Word2Vec

click here to edit the note

Neural networks and most statistical models cannot work directly with raw text. To apply machine learning to language, we first need to represent words numerically. One of the most effective ways to do this is through word embeddings—dense vector representations that capture semantic relationships between words.

This tutorial introduces one of the most influential methods for learning word embeddings: Word2Vec. It is designed to help you understand:

- *Why word embeddings are needed,*
- *How Word2Vec works (both CBOW and Skip-gram models), and*
- *How to implement it in code from scratch and with libraries.*

We'll walk through both theory and hands-on coding exercises to reinforce your understanding.

Throughout this tutorial, we'll use a simple example document: 'to be or not to be' to illustrate key concepts step by step.

One-hot Encoding

One direct thought is to use one-hot encoding, which is a method to convert categorical variables into numerical format by creating binary (0/1) indicator columns — one for each category level. In text analysis, we create a vector with dimension same as the number of vocabulary for each unique word in the document.

$$v_{to} = (1,0,0,0)^T$$
$$v_{be} = (0,1,0,0)^T$$
$$v_{or} = (0,0,1,0)^T$$
$$v_{not} = (0,0,0,1)^T$$

You can notice some limitations of this method:
- High dimensionality. As the vocabulary expands, the dimension of the vectors increases at the same rate.
- Sparsity. Most elements are zero, which can be computationally intensive for some models to handle.
- Information loss. We cannot capture word similarity because the vectors are uncorrelated with each other.

Given these drawbacks, we now introduce a more powerful word representation: embeddings, which assigns each word a short, dense vector that can somehow catch word similarity.

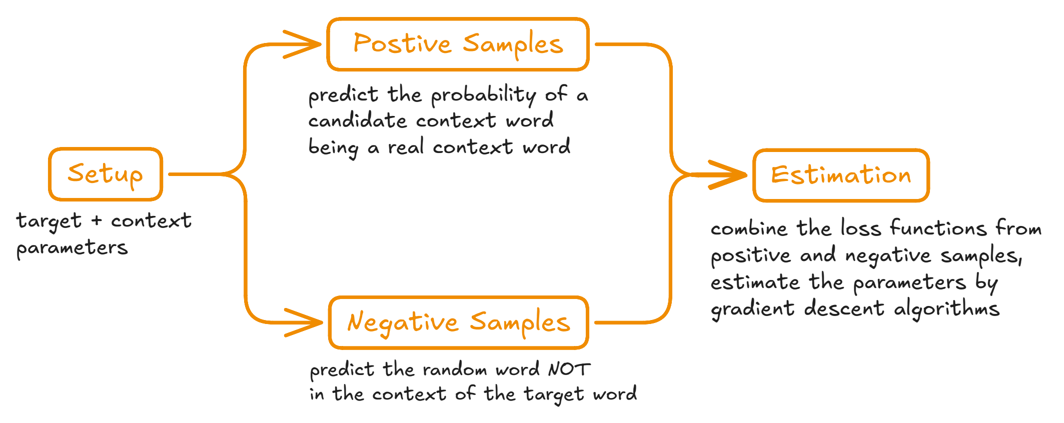
Word2Vec

The logic of word embeddings is based on a linguistic theory called distributional hypothesis, which states that "Words that appear in similar contexts tend to have similar meanings". For example, medicine and hospital, universe and planet.

Word2vec (Mikolov et al., 2013a, Mikolov et al., 2013b) is a particularly well-known algorithm for the construction of word embeddings. The intuition behind it is that instead of counting how often each word w occurs near, say, 'to', we'll instead train a classifier on a binary prediction task: "Is word w likely to show up near 'to'?" We don't actually care about this prediction task; instead we'll take the learned classifier weights (or parameters) as the word embeddings.

Skip-gram

Use logistic regression, for each target word, predict its neighboring context words as postive examples, while sampling random unrelated words as negative samples to push them away in the embedding space, thereby learning meaningful word embeddings that encode semantic relationships.



Setup
target + context parameters
Postive Samples
predict the probability of a candidate context word being a real context word
Negative Samples
predict the random word NOT in the context of the target word
Estimation
combine the loss functions from positive and negative samples, estimate the parameters by gradient descent algorithms

Setup

Endow each word w in the vocabulary with an embedding vector $\rho_w \in \mathbb{R}^K$ and a context vector $\alpha_w \in \mathbb{R}^K$ where $K \ll V$.

ρ_{to}	α_{to}
ρ_{be}	α_{be}
ρ_{or}	α_{or}
ρ_{not}	α_{not}

Postive Samples

1. Define the pairs of target word and context word (w, c) from the document.

target	context
to	be
be	to
be	or
or	be
or	not
not	or
not	to
to	not
to	be
be	to

2. Build a logistic regression model that predicts the probability of a candidate context word c being a real context word for target word w .

$$P(c \in C(w)|w) = \frac{1}{1+exp(-\underline{\rho'_w \alpha_c})}$$

probability of c being a real context word for w ↑
→ dot product ↑ → $\boxed{\rho_w \approx \alpha_c}$ → word similarity ↑

3. Write the loss function for the positive samples. Essentially this is just the negative log-likelihood from the binary logistic regression setup:

$$l_{pos} = \sigma(\rho'_{to}\alpha_{be}) \cdot \sigma(\rho'_{be}\alpha_{to}) \cdot \sigma(\rho'_{be}\alpha_{or}) \cdots \sigma(\rho'_{be}\alpha_{to}) \qquad \sigma(x) = \frac{1}{1+exp(-x)}$$
$$L_{pos} = - \sum_{(w,c)\text{in positive samples}} \log(\sigma(\rho'_w\alpha_c))$$

Negative Samples

1. We randomly select M words from the entire vocabulary for each target word w.

target	negative (M = 1)
to	or
be	not
or	to
not	to
to	be
be	to

In practice, we would have thousands of words in the vocabulary, so the probability of selecting a real context word is low.

2. Model the probability of a negative sample word neg NOT in the context of target word w.

$$P(neg \notin C(w)|w) = 1 - P(neg \in C(w)|w) = \frac{1}{1+exp(\rho'_w\alpha_{neg})}$$

3. Similarly we write out the loss function for the negative samples

$$l_{neg} = \sigma(-\rho'_{to}\alpha_{or}) \cdot \sigma(-\rho'_{be}\alpha_{not}) \cdot \sigma(-\rho'_{or}\alpha_{to}) \cdots \sigma(-\rho'_{be}\alpha_{to})$$
$$L_{neg} = - \sum_{(w,neg)\text{in negative samples}} \log(\sigma(-\rho'_w\alpha_{neg}))$$

Estimation

1. Combine the loss function from positive and negative samples

$$L = L_{pos} + L_{neg}$$

2. Estimate $\hat{\rho}$ and $\hat{\alpha}$ by gradient descent

3. The embedding for word w could be
   (a) $\hat{\rho}_w$
   (b) $(\hat{\rho}_w + \hat{\alpha}_w)/2$

📚 *Additional Resources on Word2Vec*

- Book chapter. Jurafsky & Martin (2021). Vector Semantics and Embeddings.
- Original word2vec paper. Mikolov et al. (2013). Efficient Estimation of Word Representations in Vector Space.
- Paper. Rong (2016). word2vec Parameter Learning Explained.
- Blog post. Alammar (2019). The Illustrated Word2vec.
- Applied paper. Garg et al. (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes.