Introduction to N-grams

Introduction

- N-grams, a concept found in Natural Language Processing (NLP).

- Turns out that is the simplest bit, an N-gram is simply a sequence of N words.

For instance, let us take a look at the following examples.

- San Francisco (is a 2-gram)

- The Three Musketeers (is a 3-gram)

- She stood up slowly (is a 4-gram)

- Now which of these three N-grams have you seen quite frequently?

  “San Francisco” and “The Three Musketeers”.

- On the other hand, it might not have seen “She stood up slowly” that frequently.

- Basically, “She stood up slowly” is an example of an N-gram that does not occur as often in sentences as in Examples 1 and 2.
- Now if we assign a probability to the occurrence of an N-gram or the probability of a word occurring next in a sequence of words, it can be very useful. Why?
- First of all, it can help in deciding which N-grams can be chunked together to form single entities (like “San Francisco” chunked together as one word, “high school” being chunked as one word).

- It can also help make next word predictions.

- If we have the partial sentence “Please hand over your” then it is more likely that the next word is going to be “test” or “assignment” or “paper” than the next word being “school”.
- It can also help to make spelling error corrections.

- For instance, the sentence “drink cofee” could be corrected to “drink coffee” if you knew that the word “coffee” had a high probability of occurrence after the word “drink” and also the overlap of letters between “cofee” and “coffee” is high.

- So assigning these probabilities has a huge potential in the NLP domain.

- Now that we understand this concept, we can build with it: that's the **N-gram model**. Basically, an N-gram model predicts the occurrence of a word based on the occurrence of its N – 1 previous words.
- So here we are answering the question – how far back in the history of a sequence of words should we go to predict the next word? For instance, a bigram model (N = 2) predicts the occurrence of a word given only its previous word (as N – 1 = 1 in this case). Similarly, a trigram model (N = 3) predicts the occurrence of a word based on its previous two words (as N – 1 = 2 in this case).
- Let us see a way to assign a probability to a word occurring next in a sequence of words. First of all, we need a very large sample of English sentences (called **a corpus**).
- For the purpose of our example, we'll consider a very small sample of sentences, but in reality, a corpus will be extremely large. Say our corpus contains the following sentences:
- He said thank you.
- He said bye as he walked through the door.
- He went to San Diego.
- San Diego has nice weather.
- It is raining in San Francisco.

- Let’s assume a bigram model. So we are going to find the probability of a word based only on its previous word.
- In general, we can say that this probability is (the number of times the previous word ‘wp’ occurs before the word ‘wn’) / (the total number of times the previous word ‘wp’ occurs in the corpus) =
- (Count (wp wn))/(Count (wp))
- Let’s work this out with an example.
  To find the probability of the word “you” following the word “thank”, we can write this as P (you | thank) which is a conditional probability.
  This becomes equal to:
- =(No. of times “Thank You” occurs) / (No. of times “Thank” occurs)
- = 1/1
- = 1

- Let’s calculate the probability of the word “Diego” coming after “San”. We want to find the P (Diego | San). This means that we are trying to find the probability that the next word will be “Diego” given the word “San”. We can do this by:
- =(No of times “San Diego” occurs) / (No. of times “San” occurs)
- = 2/3
- = 0.67
- This is because in our corpus, one of the three preceding “San”s was followed by “Francisco”. So, the P (Francisco | San) = 1 / 3.
- In our corpus, only “Diego” and “Francisco” occur after “San” with the probabilities 2 / 3 and 1 / 3 respectively. So if we want to create a next word prediction software based on our corpus, and a user types in “San”, we will give two options: “Diego” ranked most likely and “Francisco” ranked less likely.

