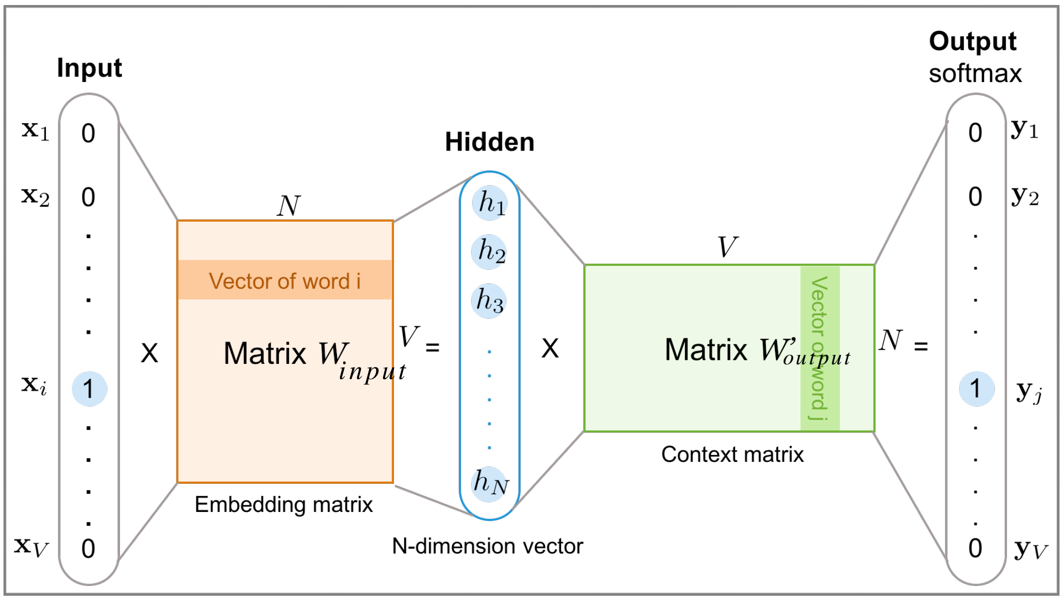
Assignment 2

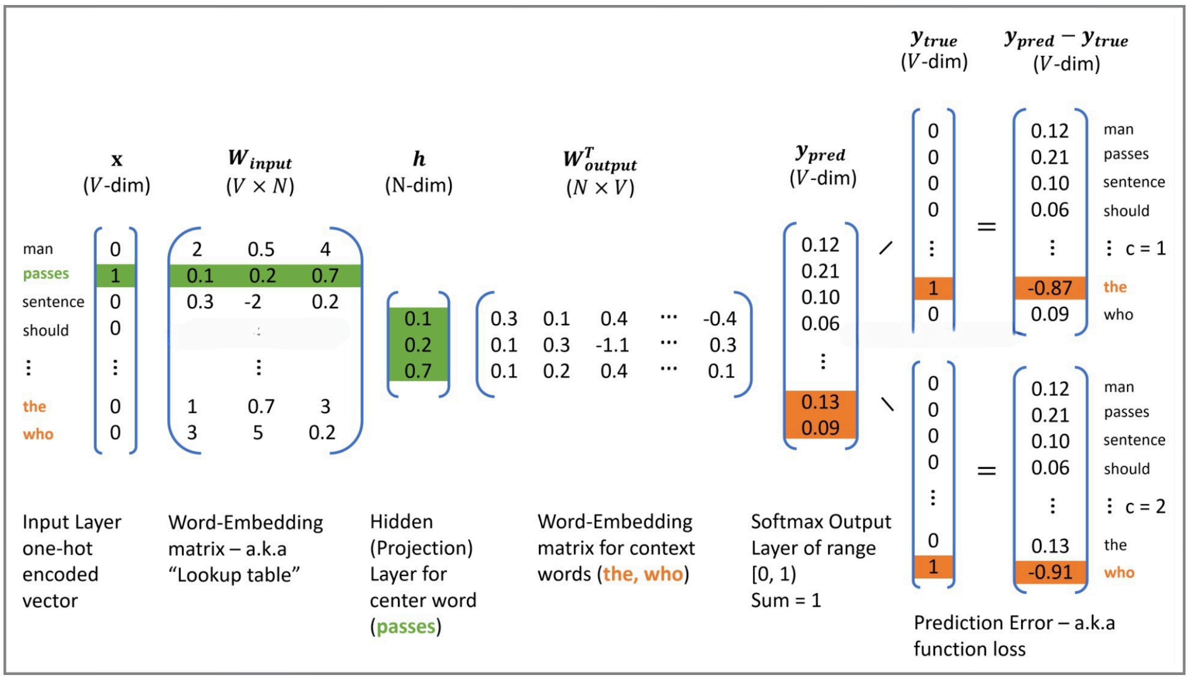
Practical Deep Learning for Language Processing

