This is AI4001

GCR : t37g47w

Problem With Text

A problem with modeling text is that it is messy, and techniques like machine learning algorithms prefer well defined fixed-length inputs and outputs.

Machine learning algorithms cannot work with raw text directly; the text must be converted into numbers. Specifically, vectors of numbers.

This is called feature extraction or feature encoding.

A popular and simple method of feature extraction with text data is called the bag-of-words model of text.

Bag Of Words

A bag-of-words is a representation of text that describes the occurrence of words within a document. It involves two things:

- A vocabulary of known words.
- A measure of the presence of known words.

It is called a “bag” of words, because any information about the order or structure of words in the document is discarded. The model is only concerned with whether known words occur in the document, not where in the document.

We look at the histogram of the words within the text, i.e. considering each word count as a feature.

Bag Of Words

The intuition is that documents are similar if they have similar content. Further, that from the content alone we can learn something about the meaning of the document.

The bag-of-words can be as simple or complex as you like. The complexity comes both in deciding how to design the vocabulary of known words (or tokens) and how to score the presence of known words.

Bag Of Words

```
It was the best of times,
it was the worst of times,
it was the age of wisdom,
it was the age of foolishness,
```

Bag Of Words

```
It was the best of times,
it was the worst of times,
it was the age of wisdom,
it was the age of foolishness,
```

Design the Vocabulary

Bag Of Words

It was the best of times,

it was the worst of times,

it was the age of wisdom,

it was the age of foolishness,

- "it"
- "was"
- "the"
- "best"
- "of"
- "times"
- "worst"
- "age"
- "wisdom"
- "foolishness"

Design the Vocabulary

Bag Of Words

```
"it was the worst of times" = [1, 1, 1, 0, 1, 1, 1, 0, 0, 0]
"it was the age of wisdom" = [1, 1, 1, 0, 1, 0, 0, 1, 1, 0]
"it was the age of foolishness" = [1, 1, 1, 0, 1, 0, 0, 1, 0, 1]
```

It was the best of times,

it was the worst of times,

it was the age of wisdom,

it was the age of foolishness,

- "it"
- "was"
- "the"
- "best"
- "of"
- "times"
- "worst"
- "age"
- "wisdom"
- "foolishness"

Create Document Vectors

Bag Of Words

All ordering of the words is nominally discarded

New documents that overlap with the vocabulary of known words, but may contain words outside of the vocabulary, can still be encoded, where only the occurrence of known words are scored and unknown words are ignored.

Bag Of Words

Review 1: This movie is very scary and long
Review 2: This movie is not scary and is slow
Review 3: This movie is spooky and good

Vector of Review 1: [1 1 1 1 1 1 1 0 0 0 0]
Vector of Review 2: [1 1 2 0 0 1 1 0 1 0 0]
Vector of Review 3: [1 1 1 0 0 0 1 0 0 1 1]

	1 This	2 movie	3 is	4 very	5 scary	6 and	7 long	8 not	9 slow	10 spooky	11 good	Length of the review(in words)
Review 1	1	1	1	1	1	1	1	0	0	0	0	7
Review 2	1	1	2	0	0	1	1	0	1	0	0	8
Review 3	1	1	1	0	0	0	1	0	0	1	1	6

Managing Vocabulary

As the vocabulary size increases, so does the vector representation of documents.

This results in a vector with lots of zero scores, called a sparse vector or sparse representation.

As such, there is pressure to decrease the size of the vocabulary when using a bag-of-words model.

Managing Vocabulary

1. Text Cleaning
   - Ignoring case
   - Ignoring punctuation
   - Ignoring stop words, like “a,” “of,” etc.
   - Fixing misspelled words.
   - Reducing words to their stem

2. Create a vocabulary of grouped words.

BOW

A bag-of-bigrams representation is much more powerful than bag-of-words, and in many cases proves very hard to beat.

— Page 75, Neural Network Methods in Natural Language Processing, 2017.

TF IDF

A problem with scoring word frequency is that highly frequent words start to dominate in the document (e.g. larger score), but may not contain as much “informational content” to the model as rarer but perhaps domain specific words.

One approach is to rescale the frequency of words by how often they appear in all documents, so that the scores for frequent words like “the” that are also frequent across all documents are penalized.

This approach to scoring is called Term Frequency – Inverse Document Frequency, or TF-IDF for short.

Tokenization

Term Frequency: is a scoring of the frequency of the word in the current document.

Inverse Document Frequency: is a scoring of how rare the word is across documents.

Thus the idf of a rare term is high, whereas the idf of a frequent term is likely to be low.

TF IDF

$$TF(w, d) = \frac{occurences\ of\ w\ in\ document\ d}{total\ number\ of\ words\ in\ document\ d}$$

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

TF IDF

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

$$TF(w,d) = \frac{occurences\ of\ w\ in\ document\ d}{total\ number\ of\ words\ in\ document\ d}$$

Words	TF (for A)	TF (for B)
Jupiter	1/5	0
Is	1/5	1/8
The	1/5	2/8
largest	1/5	0
Planet	1/5	1/8
Mars	0	1/8
Fourth	0	1/8
From	0	1/8
Sun	0	1/8

TF IDF

$$IDF(w, D) = \ln\left(\frac{Total\ number\ of\ documents\ (N)\ in\ corpus\ D}{number\ of\ documents\ containing\ w}\right)$$

Words	TF (for A)	TF (for B)	IDF
Jupiter	1/5	0	ln(2/1) = 0.69
Is	1/5	1/8	ln(2/2) = 0
The	1/5	2/8	ln(2/2) = 0
largest	1/5	0	ln(2/1) = 0.69
Planet	1/5	1/8	ln(2/2) = 0
Mars	0	1/8	ln(2/1) = 0.69
Fourth	0	1/8	ln(2/1) = 0.69
From	0	1/8	ln(2/1) = 0.69
Sun	0	1/8	ln(2/1) = 0.69

TF IDF

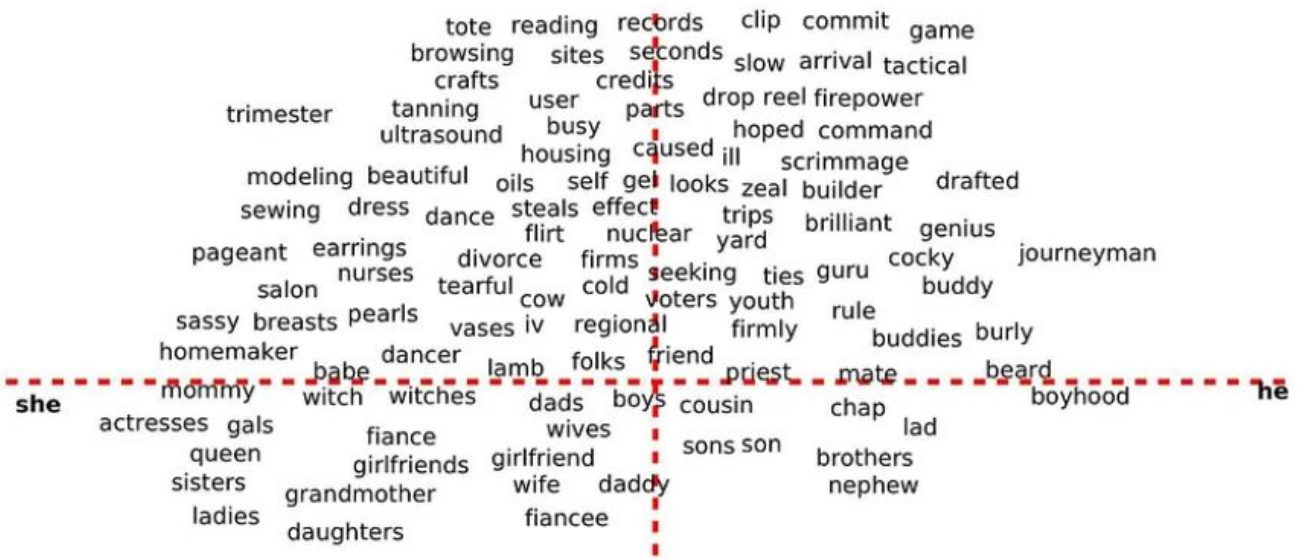
Words	TF (for A)	TF (for B)	IDF	TFIDF (A)	TFIDF (B)
Jupiter	1/5	0	ln(2/1) = 0.69	0.138	0
Is	1/5	1/8	ln(2/2) = 0	0	0
The	1/5	2/8	ln(2/2) = 0	0	0
largest	1/5	0	ln(2/1) = 0.69	0.138	0
Planet	1/5	1/8	ln(2/2) = 0	0.138	0
Mars	0	1/8	ln(2/1) = 0.69	0	0.086
Fourth	0	1/8	ln(2/1) = 0.69	0	0.086
From	0	1/8	ln(2/1) = 0.69	0	0.086
Sun	0	1/8	ln(2/1) = 0.69	0	0.086

DisAdvantage

Not capturing Semantics

Word Representation

To make a machine learn from the raw text we need to transform data into a vector format. This transformation of raw text into a vector format is known as word representation.



