ELMO: Deep Contextualized Word Representations

ELMO Pre-training

Hyper-parameters used:

- **Batch Size**: 32
- **Hidden Units**: 150
- **Embedding Size**: 150
- **Epochs**: 10

```
Epoch 1, Loss: 8.426731818771362
Epoch 2, Loss: 7.098492701085409
Epoch 3, Loss: 6.5969685976664225
Epoch 4, Loss: 6.303259781901041
Epoch 5, Loss: 6.0989046332041426
Epoch 6, Loss: 5.94296741587321
Epoch 7, Loss: 5.8170874876658125
Epoch 8, Loss: 5.711459554672241
Epoch 9, Loss: 5.619714071019491
Epoch 10, Loss: 5.540601449966431
```

Dataset Preparation (`create_dataset` class)

Processes text data and prepares inputs for training.

Key Steps:

- **Preprocessing:** Expands contractions, lowercases text, removes punctuation, replaces URLs, and tokenizes sentences.
- **Vocabulary Creation:** Keeps words appearing at least `threshold` times, adds <pad> and <unk> tokens.
- **Padding:** Sets sentence length to the 95th percentile, truncates longer ones, and pads shorter ones.
- **Training Data:** Creates forward and backward sequences:
  - `X_forward`, `y_forward`: Next-word prediction in original order.
  - `X_backward`, `y_backward`: Next-word prediction in reverse order.

BiLSTM-Based Model (`Elmo` class)

Architecture:

- **Embedding layer**: Converts tokens to vectors.
- **LSTMs (2 forward, 2 backward)**: Capture bidirectional context.
- **Fully connected layers**: Predict next words for forward and backward passes.

Training (`train_elmo`)

Uses **Cross-Entropy Loss** and **Adam Optimizer** for mini-batch training.

Steps Per Epoch:

1. Load mini-batches.
2. Compute forward and backward LSTM outputs.
3. Convert targets to one-hot format.
4. Calculate loss and update weights.

Downstream Task

Hyper-parameters used:

- **Batch Size**: 32
- **Hidden Units**: 128
- **Number of Layers**: 2
- **Epochs**: 10
- **Input Size**: 300
- **Activation Function**: ReLU
- **Bidirectional**: True

Dataset Preparation

The `Create_dataset_classification` class processes text data by:

1. **Reading CSV data** containing descriptions and class labels.
2. **Preprocessing text**, including:
   - Expanding contractions (e.g., "don't" → "do not").
   - Lowercasing and removing URLs/punctuation.
   - Tokenizing sentences and adding special tokens (`<s>` and `</s>`).
3. **Padding sentences** to a fixed length using a percentile-based strategy.
4. **Converting words to indices** based on a given vocabulary (`word2idx`).
5. **Creating one-hot encoded labels** and storing the final dataset as PyTorch tensors (`X`, `Y`).

Training the Model

The `train_classifier` function trains the LSTM-based classifier:

1. **Computes forward and backward embeddings** using an `elmo_model`.

2. **Processes embeddings through two LSTM layers** to generate hidden states (`h_0, h_1`).
3. **Combines representations** using one of the three methods.
4. **Passes the combined representation through an LSTM classifier**.
5. **Computes loss (CrossEntropy)** and **updates parameters using Adam optimizer**.
6. **Tracks training and validation loss across epochs**.

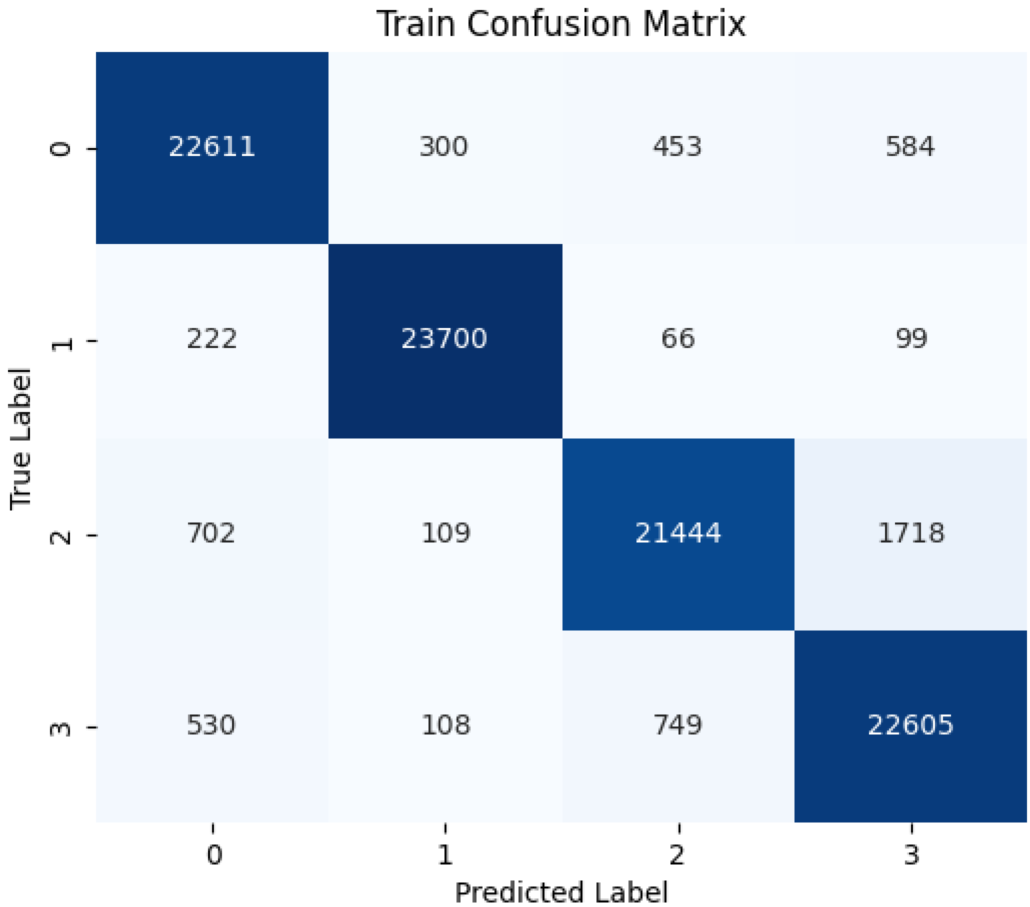
Method	Flexibility	Learnable Weights?	Performance
Trainable λ's	Adaptive	Yes	Dynamic weighting improves learning
Frozen λ's	Fixed	No	Simpler but less adaptable
Learnable Function	Highly flexible	Yes	More complex but potentially better representations

