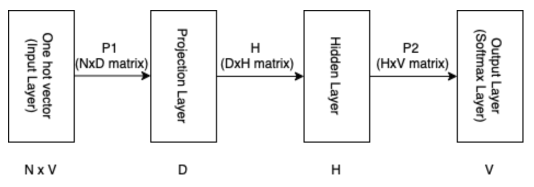
Distributed Representations of Words and Phrases and their Compositionality

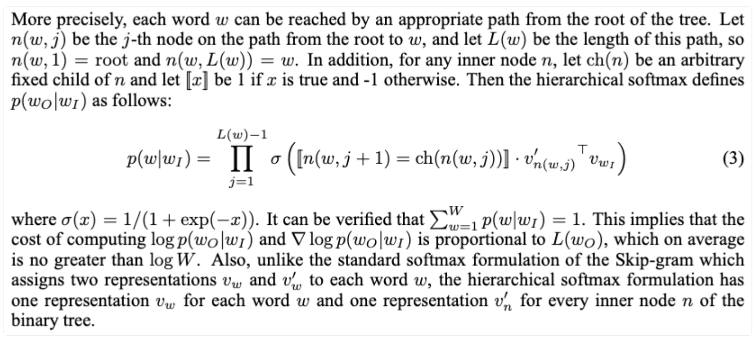
- [[📚 1310.4546.pdf]]
- **Basic Idea**:
  - In the earlier Word2Vec paper, the computational complexity of a NNLM was reduced by getting rid of the hidden layer. Can we further reduce this? (Hint: Replace hierarchical softmax with Noise Contrastive Estimation)
  - How do we improve learning by balancing frequently occurring words? Or learn better representations for rarely occurring words?
  - How do we also get representations for phrases like "Air Canada" (an airline company) which has completely different meaning from words "Air" and "Canada"? How do you statistically identify relevant phrases which represent all together different meaning from the constituent words?
  - Additional basic vector operations on word vectors such as `vec("Russia") + vec("River")` should be close to `vec("Volga River")`.
- **Skip-gram Model**:
  - 

  - There are two kinds of word representations that are being learnt corresponding to the projection matrix $P1$ and $P2$ (a.k.a. as "input" (v_w) and "output" (v'_w) representation). The final word embedding can be either of two, or even the sum/average of the two embeddings.
  - Given a sequence of training words $w_1, w_2, ..., w_T$, the training objective to maximize is given by:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} logp(w_{t+j}|w_t)$$

    Note that c is the context window length and higher value leads to higher accuracy due to more training examples but at the cost of increased training time.
  - The probability $p(w_{t+j}|w_t)$ is modelled using softmax function as follows:

$$p(w_O|w_I) = \frac{exp(v'^T_{w_O} v_{w_I})}{\sum_{w=1}^{W} exp(v'^T_w v_{w_I})}$$

    where W is the number of words in the vocabulary.
    Note that the softmax is defined using the cosine similarity between the input and output words because of the Distributional hypothesis and also if you do the maths of projection matrices, neural network and stuff it works out the same formula.
  - Since, the denominator sums over all the words in the vocabulary (typically $W = 10^5 - 10^7$), it is computationally expensive to calculate $p(w_{t+j}|w_t)$. Therefore, in order to overcome this, following solutions are used:
    - **Hierarchical Softmax**:
      - Introduced in Morin and Bengio, 2005
      - 🟡 **P3** The main advantage is that instead of evaluating W output nodes in the neural network to obtain the probability distribution, it is needed to evaluate only about log2(W) nodes.
        🟡 **P3** The hierarchical softmax uses a binary tree representation of the output layer with the W words as its leaves and, for each node, explicitly represents the relative probabilities of its child nodes. These define a random walk that assigns probabilities to words
        🟡 **P3**
        
