



Deep Learning

강사 : 구정은

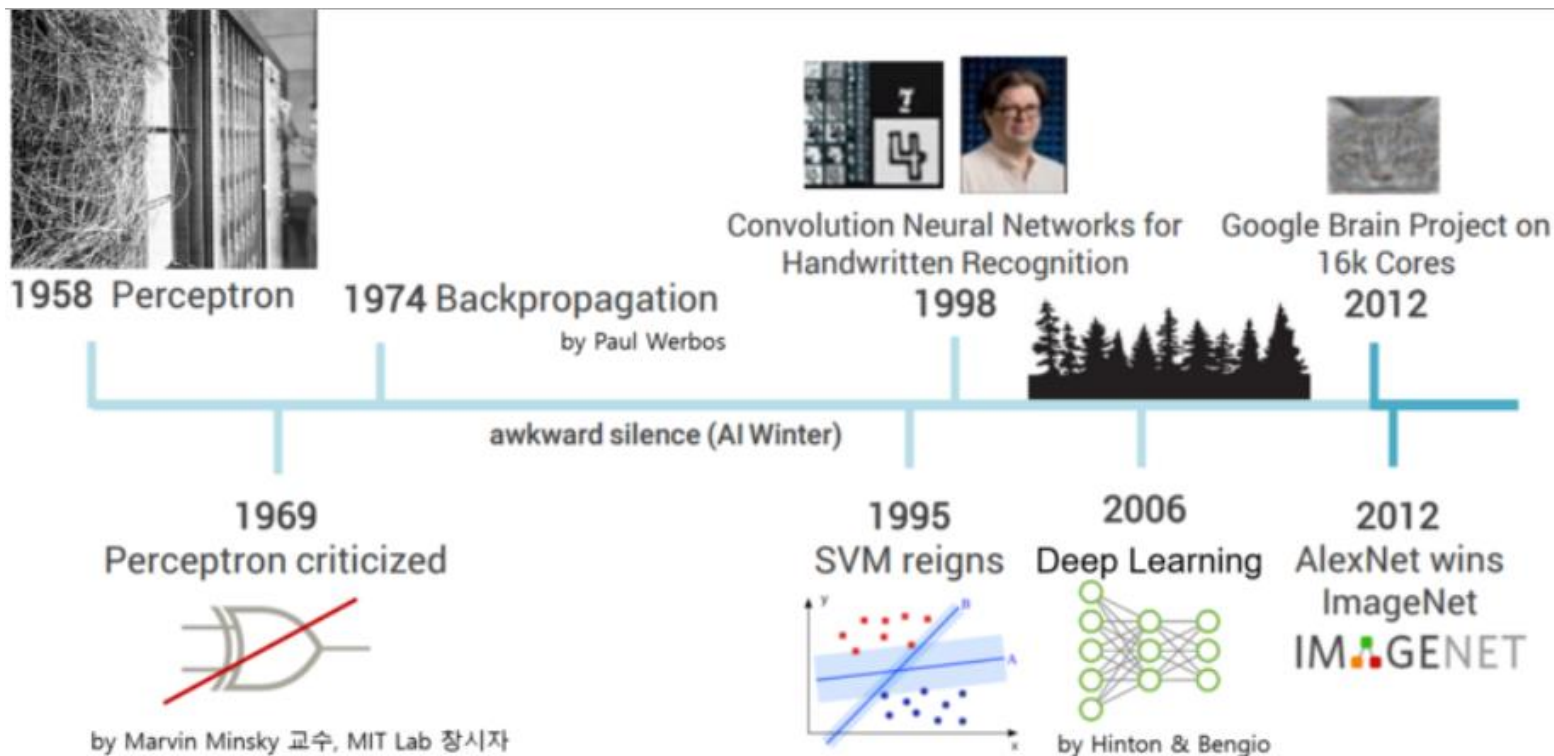


인공신경망

강사 : 구정은

인공신경망이론 Artificial Neural Network Theory

- 인경신경망 발전의 역사

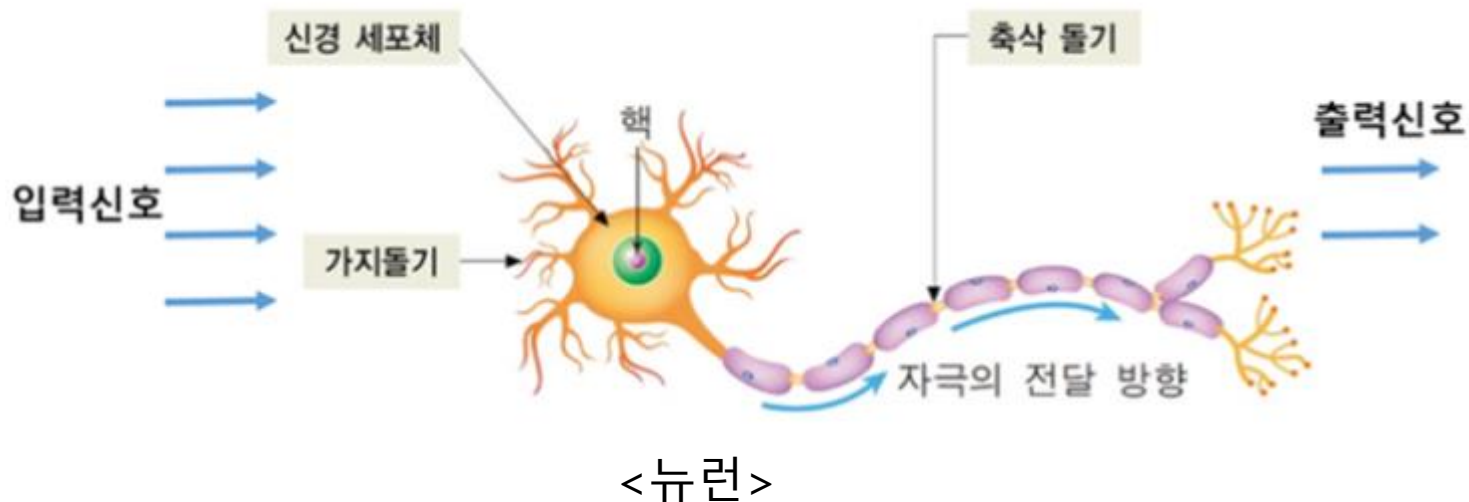


인공신경망이론 Artificial Neural Network Theory

1. 인공신경망 발전의 역사

- 1943년 McCulloch_Pitts(매컬로크-피츠) 뉴런

신경과학자인 Warren S.McCulloch과 논리학자인 Walter Pitts는 하나의 뇌 신경세포를 하나의 이진(Binary) 출력을 가지는 단순 논리게이트로 설명했는데. 이를 **McCulloch_Pitts(매컬로크-피츠) 뉴런(MCP 뉴런)**이라고 부른다.



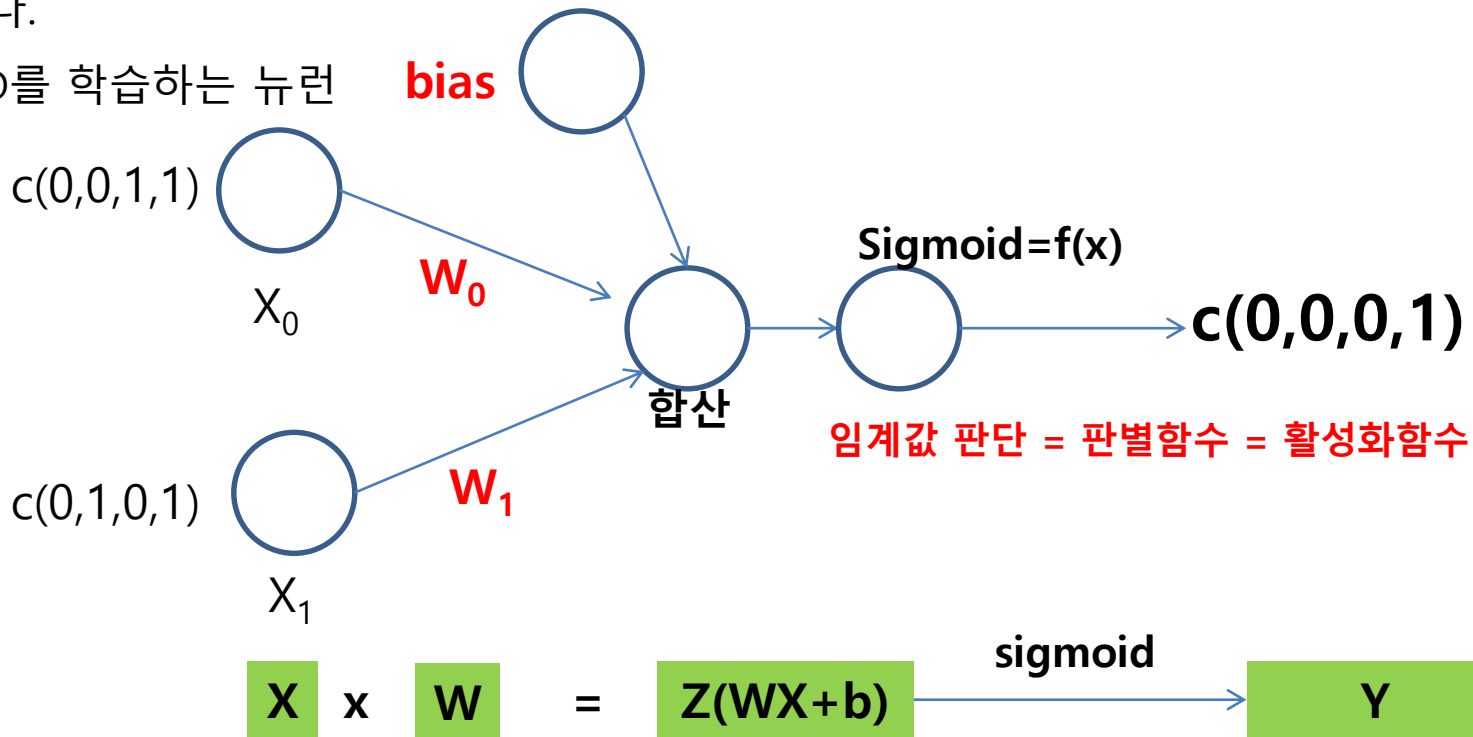
인공신경망이론 Artificial Neural Network Theory

1. 인공신경망 발전의 역사

- 1943년 McCulloch_Pitts(매컬로크-피츠) 뉴런

-앞선 이론을 기반으로 실제 만들면 AND, OR, NOT 같은 선형처리가 잘 되는 신경망이 탄생되었다.

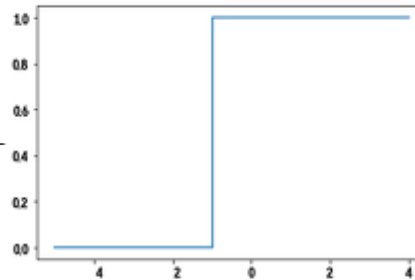
- 예) AND를 학습하는 뉴런



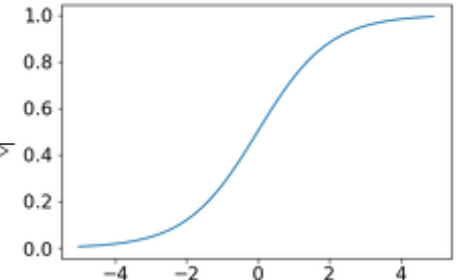
인공신경망이론

2. 퍼셉트론(Perceptron)

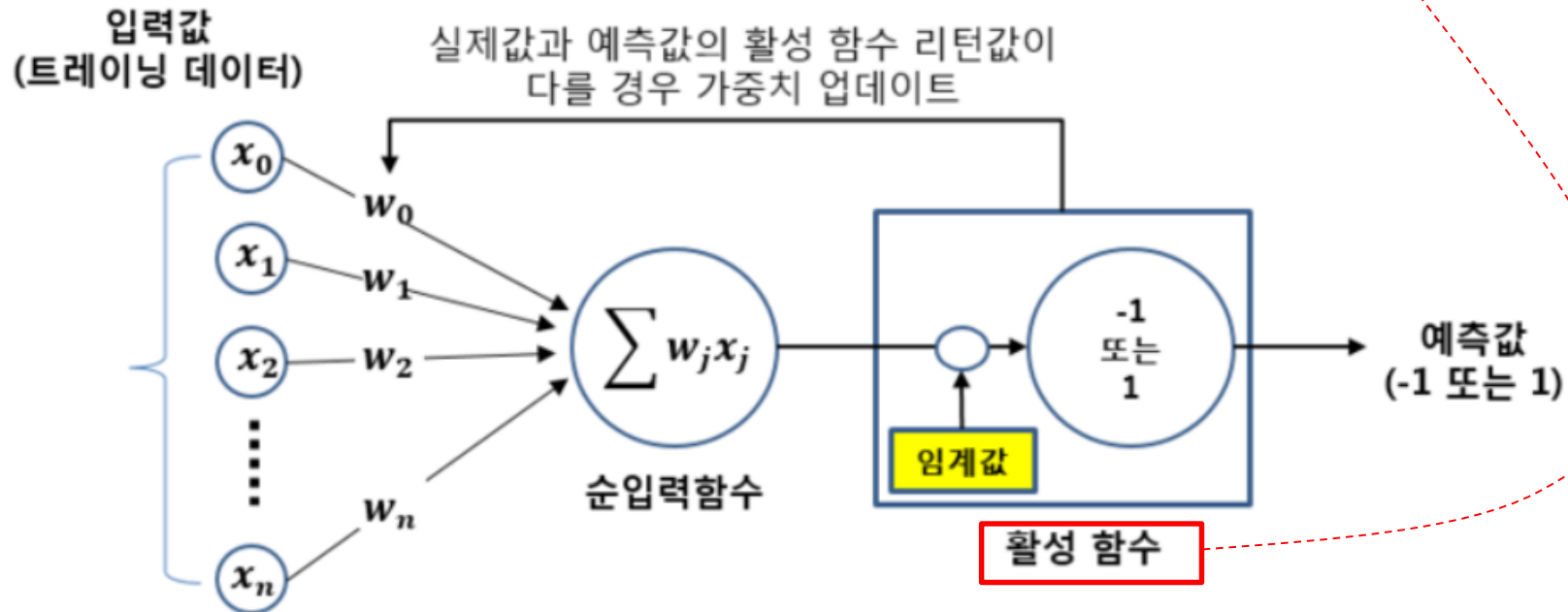
- 퍼셉트론



$$\text{Step Function } (y) = \begin{cases} 1 & (x \geq 0) \\ 0 & (x < 0) \end{cases}$$



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

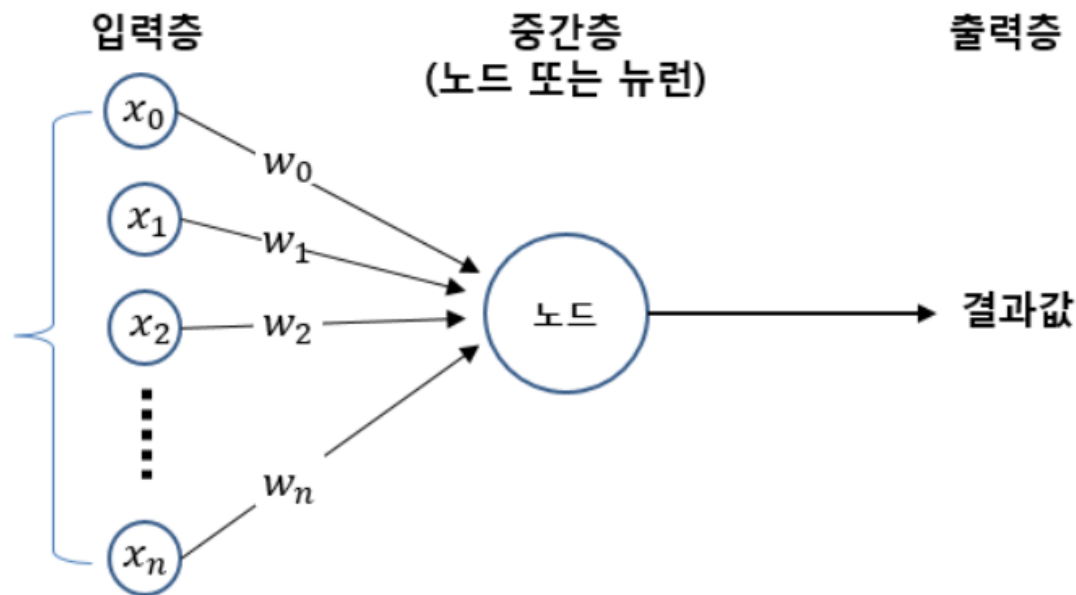


인공신경망이론

2. 퍼셉트론(Perceptron)

- 단층 퍼셉트론

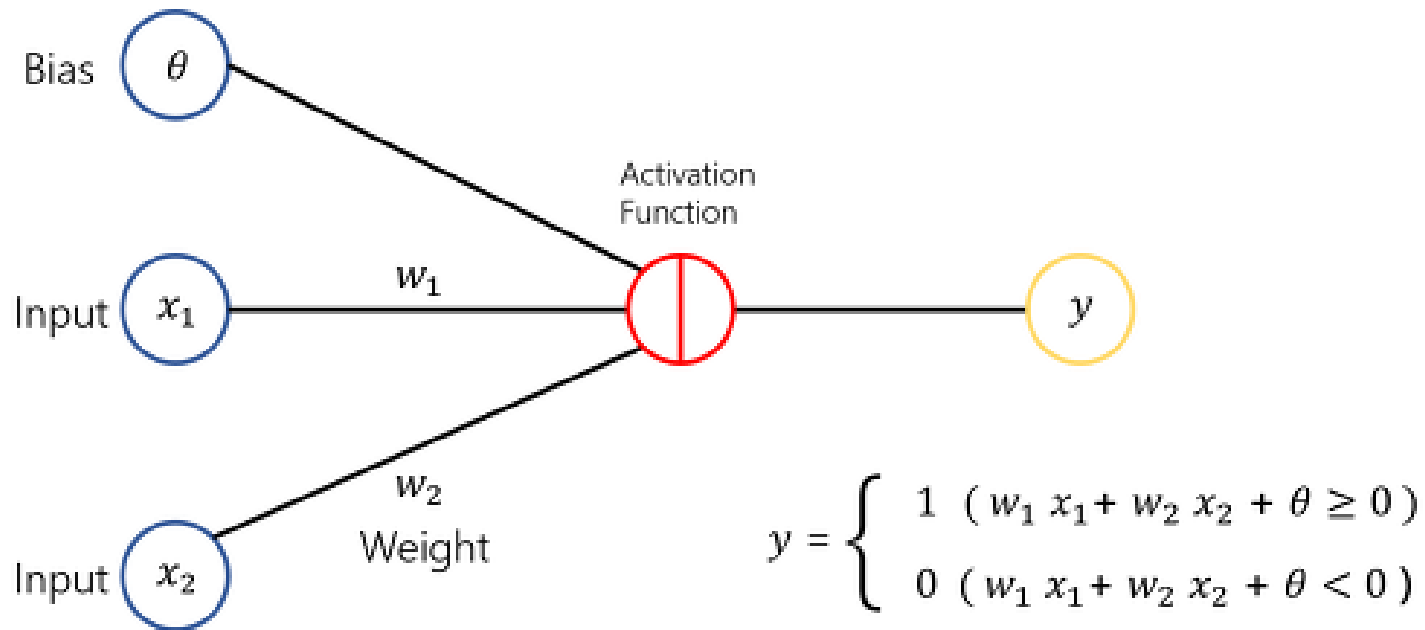
중간층이 하나의 노드로 구성되어 중간층과 출력층의 구분이 없는 구조를 단순 또는 단층 퍼셉트론이라 부른다.



인공신경망이론

2. 퍼셉트론(Perceptron)

- 단층 퍼셉트론(분류를 위한 학습모형)



인공신경망이론

2. 퍼셉트론(Perceptron)

- 손실함수(Loss Function)

- ① 평균 제곱 오차

$$E = \frac{1}{n} \sum_k (\underbrace{y_k}_{\text{예측치}} - \underbrace{t_k}_{\text{실측치}})^2$$

tk는 정답 label(실측치), yk는 예측치

- ② 교차 엔트로피 오차(Cross Entropy Error)

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

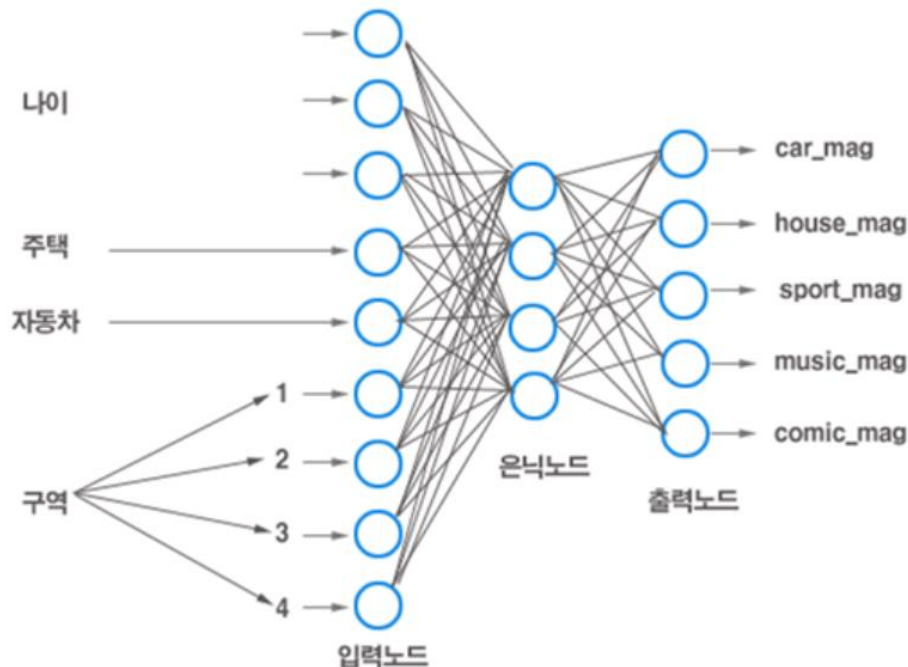
tk는 One-Hot-label로 정답만 1이고 나머지는 0인 상태, 정답이 아닐시 tk가 0이므로, log yk의 값이 무엇이던 전체 값에 영향을 미치지 못하며, 정답 레이블만이 E값에 영향을 미친다.

인공신경망이론

2. 퍼셉트론(Perceptron)

- 다층 퍼셉트론

중간층(은닉층)을 구성하는 노드가 여러 개고, 이러한 중간층이 다수로 구성되어 있는 구조를 다층 퍼셉트론이라 한다.

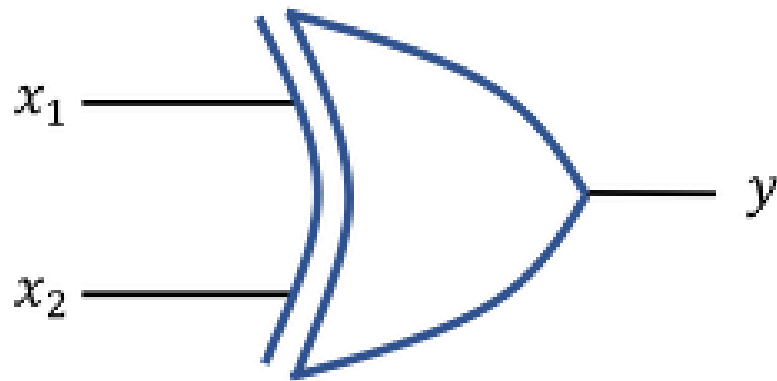


인공신경망이론

2. 퍼셉트론(Perceptron)

- 다층 퍼셉트론(MLP)

단층 퍼셉트론의 경우 XOR 문제를 학습할 수 없다.



XOR Gate 의 회로기호

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

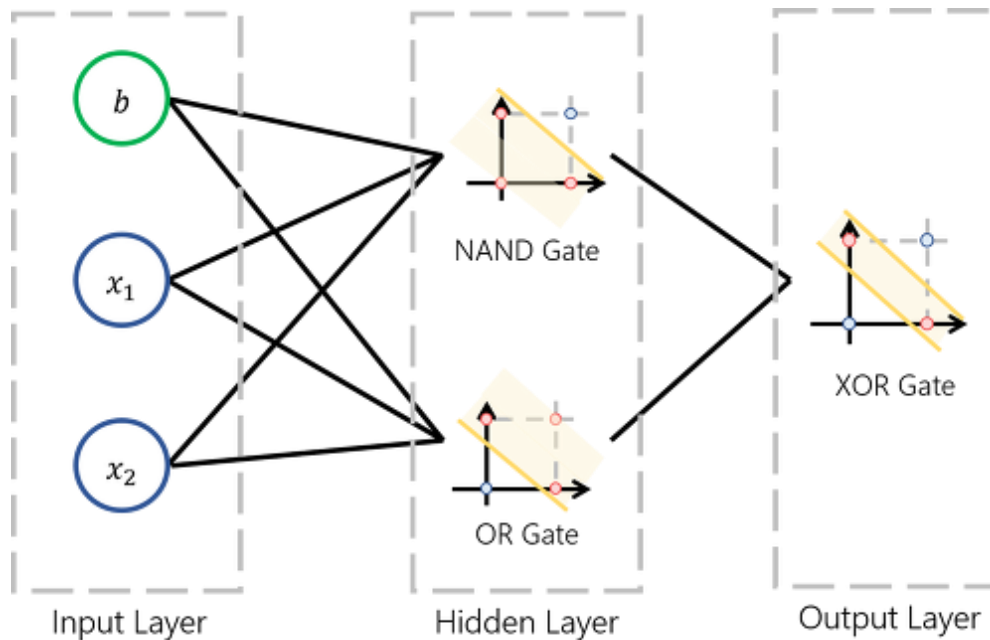
XOR Gate 의 입출력

인공신경망이론

2. 퍼셉트론(Perceptron)

- 다층 퍼셉트론

Not-AND Gate 와 OR Gate 를 한 층에, 그 두 Perceptron 의 결과값을 Input 값으로 하는 AND Gate 가 XOR Gate 가 되는 구조로 **XOR** 문제를 풀 수 있다.



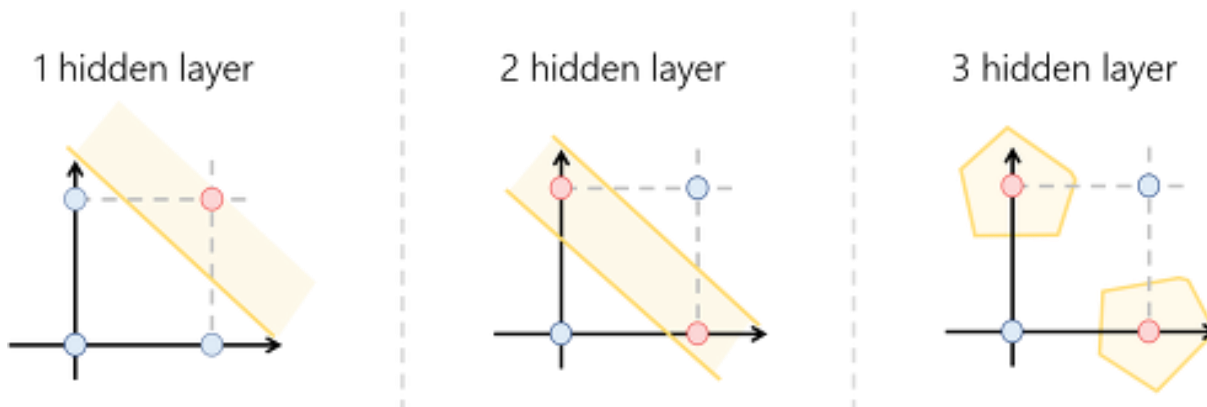
인공신경망이론

2. 퍼셉트론(Perceptron)

- 다층 퍼셉트론

단층 Perceptron 일 때는 풀 수 없었던 XOR Gate 를 2층으로만 쌓아도 풀 수 있다는 것을 알게 되었다. 그렇다면 세상의 모든 것을 다차원 좌표계로 구성한 뒤 분류 해낼 수 있다면? 이 답을 얻는 것이 우리가 실생활에서 겪는 문제를 해결하는 AI를 구현해 낼 수 있다.

