DEEP LEARNING
(CS436)

BY: Nidhi S. Periwal,
Teaching Assistant,
DoCSE, SVNIT, Surat

Application of DL-> NLP

There are a lot of foods but I like meat best. I think that everyone always eats meat and meat is popular food. Meat is sold everywhere. For example: Supermarket, Market. Everybody always buys meat for the big party because it is delicious and cheap. Meat has a lot of proteins, if everyone eats a lot of meat they are intelligent and strong that's the reason. Why everyone eats meat. I like meat but sometimes I don't eat meat because I have to change different food each day otherwise. I will become fatter if I always eat only meat. However, I like meat best.

Bag-of-words

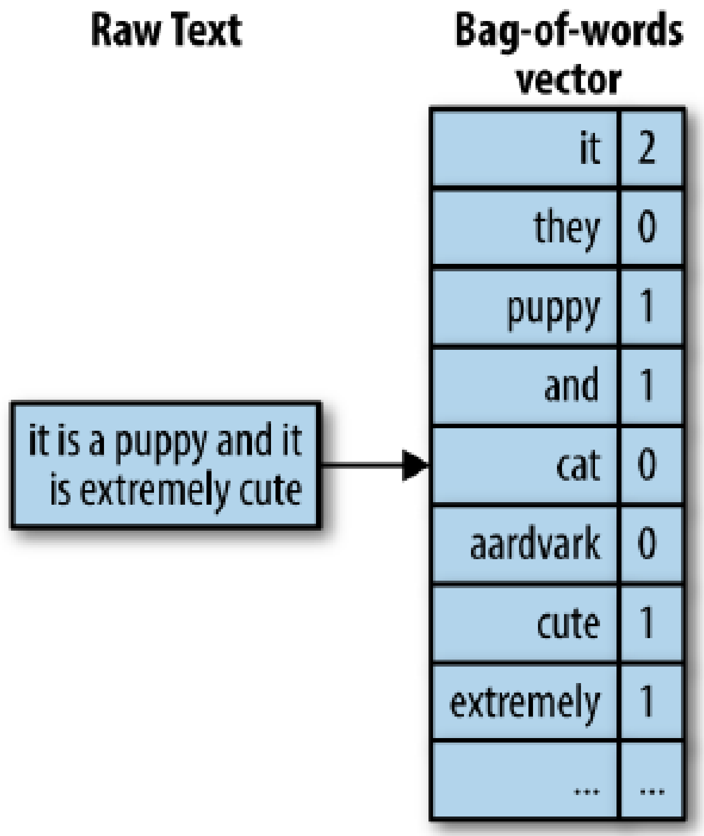


Fig. Turning raw text into a bag-of-words representation

Bag-of-Words

- *EG:*
  - Similarly for other lines:
  - "it was the worst of times" = [1, 1, 1, 0, 1, 1, 1, 0, 0, 0]
  - "it was the age of wisdom" = [1, 1, 1, 0, 1, 0, 0, 1, 1, 0]
  - "it was the age of foolishness" = [1, 1, 1, 0, 1, 0, 0, 1, 0, 1]
  - These docs can also be represented as **document-term** matrix:

	it	was	the	best	of	times	worst	age	wisdom	foolishness
Doc 1	1	1	1	0	1	1	1	0	0	0
Doc 2	1	1	1	0	1	0	0	1	1	0
Doc 3	1	1	1	0	1	0	0	1	0	1

Table 1 : An example document-term matrix

Tf-Idf

- Tf-idf is a simple twist on the bag-of-words approach. It stands for ***term frequency–inverse document frequency.***
- It is a statistical measure **used to evaluate how important a word is to a document** in a collection or corpus.
- The tf-idf weight is composed by two terms:
  - **TF: Term Frequency**
  - **IDF: Inverse Document Frequency**,

Tf-Idf

- **TF: Term Frequency**
  - How frequently a term occurs in a document.
  - **TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).**
- **IDF: Inverse Document Frequency**
  - **measures how important a term** is
  - While computing TF, all terms are considered equally important.
  - However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance.
  - Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:
  - **IDF(t) = log(Total number of documents / Number of documents with term t in it).**

Tf-Idf

– Hence, **Tf-Idf = tf*idf**

– Eg:

- Consider a document containing 100 words wherein the word *cat* appears 3 times.
- **tf for *cat* = (3 / 100) = 0.03.**
- Now, assume we have **10 million documents** and the word ***cat* appears in one thousand of these**.
- **idf = log(10,000,000 / 1,000) = 4.**
- **Tf-idf : 0.03 * 4 = 0.12.**

Tf-Idf

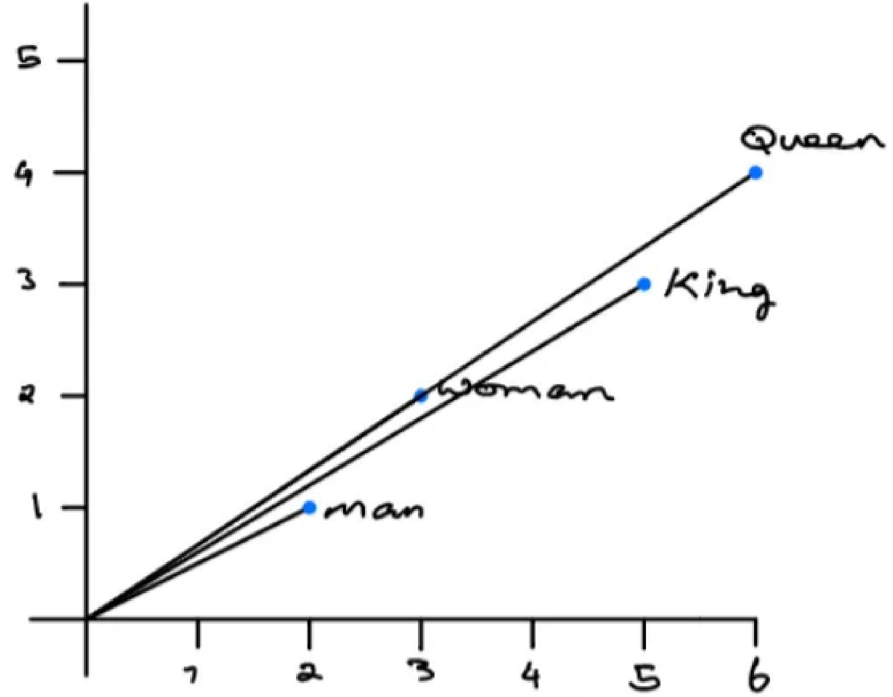
– Common words like **‘is’, ‘the’, ‘a’ etc.** tend to **appear quite frequently in comparison** to the words which are important to a document.
– For example, a document **A on Lionel Messi is going to contain more occurences of the word “Messi”** in comparison to other documents.
– But common words like **“the” etc. are also going to be present in higher frequency** in almost every document
– TF-IDF works by penalising these common words by assigning them lower weights **while giving importance to words like Messi in a particular document.**

Word Embedding

- Word Embedding is a technique where **individual words are transformed into a numerical representation of the word (a vector).**
- Where each word is mapped to one vector, this vector is then learned in a way which resembles a neural network.
- **The vectors try to capture various characteristics of that word with regard to the overall text.**
- **These characteristics can include the semantic relationship of the word, definitions, context, etc.**
- **With these numerical representations we can identify similarity or dissimilarity between words.**

