Deep Neural Network

Jeonghun Yoon

Terms

신경망neural network

퍼셉트론perceptron

활성 함수

ReLU

Feed Forward Network

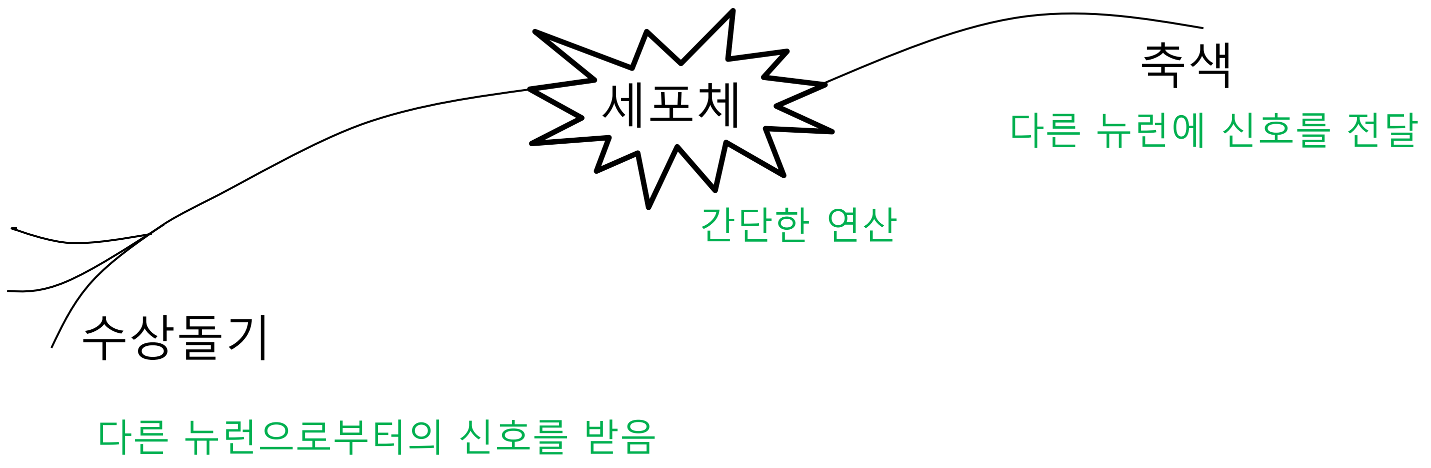
Back propagation algorithm

신경망Neural Network

기존의 폰 노이만 컴퓨터 구조를 뛰어 넘기 위해 뇌의 정보 처리를 모방하는 새로운 계산 모형을 개발하자는 목적(병렬 명령 처리)

연결주의(connectionism) : 단순한 연산을 수행하는 아주 많은 연산기와 그들 간의 아주 많은 연결을 통해 지능적인 일을 달성하려는 의도의 계산 모형

인공 신경망(Artificial Neural Network)



신경망Neural Network

일반화 능력이 뛰어나고, 구현이 쉬움.

신경망은 분류, 예측, 평가, 합성, 제어 등 다양한 분야에 적용할 수 있음.

Perceptron

- 신경망을 분류에 이용한 선형 분류기
- Rosenblatt가 제안
- 선형 분리 가능한 경우에 대해서만 완벽한 분류가 가능

학습 알고리즘의 설계

STEP1) 분류기 구조 정의(수학적 표현) : $\boldsymbol{\Theta}$

STEP2) 분류기 품질 측정을 위한 비용 함수(cost function) 정의 : $\boldsymbol{J(\Theta)}$

STEP3) $J(\Theta)$를 최대 또는 최소로 하는 Θ를 찾기 위한 알고리즘의 설계

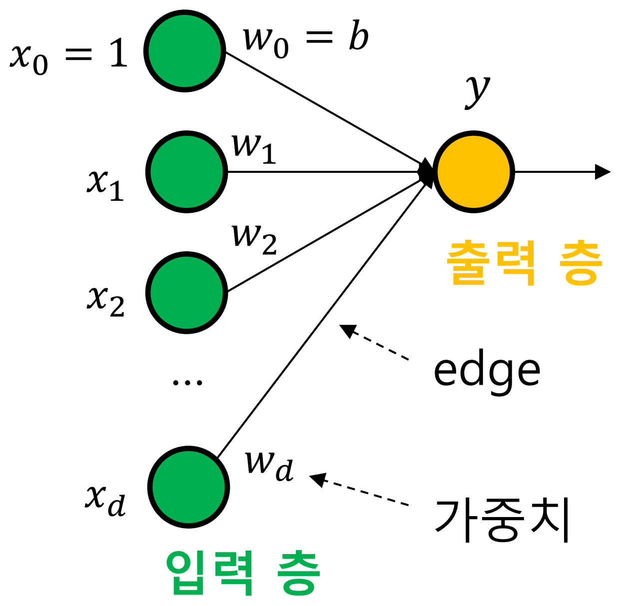
단층 퍼셉트론Single Layer Perceptron

Perceptron

- 신경망을 사용한 선형 분류기
- 선형 분리가 가능한 경우에 대해서만 완벽한 분류가 가능

Perceptron의 구조

- 입력 데이터 : 특징 벡터 $\mathbb{x} = (x_1, \ldots, x_d)$
- 입력 층 (input layer) : $d + 1$개의 노드
    - d : 특징 벡터의 차원
    - 1 : bias(항상 1의 값을 갖는다.)
- 출력 층 (output layer) : 1개의 노드
- edge : 입력 층과 출력 층을 연결
    - 가중치를 가진다. (연결 강도, connection strength)



단층 퍼셉트론 Single Layer Perceptron

Perceptron에서의 연산

- 입력 층에서의 연산 : 값을 전달해 주는 연산
- 출력 층에서의 연산
  - 합 연산
  - 활성 함수 계산

<u>출력 층에 전달되는 값</u>

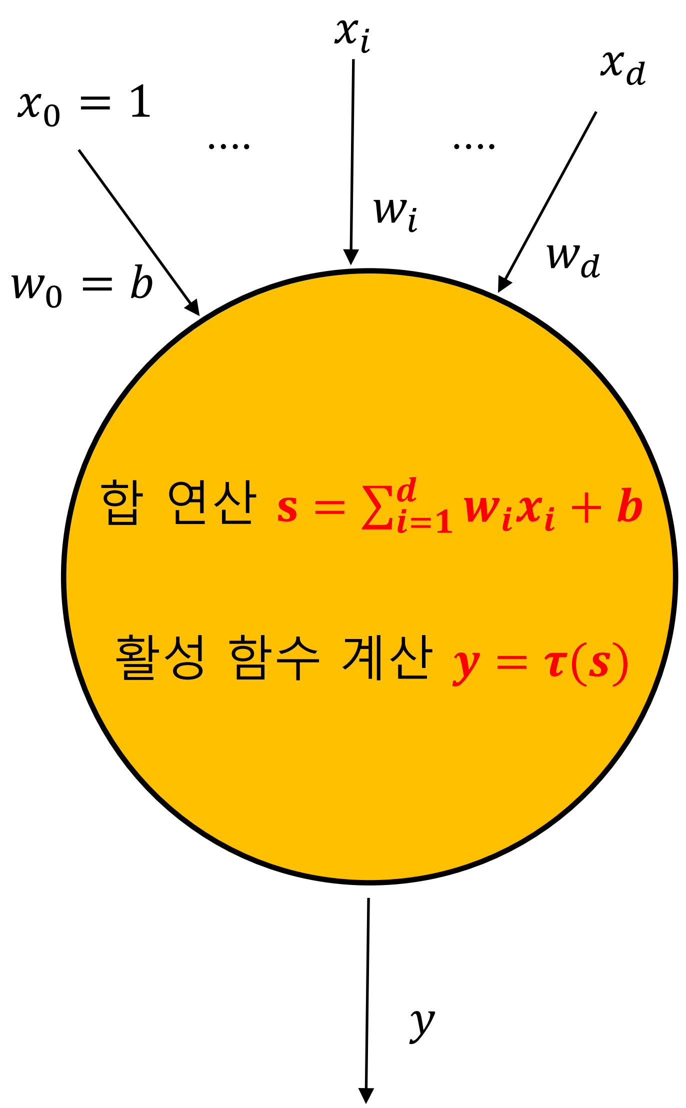
- 특징 벡터 $\mathbb{x} = (x_1, \ldots, x_d)$

- 가중치 벡터 $\mathbb{w} = (w_1, \ldots, w_d)$

<u>활성 함수 $\tau(.)$: step function</u>

- $\tau(s) = +1$ if $s \geq 0$

- $\tau(s) = -1$ if $s < 0$



단층 퍼셉트론Single Layer Perceptron

- 연산을 정리해보자. Perceptron에 사용되는 연산은 다음과 같다.

