Deep Learning with Python

Chapter 1 DNN 01

Wedatalab 김정수

■ 목차

■ 학습목표

1. 심층신경망(DNN)의 구조와 특징을 이해하고 Tensorflow를 이용하여 XOR 문제를 해결한다.

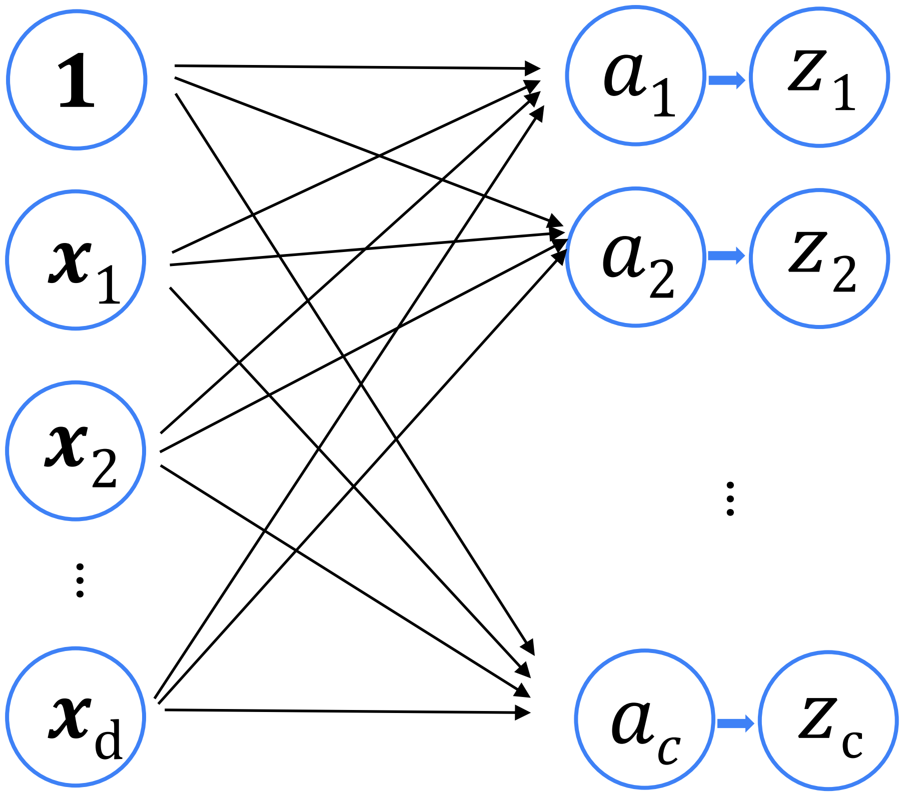
2. 신경망의 레이어가 많아 질수록 발생하는 문제점을 이해하고 DNN의 성능을 높이는 방법을 배운다.

1. Deep Neural Network 기초

1. Deep Neural Network 기초

인공신경망(ANN; Artificial Neural Network)의 구조 (1/8)

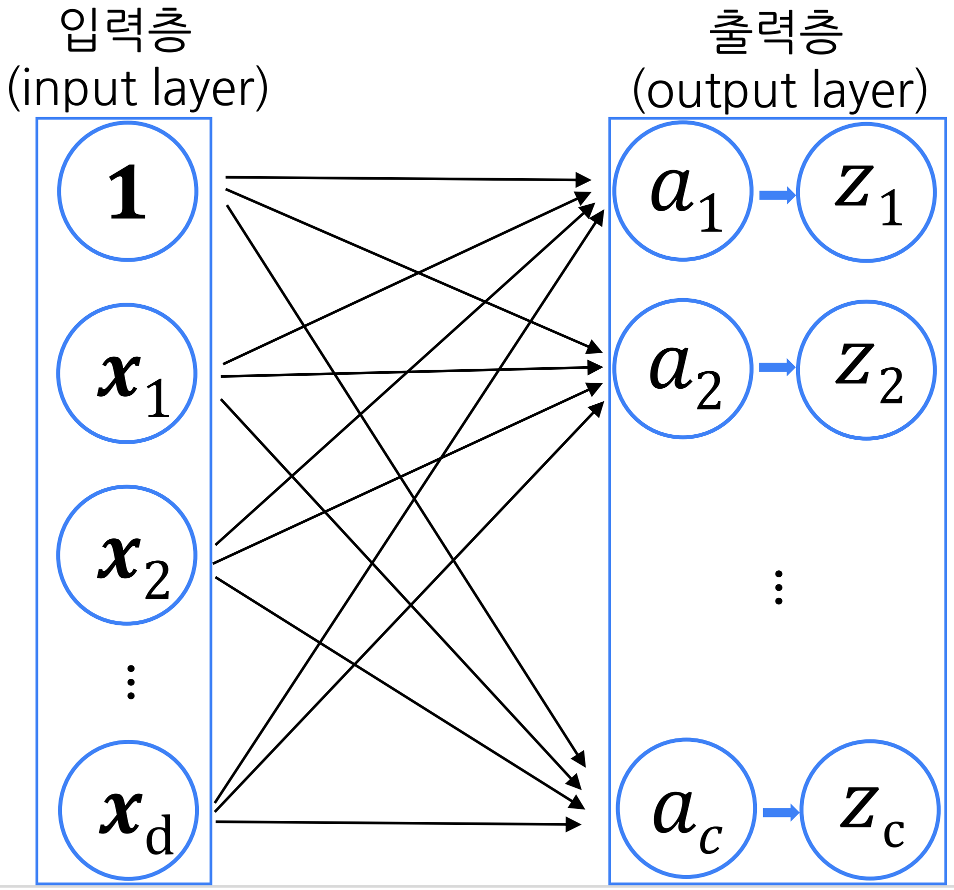
- 간단한 퍼셉트론을 생각해보자.



1. Deep Neural Network 기초

인공신경망(ANN; Artificial Neural Network)의 구조 (2/8)

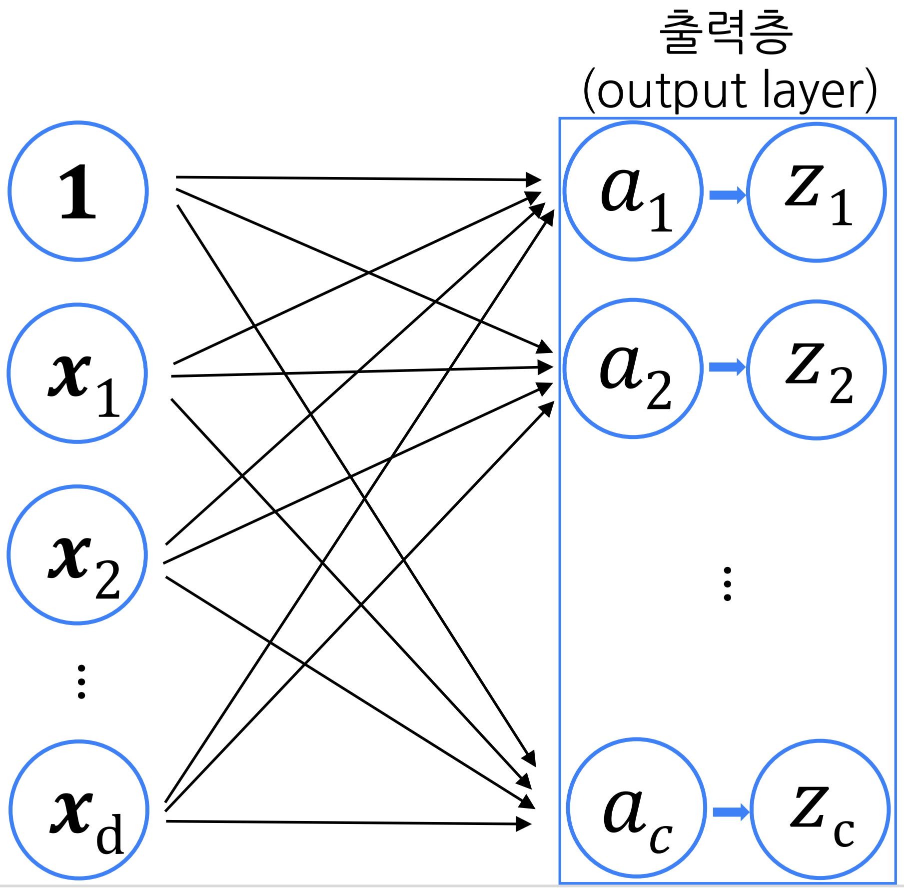
- a_1은 입력 노드들의 선형 결합, z_1은 활성화 함수를 통해 나온 결과



1. Deep Neural Network 기초

인공신경망(ANN; Artificial Neural Network)의 구조 (3/8)

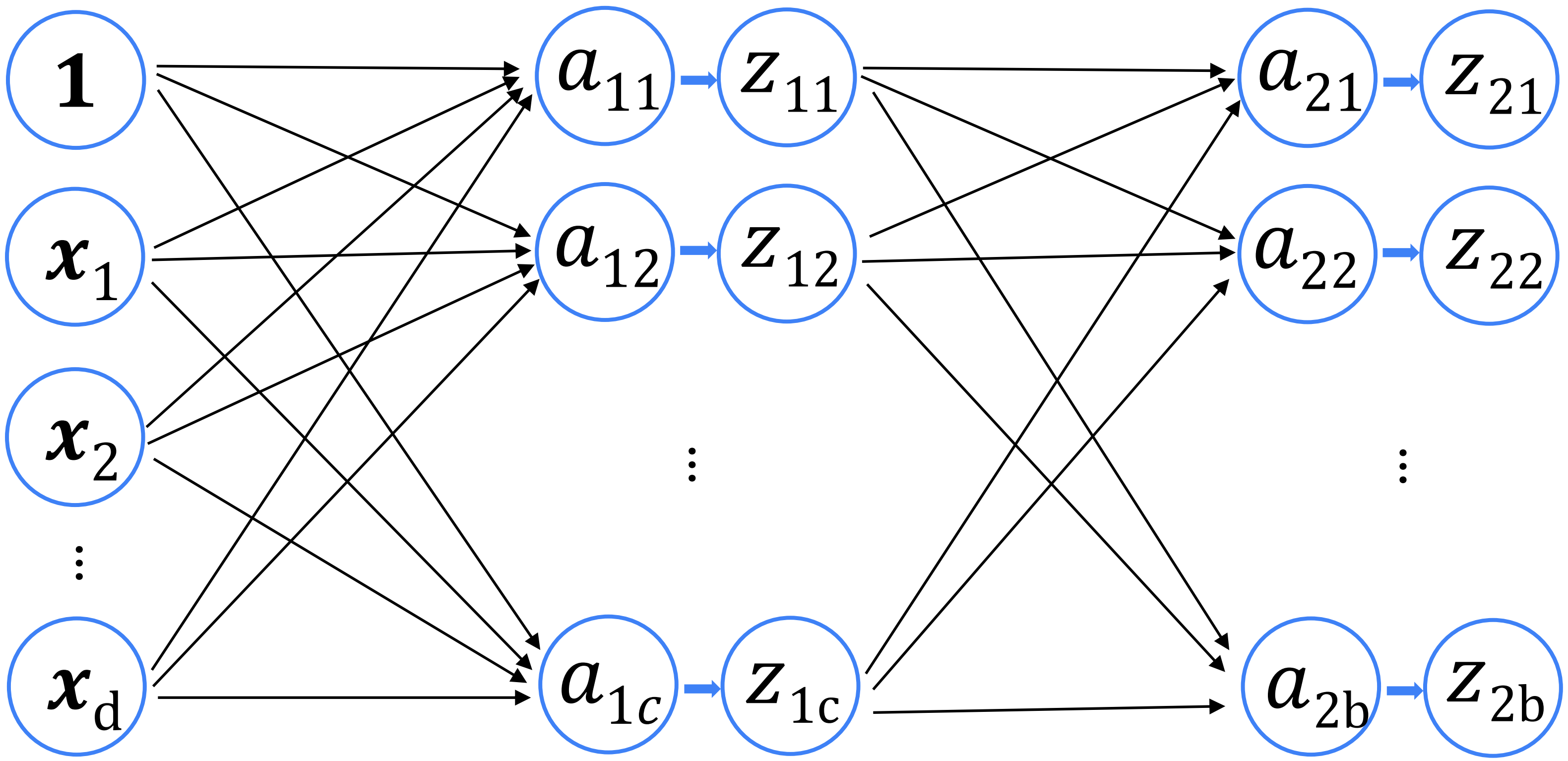
- 인공신경망은 출력층을 일종의 입력층으로 보는 것에서 시작한다.



