

VECTOR SEMANTICS AND EMBEDDINGS

Spring 2023

CS6431 Natural Language Processing

Credits

B1: *Speech and Language Processing (Third Edition draft – Jan2022)*

Daniel Jurafsky, James H. Martin

Assignment

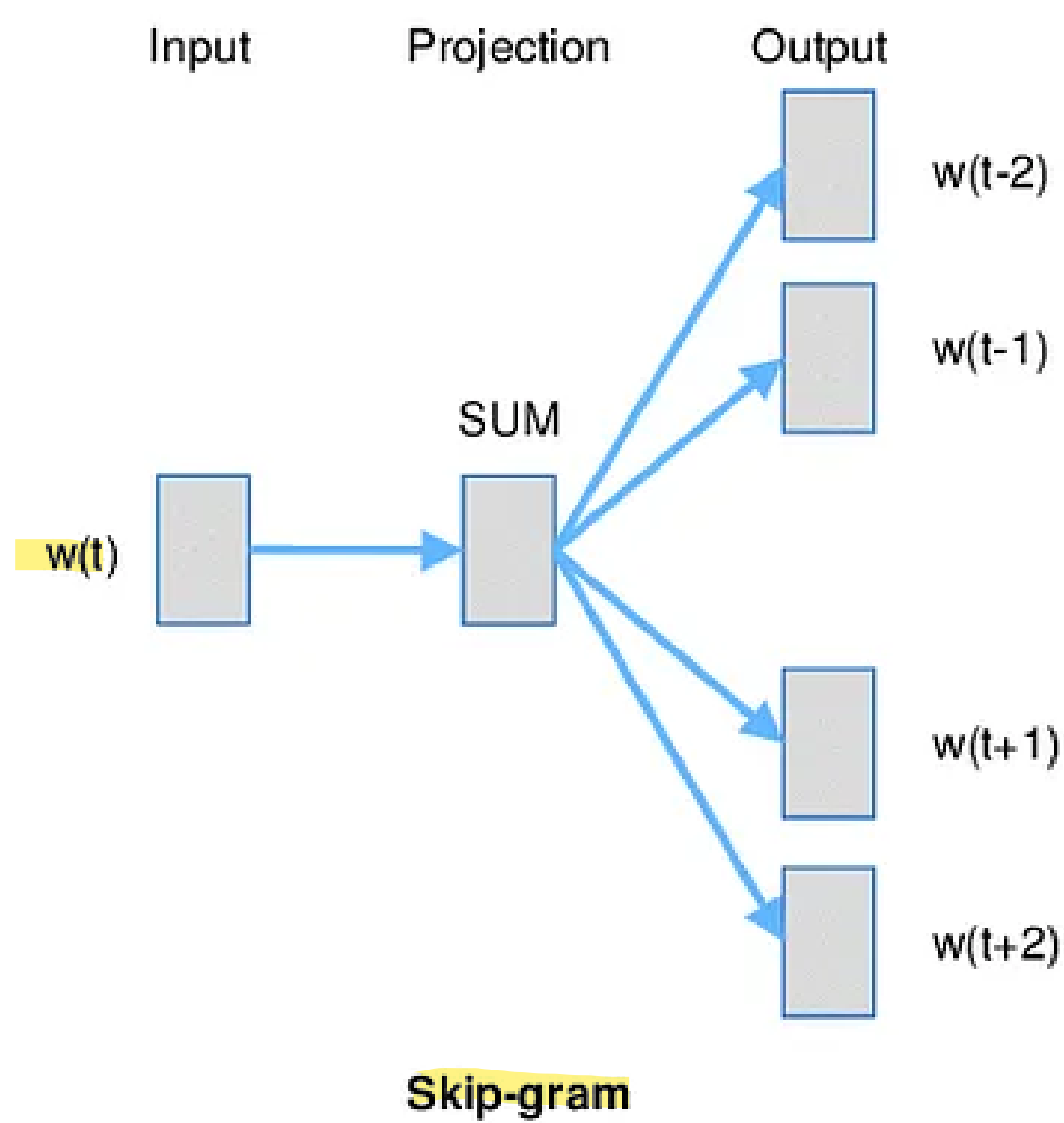
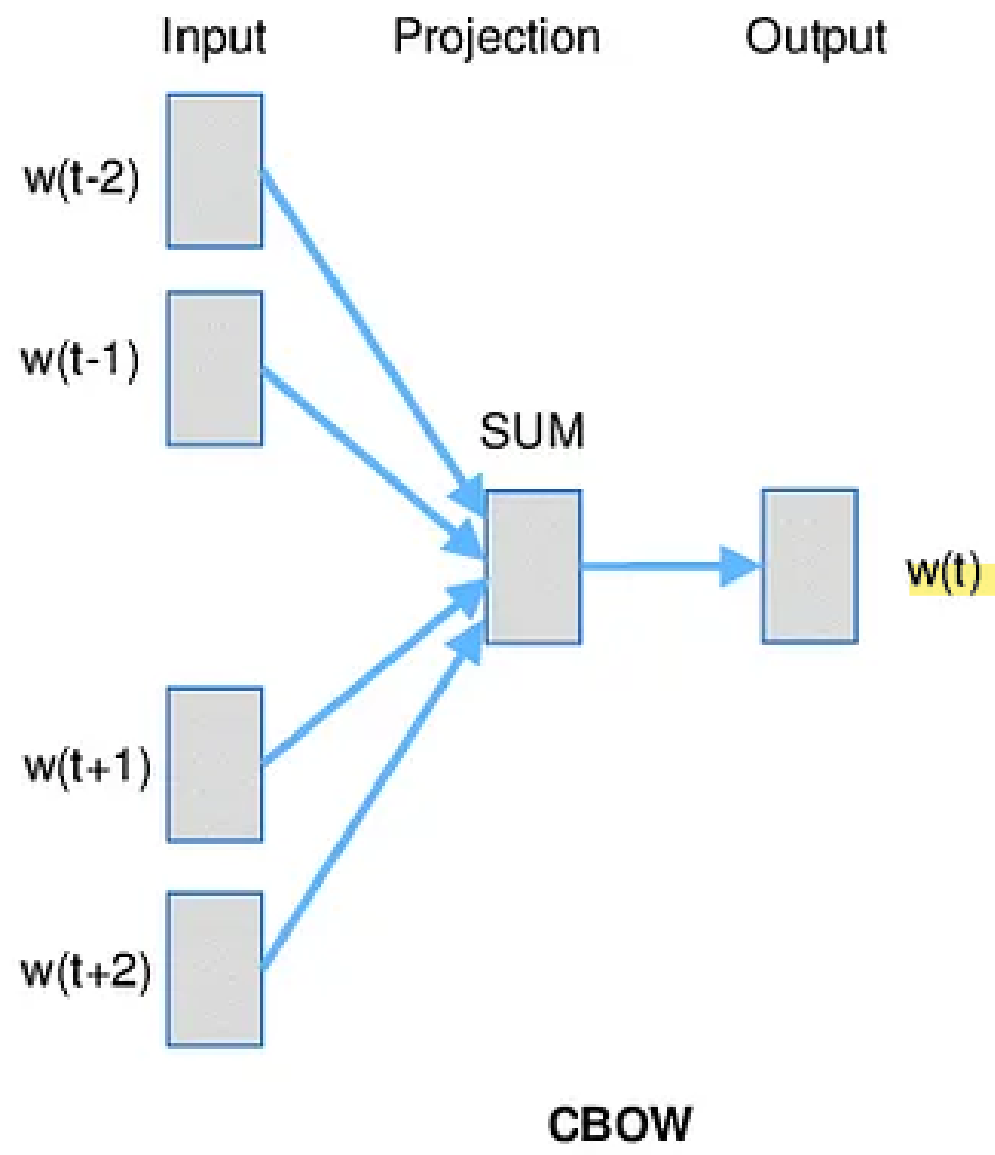
Read:

