

Linear Regression & Neural Network

최건호

01

Linear Regression

- 정의
- Loss
- Gradient Descent
- 문제점

02

Neural Network

- 정의
- Deep?
- 활용

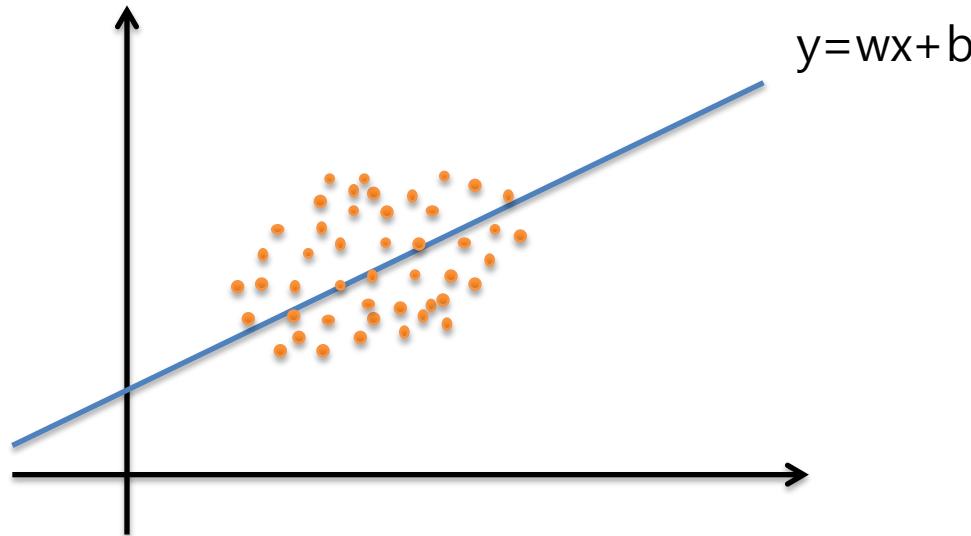
03

Propagation

- 정의
- Forward Prop.
- Back Prop.

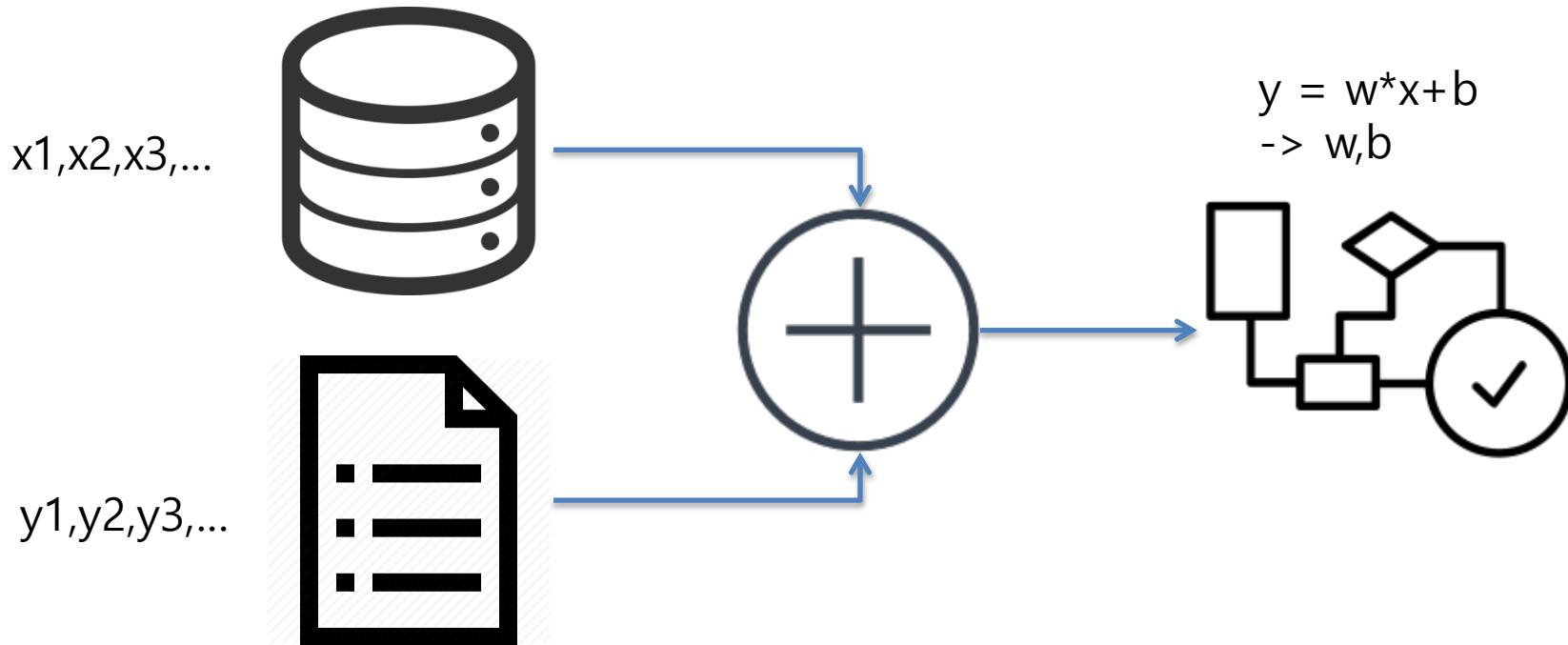
Linear Regression

x에 대한 y값을 가장 잘 설명하는
변수 w, b를 찾는 것

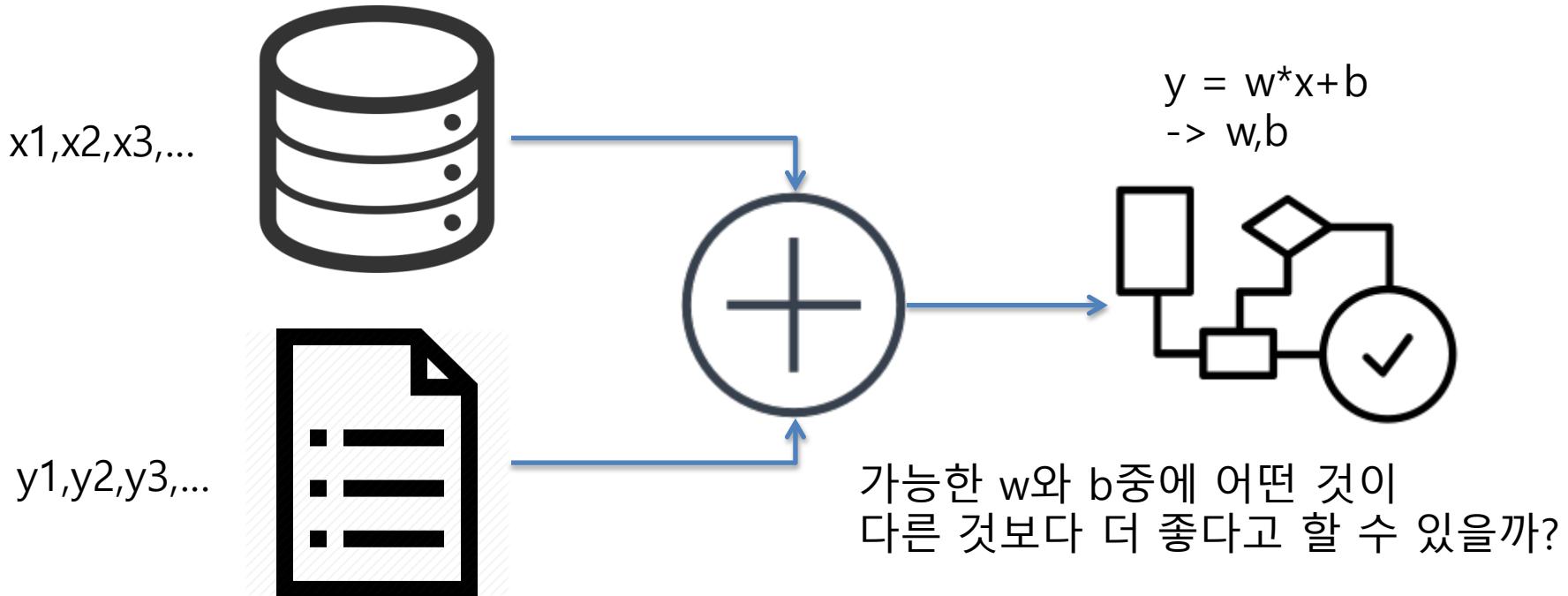


종속 변수 y 와 한 개 이상의 독립 변수 X 와의
선형 상관 관계를 모델링하는 회귀분석 기법
(출처: 위키피디아)

Linear Regression



Linear Regression



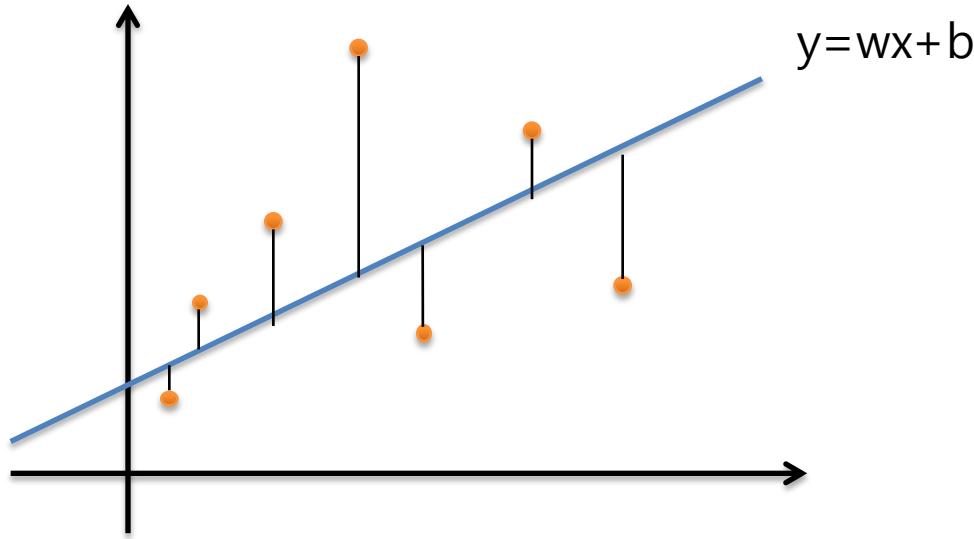
Linear Regression

잘 예측했는지 아닌지 측정할 척도(metric)가 필요함



Linear Regression

잘 예측했는지 아닌지 측정할 척도(metric)가 필요함

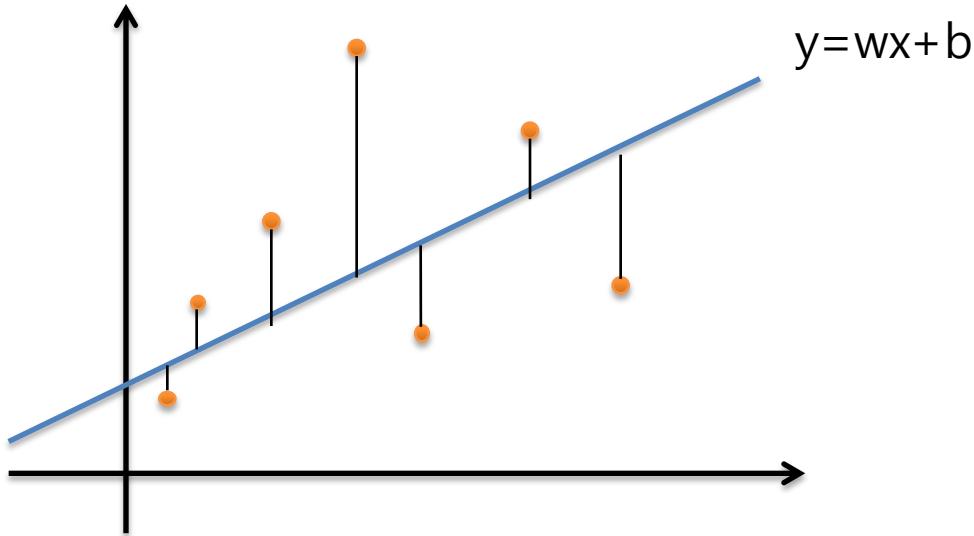


Linear Regression

잘 예측했는지 아닌지 측정할 척도(metric)가 필요함

Mean Squared Error(MSE)
$$MSE = \frac{(x_1 - x_2)^2}{n}$$

두 값의 거리의 제곱의 평균

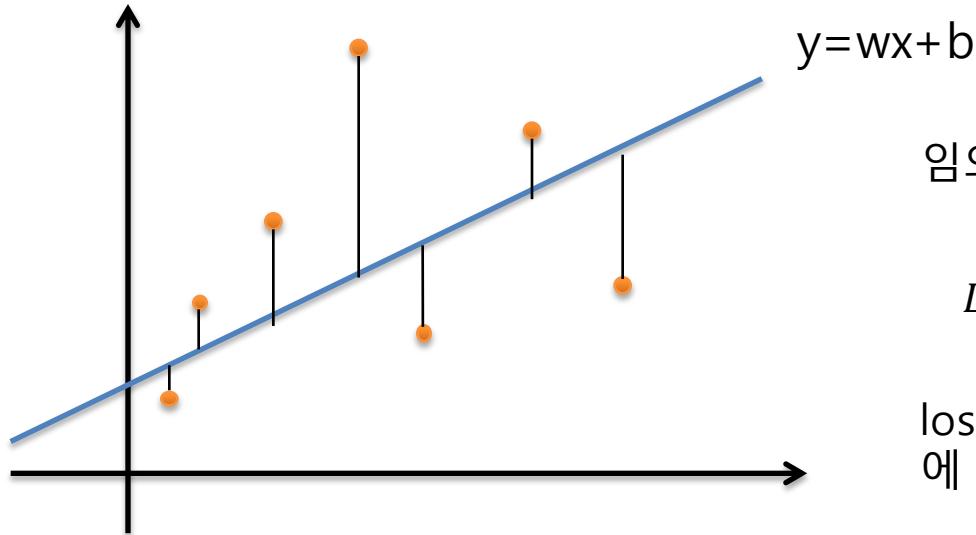


Linear Regression

잘 예측했는지 아닌지 측정할 척도(metric)가 필요함

Mean Squared Error(MSE)
$$MSE = \frac{(x_1 - x_2)^2}{n}$$

두 값의 거리의 제곱의 평균



$$y = wx + b$$

임의의 w^*, b^* 를 초기값으로 한다면

$$Loss = \frac{(y^* - y)^2}{n} = \frac{(w^*x + b^* - y)^2}{n}$$

loss값은 고정된 x, y(데이터)에서 w^*, b^* 에 의해 구해짐

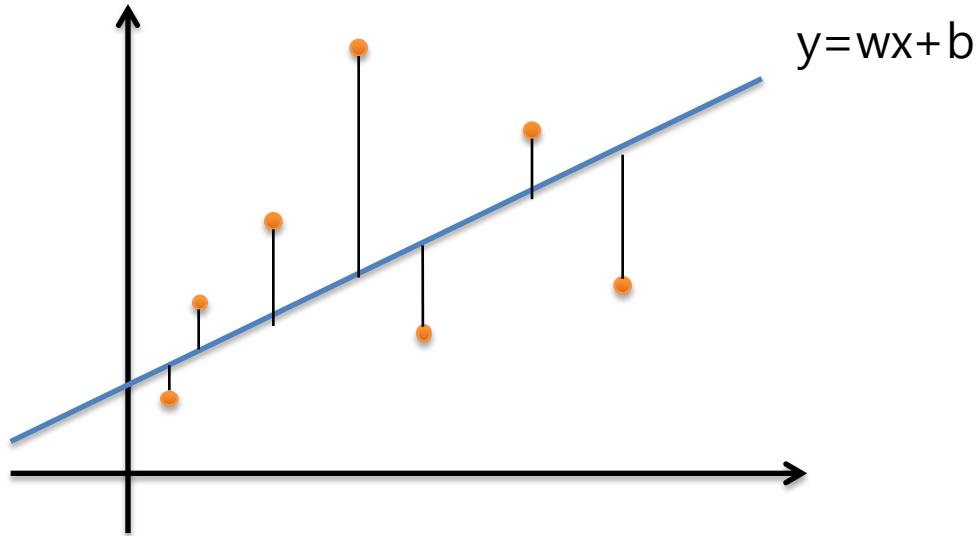
Linear Regression

잘 예측했는지 아닌지 측정할 척도(metric)가 필요함

Mean Squared Error(MSE)

$$MSE = \frac{(x_1 - x_2)^2}{n}$$

두 값의 거리의 제곱의 평균



예시)
 $y = 0.5*x+4$ 이고 $w^*=2$,
 $b^*=2$ 일 때, $x=3$ 에서 loss는?

$$Loss = \frac{(8 - 5.5)^2}{n} = \frac{(2.5)^2}{n}$$

Linear Regression

Loss를 minimize하는 w, b 를 구하고 싶다.



