XOR 이해하기 with Keras

```
# tensorflow와 tf.keras를 임포트합니다
import tensorflow as tf
from tensorflow import keras
```

```
tf.__version__
```

```
'2.3.0'
```

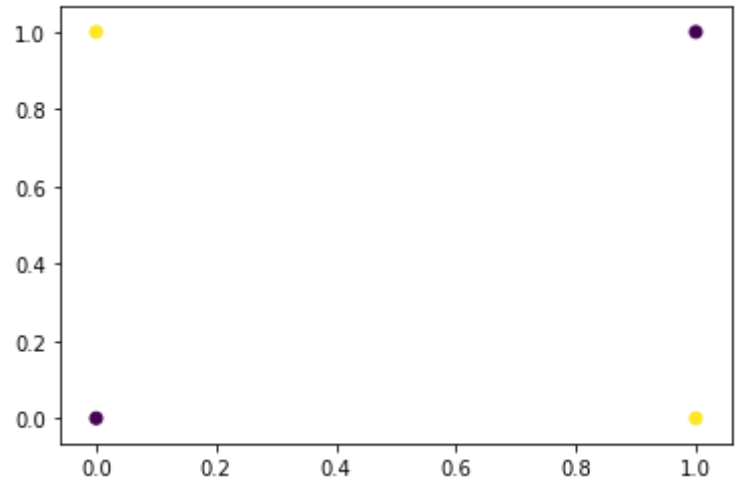
```
# 헬퍼(helper) 라이브러리를 임포트합니다
import numpy as np
import matplotlib.pyplot as plt
```

Remember !! XOR : 모두 같으면 0

```
x_data = [[0, 0],
          [0, 1],
          [1, 0],
          [1, 1]]

y_data = [0,
          1,
          1,
          0]

x_data = np.array(x_data, dtype=np.float32)
y_data = np.array(y_data, dtype=np.float32)
```

```
plt.scatter(x_data[:, 0], x_data[:, 1], c=y_data)
```

```
<matplotlib.collections.PathCollection at 0x7f239b8c6d68>
```

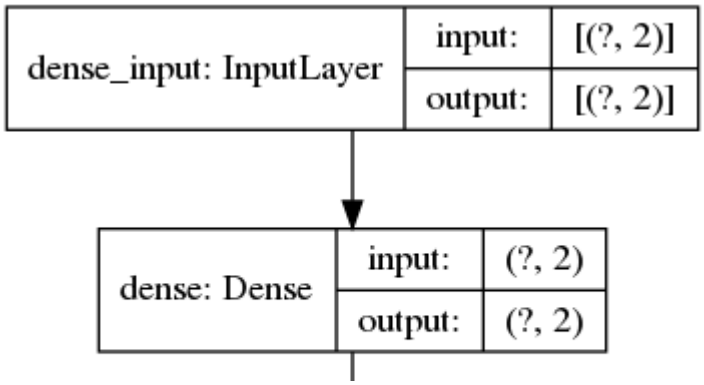



```
from tensorflow.keras import layers
from tensorflow.keras import activations
```

```
from tensorflow.keras import optimizers
from tensorflow.keras import models


model = models.Sequential()
model.add(layers.Dense(2, input_dim=2))
model.add(layers.Activation('tanh'))
#model.add(layers.Dense(2)) # comment out
#model.add(layers.Activation('tanh')) # comment out
model.add(layers.Dense(1))
model.add(layers.Activation('sigmoid'))


sgd = optimizers.SGD(lr=0.1)
model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])


model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 2)                 6
_________________________________________________________________
activation (Activation)      (None, 2)                 0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 3
_________________________________________________________________
activation_1 (Activation)    (None, 1)                 0
=================================================================
Total params: 9
Trainable params: 9
Non-trainable params: 0
_________________________________________________________________
```

```
from tensorflow.keras.utils import plot_model
plot_model(model, to_file='model_xor.png', show_shapes=True)
```




```
history = model.fit(x_data, y_data,  batch_size=1, epochs=500)
```

