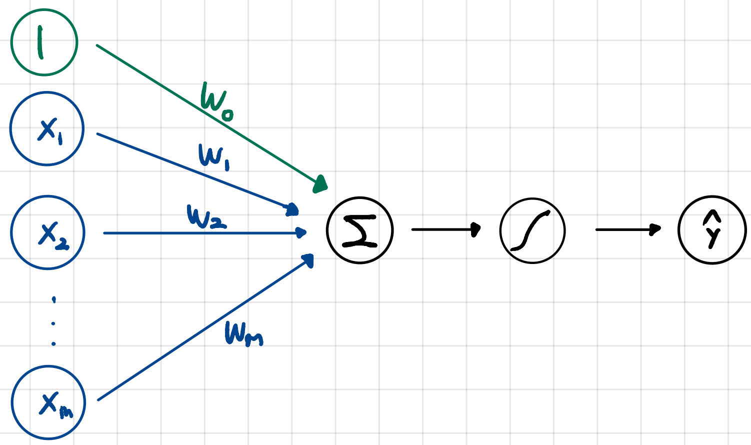
The Perceptron: The structural building block of deep learning

- Single Neuron

· Forward Propagation



Input → output → sum → Non-linearity → output

output ↓   Linear Combination ↓

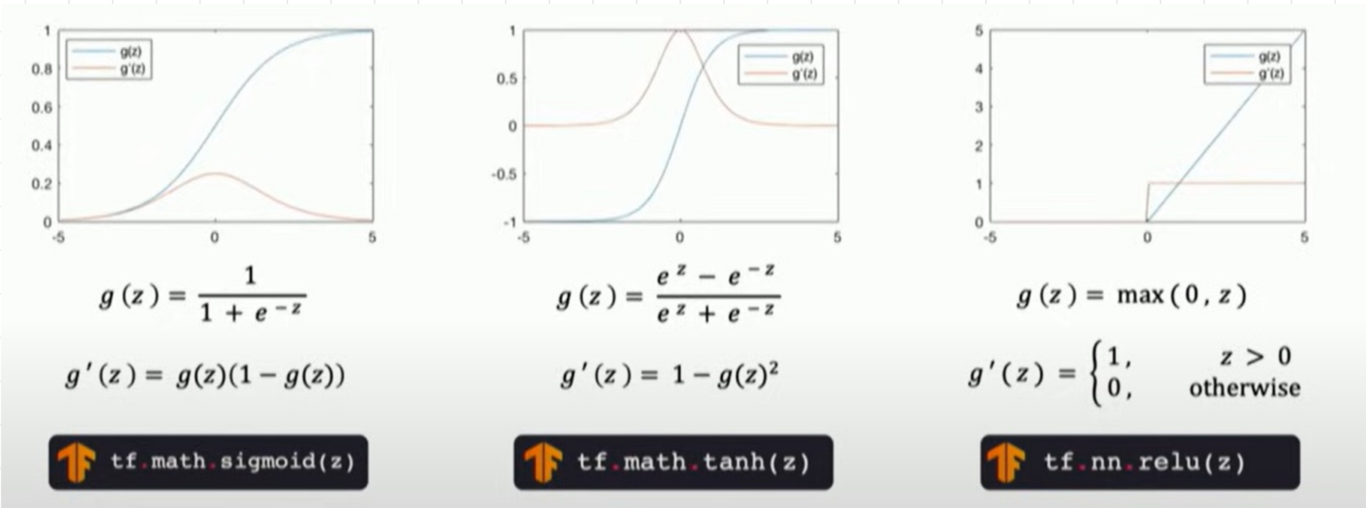
$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$$

↑ Non-linear active function   ↖ Bias

$$\hat{y} = g(w_0 + X^T W)$$

· Nonlinear active function ($z = w^T x + w_0$)

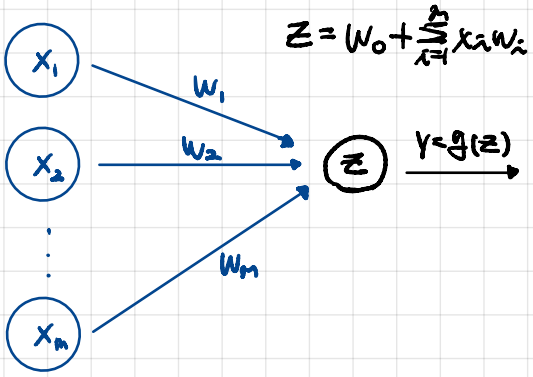
1. Sigmoid function   2. Hyperbolic Tangent   3. Rectified Linear Unit



