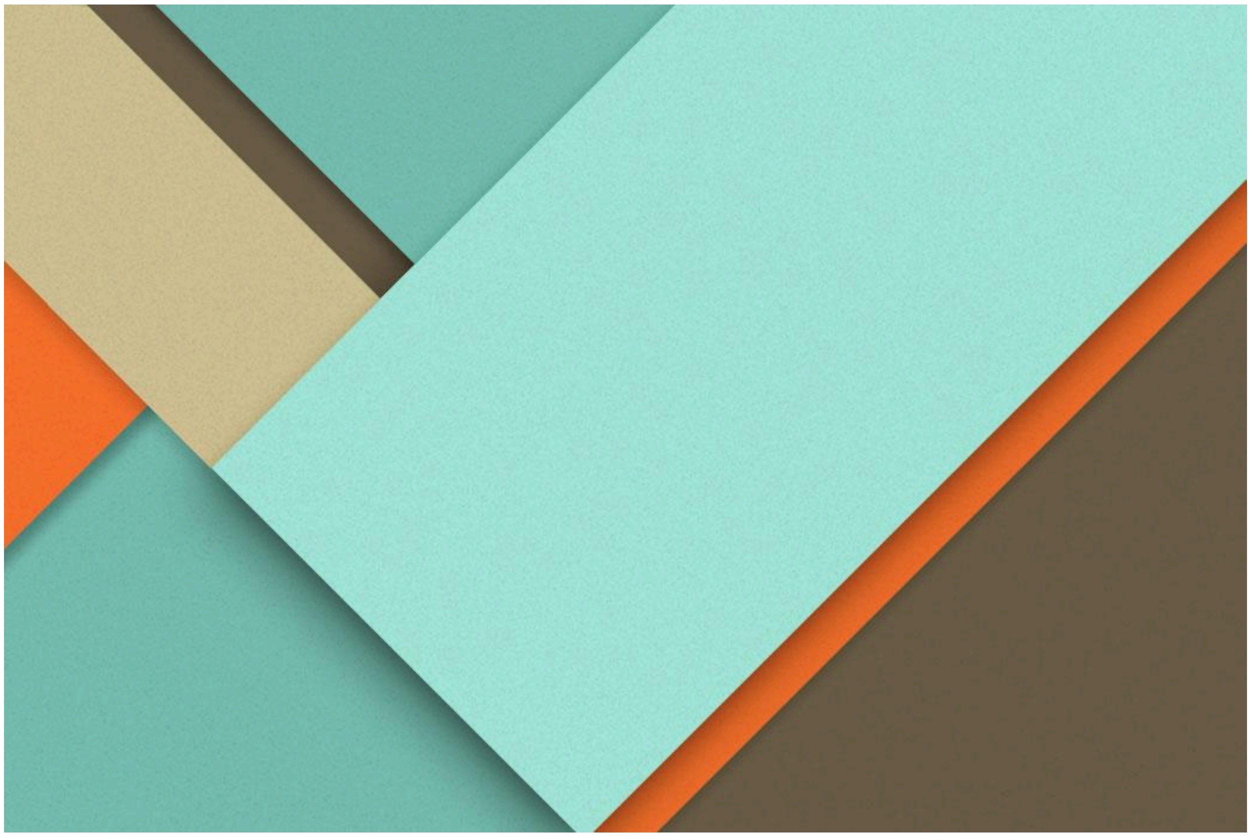


Assignment-4 Report

Elmo.py

Building an ELMo architecture and testing its efficacy on the same downstream task. ELMo looks at building a layered representation of a word through stacked Bi-LSTM layers,separately weighing in syntactic and semantic representations.

Hyperparameters Used:

1. Learning Rate (LR): 0.001 (Set for both forward and backward models)

○ Optimizer: Adam

2. Batch Size: 50

3. Number of Epochs:5

2 models forward.py and backward.py are created from elmo.py as LMs

Downstream.py

Hyperparametrs Used:

