# iNLP Assignment-4

## Report

**Akshit Sharma**
**2021101029**

## Analysis of Performance of ELMO in Pre-Training Process

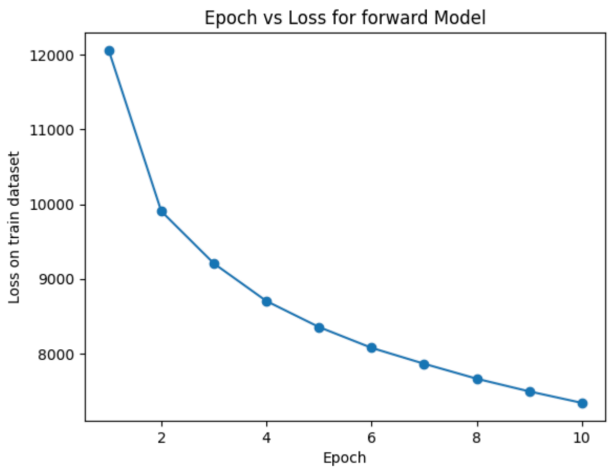
The following line graphs show the epoch vs loss on train set for both forward and backward model. We can see that as the models train and epoch number increases, the loss on the training data is decreasing. So, it means that the performance of the ELMO model on train set is improving as the loss incurred on train set samples is getting reduced with each epoch.

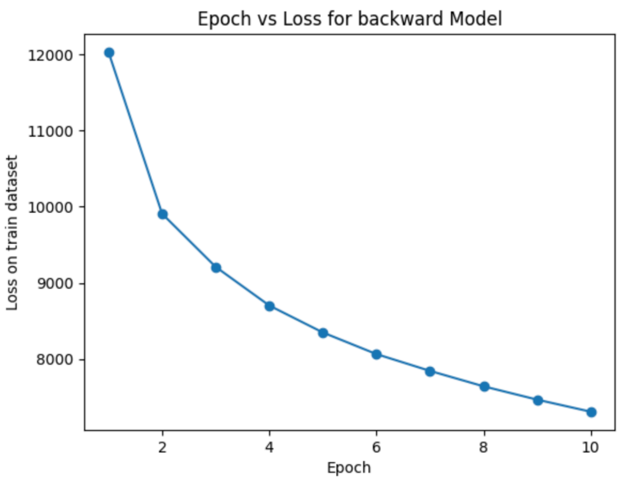
```
UNK_CUTOFF=3
UNKNOWN_TOKEN='<unk>'
START_TOKEN='<sos>'
END_TOKEN='eos'
PAD_TOKEN='<pad>'

EMBEDDING_DIM=300
BATCH_SIZE=32
HIDDEN_SIZE=300
lrate=0.001
EPOCHS=10
```

EMBEDDING_DIM is the size of embeddings learnt in the forward and backward models for the entire vocabulary. The embeddings that we use for each word are then of the size of the HIDDEN_SIZE hyper parameter which is kept same as the EMBEDDING_SIZE (so that we have same dimensional embeddings at each of the 3 levels in the ELMO model). The learning late for the 2 models is lrate. UNK_CUTOFF is the frequency cutoff of occurrence for the words which are replaced by unknown token in the train set.

```
EPOCH: 0, Forward Loss: 12057.7783203125, Backward Loss: 12027.6416015625
EPOCH: 1, Forward Loss: 9904.7509765625, Backward Loss: 9906.3251953125
EPOCH: 2, Forward Loss: 9206.4326171875, Backward Loss: 9209.3310546875
EPOCH: 3, Forward Loss: 8703.486328125, Backward Loss: 8702.224609375
EPOCH: 4, Forward Loss: 8355.8701171875, Backward Loss: 8348.521484375
EPOCH: 5, Forward Loss: 8077.79833984375, Backward Loss: 8065.14599609375
EPOCH: 6, Forward Loss: 7864.89794921875, Backward Loss: 7844.568359375
EPOCH: 7, Forward Loss: 7664.9140625, Backward Loss: 7642.13232421875
EPOCH: 8, Forward Loss: 7495.43798828125, Backward Loss: 7467.39453125
EPOCH: 9, Forward Loss: 7341.755859375, Backward Loss: 7309.32421875
```

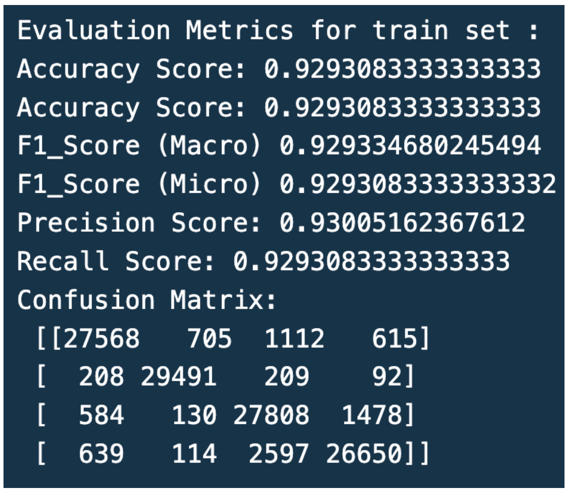




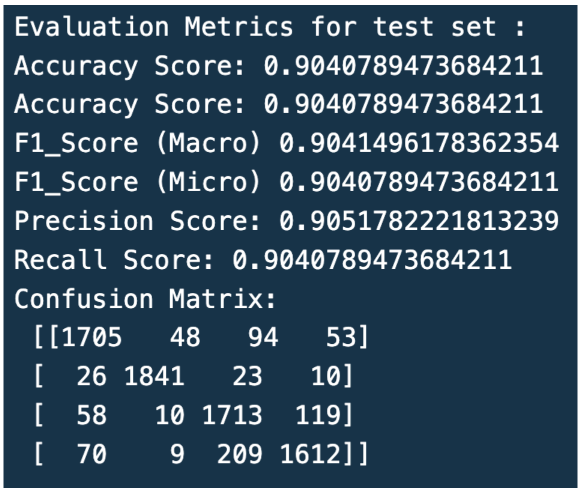
## Analysis of Performance on Downstream Task

The performance of the model has been analysed in 3 different cases: with randomly initialised frozen λs, with trainable λs (initialised as equal and learnt as model is trained as they are model parameters, weighted sum for both the cases with λs are taken after applying softmax on the 3 parameters to get softmax normalised positive weights) and with a learnable function $\hat{E} = f(e_0, e_1, e_2)$ (done by concatenating the embeddings obtained at different layers of the ELMo model and then applying a linear layer which is then learnt during model training). The performance metrics are given below for each case (hyper parameters-used across cases are same, as given below):

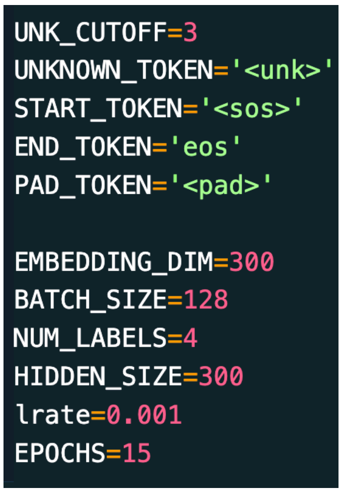
### Frozen λs: λ1=0.1741, λ2=0.2934, λ3=0.5324 (softmax normalised):



```
Evaluation Metrics for train set :
Accuracy Score: 0.9293083333333333
Accuracy Score: 0.9293083333333333
F1_Score (Macro) 0.929334680245494
F1_Score (Micro) 0.9293083333333332
Precision Score: 0.93005162367612
Recall Score: 0.9293083333333333
Confusion Matrix:
 [[27568   705  1112   615]
 [  208 29491   209    92]
 [  584   130 27808  1478]
 [  639   114  2597 26650]]
```