B1: Chapter 6

Problems:

Word2Vec

- Embeds higher dimension to lower dimension vector space using shallow neural network and predicting words using context
 - ▣ One or two hidden layer neural network.
- Two version of Word2Vec:
 - ▣ Continuous Bag-of-Word (CBow)
 - ▣ Skip-Gram



Bag-of-Word

- Gets rid of word order. Used in discrete case using counts of words that appear. E.g.,
 - ▣ S1 : John likes to watch movies. Mary likes movies too.
 - ▣ S2 : Mary also likes to watch football games.
 - ▣ BOW1 = {"John": 1, "likes": 2, "to": 1, "watch": 1, "movies": 2, "Mary": 1, "too": 1}
 - ▣ BOW2 = {"Mary": 1, "also": 1, "likes": 1, "to": 1, "watch": 1, "football": 1, "games": 1}
- Vector Representation :
 - ▣ $V1 = [1, 2, 1, 1, 2, 1, 1, 0, 0, 0]$ john likes to watch movies mary too also football games
 - ▣ $V2 = [0, 1, 1, 1, 0, 1, 0, 1, 1, 1]$
- No information about word meaning and their sequences.

Continuous Bag-of-Word (CBoW)

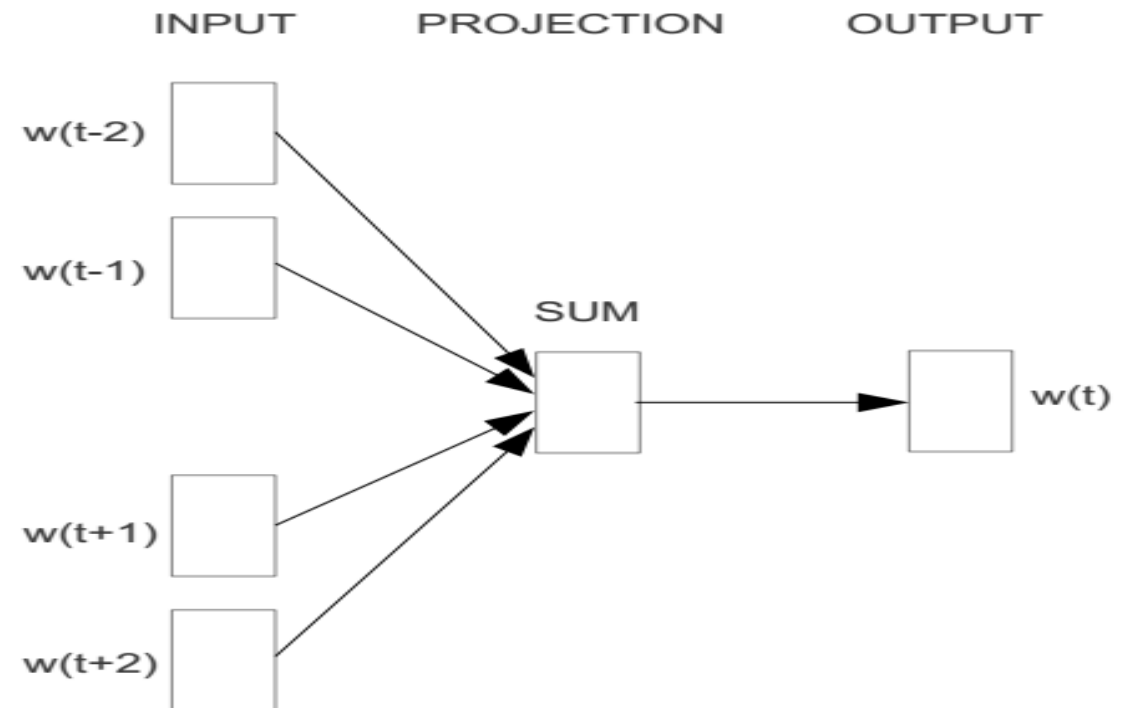
- Takes vector embedding of 'n' words before target and n words after and adds them (as vectors).
- Also removes word order, but the vector sum is meaningful enough to deduce missing word.

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

Write on desk

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

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



□ An example: 'The cat **sat** on floor.'

