

A decorative graphic consisting of two vertical lines, one blue and one red, positioned to the left of the title.

Word Embedding

Dr. G. Bharadwaja Kumar

Vector Space Model

Represent a doc by a term vector

Term: basic concept, e.g., word or phrase

Each term defines one dimension

N terms define a N-dimensional space

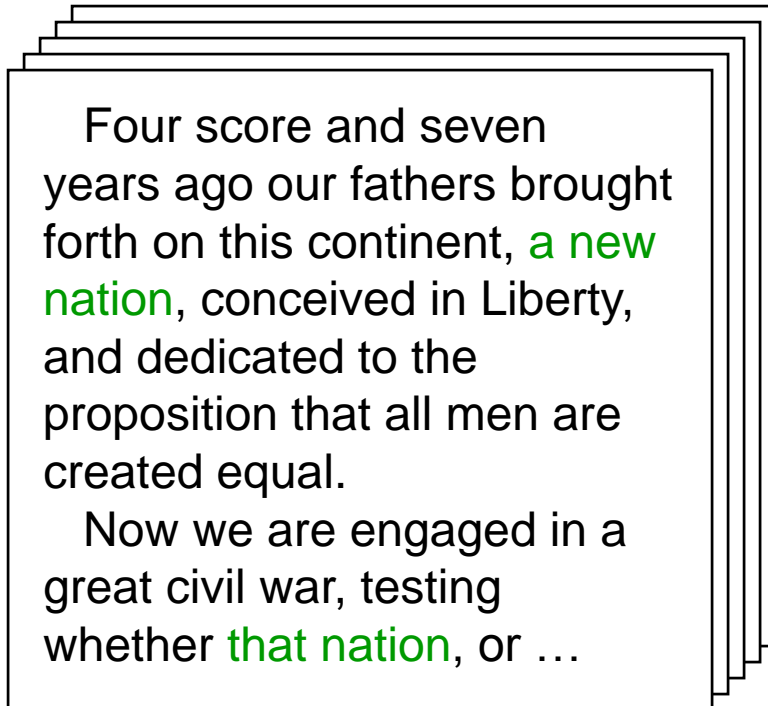
Element of vector corresponds to term weight

E.g., $d = (x_1, \dots, x_N)$, x_i is “importance” of term i

New document is assigned to the most likely category based on vector similarity.

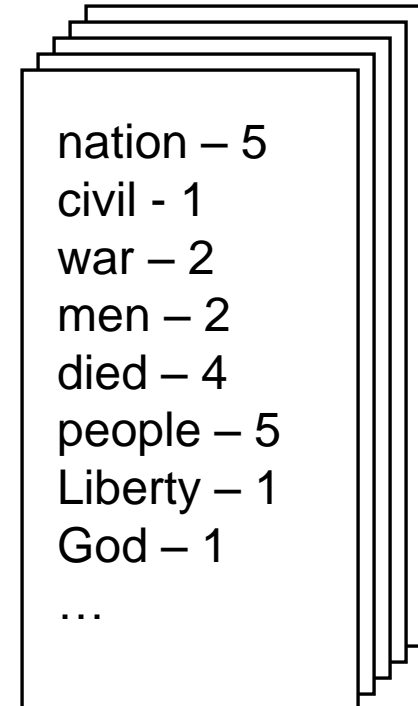
Bag-of-word Approaches

Documents



Feature
Extraction

Token Sets



Loses all order-specific information!
Severely limits context!

Vector Representation

- Suppose the document collection has n distinct words, w_1, \dots, w_n
- Each document is characterized by an n -dimensional vector whose i^{th} component is the frequency of word w_i in the document

Example

- $D1 = [\text{The cat chased the mouse.}]$
- $D2 = [\text{The dog chased the cat.}]$
- $W = [\text{The, chased, dog, cat, mouse}] \quad (n = 5)$
- $V1 = [2, 1, 0, 1, 1]$
- $V2 = [2, 1, 1, 1, 0]$

One-hot encoding

- From the word ID, we get a basic representation of a word through the one-hot encoding of the ID
- the one-hot vector of an ID is a vector filled with 0's, except for a 1 at the position associated with the ID
- ex.: for vocabulary size $D=10$, the one-hot vector of word ID $w=4$ is $e(w) = [0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0]$
- makes no assumption about word similarity • all words are equally different from each other
- this is a natural representation to start with, though a poor one

