AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

Large Language Models (LLMs)

Word Embeddings

The word embedding techniques are used to represent words mathematically

Citation Note:

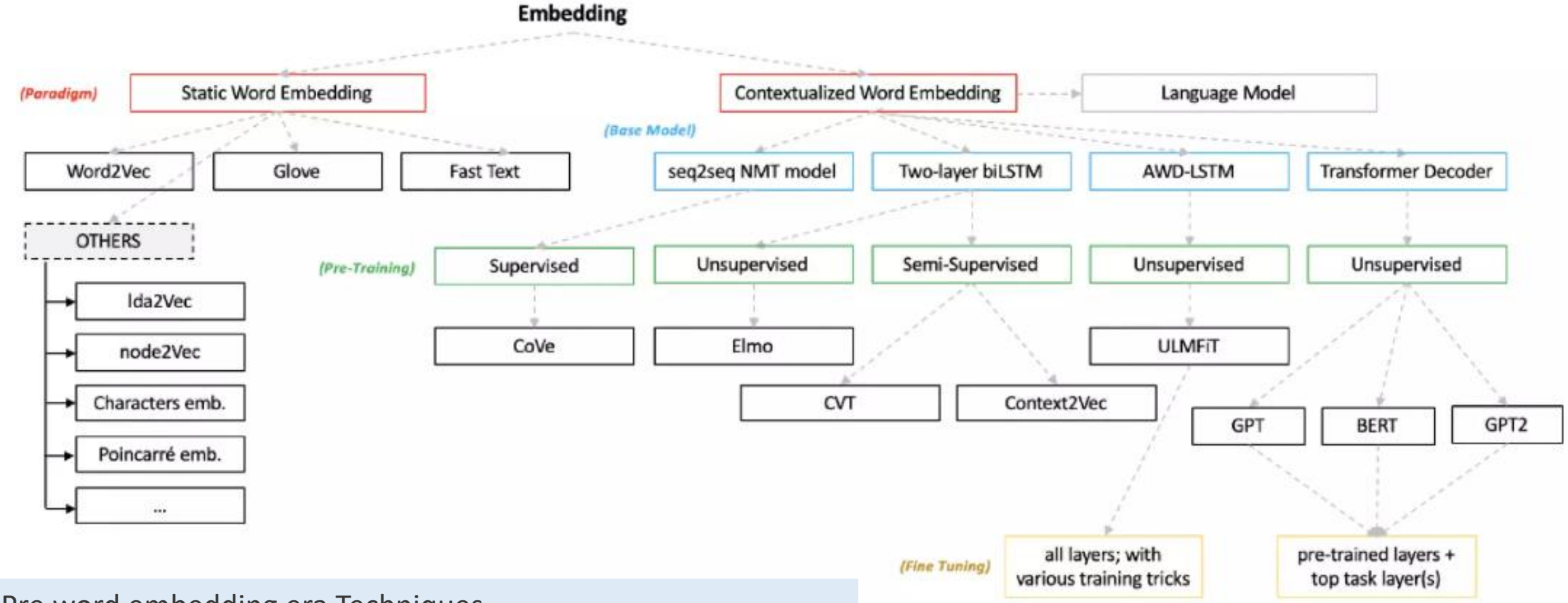
Why to process text as numbers?

- All the ML algorithms require data to be represented as numbers.
- They cannot work on the raw text directly
- Hence, they can only process numeric representation of text data
- Given a text : find a scheme to represent it mathematically (vectors or matrices)
- This is called as text representation or feature engineering.

Techniques to representing text

- Character encoding
- Basic Vectorization
  - 1-0 encoding
  - BoWs
  - Tf-idf
  - PMI
- Advanced Vectorization
  - Embeddings – distribution-based representation
    - Word embeddings – GloVe, Word2Vec, FastText
    - Contextual embeddings – ELMo, BERT,

Word Embeddings



Pre word embedding era Techniques

- One-hot Encoding (OHE)
- Count Vectorizer
- Bag-of-Words (BOW)
- N-grams
- Term Frequency-Inverse Document Frequency (TF-IDF)

Familiarization of Terms

Document
A document is a single text data point. **For Example,** a review of a particular product by the user.

Corpus
It a collection of all the documents present in our dataset.

Feature
Every unique word in the corpus is considered as a feature.

For Example, Let's consider the 2 documents shown below:
Sentences:
Dog hates a cat. It loves to go out and play.
Cat loves to play with a ball.
We can build a corpus from the above 2 documents just by combining them.

Corpus = "Dog hates a cat. It loves to go out and play. Cat loves to play with a ball."
And features will be all unique words:

Features: ['and', 'ball', 'cat', 'dog', 'go', 'hates', 'it', 'loves', 'out', 'play', 'to', 'with']

Why do we need Word Embeddings?

Word Embeddings are the texts converted into numbers and there may be different numerical representations of the same text

To build any model in machine learning or deep learning, the final level data has to be in numerical form because models don't understand text or image data directly as humans do.

To convert the text data into numerical data, we need some smart ways which are known as **vectorization,** or in the NLP world, it is known as **Word embeddings.**

Vectorization or word embedding is the process of converting text data to numerical vectors. Later those vectors are used to build various machine learning models. In this manner, we say this as **extracting features** with the help of text with an aim to build ML/DL models.

Broadly, we can classify word embeddings into the following two categories:

- Frequency-based or Statistical based Word Embedding
- Prediction based Word Embedding

One-Hot Encoding (OHE)

In this technique, we represent each unique word in vocabulary by setting a unique token with value 1 and rest 0 at other positions in the vector. In simple words, a vector representation of a one-hot encoded vector represents in the form of 1, and 0 where 1 stands for the position where the word exists and 0 everywhere else.

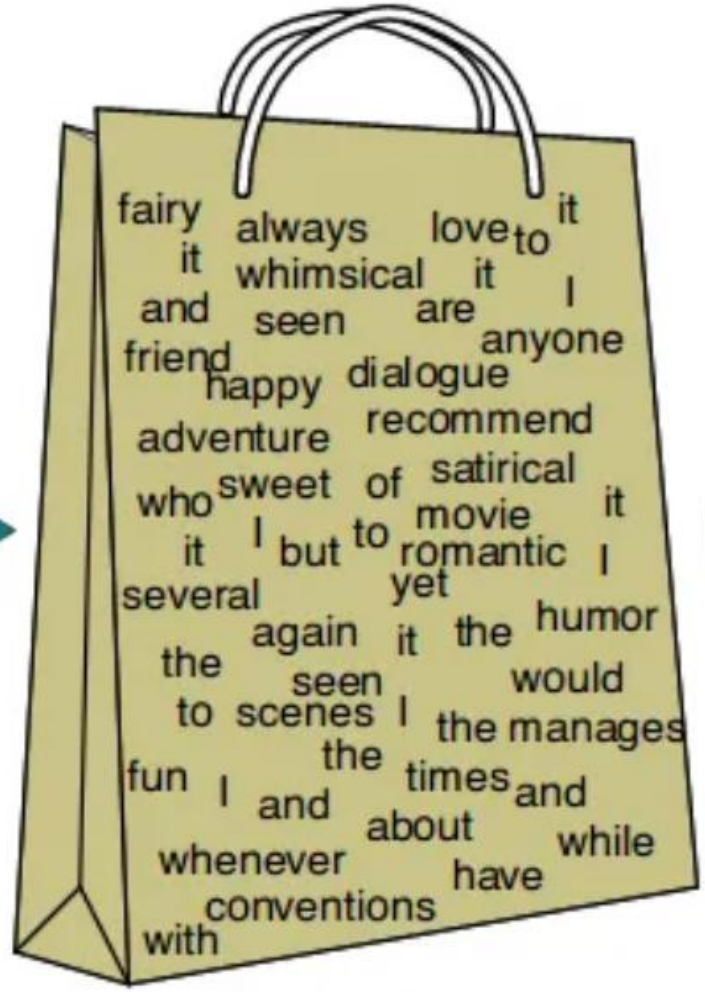
Let's consider the following sentence:
Sentence: I am teaching NLP in Python
A word in this sentence may be "NLP", "Python", "teaching", etc.
Since a dictionary is defined as the list of all unique words present in the sentence. So, a dictionary may look like –
Dictionary: ['I', 'am', 'teaching',' NLP',' in', 'Python']
Therefore, the vector representation in this format according to the above dictionary is

Vector for NLP: [0,0,0,1,0,0] Vector for Python: [0,0,0,0,0,1]
This is just a very simple method to represent a word in vector form.

Disadvantages of One-hot Encoding
1. One of the disadvantages of One-hot encoding is that the Size of the vector is equal to the count of unique words in the vocabulary.
2. One-hot encoding does not capture the relationships between different words. Therefore, it does not convey information about the context.

Bag of Words

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
...	...

https://medium.com/@manutej/the-secret-sauce-to-generative-ai-text-capabilities-nlp-word-embeddings-and-language-models-30284ecb8e55

Count Vectorizer/Bag of Words

Count vectorizer will fit and learn the word vocabulary and try to create a document term matrix in which the individual cells denote the frequency of that word in a particular document, which is also known as term frequency, and the columns are dedicated to each word in the corpus.

Document-1: He is a smart boy. She is also smart. Document-2: Chirag is a smart person.

The dictionary created contains the list of unique tokens(words) present in the corpus

Unique Words: [‘He’, ’She’, ’smart’, ’boy’, ’Chirag’, ’person’]

Here, D=2, N=6

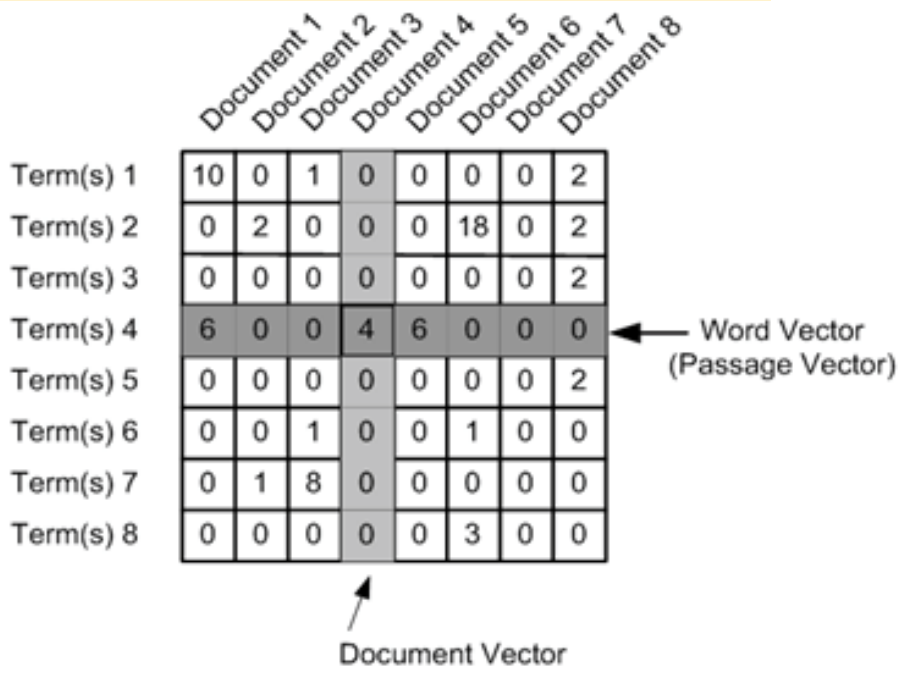
So, the count matrix M of size 2 X 6 will be represented as –

So, the count matrix M of size 2 X 6 will be represented as –

	He	She	smart	boy	Chirag	person
D1	1	1	2	1	0	0
D2	0	0	1	0	1	1

Vector for ‘smart’ is [2,1], Vector for ‘Chirag’ is [0, 1], and so on.

- The rows indicate the documents in the corpus and
- The columns indicate the tokens in the dictionary.



Disadvantages of Bag of Words

1. This method doesn’t preserve the word order.

2. It does not allow to draw of useful inferences for downstream NLP tasks.

N-grams Vectorization

N-grams Vectorization

1. Similar to the count vectorization technique, in the N-Gram method, a document term matrix is generated, and each cell represents the count.

2. The columns represent all columns of adjacent words of length n.

3. Count vectorization is a special case of N-Gram where n=1.

4. N-grams consider the sequence of n words in the text; where n is (1,2,3..) like 1-gram, 2-gram. for token pair. Unlike BOW, it maintains word order.

Disadvantages of N-Grams

1. It has too many features.

2. Due to too many features, the feature set becomes too sparse and is computationally expensive.

3. Choose the optimal value of N is not that easy task.

For Example,

“I am studying NLP” has four words and n=4.

if n=2, i.e bigram, then the columns would be — [“I am”, “am reading”, ‘studying NLP”]

if n=3, i.e trigram, then the columns would be — [“I am studying”, ”am studying NLP”]

if n=4,i.e four-gram, then the column would be -[‘“I am studying NLP”]

How does the value of N in the N-grams method create a tradeoff?

The trade-off is created while choosing the N values. If we choose N as a smaller number, then it may not be sufficient enough to provide the most useful information. But on the contrary, if we choose N as a high value, then it will yield a huge matrix with lots of features. Therefore, N-gram may be a powerful technique, but it needs a little more care

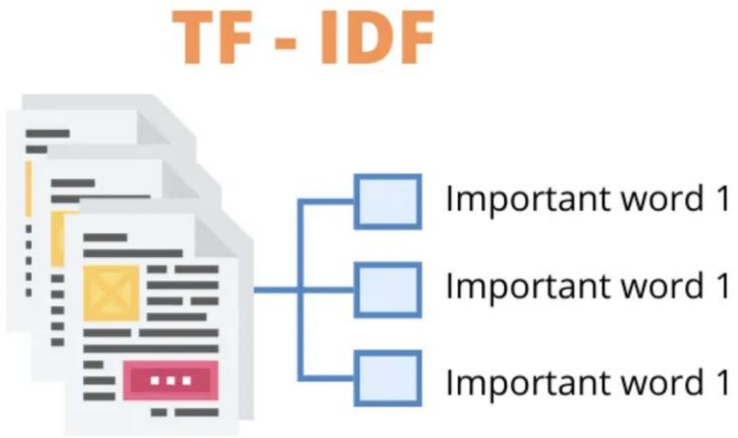
TF-IDF Vectorization

BOW treats all words equally As a result, it cannot distinguish very common words or rare words

Term frequency-inverse document frequency (TF-IDF) gives a measure that takes the importance of a word into consideration depending on how frequently it occurs in a document and a corpus.

Term Frequency
It is the percentage of the number of times a word (x) occurs in a particular document (y) divided by the total number of words in that document.

$$TF(\boldsymbol{term}) = \frac{Number\ of\ times\ \boldsymbol{term}\ appears\ in\ a\ document}{Total\ number\ of\ items\ in\ the\ document}$$



The formula for finding Term Frequency is given as:
tf ('word') = Frequency of a 'word' appears in document d / total number of words in the document d

For Example, Consider the following document
Document: Cat loves to play with a ball
For the above sentence, the term frequency value for word cat will be: **tf('cat') = 1 / 6**
Note: Sentence "Cat loves to play with a ball" has 7 total words but the word 'a' has been ignored.

Ensures that frequently occurring but unimportant words (like stopwords) get down-weighted, while rare but relevant terms receive higher weights.

Inverse Document Frequency

It measures the importance of the word in the corpus. It measures how common a particular word is across all the documents in the corpus.
It is the logarithmic ratio of no. of total documents to no. of a document with a particular word.

$$IDF(\textbf{term}) = \log\left(\frac{Total\ number\ of\ documents}{Number\ of\ documents\ with\ \textbf{term}\ in\ it}\right)$$

$$TFIDF(\textbf{term}) = TF(\textbf{term}) * \text{IDF}(\textbf{term})$$

TF-IDF assigns higher weights to terms that are frequent within a document but rare across the entire corpus.

This term TF-IDF of the equation helps in pulling out the rare words since the value of the product is maximum when both the terms are maximum. What does that mean? If a word appears multiple times across many documents, then the denominator df will increase, reducing the value of the second term.

The formula for finding Inverse Document Frequency is given as:
idf('word') = log(Total number of documents / number of document with 'word' in it)

The formula for finding TF-IDF is given as:

$$W_{x,y} = tf_{x,y} * log\left(\frac{N}{df_x}\right)$$

$W_{x,y}$= Word x within document y

$tf_{x,y}$= frequency of x in y

df_x= number of documents containing x

N=total number of documents

example

For Example, In any corpus**,** few words like **‘is’** or **‘and’** are very common, and most likely, they will be present in almost every document.

Let’s say the word ‘is’ is present in all the documents is a corpus of 1000 documents. The idf for that would be:

The idf(‘is’) is equal to log (1000/1000) = log 1 = 0

Thus common words would have lesser importance.

