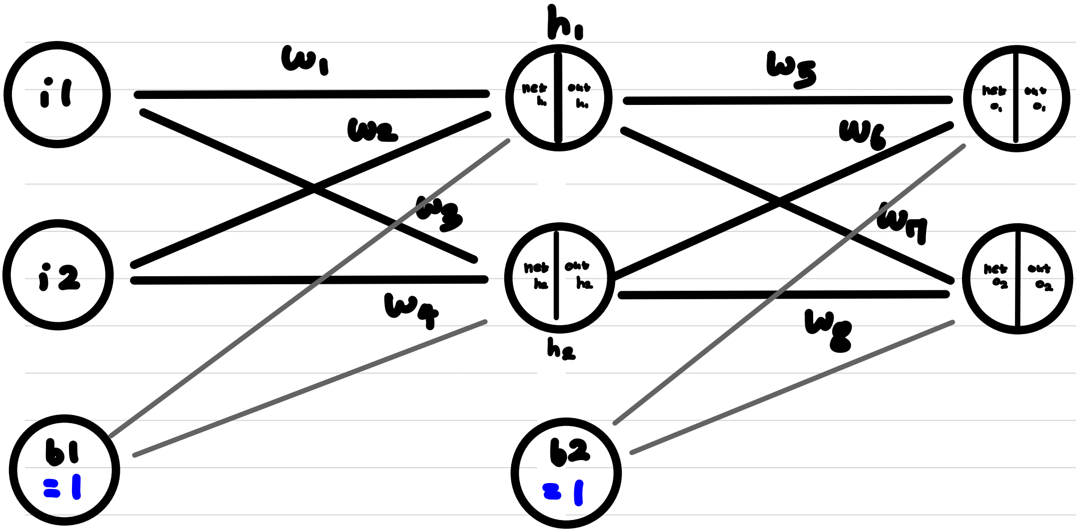

i1
i2
b1
=1
w1
w2
w3
w4
h1
h2
net h1
out h1
net h2
out h2
w5
w6
w7
w8
b2
=1
net o1
out o1
net o2
out o2

$w_n = w_n - \eta \frac{\partial E}{\partial w_n}$, Let's find $\frac{\partial E}{\partial w_n}$

$$\frac{\partial E_t}{\partial w_1} = \frac{\partial E_t}{\partial out_{h_1}} \frac{\partial out_{h_1}}{\partial net_{h_1}} \frac{\partial net_{h_1}}{\partial w_1} = \frac{\partial E_{o_1}}{\partial out_{o_1}} \frac{\partial out_{o_1}}{\partial net_{o_1}} \frac{\partial net_{o_1}}{\partial out_{h_1}} \frac{\partial out_{h_1}}{\partial net_{h_1}} \frac{\partial net_{h_1}}{\partial w_1} + \frac{\partial E_{o_2}}{\partial out_{o_2}} \frac{\partial out_{o_2}}{\partial net_{o_2}} \frac{\partial net_{o_2}}{\partial out_{h_1}} \frac{\partial out_{h_1}}{\partial net_{h_1}} \frac{\partial net_{h_1}}{\partial w_1}$$

$$\frac{\partial E_t}{\partial w_2} = \frac{\partial E_t}{\partial out_{h_1}} \frac{\partial out_{h_1}}{\partial net_{h_1}} \frac{\partial net_{h_1}}{\partial w_2} = \frac{\partial E_{o_1}}{\partial out_{o_1}} \frac{\partial out_{o_1}}{\partial net_{o_1}} \frac{\partial net_{o_1}}{\partial out_{h_1}} \frac{\partial out_{h_1}}{\partial net_{h_1}} \frac{\partial net_{h_1}}{\partial w_2} + \frac{\partial E_{o_2}}{\partial out_{o_2}} \frac{\partial out_{o_2}}{\partial net_{o_2}} \frac{\partial net_{o_2}}{\partial out_{h_1}} \frac{\partial out_{h_1}}{\partial net_{h_1}} \frac{\partial net_{h_1}}{\partial w_2}$$

