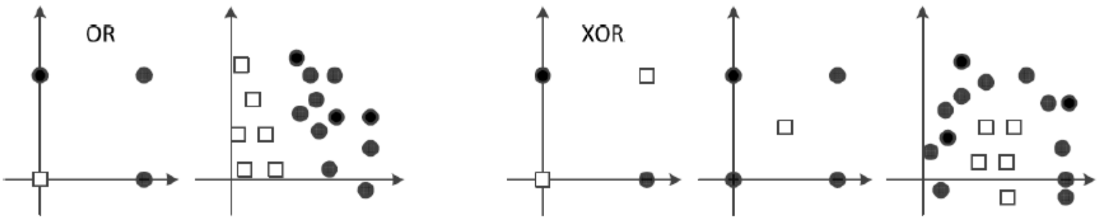
다층 퍼셉트론

- 퍼셉트론은 선형 분류기라는 **한계**
- 선형 분리 불가능한 상황에서는 일정한 양의 오류



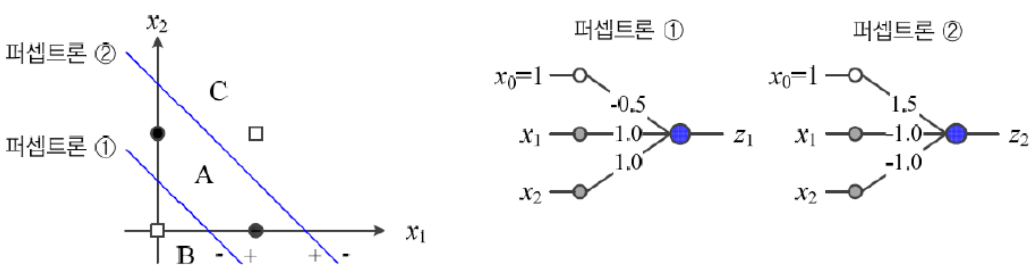
(a) 선형분리 가능 (b) 선형분리 불가능

그림 3-7 **선형분리가 가능한 상황과 불가능한 상황**

  - XOR 문제에서는 75%가 정확률 한계
- 1969년 민스키의 Perceptrons : 극복 방안 제시, 당시 기술로 실현 불가능
- 1974년 웨어보스의 박사 논문에서 오류 역전파 알고리즘 제안
- 1986년 루멜하트의 저서 Parallel Distributed Processing 다층 퍼셉트론 이론 정립, 신경망 부활
- **핵심 아이디어**
  - **은닉층**을 둔다. 은닉층은 원래 특징 공간을 분류하는 데 훨씬 **유리한 새로운 특징 공간**으로 변환
  - **시그모이드 활성함수를 도입**한다
    - 계단함수를 활성함수로 사용하면 경성 의사결정에 해당
    - 다층 퍼셉트론은 연성 의사결정이 가능한 **시그모이드 활성함수** 사용
      - **출력이 연속값**이고, 출력을 신뢰도로 간주함으로써 더 융통성 있게 의사결정
  - 오류 역전파 알고리즘 사용
    - 역방향으로 진행하면서 한 번에 한 층 씩 **그레디언트를 계산하고 가중치를 갱신**

특징 공간 변환

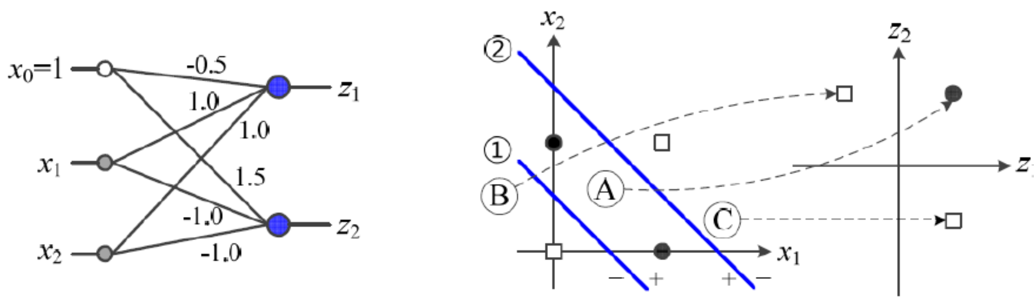
- 퍼셉트론 2개를 사용한 XOR 문제의 해결
  - 퍼셉트론 1과 퍼셉트론 2가 모두 +1이면 O 부류, 아니면 □ 부류



(a) 퍼셉트론 2개를 이용한 공간분할 (b) 퍼셉트론 2개

그림 3-8 **XOR 문제의 해결**

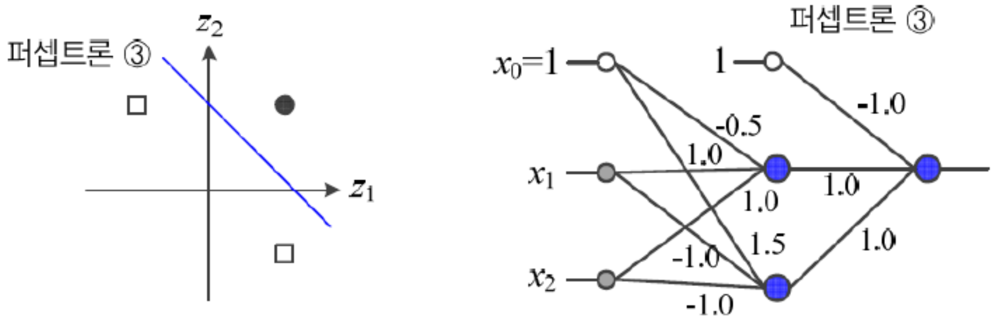
- 퍼셉트론 2개를 **병렬 결합**하면
  - **원래 공간** $\mathbf{x} = (x_1, x_2)^\mathrm{T}$를 **새로운 특징 공간** $\mathbf{z} = (z_1, z_2)^\mathrm{T}$로 **변환**
  - 새로운 특징 공간 z에서는 선형 분리 가능함



(a) 두 퍼셉트론을 병렬로 결합 (b) 원래 특징 공간 **x**를 새로운 특징 공간 **z**로 변환

그림 3-9 특징 공간의 변환

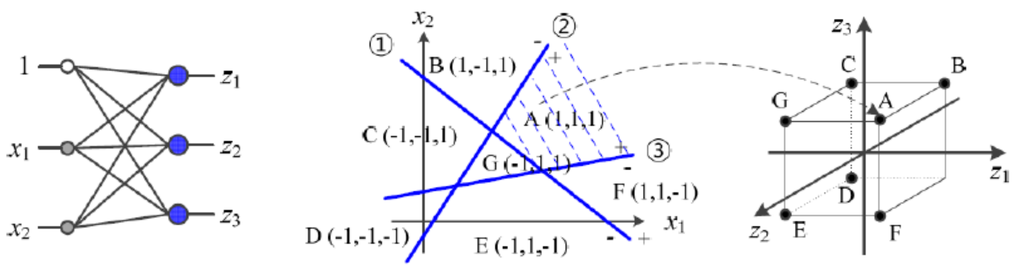
  - 사람이 수작업으로 특징 학습을 수행한 것과 유사함
- 이후, 퍼셉트론 1개를 순차 결합하면
  - 새로운 특징 공간 z에서 선형 분리를 수행하는 퍼셉트론3을 순차 결합하면 다층 퍼셉트론



(a) 새로운 특징 공간에서 분할 (b) 퍼셉트론 3개를 결합한 다층 퍼셉트론

그림 3-10 다층 퍼셉트론

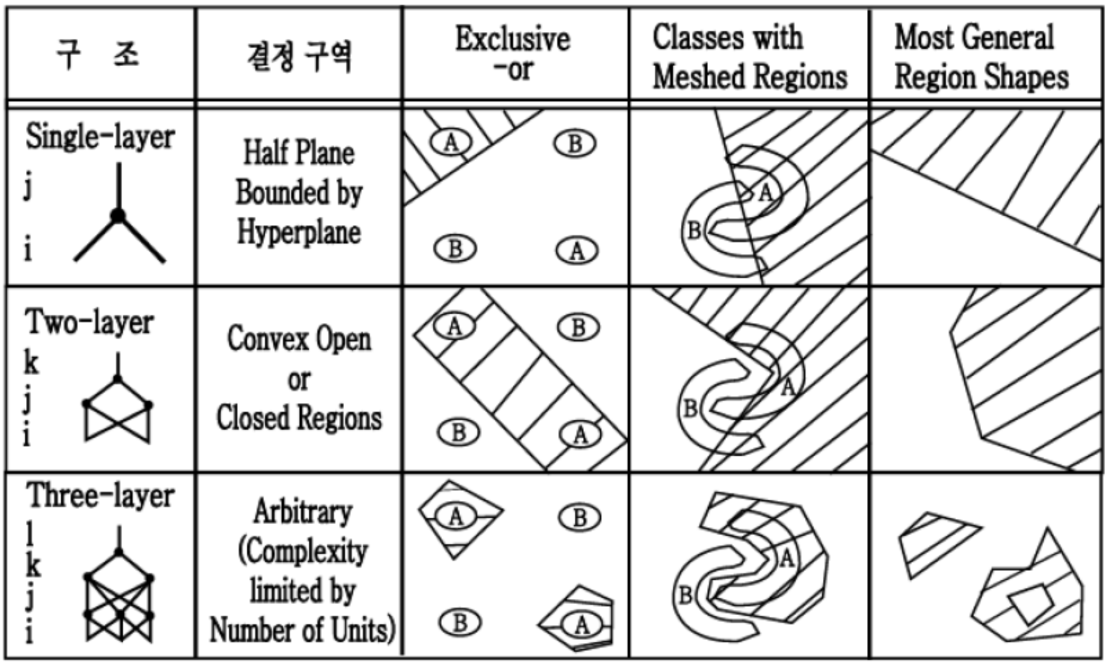
  - 이 다층 퍼셉트론은 훈련집합에 있는 4개 샘플 (0,0) (0,1) (1,0) (1,1)을 제대로 분류하는가?
- 다층 퍼셉트론의 용량
  - 3개 퍼셉트론을 결합하면, 2차원 공간을 7개 영역으로 나누고 각 영역을 3차원 점으로 변환
  - 활섬함수 τ로 계단함수를 사용하므로 영역을 점으로 변환



(a) 퍼셉트론 3개를 결합 (b) 7개 부분공간으로 나눔 (c) 3차원 공간의 점으로 매핑

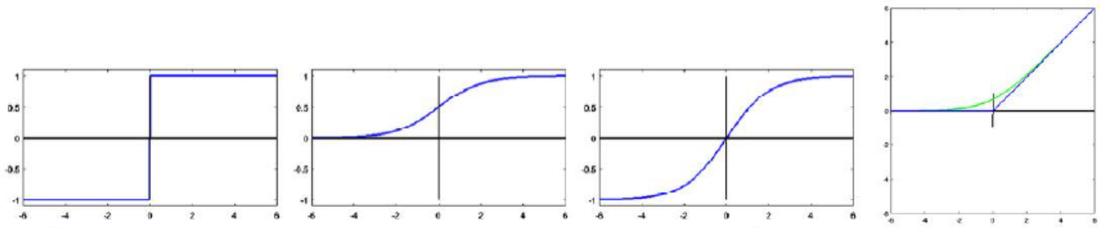
그림 3-11 퍼셉트론을 3개 결합했을 때 공간 변환

  - 일반화하여, **p개 퍼셉트론을 결합**하면 **p차원 공간으로 변환**
    - $1+\sum_{i=1}^{p} i$ **개의 영역**으로 **분할**



