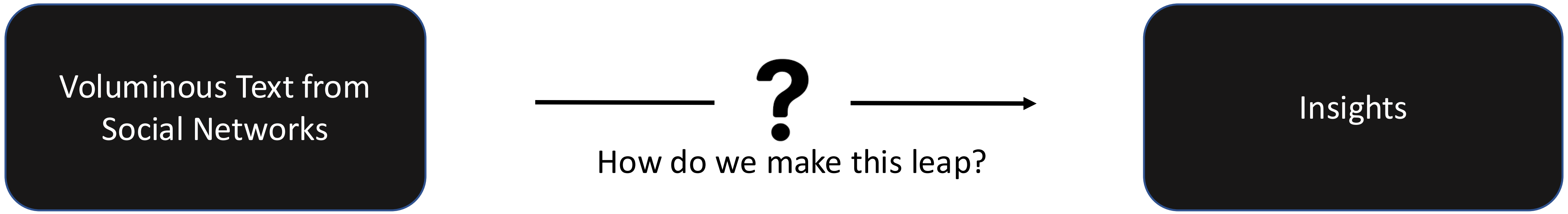
Text Embeddings

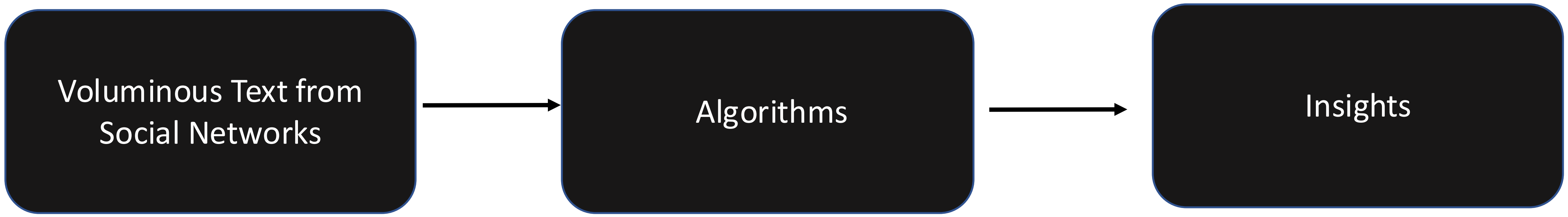
From Text towards Insights


Voluminous Text from Social Networks
?
How do we make this leap?
Insights

Posts
Comments
Reviews
.
.
.

Opinion Towards a Product
Detecting Hate Speech
What are people talking about
.
.
.

From Text towards Insights



Posts
Comments
Reviews
.
.
.

Algorithms

Rule Based Methods

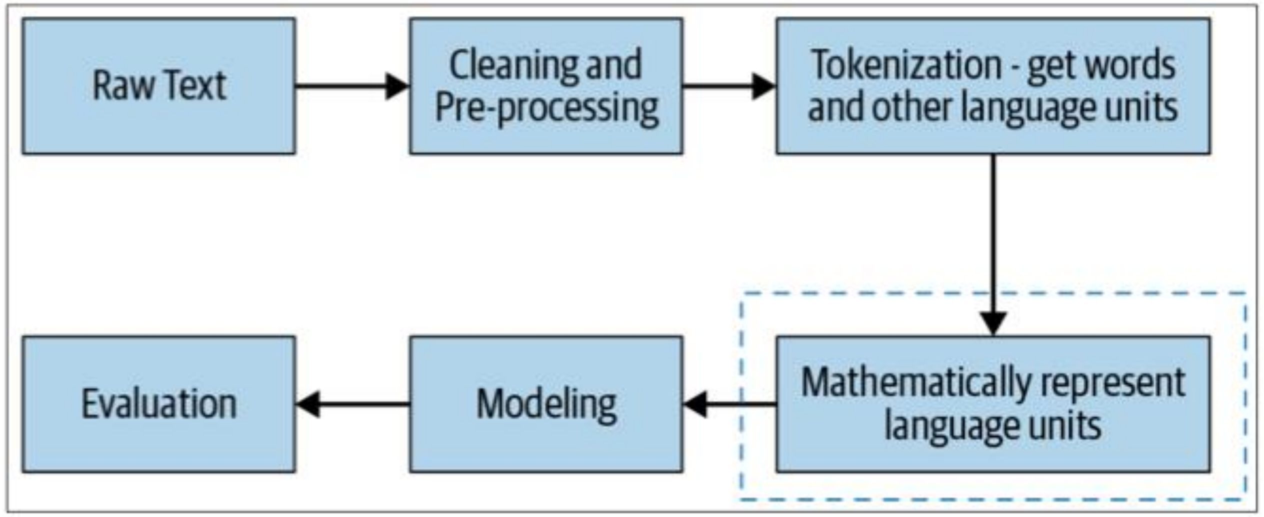
If there is abusive word == hate speech

Machine Learning Methods

Embedding based , Neural Nets etc.

Opinion Towards a Product
Detecting Hate Speech
What are people talking about
.
.
.

NLP pipeline


Raw Text
Cleaning and Pre-processing
Tokenization - get words and other language units
Mathematically represent language units
Modeling
Evaluation

Source: Practical Natural Language Processing- Oreilly

Let's delve deeper into the Algorithms

Rule Based

Hand crafted Rules .

