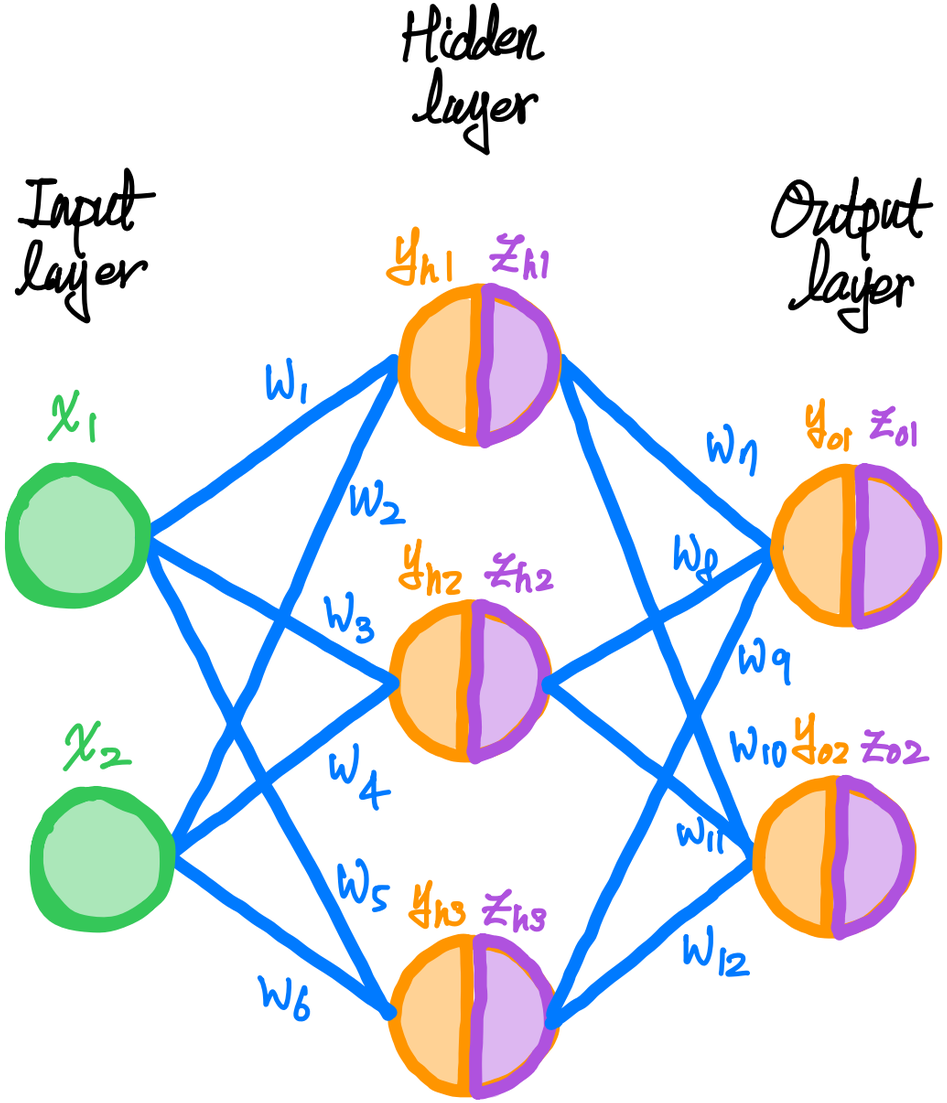
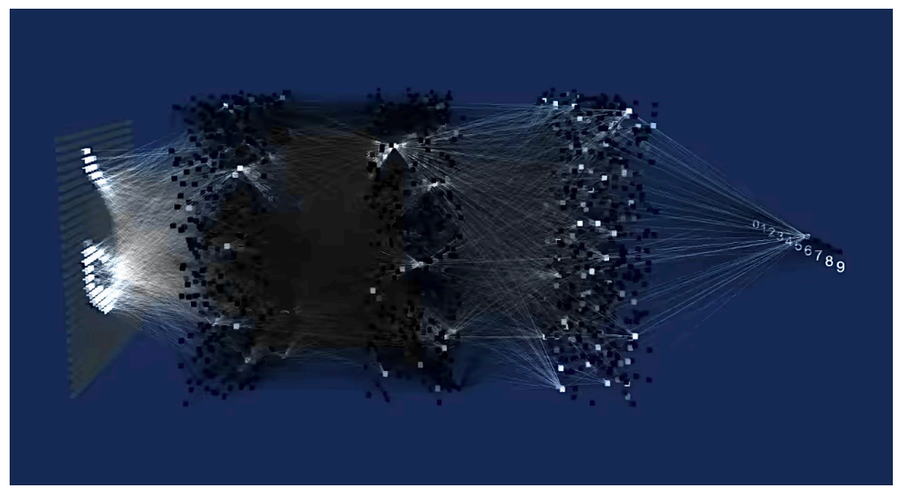
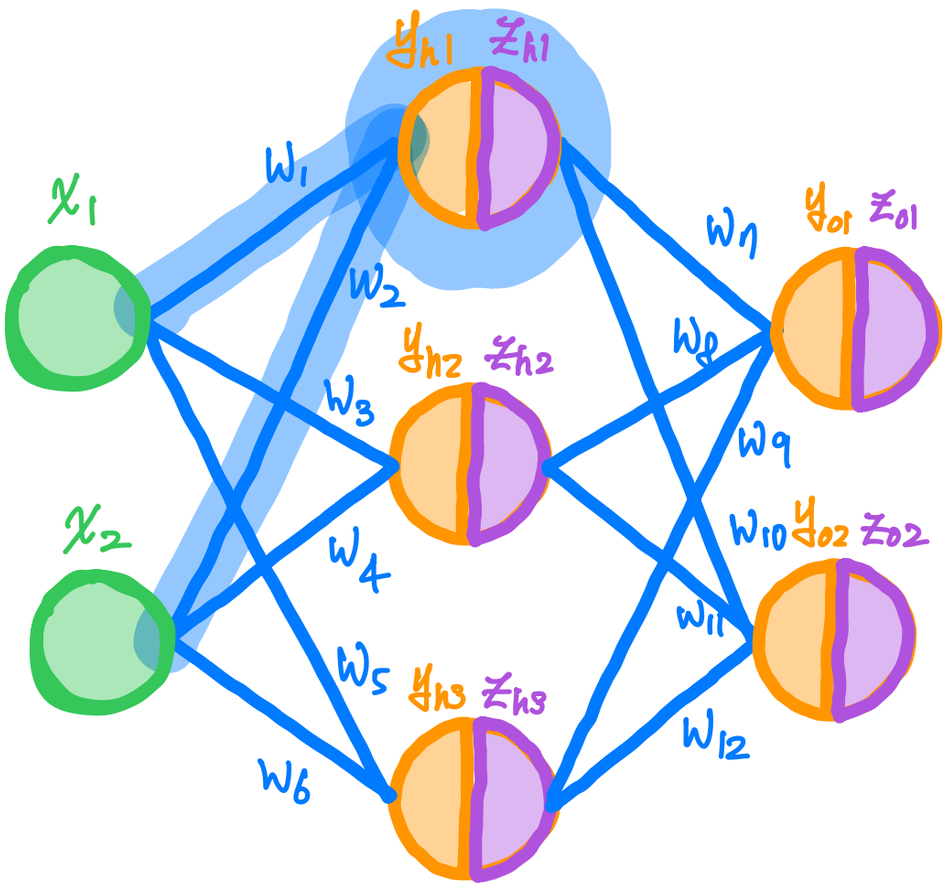




