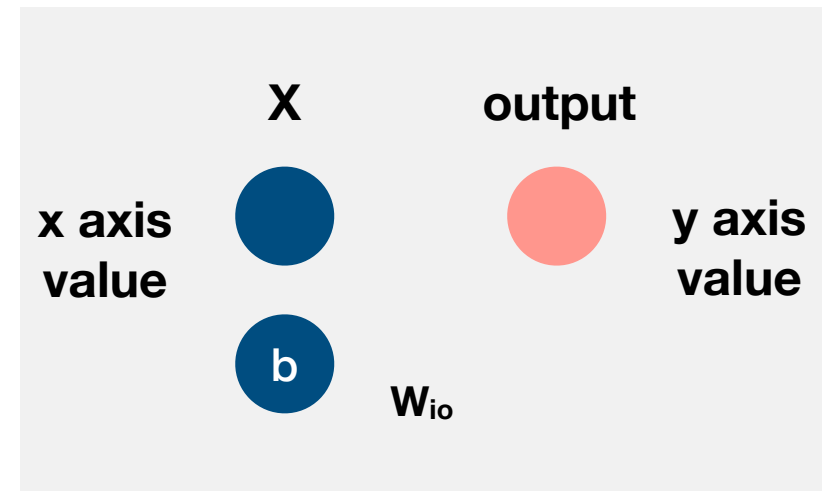
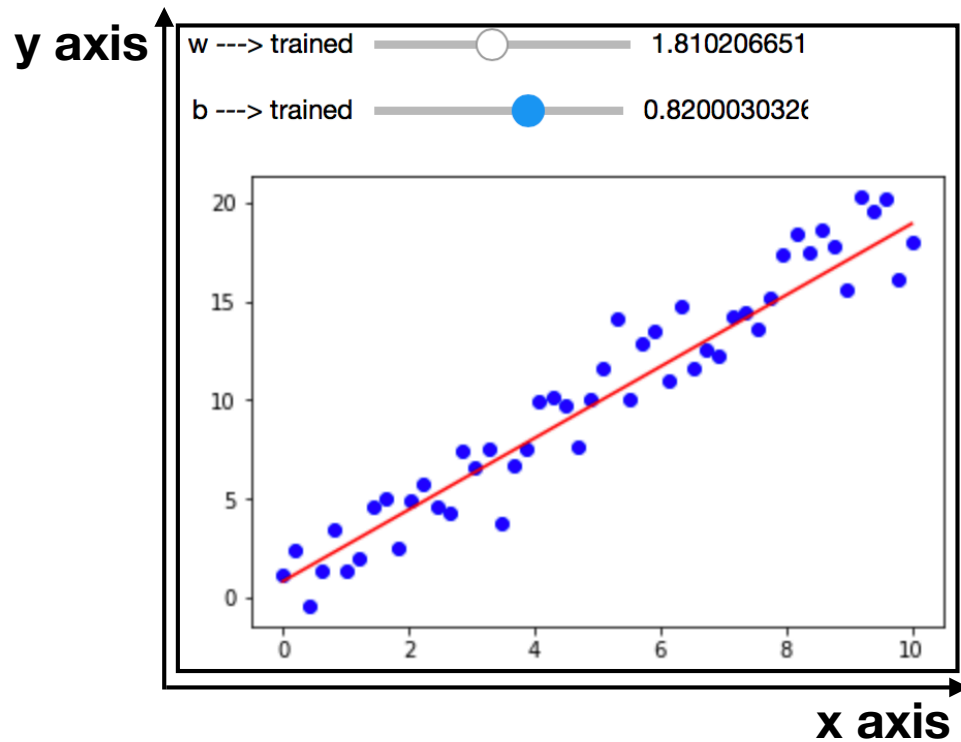


# Neural Net basics

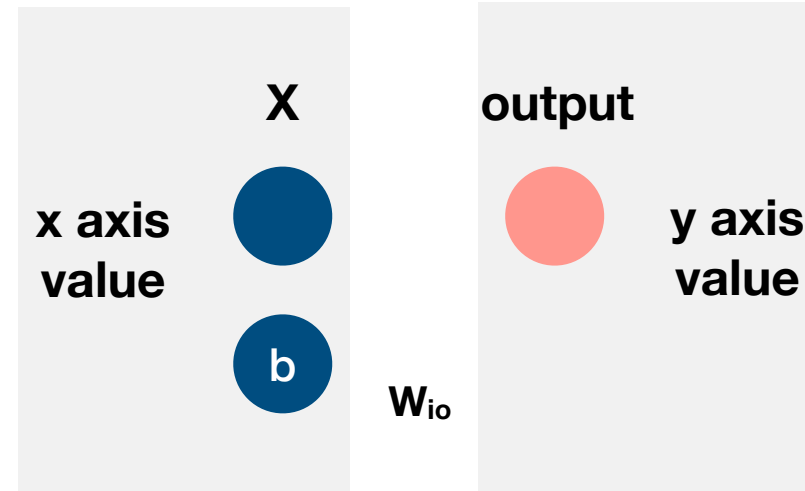
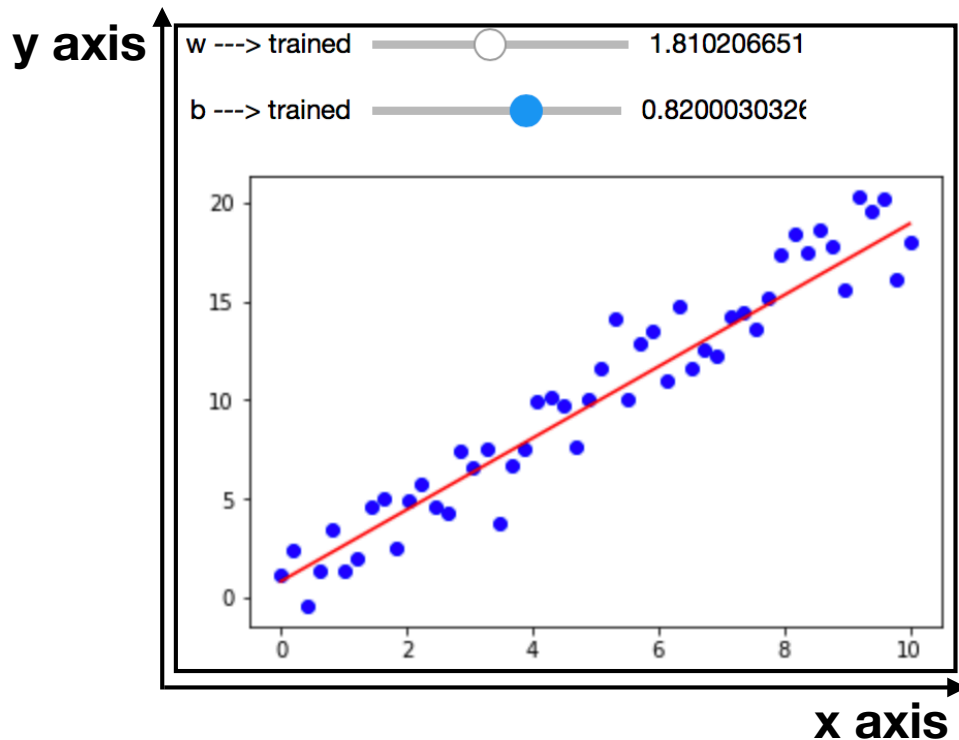
# Least Square Regression

# Least Square Regression



perceptrons를 이용한 least square regression graphical model

# Least Square Regression



least square regression의 예시

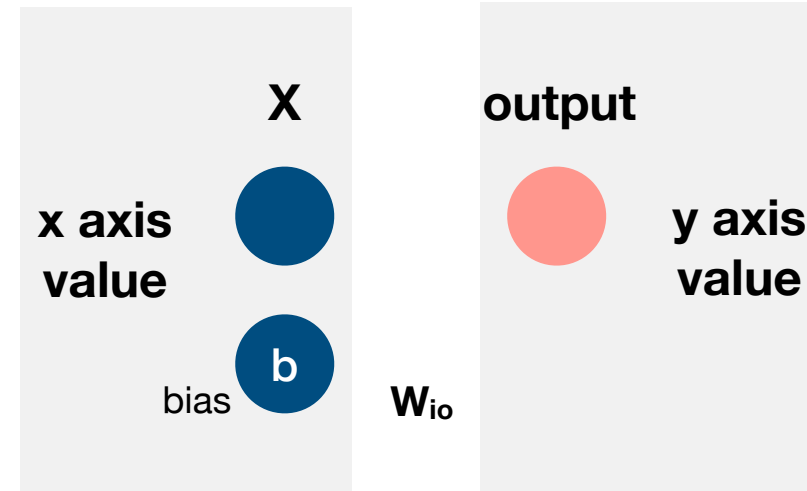
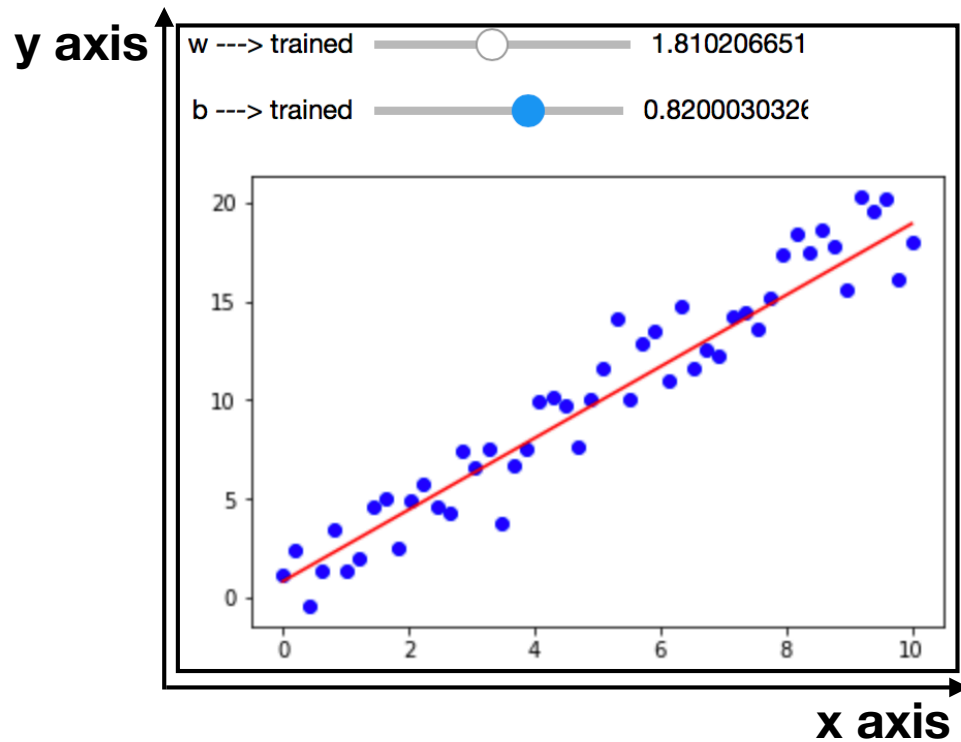
input: 파란 데이터 포인트의 x axis value

target: 파란 데이터 포인트의 y axis value

output: 빨간 선형 모델 상의 y axis value

목적: 파란 데이터를 가장 잘 나타내는 빨간 선형 모델을 그리는 것

# Least Square Regression



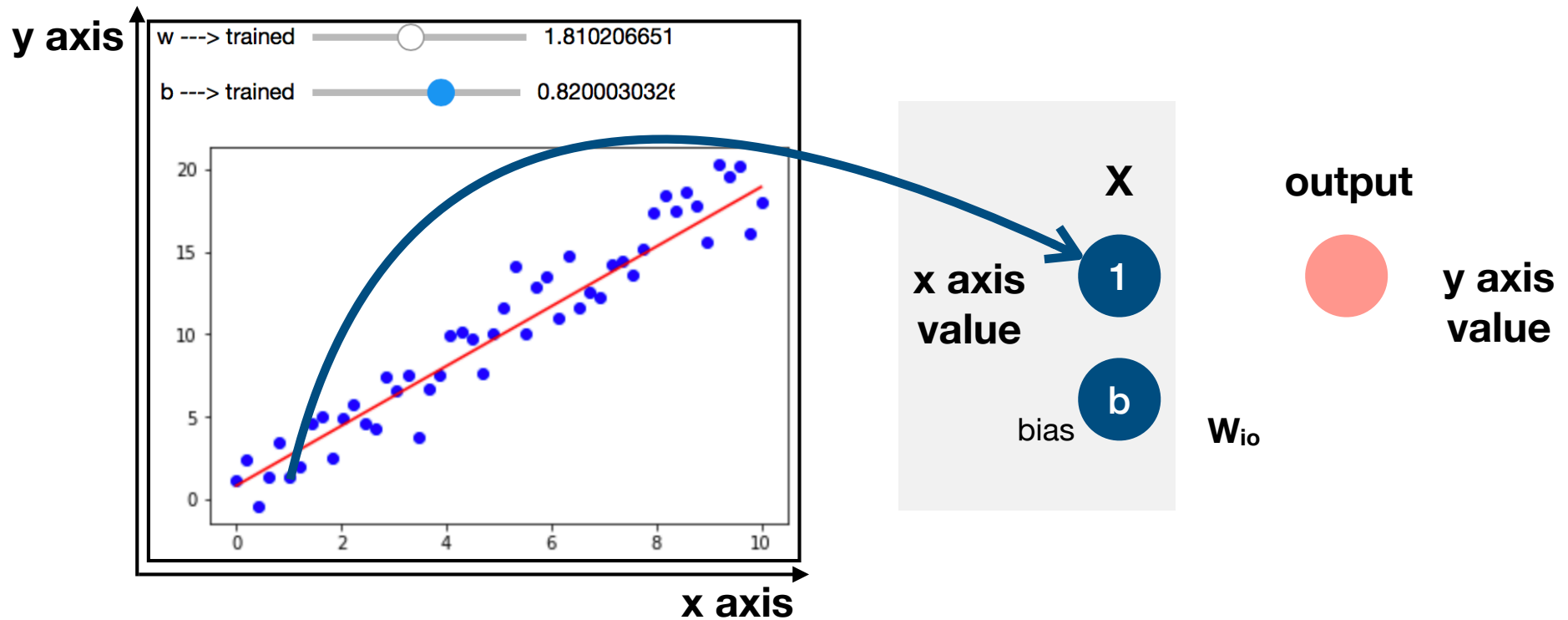
모델구조:

input layer (1 node),

output layer (1 node)

노드 숫자를 셀 때 bias node는 제외

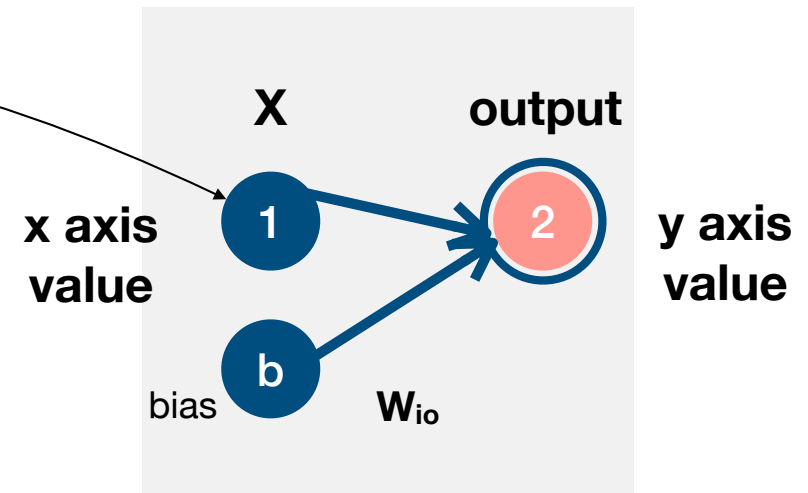
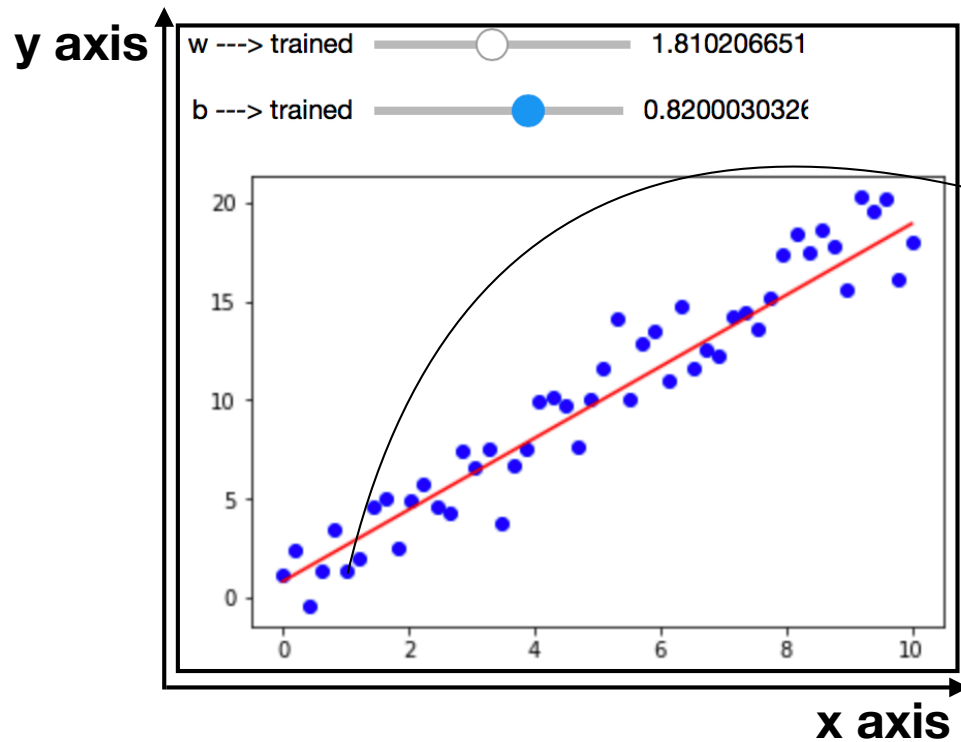
# Least Square Regression



훈련:

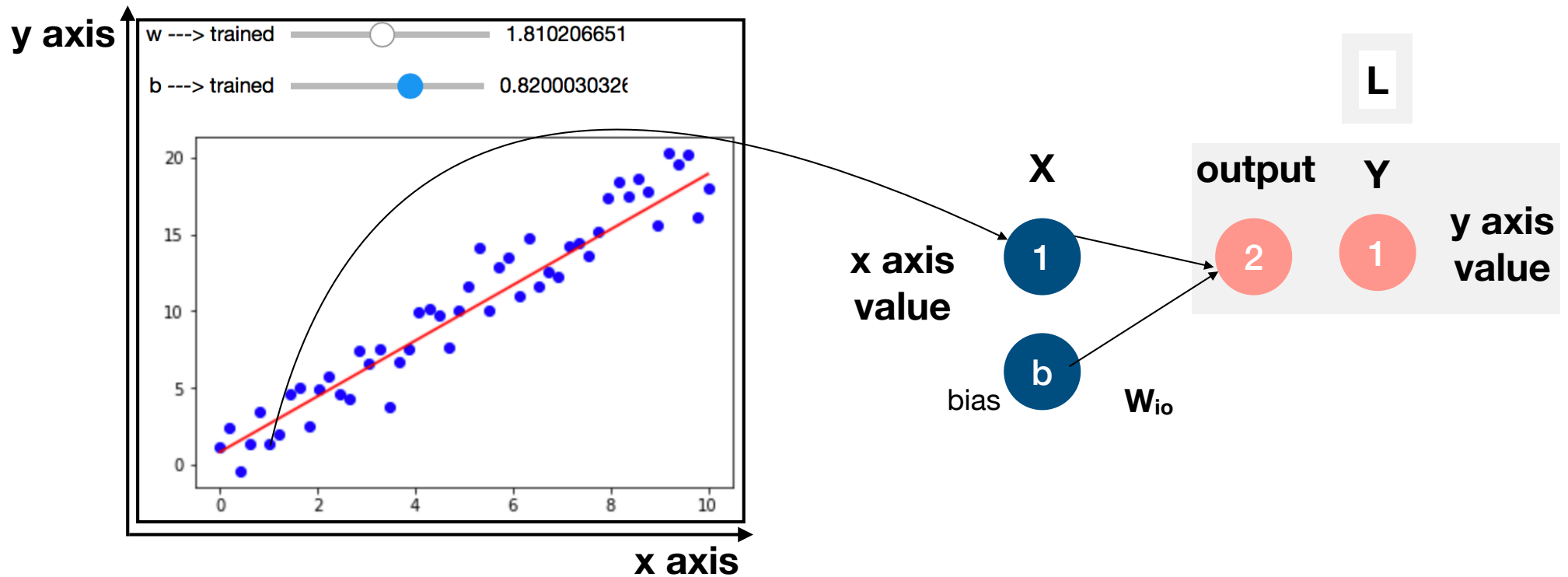
파란 데이터 포인트의 x axis value를 input layer의 node에 넣어줍니다.

# Least Square Regression



각 input layer의 node 값에 weight (진한 화살표)을 곱하고,  
곱한 값을 모두 더해 output layer node에 넣어줍니다.

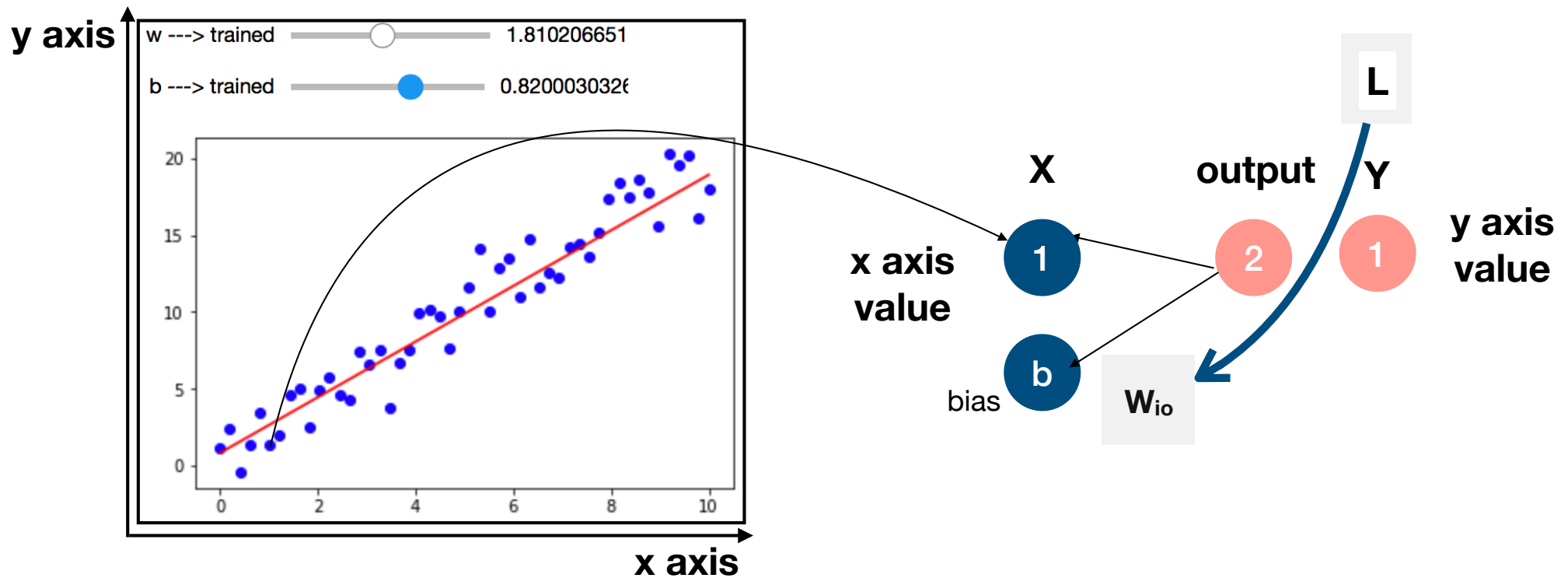
# Least Square Regression



그렇게 구한 output과 정답 target (Y)을 비교하여 loss를 구합니다.



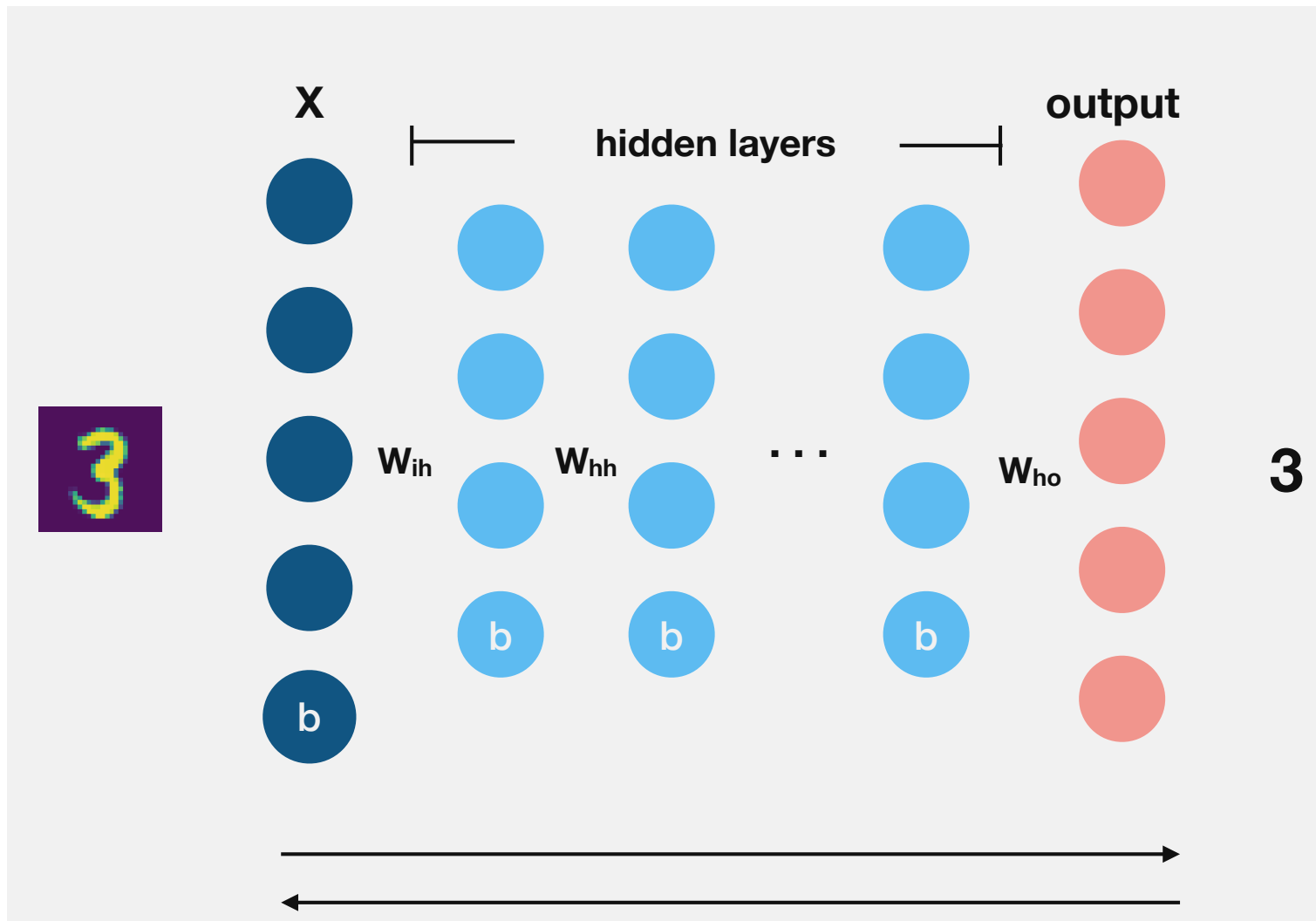
# Least Square Regression



loss를 최소화하는 방향으로, weight을 update 합니다.