Dr. Aseem Behl
School of Business and Economics
University of Tübingen

Sunday, 4. December 2022

Product Recommendations with Fast Word2vec




$\mathbf{x}$ (V-dim)
W_{input} ($V \times N$)
h (N-dim)
W^T_{output} ($N \times V$)
y_{pred} (V-dim)
y_{true} (V-dim)
$y_{pred} - y_{true}$ (V-dim)
man
passes
sentence
should
the
who
c = 1
c = 2
Input Layer one-hot encoded vector
Word-Embedding matrix – a.k.a "Lookup table"
Hidden (Projection) Layer for center word (passes)
Word-Embedding matrix for context words (the, who)
Softmax Output Layer of range [0, 1) Sum = 1
Prediction Error – a.k.a function loss

In the lecture, we had implemented vanilla *Word2vec*. As shown in the figure above, the main idea is to optimise our model so that if it is queried with a word, it should return high probabilities for all the context C words. That is, training set for our *Word2vec* models consists of $2C$ samples per input word

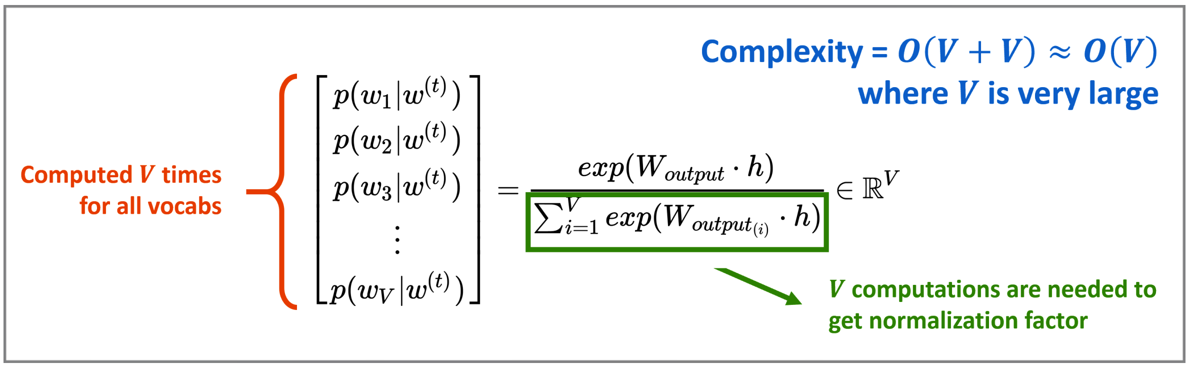
$(w_t : w_{t-c}, \cdots, w_t : w_{t-1}, w_t : w_{t+1}, \cdots, w_t : w_{t+c})$. In order to optimise our neural network model, we had used the following loss function:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\le j\le c, j\ne 0} \log p(w_{t+j} \mid w_t;\, \theta)$$

where $p(w_{t+j} \mid w_t;\, \theta)$ is a probability of observing w_{t+j} given w_t with parameters of the neural network θ. In vanilla *Word2vec*, the probability is computed with `softmax` function as its final activation. Replacing p in the equation above, our cost function is:

$$J(\theta; w^{(t)}) = -\sum_{c=1}^{C} log \frac{exp(W_{output_{(c)}} \cdot h)}{\sum_{i=1}^{V} exp(W_{output_{(i)}} \cdot h)}$$