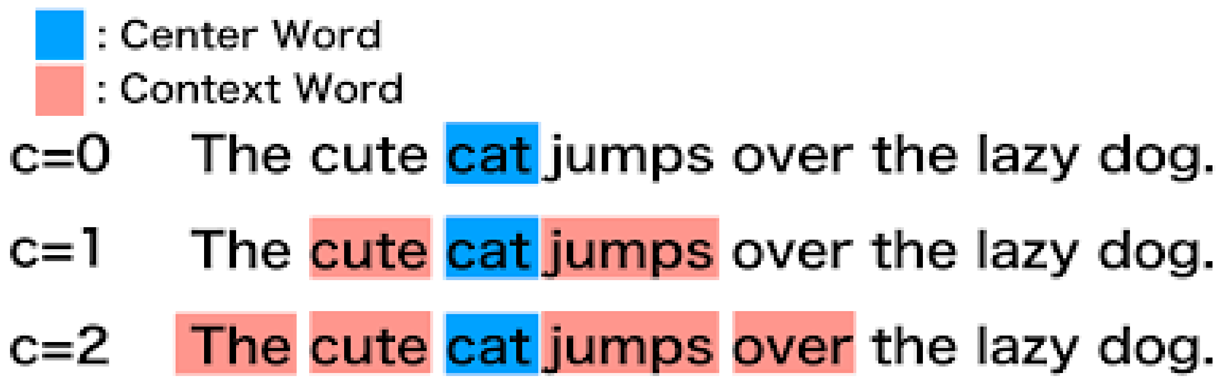
- Eg: 2 dimensional embedding vector of "king" - the 2 dimensional embedding vector of "man" + the 2 dimensional embedding vector of "woman" yielded a vector which is very close to the embedding vector of "queen".

```
King    -    Man    +    Woman    =    Queen
[5,3]   -    [2,1]  +    [3, 2]   =    [6,4]
```



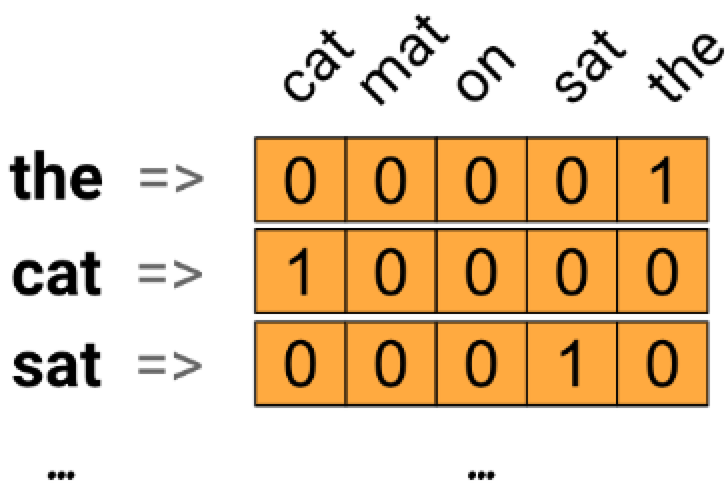
Word Embedding

- The word, which has been focused on to learn its representation is called **center word** and the words around it are called **context words**
- **Context can be** anything – a surrounding n-gram, a randomly sampled set of words from a fixed size window around the word



One hot encoding

- Simplest embedding of text data
- One hot encoding is a vector representation of words in a “vocabulary



Word2Vec

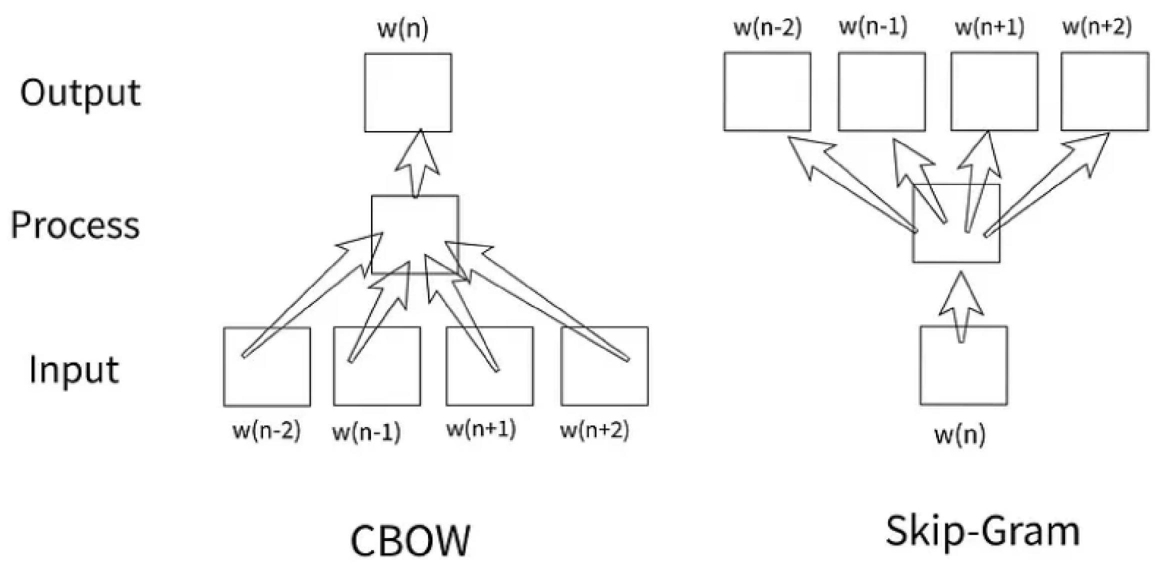
- **Word2Vec** model is used for **learning vector representations of words called "word embeddings**"
- Word2vec is algorithm for **learning a word embedding from a text corpus.**
- It learns the **similarity of word meaning from simple information**.
- The idea is based on the assumption that **the meaning of a word is affected by the words around it.**

- **Word2vec** is a technique for natural language processing (NLP) published in 2013. The word2vec algorithm uses a neural network model to learn word associations from a large corpus of text.
- Word2vec is a group of related models that are used to produce word embeddings. **These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words.**
- **Word2vec takes as its input a large corpus of text and produces a vector space**, typically of several hundred dimensions, **with each unique word in the corpus being assigned a corresponding vector in the space.**
- Unlabeled data is trained via artificial neural networks to create the Word2Vec model that generates word vectors.

- Word2vec can utilize either of two model architectures to produce these distributed representations of words: continuous bag-of-words (CBOW) or continuous skip-gram.

Word2Vec

- Two basic neural network models:
  - Continuous Bag of Word (CBOW): **use a window of word to predict the middle word**
  - Skip-gram (SG): use a word to predict the surrounding ones in window.

