Week 10:
Natural Language Processing (NLP)

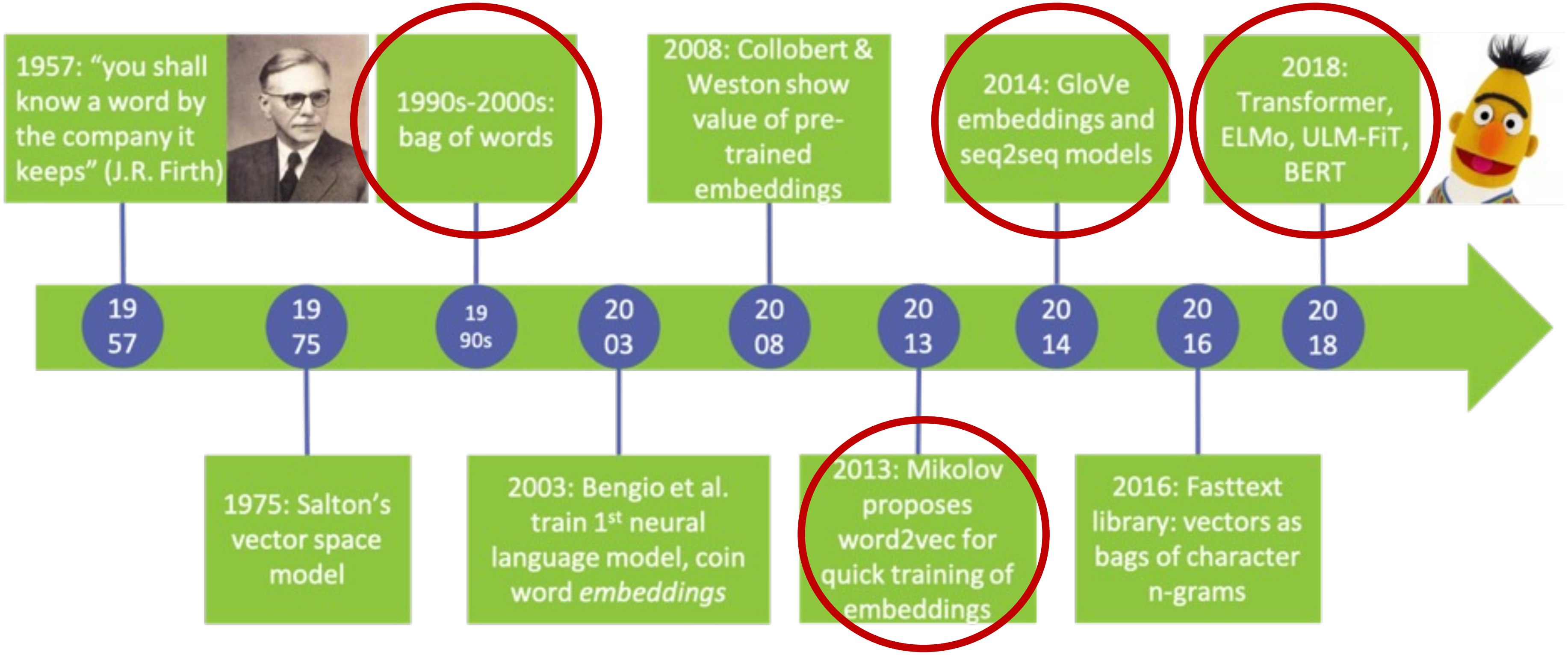
Vinay P. Namboodiri

UNIVERSITY OF BATH

Topic 1: Introduction to NLP

What is NLP?

- **Natural language processing (NLP)** is a subfield of computer science that focuses on developing algorithms and computational models to analyze, understand, and generate human language.



Challenges of NLP

- Variable input size:
  - “The alien mothership is in orbit here! If we hit that bullseye, the rest of the dominoes will fall like a house of cards! Checkmate!” – 25 words
  - “Stop exploding you cowards!” - 4 words
- Sensitive: Small changes can have large effects
  - “Let’s eat, Jack.” vs “Let’s eat Jack.” (comma)
  - ”Dog bites man.” vs “Man bites dog.” (word order)
  - ”I miss home.” vs “We might miss the train.” (same word, different meaning)
  - “I hit the man with a stick.” (who is holding the stick?)
- Redundant: Many ways to say the same thing
  - “The same thing can be said in many different ways.”
  - “There are a plurality of methods for communicating an identical concept.”
- …
- More difficult: common sense, culture information

NLP applications

- Text Classification
- Named Entity Recognition
- Document search
- Sentiment Analysis
- Text Summarization
- Topic Modelling
- Machine Translation
- Question Answering
- Chatbot
- ….

Early machine translation

- Georgetown-IBM experiment (1954): The Georgetown-IBM experiment was the first demonstration of machine translation, where researchers attempted to translate Russian sentences into English.

Early chatbot

- ELIZA (1966) was one of the first chatbots and one of the first programs capable of attempting the Turing test.
- Rule-based

```
  Welcome to
                    EEEEEE  LL      IIII   ZZZZZZ   AAAAA
                    EE      LL       II       ZZ   AA   AA
                    EEEEE   LL       II     ZZZ    AAAAAAA
                    EE      LL       II    ZZ      AA   AA
                    EEEEEE  LLLLLL  IIII ZZZZZZ    AA   AA

  Eliza is a mock Rogerian psychotherapist.
  The original program was described by Joseph Weizenbaum in 1966.
  This implementation by Norbert Landsteiner 2005.


ELIZA: Is something troubling you ?
YOU:   Men are all alike.
ELIZA: What is the connection, do you suppose ?
YOU:   They're always bugging us about something or other.
ELIZA: Can you think of a specific example ?
YOU:   Well, my boyfriend made me come here.
ELIZA: Is it important to you that your boyfriend made you come here ?
YOU:   He says I'm depressed much of the time.
ELIZA: I am sorry to hear that you are depressed.
YOU:   It's true. I am unhappy.
ELIZA: Can you explain what made you unhappy ?
YOU:
```

Topic 2: Text Pre-Processing

Tokenization

- **Tokenization** is the process of breaking down the given text in MLP into smallest unit in a sentence called a **token**.

```
import nltk
from nltk import word_tokenize
text = "Let’s first tokenize the sentence into words using nltk.word_tokenize(). "
word_list = nltk.word_tokenize(text)
print(word_list)
```

```
['Let', '’', 's', 'first', 'tokenize', 'the', 'sentence', 'into', 'words',
'using', 'nltk.word_tokenize', '(', ')', '.']
```

Q: why do we need tokenization?

Handling unknown words

- What happens when we encounter a word at test time that we've never seen in our training data?
  - With word level tokenization, we have no way of assigning an index to an unseen word!
  - This means we don't have a word embedding for that word and thus cannot process the input sequence
- Solution: replace low-frequency words in training data with a special <UNK> token, use this token to handle unseen words at test time too
  - Why use <UNK> tokens during training?

From Mohit Iyer

Limitations of <UNK>

- We lose lots of information about texts with a lot of rare words / entities

The chapel is sometimes referred to as "Hen Gapel Lligwy" ("*hen*" being the Welsh word for "old" and "*capel*" meaning "chapel").

The chapel is sometimes referred to as " Hen <unk> <unk> " (" hen " being the Welsh word for " old " and " <unk> " meaning " chapel ").

Other limitations

- Word-level tokenization treats different forms of the same word (e.g., “open”, “opened”, “opens”, “opening”, etc) as separate types —> separate embeddings for each

This can be problematic especially when training over smaller datasets, why?

From Mohit Iyer

An alternative: character tokenization

- Small vocabulary, just the number of unique characters in the training data!
- However, you pay for this with longer input sequences. Why is this a problem for the models we've discussed?

From Mohit Iyer

An alternative: character tokenization

- Small vocabulary, just the number of unique characters in the training data!

- However, you pay for this with longer input sequences. Why is this a problem for the models we've discussed?

From Mohit Iyer

Byte pair encoding

- Form base vocabulary (all characters that occur in the training data)

word	frequency
hug	10
pug	5
pun	12
bun	4
hugs	5

- Base vocab: b, g, h, n, p, s, u

Byte pair encoding

- Now, count up the frequency of each character *pair* in the data, and choose the one that occurs most frequently

word	frequency
h+u+g	10
p+u+g	5
p+u+n	12
b+u+n	4
h+u+g+s	5

character pair	frequency
ug	20
pu	17
un	16
hu	15
gs	5

...

Mohit Iyer - Example from https://huggingface.co/transformers/tokenizer_summary.html

Byte pair encoding

- Now, choose the most common pair (ug) and then merge the characters together into one symbol. Add this new symbol to the vocabulary. Then, retokenize the data

word	frequency
h+*ug*	10
p+*ug*	5
p+u+n	12
b+u+n	4
h+*ug*+s	5

character pair	frequency
un	16
h+ug	15
pu	12
p+ug	5
ug+s	5

...

Mohit Iyer - Example from https://huggingface.co/transformers/tokenizer_summary.html

Byte pair encoding

- Keep repeating this process! This time we choose *un* to merge, next time we choose *h+ug*, etc.

word	frequency
h+*ug*	10
p+*ug*	5
p+u+n	12
b+u+n	4
h+*ug*+s	5

character pair	frequency
un	16
h+ug	15
pu	12
p+ug	5
ug+s	5

...

