CS7015 (Deep Learning) : Lecture 10

Learning Vectorial Representations Of Words

Mitesh M. Khapra

Department of Computer Science and Engineering
Indian Institute of Technology Madras

Acknowledgments

- 'word2vec Parameter Learning Explained' by Xin Rong
- 'word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method' by Yoav Goldberg and Omer Levy
- Sebastian Ruder's blogs on word embeddings[a]

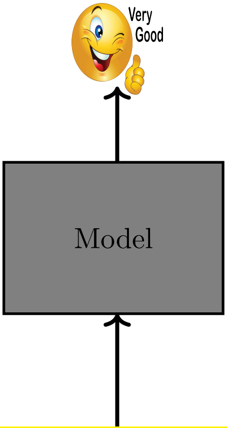
[a] Blog1, Blog2, Blog3

Module 10.1: One-hot representations of words

- Let us start with a very simple motivation for why we are interested in vectorial representations of words

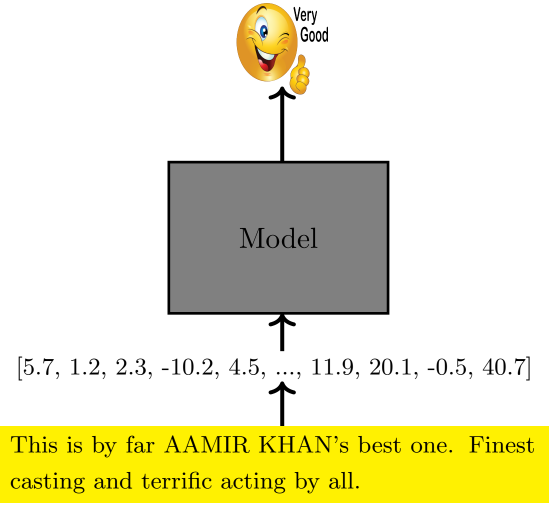
- Let us start with a very simple motivation for why we are interested in vectorial representations of words
- Suppose we are given an input stream of words (sentence, document, etc.) and we are interested in learning some function of it (say, $\hat{y} = sentiments(words)$)

This is by far AAMIR KHAN's best one. Finest casting and terrific acting by all.



This is by far AAMIR KHAN's best one. Finest casting and terrific acting by all.

- Let us start with a very simple motivation for why we are interested in vectorial representations of words
- Suppose we are given an input stream of words (sentence, document, etc.) and we are interested in learning some function of it (say, $\hat{y} = sentiments(words)$)
- Say, we employ a machine learning algorithm (some mathematical model) for learning such a function ($\hat{y} = f(\mathbf{x})$)



- Let us start with a very simple motivation for why we are interested in vectorial representations of words
- Suppose we are given an input stream of words (sentence, document, etc.) and we are interested in learning some function of it (say, $\hat{y} = sentiments(words)$)
- Say, we employ a machine learning algorithm (some mathematical model) for learning such a function ($\hat{y} = f(\mathbf{x})$)
- We first need a way of converting the input stream (or each word in the stream) to a vector $\mathbf{x}$ (a mathematical quantity)

- Given a corpus,

Corpus:

- Human machine interface for computer applications
- User opinion of computer system response time
- User interface management system
- System engineering for improved response time

- Given a corpus,

Corpus:

- Human machine interface for computer applications
- User opinion of computer system response time
- User interface management system
- System engineering for improved response time

- Given a corpus, consider the set V of all unique words across all input streams (*i.e.*, all sentences or documents)

Corpus:

- Human machine interface for computer applications
- User opinion of computer system response time
- User interface management system
- System engineering for improved response time

V = [human,machine, interface, for, computer, applications, user, opinion, of, system, response, time, management, engineering, improved]

- Given a corpus, consider the set V of all unique words across all input streams (*i.e.*, all sentences or documents)
- V is called the **vocabulary** of the corpus (*i.e.*, all sentences or documents)

Corpus:

- Human machine interface for computer applications
- User opinion of computer system response time
- User interface management system
- System engineering for improved response time

V = [human,machine, interface, for, computer, applications, user, opinion, of, system, response, time, management, engineering, improved]

- Given a corpus, consider the set V of all unique words across all input streams (*i.e.*, all sentences or documents)
- V is called the **vocabulary** of the corpus (*i.e.*, all sentences or documents)
- We need a representation for every word in V

Corpus:

- Human machine interface for computer applications
- User opinion of computer system response time
- User interface management system
- System engineering for improved response time

V = [human,machine, interface, for, computer, applications, user, opinion, of, system, response, time, management, engineering, improved]

machine:

0	1	0	...	0	0	0

- Given a corpus, consider the set V of all unique words across all input streams (*i.e.*, all sentences or documents)
- V is called the **vocabulary** of the corpus (*i.e.*, all sentences or documents)
- We need a representation for every word in V
- One very simple way of doing this is to use one-hot vectors of size $|V|$

Corpus:

- Human machine interface for computer applications
- User opinion of computer system response time
- User interface management system
- System engineering for improved response time

V = [human,machine, interface, for, computer, applications, user, opinion, of, system, response, time, management, engineering, improved]

machine:

0	1	0	...	0	0	0

- Given a corpus, consider the set V of all unique words across all input streams (*i.e.*, all sentences or documents)
- V is called the **vocabulary** of the corpus (*i.e.*, all sentences or documents)
- We need a representation for every word in V
- One very simple way of doing this is to use one-hot vectors of size $|V|$
- The representation of the i-th word will have a 1 in the i-th position and a 0 in the remaining $|V| - 1$ positions

cat:	0	0	0	0	0	1	0
dog:	0	1	0	0	0	0	0
truck:	0	0	0	1	0	0	0

Problems:

- V tends to be very large (for example, 50K for PTB, 13M for Google 1T corpus)

cat:	0	0	0	0	0	1	0
dog:	0	1	0	0	0	0	0
truck:	0	0	0	1	0	0	0

Problems:

- V tends to be very large (for example, 50K for PTB, 13M for Google 1T corpus)
- These representations do not capture any notion of similarity

cat:	0	0	0	0	0	1	0
dog:	0	1	0	0	0	0	0
truck:	0	0	0	1	0	0	0

Problems:

- V tends to be very large (for example, 50K for PTB, 13M for Google 1T corpus)
- These representations do not capture any notion of similarity
- Ideally, we would want the representations of cat and dog (both domestic animals) to be closer to each other than the representations of cat and truck

cat:	0	0	0	0	0	1	0
dog:	0	1	0	0	0	0	0
truck:	0	0	0	1	0	0	0

$$euclid\_dist(\mathbf{cat}, \mathbf{dog}) = \sqrt{2}$$
$$euclid\_dist(\mathbf{dog}, \mathbf{truck}) = \sqrt{2}$$

Problems:

- V tends to be very large (for example, 50K for PTB, 13M for Google 1T corpus)
- These representations do not capture any notion of similarity
- Ideally, we would want the representations of cat and dog (both domestic animals) to be closer to each other than the representations of cat and truck
- However, with 1-hot representations, the Euclidean distance between **any two words** in the vocabulary in $\sqrt{2}$

cat:	0	0	0	0	0	1	0
dog:	0	1	0	0	0	0	0
truck:	0	0	0	1	0	0	0

$$euclid\_dist(\mathbf{cat}, \mathbf{dog}) = \sqrt{2}$$
$$euclid\_dist(\mathbf{dog}, \mathbf{truck}) = \sqrt{2}$$
$$cosine\_sim(\mathbf{cat}, \mathbf{dog}) = 0$$
$$cosine\_sim(\mathbf{dog}, \mathbf{truck}) = 0$$

Problems:

- V tends to be very large (for example, 50K for PTB, 13M for Google 1T corpus)
- These representations do not capture any notion of similarity
- Ideally, we would want the representations of cat and dog (both domestic animals) to be closer to each other than the representations of cat and truck
- However, with 1-hot representations, the Euclidean distance between **any two words** in the vocabulary in $\sqrt{2}$
- And the cosine similarity between **any two words** in the vocabulary is 0

Module 10.2: Distributed Representations of words

- *You shall know a word by the company it keeps - Firth, J. R. 1957:11*

A bank is a **financial** institution that accepts **deposits** from the public and creates **credit**.

- *You shall know a word by the company it keeps - Firth, J. R. 1957:11*
- Distributional similarity based representations

A bank is a **financial** institution that accepts **deposits** from the public and creates **credit**.

A bank is a **financial** institution that accepts **deposits** from the public and creates **credit**.

- *You shall know a word by the company it keeps - Firth, J. R. 1957:11*
- Distributional similarity based representations
- This leads us to the idea of co-occurrence matrix

A bank is a **financial** institution that accepts **deposits** from the public and creates **credit**.

The idea is to use the accompanying words (financial, deposits, credit) to represent bank

- *You shall know a word by the company it keeps - Firth, J. R. 1957:11*
- Distributional similarity based representations
- This leads us to the idea of co-occurrence matrix

Corpus:

- Human machine interface for computer applications
- User opinion of computer system response time
- User interface management system
- System engineering for improved response time

- A co-occurrence matrix is a **terms** $\times$ **terms** matrix which captures the number of times a term appears in the context of another term

Corpus:

- Human machine interface for computer applications
- User opinion of computer system response time
- User interface management system
- System engineering for improved response time

- A co-occurrence matrix is a **terms** $\times$ **terms** matrix which captures the number of times a term appears in the context of another term
- The context is defined as a window of k words around the terms

Corpus:

- Human machine interface for computer applications
- User opinion of computer system response time
- User interface management system
- System engineering for improved response time

- A co-occurrence matrix is a **terms** $\times$ **terms** matrix which captures the number of times a term appears in the context of another term
- The context is defined as a window of k words around the terms
- Let us build a co-occurrence matrix for this toy corpus with $k = 2$

Corpus:

- Human machine interface for computer applications
- User opinion of computer system response time
- User interface management system
- System engineering for improved response time

	human	**machine**	**system**	**for**	...	**user**
human	0	1	0	1	...	0
machine	1	0	0	1	...	0
system	0	0	0	1	...	2
for	1	1	1	0	...	0
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0	0	2	0	...	0

Co-occurence Matrix

- A co-occurrence matrix is a **terms** $\times$ **terms** matrix which captures the number of times a term appears in the context of another term
- The context is defined as a window of k words around the terms
- Let us build a co-occurrence matrix for this toy corpus with $k = 2$
- This is also known as a **word** $\times$ **context** matrix

Corpus:

- Human machine interface for computer applications
- User opinion of computer system response time
- User interface management system
- System engineering for improved response time

	human	machine	system	for	...	user
human	0	1	0	1	...	0
machine	1	0	0	1	...	0
system	0	0	0	1	...	2
for	1	1	1	0	...	0
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0	0	2	0	...	0

Co-occurence Matrix

- A co-occurrence matrix is a **terms** $\times$ **terms** matrix which captures the number of times a term appears in the context of another term
- The context is defined as a window of k words around the terms
- Let us build a co-occurrence matrix for this toy corpus with $k = 2$
- This is also known as a **word** $\times$ **context** matrix
- You could choose the set of **words** and **contexts** to be same or different

Corpus:

- Human machine interface for computer applications
- User opinion of computer system response time
- User interface management system
- System engineering for improved response time

	human	machine	system	for	...	user
human	0	1	0	1	...	0
machine	1	0	0	1	...	0
system	0	0	0	1	...	2
for	1	1	1	0	...	0
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0	0	2	0	...	0

Co-occurence Matrix

- A co-occurrence matrix is a **terms** $\times$ **terms** matrix which captures the number of times a term appears in the context of another term
- The context is defined as a window of k words around the terms
- Let us build a co-occurrence matrix for this toy corpus with $k = 2$
- This is also known as a **word** $\times$ **context** matrix
- You could choose the set of **words** and **contexts** to be same or different
- Each row (column) of the co-occurrence matrix gives a vectorial representation of the corresponding word (context)

	human	machine	system	for	...	user
human	0	1	0	1	...	0
machine	1	0	0	1	...	0
system	0	0	0	1	...	2
for	1	1	1	0	...	0
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0	0	2	0	...	0

Some (fixable) problems

- Stop words (a, the, for, etc.) are very frequent $\rightarrow$ these counts will be very high

	human	machine	system	...	user
human	0	1	0	...	0
machine	1	0	0	...	0
system	0	0	0	...	2
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
user	0	0	2	...	0

Some (fixable) problems

- Stop words (a, the, for, etc.) are very frequent $\rightarrow$ these counts will be very high
- Solution 1: Ignore very frequent words

	human	machine	system	for	...	user
human	0	1	0	x	...	0
machine	1	0	0	x	...	0
system	0	0	0	x	...	2
for	x	x	x	x	...	x
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0	0	2	x	...	0

Some (fixable) problems

- Stop words (a, the, for, etc.) are very frequent $\rightarrow$ these counts will be very high
- Solution 1: Ignore very frequent words
- Solution 2: Use a threshold t (say, t = 100)

$$X_{ij} = min(count(w_i, c_j), t),$$

where w is word and c is context.

Some (fixable) problems

- Solution 3: Instead of $count(w, c)$ use $PMI(w, c)$

Some (fixable) problems

- Solution 3: Instead of $count(w, c)$ use $PMI(w, c)$

$$PMI(w, c) = \log \frac{p(c|w)}{p(c)}$$
$$= \log \frac{count(w, c) * N}{count(c) * count(w)}$$

N is the total number of words

Some (fixable) problems

	human	machine	system	for	...	user
human	0	2.944	0	2.25	...	0
machine	2.944	0	0	2.25	...	0
system	0	0	0	1.15	...	1.84
for	2.25	2.25	1.15	0	...	0
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0	0	1.84	0	...	0

- Solution 3: Instead of $count(w, c)$ use $PMI(w, c)$

$$PMI(w, c) = \log \frac{p(c|w)}{p(c)}$$
$$= \log \frac{count(w, c) * N}{count(c) * count(w)}$$

N is the total number of words

Some (fixable) problems

	human	machine	system	for	...	user
human	0	2.944	0	2.25	...	0
machine	2.944	0	0	2.25	...	0
system	0	0	0	1.15	...	1.84
for	2.25	2.25	1.15	0	...	0
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0	0	1.84	0	...	0

- Solution 3: Instead of $count(w, c)$ use $PMI(w, c)$

$$PMI(w, c) = \log \frac{p(c|w)}{p(c)}$$
$$= \log \frac{count(w, c) * N}{count(c) * count(w)}$$

N *is the total number of words*

- If $count(w, c) = 0$, $PMI(w, c) = -\infty$

	human	machine	system	for	...	user
human	0	2.944	0	2.25	...	0
machine	2.944	0	0	2.25	...	0
system	0	0	0	1.15	...	1.84
for	2.25	2.25	1.15	0	...	0
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0	0	1.84	0	...	0

Some (fixable) problems

- Solution 3: Instead of $count(w, c)$ use $PMI(w, c)$

$$PMI(w, c) = \log \frac{p(c|w)}{p(c)}$$
$$= \log \frac{count(w, c) * N}{count(c) * count(w)}$$

N *is the total number of words*

- If $count(w, c) = 0$, $PMI(w, c) = -\infty$

Instead use,

$$\begin{aligned} PMI_0(w, c) &= PMI(w, c) \quad && if\ count(w, c) > 0 \\ &= 0 && otherwise \end{aligned}$$

Some (fixable) problems

	human	machine	system	for	...	user
human	0	2.944	0	2.25	...	0
machine	2.944	0	0	2.25	...	0
system	0	0	0	1.15	...	1.84
for	2.25	2.25	1.15	0	...	0
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0	0	1.84	0	...	0

- Solution 3: Instead of $count(w, c)$ use $PMI(w, c)$

$$PMI(w, c) = \log \frac{p(c|w)}{p(c)} = \log \frac{count(w, c) * N}{count(c) * count(w)}$$

N *is the total number of words*

- If $count(w, c) = 0$, $PMI(w, c) = -\infty$

Instead use,

$$PMI_0(w, c) = PMI(w, c) \quad if\ count(w, c) > 0$$
$$= 0 \quad otherwise$$

or

$$PPMI(w, c) = PMI(w, c) \quad if\ PMI(w, c) > 0$$
$$= 0 \quad otherwise$$

Some (severe) problems

- Very high dimensional ($|V|$)

	human	machine	system	for	...	user
human	0	2.944	0	2.25	...	0
machine	2.944	0	0	2.25	...	0
system	0	0	0	1.15	...	1.84
for	2.25	2.25	1.15	0	...	0
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0	0	1.84	0	...	0

	human	machine	system	for	...	user
human	0	2.944	0	2.25	...	0
machine	2.944	0	0	2.25	...	0
system	0	0	0	1.15	...	1.84
for	2.25	2.25	1.15	0	...	0
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0	0	1.84	0	...	0

Some (severe) problems

- Very high dimensional ($|V|$)
- Very sparse

	human	machine	system	for	...	user
human	0	2.944	0	2.25	...	0
machine	2.944	0	0	2.25	...	0
system	0	0	0	1.15	...	1.84
for	2.25	2.25	1.15	0	...	0
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0	0	1.84	0	...	0

Some (severe) problems

- Very high dimensional ($|V|$)
- Very sparse
- Grows with the size of the vocabulary

	human	machine	system	for	...	user
human	0	2.944	0	2.25	...	0
machine	2.944	0	0	2.25	...	0
system	0	0	0	1.15	...	1.84
for	2.25	2.25	1.15	0	...	0
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0	0	1.84	0	...	0

Some (severe) problems

- Very high dimensional ($|V|$)
- Very sparse
- Grows with the size of the vocabulary
- **Solution:** Use dimensionality reduction (SVD)

Module 10.3: SVD for learning word representations

$$\begin{bmatrix} & & \\ & X & \\ & & \end{bmatrix}_{m \times n} =$$

$$\begin{bmatrix} \uparrow & \cdots & \uparrow \\ u_1 & \cdots & u_k \\ \downarrow & \cdots & \downarrow \end{bmatrix}_{m \times k} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{bmatrix}_{k \times k} \begin{bmatrix} \leftarrow & v_1^T & \rightarrow \\ & \vdots & \\ \leftarrow & v_k^T & \rightarrow \end{bmatrix}_{k \times n}$$

- Singular Value Decomposition gives a rank k approximation of the original matrix

$$X = X_{PPMI_{m \times n}} = U_{m \times k} \Sigma_{k \times k} V^T_{k \times n}$$

X_{PPMI} (simplifying notation to X) is the co-occurrence matrix with PPMI values

$$\begin{bmatrix} & & \\ & X & \\ & & \end{bmatrix}_{m\times n} =$$

$$\begin{bmatrix} \uparrow & \cdots & \uparrow \\ & & \\ u_1 & \cdots & u_k \\ \downarrow & \cdots & \downarrow \end{bmatrix}_{m\times k} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{bmatrix}_{k\times k} \begin{bmatrix} \leftarrow & v_1^T & \rightarrow \\ & \vdots & \\ \leftarrow & v_k^T & \rightarrow \end{bmatrix}_{k\times n}$$

- Singular Value Decomposition gives a rank k approximation of the original matrix

$$X = X_{PPMI_{m\times n}} = U_{m\times k}\Sigma_{k\times k}V^T_{k\times n}$$

X_{PPMI} (simplifying notation to X) is the co-occurrence matrix with PPMI values

- SVD gives the best rank-k approximation of the original data (X)

$$\begin{bmatrix} \\ & X & \\ \\ \end{bmatrix}_{m\times n} = \begin{bmatrix} \uparrow & \cdots & \uparrow \\ u_1 & \cdots & u_k \\ \downarrow & \cdots & \downarrow \end{bmatrix}_{m\times k} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{bmatrix}_{k\times k} \begin{bmatrix} \leftarrow & v_1^T & \rightarrow \\ & \vdots & \\ \leftarrow & v_k^T & \rightarrow \end{bmatrix}_{k\times n}$$

- Singular Value Decomposition gives a rank k approximation of the original matrix

$$X = X_{PPMI_{m\times n}} = U_{m\times k}\Sigma_{k\times k}V_{k\times n}^T$$

X_{PPMI} (simplifying notation to X) is the co-occurrence matrix with PPMI values

- SVD gives the best rank-k approximation of the original data (X)
- Discovers latent semantics in the corpus (let us examine this with the help of an example)

- Notice that the product can be written as a sum of k rank-1 matrices

$$\begin{bmatrix} & & \\ & X & \\ & & \end{bmatrix}_{m\times n} =$$

$$\begin{bmatrix} \uparrow & \cdots & \uparrow \\ & & \\ u_1 & \cdots & u_k \\ \downarrow & \cdots & \downarrow \end{bmatrix}_{m\times k} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{bmatrix}_{k\times k} \begin{bmatrix} \leftarrow & v_1^T & \rightarrow \\ & \vdots & \\ \leftarrow & v_k^T & \rightarrow \end{bmatrix}_{k\times n}$$

$$= \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_k u_k v_k^T$$

$$\begin{bmatrix} & & \\ & X & \\ & & \end{bmatrix}_{m\times n} =$$

$$\begin{bmatrix} \uparrow & \cdots & \uparrow \\ u_1 & \cdots & u_k \\ \downarrow & \cdots & \downarrow \end{bmatrix}_{m\times k} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{bmatrix}_{k\times k} \begin{bmatrix} \leftarrow & v_1^T & \rightarrow \\ & \vdots & \\ \leftarrow & v_k^T & \rightarrow \end{bmatrix}_{k\times n}$$

$$= \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_k u_k v_k^T$$

- Notice that the product can be written as a sum of k rank-1 matrices
- Each $\sigma_i u_i v_i^T \in \mathbb{R}^{m\times n}$ because it is a product of a $m \times 1$ vector with a $1 \times n$ vector

$$
\begin{bmatrix} & & \\ & X & \\ & & \end{bmatrix}_{m \times n} =
$$

$$
\begin{bmatrix} \uparrow & \cdots & \uparrow \\ u_1 & \cdots & u_k \\ \downarrow & \cdots & \downarrow \end{bmatrix}_{m \times k} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{bmatrix}_{k \times k} \begin{bmatrix} \leftarrow & v_1^T & \rightarrow \\ & \vdots & \\ \leftarrow & v_k^T & \rightarrow \end{bmatrix}_{k \times n}
$$

$$
= \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_k u_k v_k^T
$$

- Notice that the product can be written as a sum of k rank-1 matrices
- Each $\sigma_i u_i v_i^T \in \mathbb{R}^{m \times n}$ because it is a product of a $m \times 1$ vector with a $1 \times n$ vector
- If we truncate the sum at $\sigma_1 u_1 v_1^T$ then we get the best rank-1 approximation of X

$$\begin{bmatrix} & & \\ & X & \\ & & \end{bmatrix}_{m\times n} =$$

$$\begin{bmatrix} \uparrow & \cdots & \uparrow \\ u_1 & \cdots & u_k \\ \downarrow & \cdots & \downarrow \end{bmatrix}_{m\times k} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{bmatrix}_{k\times k} \begin{bmatrix} \leftarrow & v_1^T & \rightarrow \\ & \vdots & \\ \leftarrow & v_k^T & \rightarrow \end{bmatrix}_{k\times n}$$

$$= \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_k u_k v_k^T$$

- Notice that the product can be written as a sum of k rank-1 matrices
- Each $\sigma_i u_i v_i^T \in \mathbb{R}^{m\times n}$ because it is a product of a $m \times 1$ vector with a $1 \times n$ vector
- If we truncate the sum at $\sigma_1 u_1 v_1^T$ then we get the best rank-1 approximation of X (By SVD theorem! But what does this mean? We will see on the next slide)

$$\begin{bmatrix} & & \\ & X & \\ & & \end{bmatrix}_{m\times n} =$$

$$\begin{bmatrix} \uparrow & \cdots & \uparrow \\ u_1 & \cdots & u_k \\ \downarrow & \cdots & \downarrow \end{bmatrix}_{m\times k} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{bmatrix}_{k\times k} \begin{bmatrix} \leftarrow & v_1^T & \rightarrow \\ & \vdots & \\ \leftarrow & v_k^T & \rightarrow \end{bmatrix}_{k\times n}$$

$$= \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_k u_k v_k^T$$

- Notice that the product can be written as a sum of k rank-1 matrices
- Each $\sigma_i u_i v_i^T \in \mathbb{R}^{m\times n}$ because it is a product of a $m \times 1$ vector with a $1 \times n$ vector
- If we truncate the sum at $\sigma_1 u_1 v_1^T$ then we get the best rank-1 approximation of X (By SVD theorem! But what does this mean? We will see on the next slide)
- If we truncate the sum at $\sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T$ then we get the best rank-2 approximation of X and so on

- What do we mean by approximation here?

