포트 폴리오

20190717 조나영

1. AI 개요
2. 인공지능, 머신러닝, 딥러닝
3. 인공신경망 개요
4. GPU
5. 텐서플로
6. MNIST 데이터 셋
7. 딥러닝 구현 순서
8. 예측
    - 예제
9. 주요용어
10. 인공신경망 퍼셉트론
11. 논리게이트
12. 회귀와 분류
    -가설
    -손실함수
    -최적화과정
    -학습률

13. 예제
    선형회귀 y=2x 예측
    선형회귀 y=2x 예측
    2018 년 대한민국 인구증가율과 고령인구비율
    케라스 모델 미사용 텐서플로 프로그래밍
    보스톤 주택 가격 예측
    자동차 연비 데이터(auto mpg)로 회귀 분석
14. 이항 분류 및 다항 분류
    이항 분류 : 레드 와인과 화이트 와인 구분
    다항 분류 : 와인 품질 분류
    다항 분류 : 패션 MNIST

인공지능과 딥러닝 개요

AI 시작

- 앨런 튜링
    - 1950 년, 논문 <Computing machinery and intelligence>을 발표
      → 생각하는 기계의 구현 가능성에 대한 내용
    - 인공지능 실험, '튜링 테스트'
      → 텍스트로 주고받는 대화에서 기계가 사람인지 기계인지 구별할 수 없을 정도로 대화를 잘 이끌어 간다면, 이것은 기계가 "생각"하고 있다고 말할 충분한 근거가 된다 (지금의 챗봇)

- **인공지능(Artificial Intelligence)의 처음 사용**
    - 1956 년 다트머스대 학술대회
        - 세계 최초의 AI 프로그램인 논리 연산기(Logic Theorist)를 발표
        -

AI 와 딥러닝 역사

1940 년 시작 → (두번의 혹한기) → 2010 년(최고의 전성기)
• AI 의 첫번째 암흑기 (1969-1980) – 마빈 민스키(Marvin Minsky)의 인공신경망(Artificial Neural Network)인 퍼셉트론 (Perceptron)에 대한 비판으로 촉발 • AI 의 두번째 암흑기 (1987-1993) – 규칙 기반의 전문가시스템의 의구심과 한계

딥러닝이 인기인 이유

2000 년 이후의 문제가 해결이 되기 때문
→ 빅데이터 ↑, 컴퓨터 속도↑(계산 속도↑), 알고리즘

인공지능, 머신러닝, 딥러닝

인공지능 (AI : Artificial Intelligence)

- 컴퓨터가 인간처럼 지적 능력을 갖게 하거나 행동하도록 하는 모든 기술

머신러닝(machine learning)

- 기계가 스스로 학습할 수 있도록 하는 인공지능의 한 연구 분야

- SVM(Support Vector Machine): 수학적인 방식의 학습 알고리즘
-

딥러닝

- 다중 계층의 신경망 모델(뉴런네트워크)을 사용하는 머신러닝의 일종

**특징, 데이터가 많을 수록 딥러닝에 적합

머신러닝(machine learning)(=기계학습)

– 주어진 데이터를 기반으로 기계가 스스로 학습하여 성능을 향상시키거나 최적의 해답을 찾기 위한 학습 지능 방법

– 스스로 데이터를 반복적으로 학습하여 기술을 터득하는 방식

- 명시적으로 프로그래밍(explicit programming)을 하지 않아도 컴퓨터가 학습을 할 수 있도록 해주는 인공지능의 한 형태
- 더 많은 데이터가 유입되면, 컴퓨터는 더 많이 학습을 하고, 시간이 흐르면서 더 스마트 해져서 작업을 수행하는 능력과 정확도가 향상
    - 분류
        - 지도학습(supervised learning) [정답의 훈련데이터로 학습]
            - 올바른 입력과 출력의 쌍으로 구성된 정답의 훈련 데이터(labeled data)로부터 입출력 간의 함수를 학습시키는 방법
            - k-최근접 이웃 (k-Nearest Neighbors)
            - 선형 회귀 (Linear Regression)
            - 로지스틱 회귀 (Logistic Regression)
            - 서포트 벡터 머신 (Support Vector Machines (SVM))
            - 결정 트리 (Decision Tree)와 랜덤 포레스트 (Random Forests)
        - 비지도학습(자율학습)(unsupervised learning) [정답이 없는 훈련데이터로 학습]
            - 정답이 없는 훈련 데이터(unlabeled data)를 사용하여 데이터 내에 숨어있는 어떤 관계를 찾아내는 방법

              - 군집 (Clustering)
              - k-평균 (k-Means)
              - 계층 군집 분석 (Hierarchical Cluster Analysis (HCA))
              - 기댓값 최대화 (Expectation Maximization)

– 시각화 (Visualization)와 차원 축소(Dimensionality reduction)

- 주성분 분석 (Principal Component Analysis (PCA))
- 커널 (kernel PCA)
- 지역적 선형 임베딩 (Locally-Linear Embedding (LLE)) – 연관 규칙 학습 (Association rule learning)
- 어프라이어리 (Apriori)
- 이클렛 (Eclat)

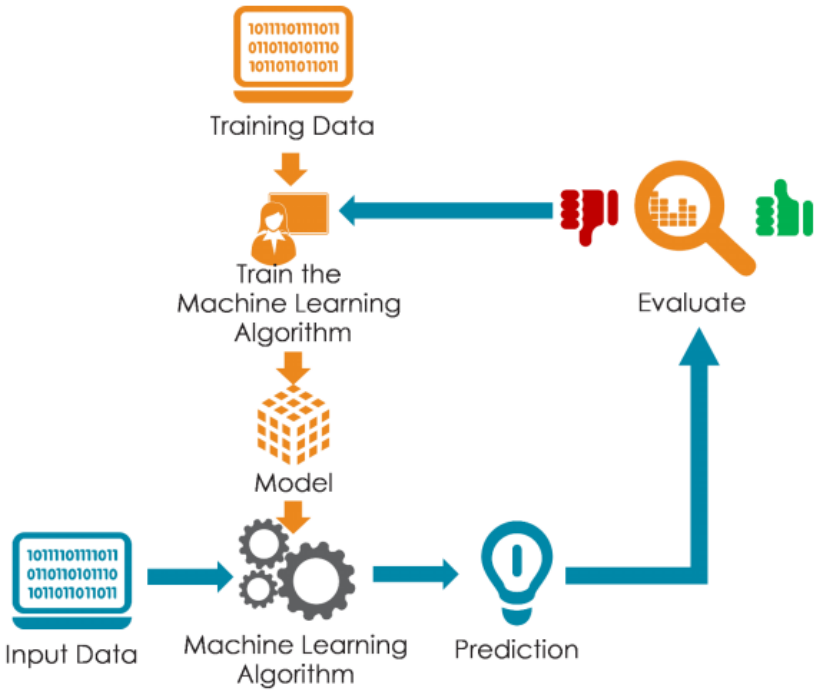
▪ 강화학습(reinforcement learning) [잘하면 상, 못하면 벌]

▪ 잘한 행동에 대해 보상을 주고 잘못한 행동에 대해 벌을 주는 경험을 통해 지식을 학 습하는 방법

– 딥마닝의 알파고

– 자동 게임분야

머신러닝 흐름



Train Data → 머신러닝알고리즘 훈련→ 모델 → 알고리즘 → 예측 → 평가 (→ 머신러닝알고리즘 훈련)

**머신러닝 알고리즘 훈련 : 딥러닝인경우 인공신경망(ANN) 사용

머신 러닝과 딥 러닝의 차이점

	기계 학습	딥 러닝
데이터 의존성	중소형 데이터 세트에서 탁월한 성능	큰 데이터 세트에서 뛰어난 성능
하드웨어 의존성	저가형 머신에서 작업하십시오.	GPU가있는 강력한 기계가 필요합니다. DL은 상당한 양의 행렬 곱셈을 수행합니다.
기능 공학	데이터를 나타내는 기능을 이해해야 함	데이터를 나타내는 최고의 기능을 이해할 필요가 없습니다
실행 시간	몇 분에서 몇 시간	최대 몇 주. 신경망은 상당한 수의 가중치를 계산해야합니다.

인공신경망에서 시작된 딥러닝

퍼셉트론(perceptron)

세계 최초의 인공신경망을 제안

• 1957 년 코넬대 교수, 심리학자인 프랭크 로젠블랫(Frank Rosenblatt)

– 신경망에서는 방대한 양의 데이터를 신경망으로 유입

• 데이터를 정확하게 구분하도록 시스템을 학습시켜 원하는 결과를 얻어냄

인공신경망 (ANN)

인공신경망(ANN: Artificial Neural Network) 사용

– 인간의 뇌는 1000 억개의 뉴런으로 구성

• 뇌를 구성하는 신경세포 뉴런(Neuron)의 동작원리에 기초한 기술

– 인간의 신경세포인 뉴런(neuron)을 모방하여 만든 가상의 신경

• 뇌와 유사한 방식으로 입력되는 정보를 학습하고 판별하는 신경 모델

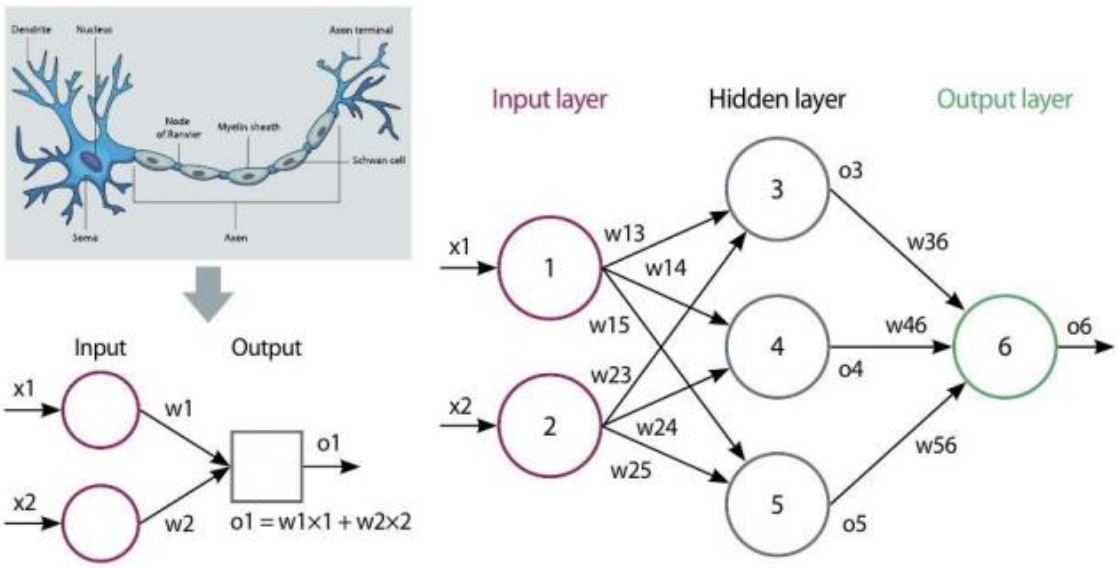
MLP(Multi Layer Perceptron)

– 입력층(input layer)과 출력층(output layer)

• 다수의 신호(input)를 입력 받아서 하나의 신호(output)를 출력

– 중간의 은닉층(hidden layer)

• 여러 개의 층으로 연결하여 하나의 신경망을 구성



DNN(deep neural network)

심층신경망(deep neural network)

– 다중 계층인 심층신경망(deep neural network)을 사용

• 학습 성능을 높이는 고유 특징들만 스스로 추출하여 학습하는 알고리즘

• 입력 값에 대해 여러 단계의 심층신경망을 거쳐 자율적으로 사고 및 결론 도출

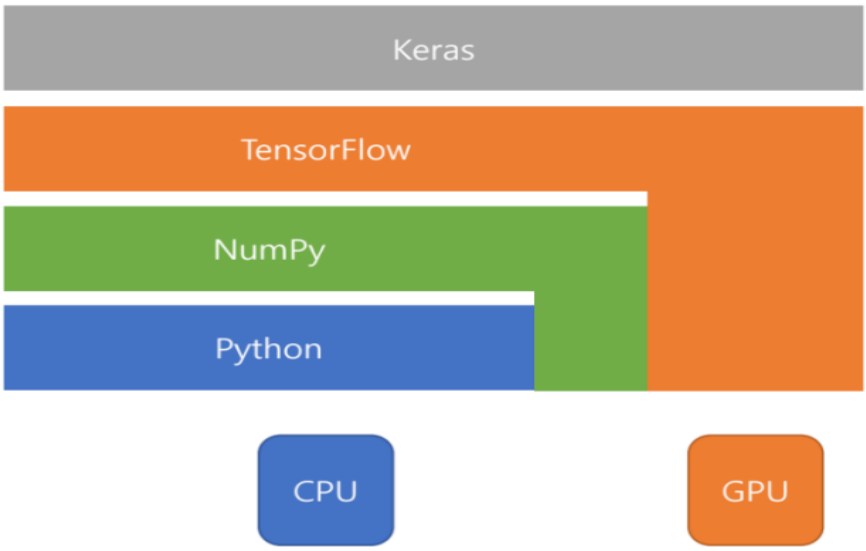
GPU

- 그래픽 처리 장치 GPU(Graphics Processing Unit)
  – 그래픽 연산 처리를 하는 전용 프로세서
  – GPU 란 용어는 1999 년 엔비디아(Nvidia)에서 처음 사용
- GPGPU(General Purpose Graphic Processing Unit)
  – 일반 CPU 프로세서를 돕는 보조프로세서(coprocessor)로서의 GPU
  – 중앙 처리 장치(CPU)가 맡았던 응용 프로그램들의 계산에 GPU 를 사용하는 기술 (계산속도 ↑)
  • GPU 컴퓨팅이란 GPGPU 를 연산에 참여
  • 고속의 병렬처리로 대량의 행렬과 벡터를 다루는 데 뛰어난 성능을 발휘
  – 딥러닝의 심층신경망에서 빅데이터를 처리하기 위해 대량의 행렬과 벡터를 사용
  • GPU 사용이 매우 효과적 – GPU(12 개) == CPU(2,000 개) [계산 능력]

CUDA(Compute Unified Device Architecture)

GPU 업체인 NVIDIA 의 GPU 를 사용하기 위한 라이브러리 소프트웨어

**PC 에서 GPU 를 이용해서 DNN 을 사용하기 위해서는 CUDA Driver, CuDNN 를 설치해야 사용가능



구글의 TPU

2016 년 텐서 처리 장치(Tensor Processing Unit)를 발표

– 텐서란 벡터·행렬 을 의미

– TPU 는 데이터 분석 및 딥러닝용 칩으로서 벡터·행렬연산의 병렬처리에 특화

– 텐서플로(TensorFlow) • TPU 를 위한 소프트웨어

⇒ GPU 보다 빠른 계산장치용 CPU 를 만듬(판매용 X)

텐서플로 개요

딥러닝 라이브러리 : 텐서플로, 케라스, 파이토치 (적합언어 : 파이썬)

