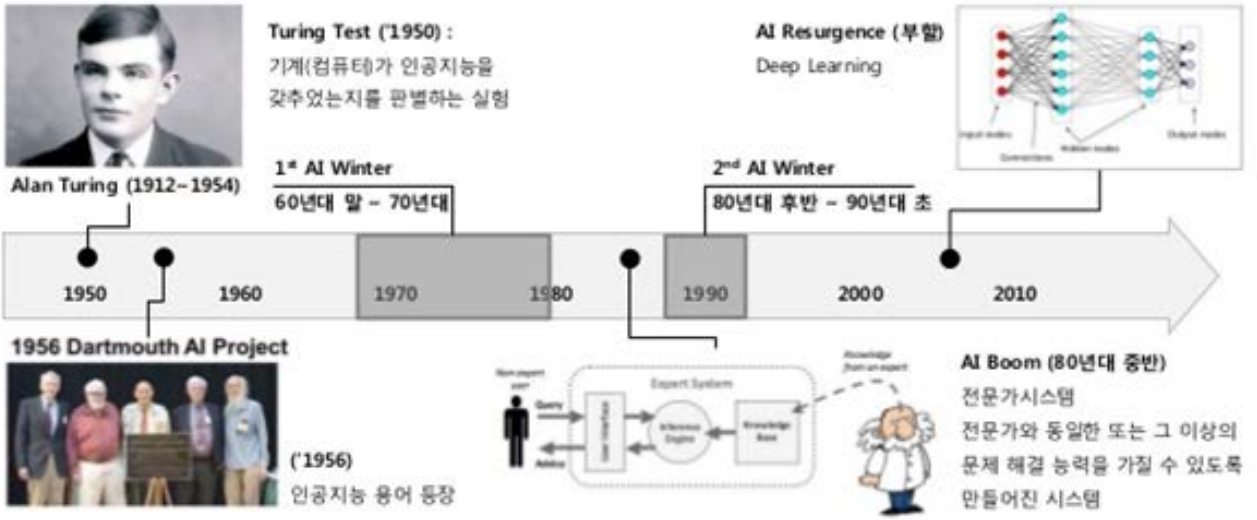
인공지능

13강

인공지능의 역사

- 1956년 존 매카시 교수 (다트머스 학술대회에서 처음 주창)
- 1980년 전문가 시스템 : 전문분야의 지식을 모음, 지식이 방대하고 복잡함
- 1990년 인터넷, 컴퓨팅 기술 등의 발달로 다시 부흥 → 정체기 반복
- 1997년 IBM 딥 블루 vs. 가리 카스파로트 (체스) → 승리
- 2006년 토론토대학 제프리 힌튼 교수 → 다계층 인공신경망의 효율적 학습 가능성 제시
- 2011년 IBM 왓슨 → 제프디 우승 → 의료, 금융분야로 확대
- 2016년 Google 알파고 vs. 이세돌 (바둑) → 4:1 승리
- 의료, 서비스, 자동차 산업분야 적용 → 인공지능이 4차 산업혁명의 핵심

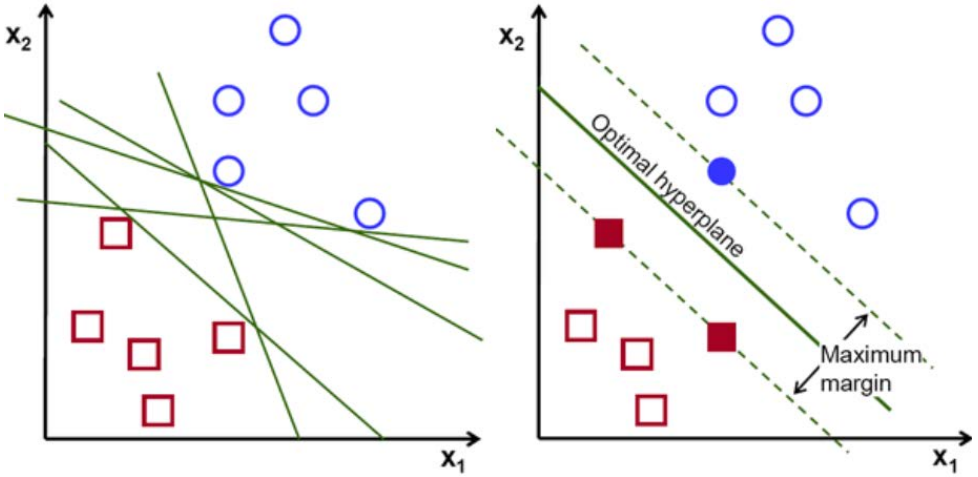


인공신경망 / 머신러닝 / 딥러닝

- 목표: 인공지능
- 방법:
  - 기존 : 정해진 알고리즘에 따라 순차적 동작
  - 사람의 지적 능력을 흉내 내기 위해 **학습 능력**이 필요
- Artificial Neural Network
  - 신경세포를 모방
- Machine Learning : 학습 능력을 갖춘 인공지능 기술
  - 신경망 초기 과적합 등의 단점을 보임
  - 모델링의 정확도 판단이 어려움
  - 복잡한 데이터에서는 적합한 모델을 찾는 것이 쉽지 않음
  - 데이터 모델링과 특징점 추출 등 수작업의 어려움
  - 수학적 방법 : 서포트 벡터 머신 등 활용 (간단하면서, 모델 수립 등이 용이함)
- Deep Learning
  - 빅데이터를 활용하는 경우 더 많은 계층을 포함
  - 모델링에 필요한 시간과 비용을 절약할 수 있어 실제 문제에 더 적합함

