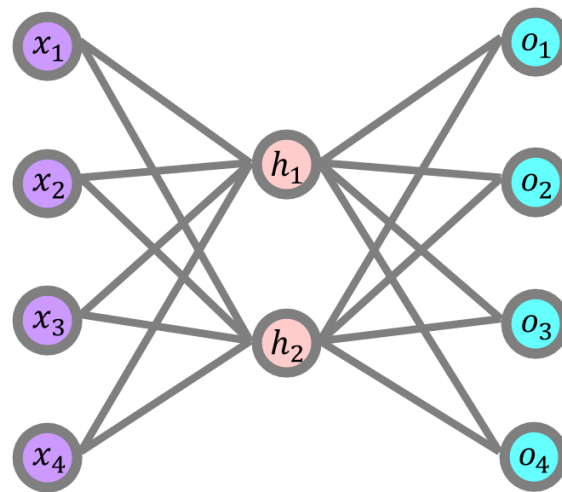


# Deep Learning 101

## Word2Vec

### 쉬운 예제로 개념잡기

One-hot encoding  
Embedding  
CBOW  
Skip-Gram



안녕하세요 여러분! 신박AI입니다

오늘은 자연어 처리(NLP)에서 매우 중요한 개념 중 하나인 Word2Vec에 대해 알아보겠습니다.

Word2Vec은 단어의 벡터 표현을 학습함으로써, NLP에서 광범위하게 적용될 수 있는 강력한 도구입니다.

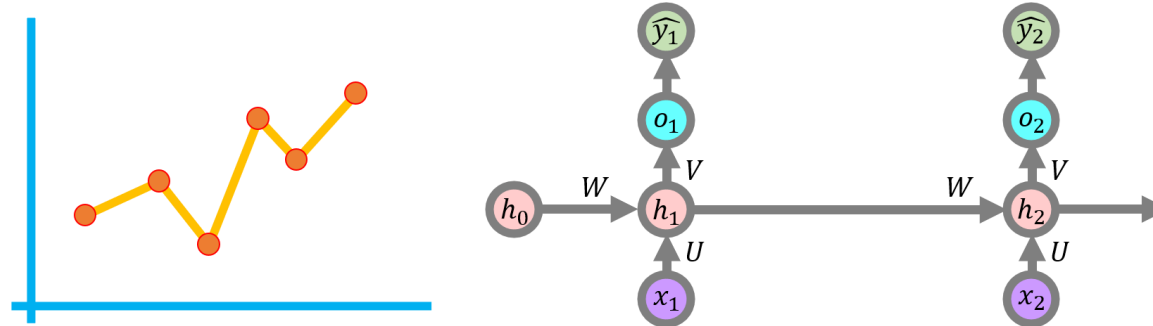
Word2Vec을 이해하기 위해서는 one-hot encoding과 embedding에 대한 기본적인 이해가 필요합니다.

지난 영상에서 잠시 살펴본 바와 같이, one-hot encoding은 단어를 벡터로 표현하는 가장 기본적인 방법 중 하나입니다.

↩ 이 영상 4:19부터 참고하세요~

## Deep Learning101

# RNN (순환신경망) Recurrent Neural Network



예를 들어, *I, love, pizza* 세 단어가 있을 때, 각 단어를 다음과 같이 표현할 수 있습니다.

*I* = [1, 0, 0, 0]

*love* = [0, 1, 0, 0]

*pizza* = [0, 0, 1, 0]

각 단어는 각각의 고유한 벡터로 표현되며, 해당 단어를 나타내는 인덱스만 1이고 나머지는 0입니다.

I = [1, 0, 0, 0]

love = [0, 1, 0, 0]

pizza = [0, 0, 1, 0]



하지만, 이 방법에는 두 가지 큰 단점이 있습니다.

`I = [1,0,0,0]`

`love = [0,1,0,0]`

`pizza= [0,0,1,0]`

첫째, 단어 간의 관계나 유사성을 전혀 표현할 수 없습니다.

I = [1, 0, 0, 0]

you = [0, 0, 0, 1]

love = [0, 1, 0, 0]

pizza = [0, 0, 1, 0]

여기서 보시듯, I 는 pizza보다는 you와 의미적으로 더 가깝지만, 그 차이를 수치로 나타낼 수가 없고

I = [1, 0, 0, 0]

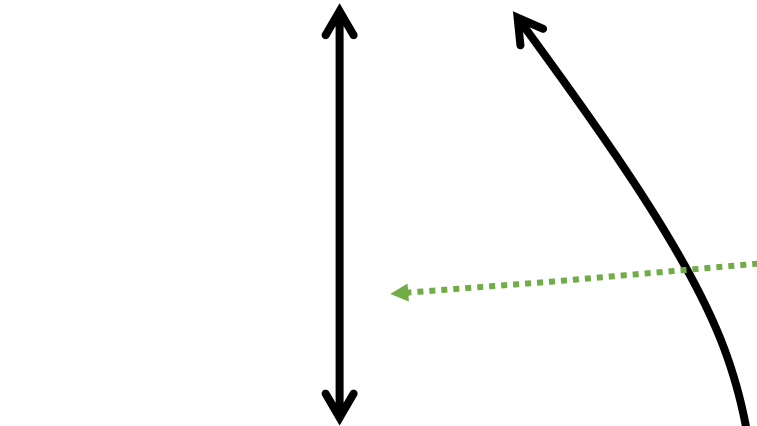
you = [0, 0, 0, 1]

love = [0, 1, 0, 0]

pizza= [0, 0, 1, 0]

여기서 보시듯, I 는 pizza보다는 you와 의미적으로 더 가깝지만, 그 차이를 수치로 나타낼 수가 없고

I = [1, 0, 0, 0]  $\longleftrightarrow$  you = [0, 0, 0, 1]



love = [0, 1, 0, 0]

pizza = [0, 0, 1, 0]

모두 다 1씩 차이가 나기 때문이죠..

둘째, 단어의 집합이 커질 수록 벡터의 차원이 매우 커지며, 이는 공간과 계산 효율성 문제를 야기합니다.

to	$= [1, 0, 0, 0, 0, 0, 0, 0, 0, \dots, 0]$
be	$= [0, 1, 0, 0, 0, 0, 0, 0, 0, \dots, 0]$
or	$= [0, 0, 1, 0, 0, 0, 0, 0, 0, \dots, 0]$
not	$= [0, 0, 0, 1, 0, 0, 0, 0, 0, \dots, 0]$
to	$= [1, 0, 0, 0, 0, 0, 0, 0, 0, \dots, 0]$
be	$= [0, 1, 0, 0, 0, 0, 0, 0, 0, \dots, 0]$
that	$= [0, 0, 0, 0, 1, 0, 0, 0, 0, \dots, 0]$
is	$= [0, 0, 0, 0, 0, 1, 0, 0, 0, \dots, 0]$
the	$= [0, 0, 0, 0, 0, 0, 1, 0, 0, \dots, 0]$
question	$= [0, 0, 0, 0, 0, 0, 0, 1, 0, \dots, 0]$

이러한 문제를 해결하기 위해 등장한 개념이 바로 임베딩embedding  
입니다.

I = [1,0,0,0]      you = [0,0,0,1]

love = [0,1,0,0]

pizza= [0,0,1,0]

임베딩embedding을 한마디로 요약하자면, 단어를 낮은 차원의 실수 벡터로 표현하는 기법입니다.

I = [1, 0, 0, 0]      you = [0, 0, 0, 1]  
[0.1, 0.2]                      [0.1, 0.1]

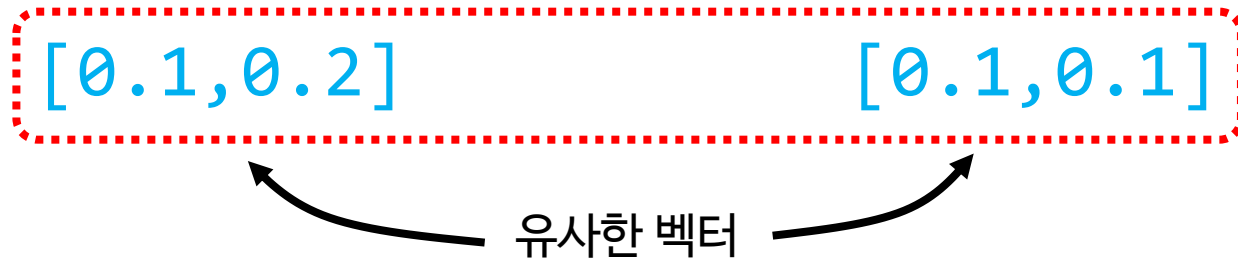
love = [0, 1, 0, 0] [0.7, 0.3]

pizza= [0, 0, 1, 0] [0.5, 0.9]

이렇게 하면, 벡터 공간에서 단어 간의 유사성을 계산할 수 있고,

I =  $[1, 0, 0, 0]$       you =  $[0, 0, 0, 1]$

$[0.1, 0.2]$        $[0.1, 0.1]$



유사한 벡터

love =  $[0, 1, 0, 0]$   $[0.7, 0.3]$

pizza =  $[0, 0, 1, 0]$   $[0.5, 0.9]$



차원이 낮아져서 계산 효율성도 대폭 향상됩니다.

$$\begin{array}{lcl} \text{I} & = & \begin{array}{l} [1, 0, 0, 0] \\ [0.1, 0.2] \end{array} \begin{array}{l} \text{4차원} \\ \downarrow \\ \text{2차원} \end{array} \text{you} & = & \begin{array}{l} [0, 0, 0, 1] \\ [0.1, 0.1] \end{array} \end{array}$$

$$\text{love} = [0, 1, 0, 0] [0.7, 0.3]$$

$$\text{pizza} = [0, 0, 1, 0] [0.5, 0.9]$$

단어 임베딩 기법에는 여러 종류가 있습니다.

대표적으로 Word2Vec, GloVe, FastText 등이 있으며,

Word2Vec

GloVe

FastText

각각의 방법론은 단어를 벡터로 변환하는 메커니즘이 조금씩 다릅니다.

Word2Vec

GloVe

FastText

오늘은 이 중에서도 가장 널리 사용되는 Word2Vec에 대해 다루어 보겠습니다.

Word2Vec

GloVe

FastText


Word2Vec 알고리즘을 간단하게 소개해드리기 위해,

# 4개의 단어만 학습한다고 가정합니다.

I	=	[1,0,0,0]
love	=	[0,1,0,0]
pizza	=	[0,0,1,0]
like	=	[0,0,0,1]

그러면 one-hot encoding 벡터의 크기가 4이기 때문에


I	=	[1,0,0,0]
love	=	[0,1,0,0]
pizza	=	[0,0,1,0]
like	=	[0,0,0,1]

  
4



# 4개의 입력을 받는 입력층을 생각해 볼 수 있습니다.

I	=	[1,0,0,0]
love	=	[0,1,0,0]
pizza	=	[0,0,1,0]
like	=	[0,0,0,1]

  
4

$x_1$

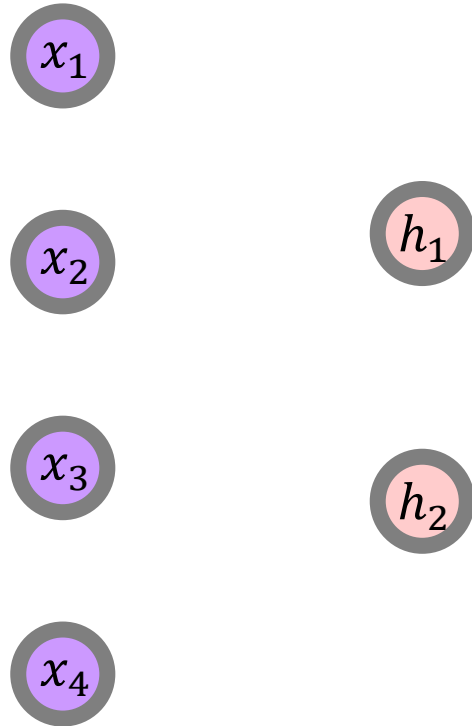
$x_2$

$x_3$

$x_4$

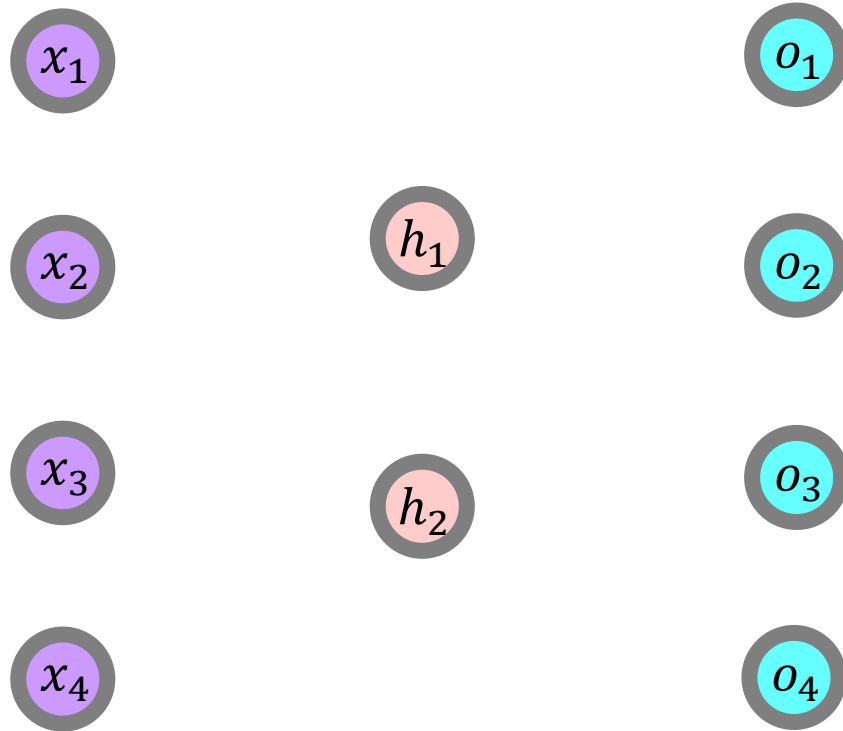
그리고 차원을 축소하고 중요한 특성들을 보존하기 위해 은닉층 노드를 두개를 추가하도록 하겠습니다.

I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]



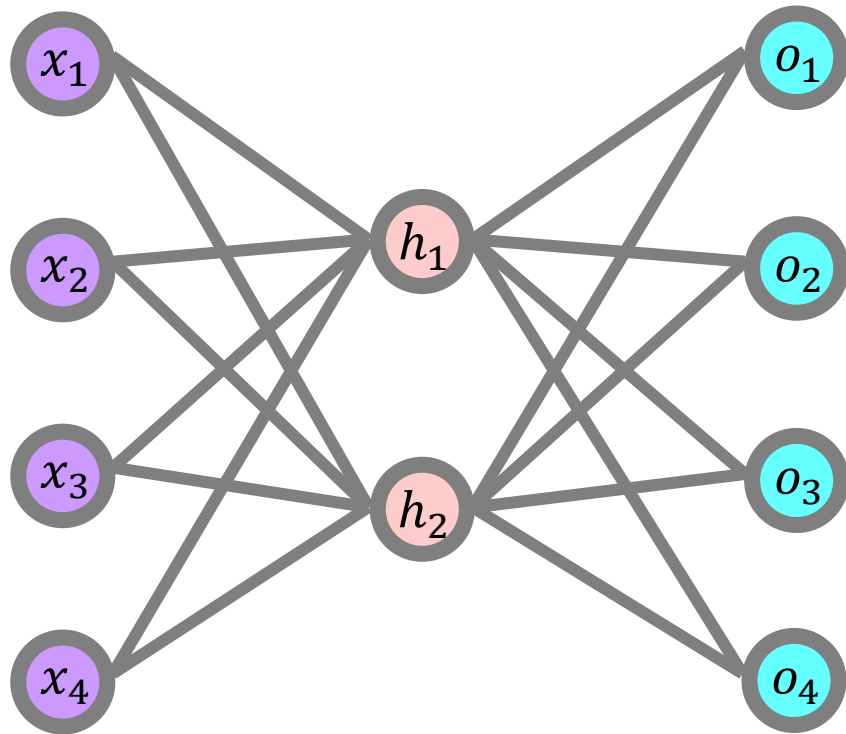
그리고 단어-단어간의 학습을 위해 출력층은 다시 4개의 노드로 구성합니다.

I = [1, 0, 0, 0]  
love = [0, 1, 0, 0]  
pizza = [0, 0, 1, 0]  
like = [0, 0, 0, 1]



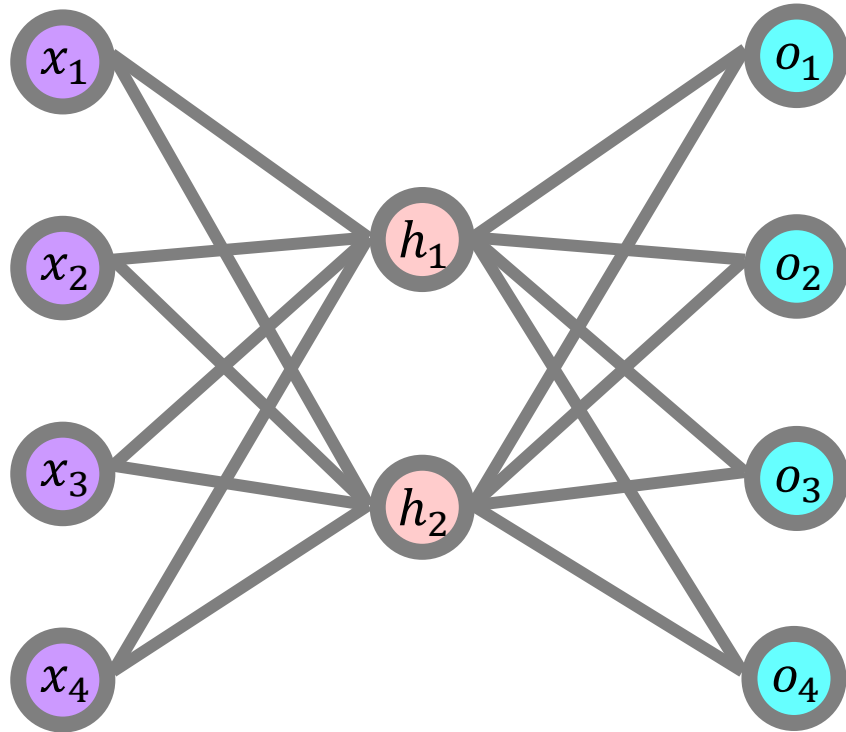
# 이렇게 word2vec 모델을 간략하게 구성할 수 있습니다.

I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]



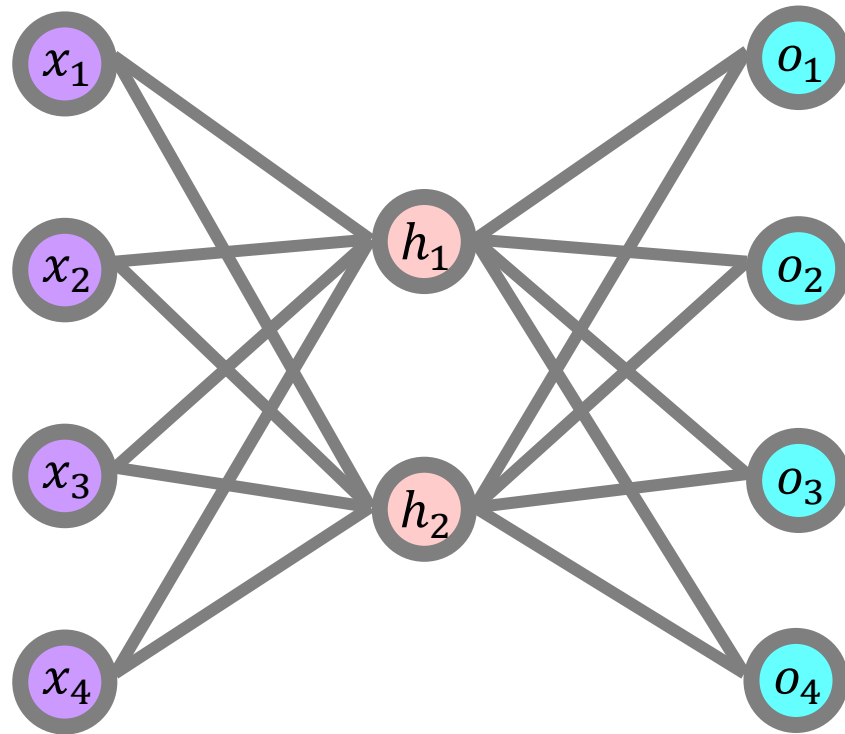
물론 실제 word2vec 모델은 입력층의 크기는 10,000개, 은닉층의 크기는 300개로 훨씬 복잡하지만,

I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]



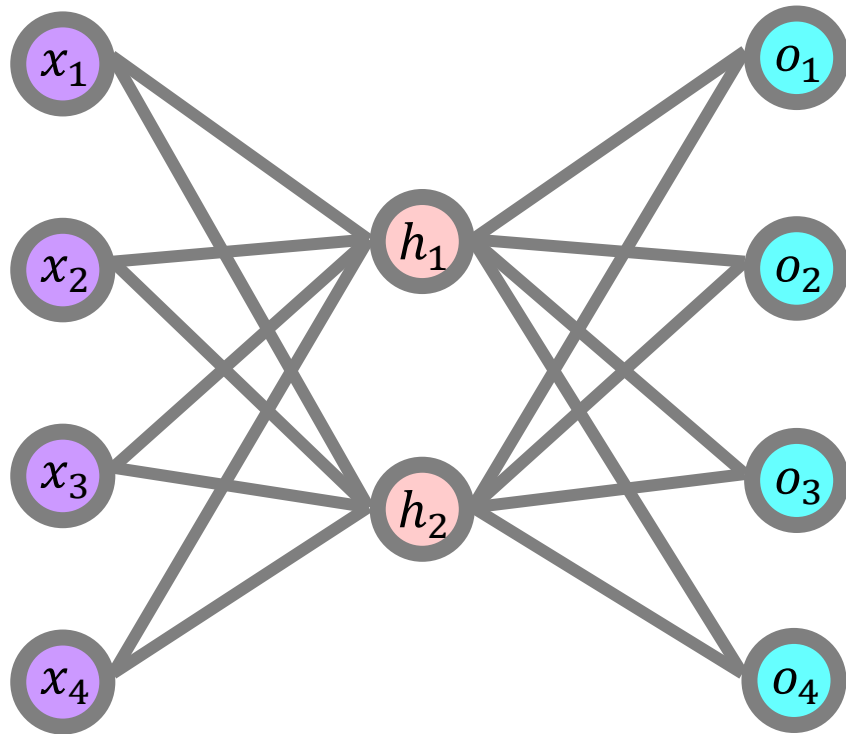
# 입력층-은닉층-출력층 이렇게 세 개의 층으로 이루어진 것은 동일하기 때문에,

I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]



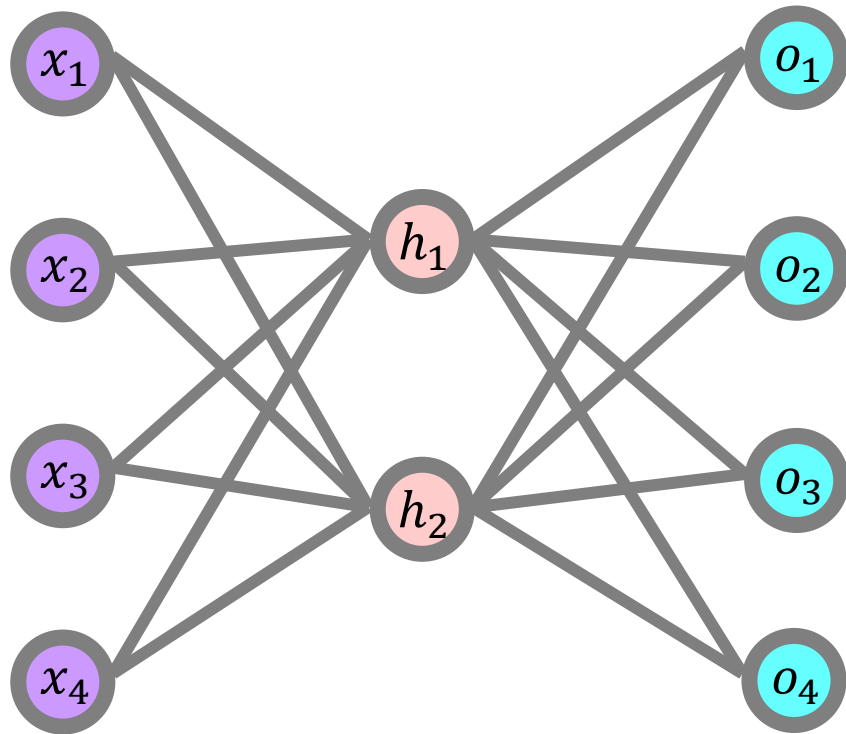
# 지금과 같은 간략화된 모델과 학습데이터로도,

I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]



# Word2vec의 대략적인 학습알고리즘을 살펴보는데는 충분하다고 생각이 듭니다.

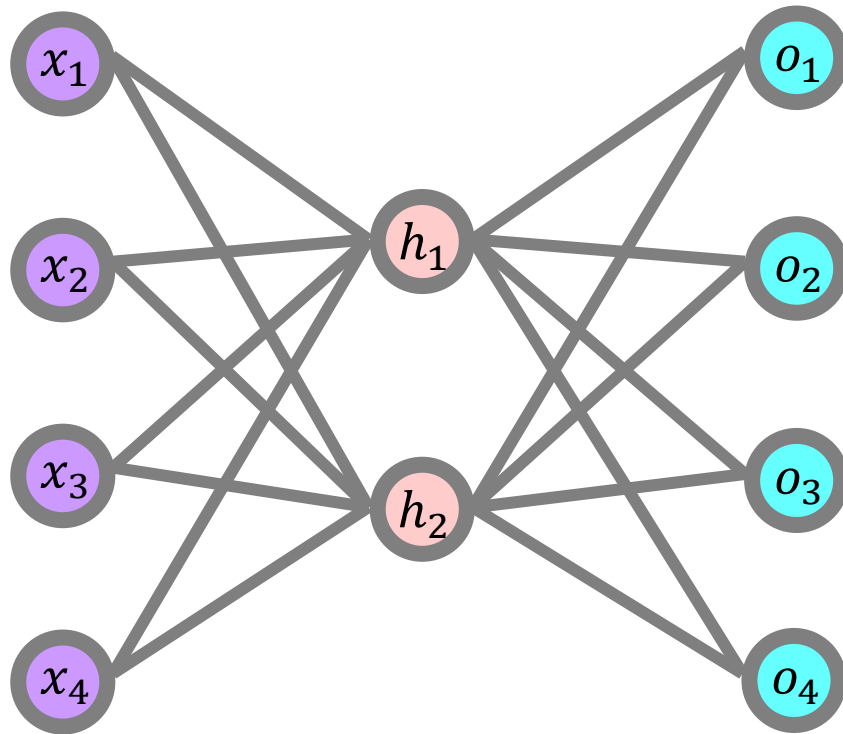
I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]