  가중치 벡터 : $\mathbb{w}^T = (w_1, \dots, w_d)$

  특성 벡터 : $\mathbb{x}^T = (x_1, \dots, x_d)$

  연산 :

$$y = \tau(s) = \tau\left(\sum_{i=1}^{d} w_i x_i + b\right) = \tau(\mathbb{w}^T \mathbb{x} + b)$$

where $\tau(s) = \begin{cases} +1, s \geq 0 \\ -1, s < 0 \end{cases}$

- 위에서 정의된 Perceptron은 선형 분류기(linear classifier)에 해당한다. 선형 분류기는 샘플을 2개의 부류 C_1, C_2 로 분류한다.

  선형 분류기가 사용하는 결정 직선 : $d(.)$

  입력된 샘플에 대한 선형 분류기의 결정 조건 :

$$d(\mathbb{x}) = \mathbb{w}^T \mathbb{x} + b > 0 \text{ 이면 } \mathbb{x} \in C_1$$

$$d(\mathbb{x}) = \mathbb{w}^T \mathbb{x} + b < 0 \text{ 이면 } \mathbb{x} \in C_2$$

Perceptron 학습

훈련 집합이 $X = \{(\mathbb{x}_1, t_1), (\mathbb{x}_2, t_2), \ldots, (\mathbb{x}_N, t_N)\}$ 주어졌을 때, 이들을 모두 옳게 분류하는 퍼셉트론 $\Theta = (\mathbb{w}, b)$ 을 찾아라. 단 샘플은 선형 분리 가능하다.

- $\mathbb{x}$: 특징 벡터
- t : 부류 표지
- $\mathbb{x}_i \in C_1 \Rightarrow \mathbb{t}_i = 1$ or -1

How to find?

- 퍼셉트론 $\Theta = (\mathbb{w}, b)$의 품질을 측정할 수 있는 비용 함수 $J(\Theta)$ 정의
- 비용 함수를 이용하여 품질을 개선시키는 방향으로 학습

비용 함수 $J(\Theta)$

- Y : Θ가 틀리게 분류하는 샘플의 집합
- $J(\Theta) = E = \sum_{\mathbb{x}_k \in Y} (-t_k)(\mathbb{w}^T \mathbb{x}_k + b)$

비용 함수 (Cost function)

비용 함수 $J(\Theta) = \sum_{\mathbb{x}_k \in Y}(-t_k)(\mathbb{w}^T \mathbb{x}_k + b)$ 선택의 당위성

- 샘플 $\mathbb{x}_i$가 오분류 되었다고 가정하자.
- Case 1) $t_i = 1$
  - $\mathbb{x}_i$가 오분류 되었으므로, $\mathbb{w}^T \mathbb{x}_i + b$의 값은 0보다 작아야 한다.
  - 따라서 $t_k(\mathbb{w}^T \mathbb{x}_k + b) < 0$ 이다. 이를 통하여 $-t_k(\mathbb{w}^T \mathbb{x}_k + b) > 0$임을 알 수 있다.
- Case 2) $t_i = -1$
  - $\mathbb{x}_i$가 오분류 되었으므로, $\mathbb{w}^T \mathbb{x}_i + b$의 값은 0보다 커야 한다.
  - 따라서 $t_k(\mathbb{w}^T \mathbb{x}_k + b) < 0$ 이다. 이를 통하여 $-t_k(\mathbb{w}^T \mathbb{x}_k + b) > 0$임을 알 수 있다.
- 즉, 모든 경우에서 $-t_k(\mathbb{w}^T \mathbb{x}_k + b) > 0$이므로,
  - <u>오분류가 증가</u>할수록 <u>비용 함수의 값은 증가</u>하고,
  - <u>오분류가 감소</u>할수록 <u>비용 함수의 값은 감소</u>한다.

비용 함수 (Cost function)

비용 함수의 최소값을 찾는 문제 = 잘못 분류된 샘플들의 총 개수를 최소화(0으로)

- Gradient descent method

Gradient method for $\mathbb{w}, b$

- $\mathbb{w}, b$의 초기값에서 시작
- 비용함수($J(\Theta)$) 의 값이 감소하는 방향으로$\mathbb{w}, b$를 이동
  - $J(\Theta)$를 $\mathbb{w}, b$로 각각 편미분하여, 기울기의 반대방향으로 $\mathbb{w}, b$의 값을 이동

시간 $t+1$에서 $\mathbb{w}, b$의 값(ρ : 학습률)

- $\mathbb{w}(t+1) = \mathbb{w}(t) - \rho \frac{\partial E}{\partial \mathbb{w}} = \mathbb{w}(t) - \rho \sum_{\mathbb{x}_k \in Y} (-\boldsymbol{t_k}) \mathbb{x}_{\boldsymbol{k}}$
- $b(t+1) = b(t) - \rho \frac{\partial E}{\partial b} = b(t) - \rho \sum_{\mathbb{x}_k \in Y} (-\boldsymbol{t_k})$

Perceptron 학습 알고리즘 (배치 모드, batch mode)

Sample : 선형 분리 가능 - 오분류하는 샘플 수를 0

입력 : $X = \{(\mathbb{x}_1, \mathbb{t}_1), (\mathbb{x}_2, \mathbb{t}_2), \ldots, (\mathbb{x}_N, \mathbb{t}_N)\}$, 학습률 ρ
출력 : Perceptron $\Theta = (\mathbb{w}, b)$

알고리즘 :

$\mathbb{w}, b$를 초기화
repeat {
  $Y = \emptyset$
  for($i = 1$ to N) {
    $y = \tau(\mathbb{w}^T \mathbb{x}_i + b)$;
    if $(y \neq t_i)$ $Y = Y \cup \{\mathbb{x}_i\}$;
  }
  $\mathbb{w} = \mathbb{w} + \rho \sum_{\mathbb{x}_k \in Y} t_k \mathbb{x}_k$;
  $b = b + \rho \sum_{\mathbb{x}_k \in Y} t_k$;
} until ($Y = \emptyset$)
$\mathbb{w}, b$를 출력

Perceptron 학습 알고리즘 (온라인 모드, online mode)

Sample : 선형 분리 가능 - 오분류하는 샘플 수를 0

입력 : $X = \{(\mathbb{x}_1, \mathbb{t}_1), (\mathbb{x}_2, \mathbb{t}_2), \dots, (\mathbb{x}_N, \mathbb{t}_N)\}$, 학습률 ρ
출력 : Perceptron $\Theta = (\mathbb{w}, b)$