Dimension Reduction → Understanding

/star-bits

Foward Propagation

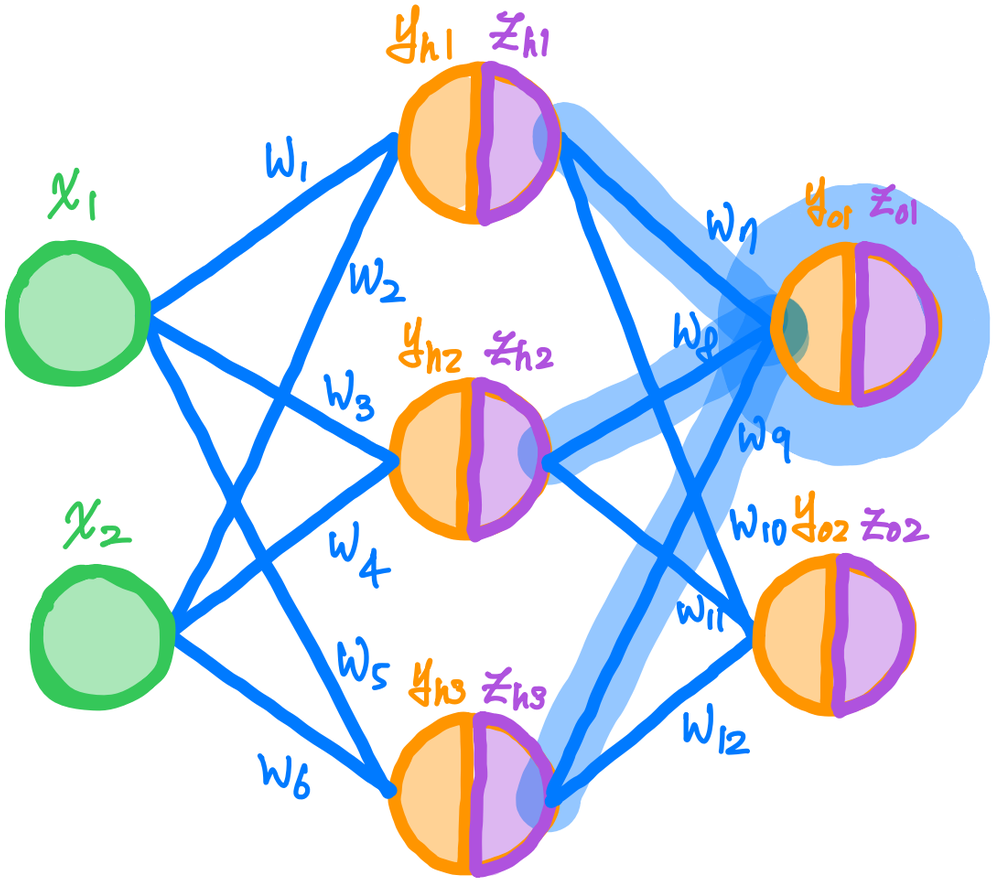


$$y_{h1} = w_1 \cdot x_1 + w_2 \cdot x_2$$

$$z_{h1} = \text{sigmoid}(y_{h1})$$

$$= \frac{1}{1 + e^{-y_{h1}}}$$

Foward Propagation



weight는 어떻게 알아? trainable elements

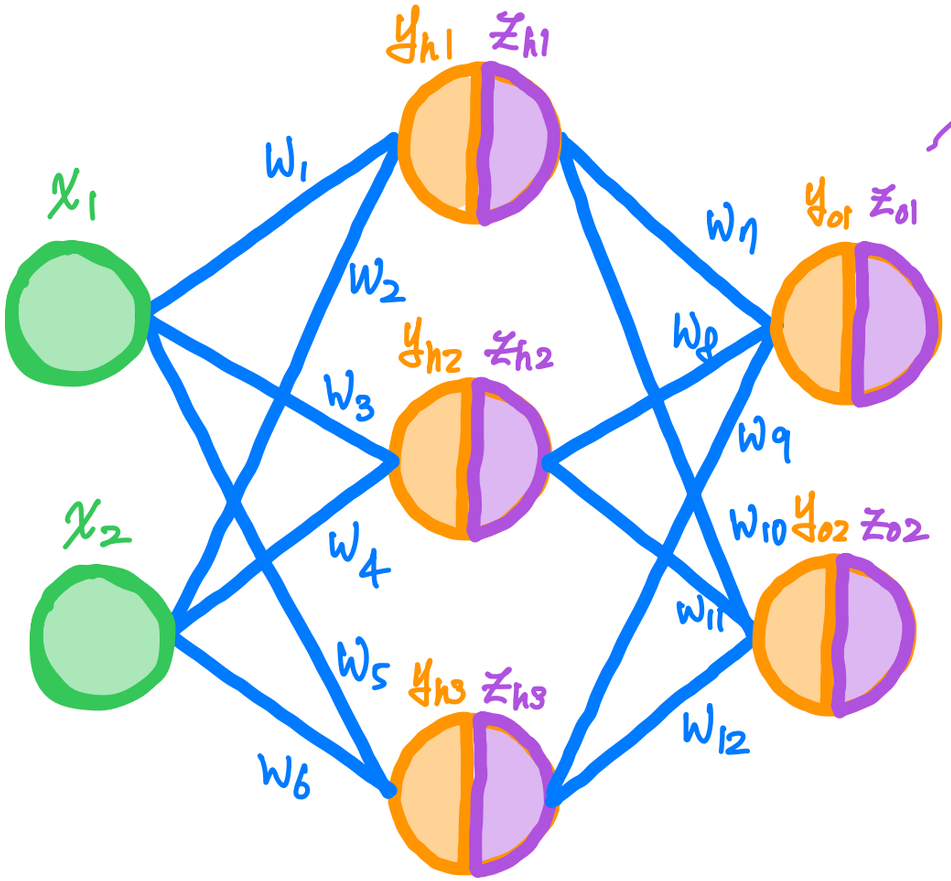
$$y_{o1} = w_7 \cdot z_{h1} + w_8 \cdot z_{h2} + w_9 \cdot z_{h3}$$

$$z_{o1} = \text{sigmoid}(y_{o1})$$

$$= \frac{1}{1 + e^{-y_{o1}}}$$

= 이 카테고리에 대한 예측값

예측이 얼만큼 틀렸을까



$$L = -\sum_i t_i \ln(z_i)$$

$$= -\left[t_1 \ln(z_{o1}) + t_2 \ln(z_{o2})\right]$$

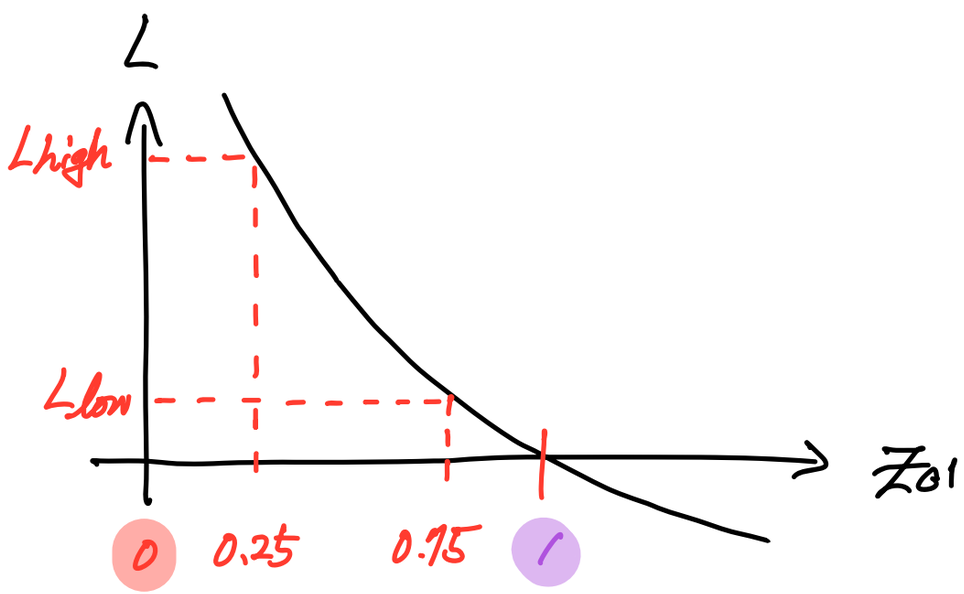
$$= -\left[t_1 \ln(z_{o1}) + (1-t_1) \ln(1-z_{o1})\right]$$

