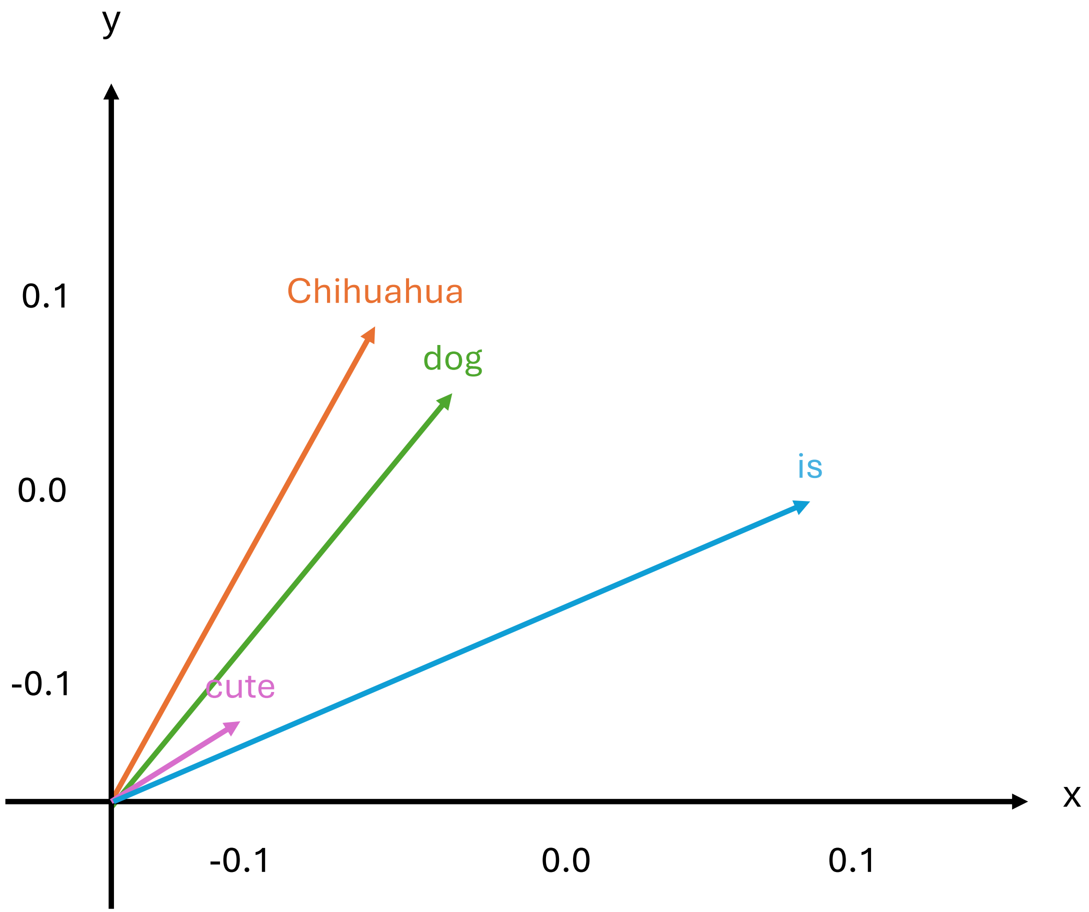
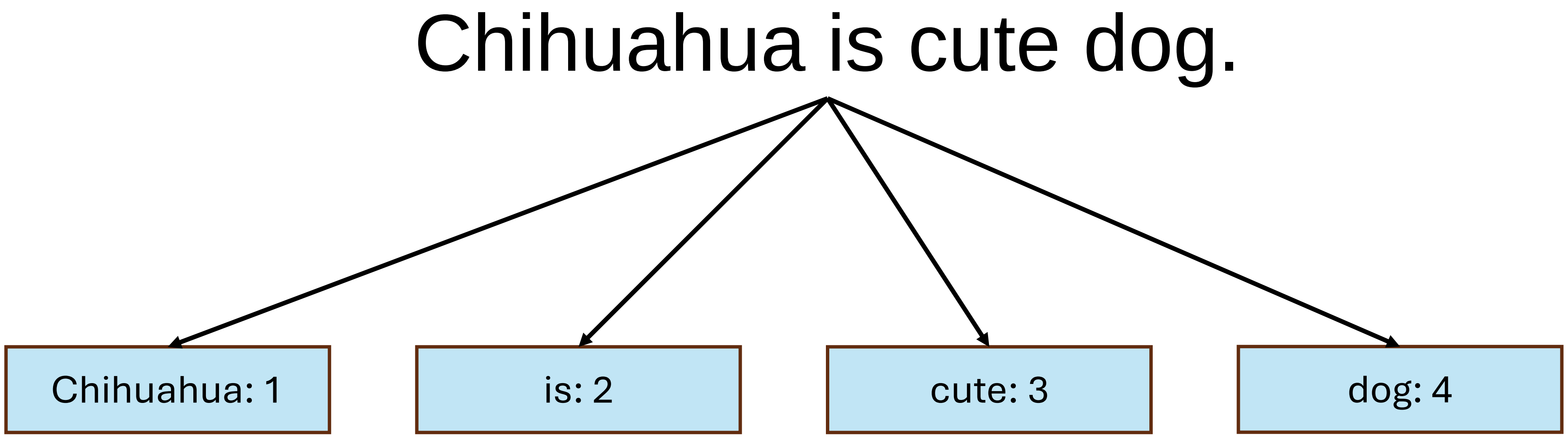
# Word Embedding + Word2Vec






