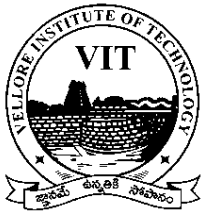


VIT-AP UNIVERSITY

Natural Language Processing

(Course Code: CSE 3015)

Module-1:Lecture-4: Word Embedding

Gundimeda Venugopal, Professor of Practice, SCOPE

Word Embeddings

- Word Embedding are vector representations of a particular word.
- Word Embeddings create a vector representation for words that capture their *meanings*, *semantic relationships* and the different types of contexts they are used in.
- Two types of Word Embeddings
  - Frequency based embedding
  - Prediction based embedding
- Frequency based Embedding
  - Count vector
  - TF – IDF Vectorization
  - Co-occurrence Matrix
- Prediction based Embedding
  - Continuous Bag of Words (CBOW)
  - Skip Gram

Representing a Word: One Hot Encoding

Vocabulary		**one-hot** encodings
dog	1	[1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
cat	2	[0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
person	3	[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
holding	4	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
tree	5	[0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
computer	6	[0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
using	7	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0]

Sparse Vectors

Representing **Phrases**: Bag-of-Words

		bag-of-words representation
person holding dog	{3, 4, 1}	[1, 0, 1, 1, 0, 0, 0]
person holding cat	{3, 4, 2}	[0, 1, 1, 1, 0, 0, 0]
person using computer	{3, 7, 6}	[0, 0, 1, 0, 0, 1, 1]

dog cat person holding tree computer using

Vocabulary

dog	1
cat	2
person	3
holding	4
tree	5
computer	6
using	7

One Hot Encoding Example

corpus
D1:"I love NLP"
D2:"NLP is amazing"
D3:"I enjoy learning NLP"

vocabulary :
[I
love
NLP
is
amazing
enjoy
learning]

Assign Indices
I -> 0
love-> 1
NLP -> 2
is -> 3
amazing -> 4
enjoy -> 5
learning -> 6

Convert Words to One-Hot Vectors

I -> [1, 0, 0, 0, 0, 0, 0]
love-> [0, 1, 0, 0, 0, 0, 0]
NLP-> [0, 0, 1, 0, 0, 0, 0]
is -> [0, 0, 0, 1, 0, 0, 0]
amazing-> [0, 0, 0, 0, 1, 0, 0]
enjoy-> [0, 0, 0, 0, 0, 1, 0]
learning -> [0, 0, 0, 0, 0, 0, 1]

D1: [[1, 0, 0, 0, 0, 0, 0][0, 1, 0, 0, 0, 0, 0][0, 0, 1, 0, 0, 0, 0]]
D2:[[0, 0, 1, 0, 0, 0, 0][0, 0, 0, 1, 0, 0, 0][0, 0, 0, 0, 1, 0, 0]]
D3: [[1, 0, 0, 0, 0, 0, 0][0, 0, 0, 0, 0, 1, 0][0, 0, 0, 0, 0, 0, 1] [0, 0, 1, 0, 0, 0, 0]]

One Hot Encoding Code Example

```
import nltk
from nltk.tokenize import word_tokenize

corpus = "I Love NLP"
# Tokenize the text into words
tokens = word_tokenize(corpus.lower())  # Tokenizing and converting to lowercase
# Remove punctuation from the tokens
tokens = [word for word in tokens if word.isalpha()]
# Create a vocabulary (list of unique words)
vocabulary = list(set(tokens))
print("Vocabulary:", vocabulary)

# Manually assigning index to each word in the vocabulary
word_to_index = {word: idx for idx, word in enumerate(vocabulary)}

# Create a one-hot encoding for each word in the tokenized text
one_hot_encoded = []

for word in vocabulary:
    encoding = [0] * len(vocabulary)  # Initialize a zero vector of length of vocabulary
    encoding[word_to_index[word]] = 1  # Set 1 at the index of the word
    one_hot_encoded.append(encoding)

# Print one-hot encoded vectors for the words
for word, encoding in zip(vocabulary, one_hot_encoded):
    print(f"Word: '{word}' -> One-hot encoding: {encoding}")
```

```
Vocabulary: ['i', 'nlp', 'love']
Word: 'i' -> One-hot encoding: [1, 0, 0]
Word: 'nlp' -> One-hot encoding: [0, 1, 0]
Word: 'love' -> One-hot encoding: [0, 0, 1]
```

Disadvantages

❖High Dimensionality:

One-hot encoding creates a large vector for each word, If the vocabulary is large (as in real-world text data), this leads to very high-dimensional vectors, which can be computationally expensive and inefficient.

❖Sparse Representation:

Most of the elements in a one-hot encoded vector are zero, leading to sparse vectors. Sparse vectors are inefficient in terms of memory and computational resources, especially when dealing with large datasets.

❖Lack of Semantic Information:

One-hot encoding does not capture any semantic relationships between words. For example, "cat" and "dog" would be represented as distinct, orthogonal vectors, even though they are semantically related. This limitation makes it difficult for the model to understand context or word similarity.

❖No Contextual Information:

Each word is treated as an independent token without considering its context. For instance, the word "bank" could refer to a financial institution or the side of a river, but one-hot encoding doesn’t provide any indication of the word's contextual meaning.

Bag of Words (BoW)

- The Bag of Words (BoW) model is a representation technique used in NLP to convert text documents into numerical vectors.
- we can represent a sentence as a bag of words vector (a string of numbers).

How does Bag of Words work?

Tokenization: Break the text into individual words or tokens.

Vocabulary Creation: Create a vocabulary (a list of unique words) from all the text.

Vector Representation: For each document, represent it as a vector.

The vector's size will be equal to the number of unique words in the vocabulary, and each element in the vector represents the frequency of the corresponding word in the document.

Bag of Words Example-1:

Corpus:
- "I love programming."
- "I love NLP."
- "Programming is fun."

Tokenize the sentences

Sentence 1: ['I', 'love', 'programming’]
Sentence 2: ['I', 'love', 'NLP’]
Sentence 3: ['Programming', 'is', 'fun']

Vocabulary:

['I', 'love', 'programming', 'NLP', 'is', 'fun']

Represent each sentence as a vector

	I	love	programming	NLP	is	fun
Sentence 1	1	1	1	0	0	0
Sentence 2	1	1	0	1	0	0
Sentence 3	0	0	1	0	1	1

Sentence 1: "I love programming."
- Vector: [1, 1, 1, 0, 0, 0]

Sentence 2: "I love NLP."
- Vector: [1, 1, 0, 1, 0, 0]

Sentence 3: "Programming is fun."
- Vector: [0, 0, 1, 0, 1, 1]

Bag of Words Example-2:

Corpus:
“The cat sat on the mat.”
“The dog played in the yard.”

Vocabulary:
[“the”, “cat”, “sat”, “on”, “mat”, “dog”, “played”, “in”, “yard”]

Represent each sentence as a vector

	the	cat	sat	on	mat	dog	played	in	yard
D1	2	1	1	1	1	0	0	0	0
D2	2	0	0	0	0	1	1	1	1

Representing these sentences using BoW:
- Sentence 1: [2, 1, 1, 1, 1, 0, 0, 0, 0]
- Sentence 2: [2, 0, 0, 0, 0, 1, 1, 1, 1,]

Bag of Words: Limitations

- **High-dimensionality**
  As the number of words increases, the feature space grows, making models computationally expensive.
- **Sparsity**
  The vector representation can be sparse, especially with a large vocabulary.
- **Ignoring Word Order**
  BoW does not consider the word order in a sentence, which may lead to losing important context.
- **Ignores Context and Semantic Relationships**
  It does not capture relationships between words or their meanings, leading to limitations in understanding context.

Bag of Words Code Example

```
from sklearn.feature_extraction.text import CountVectorizer

documents = [
    "This movie is very scary and long",
    "This movie is not scary and is slow",
    "This movie is spooky and good"
]

# Initialize the CountVectorizer
vectorizer = CountVectorizer()

# Fit and transform the documents into the Bag of Words representation
bow_matrix = vectorizer.fit_transform(documents)

# Get the feature names (unique words in the corpus)
feature_names = vectorizer.get_feature_names_out()

# Convert the BoW matrix to an array for better visualization
bow_array = bow_matrix.toarray()

# Display the BoW matrix
print("Feature Names:", feature_names)
print("\nBag of Words Matrix:")
print(bow_array)
```

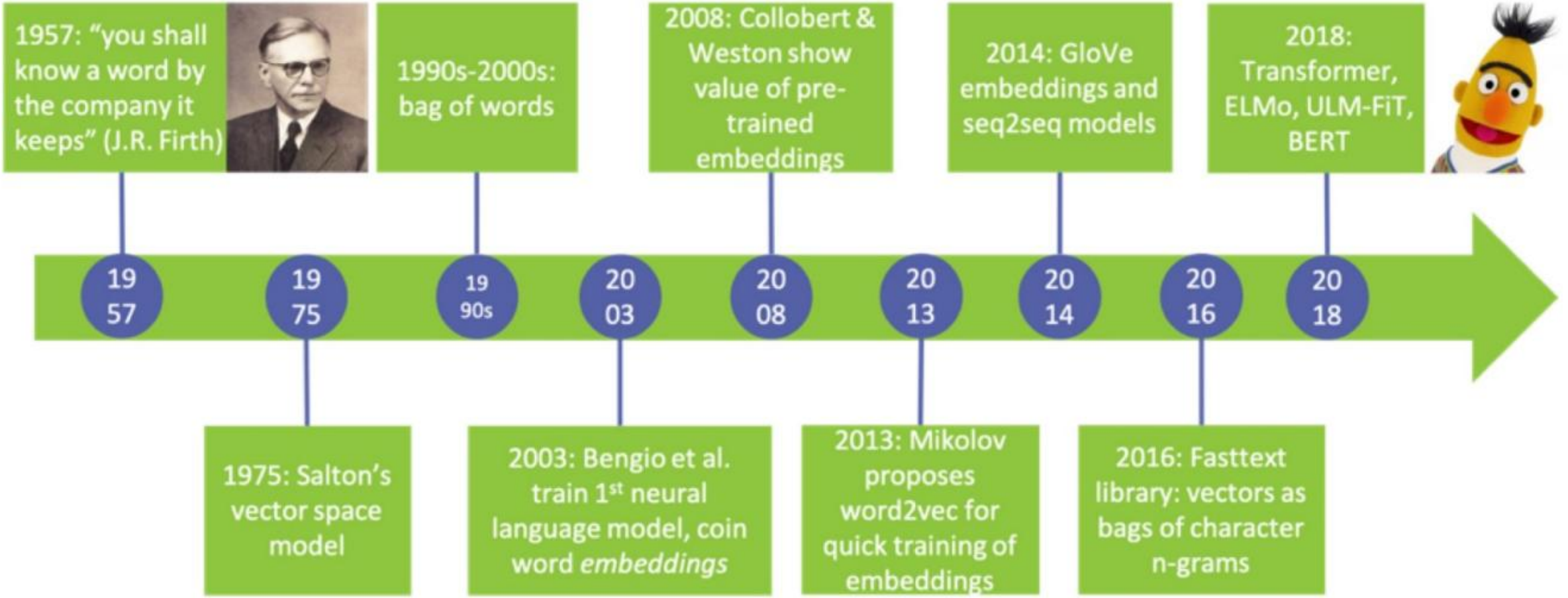
```
Feature Names: ['and' 'good' 'is' 'long' 'movie' 'not' 'scary' 'slow' 'spooky' 'this'
 'very']

Bag of Words Matrix:
[[1 0 1 1 1 0 1 0 0 1 1]
 [1 0 2 0 1 1 1 1 0 1 0]
 [1 1 1 0 1 0 0 0 1 1 0]]
```

Sparse versus dense vectors

Why dense vectors?

- Short vectors may be easier to use as **features** in machine learning (less weights to tune)
- Dense vectors may **generalize** better than storing explicit counts
- They may do better at capturing synonymy:
  - *car* and *automobile* are synonyms; but are distinct dimensions
    - a word with *car* as a neighbor and a word with *automobile* as a neighbor should be similar, but aren't
- **In practice, they work better**

Word Embeddings: Timeline



Representing words by their context

Distributional hypothesis: words that occur in similar contexts tend to have similar meanings

J.R.Firth 1957

- "You shall know a word by the company it keeps"
- One of the most successful ideas of modern statistical NLP!

…government debt problems turning into	***banking***	*crises as happened in 2009…*
…saying that Europe needs unified	***banking***	*regulation to replace the hodgepodge…*
…India has just given its	***banking***	*system a shot in the arm…*

These context words will represent *banking*.

What is the meaning of “bardiwac”?

— He handed her glass of **bardiwac**.

— Beef dishes are made to complement the **bardiwacs**.

— Nigel staggered to his feet, face flushed from too much **bardiwac**.

— Malbec, one of the lesser-known **bardiwac** grapes, responds well to Australia’s sunshine.

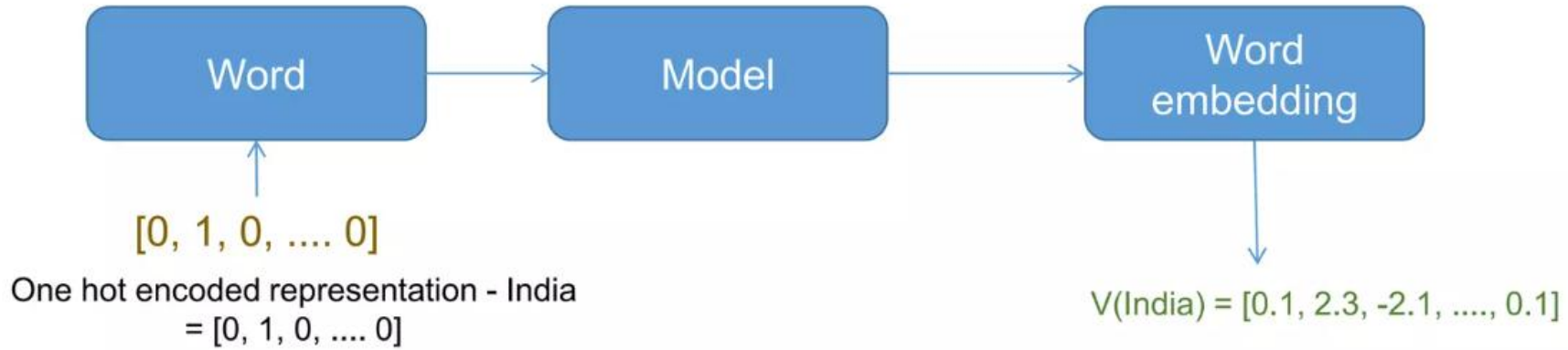
— I dined off bread and cheese and this excellent **bardiwac**.