알고리즘 :

```
        w, b를 초기화
        repeat {
                QUIT = true;
                for(i = 1 to N) {
                        y = τ(w^T x_i + b);
                        if (y ≠ t_i) {
                                QUIT = false;
                                w = w + ρ t_i x_i;
                                b = b + ρ t_i;        J(Θ) = −t(w^T x + b)
                        }
                }
        } until (QUIT = true)
        w, b를 출력
```

$J(\Theta) = -t(\mathbb{w}^T\mathbb{x} + b)$

Perceptron 학습 알고리즘

Batch mode vs Pattern mode

- Batch mode : 모든 샘플에 대하여 w, b의 이동량을 계산한 후 w, b를 한 번 업데이트
- Online mode : 샘플 각각에 대하여 w, b의 이동량을 계산한 후 w, b를 한 번 업데이트
- Batch mode는 잡음에 둔감하게 해주는 효과
- Pattern mode가 더 좋은 성능을 보임

Perceptron 학습 알고리즘 (포켓 알고리즘, pocket)

Sample : 선형 분리 불가능 - 오분류하는 샘플 수를 최소화

입력 : $X = \{(\mathbb{x}_1, \mathbb{t}_1), (\mathbb{x}_2, \mathbb{t}_2), \dots, (\mathbb{x}_N, \mathbb{t}_N)\}$, 학습률 ρ
출력 : Perceptron $\Theta = (\mathbb{w}, b)$

알고리즘 :

$\mathbb{w}, b$를 초기화, $\mathbb{w}_{best} = \mathbb{w}$, $b_{best} = b$
품질을 0으로 초기화, $q_{best} = 0$
repeat {
for($i = 1$ to N) {
$y = \tau(\mathbb{w}^T \mathbb{x}_i + b)$;
if $(y \neq t_i)$ {
$\mathbb{w} = \mathbb{w} + \rho t_i \mathbb{x}_i$;
$b = b + \rho t_i$;
}
}
$\mathbb{w}, b$로 N개의 샘플을 인식하여 정인식률 q를 구한다.
if $(q > q_{best})$ $\{\mathbb{w}_{best} = \mathbb{w}; b_{best} = b; q_{best} = q\}$
} until (stop-condition)
$\mathbb{w} = \mathbb{w}_{best}, b = b_{best}$를 출력

패턴 모드 사용.
(배치 모드로 변경 가능)

$J(\Theta) = -t(\mathbb{w}^T \mathbb{x} + b)$

선형분리 불가능 상황

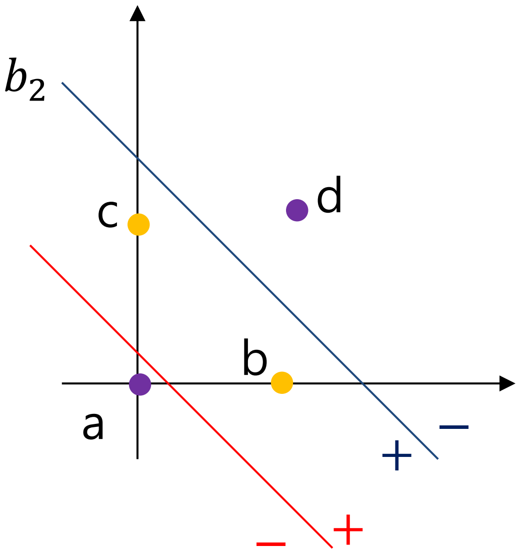
2차원 공간의 샘플을 생각해보자. $\mathbb{x} = (x_1, x_2) \in \mathbb{R}^2$

이 경우는 결정직선 1개로 샘플들을 오류없이 분류하기 불가능하다.

2개의 결정직선을 생각해보자.

$\mathbb{w}_2 = (w_{12}, w_{22}) \rightarrow d_2(\mathbb{x}) = w_{12}x_1 + w_{22}x_2 + b_2$

$\mathbb{w}_1 = (w_{11}, w_{21}) \rightarrow d_1(\mathbb{x}) = w_{11}x_1 + w_{21}x_2 + b_1$



이 경우, 다음과 같은 분류기준을 생성할 수 있다.

① $\mathbb{w}_1^T\mathbb{x} + b_1 > 0$이고, $\mathbb{w}_2^T\mathbb{x} + b_2 > 0$이면, $\mathbb{x} \in C_1$

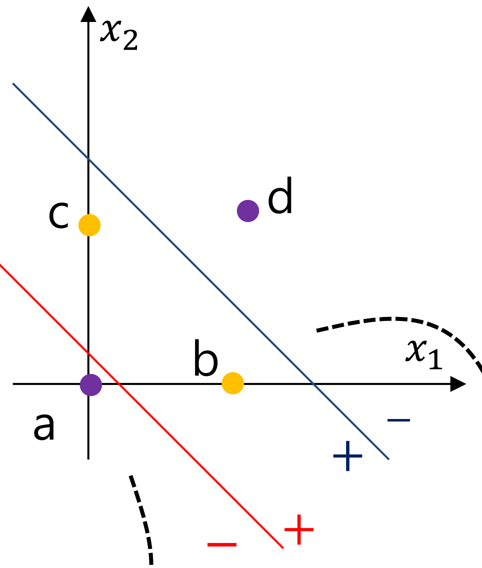
② $\mathbb{w}_1^T\mathbb{x} + b_1 < 0$이거나, $\mathbb{w}_2^T\mathbb{x} + b_2 < 0$이면, $\mathbb{x} \in C_2$

선형분리 불가능 상황

분류기준에 perceptron을 적용해보자. activation function은 step function으로 한다.

$$d_2(\mathbb{x}) = w_{12}x_1 + w_{22}x_2 + b_2$$

$$d_1(\mathbb{x}) = w_{11}x_1 + w_{21}x_2 + b_1$$

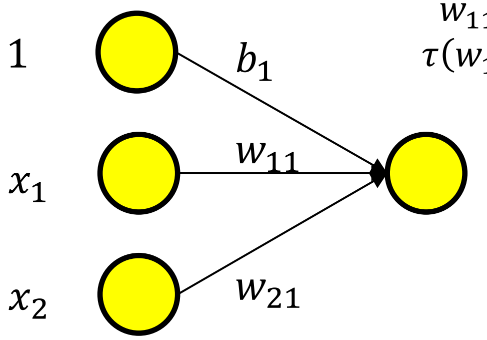


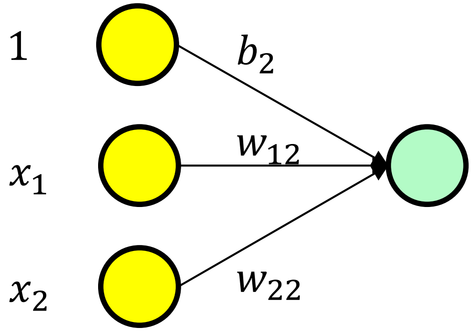
$$w_{12}x_1 + w_{22}x_2 + b_1 > 0 \rightarrow$$
$$\tau(w_{12}x_1 + w_{22}x_2 + b_1) = 1$$

$$w_{12}x_1 + w_{22}x_2 + b_1 < 0 \rightarrow$$
$$\tau(w_{12}x_1 + w_{22}x_2 + b_1) = -1$$

$$w_{11}x_1 + w_{21}x_2 + b_1 > 0 \rightarrow$$
$$\tau(w_{11}x_1 + w_{21}x_2 + b_1) = 1$$

$$w_{11}x_1 + w_{21}x_2 + b_1 < 0 \rightarrow$$
$$\tau(w_{11}x_1 + w_{21}x_2 + b_1) = -1$$



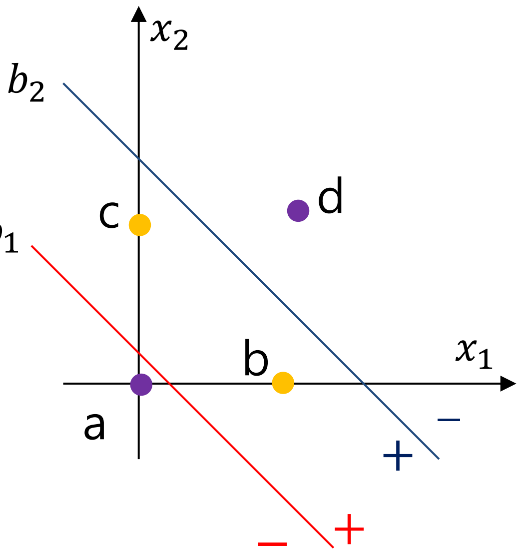


선형분리 불가능 상황

분류기준에 perceptron을 적용해보자. activation function은 step function으로 한다.