$$= L_1 + L_2 = L_{total}$$

Q4 cross entropy loss 长这样？

$$L = -[t_1 \ln(z_{01}) + (1-t_1)\ln(1-z_{01})]$$

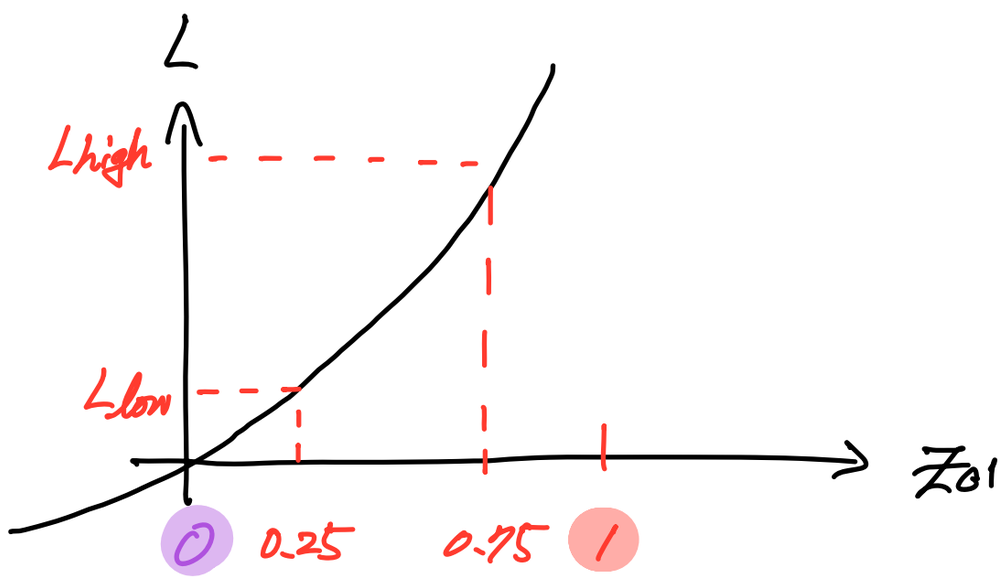
e.g. $t_1 = 1$: $L = -[\ln z_{01}]$



Q4 cross entropy loss 长这样？

$$L = -[t_1 \ln(z_{01}) + (1-t_1)\ln(1-z_{01})]$$

e.g. $t_1 = 0$: $L = -[\ln(1-z_{01})]$



왜 'cross' entropy 인가?

$S = k_B \ln \Omega$. As $\Omega_{total} = \Omega_1 \times \Omega_2$, $S_{total} = S_1 + S_2$

$S = -k_B \sum_i p_i \ln p_i$ Accounts for probabilities per microstate.

If $p = \frac{1}{\Omega}$, $S = -k_B \Omega \frac{1}{\Omega} \ln\left(\frac{1}{\Omega}\right)$

$H = -\sum_i p_i \ln p_i$ Average amount of information content in a message

$-\ln p_i$: information content of an event i.

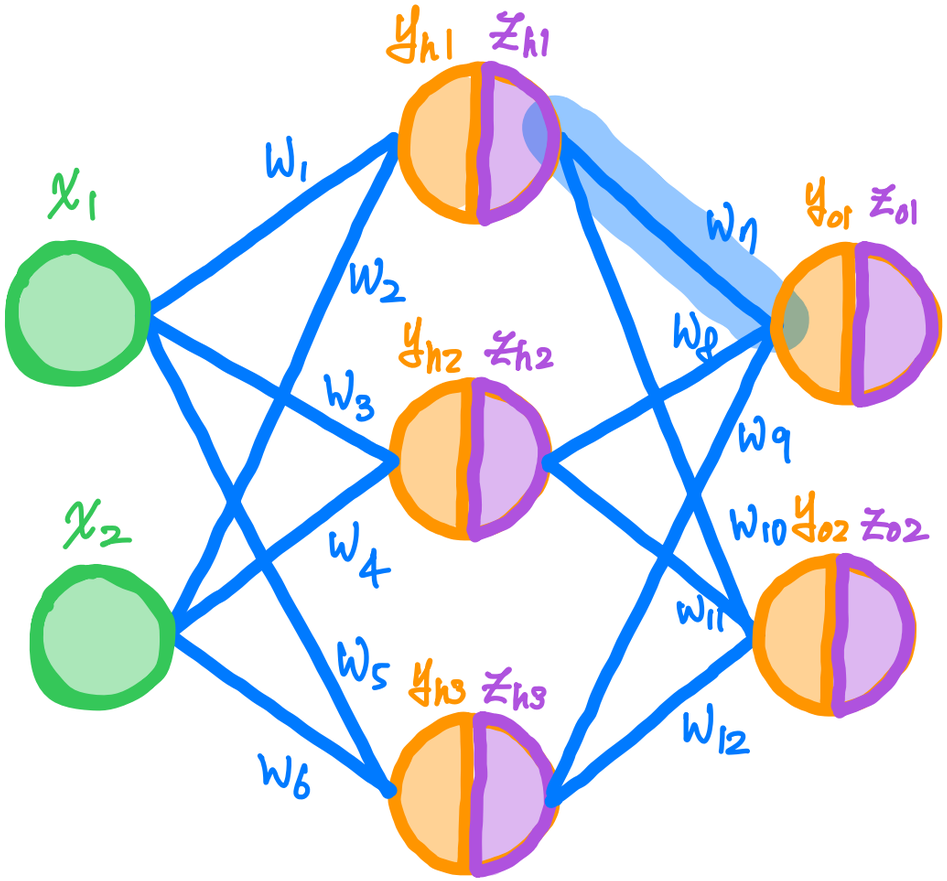
As $p_i \to 1$, surprisal $\to 0$.

As $p_i \to 0$, surprisal $\to \infty$.

$L = -\sum_i t_i \ln p_i$

'cross'