Simple but fragile because limited by the coverage of rules

Ex : Rule for Sentiment Analysis:

R1 : if "good" present in text -> Positive Sentiment

"Movie was really good" vs "Movie was not really good."

Let's delve deeper into the Algorithms

Machine Learning Methods

- Can leverage large scale data
- Can learn complex functions to do tasks and are generally robust to noise

To leverage Machine Learning Algorithms , we need to convert text into numbers which an algorithm can crunch

Representations == Numerical representation of text.

*"Embeddings" would also mean the same thing.

A simple example of representation Bag of Words

Text Corpus

I love Virginia Tech
Alexandria has Metro

Vocabulary, V = 7 unique words

[I, love, Virginia, Tech, Alexandria, has, Metro]

I	love	Virginia	Tech	Alexandria	has	Metro

V-dim Vector
Each position is associated with a word
1 == that word is present in the sentence.

Assuming that there are only those 7 words in a language, I can represent any sentence by 7-dimensional vector

Bag of Word Representations

Text Corpus

I love Virginia Tech.
Alexandria has Metro.

Vocabulary, V = 7 unique words

[I, love, Virginia, Tech, Alexandria, has, Metro]

i	love	virginia	tech	alexandria	has	metro

V-dim Vector
Each position is associated with a word
1 == that word is present in the sentence.

i love virginia tech

1	1	1	1	0	0	0
i	love	virginia	tech	alexandria	has	metro

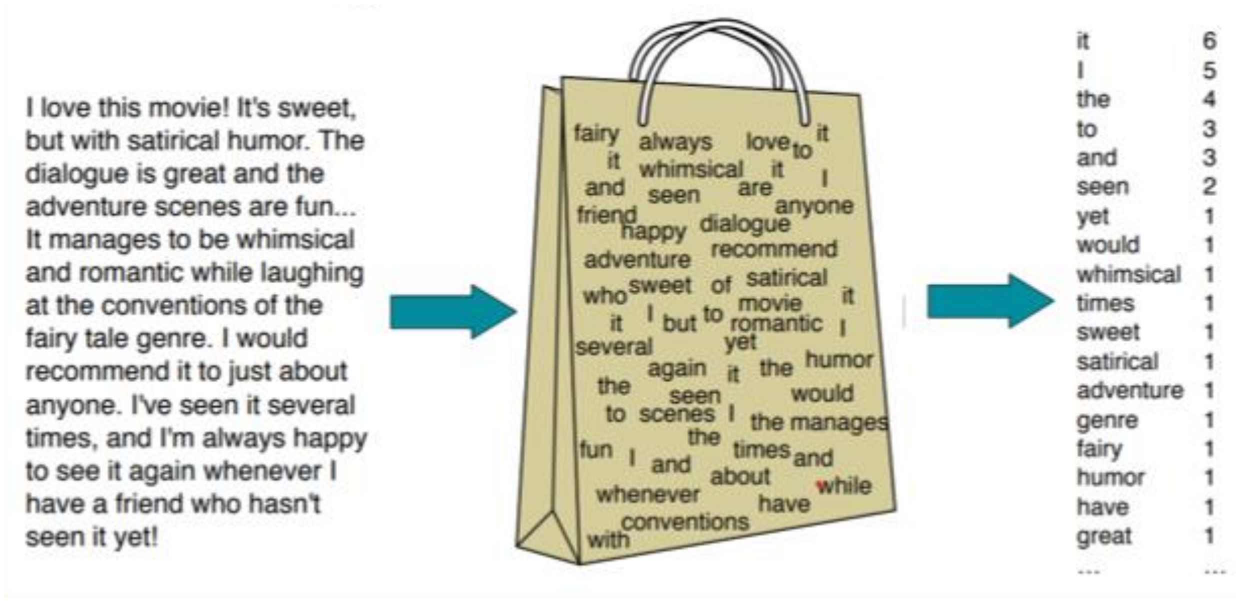
alexandria has metro

0	0	0	0	1	1	1
i	love	virginia	tech	alexandria	has	metro

virginia has metro

0	0	1	0	0	1	1
i	love	virginia	tech	alexandria	has	metro

Bag of Word Representations


I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!

it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
...	...

How to find important discriminatory words using bag?
Do Bag-of-Words representations attempt to capture meaning?

TF-IDF

- TF-IDF is a numerical statistic that reflects the importance of a word in a document. It is commonly used in NLP to represent the relevance of a term to a document or a corpus of documents.
- The TF-IDF algorithm takes into account two main factors:
  - the frequency of a word in a document, Term Frequency (TF) and
  - the frequency of the word across all documents in the corpus, Inverse Document Frequency (IDF).

Term Frequency (TF)

- The term frequency (TF) is a measure of how frequently a term appears in a document.

$$tf_{t,d} = \frac{\text{frequency of term 't' in document 'd'}}{\text{total terms in document 'd'}}$$

Formula: tf(t,d) = count of t in d / number of words in d

Inverse Document Frequency (IDF)

- The inverse document frequency (IDF) is a measure of how important a term is across all documents in the corpus. It is calculated by taking the logarithm of the total number of documents in the corpus divided by the number of documents in which the term appears. The resulting value is a number greater than or equal to 0.

$$idf_t = \log_{10} \frac{\text{total number of documents}}{\text{total documents with term 't'}}$$

Formula: $idf(t) = \log(N/(df + 1))$

Example

```
Doc1: The quick brown fox jumps over the lazy dog
Doc2: The lazy dog likes to sleep all day
Doc3: The brown fox prefers to eat cheese
Doc4: The red fox jumps over the brown fox
Doc5: The brown dog chases the fox
```

Now, let's say we want to calculate the TF-IDF scores for the word "fox" in each of these documents.

