Deep Learning for Natural Language Processing (NLP)

Representation of words as vectors

One-Hot Encoding

write all words → give them numbers.

Document ID	Text
D1	Dog bites man.
D2	Man bites dog.
D3	Dog eats meat.
D4	Man eats food.

Our toy corpus having six unique words: dog = 1, bites = 2, man = 3, meat = 4 , food = 5, eats = 6.

One-Hot Encoding

Document ID	Text
D1	Dog bites man.
D2	Man bites dog.
D3	Dog eats meat.
D4	Man eats food.

Our toy corpus having six unique words: dog = 1, bites = 2, man = 3, meat = 4 , food = 5, eats = 6.

D3 is represented as
[[1 0 0 0 0 0], [0 0 0 0 0 1], [0 0 0 1 0 0]].

Bag of Words (BoW) → Ek me likhdo.

Document ID	Text
D1	Dog bites man.
D2	Man bites dog.
D3	Dog eats meat.
D4	Man eats food.

Our toy corpus having six unique words: dog = 1, bites = 2, man = 3, meat = 4 , food = 5, eats = 6.

Bag of Words (BoW)

Document ID	Text
D1	Dog bites man.
D2	Man bites dog.
D3	Dog eats meat.
D4	Man eats food.

Our toy corpus having six unique words: dog = 1, bites = 2, man = 3, meat = 4 , food = 5, eats = 6.

D3 is represented as [1 0 0 1 0 1]

2-Grams

Bigram (Bag of 2-Grams)

Document ID	Text
D1	Dog bites man.
D2	Man bites dog.
D3	Dog eats meat.
D4	Man eats food.

{dog bites,
bites man,
man bites,
bites dog,
dog eats,
eats meat,
man eats,
eats food}.

- We need eight dimensions to represent a word in document.

Bigram (Bag of 2-Grams)

Document ID	Text
D1	Dog bites man.
D2	Man bites dog.
D3	Dog eats meat.
D4	Man eats food.

{dog bites,
bites man,
man bites,
bites dog,
dog eats,
eats meat,
man eats,
eats food}.

- We need eight dimensions to represent a word in document.
- Bigram representation for D1 is as follows: D1 : [1,1,0,0,0,0,0,0]

Bigram (Bag of 2-Grams)

Document ID	Text
D1	Dog bites man.
D2	Man bites dog.
D3	Dog eats meat.
D4	Man eats food.

{dog bites,
bites man,
man bites,
bites dog,
dog eats,
eats meat,
man eats,
eats food}.

- We need eight dimensions to represent a word in document.
- Bigram representation for D1 is as follows: D1 : [1,1,0,0,0,0,0,0]

Bag of N-Grams (BoN): Break text into chunks of n contiguous words (or tokens).

Bigram (Bag of 2-Grams)

Document ID	Text
D1	Dog bites man.
D2	Man bites dog.
D3	Dog eats meat.
D4	Man eats food.

{dog bites,
bites man,
man bites,
bites dog,
dog eats,
eats meat,
man eats,
eats food}.

- We need eight dimensions to represent a word in document.
- Bigram representation for D1 is as follows: D1 : [1,1,0,0,0,0,0,0]

Limitation: BoW or BoN representations does not have semantic meaning.

Bag of N-Grams (BoN): Break text into chunks of n contiguous words (or tokens).

Word Embedding

Vector representation

CBOW

Skip gram.

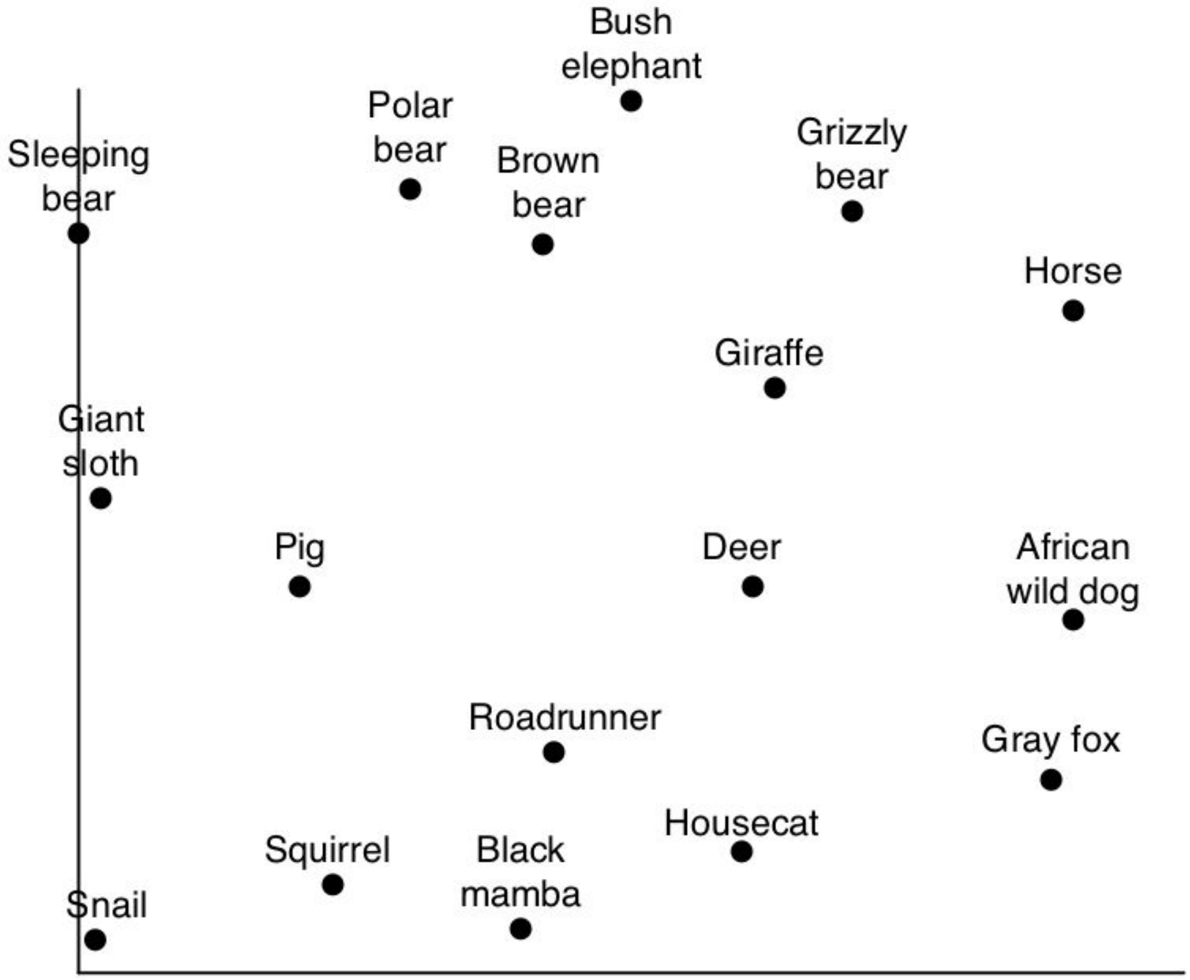
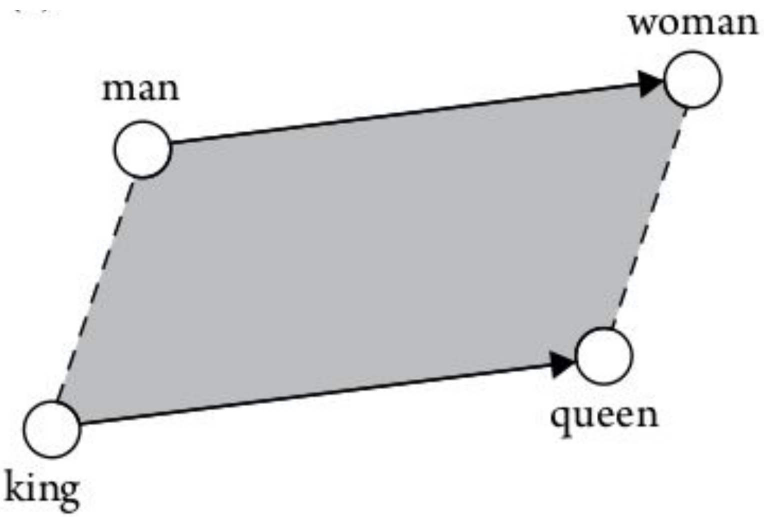


Figure 20-1: A collection of animals, organized roughly by land speed horizontally and adult weight vertically, though those axis labels aren't shown (data from Reisner 2020)