예측이 틀린 정도에 따라 weight를 조정하자

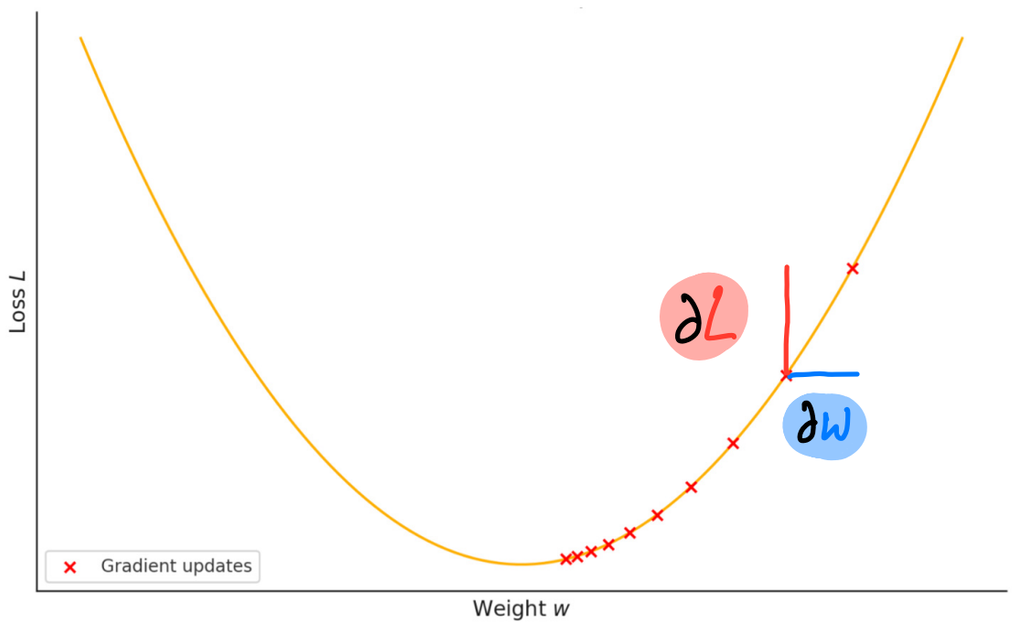


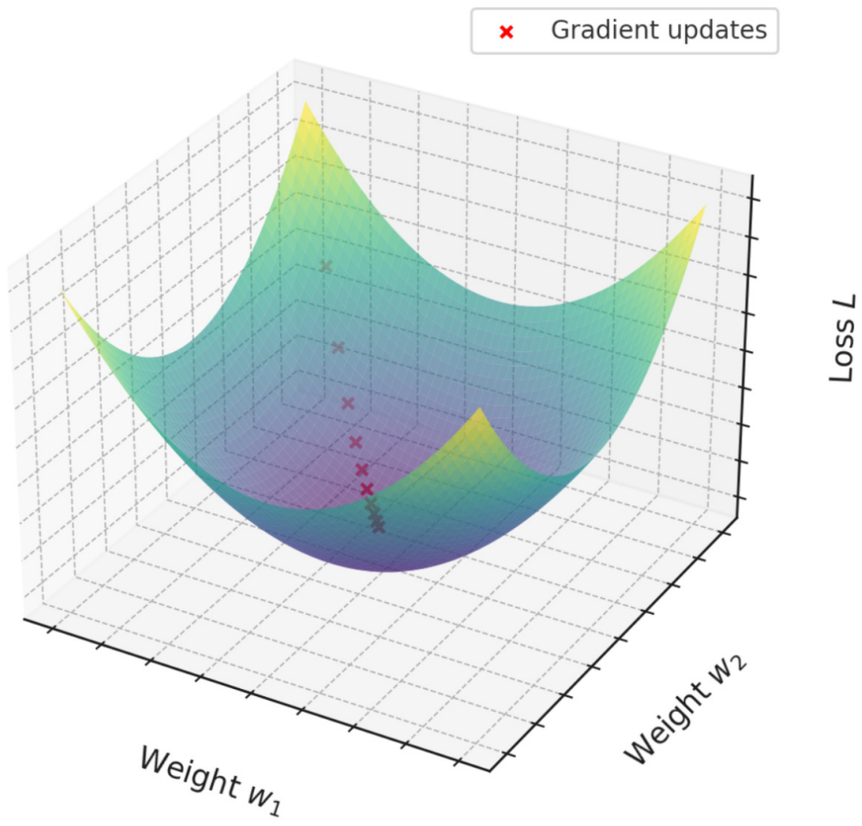
numerical gradient

$$w_{new} = w_{old} - LR \cdot \frac{\partial L_{total}}{\partial w_{old}}$$

w를 조금 바꿔봤을 때 L_{total}이 변하는 정도.

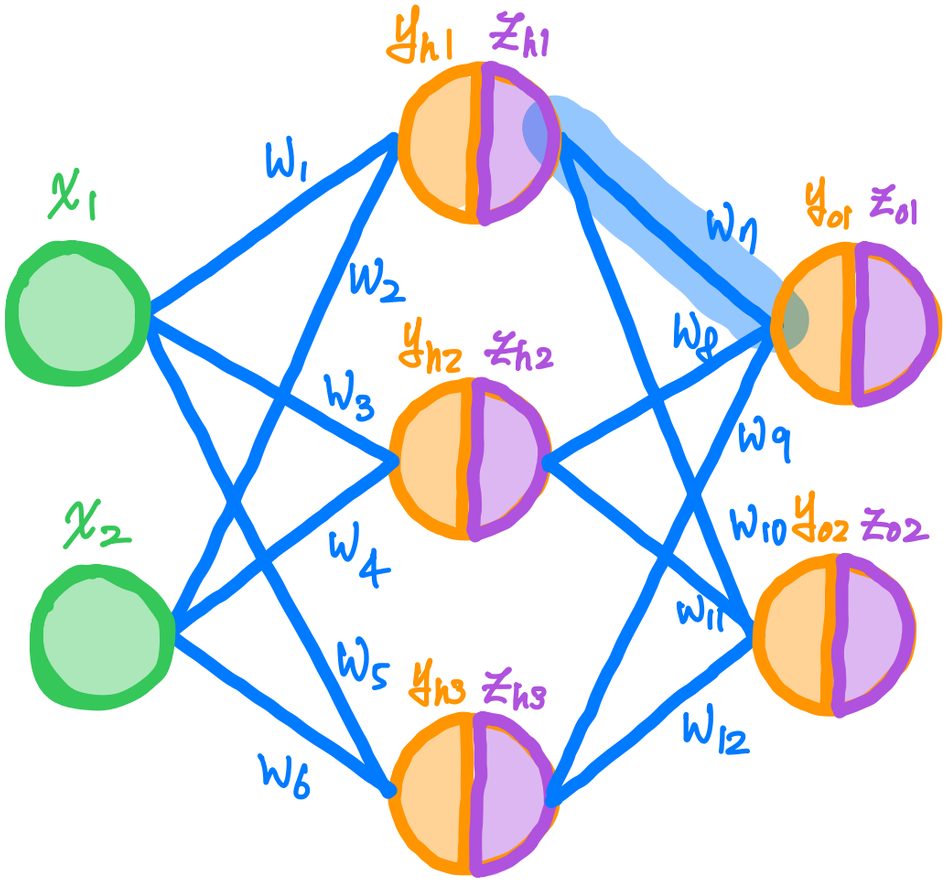
언제까지? Loss를 더 줄일 수 없을때까지





LR

목표: $\frac{\partial L_{total}}{\partial w_7}$ 구하기



$$\frac{\partial L_{total}}{\partial w_7} = \frac{\partial L_1}{\partial w_7}$$

$$L_1 = Loss(\ sigmoid(y_{o1}))$$

$$\frac{\partial L_1}{\partial w_7} = \boxed{\frac{\partial L_1}{\partial z_{o1}}} \cdot \boxed{\frac{\partial z_{o1}}{\partial y_{o1}}} \cdot \boxed{\frac{\partial y_{o1}}{\partial w_7}}$$

① ② ③

(1)

- $L_1 = -[t_1 \ln(z_{01}) + (1-t_1)\ln(1-z_{01})]$

$$\frac{\partial L_1}{\partial z_{01}} = -\left[t_1 \cdot \frac{1}{z_{01}} + (1-t_1)\frac{1}{1-z_{01}} \cdot (-1)\right] = \boxed{\frac{z_{01} - t_1}{z_{01}(1-z_{01})}}$$

(1)

(2)

- $z_{o1} = \text{sigmoid}(y_{o1}) = \dfrac{1}{1+e^{-y_{o1}}}$

$$\frac{\partial z_{o1}}{\partial y_{o1}} = \frac{(1)' \cdot (1+e^{-y_{o1}}) - 1 \cdot (1+e^{-y_{o1}})'}{(1+e^{-y_{o1}})^2} = \frac{0 - 1 \cdot (e^{-y_{o1}}) \cdot (-1)}{(1+e^{-y_{o1}})^2}$$

$$= \boxed{\frac{e^{-y_{o1}}}{(1+e^{-y_{o1}})^2}}$$

$\frac{\partial z_{o1}}{\partial y_{o1}}$ 를 in terms of z_{o1} 로 적어보자.

$$\frac{\partial z_{o1}}{\partial y_{o1}} = \frac{e^{-y_{o1}}}{(1+e^{-y_{o1}})^2} = e^{-y_{o1}} \cdot (z_{o1})^2$$

As $z_{o1} = \frac{1}{1+e^{-y_{o1}}}$, $\quad e^{-y_{o1}} = \frac{1}{z_{o1}} - 1 = \frac{1-z_{o1}}{z_{o1}}$

$$\frac{\partial z_{o1}}{\partial y_{o1}} = \frac{1-z_{o1}}{z_{o1}} \cdot (z_{o1})^2 = \boxed{(1-z_{o1}) \cdot z_{o1}}$$