Step 1: Calculate the term frequency (TF)

- TF = (Number of times word appears in the document) / (Total number of words in the document)

Word -Fox

```
Doc1: The quick brown fox jumps over the lazy dog
Doc2: The lazy dog likes to sleep all day
Doc3: The brown fox prefers to eat cheese
Doc4: The red fox jumps over the brown fox
Doc5: The brown dog chases the fox
```

```
Doc1: 1 / 9
Doc2: 0 / 8
Doc3: 1 / 7
Doc4: 2 / 8
Doc5: 1 / 6
```

Step 2: Calculate the document frequency (DF)

- The document frequency (DF) is the number of documents in the corpus that contain the word. We can calculate the DF for the word “**fox**” as follows:

```
Doc1: The quick brown fox jumps over the lazy dog
Doc2: The lazy dog likes to sleep all day
Doc3: The brown fox prefers to eat cheese
Doc4: The red fox jumps over the brown fox
Doc5: The brown dog chases the fox
```

```
DF = 4 (Doc1, Doc3, Doc4 and Doc5)
```

Step 3: Calculate the inverse document frequency (IDF)

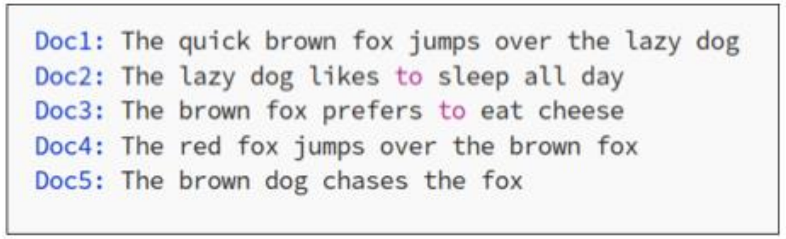
- The inverse document frequency (IDF) is a measure of how rare the word is across the corpus. It is calculated as the logarithm of the total number of documents in the corpus divided by the document frequency. In our case, we have:

```
Doc1: The quick brown fox jumps over the lazy dog
Doc2: The lazy dog likes to sleep all day
Doc3: The brown fox prefers to eat cheese
Doc4: The red fox jumps over the brown fox
Doc5: The brown dog chases the fox
```

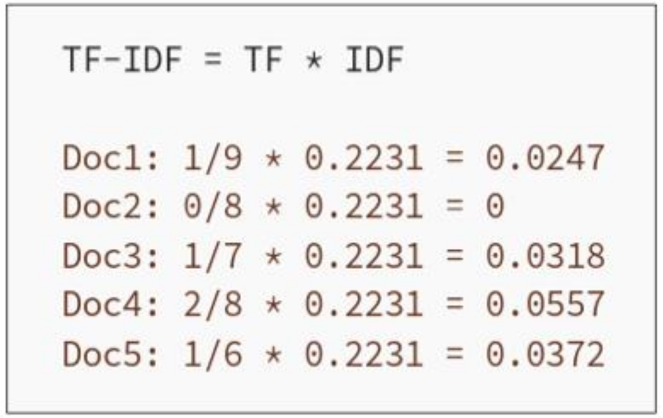
```
IDF = log(5/4) = 0.2231
```

Step 4: Calculate the TF-IDF score

- The TF-IDF score for the word “fox” in each document can now be calculated using the following formula:



```
Doc1: The quick brown fox jumps over the lazy dog
Doc2: The lazy dog likes to sleep all day
Doc3: The brown fox prefers to eat cheese
Doc4: The red fox jumps over the brown fox
Doc5: The brown dog chases the fox
```

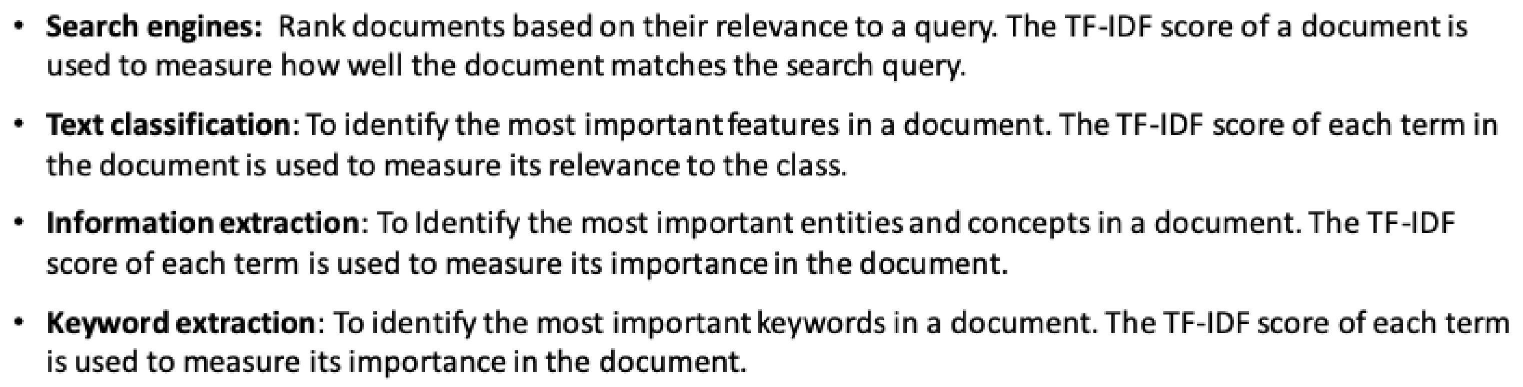


```
TF-IDF = TF * IDF

Doc1: 1/9 * 0.2231 = 0.0247
Doc2: 0/8 * 0.2231 = 0
Doc3: 1/7 * 0.2231 = 0.0318
Doc4: 2/8 * 0.2231 = 0.0557
Doc5: 1/6 * 0.2231 = 0.0372
```