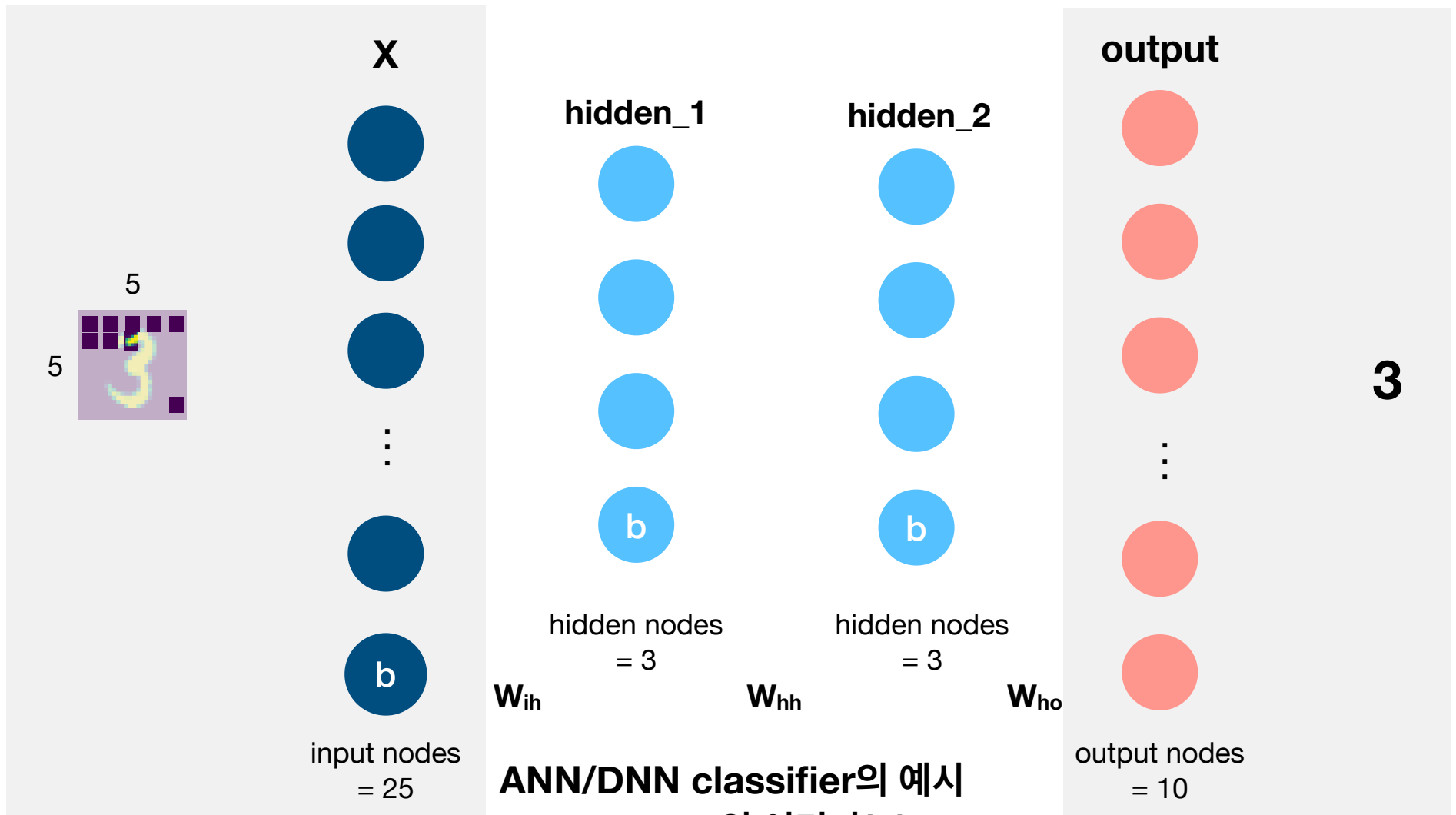
**ANN / DNN**

# ANN DNN



perceptrons를 이용한 ANN/DNN classifier graphical model

# ANN DNN



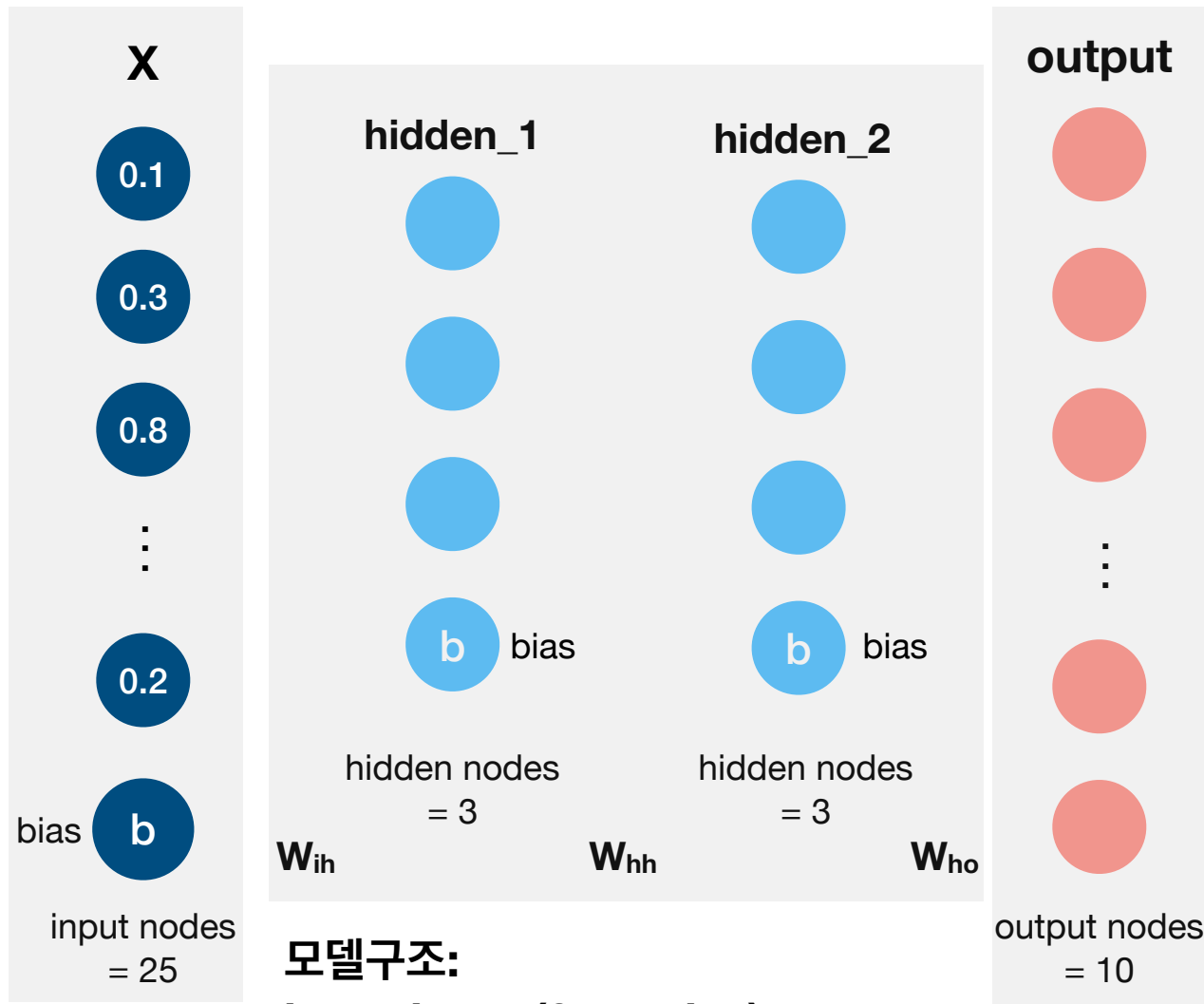
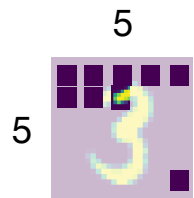
**ANN/DNN classifier의 예시**

**input: 5 by 5의 이미지(X)**

**target: 0에서 9까지의 라벨(Y)**

**목적: input이 0~9 중 어떤 숫자인지 분류**

# ANN DNN



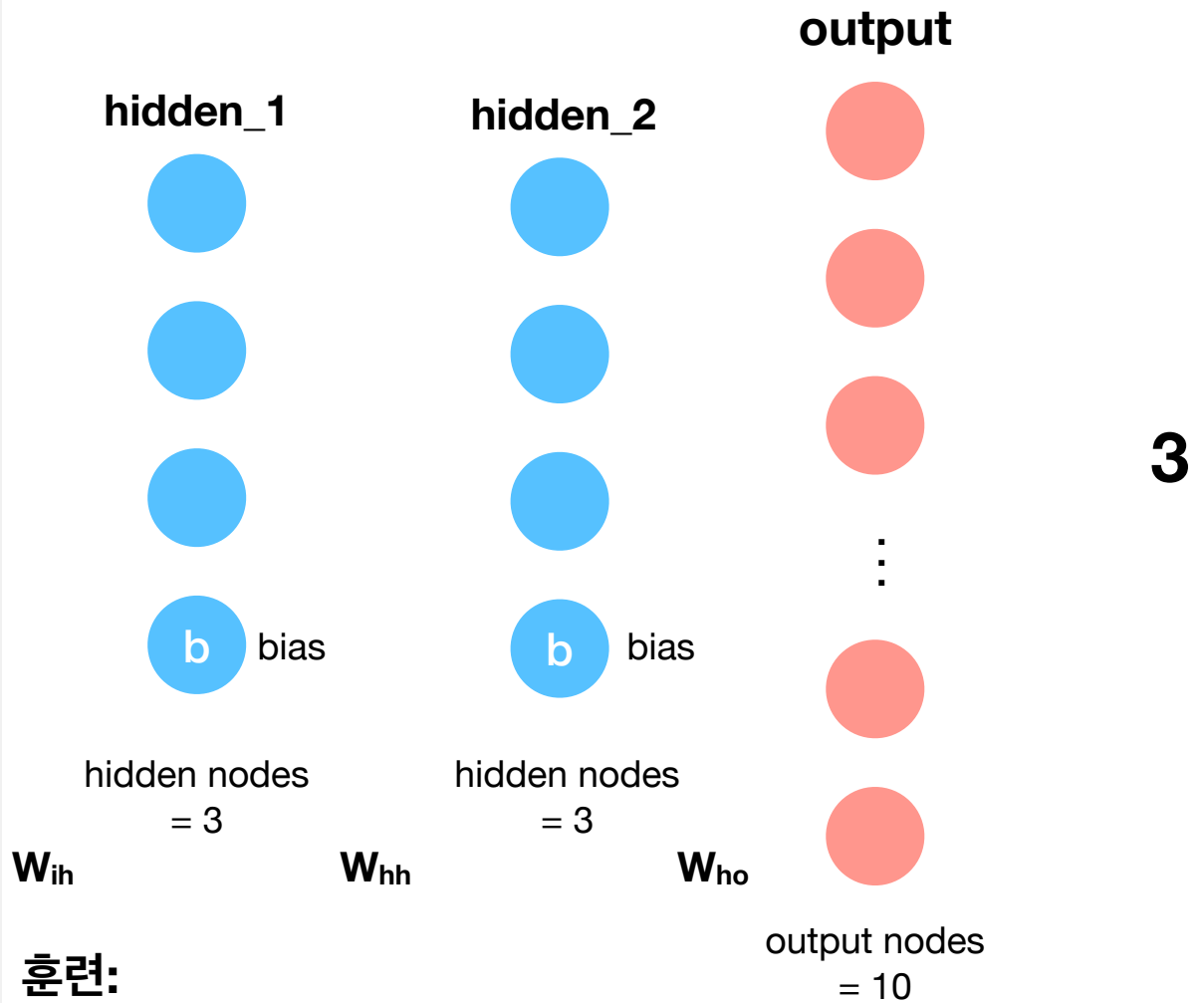
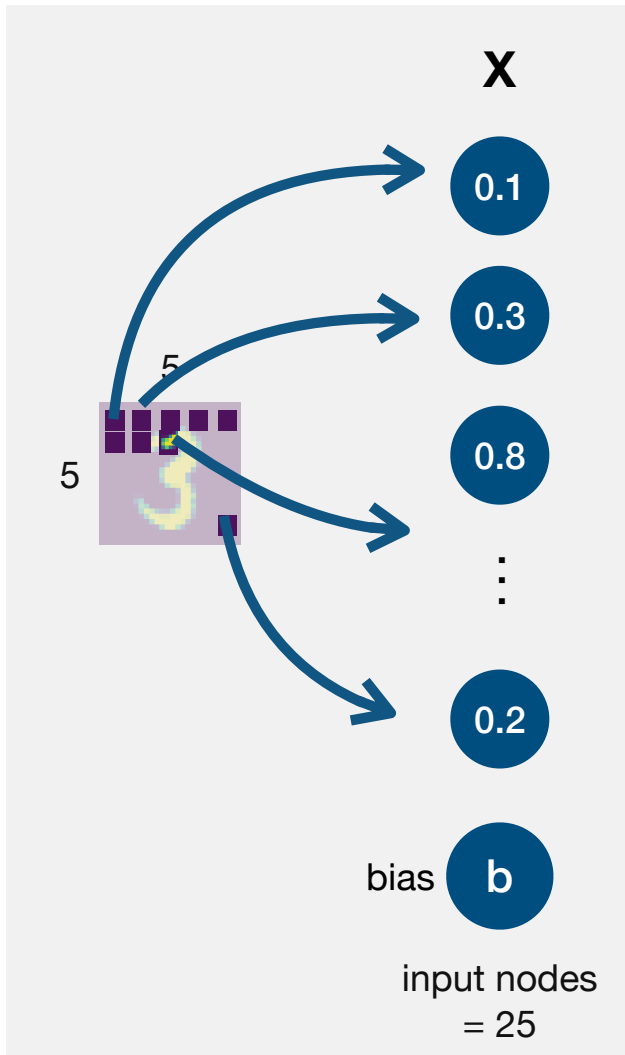
**3**

**모델구조:**

**input layer (25 nodes),  
2 hidden layers (3 nodes each),  
output layer (10 nodes)**

노드 숫자를 셀 때 bias node는 제외

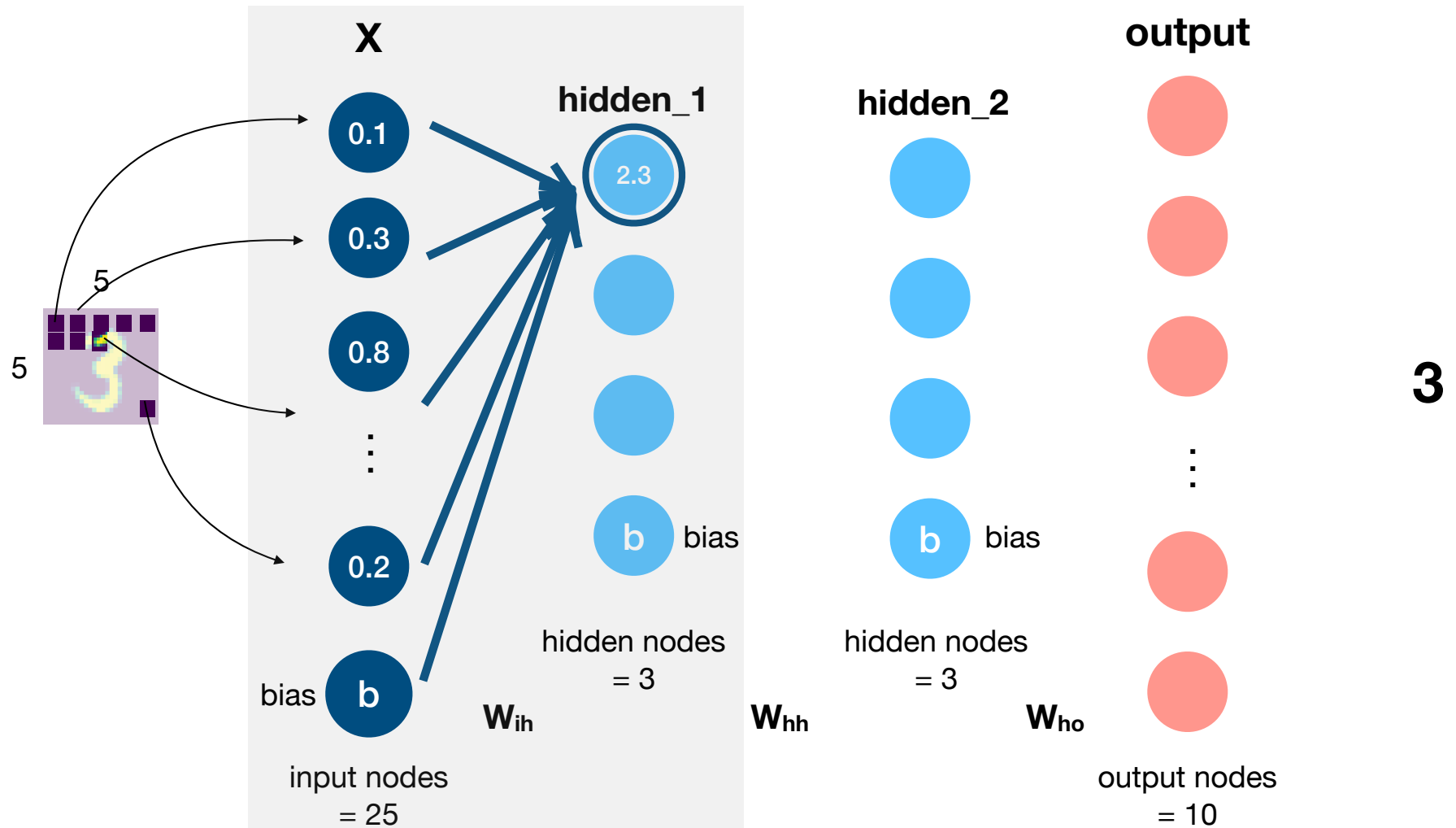
# ANN DNN



**훈련:**

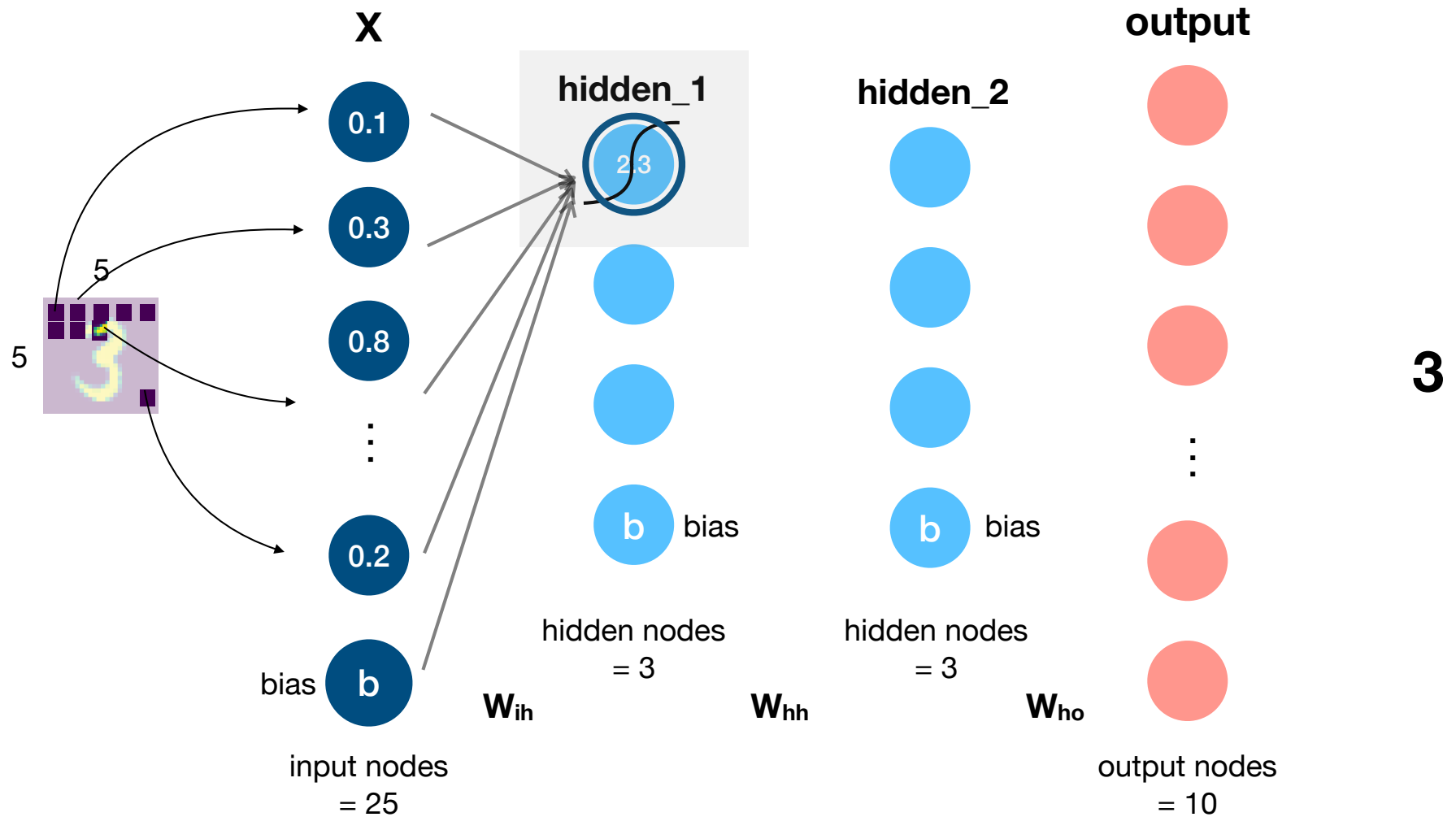
이미지를 pixel 단위로 쪼개서  
그 값을 각각 input layer의 nodes에 넣어줍니다.

# ANN DNN



각 input layer의 node 값에 weight (진한 화살표)을 곱하고,  
곱한 값을 모두 더해 첫 번째 hidden layer node에 넣어줍니다.

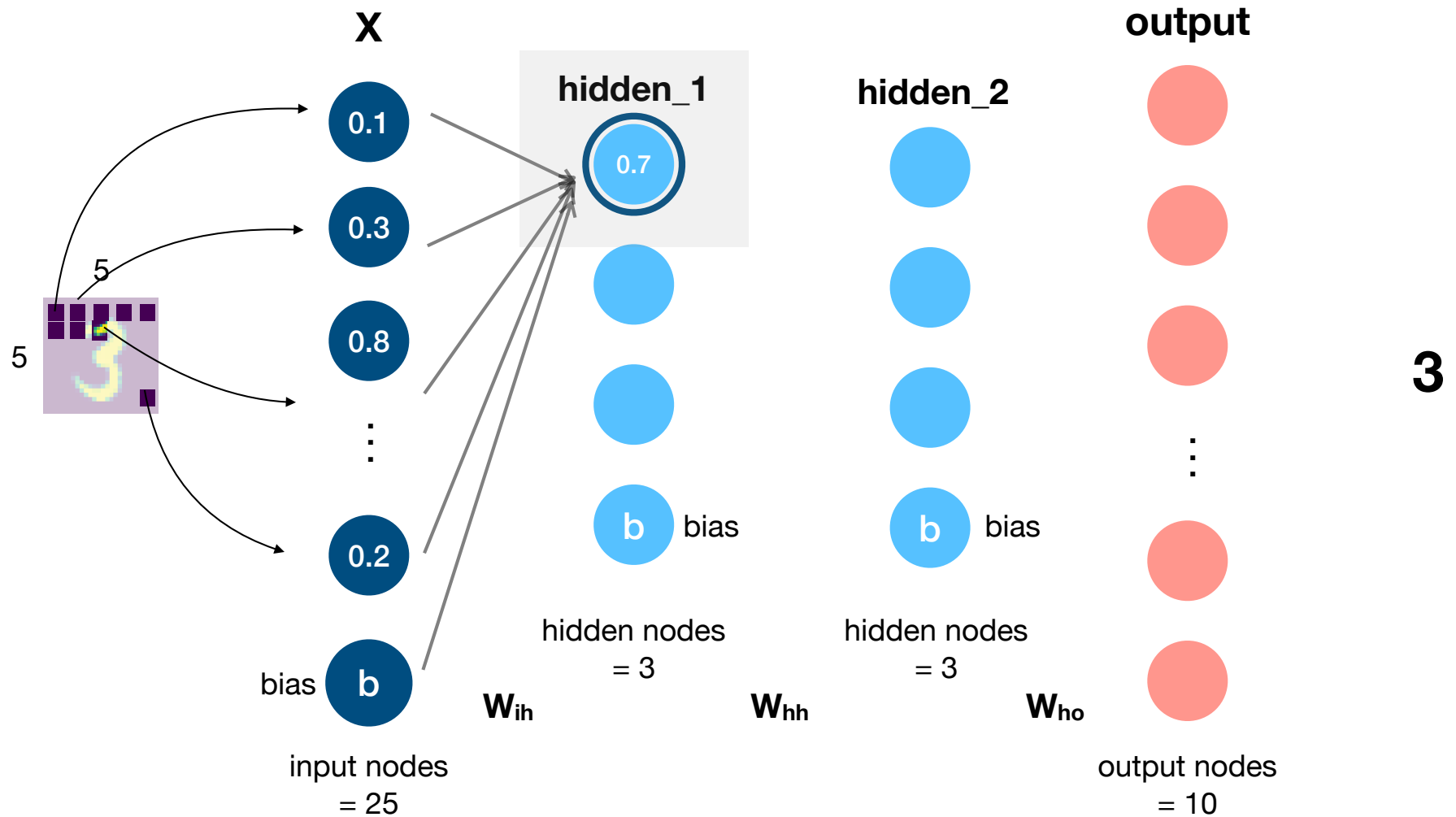
# ANN DNN



activation function을 통해서 node 값을  
0과 1 사이의 값(sigmoid function)으로 바꿔줍니다.