$$\begin{bmatrix} & & \\ & X & \\ & & \end{bmatrix}_{m \times n} =$$

$$\begin{bmatrix} \uparrow & \cdots & \uparrow \\ u_1 & \cdots & u_k \\ \downarrow & \cdots & \downarrow \end{bmatrix}_{m \times k} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{bmatrix}_{k \times k} \begin{bmatrix} \leftarrow & v_1^T & \rightarrow \\ & \vdots & \\ \leftarrow & v_k^T & \rightarrow \end{bmatrix}_{k \times n}$$

$$= \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_k u_k v_k^T$$

- What do we mean by approximation here?
- Notice that X has $m \times n$ entries

$$\begin{bmatrix} & & \\ & X & \\ & & \end{bmatrix}_{m\times n} =$$

$$\begin{bmatrix} \uparrow & \cdots & \uparrow \\ u_1 & \cdots & u_k \\ \downarrow & \cdots & \downarrow \end{bmatrix}_{m\times k} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{bmatrix}_{k\times k} \begin{bmatrix} \leftarrow & v_1^T & \rightarrow \\ & \vdots & \\ \leftarrow & v_k^T & \rightarrow \end{bmatrix}_{k\times n}$$

$$= \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_k u_k v_k^T$$

$$\begin{bmatrix} & & \\ & X & \\ & & \end{bmatrix}_{m\times n} =$$

$$\begin{bmatrix} \uparrow & \cdots & \uparrow \\ u_1 & \cdots & u_k \\ \downarrow & \cdots & \downarrow \end{bmatrix}_{m\times k} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{bmatrix}_{k\times k} \begin{bmatrix} \leftarrow & v_1^T & \rightarrow \\ & \vdots & \\ \leftarrow & v_k^T & \rightarrow \end{bmatrix}_{k\times n}$$

$$= \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_k u_k v_k^T$$

- What do we mean by approximation here?
- Notice that X has $m \times n$ entries
- When we use he rank-1 approximation we are using only $n + m + 1$ entries to reconstruct $[u \in \mathbb{R}^m, v \in \mathbb{R}^n, \sigma \in \mathbb{R}^1]$

$$\begin{bmatrix} & & \\ & X & \\ & & \end{bmatrix}_{m \times n} =$$

$$\begin{bmatrix} \uparrow & \cdots & \uparrow \\ u_1 & \cdots & u_k \\ \downarrow & \cdots & \downarrow \end{bmatrix}_{m \times k} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{bmatrix}_{k \times k} \begin{bmatrix} \leftarrow & v_1^T & \rightarrow \\ & \vdots & \\ \leftarrow & v_k^T & \rightarrow \end{bmatrix}_{k \times n}$$

$$= \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_k u_k v_k^T$$

- What do we mean by approximation here?
- Notice that X has $m \times n$ entries
- When we use he rank-1 approximation we are using only $n + m + 1$ entries to reconstruct $[u \in \mathbb{R}^m, v \in \mathbb{R}^n, \sigma \in \mathbb{R}^1]$
- But SVD theorem tells us that u_1,v_1 and σ_1 store the most information in X (akin to the principal components in X)

$$\begin{bmatrix} & & \\ & X & \\ & & \end{bmatrix}_{m \times n} =$$

$$\begin{bmatrix} \uparrow & \cdots & \uparrow \\ u_1 & \cdots & u_k \\ \downarrow & \cdots & \downarrow \end{bmatrix}_{m \times k} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{bmatrix}_{k \times k} \begin{bmatrix} \leftarrow & v_1^T & \rightarrow \\ & \vdots & \\ \leftarrow & v_k^T & \rightarrow \end{bmatrix}_{k \times n}$$

$$= \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_k u_k v_k^T$$

- What do we mean by approximation here?
- Notice that X has $m \times n$ entries
- When we use he rank-1 approximation we are using only $n + m + 1$ entries to reconstruct $[u \in \mathbb{R}^m, v \in \mathbb{R}^n, \sigma \in \mathbb{R}^1]$
- But SVD theorem tells us that u_1,v_1 and σ_1 store the most information in X (akin to the principal components in X)
- Each subsequent term ($\sigma_2 u_2 v_2^T$, $\sigma_3 u_3 v_3^T$, …) stores less and less important information

$$\overbrace{\boxed{0}\boxed{0}\boxed{0}\boxed{1}}^{verylight}\overbrace{\boxed{1}\boxed{0}\boxed{1}\boxed{1}}^{green}$$

$$\overbrace{\boxed{0}\boxed{0}\boxed{1}\boxed{0}}^{light}\overbrace{\boxed{1}\boxed{0}\boxed{1}\boxed{1}}^{green}$$

$$\overbrace{\boxed{0}\boxed{1}\boxed{0}\boxed{0}}^{dark}\overbrace{\boxed{1}\boxed{0}\boxed{1}\boxed{1}}^{green}$$

$$\overbrace{\boxed{1}\boxed{0}\boxed{0}\boxed{0}}^{verydark}\overbrace{\boxed{1}\boxed{0}\boxed{1}\boxed{1}}^{green}$$

- As an analogy consider the case when we are using 8 bits to represent colors

$$\overbrace{\boxed{0}\boxed{0}\boxed{0}\boxed{1}}^{verylight}\overbrace{\boxed{1}\boxed{0}\boxed{1}\boxed{1}}^{green}$$

$$\overbrace{\boxed{0}\boxed{0}\boxed{1}\boxed{0}}^{light}\overbrace{\boxed{1}\boxed{0}\boxed{1}\boxed{1}}^{green}$$

$$\overbrace{\boxed{0}\boxed{1}\boxed{0}\boxed{0}}^{dark}\overbrace{\boxed{1}\boxed{0}\boxed{1}\boxed{1}}^{green}$$

$$\overbrace{\boxed{1}\boxed{0}\boxed{0}\boxed{0}}^{verydark}\overbrace{\boxed{1}\boxed{0}\boxed{1}\boxed{1}}^{green}$$

- As an analogy consider the case when we are using 8 bits to represent colors
- The representation of very light, light, dark and very dark green would look different

$$\overbrace{\boxed{0}\boxed{0}\boxed{0}\boxed{1}}^{verylight}\overbrace{\boxed{1}\boxed{0}\boxed{1}\boxed{1}}^{green}$$

$$\overbrace{\boxed{0}\boxed{0}\boxed{1}\boxed{0}}^{light}\overbrace{\boxed{1}\boxed{0}\boxed{1}\boxed{1}}^{green}$$

$$\overbrace{\boxed{0}\boxed{1}\boxed{0}\boxed{0}}^{dark}\overbrace{\boxed{1}\boxed{0}\boxed{1}\boxed{1}}^{green}$$

$$\overbrace{\boxed{1}\boxed{0}\boxed{0}\boxed{0}}^{verydark}\overbrace{\boxed{1}\boxed{0}\boxed{1}\boxed{1}}^{green}$$

- As an analogy consider the case when we are using 8 bits to represent colors
- The representation of very light, light, dark and very dark green would look different
- But now what if we were asked to compress this into 4 bits? (akin to compressing $m \times m$ values into $m + m + 1$ values on the previous slide)

$$\overbrace{0\quad 0\quad 0\quad 1}^{verylight}\ \overbrace{\boxed{1\,|\,0\,|\,1\,|\,1}}^{green}$$

$$\overbrace{0\quad 0\quad 1\quad 0}^{light}\ \overbrace{\boxed{1\,|\,0\,|\,1\,|\,1}}^{green}$$

$$\overbrace{0\quad 1\quad 0\quad 0}^{dark}\ \overbrace{\boxed{1\,|\,0\,|\,1\,|\,1}}^{green}$$

$$\overbrace{1\quad 0\quad 0\quad 0}^{verydark}\ \overbrace{\boxed{1\,|\,0\,|\,1\,|\,1}}^{green}$$

- As an analogy consider the case when we are using 8 bits to represent colors
- The representation of very light, light, dark and very dark green would look different
- But now what if we were asked to compress this into 4 bits? (akin to compressing $m \times m$ values into $m + m + 1$ values on the previous slide)
- We will retain the most important 4 bits and now the previously (slightly) latent similarity between the colors now becomes very obvious

$$\overbrace{0\quad 0\quad 0\quad 1}^{verylight}\ \overbrace{\boxed{1\,|\,0\,|\,1\,|\,1}}^{green}$$

$$\overbrace{0\quad 0\quad 1\quad 0}^{light}\ \overbrace{\boxed{1\,|\,0\,|\,1\,|\,1}}^{green}$$

$$\overbrace{0\quad 1\quad 0\quad 0}^{dark}\ \overbrace{\boxed{1\,|\,0\,|\,1\,|\,1}}^{green}$$

$$\overbrace{1\quad 0\quad 0\quad 0}^{verydark}\ \overbrace{\boxed{1\,|\,0\,|\,1\,|\,1}}^{green}$$

- As an analogy consider the case when we are using 8 bits to represent colors
- The representation of very light, light, dark and very dark green would look different
- But now what if we were asked to compress this into 4 bits? (akin to compressing $m \times m$ values into $m + m + 1$ values on the previous slide)
- We will retain the most important 4 bits and now the previously (slightly) latent similarity between the colors now becomes very obvious
- Something similar is guaranteed by SVD (retain the most important information and discover the latent similarities between words)

	human	machine	system	for	...	user
human	0	2.944	0	2.25	...	**0**
machine	2.944	0	0	2.25	...	0
system	0	**0**	0	1.15	...	1.84
for	2.25	2.25	1.15	0	...	0
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0	0	1.84	0	...	0

Co-occurrence Matrix (X)

	human	machine	system	for	...	user
human	2.01	2.01	0.23	2.14	...	**0.43**
machine	2.01	2.01	0.23	2.14	...	0.43
system	0.23	**0.23**	1.17	0.96	...	1.29
for	2.14	2.14	0.96	1.87	...	-0.13
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0.43	0.43	1.29	-0.13	...	1.71

Low rank $X \rightarrow$ Low rank $\hat{X}$

- Notice that after low rank reconstruction with SVD, the latent co-occurrence between $\{system, machine\}$ and $\{human, user\}$ has become visible

$X =$

	human	**machine**	**system**	**for**	...	**user**
human	0	2.944	0	2.25	...	0
machine	2.944	0	0	2.25	...	0
system	0	0	0	1.15	...	1.84
for	2.25	2.25	1.15	0	...	0
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0	0	1.84	0	...	0

- Recall that earlier each row of the original matrix X served as the representation of a word

$X =$

	human	machine	system	for	...	user
human	0	2.944	0	2.25	...	0
machine	2.944	0	0	2.25	...	0
system	0	0	0	1.15	...	1.84
for	2.25	2.25	1.15	0	...	0
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0	0	1.84	0	...	0

$XX^T =$

	human	machine	system	for	...	user
human	32.5	23.9	7.78	20.25	...	7.01
machine	23.9	32.5	7.78	20.25	...	7.01
system	7.78	7.78	0	17.65	...	21.84
for	20.25	20.25	17.65	36.3	...	11.8
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	7.01	7.01	21.84	11.8	...	28.3

$$cosine\_sim(human, user) = 0.21$$

- Recall that earlier each row of the original matrix X served as the representation of a word
- Then XX^T is a matrix whose ij-th entry is the dot product between the representation of word i ($X[i :]$) and word j ($X[j :]$)

$$X =$$

	human	**machine**	**system**	**for**	...	**user**
human	0	2.944	0	2.25	...	0
machine	2.944	0	0	2.25	...	0
system	0	0	0	1.15	...	1.84
for	2.25	2.25	1.15	0	...	0
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0	0	1.84	0	...	0

$$XX^T =$$

	human	**machine**	**system**	**for**	...	**user**
human	32.5	23.9	7.78	20.25	...	7.01
machine	23.9	32.5	7.78	20.25	...	7.01
system	7.78	7.78	0	17.65	...	21.84
for	20.25	20.25	17.65	36.3	...	11.8
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	7.01	7.01	21.84	11.8	...	28.3

$$cosine\_sim(human, user) = 0.21$$

- Recall that earlier each row of the original matrix X served as the representation of a word
- Then XX^T is a matrix whose ij-th entry is the dot product between the representation of word i ($X[i :]$) and word j ($X[j :]$)

$$X[i :]$$

$$X[j :]$$

$$X =$$

	human	machine	system	for	...	user
human	0	2.944	0	2.25	...	0
machine	2.944	0	0	2.25	...	0
system	0	0	0	1.15	...	1.84
for	2.25	2.25	1.15	0	...	0
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0	0	1.84	0	...	0

$$XX^T =$$

	human	machine	system	for	...	user
human	32.5	23.9	7.78	20.25	...	7.01
machine	23.9	32.5	7.78	20.25	...	7.01
system	7.78	7.78	0	17.65	...	21.84
for	20.25	20.25	17.65	36.3	...	11.8
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	7.01	7.01	21.84	11.8	...	28.3

$$cosine\_sim(human, user) = 0.21$$

- Recall that earlier each row of the original matrix X served as the representation of a word
- Then XX^T is a matrix whose ij-th entry is the dot product between the representation of word i ($X[i:]$) and word j ($X[j:]$)

$$\begin{array}{c} X[i:] \\ \\ X[j:] \end{array} \underbrace{\begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 0 \\ 1 & 3 & 5 \end{bmatrix}}_{X}$$

$X =$

	human	machine	system	for	...	user
human	0	2.944	0	2.25	...	0
machine	2.944	0	0	2.25	...	0
system	0	0	0	1.15	...	1.84
for	2.25	2.25	1.15	0	...	0
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0	0	1.84	0	...	0

$XX^T =$

	human	machine	system	for	...	user
human	32.5	23.9	7.78	20.25	...	7.01
machine	23.9	32.5	7.78	20.25	...	7.01
system	7.78	7.78	0	17.65	...	21.84
for	20.25	20.25	17.65	36.3	...	11.8
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	7.01	7.01	21.84	11.8	...	28.3

$$cosine\_sim(human, user) = 0.21$$

- Recall that earlier each row of the original matrix X served as the representation of a word
- Then XX^T is a matrix whose ij-th entry is the dot product between the representation of word i ($X[i:]$) and word j ($X[j:]$)

$$\begin{array}{c} X[i:] \\ \\ X[j:] \end{array} \underbrace{\begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 0 \\ 1 & 3 & 5 \end{bmatrix}}_{X} \underbrace{\begin{bmatrix} 1 & 2 & 1 \\ 2 & 1 & 3 \\ 3 & 0 & 5 \end{bmatrix}}_{X^T}$$

$$X =$$

	human	machine	system	for	...	user
human	0	2.944	0	2.25	...	0
machine	2.944	0	0	2.25	...	0
system	0	0	0	1.15	...	1.84
for	2.25	2.25	1.15	0	...	0
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0	0	1.84	0	...	0

$$XX^T =$$

	human	machine	system	for	...	user
human	32.5	23.9	7.78	20.25	...	7.01
machine	23.9	32.5	7.78	20.25	...	7.01
system	7.78	7.78	0	17.65	...	21.84
for	20.25	20.25	17.65	36.3	...	11.8
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	7.01	7.01	21.84	11.8	...	28.3

$$cosine\_sim(human, user) = 0.21$$

- Recall that earlier each row of the original matrix X served as the representation of a word
- Then XX^T is a matrix whose ij-th entry is the dot product between the representation of word i ($X[i:]$) and word j ($X[j:]$)

$$\begin{array}{c} X[i:] \\ \\ X[j:] \end{array} \underbrace{\begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 0 \\ 1 & 3 & 5 \end{bmatrix}}_{X} \underbrace{\begin{bmatrix} 1 & 2 & 1 \\ 2 & 1 & 3 \\ 3 & 0 & 5 \end{bmatrix}}_{X^T}$$

$$= \underbrace{\begin{bmatrix} . & . & 22 \\ . & . & . \\ . & . & . \end{bmatrix}}_{XX^T}$$

$X =$

	human	machine	system	for	...	user
human	0	2.944	0	2.25	...	0
machine	2.944	0	0	2.25	...	0
system	0	0	0	1.15	...	1.84
for	2.25	2.25	1.15	0	...	0
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0	0	1.84	0	...	0

$XX^T =$

	human	machine	system	for	...	user
human	32.5	23.9	7.78	20.25	...	7.01
machine	23.9	32.5	7.78	20.25	...	7.01
system	7.78	7.78	0	17.65	...	21.84
for	20.25	20.25	17.65	36.3	...	11.8
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	7.01	7.01	21.84	11.8	...	28.3

$$cosine\_sim(human, user) = 0.21$$

- Recall that earlier each row of the original matrix X served as the representation of a word
- Then XX^T is a matrix whose ij-th entry is the dot product between the representation of word i ($X[i:]$) and word j ($X[j:]$)

$$\begin{array}{c} X[i:] \\ \\ X[j:] \end{array} \underbrace{\begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 0 \\ 1 & 3 & 5 \end{bmatrix}}_{X} \underbrace{\begin{bmatrix} 1 & 2 & 1 \\ 2 & 1 & 3 \\ 3 & 0 & 5 \end{bmatrix}}_{X^T}$$

$$= \underbrace{\begin{bmatrix} . & . & 22 \\ . & . & . \\ . & . & . \end{bmatrix}}_{XX^T}$$

- The ij-th entry of XX^T thus (roughly) captures the cosine similarity between $word_i, word_j$

- Once we do an SVD what is a good choice for the representation of $word_i$?

$$\hat{X} =$$

	human	**machine**	**system**	**for**	...	**user**
human	2.01	2.01	0.23	2.14	...	0.43
machine	2.01	2.01	0.23	2.14	...	0.43
system	0.23	0.23	1.17	0.96	...	1.29
for	2.14	2.14	0.96	1.87	...	-0.13
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0.43	0.43	1.29	-0.13	...	1.71

- Once we do an SVD what is a good choice for the representation of $word_i$?
- Obviously, taking the i-th row of the reconstructed matrix does not make sense because it is still high dimensional

$$\hat{X} =$$

	human	machine	system	for	...	user
human	2.01	2.01	0.23	2.14	...	0.43
machine	2.01	2.01	0.23	2.14	...	0.43
system	0.23	0.23	1.17	0.96	...	1.29
for	2.14	2.14	0.96	1.87	...	-0.13
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0.43	0.43	1.29	-0.13	...	1.71

$$\hat{X}\hat{X}^T =$$

	human	machine	system	for	...	user
human	25.4	25.4	7.6	21.9	...	6.84
machine	25.4	25.4	7.6	21.9	...	6.84
system	7.6	7.6	24.8	18.03	...	20.6
for	21.9	21.9	0.96	24.6	...	15.32
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	6.84	6.84	20.6	15.32	...	17.11

$$cosine\_sim(human, user) = 0.33$$

- Once we do an SVD what is a good choice for the representation of $word_i$?
- Obviously, taking the i-th row of the reconstructed matrix does not make sense because it is still high dimensional
- But we saw that the reconstructed matrix $\hat{X} = U\Sigma V^T$ discovers latent semantics and its word representations are more meaningful

$$\hat{X} =$$

	human	machine	system	for	...	user
human	2.01	2.01	0.23	2.14	...	0.43
machine	2.01	2.01	0.23	2.14	...	0.43
system	0.23	0.23	1.17	0.96	...	1.29
for	2.14	2.14	0.96	1.87	...	-0.13
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0.43	0.43	1.29	-0.13	...	1.71

$$\hat{X}\hat{X}^T =$$

	human	machine	system	for	...	user
human	25.4	25.4	7.6	21.9	...	6.84
machine	25.4	25.4	7.6	21.9	...	6.84
system	7.6	7.6	24.8	18.03	...	20.6
for	21.9	21.9	0.96	24.6	...	15.32
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	6.84	6.84	20.6	15.32	...	17.11

$$cosine\_sim(human, user) = 0.33$$

- Once we do an SVD what is a good choice for the representation of $word_i$?
- Obviously, taking the i-th row of the reconstructed matrix does not make sense because it is still high dimensional
- But we saw that the reconstructed matrix $\hat{X} = U\Sigma V^T$ discovers latent semantics and its word representations are more meaningful
- **Wishlist:** We would want representations of words (i, j) to be of smaller dimensions but still have the same similarity (dot product) as the corresponding rows of $\hat{X}$

$$\hat{X} =$$

	human	machine	system	for	...	user
human	2.01	2.01	0.23	2.14	...	0.43
machine	2.01	2.01	0.23	2.14	...	0.43
system	0.23	0.23	1.17	0.96	...	1.29
for	2.14	2.14	0.96	1.87	...	-0.13
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0.43	0.43	1.29	-0.13	...	1.71

$$\hat{X}\hat{X}^T =$$

	human	machine	system	for	...	user
human	25.4	25.4	7.6	21.9	...	6.84
machine	25.4	25.4	7.6	21.9	...	6.84
system	7.6	7.6	24.8	18.03	...	20.6
for	21.9	21.9	0.96	24.6	...	15.32
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	6.84	6.84	20.6	15.32	...	17.11

$$similarity = 0.33$$

- Notice that the dot product between the rows of the the matrix $W_{word} = U\Sigma$ is the same as the dot product between the rows of $\hat{X}$

$$\hat{X}\hat{X}^T = (U\Sigma V^T)(U\Sigma V^T)^T$$

$$\hat{X} =$$

	human	machine	system	for	...	user
human	2.01	2.01	0.23	2.14	...	0.43
machine	2.01	2.01	0.23	2.14	...	0.43
system	0.23	0.23	1.17	0.96	...	1.29
for	2.14	2.14	0.96	1.87	...	-0.13
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0.43	0.43	1.29	-0.13	...	1.71

$$\hat{X}\hat{X}^T =$$

	human	machine	system	for	...	user
human	25.4	25.4	7.6	21.9	...	6.84
machine	25.4	25.4	7.6	21.9	...	6.84
system	7.6	7.6	24.8	18.03	...	20.6
for	21.9	21.9	0.96	24.6	...	15.32
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	6.84	6.84	20.6	15.32	...	17.11

$$similarity = 0.33$$

- Notice that the dot product between the rows of the the matrix $W_{word} = U\Sigma$ is the same as the dot product between the rows of $\hat{X}$

$$\hat{X}\hat{X}^T = (U\Sigma V^T)(U\Sigma V^T)^T$$
$$= (U\Sigma V^T)(V\Sigma U^T)$$

$$\hat{X} =$$

	human	machine	system	for	...	user
human	2.01	2.01	0.23	2.14	...	0.43
machine	2.01	2.01	0.23	2.14	...	0.43
system	0.23	0.23	1.17	0.96	...	1.29
for	2.14	2.14	0.96	1.87	...	-0.13
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0.43	0.43	1.29	-0.13	...	1.71

$$\hat{X}\hat{X}^T =$$

	human	machine	system	for	...	user
human	25.4	25.4	7.6	21.9	...	6.84
machine	25.4	25.4	7.6	21.9	...	6.84
system	7.6	7.6	24.8	18.03	...	20.6
for	21.9	21.9	0.96	24.6	...	15.32
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	6.84	6.84	20.6	15.32	...	17.11

$$similarity = 0.33$$

- Notice that the dot product between the rows of the the matrix $W_{word} = U\Sigma$ is the same as the dot product between the rows of $\hat{X}$

$$\begin{aligned}\hat{X}\hat{X}^T &= (U\Sigma V^T)(U\Sigma V^T)^T \\ &= (U\Sigma V^T)(V\Sigma U^T) \\ &= U\Sigma\Sigma^T U^T \quad (\because V^T V = I)\end{aligned}$$

$$\hat{X} =$$

	human	machine	system	for	...	user
human	2.01	2.01	0.23	2.14	...	0.43
machine	2.01	2.01	0.23	2.14	...	0.43
system	0.23	0.23	1.17	0.96	...	1.29
for	2.14	2.14	0.96	1.87	...	-0.13
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0.43	0.43	1.29	-0.13	...	1.71

$$\hat{X}\hat{X}^T =$$

	human	machine	system	for	...	user
human	25.4	25.4	7.6	21.9	...	6.84
machine	25.4	25.4	7.6	21.9	...	6.84
system	7.6	7.6	24.8	18.03	...	20.6
for	21.9	21.9	0.96	24.6	...	15.32
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	6.84	6.84	20.6	15.32	...	17.11

$$similarity = 0.33$$

- Notice that the dot product between the rows of the the matrix $W_{word} = U\Sigma$ is the same as the dot product between the rows of $\hat{X}$

$$\begin{aligned}\hat{X}\hat{X}^T &= (U\Sigma V^T)(U\Sigma V^T)^T \\ &= (U\Sigma V^T)(V\Sigma U^T) \\ &= U\Sigma\Sigma^T U^T \quad (\because V^TV = I) \\ &= U\Sigma(U\Sigma)^T = W_{word}W_{word}^T\end{aligned}$$

$\hat{X} =$

	human	machine	system	for	...	user
human	2.01	2.01	0.23	2.14	...	0.43
machine	2.01	2.01	0.23	2.14	...	0.43
system	0.23	0.23	1.17	0.96	...	1.29
for	2.14	2.14	0.96	1.87	...	-0.13
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0.43	0.43	1.29	-0.13	...	1.71

$\hat{X}\hat{X}^T =$

	human	machine	system	for	...	user
human	25.4	25.4	7.6	21.9	...	6.84
machine	25.4	25.4	7.6	21.9	...	6.84
system	7.6	7.6	24.8	18.03	...	20.6
for	21.9	21.9	0.96	24.6	...	15.32
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	6.84	6.84	20.6	15.32	...	17.11

$$similarity = 0.33$$

- Notice that the dot product between the rows of the the matrix $W_{word} = U\Sigma$ is the same as the dot product between the rows of $\hat{X}$

$$\begin{aligned} \hat{X}\hat{X}^T &= (U\Sigma V^T)(U\Sigma V^T)^T \\ &= (U\Sigma V^T)(V\Sigma U^T) \\ &= U\Sigma\Sigma^T U^T \quad (\because V^T V = I) \\ &= U\Sigma(U\Sigma)^T = W_{word}W_{word}^T \end{aligned}$$

- Conventionally,

$$W_{word} = U\Sigma \in \mathbb{R}^{m\times k}$$

is taken as the representation of the m words in the vocabulary and

$$W_{context} = V$$

is taken as the representation of the context words

Module 10.4: Continuous bag of words model

- The methods that we have seen so far are called **count based models** because they use the co-occurrence counts of words

- The methods that we have seen so far are called **count based models** because they use the co-occurrence counts of words
- We will now see methods which directly **learn** word representations (these are called **(direct) prediction based models**)

The story ahead ...

