

인공신경망 모델의 구조 및 동작원리와 최적화 이론의 이해



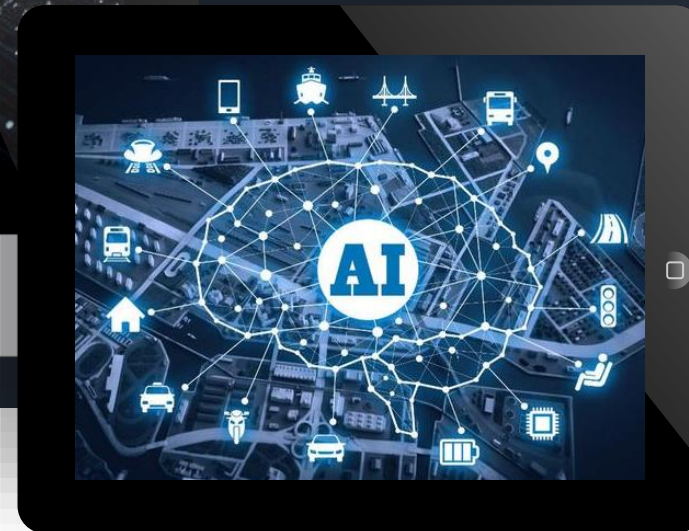
인공신경망 모델의 구조

모델 최적화

DNN 구현

이 슬라이드에서 사용한 서체 :

- Open Sans(<https://ko.cooltext.com/Download-Font-Open+Sans>)
- KoPubWorld돋움체(<http://www.kopus.org/biz/electronic/font.aspx>)



과정 안내

인공신경망 모델의 구조 및 동작원리와 최적화 이론의 이해

✓ 인공지능망 모델의 구조 및 동작원리와 최적화 이론의 이해

- ▶ 인공신경망 이론을 이해하고 구현할 수 있다.
- ▶ 모델을 최적화 할 수 있다.
- ▶ DNN 모델을 구현할 수 있다.

✓ 주요 내용의 흐름

인공신경망 딥러닝 이론



DNN 모델 구현하기

✓ 선수지식

- ▶ 파이썬 프로그래밍 언어

<https://inf.run/D9guA>



빅데이터 분석 및 인공지능 개발자를 위한 로드맵

1. 딥러닝 프레임워크 개요 및 설치 방법

개발 환경 구성



인공지능의 역사 및 딥러닝의 혁신

1. 딥러닝 프레임워크 개요 및 설치 방법

인공지능의 역사 및 딥러닝의 혁신

~1990년 : 이론 정립

~2000년 : 구현 시도

~2010년 : 본격 시도

2010년~ : 혁신의 시작
(딥러닝 기반의 인공지능)

시대별 한계 : 컴퓨팅의 한계로 제안된 이론
구현의 어려움

데이터의 한계로 현실 문제
해결하지 못함

알고리즘의 한계로 완성도 부족

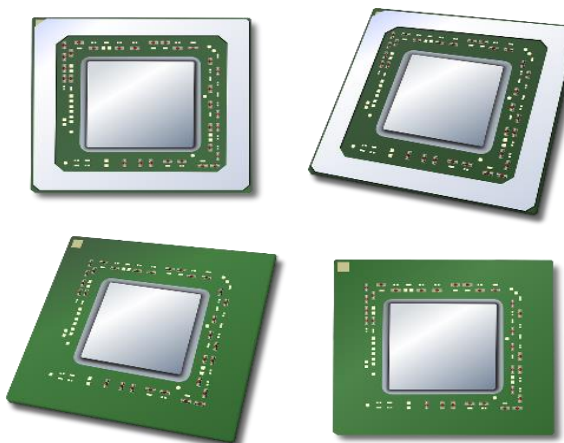
혁신적 알고리즘 제안



Geoffrey Hinton
(Univ. of Toronto, Google)

- 혁신적 딥러닝 이론 제안(2006년)
- 실제 구현으로 혁신적 성능 증명
- 이미지 인식 대회인 ImageNet Challenge
에서 압도적 성능으로 우승(2012년)

컴퓨팅 파워 발전



- CPU의 고성능화 및 저가격화
- GPU를 이용한 AI 연산

학습용 데이터셋의 증가



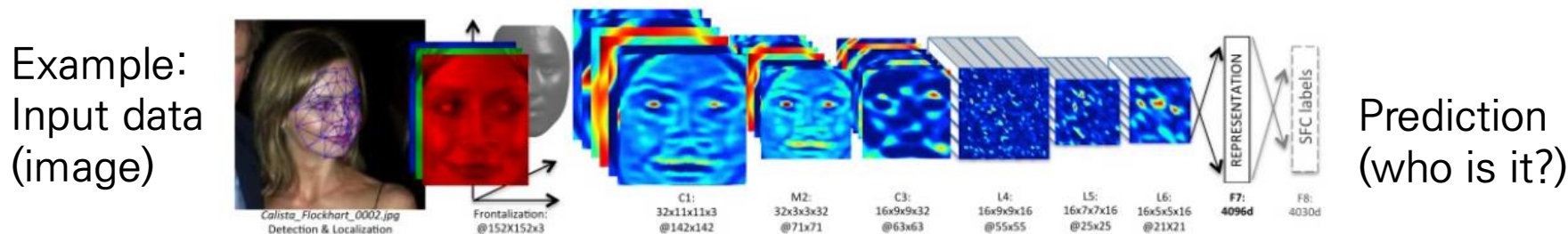
- 2025년 데이터: 175ZB (175조 GB)
- 정부차원의 AI Hub 구축

딥러닝 정의

1. 딥러닝 프레임워크 개요 및 설치 방법

- Wikipedia:

Deep learning is a class of machine learning algorithms that uses multiple layers to progressively extract higher-level features from the raw input. For example, in image processing, lower layers may identify edges, while higher layers may identify the concepts relevant to a human such as digits or letters or faces.



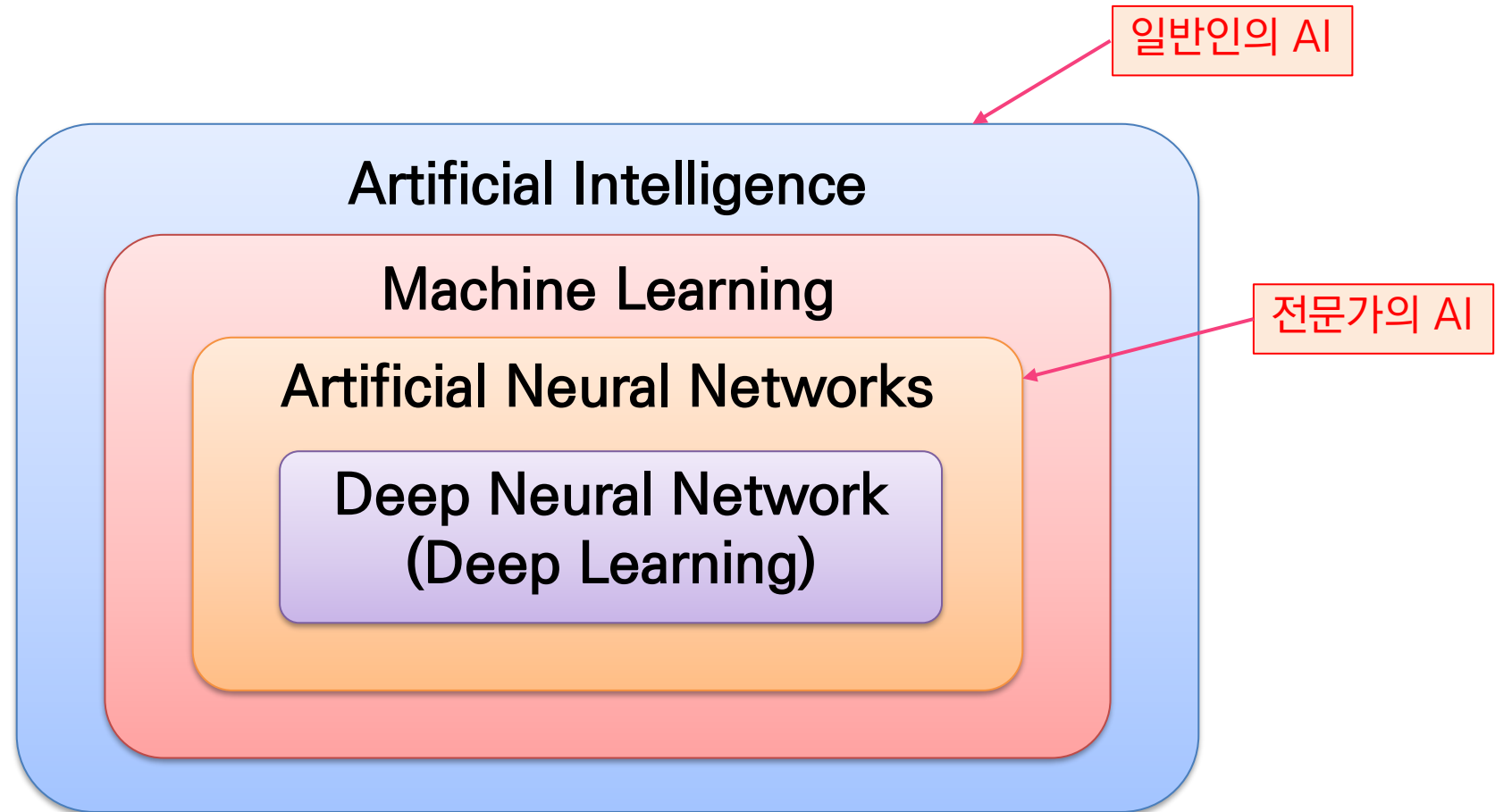
DeepFace rotating a celebrity's face to use for facial verification. Image credit: Facebook

Deep Learning

- ▶ 딥러닝은 머신러닝 알고리즘의 집합
- ▶ 최근의 딥러닝은 인공신경망 CNN을 기반으로 함
- ▶ 비선형 변환으로 구성된 아키텍처를 사용하여 데이터로부터 모델을 만듦

AI vs. 머신러닝 vs. 딥러닝

1. 딥러닝 프레임워크 개요 및 설치 방법



딥러닝 개발을 위한 언어와 프레임워크

1. 딥러닝 프레임워크 개요 및 설치 방법

Python

- **주요 역할:**
 - Python은 AI 개발에 가장 널리 사용되는 언어
 - 수많은 라이브러리와 프레임워크가 있어 AI 모델을 빠르고 효율적으로 개발할 수 있음
- **유용한 라이브러리:**
 - NumPy, SciPy: 수학 및 과학 연산 지원.
 - Pandas: 데이터 처리 및 분석.
 - Matplotlib, Plotly: 데이터 시각화.
 - TensorFlow, PyTorch: 딥러닝 프레임워크.

AI 개발 도구 및 프레임워크

- **TensorFlow**
 - **주요 역할:** Google에서 개발한 딥러닝 라이브러리로, 대규모 신경망을 구축하고 훈련시키는 데 사용.
 - **특징:** 다양한 플랫폼에서 동작 가능, 높은 확장성.
- **PyTorch**
 - **주요 역할:** Facebook에서 개발한 딥러닝 프레임워크로, 연구와 상용화 모두에 적합함.
 - **특징:** 동적 계산 그래프, 빠른 실험 가능.
- **Keras**
 - **주요 역할:** TensorFlow의 고급 API로, 신경망을 쉽게 설계하고 훈련할 수 있게 함.
 - **특징:** 간결하고 직관적인 인터페이스.
- **Scikit-learn**
 - **주요 역할:** Python에서 많이 사용되는 머신러닝 라이브러리로, 다양한 머신러닝 알고리즘과 데이터 처리 도구를 제공함.
 - **특징:** 분류, 회귀, 클러스터링 등 다양한 알고리즘 지원.

딥러닝을 위한 개발 도구

1. 딥러닝 프레임워크 개요 및 설치 방법

- **Jupyter Notebook**
 - 데이터 분석, 머신러닝 모델 개발에 널리 사용되는 대화형 개발 환경.
 - 실시간 코드 실행 및 시각화.
- **Anaconda**
 - Python과 R을 위한 데이터 과학 배포판으로, 패키지 및 환경 관리를 간편하게 도와줌.
 - 여러 라이브러리 및 도구들의 설치 및 관리를 간소화함.
- **PyCharm**
 - JetBrains에서 개발한 Python 전용 통합 개발 환경(IDE)으로 Python 개발에 최적화된 다양한 기능을 제공함
 - PyCharm은 로컬 개발 환경에서 안정적인 코드 작성과 디버깅에 초점을 둠
 - PyCharm은 대규모 프로젝트나 고급 기능(예: 코드 리팩토링, 디버깅)을 필요로 하는 경우 유리
 - Python 인터프리터를 설치해야 하며 필요한 라이브러리를 설치해야 함
 - 프로젝트별로 가상환경이 제공되어 프로젝트간 라이브러리 버전 충돌 문제를 최소화 함
- **Google Colaboratory**
 - 구글이 제공하는 무료 클라우드 기반의 Jupyter Notebook 환경
 - Python 코드를 브라우저에서 바로 실행할 수 있으며, 별도의 설치 없이도 인공지능 및 데이터 과학 프로젝트를 진행할 수 있음
 - Colab은 클라우드 기반 환경에서 손쉽게 코드를 실행하고 AI 모델을 훈련하는 데 강점을 가짐
 - 딥러닝 모델을 빠르게 실험하고 클라우드 자원을 활용하고 싶을 때 적합
- **기타**
 - Kaggle Kernels, Microsoft Azure Notebooks, Amazon SageMaker Notebooks, IBM Watson Studio 등

케라스와 파이토치

1. 딥러닝 프레임워크 개요 및 설치 방법



Keras

- 높은 수준의 API를 제공함
- 코드가 직관적이고 사용하기 쉬움
- TensorFlow의 고수준 API로 통합되었음
- TensorFlow 생태계를 잘 활용할 수 있음



PyTorch

- 낮은 수준의 API를 제공
- 더 유연하고 커스터마이징하기 좋음
- 연구 커뮤니티에서 강력한 지지를 받고 있음
- 연구 결과를 빠르게 구현할 수 있는 라이브러리나 도구가 많음

기능	케라스	파이토치
모델 정의	Sequential 또는 Functional API 사용	nn.Module 클래스 상속
레이어 추가	model.add 또는 리스트 내 정의	__init__ 메서드 내에 레이어 정의
순전파	자동으로 처리	forward 메서드 내에서 직접 정의
모델 컴파일	model.compile 사용	없음, 직접 손으로 해야 함
모델 훈련	model.fit 사용	훈련 루프 직접 작성
손실 함수	loss 인자로 지정	nn.CrossEntropyLoss 등 손실 함수를 직접 정의
최적화 알고리즘	optimizer 인자로 지정	optim.Adam 등 옵티마이저를 직접 정의



pytorch_lightning을 사용하면 fit(), eval() 등을 사용할 수 있습니다.

실습 환경

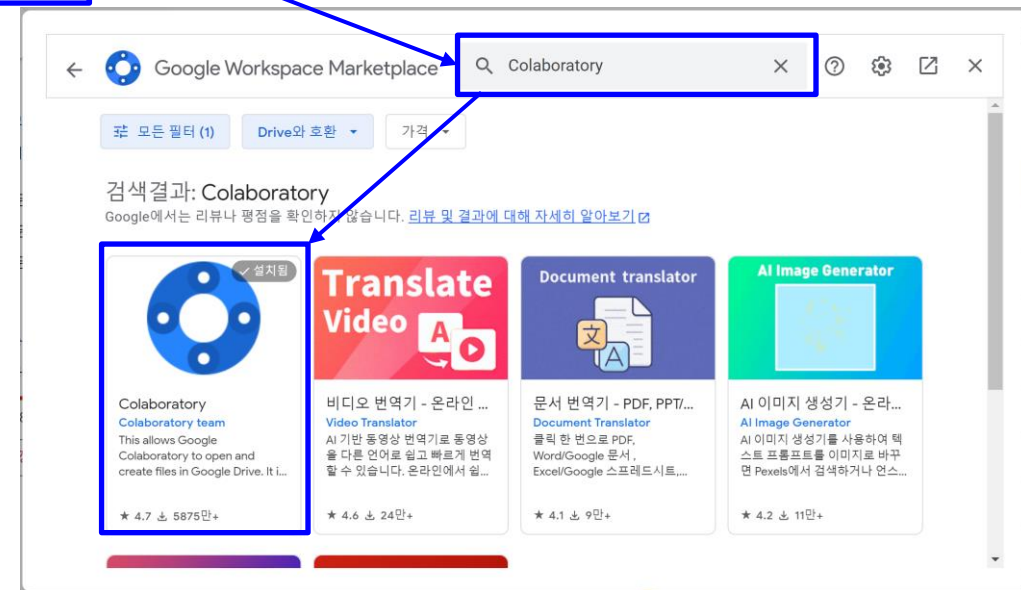
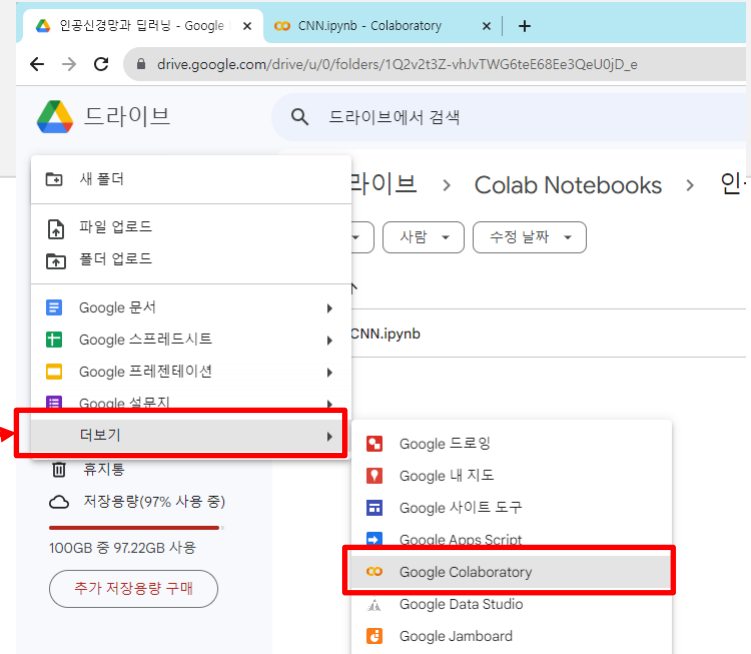
1. 딥러닝 프레임워크 개요 및 설치 방법

1. 개발 및 실행 환경은 Google Colab notebook을 사용합니다.

