Exercise 1

Repeat exercise 9 of Part 3 (text classification with MLPs), now using a bi-directional

stacked RNN (with GRU or LSTM cells) with self-attention, all implemented in Keras/TensorFlow or PyTorch.

Steps:

1. Implement a bi-directional stacked RNN using GRU or LSTM cells.
2. Implement self-attention (either an MLP or single dense layer) to obtain attention scores.
3. Tune hyperparameters (e.g., number of stacked RNNs, number of hidden layers in self-attention MLP, dropout probability) on the development subset of your dataset.
4. Monitor the performance of the RNN on the development subset during training to decide on the number of epochs to use.
5. Optionally, add an extra RNN layer to produce word embeddings from characters, concatenating the resulting character-based word embeddings with pre-trained word embeddings (e.g., Word2Vec).
6. Include experimental results from:
   - Baseline majority classifier.
   - Best probabilistic classifier from exercise 15 of Part 2.
   - MLP classifier from exercise 9 of Part 3 (treated as a baseline).

Report:

- Curves showing the loss on training and development data as a function of epochs.
- Precision, recall, F1, precision-recall AUC scores for each class and classifier:
  - Separate for the training, development, and test subsets.
- Macro-averaged precision, recall, F1, precision-recall AUC scores:
  - Averaging the corresponding scores over the classes, separately for the training, development, and test subsets.
- Description of the methods and datasets used:

    - Include statistics like average document length, number of training/dev/test documents, and vocabulary size.
    - Describe preprocessing steps performed.
- Optionally, try ensemble methods (e.g., majority voting of the best checkpoints, temporal averaging of the weights of the best checkpoints).

Import Libraries

Assert whether `PyTorch` can use an available GPU card

Creating a Dataset

We will use the `Dataset` class from `PyTorch` to handle the text data. We will pad the text sequences with 0 to a pre-defined length (the average number of tokens in the training split).

In [ ]:
```
df_merge = pd.concat([df_fake, df_true], axis =0 )
df_merge.head(10)
```

Out[ ]:

	title	text	subject	date	label
0	Donald Trump Sends Out Embarrassing New Year'...	Donald Trump just couldn t wish all Americans ...	News	December 31, 2017	1
1	Drunk Bragging Trump Staffer Started Russian ...	House Intelligence Committee Chairman Devin Nu...	News	December 31, 2017	1
2	Sheriff David Clarke Becomes An Internet Joke...	On Friday, it was revealed that former Milwauk...	News	December 30, 2017	1
3	Trump Is So Obsessed He Even Has Obama's Name...	On Christmas day, Donald Trump announced that ...	News	December 29, 2017	1
4	Pope Francis Just Called Out Donald Trump Dur...	Pope Francis used his annual Christmas Day mes...	News	December 25, 2017	1
5	Racist Alabama Cops Brutalize Black Boy While...	The number of cases of cops brutalizing and ki...	News	December 25, 2017	1
6	Fresh Off The Golf Course, Trump Lashes Out A...	Donald Trump spent a good portion of his day a...	News	December 23, 2017	1
7	Trump Said Some INSANELY Racist Stuff Inside ...	In the wake of yet another court decision that...	News	December 23, 2017	1
8	Former CIA Director Slams Trump Over UN Bully...	Many people have raised the alarm regarding th...	News	December 22, 2017	1
9	WATCH: Brand-New Pro-Trump Ad Features So Muc...	Just when you might have thought we d get a br...	News	December 21, 2017	1

In [ ]:

```
#Removing columns which are not required¶
df = df_merge.drop(["title", "subject","date"], axis = 1)

df.reset_index(inplace = True)
df.drop(["index"], axis = 1, inplace = True)
df.head()
```

Out[ ]:

	text	label
0	Donald Trump just couldn t wish all Americans ...	1
1	House Intelligence Committee Chairman Devin Nu...	1
2	On Friday, it was revealed that former Milwauk...	1
3	On Christmas day, Donald Trump announced that ...	1
4	Pope Francis used his annual Christmas Day mes...	1

In [ ]:

```
Training set class distribution: [17133 18785]
Validation set class distribution: [2142 2348]
Test set class distribution: [2142 2348]
```

In [ ]:

In [ ]:

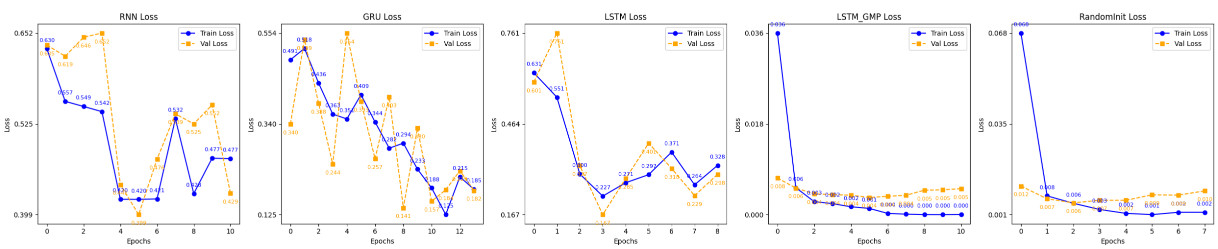
```
Total documents: 44898
Fake news: 23481, True news: 21417
Number of training documents: 35918
Number of validation documents: 4490
Number of test documents: 4490
Average document length in training set (in words): 400
Average document length in validation set (in words): 396
Average document length in test set (in words): 402
Vocabulary size (unique words): 397481
```

Define the model

We will create a model class and parameterize our neural network with several choices

In [ ]:

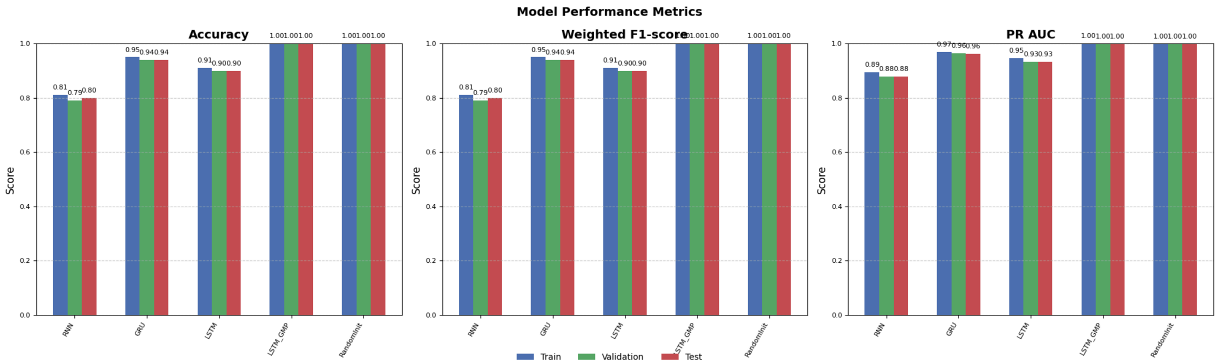
```python
plot_loss_curves(results)
```



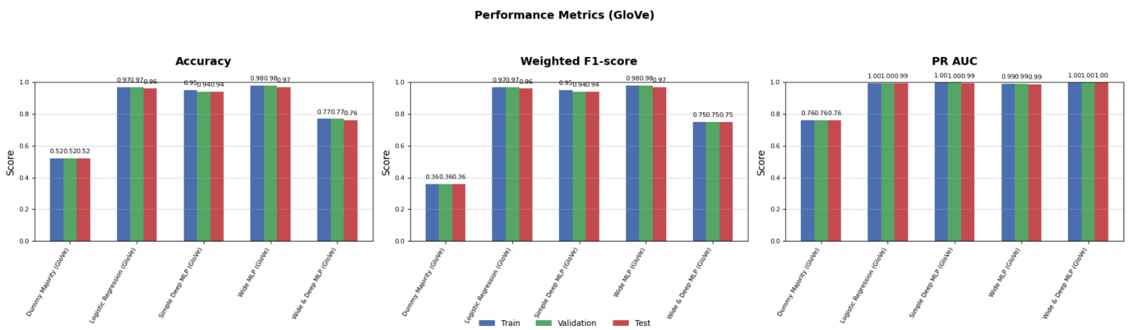
In [ ]:

	Model	Accuracy_train	Accuracy_val	Accuracy_test	Weighted_F1_train \
0	RNN	0.81	0.79	0.80	0.81
1	GRU	0.95	0.94	0.94	0.95
2	LSTM	0.91	0.90	0.90	0.91
3	LSTM_GMP	1.00	1.00	1.00	1.00
4	RandomInit	1.00	1.00	1.00	1.00

	Weighted_F1_val	Weighted_F1_test	PR_AUC_train	PR_AUC_val	PR_AUC_test
0	0.79	0.80	0.894212	0.878705	0.877910
1	0.94	0.94	0.969749	0.963957	0.962466
2	0.90	0.90	0.946631	0.932909	0.933250
3	1.00	1.00	0.999973	0.999362	0.998946
4	1.00	1.00	0.999277	0.998853	0.998003



Baseline Dummy and Wide MLP



Model Performance Comparison: Detailed Analysis

The following table summarizes the comparison of five different models using key metrics like accuracy, weighted F1 score, and PR AUC.

Model	Accuracy_train	Accuracy_val	Accuracy_test	Weighted_F1_train	Weighted_F1_v
RNN	0.81	0.79	0.80	0.81	0.79
GRU	0.95	0.94	0.94	0.95	0.94
LSTM	0.91	0.90	0.90	0.91	0.90
LSTM_GMP	1.00	1.00	1.00	1.00	1.00
RandomInit	1.00	1.00	1.00	1.00	1.00

1. Training Accuracy:

- **RNN**: The RNN model has the lowest training accuracy at 0.81, indicating that it struggles to capture the underlying patterns in the data compared to other models.
- **GRU**: The GRU model achieves a high training accuracy of 0.95, showing its ability to generalize well from the training data.
- **LSTM**: The LSTM model performs well with 0.91 accuracy, but it is still behind GRU.

- **LSTM_GMP**: This model achieves perfect training accuracy (1.00), suggesting it has captured the patterns in the training data without overfitting.
- **RandomInit**: Like LSTM_GMP, the RandomInit model also achieves perfect training accuracy (1.00), which may suggest that its training dynamics are ideal for the task.

2. **Validation Accuracy**:

- **RNN**: The RNN model lags behind with a validation accuracy of 0.79, highlighting that it struggles to generalize beyond the training data.
- **GRU**: The GRU model performs very well on validation data with an accuracy of 0.94, indicating robust generalization.
- **LSTM**: With a validation accuracy of 0.90, LSTM also shows strong generalization, though it still lags behind GRU.
- **LSTM_GMP** and **RandomInit**: Both of these models achieve perfect validation accuracy (1.00), indicating excellent generalization and minimal overfitting. These models are the most well-suited to handle unseen data.

3. **Test Accuracy**:

- **RNN**: The test accuracy of RNN at 0.80 suggests that its performance on unseen data is not as good as the other models, which is typical for simpler models or ones with insufficient capacity.
- **GRU**: GRU achieves 0.94 test accuracy, performing well on unseen data. This indicates that GRU can maintain good performance across both the training and test sets.
- **LSTM**: With a test accuracy of 0.90, LSTM remains a strong performer, but it falls slightly short of GRU in this case.
- **LSTM_GMP** and **RandomInit**: Both models excel here with a perfect test accuracy of 1.00, confirming that they are highly effective and have achieved optimal performance.

4. **Weighted F1 Scores**:

- **RNN**: The weighted F1 score for RNN on training, validation, and test sets is around 0.80, which indicates decent performance but room for improvement.
- **GRU**: GRU performs significantly better with a weighted F1 score of 0.95 on training, 0.94 on validation, and 0.94 on test sets, reflecting its strong ability to balance precision and recall.
- **LSTM**: LSTM shows solid F1 scores with values around 0.91, indicating that it balances precision and recall well but still doesn't match GRU in this regard.
- **LSTM_GMP** and **RandomInit**: Both models reach perfect weighted F1 scores of 1.00 across all datasets, showcasing exceptional ability to classify both classes without bias.