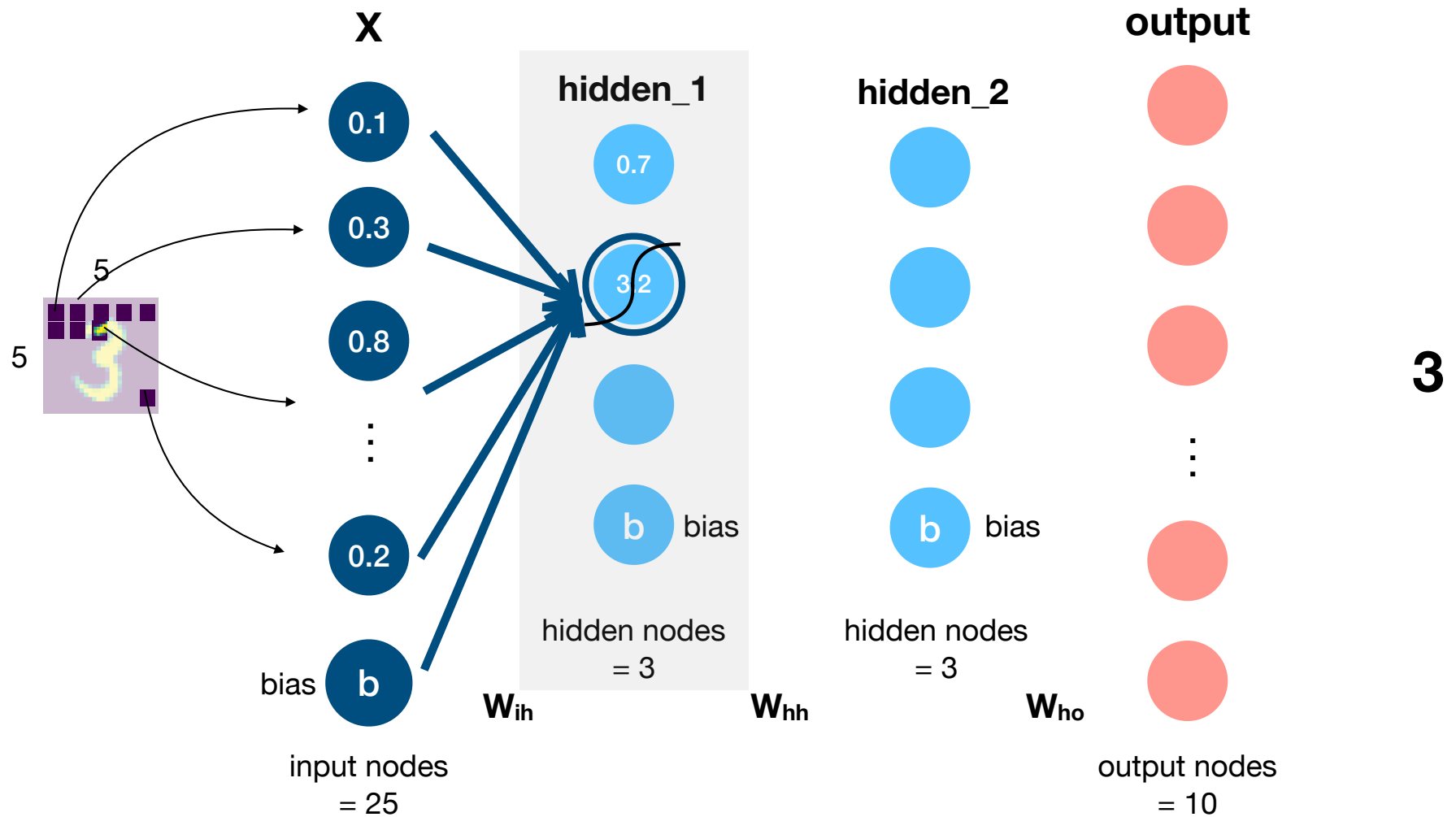


# ANN DNN



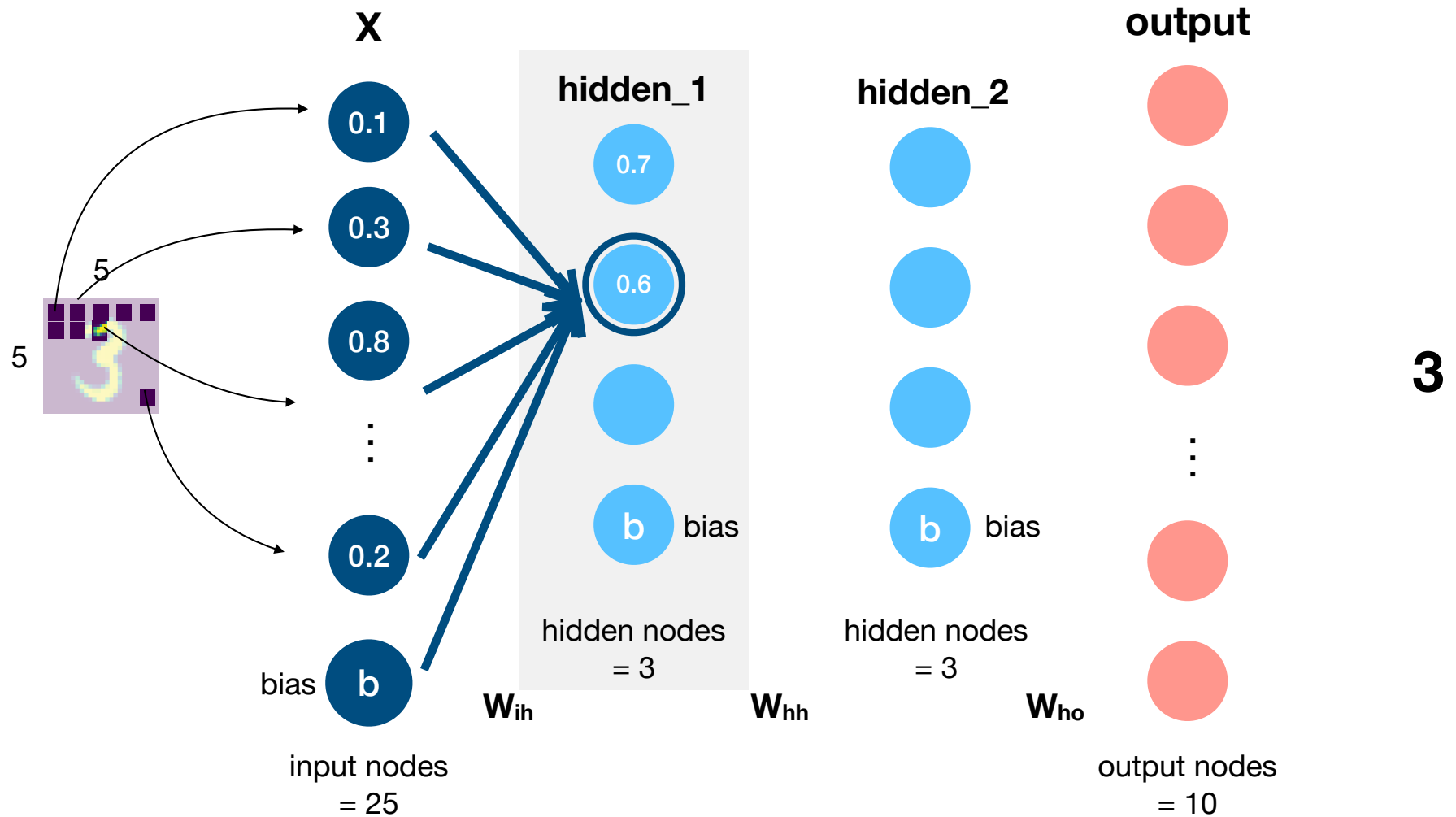
activation function을 통해서 node 값을  
0과 1 사이의 값(sigmoid function)으로 바꿔줍니다.

# ANN DNN



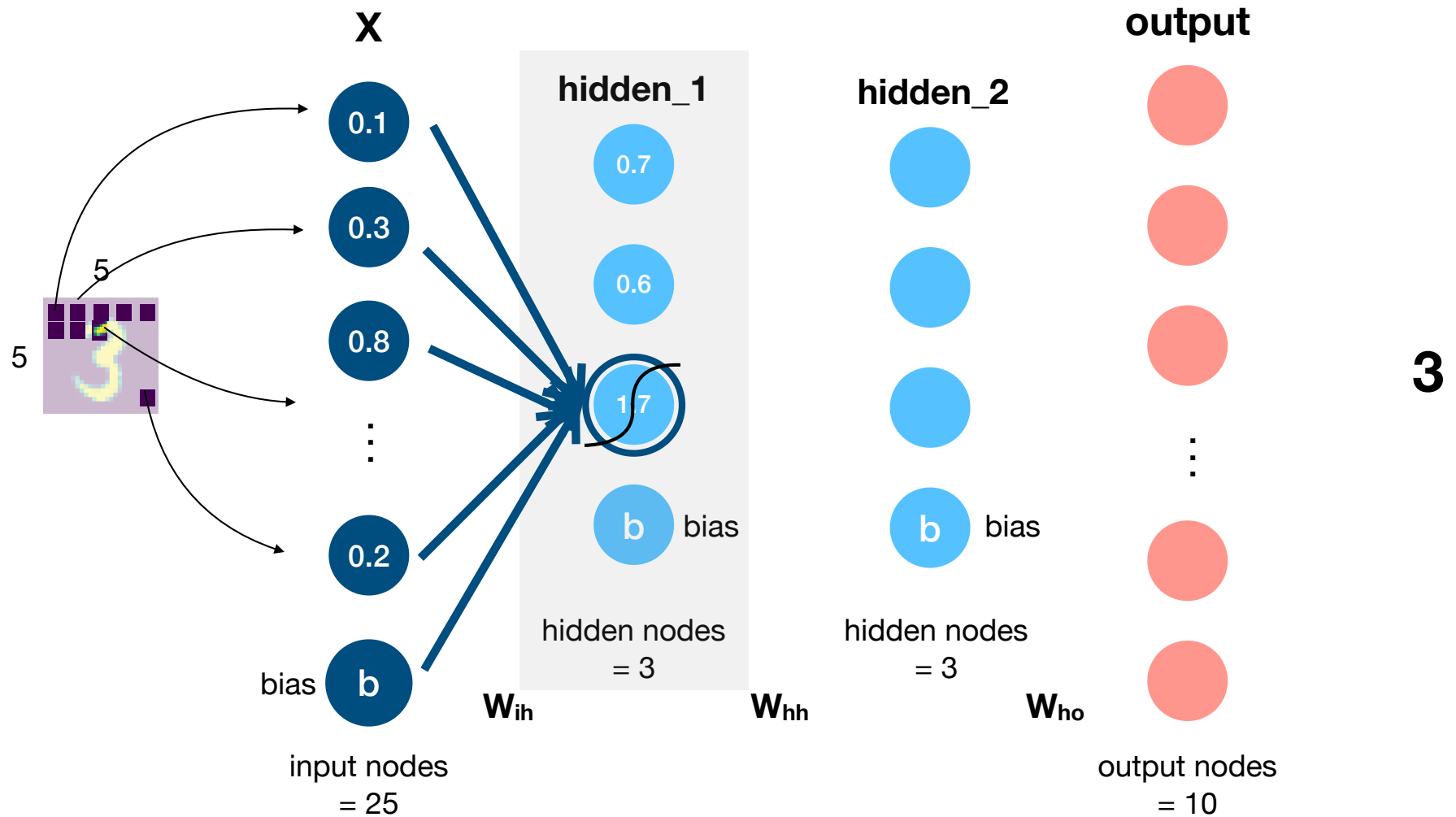
모든 hidden layer 1의 nodes에 대해 반복합니다.

# ANN DNN



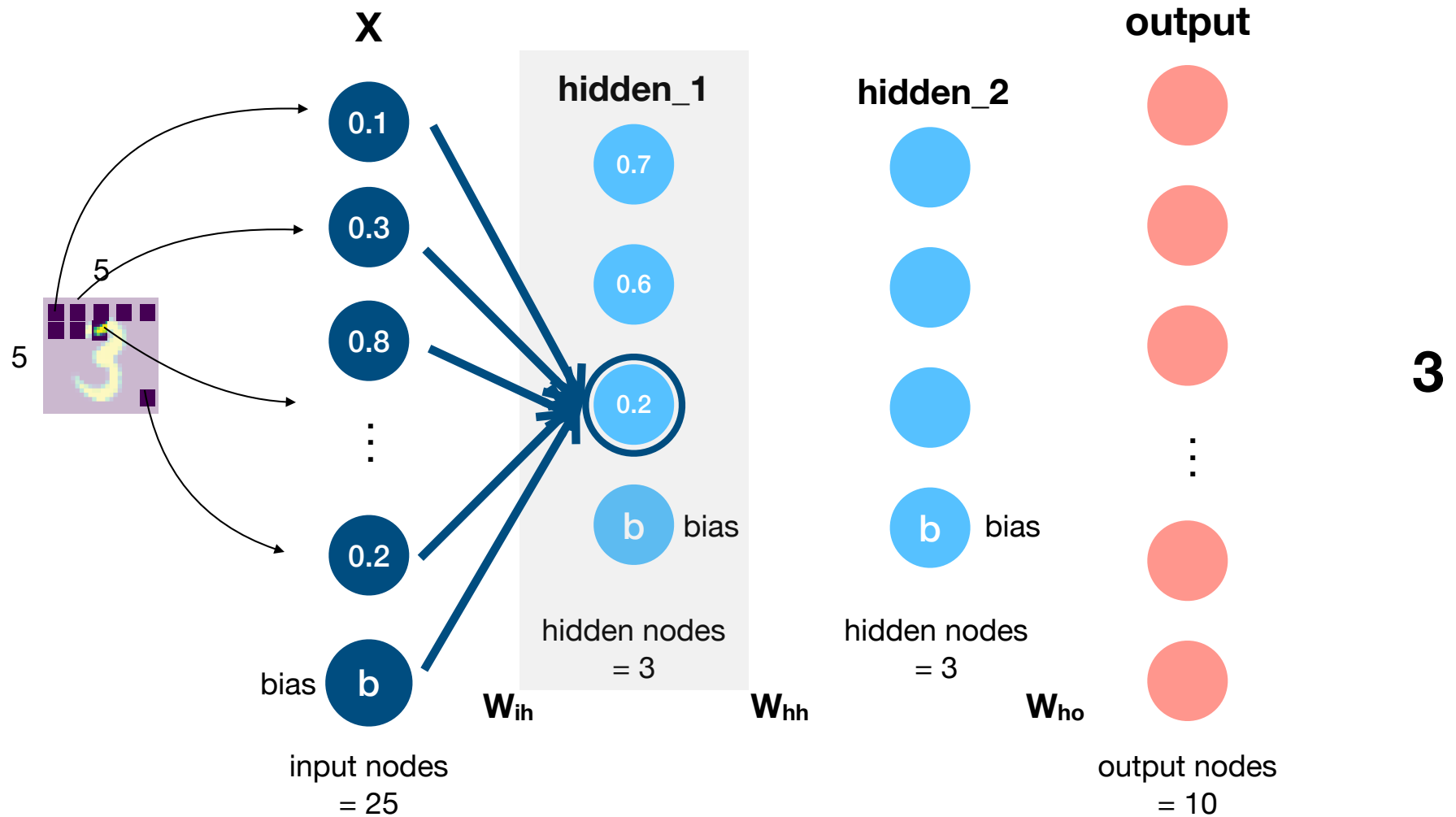
모든 hidden layer 1의 nodes에 대해 반복합니다.

# ANN DNN



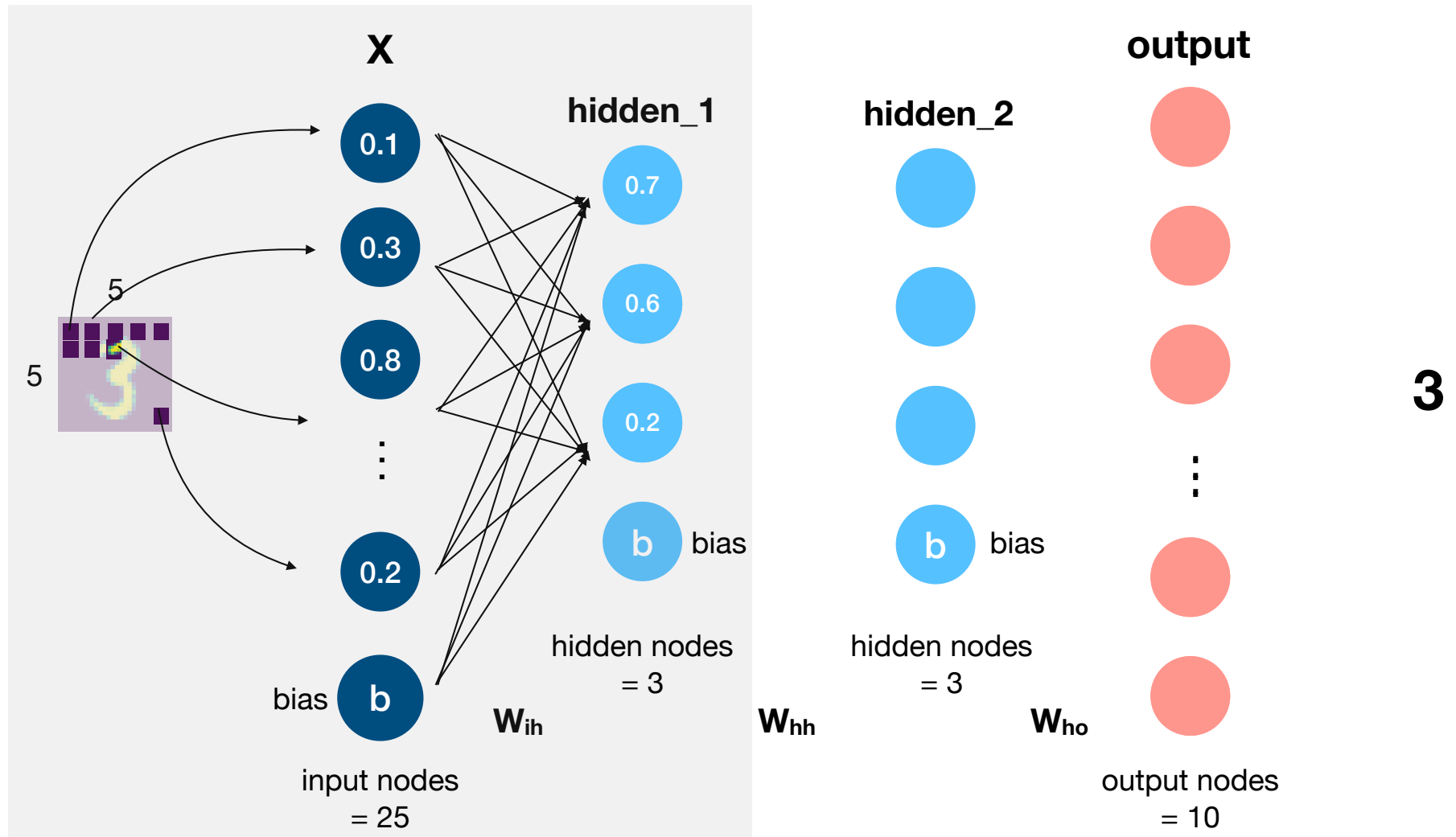
모든 hidden layer 1의 nodes에 대해 반복합니다.

# ANN DNN



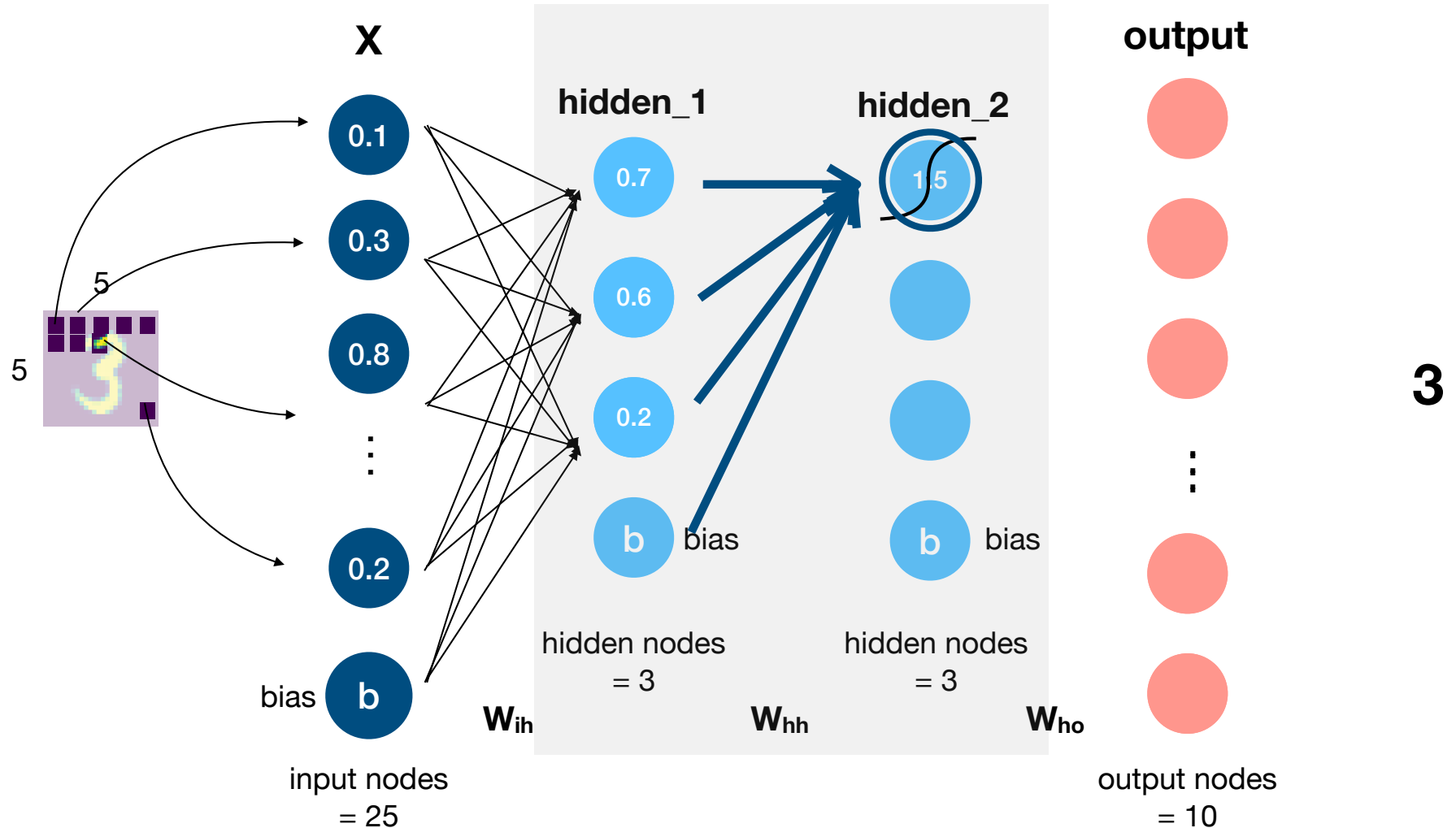
모든 hidden layer 1의 nodes에 대해 반복합니다.

# ANN DNN



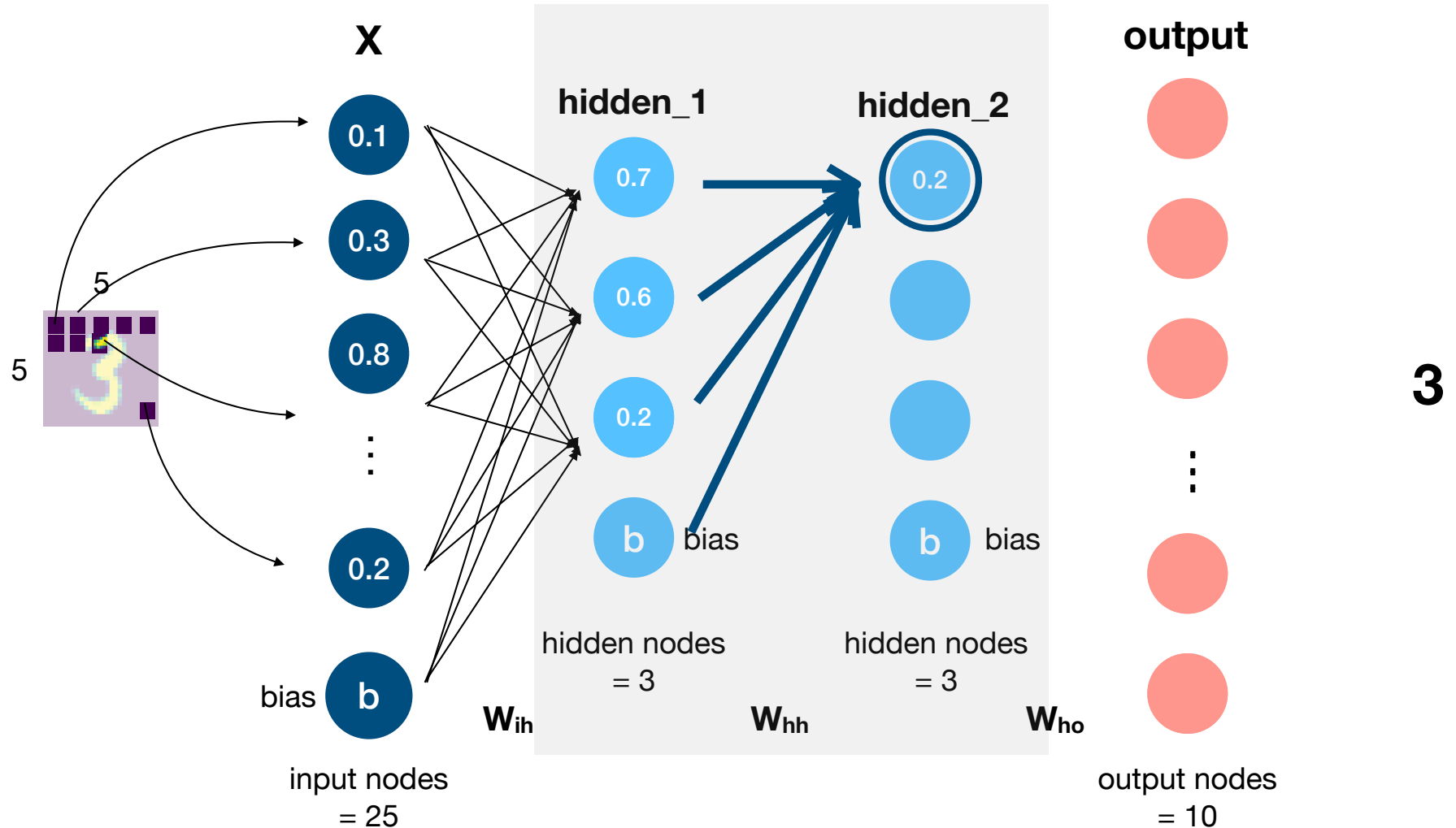
hidden layer 1까지의 nodes가 모두 계산되었습니다.  
이제 hidden layer 2로 넘어갈 차례입니다.

# ANN DNN



전과 같이, 각 hidden layer 1의 node 값에 weight (진한 화살표)을 곱하고 모두 더해 activation function을 건 후 hidden layer 2의 첫 번째 node에 넣어줍니다.

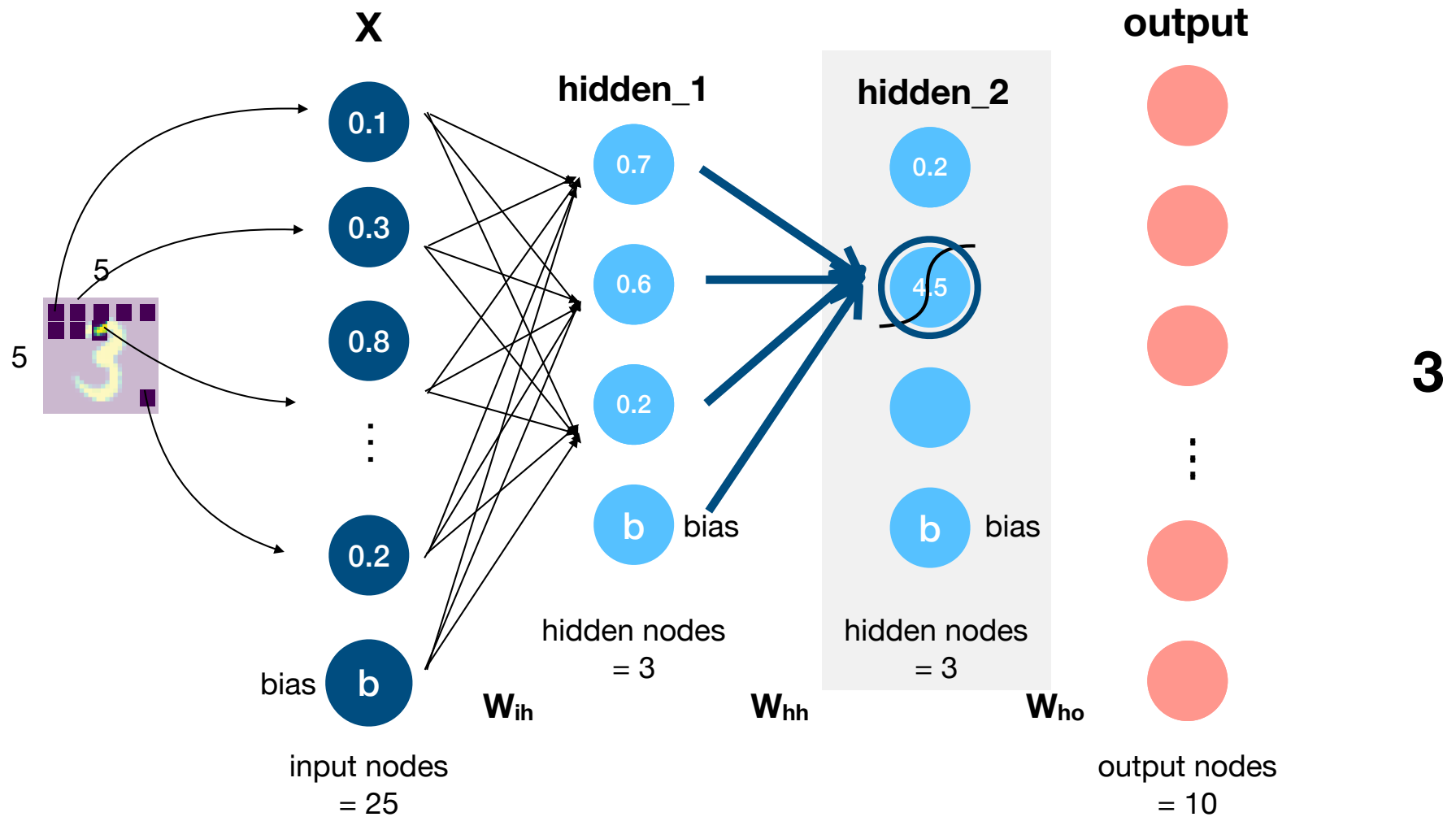
# ANN DNN



전과 같이, 각 hidden layer 1의 node 값에 weight (진한 화살표)을 곱하고 모두 더해 activation function을 건 후 hidden layer 2의 첫 번째 node에 넣어줍니다.