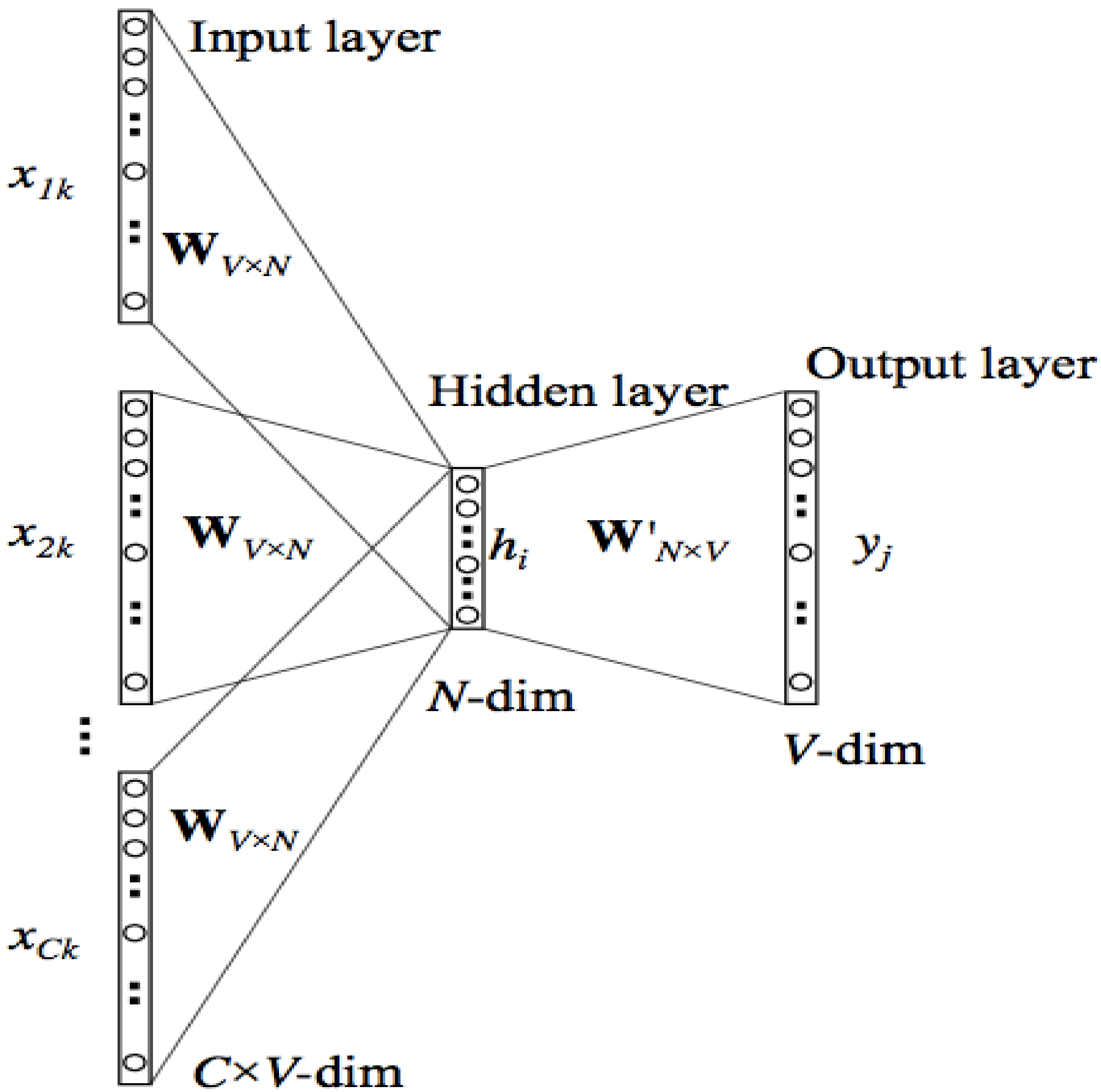


CBOW

- It takes the **context of each word as the input** and tries to predict the word corresponding to the context.
- The CBOW model takes a **window of surrounding words** as input and tries to predict the target word in the center of the window.
- The model is trained on a large text dataset and learns to predict the target word based on the patterns it observes in the input data.
- Here, we try to predict centre word by summing vectors of surrounding words.

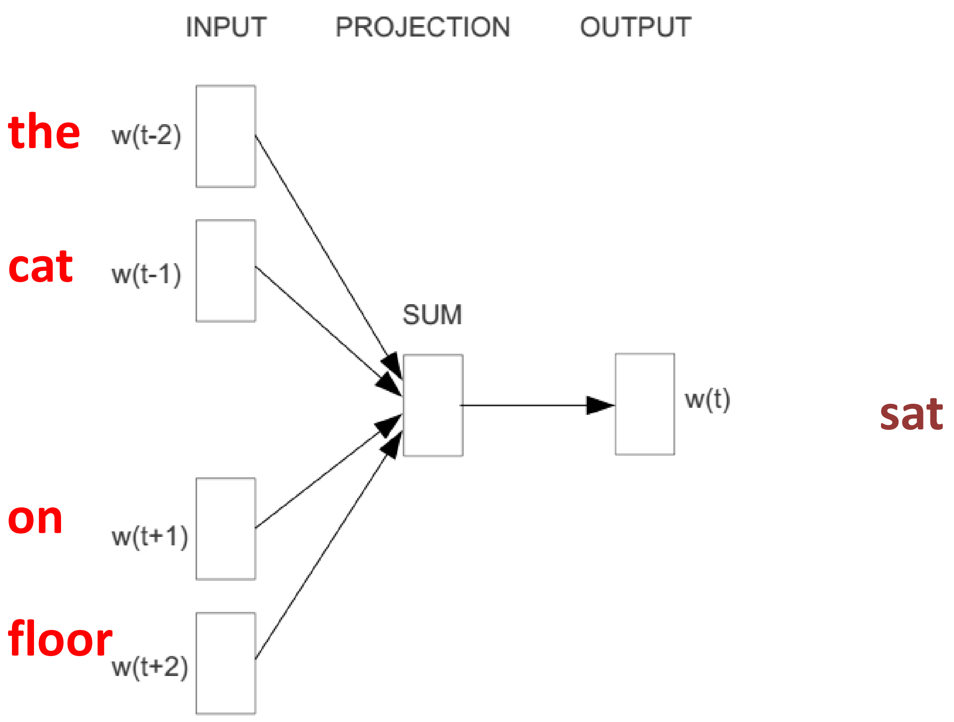
CBOW

- The architecture for multiple context words is shown in fig below:



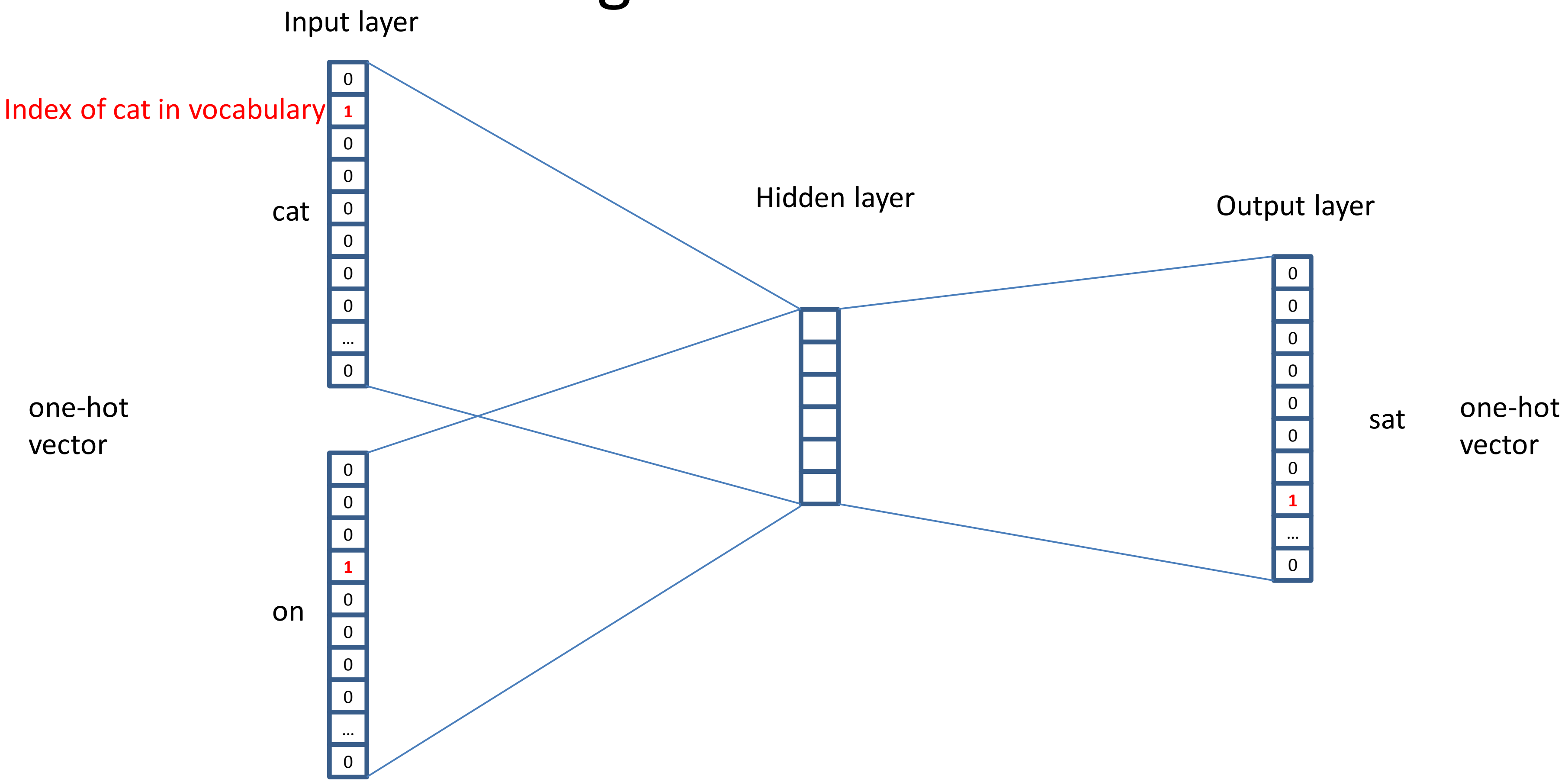
Word2Vec

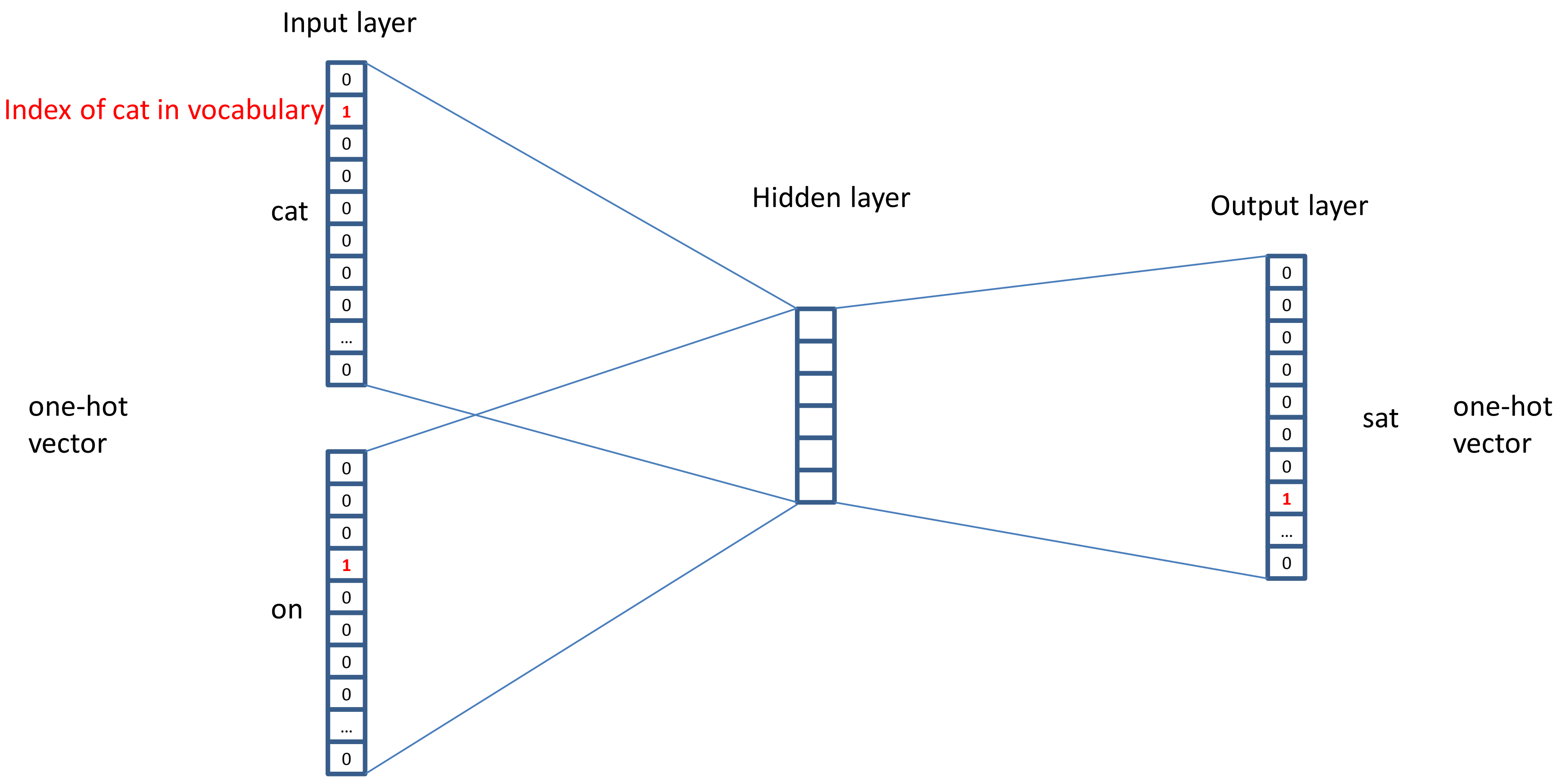
- Continuous Bag of Word (CBOW)
  - E.g. "The cat sat on floor" : Window size = 2



