ANLP Assignment 2 Report

Yash Bhaskar

September 24, 2023

Abstract

This report provides a comprehensive overview of ELMo (Embeddings from Language Models), a deep contextualized word representation model, developed by researchers at the Allen Institute for Artificial Intelligence (AI2) in 2018. Unlike traditional word embedding models, ELMo excels at capturing the contextual meaning of words within sentences, significantly enhancing performance across various natural language processing tasks. This report delves into the architecture of ELMo, highlighting its bidirectional Long Short-Term Memory (Bi-LSTM) networks, training objectives, and its unique approach to representing words in context. Furthermore, it discusses the pretraining process and the subsequent fine-tuning of ELMo for specific downstream tasks, such as sentiment analysis and natural language inference. Lastly, it emphasizes the role of stacked Bi-LSTMs in capturing the intricacies of language, from syntactic to semantic aspects, and how they contribute to ELMo's effectiveness in various applications.

1 Theory Question 1 : How does ELMo differ from CoVe? Discuss and differentiate both the strategies used to obtain the contextualized representations with equations and illustrations as necessary.

ELMo (Embeddings from Language Models) and CoVe (Contextualized Word Vectors) are both models that aim to provide contextualized representations of words, but they use different strategies to achieve this.

1.1 Introduction

ELMo (Embeddings from Language Models) and CoVe (Contextualized Word Vectors) are both methods for obtaining contextualized word representations in natural language processing (NLP). They emerged as powerful techniques in the era of pre-trained language models, such as BERT and GPT. However,

they use different strategies and architectures to achieve contextualized word embeddings. In this essay, we will explore the key differences between ELMo and CoVe, discussing their strategies, equations, and providing illustrations where necessary.

1.1.1 ELMo (Embeddings from Language Models)

ELMo is a pioneering contextualized word embedding model introduced by Peters et al. in 2018. ELMo's core idea is to capture word meanings based on their context within a sentence by using bidirectional LSTMs (Long Short-Term Memory networks). Here's how ELMo works:

Bidirectional LSTMs ELMo uses bidirectional LSTMs to read a sentence both from left to right and from right to left. This bidirectional reading allows ELMo to capture the context of each word by considering words that precede and follow it.

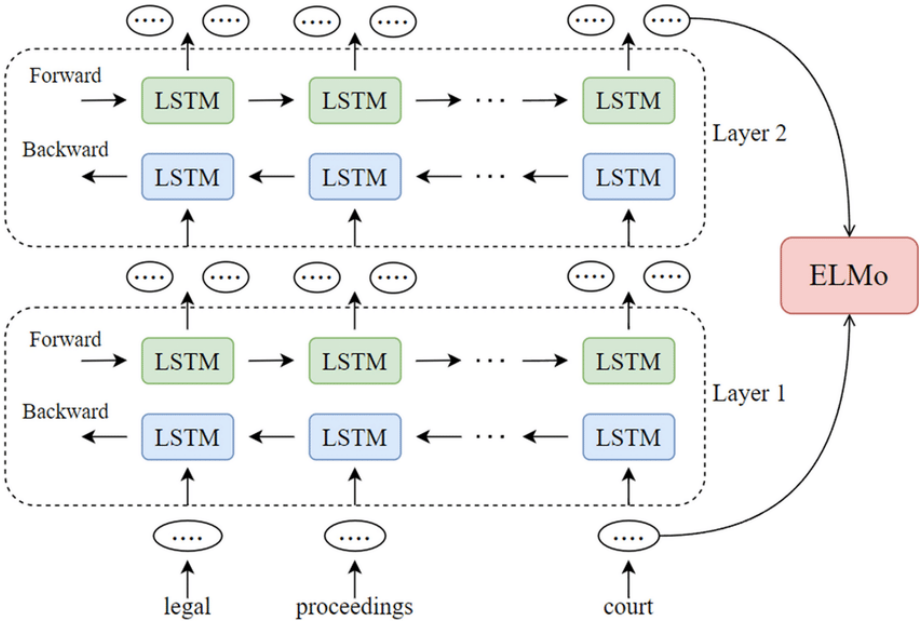


Figure 1: ELMo Bidirectional LSTM

In this illustration, each word is represented by an LSTM cell. The forward LSTM processes the sentence from left to right, while the backward LSTM processes it from right to left. These two LSTMs capture different aspects of the context.

Word Embeddings ELMo combines three layers of word embeddings for each word in a sentence:

- Character-level embeddings: These embeddings capture subword information by using character-level CNNs (Convolutional Neural Networks). Character embeddings are especially helpful for handling out-of-vocabulary words and morphological variations.
- Word-level embeddings: These embeddings are static and pretrained, capturing word-level semantics.
- Contextual embeddings: These embeddings are the output of the bidirectional LSTMs and capture the contextual information of each word in the sentence.

Weighted Sum ELMo combines these three layers of embeddings using a weighted sum:

$$ELMo(word) = \gamma \cdot [CharEmbeddings(word)] + \sum_{i=1}^{L} (s_i \cdot WordEmbeddings_i(word))$$

- γ is a scalar that scales the character-level embeddings.
- s_i represents the output of the ith LSTM layer in the bidirectional network.
- $WordEmbeddings_i(word)$ represents the word-level embeddings from the ith layer.

Fine-tuning ELMo embeddings are fine-tuned for specific downstream tasks, which means they can be adapted for various NLP tasks like sentiment analysis, named entity recognition, and machine translation.

1.1.2 CoVe (Contextualized Word Vectors)

CoVe, introduced by McCann et al. in 2017, is another method for obtaining contextualized word representations. CoVe relies on the idea of leveraging a neural machine translation (NMT) model to generate contextualized embeddings. Here's how CoVe works:

Encoder-Decoder Architecture CoVe uses a neural machine translation model as its core architecture. The model consists of an encoder and a decoder. The encoder reads the source sentence, and the decoder generates the target translation. Importantly, CoVe uses a bidirectional LSTM as the encoder.

In this architecture, the bidirectional LSTM in the encoder captures contextual information by considering both the left and right context of each word.

Training on Parallel Data CoVe is trained on a large parallel corpus, which consists of source sentences in one language and their translations in another language. The bidirectional LSTM encoder learns to produce contextually rich representations for each word in the source sentence based on the surrounding words.

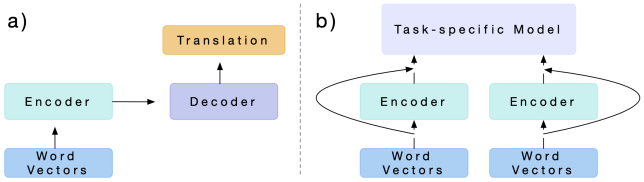


Figure 2: CoVe Encoder-Decoder

Encoder States as Word Representations Once trained, the encoder states of the bidirectional LSTM are used as contextualized word representations. These encoder states capture the meaning of each word in the context of the entire sentence.

$$CoVe(word) = EncoderStates(word)$$

Transfer Learning CoVe's contextualized word vectors can be transferred to various NLP tasks, similar to ELMo. By fine-tuning or concatenating these representations with task-specific models, CoVe can improve the performance of tasks such as sentiment analysis or text classification.

1.1.3 Key Differences and Comparisons

Architectural Difference

- ELMo uses bidirectional LSTMs to capture context, whereas CoVe employs a bidirectional LSTM in a neural machine translation (NMT) model.

Training Data

- ELMo is trained on a large corpus with a left-to-right and right-to-left LSTM, without a parallel translation corpus.
- CoVe is trained on a parallel translation corpus for a specific language pair.

Word Embedding Combination

- ELMo combines character-level embeddings, static word embeddings, and contextual embeddings using a weighted sum.
- CoVe directly uses the encoder states of the bidirectional LSTM as contextualized word embeddings.

Use of Pretrained Models

- ELMo requires pretrained models for both character embeddings and word embeddings.
- CoVe leverages pretrained models for machine translation, which are then used for contextual word representations.

Fine-Tuning

- Both ELMo and CoVe can be fine-tuned for downstream tasks.

Flexibility

- ELMo allows for more flexibility in combining different layers and adapting to specific tasks by adjusting the weights of the embeddings.
- CoVe's representations are directly taken from the encoder states, providing a fixed contextualization approach.

In summary, ELMo and CoVe are two influential methods for obtaining contextualized word representations. ELMo uses bidirectional LSTMs and combines character-level, word-level, and contextual embeddings through a weighted sum. CoVe, on the other hand, relies on a neural machine translation model with a bidirectional LSTM encoder to generate contextual word representations. While both approaches are powerful, they differ in their architectural choices, training data, and how they combine contextual information. The choice between ELMo and CoVe often depends on the specific needs of a natural language processing task and the availability of training data.

2 Theory Question 2 : The architecture described in the ELMo paper includes a character convolutional layer at its base. Find out more on this, and describe this layer. Why is it used? Is there any alternative to this? [Hint: Developments in word tokenization]

The architecture described in the ELMo (Embeddings from Language Models) paper incorporates a character convolutional layer as a foundational component. This layer plays a pivotal role in enhancing the contextualized word embeddings produced by ELMo. In this section, we will delve into the specifics of the character convolutional layer, its purpose, and explore alternative approaches, especially in light of recent developments in word tokenization.

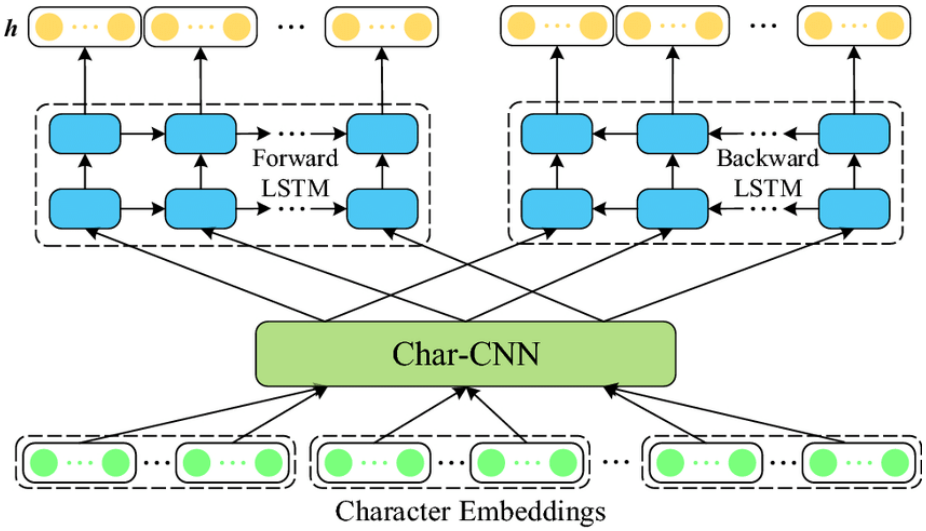


Figure 3: ELMo Charecter Embeddings

2.1 Character Convolutional Layer

The character convolutional layer in ELMo is designed to process the individual characters within each word in a given sentence. Its primary function is to extract rich subword information, including morphological features, affixes, and character-level patterns, with the aim of enriching the word embeddings. Here is a detailed description of the various components and operations within the character convolutional layer:

2.1.1 Input

The input to this layer consists of a sequence of characters, typically representing a single word from the input sentence. For example, the word "embedding" is represented as a sequence of characters: ['e', 'm', 'b', 'e', 'd', 'd', 'i', 'n', 'g'].