1. Deep Neural Network 기초

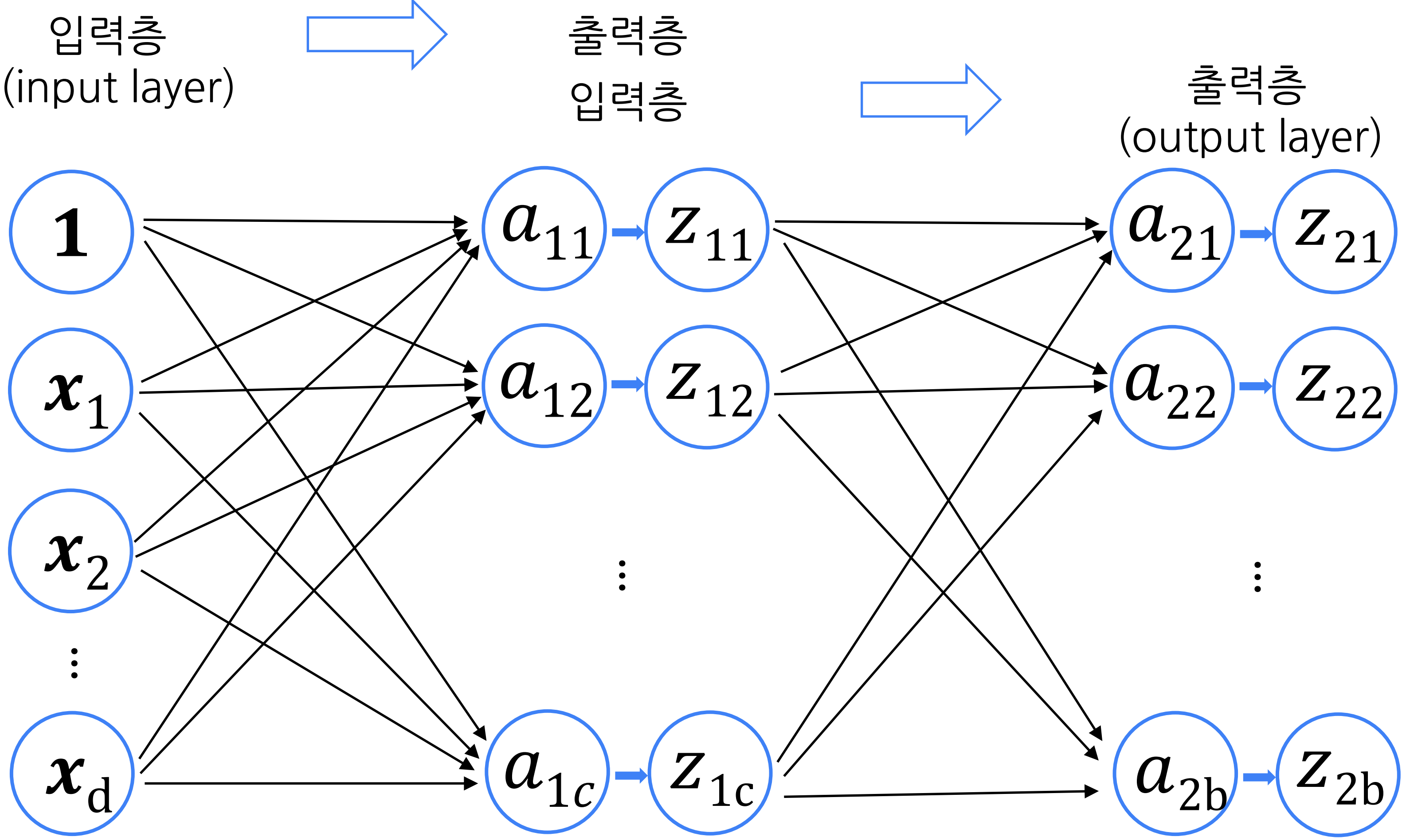
인공신경망(ANN; Artificial Neural Network)의 구조 (4/8)

- 입력층/출력층을 한번 더 쌓는다.



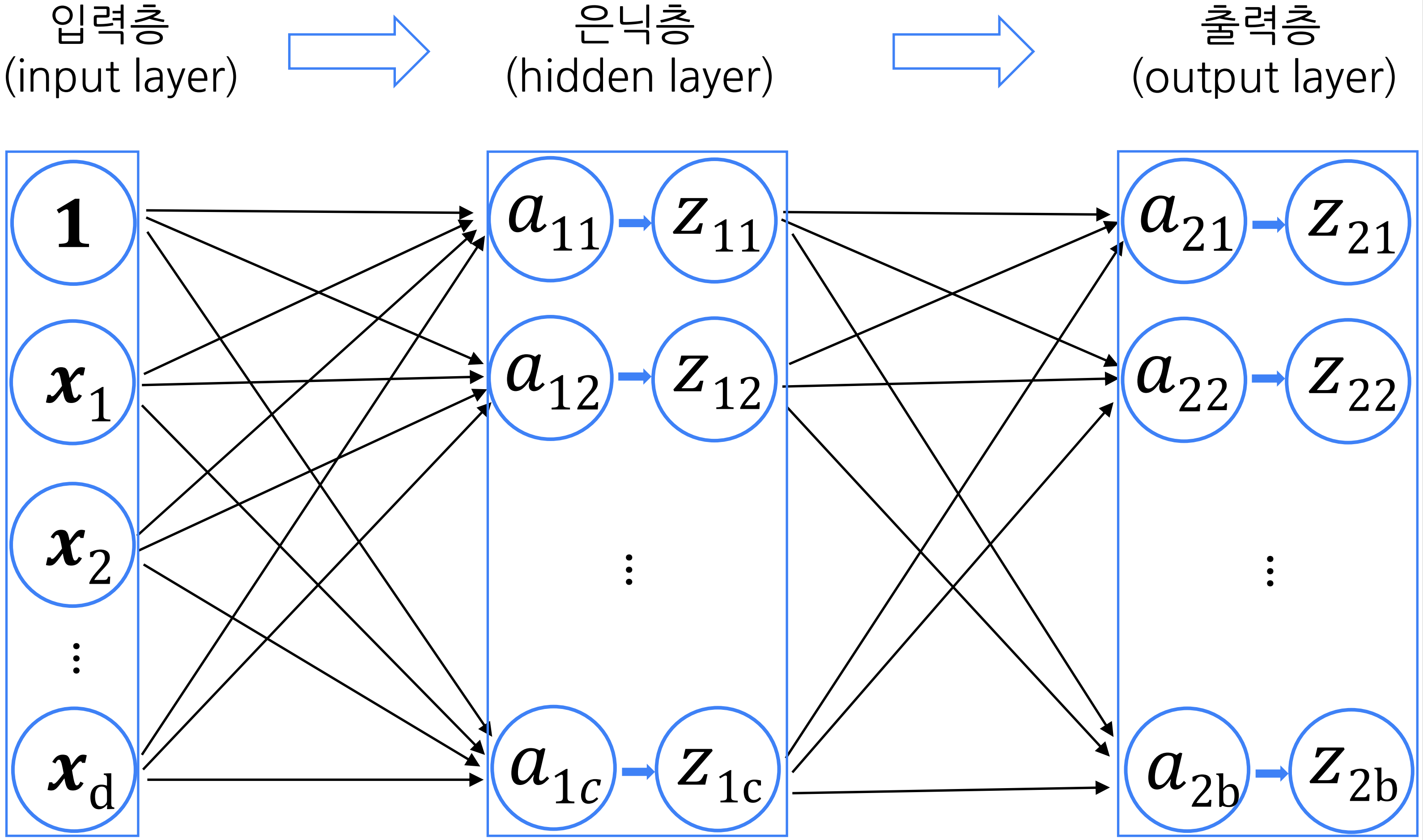
1. Deep Neural Network 기초

인공신경망(ANN; Artificial Neural Network)의 구조 (5/8)



1. Deep Neural Network 기초

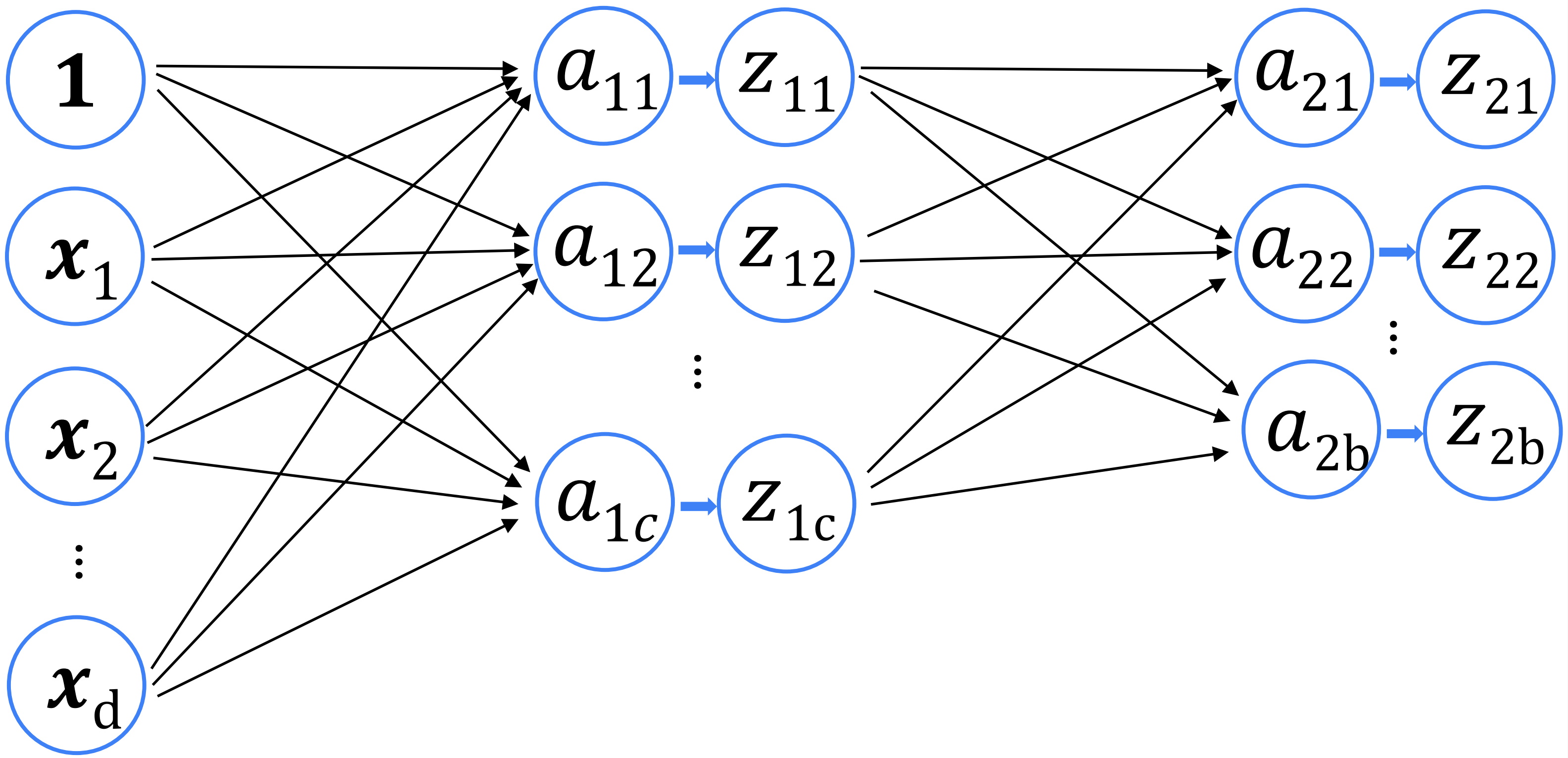
인공신경망(ANN; Artificial Neural Network)의 구조 (6/8)