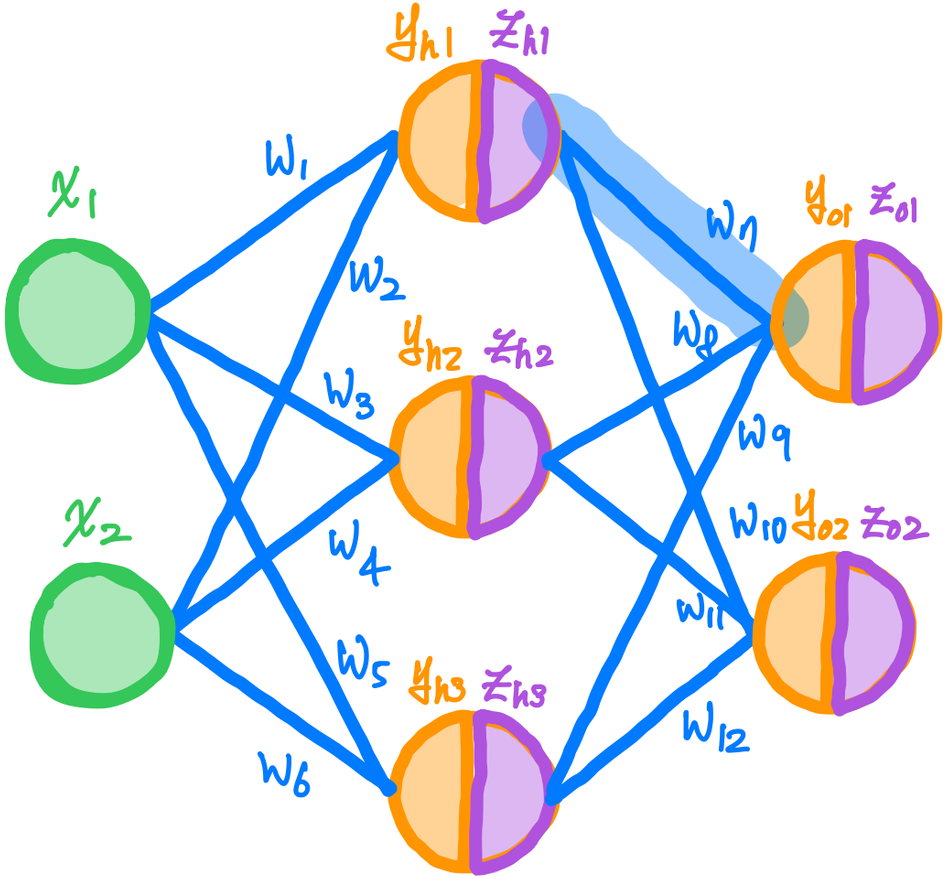
(2)

③

- $y_{o1} = w_1 \cdot z_{h1} + w_2 \cdot z_{h2} + w_4 \cdot z_{h3}$

$$\frac{\partial y_{o1}}{\partial w_1} = \boxed{z_{h1}}$$

③



$$\frac{\partial L_{total}}{\partial w_7}$$

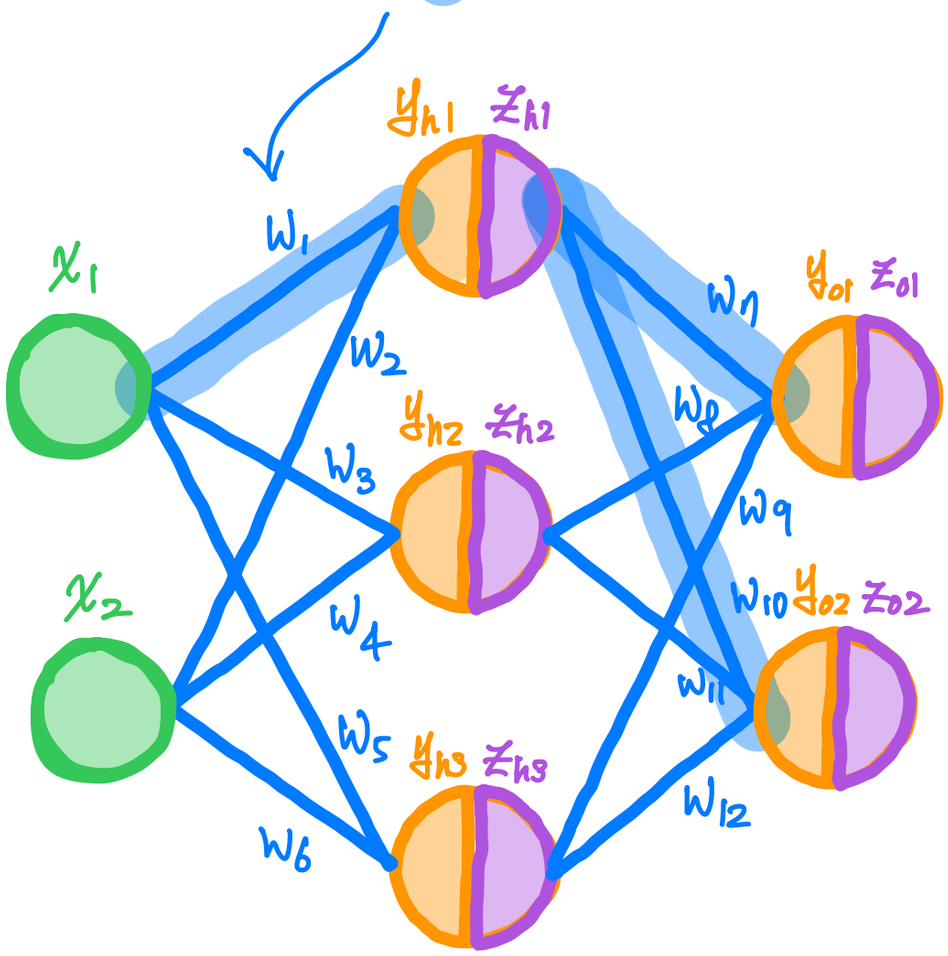
$$= \frac{\partial L_1}{\partial w_7}$$

$$= \frac{\partial L_1}{\partial z_{o1}} \cdot \frac{\partial z_{o1}}{\partial y_{o1}} \cdot \frac{\partial y_{o1}}{\partial w_7} = ① \cdot ② \cdot ③$$

$$= \frac{z_{o1} - t_1}{z_{o1}(1 - z_{o1})} \cdot z_{o1}(1 - z_{o1}) \cdot z_{h1}$$

$$= \boxed{(z_{o1} - t_1) \cdot z_{h1}}$$

목표 : $\frac{\partial L_{total}}{\partial w_1}$ 구하기



$$\frac{\partial L_{total}}{\partial w_1} = \frac{\partial L_1}{\partial w_1} + \frac{\partial L_2}{\partial w_1}$$

$$= \frac{\partial L_1}{\partial z_{o1}} \cdot \frac{\partial z_{o1}}{\partial y_{o1}} \cdot \frac{\partial y_{o1}}{\partial z_{h1}} \cdot \frac{\partial z_{h1}}{\partial y_{h1}} \cdot \frac{\partial y_{h1}}{\partial w_1}$$

$$+ \frac{\partial L_2}{\partial z_{o2}} \cdot \frac{\partial z_{o2}}{\partial y_{o2}} \cdot \frac{\partial y_{o2}}{\partial z_{h2}} \cdot \frac{\partial z_{h2}}{\partial y_{h1}} \cdot \frac{\partial y_{h1}}{\partial w_1}$$

$$= \frac{z_{o1} - t_1}{z_{o1}(1-z_{o1})} \cdot z_{o1}(1-z_{o1}) \cdot w_7 \cdot z_{h1}(1-z_{h1}) \cdot x_1$$

$$+ \frac{z_{o2} - t_2}{z_{o2}(1-z_{o2})} \cdot z_{o2}(1-z_{o2}) \cdot w_{10} \cdot z_{h1}(1-z_{h1}) \cdot x_1$$

$$= \boxed{\begin{aligned} &(z_{o1} - t_1) \cdot w_7 \cdot z_{h1}(1-z_{h1}) \cdot x_1 \\ &+ (z_{o2} - t_2) \cdot w_{10} \cdot z_{h1}(1-z_{h1}) \cdot x_1 \end{aligned}}$$

Backward Propagation

e.g. $\frac{\partial L_{total}}{\partial w_7} = (z_{o1} - t_1) \cdot z_{h1}$

$\frac{\partial L_{total}}{\partial w_1} = (z_{o1} - t_1) \cdot w_7 \cdot z_{h1}(1 - z_{h1}) \cdot x_1 + (z_{o2} - t_2) \cdot w_{10} \cdot z_{h1}(1 - z_{h1}) \cdot x_1$

지금까지 한 것.

Numerical gradient 없이 analytic하게 미분계산가능

sigmoid + cross entropy loss의 backprop은 $(z-t)$ 꼴.