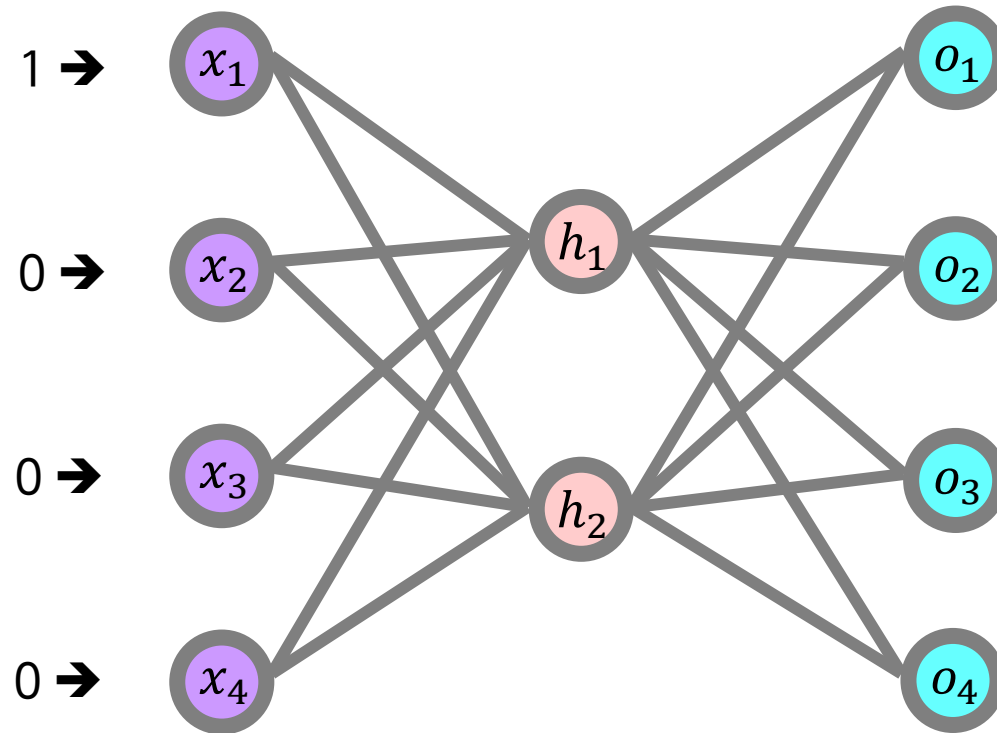
# 모든 신경망의 학습이 그러하듯,

I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]



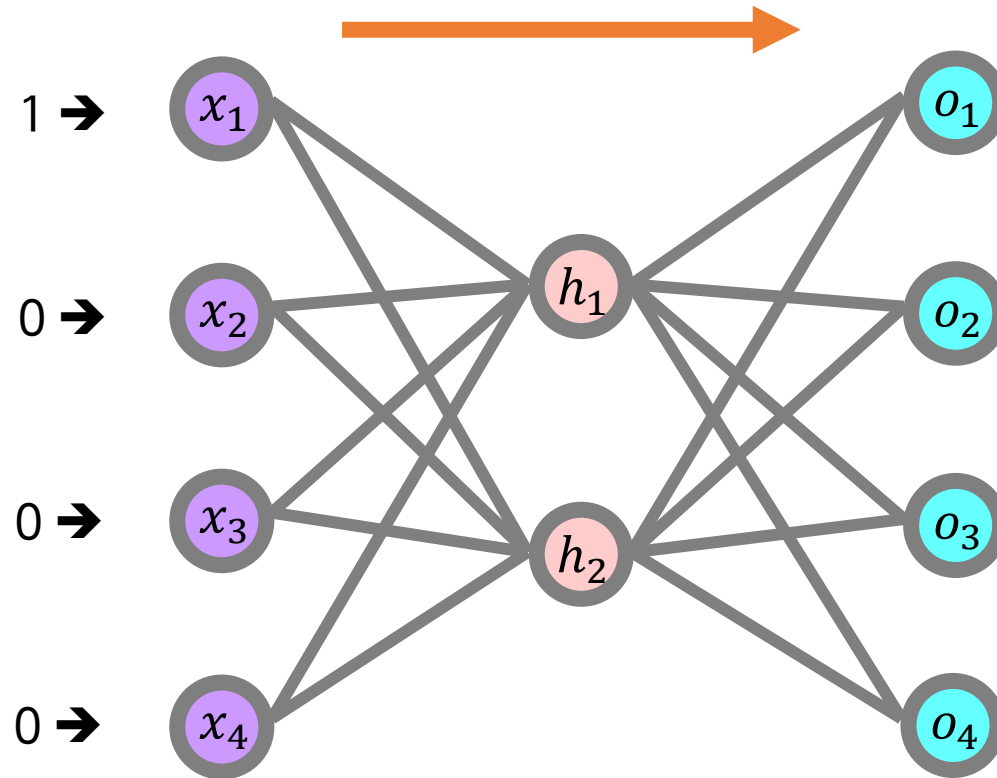
# 입력값을 넣어주고,

I = [1, 0, 0, 0]  
love = [0, 1, 0, 0]  
pizza = [0, 0, 1, 0]  
like = [0, 0, 0, 1]



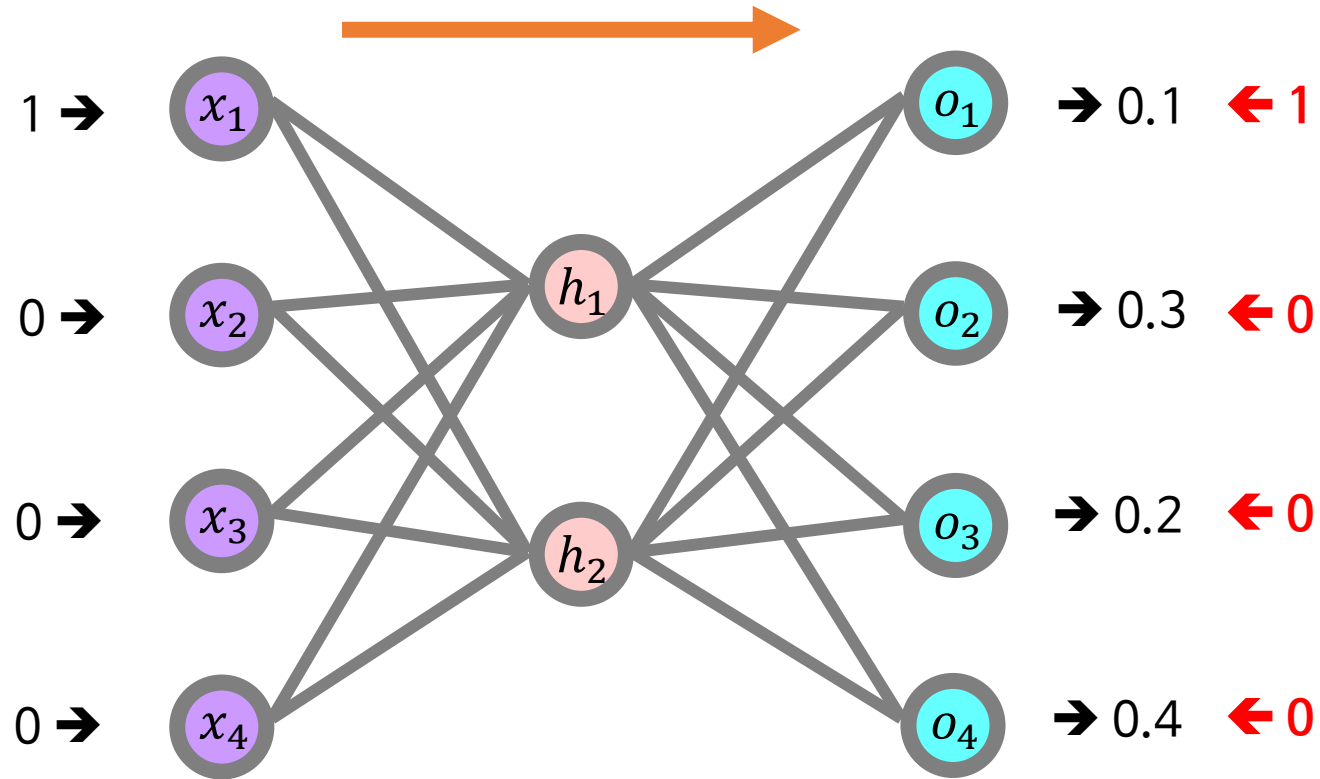
# 순전파 feedforward를 하고,

I = [1, 0, 0, 0]  
love = [0, 1, 0, 0]  
pizza = [0, 0, 1, 0]  
like = [0, 0, 0, 1]



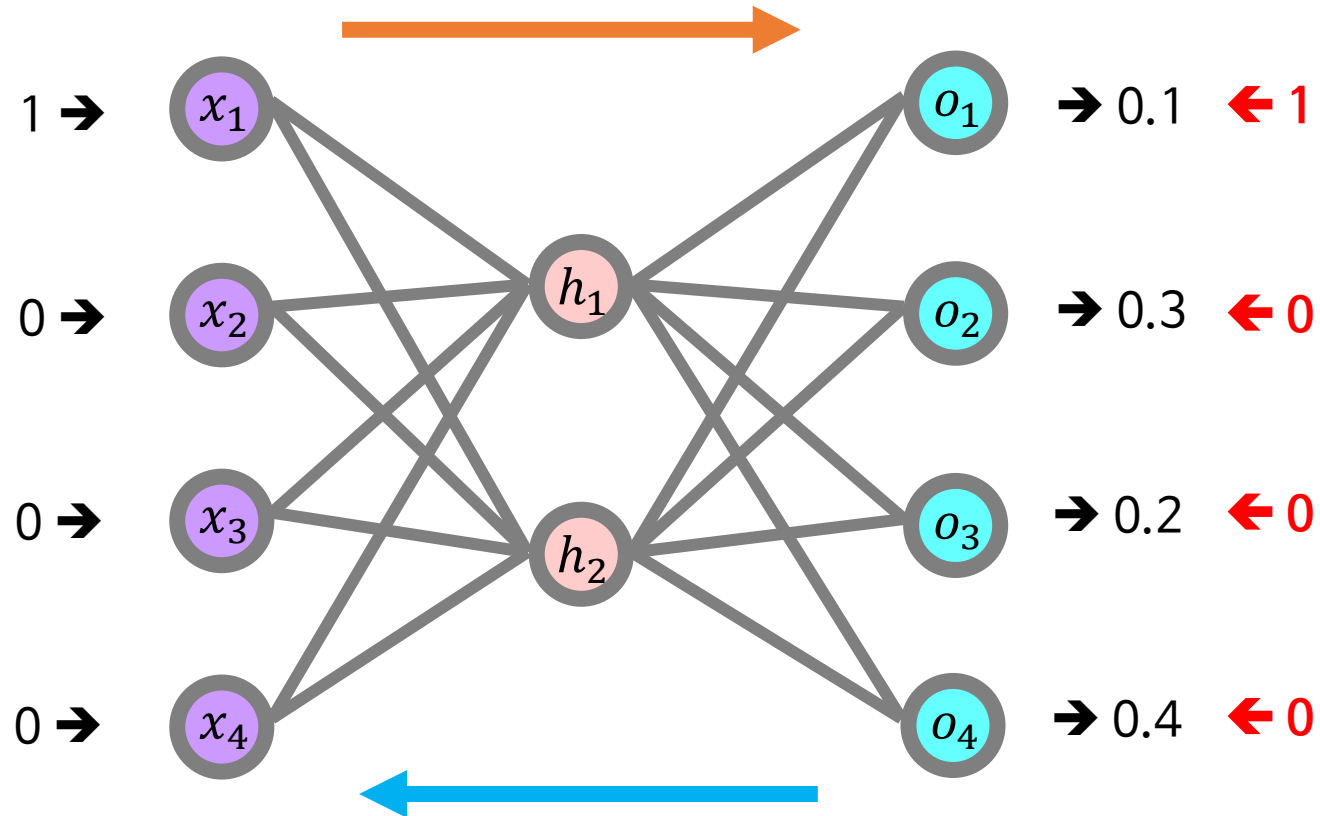
출력값을 계산하여 손실 loss를 구하고,

I = [1, 0, 0, 0]  
love = [0, 1, 0, 0]  
pizza = [0, 0, 1, 0]  
like = [0, 0, 0, 1]



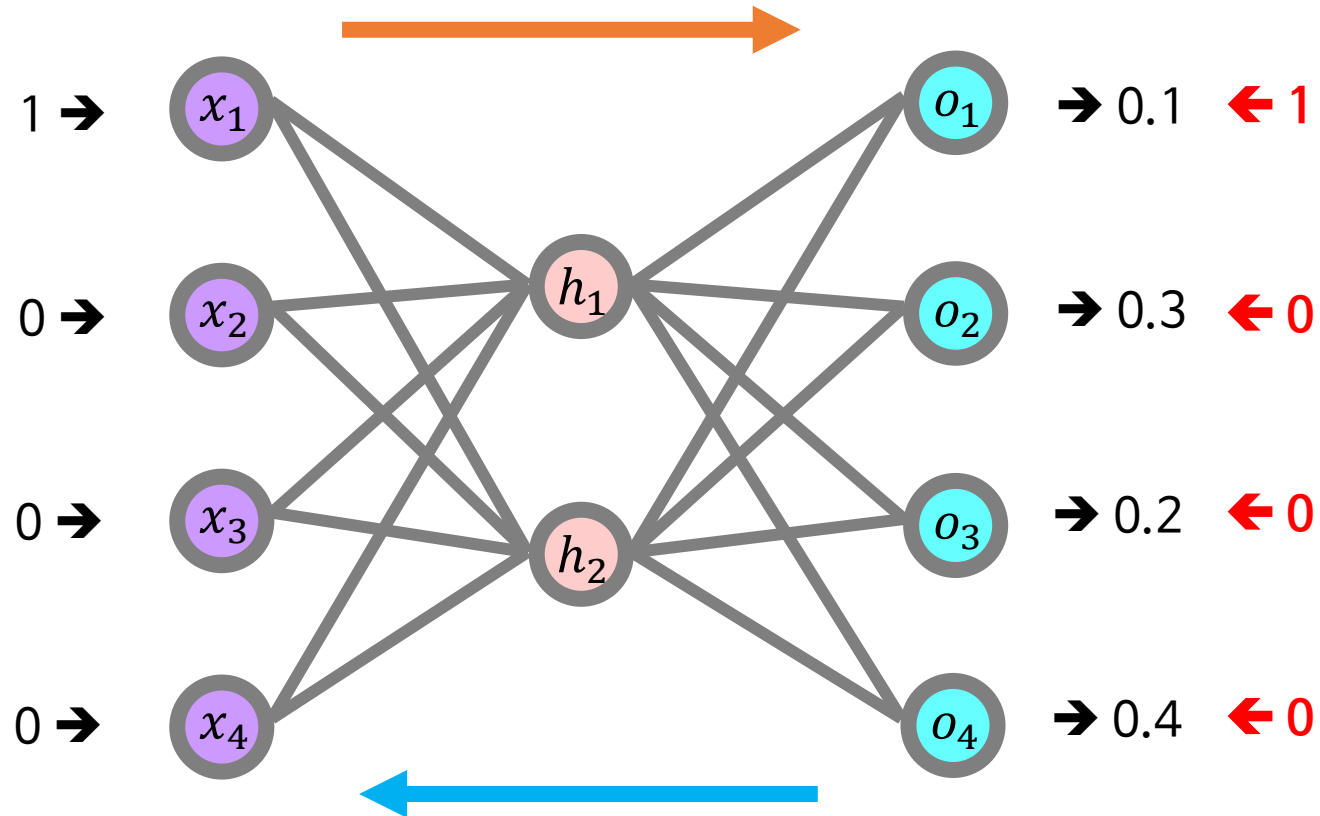
# 경사하강법과 역전파를 사용하여 모델을 학습하는 과정은,

I = [1, 0, 0, 0]  
love = [0, 1, 0, 0]  
pizza = [0, 0, 1, 0]  
like = [0, 0, 0, 1]



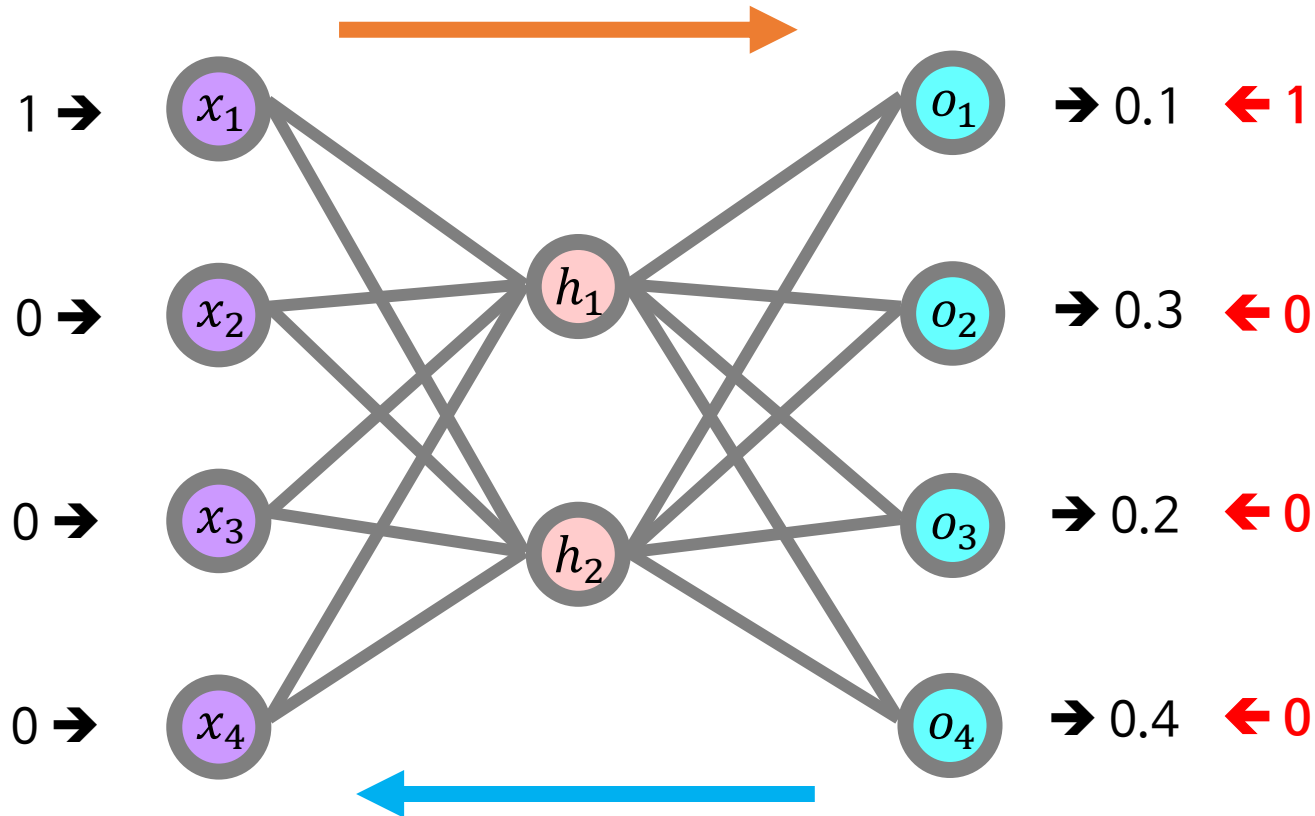
# 거의 동일하다고 보시면 됩니다.

I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]



# 사실 word2vec의 진짜 묘미는 단어 데이터셋을 훈련시키는 방법에 있습니다.

I = [1, 0, 0, 0]  
love = [0, 1, 0, 0]  
pizza = [0, 0, 1, 0]  
like = [0, 0, 0, 1]



Word2vec은 다음 두가지 방법으로 단어 데이터셋을 훈련합니다

Continuous Bag-of-words: CBOW

Skip-Gram



Continuous Bag-of-words: CBOW의 개념은 간단합니다.

# 우리는 언어를 공부할 때 이러한 형태의 빈칸 추론 문제들을 많이 풀어보았습니다.

[31~34] 다음 빈칸에 들어갈 말로 가장 적절한 것을 고르시오.

31. When two cultures come into contact, they do not exchange every cultural item. If that were the case, there would be no cultural differences in the world today. Instead, only a small number of cultural elements ever spread from one culture to another. Which cultural item is accepted depends largely on the item's use and compatibility with already existing cultural traits. For example, it is not likely that men's hair dyes designed to "get out the gray" will spread into parts of rural Africa where a person's status is elevated with advancing years. Even when a(n) \_\_\_\_\_ is consistent with a society's needs, there is still no guarantee that it will be accepted. For example, most people in the United States using US customary units (e.g., inch, foot, yard, mile, etc.) have resisted adopting the metric system even though making such a change would enable US citizens to interface with the rest of the world more efficiently. [3점]

\* metric system: 미터법

- |                  |               |
|------------------|---------------|
| ① categorization | ② innovation  |
| ③ investigation  | ④ observation |
| ⑤ specification  |               |



# 빈칸 추론 문제는 전체적인 문맥context을 통해, 빈칸에 들어갈 가장 적절한 단어를 찾는 문제입니다.

[31~34] 다음 빈칸에 들어갈 말로 가장 적절한 것을 고르시오.

1. When two cultures come into contact, they do not exchange every cultural item. If that were the case, there would be no cultural differences in the world today. Instead, only a small number of cultural elements ever spread from one culture to another. Which cultural item is accepted depends largely on the item's use and compatibility with already existing cultural traits. For example, it is not likely that men's hair dyes designed to "get out the gray" will spread into parts of rural Africa where a person's status is elevated with advancing years. Even when a (1) \_\_\_\_\_ is consistent with a society's needs, there is still no guarantee that it will be accepted. For example, most people in the United States using US customary units (e.g., inch, foot, yard, mile, etc.) have resisted adopting the metric system even though making such a change would enable US citizens to interface with the rest of the world more efficiently. [3점]

문맥Context

?

\* metric system: 미터법

- ① categorization
- ③ investigation
- ⑤ specification

- ② innovation
- ④ observation

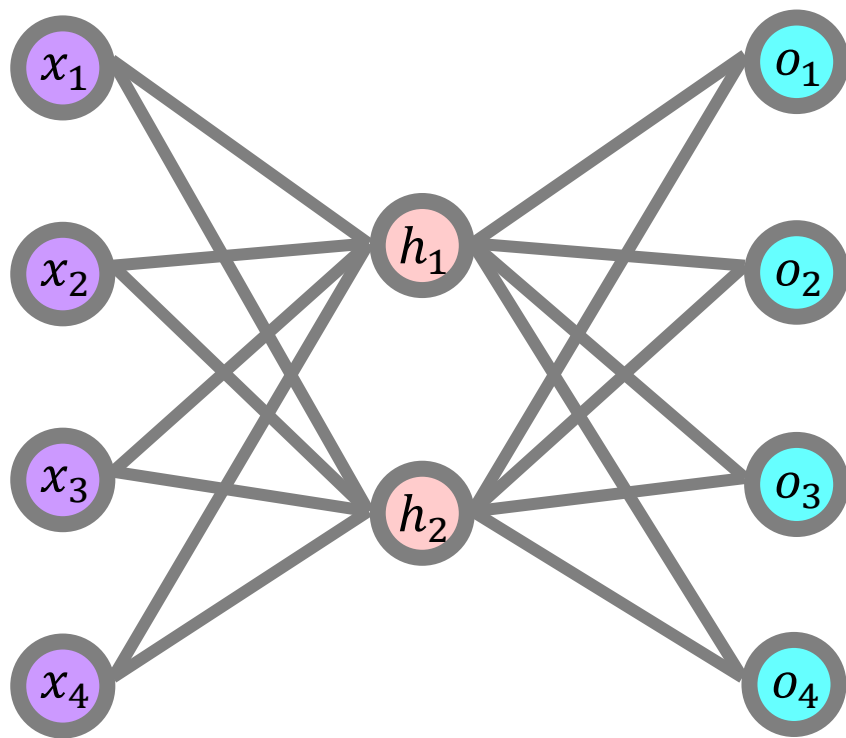


이와 같이, CBOW는 전후 문맥을 고려하여 중간에 들어갈 단어를 학습하는 과정을 말합니다.

I Love pizza  
You Love pizza  
I Love chicken  
I Love hamburger  
You Love chicken  
⋮  
I      chicken

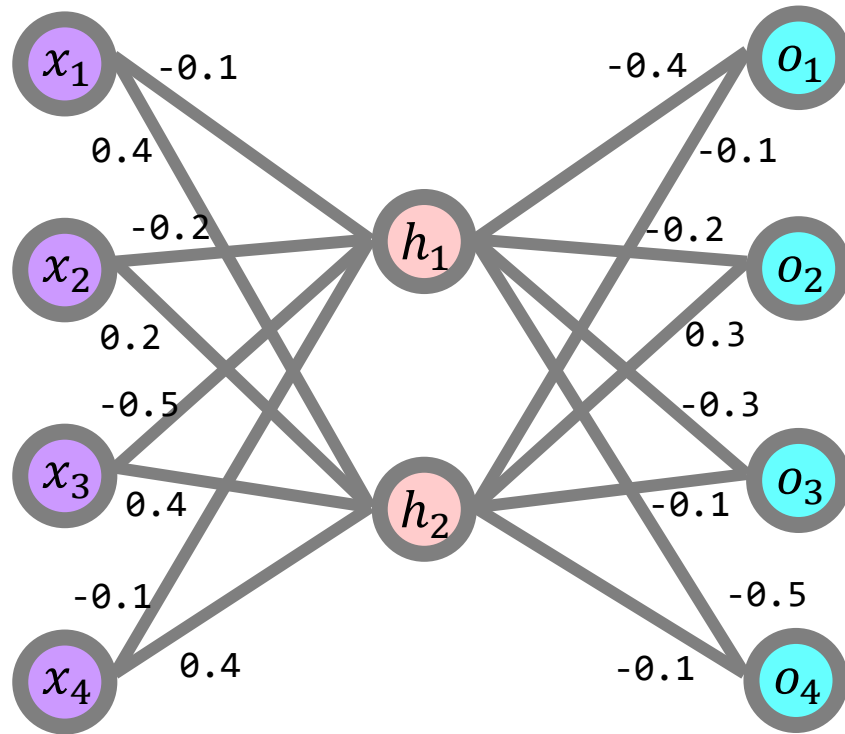
이러한 CBOW의 학습 과정을 모델을 이용, 숫자를 통해 알아보도록 하겠습니다.

I = [1, 0, 0, 0]  
love = [0, 1, 0, 0]  
pizza = [0, 0, 1, 0]  
like = [0, 0, 0, 1]



우선 모델의 가중치들을 다음과 같이 임의 초기화 하도록 하겠습니다.