1. Deep Neural Network 기초

인공신경망(ANN; Artificial Neural Network)의 구조 (7/8)

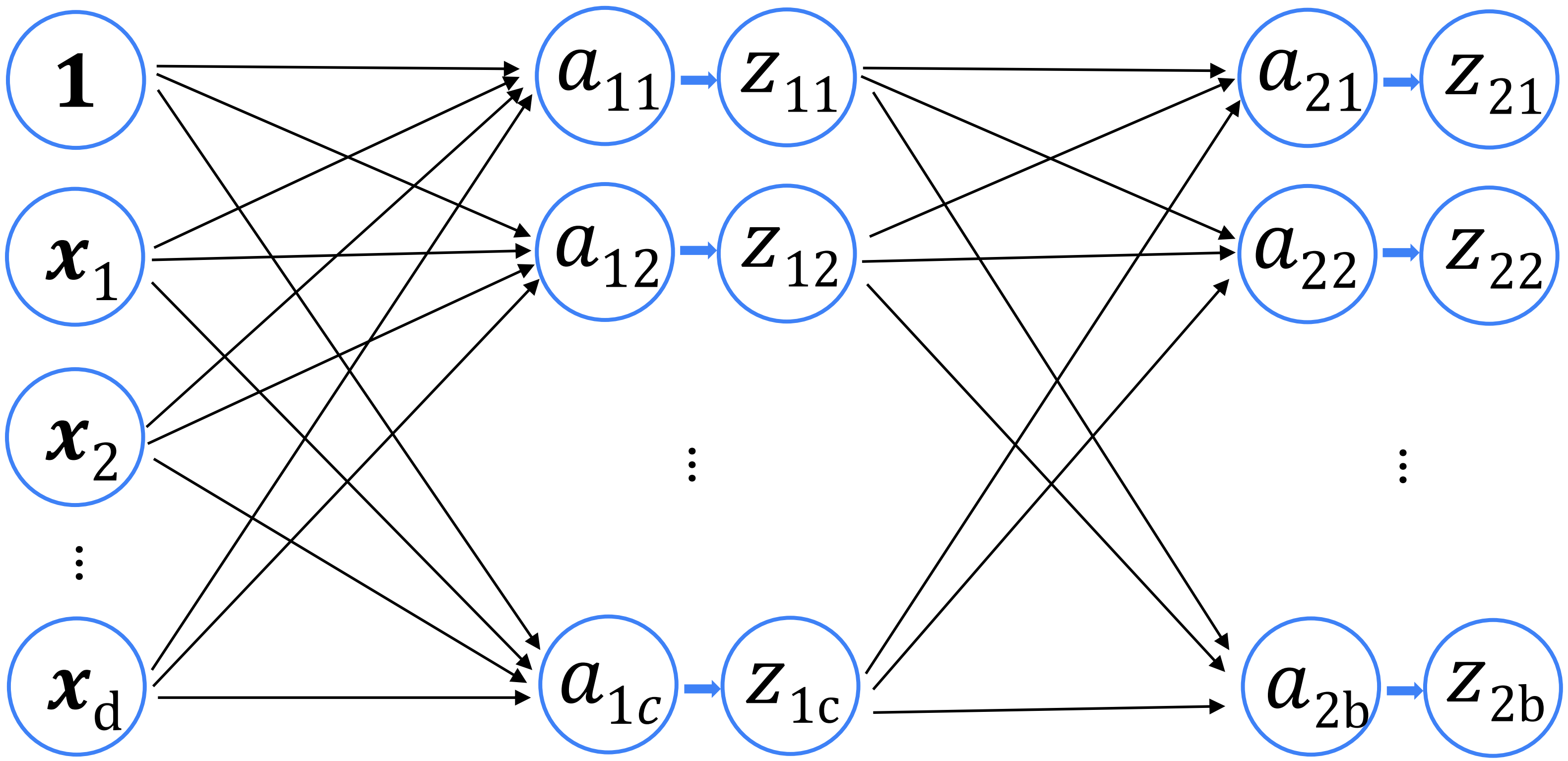
- 입력층, 은닉층, 출력층의 차원이 반드시 같을 필요는 없다.



1. Deep Neural Network 기초

인공신경망(ANN; Artificial Neural Network)의 구조 (8/8)

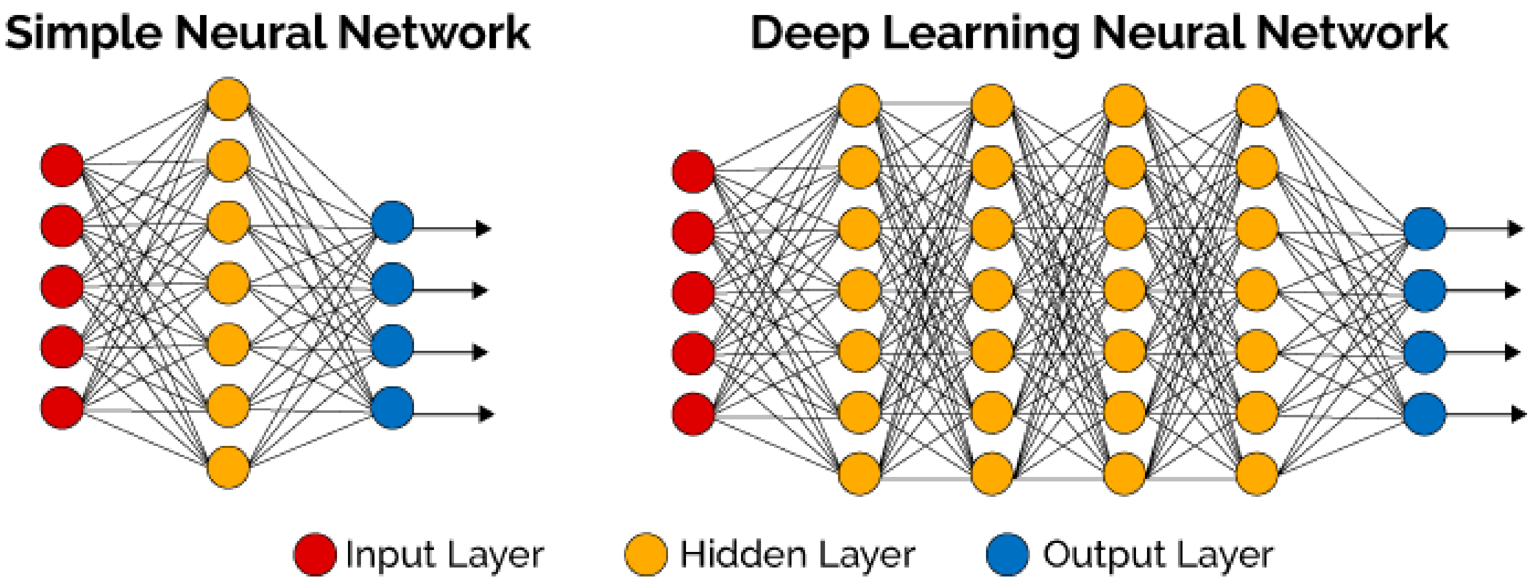
- 이와 같이 퍼셉트론을 확장하여 입력층, 은닉층, 출력층이 있는 것을 인공신경망이라 한다.



1. Deep Neural Network 기초

심층신경망(DNN; Deep Neural Network)의 개념

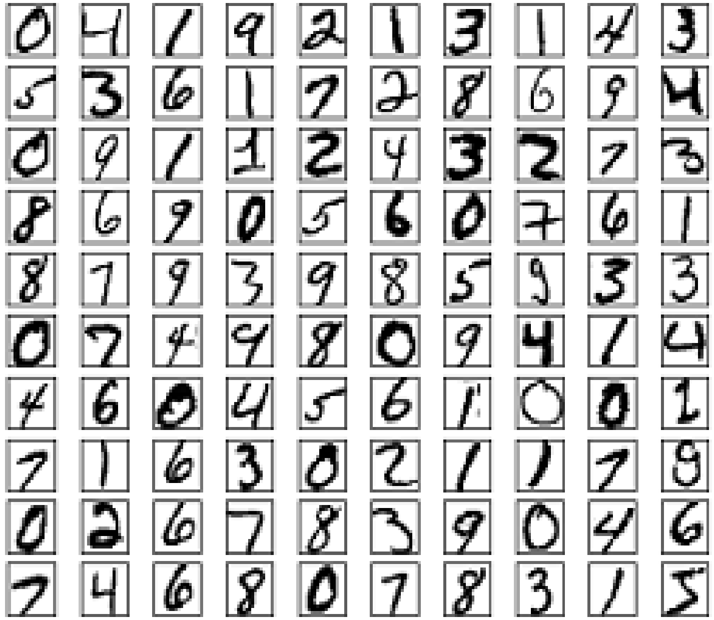
- 은닉층의 개수가 2개 이상인 것을 DNN이라 하며, DNN을 학습하는 방법을 Deep Learning이라 한다.



