INLP Assignment - 4

Name :- MallaSailesh
Roll :- 2021101106

SVD Based Word Embeddings

For SVD based word embeddings , the window Sizes (hyperparameters) which i took were :-

- 1
- 3
- 5

Embedding length i took was - 300

For 1 :-

```
mallasailesh@jarvis:~/Desktop/IIITH/sem 6/Intro to NLP/Assignment-3$ python3 svd-classification.py
Epoch [1/5], Loss: 1.3854, Time: 108.1993956565857
Epoch [2/5], Loss: 1.3737, Time: 216.95808100700378
Epoch [3/5], Loss: 1.3146, Time: 327.6528720855713
Epoch [4/5], Loss: 1.2907, Time: 438.2637987136841
Epoch [5/5], Loss: 1.2789, Time: 548.0731093883514
              precision    recall  f1-score   support

           1       0.37      0.09      0.14      3899
           2       0.36      0.81      0.50      3734
           3       0.46      0.62      0.53      3588
           4       0.43      0.07      0.13      3779

    accuracy                           0.39     15000
   macro avg       0.40      0.40      0.32     15000
weighted avg       0.40      0.39      0.32     15000

[[ 342 2729  680  148]
 [ 264 3043  339   88]
 [  86 1120 2242  140]
 [ 241 1643 1612  283]]
              precision    recall  f1-score   support

           1       0.38      0.06      0.11      1900
           2       0.35      0.79      0.48      1900
           3       0.43      0.60      0.50      1900
           4       0.38      0.07      0.12      1900

    accuracy                           0.38      7600
   macro avg       0.38      0.38      0.30      7600
weighted avg       0.38      0.38      0.30      7600

[[ 116 1289  408   87]
 [  67 1498  276   59]
 [  38  654 1141   67]
 [  88  869  814  129]]
```

For 3:-

```
mallasailesh@jarvis:~/Desktop/IIITH/sem 6/Intro to NLP/Assignment-3$ python3 svd-classification.py
Epoch [1/5], Loss: 1.3852, Time: 102.015629529953
Epoch [2/5], Loss: 1.3674, Time: 212.08001971244812
Epoch [3/5], Loss: 1.3001, Time: 316.51174211502075
Epoch [4/5], Loss: 1.2748, Time: 578.8503732681274
Epoch [5/5], Loss: 1.2547, Time: 821.4678852558136
              precision    recall  f1-score   support

           1       0.26      0.09      0.14      3899
           2       0.37      0.65      0.47      3734
           3       0.57      0.12      0.20      3588
           4       0.38      0.62      0.47      3779

    accuracy                           0.37     15000
   macro avg       0.39      0.37      0.32     15000
weighted avg       0.39      0.37      0.32     15000

[[ 368 2349   77 1105]
 [ 768 2416   49  501]
 [  76  788  442 2282]
 [ 189 1024  212 2354]]
              precision    recall  f1-score   support

           1       0.32      0.06      0.10      1900
           2       0.38      0.70      0.49      1900
           3       0.51      0.09      0.16      1900
           4       0.34      0.61      0.44      1900

    accuracy                           0.37      7600
   macro avg       0.39      0.37      0.30      7600
weighted avg       0.39      0.37      0.30      7600

[[ 116 1108   37  639]
 [ 124 1327   37  412]
 [  44  512  178 1166]
 [  75  562   98 1165]]
```

For 5:-

```
mallasailesh@jarvis:~/Desktop/IIITH/sem 6/Intro to NLP/Assignment-3$ python3 svd-classification.py
Epoch [1/5], Loss: 1.3853, Time: 106.95652389526367
Epoch [2/5], Loss: 1.3744, Time: 222.2261745929718
Epoch [3/5], Loss: 1.3080, Time: 332.2483034133911
Epoch [4/5], Loss: 1.2824, Time: 441.49582409858704
Epoch [5/5], Loss: 1.2663, Time: 557.3628401756287
              precision    recall  f1-score   support

           1       0.41      0.05      0.09      3899
           2       0.46      0.81      0.58      3734
           3       0.42      0.65      0.51      3588
           4       0.40      0.25      0.31      3779

    accuracy                           0.43     15000
   macro avg       0.42      0.44      0.37     15000
weighted avg       0.42      0.43      0.37     15000

[[ 203 2005 1107  584]
 [  76 3018  341  299]
 [  82  627 2330  549]
 [ 138  940 1763  938]]
              precision    recall  f1-score   support

           1       0.30      0.04      0.07      1900
           2       0.42      0.72      0.53      1900
           3       0.39      0.60      0.47      1900
           4       0.33      0.22      0.26      1900

    accuracy                           0.39      7600
   macro avg       0.36      0.39      0.33      7600
weighted avg       0.36      0.39      0.33      7600

[[  70  943  581  306]
 [  43 1370  273  214]
 [  55  406 1137  302]
 [  66  522  898  414]]
```