$d_2(\mathbb{x}) = w_{12}x_1 + w_{22}x_2 + b_2$

$d_1(\mathbb{x}) = w_{11}x_1 + w_{21}x_2 + b_1$



Perceptron 1

$w_{11}x_1 + w_{21}x_2 + b_1 > 0 \rightarrow$
$\tau(w_{11}x_1 + w_{21}x_2 + b_1) = 1$

$w_{11}x_1 + w_{21}x_2 + b_1 < 0 \rightarrow$
$\tau(w_{11}x_1 + w_{21}x_2 + b_1) = -1$

Perceptron 2

$w_{12}x_1 + w_{22}x_2 + b_1 > 0 \rightarrow$
$\tau(w_{12}x_1 + w_{22}x_2 + b_1) = 1$

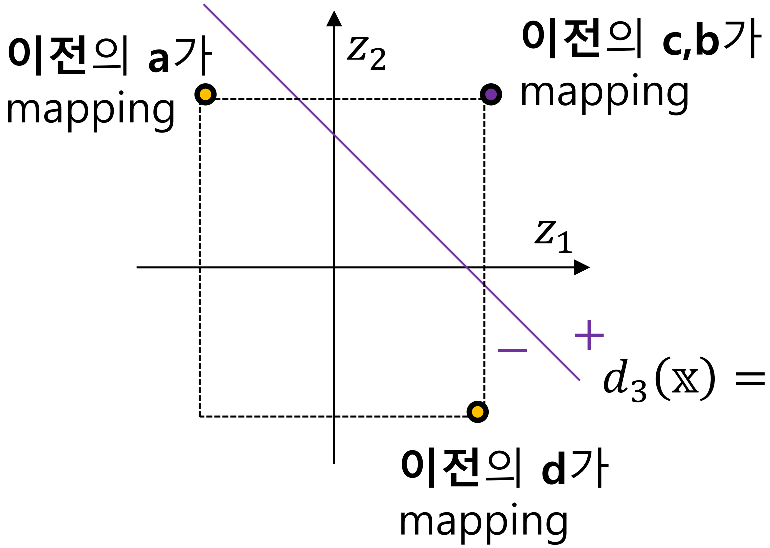
$w_{12}x_1 + w_{22}x_2 + b_1 < 0 \rightarrow$
$\tau(w_{12}x_1 + w_{22}x_2 + b_1) = -1$

perceptron에 의하여 각 영역에 속하는 샘플들은 다음에 해당하는 point로 각각 대응됨을 볼 수 있다.

- $w_{11}x_1 + w_{21}x_2 + b_1 > 0$ and $w_{12}x_1 + w_{22}x_2 + b_1 > 0$: $(1,1)$
- $w_{11}x_1 + w_{21}x_2 + b_1 < 0$ or $w_{12}x_1 + w_{22}x_2 + b_1 < 0$: $(1,-1),(-1,1)$

선형분리 불가능 상황

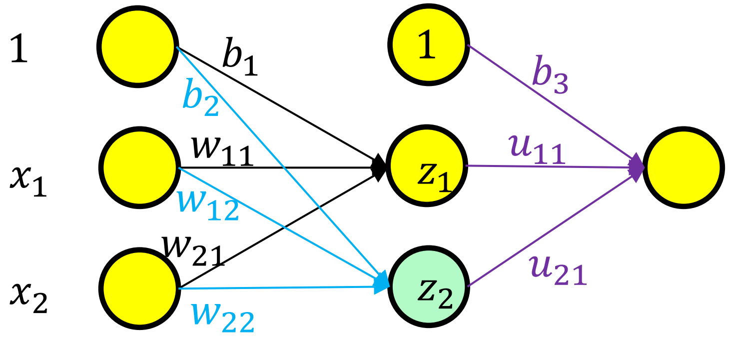
2개의 perceptron을 적용하게 되면 다음과 같이 된다.



$$d_3(\mathbb{x}) = u_{11}z_1 + u_{21}z_2 + b_3$$

새로 mapping된 포인트에 대하여
새로운 분류직선을 적용해보자.

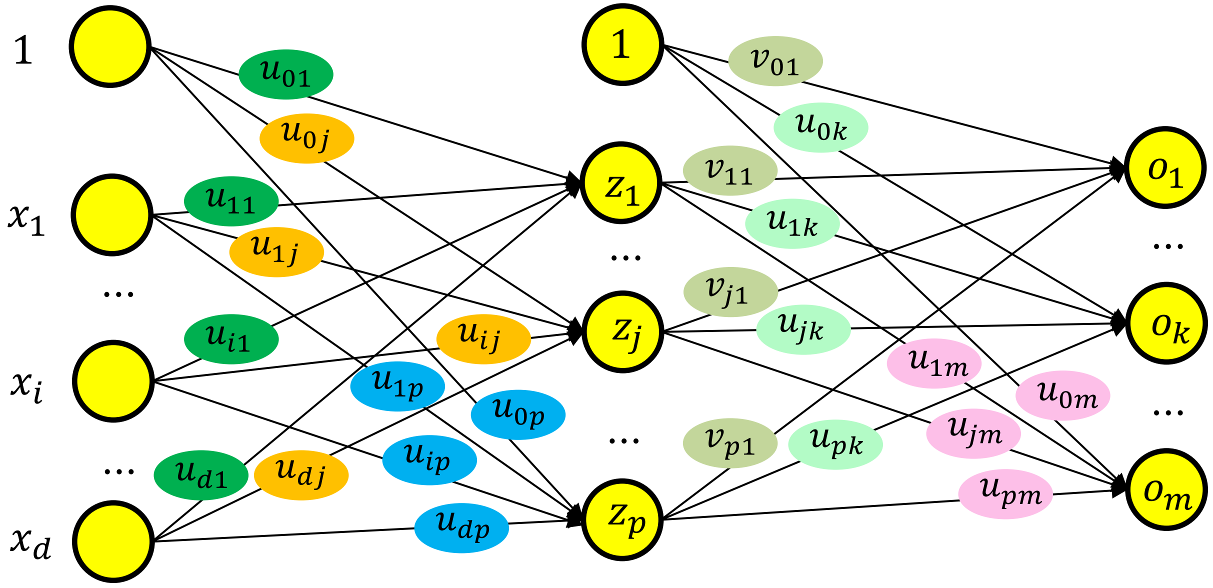
즉, 새로운 perceptron을 적용해보자.



새로운 분류직선으로 분류를 하게 되면,

우리가 원하는 결과를 얻게 된다.

