START

머신러닝과 딥러닝

Machine Learn[illegible] deep Learning

Chapter 10. 딥러닝 및 다층퍼셉트론

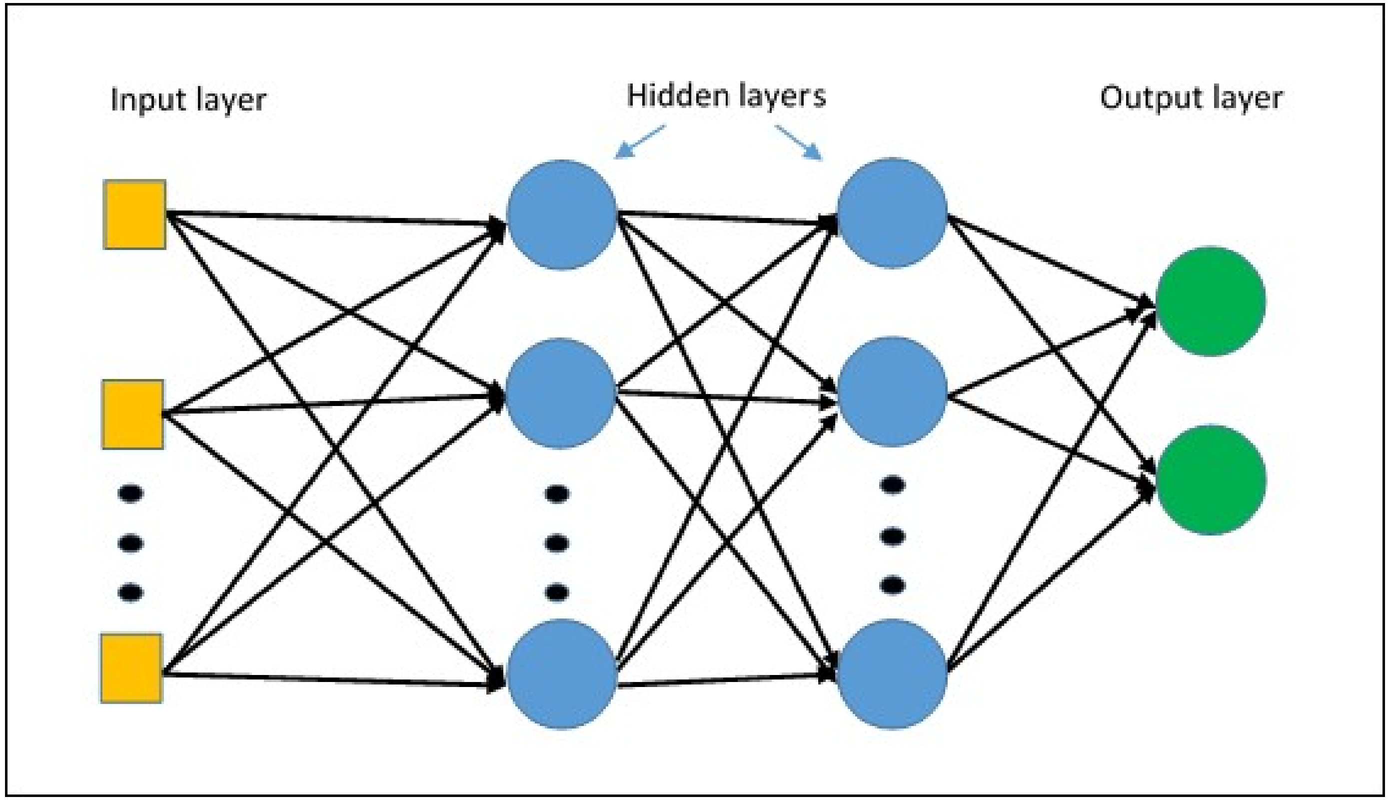
Machine Learning & Deep Learning

손영두

e-mail: youngdoo@dongguk.edu

Deep Learning

- Learning algorithms based on deep neural networks, which are composed of cascades of layers of processing units to extract features from data.



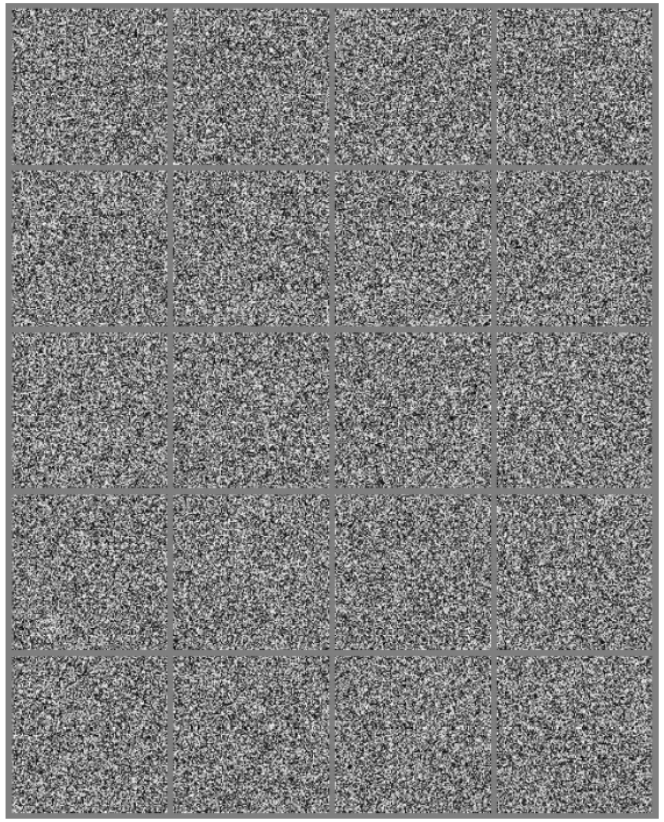
Deep Learning

- 기존의 머신러닝 알고리즘들은 많은 분야에 잘 적용되어 왔으나, 사물 인식, 음성인식 등 “인공지능”스러운 분야의 문제는 잘 해결하지 못함

- 다양체 가정

  - 학습 데이터가 분포하는 정의역에서, 공간의 대부분은 의미를 가지지 않는 입력들이고, 흥미로운 입력들은 일부 점으로 이루어진 몇몇 다양체 상에 존재한다.
  - 우리가 실세계에서 볼 수 있는 이미지, 텍스트, 소리 등은 현실에 존재하는 쪽에 확률 분포가 집중되어 있음
  - 이러한 다양체들은 하나 또는 여러 개가 존재하며, 다양체 내부에서의 움직임 또는 다양체 사이의 이동이 의미를 가진다

Deep Learning



다양체 가정

- 의미 없는 무작위 이미지
- 의미 없는 알파벳의 나열이 의미있는 단어가 될 확률은 매우 낮음

Figure 5.12: Sampling images uniformly at random (by randomly picking each pixel according to a uniform distribution) gives rise to noisy images. Although there is a non-zero probability to generate an image of a face or any other object frequently encountered in AI applications, we never actually observe this happening in practice. This suggests that the images encountered in AI applications occupy a negligible proportion of the volume of image space.

Deep Learning

다양체 가정

- 다양체 내부에서의 움직임?

Figure 5.13: Training examples from the QMUL Multiview Face Dataset (Gong et al., 2000) for which the subjects were asked to move in such a way as to cover the two-dimensional manifold corresponding to two angles of rotation. We would like learning algorithms to be able to discover and disentangle such manifold coordinates. Figure 20.6 illustrates such a feat.

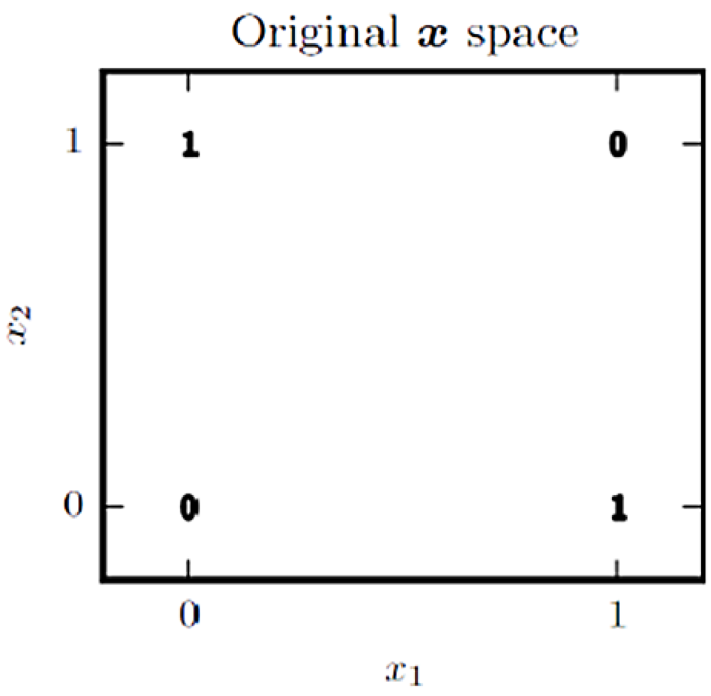
- 사람의 얼굴과 고양이의 얼굴은 서로 연결되지 않은 다양체로 설명이 가능

XOR 문제

XOR 문제

x_1	x_2
0	0
0	1
1	0
1	1

y
0
1
1
0



- XOR을 분류하는 간단한 문제
- 그러나 선형으로는 분리가 불가능
- 다음과 같은 선형 분류를 이용하여 문제를 해결해보자

$$f(\boldsymbol{x}; \boldsymbol{w}, b) = \boldsymbol{x}^\top \boldsymbol{w} + b$$

XOR 문제

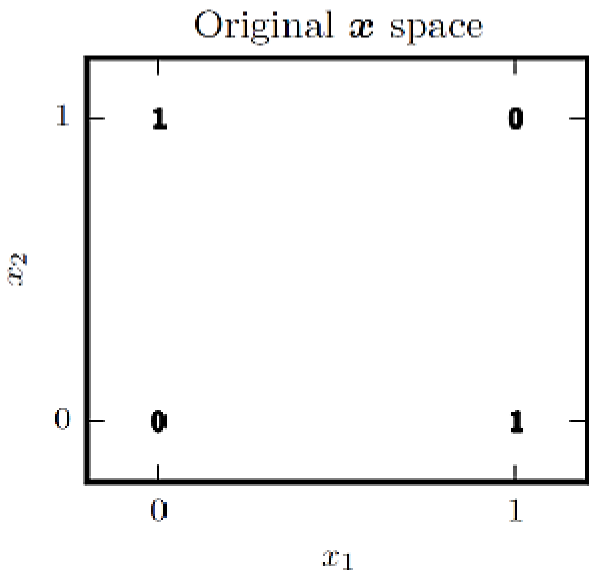
XOR 문제

- 제곱평균 비용함수를 가정하면,

$$J(\boldsymbol{\theta}) = \frac{1}{4} \sum_{\boldsymbol{x} \in \mathbb{X}} (f^*(\boldsymbol{x}) - f(\boldsymbol{x}; \boldsymbol{\theta}))^2 .$$

- w=0, b=0.5 를 얻을 수 있다

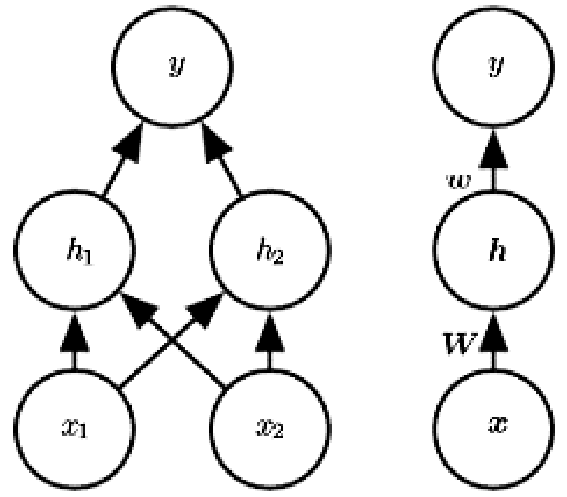
- Decision boundary?



