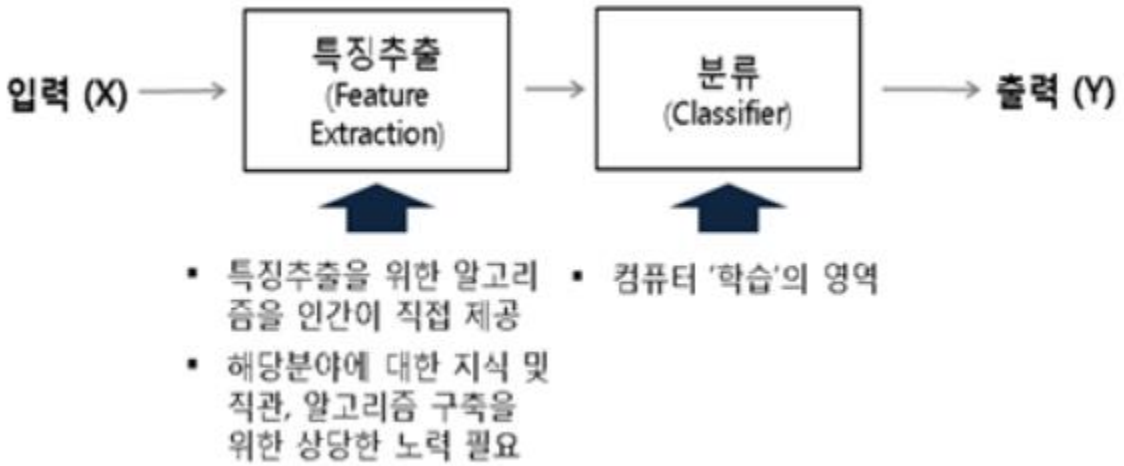
Perceptron

김균엽

Machine Learning

- **machine learning process**
  - 즉, 사람이 직접 추출한 입력 데이터의 feature를 기반으로 정답을 예측하는 것
  - 입력된 feature과 정답 사이의 pattern을 사람이 설정한 식(알고리즘)을 통해 표현하고자 하는 것

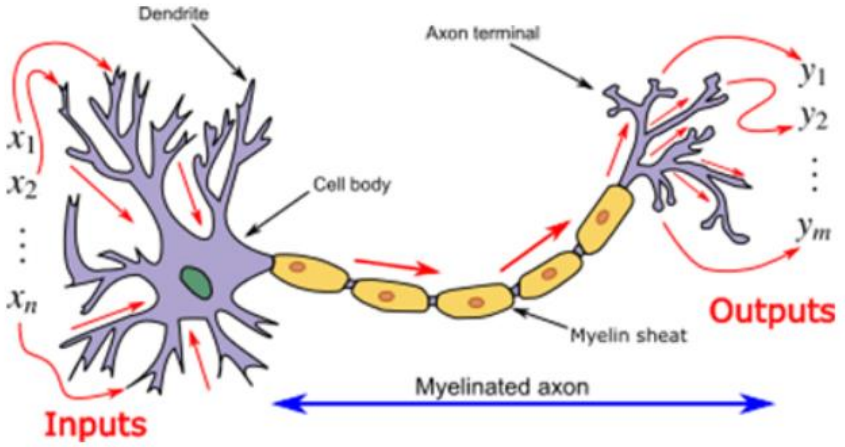


Deep Learning

- **machine learning process**
  - 딥러닝은 블랙박스 모델과 같이 딥러닝 모델에 전체 데이터를 입력으로 넣고 결과를 도출하는 과정이다
  - 모델과 데이터를 구축하는 과정이후 추가적인 프로세스가 필요하지 않는다.
  - 기존 머신러닝의 특징 추출과정또한 학습이 되도록 함으로써 추출과정을 거치지 않는다
  - 이는 사람이 직접 특징과 식을 결정하지 않기때문에 머신러닝의 이슈들을 해결할 수 있다.
  - 매우 고차원의 hypothesis를 블랙박스모델로 사용하고 GD를 통해 in-out만을 사용하여 블랙박스 모델을 사용한다.

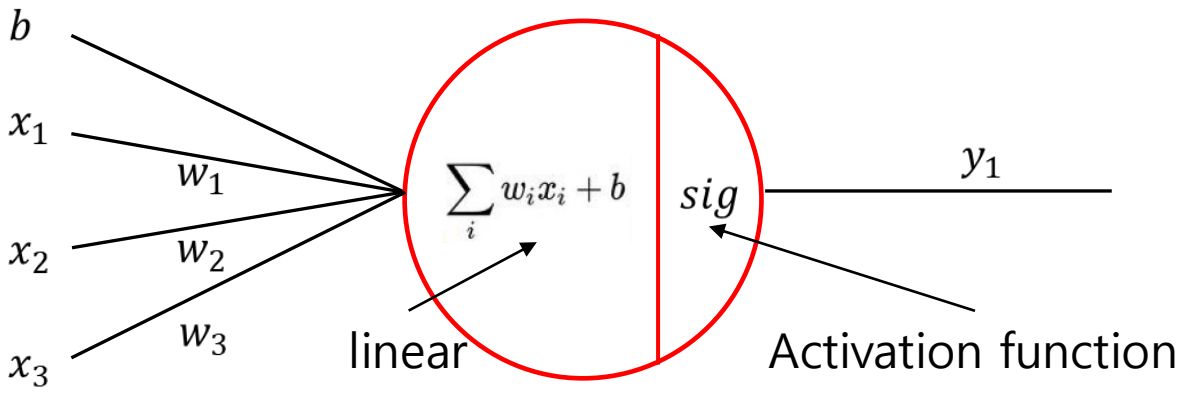
Perceptron

- **neuron**
  - 인간의 뉴런은 이전뉴런에서 보내온 정보를 세포체(cell body)에 저장한다. 이후 저장된 값이 일정 수준을 넘어가면 axon을 통해 전달물질을 내보낸다.
  - Neural network는 이러한 인간의 뉴런을 수학적으로 재구성한 perceptron을 이용한 딥러닝이다.



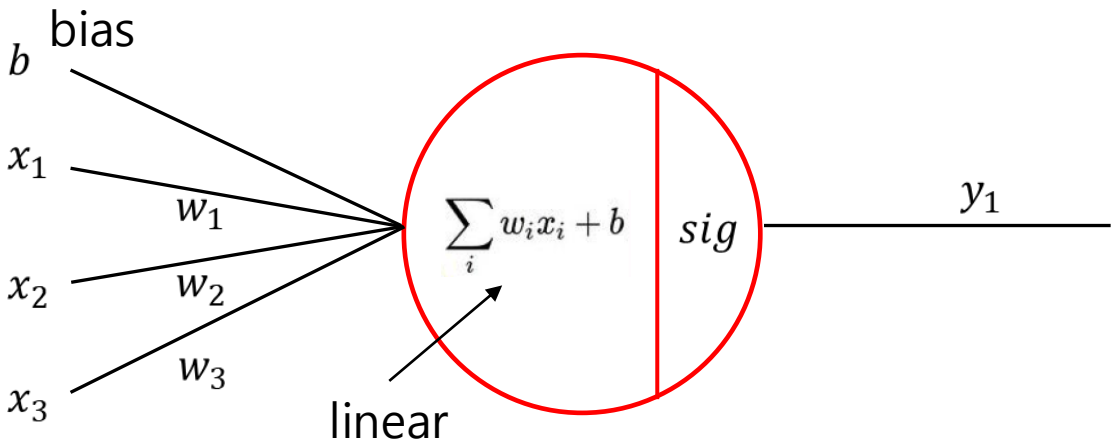
Perceptron

- **Perceptron**
  - 이전뉴런에서 보내온정보(input data)를 세포체에 저장(linear)한후 일정수준이 넘어서면 (activate function)을 통해 전달물질을 전달한다.
  - Input data들에 대해서 linear한 함수(linear regression)를 통해 정보를 취합하고 이 결과에 activation function을 취한 후 결과를 반환한다.



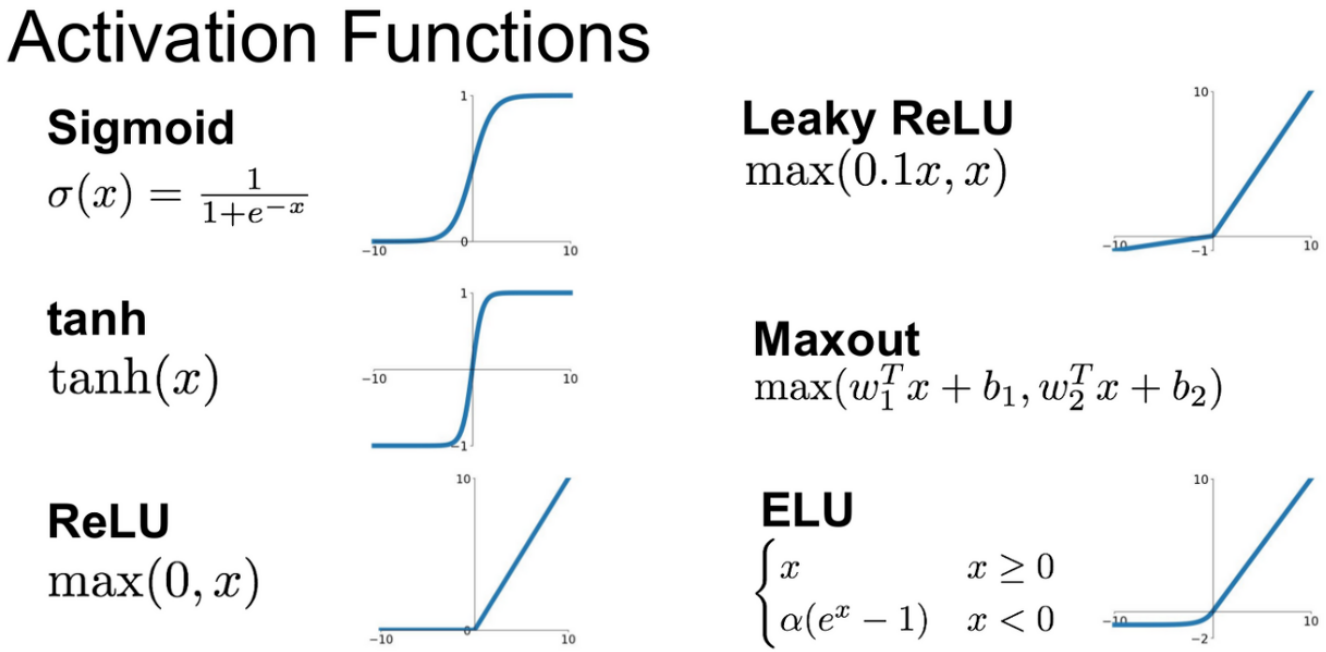
Perceptron

- **Perceptron**
  - Perceptron은 들어온 입력에 각각에 parameter를 곱한 후 더하는 linear를 통해 결과를 도출한다.
  - 이후 linear결과에 activation function을 취해주어 최종 결과를 도출한다
  - 이중 linear의 parameter는 cost function에 의해 학습된다
  - 이를 수식으로 나타내면 activationFunction(WX+b)이다.

