Capturing the meaning of words

How do you feed words to machines?

- Example: the cat sat on floor and is looking at wall

- How do you feed words to machines to perform NLP tasks?

- Possible Solutions
  - One-hot encoding
  - Context based approaches
    - Co-occurrence Matrix with SVD
    - word2vec (*Google)*
    - Global Vector Representations (GloVe)

I. One-hot encoded vectors

- Example: the cat sat on floor and is looking at wall

- How do you feed text to machines?

the	0
cat	0
sat	0
on	0
floor	0
and	0
is	0
looking	0
at	0
wall	0

Issues with one-hot-encoding vectors

- One-hot-encoded vectors doesn’t represent word meaning. Similar words such as English and French, cat and dog should have similar vector representations. However, similarity between all “one hot vectors” is the same.

- Sparsity

II. Context based approaches

- The semantics of a word is characterized by the company it keeps around i.e., *Words which are similar in meaning occur in similar contexts*

Sentence A:

I use TensorFlow to study Deep Learning

Sentence B:

Lee uses Caffe to study Deep Learning

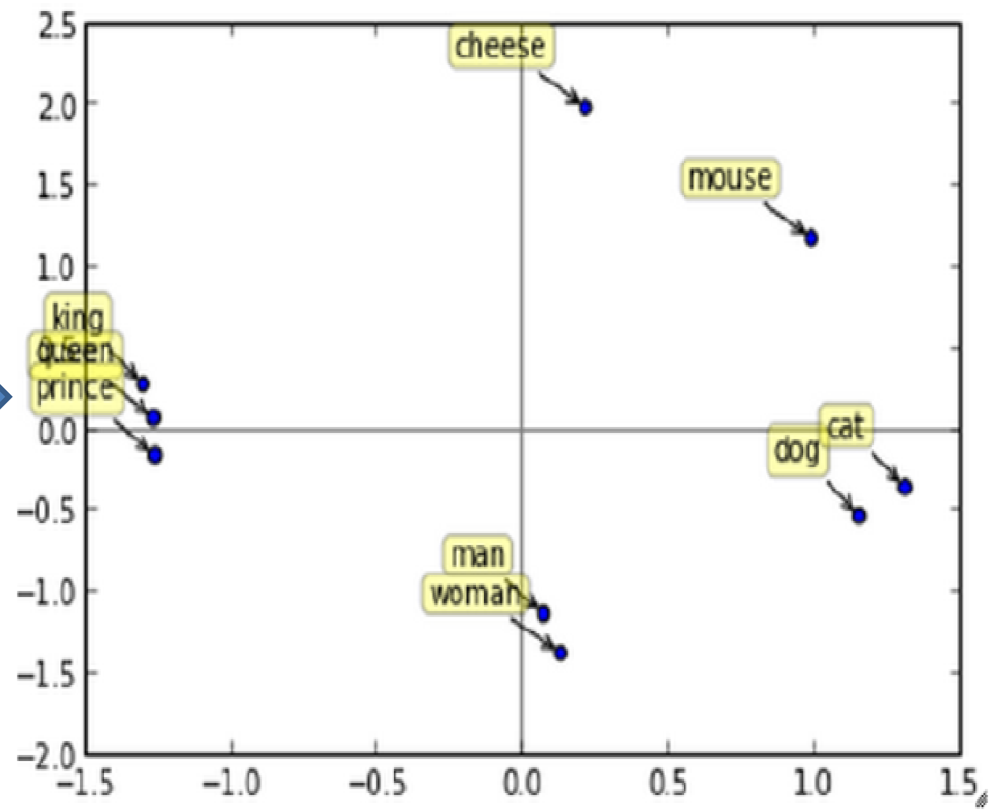
word2vec

Geometrical intuition of word2vec

- Transform the one-hot-encoded vectors into real vectors in lower(latent) dimensions such that similar words have close-by vectors

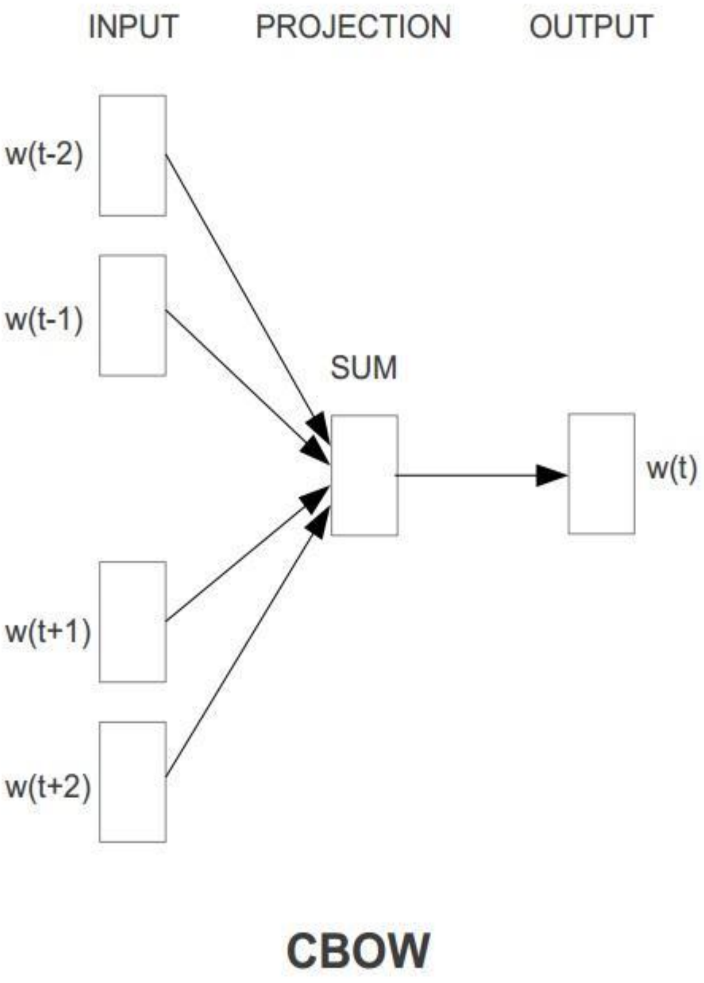
One-hot-encoding of 9 words are represented in 9 dimensions(one per each word).

transform



MLP models for word2vec: CBOW

- Continuous Bag of Word (CBOW) model: use a window of words to predict the middle word