Linear Regression

Loss를 minimize하는 w, b 를 구하고 싶다.

-> Random Search?



Linear Regression

Loss를 minimize하는 w, b 를 구하고 싶다.

-> Random Search? 어느 세월에..

Linear Regression

Loss를 minimize하는 w, b 를 구하고 싶다.

- > Random Search? 어느 세월에..
 - > Loss 값을 통해서 구할 수 없을까?
-

Linear Regression

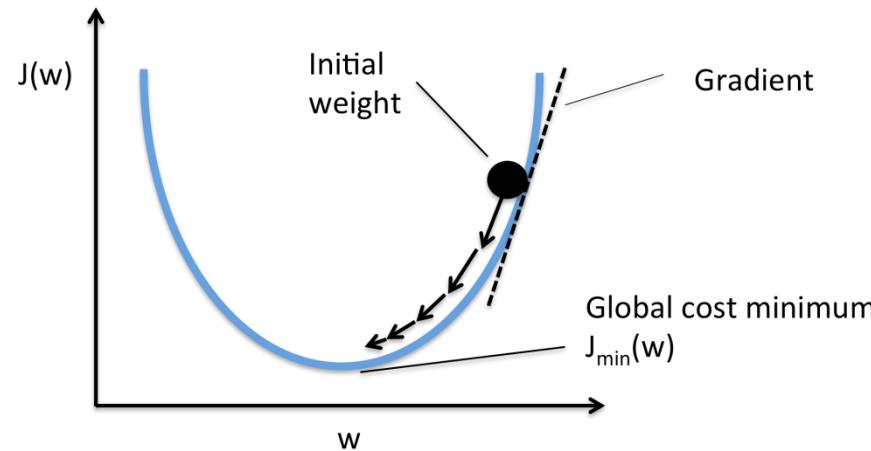
Loss를 minimize하는 w, b 를 구하고 싶다.

- > Random Search? 어느 세월에..
 - > Loss 값을 통해서 구할 수 없을까? Gradient Descent!
-

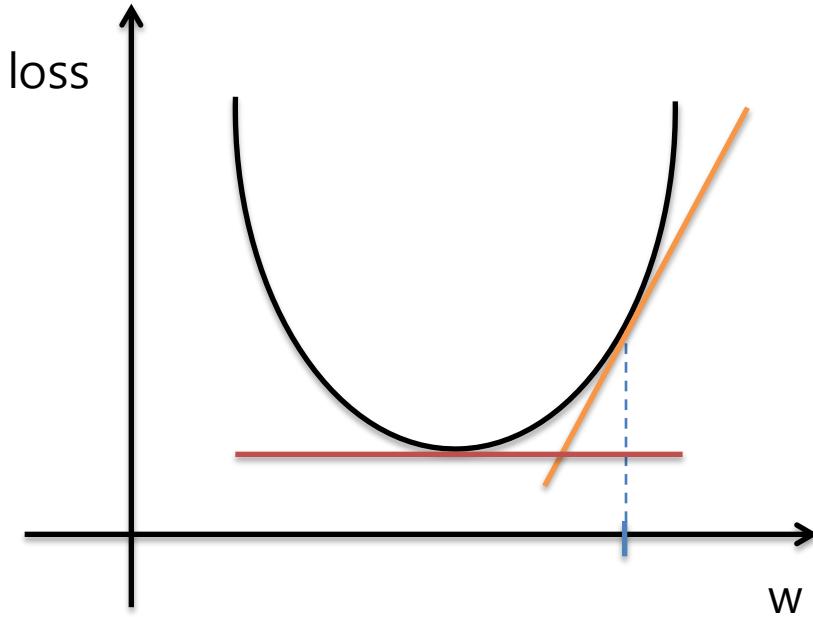
Linear Regression

Loss를 minimize하는 w, b 를 구하고 싶다.

- > Random Search? 어느 세월에..
- > Loss 값을 통해서 구할 수 없을까? Gradient Descent!

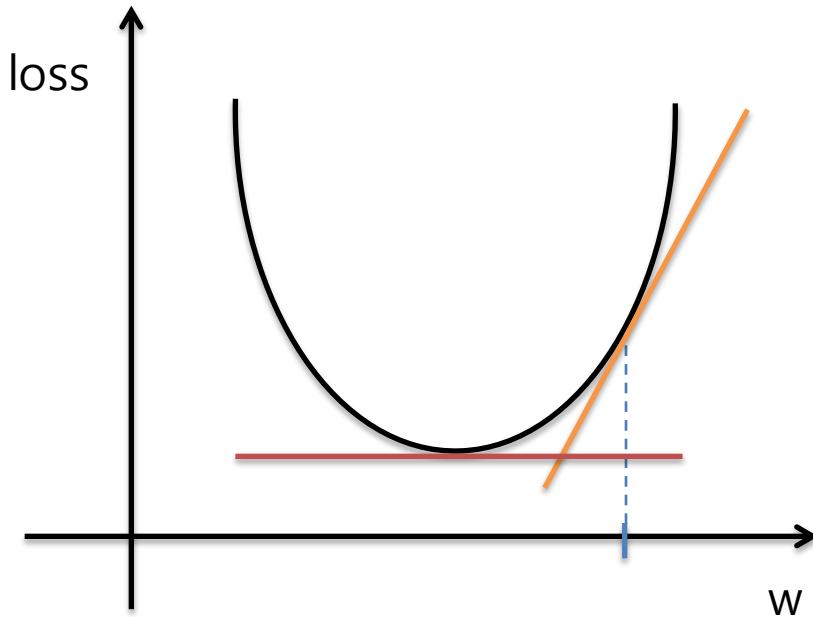


Linear Regression



그냥 바로 loss를 w 에 대해 미분 했을 때 그 값이 0이 되는 w 를 찾으면 안될까?

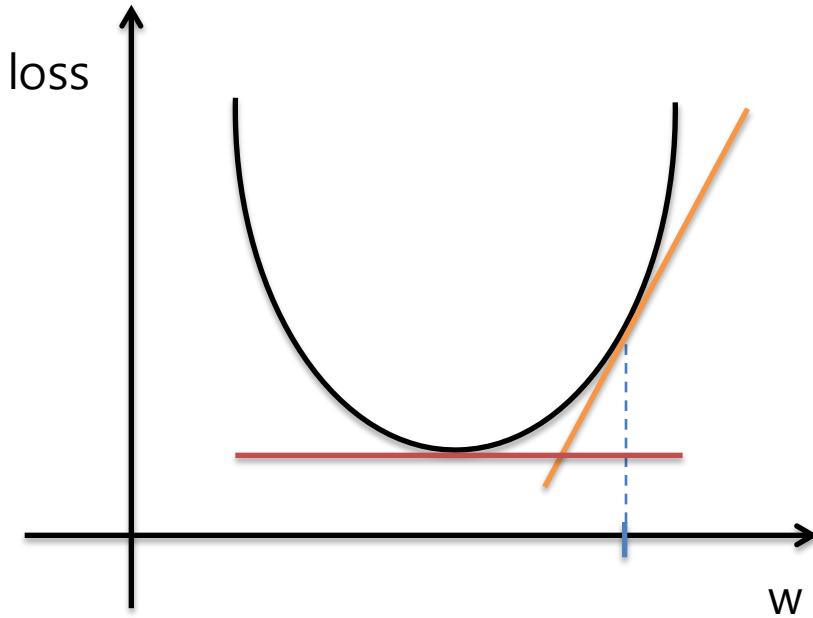
Linear Regression



그냥 바로 loss를 w 에 대해 미분했을 때 그 값이 0이 되는 w 를 찾으면 안될까?

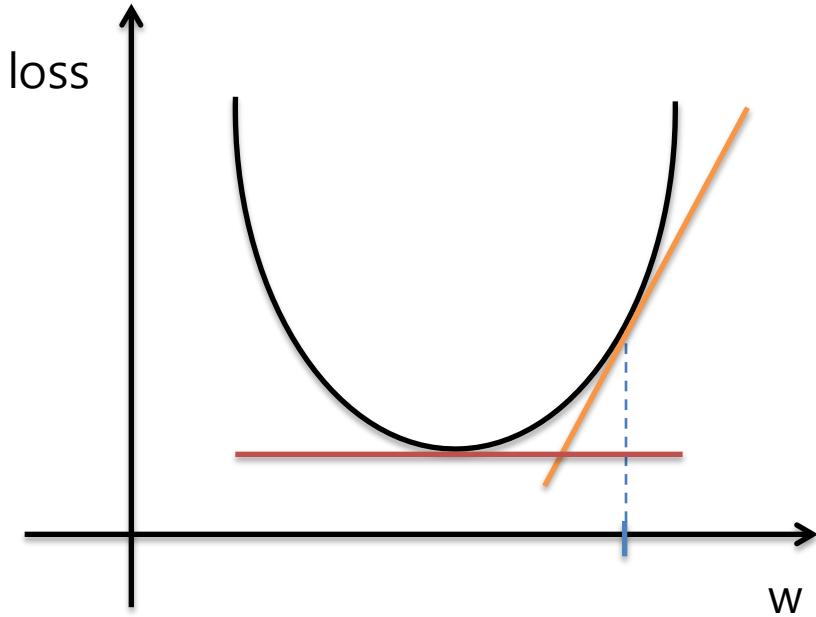
loss를 minimize하는 w 를 구하려면 $w = (x^T x)^{-1} x^T y$ 식을 풀어야 하는데, 변수가 많아지고 matrix 가 커지면 복잡도가 $O(n^3)$ 이기 때문에 exponential하게 증가한다. 변수가 많아질수록 계산이 거의 불가능.

Linear Regression



바로 minimum이 되는 지점을 구하기보다, 현재 loss에 대한 w 의 gradient(경사도)를 구하여 gradient x learning rate만큼 w 를 업데이트

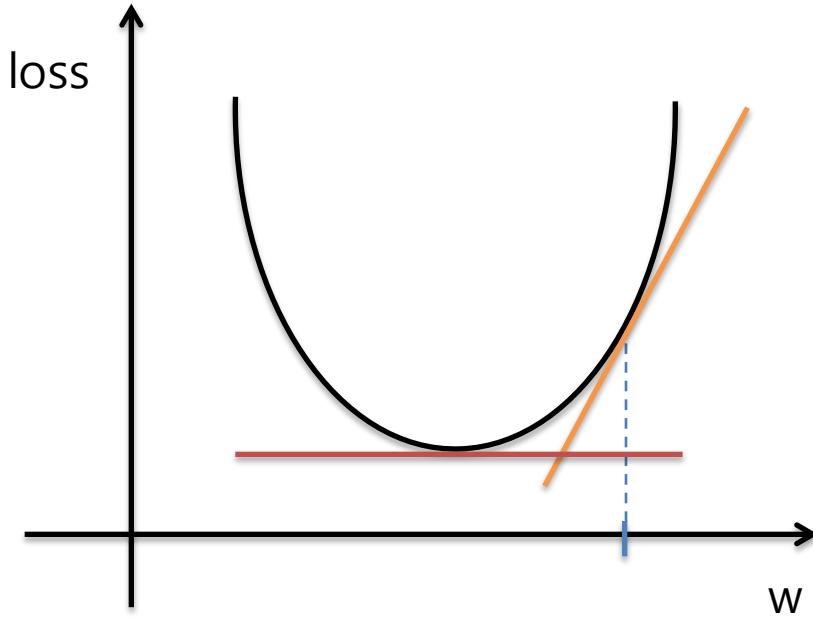
Linear Regression



바로 minimum이 되는 지점을 구하기보다, 현재 loss에 대한 w 의 gradient(경사도)를 구하여 gradient \times learning rate 만큼 w 를 업데이트

$$w_{t+1} = w_t - \text{gradient} * \text{learning}_{\downarrow} \text{rate}$$

Linear Regression

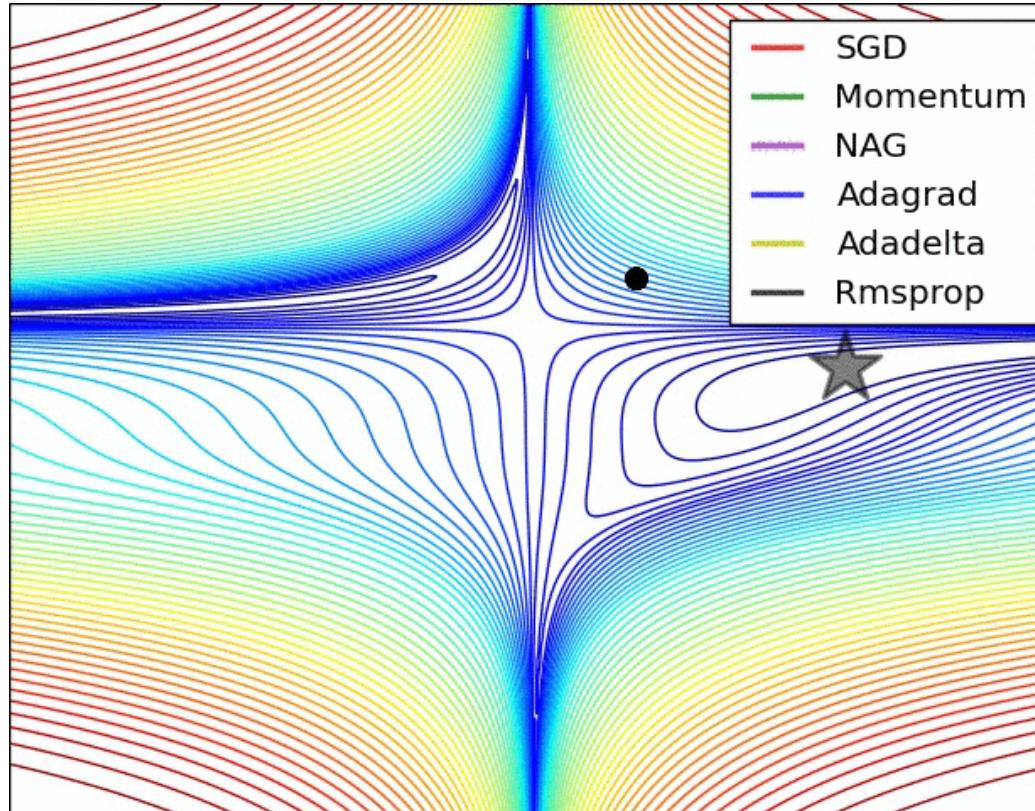


바로 minimum이 되는 지점을 구하기보다, 현재 loss에 대한 w 의 **gradient(경사도)**를 구하여 $\text{gradient} \times \text{learning rate}$ 만큼 w 를 업데이트

$$w_{t+1} = w_t - \text{gradient} * \text{learning}_{\downarrow} \text{rate}$$

계속 업데이트 하다 보면 optimal한 w 에 근접하게 된다.