Word Vector Model

Word Representation

In general, words are treated as atomic symbols.

motel [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0] AND
hotel [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0] = 0

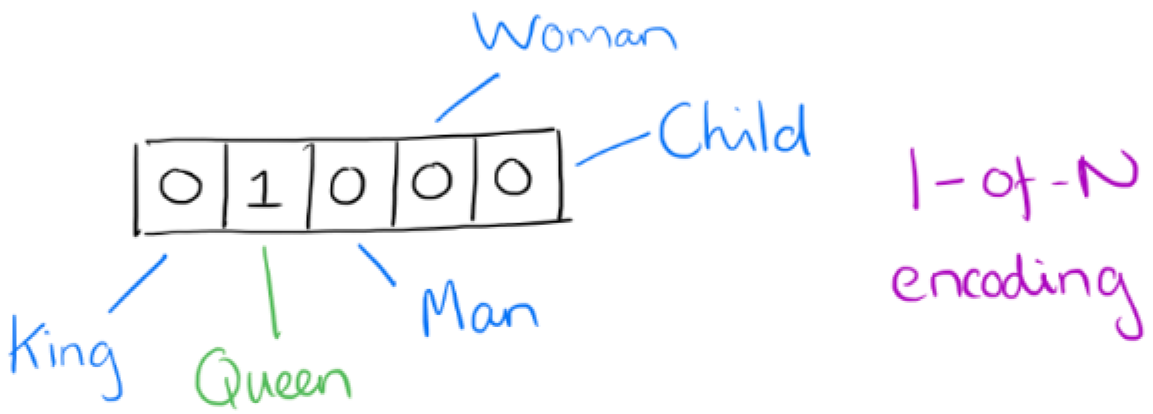
Word Vectors

- At one level, it is simply a vector of weights.
- In a simple 1-of-N (or ‘one-hot’) encoding every element in the vector is associated with a word in the vocabulary.
- The encoding of a given word is simply the vector in which the corresponding element is set to one, and all

motel $[0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0]$ AND
hotel $[0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0] = 0$

Word Vectors: One-Hot Encoding

- Suppose our vocabulary has only five words: King, Queen, Man, Woman and Child.
- We could encode the word ‘Queen’ as:



Limitations of One-hot encoding

- Word vectors are not comparable
- Using such an encoding, there is no meaningful comparison we can make between word vectors other than equality testing.

Word2Vec – A distributed representation

- Distributional representation – word embedding ?
    - Any word w_i in the corpus is given a distributional representation by an embedding
        - $w_i \in R^d$ i.e a d-dimensional vector that is learnt.
- For Example:

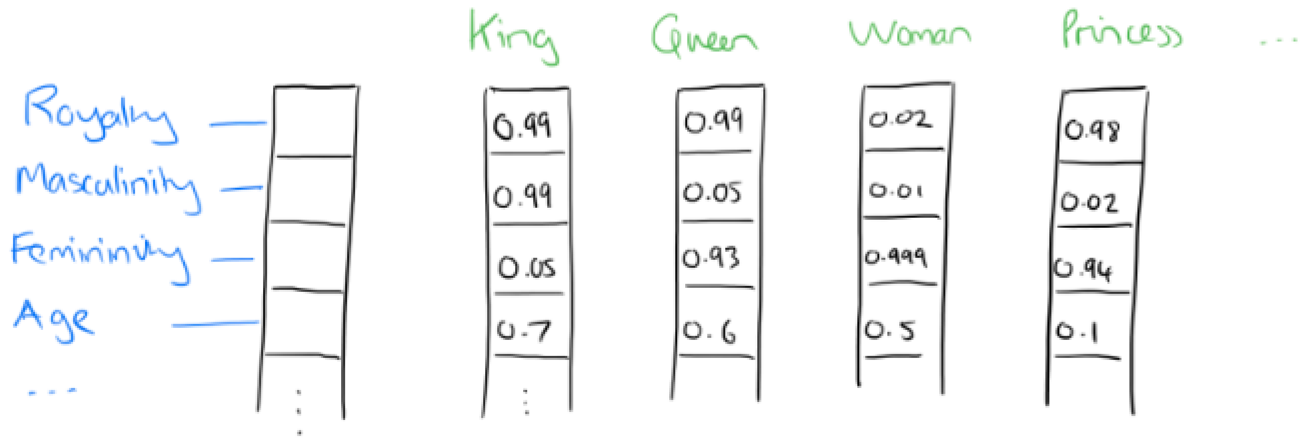
$$\textit{linguistics} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

Distributional Representation

- Take a vector with several hundred dimensions (say 1000).
- Each word is represented by a distribution of weights across those elements.
- So instead of a one-to-one mapping between an element in the vector and a word, the representation of a word is spread across all of the elements in the vector, and each element in the vector contributes to the definition of many words.

