12. Softmax Classification

- Softmax란?
- Cost function의 변경
- Softmax Classification의 구현
- 모델의 저장과 복원
- 신경망의 정확도 향상을 위한 방법들

(1) Softmax Classification

(Multinomial Classification)

Softmax란?

- $\begin{pmatrix} w_{A1} & w_{A2} & w_{A3} \\ w_{B1} & w_{B2} & w_{B3} \\ w_{C1} & w_{C2} & w_{C3} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} w_{A1}x_1 + w_{A2}x_2 + w_{A3}x_3 \\ w_{B1}x_1 + w_{B2}x_2 + w_{B3}x_3 \\ w_{C1}x_1 + w_{C2}x_2 + w_{C3}x_3 \end{pmatrix} = \begin{pmatrix} \bar{y}_A \\ \bar{y}_B \\ \bar{y}_C \end{pmatrix}$
  - $WX = Y$
  - $\begin{pmatrix} \bar{y}_A \\ \bar{y}_B \\ \bar{y}_C \end{pmatrix} = \begin{pmatrix} 2.0 \\ 1.0 \\ 0.1 \end{pmatrix} = \begin{pmatrix} ? \\ ? \\ ? \end{pmatrix}$ $\begin{pmatrix} \bar{y}_A \\ \bar{y}_B \\ \bar{y}_C \end{pmatrix} = \begin{pmatrix} 2.0 \\ 1.0 \\ 0.1 \end{pmatrix} = \begin{pmatrix} 0.7 \\ 0.2 \\ 0.1 \end{pmatrix}$
    - 이 수식의 값 2.0, 1.0, 0.1을 0~1사이의 값으로 변경할 수 없을까?
    - 즉, 값을 확률(Probability)로 변경할 수 없을까?
      - 모두 더하면 1이 되도록...

- 확률 $S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_i}}$

Softmax

- $\begin{pmatrix} \bar{y}_A \\ \bar{y}_B \\ \bar{y}_C \end{pmatrix} = \begin{pmatrix} 2.0 \\ 1.0 \\ 0.1 \end{pmatrix} \Rightarrow \boldsymbol{argmax} \begin{pmatrix} 0.7 \\ 0.2 \\ 0.1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$

  S(y) L(Label)

  - 계산결과를 확률로 표현하고,
  - 이중 가장 max인 값을 1로 나머지를 0으로하는 것을 말함

Multinomial Classification

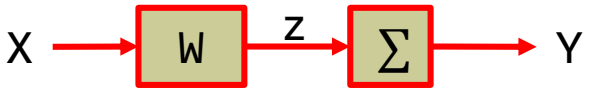
- Class가 N개인 경우
  - 주어진 데이터가 A, B, C 중 어느 클래스에 속하는가?
- (예)

hours(x1)	attendance(x2)	grade(y)
10	5	A
9	5	A
3	2	B
2	4	B
11	1	C

N개의 binary classifier 결합

Class가 2개인 경우

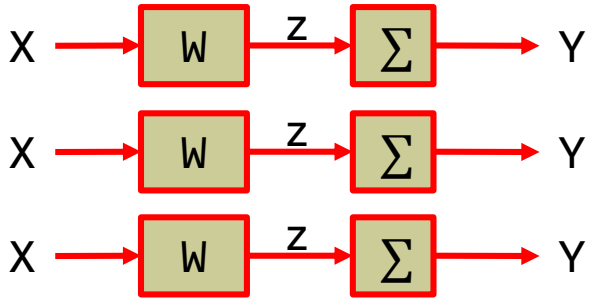
- $H(X) = \frac{1}{1+e^{-W^T X}}$



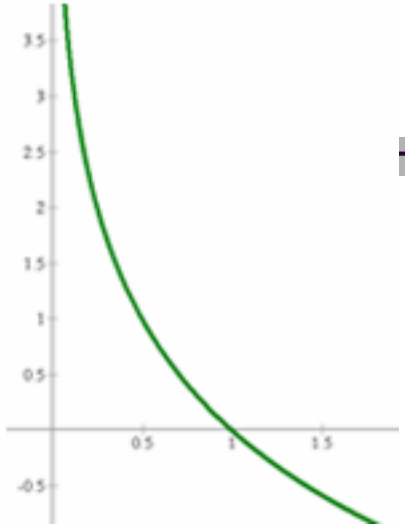
Class가 N개인 경우

- $\begin{pmatrix} w_{A1} & w_{A2} & w_{A3} \\ w_{B1} & w_{B2} & w_{B3} \\ w_{C1} & w_{C2} & w_{C3} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} w_{A1}x_1 + w_{A2}x_2 + w_{A3}x_3 \\ w_{B1}x_1 + w_{B2}x_2 + w_{B3}x_3 \\ w_{C1}x_1 + w_{C2}x_2 + w_{C3}x_3 \end{pmatrix} = \begin{pmatrix} \bar{y}_A \\ \bar{y}_B \\ \bar{y}_C \end{pmatrix}$

x1	x2	y
10	5	A
9	5	A
3	2	B
2	4	B
11	1	C



Cost Function은?



- **Cost Function은?**
  - Cross-Entropy

$$D(S, L) = -\sum_i L_i \log(S_i)$$

$$= \sum_i L_i * -\log(S_i)$$ # elementary production

  - **왜?**
    - 예측된 값 Li이에 대해,
      - **옳은 결과 : cost를 0에 가깝게**
      - **틀린 결과 : cost를 무한대에 가깝게**

$$cost(W) = \frac{1}{m}\sum_i D(S(WX_i + b), L_i)$$

Softmax Classification(Tensorflow구현)

(1) Graph 구성(가설, 비용함수, 학습함수 정의)

- Hypothesis(가설) 정의
  - 여기에 사용되는 변수(Variable)생성
- Cost/Loss함수 정의
- Train함수 정의

(2) Training…

- 충분한 만큼 반복하면서…
  - Graph상의 Tensor들을 실행
    - Train 함수 실행
    - Loss 함수 실행
  - 결과를 출력(또는, Tensorboard로 확인)

(3) Testing…

Graph 구성

- 데이터 준비

```
xy_data = np.array([
    [1, 2, 1, 1, 2],
    [2, 1, 3, 2, 2],
    [3, 1, 3, 4, 2],
    [4, 1, 5, 5, 1],
    [1, 7, 5, 5, 1],
    [1, 2, 5, 6, 1],
    [1, 6, 6, 6, 0],
    [1, 7, 7, 7, 0]
])

x_data = xy_data[:, 0:-1]   # 0 ~ 맨마지막 앞열까지 slicing
y_data = xy_data[:, -1]     # 맨마지막 열만....
nb_classes = 3 # 0 ~ 2
print('x_data=\n', x_data)
print('y_data=\n', y_data)

Y_one_hot = tf.one_hot(y_data, nb_classes).numpy()
print('Y_one_hot=\n', Y_one_hot)
```

```
x_data=
 [[1 2 1 1]
 [2 1 3 2]
 [3 1 3 4]
 [4 1 5 5]
 [1 7 5 5]
 [1 2 5 6]
 [1 6 6 6]
 [1 7 7 7]]
y_data=
 [2 2 2 1 1 1 0 0]
Y_one_hot=
 [[0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [1. 0. 0.]
 [1. 0. 0.]]
```

- one-hot 기법
  - 클래스의 label이 0 ~ n-1 사이의 값으로 되어 있는 경우
  - 해당 클래스의 위치에 1, 나머지에 0을 부여
  - (예)

```
[[1, 2, 1, 1, 2],
 [2, 1, 3, 2, 2],
 [3, 1, 3, 4, 2],
 [4, 1, 5, 5, 1],
 [1, 7, 5, 5, 1],
 [1, 2, 5, 6, 1],
 [1, 6, 6, 6, 0],
 [1, 7, 7, 7, 0]]
```

```
[[1, 2, 1, 1, 0, 0, 1],
 [2, 1, 3, 2, 0, 0, 1],
 [3, 1, 3, 4, 0, 0, 1],
 [4, 1, 5, 5, 0, 1, 0],
 [1, 7, 5, 5, 0, 1, 0],
 [1, 2, 5, 6, 0, 1, 0],
 [1, 6, 6, 6, 1, 0, 0],
 [1, 7, 7, 7, 1, 0, 0]]
```

```
Y_one_hot = tf.one_hot(y_data, nb_classes).numpy()
print('Y_one_hot=\n', Y_one_hot)
```

Hypothesis(가설) 정의

- $H(X) = softmax(WX + b)$

```
W = tf.Variable(tf.random.normal([4, nb_classes]), name='weight')
B = tf.Variable(tf.random.normal([nb_classes]), name='bias')

# define hypothesis
@tf.function
def Hypothesis(X):
    logits = tf.add(tf.matmul(tf.cast(X, tf.float32), W), B)
    return tf.nn.softmax(logits)
```

Cost/Loss함수 정의

- $cost(W) = \frac{1}{m}\sum_i D(S(WX_i + b), L_i)$

```
# define cost function
@tf.function
def loss(H, Y):
    entropy = -tf.reduce_sum(Y * tf.math.log(H), axis=1)
    # entropy = tf.losses.categorical_crossentropy(Y, H)
    cost = tf.reduce_mean(entropy)
    return cost
```

train 함수 정의

```
# minimize the cost function
@tf.function
def train(X, Y, learning_rate=0.1):
    with tf.GradientTape() as tape:
        _loss = loss(Hypothesis(X), Y)
    _w, _b = tape.gradient(_loss, [W, B])
    W.assign_sub(learning_rate * _w)
    B.assign_sub(learning_rate * _b)
```

evaluation 함수 정의

```
# accuracy computation
@tf.function
def evaluation(H, Y):
    prediction = tf.argmax(H, 1)
    correct_prediction = tf.equal(prediction, tf.argmax(Y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    return prediction, accuracy
```