케라스(Keras) 개요

독자적인 고수준 라이브러리

– 엔진으로 텐서플로, 씨아노, CNTK 등을 사용 현재는 Tensorflow 의 고수준 API 로도 사용

– 동일한 코드로 CPU 와 GPU 에서 실행 가능

– 사용하기 쉬운 API 를 가지고 있어 딥러닝 모델의 프로토타입을 빠르게 생성

텐서플로(TensorFlow) 개요

구글(Google)에서 만든 라이브러리 – 연구 및 프로덕션용 오픈소스 딥러닝 라이브러리

- 딥러닝 프로그램을 쉽게 구현할 수 있도록 다양한 기능을 제공 • 데스크톱, 모바일, 웹, 클라우드 개발용 API 를 제공 – 구현 및 사용

- Python, Java, Go 등 다양한 언어를 지원 – 텐서플로 자체는 기본적으로 C++로 구현 • 파이썬을 최우선으로 지원 – 대부분의 편한 기능들이 파이썬 라이브러리로만 구현 – Python 에서 개발하는 것이 편함
- 텐서보드(TensorBoard)
  브라우저에서 실행 가능한 시각화 도우미 – 딥러닝 학습 과정을 추적하는데 유용하게 사용

  ## 텐서 개요

  Tensor(텐서): 모든 데이터 (딥러닝에서 데이터를 표현하는 방식)
  - 0-D 텐서 : 스칼라 (차원이 없는 텐서) ex) 10
  - 1-D 텐서 : 벡터 (1 차원 텐서) ex) [10, 20]
  - 2-D 텐서 : 행렬 등 (2 차원 텐서) ex) [[1,2],[3,4]]

  – n 차원 행렬(배열) ex) [[[1,2],[2,3]],[2,4],[4,3]]]
  - 텐서는 행렬로 표현할 수 있는 n 차원 형태의 배열을 높은 차원으로 확장

TensorFlow 계산 과정

– 모두 그래프(Graph)라고 부르는 객체 내에 저장되어 실행 – 그래프를 계산하려면 외부 컴퓨터에 이 그래프 정보를 전달하고 그 결과값을 받아야 함

session

```
x = tf.constant(3)
y = x**2

sess = tf.Session()
print(sess.run(x))
print(sess.run(y))
sess.close()
```

– 세션 생성
• Session 객체 생성
– 세션 사용
• run 메서드에 그래프를 입력하면 출력 값을 계산하여 반환
– 세션 종료
• close 메서드
• with 문을 사용하면 명시적으로 호출 불필요
** 텐서 1.0 만 해당/ 19.6 월에 나온 2.0 부분은 생략가능(아예 사용불가능 sess)

TensorFlow API 계층

– TensorFlow 딥 러닝 모델 구축 작업은 서로 다른 API 수준을 사용하여 해결
• 고급 API – Keras 나 TF-SIim 과 같은 추상화 라이브러리를 제공하여 저수준 텐서플로 라이브러리에 대해 손쉽게 고수준 접근이 가능하게 해줌
• 중급 API
• 하위 수준 API
** 케라스, 텐서플로 뭐가 좋아요? 텐서플로 안에 케라스가 있음

개발환경

구글의 Colab (Google Drive + Jupyter Notebook) – 파이썬과 머신러닝, 딥러닝 개발 클라우드 서비스

- 구글 계정 필요(구글 드라이브를 기본 저장소로 사용)
- 클라우드 기반의 무료 Jupyter 노트북 개발 환경
  – 주피터 노트북을 지원하는 머신러닝, 딥러닝 클라우드 개발환경
  – 파이썬 뿐만 아니라 판다스, 멧플롯리브의 시각화 및 텐서플로우나 케라스 등 딥러닝 라이브러리도 쉽게 사용
- 구글 계정 전용의 가상 머신 지원 – GPU, TPU 지원
- Google drive 문서와 같이 링크만으로 접근 / 협업 가능

장점 : 구글 드라이브, 깃허브랑 연계
사양 : 일반 개인 pc 보다 성능우수 , CPU 사용, 딥러닝시 GPU,TPU 사용
아나콘다 (자신의 PC 에 설치해 활용가능)

- 데이터과학에 관련된 것

https://github.com/chonayoung/2020-2-AI/blob/main/code/01_tf_basic.ipynb

코랩 버전 바꾸기

%tensorflow_version 1.x → 1.x 를 실행 %tensorflow_version 2.x → 2.x 를 실행

- import 하기 전에 '런타임 다시 시작'를 누르고 해당 버전을 실행을 하면, 버전 바꿀 수 있음

버전 보기

import tensorflow as tf tf.**version**

텐서플로 2.0 에서 즉시 실행은 기본으로 활성화 tf.executing_eagerly()

텐서 정보 출력

- 1.0 일 때 세션 사용
- 2.0 일 때 값만 보려면 numpy()를 이용

a = tf.constant([1,2,3])

a.shape() #행렬 알려줌

TensorShape([3])

조건연산 tf.cond()

tf.cond(pred, true_fn=None, false_fn=None, name=None)

⇒ (조건, 정답, 오류, name=None) – pred 를 검사해 참이면 true_fc 반환 – pred 를 검사해 거짓이면 false_fc 반환

배열 텐서 연산

텐서의 브로드캐스팅 (중간고사 문제)

- Shape 이 다르더라도 연산이 가능하도록 가지고 있는 값을 이용하여 Shape 을 맞춤

import numpy as np

x = tf.constant((np.arange(3))) y = tf.constant([5], dtype=tf.int64) print((x+y).numpy())

x = tf.constant((np.ones((3, 3)))) y = tf.constant(np.arange(3), dtype = tf.double)
print((x+y).numpy())

x = tf.constant(np.arange(3).reshape(3, 1)) y = tf.constant(np.arange(3))
print((x+y).numpy())

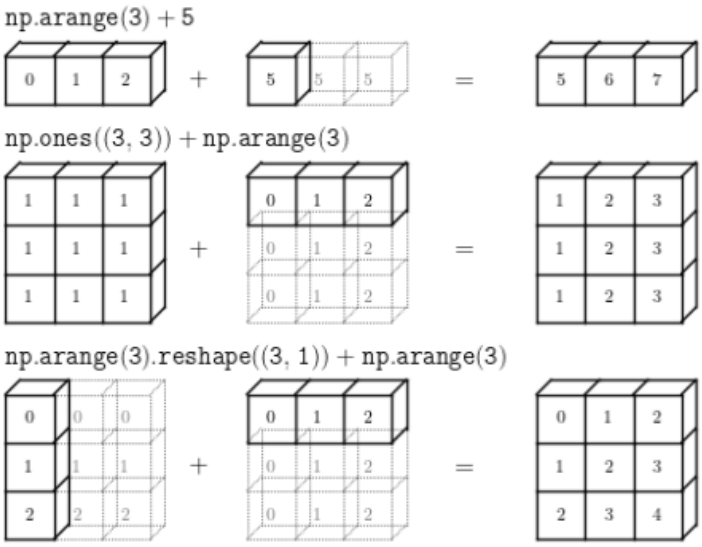
[5 6 7] [[1. 2. 3.]

[1. 2. 3.]

[1. 2. 3.]] [[0 1 2]

[1 2 3]

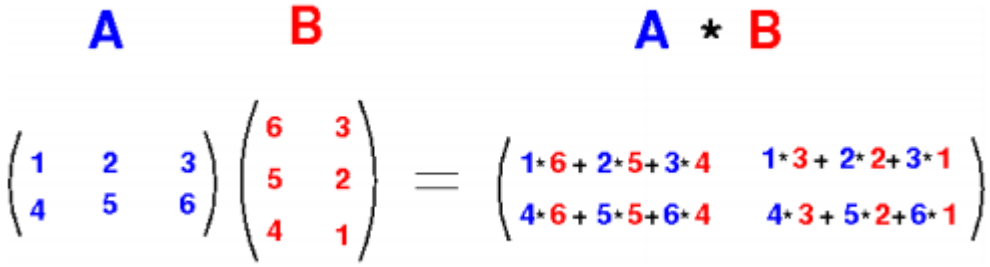
[2 3 4]]



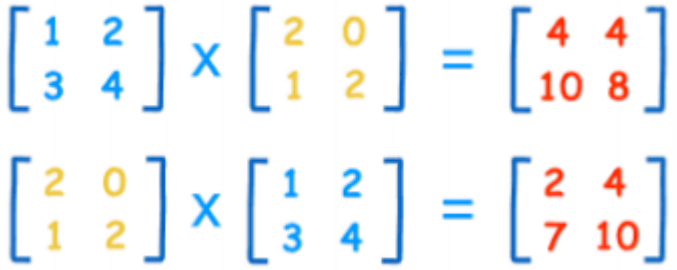
https://github.com/chonayoung/2020-2-AI/blob/main/code/02_tf_interm.ipynb

행렬 곱셈;

행렬 곱(내적)



$$C_{ij} = \sum_k A_{ik} B_{kj} = A_{ik} B_{kj}$$



Numpy : • np.dot(a, b) • a.dot(b)

Tf • tf.matmul()

텐서 연산

tf.add() : 더하기

tf.multiply() : 곱하기

tf.pow() : a 의 b tf.reduce_mean() : 평균 tf.reduce_sum() : 합

tf.rank() : 행렬의 차수반환

tf.zeros() : 0 으로 행렬 만듬 ex) tf.zeros([2,5,5,3]) 2*55*3 인 행렬임

tf.shape() : 현재 행렬 상태 보여줌

tf.reshape() : 해당 행렬로 형태 변경

- ex) tf.reshape(rank_three_tensor, [6, 10]) ⇒rank_three_tensor 를 6*10 행렬 형태변경
- ex) tf.reshape(matrix, [3, -1])

⇒ 기존 내용을 3x20 행렬로 형태 변경

⇒ -1 은 차원 크기를 계산하여 자동으로 결정하라는 의미

**형태가 변경된 텐서의 원소 개수 == 원래 텐서의 원소 개수

tf.cast : tf.Tensor 의 자료형을 다른 것으로 변경

변수 Variable

텐서플로 그래프에서 tf.Variable 의 값을 사용하려면 이를 단순히 tf.Tensor 로 취급

- 메소드 assign, assign_add – 값을 변수에 할당
- 메소드 read_value – 현재 변수값 읽기

https://github.com/chonayoung/2020-2-AI/blob/main/code/03_tf_random.ipynb

텐서플로 난수

균등 분포 난수 : tf.random.uniform([1], 0, 1) ⇒ 배열형태, [시작, 끝)

정규 분포 난수 : tf.random.normal([4],0,1) ⇒ 크기, 평균, 표준편차

섞는 것 : tf.random.shuffle(a)

https://github.com/chonayoung/2020-2-AI/blob/main/code/04_tf_mnist_basic.ipynb

MNIST 데이트셋

딥러닝 손글씨 인식에 사용되는 데이터셋(손으로 쓴 자릿수에 대한 데이터 집합)

- MNIST((Modified National Institute of Standards and Technology)
- 미국 국립 표준 기술원(NIST)
- "필기 숫자 이미지"와 정답인 " 레이블 " 의 쌍으로 구성 (지도학습)
    - 숫자의 범위는 0 에서 9 까지, 총 10 개의 패턴을 의미
    - 필기 숫자 이미지

      784 픽셀(회색조 이미지) – 28 X 28 • 내부 값은 0~255 – 이 값을 0~1 로 수정해서 사용
    - 레이블(Label)

      이미지의 정답: 필기 숫자 이미지가 나타내는 실제 숫자, 0 에서 9

케라스 딥러닝 구현

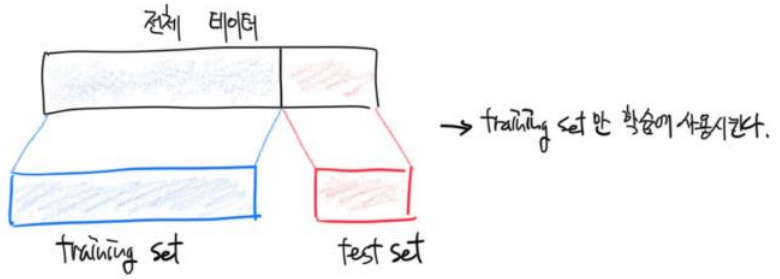
1. 딥러닝 모델 만듬 (define)
2. 주요 훈련 방법 설정 (compile)

   – 최적화 방법(optimizers), 손실 함수(losses), 훈련 모니터링 지표(metrics)
3. 훈련 (fit)
4. 테스트 데이터 평가 (evaluate)
5. 정답 예측 (predict)

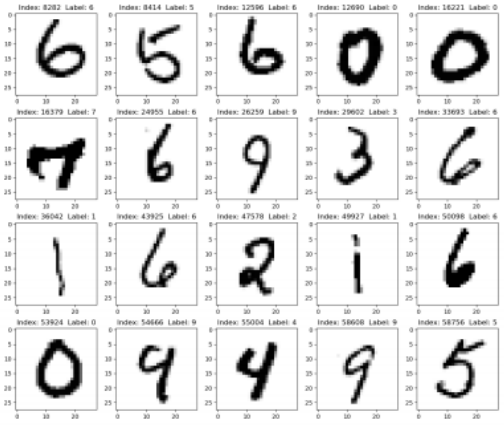
MNIST 데이터

훈련 데이터 손글씨 : 총 6 만개 ⇒ x_train(훈련) , y_train(정답)

테스트 데이터 손글씨 : 총 1 만개 ⇒ x_test(테스트) , y_test (정답)



```
# 랜덤하게 20 개의 훈련용 자료를 그려 보자.
from random import sample
nrows, ncols = 4, 5 #츨력 가로 세로 수
# 출력할 첨자 선정
idx = sorted(sample(range(len(x_train)), nrows * ncols))
#print(idx)
count = 0
plt.figure(figsize=(12, 10))
for n in idx:
        count += 1
plt.subplot(nrows, ncols, count)
tmp = "Index: " + str(n) + " Label: " + str(y_train[n])
plt.title(tmp)
plt.imshow(x_train[n], cmap='Greys')
plt.tight_layout()
plt.show()
```



MNIST 데이터 셋을 위한 딥러닝

- 0 에서 9 까지의 분류(classification) – Number of classes: 10 (클래스 10 개)
- 딥러닝 과정
    1. 모델 구성(개발)
    2. 블랙 박스(black box)
    2. 모델 훈련 (train)

– 모델이 문제를 해결하도록 훈련 ⇒ 어린 아이가 부모에게 훈련 받는 것에 비유
– 학습 방법 및 모니터링 지표 설정
• 경사하강법(내리막 경사 따라 가기)
• 손실 함수(Loss Function)
• 모니터링 지표 metrics

3. 예측 (inference, prediction)

