1$



      i. x_n: 입력값.
      ii. w_n: 가중치. 입력이 뉴런에 보내질 때 각각 고유한 가중치를 곱한다. (각 신호가 결과에 미치는 영향력을 조절하는 요소.)
      iii. θ: 임계값. 임계값을 넘는 경우에만 신호가 흐른다.

2. 기계학습
   A. 정의: 신경망의 구조, 가중치, 임계값을 컴퓨터가 자동으로 수정하도록 학습하는 것.

3. 편향(bias)
   A. 정의: 임계값을 치환한 값. ($\theta = -b$)
   B. 목적:
      i. 수학적 표현을 간단하게 하기 위함.
      ii. 계산 효율성을 높이기 위함. (경사 하강법 등의 알고리즘 적용에 유리)
   C. 임계값을 편향으로 대체. (출력이 1인 경우를 예시로 들겠다.)
      i. $x_1 \cdot w_1 + x_2 \cdot w_2 > -b : 1$
      ii. $b + x_1 \cdot w_1 + x_2 \cdot w_2 > 0 : 1$
   D. 해석: 이때 편향(b)은 뉴런이 얼마나 쉽게 활성화 되는지를 결정한다.

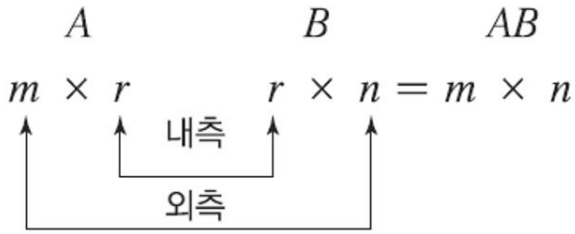
4. 퍼셉트론의 한계
   A. 입력이 2개인 퍼셉트론의 경우, 어떠한 가중치(편향을 포함)로도 좌표평면을 비선형으로 분할할 수 없다(선형 분리 불가능 문제). 입력이 2개인 퍼셉트론은 2차원 공간에서의 점들을 두 부류로 분류하는 문제를 해결할 수 있는데, 이 분류 기준이 직선이라는 뜻이다. (XOR 연산을 할 수 없다는 것이 대표적인 한계이다.)

5. 다층 퍼셉트론
   A. 등장 배경: 단일 퍼셉트론의 한계를 극복하기 위하여 등장했다.
      i. 우리는 복수의 퍼셉트론을 엮어서, 다수의 파라미터로 동작하며, 좌표공간을 잘게 잘라서 구분(분류)할 수 있도록 한다.
   B. 2층 퍼셉트론만으로 컴퓨터를 만들 수 있다는 것이 증명되었다. (비선형 시드모이드를 활성화 함수로 사용하면, 임의의 함수를 표현 가능.)
   C. NAND 연산만으로 컴퓨터를 만들 수 있고, 2층 퍼셉트론으로 NAND연산의 구현 가능.

1주차 강의 요약 끝. (2장 정리 끝)

2주차 강의 요약 시작

1. 행렬 연산 복습
   A. 행렬간 곱연산만 정리하겠다.
   B. 조건:



   i.
   ii. 즉, A의 열의 개수가 B의 행의 개수와 같아야 한다.
   iii. 바꿔 말하면, 내측의 두 값이 같아야 하고, 외측의 두 값은 새로 만들어진 행렬의 크기이다.
   C. 주의사항
      i. 행렬에 스칼라 값을 곱하는 것 과는 전혀 다른 연산이다.
      ii. 행렬간 곱셈에는 교환법칙이 성립하지 않는다.
         - $AB \neq BA$

2. 활성화 함수
   A. 정의: 입력 신호의 총합을 출력 신호로 변환하는 함수. 즉, 입력 신호의 총합이 활성화를 일으키는지를 정한다.

3. 활성화 함수의 종류
   A. 오로지 비선형함수만 활성화함수로 사용될 수 있다. 이는 직선 한 개로는 표현할 수 없는 함수를 뜻한다.
   B. 계단함수
      i. 정의: 임계값 이후 1을 출력하는 함수이다. (불연속이다.)
      ii. 퍼셉트론에서는 활성화함수로 계단함수를 사용한다.
      iii. $a = b + x_1 w_1 + x_2 w_2$
      iv. $y = h(a)$ 〈- 이 $h(x)$가 활성화함수이다. (최종 출력값을 만드는 함수.)

C. 시그모이드 함수

i. 정의: $h(x) = \frac{1}{1+\exp(-x)}$ 〈- $\exp(-x) = e^{-x}$

ii. 특징: 연속함수이다. 즉, 입력값에 따라서 값이 선형적으로 출력된다. (최대는 1이고, 최소는 0이다.)

4. 활성화함수로 선형 함수를 사용할 수 없는 이유

A. 신경망의 층을 깊게 하는 의미가 없어짐.

B. 단층 퍼셉트론이 좌표공간을 직선으로 나누던 것을 떠올려라. 우리가 기계학습으로 얻고자 하는 값들은 매우 복잡하다. 즉, 층을 쌓기 위해서는 비선형 함수를 활성화 함수로 써야만 한다. 아래의 설명을 보라.

C. $h(x) = cx$ 를 활성화 함수로 한 3층 네트워크는 아래와 같다.

i. $y(x) = h(h(h(x)))$

ii. $y(x) = c^3x$

iii. 이때 c는 상수이므로, 은닉층이 없는 네트워크와 동일하다. (아무리 층을 늘려도 아무 효과가 없다.)

5. ReLU 함수 (Rectified Linear Unit)

A. 최근 시그모이드를 대신하여 활성화 함수로 더 많이 쓰이는 함수이다. (시그모이드의 기울기 정보 손실 문제를 피할 수 있다.)

i. 기울기 소실 문제: 시그모이드 함수의 기울기는 입력 값이 0에서 멀어질수록 급격히 0에 가까워진다. 그로 인하여 역전파 과정에서 기울기가 계속하여 작아질 수 있으며, 신경망의 앞쪽 레이어들이 충분히 학습되지 않게 된다.

B. 정의:

i. $h(x) = \begin{cases} x\ (x > 0) \\ 0\ (x \leq 0) \end{cases}$

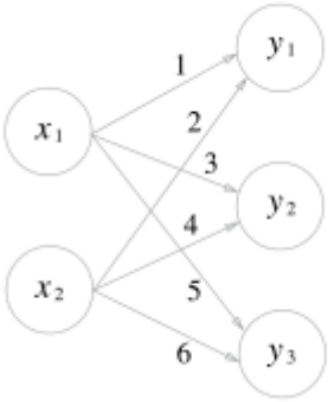
ii. 0 초과의 입력 -〉 입력을 그대로 출력.

iii. 0 이하의 입력 -〉 0 출력.

6. numpy에서의 행렬간 곱셈 방법

A. np.dot(A, B)

7. 신경망의 출력을 행렬곱으로 연산하기



$$\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

$$X \quad W \quad = \quad Y$$

$$2 \quad 2 \times 3 \quad 3$$

일치

A.

B. 그림만 보아도 이해가 될 것이다.

C. 입력: 1x2 / 가중치: 2x3 / 출력: 1x3

D. 가중치 행렬의 값:

i. $x_1 \to y_1 \quad x_1 \to y_2 \quad x_1 \to y_3$
$x_2 \to y_1 \quad x_2 \to y_2 \quad x_2 \to y_3$

E. 즉, 반복문 없이 행렬 곱 연산만으로 신경망의 출력 값을 구할 수 있다.

i. Y = np.dot(X, W)

8. 신경망의 목적 (기계학습 문제의 분류)

A. 분류: 데이터가 어느 클래스에 속하는지 분류하는 문제 해결.

i. 일반적으로 출력층의 활성화함수로 소프트맥스(9번 참고) 함수를 사용.

B. 회귀: 입력 데이터로부터 연속적인 미래 데이터 예측.

i. 일반적으로 출력층의 활성화함수로 항등함수(입력을 그대로 출력)를 사용함.

9. 소프트맥스 함수

A. 정의: 모든 입력에 영향을 받는, 1이하의 값을 출력하는, 입력의 대소관계를 해치지 않는 함수. (출력이 1 이하이기에, 모든 클래스에 대한 확률로 해석 가능하게 해 준다.)

B. 즉, 구분하고자 하는 클래스 개수만큼의 값들을 모두 확률로 바꾸는 것이다. (모든 클래스에 대한 소프트맥스 함수의 출력값 합산이 1인 것이다.)

10. 배치(batch)

A. 정의: 입력 데이터 여럿을 한번에 묶은 것.

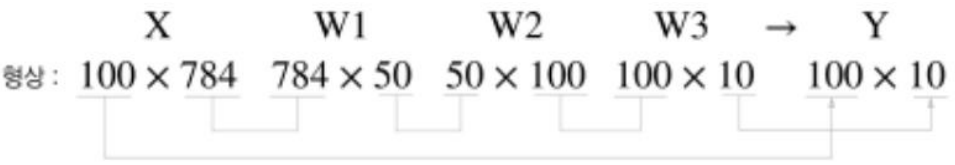
B. 목적: 큰 배열을 한번에 읽어와서 처리하는 것이 더 빠르기에, 배치 단위로 입력 데이터를 처리하기 위함.

i. 일반적으로 수치 계산 라이브러리는 큰 배열을 처리할 때 더 효율적으로 동작하기 때문이다. (I/O 속도 < CPU, GPU 연산 속도)

C. 배치 단위로 신경망에 값을 입력해도, 하나씩 입력 할 때와 동일한 연산 결과를 얻을 수 있다.

i.

ii.



iii. 100개의 이미지를 묶어서 입력한 예시이다. (배치 = 100장 인 경우)

11. 정리

A. 3장의 내용은 순전파에 대한 내용이다.

B. 순전파는 퍼셉트론과 유사하다. 다만, 활성화함수가 다르다.

i. 퍼셉트론에서는 계단함수를 활성화함수로 사용.

ii. 신경망에서는 시그모이드 함수를 활성화함수로 사용. (다른 것 또한 얼마든지 사용 가능.)

C. 신경망에는 비선형 함수만을 활성화 함수로 사용 가능.

i. 선형 함수를 사용할 경우, 층을 아무리 쌓아도 단층과 동일.

2주차 강의 요약 끝. (3장 정리 끝)

3주차 강의 요약 시작

1. 신경망 학습
   A. 정의: 매개변수를 데이터로부터 자동으로 획득하도록 하는 것.
   B. 기계학습에서는 사람의 개입을 최소화하려고 노력한다.
      -> 신경망과 딥러닝은 기존 기계학습보다 사람의 개입을 더욱 줄였다.
   C. 종단간 기계학습(end-to-end machine learning):
      i. 데이터에서 목표한 결과를 사람의 개입 없이 얻는 것.
      ii. 즉, 사람이 규칙을 만들지 않고, 기계가 데이터로부터 배운다.

2. 훈련 데이터 / 시험 데이터
   A. 훈련 데이터: 훈련(학습)에만 사용하는 데이터.
   B. 시험 데이터: 시험에만 사용하는 데이터.
   C. 구분하는 이유: 훈련된 모델이 범용 능력(일반화 성능)을 획득하도록 하는 것이 기계학습의 최종 목표이기 때문에, 데이터를 구분해야 한다.
      i. 범용 능력: 아직 보지 못한 데이터로도 문제를 올바르게 풀어내는 능력.
      ii. 과적합(overfitting): 특정 데이터셋에만 지나치게 최적화 된 상태.
      iii. 즉, 과적합을 피하고, 범용 능력을 평가하기 위하여, 훈련 데이터와 시험 데이터를 구분하는 것이다.