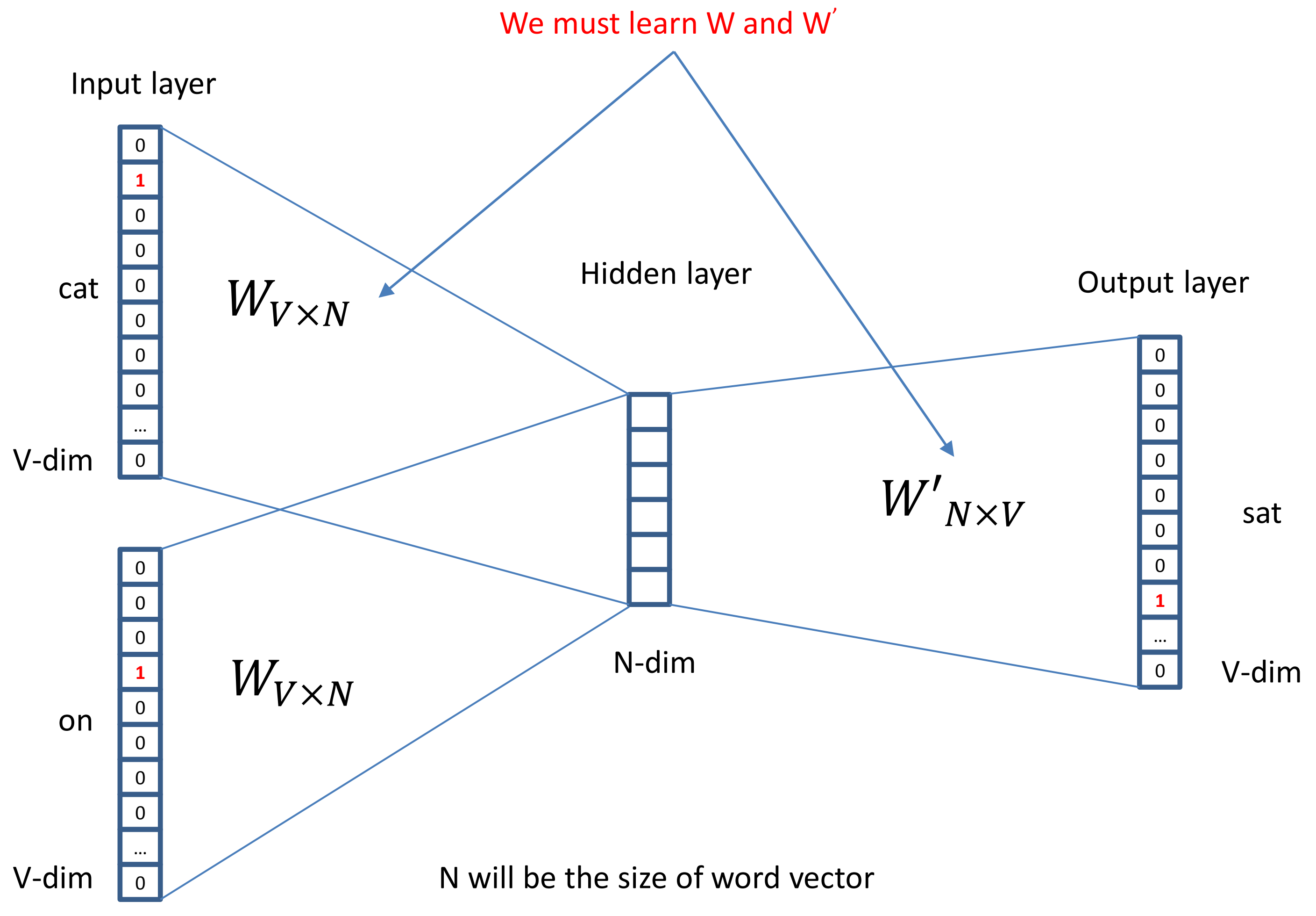
Word2Vec

- Continuous Bag of Word

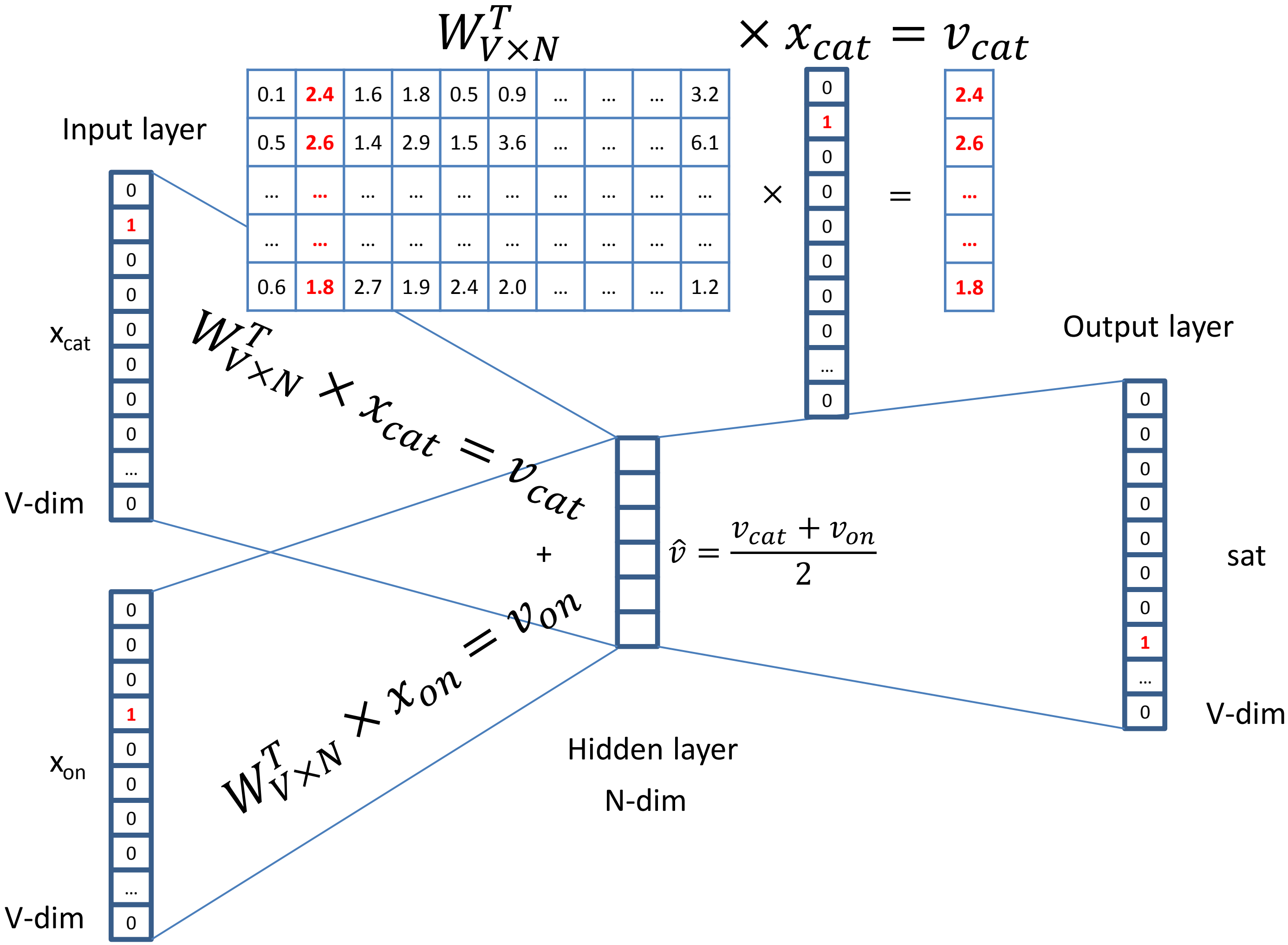



Input layer
Index of cat in vocabulary
cat
0
1
0
0
0
0
0
0
...
0
one-hot
vector
on
0
0
0
1
0
0
0
0
...
0
Hidden layer
Output layer
0
0
0
0
0
0
0
1
...
0
sat
one-hot
vector