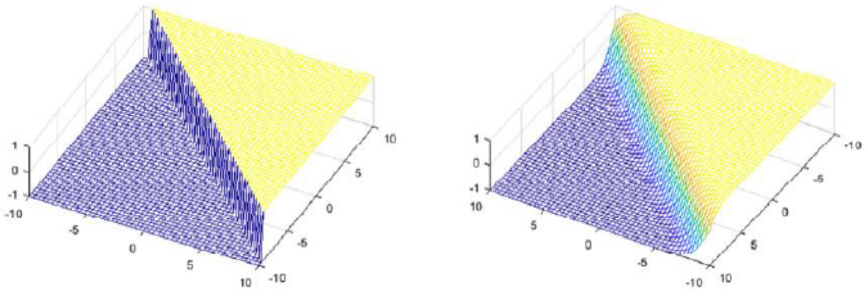
활성 함수

- **딱딱한 공간 분할**과 **부드러운 공간 분할**
  - 계단함수는 딱딱한 의사결정(영역을 점으로 변환)
  - 나머지 활성함수는 부드러운 의사결정(영역을 영역으로 변환)



(a) 계단 함수 (b) 로지스틱 시그모이드 (c) 하이퍼볼릭 탄젠트 시그모이드 (d) softplus와 rectifier

그림 3-12 신경망이 사용하는 활성함수



(a) 계단함수의 딱딱한 공간 분할 (b) 로지스틱 시그모이드의 부드러운 공간 분할

그림 3-13 퍼셉트론의 공간 분할 유형

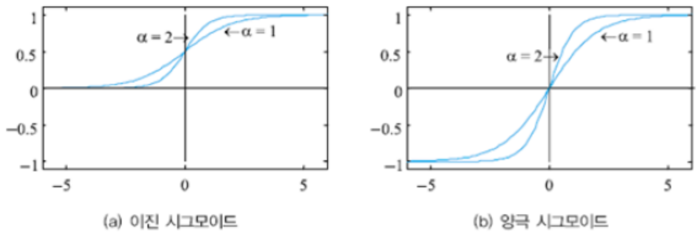
- 대표적인 **비선형 함수**인 **시그모이드**를 활성함수로 사용

이진 시그모이드 함수:

$$\tau_1(x) = \frac{1}{1+e^{-\alpha x}}$$
$$\tau_1'(x) = \alpha\tau_1(x)(1-\tau_1(x))$$

양극 시그모이드 함수:

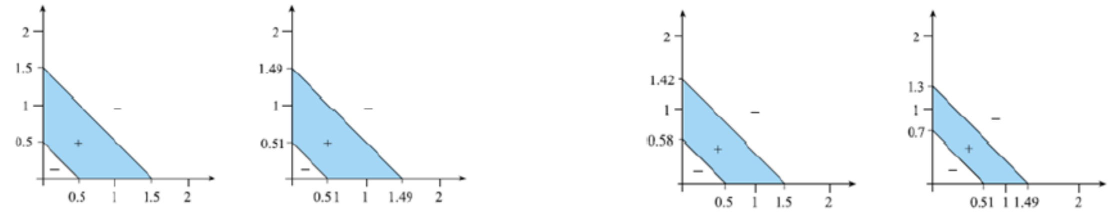
$$\tau_2(x) = \frac{2}{1+e^{-\alpha x}} - 1$$
$$\tau_2'(x) = \frac{\alpha}{2}(1+\tau_2(x))(1-\tau_2(x))$$



(a) 이진 시그모이드 (b) 양극 시그모이드

활성 함수로 널리 사용되는 두 가지 시그모이드 함수

- **활성 함수**에 따른 다층 퍼셉트론의 **공간 분할 능력 변화**



(a) 계단 함수 (양극 시그모이드 α=∞) (b) 양극 시그모이드 α=5 (c) 양극 시그모이드 α=3 (d) 양극 시그모이드 α=25

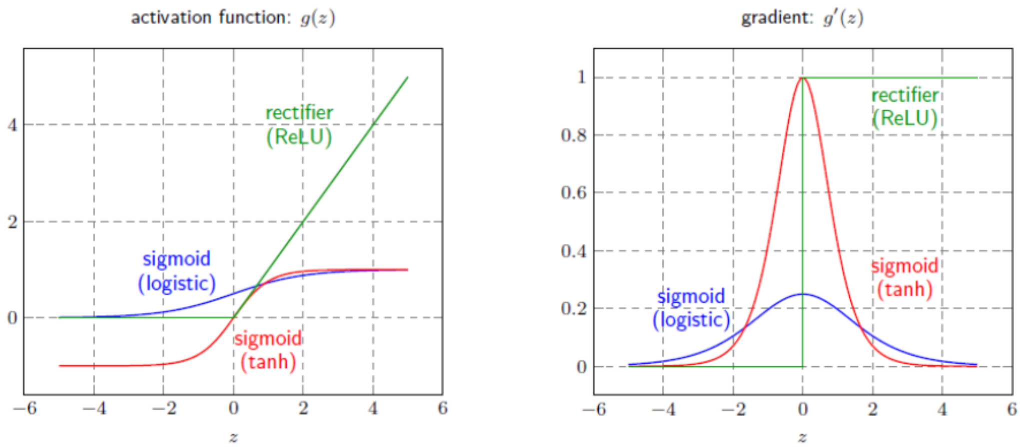
활성 함수에 따른 다층 퍼셉트론의 공간 분할 능력

- 신경망이 사용하는 **다양한 활성함수**
    - 로지스틱 시그모이드와 하이퍼볼릭 탄젠트는 a가 커질수록 계단함수에 가까워짐
    - 모두 1차 도함수 계산이 빠름 (특히 ReLU는 비교 연산 한 번)

표 3-1 활성함수로 사용되는 여러 함수

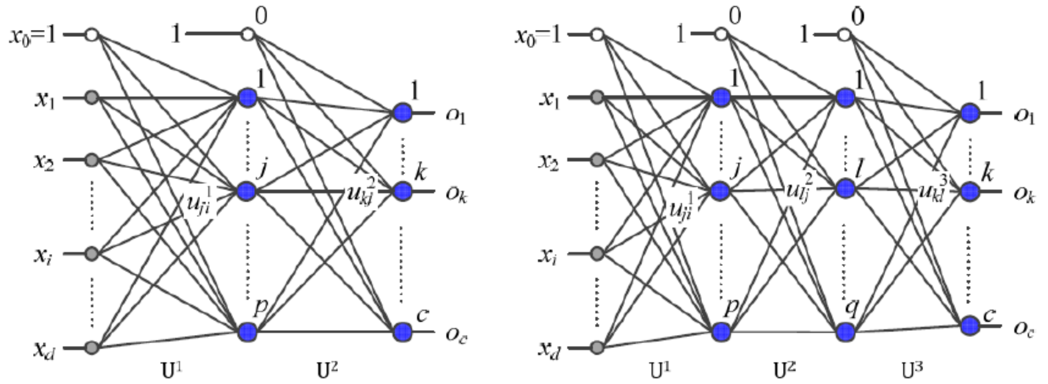
함수 이름	함수	1차 도함수	범위
계단	$\tau(s)=\begin{cases}1 & s\geq 0\\ -1 & s<0\end{cases}$	$\tau'(s)=\begin{cases}0 & s\neq 0\\ \text{불가} & s=0\end{cases}$	-1과 1
로지스틱 시그모이드	$\tau(s)=\dfrac{1}{1+e^{-as}}$	$\tau'(s)=a\tau(s)(1-\tau(s))$	(0,1)
하이퍼볼릭 탄젠트	$\tau(s)=\dfrac{2}{1+e^{-as}}-1$	$\tau'(s)=\dfrac{a}{2}(1-\tau(s)^2)$	(-1,1)
소프트플러스	$\tau(s)=\log_e(1+e^s)$	$\tau'(s)=\dfrac{1}{1+e^{-s}}$	(0, ∞)
렉티파이어(ReLU)	$\tau(s)=\max(0,s)$	$\tau'(s)=\begin{cases}0 & s<0\\ 1 & s>0\\ \text{불가} & s=0\end{cases}$	[0, ∞)

    - 퍼셉트론은 계단함수, 다층 퍼셉트론은 로지스틱 시그모이드와 하이퍼볼릭 탄젠트, **딥러닝**은 **ReLU**(Rectified Linear Activation)를 주로 사용
- 일반적으로 은닉층에서 **로지스틱 시그모이드**를 활성함수로 많이 사용
    - **시그모이드의 넓은 포화곡선**은 **그레디언트 기반한 학습을 어렵게 함**
    - ReLU 대두됨



구조

- 입력층-은닉층-출력층의 2층 구조
    - d+1개의 입력 노드(d는 특징의 개수). c개의 출력 노드(c는 부류 개수)
    - p개의 은닉 노드 : p는 하이퍼 매개변수(사용자가 정해주는 매개변수)
        - p가 **너무 크면 과잉적합, 너무 작으면 과소적합 ▶ 하이퍼 매개변수 최적화**



(a) 2층 퍼셉트론 (b) 3층 퍼셉트론

- b는 3층구조
- 다층 퍼셉트론의 **매개변수 (가중치)**
  - **입력층 - 은닉층**을 연결하는
    $\mathbf{U}^1$ (u_{ji}^1은 입력층의 i번째 노드를 은닉층의 j번째 노드와 연결)
  - **은닉층 - 출력층**을 연결하는
    $\mathbf{U}^2$ (u_{kj}^2는 은닉층의 j번째 노드를 출력층의 k번째 노드와 연결)

2층 퍼셉트론의 가중치 행렬:

$$\mathbf{U}^1 = \begin{pmatrix} u_{10}^1 & u_{11}^1 & \cdots & u_{1d}^1 \\ u_{20}^1 & u_{21}^1 & \cdots & u_{2d}^1 \\ \vdots & \vdots & \ddots & \vdots \\ u_{p0}^1 & u_{p1}^1 & \cdots & u_{pd}^1 \end{pmatrix}, \quad \mathbf{U}^2 = \begin{pmatrix} u_{10}^2 & u_{11}^2 & \cdots & u_{1p}^2 \\ u_{20}^2 & u_{21}^2 & \cdots & u_{2p}^2 \\ \vdots & \vdots & \ddots & \vdots \\ u_{c0}^2 & u_{c1}^2 & \cdots & u_{cp}^2 \end{pmatrix} \qquad (3.11)$$

  - 일반화하면 u_{ji}^l은 l-1번째 은닉층의 i번째 노드를 l번째 은닉층의 j번째 노드와 연결하는 가중치
    - 입력층을 0번째 은닉층, 출력층을 마지막 은닉층으로 간주

