# Lect 3:Word2vec

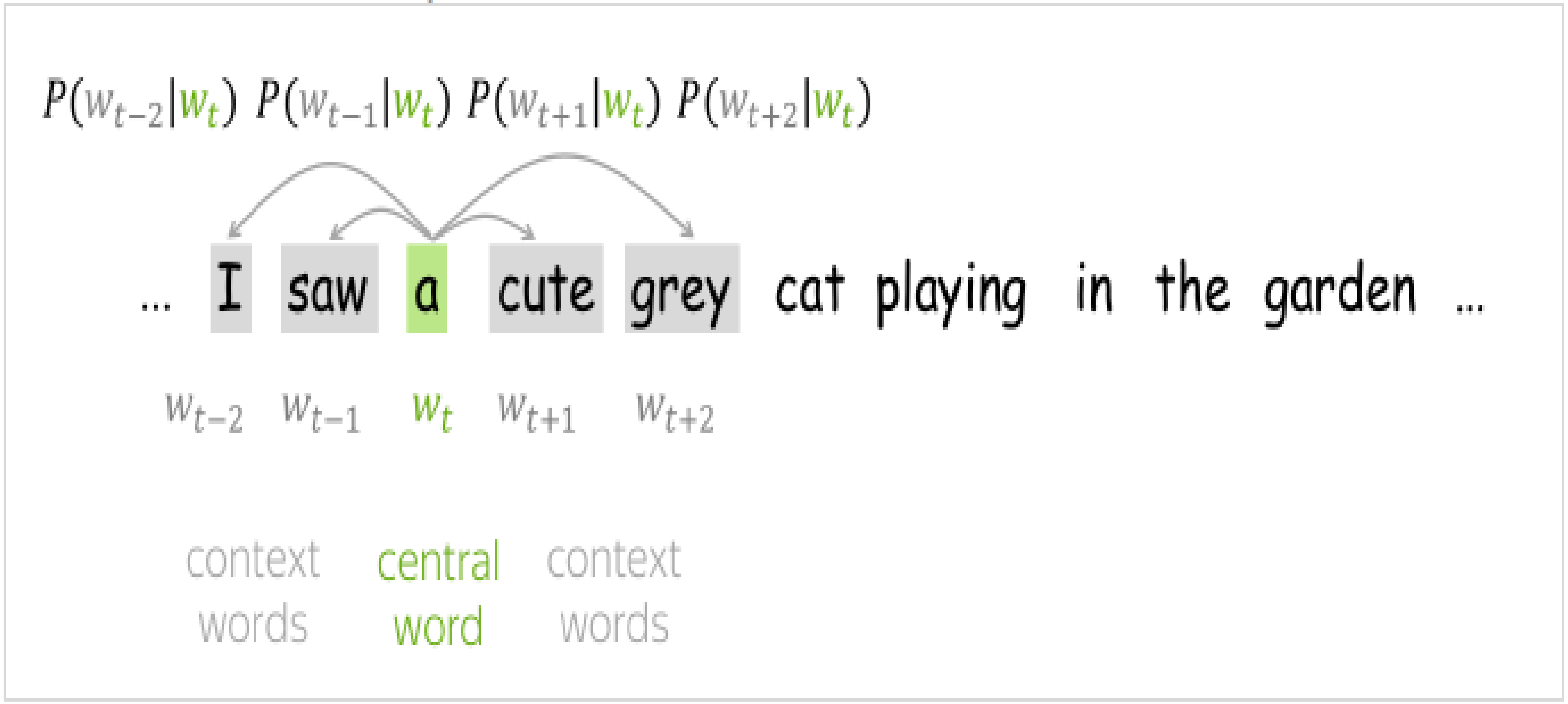
## Predictive models

- **Predictive models** directly try to <u>predict</u> a word from its neighbors in terms of learned small, dense embedding vectors (considered parameters of the model).

- **Neural-network-inspired models:**
  - **word2vec** (Mikolov et al., 2013)
  - **FastText** (Bojanowski et al., 2016)

## Word2Vec: A Prediction-Based Method

- take a huge text corpus;
- go over the text with a sliding window, moving one word at a time. At each step, there is a central word and context words (other words in this window);
- for the central word, compute probabilities of context words (or vice versa);
- adjust the vectors to increase these probabilities.



$P(w_{t-2}|w_t)$ $P(w_{t-1}|w_t)$ $P(w_{t+1}|w_t)$ $P(w_{t+2}|w_t)$
... I saw a cute grey cat playing in the garden ...
$w_{t-2}$ $w_{t-1}$ $w_t$ $w_{t+1}$ $w_{t+2}$
context words
central word
context words

## Main idea (neural network word embeddings)

- Similar to language modeling but predicting **context**, rather than next word.

$$P(context|word) \quad \leftarrow \text{maximize}$$

- In practice:

$$J = 1 - P(context|word) \quad \leftarrow \text{minimize}$$

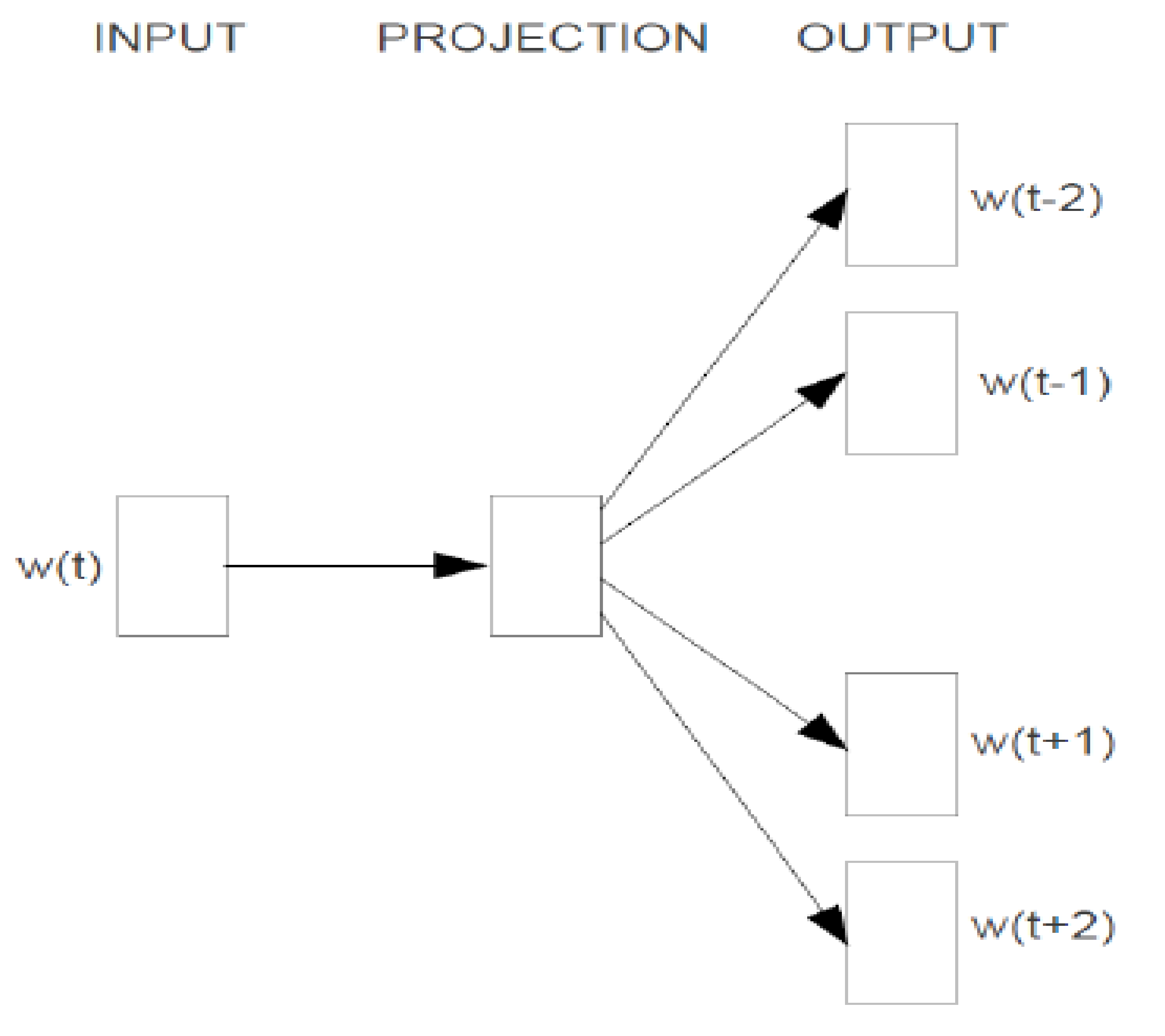
- We **adjust the vector representations of words** to minimize the loss.

## Two basic architectures

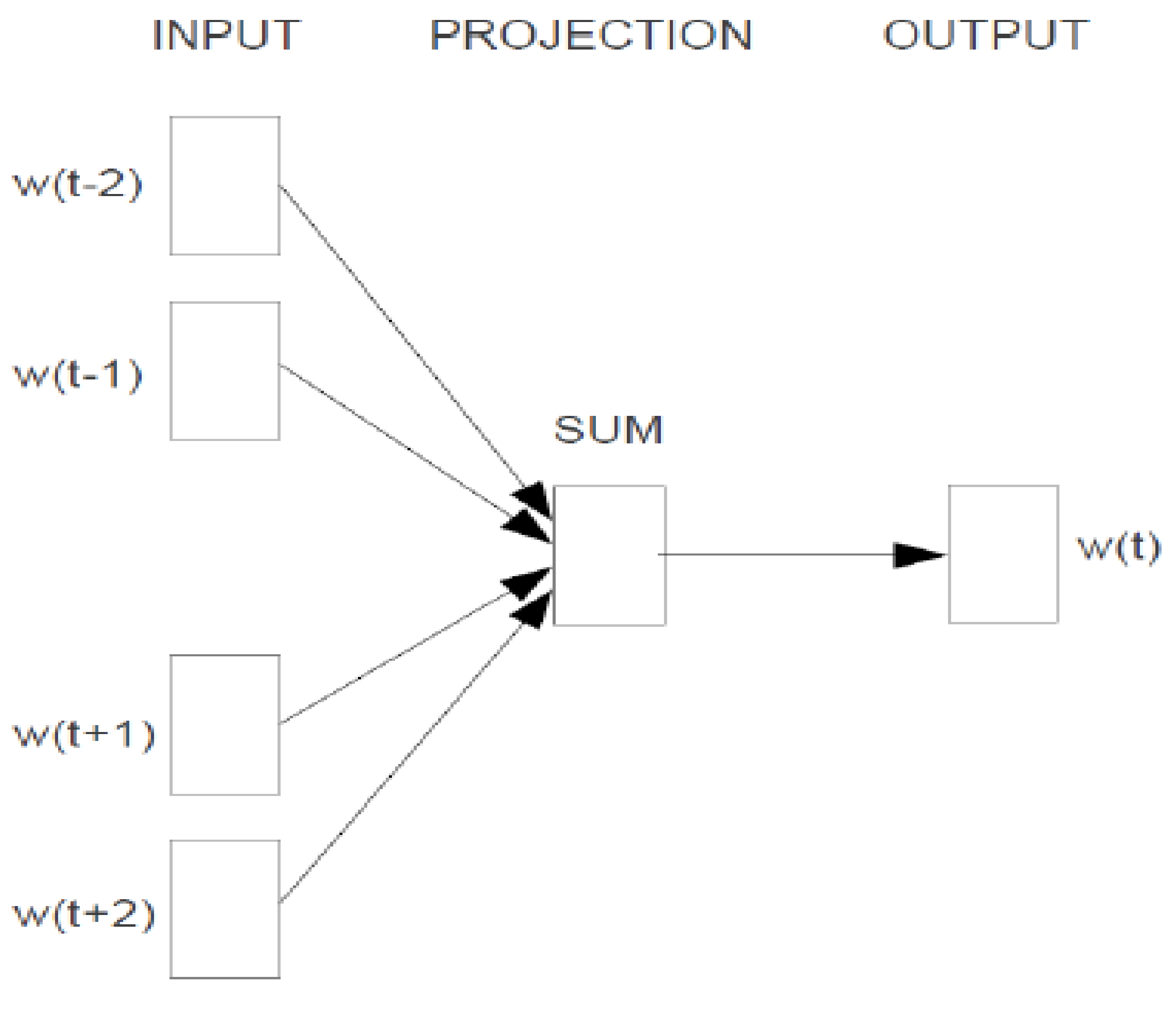
- There are **two architectures** used by word2vec:
  - **Skip-gram**
  - Continuous bag-of-words (**CBOW**)

Algorithms for producing word vectors

## word2vec Architecture



**Skip-gram**

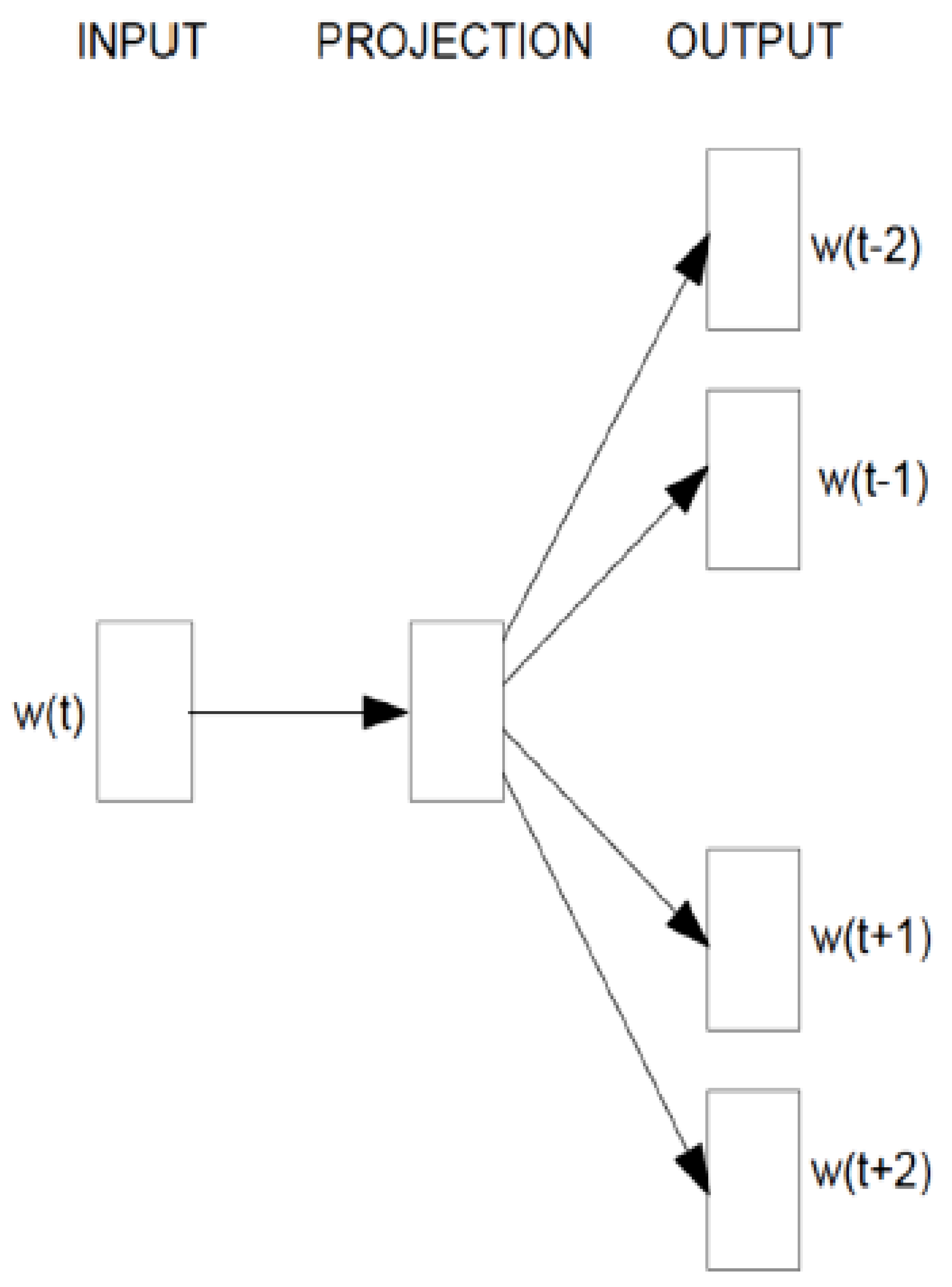


**CBOW**

# Skip-gram model

- **Predict the surrounding words** (context words), based on the current word (the center word).