XOR 문제

XOR 문제

- 은닉층이 한 층인 간단한 신경망 모형을 가정하여 보자



- 위의 신경망이 다음과 같은 연산을 수행한다고 가정

$$f^{(1)}(\boldsymbol{x}) = \mathbf{W}^T \boldsymbol{x} + \boldsymbol{c}$$

$$f^{(2)}(\boldsymbol{h}; \mathbf{w}, b) = \boldsymbol{w}^T \boldsymbol{h} + b$$

- XOR에 대한 분류가 가능할까?

XOR 문제

XOR 문제

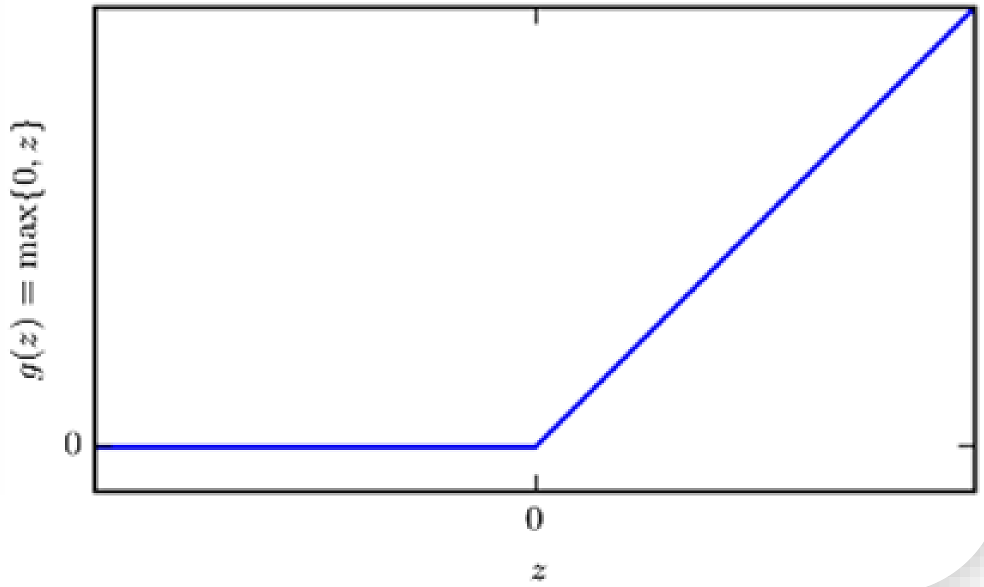
- XOR 문제를 분류하기 위해서는 비선형적인 변환이 반드시 필요
- 예시: Rectified Linear Unit (ReLU)

$$h_i = g(\boldsymbol{x}^\top \boldsymbol{W}_{:,i} + c_i)$$

$$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^\top \max\{0, \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}\} + b.$$

$$\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad \boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix},$$

$$\boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \quad b = 0.$$

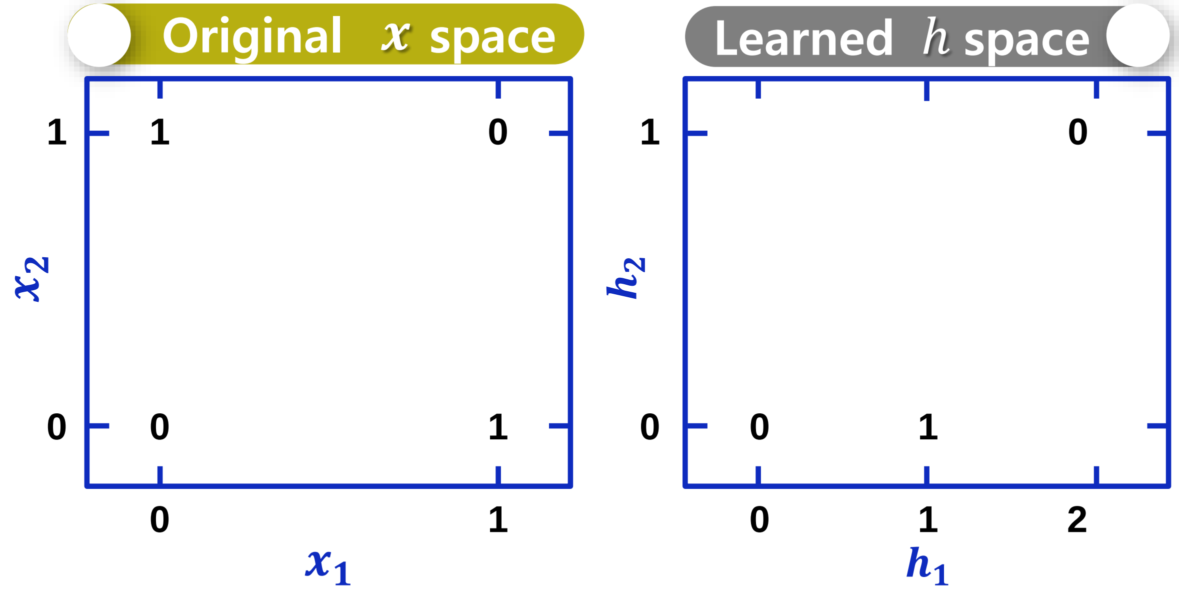


XOR 문제

XOR 문제: Representation Learning

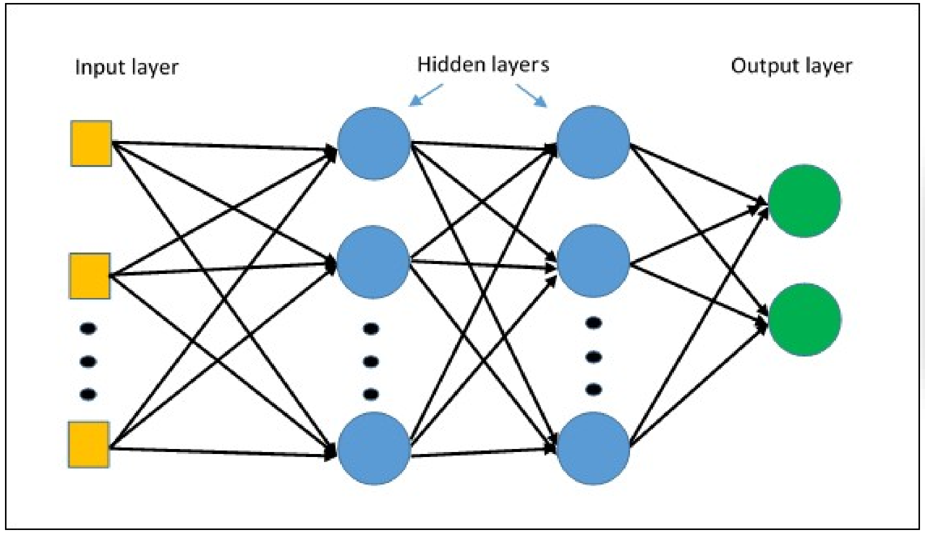
$$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^\top \max\{0, \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}\} + b.$$

- 신경망의 첫 번째 층에서는 아래 그림과 같이 표현의 변환을 학습하고, 두 번째 층에서는 선형 분류를 수행



다층 퍼셉트론 (Multilayer Perceptron, MLP)

- 심층 순방향 신경망(deep feedforward network)라고도 부름
- 입력층 (input layer), 출력층 (output layer), 그리고 그 사이의 다수의 은닉층 (hidden layers)으로 이루어짐
- **Feedforward :** 정보는 input에서 ouptut으로 한 방향으로만 흘러가며 feedback, recurrent 구조 등이 존재하지 않음
- 각 층은 여러 개의 node로 이루어져 있으며, 일반적으로 하나의 층은 이전 층으로부터 정보를 받아 다음 층으로 정보를 보내는 역할을 수행
- 신경망의 층의 수를 깊이 (depth)로 표현하며, 각 층에 존재하는 node의 수를 넓이(width)로 표현



$$f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$$

신경망의 학습

- **신경망의 학습**은 신경망의 층들 사이를 연결하는 **매개변수(가중치)들의 값**을 결정하는 것
- **신경망의 학습**은 일반적으로 **기울기 하강법 (gradient descent)**를 통하여 이루어짐
- **Gradient descent** 방법을 적용하기 위해서는 비용함수를 이러한 매개변수들과, **node**들로 표현할 필요가 있음
- **비선형적인 활성함수**를 사용하는 경우, **비용함수가 convex하지 않아** 기울기 하강법을 통하여 광역 **최적해(global optimum)**에 도달하지 못할 가능성이 존재
  - 많은 국소 최적해 (local optimum)이 존재하며, 이들 중 하나로 수렴함
  - 광역 최적해로의 수렴은 보장되지 않음

신경망의 학습

Gradient Descent Algorithm

- 다음과 같은 training data가 주어졌을 때, $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$

  $(x_{i1}, \ldots, x_{id}) \in \mathbb{R}^d$

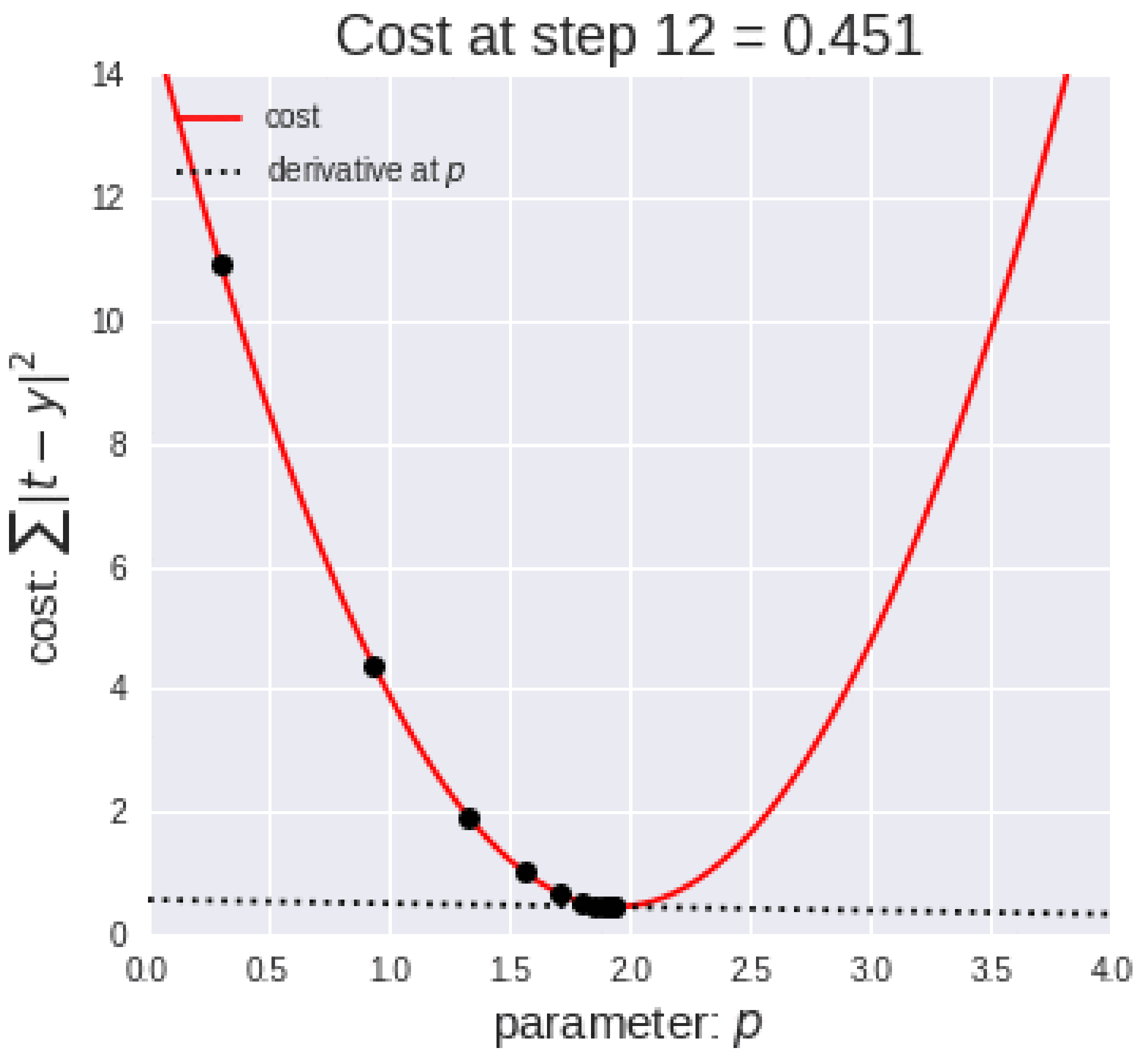
- Model: $\hat{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$