동작

- **특징 벡터** x를 **출력 벡터** o로 **매핑하는 함수**로 간주할 수 있음

$$\left.\begin{array}{l} \text{2층 퍼셉트론: } \mathbf{o} = \mathbf{f}(\mathbf{x}) = \mathbf{f}_2\left(\mathbf{f}_1(\mathbf{x})\right) \\ \text{3층 퍼셉트론: } \mathbf{o} = \mathbf{f}(\mathbf{x}) = \mathbf{f}_3\left(\mathbf{f}_2\left(\mathbf{f}_1(\mathbf{x})\right)\right) \end{array}\right\} \qquad (3.12)$$

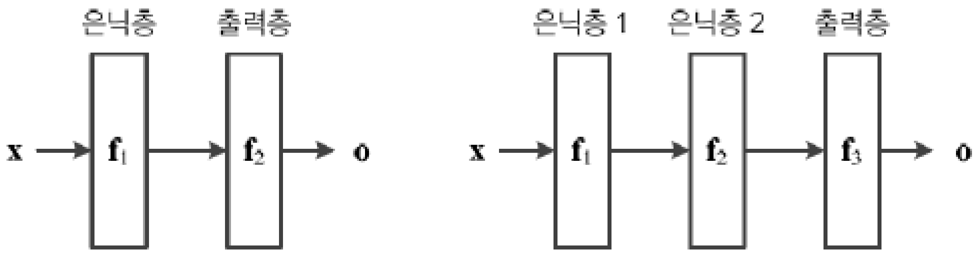


그림 3-15 다층 퍼셉트론을 간략화한 구조

- **깊은 신경망은** $\mathbf{o} = \mathbf{f}_L\left(\cdots \mathbf{f}_2\left(\mathbf{f}_1(\mathbf{x})\right)\right), L \geq 4$ (딥러닝)

- 노드가 수행하는 연산을 구체적으로 쓰면

  j번째 은닉 노드의 연산:

  $$z_j = \tau(zsum_j),\ j = 1,2,\cdots,p \tag{3.13}$$

  이때 $zsum_j = \mathbf{u}_j^1\mathbf{x}$이고 $\mathbf{u}_j^1 = (u_{j0}^1, u_{j1}^1, \cdots, u_{jd}^1)$, $\mathbf{x} = (1, x_1, x_2, \cdots, x_d)^\mathrm{T}$

  k번째 출력 노드의 연산:

  $$o_k = \tau(osum_k),\ k = 1,2,\cdots,c \tag{3.14}$$

  이때 $osum_k = \mathbf{u}_k^2\mathbf{z}$이고 $\mathbf{u}_k^2 = (u_{k0}^2, u_{k1}^2, \cdots, u_{kp}^2)$, $\mathbf{z} = (1, z_1, z_2, \cdots, z_p)^\mathrm{T}$

  - $\mathbf{u}_j^1$은 j번째 은닉 노드에 연결된 가중치 벡터 (식 (3.11)의 $\mathbf{U}^1$의 j번째 행)
  - $\mathbf{u}_k^2$는 k번째 출력 노드에 연결된 가중치 벡터 (식 (3.11)의 $\mathbf{U}^2$의 k번째 행)
- 다층 퍼셉트론의 동작을 행렬로 표기하면

  $$\mathbf{o} = \boldsymbol{\tau}\left(\mathbf{U}^2\boldsymbol{\tau}_h(\mathbf{U}^1\mathbf{x})\right) \tag{3.15}$$
- **은닉층**은 **특징 추출기**
  - 은닉층은 **특징 벡터**를 분류에 더 유리한 **새로운 특징 공간으로 변환**
  - 현대 기계 학습에서는 **특징 학습**이라 (feature learning 혹은 data-driven features) 부름
    - 딥러닝은 더 많은 단계를 거쳐 특징학습을 함

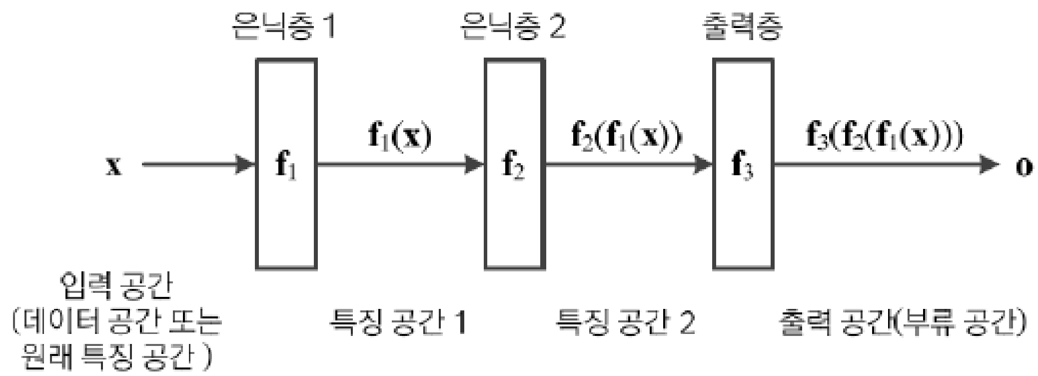
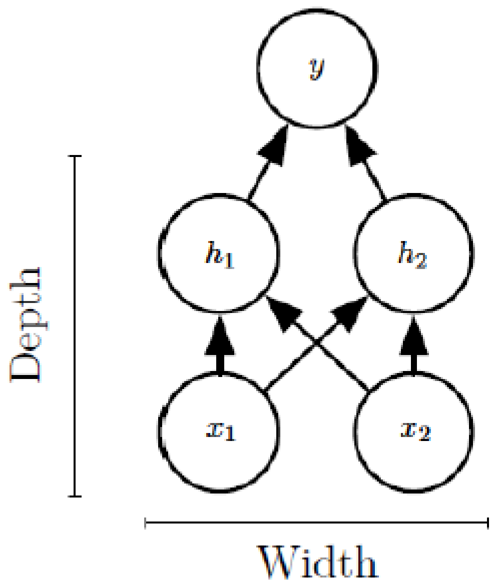


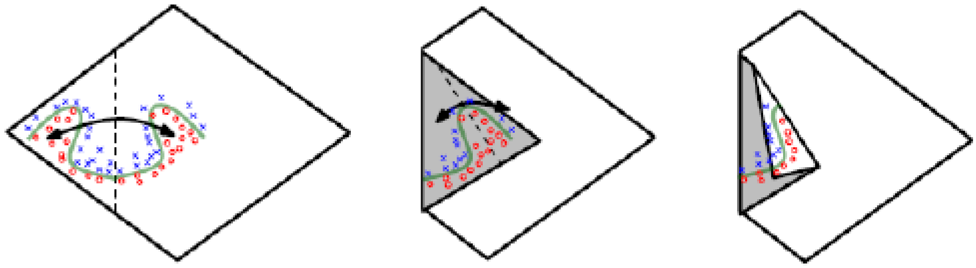
그림 3-16 특징 추출기로서의 은닉층

- 기본 구조

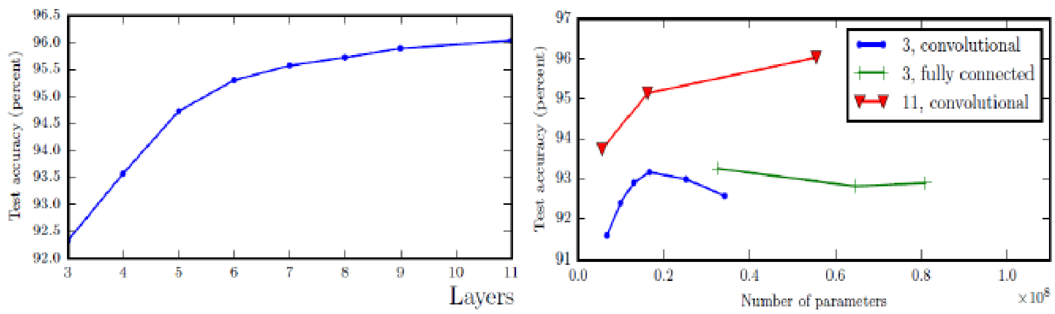


  - **범용적 근사 이론**

        - 하나의 **은닉층**은 **함수의 근사**를 표현
        - **다층 퍼셉트론**도 **공간을 변환하는 함수를 근사**함
    - **얕은 은닉층의 구조**보다
        - 지수적으로 더 넓은 폭이 필요할 수 있음
        - 더 과잉적합 되기 쉬움
        - 일반적으로 **깊은 은닉층의 구조가 좋은 성능**을 가짐
- **은닉층의 깊이**에 따른 **이점**
    - **지수의 표현**



        - 각 **은닉층**은 입력 **공간을 어디서 접을지 지정** ▶ 지수적으로 **많은 선형적인 영역 조각들**
    - **일반화 성능 향상**과 **과잉적합 해소**



오류 역전파 알고리즘

목적함수의 정의

- 훈련집합
    - 특징 벡터 집합 $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}$과 부류 벡터 집합 $\mathbb{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n\}$
    - **부류 벡터**는 **원핫 코드**로 표현됨, 즉 $\mathbf{y}_i = (0,0,\cdots,1,\cdots,0)^\mathrm{T}$
    - 설계 행렬로 쓰면

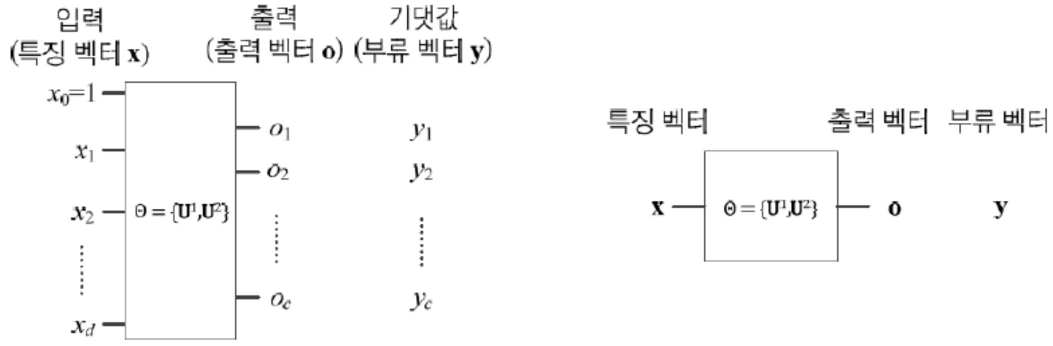
$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1^\mathrm{T} \\ \mathbf{x}_2^\mathrm{T} \\ \vdots \\ \mathbf{x}_n^\mathrm{T} \end{pmatrix}, \quad \mathbf{Y} = \begin{pmatrix} \mathbf{y}_1^\mathrm{T} \\ \mathbf{y}_2^\mathrm{T} \\ \vdots \\ \mathbf{y}_n^\mathrm{T} \end{pmatrix} \tag{3.16}$$

- 기계 학습의 목표
    - 모든 **샘플을 옳게 분류**하는 **함수 f를 찾는 일**

$$\left.\begin{array}{l} \mathbf{Y} = \mathbf{f}(\mathbf{X}) \\ \text{풀어 쓰면} \quad \mathbf{y}_i = \mathbf{f}(\mathbf{x}_i), i = 1,2,\cdots,n \end{array}\right\} \tag{3.17}$$