- Continuous bag of words model

The story ahead ...

- Continuous bag of words model
- Skip gram model with negative sampling (the famous word2vec)

The story ahead ...

- Continuous bag of words model
- Skip gram model with negative sampling (the famous word2vec)
- GloVe word embeddings

The story ahead ...

- Continuous bag of words model
- Skip gram model with negative sampling (the famous word2vec)
- GloVe word embeddings
- Evaluating word embeddings

The story ahead ...

- Continuous bag of words model
- Skip gram model with negative sampling (the famous word2vec)
- GloVe word embeddings
- Evaluating word embeddings
- Good old SVD does just fine!!

- **Consider this Task:** Predict n-th word given previous n-1 words

- **Consider this Task:** Predict n-th word given previous n-1 words
- **Example:** he sat on a chair

Sometime in the 21st century, Joseph Cooper, a widowed former engineer and former NASA pilot, runs a farm with his father-in-law Donald, son Tom, and daughter Murphy, It is post-truth society (Cooper is reprimanded for telling Murphy that the Apollo missions did indeed happen) and a series of crop blights threatens humanity's survival. Murphy believes her bedroom is haunted by a poltergeist. When a pattern is created out of dust on the floor, Cooper realizes that gravity is behind its formation, not a "ghost". He interprets the pattern as a set of geographic coordinates formed into binary code. Cooper and Murphy follow the coordinates to a secret NASA facility, where they are met by Cooper's former professor, Dr. Brand.

Some sample 4 word windows from a corpus

- **Consider this Task:** Predict n-th word given previous n-1 words
- **Example:** he sat on a chair
- **Training data:** All n-word windows in your corpus

Sometime in the 21st century, Joseph Cooper, a widowed former engineer and former NASA pilot, runs a farm with his father-in-law Donald, son Tom, and daughter Murphy, It is post-truth society (Cooper is reprimanded for telling Murphy that the Apollo missions did indeed happen) and a series of crop blights threatens humanity's survival. Murphy believes her bedroom is haunted by a poltergeist. When a pattern is created out of dust on the floor, Cooper realizes that gravity is behind its formation, not a "ghost". He interprets the pattern as a set of geographic coordinates formed into binary code. Cooper and Murphy follow the coordinates to a secret NASA facility, where they are met by Cooper's former professor, Dr. Brand.

Some sample 4 word windows from a corpus

- **Consider this Task:** Predict n-th word given previous n-1 words
- **Example:** he sat on a chair
- **Training data:** All n-word windows in your corpus
- Training data for this task is easily available (take all n word windows from the whole of wikipedia)

Sometime in the 21st century, Joseph Cooper, a widowed former engineer and former NASA pilot, runs a farm with his father-in-law Donald, son Tom, and daughter Murphy, It is post-truth society (Cooper is reprimanded for telling Murphy that the Apollo missions did indeed happen) and a series of crop blights threatens humanity's survival. Murphy believes her bedroom is haunted by a poltergeist. When a pattern is created out of dust on the floor, Cooper realizes that gravity is behind its formation, not a "ghost". He interprets the pattern as a set of geographic coordinates formed into binary code. Cooper and Murphy follow the coordinates to a secret NASA facility, where they are met by Cooper's former professor, Dr. Brand.

Some sample 4 word windows from a corpus

- **Consider this Task:** Predict n-th word given previous n-1 words
- **Example:** he sat on a chair
- **Training data:** All n-word windows in your corpus
- Training data for this task is easily available (take all n word windows from the whole of wikipedia)
- For ease of illustration, we will first focus on the case when $n = 2$ (*i.e.*, predict second word based on first word)

We will now try to answer these two questions:

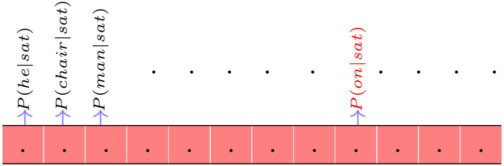
We will now try to answer these two questions:

- How do you model this task?

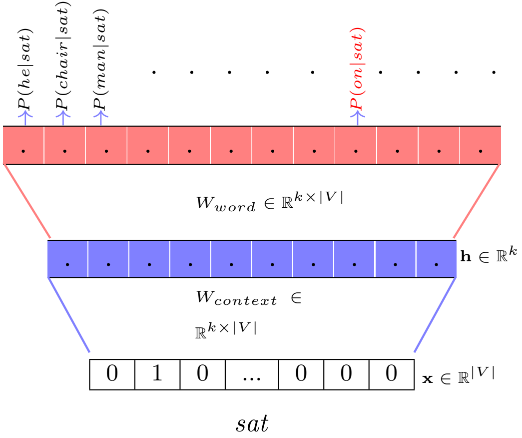
We will now try to answer these two questions:

- How do you model this task?
- What is the connection between this task and learning word representations?

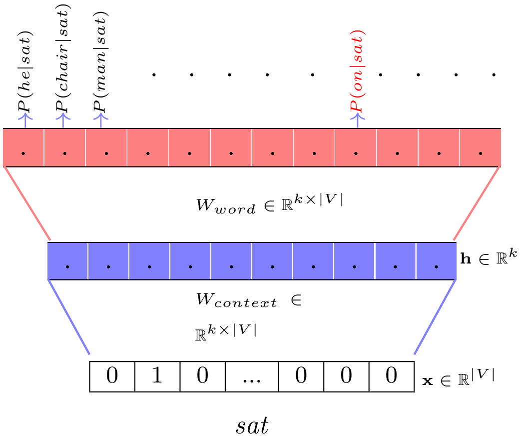
- We will model this problem using a feedforward neural network



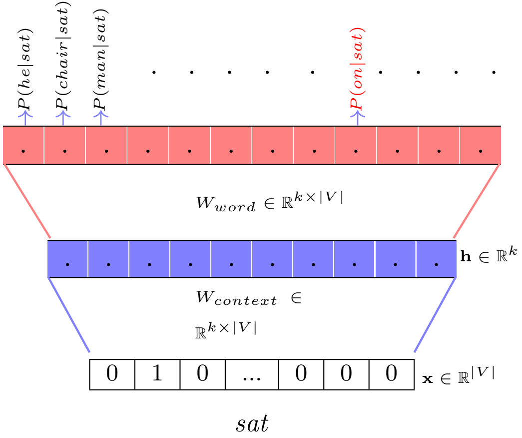
- We will model this problem using a feedforward neural network
- **Input:** One-hot representation of the **context word**



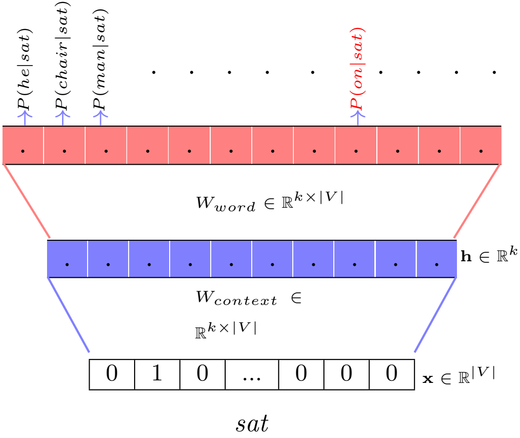
- We will model this problem using a feedforward neural network
- **Input:** One-hot representation of the **context word**
- **Output:** There are $|V|$ words (classes) possible and we want to predict a probability distribution over these $|V|$ classes (multi-class classification problem)



- We will model this problem using a feedforward neural network
- **Input:** One-hot representation of the **context word**
- **Output:** There are $|V|$ words (classes) possible and we want to predict a probability distribution over these $|V|$ classes (multi-class classification problem)
- **Parameters:** $\mathbf{W}_{context} \in \mathbb{R}^{k\times|V|}$ and $\mathbf{W}_{word} \in \mathbb{R}^{k\times|V|}$
  (we are assuming that the set of **words** and **context** words is the same: each of size $|V|$)

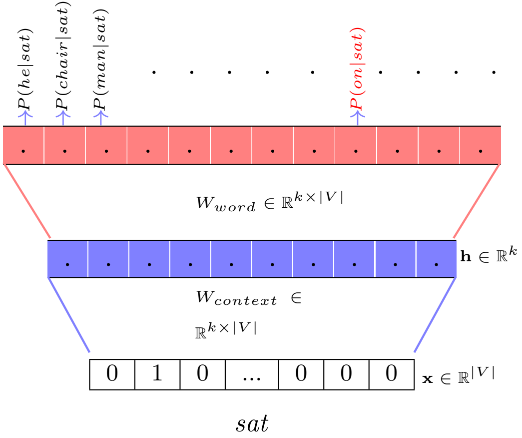


- What is the product $\mathbf{W}_{context}\mathbf{x}$ given that $\mathbf{x}$ is a one hot vector



- What is the product $\mathbf{W}_{context}\mathbf{x}$ given that $\mathbf{x}$ is a one hot vector
- It is simply the i-th column of $\mathbf{W}_{context}$

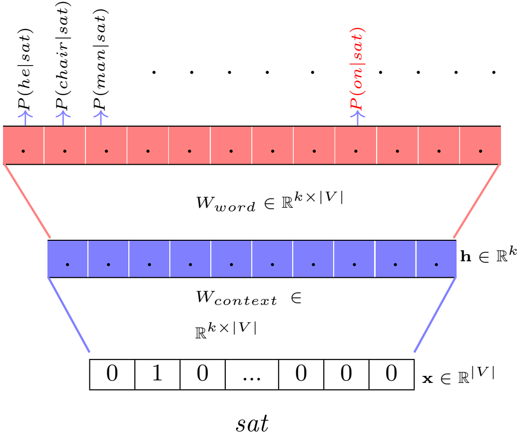
$$\begin{bmatrix} -1 & 0.5 & 2 \\ 3 & -1 & -2 \\ -2 & 1.7 & 3 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.5 \\ -1 \\ 1.7 \end{bmatrix}$$



- What is the product $\mathbf{W}_{context}\mathbf{x}$ given that $\mathbf{x}$ is a one hot vector
- It is simply the i-th column of $\mathbf{W}_{context}$

$$\begin{bmatrix} -1 & 0.5 & 2 \\ 3 & -1 & -2 \\ -2 & 1.7 & 3 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.5 \\ -1 \\ 1.7 \end{bmatrix}$$

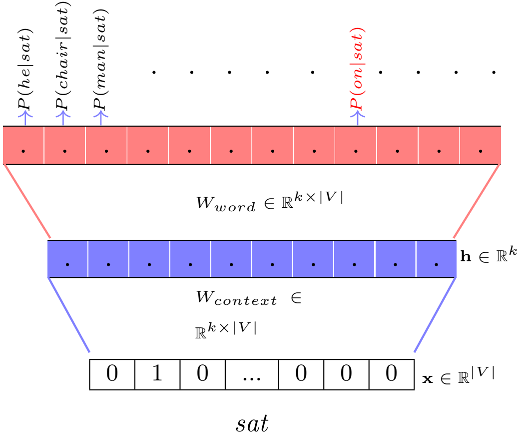
- So when the i^{th} word is present the i^{th} element in the one hot vector is ON and the i^{th} column of $\mathbf{W}_{context}$ gets selected



- What is the product $\mathbf{W}_{context}\mathbf{x}$ given that $\mathbf{x}$ is a one hot vector
- It is simply the i-th column of $\mathbf{W}_{context}$

$$\begin{bmatrix} -1 & 0.5 & 2 \\ 3 & -1 & -2 \\ -2 & 1.7 & 3 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.5 \\ -1 \\ 1.7 \end{bmatrix}$$

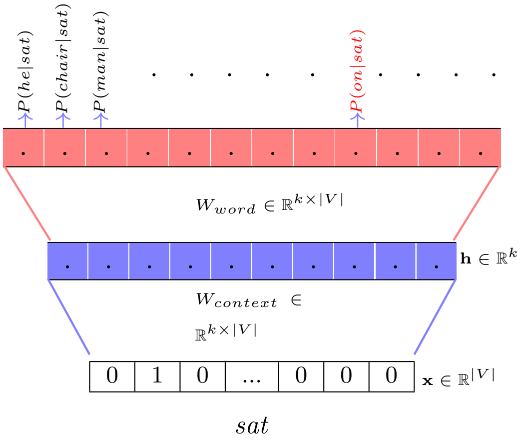
- So when the i^{th} word is present the i^{th} element in the one hot vector is ON and the i^{th} column of $\mathbf{W}_{context}$ gets selected
- In other words, there is a one-to-one correspondence between the words and the column of $\mathbf{W}_{context}$



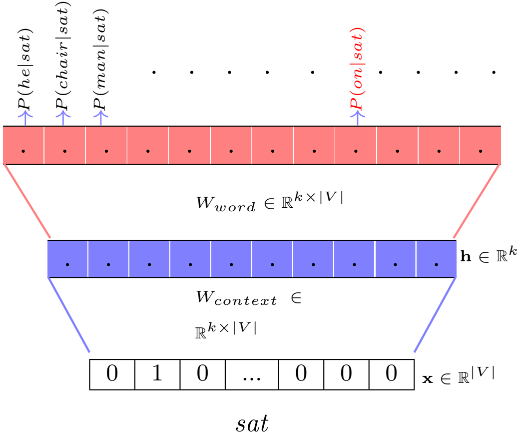
- What is the product $\mathbf{W}_{context}\mathbf{x}$ given that $\mathbf{x}$ is a one hot vector
- It is simply the i-th column of $\mathbf{W}_{context}$

$$\begin{bmatrix} -1 & 0.5 & 2 \\ 3 & -1 & -2 \\ -2 & 1.7 & 3 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.5 \\ -1 \\ 1.7 \end{bmatrix}$$

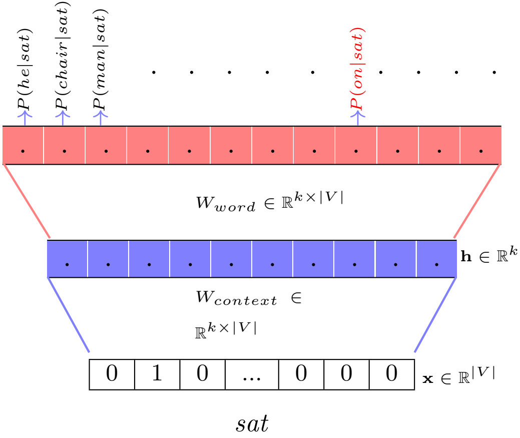
- So when the i^{th} word is present the i^{th} element in the one hot vector is ON and the i^{th} column of $\mathbf{W}_{context}$ gets selected
- In other words, there is a one-to-one correspondence between the words and the column of $\mathbf{W}_{context}$
- More specifically, we can treat the i-th column of $\mathbf{W}_{context}$ as the representation of context i



- How do we obtain $P(on|sat)$? For this multi-class classification problem what is an appropriate output function?

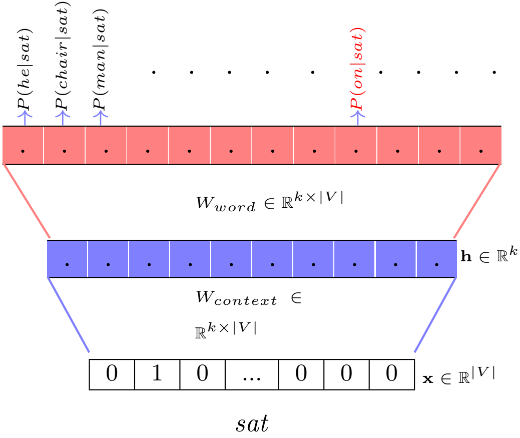


- How do we obtain $P(on|sat)$? For this multi-class classification problem what is an appropriate output function? (**softmax**)



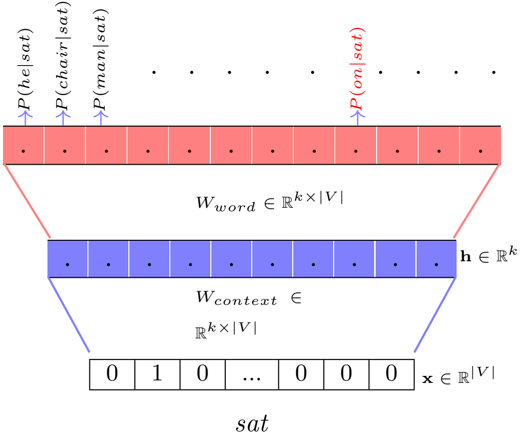
$$P(on|sat) = \frac{e^{(\mathbf{W}_{word}h)[i]}}{\sum_j e^{(\mathbf{W}_{word}h)[j]}}$$

- How do we obtain $P(on|sat)$? For this multi-class classification problem what is an appropriate output function? (**softmax**)



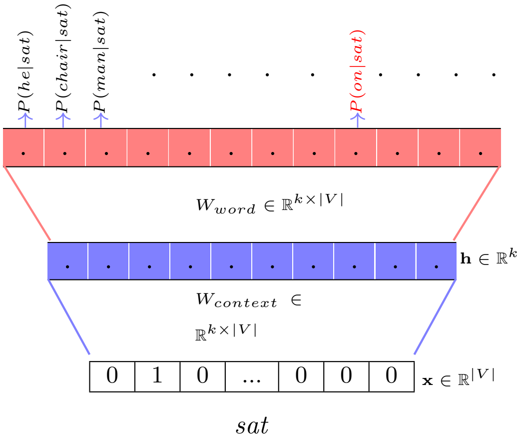
$$P(on|sat) = \frac{e^{(\mathbf{W}_{word}h)[i]}}{\sum_j e^{(\mathbf{W}_{word}h)[j]}}$$

- How do we obtain $P(on|sat)$? For this multi-class classification problem what is an appropriate output function? (**softmax**)
- Therefore, $P(on|sat)$ is proportional to the dot product between j^{th} column of $\mathbf{W}_{context}$ and i^{th} column of $\mathbf{W}_{word}$



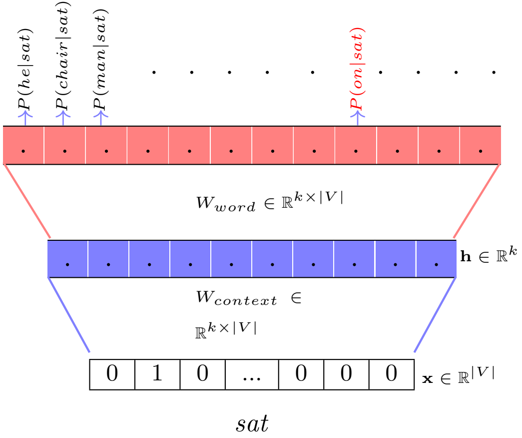
$$P(on|sat) = \frac{e^{(\mathbf{W}_{word}h)[i]}}{\sum_j e^{(\mathbf{W}_{word}h)[j]}}$$

- How do we obtain $P(on|sat)$? For this multi-class classification problem what is an appropriate output function? (**softmax**)
- Therefore, $P(on|sat)$ is proportional to the dot product between j^{th} column of $\mathbf{W}_{context}$ and i^{th} column of $\mathbf{W}_{word}$
- $P(word = i|sat)$ thus depends on the i^{th} column of $\mathbf{W}_{word}$



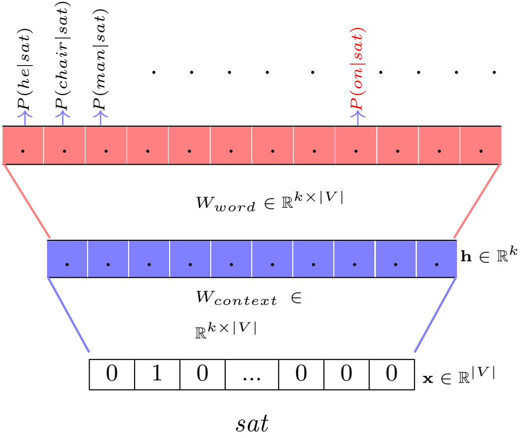
$$P(on|sat) = \frac{e^{(\mathbf{W}_{word}h)[i]}}{\sum_j e^{(\mathbf{W}_{word}h)[j]}}$$

- How do we obtain $P(on|sat)$? For this multi-class classification problem what is an appropriate output function? (**softmax**)
- Therefore, $P(on|sat)$ is proportional to the dot product between j^{th} column of $\mathbf{W}_{context}$ and i^{th} column of $\mathbf{W}_{word}$
- $P(word = i|sat)$ thus depends on the i^{th} column of $\mathbf{W}_{word}$
- We thus treat the i-th column of $\mathbf{W}_{word}$ as the representation of word i



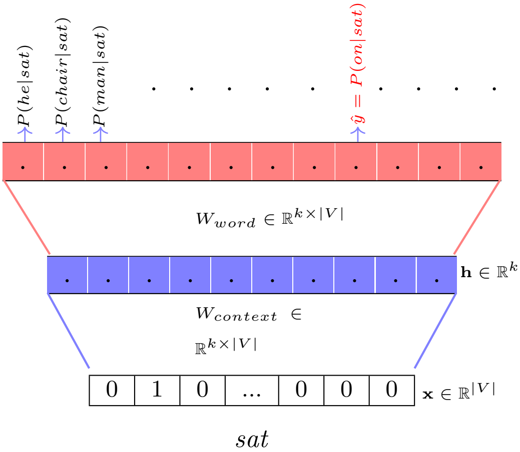
$$P(on|sat) = \frac{e^{(\mathbf{W}_{word}h)[i]}}{\sum_j e^{(\mathbf{W}_{word}h)[j]}}$$

- How do we obtain $P(on|sat)$? For this multi-class classification problem what is an appropriate output function? (**softmax**)
- Therefore, $P(on|sat)$ is proportional to the dot product between j^{th} column of $\mathbf{W}_{context}$ and i^{th} column of $\mathbf{W}_{word}$
- $P(word = i|sat)$ thus depends on the i^{th} column of $\mathbf{W}_{word}$
- We thus treat the i-th column of $\mathbf{W}_{word}$ as the representation of word i
- Hope you see an analogy with SVD! (there we had a different way of learning $\mathbf{W}_{context}$ and $\mathbf{W}_{word}$ but we saw that the i^{th} column of $\mathbf{W}_{word}$ corresponded to the representation of the i^{th} word)

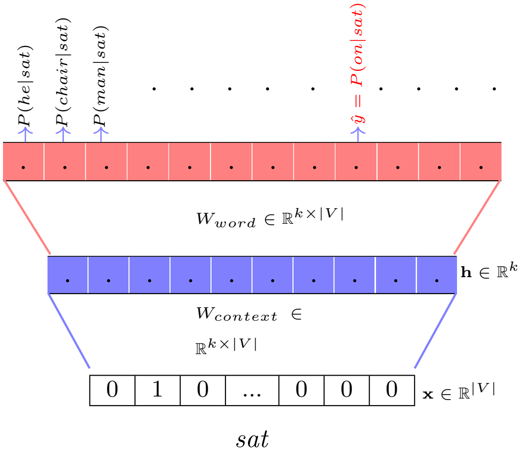


$$P(on|sat) = \frac{e^{(\mathbf{W}_{word}h)[i]}}{\sum_j e^{(\mathbf{W}_{word}h)[j]}}$$

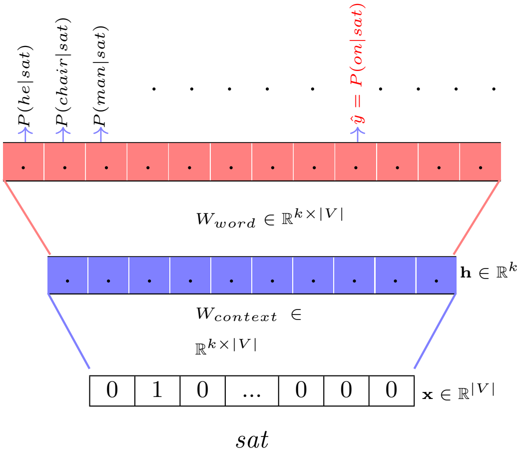
- How do we obtain $P(on|sat)$? For this multi-class classification problem what is an appropriate output function? (**softmax**)
- Therefore, $P(on|sat)$ is proportional to the dot product between j^{th} column of $\mathbf{W}_{context}$ and i^{th} column of $\mathbf{W}_{word}$
- $P(word = i|sat)$ thus depends on the i^{th} column of $\mathbf{W}_{word}$
- We thus treat the i-th column of $\mathbf{W}_{word}$ as the representation of word i
- Hope you see an analogy with SVD! (there we had a different way of learning $\mathbf{W}_{context}$ and $\mathbf{W}_{word}$ but we saw that the i^{th} column of $\mathbf{W}_{word}$ corresponded to the representation of the i^{th} word)
- Now that we understood the interpretation of $\mathbf{W}_{context}$ and $\mathbf{W}_{word}$, our aim now is to learn these parameters



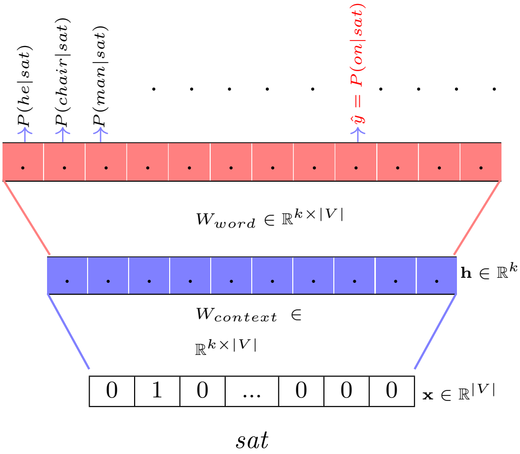
- We denote the context word (sat) by the index c and the correct output word (on) by the index w



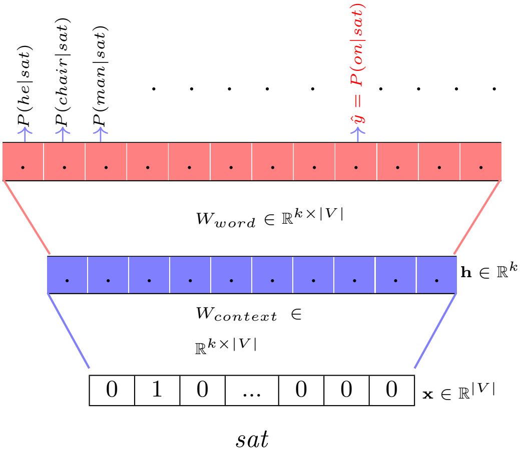
- We denote the context word (sat) by the index c and the correct output word (on) by the index w
- For this multiclass classification problem what is an appropriate output function ($\hat{y} = f(x)$) ?