3. 손실 함수
   A. 정의 1: 신경망 성능의 '나쁨'을 나타내는 지표이다. (오차를 뜻함)
      i. 왜 나쁨인가? 추후 등장하지만, 미분을 통한 학습 방향성 예측에 유리하기 때문이다. ('좋음'을 지표로 하는 함수를 사용하면, 대부분의 위치에서 기울기가 0이다.)
   B. 정의 2: 정답(타겟)과 예측한 값(출력)의 차이를 나타내는 함수이다.
   C. 일반적으로 두 가지 함수를 손실 함수로 사용한다.
      i. 오차제곱합 함수(sum of squares for error, SSE)
      ii. 교차 엔트로피 함수(cross entropy error, CEE)

4. 오차제곱합 (가장 많이 쓰인다)

A. $E = \frac{1}{2}\sum_k (y_k - t_k)^2$

B. y_k: 출력(추정값) (1=100%, 0=0%) (소프트맥스 함수 적용됨.)

C. t_k: 정답 레이블(실제값, 참값) (1=정답, 0=오답)

D. k: 데이터의 차원 수

E. 적용 예시 (한 대상에 대한 10개 클래스의 분류 문제)

k	y	t	$y-t$	$(y-t)^2$
0	0.1	0	−0.1	0.001
1	0.05	0	−0.05	0.0025
2	0.6	1	−0.4	0.16
3	0.0	0	−0.0	0.0
4	0.05	0	−0.05	0.0025
5	0.1	0	−0.1	0.01
6	0.0	0	−0.0	0.0
7	0.1	0	−0.1	0.01
8	0.0	0	−0.0	0.0
9	0.0	0	−0.0	0.0

F. 위의 표에서 $(y-t)^2$를 모두 더하면, 오차제곱합 함수의 출력 값(E)이다.

G. 즉, 정답이어도 더 높은 확률이어야 오차(E)가 감소하고,
정답이 아닌 클래스에 대하여 더 낮은 확률이어야 오차(E)가 감소한다.
(오차가 적을수록 더 예측이 정확한 것!)

5. 교차 엔트로피 오차 (이것도 자주 쓰인다)

A. $E = -\sum_k t_k \log_e y_k$

B. 특이한 점이 있다. t_k는 정답인 k에서만 1이므로,
$\sum_k t_k \log_e y_k$는 정답일 때의 추정 값의 자연로그를 계산하는 식과 같다.
즉, 정답일 때의 출력 값(y_k)이 전체 값(E)을 결정한다!
(출력이 정답에 가까워질수록 오차는 가파르게 0에 가까워진다.)

C. 4번의 표를 계산하면 다음과 같다.

i. $E = -\log 0.6$ (t=1, y=0.6)

6. 미니배치 학습 (mini-batch)

A. 교차 엔트로피 함수를 쓴다고 하자. 그렇다면 데이터의 개수만큼 오차 값을 계산해야만, 모든 데이터로부터 계산된 평균 오차 값을 구할 수 있다.
(이를 구하는 함수를 '평균 손실 함수'라고 칭한다.)

i. $E = -\frac{1}{n}\sum_n \sum_k t_k \log_e y_k$

ii. 위의 식은 모든 데이터의 오차 값으로 평균을 구한 것이다.

B. 하지만 메모리나 연산 능력이 부족할 때가 있다.
그럴 경우, 전체 데이터에서 일부를 무작위로 골라서 학습시킬 수 있고,
그것을 미니배치 학습이라 한다.

i. 예시: 60,000장의 훈련 데이터에서 100장을 무작위로 뽑아서 학습을 수행하는 경우.
이 100장이 미니배치 단위이고,
이 단위로 학습을 수행하는 것을 미니배치 학습이라 하는 것.

C. 이 방법으로 평균 손실 함수의 출력 값(오차)을 근사적으로 구할 수 있다.

7. 손실 함수를 사용하는 이유

A. 인간의 개입 없이 학습을 하기 위해서는, 손실이 가장 적은 매개변수를 기계가 스스로 찾을 수 있어야 한다.
이때 손실 함수를 사용한다면, 함수의 기울기를 통하여, 가장 작은 손실 값을 출력하는 매개변수의 방향을 알아낼 수 있다.

i. 예시: 미분 값이 음수 -> 가중치 매개변수를 양의 방향으로.

B. 그러나 손실 함수가 아닌, 정확도 함수를 사용할 시, 대부분의 장소에서 (불연속적 값을 출력하므로) 미분 값이 0이 된다.

i. 데이터에서 정답을 맞춘 횟수를 통하여 백분율을 계산하기에, 미세한 값의 변화가 무시된다. 즉, 미세한 값의 변화로 올바른 방향을 예측하기 어렵다.

C. 다만 손실 함수의 기울기를 구할 때, 해석적으로 미분을 하지는 않는다.

i. 해석적 미분을 할 때 생기는 오차와, 무한소인 h를 계산할 수 없다는 한계로 인하여, 우리는 수치 미분(아주 작은 값을 대입하여 구하는 근사적 미분)을 통해 손실 함수의 기울기를 구한다.

8. 기울기(gradient)

A. 모든 변수의 편미분을 벡터로 정리한 것을 의미한다.

i. 교재에서는 변수가 2개인 기울기 함수가 주어지는데, 실제로 신경망이 갖는 매개변수가 100개라면, 100차원의 벡터 값이 기울기 함수의 출력 값이 된다.

B. 벡터라는 것을 기억하라. 방향성과 크기를 갖는 벡터!
즉, 기울기 함수의 출력 값은, 입력받은 매개변수의 값들로부터,
"손실 값이 가장 작은 지점"을 가리키게 된다.

C. 다만, 항상 그런 것은 아니다. 실제로는 국소적인 최솟값을 가리키게 된다.

9. 경사 하강법(경사법)

A. 신경망은 학습을 통해 최적의 매개변수(가중치와 편향)를 찾아야 한다.
그러나 매개변수 공간은 매우 광대하여 최솟값의 위치를 짐작하기 어렵다.
이때 경사법을 사용하여, 최솟값의 위치로 매개변수의 값을 이동시킬 수 있다.

B. 각 지점(매개변수 값)에서의 기울기는 가장 가까운 최소값의 방향을 가리킨다. 이는 국소적인 최소값으로서, 극소값이라 한다. (기울기가 0인 지점)

C. 즉, 경사법은 다음의 프로세스를 말한다.

i. 현재의 기울기를 계산한다.

ii. 기울기가 가리키는 방향으로 '학습률'만큼 매개변수를 변경한다.

iii. 오차가 최소가 될 때 까지 i과 ii를 반복한다.

D. 경사법을 1회 적용하는 것을 수식으로 나타내면 다음과 같다.
(교재의 기울기 함수는 매개변수가 2개이다. 기울기 함수: $f(x_0, x_1)$

i. 하단의 수식에서는 우변을 좌변에 대입한다고 생각하라.

ii. η 기호는 학습률이다. (learning rate)

iii. $x_0 = x_0 - \eta \frac{\partial f}{\partial x_0}$ //첫 번째 매개변수에 대한 1회 경사법 적용.

iv. $x_1 = x_1 - \eta \frac{\partial f}{\partial x_1}$ //두 번째 매개변수에 대한 1회 경사법 적용.

E. 즉, [[기울기에 학습률을 곱한 값]을, 현재 매개변수에서 빼서 구한 값]으로, [매개변수]를 갱신하는 것이다.

F. 이때 학습률을 조정하여, 국소적인 최솟값을 넘어설 수 있다.

10. 하이퍼 파라미터 (hyper parameter)
    A. 사람이 직접 설정해야 하는 매개변수를 칭하는 용어다. (학습률, 배치 사이즈 등…)

11. 신경망 학습의 절차 (지금까지의 내용 요약)
    A. 데이터 준비
        i. (미니배치의 경우: 훈련 데이터에서 미니배치 단위로 무작위 데이터를 가져온다. 미니배치 단위로 경사 하강법을 사용한다면,
        이를 “확률적 경사 하강법”이라 한다.)
        ii. 데이터는 문제와 정답으로 구성되어 있다. (x, t)
        iii. 우리의 목표는 이 데이터를 신경망에 투입했을 때,
        출력되는 손실 함수의 값을 줄이는 것이다.
    B. 모델 선택
        i. 분류 문제인지 회귀 문제인지를 판별하여, 신경망 모델을 정한다.
        ii. 우리는 데이터 x를 신경망에 투입하여, 결과 y를 얻을 것이다.
        $x \rightarrow$ *모델* $\rightarrow y$ （ y : 예측 값, 추정 값 ）
    C. 오차 계산
        i. 손실 함수를 통하여 오차를 계산한다.
        ii. 현재 매개변수들의 값으로 출력되는 결과값들로 오차를 계산하는 것이다. ($x \rightarrow$ *모델* $\rightarrow y \rightarrow$ *손실함수* $\rightarrow$ *오차*)
    D. 기울기 계산
        i. 손실 값을 바탕으로 기울기(grad)를 구한다.
        ii. 현재 매개변수들의 값으로 기울기를 구한다는 것이다.
    E. 매개변수 갱신
        i. 학습률(lr)을 정하고, 매개변수를 갱신한다.
        ii. $w_{t+1} = w_t - lr \cdot grad$
    F. [오차 계산 -〉 기울기 계산 -〉 매개변수 갱신] 과정을,
    오차가 더 이상 낮아지지 않을 때 까지 반복한다.
        i. E단계 직후, 갱신된 매개변수로 다시 오차를 계산하라는 것이다.
        (만일 오차가 감소했다면 학습이 잘 이루어지고 있는 것이다.)

12. 에폭(epoch)

A. 학습에서 훈련 데이터를 모두 소진했을 때의 학습 횟수.

B. 예시:

i. 훈련 데이터 10,000개 + 100개를 미니배치로 학습
-〉 확률적 경사 하강법 100회 시행 시 모든 훈련 데이터 소진.

ii. 위의 경우, [100회 = 1에폭]

C. 즉, 미니배치 단위로 훈련 데이터를 모두 소진하는데 필요한 학습 횟수를 지칭하는 단위가 '에폭'인 것이다.

13. 시험 데이터로 평가

A. 학습 중에 손실함수의 값이 더 이상 낮아지지 않는다면, 신경망을 평가할 때가 온 것이다.

B. 미리 준비해 둔 시험용 데이터를 이용하여, 신경망이 범용 능력(일반화 성능)을 갖추었는지 확인해야 한다.

C. 이를 통하여 과적합(overfitting)이 일어났는지 확인이 가능하다.

D. 일반적으로 1에폭 단위로 훈련을 마칠 때 마다, 시험 데이터로 신경망을 평가하게 된다.

i. 에폭 단위로 평가하는 이유는, 큰 관점에서 신경망의 성능 추이를 알아보는 것으로 충분하기 때문이다. (훈련 데이터와 시험 데이터에 대한 성능을 알아보는 것은 에폭 단위의 검증으로 충분하다는 것. 물론, 연산력의 낭비도 막을 수 있다.)

ii. 에폭 단위로 평가한다는 뜻은, 에폭 단위로 학습 데이터를 재사용 해서 또 학습시킨다는 말이다.
(배치 단위의 데이터가 무작위로 뽑히기 때문에, 동일 데이터로 수행한 재학습이 의미 있게 신경망을 훈련시킬 수 있는 것이다.)

iii. 에폭이 진행될수록 훈련 데이터와 시험 데이터로 평가한 정확도가 개선되고, 둘 간의 차이가 줄어든다.

E. 만일 과적합이 일어났음이 발견된다면, 추후 등장할 다른 기법으로 과적합을 해소해야 한다.

3주차 강의 요약 끝. (4장 정리 끝.)