Training

```
# training...
for step in range(2001):
    _c = loss(Hypothesis(x_data), Y_one_hot)
    train(x_data, Y_one_hot, learning_rate=0.1)
    if step % 100 == 0:
        print(f"step:{step}\tloss:{_c.numpy()}")
```

```
step:0          loss:12.715958595275879
step:100        loss:0.4636692702770233
step:200        loss:0.4191420376300812
step:300        loss:0.3811357021331787
step:400        loss:0.34504932165145874
step:500        loss:0.30915528535842896
...
step:1500       loss:0.16512851417064667
step:1600       loss:0.15909215807914734
step:1700       loss:0.1534709334373474
step:1800       loss:0.14822357892990112
step:1900       loss:0.1433146595954895
step:2000       loss:0.13871297240257263
```

Testing

```
# report accuracy...
print("\nAccuracy...")
_h = Hypothesis(x_data)
_p, _a = evaluation(_h, Y_one_hot)
print("Hypothesis =\n", _h.numpy())
print("Predicted =\n", _p.numpy())
print("\nAccuracy =", _a.numpy())

# testing...
test_data = np.array([
    [1, 2, 1, 1],
    [2, 1, 3, 2],
    [3, 1, 3, 4],
    [4, 1, 5, 5],
    [1, 7, 5, 5],
    [1, 2, 5, 6],
    [1, 6, 6, 6],
    [1, 7, 7, 7]
])
print("\ntesting...")
for data in test_data:
    result = tf.argmax(Hypothesis([data]), 1)
    print(f"input: {data}\t output: {result}")
```

```
Accuracy...
Hypothesis =
 [[3.3224935e-06 1.0711577e-03 9.9892551e-01]
 [1.9420490e-03 6.9983177e-02 9.2807478e-01]
 [9.2217753e-09 1.4396164e-01 8.5603833e-01]
 [2.8240305e-07 8.6942077e-01 1.3057895e-01]
 [2.3215483e-01 7.5728875e-01 1.0556406e-02]
 [1.2213068e-01 8.7769377e-01 1.7556392e-04]
 [7.7613556e-01 2.2381115e-01 5.3263961e-05]
 [9.2647862e-01 7.3520422e-02 9.4238942e-07]]
Predicted =
 [2 2 2 1 1 1 0 0]

Accuracy = 1.0

testing...
input: [1 2 1 1]         output: [2]
input: [2 1 3 2]         output: [2]
input: [3 1 3 4]         output: [2]
input: [4 1 5 5]         output: [1]
input: [1 7 5 5]         output: [1]
input: [1 2 5 6]         output: [1]
input: [1 6 6 6]         output: [0]
input: [1 7 7 7]         output: [0]
```

전체코드

```
import tensorflow as tf
import numpy as np

xy_data = np.array([
    [1, 2, 1, 1, 2],    [2, 1, 3, 2, 2],    [3, 1, 3, 4, 2],    [4, 1, 5, 5, 1],
    [1, 7, 5, 5, 1],    [1, 2, 5, 6, 1],    [1, 6, 6, 6, 0],    [1, 7, 7, 7, 0]
])

x_data = xy_data[:, 0:-1]   # 0 ~ 맨마지막 앞열까지 slicing
y_data = xy_data[:, -1]     # 맨마지막 열만....
nb_classes = 3 # 0 ~ 2
print('x_data=\n', x_data)
print('y_data=\n', y_data)

Y_one_hot = tf.one_hot(y_data, nb_classes).numpy()
print('Y_one_hot=\n', Y_one_hot)

W = tf.Variable(tf.random.normal([4, nb_classes]), name='weight')
B = tf.Variable(tf.random.normal([nb_classes]), name='bias')

# define hypothesis
@tf.function
def Hypothesis(X):
    logits = tf.add(tf.matmul(tf.cast(X, tf.float32), W), B)
    return tf.nn.softmax(logits)

# define cost function
@tf.function
def loss(H, Y):
    entropy = -tf.reduce_sum(Y * tf.math.log(H), axis=1)
    # entropy = tf.losses.categorical_crossentropy(Y, H)
    cost = tf.reduce_mean(entropy)
    return cost
```

전체코드

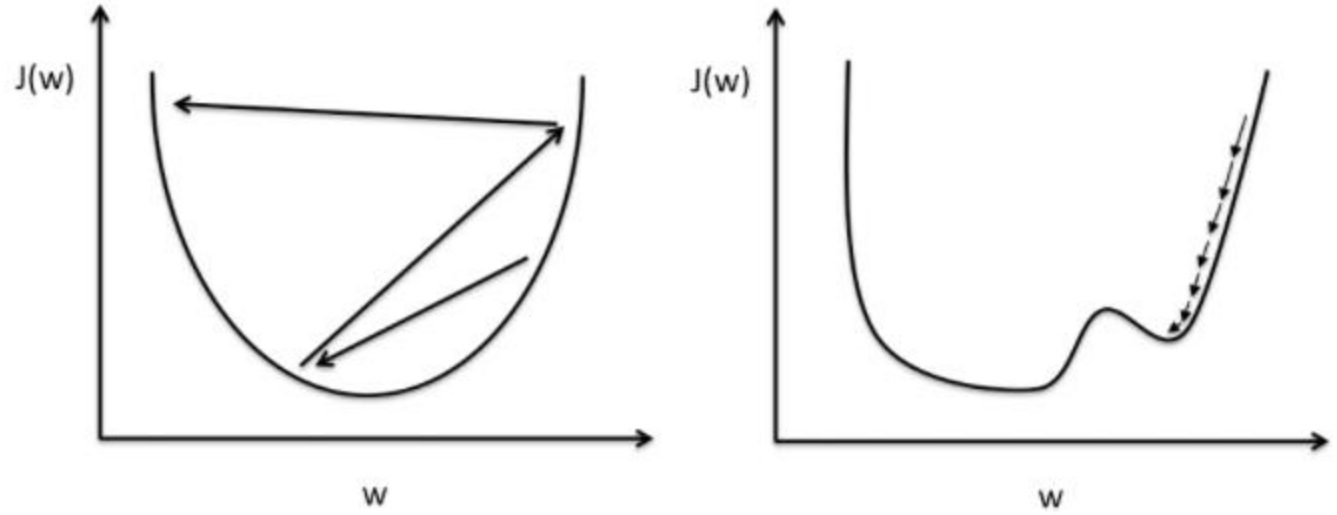
```
# minimize the cost function
@tf.function
def train(X, Y, learning_rate=0.1):
    with tf.GradientTape() as tape:
        _loss = loss(Hypothesis(X), Y)
    _w, _b = tape.gradient(_loss, [W, B])
    W.assign_sub(learning_rate * _w)
    B.assign_sub(learning_rate * _b)

# accuracy computation
@tf.function
def evaluation(H, Y):
    prediction = tf.argmax(H, 1)
    correct_prediction = tf.equal(prediction, tf.argmax(Y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    return prediction, accuracy

# training...
for step in range(2001):
    _c = loss(Hypothesis(x_data), Y_one_hot)
    train(x_data, Y_one_hot, learning_rate=0.1)
    if step % 100 == 0:
        print(f"step:{step}\tloss:{_c.numpy()}")
```

전체코드

```
# report accuracy...
print("\nAccuracy...")
_h = Hypothesis(x_data)
_p, _a = evaluation(_h, Y_one_hot)
print("Hypothesis =\n", _h.numpy())
print("Predicted =\n", _p.numpy())
print("\nAccuracy =", _a.numpy())

# testing...
test_data = np.array([
    [1, 2, 1, 1],
    [2, 1, 3, 2],
    [3, 1, 3, 4],
    [4, 1, 5, 5],
    [1, 7, 5, 5],
    [1, 2, 5, 6],
    [1, 6, 6, 6],
    [1, 7, 7, 7]
])
print("\ntesting...")
for data in test_data:
    result = tf.argmax(Hypothesis([data]), 1)
    print(f"input: {data}\t output: {result}")
```

Learning rate문제

- 너무 크거나 작으면 overshooting이 발생하거나 수렴(convergence)이 되지 않는 문제



- 해결방법
  - 0.01정도부터 시작해서 여러 번 값을 바꾸어가면서 시도(cost의 변화를 체크)
  - Tensorboard를 활용하는 것이 좋음

