Introduction to Natural Language Processing

Assignment 2

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

Here I discuss only the directly relevant parameters and metrics. The graphs of `dev-accuracy` during training and `train-loss` that are satandard and do not proide much of an insight for the given parameters, are saved in the `./plots/` directory.

Feed Forward Neural Network

I trained and evaluated multiple models on the `en-atis` dataset. The results for the top 5 models are shown below, with metrics over the 5 metrics, Accuracy, Precision, F1-Macro, F1-micro, Recall Score

Model Name	Accuracy	Precision	F1-macro	F1-micro	Recall
p=2 s=1 n=1 : [128]	**98.7234%**	**98.7492%**	**97.1516%**	**98.7234%**	**98.7234%**
p=2 s=1 n=2 : [128,64]	98.7234%	98.7398%	97.103%	98.7234%	98.7234%
p=2 s=1 n=1 : [32]	98.6474%	98.6779%	96.9825%	98.6474%	98.6474%
p=3 s=1 n=2 : [128,64]	98.6322%	98.6417%	96.7312%	98.6322%	98.6322%
p=3 s=1 n=1 : [64]	98.617%	98.6312%	96.8527%	98.617%	98.617%

The best performing model is therefore obtained for the `p=2 s=1 n=1:[128]`, which is shown in the table above. It also achieves the best scores in all the other metrics the models are evaluated on.

The confucsion metric for the model is given below:

Remarks

The models simplicity in the number of layers and context window seemed to have struck a nice balance with the large size of the hidden layer to optimize its performance for the given task.

Also, the top 3 models are the same, if measured across any of these metrics, and the common hyperparameter among all these is `p=2, s=1`. Hinting that a simplistic, and *past-biased* model tends to perform better on `POS tagging`.

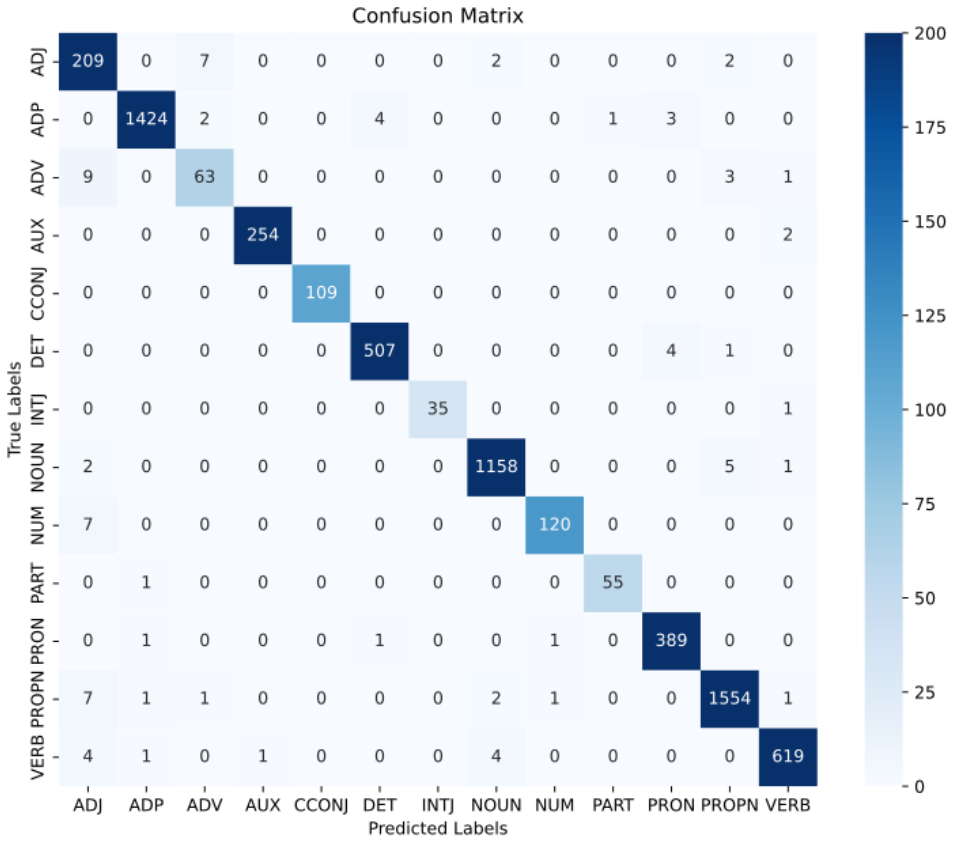

Figure 1: Confusion Matrix

Recurrent Neural Networks

I trained and evaluated approximately 150 models for hypereparameter tuning. The results for the top 5 models are as follows :

Model Name	Accuracy	Precision	F1-macro	F1-micro	Recall
$s = 3, b = True, h = 64, l = 1 : [64]$	**98.7994%**	**98.836%**	90.0127%	**98.7994%**	**98.7994%**
$s = 2, b = True, h = 64, l = 2 : [128, 64]$	98.7538 %	98.7542%	97.0912%	98.7538%	98.7538%
$s = 2, b = True, h = 128, l = 2 : [128, 64]$	98.7234 %	98.7434%	96.6337%	98.7234%	98.7234%
$s = 3, b = True, h = 64, l = 2 : [64, 64]$	98.7082 %	98.7156%	**97.344%**	98.7082%	98.7082%
$s = 2, b = True, h = 64, l = 2 : [64, 64]$	98.693 %	98.7206%	89.6664%	98.693 %	98.693 %

Here s denotes the stack size of the RNN, b denotes if the RNN is bidirectional, h denotes the hidden state size, and l denotes the number of layers in the classifier, followed by the layer sizes themselves.