4주차 강의 요약 시작

1. 오차역전파 (backward propagation of errors, 역전파)
   A. 우리는 신경망의 매개변수를, 오차가 더 적은 방향으로 변경해야 한다.
      (이때 손실함수의 기울기를 수치 미분을 통해 근사적으로 구하는 방법을 3장에서 알아보았다.<-'가중치 매개변수의 기울기'를 구하는 것.)
      그러나 모든 매개변수에 대하여 매번 기울기를 계산하는 것은 현실적으로 매우 어려운 경우가 많다. (시간 소요, 복잡성 문제)
   B. 즉, 오차역전파법은 가중치 매개변수의 기울기를 효율적으로 계산할 수 있는 방법이다. (경사하강법을 효과적으로 적용하는 방법으로 이해해도 좋다.)

2. 계산 그래프
   A. 역전파를 수식으로 계산할 수 있지만, 교재에서는 쉬운 이해를 위해 '계산 그래프'를 동원하기로 했다.
   B. 계산 그래프는 계산 과정을 그래프로 나타낸 것이다.
      i. 이때 노드에는 연산자만 표기하고,
      ii. 노드에 화살표로 값을 입력하고,
      iii. 노드에서 연산 결과가 화살표로 출력된다.
   C. 모든 노드는 방향성을 지닌다. 위의 항목B에서 말하는 입력과 출력대로 연산이 수행된다면 '순전파'라고 하며, 인간이 일반적으로 수행하는 계산과 동일한 경우이다.
      -> 하지만 계산을 꼭 정방향으로만 해야 하는 것은 아니다. 우리는 노드의 종착점에 '오차'값을 넣어서, '역전파'를 통해 모든 가중치 매개변수의 기울기를 구할 수 있다!
      i. 순전파: 종착점으로 진행되는 방향으로 연산을 수행하는 것.
      ii. 역전파: 출발점 방향으로 연산을 수행하는 것.
   D. 국소적 계산: 자신과 직접 관계된 작은 범위에서의 계산.
      i. 계산 그래프는 '국소적 계산'을 전파하는 것으로 최종 결과를 얻는다.
      ii. 이는 특정한 노드가, 인접한 화살표들의 값만으로 계산 결과를 만들 수 있다는 것이다. 아무리 복잡한 그래프에서도, 각 노드에서는 단순한 계산만 수행하면 된다는 것!

3. 연쇄법칙

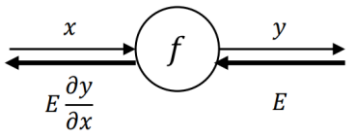
A. 합성 함수의 미분은 합성 함수를 구성하는 각 함수의 미분의 곱으로 나타낼 수 있다는 법칙이다.

B. 이 법칙에 기반하여, 역전파시의 국소적인 계산을 처리할 수 있다.
(각 노드에서의 계산 값을 계속하여 곱하며 역전파 가능.)

C. 역전파시의 국소적 미분의 의미는 다음과 같다.

i. 신호 E에 노드의 국소적 미분($\frac{\partial y}{\partial x}$)을 곱하여 다음 노드로 전달하는 것.

ii.



4. 덧셈 노드의 역전파

A. 덧셈 노드의 역전파는 입력 값을 그대로 흘려 보낸다.

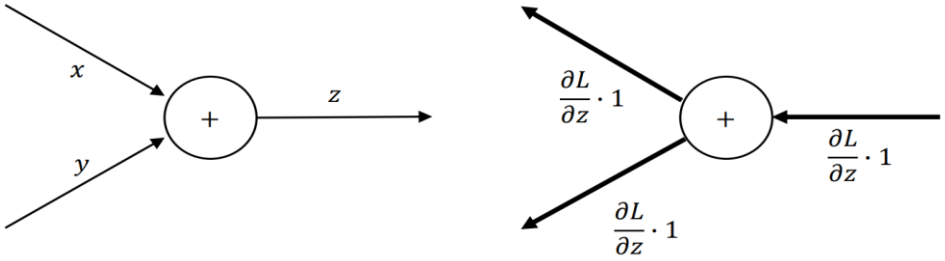
i. 노드가 덧셈을 수행한다면, 함수가 갖는 매개변수의 차수가 높아지지 않는다. 즉, 함수의 미분 된 값이 항상 1이라는 것이다.

ii. $z = x + y$를 미분한다면 다음과 같이 해석적인 계산이 가능하다.

iii. $\frac{\partial z}{\partial x} = 1$: x를 변수로 하여, z를 미분한 경우.

iv. $\frac{\partial z}{\partial y} = 1$: y를 변수로 하여, z를 미분한 경우.

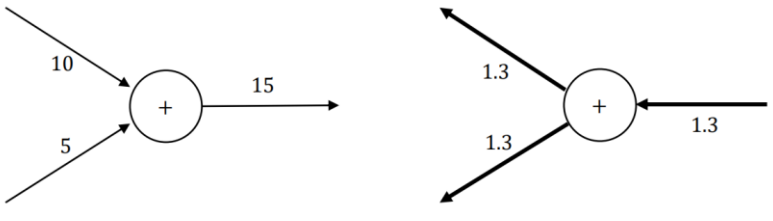
v.



vi. 이때 L은 손실 함수를 의미한다. 즉 위의 그림은 다음 설명과 같다.
z의 출력 값(손실함수의 입력)을 변수로 하여,
손실 함수(L)를 미분한 값을, (z를 변수로 하여 L 미분)
덧셈 노드에 역전파 함. (연쇄법칙대로 미분된 값인 1을 곱하여 전파.)

B. 그리하여 덧셈 노드의 역전파는, 모든 입력 신호를 그대로 통과시킨다.

i.



5. 곱셈 노드의 역전파

A. 곱셈 노드의 역전파는 입력 값에 [[순전파의 입력 신호]를 자리 바꾸어 곱한 값]을 전달한다.

i. $z = xy$를 미분한다면 다음과 같이 해석적인 계산이 가능하다.

ii. $\frac{\partial z}{\partial x} = y$: x를 변수로 하여(x에 대하여), z를 미분한 경우.

iii. $\frac{\partial z}{\partial y} = x$: y를 변수로 하여(y에 대하여), z를 미분한 경우.

B. 역전파가 연쇄법칙에 의하여 기능하므로,

우리는 이 미분 값을 순전파의 입력 값에 곱하여 전달해야 한다.

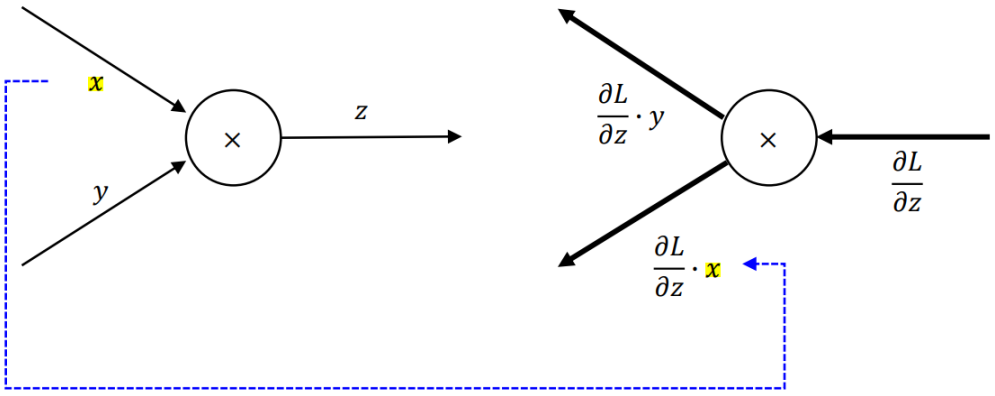
이때 x에 대하여 미분 한 값을, x가 입력된 방향으로 전달하고,

y에 대하여 미분 한 값은, y가 입력된 방향으로 전달해야 한다.

i. 이를 위해서는, 역전파시의 상류의 값인 [손실함수]의 미분 된 값을, 순전파시의 입력 값들의 자리를 바꾼 값들과 곱하여 역전파 해야 되는 것이다!

C. 예시는 다음과 같다.

i.



D. 요약!

i. 순전파의 입력 값에 대하여 미분한 상류의 값을 역전파 받아서 곱하면, 역전파 되는 상류의 미분 값에, 순전파의 입력 값의 자리를 바꾸어 곱한 것과 동일하다!

ii. $\frac{\partial z}{\partial x} = y$ 이므로 x가 입력된 방향에 y를 곱하는 것이다!

iii. $\frac{\partial z}{\partial y} = x$ 이므로 y가 입력된 방향에 x를 곱하는 것이다!

E. 한편, 곱셈 노드의 역전파시에, 순전파시의 입력 값이 필요하므로, 이 입력 신호들은 별도의 변수에 저장해 두어야 한다.

6. 활성화 함수 계층 구현

A. 이제 계산 그래프를 신경망에 적용할 수 있다. (2 ~ 5번은 역전파를 구현하기 위한 배경지식이다.)

B. 활성화함수는 신경망의 출력 신호를 만드는 함수이다.
입력 신호의 총합이 활성화를 언제 일으킬지를 포함하여 정의하게 된다.

C. 이전에 등장했던 ReLU와 Sigmoid 함수가 어떻게 역전파 되는지 알아보자.

7. ReLU 계층에서의 역전파

A. ReLU 함수는 다음과 같이 정의된다.

i. $y = \begin{cases} x\ (x > 0) \\ 0\ (x \leq 0) \end{cases}$

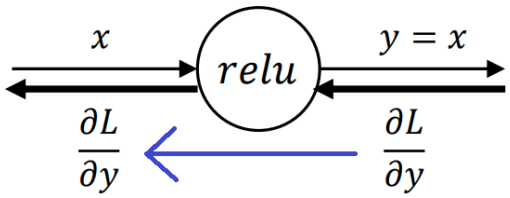
ii. 0 초과의 값은 그대로 출력하고, 0 이하의 값은 0을 출력한다.

B. ReLU 함수를 미분하면 다음과 같다.

i. $\frac{\partial y}{\partial x} = \begin{cases} 1\ (x > 0) \\ 0\ (x \leq 0) \end{cases}$

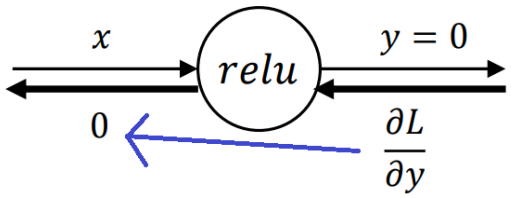
ii. 즉, ReLU 계층(노드)는 x의 값에 따라서
입력을 그대로 역전파 하거나,
0을 역전파 하게 된다.

iii.



입력을 그대로 통과시킨다.

$x > 0$



0을 역전파 한다

$x \leq 0$

C. 즉, ReLU는 스위치와 유사하다.

i. 순전파시에 데이터가 흐르고 있다 -> 역전파시에 데이터 흐름

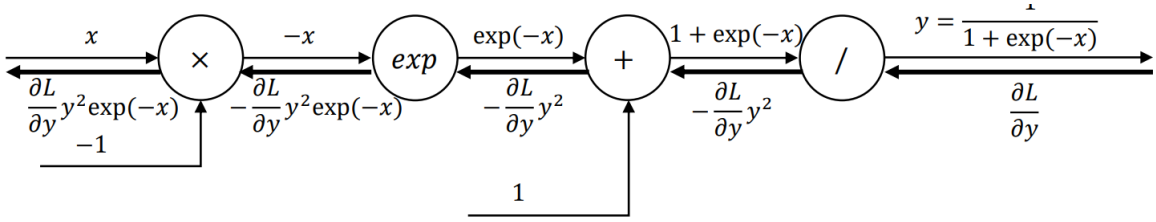
ii. 순전파시에 0이 전달되었다 -> 역전파시에 0이 전달된다.