NaN / inf 발생 문제

- 입력 데이터가 정규화(normalization)되지 않아서 발생

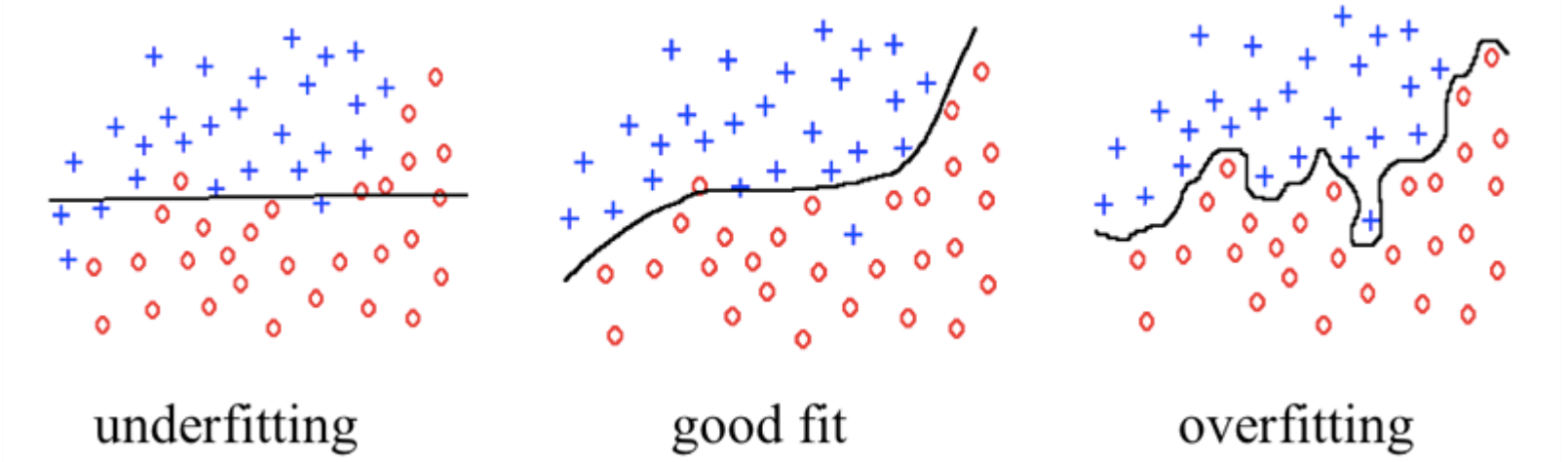
```
xy = np.array([[828.659973, 833.450012, 908100, 828.349976, 831.659973],
               [823.02002, 828.070007, 1828100, 821.655029, 828.070007],
               [819.929993, 824.400024, 1438100, 818.97998, 824.159973],
               [816, 820.958984, 1008100, 815.48999, 819.23999],
               [819.359985, 823, 1188100, 818.469971, 818.97998],
               [819, 823, 1198100, 816, 820.450012],
               [811.700012, 815.25, 1098100, 809.780029, 813.669983],
               [809.51001, 816.659973, 1398100, 804.539978, 809.559998]])
```

```
[[ 0.99999999  0.99999999  0.        1.        1.      ]
 [ 0.70548491  0.70439552  1.        0.71881782  0.83755791]
 [ 0.54412549  0.50274824  0.57608696  0.606468    0.6606331 ]
 [ 0.33890353  0.31368023  0.10869565  0.45989134  0.43800918]
 [ 0.51436     0.42582389  0.30434783 0.58504805  0.42624401]
 [ 0.49556179  0.42582389  0.31521739  0.48131134  0.49276137]
 [ 0.11436064  0.        0.20652174  0.22007776  0.18597238]
 [ 0.          0.07747099  0.5326087   0.        0.      ]]
```

- 해결방법
  - MinMax스케일러로 정규화
    - xy = MinMaxScaler(xy)

Overfitting 문제

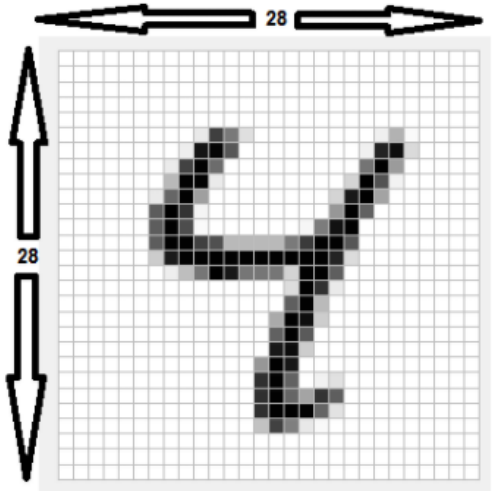
- Generalization(일반화)



- 해결방안
  - 많은 학습데이터를 확보
  - 특징의 개수를 축소
  - Regularization
    - weight가 너무 크지 않도록 조정

MNIST Dataset

- 우편번호 필기숫자 데이터베이스
  - 학습데이터 60,000개 / 테스트데이터 10,000개
  - 4개의 파일로 구성
    - train-images-idx3-ubyte.gz # training images
    - train-labels-idx1-ubyte.gz # training set labels
    - t10k-images-idx3-ubyte.gz # test images
    - t10k-labels-idx1-ubyte.gz # test set labels



참고 : epoch, batch_size

- 데이터의 크기가 매우 크므로, 데이터를 몇개로 나누어서 학습시킴
  - epoch
    - 전체 Dataset을 한 번 학습시키는 것을 1 epoch이라고 함(즉, 몇 번 반복해서 학습시킬지를 의미)
  - batch size
    - 한 번에 메모리에서 처리하는 양
  - number of iterations
    - = number of training samples / batch_size
    - (예) 전체 sample이 20,000개인 경우,
      이것을 100개로 나누어 학습시키려면
      batch_size는 200이 되어야 함

Softmax Classification(MNIST Dataset)

- **MNIST data reading**

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

x_train = x_train.reshape(60000, 784).astype('float32')
x_test = x_test.reshape(10000, 784).astype('float32')

nb_classes = 10

y_train_one_hot = tf.one_hot(y_train, nb_classes).numpy()
y_test_one_hot = tf.one_hot(y_test, nb_classes).numpy()
```

- **hypothesis정의** $H(X) = softmax(WX + b)$

```
W = tf.Variable(tf.random.normal([784, nb_classes]), name='weight')
B = tf.Variable(tf.random.normal([nb_classes]), name='bias')

# define hypothesis
@tf.function
def Hypothesis(X):
    logits = tf.add(tf.matmul(tf.cast(X, tf.float32), W), B)
    return tf.nn.softmax(logits)
```

- **Cost함수 정의** $cost(W) = \frac{1}{m}\sum_i D(S(WX_i + b), L_i)$

```
# define cost function
@tf.function
def loss(H, Y):
    entropy = -tf.reduce_sum(Y * tf.math.log(H), axis=1)
    # entropy = tf.losses.categorical_crossentropy(Y, H)
    cost = tf.reduce_mean(entropy)
    return cost
```

Train 및 Evaluation함수 정의

```
# minimize the cost function
@tf.function
def train(X, Y, learning_rate=0.1):
    with tf.GradientTape() as tape:
        _loss = loss(Hypothesis(X), Y)
    _w, _b = tape.gradient(_loss, [W, B])
    W.assign_sub(learning_rate * _w)
    B.assign_sub(learning_rate * _b)

# accuracy computation
@tf.function
def evauation(H, Y):
    prediction = tf.argmax(H, 1)
    correct_prediction = tf.equal(prediction, tf.argmax(Y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    return prediction, accuracy
```

Training

```
Epoch:  1 loss: 2.2719297409057617
Epoch:  2 loss: 0.8776038885116577
Epoch:  3 loss: 0.7150184512138367
...
Epoch: 49 loss: 0.2938329577445984
Epoch: 50 loss: 0.2925865054130554

Accuracy...

Accuracy = 0.9113
```

```python
# training...
# parameters
training_epochs = 50
batch_size = 100

for epoch in range(training_epochs):
    avg_cost = 0
    iterations = int(len(x_train) / batch_size) # iteration의 크기 구하기

    idx = 0
    for i in range(iterations):
        batch_xs, batch_ys = x_train[idx:idx+batch_size, :],
                             y_train_one_hot[idx:idx+batch_size, :]
        _c = loss(Hypothesis(batch_xs), batch_ys)
        train(batch_xs, batch_ys, learning_rate=0.15)
        avg_cost += _c / iterations
        idx += batch_size
    print("Epoch: {:2d} loss: {}".format(epoch+1, avg_cost))

# report accuracy...
print("\nAccuracy...")
_h = Hypothesis(x_test)
_p, _a = evauation(_h, y_test_one_hot)
print("\nAccuracy =", _a.numpy())
```

Keras를 이용해보자!

- Keras는 신경망을 구성하기 위한 각 구성요소를 클래스로 제공
- tf.keras는 tensorflow의 high-level API

- 주요특징
  - 모듈화 (Modularity)
    - 케라스에서 제공하는 모듈은 독립적(신경망 층, 비용함수, 옵티마이저, 초기화 기법, 활성화함수, 정규화기법 등)
  - 최소주의 (Minimalism)
    - 각 모듈은 짧고 간결하며, 이해하기 쉬움
  - 쉬운 확장성
    - 새로운 클래스나 함수로 모듈을 아주 쉽게 추가할 수 있음
  - 파이썬 기반
    - Caffe 처럼 별도의 모델 설정 파일이 필요없으며 파이썬 코드로 모델들이 정의됨

Keras 사용의 일반적인 절차

(1) model 구성
- Keras의 Sequential() 클래스로 model 객체 생성
- model에 필요한 layer를 추가

(2) model.compile() : 모델의 학습과정 설정
- optimizer와 loss함수 설정

(3) model.fit() : 학습
- batch_size, epochs 등을 설정
- loss, accuracy 측정

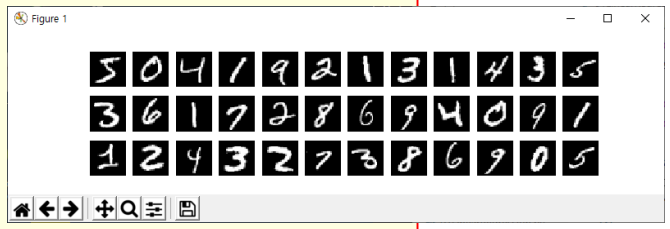
(4) model.evaluate() : 성능평가
- 준비된 test dataset으로 학습한 모델 평가

(5) model.predict() : 모델사용
- 임의의 입력 데이터에 대한 model의 예측결과 얻기

Softmax Classification(MNIST Dataset)

MNIST data reading

```
mnist = tf.keras.datasets.mnist
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0

plt.figure(figsize=(8, 2))  # 8 x 2 inchs
for i in range(36):
    plt.subplot(3, 12, i+1)
    plt.imshow(X_train[i], cmap="gray")
    plt.axis("off")
plt.show()

print(X_train.shape, X_train.dtype)
print(Y_train.shape, Y_train.dtype)
print(X_test.shape, X_test.dtype)
print(Y_test.shape, Y_test.dtype)
```




```
(60000, 28, 28) float64
(60000,) uint8
(10000, 28, 28) float64
(10000,) uint8
```

define model / compile / fit / evaluate

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(10, input_dim=784, activation='softmax')
])
```