CBOW MLP model: train data

Example: the cat sat on floor and is looking at wall

*Assume Window size = 1

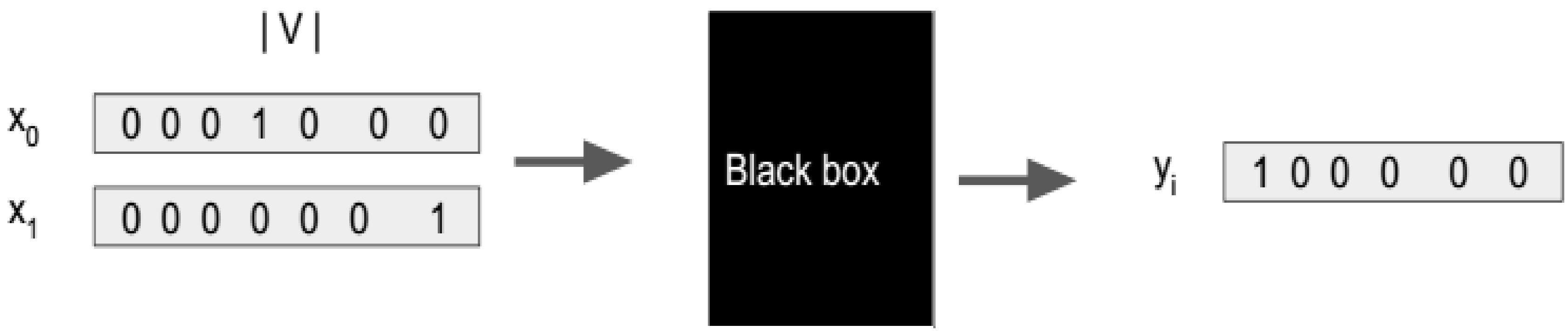
Input	Output
cat	the
the	cat
sat	cat
cat	sat
on	sat
sat	on
floor	on
on	floor
and	floor
…	…

CBOW MLP model: one-hot encoded train data

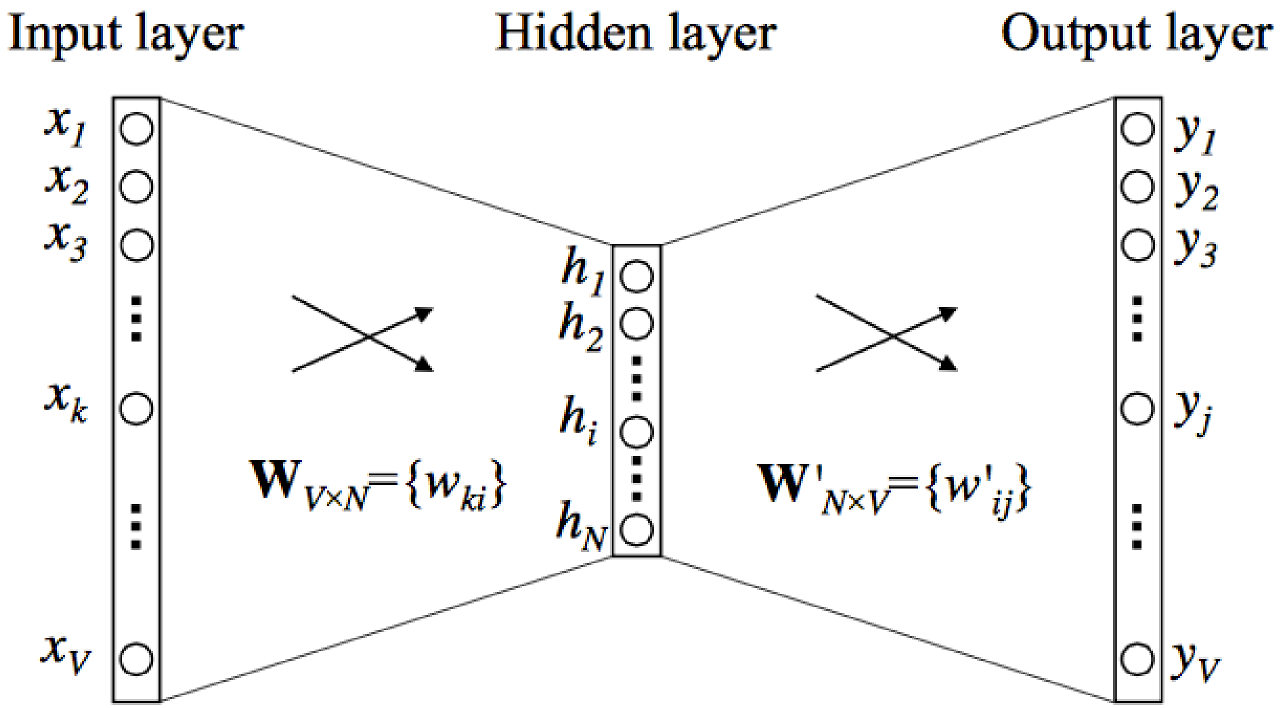
the	0
cat	0
sat	0
on	0
floor	0
and	0
is	0
looking	0
at	0
wall	0

Input	Output
cat [0100000000]	the [1000000000]
the	cat [0100000000]
sat	cat [0100000000]
cat	sat [0010000000]
on	sat [0010000000]
sat	on [0001000000]
floor	on [0001000000]
on	floor [0000100000]
and	floor [0000100000]
…	…