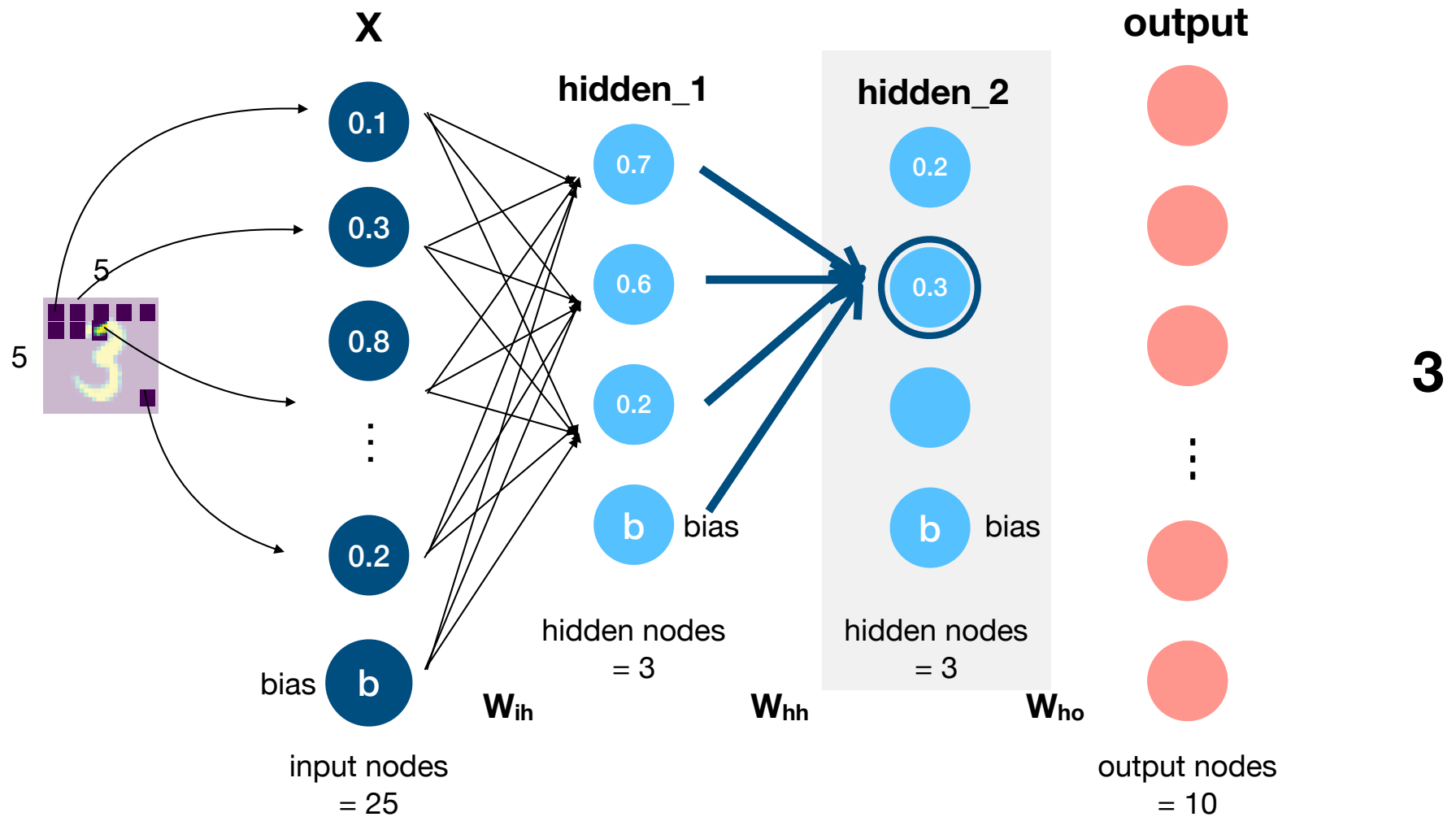


# ANN DNN



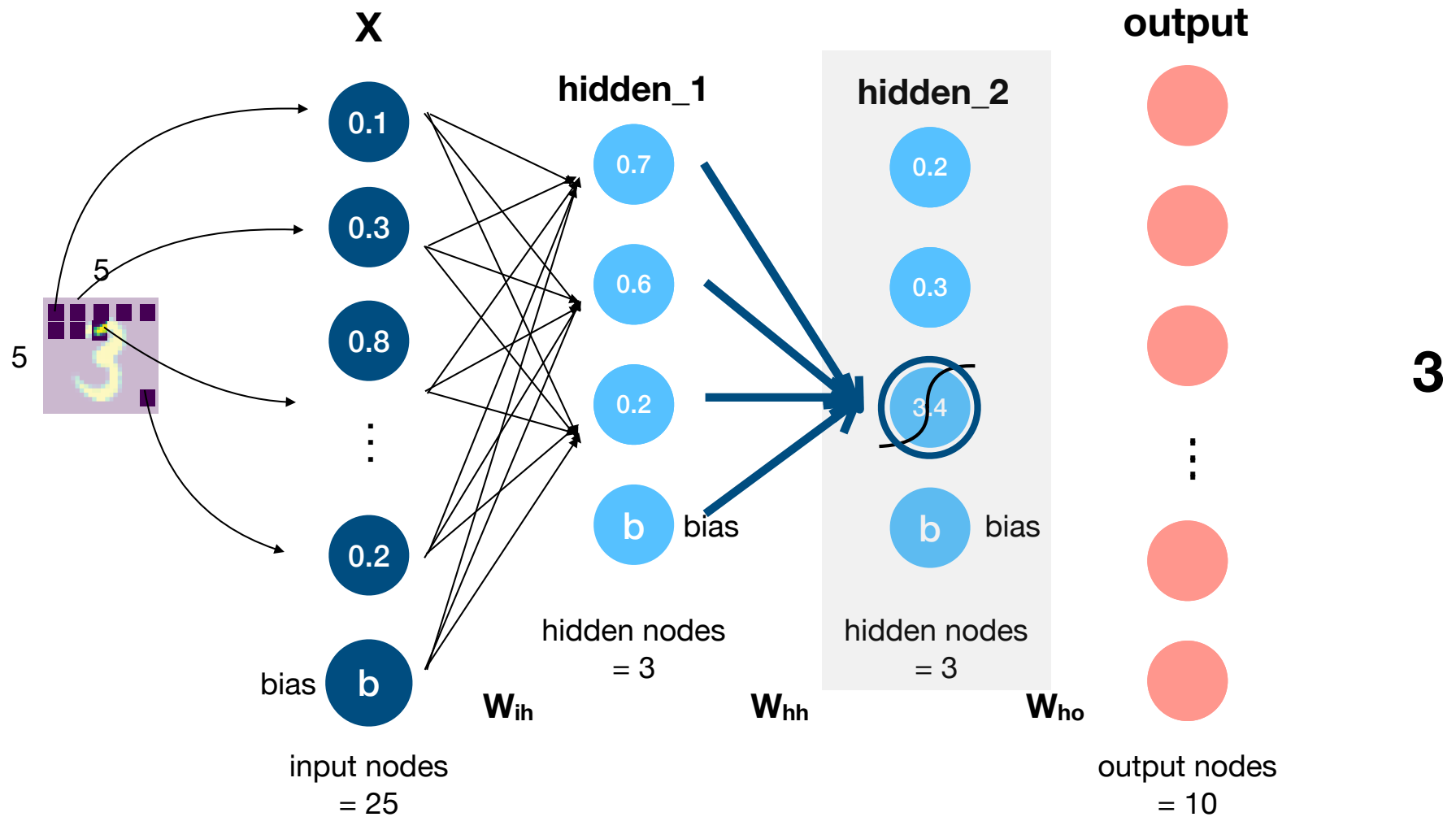
모든 hidden layer 2의 nodes에 대해 반복합니다.

# ANN DNN



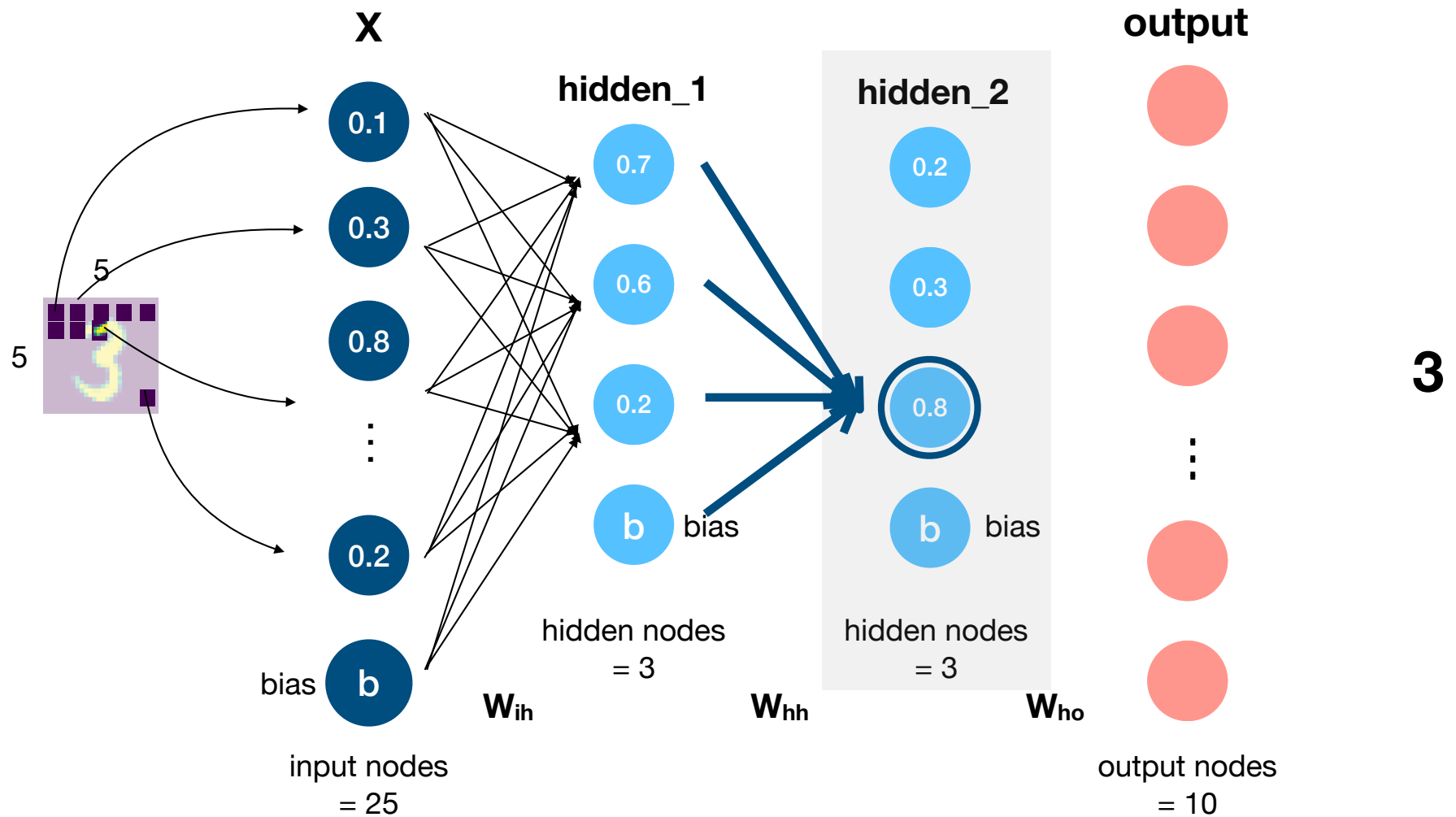
모든 hidden layer 2의 nodes에 대해 반복합니다.

# ANN DNN



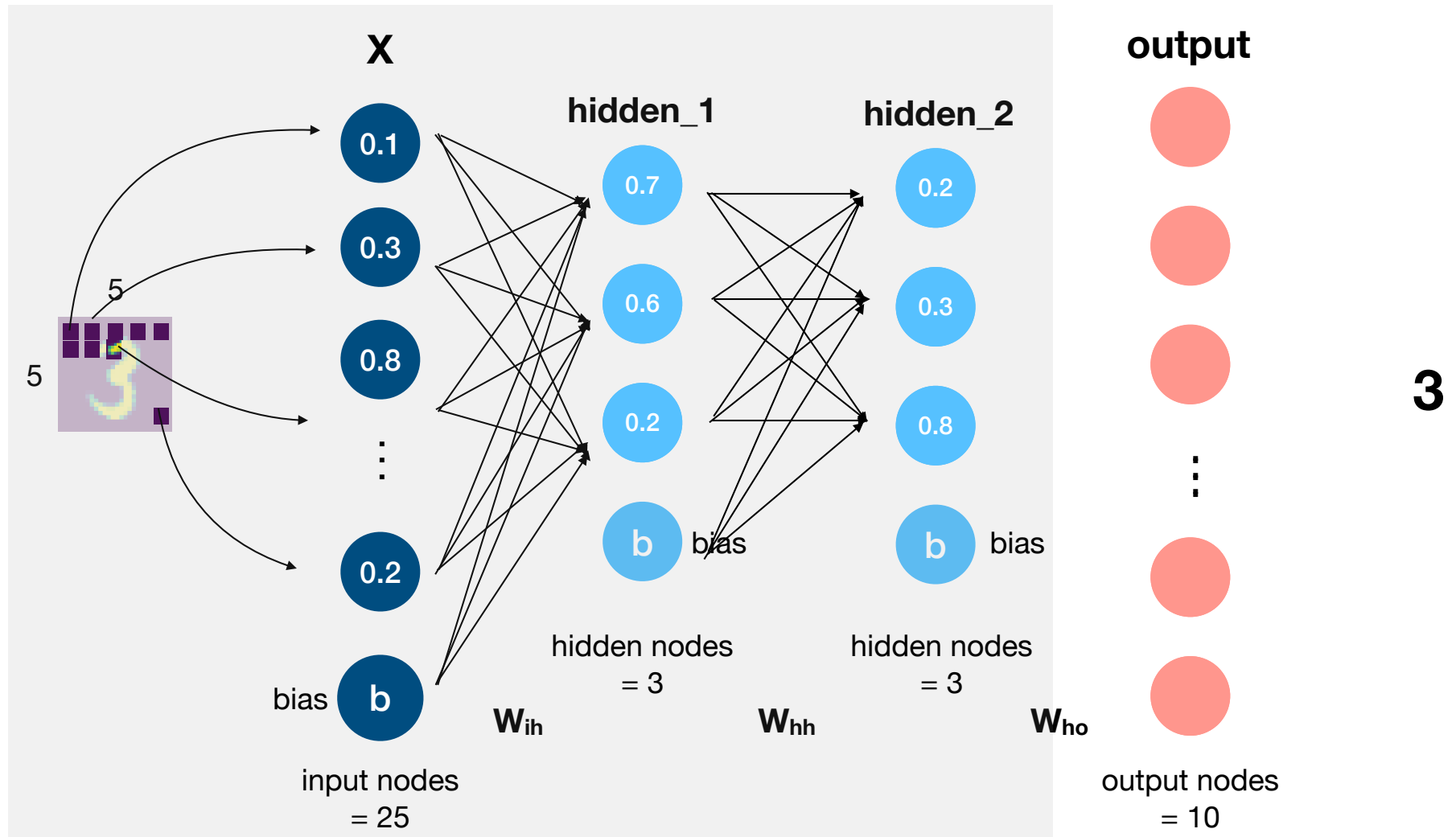
모든 hidden layer 2의 nodes에 대해 반복합니다.

# ANN DNN



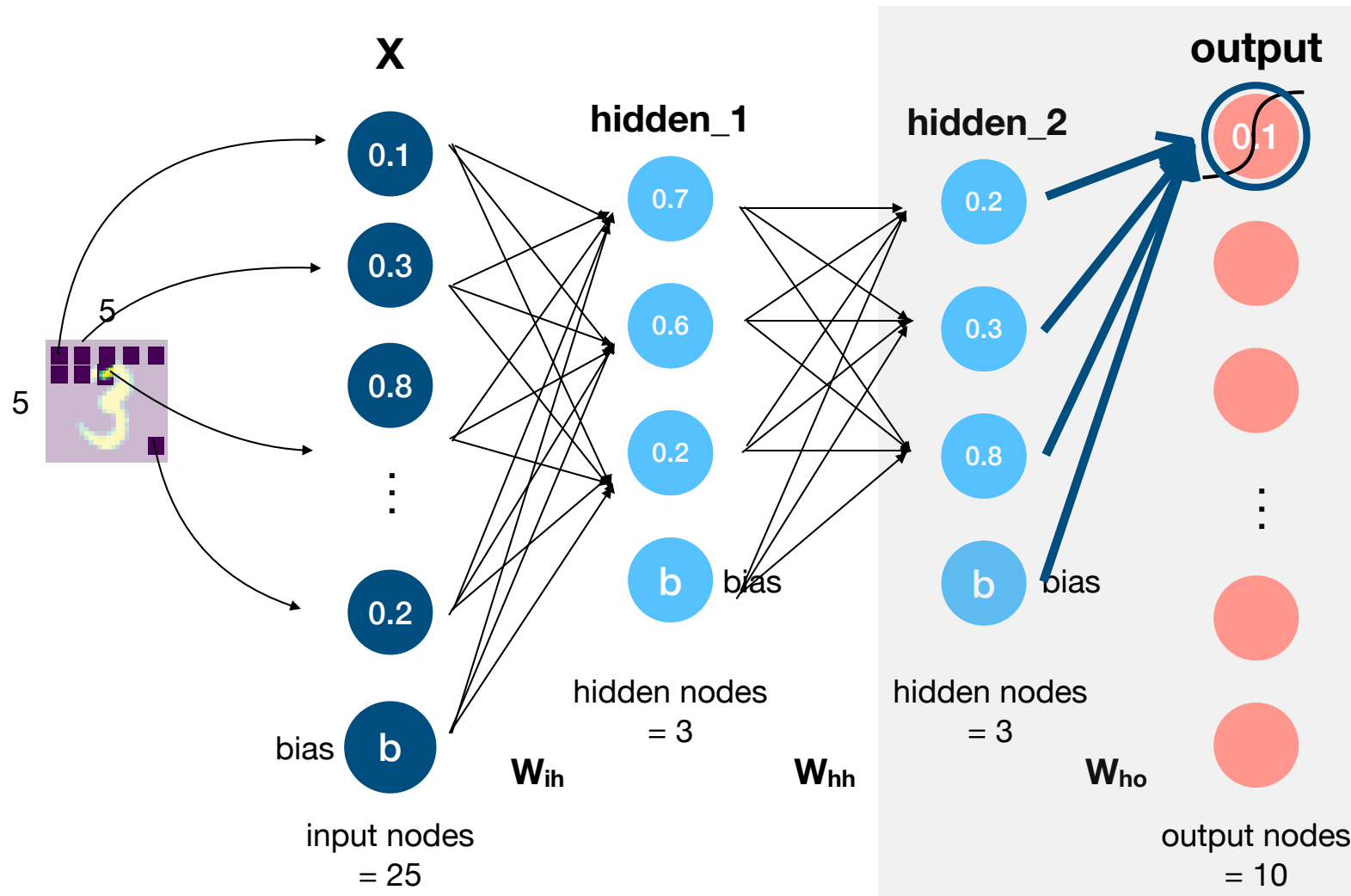
모든 hidden layer 2의 nodes에 대해 반복합니다.

# ANN DNN



hidden layer 2까지의 nodes가 모두 계산되었습니다.  
이제 output layer로 넘어갈 차례입니다.

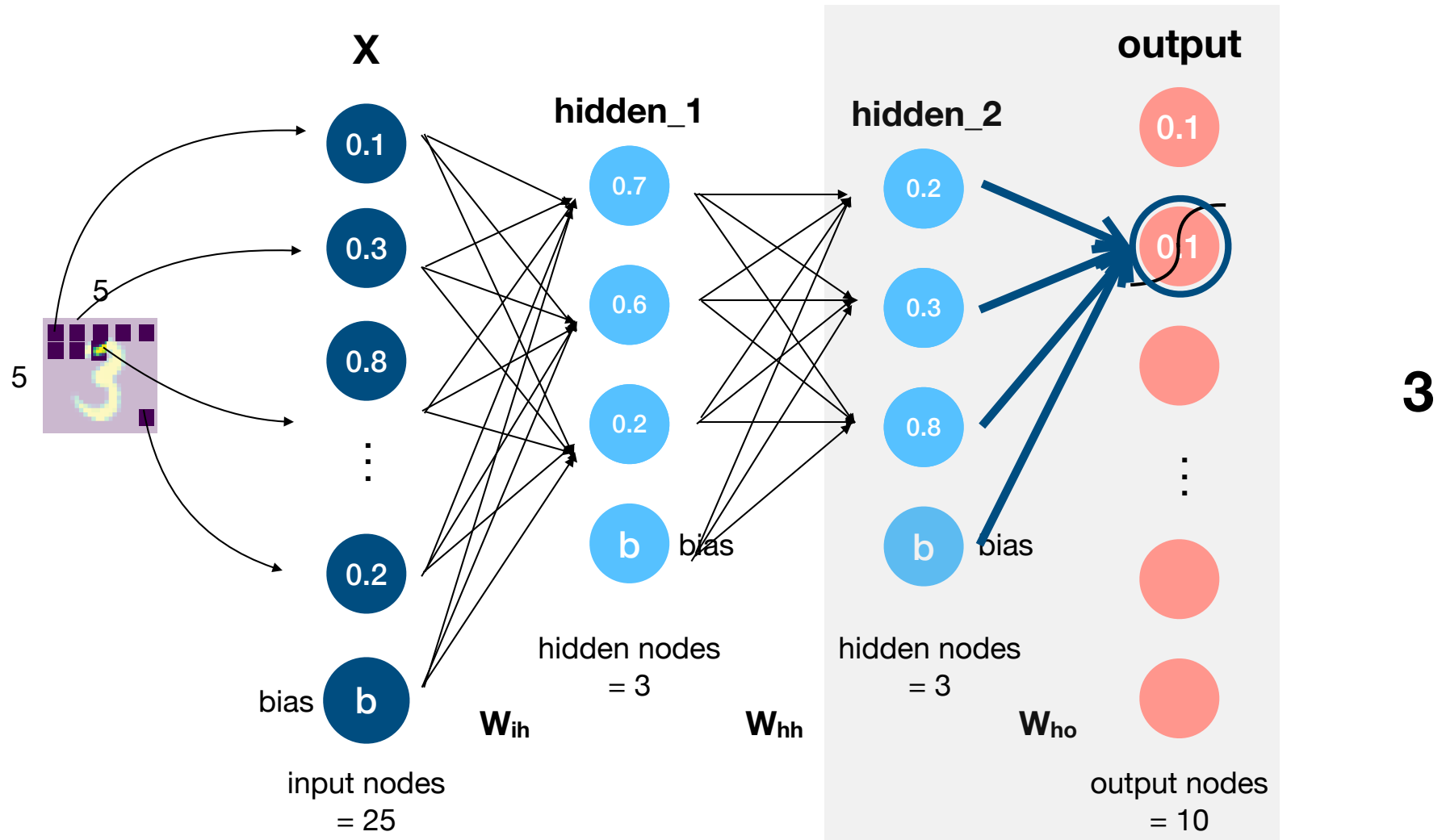
# ANN DNN



3

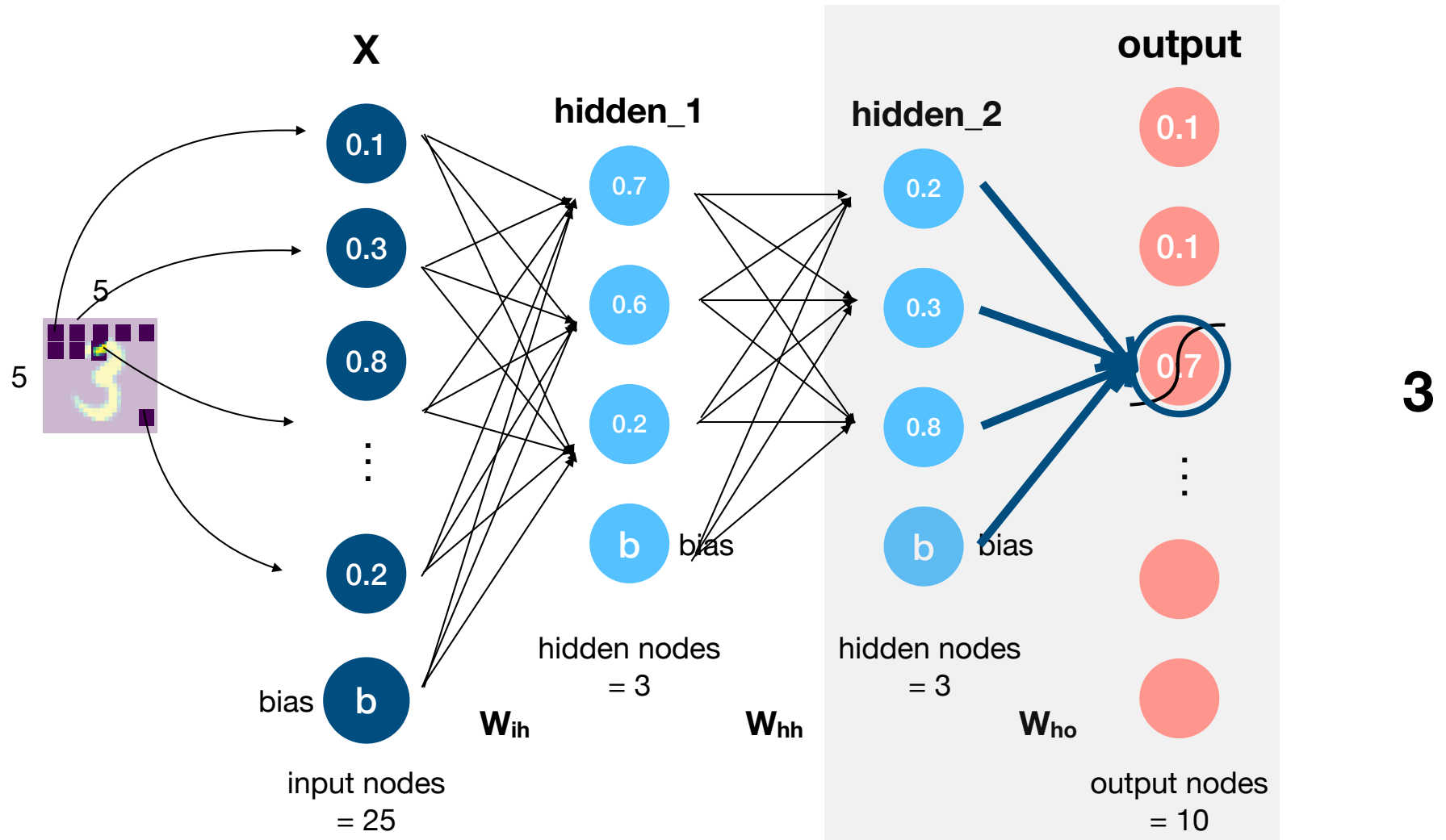
전과 같이, 각 hidden layer 2의 node 값에 weight (진한 화살표)을 곱하고 모두 더해 activation function을 건 후 output layer의 첫 번째 node에 넣어줍니다.