This is called **tokenizer!!!**

# How to change the word to vector ???

Just using word **embedding!!!**

**We’re going to do something called**

**Word2Vec**

# Vocabulary

Chihuahua

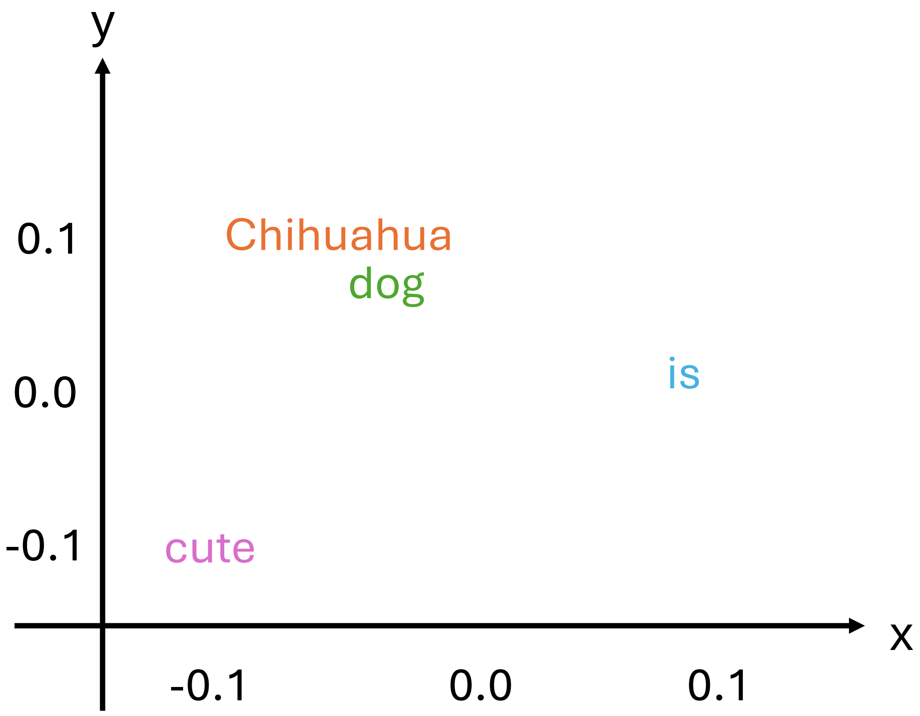
is

cute

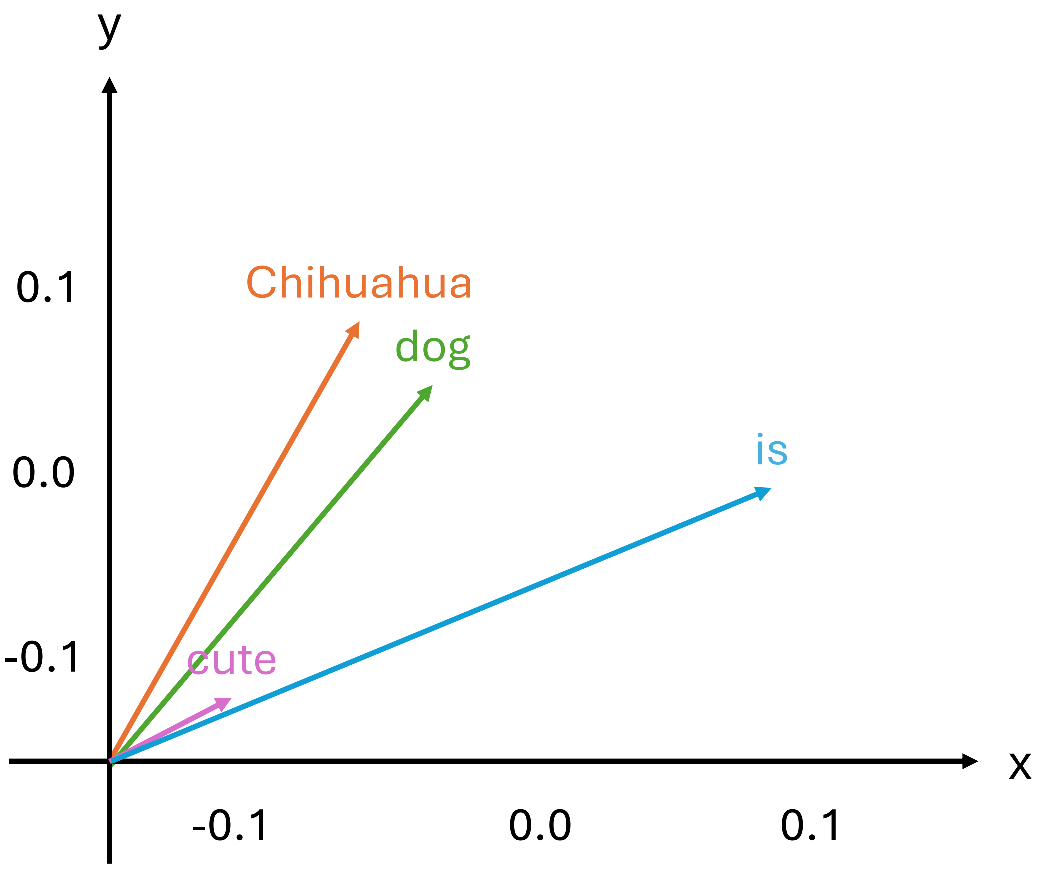
dog

# Vectors

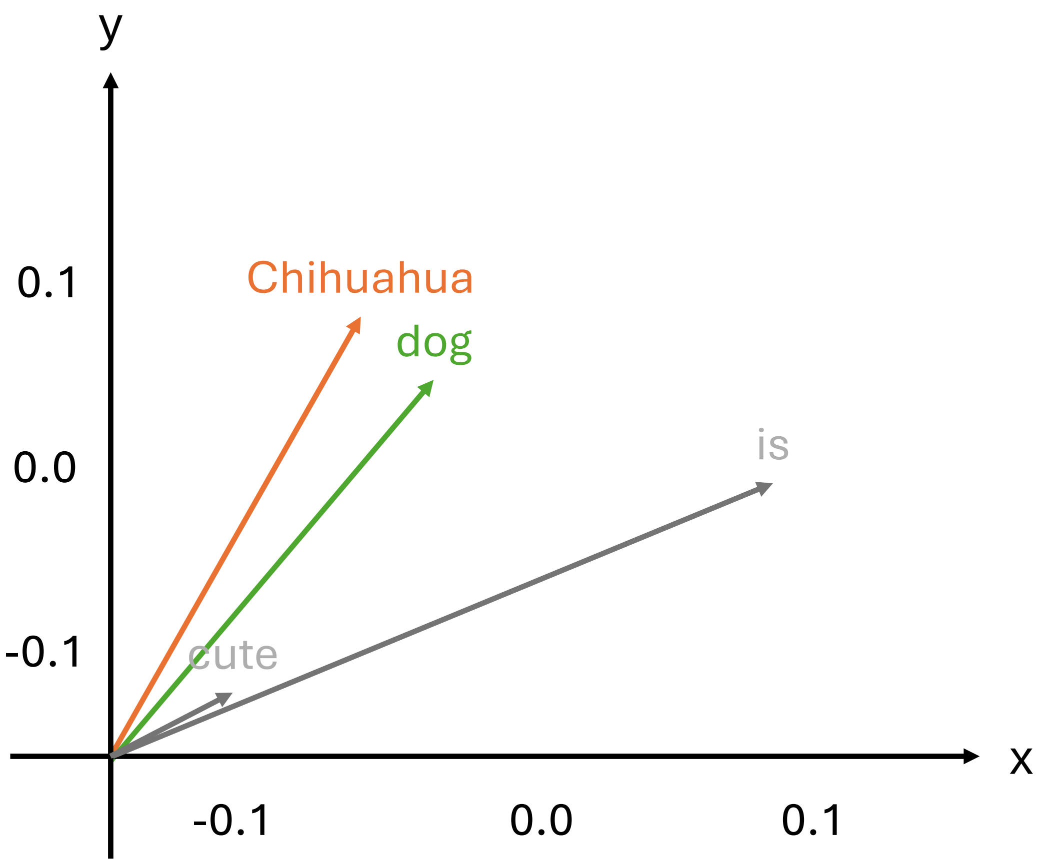
| features | Chihuahua | is | cute | dog |
|---|---|---|---|---|
| x | -0.08 | 0.1 | -0.1 | -0.09 |
| y | 0.1 | 0.02 | -0.1 | 0.09 |



| features | Chihuahua | is | cute | dog |
|---|---|---|---|---|
| x | -0.08 | 0.1 | -0.1 | -0.09 |
| y | 0.1 | 0.02 | -0.1 | 0.09 |

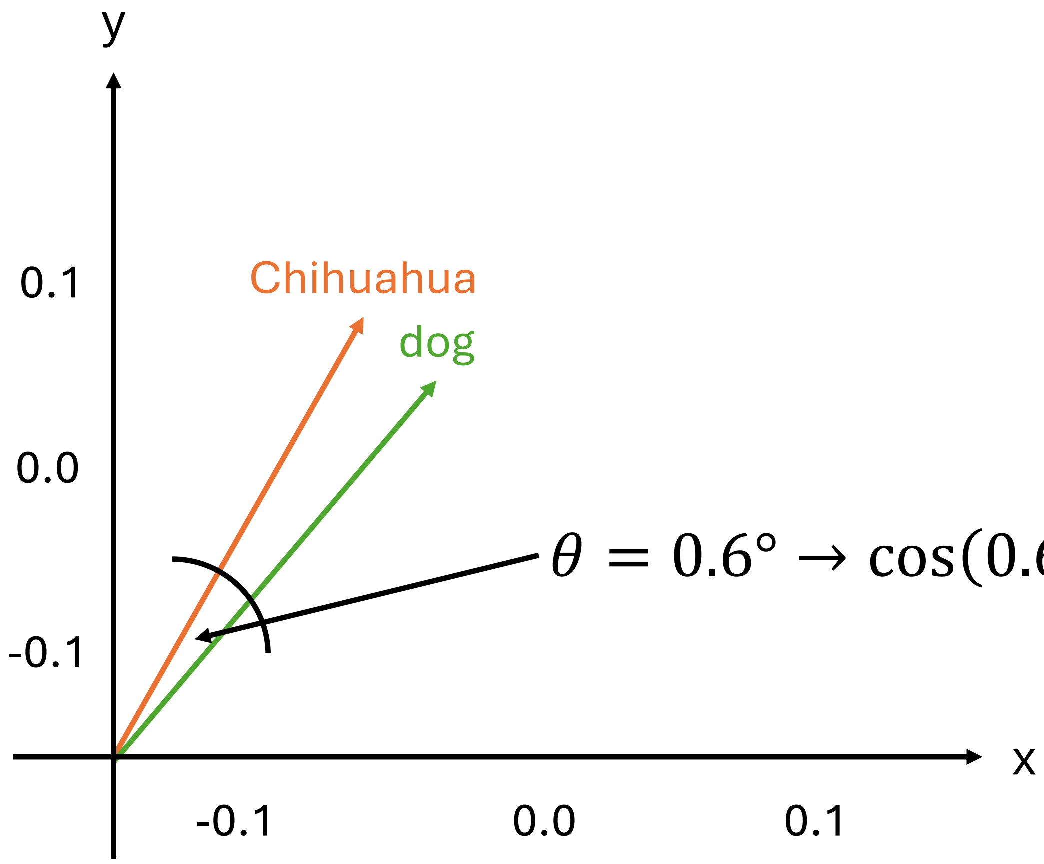


| features | Chihuahua | is | cute | dog |
|---|---|---|---|---|
| x | -0.08 | 0.1 | -0.1 | -0.09 |
| y | 0.1 | 0.02 | -0.1 | 0.09 |



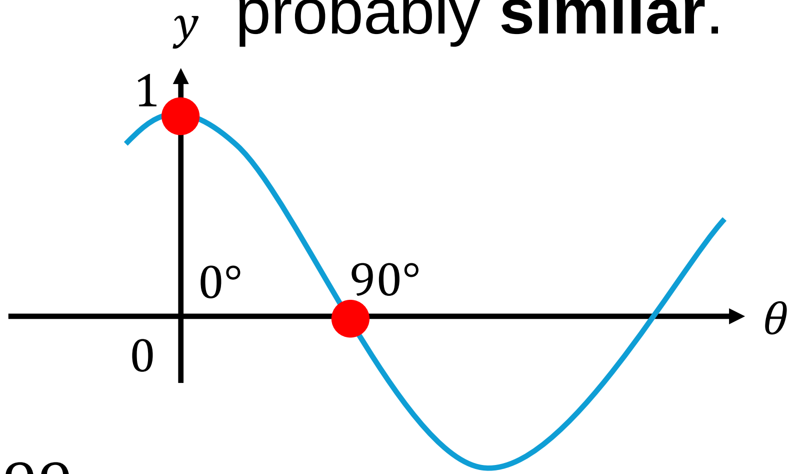

How can we determine whether words are similar or not?

# Apply Cosine Similarity




$\theta = 0.6° \rightarrow \cos(0.6°) = 0.99$

If the $\theta$ is **closer to** $0°$, then it's probably **similar**.