Mohit Iyer - Example from https://huggingface.co/transformers/tokenizer_summary.html

Byte pair encoding

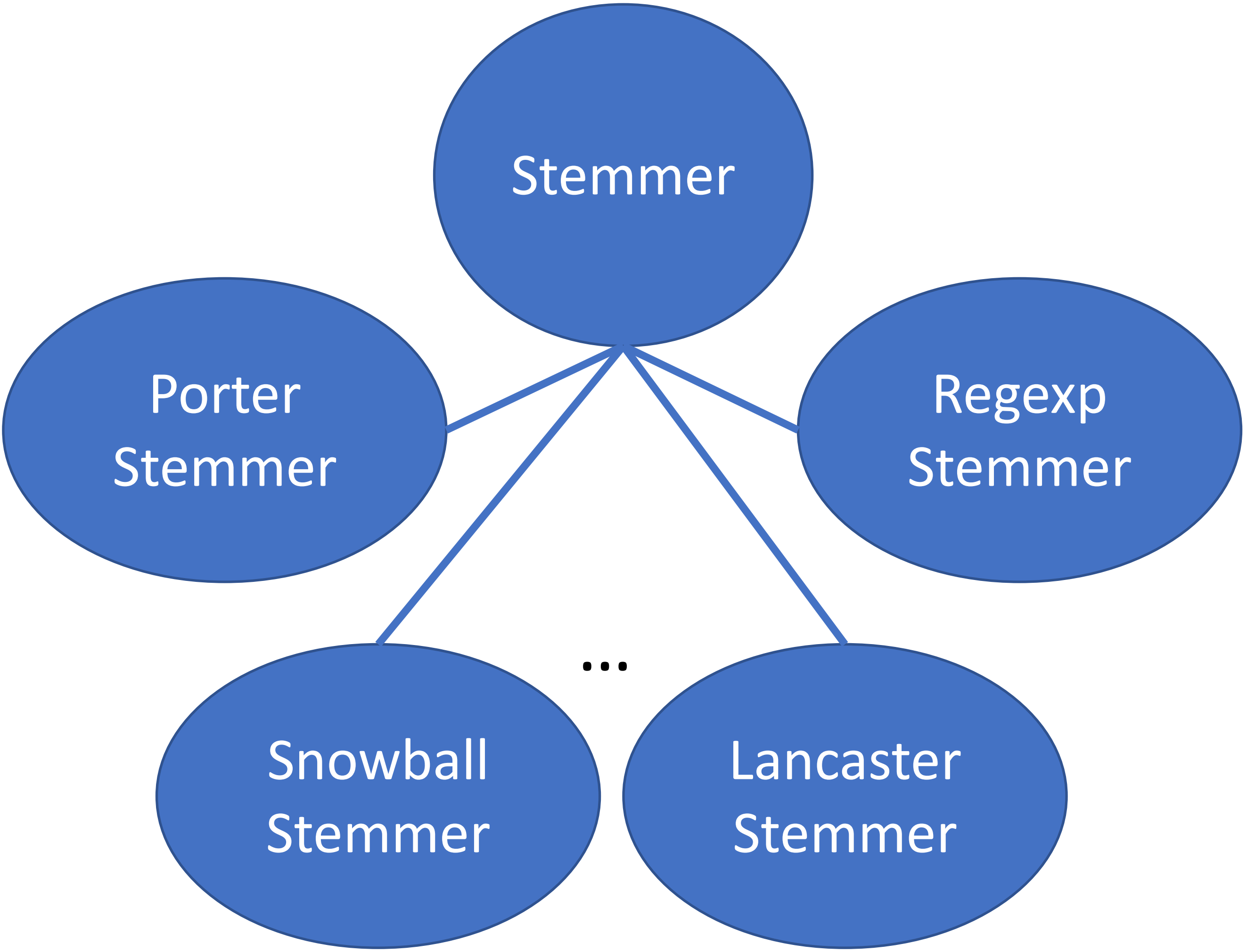
- Eventually, after a fixed number of merge steps, we stop

word	frequency
hug	10
p+*ug*	5
p+*un*	12
b+*un*	4
hug + s	5

- new vocab: b, g, h, n, p, s, u, *ug*, *un*, *hug*

Stemming

- Problem: lots of words! (150– 500k, many ways to account)
- **Stemming**: is the process of finding the root of words.
- Example:
  - “like”, “likes”, “liked”, “likely”, “liking” ➔ “like”
  - “warm”, “warmer”, “warmed” ➔ “warm”



- Rules based (E.g., Porter stemmer: 5-step rules)

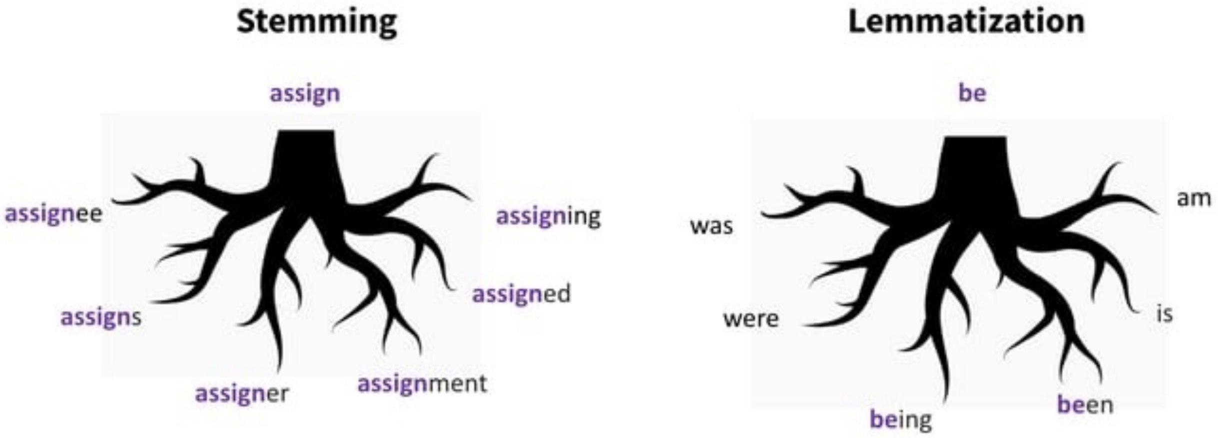
```
import nltk
from nltk.stem import PorterStemmer
ps = PorterStemmer()
words = ['caresses', 'flies', 'dies', 'mules', 'denied',
'died', 'agreed', 'owned', 'humbled', 'sized',
'meeting', 'stating', 'siezing', 'itemization',
'sensational', 'traditional', 'reference', 'plotted']
words_after_stem = [ps.stem(word) for word in words]
print(' '.join(words_after_stem))
```

```
caress fli die mule deni die agre own
humbl size meet state siez item sensat
tradit refer plot
```

The complete Rules: https://tartarus.org/martin/PorterStemmer/def.txt

Lemmatization

- Also reduce words to their base or root form.
- **Lemmatization** produces a valid base word that is a morphological variant of the original word, while stemming simply chops off the suffixes of the word.
- Examples:
    - “ran” ➔”run”, “better” ➔”good”, “am”,”is”,”was”➔”be”



Stop word removal

- For some of the applications it is also useful to omit the common stop words that are the most frequent words such as ‘a’, ‘an’, ‘the’.

```python
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
print(stopwords.words('english'))
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
"you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himsel
f', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'the
m', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "tha
t'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have',
 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'i
f', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'a
gainst', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to
', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', '
then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'eac
h', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 's
ame', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'shou
ld', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", '
couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "has
n't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't",
 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren',
 "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

Part-Of-Speech Tagging (POS Tag)

- **Part of Speech Tagging (POS-Tag)** is the labelling of the words in a text according to their word types (noun, adjective, adverb, verb, etc.).

Img from: https://blog.aaronccwong.com/2019/building-a-bigram-hidden-markov-model-for-part-of-speech-tagging/

Part-Of-Speech Tagging (POS Tag)

UNIVERSITY OF BATH

- **Part of Speech Tagging (POS-Tag)** is the labelling of the words in a text according to their word types (noun, adjective, adverb, verb, etc.).

```
import nltk
from nltk import word_tokenize
text = "The striped bats are hanging on their feet for best"
tokens = nltk.word_tokenize(text)
print("Parts of Speech: ",nltk.pos_tag(tokens))
```

Verb, non-3rd person singular present

Determiner

Adjective

Noun, plural