Remarks

Similar to the FFNN models, the RNN models underwent extensive hyperparameter tuning to achieve optimal performance. Notably, the model with parameters `s=3, b=True, h=64, l=1 : [64]` outperformed others in terms of accuracy and precision, albeit with a slightly lower F1-macro score compared to some other models. This is due to the class `SYM` that has just a single point, predicting that incorrectly costs the model heavily.

Confusion Matrix of the best performing model :

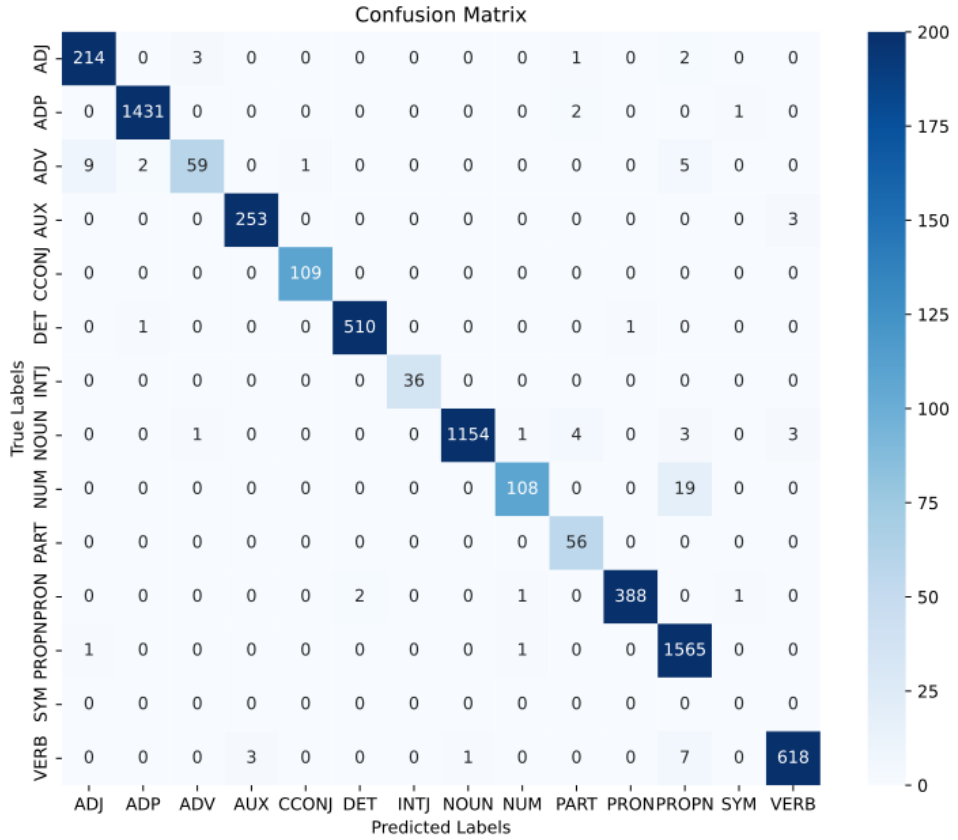

Figure 2: Confusion Matrix of the best performing model

The bidirectional nature of certain models ($b = True$) seems to contribute positively to their performance, especially when combined with larger hidden state sizes. However, it's crucial to note that in some cases, increasing the complexity of the model (such as adding more layers or increasing hidden state sizes) did not necessarily lead to improved performance across all metrics.

Additionally, as seen in the top-performing models, a shallower architecture with fewer layers but appropriately sized hidden states ($l = 1$) can sometimes yield competitive results, emphasizing the importance of balancing model complexity with dataset characteristics.

Conclusion

The detailed examination of the top-performing models provides insights into the impact of hyperparameters on model performance, highlighting the importance of careful selection and tuning to achieve optimal results. Furthermore, the comparative analysis between FFNN and RNN architectures sheds light on their respective strengths and weaknesses in tackling the POS tagging task.

Moving forward, further exploration could involve experimenting with additional architectural variations, exploring different training strategies, and investigating the generalizability of the models across diverse datasets and languages. Overall, the findings contribute to advancing the understanding of neural POS tagging methods and offer valuable insights for future research and practical applications in natural language processing.