Why log in the formula?

Applications of TF-IDF



- **Search engines:** Rank documents based on their relevance to a query. The TF-IDF score of a document is used to measure how well the document matches the search query.
- **Text classification**: To identify the most important features in a document. The TF-IDF score of each term in the document is used to measure its relevance to the class.
- **Information extraction**: To Identify the most important entities and concepts in a document. The TF-IDF score of each term is used to measure its importance in the document.
- **Keyword extraction**: To identify the most important keywords in a document. The TF-IDF score of each term is used to measure its importance in the document.

What happens when the vocabulary is too large?

What about the order of words in sentence?

What about semantics? A sentence that does not represent any meaning?

Learned Representations – Neural Methods

To do complex tasks, we need richer representations

Representations should attempt to capture semantics (meaning) of language

How can we do this?

Learned Representations – Neural Methods

To do complex tasks, we use representations

Representations should attempt to capture semantics (meaning) of language

How can we do this?
Distributed Semantics . "A word is known by the company it keeps" - Firth

If I can train model to guess its neighboring words, maybe they can capture semantics a bit better

Distributional similarity based representations

You can get a lot of value by representing a word by means of its neighbors

"You shall know a word by the company it keeps"

(J. R. Firth 1957: 11)

One of the most successful ideas of modern statistical NLP

government debt problems turning into **banking** crises as has happened in

saying that Europe needs unified **banking** regulation to replace the hodgepodge

Word meaning is defined in terms of vectors

We will build a dense vector for each word type, chosen so that it is good at predicting other words appearing in its context

... those other words also being represented by vectors ... it all gets a bit recursive

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

Word meaning is defined in terms of vectors

We will build a dense vector for each word type, chosen so that it is good at predicting other words appearing in its context

... those other words also being represented by vectors ... it all gets a bit recursive

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

What type of vector do we need? Something better than TF-IDF and bag?

Word meaning is defined in terms of vectors

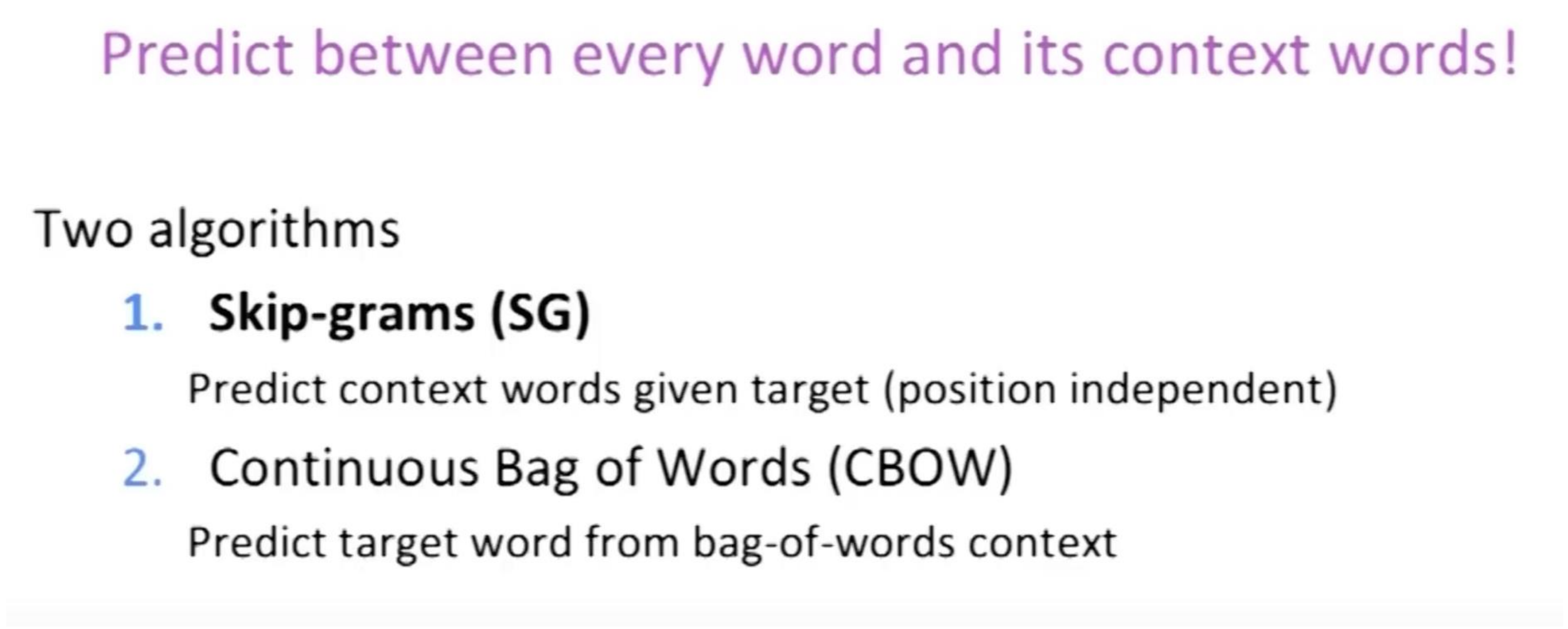
We will build a dense vector for each word type, chosen so that it is good at predicting other words appearing in its context