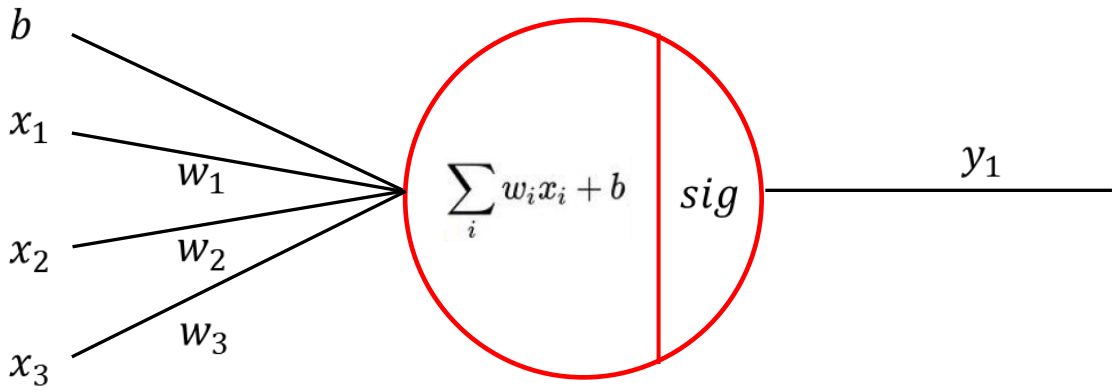


Perceptron

- **Perceptron**
  - Activation function은 perceptron의 결과 도출 직전에 적용되는 non-linear function이다.
  - Activation function은 모델의 복잡도를 높이기 위함이며 결과를 non-linaer하게 만듦으로써 classification과 같은 문제를 해결할 수 있다.
  - Activation function은 아래의 여러 함수들을 사용할 수 있다.



Perceptron



x1	x2	x3	Y
100	30	50	

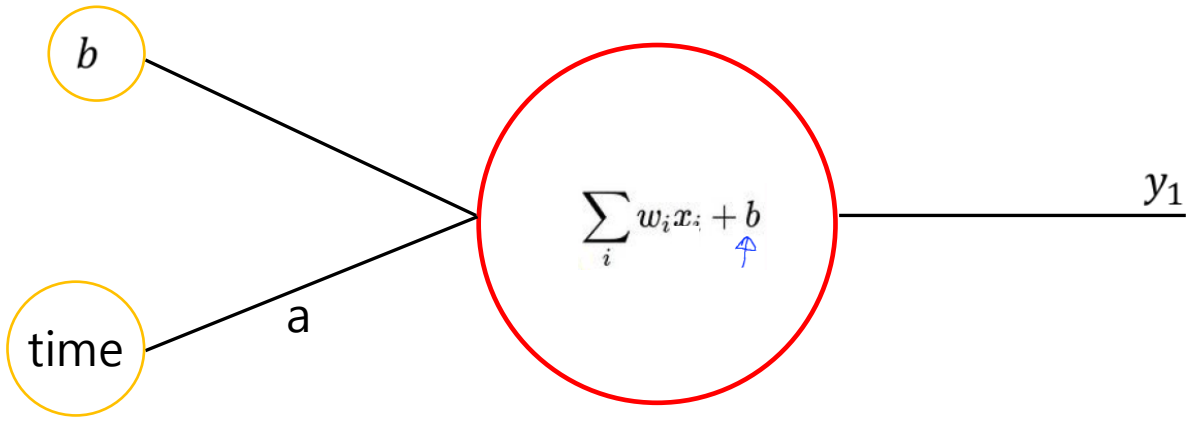
w1	w2	w3	b
1	2	3	1

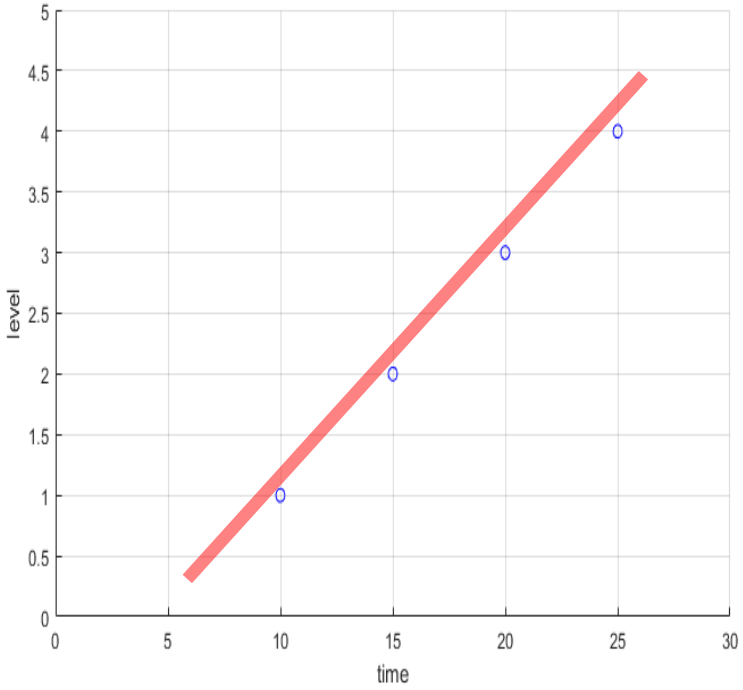
$$sig(w1 * x1 + w2 * x2 + w3 * x3 + b * 1) = predict$$

