A gentle introduction to Word2Vec

TU Wien

Bayar Menzat

April 11, 2024

Introduction

Introduction to Word2Vec

Word2Vec generates low-dimensional, dense word embeddings. These embeddings cluster similar words together based on their contexts. By analyzing words' local environments, Word2Vec captures both their semantic and syntactic characteristics. The method uses neural networks to efficiently produce these embeddings from large text corpora.

Word2Vec Overview

Word2Vec uses neural networks to learn word embeddings.

- The Skip-gram model predicts the context given a target word.
- It focuses on nearby words within a text window.
- It's a predictive model that infers from the local context of words.

Optimization with Softmax

In Word2Vec, the softmax function is used to optimize the probability of predicting a context word o given a center word c.

$$p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Here, v_c is the vector representation of the center word, u_o is the vector for the context word, and V is the vocabulary. The context words are selected from within a window of m words surrounding the center word.

Training Word2Vec with Gradient Descent

The training process involves minimizing the negative log probability of context words given a target word. Gradient descent is used to adjust word vectors and reduce the loss function iteratively.

Objective Function

The goal of training Word2Vec is to make actual words used in text (the context words) as probable as possible, given a particular word (the center word).

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\left[\sum_{-m\leq j\leq m, j\neq 0} \log p(w_{t+j}|w_t;\theta)\right]$$

- θ represents the model parameters (word vectors).
- T is the total number of words in the corpus.
- t indexes the words in the corpus, from 1 to T.
- m is the size of the training context window.
- w_t is the center word; w_{t+j} are context words.
- $p(w_{t+j}|w_t;\theta)$ is the conditional probability.
- Objective: Minimize the negative log probability average.

Implementing Gradient Descent

Gradient Descent is used to minimize the cost function $J(\theta)$. It finds the parameters (θ) that result in the minimum value of the cost function. Here's how it works:

- θ: The set of parameters (weights in a neural network) we are optimizing.
- α: Learning rate, which controls the step size in each iteration to reach the minimum.
- $\nabla_\theta J(\theta)$: Gradient of the cost function, indicating the direction of the steepest increase in the cost function space.

The parameter update rule is given by:

$$\theta_{\text{new}} = \theta_{\text{old}} - \alpha \nabla_\theta J(\theta)$$

In each iteration, θ is updated to move in the direction that reduces $J(\theta)$ the most, progressively leading to the minimum cost.

Word2Vec Toy Model: Understanding "Vienna" as Center Word

- The corpus for our example is: "Vienna is super."
- **Encode "Vienna":** Vienna, is and super are encoded using one-hot encoded vectors
- **Embedding Lookup:** Use the one-hot vector of "Vienna" to fetch its corresponding embedding vector from the weight matrix W, effectively translating the sparse representation into a dense, meaningful feature vector (h).
- **Predict Context:** Utilize the embedding (h) to predict context words by multiplying with the context matrix W', resulting logits (u) and use softmax function to obtain probabilities (y_{pred}).
- **Objective:** Maximize the probability of the actual context word ("is") in the predicted context, reflecting how well "Vienna" is associated with its surrounding words.

Simplified One-hot Encoding

We encode our vocabulary (len(V)=3) as follows:

1. **One-hot Encoding**:
    - "Vienna": [1, 0, 0].
    - "is": [0, 1, 0].
    - "super": [0, 0, 1].

Weight Matrices

1. **Weight Matrices**:
    - With 3 words and 2-dimensional embeddings, we have:
        - W (input to hidden layer): 3x2 matrix

$$\begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \\ 0.5 & 0.6 \end{bmatrix}$$

        - W' (hidden to output layer): 2x3 matrix

$$\begin{bmatrix} 0.55 & 0.65 & 0.75 \\ 0.60 & 0.70 & 0.80 \end{bmatrix}$$

Summary

In this setup:

- Use "Vienna"'s one-hot encoding to select its embedding from W.
- The selected embedding helps predict context words in W', like "is".

Getting Embeddings

Multiply one-hot encoding by W to get the "Vienna" embedding:

$$\begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \\ 0.5 & 0.6 \end{bmatrix} = \begin{bmatrix} 0.1 & 0.2 \end{bmatrix}$$

Calculating Output Vector

Using the embedding for "Vienna" to predict context words:

$$\begin{bmatrix} 0.1 & 0.2 \end{bmatrix} \times \begin{bmatrix} 0.55 & 0.65 & 0.75 \\ 0.60 & 0.70 & 0.80 \end{bmatrix} = \begin{bmatrix} 0.17 & 0.19 & 0.21 \end{bmatrix}$$

Softmax Probabilities

Softmax probabilities:

- $y_{\text{pred}} = [0.33, 0.34, 0.33]$ for words in the vocabulary.