1. Deep Neural Network 기초

심층신경망(DNN; Deep Neural Network)의 목적

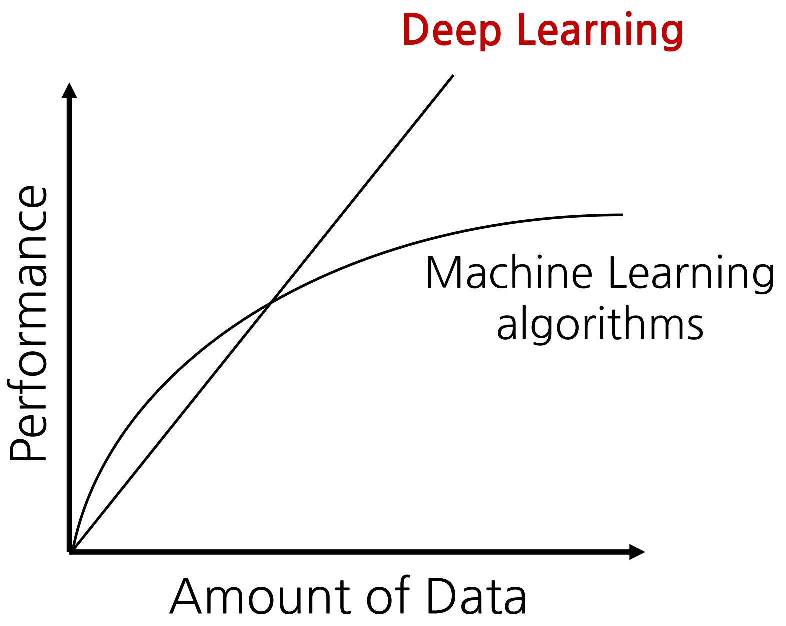
- 이미지 학습이나 문자 인식과 같은 분야에서 유용하게 쓰이고 있다.



1. Deep Neural Network 기초

심층신경망(DNN; Deep Neural Network)의 장점 (1/2)

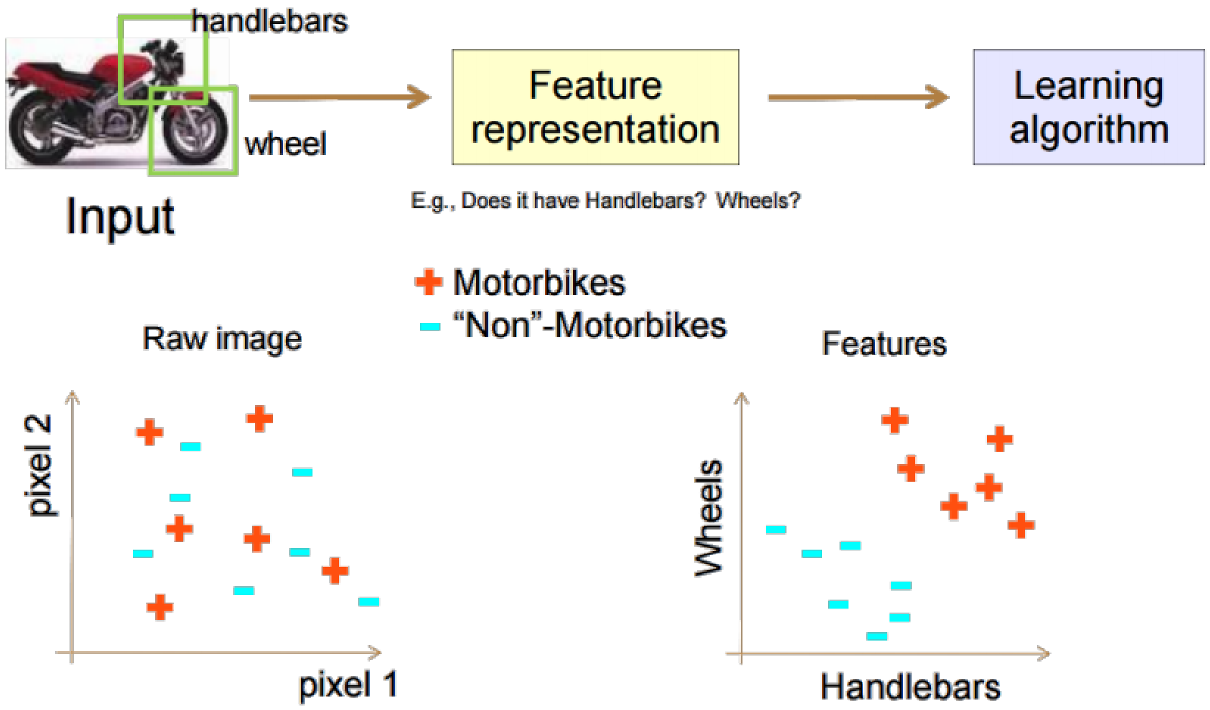
- 기존의 알고리즘들은 데이터의 양이 많아질수록 성능 향상에 한계가 있었지만 딥러닝은 기존 알고리즘들의 성능 한계를 뛰어넘었다.



1. Deep Neural Network 기초

심층신경망(DNN; Deep Neural Network)의 장점 (2/2)

- 스스로 필요한 특징을 찾아 적절하게 표현하는 학습 능력 (Feature representation)이 뛰어나다.

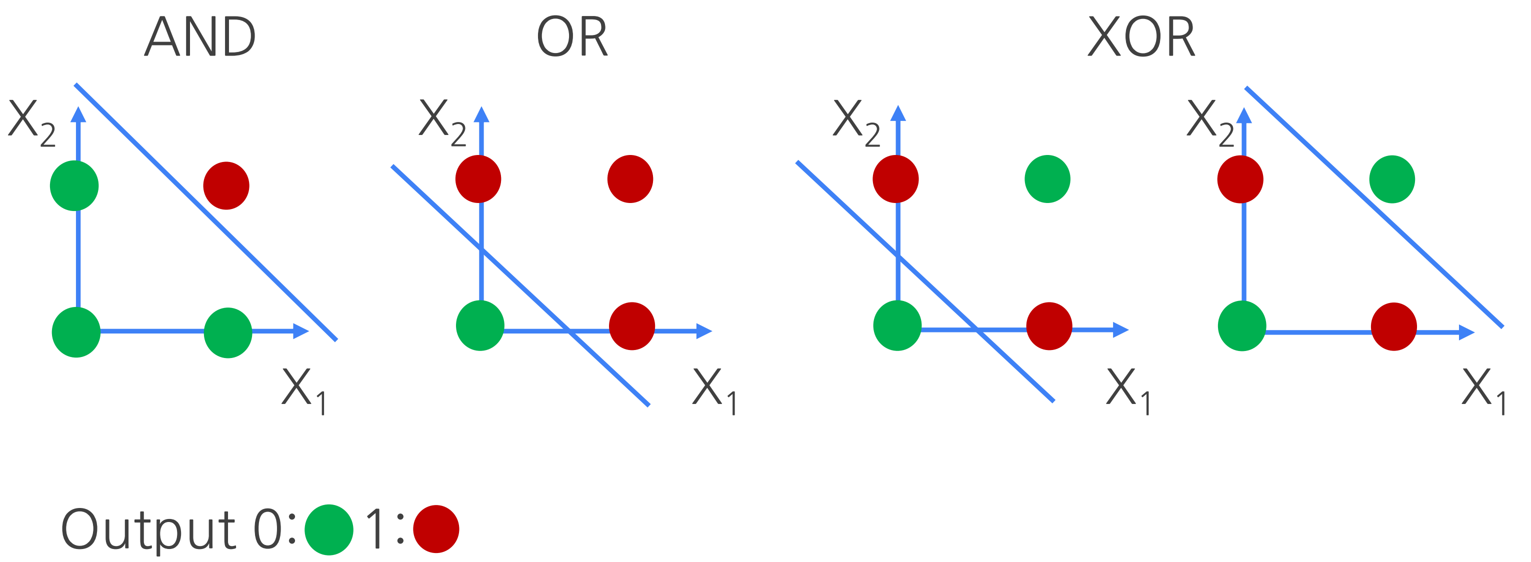


2. Tensorflow를 이용한 DNN 구현

2. Tensorflow를 이용한 DNN 구현

DNN으로 XOR 문제 해결하기 (1/2)

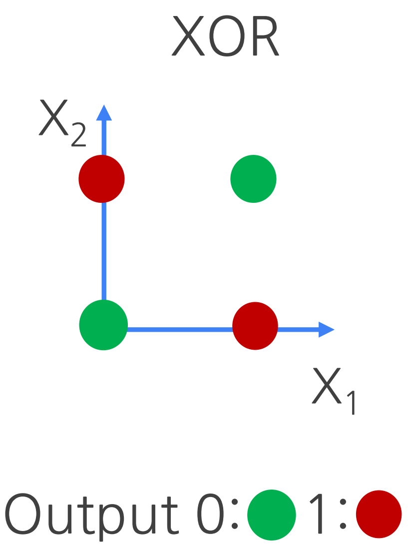
- 선형 분리가 불가능한 XOR 문제는 퍼셉트론 모델의 한계였다.



2. Tensorflow를 이용한 DNN 구현

DNN으로 XOR 문제 해결하기 (2/2)

- XOR 데이터 셋을 표로 나타내면 다음과 같다.
- Feature는 X_1, X_2이며 결과값은 Y이다.
- X_1, X_2의 값이 같으면 0 다르면 1의 결과값이다



X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

2. Tensorflow를 이용한 DNN 구현

전체 코드 (1/3)

```
import tensorflow as tf
import numpy as np

tf.reset_default_graph()
tf.set_random_seed(777)

learning_rate = 0.1
training_cnt = 10000

train_X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
train_Y = np.array([[0], [1], [1], [0]])

X = tf.placeholder(tf.float32, [None, 2])
Y = tf.placeholder(tf.float32, [None, 1])
```

2. Tensorflow를 이용한 DNN 구현

전체 코드 (2/3)

```
# Layer 1
W1 = tf.Variable(tf.random_normal([2, 2]), name='weight1')
b1 = tf.Variable(tf.random_normal([2]), name='bias1')
L1 = tf.sigmoid(tf.matmul(X, W1) + b1)

# Layer 2
W2 = tf.Variable(tf.random_normal([2, 1]), name='weight2')
b2 = tf.Variable(tf.random_normal([1]), name='bias2')
pred = tf.sigmoid(tf.matmul(L1, W2) + b2)

cost = -tf.reduce_mean(Y * tf.log(pred) + (1 - Y) * tf.log(1 - pred))
optimizer = tf.train.GradientDescentOptimizer(learning_rate)
op_train = optimizer.minimize(cost)

predicted = tf.cast(pred > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y),
dtype=tf.float32))
```

2. Tensorflow를 이용한 DNN 구현

전체 코드 (3/3)

```
sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)

for step in range(training_cnt):
    sess.run(op_train, feed_dict={X: train_X, Y: train_Y})
    if step % 1000 == 0:
        print(step, sess.run(cost, feed_dict={X: train_X, Y:
train_Y}), sess.run([W1, W2]))


p, c, a = sess.run([pred, predicted, accuracy], feed_dict={X:
train_X, Y: train_Y})
print("\nPred: ", p, "\Predicted: ", c, "\nAccuracy: ", a)
```

2. Tensorflow를 이용한 DNN 구현

모델 구축(Build graph) (1/6)

tf.set_random_seed

- 랜덤하게 생성되는 숫자들을 동일하게 생성하기 위한 것으로 실습 결과 비교를 위해 실행한다. 재실행 시 그래프 리셋이 필요하다.

```
tf.reset_default_graph()
tf.set_random_seed(777)
```

파라메터 값 설정

- 학습을 위한 기초 파라메터
- learning_rate : 값이 너무 적으면 Train 되지 않을 수 있고
  값이 너무 크면 overshooting이 발생할 수 있다.
- training_cnt : data set에 대한 training 반복 횟수

```
learning_rate = 0.1
training_cnt = 10000
```