- 목적함수
    - **평균 제곱 오차**로 정의

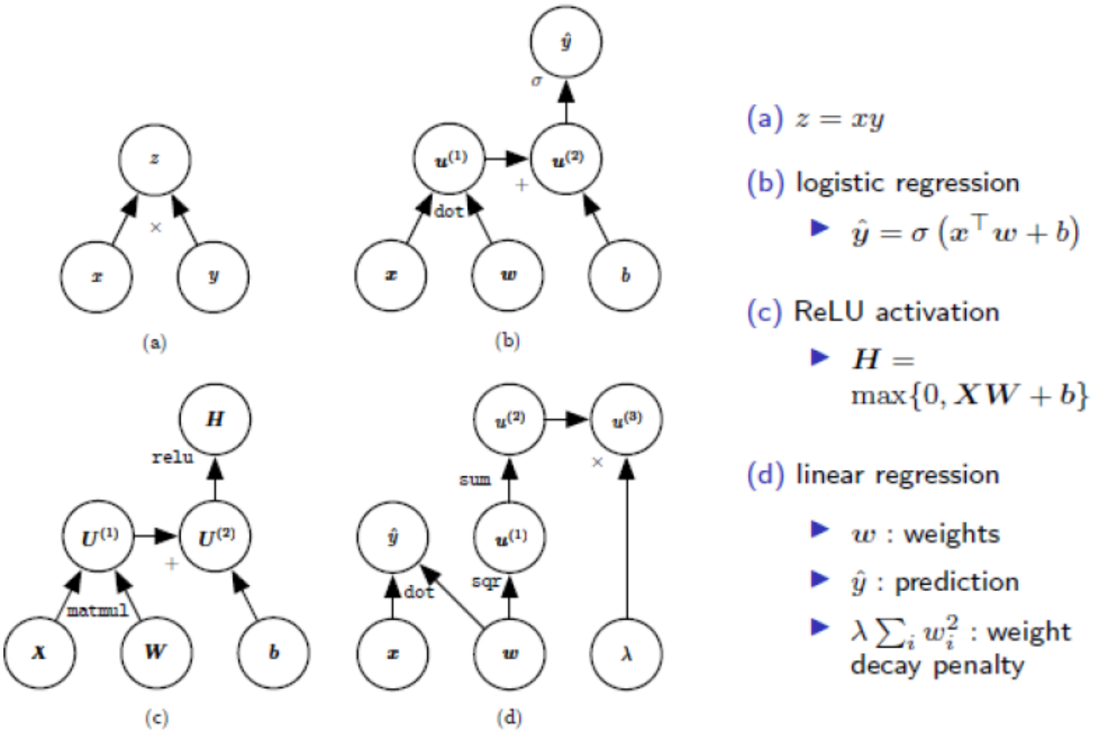
$$
\left.\begin{aligned}
\text{온라인 모드:} \quad & e = \frac{1}{2}\|\mathbf{y} - \mathbf{o}\|_2^2 \\
\text{배치 모드:} \quad & e = \frac{1}{2n}\sum_{i=1}^{n}\|\mathbf{y}_i - \mathbf{o}_i\|_2^2
\end{aligned}\right\} \tag{3.19}
$$



(a) 블록 다이어그램 (b) 축약형 블록 다이어그램

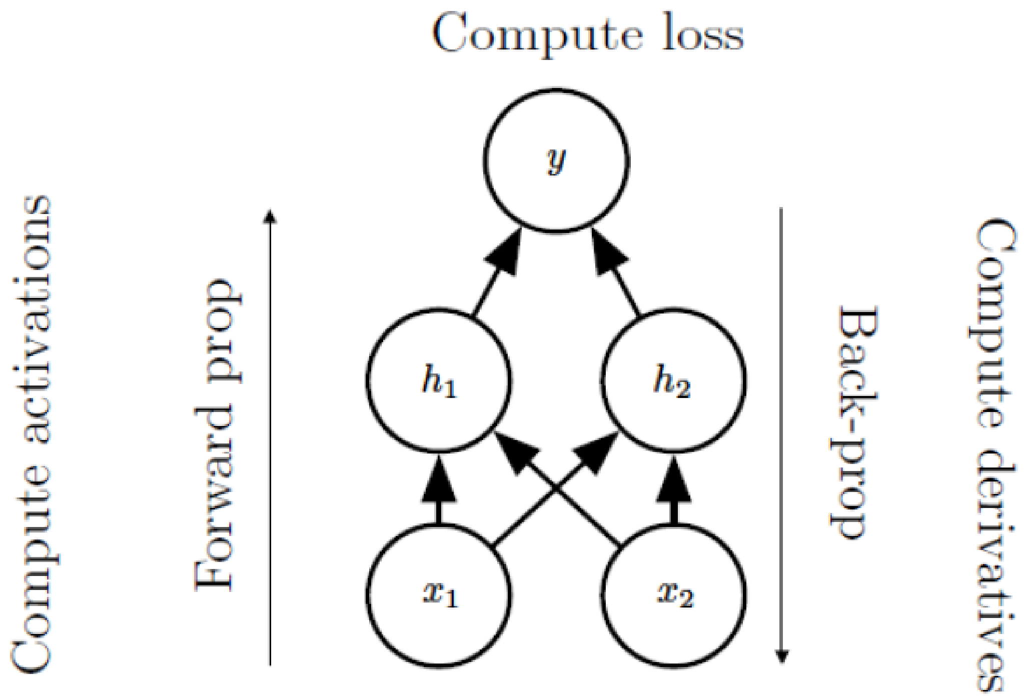
그림 3-17 목적함수 정의에 사용하는 입력과 출력, 기댓값

- **연산 그래프**의 예 : 연산을 그래프로 표현



오류 역전파 알고리즘 설계

- 간단한 오류 역전파의 연산 그래프 예



- **오류 역전파 미분의 연쇄 법칙을 이용**
  - **연쇄 법칙**
    - 수인 경우 : $\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}.$
    - **벡터인 경우** : $\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^{\top} \nabla_{\boldsymbol{y}} z.$

$\boldsymbol{x} \in \mathbb{R}^m$, $\boldsymbol{y} \in \mathbb{R}^n$ 라고 가정하면

$g : \mathbb{R}^m \mapsto \mathbb{R}^n$

$f : \mathbb{R}^n \mapsto \mathbb{R}$

$$\rightarrow \quad \boldsymbol{y} = g(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^m \mapsto \mathbb{R}^n \ni \boldsymbol{y}$$

$$\begin{aligned} \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} &= \begin{bmatrix} \frac{\partial \boldsymbol{y}}{\partial x_1} & \cdots & \frac{\partial \boldsymbol{y}}{\partial x_m} \end{bmatrix} \\ &= \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_n}{\partial x_1} & \cdots & \frac{\partial y_n}{\partial x_m} \end{bmatrix} \in \mathbb{R}^{n \times m} \end{aligned}$$

$$\nabla_{\boldsymbol{y}} z = \begin{bmatrix} \frac{\partial z}{\partial y_1} & \cdots & \frac{\partial z}{\partial y_n} \end{bmatrix}^\top \in \mathbb{R}^{n \times 1}$$

$$\left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^\top \nabla_{\boldsymbol{y}} z = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_n}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_m} & \cdots & \frac{\partial y_n}{\partial x_m} \end{bmatrix} \begin{bmatrix} \frac{\partial z}{\partial y_1} \\ \vdots \\ \frac{\partial z}{\partial y_n} \end{bmatrix}$$

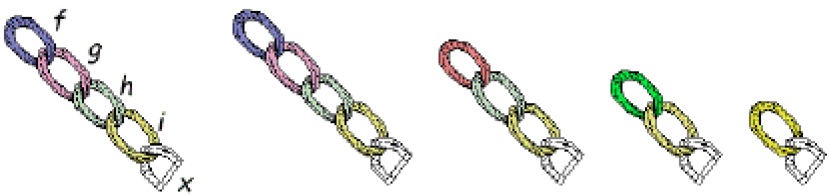
$$= \begin{bmatrix} \frac{\partial z}{\partial y_1}\frac{\partial y_1}{\partial x_1} + \cdots + \frac{\partial z}{\partial y_n}\frac{\partial y_n}{\partial x_1} \\ \vdots \\ \frac{\partial z}{\partial y_1}\frac{\partial y_1}{\partial x_m} + \cdots + \frac{\partial z}{\partial y_n}\frac{\partial y_n}{\partial x_m} \end{bmatrix} = \begin{bmatrix} \frac{\partial z}{\partial x_1} \\ \vdots \\ \frac{\partial z}{\partial x_m} \end{bmatrix} = \nabla_{\boldsymbol{x}} z$$

- **야코비안 행렬과 그레디언트를 곱한 연쇄 법칙**을 얻어서 구해짐

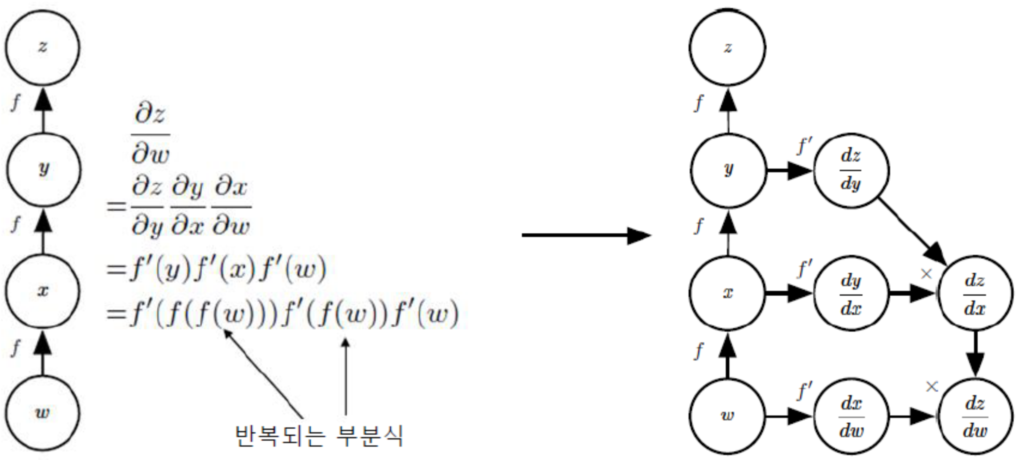
$$\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^\top \nabla_{\boldsymbol{y}} z \qquad \underbrace{\boldsymbol{x} \in \mathbb{R}^m \overset{g}{\mapsto} \boldsymbol{y}}_{\text{Jacobian: } \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}} \in \underbrace{\mathbb{R}^n \overset{f}{\mapsto} \mathbb{R} \ni z}_{\text{gradient: } \nabla_{\boldsymbol{y}} z}$$

- 연쇄법칙의 **구현**
  - **반복되는 부분식들**을 저장하거나 **재연산을 최소화**
    - ex) 동적 프로그래밍
    - 연산속도와 저장 공간의 Trade-off

$$(f \circ g \circ h \circ i)'(x) = (f' \circ g \circ h \circ i)(x) \times (g' \circ h \circ i)(x) \times (h' \circ i)(x) \times i'(x)$$



