딥러닝 기초

고려대학교 석 준 희

KOREA UNIVERSITY

목차

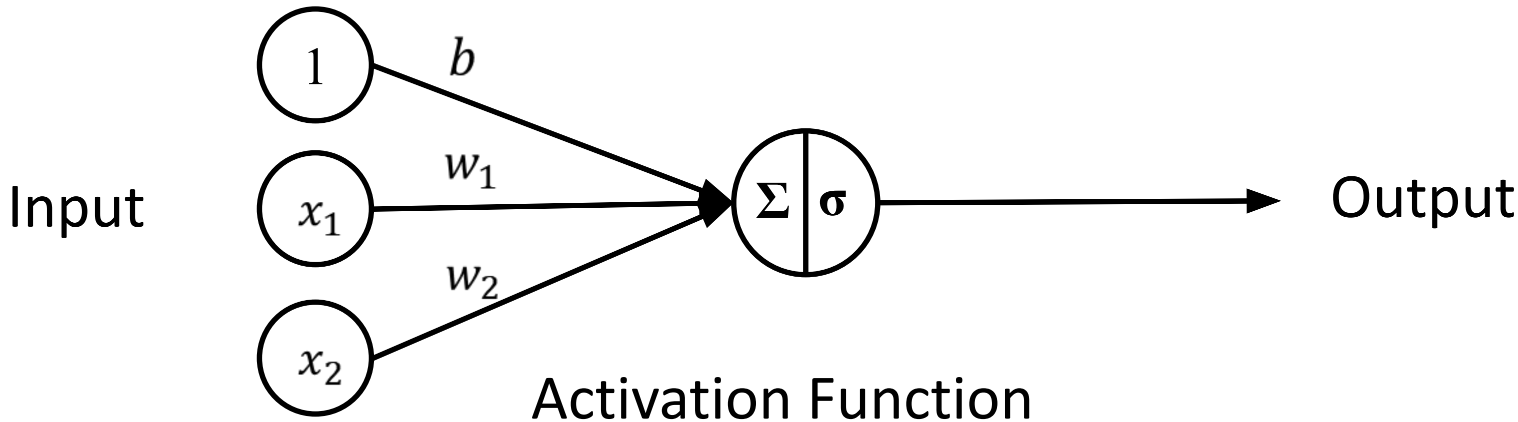
- 심층신경망과 딥러닝
- 심층신경망의 학습
- 다양한 학습 기법
- 프로그래밍 실습

딥러닝 기초

심층신경망과 딥러닝

퍼셉트론

- 뉴런에 대한 수학적 모델: 선형 결합 + 비선형 활성화 함수



$$y = \sigma\left(\sum_j w_j x_j + b\right) = \sigma(\mathbf{w}^T\mathbf{x} + b)$$

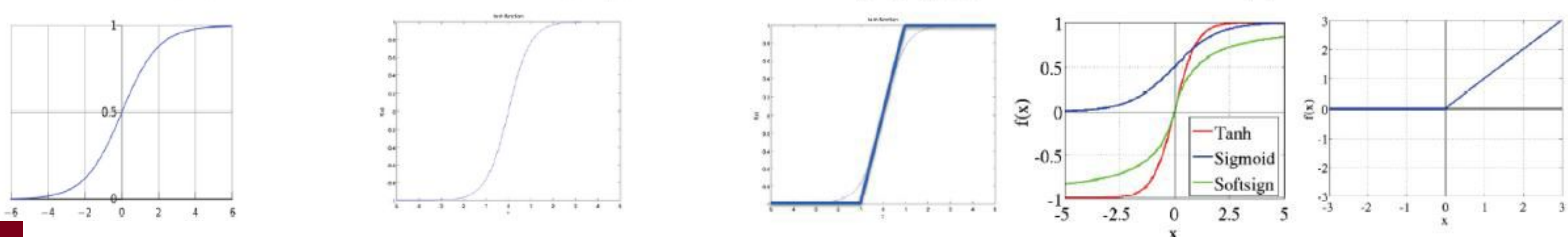
logistic ("sigmoid") $f(z) = \dfrac{1}{1+\exp(-z)}.$

tanh $f(z) = \tanh(z) = \dfrac{e^z - e^{-z}}{e^z + e^{-z}},$

hard tanh $\mathrm{HardTanh}(x) = \begin{cases} -1 & \text{if } x < -1 \\ x & \text{if } -1 <= x <= 1 \\ 1 & \text{if } x > 1 \end{cases}$

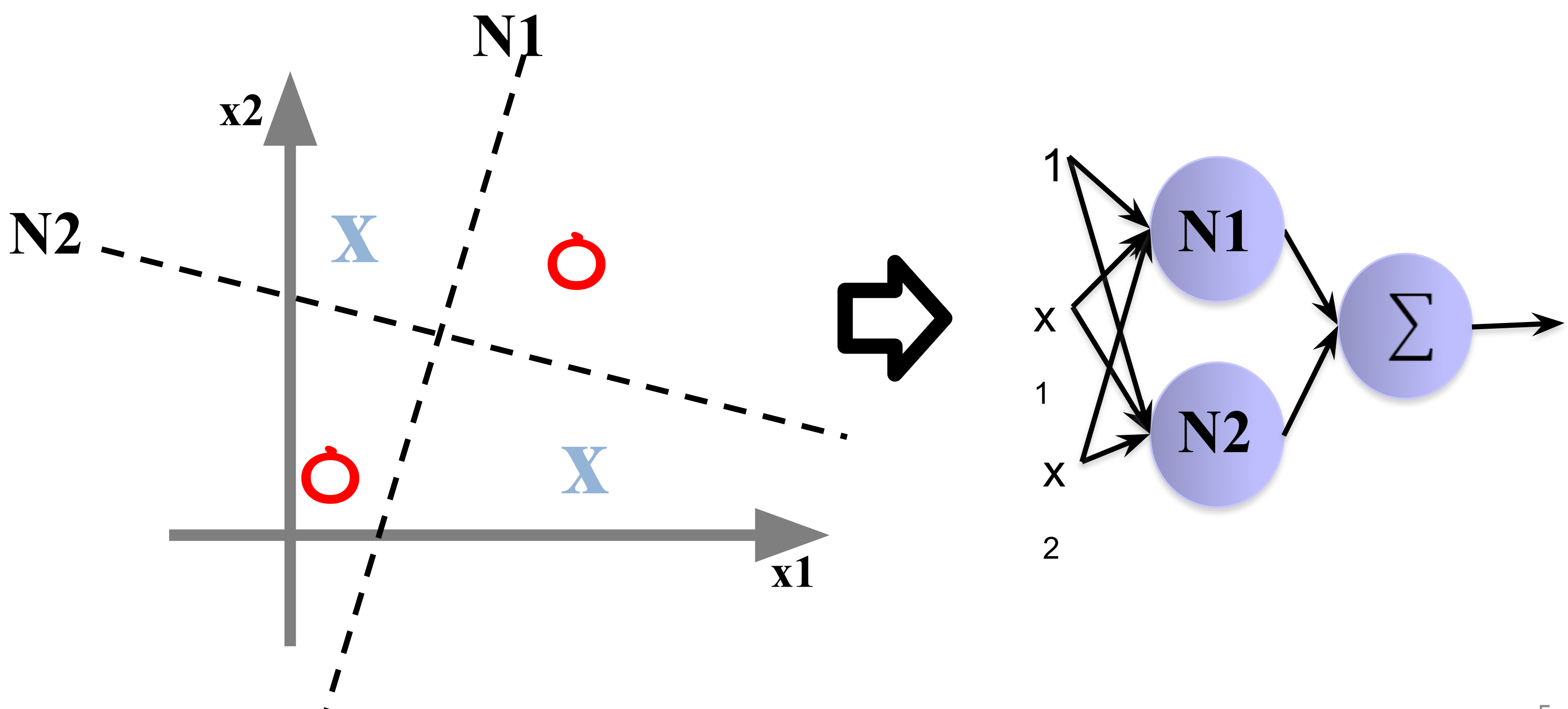
soft sign $\mathrm{softsign}(z) = \dfrac{a}{1+|a|}$

rectifier $\mathrm{rect}(z) = \max(z, 0)$

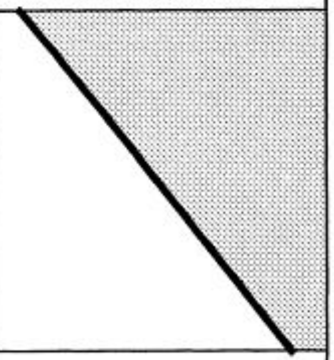
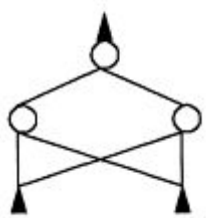
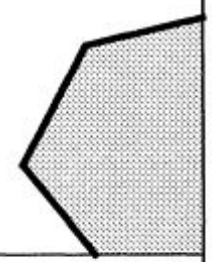
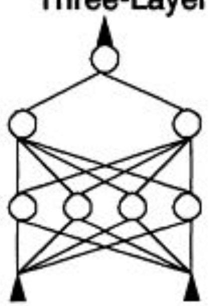
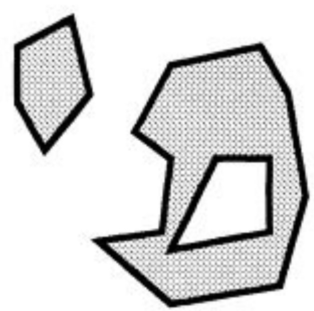