... those other words also being represented by vectors ... it all gets a bit recursive

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

Dense vector capturing meaning. Its size should not depend on vocabulary

Word2Vec Embedding



Predict between every word and its context words!

Two algorithms

1. **Skip-grams (SG)**

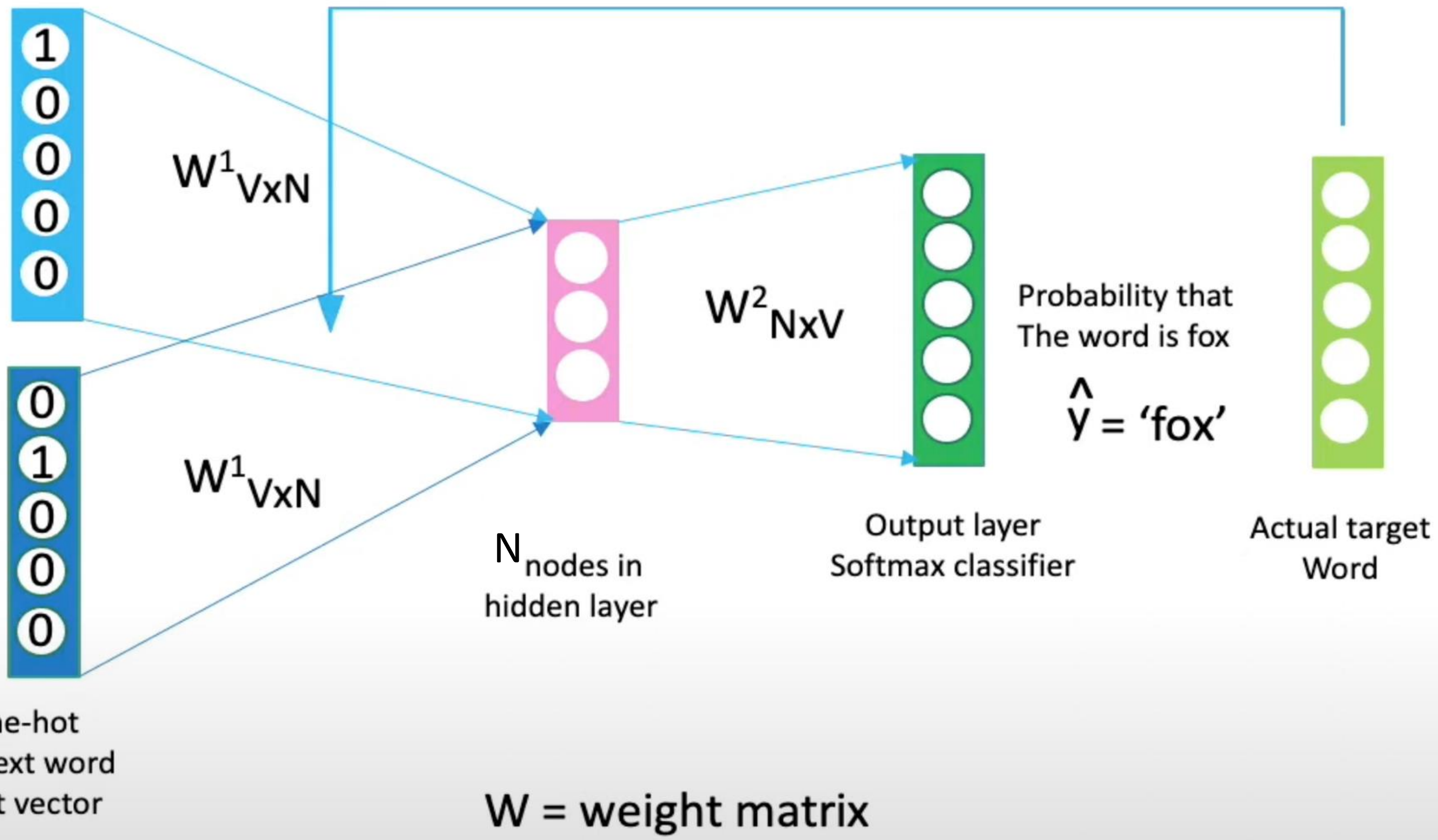
   Predict context words given target (position independent)