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=["accuracy"])
model.summary()
```

```
hist = model.fit(X_train, Y_train,
                 validation_data=(X_test, Y_test),
                 verbose=2, batch_size=100, epochs=15,
                 use_multiprocessing=True)
```

```
model.evaluate(X_test, Y_test,
               verbose=2, batch_size=100, use_multiprocessing=True)
```

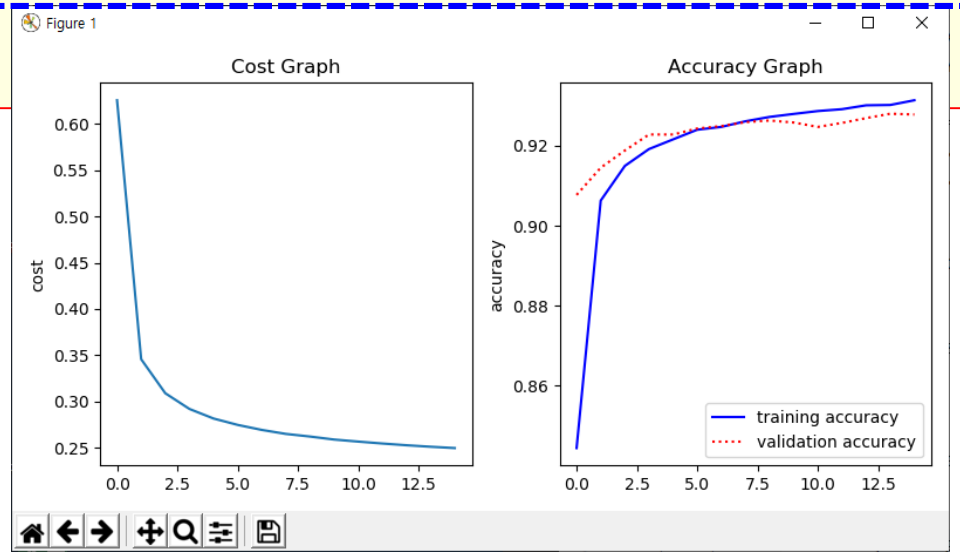
verbose
- 0 : silent
- 1 : progress bar
- 2 : one line per epoch

학습과정 및 성능평가 결과

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 10)                7850
=================================================================
Total params: 7,850
Trainable params: 7,850
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/15
60000/60000 - 1s - loss: 0.6258 - accuracy: 0.8444 - val_loss: 0.3618 - val_accuracy: 0.9077
Epoch 2/15
60000/60000 - 1s - loss: 0.3457 - accuracy: 0.9063 - val_loss: 0.3105 - val_accuracy: 0.9145
Epoch 3/15
60000/60000 - 1s - loss: 0.3089 - accuracy: 0.9150 - val_loss: 0.2905 - val_accuracy: 0.9188

...

Epoch 14/15
60000/60000 - 1s - loss: 0.2510 - accuracy: 0.9302 - val_loss: 0.2620 - val_accuracy: 0.9280
Epoch 15/15
60000/60000 - 1s - loss: 0.2496 - accuracy: 0.9314 - val_loss: 0.2626 - val_accuracy: 0.9278
10000/1 - 0s - loss: 0.3112 - accuracy: 0.9278
```

model의 학습과정 살펴보기

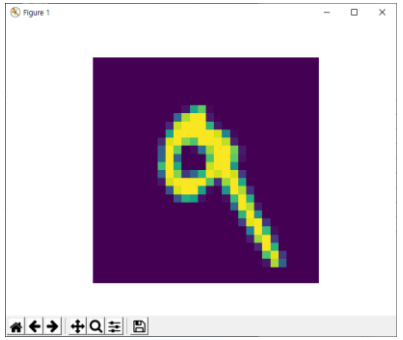
```
plt.figure(figsize=(8, 4))  # 8 x 4 inchs
plt.subplot(1, 2, 1)
plt.plot(hist.history['loss'])
plt.title("Cost Graph")
plt.ylabel("cost")
plt.subplot(1, 2, 2)
plt.title("Accuracy Graph")
plt.ylabel("accuracy")
plt.plot(hist.history['accuracy'], 'b-', label="training accuracy")
plt.plot(hist.history['val_accuracy'], 'r:', label="validation accuracy")
plt.legend()
plt.tight_layout()
plt.show()
```



model 사용하기

```
prediction = model.predict(X_test[:1, :])
prediction_class = tf.argmax(prediction, 1)
print(f"\nPrediction Result:\n{prediction}")
print("Predicted class: ", prediction_class.numpy())

plt.imshow(X_test[prediction_class[0]])
plt.grid(False)
plt.axis("off")
plt.show()
```

```
Prediction Result:
[[1.9937934e-06 9.0225865e-12 4.4221838e-06 4.0960643e-03 3.3323923e-07
  2.1545377e-05 7.7309860e-11 9.9556160e-01 1.7959275e-05 2.9616323e-04]]
Predicted class:  [7]
```



전체코드

```
import tensorflow as tf
import matplotlib.pyplot as plt

mnist = tf.keras.datasets.mnist
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0

plt.figure(figsize=(8, 2))  # 8 x 2 inchs
for i in range(36):
    plt.subplot(3, 12, i+1)
    plt.imshow(X_train[i], cmap="gray")
    plt.axis("off")
plt.show()

print(X_train.shape, X_train.dtype)
print(Y_train.shape, Y_train.dtype)
print(X_test.shape, X_test.dtype)
print(Y_test.shape, Y_test.dtype)

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(10, input_dim=784, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=["accuracy"])
model.summary()
```

continue...

```
hist = model.fit(X_train, Y_train,
                 validation_data=(X_test, Y_test),
                 verbose=2, batch_size=100, epochs=15, use_multiprocessing=True)
model.evaluate(X_test, Y_test,
               verbose=2, batch_size=100, use_multiprocessing=True)

plt.figure(figsize=(8, 4))  # 8 x 4 inchs
plt.subplot(1, 2, 1)
plt.plot(hist.history['loss'])
plt.title("Cost Graph")
plt.ylabel("cost")
plt.subplot(1, 2, 2)
plt.title("Accuracy Graph")
plt.ylabel("accuracy")
plt.plot(hist.history['accuracy'], 'b-', label="training accuracy")
plt.plot(hist.history['val_accuracy'], 'r:', label="validation accuracy")
plt.legend()
plt.tight_layout()
plt.show()

prediction = model.predict(X_test[:1, :])
prediction_class = tf.argmax(prediction, 1)
print(f"\nPrediction Result:\n{prediction}")
print("Predicted class: ", prediction_class.numpy())
plt.imshow(X_test[prediction_class[0]])
plt.grid(False)
plt.axis("off")
plt.show()
```

Save / Load of a Model or Weights

모델의 저장 및 복원

```
# 전체 모델을 HDF5 파일로 저장합니다
model.save('my_model.h5')
# 가중치와 옵티마이저를 포함하여 정확히 동일한 모델을 다시 생성합니다
new_model = tf.keras.models.load_model('my_model.h5')
new_model.summary()
```

가중치의 저장 및 복원

```
# 가중치를 저장합니다
model.save_weights('./checkpoints/my_checkpoint')

# 가중치를 복원합니다
model = create_model() # model 생성 함수 정의
model.load_weights('./checkpoints/my_checkpoint')
```

Save Model

```
import tensorflow as tf
import matplotlib.pyplot as plt

### 생략 ###
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(10, input_dim=784, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=["accuracy"])

hist = model.fit(X_train, Y_train,
             validation_data=(X_test, Y_test),
             verbose=2, batch_size=100, epochs=15, use_multiprocessing=True)

model.evaluate(X_test, Y_test,
               verbose=2, batch_size=100, use_multiprocessing=True)

# 전체 모델을 HDF5 파일로 저장합니다
file_name = "softmax_mnist_model.h5"
model.save(file_name)
print(f"\nThis model has been saved to {file_name}.")
```

Restore Model

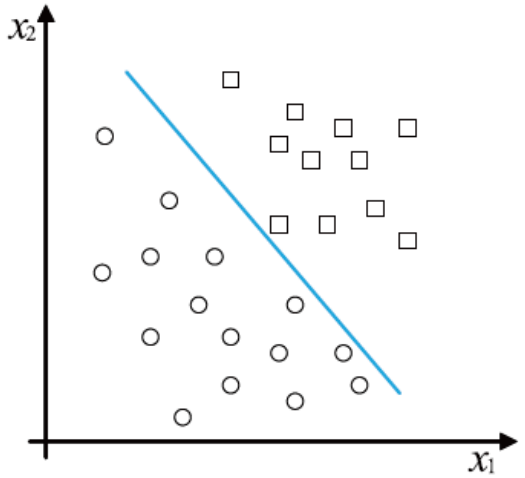
```
import tensorflow as tf
import matplotlib.pyplot as plt

### 생략 ###

# 가중치와 옵티마이저를 포함하여 정확히 동일한 전체 모델을 HDF5 파일로 부터 로딩
file_name = "softmax_mnist_model.h5"
model = tf.keras.models.load_model(file_name)
print(f"\nThis model has been loaded from {file_name}.")

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=["accuracy"])

hist = model.fit(X_train, Y_train,
              validation_data=(X_test, Y_test),
              verbose=2, batch_size=100, epochs=15, use_multiprocessing=True)

model.evaluate(X_test, Y_test,
               verbose=2, batch_size=100, use_multiprocessing=True)
```