In the same way, the idf(‘cat’) will be equal to 0 in our example.
The tf-idf is equal to the product of tf and idf values for that word:
Therefore, tf-idf(‘cat’ for document 2) = tf(‘cat’) * idf(‘cat’) = 1 / 6 * 0 = 0 Thus, tf-idf(‘cat’) for document 2 would be 0

Why IDF?

- Some words (e.g., "the", "is", "and") appear in almost every document, making them less informative.
- IDF penalizes common words and boosts rare but important words.
- If a word appears in many documents, IDF is low, meaning it's less significant.
- If a word is rare, its IDF is high, making it more important.

Important Points about TF-IDF Vectorizer

1. Similar to the count vectorization method, in the TF-IDF method, a document term matrix is generated and each column represents an individual unique word.

2. The difference in the TF-IDF method is that each cell doesn't indicate the term frequency, but contains a weight value that signifies how important a word is for an individual text message or document

3. This method is based on the frequency method but it is different from the count vectorization in the sense that it takes into considerations not just the occurrence of a word in a single document but in the entire corpus.

4. TF-IDF gives more weight to less frequently occurring events and less weight to expected events. So, it penalizes frequently occurring words that appear frequently in a document such as **"the", "is"** but assigns greater weight to less frequent or rare words.

5. The product of TF x IDF of a word indicates how often the token is found in the document and how unique the token is to the whole entire corpus of documents.

6. Same vector for same words irrespective of context
7. Not word order sensitive
8.Dimensionality is vocabulary dependent
9. No context awareness and no word similarity considered

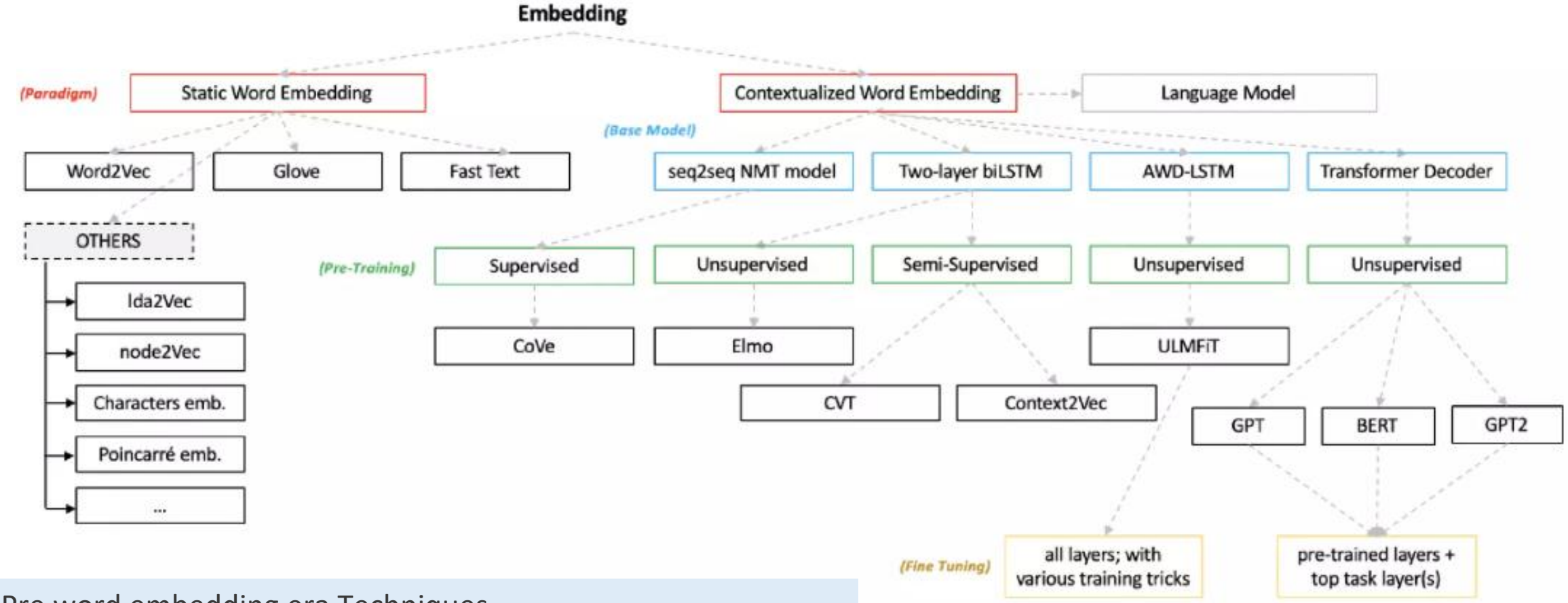
Example

Documents	Text	Total number of words in a document
A	Jupiter is the largest planet	5
B	Mars is the fourth planet from the sun	8

Words	TF (for A)	TF (for B)	IDF	TFIDF (A)	TFIDF (B)
Jupiter	1/5	0	ln(2/1) = 0.69	0.138	0
Is	1/5	1/8	ln(2/2) = 0	0	0
The	1/5	2/8	ln(2/2) = 0	0	0
largest	1/5	0	ln(2/1) = 0.69	0.138	0

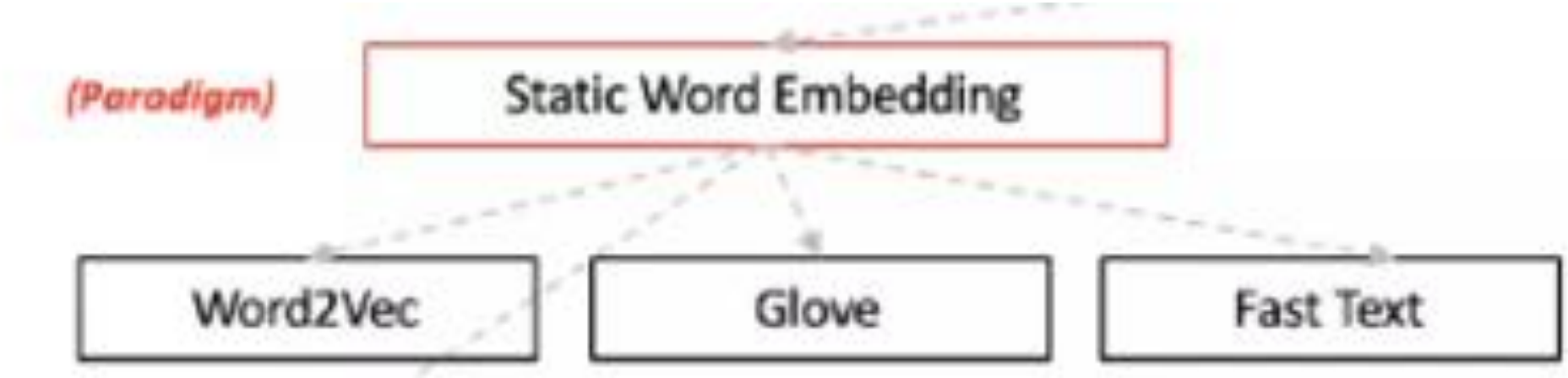
Feature	TF-IDF	Bag of Words
Captures word importance	Yes	No
Handles common words	Yes	No
Considers word order	No	No
Easy to understand and use	Yes	Yes
Handles large datasets	Yes	Yes

Word Embeddings



Pre word embedding era Techniques

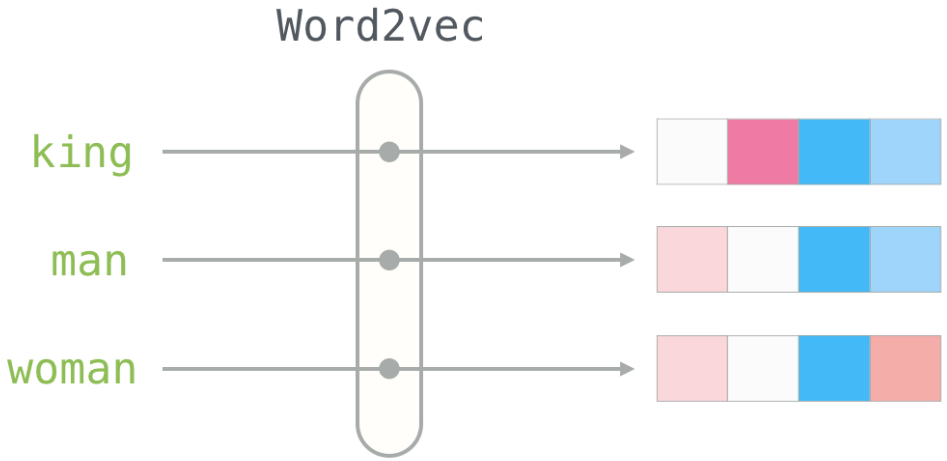
- One-hot Encoding (OHE)
- Count Vectorizer
- Bag-of-Words (BOW)
- N-grams
- Term Frequency-Inverse Document Frequency (TF-IDF)


(Paradigm)
Static Word Embedding
Word2Vec
Glove
Fast Text

Word2vec

Word2vec is a technique in natural language processing (NLP) for obtaining vector representations of words. These vectors capture information about the meaning of the word and their usage in context.

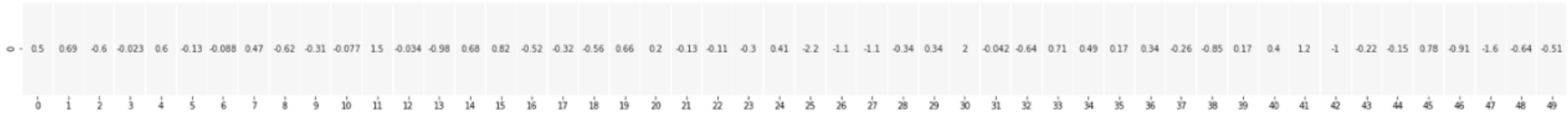
Word2Vec creates vectors of the words that are distributed numerical representations of word features – these word features could comprise of words that represent the context of the individual words present in our vocabulary.



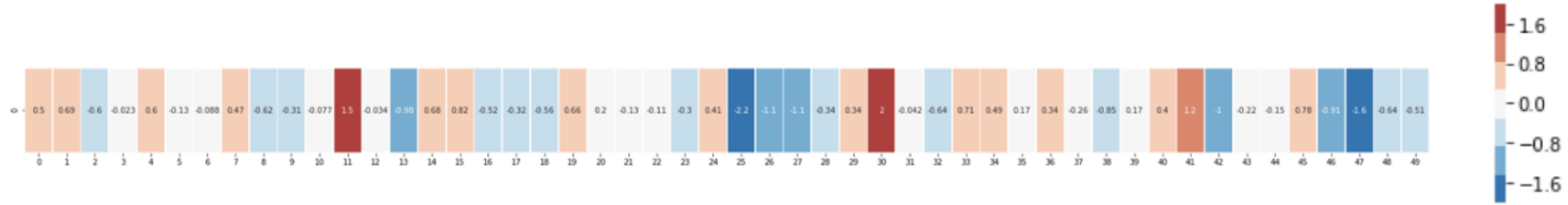
This is a word embedding for the word “king” (GloVe vector trained on Wikipedia):

```
[ 0.50451 , 0.68607 , -0.59517 , -0.022801, 0.60046 , -0.13498 , -0.08813 , 0.47377 , -0.61798 , -0.31012 ,
-0.076666, 1.493 , -0.034189, -0.98173 , 0.68229 , 0.81722 , -0.51874 , -0.31503 , -0.55809 , 0.66421 , 0.1961
, -0.13495 , -0.11476 , -0.30344 , 0.41177 , -2.223 , -1.0756 , -1.0783 , -0.34354 , 0.33505 , 1.9927 ,
-0.04234 , -0.64319 , 0.71125 , 0.49159 , 0.16754 , 0.34344 , -0.25663 , -0.8523 , 0.1661 , 0.40102 , 1.1685 ,
-1.0137 , -0.21585 , -0.15155 , 0.78321 , -0.91241 , -1.6106 , -0.64426 , -0.51042 ]
```

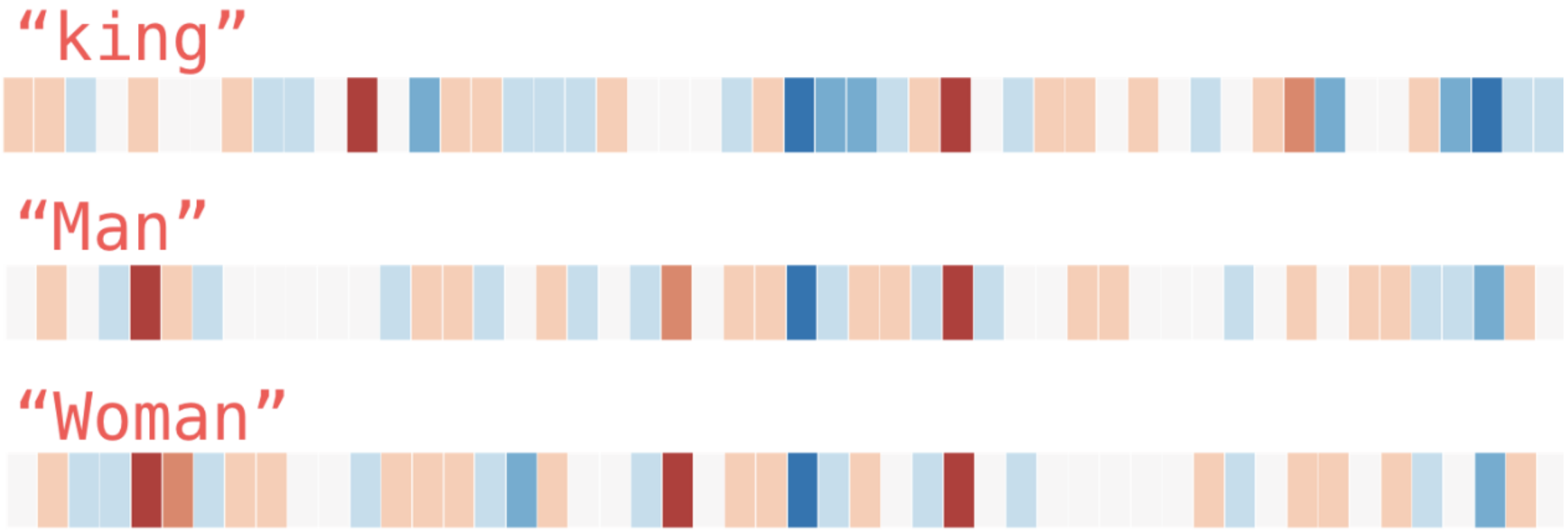
It’s a list of 50 numbers. We can’t tell much by looking at the values. But let’s visualize it a bit so we can compare it other word vectors. Let’s put all these numbers in one row:



Let’s color code the cells based on their values (red if they’re close to 2, white if they’re close to 0, blue if they’re close to -2):

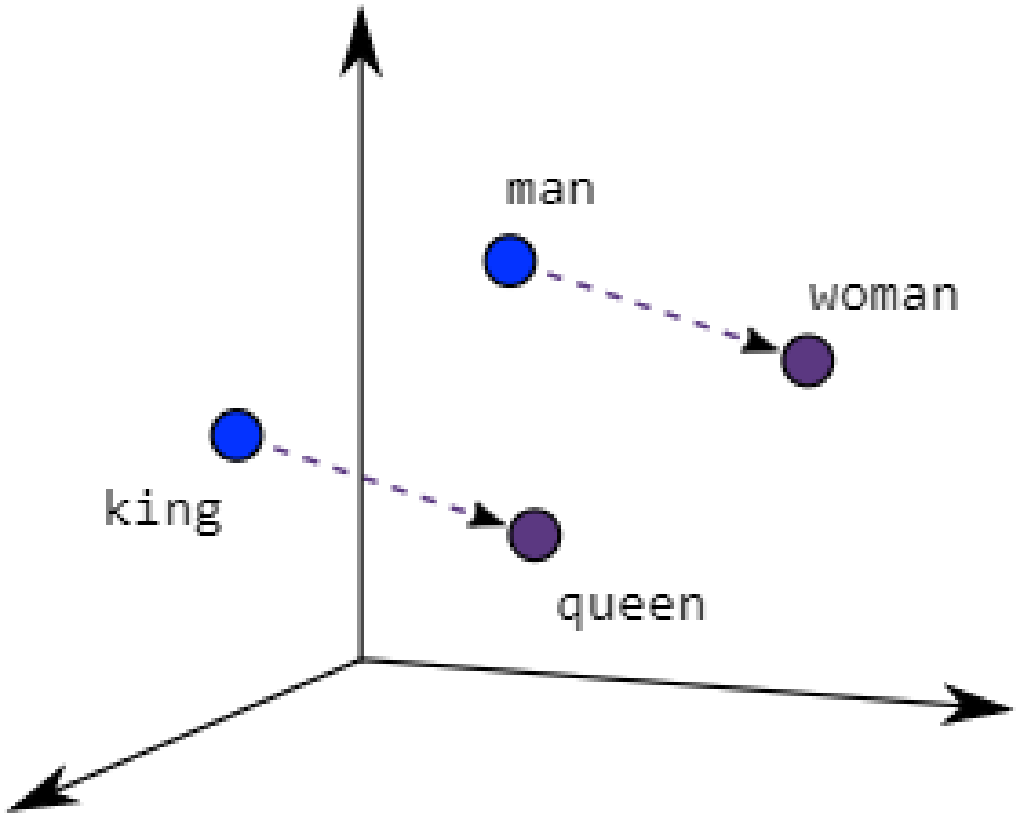


We'll proceed by ignoring the numbers and only looking at the colors to indicate the values of the cells. Let's now contrast "King" against other words:



See how "Man" and "Woman" are much more similar to each other than either of them is to "king"? This tells you something. These vector representations capture quite a bit of the information/meaning/associations of these words.

word2vec



Two different model architectures that can be used by Word2Vec to create the word embeddings are the Continuous Bag of Words (CBOW) model & the Skip-Gram model.