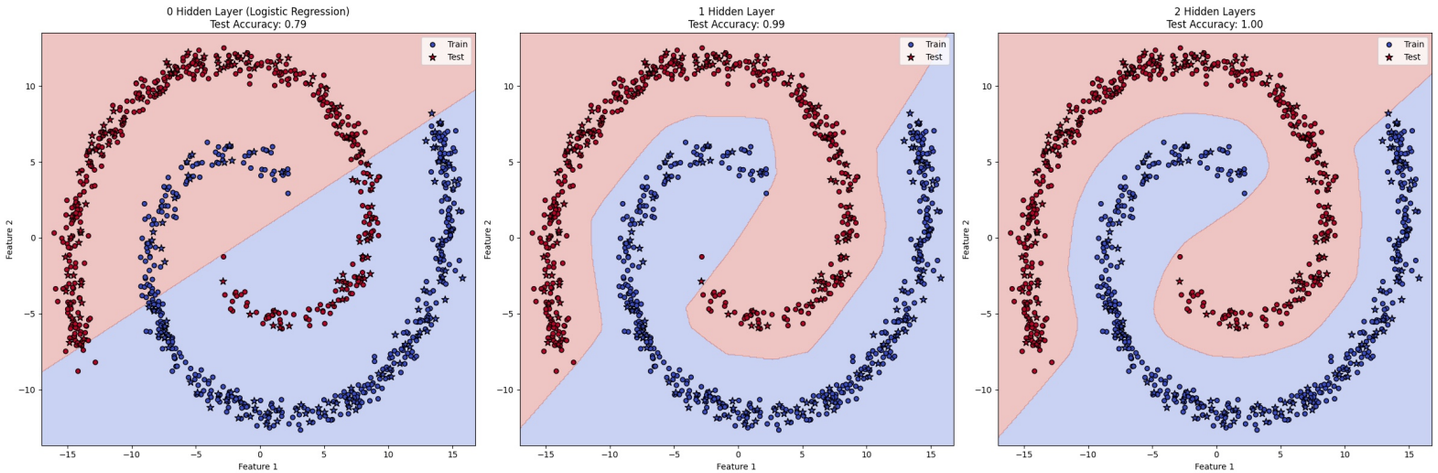
$L = Loss(sigmoid(y))$

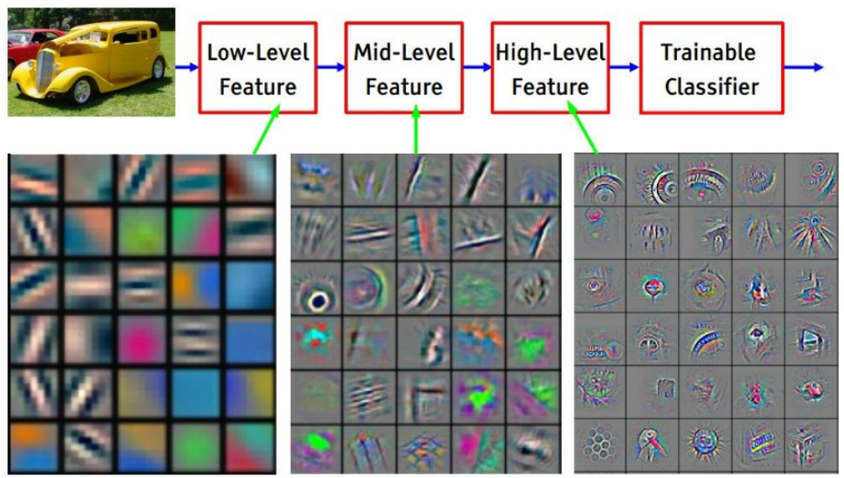
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial y} = ① \cdot ② = \frac{z-t}{z(1-z)} \cdot (1-z)z = z-t$

다른 loss function을 쓰더라도 supervised learning이기만 하면 (pred − truth)에 적당한 LR을 곱해서 gradient update 할 수 있다? 그렇다!

Loss is whatever function we've decided to use to optimize the parameters.

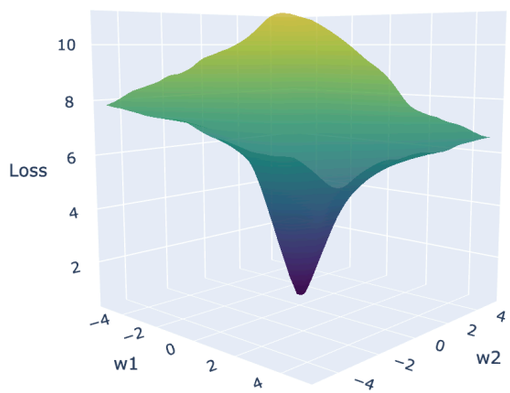
왜 layer들을 deep 하게 쌓아야 하나

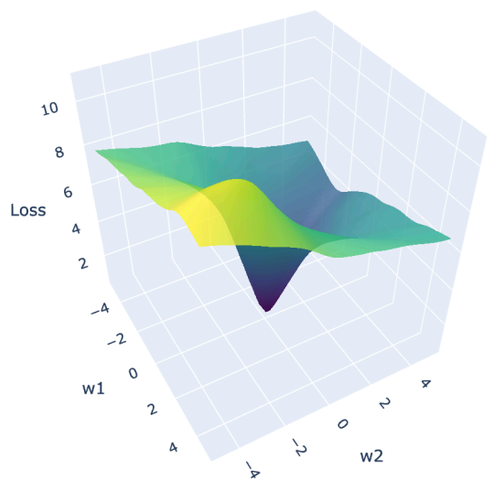


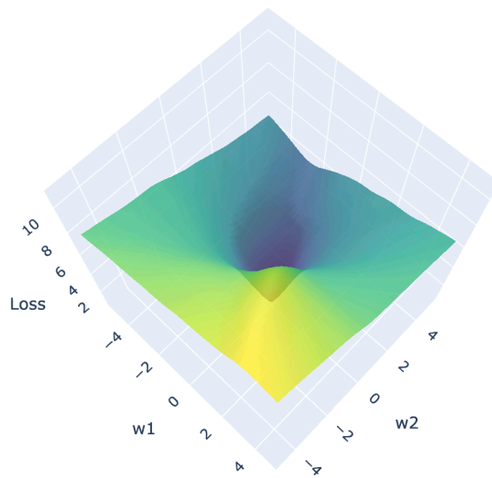


복잡한 데이터의 패턴인식이 가능해짐

이전 슬라이드 spiral dataset에 대해 weight 2개 (w_1과 w_2)가 있는 하나의 layer를 학습시켰을 때 나타나는 loss surface



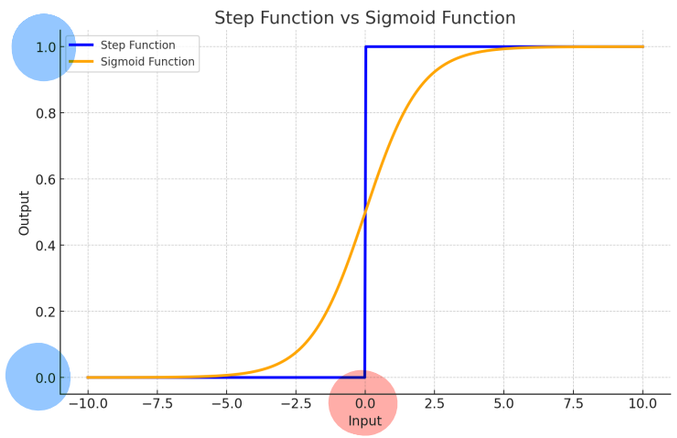




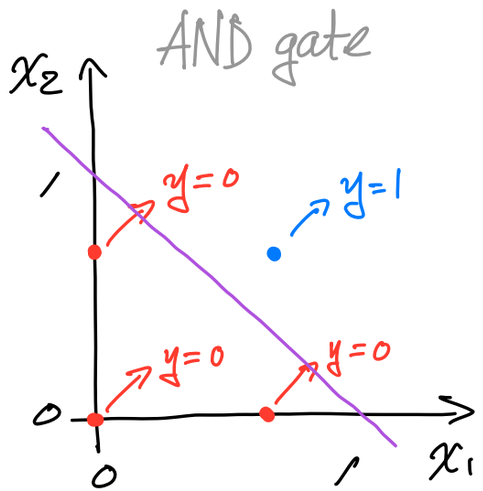
이 경우엔 dimension이 2 이지만 weight의 갯수만큼 dim이 생기게 되고, dim이 늘어날수록 local minima에 갇히지 않은

왜 activation function이 필요한가.

왜 sigmoid function을 사용하나.