(3) NN for XOR

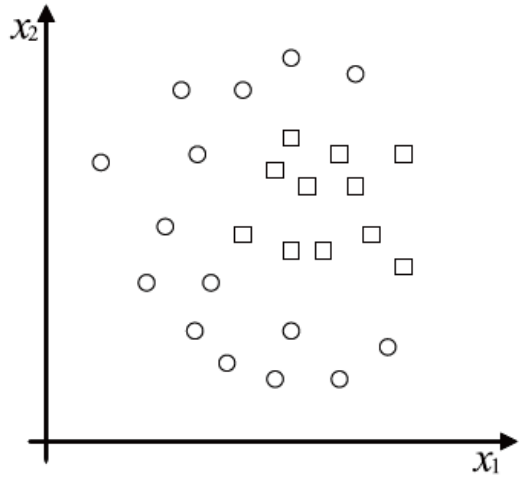
퍼셉트론의 한계

- 한계
  - 퍼셉트론은 결정초평면 역할을 하는 선형분류기
  - 선형 분리 불가능한 상황에 대처하지 못함

$$d(\mathbf{x}) = \mathbf{w}\mathbf{x}^{\mathrm{T}} + b \geq 0 \text{이면 } \mathbf{x} \in \omega_1$$
$$d(\mathbf{x}) = \mathbf{w}\mathbf{x}^{\mathrm{T}} + b < 0 \text{이면 } \mathbf{x} \in \omega_2$$



(a) 선형 분리 가능

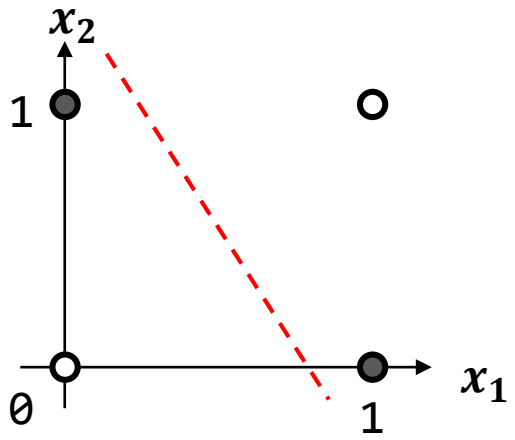


(b) 선형 분리 불가능

단순한 XOR 분류 문제

- XOR

x_1	x_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

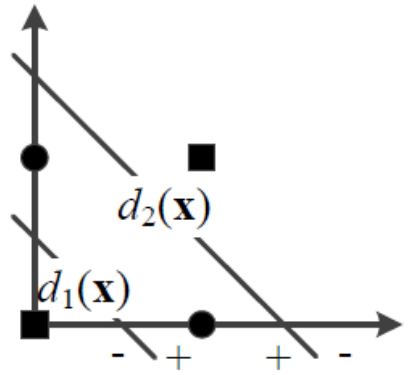


- Logistic Regression?
  - 해결 불가능

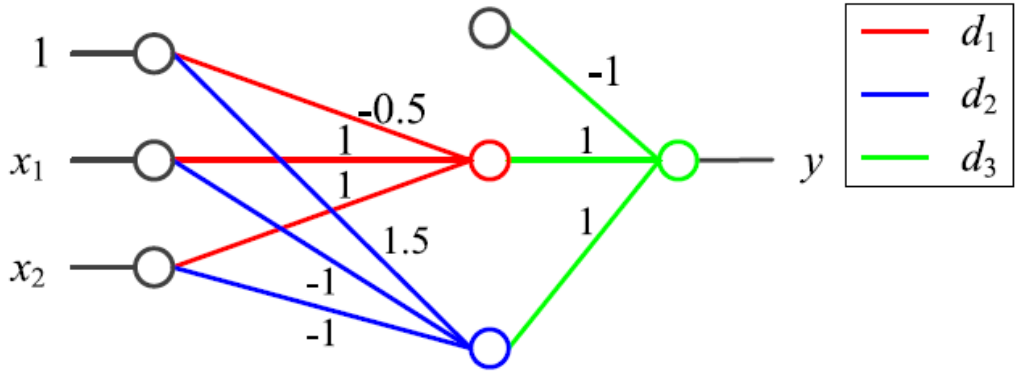
- 해결방법
  - Deep Learning....

다층 퍼셉트론(MLP)으로 확장

- XOR 분류 문제
  - 단일 Perceptron으로는 분류 불가능(75%한계)
  - 3개의 Perceptron으로 해결
    - d_1의 +영역과 d_2의 +영역이 겹친 영역은 ω_1, 나머지는 ω_2

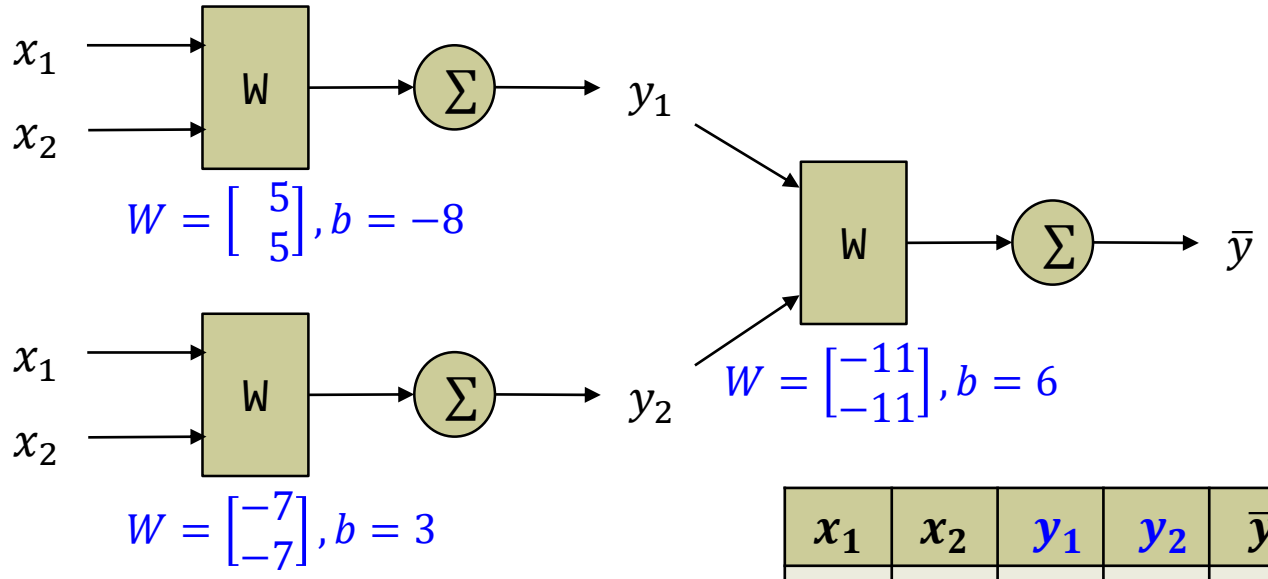


(a) XOR 분류 문제



(b) 세 개 퍼셉트론의 결합

실제 계산의 예



x_1	x_2	y_1	y_2	$\bar{y}$	XOR
0	0	0	1	0	0
0	1	0	0	1	1
1	0	0	0	1	1
1	1	1	0	0	0

$x_1 = 0, \quad x_2 = 0$

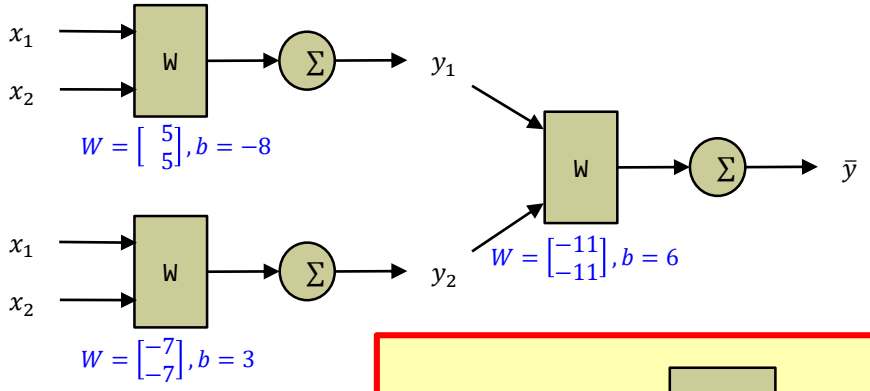
$$[0\ 0]\begin{bmatrix} 5 \\ 5 \end{bmatrix} - 8 = -8, y_1 = S(-8) = 0$$

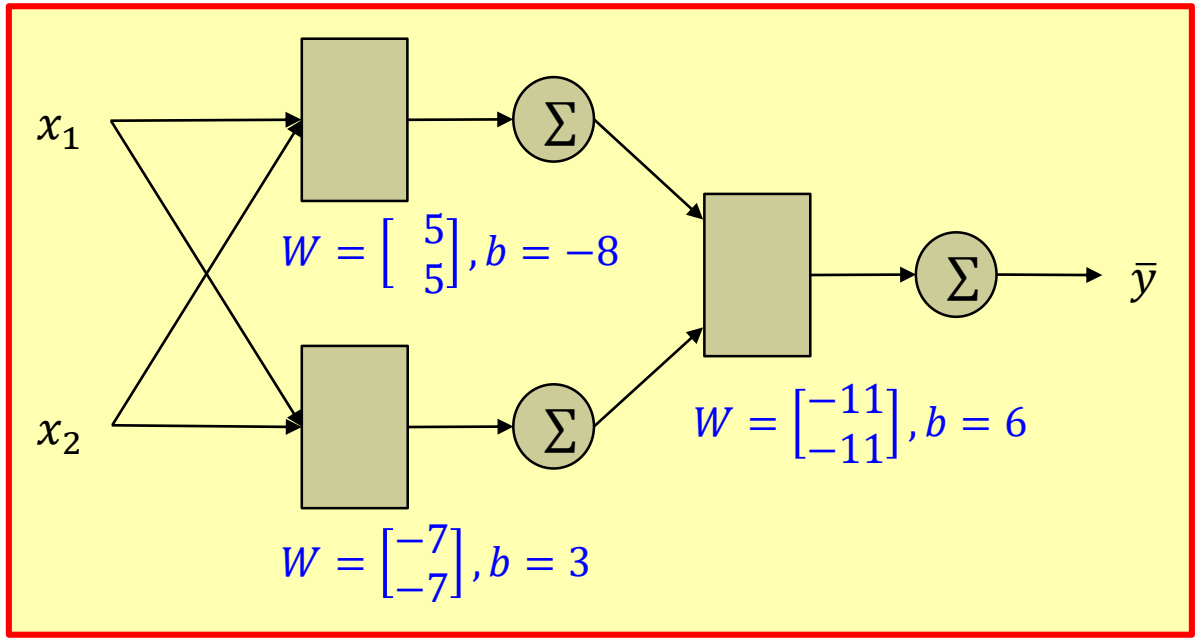
$$[0\ 0]\begin{bmatrix} -7 \\ -7 \end{bmatrix} + 3 = 3, y_2 = S(3) = 1$$