$$sig(100 * 1 + 30 * 2 + 50 * 3 + 1) = sig(311)$$

Perceptron to linear regression

기존 Linear regression hypothesis
Level = a * time + b

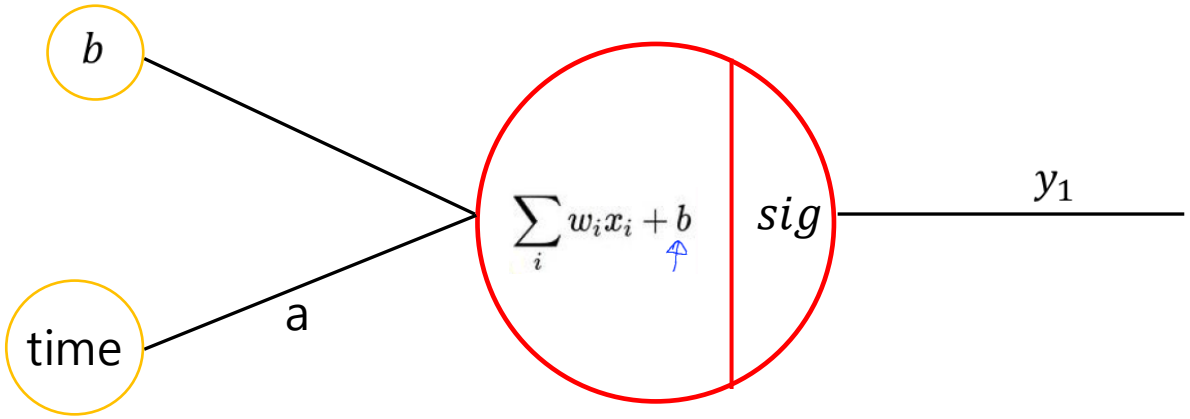


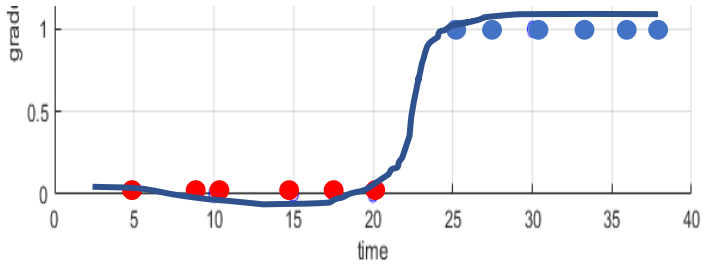


loss function = $\frac{1}{N}\sum_{i=1}^{N}(yi - y)^2$

Perceptron to Logistic regression

$$\text{result} = \frac{1}{1-e^{-(time * a + b)}}$$



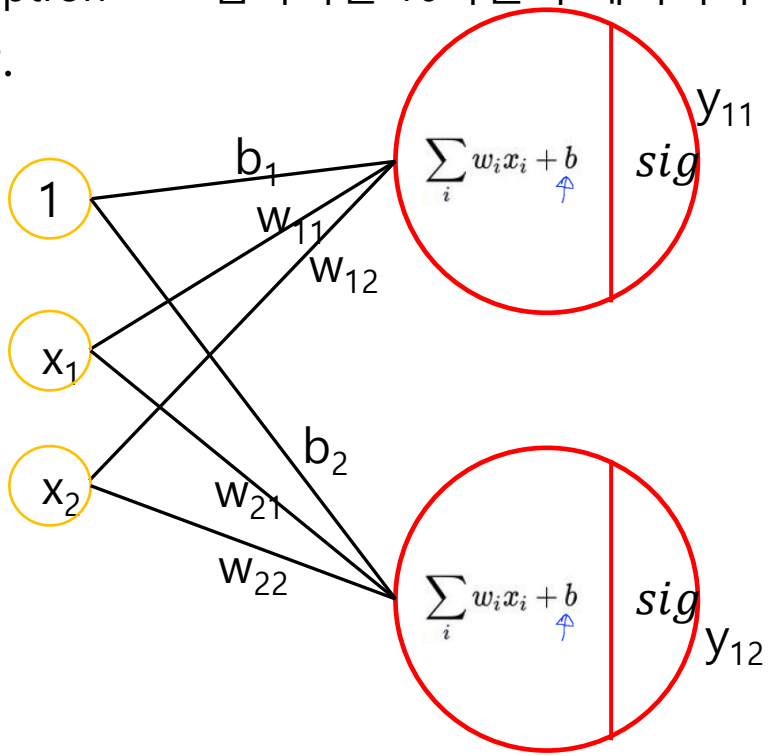


cost function

$$C(H(x), y) = -\log\big(H(x)\big) - (1-y)\log\big(1-H(x)\big)$$

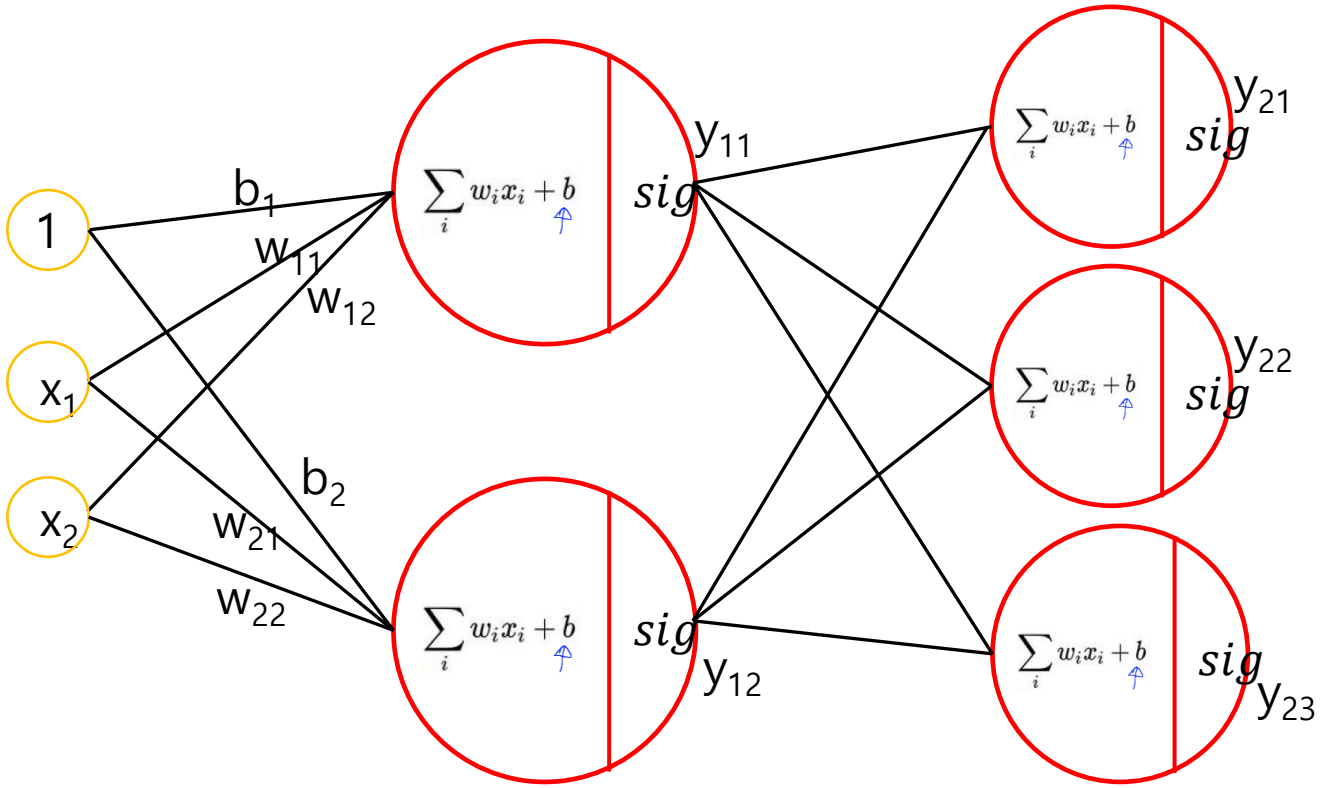
Deep learning(DNN)

- **Deep leaning**
  - 일반적으로 perceptron은 하나의 데이터에서 여러 개의 다른 perceptron을 통해 결과를 도출한다.
  - 이를 통해 입력 데이터의 차원수를 늘리거나 줄이는데 용이하다.
    - 10개의 data가 2개의 perceptron으로 입력되면 10차원의 데이터가 perceptron을 거친 후 2차원이 됨
  - 이를 한 층의 layer라 한다.



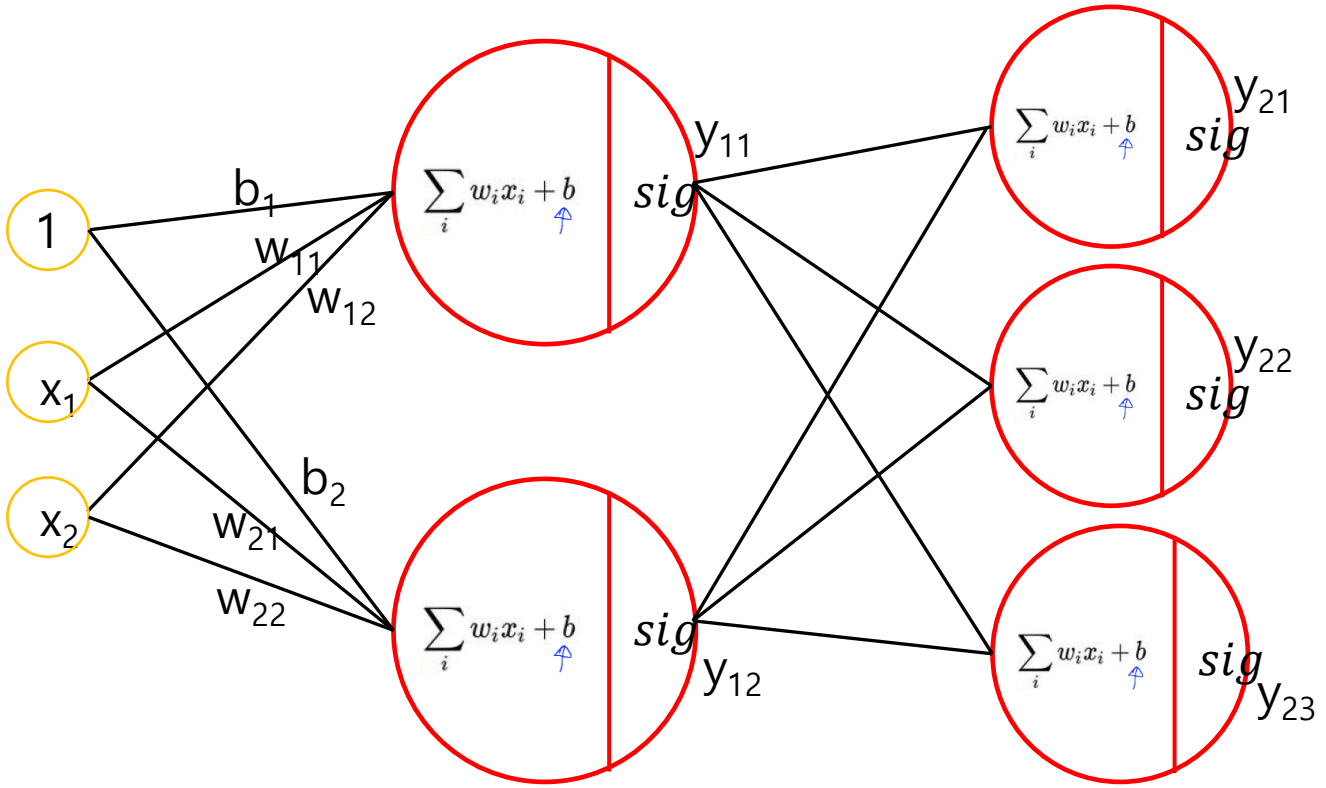
Deep learning(DNN)

- **Deep leaning**
  - 또한, 딥러닝은 여러 layer를 쌓아서 전체 모델을 구성한다.
  - 이전 layer의 perceptron의 output을 다음 layer의 perceptron의 input으로 사용된다.