퍼셉트론의 비선형화

- 커널 퍼셉트론
  - 입력에 고차항을 추가 혹은 커널을 이용해 입력 공간을 변형
- **다계층 퍼셉트론 (MLP: Multilayer Perceptron)**
  - 퍼셉트론을 쌓아 비선형성을 추가



심층 구조 (Deep Structure)

Structure	Types of Decision Regions	Most General Region Shapes
Single-Layer 	Half-Plane Bounded by A Hyper-Plane	
Two-Layer 	Convex Open, or Closed Regions	
Three-Layer 	Arbitrary (Complexity Limited by the Number of Nodes)	

- 더 많이 쌓을 수록 더 복잡한 판단의 경계를 형성

More Layers

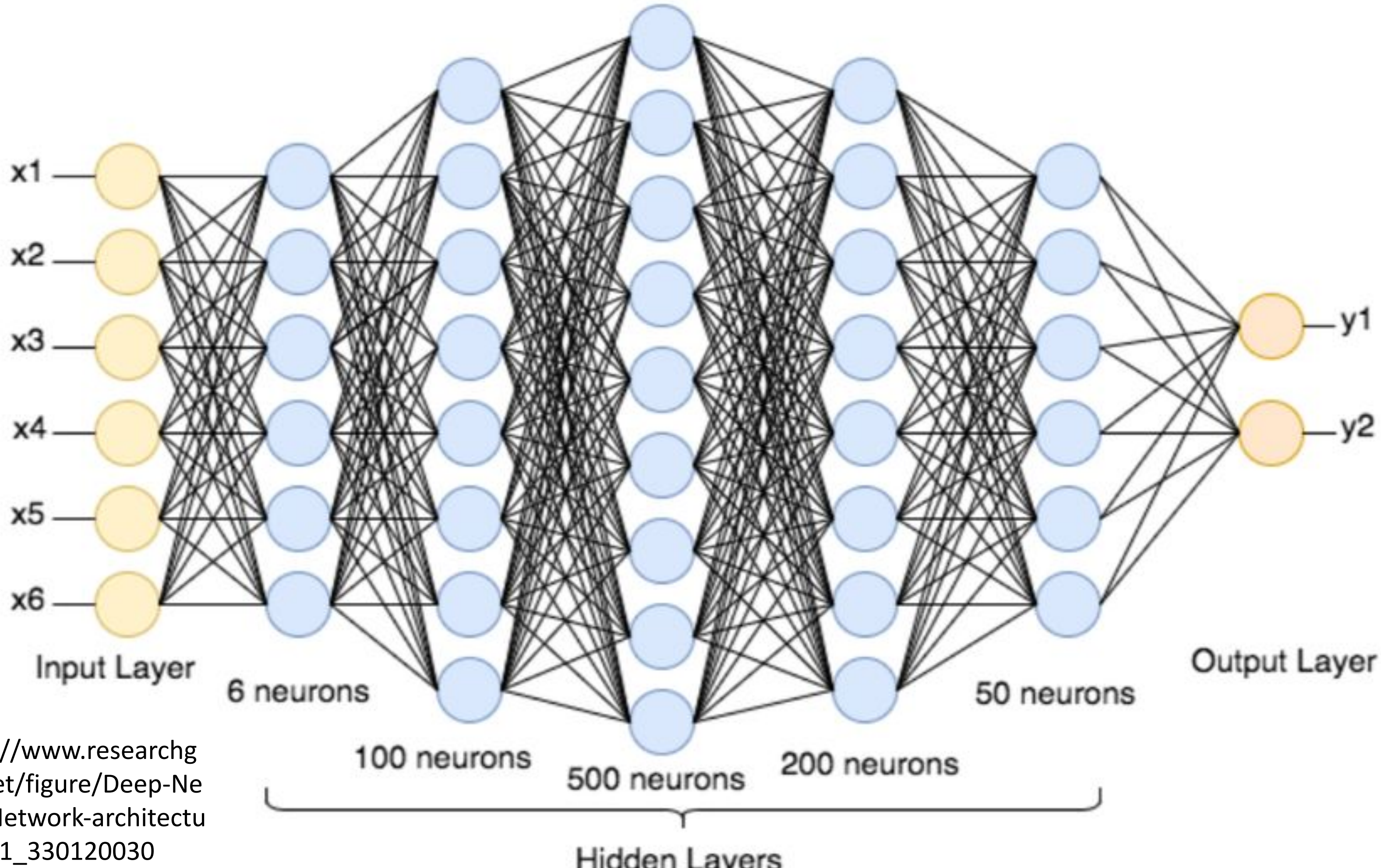
=

Deeper

장교수의 딥러닝, 2017

심층 신경망 (DNN: Deep Neural Network)

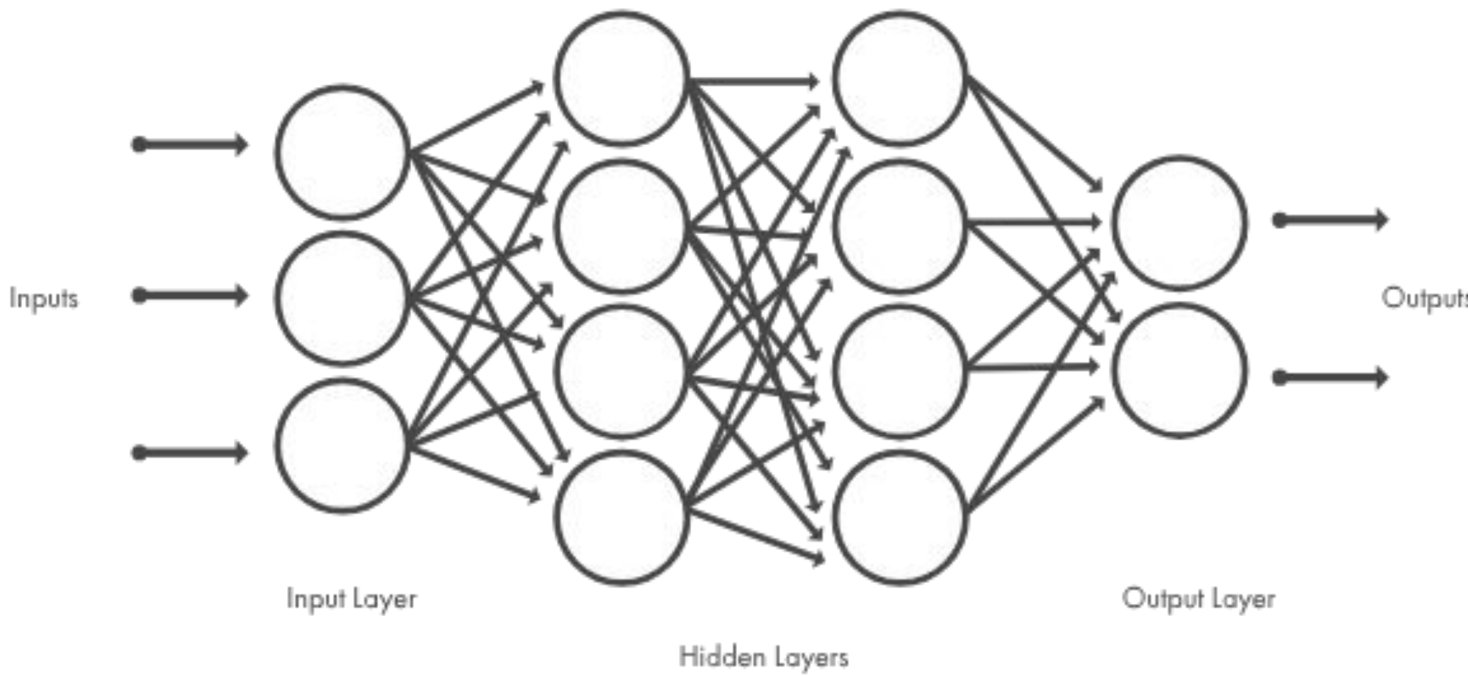
- 심층 신경망: 많은 잠재계층(2개 이상)을 갖는 신경망 구조
  - Shallow network: 0 혹은 1개의 잠재계층을 갖는 구조



https://www.researchgate.net/figure/Deep-Neural-Network-architecture_fig1_330120030

딥러닝 (Deep Learning)

- 딥러닝
  - 외부로 나타나지 않고 숨겨진 계층(hidden layer)가 다수 존재하는 신경망 모델
  - 숨겨진 변수가 학습을 통해 필요한 정보(feature)를 추출하는 모델
- 전통적 접근 vs. 딥러닝 접근
  - 전통적 접근: 인간 전문가가 유용한 정보를 선정하고 이를 데이터에서 계산
  - 딥러닝 접근: 모델이 자동으로 유용한 정보를 추출하여 사용