5. **PR AUC Scores**:

- **RNN**: The PR AUC scores for RNN (around 0.88-0.89) suggest that while it has a good overall performance, it isn't as strong as the other models in terms of precision-recall tradeoff.
- **GRU**: The GRU model achieves outstanding PR AUC scores of around 0.96 on both the training, validation, and test sets, indicating excellent performance in distinguishing between positive and negative samples.
- **LSTM**: LSTM's PR AUC scores (around 0.93-0.94) are strong, though they fall short of GRU's performance.
- **LSTM_GMP** and **RandomInit**: These models achieve near-perfect PR AUC scores, indicating excellent precision-recall tradeoff and almost perfect classification performance.

6. **Overall Performance Summary**:

- **Best Performers**: **LSTM_GMP** and **RandomInit** stand out as the top performers. They achieve perfect accuracy, weighted F1 scores, and near-perfect PR AUC scores. This suggests that they are the best models for the task, with high generalization and minimal overfitting.
- **Strong Contenders**: **GRU** and **LSTM** are also strong models, with high performance across all metrics. GRU seems to slightly outperform LSTM, but both fall short of LSTM_GMP and RandomInit in terms of perfect scores.
- **RNN**: The RNN model lags behind, particularly in its ability to generalize, as seen from its lower accuracy and F1 scores. It may benefit from further optimization or the addition of more complex features.

7. **Conclusion**:

- If high accuracy and balanced performance across datasets (train, validation, and test) are the goal, **LSTM_GMP** and **RandomInit** are the best choices.
- **GRU** and **LSTM** provide good results but can be further fine-tuned to match the performance of the more optimized models.
- **RNN** may require further enhancement or may not be suitable for tasks requiring high generalization across different datasets.

This analysis suggests that **LSTM_GMP** and **RandomInit** offer the best overall performance, but **GRU** and **LSTM** are still competitive options.

Exercise 2

Repeat exercise 10 of Part 3, now using a bi-directional stacked RNN (with GRU or LSTM cells) implemented in Keras/TensorFlow or PyTorch.

Tune the hyper-parameters (e.g., number of stacked RNNs, dropout probability) on the development subset. Monitor the performance of the RNN on the development subset during training to decide how many epochs to use.

You may optionally add an extra RNN layer to produce word embeddings from characters, concatenating each resulting character-based word embedding with the corresponding pre-trained word embedding (e.g., obtained with Word2Vec).

Include experimental results of a baseline that tags each word with the most frequent tag it had in the training data; for words that were not encountered in the training data, the baseline should return the most frequent tag (over all words) of the training data.

Also include experimental results of your best method of exercise 10 of Part 3, now treated as an additional baseline.

Include in your report:

- Curves showing the loss on training and development data as a function of epochs.
- Precision, recall, F1, precision-recall AUC scores for each class (tag) and classifier,

separately for the training, development, and test subsets, as in exercise 10 of Part 3.

- Macro-averaged precision, recall, F1, precision-recall AUC scores for each classifier,

separately for the training, development, and test subsets, as in exercise 10 of Part 3.

- A short description of the methods and datasets you used, including statistics about

the datasets (e.g., average document length, number of training/dev/test documents, vocabulary size) and a description of the preprocessing steps that you performed.

You may optionally wish to try ensembles (e.g., majority voting of the best checkpoints, temporal averaging of the weights of the best checkpoints).