2. Tensorflow를 이용한 DNN 구현

모델 구축(Build graph) (2/6)

트레이닝 데이터 변수 선언

- 입력으로 들어가는 train_X(input 2개), train_Y(output 1개) 설정
- numpy array를 사용

```
train_X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
train_Y = np.array([[0], [1], [1], [0]])
```

tf graph input

- X : 들어오는 row는 정해진 게 없고, column 은 2개, 즉 입력 변수 2개
- Y : 들어오는 row는 정해진 게 없고, column 은 1개 ,즉 output 1개
- matrix를 사용 하기 때문에 입력 변수를 담는 placeholder는 1개로 된다.

```
X = tf.placeholder(tf.float32, [None, 2])
Y = tf.placeholder(tf.float32, [None, 1])
```

2. Tensorflow를 이용한 DNN 구현

모델 구축(Build graph) (3/6)

tf.random_normal

- bias, weight의 초기값을 난수로 생성

```
# Layer 1
W1 = tf.Variable(tf.random_normal([2, 2]), name='weight1')
b1 = tf.Variable(tf.random_normal([2]), name='bias1')
```

sigmoid 함수 사용

- Y값이 0 또는 1의 binary 값을 갖기 때문에
  logistic regression을 활용하고 sigmoid 함수를 사용한다.

```
L1 = tf.sigmoid(tf.matmul(X, W1) + b1)
```

2. Tensorflow를 이용한 DNN 구현

모델 구축(Build graph) (4/6)

은닉층(hidden layer) 만들기

- "Layer 2"에선 첫번째 레이어의 출력값인 "L1"이 입력값이 된다.
- 마지막 레이어에선 Weight와 bias값이 1개가 되도록 설정하고 sigmoid 함수를 사용하여 모델을 정의한다.

```
# Layer 2
W2 = tf.Variable(tf.random_normal([2, 1]), name='weight2')
b2 = tf.Variable(tf.random_normal([1]), name='bias2')
pred = tf.sigmoid(tf.matmul(L1, W2) + b2)
```

2. Tensorflow를 이용한 DNN 구현

모델 구축(Build graph) (5/6)

cost/loss function 구현

- 0~1사이의 값을 근사화 하기 위해서 log함수를 사용

$$C(H(x), y) = \frac{1}{m}\sum(-ylog(H(x)) - (1 - y)\log(1 - H(x))$$

```
cost = -tf.reduce_mean(Y * tf.log(pred) + (1 - Y) *
                       tf.log(1 - pred))
```

학습 방법 ➔ cost를 최소화

- GradientDescent 함수 사용 (경사하강법)

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate)
op_train = optimizer.minimize(cost)
```

2. Tensorflow를 이용한 DNN 구현

모델 구축(Build graph) (6/6)

학습된 예측값을 0과 1로 변환

- 0~1사이로 학습된 예측값을 0과 1로 나누어 분류

```
predicted = tf.cast(pred > 0.5, dtype=tf.float32)
```

정확도

- accuracy를 계산하여 분류가 정확한지 확인
- 예측값과 실제 데이터의 일치 여부 계산
- 아래 코드는 평균을 이용한 정확도 계산

```
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y),
dtype=tf.float32))
```

2. Tensorflow를 이용한 DNN 구현

모델 실행(run/update) (1/3)

- pred은 sigmoid 함수를 통해 0~1사이의 값으로 나온다
- predicted는 pred에서 나온 값을 0과 1로 변환 시킨 값이다
- accuracy는 '예측한 Y값이 실제 Y값과 얼마나 일치하는가'이다.

```
sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)

for step in range(training_cnt):
    sess.run(op_train, feed_dict={X: train_X, Y: train_Y})
    if step % 1000 == 0:
        print(step, sess.run(cost, feed_dict={X: train_X, Y:
train_Y}), sess.run([W1, W2]))

p, c, a = sess.run([pred, predicted, accuracy], feed_dict={X:
train_X, Y: train_Y})
print("\nPred: ", p, "\nPredicted: ", c, "\nAccuracy: ", a)
```

2. Tensorflow를 이용한 DNN 구현

모델 실행(run/update) (2/3)

```
0 0.7539022 [array([[ 0.7988674 ,  0.6801188 ],
       [-1.2198634 , -0.30361032]], dtype=float32), array([[ 1.3752297 ],
       [-0.78823847]], dtype=float32)]
1000 0.67122906 [array([[ 1.1574563 ,  0.70470023],
       [-1.8544798 , -0.15281102]], dtype=float32), array([[ 1.3886284],
       [-0.8744256]], dtype=float32)]
2000 0.53393596 [array([[ 3.053097 ,  1.3885086],
       [-3.4558659, -0.6531137]], dtype=float32), array([[ 3.2834744],
       [-1.5618594]], dtype=float32)]
3000 0.19786675 [array([[ 4.6234245,  3.502949 ],
       [-4.752163 , -3.0361452]], dtype=float32), array([[ 5.564906],
       [-4.344725]], dtype=float32)]
4000 0.07765331 [array([[ 5.282392 ,  4.63241  ],
       [-5.363648 , -4.2742133]], dtype=float32), array([[ 7.0586023],
       [-6.2991076]], dtype=float32)]
5000 0.04536082 [array([[ 5.610321 ,  5.1595116],
       [-5.695793 , -4.8254967]], dtype=float32), array([[ 7.971818 ],
       [-7.3488626]], dtype=float32)]
6000 0.031536568 [array([[ 5.819358 ,  5.4801764],
       [-5.914306 , -5.1571383]], dtype=float32), array([[ 8.610341],
       [-8.044239]], dtype=float32)]
```

Cost

W1값

W2값

시행 횟수

2. Tensorflow를 이용한 DNN 구현

모델 실행(run/update) (3/3)

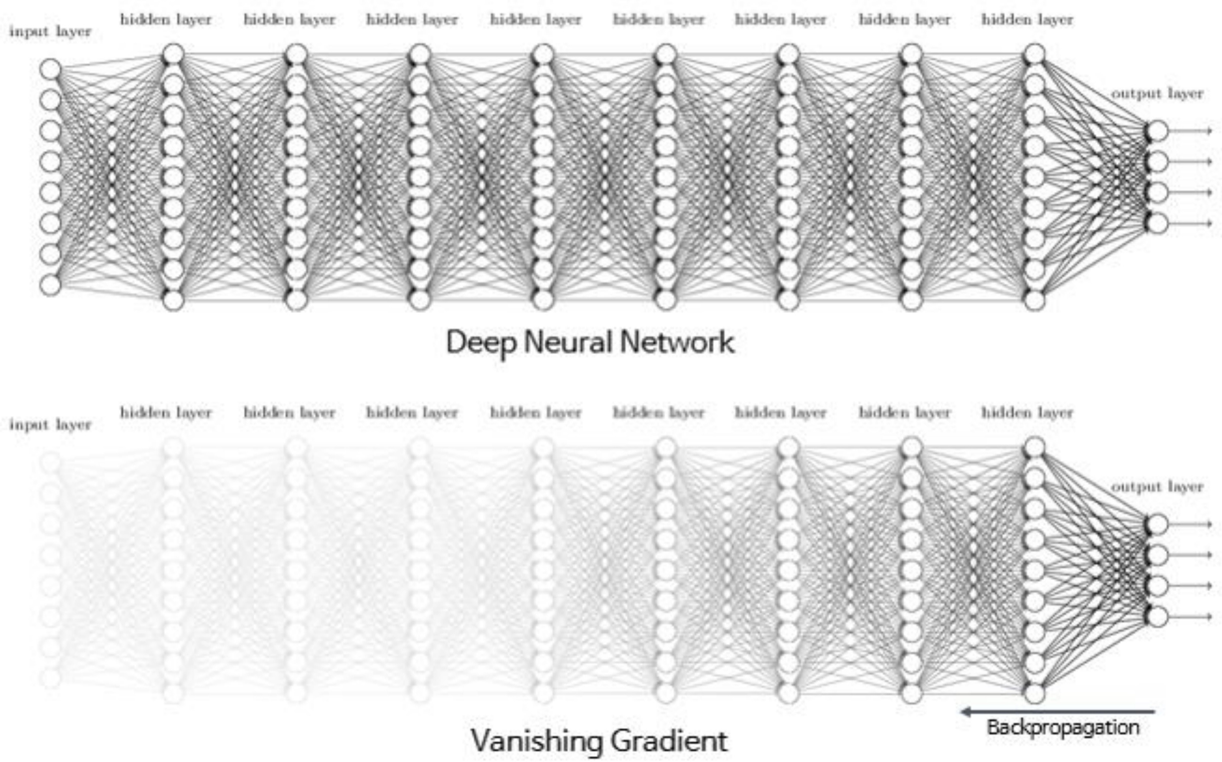
```
7000 0.02401754 [array([[ 5.970447 ,  5.7047625],
       [-6.074383 , -5.3883605]], dtype=float32), array([[ 9.097483],
       [-8.560672]], dtype=float32)]
8000 0.019331548 [array([[ 6.087807 ,  5.8752766],
       [-6.199533 , -5.5634637]], dtype=float32), array([[ 9.490048 ],
       [-8.9703865]], dtype=float32)]
9000 0.016145576 [array([[ 6.183277 ,  6.0115905],
       [-6.3016844, -5.7032   ]], dtype=float32), array([[ 9.818282],
       [-9.309538]], dtype=float32)]
10000 0.013844791 [array([[ 6.263471 ,  6.1245112],
       [-6.3876433, -5.818806 ]], dtype=float32), array([[10.100041],
       [-9.598662]], dtype=float32)]