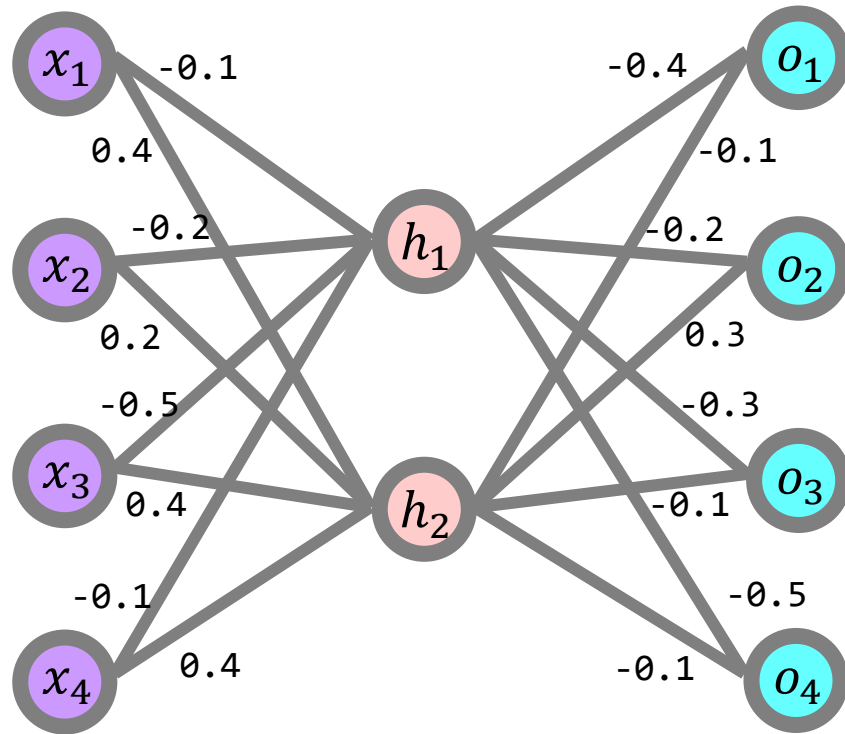
I        =  $[1, 0, 0, 0]$   
love    =  $[0, 1, 0, 0]$   
pizza   =  $[0, 0, 1, 0]$   
like    =  $[0, 0, 0, 1]$



만약 다음과 같은 문장을 학습시킨다면,

I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]

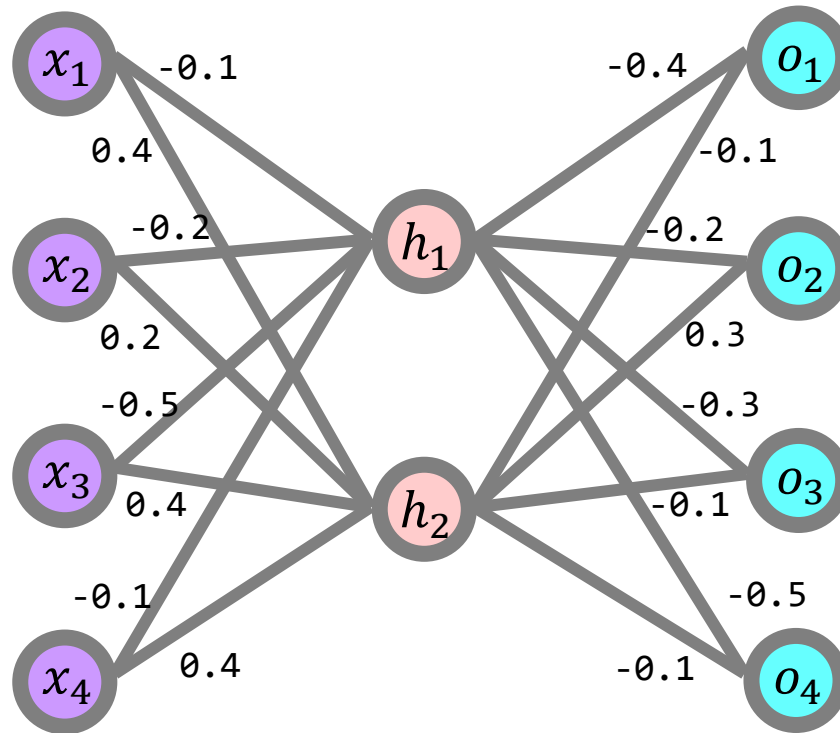
I love pizza



# I와 Pizza는 입력값이 되고,

I = [1, 0, 0, 0]  
love = [0, 1, 0, 0]  
pizza = [0, 0, 1, 0]  
like = [0, 0, 0, 1]

I love pizza

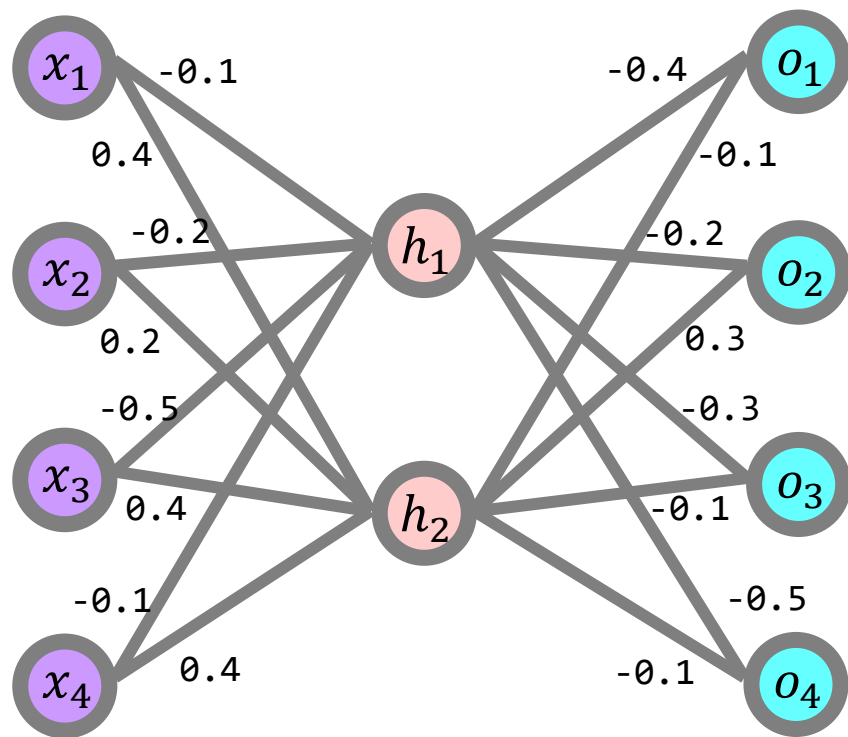




# Love는 정답이 되어야 합니다.

I = [1, 0, 0, 0]  
love = [0, 1, 0, 0]  
pizza = [0, 0, 1, 0]  
like = [0, 0, 0, 1]

I love pizza



# 그것을 숫자로 표현해 보도록 하겠습니다.

I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]

I love pizza

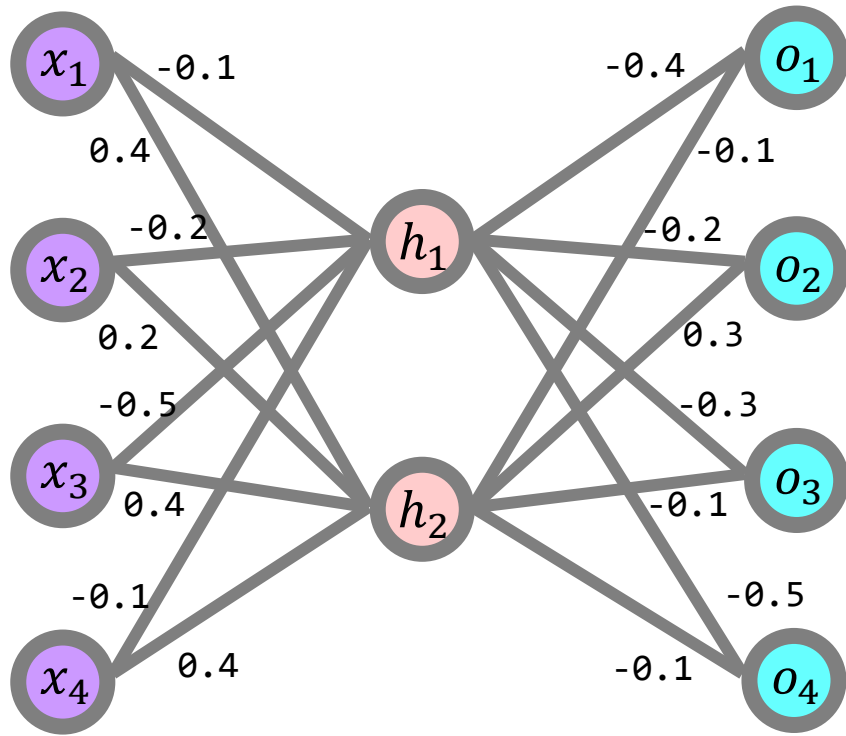
I pizza

1 0

0 0

0 1

0 0



love

0

1

0

0



우선 I는 [1,0,0,0]이므로 이렇게 입력을 할 수 있고,

I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]

I love pizza

I  
1  
0  
0  
0

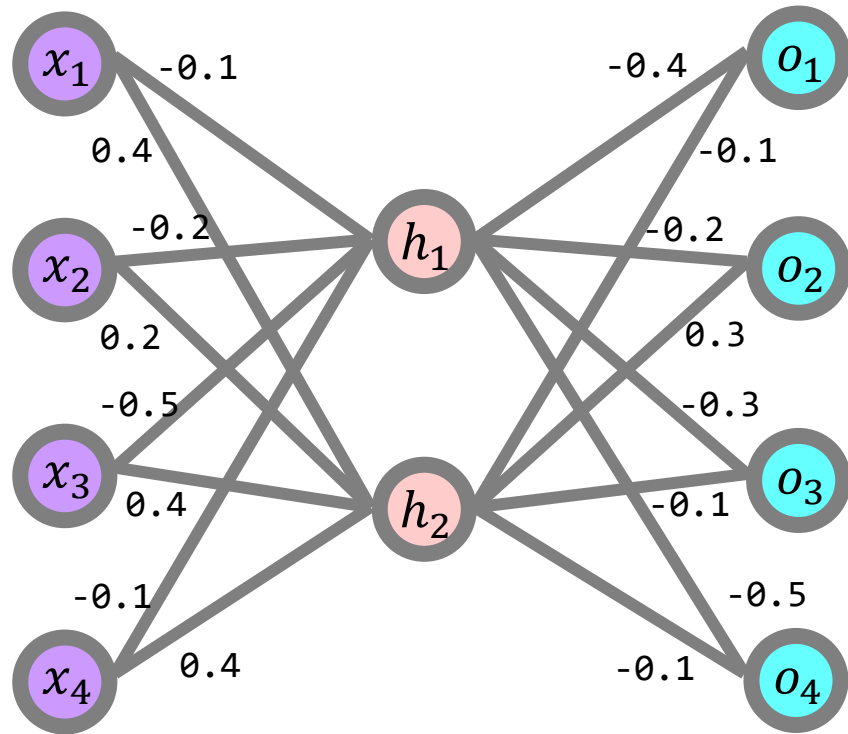
pizza

0

0

1

0



love

0

1

0

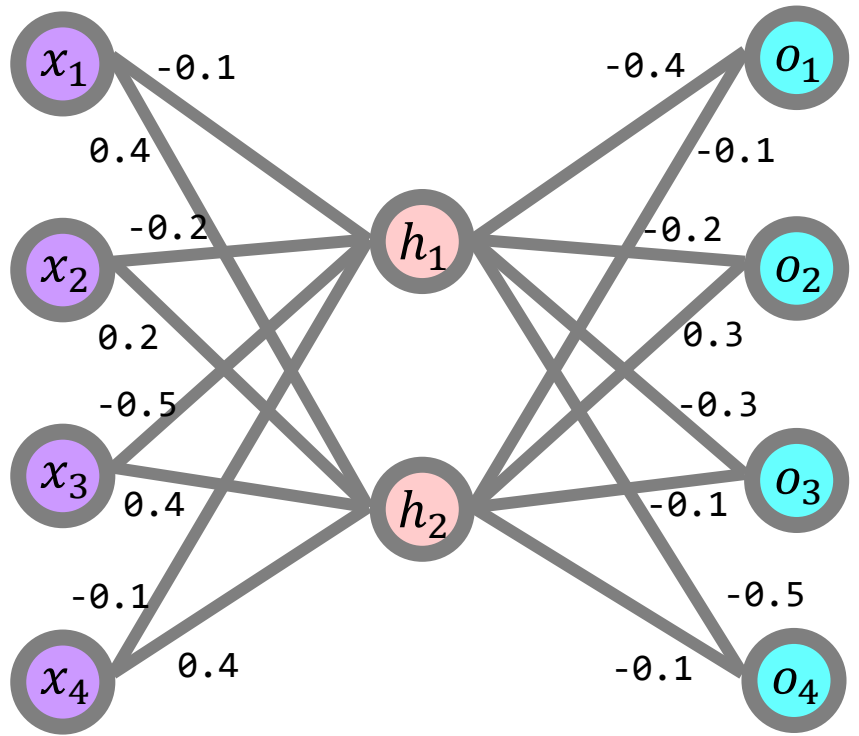
0

또 pizza는 [0,0,1,0] 이므로 이렇게 입력을 할 수 있습니다.

I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]

I love pizza

I	pizza
1	0
0	0
0	1
0	0



love
0
1
0
0

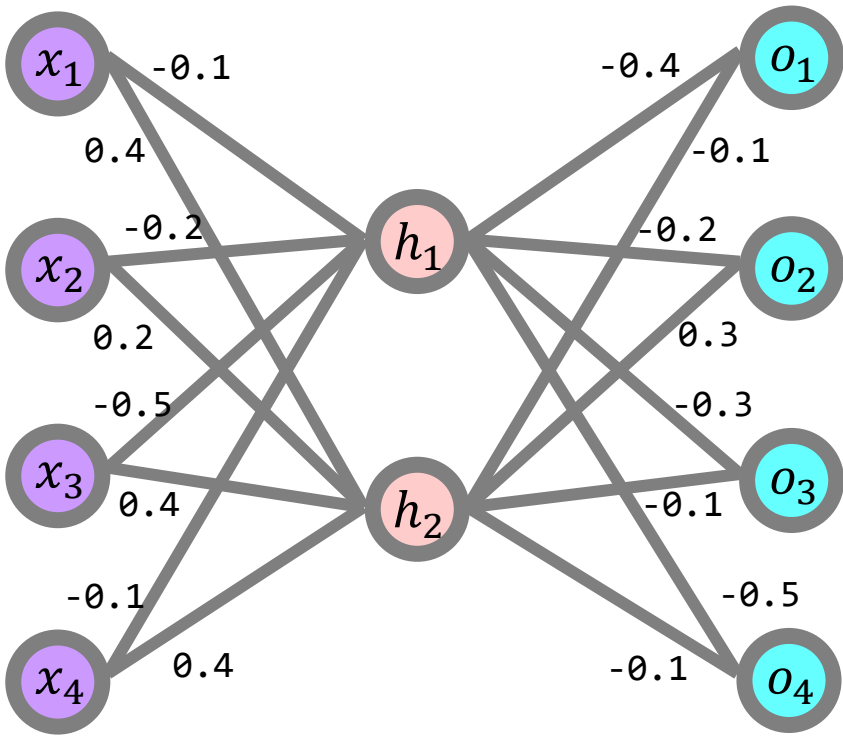
그러므로 이 둘을 같이 고려한 입력은 다음과 같이 됩니다.

I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]

I love pizza

I pizza

1	0	1
0	0	0
0	1	1
0	0	0



love

0
1
0
0

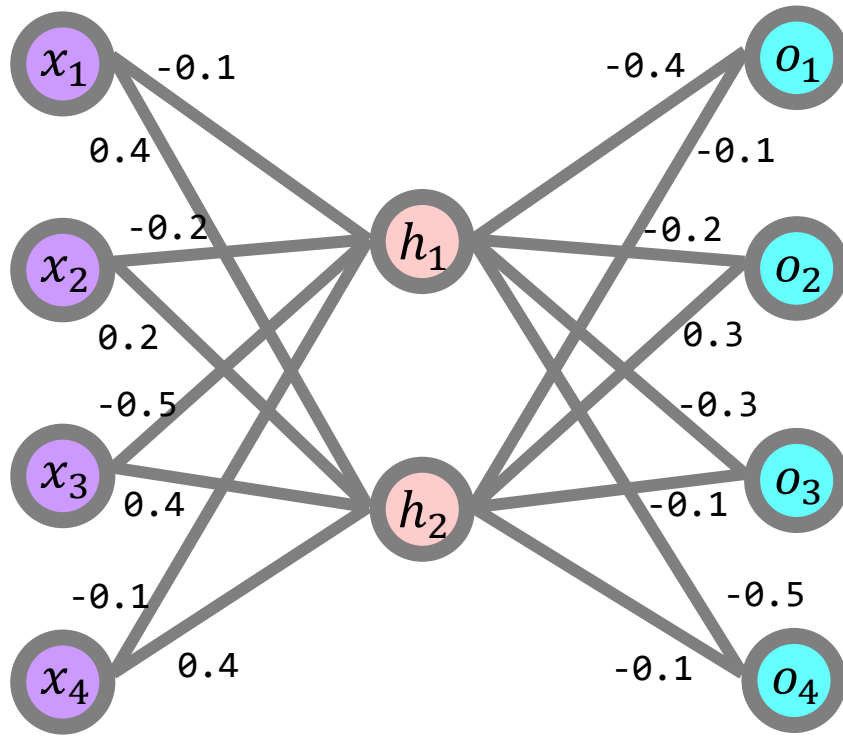
하지만 입력을 일종의 확률분포로 본다면, 입력 벡터의 합은 1이 되어야  
하므로,

I = [1, 0, 0, 0]  
love = [0, 1, 0, 0]  
pizza = [0, 0, 1, 0]  
like = [0, 0, 0, 1]

I love pizza

I pizza

1	0	1
0	0	0
0	1	1
0	0	0

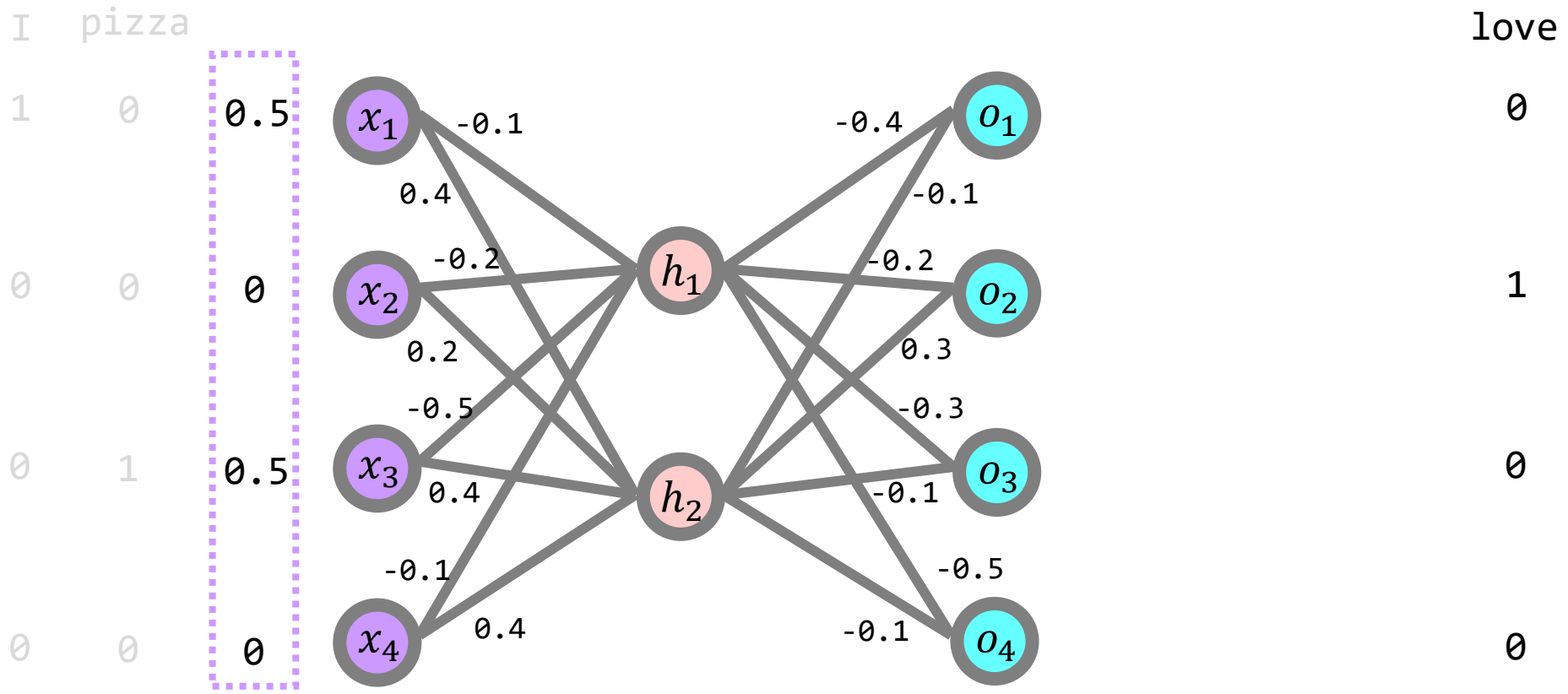


love

0
1
0
0

# 전체를 2로 나누도록 하겠습니다.

I = [1,0,0,0]      I love pizza  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]



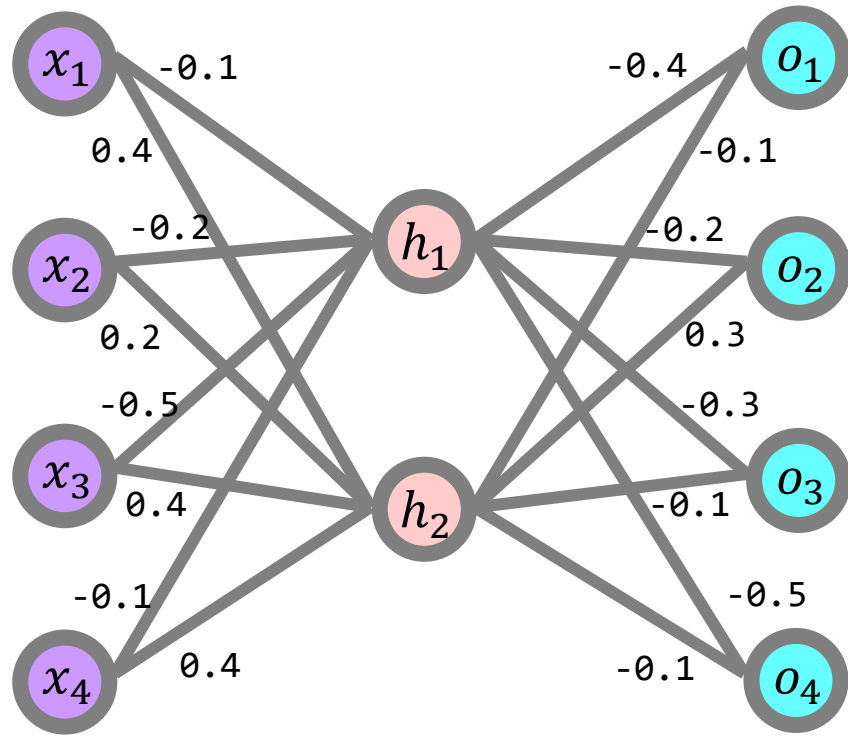
그 다음 입력값과 가중치를 곱하면 (순전파),

I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]

I love pizza

I pizza

1	0	0.5
0	0	0
0	1	0.5
0	0	0



love

0  
1  
0  
0



은닉층 값  $h_1$  과  $h_2$  는 다음과 같이 계산할 수 있습니다.

I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]

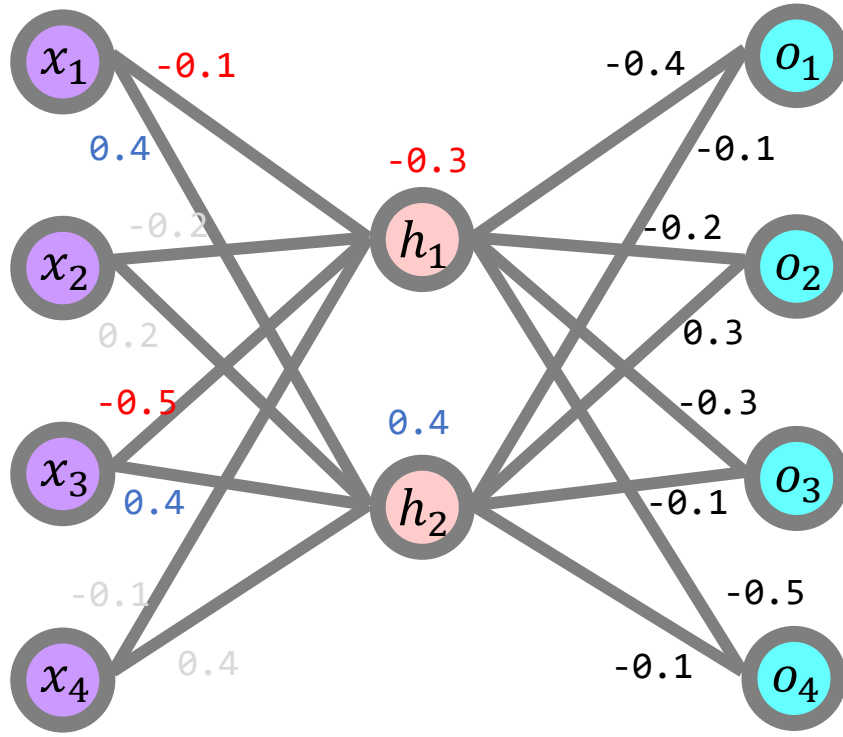
I love pizza

$$h_1 = 0.5 \times (-0.1) + 0.5 \times (-0.5) = -0.3$$

$$h_2 = 0.5 \times 0.4 + 0.5 \times 0.4 = 0.4$$

I pizza

1	0	0.5
0	0	0
0	1	0.5
0	0	0



love

0  
1  
0  
0

여기서 한가지 확인해두어야 할 부분은, word2vec에서는 은닉층에서 비선형 활성화 함수 (예를들면 시그모이드함수)를 쓰지 않습니다.

