Assignment 3

Experimental Protocol
To solve the Part-of-Speech (POS) tagging task, we designed a rigorous experimental protocol to ensure our results were reliable and reproducible.

The dataset consisted of approximately 3,914 sentences. To allow the model to generalise to unseen data, the dataset was split into three subsets. The training set (80%) was used to learn the model parameters, the validation set (10%) was used for hyperparameter tuning and monitoring overfitting during training, and the test set (10%) was reserved exclusively for the final evaluation of the selected model.

We established a consistent training environment using a fixed random seed (42) for reproducibility. Both models were trained for 30 epochs with a batch size of 32. We used the CrossEntropyLoss function, which is standard for multi-class classification tasks, and carefully configured it to ignore padding indices so that the model was not penalised for predicting padding tokens. The Adam optimiser was chosen for its ability to adapt learning rates, paired with a learning rate scheduler (ReduceLROnPlateau) that reduced the learning rate if validation loss stopped improving, allowing the model to finetune its weights in later epochs.

Dataloader and Pre-processing
Effective data management is crucial for sequence labelling. We implemented a custom POSDataset and DataLoader in PyTorch to handle the specific requirements of variable-length sentences.

Pre-processing and Encoding: The raw text data was first converted into numerical tensors. Two separate vocabularies were created: one for words and one for POS tags.

1. Lowercasing: All words were converted to lowercase to reduce the overall vocabulary size and improve generalisation, ensuring tokens such as *"The"* and *"the"* were treated identically.
2. Special Tokens: An <UNK> (unknown) token was included to handle words in the test set that were not present during training. In addition, a <PAD> token (index 0) was used to support variable sentence lengths during batching.

The Dataloader and Collation: Since sentences naturally vary in length, they cannot be stacked directly into batches. We implemented a custom ‘collate_fn’ (collation function) to address this. This function pads all sentences in a single batch to the length of the longest sentence using the <PAD> token. Crucially, we also captured the original lengths of the sentences before passing. These lengths were passed to the model to ensure that we could mask out padding during training, ensuring the network only learned from actual data.

Network Architectures
Two neural network architectures were evaluated to compare their effectiveness on the task: a Bidirectional LSTM (BiLSTM) and a Transformer.

1. Bidirectional LSTM (BiLSTM)
The first model utilised a Bidirectional Long Short-Term Memory (LSTM) network.

- Embedding Layer: Maps word indices to dense vectors (128 dimensions).
- BiLSTM Layers: We selected a bidirectional architecture because correctly tagging a word often depends on the context from both the left (previous words) and the right (future words).
  We used 2 layers with a hidden dimension of 128
- Packed Sequences: A key implementation detail was using ‘pack_padded_sequence’. This PyTorch utility allows the RNN to process variable-length sequences efficiently by skipping calculation on padding tokens, significantly speeding up training.
- Classifier: The output is passed through a Dropout layer (0.3) to prevent overfitting, followed by a fully connected linear layer that maps the hidden states to the 33 possible POS tags.

2. Transformer Encoder
The second model was based on the Transformer architecture, which relies on self-attention mechanisms rather than recurrence.

- Positional Encoding: Unlike RNNs, Transformers process data in parallel and have no inherent sense of order. We added positional encodings to the input embeddings so the model could understand the sequence of words.
- Encoder Block: We used a Transformer Encoder with 2 layers and 4 attention heads. The multi-head attention allows the model to

attend to different parts of the sentences simultaneously to derive context.
- Masking: We implemented a ‘src_key_padding_mask’ to ensure the attention mechanism ignored padding tokens, preventing them from influencing the prediction of valid words.

Training Results and Discussion

We trained both models and monitored their Loss and Accuracy on both training and validation sets. The results are summarised below:

Model	Best Validation Accuracy
BiLSTM	91.83%
Transformer	86.9%

The BiLSTM outperformed the Transformer on this dataset, achieving a validation accuracy of nearly 92% compared to around 87% for the Transformer.

The difference is likely related to the size of the dataset. Transformers typically require large amounts of data to learn effective representations because they lack the inductive bias for sequential data that RNNs possess. With only 3000 sentences, the BiLSTM was able to leverage its inherent understanding of sequential order to learn linguistic patterns more efficiently. This is also reflected in the training curves, where the BiLSTM loss decreased more steadily, while the Transformer showed signs of underfitting.

Based on these results, the BiLSTM was selected as the best model for final testing.

Final Results

We evaluated the best model (BiLSTM) on the held-out Test Set to assess its real-world performance.