- Cost function : $J(\boldsymbol{\theta}) = \frac{1}{n} \sum_{(x_i, y_i) \in D} L(y_i, \hat{y}_i)$

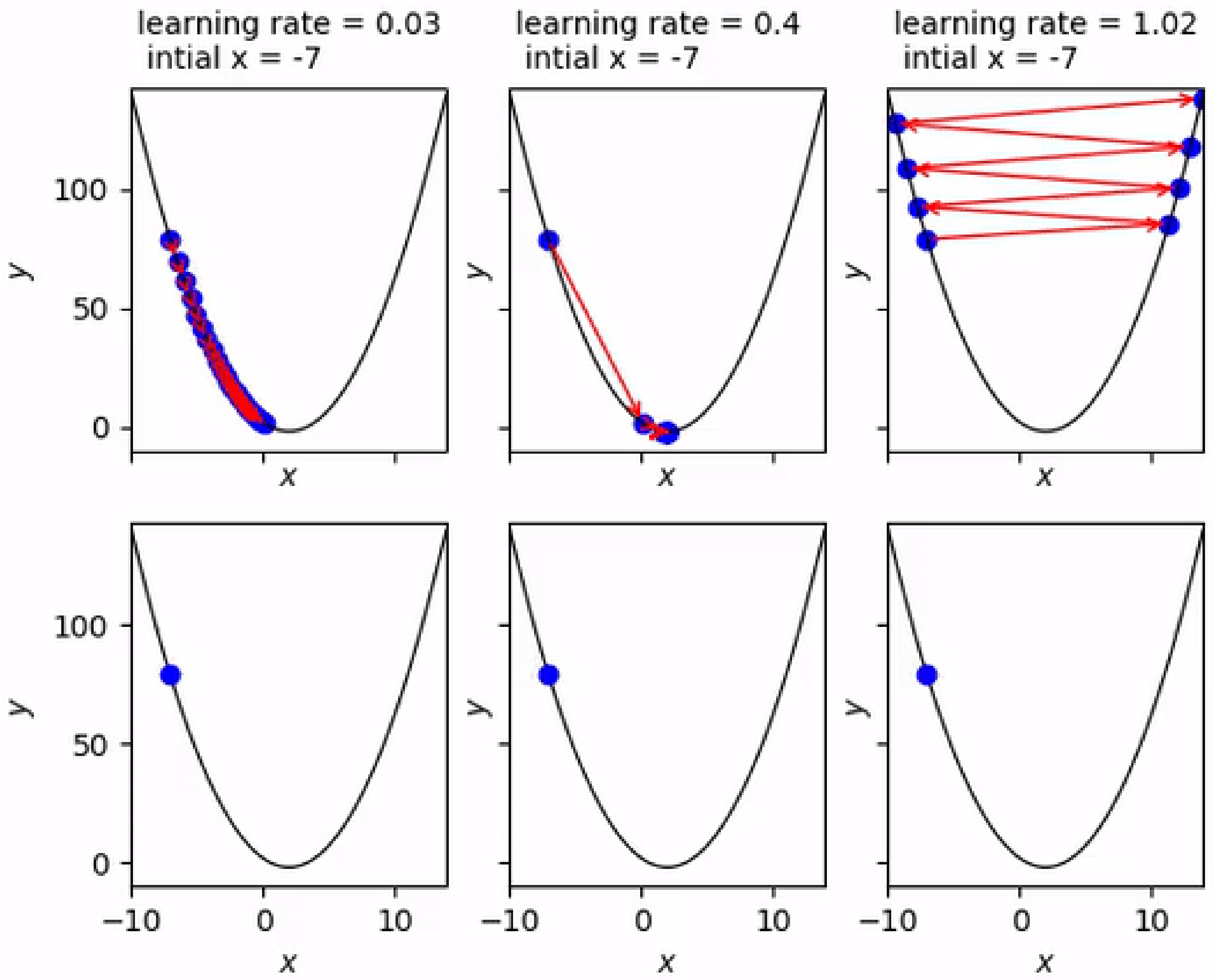
- Gradient descent 방법: $\boldsymbol{\theta} \coloneqq \boldsymbol{\theta} - \epsilon \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

  $\rightarrow \theta_j \coloneqq \theta_j - \epsilon \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}), \forall \theta_j \in \boldsymbol{\theta}$

신경망의 학습


Cost at step 12 = 0.451
cost
derivative at p
cost: $\sum |t - y|^2$
parameter: p
14
12
10
8
6
4
2
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

신경망의 학습: learning rate?



신경망의 학습

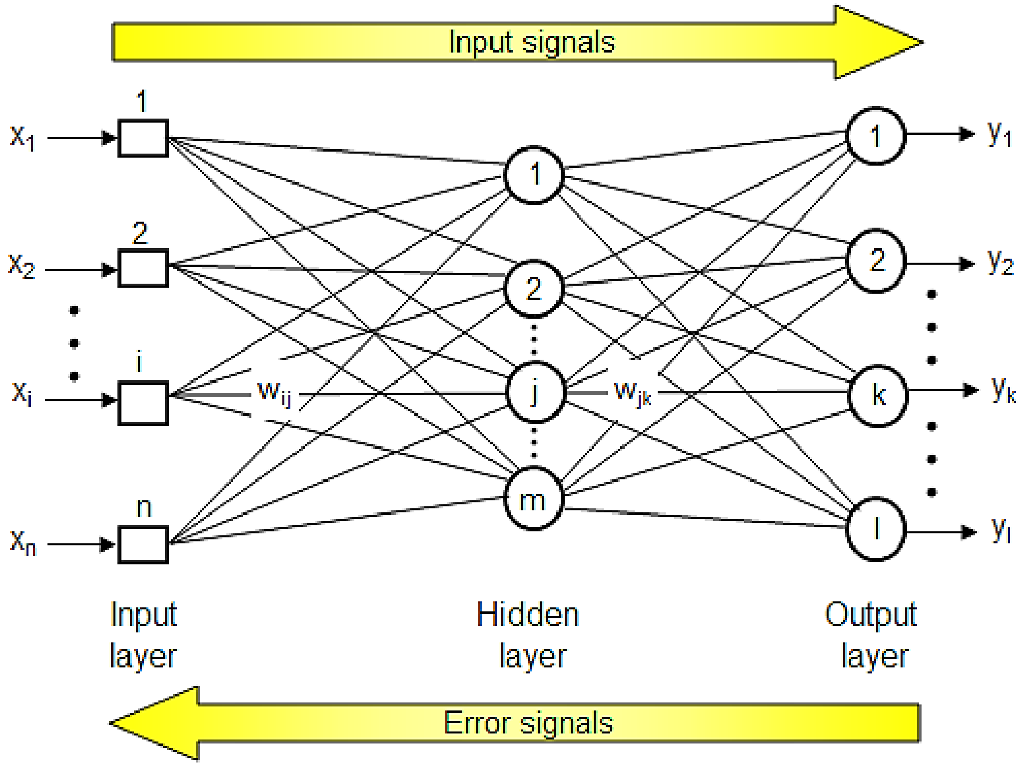
Gradient Descent Algorithm

- **Forward propagation:**

입력 x로부터 정보를 순방향으로 진행하여 예측값 $\hat{y}$를 생성하여 cost function $J(\theta)$를 계산

- **Backpropagation:**

계산된 cost function $J(\theta)$를 이용하여 역방향으로 정보를 전파하여 gradient descent를 위하여 gradient들을 계산하는 단계



신경망의 학습

Recall: Chain Rule of Calculus

- 미분의 연쇄 법칙: 도함수가 알려진 여러 함수를 결합해서 만든 함수의 미분을 계산하는 데 쓰임.

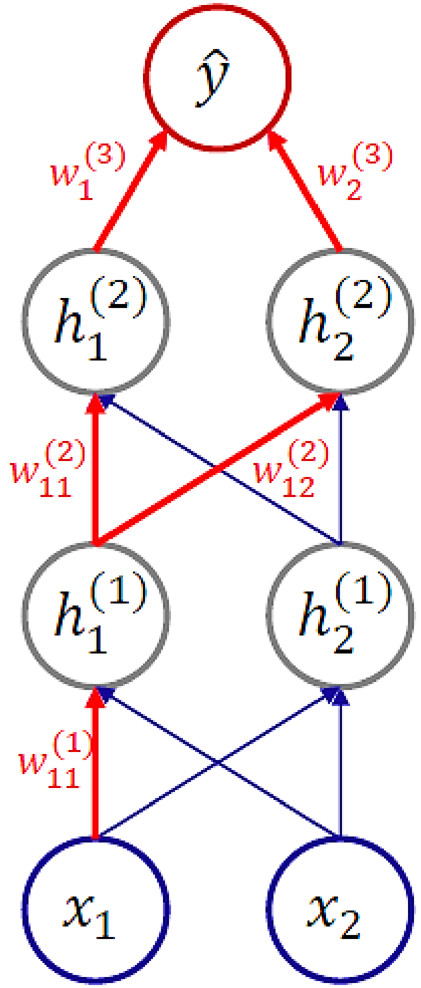
- 단일 변수 함수의 연쇄법칙 $$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

- 다변수 함수의 연쇄법칙 $$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i}.$$

- 벡터 형태 표현 $$\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^\top \nabla_{\boldsymbol{y}} z,$$

신경망의 학습

Backpropagation and Chain Rule



신경망의 학습

Backpropagation in general (optional)

- **Assume** $$a_j = \sum_i w_{ji} z_i \text{ and } z_j = h(a_j)$$

- **We get** $$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \cdot \frac{\partial a_j}{\partial w_{ji}}$$

- **To simplify,** $$\delta_j = \frac{\partial E_n}{\partial a_j}$$

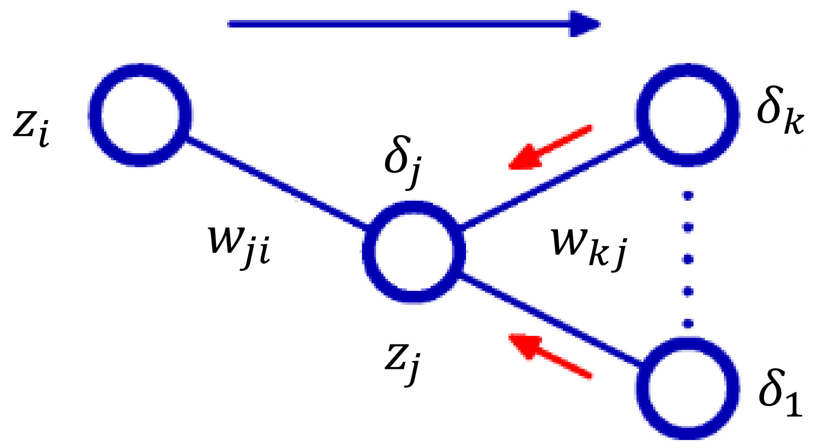
- **Then,** $$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$$

신경망의 학습

Backpropagation in general (optional)

