COL870: Assignment 1

Kartikeya Badola and Suraj Joshi

2018EE10221

2018MT10045

ABSTRACT

In part 2 of the assignment we implement LSTM to perform NER tagging on the GMB dataset from scratch and observe the impact of Layer Normalization, GloVe embeddings, character Level embeddings and CRF on the performance. We provide details of the results obtained and compare the variants of the models in both parts providing the reasoning for our observations.

Keywords: LSTM, NER-Tagging, GloVe, CRF

1 LSTM WITH LAYER NORMALIZATION AND CRF

1.1 NER Tagging with Bi-LSTM

We have implemented Bi-LSTM using the architecture from the paper by Lample et al. 2016 [1] from nn.LSTM. We also implemented our own version of Layer Norm, LSTMCell with layer normalization included and an LSTM using layer norm LSTMCell for layer normalization tasks.

We first develop the word (and character if required) vocabulary. Words that appear infrequently are labeled with the UNK tag. We give the UNK token to all words which have term frequency less than 3 and all characters which have term frequency less than 2 in our training set.

Every sentence is converted into a sequence of indices based on the using the word vocabulary. For character LSTM, we represent each word as a sequence of indices using the character vocabulary.

The model receives these sequences as input and gets initialized with pre-trained weights if necessary. We have also implemented CRF layer (using Viterbi Algorithm) which is used if CRF is set to 1.

We trained this model with Adam optimizer and Cross Entropy Loss. We experimented with SGD obtained poor results. We found [2] where it is stated that SGD is highly unstable and less performant compared to Adam for sequence labeling tasks which is why we decided to stick with Adam optimizer for this experiment. We also observed smoother curves and faster convergence using Adam which made a proper comparision between model variants possible.

For Adam we used lr=0.001, betas=(0.9, 0.999), eps=1e-08. We used gradient clipping of 5. We use dropout of 0.5 after the nn.Embedding layer. If using GloVe, we initialize our embedding layer with vectors from the GloVe representations of words in our vocabulary. Word representations built using output from character LSTM are concatenated and then sent through the word Bi-LSTM.

1.1.1 Simple Bi-LSTM

We used the simple Bi-LSTM model with 1. Randomly initialized word embeddings and 2. Pretrained GloVe word embeddings.

The results from randomly initialised word embeddings sets a benchmark to observe improvements. For the pretrained word embeddings we isolated the word embeddings for the vocabulary of our model from GloVe.

It is expected over the course of training that these pretrained embeddings will be fine-tuned for the task at hand, and the pretraining will enhance the model's ability at the task compared to the randomly initialised model.

We observe from the plots (on the next page) that GloVe embeddings indeed improve the performance of the model compared to the randomly initialized version. The explanation is that for a large portion of the vocabulary the dense representations are close to optimal from the beginning of the training when using GloVe and hence the majority of the training time is spent fine tuning the close to optimal weights, whereas the randomly initialized model is trying to find good weights for the entire vocabulary throughout. This is why the performance curves using GloVe are higher from starting epochs.

1.1.2 Using Character Level Features

We create character level embeddings as described in Lample et al. and use an additional Bi-LSTM which are then concatenated with the pre-trained GloVe word embeddings.

Observations: We plotted accuracy, micro F1 and macro F1 for the train, dev and test splits of the GMB dataset. The addition of character level features improves all the training metrics. Character LSTM adds suffix and prefix information in the word representation hence improving performance throughout.

1.1.3 Layer Normalization in LSTM

We implemented our own classes of the Layer Norm layer, the LSTM Cell with Layer Norm, and of a class which implements the complete Layer Norm LSTM. We also used character level features and pretrained GloVe embeddings in this part.

Layer Normalization trains the models to the same extent as unnormalized variants in fewer epochs. While the training curves are slightly lower than unnormalized variants, the performance on the test and validation sets are almost equal, if not better. We also notice that in the starting epochs, the test set performance when using layer norm is significantly higher than other variants. This suggests that layer norm is acting more as a regularizer in our models.

1.2 Linear Chain CRF

We implemented linear chain Conditional Random Field (CRF) using the Viterbi Algorithm. Using the work of Lample et al. we supplemented our existing Bi-LSTM models with CRF layer.

Loss Function and Viterbi Algorithm

CRF gives the conditional probability of a labeling y given an input sequence x.

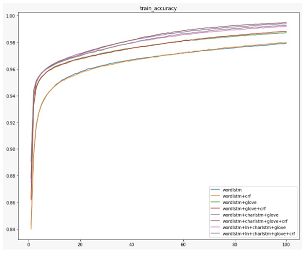
- We use the score, loss function and decoding output used in Lample et al.
- The score used is the sum of transition probabilities over the predicted labels given the input and score (outputted by the Bi-LSTM) of each of the predicted labels of the sequence.
- Since the scores involve transitions between pairs of elements dynamic programming can be used to calculate these scores and indeed this is exactly what is done in the Viterbi algorithm which we use for the same purpose in our CRF.
- Softmax of these scores over all possible label sequences for input x gives the probability of individual label y.
- The loss function is the log of the softmax probability of the score for each label y.
- While decoding we predict the output sequence that obtains the maximum score.