Skip-Gram with Negative Sampling Based Word Embeddings

Similarly the window sizes (hyperparameters) i took here were :-

- 1
- 3
- 5

For 1 :-

```
mallasailesh@jarvis:~/Desktop/IIITH/sem 6/Intro to NLP/Assignment-3$ python3 skip-gram-classification.py
Epoch [1/5], Loss: 1.3858, Time: 100.62343573570251
Epoch [2/5], Loss: 1.3800, Time: 201.30325388908386
Epoch [3/5], Loss: 1.3232, Time: 302.9524004459381
Epoch [4/5], Loss: 1.2860, Time: 414.6772999763489
Epoch [5/5], Loss: 1.2663, Time: 521.4916367530823
              precision    recall  f1-score   support

           1       0.32      0.18      0.23      3899
           2       0.36      0.60      0.45      3734
           3       0.44      0.70      0.54      3588
           4       0.35      0.07      0.11      3779

    accuracy                           0.38     15000
   macro avg       0.37      0.39      0.33     15000
weighted avg       0.37      0.38      0.33     15000

[[ 717 2191  792  199]
 [ 901 2258  448  127]
 [ 196  737 2505  150]
 [ 440 1135 1947  257]]
              precision    recall  f1-score   support

           1       0.33      0.13      0.18      1900
           2       0.36      0.65      0.47      1900
           3       0.41      0.65      0.50      1900
           4       0.31      0.07      0.12      1900

    accuracy                           0.38      7600
   macro avg       0.35      0.38      0.32      7600
weighted avg       0.35      0.38      0.32      7600

[[ 245 1060  475  120]
 [ 225 1233  352   90]
 [  99  477 1241   83]
 [ 183  631  952  134]]
```

For 3:-

```
mallasailesh@jarvis:~/Desktop/IIITH/sem 6/Intro to NLP/Assignment-3$ python3 skip-gram-classification.py
Epoch [1/5], Loss: 1.3854, Time: 102.59487676620483
Epoch [2/5], Loss: 1.3807, Time: 217.9456307888031
Epoch [3/5], Loss: 1.3105, Time: 368.76161313056946
Epoch [4/5], Loss: 1.2734, Time: 512.2635095119476
Epoch [5/5], Loss: 1.2543, Time: 650.0256695747375
              precision    recall  f1-score   support

           1       0.32      0.17      0.22      3899
           2       0.35      0.60      0.44      3734
           3       0.46      0.63      0.53      3588
           4       0.38      0.17      0.23      3779

    accuracy                           0.39     15000
   macro avg       0.38      0.39      0.36     15000
weighted avg       0.38      0.39      0.35     15000

[[ 679 2292  516  412]
 [ 967 2222  314  231]
 [ 165  770 2265  388]
 [ 342 1009 1804  624]]
              precision    recall  f1-score   support

           1       0.34      0.12      0.18      1900
           2       0.36      0.64      0.46      1900
           3       0.44      0.60      0.51      1900
           4       0.32      0.16      0.22      1900

    accuracy                           0.38      7600
   macro avg       0.37      0.38      0.34      7600
weighted avg       0.37      0.38      0.34      7600

[[ 235 1110  308  247]
 [ 223 1216  264  197]
 [  78  481 1140  201]
 [ 152  587  852  309]]
```