8. Sigmoid 계층의 역전파

A. Sigmoid 함수는 다음과 같이 정의된다.

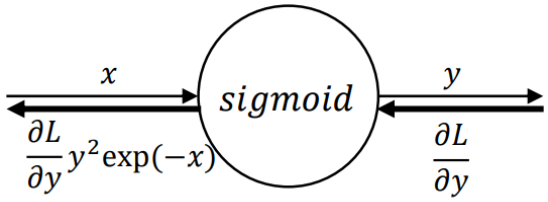
i. $y = \frac{1}{1+\exp(-x)}$

B. 위의 함수를 계산 그래프로 표현하면, 역전파를 손으로 계산해 볼 수 있다.

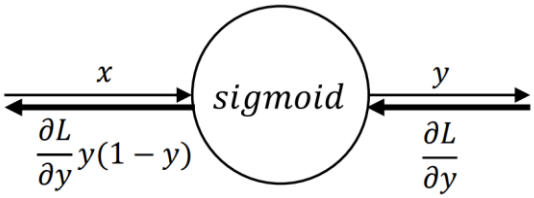
i. 

ii. 자세한 과정은 교재 p.167 ~ 169를 참고하길 바란다.

iii. 결과적으로 역전파의 중간 과정을 모두 생략하면 다음 그림과 같다.

iv. 

〈- 중간 과정 생략.

v. 

〈- iv의 결과를 정리함.

C. 결론

i. 역전파의 결과물을 보라. $\frac{\partial L}{\partial y}y(1-y)$

(y에 대하여 미분 된 손실함수‘ $\frac{\partial L}{\partial y}$ ’에 ‘$y(1-y)$’를 곱하면 역전파 결과)

y에 대한 식으로 정리된 것이 보이는가? 즉,

Sigmoid 계층의 역전파는 순전파의 출력 y만으로 계산할 수 있다!

9. numpy.shape( )

A. 이 병신 같은 함수는 굉장히 이상한 표기법을 사용한다.

B. 인자를 하나만 넣을 때는 '열'의 개수만을 입력받고,
인자를 두 개 넣을 때는 '행'과 '열'의 개수를 모두 입력받는다.

C. shape(8,) : 1x8 행렬

D. shape(2, 4) : 2x4 행렬

E. 위치가 반전된다니… 누구 머리에서 이런 더러운 방식이 탄생했는지 궁금하다. 내부적으로 함수를 잘 정의해서 쉼표 없이 8만 입력했을 때 1x8 행렬을 만들도록 해야지, 이걸 이런 방식으로 때워버리면 어떻게 하나? 게을러 빠진 모습이 심히 보기 불편하다.

10. Affine 계층

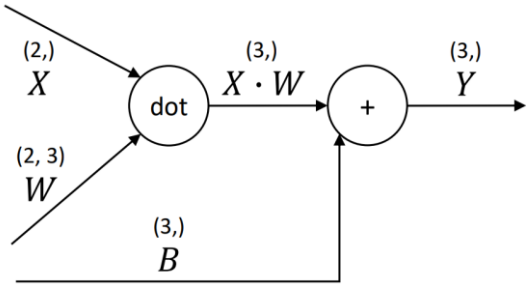
A. Affine 변환

i. 행렬의 형상을 변환하는 것을 말한다.
(이 Affine 변환을 수행하는 계층을 Affine 계층이라 말한다.)

B. Affine 변환의 의미

i. 신경망에서는 행렬 곱을 수행한다는 것이다!

ii. 행렬 곱을 수행하고, 출력된 행렬에 편향 행렬을 더하여 최종적인 출력 행렬을 만든다.

iii. 

B는 편향이다. Y는 출력.

iv. 이때 각 변수는 모두 행렬이다!

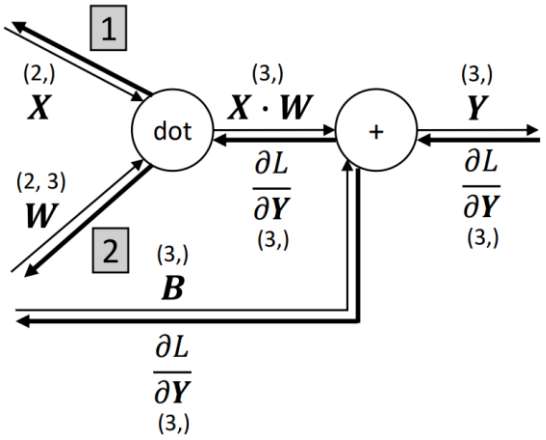
11. Affine 계층의 역전파

A. 모든 변수가 행렬이므로, 행렬 곱 연산을 할 때, 행렬의 형상을 잘 조절해야 한다. (행렬의 형상 조절을 위하여 전치행렬을 만들고, 행렬 곱의 순서를 조절해야 한다.)

B. Affine 계층은 $Y = XW + B$ (X: 입력, W: 가중치, B: 편향)와 같은 함수로 표현될 수 있고, 이 함수를 계산 그래프로 표현하면 다음과 같다.

1 $\underset{(2,)}{\frac{\partial L}{\partial X}} = \underset{(3,)}{\frac{\partial L}{\partial Y}} \cdot \underset{(3,2)}{W^T}$

2 $\underset{(2,3)}{\frac{\partial L}{\partial W}} = \underset{(2,1)}{X^T} \cdot \underset{(1,3)}{\frac{\partial L}{\partial Y}}$



i.

ii. 곱셈 노드의 역전파와 마찬가지로, 역전파시의 상류는, 하류의 값들의 위치를 서로 바꾼 후 곱해진다.

iii. 이때 행렬 곱 노드의 역전파가 가능하도록, 어떻게든 행렬의 형상을 맞추어야 한다. 우선, 전치행렬을 만들게 되는데, 전치행렬과 상류의 행렬이 곱셈이 불가능한 경우에는, 곱셈의 순서 마저도 바꾸어도 괜찮은 것이다.

iv. 이처럼 행렬 곱이 가능하도록 순서를 바꾸어도 되는 이유는, 역전파 과정에서 손실 함수를 최소화 하는 방향으로 매개변수를 바꾸기만 하면 되기 때문이다. 실제 값이 계산 그래프에서의 이상적인 역전파 과정을 따를 필요가 없다. 기울기의 방향성만 맞으면 되기에, 전치행렬로 바꾸고 행렬 곱 순서를 마구 바꾸어도 학습이 된다!

v. 이때, 역전파 된 1 $\frac{\partial L}{\partial Y} \cdot W^T$를 보라. 이 행렬은 X와 형상이 동일하다.
(X의 형상 = $\frac{\partial L}{\partial Y} \cdot W^T$의 형상 = $\frac{\partial L}{\partial X}$의 형상)

vi. 또한 2 $X^T \cdot \frac{\partial L}{\partial Y}$ 를 보라. 이 행렬은 W와 형상이 동일하다.
(W의 형상 = $X^T \cdot \frac{\partial L}{\partial Y}$의 형상 = $\frac{\partial L}{\partial W}$의 형상)

C. 배치 학습용 Affine 계층의 경우에는, 입력인 X의 형상이 (N, 2)가 됨.
(N: 배치 크기. N개의 데이터를 배치 단위로 묶은 경우.)

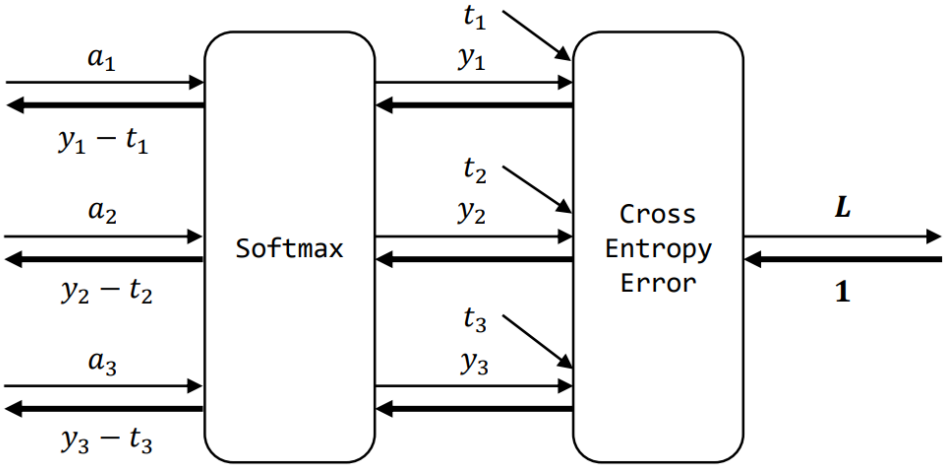
12. Softmax-with-Loss 계층

A. Softmax 함수는 입력 값을 정규화 한다. (점수를 확률로 변환.)

B. Softmax 함수는 대소관계를 바꾸지 않기에, 때에 따라서는 필요치 않을 수도 있다. 하지만, 학습을 위해서는 Softmax 계층이 꼭 필요하다!

C. Softmax 계층을 간소화 하면 아래 그림과 같다.

D.



i. a_n: 입력 (n번 클래스에 대한 점수)

ii. y_n: a를 정규화 한 값 (n번 클래스가 정답일 확률)

iii. t_n: 정답 레이블 (n번 클래스가 정답이면 1, 오답이면 0)

E. 즉, 입력인 a를 받아서, 확률로 정규화 시킨 y를 출력하는 함수이다.

F. 이때 역전파 된 값들이 $(y_1 - t_1, y_2 - t_2, y_3 - t_3)$ 으로 깔끔하게 떨어지는 것을 보라. 이는 손실 함수로 "Cross Entropy Error"를 사용했기 때문이다. (Softmax와 CEE를 함께 사용할 때 역전파의 결과가 깔끔하게 계산되도록 설계되었기 때문이다.)

i. 분류 문제의 경우: Softmax + "Cross Entropy Error"

ii. 회귀 문제의 경우: 항등함수 + 오차제곱합

G. 위의 두 경우 모두 역전파시에 동일한 형태의 데이터$(y_n - t_n)$가 흐른다.

i. '정답 확률 - 정답 여부'가 역전파 되므로, 정답에서 멀수록 더 큰 오차가 전파된다.

ii. softmax: 0.2 -> 정답:1 + CEE

iii. 위의 경우 softmax 왼쪽으로 역전파 된 값: 0.2-1 = -0.8

13. 신경망 학습

A. 오차역전파법으로 기울기를 계산하여, 이전에 설명한 대로 학습을 진행하

면 된다. 다만, 오차역전파법의 구현이 올바르게 되었는지 수치 미분된 값과의 비교가 필요하다. 이때 0에 가까운 아주 작은 차이가 보인다면 정상적으로 구현이 완료 된 것이다.

B. 신경망 학습 과정

i. 데이터 준비: 미니배치 단위로 데이터 선별

ii. 기울기 산출: 오차역전파로 각 가중치 매개변수의 기울기를 산출한다.

iii. 매개변수 갱신: 기울기 방향으로 가중치 매개변수를 조금 갱신한다.

iv. 반복: 기울기 산출과 매개변수 갱신 단계를 목표한 수준까지 반복한다.

