

시퀀셜 모델 생성 : `model = keras.Sequential()`

레이어 추가 : `model.add(~Dense(units=n(or 숫자만 입력 ), activation='tanh'))`  
(중간에 `input_shape=(-2,)` 넣어도 됨)

로스 옵티마이저 메트릭 입력(컴파일) : `model.compile()`

학습 : `model.fit(X, y, batch_size, epochs=100)`

결과값 확인 : `model.predict(구할 데이터)`

파라미터 개수 구하기

입력 개수가 2이고 히든 2개 유닛 출력 1개 유닛 인 경우

1.  $2*2+2$ (유닛당 바이어스값 1개씩) **6개** (3개면  $2*3+3$ ), 2.  $2*1+1$  총 **3개**

`evaluate(x=None, y=None)`: 테스트 모드에서 모델의 손실 함수 값과 측정 항목 값을 반환

`add(layer)`: 레이어를 모델에 추가한다.

`model = Sequential()`

`model.add(Dense(units=2, input_shape=(2,), activation='sigmoid'))` -> 히든레이어

`model.add(Dense(units=1, activation='sigmoid'))` -> 출력층

단 `def conv2d_block(x, channel):`

`x = Conv2D(channel, 3, padding="same")(x)`

`x = BatchNormalization()(x)`

`x = Activation("relu")(x)`

`x = Conv2D(channel, 3, padding="same")(x)`

`x = BatchNormalization()(x)`

`x = Activation("relu")(x)`

`return x` 이렇게 쓰기도 함

`model.add(tf.keras.layers.Dense(units=2, input_shape=(2,), activation='sigmoid'))`

`= model.add(tf.keras.layers.Input(shape=(2,)))`

`model.add(tf.keras.layers.Dense(2, activation='sigmoid'))`

모델: 하나의 신경망을 나타낸다

입력 데이터: 텐서플로우 텐서 형식

옵티마이저: 학습을 수행하는 최적화 알고리즘이다. 학습률과 모멘텀을 동적으로 변경

`Input(shape, batch_size, name)`: 입력을 받아서 케라스 텐서를 생성하는 객체

`Dense(units, activation=None, use_bias=True, input_shape)`: 유닛들이 전부 연결된 레이어

모델 저장 : `model.save('./~')`      모델 로드 : (로드모델 импорт 필요) `model = load_model('./~')`

손실함수(`loss = ''`)

회귀 손실함수 : `mse(MeanSquaredError)`

분류 손실함수 : `binary_crossentropy` - 강아지인지 강아지가 아닌지만

`categorical_crossentropy` - 정답 원하던 경우 강아지 고양이 호랑이 분류

`sparse_categorical_crossentropy` - 정답 정수인 경우 강아지 고양이 호랑이 분류

경사하강법(`optimizer = ''`)

GD(경사하강법 - 전체 데이터), SGD(확률적 경사하강법 - 랜덤 일부 데이터 추출)

NAG, Adagrad, RMSprop, Adam(가장 많이 사용)

정확도(`Metric = []`)

예측값이 정답 레이블과 얼마나 같은지 횟수 계산

(mse같은 손실함수를 써도 되나 주로 `accuracy` 사용)

batch\_size -> 1epoch동안 전체샘플수/ batch\_size의 크기로 가중치 업데이트(램, 속도, 안정성 고려)  
배치 사이즈의 크기마다 한번씩 가중치를 업데이트함 (너무 커지면 느려지고 작으면 불안정)

하이퍼 매개변수란? 학습률 은닉층을 몇 개로 할 것이며, 은닉층의 개수나 유닛, 배치의 크기를 말함  
정수형 -> 원핫인코딩 하는 법 : 변수 = tf.keras.utils.to\_categorical(바꿀 변수)

문자열 -> 원핫인코딩 하는 법 :

원핫 = boolType 인코딩

```
import numpy as np
X = np.array(['Korea', 'Japan', 'China'])

from sklearn.preprocessing import OneHotEncoder
onehotencoder = OneHotEncoder()
XX = onehotencoder.fit_transform(X.reshape(-1,1)).toarray()
print(XX)
```

```
import numpy as np #booltype
import pandas as pd #판다스 사용
X = np.array(['Korea', 'Japan', 'China'])
XX_df = pd.get_dummies(X)
XX = XX_df.values
print(XX)
```