Representing Words By Their context

Distributional semantics: A word’s meaning is given by the words that frequently appear close-by.

“You shall know a word by the company it keeps”

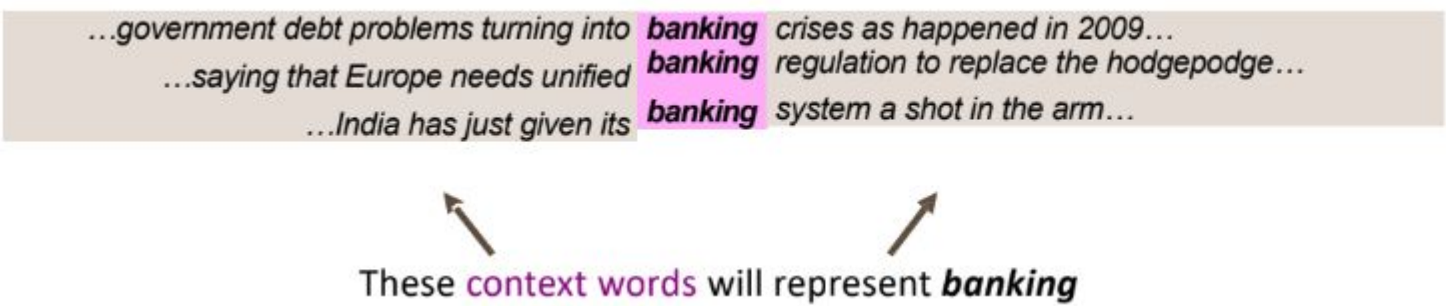
(J. R. Firth 1957: 11)

Representing Words By Their context

One of the most successful ideas of modern statistical NLP!

When a word w appears in a text, its context is the set of words that appear nearby (within a fixed-size window).

Use the many contexts of w to build up a representation of w



Representing Words By Their context

We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts

Word vectors are also called word embeddings or (neural) word representations

They are a distributed representation

$$\textit{banking} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

Word2Vec

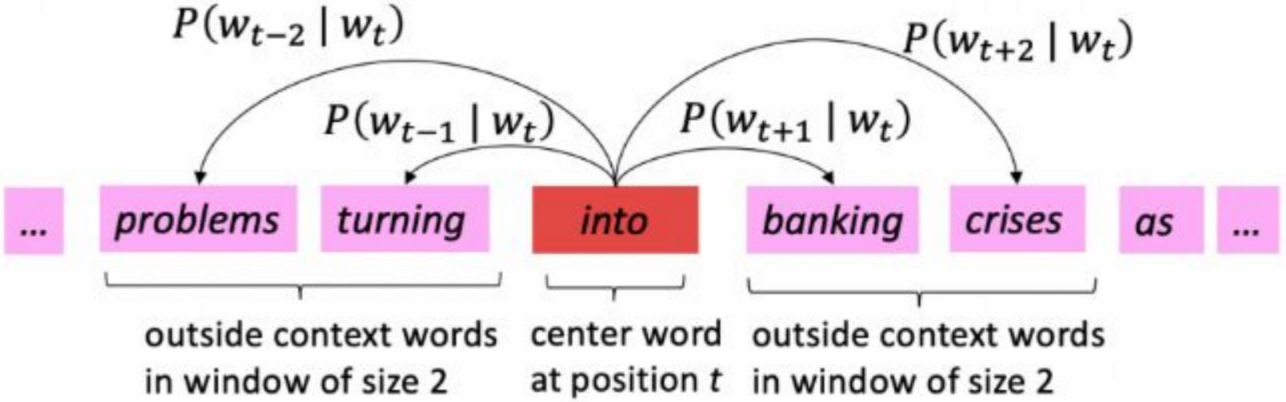
Word2vec (Mikolov et al. 2013) is a framework for learning word vectors

Idea:

- We have a large corpus ("body") of text
- Every word in a fixed vocabulary is represented by a vector
- Go through each position *t* in the text, which has a center word *c* and context ("outside") words *o*
- Use the similarity of the word vectors for *c* and *o* to calculate the probability of *o* given *c* (or vice versa)
- Keep adjusting the word vectors to maximize this probability