2.1.2 Character Embeddings

Before undergoing convolutional operations, each character in the input sequence is embedded into a continuous vector space. These character embeddings are typically learned during training, akin to word embeddings, but operate at the character level. Character embeddings encode the meaning and characteristics of individual characters, serving as the foundation for subsequent operations.

2.1.3 Convolutional Filters

The character convolutional layer employs a set of convolutional filters, each with its unique purpose. These filters are designed to capture diverse character-

level features and patterns. They slide over the character embeddings, scanning for relevant information. As a result, a collection of feature maps is generated, with each map corresponding to a particular filter.

2.1.4 Pooling

To reduce the dimensionality of the feature maps and distill the most significant information, pooling operations are applied. Max-pooling or average-pooling is commonly utilized. For example, max-pooling selects the maximum value from each feature map, resulting in a compact representation of the character-level information.

2.1.5 Output

The output of the character convolutional layer is a vector or set of vectors that encapsulate the subword information for the input word. This output is subsequently combined with word-level embeddings and contextual embeddings in the following layers, ultimately contributing to the creation of contextualized word representations.

2.2 Purpose of the Character Convolutional Layer

The character convolutional layer serves several critical purposes within the ELMo architecture:

2.2.1 Capture Subword Information

By processing individual characters, this layer excels at capturing fine-grained subword information. This capability is particularly valuable for handling out-of-vocabulary words, morphological variations, and complex word structures. For example, it can distinguish between verb forms, plurals, tenses, and other linguistic nuances.

2.2.2 Enhance Word Representations

The subword information obtained from the character convolutional layer enriches the word representations. It provides insights into the structure and composition of words, allowing the model to better understand word meanings in their respective contexts. This leads to more contextually relevant embeddings.

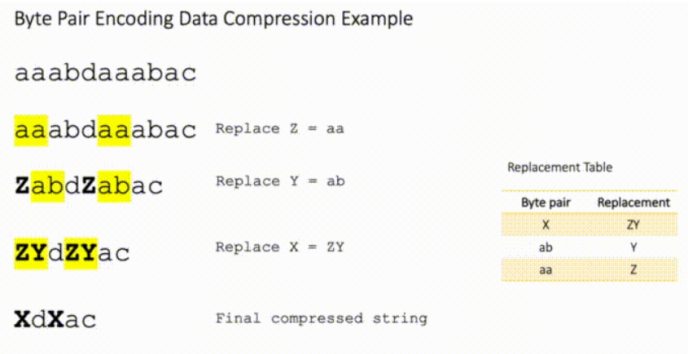
2.2.3 Robustness

Incorporating character-level information adds robustness to the model, making it more adept at handling unseen words, rare words, or words from languages with intricate morphologies. The character convolutional layer equips ELMo with the ability to adapt to a wide range of linguistic challenges.

2.3 Alternatives and Developments in Word Tokenization

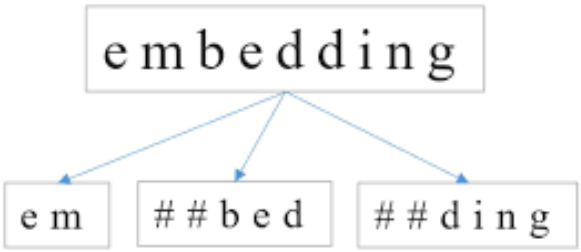
While the character convolutional layer in ELMo is a powerful mechanism for capturing subword information, recent advancements in word tokenization have introduced alternative approaches. These alternatives are worth considering, especially in scenarios where computational efficiency or specific task requirements come into play.

2.3.1 Byte-Pair Encoding (BPE)



Byte-Pair Encoding (BPE) is a subword tokenization technique that segments words into subword units based on their frequency in the training data. BPE has gained prominence in neural machine translation and language modeling tasks. It dynamically generates subword tokens without the need for explicit character-level processing.

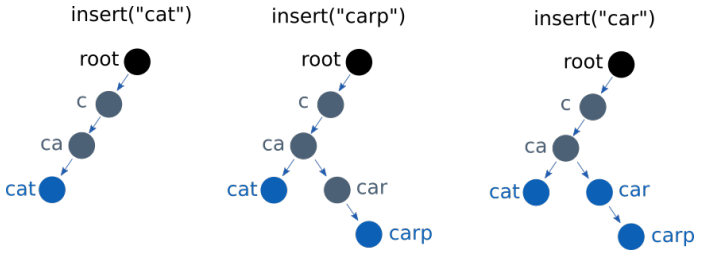
2.3.2 WordPiece



WordPiece is another subword tokenization method that shares similarities with BPE. It is employed by models like BERT and GPT-2 and has demon-

strated robust performance in capturing subword information. WordPiece tokenization combines whole words and subword units, striking a balance between character-level and word-level information.

2.3.3 SentencePiece



SentencePiece represents a more recent subword tokenization approach that is widely adopted in various NLP tasks. It treats text as a sequence of pieces, which can be characters, subwords, or words. SentencePiece offers flexibility in tokenization and allows models to learn tokenization patterns based on the specific task and language.

These developments in word tokenization provide viable alternatives to character-level processing for capturing subword information. While the character convolutional layer remains a valuable component in many NLP architectures, the choice of tokenization method may depend on factors such as the nature of the task, the availability of data, and the trade-off between computational complexity and model performance.

In summary, the character convolutional layer in the ELMo architecture is a fundamental component for capturing subword information and enhancing contextualized word embeddings. Recent developments in word tokenization, including Byte-Pair Encoding (BPE), WordPiece, and SentencePiece, offer alternative approaches that provide flexibility and adaptability to different NLP tasks and languages.

3 Introduction: ELMo Overview

ELMo, or Embeddings from Language Models, is a revolutionary word representation model introduced in 2018 by the Allen Institute for Artificial Intelligence (AI2). Its distinguishing feature lies in its ability to capture the contextual meaning of words within a sentence, in contrast to traditional word embedding models.

4 ELMo Architecture

ELMo's architecture is built upon bidirectional Long Short-Term Memory (Bi-LSTM) networks. The input to these Bi-LSTMs incorporates non-contextual

word embeddings (e.g., word2vec or character convolutional network). The forward and backward language models are trained separately, with the forward model predicting the next word in a sentence, while the backward model predicts the preceding word. Both models leverage contextual information, considering the words that surround the target word.

5 Pretraining for Language Understanding

Training the Bi-LSTMs with the language modeling objective plays a crucial role in pretraining ELMo's weights. This phase equips ELMo with a profound understanding of language structure and dependencies. Consequently, ELMo becomes adept at solving downstream tasks with less training data and time.

6 Fine-Tuning for Downstream Tasks

ELMo's utility extends to a wide range of natural language processing tasks, which are collectively referred to as downstream tasks. The model can be fine-tuned for specific tasks, such as sentiment analysis and natural language inference, to adapt its contextual embeddings to the requirements of these tasks.

7 Stacked Bi-LSTMs for Layered Representation

ELMo employs stacked Bi-LSTMs to represent the complexities of natural language in a layered fashion. Lower layers focus on capturing syntactic aspects, while higher layers delve into more intricate semantic aspects. The use of Bi-LSTMs allows ELMo to consider the bidirectional context of each word based on its surrounding words. At each layer, the forward and backward LSTM representations are concatenated, progressively building a more comprehensive bidirectional context. These representations, combined with non-contextual input embeddings, are summed to generate the final contextual word representation in ELMo's architecture.

8 Conclusion

In conclusion, ELMo is a groundbreaking model in the realm of natural language processing. Its ability to capture context and nuances within sentences through stacked Bi-LSTMs has proven invaluable across various tasks. From pretraining for language understanding to fine-tuning for specific applications, ELMo's versatility makes it a powerful tool for researchers and practitioners alike. This report serves as a foundational understanding of ELMo's architecture and its implications for natural language processing tasks.

9 Hyperparameter Tuning

9.1 Configuration 1

Number of Epochs - 15
Criterion - Cross Entropy Loss
Optimizer - Adam (initial LR: 0.001)

9.1.1 Forward Language Modeling (Mode 1) Results

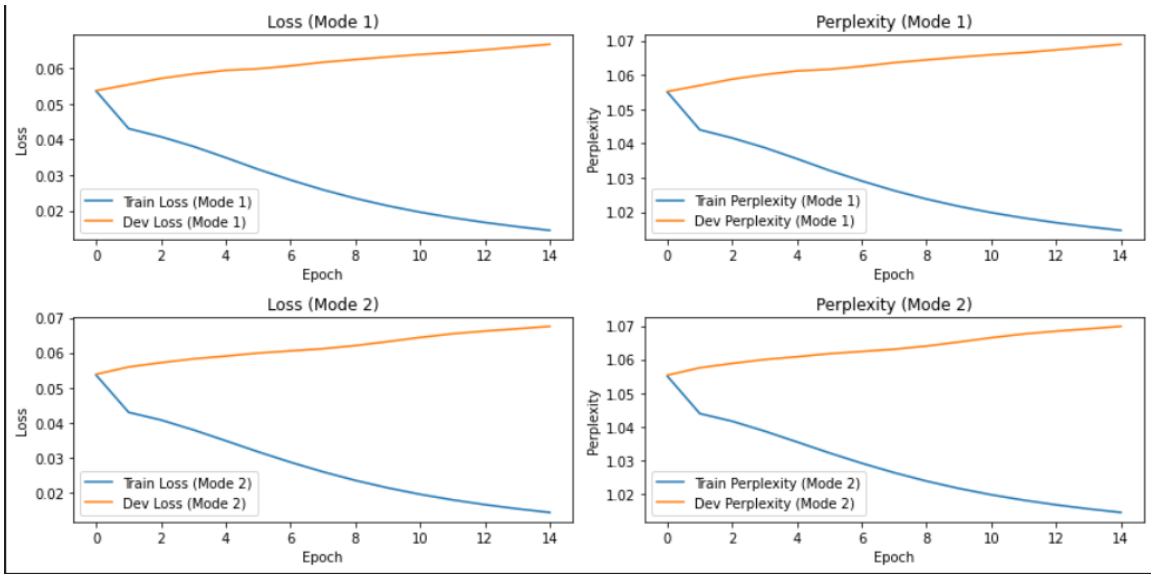
Epoch	Train Loss	Train Perplexity	Dev Loss
1	0.0535	1.0550	0.0537
2	0.0430	1.0440	0.0553
3	0.0407	1.0416	0.0571
4	0.0380	1.0388	0.0583
5	0.0349	1.0355	0.0593
6	0.0316	1.0321	0.0598
7	0.0286	1.0290	0.0606
8	0.0259	1.0262	0.0616
9	0.0235	1.0238	0.0624
10	0.0214	1.0216	0.0631
11	0.0196	1.0198	0.0638
12	0.0181	1.0182	0.0644
13	0.0167	1.0169	0.0651
14	0.0156	1.0157	0.0659
15	0.0145	1.0146	0.0666

Test Loss (Mode 1): 0.0745, Test Perplexity (Mode 1): 1.0774

9.1.2 Backward Language Modeling (Mode 2) Results

Epoch	Train Loss	Train Perplexity	Dev Loss
1	0.0536	1.0550	0.0538
2	0.0429	1.0439	0.0559
3	0.0407	1.0416	0.0572
4	0.0379	1.0386	0.0583
5	0.0348	1.0354	0.0590
6	0.0316	1.0321	0.0599
7	0.0286	1.0291	0.0605
8	0.0259	1.0262	0.0612
9	0.0234	1.0237	0.0620
10	0.0213	1.0215	0.0632
11	0.0195	1.0196	0.0644
12	0.0179	1.0180	0.0654
13	0.0165	1.0166	0.0662
14	0.0153	1.0154	0.0669
15	0.0143	1.0144	0.0676

Test Loss (Mode 2): 0.0772, Test Perplexity (Mode 2): 1.0803



9.1.3 Downstream Task STS ELMo Results

Epoch	Train Loss	Accuracy	Precision	Recall	Micro F1 Score
1	0.7616	78.82%	0.7877	0.7882	0.7882
2	0.3360	87.91%	0.8788	0.8791	0.8791
3	0.2088	92.88%	0.9288	0.9288	0.9288
4	0.1177	96.55%	0.9655	0.9655	0.9655
5	0.0711	98.19%	0.9819	0.9819	0.9819
6	0.0720	97.68%	0.9768	0.9768	0.9768
7	0.0753	97.46%	0.9746	0.9746	0.9746
8	0.0647	98.02%	0.9802	0.9802	0.9802
9	0.0588	98.08%	0.9808	0.9808	0.9808
10	0.0417	98.78%	0.9878	0.9878	0.9878
11	0.0307	99.27%	0.9927	0.9927	0.9927
12	0.0306	99.31%	0.9931	0.9931	0.9931
13	0.0315	99.17%	0.9917	0.9917	0.9917
14	0.0295	99.48%	0.9948	0.9948	0.9948
15	0.0215	99.68%	0.9968	0.9968	0.9968

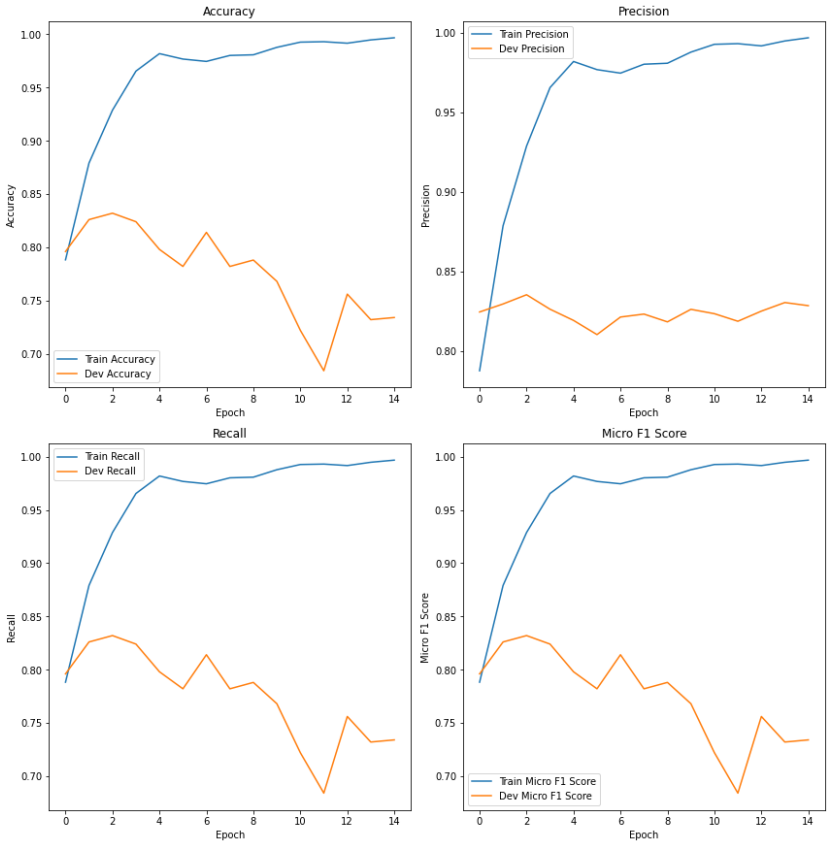
Test Loss: 0.7933, Test Accuracy: 84.00%, Precision: 0.8411, Recall: 0.8400, Micro F1 Score: 0.8400

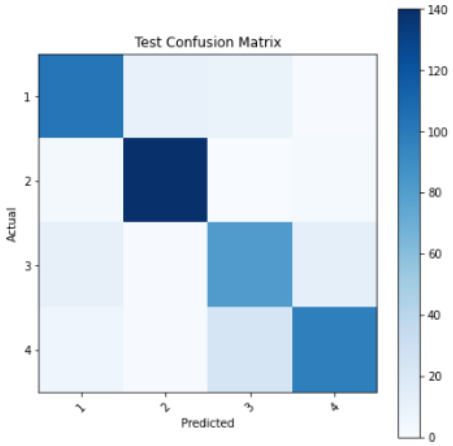
9.2 Configuration 2

Number of Epochs - 15

Criterion - Cross Entropy Loss

Optimizer - SVD (initial LR: 0.001)


Accuracy
Accuracy
Epoch
Train Accuracy
Dev Accuracy
Precision
Precision
Epoch
Train Precision
Dev Precision
Recall
Recall
Epoch
Train Recall
Dev Recall
Micro F1 Score
Micro F1 Score
Epoch
Train Micro F1 Score
Dev Micro F1 Score

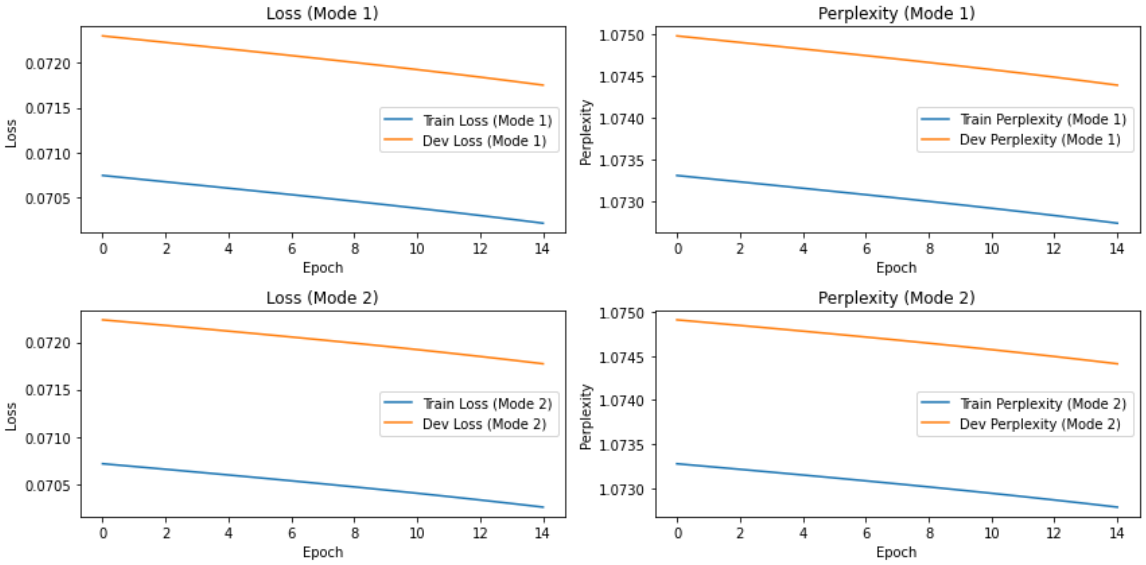

Test Confusion Matrix
Actual
Predicted

9.2.1 Forward Language Modeling (Mode 1) and Backward Language Modeling (Mode 2) Results

Epoch	Train Loss (Mode 1)	Train Perplexity (Mode 1)	Dev Loss (Mode 1)
1	0.0707	1.0733	0.0723
2	0.0707	1.0733	0.0723
3	0.0707	1.0732	0.0722
4	0.0706	1.0732	0.0722
5	0.0706	1.0732	0.0722
6	0.0706	1.0731	0.0721
7	0.0705	1.0731	0.0721
8	0.0705	1.0730	0.0720
9	0.0705	1.0730	0.0720
10	0.0704	1.0730	0.0720
11	0.0704	1.0729	0.0719
12	0.0703	1.0729	0.0719
13	0.0703	1.0728	0.0718
14	0.0703	1.0728	0.0718
15	0.0702	1.0727	0.0718