1. Learning Rate (LR): 0.001

○ Optimizer: Adam

2. Batch Size: 32 (both for training and testing data)

3. Number of Epochs: 5

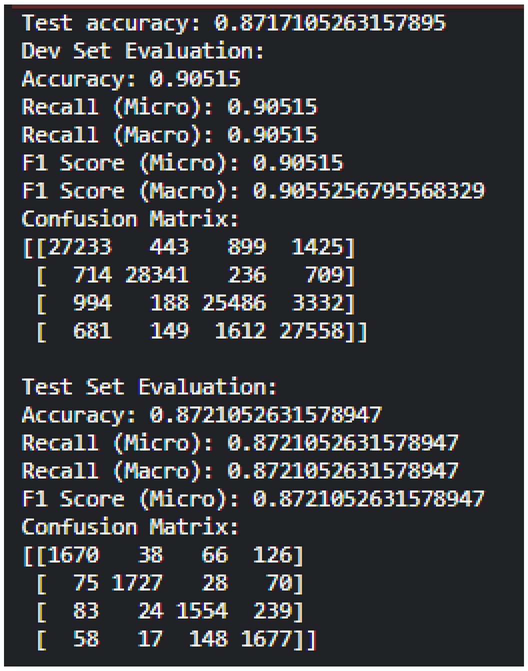
Trainable Parameters (Lambdas)

In our model architecture, we've integrated three scalar parameters known as l1, l2, and l3. These parameters act as coefficients for blending the outputs from different LSTM layers before passing them to the downstream classifier. Through the training process, these parameters undergo iterative adjustments via backpropagation, allowing the model to fine-tune their values to enhance performance.

For Elmo when lambda's are untrained we get a little higher accuracy(that can be due to choosing a mediocre value).More epoch can give a better performance as only on 5 epoch we get an accuracy of 0.88 on test with lamda=0.33,but accuracy gets better on trainable lambda for larger epochs as it get accustom to the lstm network and predicts better outputs

Usage:

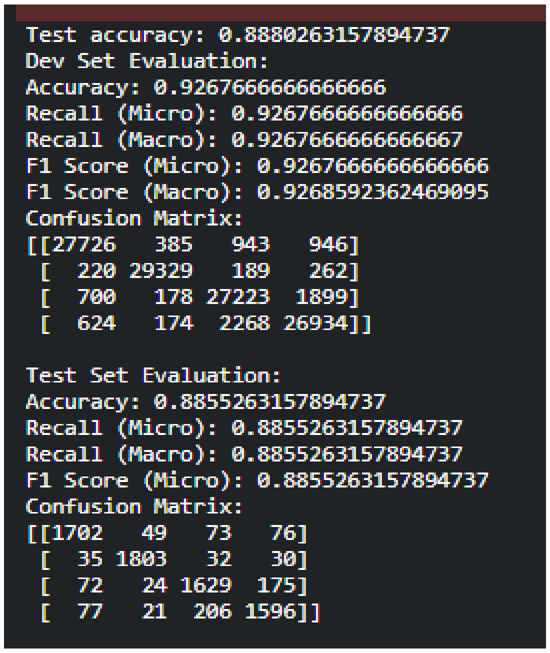
1. **Training Phase:**
   - Initially, the parameters l1, l2, and l3 are trainable, implying that their values are updated during each training iteration.

- This trainable nature empowers the model to dynamically adapt the contribution of each LSTM layer's output according to the input data and learning objectives.
- 

```
Test accuracy: 0.8717105263157895
Dev Set Evaluation:
Accuracy: 0.90515
Recall (Micro): 0.90515
Recall (Macro): 0.90515
F1 Score (Micro): 0.90515
F1 Score (Macro): 0.9055256795568329
Confusion Matrix:
[[27233   443   899  1425]
 [  714 28341   236   709]
 [  994   188 25486  3332]
 [  681   149  1612 27558]]

Test Set Evaluation:
Accuracy: 0.8721052631578947
Recall (Micro): 0.8721052631578947
Recall (Macro): 0.8721052631578947
F1 Score (Micro): 0.8721052631578947
Confusion Matrix:
[[1670   38   66  126]
 [  75 1727   28   70]
 [  83   24 1554  239]
 [  58   17  148 1677]]
```

2. **Freezing:**
   - After a certain number of training epochs or at an appropriate juncture, we have the option to freeze these parameters.
   - Freezing halts further updates to the values of l1, l2, and l3 during subsequent training iterations.
   - This freezing option proves advantageous when we need to refine other aspects of the model without affecting the learned blending coefficients.
   - By freezing these parameters, we ensure the retention of the learned contributions from each LSTM layer, fostering stability in the model's behavior and enabling more targeted optimization in other components.