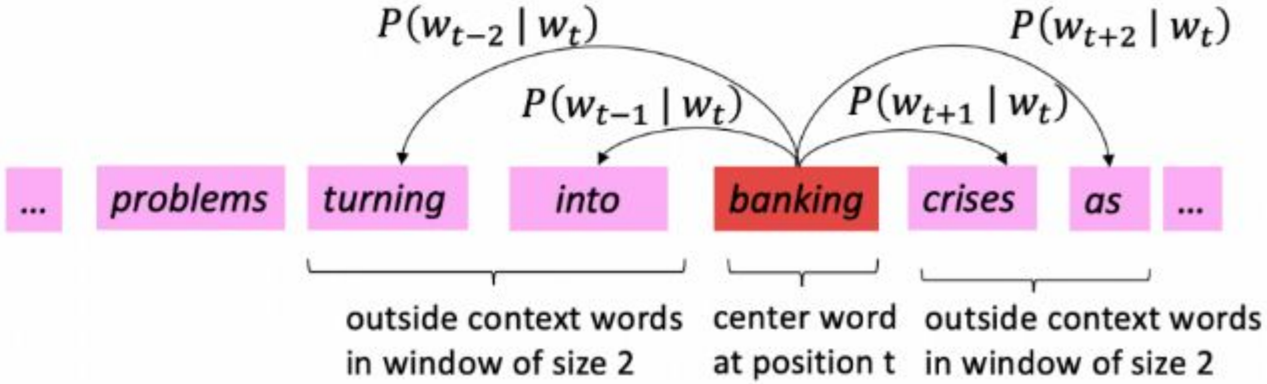
Word2Vec

Example windows and process for computing $P\left(w_{t+j} \mid w_t\right)$

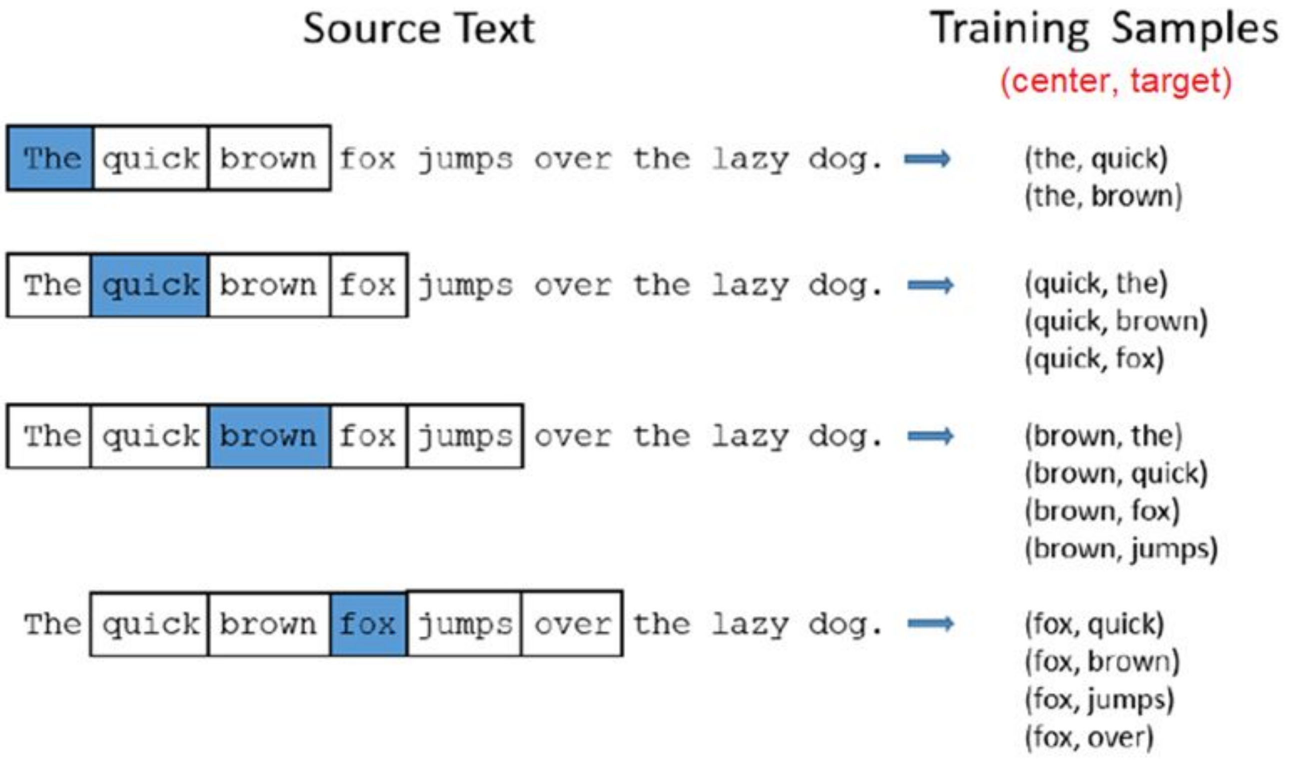


Word2Vec

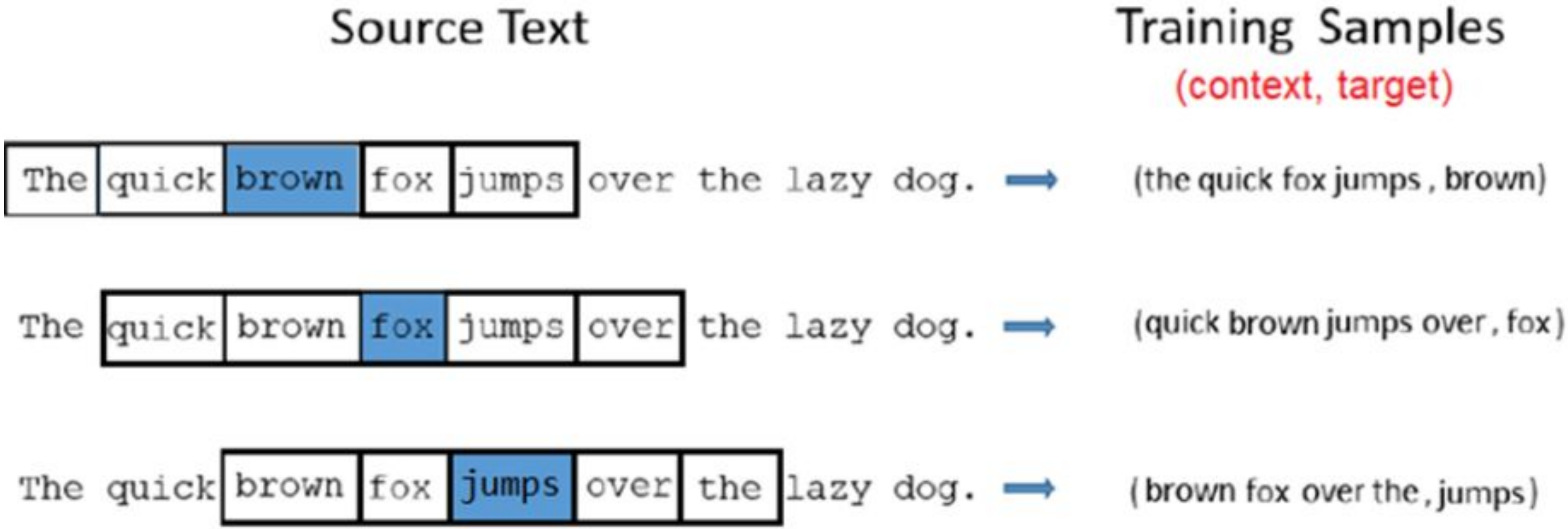
Example windows and process for computing $P(w_{t+j} \mid w_t)$



Skip Gram


Source Text
Training Samples
(center, target)
The quick brown fox jumps over the lazy dog.
(the, quick)
(the, brown)
The quick brown fox jumps over the lazy dog.
(quick, the)
(quick, brown)
(quick, fox)
The quick brown fox jumps over the lazy dog.
(brown, the)
(brown, quick)
(brown, fox)
(brown, jumps)
The quick brown fox jumps over the lazy dog.
(fox, quick)
(fox, brown)
(fox, jumps)
(fox, over)

CBOW



Source Text
Training Samples
(context, target)
The quick brown fox jumps over the lazy dog.
(the quick fox jumps , brown)
The quick brown fox jumps over the lazy dog.
(quick brown jumps over , fox)
The quick brown fox jumps over the lazy dog.
(brown fox over the , jumps)

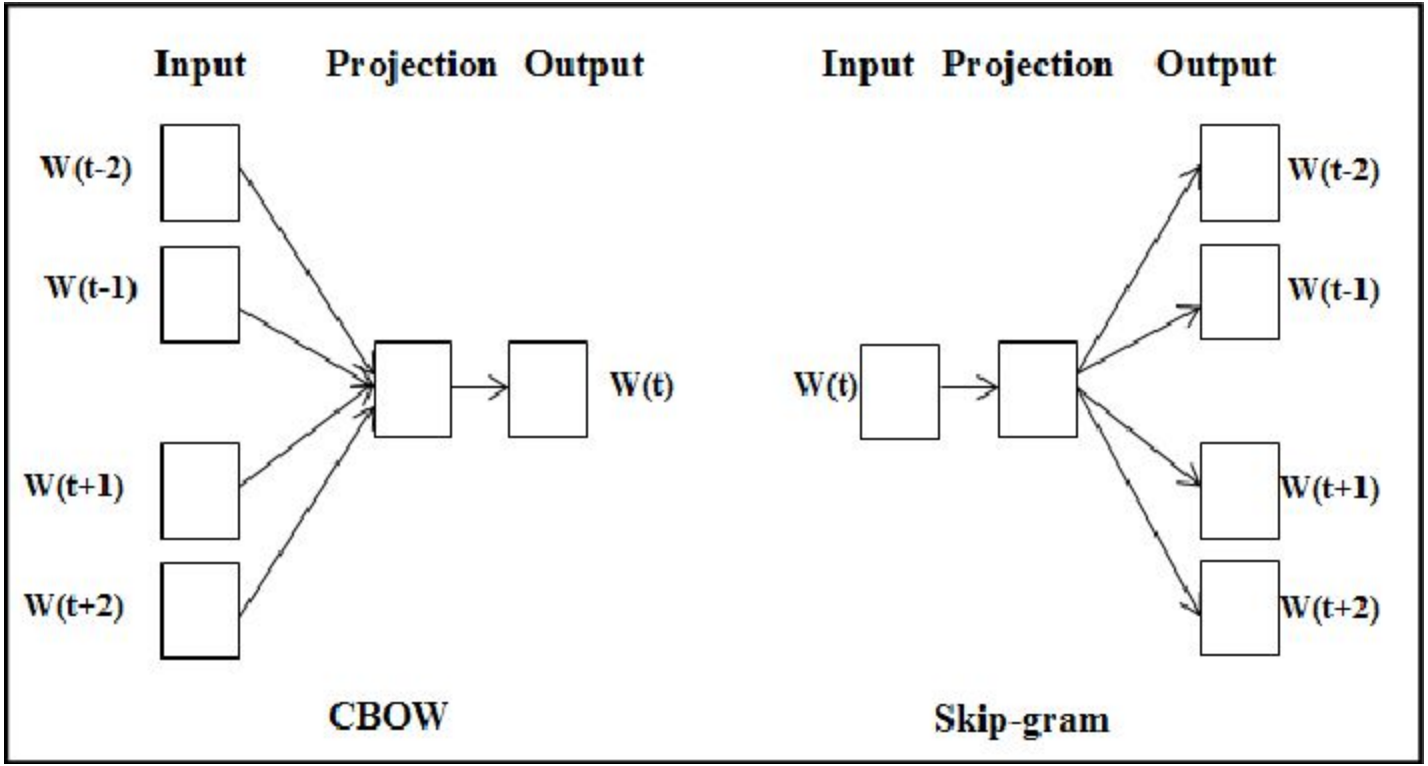
SkipGram

Unsupervised learning techniques or Semi Supervised

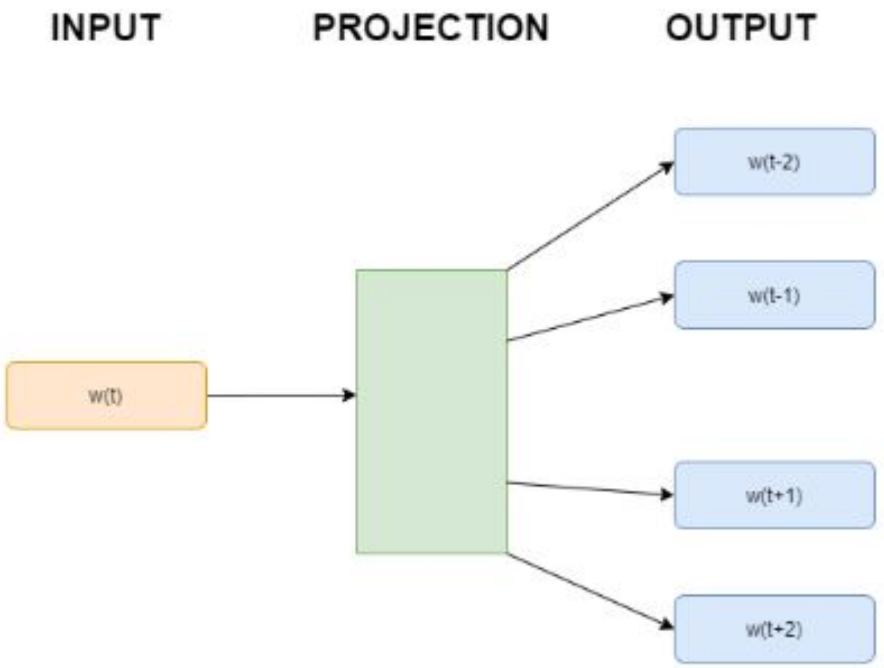
Target word is input while context words are output.