I = [1,0,0,0]  
 love = [0,1,0,0]  
 pizza = [0,0,1,0]  
 like = [0,0,0,1]

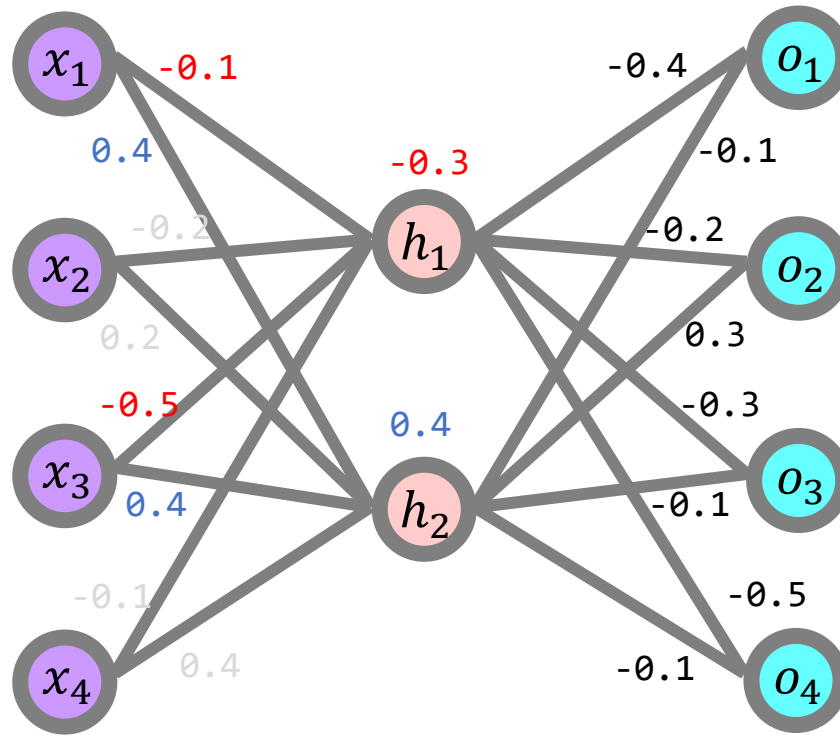
I love pizza

$$h_1 = 0.5 \times (-0.1) + 0.5 \times (-0.5) = -0.3$$

$$h_2 = 0.5 \times 0.4 + 0.5 \times 0.4 = 0.4$$

I pizza

1	0	0.5
0	0	0
0	1	0.5
0	0	0



love

0  
1  
0  
0

# Word2vec을 처음 제안한 Mikolov et al.의 논문에서 왜 활성화 함수를 쓰지 않았는가에 관한 자세한 설명을 저는 찾지 못했습니다만,

I = [1, 0, 0, 0]  
love = [0, 1, 0, 0]  
pizza = [0, 0, 1, 0]  
like = [0, 0, 0, 1]

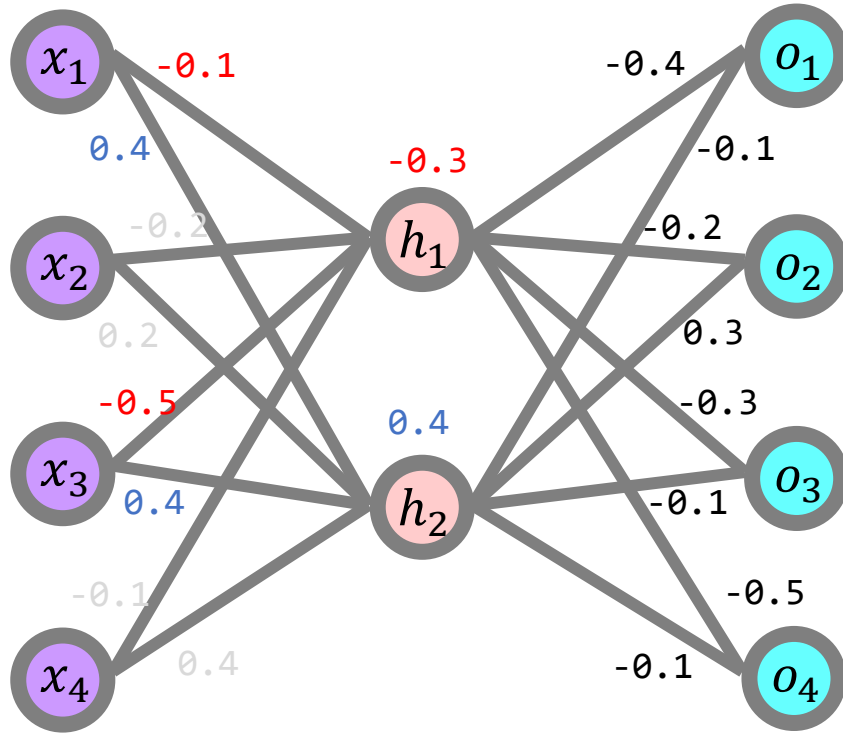
I love pizza

$$h_1 = 0.5 \times (-0.1) + 0.5 \times (-0.5) = -0.3$$

$$h_2 = 0.5 \times 0.4 + 0.5 \times 0.4 = 0.4$$

I pizza

1	0	0.5
0	0	0
0	1	0.5
0	0	0



love

0  
1  
0  
0

# 여러가지 가능성을 생각해 볼 수는 있을 것 같습니다.

I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]

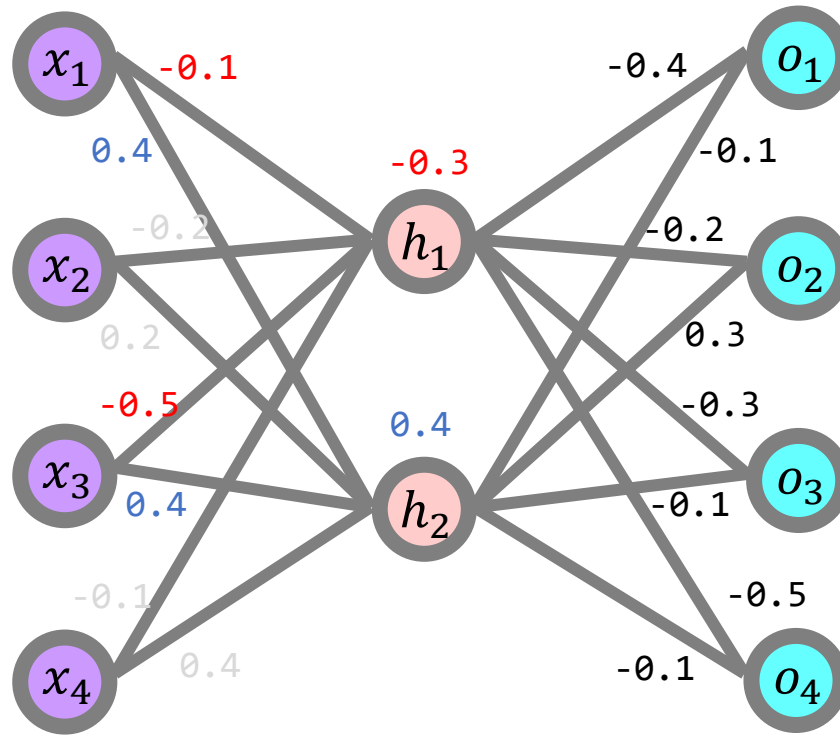
I love pizza

$$h_1 = 0.5 \times (-0.1) + 0.5 \times (-0.5) = -0.3$$

$$h_2 = 0.5 \times 0.4 + 0.5 \times 0.4 = 0.4$$

I pizza

1	0	0.5
0	0	0
0	1	0.5
0	0	0



love

0  
1  
0  
0

# 무엇보다도 방대한 양의 단어 데이터를 처리하기 위한 효율성과 단순성 때문일 수도 있고,

I = [1, 0, 0, 0]  
love = [0, 1, 0, 0]  
pizza = [0, 0, 1, 0]  
like = [0, 0, 0, 1]

I love pizza

$$h_1 = 0.5 \times (-0.1) + 0.5 \times (-0.5) = -0.3$$

$$h_2 = 0.5 \times 0.4 + 0.5 \times 0.4 = 0.4$$

I pizza

1 0

0.5

0 0

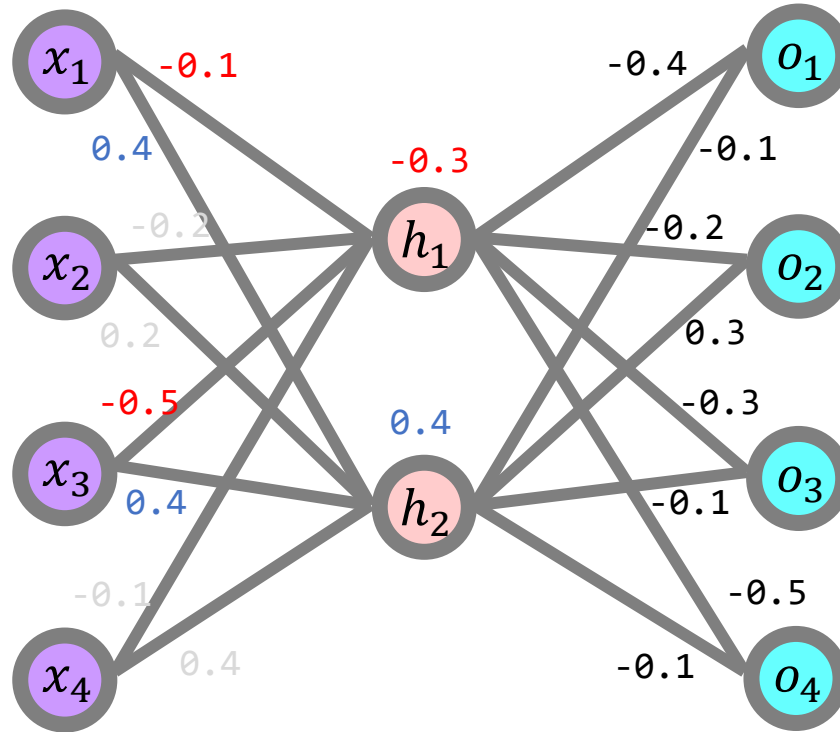
0

0 1

0.5

0 0

0



love

0

1

0

0

비선형 함수를 추가함으로서 단어와 단어들 사이에 불필요한 왜곡을 주지 않기 위함이라고 저는 생각합니다.

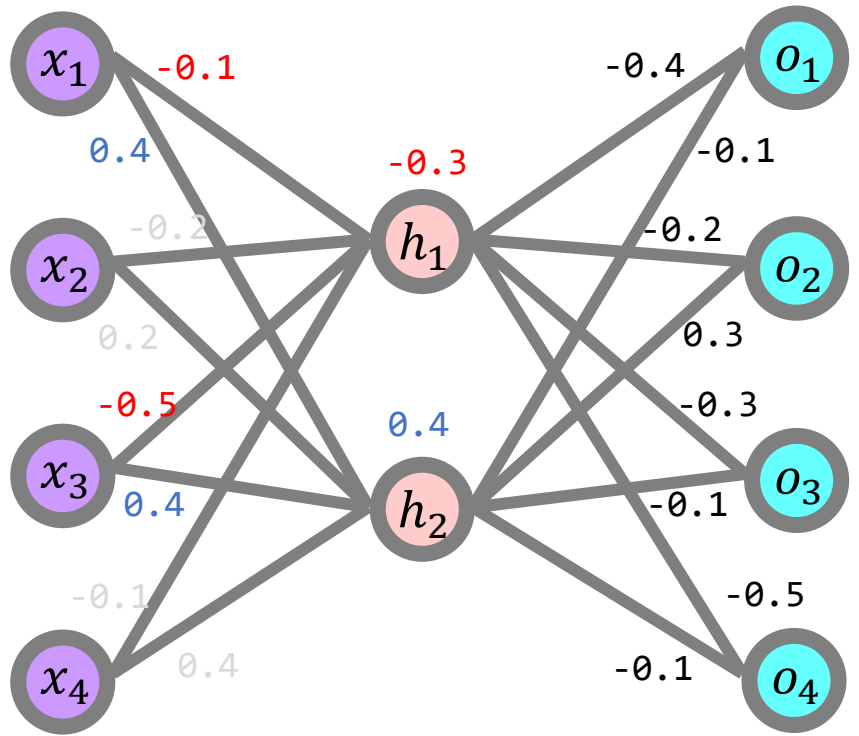
I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]

I love pizza

$$h_1 = 0.5 \times (-0.1) + 0.5 \times (-0.5) = -0.3$$
$$h_2 = 0.5 \times 0.4 + 0.5 \times 0.4 = 0.4$$

I pizza

1	0	0.5
0	0	0
0	1	0.5
0	0	0



love

0  
1  
0  
0

왜냐하면, word2vec 모델의 목적이 어떤 복잡한 데이터들 간에 차이점을 비선형적으로 확대하여 분류하려는 목적이 아니라,

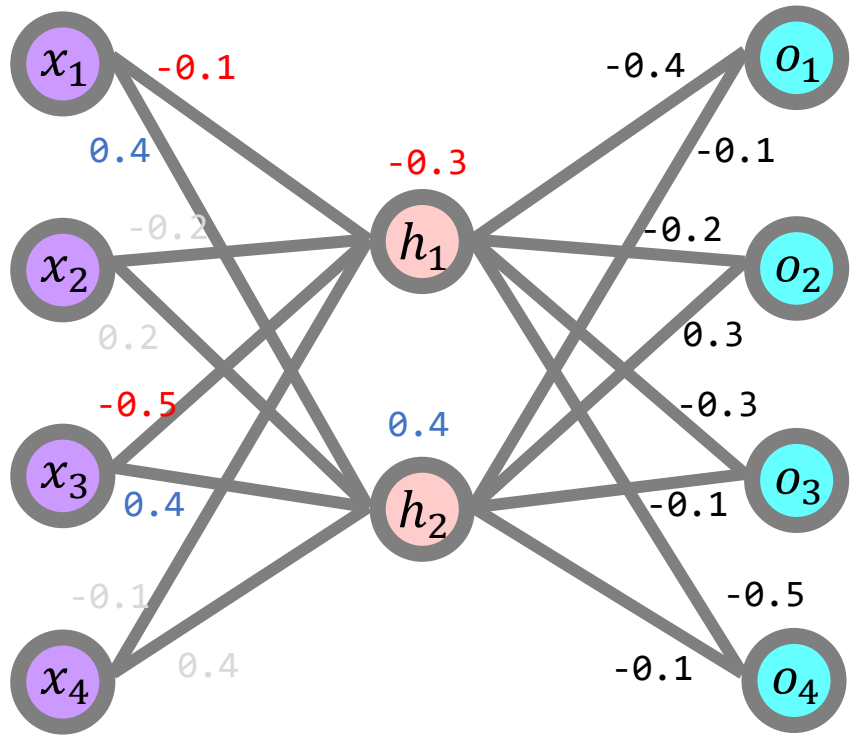
I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]

I love pizza

$$h_1 = 0.5 \times (-0.1) + 0.5 \times (-0.5) = -0.3$$
$$h_2 = 0.5 \times 0.4 + 0.5 \times 0.4 = 0.4$$

I pizza

1	0	0.5
0	0	0
0	1	0.5
0	0	0



love

0  
1  
0  
0

단어들 간의 정확한 관계성을 보다 낮은 차원의 공간에 표현하고자 하는  
것이기 때문에,

I = [1, 0, 0, 0]  
love = [0, 1, 0, 0]  
pizza = [0, 0, 1, 0]  
like = [0, 0, 0, 1]

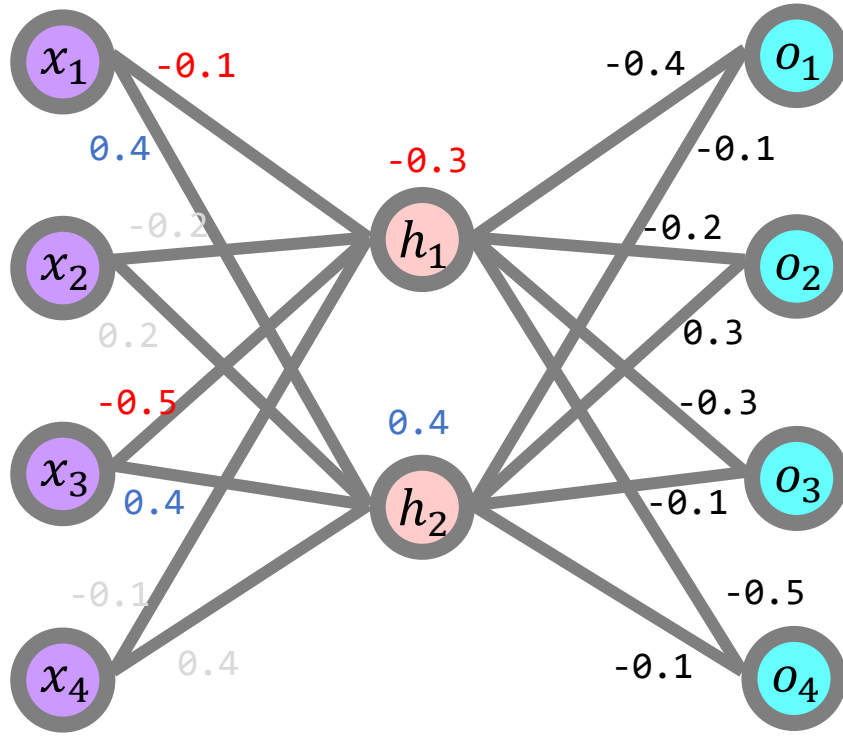
I love pizza

$$h_1 = 0.5 \times (-0.1) + 0.5 \times (-0.5) = -0.3$$

$$h_2 = 0.5 \times 0.4 + 0.5 \times 0.4 = 0.4$$

I pizza

1	0	0.5
0	0	0
0	1	0.5
0	0	0



love

0  
1  
0  
0



모든 단어들의 관계가 비선형에 의해 왜곡되지 않고 공평하게 하는 것이 중요하지 않았나 그런 생각이 듭니다.

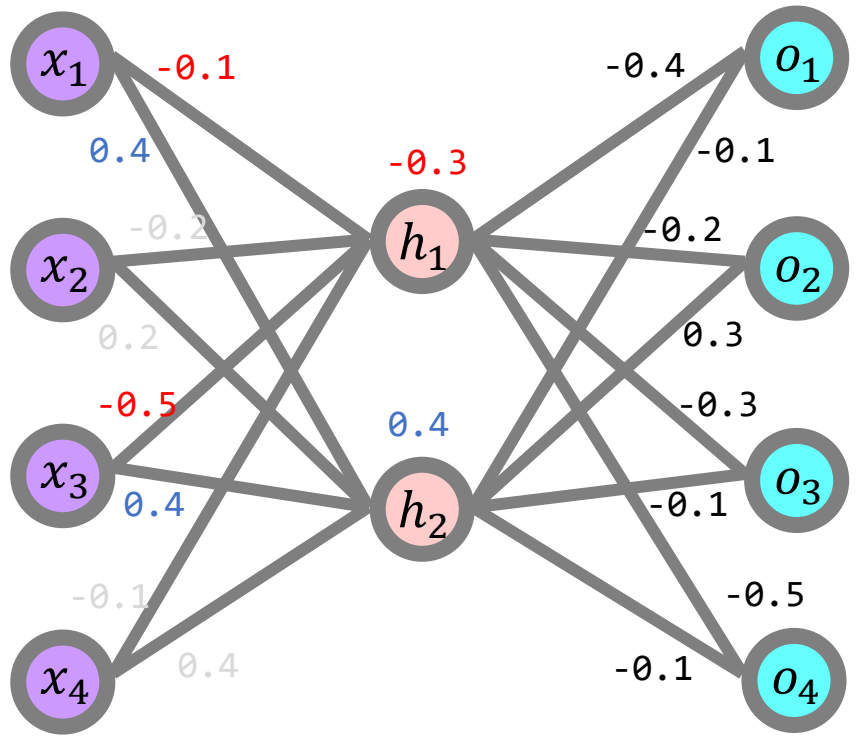
I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]

I love pizza

$$h_1 = 0.5 \times (-0.1) + 0.5 \times (-0.5) = -0.3$$
$$h_2 = 0.5 \times 0.4 + 0.5 \times 0.4 = 0.4$$

I pizza

1	0	0.5
0	0	0
0	1	0.5
0	0	0



love

0
1
0
0

# 아무튼 계속해서 출력층 값도 다음과 같이 계산할 수 있습니다.

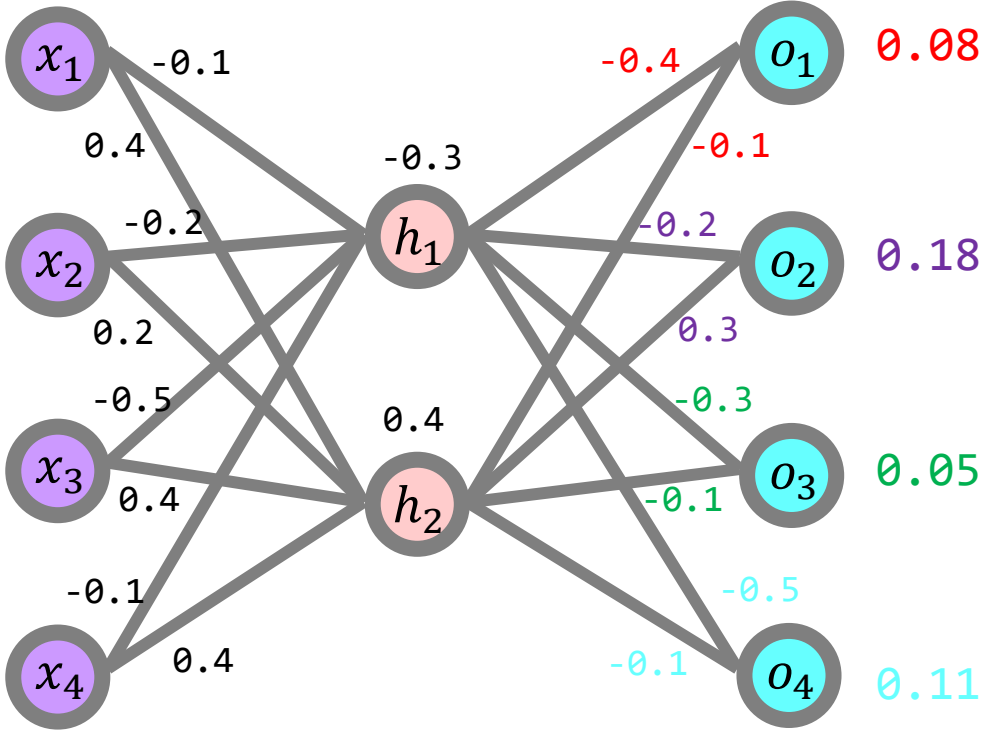
I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]

I love pizza

$$o_1 = (-0.3) \times (-0.4) + 0.4 \times (-0.1) = 0.08$$
$$o_2 = (-0.3) \times (-0.2) + 0.4 \times 0.3 = 0.18$$
$$o_3 = (-0.3) \times (-0.3) + 0.4 \times (-0.1) = 0.05$$
$$o_4 = (-0.3) \times (-0.5) + 0.4 \times (-0.1) = 0.11$$

I pizza

1	0	0.5
0	0	0
0	1	0.5
0	0	0



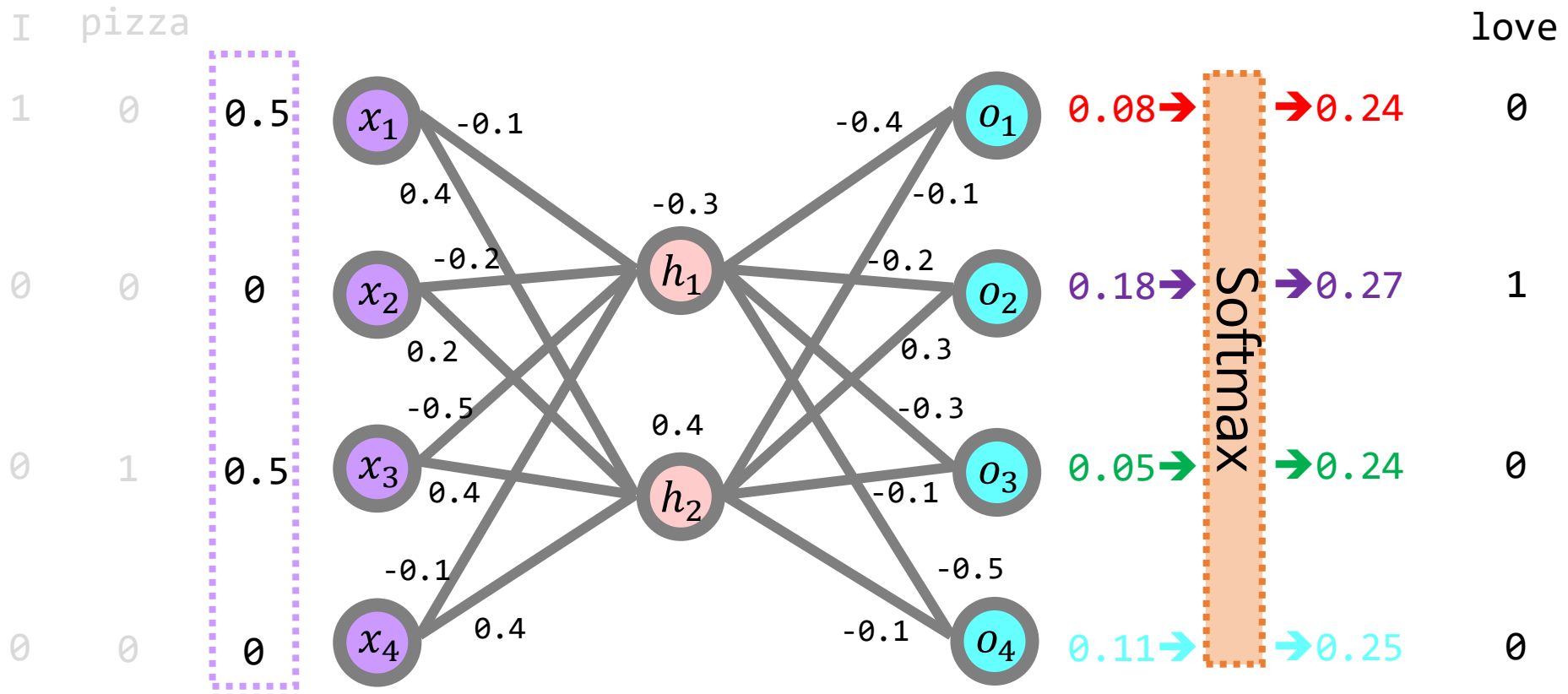
love

0
1
0
0

그 다음 출력값을 softmax 함수에 넣어서 확률로 변환시켜 줍니다.

I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]

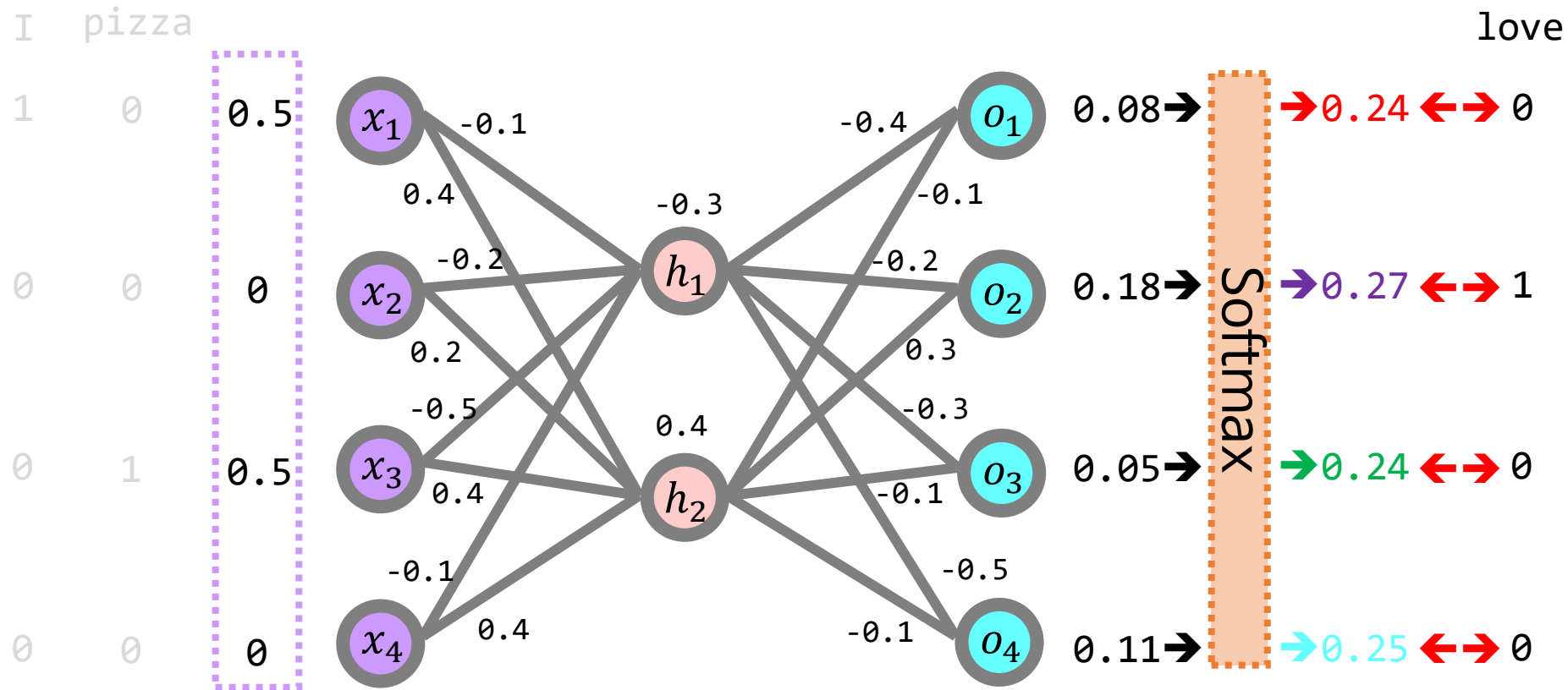
I love pizza



그러면 이제 우리는 출력값과 실제값사이의 차이 즉 손실loss를 구할 수 있게 되었습니다.

