15-780 HW3

Implementing Autoregressive Language Models

In this homework, you'll implement a simple autoregressive linear language model, to model text from the complete works of Shakespeare.

Included with this PDF is a set of three files you'll use to complete the assignment.

- `pg100.txt` - Text file of the complete works of Shakespeare.
- `code_15780.py` - utility functions, which we mostly covered in class, that you can use to complete the assignment. Here we also include a simple tokenizer class to process the text.
- `hw3.ipynb` - Jupyter notebook to implement your functions.

Note that in order to complete this assignment, you will need to set up a Jupyter notebook environment somehow. You will be able to do all of this on your own machine; a GPU system will be nice to have for future assignments, though not *strictly* needed (you can also, e.g., use Google Colab, we can't provide these for the class). We're not going to cover how to do this in this assignment, it'll be assumed you can already set this up.

Part 1: Training a Linear Language Model

For this portion, you'll implement the function:

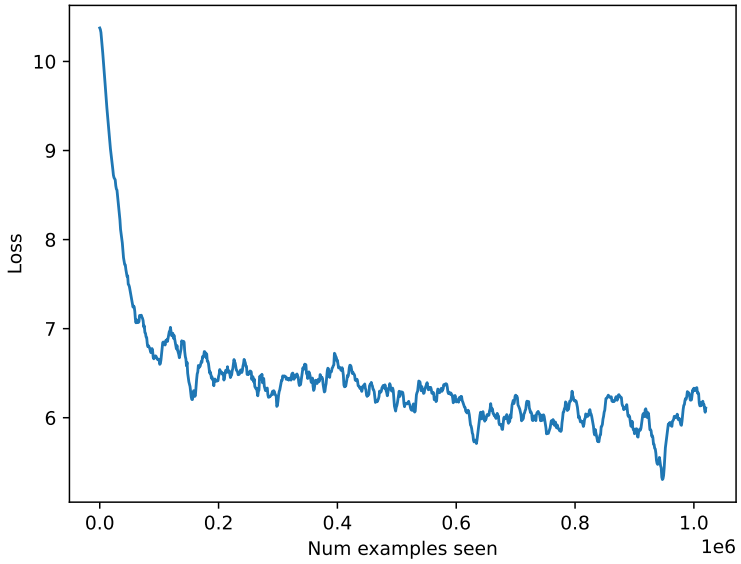
```
epoch_linear_language_model(tokens, Phi, theta, batch_size=1000, alpha=0.0, verbose=True)
```

To recall what we discussed in class (but to make it consistent with the notation used in this example), the inputs for this model will consist of a concatenation of embeddings of the the past H (referred to as `history` in the Python file) words of a document, and the target to predict, and the output y consists of the next word.

$$x = \begin{bmatrix} \Phi^T e_{\text{word}_{t-1}} \\ \Phi^T e_{\text{word}_{t-2}} \\ \vdots \\ \Phi^T e_{\text{word}_{t-H}} \end{bmatrix}, \quad y = \text{word}_t \tag{1}$$

where $\Phi \in \mathbb{R}^{V \times d}$ is an embedding matrix (with V the vocabulary size and d the embedding dimension). The corresponding linear parameter matrix with have size $\theta \in \mathbb{R}^{dH \times V}$. Note that you'll want to think about how to efficiently create a tensor X of all such examples in a batch, in a sliding window fashion, i.e., with each word in a batch of tokens (after the first H words) being a target.

Implement the function to train θ using stochastic gradient descent for one epoch (for multiple epochs, you can call the function multiple times). See the docstring for details of how the implementation should work, but the details are largely up to you. The function is a bit involved, but it's possible to be very succinct (for reference, our implementation is about 15 lines of code long, and run one epoch using the parameters provided in 2 minutes on an M1 mac on CPU, and there is certainly room to optimize). Your function should return a tensor of losses on each mini-batch; when we run with the parameters given the notebook we get this result:



Our implementation also gets an average loss of about 6.4 on the validation set.

Part 2: Sampling

Now implement the function

`sample_linear_language_model(init_tokens, Phi, theta, num_sample_tokens, temperature)`

to generate samples from the learned language model. At each step, you'll want to construct a vector x representing the history of the past H tokens (staring with the `init_tokens` input). Then sample the next token according to the multinomial probabilities given by

$$\text{softmax}(\theta^{\mathrm{T}}\mathrm{x}/\text{temperature}) \tag{2}$$

(don't worry about implementing the zero temperature case for now). Again, look at the docstring to see how this function should work.

If you code runs, you should be able to generate some example text from your language model. Don't expect too much, after training for an epoch with the parameters provided we get texts like this:

```
know thee,
That, and not so;
For that is not so camlet?
To not a fortune.

ANNE.
He hath.

RICHARD.
What, as, and her,
And, my that!

it, and I will not.

 [_Exeunt._]
```