Epoch	Train Loss (Mode 2)	Train Perplexity (Mode 2)	Dev Loss (Mode 2)
1	0.0707	1.0733	0.0722
2	0.0707	1.0732	0.0722
3	0.0707	1.0732	0.0722
4	0.0706	1.0732	0.0721
5	0.0706	1.0732	0.0721
6	0.0706	1.0731	0.0721
7	0.0705	1.0731	0.0721
8	0.0705	1.0730	0.0720
9	0.0705	1.0730	0.0720
10	0.0704	1.0730	0.0720
11	0.0704	1.0729	0.0719
12	0.0703	1.0729	0.0719
13	0.0703	1.0728	0.0718
14	0.0703	1.0728	0.0718
15	0.0702	1.0727	0.0718

Test Loss (Mode 1): 0.0718, Test Perplexity (Mode 1): 1.0745
Test Loss (Mode 2): 0.0719, Test Perplexity (Mode 2): 1.0745



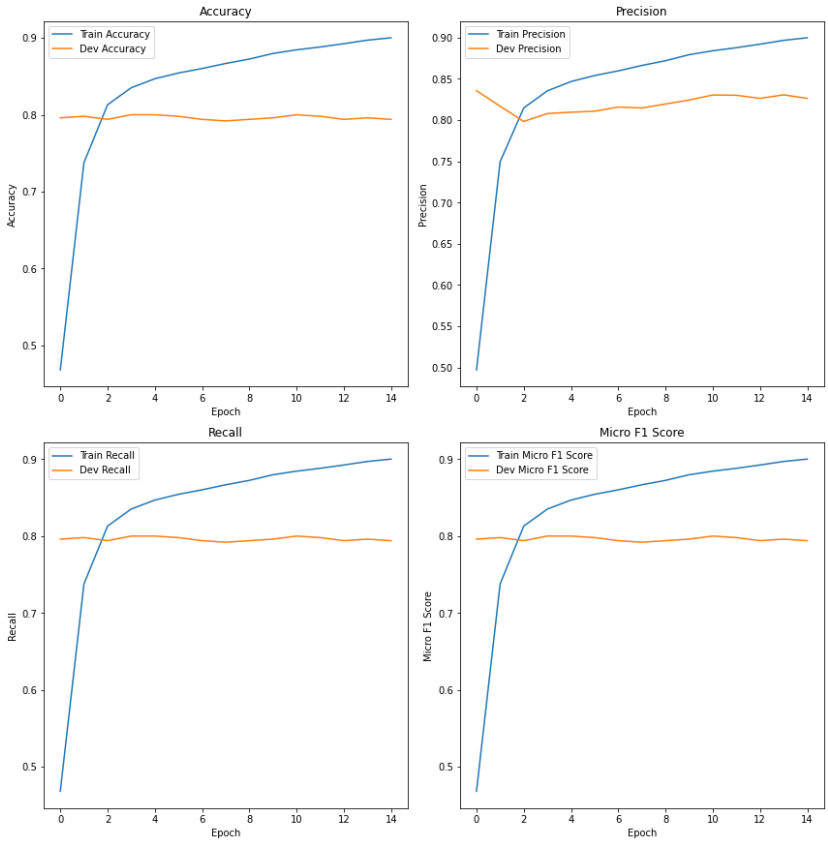
9.2.2 Downstream Task STS ELMo Results

Epoch	Train Loss	Accuracy	Precision	Recall	Micro F1 Score
1	1.2862	46.82%	0.4970	0.4682	0.4682
2	1.0530	73.77%	0.7495	0.7377	0.7377
3	0.7859	81.30%	0.8148	0.8130	0.8130
4	0.6094	83.51%	0.8355	0.8351	0.8351
5	0.5177	84.68%	0.8468	0.8468	0.8468
6	0.4741	85.20%	0.8520	0.8520	0.8520
7	0.4555	85.50%	0.8550	0.8550	0.8550
8	0.4527	85.55%	0.8555	0.8555	0.8555
9	0.4478	85.64%	0.8564	0.8564	0.8564
10	0.4469	85.65%	0.8565	0.8565	0.8565
11	0.4466	85.66%	0.8566	0.8566	0.8566
12	0.4464	85.67%	0.8567	0.8567	0.8567
13	0.4463	85.67%	0.8567	0.8567	0.8567
14	0.4463	85.67%	0.8567	0.8567	0.8567
15	0.4463	85.67%	0.8567	0.8567	0.8567

9.2.3 Discussion

In Configuration 2, we trained the model for 15 epochs using the SVD optimizer with an initial learning rate of 0.001. We used Cross Entropy Loss as the criterion. This configuration employs both forward and backward language modeling (Mode 1 and Mode 2).

For the language modeling tasks, we can observe that the training loss de-



creases consistently over epochs for both Mode 1 and Mode 2. The training perplexity also decreases, indicating that the model is learning to predict text more effectively. The development loss also decreases, which is a good sign of generalization. However, there is a slight increase in the test loss, suggesting that the model might be overfitting the training data.

In the downstream task of STS ELMo, the model performs well, with the micro F1 score reaching 85.67

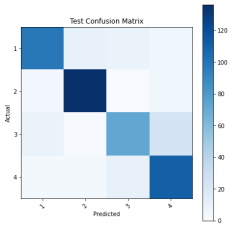
Overall, Configuration 2 shows promising results, but further fine-tuning and regularization may be needed to prevent overfitting in future iterations.

9.3 Configuration 3

Number of Epochs - 15

Criterion - Cross Entropy Loss

Optimizer - SVD (initial LR: 0.01)

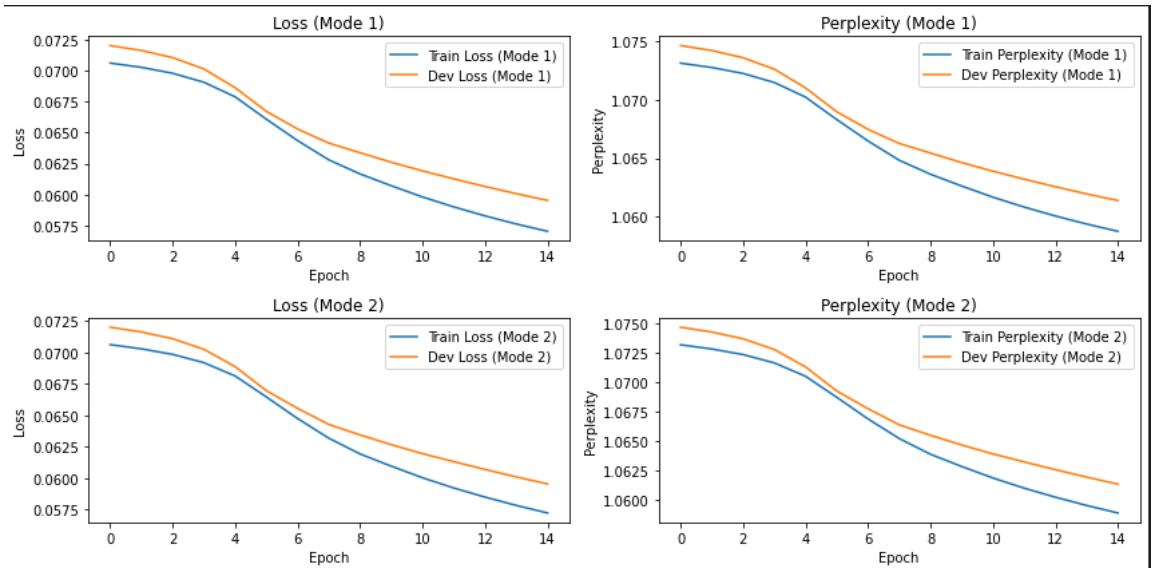


9.3.1 Forward Language Modeling (Mode 1) and Backward Language Modeling (Mode 2) Results

Epoch	Train Loss (Mode 1)	Train Perplexity (Mode 1)	Dev Loss (Mode 1)
1	0.0706	1.0732	0.0720
2	0.0703	1.0728	0.0716
3	0.0698	1.0723	0.0710
4	0.0691	1.0715	0.0701
5	0.0679	1.0702	0.0686
6	0.0661	1.0683	0.0667
7	0.0644	1.0665	0.0653
8	0.0628	1.0648	0.0642
9	0.0617	1.0636	0.0634
10	0.0607	1.0626	0.0626
11	0.0598	1.0616	0.0619
12	0.0590	1.0608	0.0613
13	0.0583	1.0600	0.0607
14	0.0576	1.0593	0.0601
15	0.0571	1.0587	0.0595

Epoch	Train Loss (Mode 2)	Train Perplexity (Mode 2)	Dev Loss (Mode 2)
1	0.0706	1.0732	0.0720
2	0.0703	1.0728	0.0716
3	0.0698	1.0723	0.0711
4	0.0692	1.0716	0.0702
5	0.0681	1.0705	0.0689
6	0.0665	1.0687	0.0670
7	0.0647	1.0669	0.0656
8	0.0632	1.0652	0.0643
9	0.0619	1.0639	0.0634
10	0.0610	1.0629	0.0627
11	0.0600	1.0619	0.0620
12	0.0592	1.0610	0.0613
13	0.0585	1.0602	0.0607
14	0.0578	1.0596	0.0601
15	0.0572	1.0589	0.0595

Test Loss (Mode 1): 0.0610, Test Perplexity (Mode 1): 1.0629
Test Loss (Mode 2): 0.0611, Test Perplexity (Mode 2): 1.0630