MLP models for word2vec: CBOW

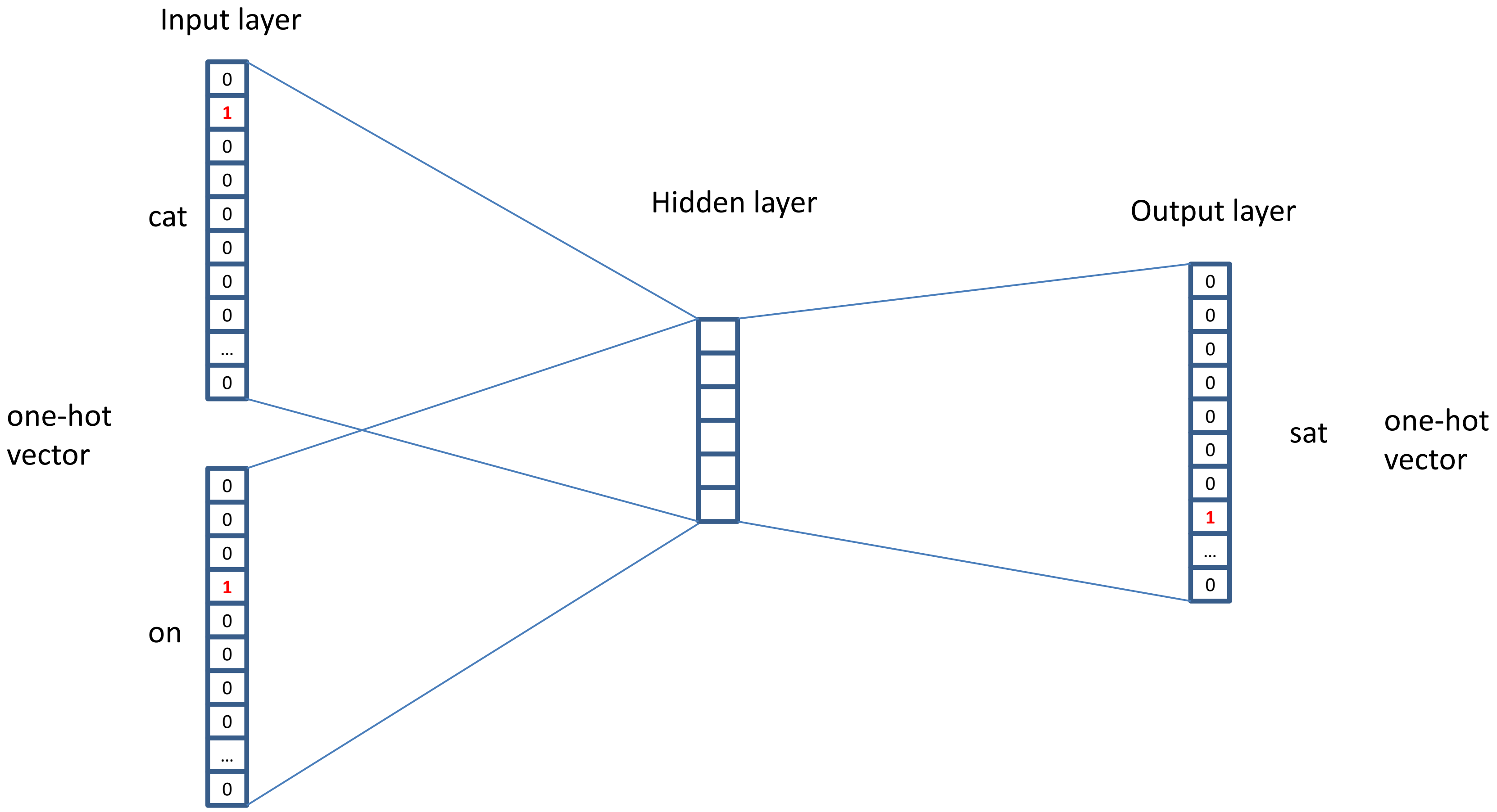


MLP models for word2vec: CBOW

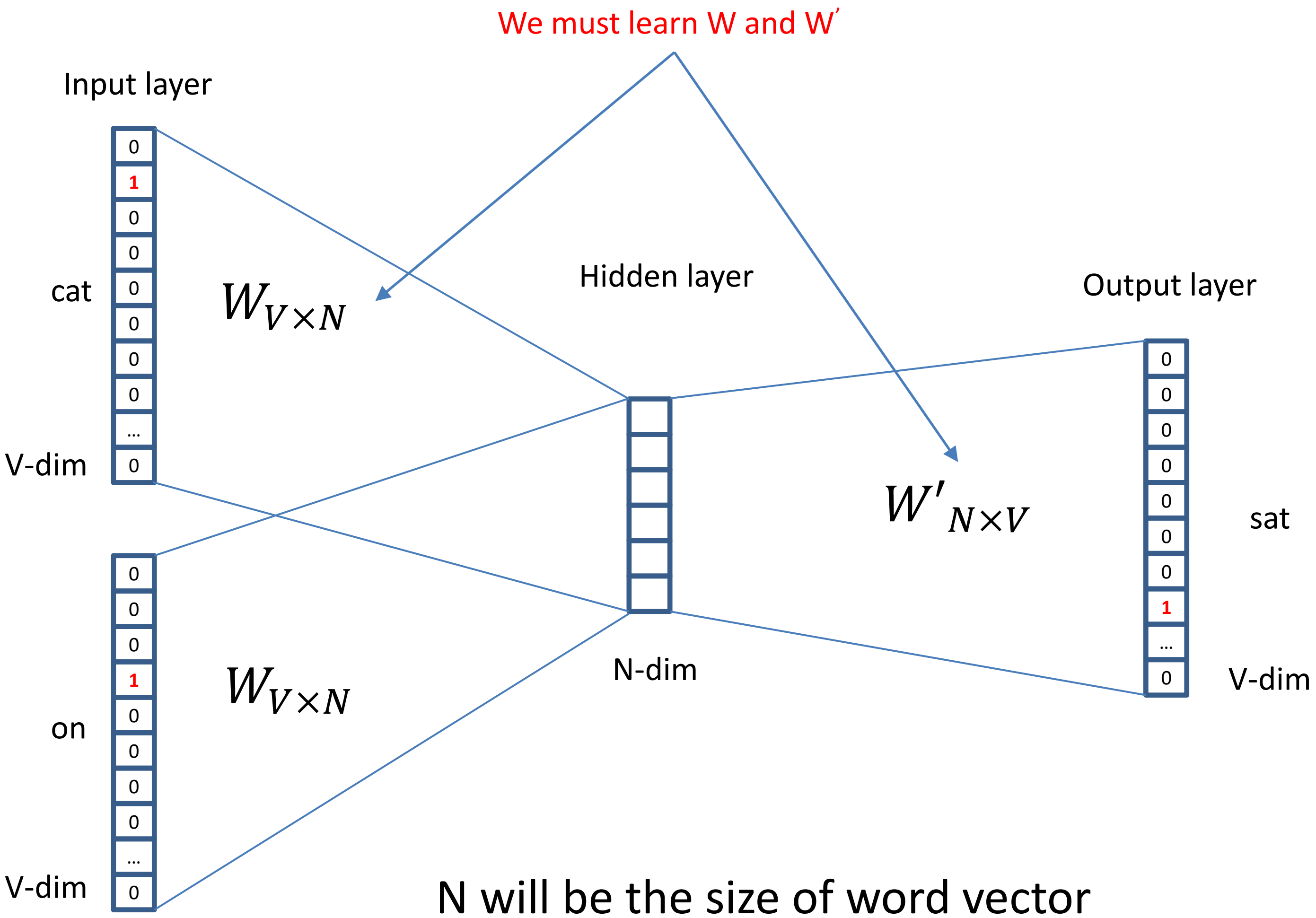


CBOW MLP model: Training Algorithm

Training Algorithm

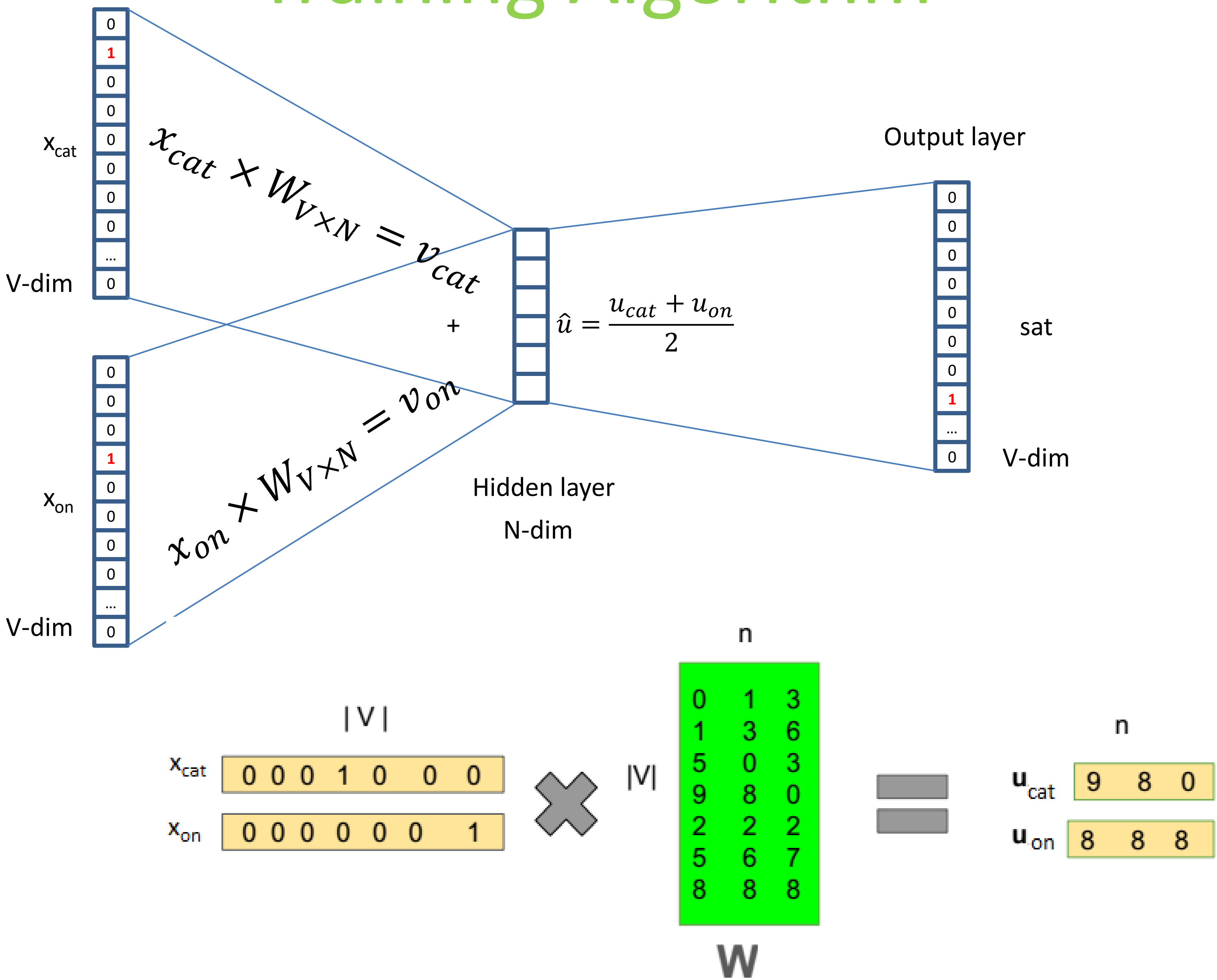


Training Algorithm

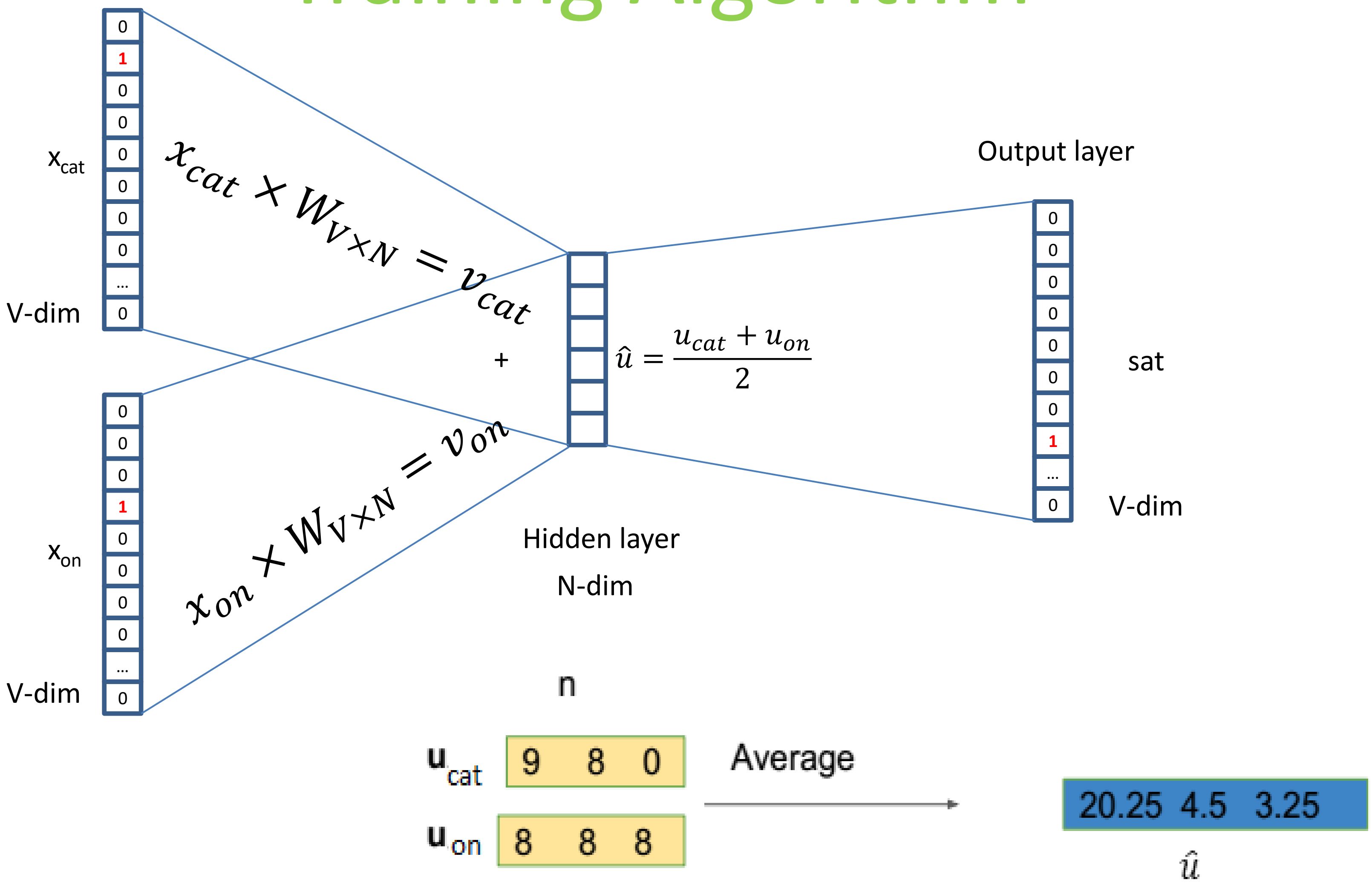


N will be the size of word vector

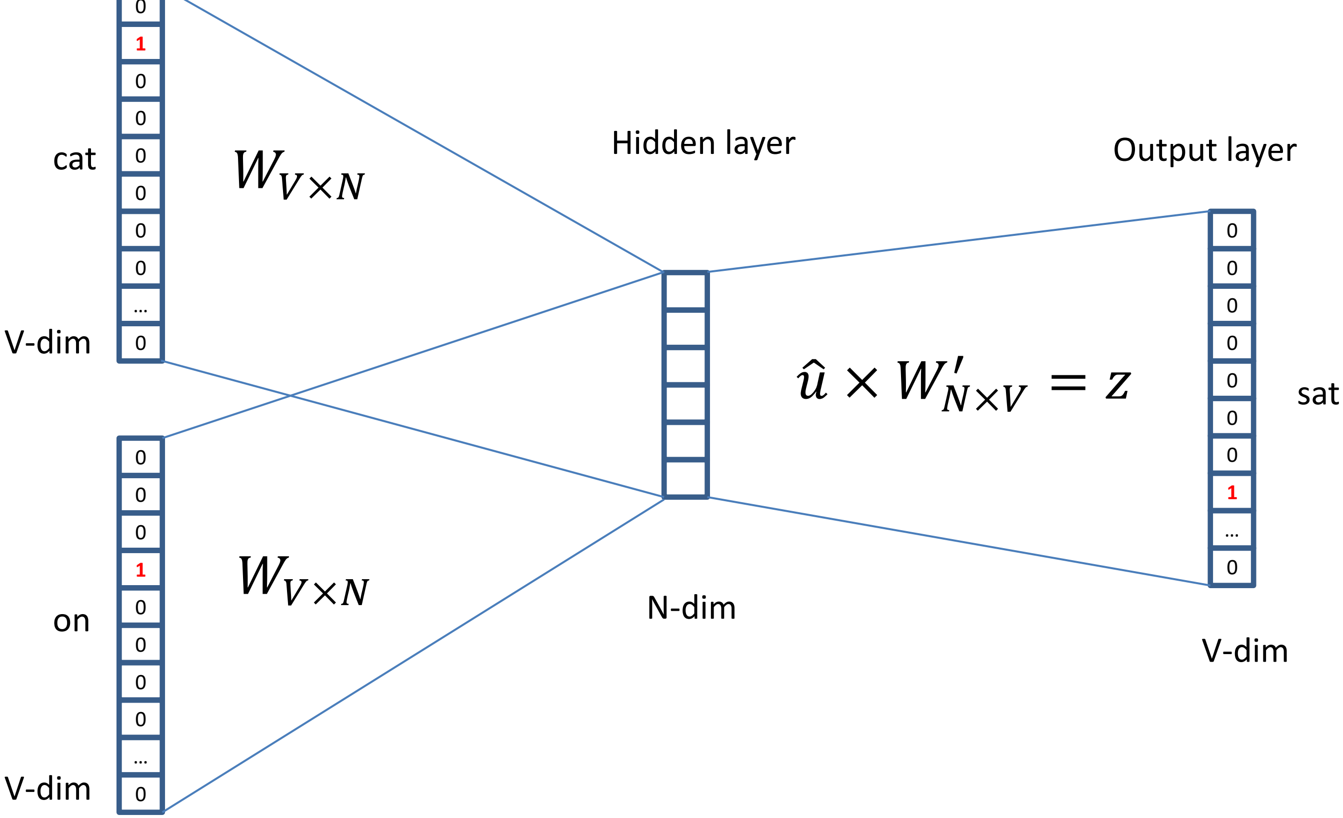
Training Algorithm