Multiple Layer Perceptron (다층 퍼셉트론)



input layer $(d+1)$

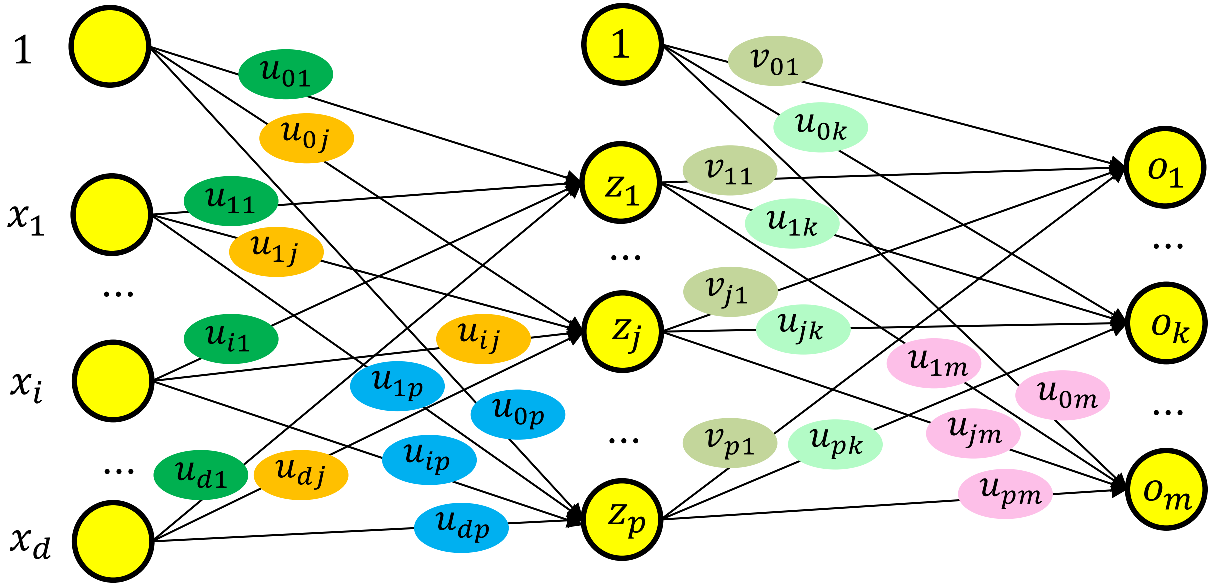
hidden layer $(p+1)$

outlayer layer (m)

input : 데이터 샘플 $\mathbb{x} = (x_1, \ldots, x_d)$

output : 부류 정보 $\mathbb{o} = (o_1, \ldots, o_m)$

Multiple Layer Perceptron (다층 퍼셉트론)



d차원 → p차원

$\mathbb{x} = (x_1, \ldots, x_d)$

$\mathbb{z} = (z_1, \ldots, z_p)$

$(u_{01}, u_{11}, \ldots, u_{d1})$
$(u_{02}, u_{12}, \ldots, u_{d2})$
$\ldots$
$(u_{0p}, u_{1p}, \ldots, u_{dp})$

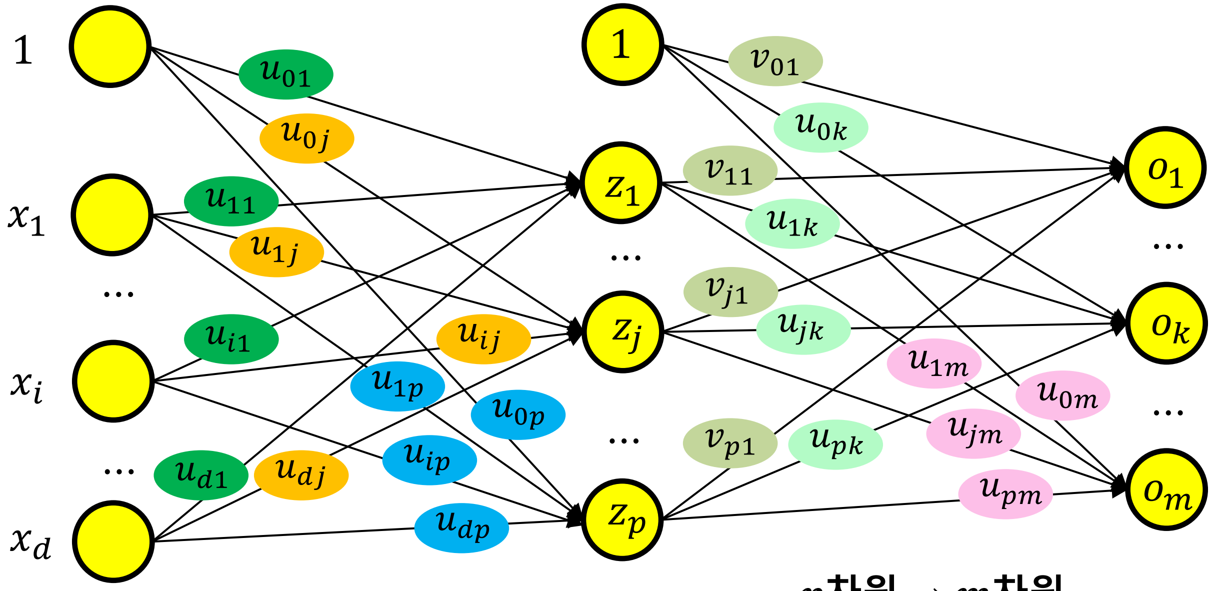
$sum_{z_1} = \sum_{i=1}^{d}(u_{i1}x_i + u_{01})$
$sum_{z_2} = \sum_{i=1}^{d}(u_{i2}x_i + u_{02})$
$\ldots$
$sum_{z_p} = \sum_{i=1}^{d}(u_{ip}x_i + u_{0p})$

$z_1 = \tau(sum_{z_1})$
$z_2 = \tau(sum_2)$
$\ldots$
$z_p = \tau(sum_{z_p})$

hidden layer의
activation 함수

p개의 결정직선 = p개의 perceptron

Multiple Layer Perceptron (다층 퍼셉트론)



p차원 → m차원

$\mathbb{Z} = (z_1, \dots, z_p)$

$\mathbb{O} = (o_1, \dots, o_m)$

$$(v_{01}, v_{11}, \dots, v_{p1})$$
$$(v_{02}, v_{12}, \dots, v_{p2})$$
$$\dots$$
$$(v_{0m}, v_{1m}, \dots, v_{pm})$$

$$sum_{o_1} = \sum_{i=1}^{p}(v_{i1}z_i + v_{01})$$
$$sum_{o_2} = \sum_{i=1}^{p}(v_{i2}z_i + v_{02})$$
$$\dots$$
$$sum_{o_m} = \sum_{i=1}^{p}(v_{im}z_i + v_{0m})$$

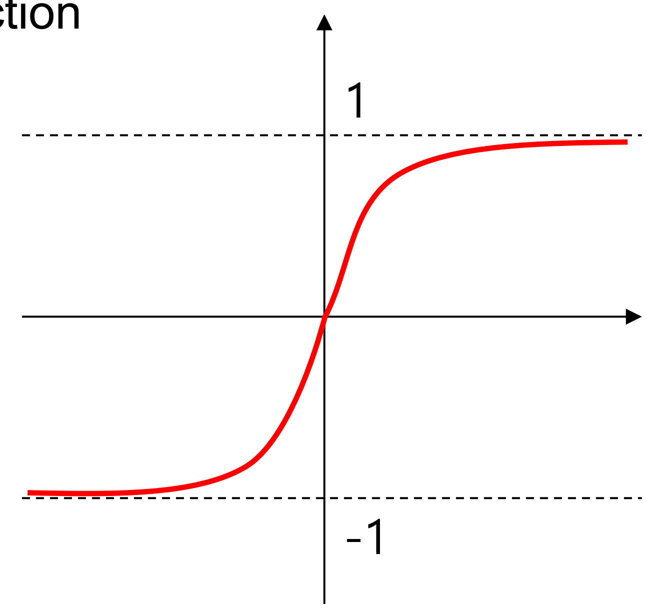
$$o_1 = \tau(sum_{o_1})$$
$$o_2 = \tau(sum_{o_2})$$
$$\dots$$
$$o_m = \tau(sum_{o_m})$$