- We denote the context word (sat) by the index c and the correct output word (on) by the index w
- For this multiclass classification problem what is an appropriate output function ($\hat{y} = f(x)$) ? **softmax**

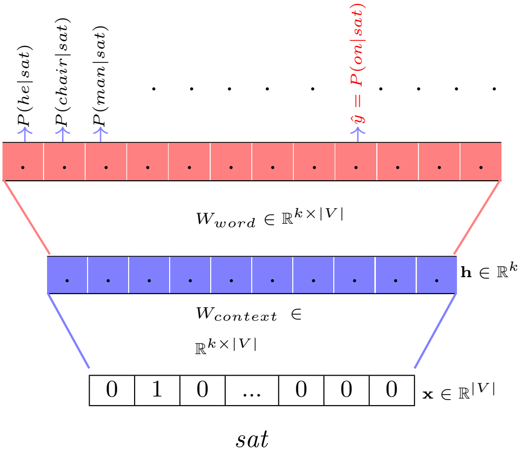


- We denote the context word (sat) by the index c and the correct output word (on) by the index w
- For this multiclass classification problem what is an appropriate output function ($\hat{y} = f(x)$) ? **softmax**
- What is an appropriate loss function?



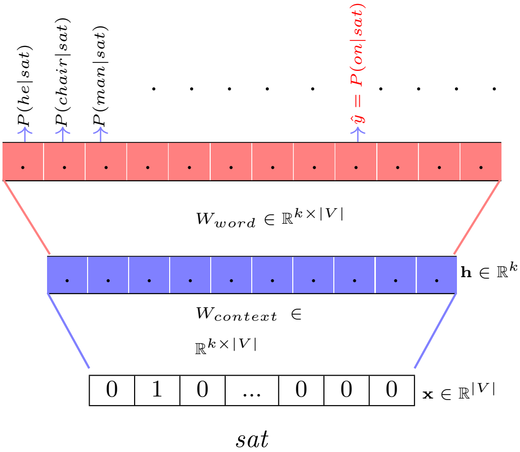
- We denote the context word (sat) by the index c and the correct output word (on) by the index w
- For this multiclass classification problem what is an appropriate output function ($\hat{y} = f(x)$) ? **softmax**
- What is an appropriate loss function? **cross entropy**

$$\mathscr{L}(\theta) = -\log \hat{y}_w = -\log P(w|c)$$



- We denote the context word (sat) by the index c and the correct output word (on) by the index w
- For this multiclass classification problem what is an appropriate output function ($\hat{y} = f(x)$) ? **softmax**
- What is an appropriate loss function? **cross entropy**

$$\mathscr{L}(\theta) = -\log \hat{y}_w = -\log P(w|c)$$
$$h = W_{context} \cdot x_c = u_c$$

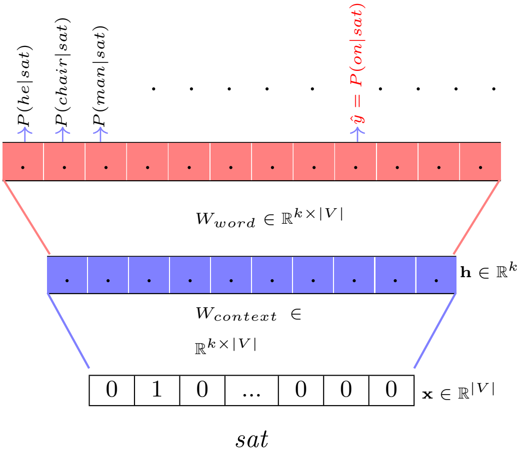


- We denote the context word (sat) by the index c and the correct output word (on) by the index w
- For this multiclass classification problem what is an appropriate output function ($\hat{y} = f(x)$) ? **softmax**
- What is an appropriate loss function? **cross entropy**

$$\mathscr{L}(\theta) = -\log \hat{y}_w = -\log P(w|c)$$

$$h = W_{context} \cdot x_c = u_c$$

$$\hat{y}_w = \frac{exp(u_c \cdot v_w)}{\sum_{w' \in V} exp(u_c \cdot v_{w'})}$$



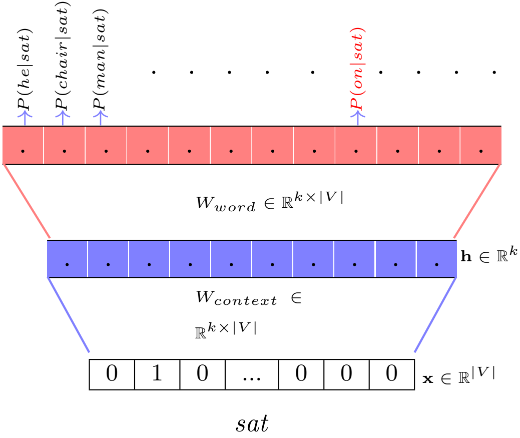
- We denote the context word (sat) by the index c and the correct output word (on) by the index w
- For this multiclass classification problem what is an appropriate output function ($\hat{y} = f(x)$) ? **softmax**
- What is an appropriate loss function? **cross entropy**

$$\mathscr{L}(\theta) = -\log \hat{y}_w = -\log P(w|c)$$

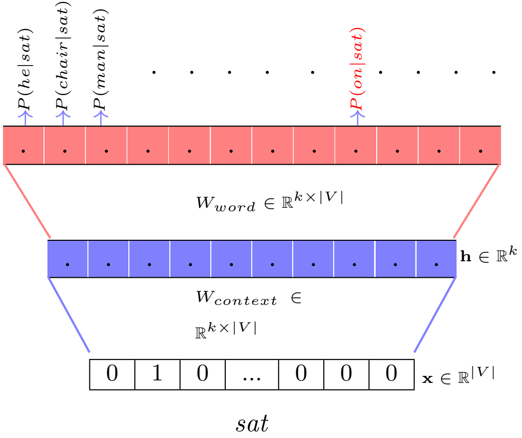
$$h = W_{context} \cdot x_c = u_c$$

$$\hat{y}_w = \frac{exp(u_c \cdot v_w)}{\sum_{w' \in V} exp(u_c \cdot v_{w'})}$$

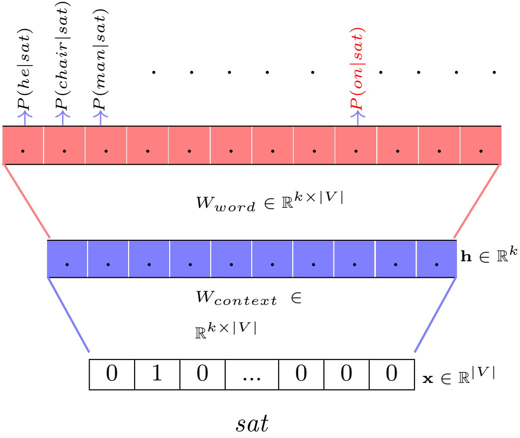
u_c is the column of $W_{context}$ corresponding to context c and v_w is the column of W_{word} corresponding to context w



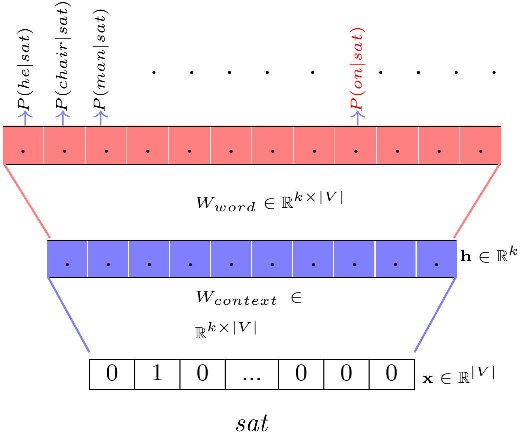
- How do we train this simple feed forward neural network?



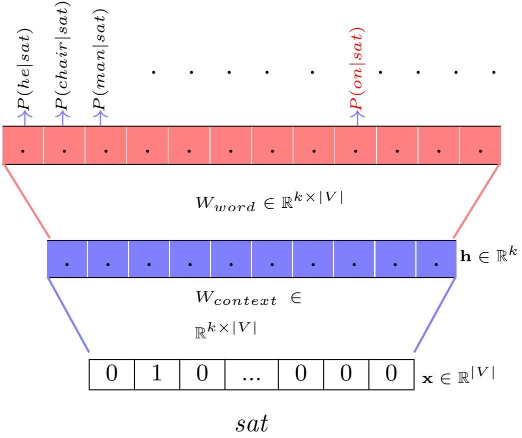
- How do we train this simple feed forward neural network? backpropagation



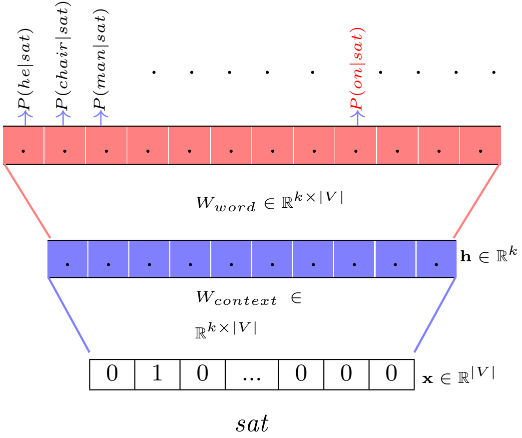
- How do we train this simple feed forward neural network? backpropagation
- Let us consider one input-output pair (c, w) and see the update rule for v_w



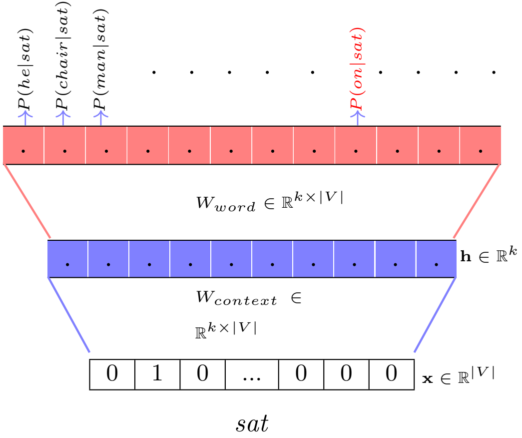
$$\mathscr{L}(\theta) = -\log \hat{y}_w$$



$$\mathscr{L}(\theta) = -\log \hat{y}_w$$
$$= -\log \frac{exp(u_c \cdot v_w)}{\sum_{w' \in V} exp(u_c \cdot v_{w'})}$$



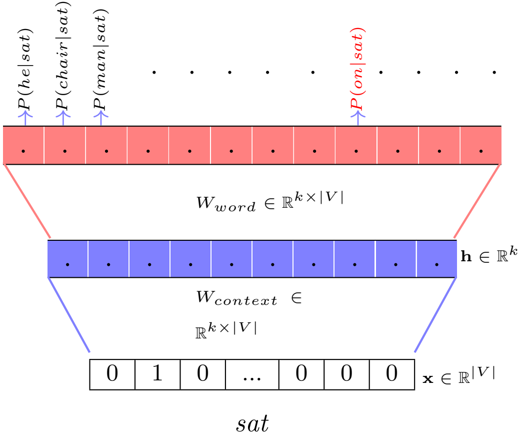
$$\begin{aligned}
\mathscr{L}(\theta) &= -\log \hat{y}_w \\
&= -\log \frac{exp(u_c \cdot v_w)}{\sum_{w' \in V} exp(u_c \cdot v_{w'})} \\
&= -(u_c \cdot v_w - \log \sum_{w' \in V} exp(u_c \cdot v_{w'}))
\end{aligned}$$



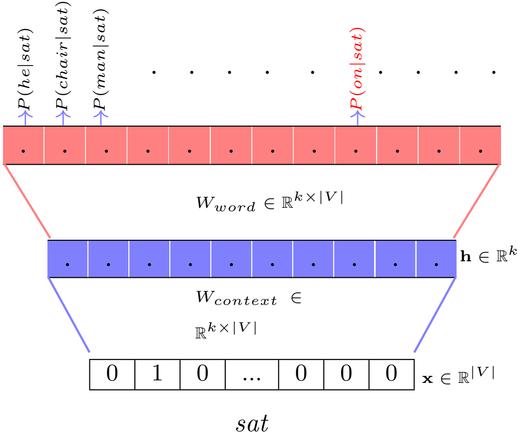
$$\nabla_{v_w} = -\frac{\partial}{\partial v_w}\mathscr{L}(\theta)$$

$$\begin{aligned}\mathscr{L}(\theta) &= -\log \hat{y}_w \\ &= -\log \frac{exp(u_c \cdot v_w)}{\sum_{w' \in V} exp(u_c \cdot v_{w'})} \\ &= -(u_c \cdot v_w - \log \sum_{w' \in V} exp(u_c \cdot v_{w'}))\end{aligned}$$

$$\nabla_{v_w} = -(u_c - \frac{exp(u_c \cdot v_w)}{\sum_{w' \in V} exp(u_c \cdot v_{w'})} \cdot u_c)$$



$$
\begin{aligned}
\mathscr{L}(\theta) &= -\log \hat{y}_w \\
&= -\log \frac{exp(u_c \cdot v_w)}{\sum_{w' \in V} exp(u_c \cdot v_{w'})} \\
&= -(u_c \cdot v_w - \log \sum_{w' \in V} exp(u_c \cdot v_{w'})) \\
\nabla_{v_w} &= -(u_c - \frac{exp(u_c \cdot v_w)}{\sum_{w' \in V} exp(u_c \cdot v_{w'})} \cdot u_c) \\
&= -u_c(1 - \hat{y}_w)
\end{aligned}
$$



$$\mathscr{L}(\theta) = -\log \hat{y}_w$$

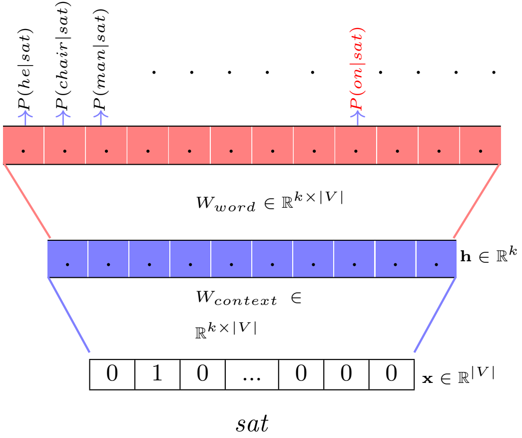
$$= -\log \frac{exp(u_c \cdot v_w)}{\sum_{w' \in V} exp(u_c \cdot v_{w'})}$$

$$= -(u_c \cdot v_w - \log \sum_{w' \in V} exp(u_c \cdot v_{w'}))$$

$$\nabla_{v_w} = -(u_c - \frac{exp(u_c \cdot v_w)}{\sum_{w' \in V} exp(u_c \cdot v_{w'})} \cdot u_c)$$

$$= -u_c(1 - \hat{y}_w)$$

And the update rule would be

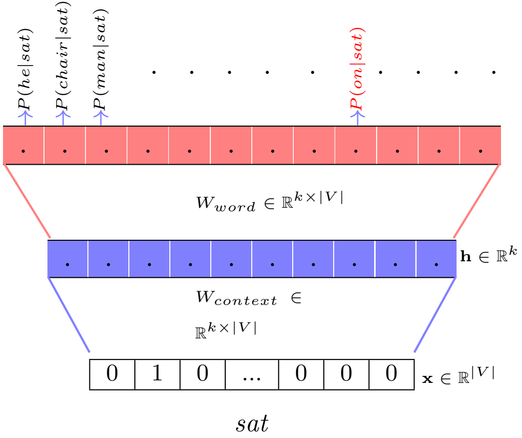


$$
\begin{aligned}
\mathscr{L}(\theta) &= -\log \hat{y}_w \\
&= -\log \frac{exp(u_c \cdot v_w)}{\sum_{w' \in V} exp(u_c \cdot v_{w'})} \\
&= -(u_c \cdot v_w - \log \sum_{w' \in V} exp(u_c \cdot v_{w'}))
\end{aligned}
$$

$$
\begin{aligned}
\nabla_{v_w} &= -(u_c - \frac{exp(u_c \cdot v_w)}{\sum_{w' \in V} exp(u_c \cdot v_{w'})} \cdot u_c) \\
&= -u_c(1 - \hat{y}_w)
\end{aligned}
$$

And the update rule would be

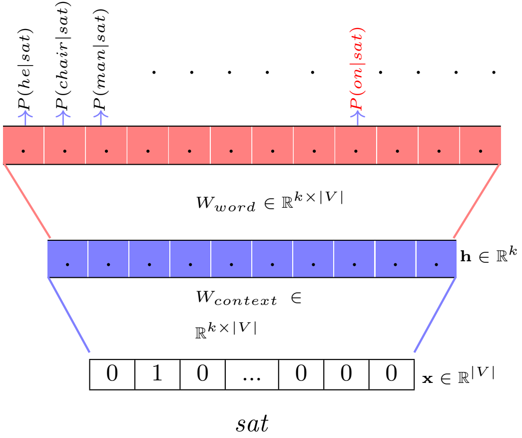
$$v_w = v_w - \eta \nabla_{v_w}$$



$$\begin{aligned}
\mathscr{L}(\theta) &= -\log \hat{y}_w \\
&= -\log \frac{exp(u_c \cdot v_w)}{\sum_{w' \in V} exp(u_c \cdot v_{w'})} \\
&= -(u_c \cdot v_w - \log \sum_{w' \in V} exp(u_c \cdot v_{w'})) \\
\nabla_{v_w} &= -(u_c - \frac{exp(u_c \cdot v_w)}{\sum_{w' \in V} exp(u_c \cdot v_{w'})} \cdot u_c) \\
&= -u_c(1 - \hat{y}_w)
\end{aligned}$$

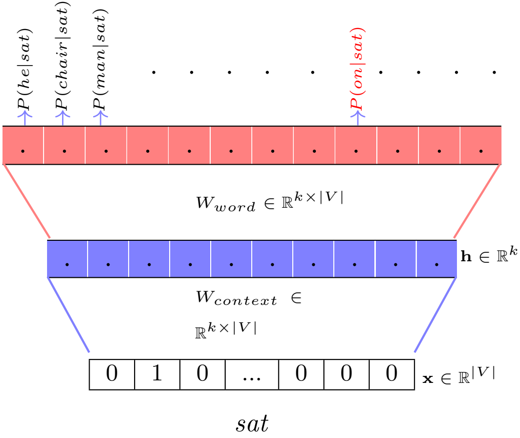
And the update rule would be

$$\begin{aligned}
v_w &= v_w - \eta \nabla_{v_w} \\
&= v_w + \eta u_c(1 - \hat{y}_w)
\end{aligned}$$



- This update rule has a nice interpretation

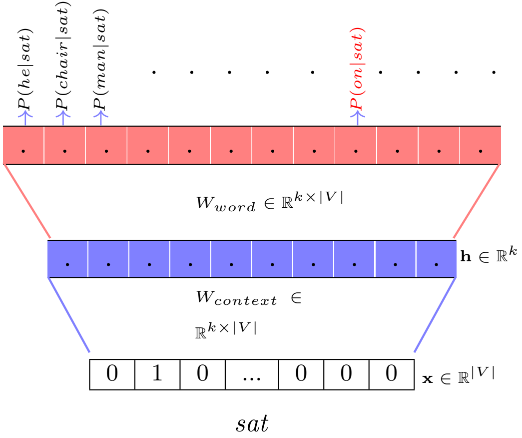
$$v_w = v_w + \eta u_c(1 - \hat{y}_w)$$



- This update rule has a nice interpretation

$$v_w = v_w + \eta u_c(1 - \hat{y}_w)$$

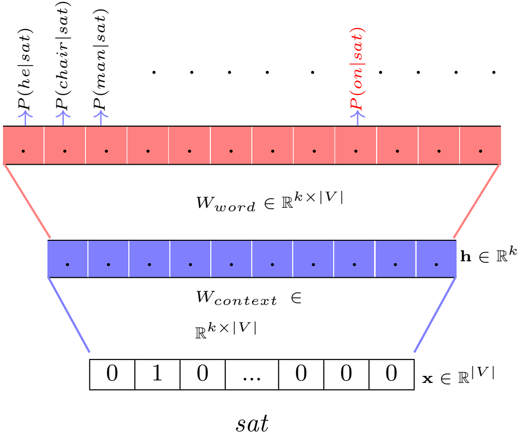
- If $\hat{y}_w \to 1$ then we are already predicting the right word and v_w will not be updated



- This update rule has a nice interpretation

$$v_w = v_w + \eta u_c(1 - \hat{y}_w)$$

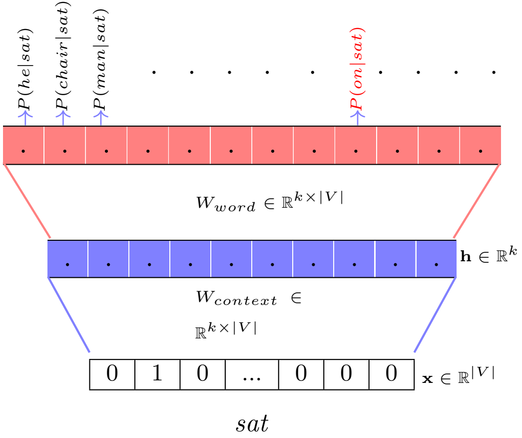
- If $\hat{y}_w \to 1$ then we are already predicting the right word and v_w will not be updated
- If $\hat{y}_w \to 0$ then v_w gets updated by adding a fraction of u_c to it



- This update rule has a nice interpretation

$$v_w = v_w + \eta u_c(1 - \hat{y}_w)$$

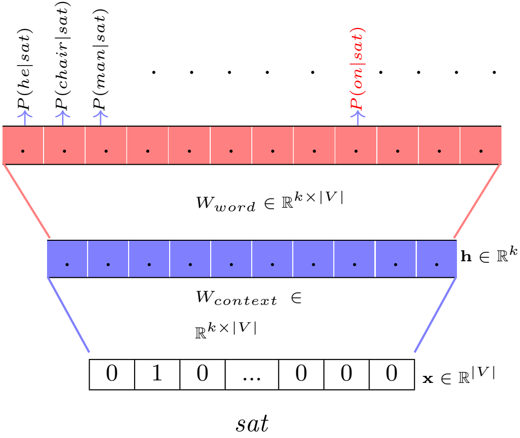
- If $\hat{y}_w \to 1$ then we are already predicting the right word and v_w will not be updated
- If $\hat{y}_w \to 0$ then v_w gets updated by adding a fraction of u_c to it
- This increases the cosine similarity between v_w and u_c (How? Refer to slide 38 of Lecture 2)