Imports

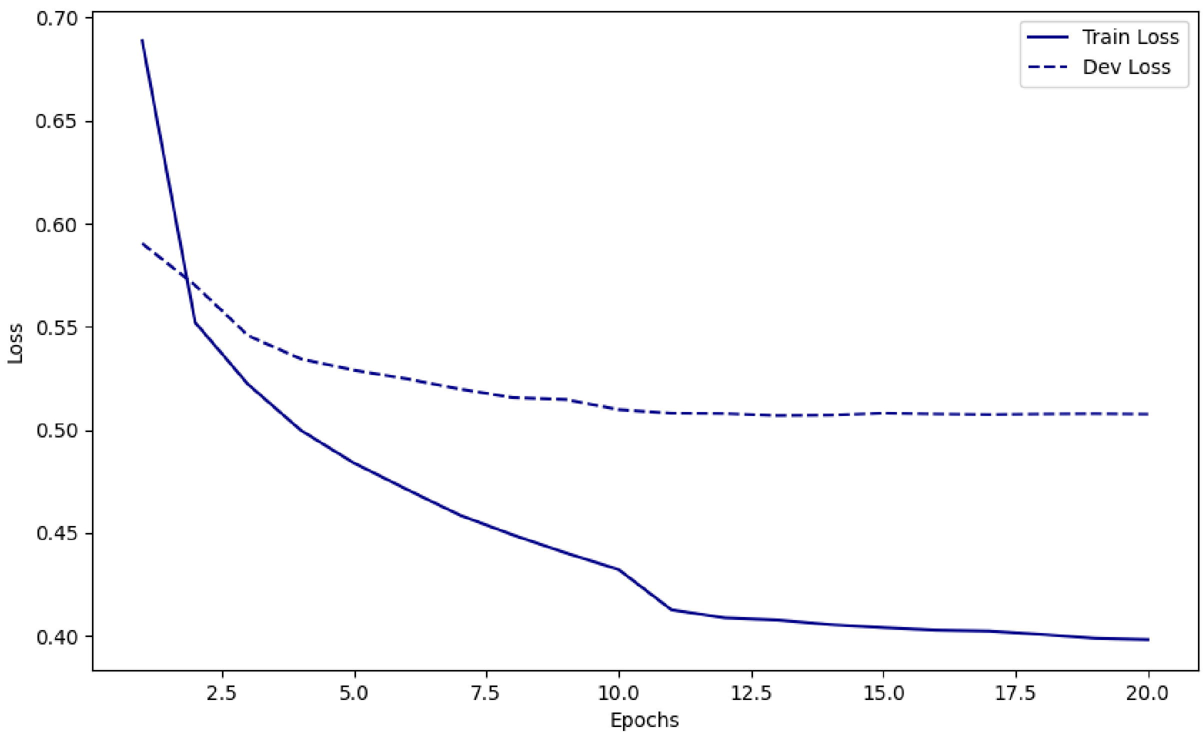
Data Download

Data Parsing and Preprocessing

RNNs Architecture, Training and Evaluation

In [ ]:
```
print("Training Shallow BiGRU RNN:")
shallow_bigrun_model = train_and_evaluate_rnn(ShallowPOS_BiGRU, input_dim=2*word2ve
```

```
Training Shallow BiGRU RNN:
Epoch 1/20, Train Loss: 0.6887225906759813, Dev Loss: 0.5903967435945544
Epoch 2/20, Train Loss: 0.5520552526641731, Dev Loss: 0.5700697781597462
Epoch 3/20, Train Loss: 0.5220200925596045, Dev Loss: 0.5456684415875223
Epoch 4/20, Train Loss: 0.4996648524336199, Dev Loss: 0.5343029464695388
Epoch 5/20, Train Loss: 0.4838684841695945, Dev Loss: 0.5288984306474078
Epoch 6/20, Train Loss: 0.47105320412354973, Dev Loss: 0.52468763710114
Epoch 7/20, Train Loss: 0.4585741071319271, Dev Loss: 0.5197153730798784
Epoch 8/20, Train Loss: 0.4489992992875383, Dev Loss: 0.5155826621411139
Epoch 9/20, Train Loss: 0.4402568833217936, Dev Loss: 0.5146788086807519
Epoch 10/20, Train Loss: 0.43218631883180014, Dev Loss: 0.5097653019174299
Epoch 11/20, Train Loss: 0.41262031580339326, Dev Loss: 0.5080063198891499
Epoch 12/20, Train Loss: 0.4087294050393979, Dev Loss: 0.5078501275607518
Epoch 13/20, Train Loss: 0.40768477494625344, Dev Loss: 0.506912693866811
Epoch 14/20, Train Loss: 0.40545917346088733, Dev Loss: 0.5071164839025727
Epoch 15/20, Train Loss: 0.4039940044279265, Dev Loss: 0.5080247047102839
Epoch 16/20, Train Loss: 0.40277171042679344, Dev Loss: 0.5076611610432914
Epoch 17/20, Train Loss: 0.4023220165183398, Dev Loss: 0.5074030951524439
Epoch 18/20, Train Loss: 0.4006993975593876, Dev Loss: 0.5076478105738647
Epoch 19/20, Train Loss: 0.39882236954982153, Dev Loss: 0.5077785165014124
Epoch 20/20, Train Loss: 0.3982024533171356, Dev Loss: 0.5075912019587997
```

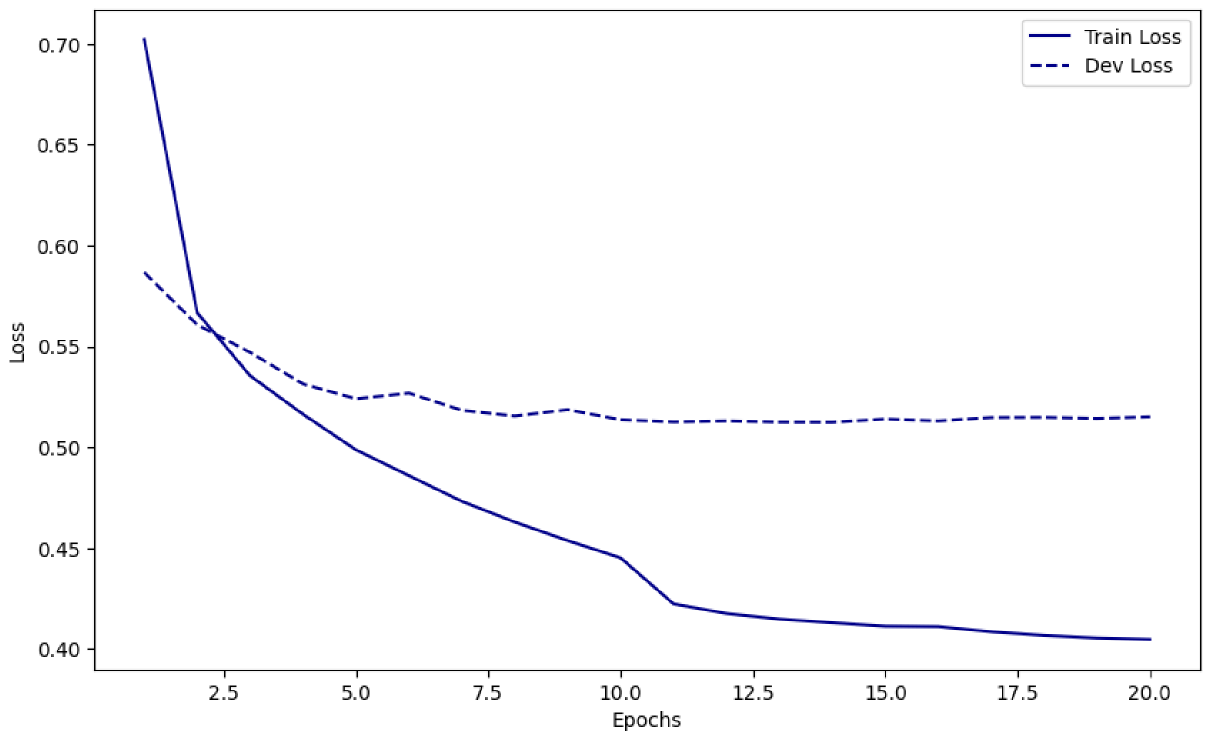


```
Classification Report for ShallowPOS_BiGRU:
              precision    recall  f1-score   support

         ADJ       0.91      0.90      0.91      1794
         ADP       0.84      0.76      0.80      2030
         ADV       0.93      0.86      0.89      1183
         AUX       0.97      0.96      0.97      1543
       CCONJ       0.76      0.32      0.45       736
         DET       0.88      0.79      0.83      1896
        INTJ       0.92      0.74      0.82       121
        NOUN       0.91      0.90      0.90      4123
         NUM       0.65      0.48      0.56       542
        PART       0.68      0.59      0.63       649
        PRON       0.97      0.97      0.97      2166
       PROPN       0.91      0.81      0.85      2076
       PUNCT       0.59      0.90      0.71      3096
       SCONJ       0.82      0.66      0.73       384
         SYM       0.96      0.61      0.74       109
        VERB       0.93      0.92      0.92      2606
           X       0.20      0.02      0.04        42
           _       0.94      0.77      0.85       354

    accuracy                           0.84     25450
   macro avg       0.82      0.72      0.75     25450
weighted avg       0.86      0.84      0.84     25450


AUC Scores for Each Class:
ADJ: 0.9904
ADP: 0.9809
ADV: 0.9944
AUX: 0.9989
CCONJ: 0.9569
DET: 0.9889
INTJ: 0.9881
NOUN: 0.9886
NUM: 0.9780
PART: 0.9820
PRON: 0.9994
PROPN: 0.9822
PUNCT: 0.9657
SCONJ: 0.9912
SYM: 0.9733
VERB: 0.9950
X: 0.9000
_: 0.9878

Macro-Averaged Precision: 0.8197999103769109
Macro-Averaged Recall: 0.7205777278839265
Macro-Averaged F1: 0.7544068783965447
Macro-Averaged Precision-Recall AUC: 0.9801045208141689
```