```
Parts of Speech:  [('The', 'DT'), ('striped', 'JJ'), ('bats', 'NNS'), ('are', 'VBP'),
 ('hanging', 'VBG'), ('on', 'IN'), ('their', 'PRP$'), ('feet', 'NNS'), ('for', 'IN'),
 ('best', 'JJS')]
```

Full list of pos tags : https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

Reference

- https://medium.com/mlearning-ai/nlp-tokenization-stemming-lemmatization-and-part-of-speech-tagging-9088ac068768

Topic 3: BoW and TF-IDF

Feature representation of a document

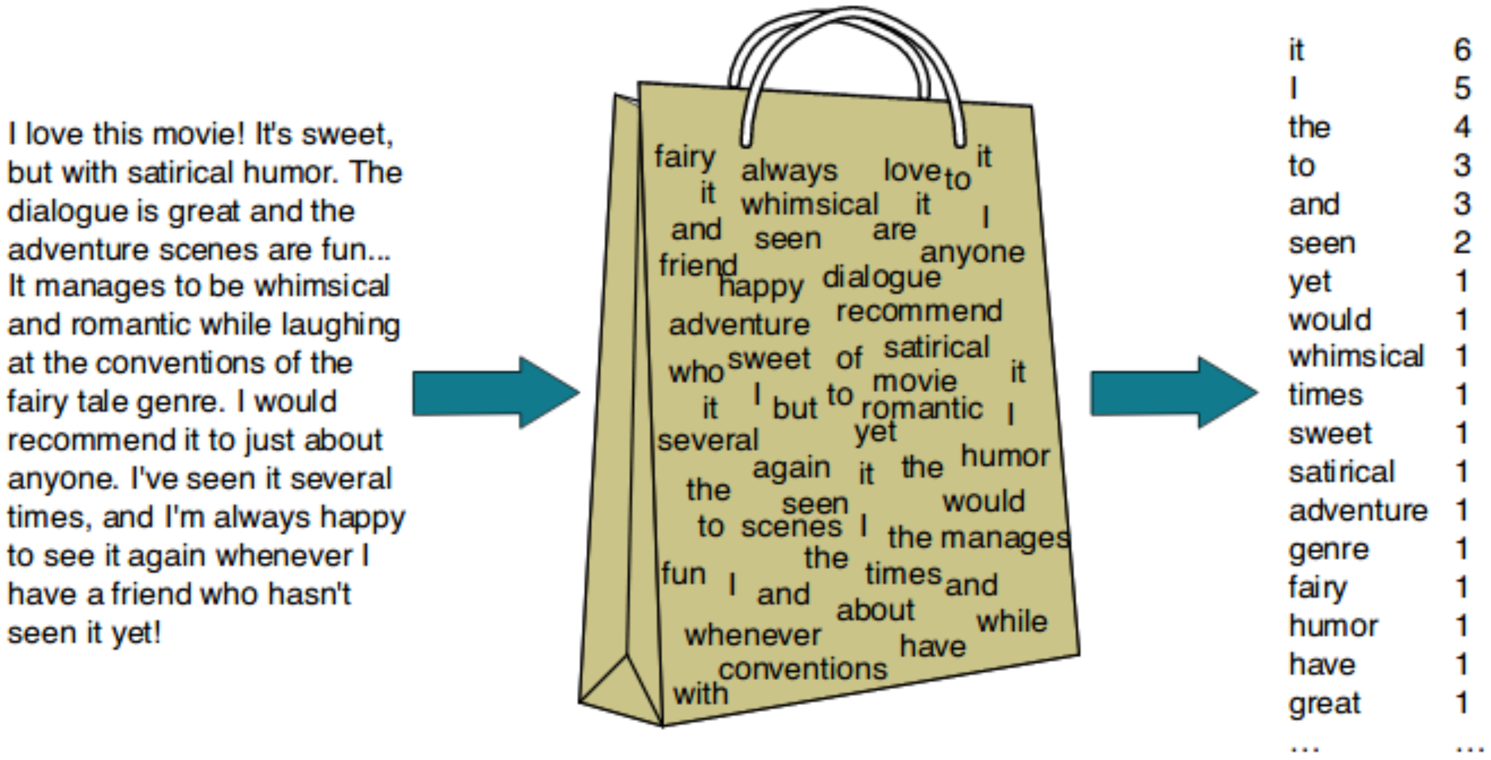
I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy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

$$\begin{bmatrix} 0.2 \\ 0 \\ 0.71 \\ -0.54 \\ 0.4 \\ 0.22 \\ 0.43 \\ \dots \\ -0.81 \end{bmatrix}$$

A sequence of words

A fix-length vector

Bag-of-words (BoW)



- Feature vector of a document = word frequency histogram
- A very long vector, mostly zeros

Q: Which of the following words are helpful for you to predict a document's topic?

"is", "get", "have", "cosine", "angle", "equal"

More frequent words $\neq$ More important

Solution: Higher weights are given to words which rarely occur in the corpus: **TF-IDF**

Term Frequency – Inverse Document Frequency (TF-IDF)

$$\mathrm{TF{-}IDF}(t,d) = \mathrm{TF}(t,d) \times \mathrm{IDF}(t)$$

Term Frequency: $\mathrm{TF}(t,d) = \dfrac{\text{Occurrence of term } t \text{ in document } d}{\text{Number of terms in } d} = \dfrac{n_{td}}{n_d}$

Inverse Document Frequency: $\mathrm{IDF}(t) = \log \dfrac{\text{Number of documents in the corpus}}{\text{Number of documents with term } t \text{ in it}} = \log \dfrac{N}{n_t}$

Example

- In a corpus of 10000 documents, you pick a document D, which has a total of 2000 words.

Term (t)	Occurrence in D	Number of documents contains t	TF(t, D)	IDF(t)	TF-IDF
get	30	6000			
cosine	6	10			

$$\mathrm{TF}(t,d) = \frac{\text{Occurrence of term } t \text{ in document } d}{\text{Number of terms in } d} = \frac{n_{td}}{n_d}$$

$$\mathrm{IDF}(t) = \log \frac{\text{Number of documents in the } corpus}{\text{Number of documents with term } t \text{ in it}} = \log \frac{N}{n_t}$$

Example

- In a corpus of 10000 documents you pick a document D, which has a total of 2000 words.

Term (t)	Occurrence in D	Number of documents contains t	TF(t, D)	IDF(t)	TF-IDF
get	30	6000	30/2000	Log(10000/6000)	0.0077
cosine	6	10	6/2000	Log(10000/10)	0.0207

$$\mathrm{TF}(t, d) = \frac{\text{Occurrence of term } t \text{ in document } d}{\text{Number of terms in } d} = \frac{n_{td}}{n_d}$$

$$\mathrm{IDF}(t) = \log \frac{\text{Number of documents in the } corpus}{\text{Number of documents with term } t \text{ in it}} = \log \frac{N}{n_t}$$

Drawbacks of BoW and TF-IDF

- Very long vector
- Ignore the word order
- Treat words independently
  - Clearly naive
  - It works well for many applications. E.g., sentiment analysis, topic identification, spam filtering, etc.

Reference

- Y. Hamdaoui's blog: "TF-IDF from scratch in python". https://towardsdatascience.com/tf-term-frequency-idf-inverse-document-frequency-from-scratch-in-python-6c2b61b78558

Topic 4: Introduction of Word Embedding

What is word embedding?

- **Word Embedding**: converting a word/phrase into a fix-length vector.

I like playing football.

