Hints for Word2Vec based Sentiment Analysis

1. word2vec training

In this part, you will implement the word2vec models and train your own word vectors with stochastic gradient descent (SGD). First, write a helper function to normalize rows of a matrix in word2vec.py. In the same file, fill in the implementation for the softmax and negative sampling cost and gradient functions. Then, fill in the implementation of the cost and gradient functions for the skipgram model. When you are done, test your implementation by running python word2vec.py.

(a) Assume you are given a predicted word vector v_c corresponding to the center word *c* for skipgram, and word prediction is made with the softmax function found in word2vec models:

$$\hat{y} = p\left(o \middle| c\right) = \frac{\exp\left(u_o^T v_c\right)}{\sum_{w=1}^{V} \exp\left(u_w^T v_c\right)}$$

where w denotes the w-th word and u_w ($w = 1, \ldots, V$) is the "output" word vector for the w-th word. Assume cross entropy (CE) cost (i.e., negative log likelihood) is applied to this prediction and word o is the expected word.

Hint: It will be helpful to use simple notation. For instance, letting $\hat{y}$ be the vector of softmax predictions for every word, y as the expected word vector (the o-th element of the one-hot label vector is one), and the function is:

$$J_{softmax-CE}(o, v_c, U) = CE(y, \hat{y})$$

Where $U = [u_1, u_2, \ldots, u_V]$ is the matrix of all the output vectors.

(b) Assume you are using the negative sampling loss, *K* negative samples (words) are drawn, and they are **1, ⋯, *K***, respectively for simplicity of notation ($o \notin \{1, \ldots, K\}$). Again, for an expected word, o, the negative sampling loss function in this case is:

$$J_{neg-sample}\left(o, v_c, U\right) = -\log\left(\sigma\left(u_o^T v_c\right)\right) - \sum_{k=1}^{K} \log\left(\sigma\left(-u_k^T v_c\right)\right)$$

where $\sigma(\cdot)$ is the sigmoid function.

Note: the cost function here is the negative of what Mikolov et al had in their original paper, because we doing a minimization instead of maximization in our code.

(c) Recall that for skip-gram, the cost for a context centered around *c* is

$$J_{skip-gram}(w_{c-m\ldots c+m}) = \sum_{-m\le j\le m, j\neq 0} F(w_{c+j},\ v_c)$$

where $[w_{c-m}, \ldots, w_{c-1}, w_{c+1}, \ldots, w_{c+m}]$ is the set of context words for w_c, v_k and u_k is the "input" and "output" word vectors for w_k, $F\left(w_{c+j},\ v_c\right)$ can be replaced with $J_{softmax-CE}(o, v_c, U)$ or $J_{neg-sample}\left(o, v_c, U\right)$. Derive gradients for all of the word vectors for skip-gram.

(d) Now we are going to load some real data and train word vectors with everything you just implemented! We are going to use the Stanford Sentiment Treebank (SST) dataset to train word vectors, and later apply them to a simple sentiment analysis task. There is no additional code to write for this part; just run python run.py. When the script finishes, a visualization for your word vectors will appear. It will also be saved as word_vectors.png in your project directory. **Include the plot in your homework write up.**

2. Sentiment analysis

Now, with the word vectors you trained, we are going to perform a simple sentiment analysis. For each sentence in the Stanford Sentiment Treebank dataset, we are going to use the average of all the word vectors in that sentence as its feature, and try to predict the sentiment level of the sentence. The sentiment level of the phrases are represented as real values in the original dataset, here we'll just use five classes:

"very negative", "negative", "neutral", "positive", "very positive"

which are represented by 0 to 4 in the code, respectively. For this part, you will learn to train a softmax regressor with SGD, and perform train/dev validation to improve generalization of your regressor.

(a) Get a sentence feature. Fill in the implementation in softmaxreg.py.

(b) Fill in the hyperparameter selection code in sentiment.py to search for the "optimal" regularization parameter. **What value did you select? Report your train, dev, and test accuracies. Justify your hyperparameter search methodology in at most one sentence.**
Note: you should be able to attain at least 25% accuracy on dev.

(c) Plot the classification accuracy on the train and dev set with respect to the regularization value, using a logarithmic scale on the x-axis. This should have been done automatically. **Include** word2vec_acc.png **in your homework write up.**