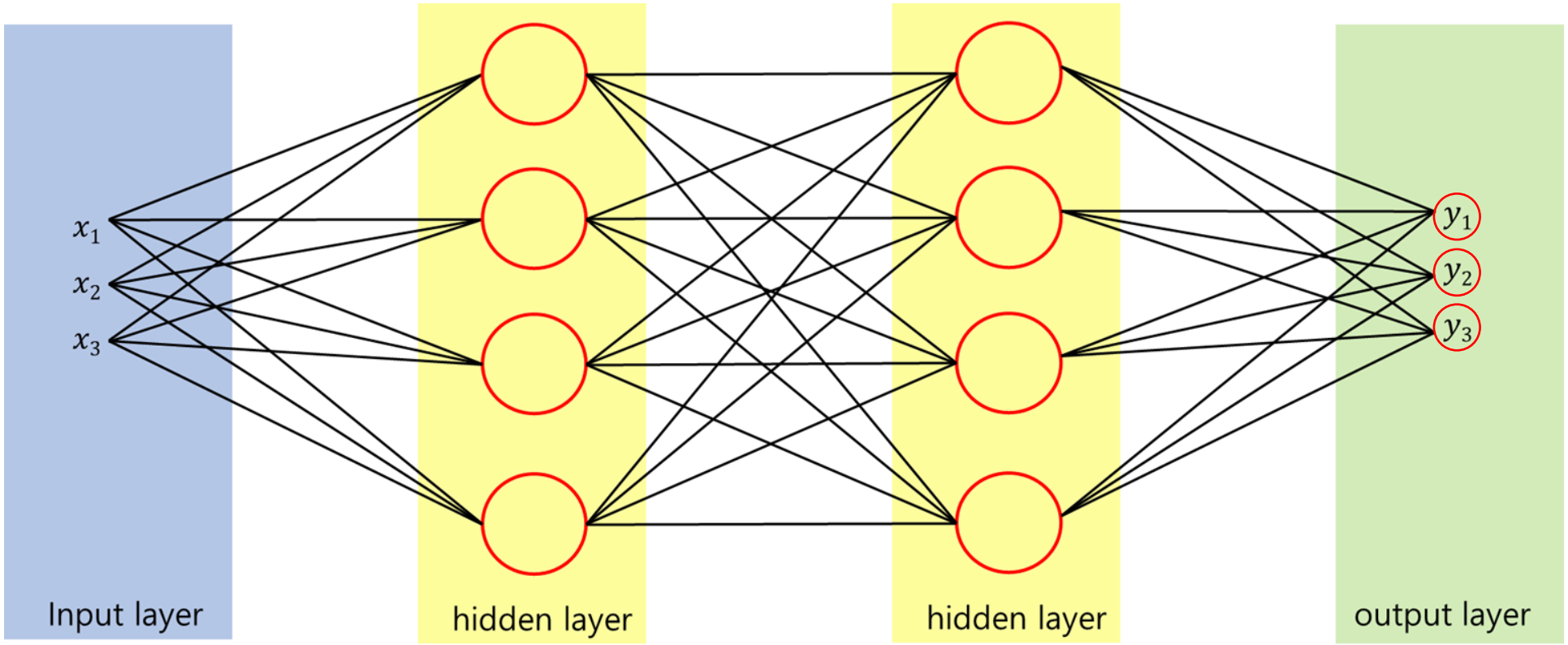


Deep learning(DNN)

- **Deep leaning**
  - 아래의 예시는 2개의 perceptron을 가진 1st-layer와 3개의 perceptron을 가진 2nd layer를 결합한 모델이다.
  - Input data로는 2차원 데이터가 입력되며 1st-layer이후 2차원의 output 이 나오며 2nd-layer이후 3차원의 output 이 도출된다.

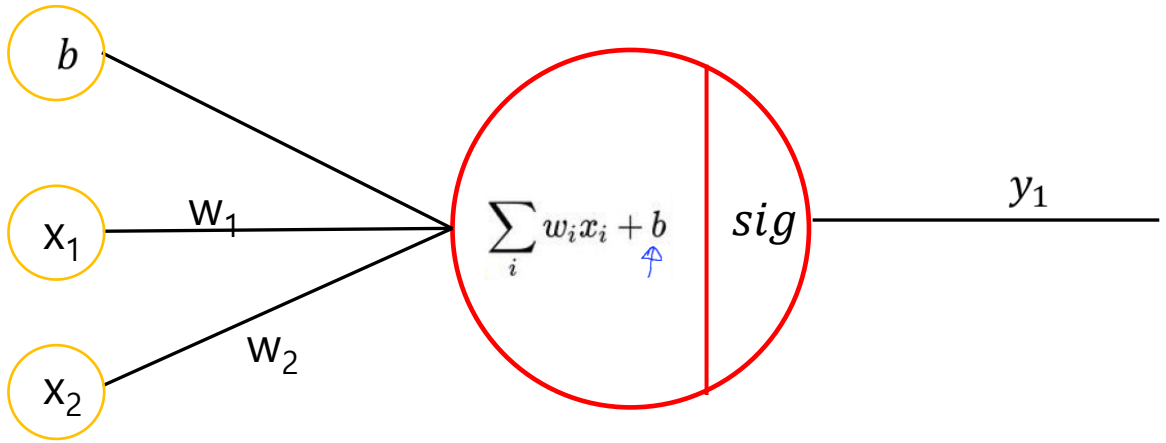


MLP(Multi-Layer-Perceptron)



Hidden layer가 1이상인 perceptron 계층

Matrix product



$$w1 * x1 + w2 * x2 + w3 * x3 + b * 1$$

$$\downarrow$$

$$[x_1 \quad x_2 \quad 1] \begin{bmatrix} w_1 \\ w_2 \\ b_1 \end{bmatrix} = [y_1]$$

$$\downarrow$$

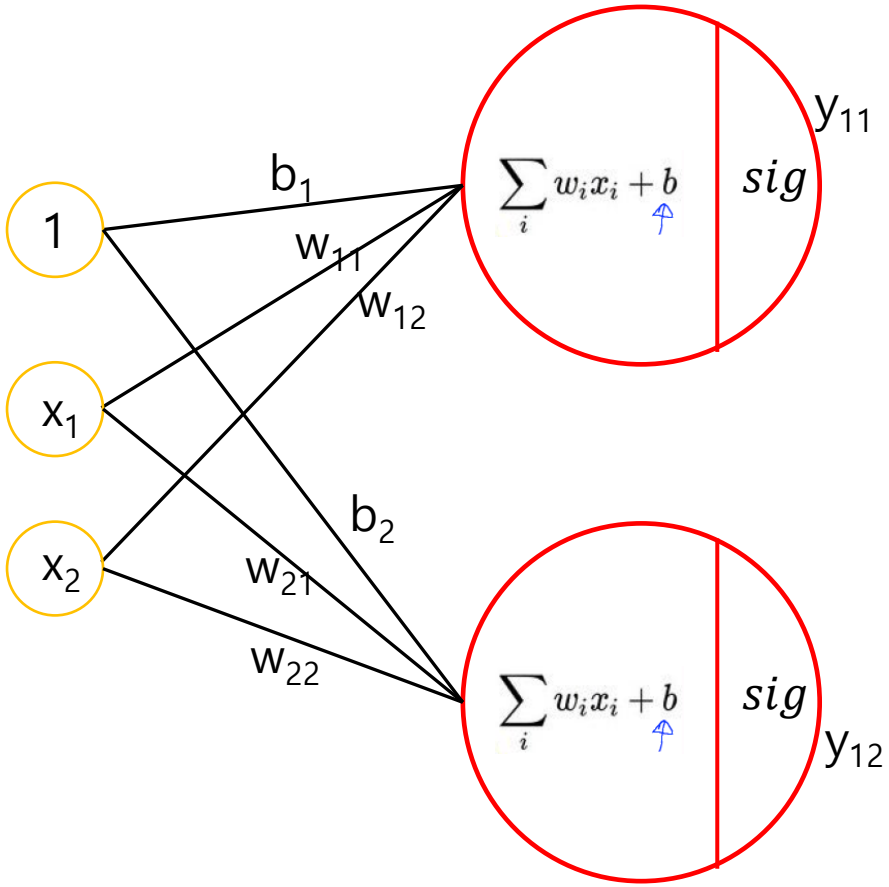
$$X * W^T + b$$

$$X * W + b$$

지정된 입력 값인 X에 정답 Y에 근사한 예측 값을 얻어내기 위해
파라미터 W를 gradient descent를 이용하여 최적 W 유도

Matrix product

Perceptron연산은 일반적으로 행렬로 표시



$$w11 * x1 + w12 * x2 + b1 * 1 = y11$$
$$w21 * x1 + w22 * x2 + b2 * 1 = \text{y12}$$

↓

$$[x_1 \quad x_2 \quad 1] \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ \text{b}_1 & \text{b}_2 \end{bmatrix} = [y_1 \quad y_2]$$

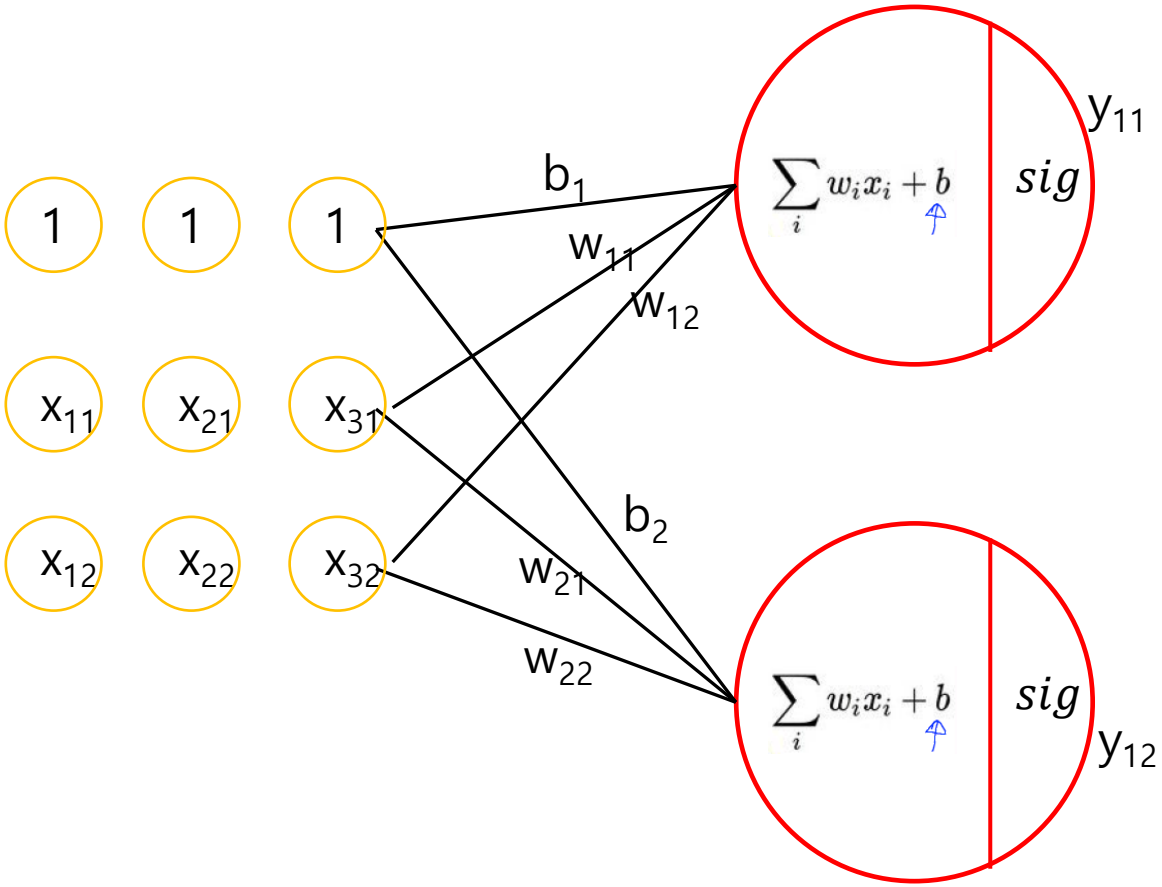
↓

$$\text{X} * \text{W}^{\text{T}} + \text{b}$$

$$\text{X} * \text{W} + \text{b}$$

Matrix product(batch processing)