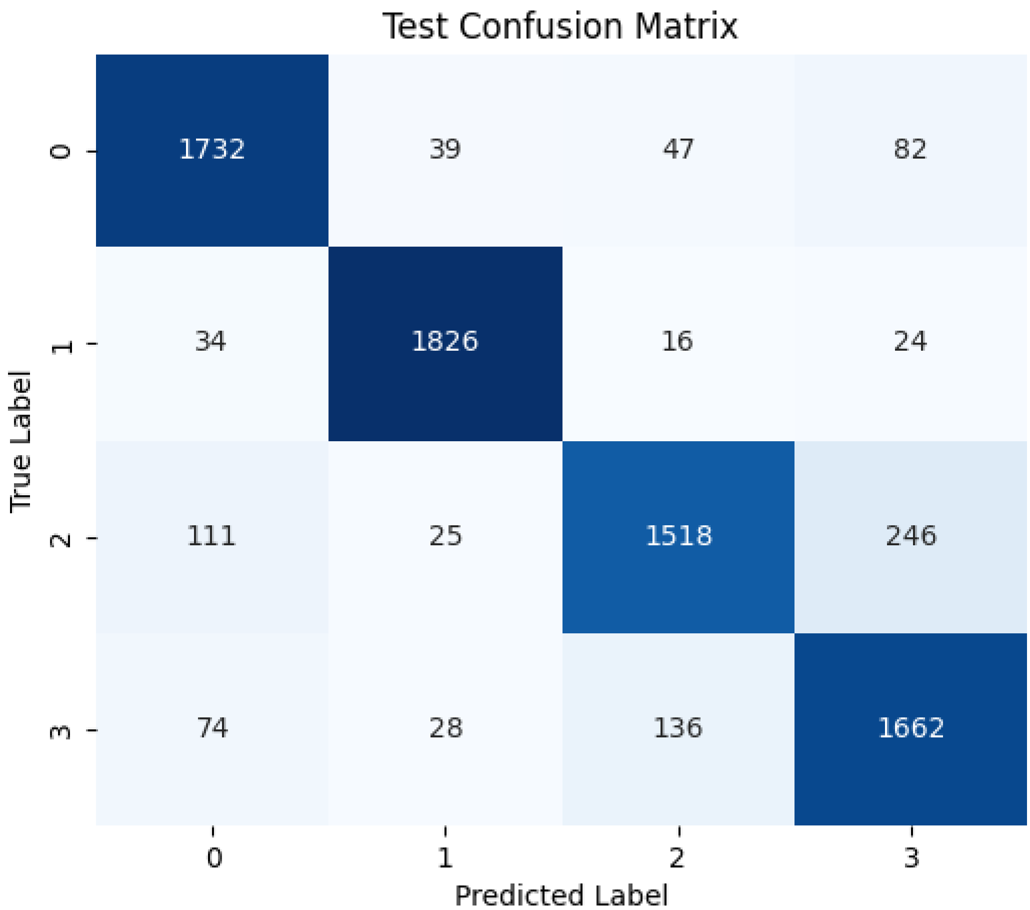
Trainable λ's

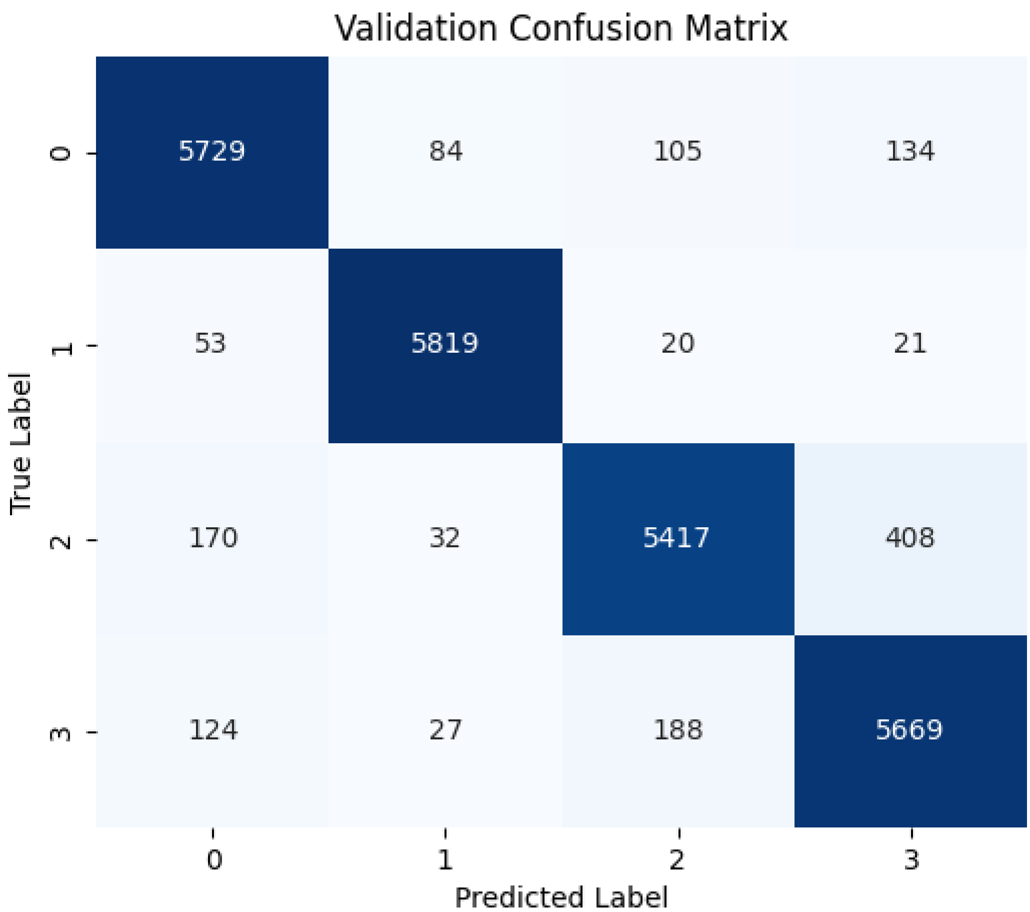
```
Epoch 1, Loss: 0.4037661294204493, Val Loss: 0.31051146374146144
Epoch 2, Loss: 0.2934947016617904, Val Loss: 0.293622295593222
Epoch 3, Loss: 0.2545347608998418, Val Loss: 0.2892797255888581
Epoch 4, Loss: 0.2166401638649404, Val Loss: 0.30148572623233
Epoch 5, Loss: 0.18255442553913842, Val Loss: 0.2960284621516864
Model for method 1 saved as 'classification_model_method1.pt'
```

- The model learns three **trainable weights** (`λ1, λ2, λ3`) that combine the embeddings and LSTM hidden states.
- Formula: $$ x = \lambda_1 \cdot e_0 + \lambda_2 \cdot h_0 + \lambda_3 \cdot h_1 $$
- The parameters (`λ1, λ2, λ3`) are initialized as learnable PyTorch tensors.
- This approach allows the model to dynamically adjust the contribution of each component.

```
Evaluating model: classification_model_method1.pt
Train Accuracy: 0.9413, F1 Score: 0.9412, Precision: 0.9416, Recall: 0.9413
Validation Accuracy: 0.9431, F1 Score: 0.9430, Precision: 0.9434, Recall: 0.9431
Test Accuracy: 0.8866, F1 Score: 0.8862, Precision: 0.8873, Recall: 0.8866
Train Confusion Matrix:
[[22611   300   453   584]
 [  222 23700    66    99]
 [  702   109 21444  1718]
 [  530   108   749 22605]]
Validation Confusion Matrix:
[[5729   84  105  134]
 [  53 5819   20   21]
 [ 170   32 5417  408]
 [ 124   27  188 5669]]
Test Confusion Matrix:
[[1732   39   47   82]
 [  34 1826   16   24]
 [ 111   25 1518  246]
 [  74   28  136 1662]]
Lambda values:
lamda1: 0.9233, lamda2: -0.4068, lamda3: -0.8539
```


Train Confusion Matrix
True Label
Predicted Label
0
1
2
3
22611
300
453
584
222
23700
66
99
702
109
21444
1718
530
108
749
22605


Test Confusion Matrix
True Label
Predicted Label
0
1
2
3
1732
39
47
82
34
1826
16
24
111
25
1518
246
74
28
136
1662