Linear Regression



Linear Regression

```
◀ ▶ test.py
1 import numpy as np
2 import torch
3 import torch.nn as nn
4 import torch.optim as optim
5 import torch.nn.init as init
6 from torch.autograd import Variable
7
8 from visdom import Visdom
9 viz = Visdom()
10
11
12 num_data = 1000
13 num_epoch = 1000
14
15 noise = init.normal(torch.FloatTensor(num_data,1),std=0.2)
16 x = init.uniform(torch.Tensor(num_data,1),-10,10)
17 y = 2*x+3
18 y_noise = y + noise
```

Linear Regression

필요한 라이브러리



```
test.py
1 import numpy as np
2 import torch
3 import torch.nn as nn
4 import torch.optim as optim
5 import torch.nn.init as init
6 from torch.autograd import Variable
7
8 from visdom import Visdom
9 viz = Visdom()
10
11
12 num_data = 1000
13 num_epoch = 1000
14
15 noise = init.normal(torch.FloatTensor(num_data,1),std=0.2)
16 x = init.uniform(torch.Tensor(num_data,1),-10,10)
17 y = 2*x+3
18 y_noise = y + noise
```

Linear Regression

필요한 라이브러리

```
test.py
1 import numpy as np
2 import torch
3 import torch.nn as nn
4 import torch.optim as optim
5 import torch.nn.init as init
6 from torch.autograd import Variable
7
8 from visdom import Visdom
9 viz = Visdom()
10
11
12 num_data = 1000
13 num_epoch = 1000
14
15 noise = init.normal(torch.FloatTensor(num_data,1),std=0.2)
16 x = init.uniform(torch.Tensor(num_data,1),-10,10)
17 y = 2*x+3
18 y_noise = y + noise
```

데이터 생성

Linear Regression

```
20 model = nn.Linear(1,1)
21
22 loss_func = nn.L1Loss()
23 optimizer = optim.SGD(model.parameters(), lr=0.01)
24
25 # train
26 loss_arr = []
27 label = Variable(y_noise)
28 for i in range(num_epoch):
29     optimizer.zero_grad()
30     output = model(Variable(x))
31
32     loss = loss_func(output,label)
33     loss.backward()
34     optimizer.step()
35     if i % 10 == 0:
36         print(loss)
37     loss_arr.append(loss.data.numpy()[0])
38
39 param_list = list(model.parameters())
40 print(param_list[0].data,param_list[1].data)
```

Linear Regression

linear regression 모델 생성

```
20 model = nn.Linear(1,1)
21
22 loss_func = nn.L1Loss()
23 optimizer = optim.SGD(model.parameters(), lr=0.01)
24
25 # train
26 loss_arr = []
27 label = Variable(y_noise)
28 for i in range(num_epoch):
29     optimizer.zero_grad()
30     output = model(Variable(x))
31
32     loss = loss_func(output,label)
33     loss.backward()
34     optimizer.step()
35     if i % 10 == 0:
36         print(loss)
37     loss_arr.append(loss.data.numpy()[0])
38
39 param_list = list(model.parameters())
40 print(param_list[0].data,param_list[1].data)
```

Linear Regression

linear regression 모델 생성

loss function 및
gradient descent
optimizer 생성

```
20 model = nn.Linear(1,1)
21
22 loss_func = nn.L1Loss()
23 optimizer = optim.SGD(model.parameters(), lr=0.01)
24
25 # train
26 loss_arr = []
27 label = Variable(y_noise)
28 for i in range(num_epoch):
29     optimizer.zero_grad()
30     output = model(Variable(x))
31
32     loss = loss_func(output,label)
33     loss.backward()
34     optimizer.step()
35     if i % 10 == 0:
36         print(loss)
37     loss_arr.append(loss.data.numpy()[0])
38
39 param_list = list(model.parameters())
40 print(param_list[0].data,param_list[1].data)
```

Linear Regression

linear regression 모델 생성

loss function 및
gradient descent
optimizer 생성

- <training 단계>
1. 모델로 결과값 추정
 2. loss 및 gradient 계산
 3. 모델 업데이트

```
20 model = nn.Linear(1,1)
21
22 loss_func = nn.L1Loss()
23 optimizer = optim.SGD(model.parameters(), lr=0.01)
24
25 # train
26 loss_arr = []
27 label = Variable(y_noise)
28 for i in range(num_epoch):
29     optimizer.zero_grad()
30     output = model(Variable(x))
31
32     loss = loss_func(output,label)
33     loss.backward()
34     optimizer.step()
35     if i % 10 == 0:
36         print(loss)
37     loss_arr.append(loss.data.numpy()[0])
38
39 param_list = list(model.parameters())
40 print(param_list[0].data,param_list[1].data)
```

Linear Regression

linear regression 모델 생성

loss function 및
gradient descent
optimizer 생성

- <training 단계>
1. 모델로 결과값 추정
 2. loss 및 gradient 계산
 3. 모델 업데이트

training 이후 파라미터 값 확인

```
20 model = nn.Linear(1,1)
21
22 loss_func = nn.L1Loss()
23 optimizer = optim.SGD(model.parameters(), lr=0.01)
24
25 # train
26 loss_arr = []
27 label = Variable(y_noise)
28 for i in range(num_epoch):
29     optimizer.zero_grad()
30     output = model(Variable(x))
31
32     loss = loss_func(output,label)
33     loss.backward()
34     optimizer.step()
35     if i % 10 == 0:
36         print(loss)
37     loss_arr.append(loss.data.numpy()[0])
38
39 param_list = list(model.parameters())
40 print(param_list[0].data,param_list[1].data)
```

Linear Regression

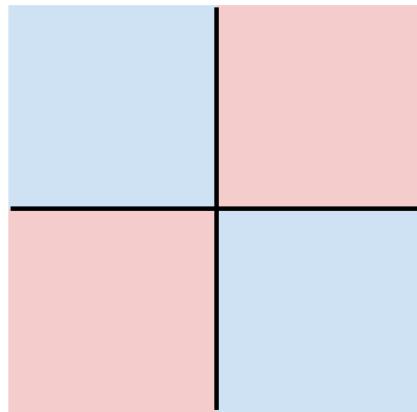
Hard cases for a linear classifier

Class 1:

number of pixels > 0 odd

Class 2:

number of pixels > 0 even

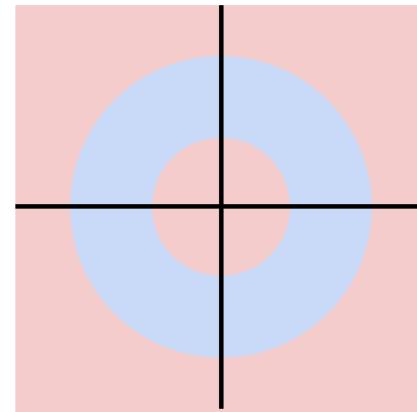


Class 1:

$1 \leq \text{L2 norm} \leq 2$

Class 2:

Everything else

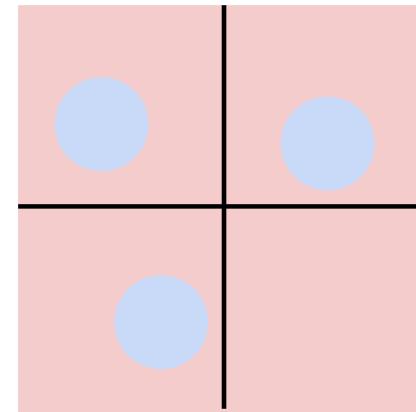


Class 1:

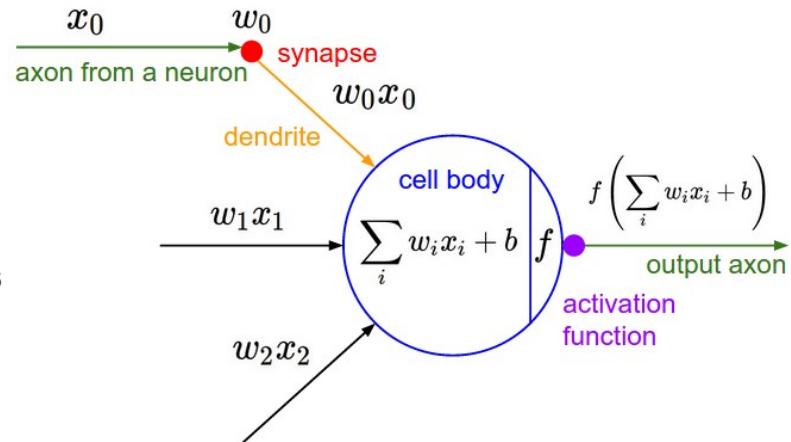
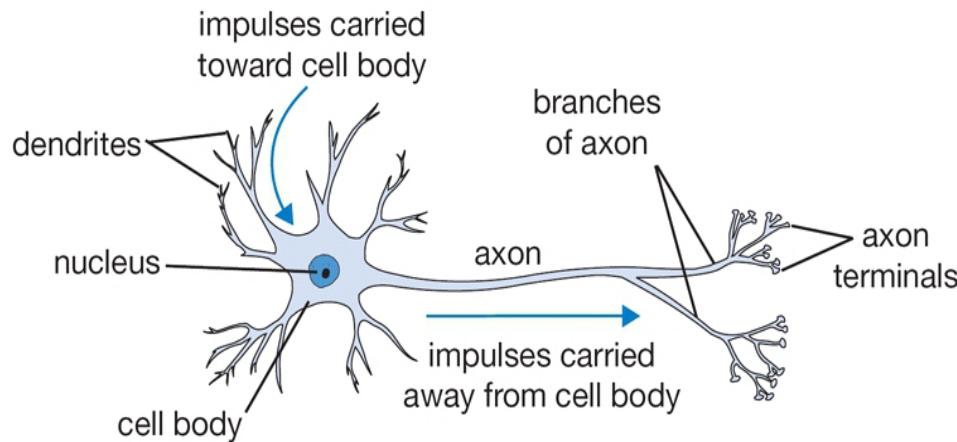
Three modes

Class 2:

Everything else



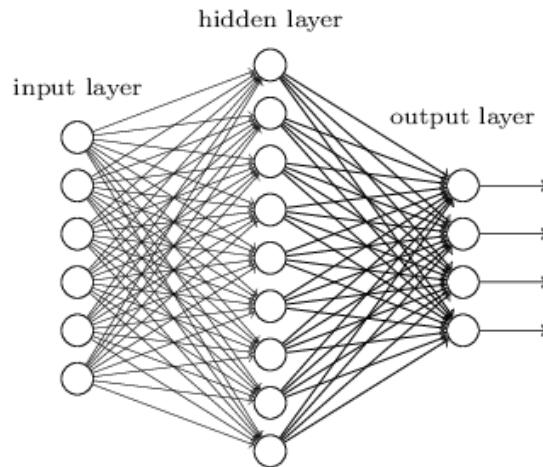
Neural Network



여러 자극이 들어오고 일정 기준을 넘으면 이를 다른 뉴런에 전달하는 구조

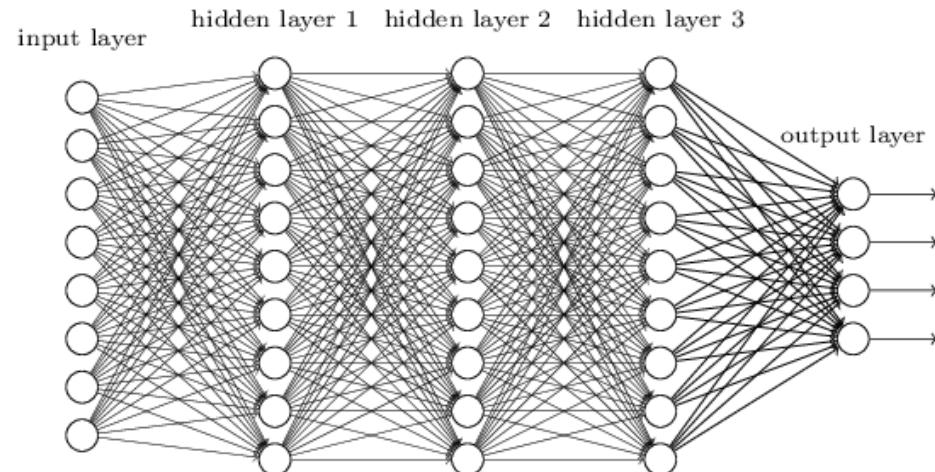
Neural Network

"Non-deep" feedforward neural network



$$y = w2 \left(\text{act}(w1 * \text{input} + b1) \right) + b2$$

Deep neural network



$$y = w4 \left(\text{act} \left(w3 \left(\text{act} \left(w2 \left(\text{act} \left(w1 * \text{input} + b1 \right) \right) + b2 \right) \right) + b3 \right) + b4 \right)$$

Neural Network

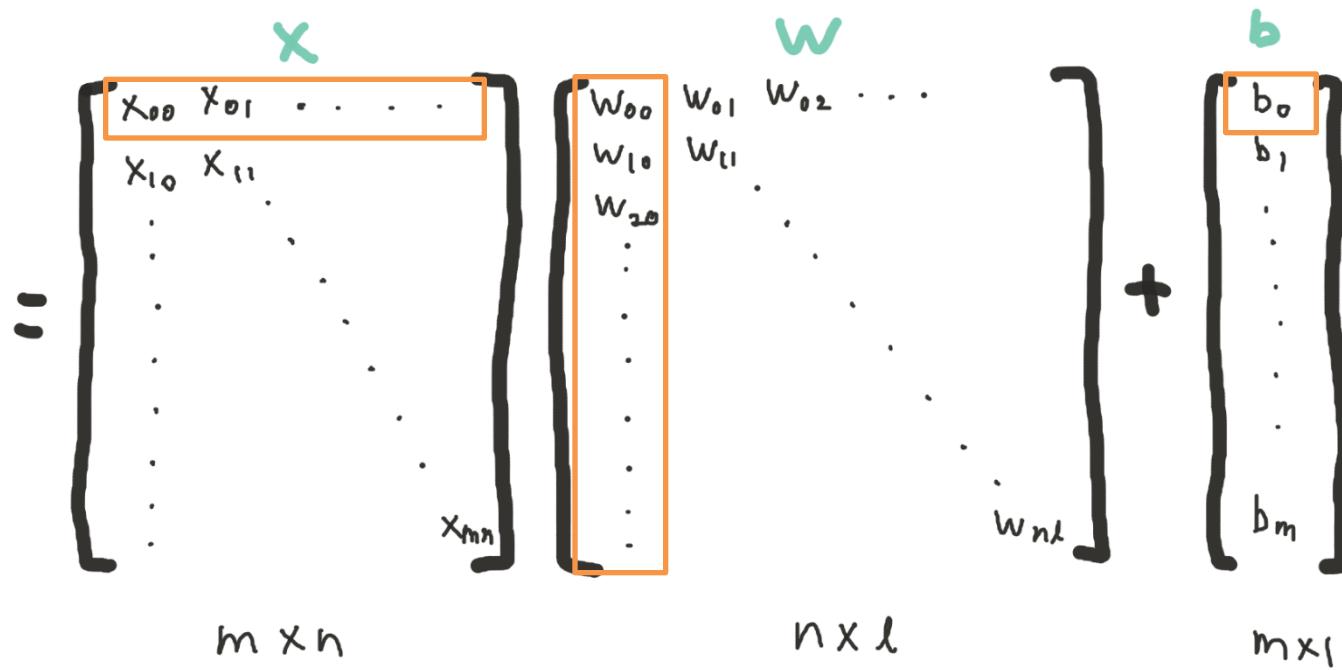
$$Y = W \cdot x + b$$

$$= \begin{bmatrix} x_{00} & x_{01} & \dots & \dots \\ x_{10} & x_{11} & \ddots & \\ \vdots & & \ddots & \\ \vdots & & & \\ \vdots & & & \\ \vdots & & & \\ x_{mn} & & & \end{bmatrix} \begin{bmatrix} w_{00} & w_{01} & w_{02} & \dots \\ w_{10} & w_{11} & \ddots & \\ \vdots & & \ddots & \\ \vdots & & & \\ \vdots & & & \\ \vdots & & & \\ w_{nl} & & & \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ \vdots \\ \vdots \\ b_m \end{bmatrix}$$