Layer 의 개수에 따라 결정할 수 있는 영역

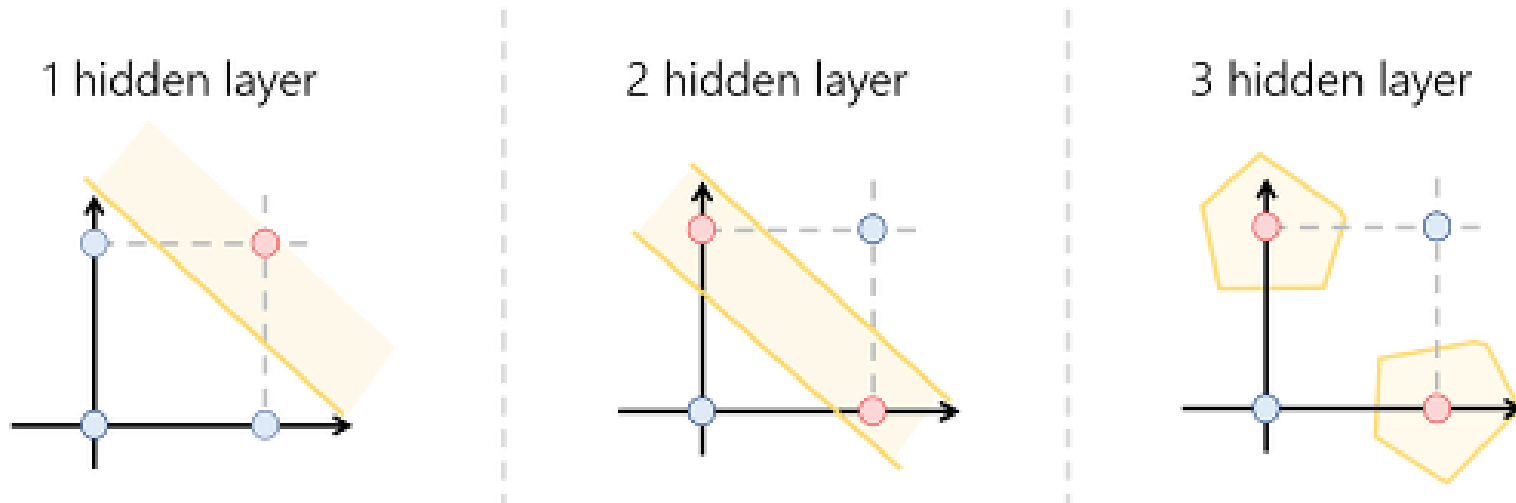


인공신경망이론

2. 퍼셉트론(Perceptron)

- 다층 퍼셉트론

- 1층일 경우에는 선형분리만 가능했다면 2층일 경우에는 구역분리가 가능하고 3층일 경우에는 더 세분화된 분리가 가능하다.
- 하지만 이렇게 깊은 층을 쌓아가면서 새로운 문제가 발생한다.

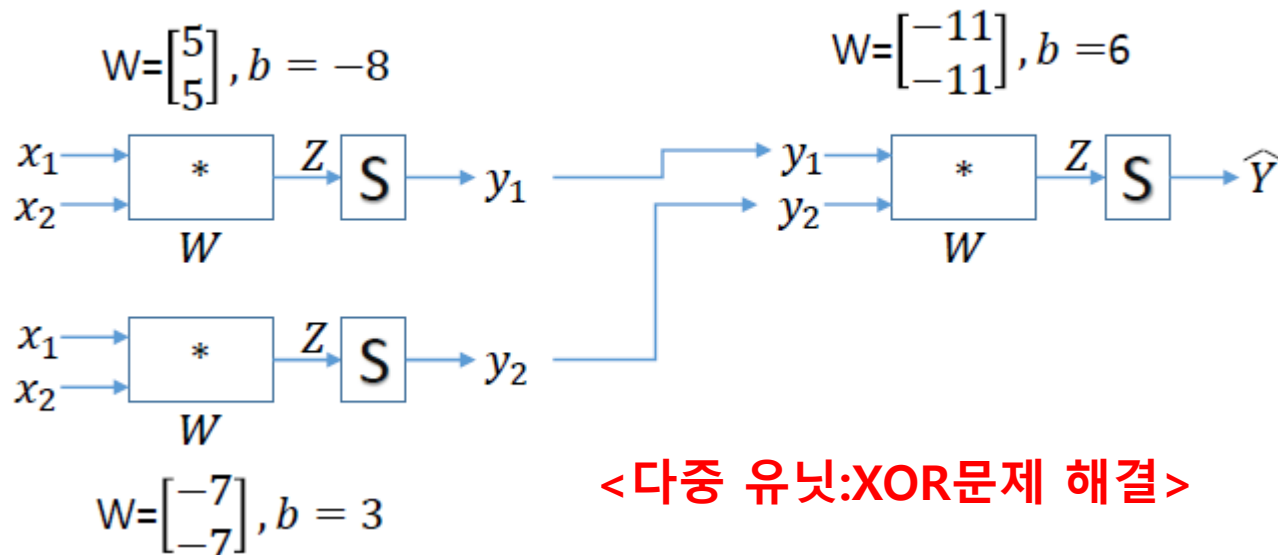
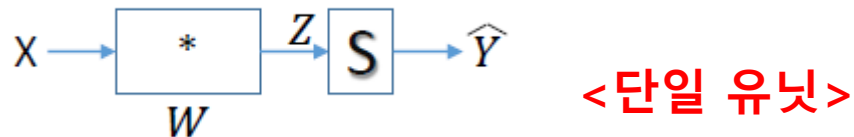


인공신경망이론

2. 퍼셉트론(Perceptron)

- 다층 퍼셉트론

(1) XOR문제를 하나의 유닛이 아닌 다중 유닛으로 해결

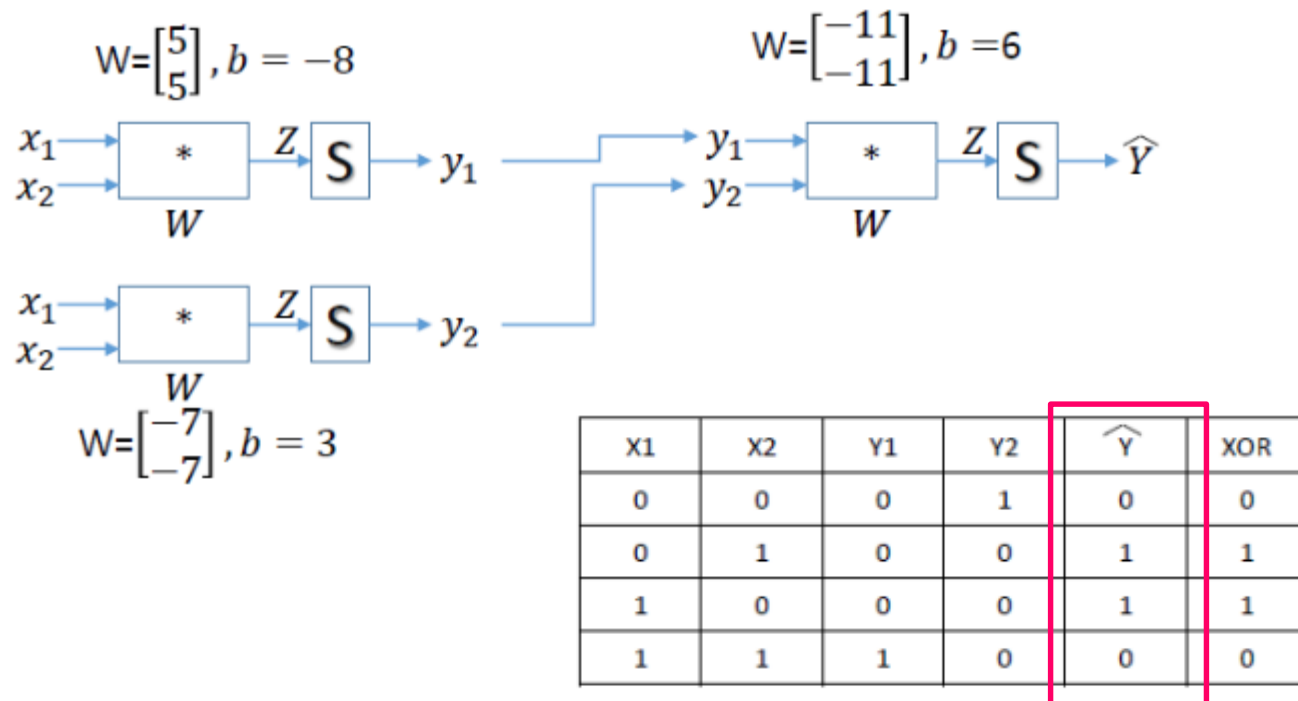


인공신경망이론

2. 퍼셉트론(Perceptron)

- 다층 퍼셉트론

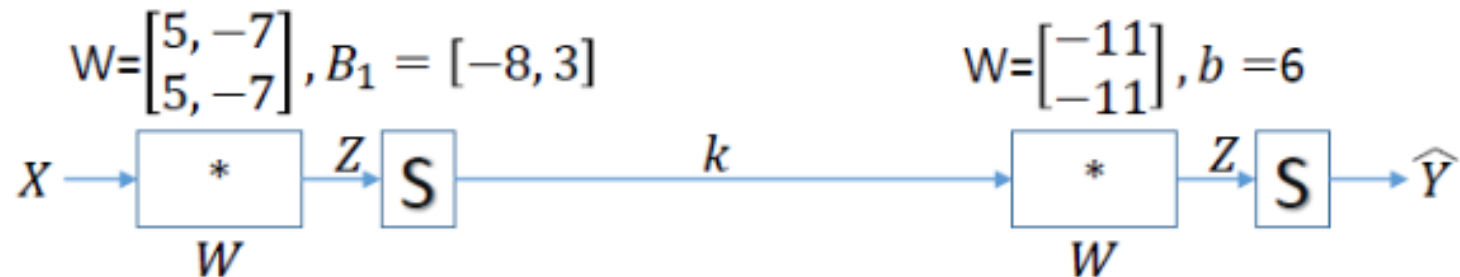
(1) XOR문제를 하나의 유닛이 아닌 다중 유닛으로 해결



인공신경망이론

2. 퍼셉트론(Perceptron)

- 다층 퍼셉트론



앞의 그래프를 **multinomial classification** 이용하여 하나로 합칠 수 있다.

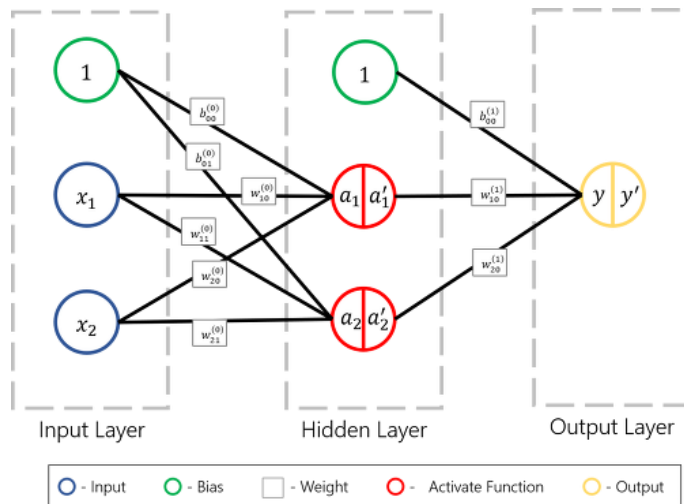
$$K(X) = \text{sigmoid}(XW_1 + B_1)$$
$$\hat{Y} = H(X) = \text{sigmoid}(K(X)W_2 + b_2)$$

인공신경망이론

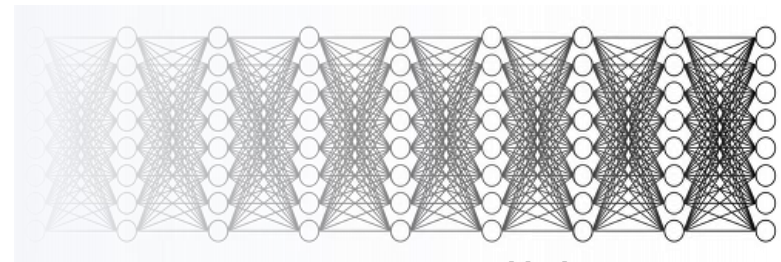
2. 퍼셉트론(Perceptron)

- 다층 퍼셉트론(MLP) & 심층퍼셉트론(DNN)

처음의 값 부분을 Input Layer, 결과가 출력되는 부분을 Output Layer, 그리고 내부의 층들을 Hidden Layer 라고 부른다. 그리고 Hidden Layer 가 3층 이상 되면 Deep Neural Network (DNN) 이라고 한다.



<MLP>

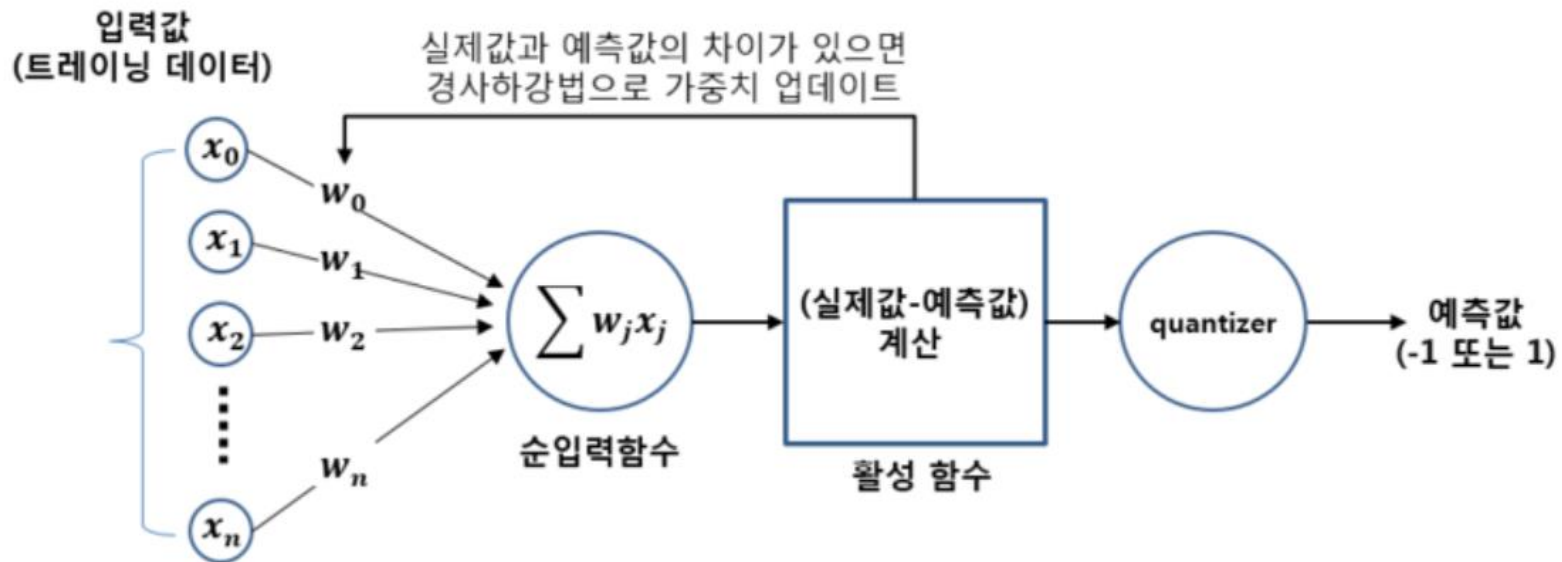


<DNN:전처리, ReLu>

인공신경망이론

3. 아달라인(Adaline)과 경사하강법(Gradient Descent)

퍼셉트론과 아달라인의 차이점은 바로 가중치 업데이트를 위한 활성화함수가 다르다는 것이다.



아달라인에서는 순입력 함수의 리턴값과 실제 결과값을 비교하여 이 오차가 최소가 되도록 가중치를 업데이트 하는 기능을 활성화 함수가 수행하게 된다.

인공신경망이론


3. 아달라인(Adaline)과 경사하강법(Gradient Descent)

- 비용함수(cost function)

이를 위해 아달라인을 발표한 논문에서는 최소제곱법을 이용한 비용함수 (cost function) $J(w)$ 를 아래와 같이 정의하였고, $J(w)$ 값이 최소가 되도록 가중치를 업데이트 하는 것이 핵심이다.

$$J(w) = \frac{1}{2} \sum_i (y^{(i)} - \underline{y^{(i)}})^2$$

비용함수



$$J(w) = \frac{1}{2} \sum_i \left(y^{(i)} - \sum_j (w_j x_j)^{(i)} \right)^2$$

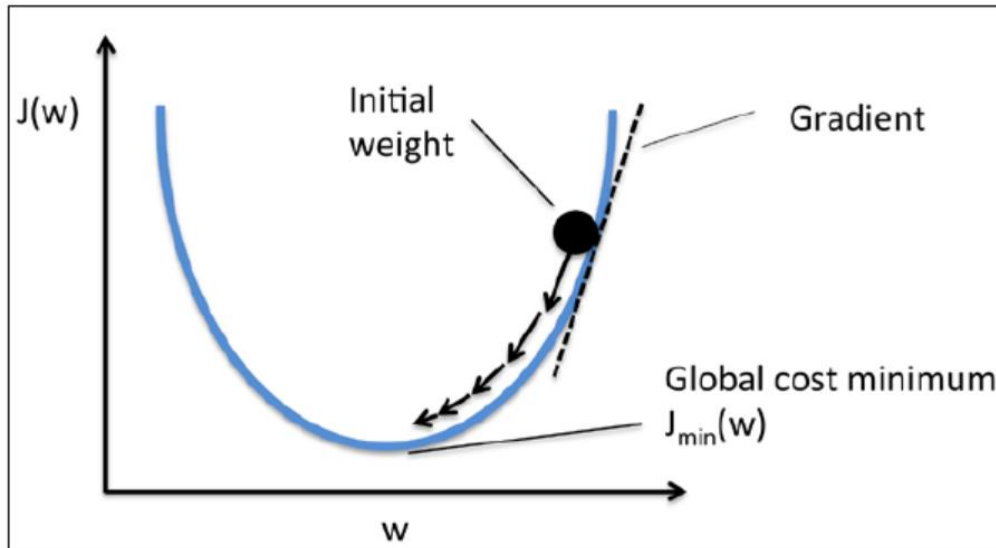
인공신경망이론

3. 아달라인(Adaline)과 경사하강법(Gradient Descent)

• 경사하강법

최적화 알고리즘을 이용하여 $J(w)$ 가 최소가 되는 가중치를 찾게된다.
경사하강법은 곡선을 따라 일정 크기(이를 스텝 크기라고 함)만큼 접선의 기울기 방향으로 내려가면서 그 값이 최소값인지 판단하는 것

→ 스텝의 크기는 접선의 기울기와 learning rate으로 결정된다.



스텝의 크기

$$\Delta w = -\eta \Delta J(w)$$

접선의 기울기
= 미분값

Learning_rate

인공신경망이론

3. 아달라인(Adaline)과 경사하강법(Gradient Descent)

- **Learning Rate & Gradient**

-학습을 나타내는 수식이다. 한 번에 얼마나 학습할지를 정해주는 Learning Rate 와 Cost 를 줄이는 방향을 나타내는 Gradient 만 구한다면 학습을 할 수 있다.

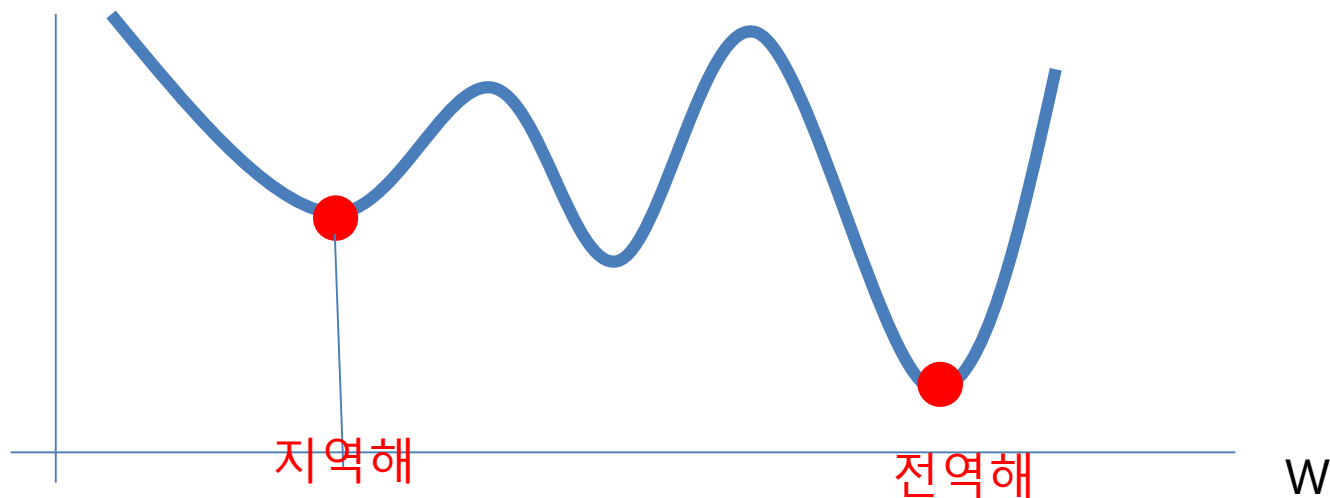
$$w^{+} = w - \underbrace{\eta}_{\substack{\text{learning rate :} \\ \text{한번에 얼마나 학습할지}}} * \underbrace{\frac{\partial E}{\partial w}}_{\substack{\text{gradient :} \\ \text{어떤 방향으로 학습할지}}}$$

-Gradient 표기의 뜻은 총 Error 에 weight 가 얼마나 영향을 끼치는가 입니다. 이를 알기 위해선 기초적인 미분 방법을 통해 처리된다.

인공신경망이론

4. 정규화

- 신경망을 잘 사용하려면 데이터에 정규화를 적용한 뒤 신경망 모델을 만들어야 한다. 정규화가 절대적인 의무사항은 아니지만 적용하면 지역해에 빠질 위험을 피한다.
- 신경망은 가중치를 임의의 값으로 초기화한 후 반복적으로 가중치를 조절하면서 SSE 또는 엔트로피 기준으로 최적화한다. 이처럼 반복적으로 답을 찾아가는 이유는 수식으로 단번에 최적의 가중치를 찾는 것이 어렵기 때문이다.

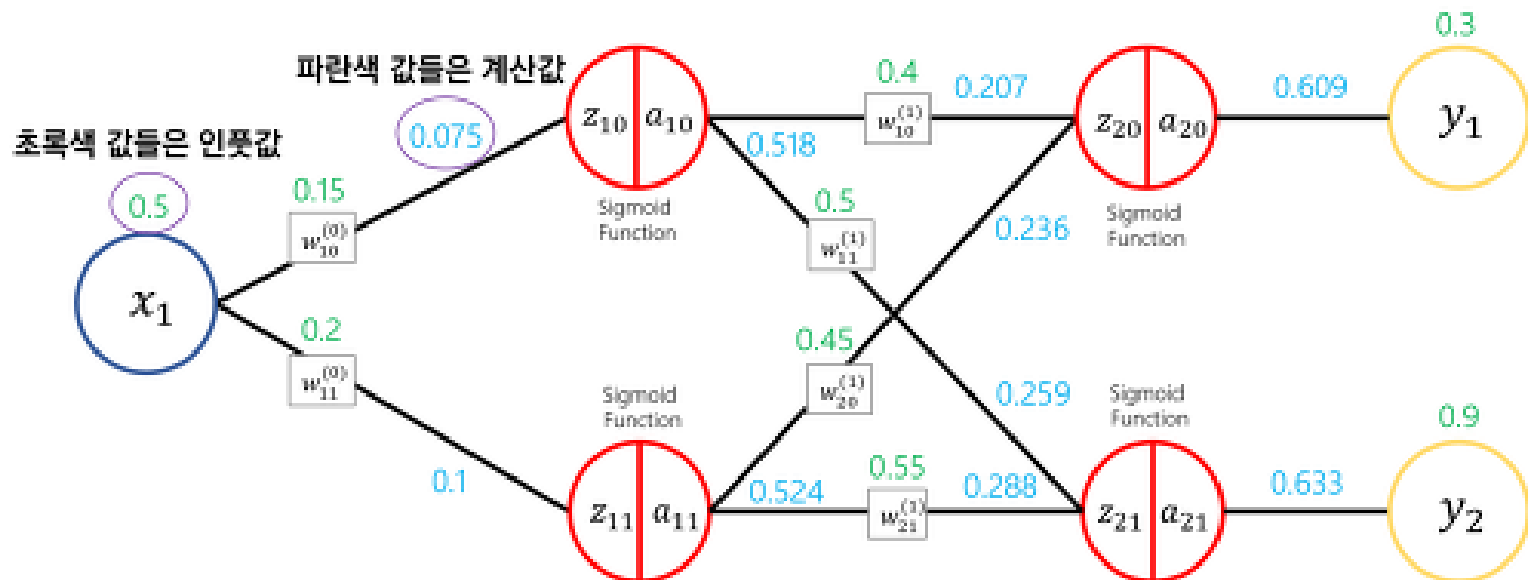


인공신경망이론

5. 역전파 알고리즘

• 역전파 알고리즘

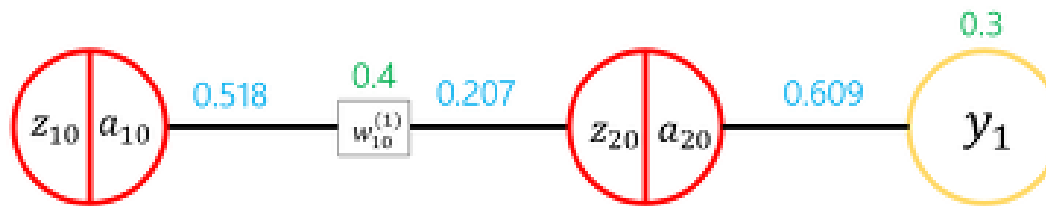
순전파(feedforward) : 다층 퍼셉트론에서는 입력층에서 전달되는 값이 은닉층의 모든 노드로 전달되며 은닉층의 모든 노드의 출력값 역시 출력층의 모든 노드로 전달된다. → 루프를 돌지 않으면서 각 층의 출력 값이 그 다음 층의 입력 값이 된다는 의미이다.



인공신경망이론

5. 역전파알고리즘

- 역전파 알고리즘 <http://gomguard.tistory.com/182?category=712467>



먼저 $w_{10}^{(1)}$ 을 학습시키자

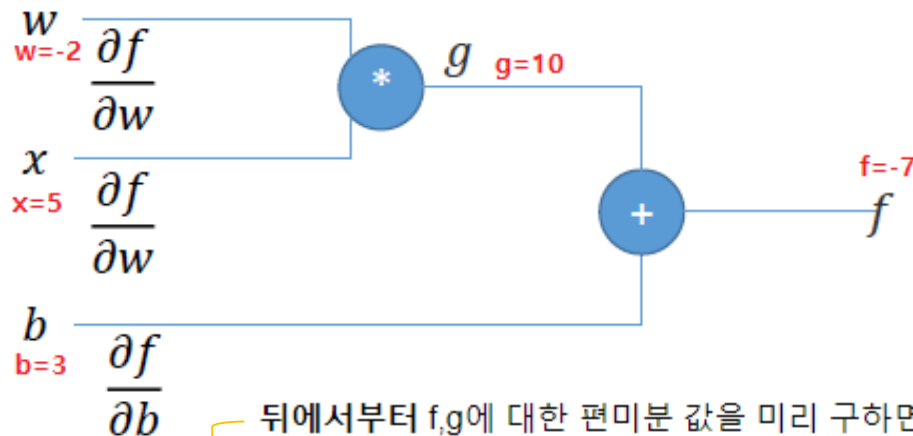
$$\frac{\partial E_{tot}}{\partial w_{10}^{(1)}} = \frac{\partial E_{tot}}{\partial a_{20}} \frac{\partial a_{20}}{\partial z_{20}} \frac{\partial z_{20}}{\partial w_{10}^{(1)}}$$

인공신경망이론

5. 역전파알고리즘

• 역전파 알고리즘

이때 w 가 f 에 미치는 영향, x 가 f 에 미치는 영향, b 가 f 에 미치는 영향을 구하고싶은데 그것이 아래와 같은 미분값이다.



뒤에서부터 f, g 에 대한 편미분 값을 미리 구하면 아래와 같다

$$\frac{\partial f}{\partial g} = 1, \frac{\partial f}{\partial b} = 1$$

$$\frac{\partial g}{\partial w} = x, \frac{\partial g}{\partial x} = w$$

인공신경망이론

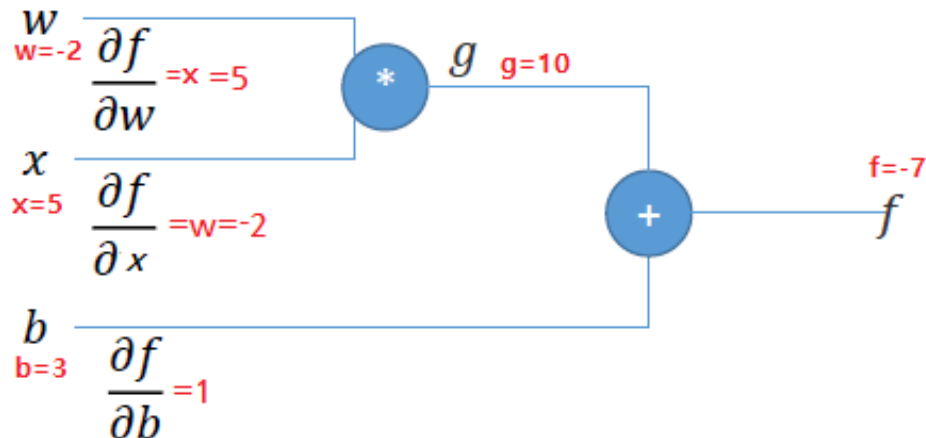
5. 역전파알고리즘

- 역전파 알고리즘 f 를 w 에 대하여 미분할때는 아래와같은 체이닝함수를 이용하여 구한다

$$\frac{\partial f}{\partial w} = \frac{\partial f}{\partial g} * \frac{\partial g}{\partial w}$$

$$\frac{\partial f}{\partial w} = \frac{\partial f}{\partial g} * \frac{\partial g}{\partial w} = \text{X}$$

| * X



인공신경망이론

5. 역전파알고리즘

- Tensorboard로 확인

1. 아래처럼 그래프로 확인할 값을 선택을 한다

```
w2_hist = tf.summary.histogram("weights2", W2)
cost_summ = tf.summary.scalar("cost", cost)
```

2. 통합을 한다

```
summary = tf.summary.merge_all()
```

3. 로그로 기록을 한다

```
writer = tf.summary.FileWriter('./logs')
writer.add_graph(sess.graph)
```

인공신경망이론

5. 역전파알고리즘

- Tensorboard로 확인

4. 실행을 한다

```
s, _ = sess.run([summary, optimizer], feed_dict=feed_dict)
writer.add_summary(s, global_step=global_step)
```

5. cmd 창을 켜서 아래 명령어로 확인을 한다

```
tensorboard --logdir=./logs
```

역전파알고리즘

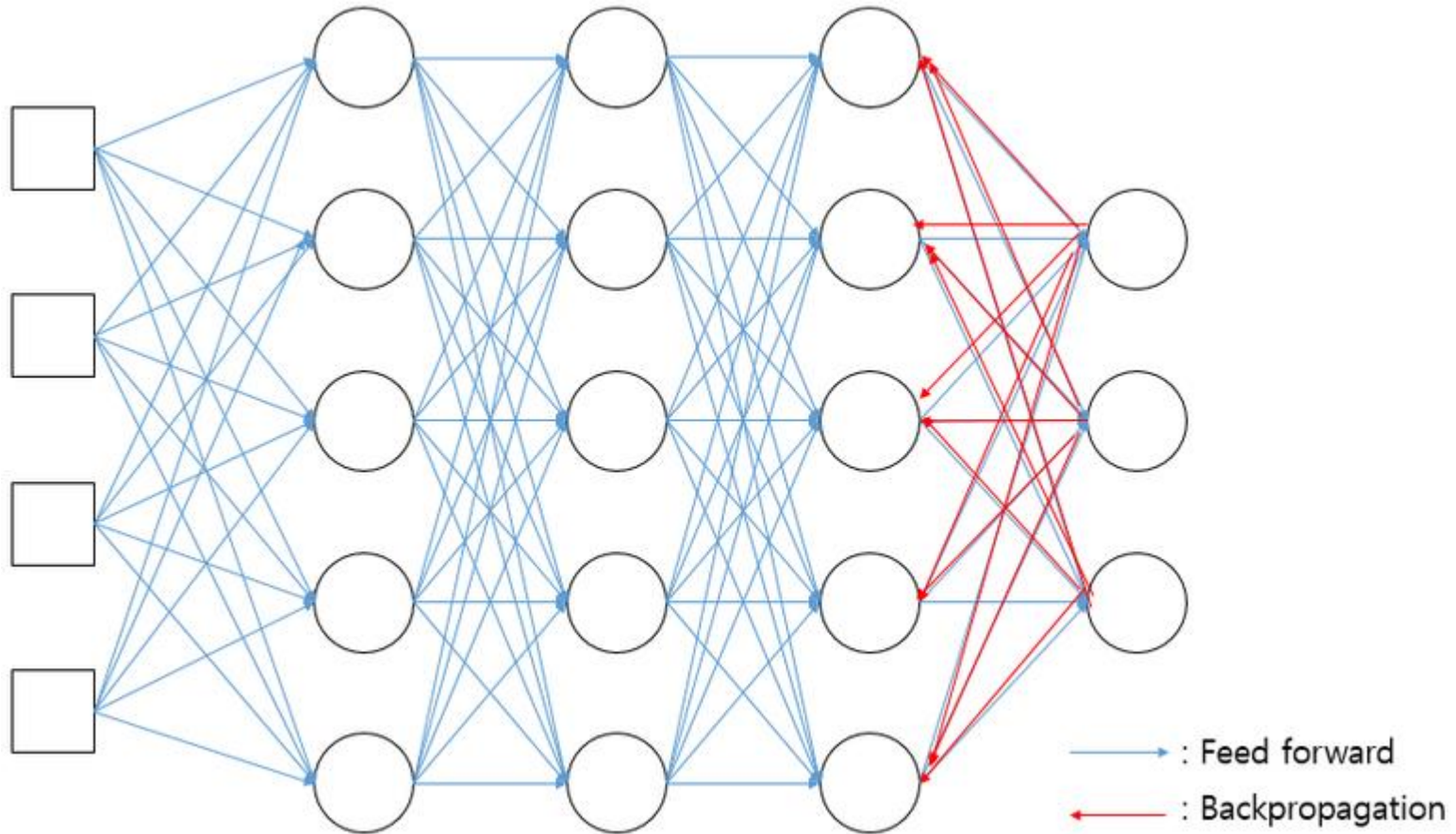
역전파 알고리즘(Backpropagation)

(1) 다층 신경망 (Multi Layer Neural Networks) 와 역전파 알고리즘 (Backpropagation)

- 지도 학습에서 신경망의 학습데이터로 { 입력, 정답 } 을 넣어준다. 단층 신경망은 출력값이 나오면 그 출력값과 정답을 빼서 오차를 구할 수 있는데 반해 다층 신경망의 경우 중간층들에서 정답이라는 개념 자체가 없기 때문에 중간층에 나온 결과값을 가지고 오차를 구할 수 없다.
- 만약 중간층들에 대한 정답이 주어진다면 오차를 구할 수 있게 되어서 신경망을 원활히 학습시킬 수 있을 것이다. 이것을 하기 위해 나온 알고리즘이 역전파 알고리즘(Backpropagation)이다.

역전파 알고리즘(Backpropagation)

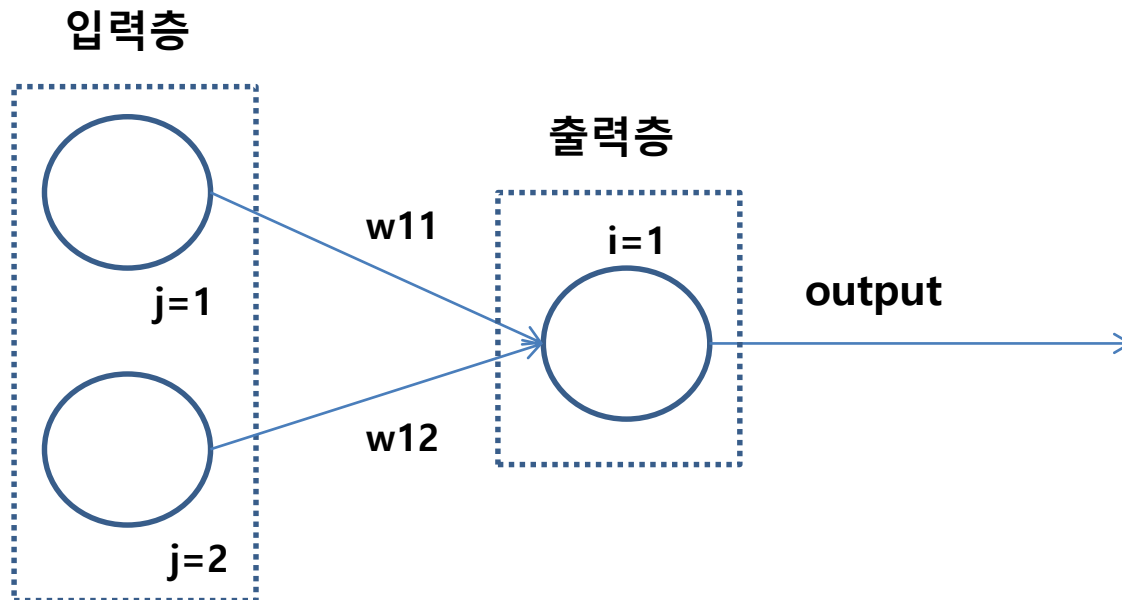
- (2) Backpropagation 방향



역전파 알고리즘(Backpropagation)

(3) 델타 규칙

- ① 역전파 알고리즘은 델타를 역전시키는 알고리즘이기 때문이다.
- ② 단층 신경망에서 많이 쓰이는 학습 규칙인 델타 규칙
- ③ 델타 규칙은 어떤 입력 노드가 출력 노드의 오차에 기여했다면, 두 노드의 연결 가중치는 해당 입력 노드의 출력과 출력 노드의 오차에 비례해 조절한다.



역전파 알고리즘(Backpropagation)

(3) 델타 규칙

델타 규칙의 식은 다음과 같다.

$$W_{ij} \leftarrow W_{ij} + \alpha e_i x_j$$

x_j ≡ 입력 노드 j의 출력

e_i ≡ 출력 노드 i의 오차

w_{ij} ≡ 출력 노드 i와 입력 노드 j의 연결 가중치

α ≡ 학습율(0 ~ 1)

학습율을 통해 가중치를 얼마나 줄 것인지 컨트롤 해줄 수 있다.

역전파 알고리즘(Backpropagation)

(4) 델타 규칙을 이용한 가중치 갱신 과정

- ① 가중치를 초기화 한다.(초기화 방법은 따로 정해져 있지는 않음)
- ② 입력 값을 넣고 출력 값을 뽑아 낸 후 (실제 정답 - 출력 값)을 통해 오차를 계산한다.
- ③ 2번에 나온 오차 값을 $\alpha e_i x_j$ 에 대입한다.
- ④ $W_{ij} \leftarrow W_{ij} + \alpha e_i x_j$ 새로 가중치를 갱신한다
- ⑤ 2~4를 반복한다.
- ⑥ 2~5를 반복한다. 오차가 줄어들 때까지...

2~5까지 전체 학습 데이터를 한번씩 모두 학습시킨 횟수를 epoch라고 부른다.

앞으로 많이 보게 될 용어이다.

역전파 알고리즘(Backpropagation)

(5) 일반화된 델타 규칙

델타 규칙의 식은 다음과 같다.

$$W_{ij} \leftarrow W_{ij} + \alpha e_i x_j$$

일반화된 델타 규칙은 다음과 같다.

$$W_{ij} \leftarrow W_{ij} + \alpha \delta_i x_j$$

• 도함수란 ? 함수 $y=f(x)$ 을 미분하여 얻은 함수 $f'(x)$ 를 말한다

바뀐 부분이라면 에러 대신에 델타로 바뀌었다는 것이다. 델타는 다음과 같다.

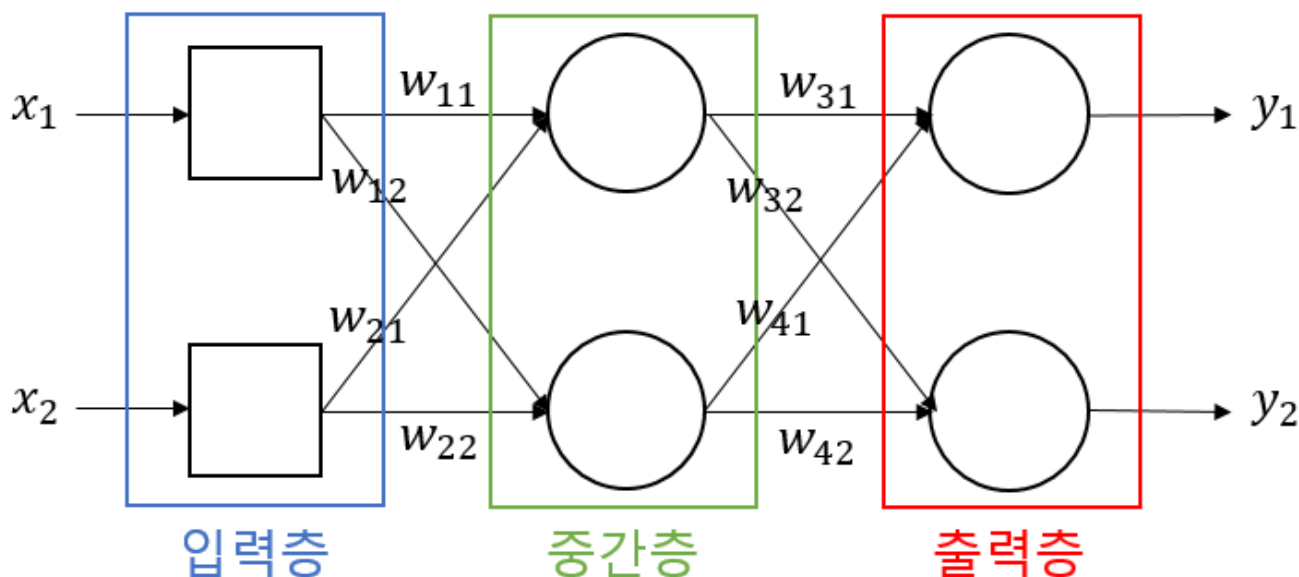
$$\delta_i = \varphi'(v_i) e_i$$

- e_i 는 출력 노드 i 의 오차
- v_i 는 출력 노드 i 의 가중 합
- φ 는 출력 노드 i 의 활성화함수의 도함수

약간 달라졌지만 출력노드의 오차와 입력 노드 출력값에 비례해서 가중치가 변한다는 것은 변함이 없다.

역전파 알고리즘(Backpropagation)

(6) 간단한 신경망 표현



입력값 2개가 들어가고 중간층을 거쳐 출력층에서 결과값이 나오게 된다.
이 예시를 가지고 델타 규칙을 보자면, 아래와 같이 구할 수 있게 된다.