# ANN DNN



마찬가지 방식으로 반복합니다.

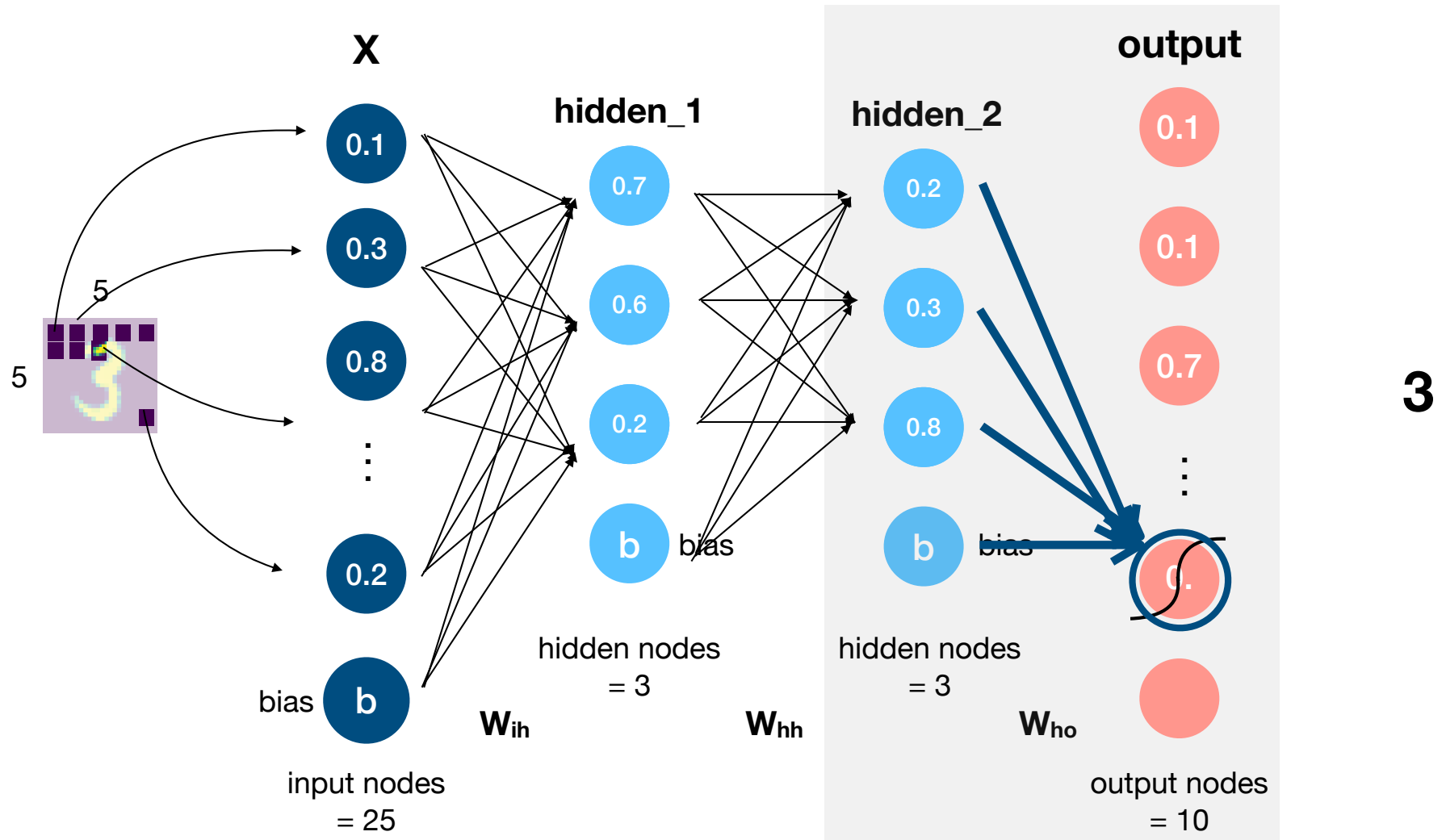
# ANN DNN



마찬가지 방식으로 반복합니다.

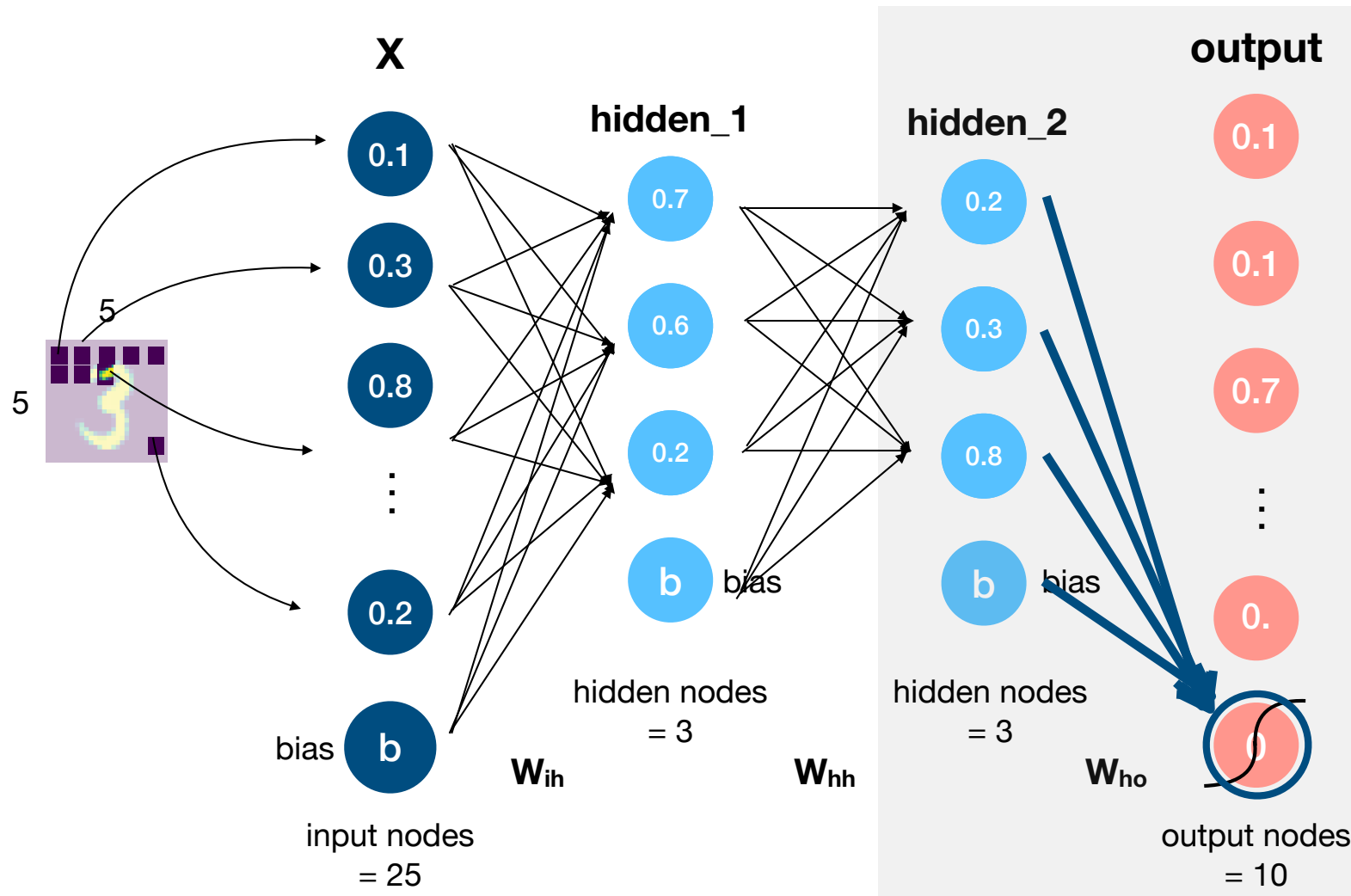


# ANN DNN



마찬가지 방식으로 반복합니다.

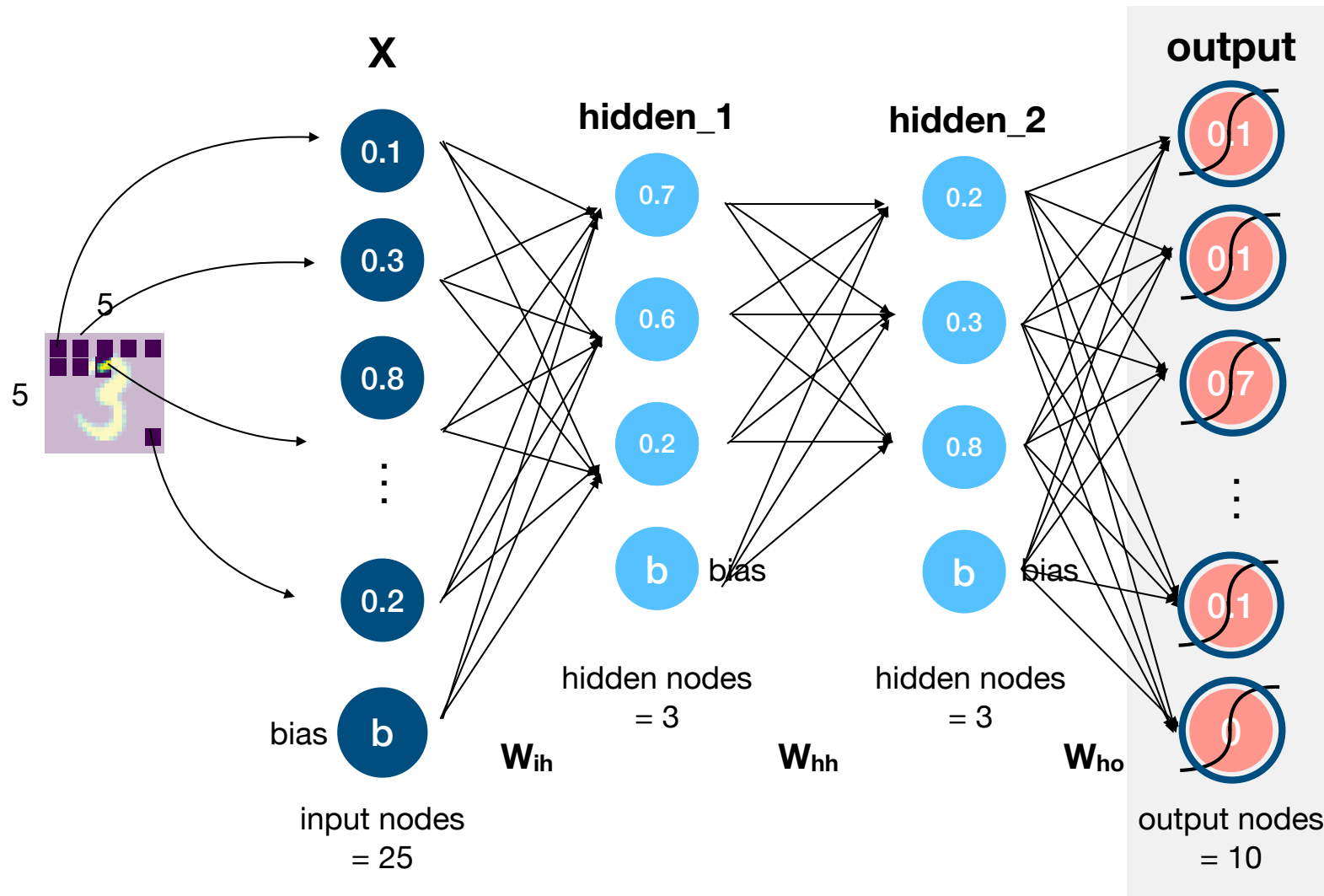
# ANN DNN



3

마찬가지 방식으로 반복합니다.

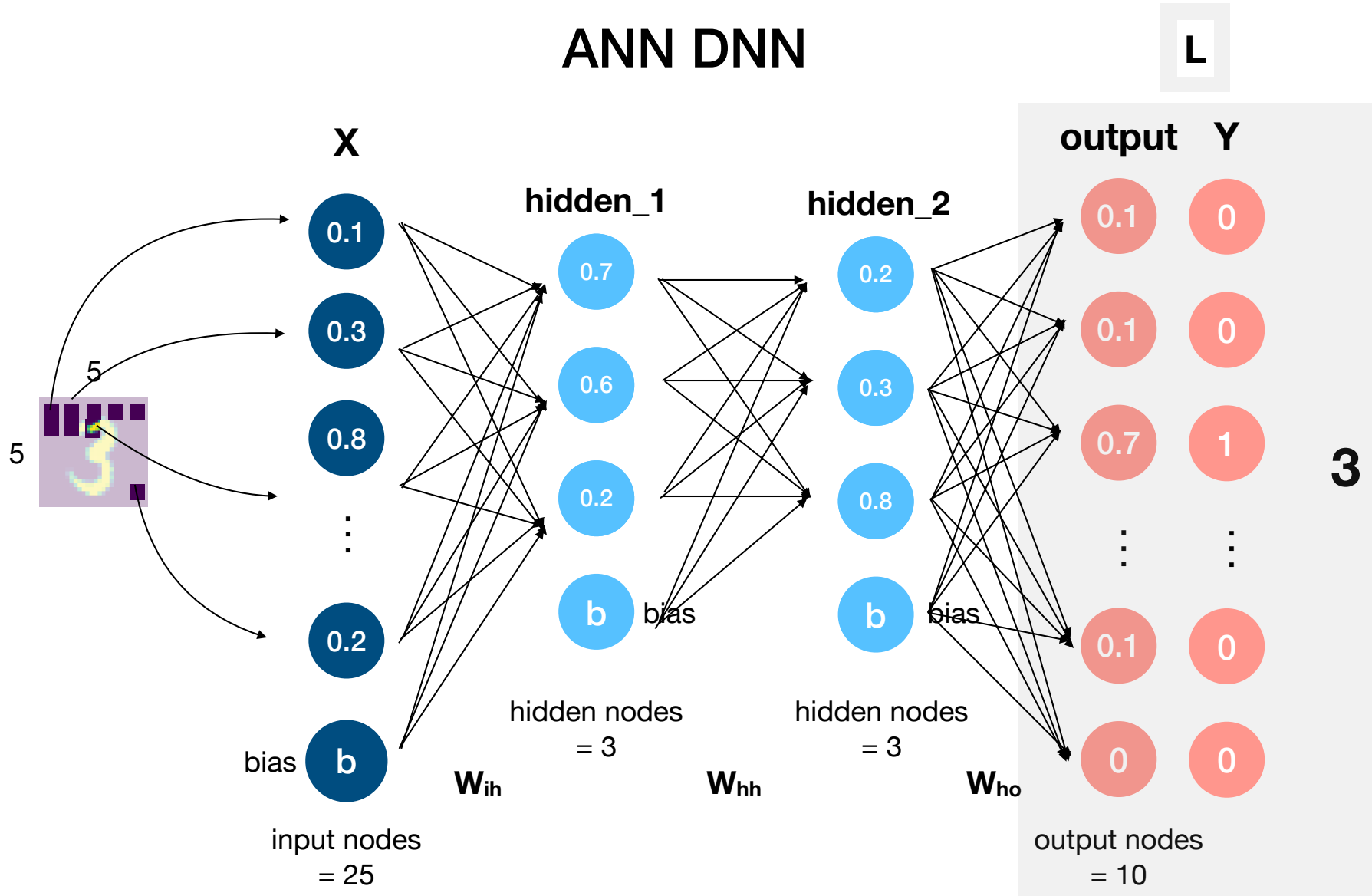
# ANN DNN



3

hidden layers의 activation function과는 다르게  
output layer에서는 activation function으로  
softmax function을 사용하며 이는 input이 각 숫자일 확률을 계산해 줍니다.

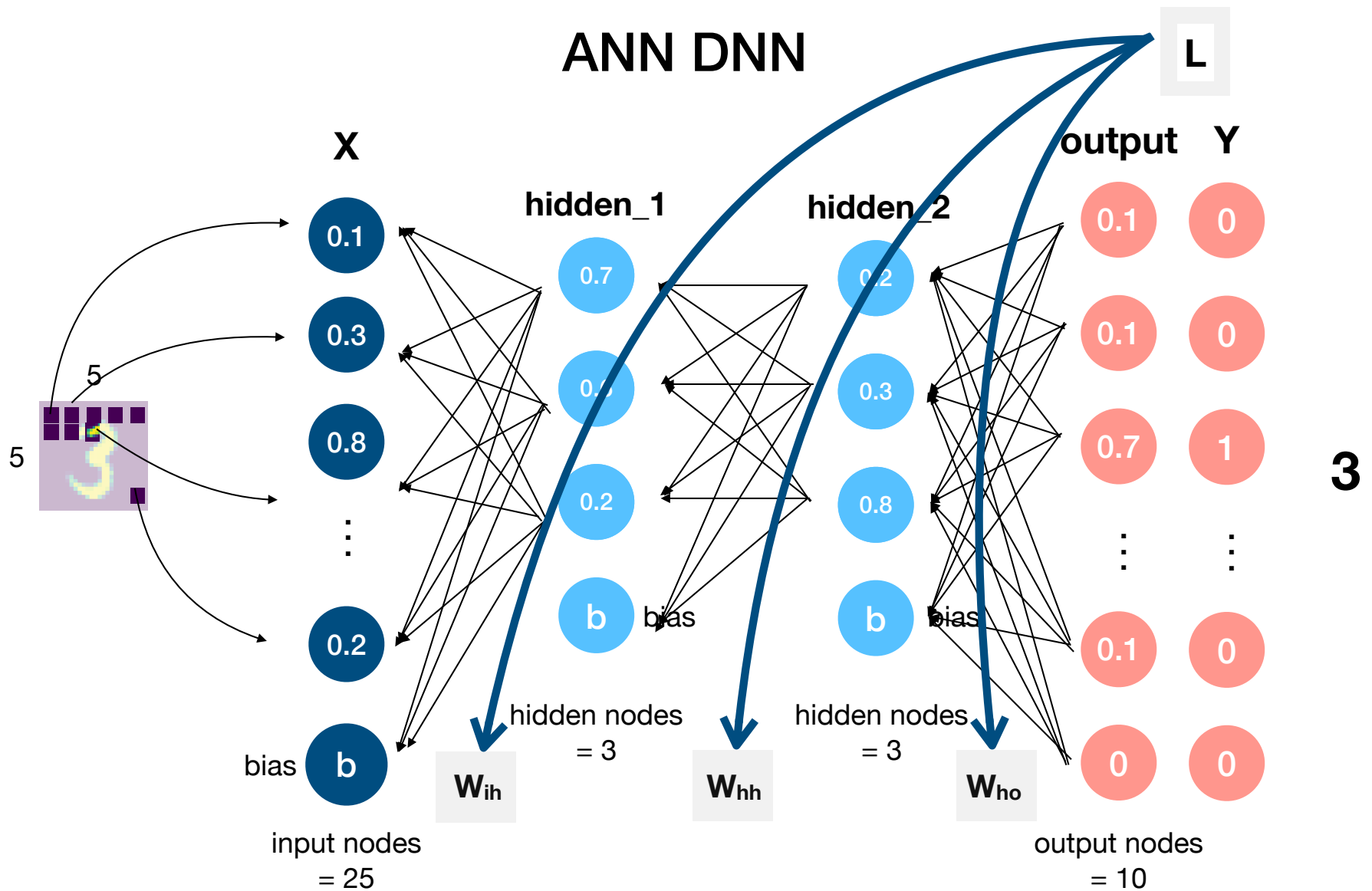
# ANN DNN



그렇게 구한 output probability와 정답 target (Y)을 비교하여 loss를 구합니다.

이 때, target은 0~9 중에 정답인 3에 해당하는 세번째 칸만 1, 나머지는 0인 one-hot vector

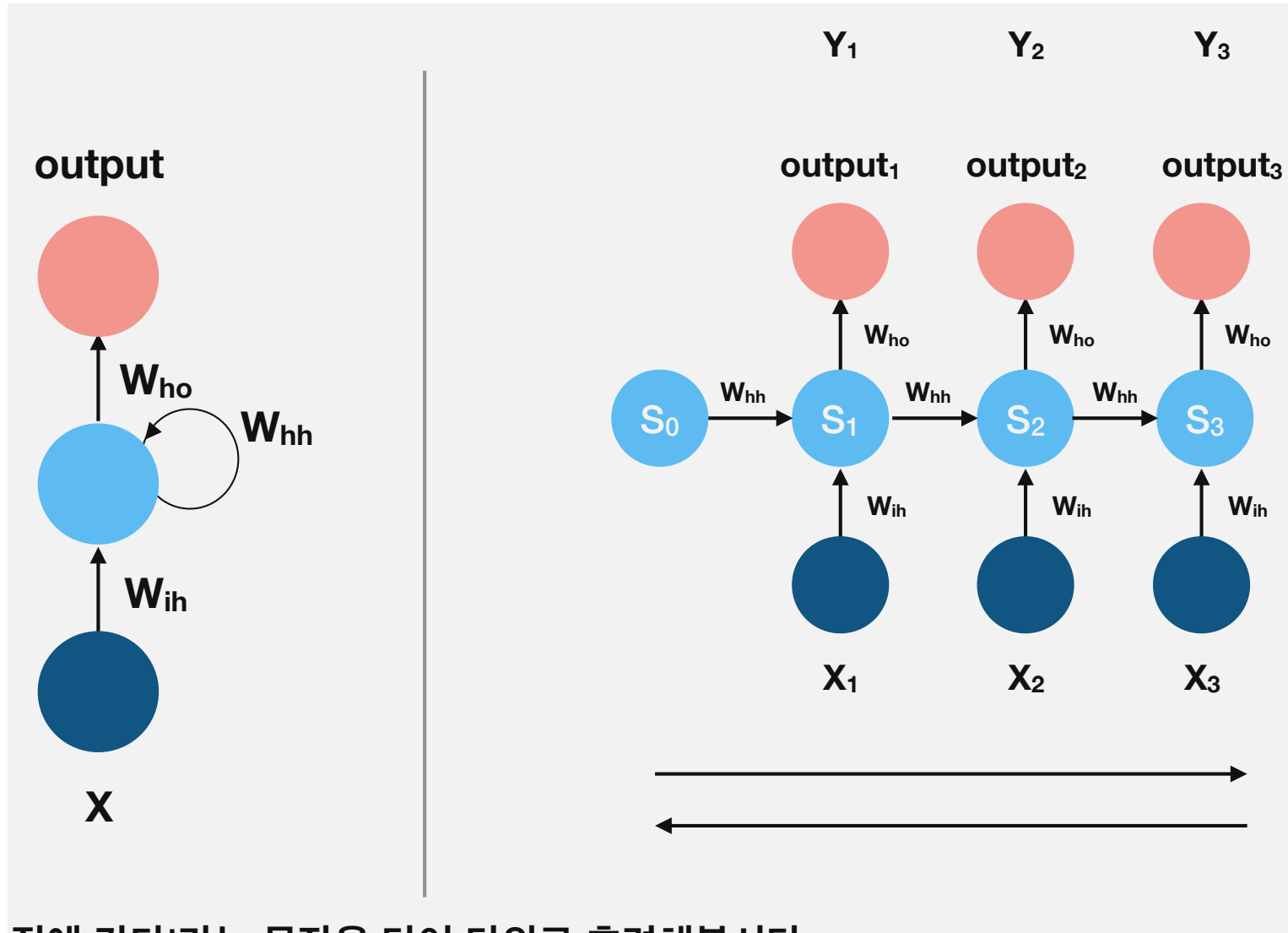
# ANN DNN



loss를 최소화하는 방향으로, weight를 update 합니다.

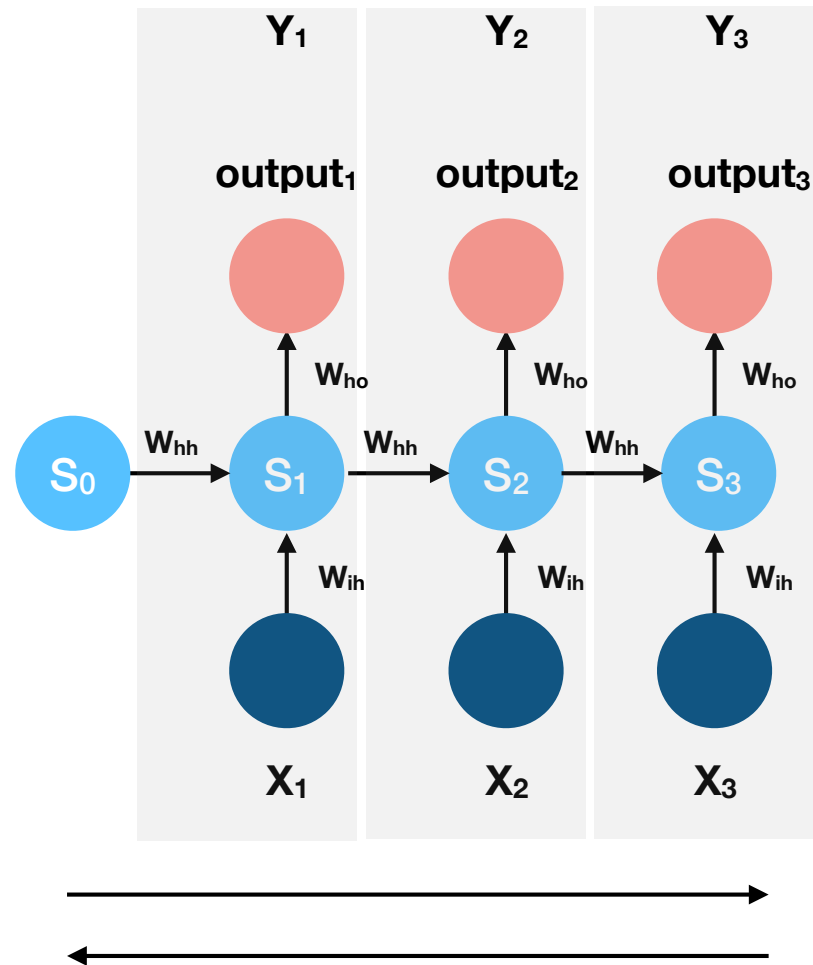
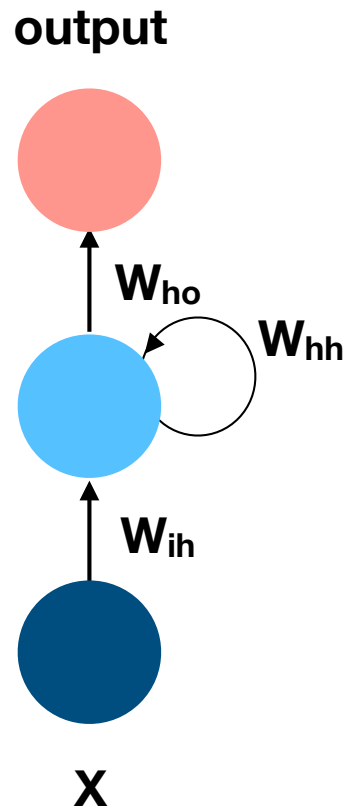
**RNN**

# RNN



'나는 오늘 집에 간다'라는 문장을 단어 단위로 훈련해봅시다.