Distributional Representation: Illustration

- If we label the dimensions in a hypothetical word vector (there are no such pre-assigned labels in the algorithm of course), it might look a bit like this:



Word Embeddings

- d typically in the range 50 to 1000
- Similar words should have similar embeddings

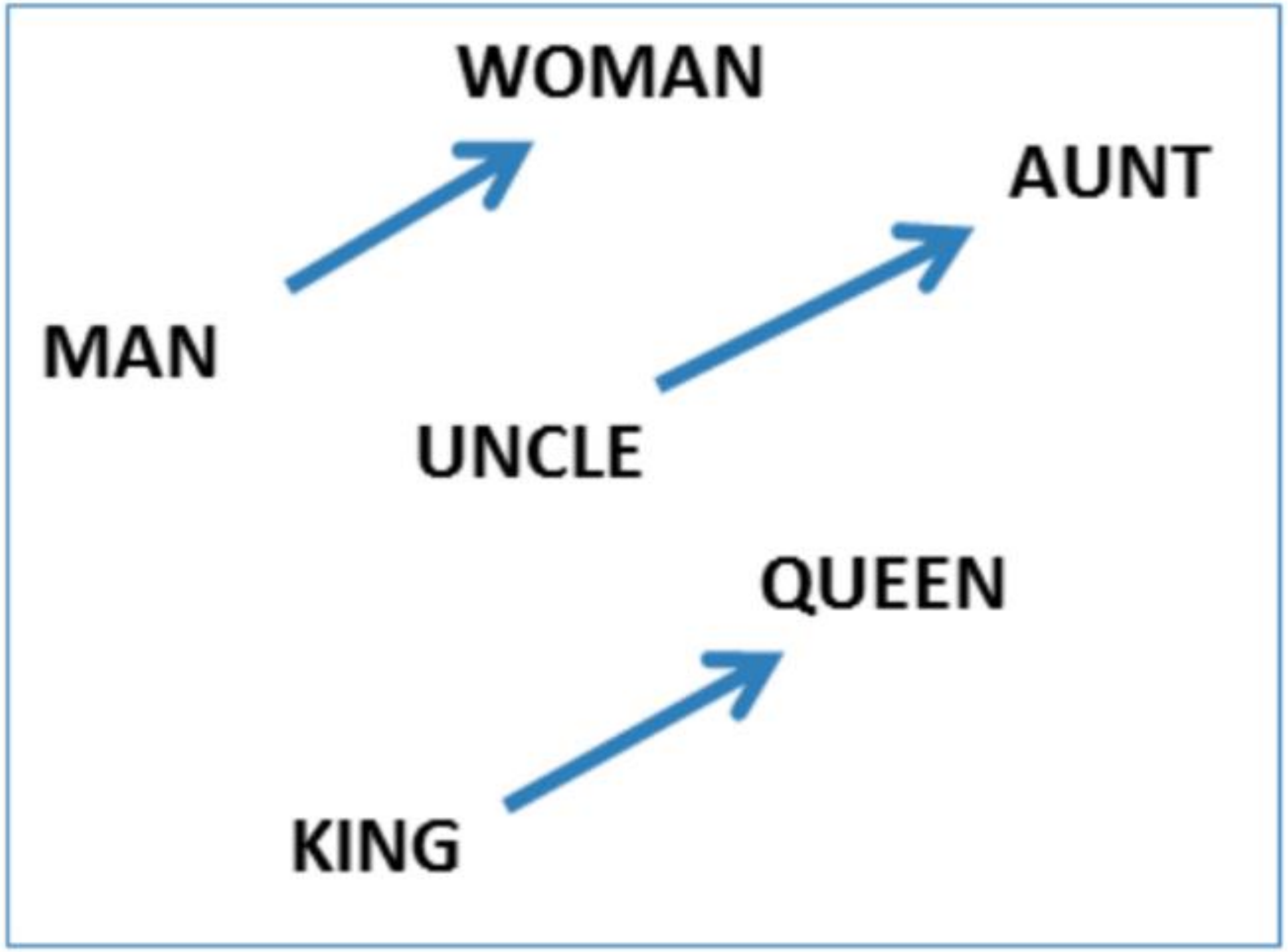
Reasoning with Word Vectors

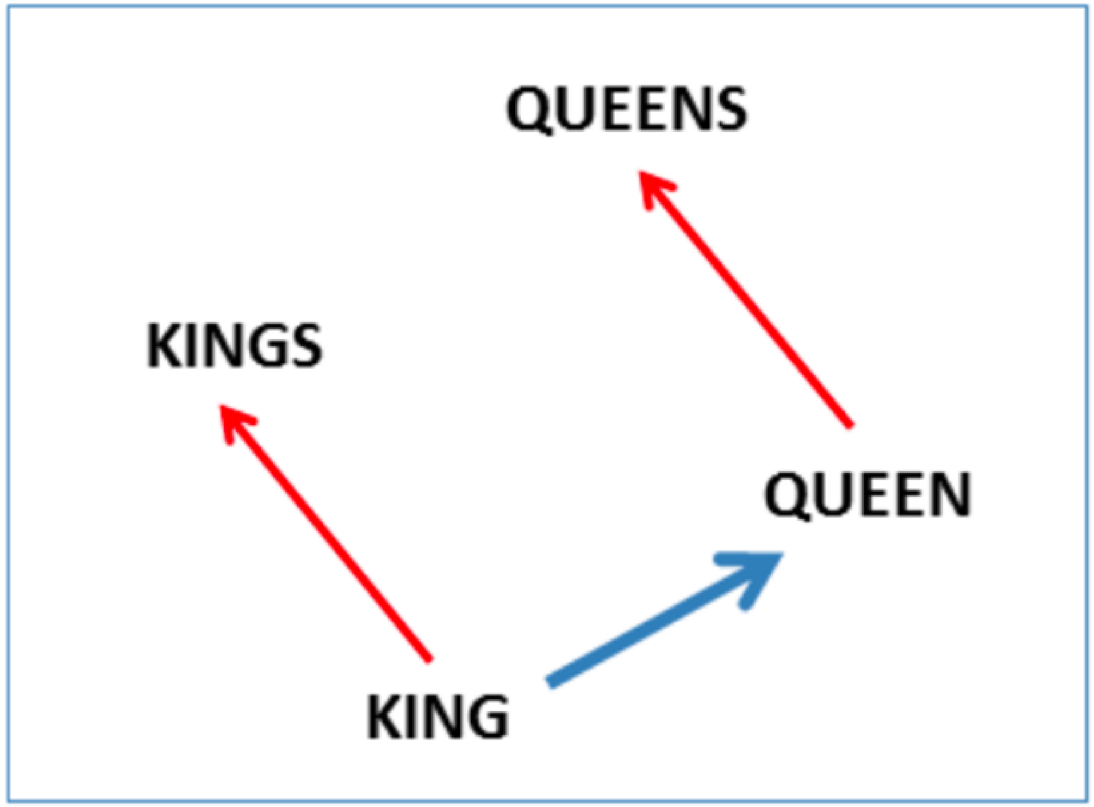
- It has been found that the learned word representations in fact capture meaningful syntactic and semantic regularities in a very simple way.
- Specifically, the regularities are observed as constant vector offsets between pairs of words sharing a particular relationship.
- Case of Singular-Plural Relations
  - If we denote the vector for word i as x_i, and focus on the singular/plural relation, we observe that
    - $x_{apple} - x_{apples} \approx x_{car} - x_{\text{cars}} \approx x_{family} - x_{families} \approx x_{car} - x_{\text{cars}}$ and so on.

Reasoning with Word Vectors

- Perhaps more surprisingly, we find that this is also the case for a variety of semantic relations.
- Good at answering analogy questions:
  - a is to b, as c is to ?
  - man is to woman as uncle is to ? (aunt)
- A simple vector offset method based on cosine distance shows the relation.

- .