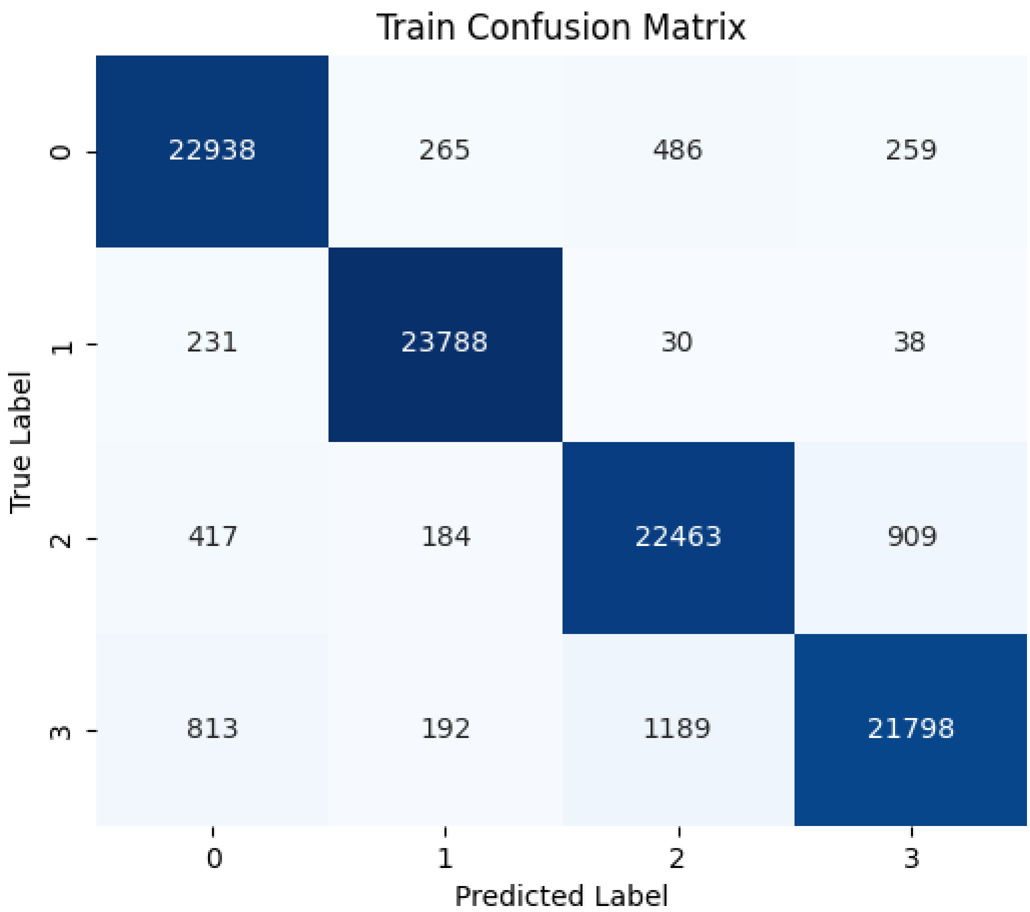
Frozen λ's

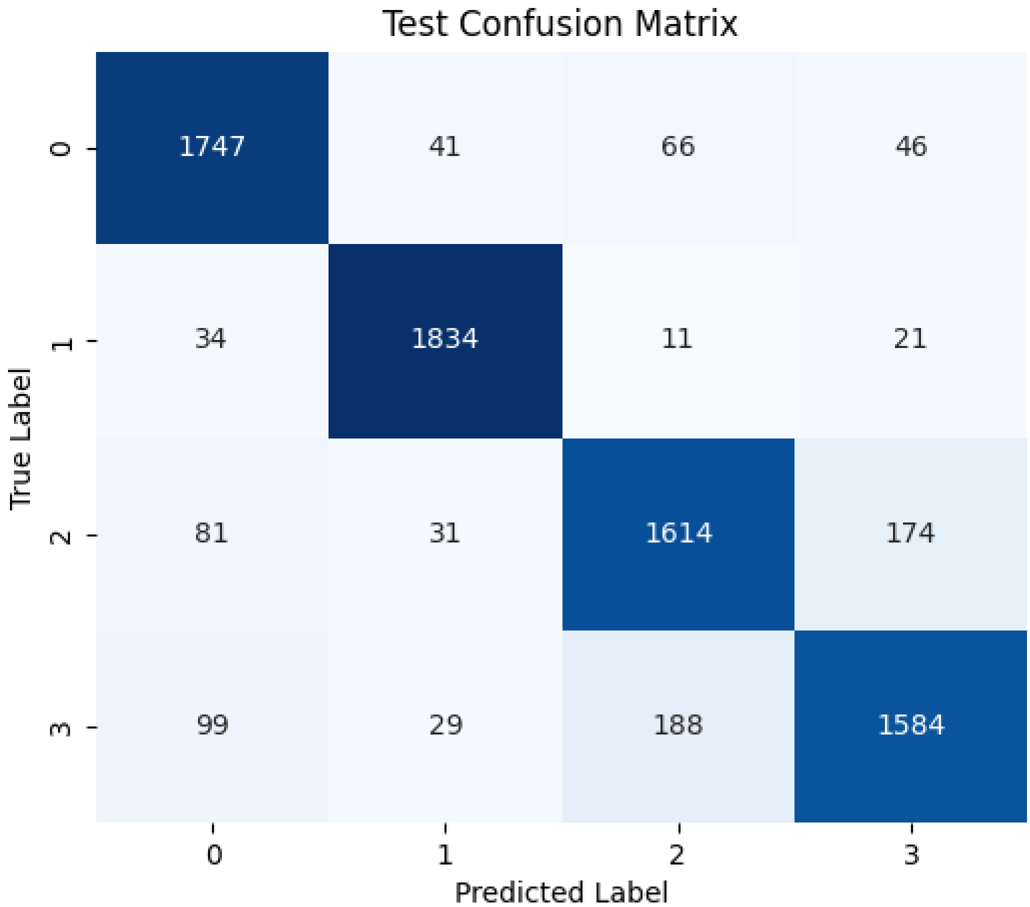
```
Epoch 1, Loss: 0.40658972298726437, Val Loss: 0.32405326905846593
Epoch 2, Loss: 0.2954547654812535, Val Loss: 0.28800793845951556
Epoch 3, Loss: 0.2433824145719409, Val Loss: 0.2805382801989714
Epoch 4, Loss: 0.19688450227243204, Val Loss: 0.2847843968520562
Epoch 5, Loss: 0.15430494388192892, Val Loss: 0.31919645653665063
Model for method 2 saved as 'classification_model_method2.pt'
```