Need for improvement

- How to select terms to capture “basic concepts”
 - Removing Stop Words
e.g. “a”, “the”, “always”, “along”
 - Stemming
e.g. “computer”, “computing”, “computerize” => “compute”
 - Latent semantic indexing
- How to assign weights
 - Not all words are equally important: Some are more indicative than others
e.g. “algebra” vs. “science”

Latent Semantic Analysis

- The term-document matrix is large
- Latent Semantic Analysis
Singular Value Decomposition to reduce dimensionality

$$\begin{matrix} & \text{documents} \\ \text{words} & \boxed{C} \end{matrix} = \begin{matrix} & \text{dims} \\ \text{words} & \boxed{U} \end{matrix} \begin{matrix} \text{dims} \\ \boxed{D} \end{matrix} \begin{matrix} \text{dims} & \boxed{V^T} \\ & \text{documents} \end{matrix}$$

- Rank of D can be reduced
- Meaning
 - U=term-topic correlation
 - D=topic importance
 - V=document-topic correlation

How to term Weighting using tf-idf

- Two-fold heuristics based on frequency
 - TF (Term frequency)
More frequent **within** a document → more relevant to semantics
e.g., “query” vs. “commercial”
 - IDF (Inverse document frequency)
Less frequent **among** documents → more discriminative
e.g. “algebra” vs. “science”

Problem: *Very* High Dimensionality

A vector of TF*IDF representing a document is high dimensional.

If we start looking at a matrix of terms by documents, it gets even worse.

Need some way to trim words looked at

First, throw away anything "not useful"

Second, identify clusters and pick representative terms

Throw Away

Most domain semantics carried by nouns,
adjectives, verbs, adverbs
throw away prepositions, articles, conjunctions,
pronouns

Very frequent words don't add to domain
semantics.

throw away common verbs (go, be, see), adjectives (big,
good, bad), adverbs (very)

throw away words which appear in most documents

Very infrequent words don't either

throw away terms which only appear in one document

N-gram Language Model

- A **language model** captures the statistical characteristics of sequences of words in a natural language, typically allowing one to make probabilistic predictions of the next word given preceding ones.
- E.g. the standard “trigram” method:

$$P(w_t \mid w_{t-2}, w_{t-1}) = \frac{\text{count}(w_{t-2}w_{t-1}w_t)}{\text{count}(w_{t-2}w_{t-1})}$$

Word Representations

Traditional Method	Word Embeddings
<ul style="list-style-type: none">• Uses one hot encoding• Each word in the vocabulary is represented by one bit position in a HUGE vector.• For example, if we have a vocabulary of 10000 words, and “Hello” is the 4th word in the dictionary, it would be represented by: 0 0 0 1 0 0 0 0 0• Context information is not utilized	<ul style="list-style-type: none">• Stores each word in as a point in space, where it is represented by a vector of fixed number of dimensions (generally 300)• For example, “Hello” might be represented as : [0.4, -0.11, 0.55, 0.3 . . . 0.1, 0.02]• Dimensions are basically projections along different axes, more of a mathematical concept.• Try to catch semantics in terms contextual information

Word embedding

- **Word embedding** is the collective name for a set of language modeling and feature learning techniques in natural language processing (NLP) where **words** or phrases from the vocabulary are mapped to vectors of real numbers.
- Most new word embedding techniques rely on a neural network architecture instead of more traditional n-gram models

Word Embedding

Suppose you have a dictionary of words.

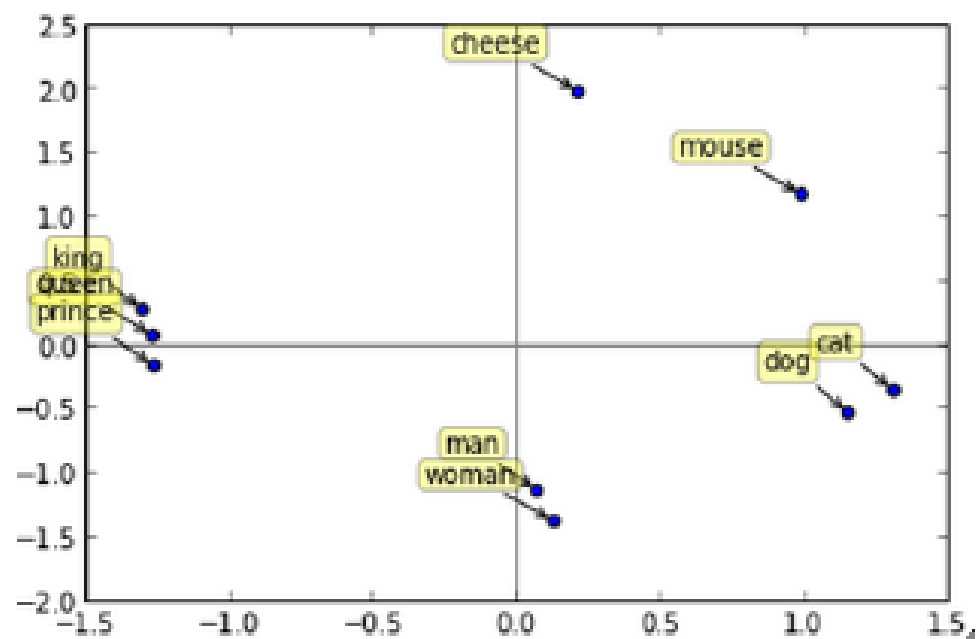
The i^{th} word in the dictionary is represented by an embedding:

