Skip-gram Model

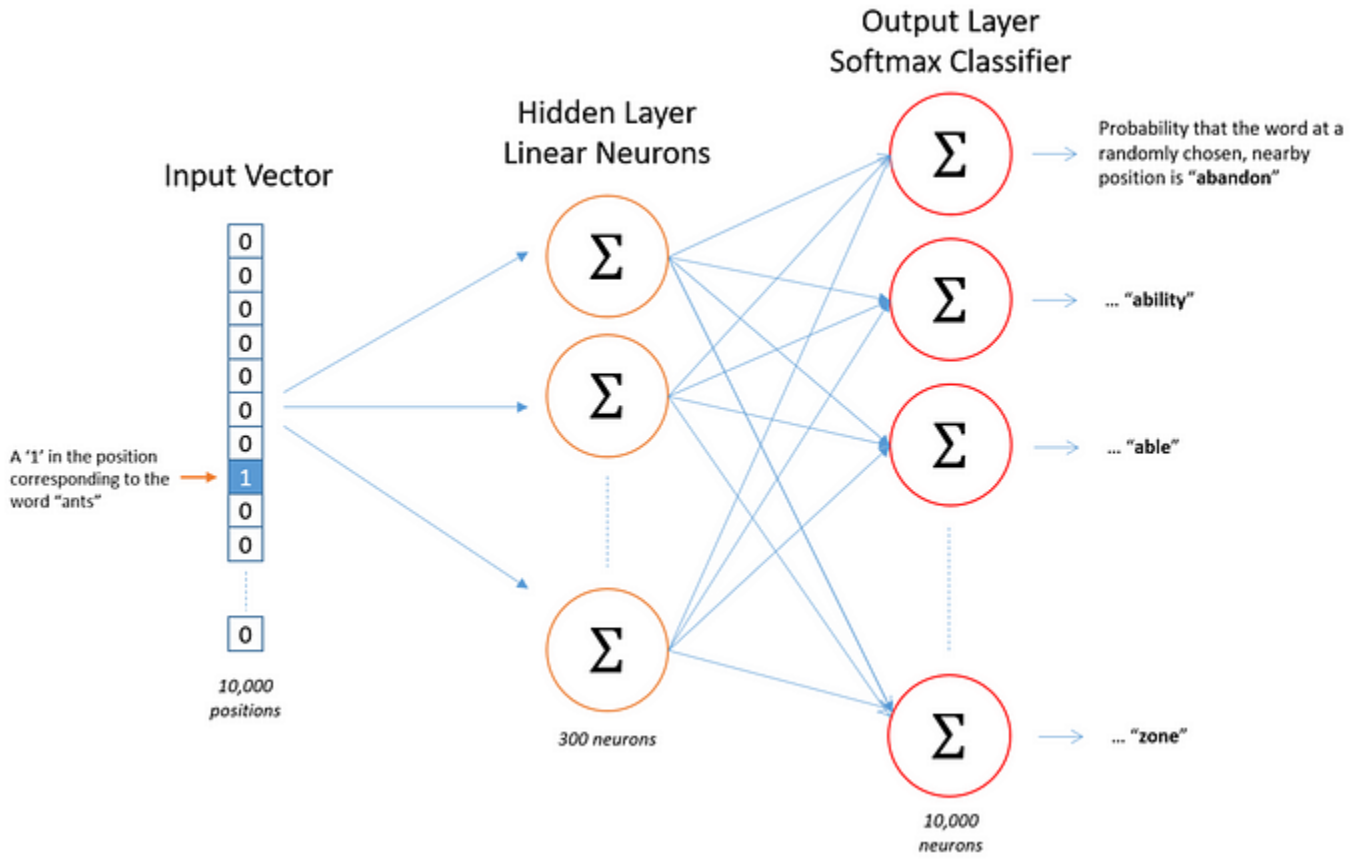
We'll do so by creating a "fake" task for the neural network to train. We won't be interested in the inputs and outputs of this network, rather the goal is actually just to learn the weights of the hidden layer that are actually the "word vectors" that we're trying to learn.

The fake task for Skip-gram model would be, given a word, we'll try to predict its neighboring words. We'll define a neighboring word by the window size — a hyper-parameter.

Source Text	Training Samples generated from source text			
I will have orange juice and eggs for breakfast	(will, I)	(will, have)	(will, orange)	
I will have orange juice and eggs for breakfast	(have, I)	(have, will)	(have, orange)	(have, juice)
I will have orange juice and eggs for breakfast	(orange, will)	(orange, have)	(orange, juice)	(orange, and)
I will have orange juice and eggs for breakfast	(juice, have)	(juice, orange)	(juice, and)	(juice, eggs)
I will have orange juice and eggs for breakfast	(and, orange)	(and, juice)	(and, eggs)	(and, for)
I will have orange juice and eggs for breakfast	(eggs, juice)	(eggs, and)	(eggs, for)	(eggs, breakfast)
I will have orange juice and eggs for breakfast	(for, and)	(for, eggs)	(for, breakfast)	

The word highlighted in yellow is the source word and the words highlighted in green are its neighboring words.

Given the sentence:
*"I will have orange **juice** and eggs for breakfast."*
and a window size of 2, if the target word is *juice,* its neighboring words will be *(have, orange, and, eggs)*. Our input and target word pair would be (juice, have), (juice, orange), (juice, and), (juice, eggs).
Also note that within the sample window, proximity of the words to the source word plays no role. So *have, orange, and,* and *eggs* will be treated the same while training.



Architecture for skip-gram model.

The dimensions of the input vector will be 1xV — where V is the *number of words in the vocabulary* — i.e one-hot representation of the word. The single hidden layer will have dimension VxE, where E is the size of the word embedding and is a hyper-parameter. The output from the hidden layer would be of the dimension 1xE, which we will feed into an softmax layer. The dimensions of the output layer will be 1xV, where each value in the vector will be the probability score of the target word at that position.

For a simple example, if we have 4 words — *mango, strawberry, city, Delhi* — in our vocabulary then we can represent them as following:

- Mango [1, 0, 0, 0]
- Strawberry [0, 1, 0, 0]
- City [0, 0, 1, 0]

- Delhi [0, 0, 0, 1]

According to our example if we have a vector [0.2, 0.1, 0.3, 0.4], the probability of the word being *mango* is 0.2, *strawberry* is 0.1, *city* is 0.3 and *Delhi* is 0.4.

The back propagation for training samples corresponding to a source word is done in one back pass. So for *juice,* we will complete the forward pass for all 4 target words *(have, orange, and, eggs)*. We will then calculate the errors vectors[1xV dimension] corresponding to each target word. We will now have 4 1xV error vectors and will perform an element-wise sum to get a 1xV vector. The weights of the hidden layer will be updated based on this cumulative 1xV error vector.

CBOW

The fake task in CBOW is somewhat similar to Skip-gram, in the sense that we still take a pair of words and teach the model that they co-occur but instead of adding the errors we add the input words for the same target word.

The dimension of our hidden layer and output layer will remain the same. Only the dimension of our input layer and the calculation of hidden layer activations will change, if we have 4 context words for a single target word, we will have 4 1xV input vectors. Each will be multiplied with the VxE hidden layer returning 1xE vectors. All 4 1xE vectors will be averaged element-wise to obtain the final activation which then will be fed into the softmax layer.

Skip-gram: works well with a small amount of the training data, represents well even rare words or phrases.
CBOW: several times faster to train than the skip-gram, slightly better accuracy for the frequent words.