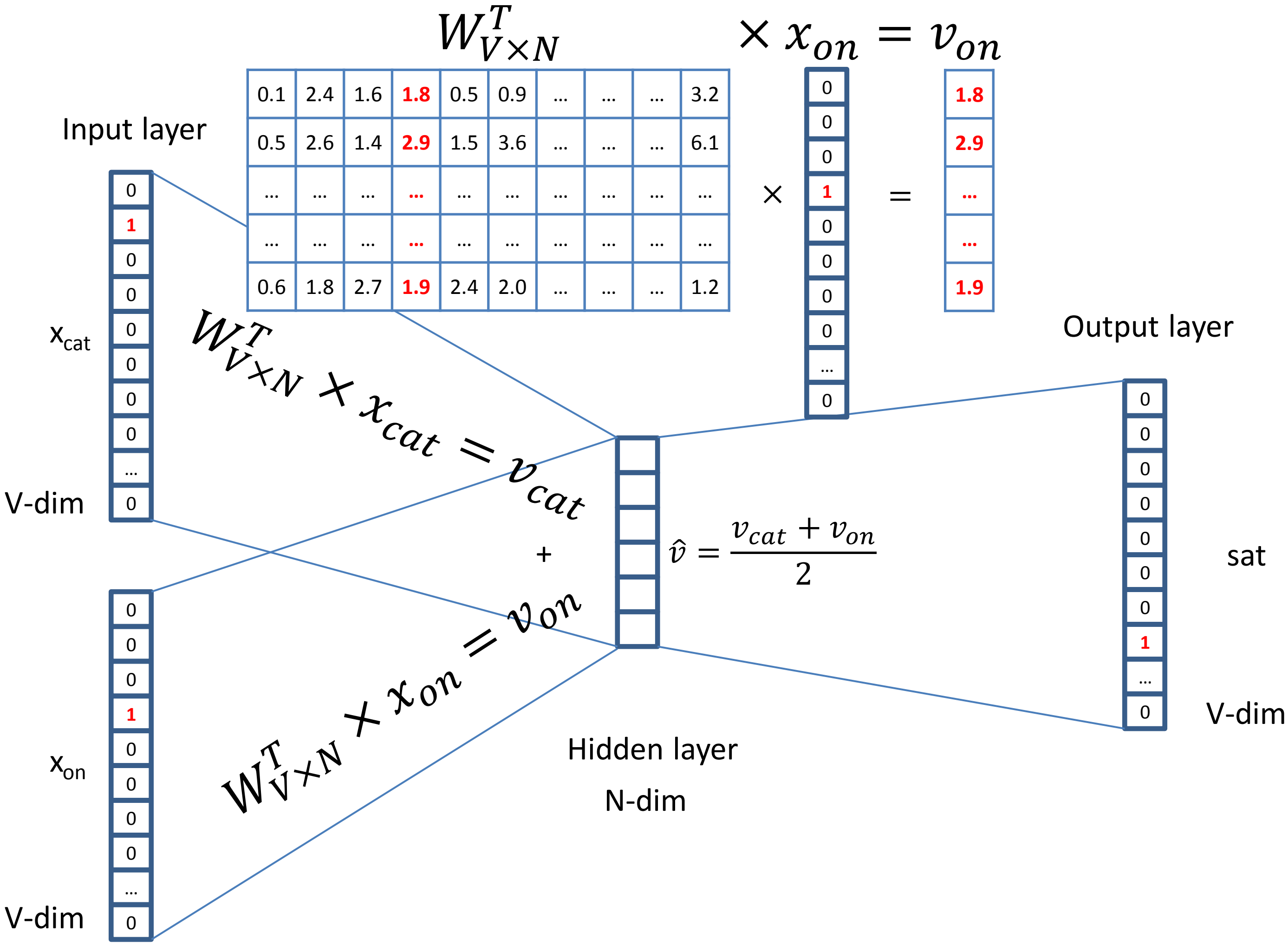
N is the number of dimensions we choose to represent our word in.
Also, **N is the number of neurons in the hidden layer**.
V can be the Vocabulary



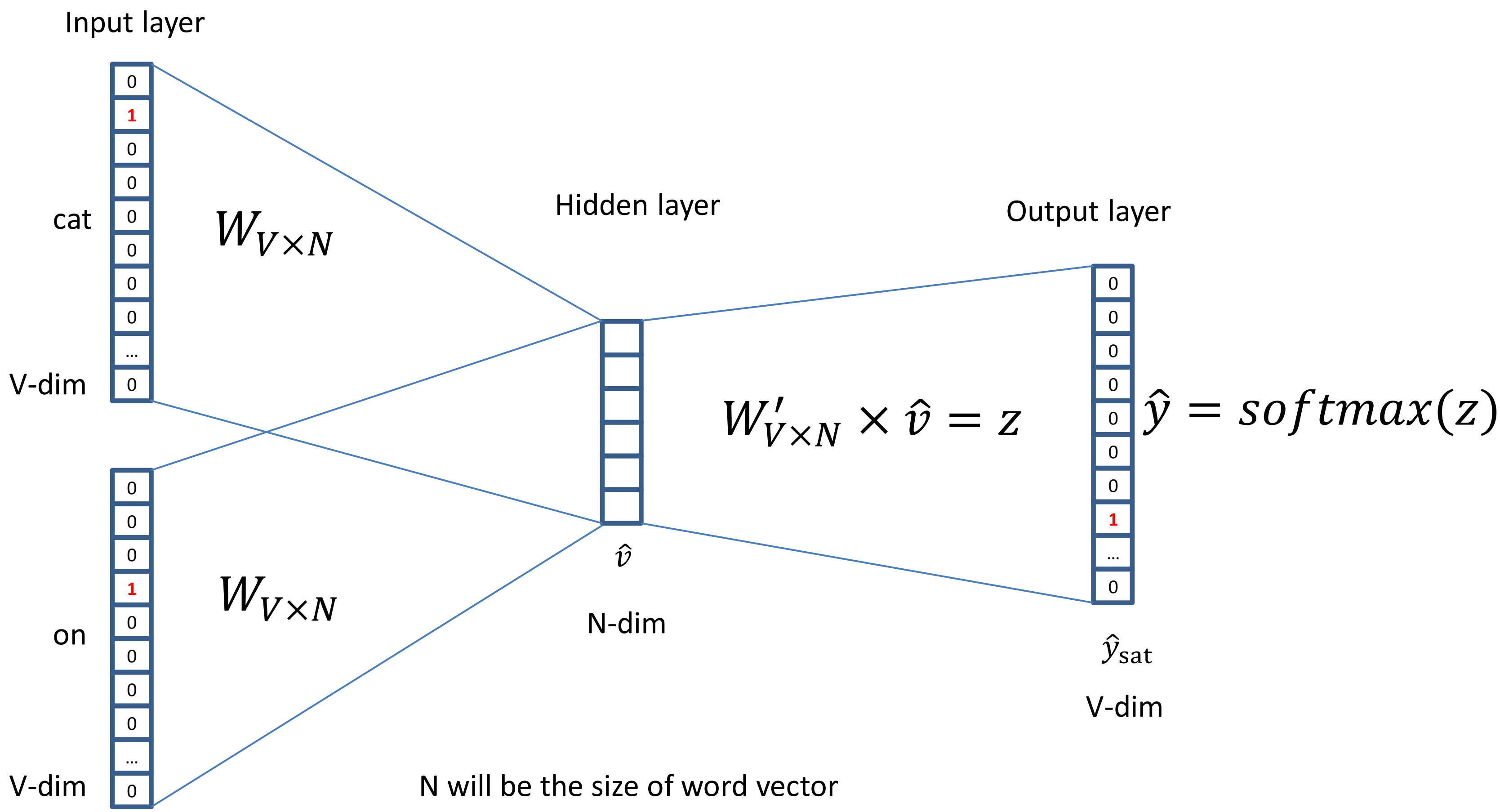
Input-Hidden layer matrix size =[V X N] ,
hidden-Output layer matrix size =[N X V]