$$w_i \in \mathbb{R}^d$$

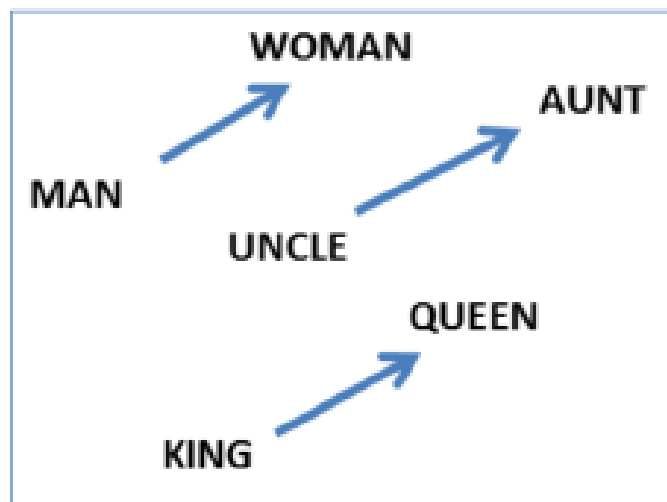
i.e. a d -dimensional vector, which is learnt!

- d typically in the range 50 to 1000.
- Similar words should have similar embeddings (share latent features).
- Embeddings can also be applied to *symbols* as well as words (e.g. Freebase nodes and edges).
- can also have embeddings of phrases, sentences, documents, or even other modalities such as images.

- spatial distance corresponds to word similarity
- words are close together \Leftrightarrow their "meanings" are similar
- notation: word $w \mapsto \text{vec}[w]$ its point in space, as a position vector.



- the displacement (vector) between the points of two words represents the word relationship.
- same word relationship \Rightarrow same vector



Source: *Linguistic Regularities in Continuous Space Word Representations*, Mikolov et al, 2013

e.g.

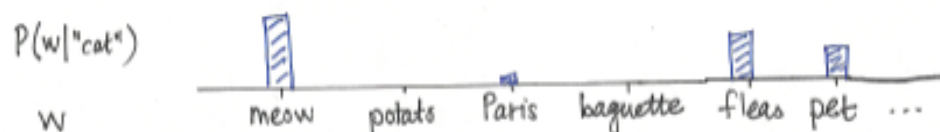
$$\text{vec}[\text{queen}] - \text{vec}[\text{king}] = \text{vec}[\text{woman}] - \text{vec}[\text{man}]$$

-
- How can a machine learn the meaning of a word? Machines only understand symbols!

- Assume the **Distributional Hypothesis (D.H.)** (Harris, 1954):

“words are characterised by the company that they keep”

- Suppose we read the word “cat”. What is the probability $P(w|\text{cat})$ that we’ll read the word w nearby?



- D.H. : the meaning of “cat” is captured by the probability distribution $P(\cdot|\text{cat})$.