▣ Context size = **2**, Vocabulary size = **V**

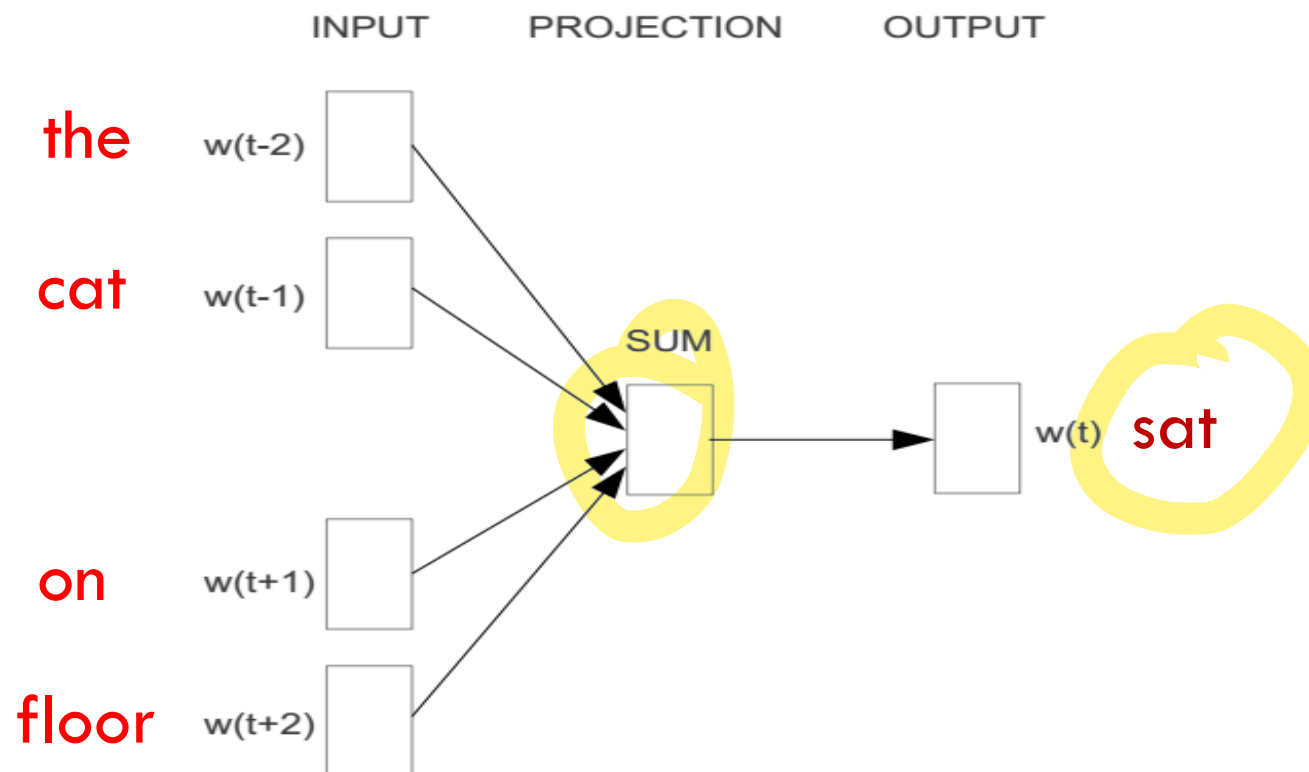
Training Samples

(the, sat)

(cat, sat)

(on, sat)

(floor, sat)



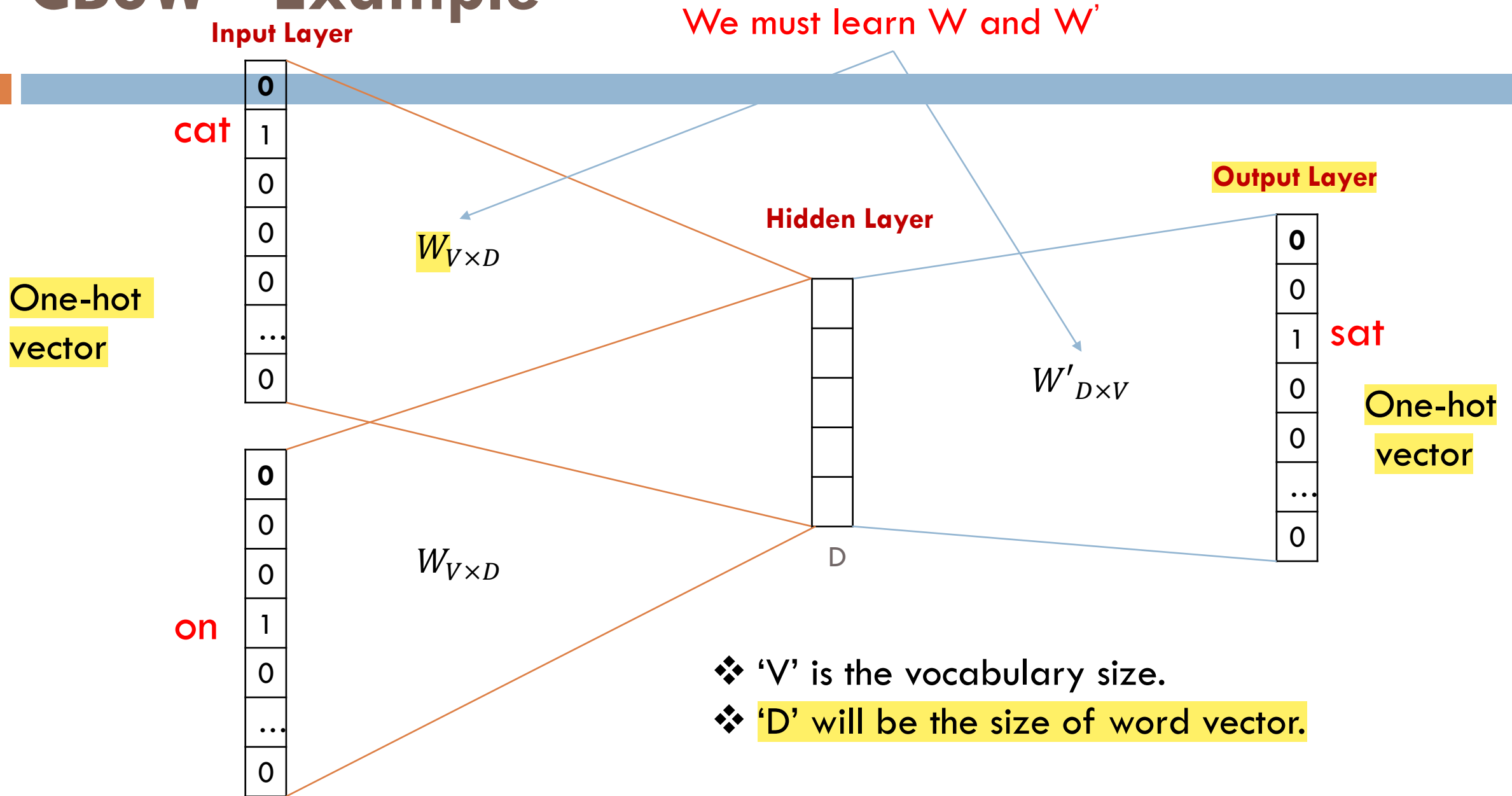
CBoW - Example

One-hot Encoding of the sample : The cat **sat** on floor.