Word Embeddings (dense vector)

$$\text{Horse} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

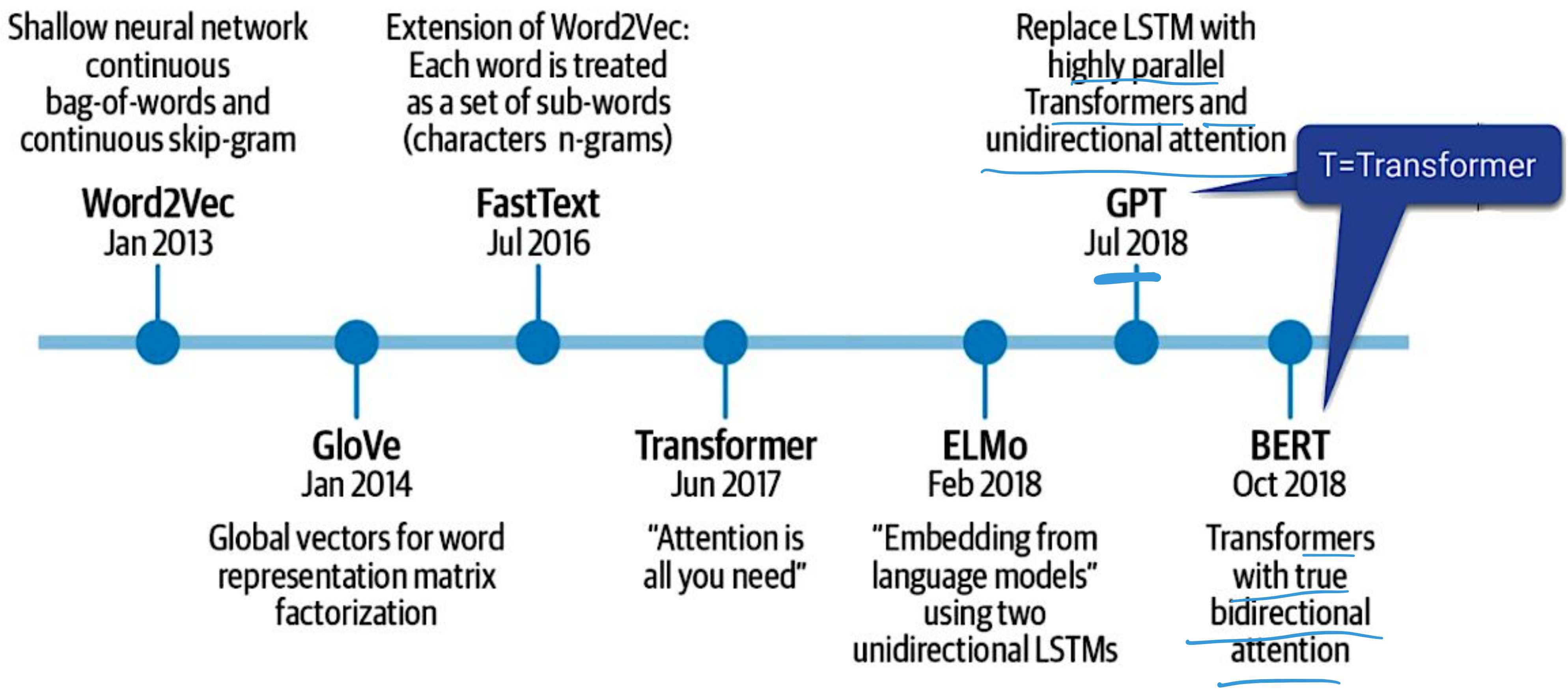
Word analogies as parallelograms



Man is to king as woman is to queen.

$$\overrightarrow{\text{man}} - \overrightarrow{\text{woman}} \approx \overrightarrow{\text{king}} - \overrightarrow{\text{queen}}$$

Progress over the years



Word2vec

Input:

- Text corpora: Wikipedia, Twitter, Common Crawl.
- V: a predefined vocabulary
- d: dimension of word vectors

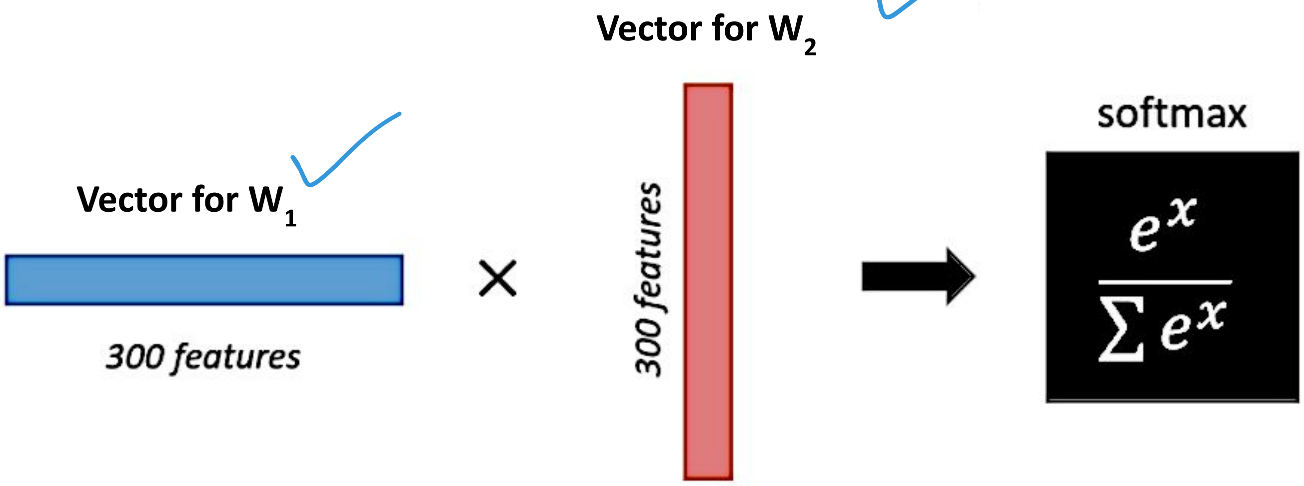
Output: $f : V \to \mathbb{R}^d$

Train a classifier on prediction task

Idea: Train a classifier on a prediction task: w_1 likely to show up near w_2?

Use running text as implicitly supervised training data!

Word2vec (intuitive idea)



= Probability that if you randomly pick a word nearby W_1, that it is W_2

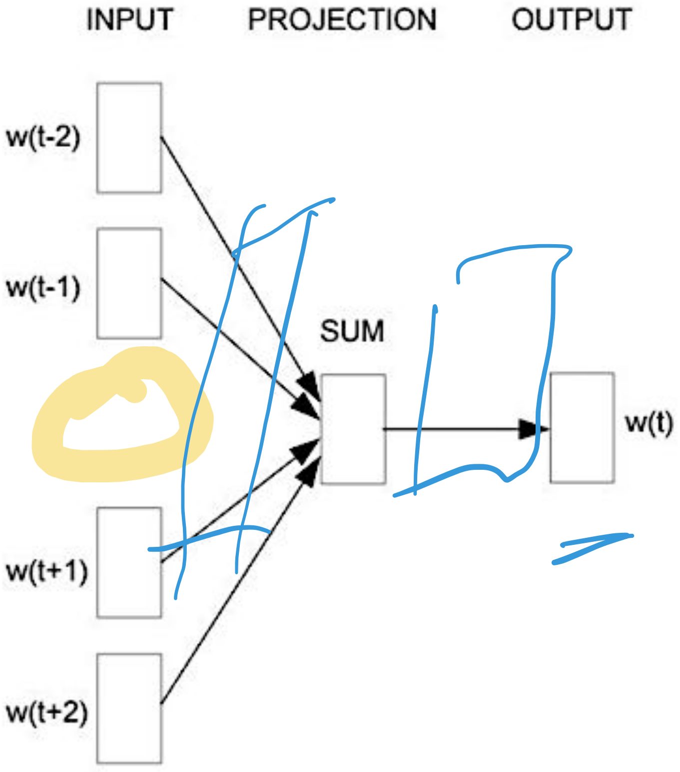
Main Idea of word2vec

- The word2vec model is a **neural network-based** approach for generating word embeddings.
- The model is trained on a large corpus of text data using either a skip-gram or a continuous bag-of-words (CBOW) architecture.
- During training, the model learns to associate words in the embedding space that appear in similar contexts.

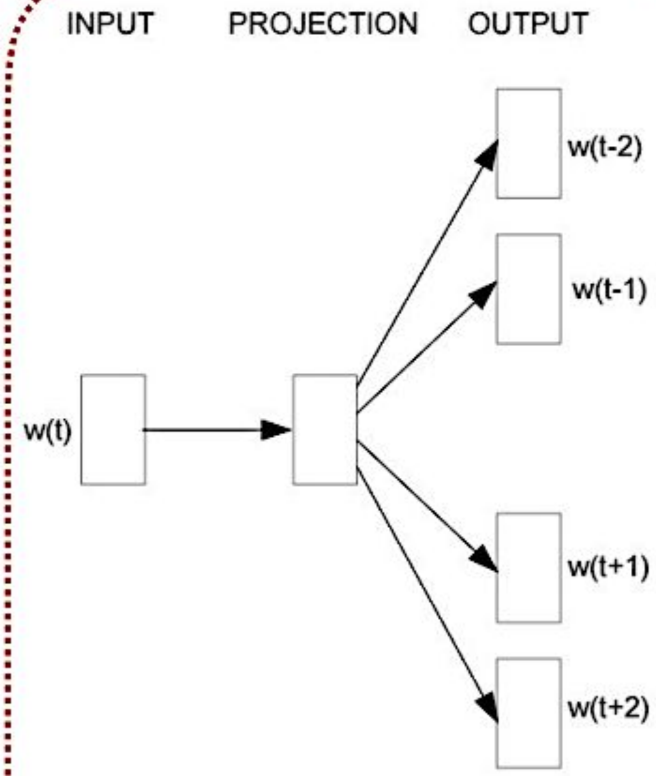
Word2vec

Content window = 2

w_t



Continuous Bag of Words (CBOW)

