# Learning Word Embedding

Oct 15, 2017 by Lilian Weng nlp language-model

> Word embedding is a dense representation of words in the form of numeric vectors. It can be learned using a variety of language models. The word embedding representation is able to reveal many hidden relationships between words. For example, vector("cat") - vector("kitten") is similar to vector("dog") - vector("puppy"). This post introduces several models for learning word embedding and how their loss functions are designed for the purpose.

Human vocabulary comes in free text. In order to make a machine learning model understand and process the natural language, we need to transform the free-text words into numeric values. One of the simplest transformation approaches is to do a one-hot encoding in which each distinct word stands for one dimension of the resulting vector and a binary value indicates whether the word presents (1) or not (0).

However, one-hot encoding is impractical computationally when dealing with the entire vocabulary, as the representation demands hundreds of thousands of dimensions. Word embedding represents words and phrases in vectors of (non-binary) numeric values with much lower and thus denser dimensions. An intuitive assumption for good word embedding is that they can approximate the similarity between words (i.e., "cat" and "kitten" are similar words, and thus they are expected to be close in the reduced vector space) or disclose hidden semantic relationships (i.e., the relationship between "cat" and "kitten" is an analogy to the one between "dog" and "puppy"). Contextual information is super useful for learning word meaning and relationship, as similar words may appear in the similar context often.

There are two main approaches for learning word embedding, both relying on the contextual knowledge.

- **Count-based**: The first one is unsupervised, based on matrix factorization of a global word co-occurrence matrix. Raw co-occurrence counts do not work well, so we want to do smart things on top.
- **Context-based**: The second approach is supervised. Given a local context, we want to design a model to predict the target words and in the meantime, this model learns the efficient word embedding representation.

# Count-Based Vector Space Model

Count-based vector space models heavily rely on the word frequency and co-occurrence matrix with the assumption that words in the same contexts share similar or related semantic meanings. The models map count-based statistics like co-occurrences between neighboring words down to a small and dense word vectors. PCA, topic models, and neural probabilistic language models are all good examples of this category.

**Answer:** **Count Based :** In count based models, the semantic similarity between words is learned by counting the co-occurrence frequency. For example, in the example
Kitty likes milk.
Cat likes milk.
the co-occurence matrix computed will be

| | Kitty | likes | Milk | Cat |
|---|---|---|---|---|
| Kitty | 0 | 1 | 1 | 0 |
| likes | 1 | 0 | 2 | 1 |
| Milk | 1 | 2 | 0 | 1 |
| Cat | 0 | 1 | 1 | 0 |

Based on the Count based matrix, we can deduce the Cat and kitty are related.

# Context-Based: Skip-Gram Model

Suppose that you have a sliding window of a fixed size moving along a sentence: the word in the middle is the "target" and those on its left and right within the sliding window are the context words. The skip-gram model (Mikolov et al., 2013) is trained to predict the probabilities of a word being a context word for the given target.

The following example demonstrates multiple pairs of target and context words as training samples, generated by a 5-word window sliding along the sentence.

> "The man who passes the sentence should swing the sword." – Ned Stark

| Sliding window (size = 5) | Target word | Context |
|---|---|---|
| [The man who] | the | man, who |
| [The man who passes] | man | the, who, passes |
| [The man who passes the] | who | the, man, passes, the |
| [man who passes the sentence] | passes | man, who, the, sentence |
| ... | ... | ... |
| [sentence should swing the sword] | swing | sentence, should, the, sword |
| [should swing the sword] | the | should, swing, sword |
| [swing the sword] | sword | swing, the |

Each context-target pair is treated as a new observation in the data. For example, the target word "swing" in the above case produces four training samples: ("swing", "sentence"), ("swing", "should"), ("swing", "the"), and ("swing", "sword").