- Mikolov et. al. 2013. Efficient Estimation of Word Representations in Vector Space.

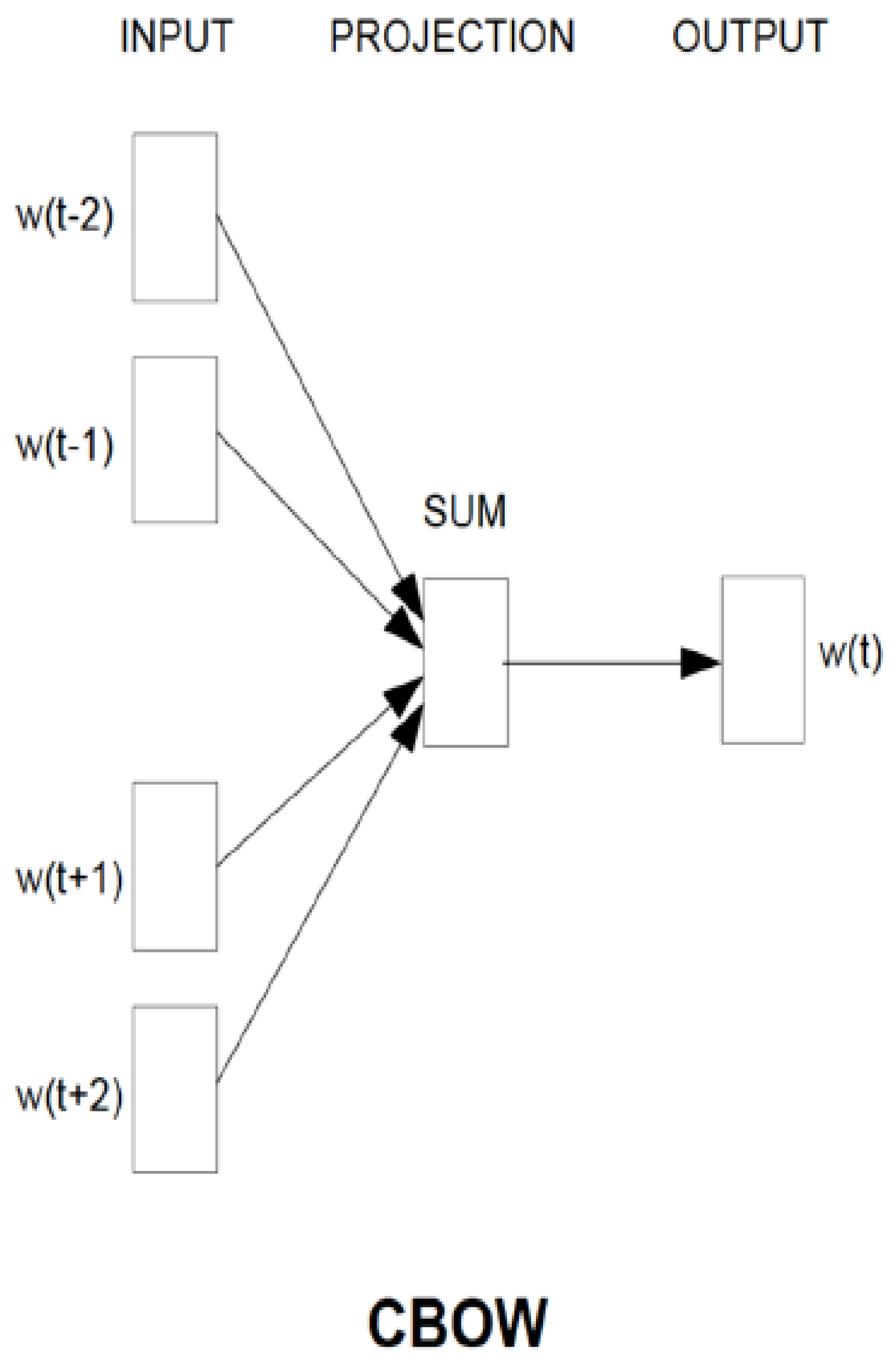


**Skip-gram**

## CBOW model

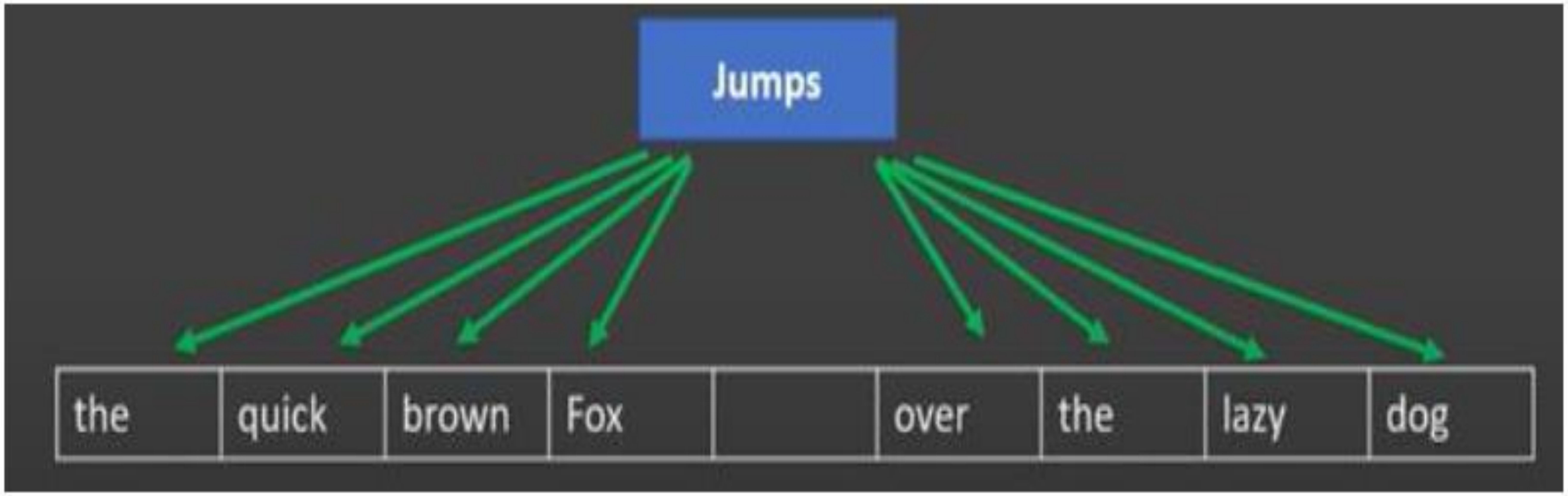
- **Predict the current word** (the center word) based on the surrounding words (context words).

- Mikolov et. al. 2013. Efficient Estimation of Word Representations in Vector Space.



## Skip-gram

- It **predicts context words** from the target word.

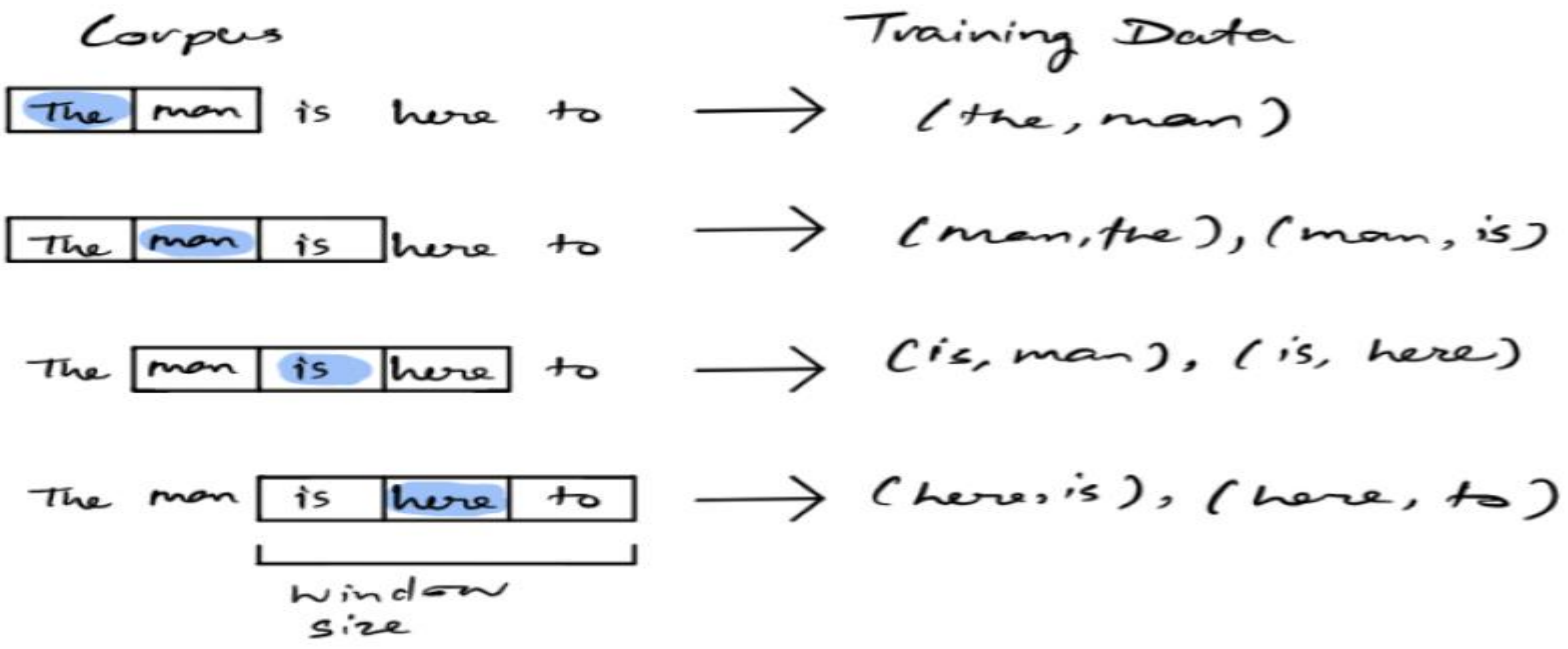


# Skip-gram

## The model (1)

- Given a sliding window of a fixed size moving along a sentence:
  - the word in the middle is the "target";
  - those on its left and right within the sliding window are the context words.

## The model (2)



# Skip-gram

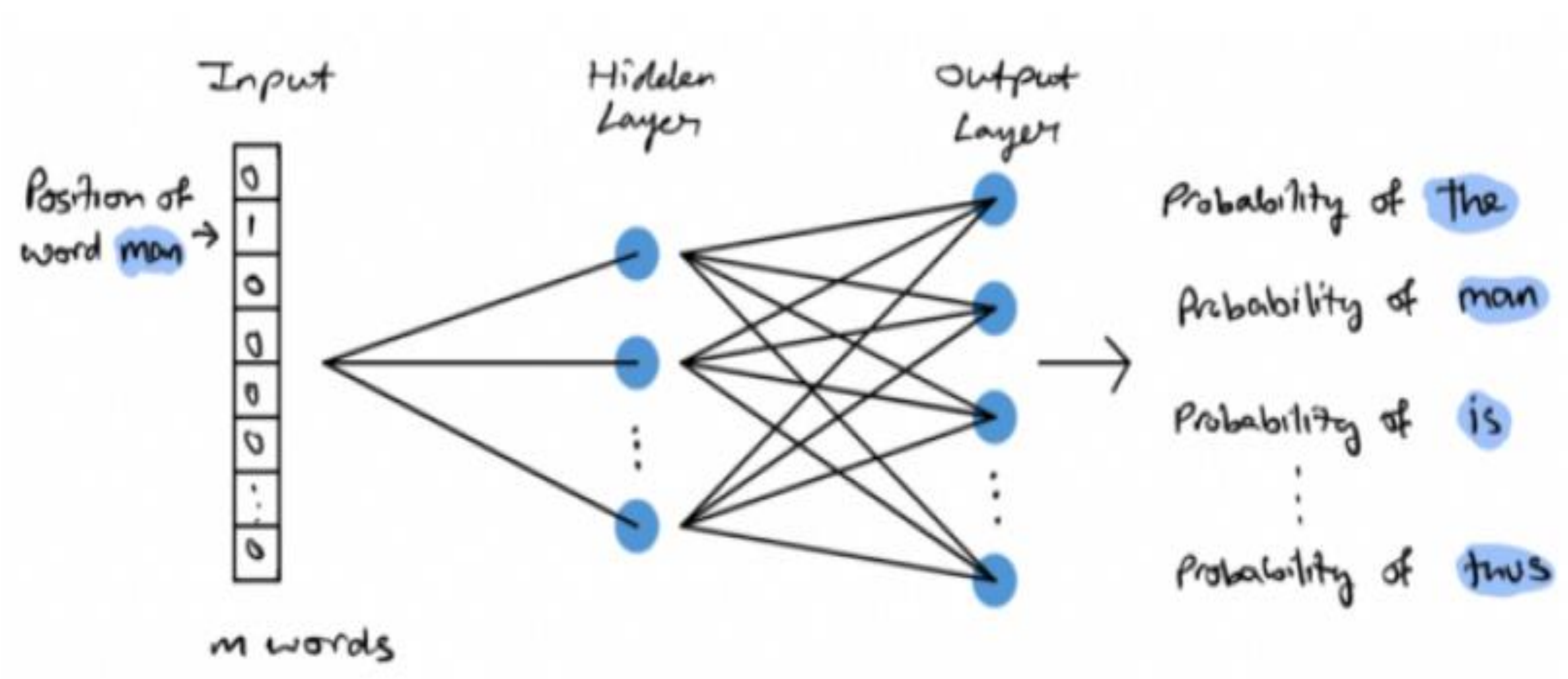
## The model (3)

- Given a sliding window of a fixed size moving along a sentence:
  - the word in the middle is the “target”;
  - those on its left and right within the sliding window are the context words.

- The skip-gram model **is trained to predict** the **probabilities** of a word being a context word for the given target.

# Skip-gram

## The model

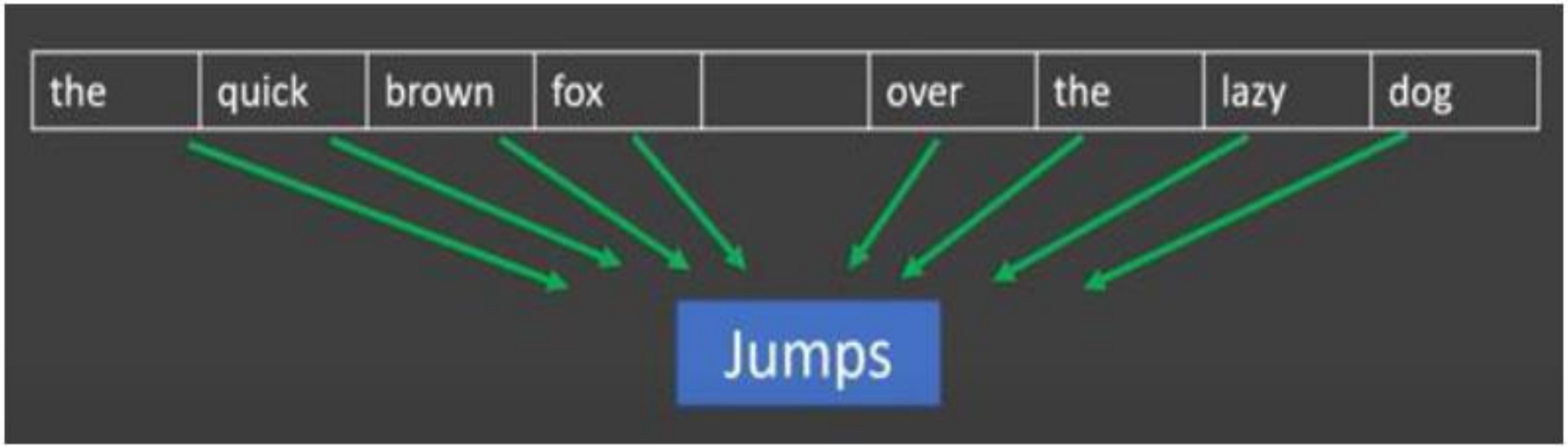


- The hidden layer **is the word embedding** of size $N$.