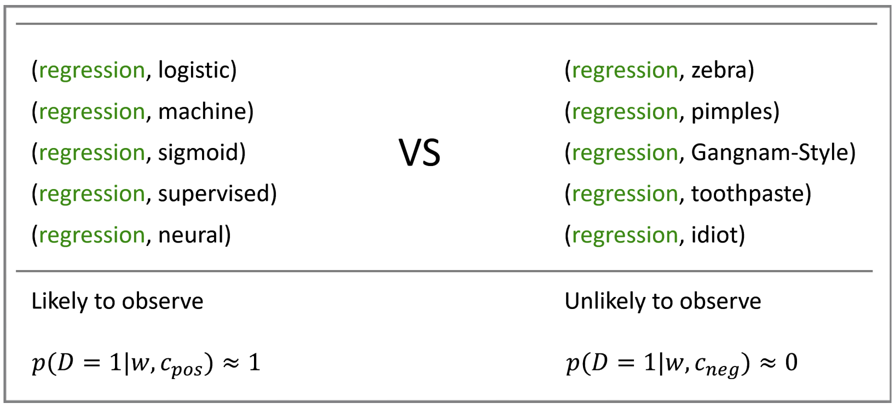
Where T is the size of the training samples, C is the window size, V s the size of unique vocabulary in the corpus, and W_{input}, W_{output} and h as shown in the figure above.
In the equation above, the fraction inside the summation of log, yields the probability distribution of all V-vocabs in the corpus, given the input word.


Complexity = $O(V + V) \approx O(V)$
where V is very large
Computed V times for all vocabs
$\begin{bmatrix} p(w_1|w^{(t)}) \\ p(w_2|w^{(t)}) \\ p(w_3|w^{(t)}) \\ \vdots \\ p(w_V|w^{(t)}) \end{bmatrix} = \frac{exp(W_{output} \cdot h)}{\boxed{\sum_{i=1}^{V} exp(W_{output_{(i)}} \cdot h)}} \in \mathbb{R}^V$
V computations are needed to get normalization factor

There is an issue with the vanilla *Word2vec* — `softmax` is computationally very expensive, as it requires scanning through the entire output embedding matrix W_{output} to compute the probability distribution of all V words, where V can be hundreds of thousands or million words. Furthermore, the normalization factor in the denominator also requires V operations. In mathematical context, the normalisation factor needs to be computed for each probability $p(w_{context} \mid w_{center})$ making the algorithm complexity $= O(V \times V)$. However, when implemented on code, the normalisation factor is computed only once and cached as a Python variable, making the algorithm complexity $= O(V + V) \approx O(V)$. This is possible because normalisation factor is the same for all words. However, this is still low.

Faster Word2Vec model

To avoid calculating high-dimensional `softmax`, we will rephrase the problem into a set of independent binary classification tasks. Instead of using `softmax` to compute the V-dimensional probability distribution of observing an output word given an input word, $p(w_O \mid w_I)$, the model uses `sigmoid` function to learn to differentiate the actual context words (positive) from randomly drawn words (negative). Assume that the centre word is *"regression"*. It is likely to observe *"regression" + {"logistic", "machine", "sigmoid", "supervised", "neural"}* pairs, but it is unlikely to observe *"regression" + {"zebra", "pimples", "Gangnam-Style", "toothpaste", "idiot"}*. The model maximises the probability $p(D = 1 \mid w, c_{pos})$ of observing positive pairs, while minimising the probability $p(D = 1 \mid w, c_{neg})$ of observing