2. Continuous Bag of Words (CBOW)

   Predict target word from bag-of-words context

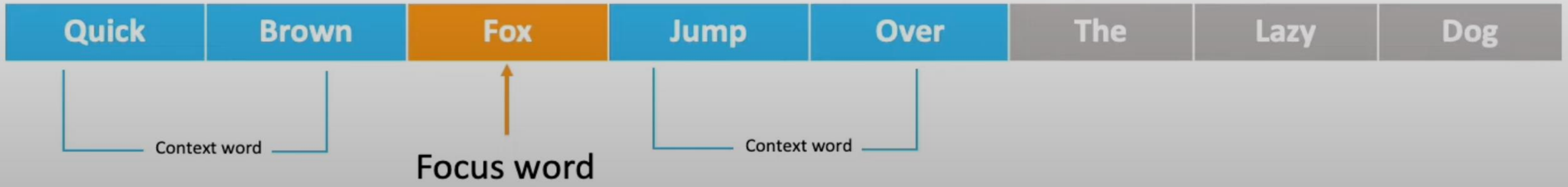
Continuous Bag Of Word (CBOW)

Order of the words does not matter



P(Fox | context words)
Maximize prob of focus word
Given context words

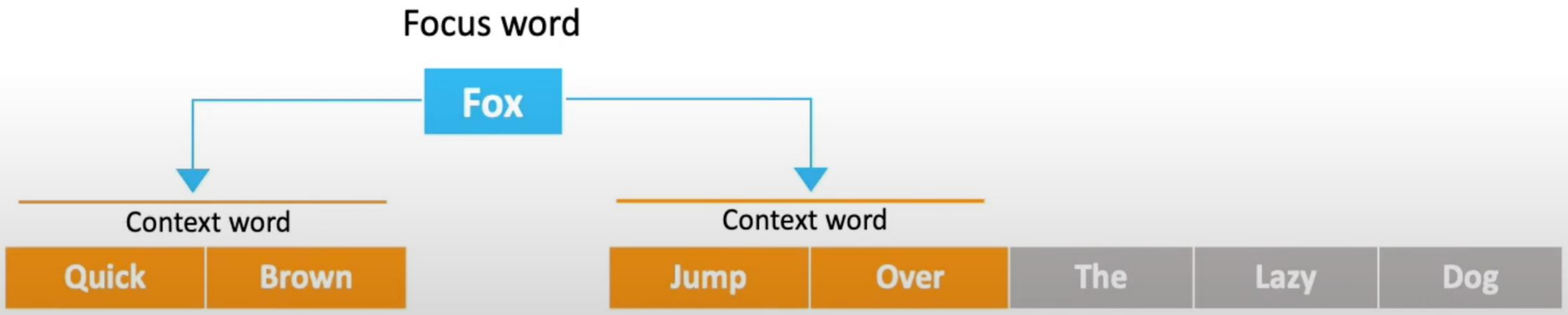
W = weight matrix



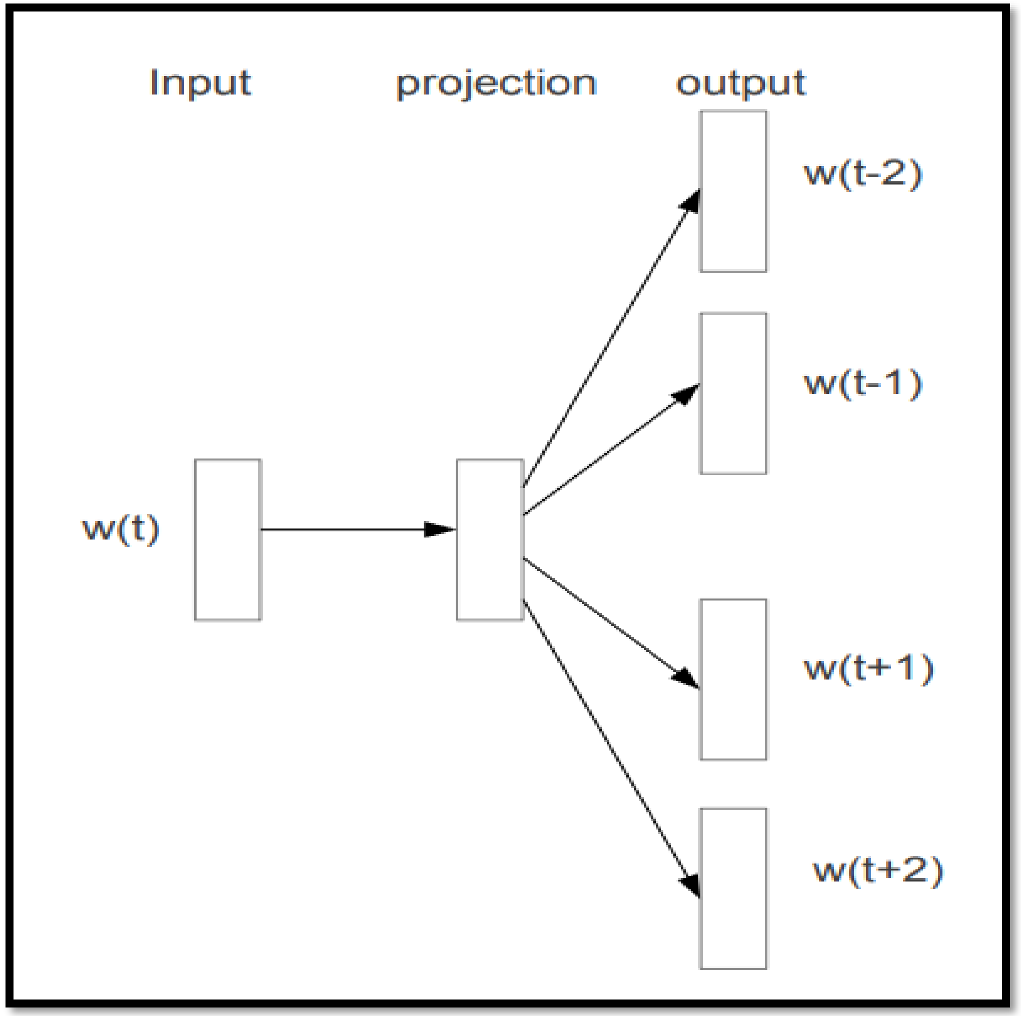
Source Text	Training Samples generated from source text			
I will have orange juice and eggs for breakfast	(will, I)	(will, have)	(will, orange)	
I will have orange juice and eggs for breakfast	(have, I)	(have, will)	(have, orange)	(have, juice)
I will have orange juice and eggs for breakfast	(orange, will)	(orange, have)	(orange, juice)	(orange, and)
I will have orange juice and eggs for breakfast	(juice, have)	(juice, orange)	(juice, and)	(juice, eggs)
I will have orange juice and eggs for breakfast	(and, orange)	(and, juice)	(and, eggs)	(and, for)
I will have orange juice and eggs for breakfast	(eggs, juice)	(eggs, and)	(eggs, for)	(eggs, breakfast)
I will have orange juice and eggs for breakfast	(for, and)	(for, eggs)	(for, breakfast)	