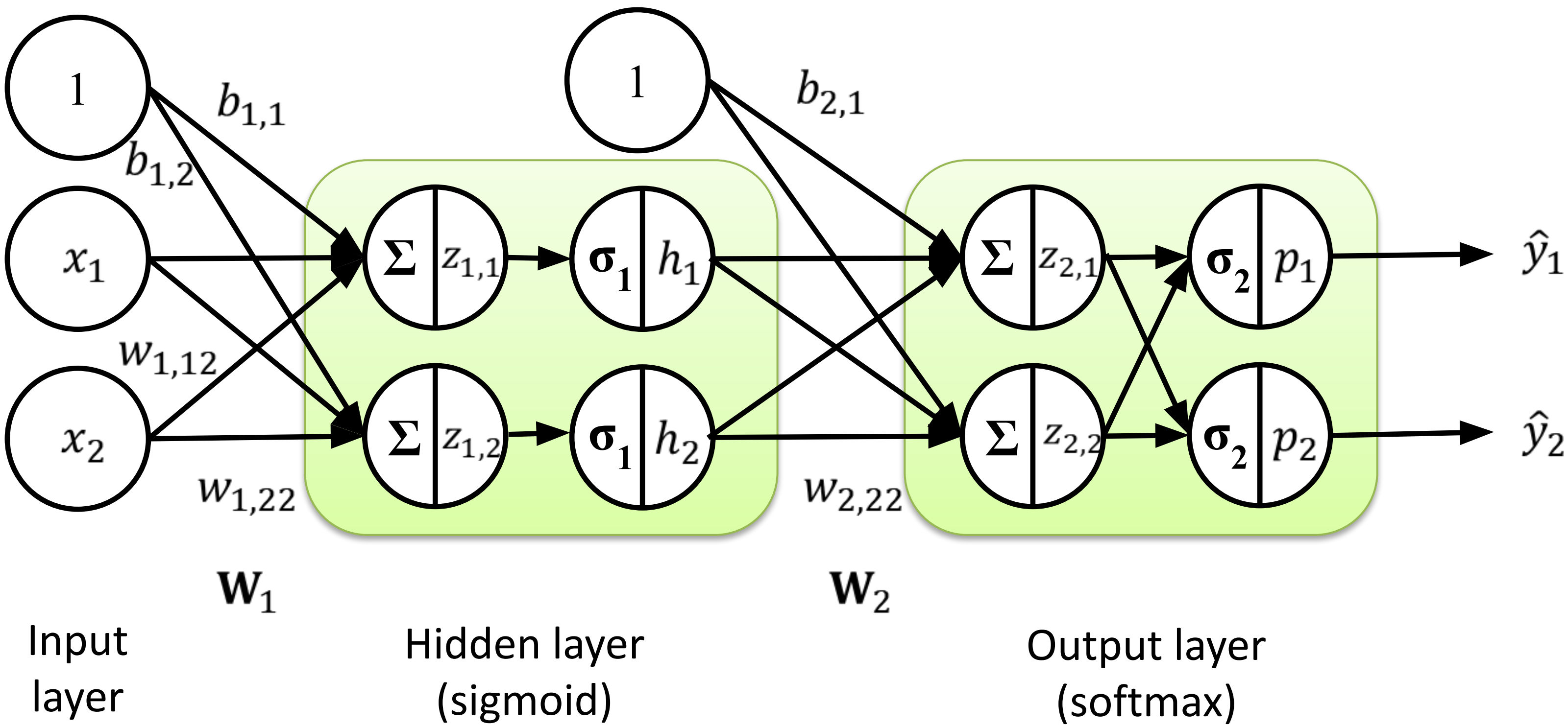
https://kr.mathworks.com/discovery/deep-learning.html

딥러닝 기초

심층신경망의 학습

다계층 퍼셉트론 (MLP: Multi-layer Perceptron)

- 예제: sigmoid layer (2) + softmax layer (2), 이진 분류를 위한 크로스 엔트로피 손실 함수 사용
  - 12개의 파라메터가 존재:

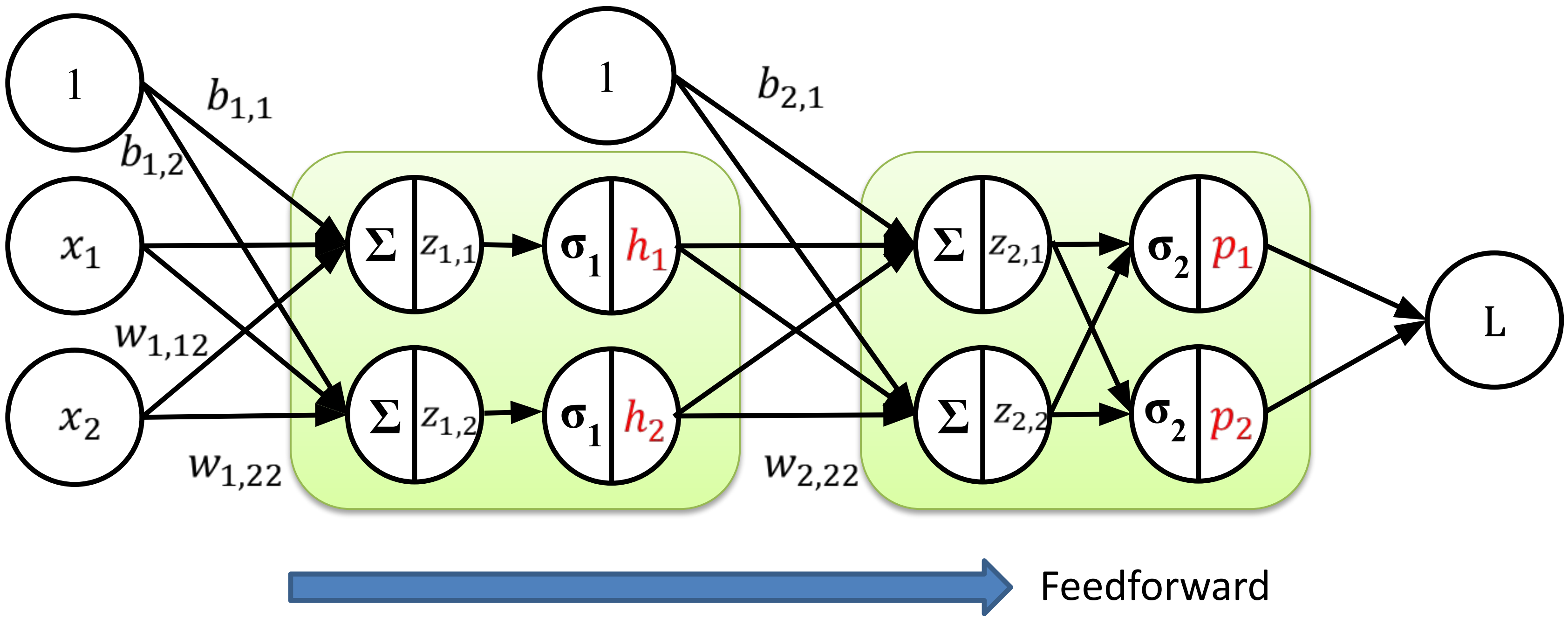


$$L = y_1 \log p_1 + y_2 \log p_2$$

KOREA UNIVERSITY

순방향 계산 (Feedforward Computation)

- 순방향 계산: 입력으로부터 파라메터를 통해 출력과 손실을 계산



$$h_1 = \sigma_1(w_{1,11}x_1 + w_{1,12}x_2 + b_{1,1})$$

$$h_2 = \sigma_1(w_{1,21}x_1 + w_{1,22}x_2 + b_{1,2})$$

$$p_1 = \sigma_2(w_{2,11}h_1 + w_{2,12}h_2 + b_{2,1})$$

$$p_2 = \sigma_2(w_{2,21}h_1 + w_{2,22}h_2 + b_{2,2})$$

$$L = y_1 \log p_1 + y_2 \log p_2$$

KOREA UNIVERSITY

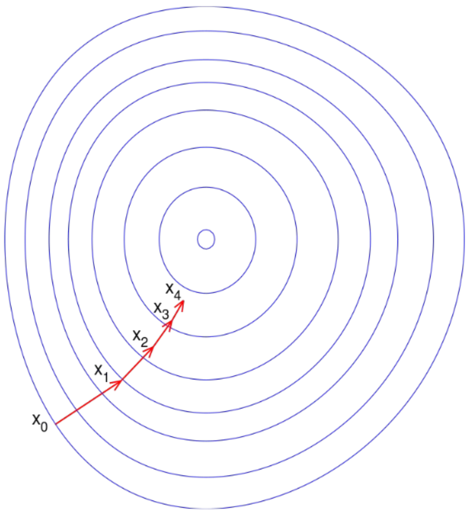
경사하강법을 이용한 손실 함수 최소화

- 손실 함수(loss function)의 최소화
  - 손실함수를 최소화하는 파라메터를 학습
  - 각 파라메터에 대한 손실함수의 미분값(경사도, gradient)를 0으로 하는 값
- 경사하강법 (gradient descent)

$\theta^{(0)}$: random position

$$\theta^{(i+1)} = \theta^{(i)} - \lambda \frac{\partial L(\theta^{(i)})}{\partial \theta}$$

