Homework 3 - Language Modeling on Penn Treebank

Introduction

For this homework, you will be designing and training a language model on the Penn Treebank (text-only). This is your first adventure in unsupervised learning!

As you know, (autoregressive) language modeling is the problem of predicting the next token in a sequence given the previous N tokens. For example,

```
<start> nlp 243 is the best <t>
0       1   2   3  4   5    6
```

your model should produce $P($`<t>`$|$nlp, 243, is, the best$)$ for each `<t>` in your vocabulary (that is, you have `vocab_size` output classes).

The Penn Treebank is a corpus of $\sim$ 1 million tokens, which may sound like a lot but is rather small for language models, some of which are trained on gigabytes of data (BERT, GPT-3, etc.).

Here are some things you may try:

- RNN, LSTM, GRU, etc.
- Attention (covering next week)
- Different tokenizers
- Pretrained embeddings (word2vec, GloVe; **note: you may NOT use any pretrained models such as those in `transformers`**)
- Hyperparameter tuning
- Regularization, dropout, etc.

As before, you may not use any external libraries to write your model and training/eval loops, such as those in `huggingface` or `keras`.

Language modeling & unsupervised learning

While you don't have hand-written labels, you still need to have target values so your model can learn calculate loss and learn. The difference is that the target values are simply the next tokens in your input:

```
input: <start>
target: nlp

input: <start> nlp
target: 243

input: <start> nlp 243
target: is

...
```

In practice, we slightly alter this to make it more computationally tractable.

Pretend your whole dataset is just one massive 1D tensor of tokens IDs (with each sentence separated with `<start>`), and then bucket the tensor into `batch_size` columns. For example, suppose our dataset was the alphabet and our batch size was 4. Then we'd have 4 columns:

$$\begin{bmatrix} A & B & C & \ldots & X & Y & Z \end{bmatrix} \Rightarrow \left[\begin{array}{c|c|c|c} A & G & M & S \\ B & H & N & T \\ C & I & O & U \\ D & J & P & V \\ E & K & Q & W \\ F & L & R & X \end{array}\right]$$

Note that we ended up throwing away the last couple letters since they didn't fit in our batches—a necessary sacrifice. Also, keep in mind that when you do this, you'll be using token IDs.

Now, fix some sequence length—let's do 3. Then your first batch's inputs would be the first 3 rows (rows 0-2), and their targets would be the inputs shifted by one: rows 1-3.

$$\texttt{input:} \quad \left[\begin{array}{c|c|c|c} A & G & M & S \\ B & H & N & T \\ C & I & O & U \end{array}\right]$$
$$\texttt{target:} \quad \left[\begin{array}{c|c|c|c} B & H & N & T \\ C & I & O & U \\ D & J & P & V \end{array}\right]$$

Notice how we're still doing the same task—each sequence (column) is being trained to predict the next token in line given (at most) 3 tokens as context. (Not always 3 though—when we're predicting `B` for example, we only have one token—`A`—as context.) This lets us process `batch_size` sequences at once, significantly speeding up training.

Dataset

We're going to do things slightly differently in this homework, in order to give you some familiarity with the datasets library from `huggingface`.

You'll load the Penn Treebank using the `load_dataset` function as follows

```
from datasets import load_dataset
ptb = load_dataset('ptb_text_only')  # download the dataset
```

You'll note that the `ptb` object is a dictionary with three splits—train, validation, and test, and each example in each split is a dictionary with one key: `sentence`.

```
ptb['train'][2]['sentence'] # mr. <unk> is chairman of <unk> n.v. the dutch publishing group
```

From here, you can do any processing you need, including making a vocabulary, formatting the data as in the above example. `datasets` has a number of features like caching and batched preprocessing which speeds things up, if you care to investigate further—it may help with your final projects.

Competition, or lack thereof

A competition doesn't make sense for this task, since you will already have access to your own test set. This means you will simply submit your final generated test values and perplexity in your report, and we will do the leaderboard manually at the end.

Please note: do not focus on the leaderboard points as your primary goal. Your foremost goal is to create a working, bug-free, well-documented language model, and to conduct experiments to see what improves and what detracts from its performance. If you do that much and have time to spare, then you can focus on eeking out every last drop in perplexity.

To encourage you to work on your code and documentation over the leaderboard, your leaderboards will only be worth 3 points instead of 5, and there will be an extra 2 points available for those who submit beautiful code. That means

- **good** and **relevant** comments (not just "# does training" next to a function called `train`).
- properly organized code: your data loading should be clearly disentangled from your model and training code, etc.
- no unnecessary code or miscellaneous print statements
- no unnecessary imports
- a progress bar for training/eval using `tqdm`; not necessary, but it helps cleans up the unnecessary print statements

And whatever else you think improves the readability.

Submission

We expect the following things in your submitted zip file:

- A cleaned and organized final copy of your code (either `.py` files with instructions, or a `.ipynb` file).
- Your report

Report

The homework report must be a detailed summary of the approach you followed to complete your work. We highly recommend that you use a LaTeX template. for your report since for your proposal and final project, you will need to prepare those using the ACL Proceedings format. You will be required to provide the following high level sections as part of your report with additional subsections as described.

Introduction

Provide a formal statement of the problem you are trying to solve— whether it is a supervised or unsupervised problem, what specific task it is. Describe the dataset that was provided to you — background information, descriptive statistics of the dataset, what the input and output of the dataset are. Provide examples from the dataset inline or in tables.

Models

Give a description of what embedding methods have used for the models that you are training and why you pick them. Include a subsection for each model that you are training. Give a brief summary of how the models are implemented, trained, and the tuneable hyperparameters of the model. Additionally, provide citations to the original work that implemented these methods.

Experiments

Provide a description of the data-set split, the method for selecting hyper- parameters of your final models, any approaches you used for handling data sparsity/imbalance. Include a subsection for each model to describe the different values for the hyper-parameters tested, any special configurations for your model such as solvers or algorithms used. Describe the methods you used to evaluate how good your models were and what criterion you used to select the models for generating your test set submissions.

Results

Describe how well each model performed on your train, validation and test sets. Describe how the performance varied with different choice of hyperparameter values. Include the requisite tables, plots and figures to illustrate your point. Identify the best performing approach(es) and validate why they performed well. Try to bolster your conclusions by finding and citing work which arrive at similar results.

References

Provide a bibliography for the literature that you cited. You can make use of bibtex or natbib packages to automatically generate the bibliography section.

Appendix

Include an appendix for more detailed table, plots and figures, if needed.