## CBOW

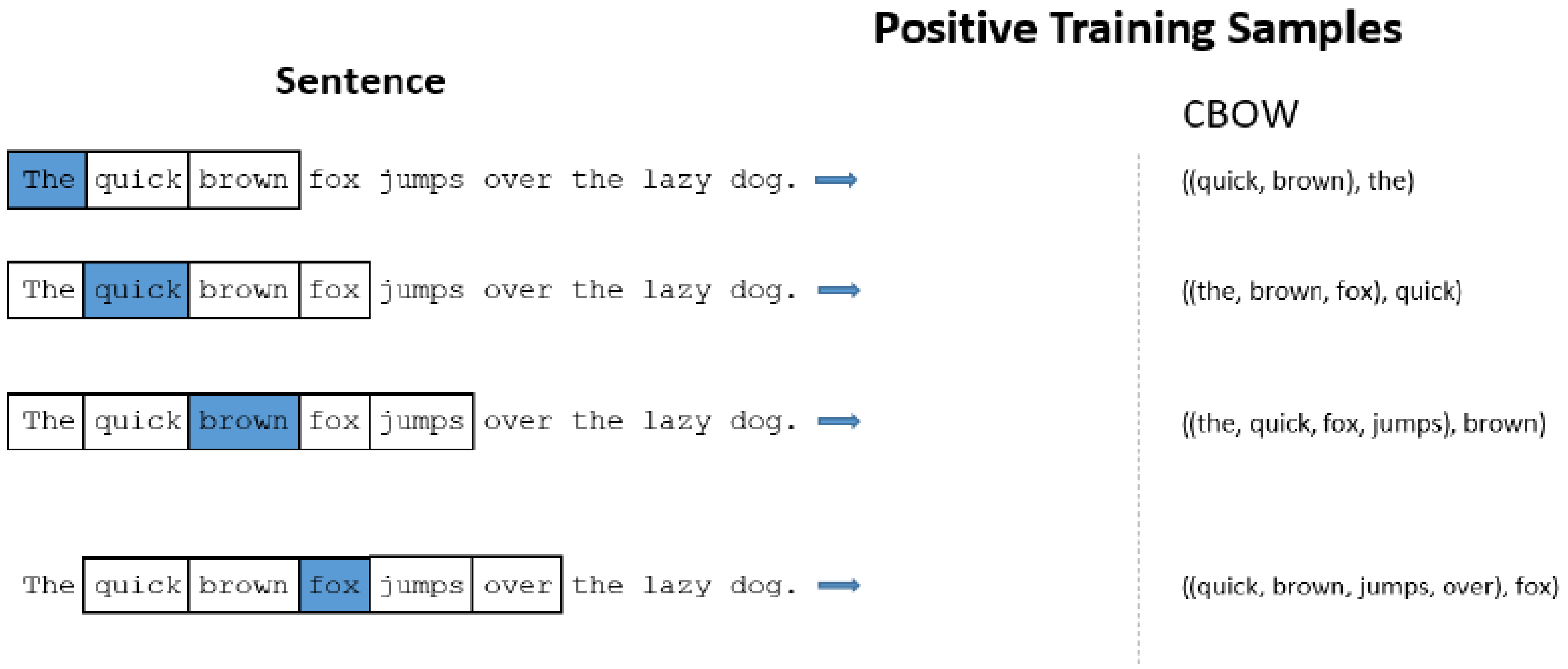
- The **Continuous Bag-of-Words (CBOW)** is another similar model for learning word vectors.

- It **predicts the target word** from source context words.



# Continuous Bag-of-Words (CBOW)

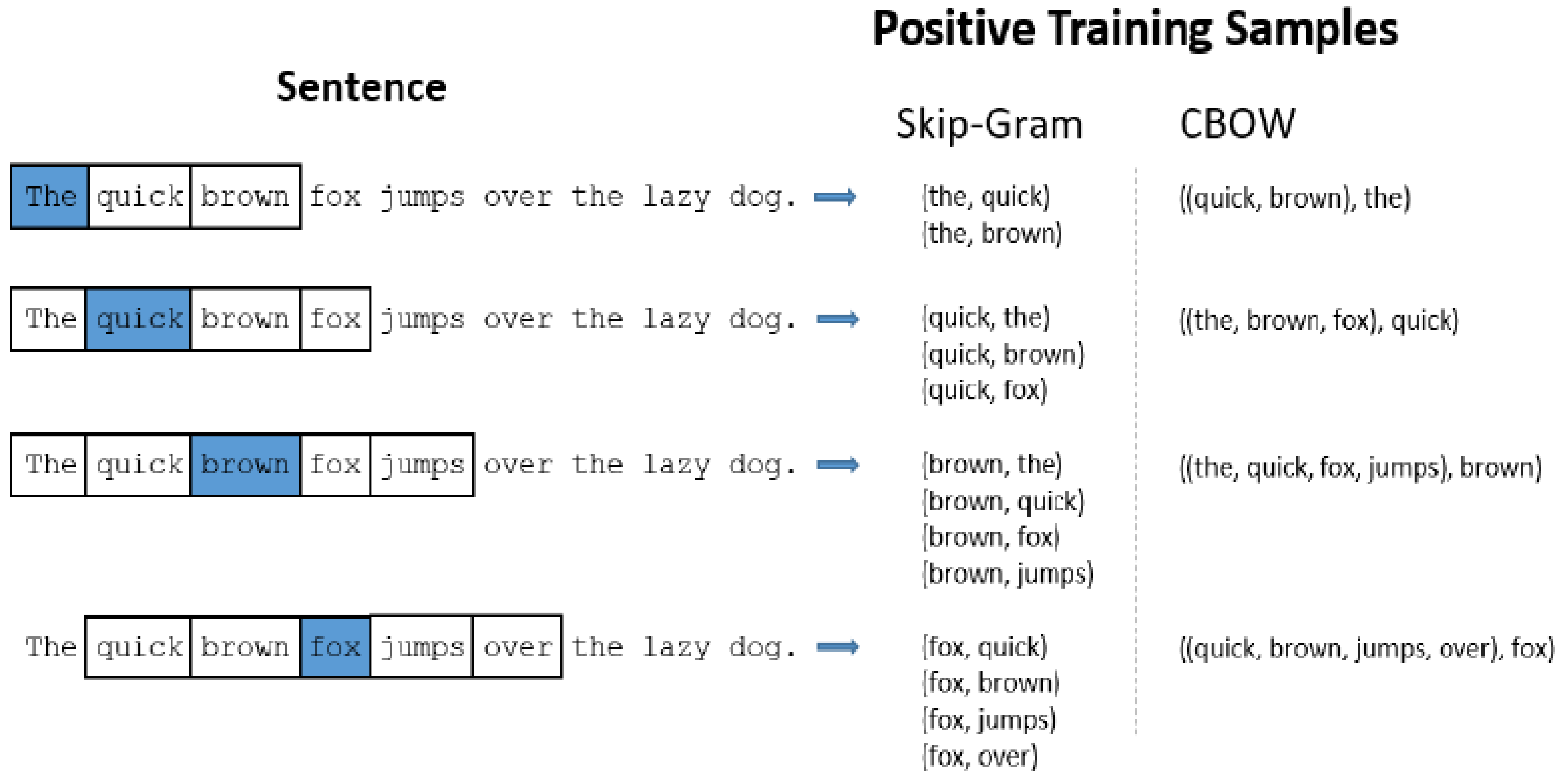
## The model (1)



Positive Training Samples
Sentence
CBOW
The quick brown fox jumps over the lazy dog.
((quick, brown), the)
The quick brown fox jumps over the lazy dog.
((the, brown, fox), quick)
The quick brown fox jumps over the lazy dog.
((the, quick, fox, jumps), brown)
The quick brown fox jumps over the lazy dog.
((quick, brown, jumps, over), fox)

# Continuous Bag-of-Words (CBOW)

## The model (1)

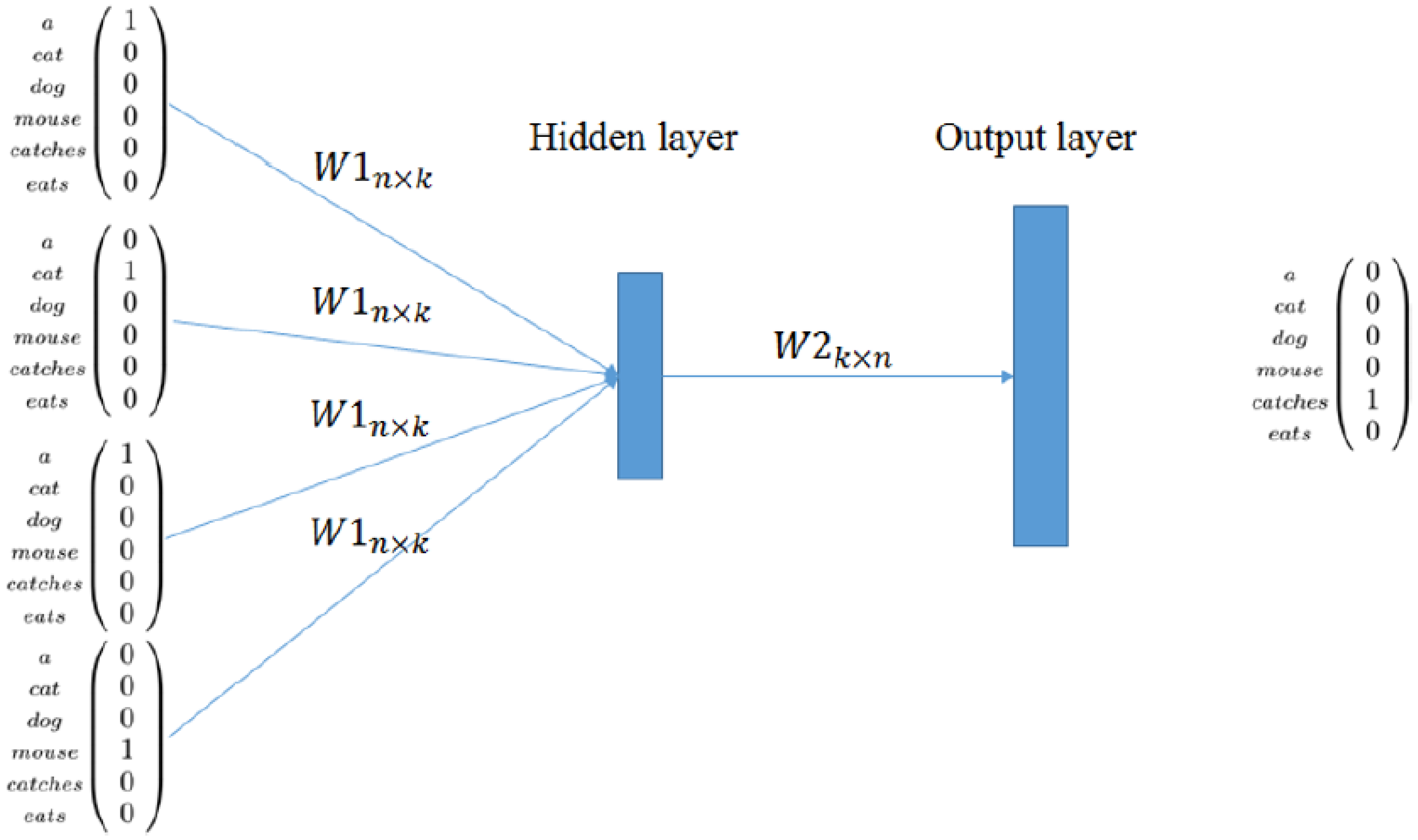


**Positive Training Samples**

| Sentence | Skip-Gram | CBOW |
|---|---|---|
| The quick brown fox jumps over the lazy dog. | (the, quick)<br>(the, brown) | ((quick, brown), the) |
| The quick brown fox jumps over the lazy dog. | (quick, the)<br>(quick, brown)<br>(quick, fox) | ((the, brown, fox), quick) |
| The quick brown fox jumps over the lazy dog. | (brown, the)<br>(brown, quick)<br>(brown, fox)<br>(brown, jumps) | ((the, quick, fox, jumps), brown) |
| The quick brown fox jumps over the lazy dog. | (fox, quick)<br>(fox, brown)<br>(fox, jumps)<br>(fox, over) | ((quick, brown, jumps, over), fox) |

# Continuous Bag-of-Words (CBOW)

## The model (2)



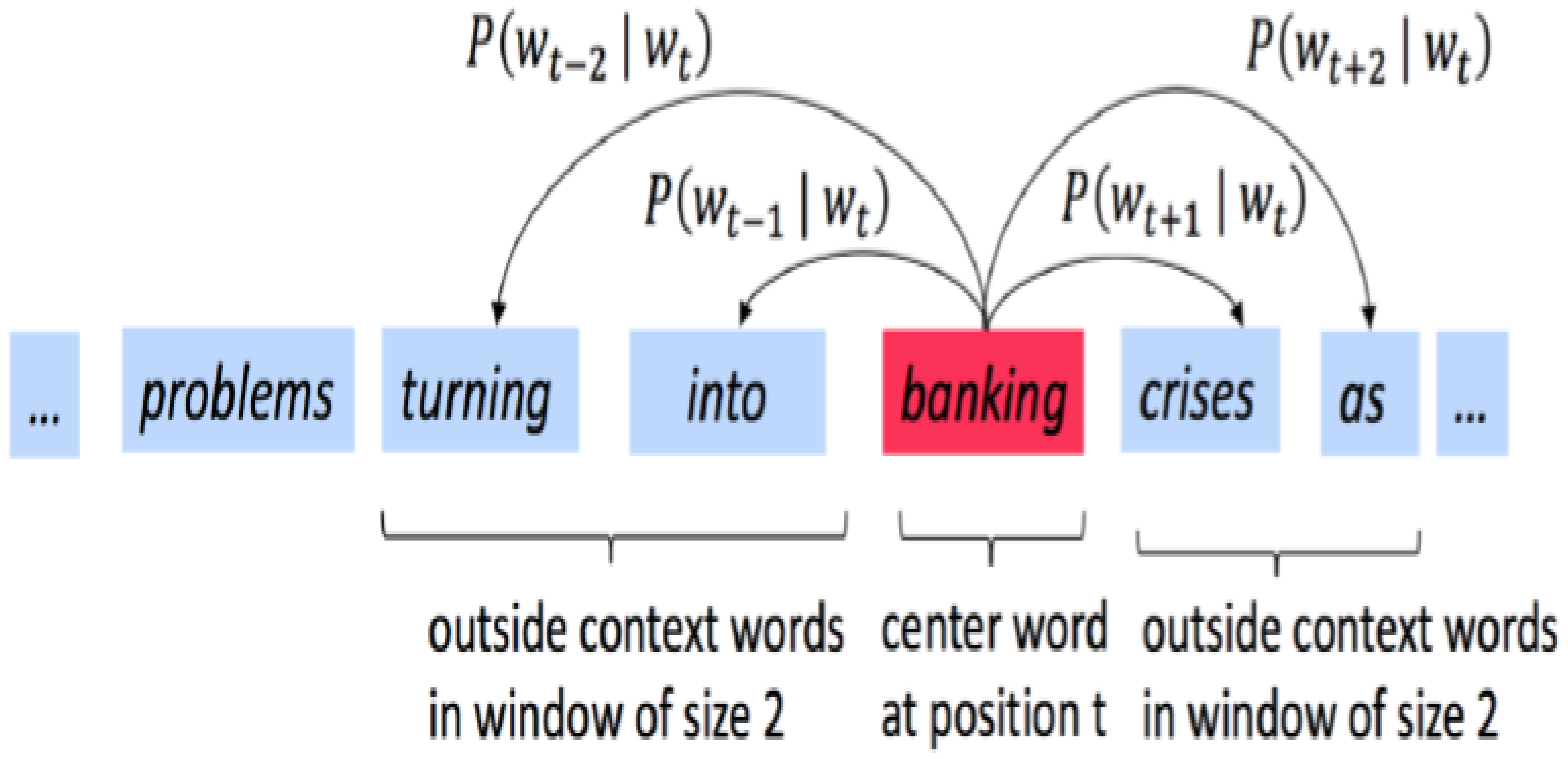
## Skip-gram VS CBOW

- **Skip-gram**: works well also with a smaller amount of the training data, represents well even rare words or phrases.