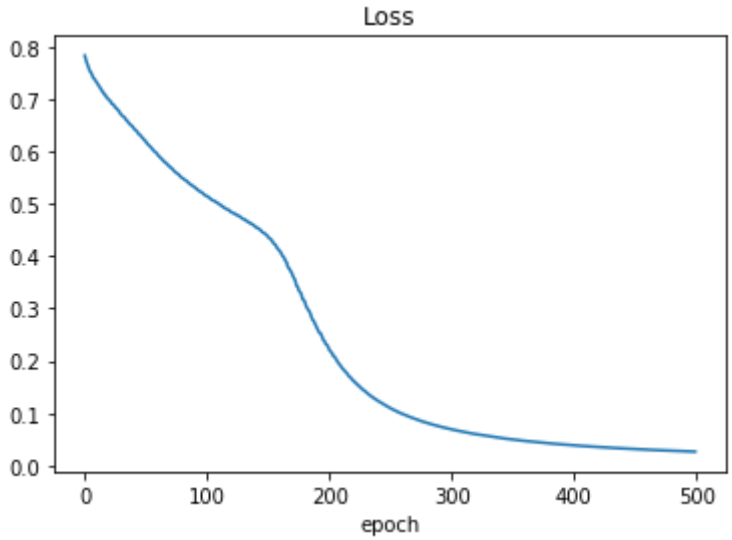
```
Epoch 1/500
4/4 [==============================] - 0s 1ms/step - loss: 0.7831 - accuracy: 0.2500
Epoch 2/500
4/4 [==============================] - 0s 2ms/step - loss: 0.7738 - accuracy: 0.2500
Epoch 3/500
4/4 [==============================] - 0s 1ms/step - loss: 0.7675 - accuracy: 0.2500
Epoch 4/500
4/4 [==============================] - 0s 2ms/step - loss: 0.7617 - accuracy: 0.2500
Epoch 5/500
4/4 [==============================] - 0s 2ms/step - loss: 0.7540 - accuracy: 0.2500
Epoch 6/500
4/4 [==============================] - 0s 1ms/step - loss: 0.7511 - accuracy: 0.2500
Epoch 7/500
4/4 [==============================] - 0s 2ms/step - loss: 0.7458 - accuracy: 0.5000
Epoch 8/500
4/4 [==============================] - 0s 2ms/step - loss: 0.7403 - accuracy: 0.5000
Epoch 9/500
4/4 [==============================] - 0s 1ms/step - loss: 0.7381 - accuracy: 0.2500
Epoch 10/500
4/4 [==============================] - 0s 1ms/step - loss: 0.7344 - accuracy: 0.2500
Epoch 11/500
4/4 [==============================] - 0s 2ms/step - loss: 0.7307 - accuracy: 0.5000
Epoch 12/500
4/4 [==============================] - 0s 1ms/step - loss: 0.7271 - accuracy: 0.7500
Epoch 13/500
4/4 [==============================] - 0s 1ms/step - loss: 0.7239 - accuracy: 0.2500
Epoch 14/500
4/4 [==============================] - 0s 2ms/step - loss: 0.7200 - accuracy: 0.7500
Epoch 15/500
4/4 [==============================] - 0s 2ms/step - loss: 0.7152 - accuracy: 0.5000
Epoch 16/500
4/4 [==============================] - 0s 2ms/step - loss: 0.7136 - accuracy: 0.5000
Epoch 17/500
4/4 [==============================] - 0s 2ms/step - loss: 0.7085 - accuracy: 0.5000
Epoch 18/500
4/4 [==============================] - 0s 2ms/step - loss: 0.7074 - accuracy: 0.5000
Epoch 19/500
4/4 [==============================] - 0s 2ms/step - loss: 0.7026 - accuracy: 0.7500
Epoch 20/500
4/4 [==============================] - 0s 1ms/step - loss: 0.7002 - accuracy: 0.5000
Epoch 21/500
4/4 [==============================] - 0s 2ms/step - loss: 0.6985 - accuracy: 0.7500
Epoch 22/500
4/4 [==============================] - 0s 2ms/step - loss: 0.6954 - accuracy: 0.5000
Epoch 23/500
4/4 [==============================] - 0s 2ms/step - loss: 0.6913 - accuracy: 0.7500
Epoch 24/500
```

```
4/4 [==============================] - 0s 2ms/step - loss: 0.6902 - accuracy: 0.7500
Epoch 25/500
4/4 [==============================] - 0s 1ms/step - loss: 0.6877 - accuracy: 0.7500
Epoch 26/500
4/4 [==============================] - 0s 1ms/step - loss: 0.6835 - accuracy: 0.5000
Epoch 27/500
4/4 [==============================] - 0s 1ms/step - loss: 0.6823 - accuracy: 0.7500
Epoch 28/500
4/4 [==============================] - 0s 2ms/step - loss: 0.6791 - accuracy: 0.7500
Epoch 29/500
4/4 [==============================] - 0s 1ms/step - loss: 0.6768 - accuracy: 0.7500
Epoch 30/500
```

학습 결과 그려보기

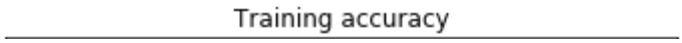
```
plt.plot(history.history['loss'])
plt.title('Loss')
plt.xlabel('epoch')
```

```
Text(0.5, 0, 'epoch')
```




```
plt.plot(history.history['accuracy'])
plt.title('Training accuracy')
plt.xlabel('epoch')
```

```
Text(0.5, 0, 'epoch')
```



확률 찍어보기


```
hypothesis = model.predict(x_data)
print(hypothesis)
```

```
[[0.03096818]
 [0.981488  ]
 [0.98172015]
 [0.03512011]]
```

```
predicted = hypothesis > 0.5
print(predicted)
```

```
[[False]
 [ True]
 [ True]
 [False]]
```

생각해보기

- XOR 분류에서 정확도 75의 의미는?

그래프를 보면 epoch가 200쯤에서 정확도가 75에서 100으로 변하기 시작하는데

아마 AND게이트의 확률이 75퍼센트이므로 AND게이트의 구조와 XOR게이트의 구조를 컴퓨터가 같게생각하다가

정확도가 100이되면 이게이트는 AND가아니라 XOR이구나라는걸 이해하는 마지노선인거같습니다.