INLP ASSIGNMENT 4

ELMO PRE-TRAINING

```
Epoch 1, Loss: 8.51672925936381
Epoch 2, Loss: 7.19869856783549
Epoch 3, Loss: 6.688641980616252
Epoch 4, Loss: 6.3848377759297685
Epoch 5, Loss: 6.175794686508179
Epoch 6, Loss: 6.017339987564087
Epoch 7, Loss: 5.889578502782186
Epoch 8, Loss: 5.782510144933065
Epoch 9, Loss: 5.6900148160298665
Epoch 10, Loss: 5.6095738613128665
```

Hyperparameters used for training the model :

embedding_dim = 150

hidden_dim = 150

n_epochs = 10

batch_size = 32

Downstream Task

Hyperparameters used :

input_dim = 300

hidden_dim = 128

n_layers = 2

activation = Relu

bidirectional = True

n_epochs = 5

batch_size = 32

1. Trainable λs

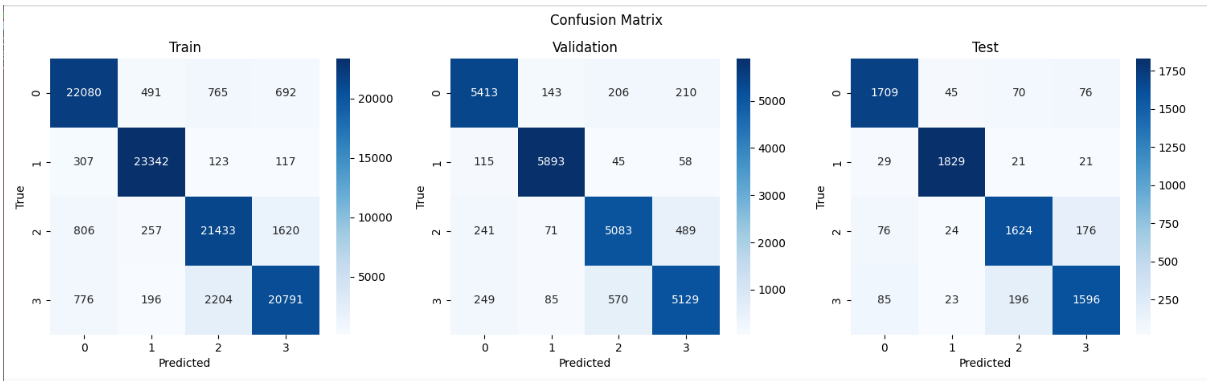
```
Epoch 1, Loss: 0.39901850585142773, Val Loss: 0.3460692796607812
Epoch 2, Loss: 0.33518507751325766, Val Loss: 0.3147899190982183
Epoch 3, Loss: 0.3065076022073627, Val Loss: 0.296415966908137
Epoch 4, Loss: 0.2859042163727184, Val Loss: 0.3109168243457874
Epoch 5, Loss: 0.26483009580274425, Val Loss: 0.29384757607678574
```

λ's after training: 0.8004, 0.2011, -1.6389

Metrics on train, val, and test data

```
Train Accuracy: 0.9130, F1 Score: 0.9128, Precision: 0.9128, Recall: 0.9130
Validation Accuracy: 0.8966, F1 Score: 0.8964, Precision: 0.8963, Recall: 0.8966
Test Accuracy: 0.8892, F1 Score: 0.8891, Precision: 0.8890, Recall: 0.8892
Train Confusion Matrix:
[[22080   491   765   692]
 [  307 23342   123   117]
 [  806   257 21433  1620]
 [  776   196  2204 20791]]
Validation Confusion Matrix:
[[5413  143  206  210]
 [ 115 5893   45   58]
 [ 241   71 5083  489]
 [ 249   85  570 5129]]
Test Confusion Matrix:
[[1709   45   70   76]
 [  29 1829   21   21]
 [  76   24 1624  176]
 [  85   23  196 1596]]
```