# What if our vocabulary is too huge and the dimension is more than 2D, How to calculate this ???

$$\boldsymbol{Cosine\ Similarity} = \frac{\sum_{i=1}^{n} A_i \cdot B_i}{\sqrt{\sum_{i=1}^{n} {A_i}^2} \cdot \sqrt{\sum_{i=1}^{n} {B_i}^2}}$$

$i \longrightarrow$

| features | Chihuahua | dog |
| --- | --- | --- |
| x | -0.08 | -0.09 |
| y | 0.1 | 0.09 |
| | $A$ | $B$ |

$$\boldsymbol{Cosine\ Similarity} = \frac{\sum_{i=1}^{n} A_i \cdot B_i}{\sqrt{\sum_{i=1}^{n} {A_i}^2} \cdot \sqrt{\sum_{i=1}^{n} {B_i}^2}}$$

$$\frac{\sum_{i=1}^{n} A_i \cdot B_i}{\sqrt{\sum_{i=1}^{n} {A_i}^2} \cdot \sqrt{\sum_{i=1}^{n} {B_i}^2}} = \frac{(-0.08 \cdot -0.09) + (0.1 \cdot 0.09)}{\sqrt{(-0.08)^2 + (0.1)^2} \cdot \sqrt{(-0.09)^2 + (0.09)^2}}$$

$$\frac{\sum_{i=1}^{n} A_i \cdot B_i}{\sqrt{\sum_{i=1}^{n} {A_i}^2} \cdot \sqrt{\sum_{i=1}^{n} {B_i}^2}} = 0.99 \rightarrow \textit{This is the same value as } \mathbf{cos(0.6°)}$$

| features | Chihuahua | is | cute | dog |
|---|---|---|---|---|
| x | -0.08 | 0.1 | -0.1 | -0.09 |
| y | 0.1 | 0.02 | -0.1 | 0.09 |

But, how to get all these value ???

| features | Chihuahua | is | cute | dog |
|---|---|---|---|---|
| Chihuahua | -0.08 | 0.1 | -0.1 | -0.09 |
| Is | 0.1 | 0.02 | -0.1 | 0.09 |
| cute | … | … | … | … |
| dog | … | … | … | … |

**Just apply the DEEP LEARNING**

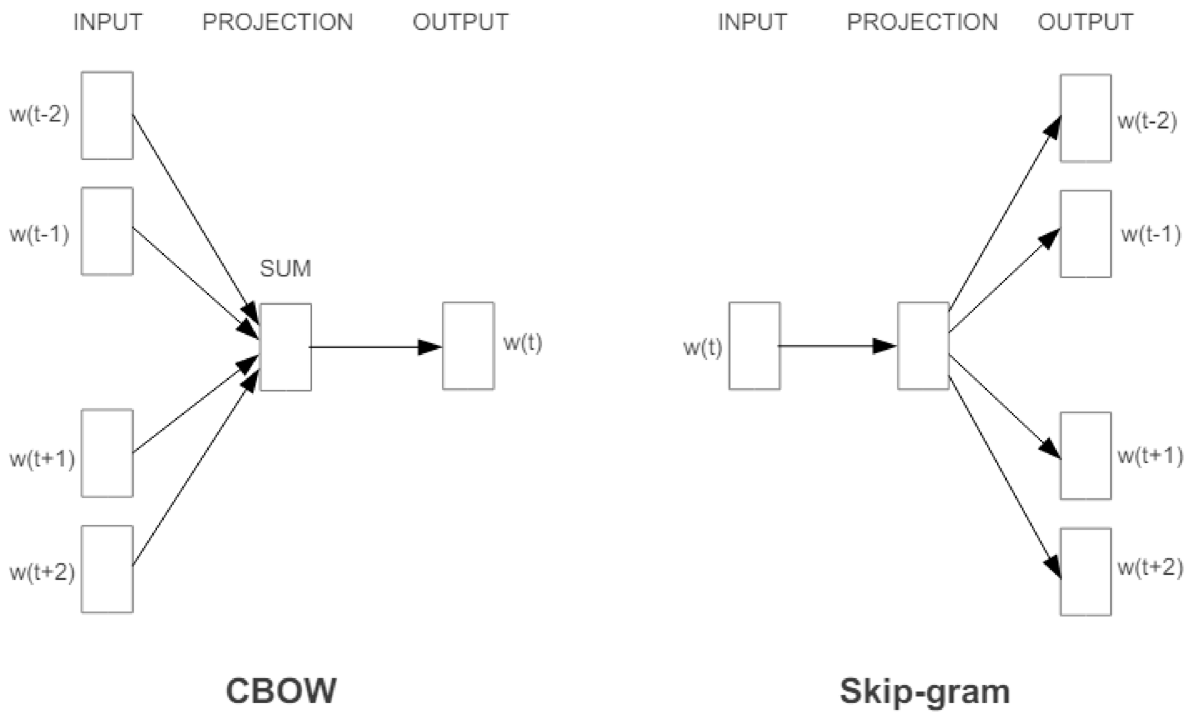
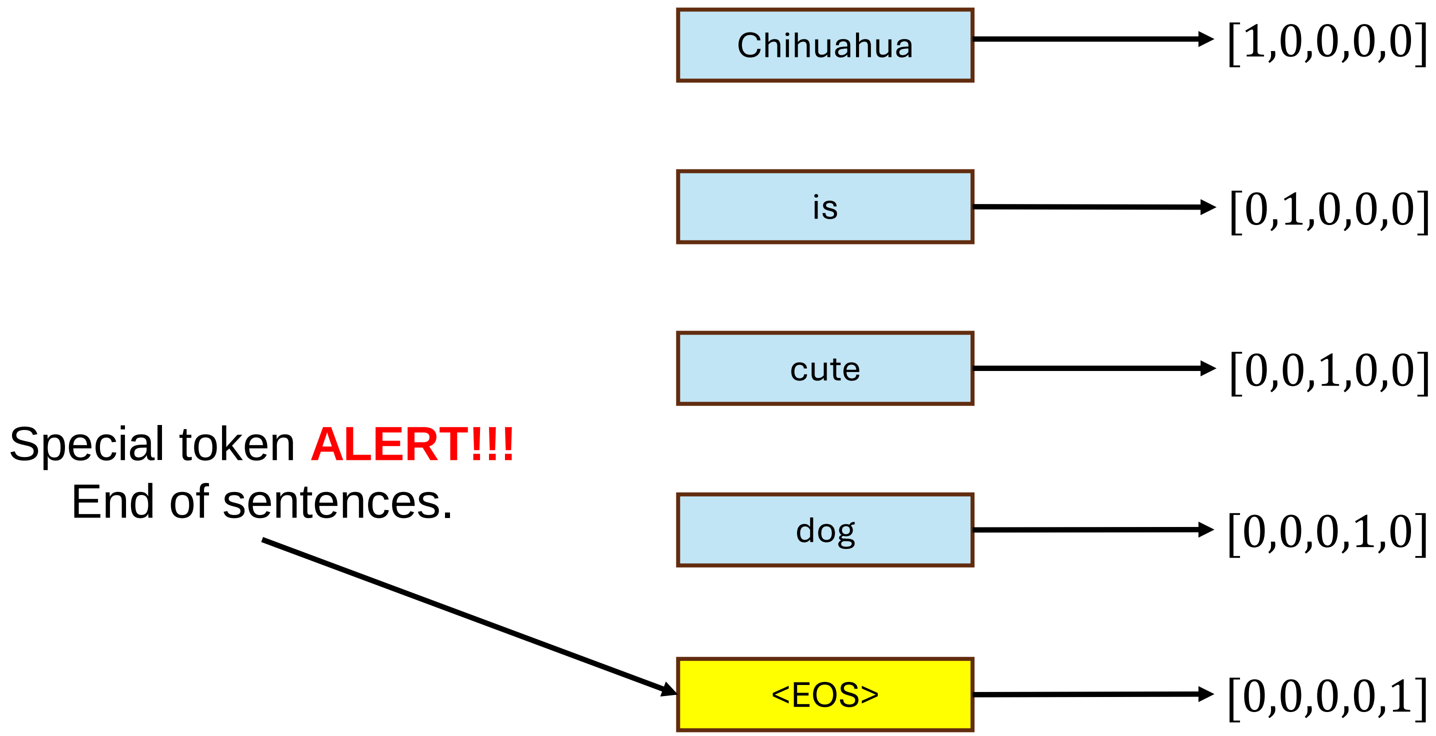
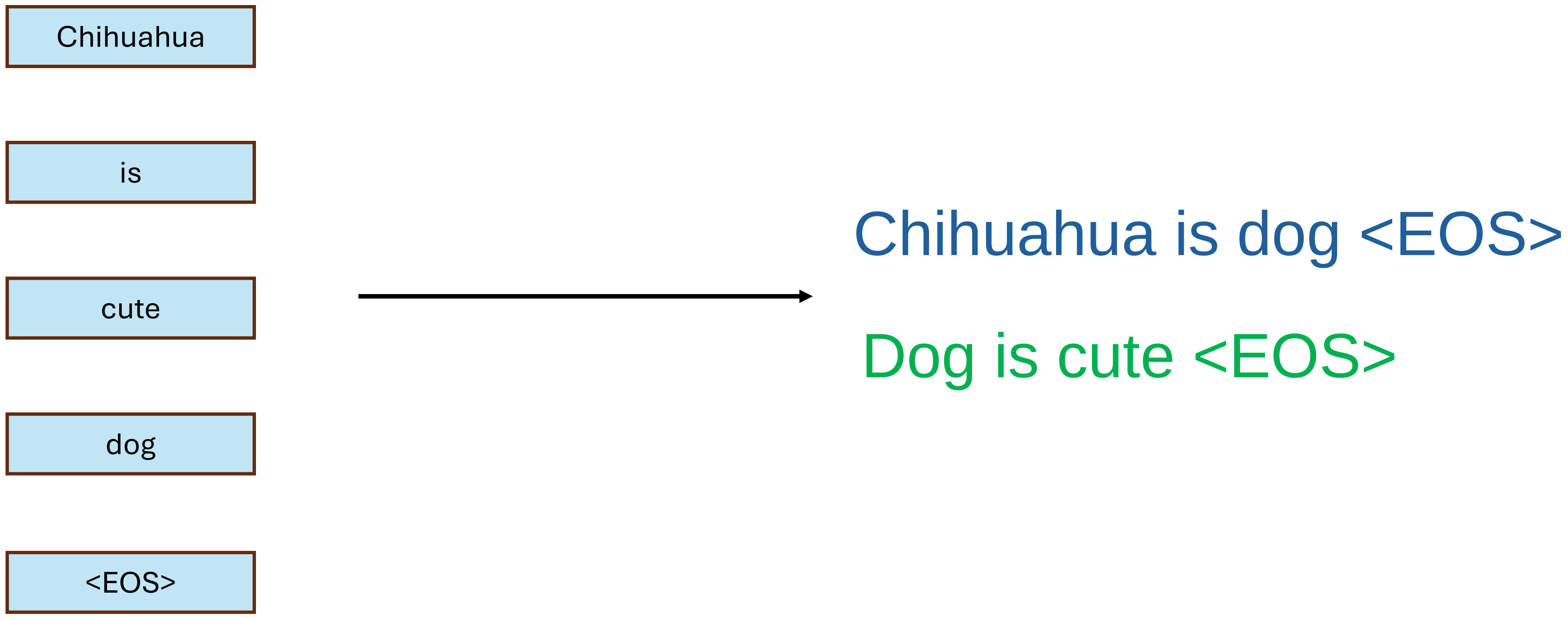



Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

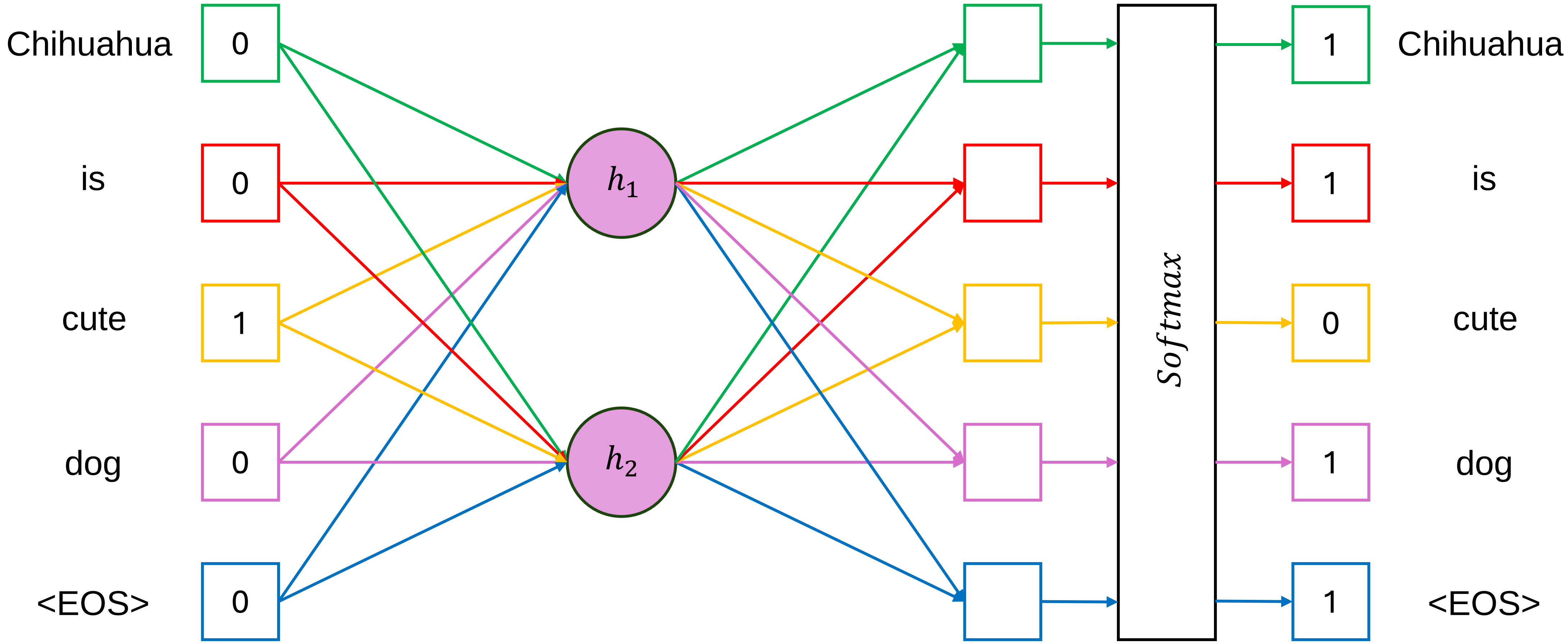
$R$ words from the future of the current word as correct labels. This will require us to do $R \times 2$ word classifications, with the current word as input, and each of the $R + R$ words as output. In the following experiments, we use $C = 10$.