Pred:  [[0.01338218]
 [0.98166394]
 [0.98809403]
 [0.01135799]]
Predicted:  [[0.]
 [1.]
 [1.]
 [0.]]
Accuracy:  1.0
```

3. DNN 성능 높이기

3. DNN 성능 높이기

DNN 학습 시 문제 1

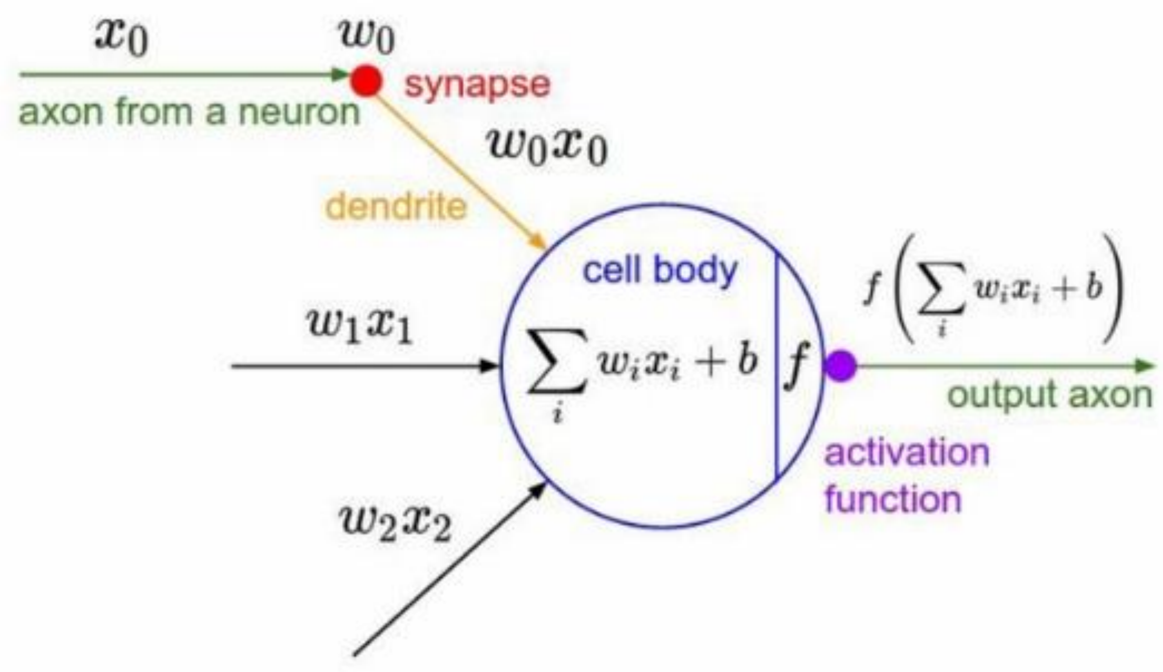
* 기울기 소실(Vanishing Gradient) 또는 폭주(Exploding) 발생가능성
* 모델이 복잡하고 커질수록 학습시간이 증가한다.



3. DNN 성능 높이기

Activation Function

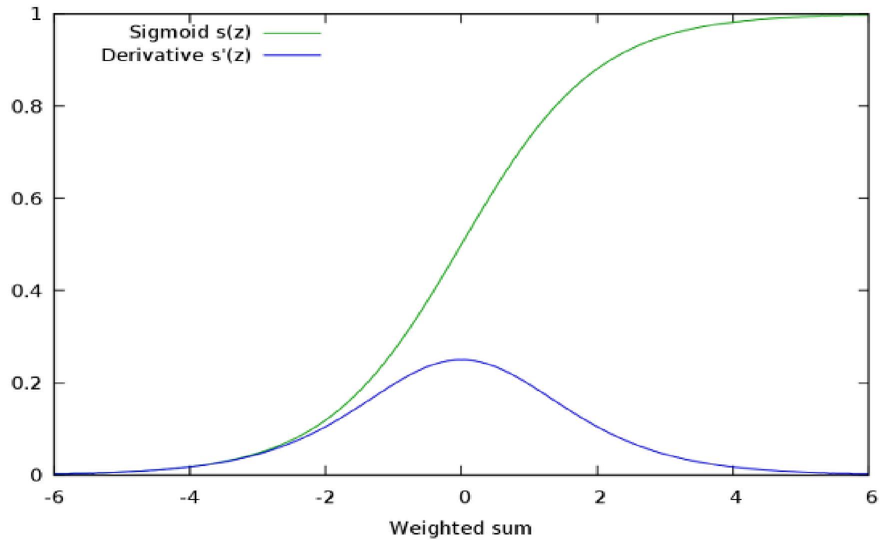
- 입력 신호의 총합($\sum_i w_i x_i + b$)을 출력 신호로 변환하는 함수(f)를 활성화 함수(Activation function)이라 한다.



3. DNN 성능 높이기

Sigmoid 함수의 문제점

- Sigmoid 함수를 미분하면 0~0.25 사이의 값을 가진다
- Backpropagation 수행 시 hidden layer수에 비례하여 미분 값이 곱해지며 앞으로 전달된다
- 0~0.25 사이의 값이 계속 곱해지며 앞 layer로 갈 수록 기울기 (gradient)가 0에 수렴한다
  결국 **기울기 소실(Gradient Vanishing) 문제가 발생**한다.

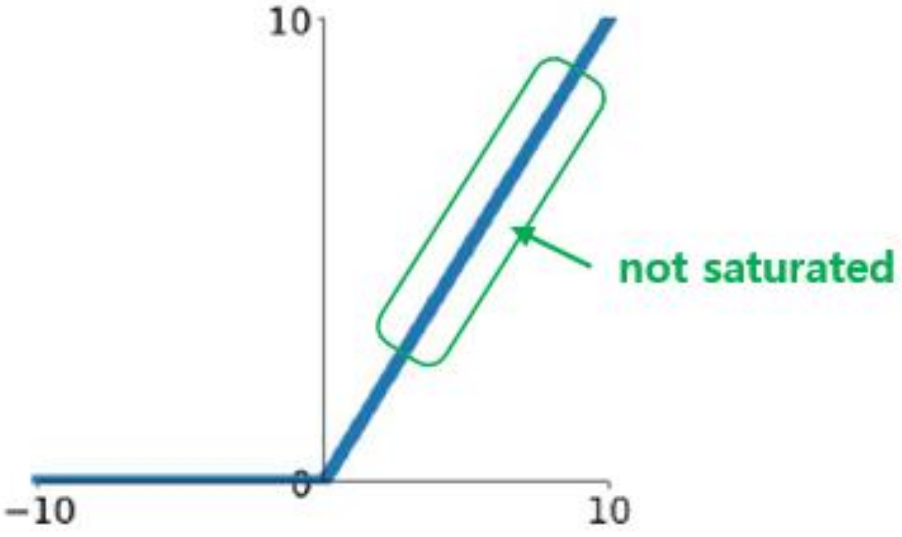


3. DNN 성능 높이기

ReLU(Rectified Linear Unit) 함수

- ReLU함수는 입력이 0보다 크면 입력을 그대로 출력, 0이하면 0을 출력하므로 0보다 큰 곳에서 수렴하는 구간이 없다.(not saturated)

* 입력값을 그대로 출력으로 내보내기 때문에 시그모이드 함수에 비해 계산 속도가 빠르다.

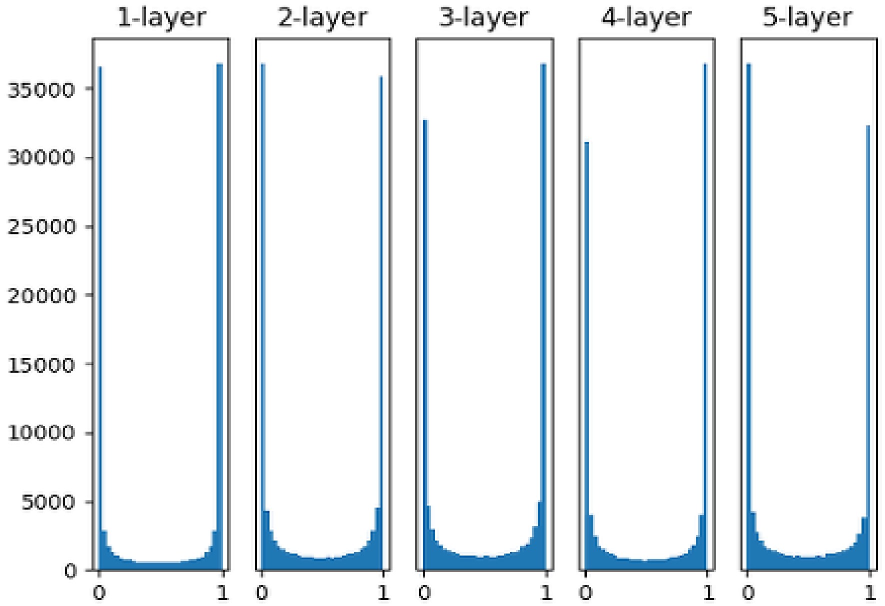


$$\mathrm{ReLU}(\boldsymbol{x}) = \max(0, \boldsymbol{x})$$

3. DNN 성능 높이기

Weight Initialization의 중요성 (1/2)

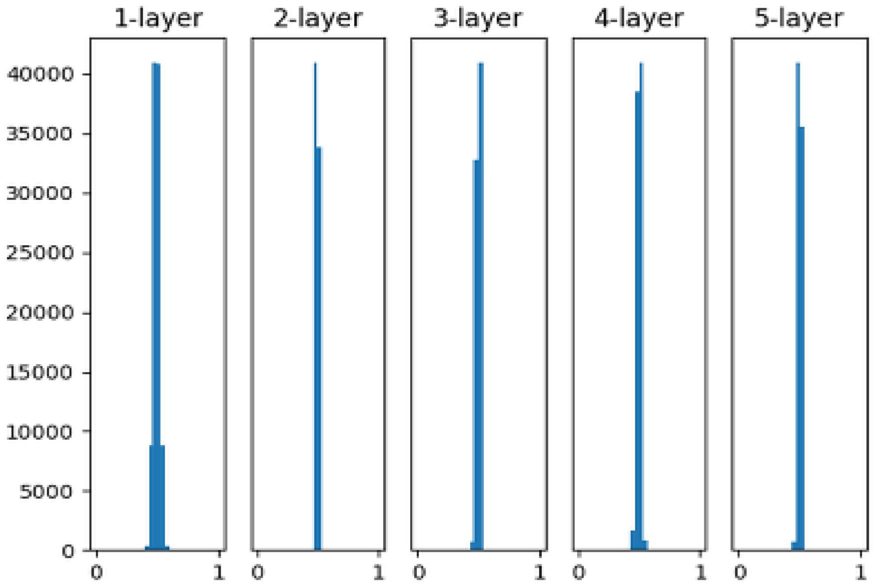
- 깊은 신경망 학습에선 가중치 초기화를 어떻게 하느냐에 따라 학습 성능이 달라질 수 있다.
- 가중치를 표준편차가 1인 정규분포로 초기화한 경우 Sigmoid 함수의 활성화 값 분포는 아래 그림처럼 0과 1에 치우친다.
- 이 경우 Gradient Vanishing(기울기 소실)문제가 발생한다.



3. DNN 성능 높이기

Weight Initialization의 중요성 (2/2)

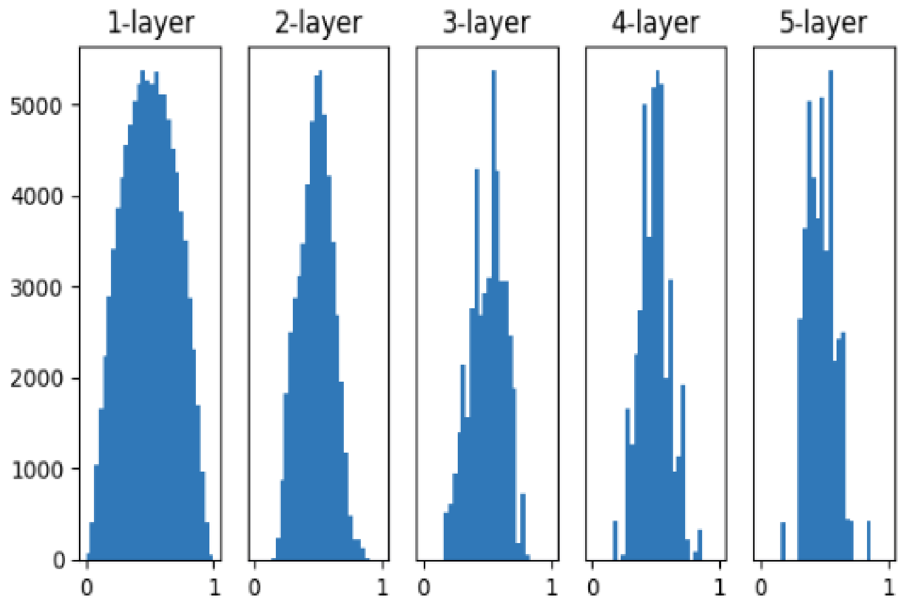
- 가중치를 표준편차가 0.01인 정규분포로 초기화한 경우 sigmoid 함수의 활성화 값은 0.5 주변에 밀집하게 된다.
- 이 경우 Gradient Vanishing(기울기 소실)은 발생하지 않지만 뉴런이 다양한 값을 표현하지 못하게 된다.
- **뉴런의 수가 많은 DNN에선 활성화 값이 넓게 분포되어야 성능이 좋다**.



3. DNN 성능 높이기

Xavier Initializer (1/2)

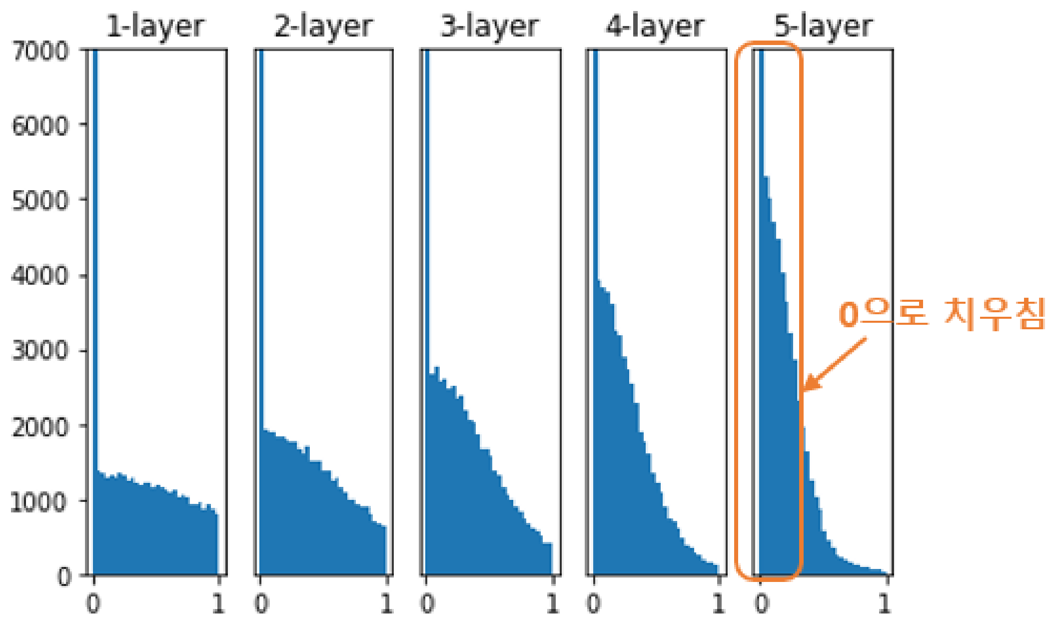