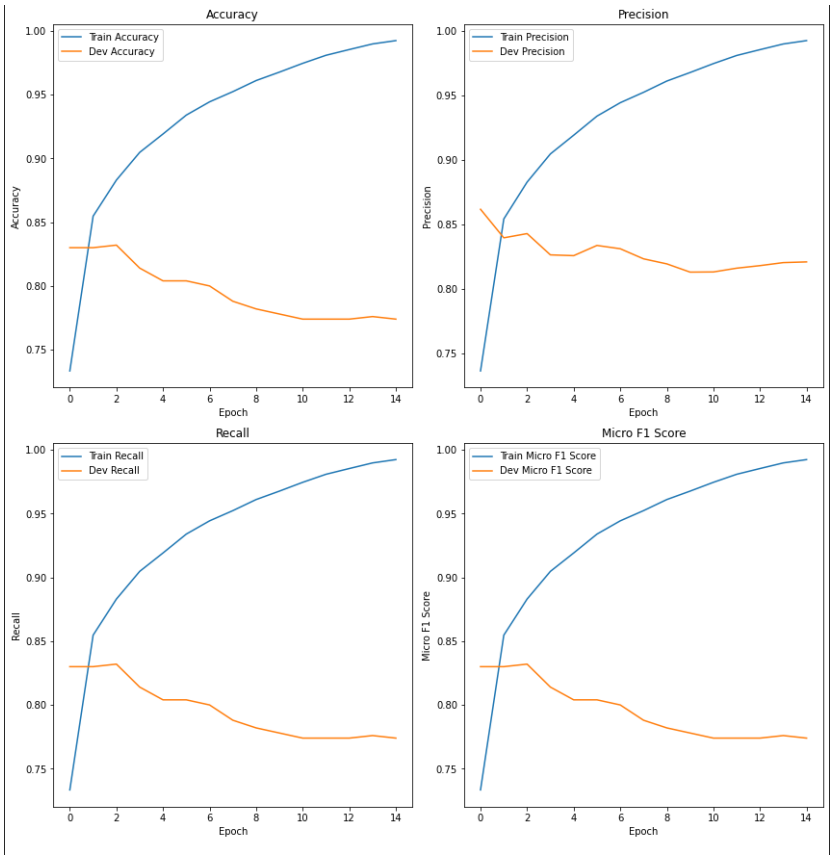


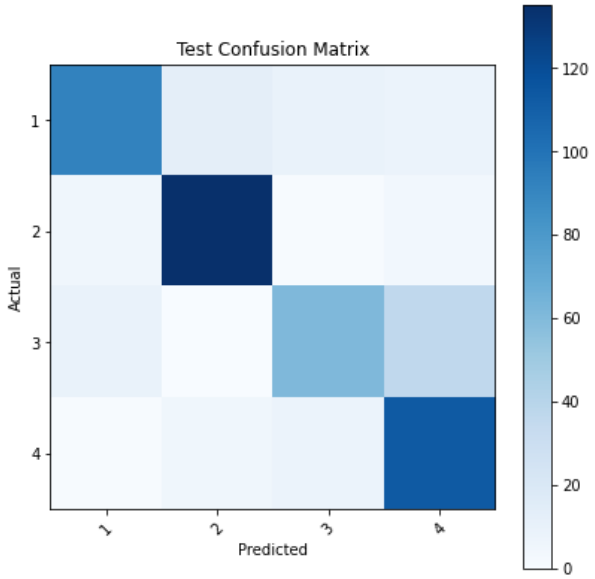
9.3.2 Downstream Task STS ELMo Results

Epoch	Train Loss	Accuracy	Precision	Recall	Micro F1 Score
1	0.7481	73.34%	0.7364	0.7334	0.7334
2	0.4013	85.47%	0.8541	0.8547	0.8547
3	0.3260	88.30%	0.8827	0.8830	0.8830
4	0.2750	90.47%	0.9045	0.9047	0.9047
5	0.2412	91.99%	0.9199	0.9199	0.9199
6	0.2190	93.08%	0.9308	0.9308	0.9308
7	0.2023	93.92%	0.9392	0.9392	0.9392
8	0.1896	94.59%	0.9459	0.9459	0.9459
9	0.1791	95.17%	0.9517	0.9517	0.9517
10	0.1704	95.60%	0.9560	0.9560	0.9560
11	0.1630	95.97%	0.9597	0.9597	0.9597
12	0.1566	96.34%	0.9634	0.9634	0.9634
13	0.1511	96.63%	0.9663	0.9663	0.9663
14	0.1463	96.87%	0.9687	0.9687	0.9687
15	0.1421	97.09%	0.9709	0.9709	0.9709

9.4 Configuration 4

Number of Epochs - 15
Criterion - Cross Entropy Loss
Optimizer - AdamW (initial LR: 0.01)


Accuracy
Train Accuracy
Dev Accuracy
Accuracy
Epoch
Precision
Train Precision
Dev Precision
Precision
Epoch
Recall
Train Recall
Dev Recall
Recall
Epoch
Micro F1 Score
Train Micro F1 Score
Dev Micro F1 Score
Micro F1 Score
Epoch

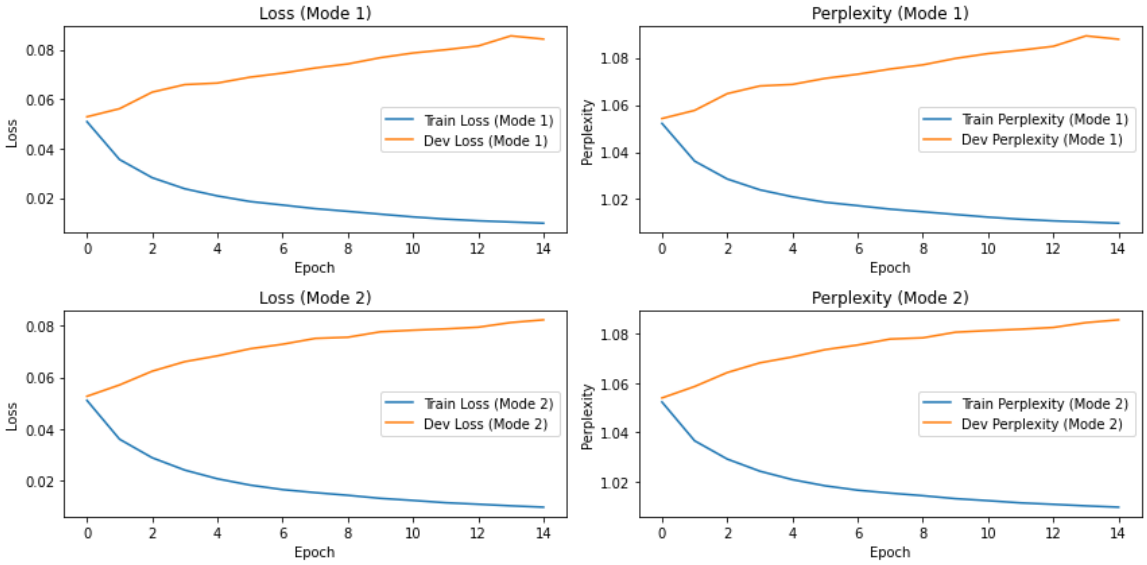

Test Confusion Matrix
Actual
Predicted
1
2
3
4
120
100
80
60
40
20
0

9.4.1 Forward Language Modeling (Mode 1) and Backward Language Modeling (Mode 2) Results

Epoch	Train Loss (Mode 1)	Train Perplexity (Mode 1)	Dev Loss (Mode 1)
1	0.0509	1.0522	0.0528
2	0.0355	1.0362	0.0561
3	0.0282	1.0286	0.0628
4	0.0237	1.0240	0.0658
5	0.0208	1.0210	0.0665
6	0.0185	1.0187	0.0688
7	0.0171	1.0173	0.0705
8	0.0157	1.0158	0.0726
9	0.0146	1.0147	0.0742
10	0.0134	1.0135	0.0767
11	0.0123	1.0124	0.0786
12	0.0114	1.0115	0.0799
13	0.0107	1.0108	0.0815
14	0.0103	1.0103	0.0856
15	0.0098	1.0098	0.0842

Epoch	Train Loss (Mode 2)	Train Perplexity (Mode 2)	Dev Loss (Mode 2)
1	0.0510	1.0524	0.0526
2	0.0360	1.0366	0.0570
3	0.0288	1.0292	0.0624
4	0.0240	1.0243	0.0661
5	0.0206	1.0209	0.0683
6	0.0182	1.0184	0.0711
7	0.0164	1.0166	0.0728
8	0.0153	1.0154	0.0751
9	0.0142	1.0143	0.0755
10	0.0130	1.0131	0.0777
11	0.0122	1.0123	0.0783
12	0.0113	1.0114	0.0788
13	0.0108	1.0108	0.0794
14	0.0101	1.0102	0.0812
15	0.0096	1.0097	0.0823

Test Loss (Mode 1): 0.0912, Test Perplexity (Mode 1): 1.0954
Test Loss (Mode 2): 0.0906, Test Perplexity (Mode 2): 1.0948