# One-Hot encode!!!

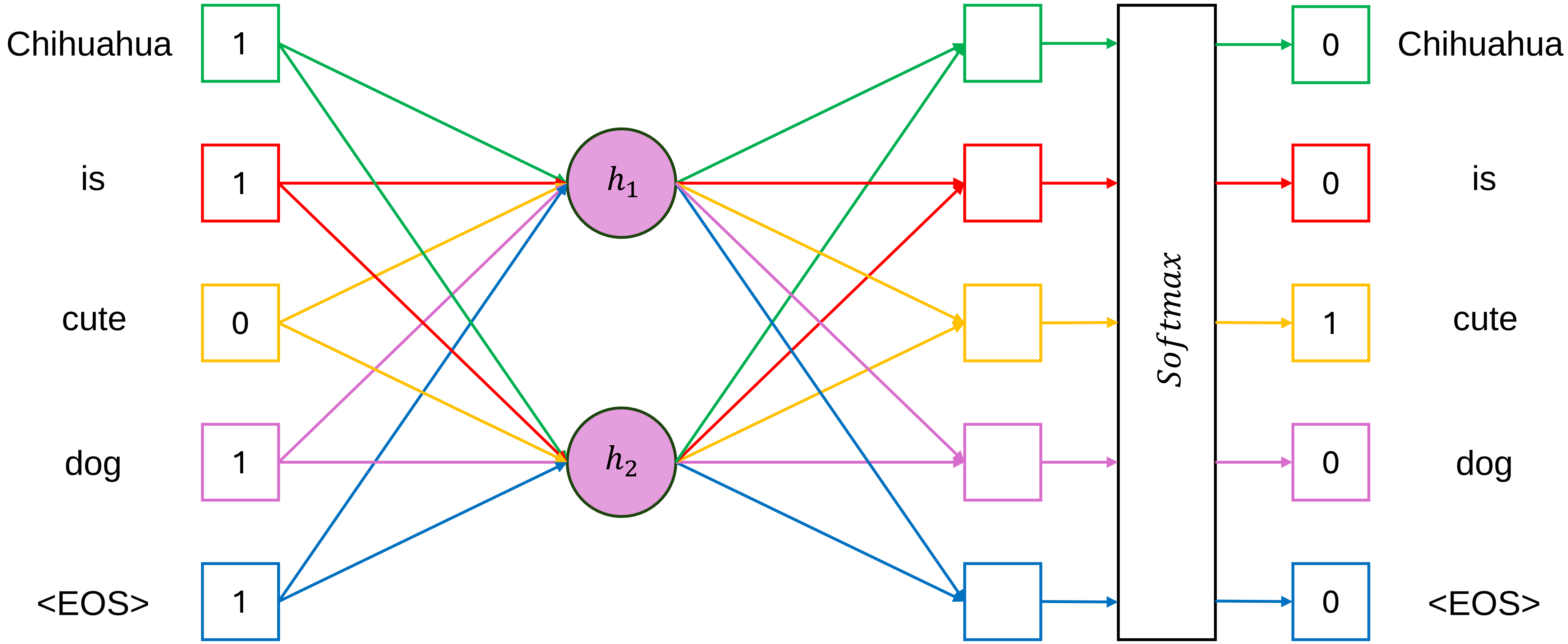



Chihuahua
is
cute
dog
<EOS>
Chihuahua is dog <EOS>
Dog is cute <EOS>

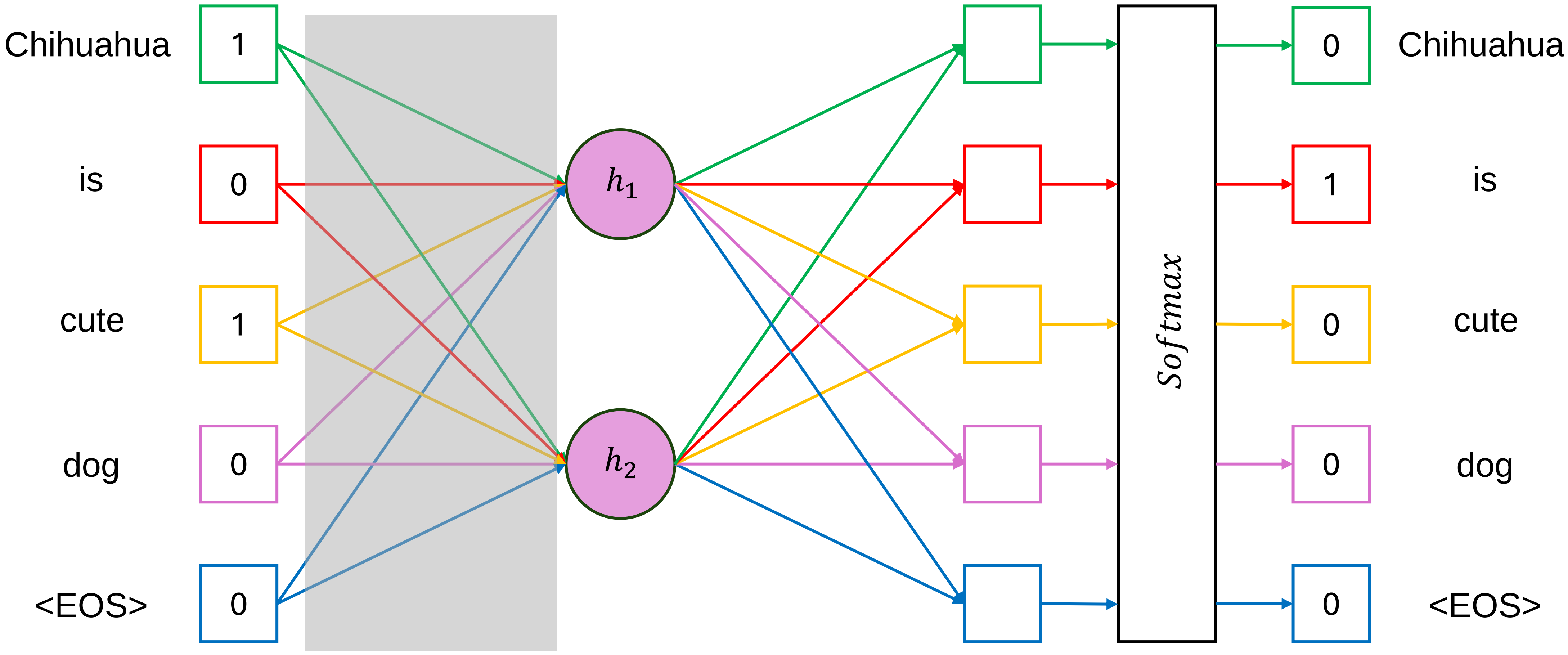
# CBOW (Continuous bag of words)



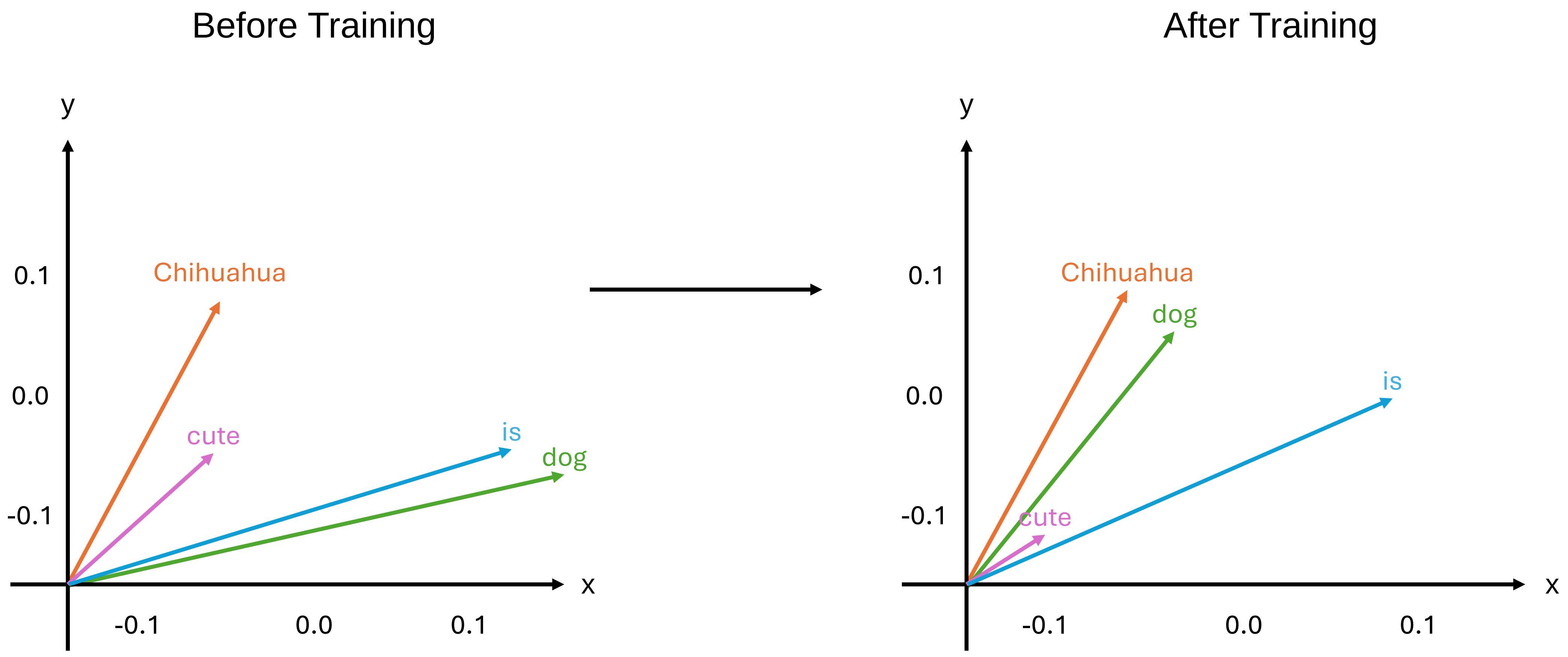
# SKIP-GRAM (Continuous skip grams)

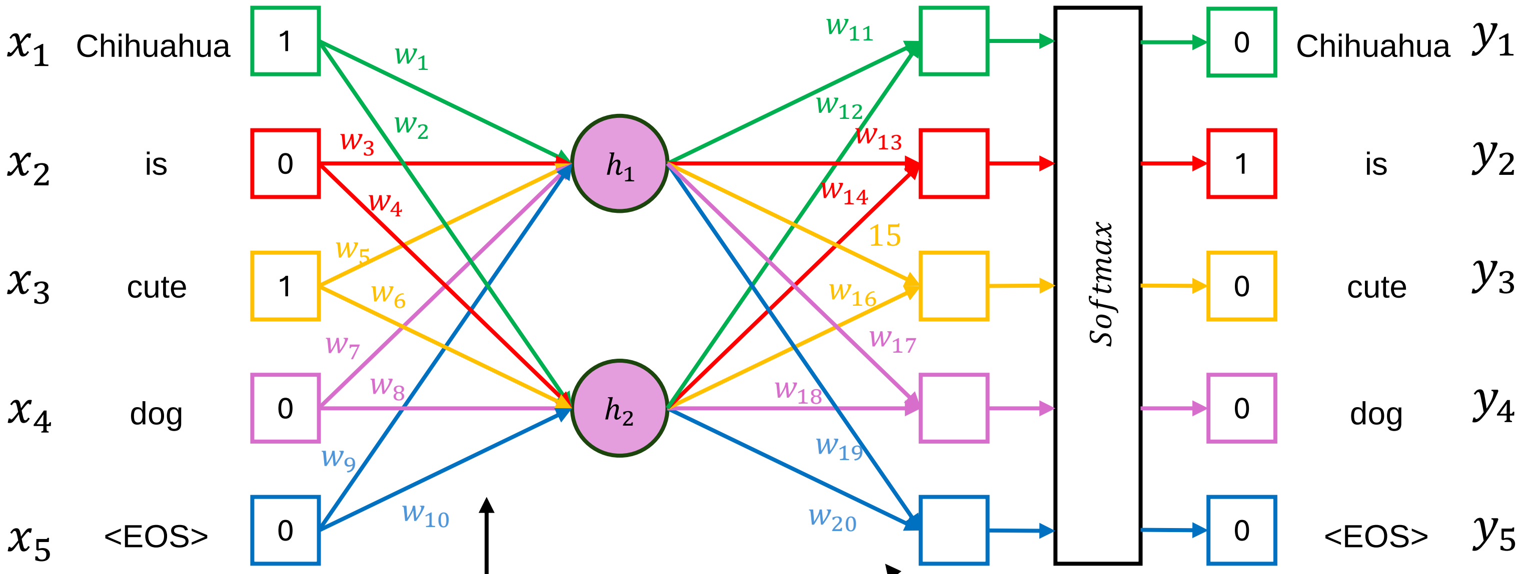


**I’m going to use CBOW**




Train and save the weight
of this state.