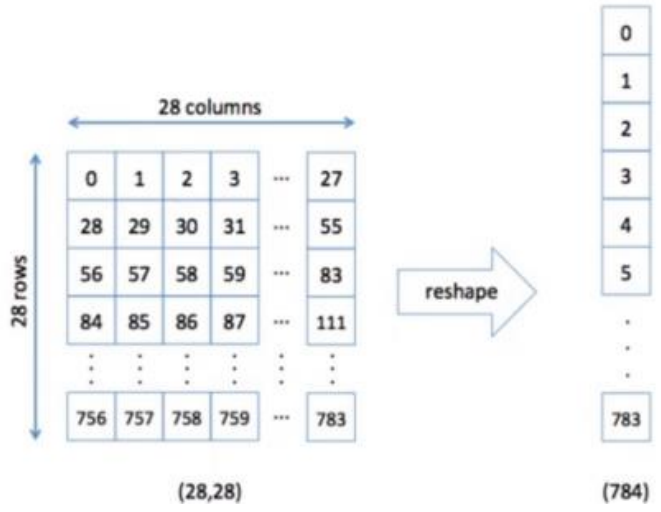
딥러닝 구현 순서

0) 필요 모듈 임포트

1) 훈련과 정답 데이터 지정
MNIST 데이터셋을 로드하여 준비
– 전처리
• 샘플 값을 정수에서 부동소수로 변환
– 한 비트의 값을 255 로 나눔

1-1) 데이터 전처리(옵션)
- 정규화 결과 (수치데이터) – 픽셀 값은 0 에서 1 사이의 값

2) 모델 구성
- 층을 차례대로 쌓아 tf.keras.models.Sequential 모델을 생성 (신경망 구성)
- Neural Networks : 입력층, 중간층(은닉층), 출력층
- 2 차원 그림 → 1 차원 평탄화
  - Flatten(input_shape=(28, 28)), – 60000 개의 (28, 28) 크기를 가진 배열 [60000 개의 (28 * 28) 크기의 배열로 수정]



- 단순 신경망 모델 (중간은닉층이 없는 구조)(입력층과 출력층만 존재)
- Dense() : 완전연결층 (입력, 은닉, 출력 있음)

3) 학습에 필요한 최적화 방법과 손실 함수 등 설정

– 옵티마이저: 입력된 데이터와 손실 함수를 기반으로 모델(w 와 b)을 업데이트하는 메커니즘

– 손실 함수: 훈련 데이터에서 신경망의 성능을 측정하는 방법 (모델이 옳은 방향으로 학습될 수 있도록 도와주는 기준 값)

– 훈련과 테스트 과정을 모니터링할 지표

• 여기에서는 정확도(정확히 분류된 이미지의 비율)만 고려

3-1) 구성된 모델 요약(옵션)

- model.summary() – 각 층의 구조와 패라미터 수 표시
- 가중치(weights)와 편향(biases) – 총 패라미터 수
- 모델이 구해야 할 수(가중치, 편향) 의 개수

4) 생성된 모델로 훈련 데이터 학습

- model.fit()
  • 훈련 횟수 epochs 에 지정
- fit() 메서드 호출 (모델의 매개변수를 정하는 과정)
  – 훈련 데이터에 모델을 학습

5) 테스트 데이터로 성능 평가

- 테스트 세트에서도 모델이 잘 작동하는지 확인
  – model.evaluate() (손실 값과 예측 정확도 반환) [loss, accuracy]

5-1) 테스트 데이터 또는 다른 데이터로 결과 예측(옵션)

```
import tensorflow as tf
#mnist 모듈 준비
mnist = tf.keras.datasets.mnist
# MNIST 데이터셋을 훈련과 테스트 데이터로 로드하여 준비
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# 샘플 값을 정수(0~255)에서 부동소수(0~1)로 변환
x_train, x_test = x_train / 255.0, x_test / 255.0
#층을 차례대로 쌓아 tf.keras.models.Sequenmtial 모델을 생성
```

```
model = tf.keras.models.Sequential([ tf.keras.layers.Flatten(input_shape=(28, 28)) #입력층,
tf.keras.layers.Dense(128, activation='relu') #은닉층, tf.keras.layers.Dropout(0.2) #드롭아웃,
tf.keras.layers.Dense(10, activation='softmax') #출력층
 ])
#훈련에 사용할 옵티마이저(optimizer)와 손실 함수, 출력정보를 모델에 설정
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy']) # metrics=['accuracy', 'mse'])
# 모델 요약 표시
Model.summary()
#모델을 훈련 데이터로 총 5 번 훈련
model.fit(x_train, y_train, epochs=5)
#모델을 테스트 데이터로 평가
model.evaluate(x_test, y_test)
```

예측

**모델 구성후 사용해야 함

model.predict(input)

- input : 모델의 fit(), evaluate()에 입력과 같은 형태가 필요 (28*28 이미지가 여러개인 3 차원)
    - 슬라이스 이용 ex) pred_result = model.predict(x_test[:1])
    - 정답으로 나오는 10 개의 실수는 확률의 값 (0 일 확률값, 1 일 확률값...)
      ** 10 개의 실수를 더하면 1 이 나옴.
- One hot encoding – 하나의 자리만 1, 나머지는 모두 0
  – 데이터가 취할 수 있는 모든 단일 범주에 대해 하나 의 새 열을 생성
- argmax() 로 가장 큰 수의 위치 첨자를 반환
  – np.argmax()
  1 차원 : 그대로 ⇒ 결과 : 첨자 (스칼라)
  2 차원 : axis=1(인자)를 넣어줘야함 ⇒ 결과 : 1 차원(벡터)
  – tf.argmax() : 텐서로 나옴
- 활성화 함수 softmax()

층을 차례대로 쌓아 tf.keras.models.Sequential 모델을 생성

model = tf.keras.models.Sequential([tf.keras.layers.Flatten(input_shape=(28, 28)),
tf.keras.layers.Dense(128, activation='relu') #은닉층,
tf.keras.layers.Dropout(0.2) #드롭아웃 (20%) 끊기,
tf.keras.layers.Dense(10, activation='softmax')])

- 평탄화 메소드 flatten (ary.flatten())
- 드롭 아웃

---

from random import sample import numpy as np
#x_test 로 직접 결과 처리
pred_result = model.predict(x_test)
print(pred_result.shape)
print(pred_result[0])
print(np.argmax(pred_result[0]))
#원핫 인코딩을 일반 데이터로 변환
pred_labels = np.argmax(pred_result, axis=1)
#예측한 답 출력
print(pred_labels)
#실제 정답 출력
print(y_test)

예측이 맞는 소스

<<임의의 20 개 샘플 예측 값과 정답 그리기 소스>>
##
from random import sample import numpy as np
#예측한 softmax 의 확률이 있는 리스트 pred_result
pred_result = model.predict(x_test)
#실제 예측한 정답이 있는 리스트 pred_labels
pred_labels = np.argmax(pred_result, axis=1)
#랜덤하게 20 개의 훈련용 자료를 예측 값과 정답, 그림을 그려 보자. samples =
sorted(sample(range(len(x_test)), nrows * ncols)) # 출력할 첨자 선정
#임의의 20 개 그리기
count = 0 nrows, ncols = 5, 4 plt.figure(figsize=(12,10)) for n in samples: count += 1
plt.subplot(nrows, ncols, count)

#예측이 틀린 것은 파란색으로 그리기

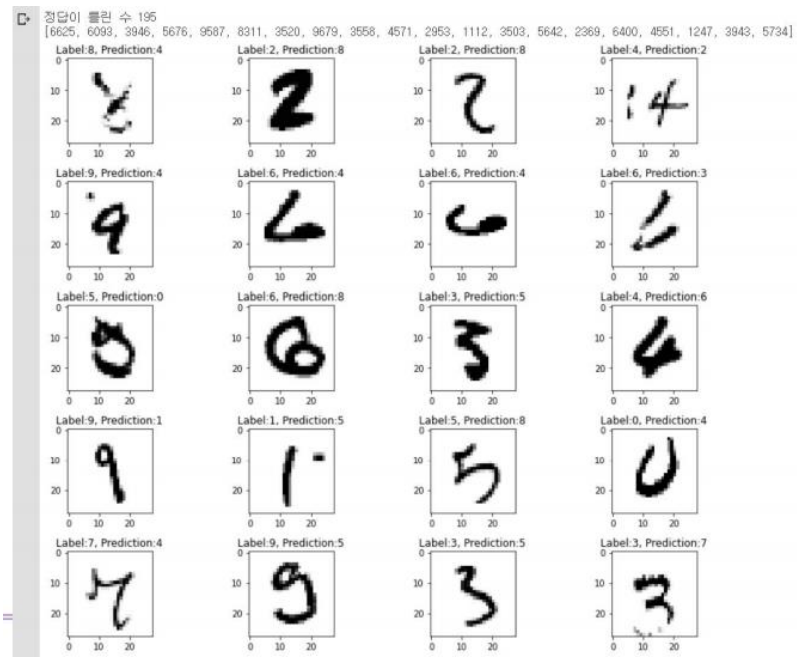
```
cmap = 'Greys' if ( pred_labels[n] == y_test[n]) else 'Blues'
plt.imshow(x_test[n].reshape(28, 28), cmap=cmap, interpolation='nearest')
tmp = "Label:" + str(y_test[n]) + ", Prediction:" + str(pred_labels[n]) plt.title(tmp)
plt.tight_layout()
plt.show()
```

예측이 틀린 소스

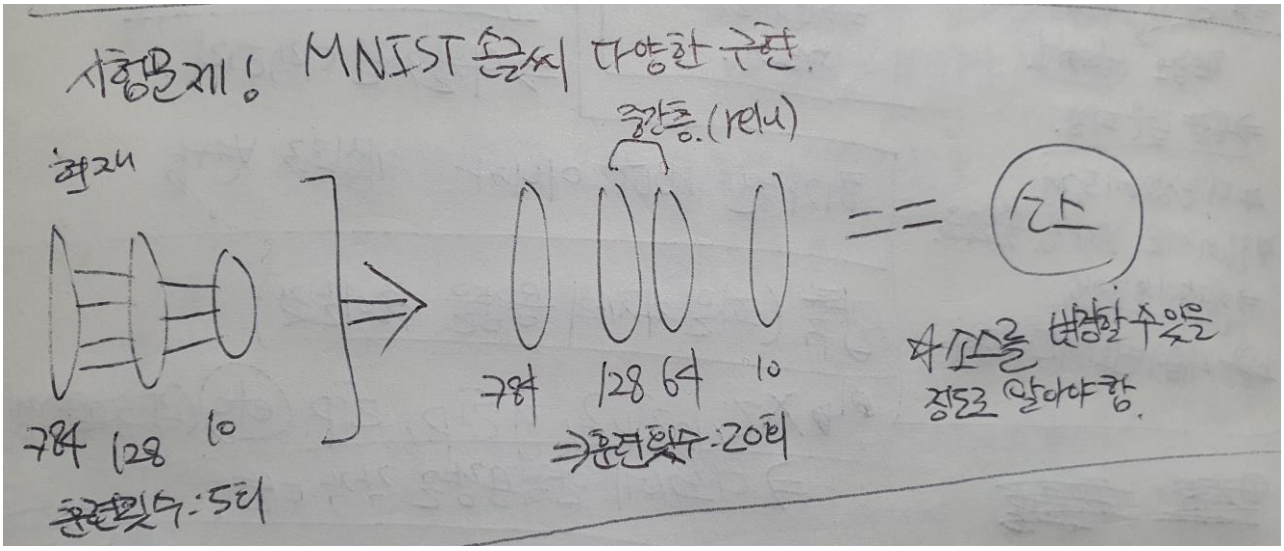
<<예측이 잘못된 20 개 그리기 소스>>

```
from random import sample
import numpy as np
##########################################
#예측 틀린 것 첨자를 저장할 리스트
mispred = []
```

```
#예측한 softmax 의 확률이 있는 리스트 pred_result
pred_result = model.predict(x_test)
#실제 예측한 정답이 있는 리스트 pred_labels
pred_labels = np.argmax(pred_result, axis=1)
for n in range(0, len(y_test)): if pred_labels[n] != y_test[n]: mispred.append(n)
print('정답이 틀린 수', len(mispred))
#랜덤하게 틀린 것 20 개의 첨자 리스트 생성
samples = sample(mispred, 20) print(samples)
#틀린 것 20 개 그리기
count = 0
nrows, ncols = 5, 4
plt.figure(figsize=(12,10))
for n in samples: count += 1
plt.subplot(nrows, ncols, count)
plt.imshow(x_test[n].reshape(28, 28), cmap='Greys', interpolation='nearest')
tmp = "Label:" + str(y_test[n]) + ", Prediction:" + str(pred_labels[n])
plt.title(tmp)
plt.tight_layout()
plt.show()
```



<예제>



- 중간층 2 개, 출력층 – 128 개 뉴런, 64 개 뉴런, 10 개 출력
- 훈련 횟수 20 회 – epochs=20

#층을 차례대로 쌓아 tf.keras.Sequential 모델을 생성

model = tf.keras.models.Sequential([tf.keras.layers.Flatten(input_shape=(28, 28)),
tf.keras.layers.Dense(128, activation='relu'), tf.keras.layers.Dropout(.2),
tf.keras.layers.Dense(64, activation='relu'), tf.keras.layers.Dropout(.2),
tf.keras.layers.Dense(10, activation='softmax')])

#훈련에 사용할 옵티마이저(optimizer)와 손실 함수, 출력정보를 선택

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

#모델 요약 표시

model.summary()

#모델을 훈련 데이터로 총 5 번 훈련

model.fit(x_train, y_train, epochs=20)

<flatten 미사용 가능>
-> reshape()으로 평탄화 작업을 수행한 후 Dense() 층 사용

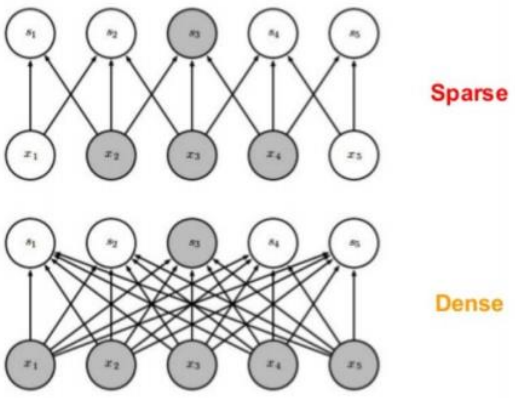
```
import tensorflow as tf

mnist = tf.keras.datasets.mnist (x_train, y_train), (x_test, y_test) = mnist.load_data()

#샘플 값을 정수(0~255)에서 부동소수(0~1)로 변환

x_train, x_test = x_train / 255.0, x_test / 255.0

#먼저 reshape()로 평탄화 작업을 수행한 후

x_train = x_train.reshape((60000, 28*28)) x_test = x_test.reshape((10000, 28*28))

#층을 차례대로 쌓아 tf.keras.models.Sequential 모델을 생성

model = tf.keras.models.Sequential([ #tf.keras.layers.Flatten(input_shape=(28, 28)),
tf.keras.layers.Dense(128, activation='relu', input_shape=(28 * 28,)),
tf.keras.layers.Dropout(0.2), tf.keras.layers.Dense(10, activation='softmax') ])

#모델 요약 표시

model.summary()

#훈련에 사용할 옵티마이저(optimizer)와 손실 함수, 출력정보를 모델에 설정

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy']) #metrics=['accuracy', 'mse'])

#모델을 훈련 데이터로 총 5 번 훈련

model.fit(x_train, y_train, epochs=5)

#모델을 테스트 데이터로 평가

model.evaluate(x_test, y_test)
```

<주요용어>

- 데이터셋 – 훈련용과 테스트용
  - Train data set, Test data set
  - x(입력, 문제), y(정답, 레이블) – 전처리

- 모델 – 딥러닝 핵심 신경망, 여러 층 구성
  - 완전연결층 [Dense() : '중간층이 없는 구조']
  - 1 차원 배열로 평탄화 [Flatten() : 784 개의 긴 배열로 변환]

- 학습 방법의 여러 요소들
  – 옵티마이저(optimizer), 최적화 방법
  - 경사하강법: 내리막 경사 따라 가기
  – 손실 함수(Loss Function)
  - Cross entropy(크로스엔트로피), MSE(Mean Square Error 평균제곱오차)
- 딥러닝 훈련
  – Epochs
  - 총 훈련 횟수, 훈련 데이터를 한번 모두 훈련시키는 것이 1 에폭
- 전체 연결 (모두 연결)(Dense)
- 부분 연결 (Sparse)



- 드롭 아웃 (2012 년 토론토 대학의 힌튼 교수)

  => 층에서 결과 값을 일정 비율로 제거하는 방법

  - 뉴런의 수 ↑, 화살표 ↑, 예측 ↓ → 모델을 단순히하자!