$m \times n \qquad n \times l \qquad m \times 1$

Neural Network

$$Y = W \cdot x + b$$



Neural Network

$$y = \text{act}(wx+b)$$

 activation

$$\left(\begin{bmatrix} w x + b \end{bmatrix} \right)_{m \times 1}$$

Neural Network

만약 activation function이 없다면 아래의 식은 결국 linear function.

$$y = w4(act(w3(act(w2(act(w1 * input + b1)) + b2)) + b3)) + b4$$

Neural Network

만약 activation function이 없다면 아래의 식은 결국 linear function.

$$y = w4(act(w3(act(w2(act(w1 * input + b1)) + b2)) + b3)) + b4$$

activation function으로 non-linearity를 추가해야 함

Neural Network

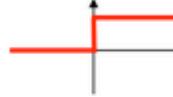
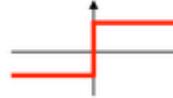
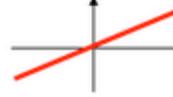
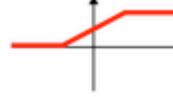
만약 activation function이 없다면 아래의 식은 결국 linear function.

$$y = w4(act(w3(act(w2(act(w1 * input + b1)) + b2)) + b3)) + b4$$

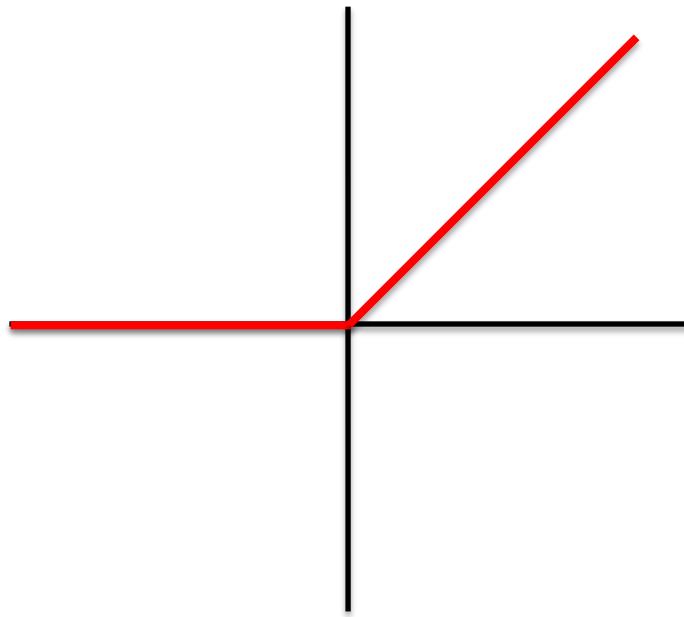
activation function으로 non-linearity를 추가해야 함

그렇다면 어떤 activation function을 써야 할까?

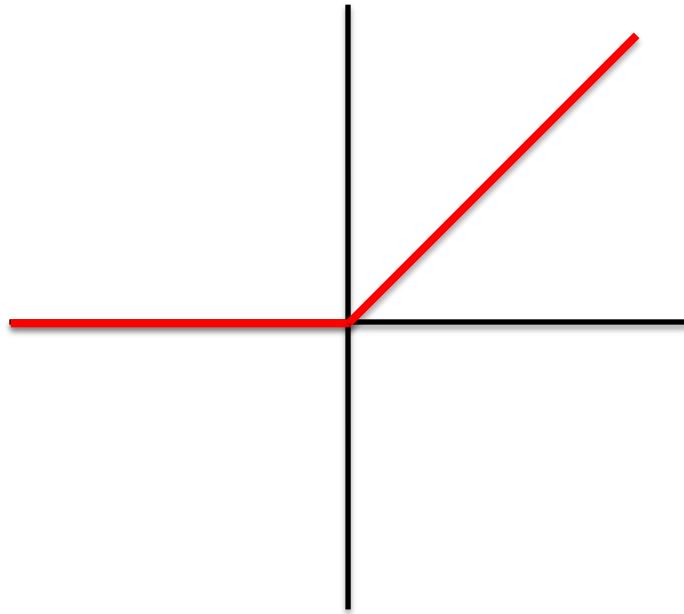
Neural Network

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer NN	

Neural Network

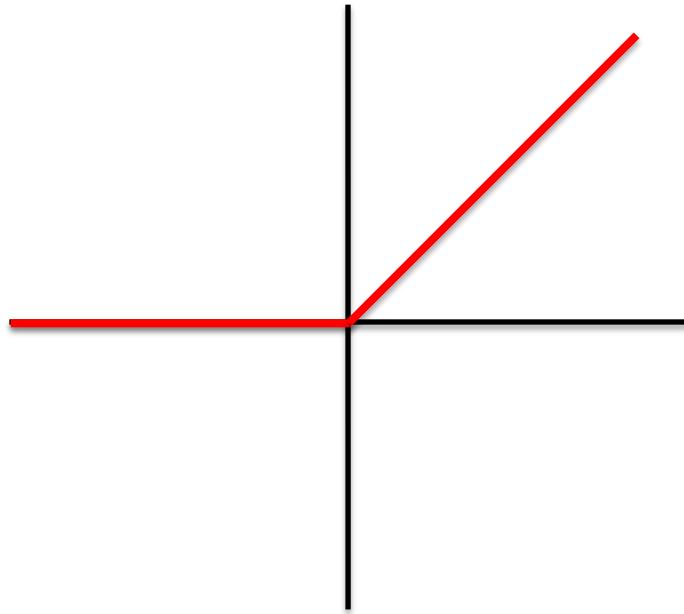


Neural Network



Rectified Linear Unit (ReLU)

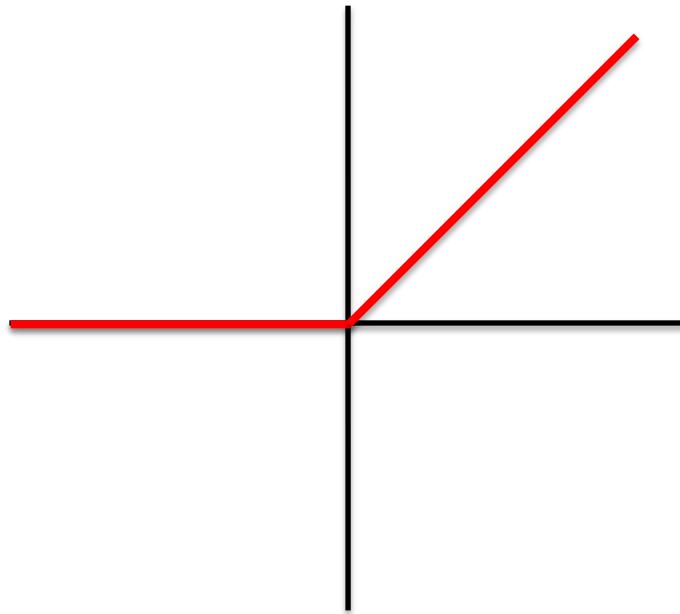
Neural Network



Rectified Linear Unit (ReLU)

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

Neural Network

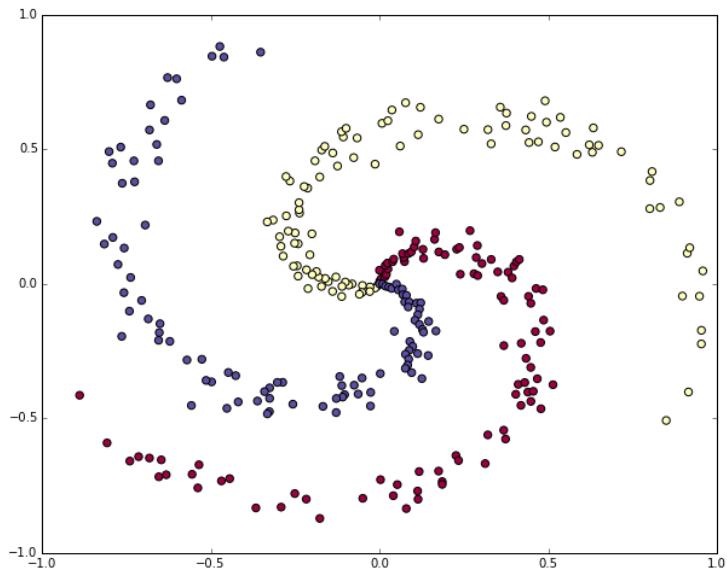


Rectified Linear Unit (ReLU)

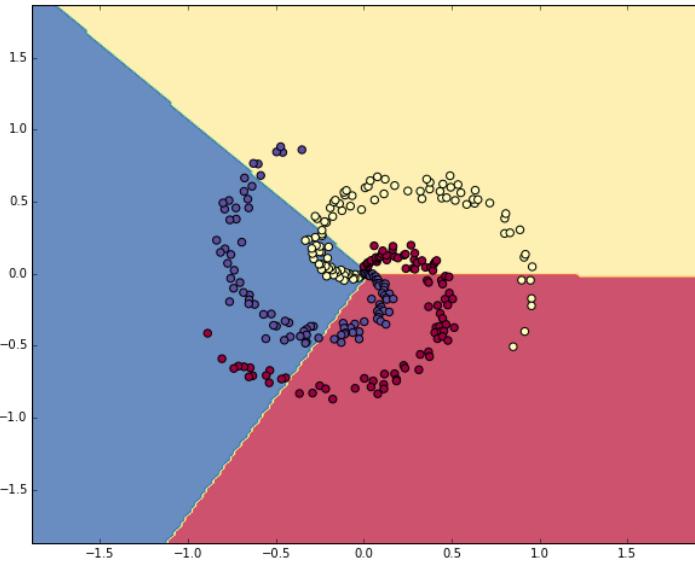
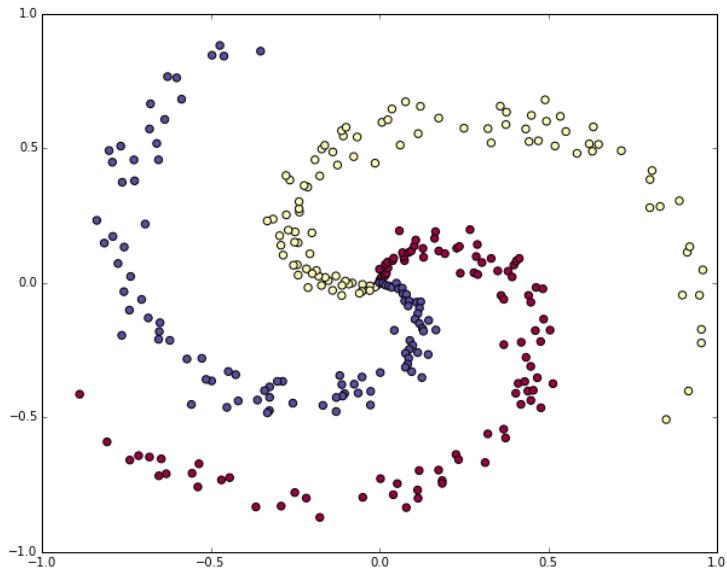
$$f(x) = \max(0, x)$$

기존의 sigmoid와 tanh로는 학습이 잘 안됐었는데 relu는 gradient의 전달이 좋아서 default로 사용되고 있음

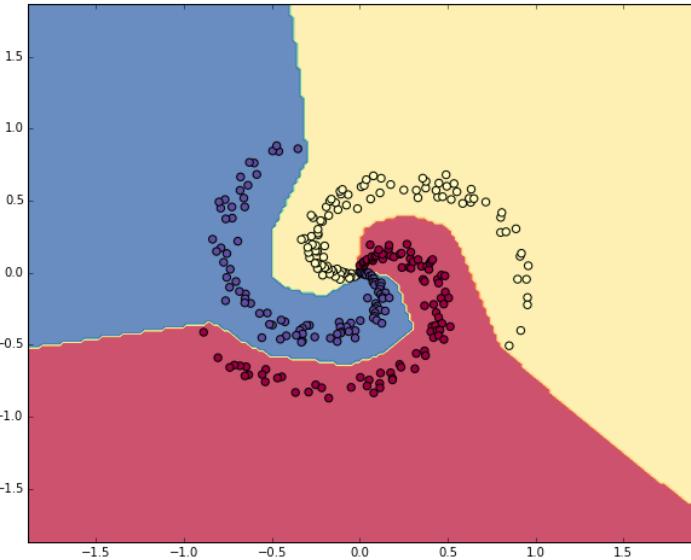
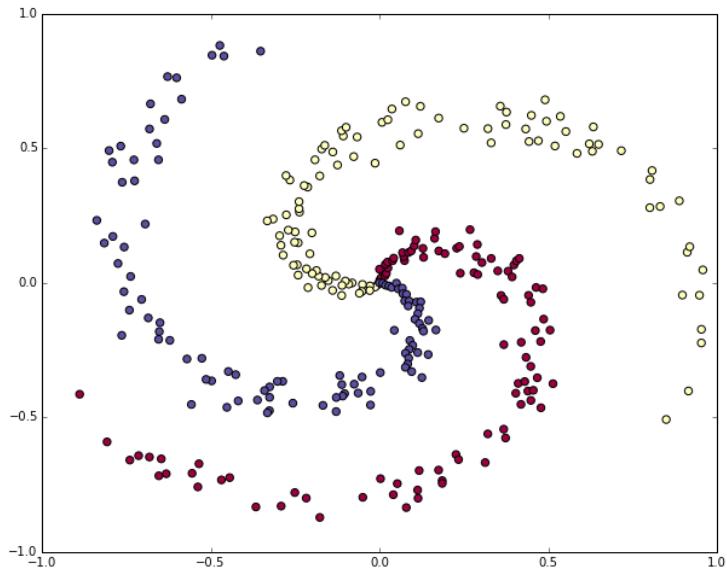
Neural Network