—The drinks were delicious: blood-red **bardiwac** as well as light, sweet Rhenish.

bardic is an alcoholic beverage made from grapes

Prediction based word Embeddings

- It is a method to learn dense representation of word from a very high dimensional representation.



- It is a modular representation, where a sparse vector is fed to generate a dense representation

What does context mean?

- Context is co-occurrence of the words. It is a sliding window around the word under the consideration.

India	is	now	inching	towards	a	self	reliant	state
India	is	now	inching	towards	a	self	reliant	state
India	is	now	inching	towards	a	self	reliant	state
India	is	now	inching	towards	a	self	reliant	state
India	is	now	inching	towards	a	self	reliant	state
India	is	now	inching	towards	a	self	reliant	state
India	is	now	inching	towards	a	self	reliant	state
India	is	now	inching	towards	a	self	reliant	state

Window size = 2, Yellow patches are words are in consideration, orange box are the context window

Distributional Hypothesis

"tejuino"

C1: A bottle of ____ is on the table.

C2: Everybody likes ____.

C3: Don't have ____ before you drive.

C4: We make ____ out of corn.

Distributional Hypothesis

C1: A bottle of ____ is on the table.

C2: Everybody likes ____.

C3: Don't have ____ before you drive.

C4: We make ____ out of corn.

	C1	C2	C3	C4
tejuino	1	1	1	1
loud	0	0	0	0
motor-oil	1	0	0	0
tortillas	0	1	0	1
choices	0	1	0	0
wine	1	1	1	0

"words that occur in similar contexts tend to have similar meanings"

Geometric Interpretation: Co-occurrence as feature

Distributional Hypothesis

— At least certain aspects of the meaning of lexical expressions depend on their distributional properties in the linguistic contexts

— The degree of semantic similarity between two linguistic expressions is a function of the similarity of the two linguistic contexts in which they can appear

— Row vector describes usage of word in a corpus of text

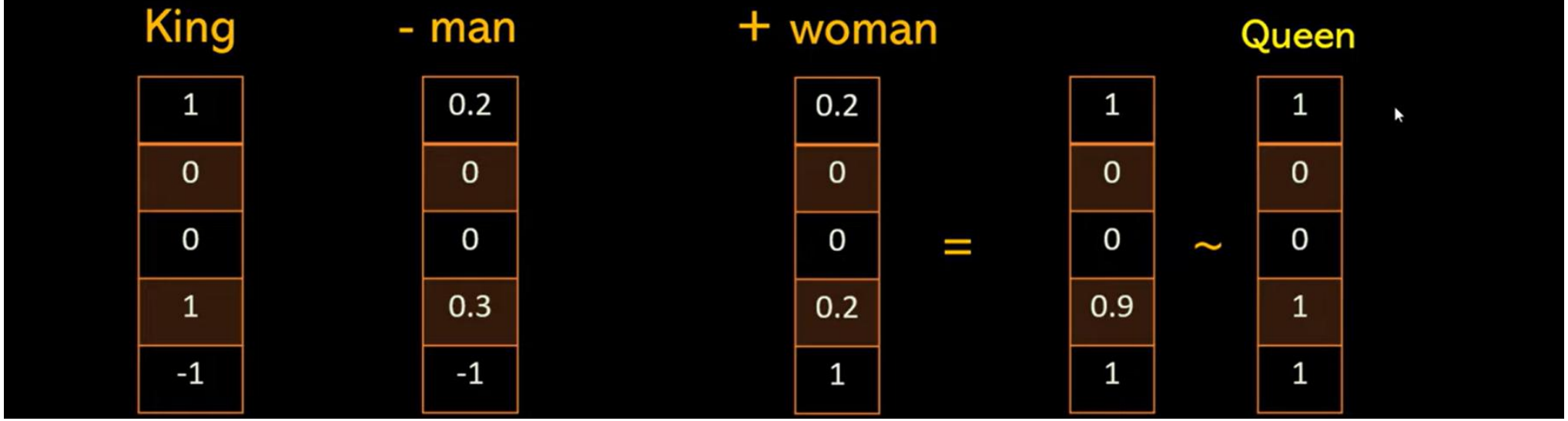
— Can be seen as coordinates o the point in an n-dimensional Euclidian space

	get	see	use	hear	eat	kill
knife	51	20	84	0	3	0
cat	52	58	4	4	6	26
dog	115	83	10	42	33	17
boat	59	39	23	4	0	0
cup	98	14	6	2	1	0
pig	12	17	3	2	9	27
banana	11	2	2	0	18	0

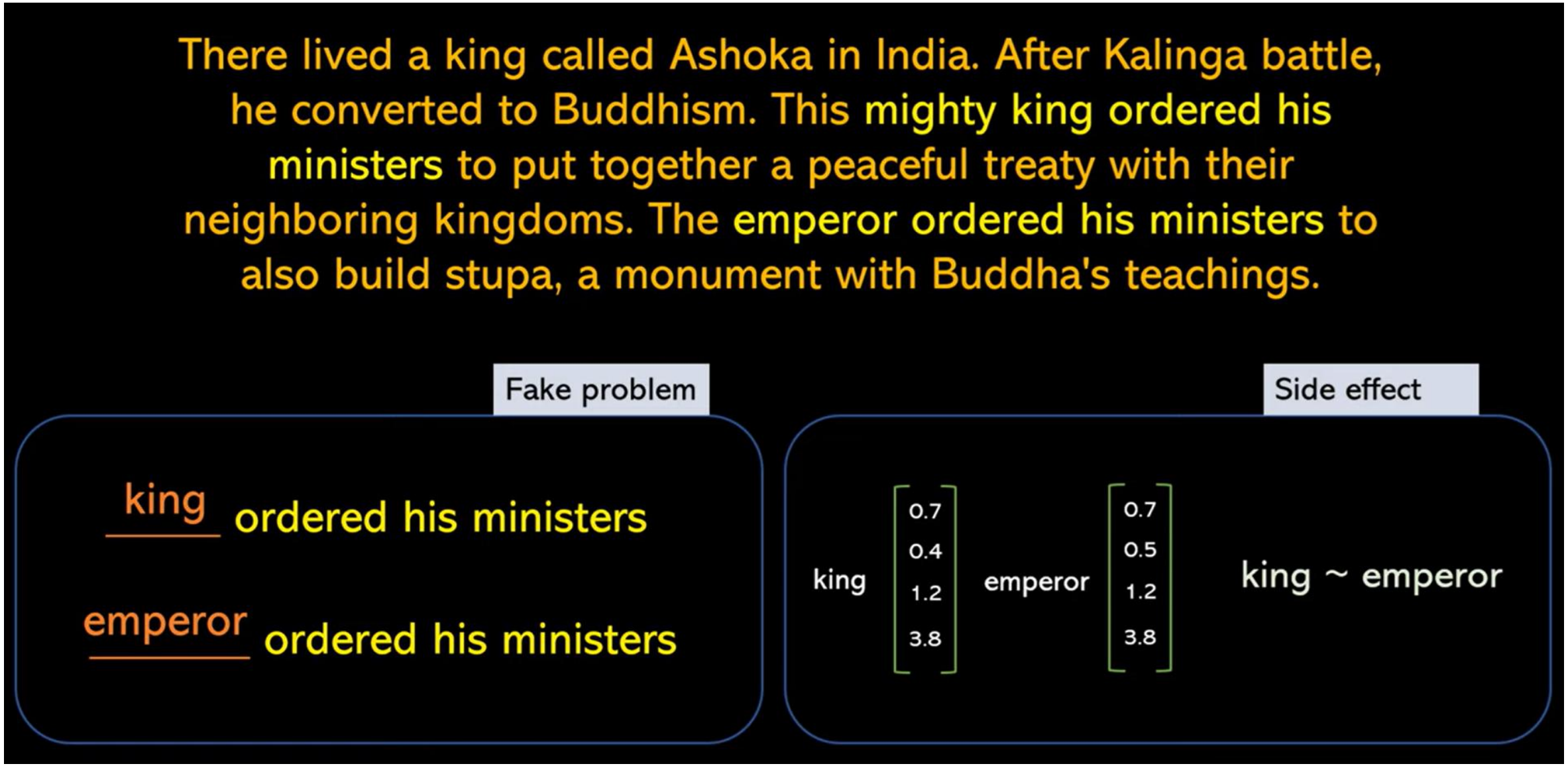
Co-occurrence Matrix

Word Embedding: Word Representation using features example

	battle	horse	king	man	queen	..	woman
authority	0	0.01	1	0.2	1	...	0.2
event	1	0	0	0	0	...	0
has tail?	0	1	0	0	0	...	0
rich	0	0.1	1	0.3	1	...	0.2
gender	0	1	-1	-1	1	...	1



Word2Vec: Intuition (how word embeddings are generated?)



Word Embedding using Word2Vec

- **Word2Vec** was proposed by Mikolov et. al. in 2013.
- It is a two-layer shallow neural network trained to learn the contextual relationship.
- It places contextually similar words nearer to each other.
- It is Co-occurrence based model
- Instead of entire documents, ***Word2Vec*** uses words ***k*** positions away from each center word.
  - These words are called **context words**.
- Example 1
  - for ***k=3***:
    - "It was a bright cold day in April, and the clocks were striking".
    - Center word: red (also called focus word).
    - Context words: blue (also called target words).
- Example 2
  - for ***k=2***:
    - "The cat sat on floor".
    - Center word: red (also called focus word).
    - Context words: blue (also called target words).

- Word2Vec considers all words as center words, and all their context words.

Brilliant insight: Use running text as implicitly supervised training data!

- A word **s** near **apricot**
- Acts as gold ‘correct answer’ to the question

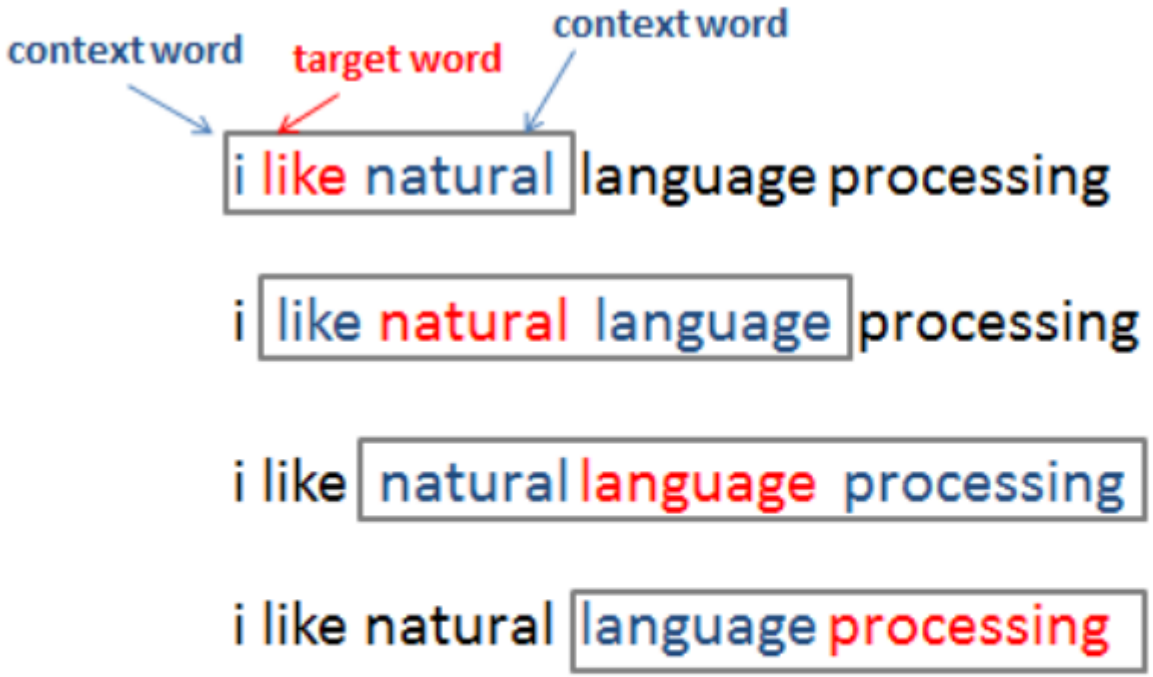
“Is word w likely to show up near apricot?”