$$\frac{\partial E_t}{\partial w_3} = \frac{\partial E_t}{\partial out_{h_2}} \frac{\partial out_{h_2}}{\partial net_{h_2}} \frac{\partial net_{h_2}}{\partial w_3} = \frac{\partial E_{o_1}}{\partial out_{o_1}} \frac{\partial out_{o_1}}{\partial net_{o_1}} \frac{\partial net_{o_1}}{\partial out_{h_2}} \frac{\partial out_{h_2}}{\partial net_{h_2}} \frac{\partial net_{h_2}}{\partial w_3} + \frac{\partial E_{o_2}}{\partial out_{o_2}} \frac{\partial out_{o_2}}{\partial net_{o_2}} \frac{\partial net_{o_2}}{\partial out_{h_2}} \frac{\partial out_{h_2}}{\partial net_{h_2}} \frac{\partial net_{h_2}}{\partial w_3}$$

$$\frac{\partial E_t}{\partial w_4} = \frac{\partial E_t}{\partial out_{h_2}} \frac{\partial out_{h_2}}{\partial net_{h_2}} \frac{\partial net_{h_2}}{\partial w_4} = \frac{\partial E_{o_1}}{\partial out_{o_1}} \frac{\partial out_{o_1}}{\partial net_{o_1}} \frac{\partial net_{o_1}}{\partial out_{h_2}} \frac{\partial out_{h_2}}{\partial net_{h_2}} \frac{\partial net_{h_2}}{\partial w_4} + \frac{\partial E_{o_2}}{\partial out_{o_2}} \frac{\partial out_{o_2}}{\partial net_{o_2}} \frac{\partial net_{o_2}}{\partial out_{h_2}} \frac{\partial out_{h_2}}{\partial net_{h_2}} \frac{\partial net_{h_2}}{\partial w_4}$$

$$\frac{\partial E_t}{\partial w_5} = \frac{\partial E_{o_1}}{\partial out_{o_1}} \frac{\partial out_{o_1}}{\partial net_{o_1}} \frac{\partial net_{o_1}}{\partial w_5}$$

$$\frac{\partial E_t}{\partial w_6} = \frac{\partial E_{o_1}}{\partial out_{o_1}} \frac{\partial out_{o_1}}{\partial net_{o_1}} \frac{\partial net_{o_1}}{\partial w_6}$$

$$\frac{\partial E_t}{\partial w_7} = \frac{\partial E_{o_2}}{\partial out_{o_2}} \frac{\partial out_{o_2}}{\partial net_{o_2}} \frac{\partial net_{o_2}}{\partial w_7}$$

$$\frac{\partial E_t}{\partial w_8} = \frac{\partial E_{o_2}}{\partial out_{o_2}} \frac{\partial out_{o_2}}{\partial net_{o_2}} \frac{\partial net_{o_2}}{\partial w_8}$$

- $E_t = E_{o_1} + E_{o_2}$

  $E_{o_1} = \frac{1}{2}(target_{o_1} - out_{o_1})^2$, $E_{o_2} = \frac{1}{2}(target_{o_2} - out_{o_2})^2$

  $\frac{\partial E_{o_1}}{\partial out_{o_1}} = -(target_{o_1} - out_{o_1})$, $\frac{\partial E_{o_2}}{\partial out_{o_2}} = -(target_{o_2} - out_{o_2})$

- $out_{o_1} = \frac{1}{1+e^{-net_{o_1}}}$, $out_{o_2} = \frac{1}{1+e^{-net_{o_2}}}$

  $\frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right) = -(1+e^{-x})^{-2} \cdot -e^{-x} = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}}\left(1 - \frac{1}{1+e^{-x}}\right) = out(1-out)$

  $\frac{\partial out_{o_1}}{\partial net_{o_1}} = out_{o_1}(1-out_{o_1})$, $\frac{\partial out_{o_2}}{\partial net_{o_2}} = out_{o_2}(1-out_{o_2})$

- $net_{o_1} = w_5 * out_{h_1} + w_6 * out_{h_2} + b_2 * 1$

$$\frac{\partial net_{o_1}}{\partial out_{h_1}} = w_5, \quad \frac{\partial net_{o_1}}{\partial out_{h_2}} = w_6$$

$net_{o_2} = w_7 * out_{h_1} + w_8 * out_{h_2} + b_2 * 1$

$$\frac{\partial net_{o_2}}{\partial out_{h_1}} = w_7, \quad \frac{\partial net_{o_2}}{\partial out_{h_2}} = w_8$$

