Exercises on text classification with Multi-Layer Perceptrons (MLPs) № 2

Kagioglou Maria, Panourgia Evangelia 10/02/2025

Assignment 9

We utilized the `DBpedia 14` dataset, available on Hugging Face, which comprises **560,000 Wikipedia articles** categorized into **14 distinct classes**. Each entry in the dataset includes the **title** of a Wikipedia article along with its **text content**. The classification categories encompass a diverse range of entities, including **Company, Educational Institution, Artist, Athlete, Office Holder**, and others.

Due to the large size of the initial dataset, which contained **560,000 instances** in the training set and **70,000 instances** in the test set, we encountered **memory constraints**. To address this issue, we **downsampled** the dataset to optimize memory usage. As a result, the final dataset sizes after downsampling were **140,000 instances** for the training set and **17,500 instances** for the test set.

After downsampling, we further split the training dataset to create a development set, using a **0.2** proportion of the training data. Since the test set was already predefined, no further modifications were needed. The final dataset sizes were as follows:

- **Training set size:** 112,000
- **Development set size:** 28,000
- **Test set size:** 17,500

The Table 1 depicts the dataset distribution across the train, test and validations sets.

	1	**4**	**12**	**6**	**7**	**0**	**13**	**5**	**11**	**3**	**9**	**8**	**10**	**2**
Train	8009	7959	8046	7992	7951	8026	8020	8018	8017	7930	7975	8046	8040	7971
Validation	1991	2041	1954	2008	2049	1974	1980	1982	1983	2070	2025	1954	1960	2029
Test	1250	1250	1250	1250	1250	1250	1250	1250	1250	1250	1250	1250	1250	1250

Table 1: Dataset Distribution Across Train, Validation, and Test Sets

Some basic statistics related to our dataset are the following ones : Before prepossessing:

- **Average Document Length (in words):** 46.104
- **Average Document Length (in characters):** 281.194

After prepossessing:

- **Average Document Length (in words:** 29.645

- **Average Document Length (in characters):** 217.337

In addition, we calculaetd the `Flesch-Kincaid Readability Score` evaluates the complexity of a text in terms of readability:

- **Higher score** = More difficult to read.
- **Lower score** = Easier to read.

Regarding nlp pre-process, we implemented `preprocess(data)`, is designed to preprocess textual data by tokenizing sentences, filtering stopwords, and returning a cleaned version of the input text.

We implemented fundamental function (`get_features()`) for text representation which can be combined with TruncatedSVD. The representations include:

- Term Frequency-Inverse Document Frequency (TF-IDF)
- Word Embeddings (word2vec-google-news-300)
- Bag-of-Words (BoW)

Before initiating the MLP training, tuning, and evaluation process, we first trained baseline models to establish reference performance benchmarks that the MLP models needed to surpass. More specifically, we trained a **Random Classifier**, a **Majority** Classifier, and a **Logistic** Classifier. We computes and generated needed metrics related to classification report, precision-recall curves, macro precision across models and macro precision recall curves. It is worth noting that both majority and random classifiers have extremely low scores due to the nature of the dataset being close to "balance". At the end of this part, it is obvious that all models related to MLP architectures outperform all baselines for example in Figure **??** all models outperform "blue" line being the line of logistic regression.

Additionally, we developed functions related to model training including tuning process, too, focusing on:

- Model Training Model `train_model()`
- Evaluation of model `evaluate_model(), compute_metrics()`
- Model architecture (`class CustomModel`)
- Optimization techniques

Furthermore, we incorporated evaluation metrics such as:

- Classification report
- Precision, Recall, and F1-score

During the hyperparameter tuning process, we used the development set to optimize the parameters based on validation and training loss. We monitored the gap between them and checked for signs of overfitting, particularly at low epochs, which would indicate that the model cannot generalize effectively. We performed separate tuning for each model—Plain MLP, MLP with Dropout, MLP with BatchNorm, and MLP with LayerNorm—to identify the most appropriate configuration. After systematically tuning each model with these techniques and conducting additional exploratory trials, we proceeded with further tuning to find the optimal model based on our observations. The combinations we evaluated are presented in the table below.

Representation	Model	Hidden Units	Dropout Rate	Additional Features
TF-IDF	Linear	[]	0.0	No / No
	Plain MLP	[256, 128]	0.0	No / No
	MLP with Dropout	[1024]	0.5	No / No
	MLP with BatchNorm	[1024]	0.0	Yes / No
	MLP with LayerNorm	[1024]	0.0	No / Yes
	MLP with Dropout/Batch/ LayerNorm	[1024]	0.5	Yes / Yes
BoW	Linear	[]	0.0	No / No
	Plain MLP	[256, 128]	0.0	No / No
	MLP with Dropout	[512]	0.5	No / No
	MLP with BatchNorm	[1024]	0.0	Yes / No
	MLP with LayerNorm	[256, 1024]	0.0	No / Yes
	MLP with Dropout/Batch/ LayerNorm	[256, 128]	0.3	Yes / Yes
Word2Vec	Linear	[]	0.0	No / No
	Plain MLP	[128, 64]	0.0	No / No
	MLP with Dropout	[512]	0.5	No / No
	MLP with BatchNorm	[1024]	0.0	Yes / No
	MLP with LayerNorm	[1024]	0.0	No / Yes
	MLP with Dropout/Batch/ LayerNorm	[512]	0.5	Yes / Yes

Table 2: Model Configurations for Different Representations

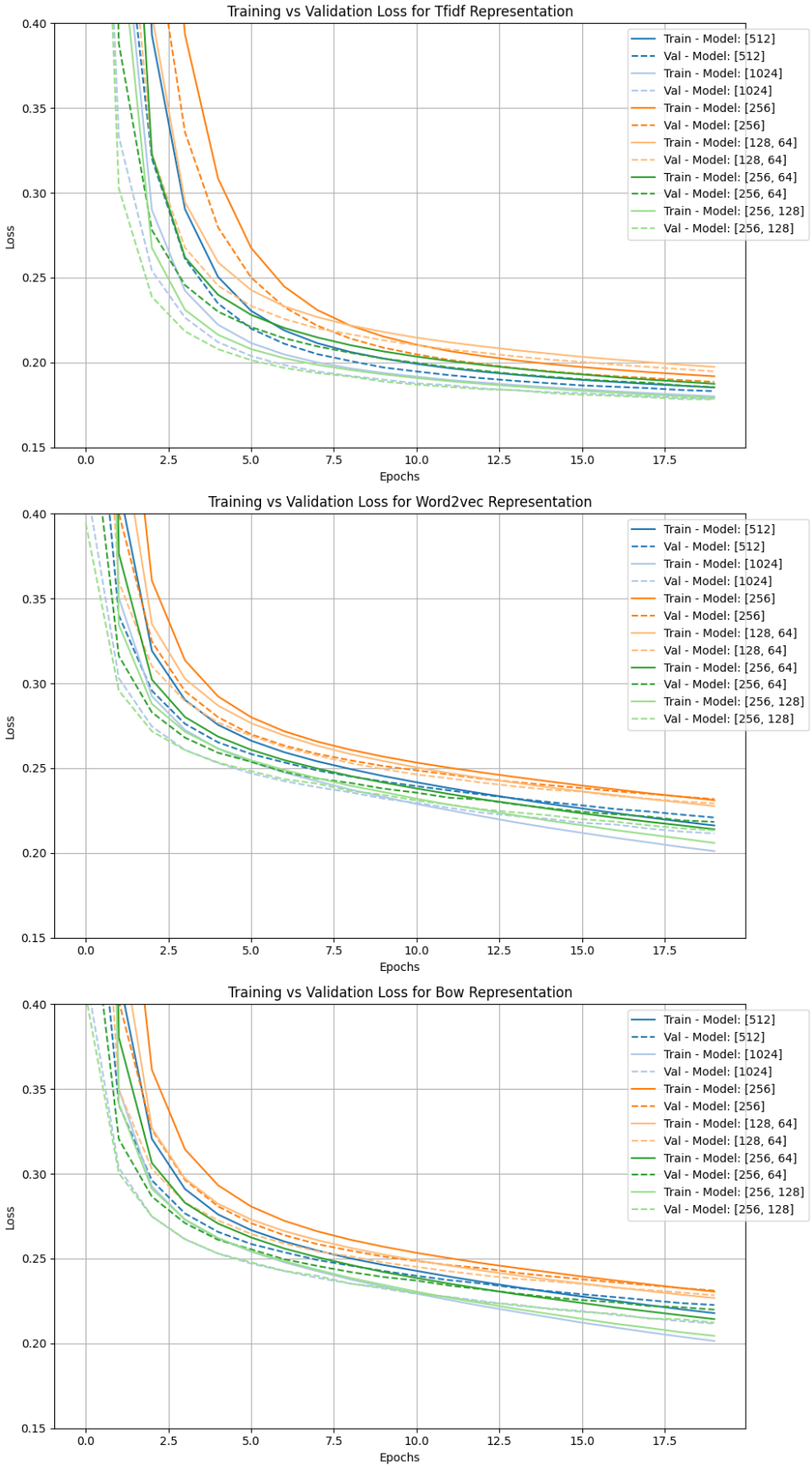

Figure 1: Tuning to Plain MLP

In the above plots, the aim is to find the best model for the Plain MLP, which consists only of hidden layers, based on the following parameter grid:

$$\text{param\_grid} = \left\{ \begin{array}{l} \text{"hidden\_layers"} : [[], [512], [1024], [256], [128, 64], [256, 64], [256, 128]], \\ \text{"dropout"} : [0.0], \\ \text{"batch\_norm"} : [\text{False}], \\ \text{"layer\_norm"} : [\text{False}] \end{array} \right\}$$

The key factors to consider in selecting the best model are: 1) The validation and training loss curves. 2) Avoiding overfitting by not training for too few epochs. 3) A small gap between the training and validation loss.

