데이터 가져오기

In [1]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
```

In [2]:
```
data = pd.read_csv('diabetes.csv')
data.head()
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

데이터 전처리

null값 제거

In [3]:
```
data = data.dropna()  # null 값 제거
```

X,Y 데이터 설정

In [4]:
```
X = data.drop('Outcome',axis = 1)
X.head()
X = np.asarray(X)
print(X)
```

Out[4]:
```
[[  6.    148.     72.    ...  33.6    0.627  50.   ]
 [  1.     85.     66.    ...  26.6    0.351  31.   ]
 [  8.    183.     64.    ...  23.3    0.672  32.   ]
 ...
 [  5.    121.     72.    ...  26.2    0.245  30.   ]
 [  1.    126.     60.    ...  30.1    0.349  47.   ]
 [  1.     93.     70.    ...  30.4    0.315  23.   ]]
```

In [5]:
```
Y = data['Outcome'].values
Y = np.asarray(Y)
np.min(X,axis =0)
np.max(Y,axis = 0)
```

Out[5]:
```
1
```

데이터 노멀라이즈 진행

In [6]:
```
# 데이터 노멀라이즈
X = ((X - np.min(X, axis = 0)) / (np.max(X, axis = 0) - np.min(X, axis = 0)))
X
```

Out[6]:
```
array([[0.35294118, 0.74371859, 0.59016393, ..., 0.50074516, 0.23441503,
        0.48333333],
       [0.05882353, 0.42713568, 0.54098361, ..., 0.39642325, 0.11656704,
        0.16666667],
       [0.47058824, 0.91959799, 0.52459016, ..., 0.34724292, 0.25362938,
        0.18333333],
       ...,
       [0.29411765, 0.6080402 , 0.59016393, ..., 0.3900462 , 0.07133658,
        0.15      ],
       [0.05882353, 0.63316583, 0.49180328, ..., 0.4485842 , 0.11573307,
        0.43333333],
       [0.05882353, 0.46733668, 0.57377049, ..., 0.45305514, 0.10139554,
        0.03333333]])
```

훈련용 테스트용 데이터 분리

In [7]:
```
train_x = X[:int(len(X) * 0.8)]
train_y = Y[:int(len(X) * 0.8)]
test_x = X[int(len(X) * 0.8):]
test_y = Y[int(len(X) * 0.8):]
```

In [8]:
```
train_x.shape
```

Out[8]:
```
(614, 8)
```

In [9]:
```
train_y.shape
```

Out[9]:
```
(614,)
```

In [10]:
```
# 훈련용,테스트용 데이터 분리2
#from sklearn.model_selection import train_test_split

#train_x,test_x, train_y, test_y = \
#    train_test_split(X,Y,test_size = 0.2)
```

In [11]:
```
#print(train_x.shape, test_x.shape)
```

신경망 구축

In [12]:
```
from tensorflow import keras
EPOCHS = 100
```

In [13]:
```
model = keras.Sequential([
    keras.layers.Dense(8), #입력층
    keras.layers.Dense(64,activation = "tanh"),
    keras.layers.Dense(64,activation = "tanh"),
    keras.layers.Dense(256,activation = "tanh"),
    keras.layers.Dense(256,activation = "relu"),
    keras.layers.Dropout(rate = 0.3),
    keras.layers.Dense(1,activation = "sigmoid")
])
```

In [14]:
```
#  0,1분류는 binary_crossentropy
model.compile(optimizer='adam', loss = "binary_crossentropy",metrics = ['accuracy'])
```

In [15]:
```
early_stop = keras.callbacks.EarlyStopping(monitor = 'val_loss',patience= 20)
```