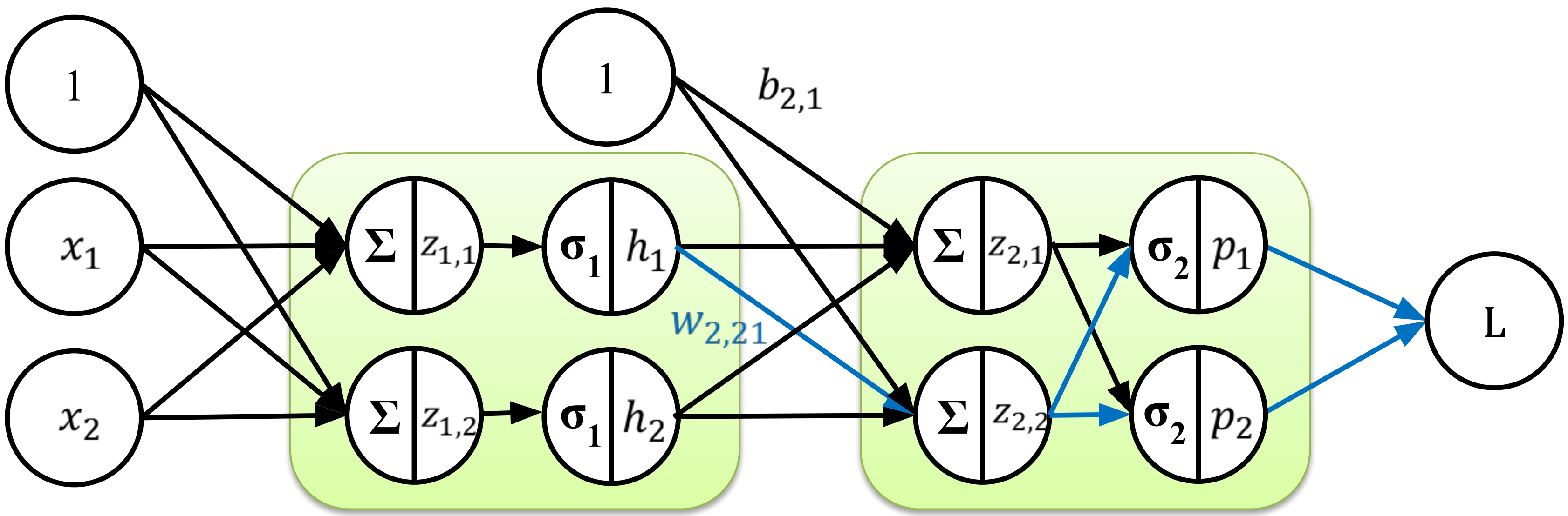
각 파라메터에 대한 경사도의 계산이 핵심!



https://en.wikipedia.org/wiki/Gradient_descent

경사도 계산

- $w_{2,21}$ 에 대한 경사도 계산



$$\frac{\partial L}{\partial w_{2,21}} = \frac{\partial L}{\partial p_1}\frac{\partial p_1}{\partial z_{2,2}}\frac{\partial z_{2,2}}{\partial w_{2,21}} + \frac{\partial L}{\partial p_2}\frac{\partial p_2}{\partial z_{2,2}}\frac{\partial z_{2,2}}{\partial w_{2,21}}$$

$$= \left(\frac{\partial L}{\partial p_1}\frac{\partial p_1}{\partial z_{2,2}} + \frac{\partial L}{\partial p_2}\frac{\partial p_2}{\partial z_{2,2}}\right)\frac{\partial z_{2,2}}{\partial w_{2,21}}$$

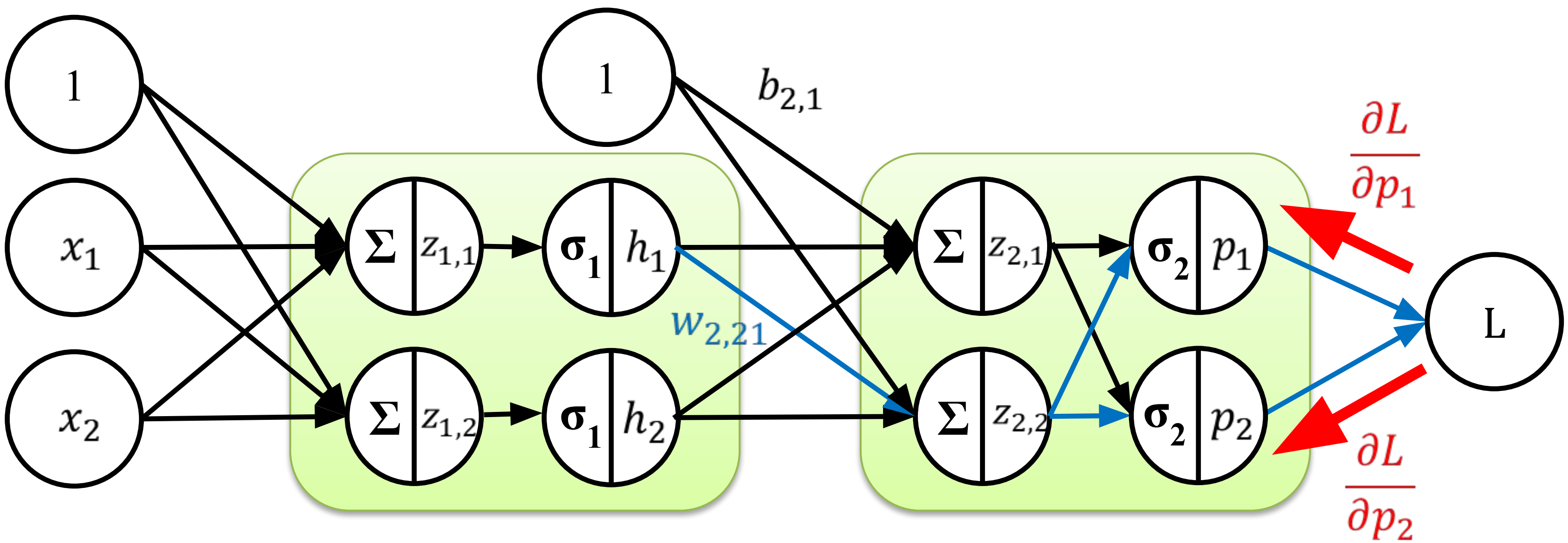
$$= \left(-\frac{y_1}{p_1}\cdot p_1(1-p_1) - \frac{y_2}{p_2}\cdot p_2(1-p_2)\right)\cdot h_1$$

순방향에서 계산된 값들

KOREA UNIVERSITY

경사도 계산

- $w_{2,21}$ 에 대한 경사도 계산



$$\frac{\partial L}{\partial w_{2,21}} = \frac{\partial L}{\partial p_1}\frac{\partial p_1}{\partial z_{2,2}}\frac{\partial z_{2,2}}{\partial w_{2,21}} + \frac{\partial L}{\partial p_2}\frac{\partial p_2}{\partial z_{2,2}}\frac{\partial z_{2,2}}{\partial w_{2,21}}$$

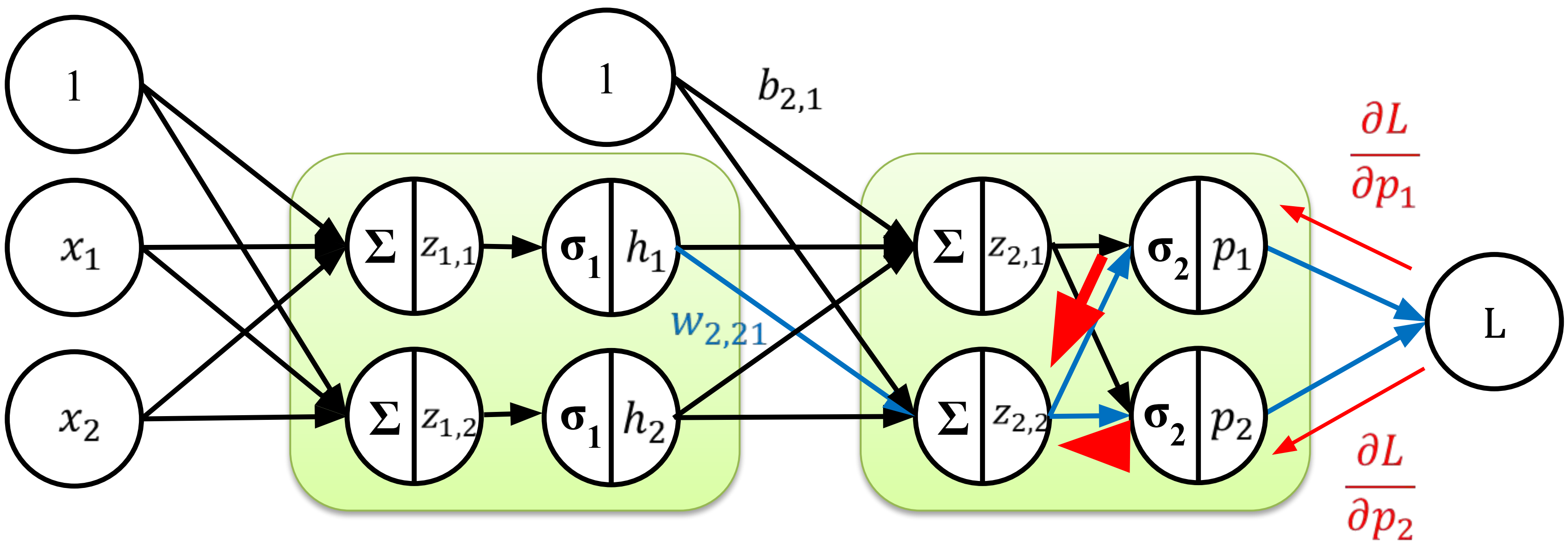
$$= \left(\frac{\partial L}{\partial p_1}\frac{\partial p_1}{\partial z_{2,2}} + \frac{\partial L}{\partial p_2}\frac{\partial p_2}{\partial z_{2,2}}\right)\frac{\partial z_{2,2}}{\partial w_{2,21}}$$

$$= \left(-\frac{y_1}{p_1}\cdot p_1(1-p_1) - \frac{y_2}{p_2}\cdot p_2(1-p_2)\right)\cdot h_1$$