계단함수 step function의 형태

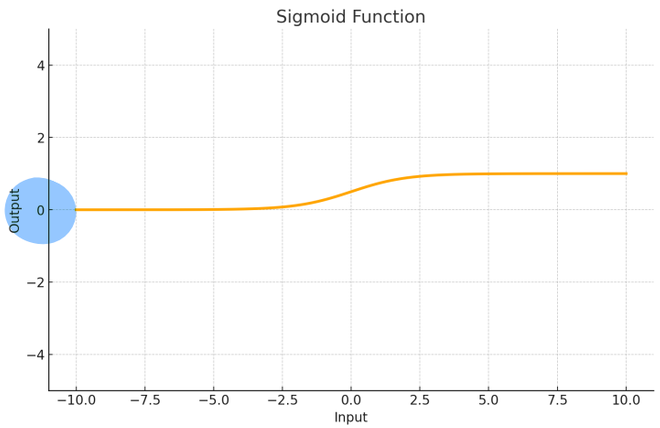


x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

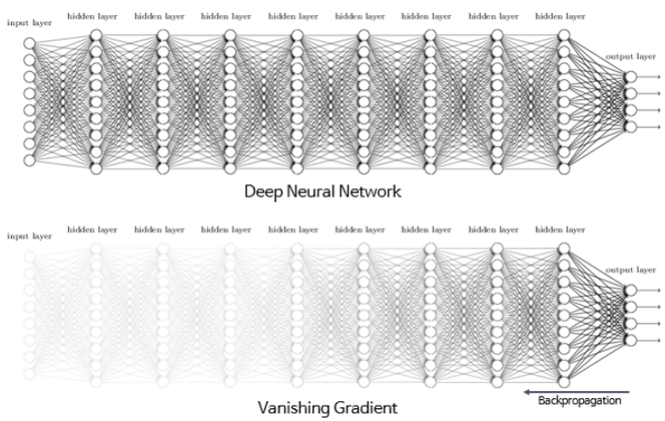
$$y = \text{step}(w_1 \cdot x_1 + w_2 \cdot x_2 + b)$$

without them, the entire network reduces to a linear relationship

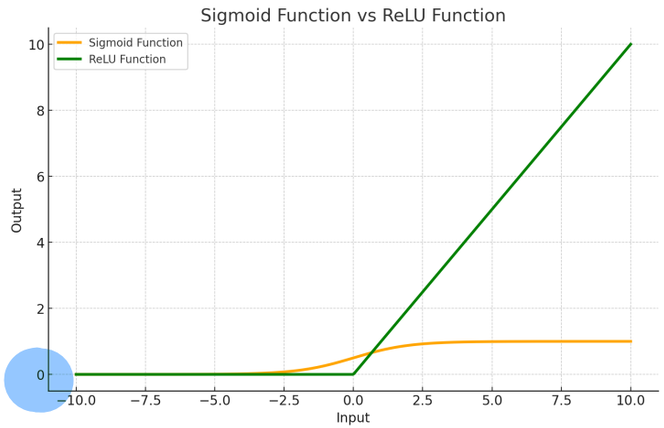
Sigmoid의 문제 - Vanishing gradient.



각 input 값에서 그래프의 기울기



Nonlinearity를 적용해주기만 하면 되더라 - ReLU



$z = \max(0, y)$

Sigmoid를 제대로 활용할 수 있게 weight 값을 작게 유지하는 법이 - Weight decay

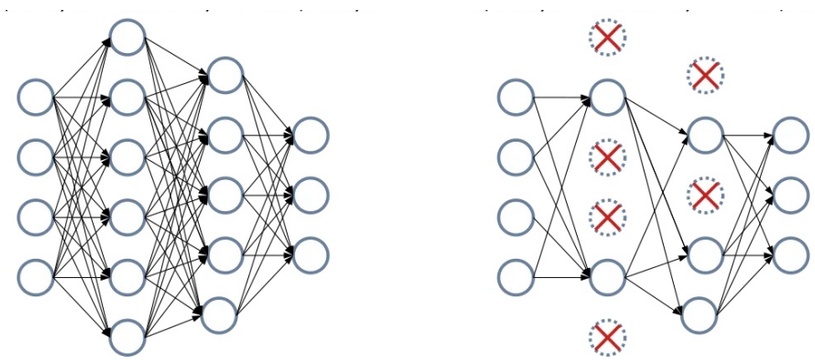
$$L_{after} = L_{before} + \lambda \sum_{i} w_i^2 \quad (L2 \text{ regularization})$$

Again, the loss is whatever function we've decided to use to optimize the parameters.

그냥 매번 layer output의 distribution을 자정 조정하면 안됨? — Batch normalization

Inference 때 train data로 얻은 mean과 variance 정보를 갖고 있어야 함.
Bias가 필요없어짐.

Training 때 뉴런 몇개씩 안쓰니까 더 robust 해지더라 — Dropout



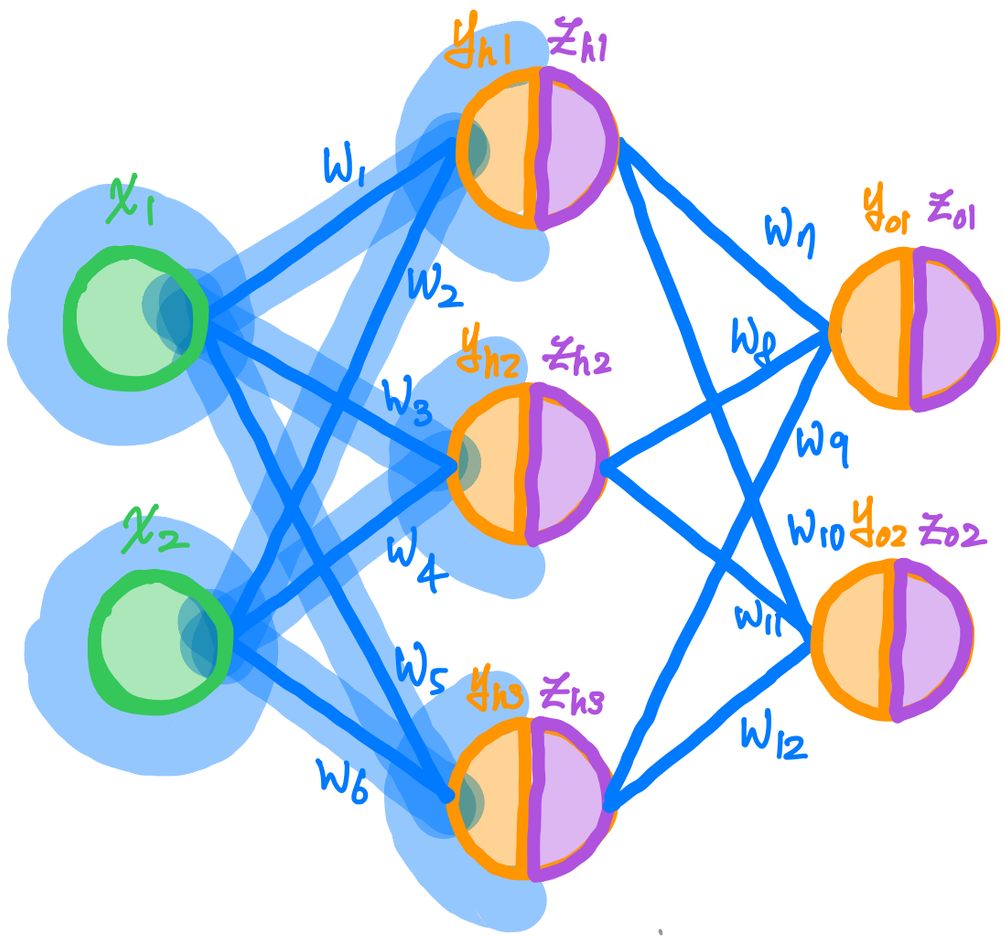
레이어의 구성요소들을 순서대로 쌓아보자

Linear layer
BatchNorm
ReLU
Dropout

```
layer1 = nn.Sequential(
    nn.Linear(input_dim, output_dim),
    nn.BatchNorm(output_dim),
    nn.ReLU(),
    nn.Dropout(dropout_ratio),
)
```

왜 이 순서인지 생각해보자

nn.Linear(input_dim, output_dim)



input_dim x output_dim matrix

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \times \begin{bmatrix} w_1 & w_3 & w_5 \\ w_2 & w_4 & w_6 \end{bmatrix} = \begin{bmatrix} y_{h1} & y_{h2} & y_{h3} \end{bmatrix}$$

m by n. the first number (m here) represents the outermost dimension.
matrix.shape → (2,3)
matrix → [[w_1 w_3 w_5][w_2 w_4 w_6]]