Word2vec- Word relationships

One of the key advantages of **Word2Vec** is its ability to learn word relationships and analogies through vector arithmetic.

Example: Gender Analogy – "Man is to Woman as King is to ?"

Word2Vec embeddings capture relationships such that:

$$\text{Vector}(\text{"King"}) - \text{Vector}(\text{"Man"}) + \text{Vector}(\text{"Woman"}) \approx \text{Vector}(\text{"Queen"})$$

Man	[0.4, 0.8, 0.1, -0.3, 0.6]
Woman	[0.5, 0.9, 0.2, -0.4, 0.7]
King	[0.6, 0.7, 0.8, 0.1, 0.9]
Queen	[0.7, 0.8, 0.9, 0.0, 1.0]

$$\text{King} - \text{Man} + \text{Woman}$$

$$[0.6, 0.7, 0.8, 0.1, 0.9] - [0.4, 0.8, 0.1, -0.3, 0.6] + [0.5, 0.9, 0.2, -0.4, 0.7]$$

$$= [0.7, 0.8, 0.9, 0.0, 1.0] \approx \text{Vector}(\text{"Queen"})$$

No Word Order Sensitivity

Word2vec :2013

Efficient Estimation of Word Representations in Vector Space

Tomas Mikolov
Google Inc., Mountain View, CA
tmikolov@google.com

Kai Chen
Google Inc., Mountain View, CA
kaichen@google.com

Greg Corrado
Google Inc., Mountain View, CA
gcorrado@google.com

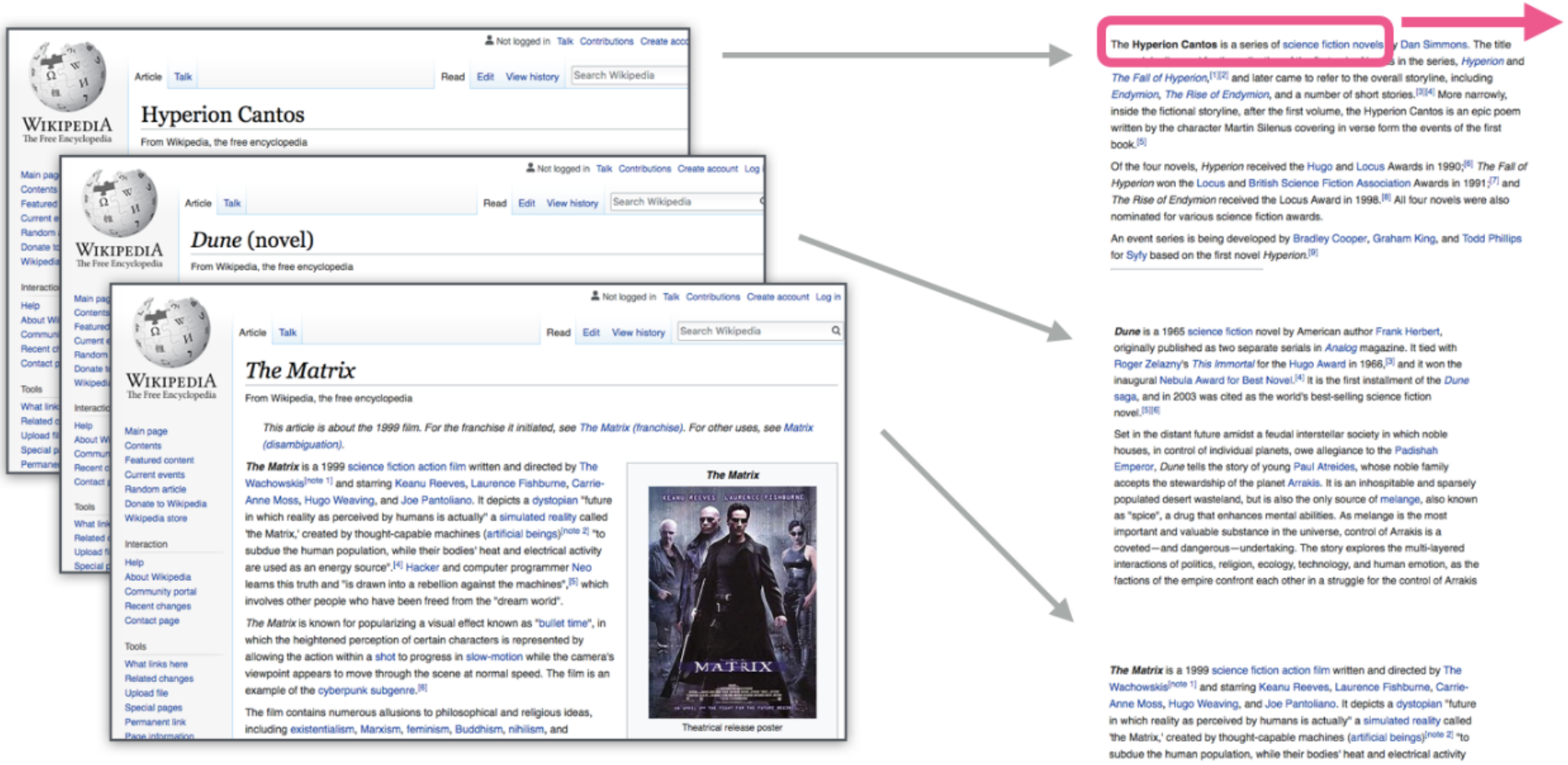
Jeffrey Dean
Google Inc., Mountain View, CA
jeff@google.com

Abstract

We propose two novel model architectures for computing continuous vector representations of words from very large data sets. The quality of these representations is measured in a word similarity task, and the results are compared to the previously best performing techniques based on different types of neural networks. We observe large improvements in accuracy at much lower computational cost, i.e. it takes less than a day to learn high quality word vectors from a 1.6 billion words data set. Furthermore, we show that these vectors provide state-of-the-art performance on our test set for measuring syntactic and semantic word similarities.

Words get their embeddings by us looking at which other words they tend to appear next to. The mechanics of that is that

1. We get a lot of text data (say, all Wikipedia articles, for example). then
2. We have a window (say, of three words) that we slide against all of that text.
3. The sliding window generates training samples for our model



man

0.52
0.76
1.21
0.22
-1.36
0.49
-3.69
-0.07

woman

0.73
0.89
-1.67
1.32
0.36
-1.49
2.71
0.05

The **kid** said he would grow up to be superman.
The **child** said he would grow up to be superman.

- The word2vec objective function causes the words that occur in similar contexts to have similar embeddings.

How word2vec works? - Intutions

Context Word

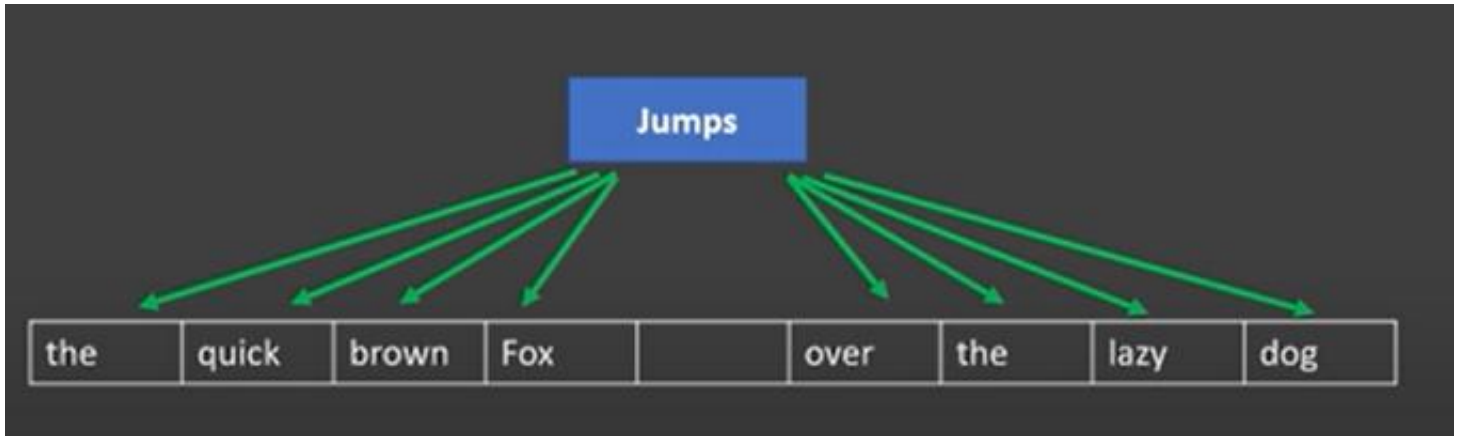
- Learn the context in which a particular word appears
- Words that occurs in similar context have similar embedding

Uses two algorithms:

1. Focus word » Continuous Bag Of Word (CBOW)
2. Context word » Skip- Gram

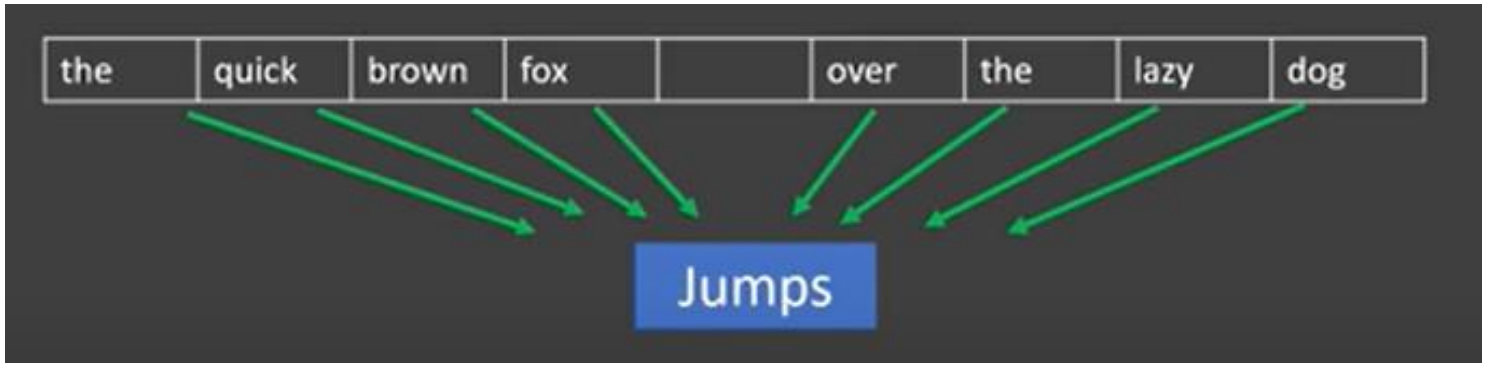
- Predict the target word from the context.

Continuous Bag Of Word (CBOW)

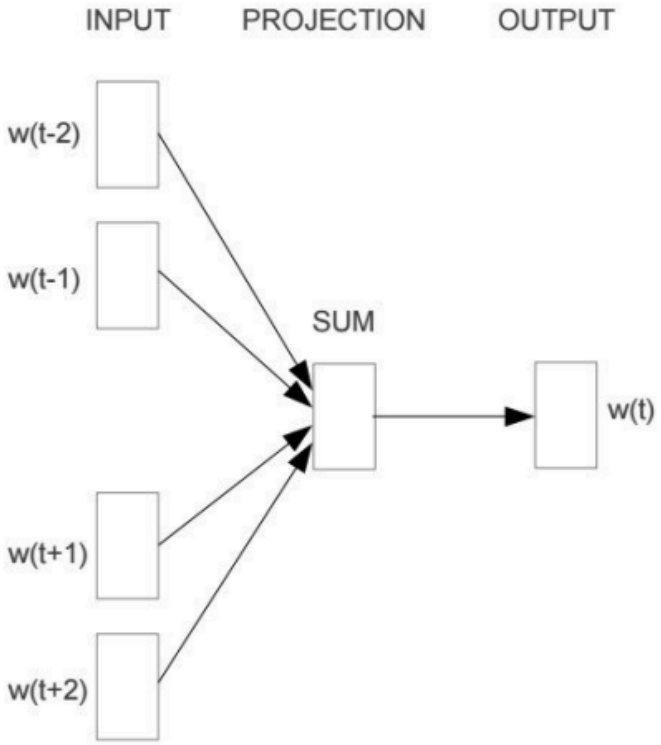


- Predict the context words from target.

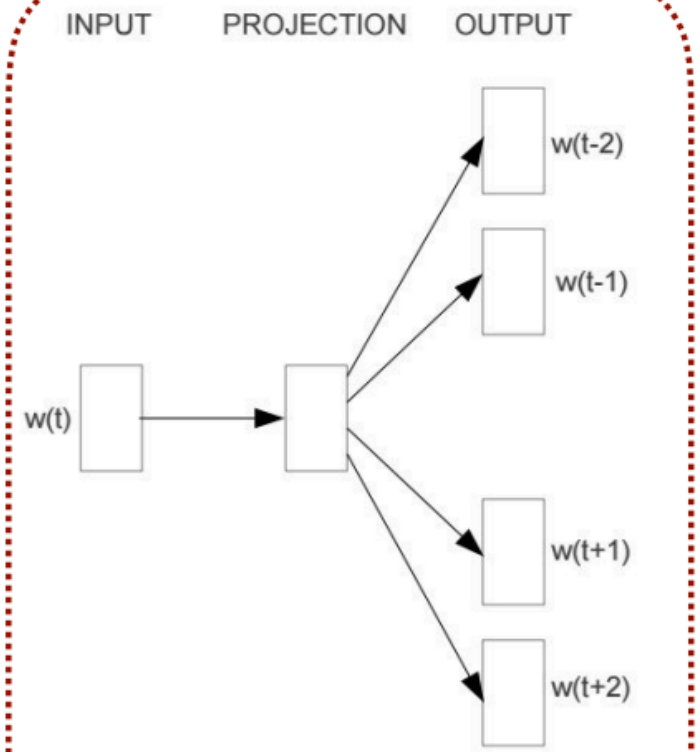
Skip-Gram



If the dataset has **frequent words**, CBOW is preferred. If it contains **rare words**, Skip-Gram is better.