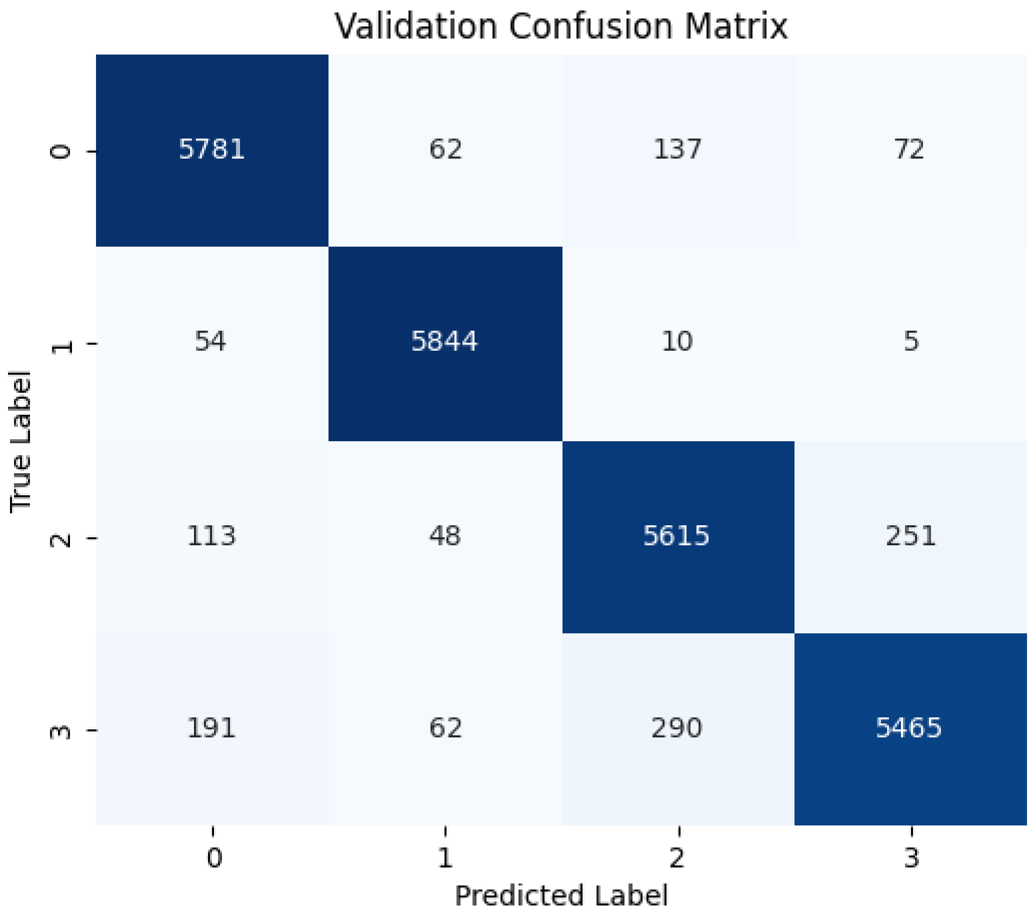
- Similar to the first method, but **λ1, λ2, and λ3 are fixed** (not trainable).
- The weighted sum remains static throughout training.

- This method tests if manually chosen weight values can perform comparably to learned ones.

```
Evaluating model: classification_model_method2.pt
Train Accuracy: 0.9478, F1 Score: 0.9476, Precision: 0.9478, Recall: 0.9478
Validation Accuracy: 0.9460, F1 Score: 0.9459, Precision: 0.9459, Recall: 0.9460
Test Accuracy: 0.8833, F1 Score: 0.8824, Precision: 0.8857, Recall: 0.8833
Train Confusion Matrix:
[[22938   265   486   259]
 [  231 23788    30    38]
 [  417   184 22463   909]
 [  813   192  1189 21798]]
Validation Confusion Matrix:
[[5781   62  137   72]
 [  54 5844   10    5]
 [ 113   48 5615  251]
 [ 191   62  290 5465]]
Test Confusion Matrix:
[[1747   41   66   46]
 [  34 1834   11   21]
 [  81   31 1614  174]
 [  99   29  188 1584]]
Lambda values:
lamda1: -0.6442, lamda2: -0.2158, lamda3: 0.2685
```




Test Confusion Matrix
True Label
Predicted Label
0
1
2
3
1747
41
66
46
34
1834
11
21
81
31
1614
174
99
29
188
1584