I = [1,0,0,0]  
 love = [0,1,0,0]  
 pizza = [0,0,1,0]  
 like = [0,0,0,1]

I love pizza

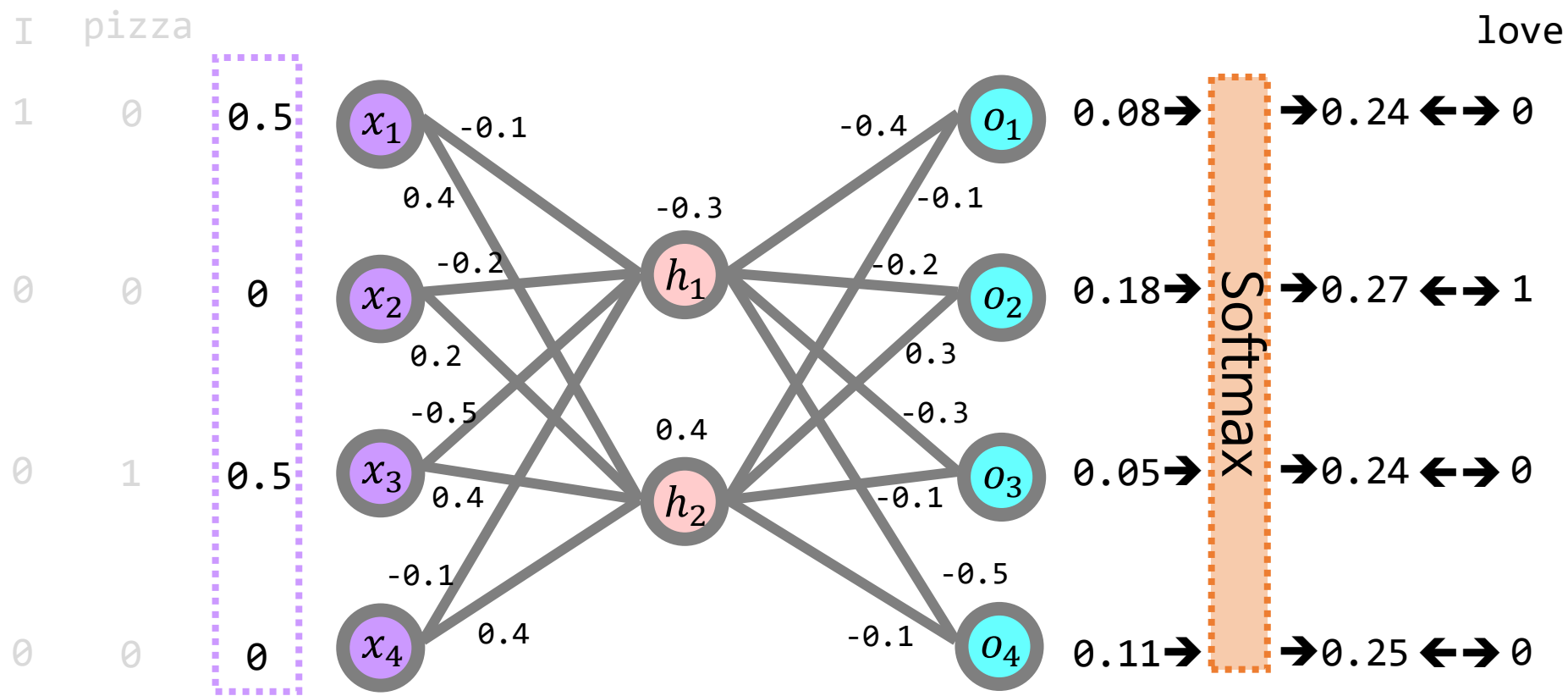


Word2vec에서 손실함수는 우리에게 익숙한 cross-entropy 함수를 사용합니다.

I = [1,0,0,0]  
 love = [0,1,0,0]  
 pizza = [0,0,1,0]  
 like = [0,0,0,1]

I love pizza

$$CE = \sum \log\left(\frac{1}{q(x)}\right) \cdot p(x)$$

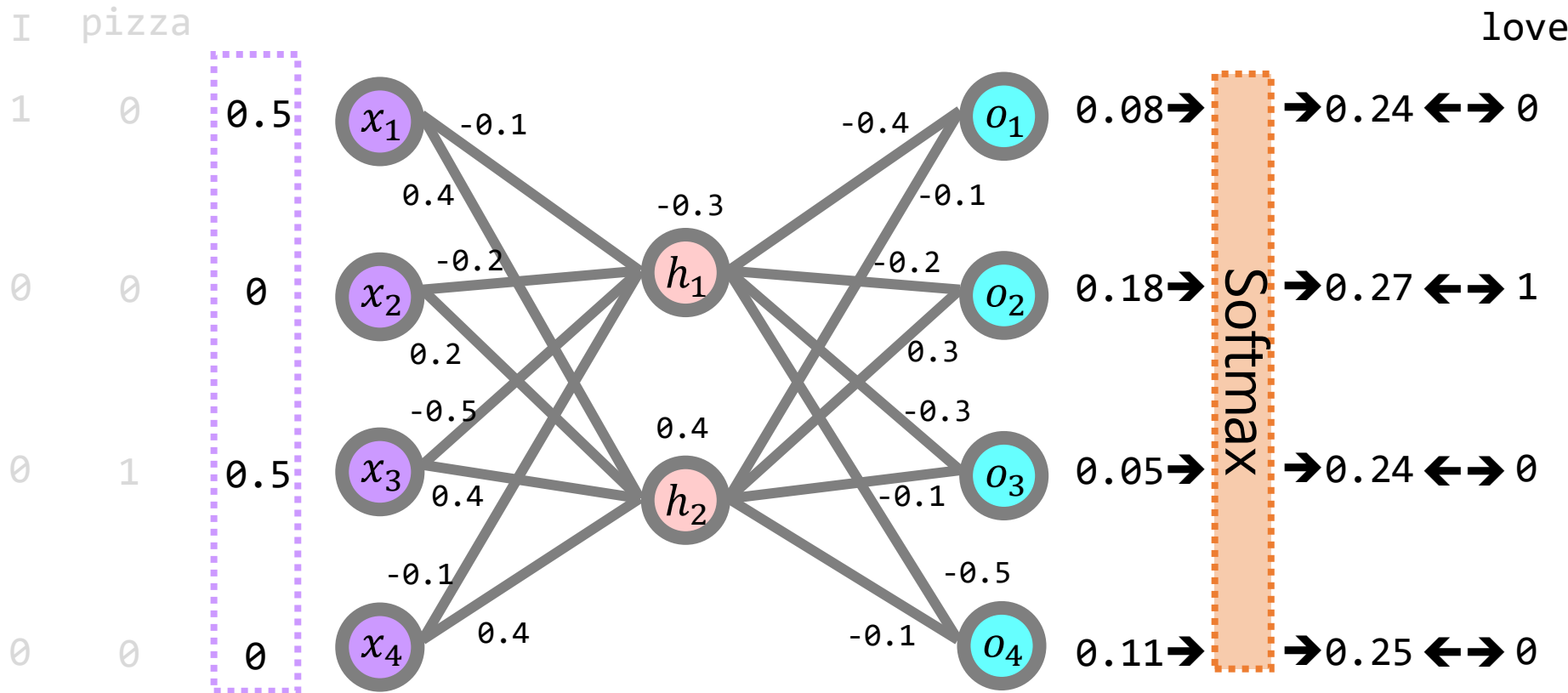


지난 영상에서도 보셨듯, cross-entropy와 softmax함수는 자주 만나는 조합입니다.

I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]

I love pizza

$CE = \sum \log(\frac{1}{q(x)}) \cdot p(x)$

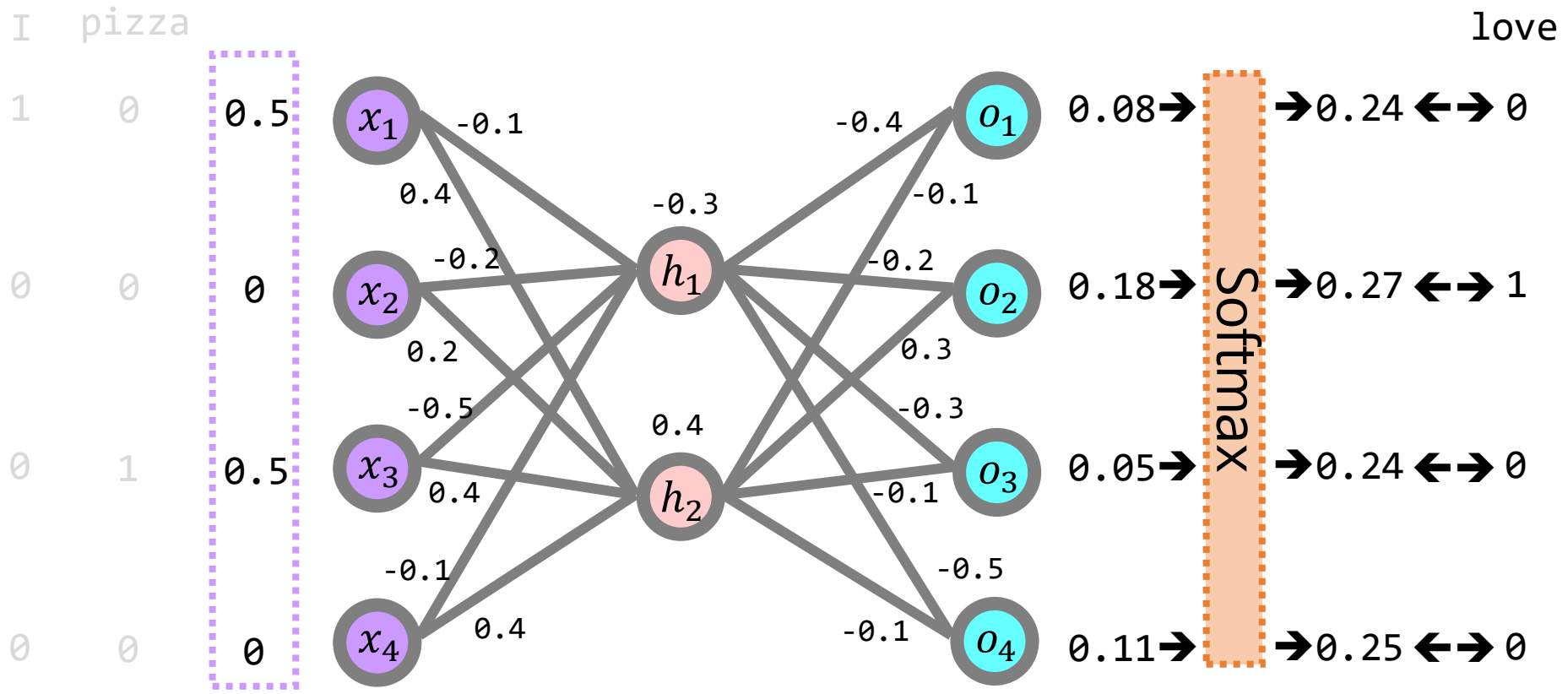


# 이렇게 손실까지 구할 수가 있습니다.

I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]

I love pizza

$CE = \sum \log(\frac{1}{q(x)}) \cdot p(x) = \log(\frac{1}{0.24}) \cdot 0 + \log(\frac{1}{0.27}) \cdot 1 + \log(\frac{1}{0.24}) \cdot 0 + \log(\frac{1}{0.25}) \cdot 0 = 1.31$



그런 뒤에 역전파와 경사하강법을 이용, 새로운 기울기 까지 구할 수 있습니다.

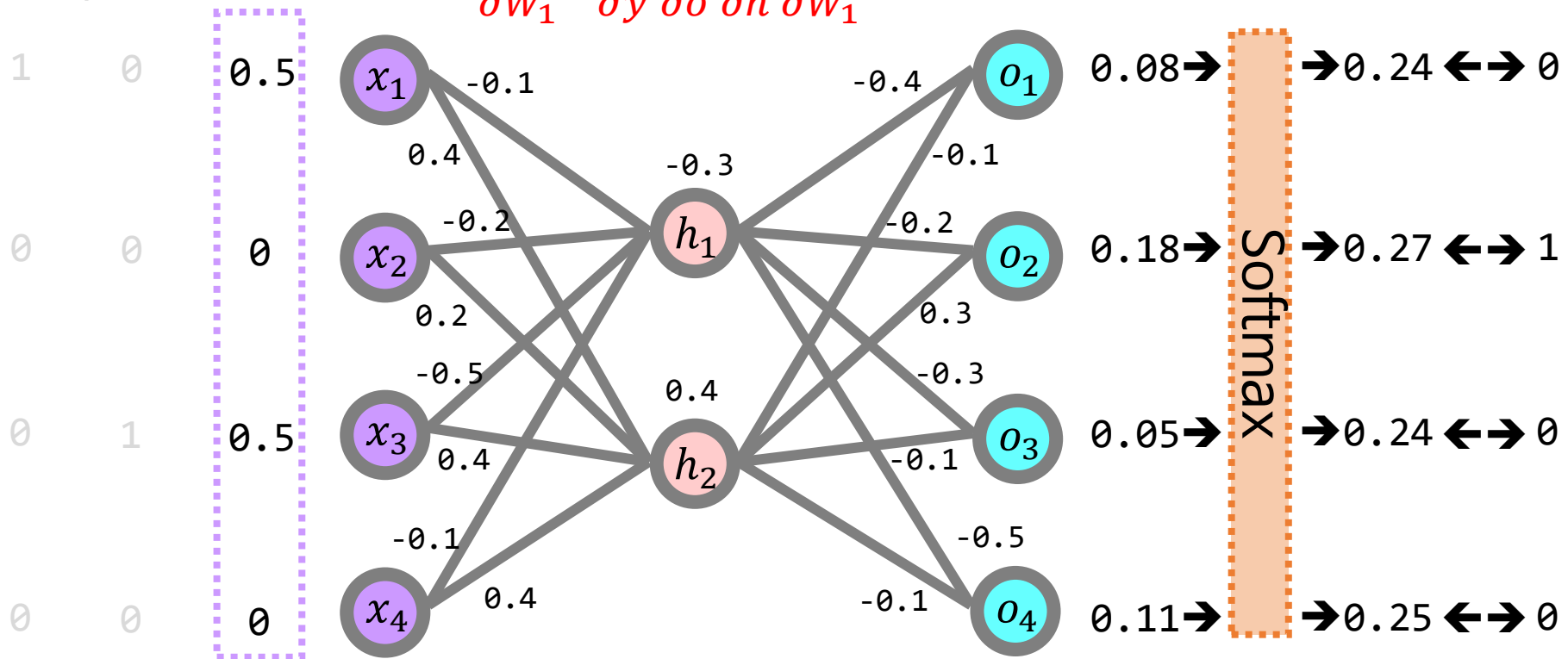
I = [1,0,0,0]  
 love = [0,1,0,0]  
 pizza = [0,0,1,0]  
 like = [0,0,0,1]

I love pizza

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o} \frac{\partial o}{\partial W_2} = h \otimes_{outer} (\hat{y} - y)$$

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o} \frac{\partial o}{\partial h} \frac{\partial h}{\partial W_1} = x \otimes_{outer} (W_2 \cdot (\hat{y} - y))$$

I pizza





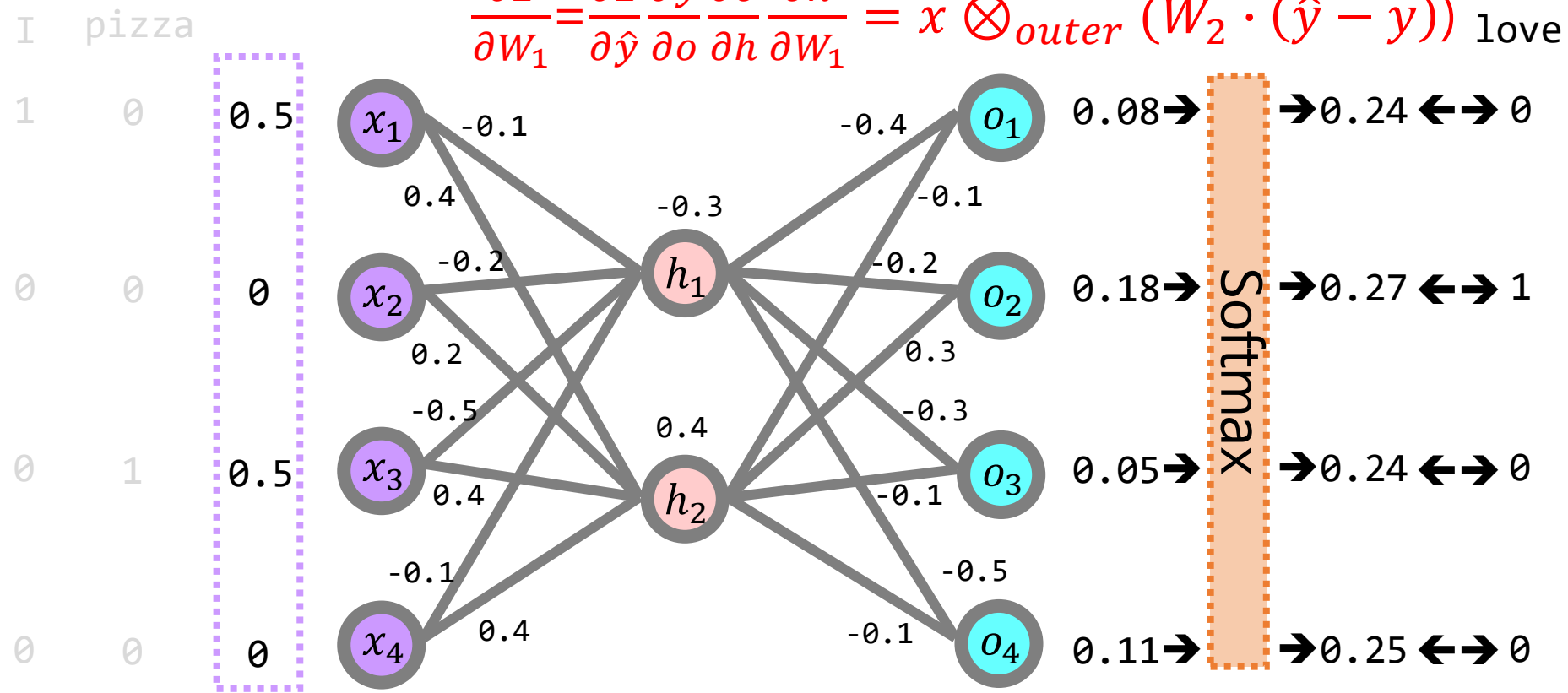
오늘은 역전파를 중점적으로 다루는 영상은 아니기 때문에,

I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]

I love pizza

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o} \frac{\partial o}{\partial W_2} = h \otimes_{outer} (\hat{y} - y)$$

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o} \frac{\partial o}{\partial h} \frac{\partial h}{\partial W_1} = x \otimes_{outer} (W_2 \cdot (\hat{y} - y))$$



# 자세한 도출 과정은 과감히 생략하도록 하겠사오니 양해를 부탁드립니다.

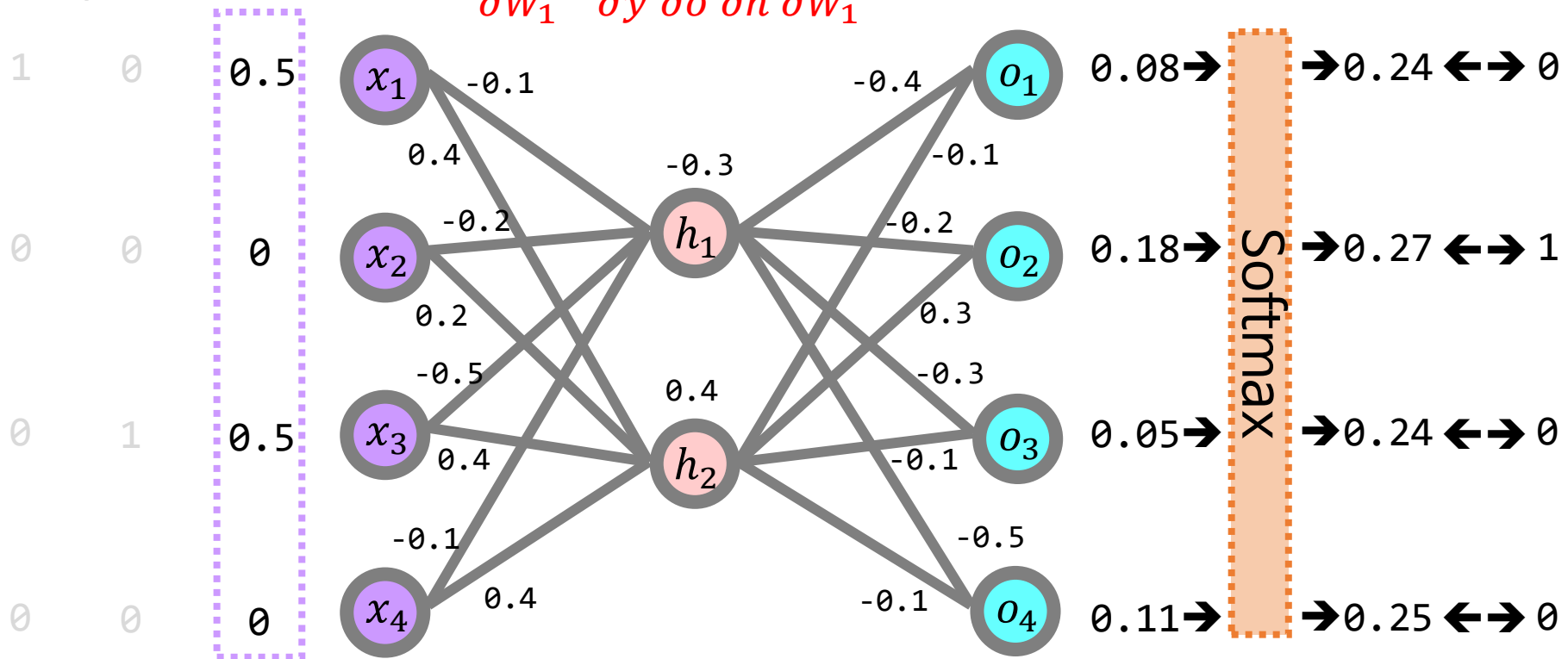
I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]

I love pizza

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o} \frac{\partial o}{\partial W_2} = h \otimes_{outer} (\hat{y} - y)$$

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o} \frac{\partial o}{\partial h} \frac{\partial h}{\partial W_1} = x \otimes_{outer} (W_2 \cdot (\hat{y} - y))$$

I pizza



그러나 역전파를 위한 체인룰 공식은 필요하신 분들을 위해서 써 두도록 하겠습니다.

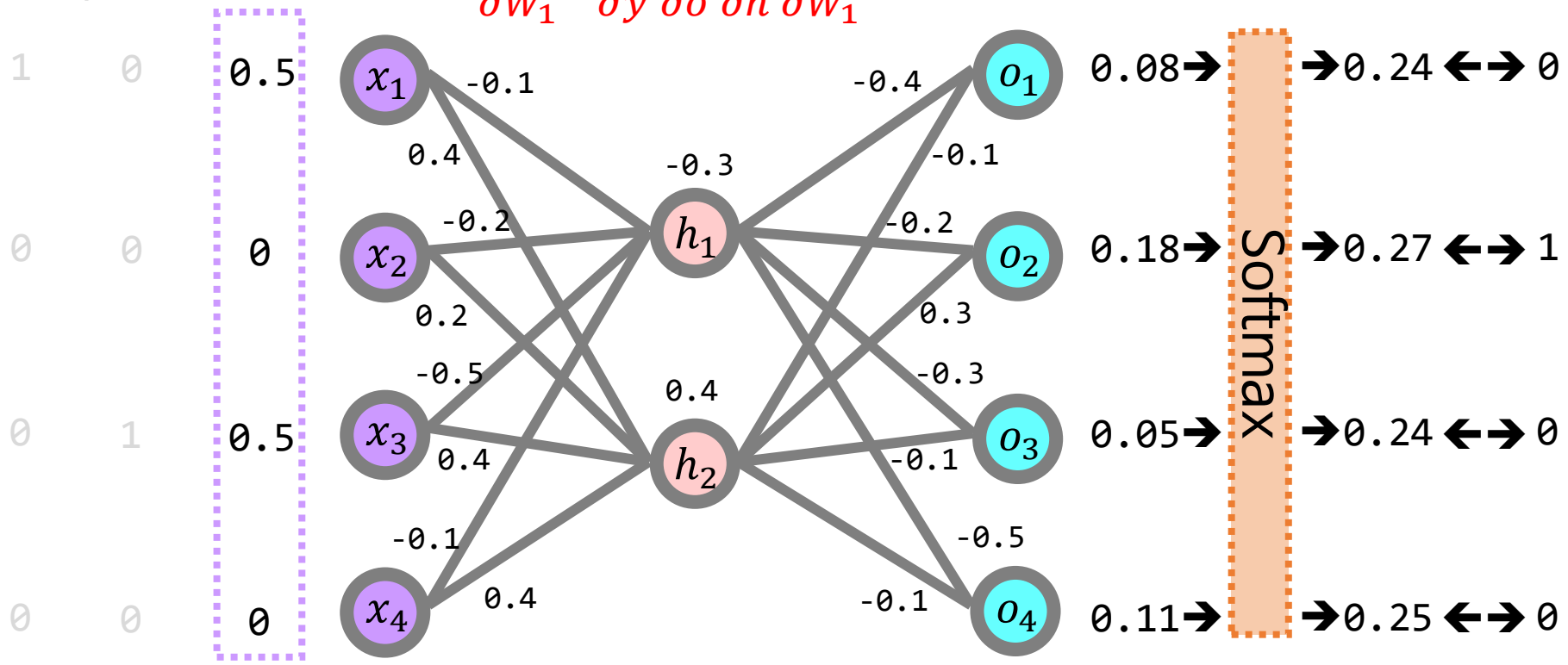
I = [1,0,0,0]  
 love = [0,1,0,0]  
 pizza = [0,0,1,0]  
 like = [0,0,0,1]

I love pizza

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o} \frac{\partial o}{\partial W_2} = h \otimes_{outer} (\hat{y} - y)$$

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o} \frac{\partial o}{\partial h} \frac{\partial h}{\partial W_1} = x \otimes_{outer} (W_2 \cdot (\hat{y} - y))$$

I pizza



# 이렇게 순전파와 손실계산, 역전파를 통한 가중치 업데이트를 계속하는 것이

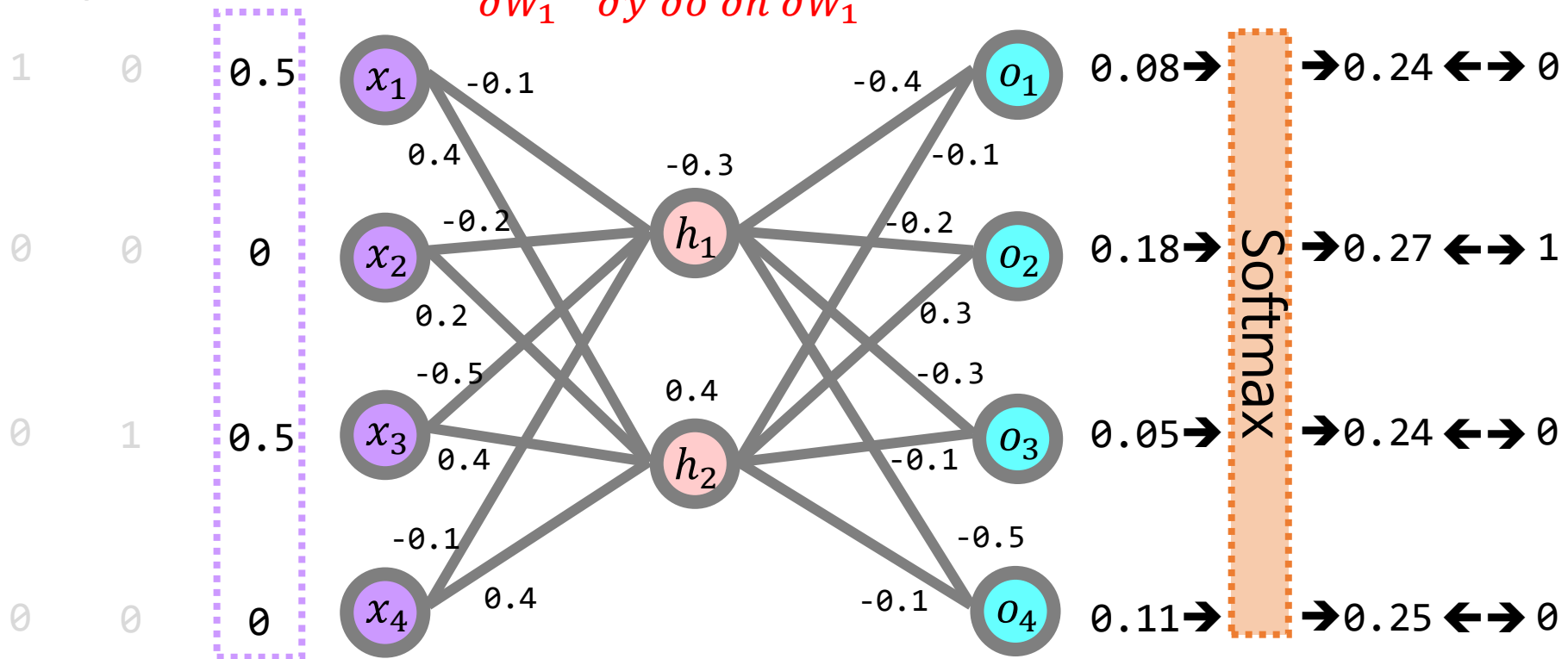
I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]