In [16]:
```
history = model.fit(train_x,train_y, epochs = EPOCHS,
                    validation_data = (test_x, test_y),
                    callbacks = [early_stop])
```

```
2022-01-07 09:42:30.139163: I tensorflow/core/platform/cpu_feature_guard.cc:145] This TensorFlow binary is optimized with Intel(R) MKL-DNN to use the following CPU instructions in performance critical operations:  SSE4.1 SSE4.2
To enable them in non-MKL-DNN operations, rebuild TensorFlow with the appropriate compiler flags.
2022-01-07 09:42:30.139571: I tensorflow/core/common_runtime/process_util.cc:115] Creating new thread pool with default inter op setting: 8. Tune using inter_op_parallelism_threads for best performance.
Train on 614 samples, validate on 154 samples
Epoch 1/100
614/614 [==============================] - 1s 2ms/sample - loss: 0.6645 - accuracy: 0.6544 - val_loss: 0.6267 - val_accuracy: 0.6494
Epoch 2/100
614/614 [==============================] - 0s 252us/sample - loss: 0.6173 - accuracy: 0.6645 - val_loss: 0.5838 - val_accuracy: 0.7403
Epoch 3/100
614/614 [==============================] - 0s 247us/sample - loss: 0.5815 - accuracy: 0.6987 - val_loss: 0.5413 - val_accuracy: 0.7338
Epoch 4/100
614/614 [==============================] - 0s 249us/sample - loss: 0.5593 - accuracy: 0.7280 - val_loss: 0.5368 - val_accuracy: 0.7597
Epoch 5/100
614/614 [==============================] - 0s 246us/sample - loss: 0.5479 - accuracy: 0.7280 - val_loss: 0.5167 - val_accuracy: 0.7273
Epoch 6/100
614/614 [==============================] - 0s 253us/sample - loss: 0.5277 - accuracy: 0.7362 - val_loss: 0.4883 - val_accuracy: 0.7857
Epoch 7/100
614/614 [==============================] - 0s 277us/sample - loss: 0.5108 - accuracy: 0.7590 - val_loss: 0.4787 - val_accuracy: 0.7727
Epoch 8/100
614/614 [==============================] - 0s 249us/sample - loss: 0.5091 - accuracy: 0.7394 - val_loss: 0.4805 - val_accuracy: 0.7727
Epoch 9/100
614/614 [==============================] - 0s 254us/sample - loss: 0.5000 - accuracy: 0.7655 - val_loss: 0.5027 - val_accuracy: 0.7273
Epoch 10/100
614/614 [==============================] - 0s 246us/sample - loss: 0.4913 - accuracy: 0.7590 - val_loss: 0.5146 - val_accuracy: 0.7208
Epoch 11/100
614/614 [==============================] - 0s 246us/sample - loss: 0.4991 - accuracy: 0.7671 - val_loss: 0.4706 - val_accuracy: 0.7662
Epoch 12/100
614/614 [==============================] - 0s 255us/sample - loss: 0.4757 - accuracy: 0.7752 - val_loss: 0.4646 - val_accuracy: 0.7922
Epoch 13/100
614/614 [==============================] - 0s 329us/sample - loss: 0.4659 - accuracy: 0.7866 - val_loss: 0.4676 - val_accuracy: 0.7792
Epoch 14/100
614/614 [==============================] - 0s 253us/sample - loss: 0.4716 - accuracy: 0.7818 - val_loss: 0.4710 - val_accuracy: 0.7857
Epoch 15/100
614/614 [==============================] - 0s 252us/sample - loss: 0.4667 - accuracy: 0.7801 - val_loss: 0.4757 - val_accuracy: 0.7792
Epoch 16/100
614/614 [==============================] - 0s 259us/sample - loss: 0.4745 - accuracy: 0.7655 - val_loss: 0.4709 - val_accuracy: 0.7597
Epoch 17/100
614/614 [==============================] - 0s 254us/sample - loss: 0.4678 - accuracy: 0.7834 - val_loss: 0.5141 - val_accuracy: 0.7662
Epoch 18/100
614/614 [==============================] - 0s 251us/sample - loss: 0.4718 - accuracy: 0.7866 - val_loss: 0.4719 - val_accuracy: 0.7922
Epoch 19/100
614/614 [==============================] - 0s 248us/sample - loss: 0.4864 - accuracy: 0.7801 - val_loss: 0.4968 - val_accuracy: 0.7727
Epoch 20/100
614/614 [==============================] - 0s 247us/sample - loss: 0.4723 - accuracy: 0.7671 - val_loss: 0.4704 - val_accuracy: 0.8117
Epoch 21/100
614/614 [==============================] - 0s 259us/sample - loss: 0.4629 - accuracy: 0.7834 - val_loss: 0.4822 - val_accuracy: 0.7727
Epoch 22/100
614/614 [==============================] - 0s 252us/sample - loss: 0.4518 - accuracy: 0.7850 - val_loss: 0.4746 - val_accuracy: 0.7468
Epoch 23/100
614/614 [==============================] - 0s 261us/sample - loss: 0.4693 - accuracy: 0.7818 - val_loss: 0.4961 - val_accuracy: 0.7468
Epoch 24/100
614/614 [==============================] - 0s 253us/sample - loss: 0.4600 - accuracy: 0.7769 - val_loss: 0.4766 - val_accuracy: 0.7792
Epoch 25/100
614/614 [==============================] - 0s 249us/sample - loss: 0.4633 - accuracy: 0.7785 - val_loss: 0.4808 - val_accuracy: 0.7597
Epoch 26/100