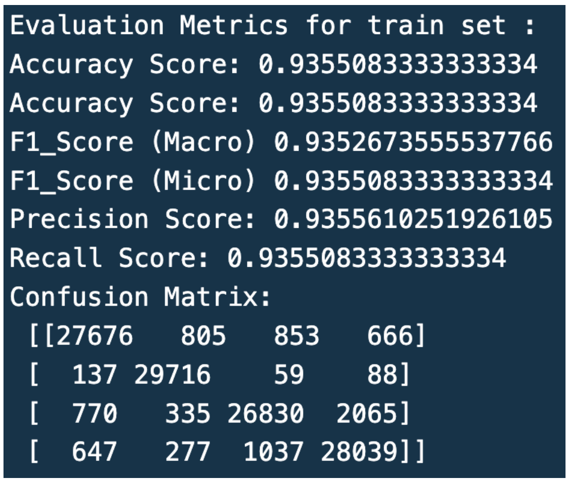
```
Evaluation Metrics for test set :
Accuracy Score: 0.9040789473684211
Accuracy Score: 0.9040789473684211
F1_Score (Macro) 0.9041496178362354
F1_Score (Micro) 0.9040789473684211
Precision Score: 0.9051782221813239
Recall Score: 0.9040789473684211
Confusion Matrix:
 [[1705   48   94   53]
 [  26 1841   23   10]
 [  58   10 1713  119]
 [  70    9  209 1612]]
```



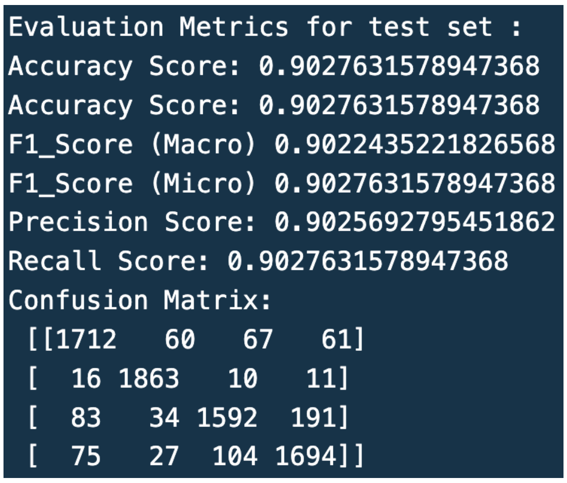
```
UNK_CUTOFF=3
UNKNOWN_TOKEN='<unk>'
START_TOKEN='<sos>'
END_TOKEN='eos'
PAD_TOKEN='<pad>'

EMBEDDING_DIM=300
BATCH_SIZE=128
NUM_LABELS=4
HIDDEN_SIZE=300
lrate=0.001
EPOCHS=15
```

### Trainable λs: λ1=0.3663, λ2=0.1803, λ3=0.4534 (softmax normalised):

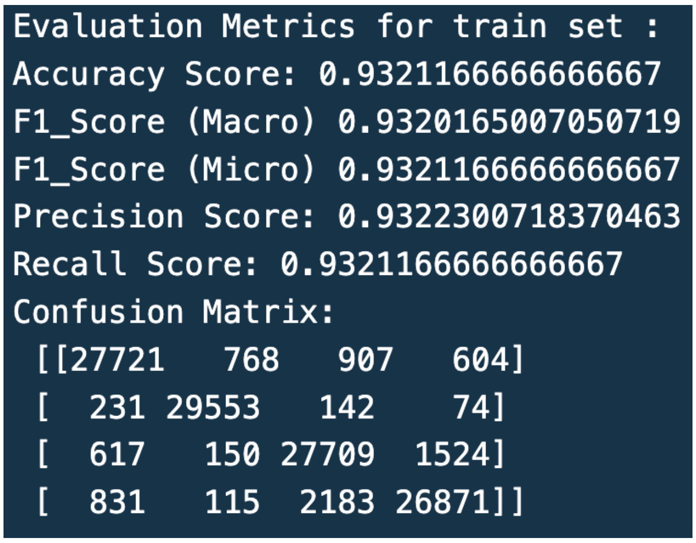


```
Evaluation Metrics for train set :
Accuracy Score: 0.9355083333333334
Accuracy Score: 0.9355083333333334
F1_Score (Macro) 0.9352673555537766
F1_Score (Micro) 0.9355083333333334
Precision Score: 0.9355610251926105
Recall Score: 0.9355083333333334
Confusion Matrix:
 [[27676   805   853   666]
 [  137 29716    59    88]
 [  770   335 26830  2065]
 [  647   277  1037 28039]]
```