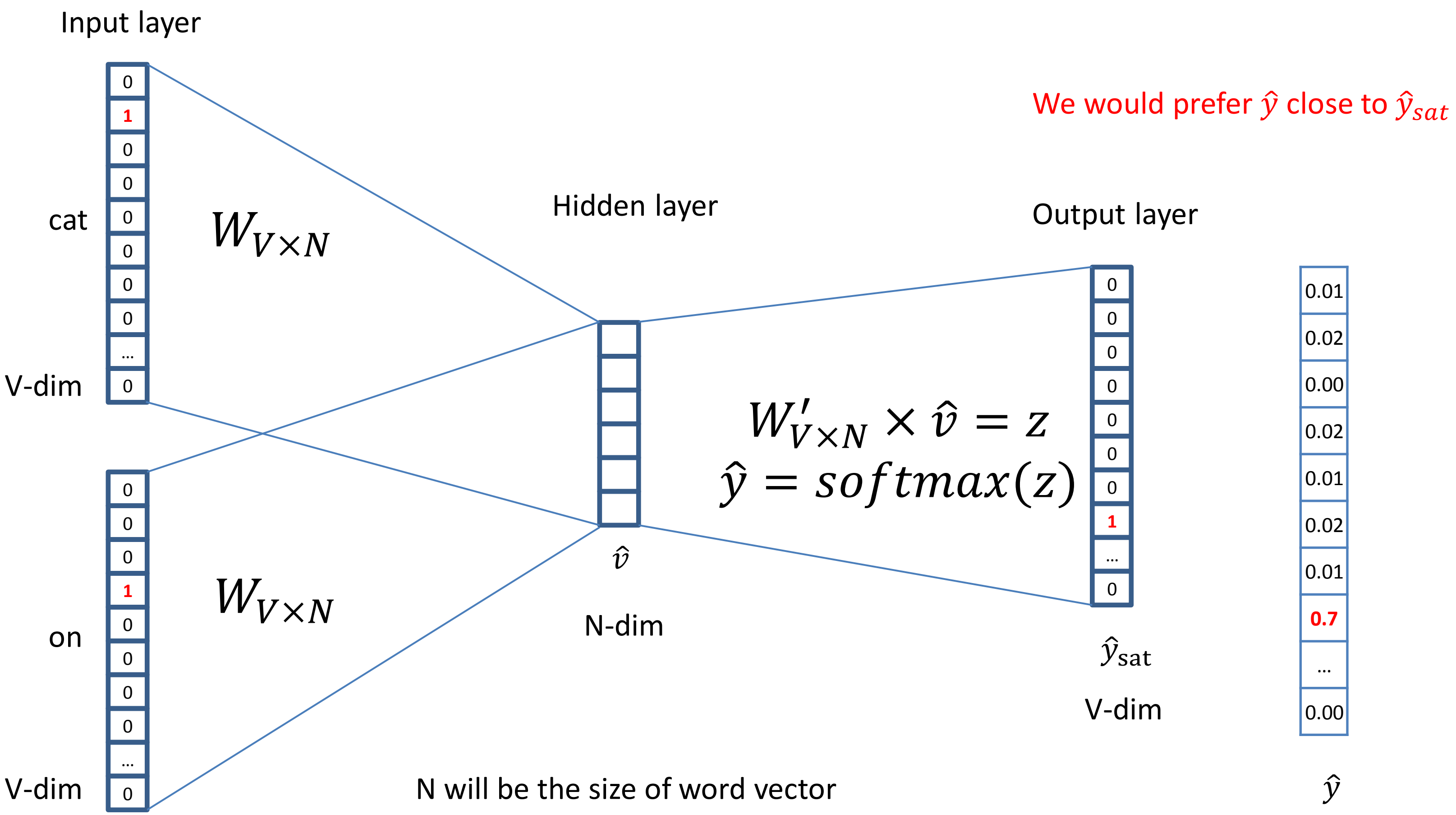
$W^T_{V\times N}$
$\times x_{cat} = v_{cat}$
Input layer
Output layer
Hidden layer
N-dim
x_{cat}
V-dim
x_{on}
V-dim
sat
V-dim
$W^T_{V\times N} \times x_{cat} = v_{cat}$
$W^T_{V\times N} \times x_{on} = v_{on}$
$\hat{v} = \frac{v_{cat} + v_{on}}{2}$



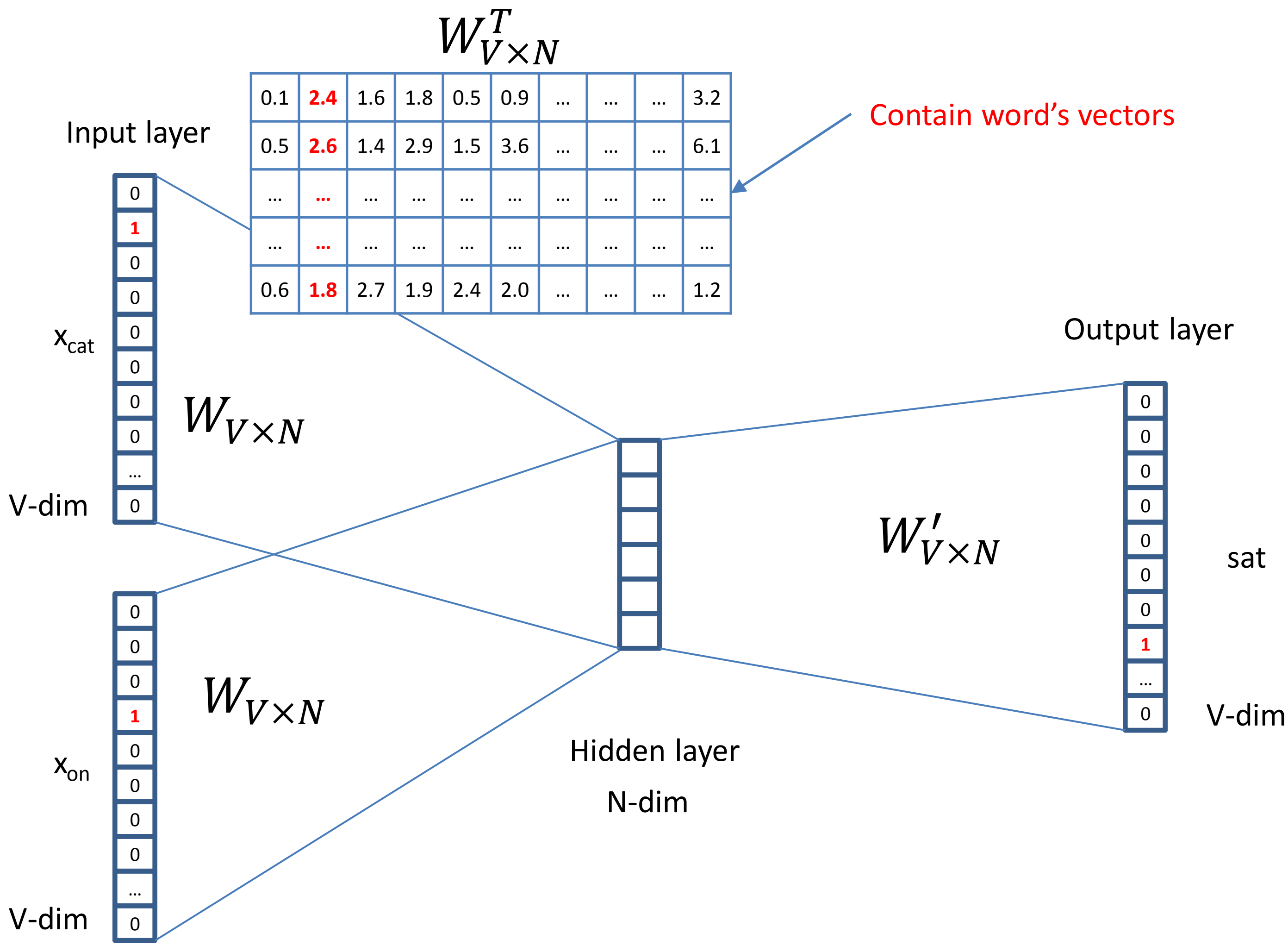
$W^T_{V\times N}$
$\times x_{on} = v_{on}$
0.1 2.4 1.6 1.8 0.5 0.9 ... ... 3.2
0.5 2.6 1.4 2.9 1.5 3.6 ... ... 6.1
0.6 1.8 2.7 1.9 2.4 2.0 ... ... 1.2
1.8 2.9 ... ... 1.9
Input layer
x_{cat}
V-dim
x_{on}
V-dim
$W^T_{V\times N} \times x_{cat} = v_{cat}$
+
$W^T_{V\times N} \times x_{on} = v_{on}$
$\hat{v}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



