CS 449

Homework 4

Group 14: Akash Vikram Shroff, Alexis Diaz-Waterman, Seung Hee Daniel Lee, Matthew Britt-Webb

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 your architecture and discuss design choices

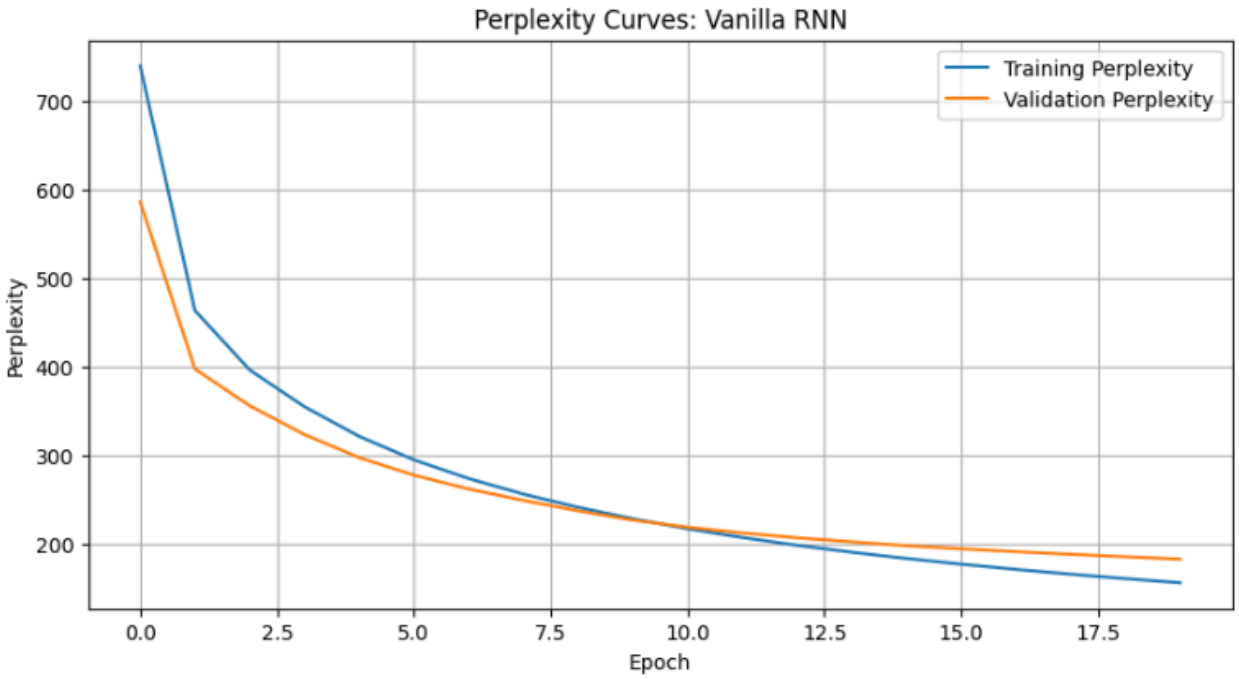
- We began by processing our data and thresholding word frequencies to reduce the corpus size from ~33k to ~10k unique words. We then converted our tokens to integers using the id of the first occurrence and loaded our train and valid tokens into batches with appropriate sequence lengths and targets (discussed more in the next question).
- Our “vanilla” RNN is comprised of an embedding layer (which uses nn.Embedding with dimensions, vocab size x emebdding_dim = 100). Then, we have a single RNN layer with parameters (100, hidden_dim = 512). This is followed by a linear layer that represents the output logits and has dimension (512 x vocab size). We chose to use a hidden layer with 512 nodes to capture the intricate relationships between sequential data. This value was chosen bearing in mind our computational abilities as well as performance on the validation set (more to follow in the next question). Moreover, our hidden state is maintained separately and is recycled over all the batches for an epoch (unless it is the last batch and has a different size).

b. list hyper-parameters used by your model and discuss how you selected these values

- Batch size = 5120. We chose this bearing in mind our computational ability (on an accelerated V100 GPU). We wanted to balance a high enough batch size that we can reasonably accommodate on the GPU as well as data fragmentation. If our batch size was too low (and we had many more batches), we could end up with a case where each batch only has about a few hundred tokens which might not be enough for our model to feasibly learn from.
- Sequence Length = 24. We chose this by finetuning performance against our validation set. This hyperparameter is effectively the number of unrolled time steps in our BPTT. We want to unroll enough to understand sequential relationships but too much might cause issues with vanishing or exploding gradients.
- Learning Rate = 0.0001. Again chosen by finetuning against the validation set.
- Hidden Layer = 512 nodes. Here, we had to bear in mind computational requirements as well as overfitting - too many hidden layer nodes and our model might overfit against the training data and too few might mean that our model simply can’t learn enough. We chose 512 by analyzing the loss curves of the validation set and ensuring that we didn’t overfit and steadily learnt more over our epochs.