여러 개의 데이터를 한번에 계산



$$w11 * x1 + w12 * x2 + b1 * 1 = y11$$
$$w21 * x1 + w22 * x2 + b2 * 1 = y12$$

↓

$$\begin{bmatrix} x_{11} & x_{12} & 1 \\ x_{21} & x_{22} & 1 \\ x_{31} & x_{32} & 1 \end{bmatrix} \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ b_1 & b_2 \end{bmatrix} = \begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \\ y_{31} & y_{32} \end{bmatrix}$$

↓

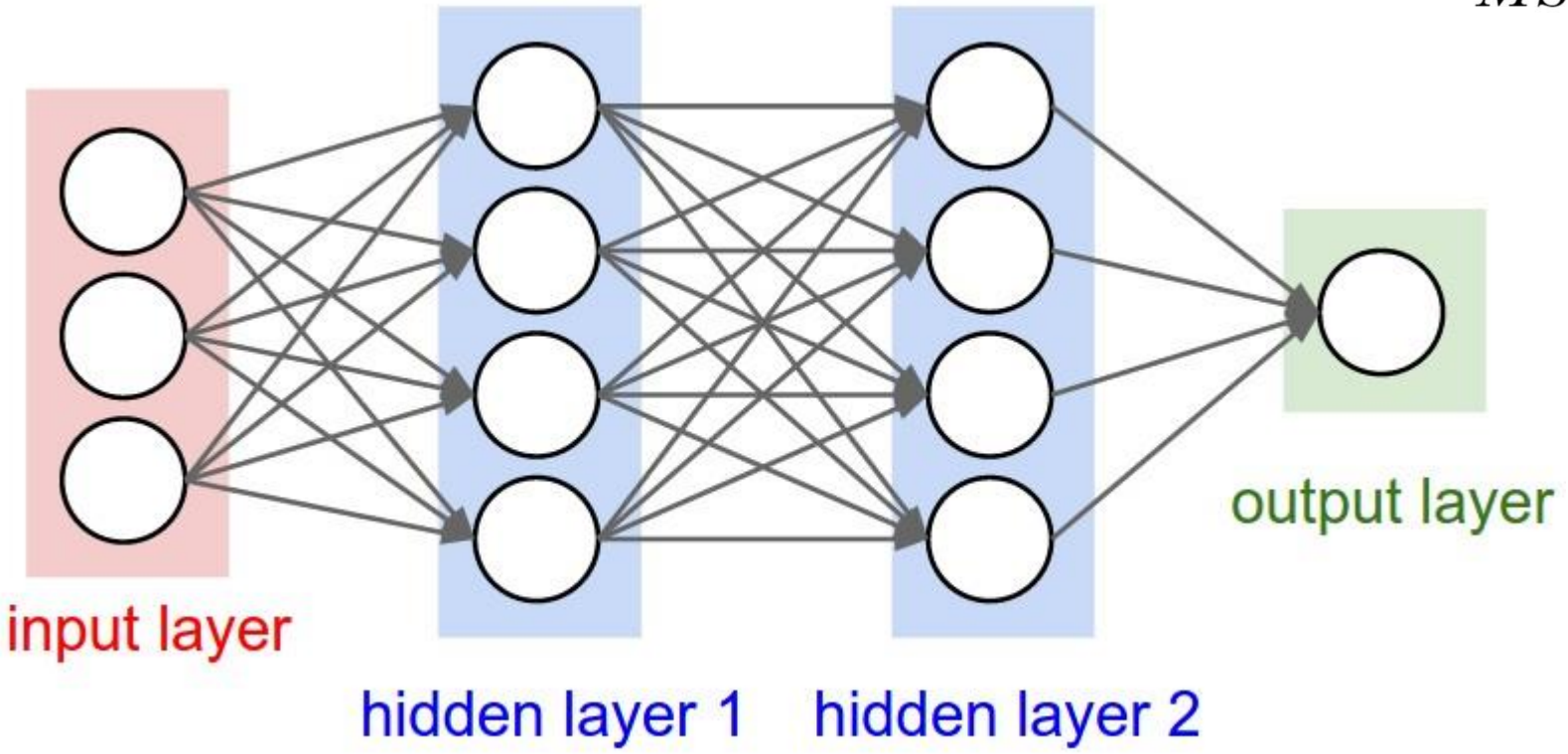
$$X * W^T + b$$

$$X * W + b$$

Regression with ANN

어떤 데이터에 대한 regression problem에 대해서는
MLP를 통해 복잡한 식을 구현하고 1개의 아웃풋에서 결과를 받음

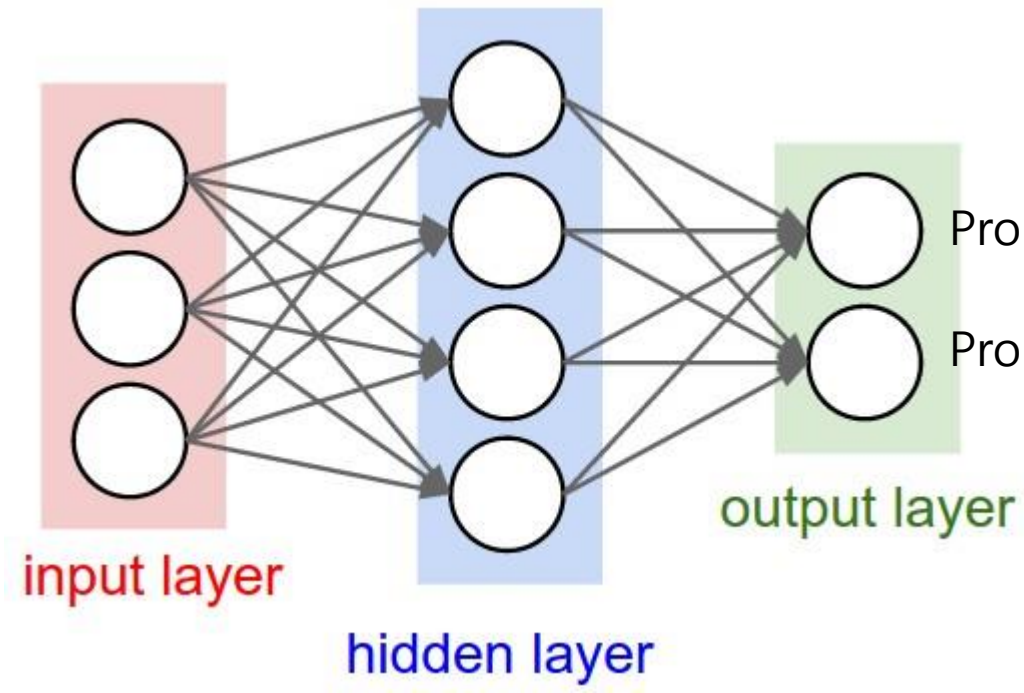
$$MSE = \frac{1}{n}\Sigma\underbrace{\left(y - \hat{y}\right)^2}_{\text{The square of the difference between actual and predicted}}$$



Classification with ANN

어떤 여러 데이터에 대한 O/X를 가르는 binary classification의 경우에는
아웃풋에서 두개의 결과를 받아서 각각에 대한 확률을 의미하도록 학습한다.

$$\text{Binary Cross-entropy} = -\Big(\underbrace{p(x)\cdot\log q(x)}_{\text{This cancels out if the target is 0}} + \underbrace{(1-p(x))\cdot\log(1-q(x))}_{\text{This cancels out if the target is 1}}\Big)$$