문자열 -> 정수형 데이터 처리 하는 법 :

```
for ix in train.index: \n if train.loc[ix, 'Sex']=="male": \n train.loc[ix, 'Sex']=1 \n else: \n train.loc[ix, 'Sex']=0
```

쉬운방식 import pandas as pd \n train['Sex'] = train['Sex'].replace({"male": 1, "female": 0})

원핫 인코딩에서 결과값 총 합을 1로 바꾸는 법 : output의 activation을 softmax  
(따라서 회귀 이용하는 경우 output의 결과에 활성화함수(엑티베이션)를 입력하지 않음)

엑셀에서 데이터 읽어오기

```
train = pd.read_csv("train.csv", sep=',')
```

->sep는 분할 방법 ' '등을 분할, (pd....).strip 써서 공백제거, ().lower로 소문자화도 가능

필요없는 컬럼 삭제

```
train.drop([컬럼명, 컬럼명], inplace = True, axis=1)
```

결손치 존재 데이터 행 삭제

```
train.dropna(inplace=True)
```

과적합(overfitting) : 학습이 훈련 데이터에 특화되어 데이터의 전체적인 패턴을 학습하지 못하는 것

과소적합(underfitting) : 너무 단순하게 학습되어 패턴을 제대로 파악하지 못하는 것

과적합 해결법 :

1. 검증 데이터 이용(6:2:2)하여 검증 손실이 감소하지 않을 때 EarlyStopping이용

(callback = EarlyStopping(monitor='val\_loss', patience=20)) #중단 시점

path = "./~" #저장 경로 지정

checkpoint = ModelCheckpoint(filepath = path, monitor='val\_loss', verbose=1, save\_best\_only = True #최적화 모델 저장 설정

hist=model.fit(X, y, epochs=n, batch\_size=n, validation\_split=0.25, verbose=1, callbacks=[callback, checkpoint]))

2. 레이어 사이에 Dropout 추가

model.add(tf.keras.layers.Dropout(0.5)) (Dense쓰면 Dense(16, activation='', -))

## 정규화 방법

```
from sklearn.preprocessing import MinMaxScaler
mms=MinMaxScaler()
mms.fit(x)
print(x.head())
x = mms.transform(x)
x = pd.DataFrame(x, columns=x.columns, index = list(x.index.values))
```

컬러이미지shape (height, width, rgb ==3)      흑백이미지shape (height, width)

## 컨볼루션 연산 하는 법

입력 [[2,1,0] [1,3,2] [4,3,2]] 인 경우  $2*1+ 1*2+ 0*0+ 1*1+ 3*2+ 2*0+ 4*3+ 3*2+ 2*4$

커널 [[1,2,0] [1,2,0] [3,2,4]] = 37

[[ -1.0, 1] [-2.0, 2] [-1.0, 1]] 커널을 사용하는 경우 상하선 표시가 잘됨

완전 연결 신경망(영상(이미지)에서는 사용 불가) : Fully Connected layer, Densely connected layer

CNN : Convolutional Neural Network (영상사용 가능 (필요한 부분만 사용))

보폭(stride) : 커널을 적용하는 거리 (1이면 한칸마다 2면 두칸마다)

tf.keras.layers.Conv2D

(filters, kernel\_size, strides=(1,1), activation=None, input\_shape=(11,), padding='valid')(입력data)

filters : 필터 개수 (그냥 숫자로 표기)

kernel\_size : 필터 크기 (3이면 3\*3)

strides : 보폭

activation : 활성화 함수

input\_shape : 입력 배열의 형상(기준 개수) (=1:] 이렇게도 사용 가능)

padding : 패딩 (디폴트는 valid, 선택하여 same)

valid시 필터에 따른 출력 사이즈 = 입력 사이즈 - (커널 사이즈 - 1)

```
model = keras.Sequential(
[
keras.Input(shape=input_shape),
layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
layers.MaxPooling2D(pool_size=(2, 2)),
layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
layers.MaxPooling2D(pool_size=(2, 2)),
layers.Flatten(),
layers.Dropout(0.5),
layers.Dense(num_classes, activation="softmax"),
]
)
```

input\_shape = (28, 28, 1)