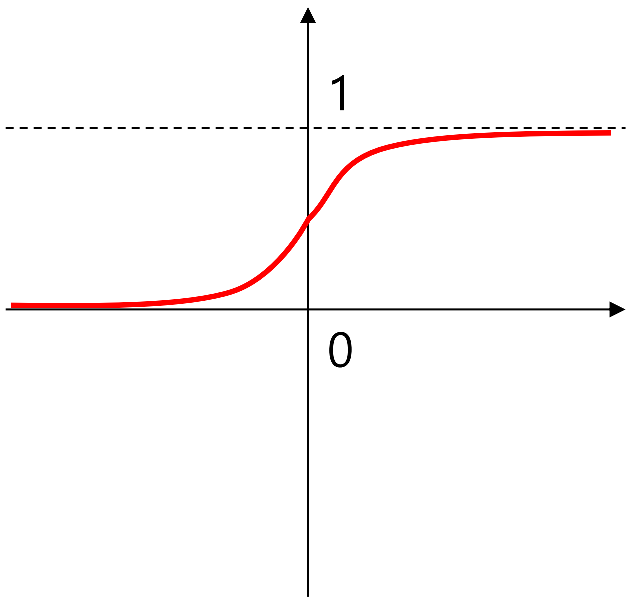
output layer의
activation 함수

m개의 결정직선 = m개의 perceptron

Hidden Layer 활성 함수 (Sigmoid)

Sigmoid Function





양극 시그모이드 함수(Hyper tangent)

$$\tau(x) = \frac{2}{1+e^{-\alpha x}} - 1$$

$$\tau'(x) = \frac{\alpha}{2}(1 + \tau(x))(1 - \tau(x))$$

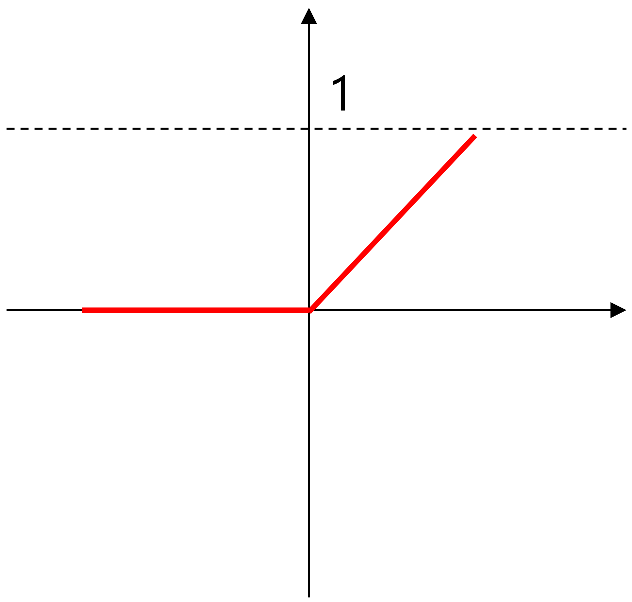
이진 시그모이드 함수

$$\tau(x) = \frac{1}{1+e^{-\alpha x}}$$

$$\tau'(x) = \alpha\tau(x)(1 - \tau(x))$$

Hidden Layer 활성 함수 (ReLU)

Recified Linear Unit (ReLU)



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

$\tau'(x) = 0(x \leq 0)$ or $1(x > 0)$

장점

1) stocastic gradient descent의 가속화가 빠르다. (sigmoid 종류보다)

2) sigmoid 종류의 exponential 연산보다 비용이 싸다. (계산의 복잡성, 시간 측면)

단점

1) training can "die".

즉, 어떤 data point에서도 다시는 activation 되지 않는 update가 발생할 확률이 높다. 40% 정도.

(gradient가 계속 zero value)

Output Layer 활성 함수

Linear Function

Sigmoid Function

SoftMax Function

$$o_i = \frac{\exp\left(sum_{o_i}\right)}{\sum_j \exp(sum_{o_j})} \qquad 0 \le o_i \le 1 \text{ , } \sum o_i = 1$$

구조 설계 시 고려 사항

몇 개의 층을 둘 것인가?

- 일반적으로, 은닉 층은 원하는 개수만큼 둘 수 있다.

층간의 연결은 어떻게 할 것인가?

- 일반적으로, 이웃하지 않은 층간에도 edge를 가질 수 있고, 오른쪽에서 왼쪽으로 가는 피드백 edge를 가질 수도 있다. 일부 노드 간에만 edge가 있는 부분 연결 구조를 가질 수도 있다.
- **FFMLP**(feed-forward MLP)
  - 앞으로만 전달되는 구조(전방 다층 퍼셉트론)
  - 가장 보편적으로 사용되는 아키텍쳐

구조 설계 시 고려 사항

각 층에 있는 노드는 몇 개로 할 것인가?

- 특징 벡터, 부류의 개수는 고정 ⇒ 입력 노드와 출력 노드의 개수는 고정
- 일반적으로, 최적의 은닉 노드의 수를 추정하는 방법은 없다.
- 보통 여러 값에 대해 신경망을 훈련하고 그 중 가장 좋은 성능을 보이는 것을 선택

어떤 활성 함수를 사용할 것인가?

- 일반적으로 노드마다 서로 다른 활성 함수를 사용할 수 있다.
- 보편적으로는 모든 노드가 같은 활성 함수를 사용한다.

MLP 학습

훈련 집합이 $X = \{(\mathbb{x}_1, \mathbb{t}_1), (\mathbb{x}_2, \mathbb{t}_2), \dots, (\mathbb{x}_N, \mathbb{t}_N)\}$ 주어졌을 때, 다중 퍼셉트론(MLP) $\boldsymbol{\Theta} = (\boldsymbol{U}, \boldsymbol{V})$를 찾아라.

- $\mathbb{x}$: 특징 벡터
- $\mathbb{t}$: 부류 표지 벡터(class label vector) or 목적 벡터(target vector)
  - K separate binary classification의 경우
    - $\mathbb{x}_i \in C_j \Rightarrow \mathbb{t}_i = (0, \dots, 1, \dots 0)^T$ or $(-1, \dots, 1, \dots, -1)^T$ where j-th position is 1

      활성 함수 : 이진 sigmoid　　　활성 함수 : 양극 sigmoid

How to find?