  - 그레디언트 계산을 위한 연산 그래프 예



- 식 (3.19)의 목적함수를 다시 쓰면
  - 2층 퍼셉트론의 경우 $\Theta = \{\mathbf{U}^1, \mathbf{U}^2\}$

$$J(\Theta) = \frac{1}{2}\|\mathbf{y} - \mathbf{o}(\Theta)\|_2^2 \qquad (3.20)$$

- 경사하강법

$J(\Theta) = J(\{\mathbf{U}^1, \mathbf{U}^2\})$의 최저점을 찾아주는 경사 하강법

$$\left.\begin{aligned}\mathbf{U}^1 &= \mathbf{U}^1 - \rho\frac{\partial J}{\partial \mathbf{U}^1}\\ \mathbf{U}^2 &= \mathbf{U}^2 - \rho\frac{\partial J}{\partial \mathbf{U}^2}\end{aligned}\right\} \tag{3.21}$$

- 식 (3.21)을 알고리즘 형태로 쓰면

알고리즘 3-3 다층 퍼셉트론을 위한 스토케스틱 경사 하강법

입력: 훈련집합 $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}$, $\mathbb{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n\}$, 학습률 ρ
출력: 가중치 행렬 $\mathbf{U}^1$과 $\mathbf{U}^2$

```
1  U¹과 U²를 초기화한다.
2  repeat
3      𝕏의 순서를 섞는다.
4      for (𝕏의 샘플 각각에 대해)
5          식 (3.15)로 전방 계산을 하여 o를 구한다.
6          ∂J/∂U¹와 ∂J/∂U²를 계산한다.
7          식 (3.21)로 U¹과 U²를 갱신한다.
8  until (멈춤 조건)
```

- 오류 역전파의 **유도**
  - 알고리즘 3-3의 라인 6을 위한 도함수 값 $\frac{\partial J}{\partial \mathbf{U}^1}$와 $\frac{\partial J}{\partial \mathbf{U}^2}$의 계산 과정
  - 먼저 $\mathbf{U}^2$를 구성하는 u_{kj}^2로 미분하면,

$$\begin{aligned}\frac{\partial J}{\partial u_{kj}^2} &= \frac{\partial(0.5\|\mathbf{y}-\mathbf{o}(\mathbf{U}^1,\mathbf{U}^2)\|_2^2)}{\partial u_{kj}^2}\\ &= \frac{\partial\left(0.5\sum_{q=1}^{c}(y_q - o_q)^2\right)}{\partial u_{kj}^2}\\ &= \frac{\partial(0.5(y_k - o_k)^2)}{\partial u_{kj}^2}\\ &= -(y_k - o_k)\frac{\partial o_k}{\partial u_{kj}^2}\\ &= -(y_k - o_k)\frac{\partial \tau(osum_k)}{\partial u_{kj}^2}\\ &= -(y_k - o_k)\tau'(osum_k)\frac{\partial osum_k}{\partial u_{kj}^2}\\ &= -(y_k - o_k)\tau'(osum_k)z_j\end{aligned}$$

mse

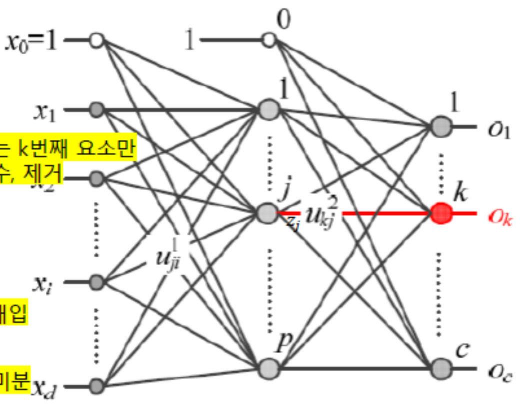
그림처럼
관심이 있는 k번째 요소만
나머지 상수, 제거

연쇄법칙

식 (3.14) 대입

활성함수 미분

osum 미분



(a) u_{kj}^2가 미치는 영향

그림 3-18 매개변수가 미치는 영향

$$o_k = \tau(osum_k),\ k = 1,2,\cdots,c \tag{3.14}$$
이때 $osum_k = \mathbf{u}_k^2\mathbf{z}$이고 $\mathbf{u}_k^2 = (u_{k0}^2, u_{k1}^2, \cdots, u_{kp}^2)$, $\mathbf{z} = (1, z_1, z_2, \cdots, z_p)^T$

- $\mathbf{U}^1$을 구성하는 u_{ji}^1로 미분하면

$$o_k = \tau(osum_k),\ k = 1,2,\cdots,c \tag{3.14}$$

이때 $osum_k = \mathbf{u}_k^2\mathbf{z}$이고 $\mathbf{u}_k^2 = (u_{k0}^2, u_{k1}^2, \cdots, u_{kp}^2)$, $\mathbf{z} = (1, z_1, z_2, \cdots, z_p)^\mathrm{T}$

$$z_j = \tau(zsum_j),\ j = 1,2,\cdots,p \tag{3.13}$$

이때 $zsum_j = \mathbf{u}_j^1\mathbf{x}$이고 $\mathbf{u}_j^1 = (u_{j0}^1, u_{j1}^1, \cdots, u_{jd}^1)$, $\mathbf{x} = (1, x_1, x_2, \cdots, x_d)^\mathrm{T}$

$$\frac{\partial J}{\partial u_{ji}^1} = \frac{\partial(0.5\|\mathbf{y}-\mathbf{o}(\mathbf{U}^1,\mathbf{U}^2)\|_2^2)}{\partial u_{ji}^1}$$

$$= \frac{\partial\left(0.5\sum_{q=1}^{c}(y_q-o_q)^2\right)}{\partial u_{ji}^1}$$

연쇄법칙
연결된 것만 관심
나머진 상수, 제거

$$= -\sum_{q=1}^{c}(y_q-o_q)\frac{\partial o_q}{\partial u_{ji}^1}$$

식 (3.14) 대입,
활성함수 미분

$$= -\sum_{q=1}^{c}(y_q-o_q)\tau'(osum_q)\frac{\partial osum_q}{\partial u_{ji}^1}$$

$$= -\sum_{q=1}^{c}(y_q-o_q)\tau'(osum_q)\frac{\partial osum_q}{\partial z_j}\frac{\partial z_j}{\partial u_{ji}^1}$$

미분하면 가중치만 남음

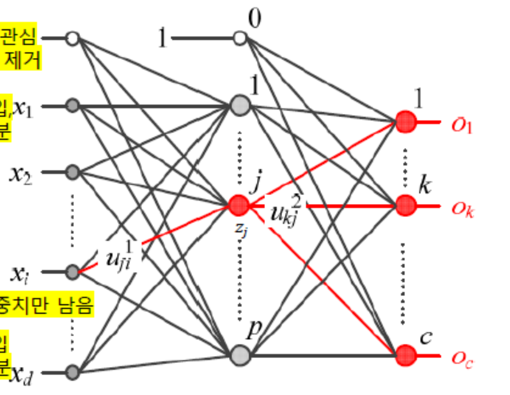
$$= -\sum_{q=1}^{c}(y_q-o_q)\tau'(osum_q)u_{qj}^2\frac{\partial z_j}{\partial u_{ji}^1}$$

식 (3.13) 대입
활성함수 미분
zsum 미분

$$= -\sum_{q=1}^{c}(y_q-o_q)\tau'(osum_q)u_{qj}^2\tau'(zsum_j)x_i$$

항 정리

$$= -\tau'(zsum_j)x_i\sum_{q=1}^{c}(y_q-o_q)\tau'(osum_q)u_{qj}^2$$



(b) u_{ji}^1이 미치는 영향

- 지금까지 유도한 식을 정리하면

$$\delta_k = (y_k - o_k)\tau'(osum_k), \qquad 1 \le k \le c \tag{3.22}$$

반복되는 부분식

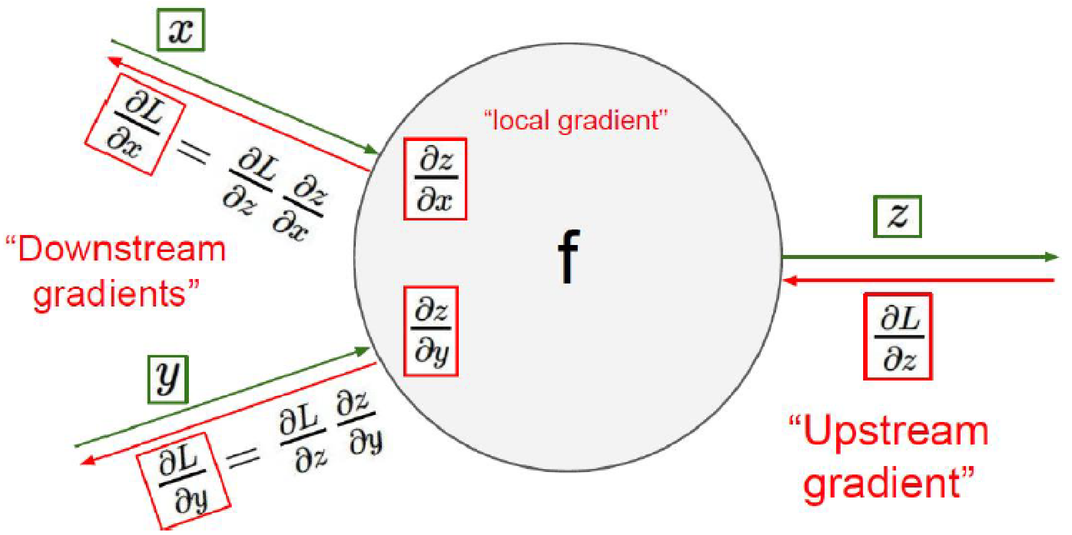
$$\frac{\partial J}{\partial u_{kj}^2} = \Delta u_{kj}^2 = -\delta_k z_j, \qquad 0 \le j \le p, 1 \le k \le c \tag{3.23}$$

$$\eta_j = \tau'(zsum_j)\sum_{q=1}^{c}\delta_q u_{qj}^2, \qquad 1 \le j \le p \tag{3.24}$$

$$\frac{\partial J}{\partial u_{ji}^1} = \Delta u_{ji}^1 = -\eta_j x_i, \qquad 0 \le i \le d, 1 \le j \le p \tag{3.25}$$

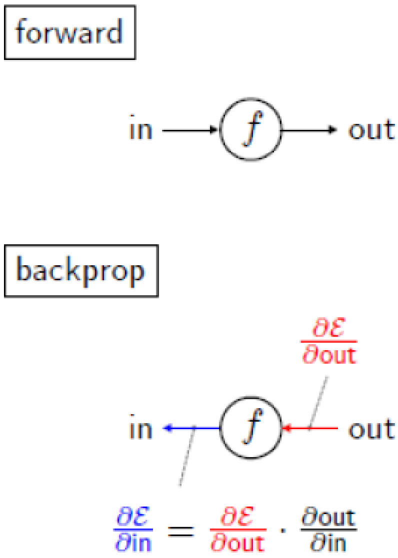
- 오류 역전파 알고리즘
  - 식 (3.22) ~ (3.25)를 이용하여 **출력층의 오류**를 **역방향(왼쪽)으로 전파**하며 **그레이디언트를 계산**하는 알고리즘

