Negative Sampling

Another problem we encounter when training a *word2vec* model naively is that our loss function -- cross-entropy loss or negative log-likelihood -- is very computationally expensive, which slows down training, especially on large datasets like ours.

Let's see why, by revisiting the negative log-likelihood loss function for the *Skip-gram* model. For simplicity let us focus on the loss which comes from predicting a single missing context word w_O from a single input word w_I. The loss term for this prediction is simply

$$-\log p_O$$

where p_O is our model's output prediction that the missing word is w_O. I.e. if w_O is, for example, the 537th word in the dictionary, then p_O is the corresponding 537th entry in the prediction vector which is the model's final output.

Now, by looking at the structure of our *Skip-gram* model -- an embedding layer, followed by a projection layer, followed by a softmax activation function -- and recalling the definition of the softmax activation, we can see that

$$p_O = \frac{\exp(P_O^T v_I)}{\sum_{j=1}^{V} \exp(P_j^T v_I)}$$

where the P_j are the rows of the projection matrix P in our *Skip-gram* model, v_I is the word embedding for the input word, and V is the number of words in the vocabulary.

There are two problems which arise from the

$$\sum_{j=1}^{V} \exp(P_j^T v_I)$$

term in the denominator, as follows.

1. It is expensive to compute this sum, which contains V terms (typically between 10,000 - 100,000) for every example.
2. Because the sum involves rows of the projection matrix corresponding to every word in the vocabulary, this means that for every example -- which is just a single word -- we end up backpropagating through every single word embedding in our whole vocabulary (rather than just the word itself, or its context words, about which the example contains the most information). This slows down training.

The way that the authors solve this problem in

https://arxiv.org/pdf/1310.4546

is to replace the softmax activation function at the end of the *Skip-gram* model with a *sigmoid activation function*, and to modify the loss function accordingly with a version of *Negative Sampling*. We will explain all of this below.

Once again, let us focus on the loss which comes from predicting a single missing context word w_O from a single input word w_I, which now looks like

$$-\log \tilde{p}_O - \sum_{i=1}^{k} \log(1 - \tilde{p}_{w_i})$$

whose components we will break down below.

1. $\tilde{p}$ is the "prediction vector" of our modified model, a vector whose entries are

$$\tilde{p}_i = \sigma(P_i^T v_I)$$

where σ is the *sigmoid activation function* which has formula

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

and replaces the entries of our softmax activation function.

Note that $\sigma(x)$ is close to 1 if x is large and positive and close to 0 if x is large and negative, so it mimics in the entries of the softmax activation in that regard, but means that our new "prediction vector" $\tilde{p}$ fails to have its entries add up to 1, meaning that it is not a true prediction vector. However, this turns out not to be a problem for reasons discussed below.

2. The first term

$$-\log \tilde{p}_O$$

measures how highly we predict that the missing context word was in fact the true label

Continue

Ask question

w_O (the *positive sample*).

3. The second term

$$-\sum_{i=1}^{k} \log(1 - \tilde{p}_{w_i})$$

samples (at random, in a way which we discuss in the next chapter) k **incorrect** missing context words $w_1, \cdots, w_k$ (the *negative samples*) for the input word w_I, and then measures how lowly we predict that the missing context word was one of these incorrect choices.

The reason that this works is that it forces the model to give high "predictions" for the correct missing context words and low "predictions" for incorrect missing context words. So, the fact that the "prediction vector" is not normalised to have its entries equal to 1 does not matter, because only the ratios of its entries matter.

Moreover, this solves the two problems which we described at the beginning, as follows.

1. We no longer have to compute a large sum over the whole vocabulary, which saves on computations.
2. For any given example, we now only backpropagate through the word embeddings for w_O and $w_1, \cdots w_k$ (where we typically pick k to be something like 20) rather than for every word in the vocabulary.

P.S. If you want a more rigorous review of why this all works, see the following paper on *Noise Contrastive Estimation*.

https://arxiv.org/pdf/1410.8251