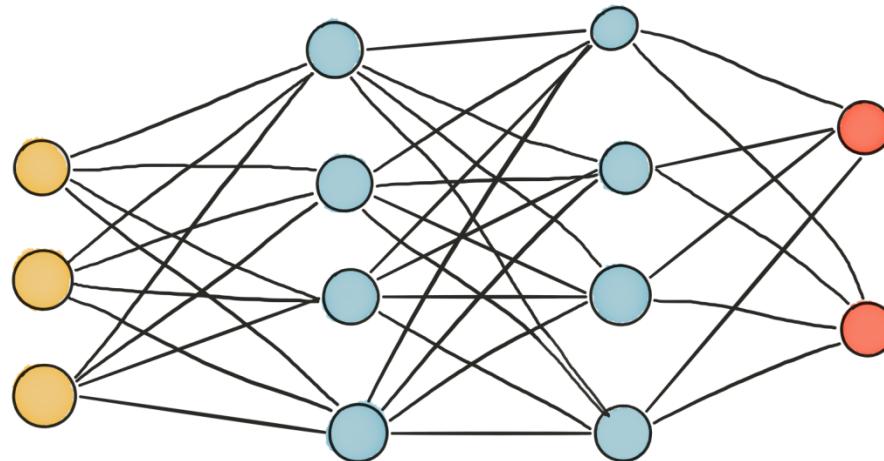
Neural Network



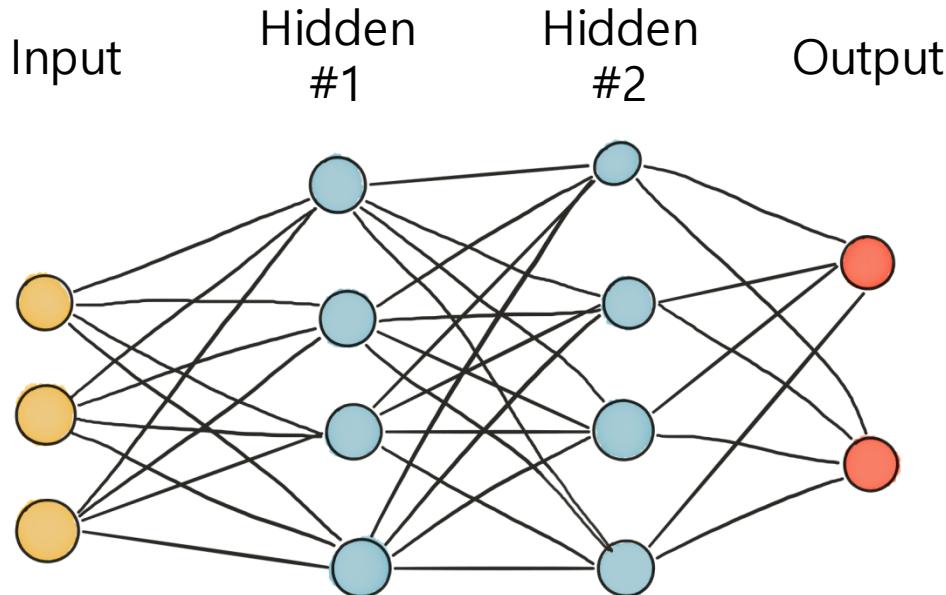
Neural Network



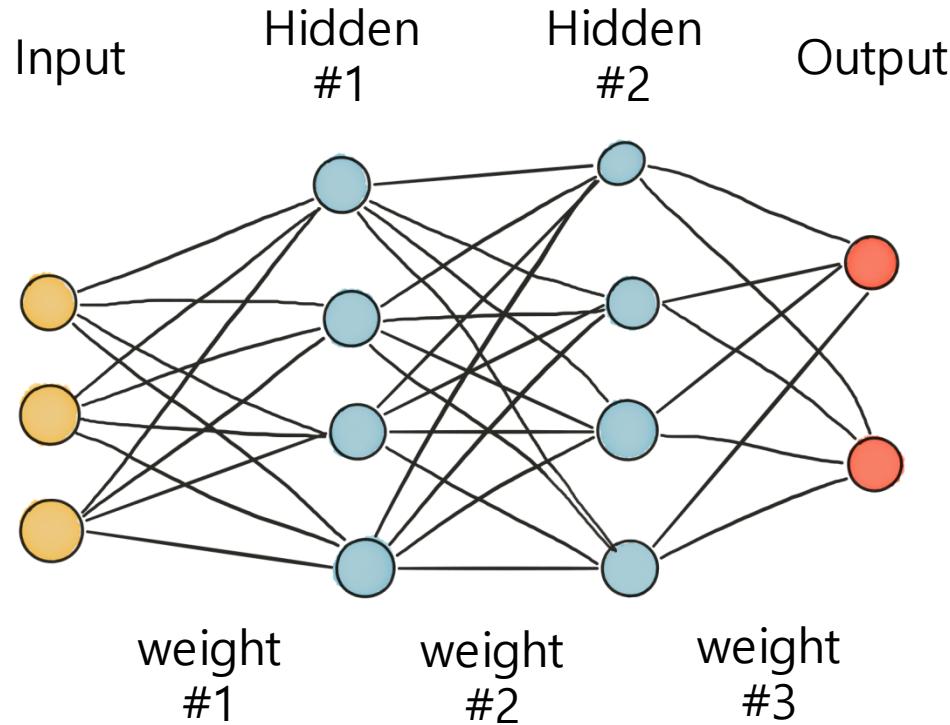
Forward & Back Prop.



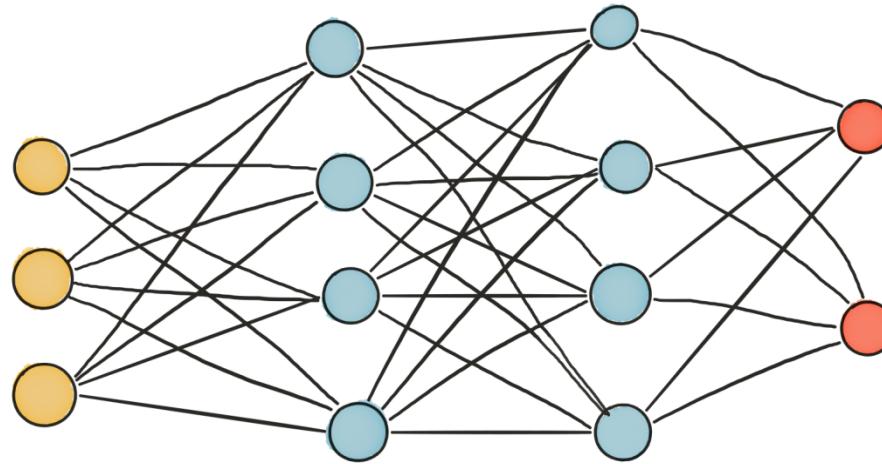
Forward & Back Prop.



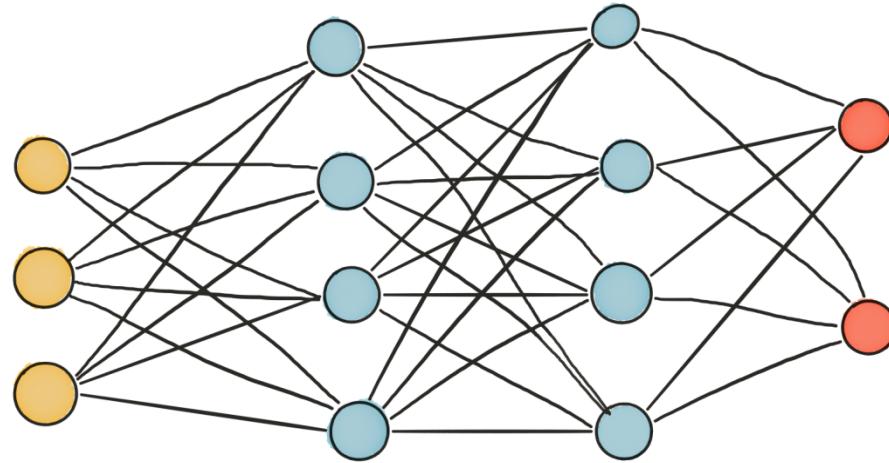
Forward & Back Prop.



Forward & Back Prop.

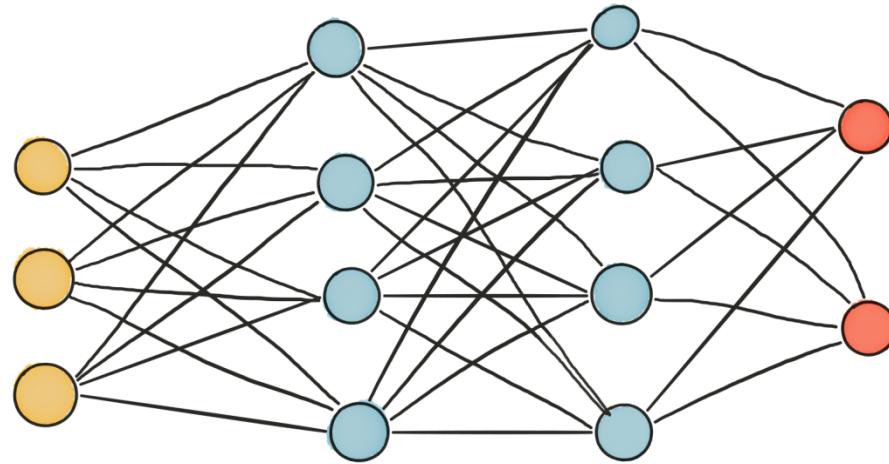


Forward & Back Prop.



$$\begin{bmatrix} w_{o0} & w_{01} & w_{02} & w_{03} \\ w_{10} & w_{11} & w_{12} & w_{13} \\ w_{20} & w_{21} & w_{22} & w_{23} \end{bmatrix} \times \begin{bmatrix} w_{o0} & w_{01} & w_{02} & w_{03} \\ w_{10} & w_{11} & w_{12} & w_{13} \\ w_{20} & w_{21} & w_{22} & w_{23} \\ w_{30} & w_{31} & w_{32} & w_{33} \end{bmatrix} \times \begin{bmatrix} w_{00} & w_{01} \\ w_{10} & w_{11} \\ w_{20} & w_{21} \\ w_{30} & w_{31} \end{bmatrix}$$

Forward & Back Prop.



$$\begin{bmatrix} w_{o0} & w_{01} & w_{02} & w_{03} \\ w_{10} & w_{11} & w_{12} & w_{13} \\ w_{20} & w_{21} & w_{22} & w_{23} \end{bmatrix} \times \begin{bmatrix} w_{o0} & w_{01} & w_{02} & w_{03} \\ w_{10} & w_{11} & w_{12} & w_{13} \\ w_{20} & w_{21} & w_{22} & w_{23} \\ w_{30} & w_{31} & w_{32} & w_{33} \end{bmatrix} \times \begin{bmatrix} w_{00} & w_{01} \\ w_{10} & w_{11} \\ w_{20} & w_{21} \\ w_{30} & w_{31} \end{bmatrix}$$

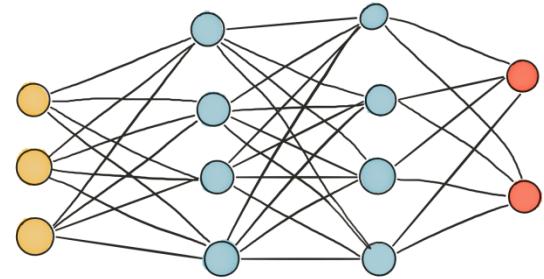
3x4

4x4

4x2

Forward & Back Prop.

$$y^* = w3 * \text{sig}(w2 * \text{sig}(w1 * x + b1) + b2) + b3$$

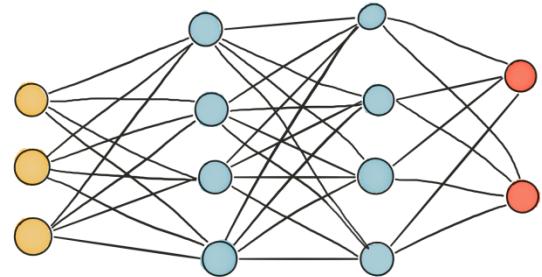


쉽게 이해되도록
loss = 예측값 - 실제로 설정

Forward & Back Prop.

$$y^* = w3 * \text{sig}(w2 * \text{sig}(w1 * x + b1) + b2) + b3$$

$$\begin{aligned} \text{loss} &= y^* - y \\ &= w3 * \text{sig}(w2 * \text{sig}(w1 * x + b1) + b2) + b3 - y \end{aligned}$$



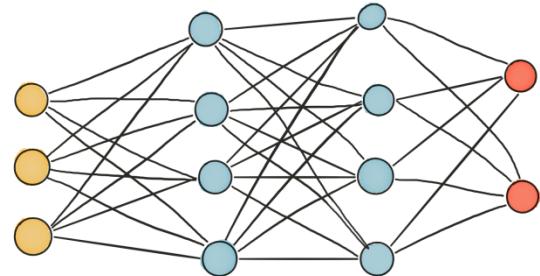
쉽게 이해되도록
loss = 예측값-실제로 설정

Forward & Back Prop.

$$y^* = w3 * \text{sig}(w2 * \text{sig}(w1 * x + b1) + b2) + b3$$

$$\begin{aligned} \text{loss} &= y^* - y \\ &= w3 * \text{sig}(w2 * \text{sig}(w1 * x + b1) + b2) + b3 - y \end{aligned}$$

$$\frac{\partial \text{loss}}{\partial w3} = \text{sig}(w2 * \text{sig}(w1 * x + b1) + b2)$$



쉽게 이해되도록
loss = 예측값-실제로 설정

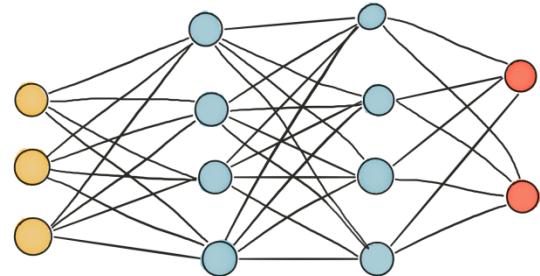
Forward & Back Prop.

$$y^* = w3 * \text{sig}(w2 * \text{sig}(w1 * x + b1) + b2) + b3$$

$$\begin{aligned} \text{loss} &= y^* - y \\ &= w3 * \text{sig}(w2 * \text{sig}(w1 * x + b1) + b2) + b3 - y \end{aligned}$$

$$\frac{\partial \text{loss}}{\partial w3} = \text{sig}(w2 * \text{sig}(w1 * x + b1) + b2)$$

$$\frac{\partial \text{loss}}{\partial b3} = 1$$



쉽게 이해되도록
loss = 예측값-실제로 설정

Forward & Back Prop.

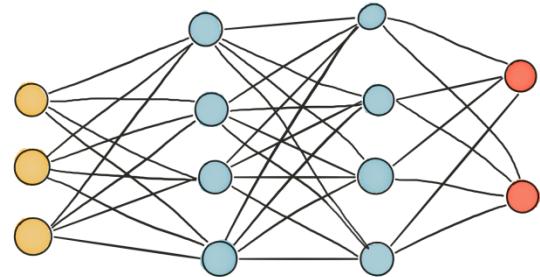
$$y^* = w3 * \text{sig}(w2 * \text{sig}(w1 * x + b1) + b2) + b3$$

$$\begin{aligned} \text{loss} &= y^* - y \\ &= w3 * \text{sig}(w2 * \text{sig}(w1 * x + b1) + b2) + b3 - y \end{aligned}$$

$$\frac{\partial \text{loss}}{\partial w3} = \text{sig}(w2 * \text{sig}(w1 * x + b1) + b2)$$

$$\frac{\partial \text{loss}}{\partial b3} = 1$$

$$\frac{\partial \text{loss}}{\partial w2} = ??$$



쉽게 이해되도록
loss = 예측값-실제로 설정

Forward & Back Prop.