- No need for hand-labeled supervision
- The idea comes from neural language modeling
  - Bengio et al. (2003)
  - Collobert et al. (2011)

Training sentence:
... lemon, a tablespoon of **apricot** jam a pinch ...
c1 c2 target c3 c4

Training data: input/output pairs centering on *apricot*
Assume context words are those in +/- 2 word window

CBOW Training Example

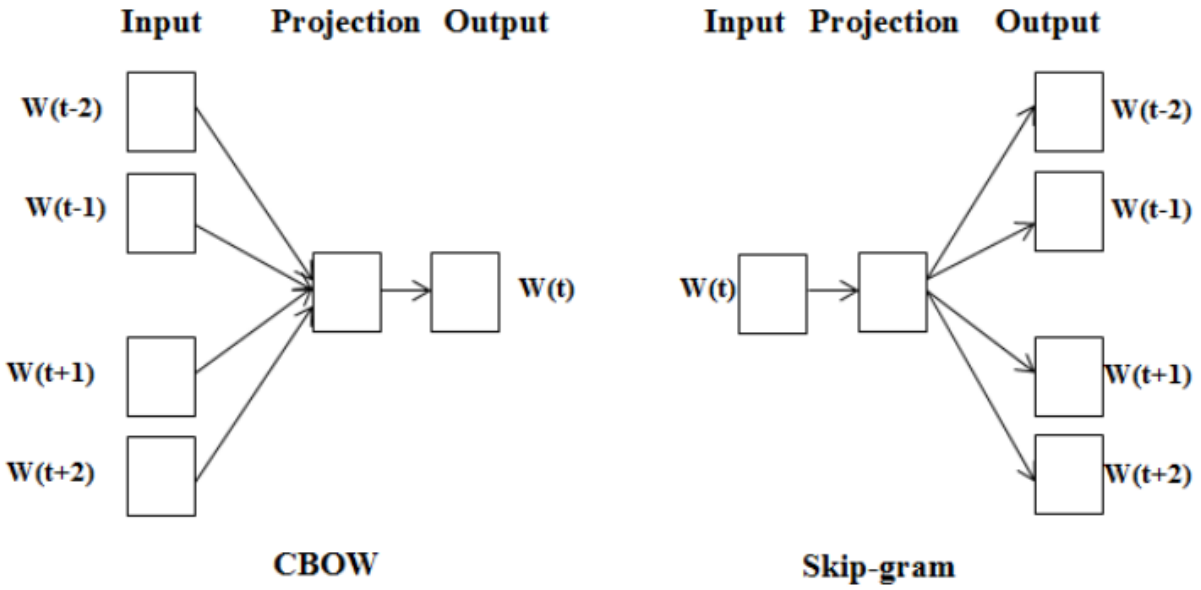


Training Example	Context Word	Target Word
#1	(i, natural)	like
#2	(like, language)	natural
#3	(natural, processing)	language
#4	(language)	processing

Source: https://thinkinfi.com/continuous-bag-of-words-cbow-multi-word-model-how-it-works/

Word2Vec: CBOW and Skip Gram models

Models for projection



Inputs and Targets (As characters and as Integer representation)

```
[['a', 'b', 'c'],
 ['b', 'a', 'c', 'd'],
 ['c', 'a', 'b', 'd', 'e'],
 ['d', 'b', 'c', 'e', 'f'],
 ['e', 'c', 'd', 'f', 'g'],
 ['f', 'd', 'e', 'g', 'h'],
 ['g', 'e', 'f', 'h', 'i'],
 ['h', 'f', 'g', 'i', 'j'],
 ['i', 'g', 'h', 'j', 'k'],
 ['j', 'h', 'i', 'k', 'l'],
 ['k', 'i', 'j', 'l', 'm'],
 ['l', 'j', 'k', 'm', 'n'],
 ['m', 'k', 'l', 'n', 'o'],
 ['n', 'l', 'm', 'o', 'p'],
 ['o', 'm', 'n', 'p', 'q'],
 ['p', 'n', 'o', 'q', 'r'],
 ['q', 'o', 'p', 'r', 's'],
 ['r', 'p', 'q', 's', 't'],
 ['s', 'q', 'r', 't', 'u'],
 ['t', 'r', 's', 'u', 'v'],
 ['u', 's', 't', 'v', 'w'],
 ['v', 't', 'u', 'w', 'x'],
 ['w', 'u', 'v', 'x', 'y'],
 ['x', 'v', 'w', 'y', 'z'],
 ['y', 'w', 'x', 'z'],
 ['z', 'x', 'y']]
```

```
[[0, 1, 2],
 [1, 0, 2, 3],
 [2, 0, 1, 3, 4],
 [3, 1, 2, 4, 5],
 [4, 2, 3, 5, 6],
 [5, 3, 4, 6, 7],
 [6, 4, 5, 7, 8],
 [7, 5, 6, 8, 9],
 [8, 6, 7, 9, 10],
 [9, 7, 8, 10, 11],
 [10, 8, 9, 11, 12],
 [11, 9, 10, 12, 13],
 [12, 10, 11, 13, 14],
 [13, 11, 12, 14, 15],
 [14, 12, 13, 15, 16],
 [15, 13, 14, 16, 17],
 [16, 14, 15, 17, 18],
 [17, 15, 16, 18, 19],
 [18, 16, 17, 19, 20],
 [19, 17, 18, 20, 21],
 [20, 18, 19, 21, 22],
 [21, 19, 20, 22, 23],
 [22, 20, 21, 23, 24],
 [23, 21, 22, 24, 25],
 [24, 22, 23, 25],
 [25, 23, 24]]
```

CBOW (Continuous Bag Of Words) and Skip-Gram are two most popular frames for word embedding. In CBOW the words occurring in context (surrounding words) of a selected word are used as inputs and middle or selected word as the target. Its the other way round in Skip-Gram, here the middle word tries to predict the words coming before and after it.

Word2Vec with Continuous Bag of Words (CBOW)

❖CBOW predicts a target word based on its **context words**.

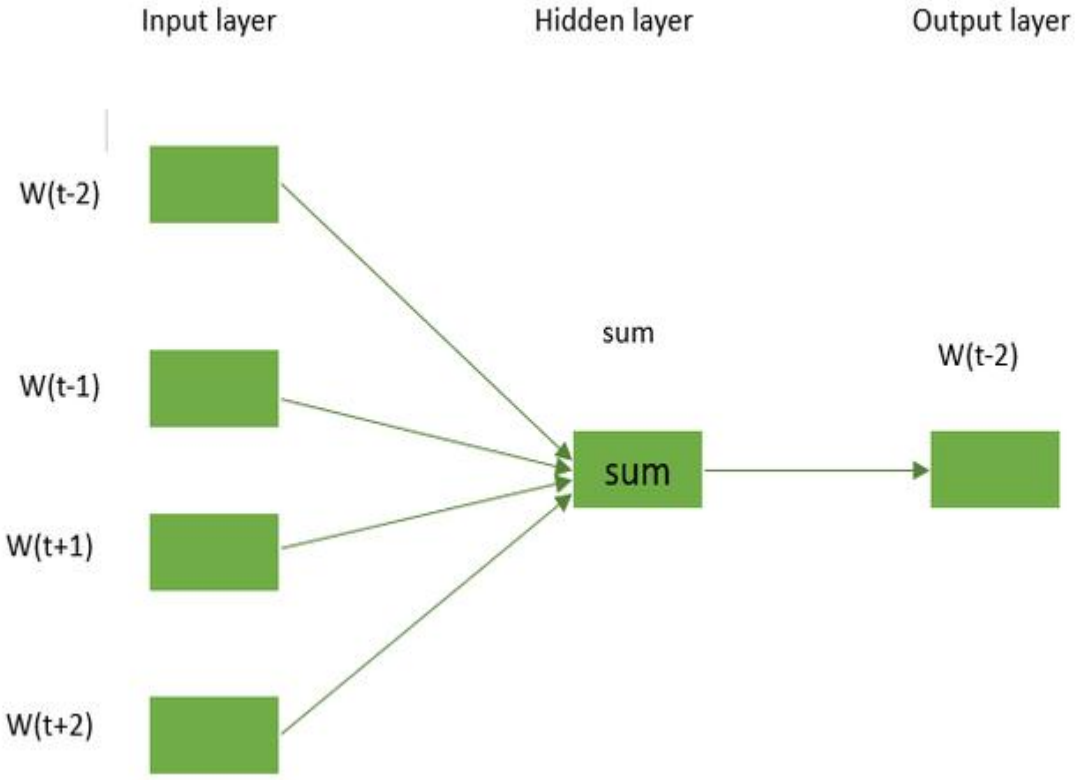
How CBOW Works:

For example, if the window size is **2**, the model will look at two words before and two words after the target word in the sentence.

"The cat sat on the mat."

And the target word is "sat" with a window size of 2, the context would be:

•Context words: ["The", "cat", "on", "the"]

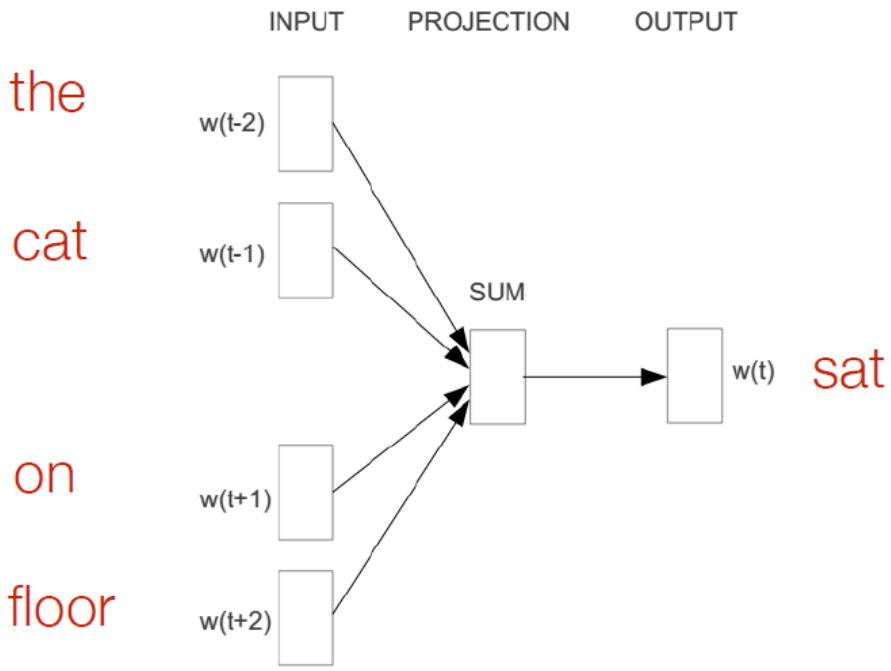


Output: Target Word

- •The model learns to predict the word sat using the context words.

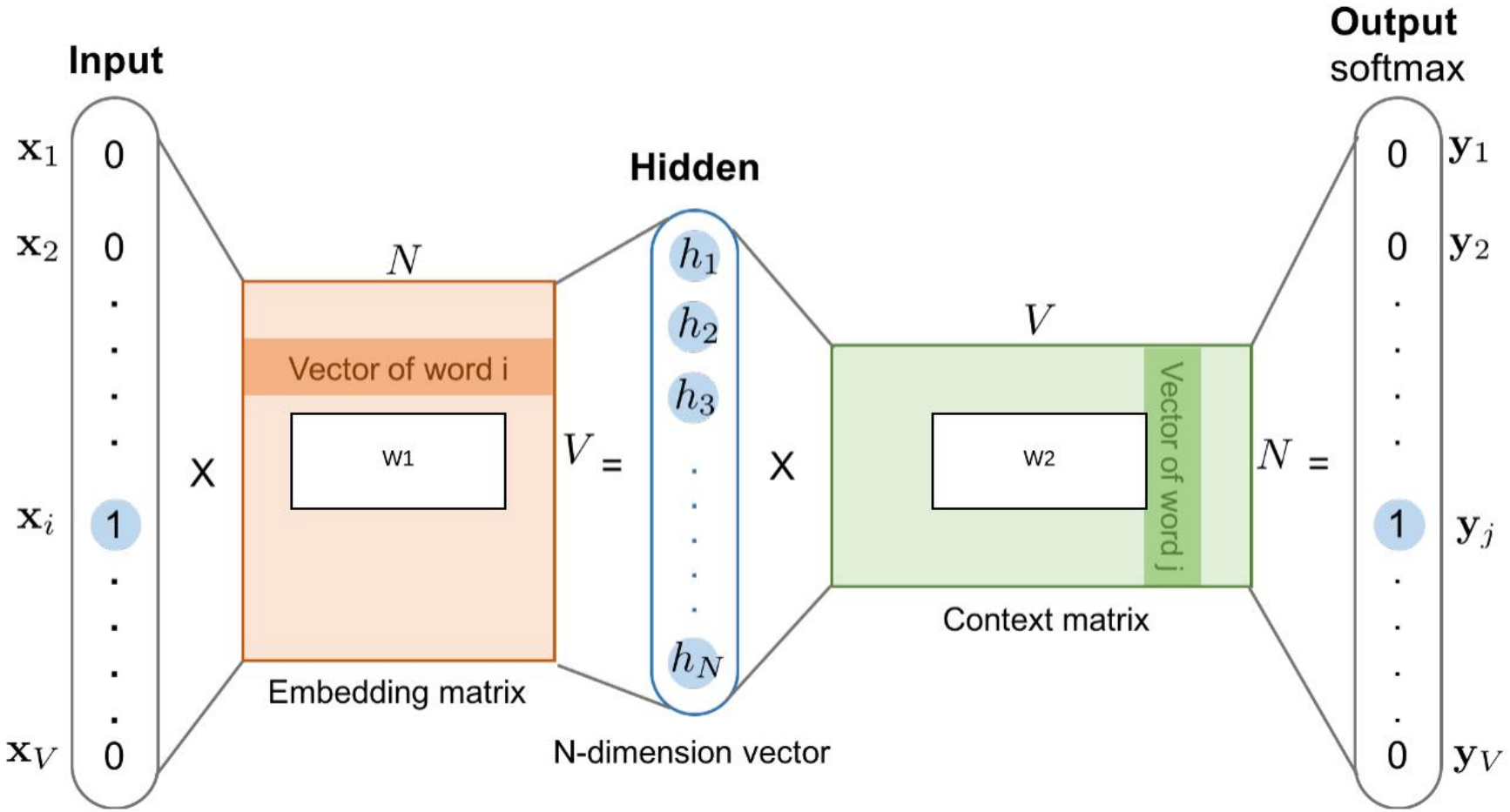
CBOW: Continuous Bag of Words

Example: “The cat sat on floor” (window size 2)