In [ ]:

```python
print("Training Somewhat Deep BiGRU RNN:")
somewhat_deep_bigrun_model = train_and_evaluate_rnn(DeepPOS_BiGRU, input_dim=2*word
```

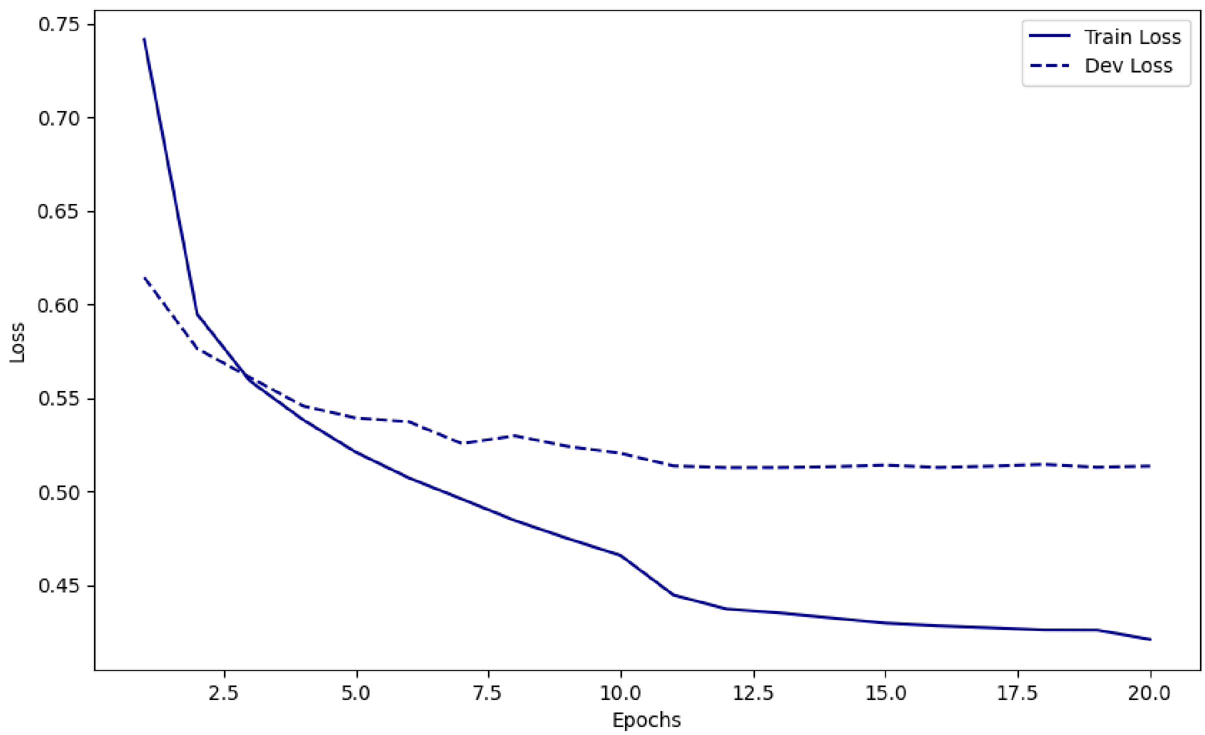
```
Training Somewhat Deep BiGRU RNN:
Epoch 1/20, Train Loss: 0.7022546846293019, Dev Loss: 0.5867635689881212
Epoch 2/20, Train Loss: 0.5667467373823074, Dev Loss: 0.5607323512472305
Epoch 3/20, Train Loss: 0.5355007804888793, Dev Loss: 0.5470747769924632
Epoch 4/20, Train Loss: 0.5163764904924496, Dev Loss: 0.5313968856009027
Epoch 5/20, Train Loss: 0.4987494789519525, Dev Loss: 0.5240710062117206
Epoch 6/20, Train Loss: 0.4859537860313045, Dev Loss: 0.5268476989334986
Epoch 7/20, Train Loss: 0.47321452441555695, Dev Loss: 0.5183539290475965
Epoch 8/20, Train Loss: 0.46294248138149996, Dev Loss: 0.5155142432540879
Epoch 9/20, Train Loss: 0.45372238668249004, Dev Loss: 0.5186465924843809
Epoch 10/20, Train Loss: 0.4451829866587269, Dev Loss: 0.5136722904772388
Epoch 11/20, Train Loss: 0.4223506854107308, Dev Loss: 0.5125919201021505
Epoch 12/20, Train Loss: 0.4175819735985578, Dev Loss: 0.5130881210018817
Epoch 13/20, Train Loss: 0.4148119417116405, Dev Loss: 0.5125126642242709
Epoch 14/20, Train Loss: 0.4130115674381789, Dev Loss: 0.5124499859442389
Epoch 15/20, Train Loss: 0.4112976084637598, Dev Loss: 0.5140063311895332
Epoch 16/20, Train Loss: 0.41111166755898343, Dev Loss: 0.513076528869476
Epoch 17/20, Train Loss: 0.4085400805171807, Dev Loss: 0.5147684721420881
Epoch 18/20, Train Loss: 0.4067330705513919, Dev Loss: 0.5148656620716391
Epoch 19/20, Train Loss: 0.40543398701478256, Dev Loss: 0.5143077934072131
Epoch 20/20, Train Loss: 0.40476068490546674, Dev Loss: 0.5151277575875285
```



```
Classification Report for DeepPOS_BiGRU:
              precision    recall  f1-score   support

         ADJ       0.91      0.91      0.91      1794
         ADP       0.84      0.76      0.80      2030
         ADV       0.94      0.85      0.89      1183
         AUX       0.97      0.96      0.97      1543
       CCONJ       0.83      0.31      0.45       736
         DET       0.89      0.78      0.83      1896
        INTJ       0.93      0.69      0.80       121
        NOUN       0.91      0.90      0.90      4123
         NUM       0.66      0.49      0.56       542
        PART       0.68      0.61      0.65       649
        PRON       0.96      0.97      0.97      2166
       PROPN       0.91      0.81      0.86      2076
       PUNCT       0.58      0.91      0.71      3096
       SCONJ       0.80      0.70      0.74       384
         SYM       0.96      0.61      0.74       109
        VERB       0.93      0.92      0.92      2606
           X       0.00      0.00      0.00        42
           _       0.96      0.78      0.86       354

    accuracy                           0.84     25450
   macro avg       0.81      0.72      0.75     25450
weighted avg       0.86      0.84      0.84     25450


AUC Scores for Each Class:
ADJ: 0.9901
ADP: 0.9810
ADV: 0.9948
AUX: 0.9988
CCONJ: 0.9573
DET: 0.9891
INTJ: 0.9802
NOUN: 0.9887
NUM: 0.9760
PART: 0.9829
PRON: 0.9994
PROPN: 0.9807
PUNCT: 0.9663
SCONJ: 0.9905
SYM: 0.9739
VERB: 0.9949
X: 0.9072
_: 0.9875

Macro-Averaged Precision: 0.8144968015657622
Macro-Averaged Recall: 0.719536159674343
Macro-Averaged F1: 0.7530539398151068
Macro-Averaged Precision-Recall AUC: 0.9799530944194375
```