1. Chrome 브라우저에서 구글 계정으로 로그인
2. 구글 드라이브로 이동(<https://drive.google.com/>)
3. + 신규 -> 더보기 -> Google Colaboratory 선택

- Colaboratory가 없을 경우 '연결할 앱 더보기'를 선택하고 Colaboratory를 검색하여 설치하세요.

4. 코드 작성 후 실행은 '컨트롤 + 엔터'



코드 작성 및 실행

1. 딥러닝 프레임워크 개요 및 설치 방법

The screenshot shows the Google Colab web interface. The browser address bar displays 'colab.research.google.com/drive/1Reft5AUhyah9r_iC5sZsC1f0nvRk8_W'. The page title is '누구나 할 수 있는 인공지능 모델 구현.ipynb'. The left sidebar shows a file explorer with a folder named 'sample_data'. The main area contains two code cells. The first cell contains a list of instructions in Korean about how to use the interface. The second cell contains Python code for loading the Iris dataset from sklearn. Below the code cells, there are buttons for '+ 코드' (Add Code) and '+ 텍스트' (Add Text). A red box highlights the Python code in the second cell. A lightbulb icon is placed next to the code, and a blue arrow points from the '+ 코드' button to the lightbulb icon.

1 #은 주석입니다.
2 # 코드를 작성하는 이 곳을 셀(Cell)이라고 부릅니다.
3 # 코드를 실행하려면 셀의 왼쪽에 있는 셀 실행 버튼을 클릭하거나 Shift+Enter 키를 누르세요.
4 # Shift+Enter로 실행하면 아래에 셀이 없으면 새로운 셀을 추가하고 포커스를 아래 셀로 이동합니다.
5 # Ctrl+Enter로 실행하면 현재 셀을 실행하고 포커스를 현재 셀에 머무르게 합니다.
6 # Alt+Enter로 실행하면 현재 셀을 실행하고 항상 아래에 새로운 셀을 만듭니다.

1 # 셀의 위쪽 또는 아래쪽 선 중간에 마우스를 올리면 '+ 코드'와 '+ 텍스트' 버튼이 보입니다.
2 # 새로운 코드 셀을 추가하려면 '+ 코드' 버튼을 클릭하세요.

✓ 인공신경망 딥러닝 모델 구현

```
[ ] 1 from sklearn import datasets  
    2 iris = datasets.load_iris()  
    3 iris
```

{'data': array([[5.1, 3.5, 1.4, 0.2],
 [4.9, 3. , 1.4, 0.2],
 [4.7, 3.2, 1.3, 0.2],
 [4.6, 3.1, 1.5, 0.2],
 [5. , 3.6, 1.4, 0.2],
 [5.4, 3.9, 1.7, 0.4],

+ 코드 + 텍스트

Python 3 Google Compute Engine 백엔드에 연결됨

Shift+Enter로 실행하면 아래에 셀이 없으면 새로운 셀을 추가하고 포커스를 아래 셀로 이동합니다.

- Ctrl+Enter로 실행하면 현재 셀을 실행하고 포커스를 현재 셀에 머무르게 합니다.
- Alt+Enter로 실행하면 현재 셀을 실행하고 항상 아래에 새로운 셀을 만듭니다.

셀의 위쪽 또는 아래쪽 선 중간에 마우스를 올리면 '+ 코드'와 '+ 텍스트' 버튼이 보입니다.

- 새로운 코드 셀을 추가하려면 '+ 코드' 버튼을 클릭하세요.

#은 주석입니다.

- 코드를 작성하는 이 곳을 셀(Cell)이라고 부릅니다.
- 코드를 실행하려면 셀의 왼쪽에 있는 셀 실행 버튼을 클릭하거나 Shift+Enter 키를 누르세요.

2. 인공신경망 모델의 구조 및 동작원리

인공신경망 이론



뉴런

2. 인공신경망 모델의 구조 및 동작원리



인간의 뉴런(neuron)

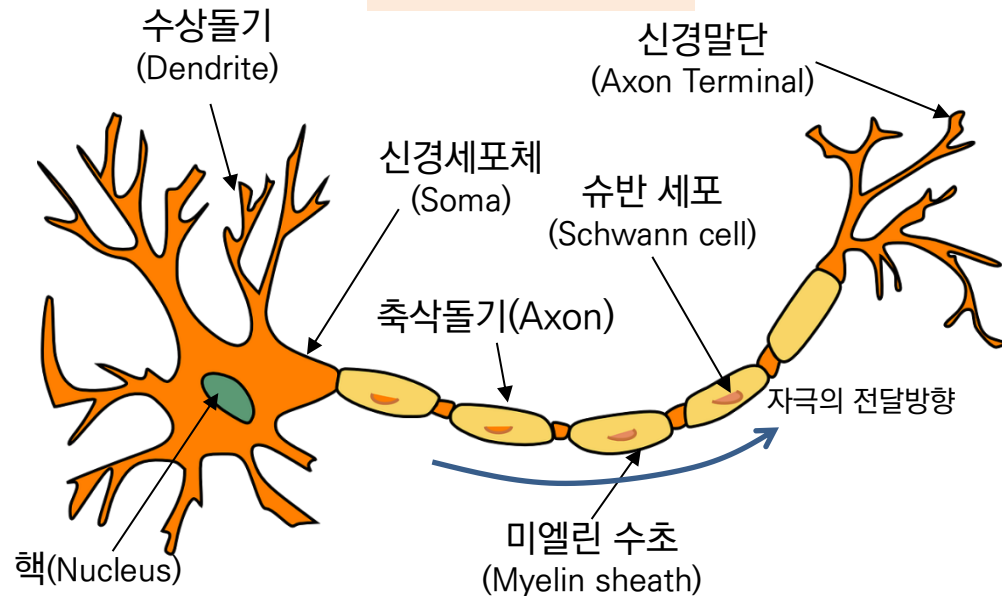
- ▶ 시냅스(synapse)를 통해 뉴런간 신호를 전달
- ▶ 각 뉴런은 수상돌기(dendrite)를 통해 입력 신호를 받음
- ▶ 입력 신호가 특정크기(threshold) 이상인 경우에만 활성화되어 축삭돌기(axon)을 통해 다음 뉴런으로 전달



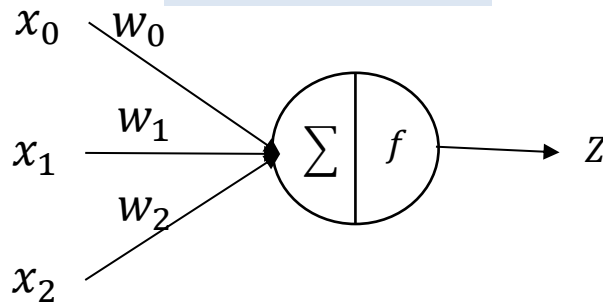
인공 뉴런(노드; node)

- ▶ 각 노드는 가중치가 있는 입력 신호를 받음
- ▶ 입력신호는 모두 더한 후, 활성화 함수(activation function)를 적용함
- ▶ 활성화 함수의 값이 특정 값 이상인 경우에만 다음 노드의 입력값으로 전달

인간의 뉴런 구조



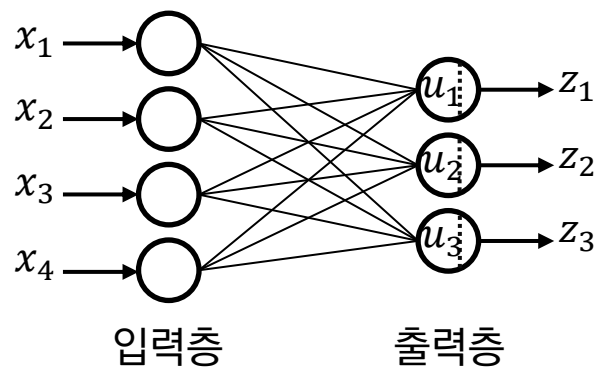
인공 뉴런의 구조



얕은 신경망

2. 인공신경망 모델의 구조 및 동작원리

Shallow Neural Network : 은닉층의 수가 적거나 없는 신경망



$$\begin{aligned}u_1 &= w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + w_{14}x_4 + b_1 \\u_2 &= w_{21}x_1 + w_{22}x_2 + w_{23}x_3 + w_{24}x_4 + b_2 \\u_3 &= w_{31}x_1 + w_{32}x_2 + w_{33}x_3 + w_{34}x_4 + b_3\end{aligned}$$

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}, \quad W = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{bmatrix}, \quad B = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}, \quad U = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}, \quad Z = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

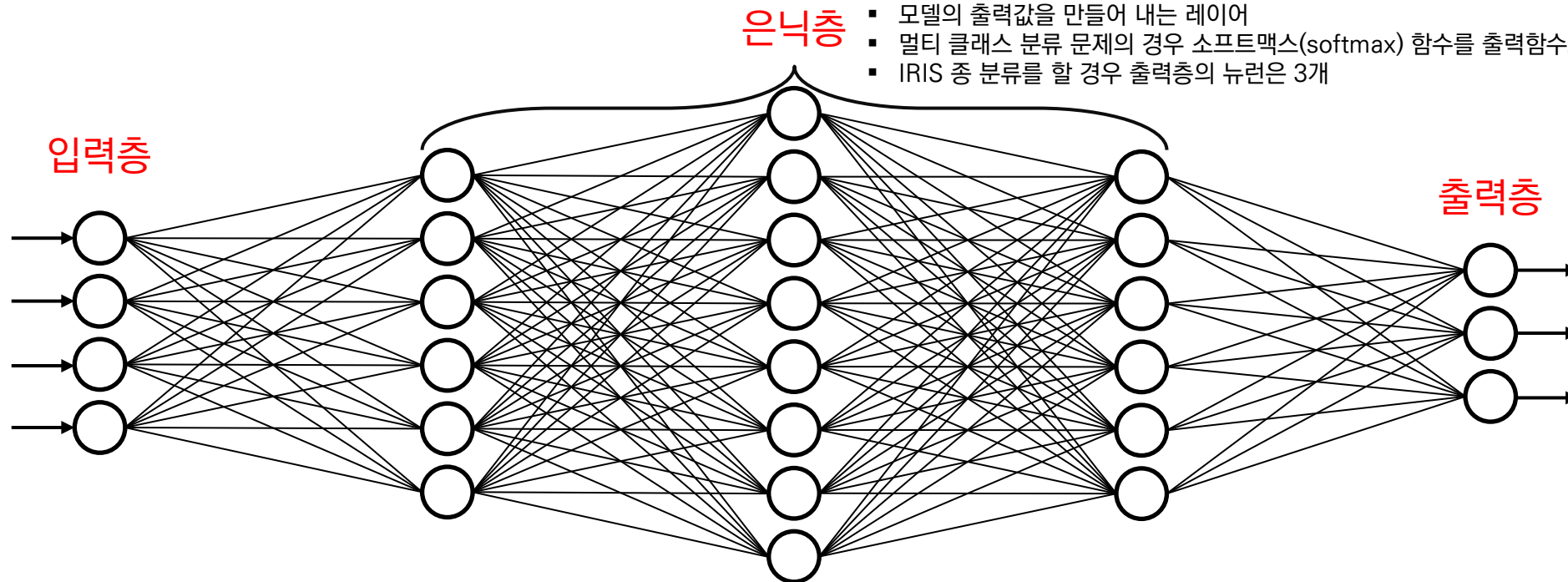
$$U = WX + B, \quad Z = f(U)$$

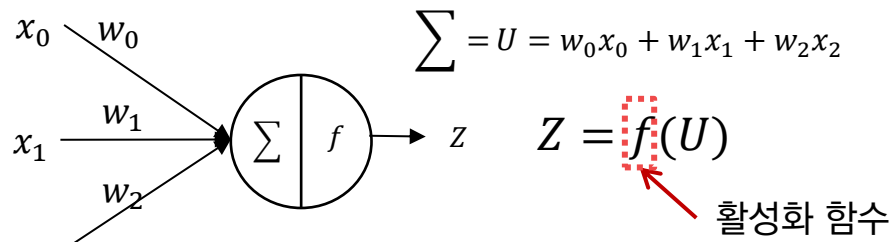
심층신경망

2. 인공신경망 모델의 구조 및 동작원리

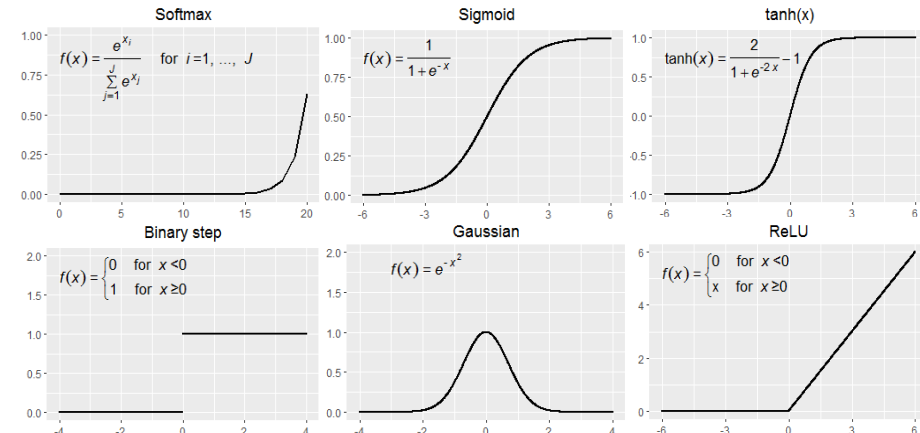
- 비선형 문제를 해결해야 할 경우면 더 깊은 신경망이 필요함
- 은닉층이 여러 개로 구성된 인공 신경망을 다층신경망(Multi Layered Perceptron)이라고 불렀음
- DNN(Deep Neural Network)은 다층 인공신경망임
- 실제로 구현된 DNN의 경우에는 단순히 층을 쌓은 구조는 아님
- GPT3는 파라미터 개수가 1750억 개로 층이 깊고 구조가 복잡함

- 입력층(input layer)
 - 입력 값으로 구성된 레이어
 - 학습 데이터셋의 “입력 변수의 개수” 만큼의 노드로 구성
 - IRIS 종 분류를 할 경우 입력층의 뉴런은 4개
- 은닉층(hidden layer)
 - 입력층과 출력층 사이의 레이어
 - 뉴런(neuron)과 시냅스(synapse)로 구성된 인간의 두뇌를 모방하는 레이어
 - 은닉층으로 들어오는 입력값의 합을 계산한 후 활성화 함수를 적용(활성화 함수에 따라 출력되는 값이 다름)
 - 은닉층의 개수와 각 층의 뉴런의 수를 정하는 것은 연구자의 몫
- 출력층(output layer)
 - 모델의 출력값을 만들어 내는 레이어
 - 멀티 클래스 분류 문제의 경우 소프트맥스(softmax) 함수를 출력함수로 사용
 - IRIS 종 분류를 할 경우 출력층의 뉴런은 3개