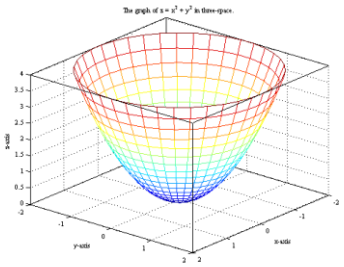
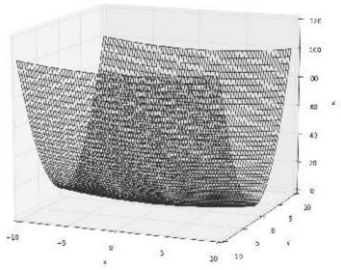
4주차 강의 요약 끝. (5장 정리 끝.)

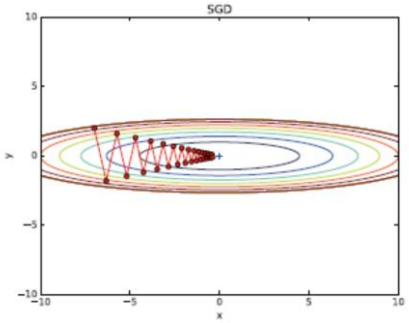
5주차 강의 요약 시작

1. 6장
   A. 6장의 내용은 크게 두 부류로 나뉜다.
      i. 매개변수 갱신 방법들에 대한 소개. (SGD-확률적 경사 하강법-가 아닌 방법들. 각 방법에 우열은 없다. 상황에 맞는 방법이 있을 뿐.)
      ii. 오버피팅을 억제하는 방법들에 대한 소개.

2. 신경망 복습
   A. 신경망 학습의 목적: 손실 함수의 값을 가능한 낮추는 매개변수 찾기.
   B. 신경망 학습 = 매개변수의 최적 값을 찾는 문제. = 최적화(optimizer)

3. SGD의 단점
   A. 우리는 이미 SGD를 배웠다. SGD는 분명히 좋은 방법(구현이 쉽고 단순함, 성능도 나쁘지 않음)이지만, 문제에 따라서 비효율적인 경우가 있다.
   B. 특히 비등방성 함수에 대하여 SGD가 비효율적인데, 이는 비등방성 함수의 매개변수에 대한 기울기가 가리키는 지점이 어느 한 곳으로 수렴되지 않기 때문이다.

	
등방성 함수 예시	비등방성 함수 예시

   C. 그리하여 학습 과정이 비효율적인 경우가 있다.
   D. 학습 과정 (손실함수가 최소값이 되는 지점까지 매끄럽게 이동 못함.)
      i. 

4. 모멘텀(Momentum)

A. SGD를 대체할 첫 번째 방법! 이것은 SGD에 관성을 더하는 것이다.

i. 누적된 힘으로 가속이 되고, 상시로 받는 저항이 존재한다.

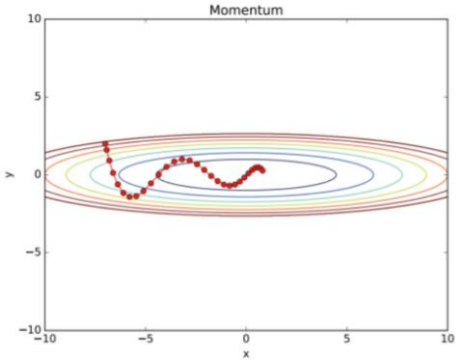
ii. 즉, 이전 단계의 업데이트가, 현재 단계에 영향을 주도록 하는 것이다.

B. SGD와 모멘텀은 각각 다음과 같이 표현된다.

i. SGD: $W \leftarrow W - \eta \frac{\partial L}{\partial W}$ (기울기 * 학습률 만큼 가중치W 갱신)

ii. 모멘텀: $W \leftarrow W + (av - \eta \frac{\partial L}{\partial W})$ (v는 물리에서의 속도)

C. 모멘텀 적용 결과

i. 

따로 무언가를 더 설명 할 이유는 없는 듯.

5. AdaGrad

A. 학습률을 점차 감소시키는 기법(learning rate decay)이 있다. 이 방법을 발전시킨 것이 AdaGrad이다.

i. 개별 매개변수에 적응적으로 학습률을 조정하며 학습을 진행하는 방법.

B. AdaGrad는 다음과 같이 표현된다.

i. $h \leftarrow h + \frac{\partial L}{\partial W} \odot \frac{\partial L}{\partial W}$ (기울기 값을 제곱하여 더한다)

ii. $W \leftarrow W - \eta \frac{1}{\sqrt{h}} \frac{\partial L}{\partial W}$ (기울기 x 학습률 x $\frac{1}{\sqrt{h}}$ 로 학습률 갱신)

iii. $\odot$: 행렬의 원소별 곱셈 연산자

C. $\frac{1}{\sqrt{h}}$을 곱하는 것의 의미

i. 크게 갱신된 매개변수는, 다음 학습에서 학습률이 크게 낮아진다. (매개변수마다 서로 다른 정도의 학습률 감소가 일어난다.)

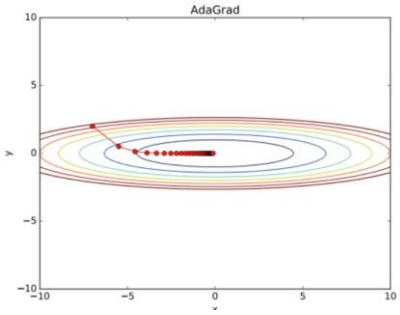
ii. 학습을 진행할수록 갱신 강도가 계속하여 약해진다. (학습률이 빠르게 감소한다는 말이다. 해결해야 할 문제점이다.)

D. RMSProp

i. 학습을 무한히 이어나가면, 어느 순간 갱신량이 0이 된다는 문제를 해결할 수 있다.

ii. RMSProp은 먼 과거의 기울기를 서서히 잊도록 설계된 기법이다. (새로운 기울기 정보를 더 크게 반영한다.)

E. AdaGrad 적용 결과

i. 

ii. 첫번째 갱신을 보라. 아주 큰 폭으로 갱신이 이루어졌다.
그리고 큰 움직임에 비례하여 다음 갱신도 크게 감소했다.

iii. 이전의 갱신이 크다면, 다음 갱신은 많이 줄이고,
이전의 갱신이 작다면, 다음 갱신은 작게 줄어든다….

6. Adam

A. 모멘텀과 AdaGrad를 융합한 기법이다.

B. 일반적으로 Adam이 가장 많이 사용되는 매개변수 갱신 방법이다.
(SGD를 추가로 사용하여 Adam과 비교하는 경우가 많다.)

C. 교재에 설명이 거의 없다. 교재에는 이론이 복잡하다고 서술되어 있다.

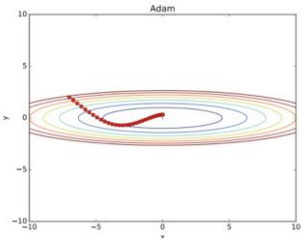
D. 특징

i. 하이퍼파라미터의 '편향 보정'이 진행된다.

ii. 3개의 하이퍼퍼라미터를 사용한다. (학습률, 1차 모멘텀 계수, 2차 모멘텀 계수)

iii. 매개변수 공간의 탐색이 효율적.

E. 적용 예시

i. 

7. 가중치의 초기값

A. 가중치의 초기값에 따라서 신경망 학습이 성공하기도, 실패하기도 한다.

B. 0으로 설정하면 안된다!

(더 정확히는, 모든 가중치가 균일하면(같으면) 안된다!)

i. 오차역전파법에서 모든 가중치가 똑같이 갱신되기 때문! (역전파시에 이전과 동일한 값을 전파하게 된다.)

C. 가중치의 대칭적 구조를 무너뜨려야 학습이 정상적으로 진행될 수 있다!

i. 즉, 가중치의 초기값을 무작위로 설정해야 함!

8. 은닉층의 활성화 값 분포

A. 가중치의 초기값에 따라서, 은닉층의 활성화 값(활성화 함수들의 출력 데이터)들이 어떻게 변화하는지 이해할 필요가 있다.

B. 가중치의 초기값과 표준편차

i. 가중치의 초기값은 정규분포로부터 생성할 수 있다.

```
0 + 0.01 * np.random.randn(10, 100)
//평균 0 + 표준편차 0.01 * 10x100형상의 정규분포를 따르는 가중치 행렬 생성.
```

ii. 이때 이 정규분포의 표준편차를 설정하는 것으로 초기값을 변경할 수 있다.

iii. 표준편차가 커질수록 데이터는 평균값에서 멀리 퍼져 있다.

iv. 표준편차가 작아질수록 데이터는 평균 근처에 밀집해 있다.

C. 은닉층의 활성화 값들이 치우친 경우는 다음 두 가지가 있다.

i. 0과 1에 치우친 경우: 기울기 소실 문제 발생.

ii. 특정 값에만 몰려 있는 경우: 표현력 저하 발생. (깊은 신경망의 장점 소실.)

D. 기울기 소실 문제

i. 표준편차를 극히 줄이면 모든 값이 평균에 극히 가까워진다. 이때 평균이 0또는 1에 가깝다면? 기울기 소실 문제가 발생한다.

ii. 또한 표준편차가 너무 크면 활성화 값들이 0와 1에 치우치게 된다. 일반적으로 표준정규분포의 표준편차는 1이지만, 신경망 학습에서 1은 너무 큰 숫자이다. (표준편차가 1만 되어도 기울기 소실 발생!)

iii. 시그모이드 함수의 모양을 떠올려 보라. 0과 1에 가까운 부분은 기울기가 0에 다가간다.

iv. 특정 층의 활성화 값들이 0과 1에 몰려 있다. = 기울기가 0에 가깝다.

v. 즉, 역전파시에 기울기가 점점 사라지게 된다. -〉 학습 중단.

E. 표현력 제한 문제

i. 교재에서는 표준편차를 0.01로 설정한 예시를 보여준다.

ii. 가중치 값들이 0과 1이 아닌 특정 지점에 몰리게 된 경우, 기울기 소실은 발생하지 않지만, 거의 모든 뉴런이 비슷한 값을 출력하고 있다는 뜻이다.

iii. 즉, 뉴런의 개수가 줄어든 것과 유사한 효과가 발생한다. 이를 표현력 제한 문제라고 함.

9. Xavier 초기값

A. 많은 딥러닝 프레임워크에서 표준으로 사용하는 가중치 설정 방법이다.

B. 각 층의 활성화 값들을 광범위하게 분포 시키는 것이 목적이다.

C. 앞 계층의 노드가 n개일 때, 표준편차를 $\frac{1}{\sqrt{n}}$로 설정하는 방법이다.

D. 하지만 이 방법으로도 기울기 소실이나 표현력 제한 문제가 완벽히 해결되는 것은 아니다. 다만, 층을 지날 때 활성화 값 분포에 문제가 생길 가능성이 더 낮을 뿐이다.

E. Sigmoid보다 tanh(hyperbolic tangent)를 사용했을 때 활성화 값 분포가 더 안정적이라고 알려져 있다.

i. 두 활성화 함수 모두 선형이며 좌우 대칭이다. 〈- 이런 활성화 함수에 대하여 Xavier 초기값이 유효하다.

10. He 초기값
    A. 선형 함수를 사용하지 않는 경우에 일반적으로 He 초기값을 사용한다.
    B. ReLU는 대표적인 비선형 활성화 함수로서, He 초기값과 함께 사용 가능하다.
    C. 앞 계층의 노드가 n개일 때, 표준편차로 $\sqrt{\frac{2}{n}}$ 를 사용하는 초기값이다.