In [ ]:
```
print("Training Very Deep BiGRU RNN:")
very_deep_bigrun_model = train_and_evaluate_rnn(VeryDeepPOS_BiGRU, input_dim=2*word
```

```
Training Very Deep BiGRU RNN:
Epoch 1/20, Train Loss: 0.7415308893886593, Dev Loss: 0.6143699082216822
Epoch 2/20, Train Loss: 0.5946611025962585, Dev Loss: 0.5762313990888739
Epoch 3/20, Train Loss: 0.5588858117699771, Dev Loss: 0.5607933800919611
Epoch 4/20, Train Loss: 0.5382228776389173, Dev Loss: 0.5455909664544246
Epoch 5/20, Train Loss: 0.5208595192831854, Dev Loss: 0.5391640282588495
Epoch 6/20, Train Loss: 0.5071301893863213, Dev Loss: 0.5371348662558654
Epoch 7/20, Train Loss: 0.4958442723432612, Dev Loss: 0.525490907088557
Epoch 8/20, Train Loss: 0.4843710587372045, Dev Loss: 0.5296513514411181
Epoch 9/20, Train Loss: 0.4747367534254936, Dev Loss: 0.5240084873255632
Epoch 10/20, Train Loss: 0.46566733328592197, Dev Loss: 0.5202993790159249
Epoch 11/20, Train Loss: 0.4445159603381982, Dev Loss: 0.513571013782855
Epoch 12/20, Train Loss: 0.4370920961473075, Dev Loss: 0.5127045885662088
Epoch 13/20, Train Loss: 0.4349677289771106, Dev Loss: 0.5127411887311099
Epoch 14/20, Train Loss: 0.432260421029372, Dev Loss: 0.5131513121582213
Epoch 15/20, Train Loss: 0.42951581396802185, Dev Loss: 0.5139819231621903
Epoch 16/20, Train Loss: 0.4280960612146298, Dev Loss: 0.5127779939270258
Epoch 17/20, Train Loss: 0.42704468451721056, Dev Loss: 0.5134377005629074
Epoch 18/20, Train Loss: 0.425876667353752, Dev Loss: 0.514354779494735
Epoch 19/20, Train Loss: 0.42580834123144334, Dev Loss: 0.5128783027181649
Epoch 20/20, Train Loss: 0.4208277455491369, Dev Loss: 0.5134728133379666
```



```
Classification Report for VeryDeepPOS_BiGRU:
              precision    recall  f1-score   support

         ADJ       0.91      0.91      0.91      1794
         ADP       0.84      0.75      0.79      2030
         ADV       0.94      0.86      0.90      1183
         AUX       0.96      0.97      0.96      1543
       CCONJ       0.71      0.34      0.46       736
         DET       0.89      0.79      0.83      1896
        INTJ       0.93      0.76      0.84       121
        NOUN       0.92      0.89      0.90      4123
         NUM       0.66      0.50      0.57       542
        PART       0.65      0.63      0.64       649
        PRON       0.96      0.98      0.97      2166
       PROPN       0.89      0.81      0.85      2076
       PUNCT       0.59      0.89      0.71      3096
       SCONJ       0.80      0.69      0.74       384
         SYM       0.93      0.61      0.73       109
        VERB       0.92      0.92      0.92      2606
           X       0.14      0.02      0.04        42
           _       0.95      0.78      0.86       354

    accuracy                           0.84     25450
   macro avg       0.81      0.73      0.76     25450
weighted avg       0.85      0.84      0.84     25450


AUC Scores for Each Class:
ADJ: 0.9900
ADP: 0.9806
ADV: 0.9944
AUX: 0.9988
CCONJ: 0.9574
DET: 0.9889
INTJ: 0.9820
NOUN: 0.9883
NUM: 0.9756
PART: 0.9828
PRON: 0.9994
PROPN: 0.9802
PUNCT: 0.9663
SCONJ: 0.9884
SYM: 0.9711
VERB: 0.9948
X: 0.8996
_: 0.9865

Macro-Averaged Precision: 0.8111026383944473
Macro-Averaged Recall: 0.7271724473564507
Macro-Averaged F1: 0.7571378802757028
Macro-Averaged Precision-Recall AUC: 0.9791739557097381
```

