Text Representation

Dr. Muneendra Ojha

Assistant Professor

Department of Information Technology

Indian Institute of Information Technology - Allahabad

Terminologies

Document

- A document is a single text data point, e.g. a review of a particular product by the user.

Corpus

- It is a collection of all the documents present in our dataset.

Feature

- Every unique word in the corpus is considered a feature.

Terminologies

Consider the 2 documents shown below:

- Sentences:
  1. Dog hates a cat. It loves to go out and play.
  2. Cat loves to play with a ball.

We can build a corpus from the above 2 documents just by combining them.

- Corpus = “Dog hates a cat. It loves to go out and play. Cat loves to play with a ball.”

And features will be all unique words:

- Features: [‘and’, ‘ball’, ‘cat’, ‘dog’, ‘go’, ‘hates’, ‘it’, ‘loves’, ‘out’, ‘play’, ‘to’, ‘with’]

Text Representation

Question:

- how do we transform a given text into numerical form so that it can be fed into NLP and ML algorithms?

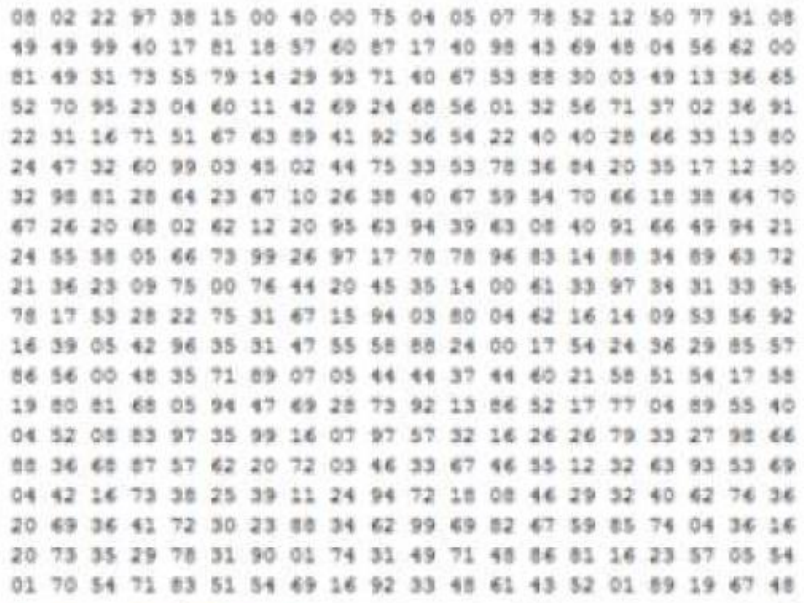
Answer:

- In NLP parlance, this conversion of raw text to a suitable numerical form is called text representation

How do we represent images?

- Image is stored in a computer in the form of a matrix of pixels. Each cell[i,j] of matrix represents pixel i,j of the image. Cell[i,j] stores the intensity of the corresponding pixel in the image.

What We See

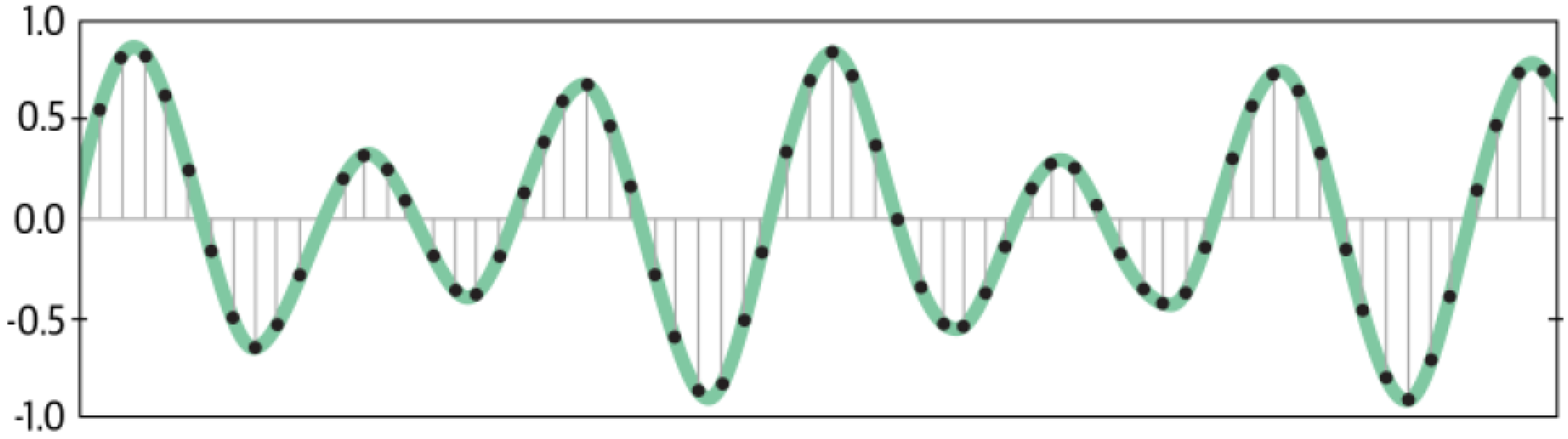


08	02	22	97	38	15	00	40	00	75	04	05	07	78	52	12	50	77	91	08
49	49	99	40	17	81	18	57	60	87	17	40	98	43	69	48	04	56	62	00
81	49	31	73	55	79	14	29	93	71	40	67	53	88	30	03	49	13	36	65
52	70	95	23	04	60	11	42	69	24	68	56	01	32	56	71	37	02	36	91
22	31	16	71	51	67	63	89	41	92	36	54	22	40	40	28	66	33	13	80
24	47	32	60	99	03	45	02	44	75	33	53	78	36	84	20	35	17	12	50
32	98	81	28	64	23	67	10	26	38	40	67	59	54	70	66	18	38	64	70
67	26	20	68	02	62	12	20	95	63	94	39	63	08	40	91	66	49	94	21
24	55	58	05	66	73	99	26	97	17	78	78	96	83	14	88	34	89	63	72
21	36	23	09	75	00	76	44	20	45	35	14	00	61	33	97	34	31	33	95
78	17	53	28	22	75	31	67	15	94	03	80	04	62	16	14	09	53	56	92
16	39	05	42	96	35	31	47	55	58	88	24	00	17	54	24	36	29	85	57
86	56	00	48	35	71	89	07	05	44	44	37	44	60	21	58	51	54	17	58
19	80	81	68	05	94	47	69	28	73	92	13	86	52	17	77	04	89	55	40
04	52	08	83	97	35	99	16	07	97	57	32	16	26	26	79	33	27	98	66
88	36	68	87	57	62	20	72	03	46	33	67	46	55	12	32	63	93	53	69
04	42	16	73	38	25	39	11	24	94	72	18	08	46	29	32	40	62	76	36
20	69	36	41	72	30	23	88	34	62	99	69	82	67	59	85	74	04	36	16
20	73	35	29	78	31	90	01	74	31	49	71	48	86	81	16	23	57	05	54
01	70	54	71	83	51	54	69	16	92	33	48	61	43	52	01	89	19	67	48

What Computers See

How do we represent speech?

- To represent it mathematically, we sample the wave and record its amplitude (height).



How do we represent speech?

- This gives us a numerical array representing the amplitude of a sound wave at fixed time intervals

```
[-1274, -1252, -1160, -986, -792, -692, -614, -429, -286, -134, -57, -41,
-169, -456, -450, -541, -761, -1067, -1231, -1047, -952, -645, -489, -448,
-397, -212, 193, 114, -17, -110, 128, 261, 198, 390, 461, 772, 948, 1451,
1974, 2624, 3793, 4968, 5939, 6057, 6581, 7302, 7640, 7223, 6119, 5461,
4820, 4353, 3611, 2740, 2004, 1349, 1178, 1085, 901, 301, -262, -499,
-488, -707, -1406, -1997, -2377, -2494, -2605, -2675, -2627, -2500, -2148,
-1648, -970, -364, 13, 260, 494, 788, 1011, 938, 717, 507, 323, 324, 325,
350, 103, -113, 64, 176, 93, -249, -461, -606, -909, -1159, -1307, -1544]
```

How do we represent Text?

- To find a scheme to represent text mathematically is called text representation.
- Text representation approaches are classified into four categories:
    - Basic vectorization approaches
    - Distributed representations
    - Universal language representation
    - Handcrafted features

A sample task: Sentiment Analysis

- To correctly predict the sentiment of a sentence, we **need to understand the meaning** of the sentence.
- To extract the meaning of a sentence, we:
  1. Break the sentence into lexical units such as lexemes, words, and phrases
  2. Derive the meaning for each of the lexical units
  3. Understand the syntactic (grammatical) structure of the sentence
  4. Understand the context in which the sentence appears

Vector Space Models

- The representation of text units (characters, phonemes, words, phrases, sentences, paragraphs, and documents) with vectors of numbers is known as the ***Vector Space Model (VSM)***.
- It's a mathematical model that **represents text units as vectors**.
- In this setting, the most common way to calculate similarity between two text blobs is using **cosine similarity**:
  - Cosine of the angle between their corresponding vectors.

Cosine Similarity

- Given two vectors, A and B, each with n components, the similarity between them is computed as follows:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|_2 \|\mathbf{B}\|_2} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

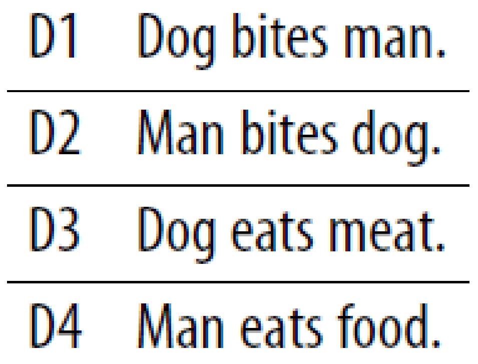
  where A_i and B_i are the i^{th} components of vectors A and B,

- The cosine of 0° is 1 and the cosine of 180° is – 1, with the cosine monotonically decreasing from 0° to 180°.

Vector Space Models

- All the text representation schemes we’ll study fall within the scope of vector space models.
- What differentiates one scheme from another is how well the resulting vector captures the linguistic properties of the text it represents.

Basic Representations



D1	Dog bites man.
D2	Man bites dog.
D3	Dog eats meat.
D4	Man eats food.

Our toy corpus

- four documents—D_1, D_2, D_3, D_4
- Lowercasing text and ignoring punctuation, the vocabulary of this corpus is comprised of six words:

[dog, bites, man, eats, meat, food]

- We can organize the vocabulary in any order.
- In this example, we simply take the order in which the words appear in the corpus.
- We'll assume that the text is already pre-processed and tokenized

One-Hot Encoding

- In one-hot encoding, each word w in the corpus vocabulary is given a unique integer ID wid that is between 1 and $|V|$, where V is the set of the corpus vocabulary.
- Each word is then represented by a V-dimensional binary vector of $0s$ and $1s$.
- This is done via a $|V|$ dimension vector filled with all $0s$ except the index, where $index \ = \ wid$.

One 1, the rest 0’s

One-Hot Encoding

D1	Dog bites man.
D2	Man bites dog.
D3	Dog eats meat.
D4	Man eats food.

