Natural Language Processing **Tel Aviv University**

Assignment 2: Language Models

Due Date: *December 15, 2020* Lecturer: Jonathan Berant

0 Preliminaries

Submission Instructions The enviroment setup and submission tutorial can be found in the following notebook: `https://colab.research.google.com/drive/1zUd7oL7GSK3hcoRYbyoEOywNzDmL9uvD?usp=sharing`

Acknowledgements This assignment was adapted from Stanford's CS224d course and work done by Sean Robertson. Their contributions are greatly appreciated.

1 Word-Level Neural Bigram Language Model

(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) 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 Theoretical Inquiry of a Simple RNN Language Model

In this section we will perform a short theoretical analysis of a simple RNN language model, adapted from a paper by Tomas Mikolov, et al.[1]. Formally, for every timestep t, the model is defined as follows:

$$\begin{aligned} \boldsymbol{e}^{(t)} &= \boldsymbol{x}^{(t)}\boldsymbol{L} \\ \boldsymbol{h}^{(t)} &= \text{sigmoid}\left(\boldsymbol{h}^{(t-1)}\boldsymbol{H} + e^{(t)}\boldsymbol{I} + \boldsymbol{b}_1\right) \\ \hat{\boldsymbol{y}}^{(t)} &= \text{softmax}\left(\boldsymbol{h}^{(t)}\boldsymbol{U} + \boldsymbol{b}_2\right) \end{aligned} \tag{1}$$

where $\boldsymbol{h}^{(0)} \in \mathbb{R}^{D_h}$ is some initialization vector for the hidden layer and $\boldsymbol{x}^{(t)}\boldsymbol{L}$ is the product of $\boldsymbol{L}$ with the one-hot vector $\boldsymbol{x}^{(t)}$ representing index of the current word. The parameters are:

$$\boldsymbol{L} \in \mathbb{R}^{|V|\times d} \qquad \boldsymbol{H} \in \mathbb{R}^{D_h\times D_h} \qquad \boldsymbol{I} \in \mathbb{R}^{d\times D_h} \qquad \boldsymbol{b}_1 \in \mathbb{R}^{D_h} \qquad \boldsymbol{U} \in \mathbb{R}^{D_h\times |V|} \qquad \boldsymbol{b}_2 \in \mathbb{R}^{|V|} \tag{2}$$

where $\boldsymbol{L}$ is the embedding matrix, $\boldsymbol{I}$ is the input word weight matrix, $\boldsymbol{H}$ is the hidden state weight matrix, $\boldsymbol{U}$ is the output word transformation matrix, and $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are biases. As for the dimensions, $|V|$ is the vocabulary size, d is the embedding dimension, and D_h is the hidden state dimension.

The output vector $\hat{\boldsymbol{y}}^{(t)} \in \mathbb{R}^{|V|}$ is a probability distribution over the vocabulary, and we optimize the cross-entropy loss:

$$J^{(t)}(\theta) = \text{CE}(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) = -\sum_{i=1}^{|V|} {y_i}^{(t)} \log(\hat{y}_i^{(t)})$$

where $\boldsymbol{y}^{(t)}$ is the one-hot vector corresponding to the target word (which in our case is equal to $\boldsymbol{x}^{(t+1)}$). Note that $J^{(t)}(\theta)$ is a loss for a single timestep.

(a) Compute the gradients for all model parameters at a single point in time (timestep) t:

$$\frac{\partial J^{(t)}}{\partial \boldsymbol{U}} \qquad \frac{\partial J^{(t)}}{\partial \boldsymbol{b}_2} \qquad \frac{\partial J^{(t)}}{\partial \boldsymbol{L}_{\boldsymbol{x}^{(t)}}} \qquad \left.\frac{\partial J^{(t)}}{\partial \boldsymbol{I}}\right|_{(t)} \qquad \left.\frac{\partial J^{(t)}}{\partial \boldsymbol{H}}\right|_{(t)} \qquad \left.\frac{\partial J^{(t)}}{\partial \boldsymbol{b}_1}\right|_{(t)}$$

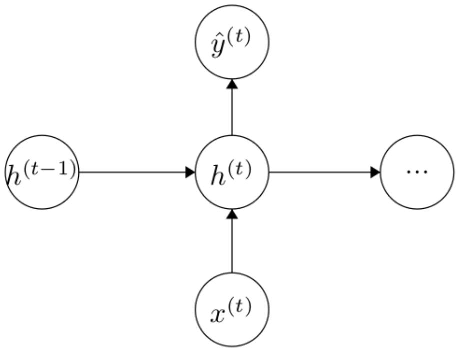
Where $\boldsymbol{L}_{\boldsymbol{x}^{(t)}}$ is the row of $\boldsymbol{L}$ corresponding to the current input word $\boldsymbol{x}^{(t)}$ and $\Big|_{(t)}$ denotes the gradient for the appearance of that parameter at time t. (Equivalently, $\boldsymbol{h}^{(t-1)}$ is taken to be fixed, and you don't need to backpropagate to earlier timesteps just yet - you'll do that in part (b)). Additionally, compute the derivative with respect to the *previous* hidden layer value:[2]

$$\frac{\partial J^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}$$

[1] `http://www.fit.vutbr.cz/research/groups/speech/publi/2010/mikolov_interspeech2010_IS100722.pdf`. You might Recognize Mikolov from `https://arxiv.org/abs/1301.3781`.

[2] For those of you who took Intro to ML, this derivative is also known as an "error term", $\boldsymbol{\delta}^{(t-1)}$.

(b) Below is a sketch of the network at a single timestep:



Draw the unrolled network for 3 timesteps and compute the "backpropagation-through-time" gradients:

$$\frac{\partial J^{(t)}}{\partial \boldsymbol{L}_{\boldsymbol{x}^{(t-1)}}} \qquad \left.\frac{\partial J^{(t)}}{\partial \boldsymbol{H}}\right|_{(t-1)} \qquad \left.\frac{\partial J^{(t)}}{\partial \boldsymbol{I}}\right|_{(t-1)} \qquad \left.\frac{\partial J^{(t)}}{\partial \boldsymbol{b}_1}\right|_{(t-1)}$$

where $\Big|_{(t-1)}$ denotes the gradient from the appearance of that parameter at time $(t-1)$. Because parameters are used multiple times in a forward computation, to implement an RNN we need to compute the gradient for each time they appear.

Use backpropagation rules and express your answer in the terms you computed in part (a). You can also use any other term mentioned in the introduction of Section 2. (This might prove easier than you expect, due to the elegance of backpropagation).

Note that the true gradient with respect to a training example requires us to run backpropagation all the way back to $t = 0$. In practice, however, we generally truncate this and only backpropagate for a fixed number of timesteps.

3 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[3]:
`https://colab.research.google.com/drive/1WIUACyCAgrPiuKzNBwXNChOzWrecLnCF?usp=sharing`

4 Perplexity

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

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

[3]Feel free to comment inside the notebook.