N will be the size of word vector



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

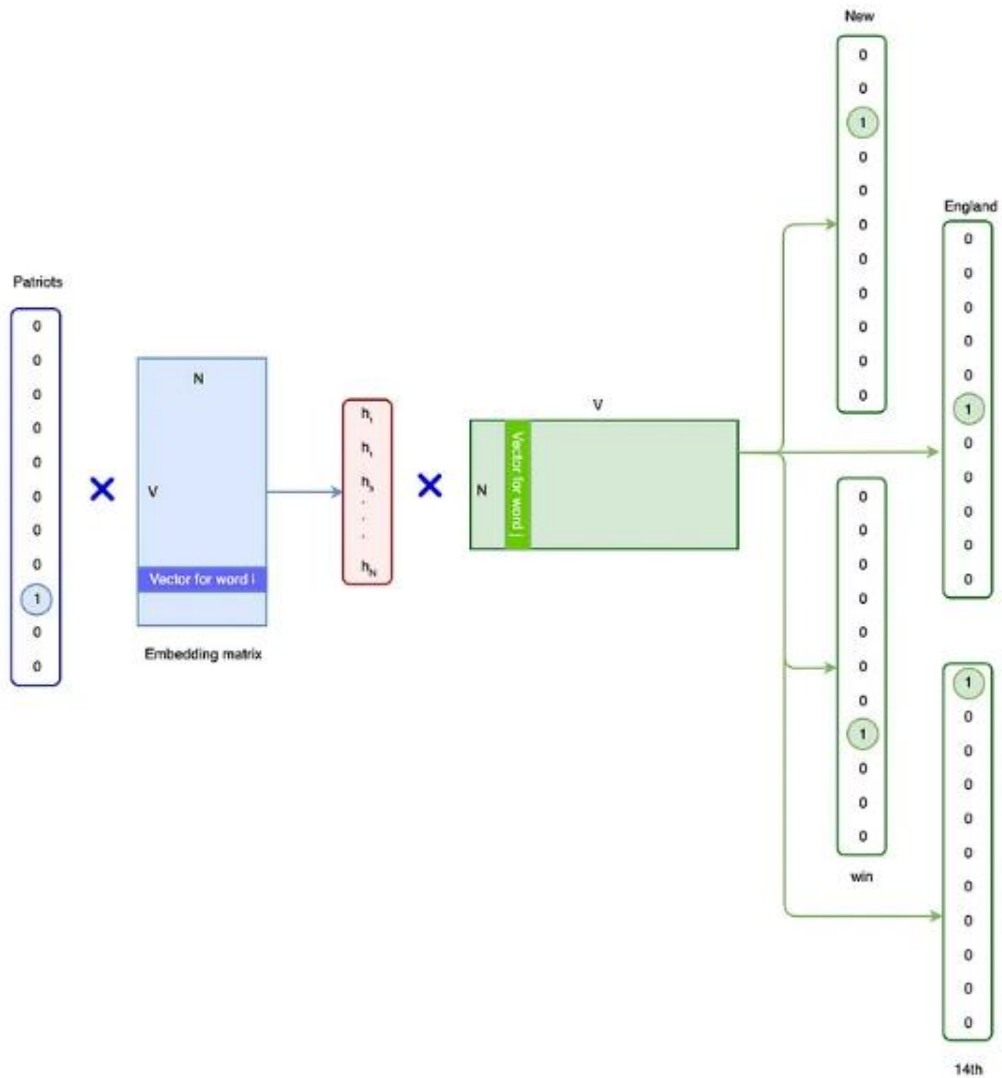


We can consider either W or W' as the word's representation. Or even take the average.

Skip gram Model

- In Skip-gram model, we take a **centre** word and a window of **context (neighbor)** words
- Here, we try to **predict context words** out to **some window size for each centre word**.
- So, our model is going to define a **probability distribution i.e. probability of a word appearing in context given a centre word** and we are going to choose our vector representations to **maximize the probability.**

- **Example:** *New England Patriots win 14th straight regular-season game at home.*



- The log-likelihood for the predicted words given the target word *t* ("Patriots") will be:

A sequence of training words (w_1, w_2, w_3, ...)

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

Surrounding words for word t

New England **Patriots** win 14th straight regular-season game at home.

Patriots → New
Patriots → England
Patriots → win
Patriots → 14th

GloVe

- GloVe stands for global vectors for word representation.
- It is an unsupervised learning algorithm developed by Stanford for generating word embeddings by aggregating global word-word co-occurrence matrix from a corpus.

- In GloVe, a word co-occurrence matrix is generated, where rows represent the words and columns represent the context.
- When co-occurrence frequencies are extracted at a sentence level, every word is said to be in the context of another word in the same sentence, and the corpus can be represented in the following matrix form.

1. I love chemistry.
2. I love maths.
3. I tolerate biology.

$$
= \begin{array}{c c}
 & \begin{array}{cccccc} I & love & Chemistry & Maths & tolerate & Biology \end{array} \\
\begin{array}{c} I \\ love \\ Chemistry \\ Maths \\ tolerate \\ Biology \end{array} &
\begin{pmatrix}
0 & 2 & 1 & 1 & 1 & 1 \\
2 & 0 & 1 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0
\end{pmatrix}
\end{array}
$$

- X represents the matrix of the co-occurrence frequencies of words in the corpus. Each value in X is interpreted as how frequently a word co-occurs with its context. Factorization of the co-occurrence matrix results in a low-dimensional matrix, where rows represent words and columns represent features.

- Let the matrix of word-word co-occurrence counts be denoted by X
- whose entries X_{ij} tabulate the number of times word j occurs in the context of word i.
- Let $X_i = \sum_k X_{ik}$ be the number of times any word appears in the context of word i.
- $P_{ij} = P(j|i) = X_{ij}/X_i$ probability that word j appear in the context of word i.

Table 1: Co-occurrence probabilities for target words *ice* and *steam* with selected context words from a 6 billion token corpus. Only in the ratio does noise from non-discriminative words like *water* and *fashion* cancel out, so that large values (much greater than 1) correlate well with properties specific to ice, and small values (much less than 1) correlate well with properties specific of steam.

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

- Let i = ice and j = steam. The relationship of these words can be examined by studying the ratio of their co-occurrence probabilities with various probe words, k.
- For words k related to **ice** but not steam, say k = **solid**, we expect the ratio P_{ik}/P_{jk} will be **large**.
- For words k related to **steam** but not ice, say k = **gas**, the ratio should be **small**.
- For words k like water or fashion, that are either related to both ice and steam, or to neither, the ratio should be close to one.

$w \in \mathbb{R}^d$ are word vectors

probe word

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

co-relations between the word w*i* and w*j*

co-occurrence probabilities for the word w*j* and w*k*

- The objective function (weighted least square) in the GloVe model

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

- Here, V refers to the vocabularies in the corpora and f is a weighting function, which assigns lower weights to rare co- occurences .
- X_{ij} is the co-occurrence matrix for a target word (i) and its context word (j), and w_i, w_j , b_i and b_j are a set of trainable parameters for i and j,
- where w_i and w_j are the embeddings,
- and b_i and b_j are their corresponding biases.
- GloVe learns neural embeddings by minimizing the reconstruction error between co-occurrence statistics predicted by the model and global co-occurrence statistics observed in the training corpus

Thank you!