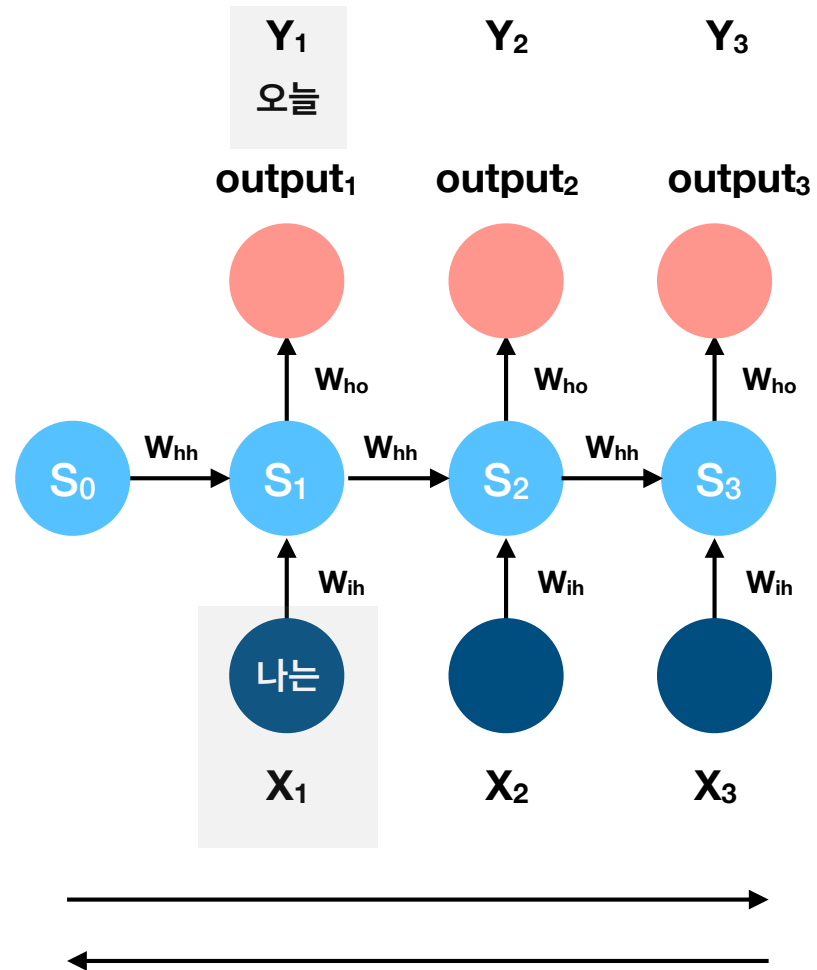
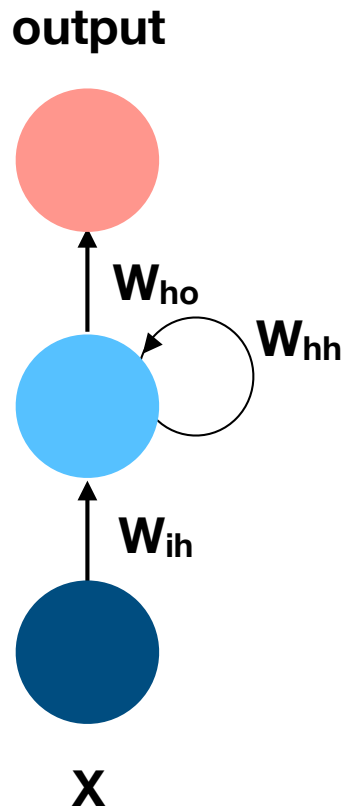
# RNN



이 예시에서는 총 3개의 time step을 가지는 RNN을 훈련할 것입니다.

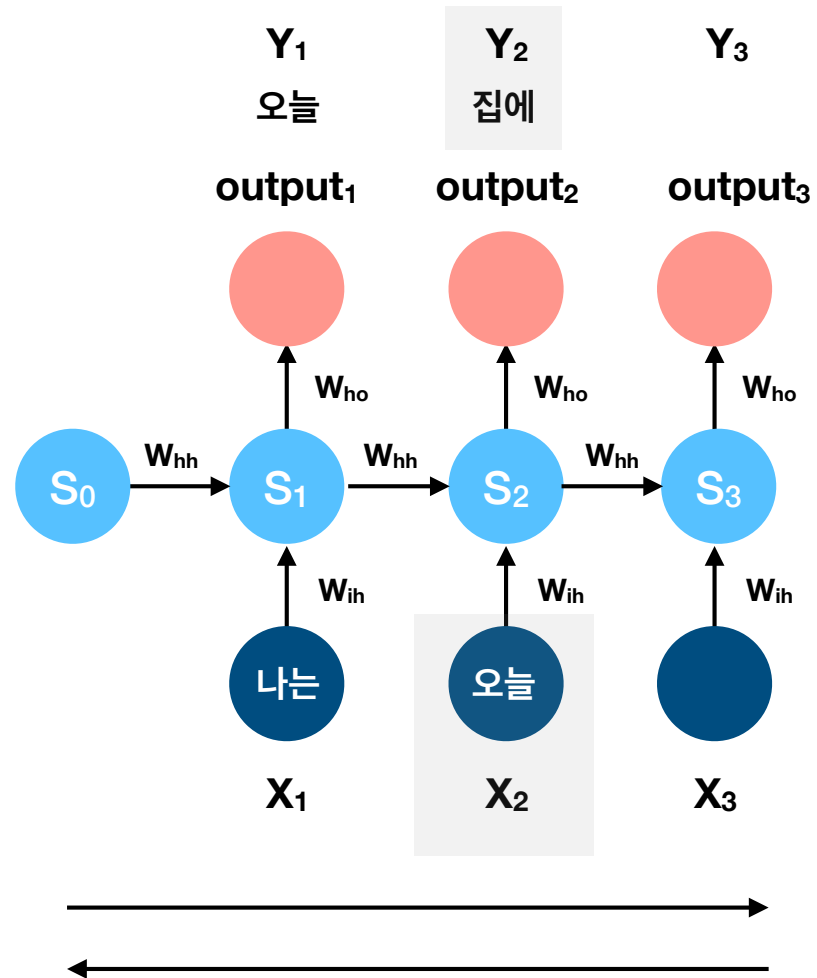
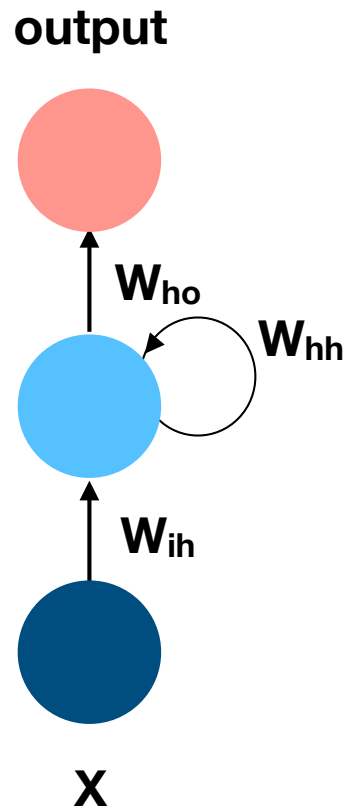


# RNN



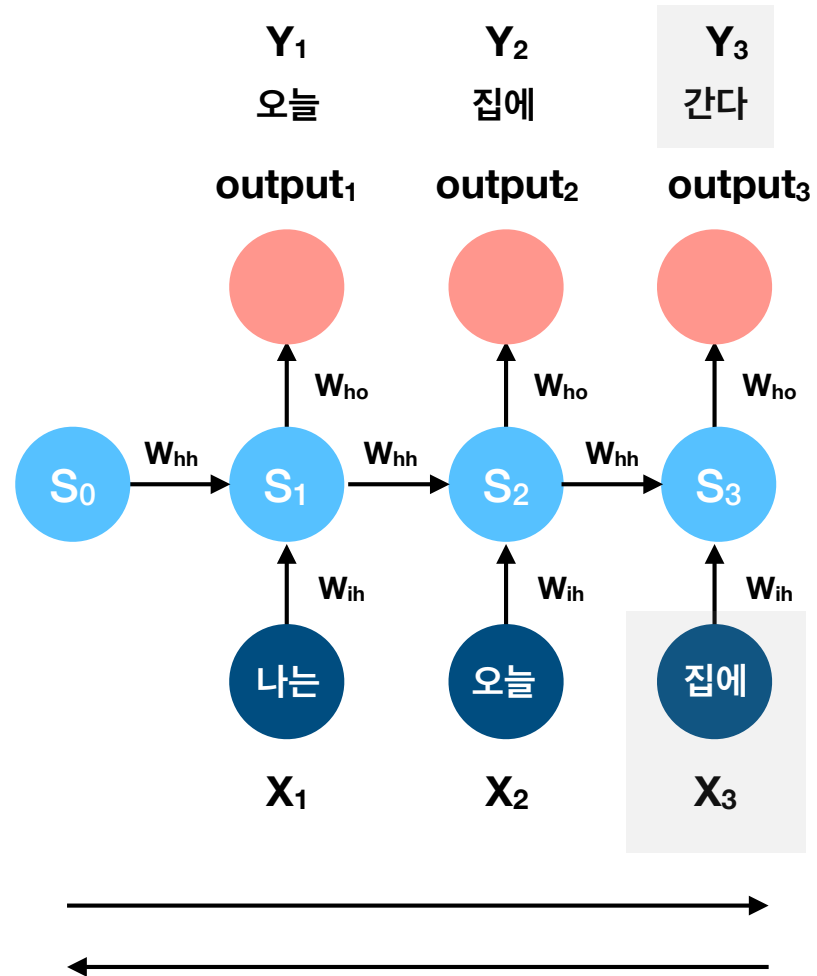
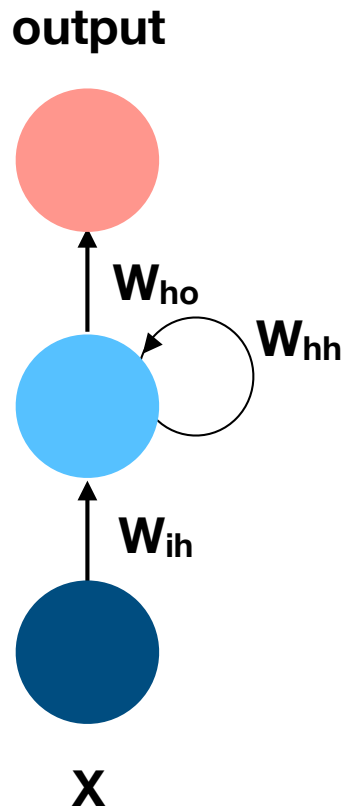
한 단어가 오면 그 다음 단어를 예측하도록 input과 target을 설정해줍니다.  
첫 time step에서는 '나는'이 input일 때 '오늘'이 target이 되도록

# RNN



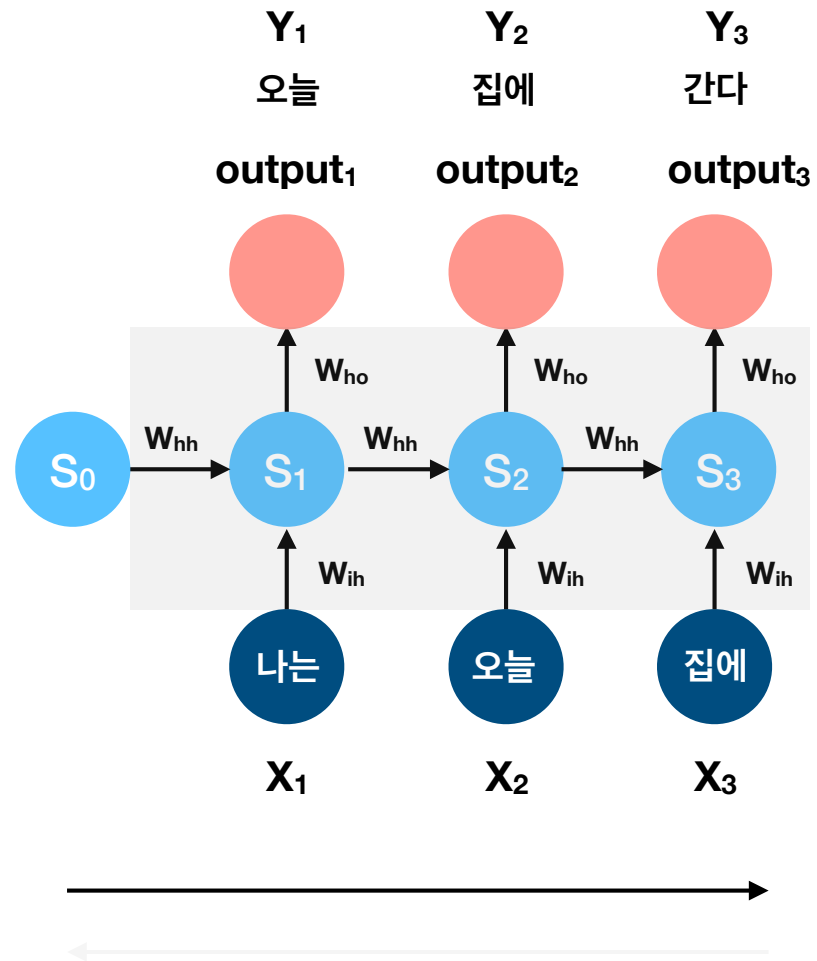
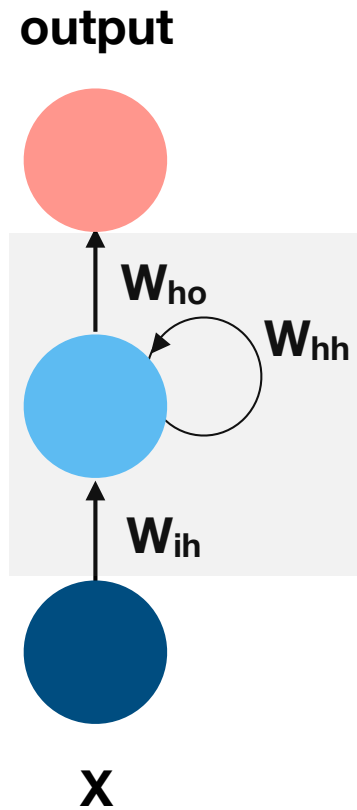
두 번째 time step에서는 '오늘'이 input일 때 '집에'가 target이 되도록

# RNN



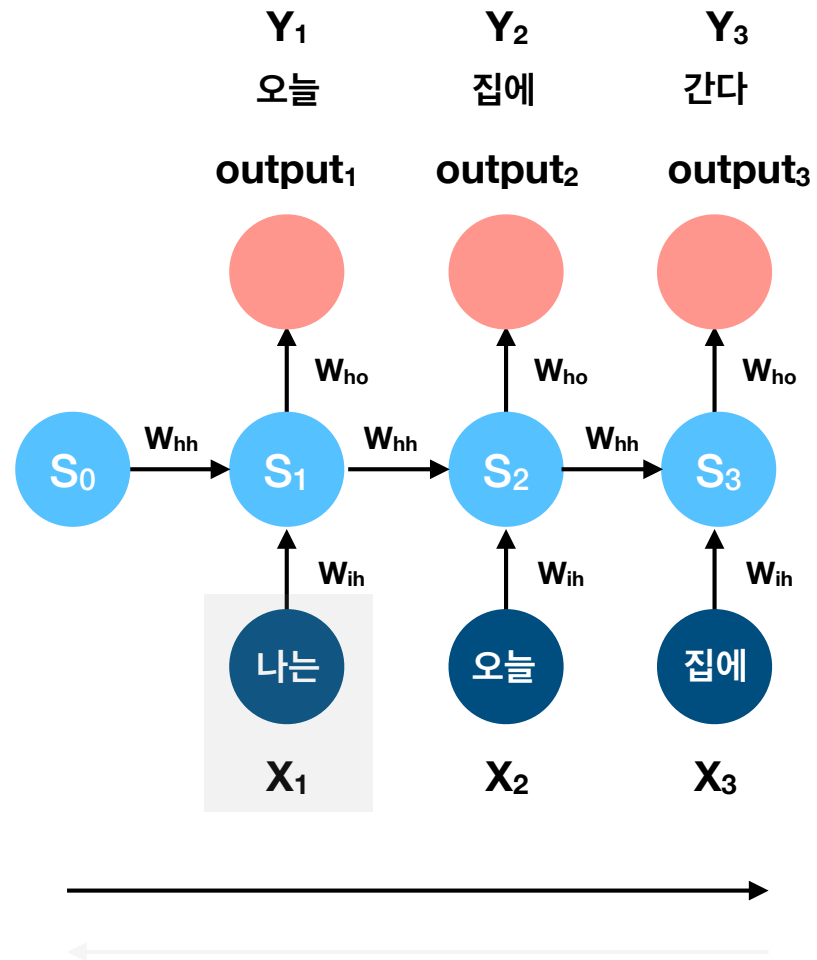
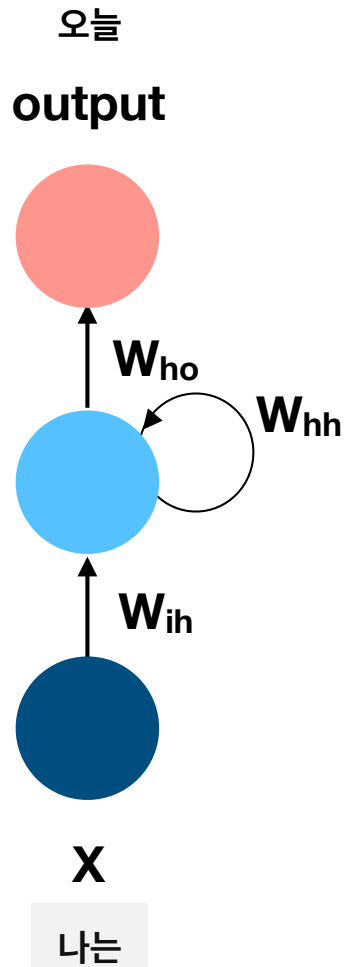
세 번째 time step에서는 '집에'가 input일 때 '간다'가 target이 되도록 RNN을 훈련해봅시다.

# RNN



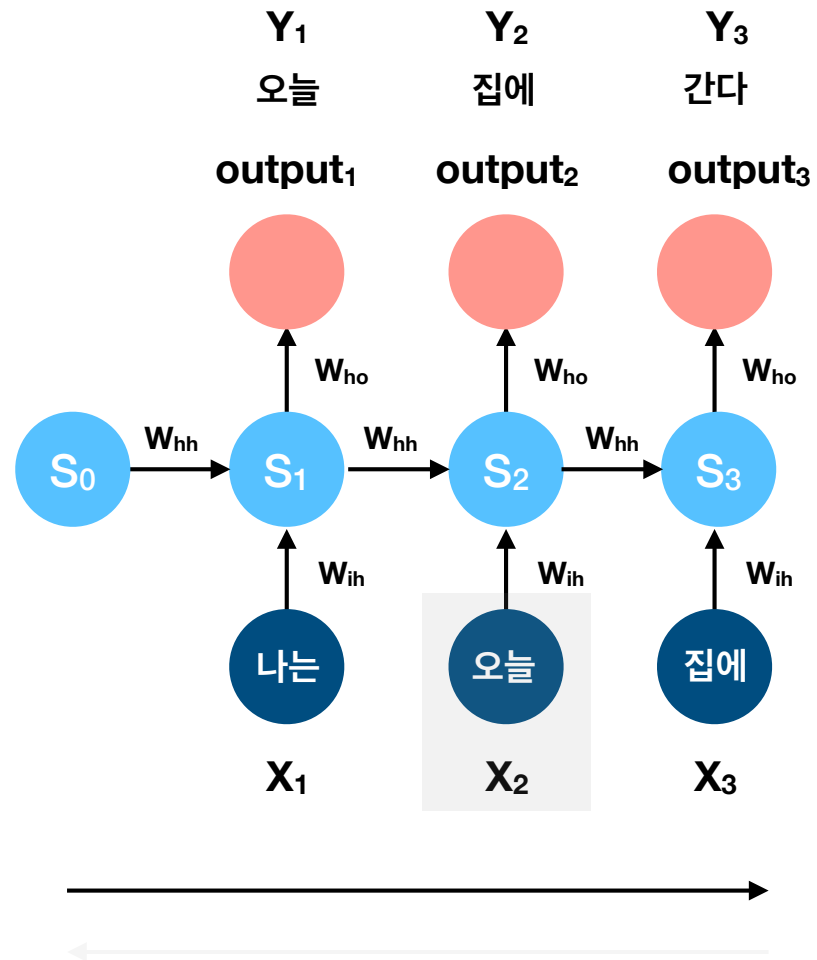
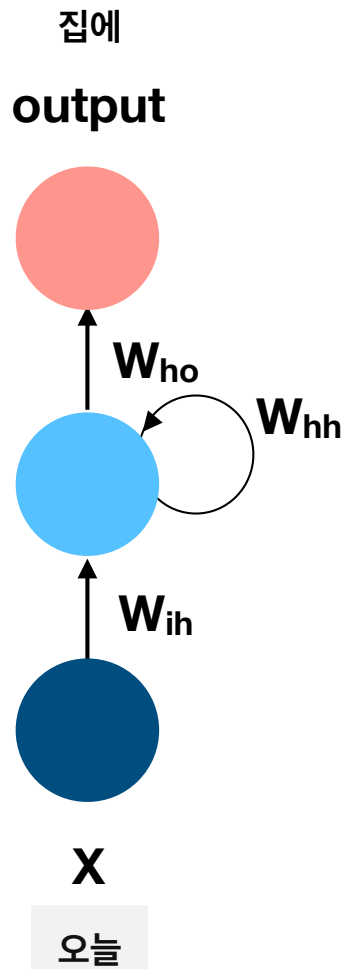
각 time step에서의  $W_{ih}$ ,  $W_{hh}$ ,  $W_{ho}$ 는 각각 동일한 weight입니다.

# RNN



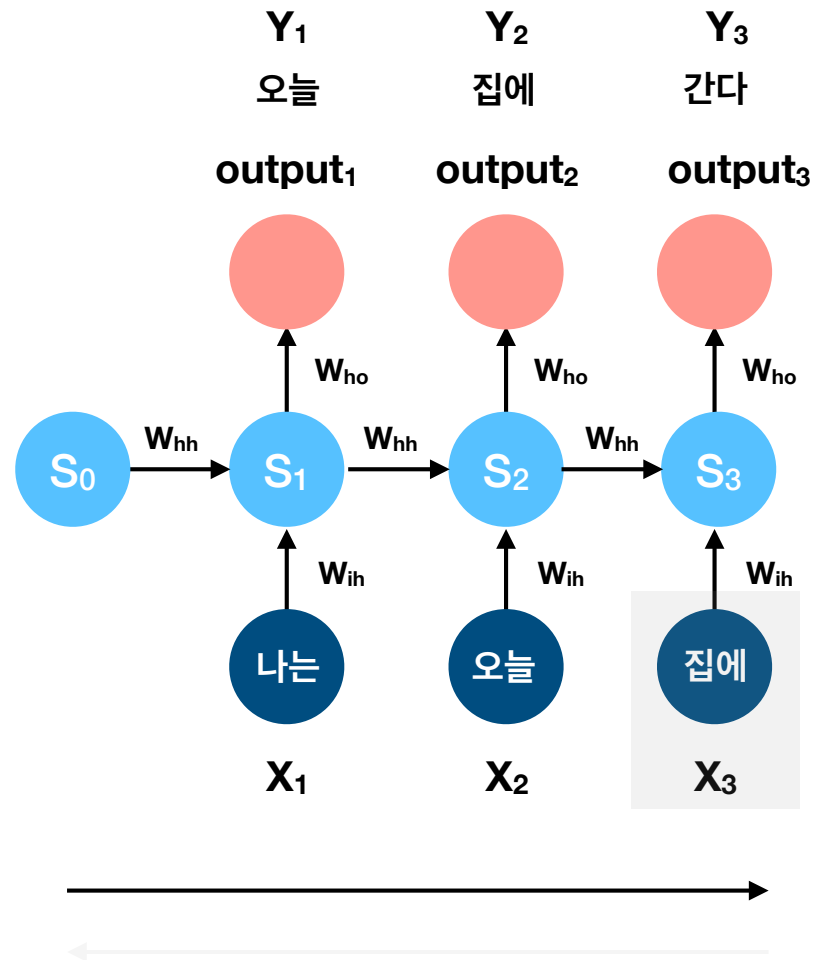
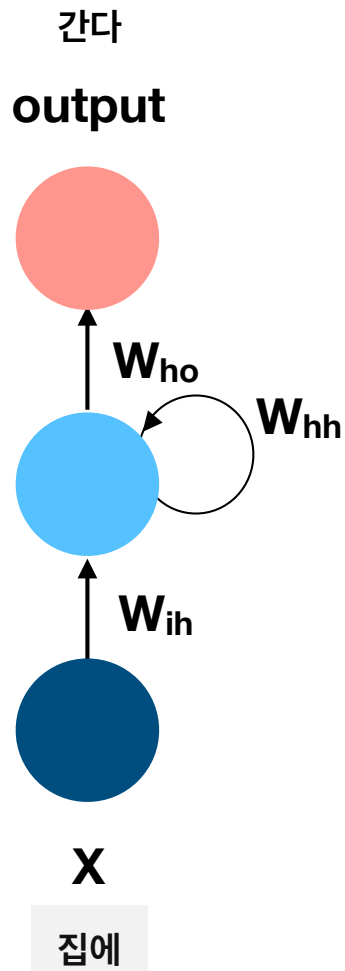
왼쪽 그림과 같은 ANN에 input  $X_1, X_2, X_3$ 가 순서대로 들어간다고 생각하면 됩니다.

# RNN



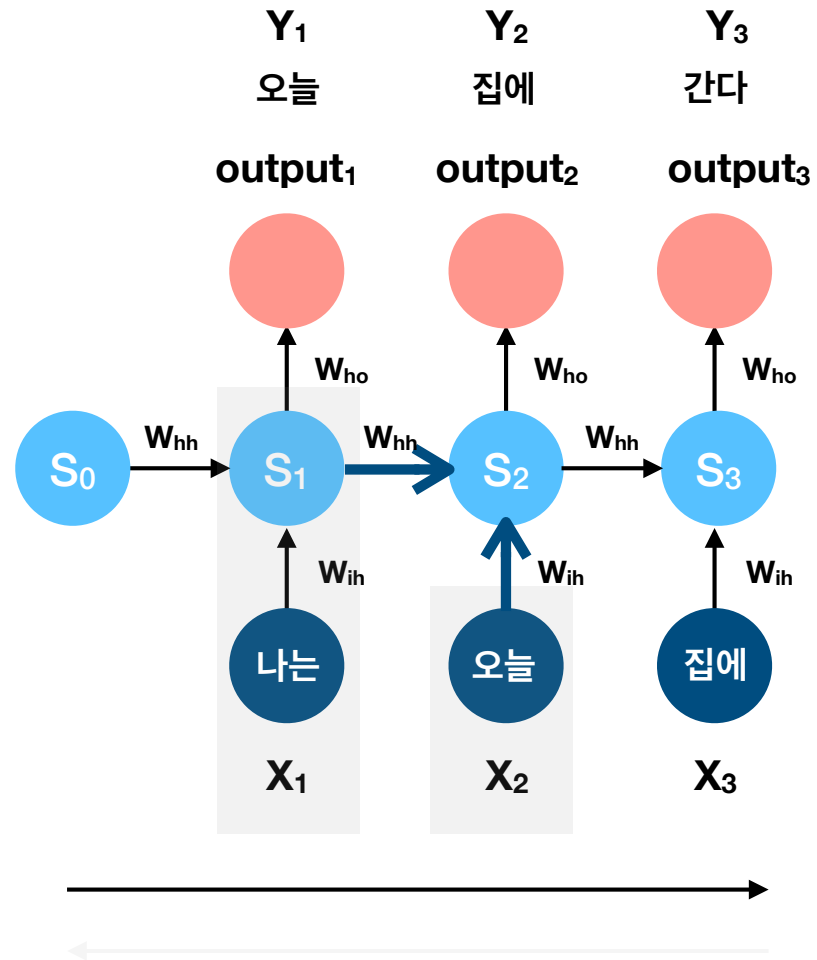
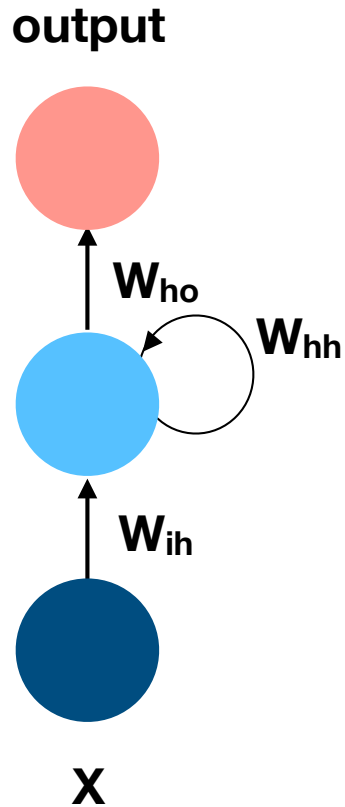
왼쪽 그림과 같은 ANN에 input  $X_1, X_2, X_3$ 가 순서대로 들어간다고 생각하면 됩니다.