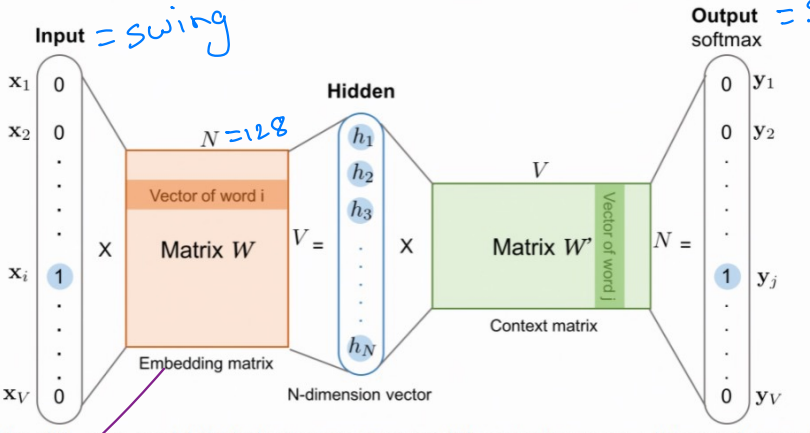



*Fig. 1. The skip-gram model. Both the input vector $\mathbf{x}$ and the output $\mathbf{y}$ are one-hot encoded word representations. The hidden layer is the word embedding of size $N$.*

Given the vocabulary size $V$, we are about to learn word embedding vectors of size $N$. The model learns to predict one context word (output) using one target word (input) at a time.

## (CBOW)

The Continuous Bag-of-Words (CBOW) is another similar model for learning word vectors. It predicts the target word (i.e. "swing") from source context words (i.e., "sentence should the sword").

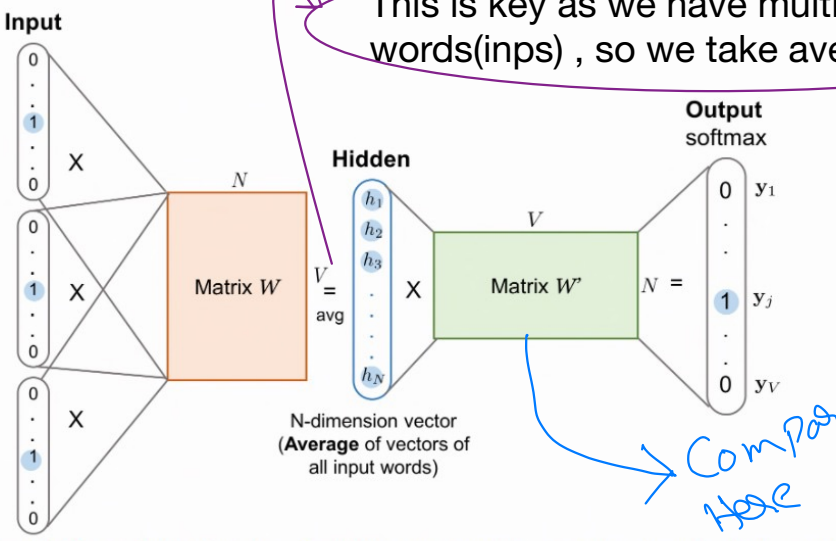
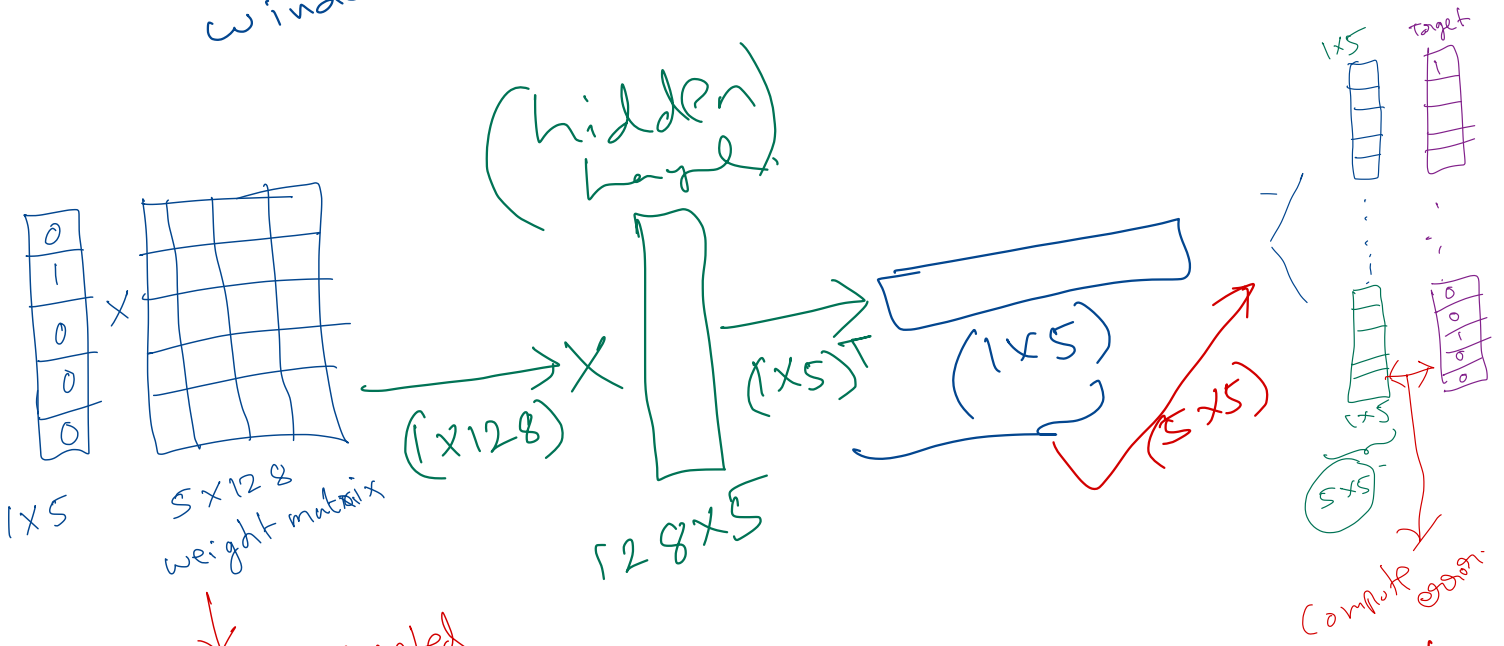