As there is more than one context word to be predicted which makes this problem difficult.

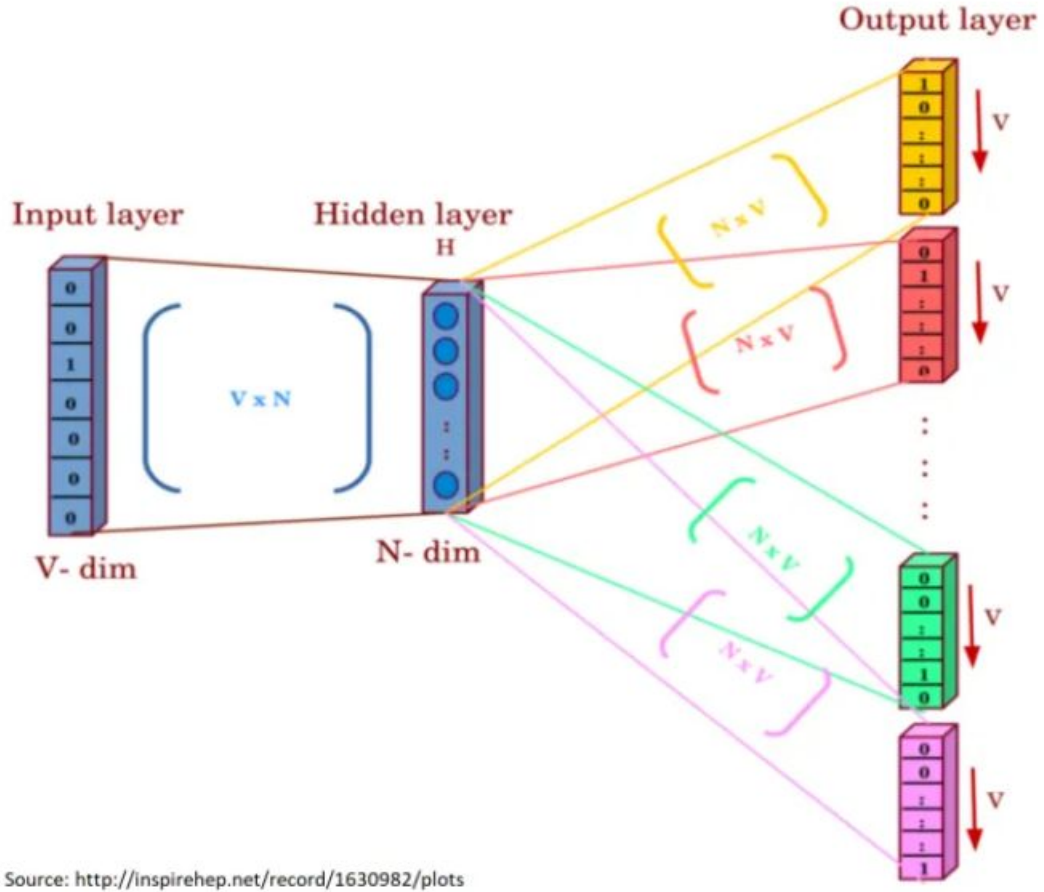
Skip Gram VS CBOW



Skip Gram



SkipGram



Source: http://inspirehep.net/record/1630982/plots

N = context window

SkipGram

> *The man who passes the sentence should swing the sword.*
>
> **- Ned Stark**

We will use `window=1`, and assume that *'passes'* is the current center word, making *'who'* and *'the'* context words. `window` is a hyper-parameter that can be empirically tuned. It typically has a range of $[1, 10]$.

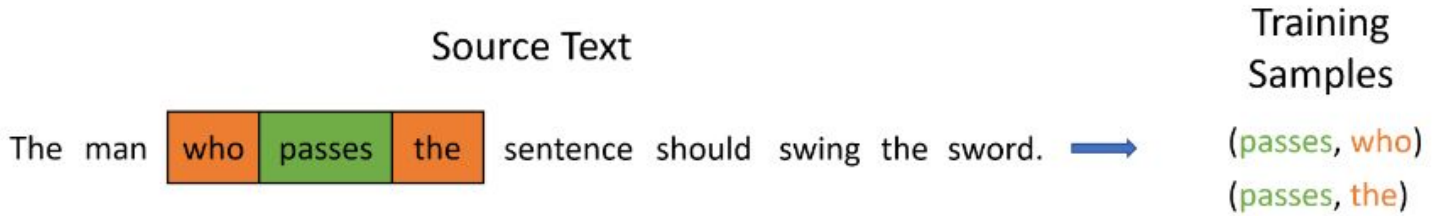


Figure 4: Training Window

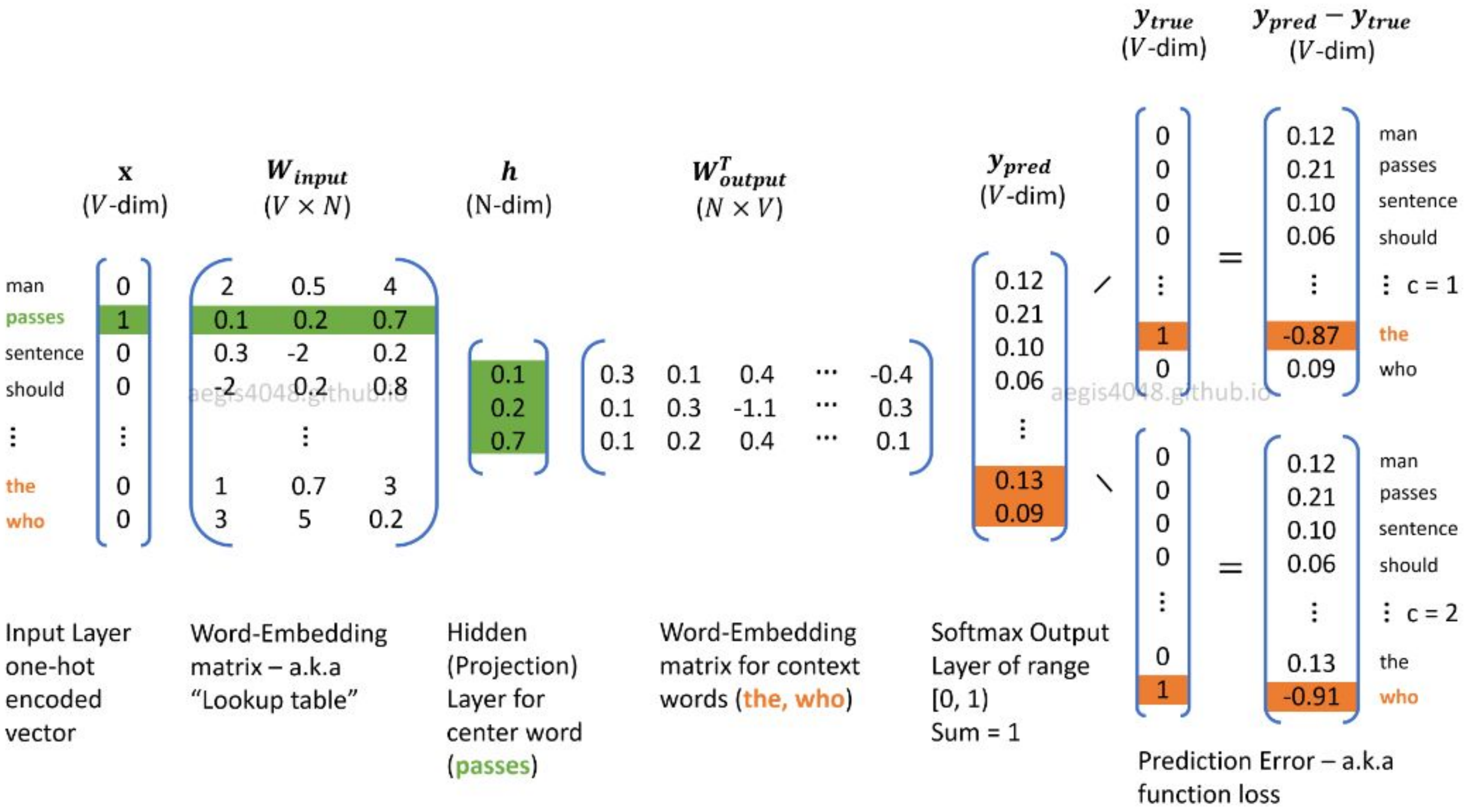


Figure 5: Skip-Gram model structure. Current center word is "passes"