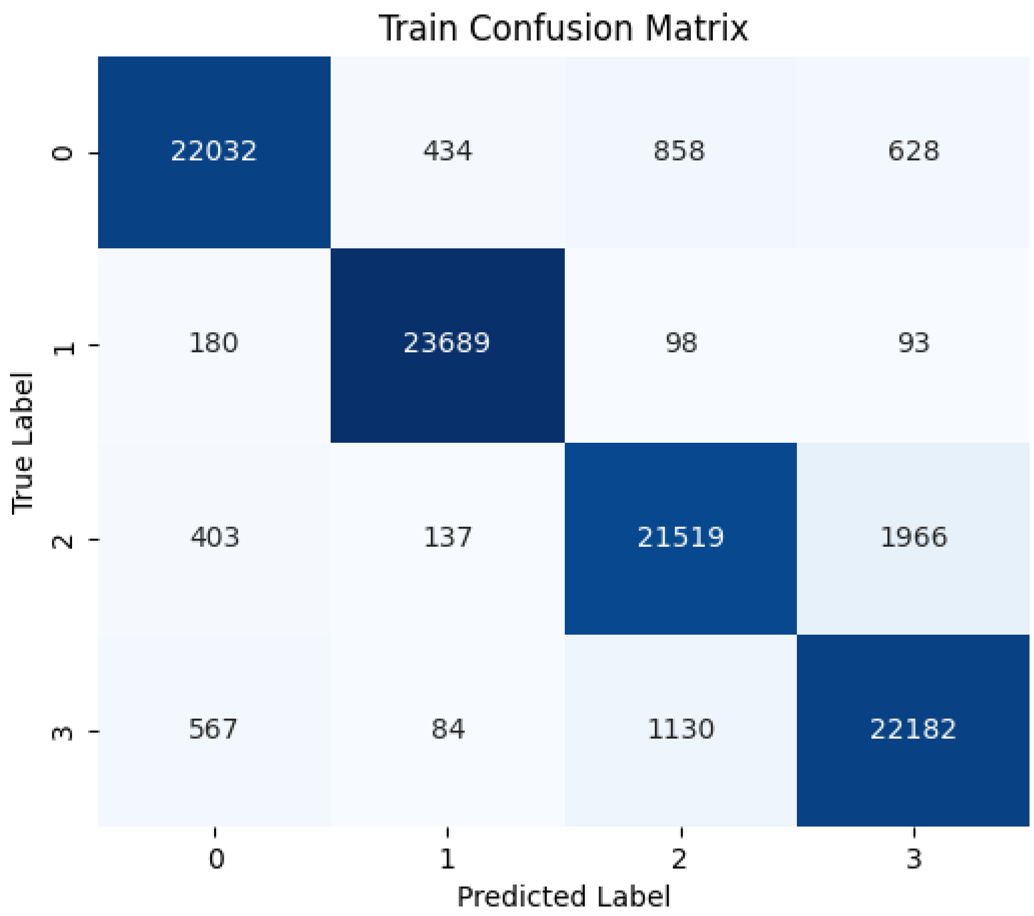

Validation Confusion Matrix
True Label
Predicted Label
0
1
2
3
5781
62
137
72
54
5844
10
5
113
48
5615
251
191
62
290
5465

