

## 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.

**Instructions:** You shall submit a zip file named A2\_LastName\_FirstName.zip including:

- A jupyter notebook which includes all your code (and your written part). We provided some code templates. It is up to you to use the templates or not.
- (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.
- Your code may run for several hours depending on your implementation and your device. Please **start working on it as early as possible!**

### 1. Multi-Layer Perceptron (MLP) (30 pts)

- (a) (5 pts) Preprocess the data: tokenization, feature extraction. You can reuse the code from Assignment 1.
- (b) (20 pts) Implement the MLP class.
  - (5 pts) Model implementation with numpy
  - (5 pts) AdaGrad implementation
  - (5 pts) Minibatch gradient GD for MLP using AdaGrad
  - (5 pts) Model implementation with Tensorflow
- (c) Run all the code to make sure your implementation works.
- (d) (5 pts) Conclusion.

### 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  $\mathbf{u}_w$  ( $w = 1, \dots, W$ ) are the “context” (output) word vectors for all words in the vocabulary. The cross entropy function is defined as:

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

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  $U = [u_1, u_2, \dots, u_W]$  is the matrix of all the context vectors. Assume cross entropy is applied to this prediction, derive the gradients with respect to  $\mathbf{v}_c$ .

- (c) (5 pts) Derive gradients for the “context” 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  $1, \dots, K$  respectively. For simplicity of notation, assume ( $o \notin \{1, \dots, K\}$ ). Again for a given word  $o$ , use  $\mathbf{u}_o$  to denote its context 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  $[\text{word}_{c-m}, \dots, \text{word}_c, \dots, \text{word}_{c+m}]$  where  $m$  is the context size. Denote the “center” and “context” 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  $J_{\text{CE}}(o, \mathbf{v}_c \dots)$  or  $J_{\text{neg-sample}}(o, \mathbf{v}_c \dots)$  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  $\frac{\partial F(o, \mathbf{v}_c)}{\partial \dots}$ . Recall that for skip-gram, the cost for a context centered around  $c$  is:

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

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

- (a) (5pts) Data processing including tokenization.
- (b) (10 pts) Training data generation.
- (5 pts) Positive samples
  - (5 pts) Negative samples
- (c) (20 pts) Skip-gram model implementation using Tensorflow.
- (10 pts) Loss function.
  - (10 pts) Model.
- (d) (5 pts) Implement the k-nearest neighbors algorithm, which will be used for visualization and 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](https://en.wikipedia.org/wiki/Cosine_similarity)).
- (e) Run the jupyter notebook code to make sure your implementation works
- (f) (5 pts) Conclusion.