- By using chain rule

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$



신경망의 디자인: Cost Functions

- 비용 함수(cost function)는 어떻게 선택해야할까?
- 확률적인 관점에서 신경망은 입력값과 출력값의 관계를 나타내는 다음의 조건부확류을 결정함

$$p(y|\boldsymbol{x};\boldsymbol{\theta})$$

- 따라서 우리는 훌륭한 모수를 학습하기 위하여 위의 조건부 확률에 대한 최우도 추정을 수행 가능

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{(x_i,y_i)\in D} \log p(y_i|\boldsymbol{x}_i;\boldsymbol{\theta})$$

- 이를 비용 함수의 형태로 표현하면 (negative log-likelihood, NLL),

$$J(\boldsymbol{\theta}) = \sum_{(x_i,y_i)\in D} -\log p(y_i|\boldsymbol{x}_i;\boldsymbol{\theta})$$

신경망의 디자인: Cost Functions

예시: 가우시안 분포

$$p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}; f(\boldsymbol{x};\boldsymbol{\theta}), \boldsymbol{I}),$$

$$J(\theta) = \frac{1}{2}\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{\text{data}}} ||\boldsymbol{y} - f(\boldsymbol{x};\boldsymbol{\theta})||^2 + \text{const}$$

예시

$$f^* = \arg\min_f \mathbb{E}_{\mathbf{x},\mathbf{y}\sim p_{\text{data}}} ||\boldsymbol{y} - f(\boldsymbol{x})||_1$$

신경망의 디자인: Output Units

- 출력단위(output unit)의 선택은 비용함수와 밀접하게 연관되어 있음
- 이러한 출력단위의 경우 수행하고자 하는 작업(task)에 맞추어 선택하는 것이 일반적임
- 선형 유닛: 회귀분석의 경우

$$\hat{y} = \mathbf{w}^{\mathsf{T}}\mathbf{h} + b$$

  - 때때로 정규 분포의 평균을 예측하기 위해 사용

$$p(y|\boldsymbol{x};\boldsymbol{\theta}) = \mathcal{N}(y|\hat{y},\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{\|y-\hat{y}\|^2}{2\sigma^2}\right)$$

  - Cost function (negative log-likelihood?)

신경망의 디자인: Output Units

시그모이드 유닛: 이원 분류의 경우

$$\hat{y} = P(y = 1|\boldsymbol{x}; \boldsymbol{\theta}) = \sigma(z) = \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{h} + b)$$

- Sigmoid 함수는 0과 1 사이의 값을 출력
- 베르누이 분포의 확률을 표현하기 위하여 사용

$$p(y|\boldsymbol{x}; \boldsymbol{\theta}) = Bernoulli(y|\hat{y}) = (\hat{y})^{y}(1 - \hat{y})^{1-y}$$

- Cost function (negative log-likelihood?)

소프트맥스 유닛: 다원 분류의 경우

$$\hat{\boldsymbol{y}} = (\hat{y}_1, \ldots, \hat{y}_k),$$

- 출력 벡터의 값은 각 클래스에 속할 확률을 의미

$$\hat{y}_i = p(y = i|\boldsymbol{x}; \boldsymbol{\theta})$$

- Multinulli distribution

$$p(y|\boldsymbol{x}; \boldsymbol{\theta}) = \prod_{i=1}^{k} \hat{y}_i^{\,\mathrm{I}(y=i)}$$

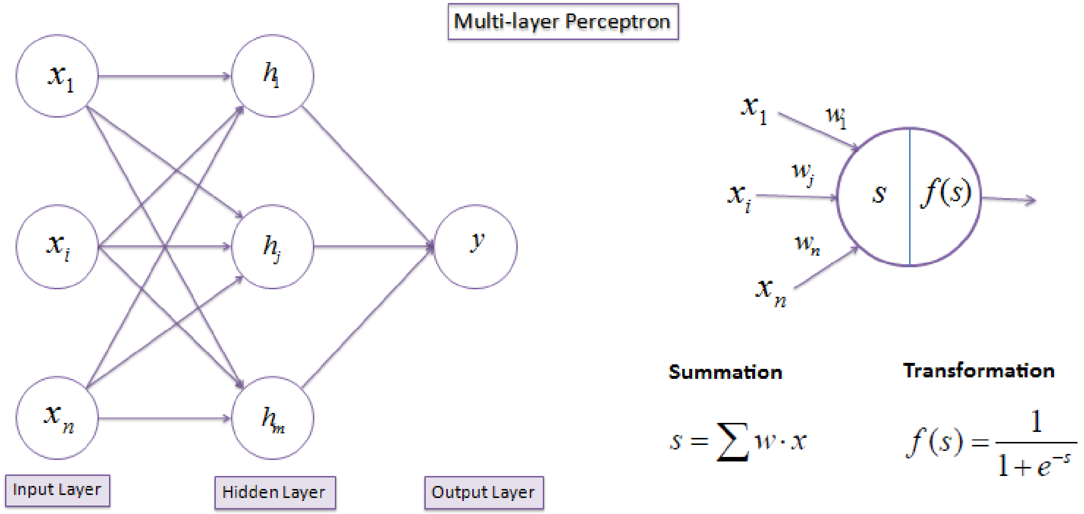
신경망의 디자인: Hidden Units

은닉층은 이전 층으로부터 값을 전달받아 (비선형) 변환을 수행하고 다음 층으로 출력 값을 전달

- 비선형 활성 함수를 사용
- 은닉층이 많을수록 더욱 복잡한 모델

은닉층의 작동 원리

- 은닉층의 각 노드는 이전 층의 노드들로부터 값을 입력받아 가중합을 계산
- 이러한 가중합을 비선형 활성함수에 통과시켜 출력을 만든 후 다음 층의 노드들에 전달

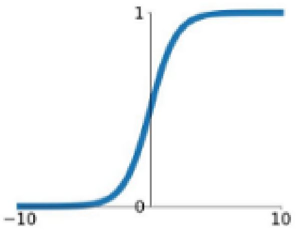


신경망의 디자인: Activation Functions

Types of activation functions

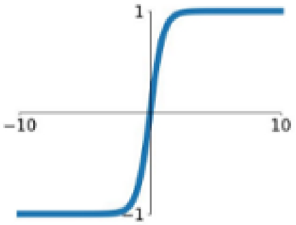
Sigmoid

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



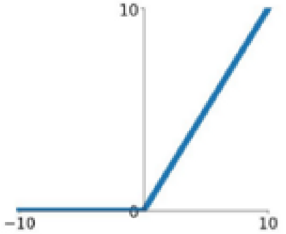
tanh

$$\tanh(x)$$



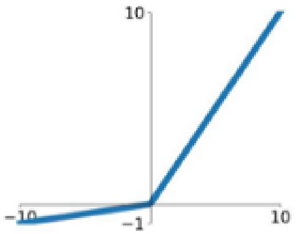
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

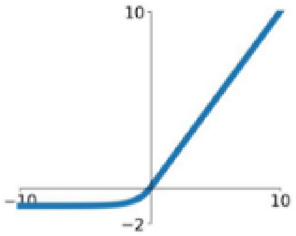


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

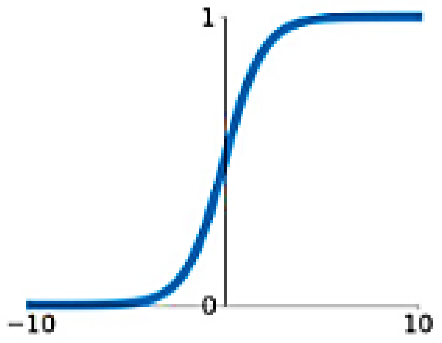


신경망의 디자인: Activation Functions

Sigmoid Activation Function

Sigmoid

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



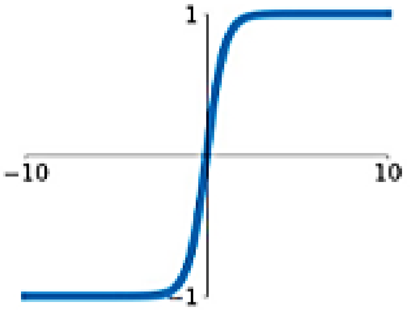
- 전통적으로 널리 사용되던 활성함수
- 0과 1 사이의 값으로 각 노드가 활성화 된 정도를 표현하기에 적합
- Step function의 근사형으로 많이 사용됨 (사람의 신경 neuron의 작동 원리)
- 문제점:
  1. 포화된 노드는 gradient값을 사라지게 만듦
  2. zero-centered 되지 않음
  3. exponential 함수의 계산이 필요

신경망의 디자인: Activation Functions

Hyperbolic Tangent Activation Function

tanh

$$\tanh(x)$$



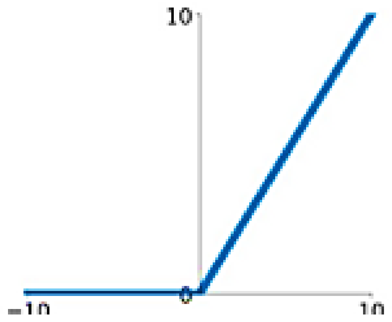
- Sigmoid와 마찬가지로 전통적으로 널리 사용되던 활성함수
- -1과 1 사이의 값을 출력
- Zero-centered 되었으나, 여전히 포화상태에서는 gradient를 사라지게 만듦

신경망의 디자인: Activation Functions

Rectified Linear Unit (ReLU) Activation Function

ReLU

$$\max(0, x)$$



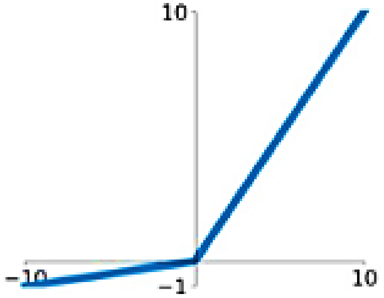
- 현대 신경망의 표준 활성함수
- 입력이 양수인 부분에서 gradient가 사라지지 않음
- 계산이 빠르고 효과적임
- Sigmoid/tanh와 비교하여 수렴이 빠름
- 그러나 zero-centered 되지 않고 입력이 0보다 작은 경우 gradient가 0이됨

신경망의 디자인: Activation Functions

Leaky ReLU Activation Function

Leaky ReLU

$$\max(0.1x, x)$$



- 어떤 부분에서도 gradient가 사라지지 않음
- 계산이 빠르고 효과적임
- Sigmoid/tanh와 비교하여 수렴이 빠름
- 음수 쪽에 곱해지는 계수를 학습으로 결정하는 방법도 존재: Parametric ReLU (PReLU)

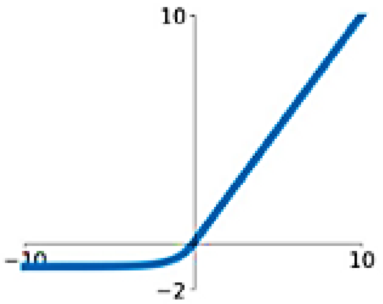
$$f(x) = \max(\alpha x, x)$$

신경망의 디자인: Activation Functions

Exponential ReLU Activation Function

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



- ReLU의 장점을 대부분 가지고 있음
- 큰 음수 부분에 대하여 영향을 덜 받아 (상대적으로) 노이즈에 강인함

신경망의 디자인: Activation Functions

Leaky ReLU Activation Function

Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

- ReLU와 Leaky ReLU의 일반화
- 두 선형 식으로 구성되기 때문에, gradient가 모든 곳에서 사라지지 않음
- 그러나 학습해야 하는 모수의 수가 증가

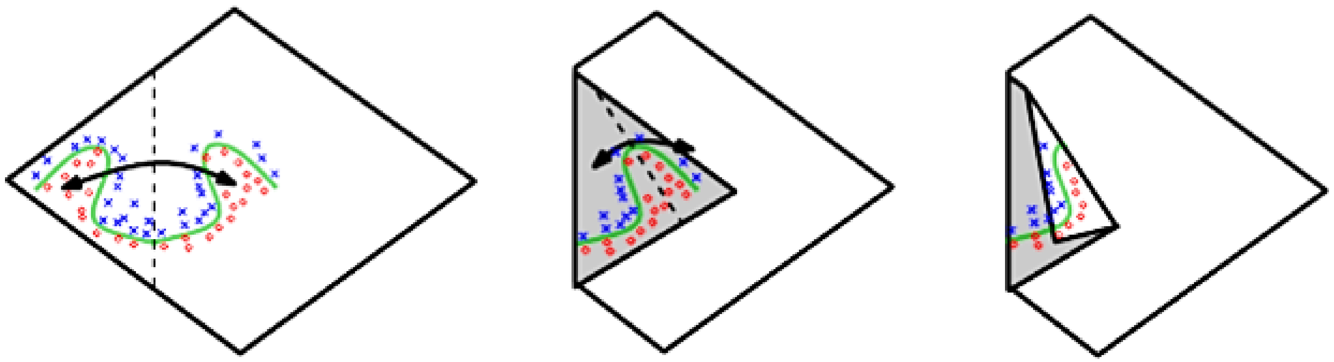
신경망의 디자인: Activation Functions

- A feedforward network with a linear output layer and at least one hidden layers with any “squashing” activation function can approximate any Borel measurable function from one finite-dimensional space to another with any desired non-zero amount of error, provided that the network is given enough hidden units.
  - The reason why we used sigmoid active function in the past
  - 모든 Borel measurable function을 근사 가능하지만, 원하는 함수를 근사하도록 학습이 보장된다는 이야기는 아님
- Then, why deeper?