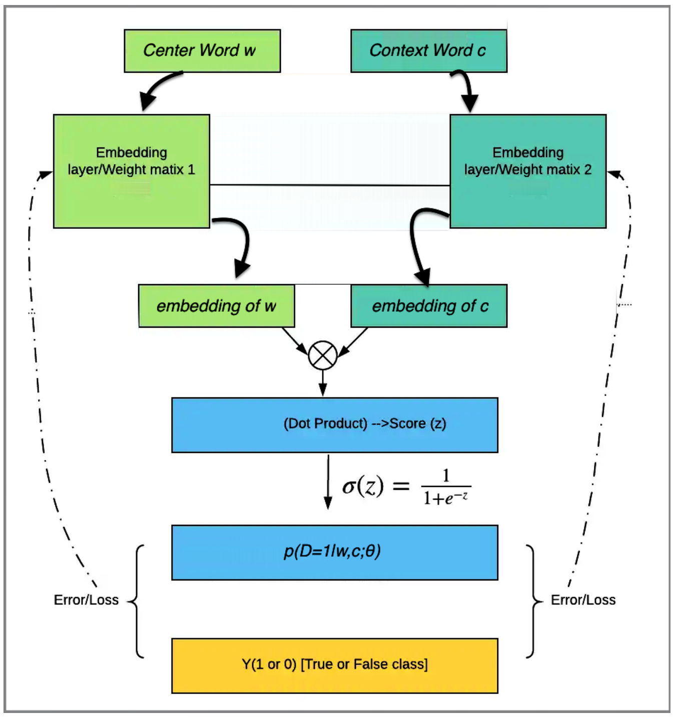


(regression, logistic)		(regression, zebra)
(regression, machine)		(regression, pimples)
(regression, sigmoid)	VS	(regression, Gangnam-Style)
(regression, supervised)		(regression, toothpaste)
(regression, neural)		(regression, idiot)
Likely to observe		Unlikely to observe
$p(D = 1 \mid w, c_{pos}) \approx 1$		$p(D = 1 \mid w, c_{neg}) \approx 0$

negative pairs. The idea is that if the model can distinguish between the likely (positive) pairs vs unlikely (negative) pairs, good word vectors will be learned.

This trick converts converts the original multi-classification task into binary-classification task. The new objective is to predict, for any given word-context pair (w, c) whether the word c is in the context window of the the centre word w or not. Since the goal is to identify a given word as True (positive, D=1) or False (negative, D=0), we use `sigmoid` function instead of `softmax` function. The probability of a word c appearing within the context of the centre word w can be defined as the following:

$$p(D = 1 \,|\, w, c; \theta) = \frac{1}{1 + exp(-embed_1(w) \cdot embed_2(c))} \in \mathbb{R}^1$$