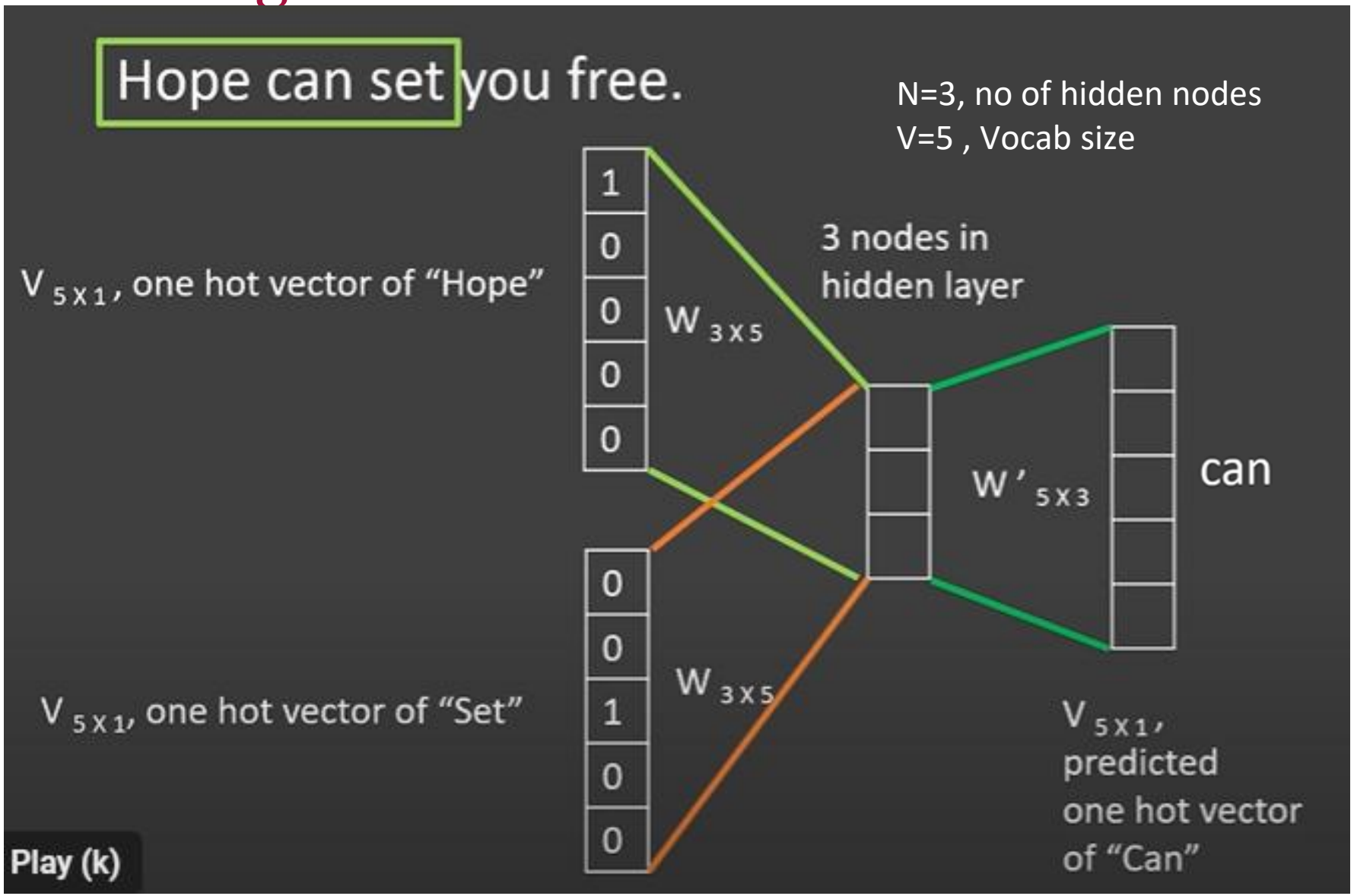


Continuous Bag of Words (CBOW)

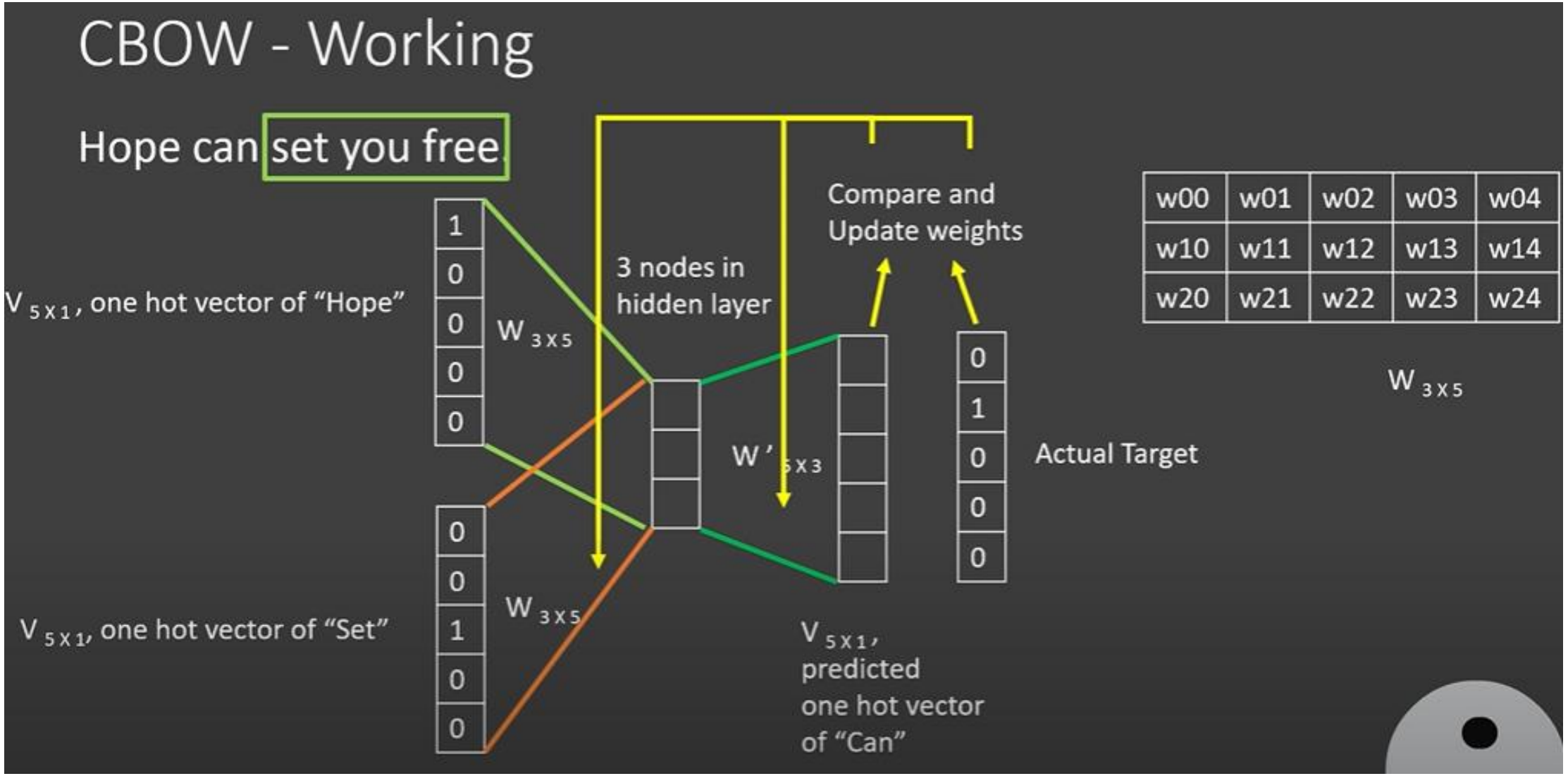


Skip-grams

CBOW Working

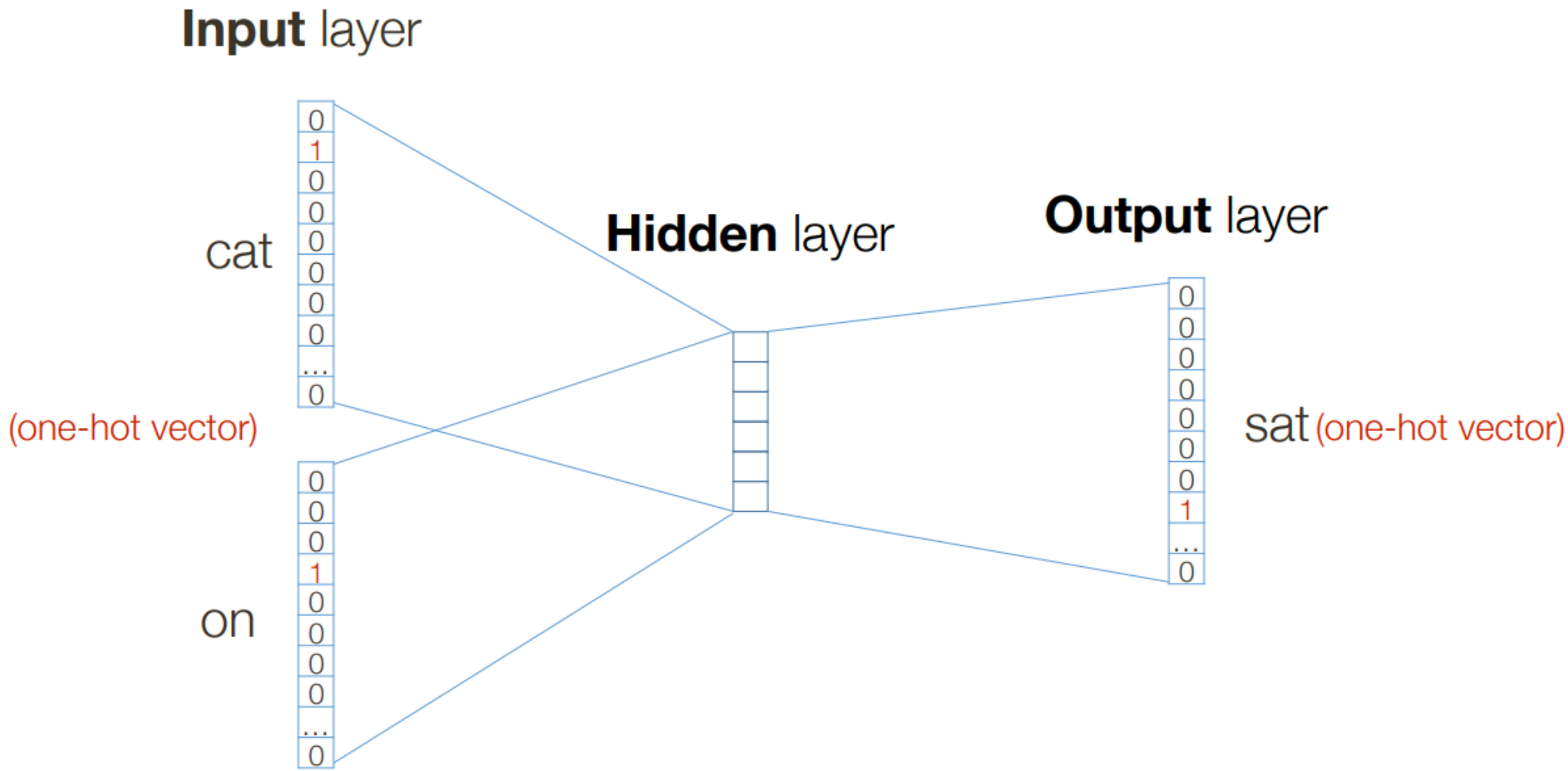


CBOW


CBOW - Working
Hope can set you free
V 5 X 1, one hot vector of "Hope"
1
0
0
0
0
W 3 X 5
3 nodes in hidden layer
W ' 5 X 3
Compare and Update weights
0
1
0
0
0
Actual Target
V 5 X 1, one hot vector of "Set"
0
0
1
0
0
W 3 X 5
V 5 X 1, predicted one hot vector of "Can"
w00 w01 w02 w03 w04
w10 w11 w12 w13 w14
w20 w21 w22 w23 w24
W 3 X 5

CBOW: Continuous Bag of Words

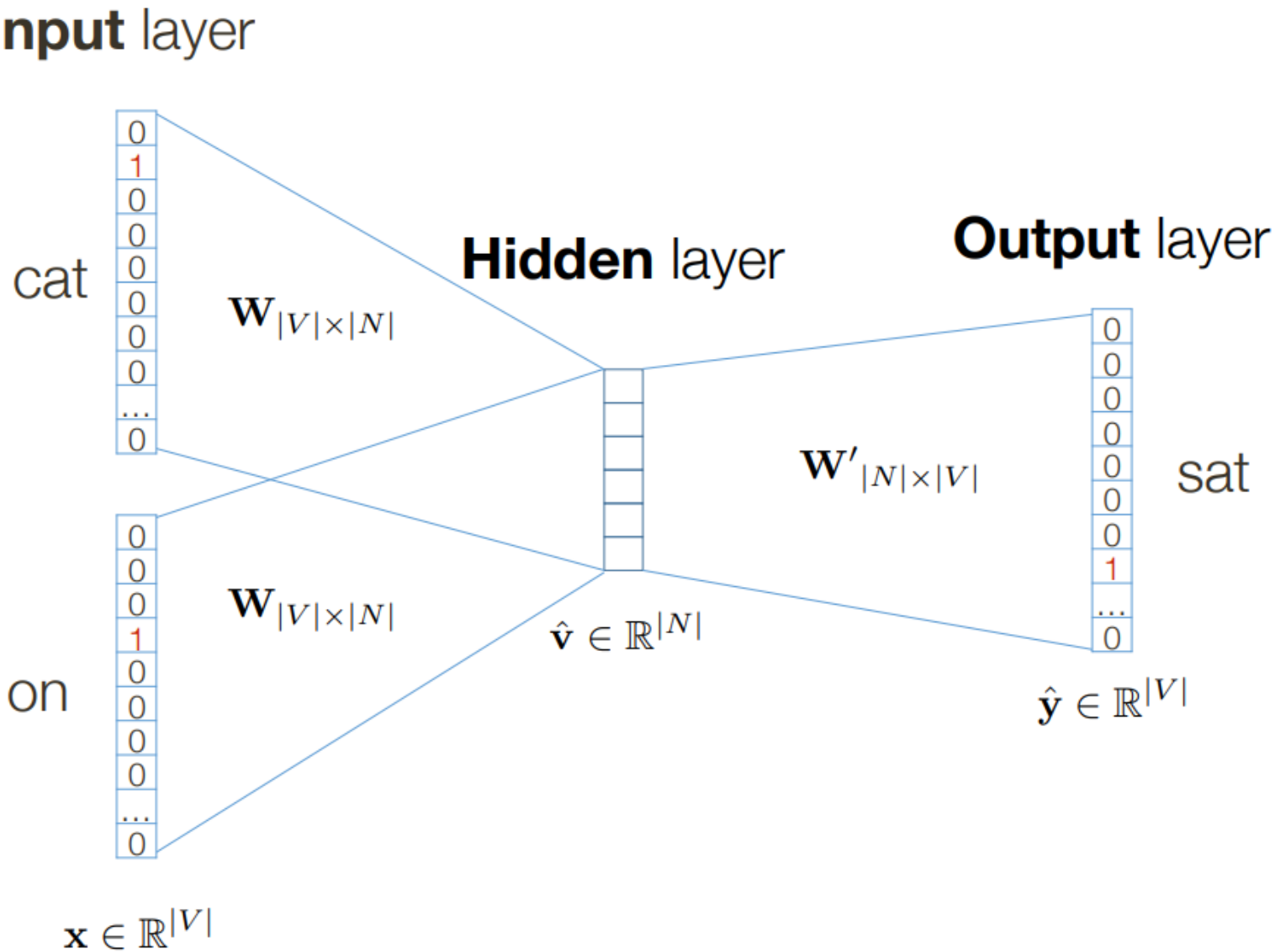
[Mikolov et al., 2013]