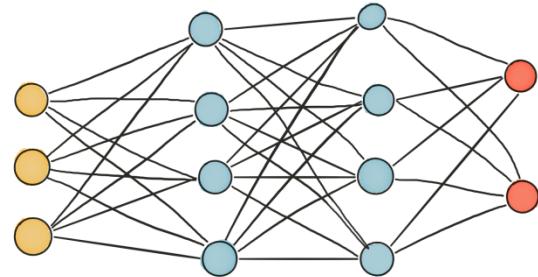
$$y^* = w3 * \text{sig}(w2 * \text{sig}(w1 * x + b1) + b2) + b3$$

$$\begin{aligned} \text{loss} &= y^* - y \\ &= w3 * \text{sig}(w2 * \text{sig}(w1 * x + b1) + b2) + b3 - y \end{aligned}$$

$$\frac{\partial \text{loss}}{\partial w3} = \text{sig}(w2 * \text{sig}(w1 * x + b1) + b2)$$

$$\frac{\partial \text{loss}}{\partial b3} = 1$$

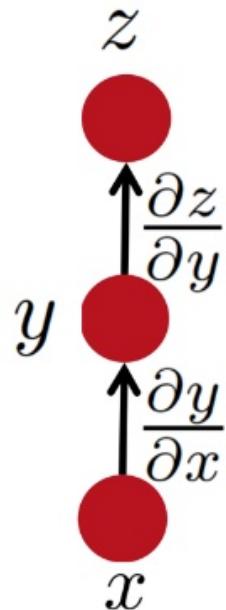
$$\frac{\partial \text{loss}}{\partial w2} = \text{chain rule}!!$$



쉽게 이해되도록
loss = 예측값-실제로 설정

Forward & Back Prop.

Simple Chain Rule



$$\Delta z = \frac{\partial z}{\partial y} \Delta y$$

$$\Delta y = \frac{\partial y}{\partial x} \Delta x$$

$$\Delta z = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \Delta x$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$$

Forward & Back Prop.

Backpropagation: a simple example

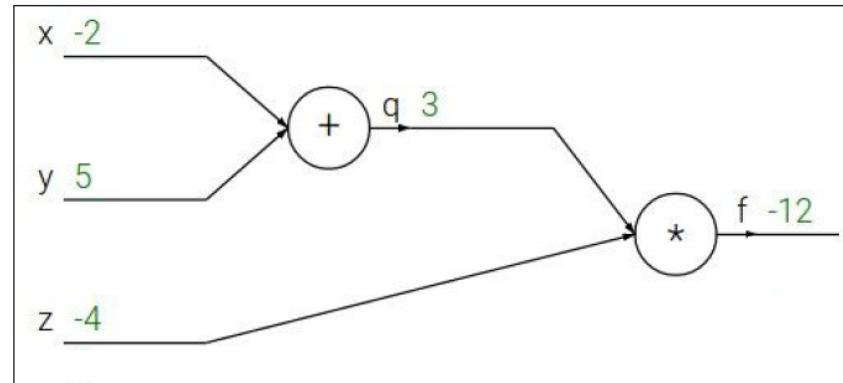
$$f(x, y, z) = (x + y)z$$

e.g. $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Forward & Back Prop.

Backpropagation: a simple example

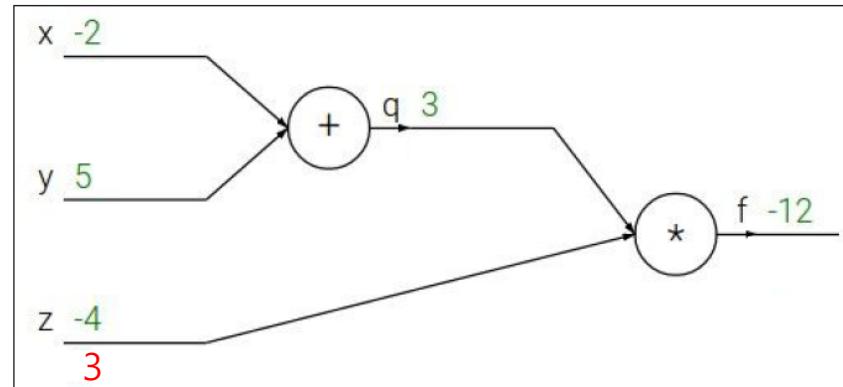
$$f(x, y, z) = (x + y)z$$

e.g. $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial z} = q = x + y = -2 + 5 = 3$$

Forward & Back Prop.

Backpropagation: a simple example

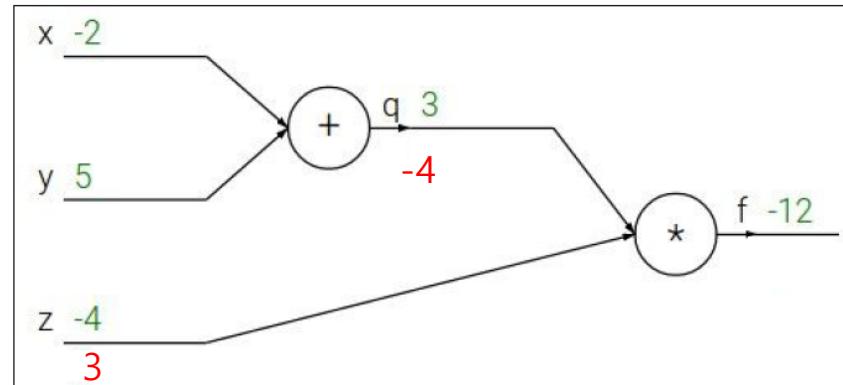
$$f(x, y, z) = (x + y)z$$

e.g. $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial z} = q = x + y = -2 + 5 = 3$$

$$\frac{\partial f}{\partial q} = z = -4$$

Forward & Back Prop.

Backpropagation: a simple example

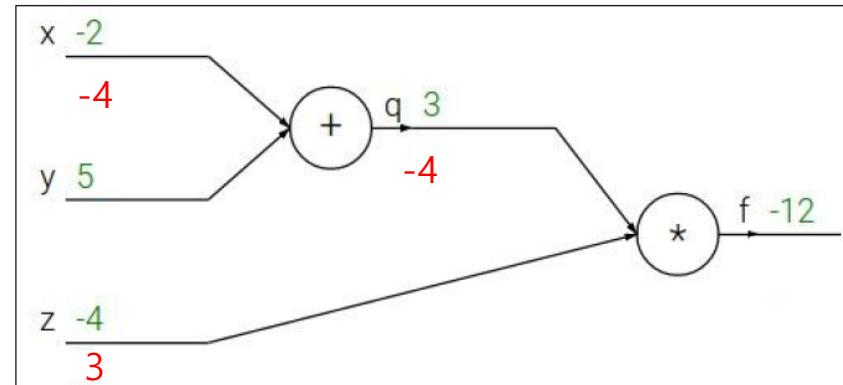
$$f(x, y, z) = (x + y)z$$

e.g. $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial z} = q = x + y = -2 + 5 = 3$$

$$\frac{\partial f}{\partial q} = z = -4 \quad \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x} = -4 * 1 = -4$$

Forward & Back Prop.

Backpropagation: a simple example

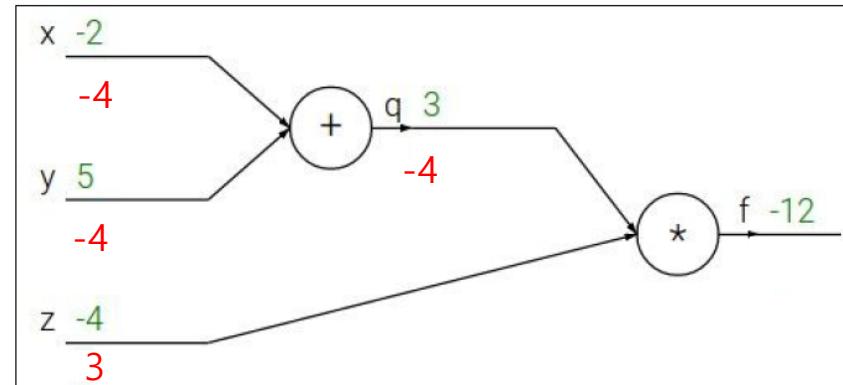
$$f(x, y, z) = (x + y)z$$

e.g. $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

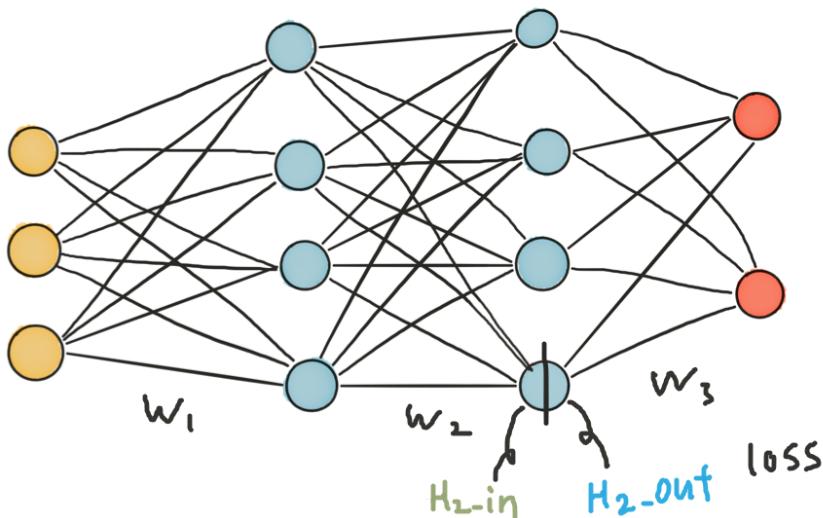


$$\frac{\partial f}{\partial z} = q = x + y = -2 + 5 = 3$$

$$\frac{\partial f}{\partial q} = z = -4 \quad \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x} = -4 * 1 = -4$$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y} = -4 * 1 = -4$$

Forward & Back Prop.



$$\text{loss} = W_3 \times \text{sig}(w_2 \times \text{sig}(w_1 \times b_1) + b_2) + b_3 - y$$

$\underbrace{\hspace{10em}}$ $H_2\text{-in}$
 $\underbrace{\hspace{10em}}$ $H_2\text{-out}$

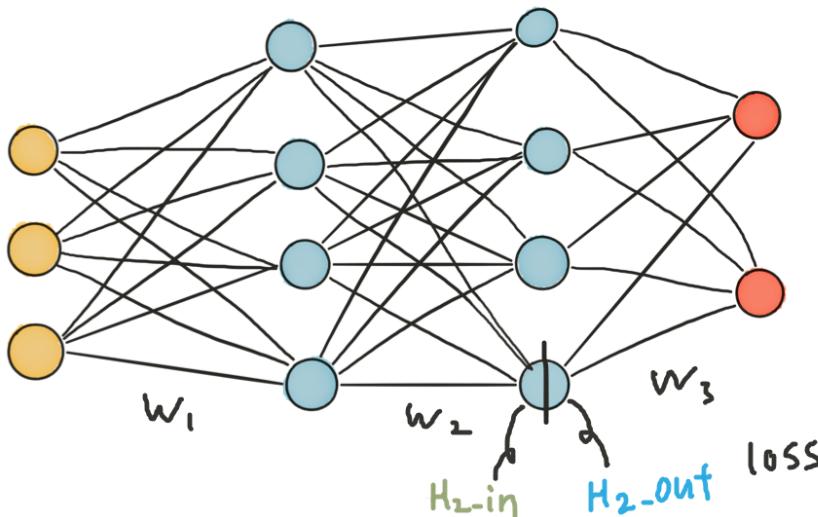
$$\frac{\partial \text{loss}}{\partial w_2} = \frac{\partial \text{loss}}{\partial H_2\text{-out}} \times \frac{\partial H_2\text{-out}}{\partial H_2\text{-in}} \times \frac{\partial H_2\text{-in}}{\partial w_2}$$

$\frac{\partial}{\partial}$
 loss
 $H_2\text{-out}$ 의 부기

$\frac{\partial}{\partial}$
 $H_2\text{-out}$
 $H_2\text{-in}$ 의 부기

$\frac{\partial}{\partial}$
 $H_2\text{-in}$
 w_2 의 부기

Forward & Back Prop.



$$\frac{\partial \text{loss}}{\partial w_2} = w_3 * \text{sigmoid}'(h_{2\downarrow \text{in}}) * \text{sigmoid}(w_1 * x + b)$$

$$\text{loss} = W_3 \times \text{sig}(w_2 \times \text{sig}(w_1 \times x + b_1) + b_2) + b_3 - y$$

$\underbrace{w_2 \times \text{sig}(w_1 \times x + b_1) + b_2}_{H_2\text{-in}}$ $\underbrace{+ b_3}_{H_2\text{-out}}$

$$\frac{\partial \text{loss}}{\partial w_2} = \frac{\partial \text{loss}}{\partial H_2\text{-out}} \times \frac{\partial H_2\text{-out}}{\partial H_2\text{-in}} \times \frac{\partial H_2\text{-in}}{\partial w_2}$$

$\frac{\partial \text{loss}}{\partial H_2\text{-out}}$ $\frac{\partial H_2\text{-out}}{\partial H_2\text{-in}}$ $\frac{\partial H_2\text{-in}}{\partial w_2}$

H₂-out의 변화율 H₂-in의 변화율 w₂의 변화율

Forward & Back Prop.

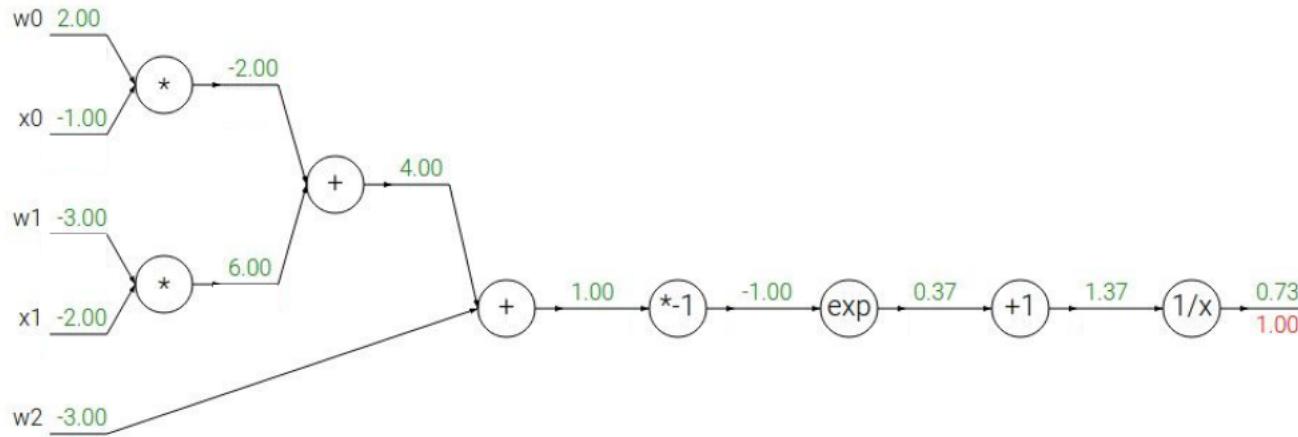
(참고) sigmoid 함수의 미분

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

$$\sigma(x)(1-\sigma(x)) = \frac{1}{1+e^{-x}} \left(1 - \frac{1}{1+e^{-x}}\right) = \frac{1}{1+e^{-x}} \left(\frac{e^{-x}}{1+e^{-x}}\right) = \frac{e^{-x}}{(1+e^{-x})^2}$$

Forward & Back Prop.