$$\begin{bmatrix} 0.1 \\ 0.9 \\ 1.2 \\ \dots \\ -1.3 \end{bmatrix}\begin{bmatrix} 0.1 \\ 0.9 \\ 1.2 \\ \dots \\ -1.3 \end{bmatrix}\begin{bmatrix} 0.1 \\ 0.9 \\ 1.2 \\ \dots \\ -1.3 \end{bmatrix}\begin{bmatrix} 0.1 \\ 0.9 \\ 1.2 \\ \dots \\ -1.3 \end{bmatrix}$$

Simplest word embedding

- One-hot vector

I like playing football.

$$\begin{bmatrix}1\\0\\0\\\dots\\0\end{bmatrix} \quad \begin{bmatrix}0\\0\\1\\\dots\\0\end{bmatrix} \quad \begin{bmatrix}0\\1\\0\\\dots\\0\end{bmatrix} \quad \begin{bmatrix}0\\0\\0\\\dots\\1\end{bmatrix}$$

Q: What is the drawback of using one-hot vector?

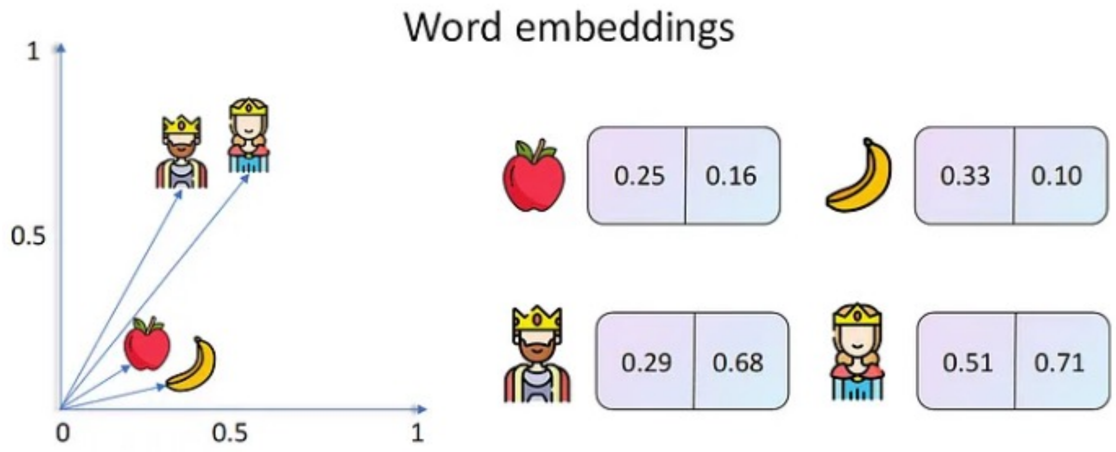
Drawbacks of one-hot vector

- The vector length is huge.
- The embedding is closely coupled to their application, requiring re-training the whole model, if the vocabulary changed.
- No context of words. All words have the same distance.

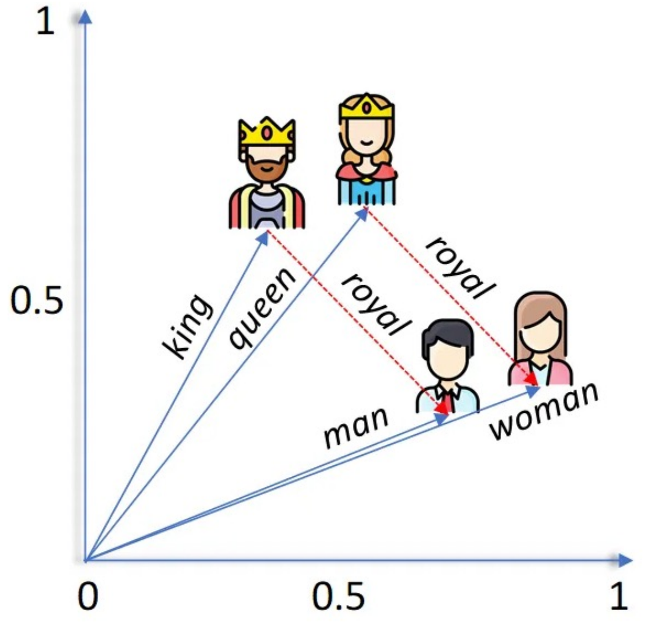
Img from: https://mlwhiz.com/blog/2017/04/09/word_vec_embeddings_examples_understanding/

What is a good word embedding?

- Word embedding aims at words that appear in similar contexts or have similar meaning are close together.



Q: v(King) – v(man) + v(woman) = ?

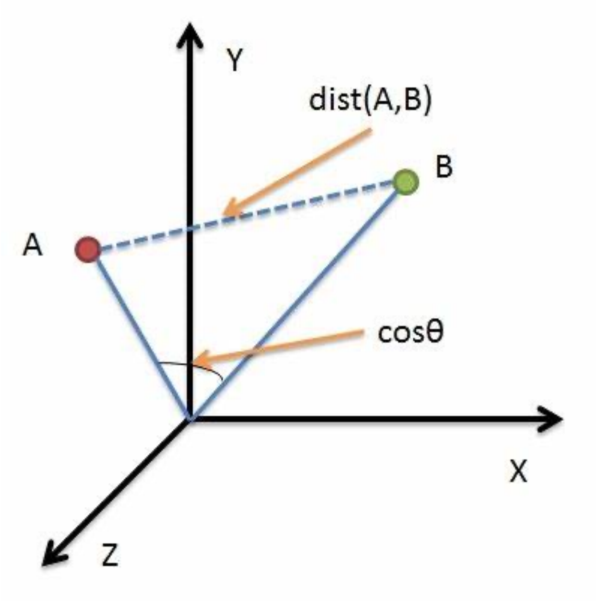


Img from: https://towardsdatascience.com/deep-learning-for-nlp-word-embeddings-4f5c90bcdab5

Similarity of words

- **Cosine similarity** is the most well known to measure how close two vectors are.

$$\text{Cos\_similarity}(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i-1}^{n} A_i^2}\sqrt{\sum_{i-1}^{n} B_i^2}}$$



Word embedding methods

SVD-based methods

Based on matrix factorization of a global word co-occurrence matrix.

Iteration based methods

Word2Vec (Mikolov et al. 2013)

- learn the underlying word representation by using neural networks.
- Two models: CBOW and Skip-gram

GloVe (Pennington et al. 2014)

- Learn word embedding by using a co-occurrence statistics of words in a corpus.

Word2Vec and GloVe are two most popular word embedding methods

Reference

- Lecture notes CS224D: Deep Learning Part 1: http://cs224d.stanford.edu/lecture_notes/notes1.pdf
- Lectures CS224n: NLP processing with Deep Learning. https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/syllabus.html

Topic 5: Word Embedding – SVD-Based Methods

- **Step 1**: First loop over a massive dataset and accumulate word co-occurrence matrix X
- **Step 2**: Perform Singular Value Decomposition on X to get a decomposition.
- **Step 3**: Use the rows of the singular matrix as the word embeddings for all words in our dictionary.

Window-based co-occurrence matrix

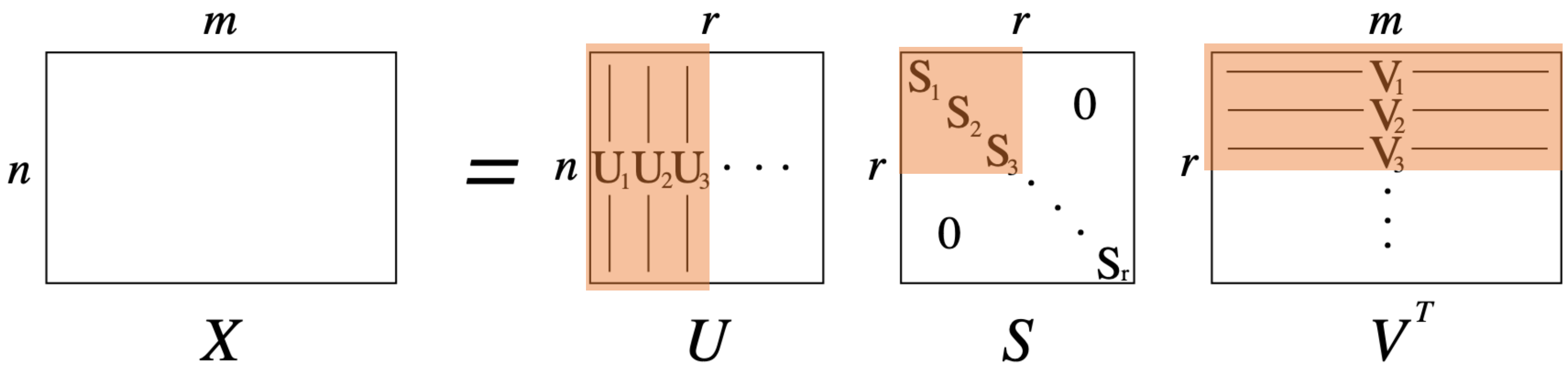
X_{ij}: The number of times each word appears inside a window of a particular size around the word of interest. We calculate this count for all the words in a corpus.

- Example corpus: (window size: 1)
  - “I like deep learning.”
  - “I like NLP.”
  - “I enjoy flying.”

$$X = \begin{array}{cc} & \begin{array}{cccccccc} I & like & enjoy & deep & learning & NLP & flying & . \end{array} \\ \begin{array}{c} I \\ like \\ enjoy \\ deep \\ learning \\ NLP \\ flying \\ . \end{array} & \begin{bmatrix} 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

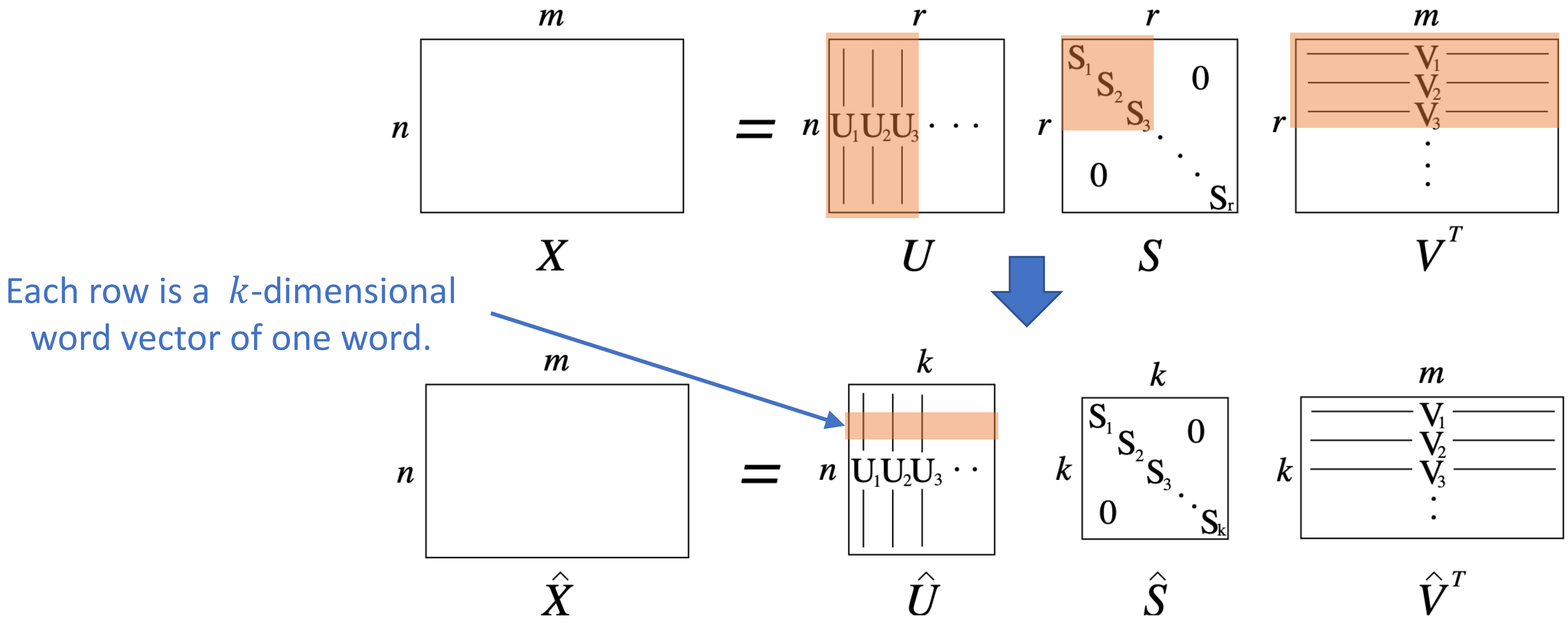
- Single value decomposition (SVD) of co-occurrence matrix X

$$X = USV^T$$



- Single value decomposition (SVD) of co-occurrence matrix X