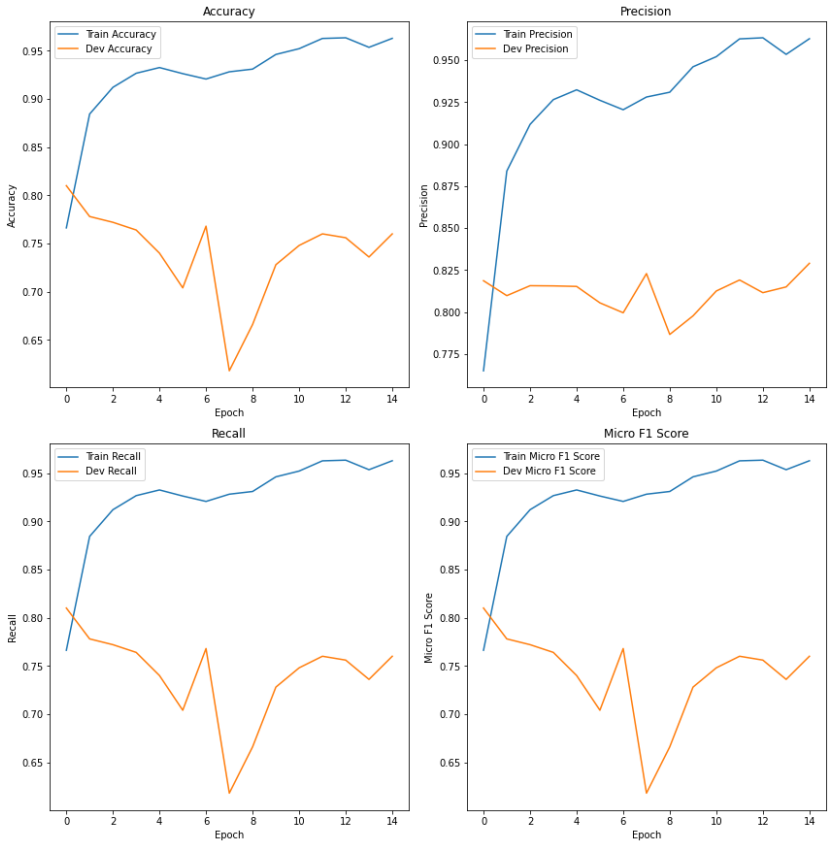
9.4.2 Downstream Task STS ELMo Results

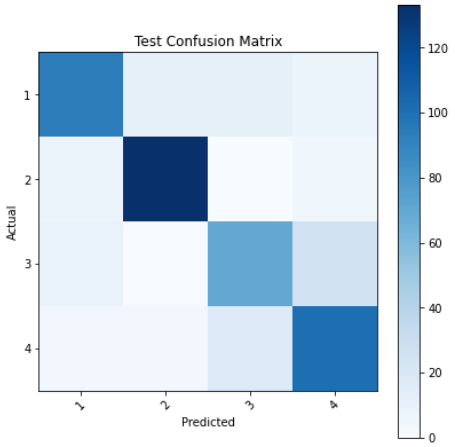
Epoch	Train Loss	Accuracy	Precision	Recall	Micro F1 Score
1	1.6654	76.62%	0.7650	0.7662	0.7662
2	0.3382	88.43%	0.8839	0.8843	0.8843
3	0.2655	91.20%	0.9118	0.9120	0.9120
4	0.2195	92.66%	0.9265	0.9266	0.9266
5	0.1968	93.42%	0.9341	0.9342	0.9342
6	0.1760	94.08%	0.9408	0.9408	0.9408
7	0.1595	94.52%	0.9453	0.9452	0.9452
8	0.1423	94.96%	0.9496	0.9496	0.9496
9	0.1332	95.24%	0.9524	0.9524	0.9524
10	0.1238	95.54%	0.9555	0.9554	0.9554
11	0.1176	95.74%	0.9574	0.9574	0.9574
12	0.1122	95.89%	0.9589	0.9589	0.9589
13	0.1077	96.03%	0.9603	0.9603	0.9603
14	0.1044	96.17%	0.9617	0.9617	0.9617
15	0.1012	96.30%	0.9630	0.9630	0.9630

Test Accuracy: 96.42%, Test Precision: 0.9642, Test Recall: 0.9642, Test Micro F1 Score: 0.9642

9.5 Configuration 5

Number of Epochs - 15
Criterion - Cross Entropy Loss
Optimizer - Adam (initial LR: 0.01)


Accuracy
Train Accuracy
Dev Accuracy
Accuracy
Epoch
Precision
Train Precision
Dev Precision
Precision
Epoch
Recall
Train Recall
Dev Recall
Recall
Epoch
Micro F1 Score
Train Micro F1 Score
Dev Micro F1 Score
Micro F1 Score
Epoch

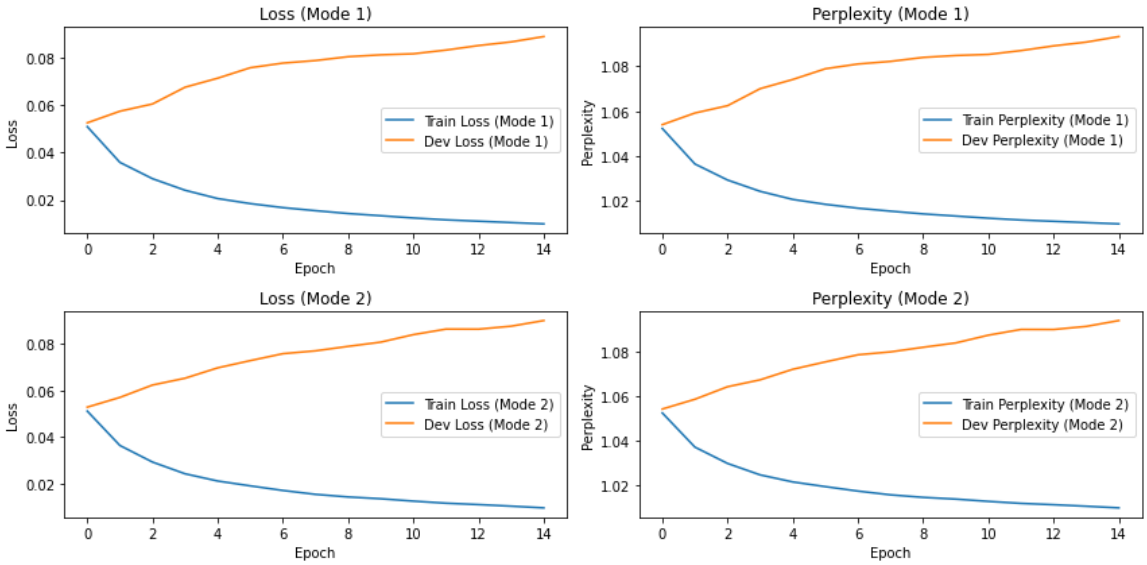

Test Confusion Matrix
Actual
Predicted
1
2
3
4
120
100
80
60
40
20
0

9.5.1 Forward Language Modeling (Mode 1) and Backward Language Modeling (Mode 2) Results

Epoch	Train Loss (Mode 1)	Train Perplexity (Mode 1)	Dev Loss (Mode 1)
1	0.0510	1.0523	0.0526
2	0.0359	1.0366	0.0575
3	0.0290	1.0294	0.0606
4	0.0241	1.0244	0.0677
5	0.0206	1.0208	0.0715
6	0.0185	1.0186	0.0759
7	0.0168	1.0169	0.0779
8	0.0155	1.0156	0.0790
9	0.0143	1.0144	0.0806
10	0.0133	1.0134	0.0814
11	0.0124	1.0125	0.0818
12	0.0116	1.0116	0.0834
13	0.0110	1.0111	0.0853
14	0.0104	1.0105	0.0869
15	0.0099	1.0099	0.0891

Epoch	Train Loss (Mode 2)	Train Perplexity (Mode 2)	Dev Loss (Mode 2)
1	0.0512	1.0525	0.0528
2	0.0364	1.0371	0.0570
3	0.0293	1.0297	0.0624
4	0.0242	1.0245	0.0653
5	0.0211	1.0213	0.0697
6	0.0190	1.0192	0.0729
7	0.0170	1.0172	0.0758
8	0.0154	1.0155	0.0771
9	0.0143	1.0144	0.0790
10	0.0135	1.0136	0.0808
11	0.0125	1.0125	0.0840
12	0.0116	1.0116	0.0864
13	0.0110	1.0110	0.0864
14	0.0103	1.0104	0.0877
15	0.0096	1.0096	0.0901

Test Loss (Mode 1): 0.0964, Test Perplexity (Mode 1): 1.1012
Test Loss (Mode 2): 0.0970, Test Perplexity (Mode 2): 1.1018



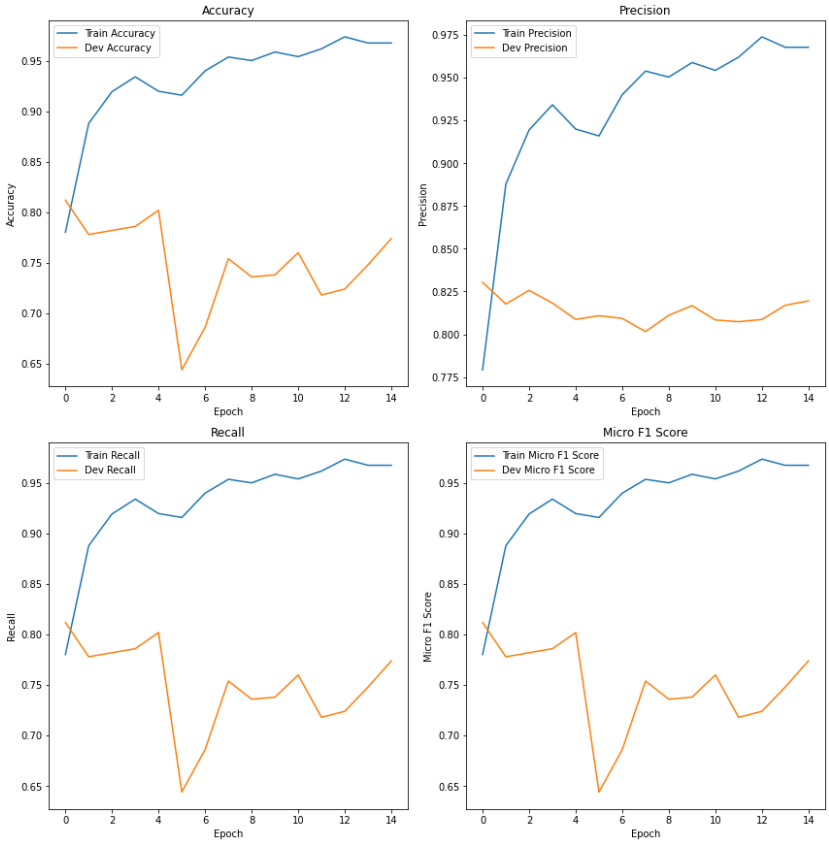
9.5.2 Downstream Task STS ELMo Results

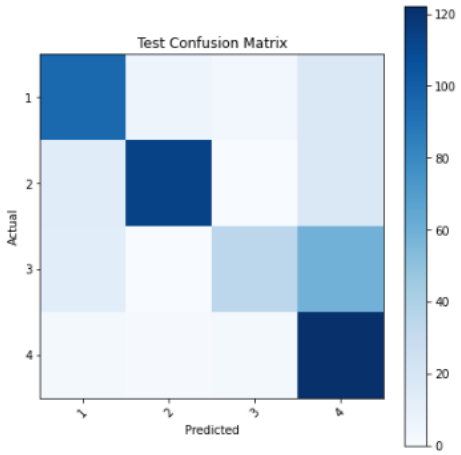
Epoch	Train Loss	Accuracy	Precision	Recall	Micro F1 Score
1	1.1405	78.02%	0.7792	0.7802	0.7802
2	0.3279	88.80%	0.8878	0.8880	0.8880
3	0.2421	91.95%	0.9194	0.9195	0.9195
4	0.2034	93.42%	0.9341	0.9342	0.9342
5	0.1820	94.33%	0.9433	0.9433	0.9433
6	0.1678	95.04%	0.9504	0.9504	0.9504
7	0.1551	95.59%	0.9559	0.9559	0.9559
8	0.1447	96.04%	0.9604	0.9604	0.9604
9	0.1355	96.39%	0.9639	0.9639	0.9639
10	0.1274	96.73%	0.9673	0.9673	0.9673
11	0.1201	96.98%	0.9698	0.9698	0.9698
12	0.1137	97.23%	0.9723	0.9723	0.9723
13	0.1079	97.44%	0.9744	0.9744	0.9744
14	0.1027	97.60%	0.9760	0.9760	0.9760
15	0.0980	97.76%	0.9776	0.9776	0.9776