Choice of Architectures for Shallow, Somewhat Deep, and Deep RNNs

The design of the three RNN models (shallow, somewhat deep, and deep) was driven by the goal of maintaining a similar structure to the original MLP models while utilizing recurrent layers (GRU or LSTM) to capture temporal dependencies. The architectures were chosen based on the depth and complexity of the original MLP models, with the following considerations:

---

1. **Shallow Bidirectional RNN (ShallowPOS_BiGRU)**

- **Depth**: The original shallow MLP contained only a single hidden layer. To maintain a similar structure in the RNN, we opted for **one bidirectional RNN layer**.
- **Bidirectional Nature**: The bidirectional RNN (using GRU or LSTM) processes the sequence in both directions (forward and backward), capturing context from both past and future words. This enriches the representation without increasing the depth of the model.
- **Output Layer**: Since the task is POS tagging, we use the final hidden state of the sequence for classification. We take the output of the last timestep from the bidirectional RNN as the representation of the input sequence, which is passed through a final linear layer for POS tagging.
- **Regularization**: Dropout is included after the RNN layer to prevent overfitting, keeping the model simple and regularized.

---

2. **Somewhat Deep Bidirectional RNN (DeepPOS_BiGRU)**

- **Depth**: The original somewhat deep MLP contained two hidden layers, and we retained the concept of depth by adding **two bidirectional RNN layers**.
- **Bidirectional Nature**: Just like the shallow model, the bidirectional RNN layers help capture both past and future context. With two layers, the model can learn richer representations of the input sequence.
- **Regularization**: Dropout is applied after each RNN layer, just like the original MLP's design, to prevent overfitting while maintaining simplicity.
- **Output Layer**: Similar to the shallow model, we take the final hidden state from the last timestep after the second RNN layer as the representation for POS tagging.

---

3. **Very Deep Bidirectional RNN (VeryDeepPOS_BiGRU)**

- **Depth**: The original very deep MLP had **three hidden layers**. To replicate this depth in the RNN model, we used **three bidirectional RNN layers**.
- **Bidirectional Nature**: With three bidirectional RNN layers, the model learns contextual information from both the past and the future at multiple levels of abstraction.
- **Regularization**: Dropout is applied after each RNN layer to reduce the risk of overfitting and ensure that the model generalizes well.

- **Output Layer**: As in the shallower models, the last timestep of the final RNN layer is used for classification, which provides the most relevant representation of the entire sequence.

Each RNN architecture was designed to match the depth and structure of the original MLP models while incorporating the ability of recurrent layers to capture sequential dependencies. The bidirectional nature of the RNNs enhances the model's performance by considering both past and future contexts for each input sequence.

Model	Accuracy	Macro avg F1-score	Weighted avg F1-score	Strengths	Weaknesses
ShallowPOS_MLP	0.83	0.74	0.83	High performance on **PRON (0.97 precision)**, **AUX (0.96 recall)**, and **VERB (0.91 f1-score)**.	Struggles with **X (f1 = 0.00)** and **CCONJ (0.45 f1)**.
DeepPOS_MLP	0.83	0.74	0.83	Improved recall for **AUX (0.94)** and **PRON (0.97)**, maintains strong performance across categories.	Struggles with rare categories like **X**.
VeryDeepPOS_MLP	0.82	0.74	0.82	Strong performance on **AUX** and **PRON**, but slightly lower overall performance compared to other MLP models.	Some decrease in performance, especially in **CCONJ (0.43 f1)** and **NUM (0.56 f1)**.
ShallowPOS_BiGRU	0.84	0.75	0.84	High performance on **PRON (0.97 precision)**, **AUX (0.97 recall)**, and **VERB (0.92 f1-score)**.	Struggles with **X (f1 = 0.04)** and **CCONJ (0.45 f1)**.
DeepPOS_BiGRU	0.84	0.75	0.84	High performance on **AUX (0.97)** and **PRON (0.97)**.	Struggles with **X** and **CCONJ (0.45 f1)**.

Model	Accuracy	Macro avg F1-score	Weighted avg F1-score	Strengths	Weaknesses
VeryDeepPOS_BiGRU	0.84	0.76	0.84	Strong performance on **AUX (0.97)**, **PRON (0.98)**, and **VERB (0.92)** with a good balance across categories.	Some degradation in performance for **CCONJ (0.46 f1)** and **NUM (0.57 f1)**, but more consistent than previous models.

Comparison:

- **Accuracy:** Both the MLP and BiGRU models generally perform similarly in terms of accuracy (ranging from **0.82 to 0.84**).
- **F1 Scores:** The BiGRU models tend to outperform the MLP models slightly in terms of both macro and weighted average F1 scores, with values like **0.75 (macro avg)** and **0.84 (weighted avg)** compared to the MLP's **0.74 (macro avg)** and **0.83 (weighted avg)**.
- **Category Performance:** Both MLP and RNN models show consistent strength in categories like **PRON** and **AUX**, but also share similar weaknesses in rare categories like **X** and **CCONJ**.
- **Deepness & Performance Trade-off:** Moving from shallow to deep MLP models generally resulted in small drops in accuracy and macro F1, suggesting diminishing returns with deeper layers. However, the BiGRU models show slight improvements in their macro and weighted F1 scores despite being deep.

