Introduction to Natural Language Processing

Assignment 1

PART OF SPEECH TAGGING USING NEURAL NETWORKS

Hardik Sharma

Introduction

This assignment revolves around the task of `POS Tagging`, that is, *Part of Speech Tagging*.

The end-goal of the task is to take a given sentence, and assign, to each token, it's corresponding POS tag. To do so, I have used the data from the Universal Dependencies Dataset's `en-atis` dataset.

To perform the task, I have employed two broad approaches, using *Feed Forward Neural Network*, `FFNN` and *Recurrent Neural Networks*, `RNN`.

Feed Forward Neural Network

Data Preparation

The data is is prepared, by first padding each sentence adequately, based on context window sizes. Thereafter, `dataPoints` are created for the model, based on the context window. The embedding size can be adjusted simply by adding another layer, so we can pass the model the `one-Hot` vectors and expect it to learn the embeddings for the tokens in the first token itself. Note that this is a bit different because the model is allowed to have interaction with other tokens.

Training

The model is trained on the training dataset using a variety of parameters tuned. I have accounted for changes in : [1]

- **Learning Rate**
- **Batch Size**
- **Number of layers in the ANN**
- **Context window**

For the sake of completeness, I have also accounted for assymteric context windows.

Testing

The model is evaluated on the `dev` and `test` datasets. The data for evaluation and inference is prepared in a similar way

Recurrent Neural Networks

Data Preparation

Data preparation is a bit simpler for the case of `RNN`. Each datapoint is simply a sentence. To take into account the difference in the sentence lengths, I pad each sentence to the size of the maximum length sentence using the `pad_sequence` function from the library `torch`.

Training

The following paramters are being tuned and changed in the model. For anyone who is new to RNNs, the outputs of RNN are interpreted through the `hiddenState` at each time step, where a time step is just a token index. So, the output of the RNN is a matrix that contains the information regarding the `hiddenState` at each of the time-steps. To actually perform any downstream task, like POS tagging in

[1] The paramaters can be passed to the model, the default values are present in config.py file.

this case, we pass these output hiddenStates from the model, to a few MLP layers in order to perform the task.

- **Learning Rate**
- **Batch Size**
- **Stack Size**
- **Hidden State Size**
- **Number of layers in the MLP**
- **Epochs**

Bidirectional RNN

You can pass Bidirectionality as a boolean to the `ReccurentNeuralLayer` model, in order to train and save a bidirectional RNN.

Testing

The model is evaluated on the `dev` and `test` datasets. The data for evaluation is prepared in a similar way, but the padding is ignored for the batch, to calculate the model metrics precisely. The inference is performed without the padding since the entire input is treated as a single sentence.

Results

Feed Forward Neural Network

Conclusion

In summary, the exploration of n-Gram language models using Good-Turing and Linear Interpolation on "Ulysses" and "Pride and Prejudice" datasets revealed clear trends in model performance. Linear Interpolation consistently outperformed Good-Turing, displaying lower perplexity scores, indicating better generalization and reduced confusion when predicting sentences.

Interestingly, "Pride and Prejudice" exhibited lower perplexity scores overall compared to "Ulysses," suggesting a smoother predictability in the former's language patterns.