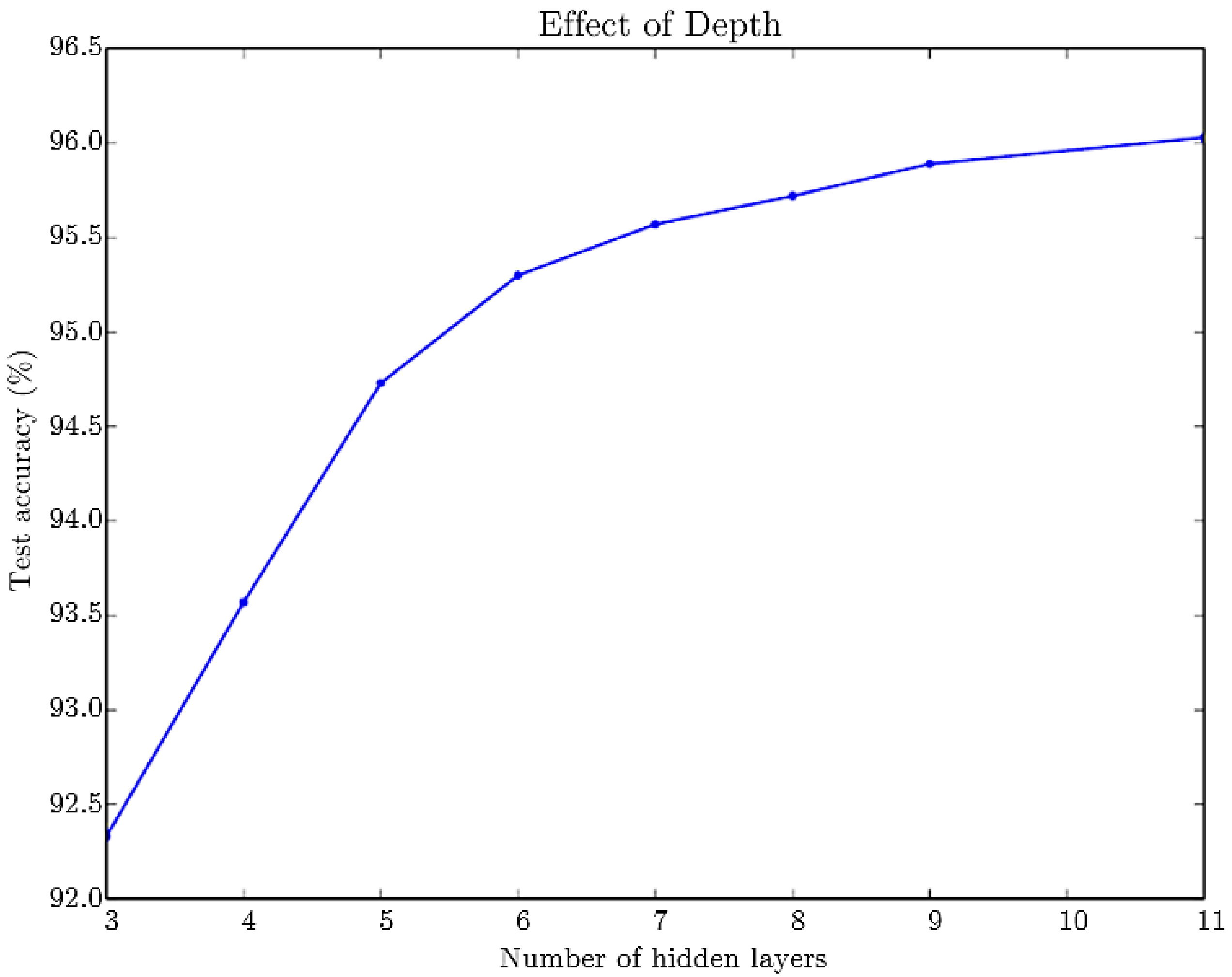
신경망의 디자인: Deeper Networks

- Universal Approximation Theorem에 의하여 hidden layer가 하나인 MLP 또한 임의의 함수를 표현할 수 있지만, width가 감당할 수 없을 정도로 커지거나, 이에 따른 학습에 문제가 발생할 수 있음

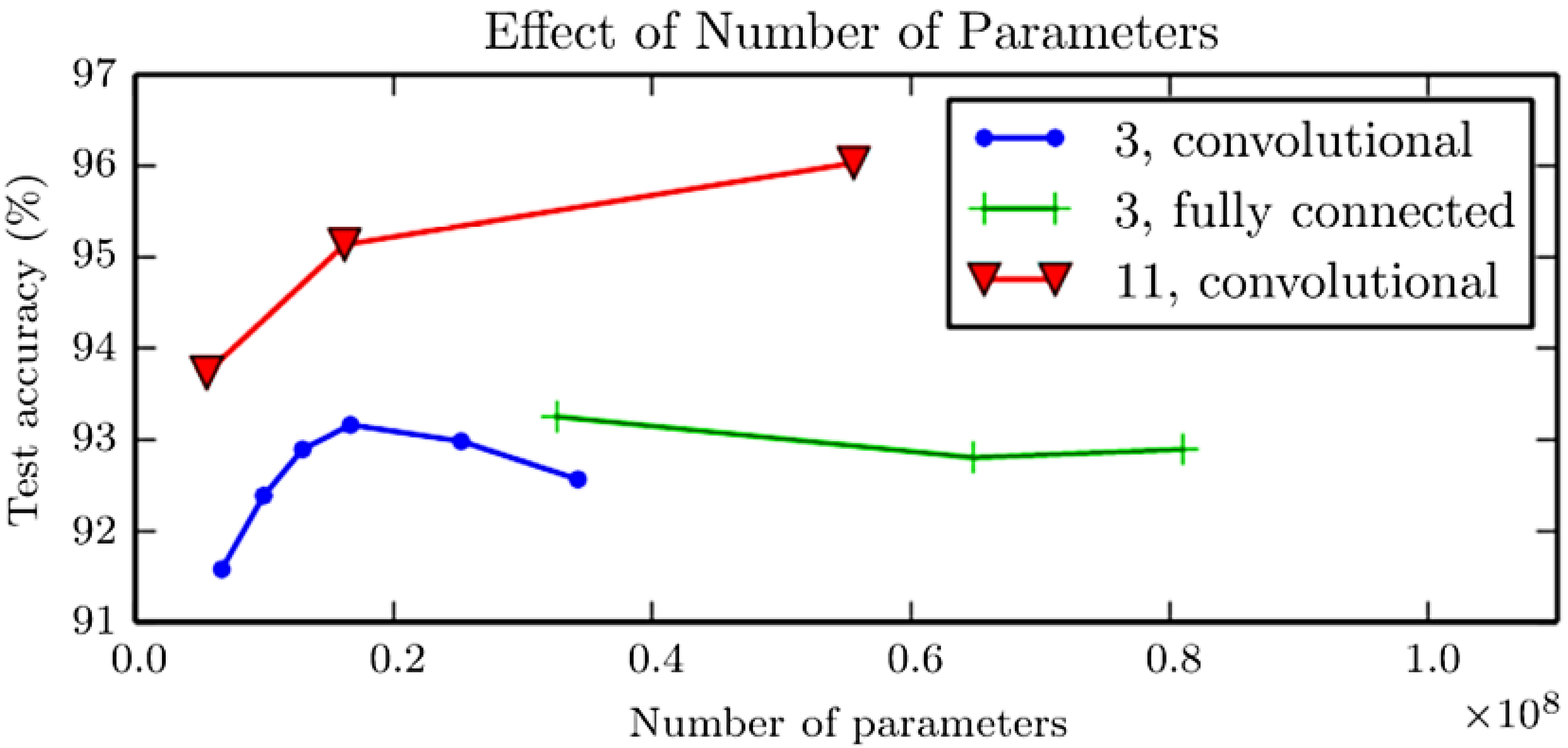
- Depth를 통하여 해결



신경망의 디자인: Deeper Networks



신경망의 디자인: Deeper Networks



실습 – Google Colab 사용하기

구글 코랩이란?

- 파이썬과 다양한 라이브러리들이 설치되어 있는 구글의 서버를 이용하는 것으로써, 사용자는 코드를 입력하기만 하면 구글 서버에서 해당 코드가 실행되어 결과를 보여줌.

- GPU를 사용할 수 있어 빠른 학습이 가능함.

- Jupyter notebook과 비슷한 환경을 제공하며, 사용자의 개별 컴퓨터가 아닌 구글 서버에서 돌아가는 jupyter notebook으로 볼 수 있음.

- 구글 드라이브에 csv 등의 데이터 파일을 업로드한 후, 구글 코랩과 연결하여 코드를 입력 및 실행하면 됨.