- 역전파 분해



- **단일 노드**의 역전파 예

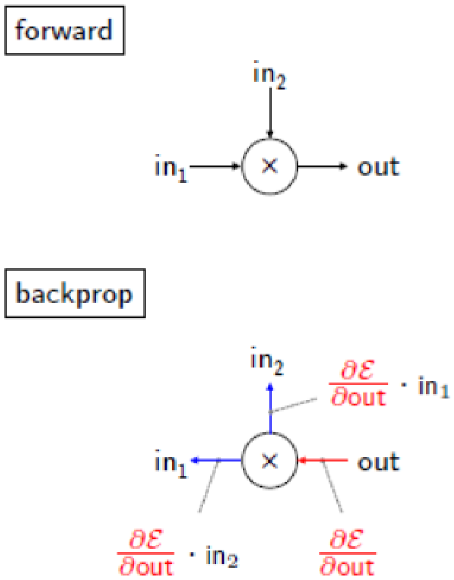
$$\text{out} = f(\text{in})$$



$$\frac{\partial \mathcal{E}}{\partial \text{in}} = \underbrace{\frac{\partial \mathcal{E}}{\partial \text{out}}}_{\substack{\text{output}\\ \text{gradient}}} \cdot \underbrace{\frac{\partial \text{out}}{\partial \text{in}}}_{\substack{\text{local}\\ \text{gradient}}} = \frac{\partial \mathcal{E}}{\partial \text{out}} \cdot f'(\text{in})$$

- **곱셈**의 역전파 예

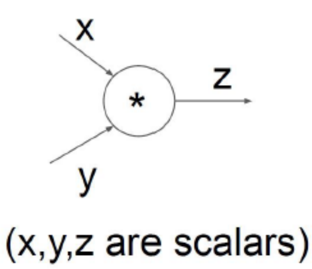
$$\text{out} = \text{in}_1 \cdot \text{in}_2$$



$$\frac{\partial \mathcal{E}}{\partial \text{in}_1} = \frac{\partial \mathcal{E}}{\partial \text{out}} \cdot \frac{\partial \text{out}}{\partial \text{in}_1} = \underbrace{\frac{\partial \mathcal{E}}{\partial \text{out}}}_{\substack{\text{output}\\ \text{gradient}}} \cdot \underbrace{\text{in}_2}_{\substack{\text{local}\\ \text{gradient}}}$$

$$\frac{\partial \mathcal{E}}{\partial \text{in}_2} = \frac{\partial \mathcal{E}}{\partial \text{out}} \cdot \frac{\partial \text{out}}{\partial \text{in}_2} = \frac{\partial \mathcal{E}}{\partial \text{out}} \cdot \text{in}_1$$

- 곱셈의 역전파 PyTorch 구현 예




```
class Multiply(torch.autograd.Function):
  @staticmethod
  def forward(ctx, x, y):
    ctx.save_for_backward(x, y)
    z = x * y
    return z
  @staticmethod
  def backward(ctx, grad_z):
    x, y = ctx.saved_tensors
    grad_x = y * grad_z    # dz/dx * dL/dz
    grad_y = x * grad_z    # dz/dy * dL/dz
    return grad_x, grad_y
```

Need to stash some values for use in backward

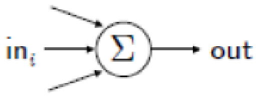
Upstream gradient

Multiply upstream and local gradients

- 덧셈의 역전파 예

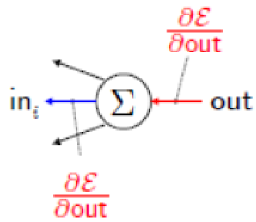
$$\text{out} = \sum_i \text{in}_i$$

forward



$$\begin{aligned}\frac{\partial \mathcal{E}}{\partial \text{in}_i} &= \frac{\partial \mathcal{E}}{\partial \text{out}} \cdot \frac{\partial \text{out}}{\partial \text{in}_i} \\ &= \underbrace{\frac{\partial \mathcal{E}}{\partial \text{out}}}_{\substack{\text{output}\\\text{gradient}}} \cdot \underbrace{1}_{\substack{\text{local}\\\text{gradient}}} \\ &= \frac{\partial \mathcal{E}}{\partial \text{out}} \\ &\triangleq \delta\end{aligned}$$

backprop



$$\text{sum (forward)} \Leftrightarrow \text{fanout (backprop)}$$

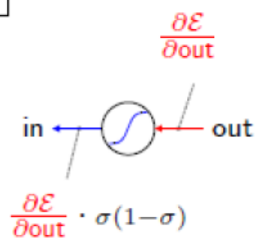
- **시그모이드**의 역전파 예

$$\text{out} = \sigma(\text{in})$$

forward

$$\begin{aligned}\frac{\partial \mathcal{E}}{\partial \text{in}} &= \frac{\partial \mathcal{E}}{\partial \text{out}} \cdot \frac{\partial \text{out}}{\partial \text{in}} \\ &= \underbrace{\frac{\partial \mathcal{E}}{\partial \text{out}}}_{\substack{\text{output}\\\text{gradient}}} \cdot \underbrace{\sigma'(\text{in})}_{\substack{\text{local}\\\text{gradient}}} \\ &= \frac{\partial \mathcal{E}}{\partial \text{out}} \cdot [\sigma(\text{in})\,(1 - \sigma(\text{in}))]\end{aligned}$$

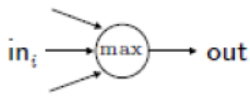
backprop



- **최대화**의 역전파 예

$$\mathsf{out} = \max_i \{\mathsf{in}_i\}$$

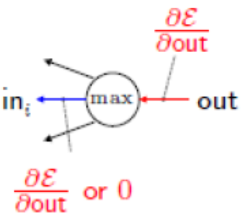
$\boxed{\text{forward}}$



$$\frac{\partial \mathcal{E}}{\partial \mathsf{in}_i} = \frac{\partial \mathcal{E}}{\partial \mathsf{out}} \cdot \underbrace{\frac{\partial \mathsf{out}}{\partial \mathsf{in}}}_{1 \text{ or } 0}$$

$$= \begin{cases} \frac{\partial \mathcal{E}}{\partial \mathsf{out}} & \text{if } \mathsf{in}_i \text{ is max} \\ 0 & \text{otherwise} \end{cases}$$

$\boxed{\text{backprop}}$

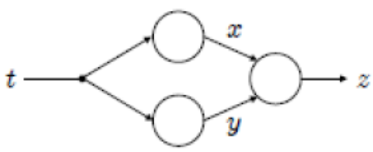


max (forward) $\Leftrightarrow$ mux (backprop)

- **전개**의 역전파 예

multivariable chain rule

$\boxed{\text{forward}}$



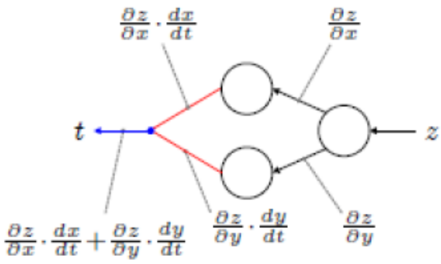
let

$$x = x(t), \;\; y = y(t)$$
$$z = f(x, y)$$

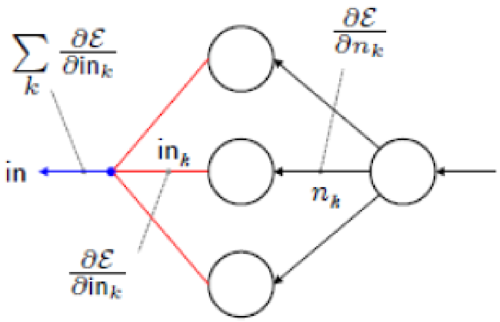
then

$$\frac{\partial z}{\partial t} = \frac{\partial z}{\partial x} \cdot \frac{\partial x}{\partial t} + \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial t}$$

$\boxed{\text{backprop}}$



fanout



fanout (forward) $\Leftrightarrow$ sum (backprop)

assuming

$$\mathcal{E} = f(n_1, \ldots, n_k, \ldots)$$

and

$$n_k = n_k(\text{in})$$

gives

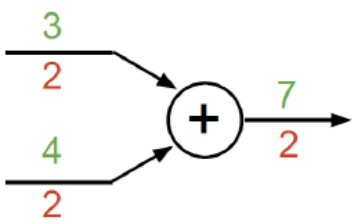
$$\frac{\partial \mathcal{E}}{\partial \text{in}} = \sum_k \frac{\partial \mathcal{E}}{\partial n_k} \cdot \frac{\partial n_k}{\partial \text{in}} = \sum_k \frac{\partial \mathcal{E}}{\partial \text{in}_k}$$

where

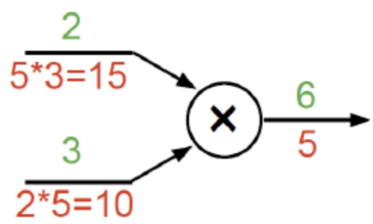
$$\text{in}_k \triangleq \text{ input to } n_k$$

- 역전파 주요 예

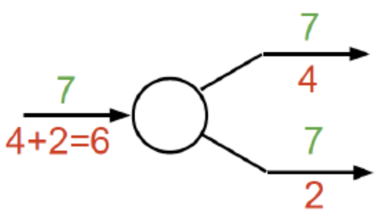
add gate: gradient distributor



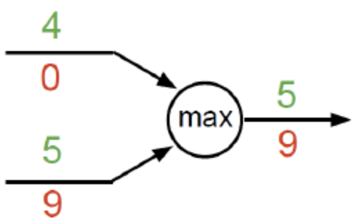
mul gate: “swap multiplier”