- In One-Hot Encoding:
  - We first map each of the six words to unique IDs:
  - dog = 1, bites = 2, man = 3, meat = 4 , food = 5, eats = 6.
- Each word is a six-dimensional vector
  - Dog is represented as [1 0 0 0 0 0]
  - Bites is represented as [0 1 0 0 0 0]

One 1, the rest 0's

One-Hot Encoding: Pros

- one-hot encoding is intuitive to understand
- straightforward to implement

One-Hot Encoding: Cons

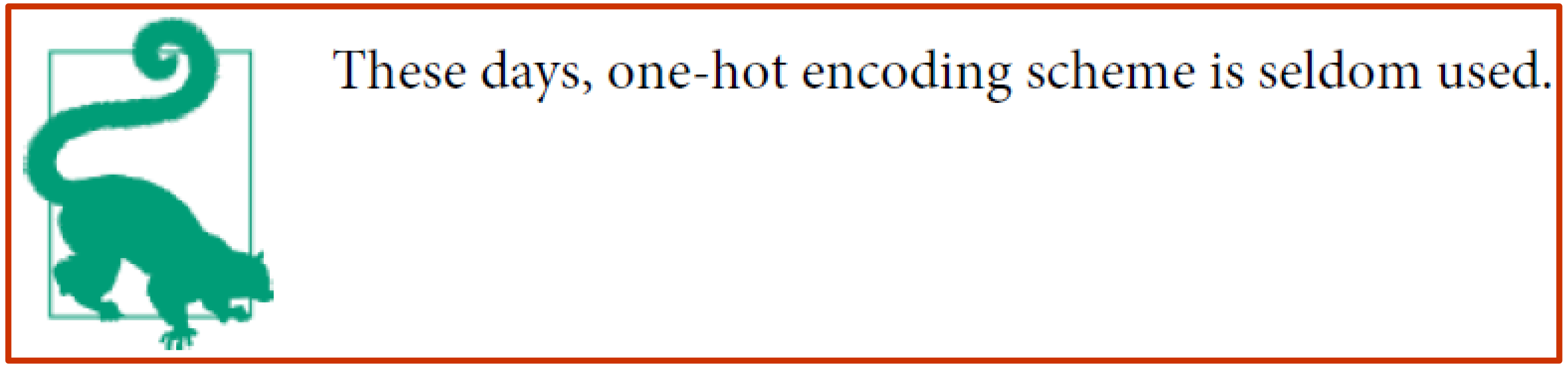
- size of a one-hot vector is directly proportional to size of the vocabulary
- results in a sparse representation (most of the entries in the vectors are zeroes)
- computationally inefficient to store, compute with, and learn from (sparsity leads to overfitting).

One-Hot Encoding: Cons

- This representation does not give a fixed-length representation for text, i.e., if a text has 10 words, you get a longer representation for it as compared to a text with 5 words.
- It treats words as atomic units and has no notion of (dis)similarity between words
- Consider three words: run, ran, and apple.
  - run and ran have similar meanings as opposed to run and apple.
  - However, Euclidean distance between all equally apart ($\sqrt{2}$).
  - Thus, semantically, they're very poor at capturing the meaning of the word in relation to other words.

One-Hot Encoding: Cons

- Say we train a model using our toy corpus. At runtime, we get a sentence: **“man eats fruits.”**
- The training data didn’t include “fruit” and there’s no way to represent it in our model.
- This is known as the **Out Of Vocabulary (OOV)** problem.


These days, one-hot encoding scheme is seldom used.

Bag of Words (BoW)

- Bag of words (BoW) is a classical text representation technique that has been used commonly in NLP, especially in text classification problems:

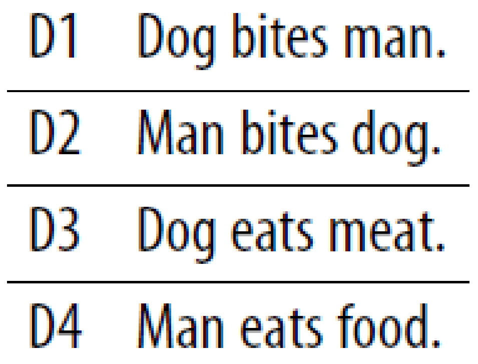
  represent the text under consideration as a bag (collection) of words while ignoring the order and context

- The basic intuition behind it is that it assumes that the text belonging to a given class in the dataset is characterized by a unique set of words.

Bag of Words (BoW)

- Similar to one-hot encoding, BoW maps words to unique integer IDs between 1 and $|V|$.
- Each document in the corpus is then converted into a vector of $|V|$ dimensions where in the i^{th} component of the vector, $i = w_{id}$, is simply the number of times the word w occurs in the document, i.e., we simply score each word in V by their occurrence count in the document.

Bag of Words (BoW)



D1	Dog bites man.
D2	Man bites dog.
D3	Dog eats meat.
D4	Man eats food.

- For our toy corpus, where the word IDs are dog = 1, bites = 2, man = 3, meat = 4 , food = 5, eats = 6,
- D_1 becomes [1 1 1 0 0 0]
- D_4 becomes [0 0 1 0 1 1]

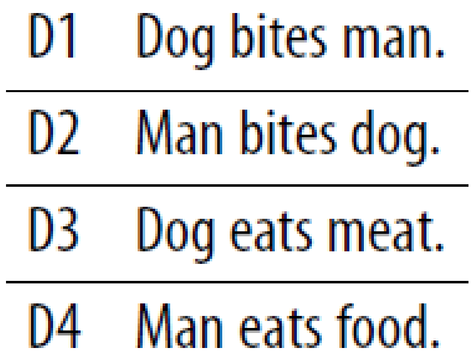
Bag of Words (BoW): Pros

- BoW is fairly simple to understand and implement.
- Documents having the same words will have their vector representations closer to each other in Euclidean space as compared to documents with completely different words.
- The distance between D_1 and D_2 is 0 as compared to the distance between D_1 and D_4, which is 2.
- Thus, the vector space resulting from the BoW scheme captures the semantic similarity of documents.
  - So if two documents have similar vocabulary, they'll be closer to each other in the vector space and vice versa.
- We have a fixed-length encoding for any sentence of arbitrary length.

Bag of Words (BoW): Cons

- The size of the vector increases with the size of the vocabulary.
  - Sparsity continues to be a problem.
- It does not capture the similarity between different words that mean the same thing. Say we have three documents: “I run”, “I ran”, and “I ate”.
  - BoW vectors of all three documents will be equally apart.

Bag of Words (BoW): Cons



D1	Dog bites man.
D2	Man bites dog.
D3	Dog eats meat.
D4	Man eats food.

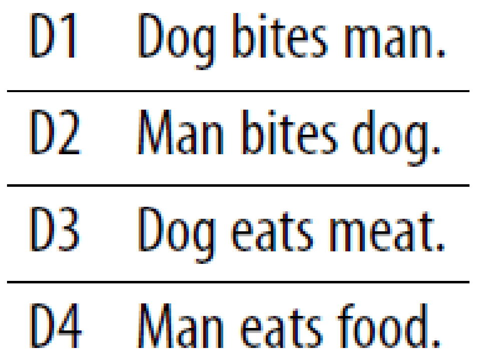
- This representation does not have any way to handle Out Of Vocabulary (OOV) words (i.e., new words that were not seen in the corpus that was used to build the vectorizer).
- As the name indicates, it is a “bag” of words—word order information is lost in this representation.
- Both D_1 and D_2 will have the same representation in this scheme.

BoW is a commonly used text representation scheme, especially for text classification among other NLP problems

Bag of N-Grams (BoN)

- All the representation schemes we’ve seen so far treat words as independent units. There is no notion of word ordering.
- The bag-of-n-grams (BoN) approach tries to remedy this.
- It does so by breaking text into **chunks of n contiguous words (or tokens) called an n-gram**
- This can help us capture some context.
- The corpus vocabulary, V, is a collection of all unique n-grams.
- Then, each document in the corpus is represented by a vector of length |V|.
- This vector simply contains the frequency counts of n-grams present in the document and zero for the n-grams that are not present.

Bag of N-Grams (BoN)



D1	Dog bites man.
D2	Man bites dog.
D3	Dog eats meat.
D4	Man eats food.

- The set of all bigrams in the corpus is as follows: {dog bites, bites man, man bites, bites dog, dog eats, eats meat, man eats, eats food}.
- Then, BoN representation consists of an eight-dimensional vector for each document.
- The bigram representation for the first two documents is as follows: D_1 : [1,1,0,0,0,0,0,0], D_2 : [0,0,1,1,0,0,0,0].

Bag of N-Grams (BoN): pros & cons

- It captures some context and word-order information.
- Resulting vector space is able to capture some semantic similarity.
- Documents having the same n-grams will have their vectors closer to each other in Euclidean space as compared to documents with completely different n-grams.
- As n increases, dimensionality (and therefore sparsity) only increases rapidly.
- It still provides no way to address the OOV problem.

TF-IDF

- In all the three approaches we’ve seen so far, all the words in the text are treated as equally important—there’s no notion of some words in the document being more important than others.
- TF-IDF, or Term Frequency–Inverse Document Frequency, addresses this issue
- It aims to quantify the importance of a given word relative to other words in the document and in the corpus.

TF-IDF

- If a word w appears many times in a documentd_i but does not occur much in the rest of the documents d_j in the corpus, then the word w must be of great importance to the document d_i.
- Mathematically, this is captured using two quantities: TF and IDF.
- The two are then combined to arrive at the $TF - IDF$ score.

Term Frequency (TF)

- TF (term frequency) **measures how often a term** or word **occurs in a document**.
- Since different documents in the corpus may be of different lengths, a term may occur more often in a longer document as compared to a shorter document.
- To normalize these counts, we divide the number of occurrences by the length of the document.
- TF of a term t in a document d is defined as:

$$\mathrm{TF}\left(t, d\right) = \frac{(\text{Number of occurrences of term } t \text{ in document } d)}{(\text{Total number of terms in the document } d)}$$

Inverse Document Frequency (IDF)

- IDF (inverse document frequency) measures the importance of the term across a corpus.
- In computing TF, all terms are given equal importance (weightage).
- However, stop words like is, are, am, etc., are not important, even though they occur frequently.
- To account for such cases, **IDF weighs down** the very **common terms** across a corpus and **weighs up** the **rare terms**.