Word2Vec

- word2vec associates words with points in space
- word meaning and relationships between words are encoded spatially
- learns from input texts developed by Mikolov, Sutskever, Chen, Corrado and Dean in 2013 at Google Research

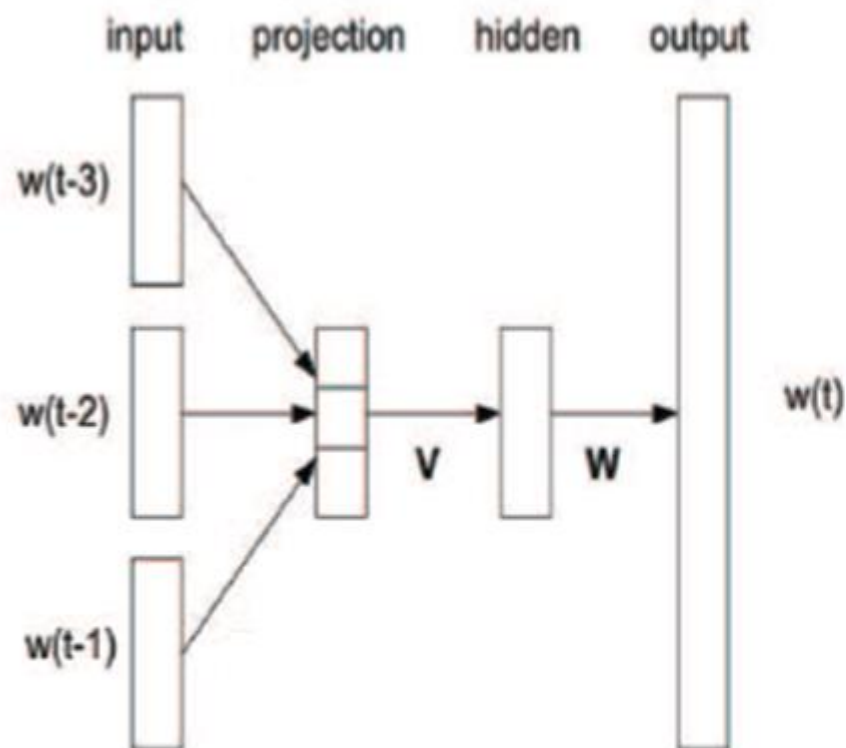
What is word2vec?

- word2vec is **not** a single algorithm
- It is a **software package** for representing words as vectors, containing:
 - Two distinct models
 - CBoW**
 - Skip-Gram**
 - Various training methods
 - Negative Sampling**
 - Hierarchical Softmax
 - A rich preprocessing pipeline
 - Dynamic Context Windows
 - Subsampling
 - Deleting Rare Words

NNLM

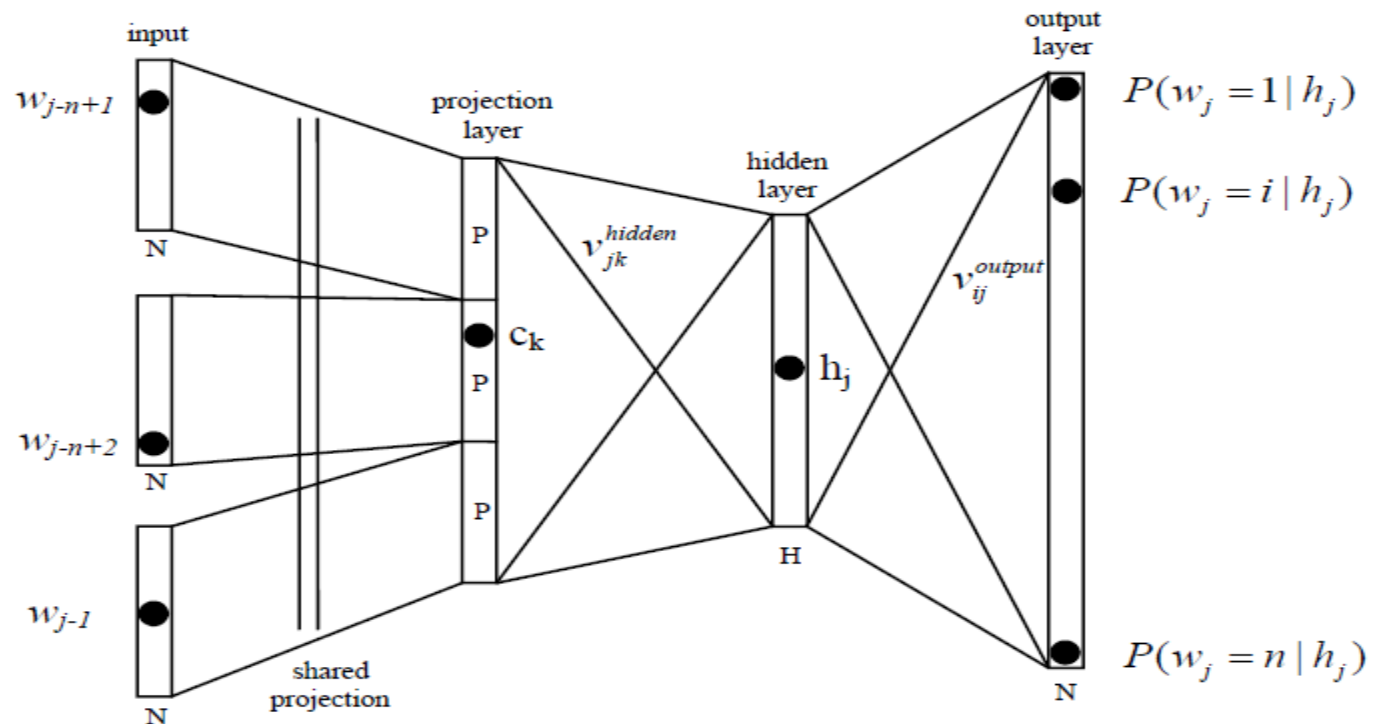
- A **neural network language model** is a language model based on neural networks, exploiting their ability to learn distributed representations.
- A **distributed representation** of a word is a vector of activations of neurons (real values) which characterizes the meaning of the word.
- A distributed representation is opposed to a *local* representation, in which only one neuron (or very few) is active at each time.