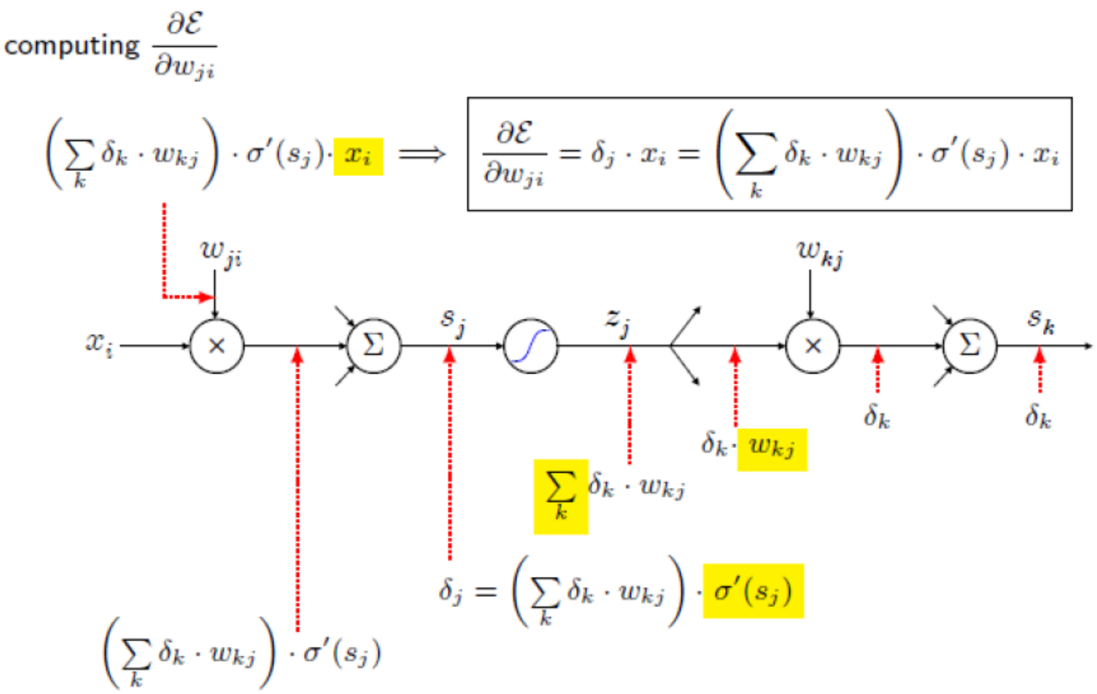
copy gate: gradient adder



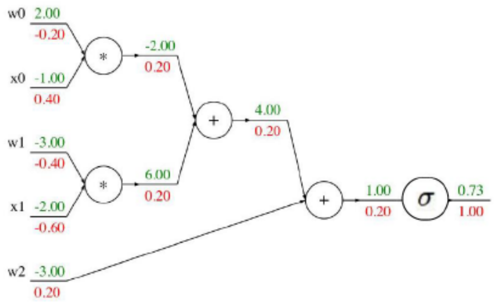
max gate: gradient router



- **실제 역전파** 예



- 역전파의 간단한 구현



Forward pass:
Compute output

```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)
```