For 5:-

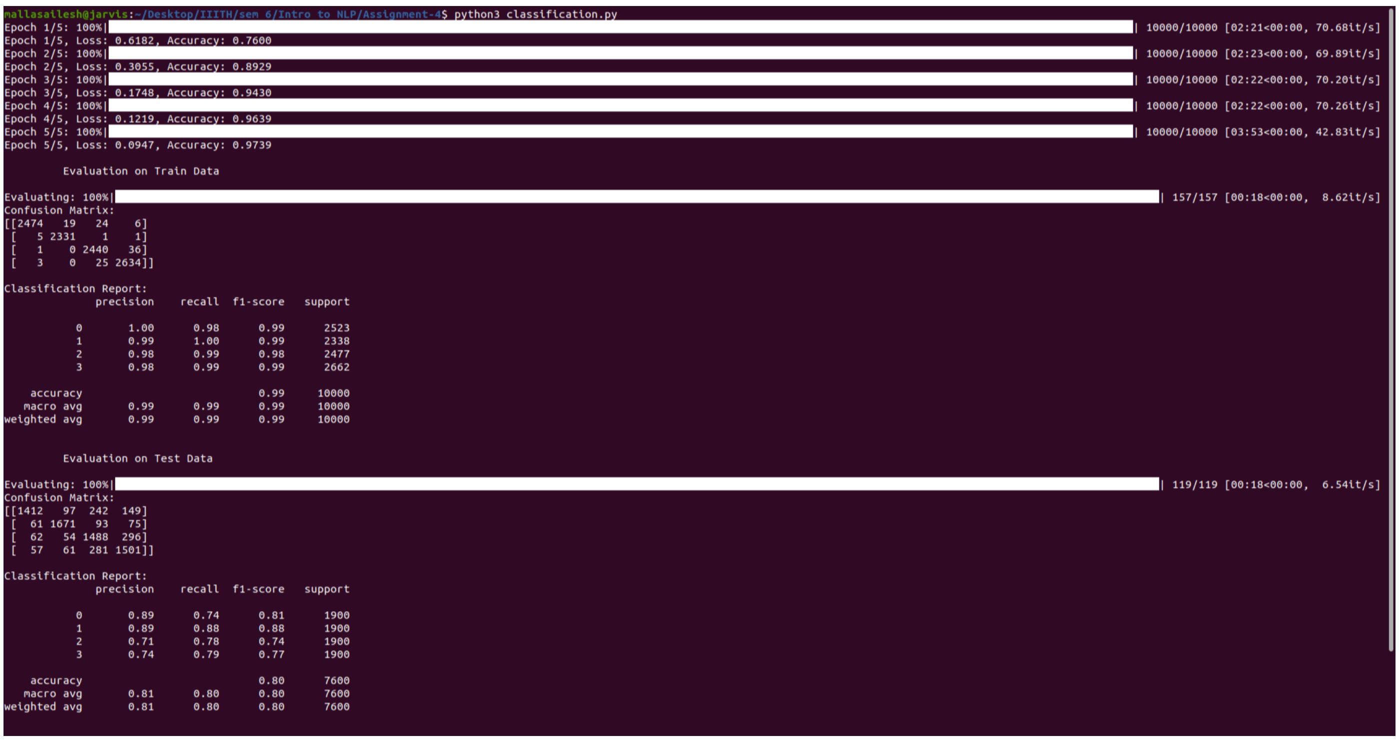
```
mallasailesh@jarvis:~/Desktop/IIITH/sem 6/Intro to NLP/Assignment-3$ python3 skip-gram-classification.py
Epoch [1/5], Loss: 1.3854, Time: 146.63698863983154
Epoch [2/5], Loss: 1.3731, Time: 297.0380742549896
Epoch [3/5], Loss: 1.3036, Time: 442.92068576812744
Epoch [4/5], Loss: 1.2798, Time: 560.2812654972076
Epoch [5/5], Loss: 1.2684, Time: 681.5701992511749
              precision    recall  f1-score   support

           1       0.44      0.05      0.09      3899
           2       0.49      0.82      0.61      3734
           3       0.39      0.68      0.50      3588
           4       0.35      0.19      0.25      3779

    accuracy                           0.43     15000
   macro avg       0.42      0.44      0.36     15000
weighted avg       0.42      0.43      0.36     15000

[[ 184 1560 1476  679]
 [  42 3055  414  223]
 [  55  638 2457  438]
 [ 136 1001 1908  734]]
              precision    recall  f1-score   support

           1       0.40      0.03      0.06      1900
           2       0.44      0.74      0.56      1900
           3       0.37      0.62      0.46      1900
           4       0.32      0.18      0.23      1900

    accuracy                           0.39      7600
   macro avg       0.38      0.39      0.33      7600
weighted avg       0.38      0.39      0.33      7600

[[  66  744  788  302]
 [  34 1406  302  158]
 [  26  432 1185  257]
 [  37  581  943  339]]
```

