Assignment 2: MLP and 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. **Multi-Layer Perceptron (MLP)** (30 pts)
   (a) Preprocess the data: tokenization, feature extraction. You can reuse the code from Assignment 1.
   (b) (20 pts) Implement the MLP class
   (c) (10 pts) Implement mini-batch GD for MLP
   (d) Run all the code to make sure your implementation works.

2. **Word2vec - Written** (25 pts)
   (a) (5 pts) 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 expressions 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:

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

   (b) (5 pts) Assume you are given a predicted word vector $\mathbf{v}_c$ corresponding to the center word c for skip-gram, 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" (context) 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 $\mathbf{v}_c$.
   (c) (5 pts) Derive gradients for the "output" word vector $\mathbf{u}_w$ (including $\mathbf{u}_o$) in (b).
   (d) (5 pts) Repeat (b) and (c) 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:

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

(e) (5 pts) 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 $J_{\text{CE}}(o, \mathbf{v}_c...)$ or $J_{\text{neg-sample}}(o, \mathbf{v}_c...)$ 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:

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

3. **Word2vec - Coding** (45 points)

(a) (5pts) Implement Sigmoid function

(b) (10 pts) Implement Naive Softmax loss and gradient function for word2vec models.

(c) (10 pts) Implement Negative sampling loss function for word2vec models

(d) (10 pts) Implement Skip-gram model in word2vec

(e) (10 pts) 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`).

(f) Load some real data and train your own word vectors. Use the training 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.

(g) Run the jupyter notebook code to make sure your implementation works

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

- The jupyter notebook which includes all your code (and your written part), and a brief report of your knn results.
- (optional) a png (or jpg) file contains the word vector plot (vector.png).
- (optional) a pdf file contains all your solutions for the Written part.