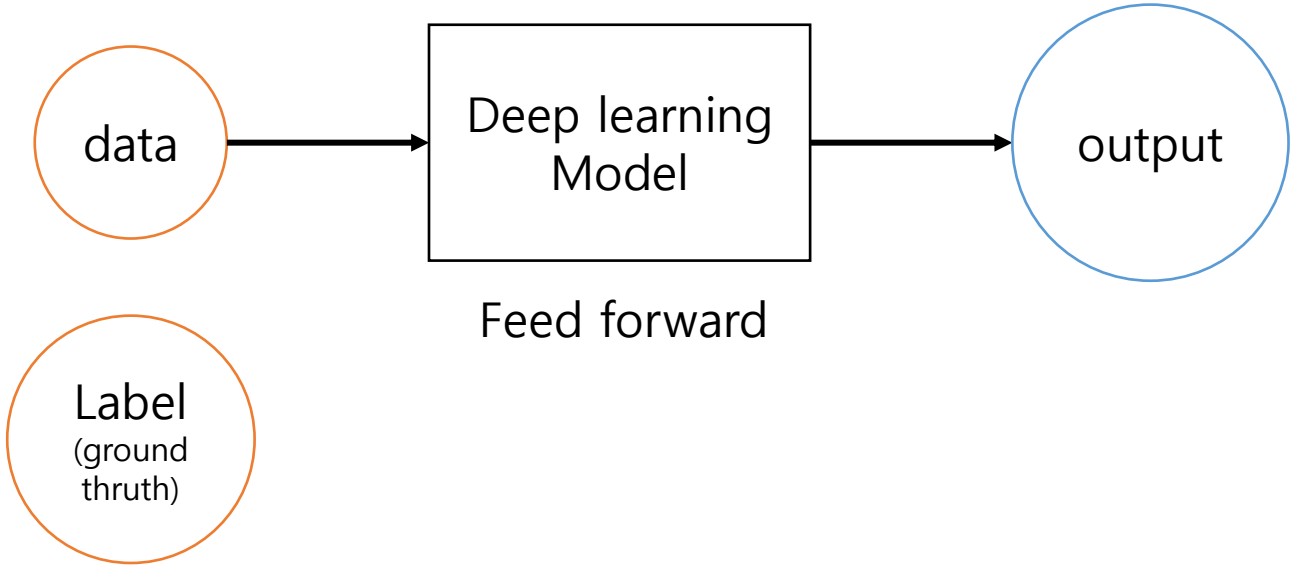
TARGET	PREDICTION
1	0.5
0	0.3
0	0.2

$$\text{Cross-entropy} = -\sum_{x} p(x)\cdot\log q(x)$$

target의 확률이 1이 되도록 유도

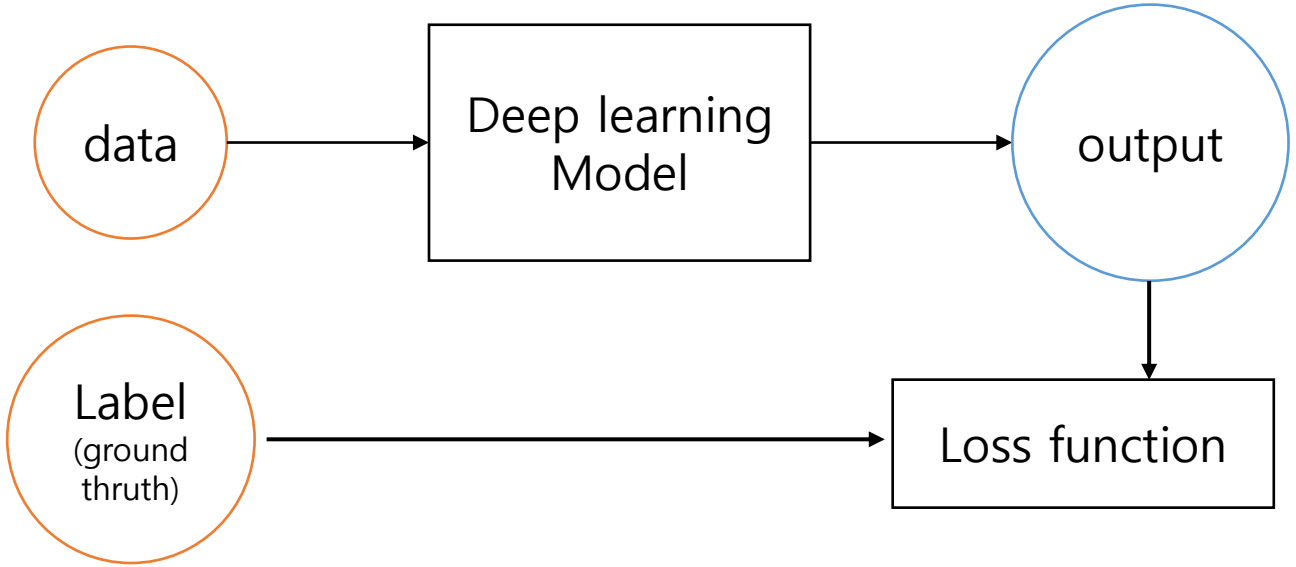
Deep Learning training

- **Deep leaning**
  - MLP를 이용해 생성된 모델은 train dataset을 이용해서 모델을 학습한다.
  - **Feed-forward: Train data를 batch 단위로 분할 후 MLP model에 입력하여 결과를 얻는다.**
  - Feed-forward를 통해 얻은 예측값과 실제 정답을 통해 loss를 연산한다.
  - Back-propagation: 연산된 loss를 기반으로 뒤쪽 레이어부터 편미분하여 gradient를 구하는 과정(chain rule)을 이용한 고차원의 hypothesis 편미분은 뒤에서부터 미분 결과를 도출함
  - 이후 연산된 gradient를 parameter에 update하여 optimization한다.



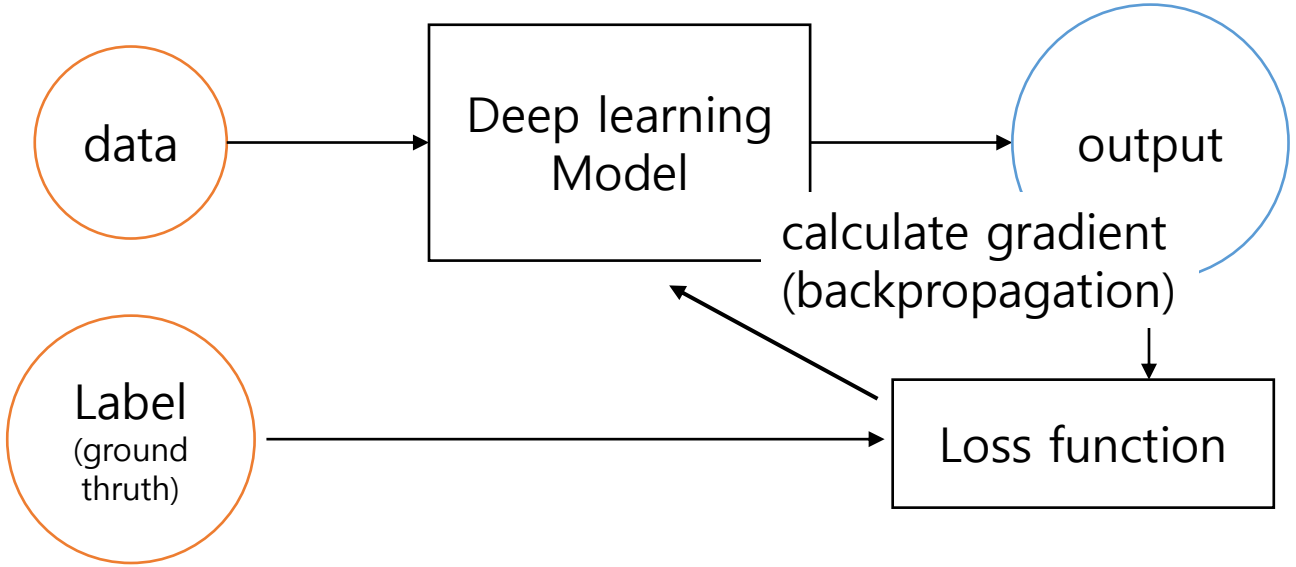
Deep Learning training

- **Deep leaning**
  - MLP를 이용해 생성된 모델은 train dataset을 이용해서 모델을 학습한다.
  - Feed-forward: Train data를 batch 단위로 분할 후 MLP model에 입력하여 결과를 얻는다.
  - **Feed-forward를 통해 얻은 예측값과 실제 정답을 통해 loss를 연산한다.**
  - Back-propagation: 연산된 loss를 기반으로 뒤쪽 레이어부터 편미분하여 gradient를 구하는 과정(chain rule)을 이용한 고차원의 hypothesis 편미분은 뒤에서부터 미분 결과를 도출함
  - 이후 연산된 gradient를 parameter에 update하여 optimization한다.



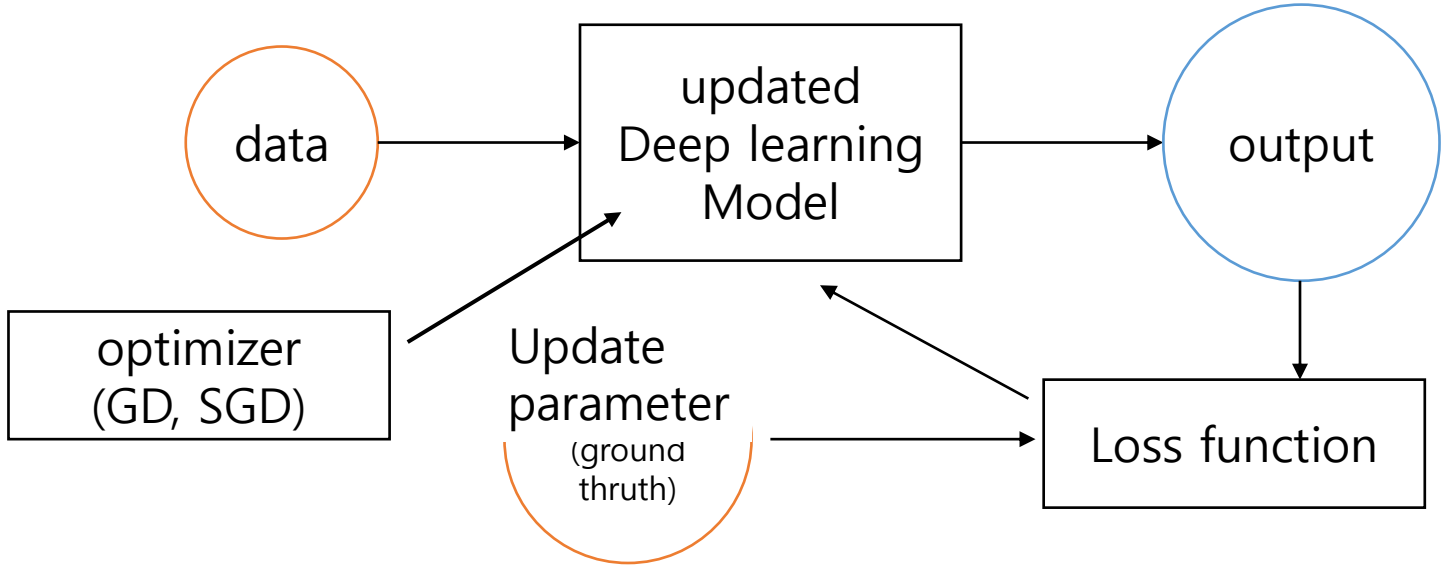
Deep Learning training

- **Deep leaning**
  - MLP를 이용해 생성된 모델은 train dataset을 이용해서 모델을 학습한다.
  - Feed-forward: Train data를 batch 단위로 분할 후 MLP model에 입력하여 결과를 얻는다.
  - Feed-forward를 통해 얻은 예측값과 실제 정답을 통해 loss를 연산한다.
  - **Back-propagation: 연산된 loss를 기반으로 뒤쪽 레이어부터 편미분하여 gradient를 구하는 과정(chain rule)을 이용한 고차원의 hypothesis 편미분은 뒤에서부터 미분 결과를 도출함**
  - 이후 연산된 gradient를 parameter에 update하여 optimization한다.