$$[0\ 1]\begin{bmatrix} -11 \\ -11 \end{bmatrix} + 6 = -5, \bar{y} = S(-5) = 0$$

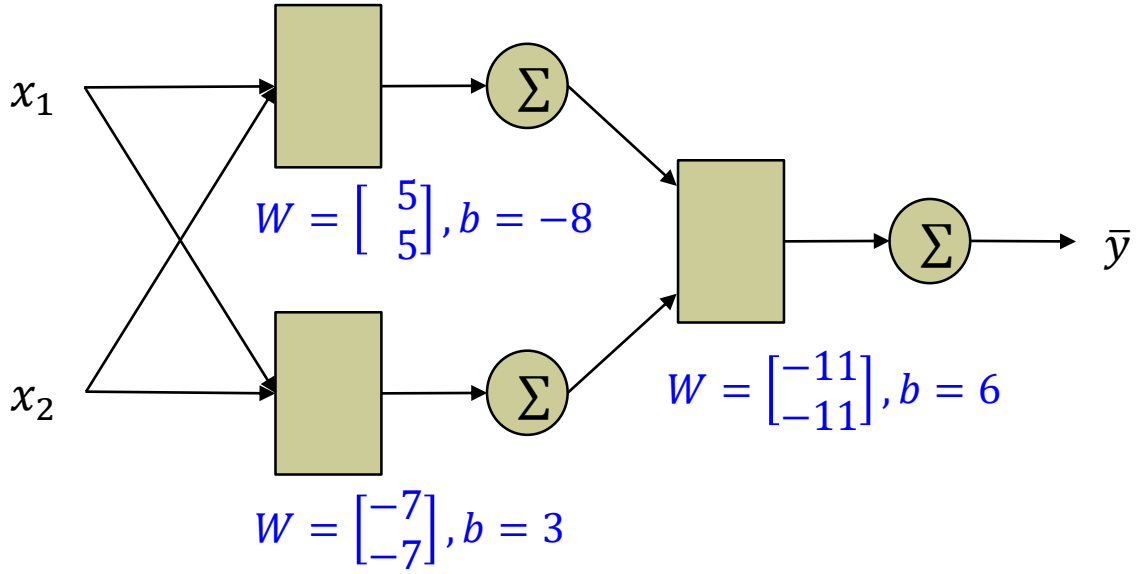
Forward Propagation

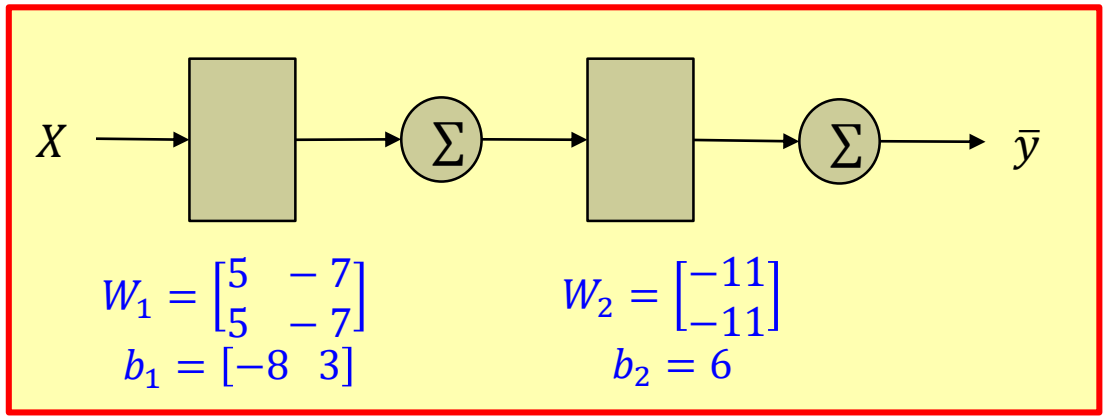
- Neural Network의 재구성





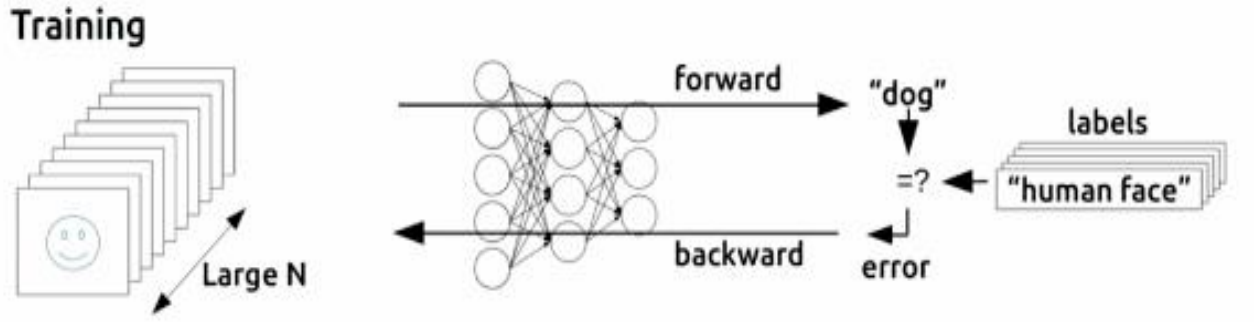
Neural Network의 재구성





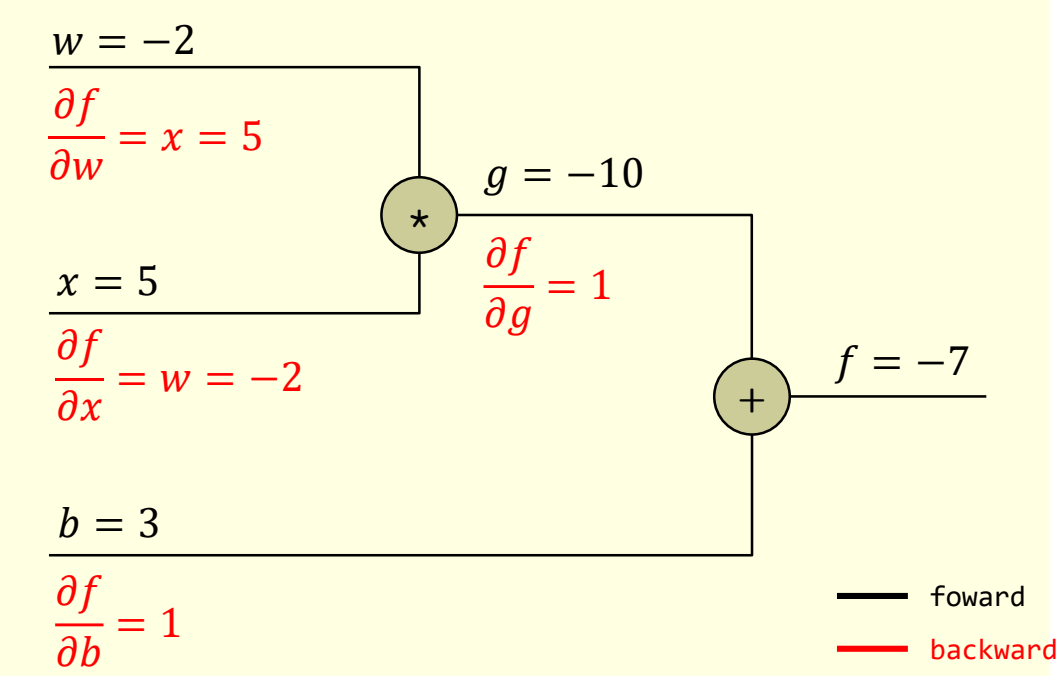
Backpropagation

- 그러면 어떻게 W와 b를 찾을 것인가?
  - gradient descent algorithm
    - 기울기를 구하기 위해 미분을 사용
    - 매우 복잡함
- Backpropagation
  - forward를 통해 나온 결과 값과 실제 데이터를 비교하여 이에 대한 차이를 뒤로 가면서 학습을 시키는 방법



chain rule

- **Backpropagation**
  - 미분공식인 chain rule을 사용
  - 함수 $f(g(x))$의 미분값 구하기
    - $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial x}$
    - (예) $f = Wx + b,\ g = Wx,$ 즉, $f = g + b$
      만일, $w = -2,\ x = 5,\ b = 3$ 이면,



