Language modeling is a central task in NLP, and language models can be found at the heart of speech recognition, machine translation, and many other systems. Given a sequence of words (represented as one-hot row vectors) $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$, a language model predicts the next word $\boldsymbol{x}^{(t+1)}$ by modeling:

$$P(\boldsymbol{x}^{(t+1)} = \boldsymbol{v}_j \mid \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

where $\boldsymbol{v}_j$ is a word in the vocabulary.

Your job is to compute the gradients of a recurrent neural network language model, which uses feedback information in the hidden layer to model the "history" $\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(1)}$. Formally, the model[6] is, for $t = 1, \ldots, n-1$:

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

where $\boldsymbol{h}^{(0)} = \boldsymbol{h}_0 \in \mathbb{R}^{D_h}$ is some initialization vector for the hidden layer and $\boldsymbol{x}^{(t)}\boldsymbol{L}$ is the product of $\boldsymbol{L}$ with the one-hot row vector $\boldsymbol{x}^{(t)}$ representing the current word. The parameters are:

$$\boldsymbol{L} \in \mathbb{R}^{|V| \times d} \quad \boldsymbol{H} \in \mathbb{R}^{D_h \times D_h} \quad \boldsymbol{I} \in \mathbb{R}^{d \times D_h} \quad \boldsymbol{b}_1 \in \mathbb{R}^{D_h} \quad \boldsymbol{U} \in \mathbb{R}^{D_h \times |V|} \quad \boldsymbol{b}_2 \in \mathbb{R}^{|V|} \tag{1}$$

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

The output vector $\hat{\boldsymbol{y}}^{(t)} \in \mathbb{R}^{|V|}$ is a probability distribution over the vocabulary. The model is trained by minimizing the (un-regularized) cross-entropy loss:

$$J^{(t)}(\theta) = CE(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) = -\sum_{j=1}^{|V|} y_j^{(t)} \log \hat{y}_j^{(t)}$$

where $\boldsymbol{y}^{(t)}$ is the one-hot vector corresponding to the target word (which here is equal to $\boldsymbol{x}^{(t+1)}$). We average the cross-entropy loss across all examples (i.e., words) in a sequence to get the loss for a single sequence.

(a) **(written)** Conventionally, when reporting performance of a language model, we evaluate on *perplexity*, which is defined as:

$$\text{PP}^{(t)}\left(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}\right) = \frac{1}{\bar{P}(\boldsymbol{x}_{\text{pred}}^{(t+1)} = \boldsymbol{x}^{(t+1)} \mid \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})} = \frac{1}{\sum_{j=1}^{|V|} y_j^{(t)} \cdot \hat{y}_j^{(t)}}$$

i.e. the inverse probability of the correct word, according to the model distribution $\bar{P}$. Show how you can derive perplexity from the cross-entropy loss (*Hint: remember that $\boldsymbol{y}^{(t)}$ is one-hot!*), and thus argue that minimizing the (arithmetic) mean cross-entropy loss will also minimize the (geometric) mean perplexity across the training set. ***This should be a very short problem - not too perplexing!***

For a vocabulary of $|V|$ words, what would you expect perplexity to be if your model predictions were completely random (chosen uniformly from the vocabulary)? Compute the corresponding cross-entropy loss for $|V| = 10000$.

(b) **(written)** Compute the gradients of the loss $J$ with respect to the following model parameters at a single point in time $t$ (to save a bit of time, you don't have to compute the gradients with the respect to $\boldsymbol{U}$ and $\boldsymbol{b}_1$):

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

where $\boldsymbol{L}_{x^{(t)}}$ is the row of $\boldsymbol{L}$ corresponding to the current 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 need not backpropagate to earlier timesteps just yet - you'll do that in part (c)).

Additionally, compute the derivative with respect to the *previous* hidden layer value:

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

(c) **(coding)** Use the corpus provided in the file to train a language model, using a sequence of words to predict the following word. The basic structure is already finished, complete the model by filling the code in ***model.py*** and ***train&test.py.***