- **CBOW**: several times faster to train than the skip-gram, slightly better accuracy for the frequent words.

## Word2Vec

- Examples windows and and process for computing $P(w_{t+j}|w_j)$



http://web.stanford.edu/class/cs224n/syllabus.html

## Word2Vec: objective function

For each position $t = 1, \ldots, T$, predict context words within a window of fixed size $m$, given center word $w_t$.

Likelihood = $$L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \ne 0}} P(w_{t+j} \mid w_t; \theta)$$

*$\theta$ is all variables to be optimized*

**m= -2, -1,1,2**

## Word2Vec: objective function

The **objective function (or loss, or cost function)** $J(\theta)$ is the (average) negative log likelihood

$$J(\theta) = -\frac{1}{T}\log L(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leq j\leq m\\ j\neq 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

Minimizing objective function $\Longleftrightarrow$ Maximizing predictive accuracy

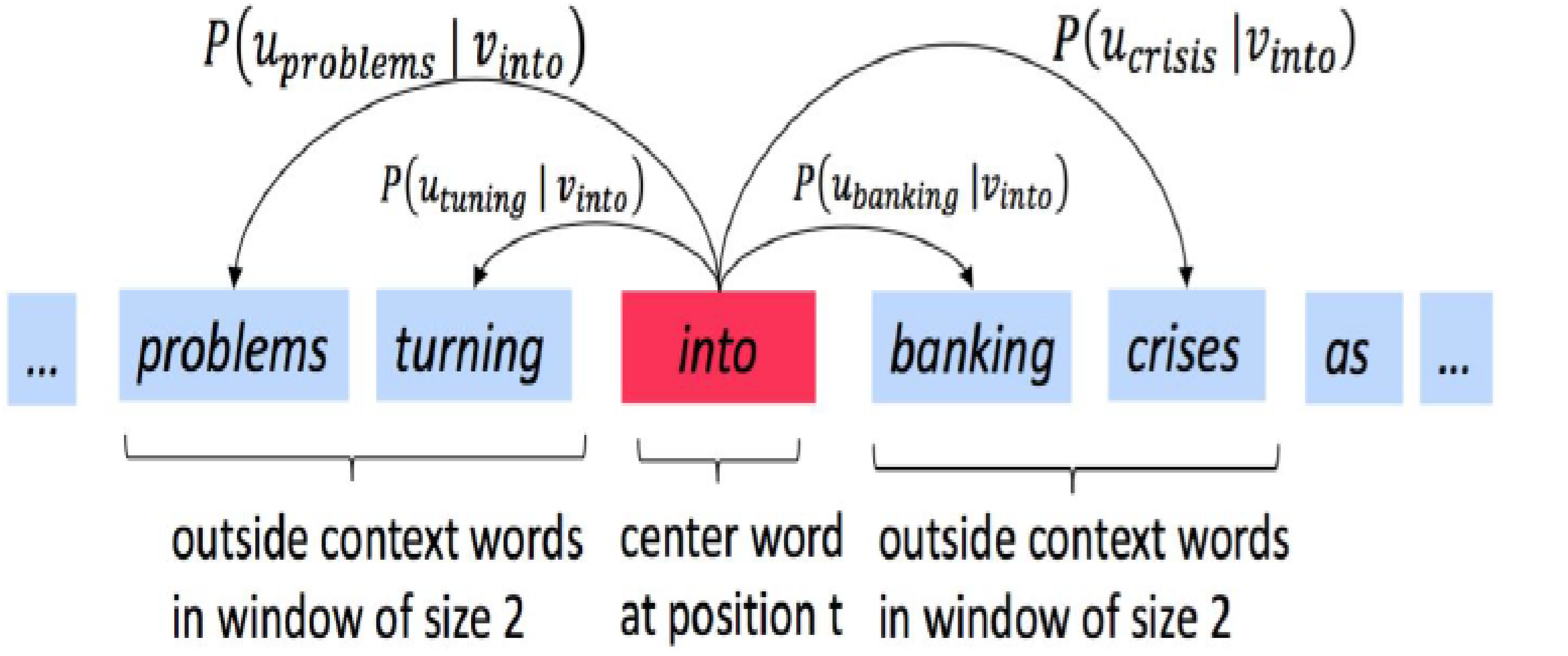
## Word2Vec: objective function

- We want to minimize objective function $J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leq j\leq m \\ j\neq 0}} \log P(w_{t+j} \mid w_t;\theta)$
- <u>Question</u>: **How to calculate $P(w_{t+j}|w_j,\theta)$?**
- <u>Answer</u>: We will use two vectors per word *w*
  - $v_w$ is a center word
  - $u_w$ is a context word
- Then for a center word **c** and a context word **o**:

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w\in V}\exp(u_w^T v_c)}$$

## Word2Vec

- Examples windows and and process for computing $P(w_{t+j}|w_j)$
- $P(u_{problems}|v_{into})$ is short for $P(problems|into;\ u_{problems}, v_{into}, \theta)$

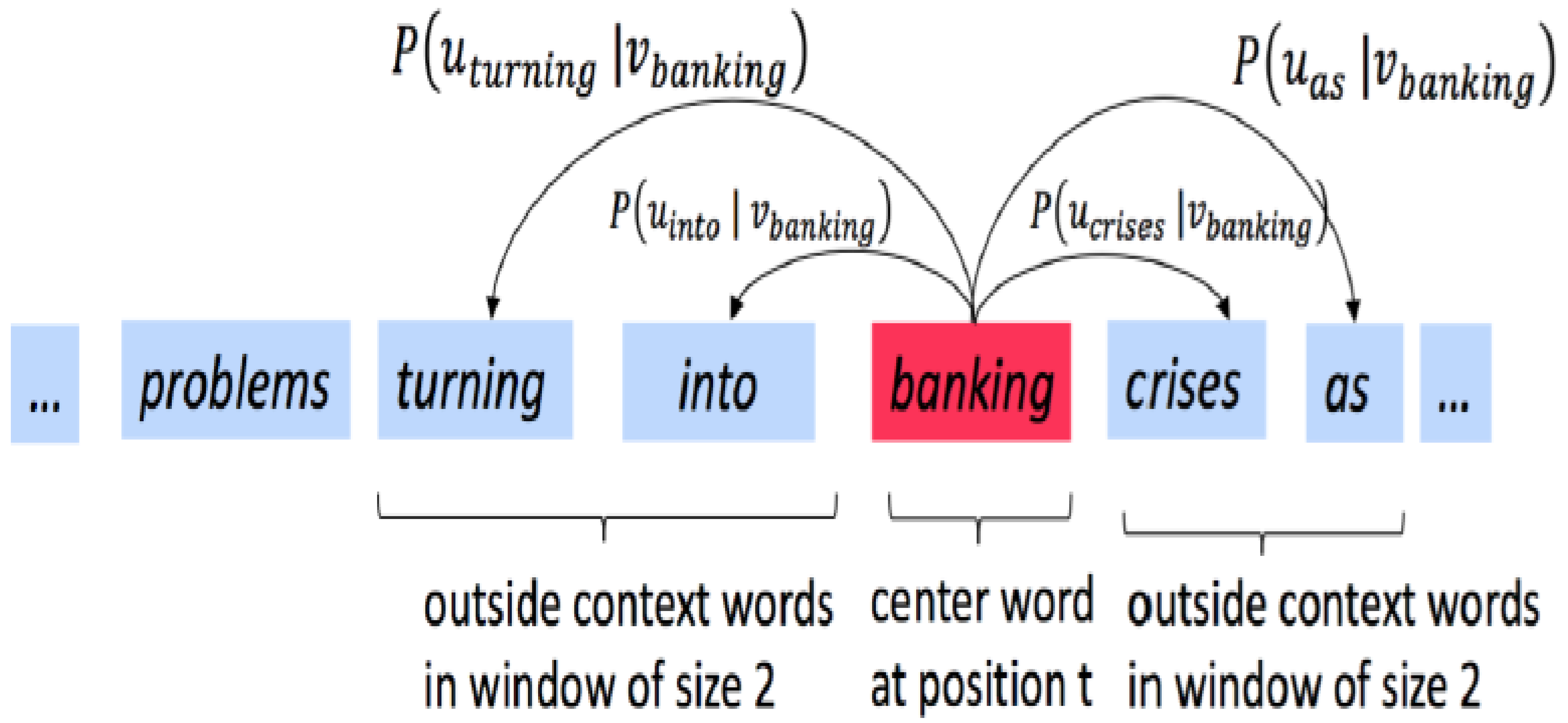


$P(u_{problems} \mid v_{into})$
$P(u_{crisis} \mid v_{into})$
$P(u_{tuning} \mid v_{into})$
$P(u_{banking} \mid v_{into})$
...
problems
turning
into
banking
crises
as
...
outside context words in window of size 2
center word at position t
outside context words in window of size 2

http://web.stanford.edu/class/cs224n/syllabus.html

## Word2Vec

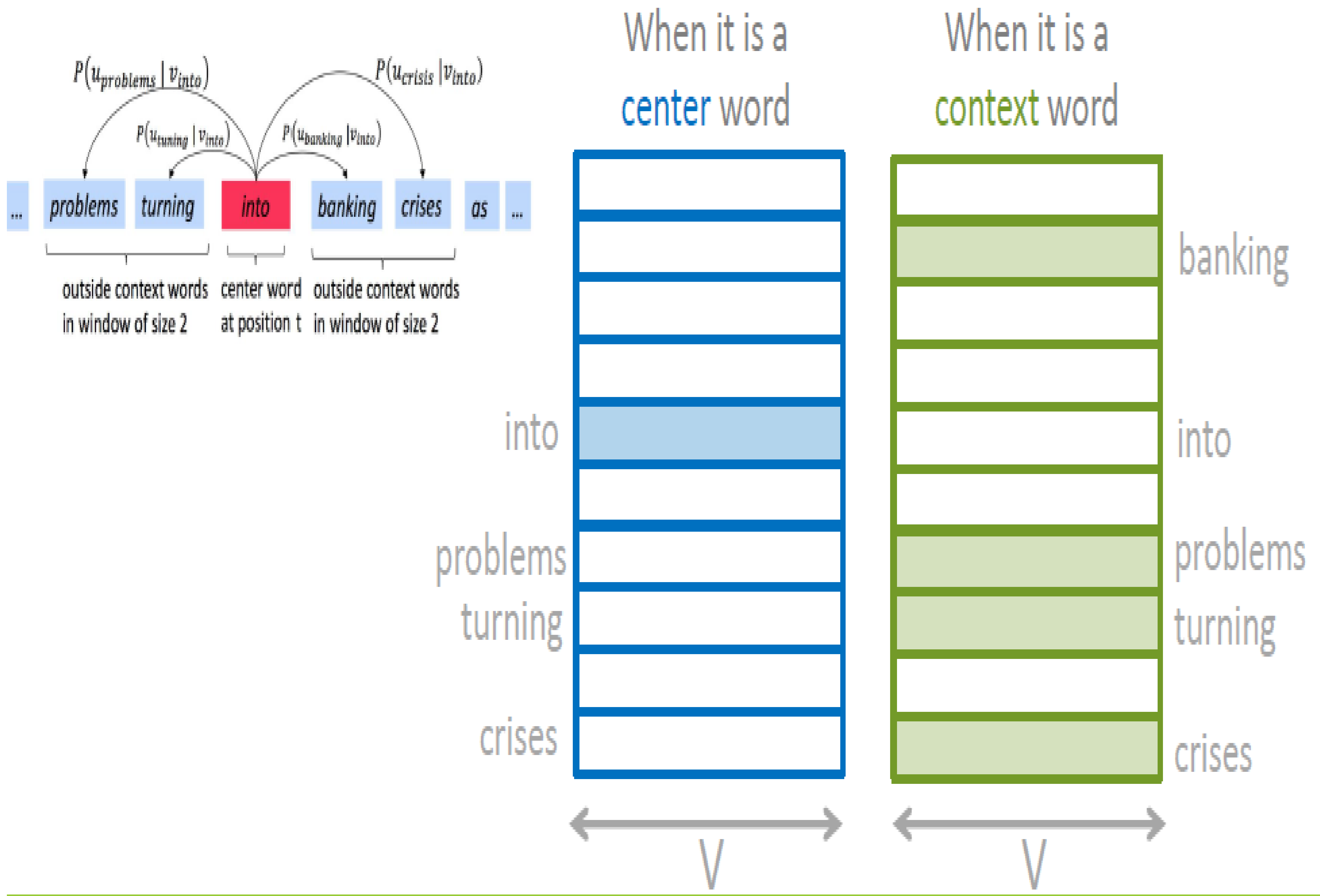
- Examples windows and and process for computing $P(w_{t+j}|w_j)$
- $P(u_{problems}|v_{into})$ is short for $P(problems|into;\ u_{problems}, v_{into}, \theta)$



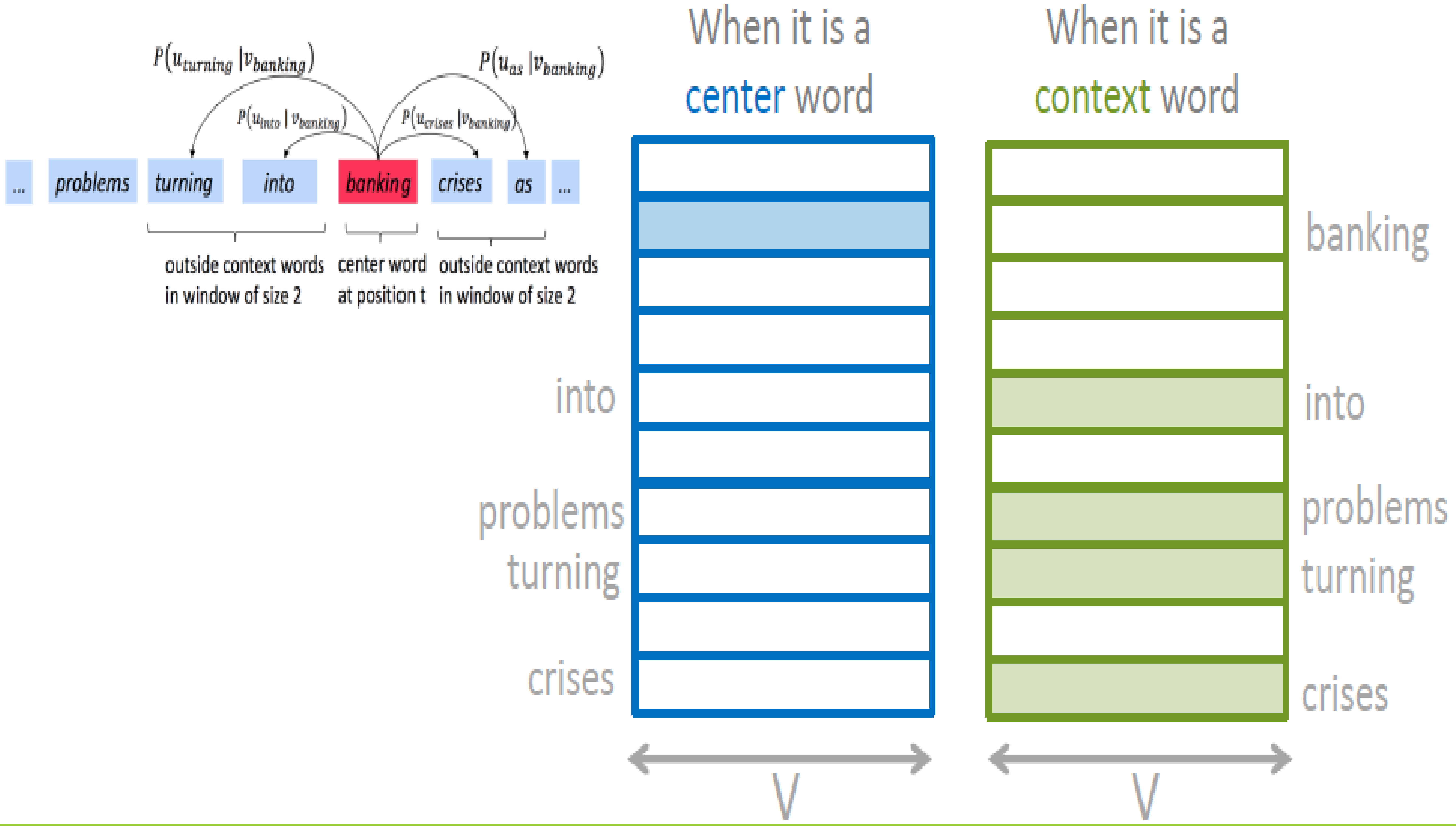
$P(u_{turning}|v_{banking})$
$P(u_{as}|v_{banking})$
$P(u_{into}|v_{banking})$
$P(u_{crises}|v_{banking})$
...
problems
turning
into
banking
crises
as
...
outside context words in window of size 2
center word at position t
outside context words in window of size 2

http://web.stanford.edu/class/cs224n/syllabus.html

# Two vectors for each word


$P(u_{problems} \mid v_{into})$
$P(u_{crisis} \mid v_{into})$
$P(u_{tuning} \mid v_{into})$
$P(u_{banking} \mid v_{into})$
...
problems
turning
into
banking
crises
as
...
outside context words in window of size 2
center word at position t
outside context words in window of size 2
When it is a center word
When it is a context word
into
problems
turning
crises
banking
into
problems
turning
crises
V
V