$$X = USV^T$$



$\hat{X}$ is the best k-rank approximation to X

Problems with SVD-based methods

- The matrix is very high dimensional in general
- Quadratic cost to train (i.e. to perform SVD)
- Hard to incorporate new words or documents
- Requires the incorporation of some hacks on X to account for the drastic imbalance in word frequency

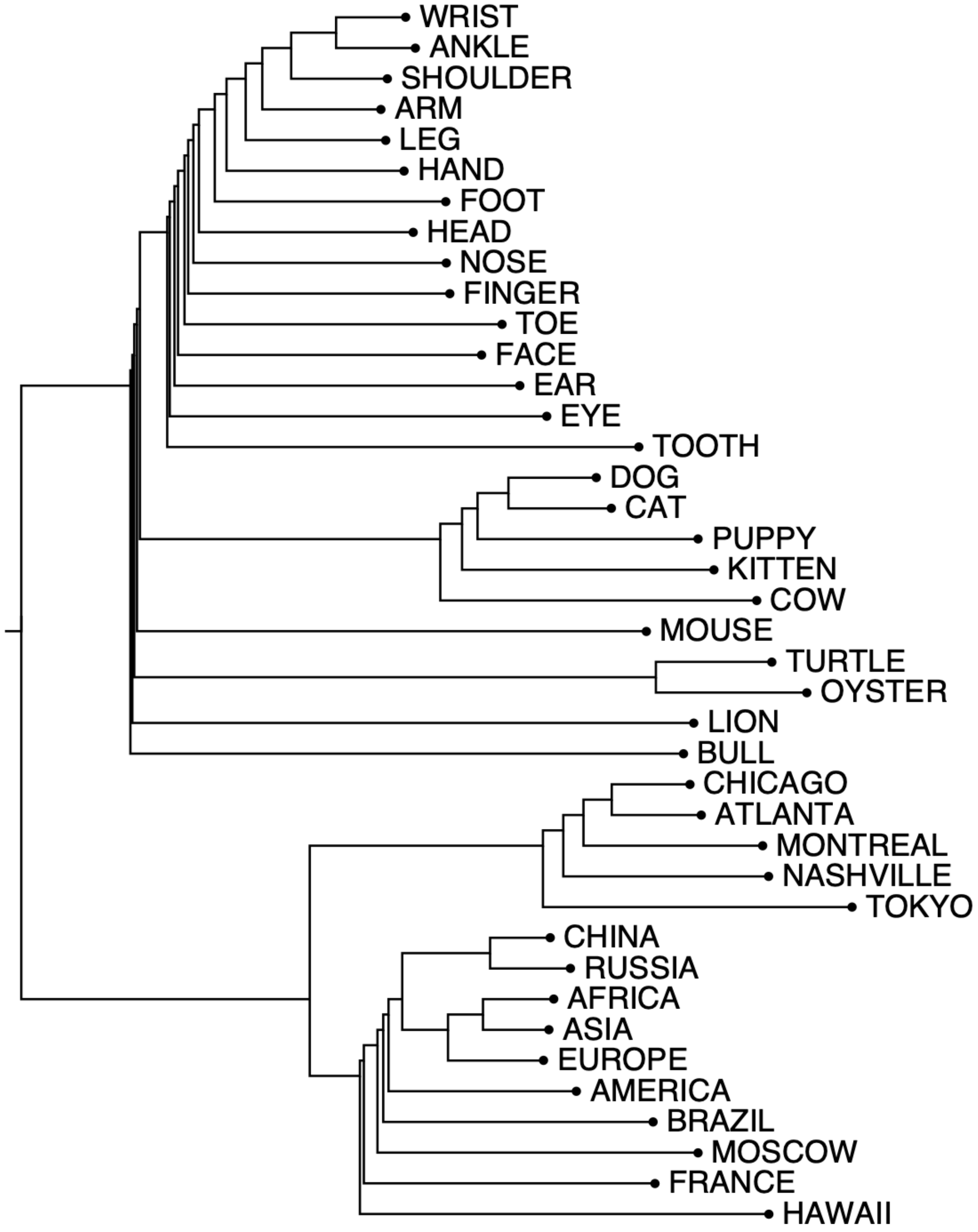
Hacks on X

- Problem: function words ("the", "he", "has”) are too frequent, leading the syntax has too much impact.
  - **Solution 1**: Cap the count min(X, m), m ~ 100.
  - **Solution 2**: Ignore function words.
- Apply **a ramp window** – i.e. weight the co-occurrence count based on distance between the words in the document.

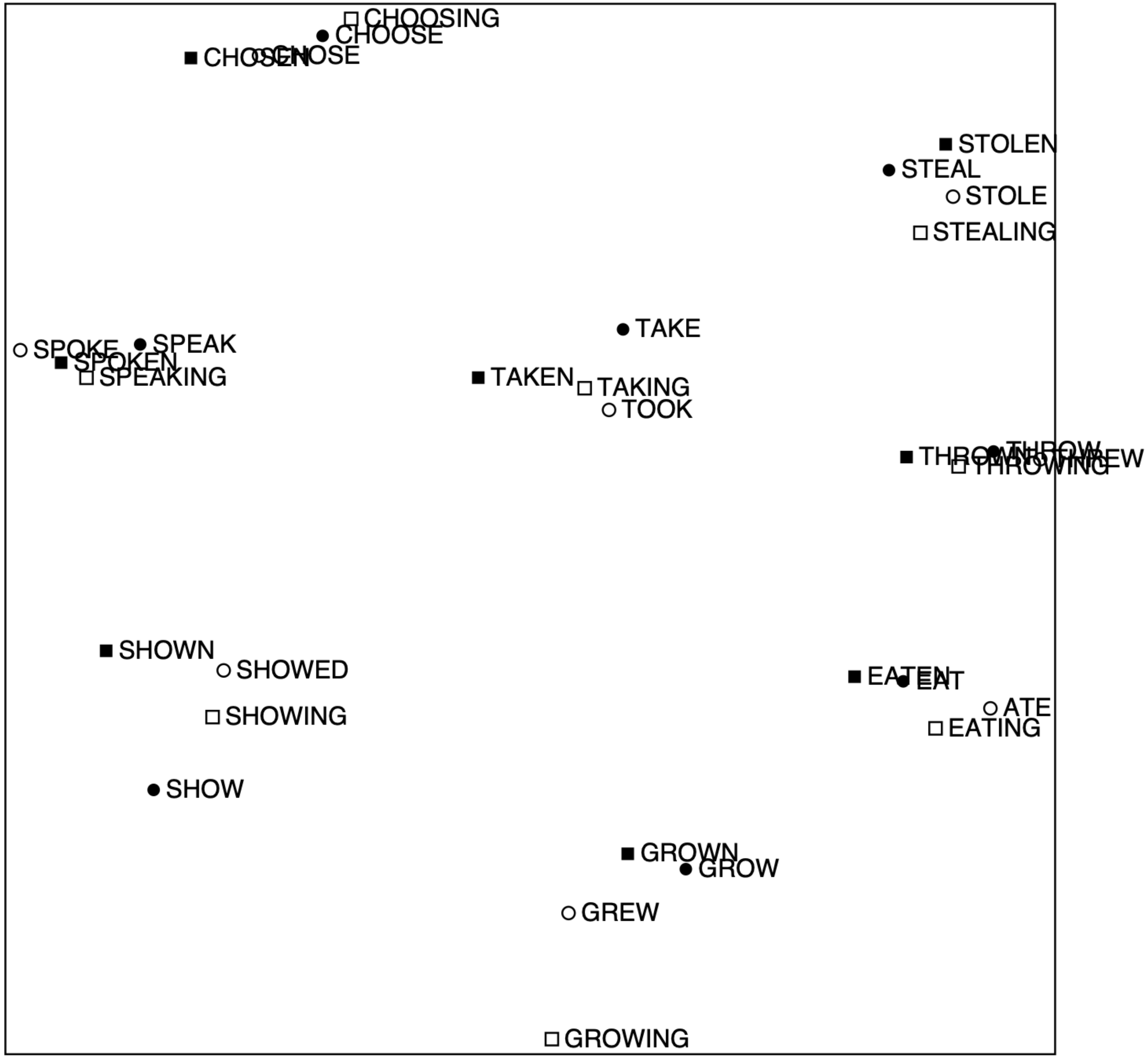
(1 2 3 4 0 4 3 2 1)

- Use **Pearson correlations** instead of counts, then set negative counts to 0

Rohde et al. 2005. An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence



Rohde et al. 2005. An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence



Rohde et al. 2005. An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence

Reference

- Lecture notes CS224D: Deep Learning Part 1: http://cs224d.stanford.edu/lecture_notes/notes1.pdf
- Lectures CS224n: NLP processing with Deep Learning. https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/syllabus.html

Topic 6: Word Embedding - Word2Vec (SkipGram & CBOW)

UNIVERSITY OF BATH

I like playing ___?___

football
basketball
tennis
table tennis
golf
chess
game
…

Words appear in similar contexts would have similar word embedding.

Word pairs for training

Example: “The quick brown fox jumps over the lazy dog.”

Window size: 2

Skip-gram: predict the context given the current word

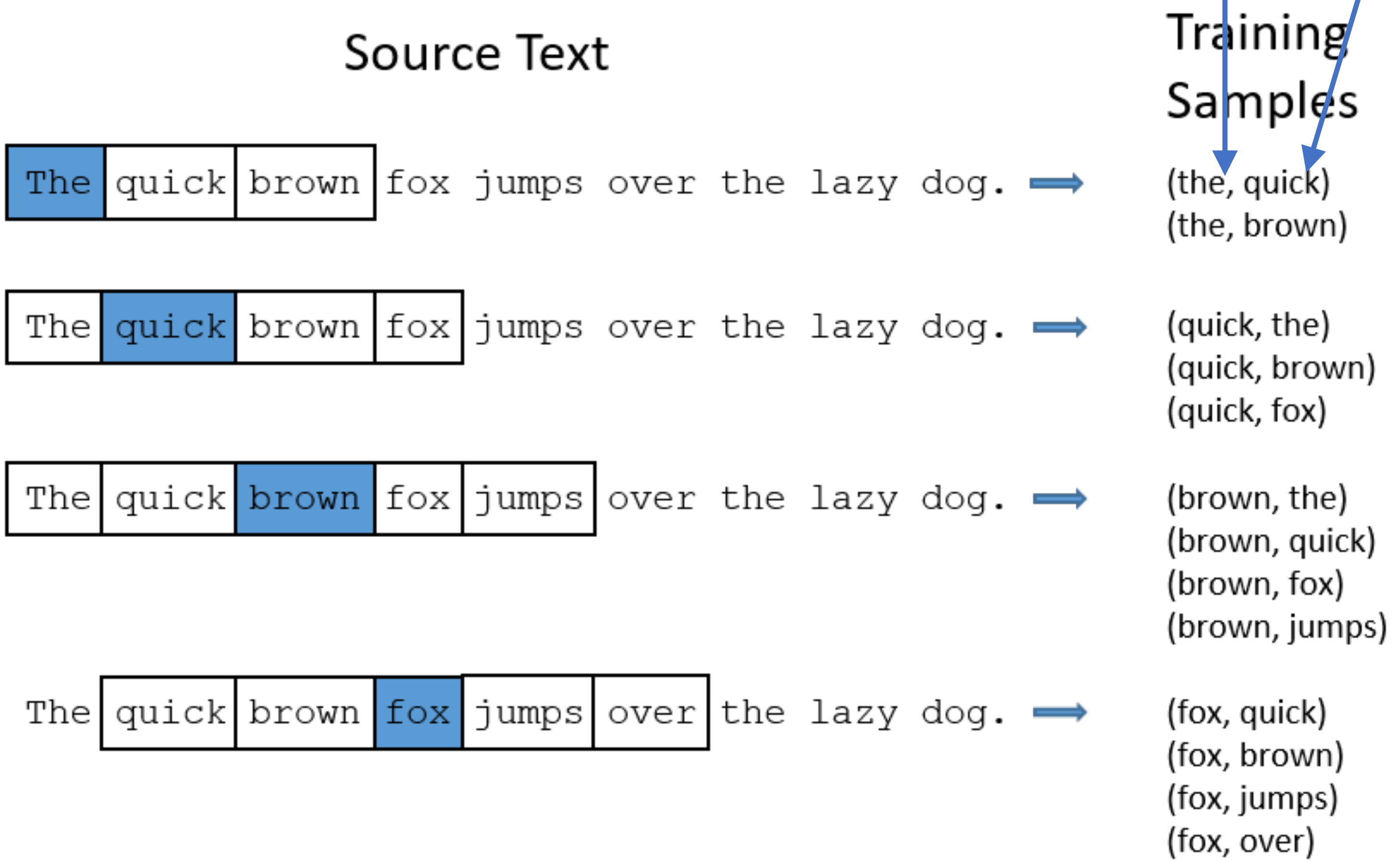
Img from: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Word pairs for training

Example: "The quick brown fox jumps over the lazy dog."

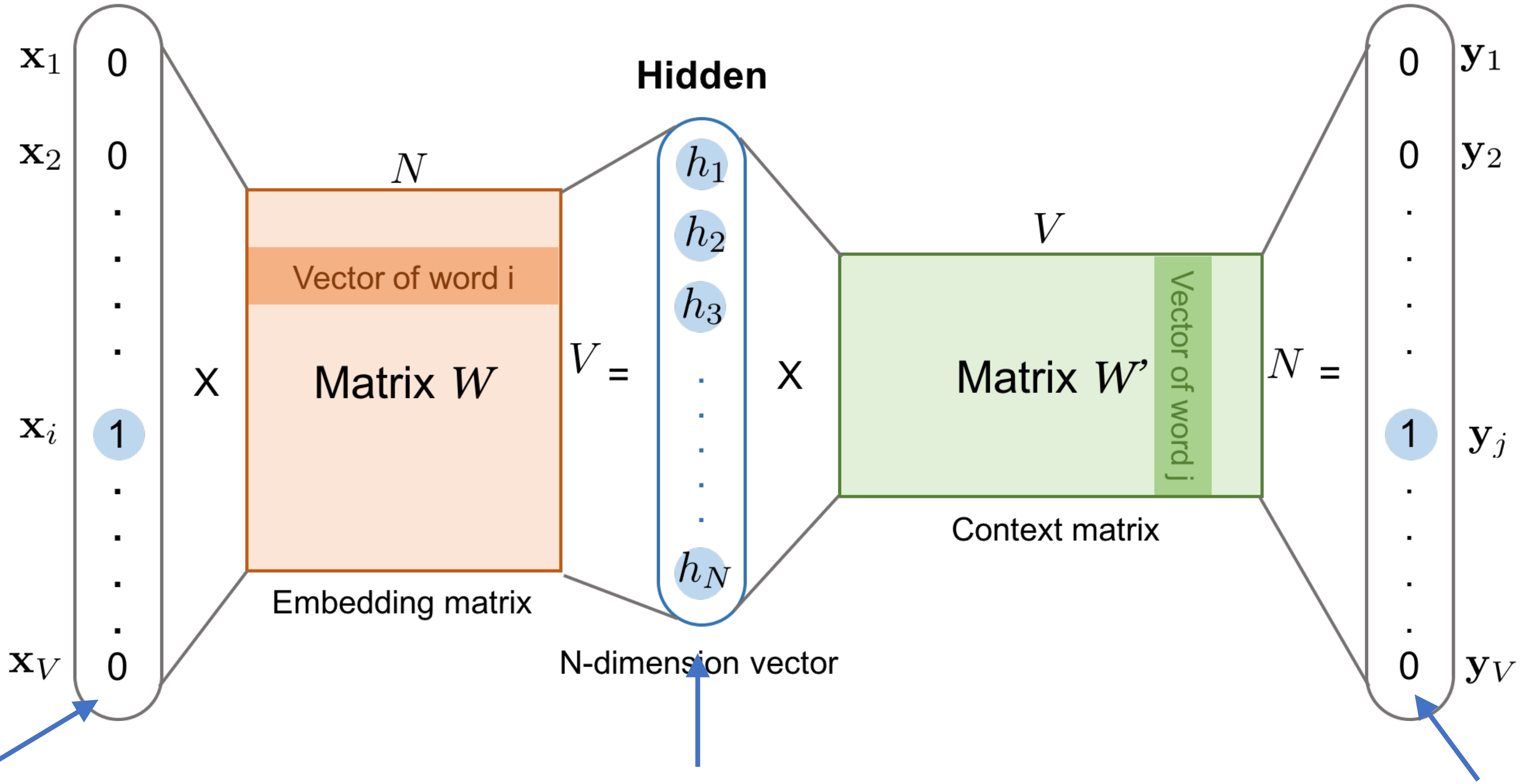
Window size: 2

Input x Output y



Img from: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Skip-gram model



Actual output:
One-hot vector of "quick"

One-hot vector of "fox"

Word embedding of "fox"

Probability for each word as the surrounding word of "fox"

For the training pair ("fox", "quick")

Hidden layer

- Fully connected layer with no activation function

Word embedding of the input word $h \in R^{N\times 1}$

$$h = W^T x$$

Embedding matrix $W \in R^{V\times N}$

$$\begin{bmatrix} 0 & 0 & 0 & \boxed{1} & 0 \end{bmatrix} \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ \boxed{10} & \boxed{12} & \boxed{19} \\ 11 & 18 & 25 \end{bmatrix} = \begin{bmatrix} 10 & 12 & 19 \end{bmatrix}$$

Img from: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

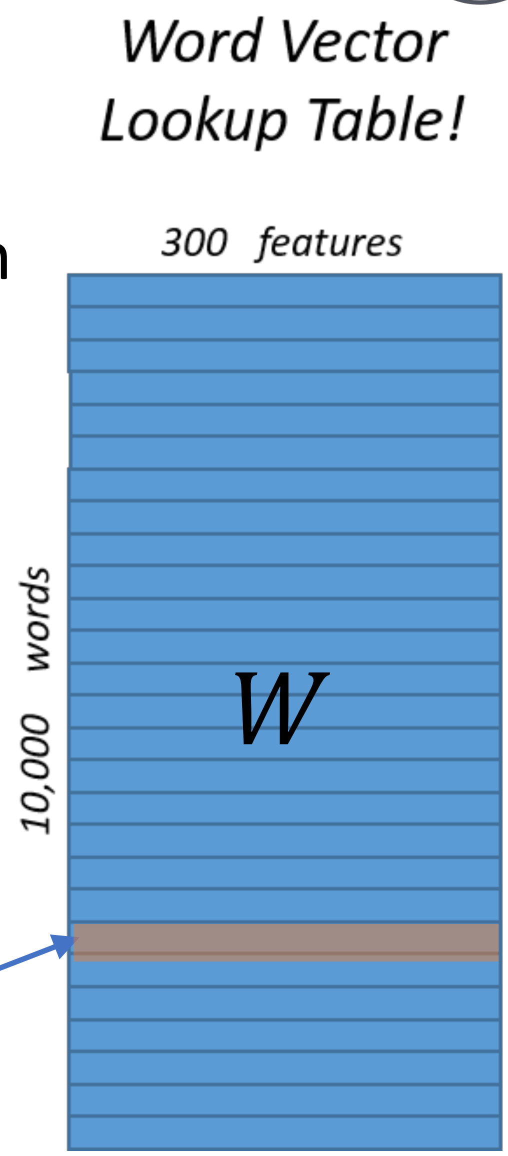
Hidden layer

- Fully connected layer with no activation function

Word embedding of the input word $h \in R^{N \times 1}$

$$h = W^T x$$

Embedding matrix $W \in R^{V \times N}$



Each row is a N-dimensional word vector of one word.

Img from: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

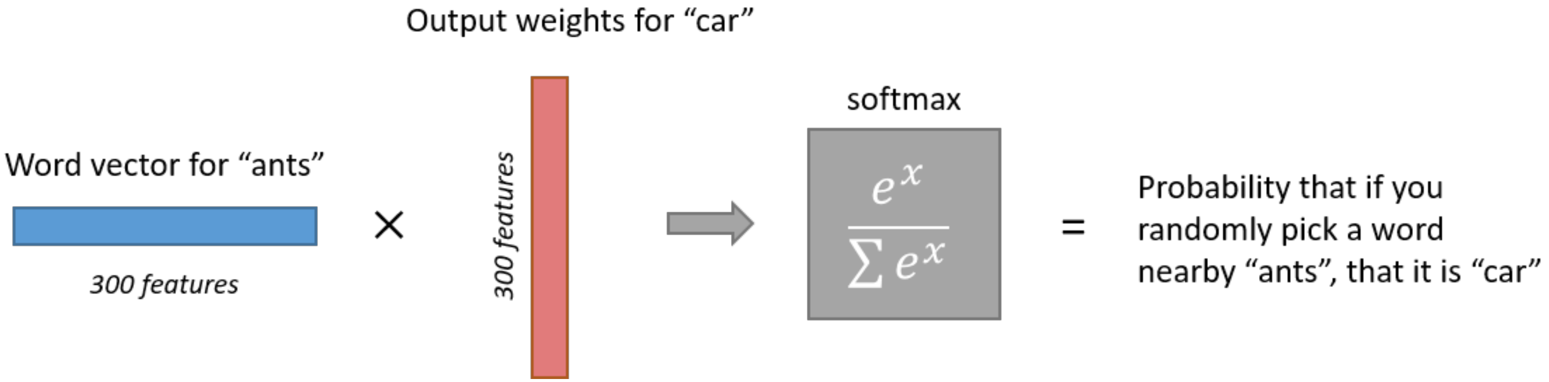
Output layer

- Fully connected layer with softmax activation function

$$\hat{y} = \text{softmax}(W'^{T}h)$$

Probability of each word in the vocabulary $\hat{y} \in R^{V\times 1}$