        More precisely, each word w can be reached by an appropriate path from the root of the tree. Let $n(w, j)$ be the j-th node on the path from the root to w, and let $L(w)$ be the length of this path, so $n(w, 1) = \text{root}$ and $n(w, L(w)) = w$. In addition, for any inner node n, let $\text{ch}(n)$ be an arbitrary fixed child of n and let $[\![x]\!]$ be 1 if x is true and -1 otherwise. Then the hierarchical softmax defines $p(w_O|w_I)$ as follows:

$$p(w|w_I) = \prod_{j=1}^{L(w)-1} \sigma\Big([\![n(w, j+1) = \text{ch}(n(w,j))]\!] \cdot {v'_{n(w,j)}}^\top v_{w_I}\Big) \qquad (3)$$

        where $\sigma(x) = 1/(1+\exp(-x))$. It can be verified that $\sum_{w=1}^{W} p(w|w_I) = 1$. This implies that the cost of computing $\log p(w_O|w_I)$ and $\nabla \log p(w_O|w_I)$ is proportional to $L(w_O)$, which on average is no greater than $\log W$. Also, unlike the standard softmax formulation of the Skip-gram which assigns two representations v_w and v'_w to each word w, the hierarchical softmax formulation has one representation v_w for each word w and one representation v'_n for every inner node n of the binary tree.

        Note:

- $p(w|w_I)$ represents the probability of a random walker reaching a leaf node corresponding to word w starting from the root node.
- The above equation is based on the fact that:

$$1 - sigmoid(x) = sigmoid(-x)$$

🟡 **P3** The structure of the tree used by the hierarchical softmax has a considerable effect on the performance. Mnih and Hinton explored a number of methods for constructing the tree structure and the effect on both the training time and the resulting model accuracy [10]. In our work we use a binary Huffman tree, as it assigns short codes to the frequent words which results in fast training. It has been observed before that grouping words together by their frequency works well as a very simple speedup technique for the neural network based language models [5, 8]

- **Negative Sampling or Noise Contrastive Estimation (NCE)**
  - Idea is similar to FaceNet, Contrastive Learning etc.
  - Therefore, the original objective of the Skip-gram model is replaced by:

$$\max \left[\underbrace{log\sigma(v'^T_{w_O} v_{w_I})}_{\text{Positive example probability}} + \underbrace{\sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)} \left[log\sigma(-v'^T_{w_i} v_{w_I})\right]}_{\text{Negative example probability}} \right]$$

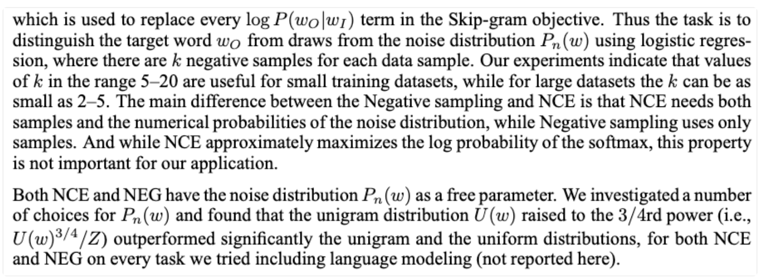
  Note for the calculation of negative example probability again the fact $1 - \sigma(x) = \sigma(-x)$ is used.

- The intuition is to contrast every positive target word against k negative words. The negative words are chosen according to the distribution $P_n(w)$ defined as follows:

$$U(w)^{3/4}/Z$$

  where $U(w)$ is the unigram distribution or the frequency of all the unigrams that appear in your training data.

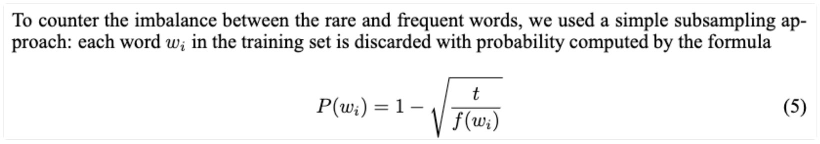
- 🟡 **P4**



which is used to replace every $\log P(w_O|w_I)$ term in the Skip-gram objective. Thus the task is to distinguish the target word w_O from draws from the noise distribution $P_n(w)$ using logistic regression, where there are k negative samples for each data sample. Our experiments indicate that values of k in the range 5–20 are useful for small training datasets, while for large datasets the k can be as small as 2–5. The main difference between the Negative sampling and NCE is that NCE needs both samples and the numerical probabilities of the noise distribution, while Negative sampling uses only samples. And while NCE approximately maximizes the log probability of the softmax, this property is not important for our application.

Both NCE and NEG have the noise distribution $P_n(w)$ as a free parameter. We investigated a number of choices for $P_n(w)$ and found that the unigram distribution $U(w)$ raised to the 3/4rd power (i.e., $U(w)^{3/4}/Z$) outperformed significantly the unigram and the uniform distributions, for both NCE and NEG on every task we tried including language modeling (not reported here).