I love pizza

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o} \frac{\partial o}{\partial W_2} = h \otimes_{outer} (\hat{y} - y)$$

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o} \frac{\partial o}{\partial h} \frac{\partial h}{\partial W_1} = x \otimes_{outer} (W_2 \cdot (\hat{y} - y))$$

I pizza



# Continuous Bag-of-Words의 대략적인 학습과정이 되겠습니다.

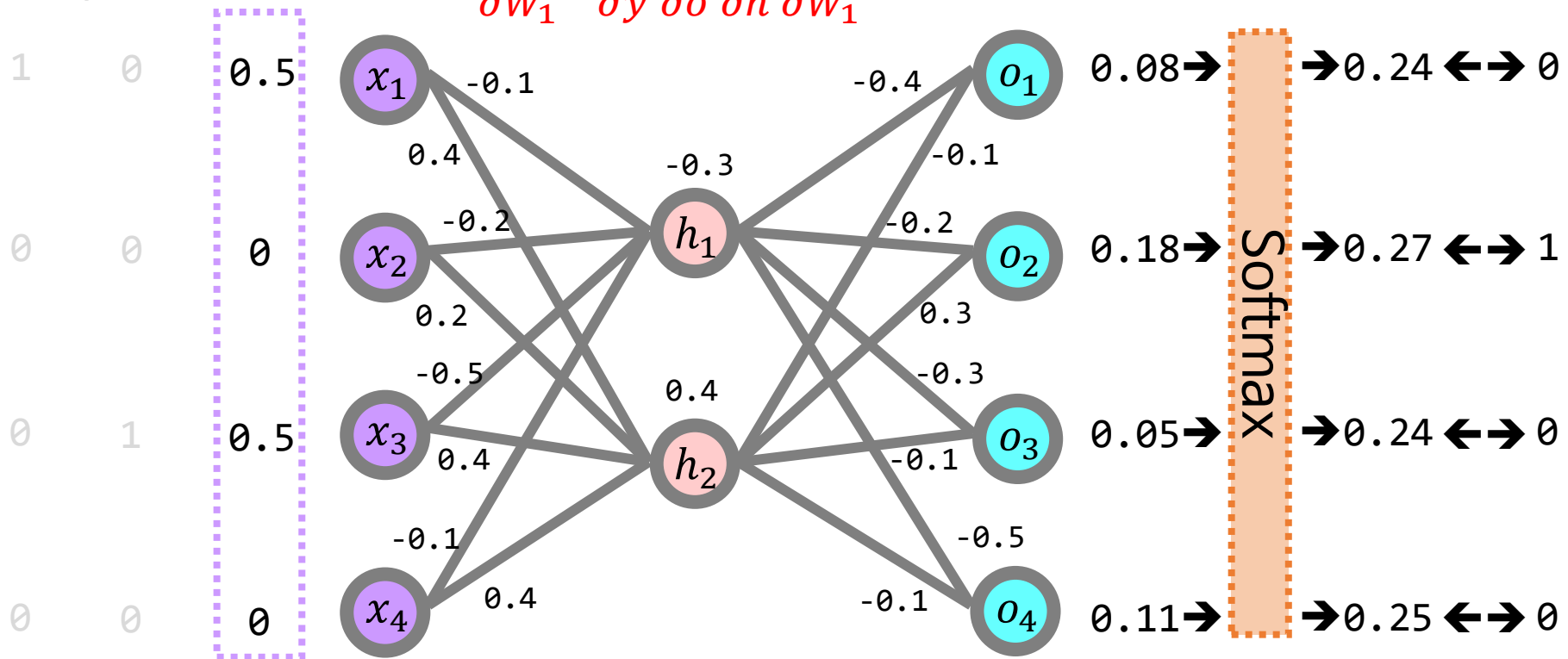
I = [1,0,0,0]  
 love = [0,1,0,0]  
 pizza = [0,0,1,0]  
 like = [0,0,0,1]

I love pizza

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o} \frac{\partial o}{\partial W_2} = h \otimes_{outer} (\hat{y} - y)$$

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o} \frac{\partial o}{\partial h} \frac{\partial h}{\partial W_1} = x \otimes_{outer} (W_2 \cdot (\hat{y} - y))$$

I pizza



# 그러면 CBOW 학습을 통해 word2vec 모델은 무엇을 학습하는 걸까요?

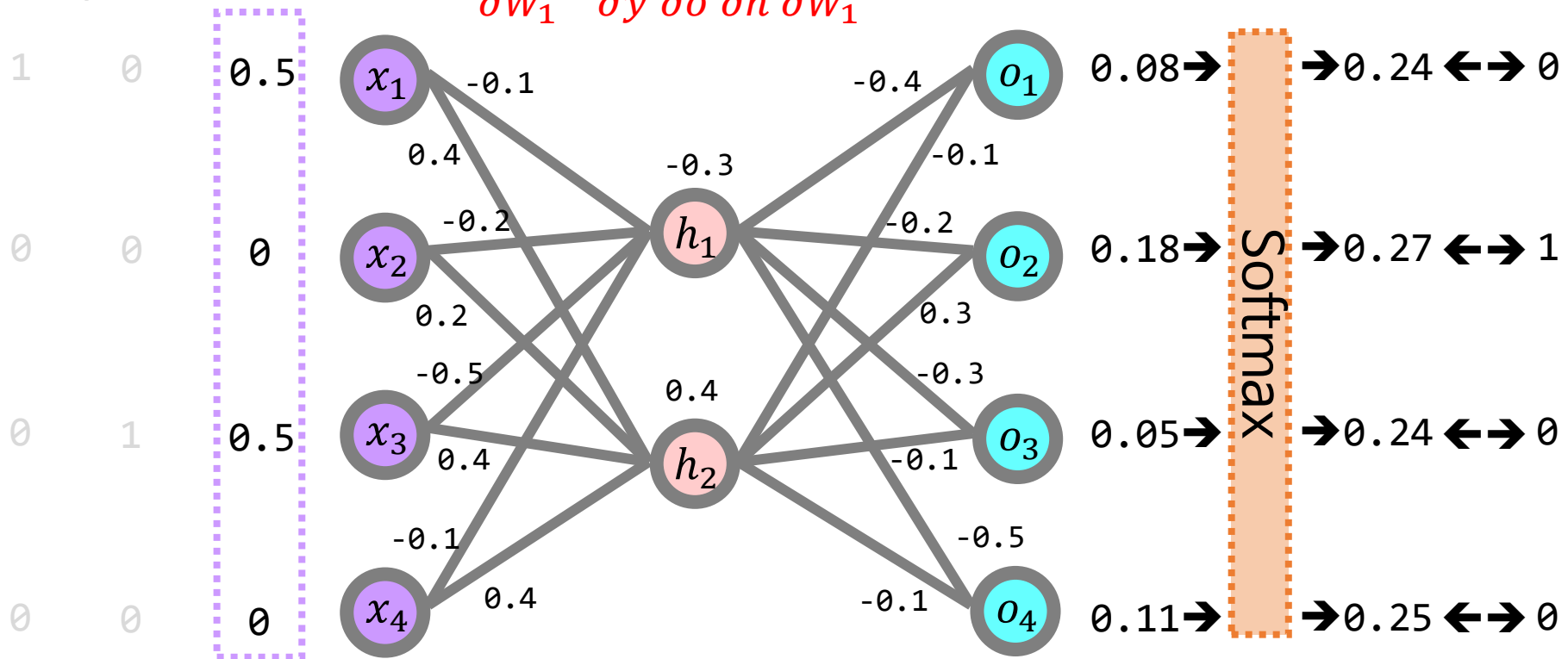
I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]

I love pizza

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o} \frac{\partial o}{\partial W_2} = h \otimes_{outer} (\hat{y} - y)$$

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o} \frac{\partial o}{\partial h} \frac{\partial h}{\partial W_1} = x \otimes_{outer} (W_2 \cdot (\hat{y} - y))$$

I pizza



보시다시피, 은닉층 노드는 4차원의 입력 벡터를 2차원으로 축소된 정보를 처리하게 됩니다.

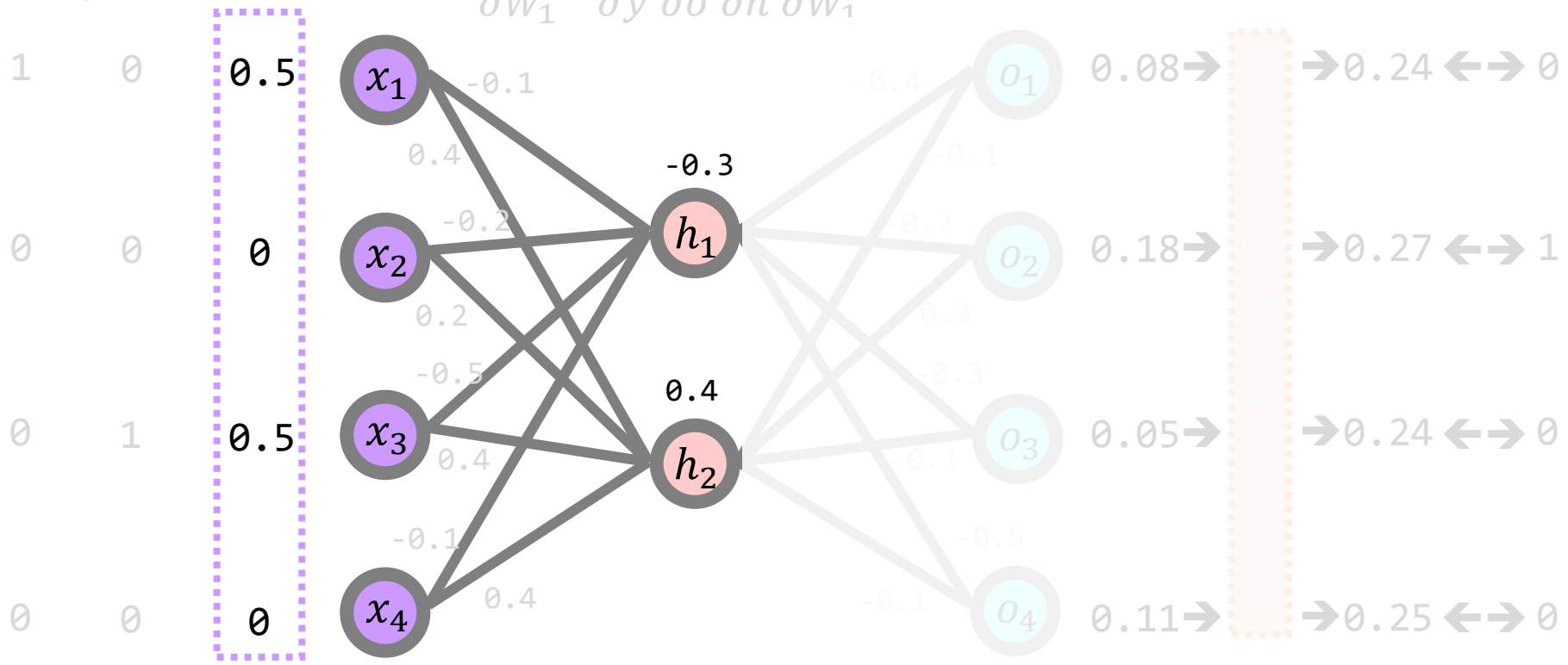
I = [1,0,0,0]  
 love = [0,1,0,0]  
 pizza = [0,0,1,0]  
 like = [0,0,0,1]

I love pizza

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o} \frac{\partial o}{\partial W_2} = h \otimes_{outer} (\hat{y} - y)$$

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o} \frac{\partial o}{\partial h} \frac{\partial h}{\partial W_1} = x \otimes_{outer} (W_2 \cdot (\hat{y} - y))_{love}$$

I pizza



2차원 데이터이기 때문에 이렇게 2차원 평면에 그 값들을 배치해 보면,

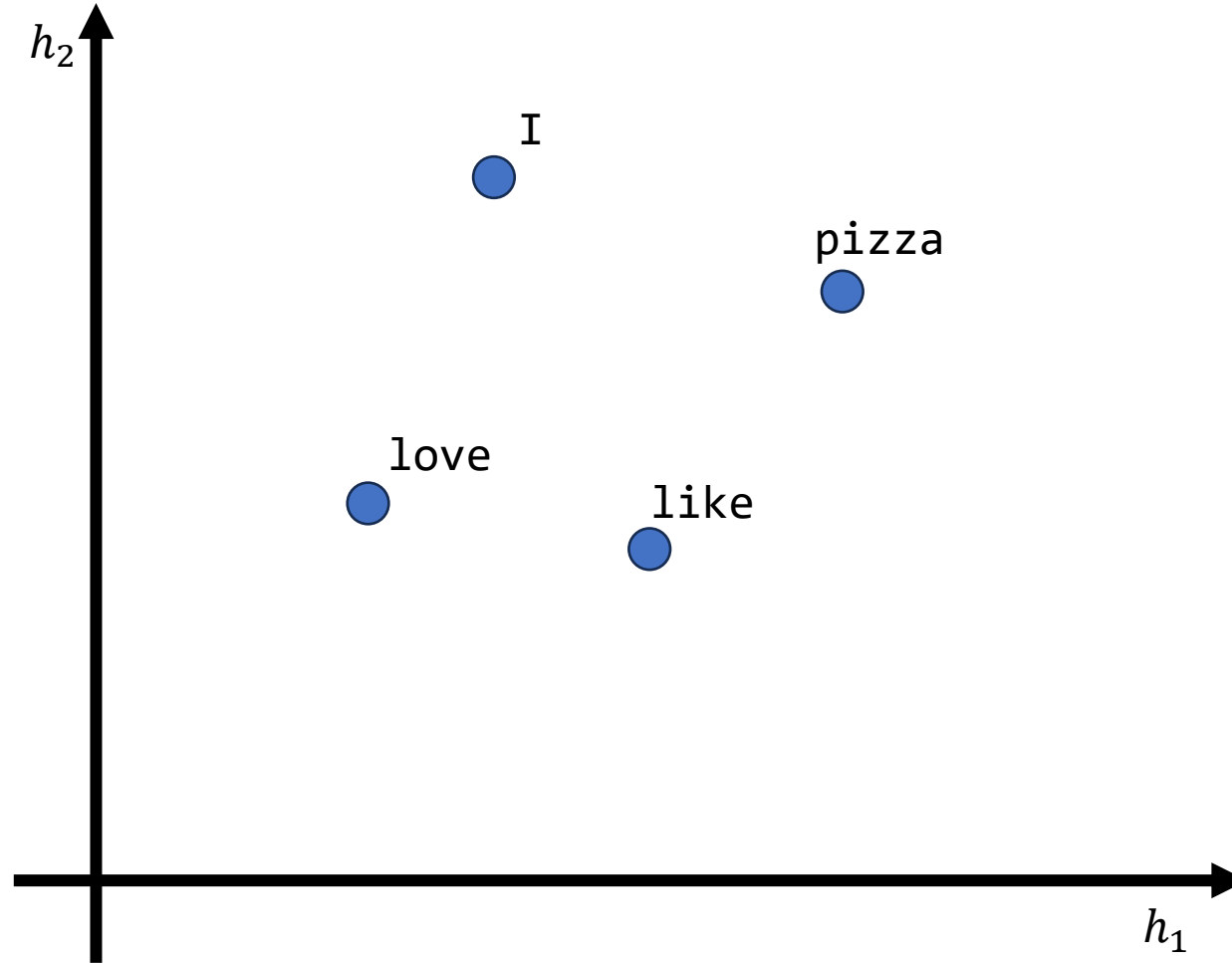
I	=	[1,0,0,0]
love	=	[0,1,0,0]
pizza	=	[0,0,1,0]
like	=	[0,0,0,1]





초기에는 가중치가 임의로 배치되었기 때문에 각 단어의 은닉층 값들은  
아마 다음과 같이 무질서할 것입니다.

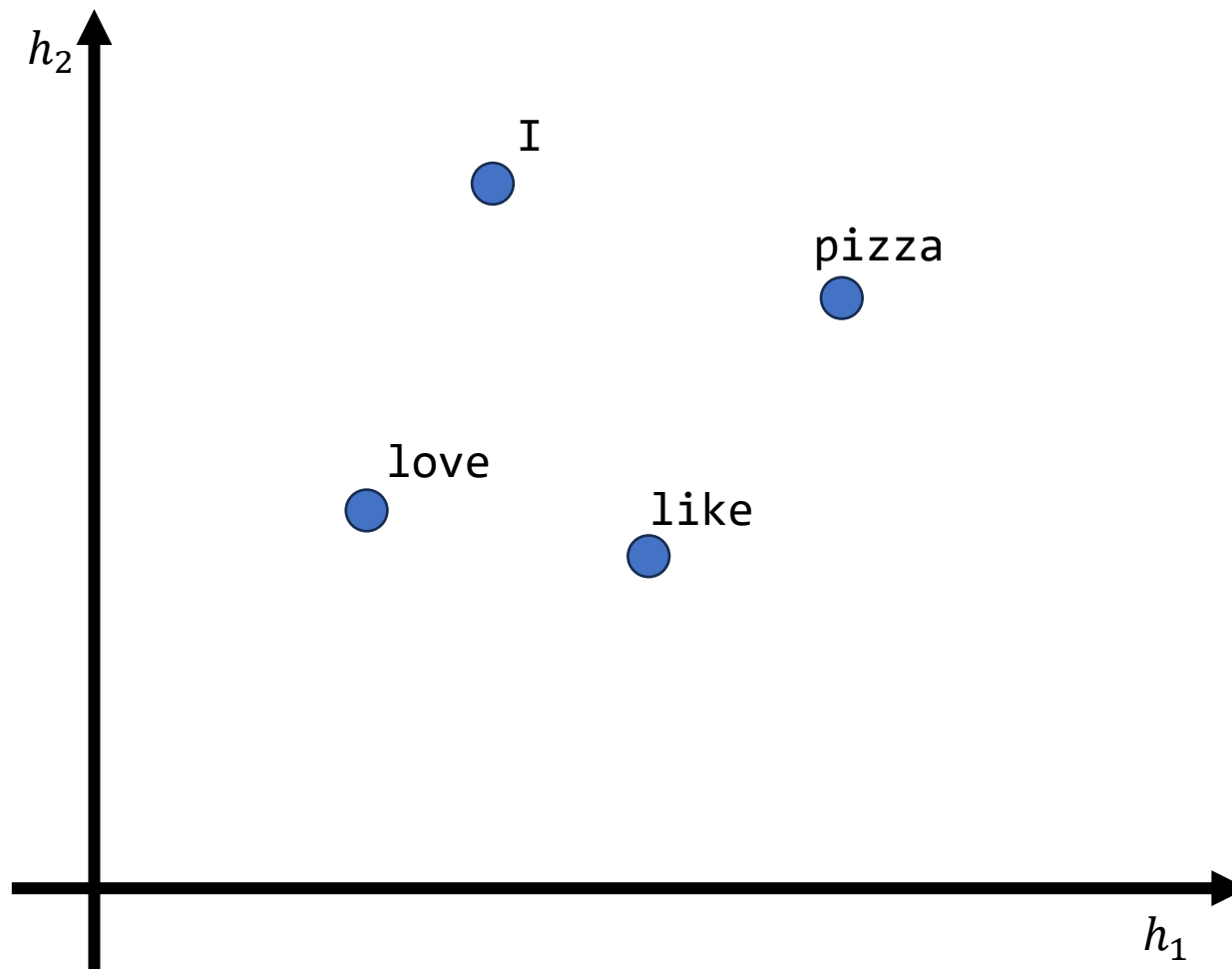
I = [1, 0, 0, 0]  
love = [0, 1, 0, 0]  
pizza = [0, 0, 1, 0]  
like = [0, 0, 0, 1]



# 하지만 다음과 같은 CBOW 학습이 지속될 경우..

I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]

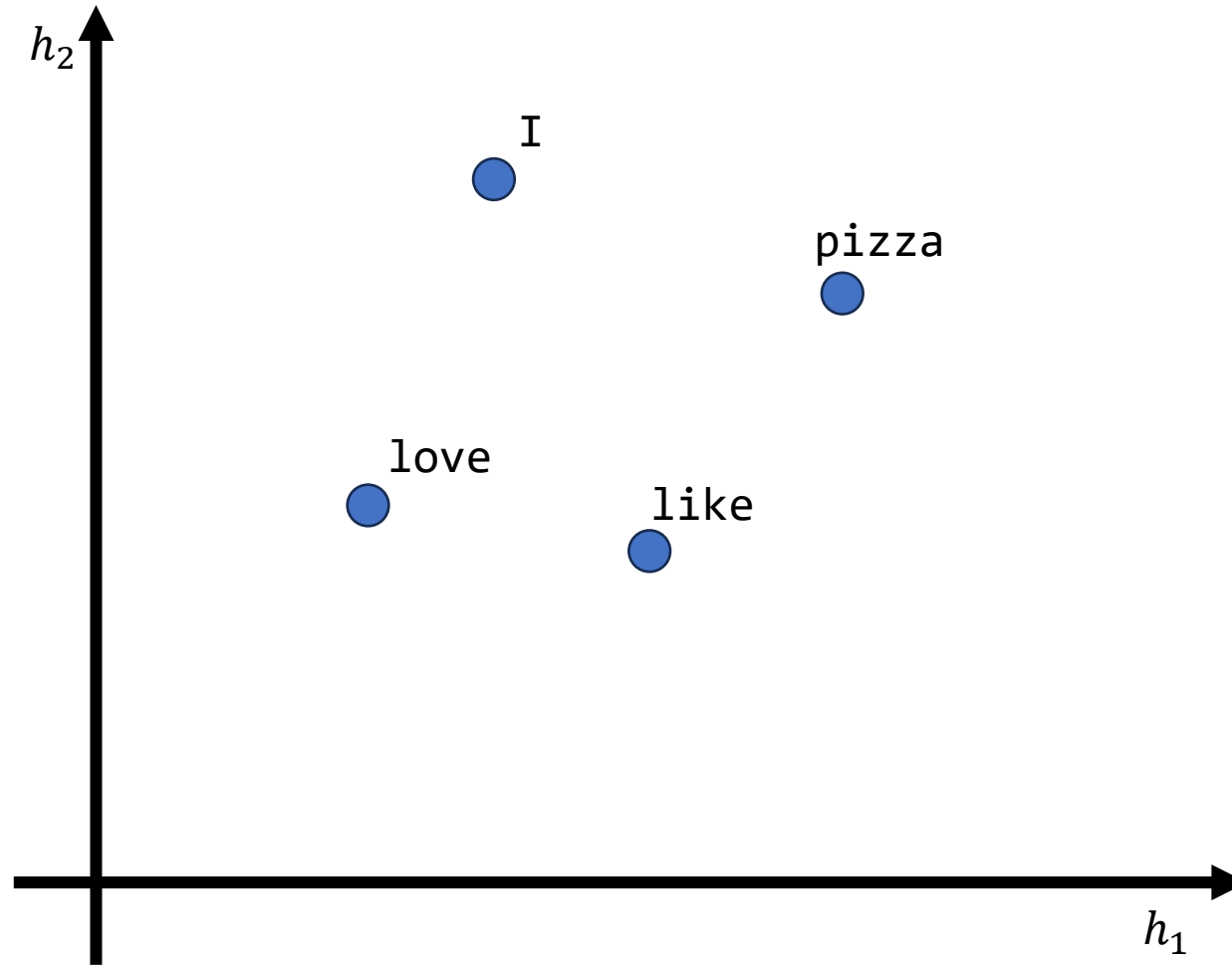
I love pizza  
I like pizza  
I love pizza  
I like pizza  
I love pizza  
I like pizza  
I love pizza  
I like pizza  
...



같은 패턴인 I 와 pizza에 대하여 love와 like의 문맥상 용법이 유사하기 때문에,

I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]

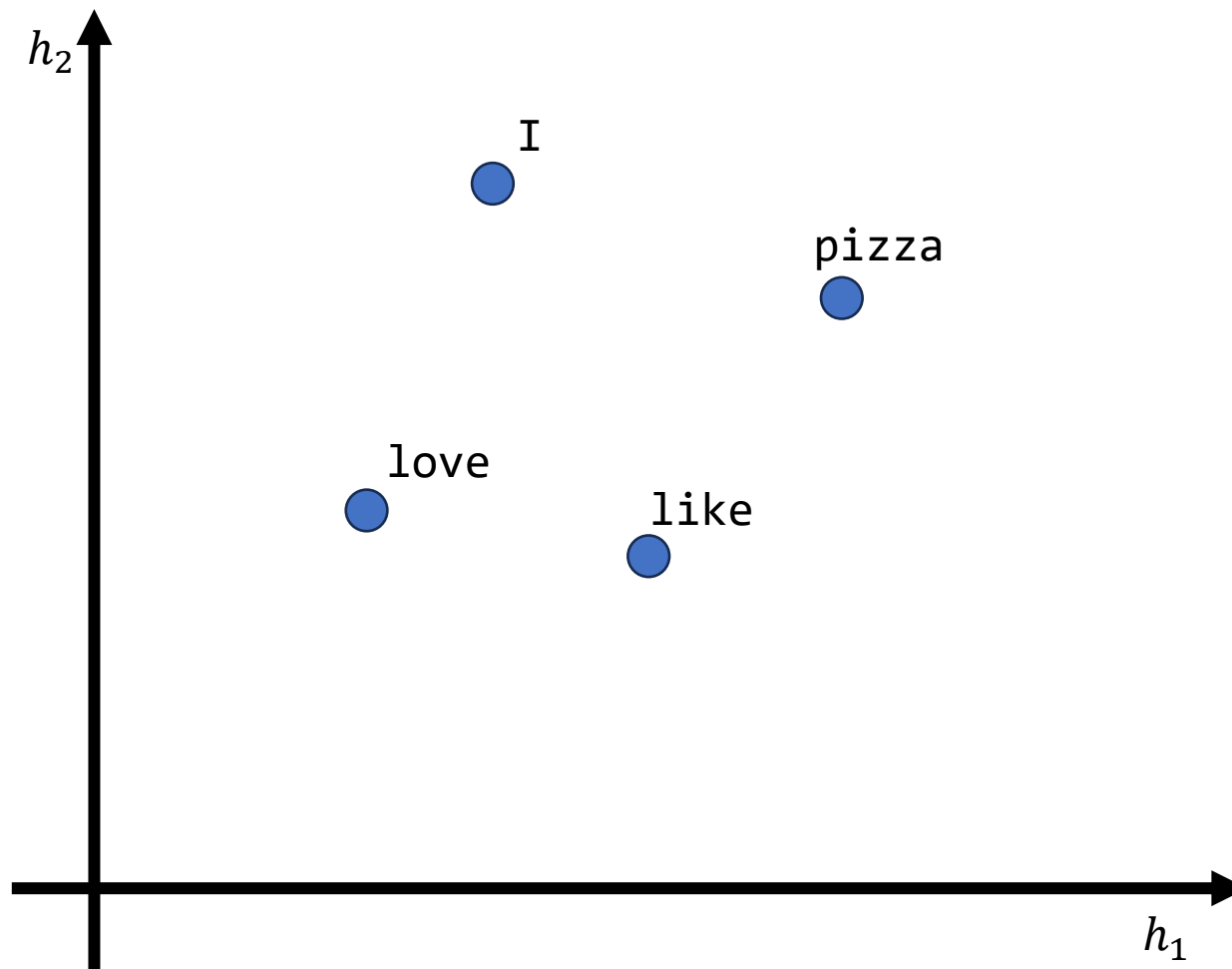
I love pizza  
I like pizza  
I love pizza  
I like pizza  
I love pizza  
I like pizza  
I love pizza  
I like pizza  
...