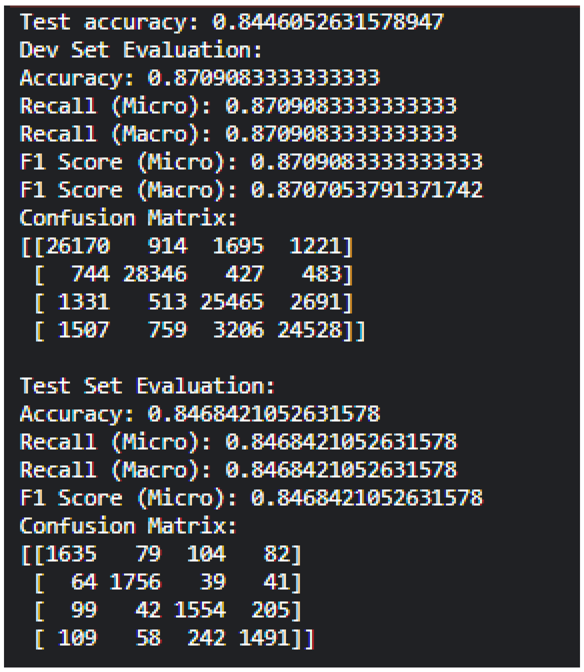
```
Test accuracy: 0.8880263157894737
Dev Set Evaluation:
Accuracy: 0.9267666666666666
Recall (Micro): 0.9267666666666666
Recall (Macro): 0.9267666666666667
F1 Score (Micro): 0.9267666666666666
F1 Score (Macro): 0.9268592362469095
Confusion Matrix:
[[27726   385   943   946]
 [  220 29329   189   262]
 [  700   178 27223  1899]
 [  624   174  2268 26934]]

Test Set Evaluation:
Accuracy: 0.8855263157894737
Recall (Micro): 0.8855263157894737
Recall (Macro): 0.8855263157894737
F1 Score (Micro): 0.8855263157894737
Confusion Matrix:
[[1702   49   73   76]
 [  35 1803   32   30]
 [  72   24 1629  175]
 [  77   21  206 1596]]
```

3. **Learnable Function:**

Learning a function to combine the word representations across layers to build the final contextual word embedding.

ˆE= f(e0, e1, e2)



```
Test accuracy: 0.8446052631578947
Dev Set Evaluation:
Accuracy: 0.8709083333333333
Recall (Micro): 0.8709083333333333
Recall (Macro): 0.8709083333333333
F1 Score (Micro): 0.8709083333333333
F1 Score (Macro): 0.8707053791371742
Confusion Matrix:
[[26170   914  1695  1221]
 [  744 28346   427   483]
 [ 1331   513 25465  2691]
 [ 1507   759  3206 24528]]

Test Set Evaluation:
Accuracy: 0.8468421052631578
Recall (Micro): 0.8468421052631578
Recall (Macro): 0.8468421052631578
F1 Score (Micro): 0.8468421052631578
Confusion Matrix:
[[1635   79  104   82]
 [  64 1756   39   41]
 [  99   42 1554  205]
 [ 109   58  242 1491]]
```

As we can see here ,we are not getting good accuracy as compared to other 2 so introduction of lambda helps in increasing accuracy

Comparison and Analysis of Word Vectorization Methods

In this report, we compare three word vectorization methods: Word2Vec, Singular Value Decomposition (SVD), and Embeddings from Language Models (ELMo). We evaluate their performance using various performance metrics such as accuracy, F1 score, precision, recall, and confusion matrix on both the train and test sets.

	Accuracy on test	Recall(micro)	F1 score	Recall(macro	
SVD	0.85	0.85	0.85	0.85	
Word2vec	0.87	0.87	0.87	0.87	
Elmo	0.88	0.88	0.88	0.88	

	Accuracy on train	Recall(micro)	F1 score	Recall(macro	
SVD	0.88	0.88	0.88	0.88	
Word2vec	0.90	0.90	0.90	0.90	
Elmo	0.92	0.92	0.92	0.92	

Analysis:

The performance of each method depends on various factors such as dataset size, complexity of the task, and available computational resources.

Word2Vec tends to perform well when dealing with large datasets and capturing semantic similarities between words. However, it may struggle with out-of-vocabulary words and context-dependent meanings.

SVD is effective for reducing the dimensionality of sparse matrices and capturing global patterns in the data. It may not capture semantic nuances as effectively as Word2Vec or ELMo.

ELMo excels in capturing contextual information and handling polysemy, making it suitable for tasks requiring fine-grained semantic understanding. However, ELMo embeddings are computationally intensive and may require substantial computational resources for training and inference.