순방향에서 계산된 값들

경사도 계산

- $w_{2,21}$ 에 대한 경사도 계산



$$\frac{\partial L}{\partial w_{2,21}} = \frac{\partial L}{\partial p_1}\frac{\partial p_1}{\partial z_{2,2}}\frac{\partial z_{2,2}}{\partial w_{2,21}} + \frac{\partial L}{\partial p_2}\frac{\partial p_2}{\partial z_{2,2}}\frac{\partial z_{2,2}}{\partial w_{2,21}}$$

$$= \left(\frac{\partial L}{\partial p_1}\frac{\partial p_1}{\partial z_{2,2}} + \frac{\partial L}{\partial p_2}\frac{\partial p_2}{\partial z_{2,2}}\right)\frac{\partial z_{2,2}}{\partial w_{2,21}}$$

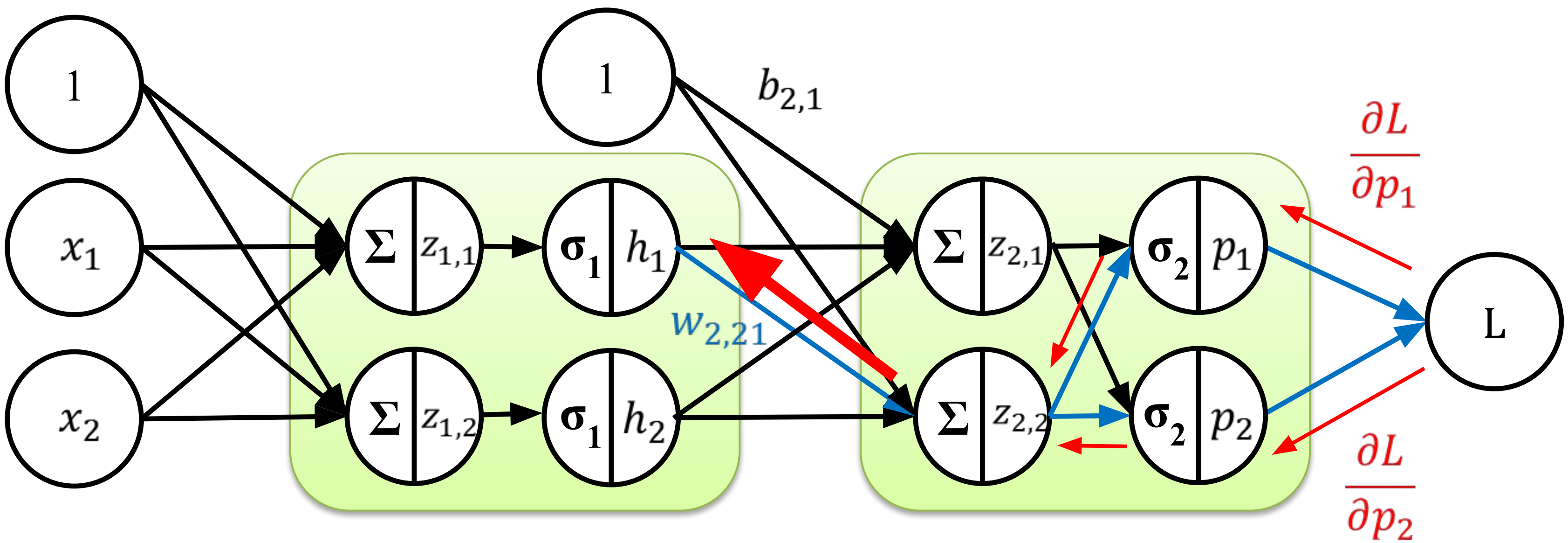
$$= \left(-\frac{y_1}{p_1}\cdot p_1(1-p_1) - \frac{y_2}{p_2}\cdot p_2(1-p_2)\right)\cdot h_1$$

순방향에서 계산된 값들

KOREA UNIVERSITY

역방향 경사도 계산

- $w_{2,21}$ 에 대한 경사도 계산



$$\frac{\partial L}{\partial w_{2,21}} = \frac{\partial L}{\partial p_1}\frac{\partial p_1}{\partial z_{2,2}}\frac{\partial z_{2,2}}{\partial w_{2,21}} + \frac{\partial L}{\partial p_2}\frac{\partial p_2}{\partial z_{2,2}}\frac{\partial z_{2,2}}{\partial w_{2,21}}$$

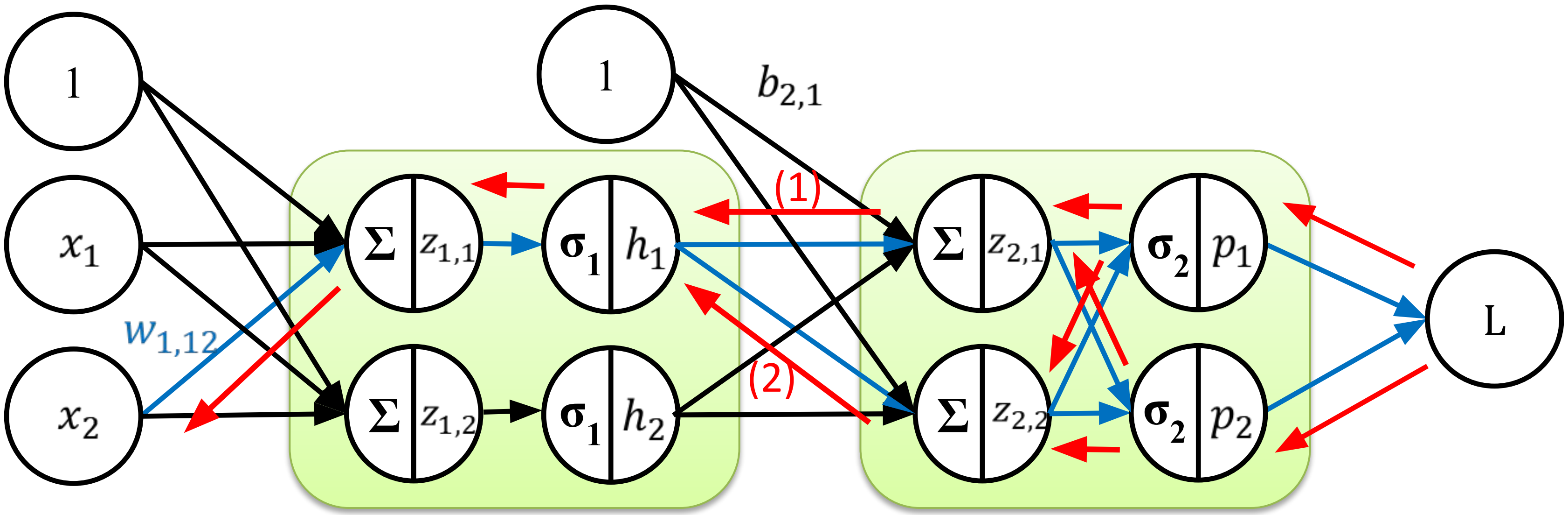
$$= \left(\frac{\partial L}{\partial p_1}\frac{\partial p_1}{\partial z_{2,2}} + \frac{\partial L}{\partial p_2}\frac{\partial p_2}{\partial z_{2,2}}\right)\frac{\partial z_{2,2}}{\partial w_{2,21}}$$

$$= \left(-\frac{y_1}{p_1}\cdot p_1(1-p_1) - \frac{y_2}{p_2}\cdot p_2(1-p_2)\right)\cdot h_1$$

순방향에서 계산된 값들

역방향 경사도 계산

- 또 다른 예제, $w_{1,12}$ 에 대한 경사도 계산



$$(1) = \left(\frac{\partial L}{\partial p_1}\frac{\partial p_1}{\partial z_{2,1}} + \frac{\partial L}{\partial p_2}\frac{\partial p_2}{\partial z_{2,1}}\right)\frac{\partial z_{2,1}}{\partial h_1}$$

$$(2) = \left(\frac{\partial L}{\partial p_1}\frac{\partial p_1}{\partial z_{2,2}} + \frac{\partial L}{\partial p_2}\frac{\partial p_2}{\partial z_{2,2}}\right)\frac{\partial z_{2,2}}{\partial h_1}$$

$$\frac{\partial L}{\partial w_{1,12}} = [(1) + (2)]\frac{\partial h_1}{\partial z_{1,1}}\frac{\partial z_{1,1}}{\partial w_{1,12}}$$

역전파 (Backpropagation)

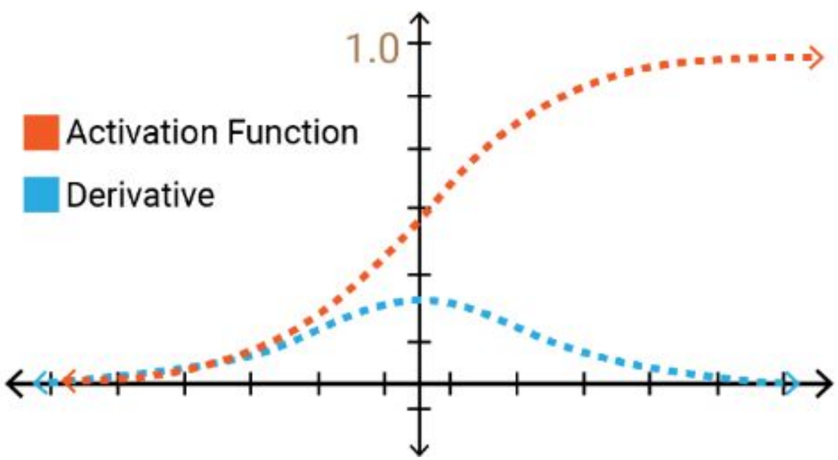
- **역전파 알고리즘**은 신경망의 가중치를 학습하는 주요한 방법으로 순방향과 역방향 계산을 반복함으로써 수행됨
  - 오늘날 신경망 훈련에 가장 많이 사용되는 알고리즘