- 오버피팅(overfitting) 문제를 해결하는 정규화(regularization) 목적을 위해서 필요

**과적합문제(overfitting) : 학습 데이터에 지나치게 집중해 실제 Test 에서는 결과가 더 나쁘게 나오는 현상

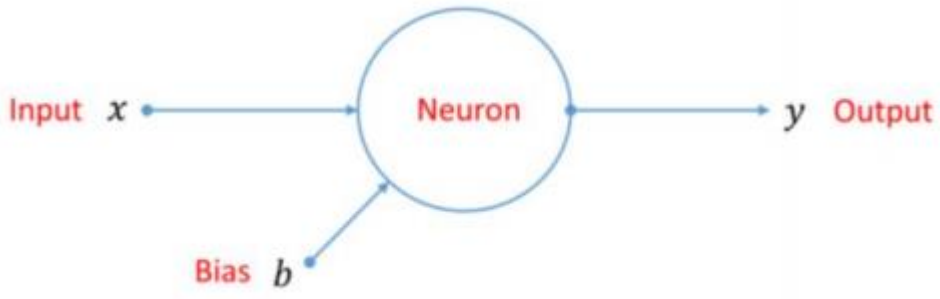
- 훈련 중 페센테이지 만큼 중간에 끊은 후 학습
- tf.keras.layers.Dropout(0.2) – 훈련 중에 20%를 0 으로 지정 – 확률 값은 0.2~0.5 주로사용
- 예측할 때는 모두 사용(어떤 유닛도 드롭아웃하지 않음)

- 활성화함수(activation function)
  - ReLU, Sigm, Tanh

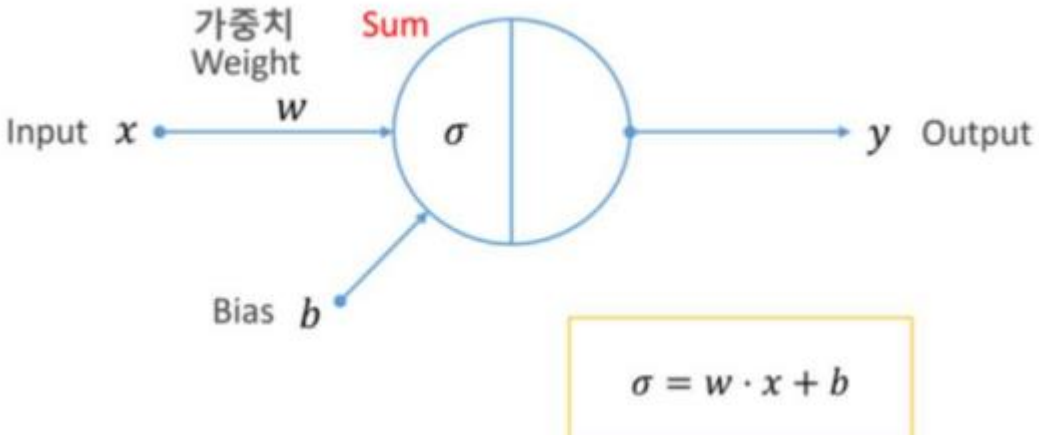
인공신경망 퍼셉트론

인공 신경 세포(Artificial Neuron)

뉴런 (입력[input] / 편향[Bias])

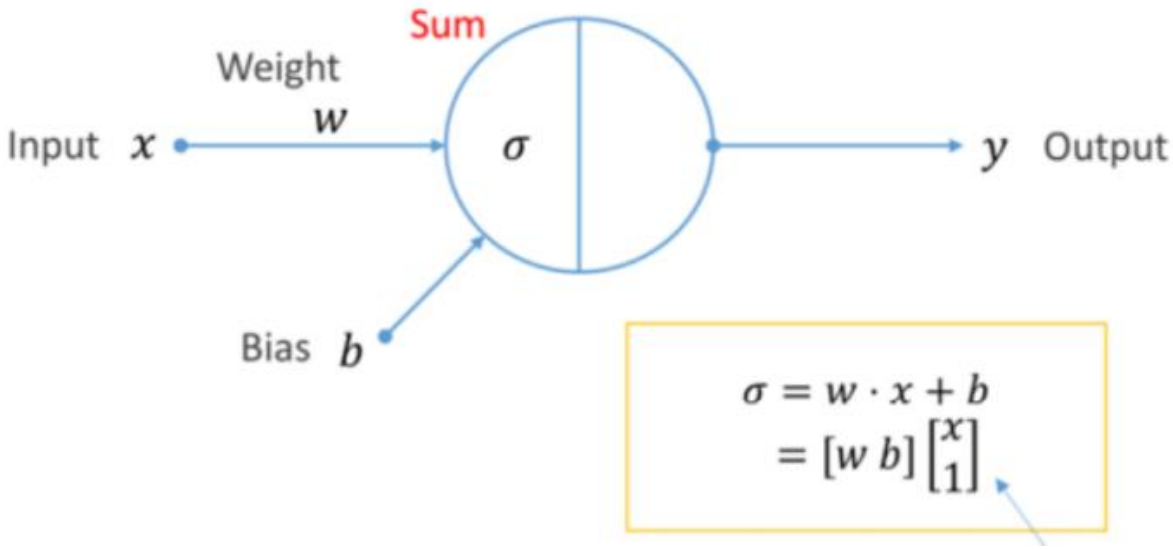


- 편향 : 편향을 조정하여 출력을 맞출 수 있음
- 식 : σ = w*x + b (w: 가중치, b:편향)



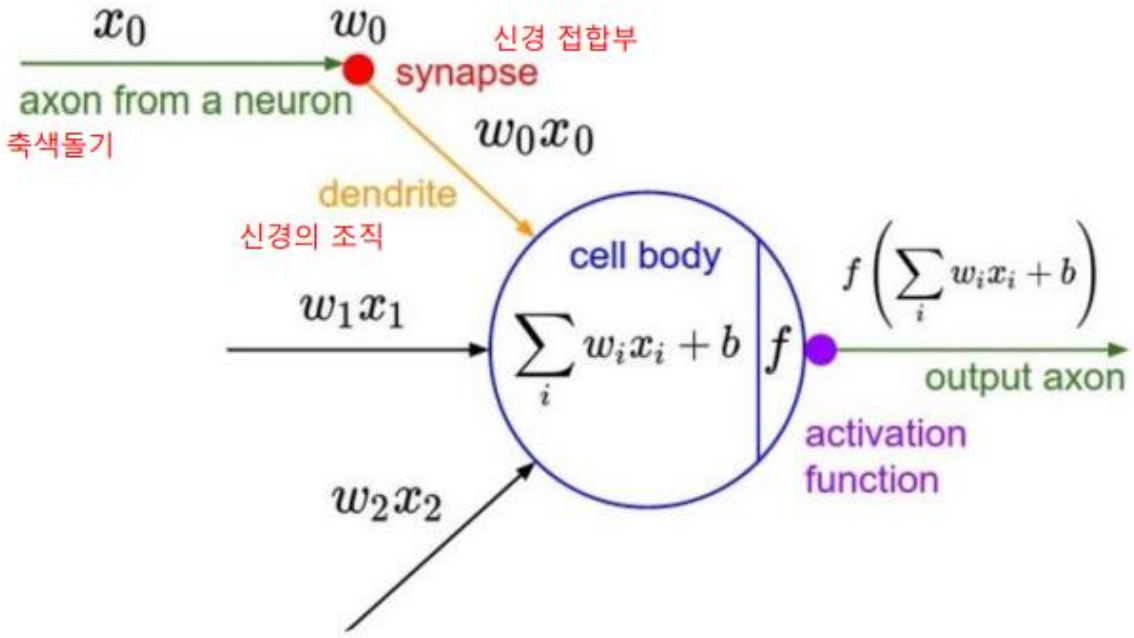
행렬의 곱 연산

식 : σ = w*x + b = [w b] [x 1] (행렬의 곱)

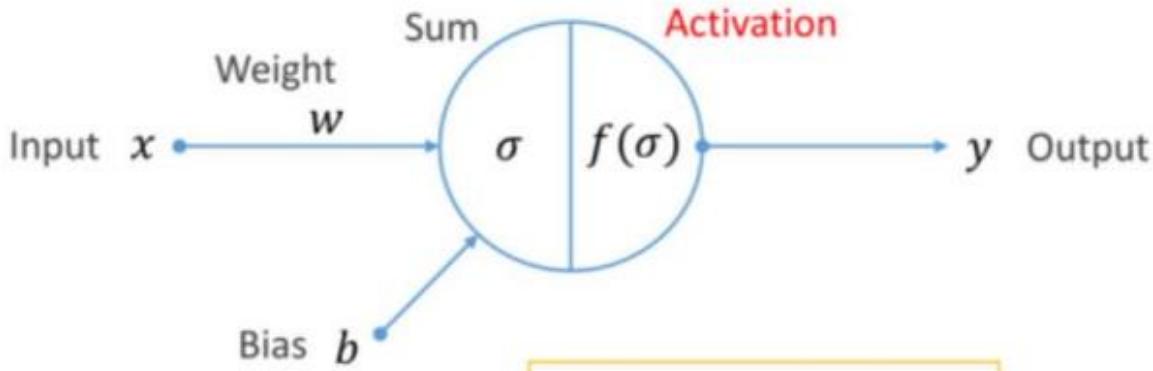


행렬의 곱을 하는 이유 : 입력이 여러개, 출력이 여러개일 경우 있기 때문

일반화된 인공신경망



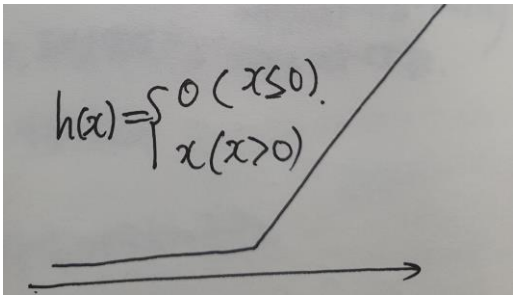
- 활성화 함수 (뉴런의 출력값을 정하는 함수)
- 결과 값이 임계 값 역할
  - 결과가 임계 값 이상이면 활성화
  - 결과가 임계 값 미만이면 비활성화



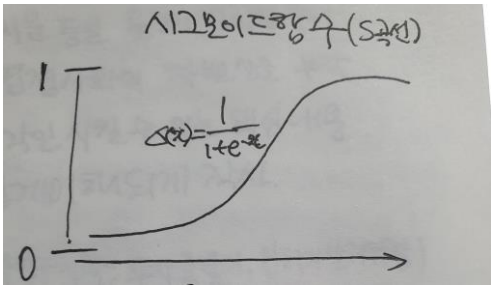
$$\sigma = w \cdot x + b$$

$$f(\sigma) = f(w \cdot x + b)$$

- 동일함수(identity)
  - f(x) = x
- ReLU (정류(수정)된 선형함수)(Rectified Linear Unit)
  - 값의 왜곡이 적어짐
  - max(x, 0) [양수만 사용][0 이하는 모두 0 으로 한 함수]



- Sigmoid (시그모이드)
  - s 자 형태의 곡선이라는 의미
  - 출력 값이 (0~1 사이)



입력의 특징

2 차원의 입력값 [x1, x2, x3 ... ,xn] ⇒ z(스칼라)

3 차원의 입력값 [[x11, x12, x13 ... ,x1n], ... [xs1, xs2, xs3 ... ,xsn]] ⇒ [z1, z2, z3 ... zn] (벡터)

** 뉴런층 : 히든층 ~ 출력층

- 행렬의 다른 표현
  - 입력을 오른쪽 행렬에 배치
  - 가중치는 왼쪽 행렬에 배치
  - 곱의 순서도 변환

활성화 함수 그리기

자연수(오일러 수) : e, 2.71828

- 시그모이드 함수

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

- ReLU 함수

$$h(x) = \begin{cases} 0\ (x \leq 0) \\ x(x > 0) \end{cases}$$

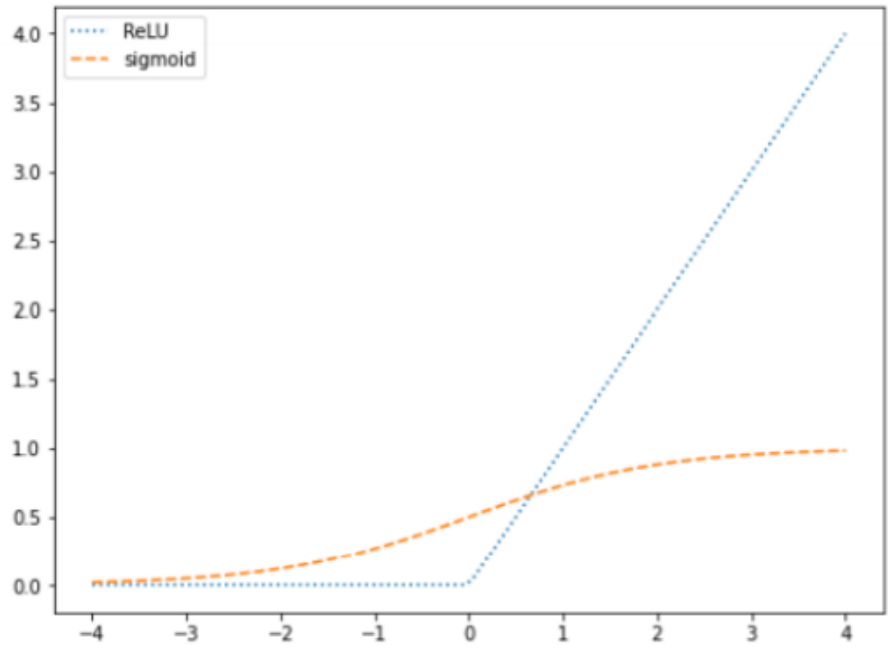
<소스>

```
import numpy as np import matplotlib.pylab as plt

#ReLU(Rectified Linear Unit

#(정류된 선형 유닛) 함수

def relu_func(x): return np.maximum(0, x) #return (x>0)*x # same

def sigm_func(x): # sigmoid 함수

  return 1 / (1 + np.exp(-x))

#그래프 그리기

plt.figure(figsize=(8, 6))

x = np.linspace(-4, 4, 100)

y = np.linspace(-0.2, 2, 100)
```

plt.plot(x, relu_func(x), linestyle=':', label="ReLU")

plt.plot(x, sigm_func(x), linestyle='--', label="sigmoid")

