Matricola: **1871835,** Department: **Artificial Intelligence and Robotics**

1.0 How datasets was used:

The **major datasets** used for the purpose of this project is **AS and MSR**, Using Hanzi-convert, AS was converted to simplified version**.**

1. The **preprocess.py** script was used to convert all original files to an **Input** and **label** file.
2. The **split_ngram.py** script was used to split our merged AS and MSR training and validation data to unigram alongside their respective labels in unigram format.
3. Using the paper approach, Wang2vec was used to **embed** our dataset after which we used the gensim KeyedVectors function was used to load our **embeddings**. The parameters used here was: `-type 1 -size 100 -window 10`
4. The **training data** used was the **unigram** interpretation of our AS and MSR training data, while **validation data** was the same set of datasets, but from the gold folder.
5. The **embedding matrix** was obtained by loading the wang2vec output data into a key-value pair word2vec decoder from which the corresponding vector for each word in it is derived. This matrix was further used as the weights for our model Embedding layer.

2.0 Description of Model

Using the paper approach and selecting the model with best performance, the **non-stacking** architecture was chosen for this project as the stacking architecture(b) performed poorly for our case, the **sum** merging technique for non-stacking was used on the results of our **forward** and **backward** LSTM. **Grid search** was trained with the following parameters in **5 epochs,** the epochs could have been more but insufficient training resource:

- Learning Rate: [0.04, 0.03]
- Character size : [256]
- Input dropout rate: [0.20, 0.25]
- Optimizer: [‘adam’, ‘sgd’]

The **best hyper parameters** found for a generalized model was: `{'dropout': 0.2, 'lr': 0.04, 'optimizer': 'adam', 'char_size': 256}`. This set of params produced a **precision** score of **0.803** on our validation data. The code for this is found in **grid_model.py**.

3.0 Decisions made to improve the model performance

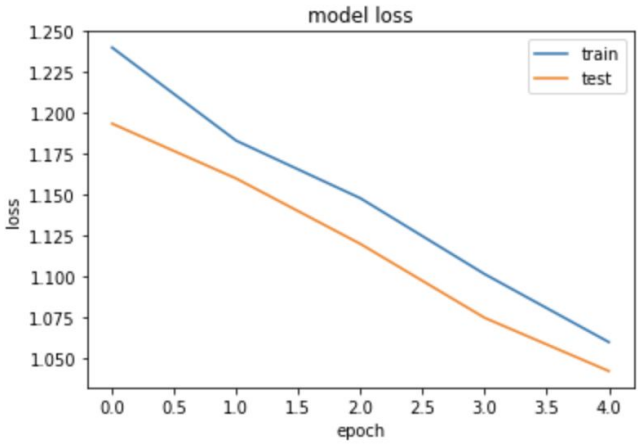
It was noticed that the BIES output class of our data was imbalanced, with the “I” especially as a minority. Hence, we employed the use of **SMOTE oversampling**, a technique used to oversample data while maintaining equal number for the 4 classes distribution. We trained a new model with the resampled data in a separate python script “**smote_model.py**” using the best parameters we got above from grid search and we were able to achieve a higher precision value of **0.858** which is slightly better than our previous value**.** We also used the **EarlyStopping** approach to detect when our model is overfitting in order to control it, we used this as our **final** model.

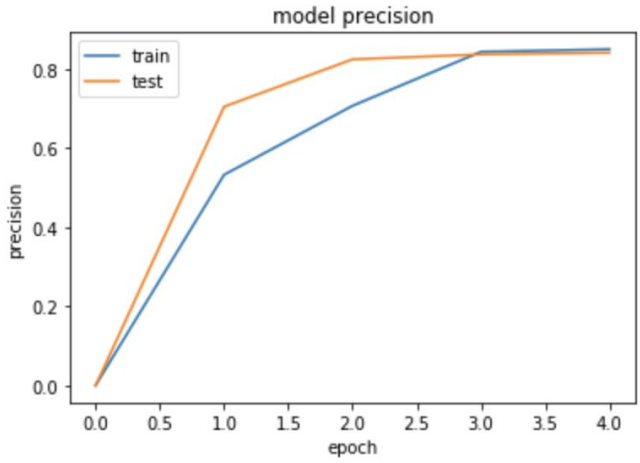
4.0 Figures and Plots

The screenshot below shows a classification report before and after the SMOTE technique was used on our datasets. We will observe our classification report metrics changed on the second figure which is the SMOTE technique used before training.

	precision	recall	f1-score	support
0	0.38	0.62	0.47	165
1	0.48	0.29	0.36	165
2	0.00	0.00	0.00	30
3	0.57	0.54	0.55	140
micro avg	0.45	0.45	0.45	500
macro avg	0.36	0.36	0.35	500
weighted avg	0.44	0.45	0.43	500

	precision	recall	f1-score	support
0	0.33	0.66	0.44	144
1	0.30	0.03	0.06	187
2	0.08	0.17	0.11	40
3	0.65	0.51	0.57	129
micro avg	0.35	0.35	0.35	500
macro avg	0.34	0.34	0.29	500
weighted avg	0.38	0.35	0.30	500





The above figures shows us the precision and loss of our models using the SMOTE model as well. As we can observe, the train and validation(test) converges in same directions which is a good measure on checking that our model is not overfitting.

5.0 Conclusion

Given the right computational resources, we believe an improvement can be achieved if we rigorously trained with all of the given datasets using a combination of pre-trained embeddings and more values for our grid search hyperparameters tuning with several architectures.