Skip-Gram

1. Inverse of CBOW model
2. Input vector - single focus word , and the target are context words
3. Typical window size = 5
4. Objective: minimize the summed prediction error across all context words in the output layer.



5. We want higher probabilities for Context word

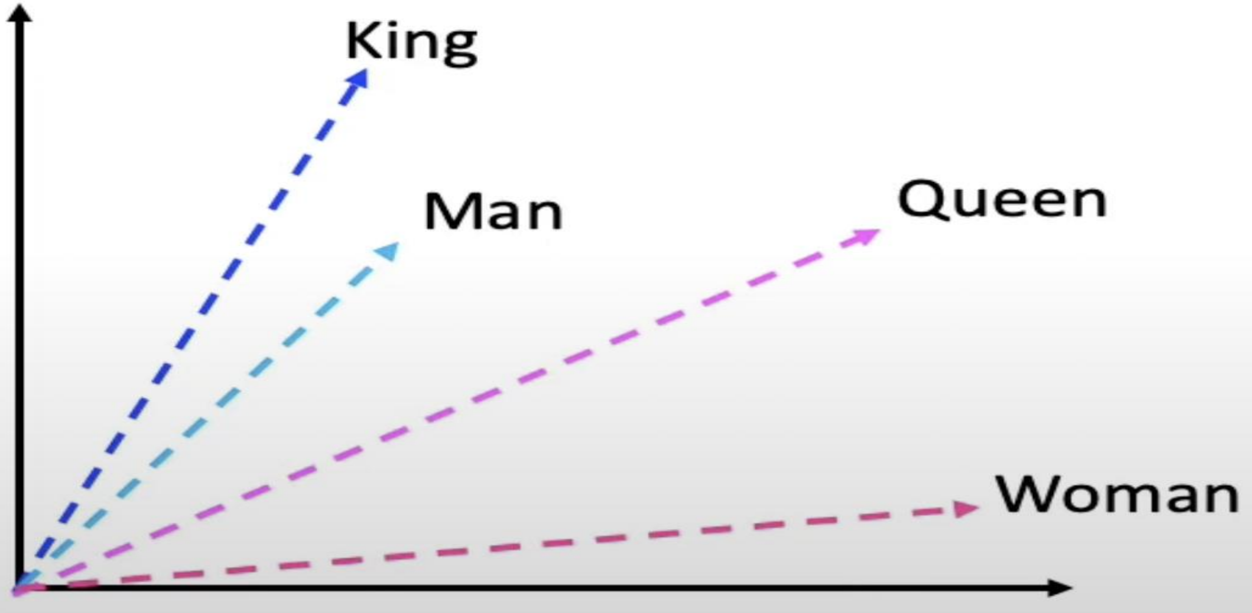


Distributed Representations of Words and Phrases and their Compositionality, Mikolov et. Al 2013