- $net_{o_1} = w_5 * out_{h_1} + w_6 * out_{h_2} + b_2 * 1$

$$\frac{\partial net_{o_1}}{\partial w_n} = out_{h_{n-4}}$$

$net_{o_2} = w_7 * out_{h_1} + w_8 * out_{h_2} + b_2 * 1$

$$\frac{\partial net_{o_2}}{\partial w_n} = out_{h_{n-6}}$$

- $out_{h_1} = \frac{1}{1+e^{-net_{h_1}}}, \quad out_{h_2} = \frac{1}{1+e^{-net_{h_2}}}$

$$\frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right) = out(1-out)$$

$$\frac{\partial out_{h_1}}{\partial net_{h_1}} = out_{h_1}(1-out_{h_1}), \quad \frac{\partial out_{h_2}}{\partial net_{h_2}} = out_{h_2}(1-out_{h_2})$$

- $net_{h_1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1, \quad net_{h_2} = w_3 * i_1 + w_4 * i_2 + b_1 * 1$

$$\frac{\partial net_{h_1}}{\partial w_n} = i_n, \quad \frac{\partial net_{h_2}}{\partial w_n} = i_{n-2}$$

2. numpy로 구성한 hidden layer 1개의 신경망

```python
import numpy as np
import math

# neuralNetwork class definition
class neuralNetwork:
    # weights of bias
    bih = 0.35
    bho = 0.60
    # learning rate
    eta = 0.5

    # initialise the neuralNetwork
    def __init__(self, wih, who):
        # link weights
        self.wih = wih
        self.who = who
        # sigmoid function
        self.sig = lambda x: 1 / (1 + pow(math.e, -x))
        self.sigp = lambda x : pow(math.e, -x) / (1 + pow(math.e, -x)) ** 2

        pass

    # train the neuralNetwork
    def train(self, inputs_list, targets_list):
        # convert inputs list to 2d array
        inputs = np.array(inputs_list, ndmin = 2).T
        targets = np.array(targets_list, ndmin = 2).T
        # calculate signals into hidden layer
        hidden_inputs = np.dot(self.wih, inputs) + self.bih * np.ones((2,1))
        # calculate the signals emerging from hidden layer
        hidden_outputs = self.sig(hidden_inputs)
        # calculate signals into final output layer
        final_inputs = np.dot(self.who, hidden_outputs) + self.bho * np.ones((2, 1))
        # calculate the signals emerging from output layer
        final_outputs = self.sig(final_inputs)

        # partial derivative of E with respect to weight
        output_errors = final_outputs - targets
        a = np.dot((output_errors * self.sigp(final_inputs)).T, self.who).T
        dEdw_ih = a * self.sigp(hidden_inputs) * inputs.T
        dEdw_ho = output_errors * self.sigp(final_inputs) * hidden_outputs.T

        # update the weights for the links between the hidden and output layers
        self.wih -= self.eta * dEdw_ih
        # update the weights for the links between the input and hidden layers
        self.who -= self.eta * dEdw_ho

        pass

    def query(self,inputs_list):
        inputs = np.array(inputs_list, ndmin=2).T
        hidden_inputs = np.dot(self.wih, inputs)+self.bih*np.ones((2,1))
        hidden_outputs = self.sig(hidden_inputs)
        final_inputs = np.dot(self.who, hidden_outputs)+self.bho*np.ones((2,1))
        final_outputs = self.sig(final_inputs)

        return final_outputs

wih = np.array([[0.15,0.20],[0.25,0.30]])
who = np.array([[0.40,0.45],[0.50,0.55]])
n = neuralNetwork(wih, who)
inputs_list = [0.05,0.10]
targets_list = [0.01,0.99]

#train
i = 0

while i<10000:
    n.train(inputs_list,targets_list)
    i += 1

outputs = n.query(inputs_list)
```

$$\begin{pmatrix} 0.15 & 0.20 \\ 0.25 & 0.30 \end{pmatrix}$$

$$\begin{pmatrix} 0.40 & 0.45 \\ 0.50 & 0.55 \end{pmatrix}$$

3-1. tensorflow로 구성한 기본 신경망 (inputs, hidden 1, outputs 구성  sequential x

inputs이 2개

ai에서 가장 유명한 패키지 : tensorflow, keras (keras는 tensorflow 위에서 사용)

```
import tensorflow as tf
from keras.models import Model
import numpy as np
from keras.layers import *
```