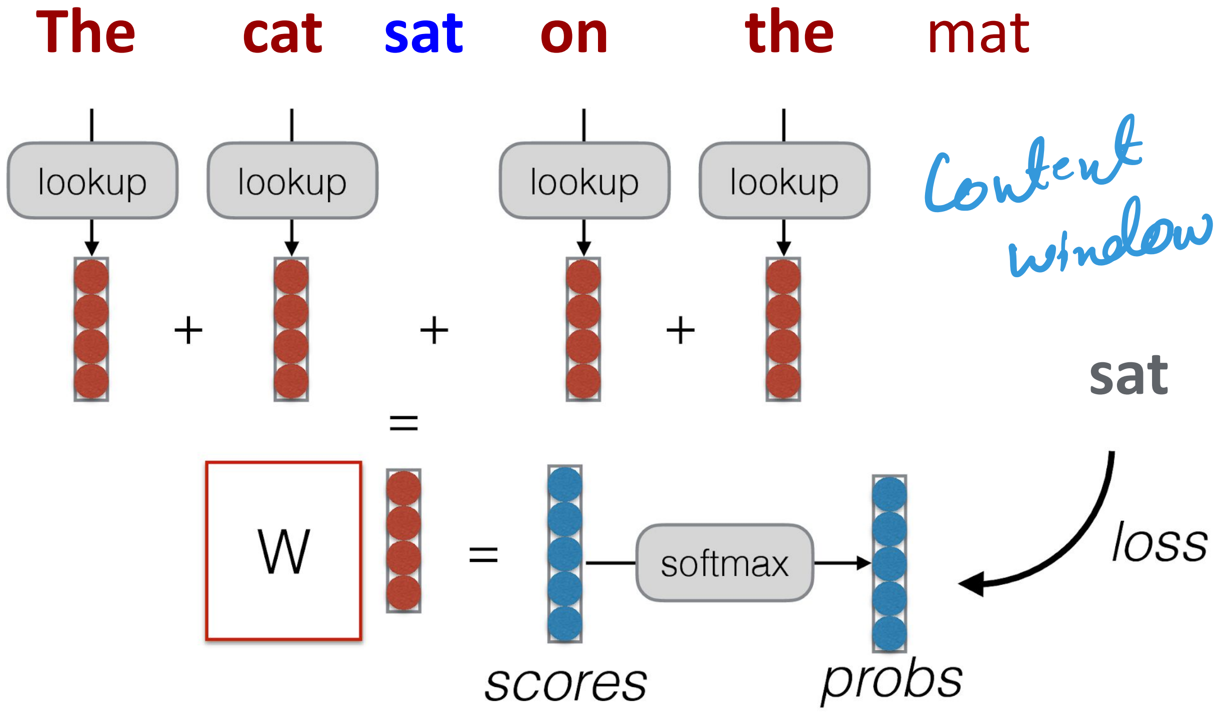


Skip-grams

Continuous Bag of Words Model (CBOW)

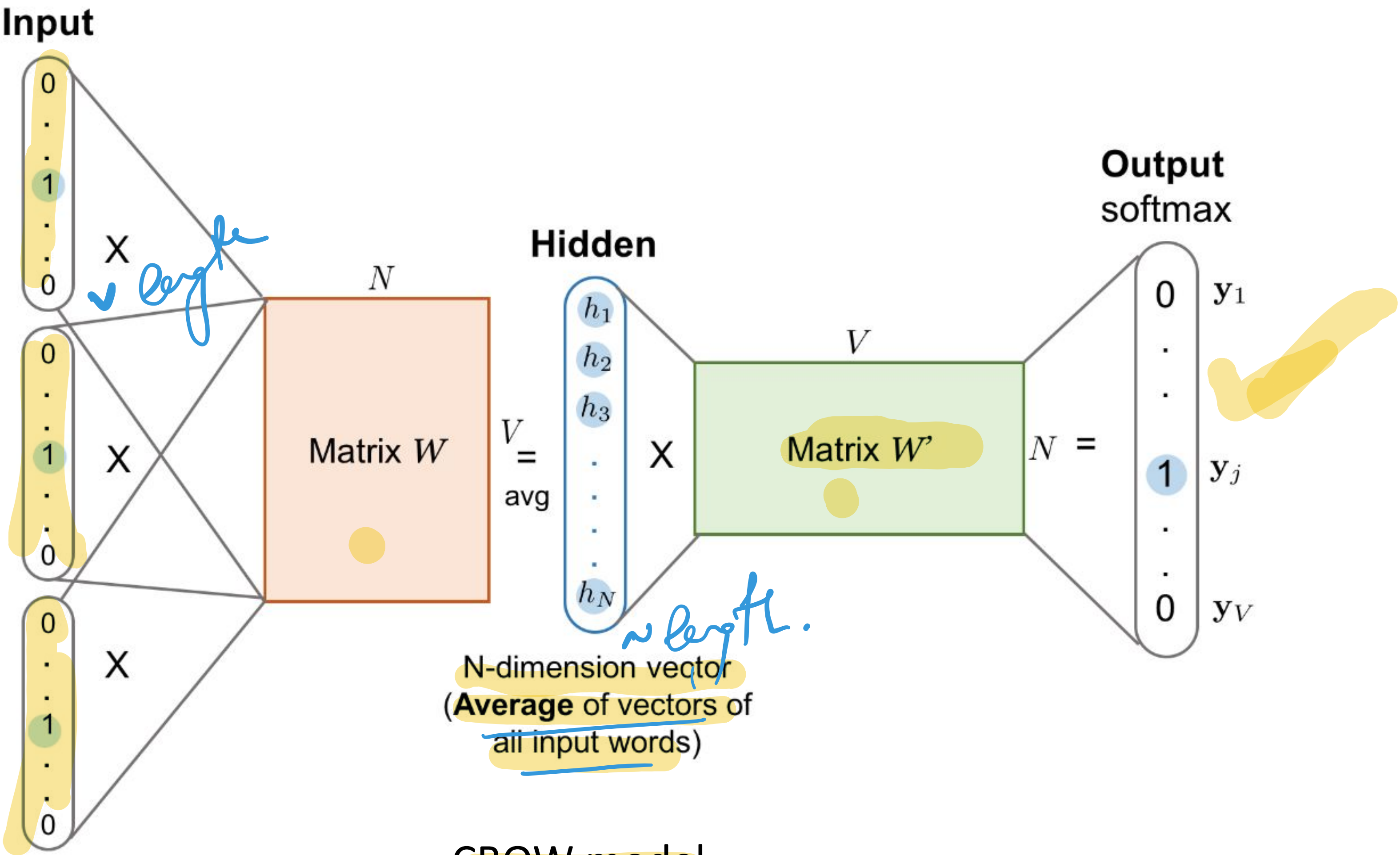
Continuous Bag of Words Model (CBOW)

- Generate/predict the center word by considering the context words.



Content window = 2.

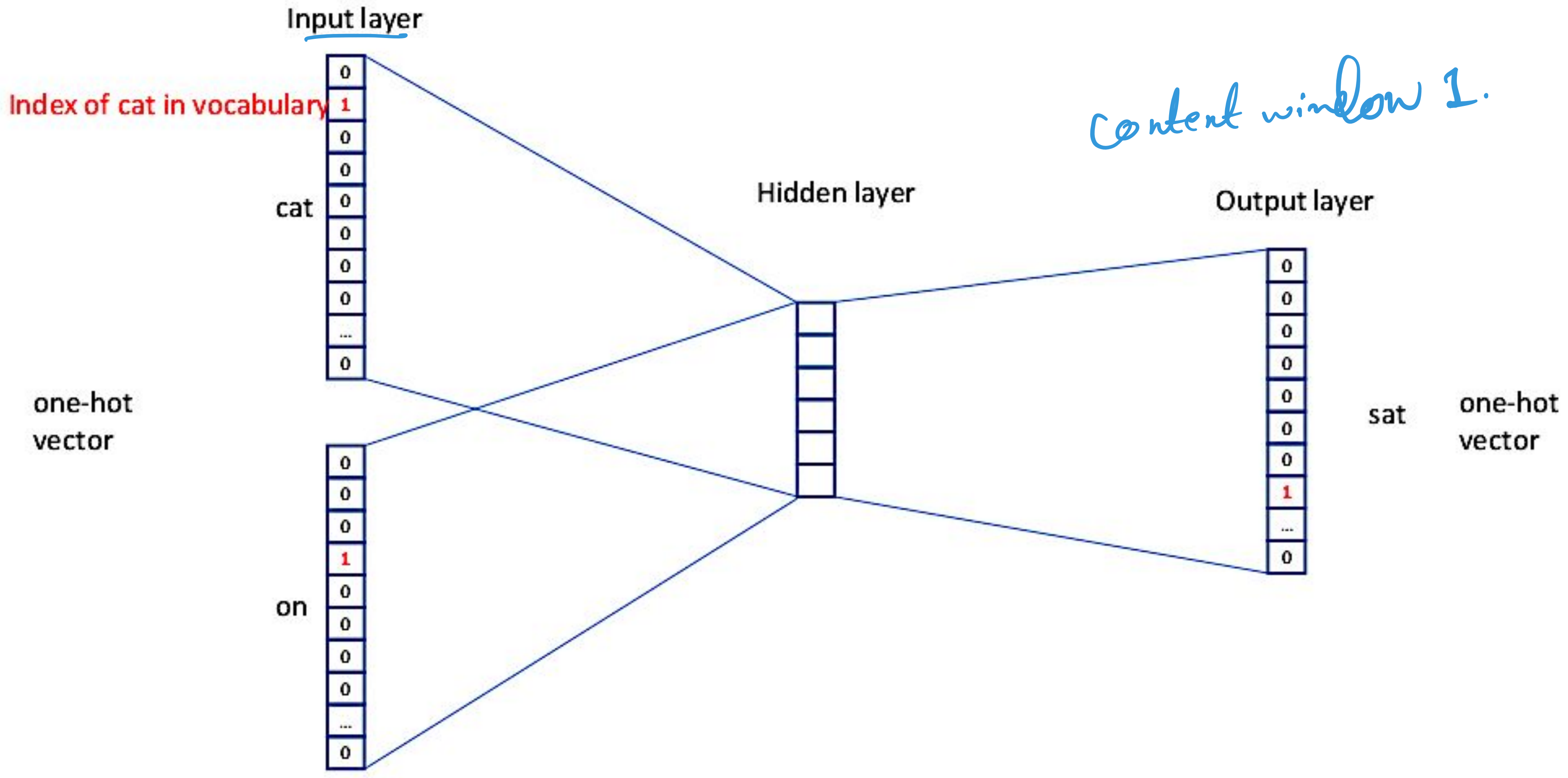
Continuous Bag of Words Model (CBOW)



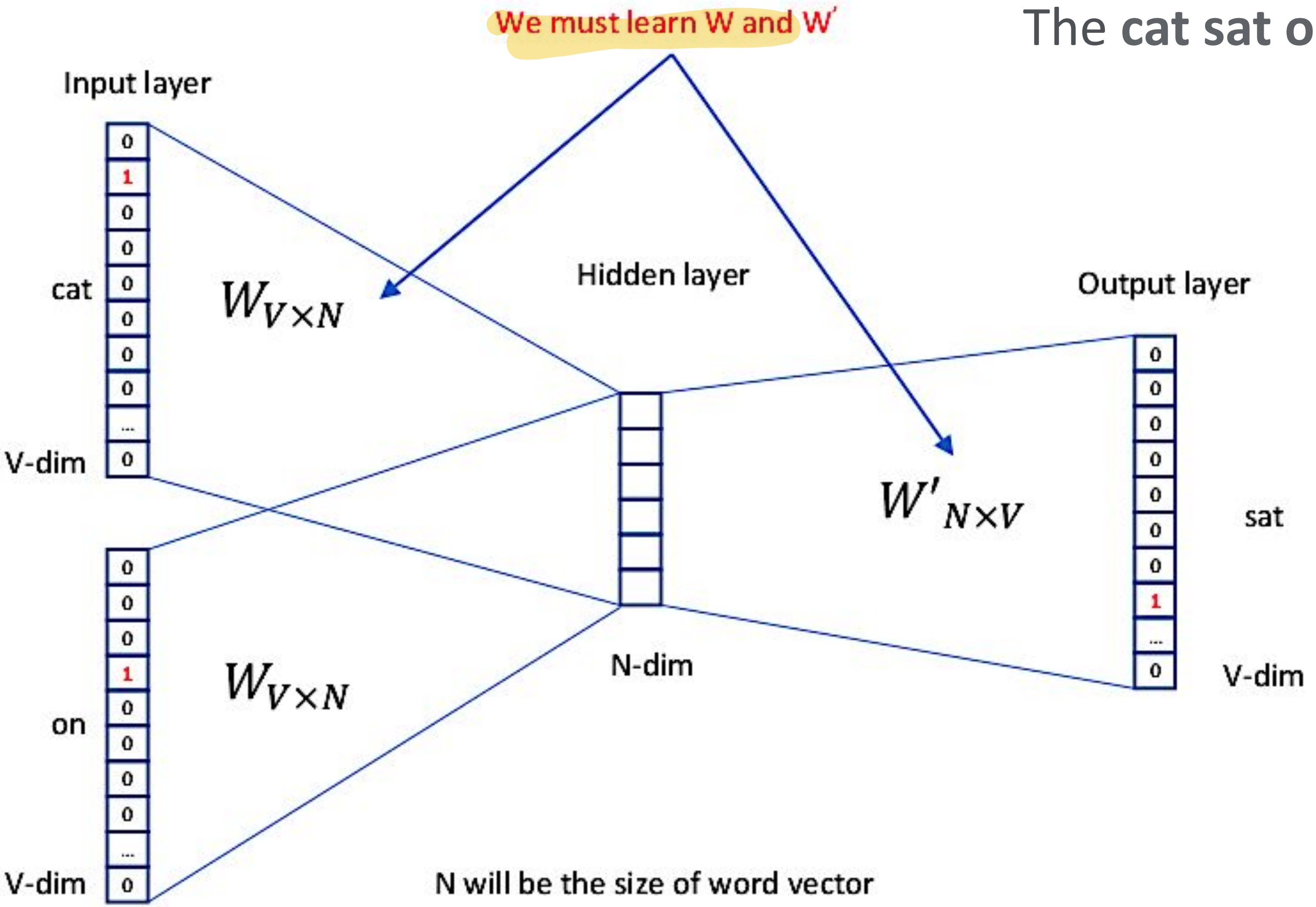
CBOW model

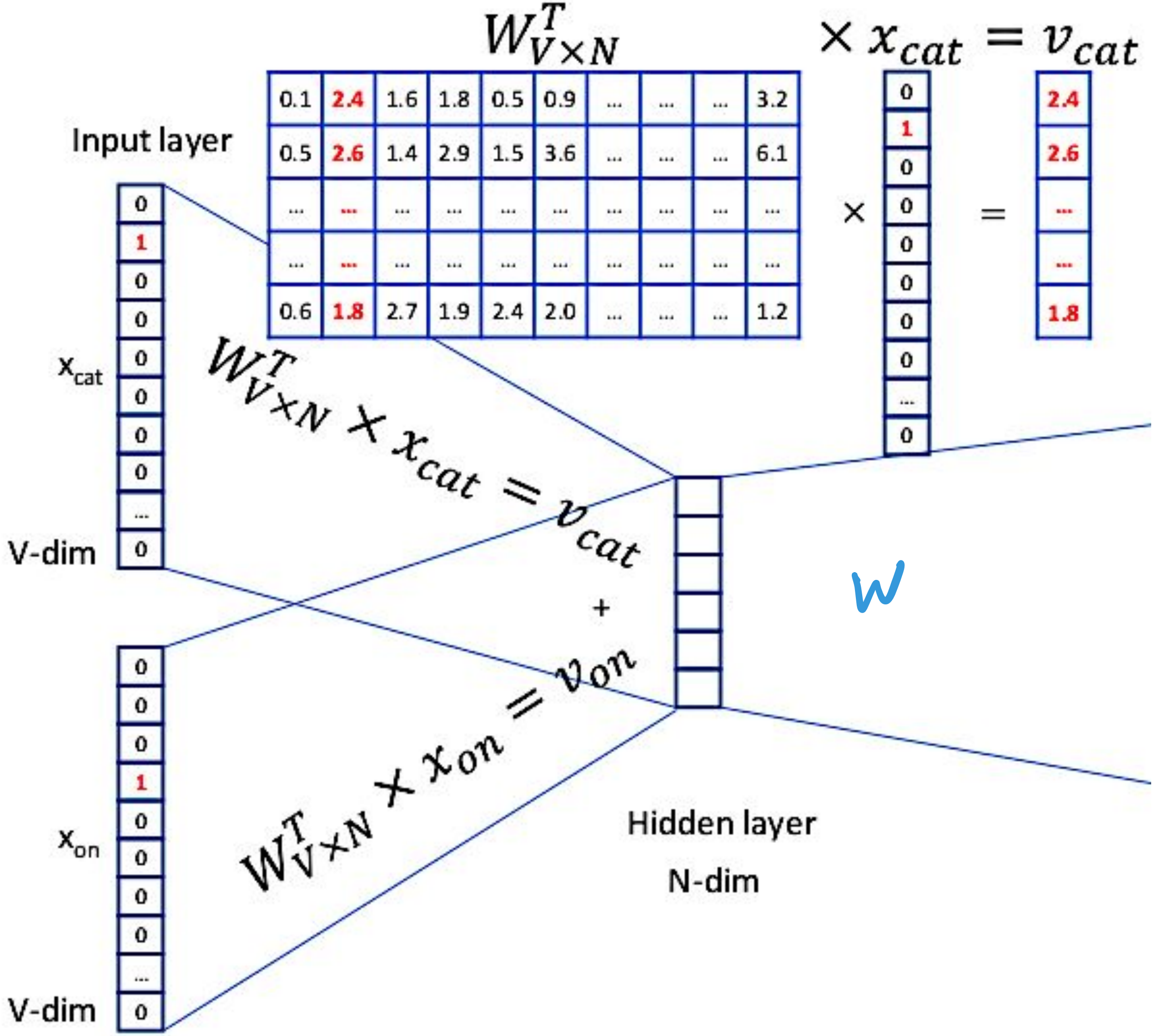
The **cat sat on** the mat

Content window 1.