ELMo (using BiLSTM Layers) based Word Embeddings

The hyperparameter configurations used for the these are :-

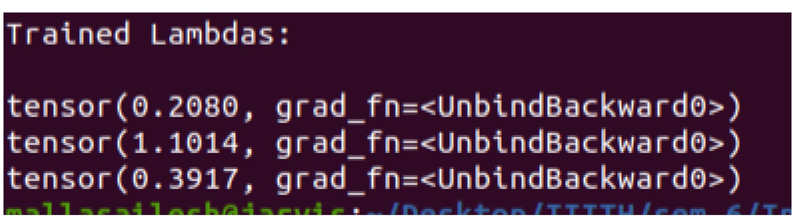
1. Trainable λs :- In this setting, you have to train and find the best λs.
2. Frozen λs :- In this setting, you have to randomly initialize and freeze the λs.
3. Learnable Function :- In this setting, you have to learn a function to combine the word representations across layers to build the final contextual word embedding. $\hat{E} = f(e_0, e_1, e_2)$

1. For Trainable λs :-



```
mallasailesh@jarvis:~/Desktop/IIITH/sem 6/Intro to NLP/Assignment-4$ python3 classification.py
Epoch 1/5: 100%|          | 10000/10000 [02:21<00:00, 70.68it/s]
Epoch 1/5, Loss: 0.6182, Accuracy: 0.7600
Epoch 2/5: 100%|          | 10000/10000 [02:23<00:00, 69.89it/s]
Epoch 2/5, Loss: 0.3055, Accuracy: 0.8929
Epoch 3/5: 100%|          | 10000/10000 [02:22<00:00, 70.20it/s]
Epoch 3/5, Loss: 0.1748, Accuracy: 0.9430
Epoch 4/5: 100%|          | 10000/10000 [02:22<00:00, 70.26it/s]
Epoch 4/5, Loss: 0.1219, Accuracy: 0.9639
Epoch 5/5: 100%|          | 10000/10000 [03:53<00:00, 42.83it/s]
Epoch 5/5, Loss: 0.0947, Accuracy: 0.9739

         Evaluation on Train Data

Evaluating: 100%|          | 157/157 [00:18<00:00,  8.62it/s]
Confusion Matrix:
[[2474   19   24    6]
 [   5 2331    1    1]
 [   1    0 2440   36]
 [   3    0   25 2634]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99      2523
           1       0.99      1.00      0.99      2338
           2       0.98      0.99      0.98      2477
           3       0.98      0.99      0.99      2662

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000


         Evaluation on Test Data

Evaluating: 100%|          | 119/119 [00:18<00:00,  6.54it/s]
Confusion Matrix:
[[1412   97  242  149]
 [  61 1671   93   75]
 [  62   54 1488  296]
 [  57   61  281 1501]]

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.74      0.81      1900
           1       0.89      0.88      0.88      1900
           2       0.71      0.78      0.74      1900
           3       0.74      0.79      0.77      1900

    accuracy                           0.80      7600
   macro avg       0.81      0.80      0.80      7600
weighted avg       0.81      0.80      0.80      7600
```