CBOW: Continuous Bag of Words

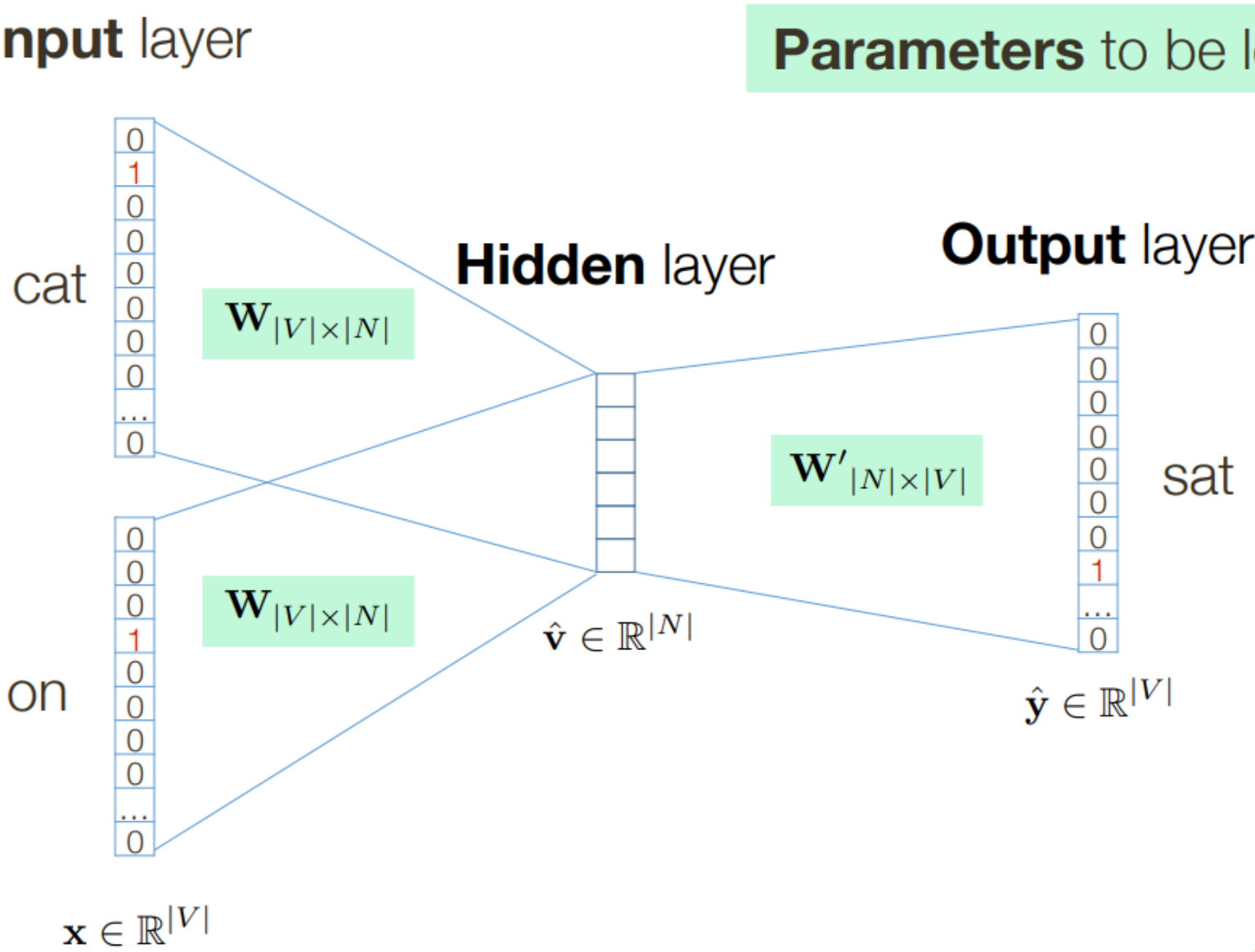
[Mikolov et al., 2013]

|V| is vocabulary size
|N| is size of hidden layer



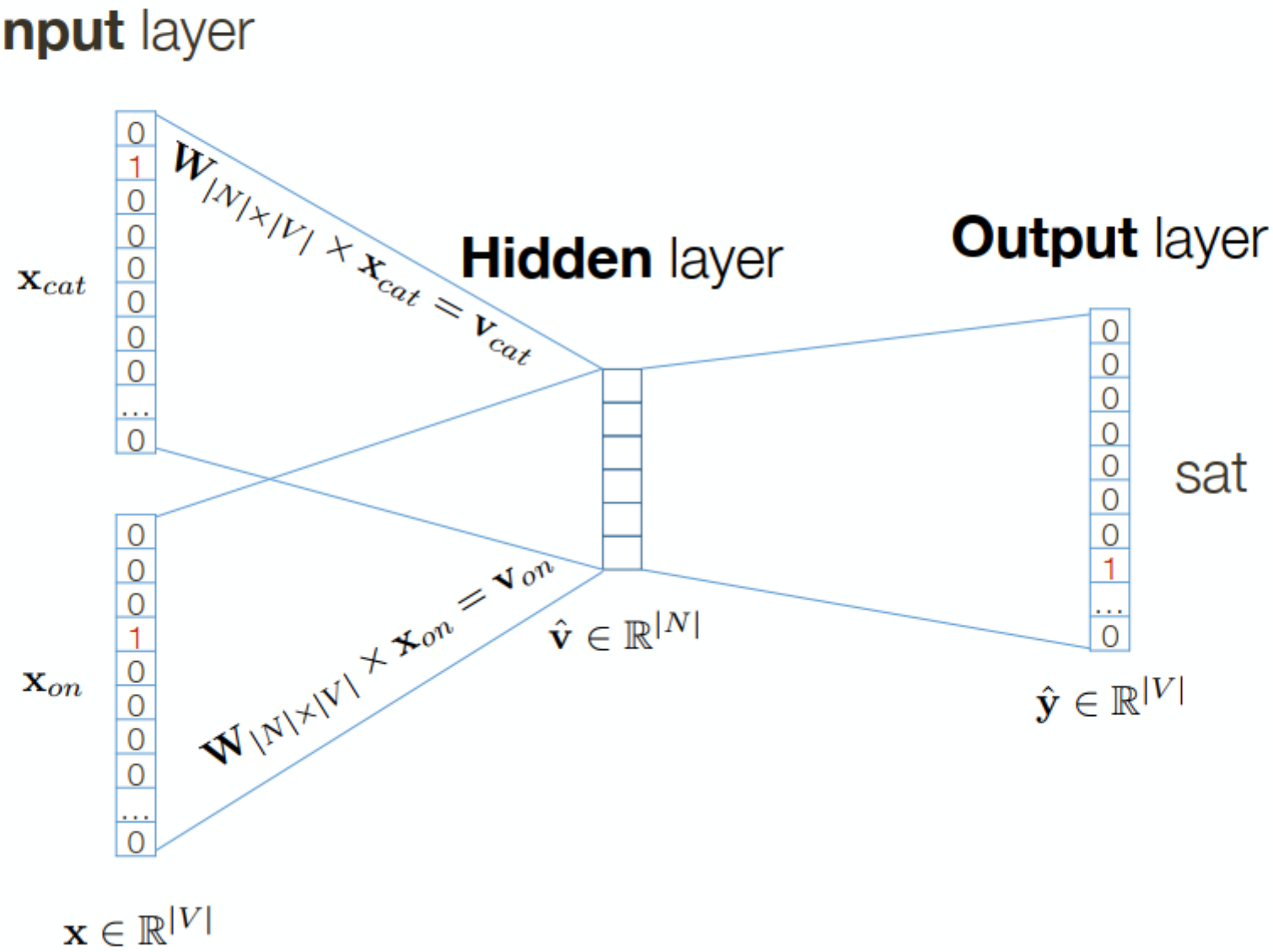
CBOW: Continuous Bag of Words

[Mikolov et al., 2013



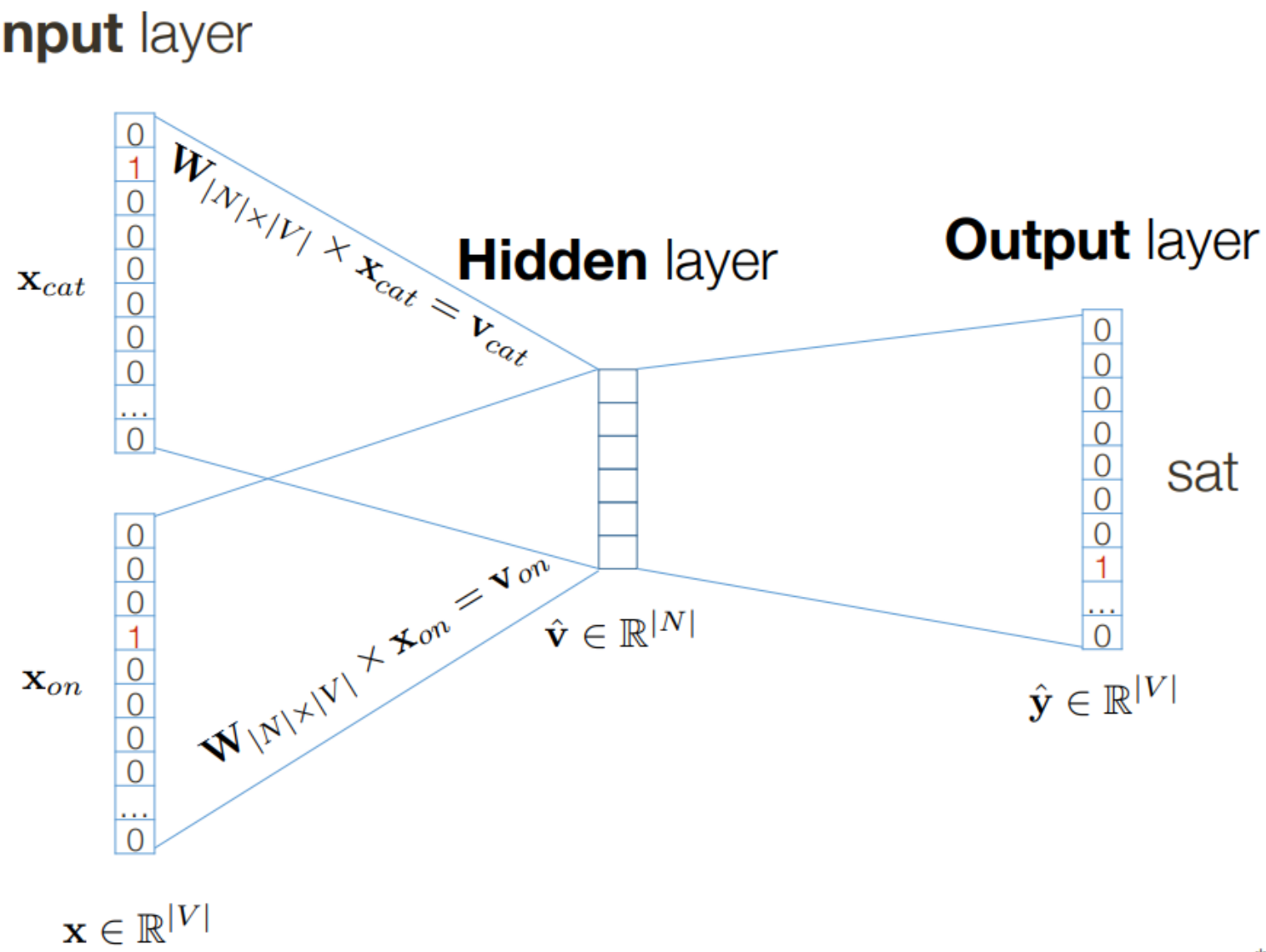
CBOW: Continuous Bag of Words

[Mikolov et al., 2013]



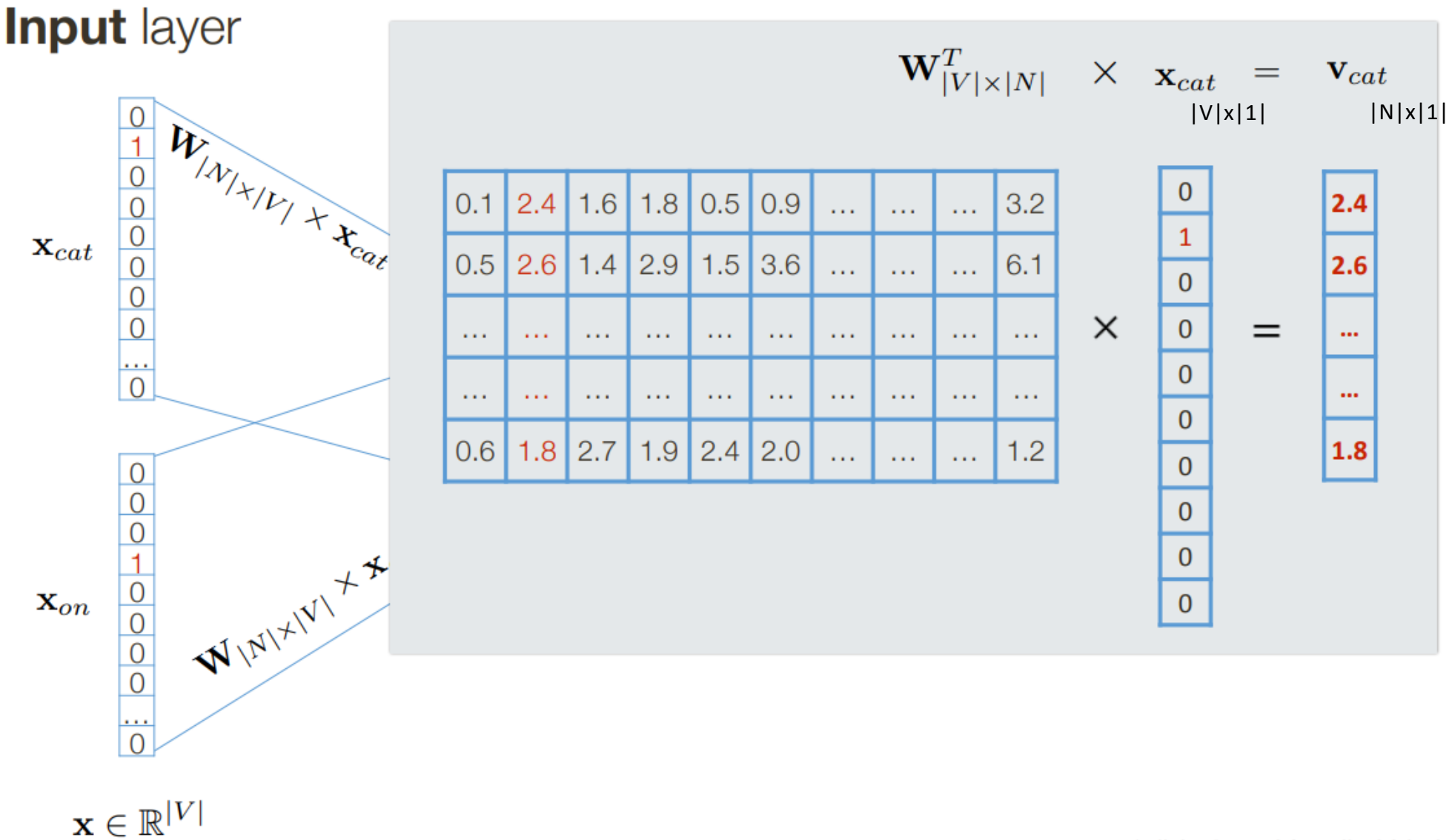
CBOW: Continuous Bag of Words

[Mikolov et al., 2013]



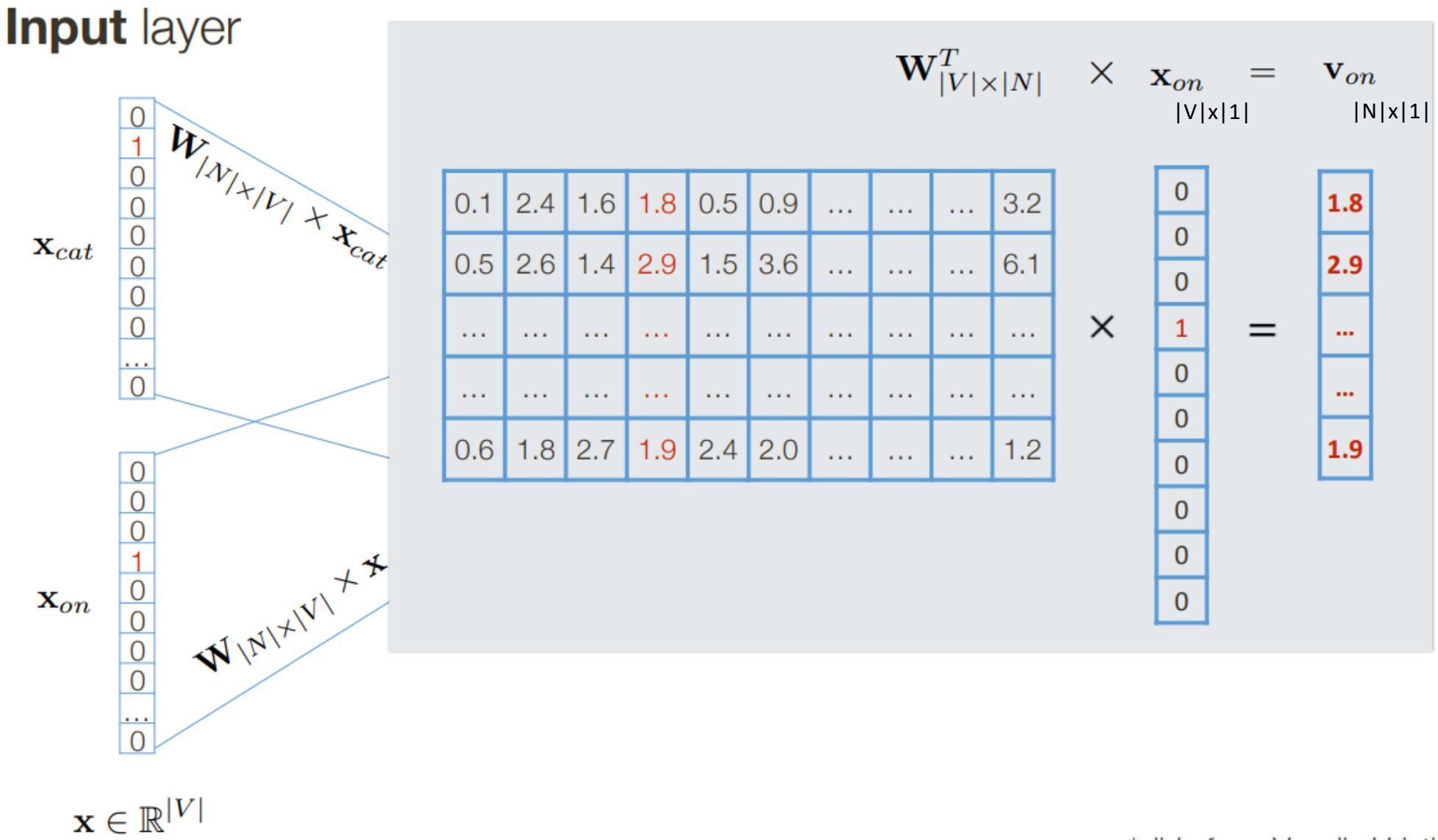
CBOW: Continuous Bag of Words

[Mikolov et al., 2013]