614/614 [==============================] - 0s 256us/sample - loss: 0.4655 - accuracy: 0.7948 - val_loss: 0.4799 - val_accuracy: 0.7338
Epoch 27/100
614/614 [==============================] - 0s 251us/sample - loss: 0.4616 - accuracy: 0.7850 - val_loss: 0.4635 - val_accuracy: 0.7922
Epoch 28/100
614/614 [==============================] - 0s 243us/sample - loss: 0.4660 - accuracy: 0.7736 - val_loss: 0.4619 - val_accuracy: 0.7922
Epoch 29/100
614/614 [==============================] - 0s 243us/sample - loss: 0.4647 - accuracy: 0.7883 - val_loss: 0.4685 - val_accuracy: 0.7987
Epoch 30/100
614/614 [==============================] - 0s 254us/sample - loss: 0.4817 - accuracy: 0.7638 - val_loss: 0.4727 - val_accuracy: 0.7987
Epoch 31/100
614/614 [==============================] - 0s 243us/sample - loss: 0.4535 - accuracy: 0.7948 - val_loss: 0.4766 - val_accuracy: 0.7468
Epoch 32/100
614/614 [==============================] - 0s 267us/sample - loss: 0.4484 - accuracy: 0.7818 - val_loss: 0.4647 - val_accuracy: 0.7727
Epoch 33/100
614/614 [==============================] - 0s 258us/sample - loss: 0.4590 - accuracy: 0.7785 - val_loss: 0.5057 - val_accuracy: 0.7273
Epoch 34/100
614/614 [==============================] - 0s 249us/sample - loss: 0.4491 - accuracy: 0.7899 - val_loss: 0.5691 - val_accuracy: 0.7468
Epoch 35/100
614/614 [==============================] - 0s 252us/sample - loss: 0.4621 - accuracy: 0.7899 - val_loss: 0.4751 - val_accuracy: 0.7857
Epoch 36/100
614/614 [==============================] - 0s 259us/sample - loss: 0.4428 - accuracy: 0.7785 - val_loss: 0.4670 - val_accuracy: 0.7922
Epoch 37/100
614/614 [==============================] - 0s 254us/sample - loss: 0.4443 - accuracy: 0.7915 - val_loss: 0.4633 - val_accuracy: 0.7922
Epoch 38/100
614/614 [==============================] - 0s 259us/sample - loss: 0.4383 - accuracy: 0.7850 - val_loss: 0.4613 - val_accuracy: 0.7857
Epoch 39/100
614/614 [==============================] - 0s 255us/sample - loss: 0.4376 - accuracy: 0.7883 - val_loss: 0.4725 - val_accuracy: 0.7857
Epoch 40/100
614/614 [==============================] - 0s 255us/sample - loss: 0.4367 - accuracy: 0.7883 - val_loss: 0.4802 - val_accuracy: 0.7857
Epoch 41/100
614/614 [==============================] - 0s 254us/sample - loss: 0.4481 - accuracy: 0.7980 - val_loss: 0.4637 - val_accuracy: 0.7727
Epoch 42/100
614/614 [==============================] - 0s 248us/sample - loss: 0.4412 - accuracy: 0.7801 - val_loss: 0.4661 - val_accuracy: 0.8052
Epoch 43/100
614/614 [==============================] - 0s 259us/sample - loss: 0.4342 - accuracy: 0.7818 - val_loss: 0.4603 - val_accuracy: 0.7662
Epoch 44/100
614/614 [==============================] - 0s 250us/sample - loss: 0.4512 - accuracy: 0.7720 - val_loss: 0.4731 - val_accuracy: 0.7662
Epoch 45/100
614/614 [==============================] - 0s 255us/sample - loss: 0.4354 - accuracy: 0.7883 - val_loss: 0.4774 - val_accuracy: 0.7987
Epoch 46/100
614/614 [==============================] - 0s 252us/sample - loss: 0.4311 - accuracy: 0.7883 - val_loss: 0.4759 - val_accuracy: 0.7922
Epoch 47/100
614/614 [==============================] - 0s 251us/sample - loss: 0.4416 - accuracy: 0.7785 - val_loss: 0.4721 - val_accuracy: 0.7857
Epoch 48/100
614/614 [==============================] - 0s 240us/sample - loss: 0.4414 - accuracy: 0.7866 - val_loss: 0.4806 - val_accuracy: 0.7792
Epoch 49/100
614/614 [==============================] - 0s 251us/sample - loss: 0.4337 - accuracy: 0.7866 - val_loss: 0.4524 - val_accuracy: 0.7987
Epoch 50/100
614/614 [==============================] - 0s 240us/sample - loss: 0.4500 - accuracy: 0.7915 - val_loss: 0.5212 - val_accuracy: 0.7532
Epoch 51/100
614/614 [==============================] - 0s 256us/sample - loss: 0.4408 - accuracy: 0.7752 - val_loss: 0.4804 - val_accuracy: 0.7987
Epoch 52/100
614/614 [==============================] - 0s 251us/sample - loss: 0.4216 - accuracy: 0.7915 - val_loss: 0.4680 - val_accuracy: 0.7597
Epoch 53/100
614/614 [==============================] - 0s 249us/sample - loss: 0.4273 - accuracy: 0.7899 - val_loss: 0.4590 - val_accuracy: 0.8052
Epoch 54/100
614/614 [==============================] - 0s 253us/sample - loss: 0.4331 - accuracy: 0.7948 - val_loss: 0.4691 - val_accuracy: 0.8117
Epoch 55/100
614/614 [==============================] - 0s 254us/sample - loss: 0.4442 - accuracy: 0.8029 - val_loss: 0.4674 - val_accuracy: 0.7987
Epoch 56/100
614/614 [==============================] - 0s 244us/sample - loss: 0.4382 - accuracy: 0.7915 - val_loss: 0.4520 - val_accuracy: 0.7987
```