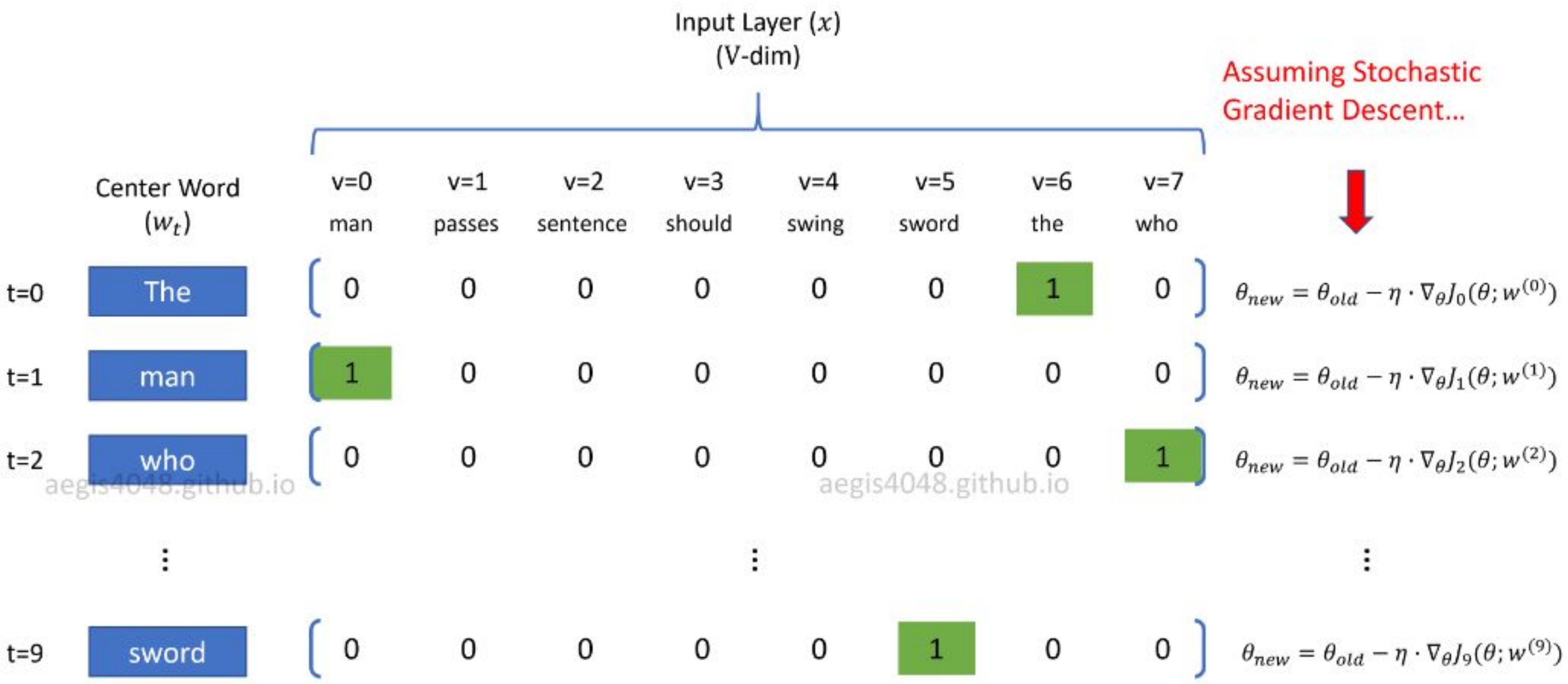


Figure 6: One-hot encoded input vector and parameter update

SkipGram

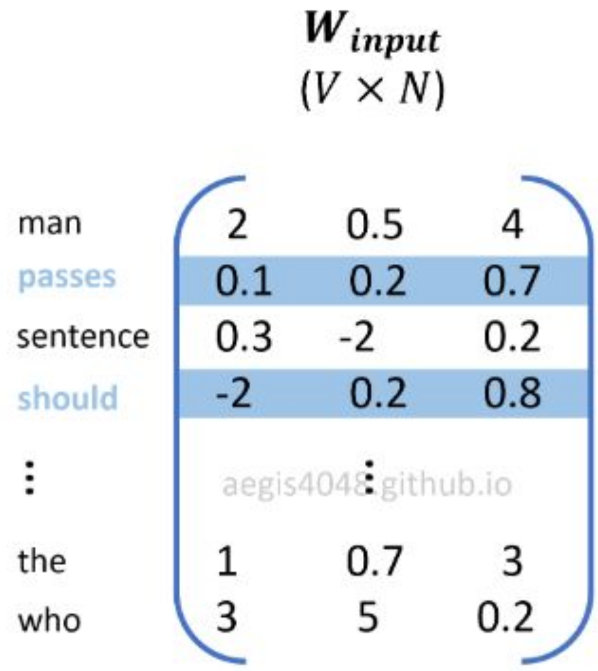


Figure 7: Word-embedding matrix, W_{input}

Notes: θ in cost function

There are two weight matrices that need to be optimized in Skip-Gram model: W_{input} and W_{output}. Often times in neural net, the weights are expressed as θ. In Skip-Gram, θ is a concatenation of input and output weight matrices — $[W_{input} \quad W_{output}]$.

$$\theta = [W_{input} \quad W_{output}] = \begin{bmatrix} u_{the} \\ u_{passes} \\ \vdots \\ u_{who} \\ v_{the} \\ v_{passes} \\ \vdots \\ v_{who} \end{bmatrix} \in \mathbb{R}^{2NV}$$

θ has a size of $2V \times N$, where V is the number of unique vocab in a corpus, and N is the dimension of word vectors in the embedding matrices. 2 is multipled to V because there are two weight matrices, W_{input} and W_{output}. u is a word vector from W_{input} and v is a word vector from W_{output}. Each word vectors are N-dim row vectors from input and output embedding matrices.

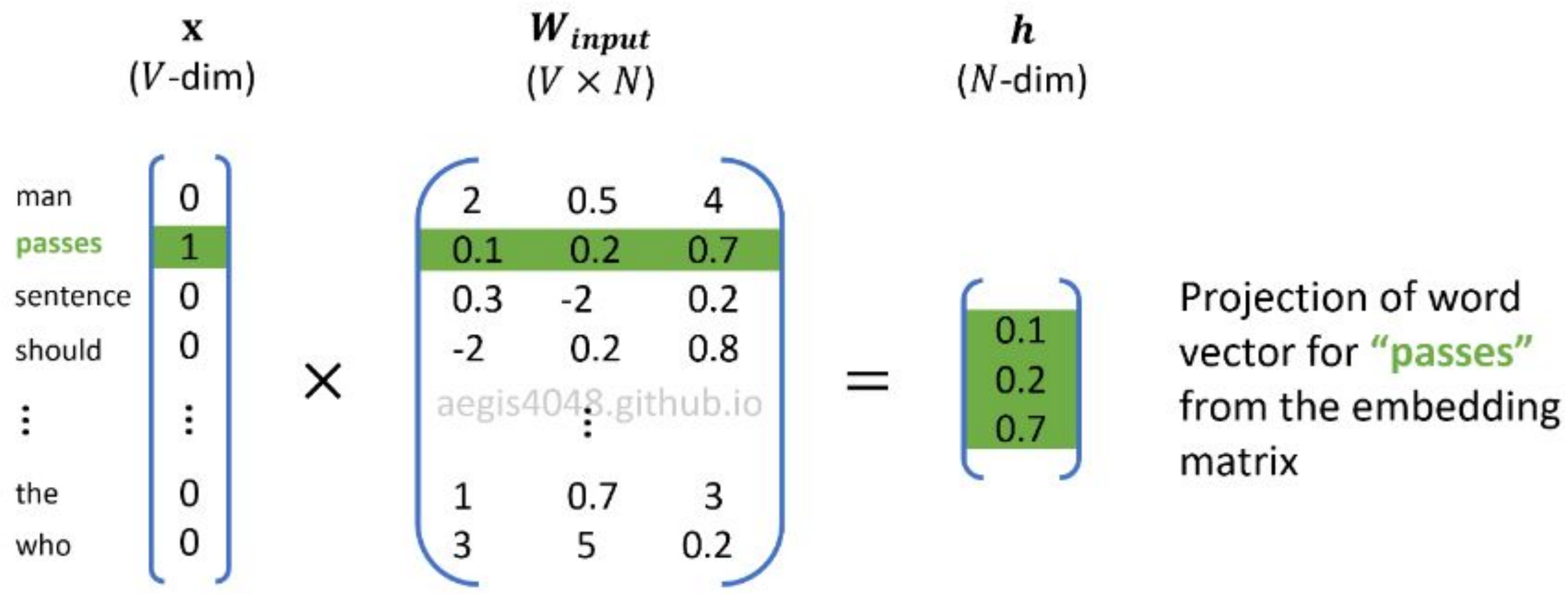


Figure 9: Computing projection layer

h is obtained by multiplying the input word embedding matrix with the V-dim input vector.