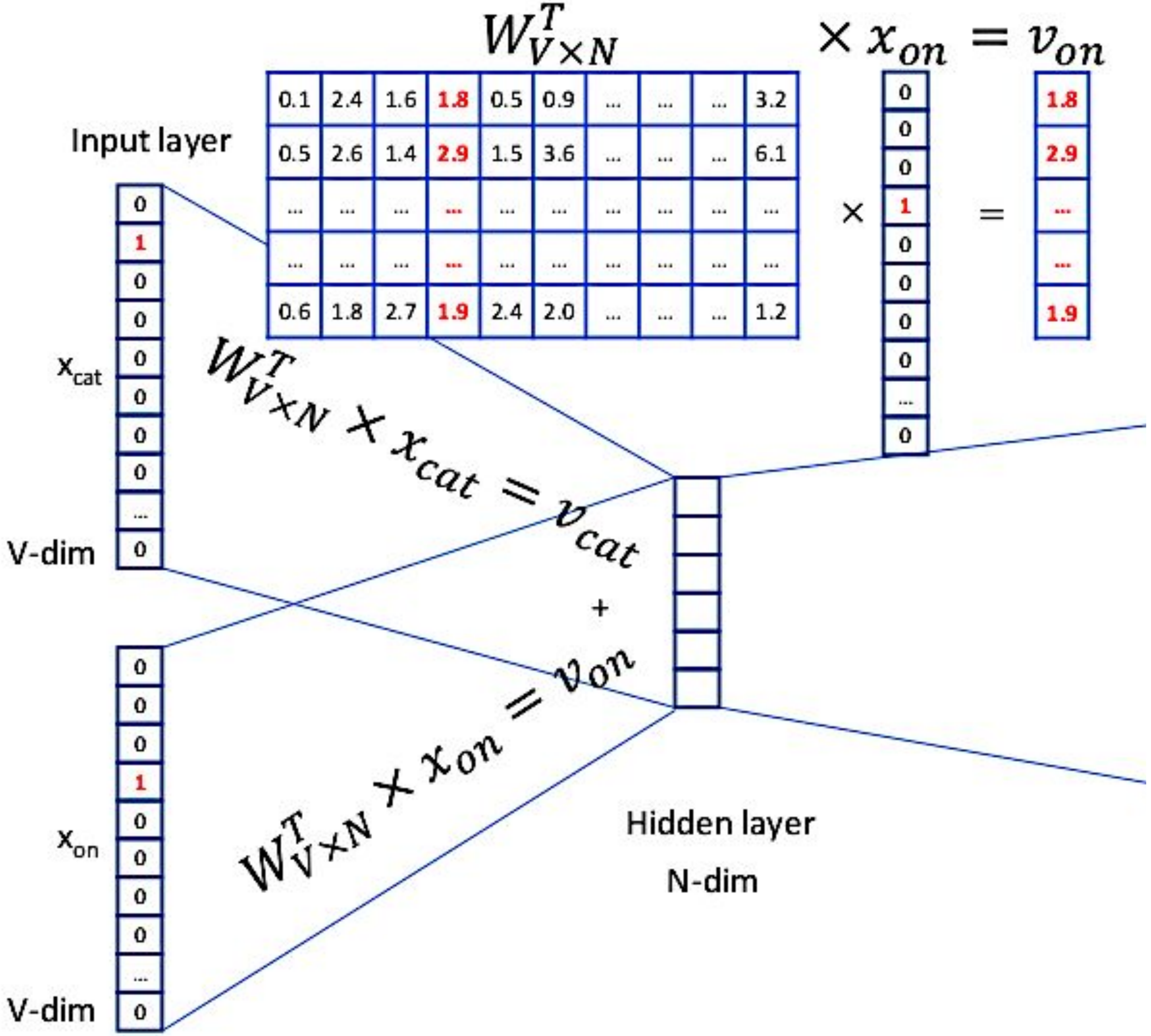


The **cat sat on** the mat

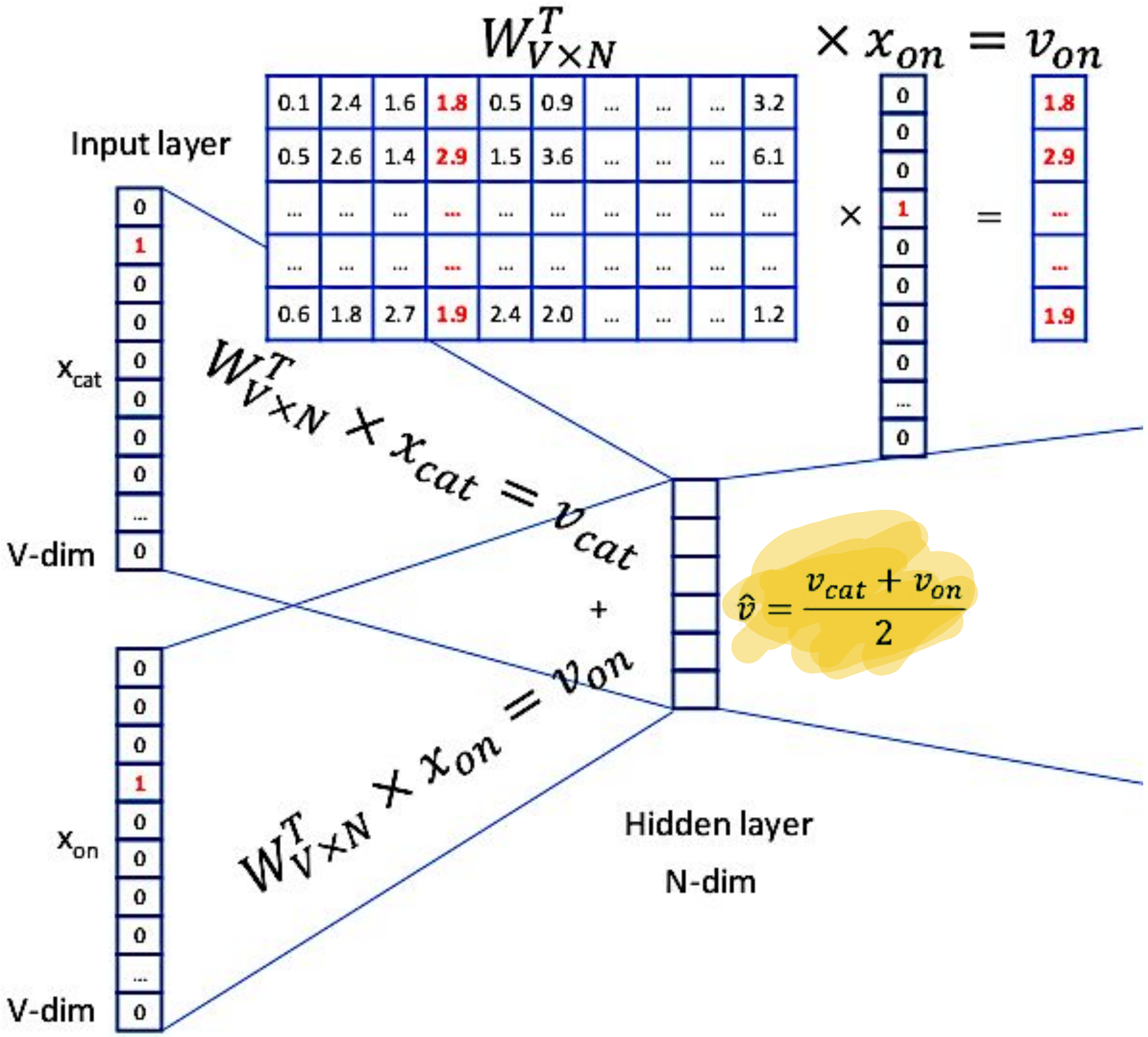




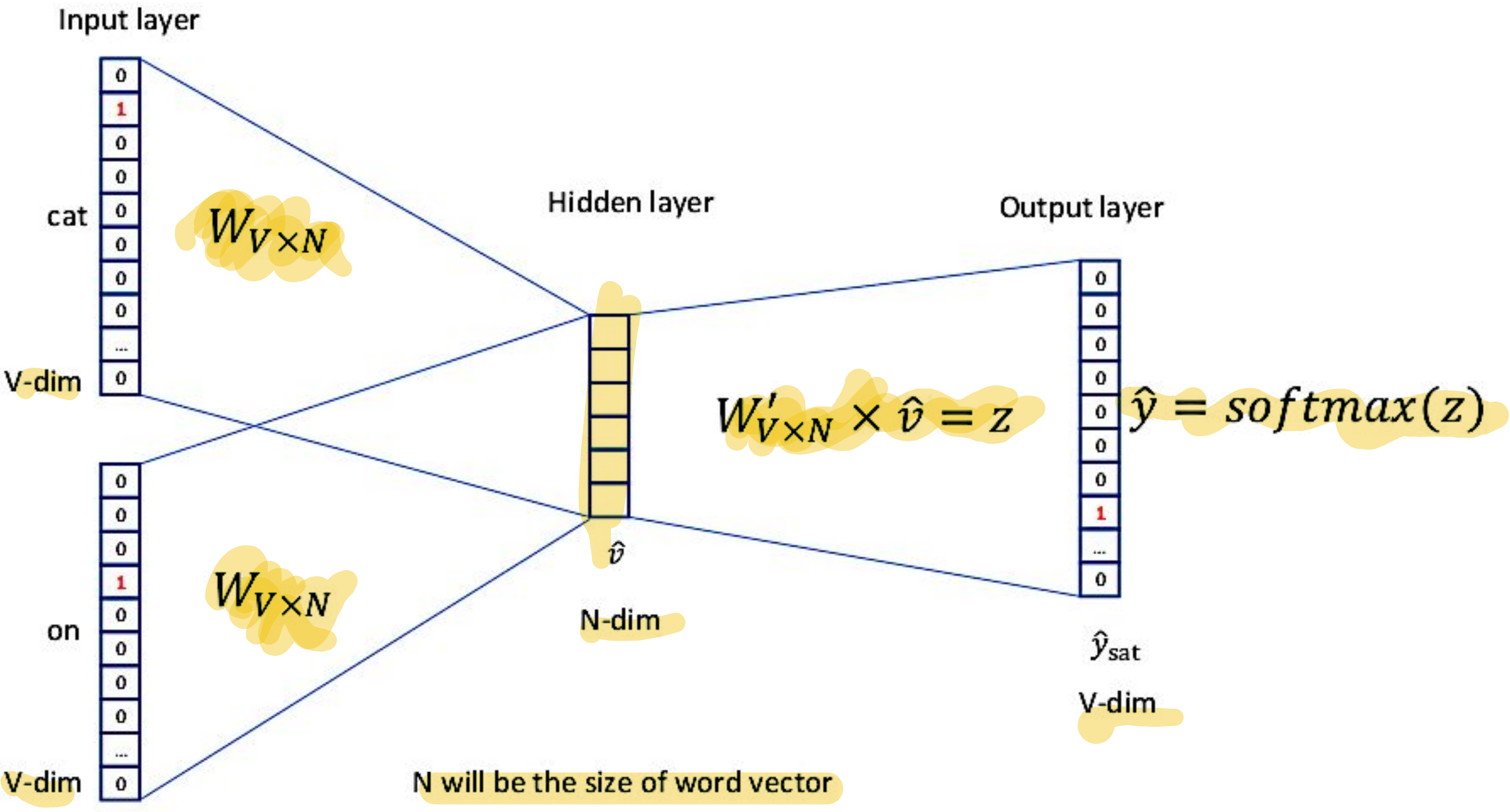
$W_{V\times N}^T \times x_{cat} = v_{cat}$
Input layer
0.1 2.4 1.6 1.8 0.5 0.9 ... ... 3.2
0.5 2.6 1.4 2.9 1.5 3.6 ... ... 6.1
0.6 1.8 2.7 1.9 2.4 2.0 ... ... 1.2
$\times$
$=$
2.4
2.6
1.8
x_{cat}
V-dim
$W_{V\times N}^T \times x_{cat} = v_{cat}$
+
$W_{V\times N}^T \times x_{on} = v_{on}$
x_{on}
V-dim
W
Hidden layer
N-dim