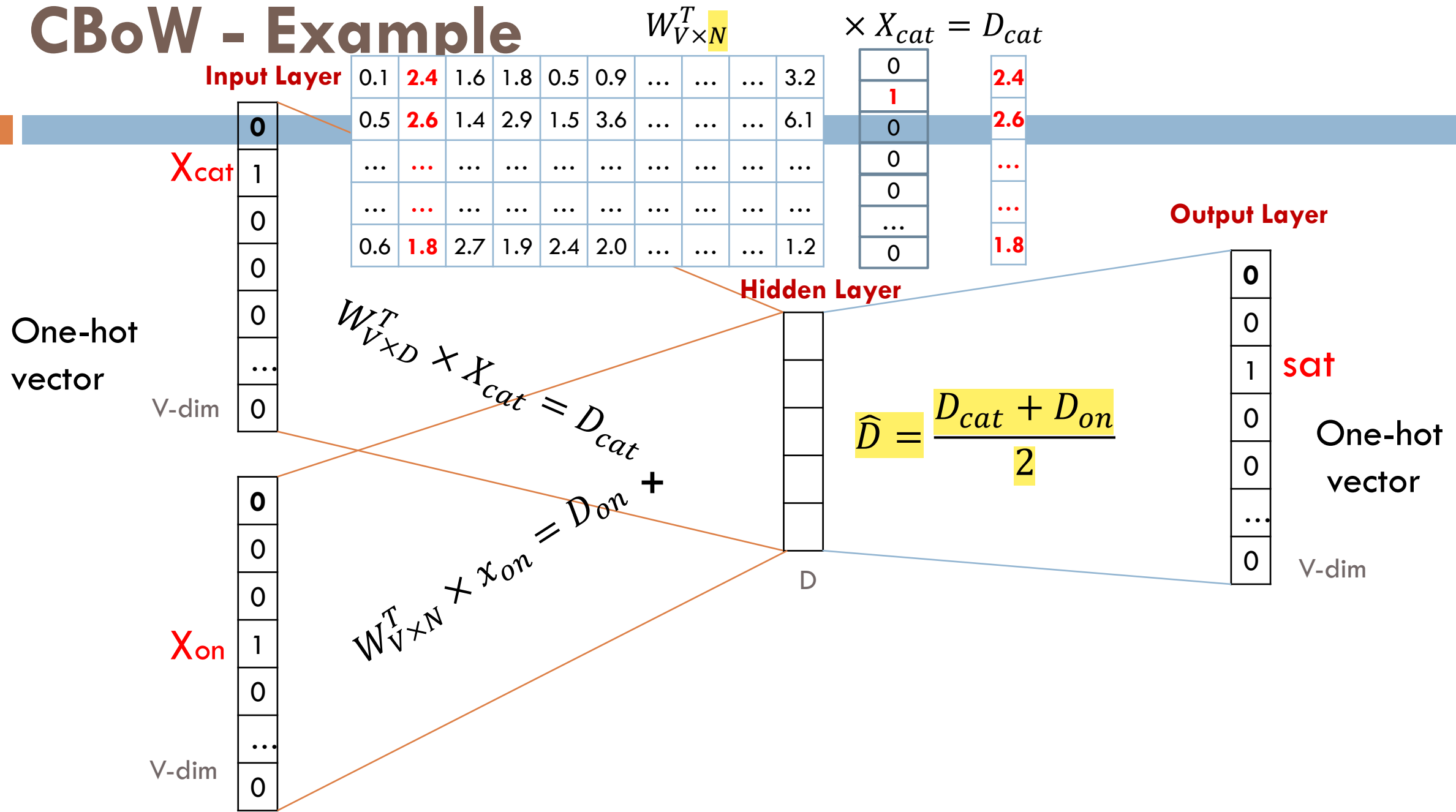
One-hot Encoding length = V (Vocabulary size)

the	cat	sat						Samples
1	0	0	0	0	0	...	0	The
0	1	0	0	0	0	...	0	Cat
0	0	1	0	0	0	...	0	Sat
0	0	0	1	0	0	...	0	On
0	0	0	0	1	0	...	0	floor