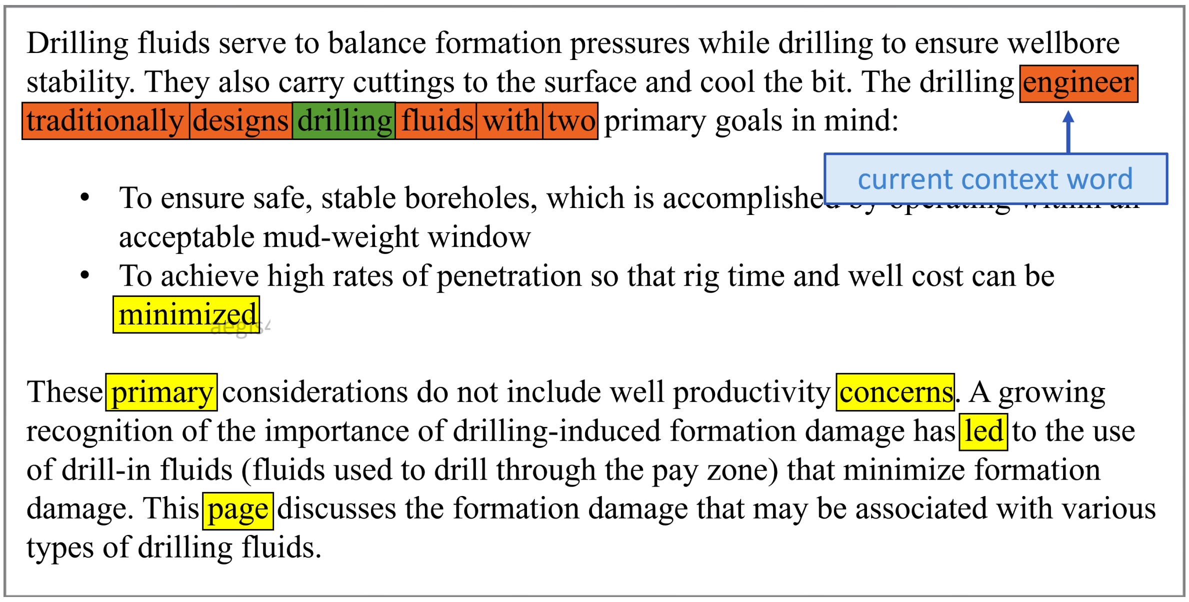


Center Word w
Context Word c
Embedding layer/Weight matix 1
Embedding layer/Weight matix 2
embedding of w
embedding of c
(Dot Product) -->Score (z)
$\sigma(z) = \frac{1}{1+e^{-z}}$
p(D=1|w,c;θ)
Error/Loss
Error/Loss
Y(1 or 0) [True or False class]

where c is the word you want to know whether it came from the context window or not, θ are the two independent embedding layers for all the words in the vocabulary when they are used as centre words and context words, respectively.

Illustrative example

For the purpose of illustration, consider the following paragraphs.


current context word

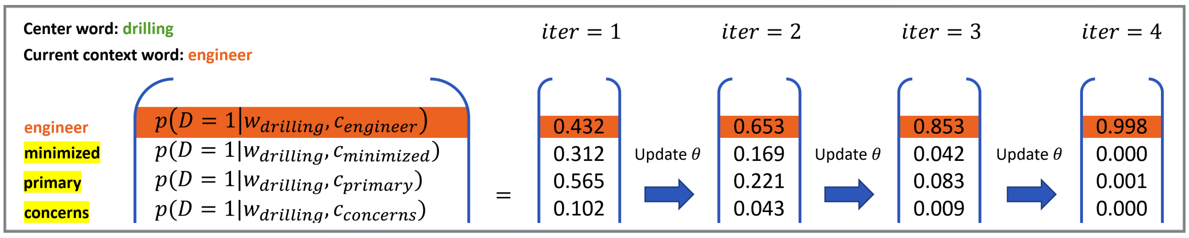
Drilling fluids serve to balance formation pressures while drilling to ensure wellbore stability. They also carry cuttings to the surface and cool the bit. The drilling engineer traditionally designs drilling fluids with two primary goals in mind:

- To ensure safe, stable boreholes, which is accomplished by operating within an acceptable mud-weight window
- To achieve high rates of penetration so that rig time and well cost can be minimized.

These primary considerations do not include well productivity concerns. A growing recognition of the importance of drilling-induced formation damage has led to the use of drill-in fluids (fluids used to drill through the pay zone) that minimize formation damage. This page discusses the formation damage that may be associated with various types of drilling fluids.

Assume that our center word w is *drilling*, window size is 3, and the number of negative samples $K = 5$. With the window size of 3, the contexts words are: *"engineer", "traditionally", "designs", "fluids", "with"*, and *"two"*. These context words are considered as positive labels (D=1). Our current context word c_{pos} is *engineer*. We also need negative words. We randomly pick 5-words from the noise distribution $P_n(w)$ of the corpus for each context word, and consider them as negative samples (D=0). For the current context word, *engineer*, the 5 randomly drawn negative words c_{neg} are: *"minimized", "primary", "concerns", "led"*, and *“page”*.

The idea here is that we are more likely to observe positive word pairs (w, c_{pos}) together than negative word pairs (w, c_{neg}) together in the corpus. The model attempts to maximize the the probability of observing positive pairs $p(c_{pos} | w) \rightarrow 1$ and minimize the probability of observing negative pairs $p(c_{neg} \mid w) \rightarrow 0$ simultaneously by iterating through the training samples and updating the weights θ.