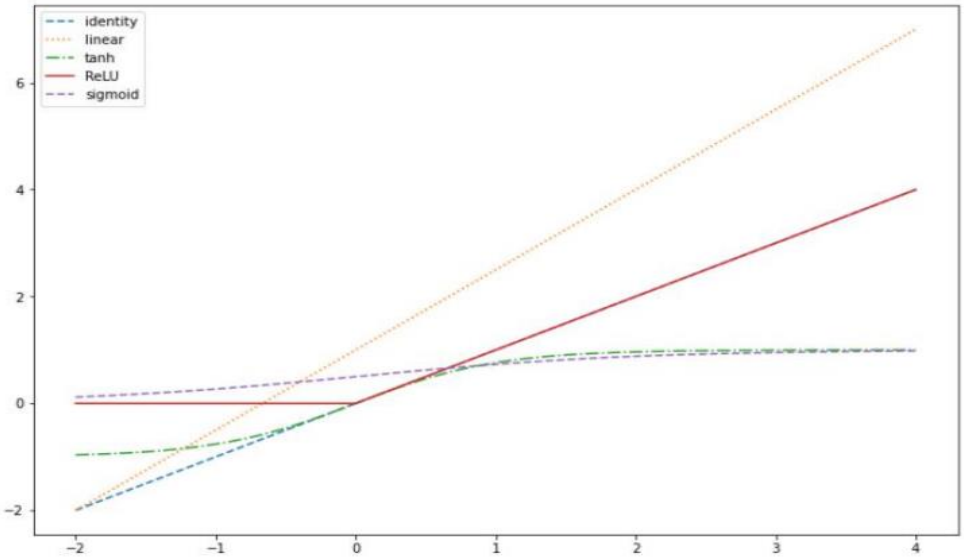
plt.legend(loc='upper left')



<다양한 활성화 함수>

import numpy as np import matplotlib.pylab as plt
def identity_func(x): # 항등함수
return x
def linear_func(x): # 1 차함수
return 1.5 * x + 1 # a 기울기(1.5), Y 절편 b(1) 조정가능
def tanh_func(x): # TanH 함수
return np.tanh(x)
def relu_func(x): # ReLU(Rectified Linear Unit, 정류된 선형 유닛) 함수 return
np.maximum(0, x) #return (x>0)*x # same
def sigm_func(x): # sigmoid 함수
return 1 / (1 + np.exp(-x))
#그래프 그리기

```
plt.figure(figsize=(12, 8)) x = np.linspace(-2, 4, 100)
plt.plot(x, identity_func(x), linestyle='--', label="identity")
plt.plot(x, linear_func(x), linestyle=':', label="linear")
plt.plot(x, tanh_func(x), linestyle='-.', label="tanh")
plt.plot(x, relu_func(x), linestyle='-', label="ReLU")
plt.plot(x, sigm_func(x), linestyle='--', label="sigmoid")
plt.legend(loc='upper left')
```



논리게이트

AND 게이트

구조 : 입력 2 개, 편향, 출력 1 개

구할 값 : 가중치 2 개, 편향 1 개

XOR 게이트

하나의 퍼셉트론으로는 불가능 → 뉴런 3 개의 2 층으로 가능

(구해야 할 총 매개변수 : 3 * 2 + 3 * 1 = 9 개)

Sequential 모델

Dense 층

- 가장 기본적인 층
- 인자 units, activation
  - **뉴런 수와 활성화 함수**
- 인자 input_shape
  - **첫 번째 층에서만 정의**
  - **입력의 차원을 명시**
    - **(2,)**
    - **2개의 입력을 받는 1차원**

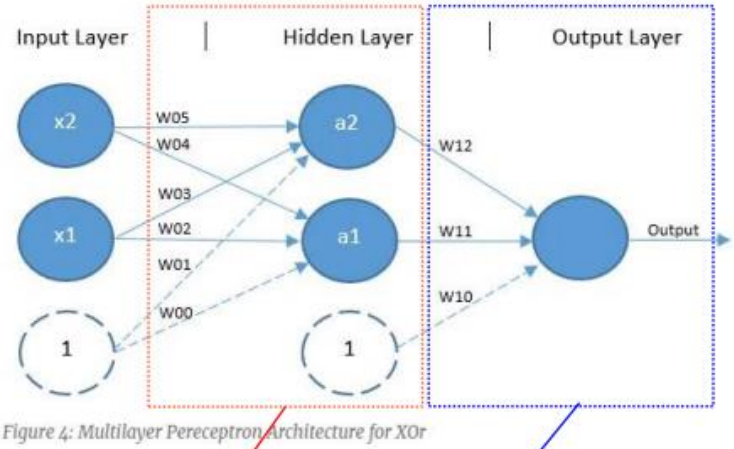


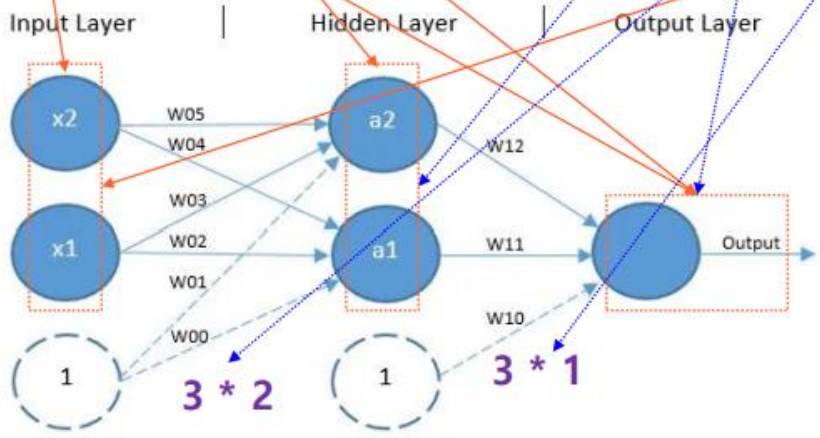
Figure 4: Multilayer Pereceptron Architecture for XOr

```
x = np.array([[1,1], [1,0], [0,1], [0,0]])
y = np.array([[0], [1], [1], [0]])

model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=2, activation='sigmoid', input_shape=(2,)),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])
```

입력, 은닉, 출력 층

- 패러미터 수
  - **(입력측 뉴런 수 + 1) * (출력측 뉴런 수)**

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 2)                 6
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 3
=================================================================
Total params: 9
Trainable params: 9
Non-trainable params: 0
_________________________________________________________________
```



- 패러미터 수 : (입력측 뉴런수 + 1)*(출력측 뉴런 수
- param : 가중치, 편향

회귀와 분류

회귀(regresson) : 연속적인 값을 예측 ex)사용자가 광고 클릭 확률?

- 회귀 분석 : 통계언어

  – 관찰된 연속형 변수들에 대해 두 변수 사이의 모형을 구한 뒤 적합도를 측정해 내는 분석 방법

  – 회귀분석은 시간에 따라 변화하는 데이터나 어떤 영향, 가설적 실험, 인과관계의 모델링 등의 통계적 예측에 이용

- 어원 : 프랜시스 골턴 "평균으로의 회귀"

분류(classification) : 불연속적인 값을 예측 ex) email 이 스팸메일인지? 아닌지?

** 클러스팅링(군집화)와 구분해야 함.

선형회귀

데이터의 경향성을 가장 잘 설명하는 하나의 직선을 예측하는 방법 [Y = aX + b][기울기 a 와 절편인 b 를 구하는 것]

- 단순 선형 회귀분석(Simple Linear Regression Analysis)
  - 입력 : 특징 1 / 출력 : 하나의 값

$$H(x) = Wx + b$$

- 다중 선형 회귀분석(Multiple Linear Regression Analysis)
  - 입력 : 특징 여러개 / 출력 : 하나의 값

$$y = W_1x_1 + W_2x_2 + \ldots W_nx_n + b$$

** 로지스틱 회귀(Logistic Regression)

⇒ 회귀가 아니고, 연속적인 값을 예측하는 것이 아님

- 이진 분류 (클래스가 2 개인 것)
- 입력 : 하나 or 여러개 / 출력 : 0 or 1

인공지능

가중치 w 와 편향 b 를 구하기 (w,b 가 매개변수[parameter])

가설

머신러닝에서 y 와 x 간의 관계를 유추한 식를 가설

→ w,b 를 찾아는 일

- 인공지능 in 가중치 == 일차함수 in 기울기
- 인공지능 in 편향 == 일차함수 in 절편

손실함수

실제 값과 가설로부터 얻은 예측 값의 오차를 계산하는 식(실제 값과 예측 값에 대한 오차에 대한 식)[낮은 값이 좋음]

→ 머신러닝은 w,b 를 찾기위해 손실함수를 정의

- MSE(Mean Square Error 평균제곱오차)

$$\frac{1}{n}\sum_{i}^{n}\left[y_i - H(x_i)\right]^2$$

오차는 실제 데이터(빨간 점)와 예측 선(파란 선)의 차이의 제곱의 합

→ 제곱을 하는 이유 : 오차 중에는 마이너스가 있으니 그냥 더하면 0 이 된다. ⇒ 마이너스를 플러스로!

평균 제곱 오차를 W와 b에 의한 비용 함수(Cost function)로 재정의

$$cost(W,b) = \frac{1}{n}\sum_{i}^{n}[y_i - H(x_i)]^2$$

- 모든 점 들과의 오차가 클수록 평균 제곱 오차는 커지며,
  - **오차가 작아질수록 평균 제곱 오차는 작아짐**

평균 제곱 오차

- cost(W, b)를 최소가 되게 만드는 W와 b를 구하면
  - **결과적으로 y와 x의 관계를 가장 잘 나타내는 직선을 그릴 수 있게 됨**

$$W, b \rightarrow minimize\ cost(W,b)$$

- Categorical crossentropy
- Sparse Categorical crossentropy

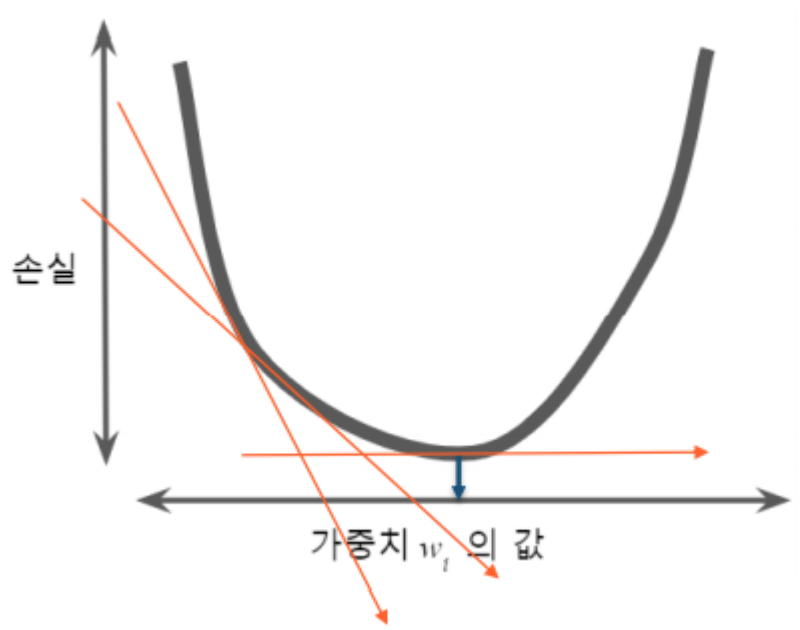
최적화과정 (옵티마이저 : Optimizer)

w,b 를 잘 구하는 것 (최적화 알고리즘)

⇒ 최적화 과정의 알고리즘 : 경사하강법(Gradient Descent)

- 경사하강법

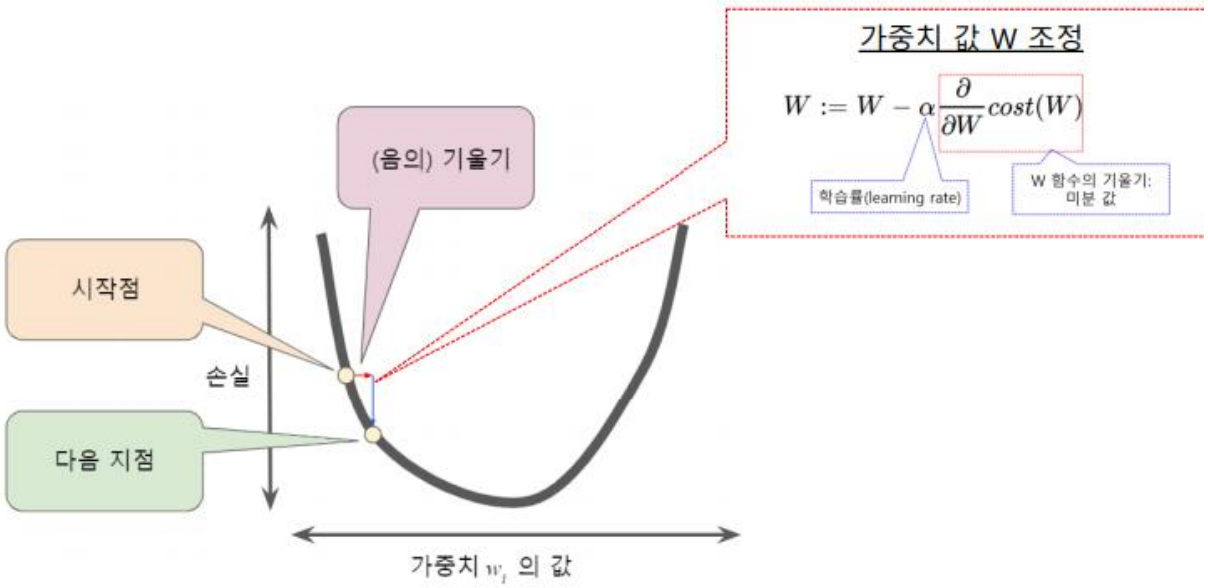
  비용 함수(Cost Function)의 값을 최소로 하는 W 와 b 를 찾는 방법 [경사 따라 내려 오기]



손실과 가중치를 대응한 그림

1. 시작 값(시작점)선택
2. 시작점 별로 중요 x → 0 으로 설정, 임의의 값 in 많은 알고리즘
3. 시작점에서 손실곡선의 기울기 계산 (미분 값)

2. 기울기가 0 인 지점 향해 이동



*현재 기울기가 음수 → 다음 가중치값은 현재보다 크게 조정

- 다음 가중치 결정 방법

- 기울기 * 학습률

  ex)w = w – (-2.5 x 0.01) = w + 0.025
- 학습률 : 시작점에서 다음시작점으로 얼만큼 이동할지 결정 (0.001 ~ 0.1)
- 학습률의 값
  - 작게 설정 → 오랜 시간
  - 크게 설정 → 최저점을 무질서하게 이탈 위험

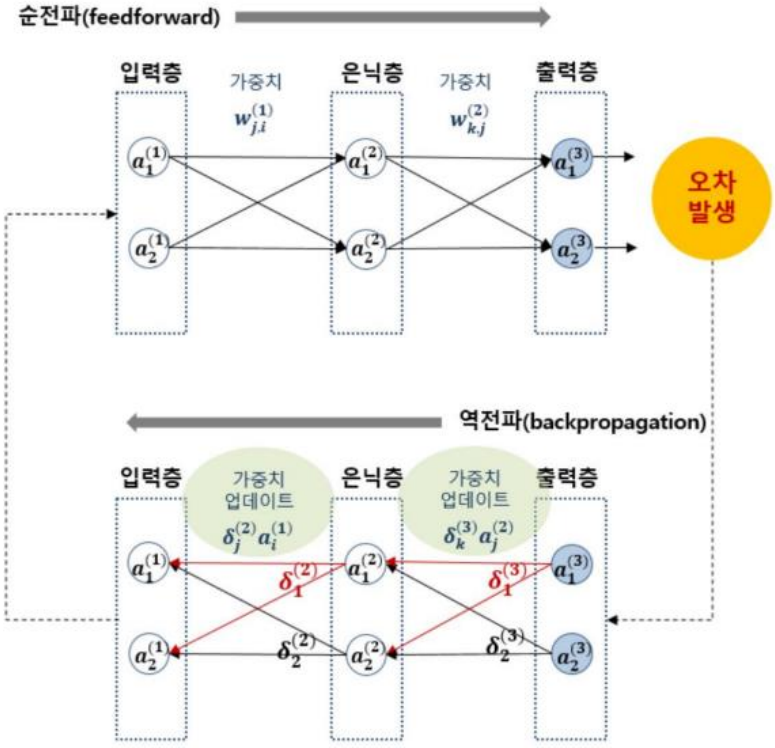
학습률

- 다양한 학습률로 실험 – 이러한 학습률이 손실 곡선의 최저점에 도달하는 데 필요한 단계 수에 어떤 영향을 미치는지 확인

실험, 경험의 의해 사람이 정하는 것 : 하이퍼 패러미터

가중치, 편향 : 일반매개변수(패러미터)

오차역전파



- 순전파 : 새로운 w 를 구해 결과를 봄

- 역전파 : 오차 결과값 이용해 역으로 input 방향으로 오차 적어지게 다시 보내 가중치 수정 → 처리속도 ↑
    - 1986, 제프리 힌튼

케라스로 예측 순서

① 케라스 패키지 임포트

– import tensorflow as tf

– import numpy as np

② 데이터 지정

– x = numpy.array([0, 1, 2, 3, 4])

– y = numpy.array([1, 3, 5, 7, 9]) #y = x * 2 + 1

③ 인공신경망 모델 구성

– model = tf.keras.models.Sequential()

– model.add(tf.keras.layers.Dense(출력수, input_shape=(입력수,)))

④ 최적화 방법과 손실 함수 지정해 인공신경망 모델 생성

– model.compile(' SGD ' , ' mse ')

⑤ 생성된 모델로 훈련 데이터 학습

– model.fit(...)

⑥ 성능 평가

– model.evaluate(...)

⑦ 테스트 데이터로 결과 예측

– model.predict(...)

<예제>

[선형회귀 y=2x 예측]

훈련 데이터 : x_train=[1,2,3,4] y_train=[2,4,6,8]
테스트 데이터 : x_test = [1.2, 2.3, 3.4, 4.5] y_test = [2.4, 4.6, 6.8, 9.0]
예측, 다음 x 에 대해 예측되는 y 를 출력
[3.5, 5, 5.5, 6]

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

① 문제와 정답 데이터 지정
x_train = [1, 2, 3, 4] y_train = [2, 4, 6, 8]

② 모델 구성(생성)
model = Sequential([ Dense(1, input_shape=(1, ), activation='linear') #Dense(1,
input_dim=1) ])

③ 학습에 필요한 최적화 방법과 손실 함수 등 지정
# 훈련에 사용할 옵티마이저(optimizer)와 손실 함수, 출력정보를 선택
# Mean Absolute Error, Mean Squared Error
model.compile(optimizer='SGD', loss='mse', metrics=['mae', 'mse'])
# 모델을 표시(시각화)
model.summary()

④ 생성된 모델로 훈련 데이터 학습
model.fit(x_train, y_train, epochs=1000)

⑤ 테스트 데이터로 성능 평가
x_test = [1.2, 2.3, 3.4, 4.5] y_test = [2.4, 4.6, 6.8, 9.0] print('정확도:', model.evaluate(x_test,
y_test)) print(model.predict([3.5, 5, 5.5, 6]))
```