Support Vector Machine

- 두 개의 그룹을 분류하는 가장 일반화된 경계선을 찾기 위한 수학 이론 기반 알고리즘
- 목적 변수에 따른 훈련 데이터가 주어졌을 때 데이터 사이의 결정 경계선을 찾는 알고리즘
- 서포트 벡터(Support vector) : 경계선 정의를 위한 데이터
- 초평면(Hyperplane) : 경계를 나누는 기하학적 구조



$$y = mx + b$$

$$\vec{w} \cdot \vec{x} + b = 0$$

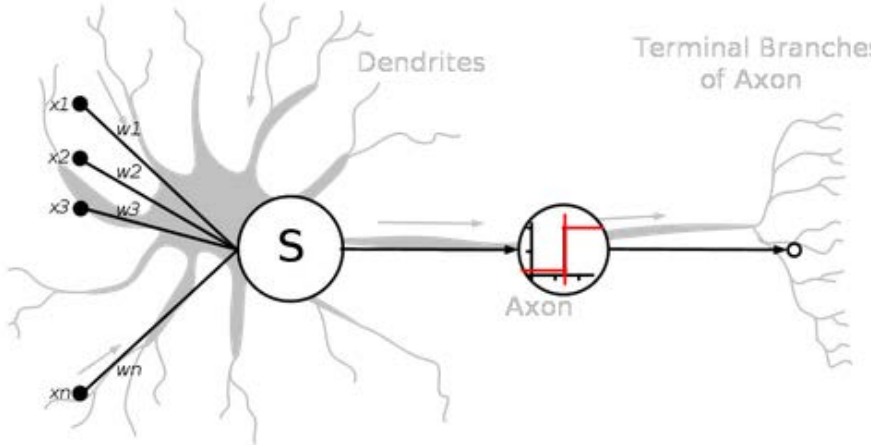
$$w_1 x_1 + w_2 x_2 + w_3 x_3 + b = 0$$

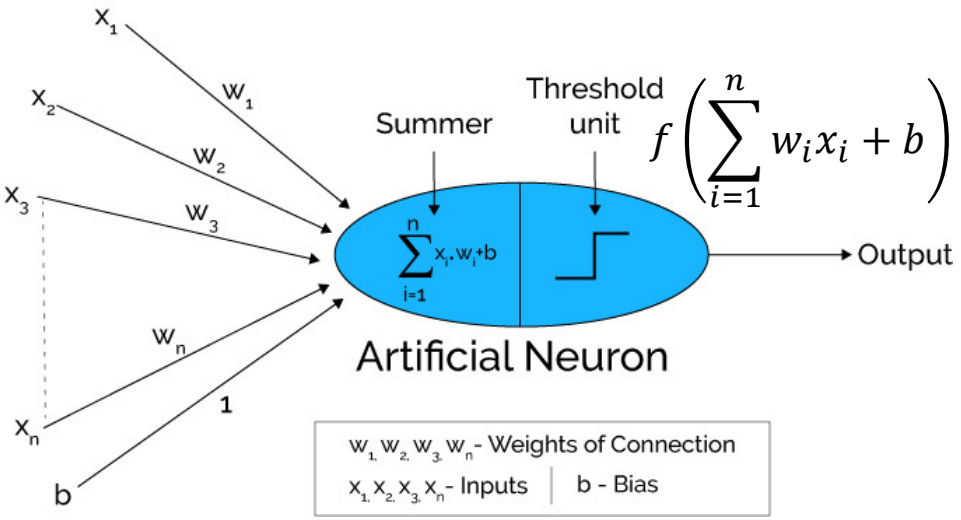
$$\sum_{i=1}^{3} w_i x_i + b = 0$$

$\vec{w}$: 가중치 벡터, b: 편향값

인공신경망 (Artificial Neural Network)

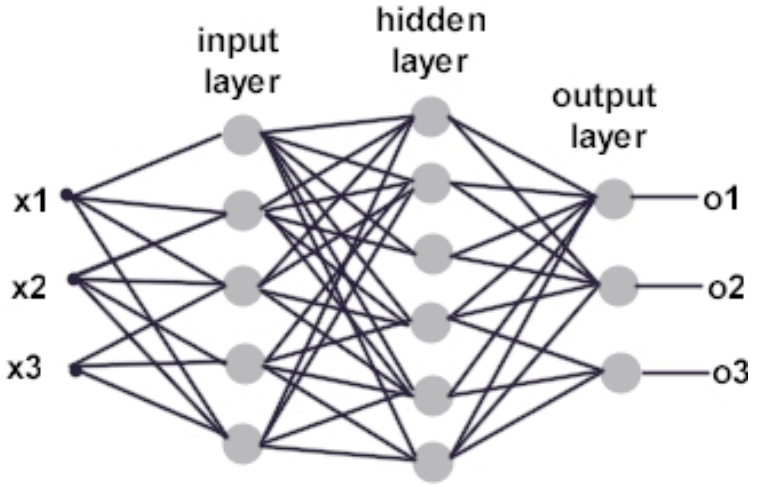
- 인공지능은 기계가 인간의 두뇌를 흉내 내도록 만든 장치와 알고리즘
- 인간의 생물학적 두뇌 구조에 대한 이해를 바탕으로 개발되어 생체 신경망을 흉내 내는 알고리즘
- 단순한 독립적 신경 세포들이 병렬적으로 작동하여 통합적인 뇌의 인지시스템 구성
- 연결 강도나 패턴은 외부 자극, 훈련에 따라 끊임없이 변화
- 딥러닝을 가능하게 한 핵심 원리
- 인공지능의 대부분 주제에 적용가능
- 잠재력과 적응력이 뛰어난 알고리즘