WOMAN
AUNT
MAN
UNCLE
QUEEN
KING


QUEENS
KINGS
QUEEN
KING

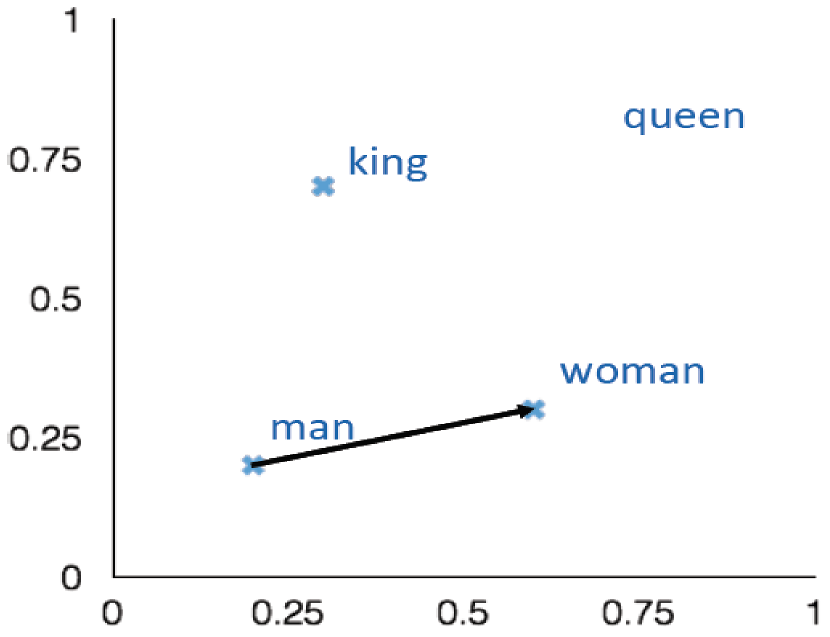
Analogy Test

a:b :: c:?

$$d = \arg\max_x \frac{(w_b - w_a + w_c)^T w_x}{||w_b - w_a + w_c||}$$

man:woman :: king:?

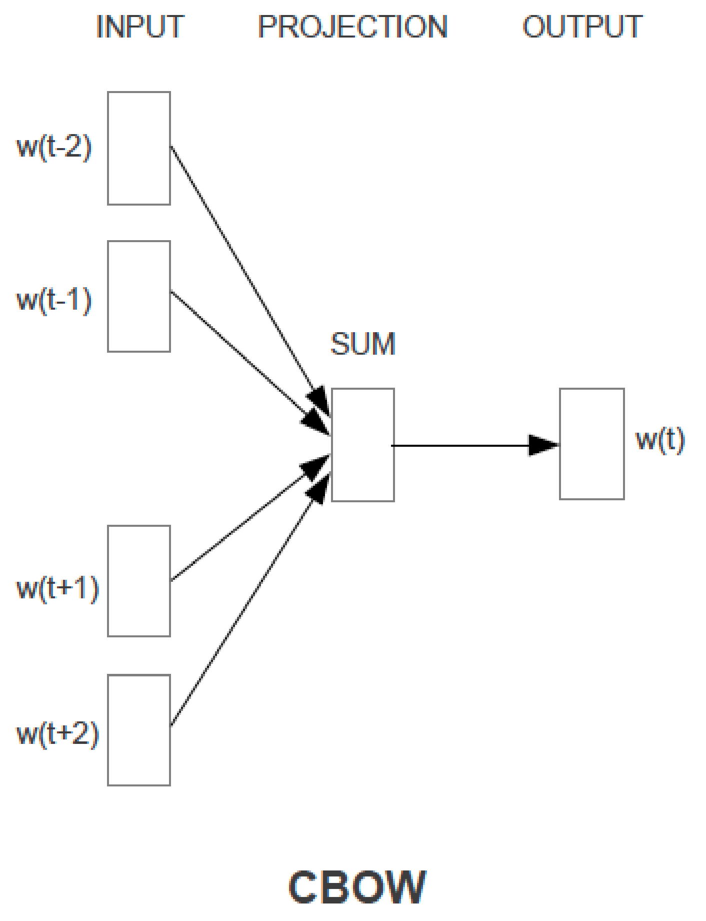
+	king	[0.30 0.70]
-	man	[0.20 0.20]
+	woman	[0.60 0.30]
	queen	[0.70 0.80]