결국 love와 like는 공간상 유사한 벡터가 되어가고 I 와 pizza로부터는 멀어지는 경향성을 보이게 될 것입니다.

I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]

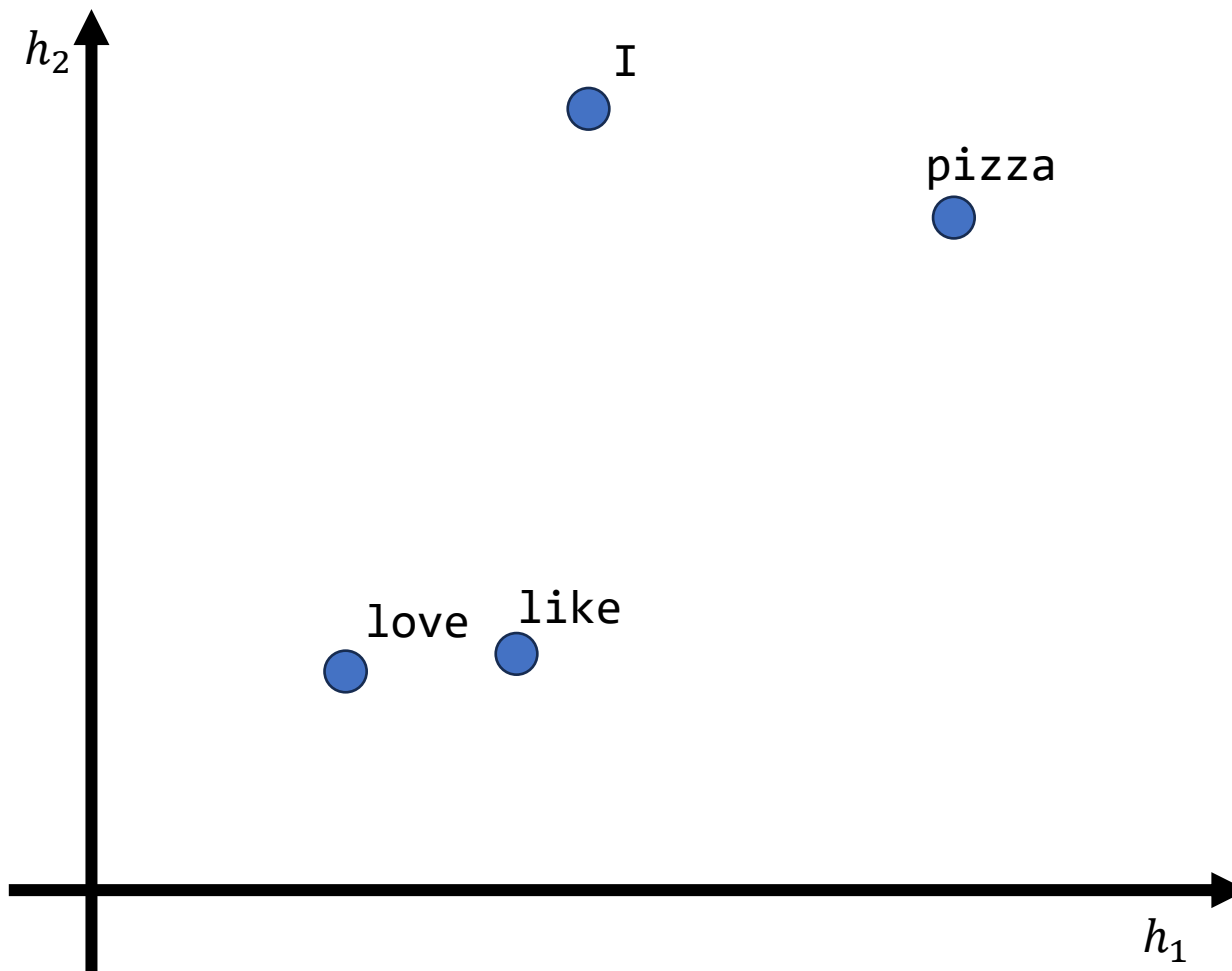
I love pizza  
I like pizza  
I love pizza  
I like pizza  
I love pizza  
I like pizza  
I love pizza  
I like pizza  
...



결국 love와 like는 공간상 유사한 벡터가 되어가고 I 와 pizza로부터는 멀어지는 경향성을 보이게 될 것입니다.

I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]

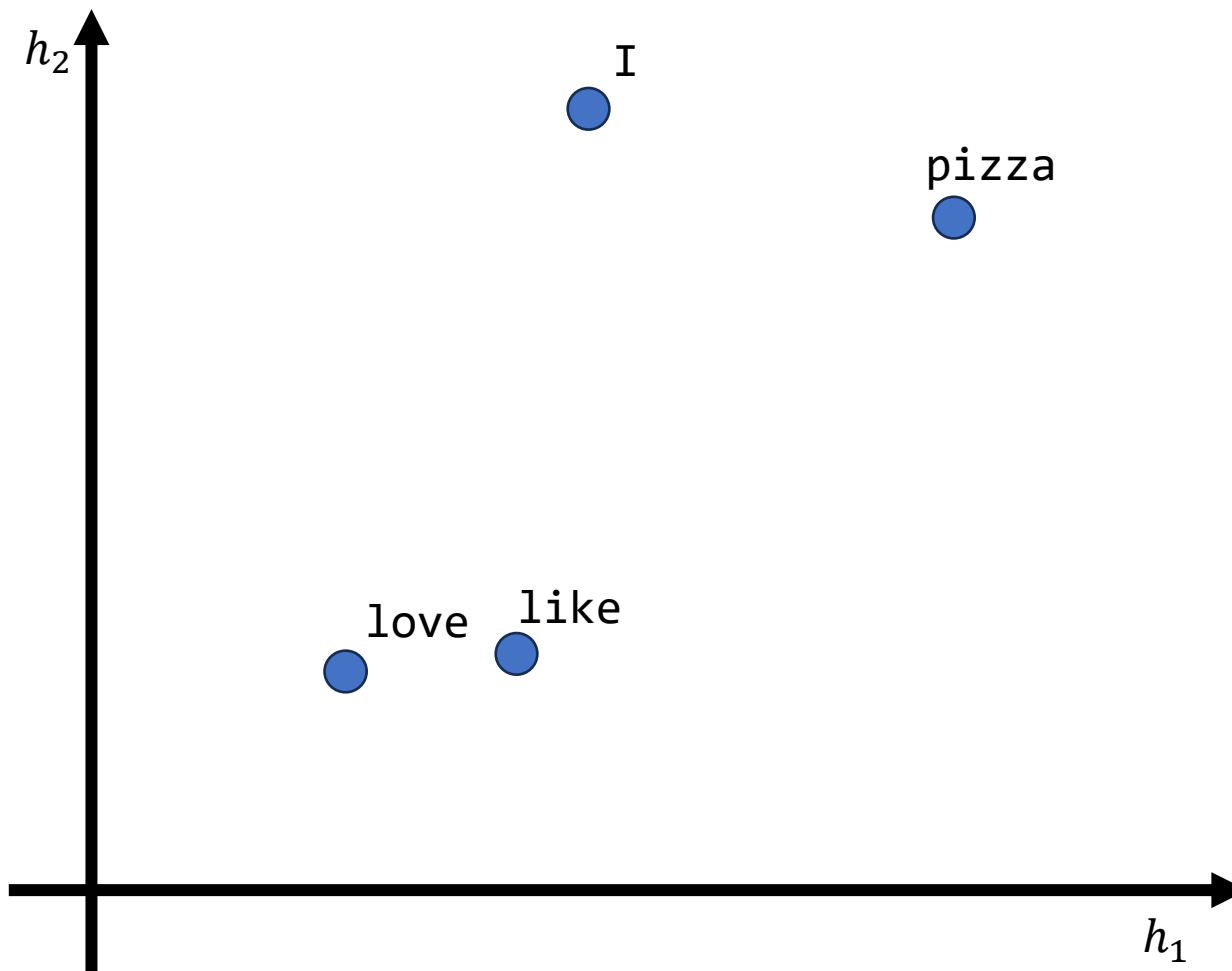
I love pizza  
I like pizza  
I love pizza  
I like pizza  
I love pizza  
I like pizza  
I love pizza  
I like pizza  
...



# 이런식으로 수십만 수백만개의 단어들의 학습을 통해

I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]

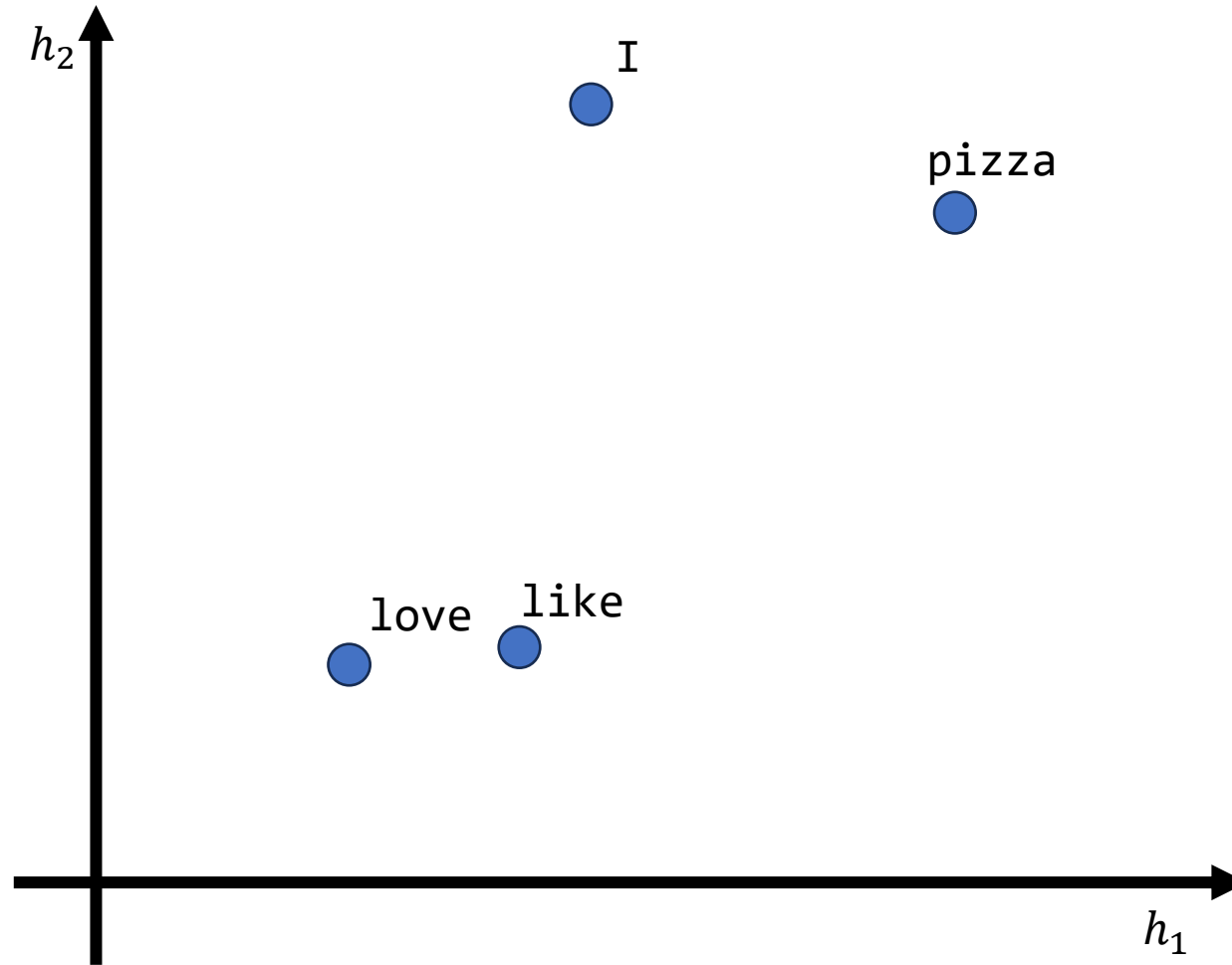
I love pizza  
I like pizza  
I love pizza  
I like pizza  
I love pizza  
I like pizza  
I love pizza  
I like pizza  
...



단어들이 문맥 안에서의 그들의 관계성을 스스로 학습해 나가는  
알고리즘이 바로 word2vec 알고리즘입니다.

I = [1, 0, 0, 0]  
love = [0, 1, 0, 0]  
pizza = [0, 0, 1, 0]  
like = [0, 0, 0, 1]

I love pizza  
I like pizza  
I love pizza  
I like pizza  
I love pizza  
I like pizza  
I love pizza  
I like pizza  
...



좀 전에 말씀드린것 처럼, Word2vec은 다음 두가지 방법으로 단어 데이터셋을 훈련합니다

Continuous Bag-of-words: CBOW

Skip-Gram



Skip-Gram은 CBOW와 반대되는 개념의 학습방법입니다.

Continuous Bag-of-words: CBOW

Skip-Gram

CBOW는 여러 개의 단어를 주고 하나의 단어를 출력하도록 하는 학습과정이라면,

Continuous Bag-of-words: CBOW

Skip-Gram

Skip-Gram은 하나의 단어를 주고 그 단어 주변에 나타난 여러 단어를 출력하도록 학습하는 방식입니다.

Continuous Bag-of-words: CBOW

Skip-Gram

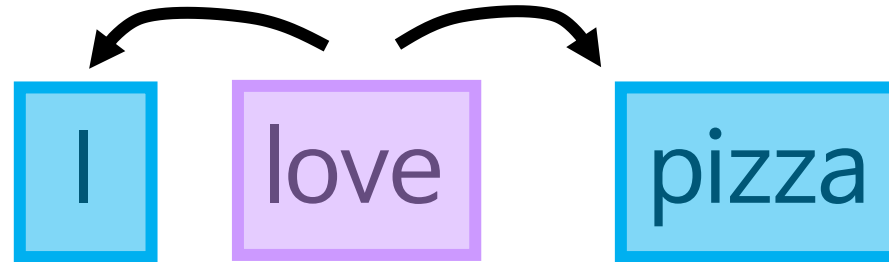
예를들어서, 다음과 같은 문장이 있다고 가정합시다

I      love      pizza

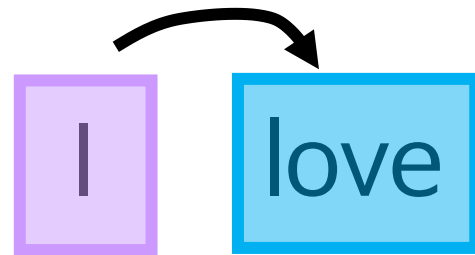
여기서 ‘주변’이란 바로 옆 단어를 가리킨다고 가정 했을 때,

I      love      pizza

Love의 주변은 I 와 pizza가 됩니다

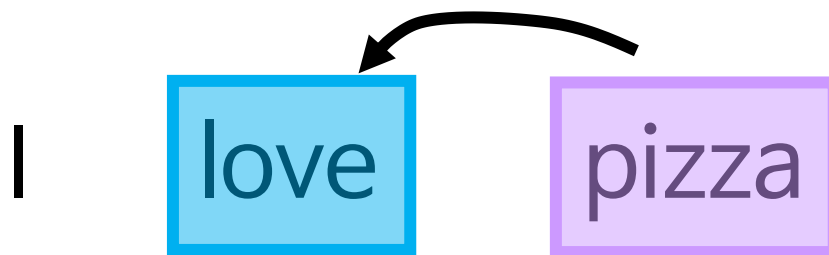


그러나 I의 주변은 love가 되고



pizza

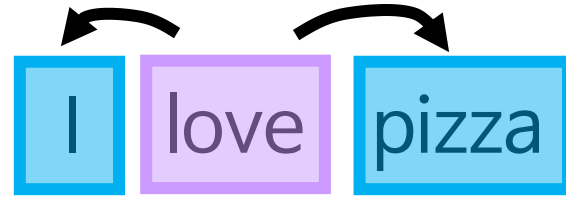
마찬가지로 pizza의 주변은 love입니다.





이처럼 Skip-Gram은 입력 단어와 그 입력단어의 주변에 해당하는 단어들의 관계를 학습합니다.

I = [1, 0, 0, 0]  
love = [0, 1, 0, 0]  
pizza = [0, 0, 1, 0]  
like = [0, 0, 0, 1]



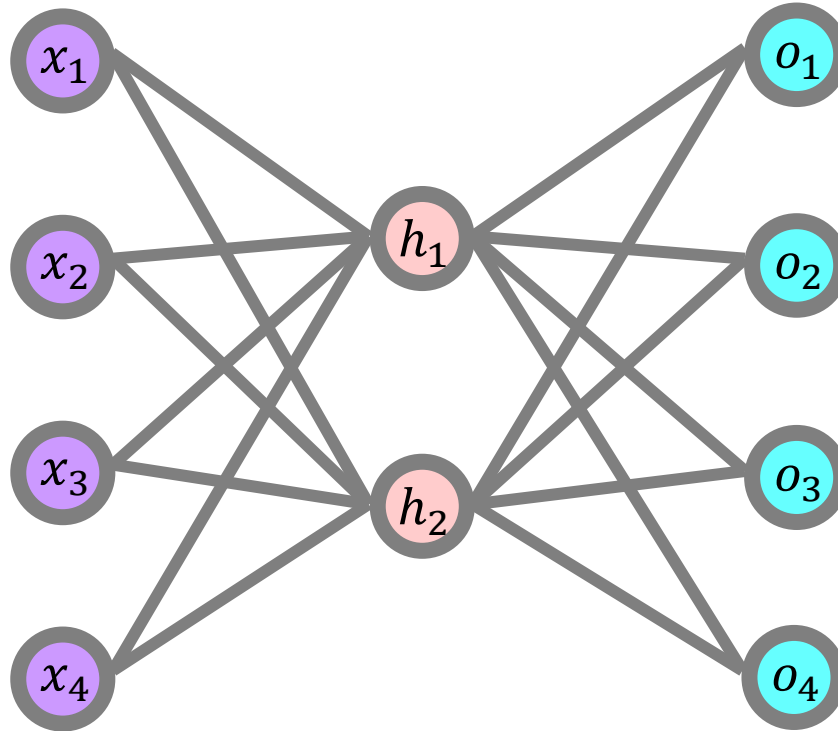
love

0

1

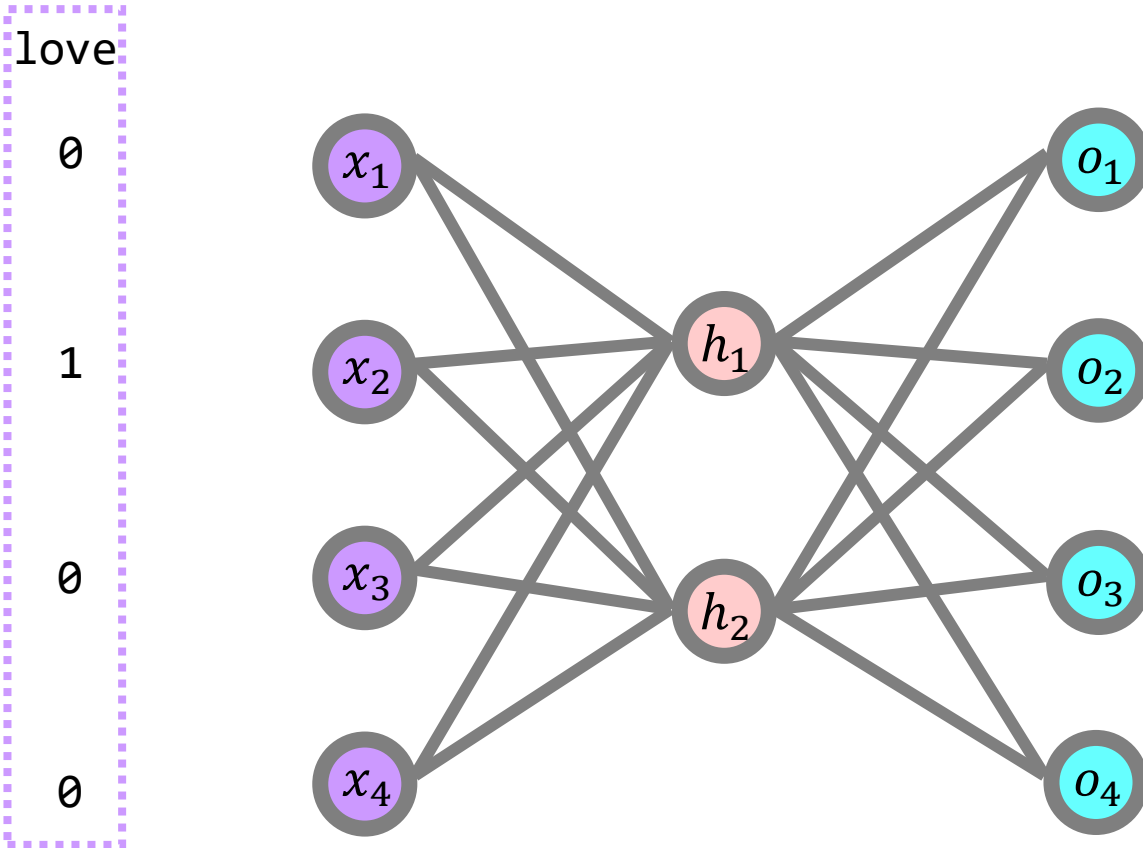
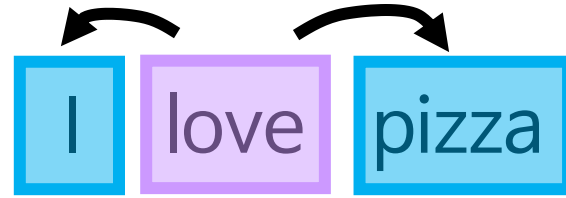
0

0



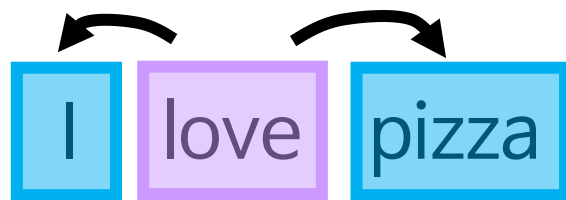
여기에서 보듯, Skip-Gram 학습 방법은, 예를들어 love라는 단어 벡터를 넣으면,

I = [1, 0, 0, 0]  
love = [0, 1, 0, 0]  
pizza = [0, 0, 1, 0]  
like = [0, 0, 0, 1]



# Love의 주변에 해당하는 I와 pizza를 학습하도록 합니다

I = [1, 0, 0, 0]  
love = [0, 1, 0, 0]  
pizza = [0, 0, 1, 0]  
like = [0, 0, 0, 1]



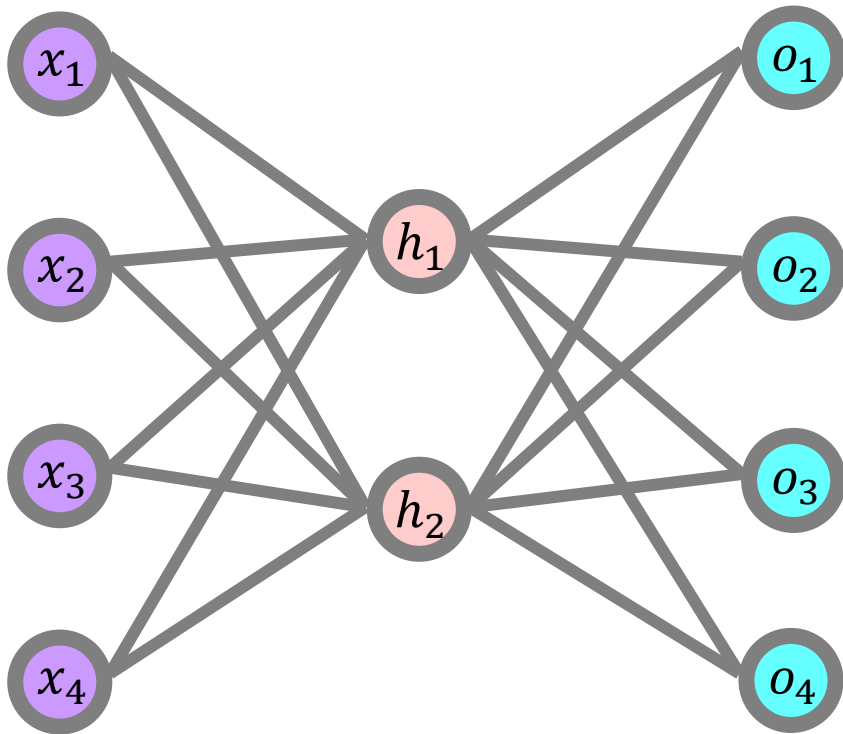
love

0

1

0

0



I pizza

1 0

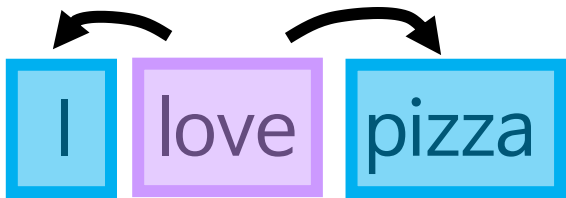
0 0

0 1

0 0

마찬가지로 이 둘을 합하여 하나의 벡터로 만들고,

I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]



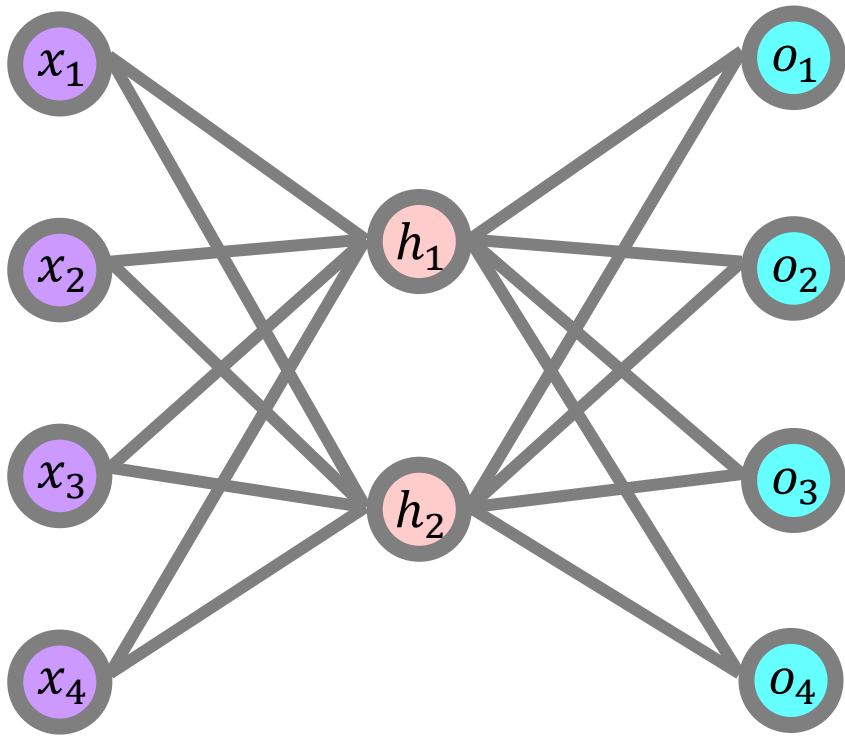
love

0

1

0

0



I & pizza

1

0

1

0

I      pizza

1      0

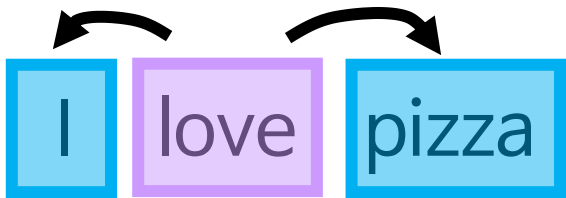
0      0

0      1

0      0

# 확률분포 형태여야 하기 때문에 합이 1이 되도록 나누어 줍니다.

I = [1,0,0,0]  
love = [0,1,0,0]  
pizza = [0,0,1,0]  
like = [0,0,0,1]