Baseline Tagger

In [ ]:

```
Baseline Tagger Classification Report:
              precision    recall  f1-score   support

         ADJ       0.91      0.83      0.87      1794
         ADP       0.87      0.88      0.88      2030
         ADV       0.94      0.79      0.86      1183
         AUX       0.93      0.89      0.91      1543
       CCONJ       0.99      1.00      0.99       736
         DET       0.96      0.97      0.96      1896
        INTJ       0.97      0.69      0.80       121
        NOUN       0.67      0.93      0.78      4123
         NUM       0.91      0.61      0.73       542
        PART       0.69      0.99      0.81       649
        PRON       0.96      0.93      0.95      2166
       PROPN       0.91      0.51      0.66      2076
       PUNCT       0.99      0.99      0.99      3096
       SCONJ       0.62      0.60      0.61       384
         SYM       0.81      0.83      0.82       109
        VERB       0.89      0.82      0.85      2606
           X       1.00      0.00      0.00        42
           _       0.97      0.81      0.89       354

    accuracy                           0.86     25450
   macro avg       0.89      0.78      0.80     25450
weighted avg       0.88      0.86      0.86     25450


AUC Scores for Each Class:
ADJ: 0.9111
ADP: 0.9350
ADV: 0.8924
AUX: 0.9422
CCONJ: 0.9985
DET: 0.9823
INTJ: 0.8429
NOUN: 0.9219
NUM: 0.8029
PART: 0.9902
PRON: 0.9629
PROPN: 0.7539
PUNCT: 0.9927
SCONJ: 0.7992
SYM: 0.9170
VERB: 0.9017
X: 0.5000
_: 0.9066

Macro-Averaged Precision: 0.8891
Macro-Averaged Recall: 0.7814
Macro-Averaged F1: 0.7974
Macro-Averaged Precision-Recall AUC: 0.8863
```

Dataset Statistics

```
In [ ]:
```

```
Training Data Statistics:
Total number of words: 207230
Vocabulary size: 20201
Average word length: 4.08 characters
Number of unique POS tags: 18
Most frequent POS tag: NOUN (occurred 34755 times)

POS tag distribution:
ADJ: 13187 occurrences
ADP: 17745 occurrences
ADV: 10117 occurrences
AUX: 12818 occurrences
CCONJ: 6687 occurrences
DET: 16299 occurrences
INTJ: 695 occurrences
NOUN: 34755 occurrences
NUM: 4127 occurrences
PART: 5748 occurrences
PRON: 18677 occurrences
PROPN: 12618 occurrences
PUNCT: 23596 occurrences
SCONJ: 3822 occurrences
SYM: 722 occurrences
VERB: 22604 occurrences
X: 399 occurrences
_: 2614 occurrences

Development Data Statistics:
Total number of words: 25512
Vocabulary size: 5638
Average word length: 4.14 characters
Number of unique POS tags: 18
Most frequent POS tag: NOUN (occurred 4212 times)

POS tag distribution:
ADJ: 1873 occurrences
ADP: 2039 occurrences
ADV: 1224 occurrences
AUX: 1567 occurrences
CCONJ: 779 occurrences
DET: 1900 occurrences
INTJ: 115 occurrences
NOUN: 4212 occurrences
NUM: 383 occurrences
PART: 647 occurrences
PRON: 2225 occurrences
PROPN: 1865 occurrences
PUNCT: 3075 occurrences
SCONJ: 397 occurrences
SYM: 83 occurrences
VERB: 2710 occurrences
X: 59 occurrences
_: 359 occurrences

Test Data Statistics:
Total number of words: 25450
```

```
Vocabulary size: 5750
Average word length: 4.13 characters
Number of unique POS tags: 18
Most frequent POS tag: NOUN (occurred 4123 times)

POS tag distribution:
ADJ: 1794 occurrences
ADP: 2030 occurrences
ADV: 1183 occurrences
AUX: 1543 occurrences
CCONJ: 736 occurrences
DET: 1896 occurrences
INTJ: 121 occurrences
NOUN: 4123 occurrences
NUM: 542 occurrences
PART: 649 occurrences
PRON: 2166 occurrences
PROPN: 2076 occurrences
PUNCT: 3096 occurrences
SCONJ: 384 occurrences
SYM: 109 occurrences
VERB: 2606 occurrences
X: 42 occurrences
_: 354 occurrences
```

Methods and Datasets

We developed a **Part-of-Speech (POS) tagger** using a variety of **Recurrent Neural Network (RNN)** architectures, using either **Bidirectional GRU** or **LSTM** models. In all cases, **Word2Vec embeddings** were used as input features. The model was evaluated on the **English Universal Dependencies Treebank** (UD_English-EWT), which contains labeled data for training, development, and testing.

The models were evaluated on several performance metrics such as **accuracy**, **precision**, **recall**, **F1-score**, and **AUC scores**.

Datasets:

- **Training**: en_ewt-ud-train.conllu
- **Development**: en_ewt-ud-dev.conllu
- **Test**: en_ewt-ud-test.conllu

The data was parsed and preprocessed to extract **words** and their corresponding **POS tags**.

Preprocessing Steps

1. **Tokenization**:
    - We parsed the **conllu** files to extract each word and its associated POS tag.
2. **Word2Vec Embeddings**:

- Pre-trained **Word2Vec** embeddings were used to represent words as vectors. This allows us to capture semantic relationships between words.

3. **Model**:

    - We trained various models, including:
        - **Bidirectional RNN with GRU/LSTM cells**: These architectures were developed to capture sequential dependencies in the data. The models had variety in their depth. Bidirectional GRU/LSTM cells were used to capture both past and future context in the sentence. Dropout was applied as a regularization technique.
        - **Comparison of Performance**: We compared different model architectures—**Shallow RNN**, **Deep RNN**, and **Very Deep RNN** with the previous best MLP performer.

4. **Evaluation**:

    - The performance of the models was evaluated on **accuracy**, **precision**, **recall**, **F1-score**, and **AUC scores**, with metrics computed separately for each POS tag and averaged across tags.