실습 – Google Colab 사용하기

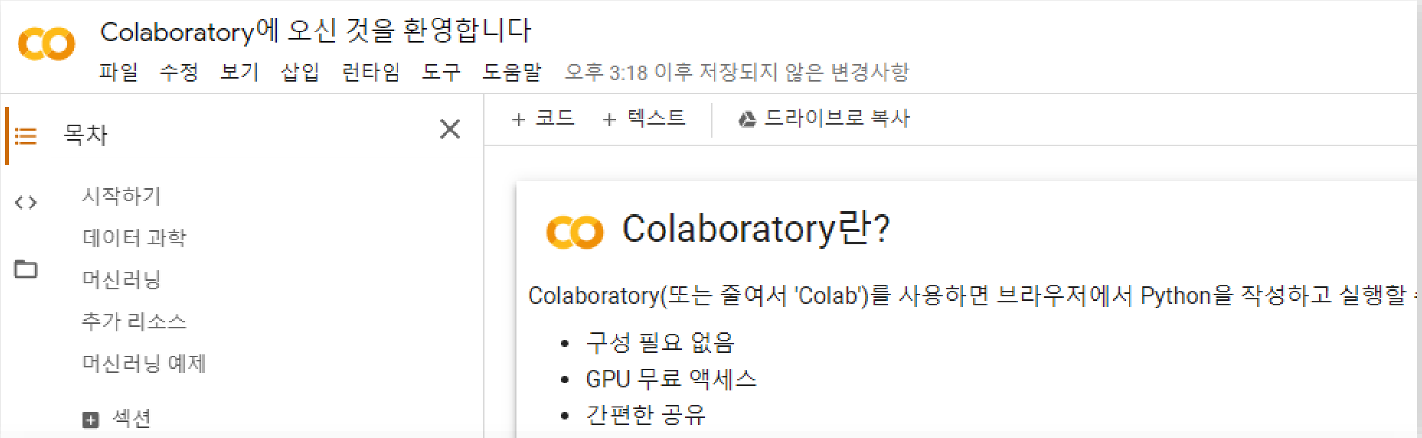
구글 코랩 사용 방법 -1

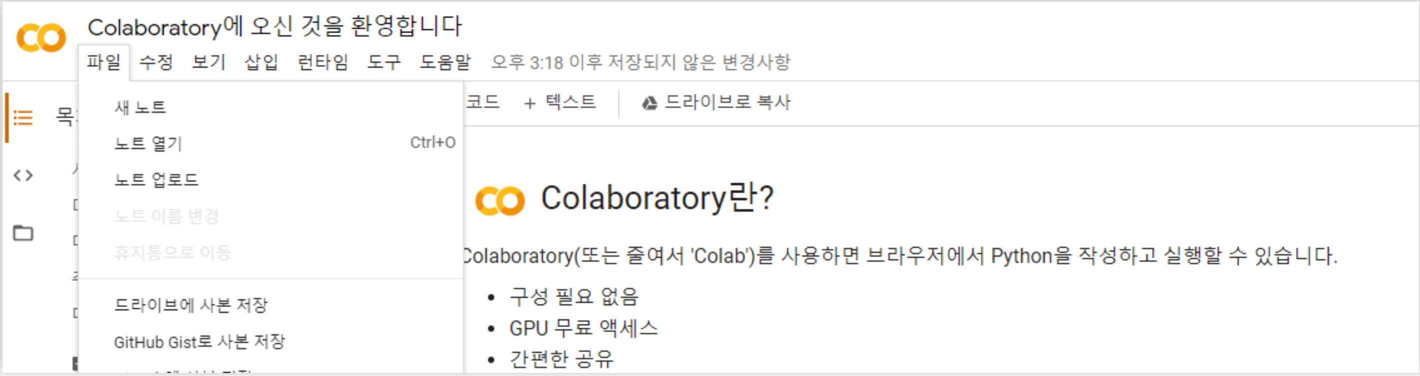
- 구글 계정에 로그인한 후, 구글 드라이브에 사용자가 분석하고자 하는 데이터 파일을 업로드. 이미지 파일도 가능함.
- 그 다음, 코랩 사이트 접속 (https://colab.research.google.com/notebooks/welcome.ipynb)

실습 – Google Colab 사용하기

구글 코랩 사용 방법 -2

- 왼쪽 상단의 "파일"→ "새 노트" 버튼을 클릭하여 새로운 노트 파일을 생성함.

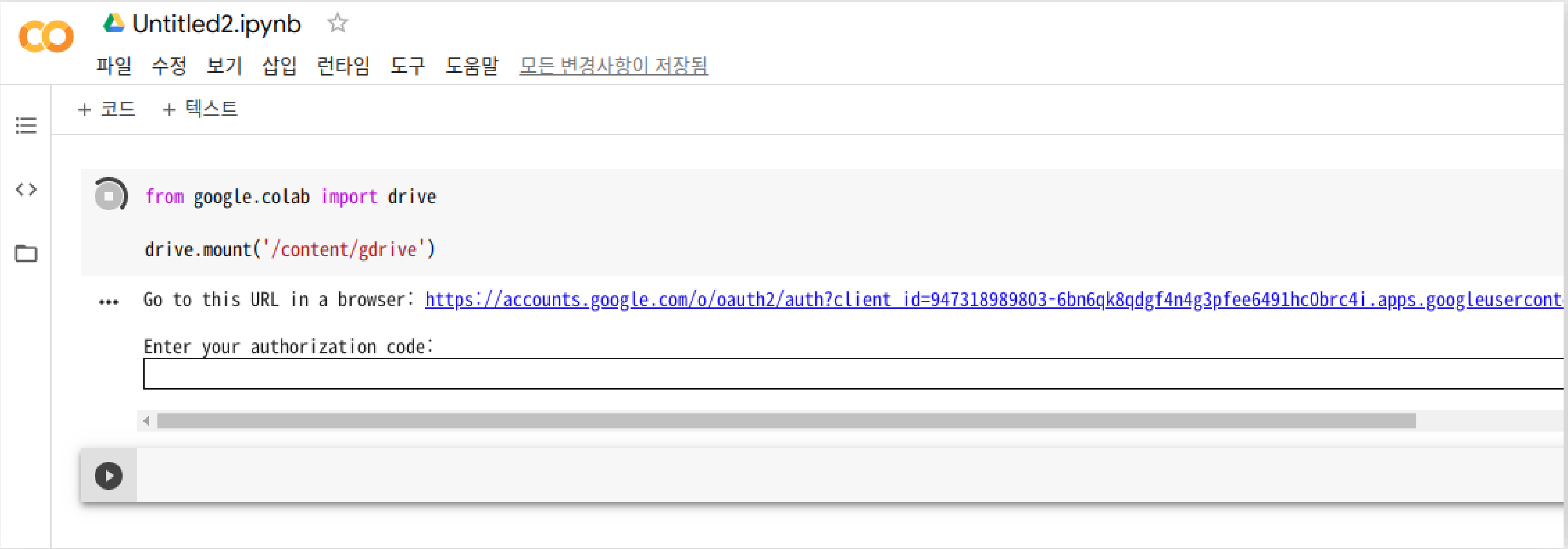




실습 – Google Colab 사용하기

구글 코랩 사용 방법 -3

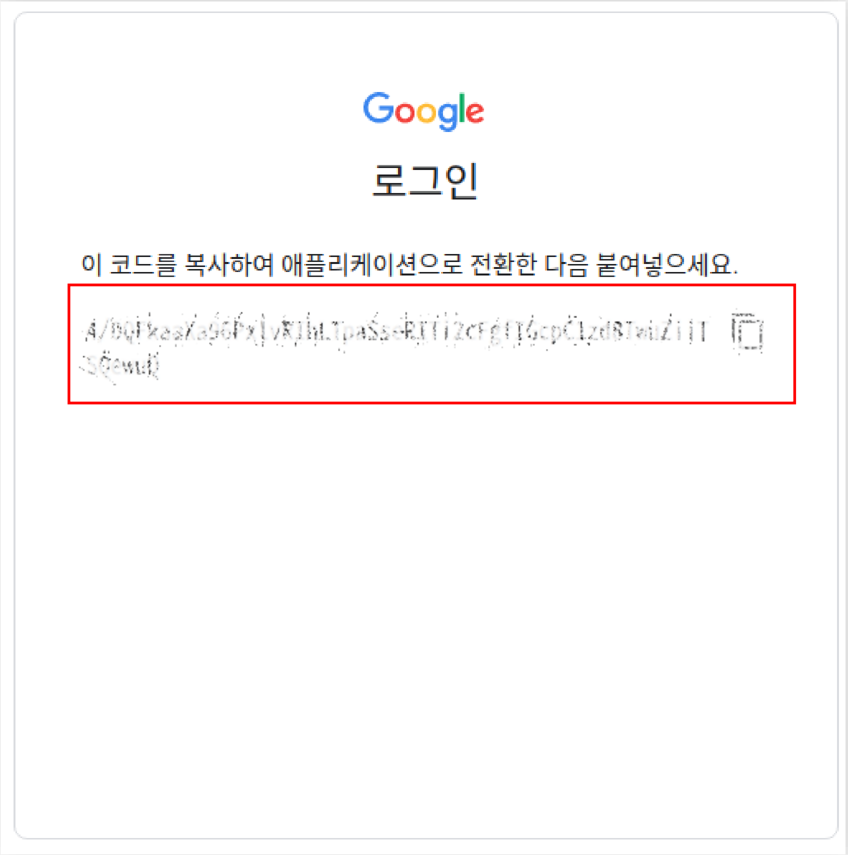
- 새로운 노트가 생성되면 다음의 그림과 같이 코드를 실행한 후, 아래에 출력되는 브라우저를 클릭하여 인증 창 진입.



실습 – Google Colab 사용하기

구글 코랩 사용 방법 -4

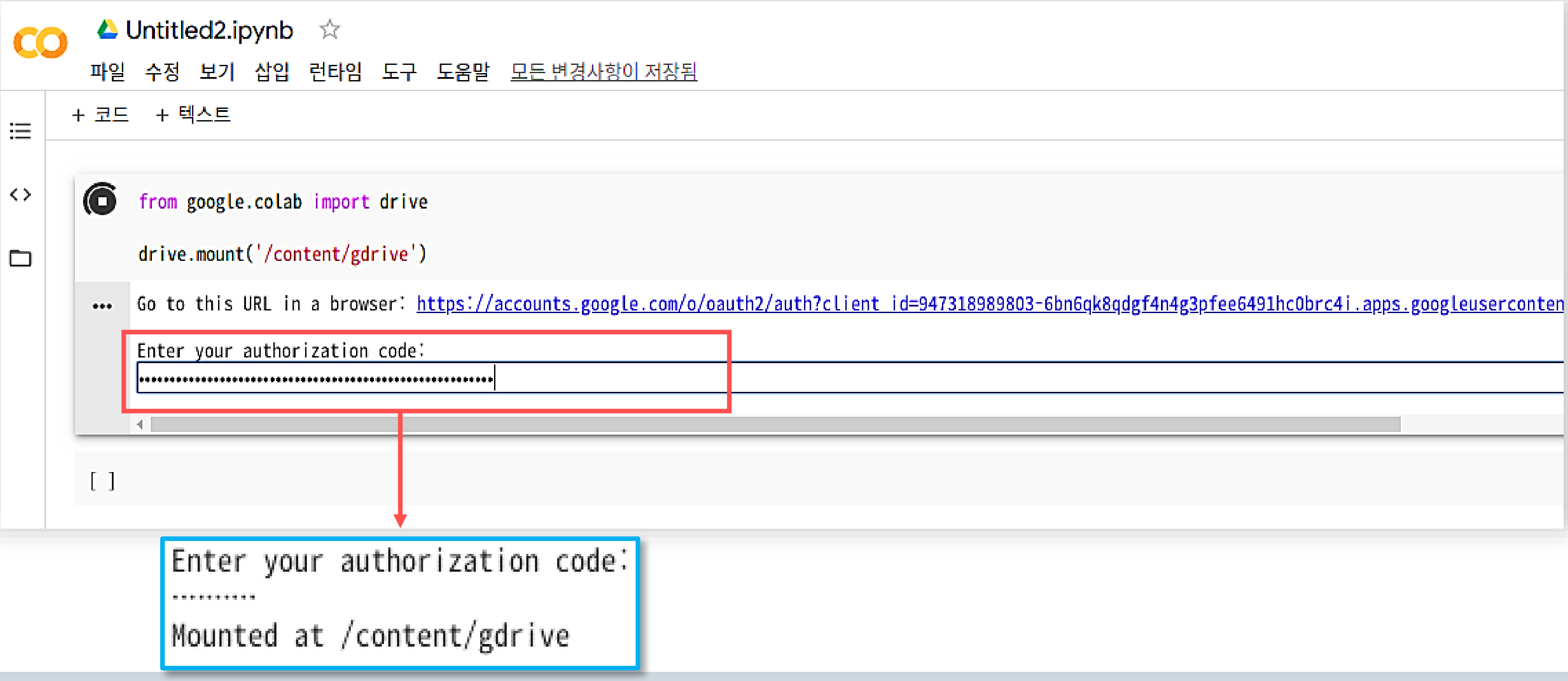
- 구글 계정을 선택하고 액세스를 허용하면, 다음과 같이 개인 인증코드가 생성됨.