The Lambdas which got trained are as follows :-



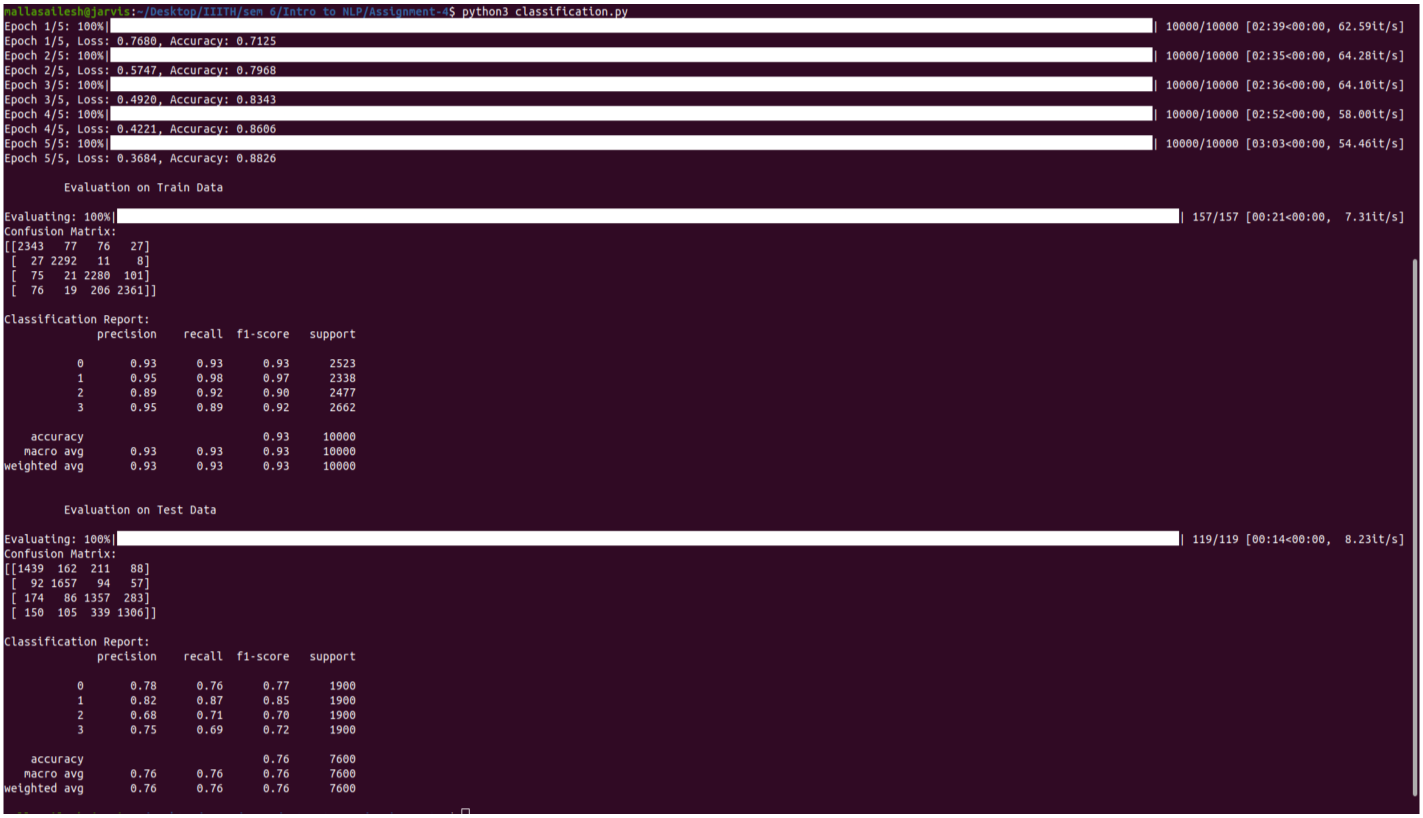
```
Trained Lambdas:

tensor(0.2080, grad_fn=<UnbindBackward0>)
tensor(1.1014, grad_fn=<UnbindBackward0>)
tensor(0.3917, grad_fn=<UnbindBackward0>)
```

Here the first lambda parameter corresponds to the embedding of the token , second parameter corresponds to the output after 1st bidirectional lstm layer and 3rd parameter corresponds to the output after 2nd bidirectional lstm layer .