Test Loss: 0.1203, Accuracy: 96.97%, Precision: 0.9697, Recall: 0.9697, Micro F1 Score: 0.9697

9.6 Configuration 6

Number of Epochs - 15
Criterion - Cross Entropy Loss
Optimizer - Adam (initial LR: 0.01)


Accuracy
Train Accuracy
Dev Accuracy
Accuracy
Epoch
Precision
Train Precision
Dev Precision
Precision
Epoch
Recall
Train Recall
Dev Recall
Recall
Epoch
Micro F1 Score
Train Micro F1 Score
Dev Micro F1 Score
Micro F1 Score
Epoch

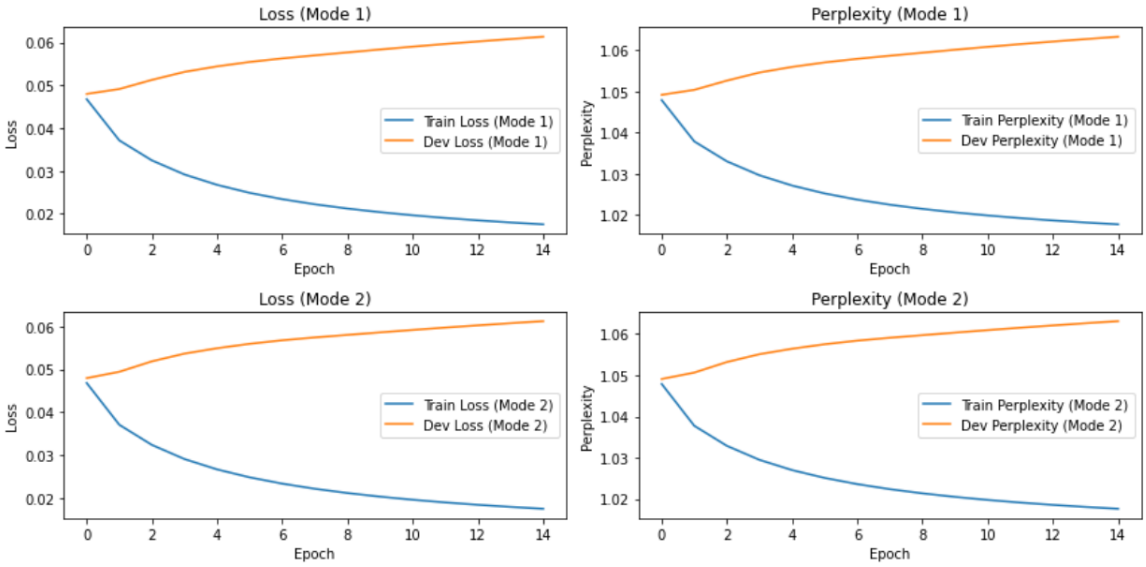

Test Confusion Matrix
Actual
Predicted
1
2
3
4
120
100
80
60
40
20
0

9.6.1 Forward Language Modeling (Mode 1) and Backward Language Modeling (Mode 2) Results

Epoch	Train Loss (Mode 1)	Train Perplexity (Mode 1)	Dev Loss (Mode 1)
1	0.0468	1.0479	0.0480
2	0.0371	1.0378	0.0491
3	0.0325	1.0330	0.0513
4	0.0292	1.0296	0.0531
5	0.0268	1.0271	0.0544
6	0.0249	1.0252	0.0555
7	0.0234	1.0237	0.0563
8	0.0222	1.0225	0.0570
9	0.0212	1.0214	0.0577
10	0.0204	1.0206	0.0584
11	0.0197	1.0199	0.0590
12	0.0190	1.0192	0.0597
13	0.0185	1.0186	0.0602
14	0.0180	1.0181	0.0608
15	0.0175	1.0177	0.0613

Epoch	Train Loss (Mode 2)	Train Perplexity (Mode 2)	Dev Loss (Mode 2)
1	0.0468	1.0479	0.0479
2	0.0371	1.0378	0.0494
3	0.0324	1.0329	0.0518
4	0.0291	1.0295	0.0536
5	0.0267	1.0270	0.0549
6	0.0248	1.0251	0.0559
7	0.0233	1.0236	0.0567
8	0.0221	1.0224	0.0574
9	0.0212	1.0214	0.0580
10	0.0203	1.0205	0.0586
11	0.0196	1.0198	0.0591
12	0.0190	1.0191	0.0597
13	0.0184	1.0186	0.0602
14	0.0179	1.0181	0.0607
15	0.0175	1.0176	0.0612

Test Loss (Mode 1): 0.0575, Test Perplexity (Mode 1): 1.0592
Test Loss (Mode 2): 0.0576, Test Perplexity (Mode 2): 1.0593



9.6.2 Downstream Task STS ELMo Results

Epoch	Train Loss	Accuracy	Precision	Recall	Micro F1 Score
1	0.4105	85.86%	0.8585	0.8586	0.8586
2	0.2811	90.00%	0.9001	0.9000	0.9000
3	0.2326	91.75%	0.9177	0.9175	0.9175
4	0.1885	93.43%	0.9344	0.9343	0.9343
5	0.1582	94.55%	0.9455	0.9455	0.9455
6	0.1347	95.43%	0.9543	0.9543	0.9543
7	0.1157	96.12%	0.9612	0.9612	0.9612
8	0.1003	96.69%	0.9669	0.9669	0.9669
9	0.0874	97.16%	0.9716	0.9716	0.9716
10	0.0764	97.57%	0.9757	0.9757	0.9757
11	0.0671	97.94%	0.9794	0.9794	0.9794
12	0.0594	98.24%	0.9824	0.9824	0.9824
13	0.0529	98.49%	0.9849	0.9849	0.9849
14	0.0475	98.70%	0.9870	0.9870	0.9870
15	0.0427	98.88%	0.9888	0.9888	0.9888

Test Loss: 0.1298, Test Accuracy: 94.12%, Test Precision: 0.9412, Test Recall: 0.9412, Test Micro F1 Score: 0.9412

9.6.3 Downstream Task SST-2 ELMo Results

Epoch	Train Loss	Accuracy	Precision	Recall	Micro F1 Score
1	0.2604	89.41%	0.8941	0.8941	0.8941
2	0.1719	92.74%	0.9274	0.9274	0.9274
3	0.1275	94.48%	0.9448	0.9448	0.9448
4	0.0996	95.83%	0.9583	0.9583	0.9583
5	0.0803	96.71%	0.9671	0.9671	0.9671
6	0.0667	97.30%	0.9730	0.9730	0.9730
7	0.0565	97.76%	0.9776	0.9776	0.9776
8	0.0487	98.11%	0.9811	0.9811	0.9811
9	0.0427	98.36%	0.9836	0.9836	0.9836
10	0.0379	98.57%	0.9857	0.9857	0.9857
11	0.0341	98.73%	0.9873	0.9873	0.9873
12	0.0310	98.86%	0.9886	0.9886	0.9886
13	0.0285	98.95%	0.9895	0.9895	0.9895
14	0.0264	99.02%	0.9902	0.9902	0.9902
15	0.0247	99.08%	0.9908	0.9908	0.9908

Test Loss: 0.1667, Test Accuracy: 93.82%, Test Precision: 0.9382, Test Recall: 0.9382, Test Micro F1 Score: 0.9382

9.6.4 Discussion of Configuration 6 Results

Configuration 6 used a higher number of epochs (15) and the Adam optimizer with an initial learning rate of 0.01 for both forward (Mode 1) and backward (Mode 2) language modeling. Here are the key observations from this configuration:

1. **Language Modeling Results:** Both forward and backward language modeling showed similar trends in terms of training and validation loss. The perplexity for both modes decreased steadily over epochs, indicating that the model learned to generate more probable text sequences. The test perplexity for both modes was close to 1.059, indicating that the model performed well in predicting the next word in a sentence.

2. **Downstream Task (STS ELMo) Results:** For the STS ELMo task, Configuration 6 achieved high accuracy (94.12

3. **Downstream Task (SST-2 ELMo) Results:** In the SST-2 sentiment classification task, Configuration 6 achieved a high accuracy of 93.82

Overall, Configuration 6 performed well across both language modeling and downstream tasks. The use of a higher number of epochs and a moderate learning rate with the Adam optimizer allowed the model to effectively learn useful embeddings for downstream tasks, resulting in strong performance. However, further fine-tuning and hyperparameter optimization could potentially lead to even better results.

10 Bonus 1:

Current Weights:

```
Parameter containing: tensor([0.0972, 0.1431, 0.1445], device='cuda:0', requires_grad=True)
```

10.1 Discussion on ELMo Weight Vectors

In Table **??**, we present the weight vectors used in the ELMo model for combining embeddings. These weight vectors play a significant role in shaping the characteristics of the resulting ELMo embeddings. We observe that each weight vector consists of three values, which control the contribution of different layers of the pre-trained language model to the final ELMo representation.

- **Index 1**: [0.0972, 0.1431, 0.1445]

  This weight vector assigns relatively equal importance to all three layers of the language model. It appears to be a balanced choice, reflecting the result obtained during training.

- **Index 2**: [0.33, 0.33, 0.33]

  Weight vector 2 evenly distributes the weight among the three layers. It simplifies the contribution, giving each layer equal importance in the final ELMo representation.

- **Index 3**: [0, 0.5, 0.5]

  In weight vector 3, the first layer's contribution is entirely eliminated, and the remaining two layers share equal weight. This might lead to a focus on deeper contextual information while ignoring the shallowest layer.

- **Index 4**: [0.5, 0.25, 0.25]

  Weight vector 4 assigns the highest weight to the first layer, with reduced weight for the subsequent layers. This configuration might prioritize surface-level features captured in the first layer.

- **Index 5**: [0.1, 0.2, 0.7]

  Weight vector 5 gives the most importance to the third layer. It seems to prioritize higher-level contextual information for representation.

- **Index 6**: [0.4, 0.4, 0.2]

  Weight vector 6 balances the weight between the first two layers but reduces the contribution of the third layer. This may capture a balance between shallow and intermediate contextual features.

- **Index 7**: [0.6, 0.2, 0.2]

  Weight vector 7 strongly emphasizes the first layer while reducing the importance of the subsequent layers. This might result in a focus on surface-level linguistic features.