실습 – Google Colab 사용하기

구글 코랩 사용 방법 -5

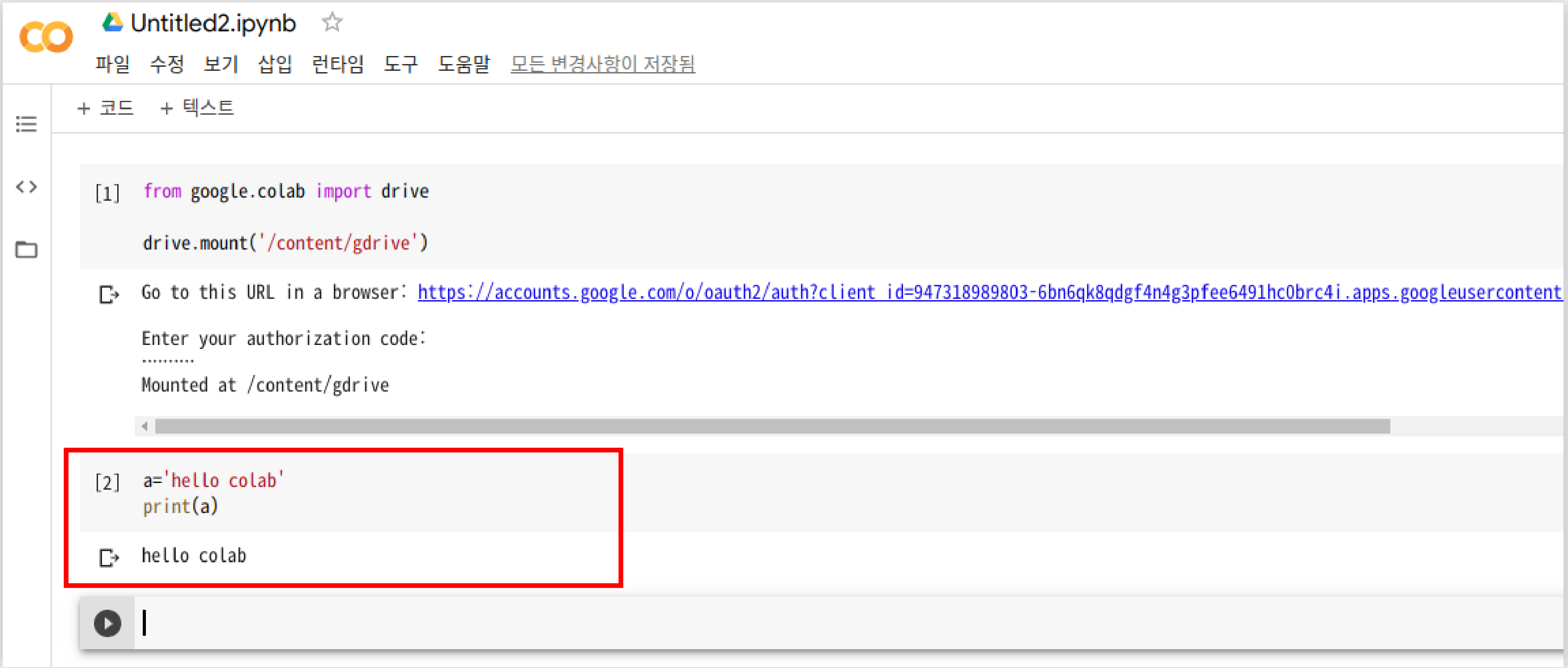
- 인증코드를 복사하여 **코랩 파일**에 붙여넣은 후 **Enter** 클릭.
- **Mounted at /content/gdrive** 라는 문구가 출력되면 드라이브와 연결 완료.



실습 – Google Colab 사용하기

구글 코랩 사용 방법 -6

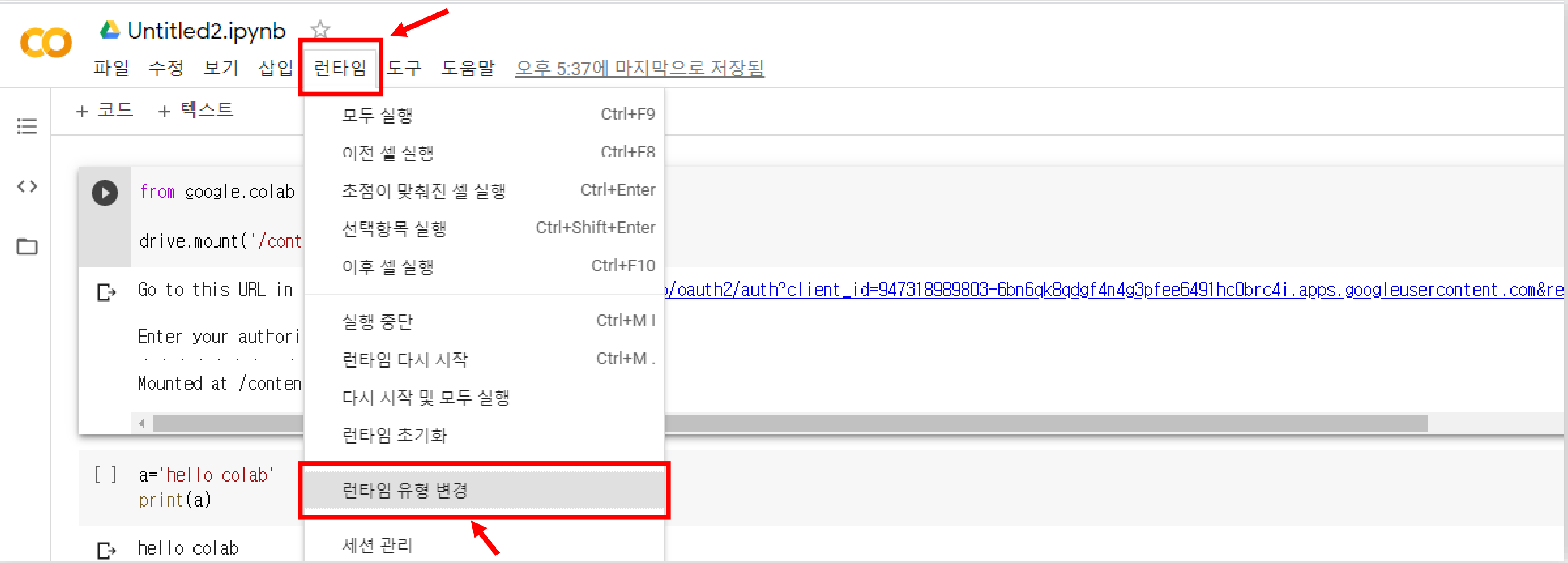
- 구글 드라이브와의 연결이 완료되었으면, 간단한 코드로 구글 서버에서 실행이 잘 되는지 확인.
- 해당 노트 파일 (.ipynb) 은 구글 드라이브에 자동 저장.



실습 – Google Colab 사용하기

구글 코랩 사용 방법 -7

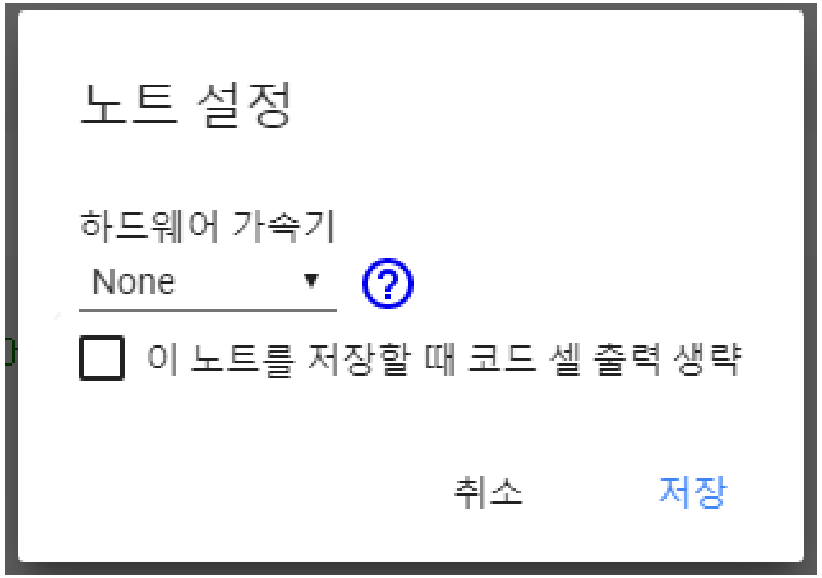
- 만약 데이터가 대용량이고 매우 복잡한 모델을 학습시킬 때는 코랩 내에서 GPU를 간단한 방법으로 활용할 수 있음.
- 상단의 "런타임" → "런타임 유형 변경" 클릭

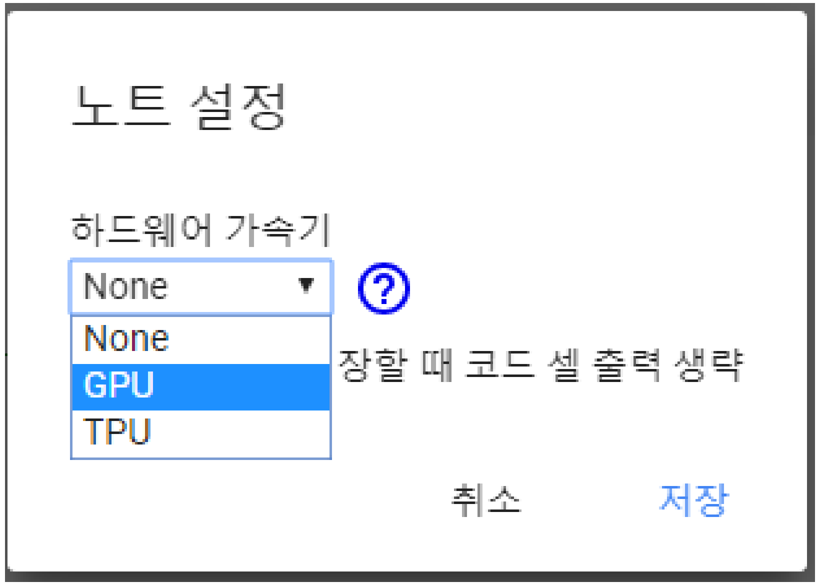


실습 – Google Colab 사용하기

구글 코랩 사용 방법 -8

- 하드웨어 가속기를 GPU로 바꿔주고 저장하면 GPU로 코드 실행 가능.





실습 – 구글 드라이브에 mnist dataset 업로드

연결된 구글 코랩 환경에서 실습 진행 -1

- 캐글에서 mnist dataset을 csv버전으로 다운로드. (https://www.kaggle.com/oddrationale/mnist-in-csv)
- 학습을 위하여 첫 번째 행은 제거.