Tensorflow를 활용한 실전 예제

강사 : 구정은

1. Tensorflow를 활용한 실전 예제

키와 몸무게 데이터를 입력하면 “저체중”, “정상”, “비만”이라고 판단하는 프로그램을 머신러닝으로 구해보자

```
import pandas as pd
import numpy as np
import tensorflow as tf
# 키, 몸무게, 레이블이 적힌 CSV 파일 읽어 들이기 --- (※1)
csv = pd.read_csv("bmi.csv")
# 데이터 정규화 --- (※2)
csv["height"] = csv["height"] / 200
csv["weight"] = csv["weight"] / 100

# 레이블을 배열로 변환하기 --- (※3)
# - thin=(1,0,0) / normal=(0,1,0) / fat=(0,0,1)
bclass = {"thin": [1,0,0], "normal": [0,1,0], "fat": [0,0,1]}
csv["label_pat"] = csv["label"].apply(lambda x : np.array(bclass[x]))
```

1. Tensorflow를 활용한 실전 예제

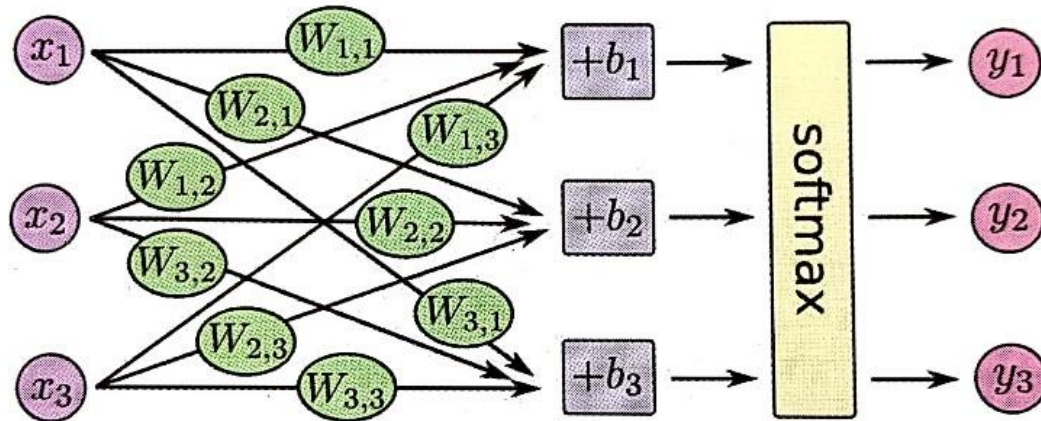
```
# 테스트를 위한 데이터 분류 --- (※4)
test_csv = csv[15000:20000]
test_pat = test_csv[["weight","height"]]
test_ans = list(test_csv["label_pat"])

# 데이터 플로우 그래프 추출하기 --- (※5)
# 플레이스홀더 선언하기
x = tf.placeholder(tf.float32, [None, 2]) # 키와 몸무게 데이터 넣기
y_ = tf.placeholder(tf.float32, [None, 3]) # 정답 레이블 넣기

# 변수 선언하기 --- (※6)
W = tf.Variable(tf.zeros([2, 3])); # 가중치
b = tf.Variable(tf.zeros([3])); # 바이어스
# 소프트맥스 회귀 정의하기 --- (※7)
y = tf.nn.softmax(tf.matmul(x, W) + b)
```


1. Tensorflow를 활용한 실전 예제

- 소프트맥스 회귀



$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

<소프트맥스 회귀 구조>

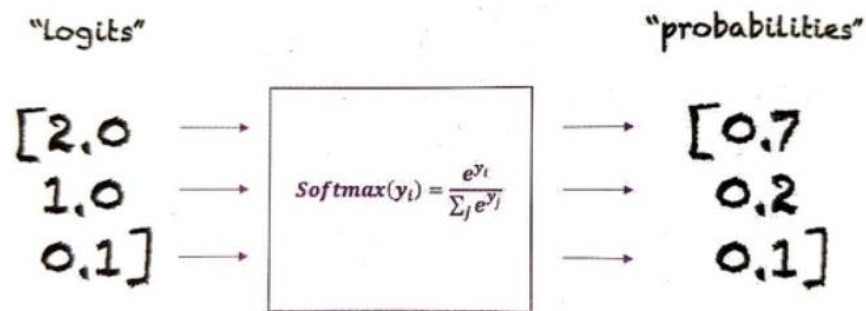
1. Tensorflow를 활용한 실전 예제

- 소프트맥스 회귀

- Softmax 함수는 Normalization 함수로써 출력값들의 합을 1로 만들어준다. 다음은 Softmax함수의 수식이다.

$$\text{Softmax}(y_i) = \frac{\exp(y_i)}{\sum_j \exp(y_j)}$$

<소프트 맥스 회귀>



<logit 함수결과에 softmax함수를 적용한 결과>

1. Tensorflow를 활용한 실전 예제

```
# 모델 훈련하기 --- (※8)
```

```
cross_entropy = -tf.reduce_sum(y_ * tf.log(y))
```

```
optimizer = tf.train.GradientDescentOptimizer(0.01)
```

```
train = optimizer.minimize(cross_entropy)
```

```
# 정답률 구하기
```

```
predict = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
```

```
accuracy = tf.reduce_mean(tf.cast(predict, tf.float32))
```

텐서를 새로운 형태로 캐스팅하는데 사용

```
# 세션 시작하기
```

```
sess = tf.Session()
```

```
sess.run(tf.global_variables_initializer()) # 변수 초기화하기
```

1. Tensorflow를 활용한 실전 예제

- cost function : 교차엔트로피

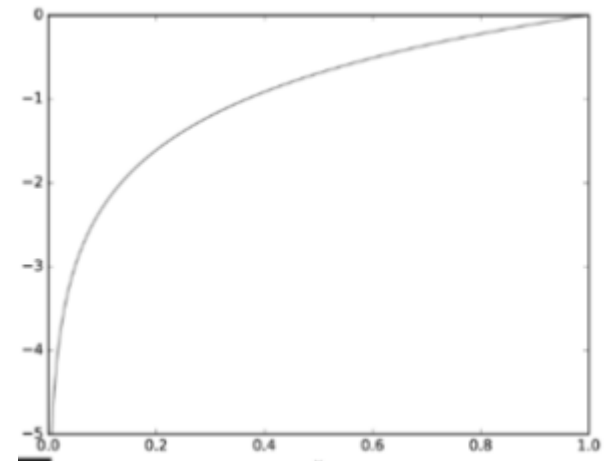
$$E = - \sum_k t_k \log_e y_k$$

tk는 One-Hot-label로 정답만 1이고 나머지는 0인 상태, 정답이 아닐시 tk가 0이므로, log yk의 값이 무엇이던 전체 값에 영향을 미치지 못하며, 정답 레이블만이 E값에 영향을 미친다.

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

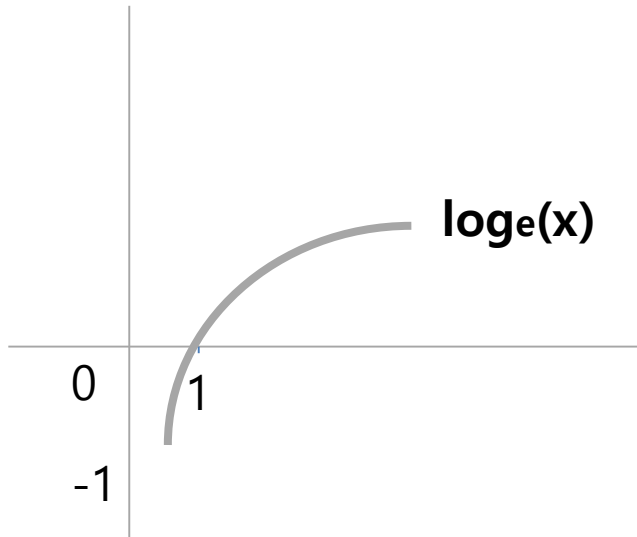
실측치

예측치=신경망 가설



1. Tensorflow를 활용한 실전 예제

- cost function : 교차엔트로피



예를 들어 실제 데이터에서 2가 정답이고 신경망을 통해 학습된 가설의 예측치가 소프트 맥스를 거쳐 0.6으로 결과가 나왔다면(2가 정답이 될 확률값으로 0.6을 출력), 이 때 교차 엔트로피 오차를 통한 cost 는 $-\log(0.6) = \log 6 = 0.51$ 이 된다.(밑수는 자연로그 e)

만약 같은 조건에서 신경망 출력이 0.6의 확률이 아니라 0.1의 확률로 예측했다면? $-\log(0.1) = \log 0.1 = 2.3$

신경망 추정치와 실제 데이터가 추론한 값이 차이가 날수록 코스트 값이 높아진다.

➔ 결국 참(1)을 예측한 신경망일 수록 오차가 0에 가까워 모델링된다.

1. Tensorflow를 활용한 실전 예제

- **cost function : 교차엔트로피**

예시) 정답 **label**이 [0,1] 인데

- 학습이 잘 안되어 [1,0]으로 학습한 경우의 cost 함수의 값은 무한대

$$-\sum_i y'_i \log(y_i) = -[0, 1] \begin{bmatrix} \log(1) \\ \log(0) \end{bmatrix} = -(0 + (-\infty)) = \infty$$

- 학습이 잘된 경우 = 오차가 없는 경우 0으로 수렴

$$-\sum_i y'_i \log(y_i) = -[0, 1] \begin{bmatrix} \log(0) \\ \log(1) \end{bmatrix} = -(0 + 0) = 0$$

1. Tensorflow를 활용한 실전 예제

```
# 학습시키기
for step in range(3500):
    i = (step * 100) % 14000
    rows = csv[1 + i : 1 + i + 100]
    x_pat = rows[["weight", "height"]]
    y_ans = list(rows["label_pat"])
    fd = {x: x_pat, y_: y_ans}
    sess.run(train, feed_dict=fd)
    if step % 500 == 0:
        cre = sess.run(cross_entropy, feed_dict=fd)
        acc = sess.run(accuracy, feed_dict={x: test_pat, y_: test_ans})
        print("step=", step, "cre=", cre, "acc=", acc)
# 최종적인 정답률 구하기
acc = sess.run(accuracy, feed_dict={x: test_pat, y_: test_ans})
print("정답률 =", acc)
```



Tensorflow를 활용한 이미지 인식

강사 : 구정은

3. Softmax를 이용한 이미지 인식

3-1. MNIST

그러면 이제 실제로 텐서플로우로 모델을 만들어서 학습을 시켜보자. 예제에 사용할 시나리오는 MNIST (Mixed National Institute of Standards and Technology database) 라는 데이터로, 손으로 쓴 숫자이다. 이 손으로 쓴 숫자 이미지를 0~9 사이의 숫자로 인식하는 예제이다.



- 이 예제는 텐서플로우 MNIST 튜토리얼 (<https://www.tensorflow.org/tutorials/mnist/beginners/>) 을 기반으로 작성하였는데, MNIST 숫자 이미지를 인식하는 모델을 softmax 알고리즘을 이용하여 만든 후에, 트레이닝을 시키고, 정확도를 체크해보자.

3. Softmax를 이용한 이미지 인식

3-2. 데이터셋

MNIST 데이터는 텐서플로우 내에 라이브러리 형태로 내장이 되어 있어서 쉽게 사용이 가능하다. `tensorflow.examples.tutorials.mnist` 패키지에 데이터가 들어 있는데, `read_data_sets` 명령어를 이용하면 쉽게 데이터를 로딩할 수 있다.