Inverse Document Frequency (IDF)

- IDF of a term t is calculated as follows:

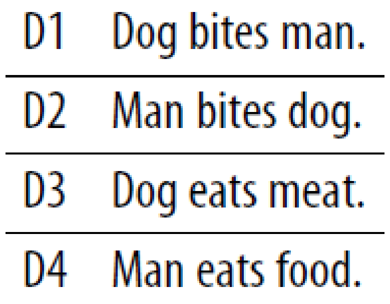
$$\mathrm{IDF}(t) = \log_e \frac{(\text{Total number of documents in the corpus})}{(\text{Number of documents with term } t \text{ in them })}$$

- The TF-IDF score is a product of these two terms.
- Thus,

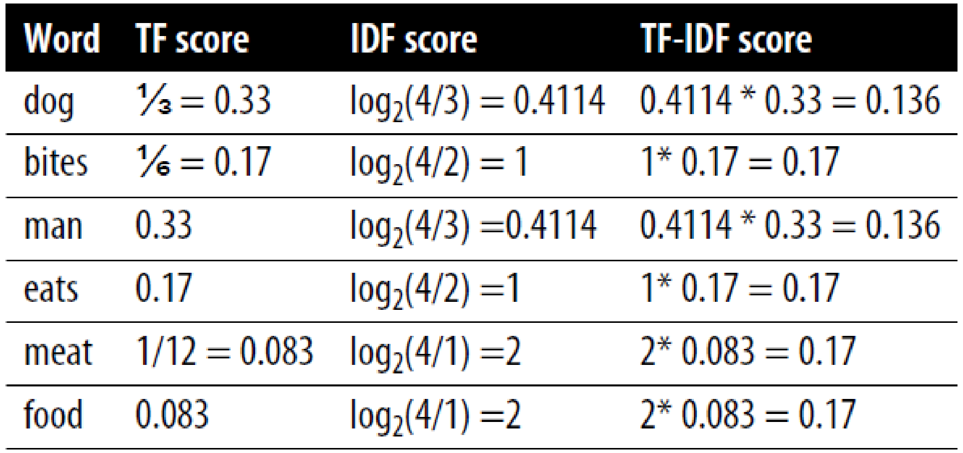
$$TF - IDF \; = TF * IDF$$

TF-IDF for toy example

- Some terms appear in only one document, some appear in two, and others appear in three documents.
- The size of our corpus is N=4.

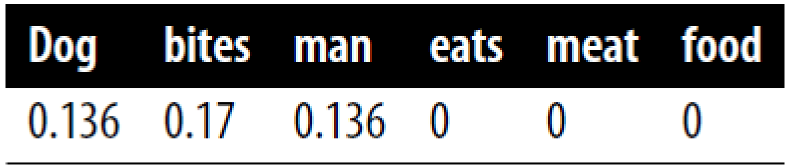


D1	Dog bites man.
D2	Man bites dog.
D3	Dog eats meat.
D4	Man eats food.



Word	TF score	IDF score	TF-IDF score
dog	⅓ = 0.33	$\log_2(4/3) = 0.4114$	0.4114 * 0.33 = 0.136
bites	⅙ = 0.17	$\log_2(4/2) = 1$	1* 0.17 = 0.17
man	0.33	$\log_2(4/3) =0.4114$	0.4114 * 0.33 = 0.136
eats	0.17	$\log_2(4/2) =1$	1* 0.17 = 0.17
meat	1/12 = 0.083	$\log_2(4/1) =2$	2* 0.083 = 0.17
food	0.083	$\log_2(4/1) =2$	2* 0.083 = 0.17

- The TF-IDF vector representation for a document is then simply the TF-IDF score for each term in that document.
- So, for D1 we get:



Dog	bites	man	eats	meat	food
0.136	0.17	0.136	0	0	0

TF-IDF for toy example

- **For Example,** In any corpus, few words like **‘is’** or **‘and’** are very common, and most likely, they will be present in almost every document.
- Suppose the word ‘is’ is present in all the documents in a corpus of 1000 documents.
- The idf for that would be:

$$\log \frac{1000}{1000} = \log 1 = 0$$

- Thus common words would have lesser importance.

TF-IDF

- TF-IDF vectors can be used to calculate similarity between two texts using a similarity measure like Euclidean distance or cosine similarity
- TF-IDF is a commonly used representation in application scenarios such as information retrieval and text classification.
- Despite the fact that TF-IDF is better than the vectorization methods, it still suffers from the curse of high dimensionality.

all schemes discussed so far

- They’re discrete representations — They treat language units (words, n-grams, etc.) as atomic units.
  - This discreteness hampers their ability to capture relationships between words.
- The feature vectors are sparse and high-dimensional representations.
  - The dimensionality increases with the size of the vocabulary, with most values being zero for any vector.
  - This hampers learning capability.
  - High-dimensionality representation makes them computationally inefficient.
- They cannot handle OOV words.

Some key terms…

Distributional similarity

- The idea is that the meaning of a word can be understood from the context in which the word appears.
  - also known as **connotation**: meaning is defined by context.
  - opposed to **denotation**: the literal meaning of any word.
- For example: “NLP rocks.”
  - The literal meaning of the word “rocks” is “stones,” but from the context, it’s used to refer to something good and fashionable.

Distributional hypothesis

- In linguistics, this hypothesizes that words that occur in similar contexts have similar meanings.
    - For example, the English words “dog” and “cat” occur in similar contexts.
    - Thus, according to the distributional hypothesis, there must be a strong similarity between the meanings of these two words.
- Now, following from VSM, the meaning of a word is represented by the vector.
- Thus, if two words often occur in similar context, then their corresponding representation vectors must also be close to each other.

Distributional representation

- This refers to representation schemes that are obtained based on distribution of words from the context in which the words appear.
- These schemes are based on distributional hypotheses.
- The distributional property is induced from context (textual vicinity).
- Mathematically, distributional representation schemes use high-dimensional vectors to represent words.
- These vectors are obtained from a co-occurrence matrix that captures co-occurrence of word and context.
- The dimension of this matrix is equal to the size of the vocabulary of the corpus.
- The four schemes that we’ve seen so far—one-hot, bag of words, bag of n-grams, and TF-IDF—all fall under the umbrella of distributional representation.

Distributed representation

- As discussed, the vectors in distributional representation are very high dimensional and sparse.
  - This makes them computationally inefficient and hampers learning.
- To alleviate this, distributed representation schemes significantly compress the dimensionality.
- This results in vectors that are compact (i.e., low dimensional) and dense (i.e., hardly any zeros).
- The resulting vector space is known as distributed representation.

Embedding

- For the set of words in a corpus, embedding is a mapping between vector space coming from distributional representation to vector space coming from distributed representation.

Vector semantics

- This refers to the set of NLP methods that aim to learn the word representations based on distributional properties of words in a large corpus.

Word Embeddings

Word2Vec

- In 2013, a seminal work by Mikolov et al. [7] showed that their neural network–based word representation model known as "Word2vec," based on "distributional similarity," can capture word analogy relationships such as:

$$\text{King} - \text{Man} + \text{Woman} \approx \text{Queen}$$

- Their model was able to correctly answer many more analogies like this.
- The Word2vec model is in many ways the dawn of modern-day NLP

Word2Vec

- Word2vec takes a large corpus of text as input and “learns” to represent the words in a common vector space based on the contexts in which they appear in the corpus.
- For every word w in corpus, we start with a vector v_w initialized with random values.
- The Word2vec model refines the values in v_w by predicting v_w, given the vectors for words in the context C.

Pre-trained word embeddings

- Training your own word embeddings is a pretty expensive process. Thus we use pre-trained word embeddings.
- **Pre-trained word embeddings**: Someone has done the hard work of training word embeddings on a large corpus, such as Wikipedia, news articles, or even the entire web, and has put words and their corresponding vectors on the web.
- These embeddings can be downloaded and used.
- Such embeddings can be thought of as a large collection of key-value pairs, where keys are the words in the vocabulary and values are their corresponding word vectors.
- Some of the most popular pre-trained embeddings are **Word2vec by Google, GloVe by Stanford, and fasttext embeddings by Facebook**
- They're available for various dimensions like d = 25, 50, 100, 200, 300, 600.

Training our own embeddings

- For this, we'll look at two architectural variants that were proposed in the original Word2vec approach.
- The two variants are:
  - Continuous bag of words (CBOW)
  - SkipGram

What is a language model?

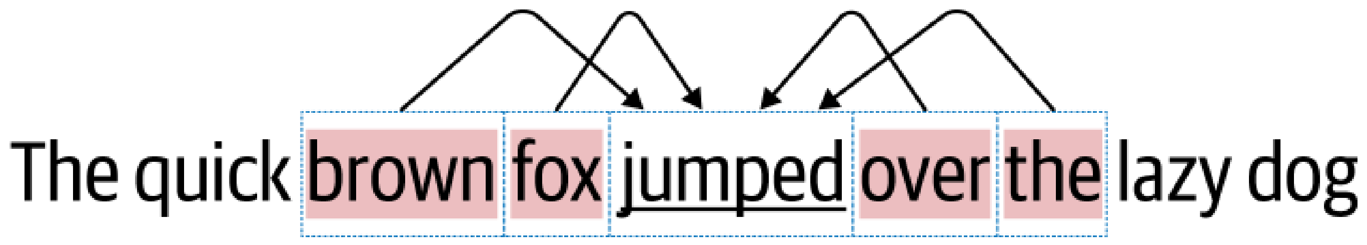
- It is a (statistical) model that tries to give a probability distribution over sequences of words.
- Given a sentence of n words, it assigns a probability $\Pr(w_1, w_2, \ldots, w_n)$ to the whole sentence.
- The objective of a language model is to assign high probabilities to “good” sentences and low probabilities to “bad” sentences.
  - By good, we mean sentences that are semantically and syntactically correct.
  - By bad, we mean sentences that are incorrect— semantically or syntactically or both.
- So, for a sentence like “The cat jumped over the dog,” it will try to assign a probability close to 1.0, whereas for a sentence like “jumped over the the cat dog,” it tries to assign a probability close to 0.0.

Continuous bag of words (CBOW)

- The primary task is to build a language model that correctly predicts the center word given the context words in which the center word appears.

Continuous bag of words (CBOW)

- Take the word “jumped” as the center word, then its context is formed by words in its vicinity.
- For a context size of 2, imagine a sliding window over the text, that includes the central word currently in focus, together with the two words that precede it, and the two words that follow it:

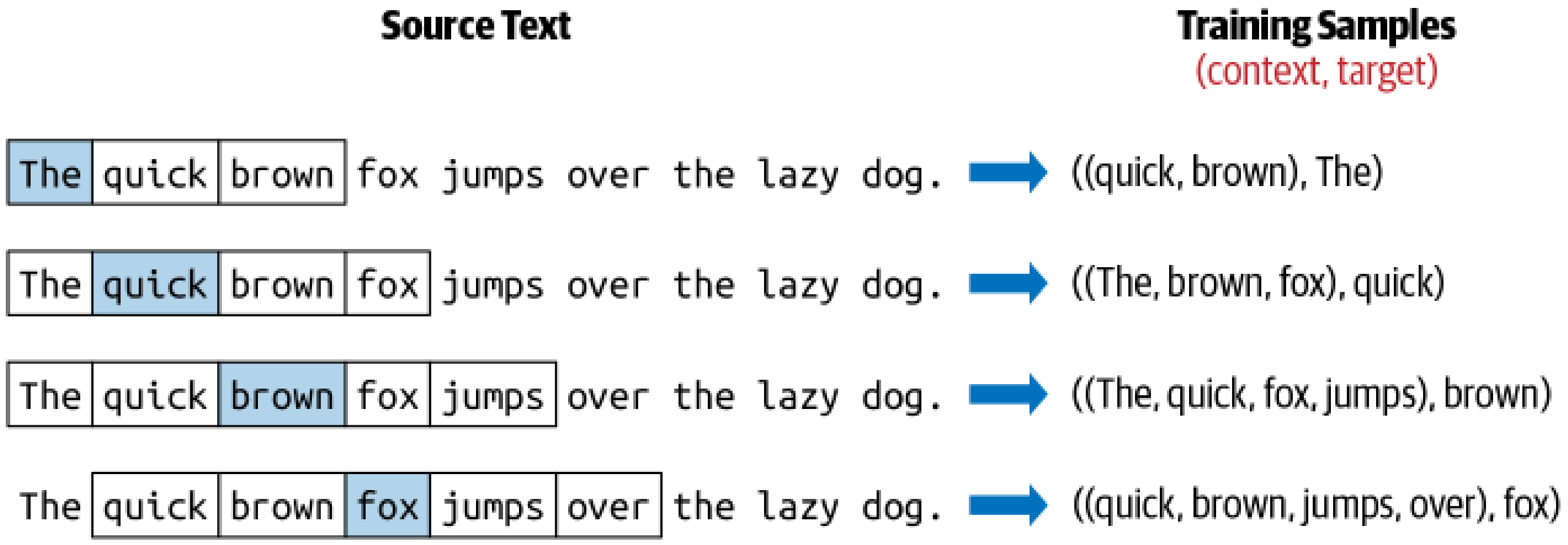


Continuous bag of words (CBOW)

- The idea is then extended to the entire corpus to build the training set.
  - We run a sliding window of size 2k+1 over the text corpus.
  - For our example, we took k as 2.
  - Each position of the window marks the set of 2k+1 words that are under consideration.
  - The center word in the window is the target, and k words on either side of the center word form the context.
- This gives us one data point. If the point is represented as (X,Y), then the context is the X and the target word is the Y.