미리 계산된 편미분

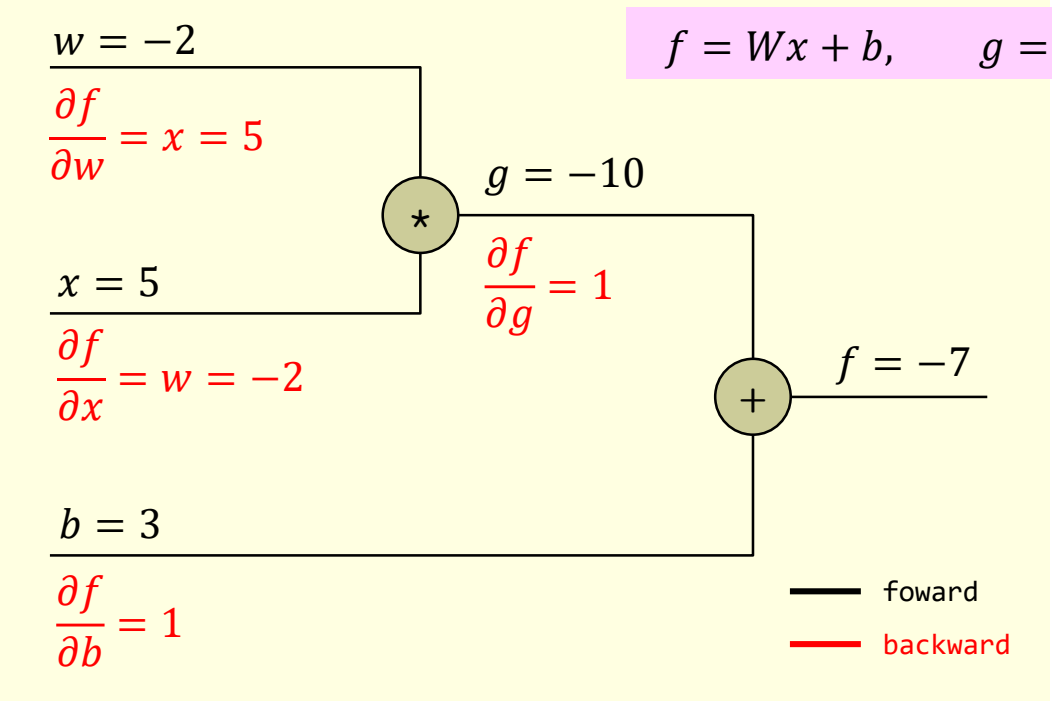
$\frac{\partial g}{\partial w} = x \quad \frac{\partial f}{\partial w} = x$

$\frac{\partial g}{\partial x} = w \quad \frac{\partial f}{\partial x} = w$

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

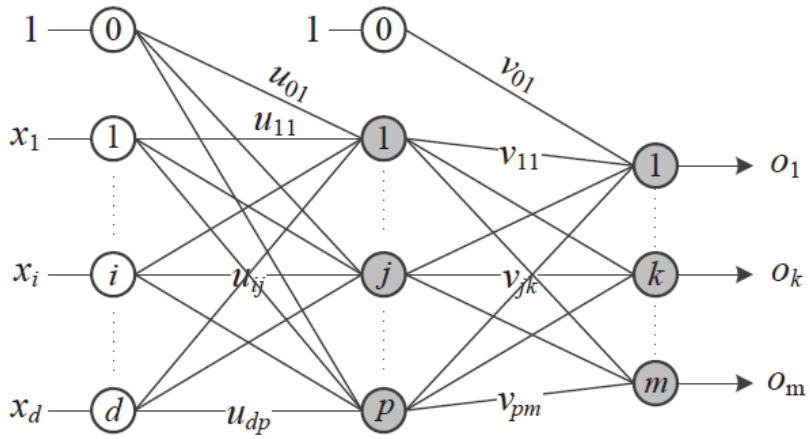
$\frac{\partial f}{\partial b} = g = 1$

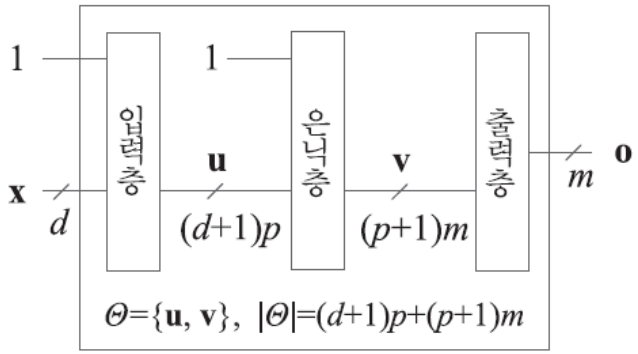
- 미분값이 계산되면,
  - f에 w는 5배, g는 1배, x는 -2배, b는 1배의 영향을 미쳤음을 알 수 있음
  - 이를 이용하여 앞에 있는 노드들의 weight조정



다층 퍼셉트론의 구조

- 입력층 → 은닉층 → 출력층
  - 입력층: 특징 벡터의 차원에 따라 d개의 노드와 여분의 바이어스 노드로 구성
  - 출력층: 부류 개수에 따라 m개의 노드로 구성
  - 은닉층: 노드 개수 p를 사용자가 설정





NN for XOR(Tensorflow 구현)

- Dataset 정의

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

# XOR data
x_data = tf.constant([[0, 0],
                      [0, 1],
                      [1, 0],
                      [1, 1]], dtype=tf.float32)
y_data = tf.constant([[0],
                      [1],
                      [1],
                      [0]], dtype=tf.float32)


print(x_data.shape, x_data.dtype)
print(y_data.shape, y_data.dtype)
```

```
(4, 2) <dtype: 'float32'>
(4, 1) <dtype: 'float32'>
```

Model 구성

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(8, input_dim=2, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 8)                 24
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 9
=================================================================
Total params: 33
Trainable params: 33
Non-trainable params: 0
```

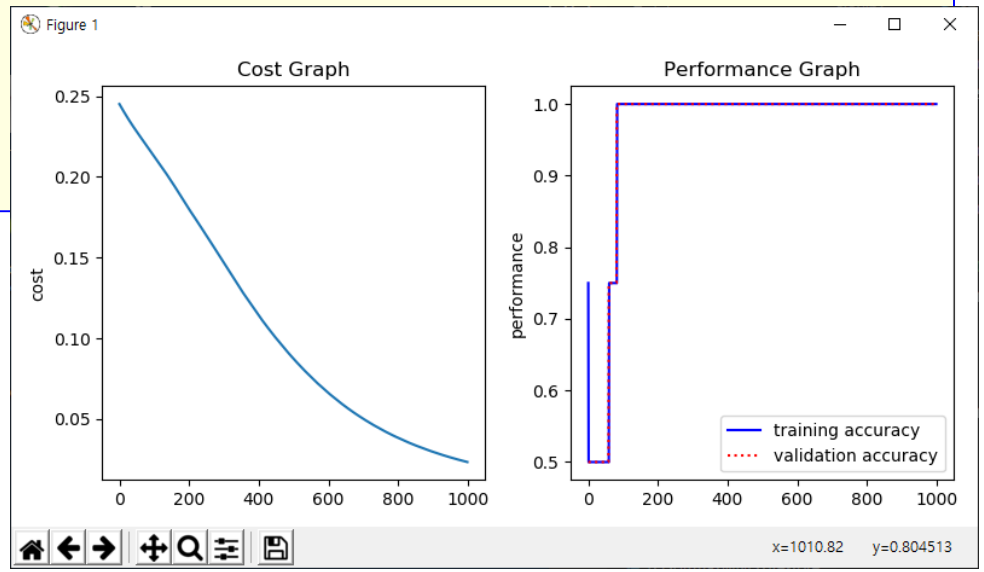
Model compile / fit / evaluate

```
model.compile(optimizer='adam',
              loss='mean_squared_error',
              metrics=["accuracy"])
model.summary()

hist = model.fit(x_data, y_data, batch_size=4,
                 epochs=100, validation_data=(x_data, y_data),
                 verbose=2, use_multiprocessing=True)

model.evaluate(x_data, y_data, verbose=2, use_multiprocessing=True)
```

```
Train on 4 samples, validate on 4 samples
Epoch 1/100
4/4 - 0s - loss: 0.2668 - accuracy: 0.7500 - val_loss: 0.2664 - val_accuracy: 0.5000
Epoch 2/100
4/4 - 0s - loss: 0.2664 - accuracy: 0.5000 - val_loss: 0.2659 - val_accuracy: 0.7500
...
Epoch 99/100
4/4 - 0s - loss: 0.2278 - accuracy: 0.7500 - val_loss: 0.2275 - val_accuracy: 0.7500
Epoch 100/100
4/4 - 0s - loss: 0.2275 - accuracy: 0.7500 - val_loss: 0.2273 - val_accuracy: 0.7500
4/1 - 0s - loss: 0.2273 - accuracy: 0.7500
```

Cost / Accuracy

```
plt.figure(figsize=(8, 4))  # 8 x 4 inchs
plt.subplot(1, 2, 1)
plt.plot(hist.history['loss'])
plt.title("Cost Graph")
plt.ylabel("cost")
plt.subplot(1, 2, 2)
plt.title("Performance Graph")
plt.ylabel("performance")
plt.plot(hist.history['accuracy'], 'b-', label="training accuracy")
plt.plot(hist.history['val_accuracy'], 'r:', label="validation accuracy")
plt.legend()
plt.tight_layout()
plt.show()
print()
```



Model predict

```
x = [[1, 0]]
prediction = model.predict(x)
prediction_class = tf.cast(prediction > 0.5, dtype=tf.float32)
print("Prediction Result of {}: {}, {}"\
                .format(x, prediction, prediction_class.numpy()))
```

```
Prediction Result of [[1, 0]]: [[0.5884579]], [[1.]]
```

참고: Keras에서 tensorboard 사용하기

(1) tensorboard callback 을 정의

(2) model의 fit()에 callback을 등록

```
### for Tensorboard  ##################################################
log_dir=r"c:\temp\log_ex"
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_f
req=1)


hist = model.fit(x_data, y_data, batch_size=4,
                 epochs=1000, validation_data=(x_data, y_data),
                 callbacks=[tensorboard_callback],
                 verbose=2, use_multiprocessing=True)
model.evaluate(x_data, y_data, verbose=2, use_multiprocessing=True)

# for Tensorboard, add to fit() method callbacks=[tensorboard_callback]
# After executed, press Alt+F12 keys and enter a below command :
#        tensorboard --logdir=c:\temp\log_ex
#######################################################################
```