Another example: $f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$



$$f(x) = e^x$$

→

$$\frac{df}{dx} = e^x$$

$$f(x) = \frac{1}{x}$$

$$\frac{df}{dx} = -1/x^2$$

$$f_a(x) = ax$$

→

$$\frac{df}{dx} = a$$

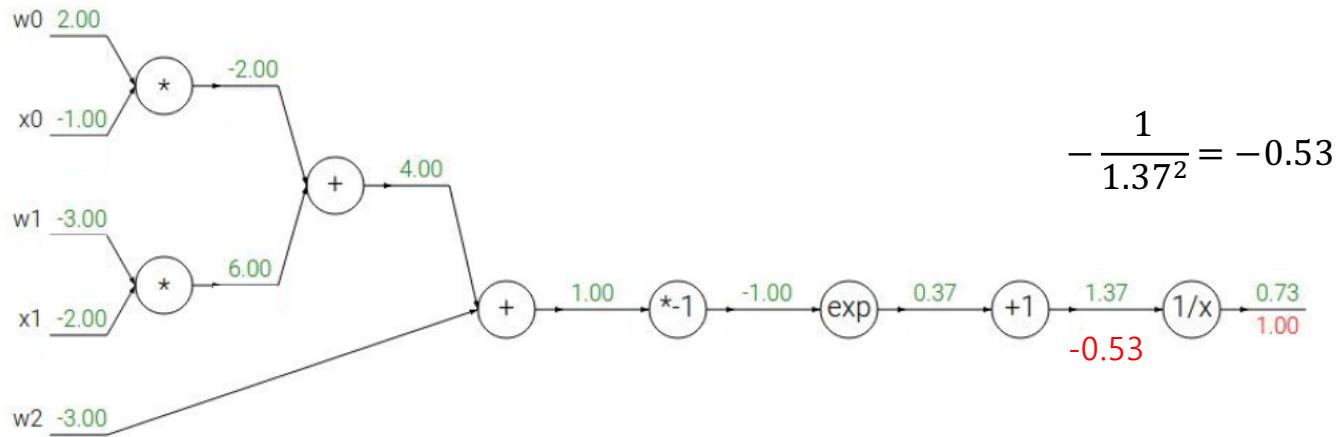
$$f_c(x) = c + x$$

→

$$\frac{df}{dx} = 1$$

Forward & Back Prop.

Another example: $f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$



$$f(x) = e^x \rightarrow$$

$$\frac{df}{dx} = e^x$$

$$f(x) = \frac{1}{x} \rightarrow$$

$$\frac{df}{dx} = -1/x^2$$

$$f_a(x) = ax \rightarrow$$

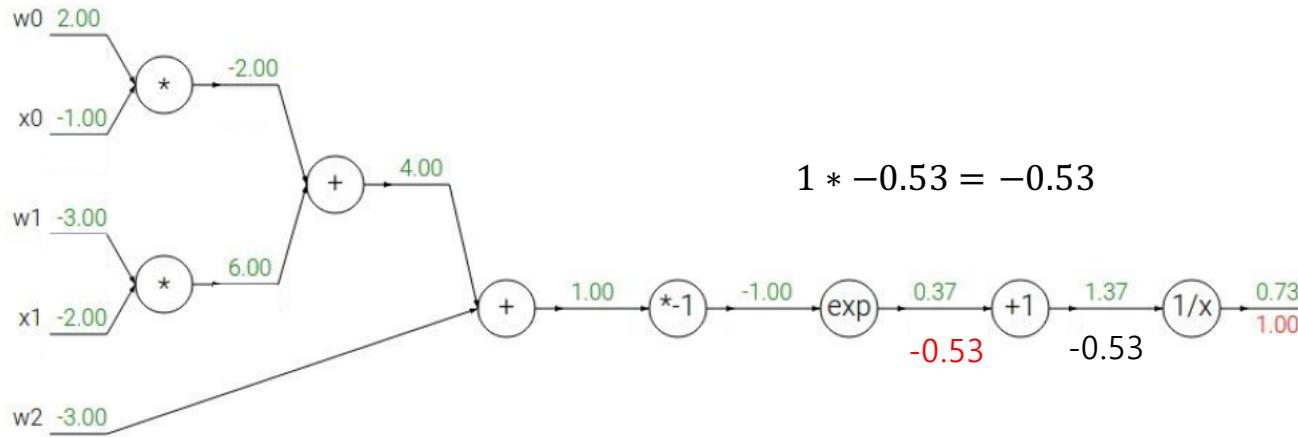
$$\frac{df}{dx} = a$$

$$f_c(x) = c + x \rightarrow$$

$$\frac{df}{dx} = 1$$

Forward & Back Prop.

Another example: $f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$



$$f(x) = e^x$$

→

$$\frac{df}{dx} = e^x$$

$$f(x) = \frac{1}{x}$$

$$\frac{df}{dx} = -1/x^2$$

$$f_a(x) = ax$$

→

$$\frac{df}{dx} = a$$

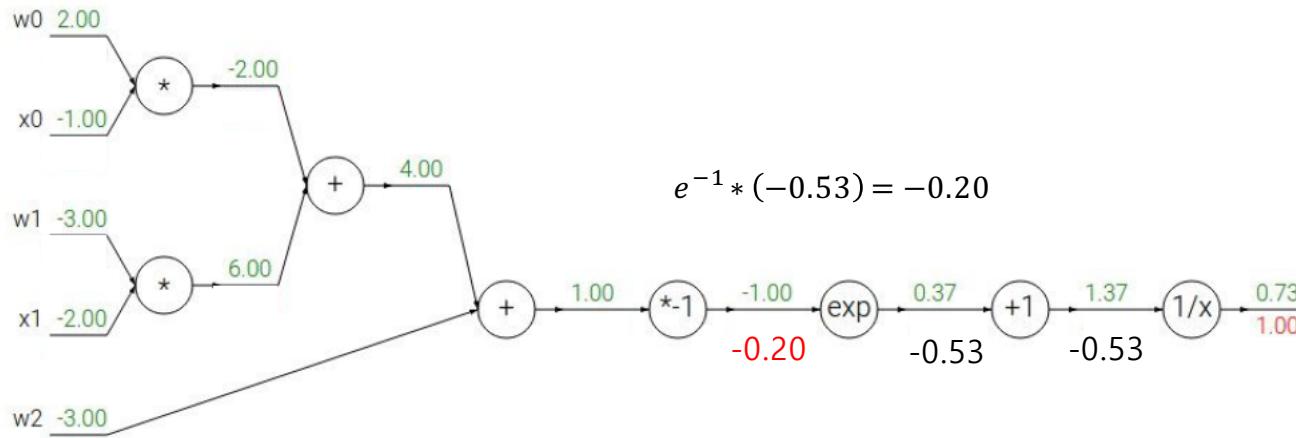
$$f_c(x) = c + x$$

→

$$\frac{df}{dx} = 1$$

Forward & Back Prop.

Another example: $f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$



$$f(x) = e^x$$

→

$$\frac{df}{dx} = e^x$$

$$f(x) = \frac{1}{x}$$

$$\frac{df}{dx} = -1/x^2$$

$$f_a(x) = ax$$

→

$$\frac{df}{dx} = a$$

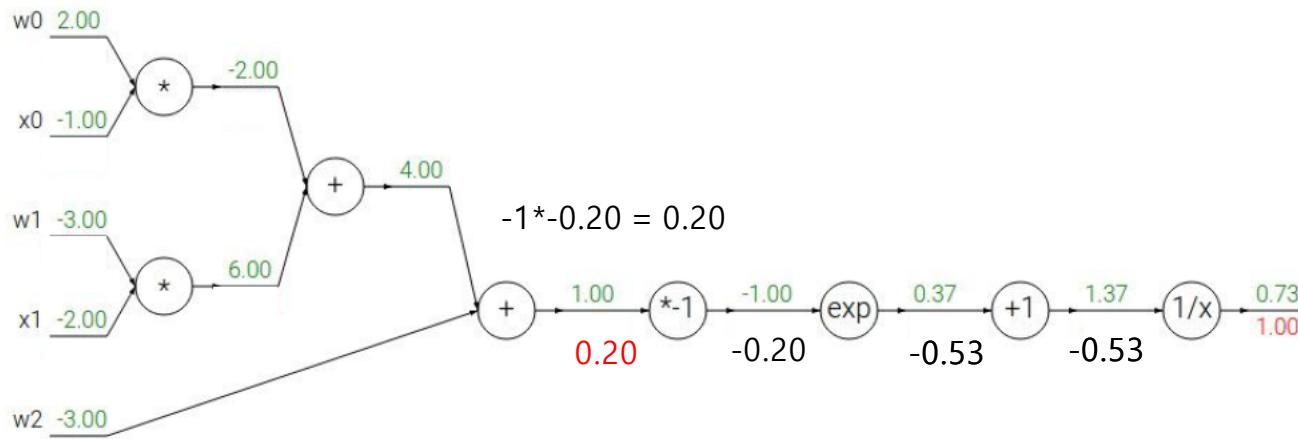
$$f_c(x) = c + x$$

→

$$\frac{df}{dx} = 1$$

Forward & Back Prop.

Another example: $f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$



$$f(x) = e^x \rightarrow$$

$$\frac{df}{dx} = e^x$$

$$f(x) = \frac{1}{x} \rightarrow$$

$$\frac{df}{dx} = -1/x^2$$

$$f_a(x) = ax \rightarrow$$

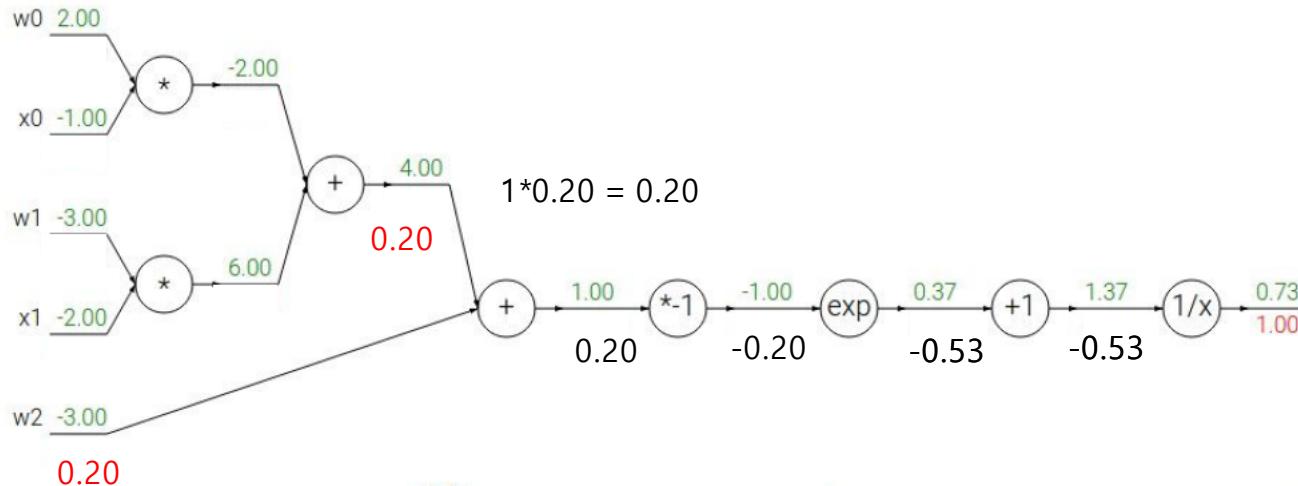
$$\frac{df}{dx} = a$$

$$f_c(x) = c + x \rightarrow$$

$$\frac{df}{dx} = 1$$

Forward & Back Prop.

Another example: $f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$



$$f(x) = e^x$$

→

$$\frac{df}{dx} = e^x$$

$$f(x) = \frac{1}{x}$$

$$\frac{df}{dx} = -1/x^2$$

$$f_a(x) = ax$$

→

$$\frac{df}{dx} = a$$

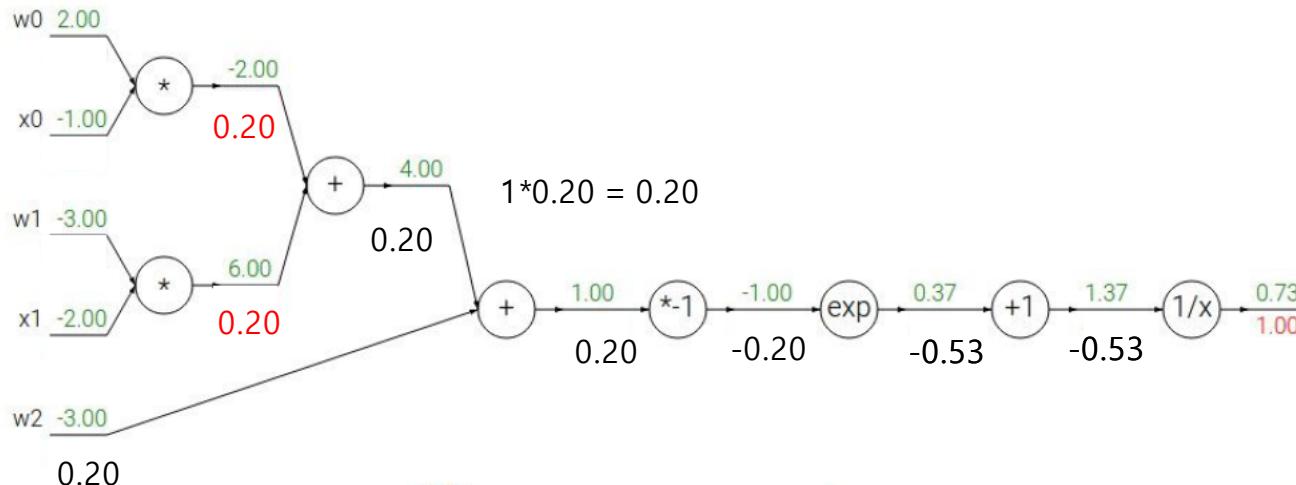
$$f_c(x) = c + x$$

→

$$\frac{df}{dx} = 1$$

Forward & Back Prop.

Another example: $f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$



$$f(x) = e^x$$

→

$$\frac{df}{dx} = e^x$$

$$f(x) = \frac{1}{x}$$

$$\frac{df}{dx} = -1/x^2$$

$$f_a(x) = ax$$

→

$$\frac{df}{dx} = a$$

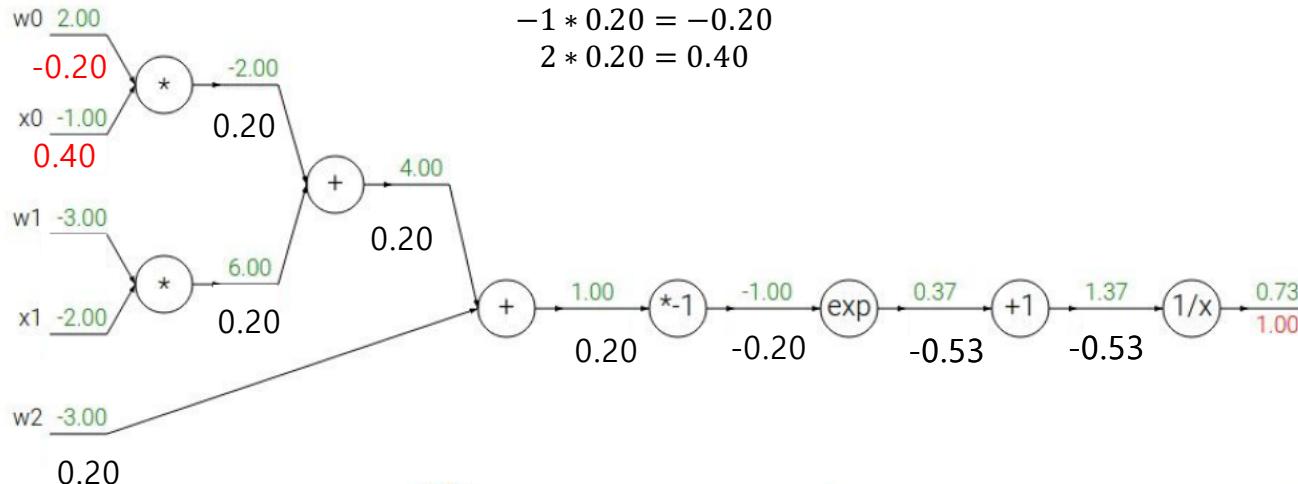
$$f_c(x) = c + x$$

→

$$\frac{df}{dx} = 1$$

Forward & Back Prop.