11. 배치 정규화 (2015년 제안됨)
    A. 지금까지 언급된 모든 방법을 사용해도, 기울기 소실이나 표현력 제한 문제가 발생할 수 있다. 초기값은 어디까지나 초기값일 뿐이고, 학습이 진행되며 가중치 매개변수가 계속 바뀌기 때문이다.
    B. 그렇다면, "학습을 진행하며 활성화 값 분포를 강제로 퍼지도록 하면 어떨까?" <- 이 아이디어가 딥러닝의 혁신 중 하나인 '배치 정규화'이다.
    C. 배치 정규화를 수행하는 층을 만들고, Affine 계층과 활성화 함수 사이에 추가한다.
        i. Affine - Batch Norm - ReLU -…반복…-Affine - Softmax ->
    D. 배치 정규화 층은 미니배치 단위로 데이터 분포의 평균이 0, 분산이 1이 되도록 정규화 한다.
    E. 자세한 정규화 과정은 교재 p.211의 [식 6.7]을 참고하길 바란다.
    F. 다양한 효과가 있다.
        i. 학습 속도 개선 (단, 항상 빨라지는 것은 아니다.)
        ii. 초기값에 덜 의존하게 된다.
        iii. 오버피팅을 억제한다.
    G. 초기값이 잘 분포되어 있지 않다면, 배치 정규화를 하는 것이 좋다고만 기억해두라.

12. 바른 학습

A. 오버피팅이 발생하면 범용 능력(일반화 성능)이 저하된다.

B. 오버피팅은 다음 두 경우 잘 발생한다.

i. 매개변수가 많고 표현력이 높은 모델 사용

ii. 훈련 데이터의 양이 적은 경우

C. 오버피팅을 억제하는 다양한 방법이 있다

i. 가중치 감소

ii. 드롭아웃

13. 가중치 감소

A. 큰 가중치에 대하여, 그에 상응하는 큰 패널티를 부과한다.
(가중치 매개변수의 값이 클 때 오버피팅이 잘 발생)

B. 가중치 감소 기법의 예시

i. L2 정규화: 손실함수 <- 손실함수 + L2 norm
(L2 norm: 벡터의 요소를 제곱하여 합한 후, 합한 값의 제곱근을 구함)
(L2 norm: $\frac{1}{2}\lambda W^2$ // λ: 정규화 강도를 조절하는 하이퍼퍼라미터)

C. 자세한 설명은 생략하겠다. (교재에도 없음)

14. 드롭아웃

A. 훈련할 때 은닉층의 뉴런을 무작위로 골라서 삭제하며 훈련하는 것.

i. 단, 시험 시에는 모든 뉴런에 신호를 전달한다. (어디까지나 학습 기법이라는 뜻.)

B. 앙상블 학습: 개별적으로 학습시킨 여러 모델의 출력을 평균 내어 추론하는 방식. <- 이를 적용하면 신경망의 정확도가 몇% 개선됨.

C. 드롭아웃은 앙상블과 같은 효과를 하나의 신경망으로 구현한 것으로 볼 수 있다.

15. 적절한 하이퍼파라미터 값 찾기

A. 하이퍼퍼라미터의 성능을 평가하기 위한 데이터를 분리해 두어야 한다. 이 때 사용하는 데이터를 '검증 데이터(validation data)'라고 부른다.
(시험 데이터로 하이퍼파라미터를 평가하면, 시험 데이터에 오버피팅 됨.)

B. 일반적으로 훈련 데이터의 20%를 검증 데이터로 분리해둔다.

i. 예시: MNIST의 경우 60,000개의 훈련 데이터 중, 48,000개만 훈련에 사용하고, 12,000개를 검증 데이터로 사용한다.

C. 하이퍼파라미터 최적화 과정

i. 하이퍼파라미터 값의 범위 설정.
ii. 설정된 범위에서 무작위로 하이퍼파라미터 설정.
iii. 앞서 설정한 파라미터로 학습하고, 검증 데이터로 정확도 평가.
iv. ii~iii를 특정 횟수 반복하며, 결과를 보고 하이퍼파라미터의 범위를 좁힌다.

D. 이는 다소 경험적으로 정리된 방법이다. 과학보다는 지혜에 가까운 것. (자세한 내용은 교재 p.224 참고.)

5주차 강의 요약 끝. (6장 정리 끝.)

6주차 강의 요약 시작

1. 서론

7장은 합성곱 신경망(CNN)에 대한 내용이다.

6장까지 배운 내용은 완전연결(fully-connectec) 신경망으로, Affine계층과 비선형 활성화 함수(ReLU, Sigmoid)를 번갈아 가며 여러 번 쌓은 꼴이다. (이러한 완전연결 신경망에서 Affine 계층은 데이터의 차원을 변경하고, 편향을 포함시키는 역할을 하며, 복잡한 패턴은 활성화 함수의 비선형 특징에 기대어 학습했다.)

2. 완전연결 계층의 문제

A. 신경망이 인접하는 계층의 모든 뉴런과 결합되어 있는 것을 완전연결이라 한다. 이런 방식으로 신경망을 구성하면 다양한 단점이 있다.

B. 데이터의 형상을 무시하고 1차원으로 변형한다는 문제가 있다.

i. 예: (1, 28, 28) -> 1x784

C. 위의 예시처럼 행렬의 형상을 변환하게 되면, 행렬 속 원소가 갖는 공간 정보(좌표)가 무시된다. (이미지는 3차원 데이터이며, 인접한 픽셀은 높은 연관성을 갖는다.) 즉, 모든 입력 데이터를 하나의 차원에 속한 뉴런으로 취급하기에, 형상에 담긴 정보가 소실된다는 것이다.

D. 모든 데이터를 직접 처리하기에, 학습 시간이 길어지고, 이미지의 회전 등에 영향을 받는 문제도 있다.

3. 완전연결 신경망 / CNN의 구조

A. 완전연결 신경망의 경우

i. [Affine - ReLU] <- 이것이 여러 번 반복됨.

ii. [Affine - ReLU] - [Affine - Softmax] (출력단)

B. CNN의 경우

i. [Conv - ReLU - Pooling] <- 이것이 여러 번 반복됨.

ii. [Conv - ReLU] - [Affine - ReLU] - [Affine - Softmax] (출력단)

iii. 새로 추가된 Conv, Pooling에 대하여 이후에 알아본다.

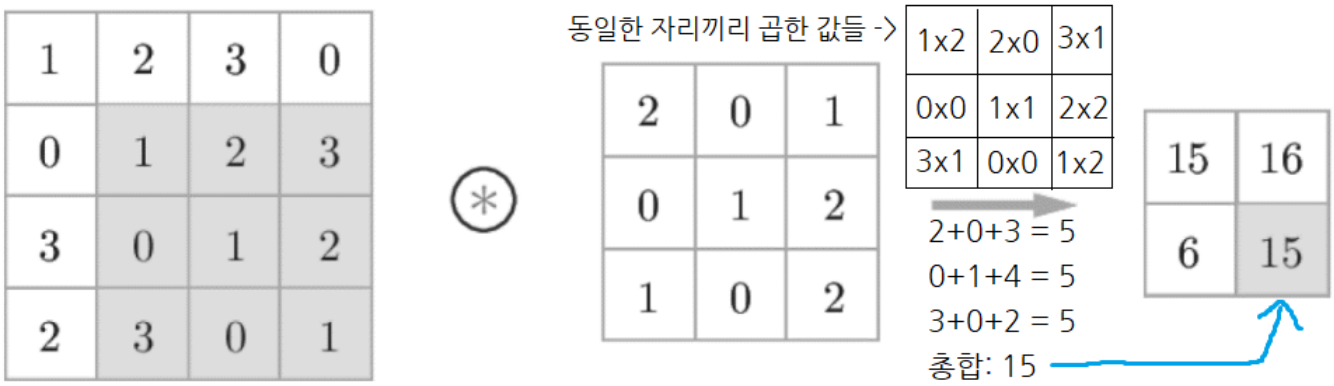
4. CNN의 합성곱 계층의 입출력 데이터

CNN에서는 입력과 출력 모두 '특징 맵(feature map)'이다. 이는 일종의 이미지로, 입력되는 특징 맵을 input-feature map, 출력되는 특징 맵을 output-feature map 이라 한다. (이 둘이 똑같이 이미지(행렬)라는 것을 기억하라.)

5. 합성곱 연산

A. [항목 3번]의 'Conv' 가 바로 합성곱 계층이다.

B. 합성곱의 과정은 다음과 같다. (필터 연산)



i. 이때 왼쪽의 행렬은 특징 맵이다.

ii. 합성곱 되는 행렬은 '커널' 혹은 '필터' 라고 불린다. (이 필터가 신경망의 가중치이다.)

iii. 우리는 이 커널을, 입력 특징 맵 전체에 대하여 합성곱 하여, 맨 오른쪽과 같은 출력 특징 맵을 만들게 된다.

iv. 그리고 이것이 합성곱 계층에서 하는 일이다.

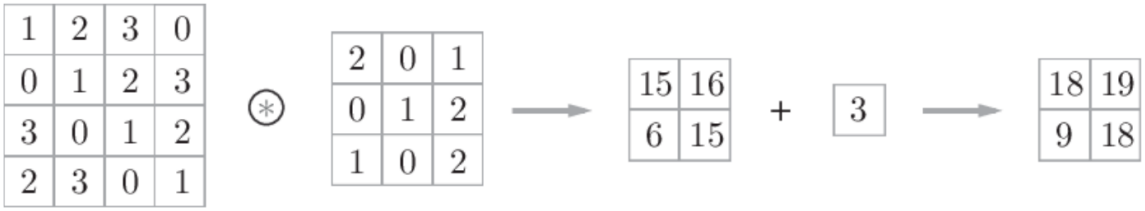
C. Affine 계층에서 '편향'을 더했던 것이 기억 나는가?

Conv 계층에서도 에도 동일한 작업을 한다.

i. 이 편향값은 1x1의 행렬로 생각해도 좋다.

ii. 다만 항상 필터 연산 이후에 만들어진 특징 맵 전체에 편향을 더해야만 한다. 아래 그림을 보라.

iii.



iv. 필터 연산의 결과물(3번째 행렬)의 모든 원소에 편향(3)이 더해져서

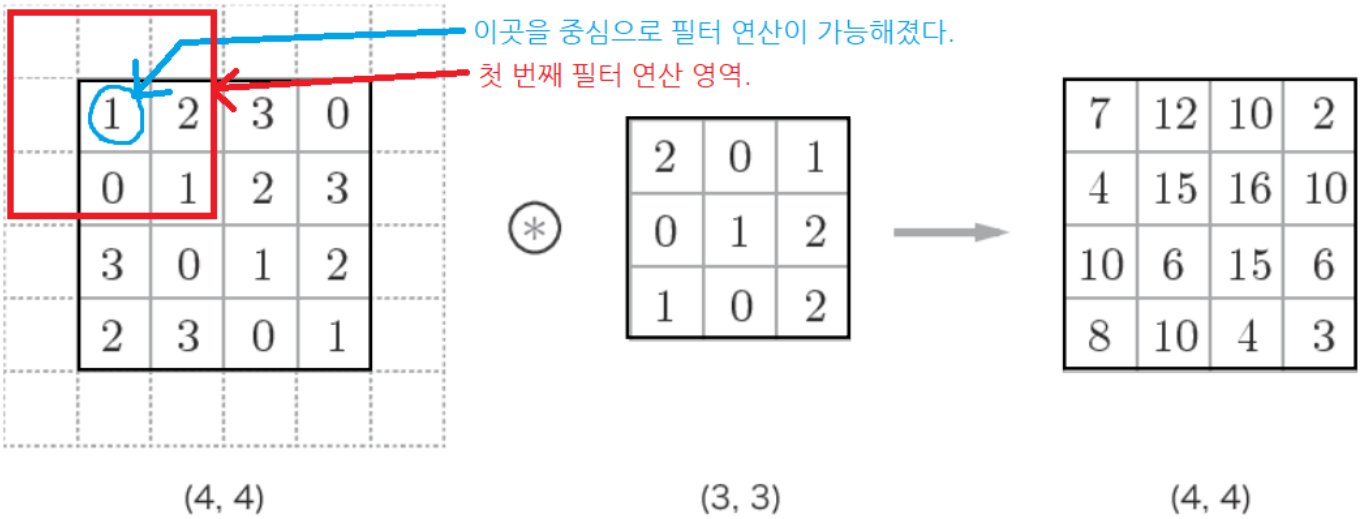
Conv 계층의 최종 출력 특징 맵이 만들어지는 것이다!

