Natural Language Processing **Tel Aviv University**

Assignment 2: Language Models

Due Date: *Saturday, January 4th, 2025* Lecturer: Maor Ivgi, TA: Noa Mark

1 Word-Level Neural Bi-gram Language Model

In this question, you will implement and train neural language model, and evaluate it on using perplexity.

(a) Derive the gradient with respect to the input of a softmax function when cross entropy loss is used for evaluation, i.e., find the gradients with respect to the softmax input vector $\boldsymbol{\theta}$, when the prediction is made by $\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{\theta})$. Cross entropy and softmax are defined as:

$$\text{CE}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_i y_i \cdot \log(\hat{y}_i)$$

$$\text{softmax}(\boldsymbol{\theta})_i = \frac{\exp(\theta_i)}{\sum_j \exp(\theta_j)}$$

The gold vector $\boldsymbol{y}$ is a one-hot vector, and the predicted vector $\hat{\boldsymbol{y}}$ is a probability distribution over the output space.

(b) Derive the gradients with respect to the input $\boldsymbol{x}$ in a one-hidden-layer neural network (i.e., find $\frac{\partial J}{\partial \boldsymbol{x}}$, where J is the cross entropy loss $\text{CE}(\boldsymbol{y}, \hat{\boldsymbol{y}})$). The neural network employs a sigmoid activation function for the hidden layer, and a softmax for the output layer. Assume a one-hot label vector $\boldsymbol{y}$ is used. The network is defined as:

$$\boldsymbol{h} = \sigma(\boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1),$$
$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{h}\boldsymbol{W}_2 + \boldsymbol{b}_2).$$

The dimensions of the vectors and matrices are $\boldsymbol{x} \in \mathbb{R}^{1\times D_x}, \boldsymbol{h} \in \mathbb{R}^{1\times D_h}, \hat{\boldsymbol{y}} \in \mathbb{R}^{1\times D_y}, \boldsymbol{y} \in \mathbb{R}^{1\times D_y}$. The dimensions of the parameters are $\boldsymbol{W}_1 \in \mathbb{R}^{D_x\times D_h}, \boldsymbol{W}_2 \in \mathbb{R}^{D_h\times D_y}, \boldsymbol{b}_1 \in \mathbb{R}^{1\times D_h}, \boldsymbol{b}_2 \in \mathbb{R}^{1\times D_y}$.

(c) Implement the forward and backward passes for a neural network with one sigmoid hidden layer. Fill in your implementation in `q1c_neural.py`. Sanity check your implementation with `python q1c_neural.py`.

(d) GloVe (Global Vectors) embeddings are a type of word embeddings that represent words as vectors in a high-dimensional space, based on the co-occurrence statistics of words in a corpus. They are related to the skip-gram embeddings you saw in class in that they both aim to capture the semantic and syntactic relationships between words, but GloVe embeddings incorporate global corpus-level information in addition to local context information. In this section you will be using GloVe embeddings to represent the vocabulary. Use the neural network to implement a bigram language model in `q1d_neural_lm.py`. Use GloVe embeddings to represent the vocabulary (`data/lm/vocab.embeddings.glove.txt`). Implement the `lm_wrapper` function, that is used by `sgd` to sample the gradient, and the `eval_neural_lm` function that is used for model evaluation. Report the dev perplexity in your written solution. Don't forget to include `saved_params_40000.npy` in your submission zip!

2 Generating Shakespeare Using a Character-level Language Model

In this section we will train a language model and use it to generate text.
Follow the instructions, complete the code, and answer the questions from this Google Colab notebook[1]:
https://colab.research.google.com/drive/1WIUACyCAgrPiuKzNBwXNCh0zWrecLnCF?usp=sharing

3 Perplexity

(a) Show that perplexity calculated using the natural logarithm $\ln(x)$ is equal to perplexity calculated using $\log_2(x)$. i.e:

$$2^{-\frac{1}{M}\sum_{i=1}^{M}\log_2 p(s_i|s_1,\ldots,s_{i-1})} = e^{-\frac{1}{M}\sum_{i=1}^{M}\ln p(s_i|s_1,\ldots,s_{i-1})}$$

(b) In this section you will be computing the perplexity of your previous trained models on two different passages. Please provide your results in the PDF file, as well as attach the code to your code files. The two different passages appear in the .zip file you've got. Their names are: `shakespeare_for_perplexity.txt` which contains a subset from the Shakespeare dataset, and `wikipedia_for_perplexity.txt` which contains a certain passage from Wikipedia. Please compute the perplexity of the bi-gram LM, and the model from section 3, on both these passages.

Use the models with their own tokenizer, without any additional prepossessing.

(c) Try to explain the results you've got. Particularly, why there might be large gaps in perplexity, while looking on different passages.

(d) Implement a prepossessing step to improve the results. Explain in the pdf your solution.

4 Deep Averaging Networks

In this question we will implement Deep Averaging Networks (DAN), and work on the IMDB dataset. The IMDB dataset consists of positive and negative reviews for movies.
This exercise will also introduce you to the popular `transformers` package from Hugging Face. It will also require reading some of the documentation of `pytorch`. The total number of lines of code you need to write for implementing the model itself is roughly 10 or less (i.e., not a lot). Complete the code, and answer the following questions: https://colab.research.google.com/drive/141W2qonpIYahv0wj-iJ-xqAhRFEa1muV?usp=sharing

Notes:

- *The model in the paper uses GloVe embeddings, in this exercise, you will implement this model using GloVe embedding trained on slightly less data, so you should expect different results then the ones shown in the paper.*
- *You may encounter some technical difficulties when accessing the data, as it can take time to load. Working on real-world problems requires handling such technical challenges, so plan your work accordingly to overcome them.*

[1] Feel free to comment inside the notebook.

One tip that might help: as you learned in HW0, you can download the data locally once and then upload it to your own drive. By mounting your drive, you can avoid loading issues and potentially reduce waiting time.

- *You may change the init argument for DAN class.*

(a) Implement the DAN model as described in section 3 in the paper. In a nutshell, the DAN model proposes to average the GloVe word embeddings to represent the sentence, and then pass this sentence representation through a multi-layer feed-forward network (or multi-layer perceptron). Your best model should get accuracy of at least 83.5% on the evaluation set (anything below will result in partial credit). Add feed-forward layers, and tune the learning rate and batch size as necessary. Include a plot of the evaluation accuracy as a function of the number of epochs.

(b) Word dropout is a popular method for regularizing the model and avoiding over-fitting. Read section 3.1 of the paper and add word dropout as described (see `pytorch` documentation), and include a plot of the accuracy of the model across different values of the dropout rate (See Figure 2 in the paper).

(c) Train 4 models with an increasing number of hidden layers (from 0 to 3 hidden layers), and compare the accuracy as the number of layers increases. Before you start training, think about when will we start seeing the effect of diminishing returns, are the results the way you expected them to be? Did the linear model outperform the model with 4 hidden layers? Include a plot of the accuracy as a function of the number of layers.

(d) Use *nn.Relu* and 2 other activation functions (of your choosing) from the torch documentation, include a plot of the accuracy across epochs. What have you learned from this experiment?

(e) For your best model, sample 5 examples from the evaluation set that the model classified incorrectly and for each example try to explain why the model classified it incorrectly.

5 Attention Exploration

Multi-head self-attention is the core modeling component of Transformers. In this question, we'll get some practice working with the self-attention equations, and motivate why multi-headed self-attention can be preferable to single-headed self-attention.

Recall that attention can be viewed as an operation on a *query* vector $q \in \mathbb{R}^d$, a set of *value* vectors $\{v_1, \dots, v_n\}, v_i \in \mathbb{R}^d$, and a set of *key* vectors $\{k_1, \dots, k_n\}, k_i \in \mathbb{R}^d$, specified as follows:

$$c = \sum_{i=1}^{n} v_i \alpha_i \tag{1}$$

$$\alpha_i = \frac{\exp(k_i^\top q)}{\sum_{j=1}^{n} \exp(k_j^\top q)} \tag{2}$$

with $alpha = \{\alpha_1, \dots, \alpha_n\}$ termed the "attention weights". Observe that the output $c \in \mathbb{R}^d$ is an average over the value vectors weighted with respect to α.

1. **Copying in attention.** One advantage of attention is that it's particularly easy to "copy" a value vector to the output c. In this problem, we'll motivate why this is the case.

(a) **Explain** why α can be interpreted as a categorical probability distribution.

(b) The distribution α is typically relatively "diffuse"; the probability mass is spread out between many different α_i. However, this is not always the case. **Describe** (in one sentence) under what conditions the categorical distribution α puts almost all of its weight on some α_j, where $j \in \{1, \dots, n\}$ (i.e. $\alpha_j \gg \sum_{i \neq j} \alpha_i$). What must be true about the query q and/or the keys $\{k_1, \dots, k_n\}$?

(c) Under the conditions you gave in (ii), **describe** the output c.

(d) **Explain** (in two sentences or fewer) what your answer to (ii) and (iii) means intuitively.

2. **An average of two.** Instead of focusing on just one vector v_j, a Transformer model might want to incorporate information from *multiple* source vectors. Consider the case where we instead want to incorporate information from **two** vectors v_a and v_b, with corresponding key vectors k_a and k_b.

(a) How should we combine two d-dimensional vectors v_a, v_b into one output vector c in a way that preserves information from both vectors? In machine learning, one common way to do so is to take the average: $c = \frac{1}{2}(v_a + v_b)$. It might seem hard to extract information about the original vectors v_a and v_b from the resulting c, but under certain conditions one can do so. In this problem, we'll see why this is the case.

Suppose that although we don't know v_a or v_b, we do know that v_a lies in a subspace A formed by the m basis vectors $\{a_1, a_2, \dots, a_m\}$, while v_b lies in a subspace B formed by the p basis vectors $\{b_1, b_2, \dots, b_p\}$. (This means that any v_a can be expressed as a linear combination of its basis vectors, as can v_b. All basis vectors have norm 1 and are orthogonal to each other.) Additionally, suppose that the two subspaces are orthogonal; i.e. $a_j^\top b_k = 0$ for all j, k.

Using the basis vectors $\{a_1, a_2, \dots, a_m\}$, construct a matrix M such that for arbitrary vectors $v_a \in A$ and $v_b \in B$, we can use M to extract v_a from the sum vector $s = v_a + v_b$. In other words, we want to construct M such that for any v_a, v_b, $Ms = v_a$. Show that $Ms = v_a$ holds for your M.

Hint: Given that the vectors $\{a_1, a_2, \dots, a_m\}$ are both *orthogonal* and *form a basis* for v_a, we know that there exist some $c_1, c_2, \dots, c_m$ such that $v_a = c_1 a_1 + c_2 a_2 + \cdots + c_m a_m$. Can you create a vector of these weights c?

(b) As before, let v_a and v_b be two value vectors corresponding to key vectors k_a and k_b, respectively. Assume that (1) all key vectors are orthogonal, so $k_i^\top k_j = 0$ for all $i \neq j$; and (2) all key vectors have norm 1.[2] **Find an expression** for a query vector q such that $c \approx \frac{1}{2}(v_a + v_b)$, and justify your answer. [3]

3. **Drawbacks of single-headed attention:** In the previous part, we saw how it was *possible* for a single-headed attention to focus equally on two values. The same concept could easily be extended to any subset of values. In this question we'll see why it's not a *practical* solution. Consider a set of key vectors $\{k_1, \dots, k_n\}$ that are now randomly sampled, $k_i \sim \mathcal{N}(\mu_i, \Sigma_i)$, where the means $\mu_i \in \mathbb{R}^d$ are known to you, but the covariances Σ_i are unknown. Further, assume that the means μ_i are all perpendicular; $\mu_i^\top \mu_j = 0$ if $i \neq j$, and unit norm, $\|\mu_i\| = 1$.

(a) Assume that the covariance matrices are $\Sigma_i = \alpha I, \forall i \in \{1, 2, \dots, n\}$, for vanishingly small α. Design a query q in terms of the μ_i such that as before, $c \approx \frac{1}{2}(v_a + v_b)$, and provide a brief argument as to why it works.

[2]Recall that a vector x has norm 1 iff $x^\top x = 1$.

[3]Hint: while the softmax function will never *exactly* average the two vectors, you can get close by using a large scalar multiple in the expression.

(b) Though single-headed attention is resistant to small perturbations in the keys, some types of larger perturbations may pose a bigger issue. Specifically, in some cases, one key vector k_a may be larger or smaller in norm than the others, while still pointing in the same direction as μ_a. As an example, let us consider a covariance for item a as $\Sigma_a = \alpha I + \frac{1}{2}(\mu_a \mu_a^\top)$ for vanishingly small α (as shown in figure 1). This causes k_a to point in roughly the same direction as μ_a, but with large variances in magnitude. Further, let $\Sigma_i = \alpha I$ for all $i \neq a$.

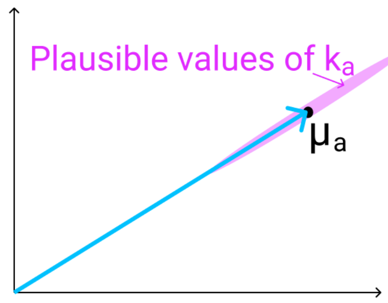

Figure 1: The vector μ_a (shown here in 2D as an example), with the range of possible values of k_a shown in red. As mentioned previously, k_a points in roughly the same direction as μ_a, but may have larger or smaller magnitude.

When you sample $\{k_1, \ldots, k_n\}$ multiple times, and use the q vector that you defined in part i., what do you expect the vector c will look like qualitatively for different samples? Think about how it differs from part (i) and how c's variance would be affected.

4. **Benefits of multi-headed attention:** Now we'll see some of the power of multi-headed attention. We'll consider a simple version of multi-headed attention which is identical to single-headed self-attention as we've presented it in this homework, except two query vectors (q_1 and q_2) are defined, which leads to a pair of vectors (c_1 and c_2), each the output of single-headed attention given its respective query vector. The final output of the multi-headed attention is their average, $\frac{1}{2}(c_1 + c_2)$. As in question 1(3), consider a set of key vectors $\{k_1, \ldots, k_n\}$ that are randomly sampled, $k_i \sim \mathcal{N}(\mu_i, \Sigma_i)$, where the means μ_i are known to you, but the covariances Σ_i are unknown. Also as before, assume that the means μ_i are mutually orthogonal; $\mu_i^\top \mu_j = 0$ if $i \neq j$, and unit norm, $\|\mu_i\| = 1$.

   (a) Assume that the covariance matrices are $\Sigma_i = \alpha I$, for vanishingly small α. Design q_1 and q_2 such that c is approximately equal to $\frac{1}{2}(v_a + v_b)$. Note that q_1 and q_2 should have different expressions.

   (b) ssume that the covariance matrices are $\Sigma_a = \alpha I + \frac{1}{2}(\mu_a \mu_a^\top)$ for vanishingly small α, and $\Sigma_i = \alpha I$ for all $i \neq a$. Take the query vectors q_1 and q_2 that you designed in part i. What, qualitatively, do you expect the output c to look like across different samples of the key vectors? Explain briefly in terms of variance in c_1 and $c2$. You can ignore cases in which $k_a^\top q_i < 0$.

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