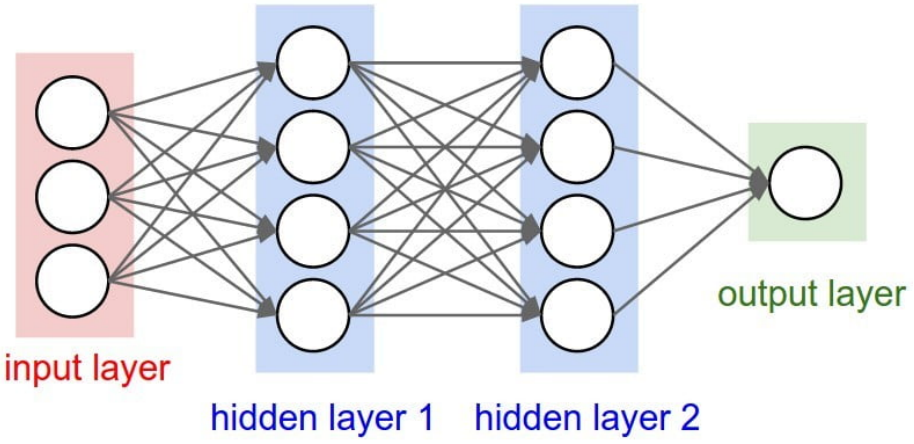




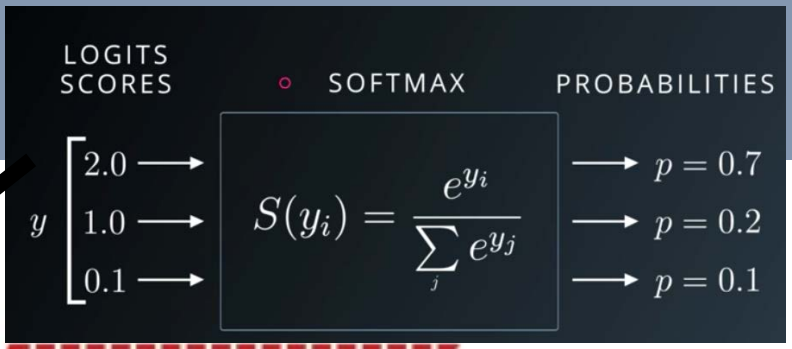
인공신경망 구조

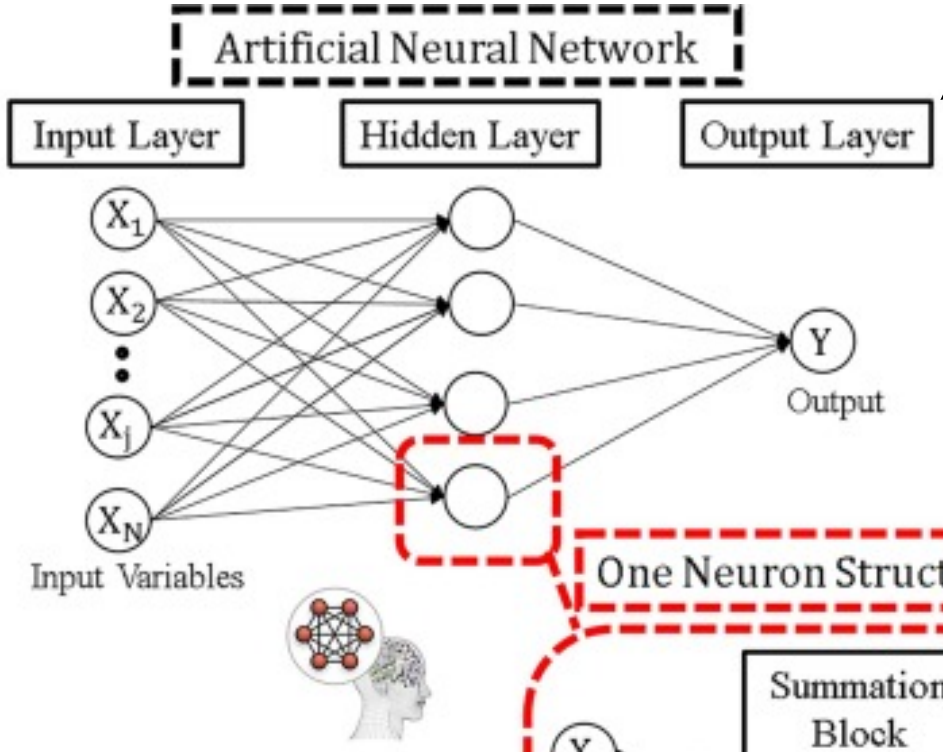
- 신경세포(뉴런) : 입력, 가중치, 고유 편향값(bias), 활성화 함수
- 신경층(layer) : 뉴런의 병렬적 배치, 수직적으로 여러 층으로 구성
- 노드(node) : 신경층 내 개별 뉴런
- 입력층(input layer): 노드의 개수 = 학습할 데이터 속성 수
- 출력층(output layer): 노드의 개수 = 목적 변수 값의 수
- 은닉층(hidden layer) : 복잡한 패턴 파악, 특별 연산 함수 수행