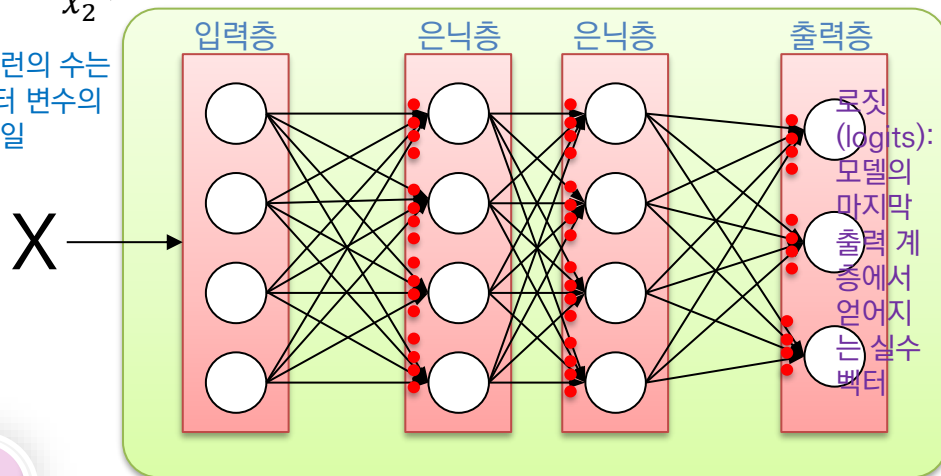


소프트맥스 함수: 로짓을 입력으로 받아 각 클래스에 대한 확률을 계산하는 함수



입력층 뉴런의 수는 입력데이터 변수의 개수와 동일

출력층 뉴런의 수는 분류분석의 경우 분류 레이블의 수이며, 회귀분석의 경우 1개



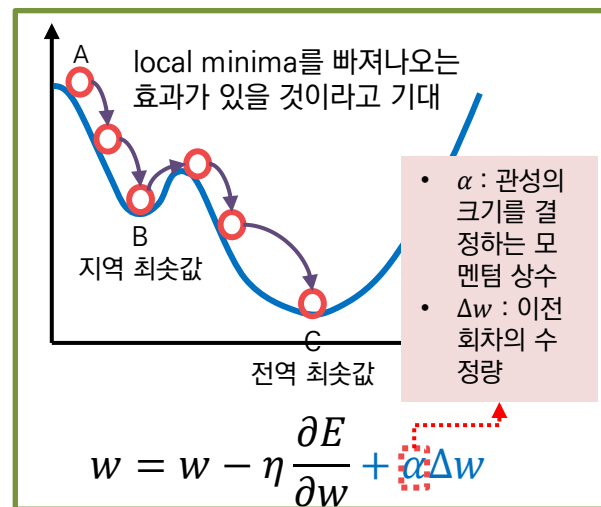
$\hat{y} \approx y$

차이(Loss, Error)

손실함수

MAE, MSE, Crossentropy

회귀, 분류



연구자가 고려해야 할 사항

- ▶ 은닉층의 수
- ▶ 은닉층별 뉴런의 수
- ▶ 활성화 함수
- ▶ 가중치 초기값
- ▶ 드롭아웃 비율
- ▶ 배치정규화
- ▶ 손실함수
- ▶ 옵티마이저, 학습률
- ▶ 학습 횟수(epochs)
- ▶ 배치 크기

가중치 업데이트

Gradient Descent

- weight
- 네트워크의 파라미터

$$w = w - \eta * \frac{\partial E}{\partial w}$$

- w에 대한 E의 편미분
- gradient
- 어떤 방향으로 학습할지

- Eta(learning rate)
- 한번에 얼마나 학습할지

- 편미분(偏微分, partial derivative)을 의미
- 기호는 델타의 변형
- 편-디(partial dee), 파셜디라고 부름

ChatGPT-3의 파라미터 개수는? 175B
ChatGPT-4의 파라미터 개수는? 1.6~16T

연구자가 결정해야 할 내용

2. 인공지능망 모델의 구조 및 동작원리



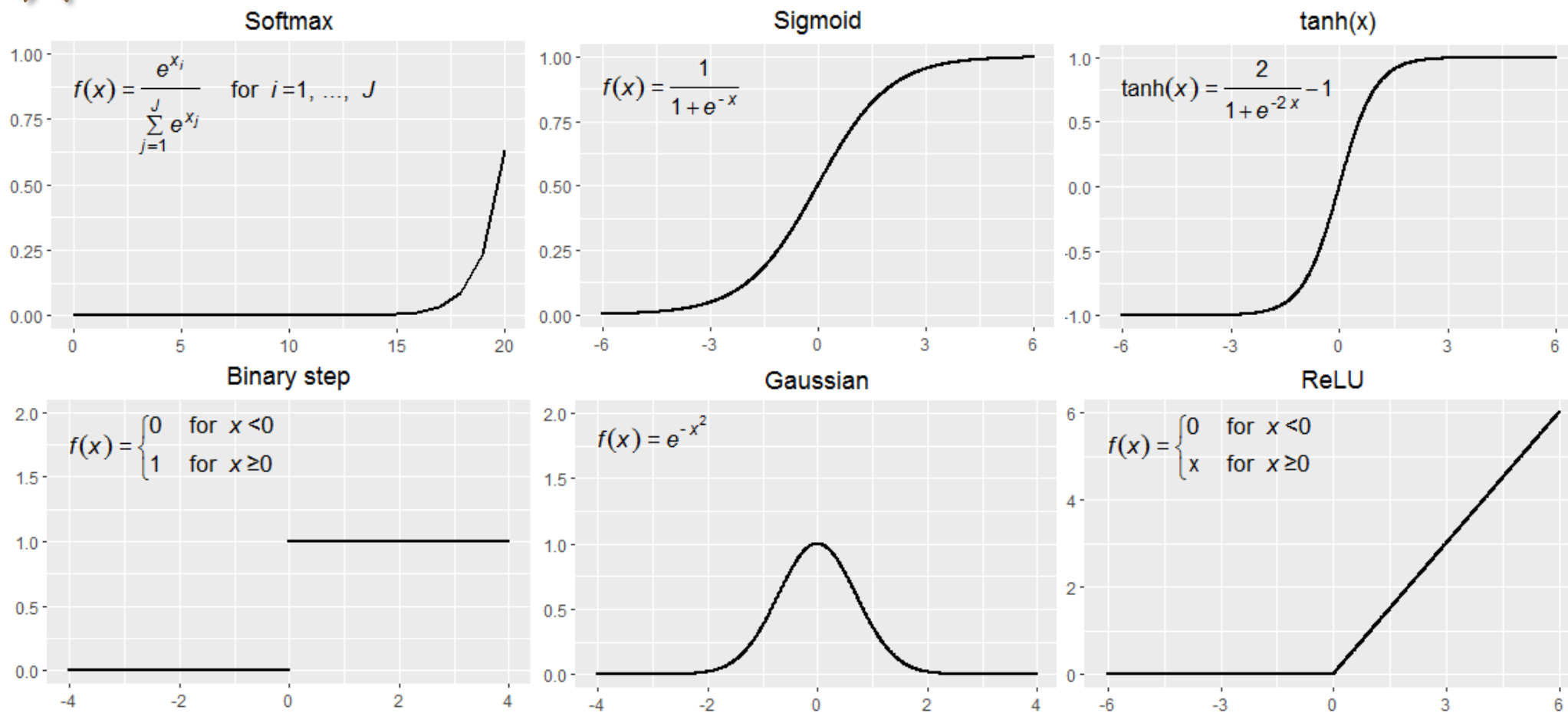
인공지능망 모델을 정의할 때에 연구자가 고려해야 할 사항

- ▶ 레이어의 수 : 인공지능망 모델에서 은닉층의 수를 지정하는 것
- ▶ 뉴런의 수 : 입력층의 뉴런의 수는 입력데이터의 변수의 수와 동일해야 하며, 출력층의 뉴런의 수는 분류분석의 경우 분류 레이블의 수이며, 회귀분석의 경우 1개. 은닉층은 연구자가 선택
- ▶ 활성화 함수 : 각 계층에서 사용할 활성화 함수를 지정하는 것
- ▶ 가중치 초기 값 : 각 층의 가중치 초기 값을 지정
- ▶ 드롭아웃 비율 : 과적합을 줄이기 위해 일부 뉴런의 출력을 0으로 하기 위한 비율
- ▶ 배치정규화 : 오차의 경사가 묻히는 현상(그래디언트 배니싱(*gradient vanishing*))을 줄이기 위해 은닉층 뉴런의 출력값을 정규화 할지 여부
- ▶ 손실함수 : 예측한 값과 실제 값과의 차이를 계산하는 함수를 지정
- ▶ 옵티마이저 : 손실함수를 최소화 하도록 가중치를 갱신시키기 위한 옵티마이저
- ▶ 학습률 : 옵티마이저가 사용할 학습률
- ▶ 학습 횟수(*epoch*) : 모든 데이터가 입력되어 가중치가 업데이트 되는 학습 횟수
- ▶ 배치 크기 : 1회 *epoch*가 학습될 동안 가중치가 업데이트 되어야 하는 입력데이터의 크기

활성화 함수(activation function)

2. 인공신경망 모델의 구조 및 동작원리

뉴런에서 입력 신호가 일정 크기 이상일 때만 신호를 전달하는 메커니즘을 모방한 함수

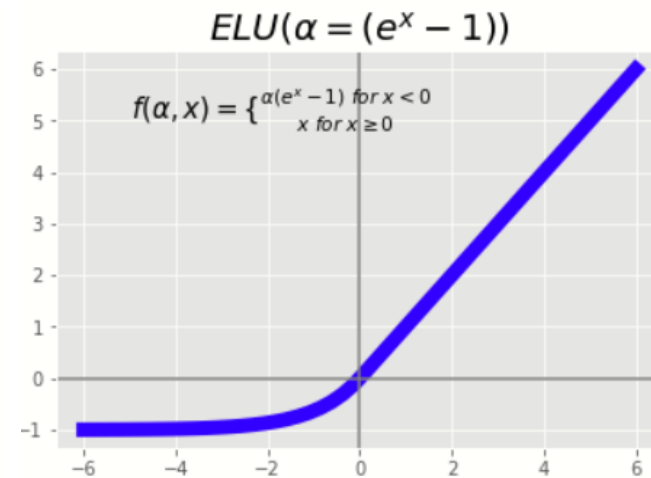
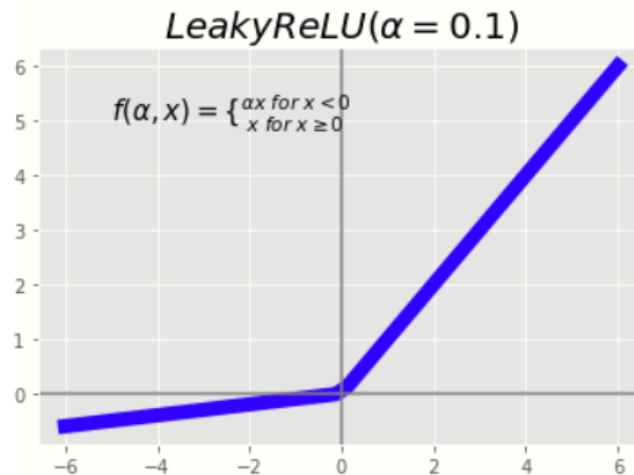
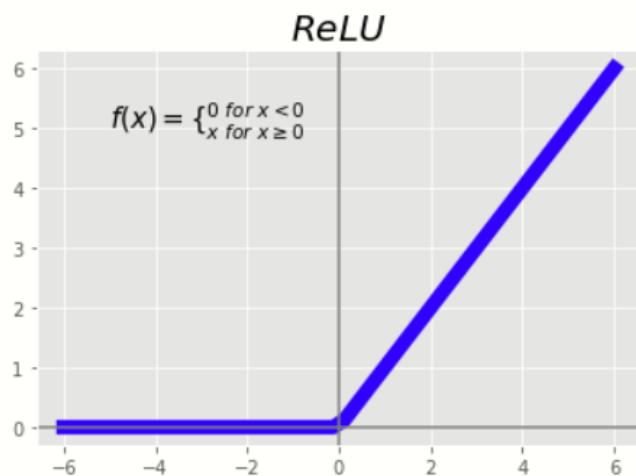


활성화 함수의 변화

2. 인공신경망 모델의 구조 및 동작원리

그래디언트 소실을 줄이는 방법 중 하나가 활성화 함수를 바꿔보는 것

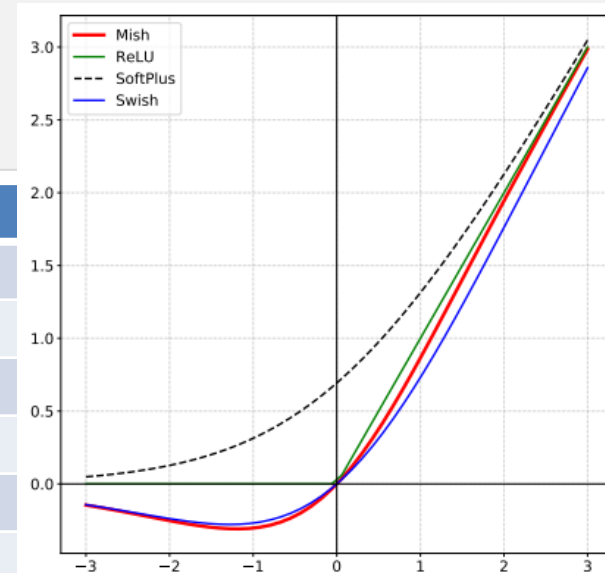
ELU → LeakyReLU → ReLU → tanh → sigmoid 순으로 사용해 볼 것을 권장



케라스의 활성화 함수

2. 인공신경망 모델의 구조 및 동작원리

이름	설명	참고 논문	
relu	Rectified Linear Unit		
sigmoid	Sigmoid		
softmax	Softmax		
softplus	Softplus		
softsign	Softsign		
tanh	Hyperbolic tangent		
selu	Scaled Exponential Linear Unit	https://arxiv.org/abs/1706.02515	2017
elu	Exponential Linear Unit	https://arxiv.org/abs/1511.07289	2016
exponential	Exponential		
leaky_relu	Leaky relu	https://arxiv.org/abs/1505.00853	2015
relu6	truncated to a maximum value of 6		
silu	Swish (or Silu)	https://arxiv.org/abs/1710.05941	2017
hard_silu	Hard SiLU	https://arxiv.org/abs/1905.02244	2019
gelu	Gaussian error linear unit	https://arxiv.org/abs/1606.08415	2016
hard_sigmoid	Hard sigmoid	https://en.wikipedia.org/wiki/Hard_sigmoid	
linear	Linear		
mish	Mish	https://arxiv.org/abs/1908.08681	2019
log_softmax	Log-Softmax		



케라스에서 활성화 함수 사용 예

2. 인공신경망 모델의 구조 및 동작원리

문자열 또는 함수 이름으로 활성화 함수 지정 가능

PReLU, LicheyReLU 등 일부 활성화 함수는 layers 모듈안에 클래스로 정의되어 있음

```
1 from tensorflow.keras.activations import softmax
2 from tensorflow.keras.layers import ELU # 패키지가 다름에 주의
3 from tensorflow.keras.layers import PReLU # 패키지가 다름에 주의
4 elu = ELU(alpha=1)
5 model.add(Input(4))
6 model.add(Dense(50, activation="sigmoid")) 문자열로 지정
7 model.add(Dense(50, activation=PReLU())) 클래스로 객체를 생성해서 지정
8 model.add(Dense(30, activation=elu)) 클래스로 객체를 생성한 변수로 지정
9 model.add(Dense(3, activation=softmax)) 함수 이름으로 지정
```

가중치 초기화

2. 인공신경망 모델의 구조 및 동작원리

가중치(Weight)의 초기값을 어떻게 주느냐에 따라 학습 진행 여부가 달라짐

- 가중치의 초기값을 모두 0으로 초기화하거나 모두 같은 값으로 초기화할 경우
- 가중치 초기값을 작은 값(Small Random numbers) 으로 초기화할 경우
- Xavier Initialization
- He Initialization

가중치 초기값이 0이거나 모두 같을 경우

2. 인공신경망 모델의 구조 및 동작원리



가중치의 초기값을 모두 0으로 초기화하거나 모두 같은 값으로 초기화할 경우

- ▶ 모든 뉴런이 같은 출력값을 내보낼 것
- ▶ 역전파(backpropagation) 단계에서 각 뉴런이 모두 같은 그래디언트 값을 가지게 됨
- ▶ 모든 뉴런이 동일한 그래디언트로 가중치 값이 변경되므로 뉴런의 개수가 아무리 많아도 뉴런이 하나뿐인 것처럼 작동하기 때문에 학습이 제대로 이루어지지 않음

작은 난수(Small Random numbers)인 경우

2. 인공신경망 모델의 구조 및 동작원리



가중치 초기값을 작은 값(Small Random numbers) 으로 초기화할 경우

- ▶ 활성화 함수가 sigmoid일 경우 만약 가중치 초기값(절대값)을 큰 값으로 한다면 0 과 1로 수렴하기 때문에 그래디언트 소실이 발생하게 됨
- ▶ 활성화 함수가 ReLU일 경우 가중치가 음수일 때는 dead ReLU 문제가 발생하고, 큰 양수일 때는 그래디언트 폭주가 일어나게 됨
- ▶ 가중치 초기값을 작게 초기화해야 하며 동일한 초기값을 가지지 않도록 랜덤하게 초기화해야 함
- ▶ 일반적으로 가중치 초기값은 평균이 0이고 표준편차가 0.01인 정규분포(가우시안 분포)를 따르는 값으로 랜덤하게 초기화
- ▶ 작은 난수 가중치 초기화 방법은 얇은 신경망에서는 나쁘지 않게 동작할 수 있으나 신경망의 깊이가 깊어질수록 문제가 발생하게 됨

케라스의 Dense 클래스 매개변수

```
kernel_initializer=initializers.RandomNormal(stddev=0.01)
```

Xavier Initialization

2. 인공신경망 모델의 구조 및 동작원리



세이비어 글로럿(Xavier Glorot)과 요수야 벤지오(Yoshua Bengio)

- ▶ <http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf>
- ▶ 적절한 데이터가 흐르기 위해서는 각 레이어의 출력에 대한 분산이 입력에 대한 분산과 같아야 하며, 역전파에서 레이어를 통과하기 전과 후의 그래디언트 분산이 동일해야 한다고 주장
- ▶ Glorot과 Bengio는 'Understanding the Difficulty of Training Deep Feedforward Neural Networks'라는 논문에서 아래의 식과 같은 가중치 초기값을 제안 했는데, 이러한 초기화 방법을 Xavier Initialization(또는 Glorot Initialization)이라고 함