# Two vectors for each word



$P(u_{turning} \mid v_{banking})$
$P(u_{into} \mid v_{banking})$
$P(u_{crises} \mid v_{banking})$
$P(u_{as} \mid v_{banking})$
...
problems
turning
into
banking
crises
as
...
outside context words in window of size 2
center word at position t
outside context words in window of size 2
When it is a center word
When it is a context word
into
problems
turning
crises
banking
into
problems
turning
crises
V
V

## Word2Vec: prediction function

$$P(o|c) = \frac{\exp(\boxed{u_o^T\, v_c})}{\boxed{\sum_{w\in V} \exp(u_w^T\, v_c)}}$$

Dot product measures similarity of ***o*** and ***c***
Larger dot product = larger probability

After taking exponent, normalize
over entire vocabulary

Mikolov et al, 2013, https://arxiv.org/pdf/1310.4546.pdf

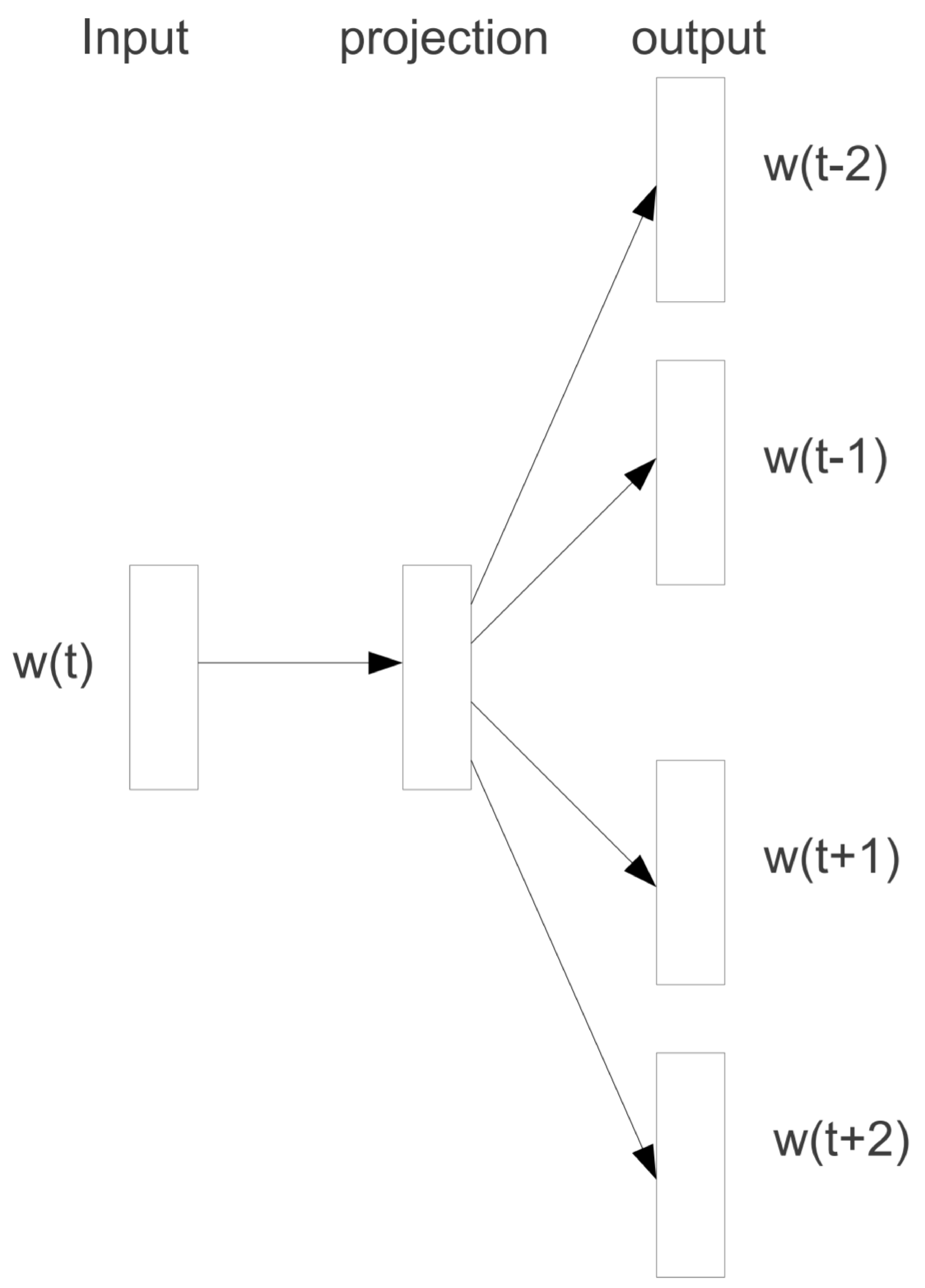
## This is softmax!

**Softmax function $\mathbb{R}^n \rightarrow \mathbb{R}^n$:**

$$softmax(\boldsymbol{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$$

- maps arbitrary values $x_i$ to a probability distribution $p_i$
- "max" because amplifies probability of largest $x_i$
- "soft" because still assigns some probability to smaller $x_i$
- **often used in Deep Learning!**

## Prediction-based word-embedding: Word2Vec Skip-Gram



*Original Skip-gram model architecture*

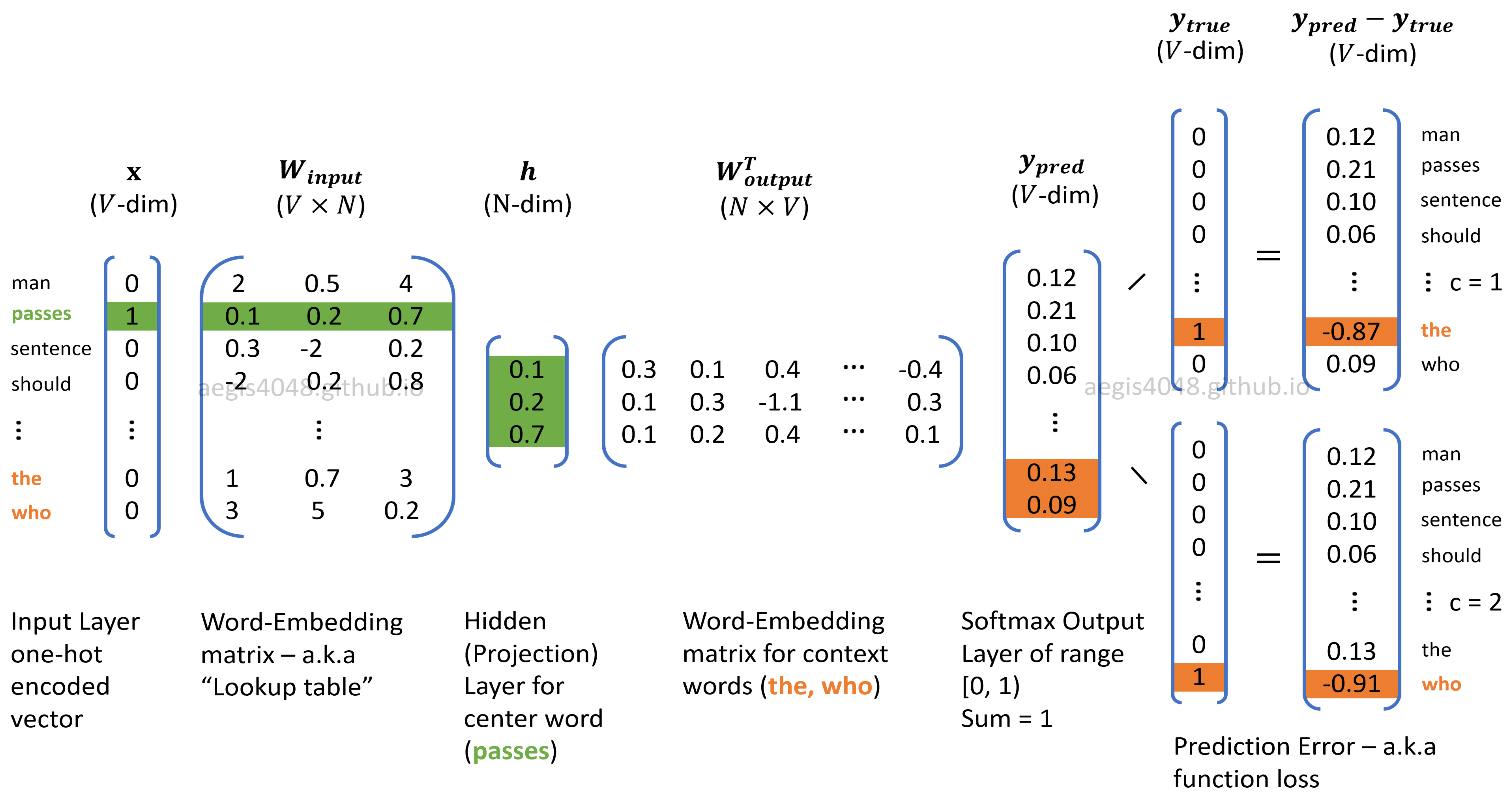
https://aegis4048.github.io/demystifying_neural_network_in_skip_gram_language_m odeling

## . Derivation of cost function. Derivation of cost function

- **Skip-Gram model seeks to optimize the word weight (embedding) matrix by correctly predicting context words, given a center word.**

This means that the input weight matrix ($W_{input}$) will have a size of $8 \times 3$, and output weight matrix ($W^T_{output}$) will have a size of $3 \times 8$.

Recall that the corpus, *"The man who passes the sentence should swing the sword"*, has 8 unique vocabularies ($V = 8$).



$\mathbf{x}$ ($V$-dim)
$W_{input}$ ($V \times N$)
$h$ (N-dim)
$W^T_{output}$ ($N \times V$)
$y_{pred}$ ($V$-dim)
$y_{true}$ ($V$-dim)
$y_{pred} - y_{true}$ ($V$-dim)
man
passes
sentence
should
⋮
the
who
c = 1
c = 2
aegis4048.github.io
Input Layer one-hot encoded vector
Word-Embedding matrix – a.k.a "Lookup table"
Hidden (Projection) Layer for center word (passes)
Word-Embedding matrix for context words (the, who)
Softmax Output Layer of range [0, 1) Sum = 1
Prediction Error – a.k.a function loss

$$\mathbf{x} = \begin{pmatrix}0\\1\\0\\0\\\vdots\\0\\0\end{pmatrix} \quad W_{input} = \begin{pmatrix}2 & 0.5 & 4\\0.1 & 0.2 & 0.7\\0.3 & -2 & 0.2\\-2 & 0.2 & 0.8\\ & \vdots & \\1 & 0.7 & 3\\3 & 5 & 0.2\end{pmatrix} \quad h = \begin{pmatrix}0.1\\0.2\\0.7\end{pmatrix} \quad W^T_{output} = \begin{pmatrix}0.3 & 0.1 & 0.4 & \cdots & -0.4\\0.1 & 0.3 & -1.1 & \cdots & 0.3\\0.1 & 0.2 & 0.4 & \cdots & 0.1\end{pmatrix}$$

$$y_{pred} = \begin{pmatrix}0.12\\0.21\\0.10\\0.06\\\vdots\\0.13\\0.09\end{pmatrix}$$

$$\begin{pmatrix}0.12\\0.21\\0.10\\0.06\\\vdots\\0.13\\0.09\end{pmatrix} - \begin{pmatrix}0\\0\\0\\0\\\vdots\\1\\0\end{pmatrix} = \begin{pmatrix}0.12\\0.21\\0.10\\0.06\\\vdots\\-0.87\\0.09\end{pmatrix} \quad c = 1$$

$$\begin{pmatrix}0.12\\0.21\\0.10\\0.06\\\vdots\\0.13\\0.09\end{pmatrix} - \begin{pmatrix}0\\0\\0\\0\\\vdots\\0\\1\end{pmatrix} = \begin{pmatrix}0.12\\0.21\\0.10\\0.06\\\vdots\\0.13\\-0.91\end{pmatrix} \quad c = 2$$

# Word2Vec: objective function

- We want to minimize objective function $$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leq j\leq m \\ j\neq 0}} \log P(w_{t+j} \mid w_t;\theta)$$
- Question: **How to calculate** $P(w_{t+j}|w_j, \theta)$?
- Answer: We will use two vectors per word *w*
  - $v_w$ is a center word
  - $u_w$ is a context word
- Then for a center word **c** and a context word **o**:

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w\in V}\exp(u_w^T v_c)}$$

for a center word **c** and a context word **o**:

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

$$p(w_{context}|w_{center};\theta) = \frac{exp(W_{output_{(context)}} \cdot h)}{\sum_{i=1}^{V} exp(W_{output_{(i)}} \cdot h)}$$

$W_{output_{(context)}}$ **is a row vector for a context word from the output embedding matrix**

**h is the hidden (projection) layer word vector for a center word**

**SoftMax function is used to yield a new objective function that maximizes the probability of observing all C context words, given a center word:**

$$\underset{\theta}{\operatorname{argmax}}\ log \prod_{c=1}^{C} \frac{exp(W_{output_{(c)}} \cdot h)}{\sum_{i=1}^{V} exp(W_{output_{(i)}} \cdot h)}$$

**Minimizing a negative log-likelihood is equivalent to maximizing a positive log-likelihood. Therefore, the cost function we want to minimize becomes:**

$$J(\theta; w^{(t)}) = -\sum_{c=1}^{C} log \frac{exp(W_{output_{(c)}} \cdot h)}{\sum_{i=1}^{V} exp(W_{output_{(i)}} \cdot h)}$$

Taking a log to the softmax function allows us to simplify the expression into simpler forms :because we can split the fraction into addtion of the numerator and the denominator

## Window size of Skip-Gram

Window size m is a hyper parameter of the model with a typical range of [1,10

- We want to minimize objective function $J(\theta) = -\frac{1}{T}\sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \ne 0}} \log P(w_{t+j} \mid w_t; \theta)$

$$\theta = [W_{input} \quad W_{output}] = \begin{bmatrix} u_{the} \\ u_{passes} \\ \vdots \\ u_{who} \\ v_{the} \\ v_{passes} \\ \vdots \\ v_{who} \end{bmatrix} \in \mathbb{R}^{2NV}$$