- 대략적 알고리즘
  - (1) 가중치의 초기화 (보통 랜덤한 값으로)
  - (2) 순방향 계산을 통해 출력과 손실을 계산
  - (3) 역방향 계산을 통해 경사치를 계산하고 파라메터를 업데이트
  - (4) 수렴할 때까지 (2)와 (3)을 반복

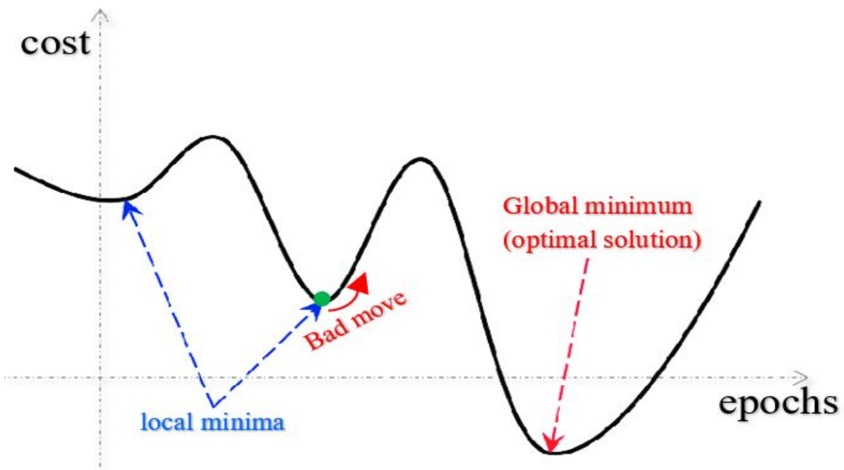
- 역전파 알고리즘은 생물학적 지식과 관련이 없음
  - 신경망은 생물학적 신경세포로부터 발전했지만, 생물학적으로 역전파와 같은 원리는 존재하지 않음
  - 역전파는 많은 양의 데이터가 필요한데 비해 생물학적 신경세포는 더 적은 데이터로부터 좀 더 천천히 학습함
    - 새로운 학습 방법이 존재할 가능성!!

심층 신경망 학습의 문제

- **경사도 소멸 (vanishing gradient):** 역전파에 따라 경사도가 점점 작아지고 결국 소멸하는 현상
  - 보통 활성화 함수의 미분값이 1보다 작기 때문에 발생하는 현상
  - 경사도 소멸은 심층 신경망의 훈련을 어렵게 하는 주된 요인
  - 이를 방지하기 위한 다양한 기법 (e.g. ReLU, LSTM, ResNet)이 고안됨
- **국소 최적화 (local minimum):** 손실함수가 충분히 최적화 되지 않았음에도 경사도가 0이 되는 문제
  - 경사하강법 류의 방법에서 공통적으로 발생하는 문제
  - 완전히 해결하는 것은 불가능하기 때문에 좋은 시작점을 반복적으로 찾고자 함



https://towardsdatascience.com/the-problem-of-vanishing-gradients-68cea05e2625



https://www.researchgate.net/figure/local-minima-vs-global-minimum_fig2_341902041

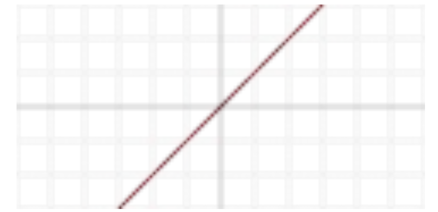
딥러닝 기초

다양한 학습 기법

활성화 함수

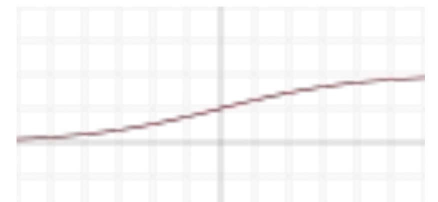
- **Identity**
  - 선형 활성화 함수

$$\sigma(z) = z$$
$$\sigma'(z) = 1$$



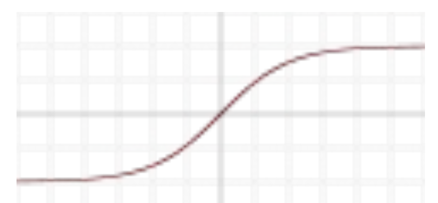
- **Sigmoid (Logistic)**
  - 기본적인 활성화 함수
  - 경사도 소멸의 문제가 발생
  - 항상 양수만을 도출하여 편향이 발생

$$\sigma(z) = \frac{1}{1+e^{-z}}$$
$$\sigma'(z) = \sigma(z)\big(1 - \sigma(z)\big)$$



- **Hyperbolic tangent**
  - Sigmoid와 유사하나 0 중심
  - 양수/음수 모두 도출되어 편향이 없음
  - 경사도 소멸의 문제가 상대적으로 적지만 여전히 존재

$$\sigma(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$$
$$\sigma'(z) = 1 - \sigma(z)^2$$

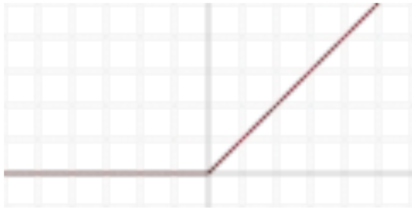


https://en.wikipedia.org/wiki/Activation_function

활성화 함수

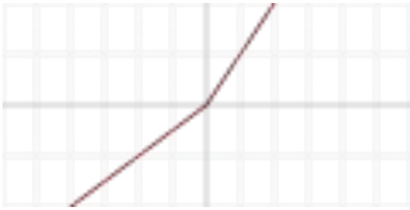
- **ReLU (Rectifier Linear Unit)**
  - 일반적으로 심층신경망에 가장 많이 사용되는 함수
  - 장점: 희소 표현이 가능. 경사도 소멸이 적음, 계산이 효율적
  - 단점: 양수만이 도출, 값의 제한이 없음, 모든 경사도가 0이 되는 현상이 발생

$$\sigma(z) = \max(0, z)$$
$$\sigma'(z) = 0 \ \text{ or } \ 1$$



- **Leaky ReLU**
  - ReLU을 보완하기 위한 변형
  - 경사도가 0이 되는 현상을 방지
  - GELU, ELU, SELU 등의 다양한 변형

$$\sigma(z) = \max(\alpha z, z)$$
$$\sigma'(z) = \alpha \ \text{ or } \ 1$$
보통 $\alpha = 0.01$



https://en.wikipedia.org/wiki/Activation_function

손실함수 최소화 기법

- **손실함수 최소화**
  – 딥러닝 모델 훈련의 핵심
  – 손실함수를 최소화하는 파라메터를 찾고자 함
- **경사하강법 (gradient descent)**
  – 임의의 파라메터에서 시작하여 경사를 따라 파라메터의 변화가 0이 될 때까지 업데이트
  – 배치 경사하강법: 모든 데이터를 이용
  – 확률적 경사하강법: 하나의 샘플을 이용
  – 미니배치 경사하강법: 소수의 샘플을 이용
- 경사하강법의 문제
  – 학습률을 정하기가 어려움
  – 너무 큰 학습률: 학습은 빠르지만 불안정
  – 너무 작은 학습률: 안정적이지만 학습이 느림

$$\min_{\theta} L(\theta)$$

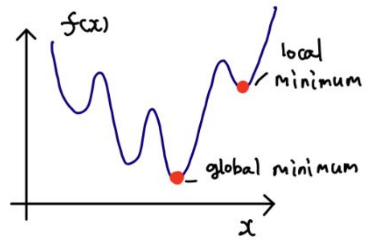
$\theta^{(0)}$: random position

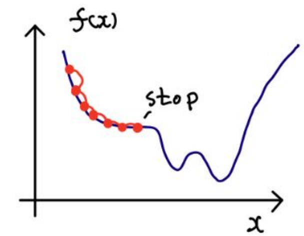
$$\theta^{(i+1)} = \theta^{(i)} - \lambda \nabla L\left(\theta^{(i)}\right)$$

KOREA UNIVERSITY

손실함수 최소화 기법

- **모멘텀 최적화 (Momentum Optimizer)**
  - 관성 계수를 추가하여 경사값을 업데이트
  - 지역최저점이나 안장점(최저점이 아니지만 경사도가 0인 지점)의 문제를 해결
  - 관성에 의해 더 빠른 학습이 가능





$$m^{(i)} = \beta_1 m^{(i-1)} + (1-\beta_1)\nabla L\left(\theta^{(i)}\right)$$

$$\theta^{(i+1)} = \theta^{(i)} - \lambda m^{(i)}$$

https://icim.nims.re.kr/post/easyMath/428

- **Adaptive Gradient (Adagrad)**
  - 일반적인 GD는 모든 파라메터가 같은 학습률로 학습되어 비효율적
  - 변화가 적은 파라메터는 더 큰 학습률을 변화가 많이 파라메터를 작은 학습률로 업데이트

k번째 파라메터 θ_k에 대한 변화량 제곱의 합

$$g_k^{(i)} = g_k^{(i-1)} + \nabla_k L\left(\theta^{(i)}\right)^2$$

$$\theta_k^{(i+1)} = \theta_k^{(i)} - \frac{\lambda}{\sqrt{g_k^{(i)} + \epsilon}} \nabla_k L\left(\theta^{(i)}\right)$$

