Assignment #4

Due Time: Mid-night at December 27th 2017

1 word2vec (70 points)

(a) (5 points) 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

$$\hat{\boldsymbol{y}}_o = p(\boldsymbol{o}|\boldsymbol{c}) = \frac{\exp(\boldsymbol{u}_o^T \boldsymbol{v}_c)}{\sum_{w=1}^{W} \exp(\boldsymbol{u}_w^T \boldsymbol{v}_c)} \tag{1}$$

where $\boldsymbol{w}$ denotes the w-th word and $\boldsymbol{u}_w (w = 1, \ldots, W)$ are the "output" word vectors for all words in the vocabulary. Assume cross entropy cost is applied to this prediction and word $\boldsymbol{o}$ is the expected word (the o-th element of the one-hot label vector is one), derive the gradients with respect to $\boldsymbol{v}_c$.
Hint: It will be helpful to use simple notation. For instance, letting $\hat{\boldsymbol{y}}$ be the vector of softmax predictions for every word, $\boldsymbol{y}$ as the expected word vector, and the loss function

$$J_{softmax-CE}(\boldsymbol{o}, \boldsymbol{v}_c, \boldsymbol{U}) = CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) \tag{2}$$

Where $\boldsymbol{U} = [\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_W]$ is the matrix of all the output vectors. *Make sure you state the orientation of your vectors and matrices.*

(b) (5 points) As in the previous part, derive gradients for the "output" word vectors $\boldsymbol{u}_w$'s (including $\boldsymbol{u}_o$).

(c) (10 points) Repeat part (a) and (b) assuming we are using the negative sampling loss for the predicted vector v_c, and the expected output word is o. Assume that K negative samples (words) are drawn, and they are $\boldsymbol{1}, \cdots, \boldsymbol{K}$, respectively for simplicity of notation ($o \notin \{1, \ldots, K\}$). Again, for a given word, $\boldsymbol{o}$, denote its output vector as $\boldsymbol{u}_o$. The negative sampling loss function in this case is

$$J_{neg-sample}(\boldsymbol{o}, \boldsymbol{v}_c, \boldsymbol{U}) = -\log\left(\sigma(\boldsymbol{u}_o^T \boldsymbol{v}_c)\right) - \sum_{k=1}^{K} \log(\sigma(-\boldsymbol{u}_k^T \boldsymbol{v}_c)) \tag{3}$$

where $\sigma(\cdot)$ is the sigmoid function.
After you've done this, describe with one sentence why this cost function is much more efficient to compute than the softmax-CE loss (you could provide a speed-up ratio, i.e. the runtime of the softmax-CE loss divided by the runtime of the negative sampling loss).
Note: the cost function here is the negative of what Mikolov et al had in their original paper, because we are doing a minimization instead of maximization in our code.

(d) (10 points) Derive gradients for all of the word vectors for skip-gram given the previous parts and given a set of context words $[word_{c-m}, \ldots, word_{c-1}, word_c, word_{c+1}, \ldots, word_{c+m}]$, where m is the context size. Denote the "input" and "output" word vectors for $word_k$ as v_k and u_k respectively.

Hint: feel free to use $F(\boldsymbol{o}, \boldsymbol{v}_c)$ (where o is the expected word) as a placeholder for the $J_{softmax-CE}(\boldsymbol{o}, \boldsymbol{v}_c, \ldots)$ or $J_{neg-sample}(\boldsymbol{o}, \boldsymbol{v}_c, \ldots)$ cost functions in this part — you'll see that this is a useful abstraction for the coding part. That is, your solution may contain terms of the form $\frac{\partial F(\boldsymbol{o}, \boldsymbol{v}_c)}{\partial \ldots}$.

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

$$J_{skip-gram}(word_{c-m \ldots c+m}) = \sum_{-m \le j \le m, j \neq 0} F(\boldsymbol{w}_{c+j}, \boldsymbol{v}_c) \tag{4}$$

where w_{c+j} refers to the word at the j-th index from the center.

(e) (30 points) 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.

(f) (10 points) Show time! 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.

Note: The training process may take a long time depending on the efficiency of your implementation ***(an efficient implementation takes approximately an hour)****. Plan accordingly!*

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 (30 points)

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 said 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) (10 points) Implement a sentence featurizer. Fill in the implementation in softmaxreg.py. Sanity check your implementation with python softmaxreg.py.

(b) (15 points) 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) (5 points) 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** reg_acc.png **in your homework write up.**