[선형회귀 y=2x 예측]

```
import tensorflow as tf
import numpy as np #훈련과 테스트 데이터
x = np.array([0, 1, 2, 3, 4]) y = np.array([1, 3, 5, 7, 9]) #y = x * 2 + 1

#인공신경망 모델 사용
model = tf.keras.models.Sequential()

#은닉계층 하나 추가
model.add(tf.keras.layers.Dense(1, input_shape=(1,)))

#모델의 패라미터를 지정하고 모델 구조를 생성
#최적화 알고리즘: 확률적 경사 하강법(SGD: Stochastic Gradient Descent)
#손실 함수(loss function): 평균제곱오차(MSE: Mean Square Error)
model.compile('SGD', 'mse')

#생성된 모델로 훈련 자료로 입력(x[:2])과 출력(y[:2])을 사용하여 학습
#키워드 매개변수 epoch(에폭): 훈련반복횟수
#키워드 매개변수 verbose: 학습진행사항 표시
model.fit(x[:3], y[:3], epochs=1000, verbose=0)

#테스트 자료의 결과를 출력
print('Targets(정답):', y[3:])

#학습된 모델로 테스트 자료로 결과를 예측(model.predict)하여 출력
print('Predictions(예측):', model.predict(x[3:]).flatten())
```

[2018 년 대한민국 인구증가율과 고령인구비율]

인구증가율과 고령인구비율

X = [0.3, -0.78, 1.26, 0.03, 1.11, 0.24, -0.24, -0.47, - 0.77, -0.37, -0.85, -0.41, -0.27, 0.02, -0.76, 2.66]

Y = [12.27, 14.44, 11.87, 18.75, 17.52, 16.37, 19.78, 19.51, 12.65, 14.74, 10.72, 21.94, 12.83, 15.51, 17.14, 14.42]

활성화 함수 tanh(하이퍼볼릭 타젠트) [S 자 곡선][-1,1]

$$\tanh(z) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad \sigma(x) = \frac{1}{1 + e^{-x}}.$$

중간층 : 6 개, 출력층 : 1 개

4.7 딥러닝 네트워크를 이용한 회귀

```
import tensorflow as tf
import numpy as np

# 인구증가율과 고령인구비율
X = [0.3, -0.78, 1.26, 0.03, 1.11, 0.24, -0.24, -0.47, -0.77, -0.37, - 0.85, -0.41, -0.27, 0.02, -0.76, 2.66]
Y = [12.27, 14.44, 11.87, 18.75, 17.52, 16.37, 19.78, 19.51, 12.65, 14.74, 1 0.72, 21.94, 12.83, 15.51, 17.14, 14.42]

model = tf.keras.Sequential([ tf.keras.layers.Dense(units=6, activation='tanh', input_shape=(1,)), tf.keras.layers.Dense(units=1) ])

model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.1), loss='mse') model.summary()

#4.8 딥러닝 네트워크의 학습
model.fit(X, Y, epochs=10)

#4.9 딥러닝 네트워크의 Y 측 예측
```

model.predict(x)

#4.10 딥러닝 네트워크의 회귀선 확인

Import matplotlib.pyplot as plt

line_x = np.arange(min(X), max(X), 0.01)

line_y = model.predict(line_x)

plt.plot(line_x, line_y, 'r-')

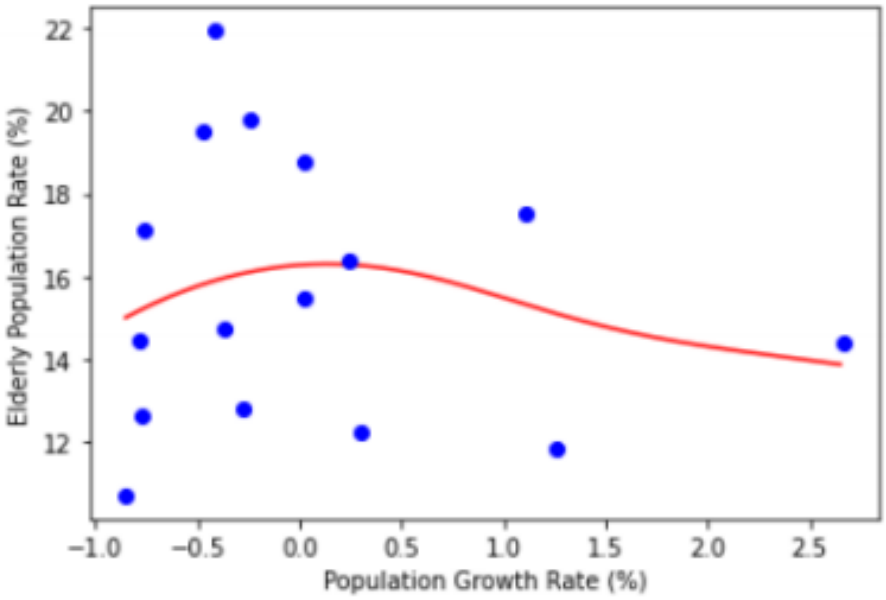
plt.plot(X,Y,'bo')

plt.xlabel('Population Gowth Rate (%)')

plt.ylabel('Elderly Population Rate (%)')

plt.show()

```
Epoch 1/10
1/1 [==============================] - 0s 1ms/step - loss: 256.3840
Epoch 2/10
1/1 [==============================] - 0s 824us/step - loss: 116.0593
Epoch 3/10
1/1 [==============================] - 0s 926us/step - loss: 9.6935
Epoch 4/10
1/1 [==============================] - 0s 952us/step - loss: 9.3609
Epoch 5/10
1/1 [==============================] - 0s 2ms/step - loss: 9.2970
Epoch 6/10
1/1 [==============================] - 0s 963us/step - loss: 9.2382
Epoch 7/10
1/1 [==============================] - 0s 991us/step - loss: 9.1770
Epoch 8/10
1/1 [==============================] - 0s 908us/step - loss: 9.1145
Epoch 9/10
1/1 [==============================] - 0s 911us/step - loss: 9.0526
Epoch 10/10
1/1 [==============================] - 0s 899us/step - loss: 8.9924
<tensorflow.python.keras.callbacks.History at 0x7f9ab36daa58>
array([[16.27147 ],
       [15.190466],
       [15.10627 ],
       [16.290047],
       [15.327049],
       [16.291916],
       [16.117289],
       [15.818182],
       [15.214785],
       [15.965492],
       [15.012625],
       [15.909767],
       [16.086227],
       [16.287073],
       [15.238832],
       [13.875053]], dtype=float32)
```



[케라스 모델 미사용 텐서플로 프로그래밍]

optimizer – 최적화 과정(복잡한 미분 계산 및 가중치 수정)을 자동으로 진행

• SGD, adam

학습률(learning rate) – 보통 0.1 ~ 0.0001

**변수 Variables

▶ 프로그램에 의해 변화하는 공유된 지속 상태를 표현하는 가장 좋은 방법

- 하나의 텐서를 표현
- 텐서 값은 텐서에 연산을 수행하여 변경 가능

– 모델 파라미터를 저장하는데 tf.Variable 을 사용

– 변수 생성 : 초기값을 설정

```
# a 와 b 를 랜덤한 값으로 초기화합니다.
# a = tf.Variable(random.random())
# b = tf.Variable(random.random())
a = tf.Variable(tf.random.uniform([1], 0, 1))
b = tf.Variable(tf.random.uniform([1], 0, 1))
```

** minimize()

▶ 첫번째 인자 : 최소화할 손실 함수

▶ 두번째 인자 : 학습할 변수 리스트, 가중치와 편향

#1000 번의 학습을 거쳐서 잔차의 제곱 평균을 최소화하는 적절한 값 a, b 에 도달
for i in range(1000): # 잔차의 제곱의 평균을 최소화(minimize)합니다.
optimizer.minimize(compute_loss, var_list=[a,b])

# 4.4 텐서플로를 이용해서 회귀선 구하기
import tensorflow as tf
import numpy as np
# import random

X = [0.3, -0.78, 1.26, 0.03, 1.11, 0.24, -0.24, -0.47, -0.77, -0.37, -0.85, -0.41, - 0.27, 0.02, -0.76, 2.66]
Y = [12.27, 14.44, 11.87, 18.75, 17.52, 16.37, 19.78, 19.51, 12.65, 14.74, 10.72, 21. 94, 12.83, 15.51, 17.14, 14.42]

# a 와 b 를 랜덤한 값으로 초기화합니다.
# a = tf.Variable(random.random())
# b = tf.Variable(random.random())
a = tf.Variable(tf.random.uniform([1], 0, 1))
b = tf.Variable(tf.random.uniform([1], 0, 1))

# 잔차의 제곱의 평균을 반환하는 함수입니다.
def compute_loss():
    y_pred = a * X + b
    loss = tf.reduce_mean((Y - y_pred) ** 2)
    return loss

optimizer = tf.keras.optimizers.Adam(lr=0.07)
for i in range(1000): # 잔차의 제곱의 평균을 최소화(minimize)합니다.
    optimizer.minimize(compute_loss, var_list=[a,b])
    if i % 100 == 99:
        print(i, 'a:', a.numpy(), 'b:', b.numpy(), 'loss:', compute_loss().numpy())

[보스톤 주택 가격 예측]

• 주요 활성화 함수 : ReLU, Sigmoid, Tanh

• 1978 년 보스톤 지역 주택 가격 데이터 셋

[타운의 주택 가격 중앙 값, 천 달러 단위, 범죄율, 방 수, 고속도로까지 거리 등]

[13 가지 특성]

– 506 개 : 학습 데이터 [404 개] / 테스트 데이터 [102 개]

**정규화가 필요 (why? 특성의 단위가 다르기 때문)

정규화 방법

– 학습 데이터: (train_Xi - 학습데이터평균) / 학습데이터 표준편차 [정규 분포를 가정] –

테스트 데이터 : (test_Xi – 학습데이터평균) / 학습데이터 표준편차

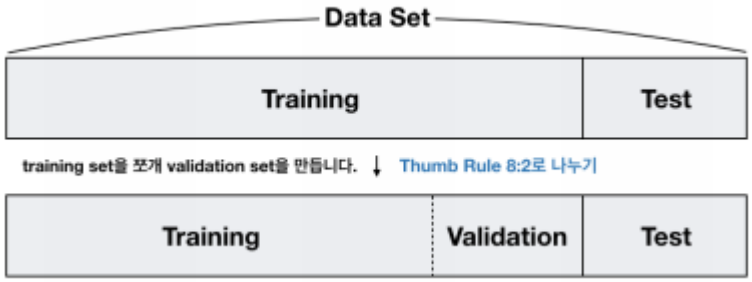
[테스트데이터가 정규 분포를 가정할 수 없음]

• 중간층 : 3 개 층

• 출력층 : 1 개 층[회귀 모델, 주택 가격이므로]

• 최적화 – 학습률 [lr=0.07] , 손실 함수 [mse]

• 훈련 데이터 404 개 중 일부를 검증 데이터로 사용



** validation_split: – 검증 용 데이터의 비율

** 훈련:검증 == 80%:20% 비중

** batch_size : 훈련에서 가중치와 편향의 패러미 터를 수정하는 데이터 단위 수

** train_size : 훈련 데이터 수

검증 데이터 손실 : 일반적으로 loss 는 꾸준히 감소

[val_loss : 일반적으로 loss 보다 높음(항상 감소하지 않음)]

평가 결과 : 손실값(작을수록 좋은 결과)

검증 손실(val_loss)가 적을수록 테스트 평가의 손실도 적음)
검증 데이터에 대한 성적이 좋게 유도
-> 과적합에 의해 검증 손실 ↑ ⇒ 학습이 중단되도록 지정 (callbacks 사용)
history = model.fit(train_X, train_Y, epochs=25, batch_size=32, validation_split=0.25,
callbacks=[tf.keras.callbacks.EarlyStopping(patience=3, monitor='val_loss')])

일찍 멈춤 기능
– tf.keras.callbacks.EarlyStopping
- monitor='val_loss' – 지켜볼 기준 값이 검증 손실
- patience=3 : 3 회의 epochs 를 실행하는 기준 값이 동안 최고 기록을 갱신하지 못하면(더 낮아지지 않으면) 학습을 멈춤

[자동차 연비 데이터(auto mpg)로 회귀 분석]

** 회귀 v/s 분류
회귀 : 가격 or 확률 같이 연속적인 출력값을 예측하는 것이 목적
분류 : 여러 개의 클래스 중 하나의 클래스를 선택하는 것이 목적

자동차 연비를 예측하는 모델
– Auto MPG 데이터 셋을 사용
– 1970 년대 후반과 1980 년대 초반의 데이터
– 이 기간에 출시된 자동차 정보를 모델에 제공
• 실린더 수, 배기량, 마력(horsepower), 공차 중량 같은 속성
– 이 예제는 tf.keras API 를 사용
• Mpg – Mile per gallon, 연비, km/l

데이터 셋 (속성 8 개)
1. mpg: continuous
2. cylinders: multi-valued discrete
3. displacement: continuous
4. horsepower: continuous
5. weight: continuous
6. acceleration: continuous
7. model year: multi-valued discrete

8. origin: multi-valued discrete

9. car name: string (unique for each instance): 없음

시본의 산점도

- 대각선 diag_kind (커널밀도추정곡선)
    - kde : Kernel Density Estimation
- 색상 : palette='bright' [pastel, bright, deep, muted, colorblind, dark]

sns.pairplot(train_dataset[["MPG", "Cylinders", "Displacement", "Weight"]], diag_kind="kde")

**데이터 정규화

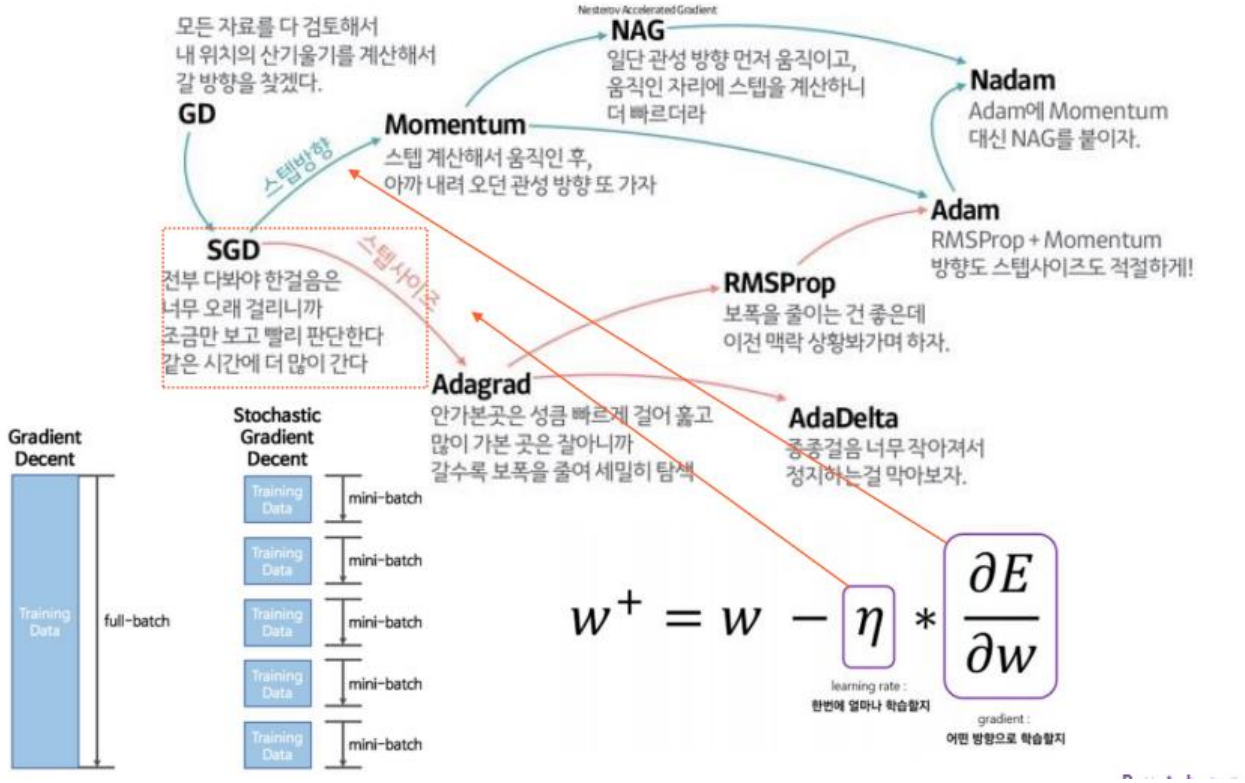
• 특성의 스케일과 범위가 다르면 정규화(normalization)하는 것이 권장

– 의도적으로 훈련 세트만 사용하여 통계치를 생성

• 테스트 세트를 정규화할 때에도 훈련 데이터의 평균과 표준편차 사용

훈련데이터의 평균과 표준편차를 사용하는 Why? 테스트 세트를 모델이 훈련에 사용했던 것과 동일한 분포로 투영하기 위해서

**옵티마이저 발전 과정



**콜백
학습 과정의 한 에폭마다 적용할 함수의 세트
– 학습의 각 단계에서 콜백의 적절한 메서드가 호출
– 모델의 내적 상태와 통계자료를 확인
– 키워드 인수 callbacks
    • Sequential 이나 Model 클래스의 .fit() 메서드에 전달이 가능

** EarlyStopping 콜백(callback)
 – model.fit 메서드를 수정하여 검증 점수가 향상 되지 않으면 자동으로 훈련을 멈춤
• 지정된 에포크 횟수 동안 성능 향상이 없으면 자동으로 훈련이 멈춤
• 옵션 monitor, patience – 손실 val_loss 가 10 회 초과해 감소하지 않으면 중단

** 메소드 axis()
x,y 축의 범위를 설정할 수 있게 하는 것과 동시에 여러 옵션을 설정할 수 있는 함수

```
# 필요 모듈 가져오기
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# 데이터 가져오기 (UCI 머신러닝 저장소에서 다운로드)
dataset_path = keras.utils.get_file("auto-mpg.data",
"http://archive.ics.uci.edu/ml/machine-learning-databases/autompg/auto-mpg.data")
print(dataset_path)

# 데이터 읽어 dataset 에 저장 (판다스를 사용하여 데이터 읽기)
col_names = ['MPG','Cylinders','Displacement','Horsepower','Weight', 'Acceleration',
'Model Year', 'Origin']
raw_data = pd.read_csv(dataset_path, names=col_names, na_values = "?", comment='₩t',
sep=" ", skipinitialspace=True)
```

```
#comment : 특정문자가 있는 행은 주석으로 간주하고 읽지 않고 건너뜀
dataset = raw_data.copy()
dataset.tail(10)

dataset.shape
#데이터 정제, 비어있는 열의 행의 수 알아내기
dataset.isna().sum()

#비어있는 열이 하나라도 있는 행을 제거
dataset = dataset.dropna()
dataset.shape

#열 'Origin'을 빼내 origin 에 저장
origin = dataset.pop('Origin')

#새로운 열 3 개 추가
#생산국(미국, 유럽, 일본)에 따른 열 추가
# "Origin"열은 수치형이 아니고 범주형이므로 원-핫 인코딩으로 변환
dataset['USA'] = (origin == 1) *1.0
dataset['Europe'] = (origin == 2) *1.0
dataset['Japan'] = (origin == 3) *1.0
dataset.tail()

# 데이터셋을 훈련 세트와 테스트 세트로 분할
# 전체 자료에서 80%를 훈현 데이터로 사용
train_dataset = dataset.sample(frac=0.8, random_state=0)
print(train_dataset)
# 전체 자료에서 나머지 20%를 테스트 데이터로 사용
test_dataset = dataset.drop(train_dataset.index)
print(test_dataset)

#전반적인 통계도 확인
train_stats = train_dataset.describe()
```

```
print(train_stats)

train_stats.pop("MPG")
train_stats = train_stats.transpose()
train_stats.head(9)

#정답인 MPG 추출
#레이블을 만들고 원 데이터 집합에서 제거
train_labels = train_dataset.pop('MPG')
test_labels = test_dataset.pop('MPG')

# 정규화된 데이터를 사용하여 모델을 훈련 – 입력 데이터를 정규화하기 위해 사용한 통계치(평균과 표준편차)는 원-핫 인코딩과 마찬가지로 모델에 주입되는 모든 데이터에 적용
def norm(x):
        return (x – train_stats['mean']) / train_stats['std']
normed_train_data=norm(train_dataset)
normed_test_data=norm(test_dataset)
normed_train_data.tail()

#모델 구성
#두 개의 완전 연결(densely connected) 은닉층으로 Sequential 모델
#출력 층은 하나의 연속적인 값을 반환
#나중에 두 번째 모델을 만들기 쉽도록 build_model 함수로 모델 구성 단계를 감쌈
def build_mode():
        model = keras.Sequent([
                layers.Dense(64,activation='relu',input_shape=[len(train_dataset.keys())]),
                layers.Dense(64,activation='relu'),
                layers.Dense(1)
        ])
        optimizer=tf.keras.optimizers.RMSprop(0.001)
        model.compole(loss='mse', optimizer=optimizer, metrics=['mae','mse'])
        return model
```

```
model=build_model()
model.summary()

#모델을 한번 실행
#훈련 세트에서 10 샘플을 하나의 배치로 만듬 (model.predict 메서드를 호출)
example_batch = normed_train_data[:10]
example_result = model.predict(example_batch)
print(example_result)

#모델 훈련
# 1,000 번의 에포크(epoch) 동안 훈련 – 훈련 정확도와 검증 정확도는 history 객체에 기록
# 에포크 중간중간에 점을 출력해 훈련 진행과정을 표시
# 에포크가 끝날 때마다 점(.)을 출력, 100 번마다 다음 줄로 이동해 훈련 진행 과정을 표시
Class PrintDot(keras.callbacks.Callback)"
        def on_epoch_end(self,epoch,logs):
                if epoch % 100 == 0 : print('')
                print('.', end ='')

EPOCHS =1000

history = model.fit(normed_train_data, train_labels, epochs =EPOCHS, validation_split = 0.2, verbose=0, callbacks=[PrintDot()])

hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
hist.tail(10)

#검증 손실이 계속 감소하는 것이 중요
#patience 매개변수는 성능 향상을 체크할 에포크 횟수
early_stop=keras.callbacks.EarlyStopping(monitor='val_loss',patience=10)
```

history = model.fit(normed_train_data, train_labels, epochs =EPOCHS, validation_split = 0.2, verbose=0, callbacks=[PrintDot()])