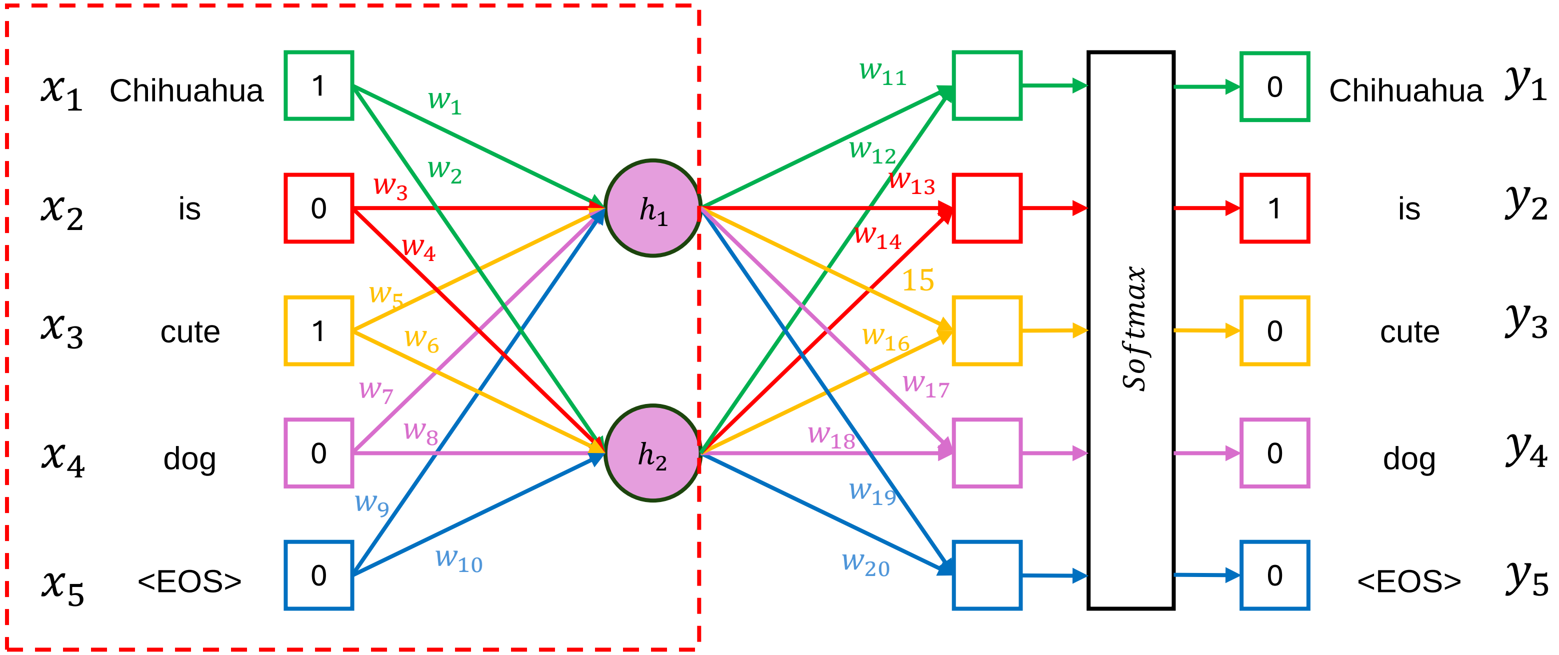

Before Training
After Training
y
x
0.1
0.0
-0.1
-0.1
0.0
0.1
Chihuahua
cute
is
dog
y
x
0.1
0.0
-0.1
-0.1
0.0
0.1
Chihuahua
dog
is
cute




$$y_1 = Softmax(sum_1)$$
$$y_2 = Softmax(sum_2)$$
$$y_3 = Softmax(sum_3)$$
$$y_4 = Softmax(sum_4)$$
$$y_5 = Softmax(sum_5)$$

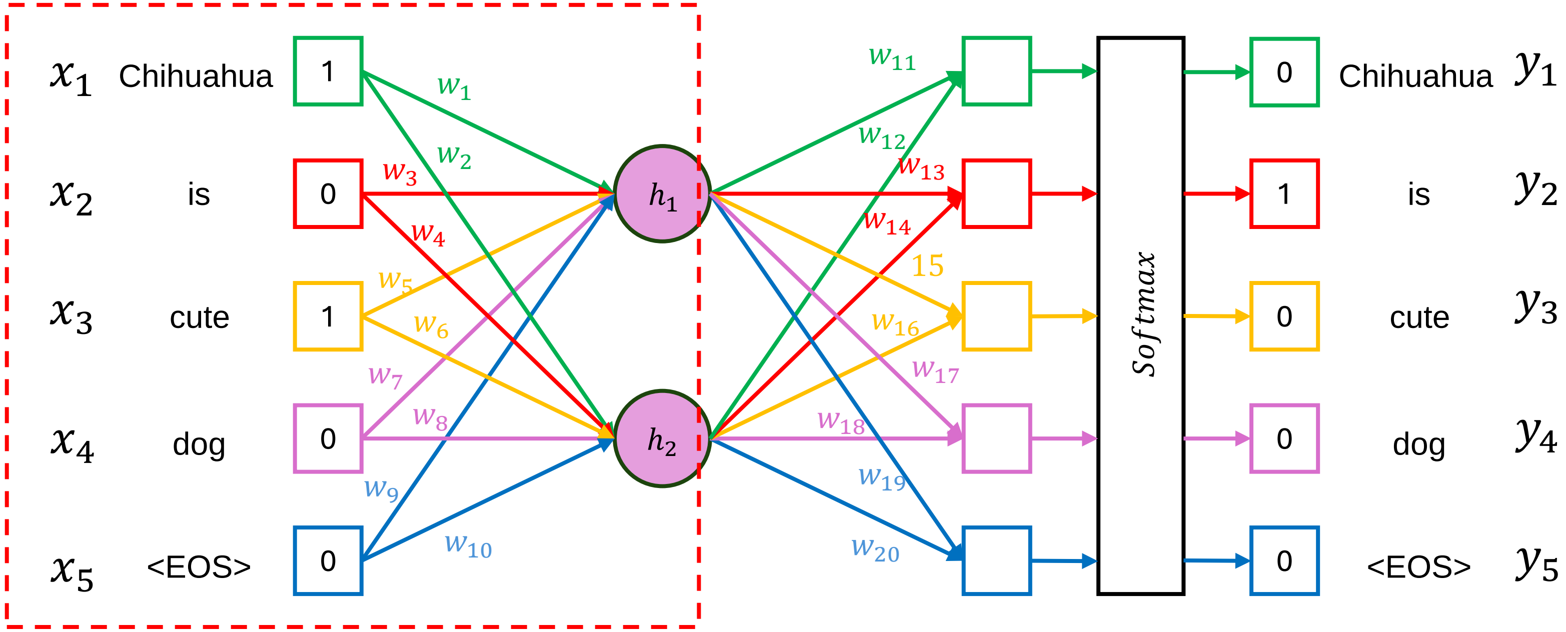
$$h_1 = (x_1w_1 + x_2w_3 + x_3w_5 + x_4w_7 + x_5w_9)$$
$$h_2 = (x_1w_2 + x_2w_4 + x_3w_6 + x_4w_8 + x_5w_{10})$$

$$sum_1 = (h_1w_{11} + h_2w_{12})$$
$$sum_2 = (h_1w_{13} + h_2w_{14})$$
$$sum_3 = (h_1w_{15} + h_2w_{16})$$
$$sum_4 = (h_1w_{17} + h_2w_{18})$$
$$sum_5 = (h_1w_{19} + h_2w_{20})$$




$$\underset{(1\times \cancel{5})}{\begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 \end{bmatrix}} \times \underset{(\cancel{5}\times 2)}{\begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \\ w_5 & w_6 \\ w_7 & w_8 \\ w_9 & w_{10} \end{bmatrix}} = \underset{(1\times 2)}{\begin{bmatrix} x_1w_1 + x_2w_3 + x_3w_5 + x_4w_7 + x_5w_9 & x_1w_2 + x_2w_4 + x_3w_6 + x_4w_8 + x_5w_{10} \end{bmatrix}}$$

