**Word2Vec**

Word2vec is a computationally efficient predictive model for learning word embedding from raw text. Word embedding helps to place similar words near each other while irrelevant words away from each other. For example, it yields a vector approximating the representation for **vec(‘Rome’)** as a result of the vector operation:  **vec(‘Paris’**) – **vec(‘France’) + vec(‘Italy’).**

Word2vec comes in two flavors, the **Continuous Bag-of-Words** (CBOW) model and the **Skip-Gram** model.

**Continuous Bag-of-Words**:

The input to the model could be **wi − 2, wi − 1, wi + 1, wi + 2,** the preceding and following words of the current word we are at. The output of the neural network will be **wi**. Hence, we can think of the task as "predicting the word given its context". The number of words we use depends on setting for the context window size. It runs several times faster to train in comparison with the skip-gram, slightly better accuracy for the frequent words.

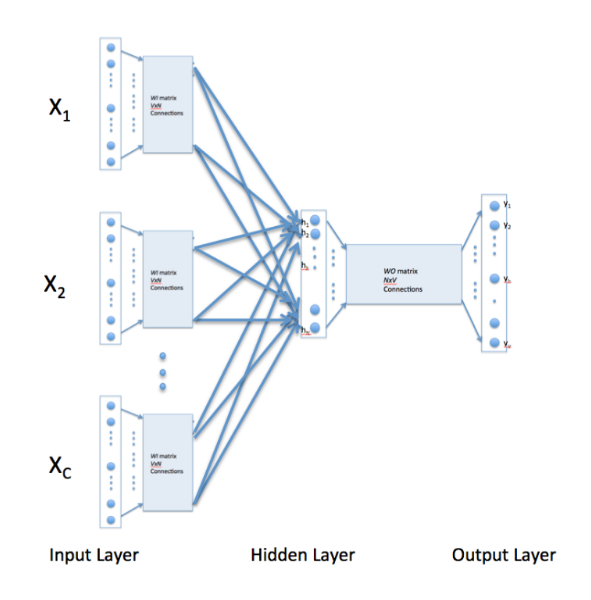


Figure 1 Continuous Bag of Words

**Skip-gram**:

The input to the model is **wi**, and the output could be **wi − 1, wi − 2, wi + 1, wi + 2**. Therefore, the task here is "**predicting the context given a word**". In addition, words that are more distant is given less weight by randomly sampling them. The contextwindow size is randomly chosen between 1 and max size for each training sample, resulting in words with the maximum distance being observed with a probability of 1/c while words directly next to the given word are always observed. It works well with small amount of the training data, represents well even rare words or phrases.

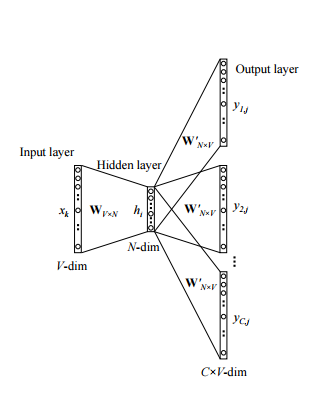


Figure 2 Skip Gram Model

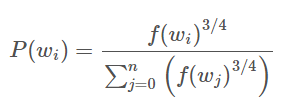
**Negative Sampling**

The size of our word vocabulary means that our Word2vec neural network has a tremendous number of weights, all of which would be updated slightly by every one of our billions of training samples. This results in need of lot of computation time. Negative sampling addresses this by having each training sample only modify a small percentage of the weights, rather than all of them.

With negative sampling, we are instead going to randomly select just a small number of “negative” words to update the weights for. (In this context, a “negative” word is one for which we want the network to output a 0 for). We will also still update the weights for our “positive” word, which are the positive input samples at hidden layer (context words in our skip gram model).

In general, we select negative samples in the range of 5-20 words works well for smaller datasets, and with 2-5 words for large datasets.

The probability for selecting a word as a negative sample is related to its frequency, with more frequent words being more likely to be selected as negative samples. Each word is given a weight equal to its frequency (word count) raised to the 3/4 power. The probability for a selecting a word is just its weight divided by the sum of weights for all words.



**Example**

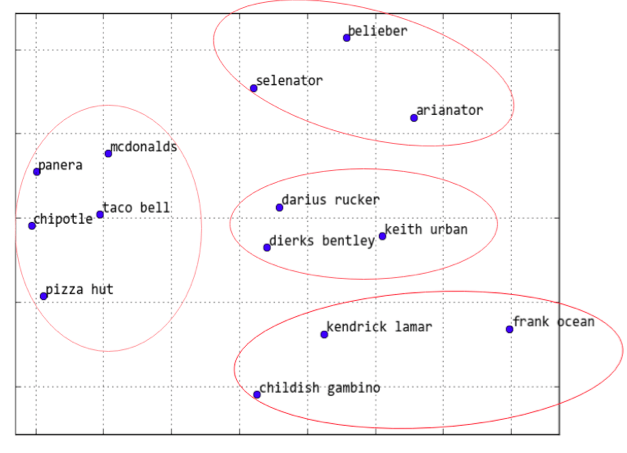


Figure Distance between Words

The above figure depicts the concept of word2vec wherein similar words are close to each other while dissimilar words are away from each other