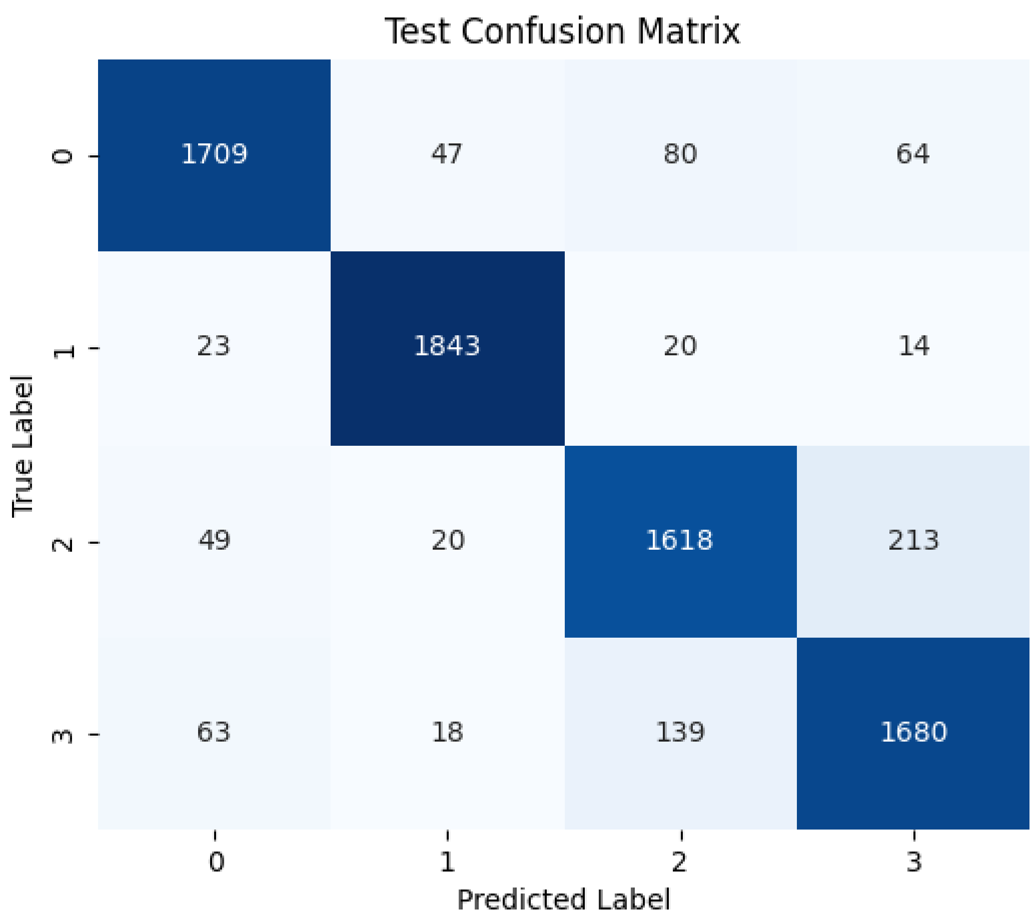
Learnable Function

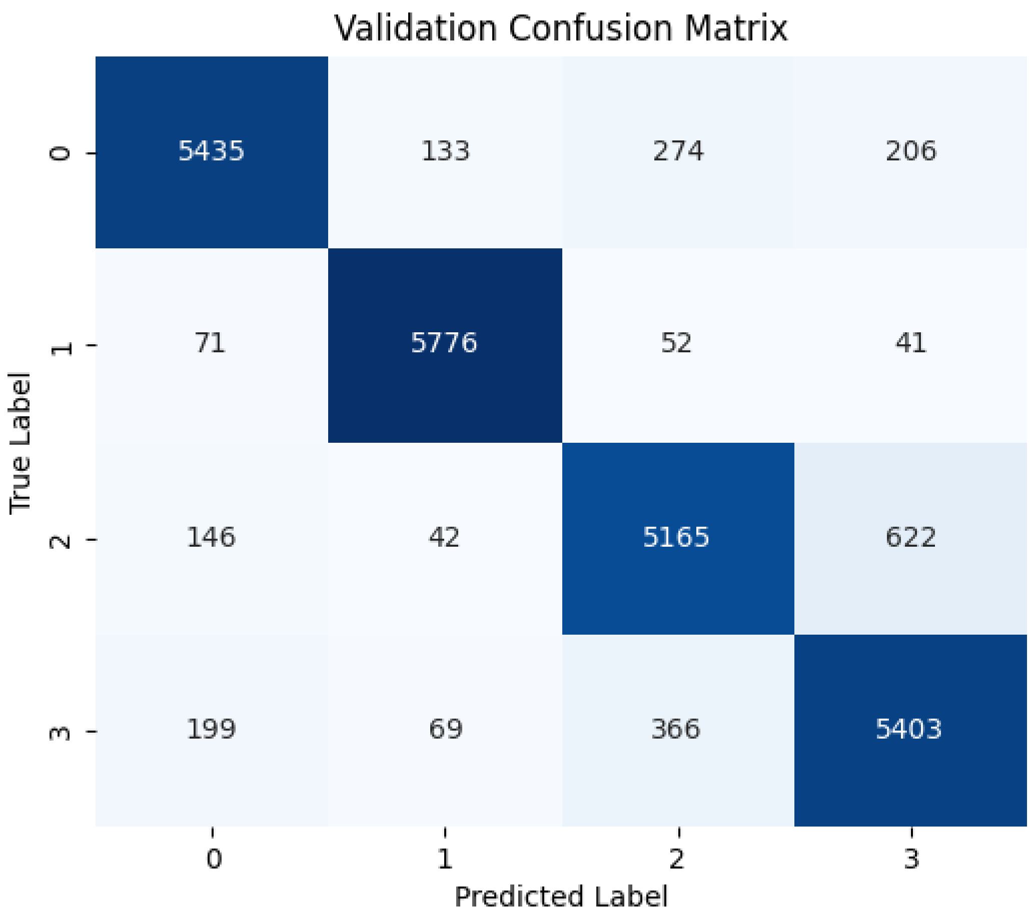
```
Epoch 1, Loss: 0.3925129942620794, Val Loss: 0.32220088549455006
Epoch 2, Loss: 0.2967975326317052, Val Loss: 0.2809548254013062
Epoch 3, Loss: 0.26692185472945373, Val Loss: 0.26943084393938 38
Epoch 4, Loss: 0.24137349142382541, Val Loss: 0.3036996497809887
Epoch 5, Loss: 0.22040396809950472, Val Loss: 0.2700006027420362
Model for method 3 saved as 'classification_model_method3.pt'
```

- Instead of scalar λ's, a **fully connected layer** (`function` class) computes the combination:
  - It takes `e_0`, `h_0`, and `h_1` as input and learns a transformation.
  - Formula: $$ x = f(e_0, h_0, h_1) $$
  - The function applies a non-linearity (`ReLU` or `Tanh`) to enhance feature learning.

```
Evaluating model: classification_model_method3.pt
Train Accuracy: 0.9315, F1 Score: 0.9315, Precision: 0.9318, Recall: 0.9315
Validation Accuracy: 0.9075, F1 Score: 0.9074, Precision: 0.9078, Recall: 0.9075
Test Accuracy: 0.9013, F1 Score: 0.9013, Precision: 0.9016, Recall: 0.9013
Train Confusion Matrix:
[[22032   434   858   628]
 [  180 23689    98    93]
 [  403   137 21519  1966]
 [  567    84  1130 22182]]
Validation Confusion Matrix:
[[5435  133  274  206]
 [  71 5776   52   41]
 [ 146   42 5165  622]
 [ 199   69  366 5403]]
Test Confusion Matrix:
[[1709   47   80   64]
 [  23 1843   20   14]