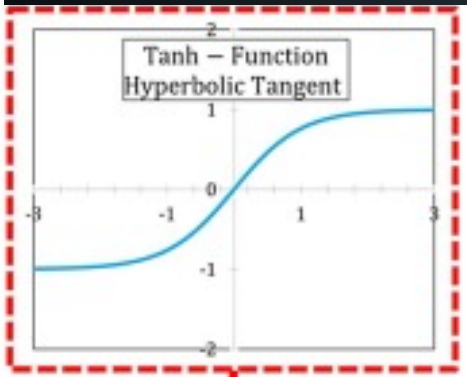




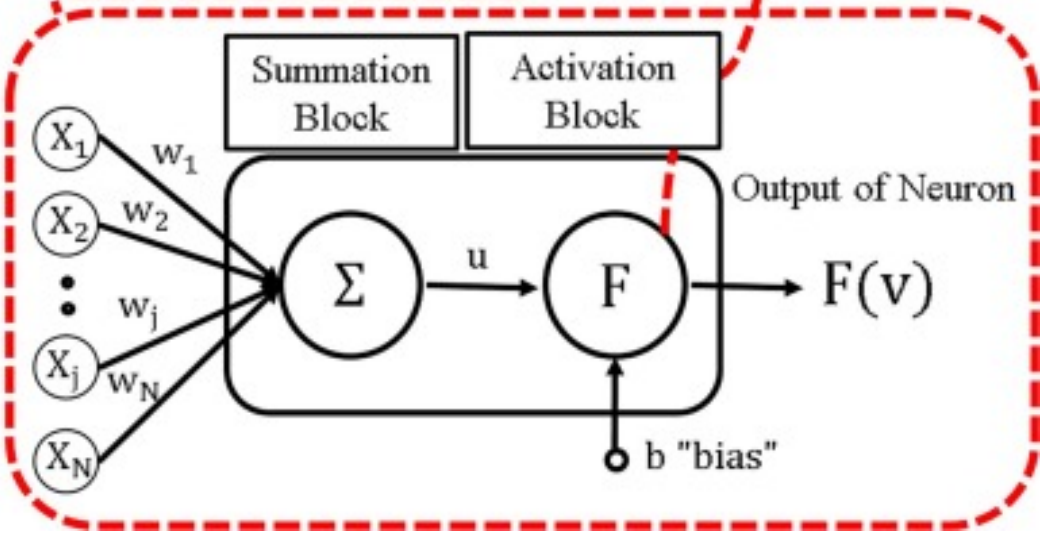
ANN Structure







One Neuron Structure



u: Output of summation block

b: bias

$v = u + b$

v: Input for activation block

F(v): Output of neuron

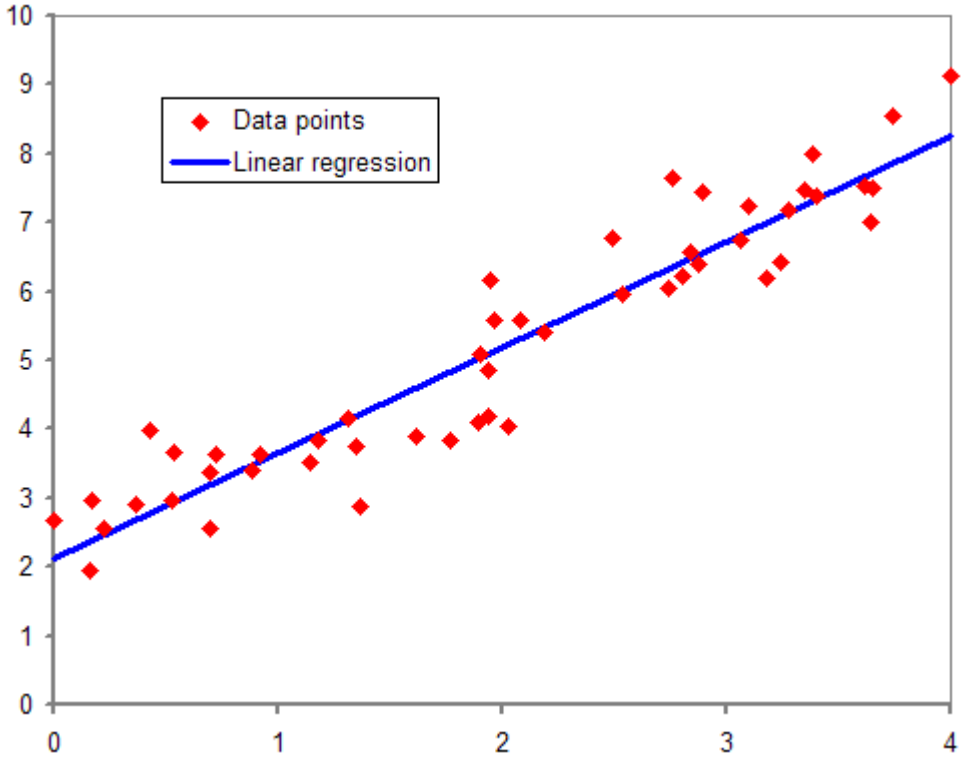
Fathalla 2018

분류 ANN

- 입력 정보를 클래스 별로 분류하는 방식
  - 예시: 필기체 숫자를 0부터 9로 분류
  - 입력: 필기체 숫자 그림
  - 출력: 분류한 결과 출력
- 판별: 두 출력 노드의 값을 비교하여 더 큰 쪽을 선택하도록 구현 (전방향 예측)
- 가중치의 학습은 예측값의 목표값에 대한 오차를 역방향으로 되돌리면서 수행: 오차역전파 (error back propagation)
  - 오차역전파는 오차를 줄이는 경사 하강법(gradient descent)에서 유도된 방법
  - 가중치에 대한 손실 함수를 미분하고 미분값의 방향과 크기를 활용해 가중치를 보상하는 방법
- 손실 함수 (loss function): 가중치에 따라 오차가 얼마나 크고 작아지는지 평가
  - 교차 엔트로피(cross-entropy)함수 사용
  - 출력 노드의 결과를 확률 값으로 변경
  - 확률 값은 출력 노드 값을 소프트맥스 연산으로 구함

회귀 ANN

- 입력 값으로부터 출력 값을 직접 예측
- 데이터의 경향성으로 연속적인 수치 예측
  - 예시 1: 최근 일주일간 평균 온도로 내일 평균 온도를 예측
  - 예시 2: 현재까지의 주가 흐름으로 내일의 주가 예측
- 평균제곱오차(Mean-square error)를 이용한 오차역전파 방법으로 학습