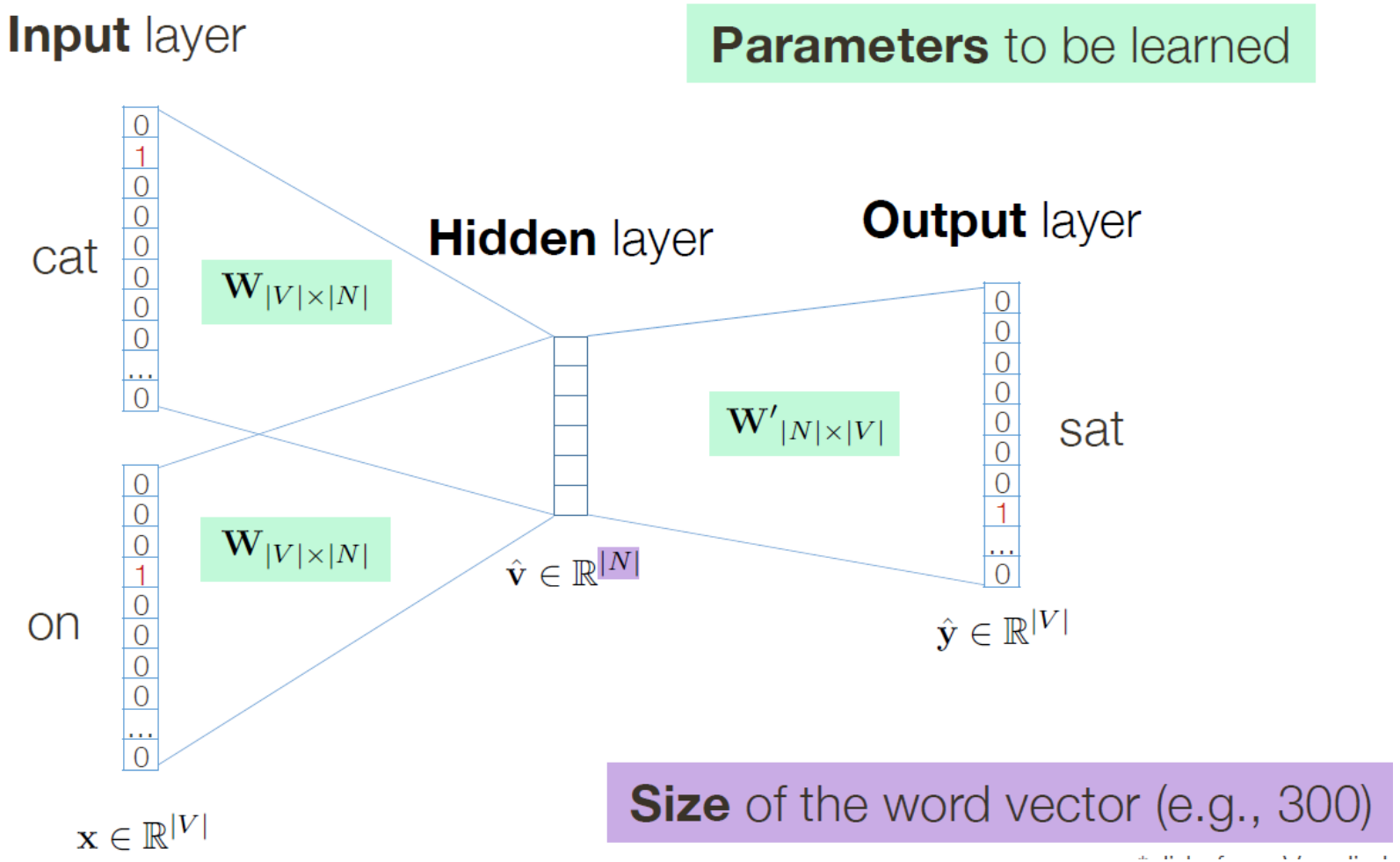
Word2vec Model : Schematic Diagram

Word2Vec is one of the most popular technique to learn word embeddings using shallow neural network. It was developed by Tomas Mikolov in 2013 at Google.

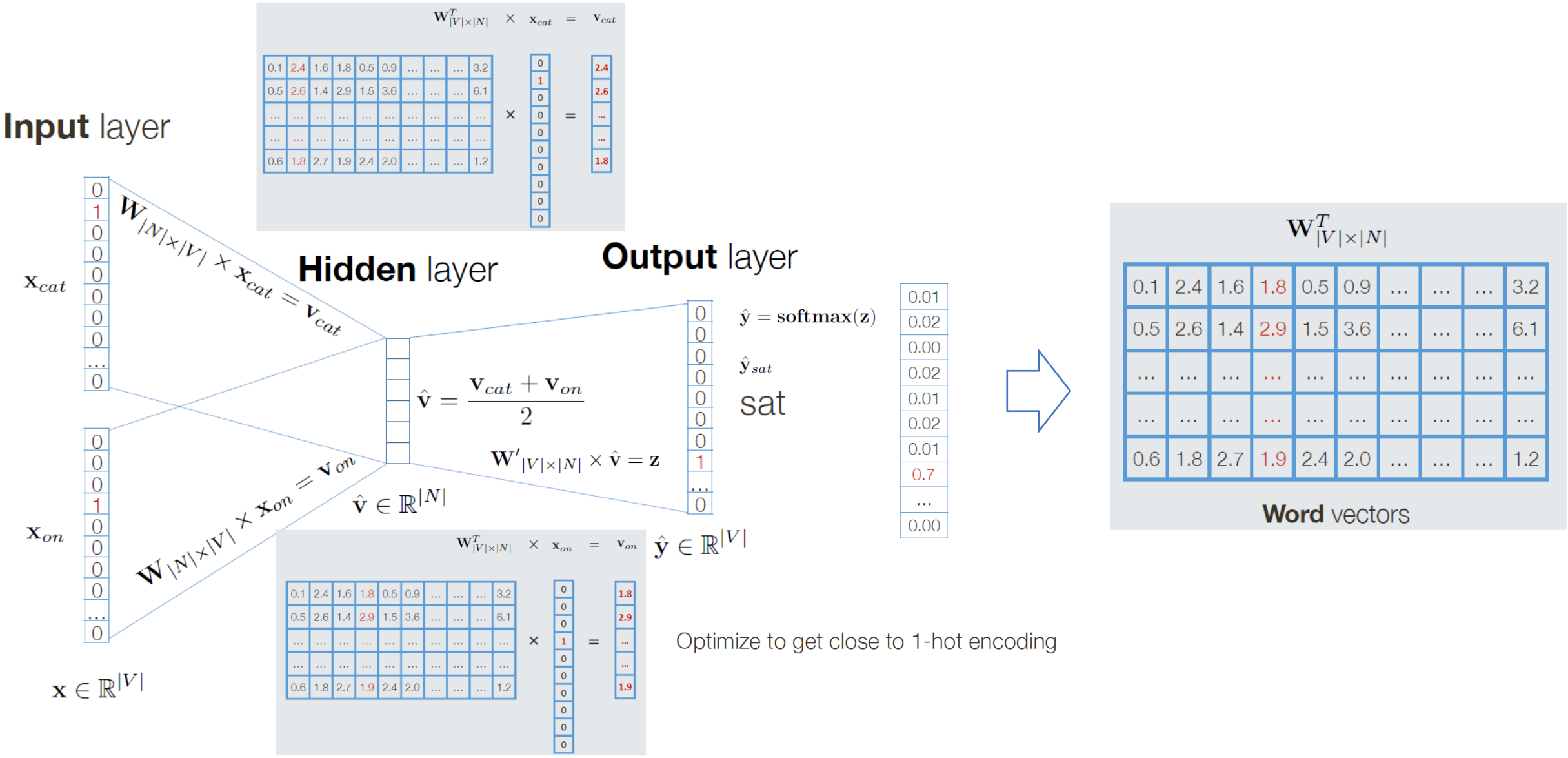


Input and Output vectors are of size V, while hidden vector (embedding) is of size N.
(Source: https://lilianweng.github.io/lil-log/2017/10/15/learning-word-embedding.html)

Continuous Bag Of Words Model



Continuous Bag Of Words Example



1. Input Layer

The CBOW model takes a **context window** around a target word as input.

- **Input:** Surrounding words (context) of the target word.
- **Representation:** Words are one-hot encoded vectors (size = vocabulary size).

2. Embedding Layer

The one-hot encoded input is passed to an embedding layer, which is essentially a weight matrix W.

- **Weight Matrix:** W of size $V \times d$, where:
  - V: Vocabulary size
  - d: Embedding size (number of dimensions of word vectors)
- **Transformation:** Multiply each one-hot vector by W:

$$h_i = W^\top x_i$$

  - x_i: One-hot vector of the i-th context word
  - h_i: Embedding vector of the i-th context word

3. Hidden Layer

The embedding vectors of the context words are averaged to create a single vector.

- **Averaging:**

$$h = \frac{1}{n} \sum_{i=1}^{n} h_i$$

  - h: Hidden layer representation (context vector)

4. Output Layer

The context vector h is passed to the output layer to predict the target word.

- **Weight Matrix:** Another weight matrix W' of size $d \times V$ is used.
- **Transformation:** Compute scores for all words in the vocabulary:

$$z = W'^\top h$$

  - z: Scores for each word in the vocabulary.
- **Softmax Function:** Convert the scores into probabilities:

$$P(w_t|context) = \frac{\exp(z_t)}{\sum_{j=1}^{V} \exp(z_j)}$$

  - w_t: Target word
  - $P(w_t|context)$: Probability of the target word given the context.

5. Loss Function

The goal is to maximize the probability of the correct target word.
Reduce Cross Entropy Loss.

6. Training

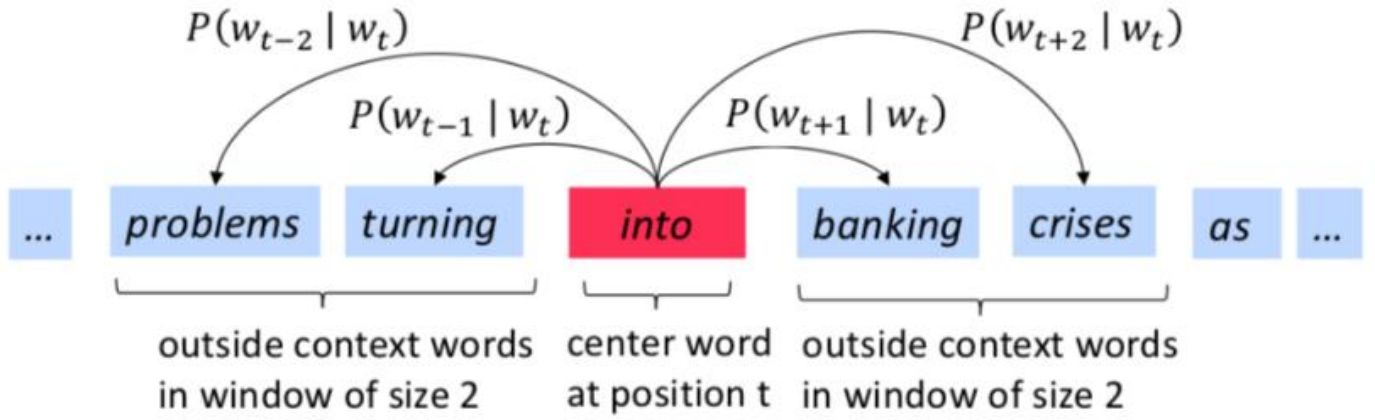
The model is trained using backpropagation

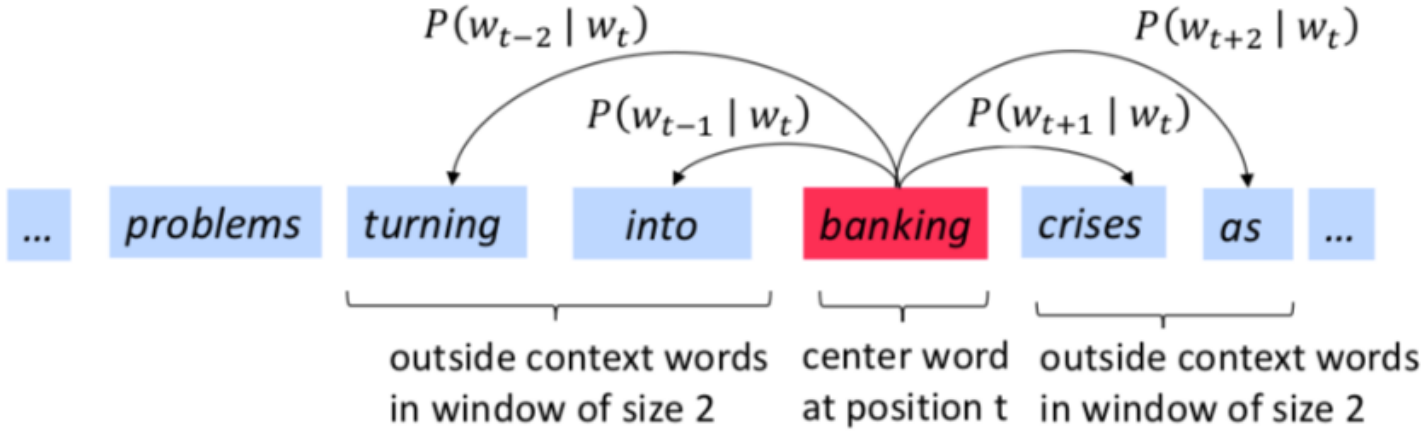
- Update the weights to minimize the loss.