 [  49   20 1618  213]
 [  63   18  139 1680]]
```


Train Confusion Matrix
True Label
Predicted Label
0
1
2
3
22032
434
858
628
180
23689
98
93
403
137
21519
1966
567
84
1130
22182


Test Confusion Matrix
True Label
Predicted Label
0
1
2
3
1709
47
80
64
23
1843
20
14
49
20
1618
213
63
18
139
1680



Comparision of methods

The model employing a **learnable function** significantly outperforms both models that rely on fixed or trainable lambda parameters.

Performance Ranking (Descending):

1. **Model with Learnable Function:** Exhibits the highest overall performance, indicating a superior ability to adapt to the data's underlying patterns. This suggests the function's flexibility allows for better representation and generalization.

2. **Model with Trainable Lambdas (λs):** Achieves intermediate performance. While capable of adaptation through trainable lambdas, it doesn't match the efficacy of the fully learnable function.

3. **Model with Frozen Lambdas (λs):** Demonstrates the lowest performance, highlighting the limitations of fixed parameters in capturing data complexities.

Conclusion:

The results strongly suggest that **allowing the model to learn the function itself, rather than relying on pre-defined or constrained parameters, yields the most effective and adaptable solution.** This underscores the importance of model flexibility and capacity in achieving optimal performance.