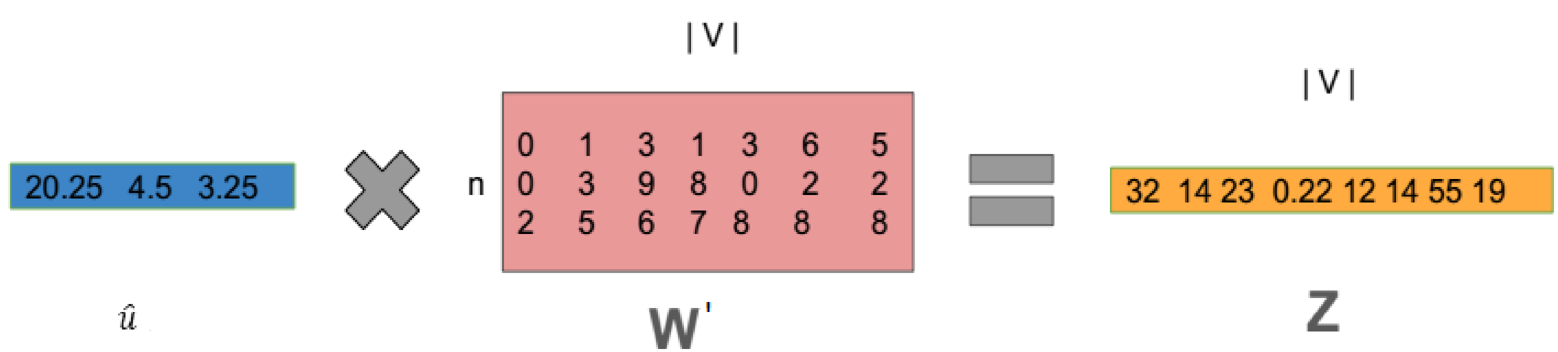


Training Algorithm

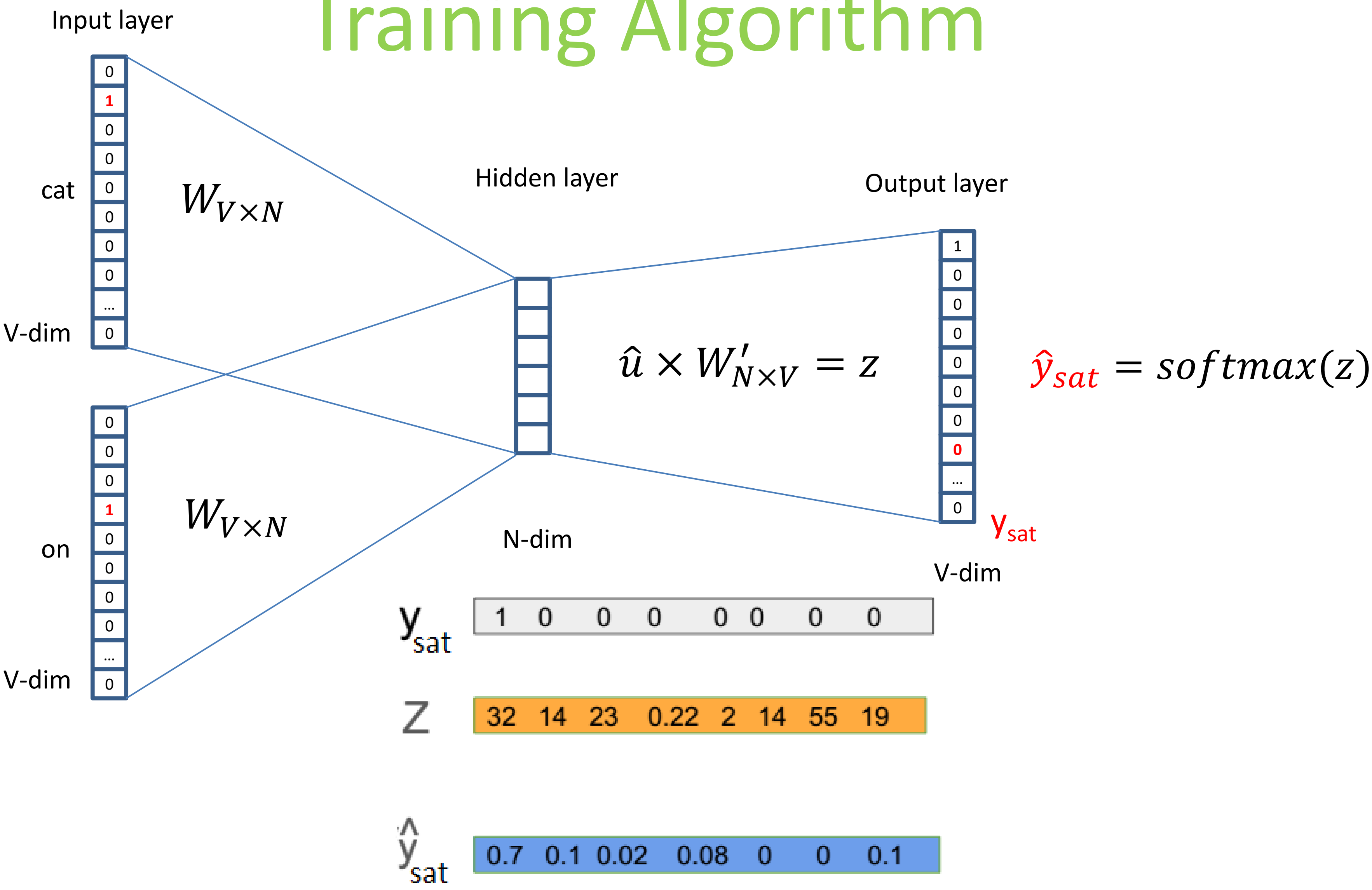


Training Algorithm





Training Algorithm



We would prefer y_{sat} close to $\hat{y}_{sat}$

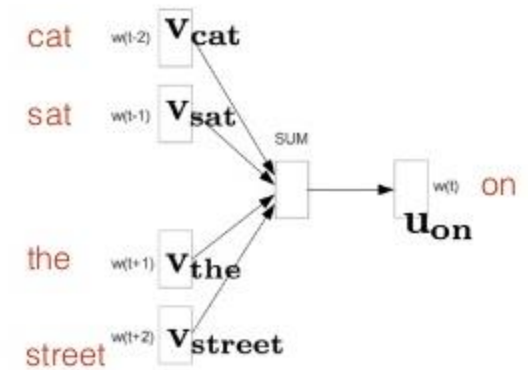
Training Algorithm

- Objective function:

$$\mathbf{L} = \sum_{t=1}^{T} \log P(\mathbf{w_t} | \mathbf{w_{t-c}}, \cdots \mathbf{w_{t+c}})$$