NNLM by Bengio



- Bengio et al. (2003, JMLR)
- projection layer, hidden layer and output layer
- The projection layer is linear
- The hidden layer is non-linear
- Softmax at the output computes probability distribution over the whole vocabulary ($g(o) = \frac{e^{o_i}}{\sum_k e^{o_k}}$)
- Model is computationally very expensive

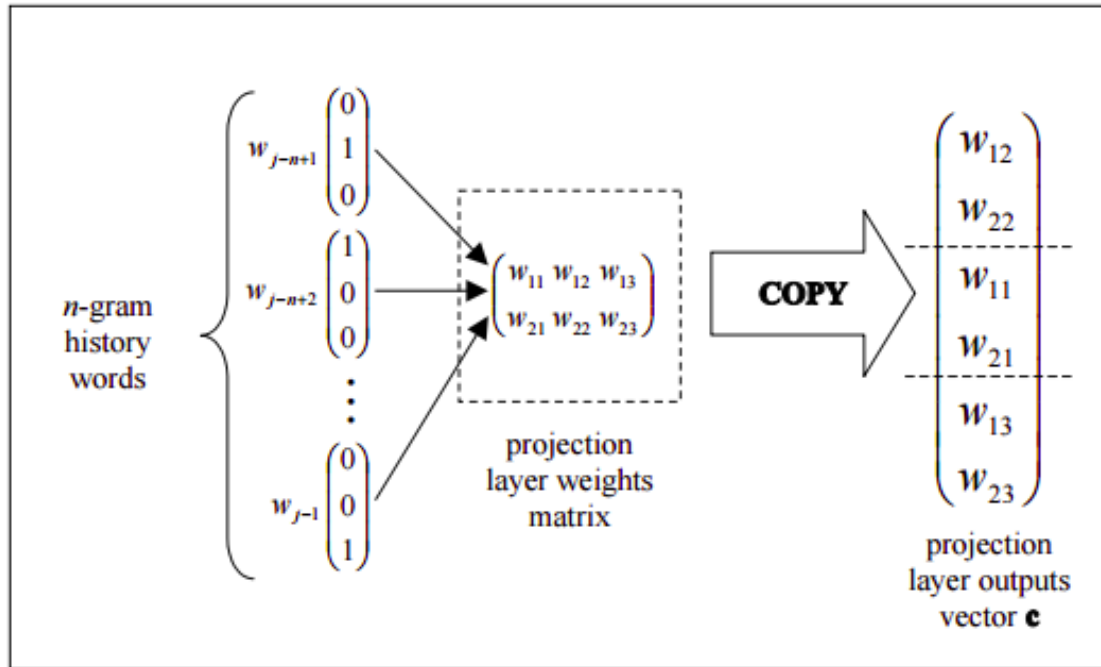
NNLM



- Activation functions
 - input→embedding: none
 - embedding→hidden: tanh
 - hidden→output: softmax

Projection or embedding Layer


The projection layer maps the discrete word indices of an n-gram context to a continuous vector space. It is linear



Hidden layer

Map each word first into a lower-dimensional real-valued space

The main reason for using tanh as an activation function instead of the more common logistic regression sigmoid is that the rate of convergence of tanh is often faster than sigmoid


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

Output layer

The purpose of the output layer is to compute the posterior probabilities of each word w_j in the vocabulary given the n -gram context h_j that was fed forward through the network.