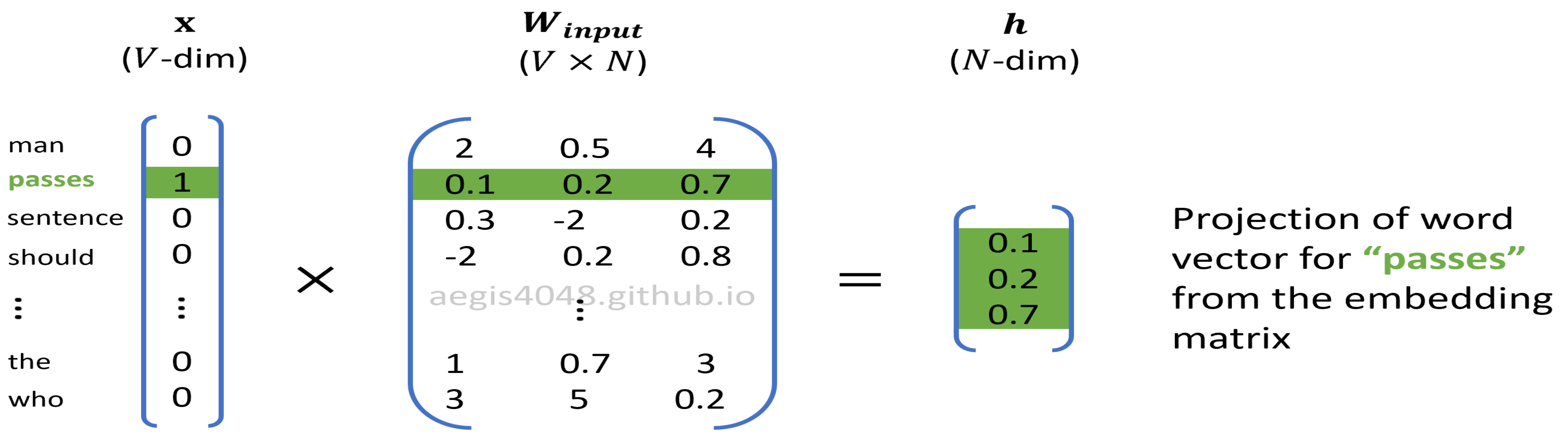
$\theta$ has a size of $2V \times N$, where $V$ is the number of unique vocab in a corpus, and $N$ is the dimension of word vectors in the embedding matrices. $2$ is multipled to $V$ because there are two weight matrices, $W_{input}$ and $W_{output}$. $u$ is a word vector from $W_{input}$ and $v$ is a word vector from $W_{output}$. Each word vectors are $N$-dim row vectors from input and output embedding matrices.

$$\theta = [W_{input} \quad W_{output}] = \begin{bmatrix} u_{the} \\ u_{passes} \\ \vdots \\ u_{who} \\ v_{the} \\ v_{passes} \\ \vdots \\ v_{who} \end{bmatrix} \in \mathbb{R}^{2NV}$$

$\theta$ has a size of $2V \times N$, where $V$ is the number of unique vocab in a corpus, and $N$ is the dimension of word vectors in the embedding matrices. $2$ is multipled to $V$ because there are two weight matrices, $W_{input}$ and $W_{output}$. $u$ is a word vector from $W_{input}$ and $v$ is a word vector from $W_{output}$. Each word vectors are $N$-dim row vectors from input and output embedding matrices.

## Hidden (projection) layer (h)

Skip-Gram uses a neural net with one hidden layer. In the context of natural language processing, hidden layer is often referred to as a *projection* layer, because h is essentially an N-dim vector projected by the one-hot encoded input vector .



## 4. Softmax output layer ($y_{pred}$)

The output layer is a $V$-dim probability distribution of all unique words in the corpus, given a center word. In statistics, the conditional probability of $A$ given $B$ is denoted as $p(A|B)$. In Skip-Gram, we use the notation, $p(w_{context}|w_{center})$, to denote the conditional probability of observing a context word given a center word. It is obtained by using the softmax function,

$$p(w_{context}|w_{center}) = \frac{exp(W_{output_{(context)}} \cdot h)}{\sum_{i=1}^{V} exp(W_{output_{(i)}} \cdot h)} \in \mathbb{R}^1 \tag{13}$$

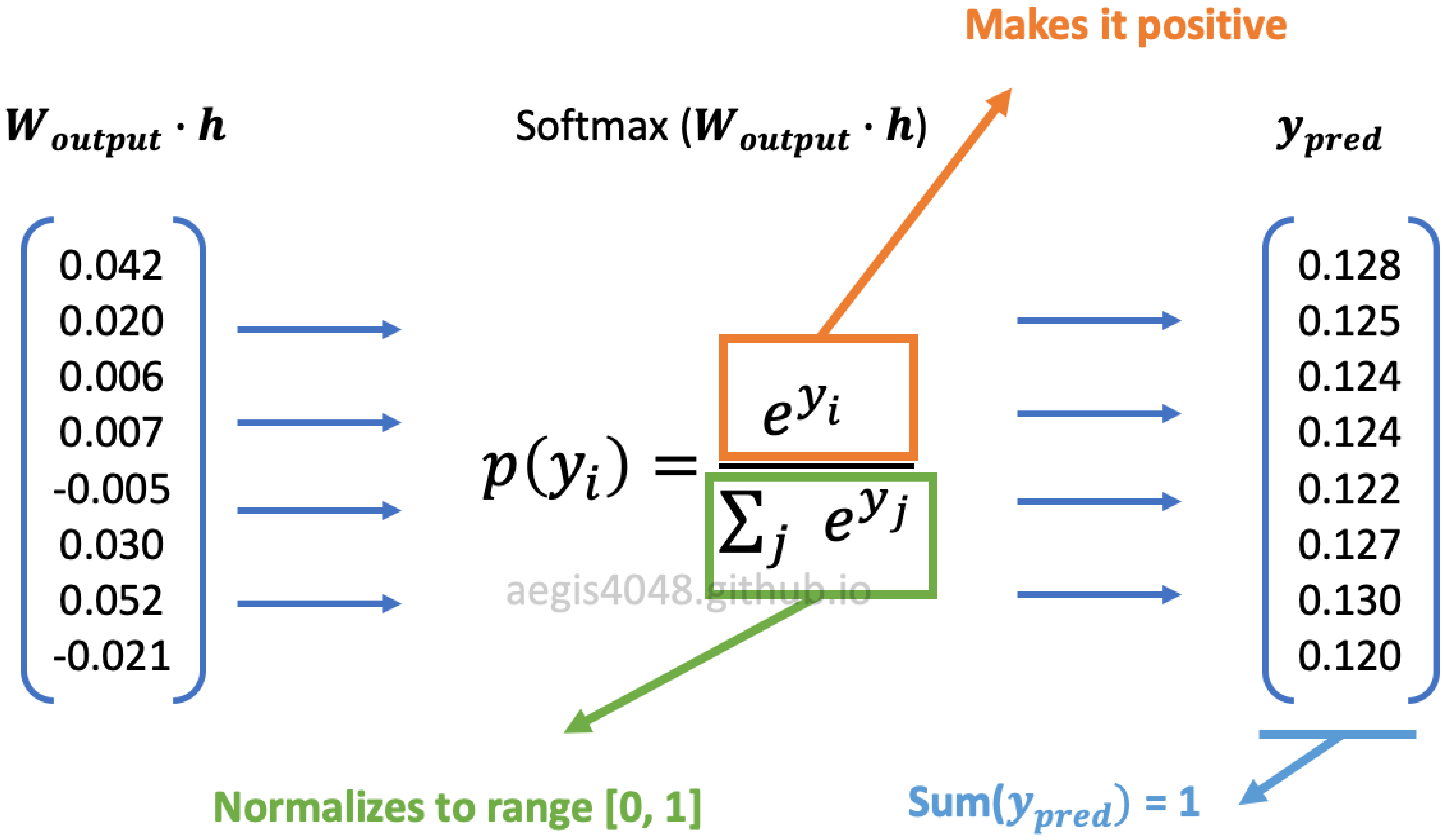
where $W_{output_{(i)}}$ is the $i$-th row vector of size $1 \times N$ from the output embedding matrix, $W_{output_{context}}$ is also a row vector of size $1 \times N$ from the output embedding matrix corresponding to the context word. $V$ is the size of unique vocab in the corpus, and $h$ is the hidden (projection) layer of size ($N \times 1$). The output is an $1 \times 1$ scalar value of probability of range $[0, 1)$.

This probability is computed $V$ times to obtain a conditional probability distribution of observing each unique vocabs in the corpus, given a center word.

$$\begin{bmatrix} p(w_1|w_{center}) \\ p(w_2|w_{center}) \\ p(w_3|w_{center}) \\ \vdots \\ p(w_V|w_{center}) \end{bmatrix} = \frac{exp(W_{output} \cdot h)}{\sum_{i=1}^{V} exp(W_{output_{(i)}} \cdot h)} \in \mathbb{R}^V \tag{14}$$

$W_{output}$ in the denominator of eq 13 has size $V \times N$. Multiplying $W_{output}$ with $h$ of size $N \times 1$ will yield a dot product vector of size $V \times 1$.

# SoftMax function transformation



Makes it positive
$\boldsymbol{W}_{output} \cdot \boldsymbol{h}$
Softmax ($\boldsymbol{W}_{output} \cdot \boldsymbol{h}$)
$y_{pred}$
0.042
0.020
0.006
0.007
-0.005
0.030
0.052
-0.021
$p(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$
aegis4048.github.io
0.128
0.125
0.124
0.124
0.122
0.127
0.130
0.120
Normalizes to range [0, 1]
Sum($y_{pred}$) = 1

**The exponentiation ensures that the transformed values are positive, and the normalization factor in the denominator ensures that the values have a range of [0,1]. The result is a conditional probability distribution of observing each unique vocabs in the corpus, given a center word .**

## Training: backward propagation

Backward propagation involves computing prediction errors, and updating the weight matrix ($\theta$) to optimize vector representation of words. Assuming stochastic gradient descent, we have the following general update equations for the weight matrix ($\theta$):

$$\theta_{new} = \theta_{old} - \eta \cdot \nabla_{J(\theta;w^{(t)})} \tag{15}$$

$\eta$ is learning rate, $\nabla_{J(\theta;w^{(t)})}$ is gradient for the weight matrix, and $J(\theta; w^{(t)})$ is the cost function defined in eq-6. Since the $\theta$ is a concatenation of input and output weight matrices ($[W_{input} \quad W_{output}]$) as described above, there are two update equations for each embedding matrix:

$$W_{input}^{(new)} = W_{input}^{(old)} - \eta \cdot \frac{\partial J}{\partial W_{input}} \tag{16}$$

$$W_{output}^{(new)} = W_{output}^{(old)} - \eta \cdot \frac{\partial J}{\partial W_{output}} \tag{17}$$

Mathematically, it can be shown that the gradients of $W_{input}$ $W_{output}$ have the following forms:

$$\frac{\partial J}{\partial W_{input}} = x \cdot (W_{output}^T \sum_{c=1}^{C} e_c) \tag{18}$$

$$\frac{\partial J}{\partial W_{output}} = h \cdot \sum_{c=1}^{C} e_c \tag{19}$$

The gradients can be substitued into eq-16 and eq-17:

$$W_{input}^{(new)} = W_{input}^{(old)} - \eta \cdot x \cdot (W_{output}^T \sum_{c=1}^{C} e_c) \tag{20}$$

$$W_{output}^{(new)} = W_{output}^{(old)} - \eta \cdot h \cdot \sum_{c=1}^{C} e_c \tag{21}$$

$W_{input}$ and $W_{output}$ are the input and output weight matrices, $x$ is one-hot encoded input layer, $C$ is window size, and $e_c$ is prediction error for $c$-th context word in the window. Note that $h$ (hidden layer) is equivalent to $W_{input}^T x$.

### 5.1.1. Prediction error ($y_{pred} - y_{true}$)

Skip-Gram model optimizes the weight matrix ($\theta$) to reduce the prediction error. Prediction error is the difference between the probability distribution of words computed from the softmax output layer ($y_{pred}$) and the true probability distribution ($y_{true}$) of the $c$-th context word. Just like the input layer, $y_{true}$ is one-hot encoded vector, in which only one element in the vector that corresponds to the $c$-th context word is $1$, and the rest is all $0$.

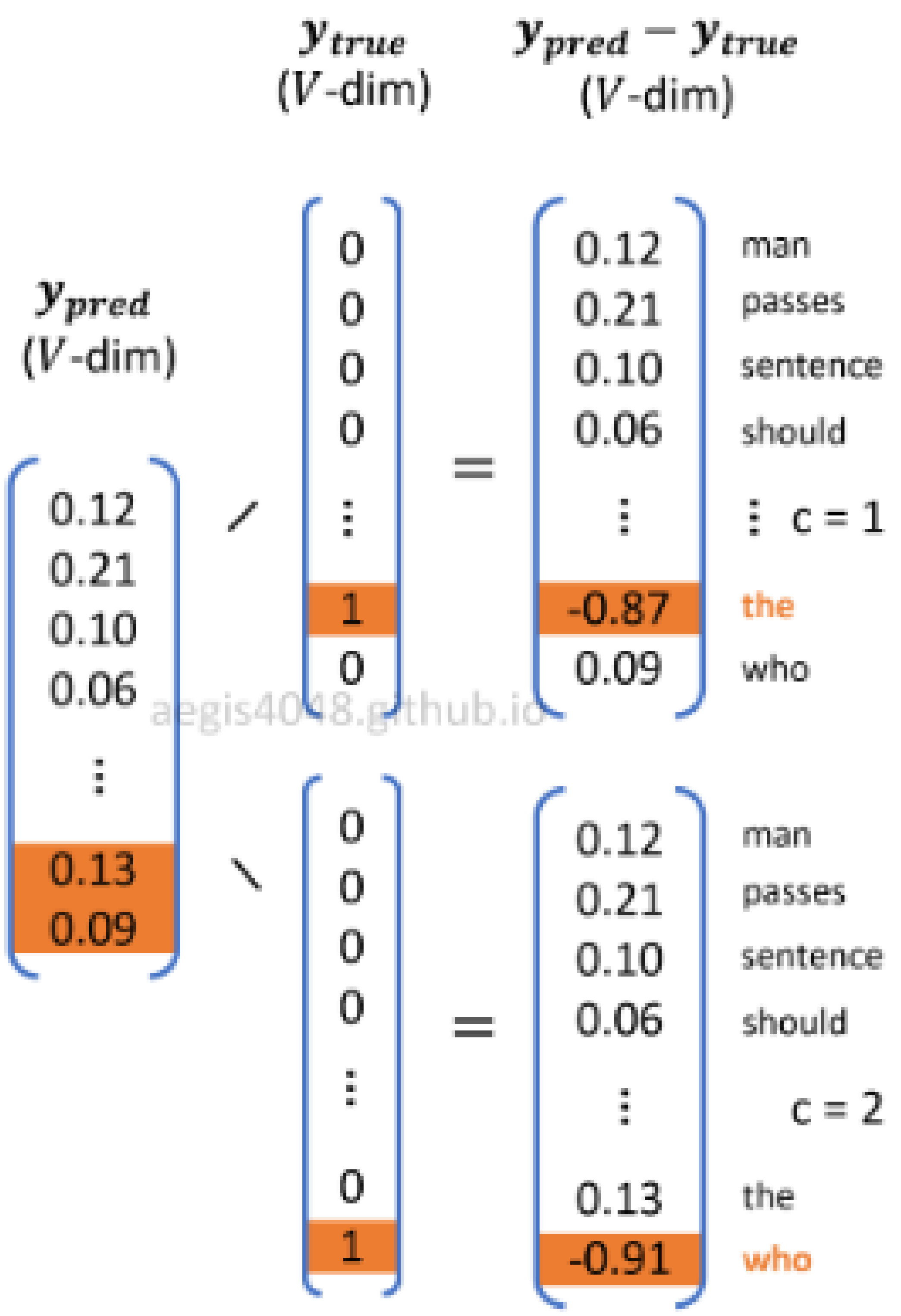


***Figure 11:*** *Prediction error window*

The figure has a window size of $2$, so two prediction errors were computed. Recall from the above notes about the window size that the original softmax regression classifier (eq-10) has $K$ labels to classify, in which $K = V$ in NLP applications because there are $V$ words to classify. Employing window size transforms eq-10 into eq-11 significantly reduces the algorithm complexity because the model only needs to compute prediction errors for 2~10 (this is a hyperparameter $C$) neighboring words, instead of computing all $V$-prediction errors for all vocabs that can be millions or more.