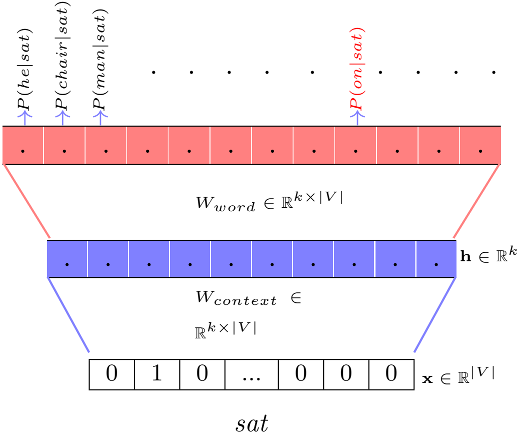
- This update rule has a nice interpretation

$$v_w = v_w + \eta u_c(1 - \hat{y}_w)$$

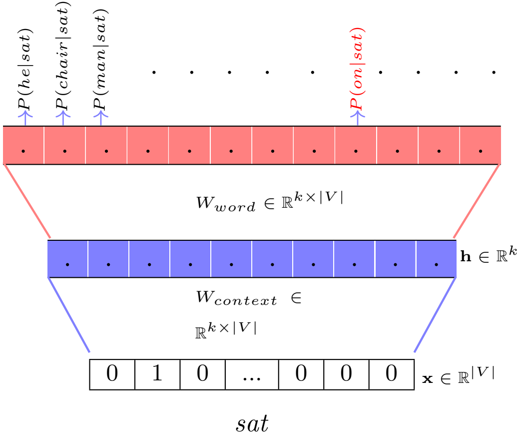
- If $\hat{y}_w \to 1$ then we are already predicting the right word and v_w will not be updated
- If $\hat{y}_w \to 0$ then v_w gets updated by adding a fraction of u_c to it
- This increases the cosine similarity between v_w and u_c (How? Refer to slide 38 of Lecture 2)
- The training objective ensures that the cosine similarity between word (v_w) and context word (u_c) is maximized



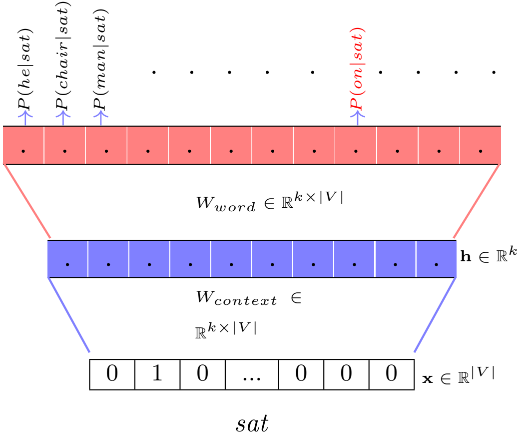
- What happens to the representations of two words w and w' which tend to appear in similar context (c)



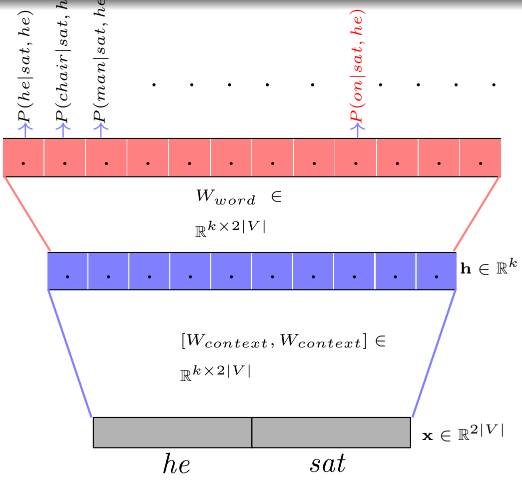
- What happens to the representations of two words w and w' which tend to appear in similar context (c)
- The training ensures that both v_w and v'_w have a high cosine similarity with u_c and hence transitively (intuitively) ensures that v_w and v'_w have a high cosine similarity with each other



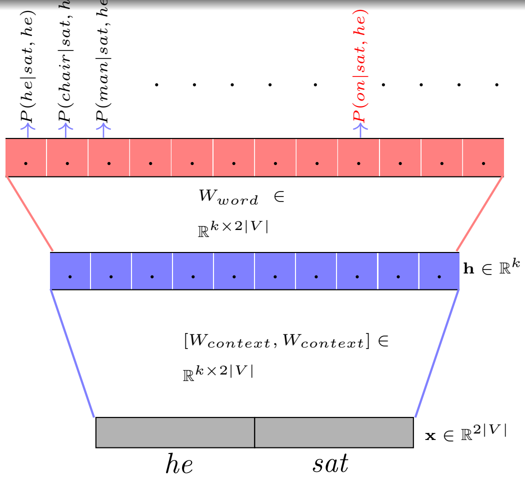
- What happens to the representations of two words w and w' which tend to appear in similar context (c)
- The training ensures that both v_w and v'_w have a high cosine similarity with u_c and hence transitively (intuitively) ensures that v_w and v'_w have a high cosine similarity with each other
- This is only an intuition (reasonable)



- What happens to the representations of two words w and w' which tend to appear in similar context (c)
- The training ensures that both v_w and v'_w have a high cosine similarity with u_c and hence transitively (intuitively) ensures that v_w and v'_w have a high cosine similarity with each other
- This is only an intuition (reasonable)
- Haven't come across a formal proof for this!

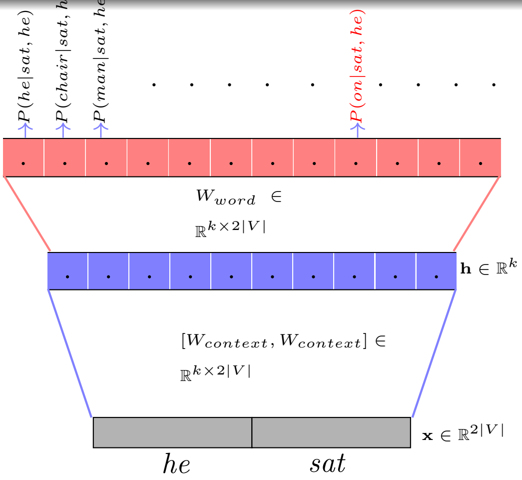


- In practice, instead of window size of 1 it is common to use a window size of d



- In practice, instead of window size of 1 it is common to use a window size of d
- So now,

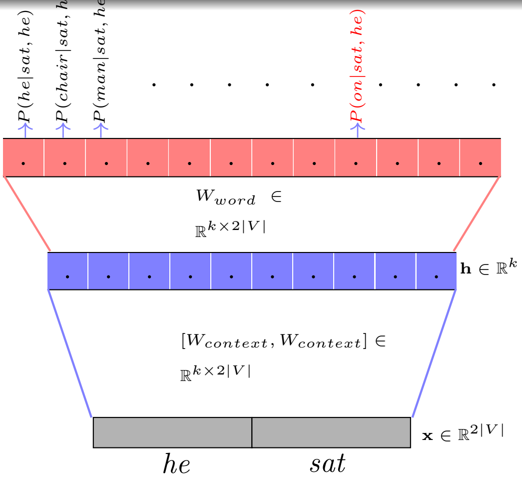
$$h = \sum_{i=1}^{d-1} u_{c_i}$$



- In practice, instead of window size of 1 it is common to use a window size of d
- So now,

$$h = \sum_{i=1}^{d-1} u_{c_i}$$

- $[W_{context}, W_{context}]$ just means that we are stacking 2 copies of $W_{context}$ matrix

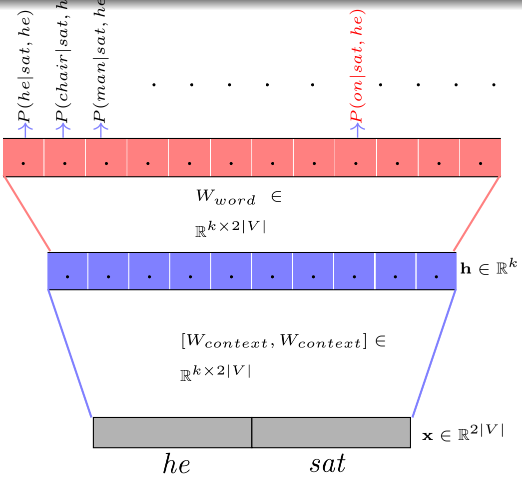


- In practice, instead of window size of 1 it is common to use a window size of d
- So now,

$$h = \sum_{i=1}^{d-1} u_{c_i}$$

- $[W_{context}, W_{context}]$ just means that we are stacking 2 copies of $W_{context}$ matrix

$$\begin{bmatrix} -1 & 0.5 & 2 & \\ 3 & -1 & -2 & \\ -2 & 1.7 & 3 & \end{bmatrix} \begin{array}{c} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \end{array} \begin{array}{l} \\ \} \, sat \\ \\ \\ \\ \} \, he \end{array}$$

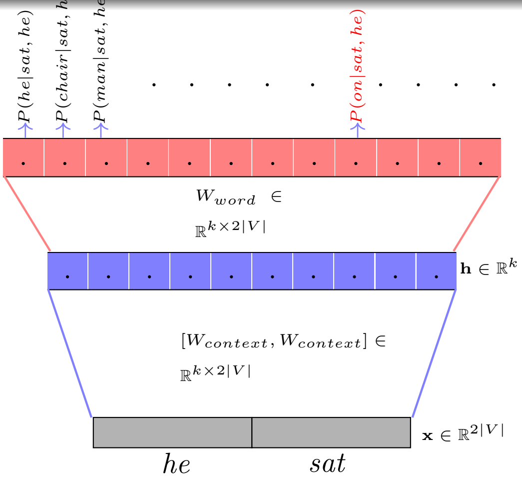


- In practice, instead of window size of 1 it is common to use a window size of d
- So now,

$$h = \sum_{i=1}^{d-1} u_{c_i}$$

- $[W_{context}, W_{context}]$ just means that we are stacking 2 copies of $W_{context}$ matrix

$$\begin{bmatrix} -1 & 0.5 & 2 & -1 & 0.5 & 2 \\ 3 & -1 & -2 & 3 & -1 & -2 \\ -2 & 1.7 & 3 & -2 & 1.7 & 3 \end{bmatrix} \begin{array}{l} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \end{array} \begin{array}{l} \\ \} \; sat \\ \\ \\ \\ \} \; he \end{array}$$



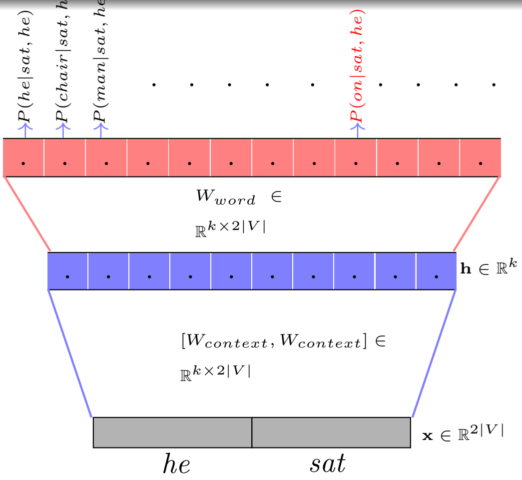
- In practice, instead of window size of 1 it is common to use a window size of d
- So now,

$$h = \sum_{i=1}^{d-1} u_{c_i}$$

- $[W_{context}, W_{context}]$ just means that we are stacking 2 copies of $W_{context}$ matrix

$$\begin{bmatrix} -1 & \boxed{0.5} & 2 & -1 & 0.5 & \boxed{2} \\ 3 & \boxed{-1} & -2 & 3 & -1 & \boxed{-2} \\ -2 & \boxed{1.7} & 3 & -2 & 1.7 & \boxed{3} \end{bmatrix} \begin{array}{l} \begin{bmatrix} 0 \\ \boxed{1} \\ 0 \\ 0 \\ 0 \\ \boxed{1} \end{bmatrix} \end{array} \begin{array}{l} \\ \} \, sat \\ \\ \\ \\ \} \, he \end{array}$$

$$= \begin{bmatrix} 2.5 \\ -3 \\ 4.7 \end{bmatrix}$$



- In practice, instead of window size of 1 it is common to use a window size of d
- So now,

$$h = \sum_{i=1}^{d-1} u_{c_i}$$

- $[W_{context}, W_{context}]$ just means that we are stacking 2 copies of $W_{context}$ matrix

$$\begin{bmatrix} -1 & 0.5 & 2 & -1 & 0.5 & 2 \\ 3 & -1 & -2 & 3 & -1 & -2 \\ -2 & 1.7 & 3 & -2 & 1.7 & 3 \end{bmatrix} \begin{array}{cl} \left[\begin{array}{c} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{array}\right] & \begin{array}{l} \\ \} \, sat \\ \\ \\ \\ \} \, he \end{array} \end{array}$$

$$= \begin{bmatrix} 2.5 \\ -3 \\ 4.7 \end{bmatrix}$$

- The resultant product would simply be the sum of the columns corresponding to 'sat' and 'he'

- Of course in practice we will not do this expensive matrix multiplication

- Of course in practice we will not do this expensive matrix multiplication
- If 'he' is i^{th} word in the vocabulary and *sat* is the j^{th} word then we will simply access columns $\mathbf{W}[i:]$ and $\mathbf{W}[j:]$ and add them

- Now what happens during backpropagation

- Now what happens during backpropagation
- Recall that

$$h = \sum_{i=1}^{d-1} u_{c_i}$$

- Now what happens during backpropagation
- Recall that

$$h = \sum_{i=1}^{d-1} u_{c_i}$$

- and

$$P(on|sat, he) = \frac{e^{(w_{word}h)[k]}}{\sum_j e^{(w_{word}h)[j]}}$$

- Now what happens during backpropagation
- Recall that

$$h = \sum_{i=1}^{d-1} u_{c_i}$$

- and

$$P(on|sat, he) = \frac{e^{(w_{word}h)[k]}}{\sum_j e^{(w_{word}h)[j]}}$$

- where ‘k’ is the index of the word ‘on’

- Now what happens during backpropagation
- Recall that

$$h = \sum_{i=1}^{d-1} u_{c_i}$$

- and

$$P(on|sat, he) = \frac{e^{(w_{word}h)[k]}}{\sum_j e^{(w_{word}h)[j]}}$$

- where 'k' is the index of the word 'on'
- The loss function depends on $\{W_{word}, u_{c_1}, u_{c_2}, \ldots, u_{c_{d-1}}\}$ and all these parameters will get updated during backpropogation

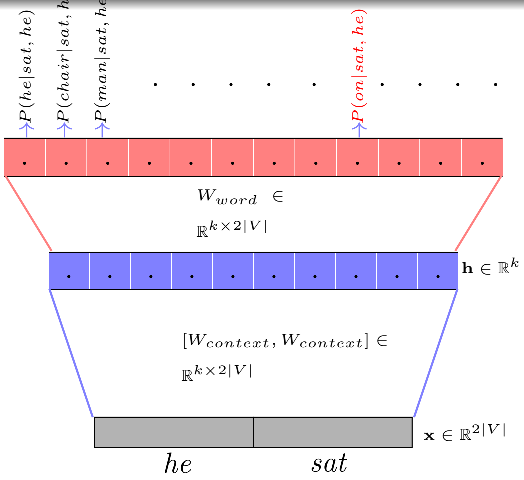
- Now what happens during backpropagation
- Recall that

$$h = \sum_{i=1}^{d-1} u_{c_i}$$

- and

$$P(on|sat, he) = \frac{e^{(w_{word}h)[k]}}{\sum_j e^{(w_{word}h)[j]}}$$

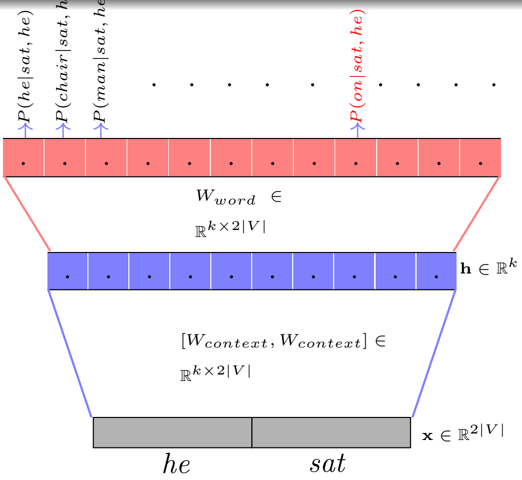
- where 'k' is the index of the word 'on'
- The loss function depends on $\{W_{word}, u_{c_1}, u_{c_2}, \ldots, u_{c_{d-1}}\}$ and all these parameters will get updated during backpropogation
- Try deriving the update rule for v_w now and see how it differs from the one we derived before



Some problems:

- Notice that the softmax function at the output is computationally very expensive

$$\hat{y}_w = \frac{exp(u_c \cdot v_w)}{\sum_{w' \in V} exp(u_c \cdot v_{w'})}$$

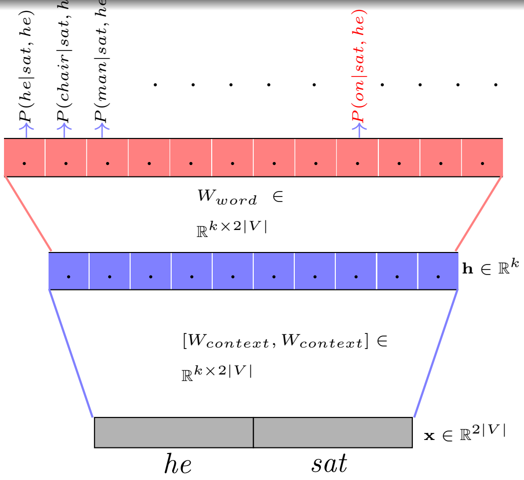


Some problems:

- Notice that the softmax function at the output is computationally very expensive

$$\hat{y}_w = \frac{exp(u_c \cdot v_w)}{\sum_{w' \in V} exp(u_c \cdot v_{w'})}$$

- The denominator requires a summation over all words in the vocabulary



Some problems:

- Notice that the softmax function at the output is computationally very expensive

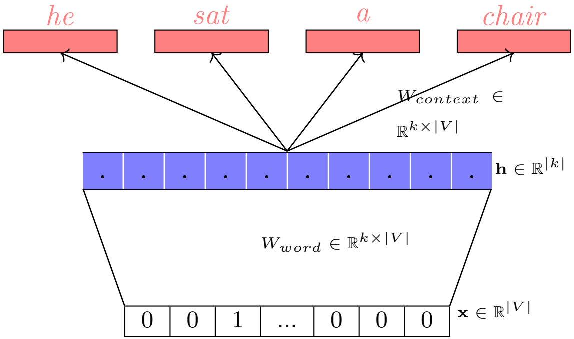
$$\hat{y}_w = \frac{exp(u_c \cdot v_w)}{\sum_{w' \in V} exp(u_c \cdot v_{w'})}$$

- The denominator requires a summation over all words in the vocabulary
- We will revisit this issue soon

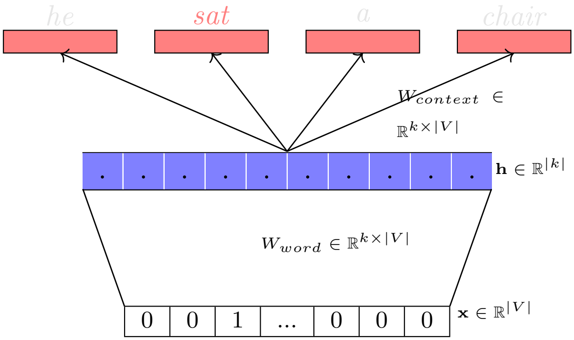
Module 10.5: Skip-gram model

- The model that we just saw is called the continuous bag of words model (it predicts an output word give a bag of context words)

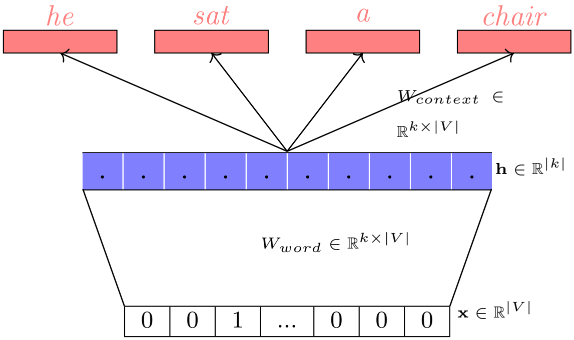
- The model that we just saw is called the continuous bag of words model (it predicts an output word give a bag of context words)
- We will now see the skip gram model (which predicts context words given an input word)



- Notice that the role of *context* and *word* has changed now

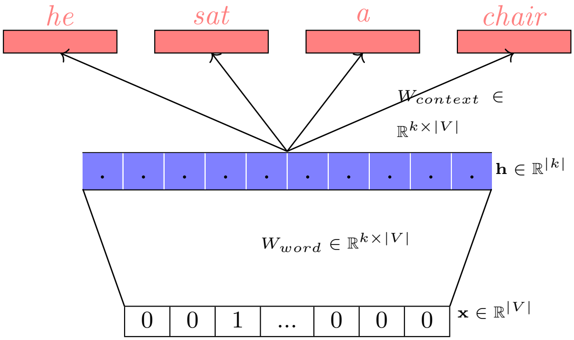


- Notice that the role of *context* and *word* has changed now
- In the simple case when there is only one *context* word, we will arrive at the same update rule for u_c as we did for v_w earlier



- Notice that the role of *context* and *word* has changed now
- In the simple case when there is only one *context* word, we will arrive at the same update rule for u_c as we did for v_w earlier
- Notice that even when we have multiple context words the loss function would just be a summation of many cross entropy errors

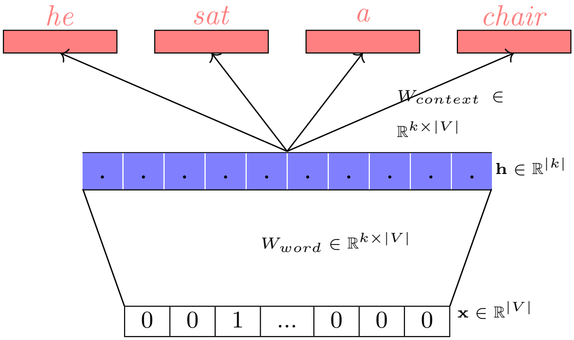
$$\mathscr{L}(\theta) = -\sum_{i=1}^{d-1} \log \hat{y}_{w_i}$$



- Notice that the role of *context* and *word* has changed now
- In the simple case when there is only one *context* word, we will arrive at the same update rule for u_c as we did for v_w earlier
- Notice that even when we have multiple context words the loss function would just be a summation of many cross entropy errors

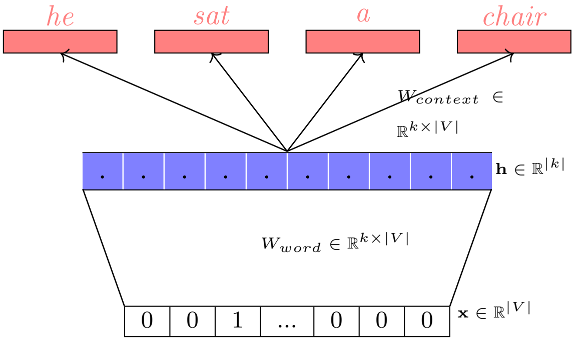
$$\mathscr{L}(\theta) = -\sum_{i=1}^{d-1} \log \hat{y}_{w_i}$$

- Typically, we predict context words on both sides of the given word



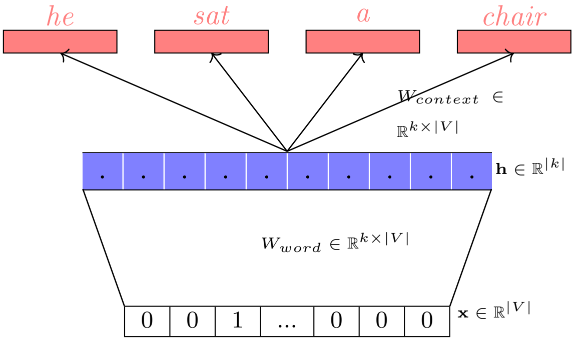
Some problems

- Same as bag of words



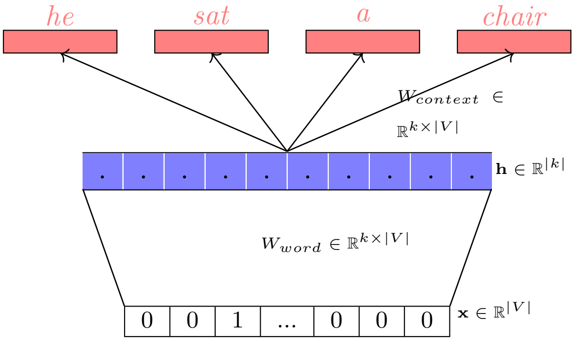
Some problems

- Same as bag of words
- The softmax function at the output is computationally expensive



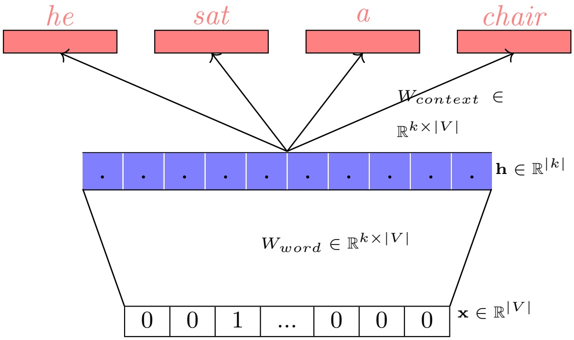
Some problems

- Same as bag of words
- The softmax function at the output is computationally expensive
- Solution 1: Use negative sampling



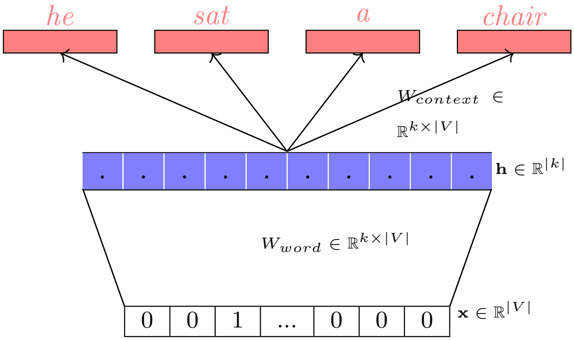
Some problems

- Same as bag of words
- The softmax function at the output is computationally expensive
- Solution 1: Use negative sampling
- Solution 2: Use contrastive estimation