import * : 해당 모듈안에 있는 모든 함수, 변수, 클래스를 불러온다는 뜻

```
# XOR data
x = np.array([[1, 1], [1, 0], [0, 1], [0, 0]])
y = np.array([[0], [1], [1], [0]])
```

x: inputs
y: outputs

Input: keras에서 정의하는 함수
shape: 정의값은 2개

```
inputs = Input(shape = (2,)) # input tensor
hidden1 = Dense(units = 2, activation = 'sigmoid')(inputs) # hidden layer 1
outputs = Dense(units = 1, activation = 'sigmoid')(hidden1) # hidden layer 2
```

units: 노드 수
activation 함수 : sigmoid
inputs: hidden layer에 들어갈 것
hidden1: outputs에 들어갈 것

신경망 모델의 구조 (inputs, hidden, outputs)을 정의
→ 모델 구조를 정의하면서 가중치를 자동(랜덤)으로 부여 (Dense 함수를 실행하며 부여)
즉, 코드를 돌릴 때마다 가중치가 바뀜.
허나 weight를 내가 부여할 수 있는 설정도 있음.

```
# define the model's start and end point
model = Model(inputs, outputs)
model.compile(optimizer = tf.keras.optimizers.SGD(learning_rate = 0.1), loss = 'mse')
history = model.fit(x, y, epochs = 3000, batch_size = 1)
model.summary()
print(model.predict(x))
```

model = Model(inputs, outputs) : 입력층과 출력층을 지정하여 모델을 정의 (이렇게 정의된 모델은 입력층에서 출력층까지의 연결을 통해 데이터가 흐르는 구조를 가지게 됨)

optimizer: 학습 방법
SGD: 가장 보편적인 학습 방법
loss: 에러 함수

model.compile: 모델을 컴파일함
이때 최적화 알고리즘으로는 SGD(Stochastic Gradient Descent)를, 손실함수로는 MSE (Mean Squared Error)를 사용.

model.fit: model을 fitting (학습) 시킴
x: input
y: output
epochs: 학습(훈련)시킬 횟수
model.summary(): Model 요약

history라는 변수는 model.fit() 함수가 반환하는 학습과정의 기록을 저장하기 위해 사용

print(model.predict(x)): 학습된 모델을 사용하여 입력 데이터(x)에 대한 예측값 출력  이 예측 값은 XOR 게이트의 출력과 유사해야함.

epochs가 3000일 때 결과:

```
[[0.09299458]
 [0.92828345]
 [0.9286832 ]
 [0.06187142]]
```

epochs가 2000일 때 결과:

```
[[0.20424953]
 [0.7612141 ]
 [0.8331123 ]
 [0.24002047]]
```

epochs가 1000일 때 결과:

```
[[0.47590804]
 [0.59198976]
 [0.44076583]
 [0.48591116]]
```

epochs는 학습을(훈련횟수) 이기에 늘리면 결과를 좋히고 줄이면 결과가 안좋아짐!

hidden layer를 2개에서 3개로 늘리고 epochs = 3000일 때 결과값:

```
[[0.14312275]
 [0.8884742 ]
 [0.905     ]
 [0.06803803]]
```

hidden layer를 늘리면 결과가 더 좋아지지 않거나 별 차이가 없음.

⟶ 내가 만드는 신경망에 알맞는 구조(hidden layer, epochs)를 사용해야함.

inputs이 4개

3-2. tensorflow로 구성한 기본 신경망 (inputs, hidden 1, outputs 구성 sequential x)

x, 즉 input의 array를 [[1,1],[1,0],[0,1],[0,0]] 에서

[[1,1,0,0],[1,0,0,0],[0,1,0,0],[0,0,0,0]] 와 같이 4개의 배열로 바꾸면 input의 shape도 4로 바꿔야 한다.

