Ling 282/482 hw3

Due 11PM on October 29, 2025

In this assignment, you will work with recurrent neural networks (RNNs) for language modeling. You will answer written questions about RNN architectures (including LSTMs), padding and masking for variable-length sequences, and perplexity-based evaluation. You will also explore a PyTorch implementation of character-level RNN language models, and train models with different hyperparameters to understand their effects on generated text.
More specifically, you will:

- Review details of RNN and LSTM internals
- Analyze masking and padding for variable-length sequences
- Compute perplexity for language model evaluation
- Understand the PyTorch implementation of RNN language models
- Train character-level language models and experiment with hyperparameters
- Analyze generated text and model behavior

Submission Instructions

Please answer the following questions and submit your answers **as a PDF on Blackboard**. Your final submission will be the one that is graded unless you specify otherwise.

1 RNN Language Models [31 pts]

Q1: Understanding RNNs [4 pts]

a. What is the main limitation of feed-forward neural networks that is overcome by recurrent networks, and how do recurrent networks achieve this? [2 pts]

b. The Vanilla RNN equation has the form $h_t = f(h_{t-1}, x_t)$. What extra 'ingredient' does the LSTM add to this general form? What problem is the LSTM designed to solve? [2 pts]

Q2: LSTM Update [11 pts] One of the "central" equations in the LSTM computation is the following:

$$c_t = f_t \odot c_{t-1} + i_t \odot \hat{c}_t$$

This equation performs an essential update of one part of the LSTM. Please answer:

a. What is c_t? [1 pt]

b. What is the range of f_t and what is its purpose? [2 pts]

c. What is the range of i_t and what is its purpose? [2 pts]

d. What is $\hat{c}_t$? [1 pt]

e. In your own words, describe how this equation implements the central "update" inside of an LSTM (2-3 sentences). [5 pts]

Q3: Counting parameters [6 pts] Let d_e be the dimension of token embeddings and d_h the hidden state size. Focusing on just the recurrent cell (and so ignoring the embedding and output layers):

a. How many parameters are there in a Vanilla RNN cell? [2 pts]

b. How many parameters are there in an LSTM cell? [4 pts]

Note: for this problem, you can assume that the RNN cell is at the 'bottom' of a possibly-deep RNN, so the inputs to the cell are token embeddings, not earlier layers' hidden states.

Q4: Understanding Masking [10 pts] Suppose that we want to train a (word-level) language model on the following two sentences:

<s> the cat sits </s>
<s> the model reads the sentence </s>

As we saw in the slides, padding is necessary to make these sentences have the same length so that they can be batched together, as:

<s> the cat sits </s> PAD PAD
<s> the model reads the sentence </s>

Please answer the following questions about these sequences:

a. In a recurrent language model, what would the input batch be? What would the target labels be? Hint: think about what the *input* and *output* tokens are at each time-step. [2 pts]

b. Recurrent language models use a *mask* of ones and zeros to 'eliminate' the loss for PAD tokens. What would the mask be for this batch? Your answer should be a matrix. [2 pts]

c. Suppose that we have the following per-token losses:

$$\begin{bmatrix} 0.1 & 0.3 & 0.2 & 0.4 & 0.7 & 0.5 \\ 0.2 & 0.6 & 0.1 & 0.8 & 0.9 & 0.4 \end{bmatrix}$$

What is the corresponding *masked* loss matrix? [2 pts]

d. Why is it important to mask losses in this way? What might a model learn to do if the loss is not masked? Answer in a few sentences. [4 pts]

2 Language Model Evaluation [16 pts]

Q1: Evaluating Language Models [16 pts] Given a corpus $W = w_1 w_2 \dots w_N$ (with N as the number of tokens in the corpus), a common (intrinsic) evaluation metric for language models is *perplexity*, defined as

$$PP(W) = P(w_1 \dots w_N)^{-\frac{1}{N}}$$

This can be thought of as the inverse probability that the model assigns to the corpus, normalized by the size of the corpus.

a. Is a lower or higher perplexity better? Why? [2 pts]

b. For a recurrent language model, write an expression for $P(w_1 \dots w_N)$ using the chain rule of probability. How is this different from the expression for a feed-forward language model? [4 pts]

c. Show that

$$PP(W) = e^{-\frac{1}{N}\sum_{i=1}^{N} \log P(w_i|w_{<i})}$$

where $w_{<i} = w_1 w_2 \dots w_{i-1}$ and log is the natural (base e) logarithm. [4 pts]

d. What is another name for the exponent $-\frac{1}{N}\sum_{i=1}^{N} \log P(w_i \mid w_{<i})$ in the above expression? Hint: it appears in training as well. [2 pts]

e. Suppose that the same text corpus were tokenized with two different vocabularies of different sizes (perhaps, e.g., one replaces infrequent tokens with an UNK token) and two language models were trained on the resulting tokenized text. All else being equal, would you expect perplexity to be lower or higher for the model with a smaller vocabulary? What consequences does this have for comparing different language models? [4 pts]

3 Understanding the Implementation [26 pts]

Accessing the code: You will receive an invitation to the Github repository for this assignment. As with hw2, you should fork the repository, clone your fork to BlueHive (or your local machine), and work from there. The code for this assignment can be found in the `decoding/` directory, which implements character-level language models. The main training script is `run.py`, which uses functions from `data.py` and model classes from `models.py`. Please answer the following questions about the implementation:

Q1: Understanding LM Forward Passes [8 pts]

a. In `models.py`, `FeedforwardLanguageModel.forward`: in your own words, what are lines 62-68 doing? What would go wrong if we didn't include this block? Go beyond the information included in the comments. [3 pts]

b. In the same function, line 77 is described as a "flattening" operation, implemented with PyTorch's `reshape` method. What is this "flattening" operation in the terms of the FFLM described in the slides and the textbook? What is its purpose? (Looking up the `reshape` method may help). [2 pts]

c. In `RNNLanguageModel.forward`, lines 150-160, we use the functions `pack_padded_sequence` and `pad_packed_sequence`. Look up the PyTorch documentation for these functions. In general terms, what is their purpose? Again, go beyond the information provided in the comments. [3 pts]

Q2: Understanding masking [4 pts] Look at the `mask_loss` function in `run.py` (lines 33-61). Explain in 2-3 sentences why we divide by `mask.sum()` rather than by the total number of elements in the loss tensor. What would happen to the gradient magnitudes if we used the wrong denominator?

Q3: Understanding generation [8 pts]

a. In 3-4 sentences of plain English, describe how the loop in the `generate` function in `run.py` works (lines 94-113). [4 pts]

b. In line 92, we use the `torch.full` method. Look up the documentation for this, and briefly explain how it works. We used it for something similar in lines 64-66 of `models.py`. Briefly summarize the connection between the two calls to this same function. [2 pts]

c. Why do we use `torch.no_grad()` during generation (`run.py`, line 94)? What advantage does it provide? [2 pts]

Q4: GPU Training [6 pts]

a. The code uses `torch.device` and the `.to(device)` pattern throughout. Find where the device is determined and where it's used. Why is it important that the model, input data, and target labels all be on the same device? Why does the code determine the device flexibly at runtime rather than hardcoding whether to use GPU or CPU? [3 pts]

b. Research and briefly explain (3-4 sentences): What makes GPUs faster than CPUs for neural network training? What kinds of operations benefit most from GPU acceleration? What kind of operations would not especially benefit? [3 pts]

4 Running the Language Model [27 pts]

NOTE: This assignment should be run on BlueHive using GPU resources that have been reserved for our class, following the instructions below.

Running on BlueHive: The repository includes a `slurm_run.sh` script for submitting batch (non-interactive) jobs to BlueHive's SLURM scheduler. To run your training job, use:

```
sbatch slurm_run.sh
```

This script includes several important `#SBATCH` directives at the top that configure how your job runs:

- `#SBATCH --reservation=ling282_2025`: This line sends your job to the GPU nodes specifically reserved for our class, ensuring you have priority access.
- `#SBATCH -o gen.out` and `#SBATCH -e gen.err`: These lines redirect the script's standard output (including your training progress and generated text) to `gen.out` and any error messages to `gen.err`. Check these files to see your results. **Hint:** You can use `tail -f gen.out` to watch your results in real time.
- Other directives specify resource requirements like the number of CPUs (`-c 9`), memory (`--mem=44g`), GPU allocation (`--gres=gpu:1`), and maximum runtime (`-t 1:00:00`).

You can modify the Python command at the bottom of `slurm_run.sh` to change hyperparameters (e.g., `python -u run.py --model_type lstm --hidden_size 256`). The code is also runnable on laptops without GPU, but training will be significantly slower and may cause your processor to heat up.

`run.py` contains a basic training loop for character-level language modeling. It will record the training and dev loss (and perplexity) at each epoch, and save the best model according to dev loss. Periodically (as specified by a command-line flag), it also outputs generated text from the best model.

Note on run time: the training script for an RNN completes about 8 epochs in 30 seconds (on our dedicated GPU node). A full training run of 64 epochs will thus take about 4 minutes. The Feedforward Language Model takes **much longer**, so I recommend decreasing the number of epochs if you experiment with this architecture.

Q1: Default parameters [7 pts] Execute `run.py` with its default arguments. Paste below the texts that are generated every 4 epochs, as well as the epoch with the best dev loss and the dev perplexity from that epoch. In 3-4 sentences, describe any trends that you see in the generated text as training progresses. (Note that generated text will not necessarily be completely coherent: recall that this is a *character-level* language model).

Q2: Modify hyper-parameters [10 pts] Re-run the training loop, modifying some combination of the following hyper-parameters, which are specified by command-line flags (see `run.py`):

- Model type (default: RNN. Options include LSTM and feedforward models)
- Hidden layer size
- Embedding size
- Batch size
- Learning rate
- Number of epochs (in particular: making it larger)

- Softmax temperature
- L_2 regularization coefficient
- Dropout (probability with which neurons are dropped during training)

Include your model's generated texts here. In 3-5 sentences, state exactly what hyper-parameter change(s) you made, and what effects (if any) you see in terms of the dev set perplexity and text that the model generated.

Q3: Modify hyper-parameters again [10 pts] Based on your results from Q1-Q2, pick another *different* set of hyper-parameters to try. Before you re-run training, write down your prediction/hypothesis for what will happen. Re-run the training loop one more time, with the new hyper-parameters. In 3-4 sentences, state what hyper-parameter change(s) you made. Was the prediction you made supported? Report any trends and why you think your prediction was or was not born out.