Error Indicator Vector

Actual context word vector for "is": $y_{\text{actual}} = [0, 1, 0]$

- Error Indicator $\text{EI} = y_{\text{pred}} - y_{\text{actual}}$
- $\text{EI} = [0.33, -0.66, 0.33]$

Backpropagation in Word2Vec

In the backpropagation process of neural networks like Word2Vec, the chain rule is used to calculate the gradient of the loss function with respect to the weights, allowing us to understand how changes in the weights affect the overall prediction error.

Outer Product Calculation for W'

The outer product in updating W' is used to calculate the gradient of the loss function with respect to each element in W'. This is determined by how each element of the embedding vector h and the error indicator vector EI contribute to the loss.

Given $h = [0.1, 0.2]$ for "Vienna" and $EI = [0.3234, -0.6668, 0.3434]$:

$$\mathbf{dW'} = \mathbf{h} \otimes \mathbf{EI} = \begin{bmatrix} 0.1 \times 0.3234 & 0.1 \times -0.6668 & 0.1 \times 0.3434 \\ 0.2 \times 0.3234 & 0.2 \times -0.6668 & 0.2 \times 0.3434 \end{bmatrix}$$

$$= \begin{bmatrix} 0.03234 & -0.06668 & 0.03434 \\ 0.06468 & -0.13336 & 0.06868 \end{bmatrix}$$

Outer Product Calculation for W

The outer product for W involves the center word vector and the gradient of the loss with respect to the outputs

$$\mathbf{dW} = \text{center\_word\_vector} \otimes (\mathbf{W}' \times \mathbf{EI})$$

where EI is previously calculated and W' is the context matrix. This operation identifies how much the word "Vienna" should adjust its vector to minimize error in prediction.

$$\mathbf{dW} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \otimes \left(\begin{bmatrix} 0.5177 & 0.7167 & 0.7157 \\ 0.5353 & 0.8334 & 0.7313 \end{bmatrix} \times \begin{bmatrix} 0.3234 \\ -0.6668 \\ 0.3434 \end{bmatrix} \right)$$

Resulting in the gradient $\mathbf{dW}$ for updating W:

$$\begin{bmatrix} -0.0647 & -0.1315 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \end{bmatrix}$$

Update W' and W

For updating W', adjust it using the gradient $\mathbf{dW}'$ and a learning rate, e.g., $\eta = 1$.

$$W'_{\text{new}} = W' - \eta \times \mathbf{dW}'$$

Similarly, adjust W using its gradient to correct the prediction.

$$W_{\text{new}} = W - \eta \times \mathbf{dW}$$

Updated Matrices

W matrix (after update):

$$\begin{bmatrix} 0.098 & 0.198 \\ 0.3 & 0.4 \\ 0.5 & 0.6 \end{bmatrix}$$

W' matrix (after update):

$$\begin{bmatrix} 0.5177 & 0.7167 & 0.7157 \\ 0.5353 & 0.8334 & 0.7313 \end{bmatrix}$$

Comparison of Probabilities

We calculate "Vienna" embedding again, multiply it by the context matrix and apply the softmax function to obtain the updated prediction probabilities.

- Old probabilities: $[0.3234, 0.3332, 0.3434]$

- New probabilities after updating the model: $[0.3183, 0.3443, 0.3374]$

Conclusion
The updated probabilities show a decrease for the first and third words and an increase for "is", which aligns with the objective of maximizing the probability of a context word to predict a centre word.

Implementation (see notebook)

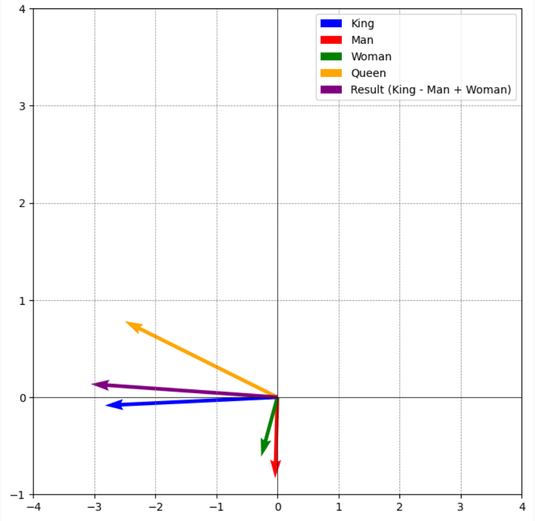

Figure 1: Embedding king - man + woman using word2vec and a toy dataset