Backward pass:
Compute grads

```
    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```

오류 역전파를 이용한 학습 알고리즘

- 식 (3.22) ~ (3.25)를 이용한 **스토캐스틱 경사 하강법**

알고리즘 3-4 다층 퍼셉트론 학습을 위한 스토캐스틱 경사 하강법

입력: 훈련집합 $\mathbb{X}$와 $\mathbb{Y}$, 학습률 ρ
출력: 가중치 행렬 $\mathbf{U}^1$과 $\mathbf{U}^2$

1 $\mathbf{U}^1$과 $\mathbf{U}^2$를 초기화한다.
2 repeat
3 　$\mathbb{X}$의 순서를 섞는다.
4 　for ($\mathbb{X}$의 샘플 각각에 대해)
5 　　현재 처리하는 샘플을 $\mathbf{x} = (x_0, x_1, x_2, \cdots, x_d)^\mathrm{T}$, $\mathbf{y} = (y_1, y_2, \cdots, y_c)^\mathrm{T}$라 표기한다.
6 　　x_0과 z_0을 1로 설정한다. // 바이어스
　　　　// 전방 계산
7 　　for (j=1 to p) $zsum_j = \mathbf{u}_j^1\mathbf{x}$, $z_j = \tau(zsum_j)$ // 식 (3.13)
8 　　for (k=1 to c) $osum_k = \mathbf{u}_k^2\mathbf{z}$, $o_k = \tau(osum_k)$ // 식 (3.14)
　　　　// 오류 역전파
9 　　for (k=1 to c) $\delta_k = (y_k - o_k)\tau'(osum_k)$ // 식 (3.22)
10 　　for (k=1 to c) for (j=0 to p) $\Delta u_{kj}^2 = -\delta_k z_j$ // 식 (3.23)
11 　　for (j=1 to p) $\eta_j = \tau'(zsum_j)\sum_{q=1}^{c} \delta_q u_{qj}^2$ // 식 (3.24)
12 　　for (j=1 to p) for (i=0 to d) $\Delta u_{ji}^1 = -\eta_j x_i$ // 식 (3.25)
　　　　// 가중치 갱신
13 　　for (k=1 to c) for (j=0 to p) $u_{kj}^2 = u_{kj}^2 - \rho\Delta u_{kj}^2$ // 식 (3.21)
14 　　for (j=1 to p) for (i=0 to d) $u_{ji}^1 = u_{ji}^1 - \rho\Delta u_{ji}^1$ // 식 (3.21)
15 until (멈춤 조건)

- 임의 샘플링 방식으로 바꾸려면

3. $\mathbb{X}$의 순서를 섞는다.
4. for ($\mathbb{X}$의 샘플 각각에 대해)
5. 　현재 처리하는 샘플을 $\mathbf{x} = (x_0, x_1, x_2, \cdots, x_d)^\mathrm{T}$, $\mathbf{y} = (y_1, y_2, \cdots, y_c)^\mathrm{T}$라 표기한다.
6. 　x_0과 z_0을 1로 설정한다.

→

3. $\mathbb{X}$에서 임의로 샘플 하나를 뽑는다.
4. 뽑힌 샘플을 $\mathbf{x} = (x_0, x_1, x_2, \cdots, x_d)^\mathrm{T}$, $\mathbf{y} = (y_1, y_2, \cdots, y_c)^\mathrm{T}$라 표기한다.
5. x_0과 z_0을 1로 설정한다.

- 도함수의 종류

Scalar to Scalar	Vector to Scalar	Vector to Vector
$x \in \mathbb{R}, y \in \mathbb{R}$	$x \in \mathbb{R}^N, y \in \mathbb{R}$	$x \in \mathbb{R}^N, y \in \mathbb{R}^M$
Regular derivative:	Derivative is **Gradient**:	Derivative is **Jacobian**:
$\frac{\partial y}{\partial x} \in \mathbb{R}$	$\frac{\partial y}{\partial x} \in \mathbb{R}^N \quad \left(\frac{\partial y}{\partial x}\right)_n = \frac{\partial y}{\partial x_n}$	$\frac{\partial y}{\partial x} \in \mathbb{R}^{N \times M} \quad \left(\frac{\partial y}{\partial x}\right)_{n,m} = \frac{\partial y_m}{\partial x_n}$
If x changes by a small amount, how much will y change?	For each element of x, if it changes by a small amount then how much will y change?	For each element of x, if it changes by a small amount then how much will each element of y change?

- 행렬 표기 : GPU를 사용한 **고속 행렬 연산**에 적합

알고리즘 3-5 다층 퍼셉트론 학습을 위한 스토캐스틱 경사 하강법(행렬 표기)

입력 : 훈련집합 $\mathbb{X}$와 $\mathbb{Y}$, 학습률 ρ
출력 : 가중치 행렬 $\mathbf{U}^1$과 $\mathbf{U}^2$

$\mathbf{U}^1$과 $\mathbf{U}^2$를 초기화한다.
repeat
  $\mathbb{X}$의 순서를 섞는다.
  for ($\mathbb{X}$의 샘플 각각에 대해)
    현재 처리하는 샘플을 $\mathbf{x} = (x_0, x_1, x_2, \cdots, x_d)^\mathrm{T}$, $\mathbf{y} = (y_1, y_2, \cdots, y_c)^\mathrm{T}$라 표기한다.
    x_0과 z_0을 1로 설정한다. // 바이어스
    // 전방 계산
    $\mathbf{zsum} = \mathbf{U}^1\mathbf{x}$, $\tilde{\mathbf{z}} = \boldsymbol{\tau}(\mathbf{zsum})$ // 식 (3.13), $\mathbf{zsum}_{p*1}$, $\mathbf{U}^1{}_{p*(d+1)}$, $\mathbf{x}_{(d+1)*1}$, $\tilde{\mathbf{z}}_{p*1}$
    $\mathbf{osum} = \mathbf{U}^2\mathbf{z}$, $\mathbf{o} = \boldsymbol{\tau}(\mathbf{osum})$ // 식 (3.14), $\mathbf{osum}_{c*1}$, $\mathbf{U}^2{}_{c*(p+1)}$, $\mathbf{z}_{(p+1)*1}$, $\mathbf{o}_{c*1}$
    // 오류 역전파
    $\boldsymbol{\delta} = (\mathbf{y} - \mathbf{o}) \odot \boldsymbol{\tau}'(\mathbf{osum})$ // 식 (3.22), $\boldsymbol{\delta}_{c*1}$
    $\Delta\mathbf{U}^2 = -\boldsymbol{\delta}\mathbf{z}^\mathrm{T}$ // 식 (3.23), $\Delta\mathbf{U}^2{}_{c*(p+1)}$
    $\boldsymbol{\eta} = \left(\boldsymbol{\delta}^\mathrm{T}\widetilde{\mathbf{U}}^2\right)^\mathrm{T} \odot \boldsymbol{\tau}'(\mathbf{zsum})$ // 식 (3.24), $\widetilde{\mathbf{U}}^2{}_{c*p}$, $\boldsymbol{\eta}_{p*1}$
    $\Delta\mathbf{U}^1 = -\boldsymbol{\eta}\mathbf{x}^\mathrm{T}$ // 식 (3.25), $\Delta\mathbf{U}^1{}_{p*(d+1)}$
    // 가중치 갱신
    $\mathbf{U}^2 = \mathbf{U}^2 - \rho\Delta\mathbf{U}^2$ // 식 (3.21)
    $\mathbf{U}^1 = \mathbf{U}^1 - \rho\Delta\mathbf{U}^1$ // 식 (3.21)
until (멈춤 조건)

미니배치 스토캐스틱 경사 하강법

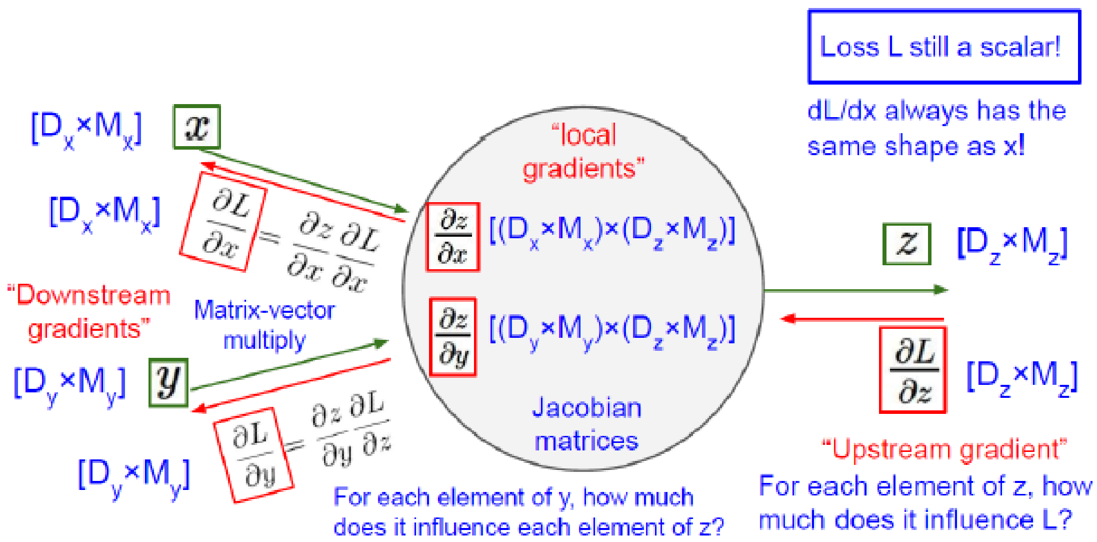
- **미니배치** 방식
  - 한번에 t개의 샘플을 처리함 (t는 미니배치 크기)
    - t = 1이면 스토캐스틱 경사 하강법
    - t = n이면 배치 경사 하강법
  - 미니배치 방식은 보통 t = 수십 ~ 수백
    - 그레이디언트의 잡음을 줄여주는 효과 때문에 **수렴이 빨라짐**
    - GPU를 사용한 **병렬처리에도 유리함**
  - 현대 기계 학습은 **미니배치를 표준**처럼 여겨 널리 사용함

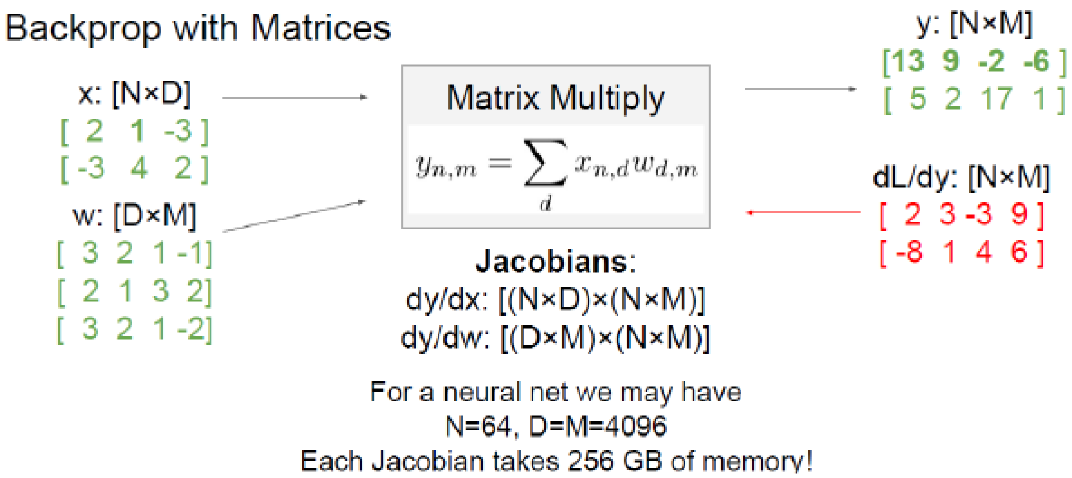
알고리즘3-6 다층 퍼셉트론 학습을 위한 '미니배치' 스토캐스틱 경사 하강법

입력: 훈련집합 $\mathbb{X}$와 $\mathbb{Y}$, 학습률 ρ, 미니배치 크기 t
출력: 가중치 행렬 $\mathbf{U}^1$과 $\mathbf{U}^2$

$\mathbf{U}^1$과 $\mathbf{U}^2$를 초기화한다.
repeat
  $\mathbb{X}$와 $\mathbb{Y}$에서 t개의 샘플을 무작위로 뽑아 미니배치 $\mathbb{X}'$와 $\mathbb{Y}'$를 만든다.
  $\Delta\mathbf{U}^2 = \mathbf{0}$, $\Delta\mathbf{U}^1 = \mathbf{0}$
  for ($\mathbb{X}'$의 샘플 각각에 대해)
    현재 처리하는 샘플을 $\mathbf{x} = (x_0, x_1, x_2, \cdots, x_d)^\mathrm{T}$, $\mathbf{y} = (y_1, y_2, \cdots, y_c)^\mathrm{T}$라 표기한다.
    x_0와 z_0를 1로 설정한다.   // 바이어스
      // 전방 계산
    $\mathbf{zsum} = \mathbf{U}^1\mathbf{x}$, $\tilde{\mathbf{z}} = \boldsymbol{\tau}(\mathbf{zsum})$ // 식 (3.13), $\mathbf{zsum}_{p*1}$, $\mathbf{U}^1{}_{p*(d+1)}$, $\mathbf{x}_{(d+1)*1}$, $\tilde{\mathbf{z}}_{p*1}$
    $\mathbf{osum} = \mathbf{U}^2\mathbf{z}$, $\mathbf{o} = \boldsymbol{\tau}(\mathbf{osum})$ // 식 (3.14), $\mathbf{osum}_{c*1}$, $\mathbf{U}^2{}_{c*(p+1)}$, $\mathbf{z}_{(p+1)*1}$, $\mathbf{o}_{c*1}$
      // 오류 역전파
    $\boldsymbol{\delta} = (\mathbf{y} - \mathbf{o}) \odot \boldsymbol{\tau}'(\mathbf{osum})$ // 식 (3.22), $\boldsymbol{\delta}_{c*1}$
    $\Delta\mathbf{U}^2 = \Delta\mathbf{U}^2 + (-\boldsymbol{\delta}\mathbf{z}^\mathrm{T})$ // 식 (3.23)을 누적, $\Delta\mathbf{U}^2{}_{c*(p+1)}$
    $\boldsymbol{\eta} = (\boldsymbol{\delta}^\mathrm{T}\tilde{\mathbf{U}}^2)^\mathrm{T} \odot \boldsymbol{\tau}'(\mathbf{zsum})$ // 식 (3.24), $\tilde{\mathbf{U}}^2{}_{c*p}$, $\boldsymbol{\eta}_{p*1}$
    $\Delta\mathbf{U}^1 = \Delta\mathbf{U}^1 + (-\boldsymbol{\eta}\mathbf{x}^\mathrm{T})$ // 식 (3.25)를 누적, $\Delta\mathbf{U}^1{}_{p*(d+1)}$
      // 가중치 갱신
  $\mathbf{U}^2 = \mathbf{U}^2 - \rho\left(\frac{1}{t}\right)\Delta\mathbf{U}^2$ // 식 (3.21) - 평균 그레이디언트로 갱신
  $\mathbf{U}^1 = \mathbf{U}^1 - \rho\left(\frac{1}{t}\right)\Delta\mathbf{U}^1$ // 식 (3.21) - 평균 그레이디언트로 갱신
until (멈춤 조건)

- 행렬의 역전파를 위한 도함수





다층 퍼셉트론에 의한 인식

- 예측 단계 (또는 테스트 단계)
    - 학습을 마친 후 현장 설치하여 사용(또는 테스트 집합으로 성능 테스트)

알고리즘3-7 다층 퍼셉트론을 이용한 인식

입력: 테스트 샘플 $\mathbf{x}$ // 신경망의 가중치 $\mathbf{U}^1$과 $\mathbf{U}^2$는 이미 설정되었다고 가정함.
출력: 부류 y

1 $\mathbf{x} = (x_0, x_1, x_2, \cdots, x_d)^T$로 확장하고, x_0과 z_0을 1로 설정한다.
2 $\mathbf{zsum} = \mathbf{U}^1\mathbf{x}$
3 $\tilde{\mathbf{z}} = \tau(\mathbf{zsum})$ // 식 (3.13)
4 $\mathbf{osum} = \mathbf{U}^2\mathbf{z}$
5 $\mathbf{o} = \tau(\mathbf{osum})$ // 식 (3.14)
6 $\mathbf{o}$에서 가장 큰 값을 가지는 노드에 해당하는 부류 번호를 y에 대입한다.

- 라인 6을 수식으로 표현하면 $y = \underset{k}{\operatorname{argmax}}\, o_k$
- 전방 계산 한번만 사용하므로 빠름

다층 퍼셉트론의 특성

오류 역전파 알고리즘의 빠른 속도

- 연산 횟수 비교

표 3-2 전방 계산과 오류 역전파 과정이 사용하는 연산 횟수

과정	수식	덧셈	곱셈
전방 계산	식 (3.13)	dp	dp
	식 (3.14)	pc	pc
오류 역전파	식 (3.22)	c	c
	식 (3.23)		cp
	식 (3.24)	cp	cp
	식 (3.25)		dp
	식 (3.21)	$cp+dp$	$cp+dp$

- 오류 역전파는 전방 계산보다 약 1.5~2배의 시간 소요 ▶ 빠른 계산 가능
- 하지만 학습 알고리즘은 수렴할 때 까지 **오류 역전파를 반복**해야 하므로 점근적 시간복잡도는

    $\Theta((dp + pc)nq)$ (n은 훈련집합의 크기, q는 에포크epoch 수)
- 에포크는 전체 학습 집합을 수행한 단위

모든 함수를 정확하게 근사할 수 있는 능력

- 호닉의 주장[Hornik1989]
    - 은닉층을 하나만 가진 다층 퍼셉트론은 범용근사자

        - **은닉 노드가 충분히 많다면** 활성함수로 무엇을 사용하든 표준 다층 퍼셉트론은 어떤 함수라도 원하는 정확도만큼 **근사화**할 수 있다
    - 은닉 노드를 무수히 많게 할 수 없으므로, 실질적으로는 복잡한 구조의 데이터에서는 성능 한계

성능 향상을 위한 휴리스틱의 중요성

- 순수한 최적화 알고리즘으로 높은 성능 불가능
    - 데이터 희소성, 잡음, 미숙한 신경망 구조 등의 이유
    - **성능 향상**을 위한 갖가지 **휴리스틱**을 개발하고 공유함
- 휴리스틱 개발에서 중요 쟁점
    - 아키텍쳐
        - 은닉층과 은닉노드의 개수를 정해야 한다. 은닉층과 은닉 노드를 놀리면 신경망의 용량은 커지는 대신, 추정할 매개변수가 많아지고 학습 과정에서 과잉적합할 가능성이 커진다.
        - 현대 기계학습은 복잡한 모델을 사용하되, 적절한 규제기법을 적용하는 경향이 있음
    - 초깃값
        - 가중치를 초기화할때 보통 난수를 생성하여 설정하는데, 값의 범위와 분포가 중요
    - 학습률
        - 처음부터 끝까지 같은 학습률을 사용하는 방식과 처음에는 큰 값으로 시작하고 점점 줄이는 적용적 방식이 있음
    - 활성함수
        - 초창기 다층 퍼셉트론은 주로 로지스틱 시그모이드나 tanh 함수를 사용했는데, 은닉층의 개수를 늘림에 따라 그레이디언트 소멸과 같은 몇가지 문제가 발생한다
        - 깊은 신경망은 주로 ReLU 함수를 사용
- 실용적인 성능
    - 1980 ~ 1990년대에 다층 퍼셉트론은 실용 시스템 제작에 크게 기여
        - 인쇄/필기 문자 인식으로 우편물 자동 분류기, 전표 인식기, 자동차 번호판 인식기 등
        - 음성 인식, 게임, 주가 예측, 정보 검색, 의료 진단, 유전자 검색, 반도체 결함 검사 등
- 하지만 한계 노출
    - 잡음이 섞인 상황에서 음성인식 성능 저하
    - 필기 주소 인식 능력 저하
    - 바둑 등의 복잡한 문제에서의 한계
- 이러한 한계를 **딥러닝**은 극복함