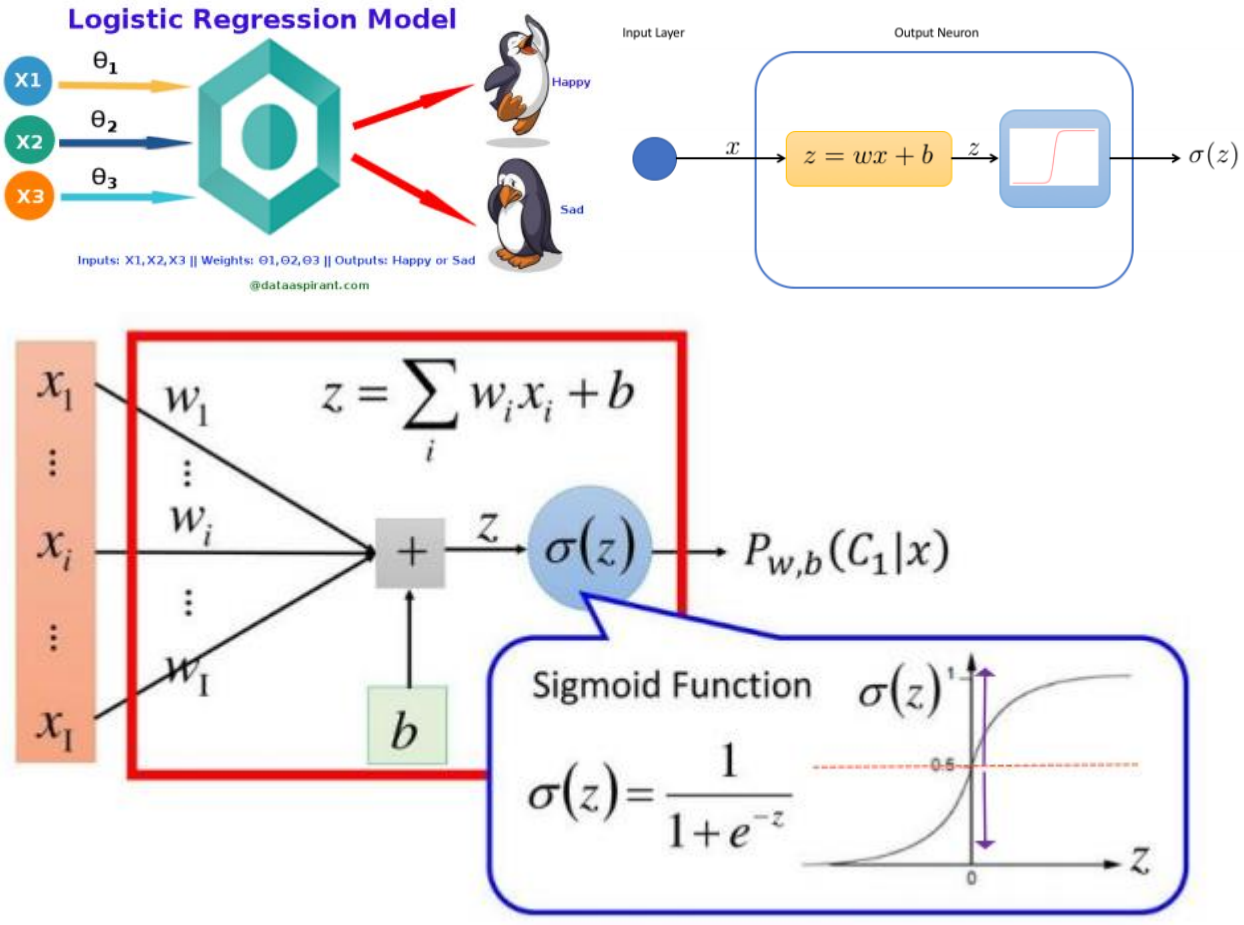
plot_history(history)

#모델성능을 확인
#테스트세트에서 모델의 성능을 확인
loss, mae, mse = model.evaluate(normd_test_data, test_labels, verbose=2)

이항 분류 및 다항 분류

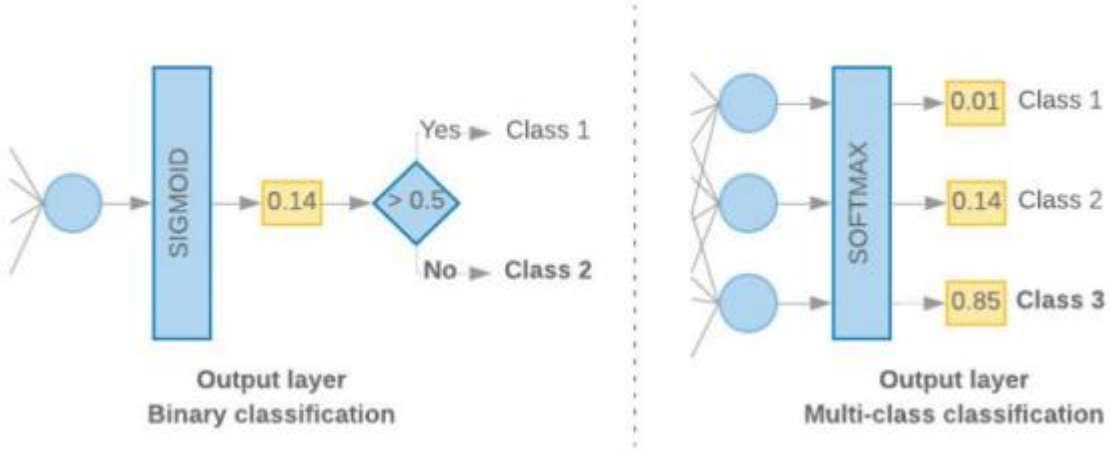
이진 분류 (로지스틱 회귀)

두 가지로 분류하는 방법 ex) PASS / FAIL , SPAM / HAM , 긍정 / 부정



- 시그모이드 함수 : 이진분류 모델의 출력층에 주로 사용되는 활성화 함수 (0 과 1 사이의 값)
- 소프트맥스 함수 : 분류의 마지막 활성화 함수로 사용
  (모든 yi 의 합은 1)(각각의 yi 는 그 분류의 확률)

**시그모이드 함수 v/s 소프트 맥스 함수 (이진 분류 v/s 다중 분류)



대표적인 다중 분류 ex) MNIST 손글씨

분류에서의 손실 함수

크로스 엔트로피(비용함수)(Cross entropy) – 실제 데이터의 결과 값인 y

$$H(p,q) = -\sum_{x} p(x)\ \log q(x).$$

p(x): 실제 분류 값, q(x)는 softmax 결과값(Y)

- y=1 일 때
  - 예측 값이 1 에 가까워질수록 cost function 의 값 ↓
  - 예측 값이 0 에 가까워질수록 cost function 의 값이 ∞ ↑ => 예측이 틀림
- y= 0 일 때
  - 예측이 0 으로 맞게 되면 cost function 은 매우 작은 값을 가짐
  - 예측이 1 로 하게 되어 예측에 실패할 경우 cost 값이 ∞ ↑ => 예측이 틀림

tf.keras.losses.categorical_crossentropy

정답 : y_true = [[0, 1, 0], [0, 0, 1]]

예측 : y_pred = [[0.05, 0.95, 0], [0.1, 0.8, 0.1]]

함수 적용 : loss = tf.keras.losses.categorical_crossentropy(y_true, y_pred)

결과 : loss.numpy()

일반 값 사용 크로스 엔트로피 손실 함수

tf.keras.losses.categorical_crossentropy – 정답: 원 핫 인코딩 유형

tf.keras.losses.sparse_categorical_crossentropy – 정답: 일반 유형

<이항 분류 : 레드 와인과 화이트 와인 구분>

캘리포니아 어바인 대학 제공

특징 12 개

['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol', 'quality']

지수승 e x 효과 : 자연수 e 를 밑으로 하는 지수 함수

-> 음수는 양수로, 작은 수는 작게, 큰 수는 더욱 크게

```
import matplotlib.pyplot as plt
import math
import numpy as np
x = np.arange(-2, 2, 0.01)
e_x = math.e ** x
plt.axhline(0, color='gray')
plt.axvline(0, color='gray')
plt.plot(x, x, 'b-', label='y=x')
plt.plot(x, e_x, 'g.', label='y=e^x')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```

```
# 5.1 와인 데이터셋 불러오기
import pandas as pd red = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learningdatabases/wine-quality/winequality-red.csv', sep=';')
white = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learningdatabases/wine-quality/winequality-white.csv', sep=';')
print(red.head())
print(white.head())

# 5.2 와인 데이터셋 합치기
# 레드와인 : 0, 화이트와인 : 1
```

```
# 열 type 추가
red['type'] = 0
white['type'] = 1
print(red.head(2))
print(white.head(2))

wine = pd.concat([red, white])
print(wine.describe())

# 5.3 레드 와인과 화이트 와인 type 히스토그램
import matplotlib.pyplot as plt
plt.hist(wine['type'])
plt.xticks([0, 1])
plt.show()
print(wine['type'].value_counts())

# 5.5 데이터 정규화
# 최소 0, 최대 1
wine_norm = (wine - wine.min()) / (wine.max() - wine.min())
print(wine_norm.head())
print(wine_norm.describe())

# 레드와인, 화이트와인 행 섞기
# 5.6 데이터 섞은 후 numpy array 로 변환
import numpy as np
wine_shuffle = wine_norm.sample(frac=1)
print(wine_shuffle.head())
wine_np = wine_shuffle.to_numpy()
print(wine_np[:5])

# 5.7 train 데이터와 test 데이터로 분리
# 특징에서 마지막 값 -> 정답 -> 원핫 인코딩
import tensorflow as tf
```

```
train_idx = int(len(wine_np) * 0.8)

train_X, train_Y = wine_np[:train_idx, :-1], wine_np[:train_idx, -1]
#첫번째 -1 : 정답제외 / 두번째 -1 : 정답만
test_X, test_Y = wine_np[train_idx:, :-1], wine_np[train_idx:, -1]
print(train_X[0])
print(train_Y[0])
print(test_X[0])
print(test_Y[0])
train_Y = tf.keras.utils.to_categorical(train_Y, num_classes=2)
test_Y = tf.keras.utils.to_categorical(test_Y, num_classes=2)
print(train_Y[0])
print(test_Y[0])

y = [0, 1, 2, 3]
tf.keras.utils.to_categorical(y, num_classes=4)

# 5.8 와인 데이터셋 분류 모델 생성
# 마지막 층은 소프트맥스 함수 (결과 총합=1, 큰 값을 강조, 작은 값은 약화)
import tensorflow as tf
model = tf.keras.Sequential([
        tf.keras.layers.Dense(units=48, activation='relu', input_shape=(12,)),
        tf.keras.layers.Dense(units=24, activation='relu'), tf.keras.layers.Dense(units=12,
        activation='relu'),
        tf.keras.layers.Dense(units=2, activation='softmax') ])
model.compile(optimizer = tf.keras.optimizers.Adam(lr=0.07), l
        oss='categorical_crossentropy', metrics=['accuracy'])
model.summary()

# 5.9 와인 데이터셋 분류 모델 학습
history = model.fit(train_X, train_Y, epochs=25, batch_size=32, validation_split=0.25)

# 5.10 분류 모델 학습 결과 시각화
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], 'g-', label='accuracy')
plt.plot(history.history['val_accuracy'], 'k--', label='val_accuracy')
plt.xlabel('Epoch')
plt.ylim(0.7, 1)
plt.legend()
plt.show()

# 5.11 분류 모델 평가
model.evaluate(test_X,test_Y)
```

<다항 분류 : 와인 품질 분류>

와인 데이터 셋의 'quality'

등급 3~9 :이 모든 등급을 예측하기에는 등급에 따른 데이터 수 차이가 큼

-> 다시 등급을 3 개 정도로 나누어 예측

```
# 새로운 등급인 new_quality 를 생성
# 조건에 맞는 값을 새로운 열에 추가 df.loc[data['컬럼']조건, '새로운 컬럼명'] = '값'
# 5.14 품질을 3 개의 범주(좋음, 보통, 나쁨)로 재분류
wine.loc[wine['quality'] <= 5, 'new_quality'] = 0
wine.loc[wine['quality'] == 6, 'new_quality'] = 1
wine.loc[wine['quality'] >= 7, 'new_quality'] = 2
print(wine['new_quality'].describe())
print(wine['new_quality'].value_counts())
```

```
# 5.15 데이터 정규화 및 train, test 데이터 분리
# 정규화, 원핫 인코딩
del wine['quality']
wine_backup = wine.copy()
wine_norm = (wine - wine.min()) / (wine.max() - wine.min()) wine_norm['new_quality'] =
wine_backup['new_quality']
wine_shuffle = wine_norm.sample(frac=1)
wine_np = wine_shuffle.to_numpy()

train_idx = int(len(wine_np) * 0.8)
train_X, train_Y = wine_np[:train_idx, :-1], wine_np[:train_idx, -1]
test_X, test_Y = wine_np[train_idx:, :-1], wine_np[train_idx:, -1]
train_Y = tf.keras.utils.to_categorical(train_Y, num_classes=3)
test_Y = tf.keras.utils.to_categorical(test_Y, num_classes=3)

# 5.16 와인 데이터셋 다항 분류 모델 생성 및 학습
model = tf.keras.Sequential([
        tf.keras.layers.Dense(units=48, activation='relu', input_shape=(12,)),
        tf.keras.layers.Dense(units=24, activation='relu'), tf.keras.layers.Dense(units=12,
        activation='relu'),
        tf.keras.layers.Dense(units=3, activation='softmax') ])

model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.003),
loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(train_X, train_Y, epochs=25, batch_size=32, validation_split=0.25)

# 5.17 다항 분류 모델 학습 결과 시각화
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('Epoch')
```

```
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], 'g-', label='accuracy')
plt.plot(history.history['val_accuracy'], 'k--', label='val_accuracy') plt.xlabel('Epoch')
plt.ylim(0.5, 0.7)
plt.legend()
plt.show()

# 5.18 다항 분류 모델 평가
model.evaluate(test_X, test_Y)
```

<다항 분류 : 패션 MNIST>

티셔츠, 부츠 등 패션의 10 개 분류 (손글씨와 구조는 비슷)

• 60000, 10000 개, 28 X 28 이미지 구조, 10 개의 분류

```
# 5.21 데이터 정규화
train_X = train_X / 255.0 test_X = test_X / 255.0 print(train_X[0])

# 5.22 Fashion MNIST 분류 모델
model = tf.keras.Sequential([
        tf.keras.layers.Flatten(input_shape=(28,28)),
        tf.keras.layers.Dense(units=128, activation='relu'), tf.keras.layers.Dense(units=10,
        activation='softmax') ])
model.compile(optimizer=tf.keras.optimizers.Adam(),
        loss='sparse_categorical_crossentropy', metrics=['accuracy']) model.summary()

# 5.23 Fashion MNIST 분류 모델 학습
history = model.fit(train_X, train_Y, epochs=25, validation_split=0.25)

# 5.25 Fashion MNIST 분류 모델 평가
model.ealuate(test_X, test_Y)
```

```
#Fashion-MNIST 데이터 저장
#미리 섞여진 fashoin-mnist 의 학습 데이터와 테스트 데이터 로드
# 필요 모듈 임포트 # tensorflow 와 tf.keras 를 임포트
import tensorflow as tf
from tensorflow import keras
# ① 문제와 정답 데이터 지정
fashion_mnist = keras.datasets.fashion_mnist (train_images, train_labels), (test_images,
test_labels) = fashion_mnist.load_data()
# 10 개의 분류 이름 지정
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker',
'Bag', 'Ankle boot']
 # 데이터 전처리
# 샘플 값을 정수(0~255)에서 부동소수(0~1)로 변환
train_images, test_images = train_images / 255.0, test_images / 255.0

# ② 모델 구성(생성)
model = keras.Sequential([
        keras.layers.Flatten(input_shape=(28, 28)),
        keras.layers.Dense(512, activation='relu'),
        keras.layers.Dense(256, activation='relu'),
        keras.layers.Dense(128, activation='relu'),
        keras.layers.Dense(10, activation='softmax') ])

# ③ 학습에 필요한 최적화 방법과 손실 함수 등 지정
# 훈련에 사용할 옵티마이저(optimizer)와 손실 함수, 출력정보를 선택
model.compile(optimizer='adam',
        loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.summary()

# ④ 생성된 모델로 훈련 데이터 학습
# 모델을 훈련 데이터로 총 5 번 훈련
model.fit(train_images, train_labels, epochs=8)
```

```
# ⑤ 테스트 데이터로 성능 평가
  test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print('₩n 테스트 정확도:', test_acc)
```