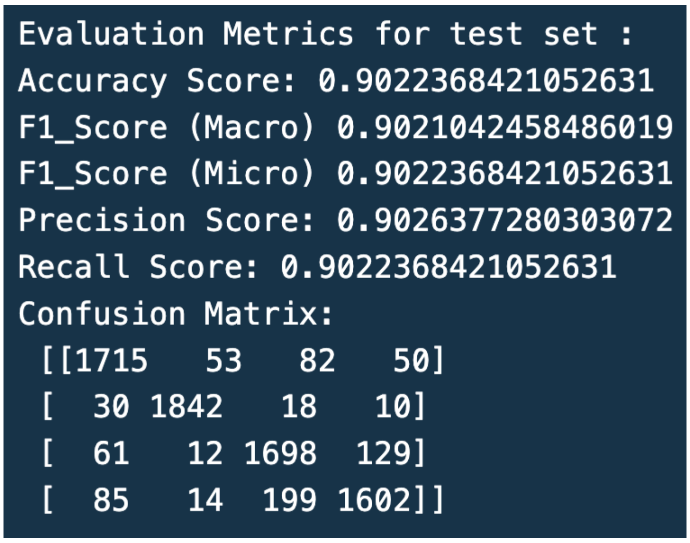


```
Evaluation Metrics for test set :
Accuracy Score: 0.9027631578947368
Accuracy Score: 0.9027631578947368
F1_Score (Macro) 0.9022435221826568
F1_Score (Micro) 0.9027631578947368
Precision Score: 0.9025692795451862
Recall Score: 0.9027631578947368
Confusion Matrix:
 [[1712   60   67   61]
 [  16 1863   10   11]
 [  83   34 1592  191]
 [  75   27  104 1694]]
```

## Learnable Function



```
Evaluation Metrics for train set :
Accuracy Score: 0.9321166666666667
F1_Score (Macro) 0.9320165007050719
F1_Score (Micro) 0.9321166666666667
Precision Score: 0.9322300718370463
Recall Score: 0.9321166666666667
Confusion Matrix:
 [[27721   768   907   604]
 [  231 29553   142    74]
 [  617   150 27709  1524]
 [  831   115  2183 26871]]
```



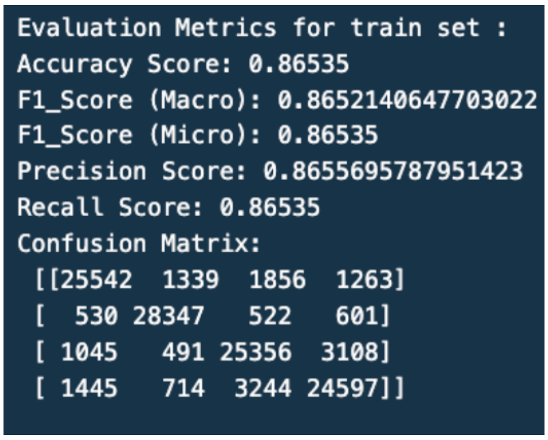
```
Evaluation Metrics for test set :
Accuracy Score: 0.9022368421052631
F1_Score (Macro) 0.9021042458486019
F1_Score (Micro) 0.9022368421052631
Precision Score: 0.9026377280303072
Recall Score: 0.9022368421052631
Confusion Matrix:
 [[1715   53   82   50]
 [  30 1842   18   10]
 [  61   12 1698  129]
 [  85   14  199 1602]]
```