Context matrix $W' \in R^{N\times V}$



Img from: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Output layer

- Fully connected layer with softmax activation function

$$\hat{y} = \text{softmax}(W'^{T}h)$$

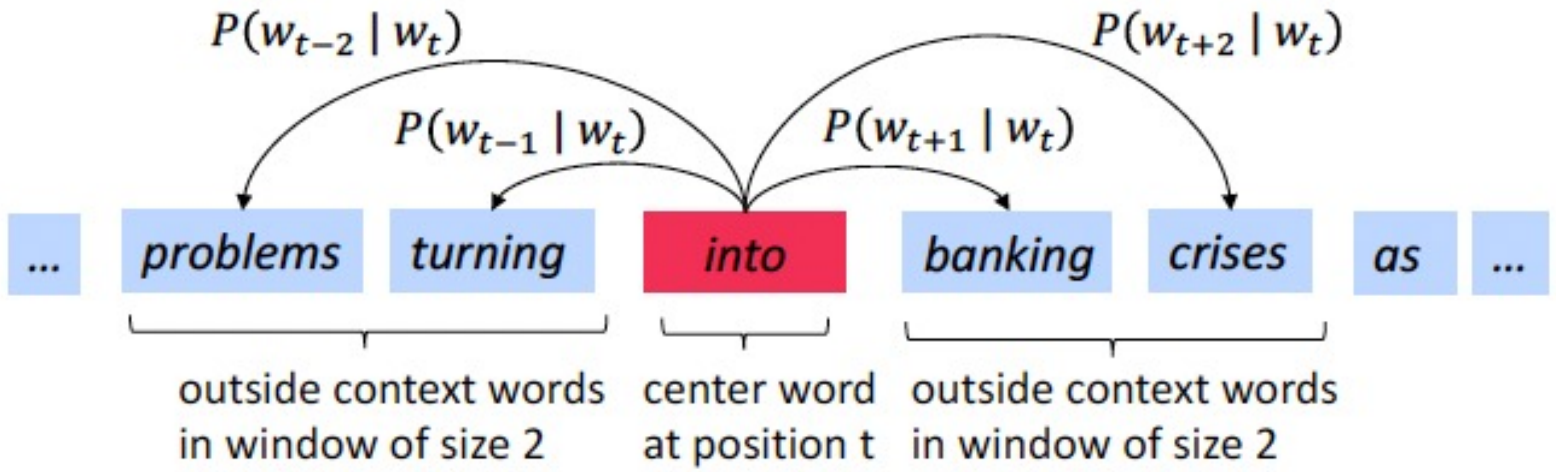
Probability of each word in the vocabulary $\hat{y} \in R^{V \times 1}$

Context matrix $W' \in R^{N \times V}$

Let's interpret this process in a statistical aspect ...

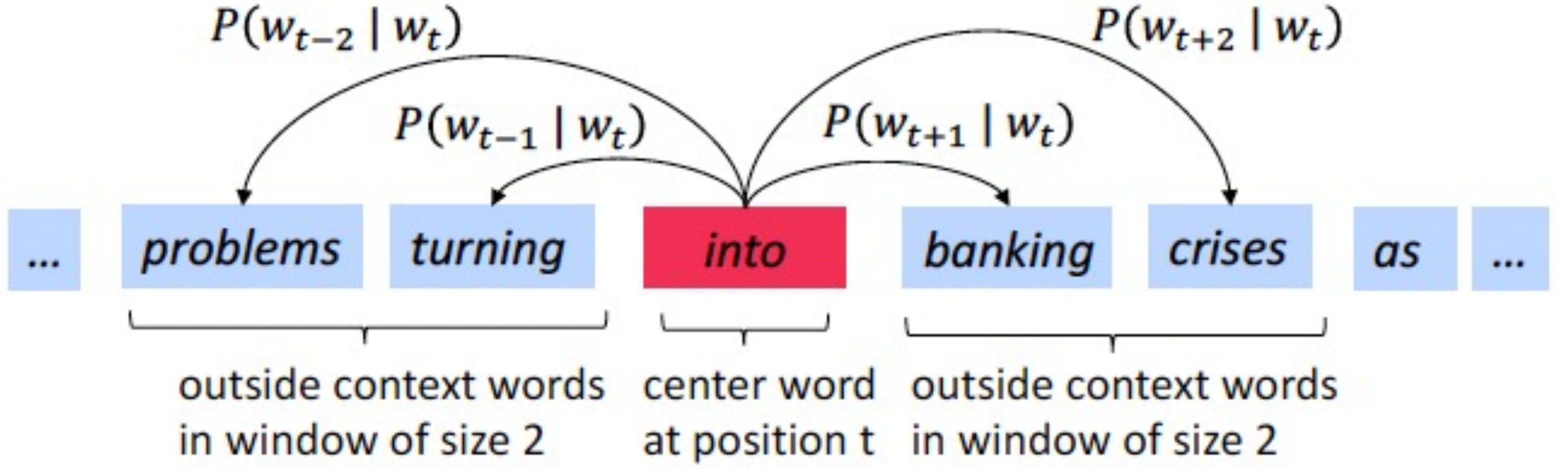
Img from: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

- Try to maximize the probability $P(w_{t+j}|w_t)$



- For each position t, maximize the likelihood of the context words within a window of size m, given the centre word w_t.

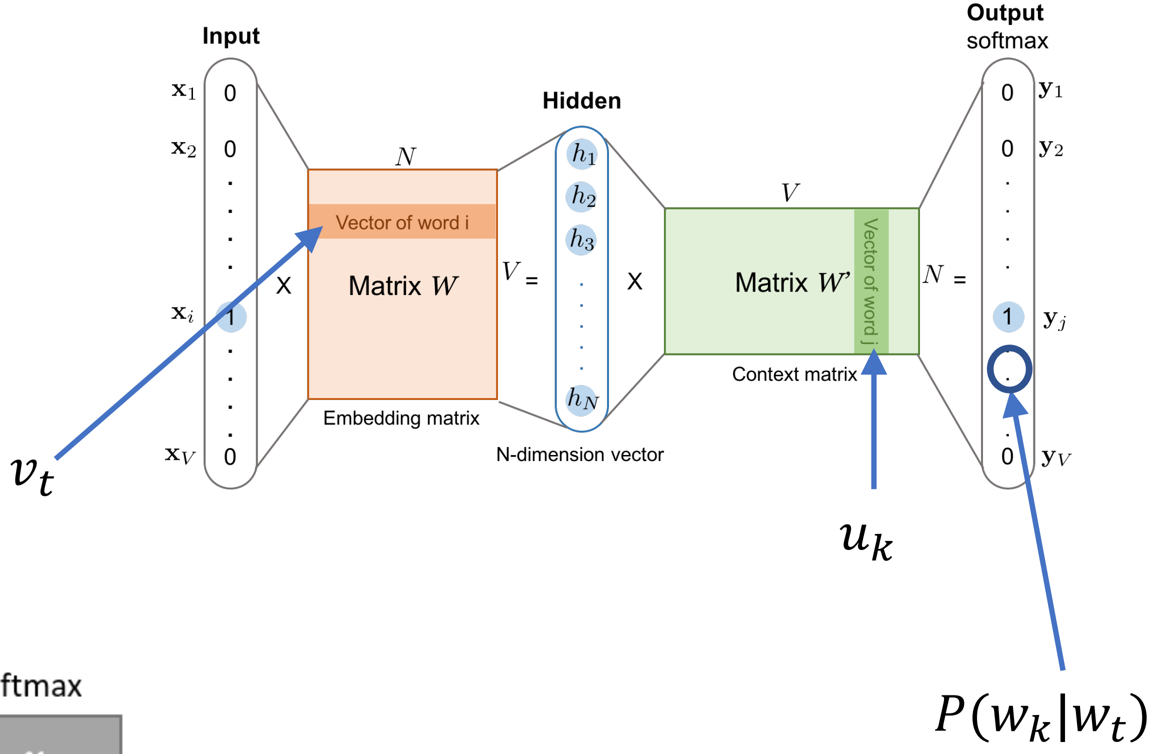
$$P(w_{t-m}, \dots, w_{t-1}, w_{t+1,} \dots, w_{t+m} | w_t) = \prod_{-m \leq j \leq m, j \neq 0} P(w_{t+j} | w_t)$$



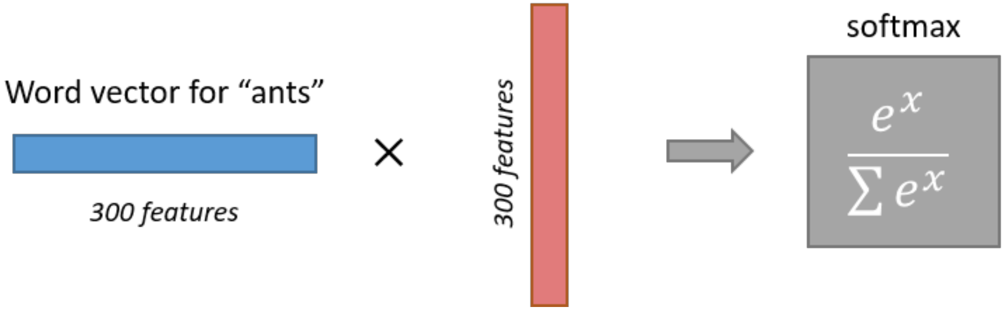
How to calculate $P(w_{t+j}|w_t)$?

- v_t: the word vector for the input word w_t
- u_k: the word vector for the output word w_k

$$P(w_{t+j}|w_t) = \frac{\exp(u_{t+j}^T v_t)}{\sum_{k=1}^{V} \exp(u_k^T v_t)}$$



Output weights for “car”



= Probability that if you randomly pick a word nearby “ants”, that it is “car”

v_t u_k $P(w_k|w_t)$

Skip-gram loss function

- Maximize the likelihood of predicting the context words

$$P\left(w_{t-m}, \dots, w_{t-1}, w_{t+1,} \dots, w_{t+m}\right) = \prod_{-m \le j \le m, j \ne 0} \frac{\exp(u_{t+j}^T v_t)}{\sum_{k=1}^{V} \exp(u_k^T v_t)}$$

- Minimize the negative log likelihood:

$$J = -\log P\left(w_{t-m}, \dots, w_{t-1}, w_{t+1,} \dots, w_{t+m}\right)$$

$$= -\log \prod_{-m \le j \le m, j \ne 0} \frac{\exp(u_{t+j}^T v_t)}{\sum_{k=1}^{V} \exp(u_k^T v_t)}$$

$$= -\sum_{-m \le j \le m, j \ne 0} u_{t+j}^T v_t + 2m \log \sum_{k=1}^{V} \exp(u_k^T v_t)$$

Skip-gram loss function

UNIVERSITY OF BATH

- All the parameters $(v_t, u_t, t = 1, \dots, V)$ will be optimized via Stochastic Gradient Descent (SGD).

$$J = -\sum_{-m \le j \le m, j \ne 0} u_{t+j}^T v_t + 2m \log \sum_{k=1}^{V} \exp(u_k^T v_t)$$

However, since V is a huge number, it is computationally intensive for every training step.

Topic 7: Word2Vec with Negative Sampling

Skip-Grams with Negative Sampling (SGNS)

Marco saw a furry little wampimuk hiding in the tree.

"word2vec Explained…"
Goldberg & Levy, arXiv 2014

Skip-Grams with Negative Sampling (SGNS)

Marco saw a furry little wampimuk hiding in the tree.

“word2vec Explained…”
Goldberg & Levy, arXiv 2014

Skip-Grams with Negative Sampling (SGNS)

Marco saw a furry little wampimuk hiding in the tree.

$$\left.\begin{array}{ll} \underline{\textbf{words}} & \underline{\textbf{contexts}} \\ \text{wampimuk} & \text{furry} \\ \text{wampimuk} & \text{little} \\ \text{wampimuk} & \text{hiding} \\ \text{wampimuk} & \text{in} \end{array}\right\} D \text{ (data)}$$

... ...

"word2vec Explained…"
Goldberg & Levy, arXiv 2014

Skip-Grams with Negative Sampling (SGNS)

- **Maximize:** $\sigma(\vec{w} \cdot \vec{c})$
  - c was **observed** with w

words	**contexts**
wampimuk	furry
wampimuk	little
wampimuk	hiding
wampimuk	in

"word2vec Explained…"
Goldberg & Levy, arXiv 2014

Skip-Grams with Negative Sampling (SGNS)

- **Maximize:** $\sigma(\vec{w} \cdot \vec{c})$