손실함수 최소화 기법

- **RMSProp**
  - Adagrad는 점차 학습률이 너무 작아져 더 이상 학습이 진행되지 않는 문제가 발생
  - $g_k^{(i)}$ 계산에 이동평균을 적용하여 0으로 빠르게 떨어지는 것을 방지

$$g_k^{(i)} = \beta g_k^{(i-1)} + (1-\beta)\nabla_k L\left(\theta^{(i)}\right)^2$$

$$\theta_k^{(i+1)} = \theta_k^{(i)} - \frac{\lambda}{\sqrt{g_k^{(i)} + \epsilon}} \nabla_k L\left(\theta^{(i)}\right)$$

- **Adam (Adaptive Momentum Estimation)**
  - Momentum + RMSProp
  - Momentum을 이용한 빠르고 지속적인 업데이트
  - RMSProp을 이용한 파라메터 별 차등적인 업데이트

$$m_k^{(i)} = \beta_1 m_k^{(i-1)} + (1-\beta_1)\nabla_k L\left(\theta^{(i)}\right)$$

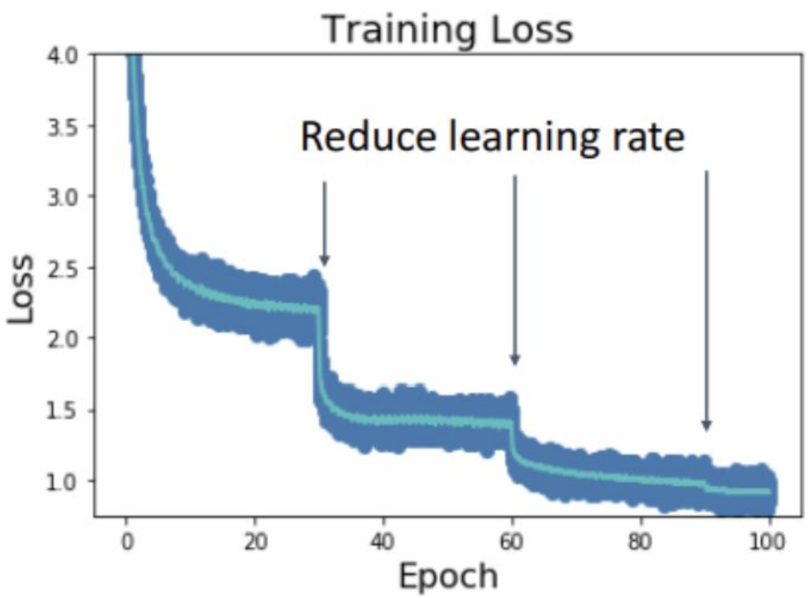
$$g_k^{(i)} = \beta_2 g_k^{(i-1)} + (1-\beta_2)\nabla_k L\left(\theta^{(i)}\right)^2$$

$$\theta_k^{(i+1)} = \theta_k^{(i)} - \frac{\lambda}{\sqrt{g_k^{(i)} + \epsilon}} m_k^{(i)}$$

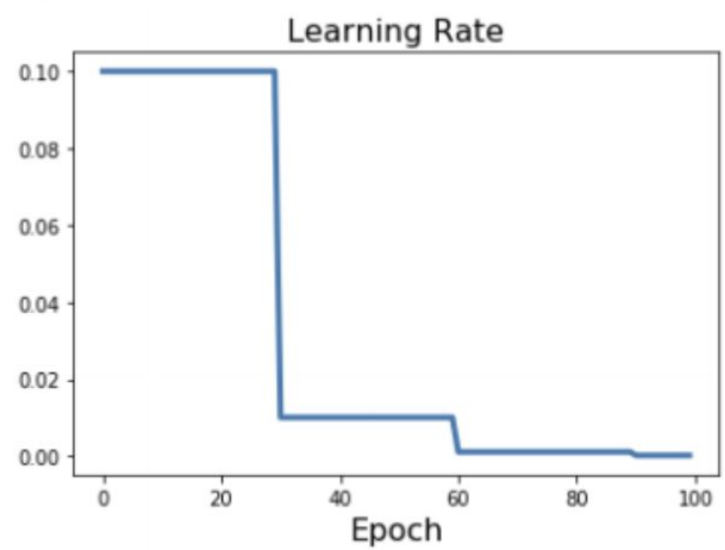
KOREA UNIVERSITY

손실함수 최소화 기법

- **Learning Rate Decay**
  - 여러가지 방식에도 불구하고 최적화가 진행될 수록 학습률은 작아져야함
  - 학습에 따라 학습률을 줄여가는 방식



Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.



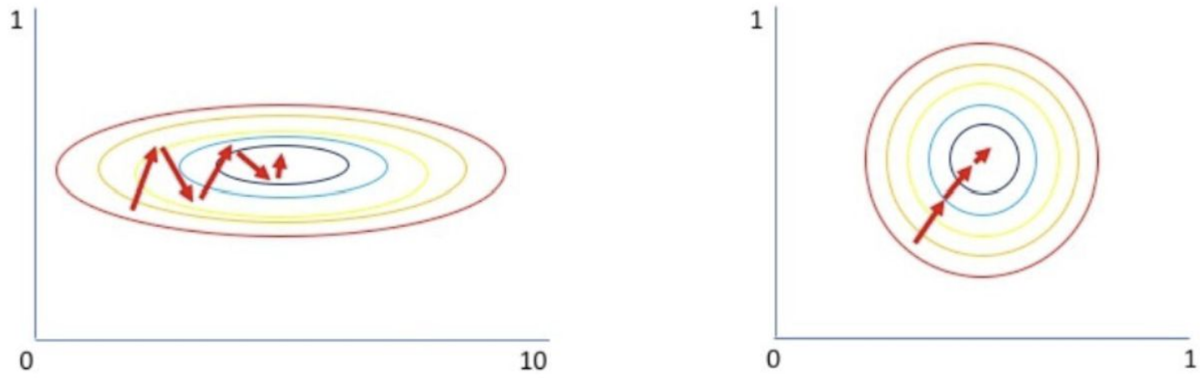
http://cs231n.stanford.edu/

KOREA UNIVERSITY

정규화 (Normalization)

- **정규화(Normalization)란?**
  - 신경망 계층의 입력으로 들어가는 데이터의 분포를 일정하게 맞추어 주는 과정
  - 안정적이고 빠른 학습을 위해 주로 사용함

https://medium.com/@limxiuxian98/batch-normalization-what-should-we-know-8c6f5d30d3ad



- **데이터 정규화 (Data Normalization)** or Data Scaling
  - 입력 계층에 들어가는 주어진 데이터(X)를 정규화하는 과정
  - 모델 훈련의 일부가 아니라 전처리의 일부
  - Min-max normalization: 데이터를 0~1사이로 정규화
  - Standardization: 데이터를 평균 0, 분산 1로 정규화

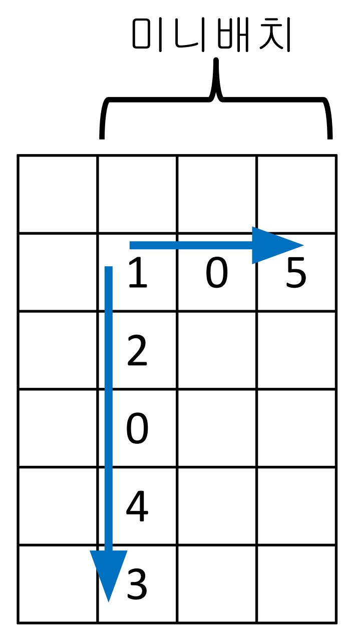
KOREA UNIVERSITY

정규화 (Normalization)

- 계층내 정규화 (In-layer Normalization)
  - 신경망 내부에서 한 계층에서 다음 계층으로 전달되는 입력을 정규화
- **배치 정규화 (Batch Normalization)**
  - 미니배치 내에서 각 변수가 같은 분포를 갖도록 정규화 (across-sample)
  - 주로 이미지 분석(CNN모델)에 사용
- **계층 정규화 (Layer Normalization)**
  - 하나의 표본 내에서 변수들의 같은 분포를 갖도록 정규화 (across-variable)
  - 주로 텍스트 분석(RNN모델)에 사용
- 가중치 정규화 (Weight Normalization)
  - 데이터 대신에 모델의 가중치를 정규화

변수별로
학습되는
파라메터

$$x_n = \frac{x-\mu}{\sigma} \cdot \gamma + \beta$$



$\mu = 2, \sigma^2 = 7$

$\mu = 2, \sigma^2 = 3.5$

규제화 (Regularization)

- **규제화(Regularization)이란?**
  - 모델의 과대적합(overfitting)을 방지하기 위해 모델의 복잡성을 줄이는 방식

- **Weight Decay**
  - 학습 과정에서 큰 파라메터에 대해서는 큰 페널티를 부여하여 파라메터가 너무 커지는 것을 방지, 손실함수에 벌점항을 추가하여 구현
  - 근본적으로 “모델 선택과 확장”에서 다룬 규제화와 동일
  - 벌점은 L_0, L_1, L_2, $L_{infinite}$ 등이 가능