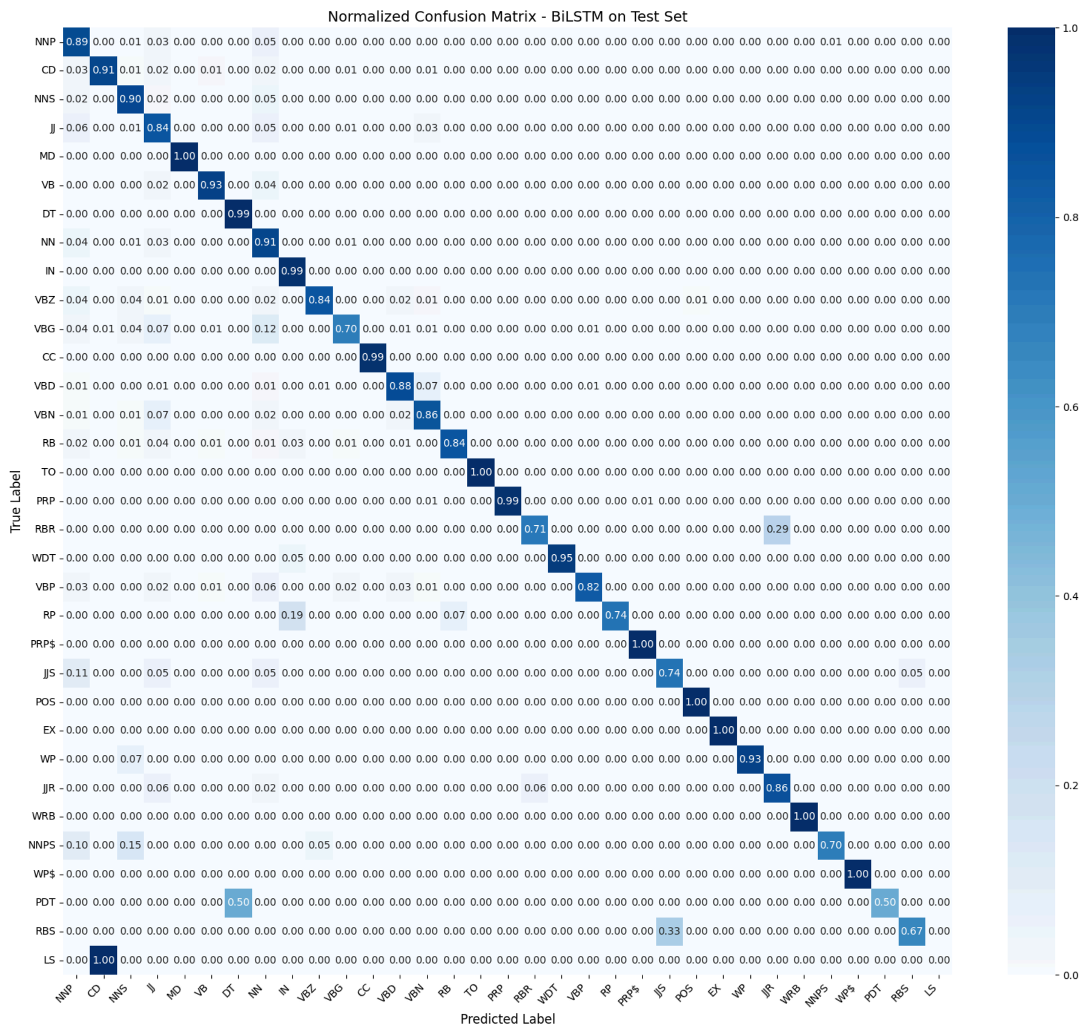
Test Set Performance

Accuracy	0.9163
Macro F1-score	0.8576

Weighted Precision	0.9183
Weighted REcall	0.9163
Weighted F1-Score	0.9166

The model maintained its high performance on the test set, confirming that it has generalised well and did not simply memorise the training data.

Confusion Matrix Analysis: To understand the errors, we analysed the confusion matrix.



The diagonal of the matrix is strongly populated, indicating that the majority of POST tags are correctly classified. Many common and function-based tags (e.g. DT, IN, CC, TO, PRP) achieve near perfect accuracy, demonstrating that the model effectively captures contextual grammatical structure.

Most errors occur between closely related tags.

- PDT (Predeterminers) vs DT (Determiners):
  For the true label PDT, only 50% of instances are correctly classified, with the remaining 50% misclassified as DT. This confusion is expected, as predeterminers (e.g. ‘all’, ‘both’) are closely related to determiners and often appear in similar positions.
- RBS (Superlative Adverbs) vs JJS (Superlative Adjectives):
  For RBS, 33% of instances are incorrectly predicted as JJS. This is because they’re very similar and is not easy to distinguish between the two even with contextual information.
- LS (List Item Marker) vs CD (Cardinal Number):
  The LS tag is entirely misclassified as CD, with a value of 1. THis shows the model can’t process the meaning of the full stop after the number.

With all this being said, the errors occurred in categories that were underrepresented so the model had less data to be trained on it.

Conclusion
Overall, the confusion matrix supports the strong performance of the BiLSTM model. Most errors arise from similar linguistic categories and underrepresented POS tags in the data, rather than a model failure. The bidirectional architecture is effective for resolving the majority of contextual ambiguities in POS tagging.