$$h = W_{input}^{T} \cdot x \in \mathbb{R}^N$$

SkipGram

$$p(w_{context}|w_{center}) = \frac{exp(W_{output_{(context)}} \cdot h)}{\sum_{i=1}^{V} exp(W_{output_{(i)}} \cdot h)} \in \mathbb{R}^1$$

$$\begin{bmatrix} p(w_1|w_{center}) \\ p(w_2|w_{center}) \\ p(w_3|w_{center}) \\ \vdots \\ p(w_V|w_{center}) \end{bmatrix} = \frac{exp(W_{output} \cdot h)}{\sum_{i=1}^{V} exp(W_{output_{(i)}} \cdot h)} \in \mathbb{R}^V$$

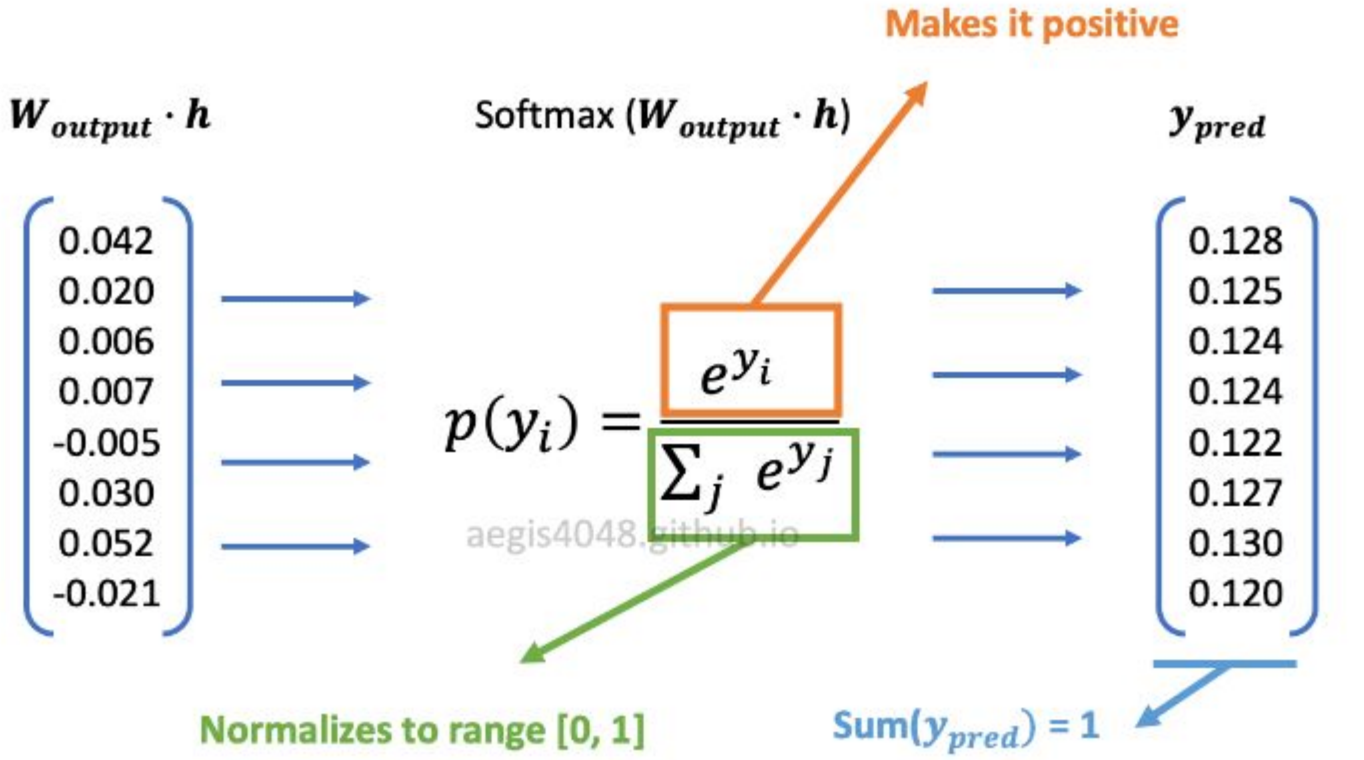


Figure 10: softmax function transformation

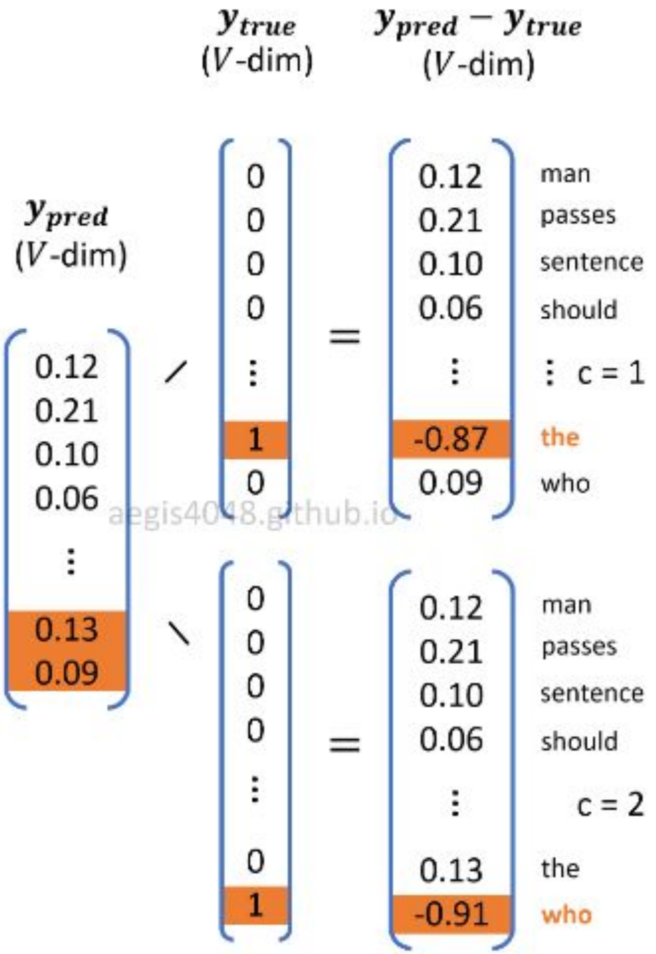


Figure 11: Prediction error window

$$\sum_{c=1}^{C} e_c = \begin{array}{c} c=1 \\ \begin{pmatrix} 0.12 \\ 0.21 \\ 0.10 \\ 0.06 \\ \vdots \\ -0.87 \\ 0.09 \end{pmatrix} \end{array} + \begin{array}{c} c=2 \\ \begin{pmatrix} 0.12 \\ 0.21 \\ 0.10 \\ 0.06 \\ \vdots \\ 0.13 \\ -0.91 \end{pmatrix} \end{array} = \begin{pmatrix} 0.24 \\ 0.42 \\ 0.20 \\ 0.12 \\ \vdots \\ -0.74 \\ -0.82 \end{pmatrix} \begin{array}{l} \text{man} \\ \text{passes} \\ \text{sentence} \\ \text{should} \\ \vdots \\ \text{the} \\ \text{who} \end{array}$$

