ITCS 6101/8101 Homework 3: Implementation (120 points)

February 19, 2026

In this assignment, you are given a code base in Python that implements a Feed-Forward Neural Network Language Model (FFLM) using pre-trained GloVe embeddings. The model takes K consecutive word embeddings as input and computes the next word distribution. The skeleton code and data are linked from the course webpage. Organize your code in folders as shown in Table 1 below.

```
hw03/
   report.pdf
   code/
     fflm.py
     lm_model.pt
     traces.txt
   data/
     glove.6B/
         glove.6B.50d.txt + glove.6B.100d.txt
         glove.6B.200d.txt + glove.6B.300d.txt
     tiny/
         tiny.train.tokens.4M + tiny.valid.tokens.200K
         tiny.test.tokens.4M
     wiki/
         wiki.train.tokens + wiki.valid.tokens
         wiki.test.tokens
```

Table 1: Folder structure.

The FFLM architectures contains H of hidden layers, each with the same number N of ReLU neurons, and one softmax output layer. The NN model takes as input the concatenated embeddings of K consecutive words and produces as output a probability distribution over all the tokens in the vocabulary. Pre-trained GloVe embeddings of various dimensions for a pre-defined set of 400K tokens are provided in the text files from the `data/glove.6B/` folder.

The `main()` function is the main entry point in the program, and accepts command line parameters for 4 modes of use:

1. `-train`: Train the model on a tokenized training corpus, by calling the `train()` function.

2. `-use`: Use the trained model in interactive mode to predict the next token for a sequence of tokens from the user, by calling the `use()` function.

3. `-prompt`: Use the trained model to generate a complete sequence of tokens conditioned on a prompt from the user, by calling the `prompt()` function.

4. `-evaluate`: Evaluate the trained model on a tokenized test corpus, by calling the `evaluate()` function.

Furthermore, the `main()` function processes the command line to read a number of program parameters, including `-H` (the number of hidden layers), `-N` (the number of neurons per hidden layer), `-K` (the number of input context words), `-E` (the number of training epochs), `-B` (the minibatch size), `-VSize` (the vocabulary size), `-greedy` (whether to use greedy vs. multinomial sampling), as well as paths for various types of input files

Complete the implementation by writing code in the sections marked by the comment `YOUR CODE HERE`:

1. **Build the neural network architecture** in the constructor of the `FFLM` class, by appending the necessary computational elements to the list `layers`. For this, you may want to use classes implemented in the `torch.nn` package.

2. **Create the set of training examples** in the function `create_examples()`.

3. Given a model and a set of examples, **compute the average cross-entropy** loss in the function `compute_loss`. You may want to compute the loss over one batch of a time, to leverage parallelism. Keep in mind that `torch.nn.CrossEntropyLoss()` itself already averages the loss.

4. **Implement the gradient update step** in the function `train()`, by using the `optimizer`, the cross-entropy `criterion`, and the `backward()` function on the loss.

5. Use the trained model to **compute the softmax probabilities** for predicting the next token, in the function `use()`.

6. **Generate the next token** using either greedy or multinomial sampling, in the function `generate_next_token()`.

7. **Compute perplexity on a test corpus**, in the function `evaluate()`.

To train, tune, and evaluate the FFLM model, you will use a portion of the TinyStories dataset, which contains short stories that only contain words that a typical 3 to 4-year-olds usually understand, generated by GPT-3.5 and GPT-4. The dataset is stored in `data/tiny/` and is partitioned into training, validation, and test sets.

Your submission should contain the trained model saved in `code/lm_model.pt`, as well as a complete set of outputs from your model as printed during training and evaluation, saved into `code/traces.txt`. Your outputs should reflect the 4 main modes of use of the program:

8. `-train`: Train the model on `tiny/tiny.train.tokens.4M`.

9. `-use`: Use the trained model in interactive mode to predict the next token for a sequence of tokens from the user. At a minimum, show the predicted token for the following sequences: *'the girl told the'*, *'the bird then flew'*, *'the boys and the'*, *'one day , a'*, *'the elf told her'*, *'the witch was very'*, *'once upon a time'*.

10. `-prompt`: Use the trained model to generate a complete sequence of tokens conditioned on a prompt from the user. At a minimum, you should show the greedy completion of the prompt *once upon a time*, as well as 5 different sample continuations obtained with multinomial sampling.

11. `-evaluate`: Evaluate the trained model on both `data/tiny/tiny.valid.tokens.200K` and `data/tiny/tiny.train.tokens.4M` and report the two perplexity numbers.

12. Write a **report** summarizing experimental results and insights.

When you evaluate the top outputs, try to explain any unexpected results or mistakes, by referring to properties of the algorithm or the data. Whenever you formulate a hypothesis, it is important that it is supported empirically. As such, feel free to train and evaluate models that go beyond the setting described above. All the required results and analyses should be included in an appropriately edited homework report, using proper indentation, section titles, and formatting. If you include formulas, make sure that you use appropriate formatting of equations. The PDF of the report `report.pdf` should be submitted on Canvas, together with the code in the folder above. The assignment will be graded based on **both** the code and the quality of the report.

1 Bonus exercises

Any non-trivial and insightful extra work can be worth bonus points. For example:

1. Tune the various hyper-parameters in order to optimize loss or perplexity on the validation data, e.g., the learning rate, the number of hidden layers, the number of neurons, the optimal embedding size, the number of K words to use in the context, the number of epochs. You may also want to try early stopping with inertia.

2. Currently, the embeddings are frozen. Try fine-tuning the embeddings while training the FFLM parameters, perhaps using a separate lower learning rate.

3. Explore various regularization techniques, e.g., Dropout or L2 parameter norm penalty.

4. Run BPE on the training dataset and learn a vocabulary of up to 10,000 tokens. Randomly initialize their embeddings, and then train the FFLM jointly with the token embeddings, using a larger version of the TinyStories dataset. Contact me if you need a tokenized version, otherwise you can get the raw version from from HuggingFace.

5. Implement an `n-gram` model, perhaps using Laplace *smoothing* and *backoff* to address sparsity, and compare its perplexity on the test data with that obtained by the FFLM.

6. The model size is dominated by the softmax parameters. A widely used technique to reduce the number of parameters is to use the embedding matrix as the softmax parameters. This requires setting the last hidden layer to have the same size as the embeddings, and removing bias parameters. Implement and evaluate this approach – you would need to tie the softmax parameters and the embeddings and allow the pre-trained embeddings to be fine-tuned.

2 Submission

Electronically submit on Canvas a hw03.zip file that contains the hw03 folder in which your code is in the required files, as well as the `report.pdf` and `traces.txt` files.

On a Linux system, creating the archive can be done using the command:
`> zip -r hw03.zip hw03`.

Please observe the following when handing in homework:

1. Structure, indent, and format your code well.
2. Use adequate comments, both block and in-line to document your code.
3. Verify your code runs correctly when used in the directory structure shown above. We will not debug your code.
4. For your report, it is recommended to use an editor such as Overleaf for Latex, Word, or Jupyter-Notebook that allows editing and proper formatting of equations, plots, and tables with results.
5. Verify that your Canvas submission contains the correct files by downloading and unzipping it after posting on Canvas.