Continuous bag of words (CBOW)

- A single data point consists of a pair of numbers: (2k indices of words in context, index of word in target).

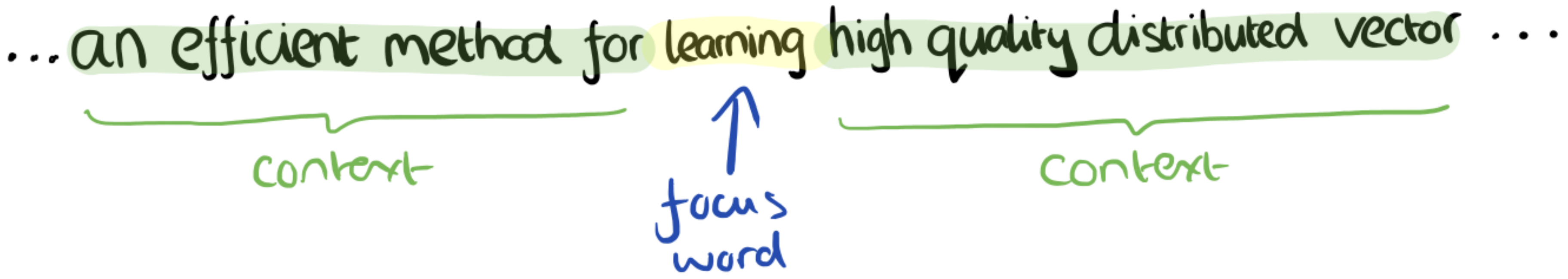

Source Text
Training Samples
(context, target)
The quick brown fox jumps over the lazy dog.
((quick, brown), The)
The quick brown fox jumps over the lazy dog.
((The, brown, fox), quick)
The quick brown fox jumps over the lazy dog.
((The, quick, fox, jumps), brown)
The quick brown fox jumps over the lazy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

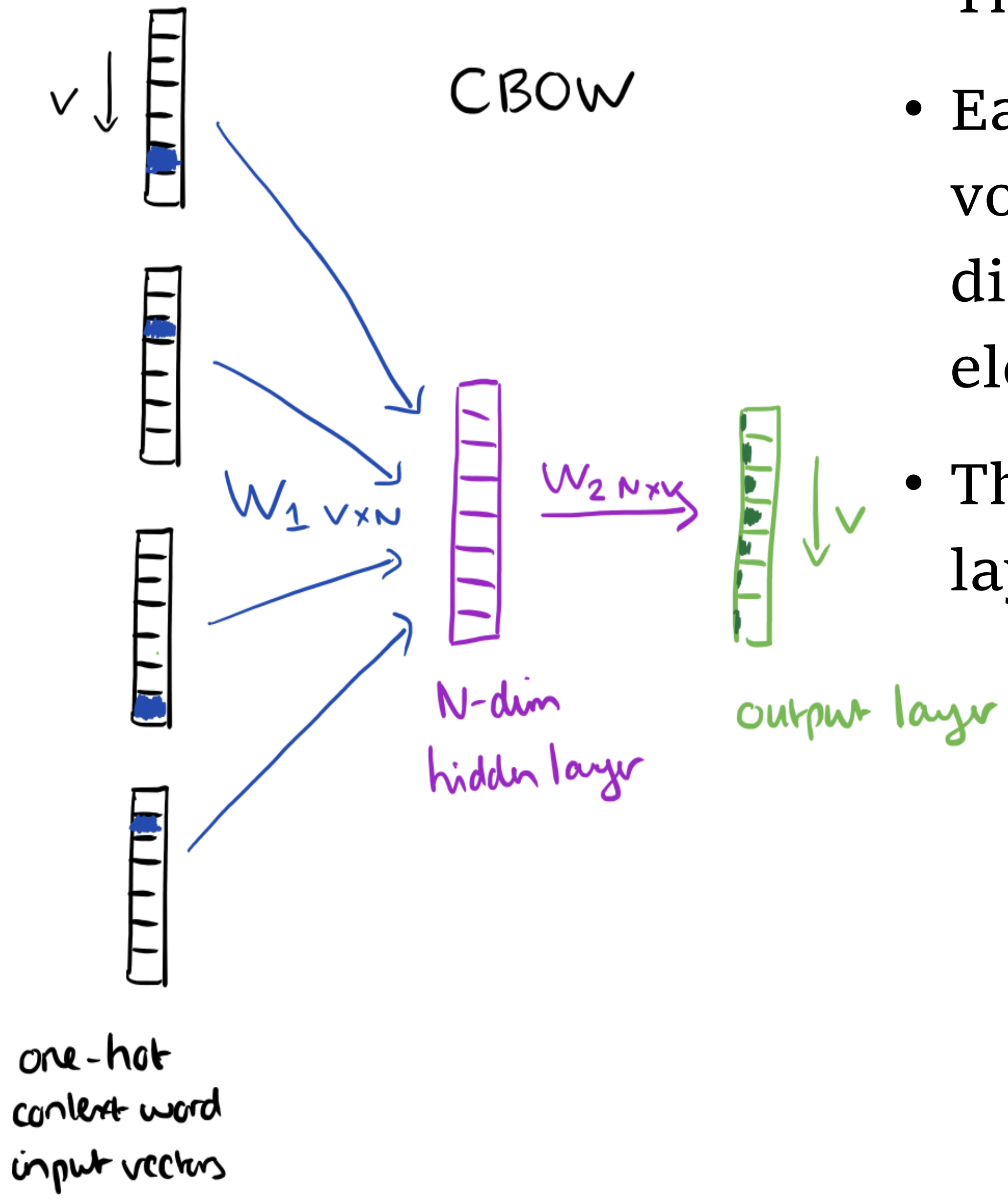
Continuous bag of words (CBOW)

- Let V be the vocabulary of the text corpus.
- Assume we want to learn D-dim word embeddings.
- Construct a shallow neural net
  - The context words form the input layer.
  - Each word is encoded in one-hot form, so if the vocabulary size is V these will be V-dimensional vectors with just one of the elements set to one, and the rest all zeros.
  - There is a single hidden layer and an output layer.

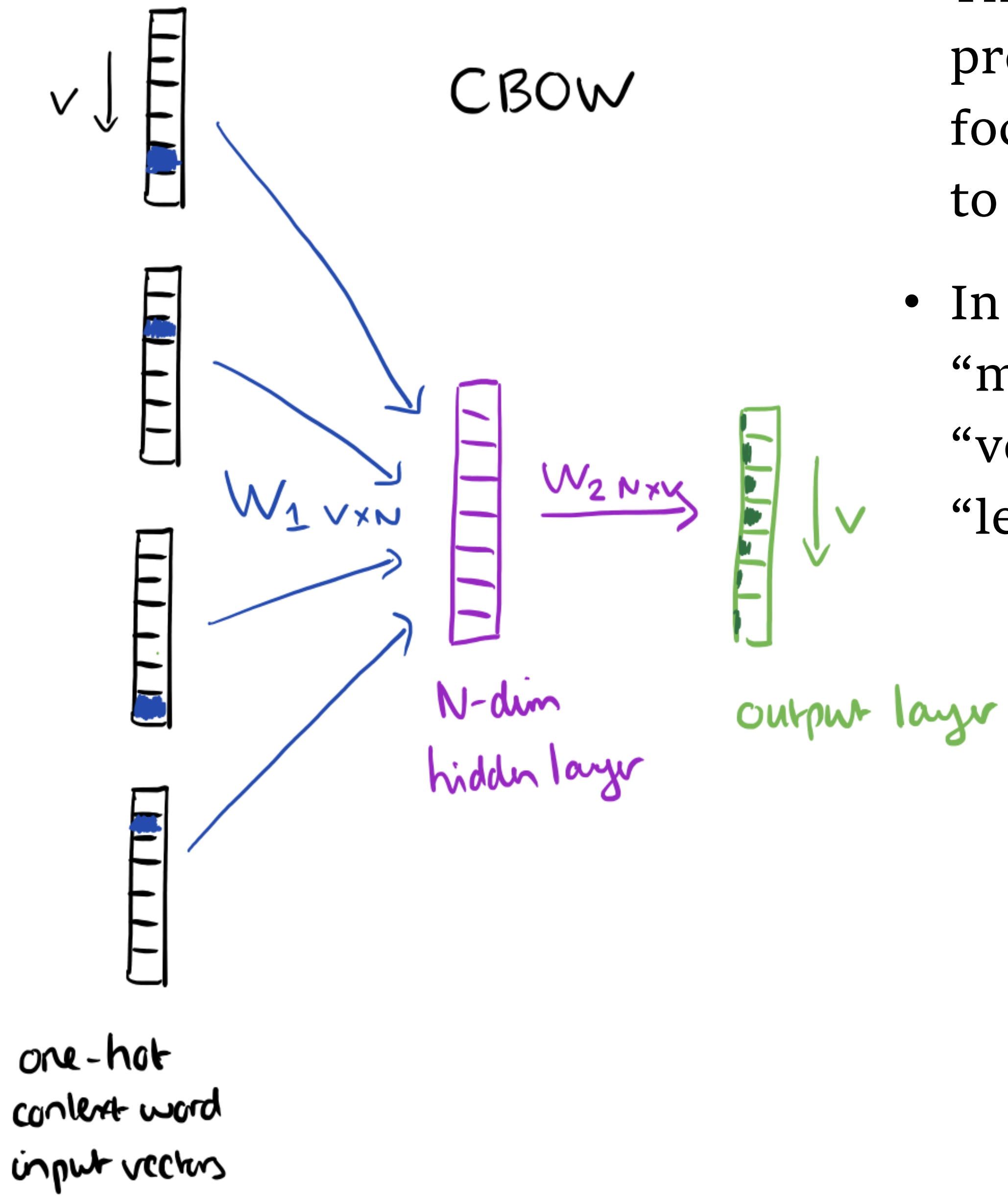
Continuous bag of words (CBOW)

- Consider a text: “The recently introduced continuous Skip-gram model is an efficient method for learning high-quality distributed vector representations that capture a large number of precise syntactic and semantic word relationships.”
- With the central word in focus with four words preceding it and the four words following it:


...an efficient method for learning high quality distributed vector ...
context
focus word
context

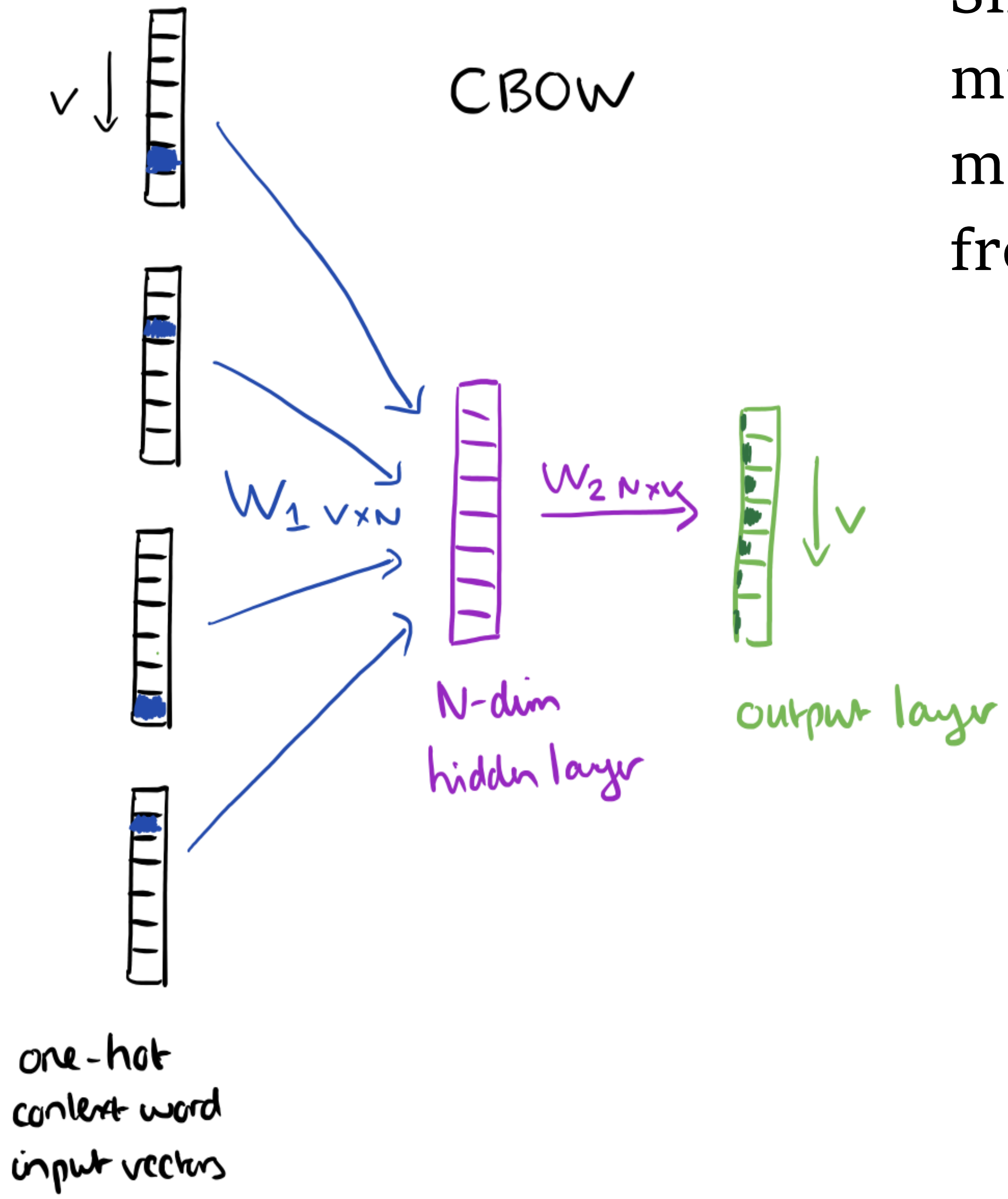


- The context words form the input layer.
- Each word is encoded in one-hot form, so if the vocabulary size is V these will be V-dimensional vectors with just one of the elements set to one, and the rest all zeros.
- There is a single hidden layer and an output layer.



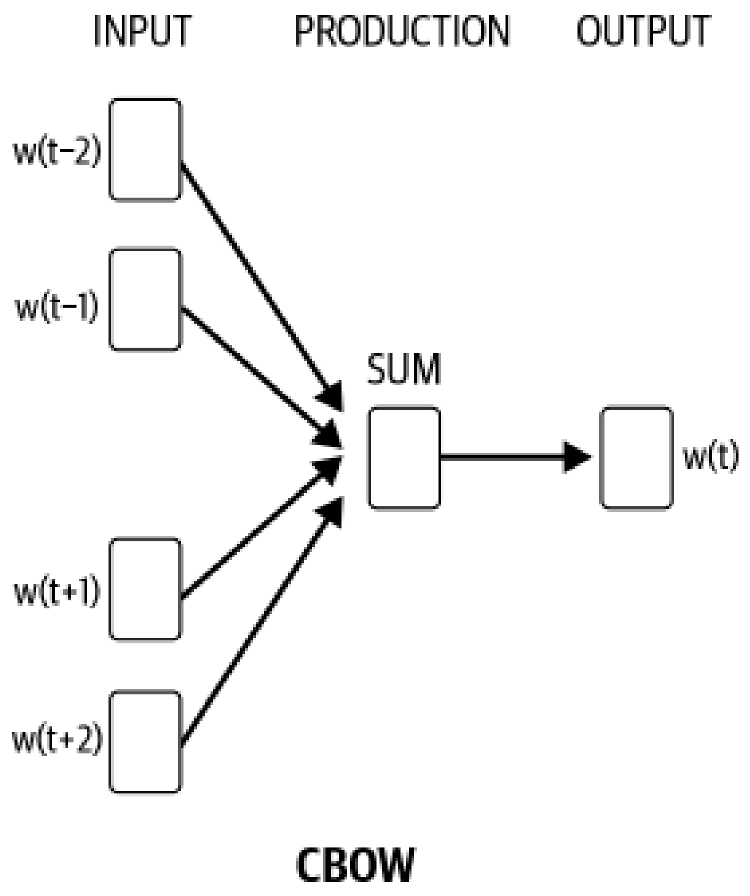
- The training objective is to maximize the conditional probability of observing the actual output word (the focus word) given the input context words, with regard to the weights.
- In our example, given the input (“an”, “efficient”, “method”, “for”, “high”, “quality”, “distributed”, “vector”) we want to maximize the probability of getting “learning” as the output.

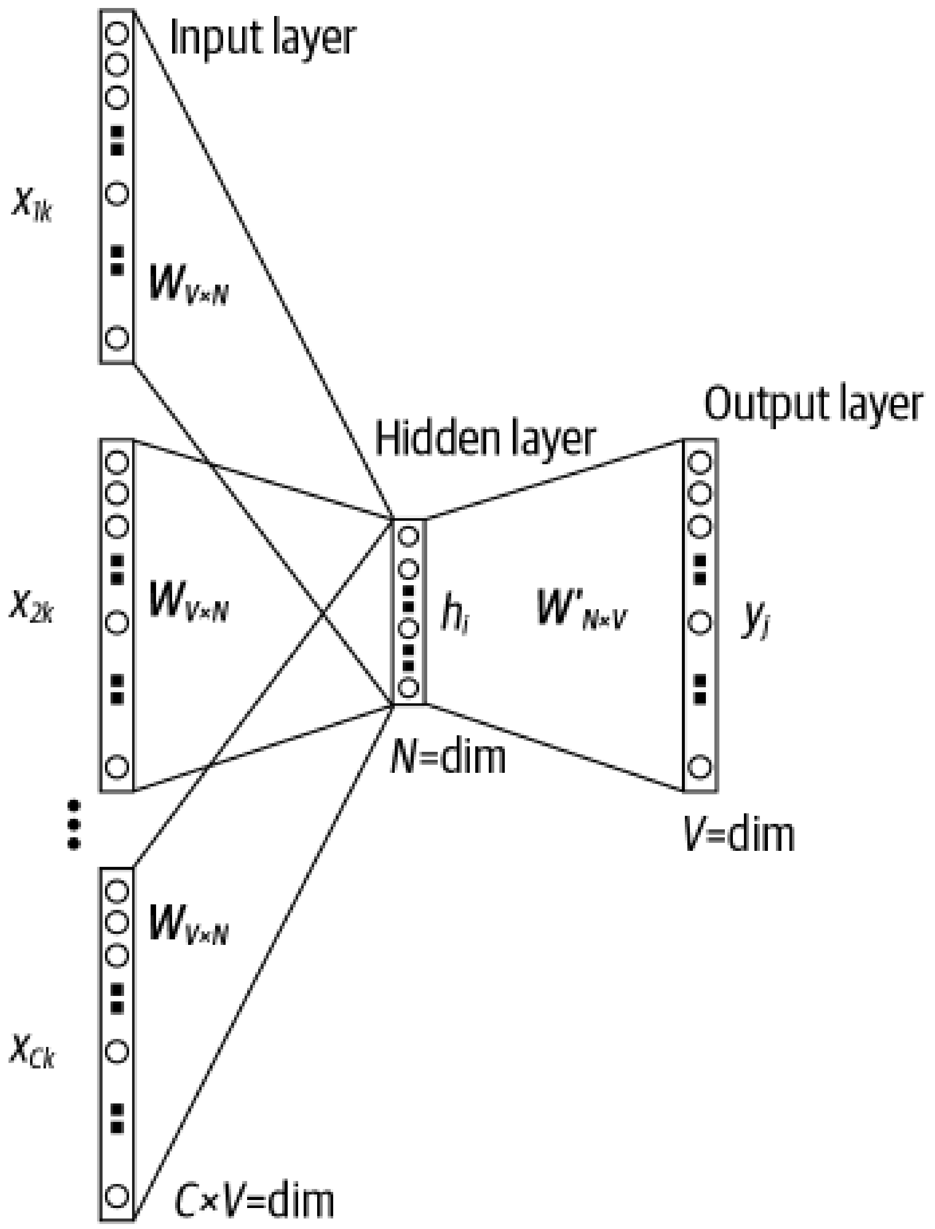
- Since our input vectors are one-hot, multiplying an input vector by the weight matrix W_1 amounts to simply selecting a row from W_1.