We observe from the plots that CRF is very powerful and shows improvement in all performance statistics, accuracy, micro F1 and macro F1 for all the splits i.e. train/dev/test over the corresponding non-CRF counterparts.

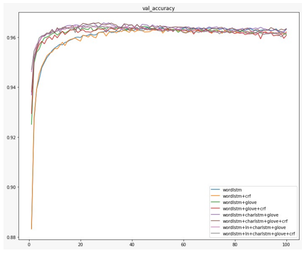
The reason for this is that CRF allows us to model dependencies within the sentence which are important for the NER task as meaningful labellings of sentences are constrained by the grammar of the language, regardless of the source of the sentence. Hence CRF improves every model it is applied to.

Initialising CRF

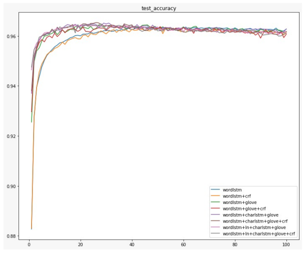
One possible initialisation of the CRF could be with the transition probabilities and labels from the transitions observed from the training set (assuming they are correctly labeled) by counting the transitions. This provides an accurate initial estimate of the values which may be fine tuned during training.



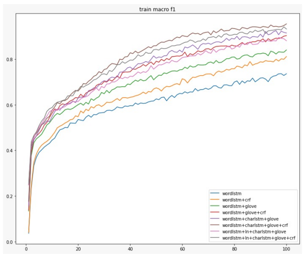
(a) Train Accuracy



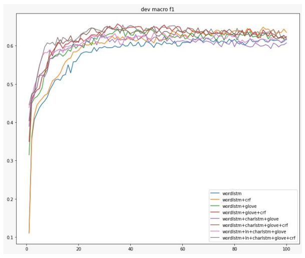
(b) Dev Accuracy



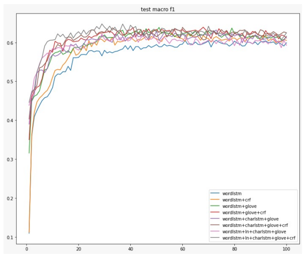
(c) Test Accuracy



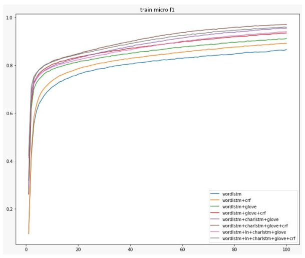
(d) Train Macro F1



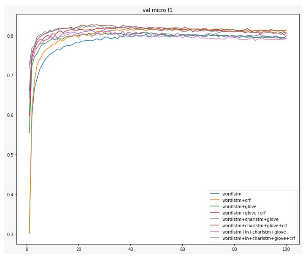
(e) Dev Macro F1



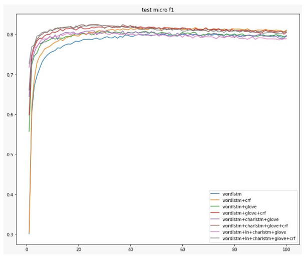
(f) Test Macro F1



(g) Train Micro F1



(h) Dev Micro F1



(i) Test Micro F1

Figure 1. Performance Statistics of Models in Part 2

The expected observation would be that adding any of Layer Normalization, pretrained word embeddings, character level embeddings or CRF would improve the performance of the basic Bi-LSTM model. A combination of these would be expected to be even more performant.

Test Macro F1 is the most important statistic as our dataset is extremely imbalanced as can be observed from the support information in the Classification Report (Figure 14). From the tables it can be observed that the order of the performance according to test macro F1 ordered in the convergence is as follows:

1. part_2.2_crf

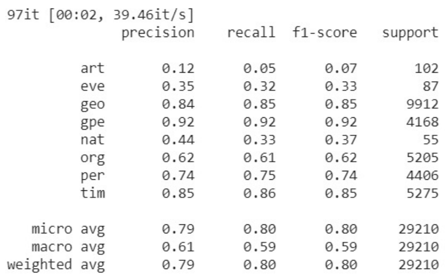
2. part_2.1_ln

3. part_2.1_char

4. part_2.1_glove

5. part_2.1_random

Classification Report Part 2

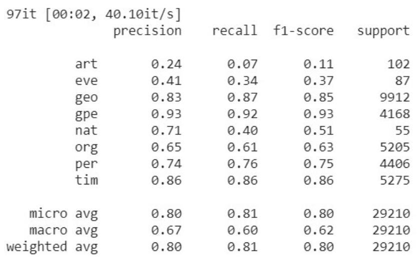


```
97it [00:02, 39.46it/s]
              precision    recall  f1-score   support

         art       0.12      0.05      0.07       102
         eve       0.35      0.32      0.33        87
         geo       0.84      0.85      0.85      9912
         gpe       0.92      0.92      0.92      4168
         nat       0.44      0.33      0.37        55
         org       0.62      0.61      0.62      5205
         per       0.74      0.75      0.74      4406
         tim       0.85      0.86      0.85      5275

   micro avg       0.79      0.80      0.80     29210
   macro avg       0.61      0.59      0.59     29210
weighted avg       0.79      0.80      0.80     29210
```