Another example: $f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$



$$f(x) = e^x$$

\rightarrow

$$\frac{df}{dx} = e^x$$

$$f(x) = \frac{1}{x}$$

$$\frac{df}{dx} = -1/x^2$$

$$f_a(x) = ax$$

\rightarrow

$$\frac{df}{dx} = a$$

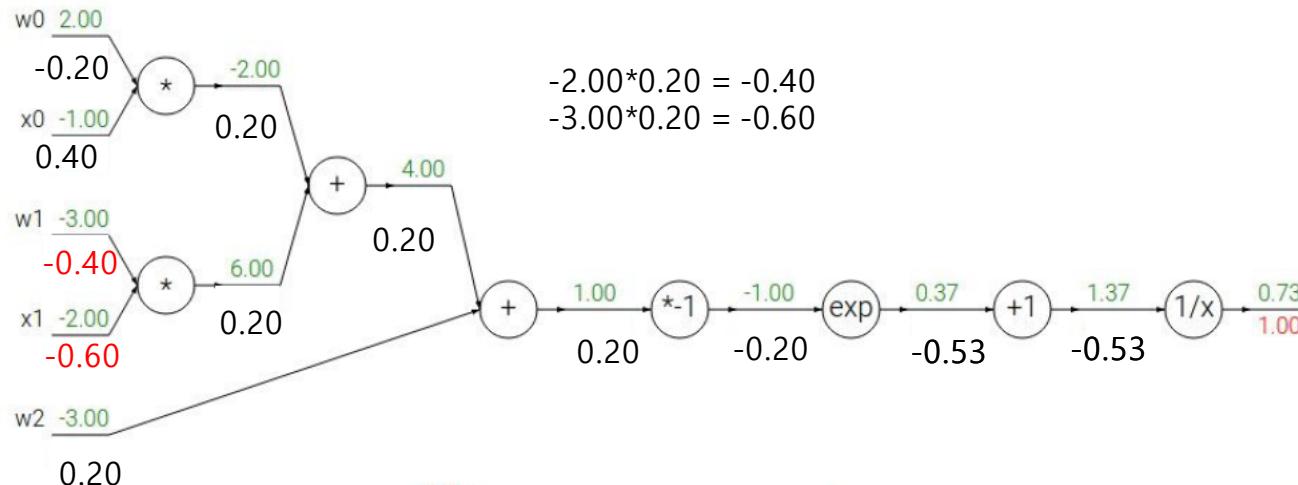
$$f_c(x) = c + x$$

\rightarrow

$$\frac{df}{dx} = 1$$

Forward & Back Prop.

Another example: $f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}}$



$$f(x) = e^x$$

→

$$\frac{df}{dx} = e^x$$

$$f(x) = \frac{1}{x}$$

$$\frac{df}{dx} = -1/x^2$$

$$f_a(x) = ax$$

→

$$\frac{df}{dx} = a$$

$$f_c(x) = c + x$$

→

$$\frac{df}{dx} = 1$$

Forward & Back Prop.

$$Y = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}} = [1 + e^{-(w_0x_0 + w_1x_1 + w_2)}]^{-1}$$

$$\text{loss} = \hat{Y} - Y$$

$$\approx [1 + e^{-(w_0x_0 + w_1x_1 + w_2)}]^{-1} - Y$$

$\alpha = -1$
 $\beta = 0.37$
 $\sigma = 1.37$

$$\frac{\partial \text{loss}}{\partial r} = (r^{-1})' = -\frac{1}{r^2} = -0.5327$$

$$\frac{\partial r}{\partial \beta} = 1$$

$$\frac{\partial \beta}{\partial \alpha} = (e^\alpha)' = e^{-\alpha} = 0.3678$$

$$\frac{\partial \alpha}{\partial w_0} = -x_0 = 1$$

$w_0 = 2$
 $w_1 = -3$
 $w_2 = -3$
 $x_0 = -1$
 $x_1 = -2$

$$\text{loss} = r^{-1} - Y$$

$$\sigma = 1 + \beta$$

$$\beta = e^\alpha$$

$$\frac{\partial \text{loss}}{\partial w_0} = \frac{\partial \text{loss}}{\partial r} \times \frac{\partial r}{\partial \beta} \times \frac{\partial \beta}{\partial \alpha} \times \frac{\partial \alpha}{\partial w_0}$$

$$\begin{aligned}
 &= -0.5327 \times 1 \times 0.3678 \times 1 \\
 &\approx -0.1959 \approx -0.2
 \end{aligned}$$

Neural Network

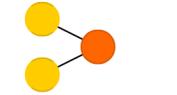
A mostly complete chart of

Neural Networks

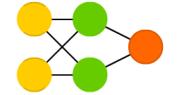
©2016 Fjodor van Veen - [asimovinstitute.org](http://www.asimovinstitute.org)

- Backfed Input Cell
- Input Cell
- △ Noisy Input Cell
- Hidden Cell
- Probabilistic Hidden Cell
- △ Spiking Hidden Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- △ Different Memory Cell
- Kernel
- Convolution or Pool

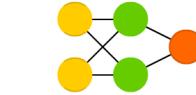
Perceptron (P)



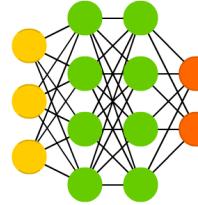
Feed Forward (FF)



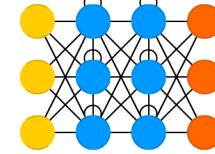
Radial Basis Network (RBF)



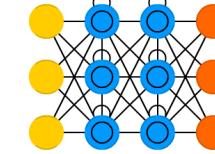
Deep Feed Forward (DFF)



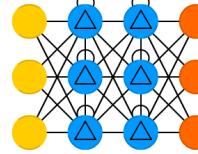
Recurrent Neural Network (RNN)



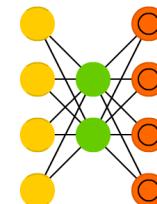
Long / Short Term Memory (LSTM)



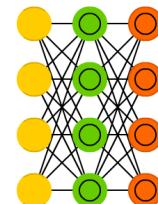
Gated Recurrent Unit (GRU)



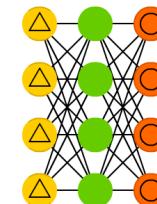
Auto Encoder (AE)



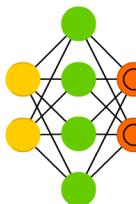
Variational AE (VAE)



Denoising AE (DAE)



Sparse AE (SAE)



Neural Network

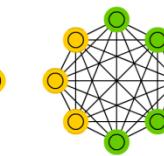
Markov Chain (MC)



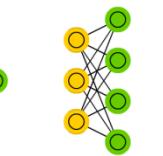
Hopfield Network (HN)



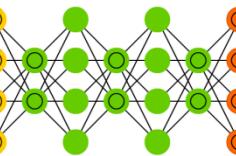
Boltzmann Machine (BM)



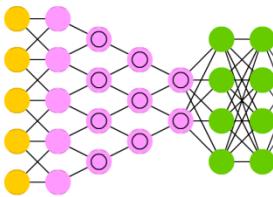
Restricted BM (RBM)



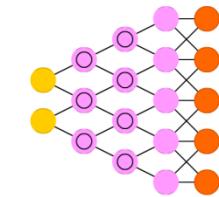
Deep Belief Network (DBN)



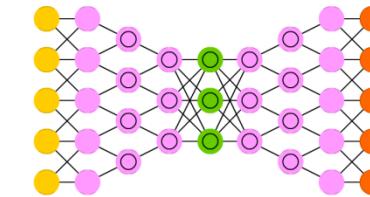
Deep Convolutional Network (DCN)



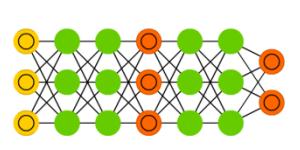
Deconvolutional Network (DN)



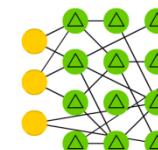
Deep Convolutional Inverse Graphics Network (DCIGN)



Generative Adversarial Network (GAN)



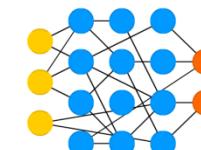
Liquid State Machine (LSM)



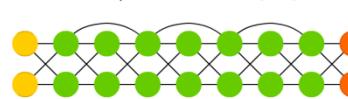
Extreme Learning Machine (ELM)



Echo State Network (ESN)



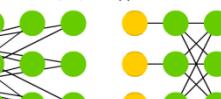
Deep Residual Network (DRN)



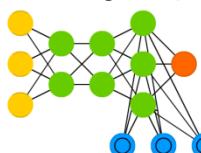
Kohonen Network (KN)



Support Vector Machine (SVM)



Neural Turing Machine (NTM)



Forward & Back Prop.

```
◀ ▶ test.py ×
1 import numpy as np
2 import torch
3 import torch.nn as nn
4 import torch.optim as optim
5 import torch.nn.init as init
6 from torch.autograd import Variable
7 from visdom import Visdom
8 viz = Visdom()
9
10 num_data = 1000
11 num_epoch = 5000
12
13 x = init.uniform(torch.Tensor(num_data,1),-15,15)
14 y = 8*(x**2) + 7*x + 3
15
16 noise = init.normal(torch.FloatTensor(num_data,1),std=1)
17 y_noise = y + noise
```

Forward & Back Prop.

필요한 라이브러리

```
test.py
```

```
1 import numpy as np
2 import torch
3 import torch.nn as nn
4 import torch.optim as optim
5 import torch.nn.init as init
6 from torch.autograd import Variable
7 from visdom import Visdom
8 viz = Visdom()

9
10 num_data = 1000
11 num_epoch = 5000
12
13 x = init.uniform(torch.Tensor(num_data,1),-15,15)
14 y = 8*(x**2) + 7*x + 3
15
16 noise = init.normal(torch.FloatTensor(num_data,1),std=1)
17 y_noise = y + noise
```

Forward & Back Prop.

필요한 라이브러리

```
test.py
```

```
1 import numpy as np
2 import torch
3 import torch.nn as nn
4 import torch.optim as optim
5 import torch.nn.init as init
6 from torch.autograd import Variable
7 from visdom import Visdom
8 viz = Visdom()

9
10 num_data = 1000
11 num_epoch = 5000
12
13 x = init.uniform(torch.Tensor(num_data,1),-15,15)
14 y = 8*(x**2) + 7*x + 3
15
16 noise = init.normal(torch.FloatTensor(num_data,1),std=1)
17 y_noise = y + noise
```

데이터 생성

Forward & Back Prop.

```
21 model = nn.Sequential(  
22     nn.Linear(1,10),  
23     nn.ReLU(),  
24     nn.Linear(10,6),  
25     nn.ReLU(),  
26     nn.Linear(6,1),  
27 ).cuda()  
28  
29 loss_func = nn.L1Loss()  
30 optimizer = optim.SGD(model.parameters(), lr=0.001)  
31  
32 loss_arr =[]  
33 label = Variable(y_noise.cuda())  
34 for i in range(num_epoch):  
35     output = model(Variable(x.cuda()))  
36     optimizer.zero_grad()  
37  
38     loss = loss_func(output,label)  
39     loss.backward()  
40     optimizer.step()  
41     if i % 100 ==0:  
42         print(loss)  
43     loss_arr.append(loss.cpu().data.numpy()[0])  
44  
45 param_list = list(model.parameters())  
46 print(param_list)
```

Forward & Back Prop.

Neural Network 모델 생성

loss function 및
gradient descent
optimizer 생성

```
21 model = nn.Sequential(  
22     nn.Linear(1,10),  
23     nn.ReLU(),  
24     nn.Linear(10,6),  
25     nn.ReLU(),  
26     nn.Linear(6,1),  
27 ).cuda()  
28  
29 loss_func = nn.L1Loss()  
30 optimizer = optim.SGD(model.parameters(), lr=0.001)  
31  
32 loss_arr = []  
33 label = Variable(y_noise.cuda())  
34 for i in range(num_epoch):  
35     output = model(Variable(x.cuda()))  
36     optimizer.zero_grad()  
37  
38     loss = loss_func(output,label)  
39     loss.backward()  
40     optimizer.step()  
41     if i % 100 ==0:  
42         print(loss)  
43     loss_arr.append(loss.cpu().data.numpy()[0])  
44  
45 param_list = list(model.parameters())  
46 print(param_list)
```

Forward & Back Prop.

Neural Network 모델 생성

```
21 model = nn.Sequential(  
22     nn.Linear(1,10),  
23     nn.ReLU(),  
24     nn.Linear(10,6),  
25     nn.ReLU(),  
26     nn.Linear(6,1),  
27 ).cuda()
```

loss function 및
gradient descent
optimizer 생성

```
29 loss_func = nn.L1Loss()  
30 optimizer = optim.SGD(model.parameters(), lr=0.001)
```

```
31  
32 loss_arr =[]  
33 label = Variable(y_noise.cuda())  
34 for i in range(num_epoch):  
35     output = model(Variable(x.cuda()))  
36     optimizer.zero_grad()
```

```
37  
38     loss = loss_func(output,label)  
39     loss.backward()  
40     optimizer.step()  
41     if i % 100 ==0:  
42         print(loss)  
43     loss_arr.append(loss.cpu().data.numpy()[0])  
44  
45 param_list = list(model.parameters())  
46 print(param_list)
```

Forward & Back Prop.

Neural Network 모델 생성

loss function 및
gradient descent
optimizer 생성

- <training 단계>
1. 모델로 결과값 추정
 2. loss 및 gradient 계산
 3. 모델 업데이트

```
21 model = nn.Sequential(  
22     nn.Linear(1,10),  
23     nn.ReLU(),  
24     nn.Linear(10,6),  
25     nn.ReLU(),  
26     nn.Linear(6,1),  
27 ).cuda()  
28  
29 loss_func = nn.L1Loss()  
30 optimizer = optim.SGD(model.parameters(), lr=0.001)  
31  
32 loss_arr = []  
33 label = Variable(y_noise.cuda())  
34 for i in range(num_epoch):  
35     output = model(Variable(x.cuda()))  
36     optimizer.zero_grad()  
37  
38     loss = loss_func(output,label)  
39     loss.backward()  
40     optimizer.step()  
41     if i % 100 ==0:  
42         print(loss)  
43     loss_arr.append(loss.cpu().data.numpy()[0])  
44  
45 param_list = list(model.parameters())  
46 print(param_list)
```

Forward & Back Prop.

Neural Network 모델 생성

```
21 model = nn.Sequential(  
22     nn.Linear(1,10),  
23     nn.ReLU(),  
24     nn.Linear(10,6),  
25     nn.ReLU(),  
26     nn.Linear(6,1),  
27 ).cuda()
```

loss function 및
gradient descent
optimizer 생성

```
29 loss_func = nn.L1Loss()  
30 optimizer = optim.SGD(model.parameters(), lr=0.001)
```

```
32 loss_arr =[]  
33 label = Variable(y_noise.cuda())  
34 for i in range(num_epoch):  
35     output = model(Variable(x.cuda()))  
36     optimizer.zero_grad()
```

```
38     loss = loss_func(output,label)  
39     loss.backward()  
40     optimizer.step()  
41     if i % 100 ==0:  
42         print(loss)  
43     loss_arr.append(loss.cpu().data.numpy()[0])
```

```
45 param_list = list(model.parameters())  
46 print(param_list)
```

- <training 단계>
1. 모델로 결과값 추정
 2. loss 및 gradient 계산
 3. 모델 업데이트

training 이후 파라미터 값 확인

Q&A