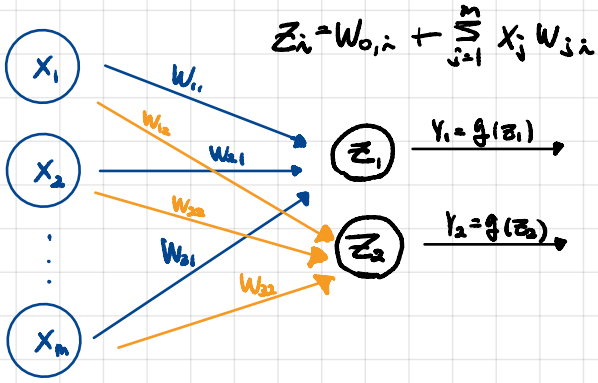
- $0 \le g(x) \le 1$ 이기 때문에 확률 다룰 때 좋음

=> Perceptron을 여러개 사용하면 더욱 복잡한 케이스를 처리할 수 있는 Neural Network를 만들 수 있다.

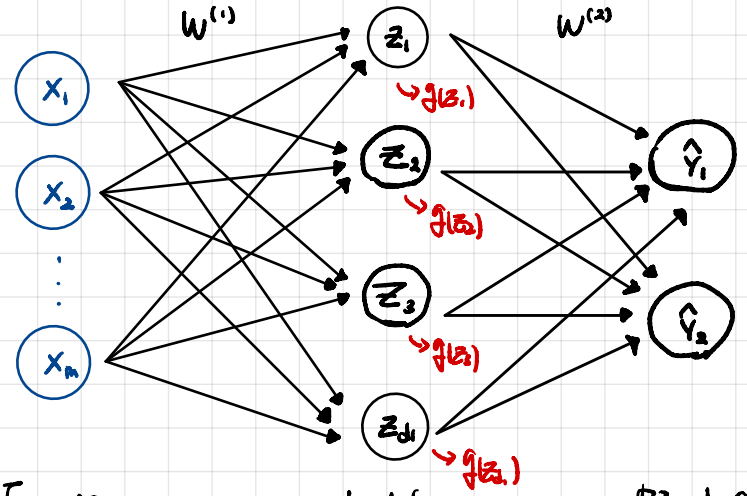
The Perceptron : Simplified



Multi output perception



Single - layer Neural Network



Inputs Hidden Final output

$W_{j,i}$ { i : 도착 / j : 출발

$g(z)$: Activation function

Generalize

$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{n_{k-1}} g(z_{k-1,j}) w_{j,i}^{(k)}$$

$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j w_{j,i}^{(1)} \qquad \hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} g(z_j) w_{j,i}^{(2)}\right)$$

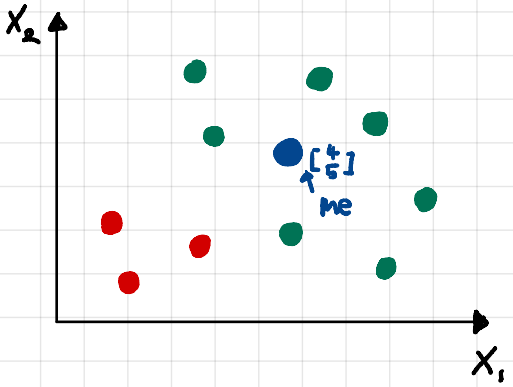
→ Hidden Layer의 Input은 이전 Layer의 Output이 된다.

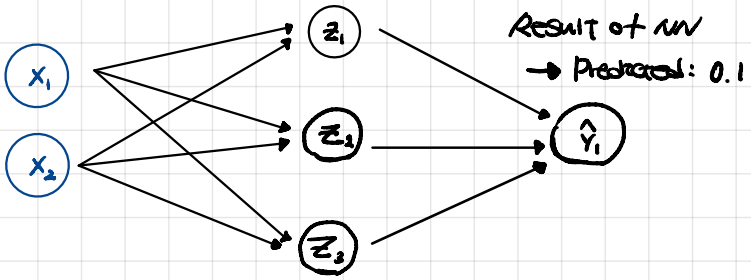
★ 더욱 복잡한 Data를 처리하는 가장 간단한 방법은 Layer를 여러겹으로 만드는 것이다.