- sigmoid처럼 중앙부분이 선형 모양인 함수에는 Xavier 초기화를 사용 - Xavier 초기화는 가중치를 표준편차가 $\frac{1}{\sqrt{n}}$인 정규분포로 초기화한다. (n은 입력층의 노드 수)
- **이 경우 Sigmoid 활성화 값이 넓게 분포되어 DNN의 성능이 좋아진다.**



3. DNN 성능 높이기

Xavier Initializer (2/2)

- Xavier 가중치 초기화는 ReLU활성화 함수를 사용했을 때 레이어가 깊어질 수록 출력값이 0으로 치우치는 문제가 발생한다.

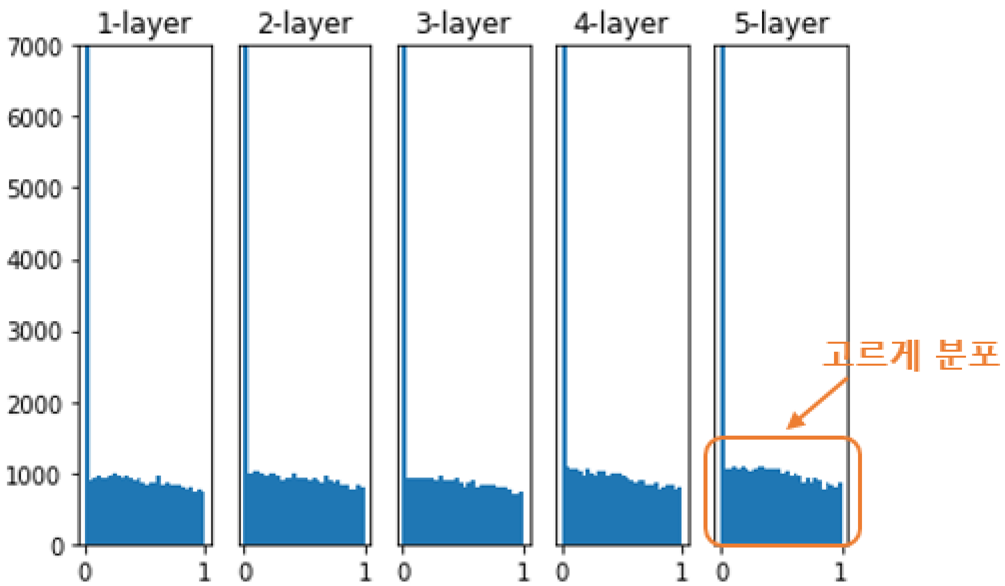


<ReLU함수에서 Xavier 초기화의 문제>

3. DNN 성능 높이기

He Initializer

- He 초기화는 Xavier 초기화에 $\sqrt{2}$를 곱해준 것이다. 즉 표준편차가 $\frac{2}{\sqrt{n}}$인 정규분포로 초기화한다. (n은 입력층의 노드수)
- ReLU는 입력이 음수일 때 출력이 모두 0이기 때문에 $\sqrt{2}$를 곱함으로써 더 넓게 분포시킨다.

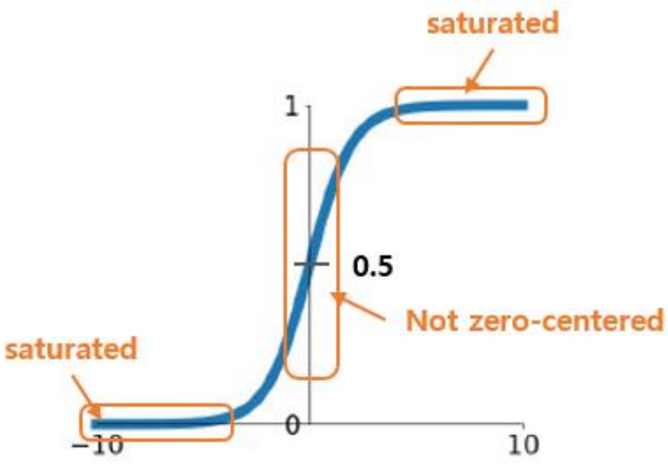


<ReLU함수에서 He 가중치 초기화 사용>

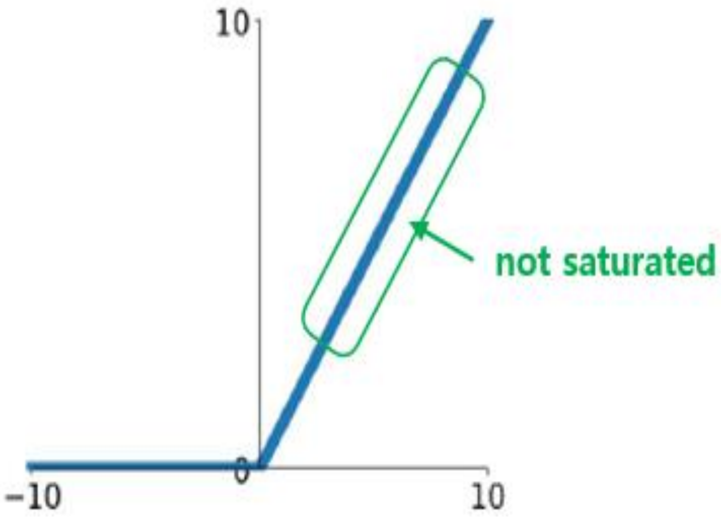
3. DNN 성능 높이기

Weight Initialization 정리

* Xavier Initializer는 Sigmoid, Tanh 함수처럼 가운데 부분이 선형이며 수렴하는 활성화 함수에 사용한다.
* He Initializer는 ReLU 함수처럼 수렴하지 않는 활성화 함수에 사용한다.



<Sigmoid 함수>

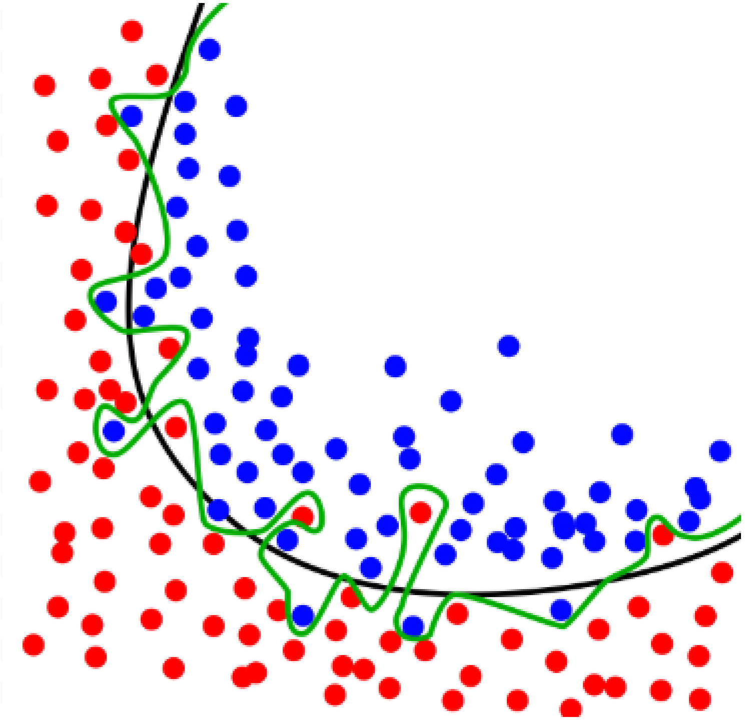


<ReLU 함수>

3. DNN 성능 높이기

DNN 학습 시 문제 2

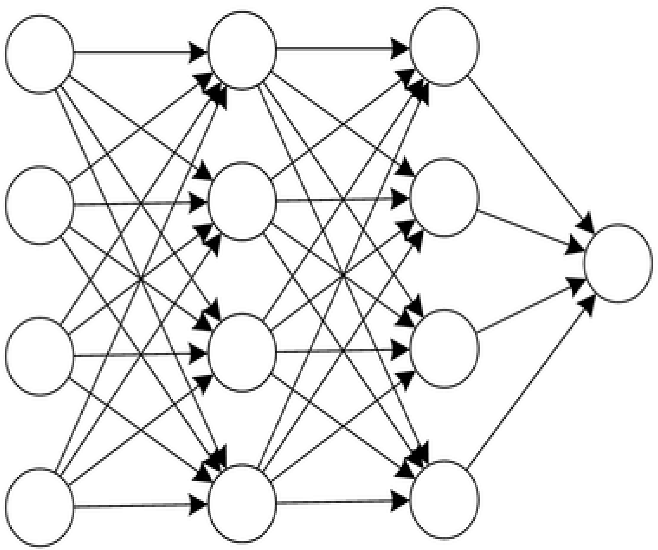
- Overfitting은 학습을 지나치게 많이 하면 오히려 학습의 정확성이 떨어지는 문제이다.
- 학습량을 적절히 조절하여 해결하는 방법이 있다.



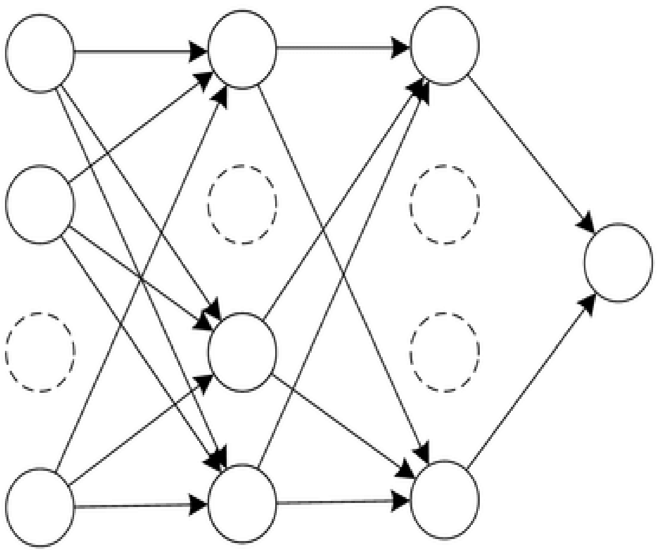
3. DNN 성능 높이기

Dropout 기법

- Dropout은 Overfitting이 일어나지 않도록 중간 중간 무작위로 뉴런을 비활성화하여 학습 효율을 향상 시키는 방법이다.



(a) Standard Neural Network



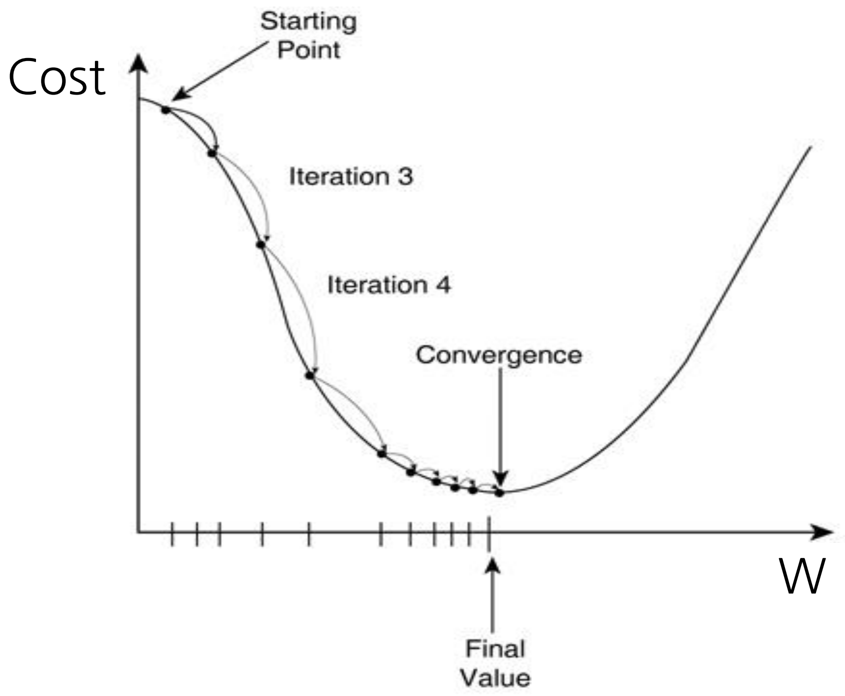
(b) Network after Dropout

4. Hyperparameter 최적화

4. Hyperparameter 최적화

Learning rate

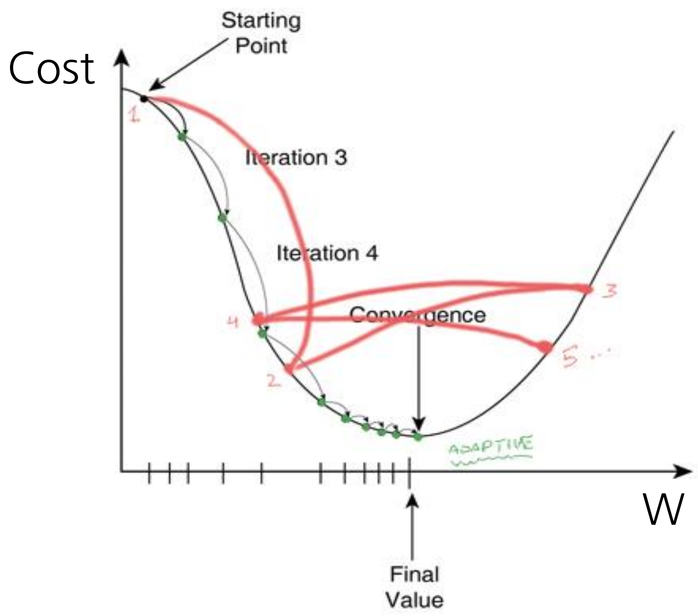
- 머신러닝에서는 학습 모델을 통해서 예측한 값과 실제 값 사이에서 비용을 산출하여 비용이 최소화되는 방향으로 학습을 전개한다.
- 학습을 수행할 때마다 다음 point의 보폭을 결정하는 것을 Learning rate라고 한다.

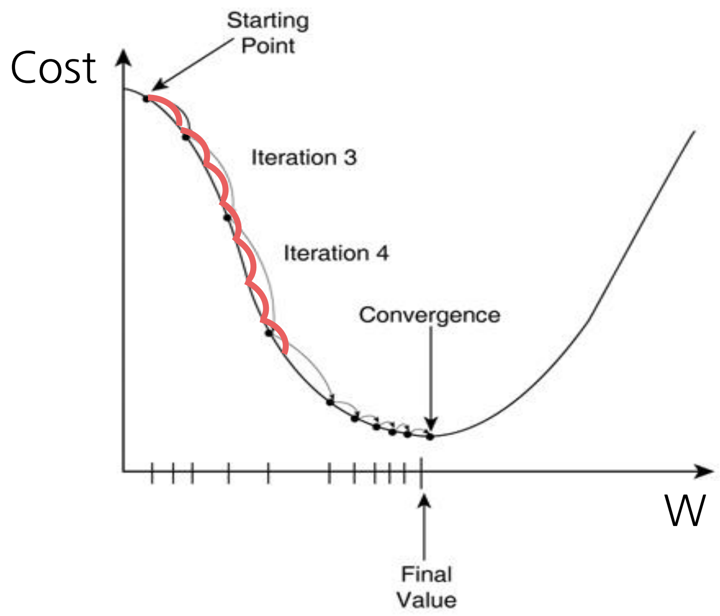


4. Hyperparameter 최적화

Learning rate 설정 시 주의할 점

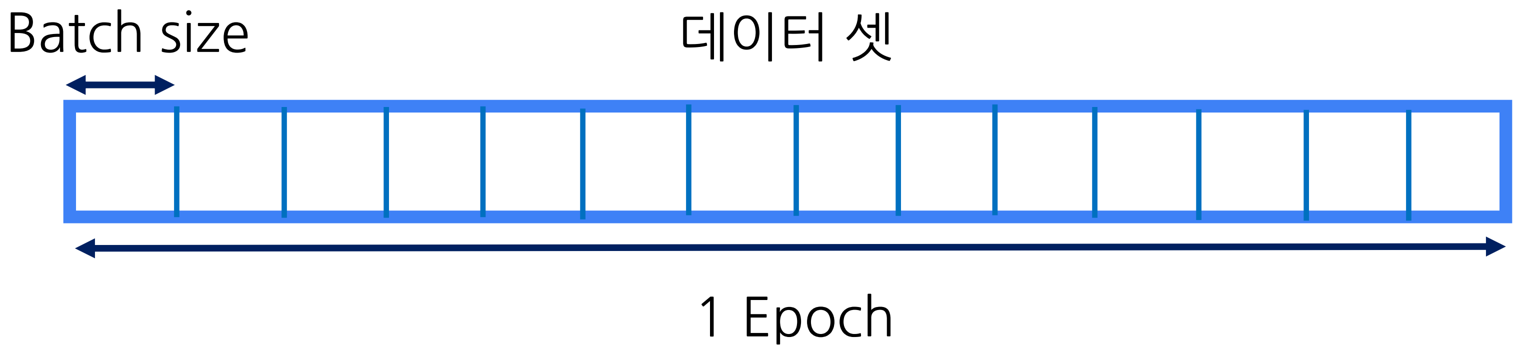
1) Learning rate가 너무 클 때 최솟값을 제대로 찾지못하고 아예 밖으로 나가버릴 수도 있다. 이러한 현상을 'Overshooting'이라 한다.
