Natural Language Processing **Tel Aviv University**

Assignment 1: Word Vectors

Due Date: *November 21 , 2020* Lecturer: Jonathan Berant

1 Preliminaries

Submission Instructions The enviroment setup and submission tutorial can be found in the following notebook: `https://colab.research.google.com/drive/1jx33OoQmrXK7t95Dxxpaj_OoBz5oKlWt?usp=sharing`
If you are comfortable with LaTeX, feel free to use the supplied `written_solution_template.tex` as a basis for you written solution.

Acknowledgements This assignment was adapted from Stanford's CS224n course. Their contributions are greatly appreciated.

2 Understanding word2vec

Let's have a quick refresher on the `word2vec` algorithm. The key insight behind `word2vec` is that '*a word is known by the company it keeps*'. Concretely, suppose we have a 'center' word c and a contextual window surrounding c. We shall refer to words that lie in this contextual window as 'outside words'. For example, in Figure 1 we see that the center word c is 'banking'. Since the context window size is 2, the outside words are 'turning', 'into', 'crises', and 'as'.

The goal of the skip-gram `word2vec` algorithm is to accurately learn the probability distribution $P(O \mid C)$. Given a specific word o and a specific word c, we want to calculate $P(O = o \mid C = c)$, which is the probability that word o is an 'outside' word for c, i.e., the probability that o falls within the contextual window of c.

In word2vec, the conditional probability distribution is given by taking vector dot-products and applying the softmax function:

$$P(O = o \mid C = c) = \frac{\exp(\boldsymbol{u}_o^\intercal \boldsymbol{v}_c)}{\sum_{w \in \mathrm{W}} \exp(\boldsymbol{u}_w^\intercal \boldsymbol{v}_c)} \tag{1}$$

Here, W is the vocabulary, $\boldsymbol{u}_o$ is the 'outside' vector representing outside word o, and $\boldsymbol{v}_c$ is the 'center' vector representing center word c. To contain these parameters, we have two matrices, $\boldsymbol{U}$ and $\boldsymbol{V}$. The columns of $\boldsymbol{U}$ are all the 'outside' vectors $\boldsymbol{u}_w$. The columns of $\boldsymbol{V}$ are all of the 'center' vectors $\boldsymbol{v}_w$. Both $\boldsymbol{U}$ and $\boldsymbol{V}$ contain a vector for every $w \in W$[1].

Recall from lectures that, for a single pair of words c and o, the loss is given by:

$$\boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U}) = -\log P(O = o \mid C = c) \tag{2}$$

[1]Assume that every word in our vocabulary is matched to an integer number k. $\boldsymbol{u}_k$ is both the k^{th} column of $\boldsymbol{U}$ and the 'outside' word vector for the word indexed by k. $\boldsymbol{v}_k$ is both the k^{th} column of $\boldsymbol{U}$ and the 'center' word vector for the word indexed by k. **In order to simplify notation we shall interchangeably use k to refer to the word and the index-of-the-word**.

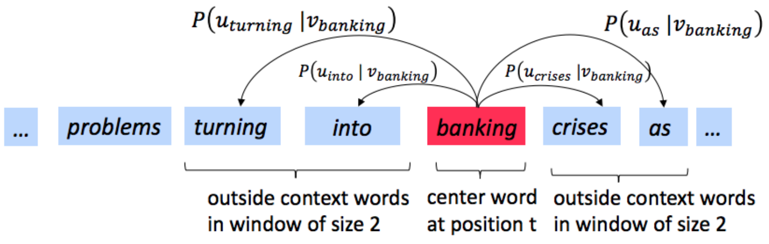
$P(u_{turning} \mid v_{banking})$
$P(u_{into} \mid v_{banking})$
$P(u_{crises} \mid v_{banking})$
$P(u_{as} \mid v_{banking})$
...
problems
turning
into
banking
crises
as
...
outside context words in window of size 2
center word at position t
outside context words in window of size 2

Figure 1: The word2vec skip-gram prediction model with window size 2

Another way to view this loss is as the cross-entropy[2] between the true distribution $\boldsymbol{y}$ and the predicted distribution $\hat{\boldsymbol{y}}$. Here, both $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ are vectors with length equal to the number of words in the vocabulary ($|W|$). Furthermore, the k^{th} entry in these vectors indicates the conditional probability of the k^{th} word being an 'outside word' for the given c. The true empirical distribution $\boldsymbol{y}$ is a one-hot vector with a 1 for the true outside word o, and 0 everywhere else. The predicted distribution $\hat{\boldsymbol{y}}$ is the probability distribution $P(O \mid C = c)$ given by our model in Equation (1).

(a) Equation (1) uses the softmax function. Prove that softmax is invariant to constant offset in the input, i.e prove that for any input vector $\boldsymbol{x}$ and any constant c,

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

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

$$\text{softmax}(\boldsymbol{x})_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

(b) Show that the naive softmax loss given in Equation (2) is the same as the cross-entropy loss between $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$, i.e., show that

$$-\sum_{w \in \mathrm{W}} y_w \log(\hat{y}_w) = -\log(\hat{y}_o) \tag{3}$$

Your answer should be one line.

(c) Compute the partial derivative of $\boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})$ with respect to $\boldsymbol{v}_c$. Please write you answer in terms of $\boldsymbol{y}$, $\hat{\boldsymbol{y}}$ and $\boldsymbol{U}$.

(d) Compute the partial derivative of $\boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})$ with respect to each of the 'outside' word vectors, $\boldsymbol{u}_w$'s. There will be two cases: when $w = o$, the true 'outside' word vector, and when $w \neq o$, for all other words. Please write you answer in terms of $\boldsymbol{y}$, $\hat{\boldsymbol{y}}$ and $\boldsymbol{v}_c$.

(e) The sigmoid function is given by

$$\sigma(\boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{x})} = \frac{\exp(\boldsymbol{x})}{\exp(\boldsymbol{x}) + 1}$$

Please compute the derivative of $\sigma(\boldsymbol{x})$ with respect to $\boldsymbol{x}$ where $\boldsymbol{x}$ is a vector. Try to express it in terms of $\sigma(\boldsymbol{x})$ and constants only.

(f) Now we shall consider the negative sampling loss, which is an alternative to the naive softmax loss. Assume that K negative samples (words) are drawn from the vocabulary W. For simplicity of notation we shall refer to them as $w_1, w_2, \ldots, w_K$ and to their corresponding outside vectors as

[2]The Cross Entropy Loss between the true (discrete) probability distribution p and another distribution q is $-\sum_i p_i \log(q_i)$.

$\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_K$. Note that $o \notin \{\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_K\}$. For a center word c and an outside word o, the negative sampling loss function is given by:

$$\boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U}) = -\log(\sigma(\boldsymbol{u}_o^\intercal \boldsymbol{v}_c)) - \sum_{k=1}^{K} \log(\sigma(-\boldsymbol{u}_k^\intercal \boldsymbol{v}_c))$$

for a sample $w_1, w_2, \ldots, w_K$, where $\sigma(\cdot)$ is the sigmoid function[3].

Please repeat parts (b) and (c), computing the partial derivatives of $\boldsymbol{J}_{\text{neg-sample}}$ with respect to $\boldsymbol{v}_c$, with respect to $\boldsymbol{u}_o$, and with respect to a negative sample $\boldsymbol{u}_k$. Please write your answers in terms of the vectors $\boldsymbol{u}_o$, $\boldsymbol{v}_c$ and $\boldsymbol{u}_k$, where $k \in [1, K]$. After you've done this, describe with one sentence why this loss function is much more efficient to compute than the naive softmax loss. Note, you should be able to use your solution of (e) to help compute the necessary gradients here.

(g) Suppose the center word is $c = w_t$ and the context window is $[w_{t-m}, \ldots, w_{t-1}, w_t, w_{t+1}, \ldots, w_{t+m}]$, where m is the context window size. Recall that for the skip-gram version of `word2vec`, the total loss for the context window is:

$$\boldsymbol{J}_{\text{skip-gram}}(\boldsymbol{v}_c, w_{t-m}, \ldots, w_{t+m}, \boldsymbol{U}) = \sum_{\substack{-m \le j \le m \\ j \ne 0}} \boldsymbol{J}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U})$$

Here, $\boldsymbol{J}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U})$ represents an arbitrary loss term for the center word $c = w_t$ and outside word w_{t+j}. $\boldsymbol{J}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U})$ could be $\boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U})$ or $\boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U})$, depending on your implementation.

Write down three partial derivatives:

(i) $\partial \boldsymbol{J}_{\text{skip-gram}}(\boldsymbol{v}_c, w_{t-m}, \ldots, w_{t+m}, \boldsymbol{U}) / \partial \boldsymbol{U}$

(ii) $\partial \boldsymbol{J}_{\text{skip-gram}}(\boldsymbol{v}_c, w_{t-m}, \ldots, w_{t+m}, \boldsymbol{U}) / \partial \boldsymbol{v}_c$

(iii) $\partial \boldsymbol{J}_{\text{skip-gram}}(\boldsymbol{v}_c, w_{t-m}, \ldots, w_{t+m}, \boldsymbol{U}) / \partial \boldsymbol{v}_w$ when $w \ne c$

Write your answers in terms of $\partial \boldsymbol{J}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U}) / \partial \boldsymbol{U}$ and $\partial \boldsymbol{J}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U}) / \partial \boldsymbol{v}_c$. This is very simple, each solution should be one line.

Once you're done: *Given that you computed the derivatives of $\boldsymbol{J}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U})$ with respect to all the model parameters $\boldsymbol{U}$ and $\boldsymbol{V}$ in parts (a) to (c), you have now computed the derivatives of the full loss function $\boldsymbol{J}_{\textit{skip-gram}}$ with respect to all parameters. You're ready to implement* `word2vec`*!*

3 Implementing word2vec

In this part you will implement the word2vec model and train your own word vectors with stochastic gradient descent (SGD).

(a) Implement the `softmax` function in the module `q3a_softmax.py`. Note that in practice, for numerical stability, we make use of the property we proved in question 2.a and choose $c = -\max_i x_i$ when computing softmax probabilities (i.e., subtracting its maximum element from all elements of x). You can test your implementation by running `python q3a_softmax.py`.

Note: The provided tests are not exhaustive. Later parts of the assignment will reference this code so it is important to have a correct implementation. Your implementation should also be efficient

[3]The loss function here is the negative of what Mikolov et al. had in their original paper, because we are minimizing rather than maximizing in our assignment code. Ultimately, this is the same objective function.

and vectorized whenever possible (i.e., use numpy matrix operations rather than `for` *loops). A non-vectorized implementation will not receive full credit!*

(b) To make debugging easier, we will now implement a gradient checker. Fill in the implementation for the `gradcheck_naive` function in the module `q3b_gradcheck`. You can test your implementation by running `python q3b_gradcheck.py`.

(c) Fill in the implementation for `naive_softmax_loss_and_gradient`, `neg_sampling_loss_and_gradient`, and `skipgram` in the module `q3c_word2vec.py`. You can test your implementation by running `python q3c_word2vec.py`. Verify that your results are approximately equal to the expected results.

(d) Complete the implementation for the SGD optimizer in the module `q3d_sgd.py`. You can test your implementation by running `python q3d_sgd.py`.

(e) 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 q3e_run.py`.

Note: The training process may take a long time depending on the efficiency of your implementation. Plan accordingly!

After 40,000 iterations, the script will finish and 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, inside the pdf (not a separate file)**. Briefly explain what you notice in the plot. Are there any reasonable clusters/trends? Are the word vectors as good as you expected? If not, what do you think could make them better?

4 Optimizing word2vec

We will now prove that in the skipgram algorithm, the maximum likelihood solution for word embedding probabilities is their empirical distribution, and show that there exists a scenario where reaching the optimum is impossible.

(a) For a corpus of length T, recall that the objective is:

$$\begin{aligned}\mathcal{L}(\theta) &= \prod_{t=1}^{T} \prod_{-m\leq j\leq m, j\neq 0} p_\theta\left(w_{t+j} \mid w_t\right)\\ J(\theta) = \log\mathcal{L}(\theta) &= \sum_{t=1}^{T} \sum_{-m\leq j\leq m} \log p_\theta\left(w_{t+j} \mid w_t\right)\end{aligned}$$

Prove that if $\theta^* = \arg\max_\theta \mathcal{L}(\theta)$ then $p_{\theta^*}(o \mid c) = \frac{\#(c,o)}{\sum_{o'} \#(c,o')}$.
Where $\#(c,o) =$ Number of co-occurrence of c and o in the corpus.

Hint: For a fixed c, o in the vocabulary, how many times does the term $p_\theta(o \mid c)$ appear in $\mathcal{L}(\theta)$?

(b) Let's assume each word is represented by a single scalar (real number). Prove that there is a corpus over a vocabulary of no more than 4 words, where reaching the optimum solution is impossible. You can assume that a corpus is a list of sentences, such as $\{$"*aa*", "*bb*", "*cc*", $\ldots, \}$.

5 Paraphrase Detection

Paraphrase detection is a binary classification task, where given two sentences, the model needs to determine if they are paraphrases of one another. For example the pair of sentences *"The school said that the buses accommodate 24 students"* and *"It was said by the school that the buses seat two dozen students"* are paraphrases. While the pair *"John ate bread"* and *"John drank juice"* are not.

Let's denote by x_1, x_2 the input pair of sentences and by $\mathbf{x}_1, \mathbf{x}_2$, a vector representation for the pair of sentences obtained using some neural network, where $\mathbf{x}_i \in \mathbb{R}^d$.

Consider the following model for paraphrase detection:

$$p(\text{the pair is a paraphrase} \mid x_1, x_2) \text{ [illegible] } \left(\text{relu}(\mathbf{x}_1)^\top \text{relu}(\mathbf{x}_2)\right),$$

where $\text{relu}(x) = \max(0, x)$.

(a) In this model, what is the maximal accuracy on a dataset where the ratio of positive to negative examples is 1:3?

(b) Suggest a simple fix for the problem.