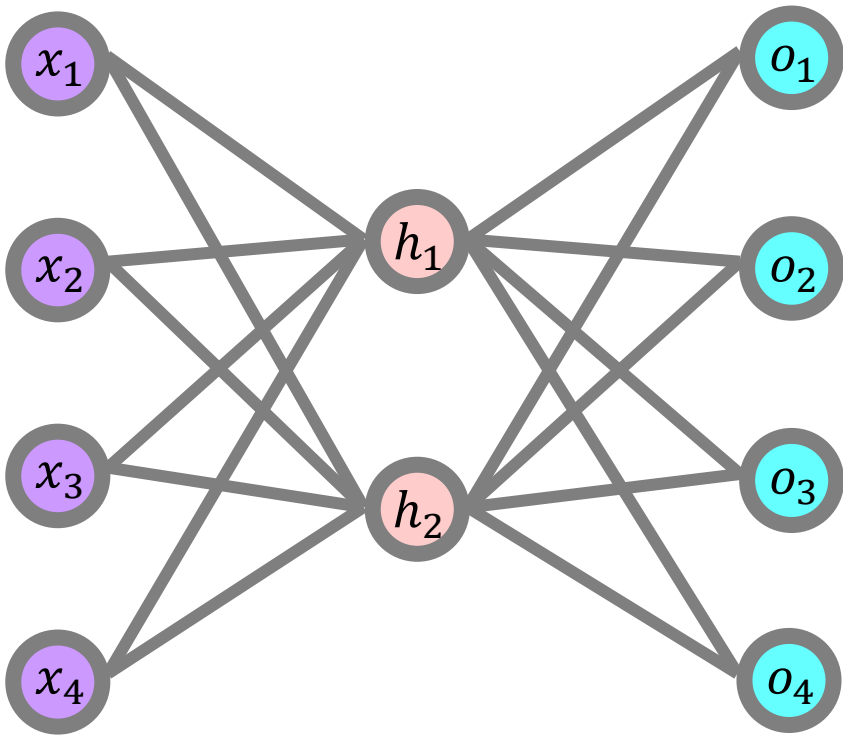
love

0

1

0

0



I & pizza

0.5

0

0.5

0

I      pizza

1      0

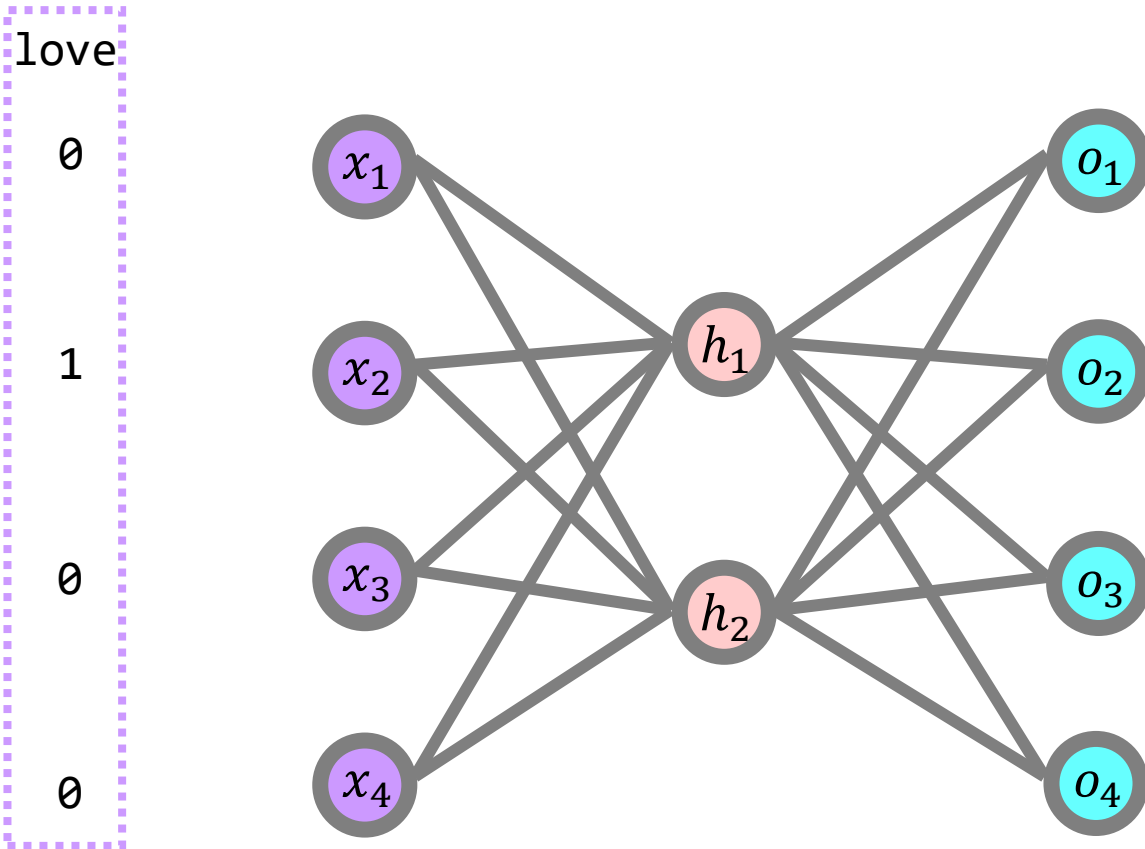
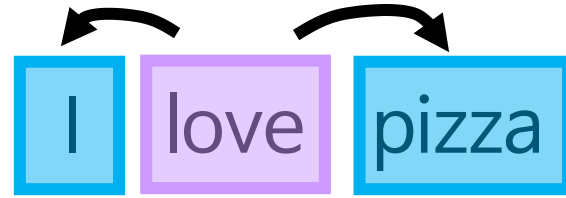
0      0

0      1

0      0

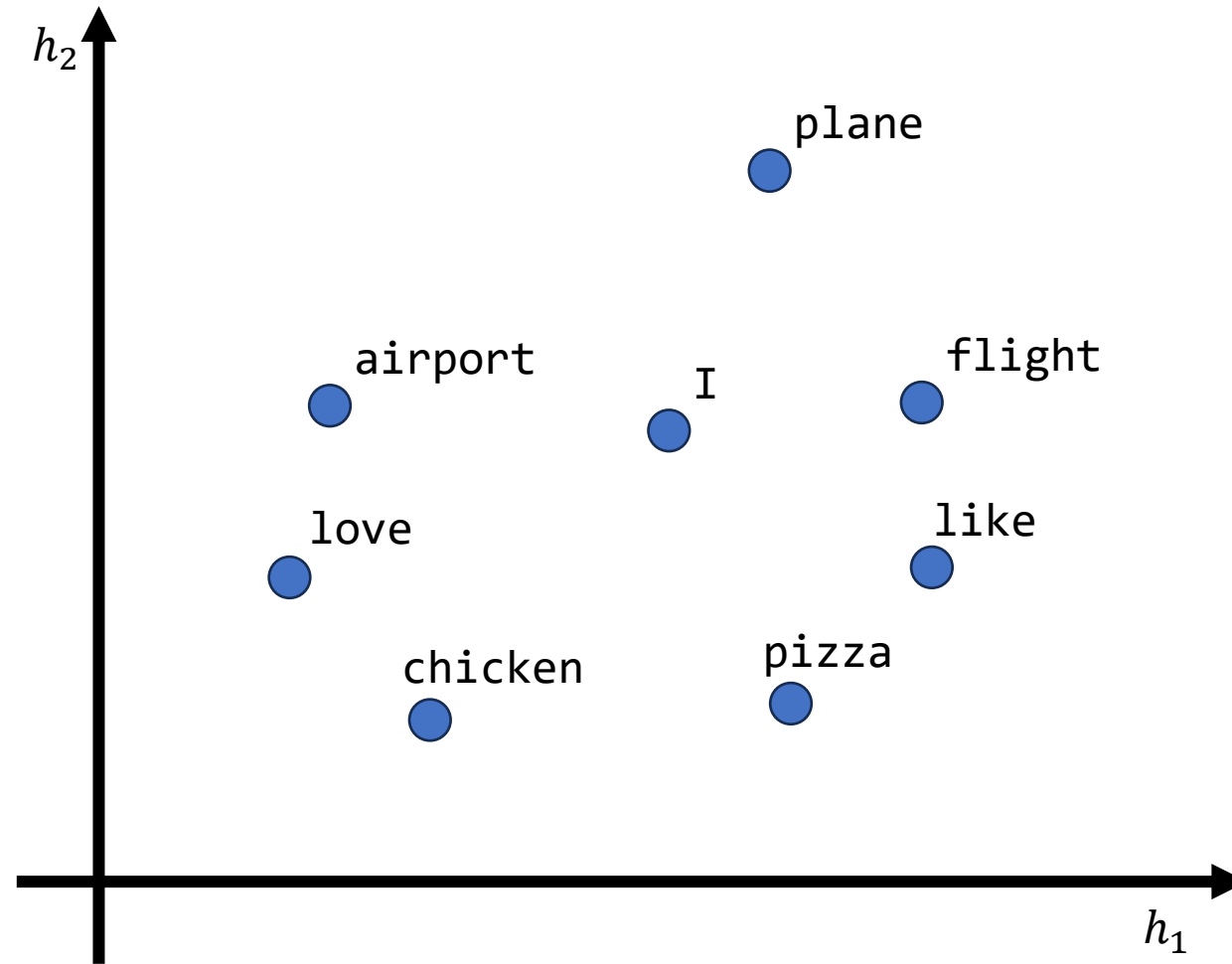
이렇게 해서 CBOW때와 마찬가지로 순전파, 손실, 역전파를 통해서 가중치를 업데이트 합니다.

I = [1, 0, 0, 0]  
love = [0, 1, 0, 0]  
pizza = [0, 0, 1, 0]  
like = [0, 0, 0, 1]

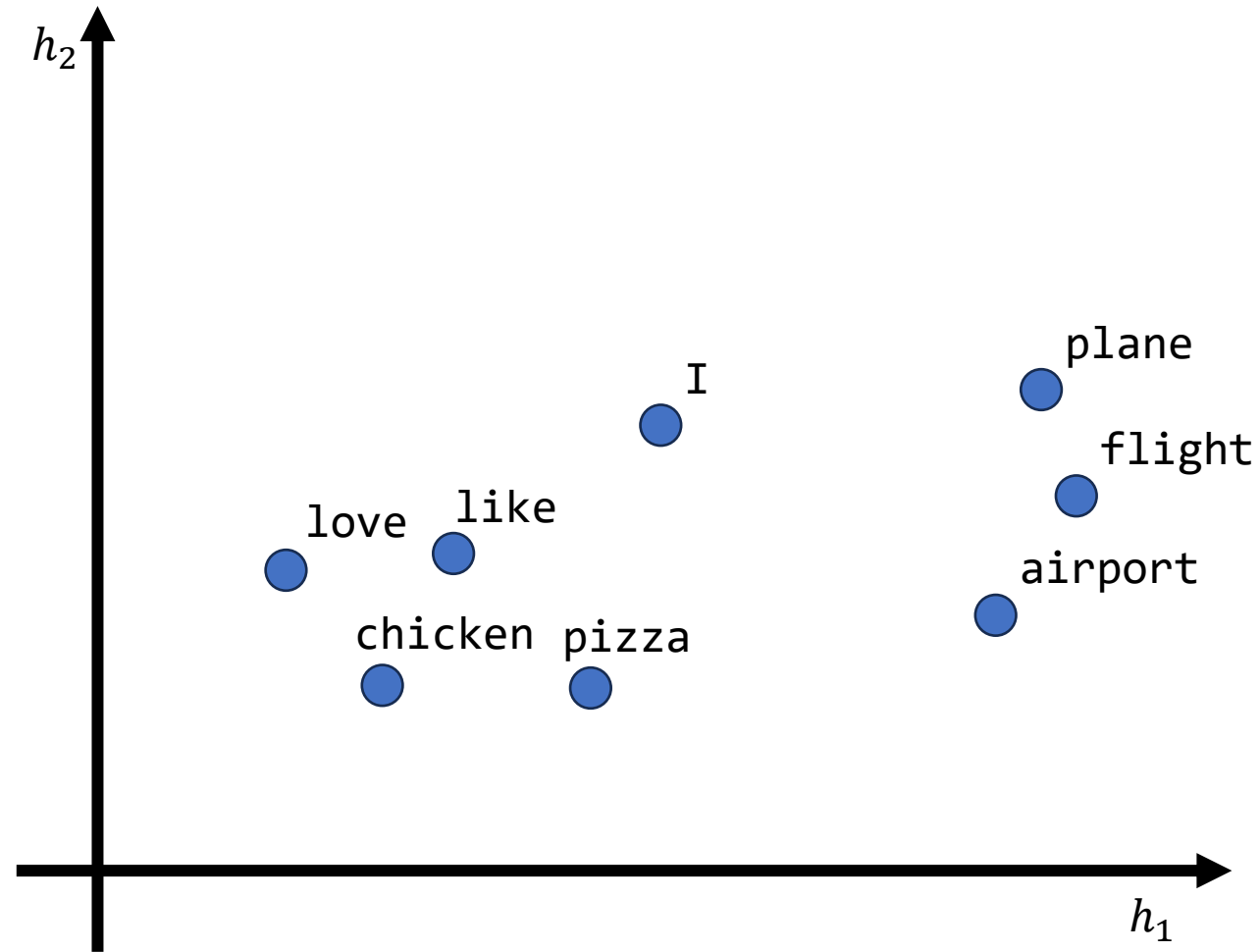


I & pizza		I	pizza
0.5	0	1	0
0	0	0	0
0.5	0	0	1
0	0	0	0

이런 과정을 통해서 CBOW때와 마찬가지로 자연스럽게,

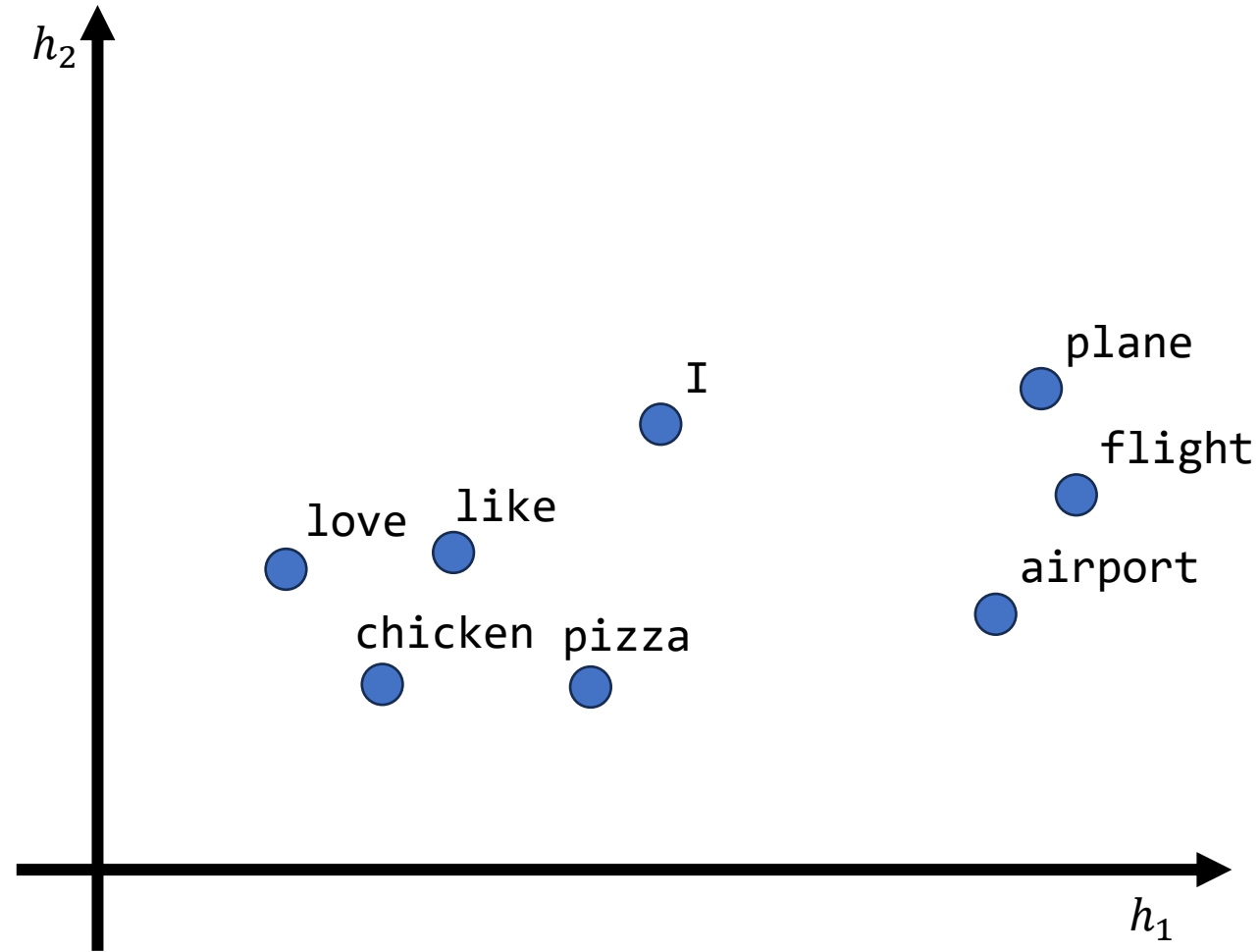


이런 과정을 통해서 CBOW때와 마찬가지로 자연스럽게,

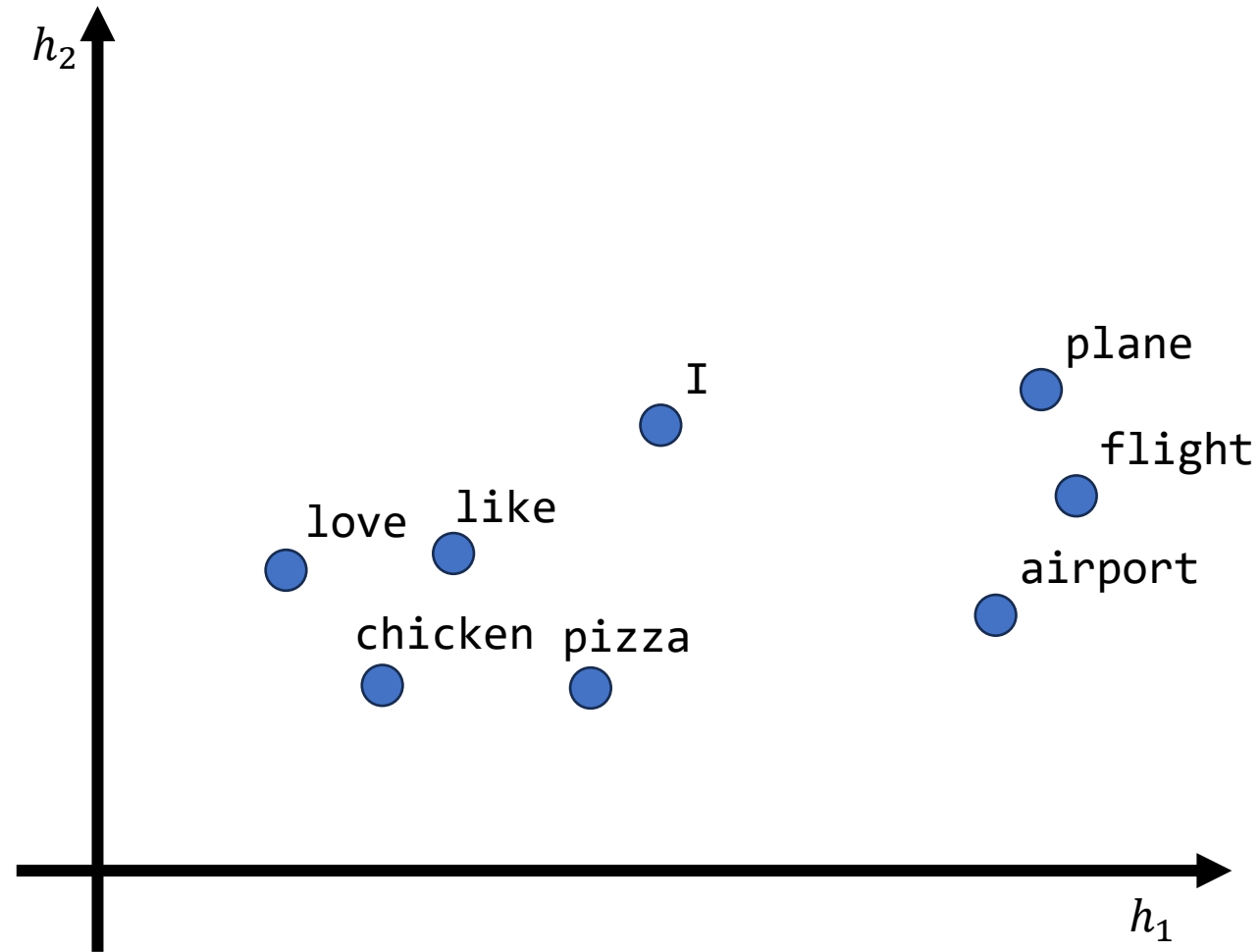




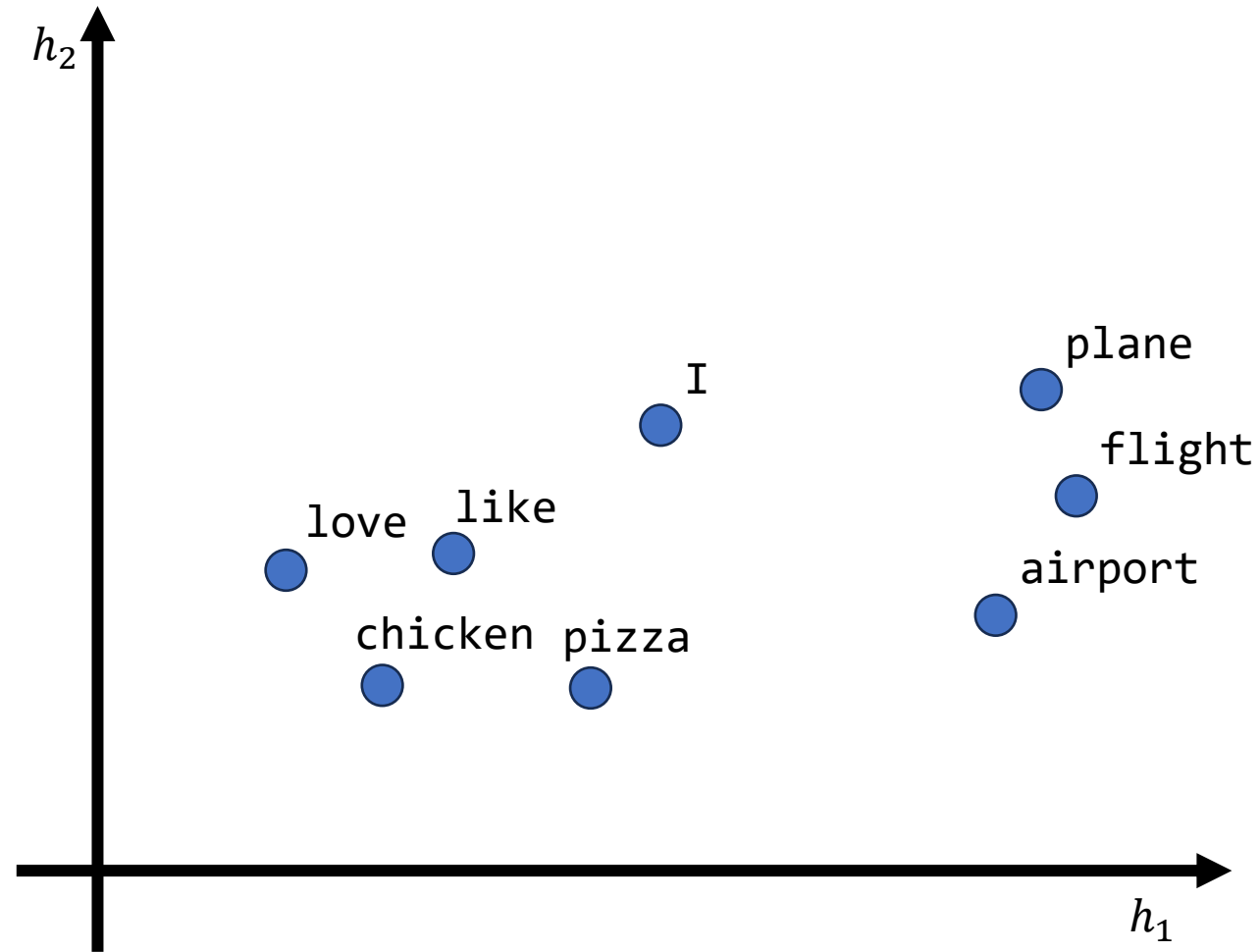
같은 문장에서 자주 쓰이는 표현들은 벡터 공간에서 가까워지게 될 것입니다.



이런 CBOW와 Skip-Gram 학습 방법을 통해, 각 단어의 임베딩 벡터는 해당 단어의 의미와 문맥 context를 반영하게 되고,



벡터 공간에서 서로 가까운 단어 임베딩은 의미적 유사성을 나타내게 되는 것입니다.



오늘 우리는 Word2Vec, 특히 그 내부  
작동 원리와 주요 학습 절차인  
Continuous Bag-of-Words  
(CBOW)와 Skip-Gram에 대해  
알아보았습니다.

이번 영상을 통해 one-hot encoding의 한계를 극복하는 단어 임베딩의 개념을 탐구했고, 이를 통해 어떻게 단어 간 의미적 관계를 벡터 공간에서 표현할 수 있는지에 대해 학습했습니다.

이를 통해 우리는 컴퓨터가 텍스트  
데이터의 의미를 더 잘 이해하고, 다양한  
작업에서 더 높은 성능을 달성할 수 있도록  
도울 수 있습니다.

Word2vec 영상 끝까지 시청해주셔서  
감사드립니다. 다음에도 흥미로운 주제로  
찾아뵙겠습니다.

그럼 다음 시간에 또 만나요!



# 감사합니다!

좋은 하루 되세요!!

이 채널은 여러분의 관심과 사랑이 필요합니다

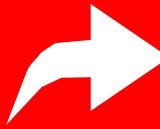
좋아요



댓글



공유



구독



‘좋아요’와 ‘구독’버튼은 강의 준비에 큰 힘이 됩니다!

좋아요



댓글



공유



구독



그리고 영상 자료를 사용하실때는  
출처 '신박AI'를 밝혀주세요





Copyright © 2024 by 신박AI

All rights reserved

본 문서(PDF)에 포함된 모든 내용과 자료는 저작권법에 의해 보호받고 있으며, 신박AI에 의해 제작되었습니다.

본 자료는 오직 개인적 학습 목적과 교육 기관 내에서의 교육용으로만 무료로 제공됩니다.

이를 위해, 사용자는 자료 내용의 출처를 명확히 밝히고,

원본 내용을 변경하지 않는 조건 하에 본 자료를 사용할 수 있습니다.

상업적 사용, 수정, 재배포, 또는 이 자료를 기반으로 한 2차적 저작물 생성은 엄격히 금지됩니다.

또한, 본 자료를 다른 유튜브 채널이나 어떠한 온라인 플랫폼에서도 무단으로 사용하는 것은 허용되지 않습니다.

본 자료의 어떠한 부분도 상업적 목적으로 사용하거나 다른 매체에 재배포하기 위해서는 신박AI의 명시적인 서면 동의가 필요합니다.

위의 조건들을 위반할 경우, 저작권법에 따른 법적 조치가 취해질 수 있음을 알려드립니다.

본 고지 사항에 동의하지 않는 경우, 본 문서의 사용을 즉시 중단해 주시기 바랍니다.