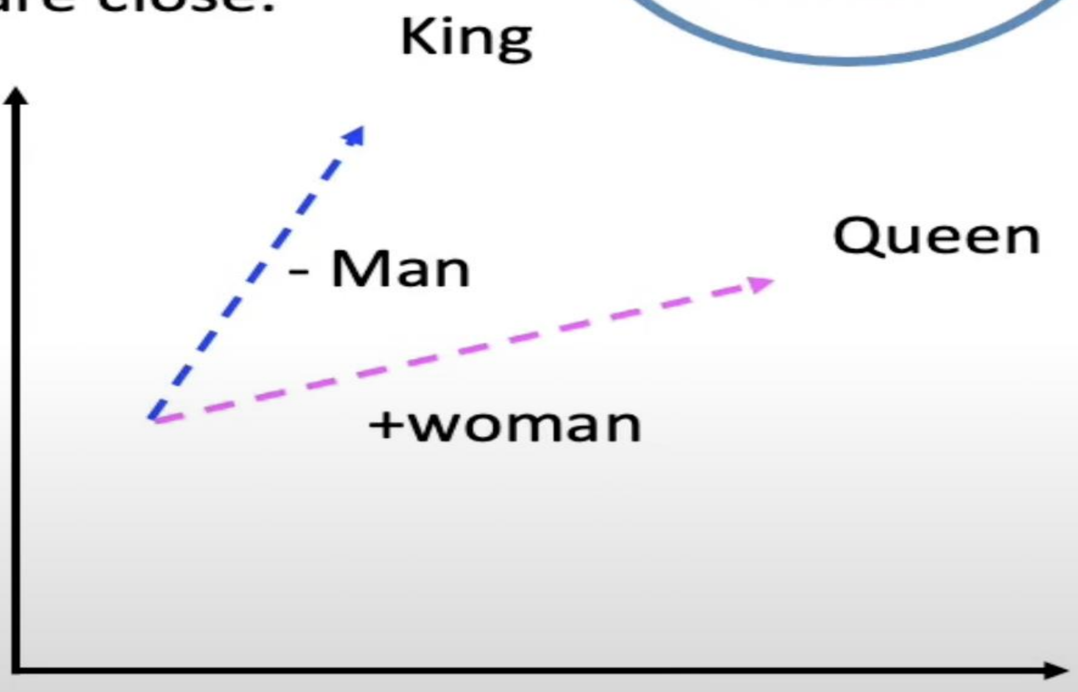
General Overview

- Map word into vector space
- Makes the words that have a similar context to have similar embeddings
- Thus, in this vector space, these words are close.

Each word is transformed into a distributed representation of its weight across all Other words



Word Vectors



Vector Composition

When to use CBOW?

1. Much faster to train than skip-gram
2. It is low on memory. It does not need to have huge RAM requirements.
3. Slightly better accuracy for frequent words

Why faster than skip gram?

When to use Skip-Gram?

1. Works well with small amount of training data.
2. When words and phrases are rare.

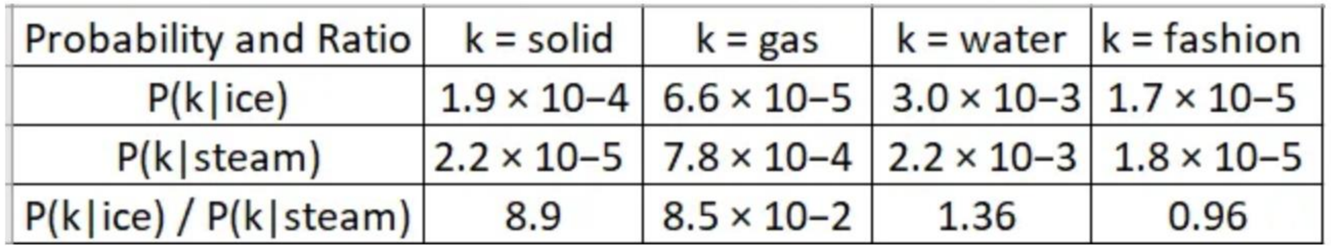
GloVe Embeddings

Global Vectors for Word Representation

Window based co-occurrence matrix

- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

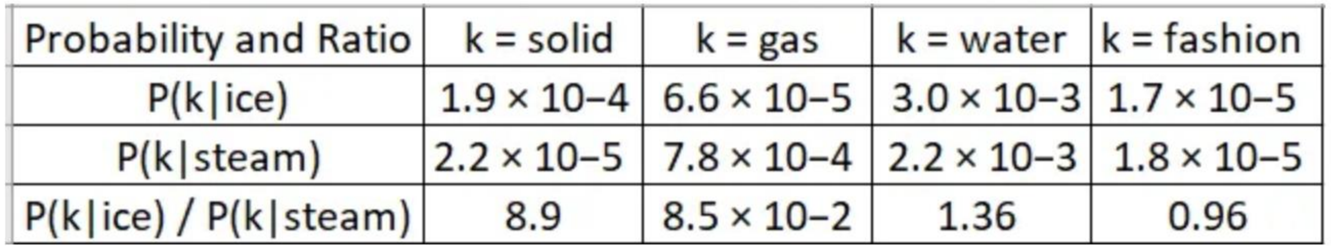
counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0



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

Table 1: Co-occurrence probability from 6 billion token corpus (Source — GloVe research paper)

What do you infer from the above table?



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

Table 1: Co-occurrence probability from 6 billion token corpus (Source — GloVe research paper)

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

GloVe Vectors

The Euclidean distance (or cosine similarity) between two word vectors provides an effective method for measuring the linguistic or semantic similarity of the corresponding words. Sometimes, the nearest neighbors according to this metric reveal rare but relevant words that lie outside an average human's vocabulary. For example, here are the closest words to the target word *frog*:

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

3. litoria

4. leptodactylidae

5. rana

7. eleutherodactylus

Pretrained Vector Embeddings

Glove trained on Wikipedia corpus
Glove trained on Twitter corpus

Word2Vec on Wikipedia
Word2Vec on Google News

Acknowledgements

Dr. Ponnurangam Kumaraguru
Dr. Nidhi Goel
Stanford and YouTube lectures on text analysis