Homework #4: RNN Language Model

Dong Shu
Chen Si
Yuchen Wang
Ding Zhang

1. (7.0 points) Implement and train your RNN language model on Wikitext-2 corpus using the guidelines specified above.

a. provide a description (or illustration) of you architecture and discuss design choices, Model Architecture:

- Preprocessing: given the three txt files (wiki2.train, wiki2.valid, wiki2.test), we design the following three functions:
  - read_corpus(files): this function takes in a list “files” that contains strings for filenames, and reads the designated files and merges them into one corpus.
  - create_vocabulary(corpus, vocab_size): this function creates a vocabulary mapping based on given corpus’ word frequencies, sorted from most to least. We can also specify the number of words taken from this frequency list, specified by vocab_size (in this case equals to 10000)
  - corpus_to_int(corpus, vocab): this function converts the given corpus to its integer representation using the vocabulary mapping ‘vocab’.
  - Pre-trained word embedding: Incorporates pre-trained word embeddings into an RNN-based neural network, allowing the network to benefit from the semantic relationships captured by GloVe vectors while also providing a pathway to learn from the specific data at hand through the rest of the neural network architecture.
- WikiDataset Class:
  - We handle the Wikitext-2 dataset with our WikiDataset Class. It organizes the dataset into sequences of a specified length (sequence_length). Each item in the dataset is a tuple of (input_sequence, target_sequence), where target_sequence is input_sequence shifted by one token to the right.

- RNN Class:
  - Embedding layer (nn.Embedding)
    - This converts token indices into dense vectors of a specified size (embedding_dim).
  - Recurrent neural network layer (nn.RNN)
    - This processes sequences of embeddings. The RNN has a specified number of hidden units (hidden_dim).
  - Dropout layer (nn.Dropout)

        - This is for regularization, applied after the embedding layer and the RNN output to reduce overfitting.
    - A fully connected layer (nn.Linear)
        - This projects the RNN outputs to the vocabulary size, producing logits for each token in the vocabulary.

Design Choices:

- Sequence Processing:
    - Sequences are processed such that each input sequence has a corresponding target sequence, shifted by one position.
- Embedding Layer:
    - The embedding layer's weights are set to not be updated during training (requires_grad = False) to keep the semantic meanings learned from the large GloVe dataset
- RNN Layer:
    - Processes the sequences of embeddings through time.
- Dropout:
    - Applied after embedding and RNN layers to combat overfitting by randomly zeroing some of the elements.
- Fully Connected Layer:
    - Maps the RNN output to the size of the vocabulary to predict the probability distribution of the next token.

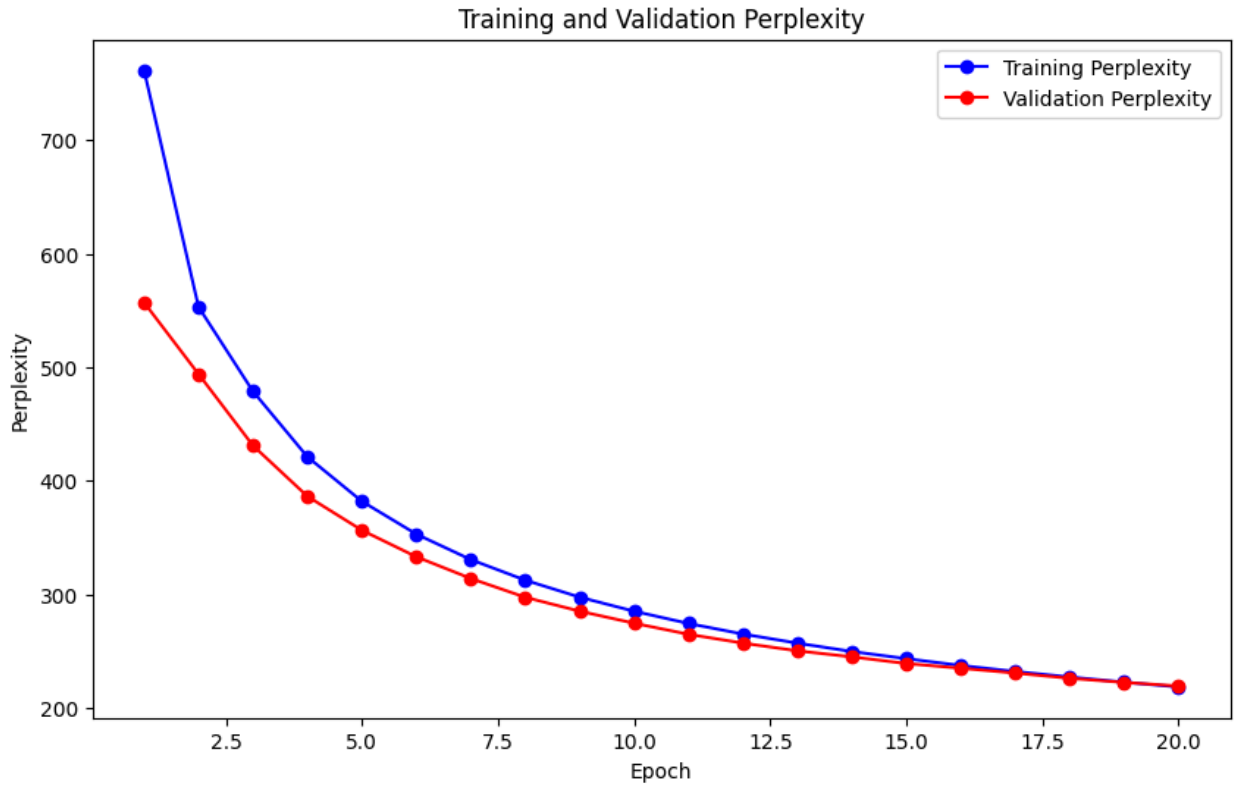
b. list hyper-parameters used by you model an discuss how you selected these values,
Model Hyperparameters:

- Vocabulary Size (vocab_size):
    - This is originally around 33K, and now is set to 10K based on the instruction
- Embedding Dimension (embedding_dim):
    - This is set to 100, based on the instruction
- Hidden Dimension (hidden_dim):
    - The number of features in the hidden state of the RNN. This is set to 128 in the model, based on the instruction and common sense
- Dropout (dropout):
    - The dropout rate applied after the embedding layer and the RNN output to prevent overfitting. In the model, this is set to 0.2, based on common sense.

Training Hyperparameters:

- Batch Size:
    - This is set to 64 based on common sense
- Learning Rate:
    - This is set to 0.0001
- Number of Epochs:
    - This is set to 20, based on the instruction

c. provide learning curves of perplexity vs. epoch on the training and validation sets, and



d. provide final test set perplexity.

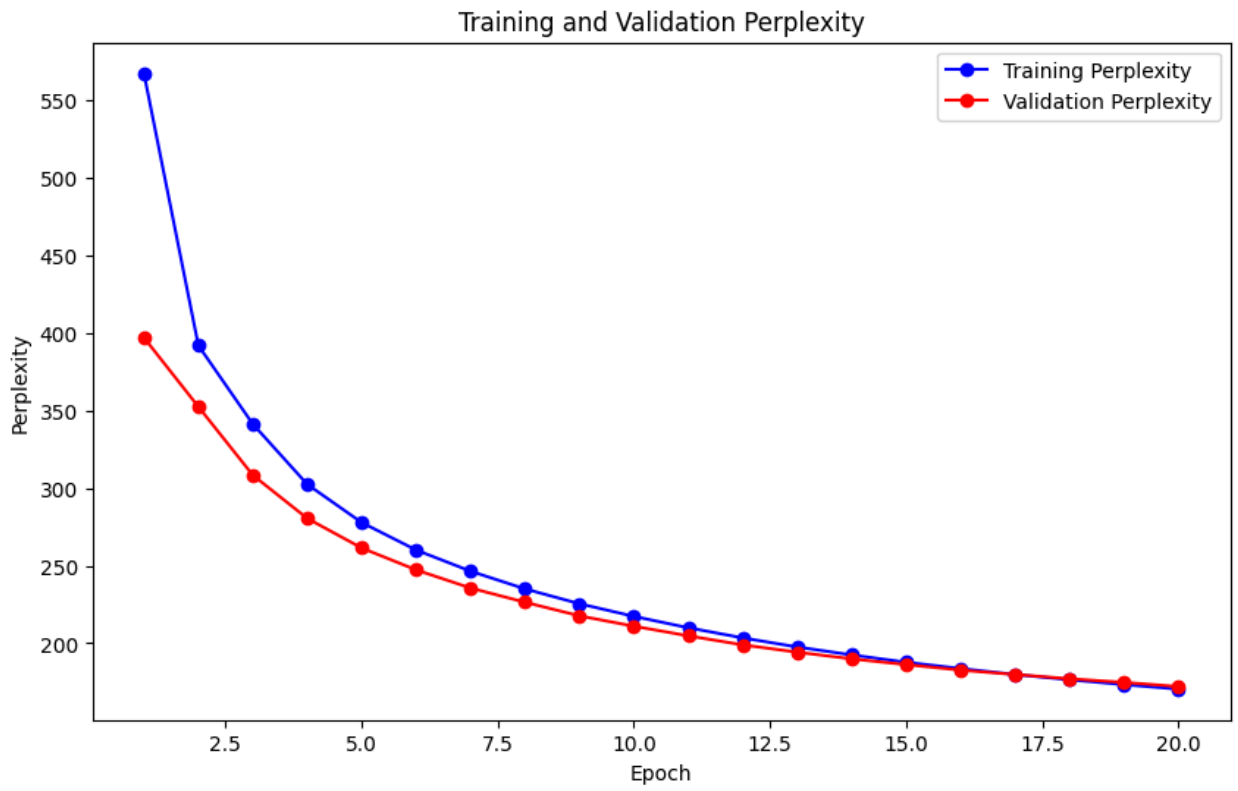
- Final test perplexity 206.835

2. (2.0 points) Discuss how you might improve this “vanilla” RNN language modeling architecture.

One approach is vocabulary richness. We can focus on enhancing vocabulary representation by normalizing words to their base or root form, for example, we will consider “eats”, “eating”, “ate” as the same word “eat”. This technique is known as lemmatization. This approach allows the model to generalize better across different forms of a word, effectively increasing the informational content that can be represented within a fixed vocabulary size. We could also implement weight decaying to prevent overfitting and deeper architecture that stacks multiple RNN layers to learn more complex features

3. (2.0 Bonus Points) Implement one (or more) of the improvements mentioned above, and provide a new set of learning curves and final test perplexity.

We implemented lemmatization to improve our model.



Test Perplexity: 163.323