The output layer uses a softmax normalization of the weighted sum of hidden layer activities

Sigmoid Vs. Softmax Vs. Logistic

Logistic Function

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}$$

Sigmoid

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

Softmax function

$$P(y = j | \mathbf{x}) = \frac{e^{\mathbf{x}^\top \mathbf{w}_j}}{\sum_{k=1}^K e^{\mathbf{x}^\top \mathbf{w}_k}}$$

- e natural logarithm base, x_0 the x -value of the sigmoid's midpoint, L the curve's maximum value, k steepness of curve
- Sigmoid is special case of the logistic function
- Softmax is a generalization of the logistic function
- $\mathbf{x}^\top \mathbf{w}$ denotes the inner product of \mathbf{x} and \mathbf{w}

Softmax function calculates the probabilities distribution of the event over 'n' different events. In general way of saying, this function will calculate the probabilities of each target class over all possible target classes.

The main advantage of using Softmax is the output probabilities range. The range will 0 to 1, and the sum of all the probabilities will be equal to one.

If the softmax function used for multi-class classification model it returns the probabilities of each class and the target class will have the high probability.

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$

You can see in the denominator that we have an operation involving all W word vectors, hence the inefficiency.

CBOW

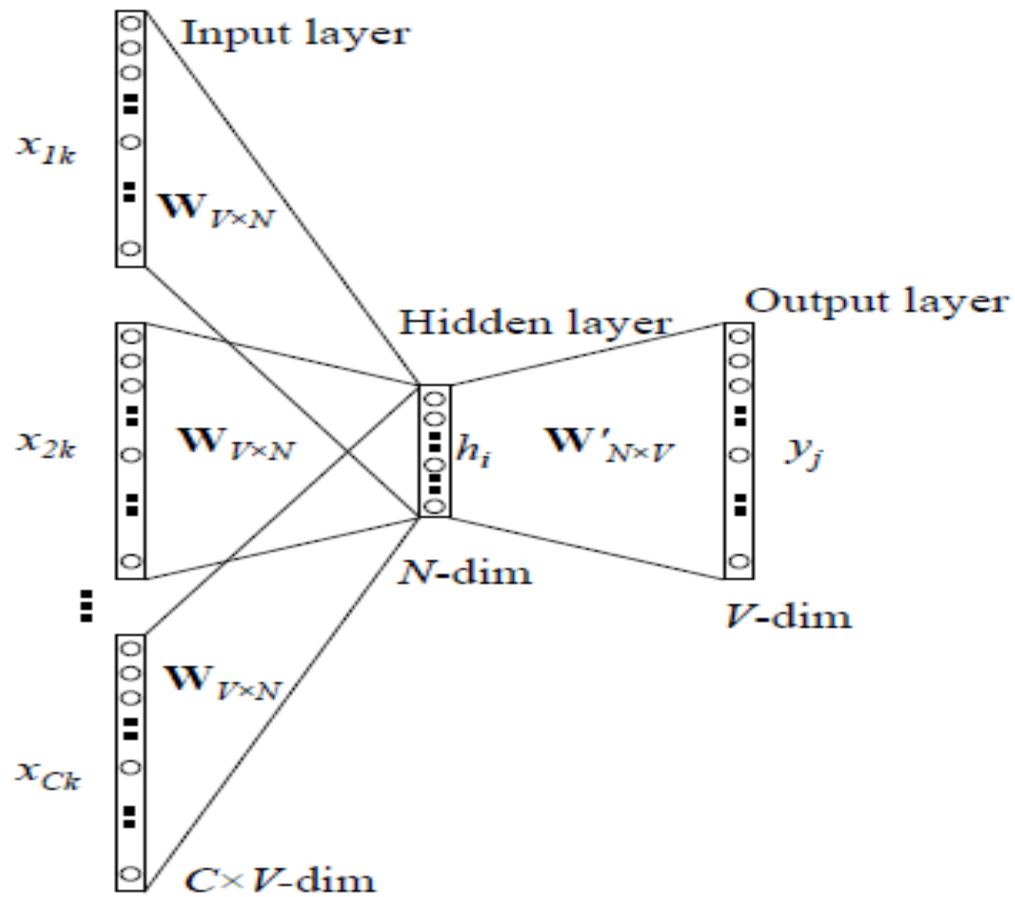


Figure 2: Continuous bag-of-words model

CBOW

In the continuous bag of words model, context is represented by multiple words for a given target words

In this approach is to treat {"The", "cat", 'over", "the', "puddle"} as a context and from these words, be able to predict or generate the center word "jumped".