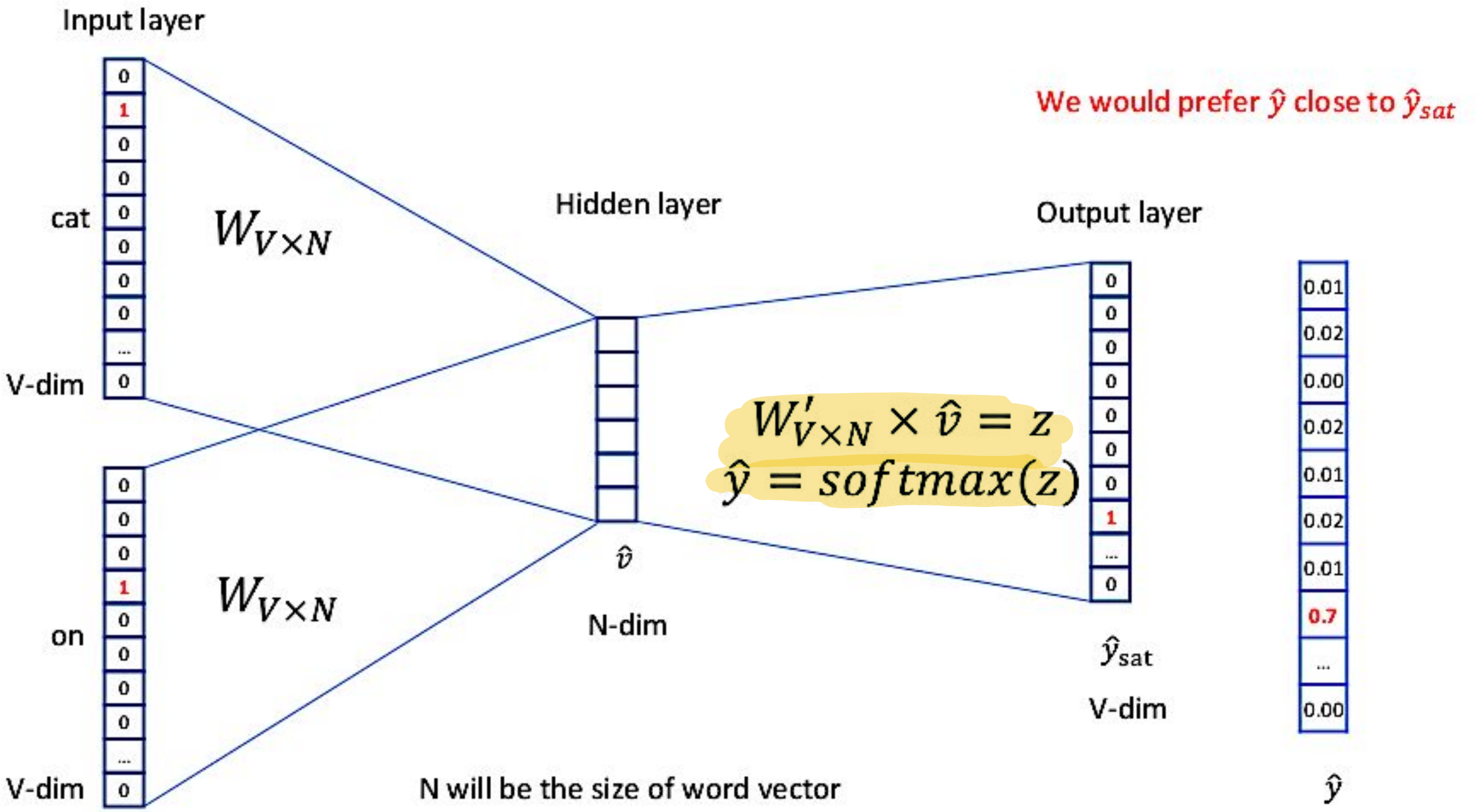


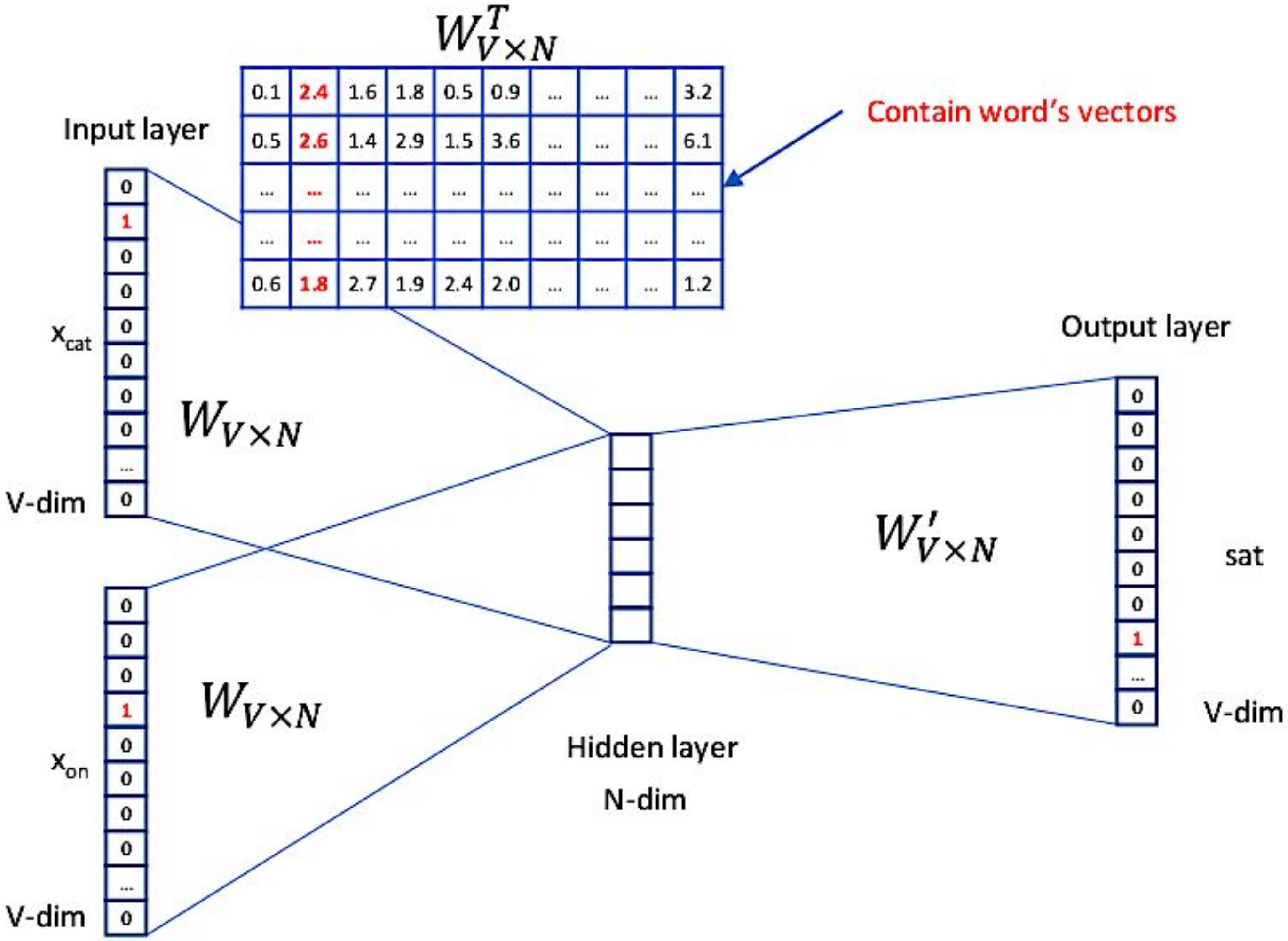
$W_{V\times N}^T \times x_{on} = v_{on}$
Input layer
0.1 2.4 1.6 1.8 0.5 0.9 … … … 3.2
0.5 2.6 1.4 2.9 1.5 3.6 … … … 6.1
0.6 1.8 2.7 1.9 2.4 2.0 … … … 1.2
1.8
2.9
1.9
x_{cat}
V-dim
$W_{V\times N}^T \times x_{cat} = v_{cat}$
+
$W_{V\times N}^T \times x_{on} = v_{on}$
x_{on}
V-dim
Hidden layer
N-dim



$W^T_{V\times N} \times x_{on} = v_{on}$
0.1 2.4 1.6 1.8 0.5 0.9 ... ... 3.2
0.5 2.6 1.4 2.9 1.5 3.6 ... ... 6.1
0.6 1.8 2.7 1.9 2.4 2.0 ... ... 1.2
1.8 2.9 ... ... 1.9
Input layer
x_{cat}
V-dim
$W^T_{V\times N} \times x_{cat} = v_{cat}$
+
$W^T_{V\times N} \times x_{on} = v_{on}$
x_{on}
V-dim
$\hat{v} = \frac{v_{cat} + v_{on}}{2}$
Hidden layer
N-dim


Input layer
Hidden layer
Output layer
cat
on
V-dim
V-dim
N-dim
V-dim
$W_{V\times N}$
$W_{V\times N}$
$W'_{V\times N} \times \hat{v} = z$
$\hat{y} = softmax(z)$
$\hat{v}$
$\hat{y}_{sat}$
0
1
0
0
0
0
0
0
...
0
0
0
0
1
0
0
0
0
...
0
0
0
0
0
0
0
0
1
...
0
N will be the size of word vector


Input layer
cat
V-dim
on
V-dim
$W_{V \times N}$
$W_{V \times N}$
Hidden layer
$\hat{v}$
N-dim
Output layer
We would prefer $\hat{y}$ close to $\hat{y}_{sat}$
$W'_{V \times N} \times \hat{v} = z$
$\hat{y} = softmax(z)$
$\hat{y}_{sat}$
V-dim
0.01
0.02
0.00
0.02
0.01
0.02
0.01
0.7
...
0.00
$\hat{y}$