- 다중 퍼셉트론 $\Theta = (U, V)$의 품질을 측정할 수 있는 비용 함수 $J(\Theta)$ 정의
- 비용 함수를 이용하여 품질을 개선시키는 방향으로 학습

비용 함수 $J(\Theta)$

- $\mathbb{x}$의 label : $\mathbb{t} = (t_1, \dots, t_m)$
- $\mathbb{x}$의 perceptron 연산 결과값 : $\mathbb{o} = (o_1, \dots, o_m)$
- SSE(sum-of-squared errors)

$$J(\Theta) = E = \frac{1}{2}\sum_{k=1}^{m}(t_k - o_k)^2$$

에러
(regression or classification)

MLP 학습

- Binaryclass Cross-entropy (single target, $t = 0$ or 1)

$$J(\Theta) = -(t \log t + (1 - t) \log(1 - t))$$

- Multiclass Cross-entropy

$$J(\Theta) = -\sum_{k=1}^{K} t_k \log o_k$$

for classification

Activation 함수 / Cost 함수

Output Activation 함수와 Cost 함수는 우리가 해결해야 할 문제의 type에 따라 선택

Regression

- Output activation function : Linear function
- Cost function : SSE (Sum-of-squared error)

Classification

- Binary or K separate binary classification
  - Output activation function : Logistic sigmoid function
  - Cost function : Cross-entropy
- Multiclass classification
  - Output activation function : Softmax function
  - Cost function : Cross-entropy

비용 함수 (Cost function)

비용 함수의 최소값을 찾는 문제 = 에러의 최소값

- Gradient descent method

Gradient method for U, V

- U, V의 초기값에서 시작
- 비용함수($J(\Theta)$) 의 값이 감소하는 방향으로 U, V를 이동
  - $J(\Theta)$를 U, V로 각각 편미분하여, 기울기의 반대방향으로 U, V의 값을 이동

시간 $t+1$에서 U, V의 값(ρ : 학습률)

- $U(t+1) = U(t) - \rho \frac{\partial E}{\partial U}$
- $V(t+1) = V(t) - \rho \frac{\partial E}{\partial V}$

MLP 학습 알고리즘 (simple)

입력 : $X = \{(\mathbb{x}_1, \mathbb{t}_1), (\mathbb{x}_2, \mathbb{t}_2), \dots, (\mathbb{x}_N, \mathbb{t}_N)\}$, 학습률 ρ
출력 : MLP $\Theta = (U, V)$

알고리즘 :

```
U,V를 초기화
repeat {
    for(X의 샘플 각각에 대해) {
        for 1 ≤ j ≤ p : hidden node z_j의 값을 구한다.
        for 1 ≤ k ≤ m : output node o_k의 값을 구한다.
        ∂E/∂U, ∂E/∂V를 구한다. where E = J(Θ)
        U = U − ρ ∂E/∂U , V = V − ∂E/∂V
    }
} until (멈춤 조건)
U,V를 출력
```

U, V를 초기화

repeat {

for(X의 샘플 각각에 대해) {

for $1 \le j \le p$: hidden node z_j의 값을 구한다.

for $1 \le k \le m$: output node o_k의 값을 구한다.

$\frac{\partial E}{\partial U}, \frac{\partial E}{\partial V}$를 구한다. where $E = J(\Theta)$

$U = U - \rho \frac{\partial E}{\partial U}$, $V = V - \frac{\partial E}{\partial V}$

}

} until (멈춤 조건)

U, V를 출력

전방 계산
(Forward computation)

오류 역전파
(Back propagation)

Gradient descent

How to compute $\frac{\partial E}{\partial U}, \frac{\partial E}{\partial \boldsymbol{V}}$?

$$U$$

$$(u_{01}, u_{11}, \dots, u_{d1})$$
$$(u_{02}, u_{12}, \dots, u_{d2})$$
$$\dots$$
$$(u_{0p}, u_{1p}, \dots, u_{dp})$$

$$V$$

$$(v_{01}, v_{11}, \dots, v_{p1})$$
$$(v_{02}, v_{12}, \dots, v_{p2})$$
$$\dots$$
$$(v_{0m}, v_{1m}, \dots, v_{pm})$$

$$\frac{\partial E}{\partial U}$$

$$\begin{pmatrix} \frac{\partial E}{\partial u_{01}} & \cdots & \frac{\partial E}{\partial u_{d1}} \\ \vdots & \ddots & \vdots \\ \frac{\partial E}{\partial u_{0p}} & \cdots & \frac{\partial E}{\partial u_{dp}} \end{pmatrix}$$

$$\frac{\partial E}{\partial V}$$

$$\begin{pmatrix} \frac{\partial E}{\partial v_{01}} & \cdots & \frac{\partial E}{\partial v_{p1}} \\ \vdots & \ddots & \vdots \\ \frac{\partial E}{\partial v_{0m}} & \cdots & \frac{\partial E}{\partial v_{pm}} \end{pmatrix}$$

How to compute $\frac{\partial E}{\partial u_{ij}}, \frac{\partial E}{\partial v_{jk}}$?

MLP 학습 알고리즘

입력 : $X = \{(\mathbb{x}_1, \mathbb{t}_1), (\mathbb{x}_2, \mathbb{t}_2), \dots, (\mathbb{x}_N, \mathbb{t}_N)\}$, 학습률 ρ
출력 : MLP $\Theta = (U, V)$

알고리즘 :

U, V를 초기화
repeat {
for(X의 샘플 각각에 대해) {
현재 샘플 : $\mathbb{x} = (x_1, \dots, x_d)$, $\mathbb{t} = (t_1, \dots, t_m)$
for $1 \le j \le p$: $z_j = \tau(sum_{z_j})$, $sum_{z_j} = \sum_{i=1}^{d} u_{ij} x_i + u_{0j}$
for $1 \le k \le m$: $o_k = \tau(sum_{o_k})$, $sum_{o_k} = \sum_{j=1}^{p} w_{jk} z_j + w_{0k}$
for $1 \le k \le m$: $\delta_k = (t_k - o_k)\tau'(sum_{o_k})$
for $\forall v_{jk}, 0 \le j \le p, 1 \le k \le m$: $\Delta v_{jk} = \rho \delta_k z_j$
for $1 \le j \le p$: $\eta_j = \tau'(sum_{z_j}) \sum_{i=1}^{d} \delta_k v_{jk}$
for $\forall u_{ij}, 0 \le i \le d, 1 \le j \le p$: $\Delta u_{ij} = \rho \eta_j x_i$
for $\forall v_{jk}, 0 \le j \le p, 1 \le k \le m$: $v_{jk} = v_{jk} + \Delta v_{jk}$
for $\forall u_{ij}, 0 \le i \le d, 1 \le j \le p$: $u_{ij} = u_{ij} + \Delta u_{ij}$
}
} until (멈춤 조건); U, V를 출력;

전방 계산
(Forward computation)

오류 역전파
(Back propagation)

가중치 업데이트

알고리즘 설계 시 고려 사항

가중치 초기화

- 학습이 어떤 초기값을 가지고 출발하느냐는 두 가지 측면에 영향을 미친다.
  - 전역 최적 점으로 수렴 vs 지역 최적 점으로 수렴
  - 수렴 속도
- 보편적으로, $-0.5 \sim 0.5$의 난수로 설정
- 일반적인 경우에 대해서, 전역 최적 점을 보장하며 가장 빠르게 수렴하는 초기값을 찾는 방법은 알려져 있지 않다.

알고리즘 종료 조건

- 여러 번 실험을 통하여 적절한 반복 회수를 찾아낸다.
- 한 번의 반복이 종료된 후, $MSE = \frac{1}{N}\sum_{i=1}^{N} E_i$의 값이 미리 설정된 임계값보다 작으면 멈춘다.
  - $E_i = \frac{1}{2}\sum_{k=1}^{m}(t_k - o_k)^2$ for $\mathbb{x}_i$

지역 최저 점 탈출

- 다중 시작(multi-start) : 여러 초기값으로 신경망을 훈련하고 그들 중에 가장 좋은 성능을 보이는 것을 선택

MLP 인식

입력 : MLP $\Theta = (U, V)$, 미지 패턴 $\mathbb{x}$
출력 : 부류 C_q

알고리즘 :

U, V를 이용하여 MLP를 설정한다.
for ($j = 1$ to p) {
$sum_{z_j} = \sum_{i=1}^{d} u_{ij} x_i + u_{0j}$;
$z_j = \tau(sum_{z_j})$;
}
for ($k = 1$ to m) {
$sum_{o_k} = \sum_{j=1}^{p} v_{jk} z_j + v_{0k}$;
$o_k = \tau(sum_{o_k})$;
}
$\mathbb{x}$를 $q = \arg\max_{j} o_j$ 인 w_q로 분류한다.