The input layer is set to have as many neurons as there are words in the vocabulary for training.

The neurons in the hidden layer are all linear neurons.

The hidden layer size is set to the dimensionality of the resulting word vectors.

Model Work flow

We breakdown the way this model works in these steps:

1. We generate our one hot word vectors $(x^{(c-m)}, \dots, x^{(c-1)}, x^{(c+1)}, \dots, x^{(c+m)})$ for the input context of size m .
2. We get our embedded word vectors for the context ($v_{c-m} = \mathcal{V}x^{(c-m)}, v_{c-m+1} = \mathcal{V}x^{(c-m+1)}, \dots, v_{c+m} = \mathcal{V}x^{(c+m)}$)
3. Average these vectors to get $\hat{v} = \frac{v_{c-m} + v_{c-m+1} + \dots + v_{c+m}}{2m}$
4. Generate a score vector $z = \mathcal{U}\hat{v}$
5. Turn the scores into probabilities $\hat{y} = \text{softmax}(z)$
6. We desire our probabilities generated, \hat{y} , to match the true probabilities, y , which also happens to be the one hot vector of the actual word.

Skip-gram model

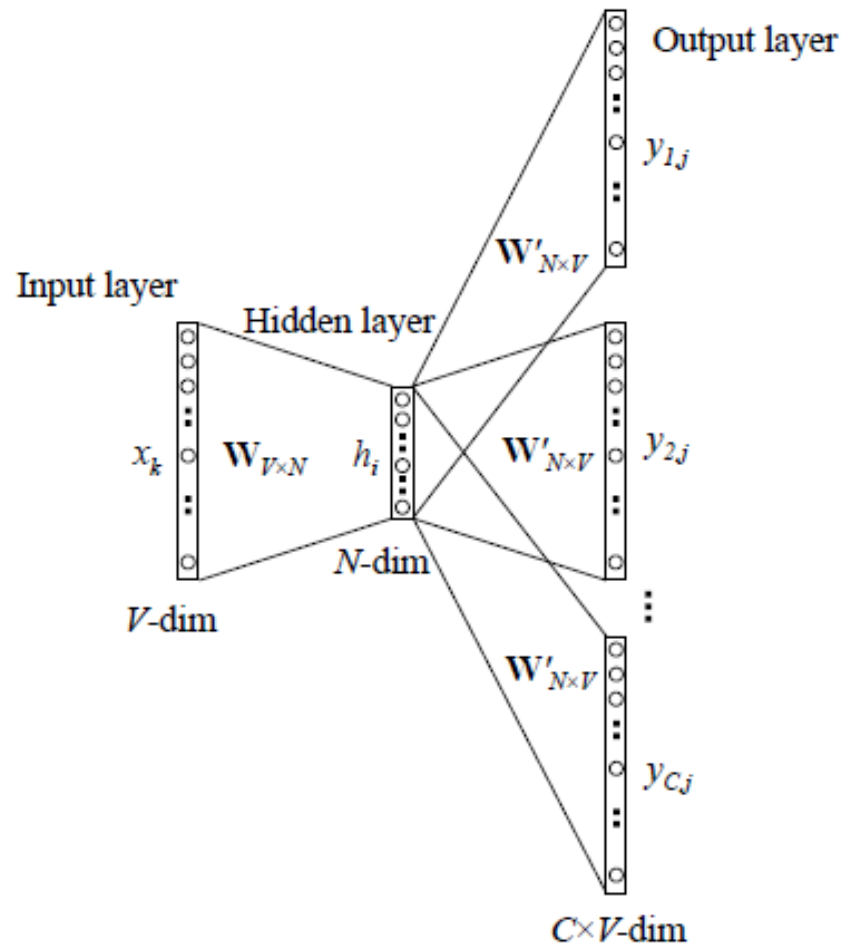


Figure 3: The skip-gram model.

Skip-gram model

Skip-gram model reverses the use of target and context words. In this case, the target word is fed at the input, the hidden layer remains the same, and the output layer of the neural network is replicated multiple times to accommodate the chosen number of context words.