(a) Random Embeddings

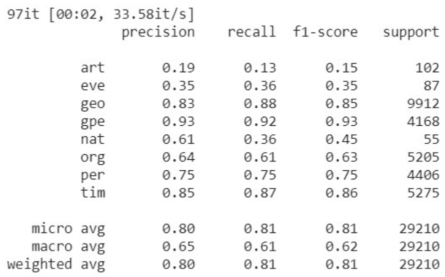


```
97it [00:02, 40.10it/s]
              precision    recall  f1-score   support

         art       0.24      0.07      0.11       102
         eve       0.41      0.34      0.37        87
         geo       0.83      0.87      0.85      9912
         gpe       0.93      0.92      0.93      4168
         nat       0.71      0.40      0.51        55
         org       0.65      0.61      0.63      5205
         per       0.74      0.76      0.75      4406
         tim       0.86      0.86      0.86      5275

   micro avg       0.80      0.81      0.80     29210
   macro avg       0.67      0.60      0.62     29210
weighted avg       0.80      0.81      0.80     29210
```

(b) GloVe Embeddings

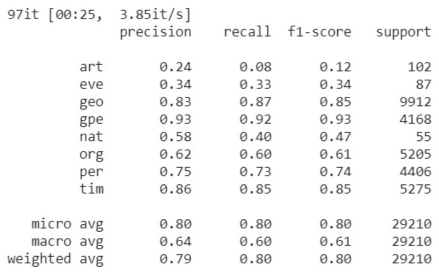


```
97it [00:02, 33.58it/s]
              precision    recall  f1-score   support

         art       0.19      0.13      0.15       102
         eve       0.35      0.36      0.35        87
         geo       0.83      0.88      0.85      9912
         gpe       0.93      0.92      0.93      4168
         nat       0.61      0.36      0.45        55
         org       0.64      0.61      0.63      5205
         per       0.75      0.75      0.75      4406
         tim       0.85      0.87      0.86      5275

   micro avg       0.80      0.81      0.81     29210
   macro avg       0.65      0.61      0.62     29210
weighted avg       0.80      0.81      0.81     29210
```

(c) Character Level Embeddings

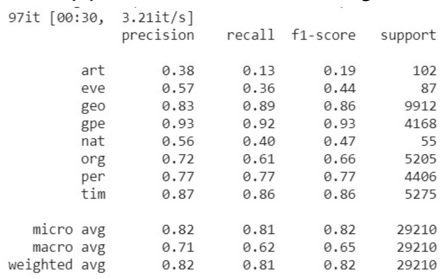


```
97it [00:25,  3.85it/s]
              precision    recall  f1-score   support

         art       0.24      0.08      0.12       102
         eve       0.34      0.33      0.34        87
         geo       0.83      0.87      0.85      9912
         gpe       0.93      0.92      0.93      4168
         nat       0.58      0.40      0.47        55
         org       0.62      0.60      0.61      5205
         per       0.75      0.73      0.74      4406
         tim       0.86      0.85      0.85      5275

   micro avg       0.80      0.80      0.80     29210
   macro avg       0.64      0.60      0.61     29210
weighted avg       0.79      0.80      0.80     29210
```

(d) Layer Normalization



```
97it [00:30,  3.21it/s]
              precision    recall  f1-score   support

         art       0.38      0.13      0.19       102
         eve       0.57      0.36      0.44        87
         geo       0.83      0.89      0.86      9912
         gpe       0.93      0.92      0.93      4168
         nat       0.56      0.40      0.47        55
         org       0.72      0.61      0.66      5205
         per       0.77      0.77      0.77      4406
         tim       0.87      0.86      0.86      5275

   micro avg       0.82      0.81      0.82     29210
   macro avg       0.71      0.62      0.65     29210
weighted avg       0.82      0.81      0.82     29210
```

(e) Conditional Random Field

Figure 2. Performance Statistics of Models in Part 2

We observe that the final test performance on all the non-CRF models is quite similar, although the order is the expected order, we think that this is because the data is highly imbalanced and hence normal cross entropy is not a very good measure for working on this dataset. An alternative loss could be weighted cross entropy (where weights are inversely proportional to the number of samples in the class) or focal loss.

CRF is better able to deal with this imbalance which gives a jump in macro F1.

REFERENCES

1. https://arxiv.org/pdf/1603.01360.pdf
2. https://arxiv.org/pdf/1707.06799.pdf