Appendix

Which cost function do we use?

How to compute $\frac{\partial E}{\partial u_{ij}}, \frac{\partial E}{\partial v_{jk}}$?

Which cost function do we use?

Why **SSE**?

We consider **regression**.

우리가 구하고자 하는 target variable를 t 라고 하자.

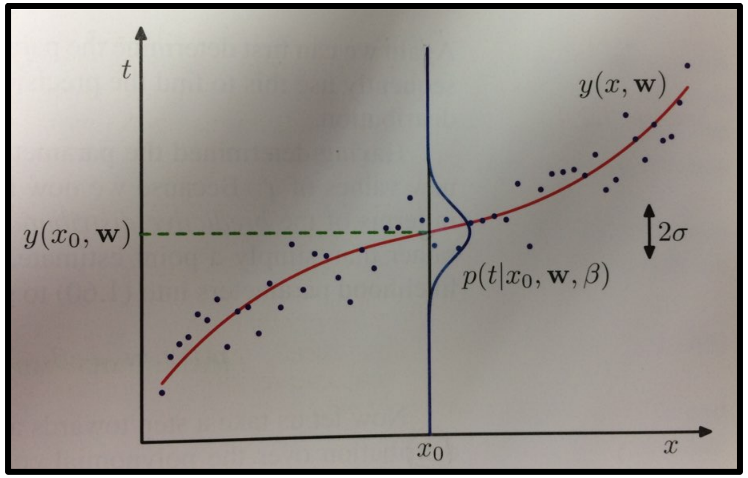
t 가 Gaussian distribution를 따른다고 가정하면,

neural network output
(회귀선)

$$\boldsymbol{P(t|\mathbb{x}, \mathbb{w})} = \mathcal{N}(t|y(\mathbb{x}, \mathbb{w}), \beta^{-1})$$

즉, 여기서 β는 Gaussian noise의 precision이다. (분산의 역수 값)

From Bishop's book (page 29)



Which cost function do we use?

N independent, identically distributed obsevation $X = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$ and $T = (t_1, \ldots, t_N)$ 이 주어졌다고 하자.

이에 대응하는 likelihood function은 (회귀에서 우리가 maximize하고 싶은)

$$\boldsymbol{P(T|X, \mathbf{w}, \beta)} = \prod_{n=1}^{N} P(t_n|\mathbf{x}_n, \mathbf{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n|y(\mathbf{x}_n, \mathbf{w}), \beta^{-1})$$

이에 대응하는 negative log likelihood는,

$$\frac{\beta}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 - \frac{N}{2}\ln\beta + \frac{N}{2}\ln(2\pi)$$

따라서,

$$\frac{1}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2$$

를 minimize하는 것과 동일하다.

SSE

Which cost function do we use?

Why **cross-entropy**?

We consider **binary classification**.

우리는 activation 함수로, Logistic sigmoid function을 사용할 것이다.

Binary classification 인 경우, neural network의 output layer는 single target이다.

$t = 1$이면 부류 C_1에 속할 것이고, $t = 0$이면 부류 C_2에 속할 것이다.

output layer의 single target 값을 $y(\mathbb{x}, \mathbb{w})$라고 하자. $0 \leq y(\mathbb{x}, \mathbb{w}) \leq 1$ 이다.

$y(\mathbb{x}, \mathbb{w})$ 값을 $P(C_1|\mathbb{x})$, $1 - y(\mathbb{x}, \mathbb{w})$ 값을 $P(C_2|\mathbb{x})$ 라고 할 수 있다.

따라서, likelihood 값, $\boldsymbol{P(t|\mathbb{x}, \mathbb{w})}$은 Bernoulli distribution을 따른다.

Which cost function do we use?

$$\boldsymbol{P}(\boldsymbol{t}|\mathbb{x},\mathbb{w}) = P(C_1|\mathbb{x})^t P(C_2|\mathbb{x})^{1-t} = y(\mathbb{x},\mathbb{w})^t\big(1 - y(\mathbb{x},\mathbb{w})\big)^{1-t}$$

likehood function의 값을 극대화하는 문제는 MLE 문제이다.

MLE문제는 negative log likehood를 minimize 하는 문제로 바꿀 수 있다.

즉, $-\big(t\ln y(\mathbb{x},\mathbb{w}) + (1-t)\ln\big(1 - y(\mathbb{x},\mathbb{w})\big)\big)$의 극소화 문제이다.

이것은 cross-entropy의 형태와 동일하다.

How to compute $\frac{\partial E}{\partial u_{ij}}, \frac{\partial E}{\partial v_{jk}}$?

$$\frac{\partial E}{\partial v_{jk}}$$

$$= \frac{\partial(\frac{1}{2}\sum_{r=1}^{m}(t_r - o_r)^2)}{\partial v_{jk}}$$

$$= \frac{\partial(\frac{1}{2}(t_k - o_k)^2)}{\partial v_{jk}}$$

$$= -(t_k - o_k)\frac{\partial o_k}{\partial v_{jk}}$$

$$= -(t_k - o_k)\frac{\partial \tau(sum_{o_k})}{\partial v_{jk}}$$

$$= -(t_k - o_k)\tau'(sum_{o_k})\frac{\partial sum_{o_k}}{\partial v_{jk}}$$

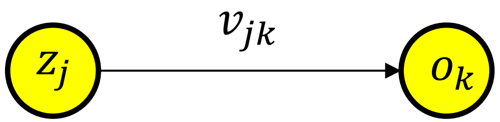
$$= \underbrace{-(t_k - o_k)\tau'(sum_{o_k})}_{\delta_k} z_j$$

$$= -\delta_k z_j$$

① $sum_{o_k} = \sum_{j=1}^{p} v_{jk} z_j + v_{0k}$

② $\tau(sum_{o_k}) = o_k$

③ v_{jk}가 미치는 영향



How to compute $\frac{\partial E}{\partial u_{ij}}, \frac{\partial E}{\partial v_{jk}}$?

$$\frac{\partial E}{\partial u_{jk}}$$

$$= \frac{\partial(\frac{1}{2}\sum_{r=1}^{m}(t_r - o_r)^2)}{\partial u_{jk}}$$

$$= -\sum_{k=1}^{m}(t_k - o_k)\frac{\partial o_k}{\partial u_{ij}} = -\sum_{k=1}^{m}(t_k - o_k)\frac{\partial \tau(sum_{o_k})}{\partial u_{ij}}$$

$$= -\sum_{k=1}^{m}(t_k - o_k)\,\tau'(sum_{o_k})\frac{\partial sum_{o_k}}{\partial u_{ij}}$$

$$= -\sum_{k=1}^{m}(t_k - o_k)\,\tau'(sum_{o_k})\frac{\partial sum_{o_k}}{z_j}\frac{z_j}{\partial u_{ij}}$$

$$= -\sum_{k=1}^{m}(t_k - o_k)\,\tau'(sum_{o_k})v_{jk}\tau'\left(sum_{z_j}\right)x_i$$

$$= \underbrace{-\sum_{k=1}^{m}\delta_k\, v_{jk}\tau'\left(sum_{z_j}\right)}_{\eta_j}x_i$$

$$= -\eta_j x_i$$

① $sum_{o_k} = \sum_{j=1}^{p} v_{jk}z_j + v_{0k}$

② $\tau\left(sum_{o_k}\right) = o_k$

③ $sum_{z_j} = \sum_{i=1}^{d} u_{ij}x_i + u_{0j}$

④ $\tau\left(sum_{z_j}\right) = z_j$

⑤ u_{ij}가 미치는 영향