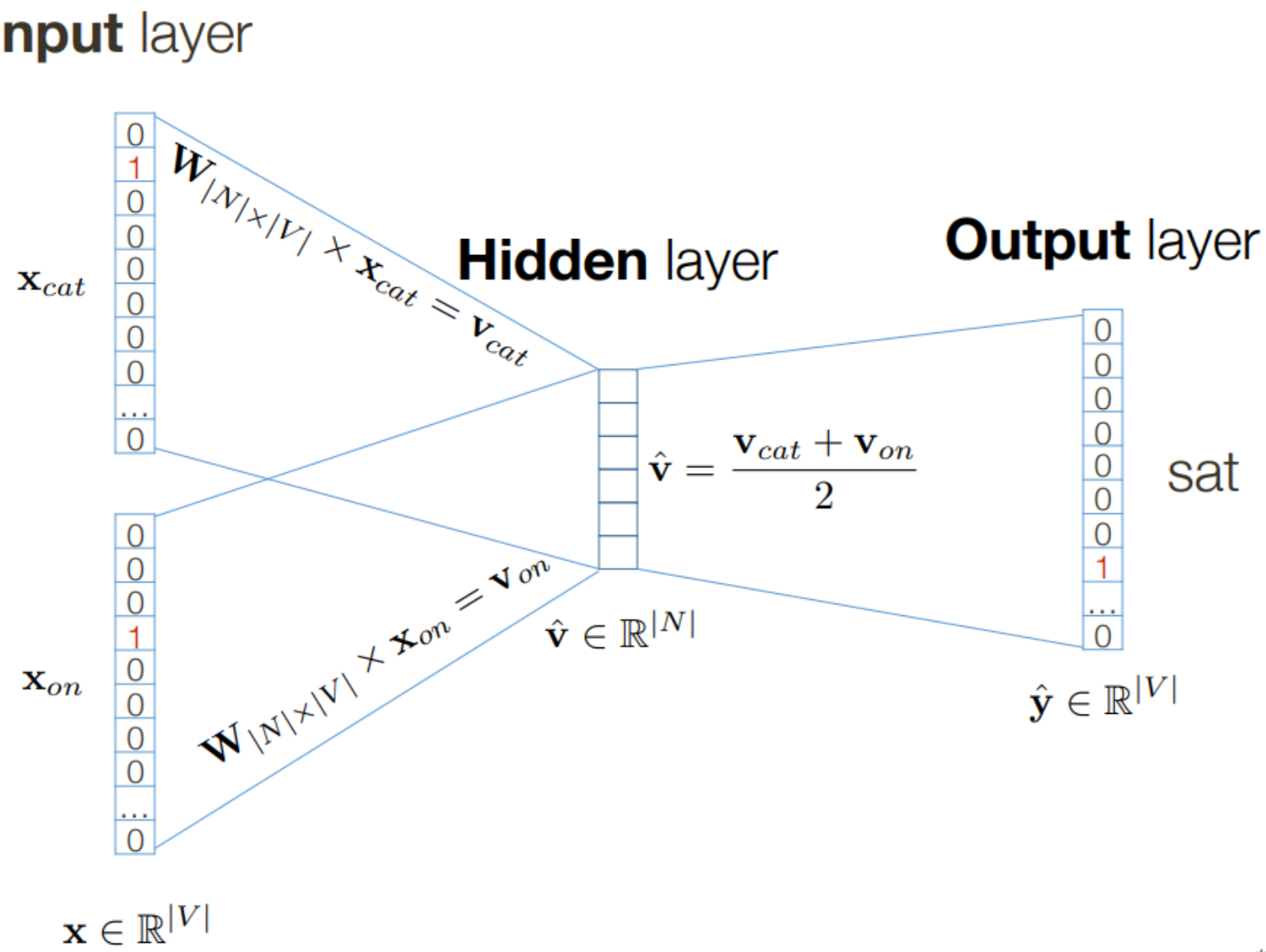
CBOW: Continuous Bag of Words

[Mikolov et al., 2013]



CBOW: Continuous Bag of Words

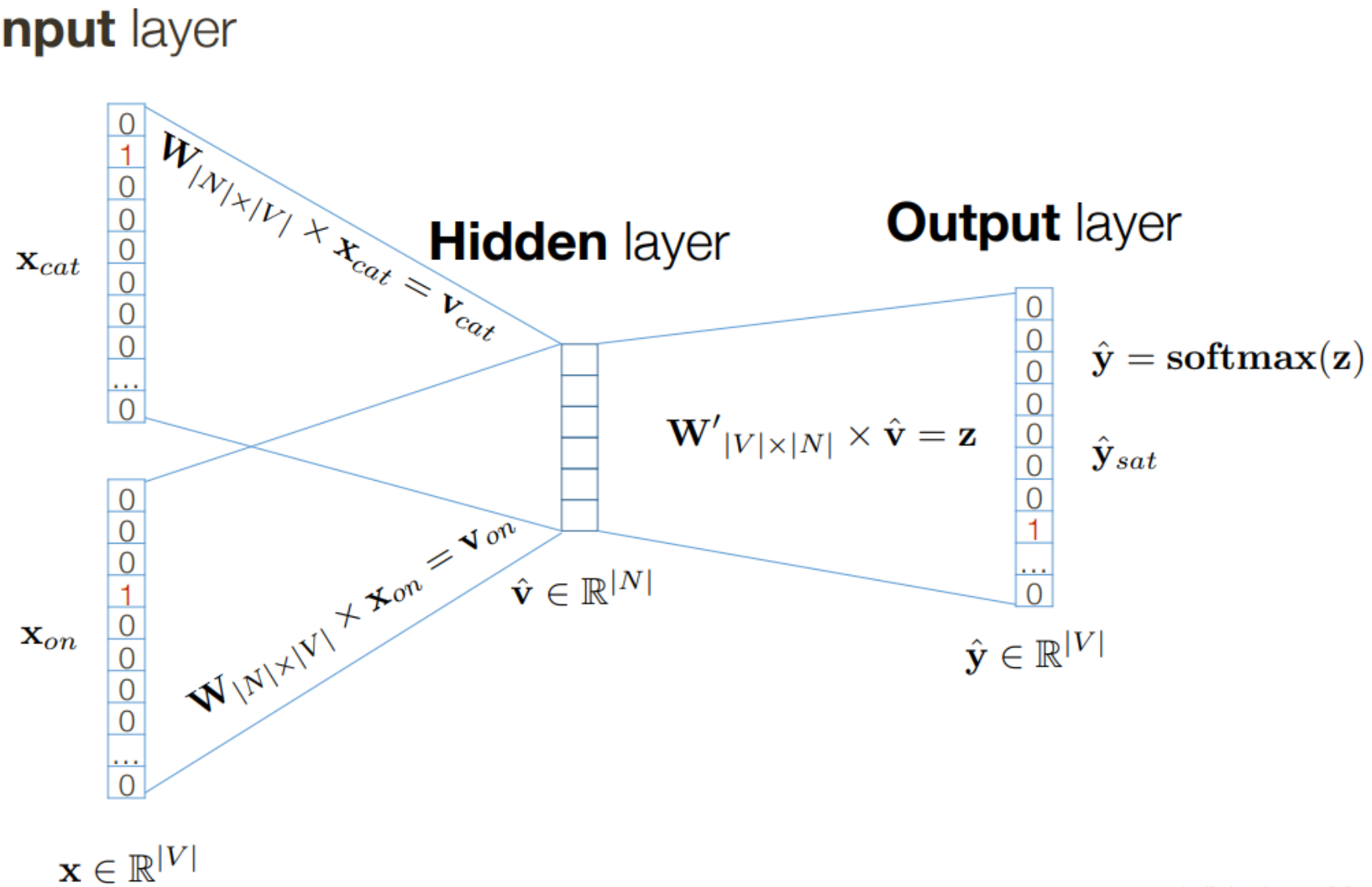
[Mikolov et al., 2013]



*slide from Vogelis Hristidis

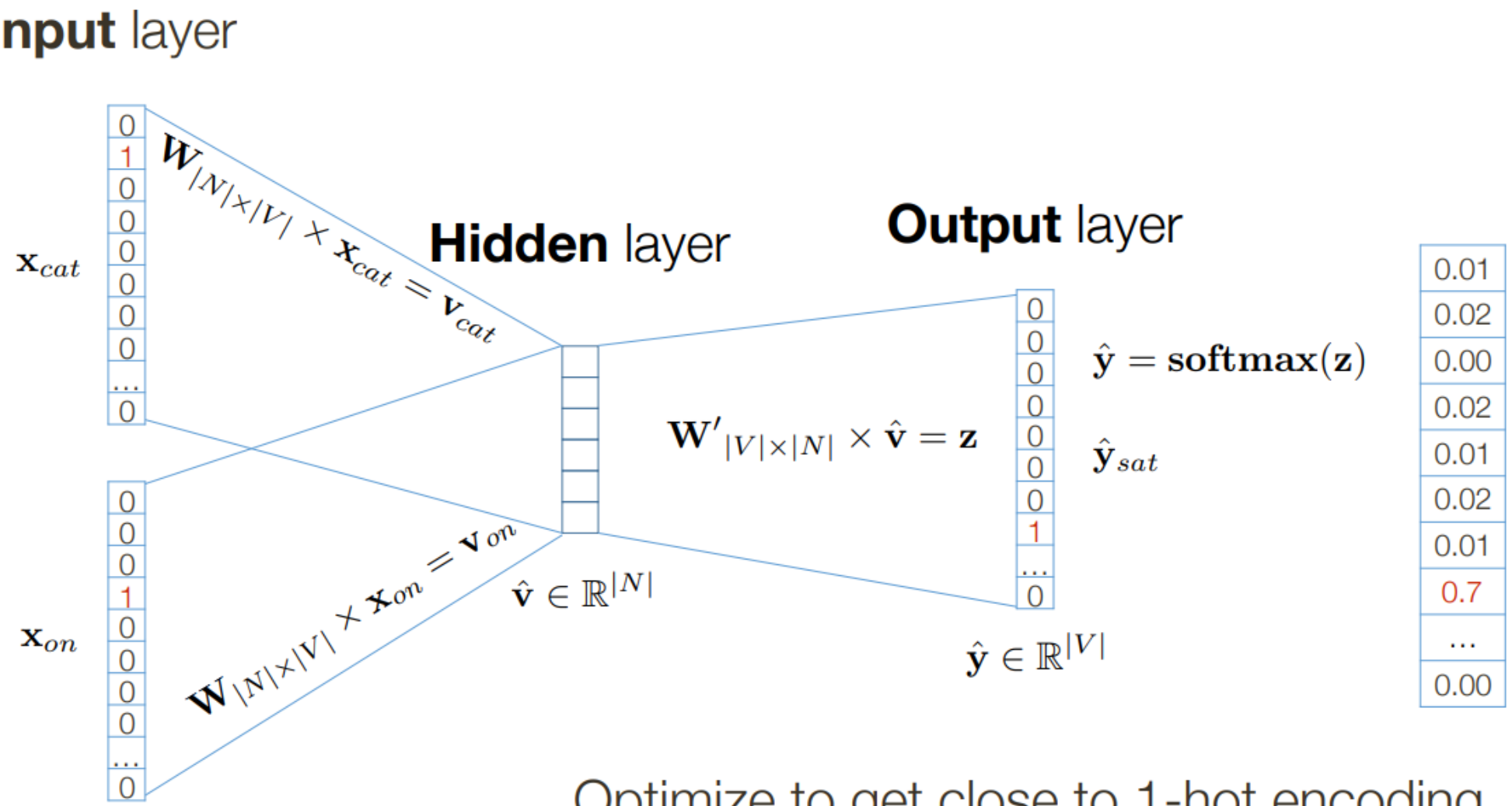
CBOW: Continuous Bag of Words

[Mikolov et al., 2013]



CBOW: Continuous Bag of Words

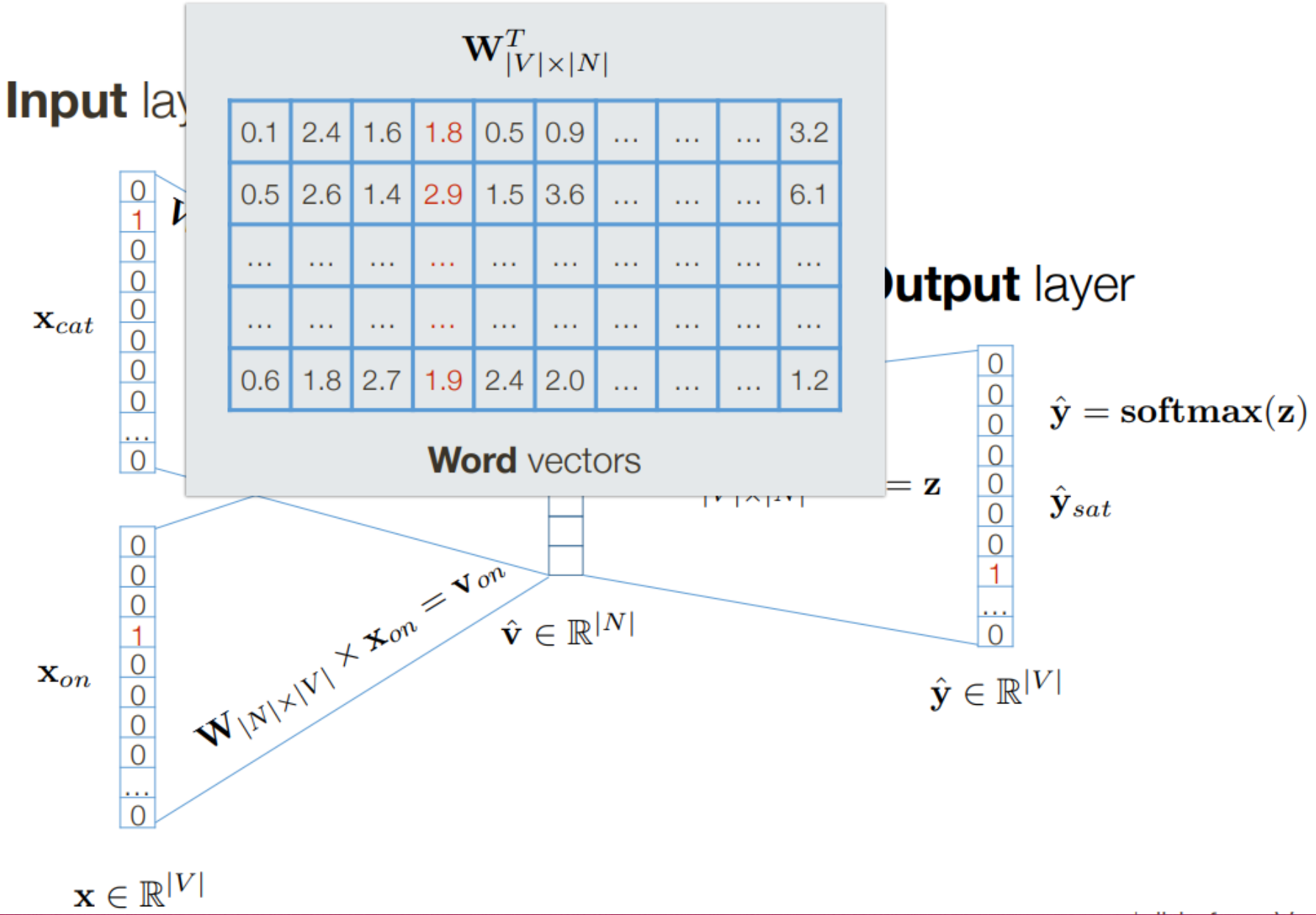
[Mikolov et al., 2013



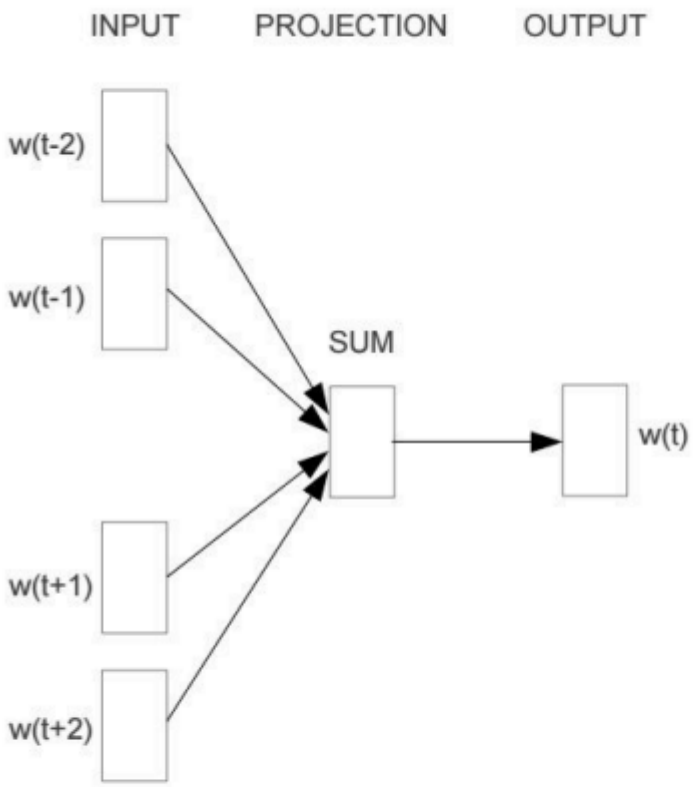
Optimize to get close to 1-hot encoding

CBOW: Continuous Bag of Words

[Mikolov et al., 2013



CBOW



$$L(\theta) = \prod_{t=1}^{T} P\left(w_t \mid \{w_{t+j}\}, -m \leq j \leq m, j \neq 0\right)$$

$$\bar{\mathbf{v}}_t = \frac{1}{2m} \sum_{-m \leq j \leq m, j \neq 0} \mathbf{v}_{t+j}$$

$$P(w_t \mid \{w_{t+j}\}) = \frac{\exp(\mathbf{u}_{w_t} \cdot \bar{\mathbf{v}}_t)}{\sum_{k \in V} \exp(\mathbf{u}_k \cdot \bar{\mathbf{v}}_t)}$$

CBOW – Loss function

The **CBOW (Continuous Bag of Words)** model predicts a **center word** given surrounding **context words**.