Here the approach is to create a model such that given the center word "jumped", the model will be able to predict or generate the surrounding words "The", "cat", "over", "the", "puddle".

Skip-gram model work flow

We breakdown the way this model works in these 6 steps:

1. We generate our one hot input vector x
2. We get our embedded word vectors for the context $v_c = \mathcal{V}x$
3. Since there is no averaging, just set $\hat{v} = v_c$?
4. Generate $2m$ score vectors, $u_{c-m}, \dots, u_{c-1}, u_{c+1}, \dots, u_{c+m}$ using $u = \mathcal{U}v_c$
5. Turn each of the scores into probabilities, $y = \text{softmax}(u)$
6. We desire our probability vector generated to match the true probabilities which is $y^{(c-m)}, \dots, y^{(c-1)}, y^{(c+1)}, \dots, y^{(c+m)}$, the one hot vectors of the actual output.

Problem with Softmax

v'_{w_O} is the output vector of the training word we want to predict, given v'_{w_I} , which is the vector of the input word we are using for the basis of prediction (both vectors are initially random, and are learned during training).

$$p(w_O|w_I) = \frac{\exp(v'_{w_O}^T v_{w_I})}{\sum_{w=1}^W \exp(v'_w{}^T v_{w_I})}$$

Computing the denominator in the softmax function for every word is inefficient

During training you only need to calculate the probability of one word in CBOW model) no need to probability of every single word in your vocabulary. However, when you use the softmax you will need to calculate the activations for every single word. By using hierarchical softmax the training complexity is reduced from $O(V)$ to $O(\log(V))$ where V is the number of words in your vocabulary.

hierarchical softmax

The “hierarchical softmax” approach is one method proposed to address this inefficiency.

Rather than update all words in the vocabulary during training, only $\log(W)$ words are updated (where W is the size of the vocabulary).

Use the Huffman tree to redefine our probability optimization function:

-
- Huffman coding is based on the frequency of occurrence of a data item.
 - The principle is to use a lower number of bits to encode the data that occurs more frequently.
 - Codes are stored in a Code Book which may be constructed for each image or a set of images. In all cases the code book plus encoded data must be transmitted to enable decoding.

Negative Sampling

Instead of propagating signal from the hidden layer to the whole output layer, only the output neuron that represents the positive class + few randomly sampled neurons are evaluated

Modifying the optimization objective with a technique they called “Negative Sampling”, which causes each training sample to update only a small percentage of the model’s weights.

-
- Let D denote dataset of observed (w, c) pairs
 - $p(D = 1|w, c)$ denotes probability that (w, c) came from D
 - $p(D = 0|w, c) = 1 - p(D = 1|w, c)$ that it didn't

$$\operatorname{argmax}_{\theta} \prod_{(w,c) \in D} p(D = 1|w, c; \theta) \prod_{(w,c) \in D'} p(D = 0|w, c; \theta)$$

$$\operatorname{argmax}_{\theta} \sum_{(w,c) \in D} \log p(D = 1|w, c; \theta) + \sum_{(w,c) \in D'} \log(1 - p(D = 1|w, c; \theta))$$

$$\operatorname{argmax}_{\theta} \sum_{(w,c) \in D} \log \frac{1}{1 + e^{-v_c \cdot v_w}} + \sum_{(w,c) \in D'} \log \frac{1}{1 + e^{v_c \cdot v_w}}$$

Subsampling

subsampling is a method of diluting very frequent words, akin to removing stop-words. The subsampling method presented in (Mikolov et al., 2013) randomly removes words that are more frequent than some threshold t with a probability of p , where f marks the word's corpus frequency:

$$p = \frac{f-t}{f} - \sqrt{\frac{t}{f}}$$

The recommended value for t is $t=10^{-5}$.

For example, "the quick brown fox jumped over the lazy dog" might be reduced to "quick brown fox jumped lazy dog". In this case, "dog" is now in the 2-word context window of "jumped", even though it was originally 4 tokens away from "jumped".

-
- **Architecture:** skip-gram (slower, better for infrequent words) vs CBOW (fast)
 - **The training algorithm:** hierarchical softmax (better for infrequent words) vs negative sampling (better for frequent words, better with low dimensional vectors)
 - **Sub-sampling of frequent words:** can improve both accuracy and speed for large data sets (useful values are in range $1e-3$ to $1e-5$)
 - **Dimensionality of the word vectors:** usually more is better, but not always
 - **Context (window) size:** for skip-gram usually around 10, for CBOW around 5