Figure 12: Sum of prediction errors

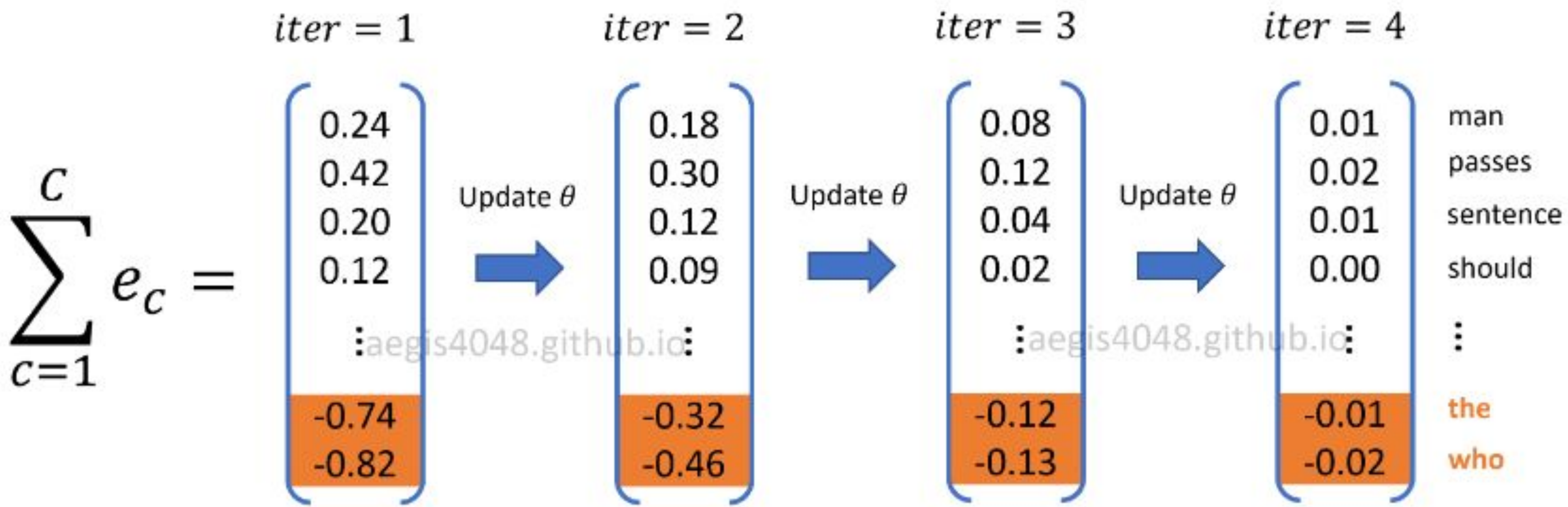


Figure 13: Prediction errors converging to zero with optimization

Corpus = "**The man** who passes the **sentence should swing the sword.**"

#	Token	x
0	man	0
1	**passes**	**1**
2	sentence	0
3	should	0
4	swing	0
5	sword	0
6	**the**	**0**
7	**who**	**0**

(1 X 8)

×

W_input

-0.078	0.018	0.033
0.068	0.170	-0.109
-0.158	-0.081	-0.151
0.150	0.064	0.145
-0.097	-0.055	0.188
0.036	0.071	0.059
0.168	-0.060	-0.058
0.098	0.015	0.096

(8X3)

Figure 14: Computing hidden (projection) layer

Corpus = "**The man** who passes the **sentence should swing the sword.**"

#	Token	x
0	man	0
1	**passes**	**1**
2	sentence	0
3	should	0
4	swing	0
5	sword	0
6	**the**	**0**
7	**who**	**0**

(1 X 8)

$$
x \times \underbrace{\begin{bmatrix} -0.078 & 0.018 & 0.033 \\ 0.068 & 0.170 & -0.109 \\ -0.158 & -0.081 & -0.151 \\ 0.150 & 0.064 & 0.145 \\ -0.097 & -0.055 & 0.188 \\ 0.036 & 0.071 & 0.059 \\ 0.168 & -0.060 & -0.058 \\ 0.098 & 0.015 & 0.096 \end{bmatrix}}_{W\_input \ (8X3)} = \underbrace{\begin{bmatrix} 0.068 \\ 0.170 \\ -0.109 \end{bmatrix}}_{h \ (1X3)}
$$

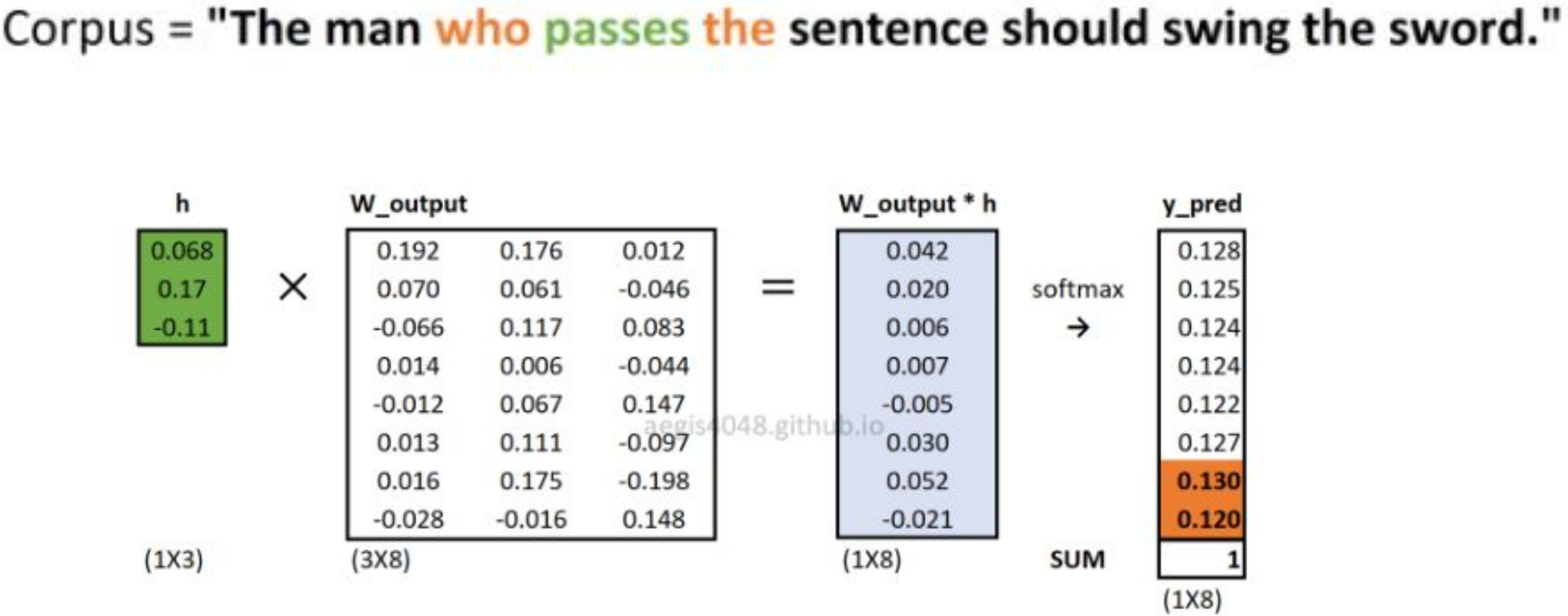


Figure 15: Softmax output layer

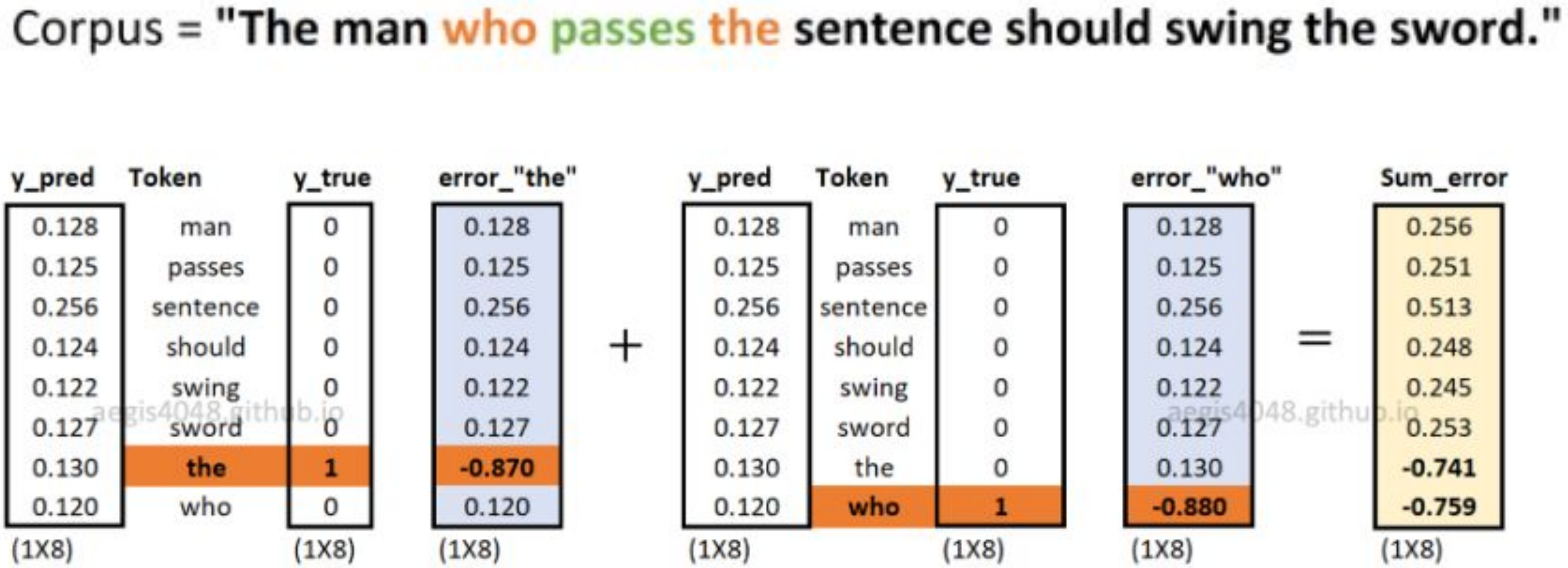


Figure 16: Prediction errors of context words

Backward propagation

$$\frac{\partial J}{\partial W_{input}} = x \cdot (W^T_{output} \sum_{c=1}^{C} e_c)$$

$$\frac{\partial J}{\partial W_{output}} = h \cdot \sum_{c=1}^{C} e_c$$

W_output

0.192	0.176	0.012
0.070	0.061	-0.046
-0.066	0.117	0.083
0.014	0.006	-0.044
-0.012	0.067	0.147
0.013	0.111	-0.097
0.016	0.175	-0.198
-0.028	-0.016	0.148