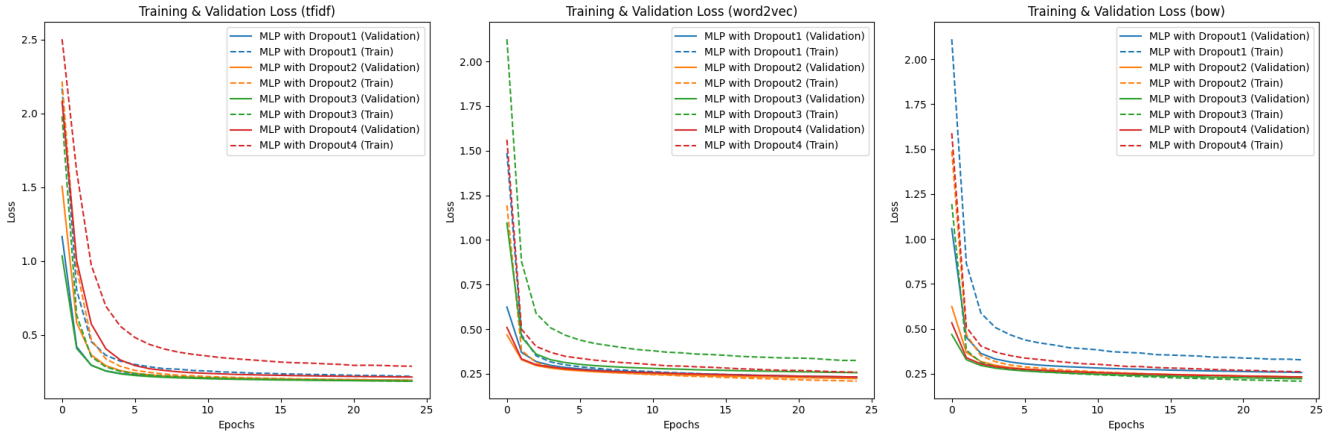

Figure 2: Tuning to MLP with Dropout

Specifically, for the models "MLP with BatchNorm" and "MLP with LayerNorm," we observed that while they reduce the losses, they also lead to overfitting at too low epochs.

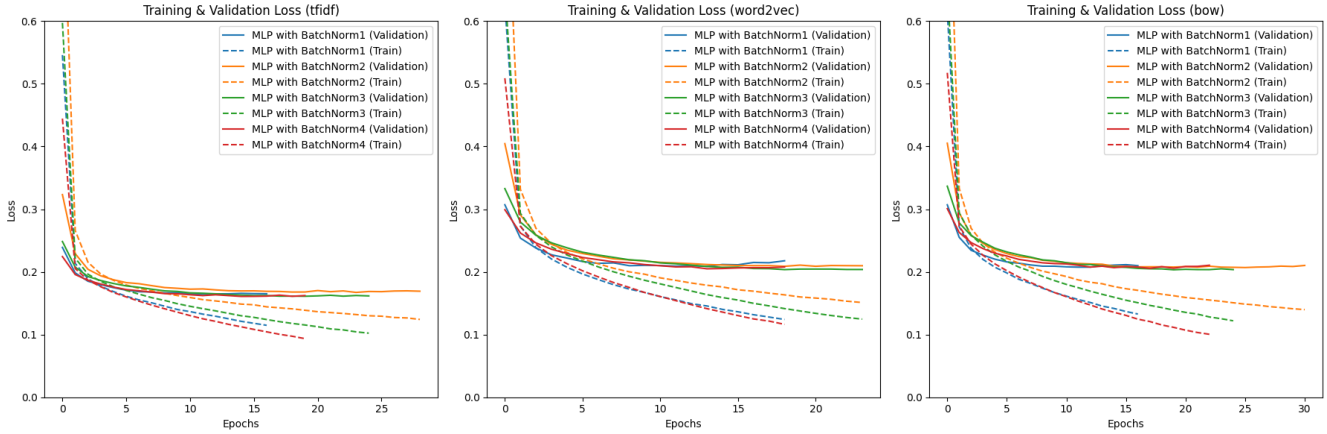

Figure 3: Tuning to MLP with BatchNorm

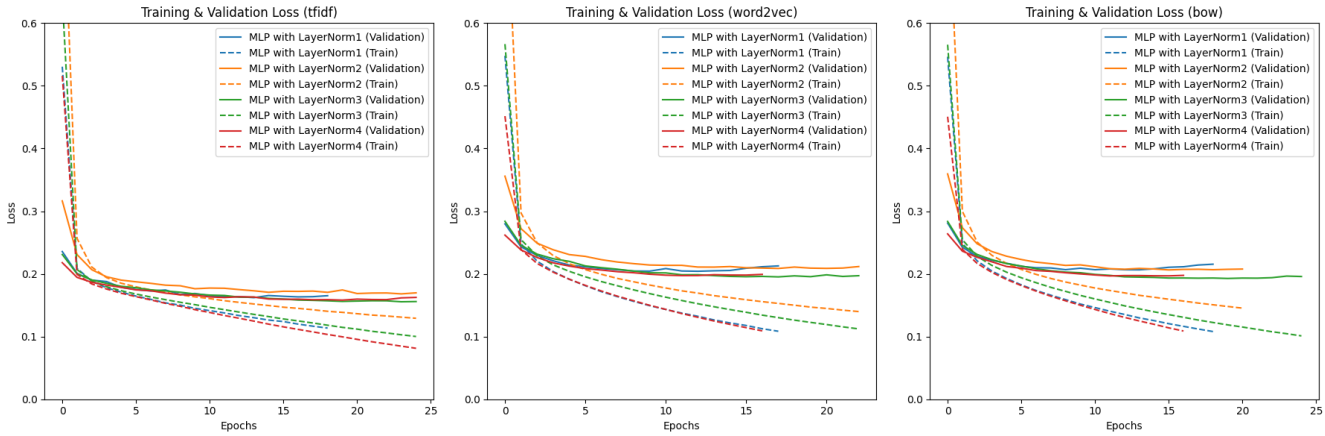

Figure 4: TuninG to MLP with LayerNorm

For each parameter combination and dataset split (train, test, and development), we generated the corresponding classification reports, including the `macro avg` values for precision, recall, and F1-score, as well as individual class-wise precision and recall metrics. Additionally, we visualized the results for each combination through plots of the following format **??**, which illustrates the precision-recall AUC scores and macro-averaged PR (with black –).

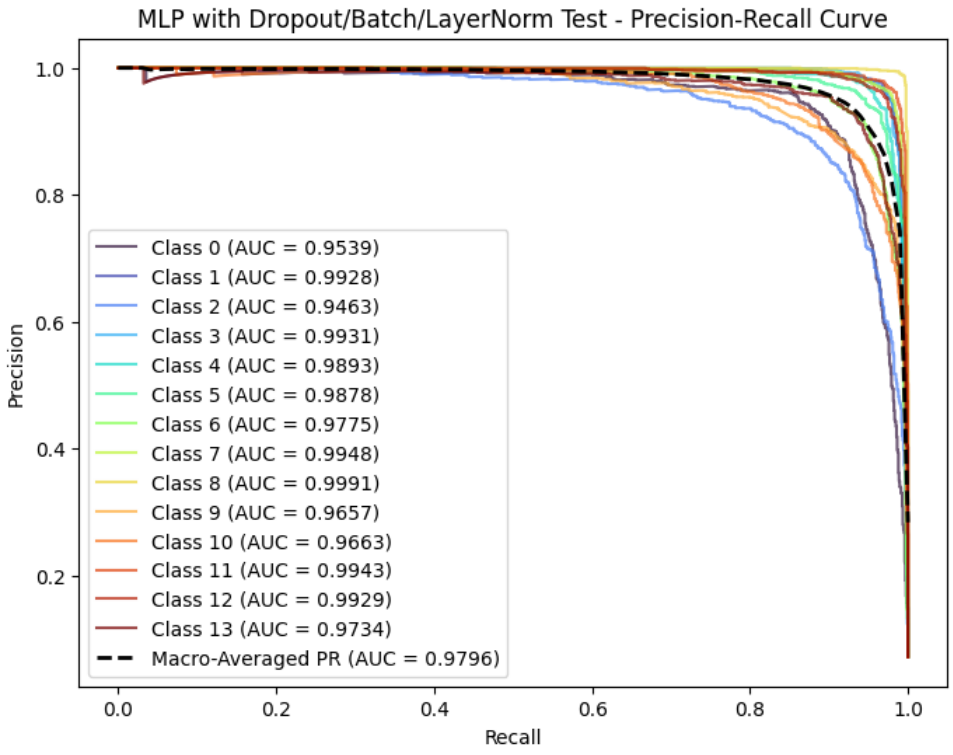

Figure 5: Precision-recall AUC scores for MLP with Layer Norm for validation set.

Regarding curves shown the loss on training, test and dev set we created the plots 6 and **??** (being the zoom of lines except the linear one).

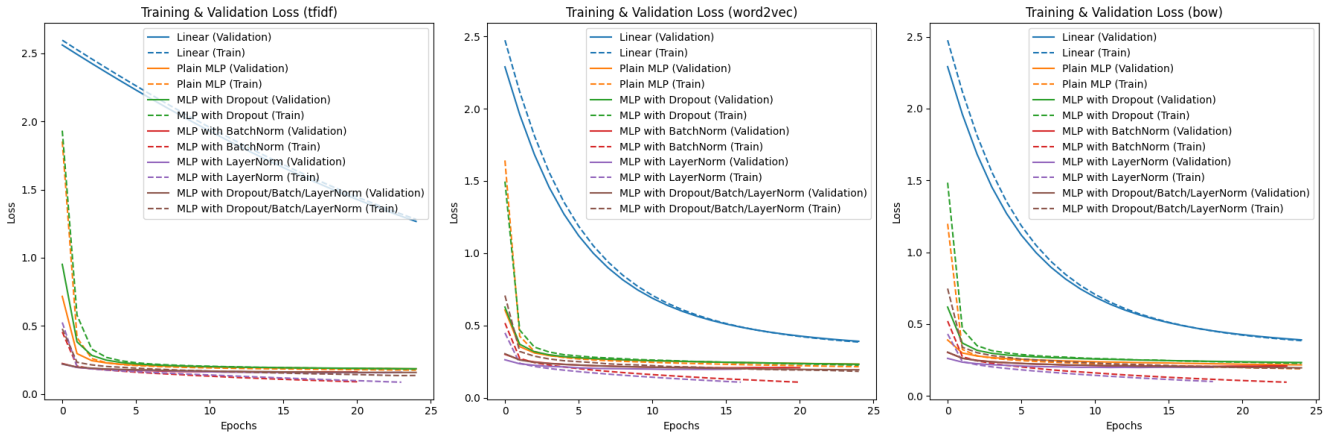

Figure 6: Loss curves.

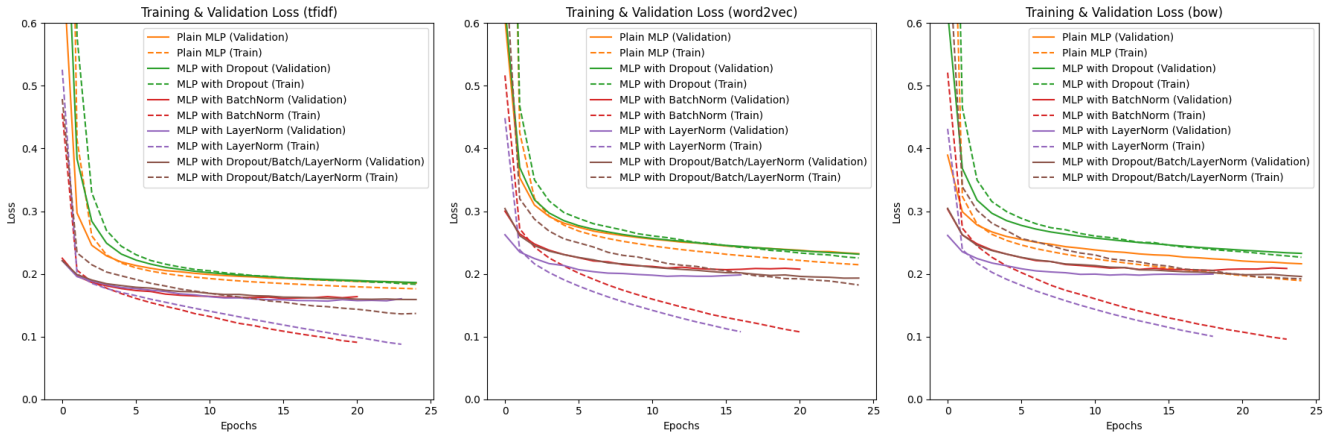

Figure 7: Loss curves.

Notes

- All models exhibit a decreasing training and validation loss, indicating proper convergence.
- The **Linear model** (blue lines) has significantly higher loss values compared to other models, suggesting that the Linear model is less effective than MLP-based architectures.
- As shown in 7, all three representations (BoW, TF-IDF, Word2Vec) yield good to acceptable results for this dataset.

Word2Vec

- **Worst Models:** The training and validation losses for `MLP with BatchNorm` (red lines) and `MLP with LayerNorm` (purple lines) exhibit a large gap, indicating overfitting.
- **Best Models:** `MLP with Dropout/BatchNorm/LayerNorm` (brown lines)

TF-IDF

- **Worst Models:** `MLP with BatchNorm` and `MLP with LayerNorm`.

- **Best Models:** `MLP with Dropout/BatchNorm/LayerNorm` (brown lines) being the best as it demonstrates a more controlled gap between training and validation loss, suggesting better regularization, as well as it keeps the low validation and training loss.

BoW

- **Worst Models:** `MLP with Dropout` (green lines), `Plain MLP` (orange lines), and `MLP with LayerNorm` (purple lines).
- **Best Models:** `MLP with BatchNorm` (red lines).

General Observation:
All models reach overfitting at approximately the same time, suggesting that they extract useful information at a similar rate. This may indicate that the dataset lacks sufficient complexity for further learning or that it is highly coherent, containing consistent, well-structured patterns. Consequently, models learn these patterns quickly, causing the training loss to decrease rapidly and explaining why all models reach a natural learning limit at the same epoch.

Additionally, we generated plots for Macro Precision-Recall curves and Macro Precision trends across different models for all dataset splits (train, test, and development) **??**, **??**. It is obvious that all models related to MLP architectures outperform all baselines for example in Figure **??** all models outperform "blue" line being the line of logistic regression.

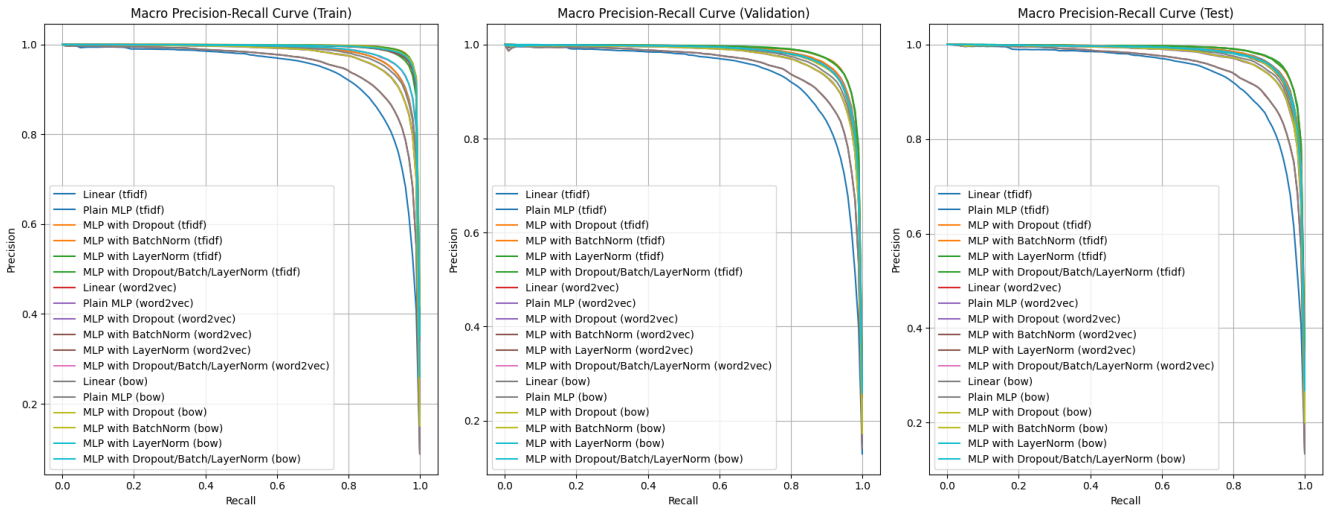

Figure 8: Macro Precision Recall curves across sets.

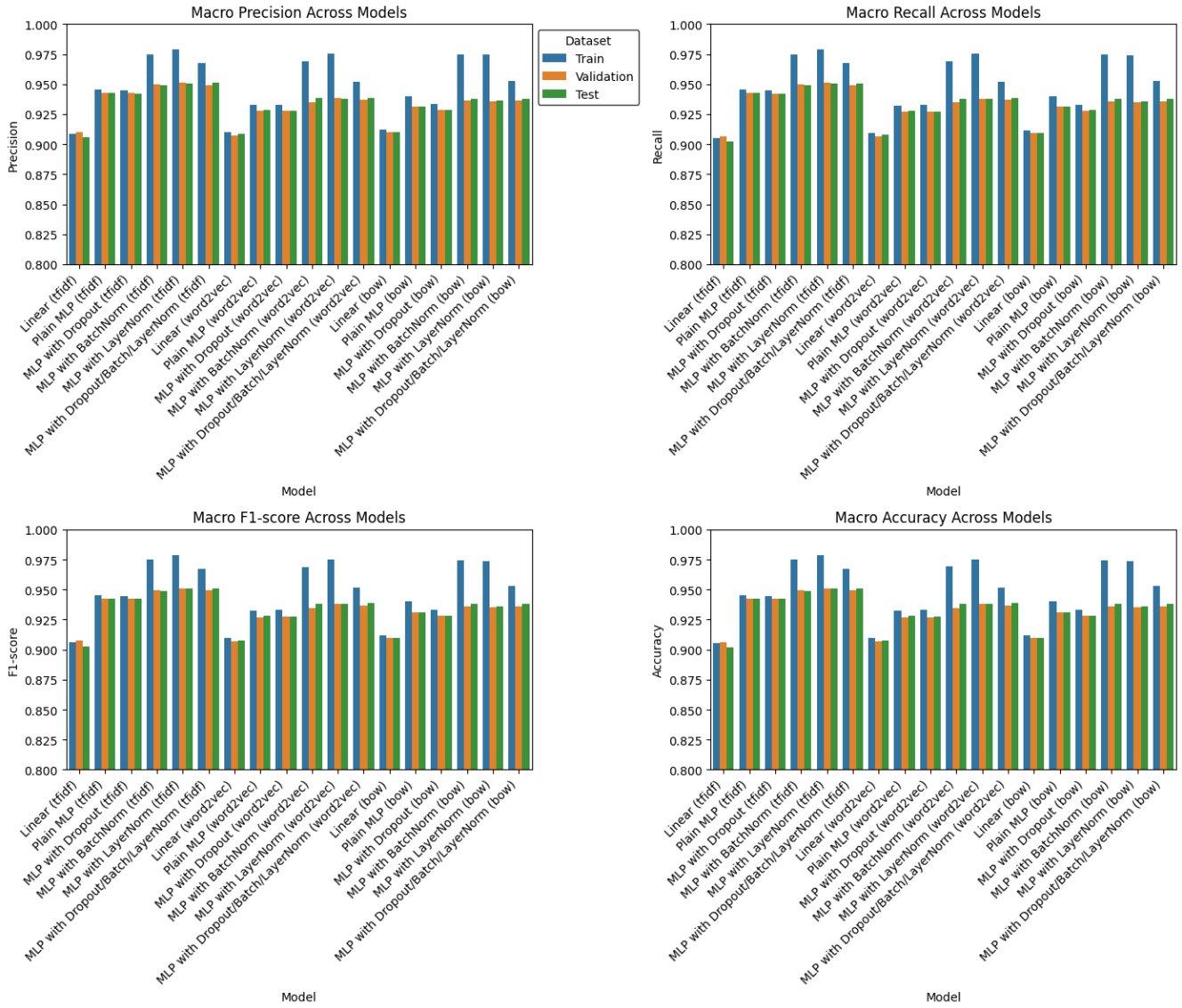

Figure 9: Macro precision across models and sets.

- As shown in Figure 9, the Linear MLP displays the lowest macro recall and precision across all datasets (train, validation, and test).
- All other models display approximately the same macro recall and precision, with differences among them ranging within 0.05.
- From Figure 8, we can conclude that the Linear model is the worst among all, while the second-best model is the MLP with Layer Norm.

Baseline

Below, we implement a baseline approach using Logistic Regression, along with Majority and Random Classifiers.

Logistic Regression

For Logistic Regression we employed grid search to find the optimal model parameters. The grid search parameters were defined as follows:

```
parameters = {
    'solver': ['liblinear', 'saga'],  # solvers to try
    'penalty': ['l1', 'l2'],  # regularization penalties
    'C': [0.01, 0.1, 1],  # inverse of regularization strength
}
```

We defined the Logistic Regression model and applied GridSearchCV as follows:

```
logreg = LogisticRegression(max_iter=1000)
grid_search = GridSearchCV(logreg, parameters, cv=3, scoring='accuracy', n_jobs=-1)

# Train model using grid search
grid_search.fit(X_train, y_train)

# Get best model
best_model_tfidf = grid_search.best_estimator_
```

We applied this grid search for three different representations: Bag of Words (BoW), Word2Vec, and TF-IDF.

Below, we can see the results in test set:

```
=== Logistic Regression (WORD2VEC) Test Set Classification Report ===
              precision    recall  f1-score   support

           0     0.8668    0.8696    0.8682      1250
           1     0.9694    0.9624    0.9659      1250
           2     0.8418    0.8640    0.8527      1250
           3     0.9678    0.9632    0.9655      1250
           4     0.9474    0.9360    0.9416      1250
           5     0.9266    0.9296    0.9281      1250
           6     0.9208    0.9024    0.9115      1250
           7     0.9318    0.9624    0.9469      1250
           8     0.9857    0.9904    0.9880      1250
           9     0.8459    0.8960    0.8702      1250
```

```
           10     0.9033    0.8368    0.8688      1250
           11     0.9731    0.9560    0.9645      1250
           12     0.9630    0.9568    0.9599      1250
           13     0.8949    0.9056    0.9002      1250

    accuracy                         0.9237     17500
   macro avg     0.9242    0.9237    0.9237     17500
weighted avg     0.9242    0.9237    0.9237     17500

Logistic Regression (WORD2VEC) Test Set - Class 0 Precision-Recall AUC: 0.9204
Logistic Regression (WORD2VEC) Test Set - Class 1 Precision-Recall AUC: 0.9864
Logistic Regression (WORD2VEC) Test Set - Class 2 Precision-Recall AUC: 0.9203
Logistic Regression (WORD2VEC) Test Set - Class 3 Precision-Recall AUC: 0.9884
Logistic Regression (WORD2VEC) Test Set - Class 4 Precision-Recall AUC: 0.9798
Logistic Regression (WORD2VEC) Test Set - Class 5 Precision-Recall AUC: 0.9823
Logistic Regression (WORD2VEC) Test Set - Class 6 Precision-Recall AUC: 0.9660
Logistic Regression (WORD2VEC) Test Set - Class 7 Precision-Recall AUC: 0.9863
Logistic Regression (WORD2VEC) Test Set - Class 8 Precision-Recall AUC: 0.9979
Logistic Regression (WORD2VEC) Test Set - Class 9 Precision-Recall AUC: 0.9352
Logistic Regression (WORD2VEC) Test Set - Class 10 Precision-Recall AUC: 0.9361
Logistic Regression (WORD2VEC) Test Set - Class 11 Precision-Recall AUC: 0.9830
Logistic Regression (WORD2VEC) Test Set - Class 12 Precision-Recall AUC: 0.9824
Logistic Regression (WORD2VEC) Test Set - Class 13 Precision-Recall AUC: 0.9439

=== Logistic Regression (WORD2VEC) Test Set - Macro Precision-Recall AUC: 0.9643 ===
```

```
=== Logistic Regression (TF-IDF) Test Set Classification Report ===
              precision    recall  f1-score   support

           0     0.8816    0.8872    0.8844      1250
           1     0.9688    0.9680    0.9684      1250
           2     0.8766    0.8976    0.8870      1250
           3     0.9719    0.9672    0.9695      1250
           4     0.9599    0.9568    0.9583      1250
           5     0.9403    0.9456    0.9430      1250
           6     0.9202    0.9312    0.9256      1250
           7     0.9661    0.9568    0.9614      1250
           8     0.9864    0.9880    0.9872      1250
           9     0.8936    0.9272    0.9101      1250
          10     0.9406    0.8872    0.9131      1250
          11     0.9748    0.9600    0.9674      1250
          12     0.9640    0.9632    0.9636      1250
          13     0.9131    0.9168    0.9150      1250

    accuracy                         0.9395     17500
   macro avg     0.9398    0.9395    0.9396     17500
weighted avg     0.9398    0.9395    0.9396     17500

Logistic Regression (TF-IDF) Test Set - Class 0 Precision-Recall AUC: 0.9425
Logistic Regression (TF-IDF) Test Set - Class 1 Precision-Recall AUC: 0.9904
Logistic Regression (TF-IDF) Test Set - Class 2 Precision-Recall AUC: 0.9556
Logistic Regression (TF-IDF) Test Set - Class 3 Precision-Recall AUC: 0.9931
Logistic Regression (TF-IDF) Test Set - Class 4 Precision-Recall AUC: 0.9871
Logistic Regression (TF-IDF) Test Set - Class 5 Precision-Recall AUC: 0.9863
Logistic Regression (TF-IDF) Test Set - Class 6 Precision-Recall AUC: 0.9766
Logistic Regression (TF-IDF) Test Set - Class 7 Precision-Recall AUC: 0.9920
Logistic Regression (TF-IDF) Test Set - Class 8 Precision-Recall AUC: 0.9987
Logistic Regression (TF-IDF) Test Set - Class 9 Precision-Recall AUC: 0.9730
Logistic Regression (TF-IDF) Test Set - Class 10 Precision-Recall AUC: 0.9720
Logistic Regression (TF-IDF) Test Set - Class 11 Precision-Recall AUC: 0.9910
Logistic Regression (TF-IDF) Test Set - Class 12 Precision-Recall AUC: 0.9909
Logistic Regression (TF-IDF) Test Set - Class 13 Precision-Recall AUC: 0.9647

=== Logistic Regression (TF-IDF) Test Set - Macro Precision-Recall AUC: 0.9789 ===

=== Logistic Regression (BoW) Test Set Classification Report ===
              precision    recall  f1-score   support
```

```
              0     0.8584    0.8648    0.8616      1250
              1     0.9595    0.9592    0.9594      1250
              2     0.8346    0.8560    0.8449      1250
              3     0.9616    0.9592    0.9604      1250
              4     0.9382    0.9248    0.9315      1250
              5     0.9161    0.9136    0.9148      1250
              6     0.9096    0.8920    0.9007      1250
              7     0.9259    0.9512    0.9384      1250
              8     0.9822    0.9864    0.9843      1250
              9     0.8320    0.8744    0.8527      1250
             10     0.8955    0.8232    0.8583      1250
             11     0.9709    0.9488    0.9597      1250
             12     0.9514    0.9408    0.9461      1250
             13     0.8835    0.8904    0.8869      1250

    accuracy                           0.9167     17500
   macro avg     0.9170    0.9167    0.9168     17500
weighted avg     0.9170    0.9167    0.9168     17500

Logistic Regression (BoW) Test Set - Class 0 Precision-Recall AUC: 0.9103
Logistic Regression (BoW) Test Set - Class 1 Precision-Recall AUC: 0.9867
Logistic Regression (BoW) Test Set - Class 2 Precision-Recall AUC: 0.9065
Logistic Regression (BoW) Test Set - Class 3 Precision-Recall AUC: 0.9921
Logistic Regression (BoW) Test Set - Class 4 Precision-Recall AUC: 0.9743
Logistic Regression (BoW) Test Set - Class 5 Precision-Recall AUC: 0.9706
Logistic Regression (BoW) Test Set - Class 6 Precision-Recall AUC: 0.9517
Logistic Regression (BoW) Test Set - Class 7 Precision-Recall AUC: 0.9864
Logistic Regression (BoW) Test Set - Class 8 Precision-Recall AUC: 0.9973
Logistic Regression (BoW) Test Set - Class 9 Precision-Recall AUC: 0.9340
Logistic Regression (BoW) Test Set - Class 10 Precision-Recall AUC: 0.9345
Logistic Regression (BoW) Test Set - Class 11 Precision-Recall AUC: 0.9849
Logistic Regression (BoW) Test Set - Class 12 Precision-Recall AUC: 0.9841
Logistic Regression (BoW) Test Set - Class 13 Precision-Recall AUC: 0.9504

=== Logistic Regression (BoW) Test Set - Macro Precision-Recall AUC: 0.9682 ===
```

Majority Classifier

Strategy: `most_frequent`

- The classifier always predicts the most common class found in the training data.

- It ignores the features and simply returns the majority class for all instances.
- This serves as a strong baseline for imbalanced datasets, as it demonstrates the performance that could be achieved by always guessing the majority class.

Here are the results from the Majority Classifier, which predicts the most frequent class for the training, validation, and test sets.

Training Set Classification Report (Majority Classifier)

	precision	recall	f1-score	support
0	1.00	0.00	0.00	8026
1	1.00	0.00	0.00	8009
2	1.00	0.00	0.00	7971
3	1.00	0.00	0.00	7930
4	1.00	0.00	0.00	7959
5	1.00	0.00	0.00	8018
6	1.00	0.00	0.00	7992
7	1.00	0.00	0.00	7951
8	0.07	1.00	0.13	8046
9	1.00	0.00	0.00	7975
10	1.00	0.00	0.00	8040
11	1.00	0.00	0.00	8017
12	1.00	0.00	0.00	8046
13	1.00	0.00	0.00	8020
accuracy			0.07	112000
macro avg	0.93	0.07	0.01	112000
weighted avg	0.93	0.07	0.01	112000

Validation Set Classification Report (Majority Classifier)

	precision	recall	f1-score	support
0	1.00	0.00	0.00	1974
1	1.00	0.00	0.00	1991
2	1.00	0.00	0.00	2029
3	1.00	0.00	0.00	2070
4	1.00	0.00	0.00	2041
5	1.00	0.00	0.00	1982
6	1.00	0.00	0.00	2008
7	1.00	0.00	0.00	2049

8	0.07	1.00	0.13	1954
9	1.00	0.00	0.00	2025
10	1.00	0.00	0.00	1960
11	1.00	0.00	0.00	1983
12	1.00	0.00	0.00	1954
13	1.00	0.00	0.00	1980
accuracy			0.07	28000
macro avg	0.93	0.07	0.01	28000
weighted avg	0.94	0.07	0.01	28000

Test Set Classification Report (Majority Classifier)

	precision	recall	f1-score	support
0	1.00	0.00	0.00	1250
1	1.00	0.00	0.00	1250
2	1.00	0.00	0.00	1250
3	1.00	0.00	0.00	1250
4	1.00	0.00	0.00	1250
5	1.00	0.00	0.00	1250
6	1.00	0.00	0.00	1250
7	1.00	0.00	0.00	1250
8	0.07	1.00	0.13	1250
9	1.00	0.00	0.00	1250
10	1.00	0.00	0.00	1250
11	1.00	0.00	0.00	1250
12	1.00	0.00	0.00	1250
13	1.00	0.00	0.00	1250
accuracy			0.07	17500
macro avg	0.93	0.07	0.01	17500
weighted avg	0.93	0.07	0.0	17500

Analysis of Majority Classifier's Behavior:

- While the test set is balanced, the training set might be imbalanced. If class 8 is overrepresented in the training data, the model may be biased toward predicting class 8. - Since both class 12 and class 8 have the same count (8046), the DummyClassifier might be: - Choosing the smallest numerical label (8 instead of 12). - Selecting based on the internal ordering of classes.

To check the majority class selected by the DummyClassifier, we can print the prediction for a sample instance:

```
print("Majority class selected by DummyClassifier:", majority_clf.predict([X_test[0]]))
```

If it prints 8, the classifier is likely choosing the lowest numerical label in case of a tie between class 8 and class 12.

DummyClassifier is working correctly, but it's picking class 8 due to tie-breaking rules.

Random Classifier

Next, we use a `RandomClassifier`.

Strategy: `uniform`

- The classifier randomly assigns labels with equal probability across all classes.
- Each class has an equal chance of being predicted, regardless of the actual class distribution in the training data.
- This is useful as a baseline for comparison when dealing with imbalanced datasets.

Here are the results from the RandomClassifier:

Random Classifier Train Set Classification Report

Class	Precision	Recall	F1-Score	Support
0	0.0727	0.0700	0.0713	8026
1	0.0715	0.0715	0.0715	8009
2	0.0724	0.0713	0.0718	7971
3	0.0733	0.0757	0.0745	7930
4	0.0722	0.0735	0.0729	7959
5	0.0712	0.0710	0.0711	8018
6	0.0705	0.0708	0.0707	7992
7	0.0701	0.0697	0.0699	7951
8	0.0753	0.0772	0.0763	8046
9	0.0695	0.0697	0.0696	7975
10	0.0718	0.0725	0.0721	8040
11	0.0706	0.0695	0.0701	8017
12	0.0763	0.0756	0.0759	8046
13	0.0720	0.0716	0.0718	8020
Accuracy		0.0721		112000
Macro Avg	0.0721	0.0721	0.0721	112000
Weighted Avg	0.0721	0.0721	0.0721	112000

Table 3: Classification Report for Random Classifier on Train Set

Random Classifier Validation Set Classification Report

Class	Precision	Recall	F1-Score	Support
0	0.0777	0.0770	0.0774	1974
1	0.0712	0.0698	0.0705	1991
2	0.0681	0.0655	0.0668	2029
3	0.0623	0.0618	0.0620	2070
4	0.0684	0.0681	0.0682	2041
5	0.0768	0.0777	0.0773	1982
6	0.0741	0.0722	0.0732	2008
7	0.0841	0.0839	0.0840	2049
8	0.0620	0.0686	0.0651	1954
9	0.0719	0.0711	0.0715	2025
10	0.0663	0.0668	0.0666	1960
11	0.0676	0.0656	0.0666	1983
12	0.0697	0.0711	0.0704	1954
13	0.0689	0.0692	0.0690	1980
Accuracy		0.0706		28000
Macro Avg	0.0707	0.0706	0.0706	28000
Weighted Avg	0.0707	0.0706	0.0706	28000

Table 4: Classification Report for Random Classifier on Validation Set

Random Classifier Test Set Classification Report

Class	Precision	Recall	F1-Score	Support
0	0.0647	0.0640	0.0643	1250
1	0.0720	0.0744	0.0732	1250
2	0.0612	0.0600	0.0606	1250
3	0.0742	0.0736	0.0739	1250
4	0.0767	0.0776	0.0772	1250
5	0.0655	0.0648	0.0651	1250
6	0.0677	0.0664	0.0670	1250
7	0.0652	0.0656	0.0654	1250
8	0.0656	0.0656	0.0656	1250
9	0.0778	0.0784	0.0781	1250
10	0.0852	0.0840	0.0846	1250
11	0.0729	0.0720	0.0724	1250
12	0.0680	0.0688	0.0684	1250
13	0.0712	0.0728	0.0720	1250
Accuracy		0.0706		17500
Macro Avg	0.0706	0.0706	0.0706	17500
Weighted Avg	0.0706	0.0706	0.0706	17500

Table 5: Classification Report for Random Classifier on Test Set

Analysis of Precision-Recall AUC

- **Low Discriminatory Power:** The Precision-Recall AUC score of 0.5357, close to 0.5, indicates poor model performance, with the classifier struggling to distinguish between classes. A score closer to 1 would suggest a better-performing classifier.
- **No Class Bias:** The uniform AUC score across all classes suggests the model is not biased towards any class but is consistently performing poorly.
- **Imbalance or Model Limitation:** In the case of imbalanced datasets, the Random Classifier would predict the majority class most of the time, resulting in low AUC scores, which implies the model isn't effectively utilizing the features.

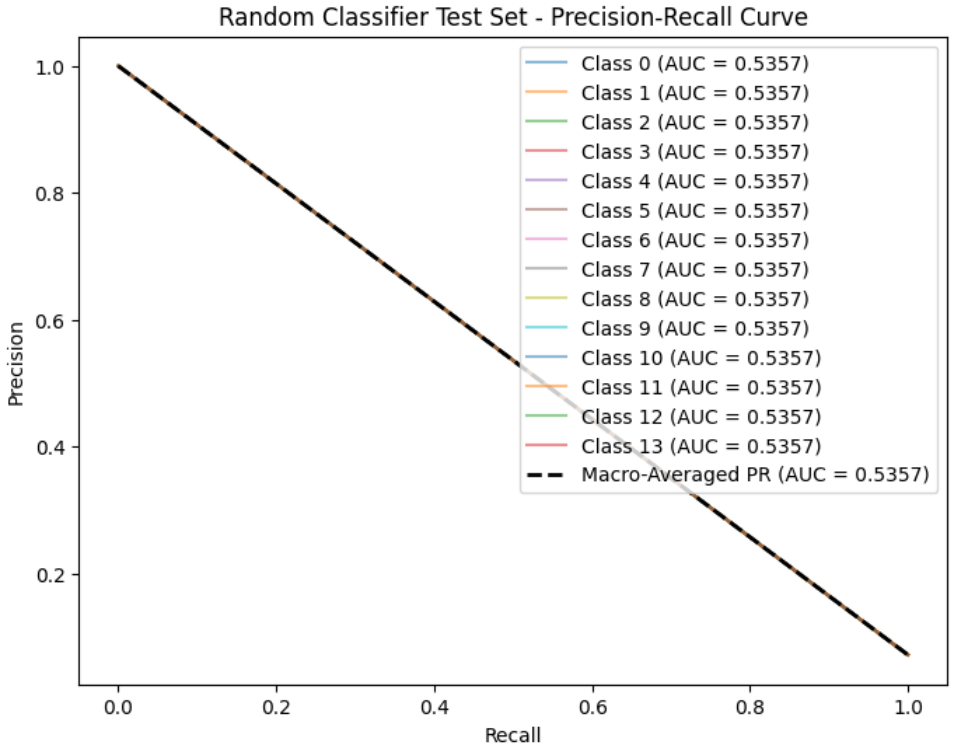

Figure 10: Precision-Recall Curves for Random Classifier

Comparison of Logistic Regression & MLP

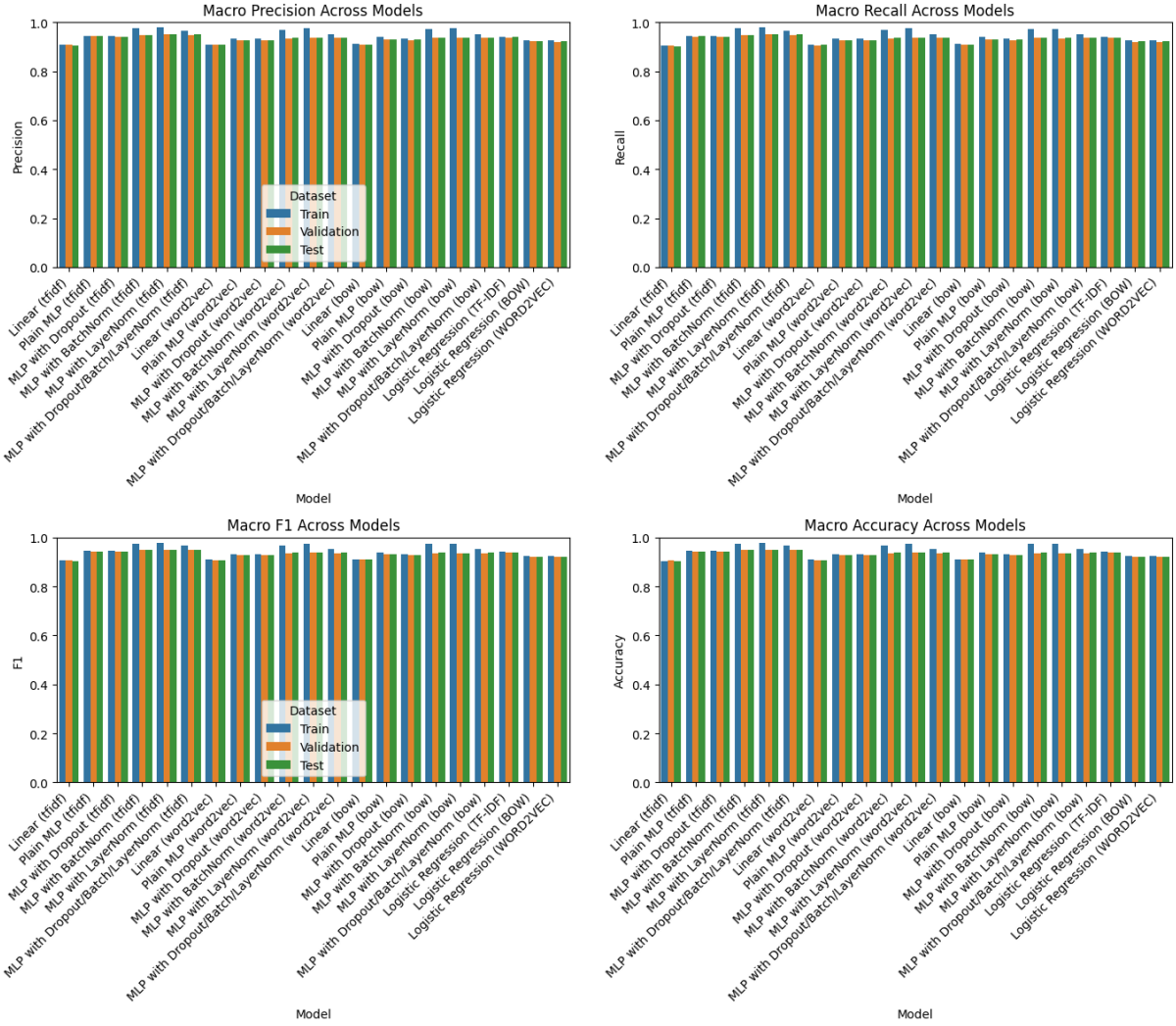

Figure 11: Macro Metrics Comparison between Logistic Regression and MLP

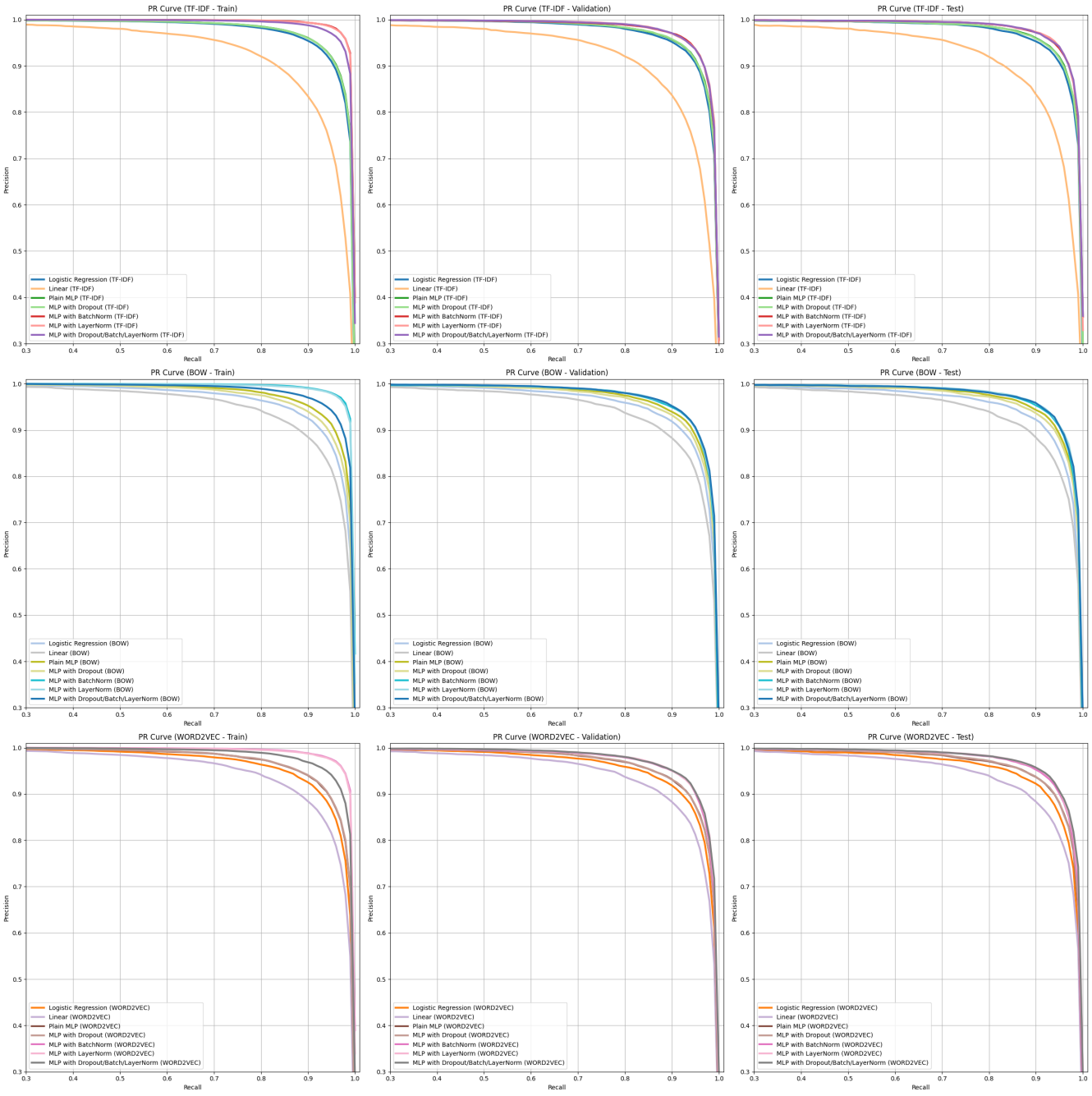

Figure 12: Precision-Recall Curves for Logistic Regression and MLP

Statistics

Figure 13 illustrates the number of samples for each class across all datasets:

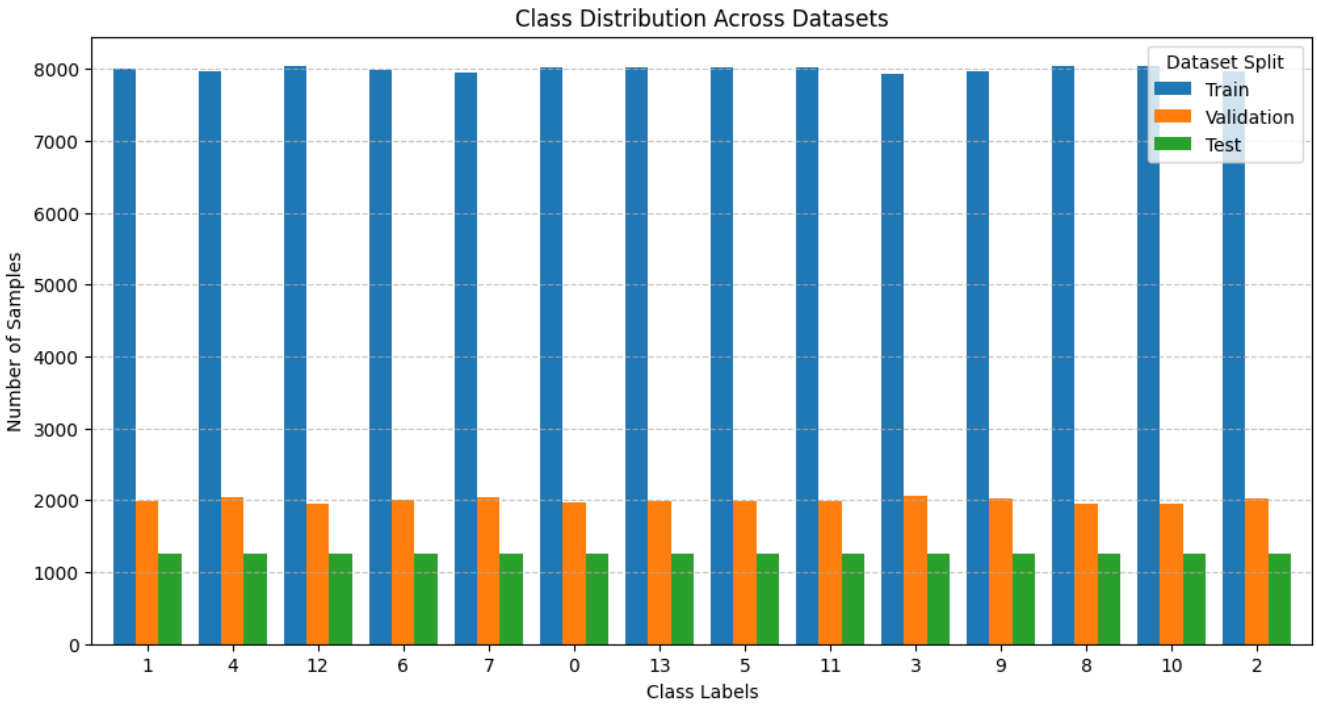

Figure 13: Class Distribution Across Datasets

The results below demonstrate the average document length, measured in words and characters, before and after preprocessing for the training set.

Before Preprocessing:

- **Average Document Length (in words):** 46.104
- **Average Document Length (in characters):** 281.194

After Preprocessing:

- **Average Document Length (in words):** 29.645
- **Average Document Length (in characters):** 217.337

Flesch-Kincaid Readability Scores: This metric evaluates the complexity of a text by considering sentence length and syllable count per word. A higher score indicates that the text is more difficult to read, whereas a lower score suggests that the text is easier to comprehend.
Average Flesch-Kincaid Grade Level: 10.94

This result suggests that the text is moderately complex, using a mix of longer sentences and more syllables per word. In many contexts, a grade level around 10.94 is acceptable; however, it might be challenging for readers with lower reading proficiency.

Below we can see the number of unique tokens (i.e., the vocabulary size) in each dataset after the preprocessing stage:

After Preprocessing:

- **Vocabulary Size (Training Set):** 290,636
- **Vocabulary Size (Testing Set):** 80,107
- **Vocabulary Size (Validation Set):** 111,160

General Notes

- **Both the Majority and Random Classifiers struggle to correctly classify instances and underperform compared to the MLP.**
- **Logistic Regression appears to work well on this dataset; however, its strong performance may be attributed to the dataset's low complexity.**
- **According to the precision-recall curve in Figure 12, the best-performing model is the MLP with regularization (BatchNorm, Dropout).**
- **As shown in Figure 11, Logistic Regression achieves high precision and recall.**
- **The MLP demonstrates high precision and recall across all models because it can learn intricate relationships between features. This capability allows it to generalize better to both positive and negative instances, resulting in fewer false positives (higher precision) and fewer false negatives (higher recall).**
- **Overall, the MLP's ability to capture complex patterns enables it to outperform simpler, linear models such as Logistic Regression.**

Assignment 10

[Dataset]: Part-of-Speech (POS) Tagger

To build our Part-of-Speech tagger, we utilize English words, sentences, and tags from the Universal Dependencies dataset available at: `https://github.com/UniversalDependencies/UD_English-EWT`.

This dataset provides three '*.conllu*' files: `train`, `dev`, and `test`. The method `read_conllu_file` extracts **sentences, words,** and **POS tags** from these files. So, we created two separate data frames for each set (train, test and dev) containing the former an index with the sentence (in the format <s> . . . <e>), and the latter being in the format of the table below where the column sentence sn the "key" between the the separate data frames.

Words	Sentence	POS
What	0	PRON
if	0	SCONJ
Google	0	PROPN

Then, we applied an additional preprocessing step to transform the data into the `Window` format. Specifically, we extracted three-word sliding windows from each sentence, ensuring that each window consists of a previous word (`Wi-1`), a center word (`Wi`), and a next word (`Wi+1`). Special tokens such as <s> and <e> were excluded. Additionally, we mapped the POS tag of the center word (`Wi`) from the word-level annotations to create a structured dataset in a tabular format, facilitating contextual analysis.

Having the new dataset, we cleaned it removing rows having the symbol "-" in column Wi_POS_tag. Furthermore, we transformed the text-based data into numerical representations by leveraging FastText word embeddings.

Specifically, we utilized a pre-trained FastText model to obtain word vectors for each word in the sliding window format. The function `get_word_vector` retrieves the word embedding for a given word, ensuring valid input before processing. These embeddings were then applied to the training, test, and development sets by mapping each word in the window (`Wi-1`, `Wi`, `Wi+1`) to its corresponding FastText vector (`Ei-1`, `Ei`, `Ei+1`). This step allows the dataset to be transformed into a numerical format suitable for machine learning models.

Then, we removed potential missing values from all data frames across different dataset splits. Following this, we transformed the feature representation by concatenating the embeddings of the three consecutive tokens (`Ei-1`, `Ei`, `Ei+1`) into a single vector.

To achieve this, we implemented the function `concatenate_vectors`, which merges the embeddings of the three words in each window, provided all necessary embeddings exist. This ensures that each row in the dataset now represents a contextualized word embedding that encapsulates local word dependencies.

Finally, we applied this transformation to the training, test, and development datasets by adding a new column, `Concatenated_Embeddings`, which stores the resulting combined feature

vectors. This step ensures that our data is in a format suitable for training machine learning models that leverage contextual word representations.

We then transformed `X` and `y` for each dataset into the appropriate format to ensure compatibility as valid inputs for the training process using `PyTorch`.

During the training, tuning, and evaluation process, we adhered to the same coding logic as in Assignment 9, while incorporating customized modifications specific to our Multi-Layer Perceptron (MLP) architecture. It is important to be noted that during training similarly to assignment 9 we used early stopping to avoid in a way the over-fit.

We utilized the validation set to identify the most optimal model architecture and its corresponding hyper parameters. In Table 7, we have presented the combinations of hyper parameters we examined. The optimal hyperparameter combination, designated as "Optimal" in the aforementioned table, was determined through a comprehensive evaluation of multiple metrics, including loss trends (seee Figure 14) across epochs—crucial for mitigating overfitting—(meaning small gap between train and val set and low value in loss function), classification report metrics based mostly on accuracy and macro average of f1 column and on Macro Average PR (AUC) in order to have an overview of how well the model predicts the classes. More specifically these metrics are useful as **accuracy** shows the overall correctness of the model, the **Macro F1-score** considers both precision and recall. It gives equal importance to all classes, making it useful for imbalanced datasets (which is our case) and **Macro PR AUC** is the area under the Precision-Recall curve. It measures how well the model ranks positive cases over negative ones. It is important for handling imbalanced datasets.

Configuration	Hidden Layers	Dropout	Batch Norm	Layer Norm
1	[512]	0.6	False	True
Optimal	[1024]	0.6	False	True
2	[256, 128]	0.6	False	True
3	[128, 64]	0.6	False	True

Table 7: Examined MLP configurations for Word2Vec during the tuning process based on hidden layers.

Regarding the baseline implementation, we developed a majority baseline model for classification, which assigns each word to its most frequently occurring tag in the training data. For unseen words, the model assigns the most common tag observed across the entire training set.

Regarding the baseline comparison, - **MLP does not consistently outperform the baseline in all categories but in the majority of them**. - **MLP improves performance in many cases**, particularly in **recall and F1-score for most classes**. More specifically, based on the summary values of accuracy (F1 score) and macro-averaged F1 score, we observe the following:

- **Accuracy (F1 Score):**
  - Majority Baseline: 0.86
  - Optimal MLP: 0.91

- **Macro AVG (F1 Score):**
  - Majority Baseline: 0.80
  - Optimal MLP: 0.83

These results demonstrate that the **Optimal MLP model** achieves better overall classification performance, particularly in handling class imbalances, as indicated by the improvement in the **macro-averaged F1 score**.

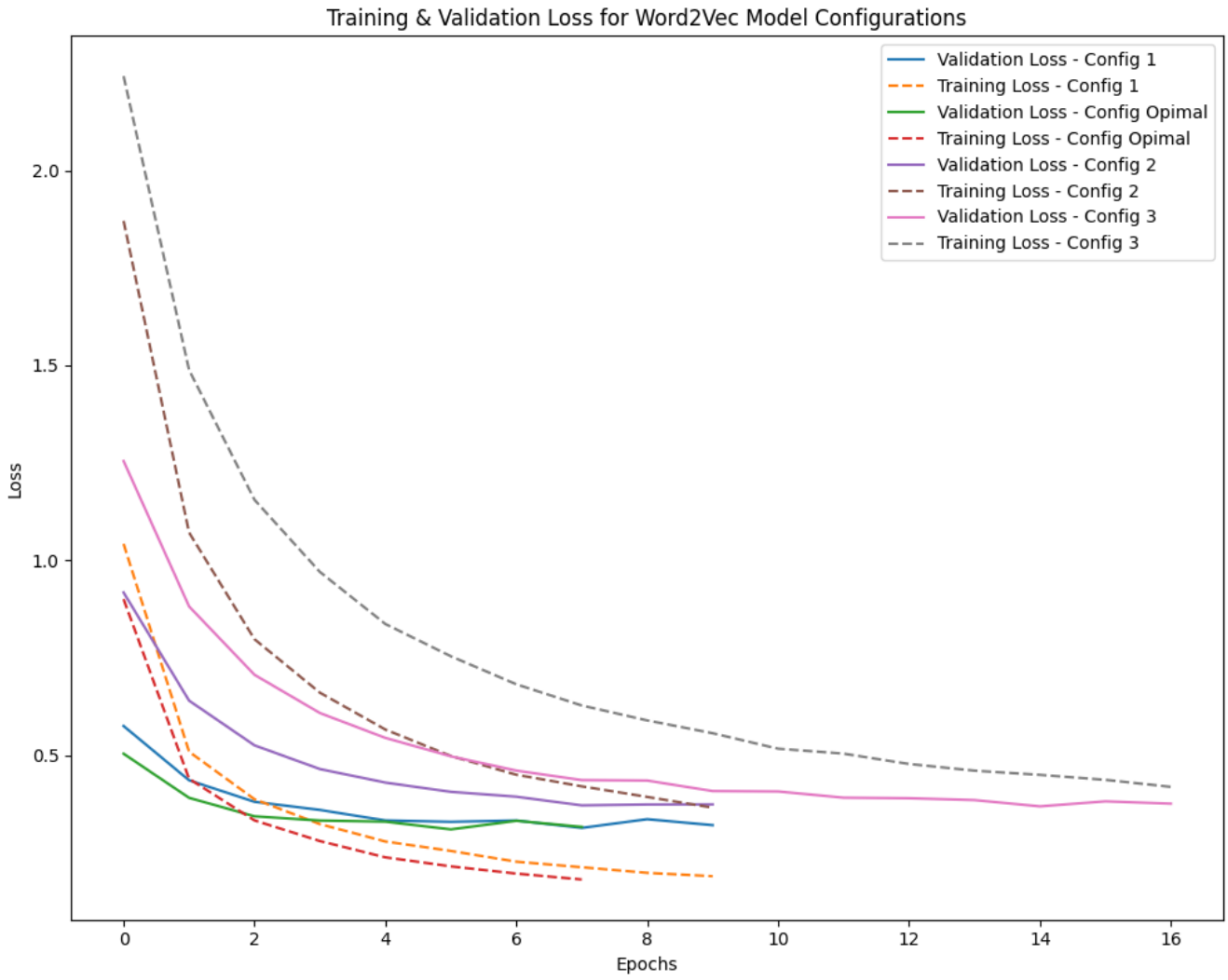

Figure 14: Loss curves.

In conclusion, the best-performing model (having the best combination of hidden layers stemming from the tuning process) achieved the following metrics across the sets of test, val and train respectively:

- **Accuracy**: $0.91,\ 0.92\ ,\ 0.94$
- **Macro Average F1-score**: $0.83\ ,\ 0.84,\ 0.89$
- **Macro Average PR (AUC)**: $0.92,\ 0.91,\ 0.97$

These results indicate that the model has high overall correctness, balances precision and recall well across classes, and performs effectively in ranking positive examples over negative ones. The respective loss plot across train and dev set is presented in 16. The training loss (orange) decreases steadily, indicating that the model is learning effectively over epochs. The validation loss (blue) starts decreasing but then plateaus, suggesting that the model's performance on unseen data stops improving after a few epochs. The gap is not quite high so there is not strong evidence for high over fit.

In addition, for a better overview of metrics of our optimal configuration model across the different sets in presented in Figure15.

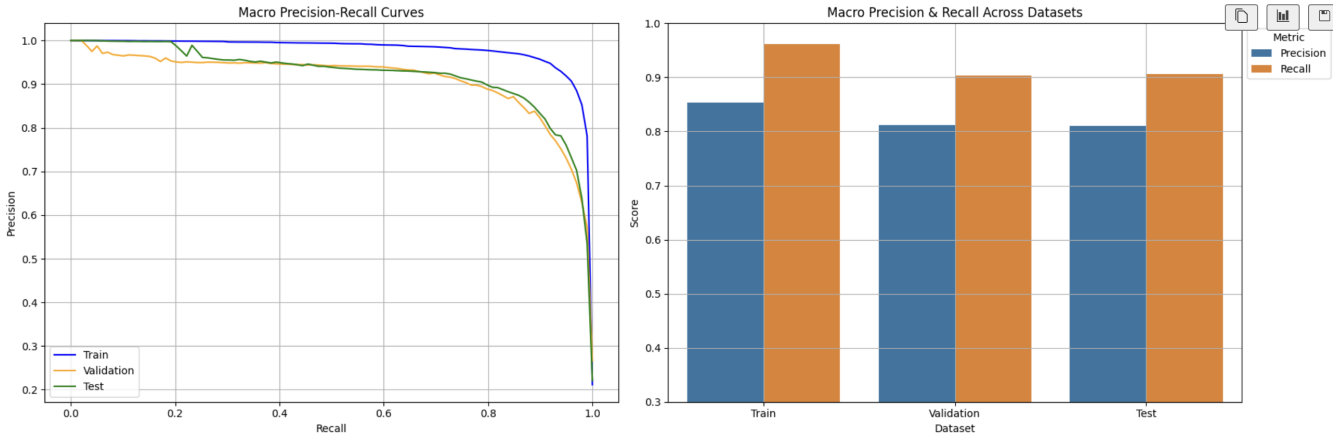

Figure 15: Metrics of the Optimal model across training , test and val sets.

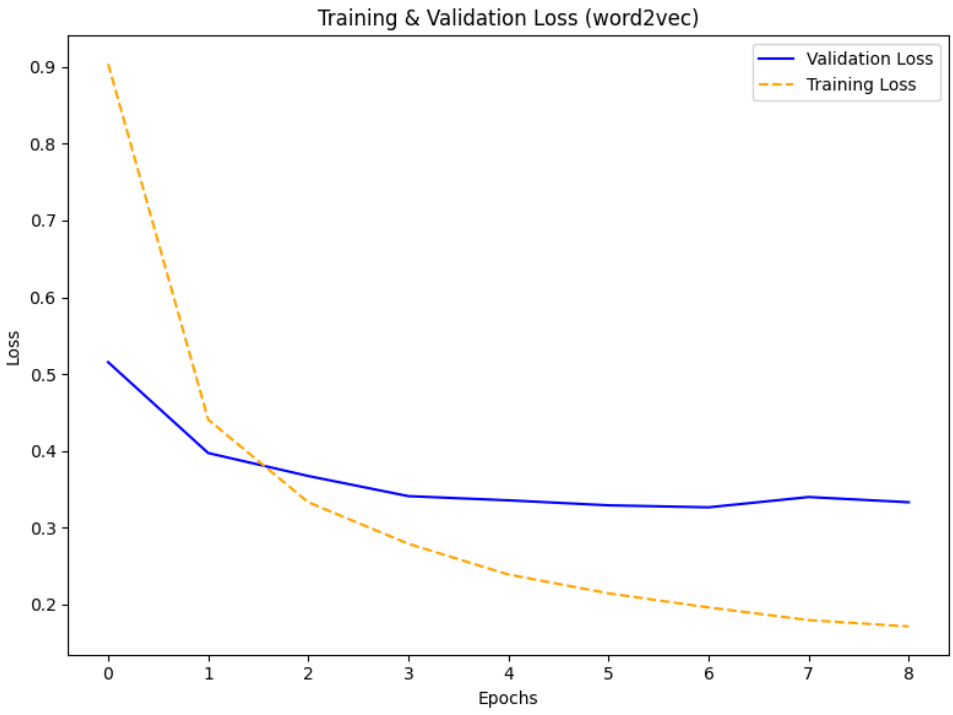

Figure 16: Loss curves.

In addition, in order to have a better overview what classes the MLP created a confusion matrix Figure 17. Most predictions are correctly classified, as seen from the strong diagonal

values. The highest correct predictions occur for **AUX (3190)** and **INTJ (2300)**, indicating strong model performance in these categories. Some misclassifications occur in similar categories, such as **ADJ vs. NOUN** etc.

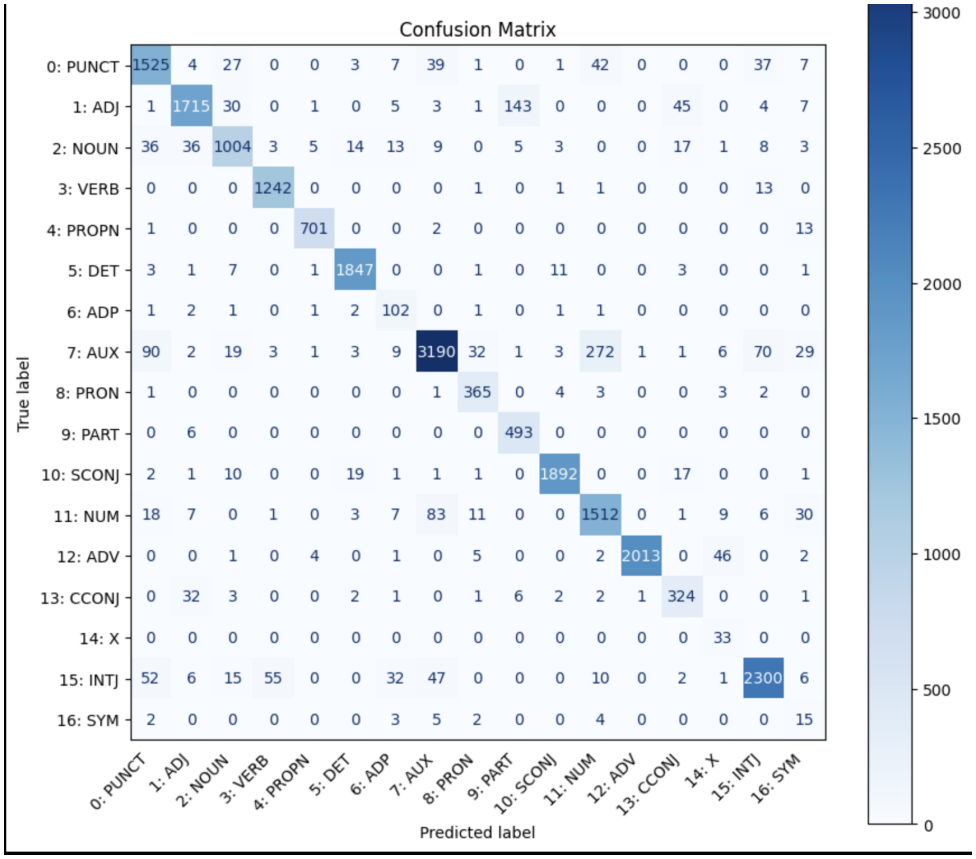

Figure 17: Confusion Matrix.