평균이 0이고 표준편차가 $\sigma = \sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$ 인 정규분포

$r = \sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$ 일때 $-r$ 과 $+r$ 사이의 균등분포

입력의 연결 개수와 출력의 연결 개수가 비슷할 경우 $\sigma = 1 / \sqrt{n_{\text{inputs}}}$ $r = \sqrt{3} / \sqrt{n_{\text{inputs}}}$

He Initialization

2. 인공지능망 모델의 구조 및 동작원리



He Initialization

- ▶ Xavier 초기값은 ReLU 활성화 함수에서는 레이어가 깊어질수록 출력값이 0으로 치우치는 문제가 발생
 - ▶ Kaiming He는 ReLU에 적합한 초기값을 제안
 - ▶ He 초기값은 Xavier 초기값에서 $\sqrt{2}$ 배 함, 그 이유는 ReLU는 입력이 음수일 때 출력이 전부 0이기 때문에 더 넓게 분포시키기 위해서임
-
- He 초기값 또한 텐서플로우에서 쉽게 사용할 수 있도록 `tf.keras.initializers.he_normal` 과 `tf.keras.initializers.he_uniform`을 제공
 - 케라스를 사용한다면 `Dense()` 클래스의 `kernel_initializer="he_normal"` 매개변수를 통해 He 초기화를 사용할 수 있음(기본값은 `kernel_initializer="glorot_uniform"`)

tf.keras.initializers 모듈

2. 인공신경망 모델의 구조 및 동작원리

Initializer	클래스 원형
RandomNormal	<code>tf.keras.initializers.RandomNormal(mean=0., stddev=1.)</code>
RandomUniform	<code>tf.keras.initializers.RandomUniform(minval=-0.05, maxval=0.05, seed=None)</code>
TruncatedNormal	<code>tf.keras.initializers.TruncatedNormal(mean=0.0, stddev=0.05, seed=None)</code>
Zeros	<code>tf.keras.initializers.Zeros()</code>
Ones	<code>tf.keras.initializers.Ones()</code>
GlorotNormal	<code>tf.keras.initializers.GlorotNormal(seed=None)</code>
GlorotUniform	<code>tf.keras.initializers.GlorotUniform(seed=None)</code>
Identity	<code>tf.keras.initializers.Identity(gain=1.0)</code>
Orthogonal	<code>tf.keras.initializers.Orthogonal(gain=1.0, seed=None)</code>
Constant	<code>tf.keras.initializers.Constant(value=0)</code>
VarianceScaling	<code>tf.keras.initializers.VarianceScaling(scale=1.0, mode="fan_in", distribution="truncated_normal", seed=None)</code>

케라스에서 가중치 초기화 사용 예

2. 인공신경망 모델의 구조 및 동작원리

kernel_initializer 매개변수에 문자열 또는 함수 이름으로 활성화 함수 지정 가능

Initializers 모듈안의 클래스를 이용해서 지정 가능함

```
1 from tensorflow.keras.activations import softmax
2 from tensorflow.keras.initializers import RandomNormal
3
4 model.add(Input(4))
5 model.add(Dense(50, activation="sigmoid",
6                 kernel_initializer="he_normal"))
7 model.add(Dense(50, activation="sigmoid",
8                 kernel_initializer="glorot_uniform"))
9 model.add(Dense(30, activation="sigmoid",
10                kernel_initializer=RandomNormal(mean=0, stddev=0.1)))
11 model.add(Dense(3, activation="softmax"))
```

과적합

2. 인공신경망 모델의 구조 및 동작원리

과적합은 과대적합(Overfitting)과 과소적합(Underfitting)으로 나뉘짐

우리가 보통 말하는 과적합은 과대적합(Overfitting)을 의미함

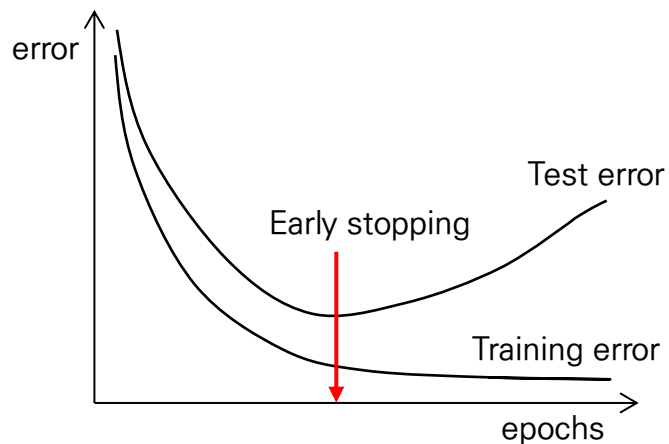
과대적합의 판단 : 훈련 데이터셋의 오차(또는 정확도)와 검증 데이터셋의 오차(또는 정확도)의 차이를 보고 판단

조기 종료

2. 인공신경망 모델의 구조 및 동작원리

훈련 데이터(Training set)의 오차는 지속적으로 줄어들지만, 검증 데이터(Test set)의 오차는 일정 시점이 지나면 오히려 증가하는 경향이 있음

학습을 도중에 멈추게 하는 방법으로 과적합 문제를 해결



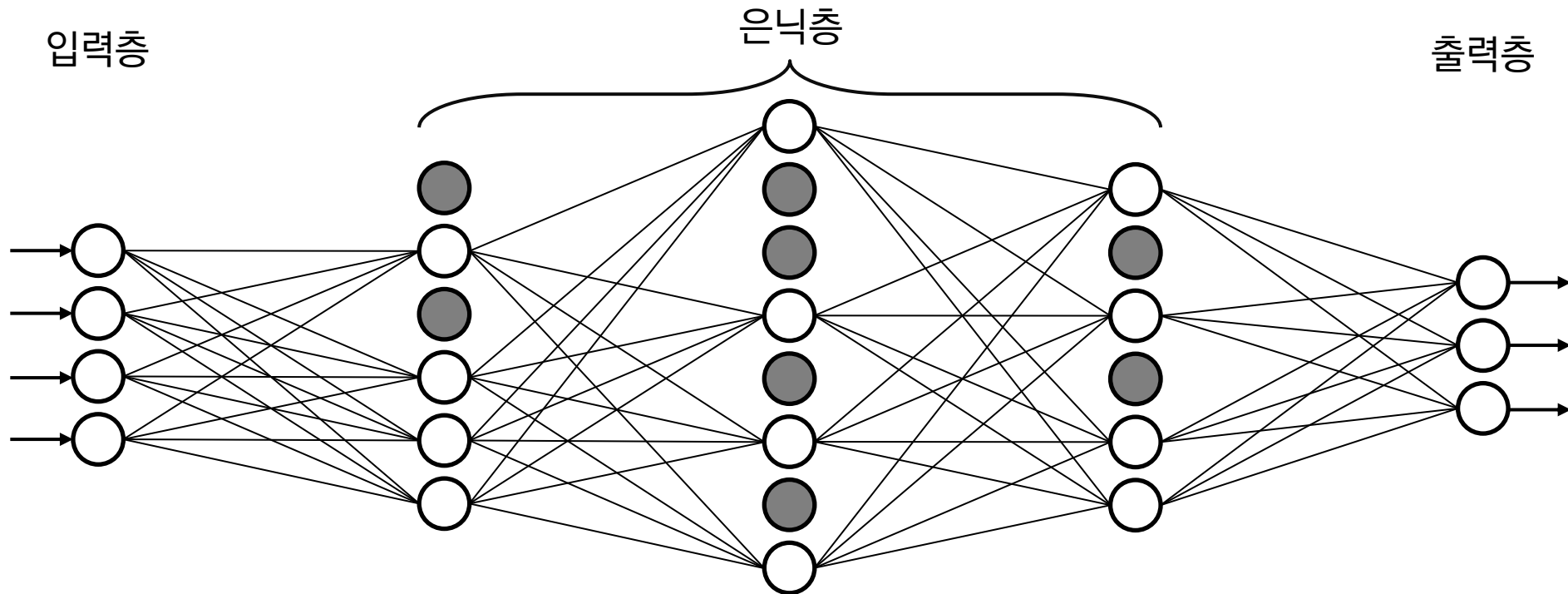
- 케라스의 fit() 함수는 매 이폭마다 loss, accuracy를 반환함
- 이를 이용해서 조기 종료할 학습횟수를 정하는 것은 과적합을 줄일 수 있음

드롭아웃

2. 인공신경망 모델의 구조 및 동작원리

층이 많고 뉴런의 수가 많으면 과적합으로 인해 검증 데이터셋의 오차가 늘어날 수 있음

드롭아웃(Dropout)은 인공신경망의 일부 뉴런이 학습하지 않도록 하는 것



케라스의 드롭아웃 사용 예

2. 인공신경망 모델의 구조 및 동작원리

케라스는 Dropout 클래스를 이용해서 드롭아웃 비율을 지정할 수 있음(0.1은 10%)

```
1. from tensorflow.keras.layers import Dense, Input, Dropout
2. from tensorflow.keras.models import Sequential
3.
4. model = Sequential()
5. model.add(Input(layer=(4,)))
6. model.add(Dense(50, activation="sigmoid"))
7. model.add(Dropout(0.4))
8. model.add(Dense(30, activation="sigmoid"))
9. model.add(Dense(3, activation="softmax"))
10.model.summary()
```

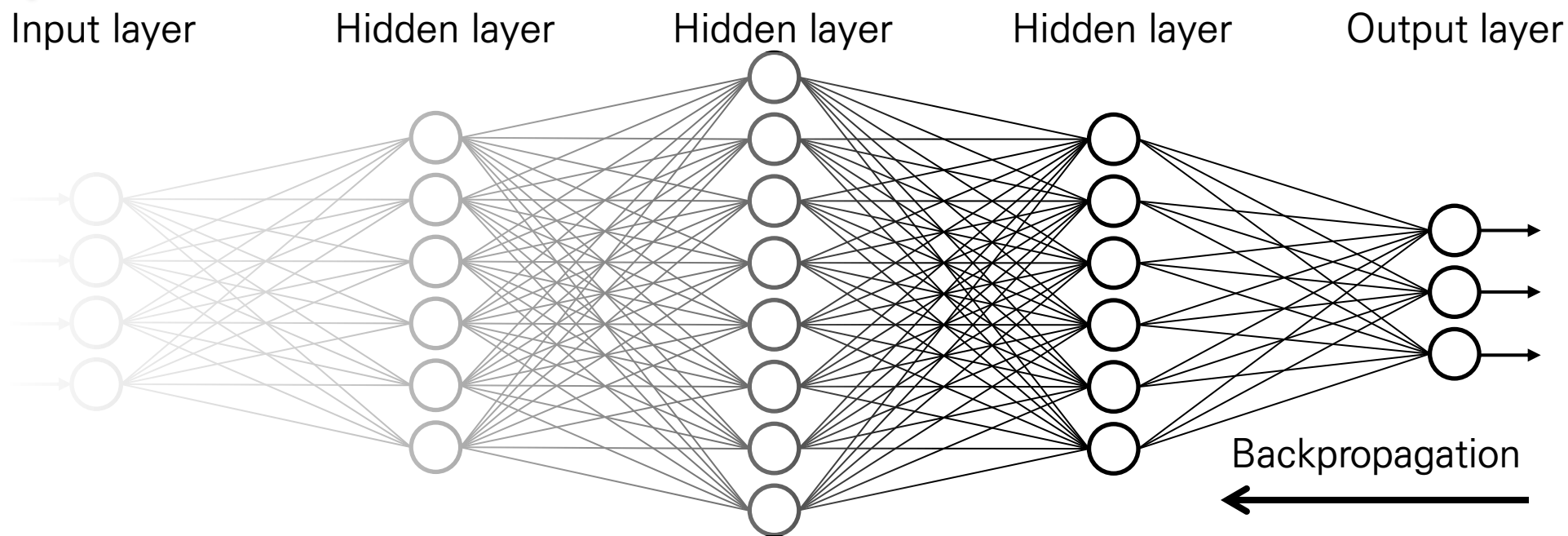
Gradient Vanishing

2. 인공신경망 모델의 구조 및 동작원리

은닉층이 많아질 수록...

- 경사 소실(Gradient Vanishing) 또는 폭주(Exploding)
- 모델이 복잡하고 커질수록 학습시간이 오래 걸림
- 모델이 복잡할수록 과적합(Overfitting)될 위험이 커짐

경사 소실 : 그래디언트가 점점 작아져 결국 가중치 매개변수가 업데이트 되지 않는 경우



Gradient Vanishing

2. 인공신경망 모델의 구조 및 동작원리

 Gradient Vanishing과 Gradient Exploding이 일어나지 않도록 다음의 방법들을 사용

- 활성화 함수의 변경(Change Activation Function)
- 가중치 초기값 변경(Weight Initialization)
- 작은 학습률(Small Learning Rate)
- 정규화(Normalization)

그래디언트 소실

2. 인공신경망 모델의 구조 및 동작원리

Gradient Vanishing과 Gradient Exploding이 일어나지 않도록 하는 방법

- ▶ Activation 함수의 변화 (ReLU 등)
- ▶ Carefully Weight Initialization
- ▶ Small Learning Rate

불안정화의 이유는 'Internal Covariance Shift'

- Internal Covariance Shift : 네트워크의 각 층이나 활성화 함수마다 입력의 분포가 달라지는 현상
- 학습하는 과정 자체를 전체적으로 안정화하여 학습 속도를 가속 시킬 수 있는 근본적인 방법을 사용해야 함
- ICS 현상을 막기 위해서 간단하게 각 층의 입력분포를 평균 0, 표준편차 1인 입력으로 정규화(normalize)시키는 방법을 사용

배치 정규화(Batch Normalization)

2. 인공신경망 모델의 구조 및 동작원리

- Gradient Vanishing/Gradient Exploding 이 일어나지 않도록 하는 아이디어 중 하나
- 지금까지는 간접적인 방법을 사용 했음
 - Activation 함수의 변화 (ReLU 등)
 - Careful Weight Initialization(mean 0, std 1, Gaussian Distribution)
 - Small Learning Rate 등으로 해결
- 불안정화가 일어나는 이유가 'Internal Covariance Shift' 라고 주장
 - Internal Covariance Shift라는 현상은 Network의 각 층이나 Activation 마다 input의 distribution이 달라지는 현상
 - Training 하는 과정 자체를 전체적으로 안정화하여 학습 속도를 가속시킬 수 있는 근본적인 방법을 사용
 - ICS 현상을 막기 위해서 간단하게 각 층의 input의 distribution을 평균 0, 표준편차 1인 input으로 normalize 시키는 방법(Keras의 BatchNormalization)
- Batch Normalization은 Whitening이라는 방법으로 해결할 수 있음
 - Whitening은 기본적으로 들어오는 input의 feature들을 uncorrelated 하게 만들어주고, 각각의 variance를 1로 만들어주는 작업
 - Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift (<https://arxiv.org/abs/1502.03167>)
- <https://keras.io/layers/normalization/>

케라스의 배치 정규화(Batch Normalization) 사용 예

2. 인공신경망 모델의 구조 및 동작원리

```
keras.layers.BatchNormalization(axis=-1, momentum=0.99,  
epsilon=0.001, center=True, scale=True, beta_initializer='zeros',  
gamma_initializer='ones', moving_mean_initializer='zeros',  
moving_variance_initializer='ones', beta_regularizer=None,  
gamma_regularizer=None, beta_constraint=None,  
gamma_constraint=None)
```

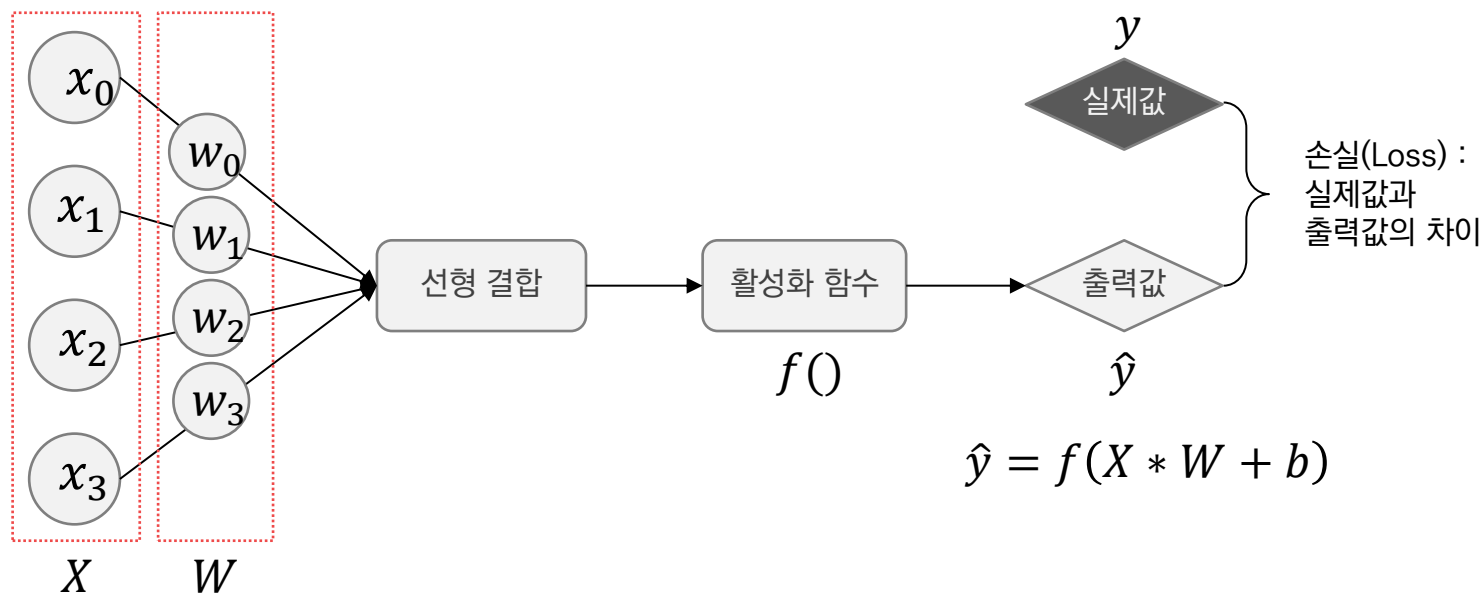
```
model = Sequential()  
model.add(Flatten(input_shape=(28, 28)))  
model.add(Dense(360, activation='relu'))  
model.add(Dropout(0.1))  
model.add(BatchNormalization())  
model.add(Dense(128, activation='relu'))
```