Confusion Matrices



2. Frozen λs

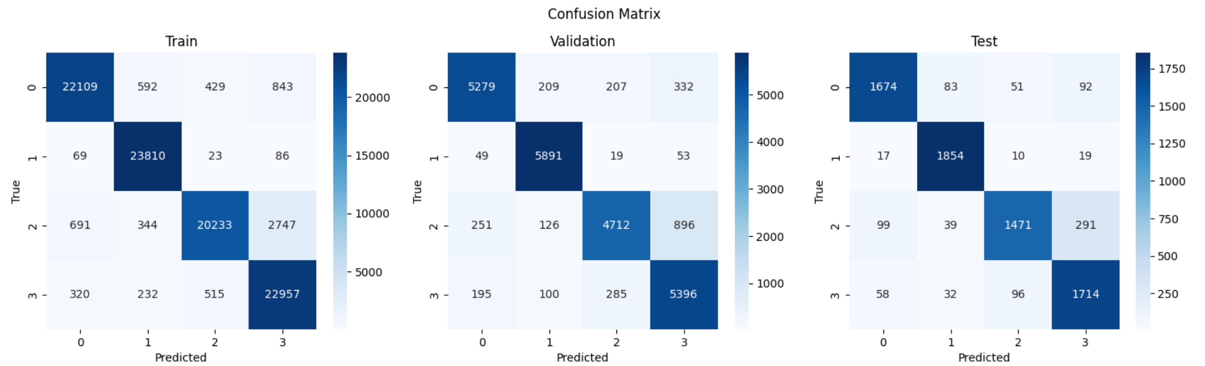
Frozen λ values: 0.9944, 1.3914, 0.3407

```
Epoch 1, Loss: 0.4421976037248969, Val Loss: 0.35300256941715874
Epoch 2, Loss: 0.32395926910390455, Val Loss: 0.33007342617710433
Epoch 3, Loss: 0.2860398773333679, Val Loss: 0.31152045454084876
Epoch 4, Loss: 0.25859664151196676, Val Loss: 0.30840833484133084
Epoch 5, Loss: 0.23073140067172548, Val Loss: 0.3268421005308628
```

Metrics on train, val, and test data

```
Train Accuracy: 0.9282, F1 Score: 0.9278, Precision: 0.9307, Recall: 0.9282
Validation Accuracy: 0.8866, F1 Score: 0.8859, Precision: 0.8891, Recall: 0.8866
Test Accuracy: 0.8833, F1 Score: 0.8824, Precision: 0.8857, Recall: 0.8833
Train Confusion Matrix:
[[22109   592   429   843]
 [   69 23810    23    86]
 [  691   344 20233  2747]
 [  320   232   515 22957]]
Validation Confusion Matrix:
[[5279  209  207  332]
 [  49 5891   19   53]
 [ 251  126 4712  896]
 [ 195  100  285 5396]]
Test Confusion Matrix:
[[1674   83   51   92]
 [  17 1854   10   19]
 [  99   39 1471  291]
 [  58   32   96 1714]]
```

Confusion matrices



3. Learnable Function

Every neural network learns a function. Therefore I used a neural network to learn a function.

```python
class function(nn.Module):
    def __init__(self, input_dim,output_dim, activation='relu'):
        super(function, self).__init__()
        self.fc1 = nn.Linear(input_dim, output_dim)
        if activation == 'relu':
            self.activation = nn.ReLU()
        elif activation == 'tanh':
            self.activation = nn.Tanh()
    def forward(self, e_0, h_0, h_1):
        x = torch.cat((e_0, h_0, h_1), dim=2)
        x = self.fc1(x)
        x = self.activation(x)
        return x
```

input_dim = 900

output_dim = 300

activation = Relu

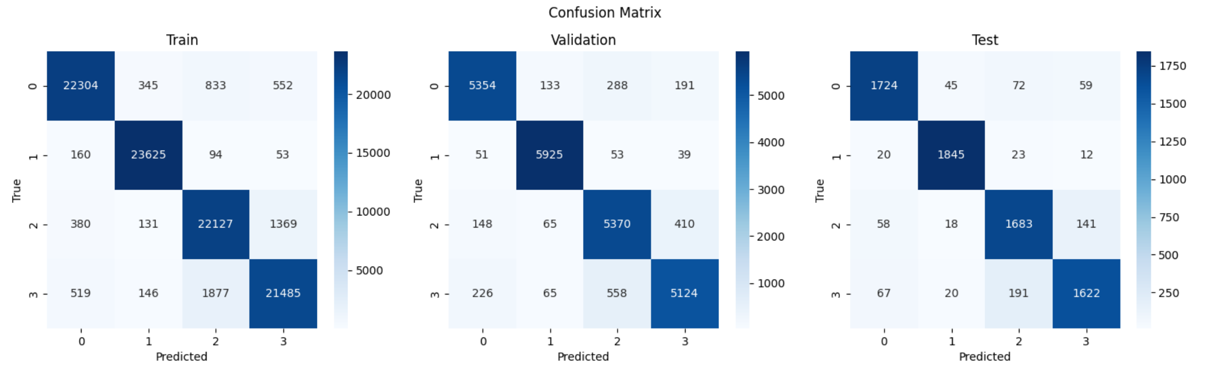
Training:

```
Epoch 1, Loss: 0.39327191315591337, Val Loss: 0.31366943327585856
Epoch 2, Loss: 0.2994465906197826, Val Loss: 0.2826680856992801
Epoch 3, Loss: 0.26756282774483164, Val Loss: 0.26819994534055539
Epoch 4, Loss: 0.24246932684878508, Val Loss: 0.25921959049999715
Epoch 5, Loss: 0.2215364538716773, Val Loss: 0.263000239931047
```