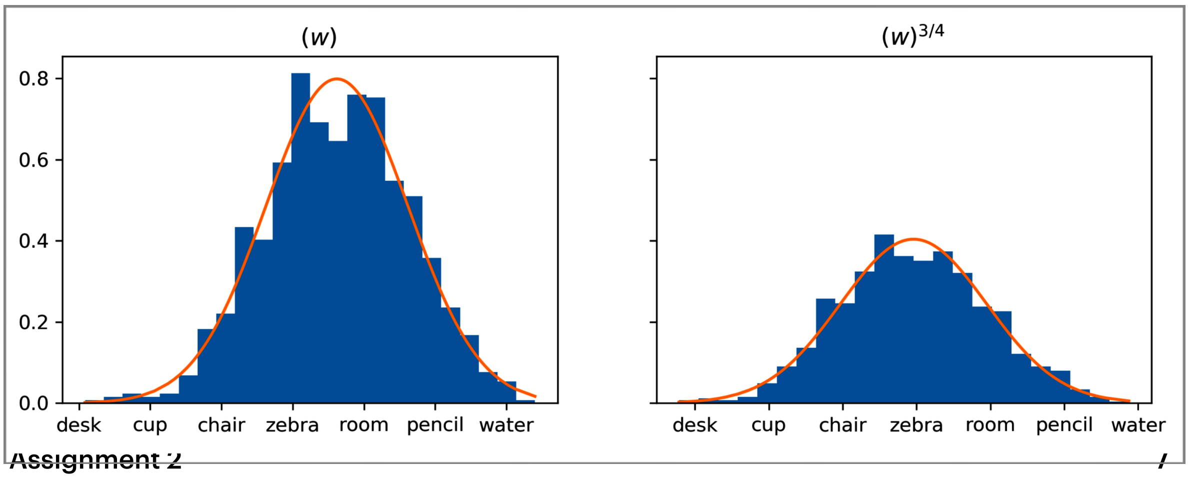
By the time the output probability distribution is nearly one-hot-encoded as in $iter = 4$ of the above figure, weight matrices are optimised and good word vectors are learned.

Generating negative samples

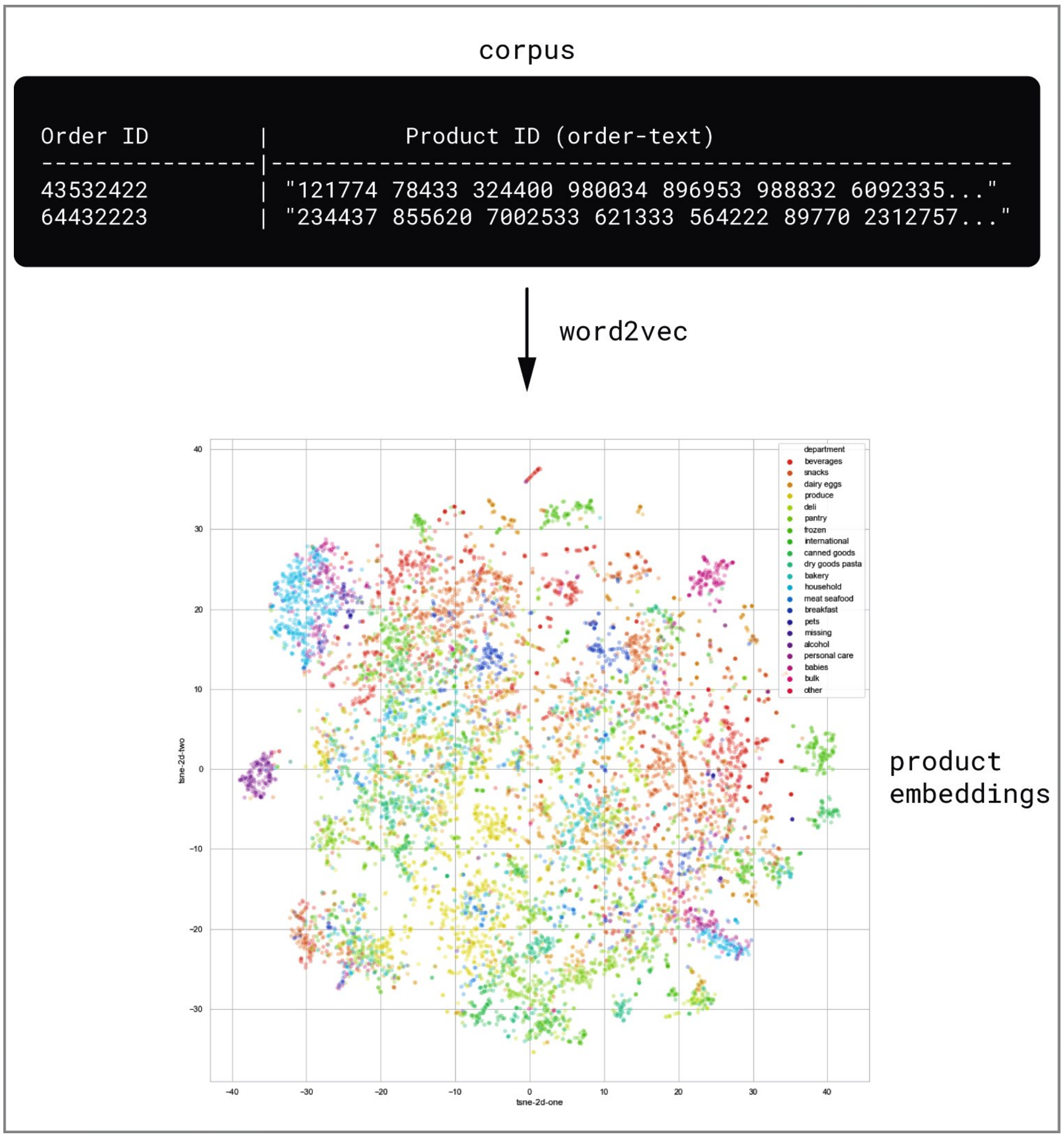
When generating negative samples, it is better to use words with less frequency more often than its actual frequency. To be specific, we adjust the negative sampling probability to be

