Assignment 2: Word Vectors

Homework assignments will be done individually: each student must hand in their own answers. Use of partial or entire solutions obtained from others or online is strictly prohibited. Electronic submission on Canvas is mandatory.

1. **Written** (30 pts)
   (a) **Softmax** (5pts) Prove that softmax is invariant to constant offset in the input, that is, for any input vector $\mathbf{x}$ and any constant c,

$$\text{softmax}(\mathbf{x}) = \text{softmax}(\mathbf{x} + c)$$

   where $\mathbf{x} + c$ means adding the constant c to every dimension of $\mathbf{x}$. Remember that

$$\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

   Note: In practice, we make use of this property and choose $c = -\max_i x_i$ when computing softmax probabilities for numerical stability (i.e., subtracting its maximum element from all elements of x).

   (b) **Sigmoid** (5pts)Derive the gradients of the sigmoid function and show that it can be rewritten as a function of the function value (i.e., in some expression where only $\sigma(x)$, but not x, is present). Assume that the input x is a scalar for this question. Recall, the sigmoid function is:

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

   (c) **Word2vec**
      i. (5pts) Assume you are given a predicted word vector $\mathbf{v}_c$ corresponding to the center word c for skipgram, and the word prediction is made with the `softmax` function

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

      where o is the expected word, w denotes the w-th word and $\mathbf{u}_w$ (w = 1, ..., W) are the "output" word vectors for all words in the vocabulary. The cross entropy function is defined as:

$$J_{\text{CE}}(o, \mathbf{v}_c, U) = CE(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_i y_i \log(\hat{y}_i)$$

      where the gold vector $\mathbf{y}$ is a one-hot vector, the softmax prediction vector $\hat{\mathbf{y}}$ is a probability distribution over the output space, and $U = [u_1, u_2, ..., u_W]$ is the matrix of all the output vectors. Assume cross entropy cost is applied to this prediction, derive the gradients with respect to $\mathbf{v}_c$.
      ii. (5pts) As in the previous part, derive gradients for the "output" word vector $\mathbf{u}_w$ (including $\mathbf{u}_o$).

iii. (5pts) Repeat a and b assuming we are using the negative sampling loss for the predicted vector $\mathbf{v}_c$. Assume that K negative samples (words) are drawn and they are 1,...,K respectively. For simplicity of notation, assume ($o \notin \{1, ..., K\}$). Again for a given word o, use $\mathbf{u}_o$ to denote its output vector. The negative sampling loss function in this case is:

$$J_{\text{neg-sample}}(o, \mathbf{v}_c, U) = -\log(\sigma(\mathbf{u}_o^\top \mathbf{v}_c)) - \sum_{k=1}^{K} \log(\sigma(-\mathbf{u}_k^\top \mathbf{v}_c))$$

iv. (5pts) Derive gradients for all of the word vectors for skip-gram given the previous parts and given a set of context words $[\text{word}_{c-m}, ..., \text{word}_c, ..., \text{word}_{c+m}]$ where m is the context size. Denote the "input" and "output" word vectors for word k as $\mathbf{v}_k$ and $\mathbf{u}_k$ respectively.
Hint: feel free to use $F(o, \mathbf{v}_c)$ (where o is the expected word) as a placeholder for the $J_{CE}(o, \mathbf{v}_c ...)$ or $J_{neg\text{-}sample}(o, \mathbf{v}_c ...)$ 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 $\frac{\partial F(o, \mathbf{v}_c)}{\partial ...}$ Recall that for skip-gram, the cost for a context centered around c is:

$$\sum_{-m \le j \le m, j \neq 0} F(w_{c+j}, \mathbf{v}_c)$$

2. **Coding** (70 points)

(a) (5pts) Given an input matrix of N rows and D columns, compute the softmax prediction for each row using the optimization in 1.(a). Write your implementation in `utils.py`.

(b) (5pts) Implement the sigmoid function in `word2vec.py` and test your code.

(c) (45pts)Implement the word2vec models with stochastic gradient descent (SGD).

i. (15pts) Write a helper function `normalizeRows` in `utils.py` 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 skip-gram model. When you are done, test your implementation by running python `word2vec.py`.

ii. (15pts) Complete the implementation for your SGD optimizer in `sgd.py`. Test your implementation by running python `sgd.py`.

iii. (15pts) Implement the k-nearest neighbors algorithm, which will be used for analysis. The algorithm receives a vector, a matrix and an integer k, and returns k indices of the matrix's rows that are closest to the vector. Use the cosine similarity as a distance metric (`https://en.wikipedia.org/wiki/Cosine_similarity`). Implement the algorithm in `knn.py`. Print out 10 examples: each example is one word and its neighbors.

(d) (15pts)Load some real data and train your own word vectors.
Use the StanfordSentimentTreeBank data to train word vectors. Process the dataset and use the sgd function and word2vec to generate word vectors. Visualize a few word examples. 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. 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. In addition, the script should print the nearest neighbors of a few words (using the knn function you implemented in 2(g)). Include the plot and the nearest neighbors lists in your homework write up, and briefly explain those results.

3. **Submission Instructions** You shall submit a zip file named Assignment2_LastName_FirstName.zip which contains:

- a pdf(or jpg) file contains all your solutions for the Written part
- a pdf(or jpg) file contains the word vector plot(vector.png), a brief report of your knn results.
- python files(sgd.py, word2vec.py, run.py, knn.py, utils.py)