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

- Our perplexity values plotted against epochs for the training and validation sets are as follows:



d. provide final test set perplexity.

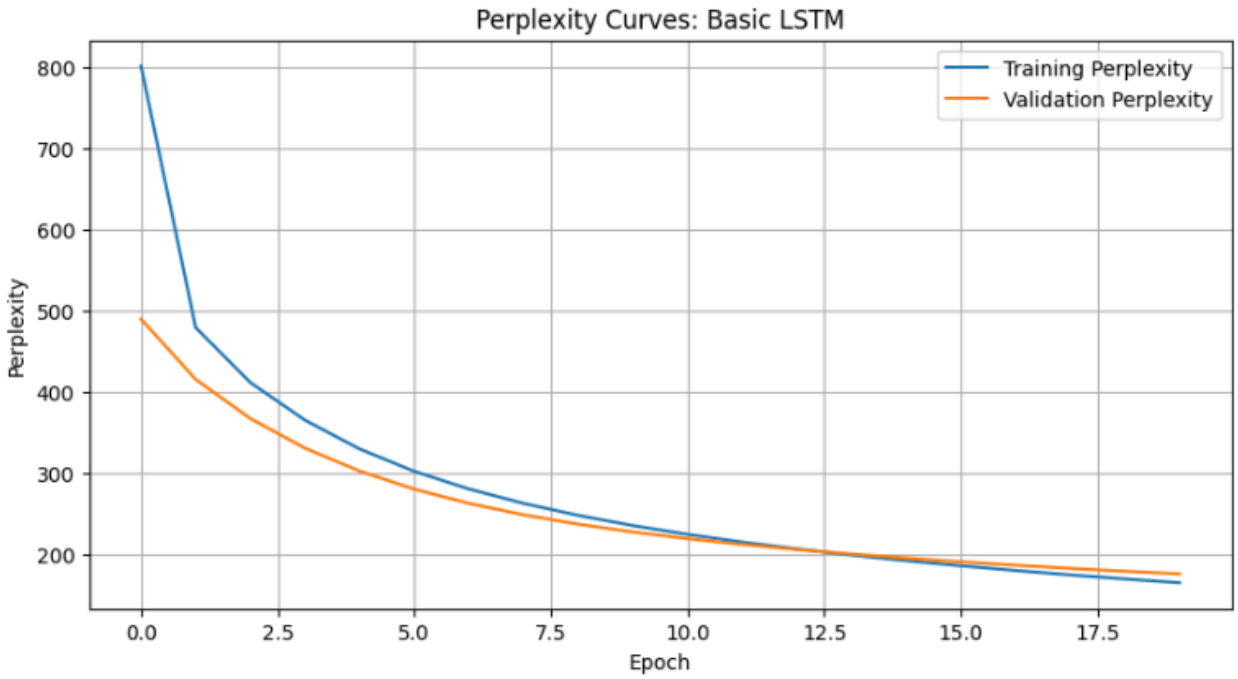
- Our final test set perplexity using the Vanilla RNN was roughly **182.823**, which is below the required threshold of 250.

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

- The lowest-hanging fruit when it comes to improving any ML model is adding some form of regularization. Given that our model doesn’t necessarily struggle with overfitting, regularization might not provide the most impact. Another potential avenue is to improve the complexity of our RNN itself - stacking more RNN layers in our model. However, this comes with its own set of computational challenges.
- A common way to improve RNN architecture is to use one of the variations that are better suited to handling certain kinds of sequential data. For example, a bidirectional RNN would be better suited for tasks like text classification. More advanced architectures like LSTMs or GRUs can improve the model's ability to capture longer-range sequential relations and could improve performance.

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 chose to improve our RNN by architecting an LSTM model. Specifically, we wanted to observe changes in performance using the exact same hyperparameters and only swapping out the RNN layer with an LSTM layer. Our state must also now comprise two separate entities - h_t, our hidden state and c_t, our cell state. With only these minor changes and the same training schedule, here are our loss curves:



- Our test set perplexity value was roughly **176.510**, which is an improvement compared to the vanilla RNN, indicating that the model did a better job learning dependencies in our data. With some parameter finetuning, the LSTM performance would certainly improve and demonstrate a bigger gulf in the capabilities of the two models.