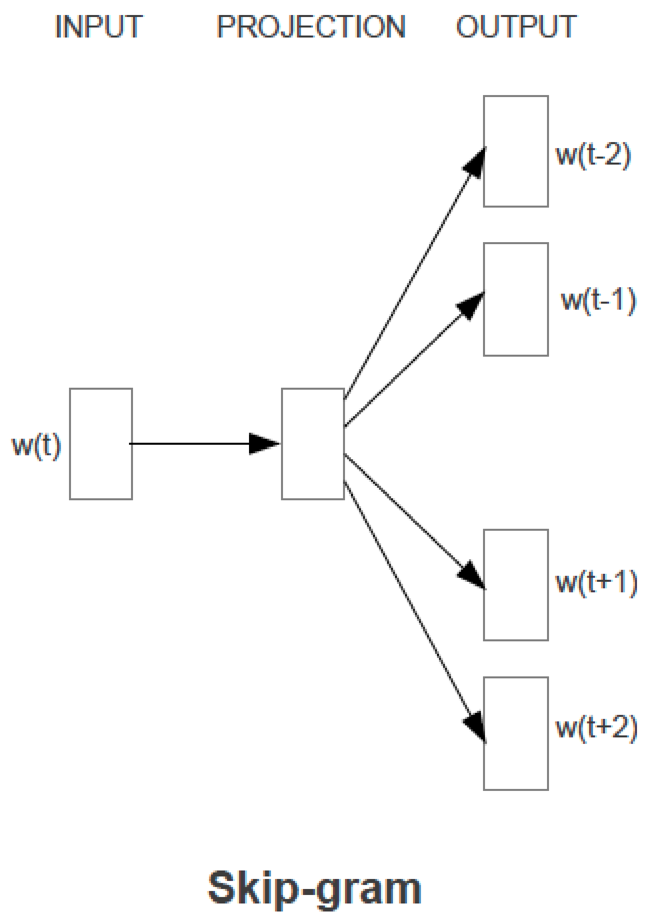


Learning Word Vectors

- Instead of capturing co-occurrence counts directly, predict (using) surrounding words of every word.
- Code as well as word-vectors: https://code.google.com/p/word2vec/