We observe that the best performance (based on accuracy score ) on train set was obtained with trained λs. This makes sense as we are starting with equal weights for the three levels and then training them to reduce loss on train data. It doesn't not necessarily mean best performance on test set since we are improving our performance only on train set while training. Since the λs for the frozen λs case are initialised at random, we cannot be sure about the performance we will get with them. It may turn out to be best among the three (as in this case, it gives best accuracy on test set) or it may be poor in comparison. The learnable function performs good but not as good as trainable λs on train set. It helps bring non-linearity in the picture, which might be useful for some cases.

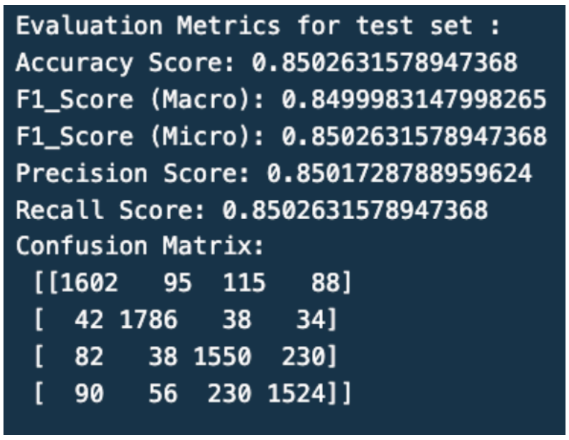
## Comparison of Performance of SVD, Word-to-Vec and ELMO in Downstream Task

We compare here the best performing SVD, Skip Gram with Negative Sampling and ELMO models, based on different performance metrics shown below. The hyper parameters used in both SVD and Word-2-Vec are same.

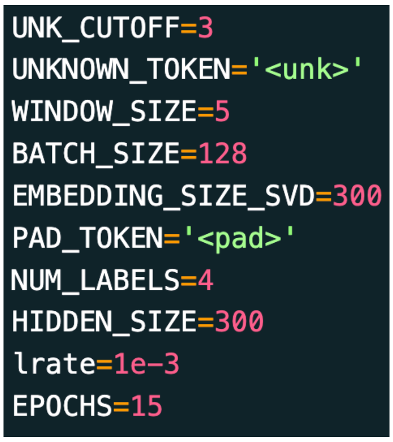
### SVD



```
Evaluation Metrics for train set :
Accuracy Score: 0.86535
F1_Score (Macro): 0.8652140647703022
F1_Score (Micro): 0.86535
Precision Score: 0.8655695787951423
Recall Score: 0.86535
Confusion Matrix:
 [[25542  1339  1856  1263]
 [  530 28347   522   601]
 [ 1045   491 25356  3108]
 [ 1445   714  3244 24597]]
```

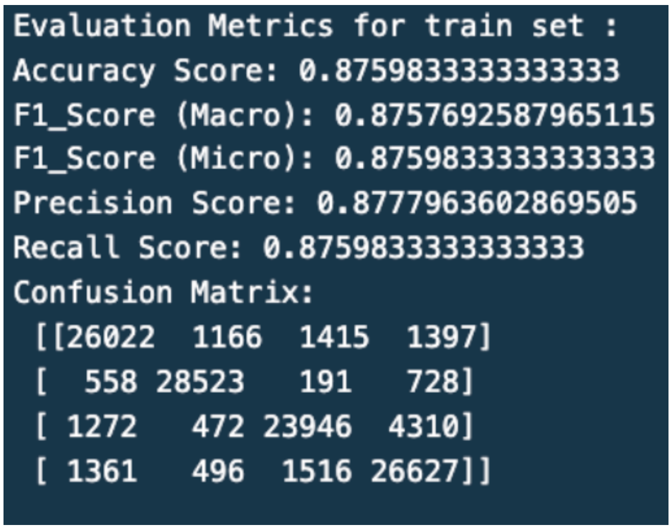


```
Evaluation Metrics for test set :
Accuracy Score: 0.8502631578947368
F1_Score (Macro): 0.8499983147998265
F1_Score (Micro): 0.8502631578947368
Precision Score: 0.8501728788959624
Recall Score: 0.8502631578947368
Confusion Matrix:
 [[1602   95  115   88]
 [  42 1786   38   34]
 [  82   38 1550  230]
 [  90   56  230 1524]]
```