Once the training is done in the whole vocabulary, the weight matrix $\mathbf{W_{VxN}}$ contains the word embeddings, where each row corresponds to the vector representation of a word

Skip-gram

- The idea: we want to use words to **predict** their context words
- Context: a fixed window of size $2m$

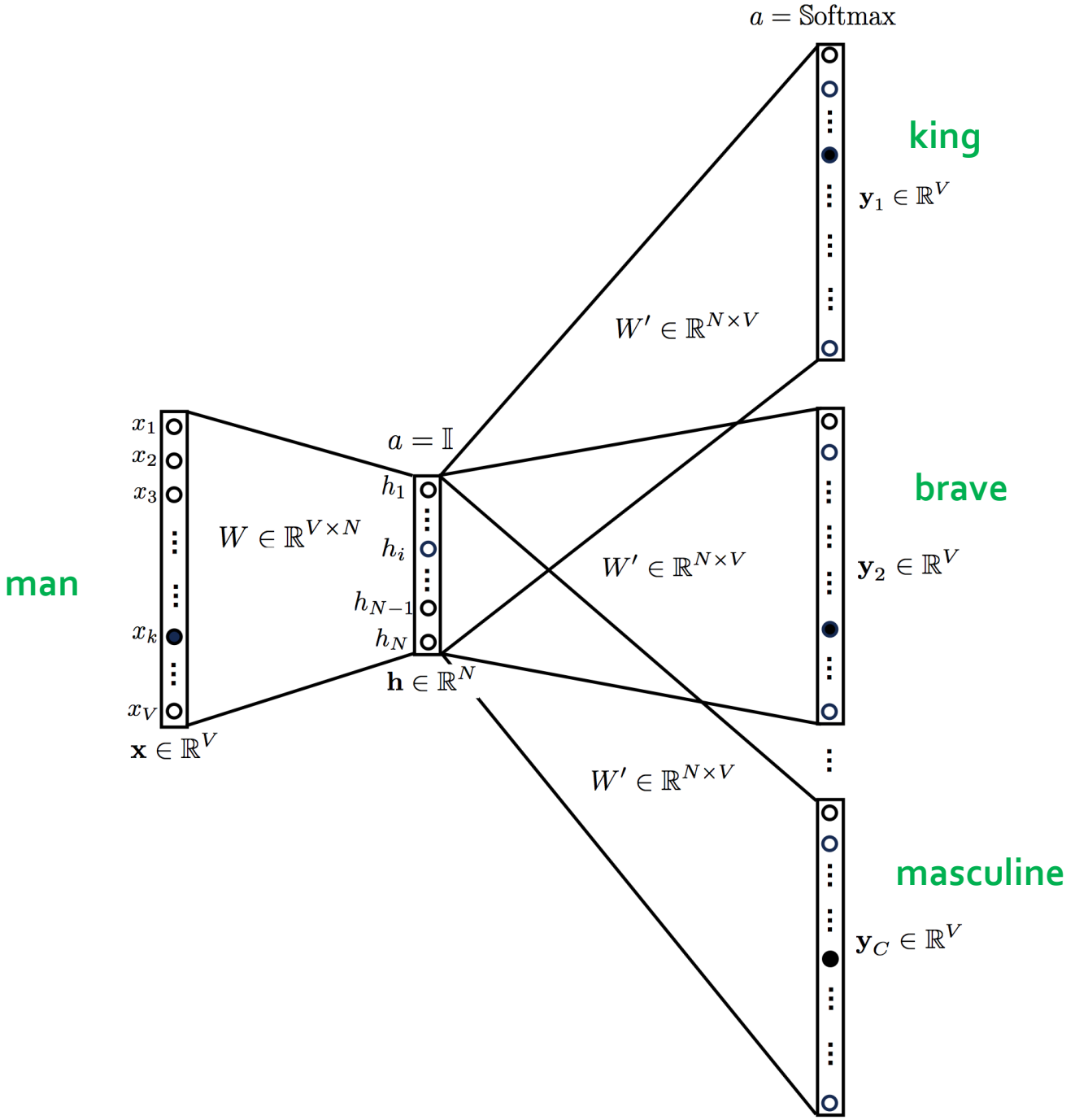




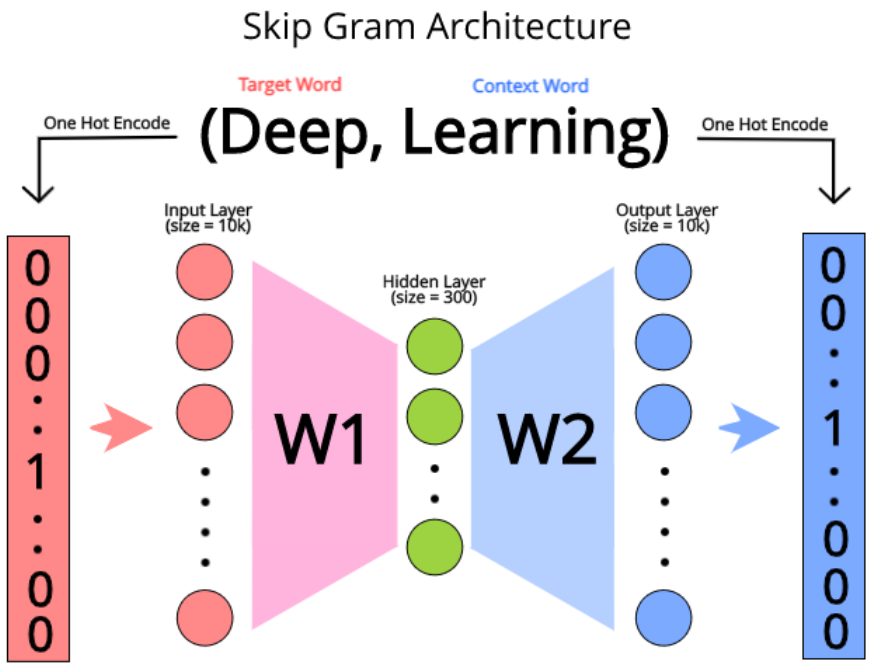
Skip-Gram Model

Skip-gram algorithm

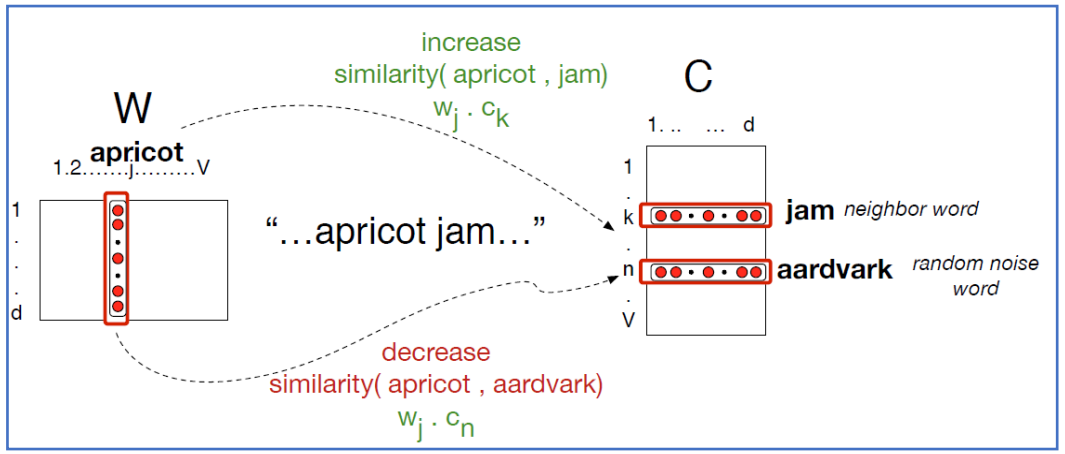
1. Treat the target word and a neighboring context word as positive examples.
2. Randomly sample other words in the lexicon to get negative samples
3. Use logistic regression to train a classifier to distinguish those two cases
4. Use the weights as the embeddings



Skip Gram Model



Iterative approach: Assume an initial set of vectors, and then adjust them during training to maximize the probability of the training examples



Skip-gram algorithm

1. Treat the target word and a neighboring context word as positive examples.
2. Randomly sample other words in the lexicon to get negative samples
3. Use logistic regression to train a classifier to distinguish those two cases
4. Use the weights as the embeddings

Training sentence:

... lemon, a tablespoon of **apricot** jam a pinch ...

c1 c2 target c3 c4

Training data: input/output pairs centering on *apricot*

Assume context words are those in +/- 2 word window

Given a tuple (t,c) = target, context

- (*apricot*, *jam*)
- (*apricot*, *aardvark*)

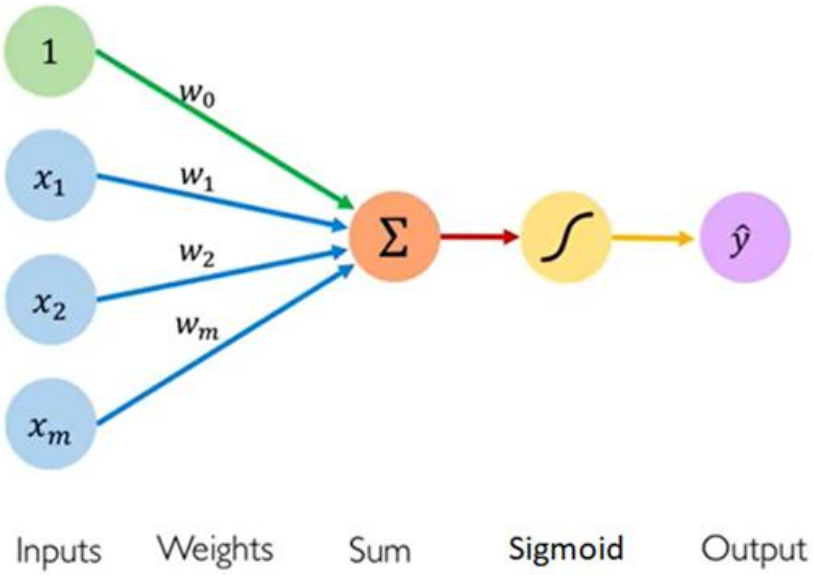
Return probability that c is a real context word:

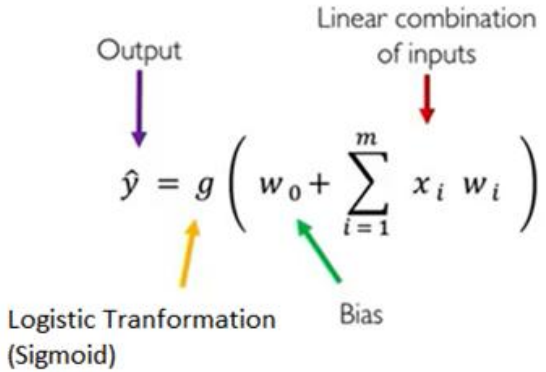
P(+|t,c) **Positive Sample**

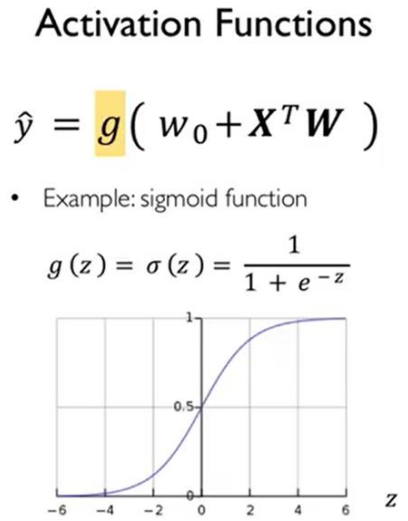
***P*(−|*t*,*c*) = 1−*P*(+|*t*,*c*)** **Negative Sample**

Logistic Regression

- Perceptron with a sigmoid activation function is used to solve Logistic regression problems
- The output for a logistic regression is a number that represents the probability of the event happening (Yes / No)







Source: *MIT 6.S191 (2020): Introduction to Deep Learning*

How to compute p(+|t,c)?

Intuition:

- Words are likely to appear near similar words
- Model similarity with dot-product!
- Similarity(t,c) $\propto t \cdot c$

Problem:

- *Dot product is not a probability!*
  - *(Neither is cosine)*

For all the context words:

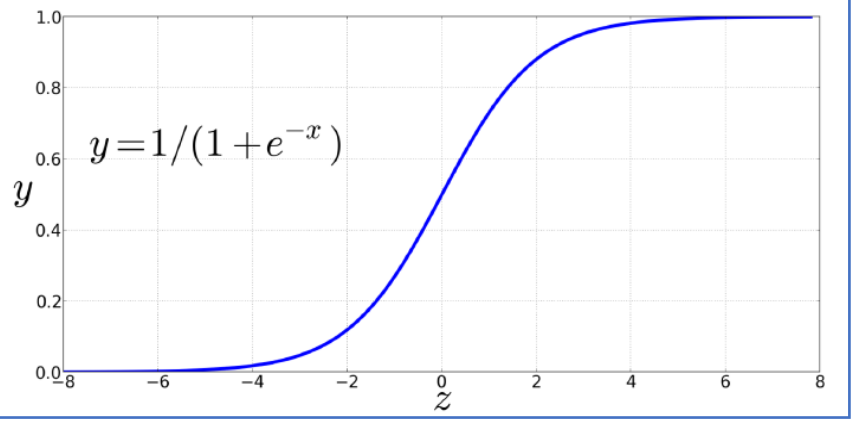
Assume all context words are independent

$$P(+|t, c_{1:k}) = \prod_{i=1}^{\kappa} \frac{1}{1+e^{-t \cdot c_i}}$$

$$\log P(+|t, c_{1:k}) = \sum_{i=1}^{k} \log \frac{1}{1+e^{-t \cdot c_i}}$$

The sigmoid lies between 0 and 1:

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



Could pick w according to their unigram frequency P(w)

More common to chosen then according to $p_\alpha(w)$

$$P_\alpha(w) = \frac{count(w)^\alpha}{\sum_w count(w)^\alpha}$$

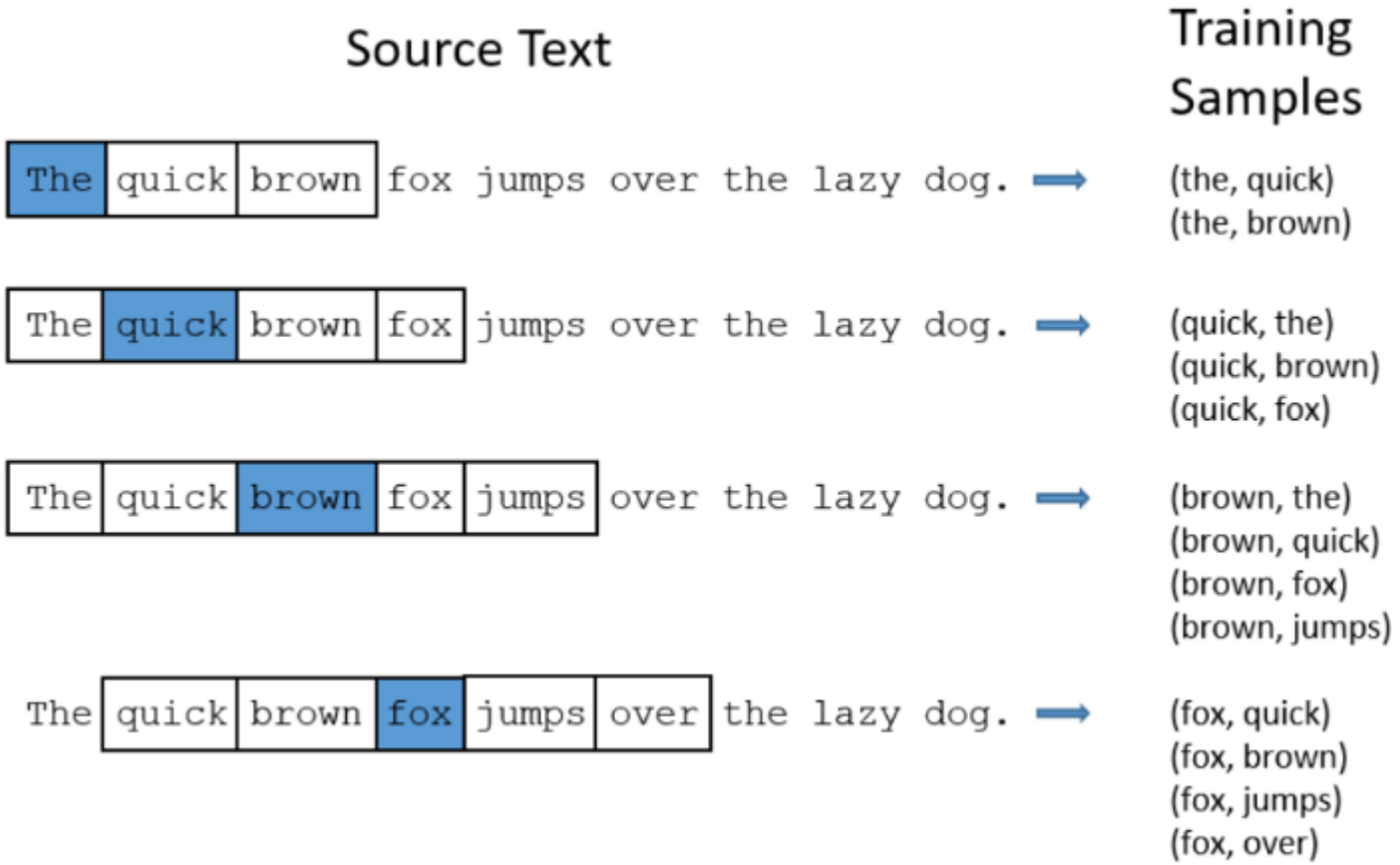
α= ¾ works well because it gives rare noise words slightly higher probability