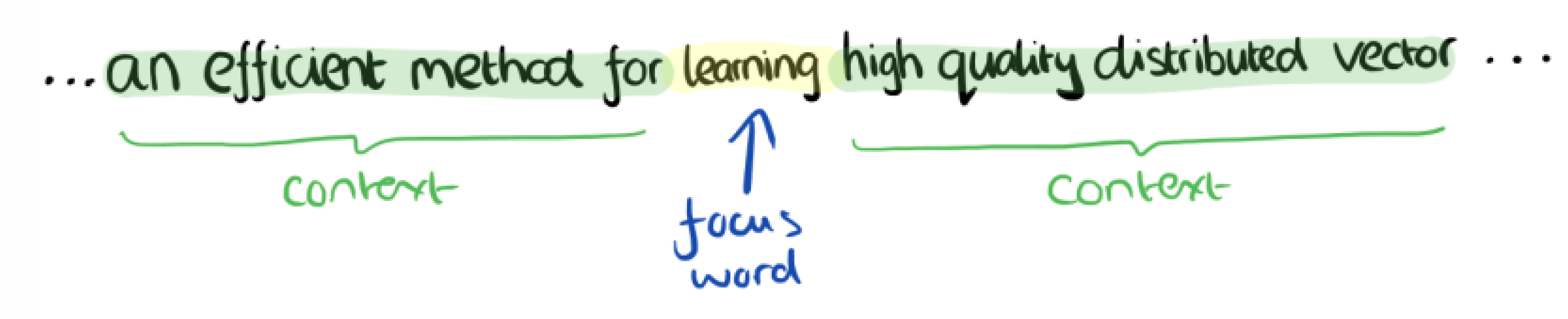
Two Variations: CBOW and Skip-grams





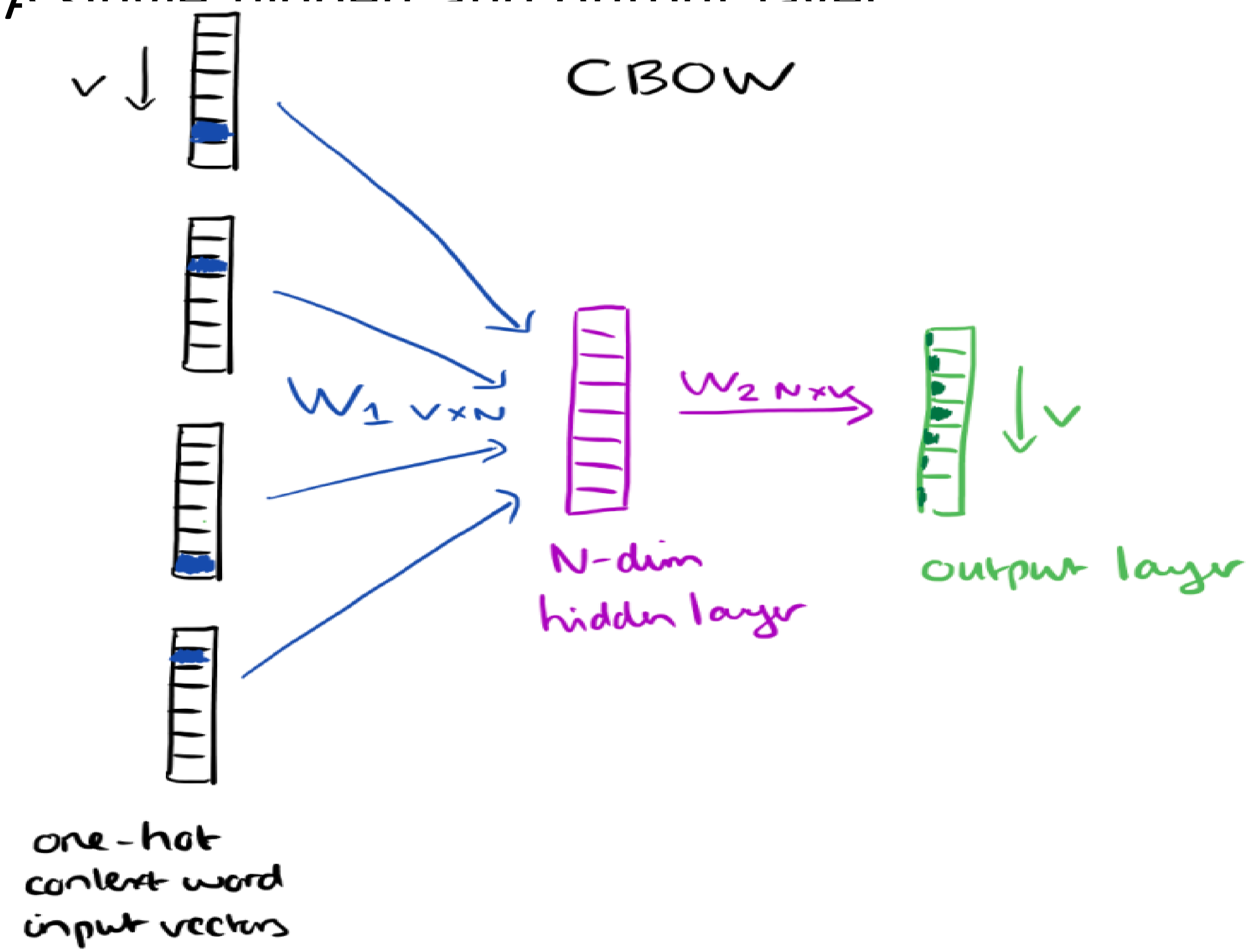
CBOW

- Consider a piece of prose such as:
- “The recently introduced continuous Skip-gram model is an efficient method for learning high-quality distributed vector representations that capture a large number of precises syntatic and semantic word relationships.”
- Imagine a sliding window over the text, that includes the


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

CBOW

- The context words form the input layer. Each word is encoded in one-hot form.
- A single hidden and output layer



CBOW: Training Objective

- 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.

CBOW: Input to Hidden Layer

- 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.