- 데이터 로딩 코드

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets('/tmp/tensorflow/mnist/input_data',
                                  one_hot=True)
```

- Mnist 데이터셋에는 총 60,000개의 데이터가 있는데, 이 데이터는 크게 아래와 같이 세 종류의 데이터 셋으로 나뉜다. 모델 학습을 위한 학습용 데이터인 `mnist.train` 그리고, 학습된 모델을 테스트하기 위한 테스트 데이터 셋은 `mnist.test`, 그리고 모델을 확인하기 위한 `mnist.validation` 데이터셋으로 구별된다.

3. Softmax를 이용한 이미지 인식

- 각 데이터는 아래와 같이 학습용 데이터 55000개, 테스트용 10,000개, 그리고, 확인용 데이터 5000개로 구성되어 있다.

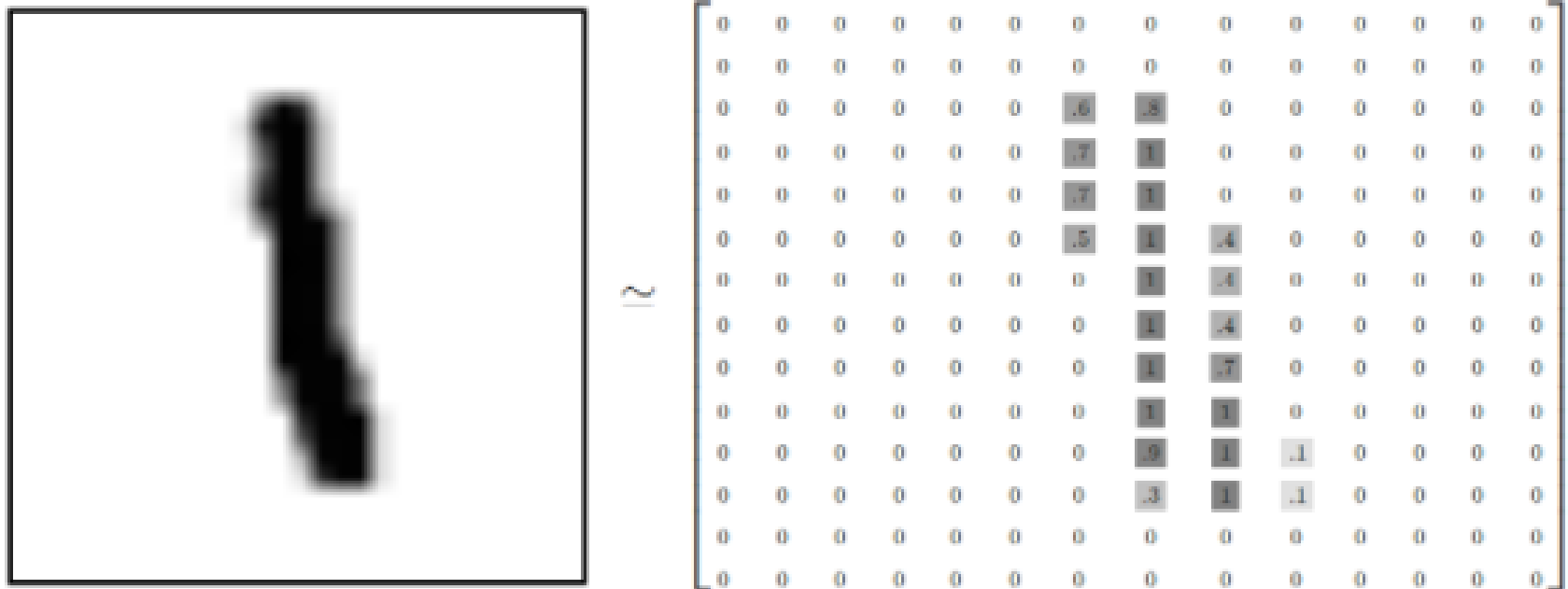
데이터셋 명	행렬 차원	데이터 종류	노트
mnist.train.images	55000 x 784	학습 이미지 데이터	
mnist.train.labels	55000 x 10	학습 라벨 데이터	
mnist.test.images	10000 x 784	테스트용 이미지 데이터	
mnist.test.labels	10000 x 10	테스트용 라벨 데이터	
mnist.validation.images	5000 x 784	확인용 이미지 데이터	
mnist.validation.labels	5000 x 10	확인용 라벨 데이터	

- 각 데이터셋은 학습을 위한 글자 이미지를 저장한 데이터 `image` 와, 그 이미지가 어떤 숫자인지를 나타낸 라벨 데이터인 `label`로 두개의 데이터 셋으로 구성되어 있다.

3. Softmax를 이용한 이미지 인식

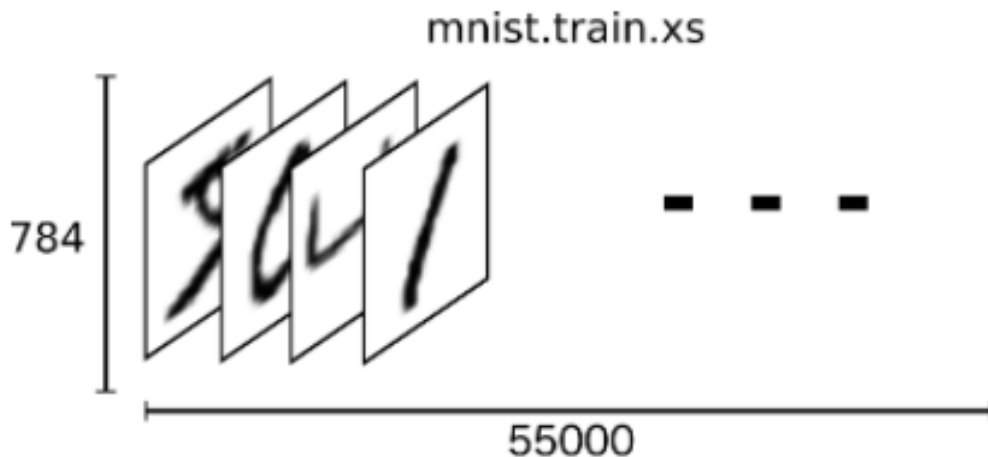
3-3. 이미지

먼저 이미지 데이터를 보면 아래 그림과 같이 28x28 로 구성되어 있는데,



3. Softmax를 이용한 이미지 인식

- 이를 2차원 행렬에서 1차원으로 쭉욱 핀 형태로 784개의 열을 가진 1차원 행렬로 변환되어 저장되어 있다.
- `mnist.train.image`는 이러한 784개의 열로 구성된 이미지가 55000개가 저장되어 있다.



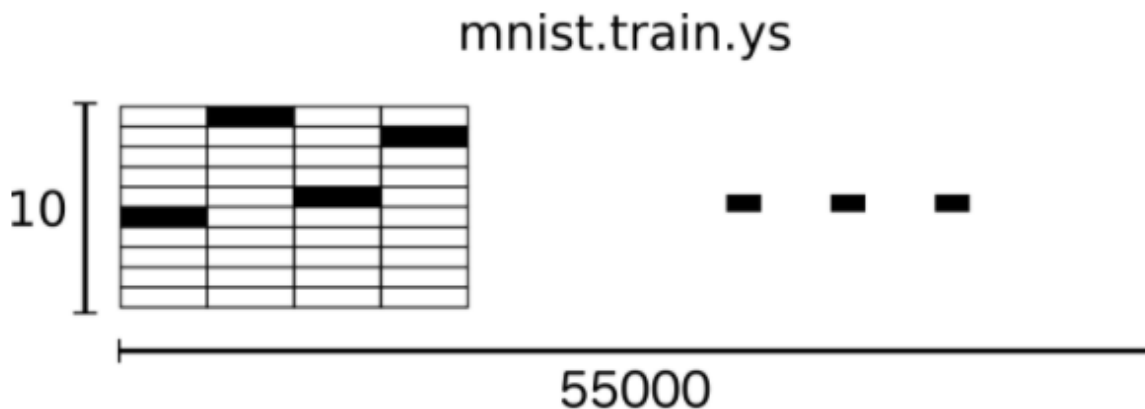
- 텐서플로우의 행렬을 나타내는 `shape`의 형태로는 `shape=[55000,784]` 이 된다. 마찬가지로, `mnist.train.image` 도 784개의 열로 구성된 숫자 이미지 데 이 타 를 10000 개 를 가 지 고 있 고 텐서플로우의 `shape` 으로는 `shape=[10000,784]` 로 표현될 수 있다.]

3. Softmax를 이용한 이미지 인식

3-4. 라벨

Label 은 이미지가 나타내는 숫자가 어떤 숫자인지를 나타내는 라벨 데이터로 10개의 숫자로 이루어진 1행 행렬이다. 0~9 순서로, 그 숫자이면 1 아니면 0으로 표현된다. 예를 들어 1인 경우는 $[0,1,0,0,0,0,0,0,0,0]$ 9인 경우는 $[0,0,0,0,0,0,0,0,0,1]$ 로 표현된다.

이미지 데이터에 대한 라벨이기 때문에, 당연히 이미지 데이터 수만큼의 라벨을 가지게 된다.

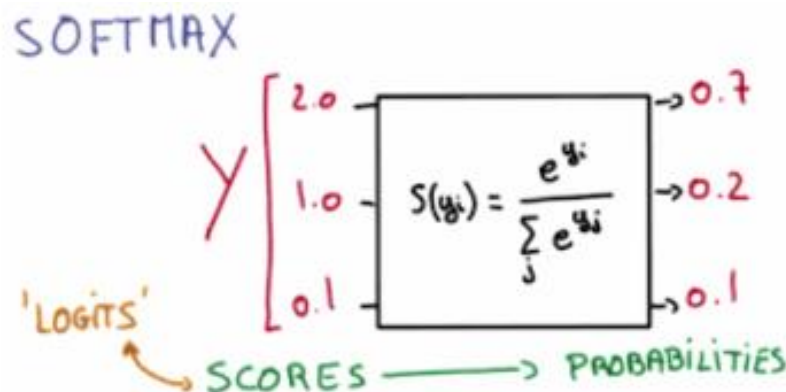


Train 데이터 셋은 이미지가 55000개 있기 때문에, Train의 label의 수 역시도 55000개가 된다.

3. Softmax를 이용한 이미지 인식

3-5. 소프트맥스 회귀(Softmax regression)

- 숫자 이미지를 인식하는 모델은 많지만, 여기서는 간단한 알고리즘 중 하나인 소프트맥스 회귀 모델을 사용하겠다.
- 소프트맥스 회귀에 대한 알고리즘 자체는 자세히 설명하지 않는다. 소프트맥스 회귀는 classification 알고리즘중의 하나로, 들어온 값이 어떤 분류인지 구분해주는 알고리즘이다.
- 예를 들어 A,B,C 3개의 결과로 분류해주는 소프트맥스의 경우 결과값은 [0.7,0.2,0.1] 와 같이 각각 A,B,C일 확률을 리턴해준다. (결과값의 합은 1.0 이 된다.)

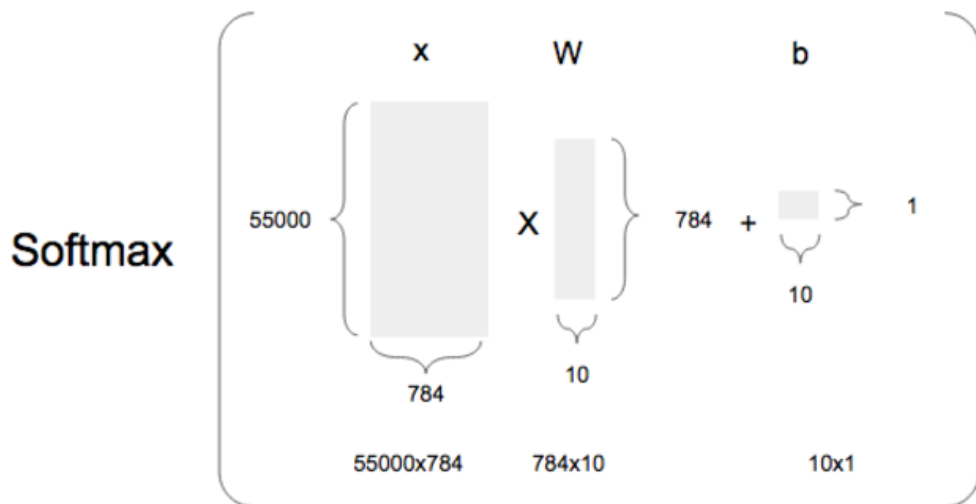


- (cf. 로지스틱 회귀는 두 가지로만 분류가 가능하지만, 소프트맥스 회귀는 n 개의 분류로 구분이 가능하다.)

3. Softmax를 이용한 이미지 인식

3-6. 모델 정의

- 소프트맥스로 분류를 할때, x 라는 값이 들어 왔을때, 분류를 한다고 가정했을때, 모델에서 사용하는 가설은 다음과 같다.
- $y = \text{softmax}(W \cdot x + b)$ # W 는 weight, 그리고 b 는 bias 값이다.
- y 는 최종적으로 10개의 숫자를 감별하는 결과가 나와야 하기 때문에, 크기가 10인 행렬이 되고, 10개의 결과를 만들기 위해서 W 역시 10개가 되어야 하며, 이미지 하나는 784개의 숫자로 되어 있기 때문에, 10개의 값을 각각 784개의 숫자에 적용해야 하기 때문에, W 는 784×10 행렬이 된다. 그리고, b 는 10개의 값에 각각 더하는 값이기 때문에, 크기가 10인 행렬이 된다.

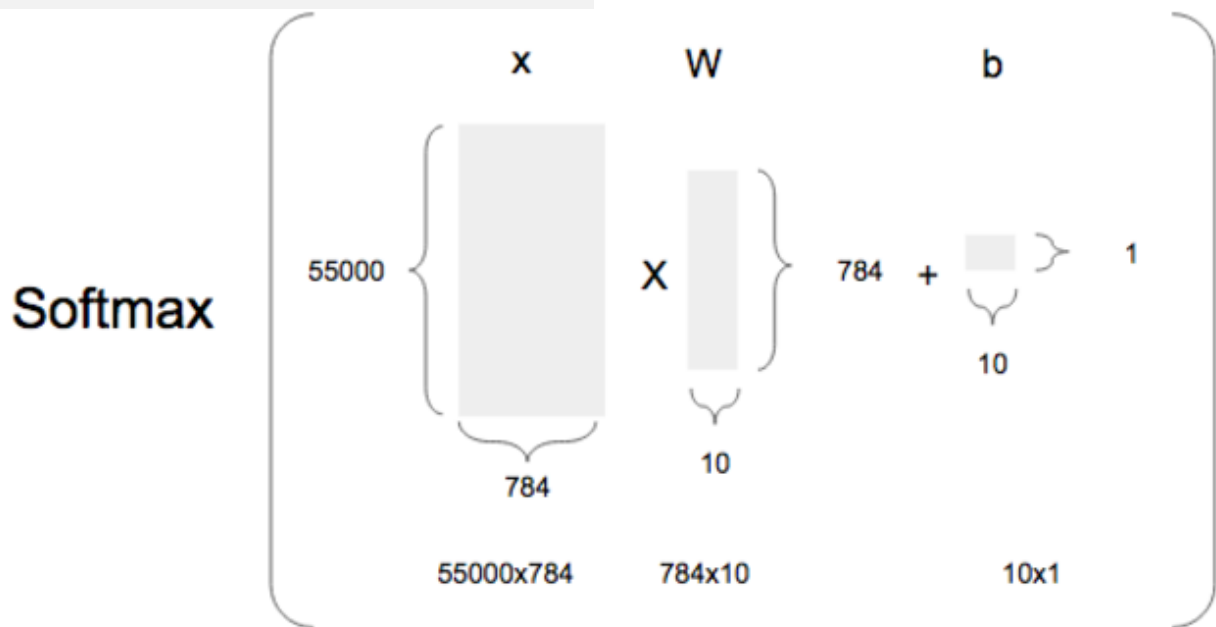


3. Softmax를 이용한 이미지 인식

3-5. 소프트맥스 회귀(Softmax regression)

- 이를 텐서플로우 코드로 표현하면 다음과 같다.

```
x = tf.placeholder(tf.float32, [None, 784])  
W = tf.Variable(tf.zeros([784, 10]))  
b = tf.Variable(tf.zeros([10]))  
k = tf.matmul(x, W) + b  
y = tf.nn.softmax(k)
```



3. Softmax를 이용한 이미지 인식

3-5. 소프트맥스 회귀(Softmax regression)

- 우리가 구하고자 하는 값은 x 값으로 학습을 시켜서 0~9를 가장 잘 구별해 내는 W 와 b 의 값을 찾는 일이다.
- 여기서 코드를 주의깊게 봤다면 하나의 의문이 생길 것이다.
- x 의 데이터는 총 55000개로, 55000×784 행렬이 되고, W 는 784×10 행렬이다. 이 둘을 곱하면, 55000×10 행렬이 되는데, b 는 1×10 행렬로 차원이 달라서 합이 되지 않는다.
- 텐서플로우와 파이썬에서는 이렇게 차원이 다른 행렬을 큰 행렬의 크기로 늘려주는 기능이 있는데, 이를 브로드 캐스팅이라고 한다.
- 브로드 캐스팅에 의해서 b 는 55000×10 사이즈로 자동으로 늘어나고 각 행에는 첫행과 같은 데이터들로 채워지게 된다.

3. Softmax를 이용한 이미지 인식

3-5. 소프트맥스 회귀(Softmax regression)

- tensorflow & 머신러닝 알고리즘

소프트맥스 알고리즘을 이해하고 사용해도 좋지만, 텐서플로우에는 이미 `tf.nn.softmax` 라는 함수로 만들어져 있고, 대부분 많이 알려진 머신러닝 모델들은 샘플들이 많이 있기 때문에, 대략적인 원리만 이해하고 가져다 쓰는 것을 권장한다. 보통 모델을 다 이해하려고 하다가 수학에서 부딪혀서 포기하는 경우가 많은데, 디테일한 모델을 이해하기 힘들면, 그냥 함수나 예제코드를 가져다 쓰는 방법으로 접근하자. 우리가 일반적인 프로그래밍에서도 해쉬테이블이나 트리와 같은 자료구조에 대해서 대략적인 개념만 이해하고 미리 정의된 라이브러리를 사용하지 직접 해쉬 테이블등을 구현하는 경우는 드물다.

3. Softmax를 이용한 이미지 인식

3-6. 코스트(비용) 함수

- 이 소프트맥스 함수에 대한 코스트 함수는 크로스엔트로피 (Cross entropy) 함수의 평균을 이용하는데, 크로스엔트로피 함수 역시 함수로 구현이 되어있다.