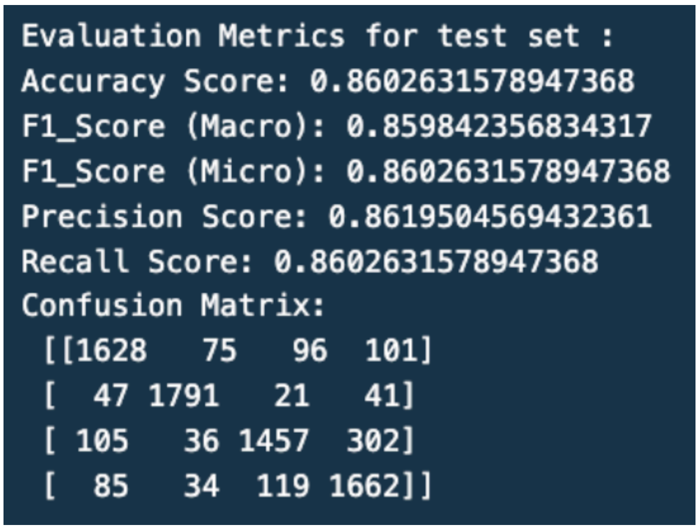


```
UNK_CUTOFF=3
UNKNOWN_TOKEN='<unk>'
WINDOW_SIZE=5
BATCH_SIZE=128
EMBEDDING_SIZE_SVD=300
PAD_TOKEN='<pad>'
NUM_LABELS=4
HIDDEN_SIZE=300
lrate=1e-3
EPOCHS=15
```

### Word-to-Vec (Skip Gram with Negative Sampling)

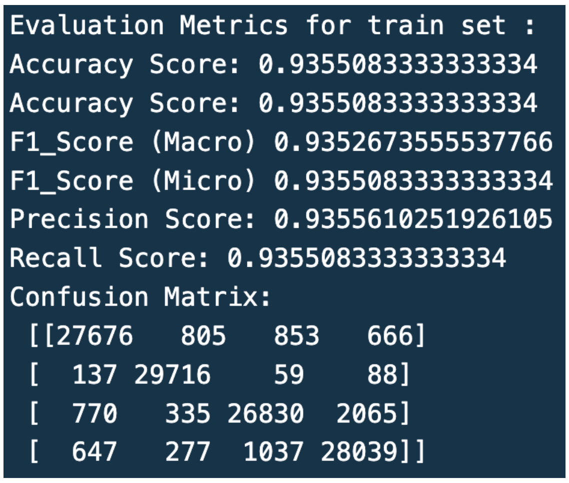


```
Evaluation Metrics for train set :
Accuracy Score: 0.8759833333333333
F1_Score (Macro): 0.8757692587965115
F1_Score (Micro): 0.8759833333333333
Precision Score: 0.8777963602869505
Recall Score: 0.8759833333333333
Confusion Matrix:
 [[26022  1166  1415  1397]
 [  558 28523   191   728]
 [ 1272   472 23946  4310]
 [ 1361   496  1516 26627]]
```