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

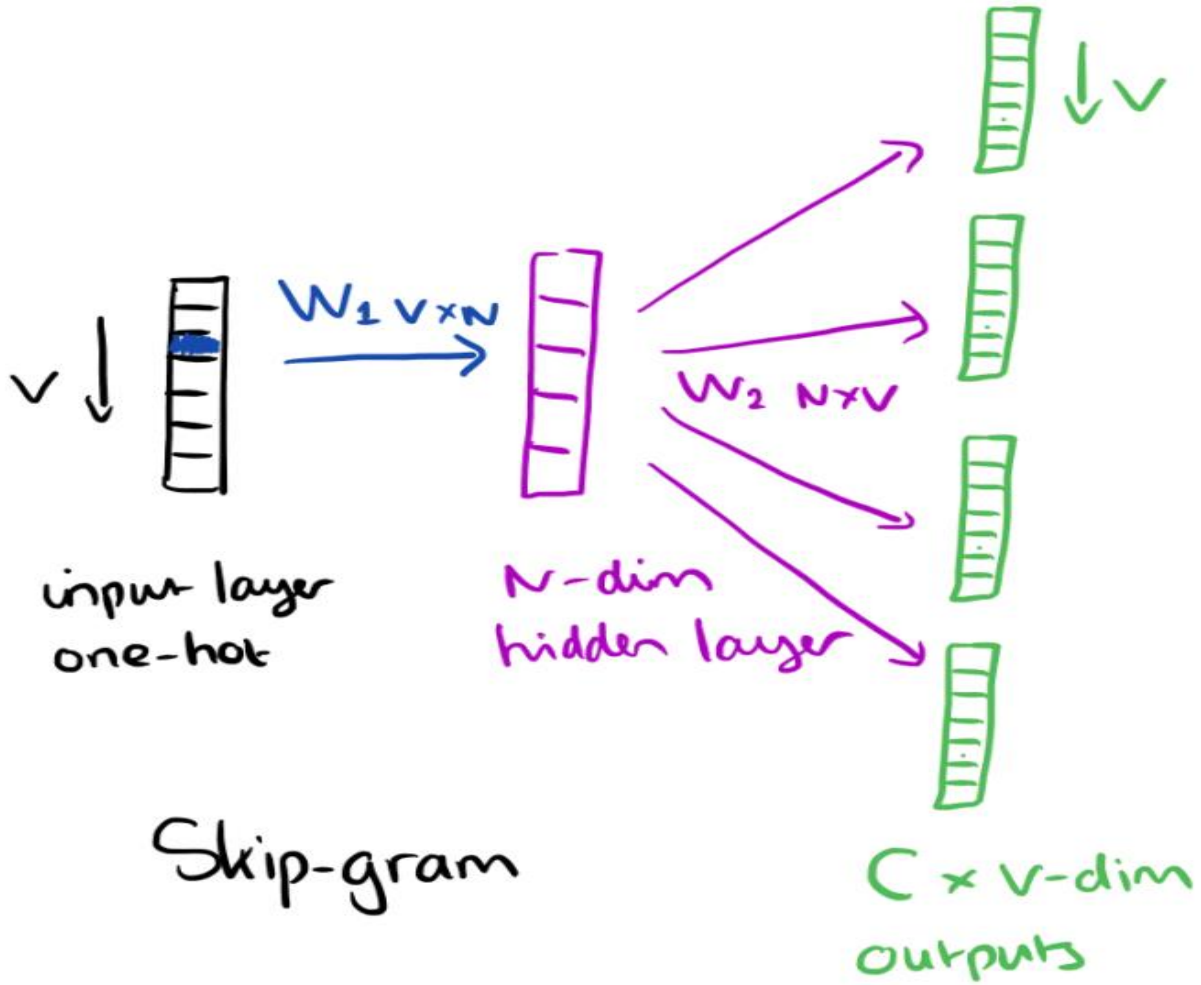
- Given C input word vectors, the activation function for the hidden layer h amounts to simply summing the corresponding ‘hot’ rows in W_1, and dividing by C to take their average.

CBOW: Hidden to Output Layer

- From the hidden layer to the output layer, the second weight matrix W_2 can be used to compute a score for each word in the vocabulary, and softmax can be used to obtain the posterior distribution of words.

Skip-gram Model

- The skip-gram model is the opposite of the CBOW model. It is constructed with the focus word as the single input vector and the target context words are now at



Skipgram : Objective Fn

- The activation function for the hidden layer simply amounts to copying the corresponding row from the weights matrix W_1 (linear) as we saw before.
- At the output layer, we now output C multinomial distributions instead of just one.
- The training objective is to minimize the summed prediction error across all context words in the output layer. In our example, the input would be "learning", and we hope to see ("an", "efficient", "method", "for", "high","quality", "distributed", "vector") at the output layer.

Skip-gram Model

- Predict surrounding words in a window of length c of each word
- Objective Function: Maximize the log probablility of any context word given
- the current center word:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} log\ p(w_{t+j} | w_t)$$

Thank You