*Fig. 2. The CBOW model. Word vectors of multiple context words are averaged to get a fixed-length vector as in the hidden layer. Other symbols have the same meanings as in Fig 1.*

word2vec Skigram:

Hope can set you free

window size=5, Can is our input.




we are interested to learn this

(categorical Cross Entropy)

to

we have used negative sampling to overcome softmax prob

# Loss Functions

Both the skip-gram model and the CBOW model should be trained to minimize a well-designed loss/objective function. There are several loss functions we can incorporate to train these language models. In the following discussion, we will use the skip-gram model as an example to describe how the loss is computed.

## Full Softmax

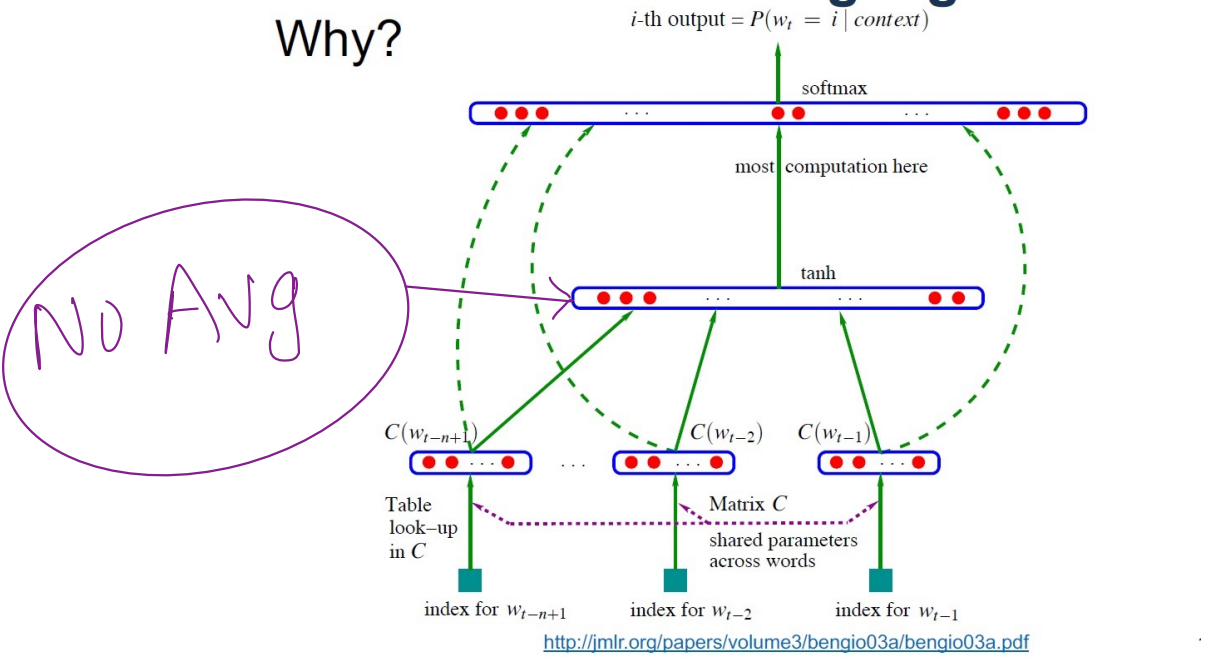
The skip-gram model defines the embedding vector of every word by the matrix $W$ and the context vector by the output matrix $W'$. Given an input word $w_I$, let us label the corresponding row of $W$ as vector $v_{w_I}$ (embedding vector) and its corresponding column of $W'$ as $v'_{w_I}$ (context vector). The final output layer applies softmax to compute the probability of predicting the output word $w_O$ given $w_I$, and therefore:

$$p(w_O|w_I) = \frac{\exp({v'_{w_O}}^\top v_{w_I})}{\sum_{i=1}^{V} \exp({v'_{w_i}}^\top v_{w_I})}$$

This is accurate as presented in Fig. 1. However, when $V$ is extremely large, calculating the denominator by going through all the words for every single sample is computationally impractical. The demand for more efficient conditional probability estimation leads to the new methods like *hierarchical softmax*.

## CBOW != Neural Language Model

Why?



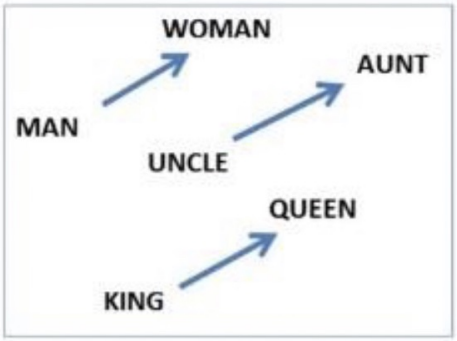

http://jmlr.org/papers/volume3/bengio03a/bengio03a.pdf

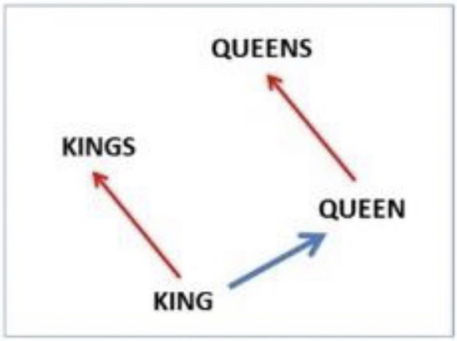
How can we assess the quality of word embeddings.

# Word Arithmetic

(word embeddings)

vec("king") – vec("man") + vec("woman") = vec("queen")




https://code.google.com/archive/p/word2vec/

## Challenges

Which common challenge arises for the loss computation when training word vectors and how can it be addressed?

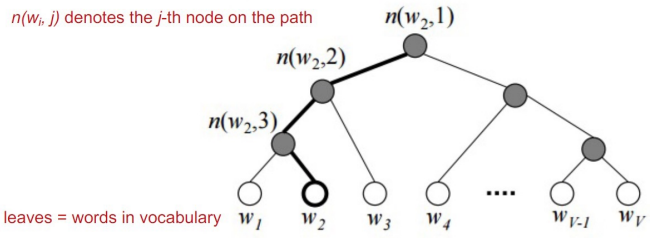
## Softmax Bottleneck

$$\text{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

need to sum over **large** vocabulary!

- workarounds:
  - a) **softmax-based approaches:** keep softmax layer intact, make it more efficient
  - b) **sampling-based approaches:** optimize a different loss function that approximates the softmax

## Hierarchical Softmax




- each word has a unique path => determines probability
- good tree structure is crucial for model performance
- O(log V) at training time (when target word known)
- still need to consider all words during test time

## Negative Sampling

- simplified Noise Contrastive Estimation
- loss: cross-entropy between predicted probabilities p and the true binary labels y
- only consider probabilities for some noise words (negative samples) => contrast

# Design Decisions

- **Architecture**: skip-gram (slower, better for infrequent words) vs CBOW (fast).
- **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.