손실함수

2. 인공신경망 모델의 구조 및 동작원리

인공신경망에서 가중치(Weight)는 학습을 통해 손실(Loss)을 최소화 하는 방향으로 추정

추론한 값과 실제 값의 차이를 손실(Loss)로 정의하고 이러한 차이를 표현한 함수를 손실 함수(Loss Function)이라고 함



케라스의 손실함수

2. 인공신경망 모델의 구조 및 동작원리

이름	함수
mean_squared_error	<code>keras.losses.mean_squared_error(y_true, y_pred)</code>
mean_absolute_error	<code>keras.losses.mean_absolute_error(y_true, y_pred)</code>
mean_absolute_percentage_error	<code>keras.losses.mean_absolute_percentage_error(y_true, y_pred)</code>
mean_squared_logarithmic_error	<code>keras.losses.mean_squared_logarithmic_error(y_true, y_pred)</code>
squared_hinge	<code>keras.losses.squared_hinge(y_true, y_pred)</code>
hinge	<code>keras.losses.hinge(y_true, y_pred)</code>
categorical_hinge	<code>keras.losses.categorical_hinge(y_true, y_pred)</code>
logcosh	<code>keras.losses.logcosh(y_true, y_pred)</code>
categorical_crossentropy	<code>keras.losses.categorical_crossentropy(y_true, y_pred)</code>
sparse_categorical_crossentropy	<code>keras.losses.sparse_categorical_crossentropy(y_true, y_pred)</code>
binary_crossentropy	<code>keras.losses.binary_crossentropy(y_true, y_pred)</code>
kullback_leibler_divergence	<code>keras.losses.kullback_leibler_divergence(y_true, y_pred)</code>
poisson	<code>keras.losses.poisson(y_true, y_pred)</code>
cosine_proximity	<code>keras.losses.cosine_proximity(y_true, y_pred)</code>

케라스의 손실함수 설정 예

2. 인공신경망 모델의 구조 및 동작원리

compile() 함수의 loss 매개변수를 이용해서 손실함수 지정

함수 이름으로 지정하거나 문자열 형식으로 지정 가능

```
1 from tensorflow.keras import losses
2 model.compile(loss=losses.mean_squared_error,
                 optimizer='sgd')
```

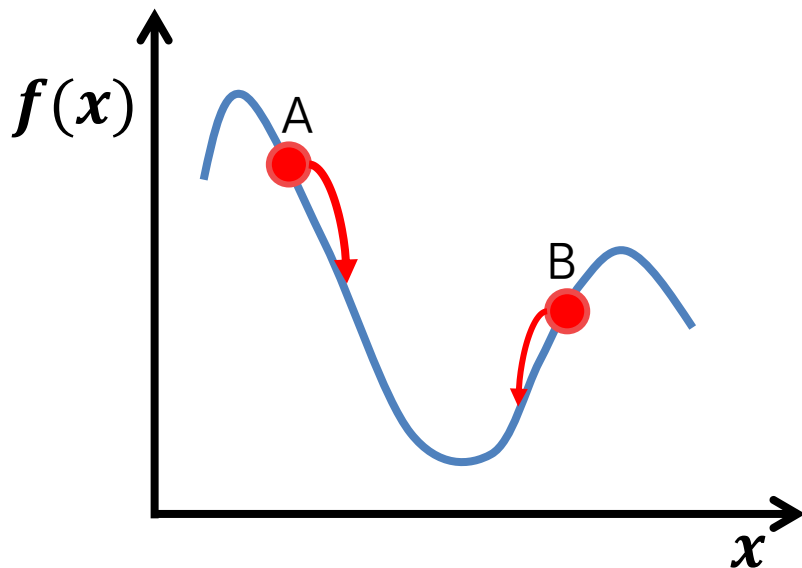
```
1 model.compile(loss='mean_squared_error', optimizer='sgd')
```

경사하강법

2. 인공신경망 모델의 구조 및 동작원리

가중치는 손실을 최소화하는 방향으로 추정

내려가는 방향은? → 미분으로 결정



$$\frac{df}{dx}$$

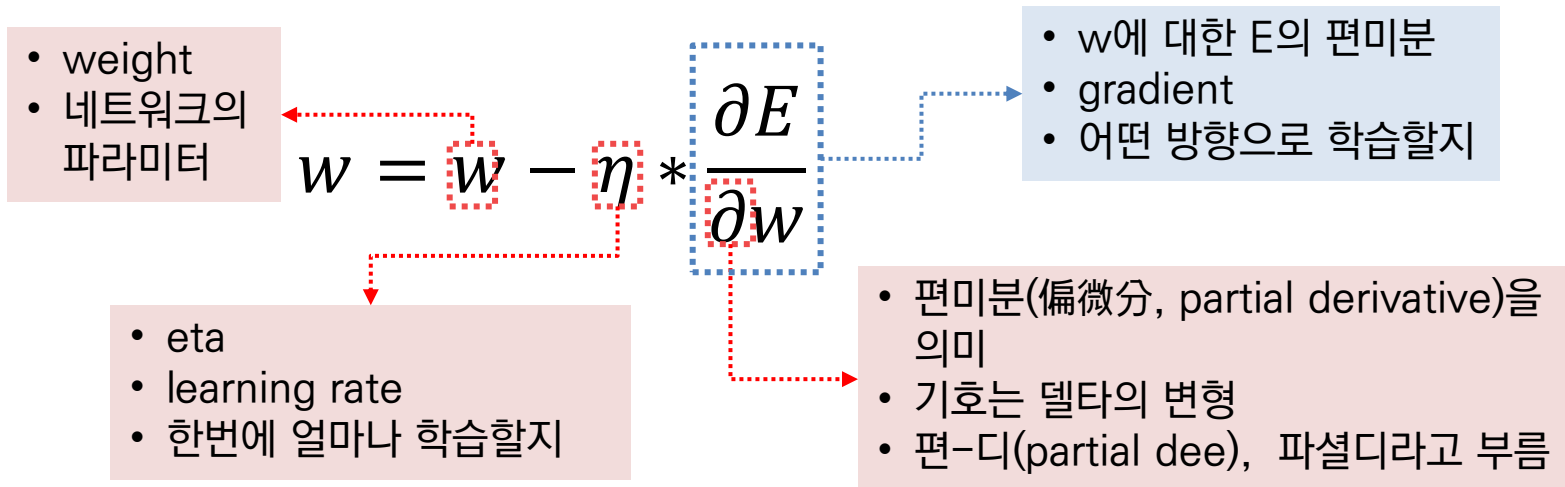


- 0보다 크면 B 지점
- 0보다 작으면 A 지점

딥러닝에서 Optimization은 학습 속도를 빠르고 안정적이게 하는 것

Stochastic gradient descent (확률적경사하강법, SGD)

- ▶ Support vector machine, Logistic regression 등 머신러닝에서 모델 훈련에 널리 사용되는 알고리즘
- ▶ 역 전파 알고리즘과 결합하면 인공 신경망을 훈련시키기위한 사실상의 표준 알고리즘



An overview of gradient descent optimization algorithms : <https://ruder.io/optimizing-gradient-descent/>

Momentum

2. 인공신경망 모델의 구조 및 동작원리

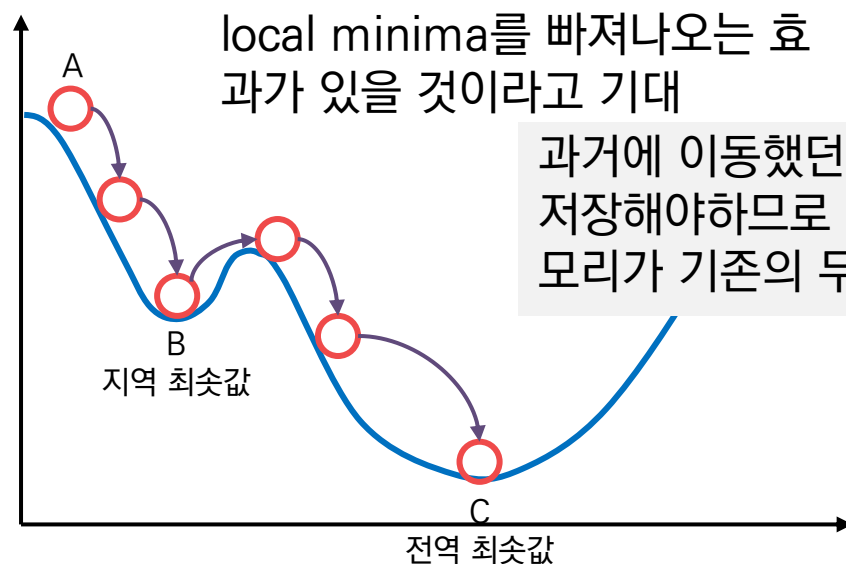


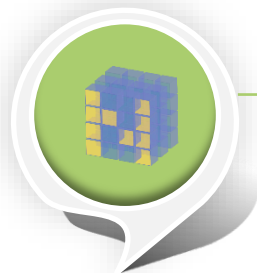
Momentum 방식

- ▶ Gradient Descent를 통해 이동하는 과정에 일종의 ‘관성’을 주는 것
- ▶ 현재 Gradient를 통해 이동하는 방향과는 별개로, 과거에 이동했던 방식을 기억하면서 그 방향으로 일정 정도를 추가적으로 이동하는 방식
- ▶ α 는 얼마나 momentum을 줄 것인지에 대한 momentum term으로서, 보통 0.9 정도의 값을 사용

$$w = w - \eta \frac{\partial E}{\partial w} + \alpha \Delta w$$

- α : 관성의 크기를 결정하는 모멘텀 상수
- Δw : 이전 회차의 수정량





NAG(Nesterov Accelerated Gradient)

- ▶ Nesterov Accelerated Gradient(NAG)는 Momentum 방식을 기초로 한 방식이지만, Gradient를 계산하는 방식이 조금 다름
- ▶ NAG에서는 momentum step을 먼저 고려하여, momentum step을 먼저 이동했다고 생각한 후 그 자리에서의 gradient를 구해서 gradient step을 이동함
- ▶ NAG를 이용할 경우 Momentum 방식에 비해 보다 효과적으로 이동할 수 있다.
- ▶ Momentum 방식의 경우 멈춰야 할 시점에서도 관성에 의해 훨씬 멀리 갈수도 있다는 단점이 존재하는 반면, NAG 방식의 경우 일단 모멘텀으로 이동을 반정도 한 후 어떤 방식으로 이동해야 할 지를 결정한다.
- ▶ 그러므로, Momentum 방식의 빠른 이동에 대한 이점은 누리면서도, 멈춰야 할 적절한 시점에서 제동을 거는 데에 훨씬 용이하다고 생각할 수 있음

Momentum

$$w = w - \eta \frac{\partial E}{\partial w} + \alpha \Delta w$$

NAG

$$w = w - \eta \frac{\partial E}{\partial w - \alpha \Delta w} + \alpha \Delta w$$

Adagrad (1/2)

2. 인공신경망 모델의 구조 및 동작원리



Adagrad(Adaptive Gradient)

- ▶ 변수들을 update할 때 각각의 변수마다 step size를 다르게 설정해서 이동하는 방식(2011년 John Duchi가 처음 제안한 알고리즘)
- ▶ 기본적인 아이디어는 ‘지금까지 많이 변화하지 않은 변수들은 step size를 크게 하고, 지금까지 많이 변화했던 변수들은 step size를 작게 하자’ 라는 것
- ▶ 자주 등장하거나 변화를 많이 한 변수들의 경우 optimum에 가까이 있을 확률이 높기 때문에 작은 크기로 이동하면서 세밀한 값을 조정하고, 적게 변화한 변수들은 optimum 값에 도달하기 위해서는 많이 이동해야 할 확률이 높기 때문에 먼저 빠르게 loss 값을 줄이는 방향으로 이동하려는 방식
- ▶ word2vec이나 GloVe 같이 word representation을 학습시킬 경우 단어의 등장 확률에 따라 variable의 사용 비율이 확연하게 차이 나기 때문에 Adagrad와 같은 학습 방식을 이용하면 훨씬 더 좋은 성능을 거둘 수 있음

Adagrad (2/2)

2. 인공신경망 모델의 구조 및 동작원리

$$h = h + \left(\frac{\partial E}{\partial w} \right)^2 \qquad w = w - \eta \frac{1}{\sqrt{h}} \frac{\partial E}{\partial w}$$

h 는 반드시 증가하고 w 에서 h 는 분모에 있기때문에 수정량은 반드시 감소함.

총 수정량이 적은 가중치는 새로운 수정량이 커짐.

이를 통해 처음에는 넓은 영역에서 탐색하여 점차 범위를 좁혀가는 효율성을 보임.

- 장점

효율성이 좋고, AdaGrad에서 설정해야하는 상수는 η 밖에 없으므로 상수 조정 번거로움이 없음.

- 단점

계속되는 수정량 감소로 도중에 수정량이 0이 되어버려 최적화가 더이상 진행되지 않는 단점이 존재.

RMSProp

2. 인공지능망 모델의 구조 및 동작원리



RMSProp

- ▶ Geoffrey Hinton이 코세라 강의에서 제안
- ▶ AdaGrad에서 수정량 감소를 통해 학습이 정체되는 단점을 극복한 방법

AdaGrad는 간단한 convex function에서 잘 동작하지만, 복잡한 다차원 곡면의 function에서는 global minimum에 도달하기 전에 학습률이 0에 수렴할 수 있기 때문에 RMSProp에서 이를 보완

$$h = \beta h + (1 - \beta) \left(\frac{\partial E}{\partial w} \right)^2$$

$$w = w - \eta * \frac{1}{\sqrt{h}} * \frac{\partial E}{\partial w}$$

- AdaGrad의 h에 hyper parameter β 가 추가됨
- β 가 작을수록 가장 최신의 기울기를 더 크게 반영
- β : h를 구하는 식에 β 가 존재함으로써 이전 시점의 h를 적당한 비율로 감소시킴
- Geoffrey Hinton은 β 값으로 0.9를 추천

AdaDelta

2. 인공신경망 모델의 구조 및 동작원리



AdaDelta(Adaptive Delta)

- ▶ AdaDelta는 RMSProp과 유사하게 AdaGrad의 단점을 보완하기 위해 제안된 방법
- ▶ <https://www.matthewzeiler.com/mattzeiler/adadelta.pdf>
- ▶ AdaDelta는 RMSProp과 동일하게 h를 구할 때 합을 구하는 대신 지수평균을 구한다. 다만, 여기에서는 step size를 단순히 η 로 사용하는 대신 step size의 변화값의 제곱을 가지고 지수평균 값을 사용

RMSProp

$$h = \beta h + (1 - \beta) \left(\frac{\partial E}{\partial w} \right)^2$$

$$w = w - \eta \frac{1}{\sqrt{h}} \frac{\partial E}{\partial w}$$

$$h = h + \left(\frac{\partial E}{\partial w} \right)^2$$

$$s = \beta s - (1 - \beta) \left(\frac{\partial E}{\partial w} \right)^2$$

$$w = w - \eta \frac{\sqrt{s}}{\sqrt{h}} \frac{\partial E}{\partial w}$$

Adam (1/2)

2. 인공신경망 모델의 구조 및 동작원리



Adam(Adaptive Moment estimation)

- ▶ 2014년 ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION(Diederik P.kingma, Jimmy Lei Ba)에서 제안
- ▶ <https://arxiv.org/abs/1412.6980>
- ▶ RMSProp과 Momentum 방식을 합친 것 같은 알고리즘
- ▶ Momentum 방식과 유사하게 지금까지 계산해온 기울기의 지수평균을 저장하며, RMSProp과 유사하게 기울기의 제곱값의 지수평균을 저장

Adam (2/2)