Then, prediction errors for all $C$ context words are summed up to compute weight gradients to the update weight matrices, according to eq-18 and eq-19.

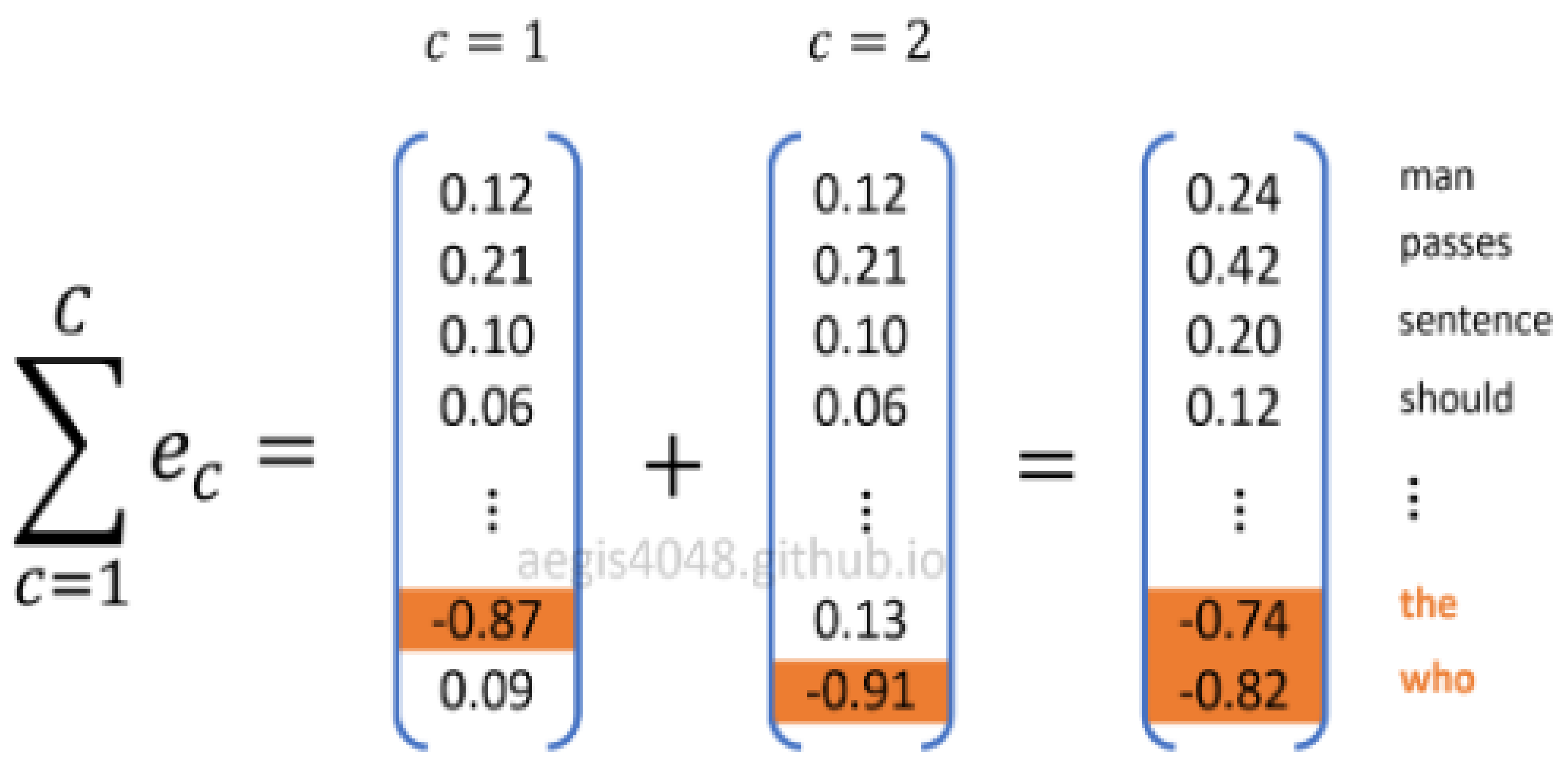


***Figure 12:*** *Sum of prediction errors*

As the weight matrices are optimized, the prediction error for all words in the prediction error vector $\sum_{(c=1)}^{C} e_c$ converges to 0.

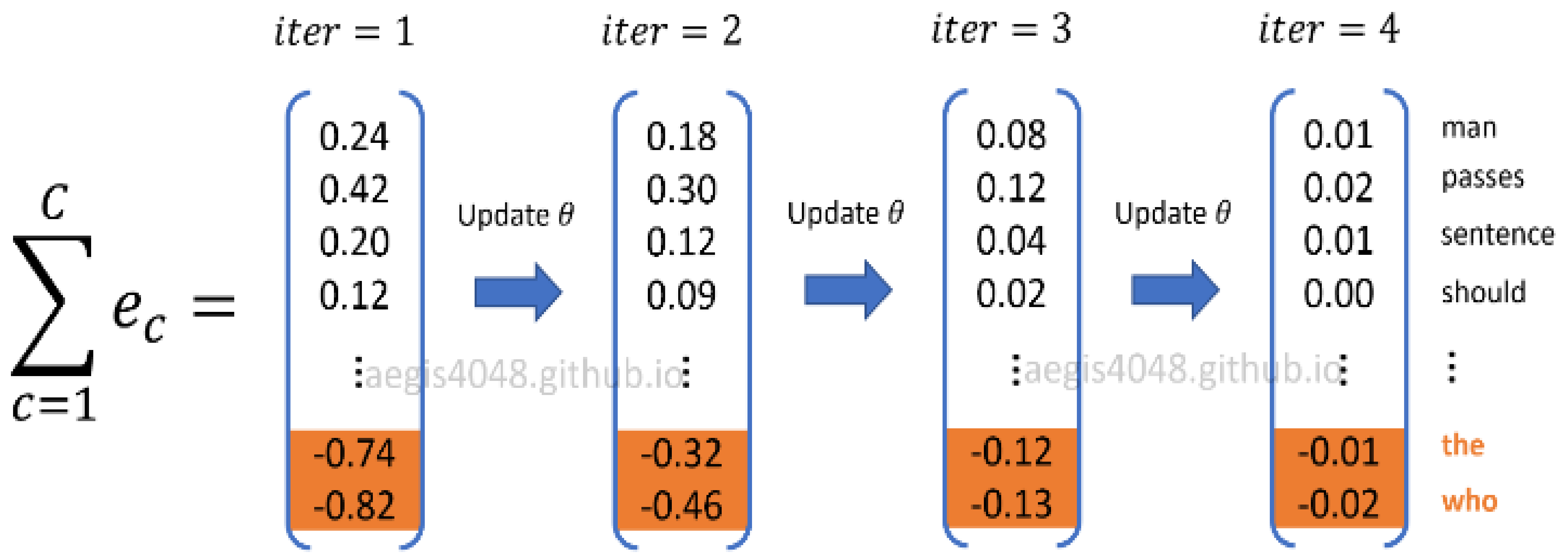


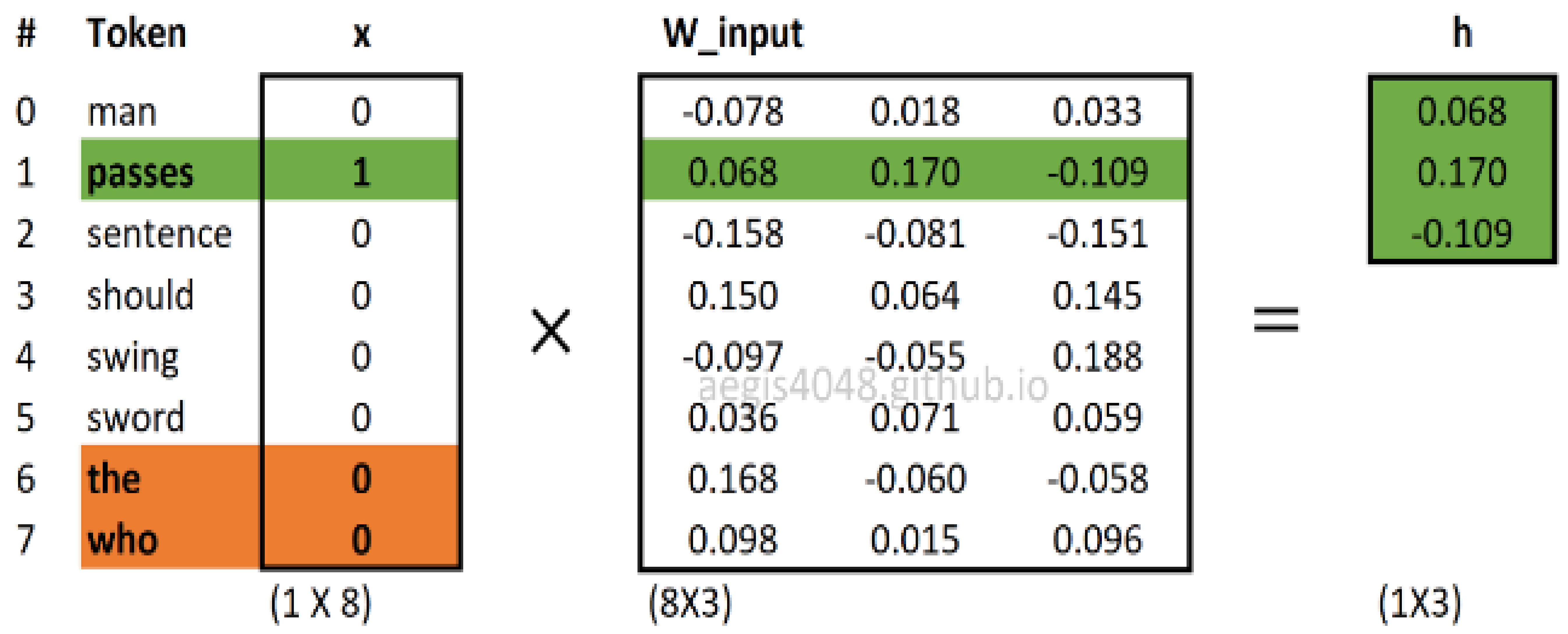
***Figure 13:*** *Prediction errors converging to zero with optimization*

## Numerical demonstration

**Forward propagation: computing hidden (projection) layer**

Center word is *"passes"*. Window size is `size=1`, making *"the"* and *"who"* context words. Hidden layer ($h$) is *looked up* from the input weight matrix. It is computed with eq-12.

Corpus = **"The man who passes the sentence should swing the sword."**



| # | Token | x |
|---|---|---|
| 0 | man | 0 |
| 1 | passes | 1 |
| 2 | sentence | 0 |
| 3 | should | 0 |
| 4 | swing | 0 |
| 5 | sword | 0 |
| 6 | the | 0 |
| 7 | who | 0 |

$$\underset{(1 \times 8)}{x} \times \underset{(8X3)}{W\_input = \begin{bmatrix} -0.078 & 0.018 & 0.033 \\ 0.068 & 0.170 & -0.109 \\ -0.158 & -0.081 & -0.151 \\ 0.150 & 0.064 & 0.145 \\ -0.097 & -0.055 & 0.188 \\ 0.036 & 0.071 & 0.059 \\ 0.168 & -0.060 & -0.058 \\ 0.098 & 0.015 & 0.096 \end{bmatrix}} = \underset{(1X3)}{h = \begin{bmatrix} 0.068 \\ 0.170 \\ -0.109 \end{bmatrix}}$$

***Figure 14:*** *Computing hidden (projection) layer*

**Forward propagation: softmax output layer**

Output layer is a probability distribution of all words, given a center word. It is computed with eq-14. Note that all context windows share the same output layer ($y_{pred}$). Only the errors ($e_c$) are different.

**Forward propagation: softmax output layer**

Output layer is a probability distribution of all words, given a center word. It is computed with eq-14. Note that all context windows share the same output layer ($y_{pred}$). Only the errors ($e_c$) are different.

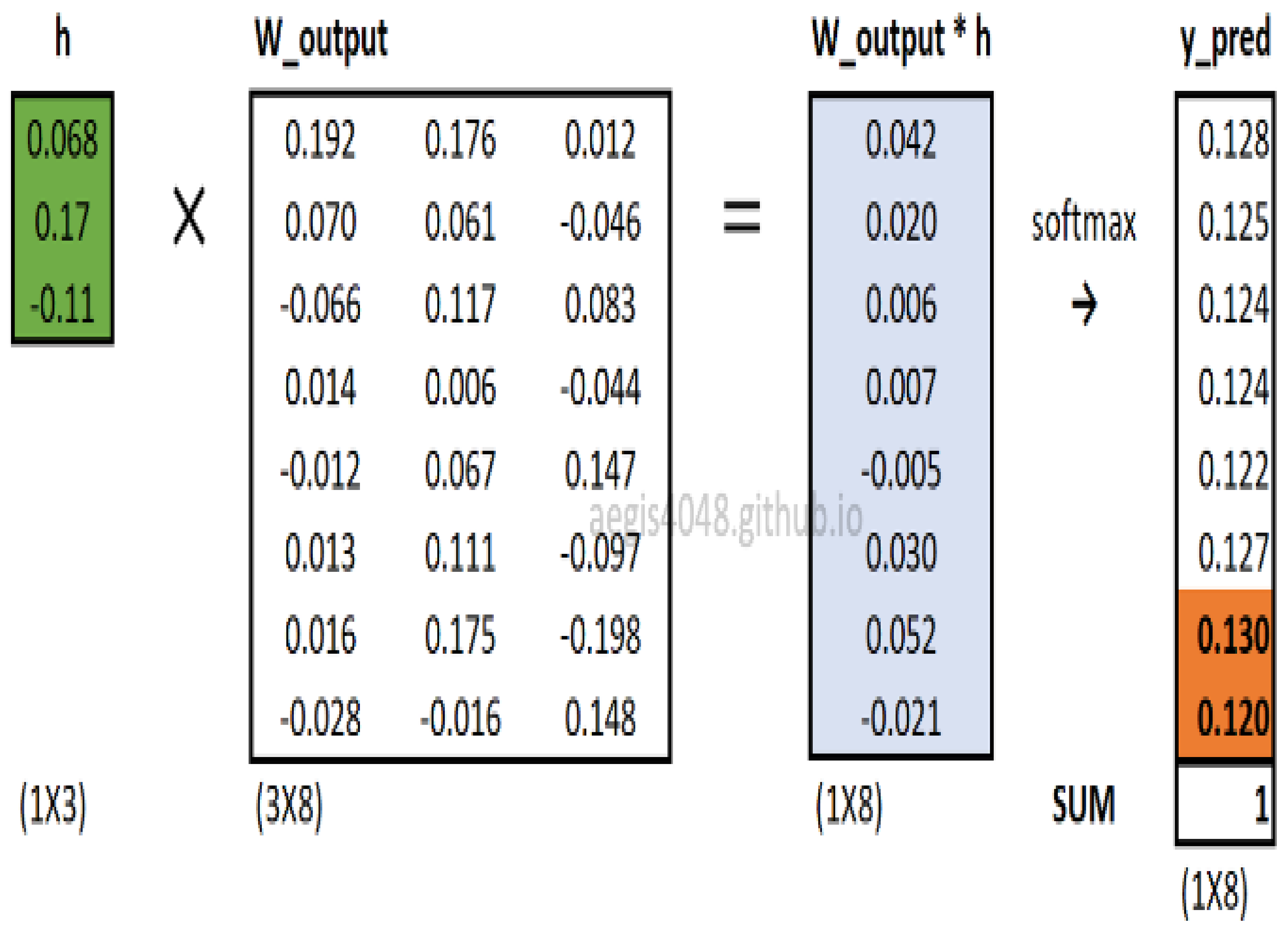
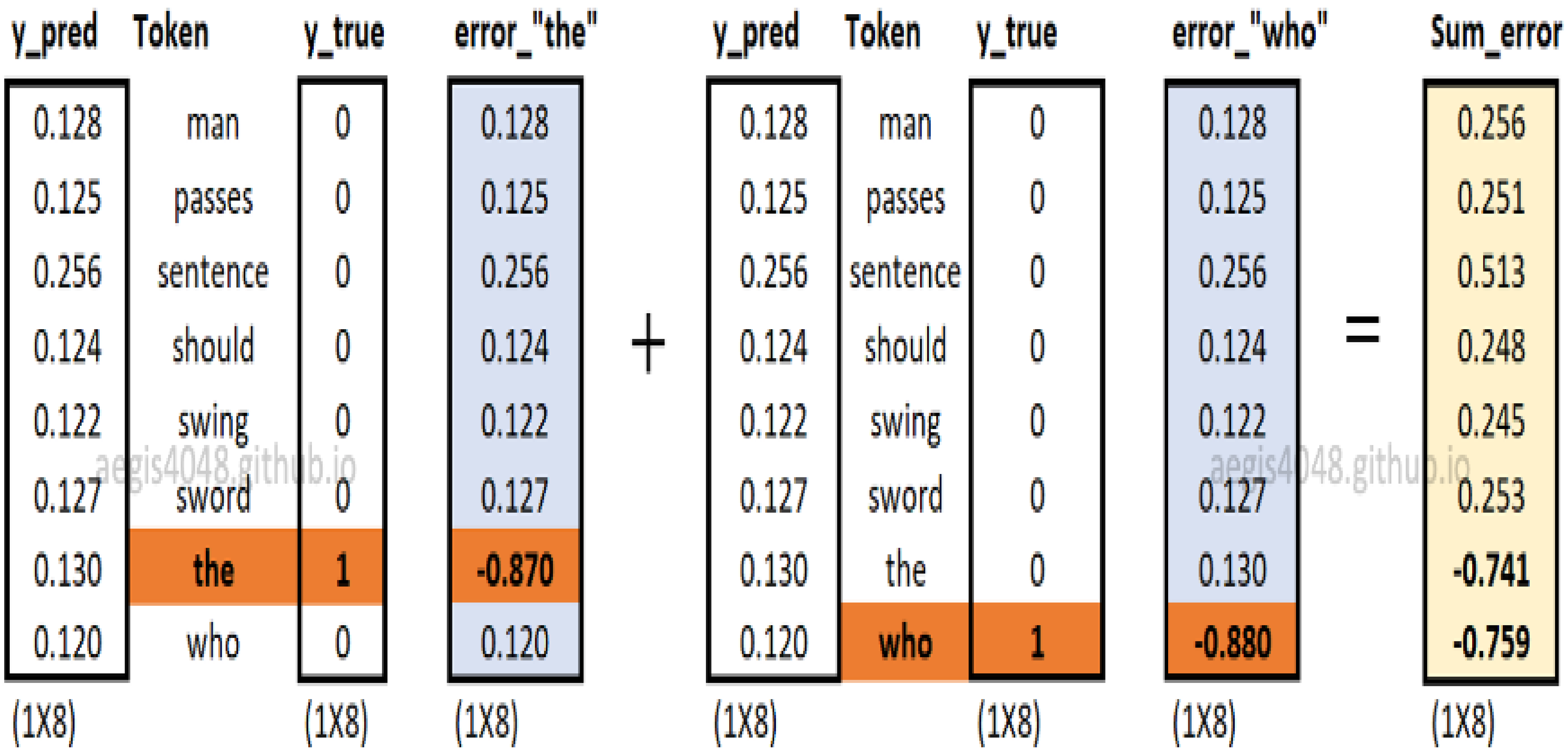