(8X3)

$\times$

Sum_error

0.256
0.251
0.513
0.248
0.245
0.253
-0.741
-0.759

(1X8)

$=$

$W^T_{output} \sum_{c=1}^{C} e_c$

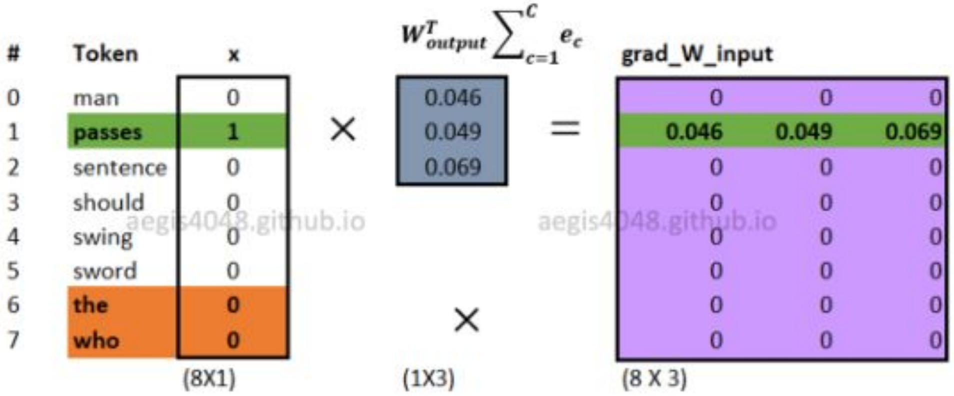
0.046
0.049
0.069

(1X3)

Backward propagation

$$\frac{\partial J}{\partial W_{input}} = x \cdot \left(W_{output}^{T} \sum_{c=1}^{C} e_c\right)$$

$$\frac{\partial J}{\partial W_{output}} = h \cdot \sum_{c=1}^{C} e_c$$



Backward propagation

$$\frac{\partial J}{\partial W_{input}} = x \cdot (W_{output}^T \sum_{c=1}^{C} e_c)$$

$$\frac{\partial J}{\partial W_{output}} = h \cdot \sum_{c=1}^{C} e_c$$

$$\underset{(1X3)}{\overset{\textbf{h}}{\begin{bmatrix} 0.068 \\ 0.17 \\ -0.109 \end{bmatrix}}} \times \underset{(1X8)}{\overset{\textbf{Sum\_error}}{\begin{bmatrix} 0.256 \\ 0.251 \\ 0.513 \\ 0.248 \\ 0.245 \\ 0.253 \\ \mathbf{-0.741} \\ \mathbf{-0.759} \end{bmatrix}}} = \underset{(8 \text{ X } 3)}{\overset{\textbf{grad\_W\_output}}{\begin{bmatrix} 0.017 & 0.044 & -0.028 \\ 0.017 & 0.043 & -0.027 \\ 0.035 & 0.087 & -0.056 \\ 0.017 & 0.042 & -0.027 \\ 0.017 & 0.042 & -0.027 \\ 0.017 & 0.043 & -0.028 \\ -0.050 & -0.126 & 0.081 \\ -0.052 & -0.129 & 0.083 \end{bmatrix}}}$$

Backward propagation

W_input (old)

-0.078	0.018	0.033
0.068	0.170	-0.109
-0.158	-0.081	-0.151
0.150	0.064	0.145
-0.097	-0.055	0.188
0.036	0.071	0.059
0.168	-0.060	-0.058
0.098	0.015	0.096

(8X3)

−

Learning R.

$\boxed{0.05}$

×

grad_W_input

0.000	0.000	0.000
0.046	0.049	0.069
0.000	0.000	0.000
0.000	0.000	0.000
0.000	0.000	0.000
0.000	0.000	0.000
0.000	0.000	0.000
0.000	0.000	0.000

(8 X 3)

=

W_input (new)

-0.078	0.018	0.033
0.066	0.168	-0.112
-0.158	-0.081	-0.151
0.150	0.064	0.145
-0.097	-0.055	0.188
0.036	0.071	0.059
0.168	-0.060	-0.058
0.098	0.015	0.096

(8X3)

Backward propagation

$$\underset{(8X3)}{\overset{\textbf{W\_output (old)}}{\begin{bmatrix} 0.192 & 0.176 & 0.012 \\ 0.070 & 0.061 & -0.046 \\ -0.066 & 0.117 & 0.083 \\ 0.014 & 0.006 & -0.044 \\ -0.012 & 0.067 & 0.147 \\ 0.013 & 0.111 & -0.097 \\ 0.016 & 0.175 & -0.198 \\ -0.028 & -0.016 & 0.148 \end{bmatrix}}} - \overset{\textbf{Learning R.}}{\boxed{0.05}} \times \underset{(8 \text{ X } 3)}{\overset{\textbf{grad\_W\_output}}{\begin{bmatrix} 0.017 & 0.044 & -0.028 \\ 0.017 & 0.043 & -0.027 \\ 0.035 & 0.087 & -0.056 \\ 0.017 & 0.042 & -0.027 \\ 0.017 & 0.042 & -0.027 \\ 0.017 & 0.043 & -0.028 \\ -0.050 & -0.126 & 0.081 \\ -0.052 & -0.129 & 0.083 \end{bmatrix}}} = \underset{(8X3)}{\overset{\textbf{W\_output (new)}}{\begin{bmatrix} 0.191 & 0.174 & 0.013 \\ 0.069 & 0.059 & -0.045 \\ -0.068 & 0.113 & 0.086 \\ 0.013 & 0.004 & -0.043 \\ -0.013 & 0.065 & 0.148 \\ 0.012 & 0.109 & -0.096 \\ 0.019 & 0.181 & -0.202 \\ -0.025 & -0.010 & 0.144 \end{bmatrix}}}$$

Glove -------> Global Vectors

Example: Window based co-occurrence matrix

- Window length 1 (more common: 5–10)
- Symmetric (irrelevant whether left or right context)
- Example corpus:
  - I like deep learning
  - I like NLP
  - I enjoy flying

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

5. Towards GloVe: Count based vs. direct prediction

- LSA, HAL (Lund & Burgess),
- COALS, Hellinger-PCA (Rohde et al, Lebret & Collobert)

- Fast training
- Efficient usage of statistics
- Primarily used to capture word similarity
- Disproportionate importance given to large counts

- Skip-gram/CBOW (Mikolov et al)
- NNLM, HLBL, RNN (Bengio et al; Collobert & Weston; Huang et al; Mnih & Hinton)

- Scales with corpus size
- Inefficient usage of statistics
- Generate improved performance on other tasks
- Can capture complex patterns beyond word similarity

Combining the best of both worlds

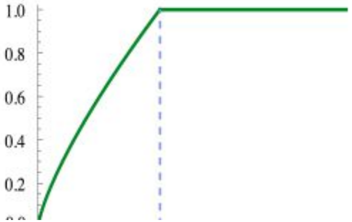
GloVe [Pennington, Socher, and Manning, EMNLP 2014]

$$w_i \cdot w_j = \log P(i|j)$$

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

- Fast training
- Scalable to huge corpora
- Good performance even with small corpus and small vectors

$f \sim$



GloVe results

Nearest words to frog:

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus

litoria

leptodactylidae

rana

eleutherodactylus

Glove vs Word2Vec

The advantage of GloVe is that, unlike Word2vec, GloVe does not rely just on local statistics (local context information of words), but incorporates global statistics (word co-occurrence) to obtain word vectors.

Word2vec relies only on ***local information*** of language. That is, the semantics learnt for a given word, is only affected by the surrounding words.

Word2Vec takes texts as training data for a neural network. The resulting embedding captures whether words appear in similar contexts. GloVe focuses on words co-occurrences over the whole corpus. Its embeddings relate to the probabilities that two words appear together

Computational Performance

PS: Recall how we trained Word2Vec model over individual context windows

References

https://towardsdatascience.com/text-vectorization-term-frequency-inverse-document-frequency-tfidf-5a3f9604da6d

https://aegis4048.github.io/demystifying_neural_network_in_skip_gram_language_modeling

https://aegis4048.github.io/optimize_computational_efficiency_of_skip-gram_with_negative_sampling

https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/slides/cs224n-2021-lecture02-wordvecs2.pdf