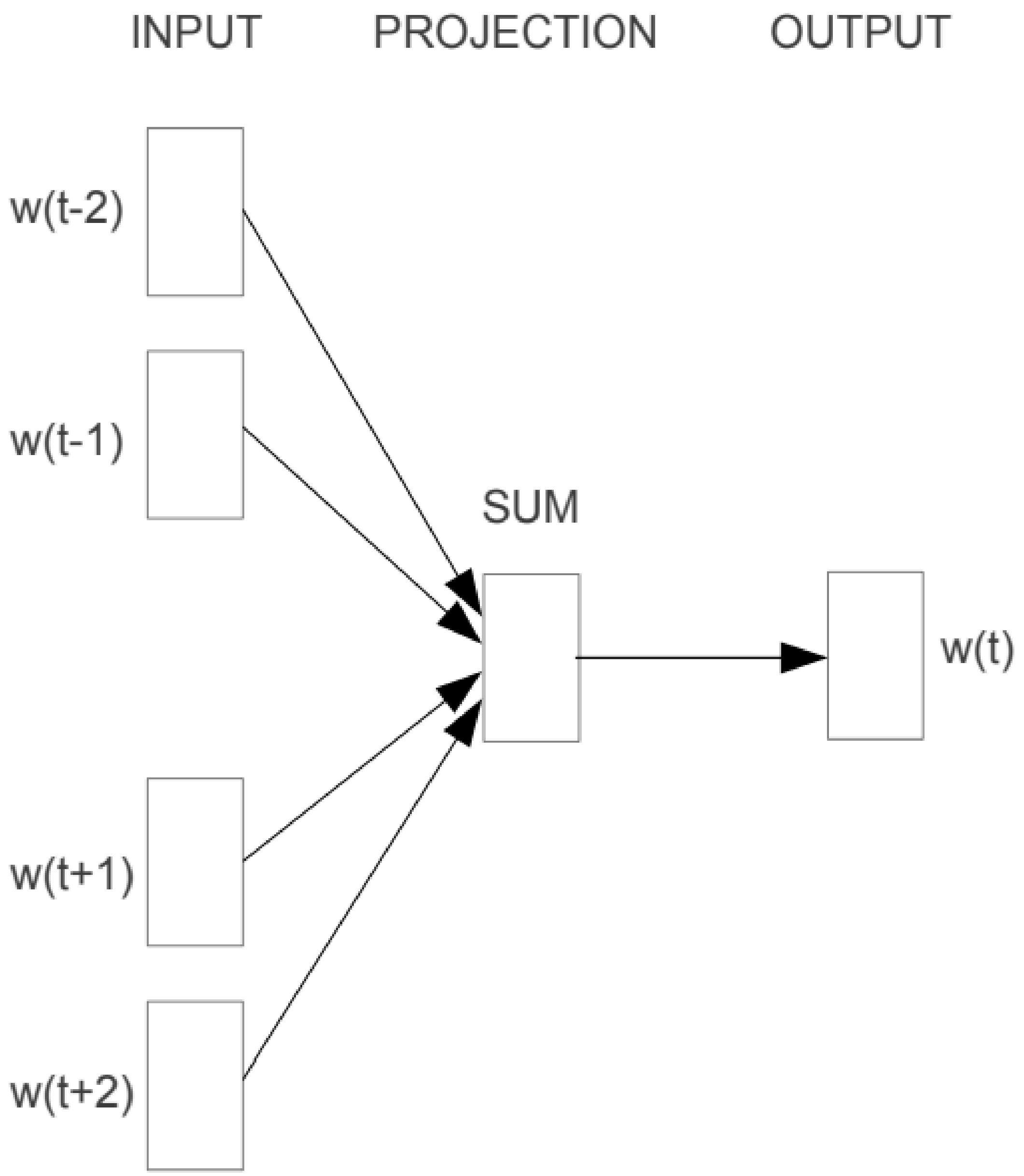


$$\underset{\substack{\text{input} \\ 1\times V}}{\begin{bmatrix} 0 & 1 & 0 \end{bmatrix}} \underset{W_1}{\overset{\substack{W_1 \\ V\times N}}{\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{bmatrix}}} = \overset{\substack{\text{hidden} \\ 1\times N}}{\begin{bmatrix} e & f & g & h \end{bmatrix}}$$

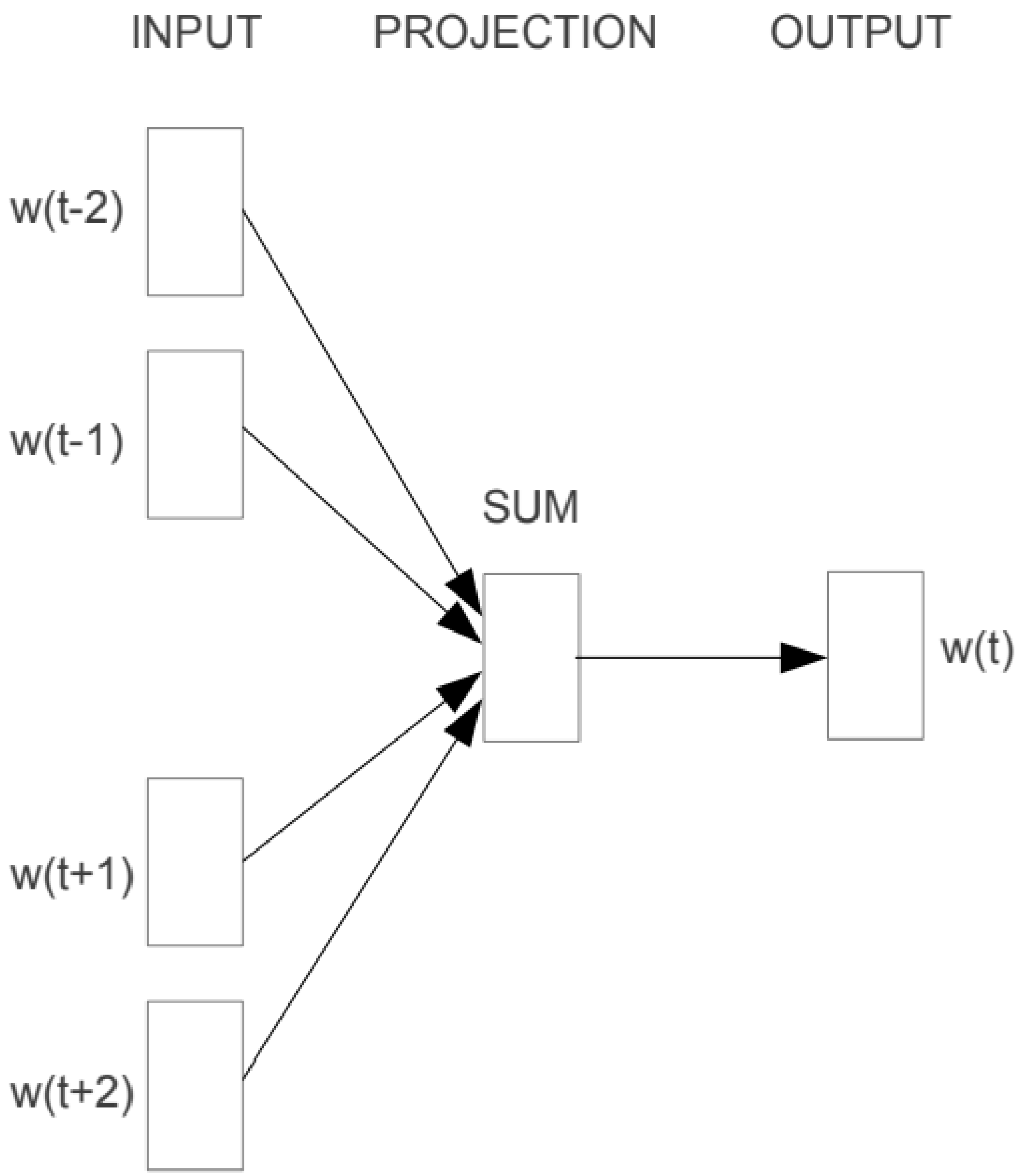
CBOW Model


INPUT
PRODUCTION
OUTPUT
w(t-2)
w(t-1)
SUM
w(t)
w(t+1)
w(t+2)
CBOW


Input layer
x_{1k}
$W_{V \times N}$
Hidden layer
Output layer
x_{2k}
$W_{V \times N}$
h_i
$W'_{N \times V}$
y_j
N=dim
V=dim
$W_{V \times N}$
x_{Ck}
$C \times V$=dim

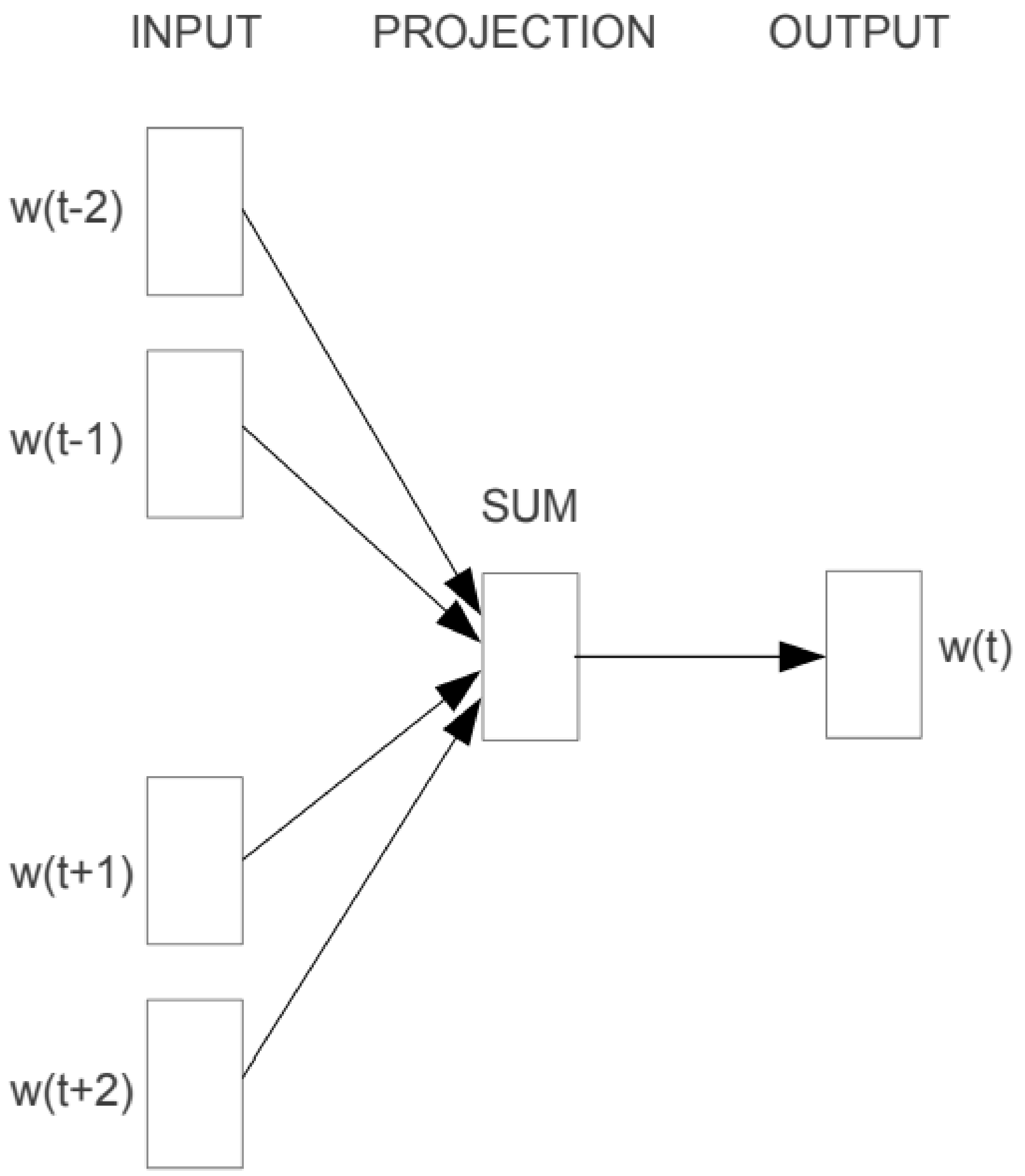


- The objective is to learn an embedding matrix $E_{|V| \times d}$.
- To begin with, we initialize the matrix randomly.
- Here, $|V|$ is the size of corpus vocabulary and d is the dimension of the embedding.