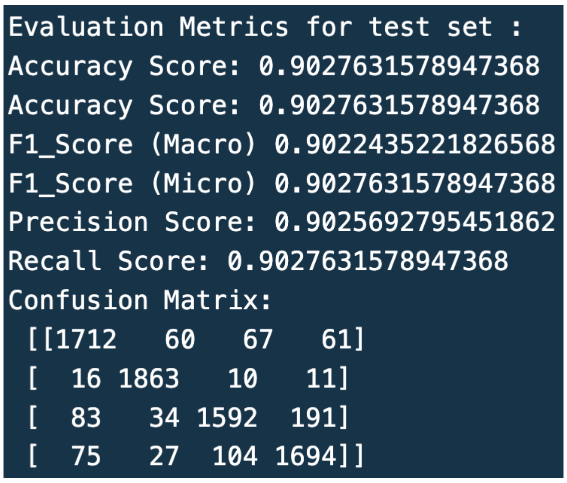


```
Evaluation Metrics for test set :
Accuracy Score: 0.8602631578947368
F1_Score (Macro): 0.859842356834317
F1_Score (Micro): 0.8602631578947368
Precision Score: 0.8619504569432361
Recall Score: 0.8602631578947368
Confusion Matrix:
 [[1628   75   96  101]
 [  47 1791   21   41]
 [ 105   36 1457  302]
 [  85   34  119 1662]]
```

## ELMO (Trainable λs)



```
Evaluation Metrics for train set :
Accuracy Score: 0.9355083333333334
Accuracy Score: 0.9355083333333334
F1_Score (Macro) 0.9352673555537766
F1_Score (Micro) 0.9355083333333334
Precision Score: 0.9355610251926105
Recall Score: 0.9355083333333334
Confusion Matrix:
 [[27676   805   853   666]
 [  137 29716    59    88]
 [  770   335 26830  2065]
 [  647   277  1037 28039]]
```



```
Evaluation Metrics for test set :
Accuracy Score: 0.9027631578947368
Accuracy Score: 0.9027631578947368
F1_Score (Macro) 0.9022435221826568
F1_Score (Micro) 0.9027631578947368
Precision Score: 0.9025692795451862
Recall Score: 0.9027631578947368
Confusion Matrix:
 [[1712   60   67   61]
 [  16 1863   10   11]
 [  83   34 1592  191]
 [  75   27  104 1694]]
```

## Which one (SVD, Word-to-Vec or ELMO) Performs Best and Why?

We can clearly see that performance on the downstream task is highest for ELMo, followed by SGNS and then SVD. This is same for all the performance metrics shown above, like accuracy score, F1_score (macro and micro), precision and recall scores.

### Analysis

SVD is a simple and interpretable method for generating word embeddings. It can capture **global co-occurrence statistics** in a corpus. However, it **may not capture subtle semantic relationships between words** effectively and may **not be scalable for large corpora**. It does not give contextual embeddings unlike ELMo.

Skip-gram with Negative Sampling, while **more contextually aware compared to SVD** as it captures local context of a word, may still **struggle with capturing fine-grained nuances of meaning present in individual sentences**. This is because each word has a fixed embedding representation regardless of its context in a sentence (treats each word in isolation).

ELMO, being a contextual embedding method, excels in capturing contextual information at the sentence level, which could be **crucial for sentence classification tasks where understanding the context of each sentence is essential.** It utilises a deep bidirectional LSTM (Long Short-Term Memory) architecture. LSTMs are a type of recurrent neural network (RNN) that can **capture long range dependencies in sequences**. By using a bidirectional LSTM, ELMo is able to **consider both past and future context when generating embeddings for each word.** It produces word embeddings by **concatenating the hidden states of multiple layers of the pre-trained LSTM**. This allows ELMo to **capture both lower level and higher level linguistic features**, providing a **more comprehensive representation of each word**. ELMo is pre-trained on a large corpus of text using a language modelling objective. This pre-training allows ELMo to **learn rich representations of words and phrases** from vast amounts of unlabelled text data. These learned representations can then be fine-tuned on specific downstream tasks, such as the one in our assignment for sentence classification. These together are the reasons for the performance gains that come with using ELMo in the downstream task.

## **THANK YOU!!**