2. 인공신경망 모델의 구조 및 동작원리

Momentum과 Adagrad는 각각 v 와 h 가 처음에 0으로 초기화되면 W 가 학습 초반에 0으로 biased되는 문제가 있음

Momentum

$$w = w - \eta \frac{\partial E}{\partial w} + \alpha \Delta w$$

RMSProp

$$h = \beta h + (1 - \beta) \left(\frac{\partial E}{\partial w} \right)^2$$

$$w = w - \eta * \frac{1}{\sqrt{h}} * \frac{\partial E}{\partial w}$$

그래서 unbiased 하게 만들어주는 작업을 거친다.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \left(\frac{\partial E}{\partial w} \right)^2$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left(\frac{\partial E}{\partial w} \right)^2$$

$$\hat{m}_t = \frac{m_t}{\sqrt{1 - \beta_1^t}}$$

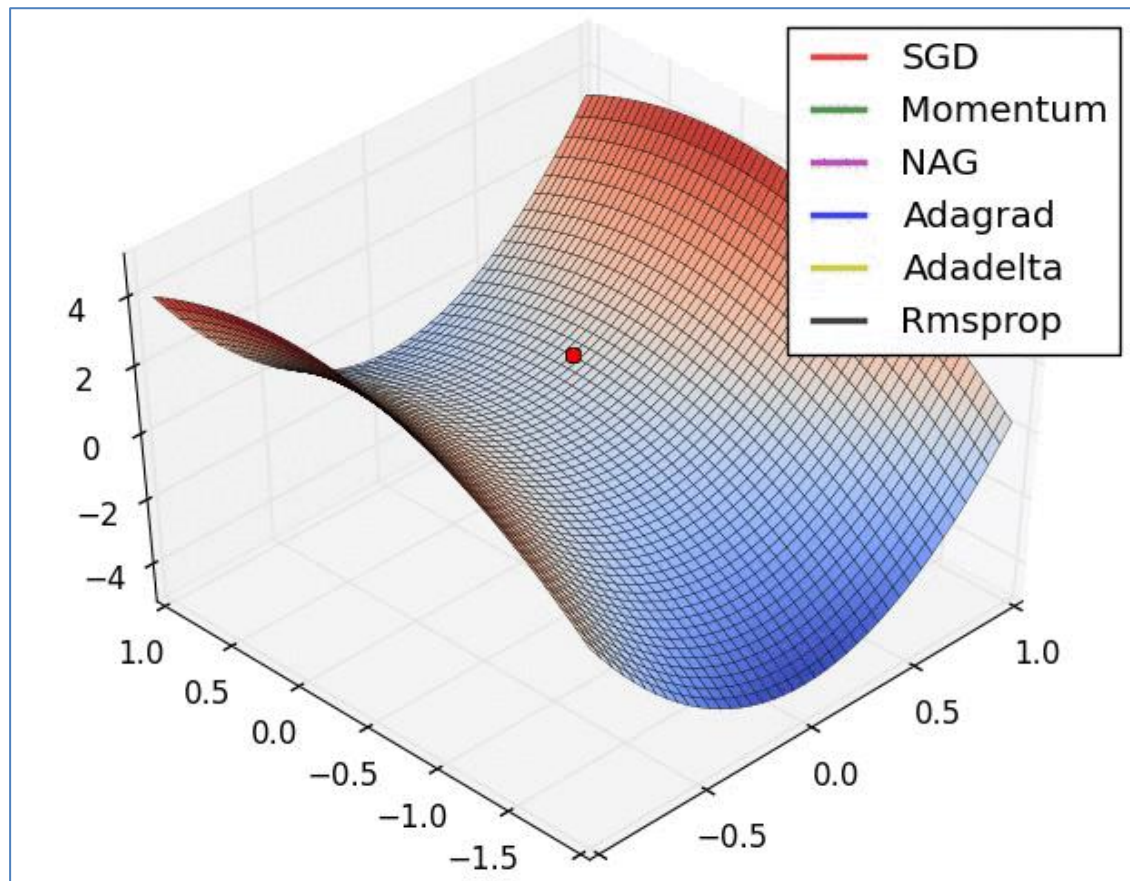
$$\hat{v}_t = \frac{v_t}{\sqrt{1 - \beta_2^t}}$$

$$w = w - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} * \hat{m}_t$$

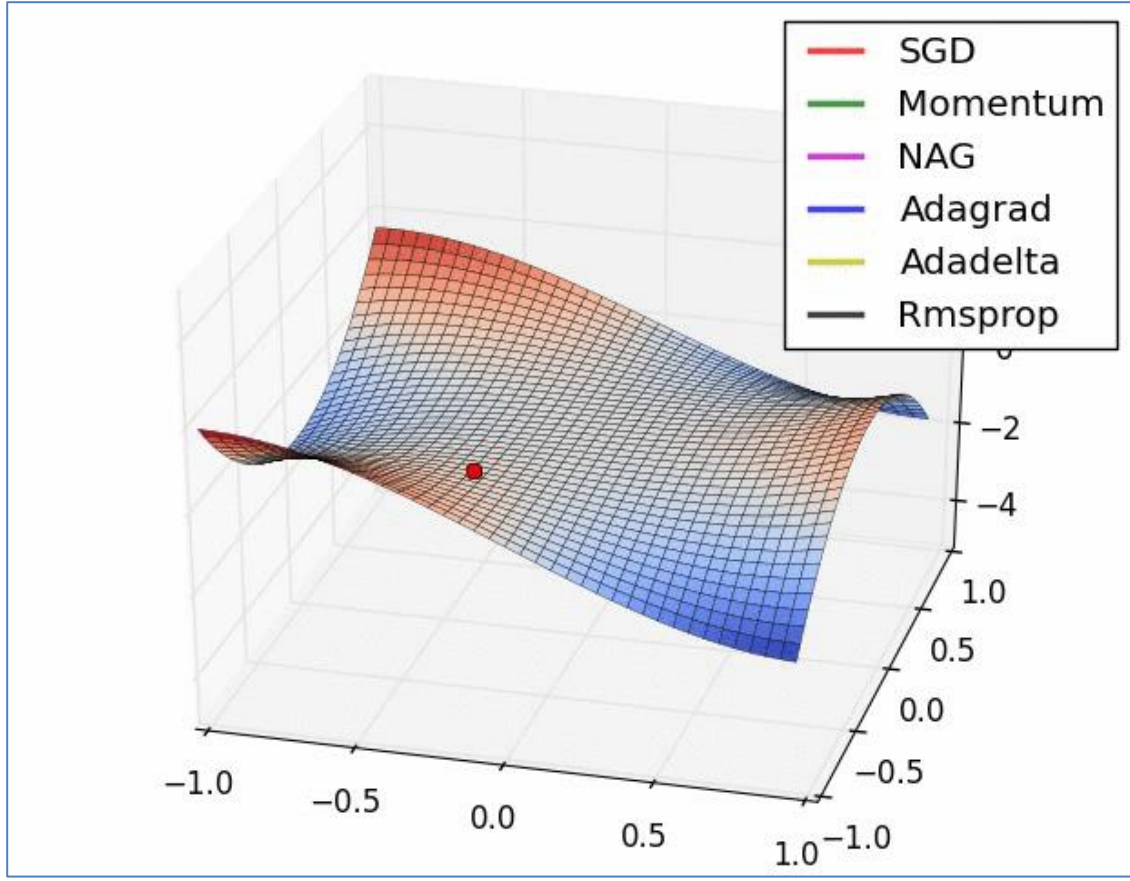
옵티마이저 비교

2. 인공신경망 모델의 구조 및 동작원리

Gradient Descent Optimization Algorithms at Long Valley



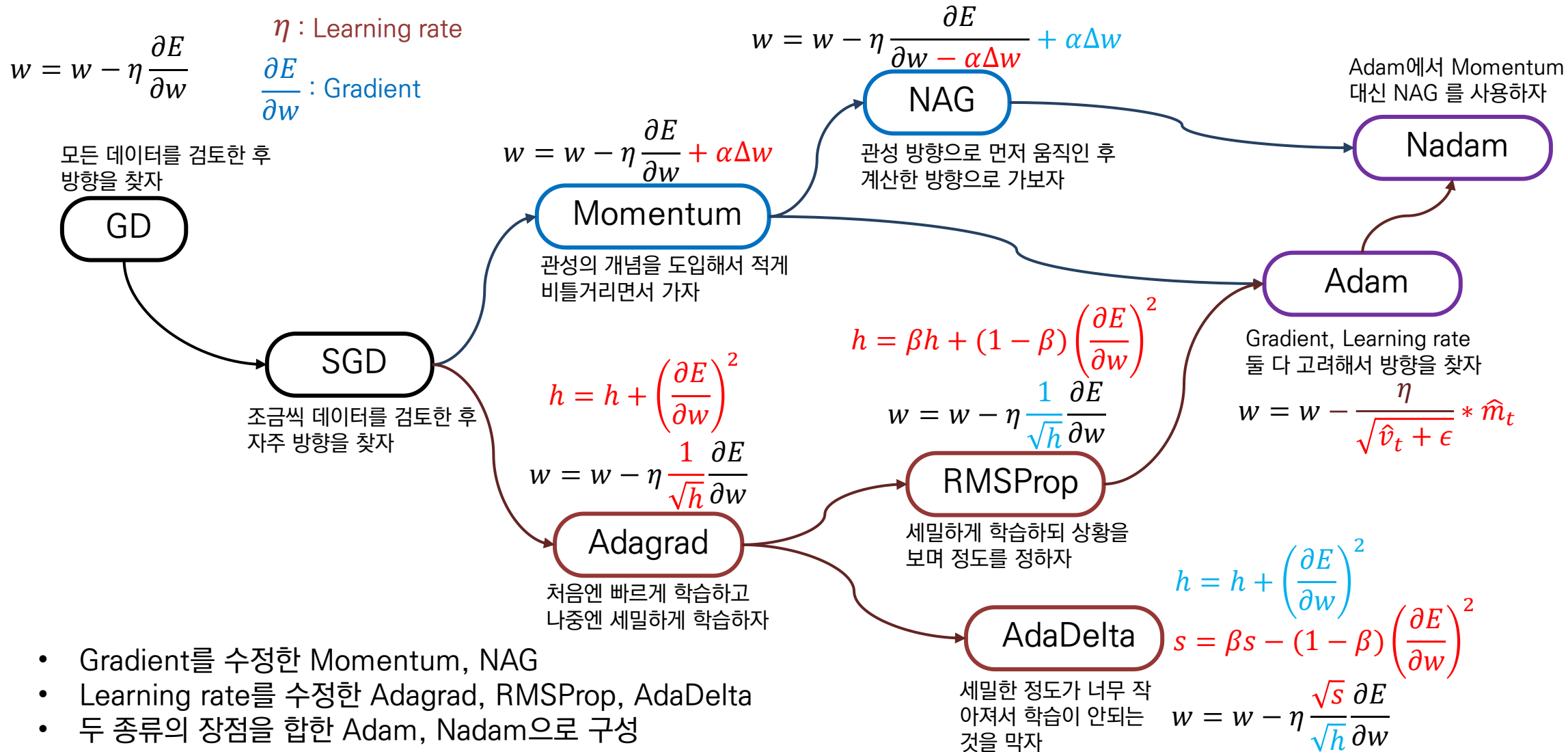
Gradient Descent Optimization Algorithms at Saddle Point



출처 : <https://ruder.io/optimizing-gradient-descent/>

옵티마이저 발전 과정

2. 인공신경망 모델의 구조 및 동작원리



- Gradient를 수정한 Momentum, NAG
- Learning rate를 수정한 Adagrad, RMSProp, AdaDelta
- 두 종류의 장점을 합한 Adam, Nadam으로 구성

케라스의 옵티마이저


2. 인공신경망 모델의 구조 및 동작원리

옵티마이저	클래스(기본 파라미터)	설명
SGD	<code>tf.keras.optimizers.SGD(lr=0.01, momentum=0.0, decay=0.0, nesterov=False)</code>	Stochastic gradient descent optimizer.
RMSprop	<code>tf.keras.optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=None, decay=0.0)</code>	RMSProp optimizer.
Adagrad	<code>tf.keras.optimizers.Adagrad(lr=0.01, epsilon=None, decay=0.0)</code>	Adagrad optimizer.
Adadelta	<code>tf.keras.optimizers.Adadelta(lr=1.0, rho=0.95, epsilon=None, decay=0.0)</code>	Adadelta optimizer.
Adam	<code>tf.keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)</code>	Adam optimizer.
Adamax	<code>tf.keras.optimizers.Adamax(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0)</code>	Adamax optimizer from Adam paper's Section 7.
Nadam	<code>tf.keras.optimizers.Nadam(lr=0.002, beta_1=0.9, beta_2=0.999, epsilon=None, schedule_decay=0.004)</code>	Nesterov Adam optimizer.

케라스의 옵티마이저 설정 예

2. 인공신경망 모델의 구조 및 동작원리

 compile() 함수의 optimizer 매개변수를 이용해서 설정

 문자열을 이용하거나 클래스를 이용할 수 있음

```
1.model.compile(loss='sparse_categorical_crossentropy',  
2.               optimizer='adam')
```

```
1.import tensorflow  
2.model.compile(loss='sparse_categorical_crossentropy',  
3.               optimizer=tensorflow.keras.optimizers.Adam())
```

3. DNN 인공신경망 모델 구현하기

DNN 모델 구현



텐서플로우 케라스

3. DNN 인공지능망 모델 구현하기



텐서플로우 케라스

- ▶ 텐서플로우의 케라스 API는 딥러닝 모델을 구축하고 훈련하기 위한 고수준 API로, 다양한 신경망 구조를 쉽게 구축할 수 있도록 설계되었음
- ▶ 텐서플로우 2.x이후부터 공식적으로 케라스를 포함함

- `tf.keras.Sequential`: 순차 모델은 여러 층을 연이어 쌓아서 신경망을 구축하는 데 사용됨. 각 층은 이전 층의 출력을 입력으로 받음
- `tf.keras.Model`: 사용자 정의 모델을 만들 때 이 클래스를 상속하여 모델을 정의할 수 있습니다. 이를 사용하면 복잡한 모델 구조를 구현할 수 있음.
- `tf.keras.layers`: 이 모듈은 다양한 종류의 층을 포함하고 있으며, 이를 사용하여 모델의 구조를 정의함. 예를 들어, **Dense** 층은 완전 연결된 층을 생성하고, **Conv2D** 층은 합성곱 신경망을 생성함
- `tf.keras.losses`: 손실 함수를 정의하는 데 사용. 모델의 훈련 중에 사용되며, 훈련 중에 최소화하고자 하는 목표 함수를 정의함. 예를 들어, `categorical_crossentropy`는 다중 클래스 분류 문제의 크로스 엔트로피 손실 함수임
- `tf.keras.optimizers`: 최적화 알고리즘을 선택하고 모델을 훈련하기 위해 사용. 예를 들어, Adam, SGD, RMSprop와 같은 최적화 알고리즘을 제공함.
- `tf.keras.metrics`: 모델의 성능을 평가하기 위한 메트릭을 정의하는 데 사용. 예를 들어, 정확도(**accuracy**)나 정밀도(**precision**)를 계산할 수 있음
- `tf.keras.callbacks`: 모델 훈련 중에 호출되는 콜백 함수를 정의하는 데 사용. 예를 들어, 학습률을 동적으로 조정하거나 모델 저장을 자동화하는 데 유용함
- `tf.keras.utils`: 데이터 전처리 및 유틸리티 함수를 제공하는 모듈. 데이터 로드, 정규화, 원-핫 인코딩 등의 작업을 처리할 때 유용

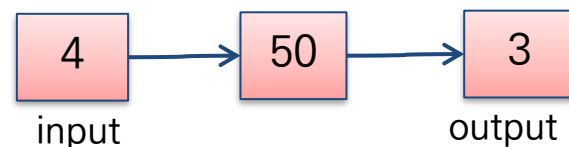
Sequential 클래스 이용

3. DNN 인공신경망 모델 구현하기



예

```
1. from tensorflow.keras import Sequential
2. from tensorflow.keras.layers import Input, Dense
3.
4. model = Sequential()
5. model.add(Input(shape=(4,)))
6. model.add(Dense(50, activation='relu'))
7. model.add(Dense(3, activation='softmax'))
8. model.summary()
```



Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 50)	250
dense_1 (Dense)	(None, 3)	153
Total params: 403 (1.57 KB)		
Trainable params: 403 (1.57 KB)		
Non-trainable params: 0 (0.00 Byte)		



장점

- Sequential API로 모델을 설계하는 것은 직관적이고 편리함



단점

- 단순히 층을 쌓는 것만으로는 모든 인공신경망을 구현할 수 없음.
즉, 복잡한 인공 신경망을 구현할 수 없음.