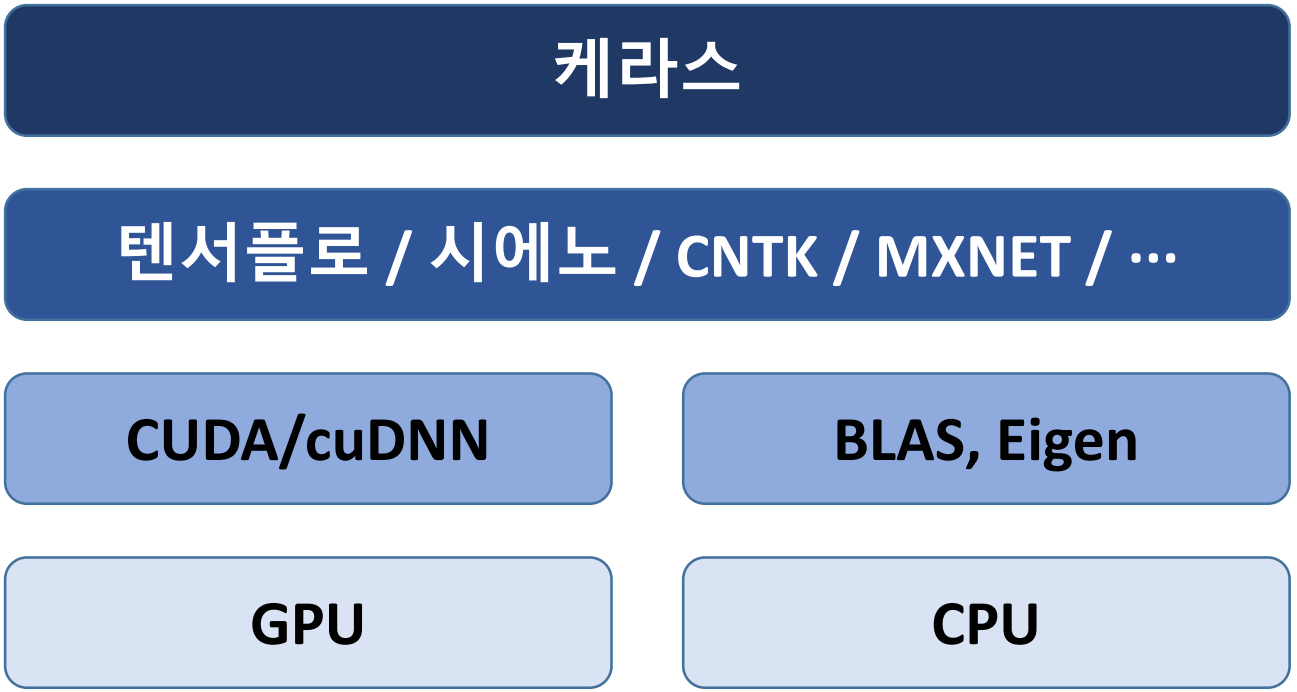


인공신경망 구현 방법 및 단계

- 함수형 구현 (전문가)
- 객체지향형 구현 (사용자)
- ANN 모델링:
  - 분산 방식 – 구조가 복잡한 경우
  - 연쇄 방식 – 하나의 순서로 구성된 간단한 신경망 구현에 적합
  - 혼합 방식
- 프로그램 단계
  - 1단계 : 인공지능 구현용 패키지 불러오기
  - 2단계 : 인공지능에 필요한 파라미터 설정
  - 3단계 : 인공지능 모델 구현
  - 4단계 : 학습과 성능 평가용 데이터 불러오기
  - 5단계 : 인공지능 학습 및 성능 평가
  - 6단계 : 인공지능 학습 결과 분석

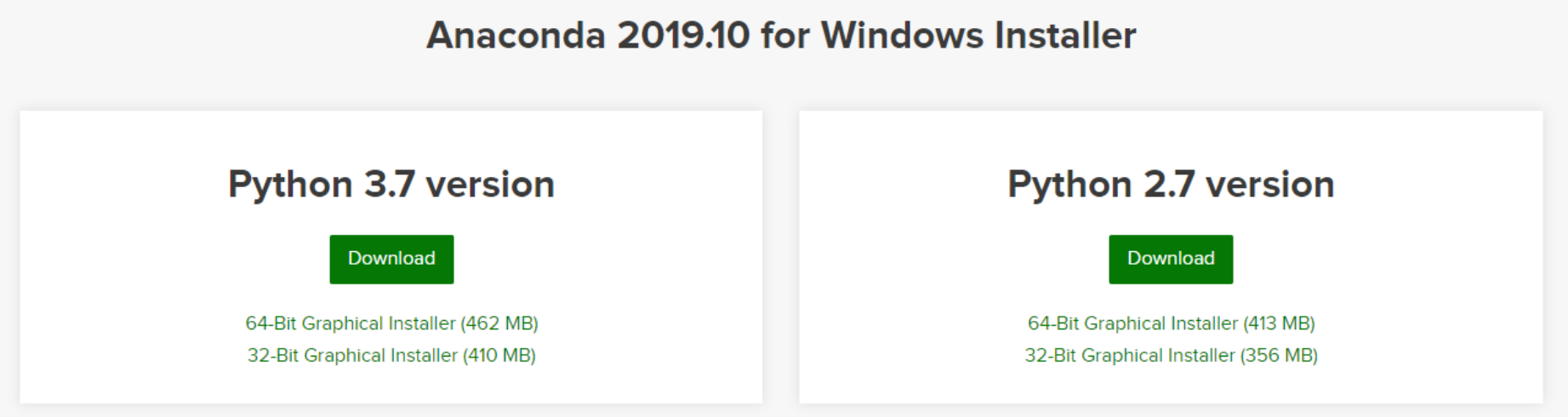
케라스

- 케라스 = 인공지능 코딩을 쉽게 할 수 있는 파이썬 라이브러리
- 인공지능을 쉽게 구현할 수 있는 다양한 상위 레벨 인터페이스 제공
- 하위 레벨의 계산을 직접 수행하는 라이브러리는 아님



아나콘다 파이썬 설치

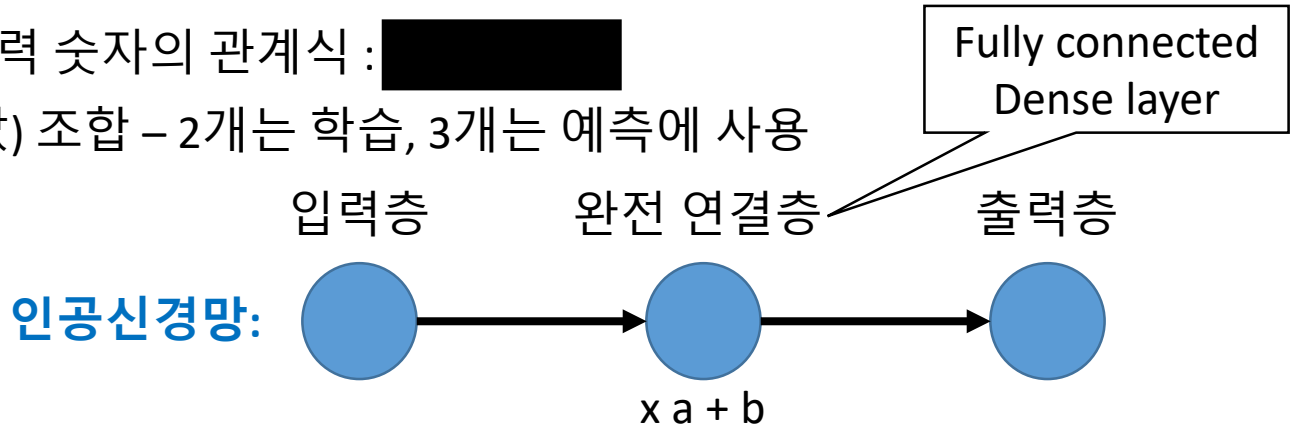
- www.anaconda.com
  https://www.anaconda.com/distribution/#download-section



- (base) C:\Users\User>conda list
- 아나콘다와 함께 설치된 패키지 목록
- (base) C:\Users\User>conda install keras
- (base) C:\Users\User>python
- 윈도우에 텐서플로우를 엔진으로 하는 케라스 설치 완료
  >>> import keras
  Using TensorFlow backend.

케라스 사용 예제 (텍스트 모드)

- 학습할 내용은 입력 숫자와 출력 숫자의 관계식 : [redacted]
- 데이터 : 5개의 (입력값, 출력값) 조합 – 2개는 학습, 3개는 예측에 사용
- (0, 1), (1, 3), (2, ?), (3, ?), (4, ?)
- 텍스트 모드 실습

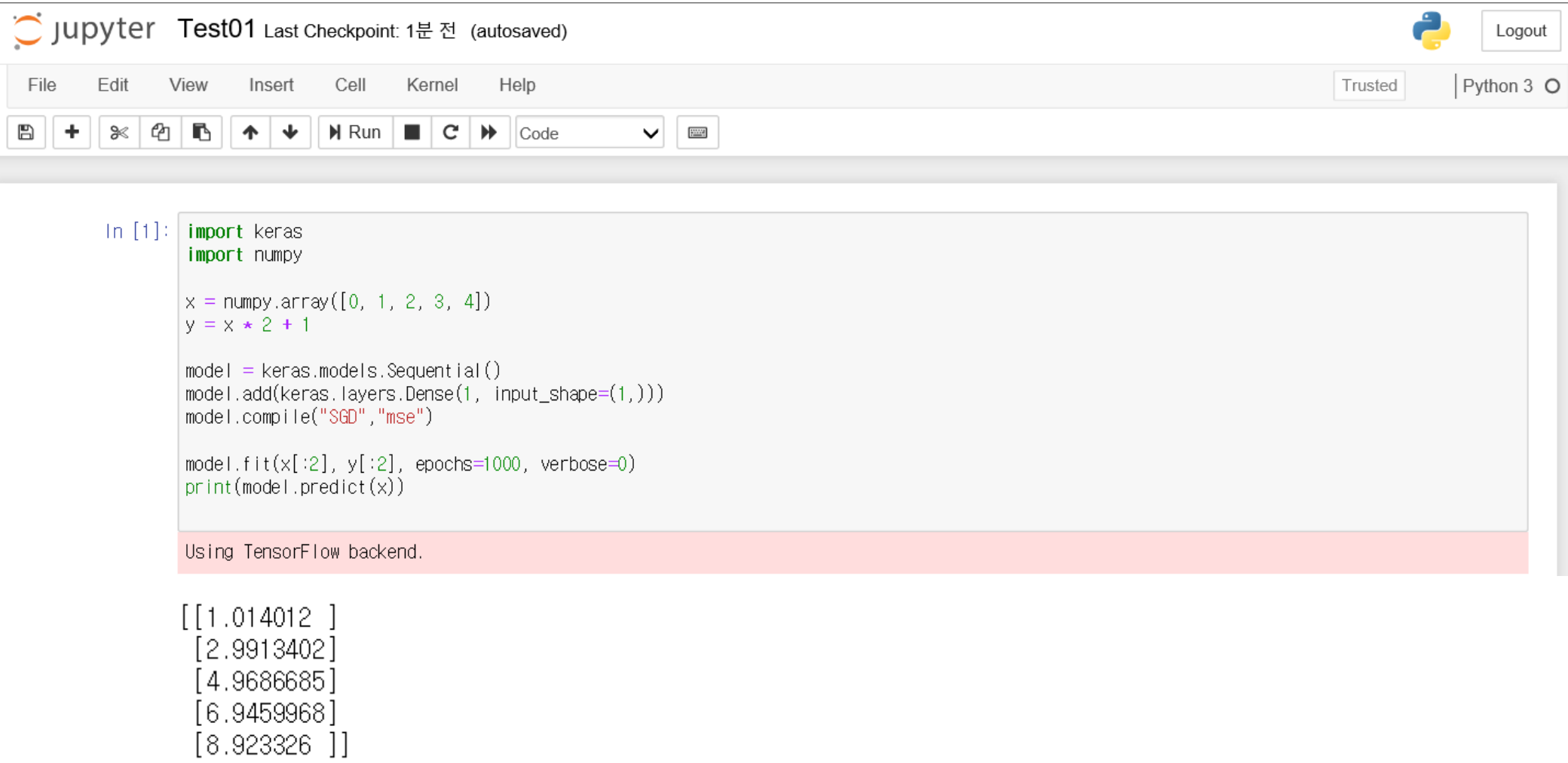



```
>>> import keras
Using TensorFlow backend.
>>> import numpy                                          배열처리 : 넘파이 패키지 사용
>>> x = numpy.array([0, 1, 2, 3, 4])
>>> y = x * 2 + 1                                         입력, 출력 값 준비
>>> model = keras.models.Sequential()                     각 계층을 연결하여 하나의 모델 생성, 컴파일, 학습, 예측 담당
>>> model.add(keras.layers.Dense(1, input_shape=(1,)))    각 계층을 만드는 클래스 제공
>>> model.compile('SGD','mse')                            모델 학습을 파라미터로 지정하고 컴파일
>>> model.fit(x[:2], y[:2], epochs=1000, verbose=0)       2개의 데이터 쌍으로 가중치와 편향값 학습
>>> print('Targets:', y[2:])
Targets: [5 7 9]
>>> print('Predictions: ', model.predict(x[2:]).flatten())
Predictions:  [4.967579  6.9441195 8.920659 ]
```

SGD: 확률적 경사 하강법
mse: 평균제곱오차

주피터 노트북 모드 사용

- (base) C:\Users\User\Test>jupyter notebook



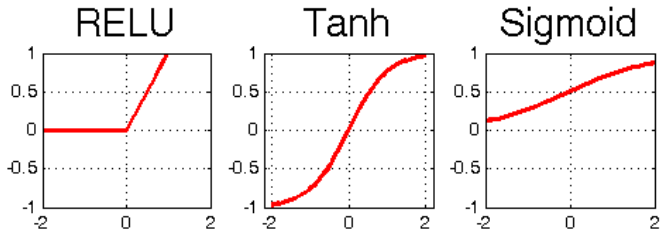
예제1: 필기체 구분 분류 인공신경망

- 분류 ANN은 클래스가 둘 이상인 데이터를 분류하는 인공지능 방법
- 분류 ANN을 위한 인공지능 모델 구현
  - 1단계 : 분류ANN 구현용 패키지 불러오기
  - 2단계 : 분류ANN 에 필요한 파라미터 설정
  - 3단계 : 분류ANN 모델 구현
  - 4단계 : 학습과 성능 평가용 데이터 불러오기
  - 5단계 : 분류ANN 학습 및 검증
  - 6단계 : 분류ANN 학습 결과 분석
- 1단계
  - 케라스 패키지 모듈 불러오기
  - layers 모듈 : 각 계층을 만드는 모듈
  - models 모듈 : 각 layer 들을 연결하여 신경망 모델을 만든 후, 컴파일, 학습, 평가하는 역할
  - Model 객체에는 compile(), fit(), predict(), evaluate() 등 딥러닝 처리 함수 대부분을 제공
  - from keras import layers, models
- 2단계
  - 분류 ANN에 필요한 파라미터 설정
  - Nin(입력 계층의 노드 수), Nh(은닉 계층의 노드 수), number_of_class(출력값이 가질 클래스 수), Nout(출력 노드 수)
- 3단계
  - 모델링: 연쇄 방식(간단한 모델) vs. 분산 방식(복잡도가 높은 모델) 구현
  - 구현방식: 함수형 vs. 객체지향형 방법