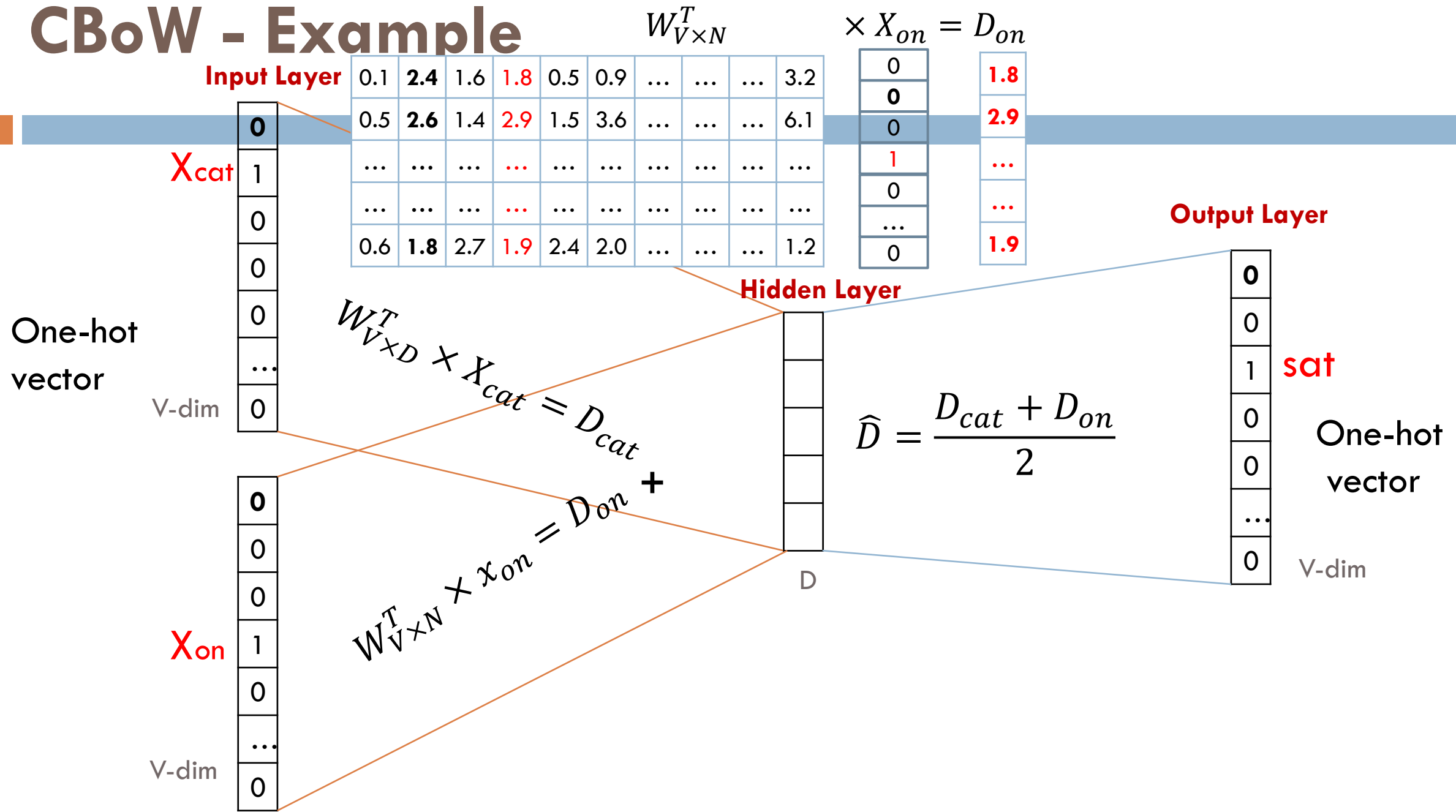
CBoW - Example



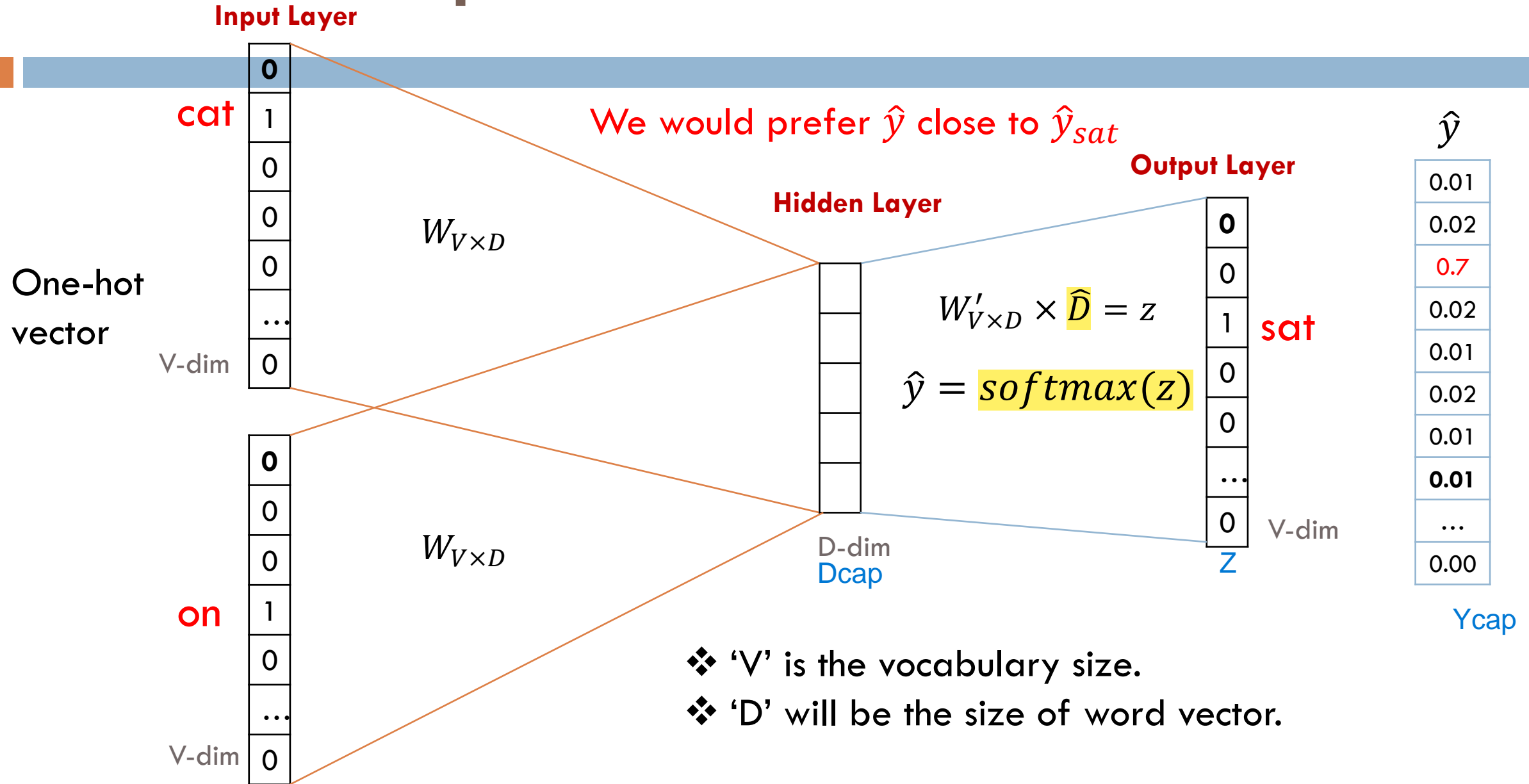
CBoW - Example



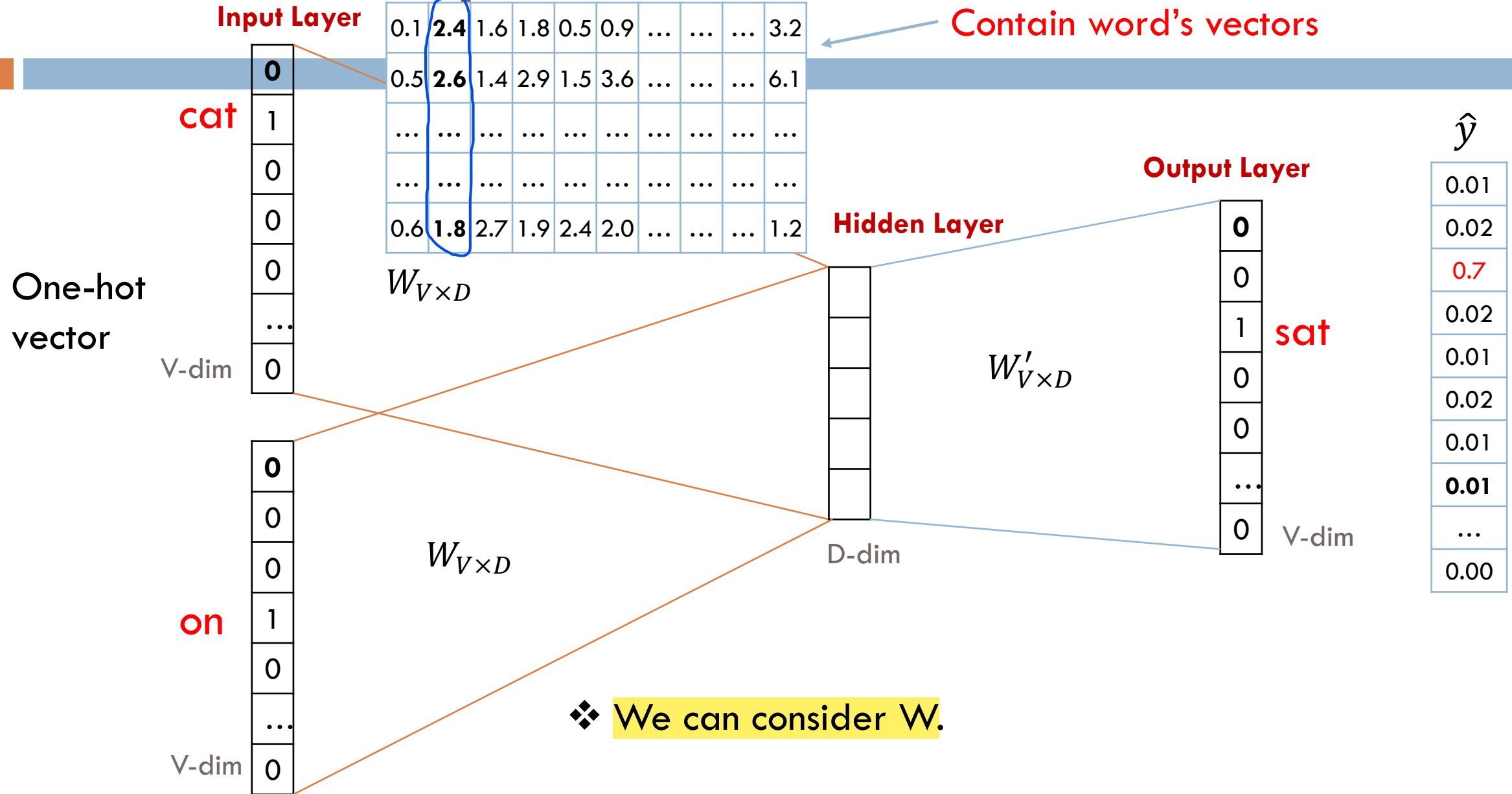
CBoW - Example



CBoW - Example



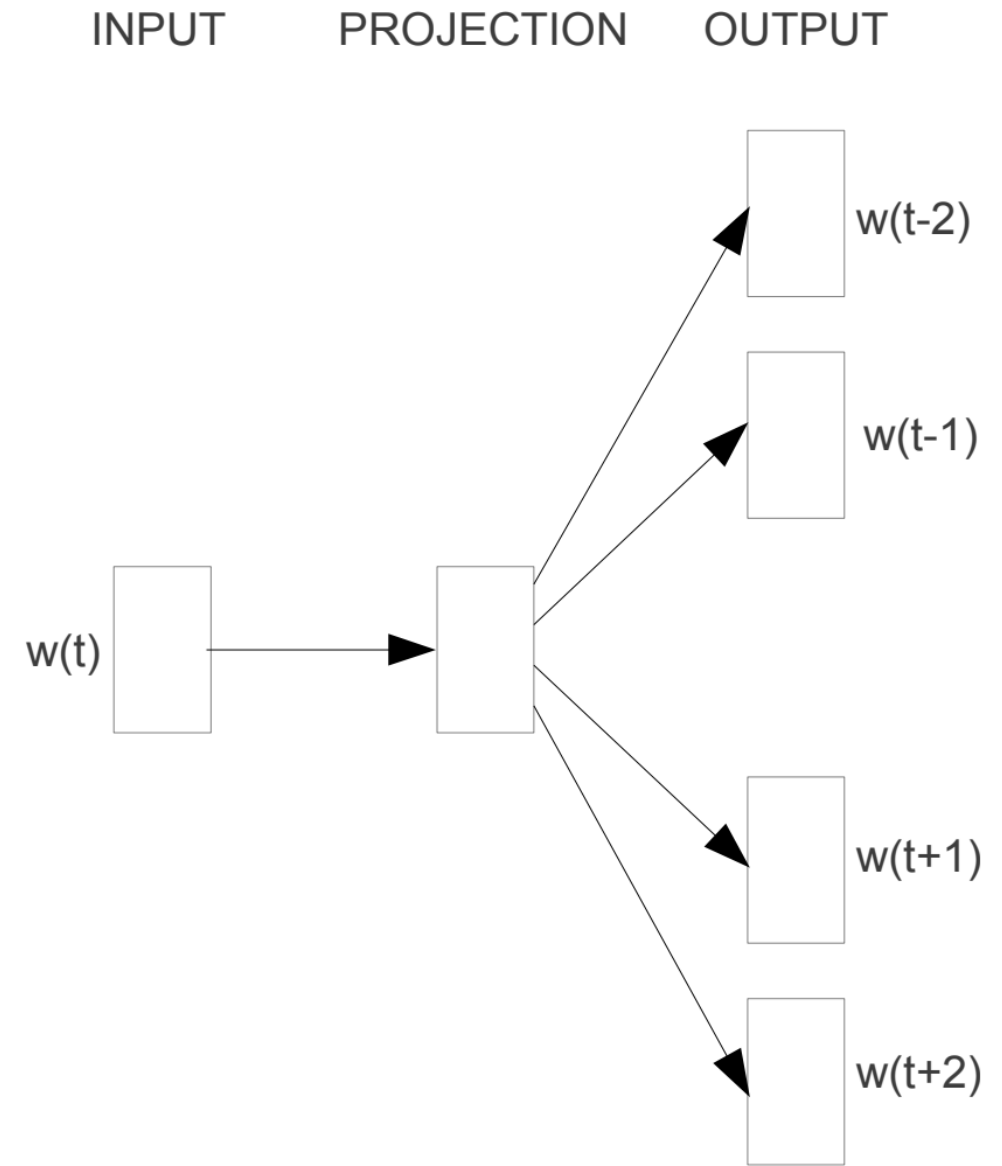
CBoW - Example



Skip Gram

Skip gram – alternative to CBOW

- ❑ Start with a single word embedding and try to predict the surrounding words.
- ❑ Much less well-defined problem, but works better in practice (scales better).