6. 패딩(padding)

A. [항목 5]에서 합성곱을 수행하면 행렬이 작아지는 것을 보았을 것이다. 이는 필터 연산을 수행할 때, 연산 위치가 원본 행렬의 외곽에 위치할 수 없기 때문이다.

-> 이를 해결하기 위해서, 원본 이미지의 외곽에 데이터를 추가해서, 필터 연산의 대상이 되는 행렬의 크기를 늘리는데 사용되는 것이 '패딩'이다.

B. 예시



i. 그림의 점선 부분이 '폭이 1인 패딩'을 적용 한 부분이다.

ii. 패딩을 적용하면, 원본 행렬의 크기를 유지하며 연산 할 수 있다.
(입력 데이터의 공간적 크기를 고정한 채로 다음 계층에 전달 가능.)

C. 패딩을 채우는 방법은 다양하지만, 책에서는 설명하지 않는다. 다만, 단순한 알고리즘에 의하여 빈 공간을 채우는 것 뿐이다. 어렵게 생각 할 이유는 없다.

7. 스트라이드(stride)

A. 필터를 적용하는 위치의 간격을 말한다.

B. 지금까지 예시로 본 필터 연산은, 모두 스트라이드가 1인 경우이다.

C. 스트라이드를 키우면, 필연적으로 출력되는 행렬의 크기가 줄어든다.

8. 합성곱 계층의 출력 피처 맵 크기

A. 우리는 이제 출력 피처 맵의 크기를 계산할 수 있다.

B. 영향을 끼치는 요소는 다음과 같다.

i. 입력 행렬 크기 (H, W) (행, 열)

ii. 필터 크기 (FH, FW)

iii. 패딩 크기 (P)

iv. 스트라이드 값 (S)

C. 출력 크기 (OH, OW)를 결정하는 공식은 다음과 같다.

i. $OH = \frac{H+2P-FH}{S} + 1 \ / \ \frac{행+2*패딩-필터_{행}}{스트라이드} + 1$

ii. $OW = \frac{W+2P-FW}{S} + 1$

iii. 복잡해 보이지만, 사실 별 것 아니다. 우리는 이미 출력 행렬의 크기가 어떤 방식으로 형성되는지 알고 있으므로, 주어진 조건대로 직접 출력 행렬의 행, 열 개수를 세어보면 된다. (필터의 스캔 작업을 수동으로 수행)

iv. 하지만 조금만 생각해 보면, 공식을 만들 수 있을 것이다.

D. 연습문제

i. 입력: (28, 31), 패딩: 2, 스트라이드: 3, 필터: (5, 5)

ii. 답: (10, 11)

iii. 하단에 풀어보시오

9. 3차원 데이터의 합성곱 연산

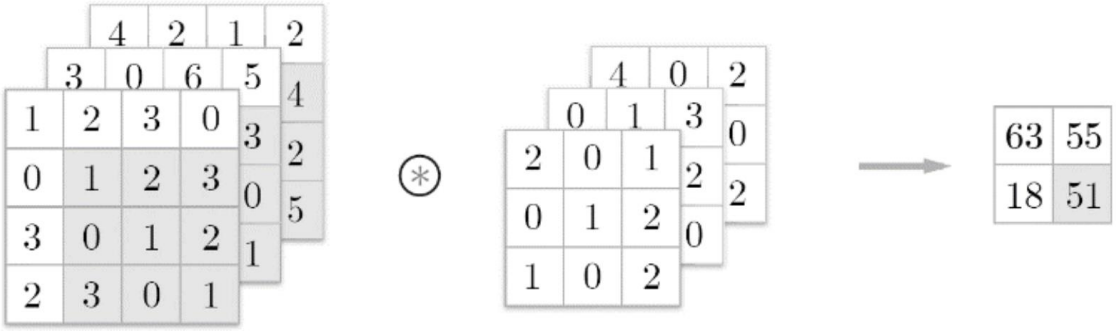
A. 컬러 영상은 3차원 데이터이다. 이는 하나의 이미지가 3개의 행렬로 구성되어 있다는 뜻이다.

B. 3차원 데이터를 합성곱 하려면, 각 차원에 사용할 필터가 정의되어 있어야 한다.

(항상 다음 조건 충족 필요: 입력 데이터의 채널 수 = 필터의 채널 수)

C. 그리고 각 채널별로 수행된 필터 연산의 결과물이 최종적으로 1차원 특징 맵이 된다는 점이 중요하다. (3개의 특징 맵에서 동일 위치의 값들을 더하여 새로운 하나의 행렬을 만든다.)

i.



10. 합성곱 연산을 블록으로 보자

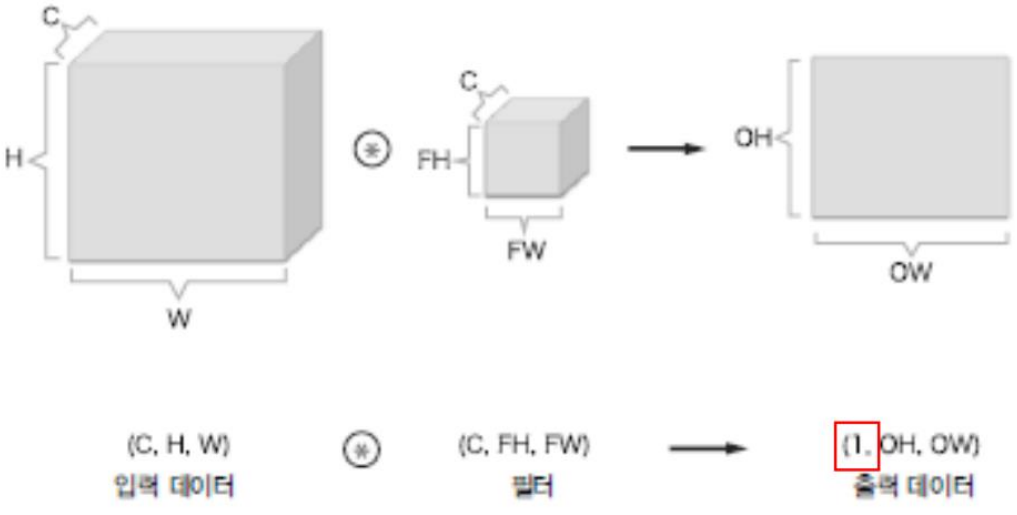
A. 앞으로 신경망의 구조를 설명할 때, 각각의 층의 현재 형상을 '블록'으로 표현한다.

B. 이는 단순히 다차원 배열을 육면체로 본다는 말이다.

i. 다차원 데이터: (채널 수 C, 높이 H, 너비 W)

ii. 다차원 필터: (채널 수 C, 필터 높이 FH, 필터 너비 FW)

iii.



11. 복수의 필터를 적용

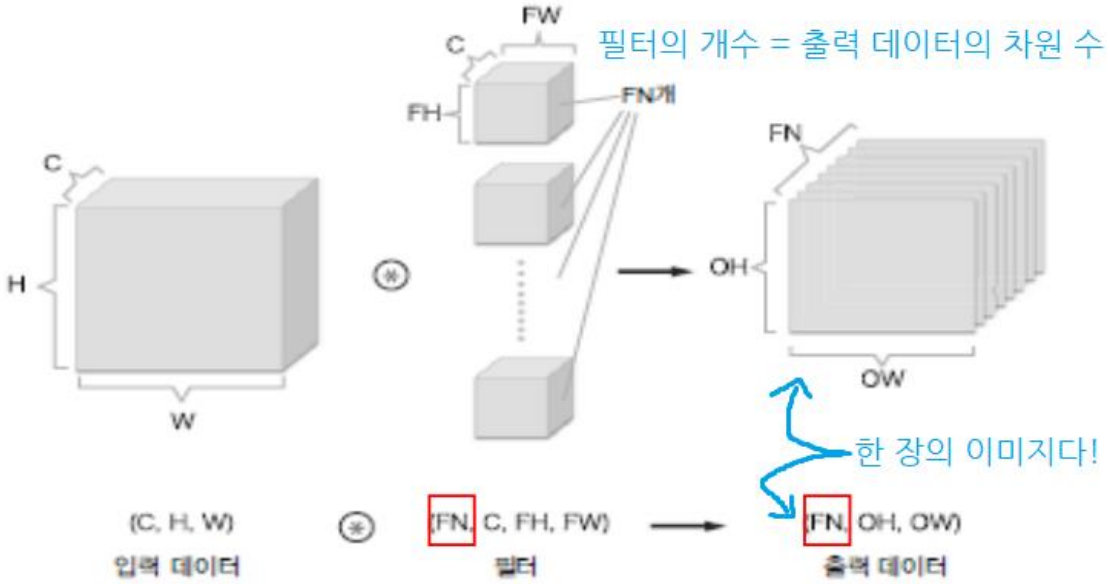
A. 합성곱 연산에 여러 장의 필터를 사용할 수 있다.

B. [10번]에서는 “3차원 이미지 1장 + 3차원 필터 1개 -> 1차원 특징 맵” 의 연산을 수행했다. 즉, ‘하나의 이미지’와 ‘하나의 필터’를 합성곱 하면 그들의 차원에 관계 없이, 하나의 ‘1차원 특징 맵’을 출력하는 것이다.

C. 그렇다면 필터의 개수가 더 많다면 어떻게 연산될까?

i. 필터의 차원 수 = 출력되는 특징 맵의 차원 수

ii. 즉, 다수의 필터에 의해 만들어진 복수의 이미지를, 한 장의 다차원 이미지로 취급하여 신경망의 다음 층에 넘긴다는 것이다!



iii.

iv. 이때 필터가 4차원 데이터임을 보라.

(출력 채널 수 FN, 입력 채널 수 C, 높이 FH, 너비 FW)

C, H, W 의 채널이 N개 있는 것이다.

12. 편향

A. 필터 연산의 결과물에 편향(1x1) 값을 더하여, 특징 맵으로 출력했던 것이 기억 날 것이다. 그렇다면, 합성곱 층이 다차원 데이터를 처리할 때, 편향 값을 어떻게 더할까?

B. 1x1의 편향 값이, N채널로 존재하도록 하여, 더하면 된다. (FN, 1, 1)

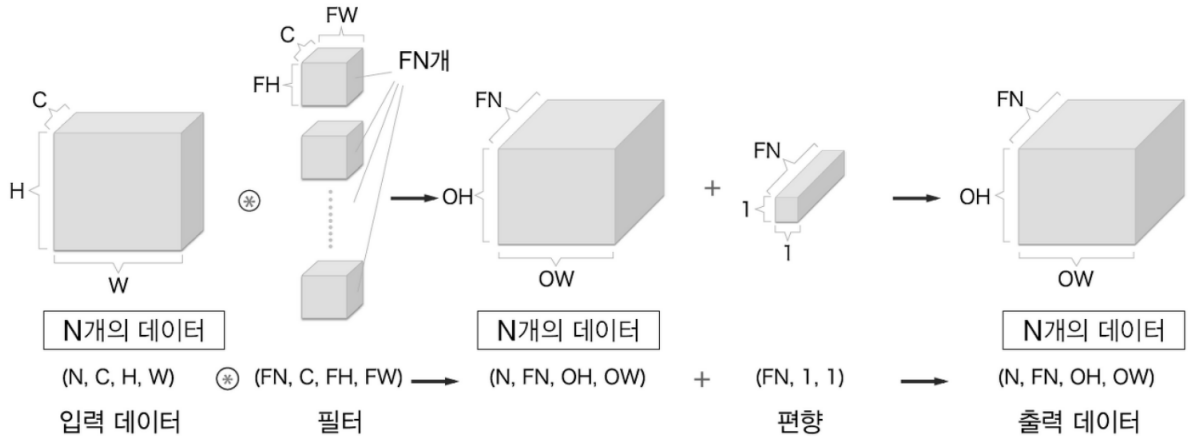
C. 즉, 필터 연산이 끝난 다차원 이미지의 모든 픽셀에, 1x1의 편향 값을 더하는 것이다. (본질적으로 [5번]의 내용과 완전히 동일함.)