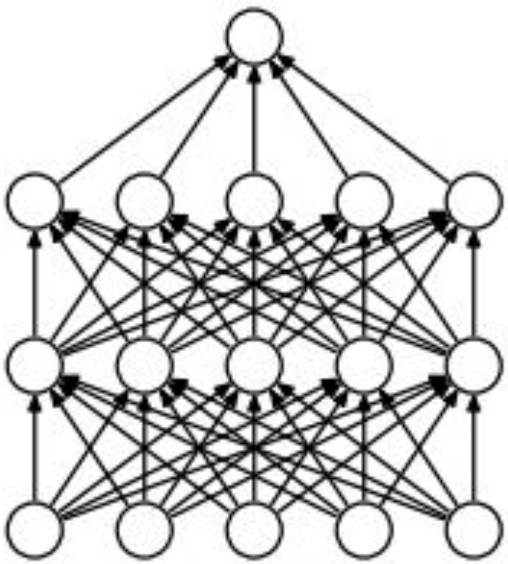
$$\mathrm{Loss}(\theta, \lambda | X, Y) = \mathrm{Error}(\theta | X, Y) + \lambda \mathrm{Penalty}(\theta)$$

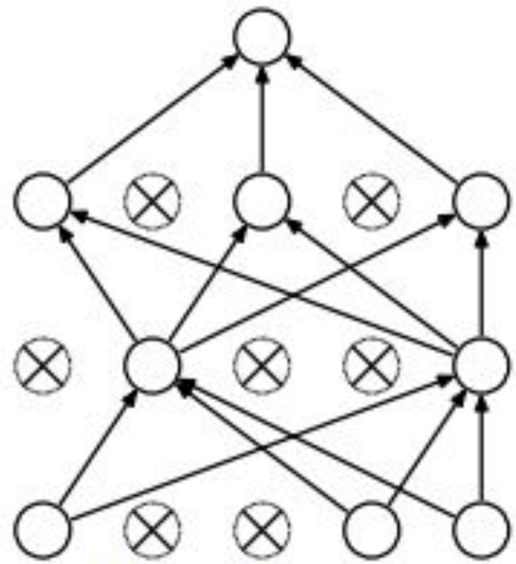
$$Loss = -\sum_i \left[y_i \log\left(\frac{e^{\beta_0 + \beta_1 x}}{1 + e^{\beta_0 + \beta_1 x}} \right) + (1 - y_i) \log\left(\frac{1}{1 + e^{\beta_0 + \beta_1 x}} \right) \right] + \lambda \beta_1^2$$

KOREA UNIVERSITY

규제화 (Regularization)

- **Dropout**
  - 훈련과정에서 노드를 임의로 삭제하여 출력이 일부 노드에만 의존하는 현상을 방지 → 좀 더 안정적인 예측이 가능
  - 훈련 과정에서는 매 순방향 계산에서 p의 비율로 노드를 삭제하고, 그 대신 각 출력을 $1/(1-p)$만큼 스케일하여 계산
  - 예측을 할 때는 모든 노드를 그대로 사용


(a) Standard Neural Net


(b) After applying dropout.

Srivastava et al, 2014

딥러닝 기초

프로그래밍 실습

라이브러리와 데이터

```
[ ]  # 라이브러리
     import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     from sklearn.model_selection import train_test_split
```

```
[ ]  # 데이터셋
     from sklearn.datasets import load_diabetes
     X, y_numeric = load_diabetes(return_X_y=True)
     # numpy 형태로 y를 배열
     y = np.array([ 0 if y_numeric[i]<140 else 1 for i in range(len(y_numeric)) ])
     xtrain, xtest, ytrain, ytest = train_test_split(X,y,test_size=0.4,random_state=42)
```

Scikit-Learn을 이용한 신경망 모델

```
[ ]  from sklearn.neural_network import MLPClassifier
     f = MLPClassifier(
             hidden_layer_sizes = (10,5),
             activation = 'logistic',
             solver = 'lbfgs', # for small data set, sgd/adam for large data set
             alpha = 0.01, # L2 regularization
             batch_size = 'auto',
             learning_rate = 'constant',
             learning_rate_init = 0.001,
             random_state = 0,
             max_iter = 10000)
```

```
[ ]  f.fit(xtrain,ytrain)
     print( f.score(xtrain,ytrain), f.score(xtest,ytest) )
```

```
0.9962264150943396 0.711864406779661
```

KOREA UNIVERSITY

Tensorflow를 이용한 신경망 모델

```
[ ]  # 텐서플로우 라이브러리
     import tensorflow as tf
```

신경망 모델의 선언

```
[ ]  # 인공신경망 모델
     model = tf.keras.models.Sequential()
     model.add( tf.keras.layers.Input(shape=(10,)) )     # 입력 변수의 수 10
     model.add( tf.keras.layers.Dense(10,activation='sigmoid') )
     model.add( tf.keras.layers.Dense(5,activation='sigmoid') )
     model.add( tf.keras.layers.Dense(2,activation='softmax'))
```

```
[ ]  # 모델 컴파일
     model.compile(optimizer='adam',
                   loss='sparse_categorical_crossentropy',
                   metrics=['accuracy'])
```

Tensorflow를 이용한 신경망 모델

모델 훈련

```
[ ]  model.fit(xtrain,ytrain,epochs=5) # 최초 5번
```

```
Epoch 1/5
9/9 [==============================] - 2s 7ms/step - loss: 0.9480 - accuracy: 0.5094
Epoch 2/5
9/9 [==============================] - 0s 5ms/step - loss: 0.9156 - accuracy: 0.5094
Epoch 3/5
9/9 [==============================] - 0s 9ms/step - loss: 0.8833 - accuracy: 0.5094
Epoch 4/5
9/9 [==============================] - 0s 12ms/step - loss: 0.8564 - accuracy: 0.5094
Epoch 5/5
9/9 [==============================] - 0s 5ms/step - loss: 0.8289 - accuracy: 0.5094
<keras.callbacks.History at 0x78d2020d5e10>
```

```
[ ]  model.fit(xtrain,ytrain,epochs=1000) # 추가 1000 번
```

```
[ ]  model.fit(xtrain,ytrain,epochs=5)  # 추가 5 번
```

```
Epoch 1/5
9/9 [==============================] - 0s 3ms/step - loss: 0.5134 - accuracy: 0.7321
Epoch 2/5
9/9 [==============================] - 0s 3ms/step - loss: 0.5129 - accuracy: 0.7321
Epoch 3/5
9/9 [==============================] - 0s 4ms/step - loss: 0.5130 - accuracy: 0.7245
Epoch 4/5
9/9 [==============================] - 0s 3ms/step - loss: 0.5133 - accuracy: 0.7245
Epoch 5/5
9/9 [==============================] - 0s 4ms/step - loss: 0.5135 - accuracy: 0.7245
<keras.callbacks.History at 0x78d1fe45b610>
```

Tensorflow를 이용한 신경망 모델

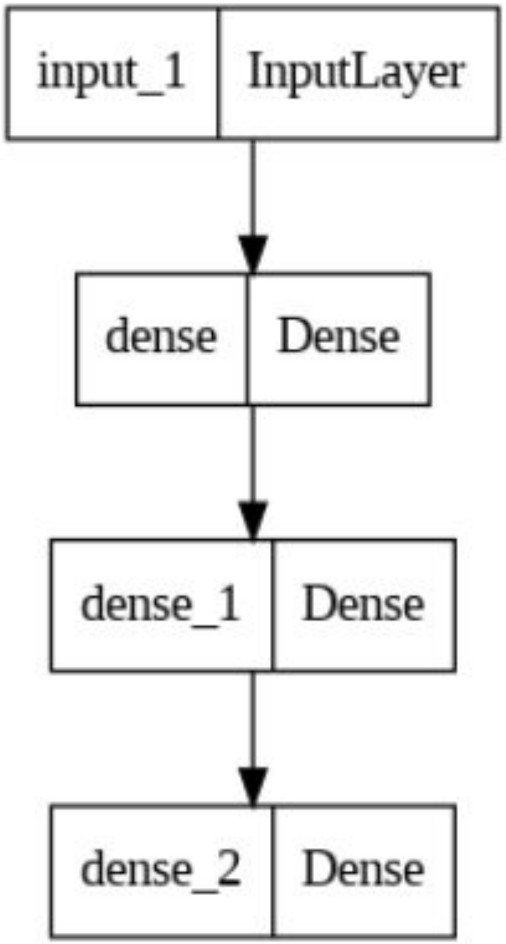
모델의 모습

```
[ ]  model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 10)                110

 dense_1 (Dense)             (None, 5)                 55

 dense_2 (Dense)             (None, 2)                 12

=================================================================
Total params: 177
Trainable params: 177
Non-trainable params: 0
_________________________________________________________________
```

Tensorflow를 이용한 신경망 모델

```
[ ]  tf.keras.utils.plot_model(model)
```



Tensorflow를 이용한 신경망 모델

모델을 이용한 예측

```
[ ]  xtrain[0]
```

```
array([-0.01277963, -0.04464164,  0.06061839,  0.05285804,  0.04796534,
        0.02937467, -0.01762938,  0.03430886,  0.07020738,  0.00720652])
```

```
[ ]  model.predict(xtrain[[0]])
```

```
1/1 [==============================] - 0s 117ms/step
array([[0.08856107, 0.91143894]], dtype=float32)
```

KOREA UNIVERSITY

Tensorflow를 이용한 신경망 모델

모델 평가

```
[ ]  model.evaluate(xtrain,ytrain)
```

```
9/9 [==============================] - 0s 3ms/step - loss: 0.5132 - accuracy: 0.7245
[0.5132156014442444, 0.7245283126831055]
```

```
[ ]  model.evaluate(xtest,ytest)
```

```
6/6 [==============================] - 0s 3ms/step - loss: 0.4408 - accuracy: 0.7853
[0.440816730260849, 0.7853107452392578]
```

KOREA UNIVERSITY

감사합니다