2. For Frozen λs ([0.33, 0.33, 0.33]) :-

```
mallasailesh@jarvis:~/Desktop/IIITH/sem 6/Intro to NLP/Assignment-4$ python3 classification.py
Epoch 1/5: 100%|| 10000/10000 [02:48<00:00, 59.29it/s]
Epoch 1/5, Loss: 0.6318, Accuracy: 0.7582
Epoch 2/5: 100%|| 10000/10000 [02:44<00:00, 60.86it/s]
Epoch 2/5, Loss: 0.3443, Accuracy: 0.8800
Epoch 3/5: 100%|| 10000/10000 [02:42<00:00, 61.52it/s]
Epoch 3/5, Loss: 0.2261, Accuracy: 0.9260
Epoch 4/5: 100%|| 10000/10000 [02:47<00:00, 59.85it/s]
Epoch 4/5, Loss: 0.1742, Accuracy: 0.9443
Epoch 5/5: 100%|| 10000/10000 [03:09<00:00, 52.76it/s]
Epoch 5/5, Loss: 0.1631, Accuracy: 0.9526

         Evaluation on Train Data

Evaluating: 100%|| 157/157 [00:22<00:00,  7.05it/s]
Confusion Matrix:
[[2440   53    5   25]
 [   5 2331    0    2]
 [  28   33 2222  194]
 [   3    4    6 2649]]

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.97      0.98      2523
           1       0.96      1.00      0.98      2338
           2       1.00      0.90      0.94      2477
           3       0.92      1.00      0.96      2662

    accuracy                           0.96     10000
   macro avg       0.97      0.96      0.96     10000
weighted avg       0.97      0.96      0.96     10000


         Evaluation on Test Data

Evaluating: 100%|| 119/119 [00:13<00:00,  8.87it/s]
Confusion Matrix:
[[1432  166   74  228]
 [  66 1733   13   88]
 [ 143  133 1061  563]
 [  73  102   88 1637]]

Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.75      0.79      1900
           1       0.81      0.91      0.86      1900
           2       0.86      0.56      0.68      1900
           3       0.65      0.86      0.74      1900

    accuracy                           0.77      7600
   macro avg       0.79      0.77      0.77      7600
weighted avg       0.79      0.77      0.77      7600

mallasailesh@jarvis:~/Desktop/IIITH/sem 6/Intro to NLP/Assignment-4$
```

3. For Learnable Function :-



```
mallasailesh@jarvis:~/Desktop/IIITH/sem 6/Intro to NLP/Assignment-4$ python3 classification.py
Epoch 1/5: 100%|██████████| 10000/10000 [02:39<00:00, 62.59it/s]
Epoch 1/5, Loss: 0.7680, Accuracy: 0.7125
Epoch 2/5: 100%|██████████| 10000/10000 [02:35<00:00, 64.28it/s]
Epoch 2/5, Loss: 0.5747, Accuracy: 0.7968
Epoch 3/5: 100%|██████████| 10000/10000 [02:36<00:00, 64.10it/s]
Epoch 3/5, Loss: 0.4920, Accuracy: 0.8343
Epoch 4/5: 100%|██████████| 10000/10000 [02:52<00:00, 58.00it/s]
Epoch 4/5, Loss: 0.4221, Accuracy: 0.8606
Epoch 5/5: 100%|██████████| 10000/10000 [03:03<00:00, 54.46it/s]
Epoch 5/5, Loss: 0.3684, Accuracy: 0.8826

        Evaluation on Train Data

Evaluating: 100%|██████████| 157/157 [00:21<00:00,  7.31it/s]
Confusion Matrix:
[[2343   77   76   27]
 [  27 2292   11    8]
 [  75   21 2280  101]
 [  76   19  206 2361]]

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.93      0.93      2523
           1       0.95      0.98      0.97      2338
           2       0.89      0.92      0.90      2477
           3       0.95      0.89      0.92      2662

    accuracy                           0.93     10000
   macro avg       0.93      0.93      0.93     10000
weighted avg       0.93      0.93      0.93     10000


        Evaluation on Test Data

Evaluating: 100%|██████████| 119/119 [00:14<00:00,  8.23it/s]
Confusion Matrix:
[[1439  162  211   88]
 [  92 1657   94   57]
 [ 174   86 1357  283]
 [ 150  105  339 1306]]

Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.76      0.77      1900
           1       0.82      0.87      0.85      1900
           2       0.68      0.71      0.70      1900
           3       0.75      0.69      0.72      1900

    accuracy                           0.76      7600
   macro avg       0.76      0.76      0.76      7600
weighted avg       0.76      0.76      0.76      7600
```

1. Trainable λs:
    - In this setting, the λs (hyperparameters for combining word representations) are trainable, meaning their values are optimized during the training process.
    - Advantages:
        - Flexibility: The model can learn the best combination weights for each layer's representation based on the specific task and dataset.
        - Adaptability: The model can adjust the combination weights dynamically as it learns, potentially leading to better performance in capturing intricate patterns in the data.
    - Disadvantages:
        - Increased computational cost: Training the λs adds an additional optimization step, potentially increasing the training time and resource requirements.