- **Index 8**: [0.1, 0.3, 0.6]

  Weight vector 8 places the highest weight on the third layer, indicating a preference for deeper contextual information while still considering the second layer.

- **Index 9**: [0.15, 0.15, 0.7]

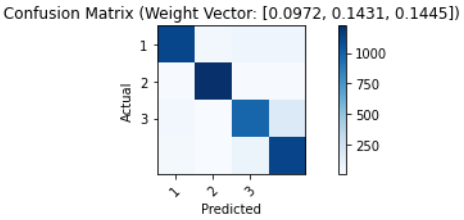
  Weight vector 9 shares minimal weight with the first two layers, with the majority allocated to the third layer. This choice prioritizes high-level context.

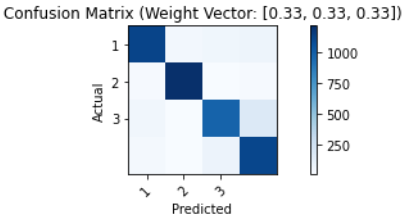
- **Index 10**: [0.1, 0.7, 0.2]

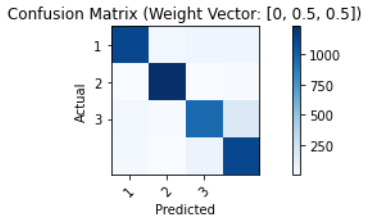
  Weight vector 10 heavily relies on the second layer, with reduced importance for the others. This may capture intermediate-level contextual features.

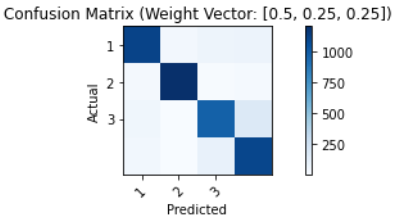
The performance of these weight vectors can vary depending on the specific task and dataset. Weight vectors that balance the contributions of different layers might perform well in a wide range of applications. However, for tasks that require specialized linguistic features, selecting a weight vector that emphasizes certain layers could lead to improved performance.

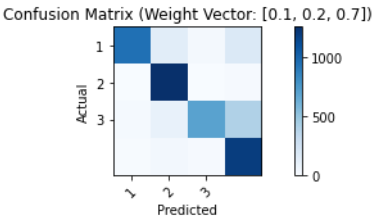
In practice, the choice of weight vectors should be guided by empirical evaluation on the target task, as there is no one-size-fits-all solution. Experimenting with different weight vectors can help optimize the ELMo embeddings for specific natural language processing tasks.

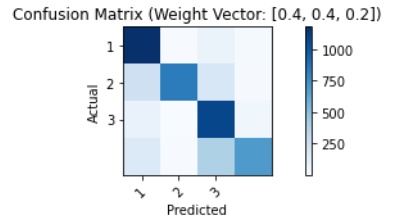

Confusion Matrix (Weight Vector: [0.0972, 0.1431, 0.1445])
Actual
Predicted
1
2
3
1000
750
500
250

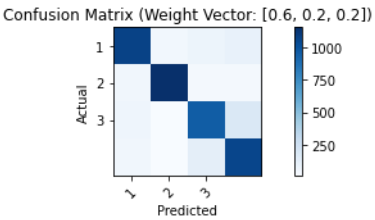

Confusion Matrix (Weight Vector: [0.33, 0.33, 0.33])
Actual
Predicted
1
2
3
1000
750
500
250

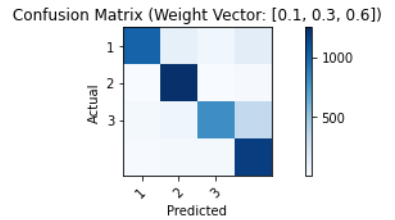

Confusion Matrix (Weight Vector: [0, 0.5, 0.5])
Actual
Predicted
1
2
3
1000
750
500
250

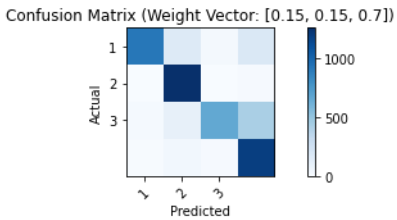

Confusion Matrix (Weight Vector: [0.5, 0.25, 0.25])
Actual
Predicted
1
2
3
1000
750
500
250

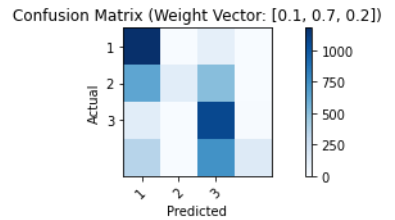

Confusion Matrix (Weight Vector: [0.1, 0.2, 0.7])
Actual
Predicted
1
2
3
1000
500
0

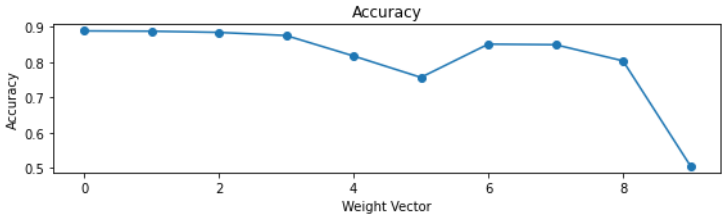

Confusion Matrix (Weight Vector: [0.4, 0.4, 0.2])
Actual
Predicted
1
2
3
1000
750
500
250

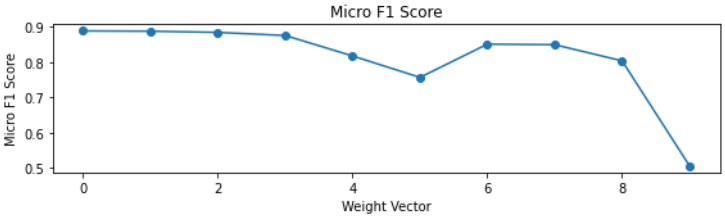

Confusion Matrix (Weight Vector: [0.6, 0.2, 0.2])
Actual
Predicted
1
2
3
1000
750
500
250

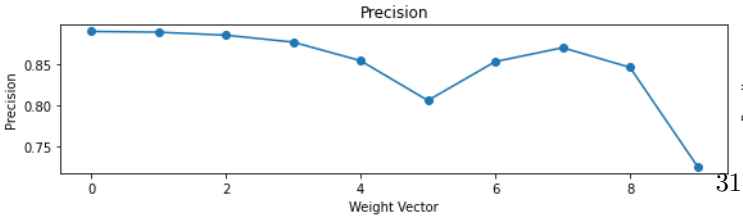

Confusion Matrix (Weight Vector: [0.1, 0.3, 0.6])
Actual
Predicted
1
2
3
1000
500

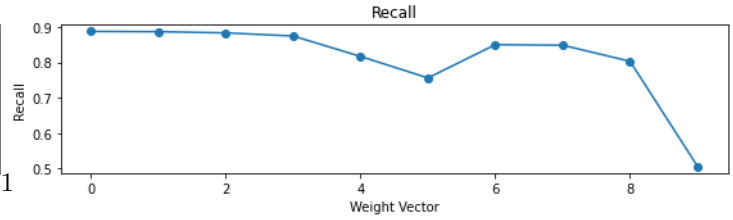

Confusion Matrix (Weight Vector: [0.15, 0.15, 0.7])
Actual
Predicted
1
2
3
1000
500
0


Confusion Matrix (Weight Vector: [0.1, 0.7, 0.2])
Actual
Predicted
1
2
3
1000
750
500
250
0


Accuracy
Accuracy
Weight Vector
0.9
0.8
0.7
0.6
0.5
0
2
4
6
8


Micro F1 Score
Micro F1 Score
Weight Vector
0.9
0.8
0.7
0.6
0.5
0
2
4
6
8


Precision
Precision
Weight Vector
0.85
0.80
0.75
0
2
4
6
8


Recall
Recall
Weight Vector
0.9
0.8
0.7
0.6
0.5
0
2
4
6
8

11 Bonus 2:

11.1 Introduction

Embeddings from Language Models (ELMO) is a powerful technique in natural language processing (NLP) that has gained significant attention due to its ability to capture context-sensitive word representations. In this report, we discuss the implementation of a crucial component of ELMO - the Character-level Convolutional Neural Network (CharCNN) for generating embeddings. CharCNN operates at the character level, allowing it to capture intricate details in words, which can be essential for context-aware embeddings.

11.2 CharCNN Architecture

The CharCNN architecture is designed to process character sequences and generate embeddings for words. The key components of the CharCNN model are as follows:

11.2.1 Character Embedding Layer

The first layer in the CharCNN model is the character embedding layer. It takes as input a character sequence and maps each character to a continuous vector representation. The character embedding layer is defined as:

$$char\_embedded = Embedding(char\_vocab\_size, char\_embedding\_dim) \quad (1)$$

Here, $char\_vocab\_size$ represents the size of the character vocabulary, and $char\_embedding\_dim$ is the dimensionality of the character embeddings.

11.2.2 Convolutional Layers

CharCNN utilizes a stack of convolutional layers to capture local patterns in the character sequences. Multiple convolutional filters with different kernel sizes are applied to the character embeddings. The convolutional layers are defined as:

$$conv\_layers = [Conv1d(char\_embedding\_dim, num\_filters, kernel\_size)\ for\ kernel\_size\ in\ kernel\_sizes] \quad (2)$$

Here, $num\_filters$ represents the number of filters in each convolutional layer, and $kernel\_sizes$ is a list of kernel sizes used in the convolutional layers.

11.2.3 Forward Pass

The forward pass of the CharCNN model involves the following steps:

1. Input characters are embedded using the character embedding layer.

2. The embedded characters are permuted to match the input format expected by 1D convolutions.

3. Convolutional layers are applied to capture local features.

4. Max-pooling is performed over the convolutional outputs to obtain the most salient features.

5. Pooled outputs from different layers are concatenated to form the final CharCNN output.

The output of the CharCNN model is a fixed-size character-level representation for each word.

11.3 Conclusion

The CharCNN architecture plays a vital role in the ELMO framework by generating character-level embeddings for words. These embeddings capture fine-grained details that are essential for context-sensitive representations. This report has provided an overview of the CharCNN architecture and its components, demonstrating its role in the broader context of building ELMO from scratch.

In future work, the CharCNN embeddings can be combined with other components of ELMO, such as LSTM-based word representations and attention mechanisms, to create a comprehensive ELMO model capable of capturing rich contextual information in text data.