To show this, imagine two events p(a)=.99 and p(b) = .01:

$$P_\alpha(a) = \frac{.99^{.75}}{.99^{.75}+.01^{.75}} = .97$$

$$P_\alpha(b) = \frac{.01^{.75}}{.99^{.75}+.01^{.75}} = .03$$

Skip-gram training example



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

Skip-Gram Training

Training sentence:

... lemon, a tablespoon of **apricot** jam a pinch ...

c1 c2 t c3 c4

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	preserves
apricot	or

- For each positive example, we'll create *k* negative examples.
- Using *noise* words
- Any random word that isn't *t*

negative examples - k=2

t	c	t	c
apricot	aardvark	apricot	twelve
apricot	puddle	apricot	hello
apricot	where	apricot	dear
apricot	coaxial	apricot	forever

Skip-gram algorithm to learn word2vec embeddings

Setup

Let's represent words as vectors of some length (say 300), randomly initialized.

So we start with 300 * V random parameters

Over the entire training set, we'd like to adjust those word vectors such that we

- Maximize the similarity of the target word, context word pairs (t,c) drawn from the positive data
- Minimize the similarity of the (t,c) pairs drawn from the negative data.

Objective Criteria

We want to maximize…

$$\sum_{(t,c)\in+} logP(+|t,c) + \sum_{(t,c)\in-} logP(-|t,c)$$

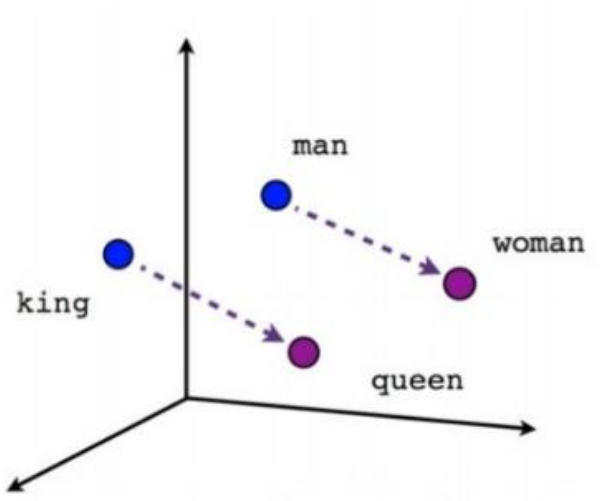
Maximize the + label for the pairs from the positive training data, and the – label for the pairs sample from the negative data.

Start with V random 300-dimensional vectors as initial embeddings

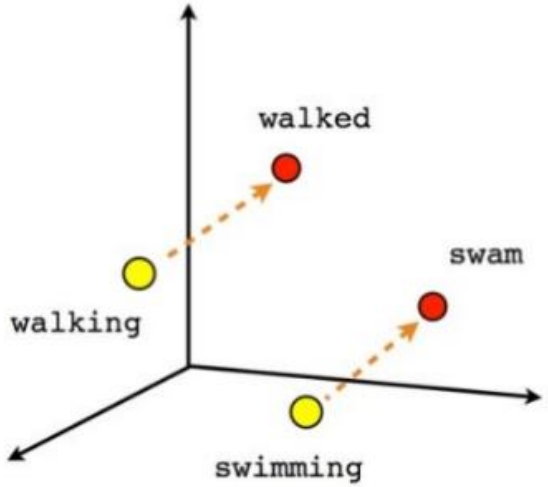
Use logistic regression, the second most basic classifier used in machine learning after naïve bayes

- Take a corpus and take pairs of words that co-occur as positive examples
- Take pairs of words that don't co-occur as negative examples
- Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance
- Throw away the classifier code and keep the embeddings.

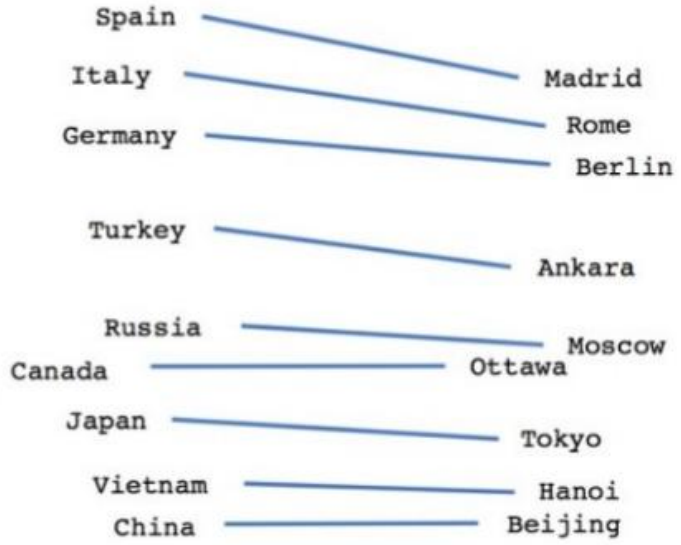
Word Vectors



Male-Female



Verb tense

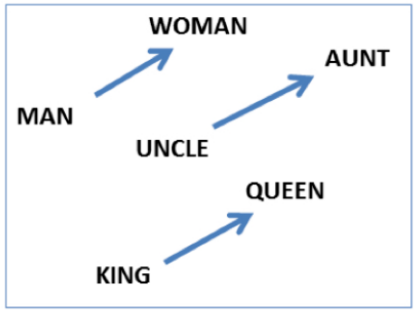


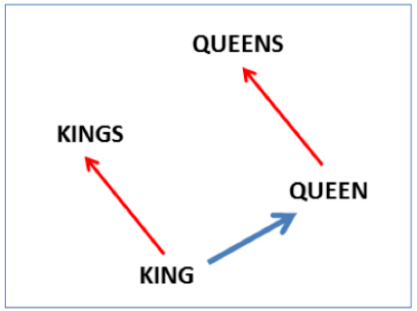
Country-Capital

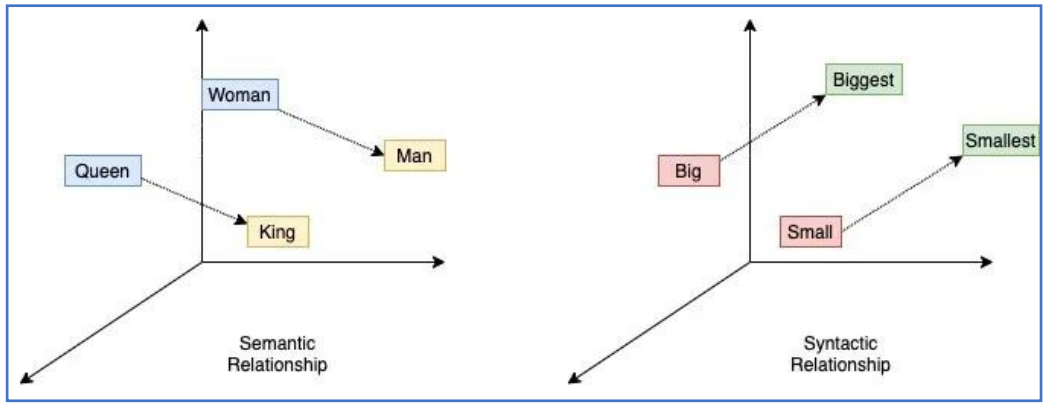
Embeddings capture meaning!

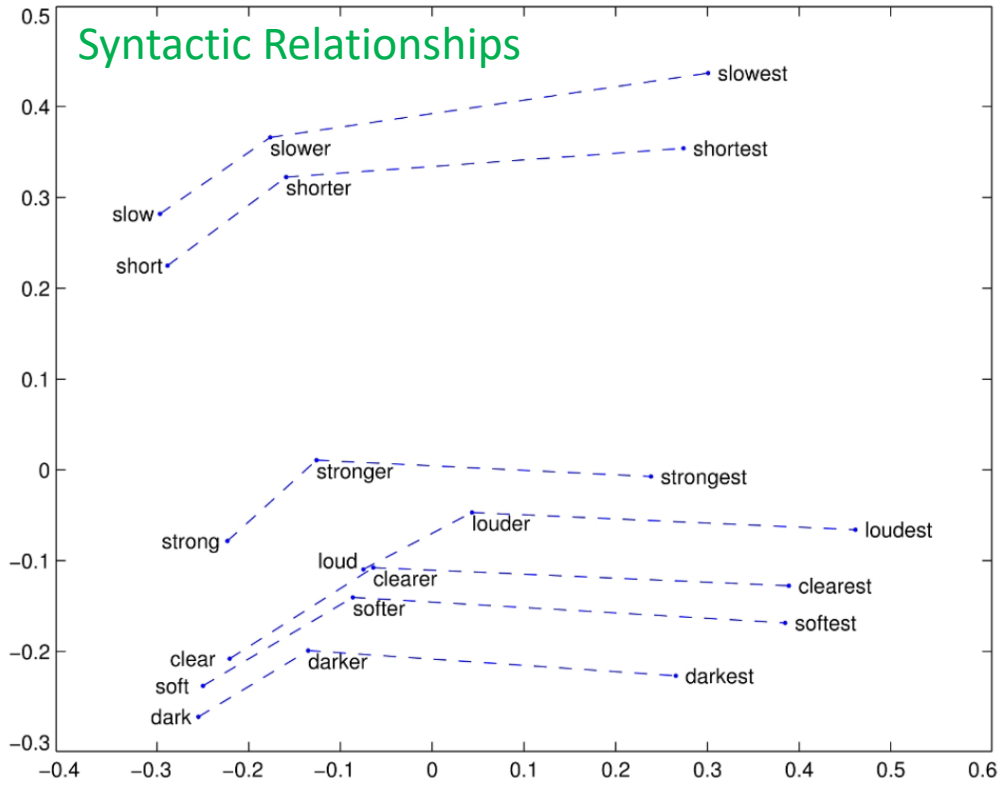
vector(*'king'*) - vector(*'man'*) + vector(*'woman'*) ≈ vector('queen')

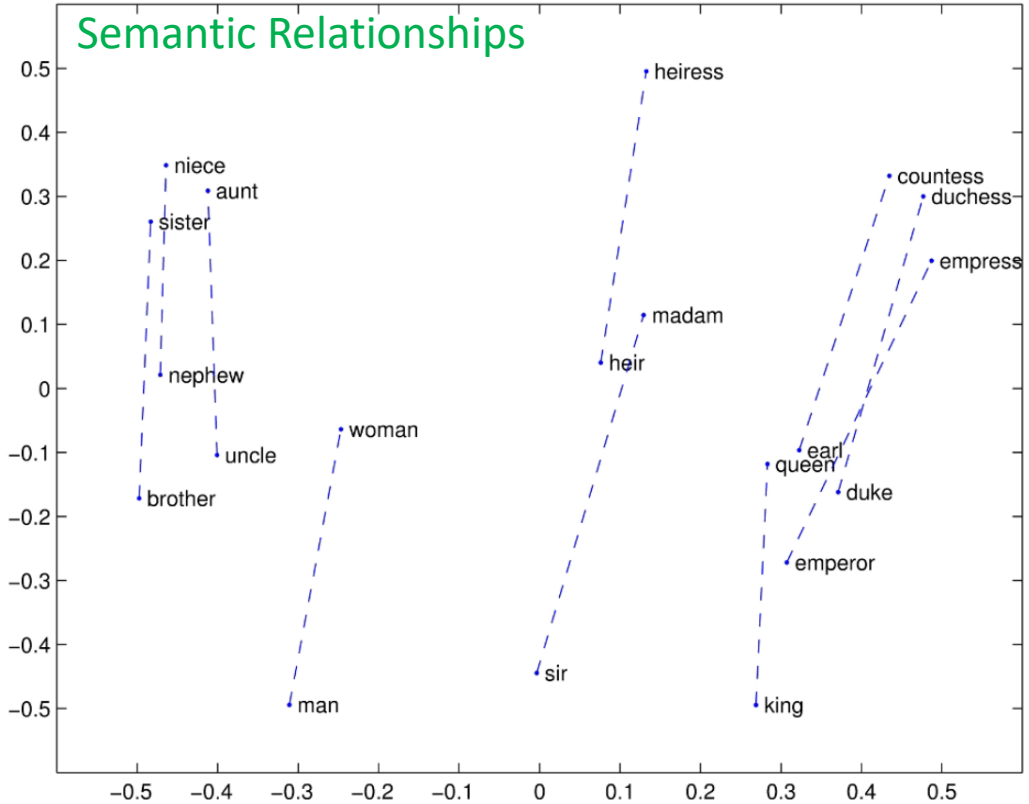
vector(*'Paris'*) - vector(*'France'*) + vector(*'Italy'*) ≈ vector('Rome')