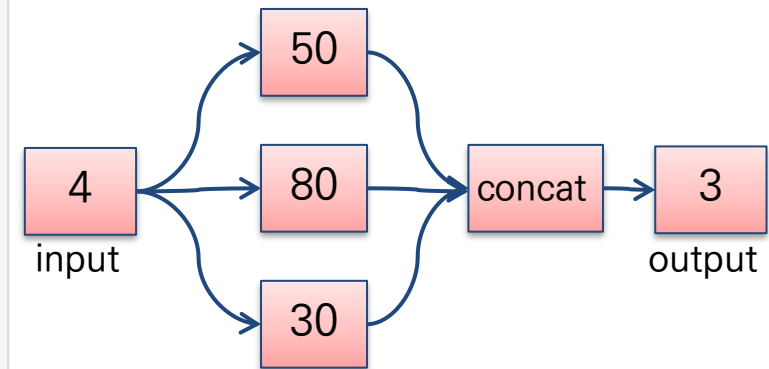
함수형 API 이용

3. DNN 인공신경망 모델 구현하기

✓ 함수형 API를 이용하면 다중 입력 및 다중 출력 모델 생성 가능

```
1. from tensorflow.keras import Model
2. from tensorflow.keras.layers import Input, Dense
3. from tensorflow.keras.layers import concatenate, Activation
4.
5. input = Input(shape=(4,))
6. dense1 = Dense(50, activation='relu')(input)
7. dense2 = Dense(80, activation='relu')(input)
8. dense3 = Dense(30, activation='relu')(input)
9. x = concatenate([dense1, dense2, dense3])
10. output = Dense(3, activation='softmax')(x)
11. model = Model(inputs=input, outputs=output)
12. model.summary()
```

`x = Dense(64)(input)`
위의 코드는 아래의 코드와 같음
`x = Dense(64)`
`x(input)`



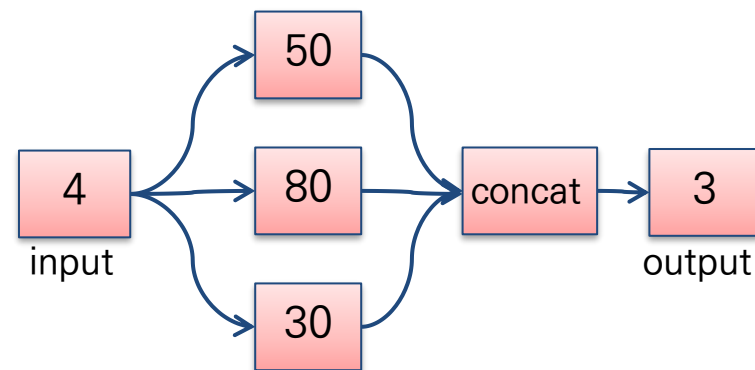
함수형 API 이용

3. DNN 인공신경망 모델 구현하기

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	[(None, 4)]	0	[]
dense_2 (Dense)	(None, 50)	250	['input_2[0][0]']
dense_3 (Dense)	(None, 80)	400	['input_2[0][0]']
dense_4 (Dense)	(None, 30)	150	['input_2[0][0]']
concatenate (Concatenate)	(None, 160)	0	['dense_2[0][0]', 'dense_3[0][0]', 'dense_4[0][0]']
dense_5 (Dense)	(None, 3)	483	['concatenate[0][0]']

=====
Total params: 1283 (5.01 KB)
Trainable params: 1283 (5.01 KB)
Non-trainable params: 0 (0.00 Byte)



3. DNN 인공신경망 모델 구현하기

▶ **붓꽃의 3가지 종(setosa(세토사), versicolor(버시컬러), virginica(버지니카))별로 각각 50개씩 꽃받침과 꽃잎의 길이와 너비를 센티미터 단위로 측정하여 정리한 150개 데이터셋**

```
{'data': array([[5.1, 3.5, 1.4, 0.2],  
               [4.9, 3. , 1.4, 0.2],  
               [4.7, 3.2, 1.3, 0.2],  
               [4.6, 3.1, 1.5, 0.2],  
               [5. , 3.6, 1.4, 0.2],  
               ...  
             ], dtype=float),  
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
                  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
                  0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
                  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
                  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2,  
                  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,  
                  2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]),  
 'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'),  
 'DESCR': 'Iris Plants Database...'  
 ...  
 'feature_names': ['sepal length (cm)',  
                   'sepal width (cm)',  
                   'petal length (cm)',  
                   'petal width (cm)']}]}
```

Diagram illustrating the three Iris species and their corresponding parts:

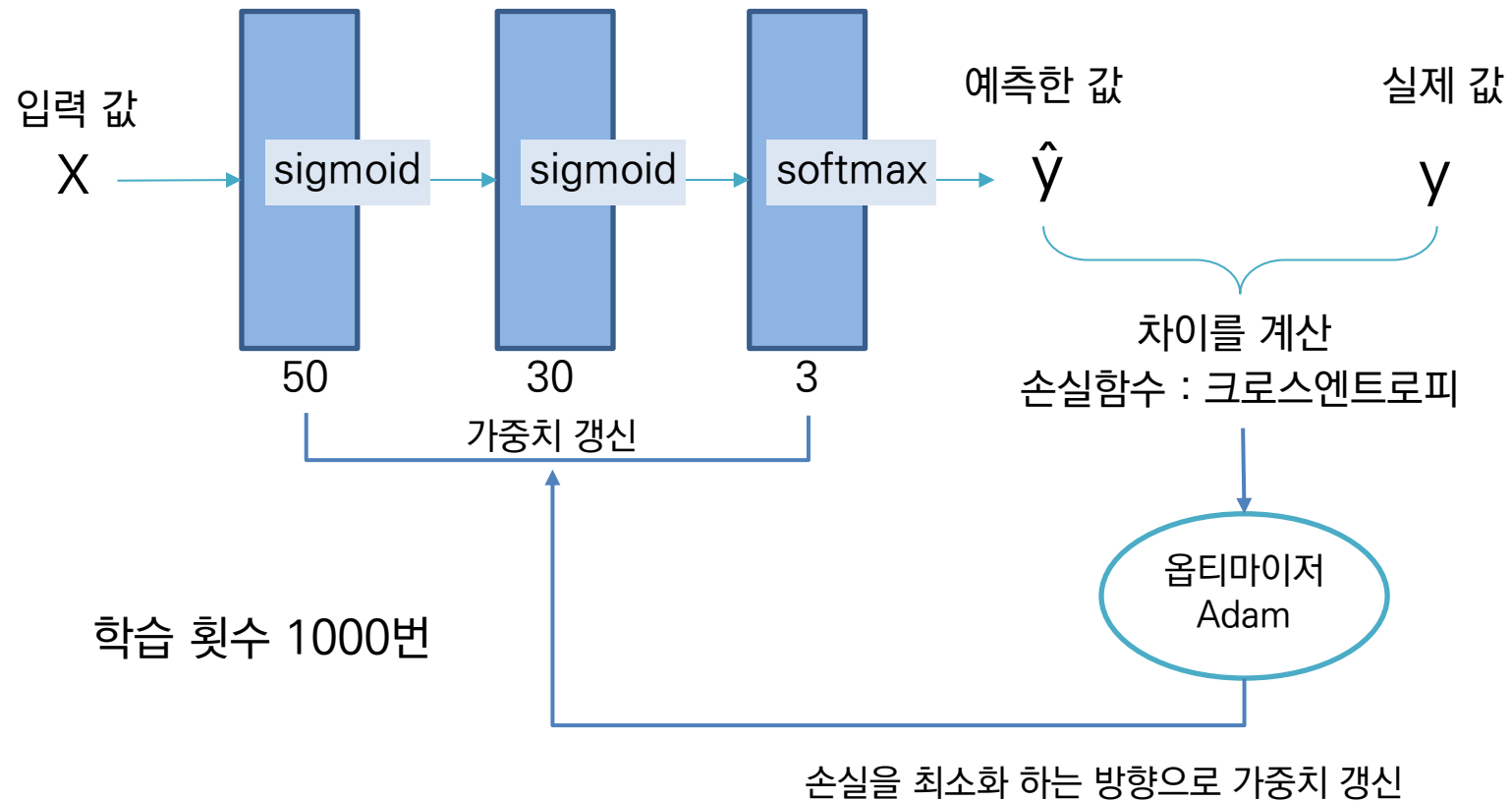
- setosa**
- versicolor**
- virginica**

Labels for the parts of the flower:

- Sepal(꽃받침)**
- Petal(꽃잎)**

구현할 인공신경망 모델의 구조

3. DNN 인공신경망 모델 구현하기



텐서플로우(케라스)를 이용한 iris 분류 모델

3. DNN 인공신경망 모델 구현하기

케라스 인공신경망 모델의 구조와 활성화 함수 설정

```
1. from tensorflow.keras import Sequential
2. from tensorflow.keras.layers import Dense, Input
3.
4. model = Sequential()
5. model.add(Input(shape=(4,)))
6. model.add(Dense(50, activation="sigmoid"))
7. model.add(Dense(30, activation="sigmoid"))
8. model.add(Dense(3, activation="softmax"))
9. model.summary()
```

텐서플로우(케라스)를 이용한 iris 분류 모형 - 훈련 정의, 데이터 로드

3. DNN 인공신경망 모델 구현하기

훈련을 정의('compile 한다'라고 표현함) : 옵티마이저, 손실함수, 평가방법을 지정

```
1. model.compile(loss='sparse_categorical_crossentropy',  
2.                optimizer='adam', metrics=['accuracy'])
```

학습용 데이터셋 불러오고, 7:3의 비율로 train 데이터와 test 데이터로 나눔

```
from sklearn import datasets  
iris = datasets.load_iris()  
X, y = iris.data, iris.target
```



Iris.data는 독립변수(꽃받침의 길이와 폭, 꽃잎의 길이와 폭)을 가지고 있으며, iris.target은 종(species) 정보를 가지고 있습니다.

```
from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,  
                                                    random_state=1)
```



X_train, y_train은 학습에 사용하며, X_test와 y_test는 모델을 평가하는데 사용합니다.

모델 학습 및 평가

3. DNN 인공신경망 모델 구현하기

학습시킴 : epochs는 학습횟수를 지정

```
1. model.fit(X_train, y_train, epochs=200)
```

```
Epoch 1/200  
4/4 [=====] - 1s 9ms/step - loss: 1.0811 - accuracy: 0.3524  
Epoch 2/200  
4/4 [=====] - 0s 6ms/step - loss: 1.0669 - accuracy: 0.3524  
Epoch 3/200  
4/4 [=====] - 0s 5ms/step - loss: 1.0557 - accuracy: 0.5905  
...생략...
```

evaluate()함수를 이용해서 쉽게 모델을 평가할 수 있음

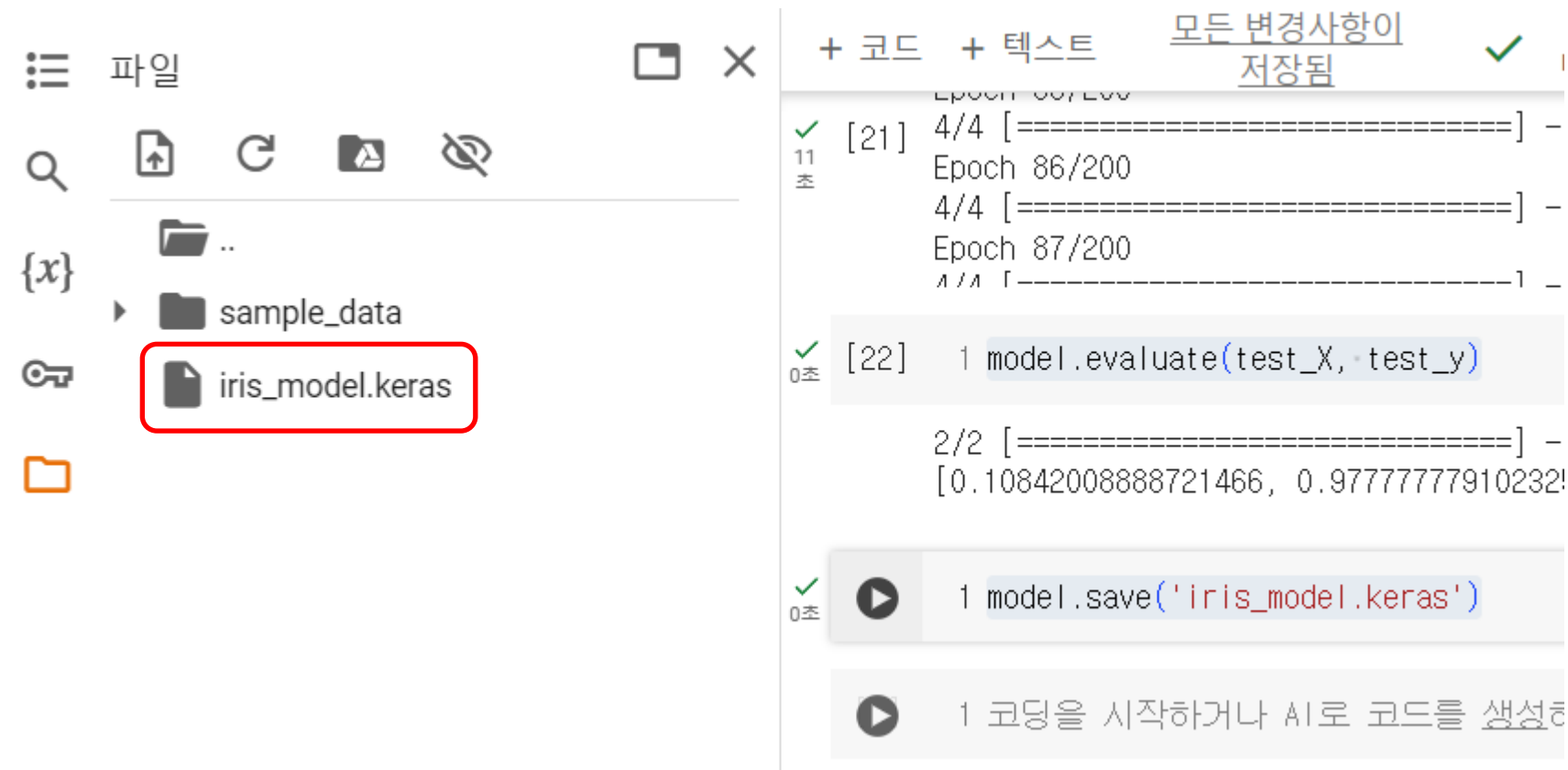
```
1. model.evaluate(X_test, y_test)
```

```
2/2 [=====] - 0s 14ms/step - loss: 0.1084 - accuracy: 0.9778  
[0.10842008888721466, 0.9777777791023254]
```

모델 저장하기

3. DNN 인공신경망 모델 구현하기

1. `model.save('iris_model.keras')`



The screenshot shows a Jupyter Notebook interface. On the left, the file explorer displays a folder named 'sample_data' and a file named 'iris_model.keras', which is highlighted with a red rectangle. The main area shows the notebook's code cells. The top cell contains training progress output for epochs 86 and 87. The second cell contains the code `model.evaluate(test_X, test_y)` and its output. The third cell contains the code `model.save('iris_model.keras')` and is highlighted with a red rectangle. The bottom cell contains the text '1 코딩을 시작하거나 시로 코드를 생성'.

모든 변경사항이 저장됨

11초

[21] 4/4 [=====] -
Epoch 86/200
4/4 [=====] -
Epoch 87/200
1/1 [=====] -

0초

[22] 1 model.evaluate(test_X, test_y)

2/2 [=====] -
[0.10842008888721466, 0.97777777910232]

0초

1 model.save('iris_model.keras')

1 코딩을 시작하거나 시로 코드를 생성



모델이 크고 학습시간이 오래 걸릴 경우 학습한 모델을 저장하고 불러올 수 있어야 합니다.

모델 불러오기

3. DNN 인공신경망 모델 구현하기

```
1. from tensorflow.keras.models import load_model
2. model = load_model('/content/iris_model.keras')
```

```
1. import numpy as np
2. pred = np.argmax(model.predict(test_X), axis=1)
3. np.mean(np.equal(test_y, pred))
```

```
2/2 [=====] - 0s 9ms/step
0.9777777777777777
```

모델의 구조 저장하기

3. DNN 인공신경망 모델 구현하기

 to_json() 함수를 이용하면 모델의 구조를 JSON 문자열로 변환

```
1. import json
2. json_str = model.to_json()
3. json_dic = json.loads(json_str)
4.
5. with open('iris_model.json', 'w') as f:
6.     json.dump(json_dic, f, indent=4)
```



필요하다면 json 파일을 열어
모델의 구조를 수정할 수 있음

- 기존에 작성했던 코드의 반복을 줄일 수 있음

```
{
  "class_name": "Sequential",
  "config": {
    "name": "sequential_1",
    "layers": [
      {
        "module": "keras.layers",
        "class_name": "InputLayer",
        "config": {
          "batch_input_shape": [
            null,
            4
          ],
          "dtype": "float32",
          "sparse": false,
          "ragged": false,
          "name": "input_3"
        },
        "registered_name": null
      },
      {
        "module": "keras.layers",
        "class_name": "Dense",
        "config": {
          "name": "dense_6",
          "trainable": true,
          "dtype": "float32",
          "units": 50,
          "activation": "sigmoid",
          "use_bias": true,

```

저장한 모델의 구조 불러오기

3. DNN 인공신경망 모델 구현하기

model_from_json은 JSON 문자열을 이용해서 케라스 모델을 생성함

```
1. with open('iris_model.json', 'r') as f:  
2.     model_json = f.read()
```

```
1. from tensorflow.keras.models import model_from_json  
2. model = model_from_json(model_json)  
3. model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_6 (Dense)	(None, 50)	250
dense_7 (Dense)	(None, 30)	1530
dense_8 (Dense)	(None, 3)	93

=====
Total params: 1873 (7.32 KB)
Trainable params: 1873 (7.32 KB)
Non-trainable params: 0 (0.00 Byte)
=====

모델의 구조만 불러와 학습시키기

3. DNN 인공신경망 모델 구현하기

새로 불러온 모델을 `fit()` 함수를 사용하여 학습하거나 `evaluate()` 함수를 사용하여 테스트하려면, 먼저 `compile()` 함수를 실행해야 함

이 단계를 통해 모델에 손실 함수, 옵티마이저 및 평가 방법을 설정할 수 있음

```
1. model.compile(loss='sparse_categorical_crossentropy',  
2.               optimizer='adam', metrics=['accuracy'])
```