Some problems

- Same as bag of words
- The softmax function at the output is computationally expensive
- Solution 1: Use negative sampling
- Solution 2: Use contrastive estimation
- Solution 3: Use hierarchical softmax



Some problems

- Same as bag of words
- The softmax function at the output is computationally expensive
- Solution 1: Use negative sampling
- Solution 2: Use contrastive estimation
- Solution 3: Use hierarchical softmax

- D = [(sat, on), (sat, a), (sat, chair), (on, a), (on,chair), (a,chair), (on,sat), (a, sat), (chair,sat), (a, on), (chair, on), (chair, a)]

- Let D be the set of all correct (w, c) pairs in the corpus

- D = [(sat, on), (sat, a), (sat, chair), (on, a), (on,chair), (a,chair), (on,sat), (a, sat), (chair,sat), (a, on), (chair, on), (chair, a)]
- D' = [(sat, oxygen), (sat, magic), (chair, sad), (chair, walking)]

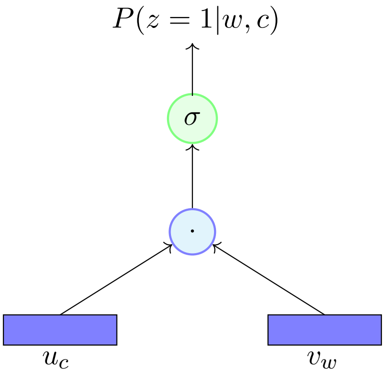
- Let D be the set of all correct (w, c) pairs in the corpus
- Let D' be the set of all incorrect (w, r) pairs in the corpus

- D = [(sat, on), (sat, a), (sat, chair), (on, a), (on,chair), (a,chair), (on,sat), (a, sat), (chair,sat), (a, on), (chair, on), (chair, a)]
- D' = [(sat, oxygen), (sat, magic), (chair, sad), (chair, walking)]

- Let D be the set of all correct (w, c) pairs in the corpus
- Let D' be the set of all incorrect (w, r) pairs in the corpus
- D' can be constructed by randomly sampling a context word r which has never appeared with w and creating a pair (w, r)

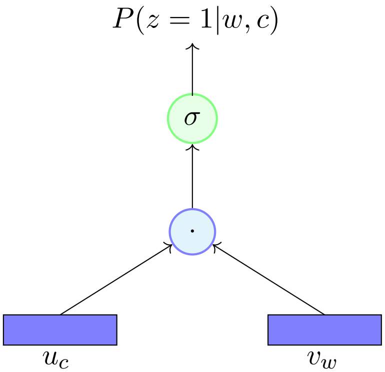
- D = [(sat, on), (sat, a), (sat, chair), (on, a), (on,chair), (a,chair), (on,sat), (a, sat), (chair,sat), (a, on), (chair, on), (chair, a)]
- D' = [(sat, oxygen), (sat, magic), (chair, sad), (chair, walking)]

- Let D be the set of all correct (w, c) pairs in the corpus
- Let D' be the set of all incorrect (w, r) pairs in the corpus
- D' can be constructed by randomly sampling a context word r which has never appeared with w and creating a pair (w, r)
- As before let v_w be the representation of the word w and u_c be the representation of the context word c



- For a given $(w, c) \in D$ we are interested in maximizing

$$p(z = 1|w, c)$$

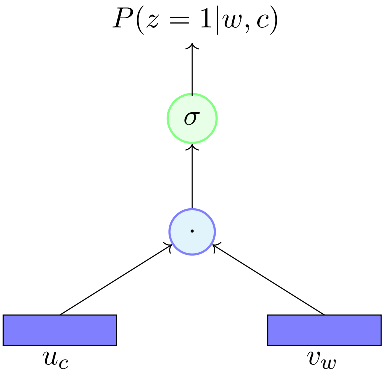


- For a given $(w, c) \in D$ we are interested in maximizing

$$p(z = 1|w, c)$$

- Let us model this probability by

$$\begin{aligned} p(z = 1|w, c) &= \sigma(u_c^T v_w) \\ &= \frac{1}{1 + e^{-u_c^T v_w}} \end{aligned}$$



- For a given $(w, c) \in D$ we are interested in maximizing

$$p(z = 1|w, c)$$

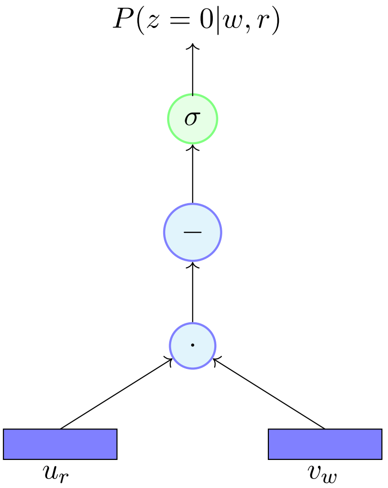
- Let us model this probability by

$$p(z = 1|w, c) = \sigma(u_c^T v_w)$$
$$= \frac{1}{1 + e^{-u_c^T v_w}}$$

- Considering all $(w, c) \in D$, we are interested in

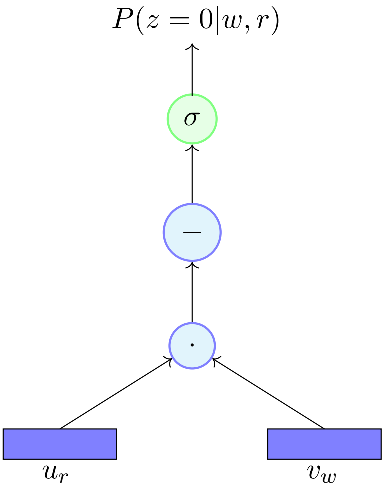
$$\underset{\theta}{maximize} \prod_{(w,c) \in D} p(z = 1|w, c)$$

where θ is the word representation (v_w) and context representation (u_c) for all words in our corpus



- For $(w, r) \in D'$ we are interested in maximizing

$$p(z = 0|w, r)$$

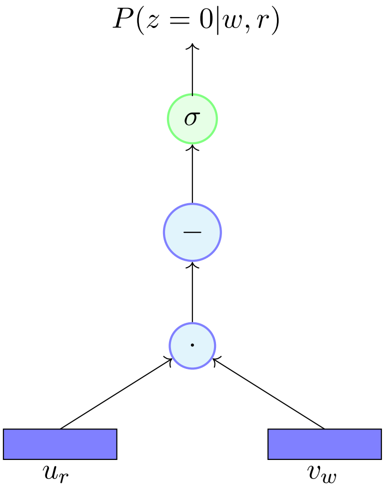


- For $(w, r) \in D'$ we are interested in maximizing

$$p(z = 0|w, r)$$

- Again we model this as

$$p(z = 0|w, r) = 1 - \sigma(u_r^T v_w)$$

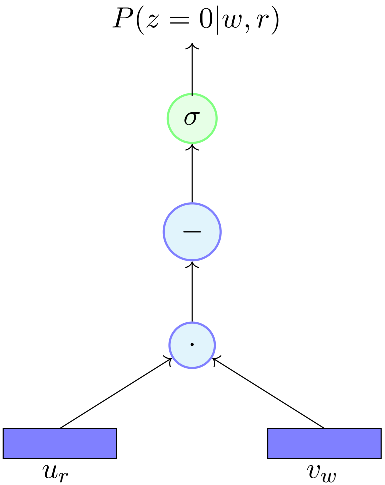


- For $(w, r) \in D'$ we are interested in maximizing

$$p(z = 0|w, r)$$

- Again we model this as

$$\begin{aligned} p(z = 0|w, r) &= 1 - \sigma(u_r^T v_w) \\ &= 1 - \frac{1}{1 + e^{-v_r^T v_w}} \end{aligned}$$

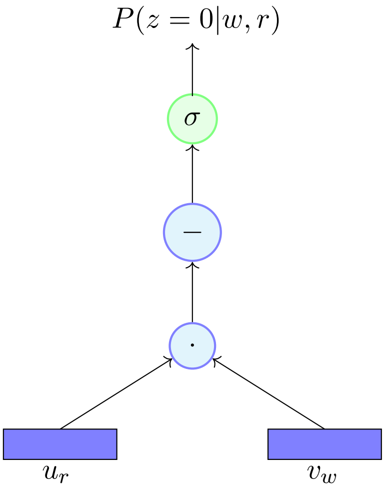


- For $(w, r) \in D'$ we are interested in maximizing

$$p(z = 0|w, r)$$

- Again we model this as

$$\begin{aligned} p(z = 0|w, r) &= 1 - \sigma(u_r^T v_w) \\ &= 1 - \frac{1}{1 + e^{-v_r^T v_w}} \\ &= \frac{1}{1 + e^{u_r^T v_w}} \end{aligned}$$

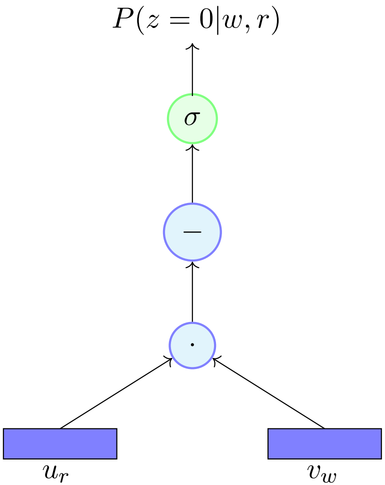


- For $(w, r) \in D'$ we are interested in maximizing

$$p(z = 0|w, r)$$

- Again we model this as

$$\begin{aligned} p(z = 0|w, r) &= 1 - \sigma(u_r^T v_w) \\ &= 1 - \frac{1}{1 + e^{-v_r^T v_w}} \\ &= \frac{1}{1 + e^{u_r^T v_w}} = \sigma(-u_r^T v_w) \end{aligned}$$



- For $(w,r) \in D'$ we are interested in maximizing

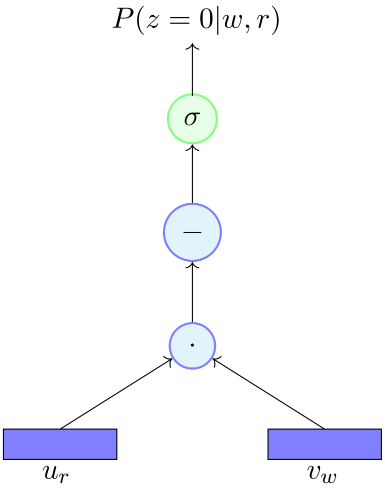
$$p(z=0|w,r)$$

- Again we model this as

$$\begin{aligned} p(z=0|w,r) &= 1 - \sigma(u_r^T v_w) \\ &= 1 - \frac{1}{1+e^{-v_r^T v_w}} \\ &= \frac{1}{1+e^{u_r^T v_w}} = \sigma(-u_r^T v_w) \end{aligned}$$

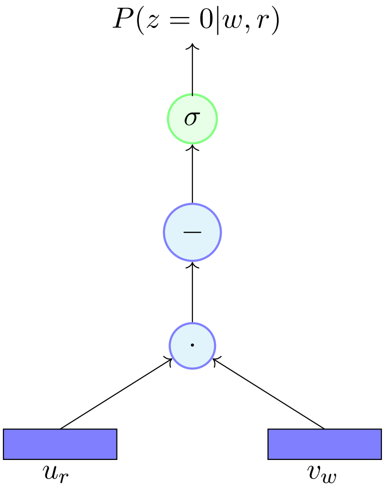
- Considering all $(w,r) \in D'$, we are interested in

$$\underset{\theta}{maximize} \prod_{(w,r)\in D'} p(z=0|w,r)$$



- Combining the two we get:

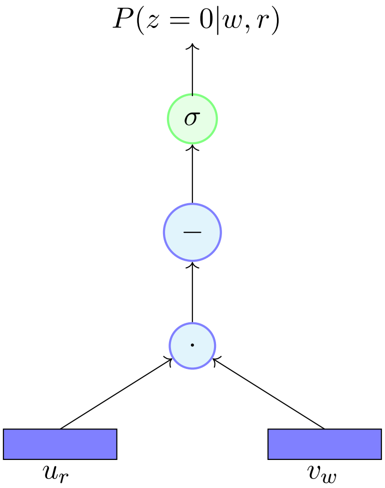
$$\underset{\theta}{maximize} \prod_{(w,c)\in D} p(z = 1|w, c) \prod_{(w,r)\in D'} p(z = 0|w, r)$$



- Combining the two we get:

$$\underset{\theta}{maximize} \prod_{(w,c)\in D} p(z = 1|w, c) \prod_{(w,r)\in D'} p(z = 0|w, r)$$

$$= \underset{\theta}{maximize} \prod_{(w,c)\in D} p(z = 1|w, c) \prod_{(w,r)\in D'} (1 - p(z = 1|w, r))$$



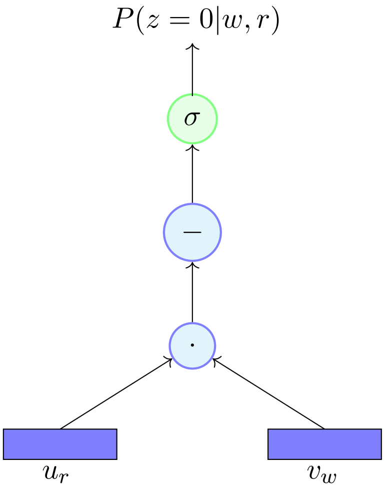
- Combining the two we get:

$$\underset{\theta}{maximize} \prod_{(w,c)\in D} p(z = 1|w, c) \prod_{(w,r)\in D'} p(z = 0|w, r)$$

$$= \underset{\theta}{maximize} \prod_{(w,c)\in D} p(z = 1|w, c) \prod_{(w,r)\in D'} (1 - p(z = 1|w, r))$$

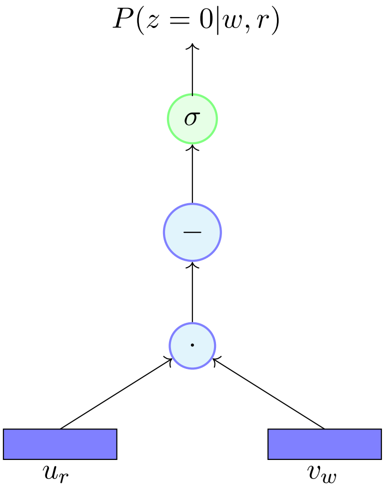
$$= \underset{\theta}{maximize} \sum_{(w,c)\in D} \log p(z = 1|w, c)$$

$$+ \sum_{(w,r)\in D'} \log(1 - p(z = 1|w, r))$$



- Combining the two we get:

$$
\begin{aligned}
&\underset{\theta}{maximize} \prod_{(w,c)\in D} p(z=1|w,c) \prod_{(w,r)\in D'} p(z=0|w,r) \\
=&\underset{\theta}{maximize} \prod_{(w,c)\in D} p(z=1|w,c) \prod_{(w,r)\in D'} (1-p(z=1|w,r)) \\
=&\underset{\theta}{maximize} \sum_{(w,c)\in D} \log p(z=1|w,c) \\
&\qquad + \sum_{(w,r)\in D'} \log(1-p(z=1|w,r)) \\
=&\underset{\theta}{maximize} \sum_{(w,c)\in D} \log \frac{1}{1+e^{-v_c^T v_w}} + \sum_{(w,r)\in D'} \log \frac{1}{1+e^{v_r^T v_w}}
\end{aligned}
$$



- Combining the two we get:

$$\underset{\theta}{maximize} \prod_{(w,c)\in D} p(z = 1|w, c) \prod_{(w,r)\in D'} p(z = 0|w, r)$$

$$= \underset{\theta}{maximize} \prod_{(w,c)\in D} p(z = 1|w, c) \prod_{(w,r)\in D'} (1 - p(z = 1|w, r))$$

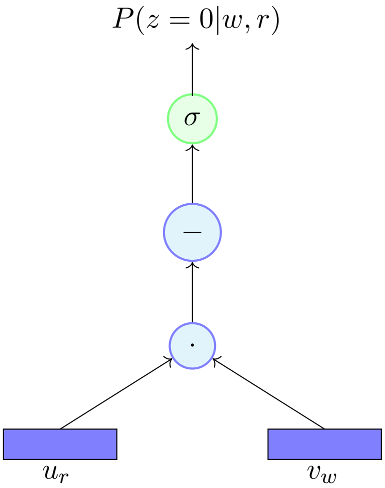
$$= \underset{\theta}{maximize} \sum_{(w,c)\in D} \log p(z = 1|w, c)$$

$$+ \sum_{(w,r)\in D'} \log(1 - p(z = 1|w, r))$$

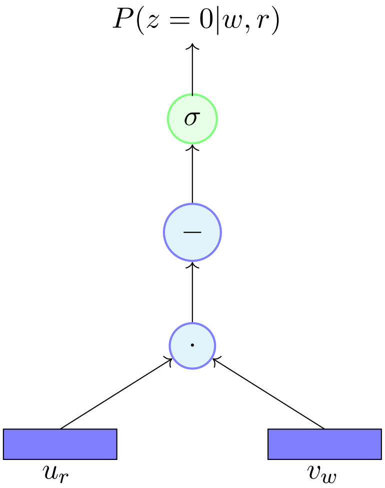
$$= \underset{\theta}{maximize} \sum_{(w,c)\in D} \log \frac{1}{1 + e^{-v_c^T v_w}} + \sum_{(w,r)\in D'} \log \frac{1}{1 + e^{v_r^T v_w}}$$

$$= \underset{\theta}{maximize} \sum_{(w,c)\in D} \log \sigma(v_c^T v_w) + \sum_{(w,r)\in D'} \log \sigma(-v_r^T v_w)$$

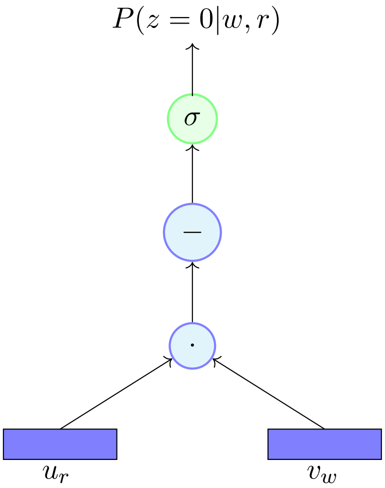
where $\sigma(x) = \frac{1}{1+e^{-x}}$



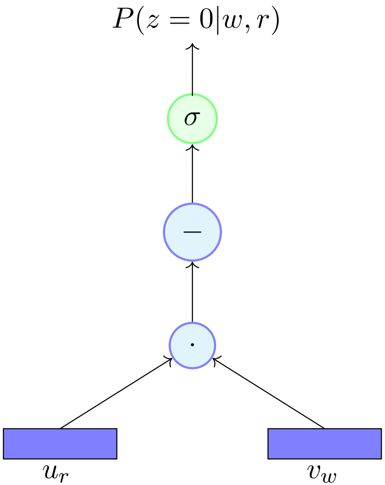
- In the original paper, *Mikolov et. al.* sample k negative (w, r) pairs for every positive (w, c) pairs



- In the original paper, *Mikolov et. al.* sample k negative (w,r) pairs for every positive (w,c) pairs
- The size of D' is thus k times the size of D



- In the original paper, *Mikolov et. al.* sample k negative (w, r) pairs for every positive (w, c) pairs
- The size of D' is thus k times the size of D
- The random context word is drawn from a modified unigram distribution



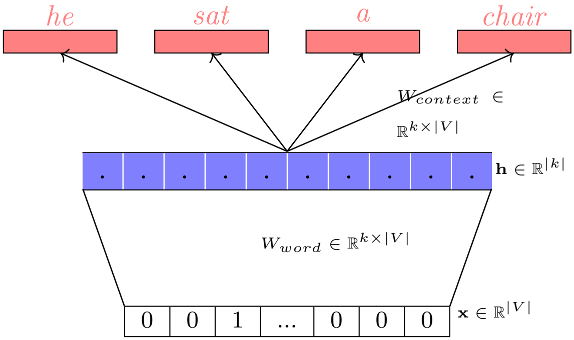
- In the original paper, *Mikolov et. al.* sample k negative (w, r) pairs for every positive (w, c) pairs
- The size of D' is thus k times the size of D
- The random context word is drawn from a modified unigram distribution

$$r \sim p(r)^{\frac{3}{4}}$$

$$r \sim \frac{count(r)^{\frac{3}{4}}}{N}$$

N = total number of words in the corpus

Module 10.6: Contrastive estimation



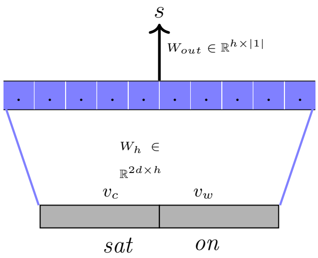
Some problems

- Same as bag of words
- The softmax function at the output is computationally expensive
- Solution 1: Use negative sampling
- Solution 2: Use contrastive estimation
- Solution 3: Use hierarchical softmax

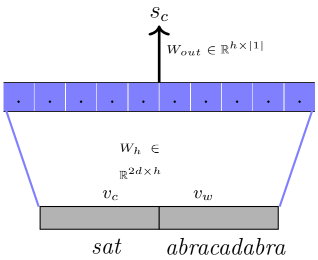
Positive: *He sat on a chair*

Negative: *He sat abracadabra a chair*

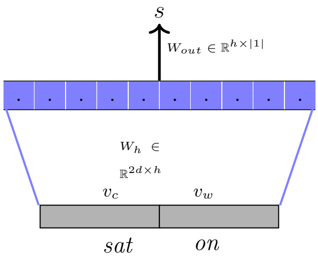
Positive: *He sat on a chair*



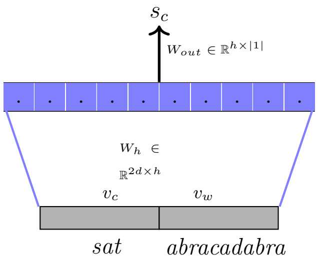
Negative: *He sat abracadabra a chair*



Positive: *He sat on a chair*

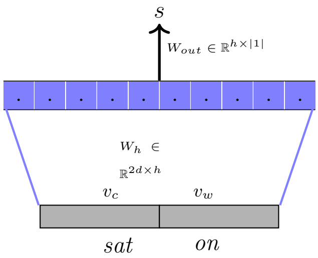


Negative: *He sat abracadabra a chair*

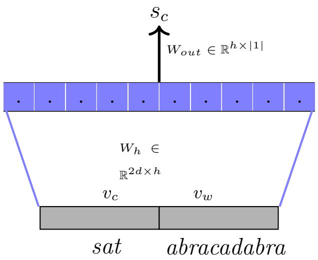


- We would like s to be greater than s_c

Positive: *He sat on a chair*

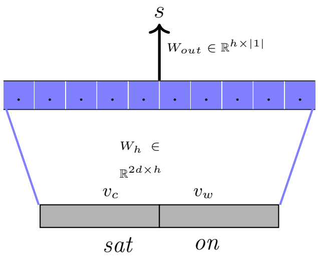


Negative: *He sat abracadabra a chair*

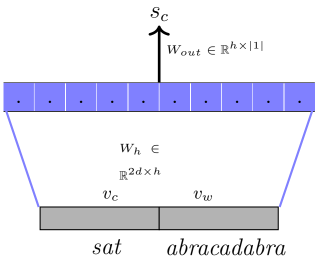


- We would like s to be greater than s_c
- Okay, so let us try to maximize $s - s_c$

Positive: *He sat on a chair*

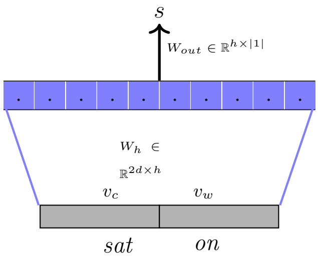


Negative: *He sat abracadabra a chair*



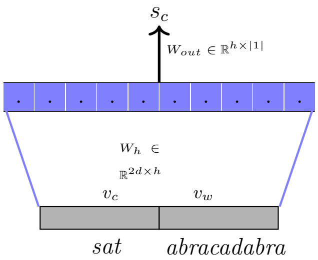
- We would like s to be greater than s_c
- Okay, so let us try to maximize $s - s_c$
- But we would like the difference to be at least m

Positive: *He sat on a chair*



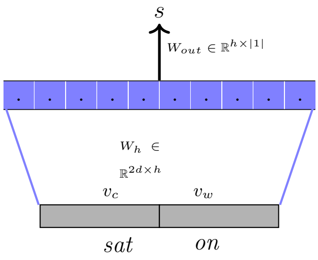
- We would like s to be greater than s_c
- Okay, so let us try to maximize $s - s_c$
- But we would like the difference to be at least m