$$P(\mathbf{w_t} | \mathbf{w_{t-c}}, \cdots \mathbf{w_{t+c}}) = \frac{\exp(\mathbf{u_{w_t}} \cdot \boxed{\mathbf{v}})}{\sum_W \exp(\mathbf{u_W} \cdot \mathbf{v})}$$

$$\mathbf{v} = \sum_{t' \neq t, -c \leq t' \leq c} \mathbf{v_{w'_t}}$$



- Gradient Descent Algorithm
  - Learn W and W' to minimize the cost function over all the dataset
  - Using back propagation, update weights in W and W'

Getting those magical wordvectors!!

$$
\begin{array}{cc}
 & n \\
|V| & \begin{bmatrix} 0 & 1 & 3 \\ 1 & 3 & 6 \\ 5 & 0 & 3 \\ 9 & 8 & 0 \\ 2 & 2 & 2 \\ 5 & 6 & 7 \\ 8 & 8 & 8 \end{bmatrix} \\
 & \mathbf{W}
\end{array}
\qquad
\begin{array}{cc}
 & |V| \\
n & \begin{bmatrix} 0 & 1 & 3 & 1 & 3 & 6 & 5 \\ 0 & 3 & 9 & 8 & 0 & 2 & 2 \\ 2 & 5 & 6 & 7 & 8 & 8 & 8 \end{bmatrix} \\
 & \mathbf{W}'
\end{array}
$$