- In the input layer, indices* of the words in context are used to fetch the corresponding rows from the embedding matrix $E_{|V| \times d}$.
- The vectors fetched are then added to get a single d-dim vector, and this is passed to the next layer.

**As per logic in slide 59*



- The next layer simply takes this d vector and multiplies it with another matrix $E'_{d \times |V|}$
- This gives a $1 \times |V|$ vector, which is fed to a softmax function to get probability distribution over the vocabulary space.
- This distribution is compared with the label and uses backpropagation to update both the matrices E and E' accordingly.
- Finally, E is the embedding matrix we wanted to learn.

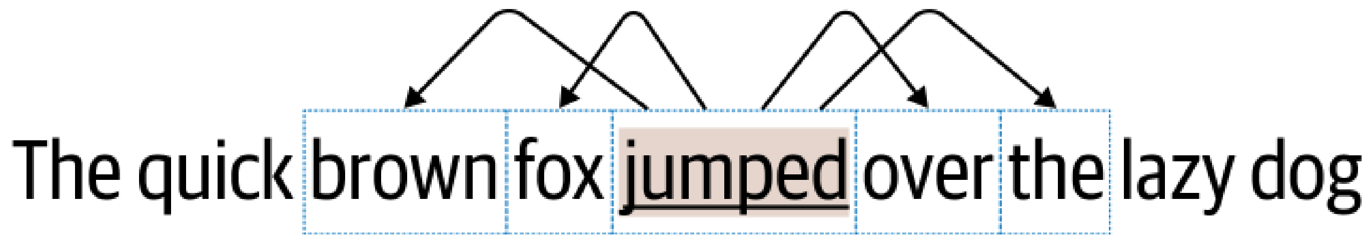
**As per logic in slide 59*

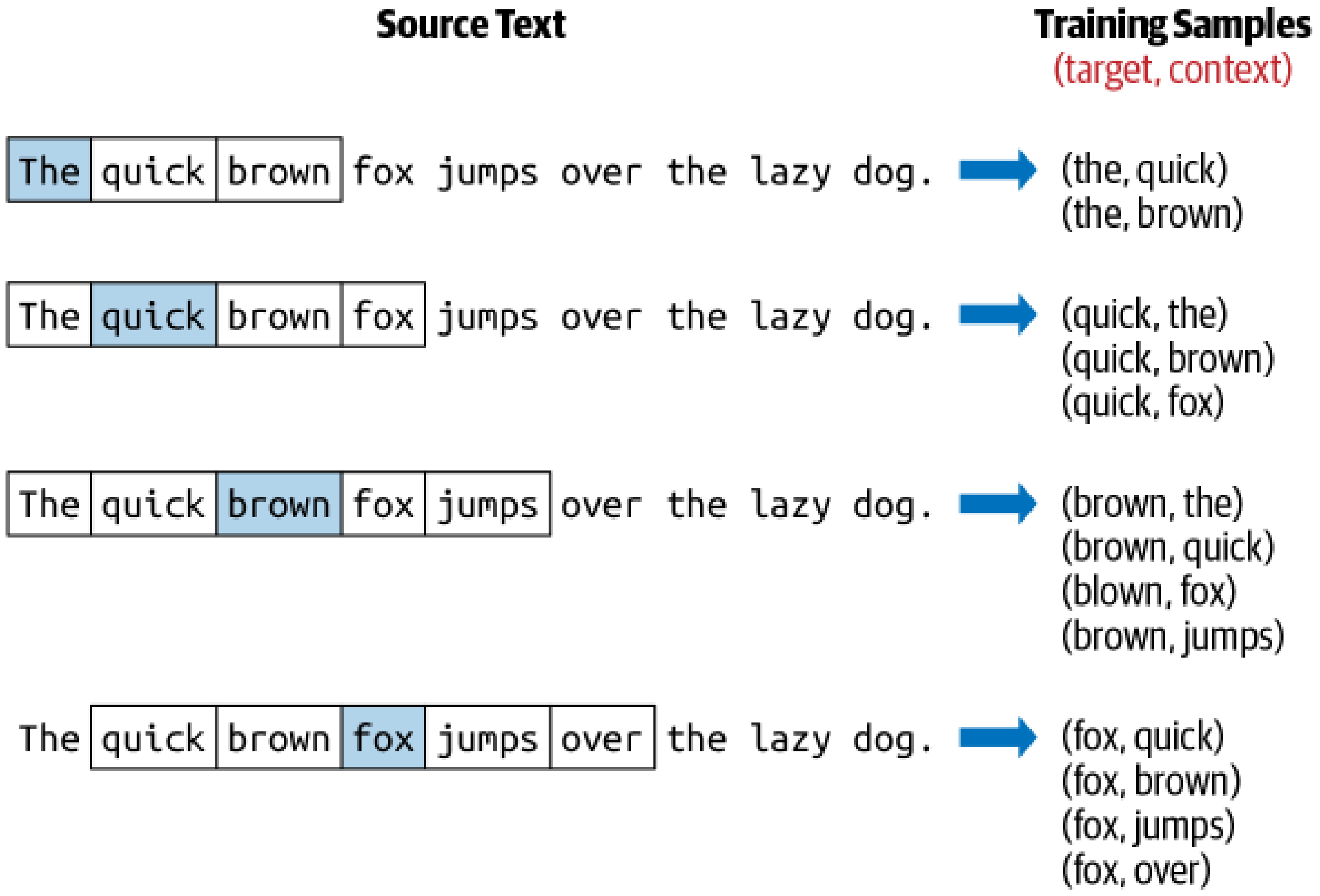
SkipGram

- SkipGram is very similar to CBOW, with some minor changes.
- In SkipGram, the task is to predict the context words from the center word.
- For example: For our toy corpus with context size 2, using the center word “jumps,” we try to predict every word in context—“brown,” “fox,” “over,” “the”

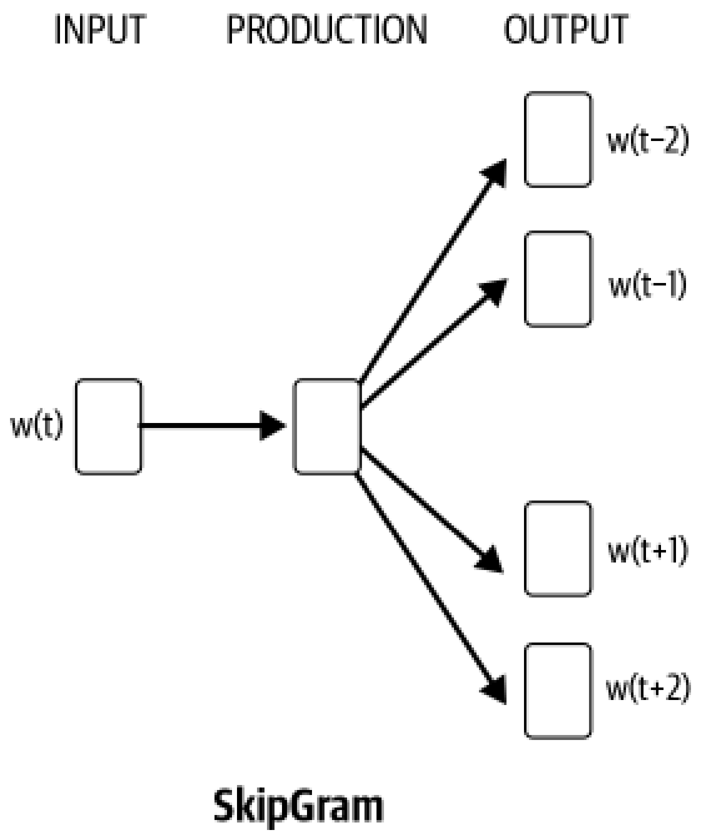
SkipGram

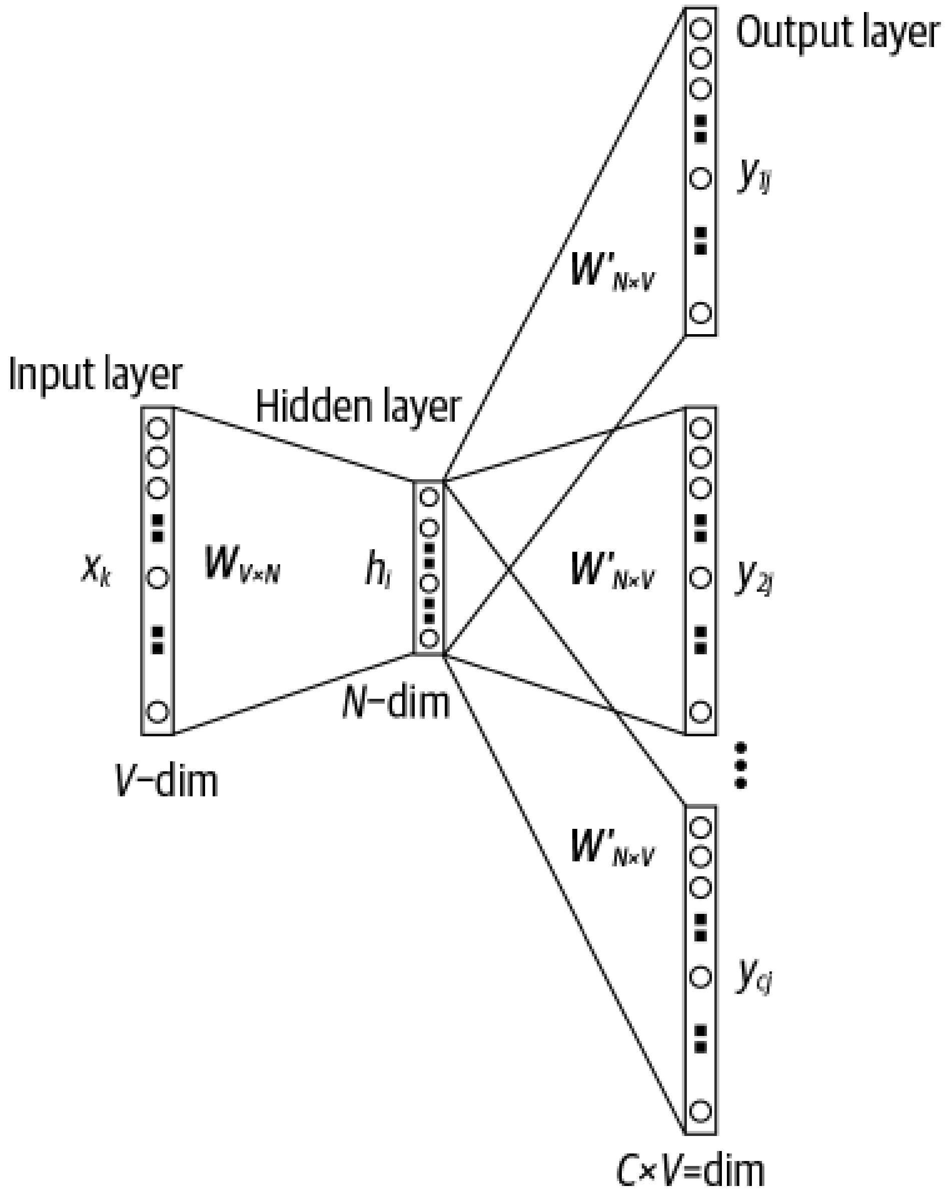
- The dataset to train a SkipGram is prepared as follows: we run a sliding window of size $2k + 1$ over the text corpus to get the set of $2k + 1$ words that are under consideration.
- The center word in the window is the X, and k words on either side of the center word are Y.
- Unlike CBOW, this gives us $2k$ data points. A single data point consists of a pair: (index of the center word, index of a target word). We then shift the window to the right on the corpus by one word and repeat the process.
- This way, we slide the window across the entire corpus to create the training set.