Neural Network의 정확도 향상방법

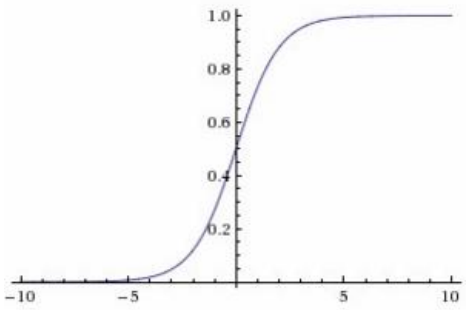
- **Activation Function**
  - ReLU ⇐ Sigmoid
  - Relu함수가 깊고 넓은 NN에 대해서는 효과적

- **Initalize weigths**
  - Xavier ⇐ Gradient descent algorithm
  - 많은 weight초기화 방법이 있으나, Xavier방법(2010년)을 많이 사용

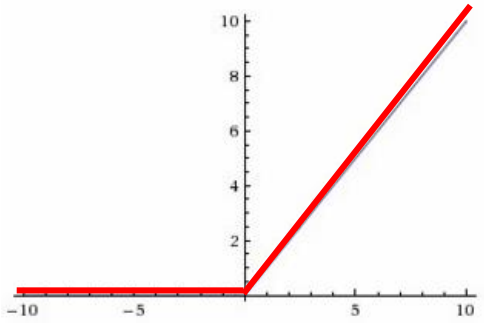
- **Solutions for Overfitting**
  - NN Dropout
  - Neuron을 랜덤하게 zero로 만드는 방법

ReLU

- **Deep & Wide Neural Network**
  - **넓고 깊게 학습하면 오히려 결과가 좋지 않다!?**
  - Backpropagation
    - 2 ~ 3 layer인 경우에는 문제가 없으나, 그 이상에서는 이상한 결과가 나온다
    - 이유 : Backpropagation 과정에서 Sigmoid 함수를 통과하기 때문에 계속 값들이 너무 작아진다... 즉, gradient가 vanishing(소멸)된다.
  - **해결방법 : Activation Function을 바꾸자**



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

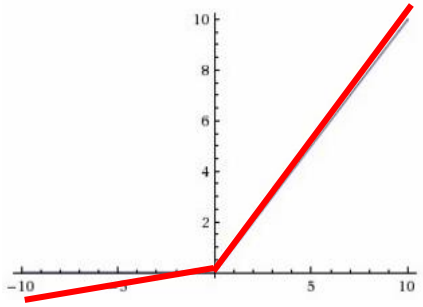


$$ReLU(x) = \max(0, x)$$

- **ReLU**
  - ~~layer1 = tf.simoid(tf.matmul(X,W1)+b1)~~
  - `layer1 = tf.nn.relu(tf.matmul(X,W1)+b1)`

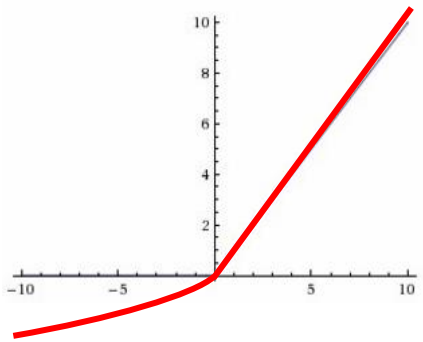
- **최근의 Activation Functions**
  - **Leaky ReLU**

$$LReLU(x) = \max(\alpha x, x)\,, \alpha = 0.1$$

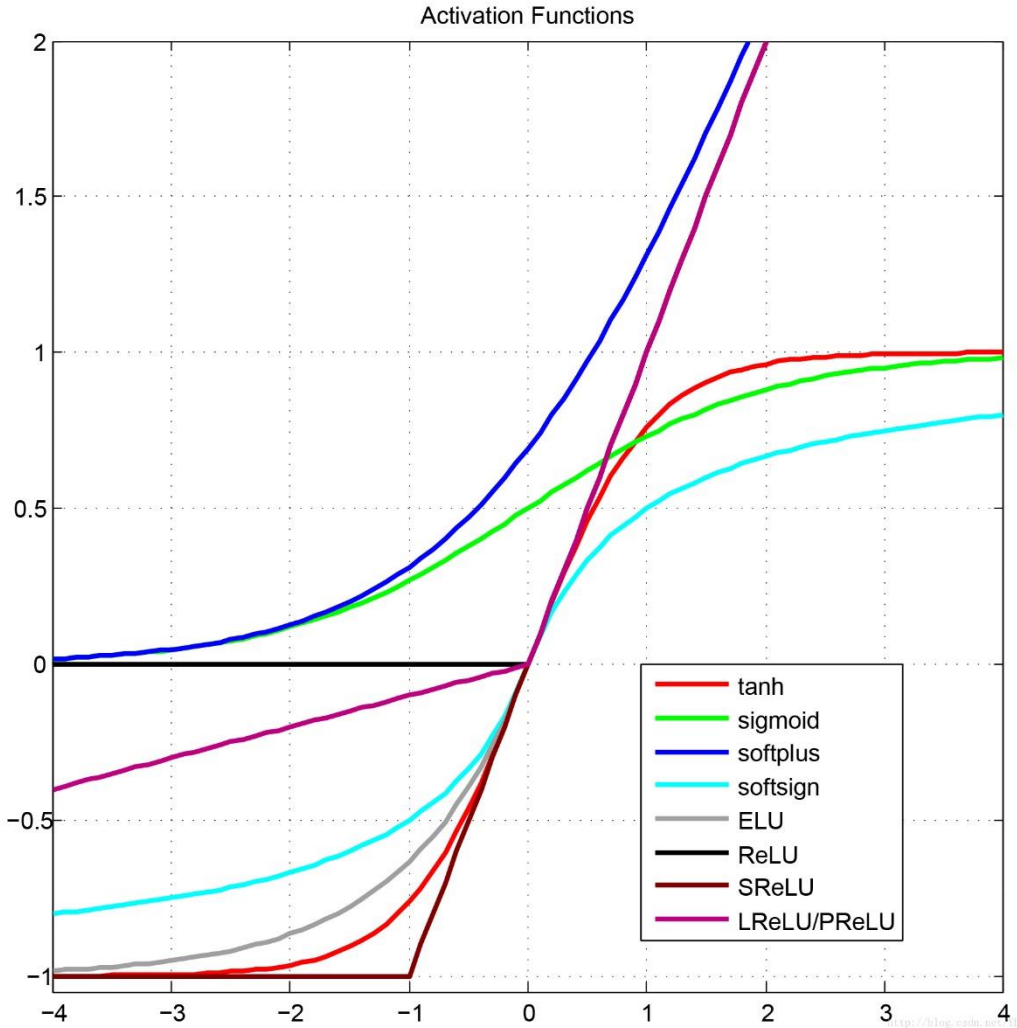


  - **ELU(Exponential Linear Units)**

$$ELU(x) = \begin{cases} x, & if\ x > 0 \\ \alpha(\exp(x) - 1), & if\ x \leq 0 \end{cases}$$

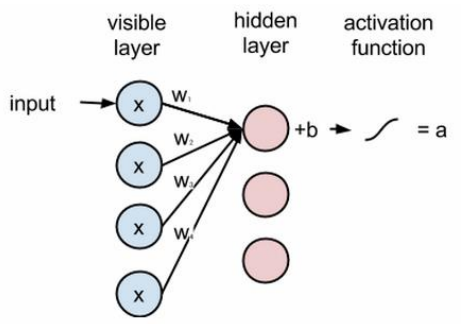


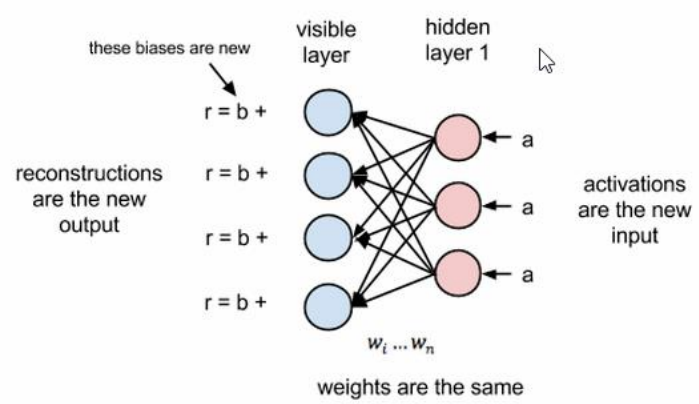
Activation Functions



Xavier/He's Initializer

- **Hinton(2006)의 논문**
  - "A Fast Learning Algorithm for Deep Belief Nets"
    - RBM(Restricted Boltzmann Machine)을 이용하여 초기값을 주면 성능이 크게 개선됨을 증명
      - **2개의 layer로 구성되어 있는 신경망**
      - **같은 layer의 node들끼리 연결이 없음(restricted)**
    - RBM은 비지도학습을 수행하여 weight들을 재구성





- 그러나 RBM은 매우 복잡함
- 매우 단순한 initializer들이 개발됨
  - 2010년, Xavier initializer
  - 2015년, He's initializer
    - node에 대한 입력의 개수(fan in)와 출력의 개수(fan out)을 이용하여 random하게 주어도 RBM과 비슷함을 증명

  - Xavier initializer

```
W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in)
```

  - He's initializer

```
W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in/2)
```

Dropout

- **Overfitting을 줄이는 단순한 방법**
  - 2014년 Srivastava등의 논문
    - NN에서 학습할때, 어떤 노드들의 연결을 random하게 끊어주면 성능이 개선될 수 있음을 증명
    - **주의** : 학습할 때는 일정 비율만큼 dropout하지만, 테스트할 때는 dropout하지 않음