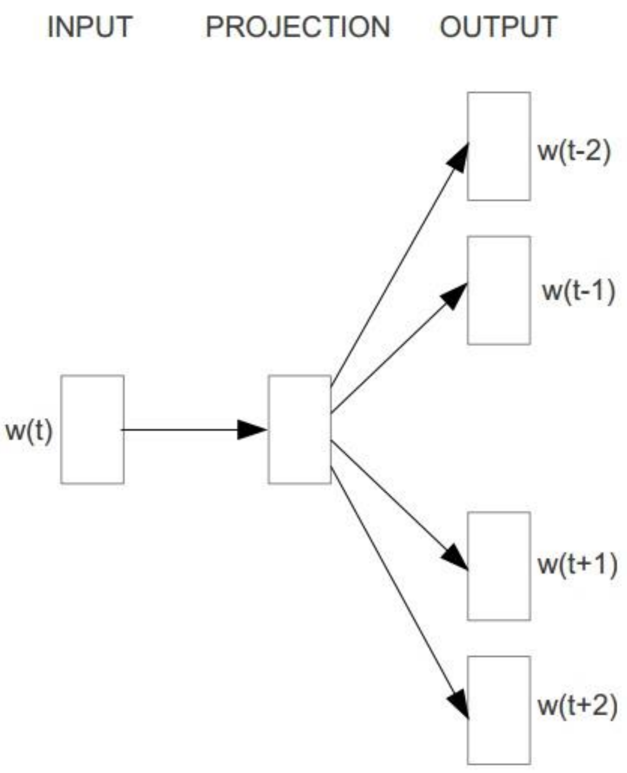
Getting those magical wordvectors!!

After training over a large corpus,

- We can take each row of W(or column of W' or average of both) as wordvector for each word in the vocabulary
- These word vectors contains better semantic and syntactic representation than other dense vectors
- These word vectors performs better for all NLP tasks