***Figure 15:*** *Softmax output layer*

**Backward propagation: sum of prediction errors**

$C$ different prediction errors are computed, then summed up. In this case, since we set `window=1` above, only two errors are computed.

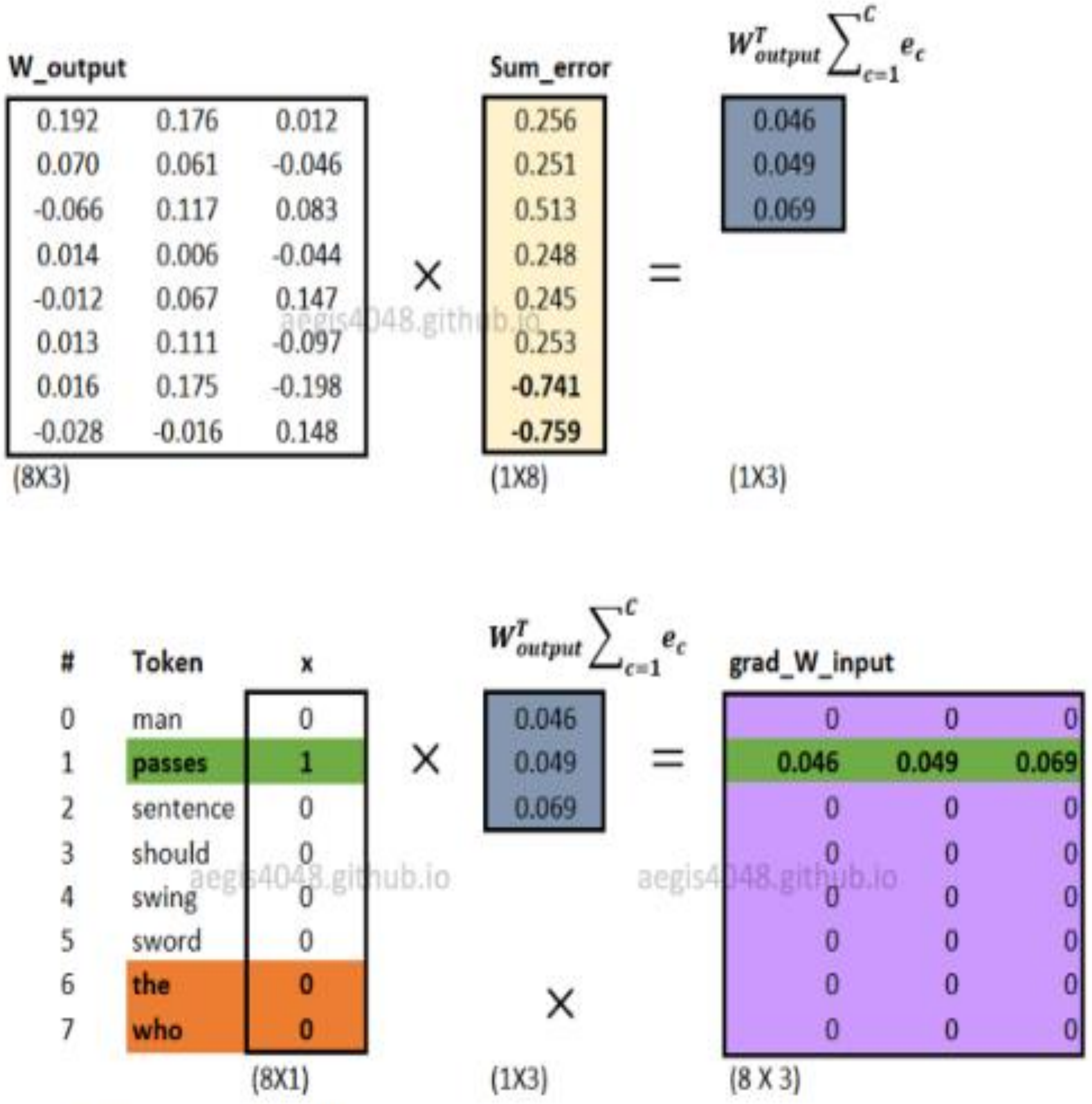


| y_pred | Token | y_true | error_"the" | | y_pred | Token | y_true | error_"who" | | Sum_error |
|---|---|---|---|---|---|---|---|---|---|---|
| 0.128 | man | 0 | 0.128 | | 0.128 | man | 0 | 0.128 | | 0.256 |
| 0.125 | passes | 0 | 0.125 | | 0.125 | passes | 0 | 0.125 | | 0.251 |
| 0.256 | sentence | 0 | 0.256 | | 0.256 | sentence | 0 | 0.256 | | 0.513 |
| 0.124 | should | 0 | 0.124 | + | 0.124 | should | 0 | 0.124 | = | 0.248 |
| 0.122 | swing | 0 | 0.122 | | 0.122 | swing | 0 | 0.122 | | 0.245 |
| 0.127 | sword | 0 | 0.127 | | 0.127 | sword | 0 | 0.127 | | 0.253 |
| 0.130 | **the** | **1** | **-0.870** | | 0.130 | the | 0 | 0.130 | | **-0.741** |
| 0.120 | who | 0 | 0.120 | | 0.120 | **who** | **1** | **-0.880** | | **-0.759** |
| (1X8) | | (1X8) | (1X8) | | (1X8) | | (1X8) | (1X8) | | (1X8) |

***Figure 16:*** *Prediction errors of context words*

## Backward propagation: computing $\nabla W_{input}$

Gradients of input weight matrix ($\frac{\partial J}{\partial W_{input}}$) are computed using eq-18. Note that multiplying $W_{output}^T \sum_{c=1}^{C} e_c$ with the one-hot-encoded input vector ($x$) makes the neural net to update only one word vector that corresponds to the input (center) word.



$$\underset{(8X3)}{\overset{\textbf{W\_output}}{\begin{bmatrix} 0.192 & 0.176 & 0.012 \\ 0.070 & 0.061 & -0.046 \\ -0.066 & 0.117 & 0.083 \\ 0.014 & 0.006 & -0.044 \\ -0.012 & 0.067 & 0.147 \\ 0.013 & 0.111 & -0.097 \\ 0.016 & 0.175 & -0.198 \\ -0.028 & -0.016 & 0.148 \end{bmatrix}}} \times \underset{(1X8)}{\overset{\textbf{Sum\_error}}{\begin{bmatrix} 0.256 \\ 0.251 \\ 0.513 \\ 0.248 \\ 0.245 \\ 0.253 \\ \mathbf{-0.741} \\ \mathbf{-0.759} \end{bmatrix}}} = \underset{(1X3)}{\overset{W_{output}^T \sum_{c=1}^{C} e_c}{\begin{bmatrix} 0.046 \\ 0.049 \\ 0.069 \end{bmatrix}}}$$

$$\begin{array}{ccc} \textbf{\#} & \textbf{Token} & \textbf{x} \\ 0 & \text{man} & 0 \\ 1 & \textbf{passes} & \mathbf{1} \\ 2 & \text{sentence} & 0 \\ 3 & \text{should} & 0 \\ 4 & \text{swing} & 0 \\ 5 & \text{sword} & 0 \\ 6 & \textbf{the} & \mathbf{0} \\ 7 & \textbf{who} & \mathbf{0} \\ & & (8X1) \end{array} \times \underset{(1X3)}{\overset{W_{output}^T \sum_{c=1}^{C} e_c}{\begin{bmatrix} 0.046 \\ 0.049 \\ 0.069 \end{bmatrix}}} = \underset{(8 X 3)}{\overset{\textbf{grad\_W\_input}}{\begin{bmatrix} 0 & 0 & 0 \\ \mathbf{0.046} & \mathbf{0.049} & \mathbf{0.069} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}}$$

$\times$

*Figure 17: Computing input weight matrix gradient* $\nabla W_{input}$

## Backward propagation: computing $\nabla W_{output}$

Gradients of output weight matrix ($\frac{\partial J}{\partial W_{output}}$) are computed using eq-19. Unlike the input weight matrix ($W_{input}$), all word vectors in the output weight matrix ($W_{output}$) are updated.

$$\begin{array}{c} \text{h} \\ \begin{bmatrix} 0.068 \\ 0.17 \\ -0.109 \end{bmatrix} \\ \text{(1X3)} \end{array} \times \begin{array}{c} \text{Sum\_error} \\ \begin{bmatrix} 0.256 \\ 0.251 \\ 0.513 \\ 0.248 \\ 0.245 \\ 0.253 \\ \mathbf{-0.741} \\ \mathbf{-0.759} \end{bmatrix} \\ \text{(1X8)} \end{array} = \begin{array}{c} \text{grad\_W\_output} \\ \begin{bmatrix} 0.017 & 0.044 & -0.028 \\ 0.017 & 0.043 & -0.027 \\ 0.035 & 0.087 & -0.056 \\ 0.017 & 0.042 & -0.027 \\ 0.017 & 0.042 & -0.027 \\ 0.017 & 0.043 & -0.028 \\ -0.050 & -0.126 & 0.081 \\ -0.052 & -0.129 & 0.083 \end{bmatrix} \\ \text{(8 X 3)} \end{array}$$

*Figure 18: Computing output weight matrix gradient* $\nabla W_{output}$

## Backward propagation: updating weight matrices

Input and output weight matrices ($[W_{input} \quad W_{output}]$) are updated using eq-20 and eq-21.

$$\begin{array}{c} \text{W\_input (old)} \\ \begin{bmatrix} -0.078 & 0.018 & 0.033 \\ 0.068 & 0.170 & -0.109 \\ -0.158 & -0.081 & -0.151 \\ 0.150 & 0.064 & 0.145 \\ -0.097 & -0.055 & 0.188 \\ 0.036 & 0.071 & 0.059 \\ 0.168 & -0.060 & -0.058 \\ 0.098 & 0.015 & 0.096 \end{bmatrix} \\ \text{(8X3)} \end{array} - \begin{array}{c} \text{Learning R.} \\ \boxed{0.05} \end{array} \times \begin{array}{c} \text{grad\_W\_input} \\ \begin{bmatrix} 0.000 & 0.000 & 0.000 \\ \mathbf{0.046} & \mathbf{0.049} & \mathbf{0.069} \\ 0.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 \end{bmatrix} \\ \text{(8 X 3)} \end{array} = \begin{array}{c} \text{W\_input (new)} \\ \begin{bmatrix} -0.078 & 0.018 & 0.033 \\ \mathbf{0.066} & \mathbf{0.168} & \mathbf{-0.112} \\ -0.158 & -0.081 & -0.151 \\ 0.150 & 0.064 & 0.145 \\ -0.097 & -0.055 & 0.188 \\ 0.036 & 0.071 & 0.059 \\ 0.168 & -0.060 & -0.058 \\ 0.098 & 0.015 & 0.096 \end{bmatrix} \\ \text{(8X3)} \end{array}$$

*Figure 19: Updating* $W_{input}$

$$\underset{(8X3)}{\overset{\text{W\_output (old)}}{\begin{bmatrix} 0.192 & 0.176 & 0.012 \\ 0.070 & 0.061 & -0.046 \\ -0.066 & 0.117 & 0.083 \\ 0.014 & 0.006 & -0.044 \\ -0.012 & 0.067 & 0.147 \\ 0.013 & 0.111 & -0.097 \\ 0.016 & 0.175 & -0.198 \\ -0.028 & -0.016 & 0.148 \end{bmatrix}}} - \overset{\text{Learning R.}}{\boxed{0.05}} \times \underset{(8\ X\ 3)}{\overset{\text{grad\_W\_output}}{\begin{bmatrix} 0.017 & 0.044 & -0.028 \\ 0.017 & 0.043 & -0.027 \\ 0.035 & 0.087 & -0.056 \\ 0.017 & 0.042 & -0.027 \\ 0.017 & 0.042 & -0.027 \\ 0.017 & 0.043 & -0.028 \\ -0.050 & -0.126 & 0.081 \\ -0.052 & -0.129 & 0.083 \end{bmatrix}}} = \underset{(8X3)}{\overset{\text{W\_output (new)}}{\begin{bmatrix} 0.191 & 0.174 & 0.013 \\ 0.069 & 0.059 & -0.045 \\ -0.068 & 0.113 & 0.086 \\ 0.013 & 0.004 & -0.043 \\ -0.013 & 0.065 & 0.148 \\ 0.012 & 0.109 & -0.096 \\ 0.019 & 0.181 & -0.202 \\ -0.025 & -0.010 & 0.144 \end{bmatrix}}}$$

*Figure 20: Updating $W_{output}$*

Note that for each iteration in the learning process, all weights in $W_{output}$ are updated, but only one row vector that corresponds to the center word is updated in $W_{input}$. When the model finishes updating both of the weight matrices, then one iteration is completed. The model then moves to the next iteration with the next center word. However, remember that this uses eq-8 as the cost function and assumes `stochastic gradient descent`. This means that one update is made for each training example. If eq-9 is used as a cost function instead (which is almost never the case), then one update is made for all $T$ training examples in the corpus.