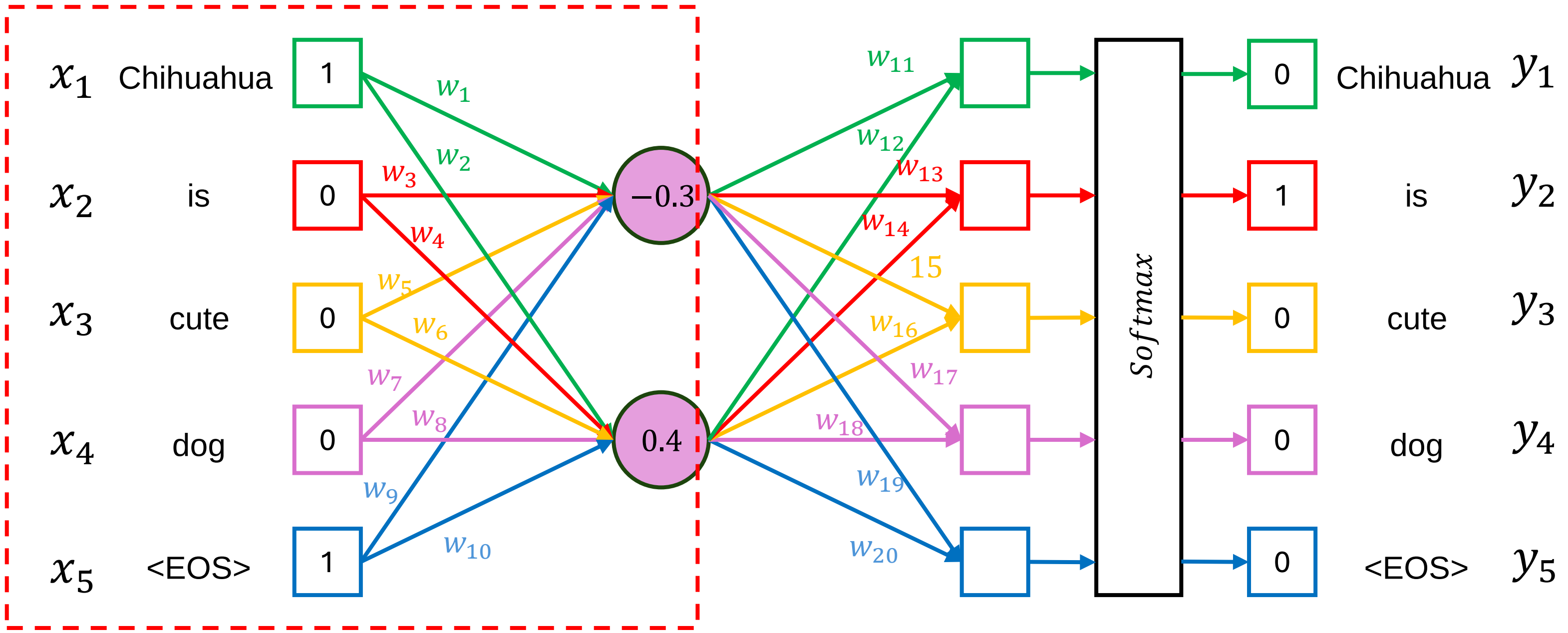
***Dot Product**: Row With Column*




$$[x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5] \times \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \\ w_5 & w_6 \\ w_7 & w_8 \\ w_8 & w_{10} \end{bmatrix} = [h_1 \quad h_2]$$

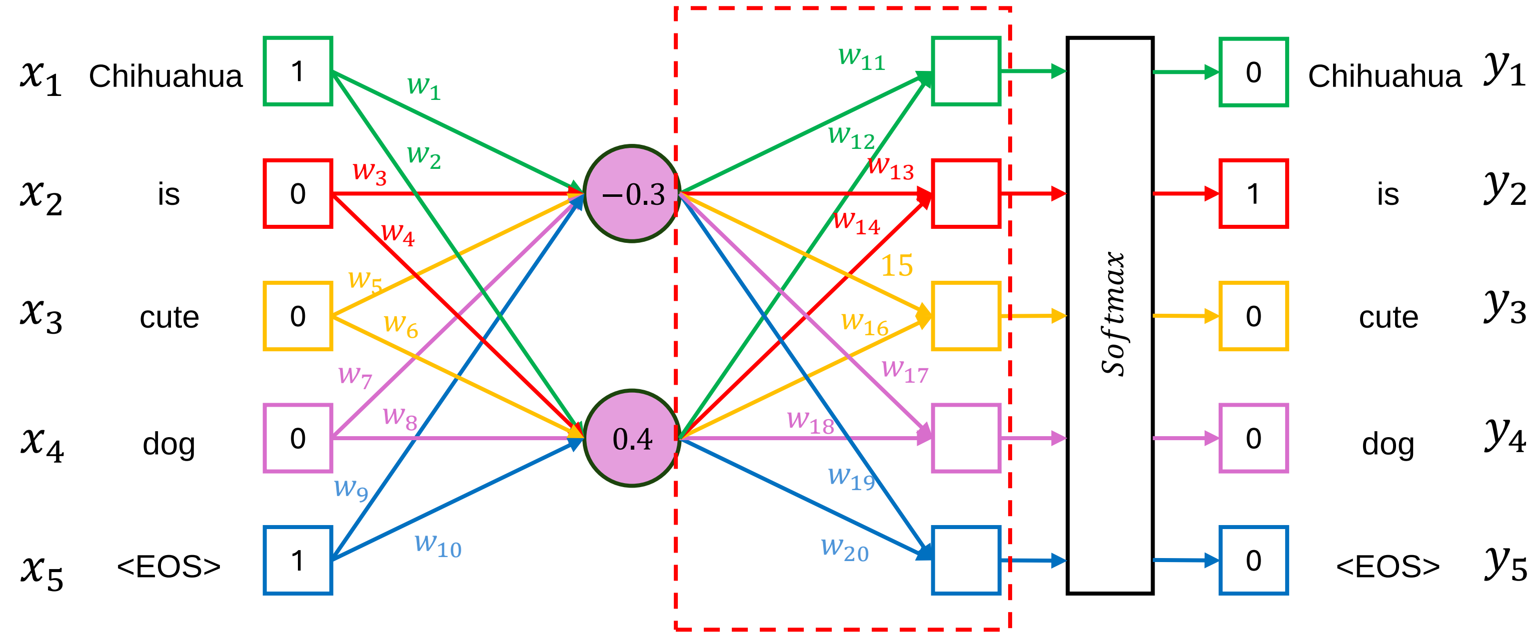
Random all weights

$$[h_1 \quad h_2] = [1 \quad 0 \quad 1 \quad 0 \quad 0] \times \begin{bmatrix} 0.1 & 0.6 \\ 0.3 & -0.2 \\ -0.4 & -0.2 \\ -0.1 & 0.5 \\ 0.7 & -0.1 \end{bmatrix} = [-0.3 \quad 0.4]$$




$$[x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5] \times \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \\ w_5 & w_6 \\ w_7 & w_8 \\ w_8 & w_{10} \end{bmatrix} = [h_1 \quad h_2]$$

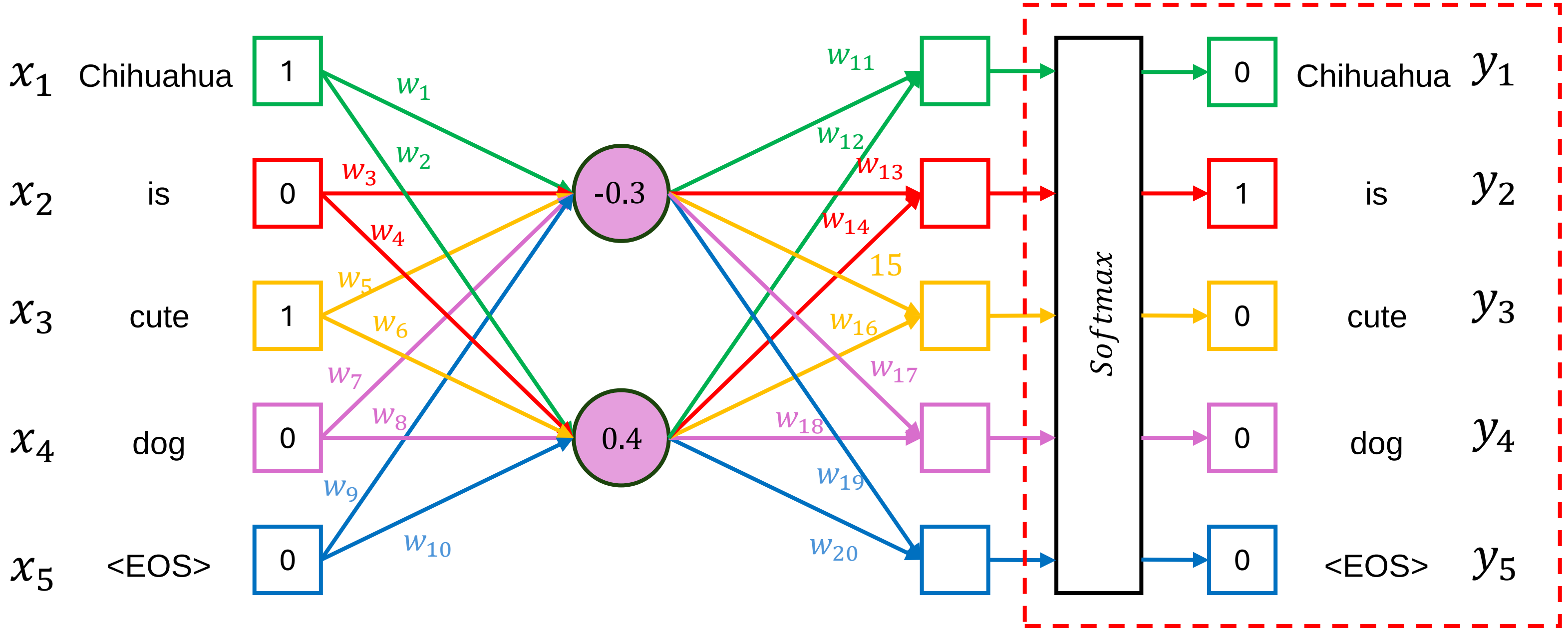
$$[h_1 \quad h_2] = [1 \quad 0 \quad 1 \quad 0 \quad 0] \times \begin{bmatrix} 0.1 & 0.6 \\ 0.3 & -0.2 \\ -0.4 & -0.2 \\ -0.1 & 0.5 \\ 0.7 & -0.1 \end{bmatrix} = [-0.3 \quad 0.4]$$




$$[sum_1 \quad sum_2 \quad sum_3 \quad sum_4 \quad sum_5] = [-0.3 \quad 0.4] \begin{bmatrix} w_{11} & w_{13} & w_{15} & w_{17} & w_{19} \\ w_{12} & w_{14} & w_{16} & w_{18} & w_{20} \end{bmatrix}$$

$$[sum_1 \quad sum_2 \quad sum_3 \quad sum_4 \quad sum_5] = [-0.3 \quad 0.4] \begin{bmatrix} 0.5 & 0.1 & 0.3 & -0.1 & 0.8 \\ 0.4 & -0.2 & -0.1 & 0.9 & 0.3 \end{bmatrix}$$

$$[sum_1 \quad sum_2 \quad sum_3 \quad sum_4 \quad sum_5] = [0.01 \quad -0.11 \quad -0.13 \quad 0.39 \quad -0.12]$$




$$[sum_1 \quad sum_2 \quad sum_3 \quad sum_4 \quad sum_5] = [0.01 \quad -0.11 \quad -0.13 \quad 0.39 \quad -0.12]$$

$$[S_1 \quad S_2 \quad S_3 \quad S_4 \quad S_5] = [0.20 \quad 0.17 \quad 0.17 \quad 0.29 \quad 0.17]$$