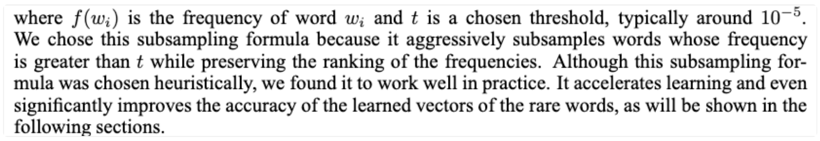
- **Subsampling of Frequent Words**
  - 🟡 **P4**



To counter the imbalance between the rare and frequent words, we used a simple subsampling approach: each word w_i in the training set is discarded with probability computed by the formula

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}} \quad (5)$$

🟡 **P5**



where $f(w_i)$ is the frequency of word w_i and t is a chosen threshold, typically around 10^{-5}. We chose this subsampling formula because it aggressively subsamples words whose frequency is greater than t while preserving the ranking of the frequencies. Although this subsampling formula was chosen heuristically, we found it to work well in practice. It accelerates learning and even significantly improves the accuracy of the learned vectors of the rare words, as will be shown in the following sections.

  - **Intuition for the above formula**:
    - $\sqrt{\frac{t}{f(w_i)}}$ is the probability that the word w_i is presented to the network while training. Intuitively, this probability should be inversely proportional to the frequency $f(w_i)$. Moreover, we take the square root because the typical vocabulary size used is $10^5 - 10^7$, therefore, the if we just take simple inverse without square the root, the probability of selection of rare word would be quite high and for a frequent word would be very low. So, the square root kind of shrinks the domain of possible probability values. However, since the frequent word will still be available a lot number of time, hence, there is a threshold t. Think of this threshold as $\frac{1}{\frac{f(w_i)}{t}}$, i.e., kind of normalizing the frequency.

- **Learning Phrases**
  - How do you statistically identify relevant phrases which represent all together different meaning from the constituent words?

🟡 **P5** To learn vector representation for phrases, we first find words that appear frequently together, and infrequently in other contexts. For example, “New York Times” and“Toronto Maple Leafs” are replaced by unique tokens in the training data, while a bigram “this is” will remain unchanged
"New York Times" or "Toronto Maple Leafs" will appear frequently together but infrequently in other usage of "New York" or "times". However, throughout the training set, there is a high possibility that "this is" will always occur together. Thereby, "this is" is not treated as a separate token.

- The authors use a simple data-driven approach for forming phrases using unigram and bigram counts using

$$\text{score}(w_i, w_j) = \frac{count(w_i w_j) - \delta}{count(w_i) \times count(w_j)}$$

- **Intuition for the above formula**:
  - The above formula is based on the naive definition of probability i.e.,

$$\mathbb{P}(\text{event}) = \frac{\text{Number of favourable outcomes}}{\text{Total number of possible outcomes}}$$

  - $count(w_i w_j)$ represents the number of times the words w_i and w_j appear together in the training set (favourable event).
  - $count(w_i) \times count(w_j)$ represent the number of times the words w_i and w_j could have appeared together if they were to be distributed independently assuming each occurrence of the word as separate/distinct (total number of possible outcomes).
  - Therefore, $\text{score}(w_i, w_j)$ just represents the probability of the words w_i, w_j occurring together.
  - δ is a discounting coefficient and prevents too many phrases consisting of very infrequent words to be formed.
- 🟡 **P6** The bigrams with score above the chosen threshold are then used as phrases. Typically, we run 2-4 passes over the training data with decreasing threshold value, allowing longer phrases that consists of several words to be formed.

- **Explanation for Additive Structure of Word Vectors**
  - 🟡 **P7** The additive property of the vectors can be explained by inspecting the training objective. The word vectors are in a linear relationship with the inputs to the softmax nonlinearity. As the word vectors are trained to predict the surrounding words in the sentence, the vectors can be seen as representing the distribution of the context in which a word appears. These values are related logarithmically to the probabilities computed by the output layer, so the sum of two word vectors is related to the product of the two context distributions. The product works here as the AND function: words that are assigned high probabilities by both word vectors will have high probability, and the other words will have low probability. Thus, if “Volga River” appears frequently in the same sentence together with the words “Russian” and “river”, the sum of these two word vectors will result in such a feature vector that is close to the vector of “Volga River”.

- **Important hyperparameters for Skip-gram model**
  - Choice of model architecture
  - Size (dimension) of the word vector
  - Subsampling rate
  - Size of the training window (c)

Unlinked References