분산 방식 모델링을 포함하는 함수형 구현

- 신경망 구조 지정
  - 입력 계층 정의: 원소를 Nin개 가지는 입력신호 벡터는 입력 노드에 따른 계층의 shape을 지정
    - x = layers.Input(shape=(Nin,))
  - 은닉 계층 정의: 노드가 Nh개인 은닉 계층을 구성. x를 입력 받음
  - 활성화 함수: layers.Activation('relu')로 지정, ReLU- f(x)=max(x, 0)
    - h = layers.Activation('relu')(layers.Dense(Nh)(x))

[참고] tanh(), sigmoid()



  - 출력 계층: 출력 노드 수 = 클래스 수로 지정, 출력 노드에 입력되는 정보는 은닉 노드의 출력 값
  - 출력 노드의 활성화 함수로 소프트맥스 연산을 주로 사용
    - y = layers.Activation('softmax')(layers.Dense(Nout)(h))
  - 모델은 입력과 출력을 지정하여 만듦
  - 중간 계층들은 계층 간 신호의 연결 관계 대로 자동 설정됨
  - Model은 딥러닝 구조가 여러 개의 필요한 함수와 연계되도록 만드는 역할을 함
- 컴파일 과정
  - 손실함수, 최적화함수, 학습/예측 시 성능 검증을 위한 파라미터 설정
  - model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

연쇄 방식 모델링을 포함하는 함수형 구현

- 분산 방식과 모델을 지정하는 방식만 다르고, 그 이외 코드는 동일
- 모델 설정
  - 연쇄 방식은 모델 구조를 정의하기 전에 Sequential() 함수로 모델을 초기화 해야 함
  - Model = models.Sequential()
- 모델 구조
  - 입력 계층과 은닉 계층의 형태가 동시 설정
  - 입력 노드 Nin개는 완전 연결 계층 Nh개로 구성된 은닉 계층으로 전송
  - 은닉 계층의 노드들은 ReLU를 활성화 함수로 사용
    - model.add(layers.Dense(Nh, activation='relu', input_shape=(Nin,)))
  - 은닉 계층의 출력은 출력이 Nout개 출력 노드로 전송, 출력 노드의 활성화 함수는 softmax 연산
    - model.add(layers.Dense(Nout, activation='softmax'))
- 연쇄 방식은 추가되는 계층을 기술할 때, 간편하게 기술 가능
- add() 이용 연속되는 계층을 계속 추가 가능
- 복잡한 인공신경망을 기술하는데 한계가 있음

분산 방식 모델링을 포함하는 객체지향형 구현

- 전문가가 만든 인공지능 모델을 객체로 불러 쉽게 활용 가능
- 코드의 재사용성이 높음
- 먼저 클래스를 만들고 models.Model로부터 특성을 상속
- models.Model은 신경망에서 사용하는 학습, 예측, 평가와 같은 다양한 함수 제공
- 초기화 함수 정의
  - 입력 계층(Nin), 은닉 계층(Nh), 출력 계층(Nout)의 노드 수
  - class ANN(models.Model):
    - def __init__(self, Nin, Nh, Nout):
- 신경망 모델에 사용할 계층 정의
  - 은닉 계층이 1개 일 때,
    - hidden = layers.Dense(Nh)
  - 은닉 계층이 여러 개 일 때,
  - Nout개의 출력 계층 정의,
    - output = layers.Dense(Nout)
- Activation 함수 정의 (비선형성 부여)
  - relu = layers.Activation('relu')
  - softmax = layers.Activation('softmax')

```
Nh_l = [5, 10, 5]
hidden_l =map(layers.Dense, Nh_l)
hidden_l =[layers.Dense(n) for n in Nh_l]
hidden_l =[]
for n in Nh_l:
    hidden_l.append(layers.Dense(n))
```

- 상속받은 부모 클래스 초기화 진행
  - super().__init__(x, y)
- 인스턴스 생성
  - model = ANN(Nin, Nh, Nout)

연쇄 방식 모델링을 포함하는 객체지향형 구현

- 신경망 모델이 연속적인 하나의 고리로 연결되어 있다는 가정하에 모델링
- 각 층의 동작 단계가 연속으로 기술되기 때문에 표현이 쉽고 개념적으로 이해가 쉬움
- models.Sequential 에서 상속받음
- 초기화 함수 정의
  - class ANN(models.Sequential):

    def __init__(self, Nin, Nh, Nout):

    super().__init__()
- 모델링: 앞의 계층에 새로운 계층을 계속 추가
- 입력 계층을 별도로 정의하지 않고, 은닉 계층부터 추가
  - self.add(layers.Dense(Nh, activation='relu', input_shape=(Nin,)))
- 출력 계층 정의
  - self.add(layers.Dense(Nout, activation='softmax'))

분류 ANN에 사용할 데이터 불러오기 (4단계)

- MNIST : 6만 건의 필기체 숫자를 모은 공개 데이터 http://yann.lecun.com/exdb/mnist/

1) 데이터 처리에 사용할 패키지 임포트

```
import numpy as np
from keras import datasets
from keras.utils import np_utils
```

reshape()
mnist()
to_categorical()
함수 사용

2) 데이터 불러오기
   - 활용도에 따라 학습(training), 검증(validation), 평가(test) 데이터

```
(X_train, y_train), (X_test, y_test) = datasets.mnist.load_data()
```

3) 출력값 변수를 이진 벡터 형태로 바꾸기

```
Y_train = np_utils.to_categorical(y_train)
Y_test = np_utils.to_categorical(y_test)
```

   - 0~9 숫자로 구성된 출력값을 0과 1로 표현되는 벡터 10개로 변경
   - ANN 분류 작업 시 정수보다 이진 벡터로 출력 변수를 구성하는 것이 더 효율적이기 때문