Argmax

Input Output

$$[S_1 \quad S_2 \quad S_3 \quad S_4 \quad S_5] = [0 \quad 0 \quad 0 \quad 1 \quad 0] \longrightarrow (Chihuahua, cute) \rightarrow dog$$

$\boldsymbol{But\ we\ need\ this\ answer}$: $[S_1 \quad S_2 \quad S_3 \quad S_4 \quad S_5] = [0 \quad 1 \quad 0 \quad 0 \quad 0] \rightarrow is$

# Loss Calculation

$$CE = -\sum_{i=1}^{n} Observed \cdot \log(Softmax_i)$$

$$CE = -\log(0.17)$$

$$CE = 0.76$$

# Then, Backpropagation

$$w_{new} = w_{current} - \alpha \sum_{i=1}^{n} \frac{\partial CE_i}{\partial w}$$

To minimize **“Cross Entropy”**

# Time to code in

PyTorch

# 1. Import Library

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import numpy as np
```

[1]

```
device = ("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using {device} device")
```

[2]

```
Using cuda device
```

Import library

Declare to using **GPU,** in case GPU nor found it's going to use CPU instead.

# 2. Do Tokenizer

```python
sentence = "Chihuahua is cute dog"
```

[3] ✓ 0.0s

Train data

```python
def tokenizer(sentence):
    vocabulary = sentence.split()
    vocabulary.append("<EOS>")

    return vocabulary
```

[4] ✓ 0.0s

Tokenizer function

Do tokenizer

```python
tokens = tokenizer(sentence)
print(tokens)
```

[8] ✓ 0.0s

Output

```
['Chihuahua', 'is', 'cute', 'dog', '<EOS>']
```

# 3. Convert Tokens to One Hot Vector

```python
def vocabulary_one_hot_encoder(tokens):
    results = []

    for i, _ in enumerate(tokens):
        one_hot_encode_vector = [0. for i in range(len(tokens))]

        for j in range(i + 1):
            one_hot_encode_vector[j] = 1. if j == i else 0.

        results.append(one_hot_encode_vector)

    return results
```

[5] 0.0s

One hot vector encoder function

```python
training_x = vocabulary_one_hot_encoder(tokens)
print(training_x)
```

[9] 0.0s

```
[[1.0, 0.0, 0.0, 0.0, 0.0], [0.0, 1.0, 0.0, 0.0, 0.0], [0.0, 0.0, 1.0, 0.0, 0.0], [0.0, 0.0, 0.0, 1.0, 0.0], [0.0, 0.0, 0.0, 0.0, 1.0]]
```

Output

# 4. Make CBOW Training Data

```python
def create_cbow_pairs(sequence, window_size):
    cbow_pairs = []

    for i in range(len(sequence)):
        context = []

        for j in range(-window_size, window_size + 1):
            if i + j >= 0 and i + j < len(sequence) and j != 0:
                context.append(sequence[i + j])

        if context:
            context = np.sum(context, axis=0) / len(context)
            cbow_pairs.append(context)

    return cbow_pairs
```
[6] ✓ 0.0s

CBOW pairs maker function

```python
CONTEXT_SIZE = 2

training_y = create_cbow_pairs(training_x, CONTEXT_SIZE)
print(training_y)
```
[10] ✓ 0.0s

```
[array([0.  , 0.5, 0.5, 0.  , 0.  ]), array([0.33333333, 0.        , 0.33333333, 0.33333333, 0.
```

Context size it to define the words surrounding size.

Example: CONTEXT_SIZE=2
[0.25, 0.25, 0., 0.25, 0.25]

# 5. Declaring Deep Learning The Model

```python
class CBOWModeler(nn.Module):
    def __init__(self, volcabulary_size, embedding_dim):
        super(CBOWModeler, self).__init__()
        self.linear1 = nn.Linear(volcabulary_size, embedding_dim, bias=False)
        self.linear2 = nn.Linear(embedding_dim, volcabulary_size, bias=False)

    def forward(self, x):
        out = self.linear1(x)
        out = self.linear2(out)
        return F.softmax(out, dim=-1)

    def save_embedding_model(self):
        params = self.linear1.state_dict()
        torch.save(params, "./embedding_model.pt")
```

[11] ✓ 0.0s

Layers

Forward propagation

Save weights for linear1 layer.

# 6. Training Setup

```python
EMBEDDING_DIM = 2

losses = []
loss_function = nn.CrossEntropyLoss()
model = CBOWModeler(len(tokens), EMBEDDING_DIM).to(device)
optimizer = optim.Adam(model.parameters(), lr=0.001)

print(model)
```

[12] ✓ 0.7s

```
CBOWModeler(
  (linear1): Linear(in_features=5, out_features=2, bias=False)
  (linear2): Linear(in_features=2, out_features=5, bias=False)
)
```

Loss Function

Model.to(device) means to using GPU.

Optimizer

# 7. Training Step

```python
for epoch in range(100):
    total_loss = 0

    for x, y in zip(training_x, training_y):
        x, y = torch.tensor(x).to(device), torch.tensor(y).to(device)

        model.zero_grad()

        y_hat = model(x)

        loss = loss_function(y, y_hat)

        loss.backward()
        optimizer.step()

        total_loss += loss.item()
    losses.append(total_loss)

model.save_embedding_model()
```

[13] ✓ 0.6s

Don’t forget to change all tensors to .to(device) while training with GPU.

Stop if zero of gradient

Loss comparing

Backpropagation

Stack loss to see performance of training

Save model

# 7. Losses Inspection

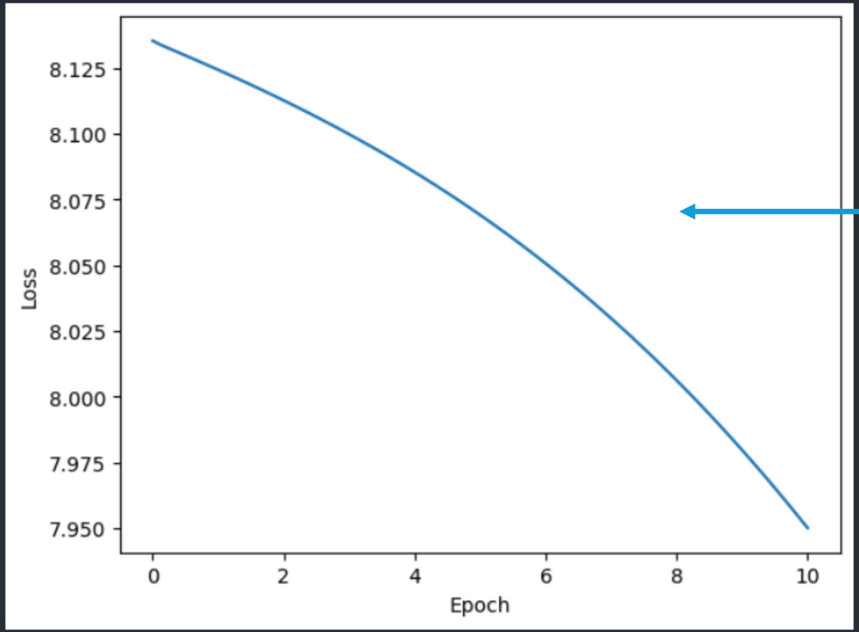
```
def plot_losses(ax, t, losses):
    ax.plot(t, losses)
    ax.set_xlabel("Epoch")
    ax.set_ylabel("Loss")
```

[15] ✓ 0.0s

```
fig, ax = plt.subplots()
plot_losses(ax, np.linspace(0., 10., len(losses)), losses)
```

[16] ✓ 0.1s

...




Looks Bad ☹

# View Result

```
for x, y in zip(training_x, training_y):
    # Get predicted vector
    pred = model(torch.tensor(x))

    # Get the argmax index
    argmax_idx = torch.argmax(pred, keepdim=True)

    # Create a one-hot encoded tensor
    anwser_vector = F.one_hot(argmax_idx, num_classes=pred.size(-1)).int()
    anwser_list = anwser_vector.tolist()

    question = word_decoder(x, tokens)
    answer = word_decoder(anwser_list[0], tokens)
    print(question, answer)
```

[17] ✓ 0.0s

```
Chihuahua is
is cute
cute dog
dog cute
<EOS> dog
```

Argmax is only output as index of output vector

Convert one hot vector of result of training to human readable

```
def word_decoder(anwser_list, vocabulary):
    idx = anwser_list.index(max(anwser_list))
    return vocabulary[idx]
```

[7] ✓ 0.0s