Training은 코드상에서 어떤 형식으로 진행되나

num_epochs 만큼 loop을 돌면서,

각 카테고리에 대한 pred를 생성하고,

pred에 따른 loss 값을 계산하고

weight를 업데이트 한 gradient를 초기화하고

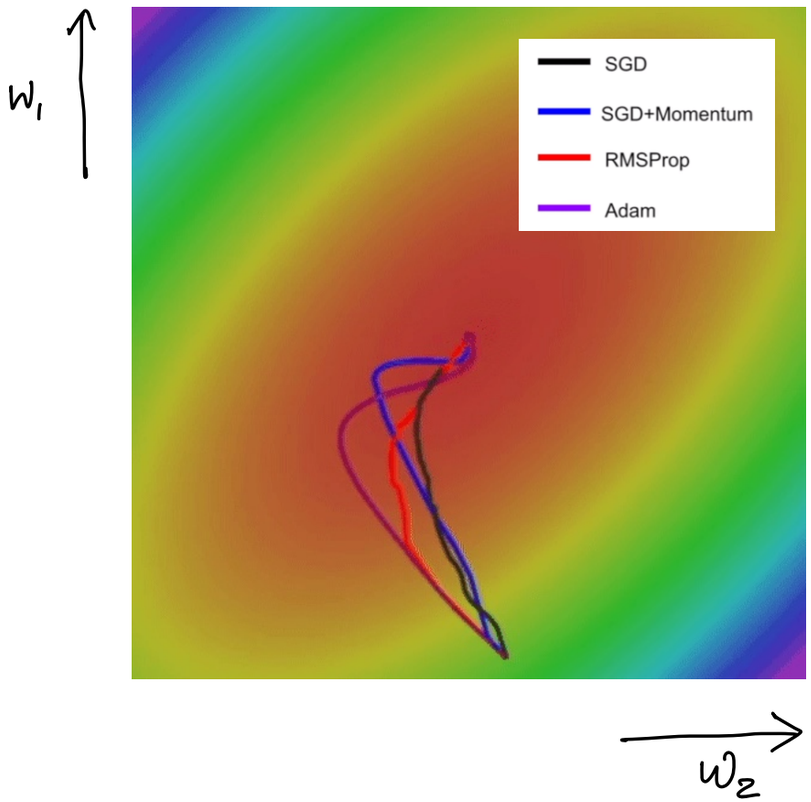
backprop으로 gradient를 계산하고

gradient를 이용해 weight를 업데이트 함.

(hidden layer 없이 하나의 weight 행렬만 있는 경우의 예)

```
# 여기가 데이터입니다
X = torch.FloatTensor(X_np) # 200개의 데이터포인트에 대해 Feature 1과 Feature 2의 2차원 정보가 있겠죠? 그럼 행렬의 사이즈는? --> (200, 2)
y = torch.FloatTensor(y_np) # 200개의 데이터포인트에 대해 0 또는 1의 레이블이 있겠죠? 그럼 행렬의 사이즈는? --> (200,)

# 여기가 가중치입니다
W = torch.zeros(2, requires_grad=True) # (z = w1 * x1 + w2 * x2)이니까 W에는 w1과 w2가 필요하겠죠? 그럼 행렬의 사이즈는? --> (2,)

learning_rate = 0.01
optimizer = optim.SGD([W], lr=learning_rate)

epochs = 100

# 100번 반복합시다
for epoch in range(epochs + 1):

    z = X @ W # 200개의 데이터포인트에 대해 행렬연산을 해줍시다. 아웃풋 행렬의 사이즈는? --> (200,)

    # 여기가 pred입니다. 이러면 forward pass를 한 번 한거죠.
    pred = torch.sigmoid(z) # 200개의 데이터포인트에 대해 시그모이드 함수를 통과시켜줍시다

    # 여기가 loss function입니다
    loss = -(y * torch.log(pred + 1e-7) + (1 - y) * torch.log(1 - pred + 1e-7)).mean() # 200개의 prediction에 대한 loss의 평균값

    # 여기가 backprop입니다. 파이토치에서는,
    optimizer.zero_grad()  # 이전 epoch의 gradient(즉, dL/dw)를 리셋시켜 주는 과정이 필요합니다
    loss.backward()        # 이렇게 gradient를 계산하고,
    optimizer.step()       # weight들을 gradient를 이용해 업데이트 시켜줍니다
```

경사를 내려가는 방법의 대해서 — Optimizer



GD : 전체 train dataset 에 대해 업데이트

SGD : minibatch 에 대해 업데이트

SGD + momentum : 과거의 gradient가 현재의 업데이트에 영향.

Adagrad : 각 dimension 에서 업데이트가 들인 바율로 알아서 조절.

RMSProp : Adagrad의 속도조절가가 시간이 지날수록 decay 함.

Adam : Adagrad + Momentum

파이토치 모델 클래스의 구조, parameter 갯수 확인

```python
class Model(nn.Module):
    def __init__(self):
        super().__init__()

        self.layer1 = nn.Sequential(
            nn.Linear(28*28, 50, bias=False),
            nn.BatchNorm1d(50),
            nn.ReLU(),
            nn.Dropout(0.5),
        )
        self.layer2 = nn.Sequential(
            nn.Linear(50, 10, bias=False),
        )

    def forward(self, x):
        x = x.view(-1, 28*28)
        x = self.layer1(x)
        x = self.layer2(x)

        return x

model = Model()
print(model)
```

```
Model(
  (layer1): Sequential(
    (0): Linear(in_features=784, out_features=50, bias=False)
    (1): BatchNorm1d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): Dropout(p=0.5, inplace=False)
  )
  (layer2): Sequential(
    (0): Linear(in_features=50, out_features=10, bias=False)
  )
)
```

```python
summary(model, input_size=(100, 784))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 50]          39,200
       BatchNorm1d-2                [-1, 50]             100
              ReLU-3                [-1, 50]               0
           Dropout-4                [-1, 50]               0
            Linear-5                [-1, 10]             500
================================================================
Total params: 39,800
Trainable params: 39,800
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.30
Forward/backward pass size (MB): 0.00
Params size (MB): 0.15
Estimated Total Size (MB): 0.45
----------------------------------------------------------------
```

Training loop의 형태

```python
criterion = nn.CrossEntropyLoss().to(device)
optimizer = torch.optim.SGD(model.parameters(), lr=0.1)

epochs = 20
for epoch in range(epochs + 1):

    avg_loss = 0
    total_batch = len(dataloader)

    for X, y in dataloader:
        X = X.view(-1, 28*28).to(device)
        y = y.to(device)

        pred = model(X)

        loss = criterion(pred, y)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        avg_loss += loss / total_batch

    print(f'Epoch: {epoch:4d} Loss: {avg_loss:.6f}')
```

```
Epoch:    0 Loss: 0.576404
Epoch:    1 Loss: 0.416413
Epoch:    2 Loss: 0.378474
Epoch:    3 Loss: 0.352650
Epoch:    4 Loss: 0.335536
Epoch:    5 Loss: 0.324588
Epoch:    6 Loss: 0.316307
Epoch:    7 Loss: 0.304487
Epoch:    8 Loss: 0.299579
Epoch:    9 Loss: 0.293268
Epoch:   10 Loss: 0.288154
Epoch:   11 Loss: 0.282893
Epoch:   12 Loss: 0.280186
Epoch:   13 Loss: 0.275586
Epoch:   14 Loss: 0.277942
Epoch:   15 Loss: 0.270528
Epoch:   16 Loss: 0.267779
Epoch:   17 Loss: 0.267178
Epoch:   18 Loss: 0.265811
Epoch:   19 Loss: 0.258809
Epoch:   20 Loss: 0.260383
```

훈련 시작 전의 loss 확인

$$L = -\sum_{i=1}^{num\_classes} t_i \ln(z_i)$$

정답이 아닌 경우 $t_i = 0$

모델 확률은 $\frac{1}{num\_classes}$

$$L = -\ln\left(\frac{1}{num\_classes}\right)$$

$$L = \ln(num\_classes)$$

$$\ln(10) = 2.30$$

코드 + 과제 파일의 위치 :

github.com / star-bits / sogangparrot