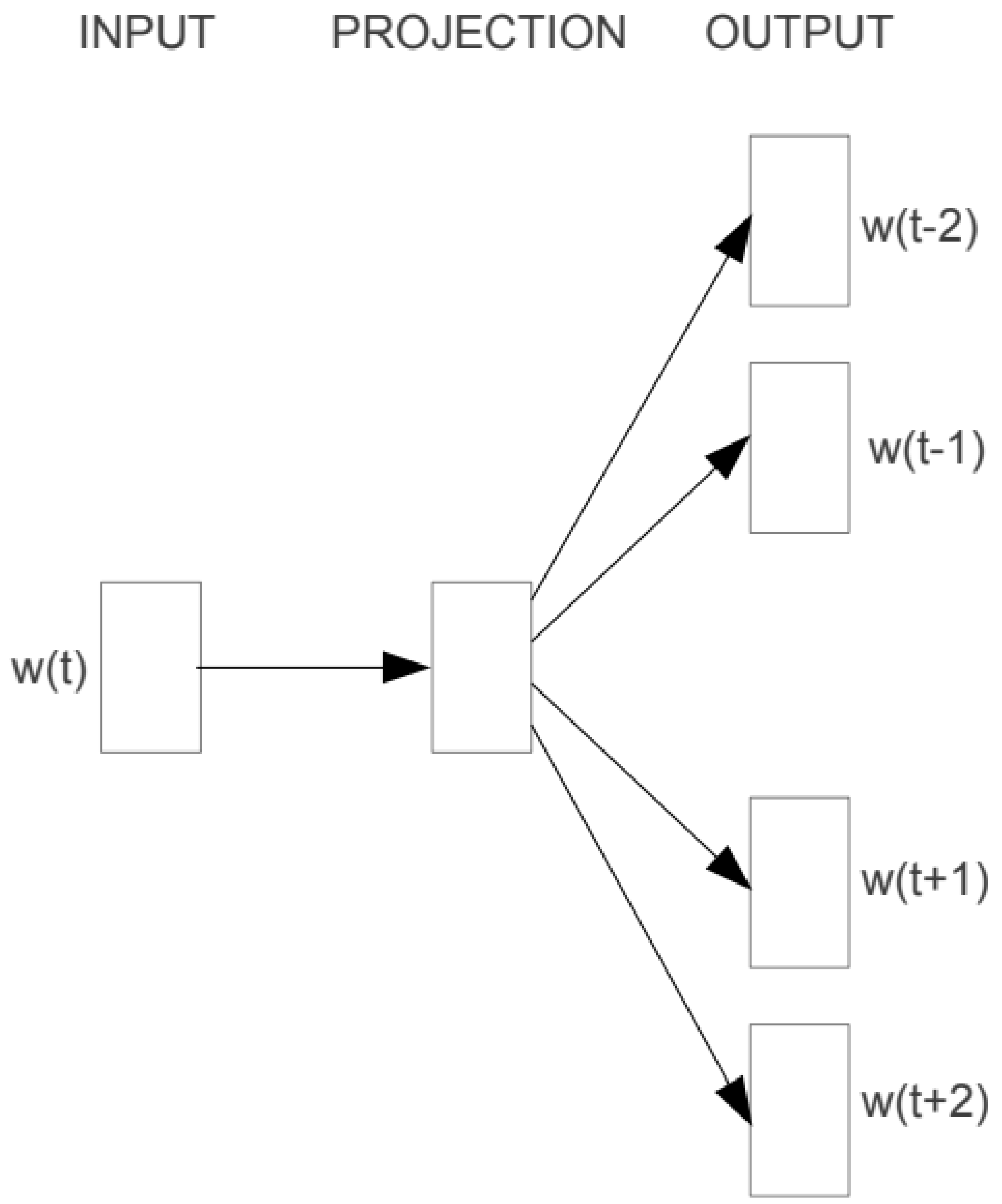

The quick brown fox jumped over the lazy dog


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

Continuous SkipGram Model







- In the input layer, the index of the word in the target is used to fetch the corresponding row from the embedding matrix $E_{|V| \times d}$.
- The vectors fetched are then passed to the next layer.
- The next layer simply takes this d vector and multiplies it with another matrix $E'_{d \times |V|}$.
- This gives a $1 \times |V|$ vector, which is fed to a softmax function to get probability distribution over the vocabulary space.
- This distribution is compared with the label and backpropagation is used to update both the matrices E and E' accordingly.
- At the end of the training, E is the embedding matrix we wanted to learn.

*As per logic in slide 59

GloVe: Global Vectors

- While word2Vec is 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 the co-occurrence counts matrix to yield a lower-dimensional matrix, where each row yields a vector representation for each word.
    - The counts matrix is preprocessed by normalizing the counts and log-smoothing them.

GloVe: Training

- The prediction problem is given by:

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

  - b_w and b_c are bias terms.

- The objective function:

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

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

GloVe: Training

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

Going Beyond Words

- In NLP applications we deal with sentences, paragraphs, or even full texts
- We can use word embeddings to get feature representations for larger units of text?
- A simple approach is to break the text into constituent words, take the embeddings for individual words, and combine them to form the representation for the text.

Going Beyond Words

- There are various ways to combine them, the most popular being sum, average, etc., but these may not capture many aspects of the text as a whole, such as ordering.
- Surprisingly, they work very well in practice
  - In CBOW, this was demonstrated by taking the sum of word vectors in context.
  - The resulting vector represents the entire context and is used to predict the center word.

Solving OOV problem

- Still, we don’t have a good way of handling OOV words.
- A simple approach may be to exclude OOV words from the feature extraction.
  - If we’re using a model trained on a large corpus, we shouldn’t see too many OOV words anyway.
- If a large fraction of words from our production data is not present in the word embedding’s vocabulary, model shows poor performance.

Solving OOV problem

- Vocabulary overlap is a great heuristic to gauge the performance of an NLP model.
- If the overlap between corpus vocabulary and embedding vocabulary is less than 80%, we're unlikely to see good performance from the NLP model.

Solving OOV problem

- Another way to deal with the OOV problem for word embeddings is to create vectors that are initialized randomly, where each component is between –0.25 to +0.25, and continue to use these vectors throughout the application we're building.
- As per literature, this can give us a jump of 1–2% in performance.

fastText from Facebook AI research

- fastText from Facebook AI research handles the OOV problem by using subword information and morphological properties such as prefixes, suffixes, word endings, etc. or by using character representations.
- A word can be represented by its constituent character ngrams.
- Following a similar architecture to Word2vec, fastText learns embeddings for words and character n-grams together and views a word’s embedding vector as an aggregation of its constituent character n-grams.
- This makes it possible to generate embeddings even for words that are not present in the vocabulary.

fastText from Facebook AI research

- Say there’s a word, “gregarious,” that’s not found in the embedding’s word vocabulary.
- We break it into character n-grams—gre, reg, ega, ….ous—and combine these embeddings of the ngrams to arrive at the embedding of “gregarious.”
- gensim’s fastText wrapper can be used both for loading pre-trained models or training models using fastText in a way similar to Word2vec.

Doc2Vev

- Doc2vec is based on the paragraph vectors framework and is implemented in gensim.
- This is similar to Word2vec in terms of its general architecture, except that, in addition to the word vectors, it also learns a “paragraph vector” that learns a representation for the full text (i.e., with words in context).
- When learning with a large corpus of many texts, the paragraph vectors are unique for a given text (where “text” can mean any piece of text of arbitrary length), while word vectors will be shared across all texts.
- The shallow neural networks used to learn Doc2vec embeddings are very similar to the CBOW and SkipGram architecture of Word2vec. The two architectures are called distributed memory (DM) and distributed bag of words (DBOW).

Universal Text Representations

- Words can mean different things in different contexts.
  - “I went to a bank to withdraw money”
  - “I sat by the river bank and pondered about text representations”
- both use the word “bank” but carry different meaning.
- The contextual word representations addresses this issue.

Universal Text Representations

- It uses language modeling:
  - The task of predicting the next likely word in a sequence of words.
- In its earliest form, it used the idea of n-gram frequencies to estimate the probability of the next word given a history of words.
- The advanced neural language models (e.g., transformers) make use of word embeddings and complex architectures involving multiple passes through the text and multiple reads from left to right and right to left to model the context of language use.

Universal Text Representations

- Neural architectures such as RNNs and transformers were used to develop large-scale models of language (ELMo, BERT) that can be used as pre-trained models to get text representations.
- The key idea is to leverage "transfer learning"—that is, to learn embeddings on a generic task (like language modeling) on a massive corpus and then fine-tune learnings on task-specific data.
- These models have shown significant improvements on some fundamental NLP tasks, such as question answering, semantic role labeling, named entity recognition etc.

Key takeaways

- All text representations are inherently biased based on what they saw in training data.
  - It is important to know whether Apple will be found closer to Microsoft or to orange!
  - These biases are an important factor to consider in any NLP software development.
- Pre-trained embeddings are generally large-sized files (Word2vec model takes ~4.5 GB RAM), which may pose problems in certain deployment scenarios.

Key takeaways

- Modeling language for a real-world application is more than capturing the information via word and sentence embeddings.
  - Sarcasm detection requires nuances that are not captured by embedding techniques
- Neural text representation is an evolving area in NLP, with rapidly changing state of the art.

Handcrafted Feature Representations

- Sometimes we need domain-specific knowledge to be incorporated into the model we're building.
- In such cases, we resort to handcrafted features.
- Examples: TextEvaluator.
  - It's a software developed by Educational Testing Service (ETS).

Handcrafted Feature Representations

- The goal of this tool is to help teachers and educators provide support in choosing grade-appropriate reading materials for students and identifying sources of comprehension difficulty in texts.
- This is a very specific problem.
- Having general-purpose word embeddings will not help much.
- It needs specialized features extracted from text to model some form of grade appropriateness.

Handcrafted Feature Representations

- Another software tool from ETS that’s popular for grading is the automated essay scorer used in online exams, such as the Graduate Record Examination (GRE) and Test of English as a Foreign Language (TOEFL), to evaluate test-taker essays.
- Another NLP application where one may need such specialized feature engineering is the spelling and grammar correction we use in tools such as Microsoft Word, Grammarly etc.

References

- Practical Natural Language Processing: A Comprehensive Guide to Building Real-World NLP Systems by Sowmya Vajjala, Bodhisattwa Majumder, Anuj Gupta, and Harshit Surana, O’Reilly Publications.