# RNN



왼쪽 그림과 같은 ANN에 input  $X_1, X_2, X_3$ 가 순서대로 들어간다고 생각하면 됩니다.

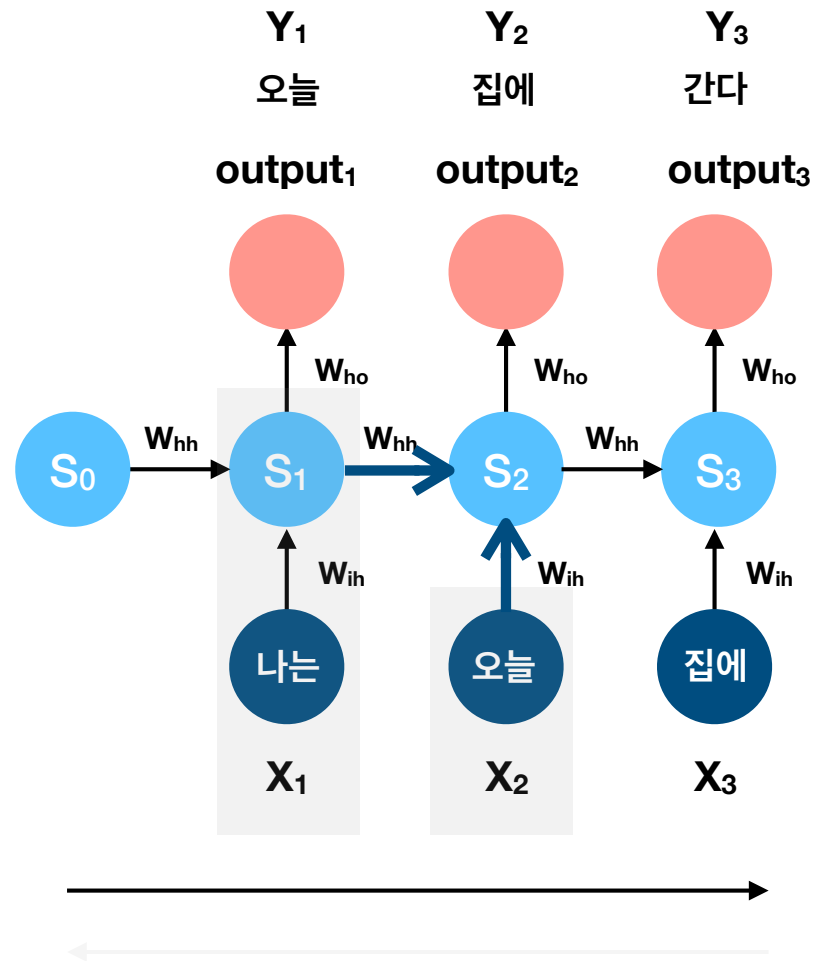
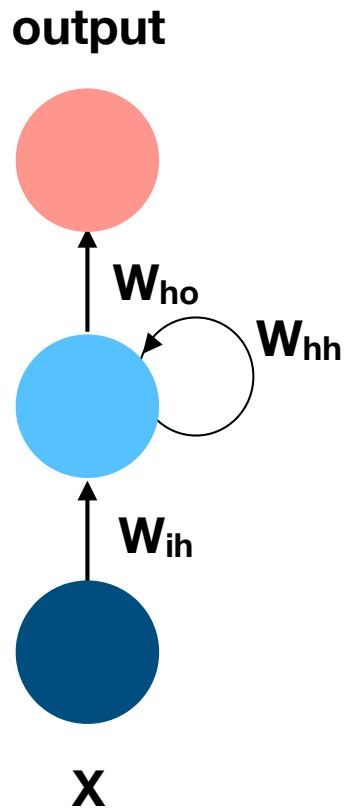
# RNN



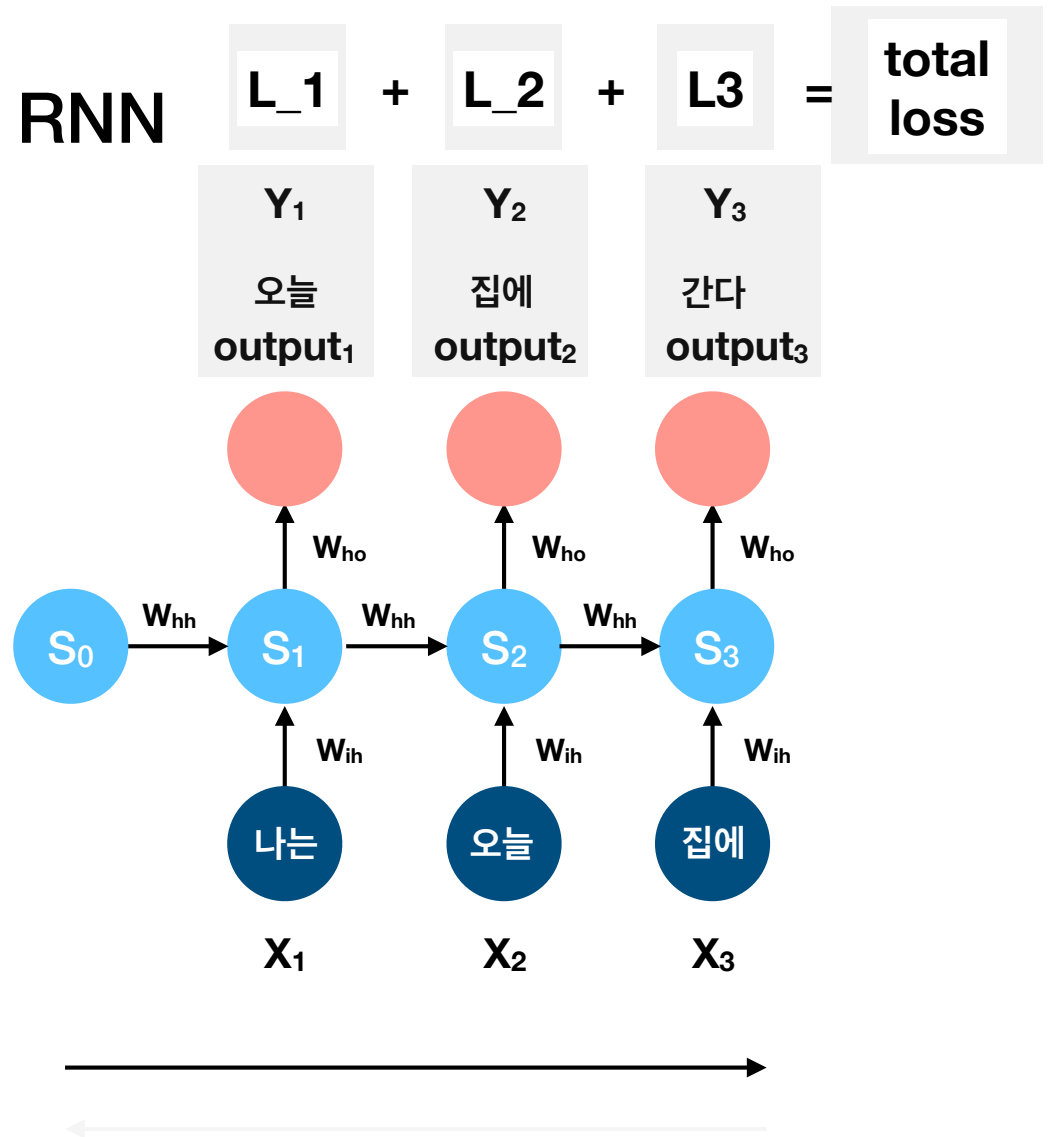
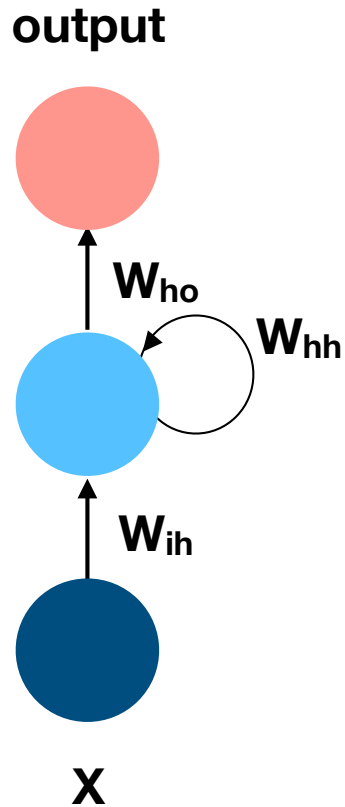
하지만 ANN과는 다르게 hidden layer는 현재 time step의 input 뿐만 아니라  
이전 time step의 hidden layer의 영향도 받습니다. (  $S(t) = S(t-1) * W_{hh} + X(t) * W_{ih}$  )



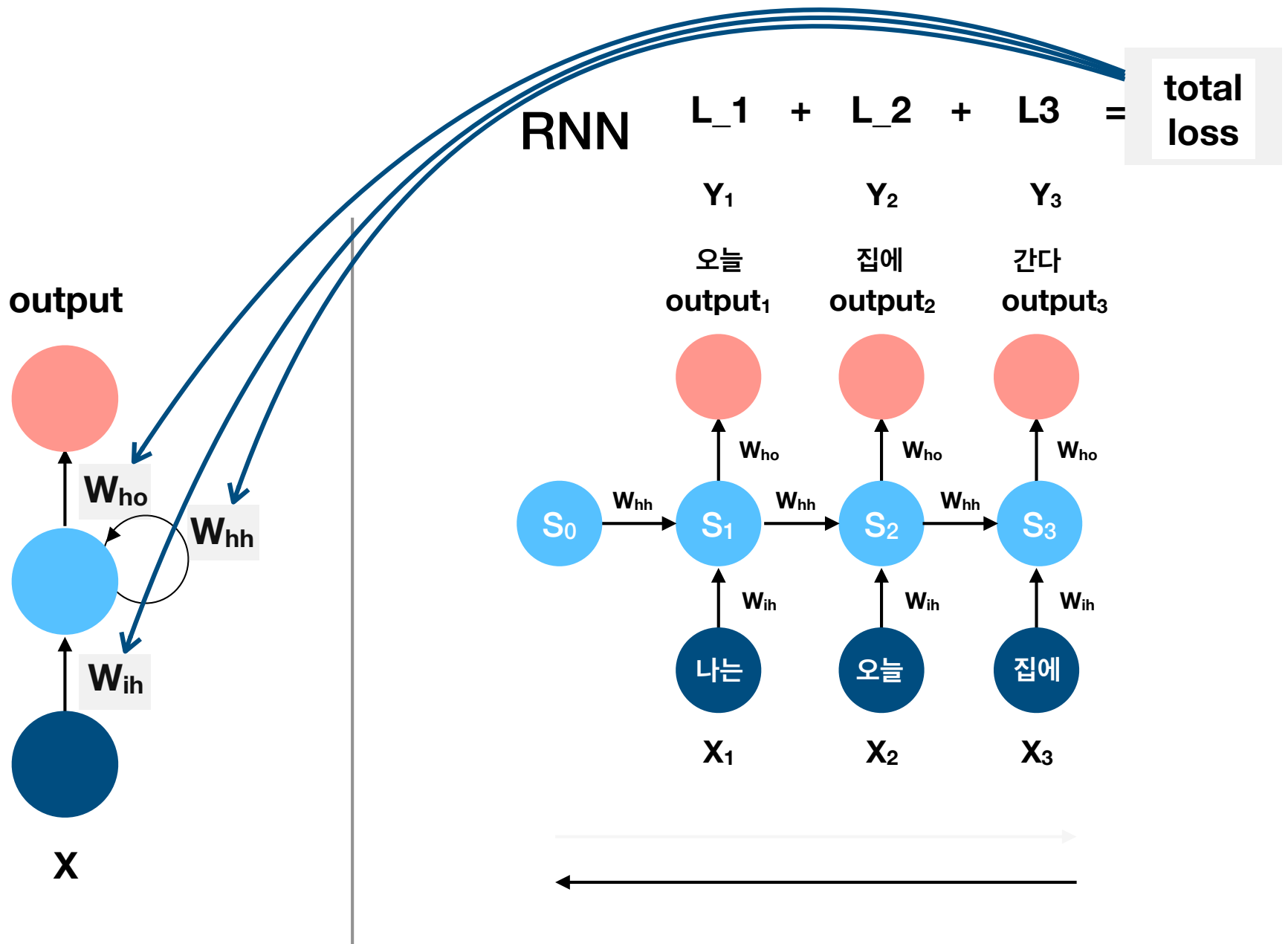
# RNN



즉 '오늘'이란 단어가 두 번째 input ( $X_2$ )으로 들어갔을 때 이전 input ( $X_1$ )인 '나는'이 영향을 줍니다.

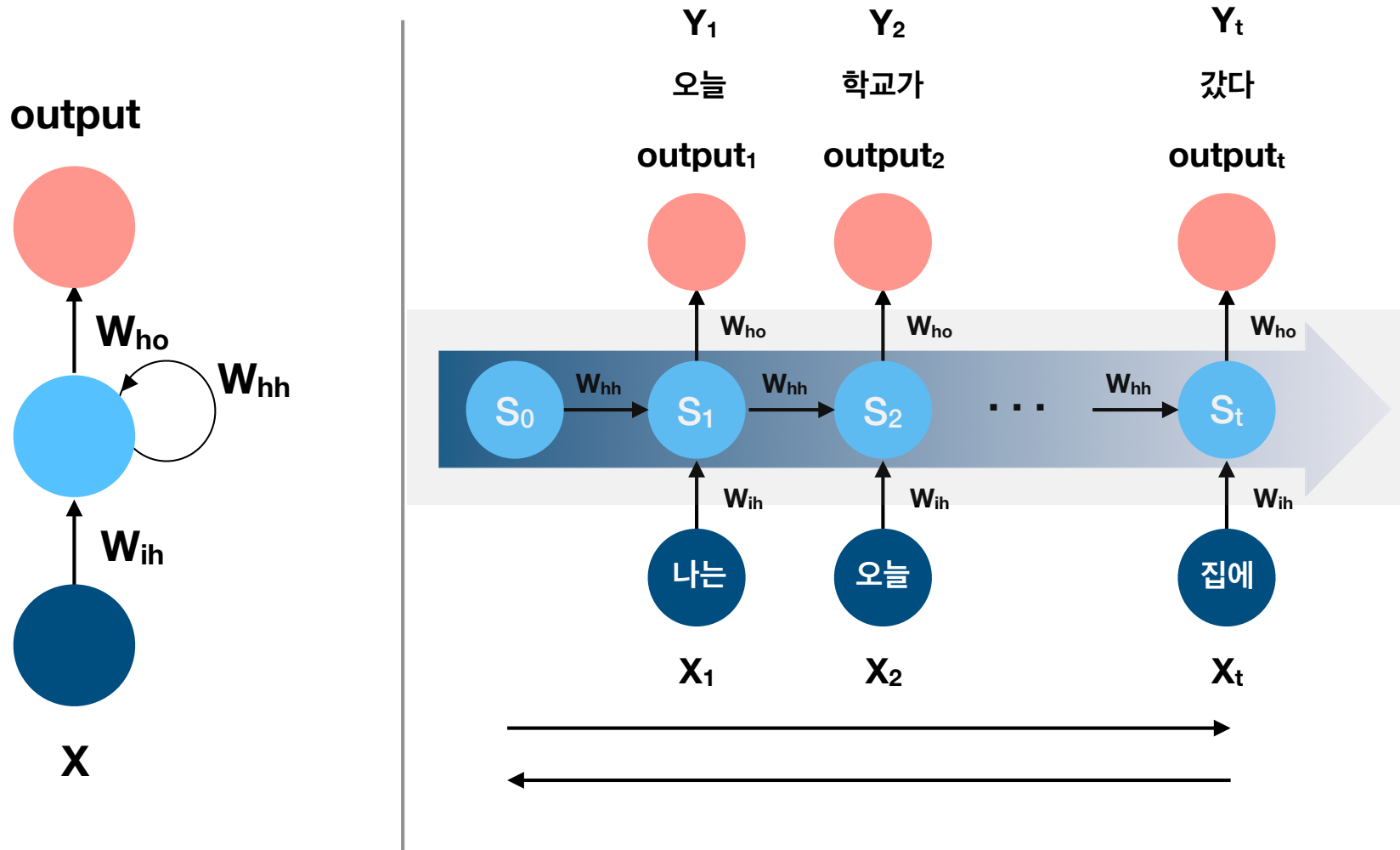


output과 정답 target (Y)을 비교하여 loss를 구합니다.  
 그러면 각각의 time step에서 loss가 나오게 될 것이고 이를 모두 더하여 전체 loss라고 정의합니다.



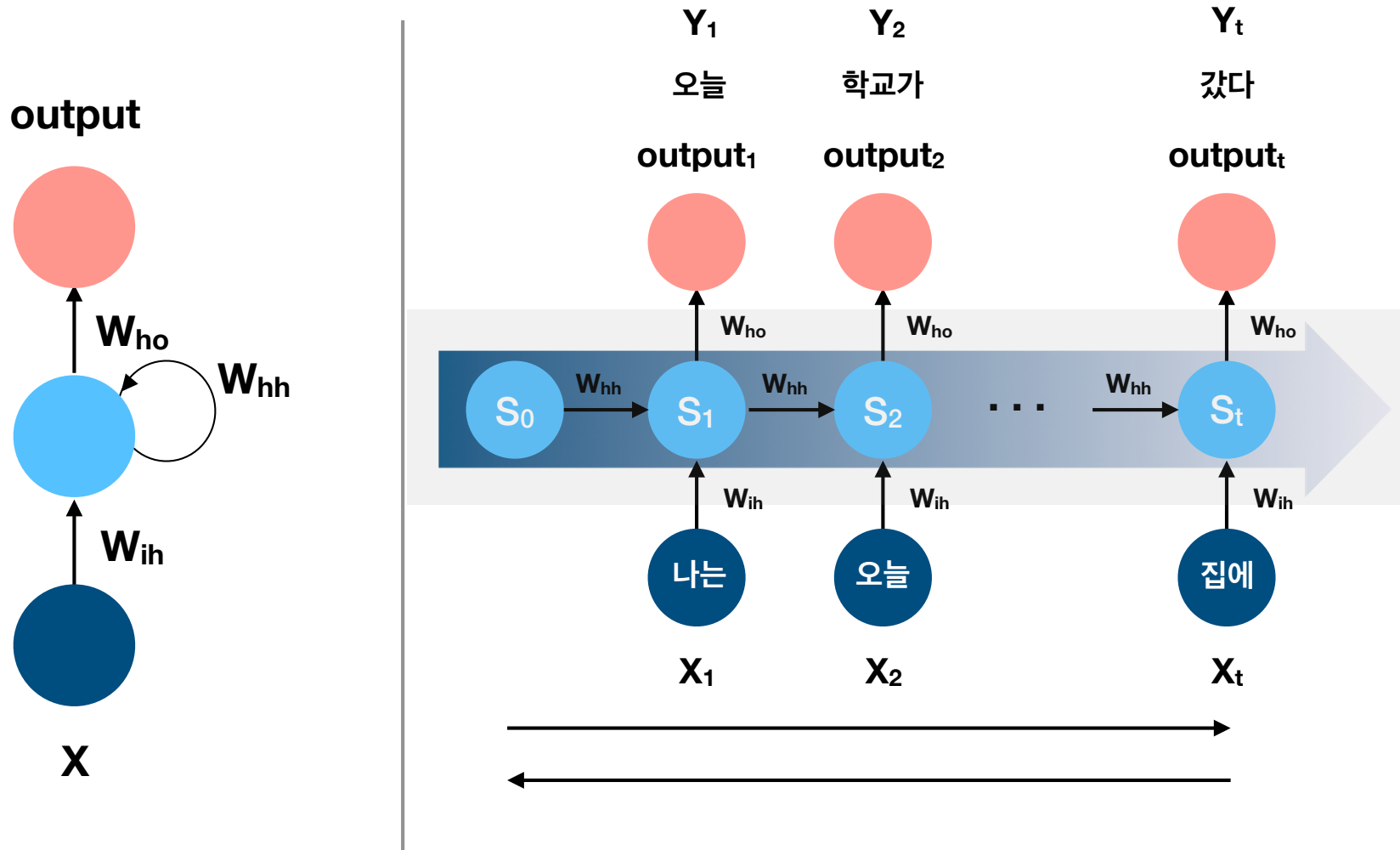
이 전체 loss를 최소화하는 방향으로, weight을 update 합니다.

# RNN



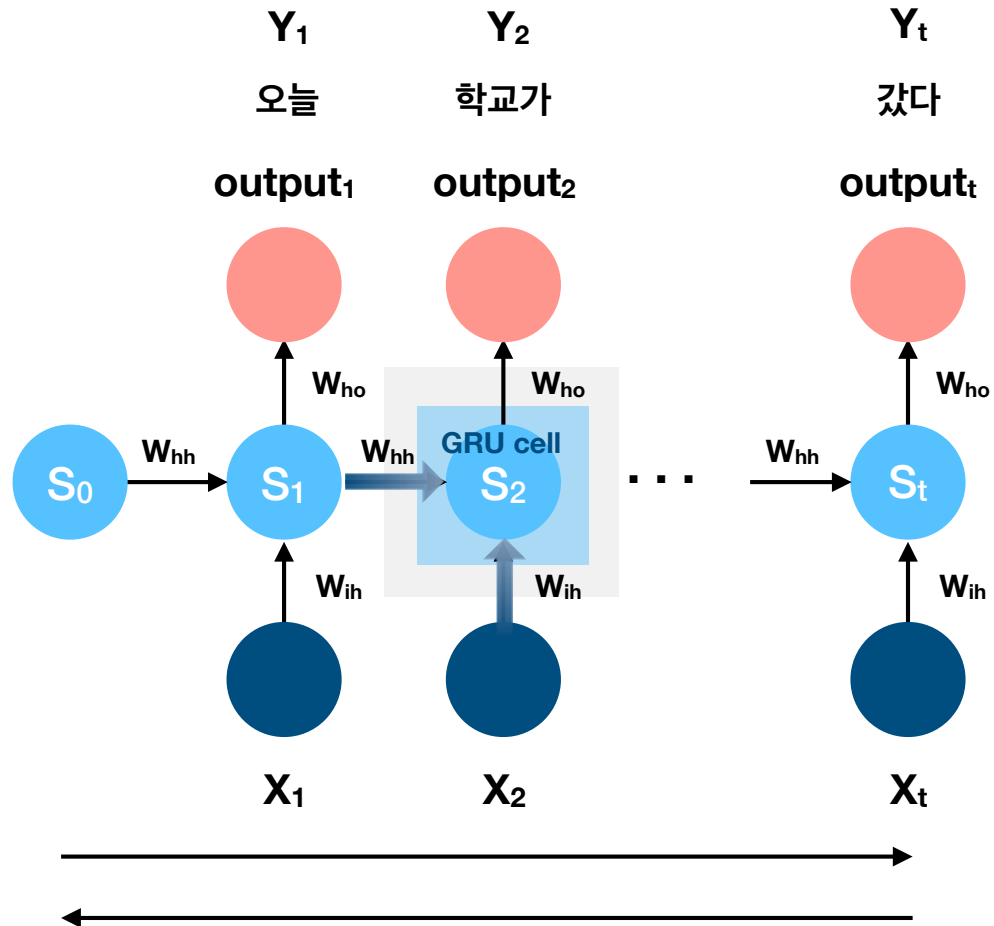
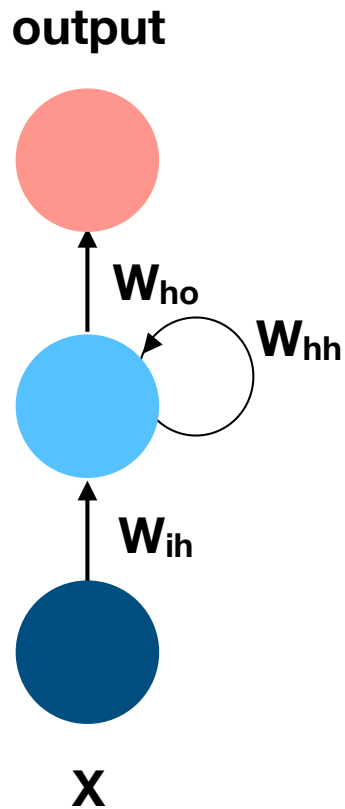
가장 기본적인 RNN은 훈련할 time step의 수가 많아졌을 때,  
뒤쪽의 time step이 앞쪽 time step의 정보를 제대로 반영하지 못하는 한계가 있습니다.

# RNN



예를 들어, '나는 오늘 학교가 끝나고 서점에 들러 책을 산 후 집에 갔다'라는 11개의 time step으로 이루어진 문장을 훈련할 때 마지막 11번째 time step의 hidden node  $S_{11}$ 은 첫 번째 time step의 input인 '나는'에 대한 정보를 거의 갖고 있지 않습니다.