Metrics on train, val, and test data

```
Train Accuracy: 0.9327, F1 Score: 0.9327, Precision: 0.9331, Recall: 0.9327
Validation Accuracy: 0.9072, F1 Score: 0.9071, Precision: 0.9075, Recall: 0.9072
Test Accuracy: 0.9045, F1 Score: 0.9044, Precision: 0.9046, Recall: 0.9045
Train Confusion Matrix:
[[22304   345   833   552]
 [  160 23625    94    53]
 [  380   131 22127  1369]
 [  519   146  1877 21485]]
Validation Confusion Matrix:
[[5354  133  288  191]
 [  51 5925   53   39]
 [ 148   65 5370  410]
 [ 226   65  558 5124]]
Test Confusion Matrix:
[[1724   45   72   59]
 [  20 1845   23   12]
 [  58   18 1683  141]
 [  67   20  191 1622]]
```

Confusion Matrices



Comparing the above 3 models

We can see that the model with a learnable function performs better than the other two.

Order: Learnable function > Trainable λs > Frozen λs

Comparing SVD, SGNS, ELMO

Best performing 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]]
```

Best performing SGNS model

```
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]]
```

Best performing ELMO

```
Train Accuracy: 0.9327, F1 Score: 0.9327, Precision: 0.9331, Recall: 0.9327
Validation Accuracy: 0.9072, F1 Score: 0.9071, Precision: 0.9075, Recall: 0.9072
Test Accuracy: 0.9045, F1 Score: 0.9044, Precision: 0.9046, Recall: 0.9045
Train Confusion Matrix:
[[22304   345   833   552]
 [  160 23625    94    53]
 [  380   131 22127  1369]
 [  519   146  1877 21485]]
Validation Confusion Matrix:
[[5354  133  288  191]
 [  51 5925   53   39]
 [ 148   65 5370  410]
 [ 226   65  558 5124]]
Test Confusion Matrix:
[[1724   45   72   59]
 [  20 1845   23   12]
 [  58   18 1683  141]
 [  67   20  191 1622]]
```

We can see that ELMO embeddings outperform SVD and SGNS

Order: ELMO > SGNS > SVD

ELMO outperforms SVD and SGNS:

Factors contributing to the superior performance of ELMO embeddings compared to SVD and skip-gram:

1. **Contextualization**:
   ELMO embeddings capture word meanings based on their context within sentences,

enabling them to effectively handle polysemy and context-dependent semantics.

In contrast, SVD and skip-gram generate static embeddings that do not consider contextual information.

2. **Transfer Learning:**

   ELMO embeddings are pre-trained on large corpora using deep bidirectional language models, allowing them to learn rich linguistic representations. This pre-training facilitates transfer learning, where the embeddings can be fine-tuned on specific downstream tasks, enhancing performance.

3. **Flexibility:**

   ELMO embeddings are flexible and adaptive, capable of capturing complex linguistic patterns and adapting to diverse tasks and domains.

   In contrast, SVD and skip-gram embeddings may struggle with capturing intricate semantic relationships and contextual nuances.

4. ELMO embeddings, being based on deep neural networks, may have a higher model capacity and learning efficiency compared to SVD and skip-gram, contributing to faster convergence. The complexity of the model architecture plays a significant role in determining convergence speed.

5. **Data Efficiency:**

   ELMO embeddings require less annotated data for training compared to traditional techniques like SVD. This is attributed to the transfer learning paradigm, where pre-trained embeddings can be fine-tuned on smaller task-specific datasets, reducing the need for extensive labeled data.