```
Cost = tf.reduce_mean(  
    tf.nn.softmax_cross_entropy_with_logits(tf.matmul(x, W) + b, y_))
```

- 가설에 의해 계산된 값 y 를 넣지 않고 $\text{tf.matmul}(x, W) + b$ 를 넣은 이유는 $\text{tf.nn.softmax_cross_entropy_with_logits}$ 함수 자체가 softmax를 포함하기 때문이다.
- $y_$ 은 학습을 위해서 입력된 값이다.

3. Softmax를 이용한 이미지 인식

3-7. 텐서플로우로 구현

•전체코드

```
# Import data
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

mnist = input_data.read_data_sets('/tmp/tensorflow/mnist/input_data',
one_hot=True)

# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
k = tf.matmul(x, W) + b
y = tf.nn.softmax(k)
```

3. Softmax를 이용한 이미지 인식

3-7. 텐서플로우로 구현

•전체코드

```
# Define loss and optimizer
y_ = tf.placeholder(tf.float32,[None,10])
learning_rate = 0.5
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(k, y_))
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
print ("Training")
sess = tf.Session()
init = tf.global_variables_initializer() #.run()
sess.run(init)
for _ in range(1000):
    # 1000번씩, 전체 데이터에서 100개씩 뽑아서 트레이닝을 함.
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

print ('b is ',sess.run(b))
print('W is',sess.run(W))
```

3. Softmax를 이용한 이미지 인식

3-7. 텐서플로우로 구현

- 데이터 로딩

```
# Import data
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

mnist = input_data.read_data_sets('/tmp/tensorflow/mnist/input_data',
one_hot=True)
```

- 앞에서 데이터에 대해서 설명한것과 같이 데이터를 로딩하는 부분이다. `read_data_sets`에 들어가 있는 디렉토리는 샘플데이터를 온라인에서 다운 받는데, 그 데이터를 임시로 저장해놓을 위치이다.

3. Softmax를 이용한 이미지 인식

3-7. 텐서플로우로 구현

- 모델 정의
- 다음은 소프트맥스를 이용하여 모델을 정의한다.

```
# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
k = tf.matmul(x, W) + b
y = tf.nn.softmax(k)
```

- x 는 트레이닝 데이터를 저장하는 스테이크홀더, W 는 Weight, b 는 bias 값이고, 모델은 $y = \text{tf.nn.softmax}(\text{tf.matmul}(x, W) + b)$ 이 된다.

`tf.nn.softmax_cross_entropy_with_logits` 함수는 softmax를 포함하고 있다. 그래서 softmax를 적용한 y 를 넣으면 안되고 softmax 적용전인 k 를 넣어야 한다.

3. Softmax를 이용한 이미지 인식

3-7. 텐서플로우로 구현

- 코스트함수와 옵티마이저 정의

모델을 정의했으면 학습을 위해서, 코스트 함수를 정의한다.

```
# Define loss and optimizer
```

```
y_ = tf.placeholder(tf.float32, [None, 10])
```

```
learning_rate = 0.5
```

```
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(k, y_))
```

```
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

3. Softmax를 이용한 이미지 인식

3-7. 텐서플로우로 구현

- 세션 초기화

```
print ("Training")  
sess = tf.Session()  
init = tf.global_variables_initializer() #.run()  
sess.run(init)
```

- **tf.Session()** 을 이용해서 세션을 만들고, **global_variable_initializer()**를 이용하여, 변수들을 모두 초기화한후, 초기화 값을 **sess.run**에 넘겨서 세션을 초기화 한다.

3. Softmax를 이용한 이미지 인식

(텐서플로우 문서에 따르면, 전체 데이터를 순차적으로 학습 시키기에는 연산 비용이 비싸기 때문에, 샘플링을 해도 비슷한 정확도를 낼 수 있기 때문에, 예제 차원에서 간단하게, Stochastic training을 사용한 것으로 보인다.)

3-7. 텐서플로우로 구현

- 트레이닝 시작
- 세션이 생성되었으면 이제 트레이닝을 시작한다.

```
for _ in range(1000):  
    # 1000번씩, 전체 데이터에서 100개씩 뽑아서 트레이닝을 함.  
    batch_xs, batch_ys = mnist.train.next_batch(100)  
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```

- 여기서 주목할점은 Batch training 과 Stochastic training 인데, Batch training이란, 학습을 할때 전체 데이터를 가지고 한번에 학습을 하는게 아니라 전체 데이터셋을 몇 개로 쪼갬후 나눠서 트레이닝을 하는 방법을 배치 트레이닝이라고 한다. 그중에서 여기에 사용된 배치 방법은 Stochastic training 이라는 방법인데, 원칙대로라면 전체 55000개 의 학습 데이터가 있기 때문에 배치 사이즈를 100으로 했다면, 100개씩 550번 순차적으로 데이터를 읽어서 학습을 해야겠지만, Stochastic training은 전체 데이터중 일부를 샘플링해서 학습하는 방법으로, 여기서는 배치 한번에 100개씩의 데이터를 뽑아서 1000번 배치로 학습을 하였다.

3. Softmax를 이용한 이미지 인식

3-7. 텐서플로우로 구현

- 결과값 출력
- `print('b is ',sess.run(b))`
- `print('W is',sess.run(W))`
마지막으로 학습에서 구해진 W와 b를 출력해보자

```
Extracting /tmp/tensorflow/mnist/input_data/train-images-idx3-ubyte.gz
Extracting /tmp/tensorflow/mnist/input_data/train-labels-idx1-ubyte.gz
Extracting /tmp/tensorflow/mnist/input_data/t10k-images-idx3-ubyte.gz
Extracting /tmp/tensorflow/mnist/input_data/t10k-labels-idx1-ubyte.gz
Training
('b is ', array([-0.11368412,  0.34618276, -0.06817129, -0.12768586,  0.14948681,
                 0.36769056,  0.03084462,  0.29830146, -0.73656094, -0.1464057 ], dtype=float32))
('W is', array([[ 0.,  0.,  0., ...,  0.,  0.,  0.],
                 [ 0.,  0.,  0., ...,  0.,  0.,  0.],
                 [ 0.,  0.,  0., ...,  0.,  0.,  0.],
                 ...,
                 [ 0.,  0.,  0., ...,  0.,  0.,  0.],
                 [ 0.,  0.,  0., ...,  0.,  0.,  0.],
                 [ 0.,  0.,  0., ...,  0.,  0.,  0.]], dtype=float32))
```

3. Softmax를 이용한 이미지 인식

3-7. 텐서플로우로 구현

- 결과값 출력
- 먼저 앞에서 데이터를 로딩하도록 지정한 디렉토리에, 학습용 데이터를 다운 받아서 압축 받는 것을 확인할 수 있다. (Extracting.. 부분)
- 그 다음 학습이 끝난 후에, b와 W 값이 출력되었다. W는 784 라인이기 때문에, 중간을 생략하고 출력되었으나, 각 행을 모두 찍어보면 아래와 같이 W 값이 들어가 있는 것을 볼 수 있다.

```
(9, array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.], dtype=float32))
(10, array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.], dtype=float32))
(11, array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.], dtype=float32))
(12, array([ -5.53695190e-06, -7.94481912e-06, -1.17996518e-04,
             -1.55862726e-05, -6.33553544e-04, -7.89074238e-06,
              9.26191802e-04, -6.59273519e-06, -3.50811206e-05,
             -9.60088291e-05], dtype=float32))
(13, array([ -2.87244147e-05, -2.74839131e-05,  3.27478920e-04,
             -3.99608616e-05, -1.66243373e-03, -2.43990653e-05,
              1.85890321e-03, -2.96408980e-05, -1.02268248e-04,
             -2.71470431e-04], dtype=float32))
(14, array([ -1.84730688e-05, -1.12254556e-05,  6.51939656e-04,
             -6.49023241e-06, -3.06211441e-04, -7.92438004e-06,
             -1.88217658e-04, -1.69203449e-05, -2.83239824e-05,
             -6.81530873e-05], dtype=float32))
```

3. Softmax를 이용한 이미지 인식

3-8. 모델 검증

이제 모델을 만들고 학습을 시켰으니, 이 모델이 얼마나 정확하게 작동하는지를 테스트 해보자. `mnist.test.image` 와 `mnist.test.labels` 데이터셋을 이용하여 테스트를 진행하는데, 앞에서 나온 모델에 `mnist.test.image` 데이터를 넣어서 예측을 한 후에, 그 결과를 `mnist.test.labels` (정답)과 비교해서 정답률이 얼마나 되는지를 비교한다. 다음은 모델 테스트 코드이다. 이 코드를 위의 코드 뒤에 붙여서 실행하면 된다.

```
print ("Testing model")
# Test trained model
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('accuracy ',sess.run(accuracy, feed_dict={x: mnist.test.images,
                                                y_: mnist.test.labels}))
print ("done")
```

3. Softmax를 이용한 이미지 인식

3-8. 모델 검증

```
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
```

코드를 보자, `tf.argmax` 함수를 이해해야 하는데, `argmax(y,1)`은 행렬 `y`에서 몇 번째에 가장 큰 값이 들어가 있는지를 리턴해주는 함수이다. 아래 예제 코드를 보면

```
session = tf.InteractiveSession()

data = tf.constant([9,2,11,4])
idx = tf.argmax(data,0)
print idx.eval()

session.close()
```

3. Softmax를 이용한 이미지 인식

3-8. 모델 검증

- [9,2,11,4] 에서 최대수는 11이고, 이 위치는 두번째 (0 부터 시작한다)이기 때문에 0을 리턴한다.
- 두번째 변수는 어느 축으로 카운트를 할것인지를 선택한다. , 1차원 배열의 경우에는 0을 사용한다.
- 여기서 y는 2차원 행렬인데, 0이면 같은 열에서 최대값인 순서, 1이면 같은 행에서 최대값인 순서를 리턴한다.
- 그럼 원래 코드로 돌아오면 `tf.argmax(y,1)`은 y의 각행에서 가장 큰 값의 순서를 찾는다. y의 각행을 0~9으로 인식한 이미지의 확률을 가지고 있다.
- 아래는 4를 인식한 y 값인데, 4의 값이 0.7로 가장 높기 (4일 확률이 70%, 3일 확률이 10%, 1일 확률이 20%로 이해하면 된다.) 때문에, 4로 인식된다.

3. Softmax를 이용한 이미지 인식

3-8. 모델 검증

- [9,2,11,4] 에서 최대수는 11이고, 이 위치는 두번째 (0 부터 시작한다)이기 때문에 0을 리턴한다.
- 두번째 변수는 어느축으로 카운트를 할것인지를 선택한다. , 1차원 배열의 경우에는 0을 사용한다.
- 여기서 y는 2차원 행렬인데, 0이면 같은 열에서 최대값인 순서, 1이면 같은 행에서 최대값인 순서를 리턴한다.
- 그럼 원래 코드로 돌아오면 `tf.argmax(y,1)`은 y의 각행에서 가장 큰 값의 순서를 찾는다. y의 각행을 0~9으로 인식한 이미지의 확률을 가지고 있다.
- 아래는 4를 인식한 y 값인데, 4의 값이 0.7로 가장높기 (4일 확률이 70%, 3일 확률이 10%, 1일 확률이 20%로 이해하면 된다.) 때문에, 4로 인식된다.

0	1	2	3	4	5	6	7	8	9
0.0	0.2	0.0	0.1	0.7	0.0	0.0	0.0	0.0	0.0

3. Softmax를 이용한 이미지 인식

3-8. 모델 검증

- 여기서 `tf.argmax(y,1)`을 사용하면, 행별로 가장 큰 값을 리턴하기 때문에, 위의 값에서는 4가 리턴이된다.
- 테스트용 데이터에서 원래 정답이 4로 되어 있다면, `argmax(y_,1)`도 4를 리턴하기 때문에, `tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))`는 `tf.equals(4,4)`로 True를 리턴하게 된다.
- 모든 테스트 셋에 대해서 검증을 하고 나서 그 결과에서 True만 더해서, 전체 트레이닝 데이터의 수로 나눠 주면 결국 정확도가 나오는데, `tf.cast(boolean, tf.float32)`를 하면 텐서플로우의 bool 값을 float32 (실수)로 변환해준다. True는 1.0으로 False는 0.0으로 변환해준다. 이렇게 변환된 값들의 전체 평균을 구하면 되기 때문에, `tf.reduce_mean`을 사용한다.

3. Softmax를 이용한 이미지 인식

3-8. 모델 검증

- 여기서 `tf.argmax(y,1)`을 사용하면, 행별로 가장 큰 값을 리턴하기 때문에, 위의 값에서는 4가 리턴이 된다.
- 테스트용 데이터에서 원래 정답이 4로 되어 있다면, `argmax(y_,1)`도 4를 리턴하기 때문에, `tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))`는 `tf.equals(4,4)`로 True를 리턴하게 된다.
- 모든 테스트 셋에 대해서 검증을 하고 나서 그 결과에서 True만 더해서, 전체 트레이닝 데이터의 수로 나눠 주면 결국 정확도가 나오는데, `tf.cast(boolean, tf.float32)`를 하면 텐서플로우의 bool 값을 float32 (실수)로 변환해준다. True는 1.0으로 False는 0.0으로 변환해준다. 이렇게 변환된 값들의 전체 평균을 구하면 되기 때문에, `tf.reduce_mean`을 사용한다.

3. Softmax를 이용한 이미지 인식

3-8. 모델 검증

- 이렇게 정확도를 구하는 함수가 정의되었으면 이제 정확도를 구하기 위해 데이터타를 넣어보자
- `sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels})`
- `x`에 `mnist.test.images` 데이터셋으로 이미지 데이터를 입력받아서 `y` (예측 결과)를 계산하고, `y_`에는 `mnist.test.labels` 정답을 입력 받아서, `y`와 `y_`로 정확도 `accuracy`를 구해서 출력한다.
- 최종 출력된 `accuracy` 정확도는 0.9 로 대략 90% 정도가 나온다.

```
Testing model  
( 'accuracy ', 0.90719998)  
done
```

- 다른 알고리즘의 정확도는 http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html 를 참고하면 된다.