Deep Learning training

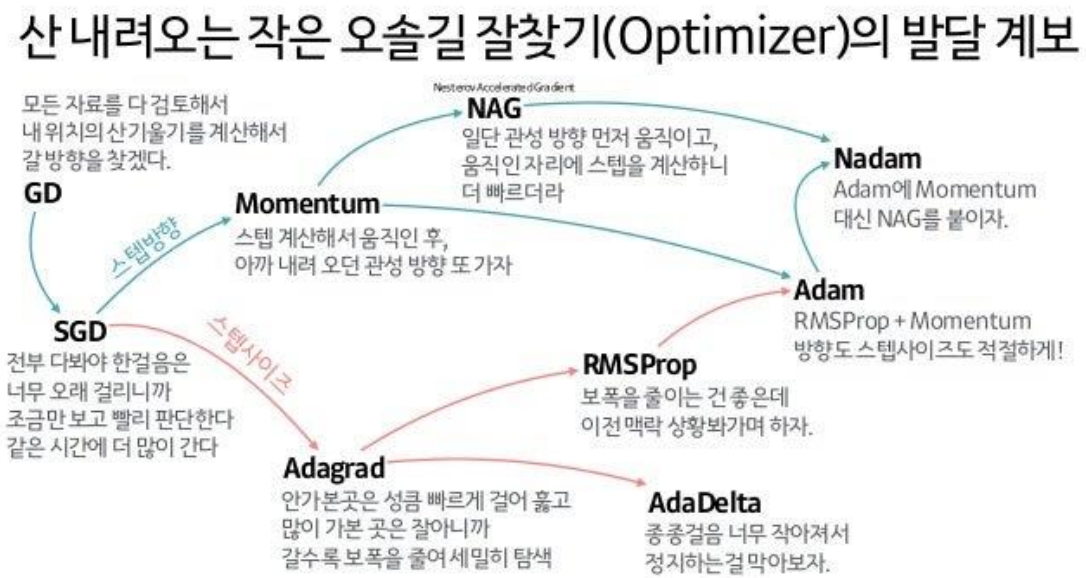
- **Deep leaning**
  - MLP를 이용해 생성된 모델은 train dataset을 이용해서 모델을 학습한다.
  - Feed-forward: Train data를 batch 단위로 분할 후 MLP model에 입력하여 결과를 얻는다.
  - Feed-forward를 통해 얻은 예측값과 실제 정답을 통해 loss를 연산한다.
  - Back-propagation: 연산된 loss를 기반으로 뒤쪽 레이어부터 편미분하여 gradient를 구하는 과정(chain rule)을 이용한 고차원의 hypothesis 편미분은 뒤에서부터 미분 결과를 도출함
  - **이후 연산된 gradient를 parameter에 update하여 optimization한다.**



Deep Learning training

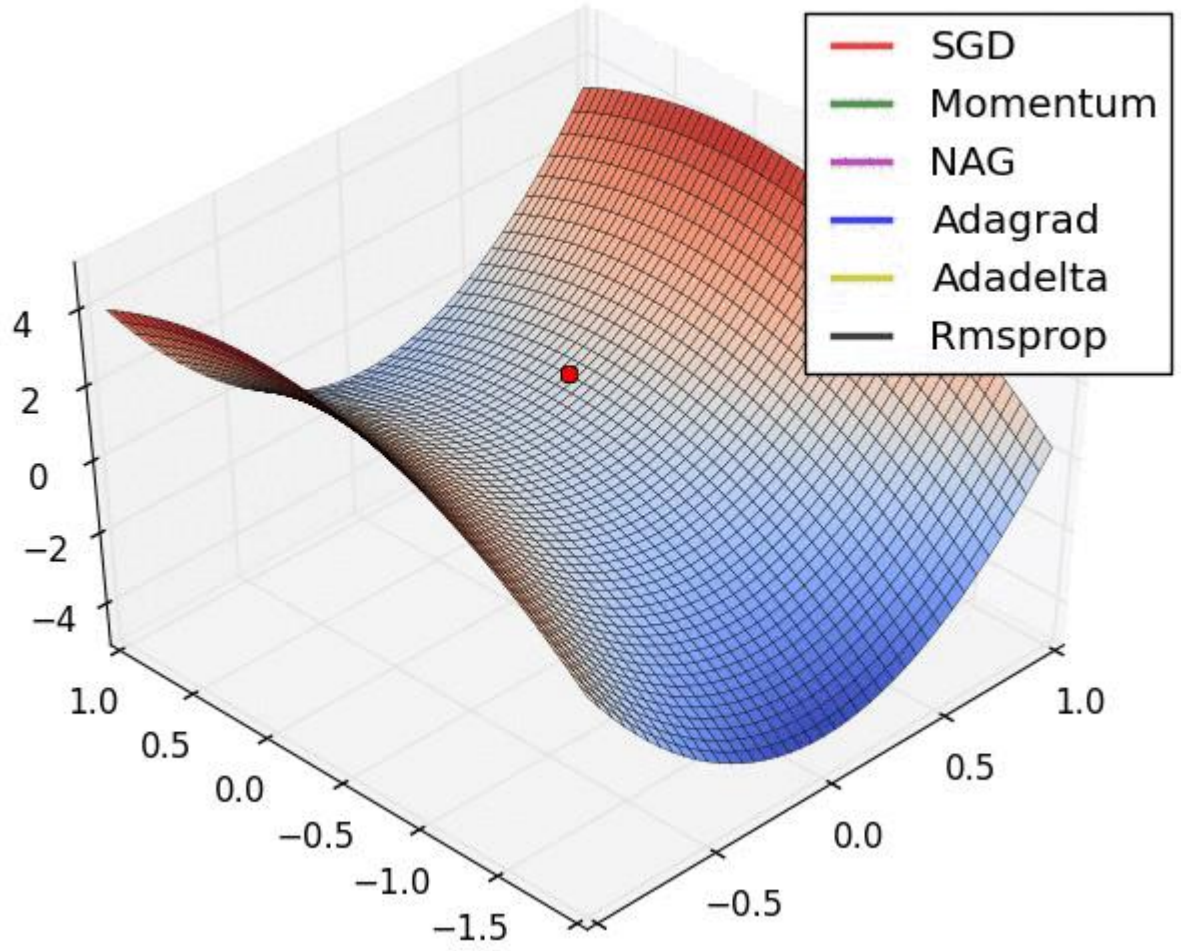
- **Optimizer**
  - Loss를 통해 각 parameter에 대한 gradient를 얻은 후 해당 gradient를 paramter에 update 하는 방법론
  - 다음 식과 같은 GD가 가장 기본적인 방법론이며 이후 여러가지 문제점을 보완한 방법론이 제안되었음

$$W := W - \alpha \frac{\partial}{\partial W} cost(W)$$



Deep Learning training

- **Optimizer**



용어정리

용어	의미
MLP, feed forward network, fully connected network	Perceptron으로 이루어진 layer
Loss function, cost function, objective function	Loss function
Label, ground truth, gold	정답데이터
Train dataset	학습을 위한 dataset으로 gradient를 연산하고 parameter를 업데이트하는데 쓰임
development dataset, validation dataset	각 epoch에 대한 모델이 얼마나 잘 예측하는지만을 평가 parameter를 업데이트하지 않음
Test dataset	최고의 성능을 도출한 모델을 평가하기 위한 데이터셋