Skip-gram


Skip Gram Objective Function

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

write on desk

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

all the parameters to be optimized



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

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

How to define $P(w_{t+j} | w_t; \theta)$

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

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


$\mathbf{v}_{i'} \in \mathbb{R}^d$: 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

Write on desk

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

Softmax

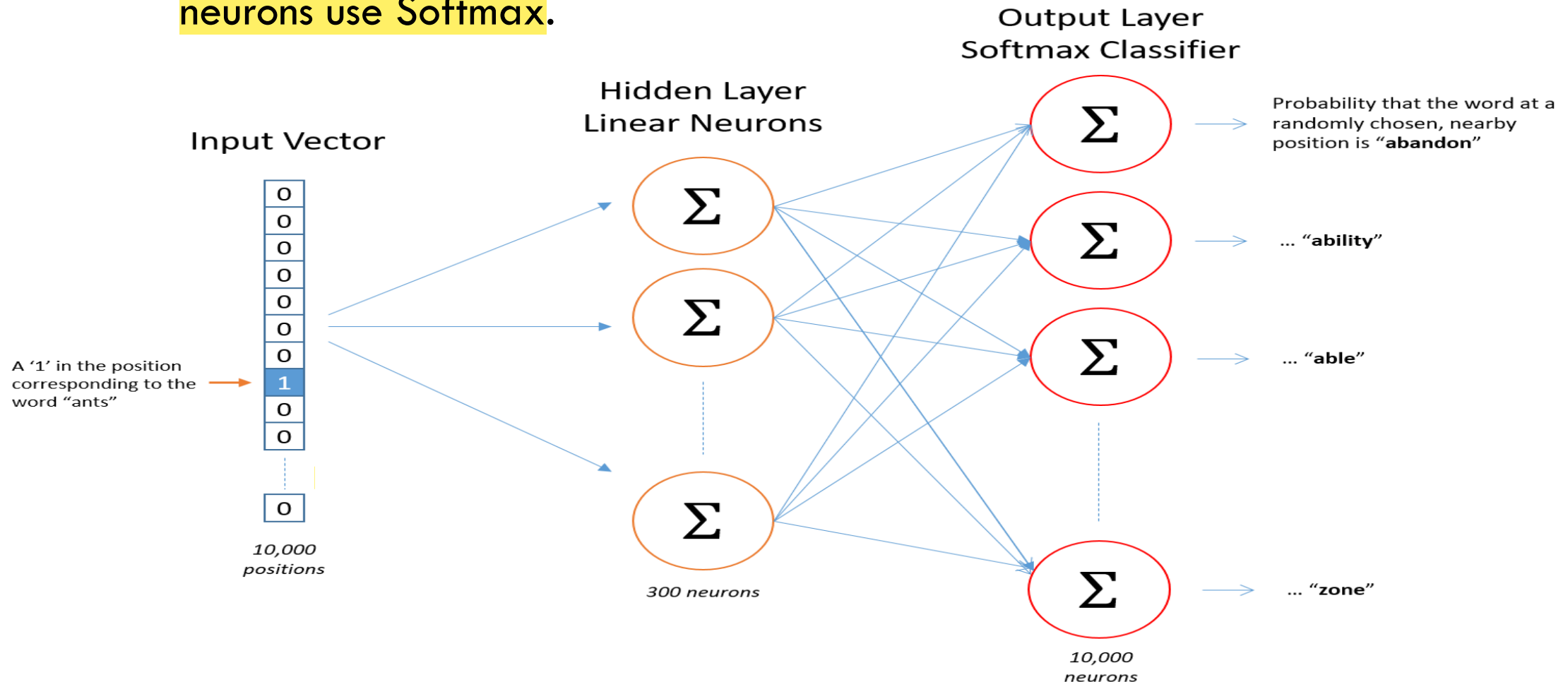


$\theta = \{\{\mathbf{u}_k\}, \{\mathbf{v}_k\}\}$ are all the parameters in this model!

Skip gram

□ Map from center word to probability on surrounding words. One input/output unit below.