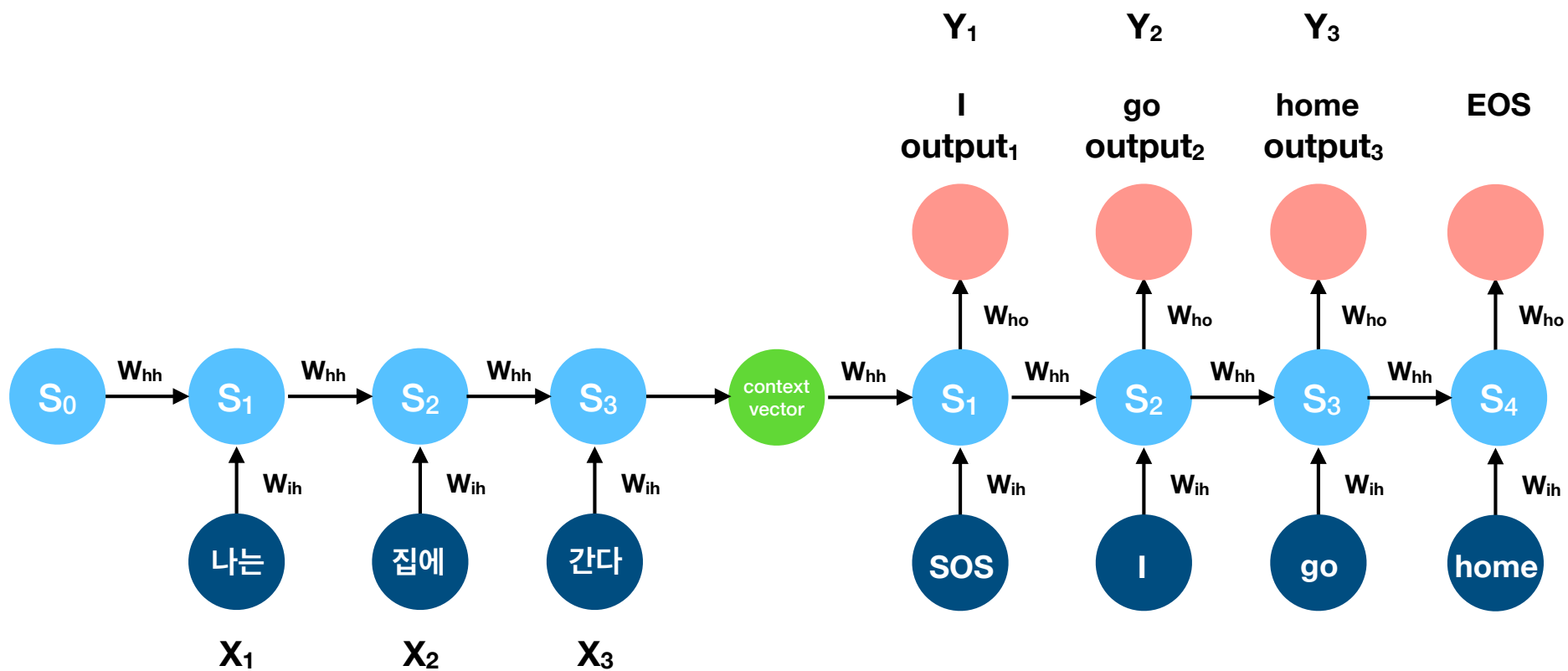
# RNN



따라서 hidden layer가 이전 time step의 정보를 보다 효율적으로 기억할 수 있도록 하는 방법이 고안되었습니다.

**SEQ2SEQ**

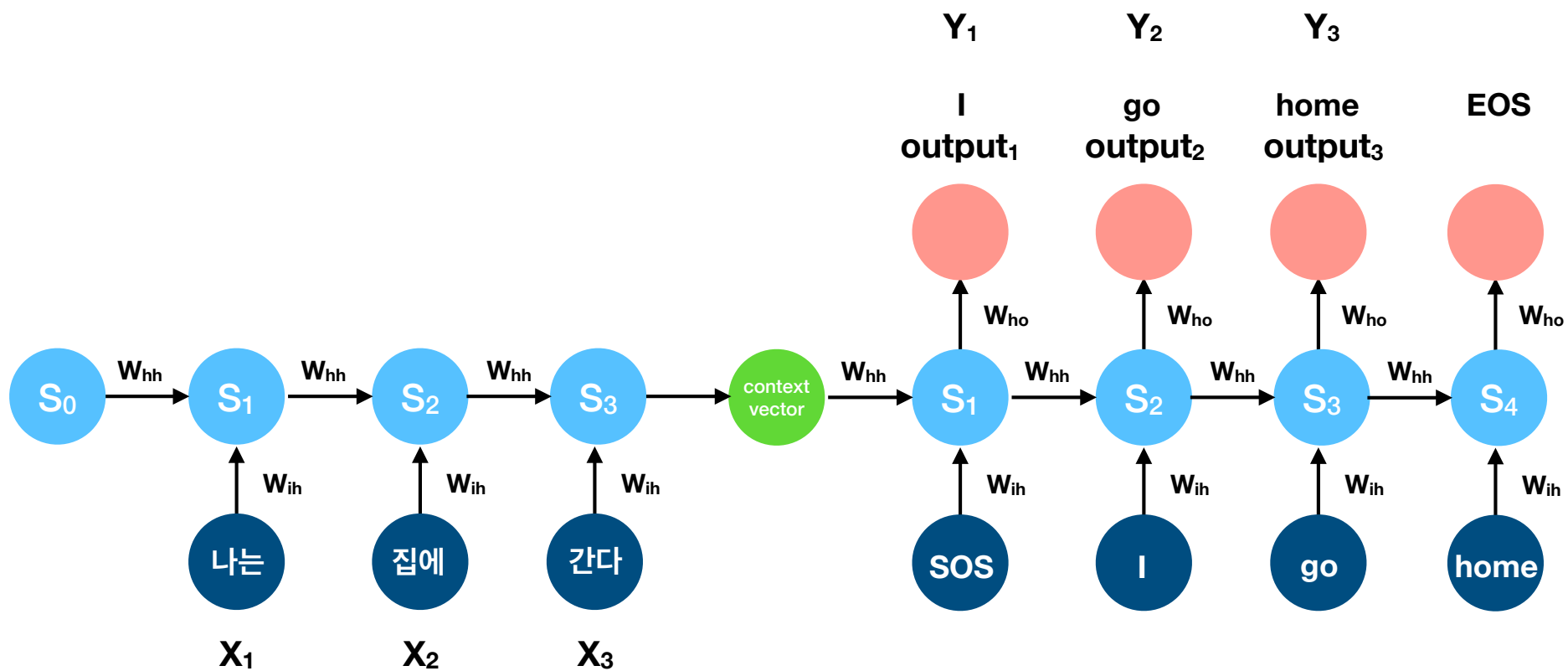
# SEQ2SEQ



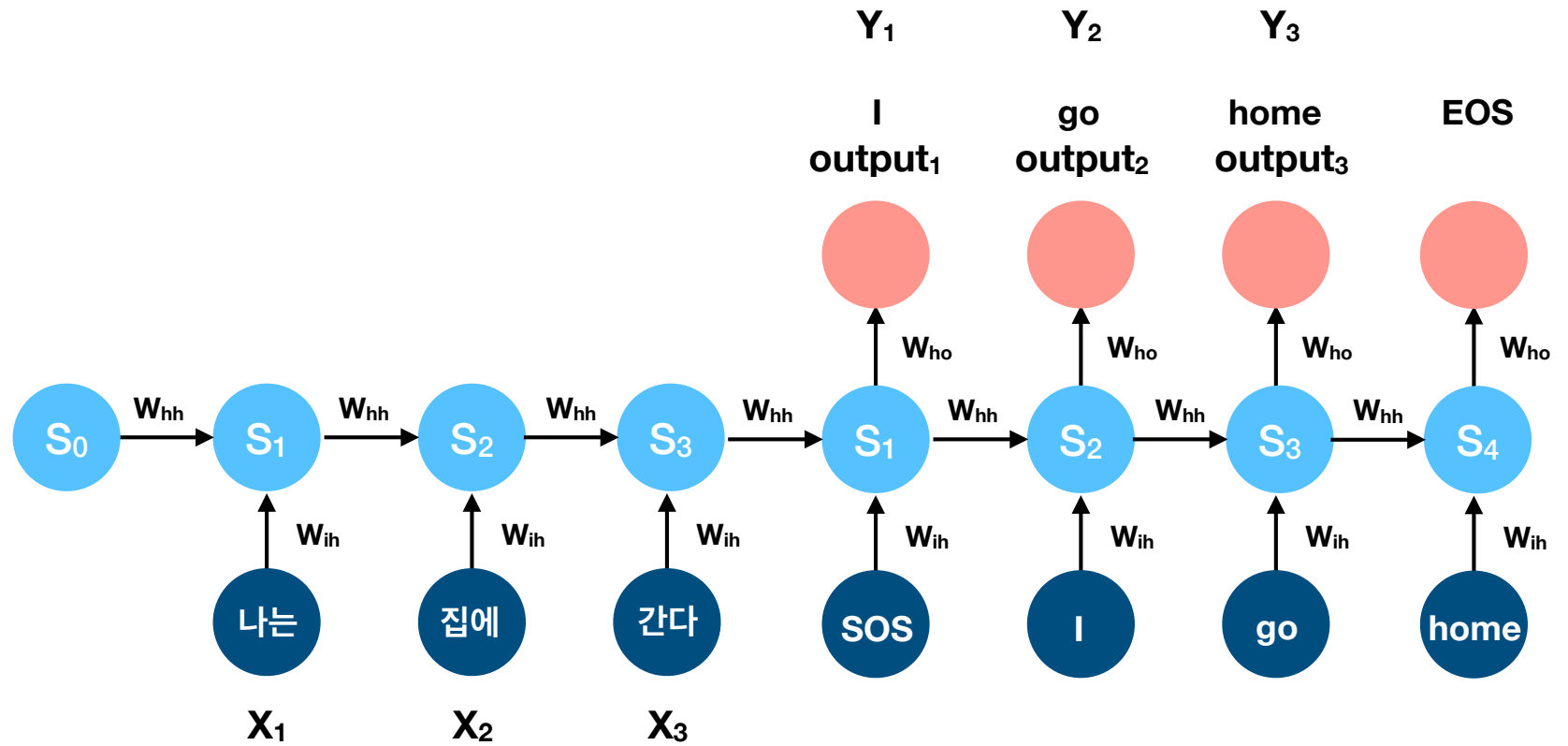


# Transformer

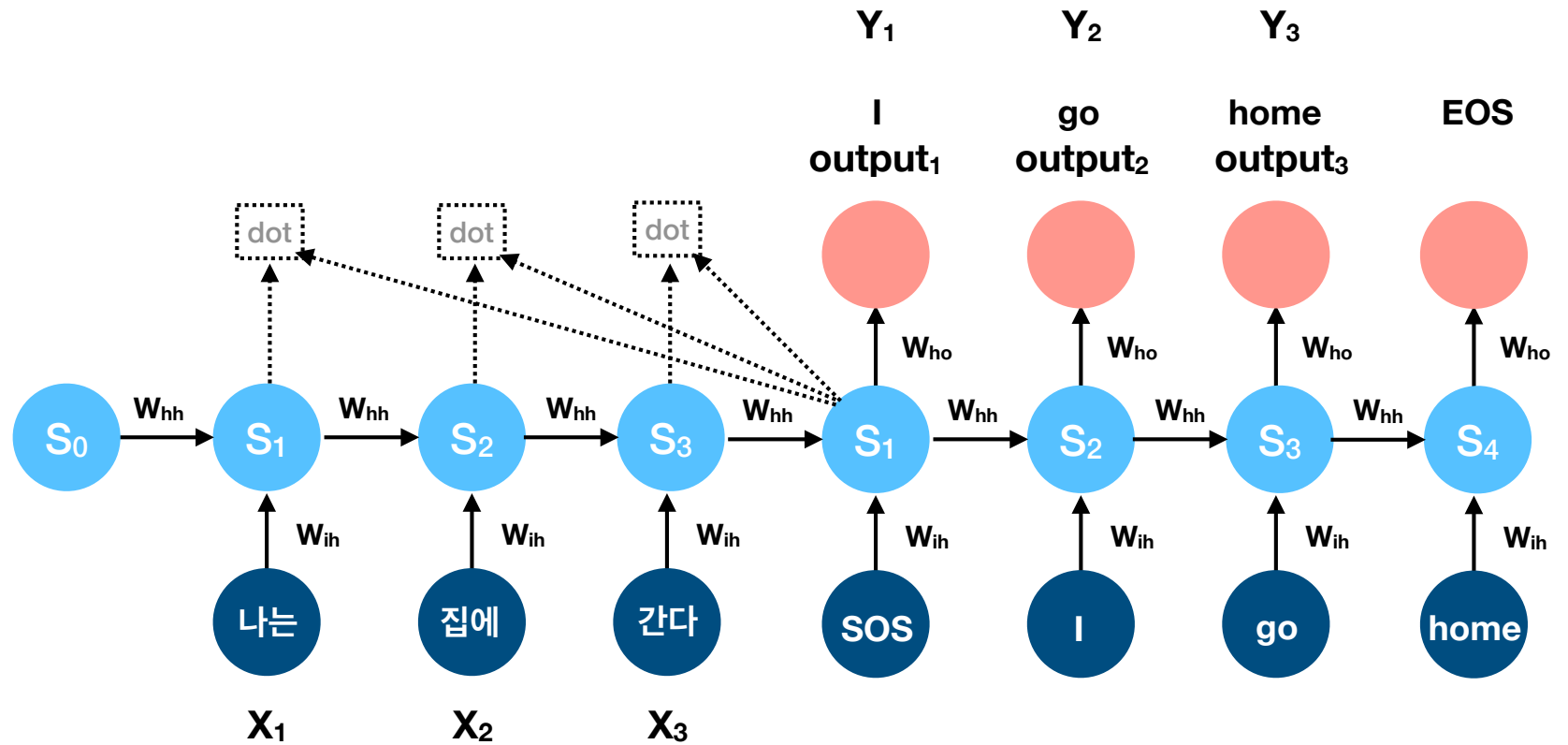
# SEQ2SEQ



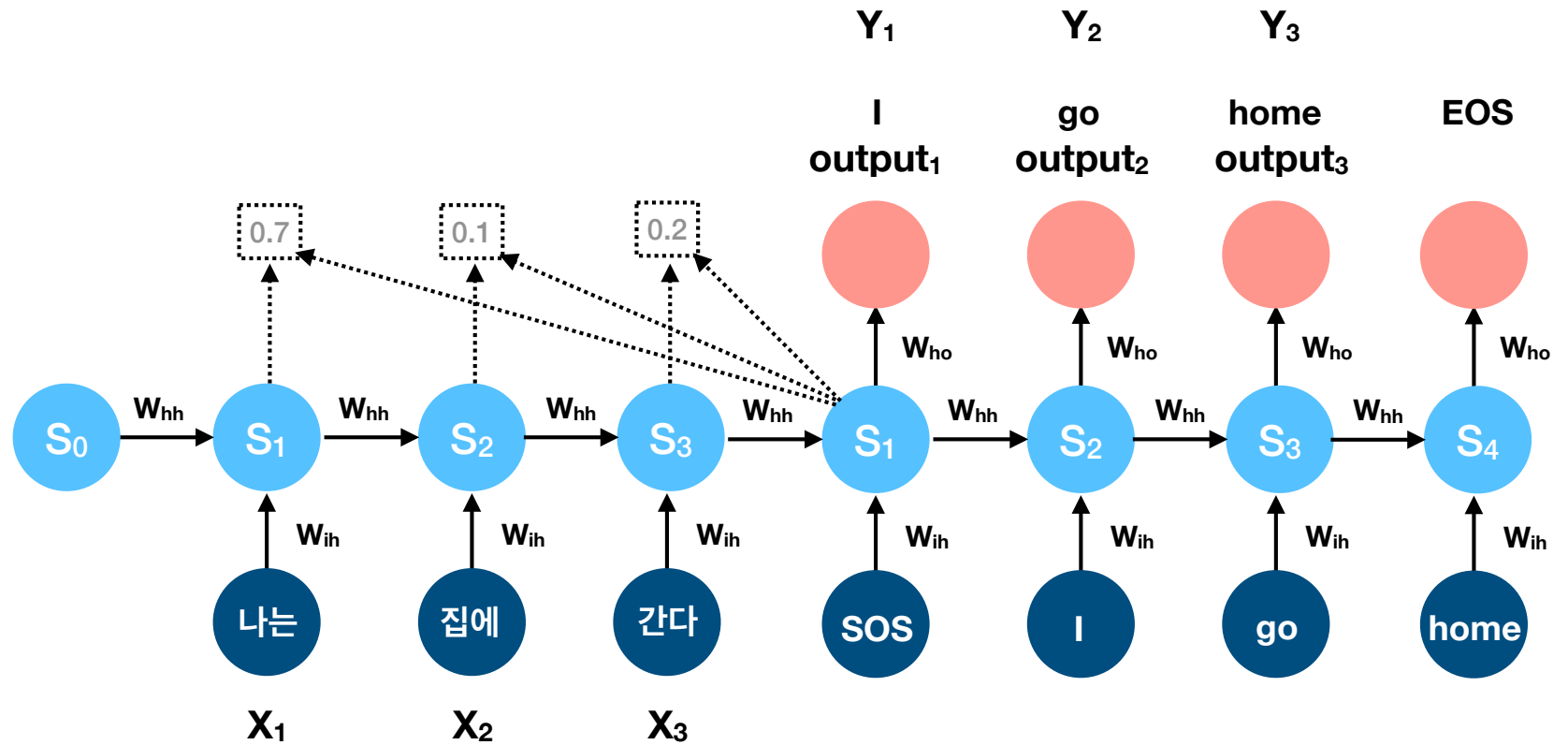
# Transformer



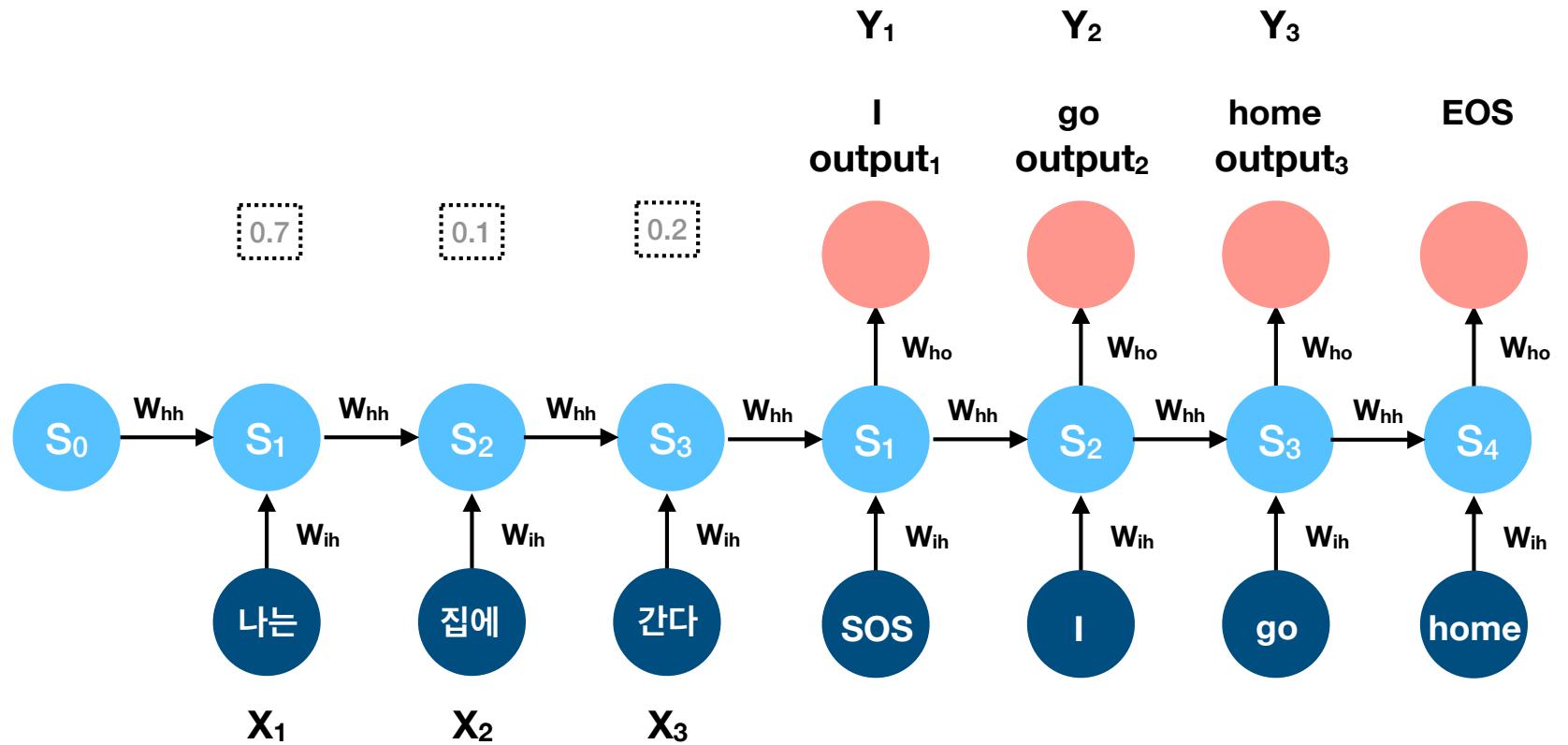
# Transformer



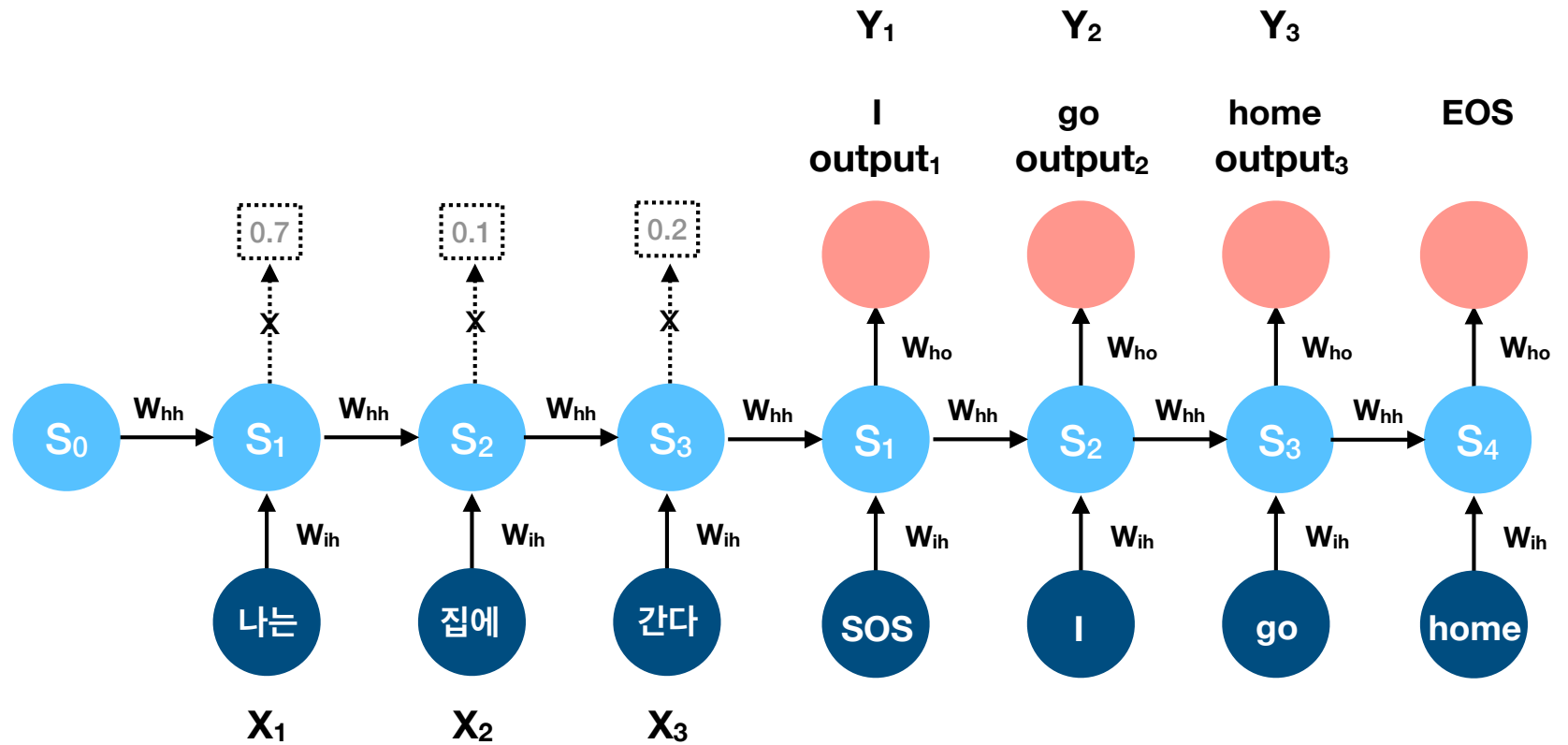
# Transformer



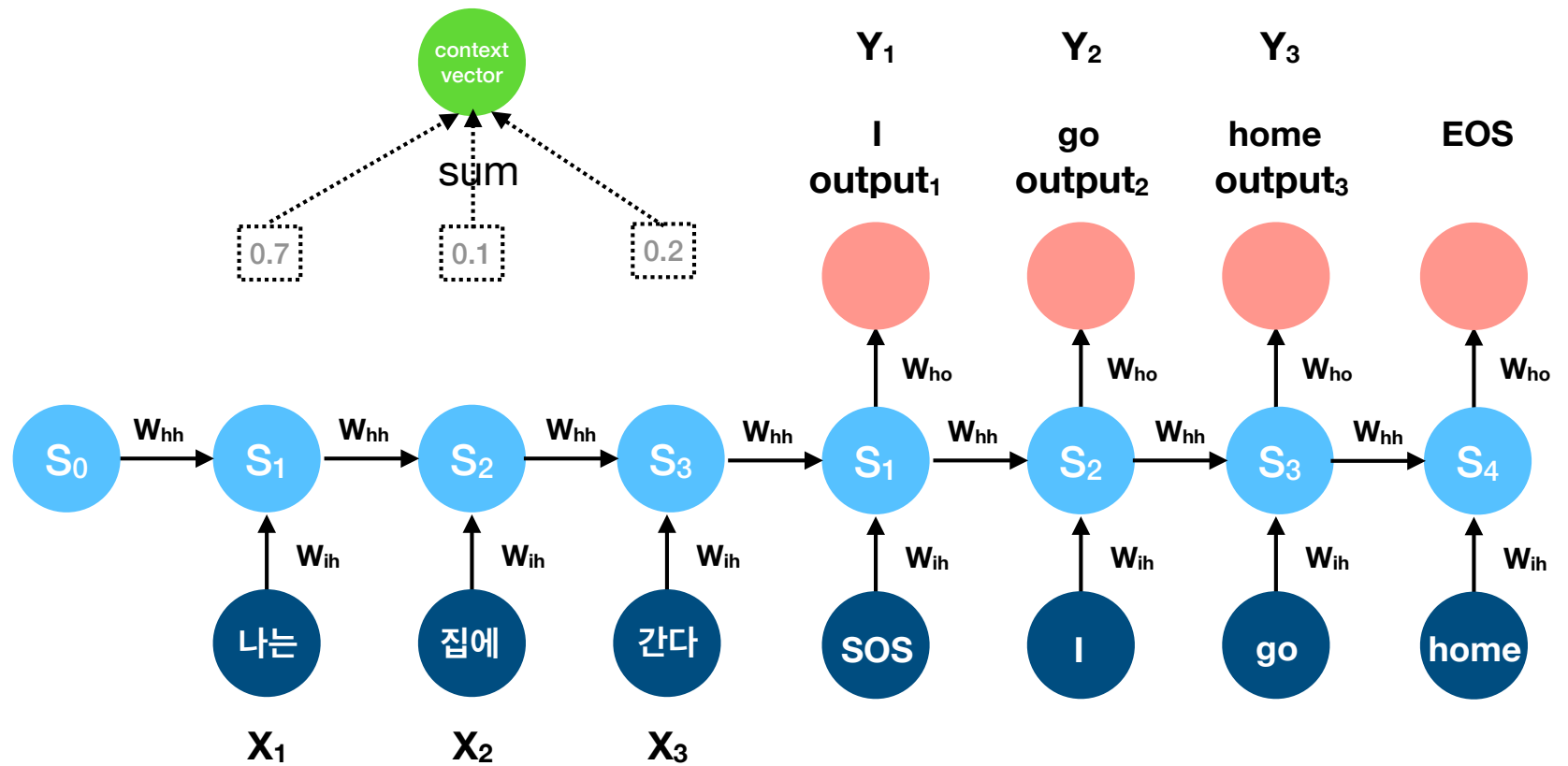
# Transformer



# Transformer



# Transformer





# Transformer