Applying Neural Networks - Example

Question : Will I pass this class? [x_1 : # of lectures / x_2 : Hours spend on final project

Data





Q) Data 상으로는 합격해야하는데 Neural Network에서는 합격확률이 10%인 이유?

→ 신경망이 학습되지 않았기 때문! 학습을 시켜야한다.

Quantifying loss

신경망의 예측과 실제 값의 차이, Loss를 통한 학습으로 신경망은 더욱 정확해진다.

· Empirical Loss (cost function, objective function)

$$J(W) = \frac{1}{n}\sum_{i=1}^{n} L\left(\underbrace{f(x^{(i)}; W)}_{\text{Prediction}}, \underbrace{y^{(i)}}_{\text{Actual}}\right)$$

1. Cross Entropy

$$J(W) = -\frac{1}{n}\sum_{i=1}^{n} y^{(i)} \log\left(f(x^{(i)}; W)\right) + (1-y^{(i)}) \log\left(1 - f(x^{(i)}; W)\right)$$

2. Mean Squared Error

$$J(W) = \frac{1}{n}\sum_{i=1}^{n} \left(y^{(i)} - f(x^{(i)}; W)\right)^2$$

Training Neural Networks: Find the lowest loss

$$W^* = \underset{W}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^{n} L\left(f(x^{(i)}; W), y^{(i)}\right) = \underset{W}{\operatorname{argmin}}\, J(W)$$

→ Loss를 최소화하는 W를 찾자

Using Gradient descent

1. Initialize weights randomly $\sim N(0, \sigma^2)$ 로 초기화
2. Compute gradient $\frac{\partial J(W)}{\partial W}$
3. Update weight $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
4. Return weight

← 수렴할 때까지 반복

η: learning rate
→ 모든 weight에서 손실이 수렴할 정도
→ 적절한 learning rate 설정이 중요

Computing Gradient: Back propagation

$$\frac{\partial J(W)}{\partial W_1} = \frac{\partial J(W)}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_1} \cdot \frac{\partial z_1}{\partial W_1} \quad \text{(By Chain rule)}$$

→ Loss를 계산한 후, Loss를 줄이기 위해 신경망 내 모든 가중치에 대해 Gradient descent를 진행하는 방법

How to get Adaptive Learning rate?

1. 다양한 learning rate로 테스트해보고 최적 값을 찾는 방법
2. Learning Rate를 고정하지 않고 gradient의 크기, 신뢰하는 정도 등에 의해 변수가 변동하게 하는 방법
   - SGD, Adam, Adadelta, Adagrad, RMSprop가 같은 Gradient descent algorithm을 사용할 수 있다.

Mini-batches : Tip for training Neural Network

→ 전체 데이터셋에 대해 한 번에 계산하기보다 데이터셋을 쪼개어 계산하자

Stochastic Gradient Descent

1. Initialize weights randomly $\sim N(0, \sigma^2)$를 따르게
2. Pick Batch of B data points
3. Compute gradient $\frac{\partial J(W)}{\partial W} = \frac{1}{B}\sum_{k=1}^{B} \frac{\partial J_k(W)}{\partial W}$
4. Update weight $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
5. Return weight

(2~4 반복)

※ 장점은 즉각
- Smoother Convergence
- Allows for larger learning rates
- Fast training

Problem of Neural Network : Overfitting

↳ Test data보다 Training data를 더 잘 표현하는 경우를 뜻한다.

→ Neural Network의 Data에 대한 의존가 너무 큰 상태
Over parameter, Too complex

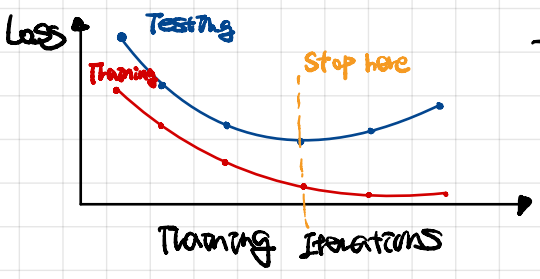
Solution? → Regularization

Improve generalization of model

1. Dropout

- 학습 과정에서 일부 Activation을 0으로 만든다. (일부 node를 제거)
- 보통 50% Dropout

2. Early Stopping



→ 차이가 생기는 부분(overfit)에서 training을 종료하는 방법