MLP models for word2vec: Skipgram

- SkipGram model: use a word to predict the surrounding ones in



Skip-gram

Co-occurence matrix decomposition

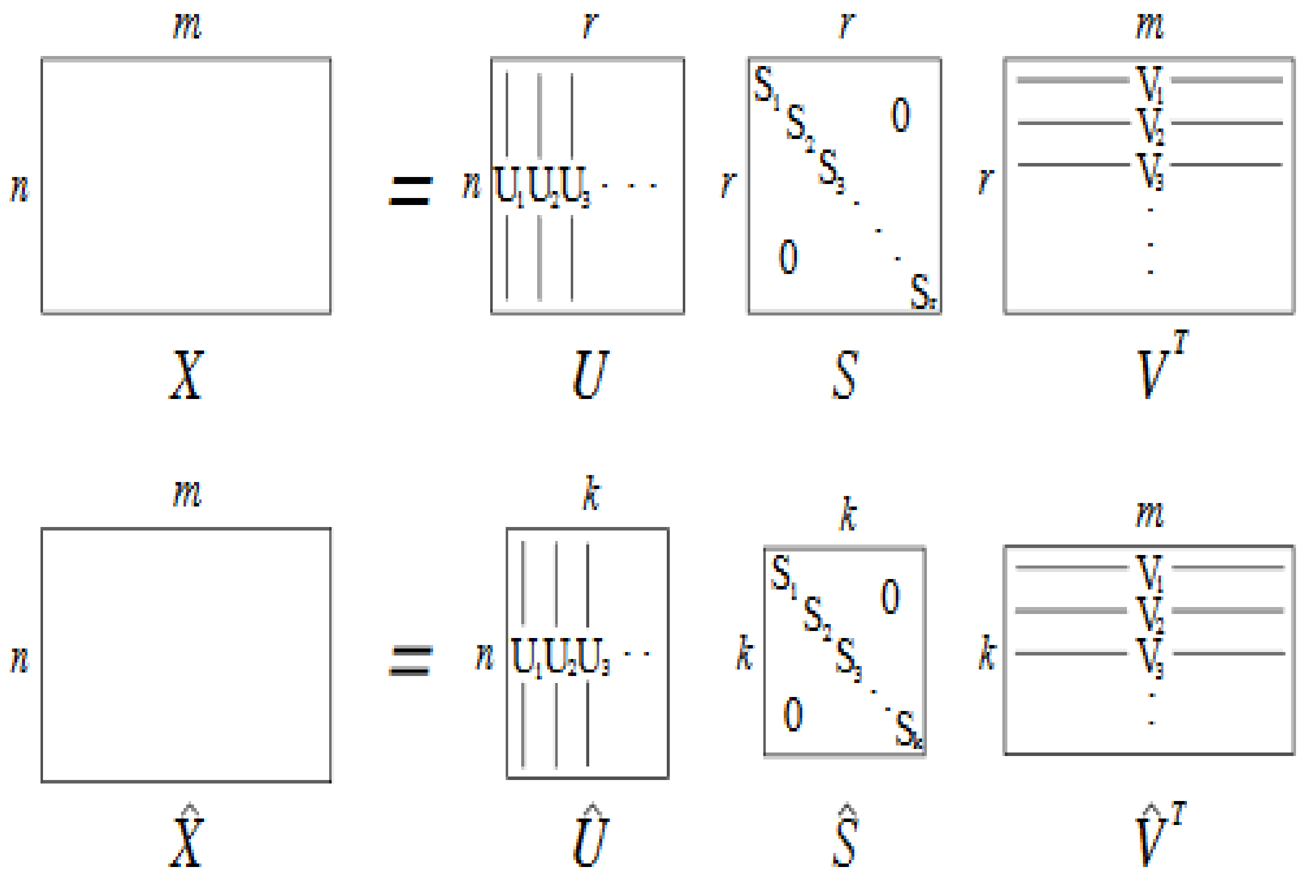
Building a co-occurrence matrix

Corpus = {"I like deep learning"
"I like NLP"
"I enjoy flying"}

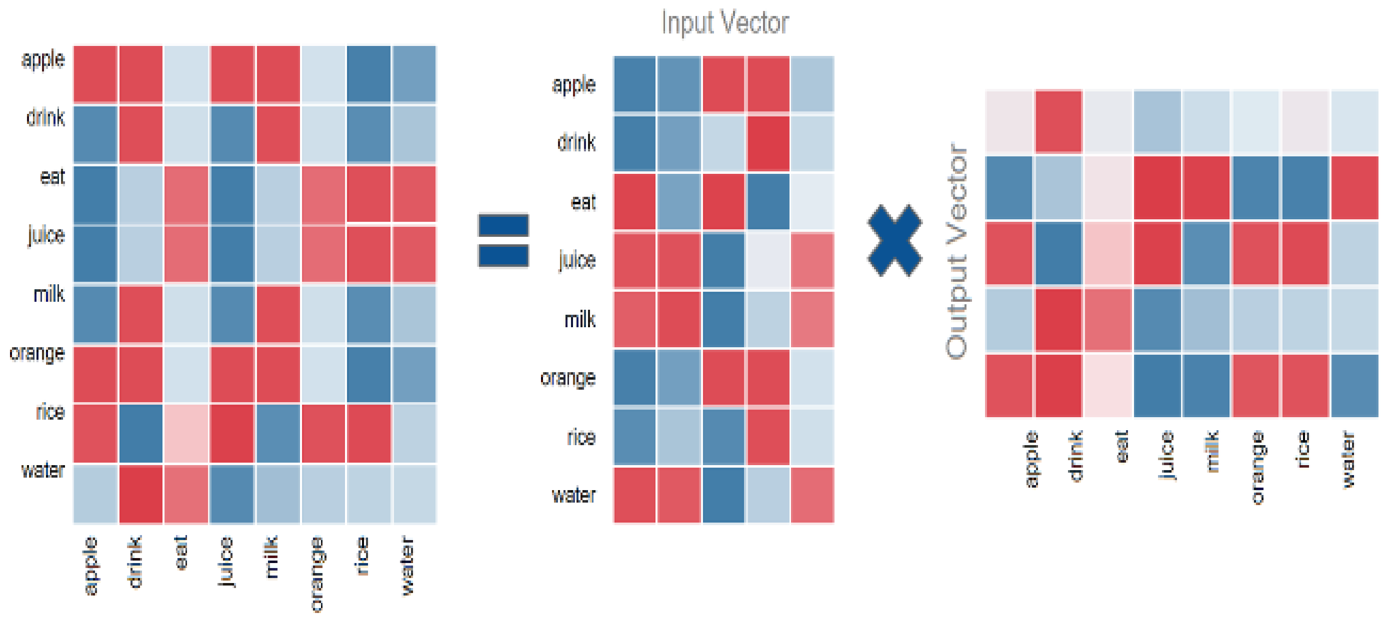
Context = previous word and next word

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

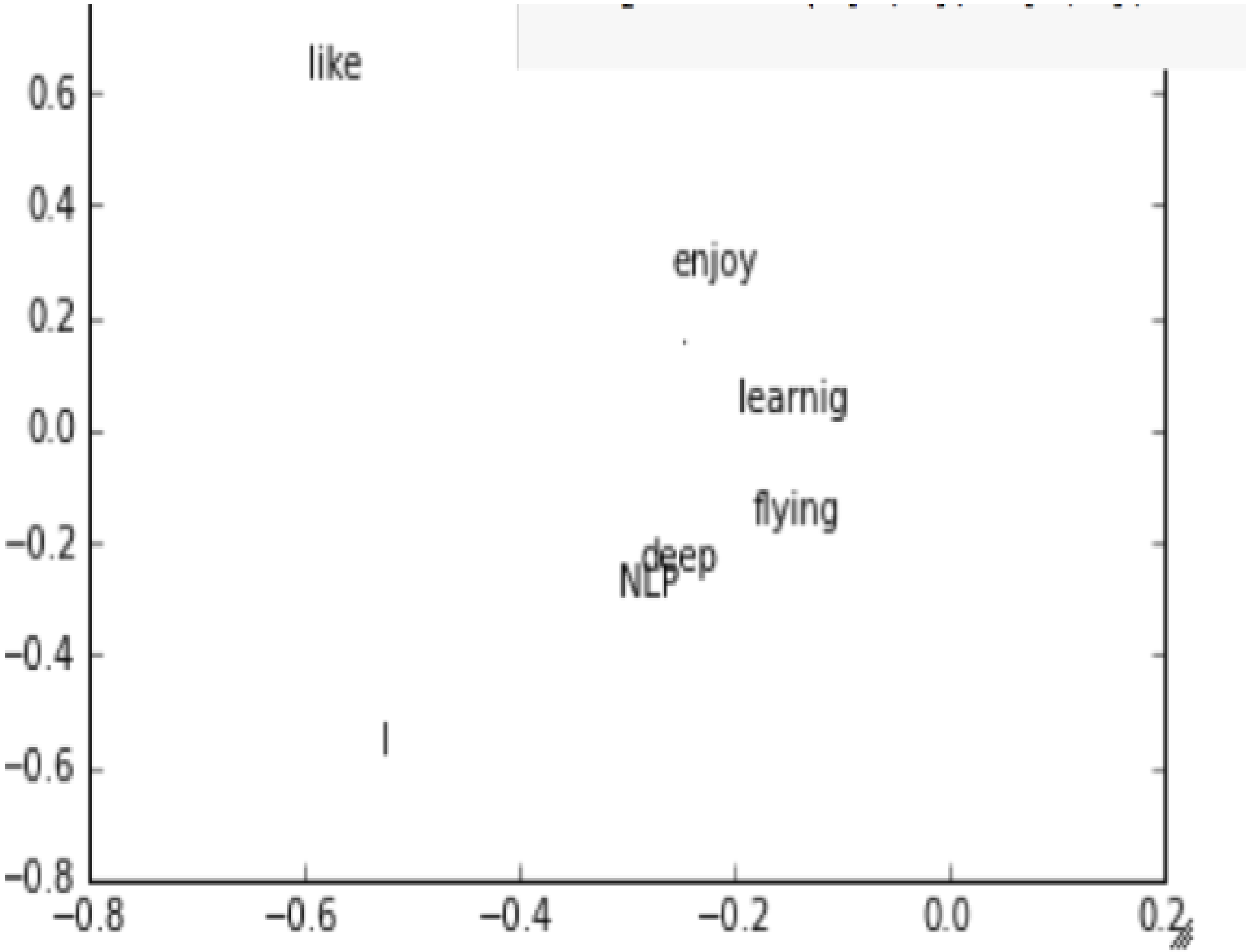
Dimension Reduction using Singular Value Decomposition



Singular Value Decomposition



The problem with this method, is that we may end up with matrices having billions of rows and columns, which makes SVD computationally restrictive.


like
enjoy
learnig
flying
deep
NLP
I
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−0.8
−0.6
−0.4
−0.2
0.0
0.2

Glove

Main Idea

- Uses ratios of co-occurrence probabilities, rather than the co-occurrence probabilities themselves

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Least Squares Problem

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2 ,$$

Weakness of Word Embedding

- Very vulnerable, and not a robust concept
- Can take a long time to train
- Non-uniform results
- Hard to understand and visualize