Pytorch
deep learning

김균엽

Pytorch deep learning process

- **pytorch**
  - Facebook에서 개발한 deep learning framework
  - GPU를 사용한 tensor연산을 지원하여 deep learning을 빠르게 연산할 수 있다.
  - Class기반의 구현을 제공하여 쉽게 익힐 수 있다.
  - **주요 구성 요소:**
    - **nn.Module – model의 각 layer를 선언하고 feed forward를 진행하는 class**
    - **utils.data.Dataset, utils.data.DataLoader – 학습을 위한 data를 tensor형태로 변환해고 batch단위로 묶는 class**
    - **nn.(lossfunction) – nn.Module을 이용한 예측값과 정답을 비교하여 loss를 구하고 gradient를 계산하는 class**
    - **Utils.optim.(optimizer) - gradient를 이용하여 parameter에 업데이트하는 class**
    - **nn.linear – perceptron에서 linear에 해당하는 기능을 하는 layer, parameter가 포함되어있어 학습됨**
    - **nn.ReLU – ReLU activation의 역할을 하는 layer, 단순 연산이기에 parameter가 없음**

dataset

- **Dataset**
  - 각 데이터를 tensor로 변환하여 반환하는 클래스
  - __init__ : class의 선언자, 데이터 경로나 데이터 파일(dataframe)등을 선언하여 사용
  - __len__ : 전체 데이터셋 길이를 반환
  - __getitem__ : 각데이터와 정답을 텐서로 변환해서 반환

```
# dataset definition
class CSVDataset(Dataset):
    # load the dataset
    def __init__(self, path):
        # load the csv file as a dataframe
        df = read_csv(path, header=None)
        # store the inputs and outputs
        self.X = df.values[:, :-1]
        self.y = df.values[:, -1]
        # ensure input data is floats
        self.X = self.X.astype('float32')
        # label encode target and ensure the values are floats
        self.y = LabelEncoder().fit_transform(self.y)
        self.y = self.y.astype('float32')
        self.y = self.y.reshape((len(self.y), 1))

    # number of rows in the dataset
    def __len__(self):
        return len(self.X)

    # get a row at an index
    def __getitem__(self, idx):
        return [self.X[idx], self.y[idx]]
```

dataset

- **DataLoader**
  - DataLoader(dataset, batch_size=batch_size)
  - 데이터를 한개씩 반환하는 데이터와 batch_size를 argument로 넣어주면 배치개씩 묶어서 반환

```
train_dl = DataLoader(train, batch_size=32, shuffle=True)
test_dl = DataLoader(test, batch_size=1024, shuffle=False)
```

Model

- **nn.Linear**
  - nn.Linear(in_feature, out_feature, bias=False)
  - Perceptron의 linear(linear regression)부분
  - Input data의 차원수를 in_feature에 입력하고
  - Output data의 차원수를 out_feature에 입력
  - 결과물의 차원수는 out_feature와 같음

```
self.hidden1 = Linear(n_inputs, 10)
```

- **nn.ReLU**
  - ReLU activation을 수행하는 함수

```
self.act1 = ReLU()
```

  - 13p의 예시와 동일한 형태로 만들기 위해서는 nn.Linaer(2,2,bias=True), nn.Sigmoid(), nn.Linear(2,3)), nn.Sigmoid()을 이어 붙여 구현

Model

- **nn.Module**
  - 각 layer를 선언하고 feed forward 순서를 정의하여 결과를 도출하는 클래스
  - __init__: 클래스의 선언자로 일반적으로 nn.linear같은 layer선언
  - forward(data): input tensor와 layer의 처리순서와 같이 feed forward 순서와 결과 반환
  - nn.Module.parameters()로 학습가능한 parameter들을 모두 불러올 수 있음

```
# model definition
class MLP(Module):
    # define model elements
    def __init__(self, n_inputs):
        super(MLP, self).__init__()
        # input to first hidden layer
        self.hidden1 = Linear(n_inputs, 10)
        kaiming_uniform_(self.hidden1.weight, nonlinearity='relu')
        self.act1 = ReLU()
        # second hidden layer
        self.hidden2 = Linear(10, 8)
        kaiming_uniform_(self.hidden2.weight, nonlinearity='relu')
        self.act2 = ReLU()
        # third hidden layer and output
        self.hidden3 = Linear(8, 1)
        xavier_uniform_(self.hidden3.weight)
        self.act3 = Sigmoid()

    # forward propagate input
    def forward(self, X):
        # input to first hidden layer
        X = self.hidden1(X)
        X = self.act1(X)
         # second hidden layer
        X = self.hidden2(X)
        X = self.act2(X)
        # third hidden layer and output
        X = self.hidden3(X)
        X = self.act3(X)
        return X
```

Loss function

- **nn.MSELoss**
  - Loss = MSELoss(pred, gold)
  - 예측값과 정답을 입력하여 loss를 계산하는 class
  - Loss계산후 loss.backward()를 통해 gradient를 연산할 수 있다.

```
criterion = MSELoss()
```

```
# compute the model output
yhat = model(inputs)
# calculate loss
loss = criterion(yhat, targets)
# credit assignment
loss.backward()
```

optimizer

- **Optim.Adam**
  - 옵티마이저중 Adam optimizer를 일반적으로 사용
  - Optimizer = Adam(model.parameters(), lr=learning rate)
  - 학습할 파라미터 목록과 learning rate를 입력하여 선언
  - Lossfunction으로 gradient연산후 optimizer.step()을 이용해 parameter update

```
optimizer = SGD(model.parameters(), lr=0.01, momentum=0.9)
```

```
optimizer.step()
```

Pytorch 순서

1. Dataset을 통해 dataset load및 dataloader를 통해 batch단위로 불러옴
2. Nn.Module을 통해 모델 선언후 feedforward과정 정립
3. Model을 통해 결과 반환
4. Nn.MSEloss를 통해 loss연산
5. Loss.backward를 통해 gradient 연산
6. Optimizer.step을 통해 parameter 업데이트

Appendix

딥러닝은 블랙박스 모델을 통해 결과를 예측
그렇기에 모델의 내부 구조가 어떻든 원하는 형태의 결과가 나오도록 유도하면 학습이 가능
입력된 정답의 형태(차원수, shape)가 model output의 shape와 동일해야함

예를들어 1개의 값을 예측하는 regression task의 경우에는 각 데이터가 입력되고 나오는 출력이 1차원의 값이 나오도록 유도

또한 예측해야 하는 값이 n개인 regression task의 경우에는 결과가 n차원의 값이 나와 각각의 value가 n개의 예측값인 방향으로 사용

Appendix

Classification task의 경우에는 정답을 변형하여 deep learning에 유리한 방향이 되도록 변형

기존의 true/false로 구분되는 binary classification의 경우 정답이면 1 아니면 0 으로 사용이 가능하다.

Model의 output 또한 1차원의 vector가 나오도록 하여 해당 값을 true일 확률 처럼 사용

Appendix

Classification task의 경우에는 정답을 변형하여 deep learning에 유리한 방향이 되도록 변형

기존의 true/false로 구분되는 binary classification의 경우 정답이면 1 아니면 0 으로 사용이 가능하다.

Model의 output 또한 1차원의 vector가 나오도록 하여 해당 값을 true일 확률 처럼 사용

하지만 label이 여러 개인 multi class classification의 경우 모든 클래스에 대한 확률을 1차원의 텐서에 나타낼 수 없음

자세한 내용은 이후의 2020-2 DL강의자료 참조

Multi-class classification

- How can we define model?
  - Example.
  - Classification of Felidae

Multi-class classification

- How can we define model?
  - Example.
  - Classification of Felidae
  - Try to simply classify by their height, weight, caloric intake a day.

X			Y
Height	Weight	calory	
80	220	6300	Cheetah
75	167	4500	jaguar
86	210	7500	leopard
110	330	9000	tiger
95	280	8700	tiger
67	190	6800	jaguar

Multi-class classification

- Hypothesis
  - Convert data and weight into matrix form

X			Y
Height	Weight	calory	
80	220	6300	Cheetah
75	167	4500	jaguar
86	210	7500	leopard
110	330	9000	tiger
95	280	8700	tiger
67	190	6800	jaguar

$$\begin{bmatrix} 80 & 220 & 6300 \\ 75 & 167 & 4500 \\ 86 & 210 & 7500 \\ 110 & 330 & 9000 \\ 95 & 280 & 8700 \\ 67 & 190 & 6800 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} \text{cheetah} \\ \text{jaguar} \\ \text{leopard} \\ \text{tiger} \\ \text{tiger} \\ \text{jaguar} \end{bmatrix}$$

  - Transform text to the number for formal equation
    (0: tiger, 1: leopard, 2:cheetah, 3:jaguar)

$$\begin{bmatrix} 80 & 220 & 6300 \\ 75 & 167 & 4500 \\ 86 & 210 & 7500 \\ 110 & 330 & 9000 \\ 95 & 280 & 8700 \\ 67 & 190 & 6800 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 1 \\ 0 \\ 0 \\ 3 \end{bmatrix}$$

Complete?

Multi-class classification

- Hypothesis
  - You should never make something like this.

$$\begin{bmatrix} 80 & 220 & 6300 \\ 75 & 167 & 4500 \\ 86 & 210 & 7500 \\ 110 & 330 & 9000 \\ 95 & 280 & 8700 \\ 67 & 190 & 6800 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 1 \\ 0 \\ 0 \\ 3 \end{bmatrix}$$

0: tiger
1: leopard
2:cheetah
3:jaguar

  - This model is for linear regression, not classification.
    - Is a cheetah more like a leopard than a tiger?
    - If prediction is 2.5, then it means the half of chhetah and jaguar?

  - This is one of the most common mistakes.

    → Use binary classification for each class.

Multi-class classification

- New (and correct) hypothesis

X			Y			
Height	Weight	calory	Tiger	Leopard	Cheetah	jaguar
80	220	6300	0	0	1	0
75	167	4500	0	0	0	1
86	210	7500	0	1	0	0
110	330	9000	1	0	0	0
95	280	8700	1	0	0	0
67	190	6800	0	0	0	1

$$\begin{bmatrix} 80 & 220 & 6300 \\ 75 & 167 & 4500 \\ 86 & 210 & 7500 \\ 110 & 330 & 9000 \\ 95 & 280 & 8700 \\ 67 & 190 & 6800 \end{bmatrix} \begin{bmatrix} w_{11} w_{12} w_{13} w_{14} \\ w_{21} w_{22} w_{23} w_{24} \\ w_{31} w_{32} w_{33} w_{34} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Multi-class classification

- New (and correct) hypothesis

X			Y			
Height	Weight	calory	Tiger	Leopard	Cheetah	jaguar
80	220	6300	0	0	1	0
75	167	4500	0	0	0	1
86	210	7500	0	1	0	0
110	330	9000	1	0	0	0
95	280	8700	1	0	0	0
67	190	6800	0	0	0	1

$$\begin{bmatrix} 80 & 220 & 6300 \\ 75 & 167 & 4500 \\ 86 & 210 & 7500 \\ 110 & 330 & 9000 \\ 95 & 280 & 8700 \\ 67 & 190 & 6800 \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

classifier