2) Learning rate가 너무 작을 때 최솟값까지 가는데 너무 오랜 시간이 걸리며, local minimum애 빠질 수 도 있다.
3) 적절한 Learning rate를 설정하기 위해선 경험이 중요하다.





4. Hyperparameter 최적화

Epoch, Batch size, Iteration



위 그림을 전체 데이터 셋이라고 하면 각 의미는 다음과 같다.
1 Epoch : 전체 데이터 셋을 한 번 학습하는 것
Batch size : 1회 학습 시 사용되는 데이터의 수
Iteration : 정해진 Batch size를 이용하여 학습을 반복하는 횟수

예를 들어 전체 데이터 셋이 1000개, Batch size가 20개라고 하면
1 Epoch = 50 Iteration

감사합니다.

참고자료(Reference)

1. “모두를 위한 딥러닝 강좌 시즌 1”, 김성훈

2. DNN 이미지

https://cdn-images-1.medium.com/max/1600/1*r0fxAZRpRGapPnC4bniDiQ.png

3. DNN 특징

https://blog.naver.com/wjddudwo209/220634377686

4. 이미지 넷

http://www.image-net.org/

5. DNN MNIST 데이터 셋

http://goodtogreate.tistory.com/480

6. 과적합 이미지

https://qph.fs.quoracdn.net/main-qimg-5c2456c21e58cf206a451ea7ca01750e

7. 가중치 초기화 개념 및 이미지

http://excelsior-cjh.tistory.com/177

https://t1.daumcdn.net/cfile/tistory/994C2F3C5AB623C526

https://t1.daumcdn.net/cfile/tistory/993C01365AB6262903

http://img1.daumcdn.net/thumb/R1920x0/?fname=http%3A%2F%2Fcfile5.uf.tistory.com%2Fimage%2F99585B445BB6EC2513865F

참고자료(Reference)

8. Gradient Vanishing 이미지

http://img1.daumcdn.net/thumb/R1920x0/?fname=http%3A%2F%2Fcfile2.uf.tistory.com%2Fimage%2F997E1B4C5BB6EAF239A91B

9. 활성화 함수 이미지

http://img1.daumcdn.net/thumb/R1920x0/?fname=http%3A%2F%2Fcfile10.uf.tistory.com%2Fimage%2F99FC1C425BB6EB150A6F6E

10. Dropout 이미지

https://www.researchgate.net/profile/Amine_Ben_khalifa/publication/309206911/figure/fig3/AS:418379505651712@1476760855735/Dropout-neural-network-model-a-is-a-standard-neural-network-b-is-the-same-network.png

11. Learning rate 이미지

http://www.yaldex.com/game-development/1592730043_ch18lev1sec4.html