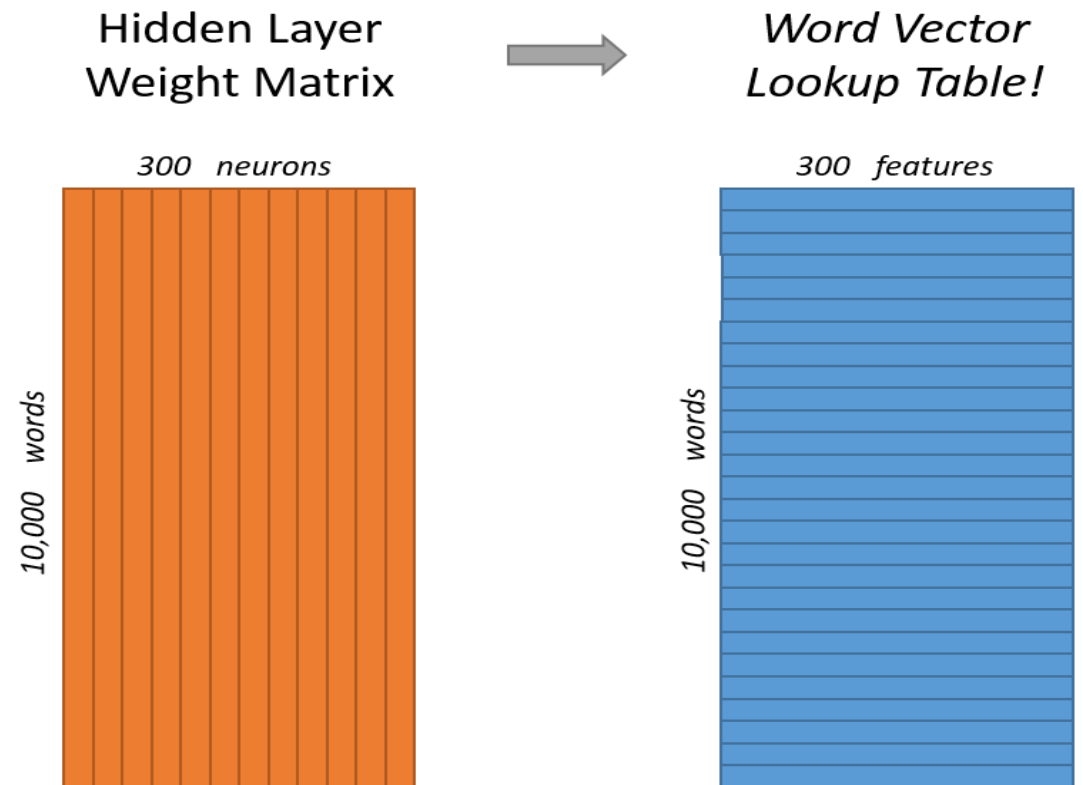
➤ There is no activation function on the hidden layer neurons, but the output neurons use Softmax.



Skip gram Example

- ❑ Vocabulary of 10,000 words.
- ❑ Embedding vectors with 300 features.
- ❑ So the hidden layer is going to be represented by a weight matrix with 10,000 rows (multiply by vector on the left).

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = \begin{bmatrix} 10 & 12 & 19 \end{bmatrix}$$



Word2vec shortcomings

Problem: 10,000 words and 300 dim embedding gives a large parameter space to learn. And 10K words is minimal for real applications.

Slow to train, and need lots of data, particularly to learn uncommon words.

Word2vec improvements: word pairs and phrases

Idea: Treat common word pairs or phrases as single “words.”

E.g., Boston Globe (newspaper) is different from Boston and Globe separately.

Embed Boston Globe as a single word/phrase.

Method: make phrases out of words which occur together often relative to the number of individual occurrences. Prefer phrases made of infrequent words in order to avoid making phrases out of common words like “and the” or “this is”.

Pros/cons: Increases vocabulary size but decreases training expense.

Results: Led to 3 million “words” trained on 100 billion words from a Google News dataset.

Word2vec improvements: subsample frequent words

Idea: Subsample frequent words to decrease the number of training examples.

The probability that we cut the word is related to the word's frequency. More common words are cut more.

Uncommon words (anything $< 0.26\%$ of total words) are kept

E.g., remove some occurrences of “the.”

Method: For each word, cut the word with probability related to the word's frequency.

Benefits: If we have a window size of 10, and we remove a specific instance of “the” from our text:

As we train on the remaining words, “the” will not appear in any of their context windows.

Word2vec improvements: selective updates

Idea: Use “Negative Sampling”, which causes each training sample to update only a small percentage of the model’s weights.

Observation: A “correct output” of the network is a **one-hot vector**. That is, one neuron should output a 1, and *all* of the other thousands of output neurons to output a 0.

Method: With negative sampling, randomly select just a small number of “negative” words (let’s say 5) to update the weights for. (In this context, a “negative” word is one for which we want the network to output a 0 for). We will also still update the weights for our “positive” word.

Skip-gram with negative sampling (SGNS)

Idea: recast problem as binary classification!

- Target word is positive example
- All words not in context are negative

$$P(D = 1 \mid t, c) = \sigma(\mathbf{u}_t \cdot \mathbf{v}_c)$$

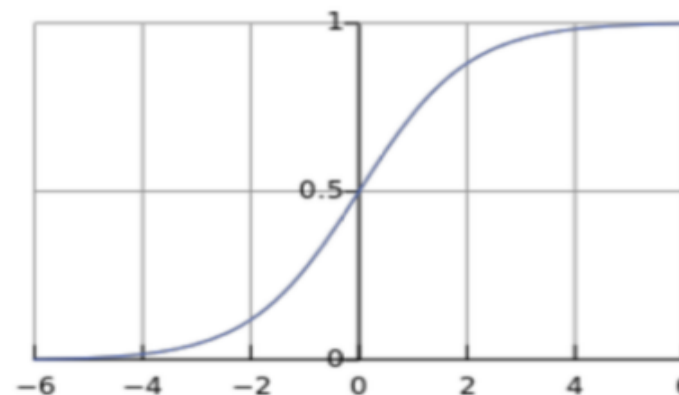
$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -

t	c	t	c
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if



To compute loss, pick K random words as negative examples:

$$J(\theta) = -P(D = 1 \mid t, c) - \frac{1}{K} \sum_{i=1}^K P(D = 0 \mid t_i, c)$$

Loss Function

- ❑ Maximize the similarity of the target word, context word pairs (w, c_{pos}) drawn from the positive examples.
- ❑ Minimize the similarity of the (w, c_{neg}) pairs from the negative examples.
- ❑ Maximize the dot product of the word with the actual context words, and minimize the dot products of the word with the k negative sampled non-neighbor words.

$$\begin{aligned} L_{CE} &= -\log \left[P(+|w, c_{pos}) \prod_{i=1}^k P(-|w, c_{neg_i}) \right] \\ &= - \left[\log P(+|w, c_{pos}) + \sum_{i=1}^k \log P(-|w, c_{neg_i}) \right] \\ &= - \left[\log P(+|w, c_{pos}) + \sum_{i=1}^k \log (1 - P(+|w, c_{neg_i})) \right] \\ &= - \left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^k \log \sigma(-c_{neg_i} \cdot w) \right] \end{aligned}$$

Glove: Global Vectors for Word Representation

- ❑ Word2Vec is a predictive model — learning vectors to improve the predictive ability on local context word, GloVe is a count-based model uses global statistic with local information for word vector.
- ❑ Count-based models learn vectors by doing dimensionality reduction on a co-occurrence counts matrix.
 - ❑ Factorize this matrix to yield a lower-dimensional matrix of words and features, where each row yields a vector representation for each word.
 - ❑ The counts matrix is preprocessed by normalizing the counts and log-smoothing them.