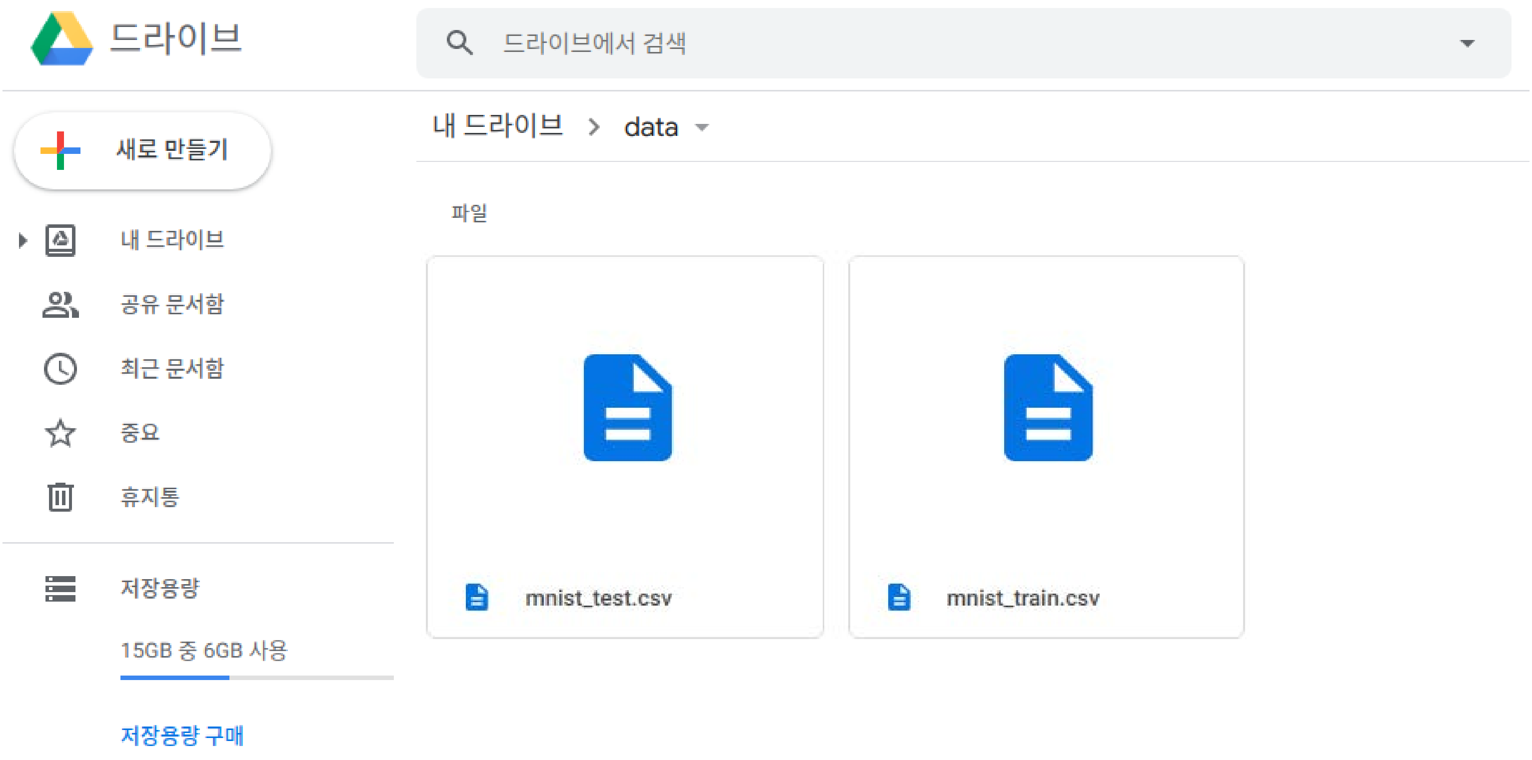
제거

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	label	1x1	1x2	1x3	1x4	1x5	1x6	1x7	1x8	1x9	1x10	1x11	1x12	1x13	1x14	1x15	1x16	1x17	1x18
2	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

실습 – 구글 드라이브에 mnist dataset 업로드

연결된 구글 코랩 환경에서 실습 진행 -2

- 연결된 계정의 구글 드라이브에 “data” 라는 이름의 폴더 생성.
- 다운로드 받은 “mnist_train”, “mnist_test” 두 개의 파일을 해당 폴더에 업로드.



실습 – MLP 및 Back-propagation 직접 코딩

연결된 구글 코랩 환경에서 실습 진행 -3

- 코드 별 설명은 주석으로 표시함.

```
[3] #mnist_train(60000개)과 mnist_test(10000개) 데이터를 각각 불러온다.
    data_file = open("/content/gdrive/My Drive/data/mnist_train.csv", "r") #연결되어 있는 구글 드라이브에 업로드했던 데이터를 코랩 환경으로 불러오는 코드
    training_data = data_file.readlines()
    data_file.close()

    test_data_file = open("/content/gdrive/My Drive/data/mnist_test.csv", "r")
    test_data = test_data_file.readlines()
    test_data_file.close()
```

실습 – MLP 및 Back-propagation 직접 코딩

연결된 구글 코랩 환경에서 실습 진행 -4

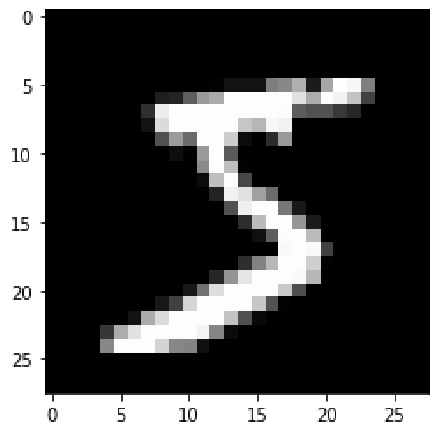
- 코드 별 설명은 주석으로 표시함.

[4]
```
#matplotlib과 numpy라이브러리를 불러온 후 데이터 하나를 시각화해본다.
import matplotlib.pyplot as plt
import numpy as np

t = np.asfarray(training_data[0].split(","))

# 일렬로 늘어진 픽셀정보를 28x28 행렬로 바꾼다
n = t[1:].reshape(28,28)

plt.imshow(n, cmap='gray')
plt.show()
```



실습 – MLP 및 Back-propagation 직접 코딩

연결된 구글 코랩 환경에서 실습 진행 -5

- 코드 별 설명은 주석으로 표시함. (Deep Neural Network 클래스 정의 부분)

```
[5]  class DeepNeuralNetwork:
         #DeepNeuralNetwork 클래스를 initialize
         def __init__(self, input_layers, hidden_layer_1, hidden_layer_2, hidden_layer_3, output_layers):
             self.inputs = input_layers
             self.hidden_1 = hidden_layer_1
             self.hidden_2 = hidden_layer_2
             self.hidden_3 = hidden_layer_3
             self.outputs = output_layers
             self.test_data = None

             #가중치 값들을 모두 랜덤으로 초기화
             self.w_ih = np.random.randn(self.inputs, self.hidden_1) / np.sqrt(self.inputs/2)
             self.w_hh_12 = np.random.randn(self.hidden_1, self.hidden_2) / np.sqrt(self.hidden_1/2)
             self.w_hh_23 = np.random.randn(self.hidden_2, self.hidden_3) / np.sqrt(self.hidden_2/2)
             self.w_ho = np.random.randn(self.hidden_3, self.outputs) / np.sqrt(self.hidden_3/2)

         # feed-forward를 진행한다.
         def predict(self, x):
             # 문자열을 float array로 바꾸는 과정
             data = self.normalize(np.asfarray(x.split(',')))

             # 0번은 레이블이므로 제외
             data = data[1:]

             #3개의 은닉층(2개의 sigmoid와 1개의 tanh)과 하나의 출력층(softmax)
             layer_1 = self.sigmoid(np.dot(data, self.w_ih))
             layer_2 = self.tanh(np.dot(layer_1, self.w_hh_12))
             layer_3 = self.sigmoid(np.dot(layer_2, self.w_hh_23))
             output = self.softmax(np.dot(layer_3, self.w_ho))
             return output
```

실습 – MLP 및 Back-propagation 직접 코딩

연결된 구글 코랩 환경에서 실습 진행 -6

- 코드 별 설명은 주석으로 표시함. (Deep Neural Network 클래스 정의 부분 계속)

```
# training_data로 학습 진행
def train(self, training_data, learning_rate, epoch):
    for ech in range(0, epoch):
        for i, x in enumerate(training_data):
            target = np.array(np.zeros(self.outputs) + learning_rate, ndmin=2)
            target[0][int(x[0])] = 1-learning_rate
            x = self.normalize(np.asfarray(x.split(",")))

            # feed-forward propagation
            layer1 = self.sigmoid(np.dot(x[1:], self.w_ih))
            layer2 = self.tanh(np.dot(layer1, self.w_hh_12))
            layer3 = self.sigmoid(np.dot(layer2, self.w_hh_23))
            layer4 = self.softmax(np.dot(layer3, self.w_ho))

            # back propagation
            layer4_reverse = (target - layer4)
            layer3_reverse = layer4_reverse.dot(self.w_ho.T) * (layer3 * (1 - layer3))
            layer2_reverse = layer3_reverse.dot(self.w_hh_23.T) * (1 - layer2) * (1 + layer2)
            layer1_reverse = layer2_reverse.dot(self.w_hh_12.T) * (layer1 * (1 - layer1))

            # weight update
            self.w_ho = self.w_ho + learning_rate * layer4_reverse.T.dot(np.array(layer3, ndmin=2)).T
            self.w_hh_23 = self.w_hh_23 + learning_rate * layer3_reverse.T.dot(np.array(layer2, ndmin=2)).T
            self.w_hh_12 = self.w_hh_12 + learning_rate * layer2_reverse.T.dot(np.array(layer1, ndmin=2)).T
            self.w_ih = self.w_ih + learning_rate * layer1_reverse.T.dot(np.array(x[1:], ndmin=2)).T

            #2000개에 한 번씩 accuracy 출력
            if i % 2000 == 0 :
                self.print_accuracy()
```

Sigmoid(x)의 미분 = sigmoid(x)(1-sigmoid(x))
Tanh(x)의 미분 = (1-tanh(x))(1+tanh(x))

각 층에 사용된 활성함수의 미분을 이용하여
역방향으로 각 층의 가중치 값을 업데이트

실습 – MLP 및 Back-propagation 직접 코딩

연결된 구글 코랩 환경에서 실습 진행 -7

- 코드 별 설명은 주석으로 표시함. (Deep Neural Network 클래스 정의 부분 계속)

```
    # 현재 neural network의 accuracy를 출력한다.
    def print_accuracy(self):
        matched = 0

        for x in self.test_data:
            label = int(x[0])
            predicted = np.argmax(self.predict(x))
            if label == predicted :
                matched = matched + 1
        print('accuracy : {0}'.format(matched/len(self.test_data)))

    #sigmoid함수 정의
    def sigmoid(self, x):
        return 1.0/(1.0 + np.exp(-x))

    #feature scaling을 위한 normalize 함수 정의
    def normalize(self, x):
        return (x / 255.0) * 0.99 + 0.01

    #tanh함수 정의
    def tanh(self, x):
        return (np.exp(x) - np.exp(-x))/(np.exp(x) + np.exp(-x))

    #softmax함수 정의
    def softmax(self, x):
        e_x = np.exp(x - np.max(x))
        return e_x / e_x.sum()
```

Feed-forward propagation, back propagation, 활성함수의 수식을 모두 포함한 하나의 MLP 모형 완성 (DeepNeuralNetwork 클래스)

실습 – MLP 및 Back-propagation 직접 코딩

연결된 구글 코랩 환경에서 실습 진행 -8

- 코드 별 설명은 주석으로 표시함. (앞서 정의한 Deep Neural Network 클래스 활용)

```
[6] #input layer, hidden layer 1, 2, 3, output layer의 노드 수를 각각 784, 100, 100, 100, 10개로 설정
    network = DeepNeuralNetwork(784, 100, 100, 100, 10)
    network.test_data = test_data
    #learning rate은 0.01, epoch는 1로 설정
    network.train(training_data, 0.01, 1)
```

```
accuracy : 0.0974
accuracy : 0.5862
accuracy : 0.8055
accuracy : 0.8391
accuracy : 0.8497
accuracy : 0.8657
accuracy : 0.8789
accuracy : 0.8076
accuracy : 0.8849
accuracy : 0.8804
accuracy : 0.876
accuracy : 0.8826
accuracy : 0.8983
accuracy : 0.8891
accuracy : 0.8909
accuracy : 0.8781
accuracy : 0.9097
accuracy : 0.9084
accuracy : 0.8837
accuracy : 0.9036
accuracy : 0.9004
accuracy : 0.9091
accuracy : 0.9159
accuracy : 0.9068
accuracy : 0.9171
accuracy : 0.9237
accuracy : 0.9132
accuracy : 0.9157
accuracy : 0.9008
accuracy : 0.9159
```

정의된 MLP 모형을 활용하여 test 데이터에서의 예측까지 수행.