N will be the size of word vector

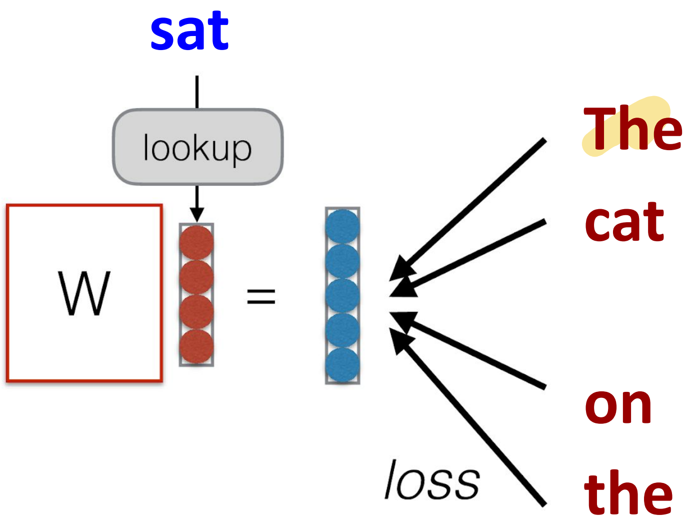


We can consider either W or W' as the word's representation. Or even take the average.

Skip-Gram Task

Skip-gram

- Predict each word in the context given the word



Skip-gram training data example

data of CBOW will be the reverse.

The man who passes the sentence should swing the sword

context window = 2

Input	Context words
The	["man" "who"]
man	The who passes.
who	The man passes
passes	

Skip-gram training data example

The man who passes the sentence should swing the sword

Sliding window (size = 5)	Target word	Context
[The man who]	the	man, who
[The man who passes]	man	the, who, passes
[The man who passes the]	who	the, man, passes, the
[man who passes the sentence]	passes	man, who, the, sentence
...	...	...
[sentence should swing the sword]	swing	sentence, should, the, sword
[should swing the sword]	the	should, swing, sword
[swing the sword]	sword	swing, the

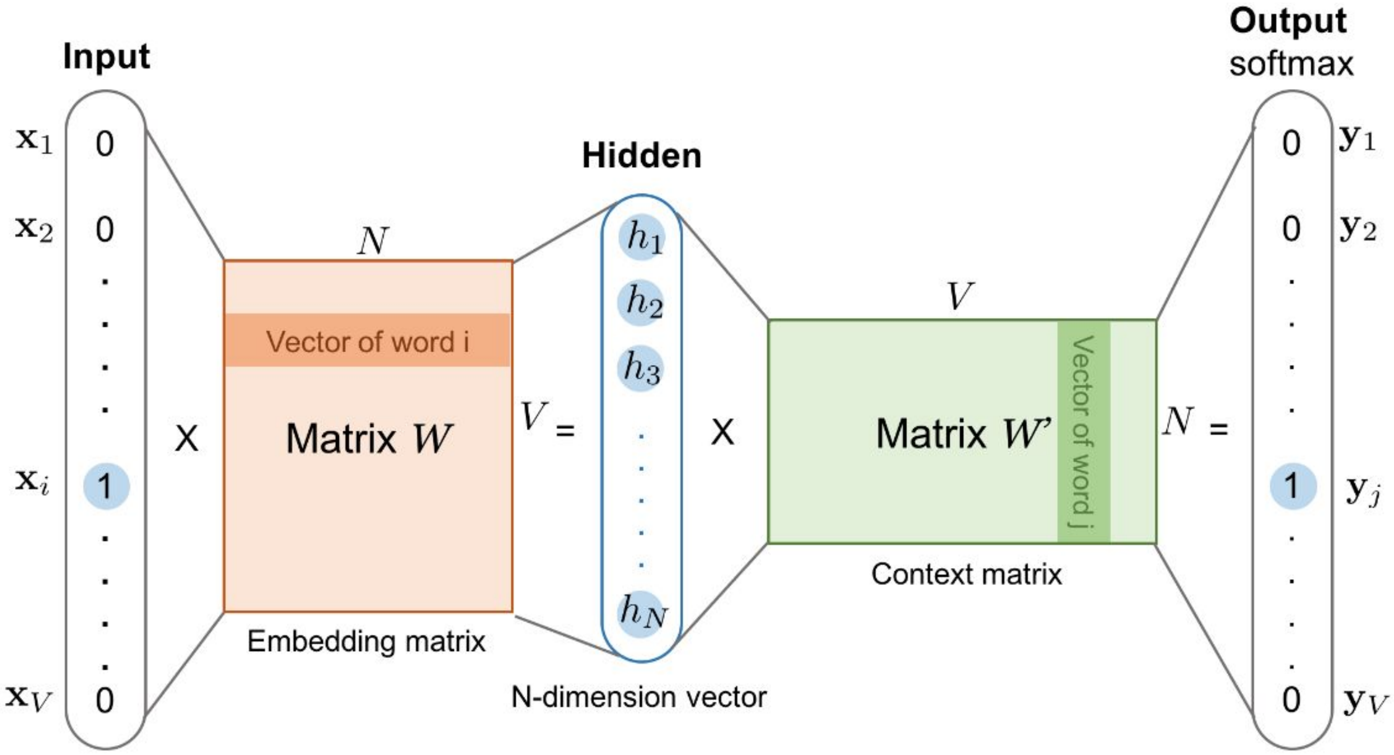
Skip-gram

The man who passes the sentence should swing the sword

- Each context-target pair is treated as a new observation in the data.
- For example, the target word “swing” in the above case produces four training samples:
  - (“swing”, “sentence”), (“swing”, “should”), (“swing”, “the”), and (“swing”, “sword”).