The loss function in CBOW is similar to Skip-Gram but reverses the prediction:

$$L_{\text{CBOW}} = -\frac{1}{T}\sum_{t=1}^{T}\log P(w_t|w_{t-j},...,w_{t+j})$$

T = Total words in the corpus

where we predict w_t using the surrounding words $w_{t-j},...,w_{t+j}$.

Using Softmax:

$$P(w_t|w_{t-j},...,w_{t+j}) = \frac{\exp(v'_{w_t}\cdot v_C)}{\sum_{w=1}^{V}\exp(v'_w\cdot v_C)}$$

where:

- v_C is the average of the embeddings of all context words.

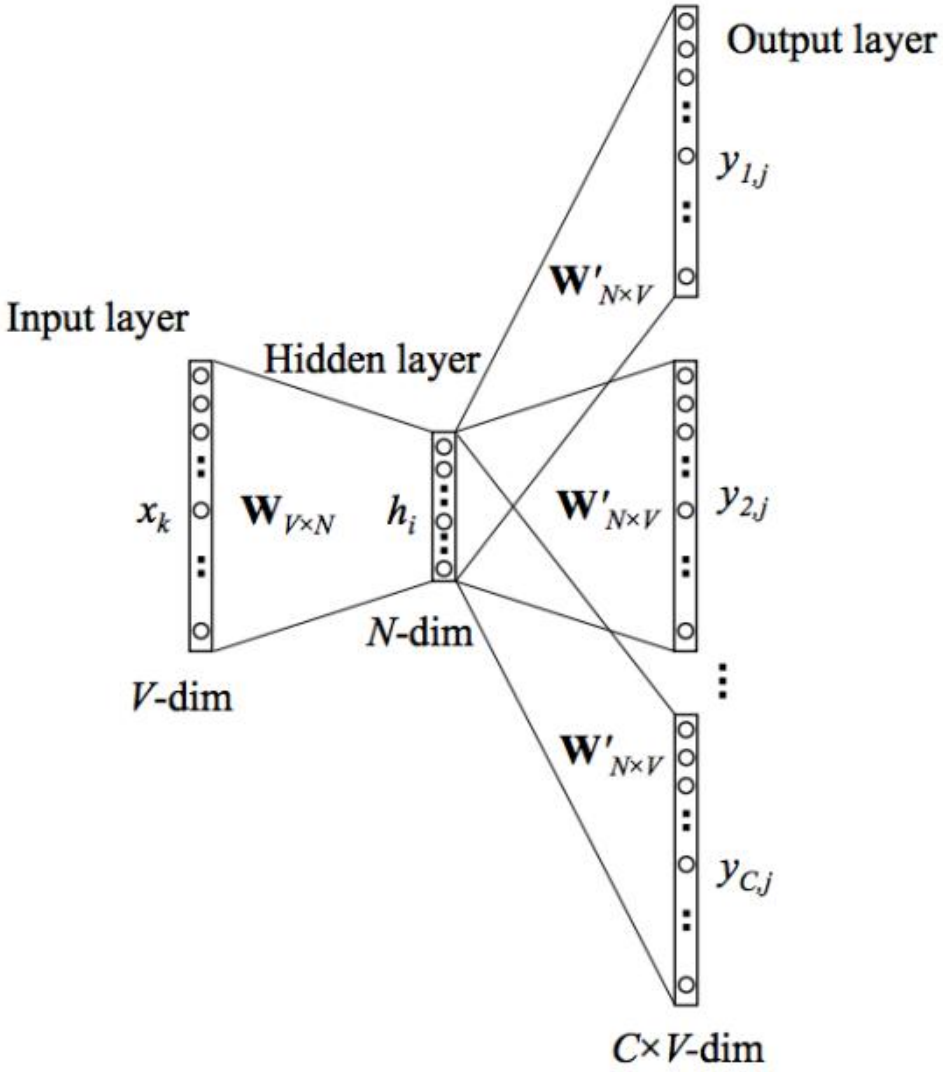
$$P(w_t|w_{t-j},...,w_{t+j}) = \frac{\exp(v'_{w_t}\cdot v_C)}{\sum_{w=1}^{V}\exp(v'_w\cdot v_C)}$$

Where:

- $P(w_t|w_{t-j},...,w_{t+j})$ → The probability of predicting the **target word** w_t given the **context words**.
- v_C → The **context vector** (sum of embeddings of surrounding words).
- v'_{w_t} → The **target word's embedding** (output vector of the target word).
- V → The **total vocabulary size**.
- $\exp(v'_{w_t}\cdot v_C)$ → Measures the similarity between the target word w_t and the context.
- $\sum_{w=1}^{V}\exp(v'_w\cdot v_C)$ → The **normalization term**, ensuring the probability distribution sums to 1.

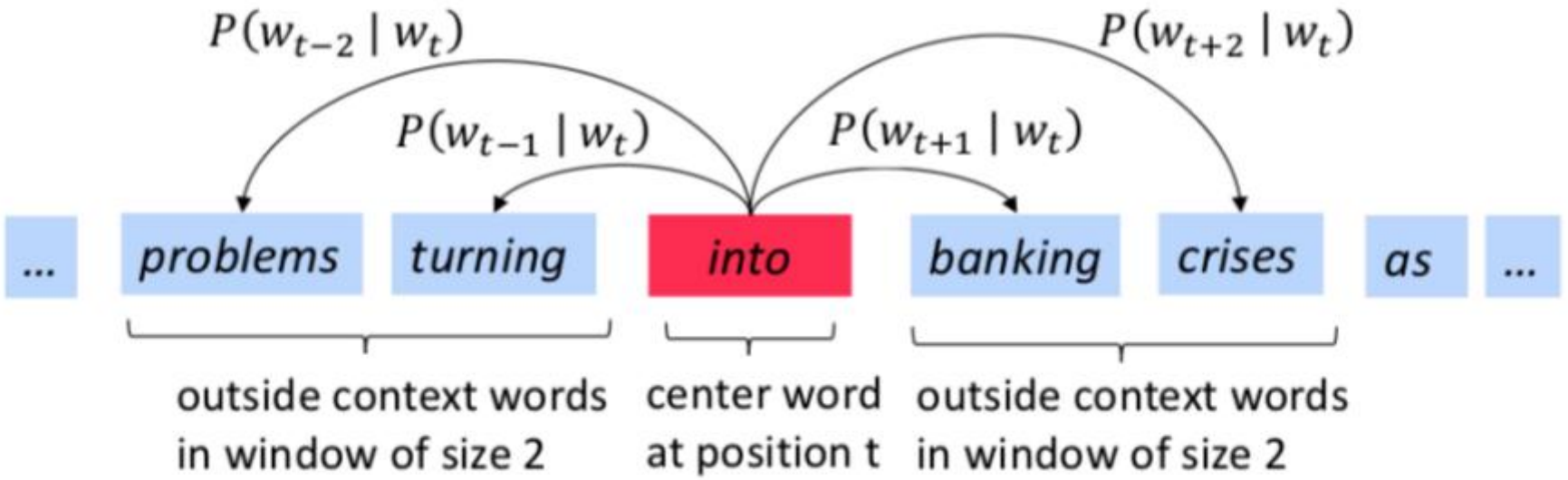
Skip-Gram Model

[Mikolov et al., 2013]

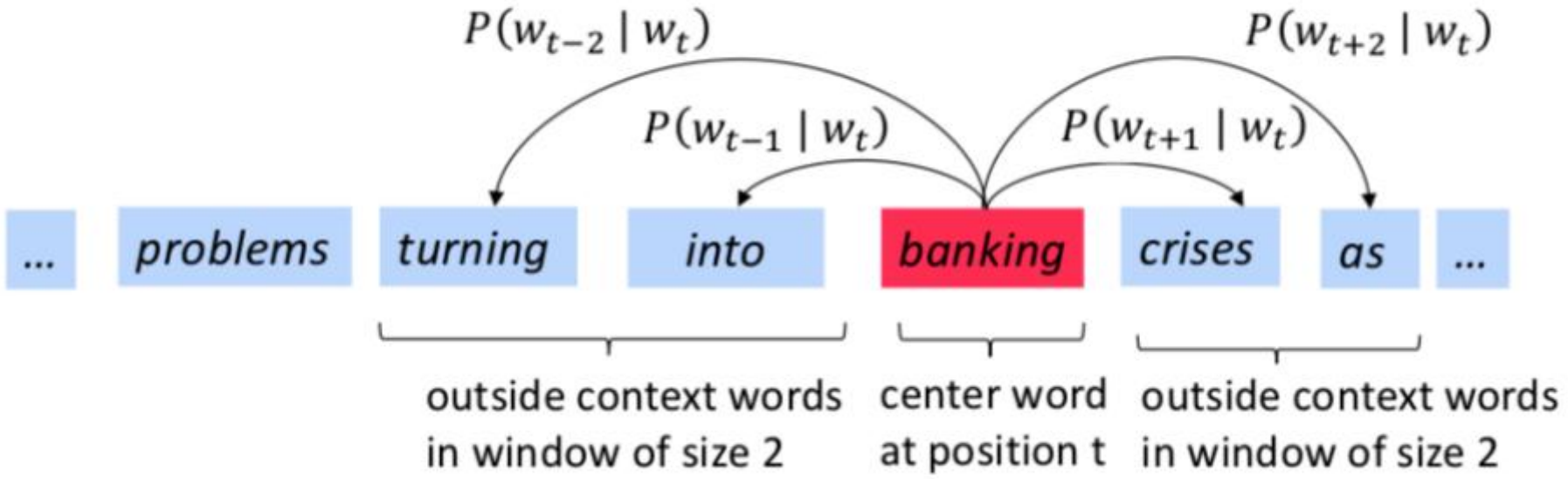


Skip-gram

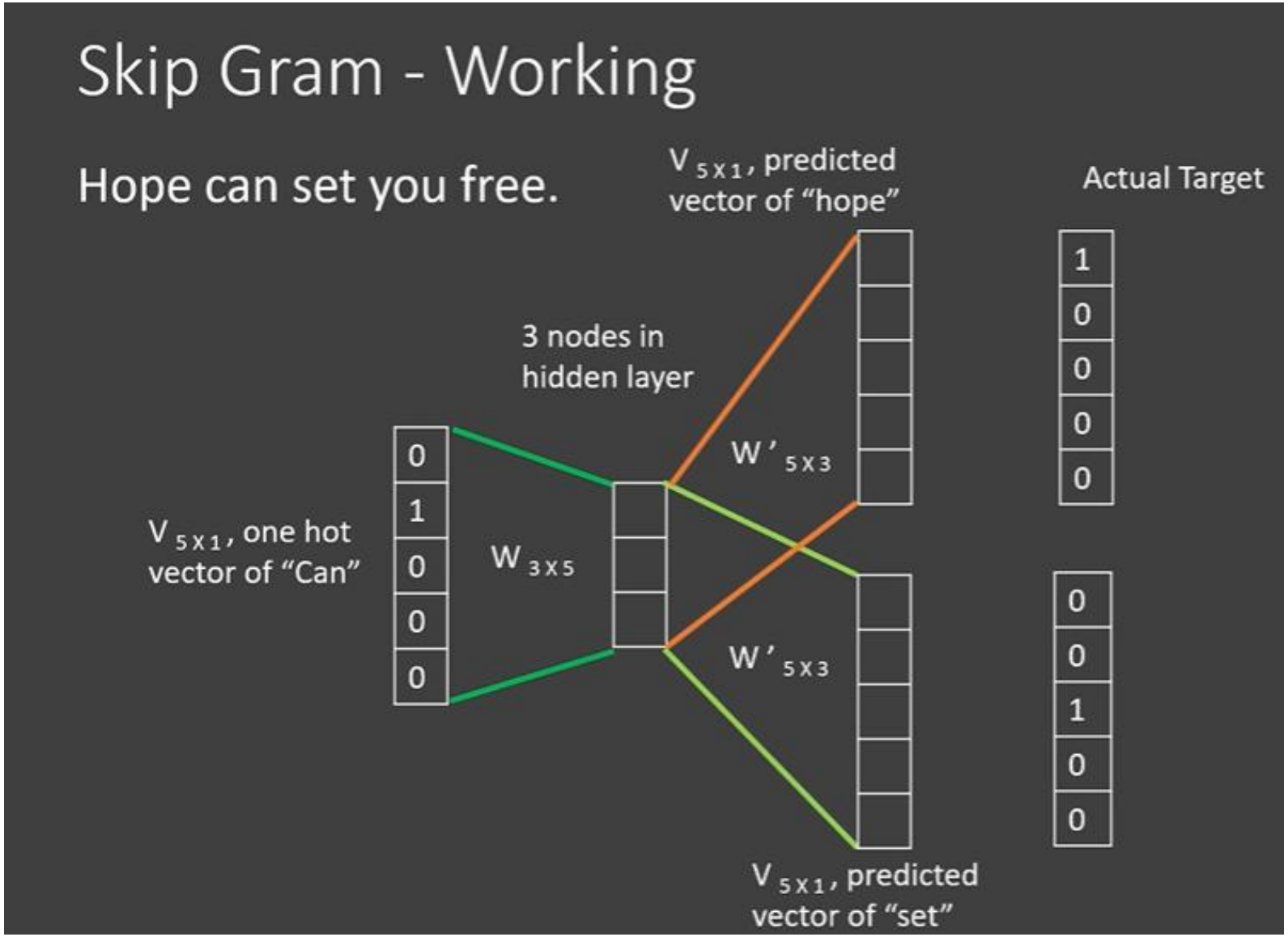
- The idea: we want to use words to **predict** their context words
- Context: a fixed window of size $2m$