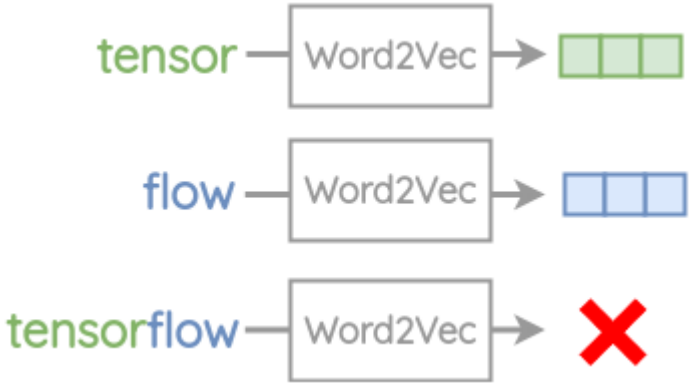
Word2Vec: CBOW and Skip-gram model comparison

- **CBOW** is not great for rare words and typically needs less data to train
- **Skip-gram** better for rate words and needs more data to train the model

Model	Vector Dimensionality	Training words	Accuracy [%]		
			Semantic	Syntactic	Total
Collobert-Weston NNLM	50	660M	9.3	12.3	11.0
Turian NNLM	50	37M	1.4	2.6	2.1
Turian NNLM	200	37M	1.4	2.2	1.8
Mnih NNLM	50	37M	1.8	9.1	5.8
Mnih NNLM	100	37M	3.3	13.2	8.8
Mikolov RNNLM	80	320M	4.9	18.4	12.7
Mikolov RNNLM	640	320M	8.6	36.5	24.6
Huang NNLM	50	990M	13.3	11.6	12.3
Our NNLM	20	6B	12.9	26.4	20.3
Our NNLM	50	6B	27.9	55.8	43.2
Our NNLM	100	6B	34.2	**64.5**	50.8
CBOW	300	783M	15.5	53.1	36.1
Skip-gram	300	783M	**50.0**	55.9	**53.3**

Word2vec Limitations

- Out of Vocabulary(OOV) Words:
  - In Word2Vec, an embedding is created for each word. As such, it can't handle any words it has not encountered during its training.
  - For example, words such as "tensor" and "flow" are present in the vocabulary of Word2Vec. But if you try to get embedding for the compound word "tensorflow", you will get an out of vocabulary error.
- Morphology:
  - For words with same radicals such as "eat" and "eaten", Word2Vec doesn't do any parameter sharing.
  - Each word is learned uniquely based on the context it appears in. Thus, there is scope for utilizing the internal structure of the word to make the process more efficient.
- Uniform Representation Across Contexts + Single Vector per word:
  - **Single Vector Per Word**: Word2Vec assigns a unique vector to each word, which remains static regardless of the word's varying context in different sentences. This results in a representation that cannot dynamically adapt to different usages of the same word.
  - Word2Vec, like other traditional methods, generates a global vector representation for each word, which does not account for the various meanings a word can have in different contexts.
  - In cases where a word like "bank" appears in multiple contexts ("river bank" vs. "financial bank"), Word2Vec does not generate distinct embeddings for each scenario. Instead, it creates a singular, averaged representation that amalgamates all the contexts in which the word appears, leading to a generalized semantic representation.



Shared radical

eat eats eaten eater eating

bank = financial institution or river bank

Word2Vec: Key Limitations

While Word2Vec's method of word embedding results in a context-agnostic, single representation for each word, this becomes a significant limitation when dealing with polysemous words, i.e., words that possess multiple meanings based on their context. More advanced models such as BERT and ELMo have since been developed to provide context-dependent embeddings, where the representation of a word can dynamically vary based on its usage within a sentence. Here are the specifics:

1. **Static, Non-Contextualized Nature**:
    1. **Single Vector Per Word**: Word2Vec assigns a unique vector to each word, which remains static regardless of the word's varying context in different sentences. This results in a representation that cannot dynamically adapt to different usages of the same word.
    2. **Combination of Contexts**: In cases where a word like "bank" appears in multiple contexts ("river bank" vs. "financial bank"), Word2Vec does not generate distinct embeddings for each scenario. Instead, it creates a singular, averaged representation that amalgamates all the contexts in which the word appears, leading to a generalized semantic representation.
    3. **Lack of Disambiguation**: The model's inability to differentiate between the multiple meanings of polysemous words means that words like "bank" are represented by a single vector, irrespective of the specific meaning in a given context.
    4. **Context Window Limitation**: Word2Vec employs a fixed-size context window, capturing only local co-occurrence patterns without a deeper understanding of the word's role in the broader sentence or paragraph.
2. **Training Process and Computational Intensity**:
    1. **Adjustments During Training**: Throughout the training process, the word vectors are continually adjusted, not to switch between meanings but to refine the word's placement in the semantic space based on an aggregate of its various uses.
    2. **Resource Demands**: Training Word2Vec, particularly for large vocabularies, requires significant computational resources and time. Techniques like negative sampling were introduced to alleviate some of these demands, but computational intensity remains a challenge.
3. **Handling of Special Cases**:
    1. **Phrase Representation**: Word2Vec struggles with representing phrases or idioms where the meaning is not simply an aggregation of the meanings of individual words.
    2. **Out-of-Vocabulary Words**: The model faces challenges with unknown or out-of-vocabulary (OOV) words. This issue is better addressed in models that treat words as compositions of characters, such as character embeddings, which are especially beneficial for languages with non-segmented script.
4. **Global Vector Representation Limitations**:
    1. **Uniform Representation Across Contexts**: Word2Vec, like other traditional methods, generates a global vector representation for each word, which does not account for the various meanings a word can have in different contexts. For example, the different senses of "bank" in diverse sentences are not captured distinctively.
5. **Resulting Embedding Compromises**:
    1. The resulting vector for words with distinct meanings is a compromise that reflects its diverse uses, leading to less precise representations for tasks requiring accurate contextual understanding.

These limitations of Word2Vec have spurred advancements in the field, leading to the development of more sophisticated language models that address issues of context sensitivity, polysemy, computational efficiency, and handling of OOV words. This progression towards more robust and context-aware word representations underscores the ongoing evolution and potential of natural language processing.

Fast Text Improvements over Skipgram models

To solve some of the above challenges, Bojanowski et al. proposed a new embedding method called FastText.

Source: https://amitness.com/2020/06/fasttext-embeddings/

- Subword level features
  - Motivation: Word internal structure contains rich information which we might want to use in learning word representations.
  - Words that do not occur (frequently) in the dataset (also due to spelling errors) contain character sequences that may hint at their meaning.
  - morphological information
    - e.g. preadolescent → pre + adolesc + ent
- Phrase representations
  - Motivation: Many phrases have a meaning that is not a simple composition of the meanings of its individual words.

  1. New York Times = New + York + Times
  2. this is a = this + is + a
- Position-dependent weighting
  - Motivation: The position of word in the context window also contains useful information which is disregarded in the original Word2vec Skipgram model

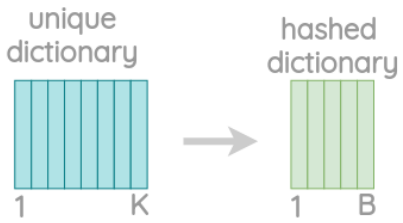
FastText

❖ Key insight: Use the internal structure of a word to improve vector representations obtained from the skip-gram method.

❖ The modification to Skip Gram method:

1. Sub-word generation:

- Take a word and add angular brackets to denote the beginning and end of a word
- Generate character n-grams of length n. Here, we shift the window one step each time.
- Since there can be huge number of unique n-grams, we apply hashing to bound the memory requirements. Instead of learning an embedding for each unique n-gram, we learn total B embeddings where B denotes the bucket size. The paper used a bucket of a size of 2 million.



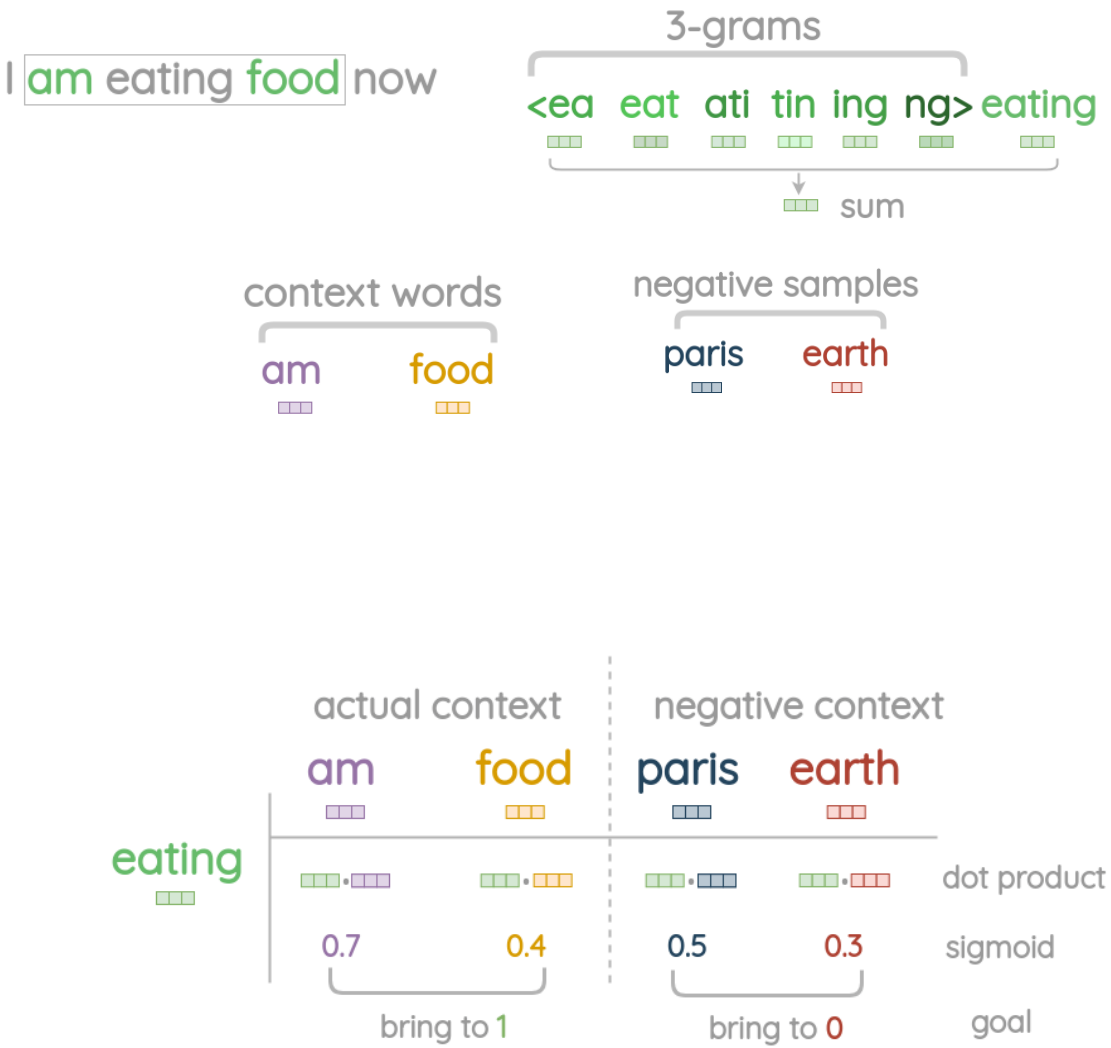
Examples of different length character n-grams:

Word	Length(n)	Character n-grams
eating	3	<ea, eat, ati, tin, ing, ng>
eating	4	<eat, eati, atin, ting, ing>
eating	5	<eati, eatin, ating, ting>
eating	6	<eatin, eating, ating>

FastText

2. Skip-gram with negative sampling:

1. First, the embedding for the center word is calculated by taking a sum of vectors for the character n-grams and the whole word itself.
2. For the actual context words, we directly take their word vector from the embedding table without adding the character n-grams.
3. Now, we collect negative samples randomly with probability proportion to the square root of the unigram frequency. For one actual context word, 5 random negative words are sampled.
4. We take dot product between the center word and the actual context words and apply sigmoid function to get a match score between 0 and 1.
5. Based on the loss, we update the embedding vectors with SGD optimizer to bring actual context words closer to the center word but increase distance to the negative samples.



GloVe: Global Vectors for Word Representation

- Consider two sentences -
  - I am a data science enthusiast
  - I am looking for a data science job

The co-occurrence matrix involved in GloVe would look like this for the above sentences:

	I	am	a	data	science	enthusiast	looking	for	job
I	0	2	0	0	0	0	0	0	0
am	2	0	1	0	0	0	1	0	0
a	0	1	0	2	0	0	0	1	0
data	0	0	2	0	2	0	0	0	0
science	0	0	0	2	0	1	0	0	1
enthusiast	0	0	0	0	1	0	0	0	0
looking	0	1	0	0	0	0	0	1	0
for	0	0	1	0	0	0	1	0	0
job	0	0	0	0	1	0	0	0	0

Window Size = 1

Each value in this matrix represents the count of co-occurrence with the corresponding word in row/column. Observe here - this co-occurrence matrix is created using global word co-occurrence count (no. of times the words appeared consecutively; for window size=1). If a text corpus has 1m unique words, the co-occurrence matrix would be 1m x 1m in shape. The core idea behind GloVe is that the word co-occurrence is the most important statistical information available for the model to 'learn' the word representation.

Co-occurrence Probability

The distributional word representation is learned from count based methods (using co-occurrence statistics of corpus).

$$P(\text{j}|\text{i}) = \text{Probablity(j is in the context of i)}$$

Mathematically -

$$P(\text{j}|\text{i}) = \frac{x_{ij}}{x_i}$$

Where,

$x_{ij} = \text{Co-occurance Matrix[i][j]}$: number of times word ***j*** occurs in the context of word ***i***

$$x_i = \sum_{1}^{k} x_{ik} \text{ (number of times any word appears in the context of word i)}$$

Co-occurrence Probability

- How certain aspects of meaning can be extracted directly from co-occurrence probabilities.