13. 배치 처리

A. 지금까지 언급된 모든 과정을 배치 단위로 묶어서 수행하려면 어떻게 해야 할까?

B. 데이터의 차원 수를 하나 늘려서 처리하면 된다.

C.



D. 그림에서 입력 데이터, 필터 연산 결과, 합성곱 층의 출력(특징 맵)의 차원이 하나씩 늘어난 것을 볼 수 있다. -> 지금까지 설명한 연산을 N회분의 연산을 한번에 하면 된다.

14. 풀링 계층

A. 드디어 풀링 단계다. 13번까지의 항목은 주로 합성곱(Conv) 계층에 대하여 이야기했다. [3번] 항목의 내용 일부를 다시 가져오겠다.

i. [Conv – ReLU – Pooling] <- 이것이 여러 번 반복됨.

ii. [Conv – ReLU] – [Affine – ReLU] – [Affine – Softmax] (출력단)

B. 합성곱 계층이 출력한 특징 맵을 ReLU에 입력하고, ReLU의 출력이 Pooling계층에 투입된다.

C. 풀링: 이미지의 크기를 축소하는 연산. ($n*m$: 윈도우 크기)

i. $n*m$ max pooling: $n*m$ 영역에서 가장 큰 원소만 남긴다.

ii. $n*m$ average pooling: $n*m$ 영역의 평균 값을 남긴다.

iii. 2x2 max pooling의 예시는 다음과 같다.

iv.

v. 일반적으로 풀링은 영역이 겹치지 않게 수행한다.
즉, 윈도우 크기와 스트라이드의 값이 같다는 것이다.

D. 이미지 인식 분야에서는 주로 최대 풀링을 사용한다.

E. 특징

i. 풀링 연산은 단순한 수학적 계산이므로, 학습해야 할 매개변수가 없다.

ii. 풀링 연산은 1채널 이미지의 크기를 축소하므로, 채널의 개수가 변하지 않는다. (입력 데이터의 채널 수를 유지한다.)

iii. 입력의 변화에 영향을 적게 받는다. (강건하다. 입력 데이터가 조금 변해도 풀링의 결과는 큰 변화가 없다.)

15. 가중치의 개수

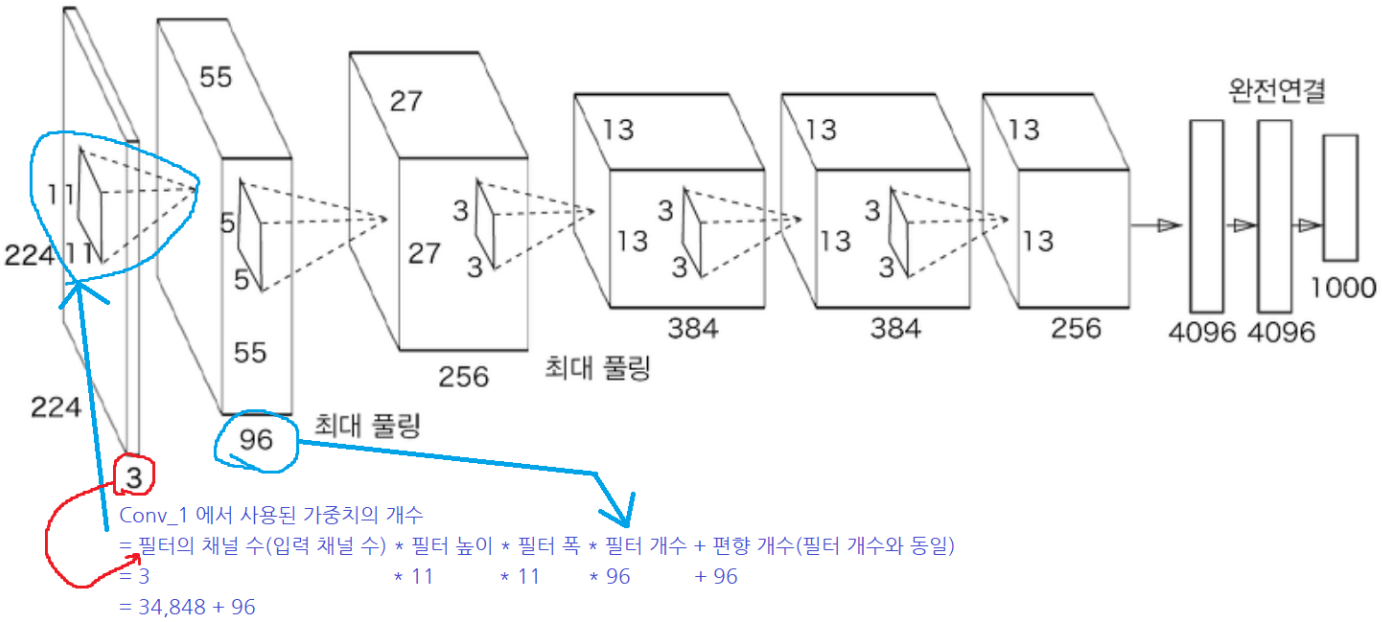
A. 필터도 가중치이고, 편향도 가중치이다.

B. 더 명확히 하자면, 필터와 편향에 포함된 모든 픽셀이 가중치이다.

C. 즉, 가중치의 개수를 묻는다면,
특정한 층에 존재하는 모든 매개변수의 개수를 묻는 것이고,
해당 층의 필터와 편향의 픽셀 수를 묻는 것이다.

D. 예시

[그림 7-28] AlexNet의 구성



i. 필터의 채널 수는 입력 이미지의 채널 수와 같아야 한다.

ii. 필터의 개수만큼의 이미지가 합성곱 결과로 출력되지만, 그 모든 이미지는 한 장으로 취급된다. (96개 필터 -> 96채널 이미지 1장)

iii. 편향의 개수는 필터의 개수와 동일하다. 1채널 전체에 더해지는 값이므로, 채널의 수와 동일한 것이다.

6주차 강의 요약 끝. (7장 정리 끝.)

7주차 강의 요약 시작

1. 서론
   A. 8장의 내용은 가볍게 이해하고 넘어가라.
   B. 딥러닝의 특징과 한계에 대하여 이야기하는 장이다.

2. 딥러닝
   A. 딥러닝은 층을 깊게 한 신경망이다. (심층신경망)
   B. 그러므로 딥러닝의 문제는, 더 많은 층으로 구성된 신경망을 만드는 문제이다.

3. 정확도 향상 기법
   A. 앙상블 학습, 학습률 감소, 데이터 확장 등의 방법이 있다. 앞의 두 개는 이미 배웠기에, 데이터 확장에 대하여 설명하겠다.
   B. 데이터 확장
      i. 입력 이미지를 알고리즘을 동원해 인위적으로 수정하여, 입력 데이터의 양을 늘리는 방법이다. (이를 '데이터 증강'이라고도 함.)
      ii. 일종의 전처리 기법이며, 오버피팅을 줄일 수 있는 방법이다.
      iii. 특별한 상황에 대한 강건함을 향상시킬 수 있다. (예: 어두운 환경)

4. 층을 깊게 하는 이유
   A. 이점 1. 신경망의 매개변수 수가 줄어든다. (연산 요구량이 감소.)
      i. 5x5 합성곱 연산 = 3x3 합성곱 연산 2회
      ii. 이때 행렬의 크기가 감소하면, 총 연산 요구량이 감소한다.
   B. 이점 2. 학습 효율성이 증가한다.
      i. 학습해야 할 문제를 분해하여, 각 층에 필요한 데이터의 양을 줄이는 것으로, 학습을 고속으로 수행할 수 있는 것이다.
      ii. CNN에서는, 신경망의 앞단에서는 단순한 패턴에 뉴런이 반응하고, 층이 깊어지면서 더 복잡한 패턴에 반응하게 된다. 이때 학습해야 할 문제를 분해한다는 것은, 인지하는 패턴을 더 세분화 하여 서로 다른 층에 할당할 수 있다는 것이다.

5. 딥러닝의 초기 역사
   A. ILSVRC 2012년도 대회에서 AlexNet이 압도적인 성적으로 우승하며 딥러닝에 대한 관심이 증가함.
   B. VGG 〈- 구성이 간단하고 성능이 썩 괜찮은 신경망. 존재만 알아두라.
   C. GoogLeNet 〈- 가로 방향의 '폭' 개념을 추가했다. 이를 인셉션 구조라고 한다.
   D. ResNet 〈- 층이 너무 많을 때 학습이 중단되는 문제를 해결했다.
      i. 스킵 연결: 입력 데이터를 합성곱 계층을 건너뛰어 출력에 바로 더하도록 함.
         (즉, 기존 신경망의 출력 $f(x)$를 $f(x)+x$ 로 대체한다는 뜻이다.)

6. 전이 학습
   A. 이미지넷 데이터를 학습한 많은 신경망들은 실제 제품에 활용해도 효과적이다. 이때 활용을 위해 기존에 학습된 가중치를 가져와서, 재 학습을 시킬 수 있다.
   B. 즉, 학습된 가중치를 복사하여, 그 상태로 재 학습(fine tuning)을 수행하는 것을 '전이 학습'이라 한다.
      i. 보유한 데이터셋이 적을 때도 유용하다.

7. 딥러닝 고속화
   A. 딥러닝에는 대량의 단일 곱셈 연산이 필요하다. -〉 GPU 쓰면 좋다 -〉 복수의 컴퓨터를 동원하면 더 좋다.
   B. 복수의 컴퓨터로 연산 = 분산 학습 = 수평 확장

8. 딥러닝의 활용 및 미래

A. 사물 검출(object detection)

i. 이미지 속에 담긴 사물의 위치와 종류(클래스)를 알아내는 것.

ii. = 사물 인식 + 위치 구분

B. 분할

i. 이미지를 픽셀 수준에서 분류하는 문제. (이미지의 모든 픽셀을 클래스별로 분류.)

C. 사진 캡션 생성

i. 이미지 분석(CNN) -> 자연어 처리(RNN)

D. 이미지 생성

i. GAN = Generator(이미지 생성 신경망) + Discriminator(이미지가 생성되었는지 아닌지 판별)

ii. 위의 두 신경망이 경쟁하여 이미지 생성을 더 정밀하게 학습.

E. 강화학습

i. 지도학습과 다른 분야로, 강화학습이 있다.

ii. 지도학습은 손실을 낮추도록 학습하지만,
강화학습은 보상을 늘리도록 학습한다.

iii. 강화학습만 다루는 책이 많을 정도로, 지도학습과 전혀 다른 분야.

7주차 강의 요약 끝. (8장 정리 끝.)

이로서 〈밑바닥부터 시작하는 딥러닝 1〉(한빛미디어 출판, 사이토 고키 지음)의 전체 범위 요약이 끝났다.