  - c was **observed** with w

words	**contexts**
wampimuk	furry
wampimuk	little
wampimuk	hiding
wampimuk	in

- **Minimize:** $\sigma(\vec{w} \cdot \vec{c}')$
  - c' was **hallucinated** with w

words	**contexts**
wampimuk	Australia
wampimuk	cyber
wampimuk	the
wampimuk	1985

"word2vec Explained…"
Goldberg & Levy, arXiv 2014

Negative Sampling

- **Idea**: randomly select just a small number of "negative" words (say 5) to update the weights for.

$300\times 10{,}000$ → 300×5

Context matrix $W' \in R^{N\times V}$

Context matrix with NS: $W' \in R^{N\times M}$

For every training step, only modify a small percentage of weights, rather than the big weight matrix for all training samples.

Negative Sampling

Basic Word2Vec

A multi-class classification problem

Predict probability of each word being a nearby word

$$P(\text{quick}|\text{fox}) = \text{softmax}\left(u_{\text{quick}}^{T} v_{\text{fox}}\right)$$

Word2Vec with Negative Sampling

A binary classification problem

Predict probability of two words are neighbors

$$P(D = 1 \,|\, \text{fox, quick}) = \sigma\left(u_{\text{quick}}^{T} v_{\text{fox}}\right)$$

(fox, quick)	1
(fox, jumps)	1
(fox, brown)	1
(fox, chair)	0
(fox, artificial)	0
(fox, kiwi)	0
(fox, sign)	0

$$J_{NS} = -\log\left(\sigma\left(u_o^T v_c\right)\right) - \sum_{\substack{k=1 \\ u_k \sim P(w)}}^{K} \log\left(\sigma\left(-u_k^T v_c\right)\right)$$

$\sigma(\cdot)$Is the sigmoid function

u_k is a negative sample of word v_c

Small K (around 2 to 5) for large dataset. Large K (around 5 to 20) for smaller dataset.

Selecting Negative Samples

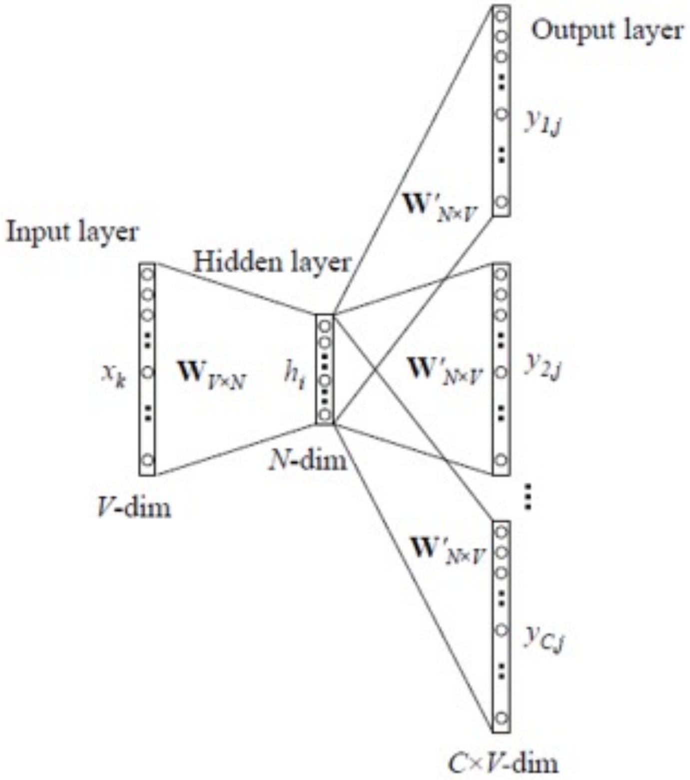
- u_k is sampled from P(w) = U(w)^3/4, where U(w) is a unigram distribution

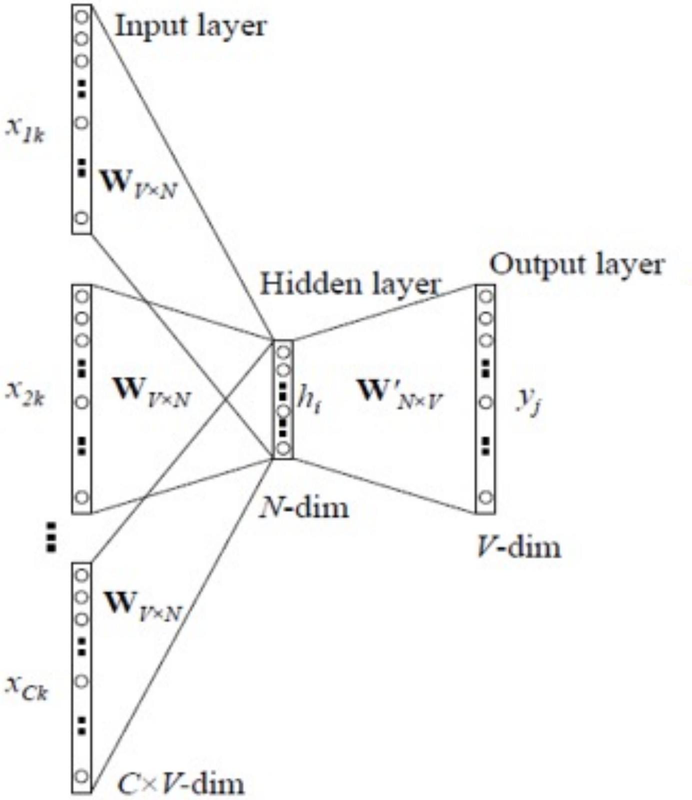
$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n} \left(f(w_j)^{3/4}\right)}$$

- The ¾ power makes less frequent word be sampled more often.

"is": $0.9^{3/4} = 0.92$
"constitution": $0.09^{3/4} = 0.16$
"bombastic": $0.01^{3/4} = 0.032$

Skip-gram vs CBOW (Continuous BoW)





Skip-gram: predict the context given the current word

I like playing football

CBOW: predict the the current word using its context

I like playing football

Reference

- Lecture notes CS224D: Deep Learning Part: http://cs224d.stanford.edu/lecture_notes/notes1.pdf
- Ria Kushrestha's blog: NLP 102: Negative Sampling and GloVe https://towardsdatascience.com/nlp-101-negative-sampling-and-glove-936c88f3bc68
- Chris McCormick's blog: Word2Vec Tutorial – The Skip-Gram Model. http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/
- Chris McCormick's blog: Word2Vec Tutorial Part 2 – Negative sampling. http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/

Topic 8: Word Embedding - GloVe

GloVe

- Proposed by Pennington et al. 2014.
- Unlike Word2vec, GloVe does not reply just on local statistics (local context information of words), but incorporates global statistics (word co-occurrence matrix) to obtain word vectors.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. [GloVe: Global Vectors for Word Representation].

GloVe

Counts the co-occurrence between words w_k, w_i

$$P(w_k|w_i) = \frac{C(w_k, w_i)}{C(w_i)}$$

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

"solid" is more related to "ice" than "steam"