$$P_n(w_i) = \frac{f(w_i)^{3/4}}{\sum f(w_i)^{3/4}}$$

where $f(w_i)$ is the frequency of the ith word. Raising the unigram distribution to the power of $3/4$ has an effect of smoothing out the distribution. It attempts to combat the imbalance between common words and rare words by decreasing the probability of drawing common words, and increasing the probability drawing rare words.



Product Recommendations with Word2vec



```
Order ID        |            Product ID (order-text)
----------------|-----------------------------------------------------
43532422        | "121774 78433 324400 980034 896953 988832 6092335..."
64432223        | "234437 855620 7002533 621333 564222 89770 2312757..."
```

In natural language sentences, words that are related appear together. The word *"steak"* is more likely to appear close to words such as *rosemary, salt, pepper, oil and butter*. In this sense, these words are thematically related. Similarly, we can exploit the fact that products appearing in a sequence close together in the order basket of a online retailer are likely share some similarity. A person browsing for gardening equipment is probably not looking at surfboards and vice versa.

Thus, given a database of customer orders in which each order typically includes multiple products. We can interpret each order as a sentence and each product as a word and straightforwardly apply the standard *word2vec* model to learn product embeddings, as shown in the figure above. This simple approach is known to produce useful product embeddings that capture purchasing or browsing patterns and reveal the affinity between products that are perceived or used in similar ways by customers.

Dataset

We will use the Online Retail Data Set for this assignment. There are 4,372 customers in our dataset. For each of these customers we will extract their buying history. In other words, we can have 4,372 sequences of purchases. We will set aside a 10% of the customers randomly from our dataset for testing purpose. Therefore, we will use data of 90% of the customers to train the word2vec embeddings model. Let's split the data.

Tasks

1. Implement the fast Word2Vec training algorithm as described in the previous section.

2. Train the complementary product embeddings on the order baskets of the training dataset.

3. Write a function `find_complementary_items()` which takes as input an item and return the top most similar items from our online retail dataset.

4. Create a function `purchase_history_embeddings()` to obtain customer embeddings by averaging embeddings of the last $n_{history}$

products associated with the customer purchase history.

5. Validate your model using examples in the test set. Compute purchase history embeddings for customers in test set. Using `find_complementary_items()` function get products suggestions based on the last $n_{history}$ purchases for the customer in test set.

The task is worth **15 points** and is due on **Jan 10, at 9:00 CET**. For submission, you only need to upload your notebook to ILIAS. Run all cells, and do not clear out the outputs, before submitting. In order to receive credit for the assignment, please be prepared to present your solutions during the lecture on **Tuesday, January 17**. You may be asked other questions from the lecture material related to the assignment.