4) 이미지를 나타내는 아규먼트를 1차원 벡터 형태로 바꾸기
   - X_train, X_test에 (x, y) 축에 대한 픽셀 정보가 들어 있는 3차원 데이터인 실제 학습 및 평가용 이미지를 조정
   - 학습 데이터셋의 샘플은 L개, L*W*H로 저장되어 있음

```
L, W, H = X_train.shape
X_train = X_train.reshape(-1, W*H)
X_test = X_test.reshape(-1, W*H)
```

5) ANN을 위해 입력값들을 정규화(regularization) 하기
   - ANN의 최적화를 위해 아규먼트를 정규화
   - 0~255 사이 정수로 된 입력값을 0~1 사이 실수로 변경

```
X_train = X_train / 255.0
X_test = X_test / 255.0
```

분류 ANN 학습 결과 그래프 구현

- ANN 학습 결과를 분석, 분석 결과를 기반으로 학습 결과 평가 및 하이퍼 파라미터 조절
- ANN 학습 결과 분석은 학습 중 손실과 정확도의 추이를 관찰
- fit()함수의 결과인 history 변수에 저장됨
- 그래프 그리는 라이브러리 plt

```
import matplotlib.pyplot as plt
```

plt.polt(): 선 그리기
Plt.title(): 그래프 제목
plt.xlabel(): x 축 이름 표시
plt.ylabel(): y 축 이름 표시
plt.legend(): 라인의 표식

- 손실을 그리는 함수

```
def plot_loss(history):
    plt.plot(history.history['loss'])
    plt.plot(history.history[‘val_loss’])
    plt.title(‘Model Loss’)
    plt.ylabel(‘Loss’)
    plt.xlabel(‘Epoch’)
    plt.legend([‘Train’, ‘Test’], loc=0)
```

- 정확도를 그리는 함수

```
def plot_acc(history):
    plt.plot(history.history[‘acc'])
    plt.plot(history.history[‘val_acc’])
    plt.title(‘Model accuracy’)
    plt.ylabel(‘Accuracy’)
    plt.xlabel(‘Epoch’)
    plt.legend([‘Train’, ‘Test’], loc=0)
```

분류 ANN 학습 및 성능 분석 (5단계)

- ANN 에서 사용할 파라미터 정의

```
def main():
    Nin = 784
    Nh = 100
    number_of_class = 10
    Nout = number_of_class
```

- 모델의 인스턴스를 만들고 데이터를 불러옴

```
model = ANN_seq_class(Nin, Nh, Nout)
(X_train, Y_train), (X_test, Y_test) = Data_func()
```

- 모델을 학습하는 방법

```
history = model.fit(X_train, Y_train, epochs=5, batch_size=100, validation_split=0.2)
```

입력 출력 반복학습 20% 데이터를 검증에 사용

한 번에 계산할 데이터 길이 지정

- 성능 최종 평가

```
performace_test = model.evaluate(X_test, Y_test, batch_size=100)
print('Test Loss and Accuracy ->', performace_test)
```

- 손실과 정확도 추이 그리기

```
plot_loss(history)
plt.show()
plot_acc(history)
plt.show()
```

예제2: 시계열 데이터를 예측하는 회귀 ANN

- 보스턴 집값을 예측하는 회귀 ANN 구현

- 1단계 : 회귀ANN 구현용 패키지 불러오기
- 2단계 : 회귀 ANN 구현
- 3단계 : 학습과 평가용 데이터 불러오기
- 4단계 : 회귀ANN 학습 및 성능 평가
- 5단계 : 회귀ANN 학습 결과 분석

- 1단계 : 케라스 패키지 불러오기 from keras import datasets
  - layers : 각 계층을 구성하는 툴
  - models : 계층을 하나로 합쳐 하나의 모델을 만드는 툴
- 2단계 : 회귀 ANN 구현
  - 클래스 생성, 신경망 계층 정의
    - 은닉 계층에 사용할 노드 수 = Nh개 완전 연결 계층
    - 출력 계층은 Nout개 완전 연결 계층 생성
    - 활성화 함수는 relu
  - ANN 각 계층의 신호 연결 상황 정의
    - x : Nin 길이를 가지는 1차원 열 벡터
    - 입력신호가 hidden 통과 후 relu() 단계 통과
    - h는 출력 계층 후 바로 y
  - 출력 노드에 활성화 함수를 사용하지 않음
  - 입력과 출력을 이용한 모델 사용하도록 컴파일

    mse: 평균제곱오차, sgd: 확률적 경사 하강법

```
class ANN(models.Model):
  def __init__(self, Nin, Nh, Nout):
    hidden = layers.Dense(Nh)
    output = layers.Dense(Nout)
    relu = layers.Activation('relu')

    x = layers.Input(shape=(Nin,))
    h = relu(hidden(x))
    y = output(h)

    super().__init__(x, y)
    self.compile(loss='mse', optimizer='sgd')
```

회귀 ANN에 사용할 데이터 불러오기 (3단계)

- Boston housing : 506 건의 보스턴 집값과 관련된 13가지 정보 포함

1) 데이터 처리에 사용할 패키지 임포트

```
from keras import datasets
from sklearn import preprocessing
```

Boston housing 데이터
머신러닝 패키지

2) Boston Housing 데이터 가져오기와 정규화

```
(X_train, y_train), (X_test, y_test) = datasets.boston_housing.load_data()
scaler = preprocessing.MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

학습과 변환
변환

3) 회귀 ANN 학습 결과 그래프 구현

회귀 ANN 학습 및 성능 분석 (4단계)

- ANN 모델링과 관련된 파라미터 정의
  - Nin : 입력 벡터의 길이
  - Nh : 은닉 계층 수 = 5
  - Nout : 회귀를 통해 결과값 예측이기 때문 = 1

```
def main():
    Nin = 13
    Nh = 5
    Nout = 1
```

- 구현한 회귀 ANN 모델의 인스턴스를 생성
- 적용할 데이터 불러옴
- 불러온 데이터를 이용하여 생성된 모델의 인스턴스 학습

```
model = ANN(Nin, Nh, Nout)
(X_train, y_train), (X_test, y_test) = Data_func()
history = model.fit(X_train, y_train, epochs=100, batch_size=100, validation_split=0.2, verbose=2)
```

- 성능 평가
  - 인공 지능 학습과정에서 전혀 사용하지 않은 데이터만 사용
  - model.evaluate()
- 학습 진행 사항을 그래프로 그림
  - 하이퍼 파라미터 조정

```
plot_loss(history)
plt.show()
```

EOD