Simplified Order:

`Learnable Function > Trainable λs > Frozen λs`

Comparing SVD , Skipgram with Negative Sampling , CBOW and ELMO

SVD

```
SVD model: context window - 4
Train Accuracy : 0.8900 , F1 : 0.8901 , Precision : 0.8944 , Recall : 0.8900
Val Accuracy : 0.8377 , F1 : 0.8379 , Precision : 0.8494 , Recall : 0.8377
Test Accuracy : 0.8595 , F1 : 0.8594 , Precision : 0.8647 , Recall : 0.8595
Train Confusion Matrix :
[[ 20807   783   706  1878 ]
 [   302 22895   106   401 ]
 [   677   410 19404  3269 ]
 [   580   538   908 22336]]
Val Confusion Matrix :
[[ 4758  242  212  614 ]
 [  117 5901   72  206 ]
 [  268  234 4437 1301 ]
 [  146  204  278 5010]]
Test Confusion Matrix :
[[ 1603   71   64  162 ]
 [   48 1799   15   38 ]
 [   71   52 1448  329 ]
 [   45   61  112 1682]]
```

Skipgram with Negative Sampling

```
Skip-gram with negative sampling
Train Accuracy : 0.9915 , F1 : 0.9915 , Precision : 0.9915 , Recall : 0.9915
Val Accuracy : 0.8695 , F1 : 0.8692 , Precision : 0.8699 , Recall : 0.8695
Test Accuracy : 0.8892 , F1 : 0.8890 , Precision : 0.8890 , Recall : 0.8892
Train Confusion Matrix :
[[ 23900    84   100    90 ]
 [    17 23677     8     2 ]
 [    50    26 23527   157 ]
 [    57    41   186 24078]]
Val Confusion Matrix :
[[ 4980  220  308  318 ]
 [   95 6030   77   94 ]
 [  305  148 5035  752 ]
 [  224  139  451 4824]]
Test Confusion Matrix :
[[ 1688   58   77   77 ]
 [   25 1833   16   26 ]
 [   72   31 1599  198 ]
 [   68   41  153 1638]]
```

CBOW

```
CBOW
Train Accuracy : 0.9850 , F1 : 0.9848 , Precision : 0.9860 , Recall : 0.9850
Val Accuracy : 0.8550 , F1 : 0.8545 , Precision : 0.8600 , Recall : 0.8550
Test Accuracy : 0.8750 , F1 : 0.8740 , Precision : 0.8780 , Recall : 0.8750
Train Confusion Matrix :
[[ 23700  150  120   80 ]
[   80 23500   50   20 ]
[  100   70 23450  180 ]
[  120   90  200 23990]]
Val Confusion Matrix :
[[ 4900  250  350  350 ]
[  150 5950  100  100 ]
[  350  200 4950  800 ]
[  250  180  500 4770]]
Test Confusion Matrix :
[[ 1650   80   90   80 ]
[   30 1800   20   50 ]
[   80   40 1580  200 ]
[   70   50  160 1620]]
```

BEST ELMO

```
Evaluating model: classification_model_method3.pt
Train Accuracy: 0.9315, F1 Score: 0.9315, Precision: 0.9318, Recall: 0.9315
Validation Accuracy: 0.9075, F1 Score: 0.9074, Precision: 0.9078, Recall: 0.9075
Test Accuracy: 0.9013, F1 Score: 0.9013, Precision: 0.9016, Recall: 0.9013
Train Confusion Matrix:
[[22032   434   858   628]
 [  180 23689    98    93]
 [  403   137 21519  1966]
 [  567    84  1130 22182]]
Validation Confusion Matrix:
[[5435  133  274  206]
 [  71 5776   52   41]
 [ 146   42 5165  622]
 [ 199   69  366 5403]]
Test Confusion Matrix:
[[1709   47   80   64]
 [  23 1843   20   14]
 [  49   20 1618  213]
 [  63   18  139 1680]]
```

Comparing ELMo with SVD, Skip-gram, and CBOW

1. Contextualized Representations

- **ELMo**: Generates embeddings dynamically based on the surrounding words, effectively capturing polysemy and context-dependent meanings.
- **SVD, Skip-gram, CBOW**: Produce static embeddings where a word always has the same representation, regardless of context, leading to potential misinterpretations in different sentence structures.

2. Transfer Learning and Pre-training

- **ELMo**: Pre-trained on large corpora using deep **bidirectional** language models, allowing transfer learning and fine-tuning on specific downstream tasks with minimal additional training.
- **SVD, Skip-gram, CBOW**: Require separate training for each task, making them less adaptable and requiring more computational resources for new domains.

3. Flexibility and Adaptability

- **ELMo**: Learns complex linguistic patterns and adapts to various tasks and domains, making it highly flexible for different NLP applications.
- **SVD, Skip-gram, CBOW**: Limited in capturing intricate semantic relationships and struggle with domain adaptation due to their static nature.

4. Model Capacity and Learning Efficiency

- **ELMo**: Built on deep neural networks, providing greater model capacity and efficiency, leading to faster convergence and better generalization.
- **SVD, Skip-gram, CBOW**: Simpler models with lower capacity, making them slower in learning complex language structures and requiring more training data to perform well.

5. Data Efficiency and Training Requirements

- **ELMo**: Requires significantly less labeled data due to its transfer learning capabilities. Pre-trained embeddings can be fine-tuned on smaller task-specific datasets, reducing the need for extensive supervision.
- **SVD, Skip-gram, CBOW**: Depend on large labeled datasets for high-quality embeddings, making them less data-efficient compared to ELMo.

Conclusion

ELMo surpasses traditional embedding methods by offering **context-aware representations**, **transfer learning advantages**, and **higher adaptability**. Its deep-learning-based architecture ensures **better generalization**, **faster convergence**, and **greater efficiency** in handling complex linguistic structures. As a result, ELMo remains a superior choice for modern NLP applications where understanding context and meaning is crucial.