결과는 같다.

```
import tensorflow as tf
from keras.models import Model
import numpy as np
from keras.layers import *

# XOR data
x = np.array([[1, 1, 0, 0], [1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 0, 0]])
y = np.array([[0], [1], [1], [0]])

inputs = Input(shape = (4,)) # input tensor
hidden1 = Dense(units = 2, activation = 'sigmoid')(inputs) # hidden layer 1
outputs = Dense(units = 1, activation = 'sigmoid')(hidden1) # hidden layer 2

# define the model's start and end point
model = Model(inputs, outputs)
model.compile(optimizer = tf.keras.optimizers.SGD(learning_rate = 0.1), loss = 'mse')
history = model.fit(x, y, epochs = 3000, batch_size = 1)
model.summary()
print(model.predict(x))
```

```
[[0.08088168]
 [0.91865736]
 [0.8910634 ]
 [0.09297055]]
```

위의 코드에서 history가 없다면 아래와 같이 값이 이상하게 나온다.

```
[[0.5535673 ]
 [0.5976966 ]
 [0.5891924 ]
 [0.64340854]]
```

왜? model.fit 함수는 model을 학습시키며,

history는 model.fit 함수가 반환하는 학습과정을 기록하기 위해 사용한다.

따라서 history가 없으면 model을 학습시킬 수 없으므로 결과값이 이상하게 나온다.

---

3-3. tensorflow로 구성한 신경망 - inputs, hidden 1, outputs를 따로 구현하지 않고 sequential을 이용하여 한번에 모델의 구조를 정의

```
import tensorflow as tf
import numpy as np

# XOR data
x = np.array([[1, 1], [1, 0], [0, 1], [0, 0]])
y = np.array([[0], [1], [1], [0]])

model = tf.keras.Sequential([tf.keras.layers.Dense(units = 2, activation = 'sigmoid', input_shape = (2,)), tf.keras.layers.Dense(units = 1, activation = 'sigmoid')])

model.compile(optimizer = tf.keras.optimizers.SGD(learning_rate = 0.1), loss = 'mse')
history = model.fit(x, y, epochs = 3000, batch_size = 1)
model.summary()
print(model.predict(x))
```

```
[[0.06960659]
 [0.9049232 ]
 [0.93003523]
 [0.08085   ]]
```

3-4. tensorflow로 구성한 신경망 - tensorflow에 있는 mnist (app data) 라는 dataset (숫자 손글씨 dataset) 에서 손글씨를 판별하는 신경망

mnist:

```
import tensorflow as tf
import numpy as np

# download MNIST dataset
mnist = tf.keras.datasets.mnist

# load MNIST dataset and split to trainset and testset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# normalize the image (0 ~ 255 => 0 ~ 1)
x_train, x_test = x_train / 255.0 , x_test / 255.0

# make neural network model
model = tf.keras.models.Sequential([tf.keras.layers.Flatten(input_shape=(28, 28)), tf.keras.layers.Dense(128, activation = 'relu'), tf.keras.layers.Dense(10, activation = 'softmax')])

# add training setting
model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])

model.fit(x_train, y_train, epochs = 5)
model.evaluate(x_test, y_test, verbose = 2)
```

→ 손으로 쓴 숫자들로 이루어진 대형 database

: MNIST 데이터셋을 로딩하고, 훈련 데이터와 테스트 데이터로 분리

인공지능에서 쓸 수 있도록 load

이미지 데이터

이미지에 대응되는 실제 숫자

0~255의 픽셀을 0~1로 만듦 : 계산의 편의를 위해서

28x28을 일자로 펼쳐서 input 노드 구성

28x28 이미지

hidden layer 128개

output 10개

이미지: 28 × 28

→ 이제 이미지 는 픽셀 (255)

모델을 컴파일 - 최적화 알고리즘: adam

손실함수: sparse_categorical_crossentropy

평가지표: accuracy (정확도)

: 모델 학습

테스트 데이터를 이용해 모델의 성능 평가.

정보출력 여부: verbose = 0 (출력 X)

verbose = 1 (정보를 상세하게 출력)

verbose = 2 (정보를 함축적으로 출력)

```
Epoch 1/5
1875/1875 [==============================] - 10s 5ms/step - loss: 0.2599 - accuracy: 0.9257
Epoch 2/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1150 - accuracy: 0.9661
Epoch 3/5
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0782 - accuracy: 0.9761
Epoch 4/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0580 - accuracy: 0.9818
Epoch 5/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0442 - accuracy: 0.9865
313/313 - 1s - loss: 0.0744 - accuracy: 0.9782 - 654ms/epoch - 2ms/step
[0.07440090924501419, 0.9782000184059143]
```

→ 훈련 데이터 1875개

epochs: 학습(반복) 횟수 ⟶ 이를 늘릴수 성능이 더 좋아짐.

이 코드에서 5번 반복함.