Negative: *He sat abracadabra a chair*

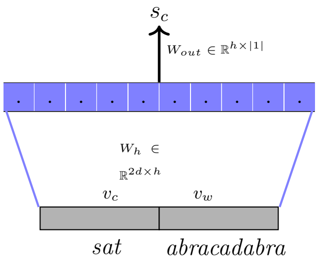


- So we can maximize $s - (s_c + m)$

Positive: *He sat on a chair*



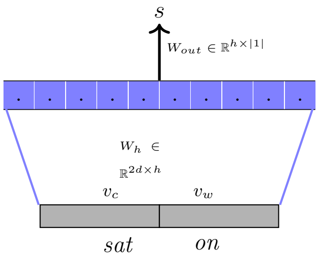
Negative: *He sat abracadabra a chair*



- We would like s to be greater than s_c
- Okay, so let us try to maximize $s - s_c$
- But we would like the difference to be at least m

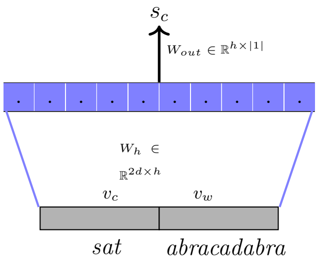
- So we can maximize $s - (s_c + m)$
- What if $s > s_c + m$

Positive: *He sat on a chair*



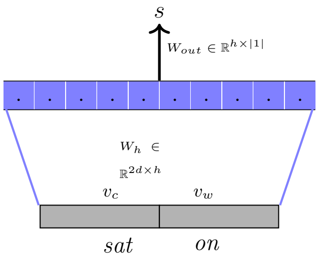
- We would like s to be greater than s_c
- Okay, so let us try to maximize $s - s_c$
- But we would like the difference to be at least m

Negative: *He sat abracadabra a chair*

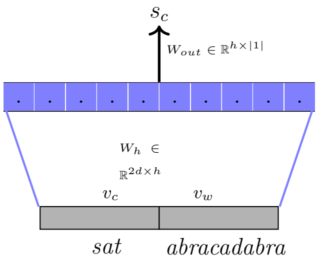


- So we can maximize $s - (s_c + m)$
- What if $s > s_c + m$ *(don't do any thing)*

Positive: *He sat on a chair*



Negative: *He sat abracadabra a chair*

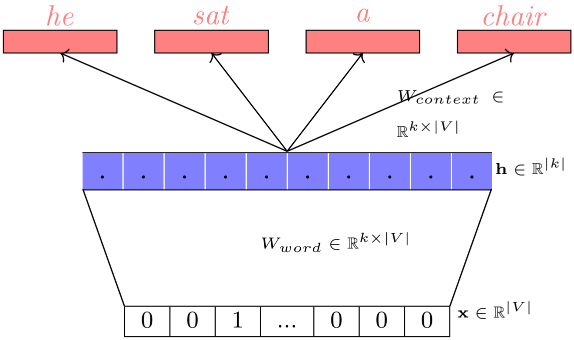


- We would like s to be greater than s_c
- Okay, so let us try to maximize $s - s_c$
- But we would like the difference to be at least m

- So we can maximize $s - (s_c + m)$
- What if $s > s_c + m$ *(don't do any thing)*

$$\text{maximize} \quad max(0, s - (s_c + m))$$

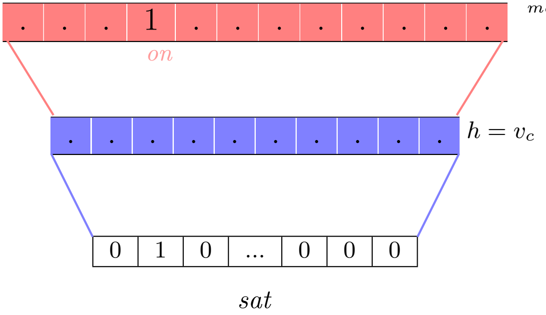
Module 10.7: Hierarchical softmax



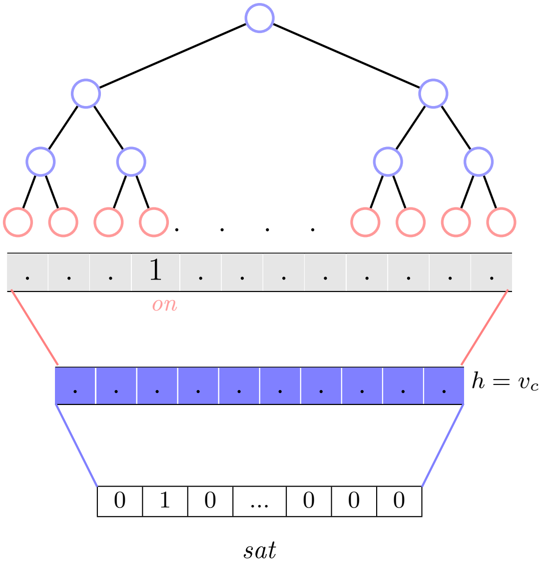
Some problems

- Same as bag of words
- The softmax function at the output is computationally expensive
- Solution 1: Use negative sampling
- Solution 2: Use contrastive estimation
- Solution 3: Use hierarchical softmax

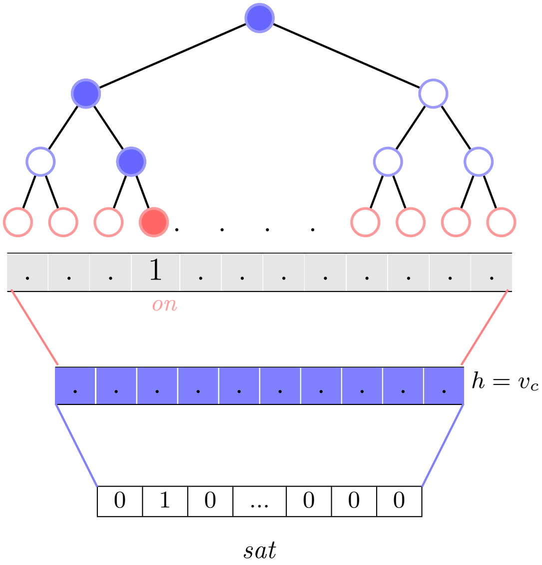
- Construct a binary tree such that there are $|V|$ leaf nodes each corresponding to one word in the vocabulary



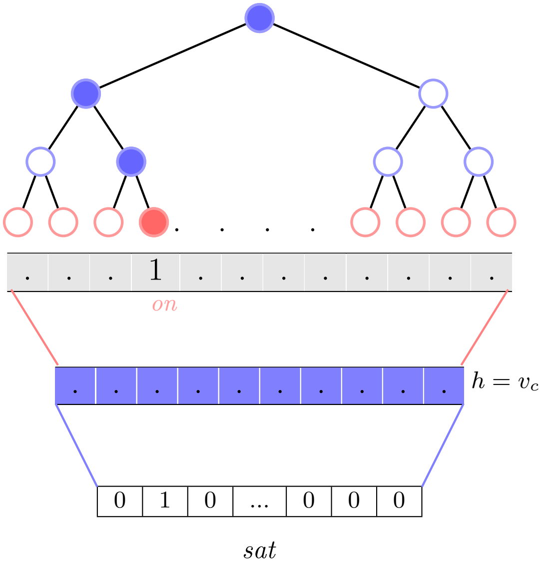
$$max \frac{e^{v_c^T u_w}}{\sum_{|V|} e^{v_c^T u_w}}$$



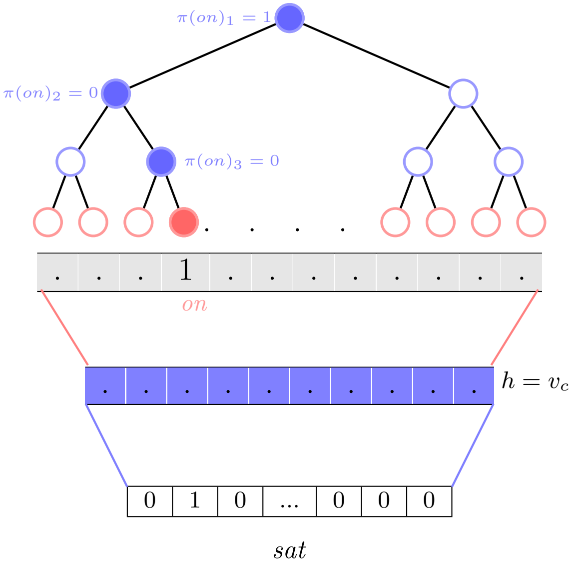
- Construct a binary tree such that there are $|V|$ leaf nodes each corresponding to one word in the vocabulary



- Construct a binary tree such that there are $|V|$ leaf nodes each corresponding to one word in the vocabulary
- There exists a unique path from the root node to a leaf node.

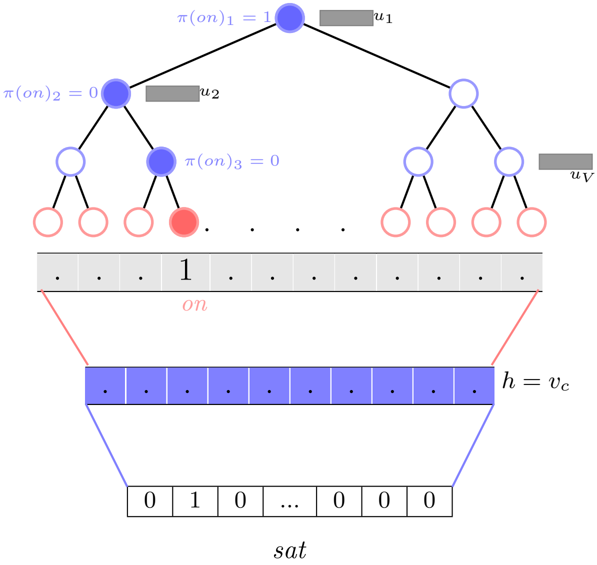


- Construct a binary tree such that there are $|V|$ leaf nodes each corresponding to one word in the vocabulary
- There exists a unique path from the root node to a leaf node.
- Let $l(w_1)$, $l(w_2)$, ..., $l(w_p)$ be the nodes on the path from root to w



- Construct a binary tree such that there are $|V|$ leaf nodes each corresponding to one word in the vocabulary
- There exists a unique path from the root node to a leaf node.
- Let $l(w_1)$, $l(w_2)$, ..., $l(w_p)$ be the nodes on the path from root to w
- Let $\pi(w)$ be a binary vector such that:

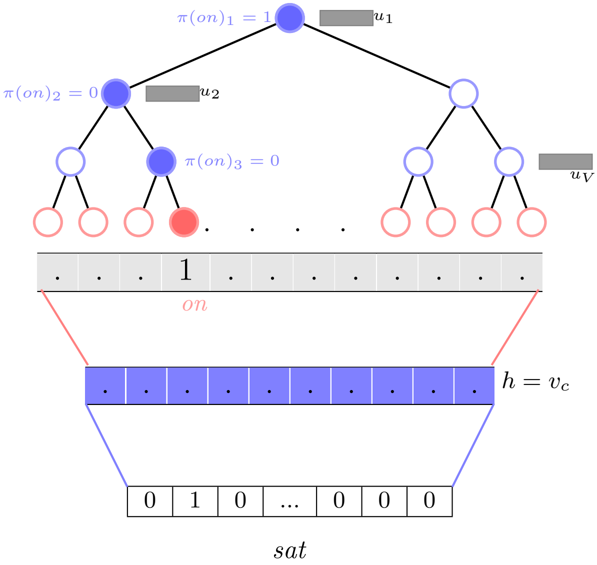
$$\begin{aligned} \pi(w)_k &= 1 \quad \text{path branches left at node } l(w_k) \\ &= 0 \quad \textit{otherwise} \end{aligned}$$



- Construct a binary tree such that there are $|V|$ leaf nodes each corresponding to one word in the vocabulary
- There exists a unique path from the root node to a leaf node.
- Let $l(w_1)$, $l(w_2)$, ..., $l(w_p)$ be the nodes on the path from root to w
- Let $\pi(w)$ be a binary vector such that:

$$\begin{aligned} \pi(w)_k &= 1 \quad \text{path branches left at node } l(w_k) \\ &= 0 \quad \textit{otherwise} \end{aligned}$$

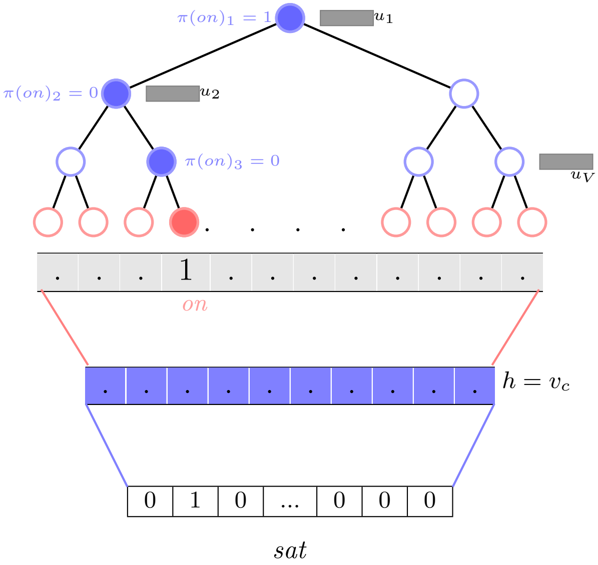
- Finally each internal node is associated with a vector u_i



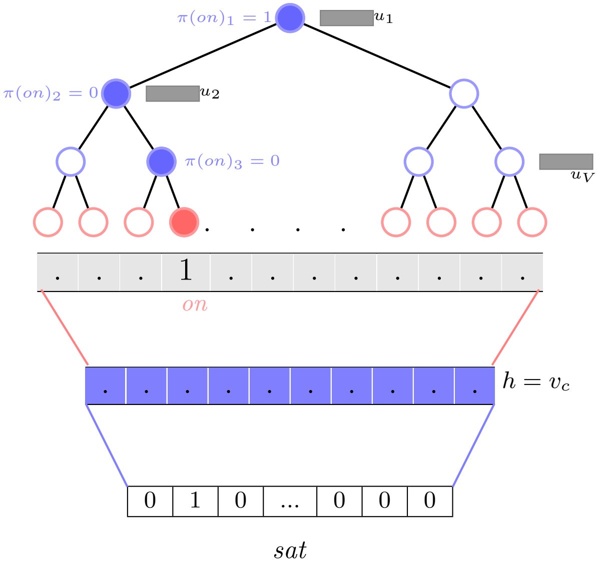
- Construct a binary tree such that there are $|V|$ leaf nodes each corresponding to one word in the vocabulary
- There exists a unique path from the root node to a leaf node.
- Let $l(w_1)$, $l(w_2)$, ..., $l(w_p)$ be the nodes on the path from root to w
- Let $\pi(w)$ be a binary vector such that:

$$
\begin{aligned}
\pi(w)_k &= 1 \quad \text{path branches left at node } l(w_k) \\
&= 0 \quad \textit{otherwise}
\end{aligned}
$$

- Finally each internal node is associated with a vector u_i
- So the parameters of the module are $\mathbf{W}_{context}$ and $u_1, u_2, \ldots, u_v$ (in effect, we have the same number of parameters as before)

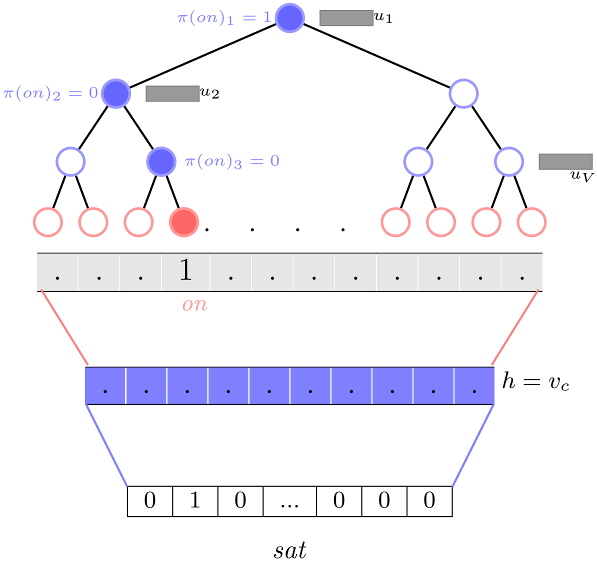


- For a given pair (w, c) we are interested in the probability $p(w|v_c)$



- For a given pair (w, c) we are interested in the probability $p(w|v_c)$
- We model this probability as

$$p(w|v_c) = \prod_k (\pi(w_k)|v_c)$$

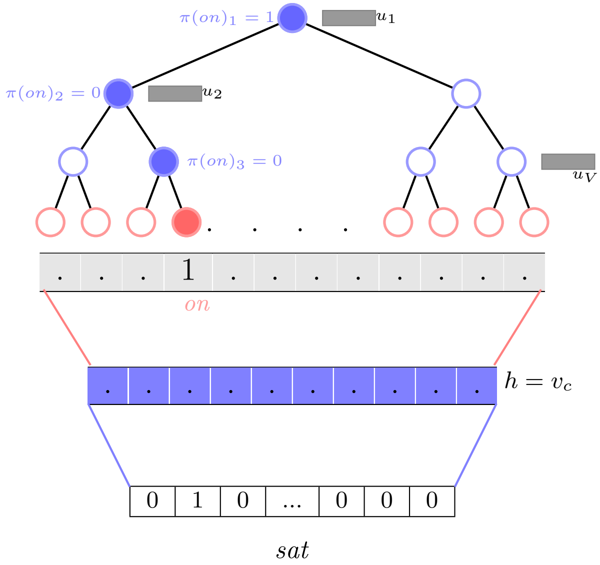


- For a given pair (w, c) we are interested in the probability $p(w|v_c)$
- We model this probability as

$$p(w|v_c) = \prod_k (\pi(w_k)|v_c)$$

- For example

$$\begin{aligned} P(on|v_{sat}) &= P(\pi(on)_1 = 1|v_{sat}) \\ &\quad *P(\pi(on)_2 = 0|v_{sat}) \\ &\quad *P(\pi(on)_3 = 0|v_{sat}) \end{aligned}$$



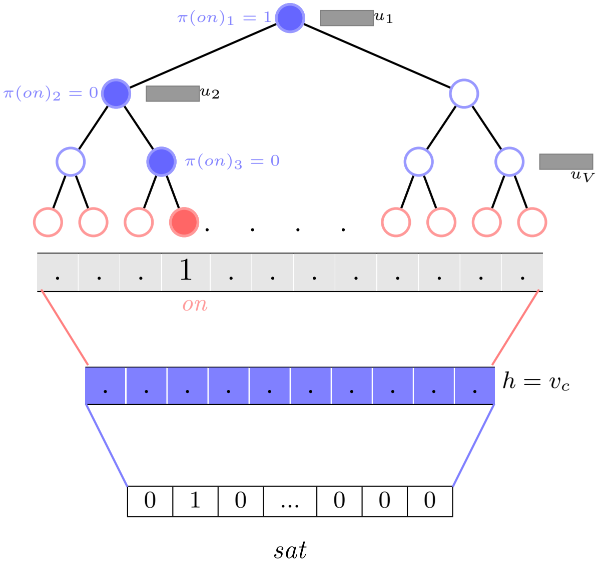
- For a given pair (w, c) we are interested in the probability $p(w|v_c)$
- We model this probability as

$$p(w|v_c) = \prod_k (\pi(w_k)|v_c)$$

- For example

$$\begin{aligned} P(on|v_{sat}) = & P(\pi(on)_1 = 1|v_{sat}) \\ & *P(\pi(on)_2 = 0|v_{sat}) \\ & *P(\pi(on)_3 = 0|v_{sat}) \end{aligned}$$

- In effect, we are saying that the probability of predicting a word is the same as predicting the correct unique path from the root node to that word

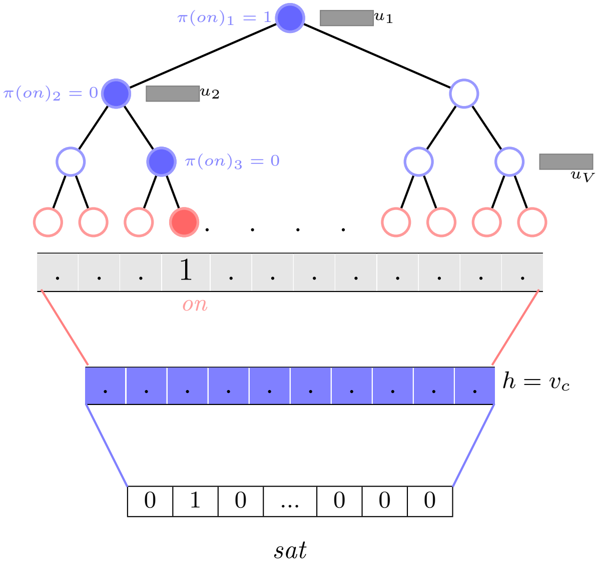


- We model

$$P(\pi(on)_i = 1) = \frac{1}{1 + e^{-v_c^T u_i}}$$

$$P(\pi(on)_i = 0) = 1 - P(\pi(on)_i = 1)$$

$$P(\pi(on)_i = 0) = \frac{1}{1 + e^{v_c^T u_i}}$$



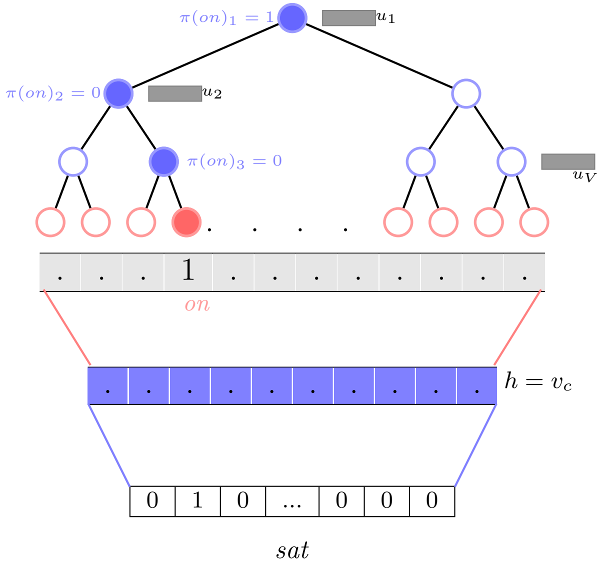
- We model

$$P(\pi(on)_i = 1) = \frac{1}{1 + e^{-v_c^T u_i}}$$

$$P(\pi(on)_i = 0) = 1 - P(\pi(on)_i = 1)$$

$$P(\pi(on)_i = 0) = \frac{1}{1 + e^{v_c^T u_i}}$$

- The above model ensures that the representation of a context word v_c will have a high(low) similarity with the representation of the node u_i if u_i appears and the path branches to the left(right) at u_i



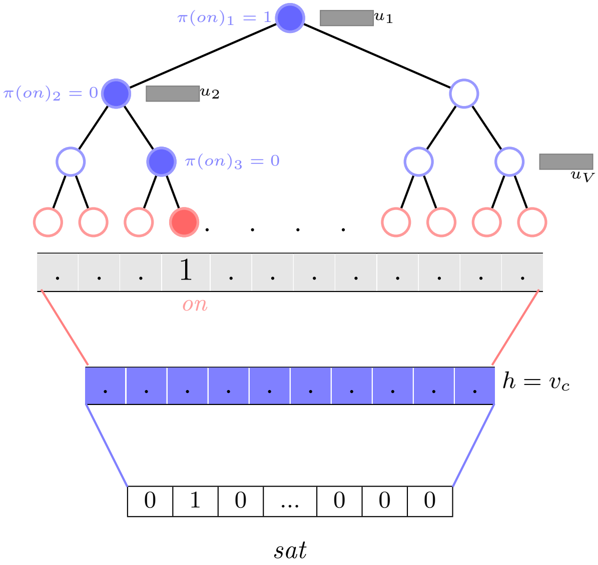
- We model

$$P(\pi(on)_i = 1) = \frac{1}{1 + e^{-v_c^T u_i}}$$

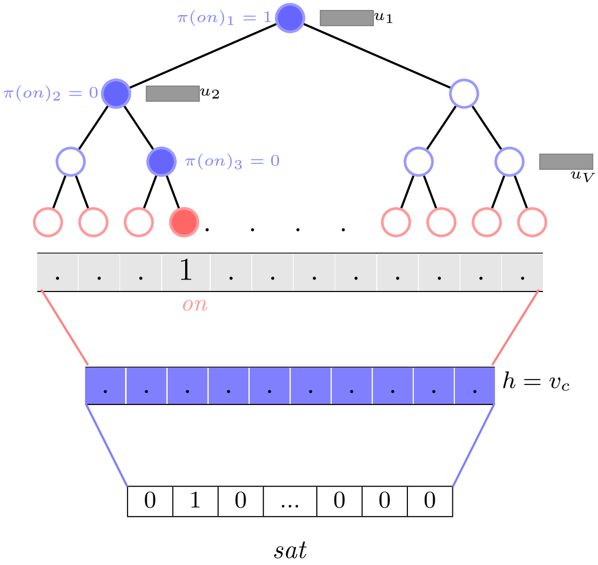
$$P(\pi(on)_i = 0) = 1 - P(\pi(on)_i = 1)$$

$$P(\pi(on)_i = 0) = \frac{1}{1 + e^{v_c^T u_i}}$$

- The above model ensures that the representation of a context word v_c will have a high(low) similarity with the representation of the node u_i if u_i appears and the path branches to the left(right) at u_i
- Again, transitively the representations of contexts which appear with the same words will have high similarity