$$F(w_i, w_j, w_k) = \frac{P(w_k|w_i)}{P(w_k|w_j)}$$

"Water" is equally (ir)relavant to "ice" and "steam"

Intuition: **co-occurence probabilities ratios** gathers more information than the raw probabilities and better capture relevant information about words' relationship

What is the function $F(\cdot)$?

$$F(w_i, w_j, w_k) = \frac{P(w_k|w_i)}{P(w_k|w_j)}$$

Since the goal is to learn meaningful word vectors, $F(\cdot)$ is designed to be a function of the linear difference between two words w_i and w_j

$$F((w_i - w_j)^T w_k) = \frac{P(w_k|w_i)}{P(w_k|w_j)}$$

The final solution is to $F(\cdot)$ as an **exponential** function.

$$F\left((w_i - w_j)^T w_k\right) = \exp\left((w_i - w_j)^T w_k\right) = \frac{\exp(w_i{}^T w_k)}{\exp\left(w_j{}^T w_k\right)} = \frac{P(w_k|w_i)}{P(w_k|w_j)}$$

Loss function of GloVe

Replace it with a bias term b_i

$$w_i{}^T w_k = log \frac{C(w_k,w_i)}{C(w_i)} = \log C(w_k,w_i) - \log C(w_i)$$

$$\log C(w_k,w_i) = w_i{}^T w_k + b_i + b_k$$

To keep the symmetric form, we also add in a bias term b_k

The loss function for the GloVe model is designed to preserve the above formula by minimizing the sum of the squared errors

$$\mathcal{L} = \sum_{i=1,j=1}^{V} (w_i^T w_j + b_i + b_j - \log C(w_j,w_i))^2$$

Add a weighting function $f()$ that is used to downweight the importance of very frequent co-occurrences

$$\mathcal{L} = \sum_{i=1,j=1}^{V} f(C(w_j,w_i))(w_i^T w_j + b_i + b_j - \log C(w_j,w_i))^2$$

Loss function of GloVe

To penalize the difference between the dot product of two word vectors and the logarithm of the co-occurrence count, with the bias terms added.

$$\mathcal{L} = \sum_{i=1,j=1}^{V} f(C(w_j, w_i))(w_i^T w_j + b_i + b_j - \log C(w_j, w_i))^2$$

w_i, w_j are the word vectors for words i and j

b_i, b_j are bias terms

$$f(c) = \begin{cases} \left(\frac{c}{c_{\max}}\right)^{\alpha} & \text{if } c < c_{\max}, c_{\max} \text{ is adjustable.} \\ 1 & \text{if otherwise} \end{cases}$$

Nearest words to frog:

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

litoria

leptodactylidae

rana

eleutherodactylus

Advantages of GloVe

- Computationally efficient
- Scales well to large datasets
- Produces embedding that capture both syntactic and semantic relationships between words.

GloVe vs. Word2Vec

- Both are popular algorithms for generating word embeddings.
- Both are unsupervised learning algorithms
- Both are able to capture semantic relationships between words.

Word2Vec

Use a neural network to learn embedding

Focus more on local context

Is able to handle larger corpora of text

GloVe

Based on co-occurrence matrix

Capture global relationships between words.

Is generally faster than Word2Vec

Reference

- Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global Vectors for Word Representation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP).
- https://nlp.stanford.edu/projects/glove/
- Lectures of CS224n: NLP with Deep learning. https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/syllabus.html
- Matyas Amrouche's blog: Word embedding (Part II). https://towardsdatascience.com/word-embedding-part-ii-intuition-and-some-maths-to-understand-end-to-end-glove-model-9b08e6bf5c06
- Lilian Weng's blog: Learning Word Embedding. https://lilianweng.github.io/posts/2017-10-15-word-embedding/