```
1. model.fit(train_X, train_y, epochs=200, verbose=0)
```

```
1. model.evaluate(test_X, test_y)
```

참고: 케라스와 파이토치

딥러닝 프레임워크



케라스와 파이토치

참고: 케라스와 파이토치



Keras

- 높은 수준의 API를 제공함
- 코드가 직관적이고 사용하기 쉬움
- TensorFlow의 고수준 API로 통합되었음
- TensorFlow 생태계를 잘 활용할 수 있음



PyTorch

- 낮은 수준의 API를 제공
- 더 유연하고 커스터마이징하기 좋음
- 연구 커뮤니티에서 강력한 지지를 받고 있음
- 연구 결과를 빠르게 구현할 수 있는 라이브러리나 도구가 많음

기능	케라스	파이토치
모델 정의	Sequential 또는 Functional API 사용	nn.Module 클래스 상속
레이어 추가	model.add 또는 리스트 내 정의	__init__ 메서드 내에 레이어 정의
순전파	자동으로 처리	forward 메서드 내에서 직접 정의
모델 컴파일	model.compile 사용	없음, 직접 손으로 해야 함
모델 훈련	model.fit 사용	훈련 루프 직접 작성
손실 함수	loss 인자로 지정	nn.CrossEntropyLoss 등 손실 함수를 직접 정의
최적화 알고리즘	optimizer 인자로 지정	optim.Adam 등 옵티마이저를 직접 정의



pytorch_lightning을 사용하면 fit(), eval() 등을 사용할 수 있습니다.

Keras vs. PyTorch

참고: 케라스와 파이토치

기능	TensorFlow/Keras	PyTorch
기본 라이브러리	tensorflow	torch
기본 뉴럴 네트워크 모듈	tensorflow.keras	torch.nn
순차 모델	tensorflow.keras.Sequential	torch.nn.Sequential
복잡한 모델	tensorflow.keras.Model	torch.nn.Module
입력 데이터 정의	tensorflow.keras.Input	torch.Tensor
기본 레이어 클래스	tensorflow.keras.layers.Layer	torch.nn.Module
완전 연결층	tensorflow.keras.layers.Dense	torch.nn.Linear
2D 합성곱 층	tensorflow.keras.layers.Conv2D	torch.nn.Conv2d
1D 최대 풀링	tensorflow.keras.layers.MaxPooling1D	torch.nn.MaxPool1d
2D 최대 풀링	tensorflow.keras.layers.MaxPooling2D	torch.nn.MaxPool2d
2D 역합성곱 층	tensorflow.keras.layers.Conv2DTranspose	torch.nn.ConvTranspose2d
순환 신경망	tensorflow.keras.layers.RNN	torch.nn.RNN
장단기 메모리	tensorflow.keras.layers.LSTM	torch.nn.LSTM
게이트 순환 유닛	tensorflow.keras.layers.GRU	torch.nn.GRU
양방향 RNN	tensorflow.keras.layers.Bidirectional	torch.nn.Bidirectional

내장된 학습용 데이터셋

참고: 케라스와 파이토치

단계	Keras
라이브러리 импорт	<code>from tensorflow.keras.datasets import mnist</code>
데이터셋 로드	<code>(x_train, y_train), (x_test, y_test) = mnist.load_data()</code>
데이터셋 전처리	<code>X_train = x_train / 255.0</code> <code>X_test = x_test / 255.0</code>

단계	PyTorch
라이브러리 импорт	<code>import torch</code> <code>from torch.utils.data import DataLoader</code> <code>from torchvision import datasets, transforms</code>
데이터셋 로드	<code>transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.1307,), (0.3081,))])</code> <code>train_dataset = datasets.MNIST(root='./data', train=True, download=True, transform=transform)</code> <code>test_dataset = datasets.MNIST(root='./data', train=False, download=True, transform=transform)</code>
데이터셋 전처리	<code>train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)</code> <code>test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)</code>



MNIST 데이터셋의 픽셀 값은 0부터 255까지의 값을 가지며, 이를 0에서 1 사이로 스케일링한 후 평균과 표준편차를 계산하면 대략 다음과 같습니다:

- 평균(mean): 약 0.1307
- 표준편차(std): 약 0.3081

$$output = \frac{input - mean}{std}$$

Sequential

참고: 케라스와 파이토치

Keras 예시	PyTorch 예시
<code>from tensorflow.keras import Sequential</code>	<code>import torch</code>
<code>from tensorflow.keras import layers</code>	<code>import torch.nn as nn</code>
<code>model = Sequential([</code>	<code>model = nn.Sequential(</code>
<code> layer.Input(shape=(784,)),</code>	
<code> layers.Dense(128, activation='relu'),</code>	<code> nn.Linear(784, 128),</code>
	<code> nn.ReLU(),</code>
<code> layers.Dropout(0.5),</code>	<code> nn.Dropout(0.5),</code>
<code> layers.Dense(10, activation='softmax')</code>	<code> nn.Linear(128, 10),</code>
	<code> nn.Softmax(dim=1)</code>
<code>)</code>	<code>)</code>
<code>model.summary()</code>	<code>from torchsummary import summary</code> <code>summary(model, input_size=(1, 784))</code>

서브클래싱

참고: 케라스와 파이토치

Keras 예시	PyTorch 예시
<pre>from tensorflow.keras import Model</pre>	<pre>import torch</pre>
<pre>from tensorflow.keras.layers import Dense, Dropout</pre>	<pre>import torch.nn as nn</pre>
<pre>class MyModel(Model):</pre>	<pre>class MyModel(nn.Module):</pre>
<pre> def init(self):</pre>	<pre> def init(self):</pre>
<pre> super(MyModel, self).init()</pre>	<pre> super(MyModel, self).init()</pre>
<pre> self.dense1 = Dense(128, activation='relu')</pre>	<pre> self.fc1 = nn.Linear(784, 128)</pre>
<pre> self.dropout = Dropout(0.5)</pre>	<pre> self.dropout = nn.Dropout(0.5)</pre>
<pre> self.dense2 = Dense(10, activation='softmax')</pre>	<pre> self.fc2 = nn.Linear(128, 10)</pre>
<pre> def call(self, inputs):</pre>	<pre> def forward(self, x):</pre>
<pre> x = self.dense1(inputs)</pre>	<pre> x = torch.relu(self.fc1(x))</pre>
<pre> x = self.dropout(x)</pre>	<pre> x = self.dropout(x)</pre>
<pre> return self.dense2(x)</pre>	<pre> return torch.softmax(self.fc2(x), dim=1)</pre>
<pre>model = MyModel()</pre>	<pre>model = MyModel()</pre>

활성화 함수

참고: 케라스와 파이토치

기능	Keras	PyTorch
ReLU	activation='relu' 또는 tf.keras.layers.ReLU	torch.nn.functional.relu 또는 nn.ReLU
Leaky ReLU	tf.keras.layers.LeakyReLU	torch.nn.functional.leaky_relu 또는 nn.LeakyReLU
ELU	tf.keras.layers.ELU	torch.nn.functional.elu 또는 nn.ELU
SELU	tf.keras.layers.SELU	torch.nn.functional.selu 또는 nn.SELU
Sigmoid	activation='sigmoid' 또는 tf.keras.activations.sigmoid	torch.nn.functional.sigmoid 또는 nn.Sigmoid
Tanh	activation='tanh' 또는 tf.keras.activations.tanh	torch.nn.functional.tanh 또는 nn.Tanh
Softmax	activation='softmax' 또는 tf.keras.activations.softmax	torch.nn.functional.softmax 또는 nn.Softmax

PyTorch의 손실함수

참고: 케라스와 파이토치

손실 함수	설명	사용 예시
<code>nn.CrossEntropyLoss</code>	분류 문제에 사용되는 교차 엔트로피 손실 함수	다중 클래스 분류 문제
<code>nn.MSELoss</code>	회귀 문제에 사용되는 평균 제곱 오차 손실 함수	회귀 문제
<code>nn.BCELoss</code>	이진 분류 문제에 사용되는 이진 교차 엔트로피 손실 함수	이진 분류 문제
<code>nn.BCEWithLogitsLoss</code>	이진 분류 문제에 사용되는 로짓을 포함한 이진 교차 엔트로피 손실 함수	이진 분류 문제 (출력에 시그모이드 함수 적용이 필요 없음)
<code>nn.NLLLoss</code>	네거티브 로그 가능도 손실 함수	다중 클래스 분류 문제 (출력에 <code>log_softmax</code> 함수 적용)
<code>nn.L1Loss</code>	L1 손실 함수 (MAE)	회귀 문제 (로버스트 회귀)
<code>nn.SmoothL1Loss</code>	허브 손실 함수	회귀 문제 (MAE와 MSE 사이의 손실 함수)
<code>nn.MarginRankingLoss</code>	마진 랭킹 손실 함수	순위 학습 문제
<code>nn.HingeEmbeddingLoss</code>	힌지 임베딩 손실 함수	분류 문제 (서포트 벡터 머신과 유사한 목적)
<code>nn.KLDivLoss</code>	Kullback-Leibler 발산 손실 함수	분포 간의 차이를 측정 (지식 증류 등)

PyTorch의 옵티마이저

참고: 케라스와 파이토치

옵티마이저	설명	사용 예시
optim.SGD	확률적 경사 하강법 (SGD) 옵티마이저	일반적인 경사 하강법을 사용할 때
optim.Adam	어댑티브 모멘트 추정 (Adam) 옵티마이저	대부분의 딥러닝 모델에 일반적으로 사용
optim.AdamW	가중치 감쇠가 포함된 Adam 옵티마이저	Adam과 유사하지만, 가중치 감쇠(weight decay)를 별도로 적용
optim.RMSprop	루트 평균 제곱 전파 (RMSprop) 옵티마이저	순환 신경망(RNN) 및 LSTM 모델에서 주로 사용
optim.Adagrad	적응형 경사 하강법 (Adagrad) 옵티마이저	학습률이 점차 줄어들어야 하는 문제에서 사용
optim.Adadelta	Adadelta 옵티마이저	Adagrad의 학습률 감소 문제를 해결
optim.ASGD	평균 SGD (ASGD) 옵티마이저	장기적으로 안정적인 학습을 위해 사용
optim.LBFGS	제한된 메모리 BFGS 옵티마이저	배치 크기가 작은 문제에서 사용
optim.Rprop	Resilient Backpropagation (Rprop) 옵티마이저	뉴럴 네트워크의 경사 하강법에서 사용
optim.SparseAdam	희소 데이터에 특화된 Adam 옵티마이저	희소 행렬을 사용하는 모델에서 사용

학습과 예측

참고: 케라스와 파이토치

기능	Keras	PyTorch
모델 컴파일	<code>compile(optimizer, loss)</code>	옵티마이저(<code>optim.SGD</code>) 및 손실 함수(<code>nn.CrossEntropyLoss</code>) 설정 <code>criterion = nn.CrossEntropyLoss()</code> <code>optimizer = optim.SGD(model.parameters(), lr=0.01)</code>
모델 학습	<code>fit(x_train, y_train, epochs, batch_size)</code>	<code>train(model, train_loader, criterion, optimizer)</code> 직접 구현
모델 예측	<code>predict(x_test)</code>	<code>predict(model, test_loader)</code> 직접 구현

기능	설명
모델 정의	SimpleModel 클래스를 정의하여 신경망 구조 구현
옵티마이저 설정	<code>optim.SGD</code> 를 사용하여 경사 하강법(SGD) 옵티마이저 정의
손실 함수 설정	<code>nn.CrossEntropyLoss</code> 를 사용하여 손실 함수 정의
데이터 로더	<code>DataLoader</code> 를 사용하여 훈련 및 테스트 데이터를 미니배치로 로드
학습 루프	<code>train</code> 함수를 정의하여 모델 학습 루프 구현
예측 함수	<code>predict</code> 함수를 정의하여 모델 예측 구현

학습 데이터셋

참고: 케라스와 파이토치

기능	Keras	PyTorch
NumPy 배열	넘파이 배열을 직접 학습 데이터로 사용할 수 있음	넘파이 배열을 직접 학습 데이터로 사용할 수 없음
텐서 변환		<pre>x_train_tensor = torch.tensor(X_train, dtype=torch.float32) y_train_tensor = torch.tensor(y_train, dtype=torch.long)</pre>
데이터 로더		<pre>from torch.utils.data import DataLoader, TensorDataset # 텐서 데이터셋 생성 train_dataset = TensorDataset(x_train_tensor, y_train_tensor) # 데이터 로더 생성 train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)</pre>



1. NumPy 배열 생성: 학습 데이터를 NumPy 배열로 생성합니다.
2. NumPy 배열을 PyTorch 텐서로 변환: torch.tensor 함수를 사용하여 NumPy 배열을 PyTorch 텐서로 변환합니다.
3. TensorDataset 생성: TensorDataset을 사용하여 텐서 데이터를 묶어줍니다.
4. DataLoader 생성: DataLoader를 사용하여 데이터를 배치 단위로 로드합니다.
5. 학습 루프 정의: 변환된 데이터를 사용하여 모델을 학습시킵니다.

학습

참고: 케라스와 파이토치

```
1. def train(model, train_loader, criterion, optimizer, epochs=100):
2.     model.train() # 모델을 학습 모드로 설정
3.     for epoch in range(epochs):
4.         running_loss = 0.0 # 현재 epoch의 누적 손실 초기화
5.         for data, target in train_loader: # 미니배치 단위로 데이터 로드
6.             optimizer.zero_grad() # 옵티마이저의 기울기 버퍼 초기화
7.             output = model(data) # 모델을 통해 예측값 계산
8.             loss = criterion(output, target) # 손실 계산
9.             loss.backward() # 역전파를 통해 기울기 계산
10.            optimizer.step() # 옵티마이저를 통해 가중치 업데이트
11.            running_loss += loss.item() # 손실 누적
12.        # epoch 종료 후 평균 손실 출력
13.        print(f'Epoch {epoch+1}, Loss: {running_loss/len(train_loader)}')
14.
15.
16. train(model, train_loader, criterion, optimizer) # 모델 학습
```

예측

참고: 케라스와 파이토치



모델 평가 모드 설정: `model.eval()`을 호출하여 모델을 평가 모드로 설정합니다. 이는 드롭아웃이나 배치 정규화 레이어가 학습 모드와 다르게 동작하게 합니다.

- 기울기 계산 비활성화: `with torch.no_grad()` 블록 안에서 예측을 수행하여 기울기 계산을 비활성화합니다. 이는 메모리 사용을 줄이고 연산 속도를 높여줍니다.
- 데이터 로드: `DataLoader`를 사용하여 데이터를 배치 단위로 로드합니다. 여기서 `TensorDataset`을 사용하여 입력 데이터를 묶어줍니다.
- 모델 예측: 모델을 통해 예측값을 계산하고, `argmax`를 사용하여 가장 높은 값을 가진 클래스를 선택합니다.
- 결과 반환: 예측값을 리스트에 저장한 후, 이를 텐서로 변환하고 다시 NumPy 배열로 반환합니다.

```
1. def predict(model, test_loader):
2.     model.eval() # 모델을 평가 모드로 설정
3.     predictions = [] # 예측값을 저장할 리스트 초기화
4.     with torch.no_grad(): # 기울기 계산 비활성화
5.         for data in test_loader: # 미니배치 단위로 데이터 로드
6.             inputs = data[0] # 텐서 데이터셋에서 입력 데이터 추출
7.             output = model(inputs) # 모델을 통해 예측값 계산
8.             predicted_classes = output.argmax(dim=1, keepdim=True) # 가장 높은 값을 가진 클래스 선택
9.             predictions.append(predicted_classes) # 예측값 리스트에 추가
10.    return torch.cat(predictions).numpy() # 리스트를 텐서로 변환하고 numpy 배열로 반환
11.
12.
13. predictions = predict(model, test_loader) # 모델 예측
```

모델 저장하고 불러오기

참고: 케라스와 파이토치

기능	Keras	PyTorch
모델 가중치 저장	<code>model.save_weights('model_weights.keras')</code>	<code>torch.save(model.state_dict(), 'model_weights.pth')</code>
모델 가중치 로드	<code>model.load_weights('model_weights.keras')</code>	<code>model.load_state_dict(torch.load('model_weights.pth'))</code>
모델 전체 저장	<code>model.save('model.keras')</code>	<code>torch.save(model, 'model.pth')</code>
모델 전체 로드	<code>model = keras.models.load_model('model.keras')</code>	<code>model = torch.load('model.pth')</code>
모델 구조 저장	<pre>model_json = model.to_json() with open("model.json", "w") as json_file: json_file.write(model_json)</pre>	<pre>model_dict = { "fc1": { "in_features": model.fc1.in_features, "out_features": model.fc1.out_features }, "fc2": {... 생략...} } with open("model.json", "w") as json_file: json.dump(model_dict, json_file)</pre>
모델 구조 불러오기	<pre>from tensorflow.keras.models import model_from_json # JSON 파일에서 모델 구조 불러오기 with open("model.json", "r") as json_file: loaded_model_json = json_file.read() loaded_model = model_from_json(loaded_model_json)</pre>	<pre>with open("model.json", "r") as json_file: model_dict = json.load(json_file) class SimpleModel(nn.Module): def __init__(self, in_features1, out_features1, ...생략...): super(SimpleModel, self).__init__() self.fc1 = nn.Linear(in_features1, out_features1) ... 생략... loaded_model = SimpleModel(model_dict["fc1"]["in_features"], model_dict["fc1"]["out_features"], ...생략...)</pre>