3\*3 -> 26, 26, 1

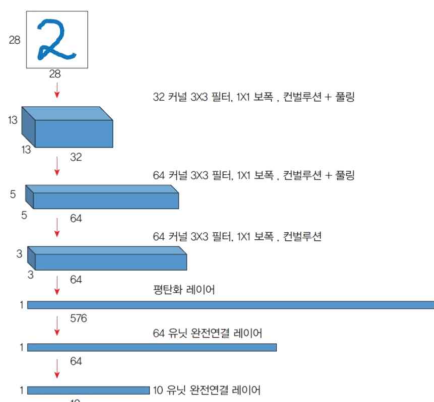
pool -> 13, 13, 1

3\*3 -> 11, 11, 1

pool -> 5, 5, 1

flatten -> 25

Tip : Flatten은 Dense 쓰기 전에 사용



Model: "sequential"		
Layer (type)	Output Shape	Param #
=====Model: "sequential"=====		
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dropout (Dropout)	(None, 1600)	0
dense (Dense)	(None, 10)	16010
=====		
Total params: 34,826		
Trainable params: 34,826		
Non-trainable params: 0		

**데이터 증대\* (data augmentation)** : 방법은 (10)px 이동, 노이즈 추가, 각도 돌리기, 반전, 확대/축소  
**전이 학습\*(transfer learning)** : 이미 학습한 신경망의 모델과 가중치를 조정해 새 문제에 적용하는 것  
(Xception, VGG, Resnet, Mobilenet 등)  
lib로 존재하는걸 임포트해서 씀

super resolution -> 업스케일링으로 사용  
style transfer -> 이미지 화풍 변경 가능

시계열 데이터 : 시간이나 공간등의 순서가 있는 데이터 (주식, 텍스트, 오디오)

**RNN\*(Recurrent Neural Network)** : 시계열 데이터 처리를 위해 이전 내용이 학습되는 CNN  
(지나면서 계속 입력을 하고 결과값 데이터로 학습이 해야함 - 과거를 기억)

**LSTM\*(Long-Shor Term Memory)** : RNN에서 데이터 저장여부를 선택 가능함

형식은 `model.add(SimpleRNN(16, activation='', return_sequences=True, input_shape()))`  
(단 RNN1은 return\_sequences가 빠짐)

**자연어 처리 순서** : 음성인식 -> 자연어 이해 -> 대화 -> 응용 프로그램 조직 -> 로직 -> 처리기 ->  
자연어 생성 -> TTS (요약 : 음성 -> 음성인식 -> 텍스트 -> 자연어 처리)

다음 처리 순서: 텍스트 -> data정제 -> 텍스트 전처리(word embedding) -> 딥러닝 -> 분류 -> 처리

텍스트 전처리 : 1.토큰화 2.특수문자 제거 3.소문자 변환 4.불용어 제거

토큰화 : 주로 단어 단위로 나눔

실제로 자연어 처리는 이렇게 이루어진다고 한다.

특수문자 제거 : 구두점이나 특수문자를 제거

요약 버전인 듯 하다.

소문자 변환 : .lower()등을 이용해 전부 소문자로 변환

음성인식 → 텍스트처리 → 토큰화전처리 → 임베딩 → 모델 → 결과

불용어 제거 : 내용 분석에 도움되지 않는 단어들 제거

자연어 처리시 정수처리(숫자가 너무 커짐), 원핫(너무 길어짐)의 문제로 **워드 임베딩**을 사용  
(예로 blue = [-0.7, 1.4, -0.2, -0.1] 의 밀집 벡터로 표현 )

샘플의 패딩 : 하나만 있으면 앞에 0으로 채움

**LLM\*(Large Language Model)** : 방대한 데이터로 사전 학습된 초대형 딥 러닝 모델

**Chat GPT\*** : Transformer 모델 기반으로 Generative Pre-trained Transformer

Transformer 모델은 self-attention기반임

**강화학습\*** : 딥러닝을 탑재한 에이전트가, 주어진 환경에서 스스로 행동해서 (시행착오로)학습

- **에이전트(Agent)** : 강화 학습을 하는 객체
- **환경(Environment)** : 에이전트가 작동하는 물리적 세계
- **상태(State)** : 에이전트의 현재 상황 (예시로 미로에서 에이전트의 위치)
- **보상(Reward)** : 환경으로부터의 피드백 (성공 실패에 대한 보상)
- **액션(Action)** : 에이전트의 행동