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

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

solid has a much higher co-occurrence probability with **ice** than with **steam**

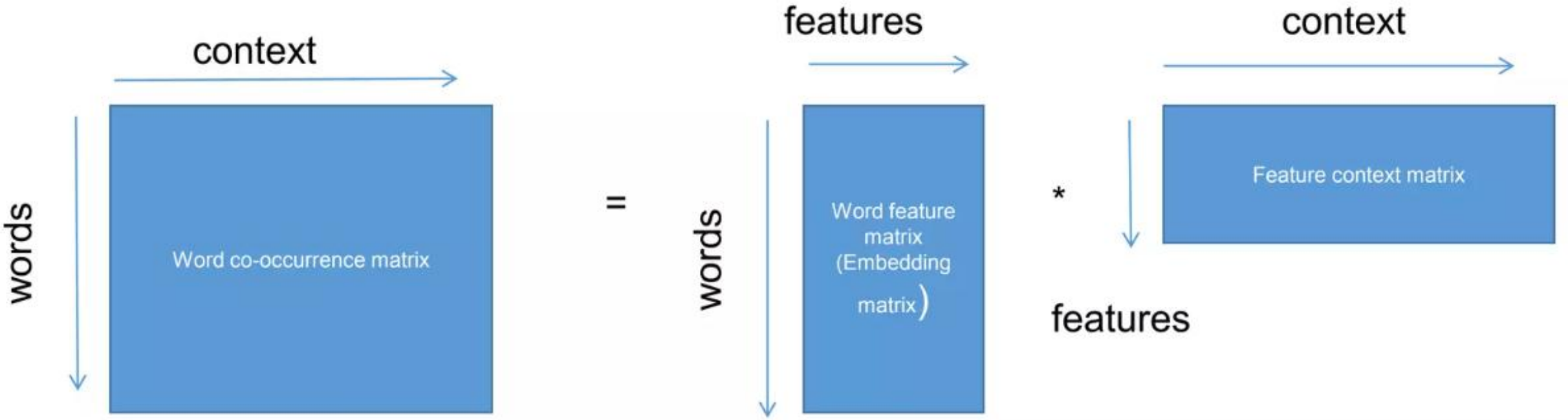
gas has a much higher co-occurrence probability with **steam** than with **ice**

water has a similar (high) co-occurrence probability with both **steam** and **ice**

GloVe: Global Vectors for Word Representation

- GloVe is an Unsupervised learning method to learn word representation
- GloVe is based on co-occurrence matrix
  - The co-occurrence matrix is built on whole corpus (Statistical Measure)
  - It is able to capture Global context by considering neighbouring words
- GloVe focuses on global context to create word embeddings which gives it an edge over Word2Vec. In GloVe, the semantic relationship between the words is obtained using a co-occurrence matrix.
- While word2Vec is a predictive model — learning vectors to improve the predictive ability, **GloVe is a count-based model**.
- Count-based models learn vectors by doing dimensionality reduction on a **co-occurrence counts matrix**.
  - Factorize this matrix to yield a lower-dimensional matrix of words and features, where each row yields a vector representation for each word.
  - The counts matrix is preprocessed by normalizing the counts and log-smoothing them.

GloVe: Global Vectors for Word Representation



❖An Unsupervised learning method to learn word representation

GloVe: Optimization function

- What is the optimization function for GloVe?
  - In a co-occurrence matrix X the X_{ij} represents the co-occurrence count.
  - X_i is the total number of times the word appears in the context.
  - $P_{ij} = P(j/i) = X_{ij}/X_i$ is the probability of word j appear in the context of word X_i
  - For a combination of three words i, j, k. A general representation of the model is

$$F(W_i, W_j, \widetilde{W}_k) = \frac{P_{ik}}{P_{jk}}$$

  - The optimization function proposed by authors are:

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

GloVe: Optimization function (continued)

- Where V is the size of vocabulary and W_i^T and b_i is the vector and bias of the word W_i and $\widetilde{W}_j$ and b_j is the context vector and its bias. The last term is the probability of occurring i in the context of j.

- The function $f(X)$ should have following properties:
  - It tends to zero at when $X \rightarrow 0$
  - It should be non-decreasing so that it can discriminate rare co-occurrence instances.
  - It should not overweight frequent co-occurrence.
- The choice of $f(X)$ is

$$f(X) = (X/X_{max})^{\alpha} \quad if\ X < X_{max}\ otherwise\ 1$$

GloVe: Optimization function (continued)

- The model has following computational bottlenecks:
  - Creating a big co-occurrence matrix of size $V\ X\ V$.
  - The model computational complexity depends on the number of non-zero elements.
- During training the context window needs to be sufficiently large so that the model can distinguish left context and right context.
- Words which are more distant to each other contribute less in the count. Because, distant words contribute less to the relationship of the words.
- The model generates two set of $W\ and\ \widetilde{W}$. An average of both is used as the representation of words.

GloVe: Training

- The prediction problem is given by:

$$w_i^T \cdot \tilde{w}_j + b_i + \tilde{b}_j = \log X_{i,j}$$

  - b_w and b_c are bias terms.
- The objective function:

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

  - $f(X_{i,j})$ is a weighting function to penalize rare co-occurrences.

- The model generates two sets of word vectors, W and $\tilde{W}$.
- W and $\tilde{W}$ are equivalent and differ only as a result of their random initializations.
  - The two sets of vectors should perform equivalently.
- Authors proposed to use $\frac{W+\tilde{W}}{2}$ to get word vectors.

GloVe: Training

- Training: Learn two vectors for each word, such that we have a multinomial logistic regression of the the co-ocurrence probability

$$w_i \cdot \tilde{w}_j \approx \log P(j|i)$$

- We can solve it minimizing :

$$L = \sum_{i,j \in V} \left(w_i \cdot \tilde{w}_j - \log P(j|i) \right)^2$$

- To reduce noise introduced by rare co-occurrences, the loss is actualy weigthed:

$$L = \sum_{i,j \in V} f(X_{ij}) \left(w_i \cdot \tilde{w}_j - \log P(j|i) \right)^2$$

where $f(X_{ij})$ is a weight in [0,1] that grows with the co-occurrence frequency X_{ij}

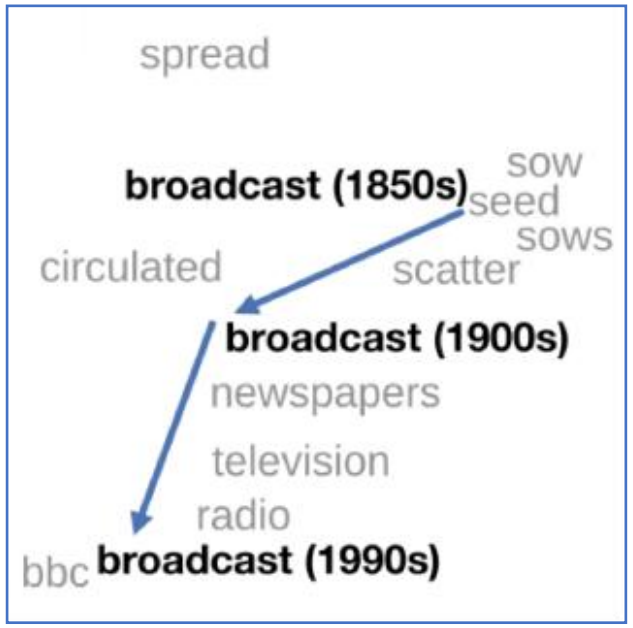
GloVe: Training

- Stochastic gradient descent (SGD) is used to minimize the squared error between the dot product of word vectors and the logarithm of their co-occurrence count.
- Once the model is trained, we have 2 vectors for each word
  - w_i : vector representing word i as the center word
  - $\widetilde{w}_i$: vector representing word i as a context word
- Either one of them (or their sum $w_i + \widetilde{w}_i$) are used as the word representation vector.

Embeddings can help study word history!

❖Visualizing changes

Project 300 dimensions down into 2



Embeddings reflect cultural bias

Bolukbasi, Tolga, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." In *Advances in Neural Information Processing Systems*, pp. 4349-4357. 2016.

Ask "Paris : France :: Tokyo : x"
- x = Japan

Ask "father : doctor :: mother : x"
- x = nurse

Ask "man : computer programmer :: woman : x"
- x = homemaker

Implicit Association test (Greenwald et al 1998): How associated are
- concepts (*flowers*, *insects*) & attributes (*pleasantness*, *unpleasantness*)?
- Studied by measuring timing latencies for categorization.

Psychological findings on US participants:
- African-American names are associated with unpleasant words (more than European-American names)
- Male names associated more with math, female names with arts
- Old people's names with unpleasant words, young people with pleasant words.

Caliskan et al. replication with embeddings:
- African-American names (*Leroy, Shaniqua*) had a higher GloVe cosine with unpleasant words (*abuse, stink, ugly*)
- European American names (*Brad, Greg, Courtney*) had a higher cosine with pleasant words (*love, peace, miracle*)

Embeddings reflect and replicate all sorts of pernicious biases.

Use embeddings as a historical tool to study bias

Are GloVe and Word2vec enough?

- Both the embeddings can not deal with out of vocabulary words.
- Both can capture the context, but in a limited sense.
  - They always produce single embedding for the word in cosideration.
  - They can't distinguish:
    - "I went to a **bank.**" and "I was standing at a river **bank.**"
    - It will always produce single representation for both the context.
- Both gives decent performance than encoding like tf-idf, count vector etc.
- Does pretrained model helps the case?
  - Pretrained models on huge corpus shows better performance compared to small corpus.
  - Pretrained models of Word2vec[2] is available from Google and GloVe[1] is available on Stanford's website.

1. https://nlp.stanford.edu/projects/glove/
2. https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit?usp=sharing

References

1) Efficient Estimation of Word Representations in Vector Space by Mikolov et al (original Word2vec paper)
2) Distributed Representations of Words and Phrases and their Compositionality by Mikolov et al (Original Skipgram paper)
3) What is Word2Vec? A Simple Explanation
4) Word Embedding basics: https://medium.com/data-science-group-iitr/word-embedding-2d05d270b285
5) What is Word2Vec? A Simple Explanation | Deep Learning Tutorial 41 (Tensorflow, Keras & Python)
6) https://machinelearningmastery.com/what-are-word-embeddings/
7) https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa
8) https://towardsdatascience.com/word2vec-made-easy-139a31a4b8ae
9) Lecture 2 | Word Vector Representations: word2vec: https://www.youtube.com/watch?v=ERibwqs9p38
10) https://www.cip.ifi.lmu.de/~nie/Files/handout/fasttext_handout.pdf
11) https://www.cs.upc.edu/~padro/ahlt/lectures/05-WordEmbeddings.pdf

Continuous Bag Of Words Model

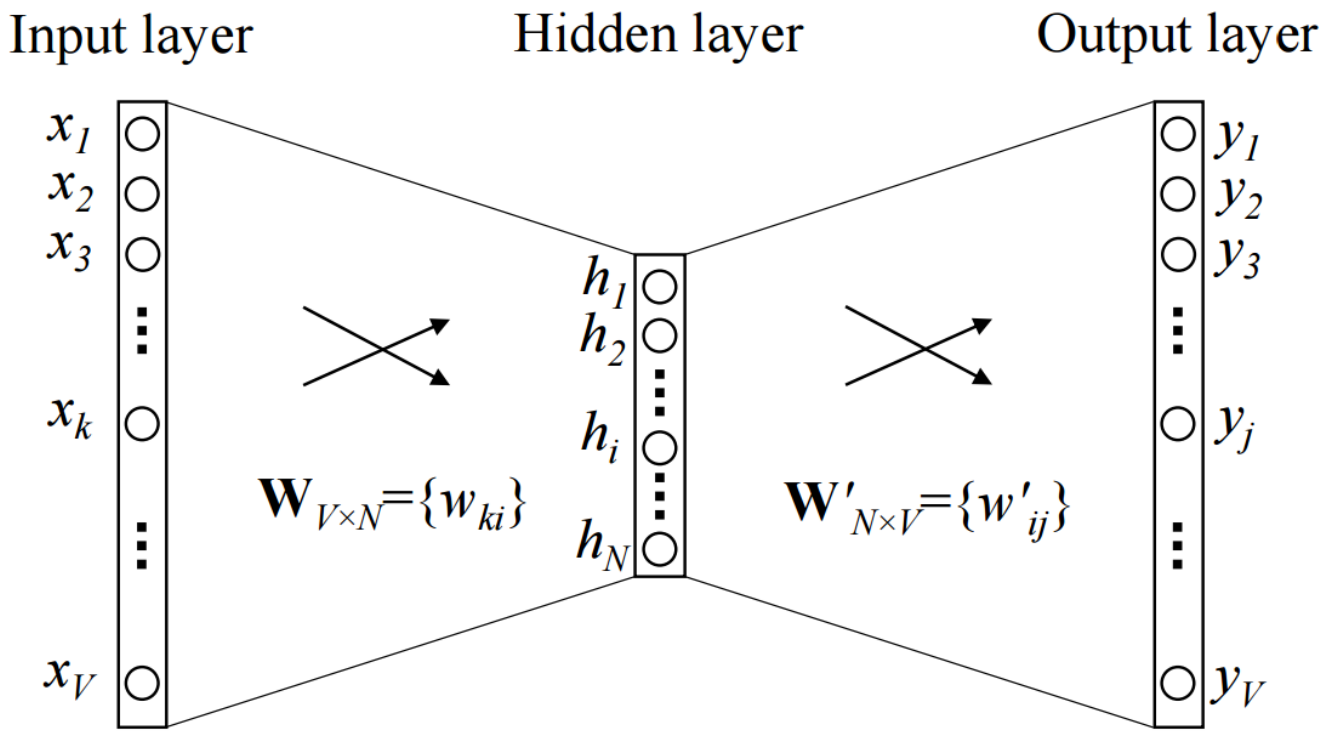


Figure 1: A simple CBOW model with only one word in the context