모델의 정확도 81%

In [17]:
```
np.max(history.history["val_accuracy"])
```

Out[17]:
```
0.8116883
```

In [18]:
```
# 예측값 비교해보기
pred = model.predict(train_x[:5])
pred
```

Out[18]:
```
array([[5.49149512],
       [0.0321609 ],
       [0.8320242 ],
       [0.09425655],
       [0.9574676 ]], dtype=float32)
```

In [19]:
```
train_y[:5]
```

Out[19]:
```
array([1, 0, 1, 0, 1])
```

모델 시각화

In [20]:
```
import matplotlib.pyplot as plt
```

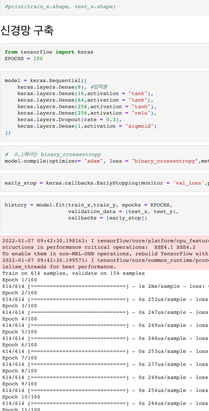
In [21]:
```
train_history = history.history["loss"]
validate_history = history.history["val_loss"]

fig = plt.figure(figsize = (8,8))
plt.title("loss history")
plt.xlabel("EPOCH")
plt.ylabel("LOSS Function")
plt.plot(train_history, "red")
plt.plot(validate_history,"blue")
plt.show()
```

In [22]:
```
train_history = history.history['accuracy']
validate_history = history.history['val_accuracy']

fig = plt.figure(figsize = (8,8))
plt.title('accuracy history')
plt.xlabel('EPOCH')
plt.ylabel('accuracy Function')
plt.plot(train_history, "red")
plt.plot(validate_history,"blue")

plt.show()
```



In [ ]:
```
# 감사합니다.
```

In [ ]:
```

```