# 1-1. 모델의 정의_분류

| | |
|---|---|
| Category | modeling |
| Files | |
| Reminder | |
| Status | Open |
| URL | |
| Updated | @2022년 5월 24일 오후 10:58 |

1. 딥러닝을 구동하는데 필요한 케라스 함수 호출

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

import numpy as np
import pandas as pd
import tensorflow as tf
```

2. 실행할 때마다 같을 결과 출력하기 위해 설정해주기

```
np.random.seed(42)      # 넘파이의 random 함수의 테이블의 값 지정
tf.random.set_seed(42)  # set_seed로 지정된 난수 값 사용
```

3. 준비된 수술 환자 데이터 불러오기

```
dataset = np.loadtxt(my_data, delimiter = ',')
```

```
array([[  6.   , 148.   ,  72.   , ...,   0.627,  50.   ,   1.   ],
       [  1.   ,  85.   ,  66.   , ...,   0.351,  31.   ,   0.   ],
       [  8.   , 183.   ,  64.   , ...,   0.672,  32.   ,   1.   ],
       ...,
       [  5.   , 121.   ,  72.   , ...,   0.245,  30.   ,   0.   ],
       [  1.   , 126.   ,  60.   , ...,   0.349,  47.   ,   1.   ],
       [  1.   ,  93.   ,  70.   , ...,   0.315,  23.   ,   0.   ]])
```

4. 환자의 기록과 수술 결과를 x와 y로 구분하여 저장

```
x = dataset[:,:-1]
y = dataset[:,-1]
```

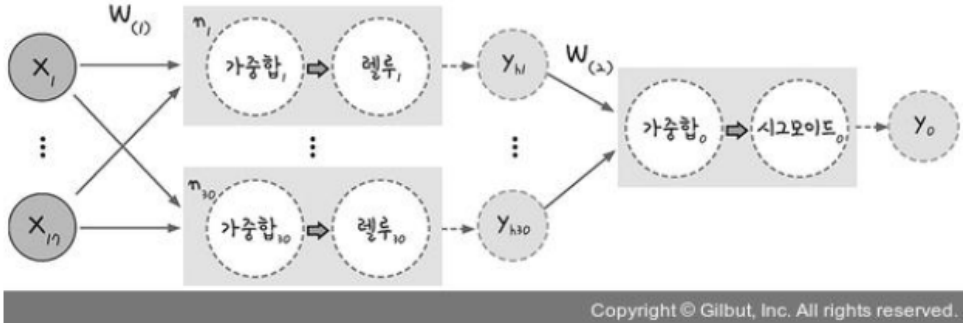
5. 모델 설정 (은닉층 2개, 출력층 1개)

```
model = Sequential() # 1
model.add(Dense(12, input_dim = 8, activation = 'relu')) # 2
model.add(Dense(8, activation = 'relu'))
model.add(Dense(1, activation = 'sigmoid')) # 3
```

- 1) 딥러닝의 구조를 짜고 층을 설정
- 2) 12개의 노드 생성, 8개 입력값 설정, 활성화함수 `relu` 사용

  → 데이터에서 8개의 값을 받아 은닉층의 12개 노드로 보낸다는 뜻
- 3) 이진 분류(1,0) → 활성화 함수 `sigmoid` 사용

  다중 분류 → 활성화 함수 `softmax` 사용 : 총합이 1인 형태로 바꿔서 계산해줌.
- add 라인이 세 개이므로 세 개의 층을 가진 모델을 만드는 것(입력층+은닉층-은닉층-출력층)



## 6. 모델 컴파일(환경 설정)

```
model.compile(loss = 'binary_crossentropy', # 오차함수 : 이진 교차 엔트로피
              optimizer = 'adam',           # 최적화함수 : adam
              metrics = ['accuracy'])       # 측정 : 정확도
```

- loss
  - 교차 엔트로피 계열의 함수 : 출력 값에 로그를 취해서, 오차가 커지면 수렴 속도가 빨라지고, 오차가 작아지면 수렴 속도가 감소하게끔 만드는 것
  - 평균 제곱 오차 계열의 함수 : 수렴하기까지 속도가 많이 걸리는 단점.
- metrics
  - 모델이 컴파일될 때 모델 수행 결과를 나타내게끔 설정

* 실제 값을 yt, 예측 값을 yo라고 가정할 때

| | | |
|---|---|---|
| 평균 제곱 계열 | mean_squared_error | 평균 제곱 오차<br>계산: mean(square(yt - yo)) |
| | mean_absolute_error | 평균 절대 오차(실제 값과 예측 값 차이의 절댓값 평균)<br>계산: mean(abs(yt - yo)) |
| | mean_absolute_percentage_error | 평균 절대 백분율 오차(절댓값 오차를 절댓값으로 나눈 후 평균)<br>계산: mean(abs(yt - yo)/abs(yt)) (단, 분모 ≠ 0) |
| | mean_squared_logarithmic_error | 평균 제곱 로그 오차(실제 값과 예측 값에 로그를 적용한 값의 차이를 제곱한 값의 평균)<br>계산: mean(square((log(yo) + 1) - (log(yt) + 1))) |
| 교차 엔트로피 계열 | categorical_crossentropy | 범주형 교차 엔트로피(일반적인 분류) |
| | binary_crossentropy | 이항 교차 엔트로피(두 개의 클래스 중에서 예측할 때) |

## 7. 모델 실행

```
model.fit(x,y, epochs = 200, batch_size = 10)
```

- 실행 : fit
- epochs : 반복 횟수
- batch_size : 배치 단위 10개 묶어서 실행함

```
Epoch 1/200
77/77 [==============================] - 1s 2ms/step - loss: 2.6924 - accuracy: 0.4896
Epoch 2/200
77/77 [==============================] - 0s 2ms/step - loss: 1.2286 - accuracy: 0.5703
Epoch 3/200
77/77 [==============================] - 0s 2ms/step - loss: 1.1255 - accuracy: 0.6211
Epoch 4/200
77/77 [==============================] - 0s 2ms/step - loss: 1.0235 - accuracy: 0.5990
Epoch 5/200
77/77 [==============================] - 0s 3ms/step - loss: 0.9394 - accuracy: 0.6120
```

## 8. 결과 출력

```
print('\n Accuracy : %.4f'%(model.evaluate(x,y)[1]))
```

```
24/24 [==============================] - 0s 1ms/step - loss: 0.4551 - accuracy: 0.7917

 Accuracy : 0.7917
```