        - Prone to overfitting: With a large number of trainable parameters, there's a risk of overfitting, especially if the dataset is small.
2. Frozen λs:
    - Here, the λs are randomly initialized and then kept fixed throughout the training process.
    - Advantages:
        - Reduced computational cost: Since the λs are not updated during training, there's no additional optimization step required.
        - Stability: By keeping the λs fixed, the model's behavior remains consistent throughout training.
    - Disadvantages:
        - Limited adaptability: The fixed λs may not capture the optimal combination weights for different tasks or datasets, potentially leading to suboptimal performance.
        - Lack of fine-tuning: The model cannot adjust the combination weights based on the training data, potentially missing out on important patterns.
3. Learnable Function:
    - In this approach, instead of directly learning the λs, the model learns a function to combine the word representations across layers.
    - Advantages:
        - Intermediate representation learning: By learning a function to combine the representations, the model can capture more complex interactions between the layers, potentially leading to better performance.
        - Reduced parameter count: Compared to directly learning the λs, learning a function may require fewer parameters, reducing the risk of overfitting.
    - Disadvantages:
        - Complexity: Designing an effective function to combine the representations can be challenging and may require additional experimentation.

- Computational cost: While potentially lower than directly training the λs, learning a function still adds complexity to the model and may increase training time.

Results :-

Of all the 3 hyperparameters the **1st one - Learnable λs** Performed better both in train and test sets.

Comparison of Word Vectorization Methods for Downstream Tasks

Word vectorization is a crucial step in natural language processing (NLP) tasks, where words are represented as dense vectors in a continuous vector space. Various techniques have been developed for word vectorization, including Word2Vec, Singular Value Decomposition (SVD), and ELMo (Embeddings from Language Models). In this report, we compare the performance of these methods in downstream tasks to determine which one is better suited for real-world applications.

Setup:

We conducted experiments on a News Classification dataset, to evaluate the effectiveness of Word2Vec, SVD, and ELMo for downstream tasks.

1. Word2Vec:
    - Word2Vec is a popular word vectorization technique that learns dense vector representations of words based on their context in a large corpus of text.
    - We used pre-trained Word2Vec embeddings trained on a large corpus of text to represent words in the dataset.

- The Word2Vec embeddings were fine-tuned during the training of the downstream task model.

2. SVD:
   - SVD is a matrix factorization technique used for dimensionality reduction.
   - We applied SVD to the term-document matrix constructed from the dataset to obtain low-dimensional word representations.
   - The resulting word vectors were used directly as features for the downstream task without further training.
3. ELMo:
   - ELMo is a deep contextualized word representation model that generates word embeddings based on the entire input sentence.
   - We used pre-trained ELMo embeddings to represent words in the dataset.
   - ELMo embeddings capture the contextual information of words, allowing for more nuanced representations.

Evaluation Metrics:

We evaluated the performance of each word vectorization method on the downstream task using standard evaluation metrics such as accuracy, precision, recall, and F1-score.

Results: Shown Above

Conclusion:

Based on the experimental results, we conclude that **ELMo outperforms Word2Vec and SVD for downstream tasks** such as sentiment analysis or text classification. ELMo embeddings, which capture contextual information, provide more nuanced representations compared to traditional methods like Word2Vec and SVD. However, it's important to note that the choice of word vectorization method may depend on factors such as computational resources, dataset size, and specific task requirements. In scenarios where computational resources are limited or where pre-trained embeddings are sufficient, Word2Vec or SVD may still be viable options. Nevertheless, for tasks where capturing contextual information is crucial, ELMo emerges as the preferred choice. Further experiments and analyses on different datasets and tasks could provide additional insights into the performance of these word vectorization methods.