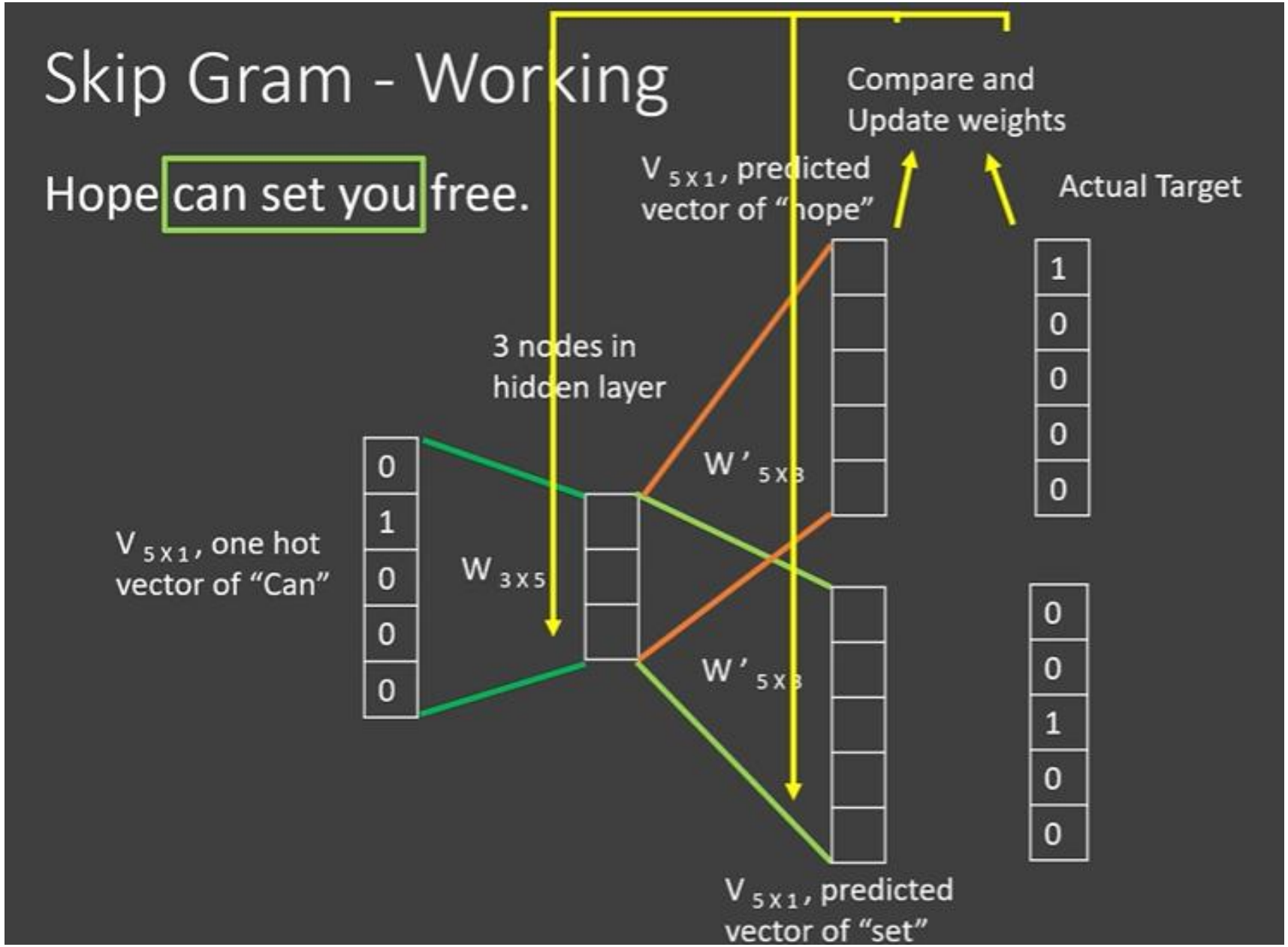


Skip-gram

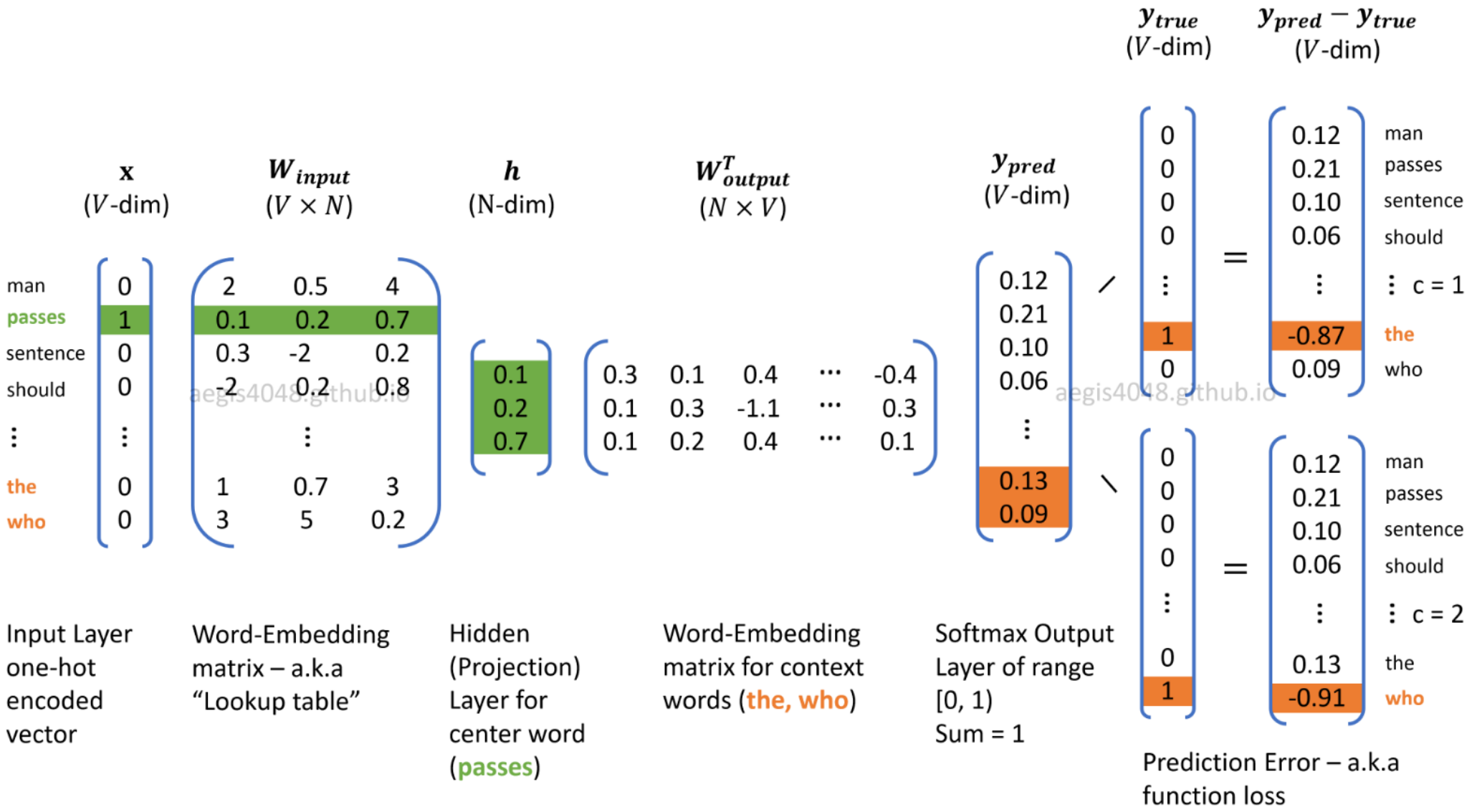

$P(w_{t-2} \mid w_t)$
$P(w_{t+2} \mid w_t)$
$P(w_{t-1} \mid w_t)$
$P(w_{t+1} \mid w_t)$
...
problems
turning
into
banking
crises
as
...
outside context words in window of size 2
center word at position t
outside context words in window of size 2

Skip Gram Working




Skip Gram - Working
Hope can set you free.
Compare and
Update weights
V 5 X 1 , predicted
vector of "hope"
Actual Target
3 nodes in
hidden layer
W ' 5 X 3
V 5 X 1 , one hot
vector of "Can"
W 3 X 5
W ' 5 X 3
V 5 X 1 , predicted
vector of "set"

Skip-gram for the word passes



Skip-gram

- For each position $t = 1,2,\ldots T$, predict context words within context size m, given center word w_j:

all the parameters to be optimized

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

- The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log \mathcal{L}(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \le j \le m, j \ne 0} \log P(w_{t+j} \mid w_t; \theta)$$

Skip-gram

- We have two sets of vectors for each word in the vocabulary

$$\mathbf{u}_i \in \mathbb{R}^d \text{ : embedding for target word } i$$

$$\mathbf{v}_{i'} \in \mathbb{R}^d \text{ : embedding for context word } i'$$

- Use inner product $\mathbf{u}_i \cdot \mathbf{v}_{i'}$ to measure how likely word i appears with context word i', the larger the better

"softmax" we learned last time!

$$P(w_{t+j} \mid w_t) = \frac{\exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_{w_{t+j}})}{\sum_{k \in V} \exp(\mathbf{u}_{w_t} \cdot \mathbf{v}_k)}$$

$\theta = \{\{\mathbf{u}_k\}, \{\mathbf{v}_k\}\}$ are all the parameters in this model!

Skip-gram Model Training

Calculating all the gradients together!

$$\theta = \{\{\mathbf{u}_k\}, \{\mathbf{v}_k\}\}$$

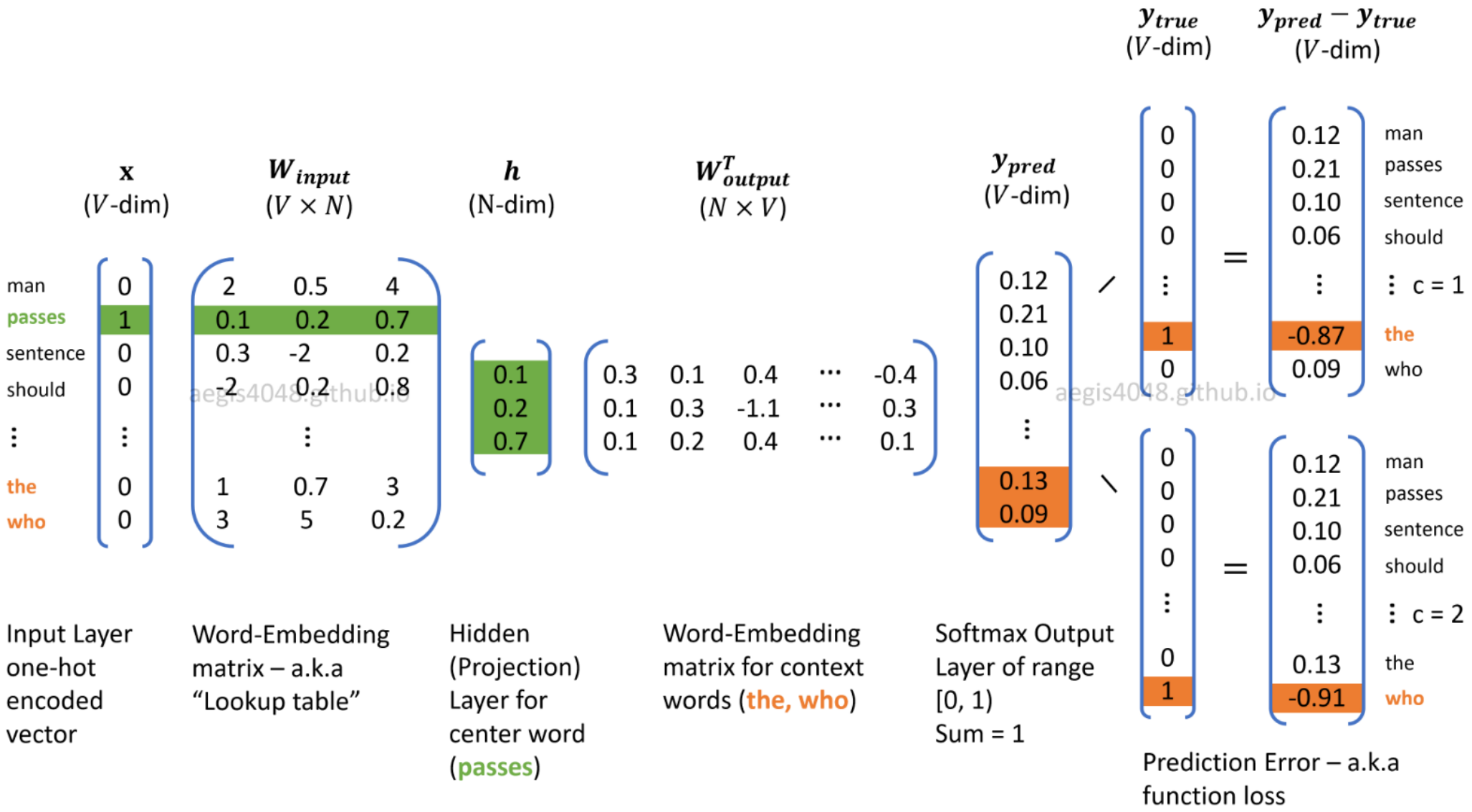
$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0} \log P(w_{t+j} \mid w_t; \theta) \qquad \nabla_\theta J(\theta) = ?$$

Q: How many parameters are in total?

We can apply stochastic gradient descent (SGD)!

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta J(\theta)$$

Skip-gram for the word passes



Skip-Gram Loss function

The **Skip-Gram** model predicts **context words** given a **center word**. It aims to maximize the probability of context words occurring around a given center word.

For a given word w_t, the model predicts surrounding words $w_{t-j}, \ldots, w_{t+j}$ in a fixed window size.

$$L = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-j \leq k \leq j, k \neq 0} \log P(w_{t+k} | w_t)$$

where:

- T = Total words in the corpus
- j = Window size
- w_t = Center word
- w_{t+k} = Context word
- $P(w_{t+k}|w_t)$ is the probability of a context word given a center word.

$P(w_{t+k}|w_t)$ is the probability of a context word given a center word.

$$P(w_O|w_I) = \frac{\exp(v'_{w_O} \cdot v_{w_I})}{\sum_{w=1}^{V} \exp(v'_w \cdot v_{w_I})}$$

Where:

- $P(w_O|w_I)$ → The probability of predicting the **output (context) word** w_O given the **input (center word** w_I.
- v_{w_I} → The **input (center) word vector**.
- v'_{w_O} → The **output (context) word vector**.
- V → The **total vocabulary size**.
- $\exp(v'_{w_O} \cdot v_{w_I})$ → Measures the similarity between the center word and a possible context word.
- $\sum_{w=1}^{V} \exp(v'_w \cdot v_{w_I})$ → **Softmax normalization term** that ensures the output is a valid probability distribution.

Getting word embeddings

Weights after training

$W_{3 \times 5}$

w00	w01	w02	w03	w04
w10	w11	w12	w13	w14
w20	w21	w22	w23	w24

One Hot vector of words $V_{5 \times 1}$

Hope	can	set	you	free
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

Word Vector for hope $= W_{3 \times 5} \times V_{5 \times 1}$

w00	w01	w02	w03	w04
w10	w11	w12	w13	w14
w20	w21	w22	w23	w24

X

1
0
0
0
0

=

$V_{3 \times 1}$

w00
w10
w20

Word Vector for Hope

Comparison

[Mikolov et al., 2013

- **CBOW** is not great for rare words and typically needs less data to train
- **Skip-gram** better for rate words and needs more data to train the model

Skip Gram: Better for Rare Words: Since it learns from a **single word** instead of averaging context words, it gives better representations for infrequent words.

Model	Vector Dimensionality	Training words	Accuracy [%]		
			Semantic	Syntactic	Total
Collobert-Weston NNLM	50	660M	9.3	12.3	11.0
Turian NNLM	50	37M	1.4	2.6	2.1
Turian NNLM	200	37M	1.4	2.2	1.8
Mnih NNLM	50	37M	1.8	9.1	5.8
Mnih NNLM	100	37M	3.3	13.2	8.8
Mikolov RNNLM	80	320M	4.9	18.4	12.7
Mikolov RNNLM	640	320M	8.6	36.5	24.6
Huang NNLM	50	990M	13.3	11.6	12.3
Our NNLM	20	6B	12.9	26.4	20.3
Our NNLM	50	6B	27.9	55.8	43.2
Our NNLM	100	6B	34.2	**64.5**	50.8
CBOW	300	783M	15.5	53.1	36.1
Skip-gram	300	783M	**50.0**	55.9	**53.3**

CBOW

- Faster Training: Since CBOW averages context word embeddings, it converges faster.
- Works Well on Small Datasets: Suitable when the dataset is not very large.
- Ignores Word Order: Since it takes an average of surrounding words, it loses word order information.
- Less Effective for Rare Words: Frequent words dominate learning, making rare words harder to train

Skip Gram

- Better for Rare Words: Since it learns from a single word instead of averaging context words, it gives better representations for infrequent words.
- Captures Complex Semantic Relationships:
- Slower Training: Skip-gram requires more updates since it predicts multiple context words per target word.
- Requires More Data: Works best when trained on large datasets.

DrawBacks of word2vec

It mainly has 2 drawbacks:

- Captured only local-context but the overall corpus context was modelled effectively.
- Not able to generate vectors for unseen words – Hence method: Gloves

- word embedding models like word2vec and GloVe cannot provide embeddings for out-of-vocabulary words. – Hence Fasttext
- The intuition behind fastText is that by using a bag of character n-grams, you can learn representations for morphologically rich languages