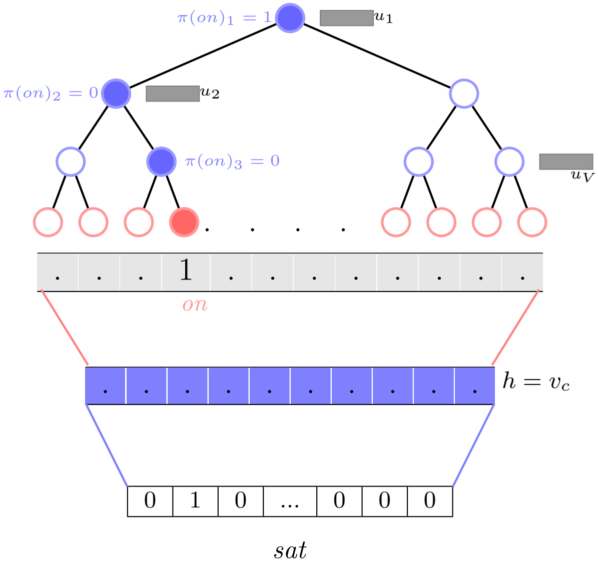


$$P(w|v_c) = \prod_{k=1}^{|\pi(w)|} P(\pi(w_k)|v_c)$$



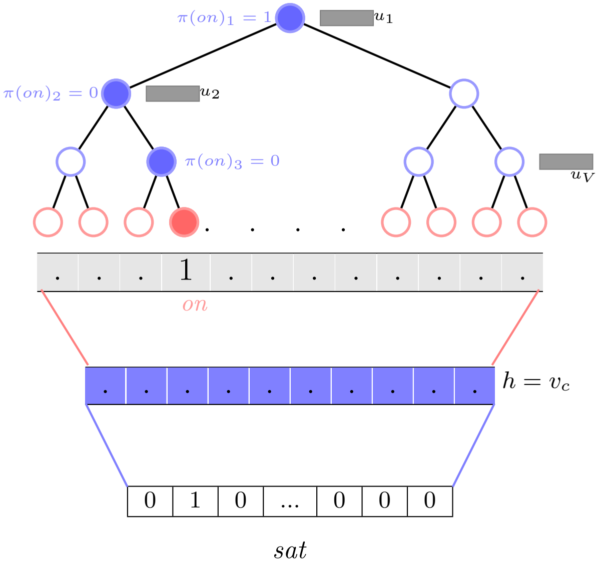
$$P(w|v_c) = \prod_{k=1}^{|\pi(w)|} P(\pi(w_k)|v_c)$$

- Note that $p(w|v_c)$ can now be computed using $|\pi(w)|$ computations instead of $|V|$ required by softmax



$$P(w|v_c) = \prod_{k=1}^{|\pi(w)|} P(\pi(w_k)|v_c)$$

- Note that $p(w|v_c)$ can now be computed using $|\pi(w)|$ computations instead of $|V|$ required by softmax
- How do we construct the binary tree?



$$P(w|v_c) = \prod_{k=1}^{|\pi(w)|} P(\pi(w_k)|v_c)$$

- Note that $p(w|v_c)$ can now be computed using $|\pi(w)|$ computations instead of $|V|$ required by softmax
- How do we construct the binary tree?
- Turns out that even a random arrangement of the words on leaf nodes does well in practice

Module 10.8: GloVe representations

- **Count** based methods (SVD) rely on global co-occurrence counts from the corpus for computing word representations

- **Count** based methods (SVD) rely on global co-occurrence counts from the corpus for computing word representations
- Predict based methods **learn** word representations using co-occurrence information

- **Count** based methods (SVD) rely on global co-occurrence counts from the corpus for computing word representations
- Predict based methods **learn** word representations using co-occurrence information
- Why not combine the two (**count** and **learn**) ?

Corpus:

- Human machine interface for computer applications
- User opinion of computer system response time
- User interface management system
- System engineering for improved response time

$$X =$$

	human	**machine**	**system**	**for**	...	**user**
human	2.01	2.01	0.23	2.14	...	0.43
machine	2.01	2.01	0.23	2.14	...	0.43
system	0.23	0.23	1.17	0.96	...	1.29
for	2.14	2.14	0.96	1.87	...	-0.13
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0.43	0.43	1.29	-0.13	...	1.71

$$P(j|i) = \frac{X_{ij}}{\sum X_{ij}} = \frac{X_{ij}}{X_i}$$

$$X_{ij} = X_{ji}$$

- X_{ij} encodes important global information about the co-occurrence between i and j (global: because it is computed for the entire corpus)

Corpus:

- Human machine interface for computer applications
- User opinion of computer system response time
- User interface management system
- System engineering for improved response time

$$X =$$

	human	machine	system	for	...	user
human	2.01	2.01	0.23	2.14	...	0.43
machine	2.01	2.01	0.23	2.14	...	0.43
system	0.23	0.23	1.17	0.96	...	1.29
for	2.14	2.14	0.96	1.87	...	-0.13
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0.43	0.43	1.29	-0.13	...	1.71

$$P(j|i) = \frac{X_{ij}}{\sum X_{ij}} = \frac{X_{ij}}{X_i}$$

$$X_{ij} = X_{ji}$$

- X_{ij} encodes important global information about the co-occurrence between i and j (global: because it is computed for the entire corpus)
- Why not learn word vectors which are faithful to this information?

Corpus:

- Human machine interface for computer applications
- User opinion of computer system response time
- User interface management system
- System engineering for improved response time

$$X =$$

	human	machine	system	for	...	user
human	2.01	2.01	0.23	2.14	...	0.43
machine	2.01	2.01	0.23	2.14	...	0.43
system	0.23	0.23	1.17	0.96	...	1.29
for	2.14	2.14	0.96	1.87	...	-0.13
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0.43	0.43	1.29	-0.13	...	1.71

$$P(j|i) = \frac{X_{ij}}{\sum X_{ij}} = \frac{X_{ij}}{X_i}$$

$$X_{ij} = X_{ji}$$

- X_{ij} encodes important global information about the co-occurrence between i and j (global: because it is computed for the entire corpus)
- Why not learn word vectors which are faithful to this information?
- For example, enforce

$$\begin{aligned} v_i^T v_j &= \log P(j|i) \\ &= \log X_{ij} - \log(X_i) \end{aligned}$$

Corpus:

- Human machine interface for computer applications
- User opinion of computer system response time
- User interface management system
- System engineering for improved response time

$$X =$$

	human	**machine**	**system**	**for**	...	**user**
human	2.01	2.01	0.23	2.14	...	0.43
machine	2.01	2.01	0.23	2.14	...	0.43
system	0.23	0.23	1.17	0.96	...	1.29
for	2.14	2.14	0.96	1.87	...	-0.13
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0.43	0.43	1.29	-0.13	...	1.71

$$P(j|i) = \frac{X_{ij}}{\sum X_{ij}} = \frac{X_{ij}}{X_i}$$

$$X_{ij} = X_{ji}$$

- X_{ij} encodes important global information about the co-occurrence between i and j (global: because it is computed for the entire corpus)
- Why not learn word vectors which are faithful to this information?
- For example, enforce

$$\begin{aligned} v_i^T v_j &= \log P(j|i) \\ &= \log X_{ij} - \log(X_i) \end{aligned}$$

- Similarly,

$$v_j^T v_i = \log X_{ij} - \log X_j \quad (X_{ij} = X_{ji})$$

Corpus:

- Human machine interface for computer applications
- User opinion of computer system response time
- User interface management system
- System engineering for improved response time

$$X =$$

	human	machine	system	for	...	user
human	2.01	2.01	0.23	2.14	...	0.43
machine	2.01	2.01	0.23	2.14	...	0.43
system	0.23	0.23	1.17	0.96	...	1.29
for	2.14	2.14	0.96	1.87	...	-0.13
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0.43	0.43	1.29	-0.13	...	1.71

$$P(j|i) = \frac{X_{ij}}{\sum X_{ij}} = \frac{X_{ij}}{X_i}$$
$$X_{ij} = X_{ji}$$

- X_{ij} encodes important global information about the co-occurrence between i and j (global: because it is computed for the entire corpus)
- Why not learn word vectors which are faithful to this information?
- For example, enforce

$$v_i^T v_j = \log P(j|i)$$
$$= \log X_{ij} - \log(X_i)$$

- Similarly,

$$v_j^T v_i = \log X_{ij} - \log X_j \quad (X_{ij} = X_{ji})$$

- Essentially we are saying that we want word vectors v_i and v_j such that $v_i^T v_j$ is faithful to the globally computed $P(j|i)$

Corpus:

- Human machine interface for computer applications
- User opinion of computer system response time
- User interface management system
- System engineering for improved response time

$$X =$$

	human	machine	system	for	...	user
human	2.01	2.01	0.23	2.14	...	0.43
machine	2.01	2.01	0.23	2.14	...	0.43
system	0.23	0.23	1.17	0.96	...	1.29
for	2.14	2.14	0.96	1.87	...	-0.13
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0.43	0.43	1.29	-0.13	...	1.71

$$P(j|i) = \frac{X_{ij}}{\sum X_{ij}} = \frac{X_{ij}}{X_i}$$

$$X_{ij} = X_{ji}$$

- Adding the two equations we get

$$2v_i^T v_j = 2\log X_{ij} - \log X_i - \log X_j$$

$$v_i^T v_j = \log X_{ij} - \frac{1}{2}\log X_i - \frac{1}{2}\log X_j$$

Corpus:

- Human machine interface for computer applications
- User opinion of computer system response time
- User interface management system
- System engineering for improved response time

$$X =$$

	human	machine	system	for	...	user
human	2.01	2.01	0.23	2.14	...	0.43
machine	2.01	2.01	0.23	2.14	...	0.43
system	0.23	0.23	1.17	0.96	...	1.29
for	2.14	2.14	0.96	1.87	...	-0.13
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0.43	0.43	1.29	-0.13	...	1.71

$$P(j|i) = \frac{X_{ij}}{\sum X_{ij}} = \frac{X_{ij}}{X_i}$$

$$X_{ij} = X_{ji}$$

- Adding the two equations we get

$$2v_i^T v_j = 2\log X_{ij} - \log X_i - \log X_j$$

$$v_i^T v_j = \log X_{ij} - \frac{1}{2}\log X_i - \frac{1}{2}\log X_j$$

- Note that $\log X_i$ and $\log X_j$ depend only on the words i & j and we can think of them as word specific biases which will be learned

$$v_i^T v_j = \log X_{ij} - b_i - b_j$$

$$v_i^T v_j + b_i + b_j = \log X_{ij}$$

Corpus:

- Human machine interface for computer applications
- User opinion of computer system response time
- User interface management system
- System engineering for improved response time

$$X =$$

	human	machine	system	for	...	user
human	2.01	2.01	0.23	2.14	...	0.43
machine	2.01	2.01	0.23	2.14	...	0.43
system	0.23	0.23	1.17	0.96	...	1.29
for	2.14	2.14	0.96	1.87	...	-0.13
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0.43	0.43	1.29	-0.13	...	1.71

$$P(j|i) = \frac{X_{ij}}{\sum X_{ij}} = \frac{X_{ij}}{X_i}$$

$$X_{ij} = X_{ji}$$

- Adding the two equations we get

$$2v_i^T v_j = 2\log X_{ij} - \log X_i - \log X_j$$

$$v_i^T v_j = \log X_{ij} - \frac{1}{2}\log X_i - \frac{1}{2}\log X_j$$

- Note that $\log X_i$ and $\log X_j$ depend only on the words i & j and we can think of them as word specific biases which will be learned

$$v_i^T v_j = \log X_{ij} - b_i - b_j$$

$$v_i^T v_j + b_i + b_j = \log X_{ij}$$

- We can then formulate this as the following optimization problem

$$\min_{v_i, v_j, b_i, b_j} \sum_{i,j} (\underbrace{v_i^T v_j + b_i + b_j}_{\substack{\text{predicted value} \\ \text{using model} \\ \text{parameters}}} - \underbrace{\log X_{ij}}_{\substack{\text{actual value} \\ \text{computed from} \\ \text{the given corpus}}})^2$$

Corpus:

- Human machine interface for computer applications
- User opinion of computer system response time
- User interface management system
- System engineering for improved response time

$$\min_{v_i, v_j, b_i, b_j} \sum_{i,j} (v_i^T v_j + b_i + b_j - \log X_{ij})^2$$

$$X =$$

	human	machine	system	for	...	user
human	2.01	2.01	0.23	2.14	...	0.43
machine	2.01	2.01	0.23	2.14	...	0.43
system	0.23	0.23	1.17	0.96	...	1.29
for	2.14	2.14	0.96	1.87	...	-0.13
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0.43	0.43	1.29	-0.13	...	1.71

$$P(j|i) = \frac{X_{ij}}{\sum X_{ij}} = \frac{X_{ij}}{\sum X_i}$$

$$X_{ij} = X_{ji}$$

Corpus:

- Human machine interface for computer applications
- User opinion of computer system response time
- User interface management system
- System engineering for improved response time

$$X =$$

	human	machine	system	for	...	user
human	2.01	2.01	0.23	2.14	...	0.43
machine	2.01	2.01	0.23	2.14	...	0.43
system	0.23	0.23	1.17	0.96	...	1.29
for	2.14	2.14	0.96	1.87	...	-0.13
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0.43	0.43	1.29	-0.13	...	1.71

$$P(j|i) = \frac{X_{ij}}{\sum X_{ij}} = \frac{X_{ij}}{\sum X_i}$$

$$X_{ij} = X_{ji}$$

$$\min_{v_i, v_j, b_i, b_j} \sum_{i,j} (v_i^T v_j + b_i + b_j - \log X_{ij})^2$$

- **Drawback:** weighs all co-occurrences equally

Corpus:

- Human machine interface for computer applications
- User opinion of computer system response time
- User interface management system
- System engineering for improved response time

$$X =$$

	human	machine	system	for	...	user
human	2.01	2.01	0.23	2.14	...	0.43
machine	2.01	2.01	0.23	2.14	...	0.43
system	0.23	0.23	1.17	0.96	...	1.29
for	2.14	2.14	0.96	1.87	...	-0.13
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0.43	0.43	1.29	-0.13	...	1.71

$$P(j|i) = \frac{X_{ij}}{\sum X_{ij}} = \frac{X_{ij}}{\sum X_i}$$

$$X_{ij} = X_{ji}$$

$$\min_{v_i, v_j, b_i, b_j} \sum_{i,j} (v_i^T v_j + b_i + b_j - \log X_{ij})^2$$

- **Drawback:** weighs all co-occurrences equally
- **Solution:** add a weighting function

$$\min_{v_i, v_j, b_i, b_j} \sum_{i,j} f(X_{ij})(v_i^T v_j + b_i + b_j - \log X_{ij})^2$$

Corpus:

- Human machine interface for computer applications
- User opinion of computer system response time
- User interface management system
- System engineering for improved response time

$$X =$$

	human	machine	system	for	...	user
human	2.01	2.01	0.23	2.14	...	0.43
machine	2.01	2.01	0.23	2.14	...	0.43
system	0.23	0.23	1.17	0.96	...	1.29
for	2.14	2.14	0.96	1.87	...	-0.13
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
user	0.43	0.43	1.29	-0.13	...	1.71

$$P(j|i) = \frac{X_{ij}}{\sum X_{ij}} = \frac{X_{ij}}{\sum X_i}$$

$$X_{ij} = X_{ji}$$

$$\min_{v_i, v_j, b_i, b_j} \sum_{i,j} (v_i^T v_j + b_i + b_j - \log X_{ij})^2$$

- **Drawback:** weighs all co-occurrences equally
- **Solution:** add a weighting function

$$\min_{v_i, v_j, b_i, b_j} \sum_{i,j} f(X_{ij})(v_i^T v_j + b_i + b_j - \log X_{ij})^2$$

- **Wishlist:** $f(X_{ij})$ should be such that neither rare nor frequent words are over-weighted.

$$f(x) = \left\{ \begin{array}{ll} (\frac{x}{x_{max}})^{\alpha}, & \text{if } x < x_{max} \\ 1, & \text{otherwise} \end{array} \right\}$$

where α can be tuned for a given dataset

Module 10.9: Evaluating word representations

How do we evaluate the learned word representations ?

Semantic Relatedness

Semantic Relatedness

- Ask humans to judge the relatedness between a pair of words

$$S_{human}(cat, dog) = 0.8$$

Semantic Relatedness

- Ask humans to judge the relatedness between a pair of words
- Compute the cosine similarity between the corresponding word vectors learned by the model

$$S_{human}(cat, dog) = 0.8$$

$$S_{model}(cat, dog) = \frac{v_{cat}^T v_{dog}}{\| v_{cat} \| \| v_{dog} \|} = 0.7$$

Semantic Relatedness

$$S_{human}(cat, dog) = 0.8$$

$$S_{model}(cat, dog) = \frac{v_{cat}^T v_{dog}}{\| v_{cat} \| \| v_{dog} \|} = 0.7$$

- Ask humans to judge the relatedness between a pair of words
- Compute the cosine similarity between the corresponding word vectors learned by the model
- Given a large number of such word pairs, compute the correlation between S_{model} & S_{human}, and compare different models

Semantic Relatedness

$$S_{human}(cat, dog) = 0.8$$

$$S_{model}(cat, dog) = \frac{v_{cat}^T v_{dog}}{\| v_{cat} \| \| v_{dog} \|} = 0.7$$

- Ask humans to judge the relatedness between a pair of words
- Compute the cosine similarity between the corresponding word vectors learned by the model
- Given a large number of such word pairs, compute the correlation between S_{model} & S_{human}, and compare different models
- Model 1 is better than Model 2 if

$$correlation(S_{model1}, S_{human})$$
$$> correlation(S_{model2}, S_{human})$$

Synonym Detection

Synonym Detection

- Given: a term and four candidate synonyms

Term : levied

Candidates : {unposed, believed, requested, correlated}

Synonym Detection

Term : levied

Candidates : {unposed, believed, requested, correlated}

$$\textbf{Synonym} := \underset{c \in C}{argmax}\ cosine(v_{term}, v_c)$$

- Given: a term and four candidate synonyms
- Pick the candidate which has the largest cosine similarity with the term

Synonym Detection

- Given: a term and four candidate synonyms
- Pick the candidate which has the largest cosine similarity with the term
- Compute the accuracy of different models and compare

Term : levied

Candidates : {unposed, believed, requested, correlated}

$$\textbf{Synonym} := \underset{c \in C}{argmax}\ cosine(v_{term}, v_c)$$

Analogy

Analogy

- Semantic Analogy: Find nearest neighbour of $v_{sister} - v_{brother} + v_{grandson}$

brother : sister :: grandson : ?

Analogy

- Semantic Analogy: Find nearest neighbour of $v_{sister} - v_{brother} + v_{grandson}$
- Syntactic Analogy: Find nearest neighbour of $V_{works} - v_{work} + v_{speak}$

brother : sister :: grandson : ?
work : works :: speak : ?

- So which algorithm gives the best result ?

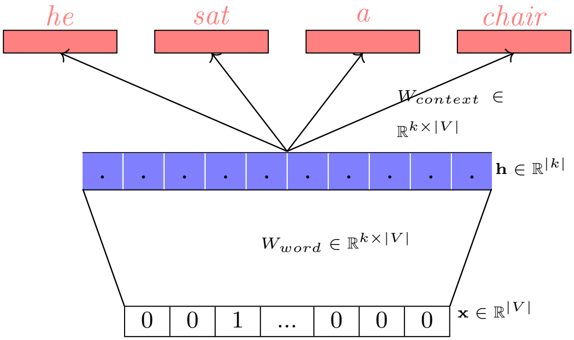
- So which algorithm gives the best result ?
- Boroni et.al [2014] showed that predict models consistently outperform count models in all tasks.

- So which algorithm gives the best result ?
- Boroni et.al [2014] showed that predict models consistently outperform count models in all tasks.
- Levy et.al [2015] do a much more through analysis (IMO) and show that good old SVD does better than prediction based models on similarity tasks but not on analogy tasks.

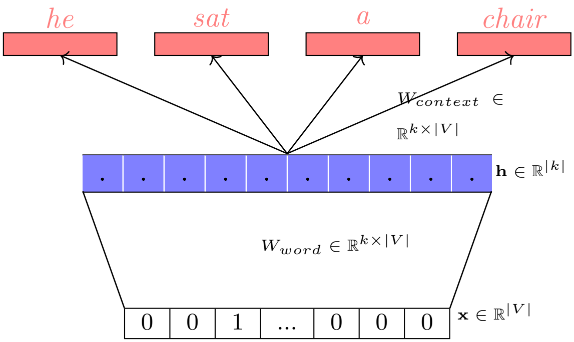
Module 10.10: Relation between SVD & word2Vec

The story ahead ...

- Continuous bag of words model
- Skip gram model with negative sampling (the famous word2vec)
- GloVe word embeddings
- Evaluating word embeddings
- Good old SVD does just fine!!

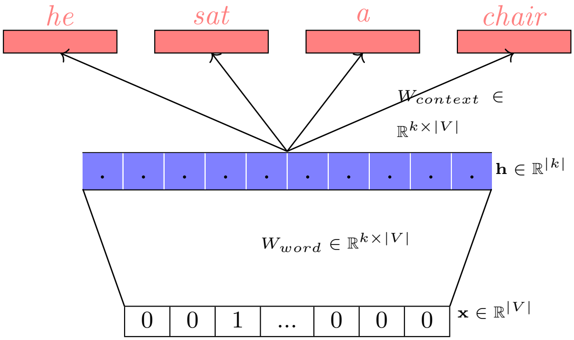


- Recall that SVD does a matrix factorization of the co-occurrence matrix



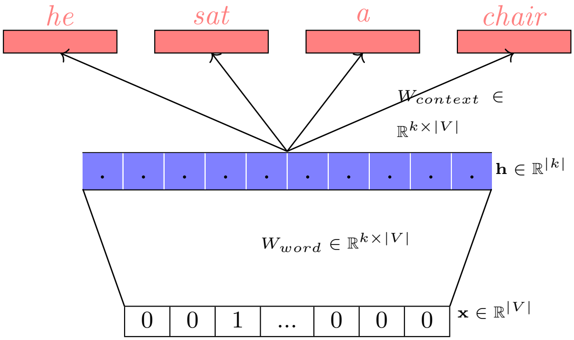
- Recall that SVD does a matrix factorization of the co-occurrence matrix
- Levy et.al [2015] show that word2vec also implicitly does a matrix factorization

.

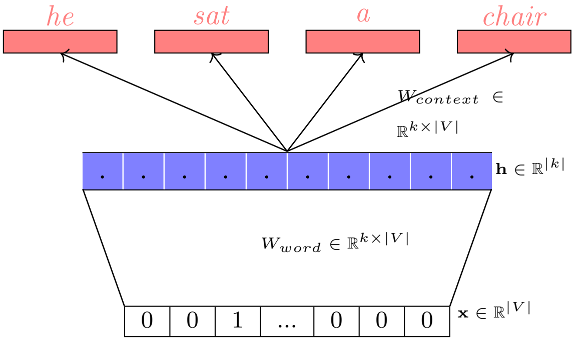


- Recall that SVD does a matrix factorization of the co-occurrence matrix
- Levy et.al [2015] show that word2vec also implicitly does a matrix factorization
- What does this mean ?

.



- Recall that SVD does a matrix factorization of the co-occurrence matrix
- Levy et.al [2015] show that word2vec also implicitly does a matrix factorization
- What does this mean ?
- Recall that word2vec gives us $W_{context}$ & W_{word} .



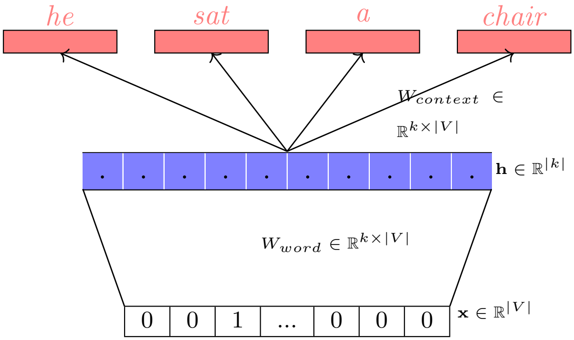
- Recall that SVD does a matrix factorization of the co-occurrence matrix
- Levy et.al [2015] show that word2vec also implicitly does a matrix factorization
- What does this mean ?
- Recall that word2vec gives us $W_{context}$ & W_{word} .
- Turns out that we can also show that

$$M = W_{context} * W_{word}$$

where

$$M_{ij} = PMI(w_i, c_i) - log(k)$$

$$k = \text{number of negative samples}$$



- Recall that SVD does a matrix factorization of the co-occurrence matrix
- Levy et.al [2015] show that word2vec also implicitly does a matrix factorization
- What does this mean ?
- Recall that word2vec gives us $W_{context}$ & W_{word} .
- Turns out that we can also show that

$$M = W_{context} * W_{word}$$

where

$$M_{ij} = PMI(w_i, c_i) - log(k)$$
$$k = \text{number of negative samples}$$

- So essentially, word2vec factorizes a matrix M which is related to the PMI based co-occurrence matrix (very similar to what SVD does)