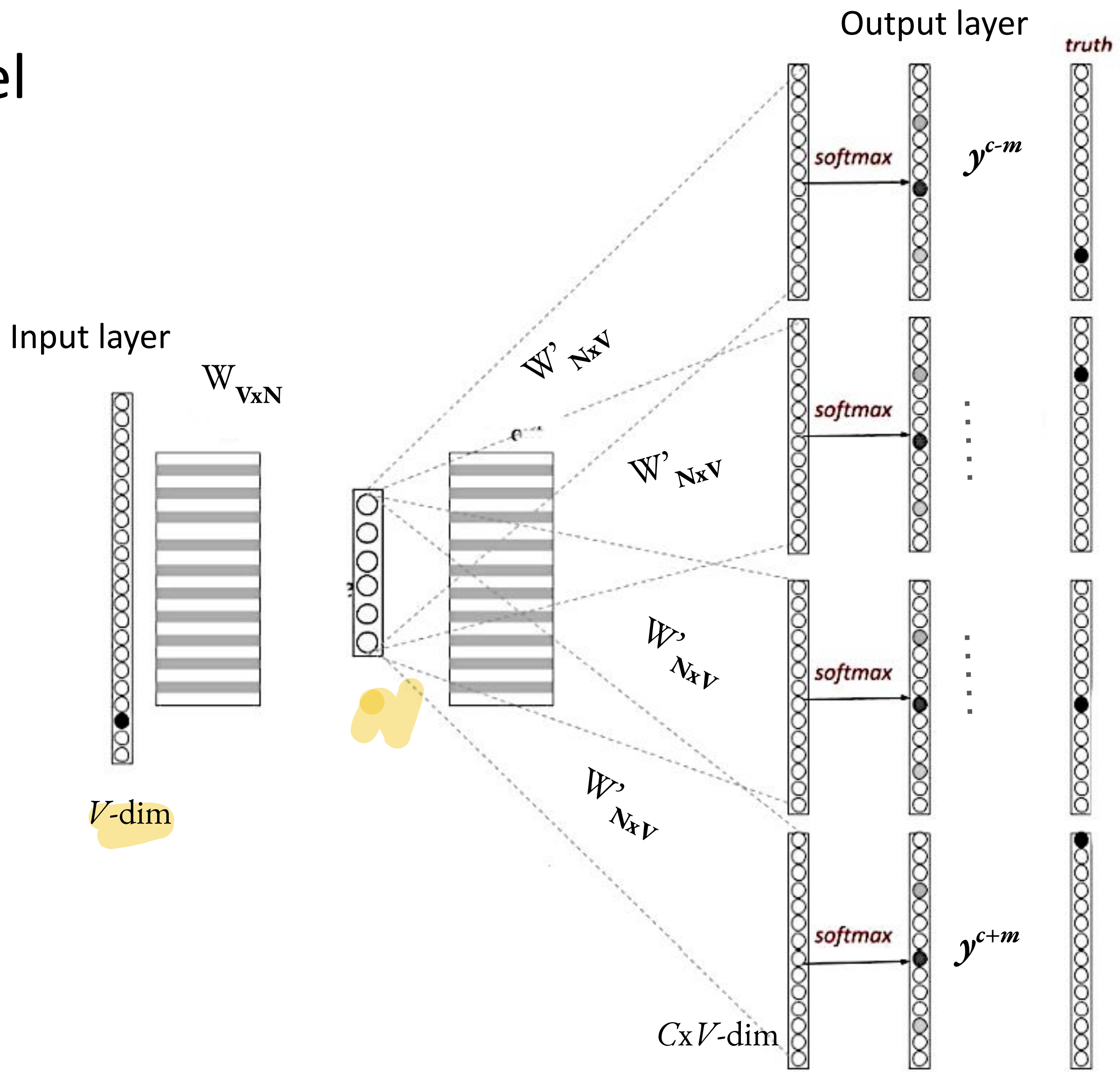
swing	sentence, should, the, sword

Skip-gram



The skip-gram model. Both the input vector x and the output y are one-hot encoded word representations. The hidden layer is the word embedding of size N.

Skip-gram Model



FastText

$s(w, c)$

context

$p(w_t | w_c) = \frac{e^{s(w_c, w_t)}}{\sum}$

$\sum_{g} z_g v_c$

FastText

Represents a word by sum of its n-grams.

- It enhances the Word2Vec algorithm
- Limitation of Word2Vec:
  - Out of Vocabulary(OOV) Words
    - Tensor flow TensorFlow
  - Morphology
    - go, goes, going, gone
- FastText overcome these limitations
- FastText allows to compute word representation for words not in training data

FastText and n-grams

→ represent words as a sum of its character n-grams.

- FastText is the modification to the skip-gram method Sub-word generation using generation of character n-grams of length 3 to 6.
- E.g., *where* with n=3: <wh, whe, her, ere, re>, <where>
- Represent words as sum of its character n-grams
- Grammatical variations still share most of n-grams.
- Compound nouns are easy to model, e.g., Noun + Noun: lunchtime

goes

↓

<go, goe, oes, es>

FastText

- Let us consider that we are given a scoring function $s(w, c)$ which maps pairs of (word, context) to scores.

FastText

- Let us consider that we are given a scoring function $s(w, c)$ which maps pairs of (word, context) to scores.
- Suppose that you are given a dictionary of n grams of size G.

FastText

- Let us consider that we are given a scoring function $s(w, c)$ which maps pairs of (word, context) to scores.
- Suppose that you are given a dictionary of n grams of size G.
- Given a word w, let us denote by $\mathcal{G}_w \subset \{1, \ldots, G\}$ the set of n-grams appearing in w.

FastText

- Let us consider that we are given a scoring function $s(w, c)$ which maps pairs of (word, context) to scores.
- Suppose that you are given a dictionary of n grams of size G.
- Given a word w, let us denote by $\mathcal{G}_w \subset \{1, \ldots, G\}$ the set of n-grams appearing in w.
- We associate a vector representation $\mathbf{z}_g$ to each n-gram g.

FastText

- Let us consider that we are given a scoring function $s(w, c)$ which maps pairs of (word, context) to scores.
- Suppose that you are given a dictionary of n grams of size G.
- Given a word w, let us denote by $\mathcal{G}_w \subset \{1, \ldots, G\}$ the set of n-grams appearing in w.
- We associate a vector representation $\mathbf{z}_g$ to each n-gram g.
- We represent a word by the sum of the vector representations of its n-grams. We thus obtain the scoring function:

$$s(w, c) = \sum_{g \in \mathcal{G}_w} \mathbf{z}_g^\top \mathbf{v}_c. \tag{1}$$

FastText

- Let us consider that we are given a scoring function $s(w, c)$ which maps pairs of (word, context) to scores.
- Suppose that you are given a dictionary of n grams of size G.
- Given a word w, let us denote by $\mathcal{G}_w \subset \{1, \ldots, G\}$ the set of n-grams appearing in w.
- We associate a vector representation $\mathbf{z}_g$ to each n-gram g.
- We represent a word by the sum of the vector representations of its n-grams. We thus obtain the scoring function:

$$s(w, c) = \sum_{g \in \mathcal{G}_w} \mathbf{z}_g^\top \mathbf{v}_c. \tag{1}$$

- This simple model allows sharing the representations across words, thus allowing to learn reliable representation for rare words.

FastText

The scoring function is described as:

$$s(w, c) = \sum_{g \in \mathcal{G}_w} \mathbf{z}_g^\top \mathbf{v}_c .$$

Now, the final model (similar to skip-gram) is given by:

target

$$p(w_c \mid w_t) = \frac{e^{s(w_t,\, w_c)}}{\sum_{j=1}^{V} e^{s(w_t,\, j)}}$$

context

References

- Deep Learning: A Visual Approach Book by Andrew Glassner.
- Bolukbasi, Tolga, et al. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." Advances in neural information processing systems (2016).
- Practical Natural Language Processing: A Comprehensive Guide to Building Real-World NLP Systems by Anuj Gupta.
- Bojanowski, Piotr, Edouard Grave, Armand Joulin, and Tomas Mikolov. "Enriching word vectors with subword information." Transactions of the association for computational linguistics 5 (2017): 135-146.
- https://lilianweng.github.io/posts/2017-10-15-word-embedding/