Assignment 1: POS Tagging using Neural Architectures

Sandeep Kumar Kushwaha (sandeep.kushwaha@studio.unibo.it),

Zarmina Ursino (zarmina.ursino@studio.unibo.it),

Jyoti Yadav (jyoti.yadav@studio.unibo.it)

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Abstract

In this assignment, we are performing the Sequence Labelling of Part-of Speech (POS) tagging over the dependency treebank dataset, using neural architectures. In the end, we are going to do the analysis of the loss function, accuracy and f1-macro metrics based on four different models by tuning the hyperparameters (two batch_size) values. We will consider f1-Macro as our final metrics to evaluate the best two models in the end.

1. Data Preparation & Exploration

We downloaded and extracted the dataset, and we created the three different data frames *df_train*, *df_val* and *df_test*, [Figure 1] whose features are 'tokens' and 'tags' are the targets, we ignored the third last value. After creating the dataset, we did data exploration where we checked about the most frequent tokens and most frequent tags by using a *nltk* method, 'FreqDist()' [*Figure 2 and 3*] and we found out that the most common token value is '*departments*'.

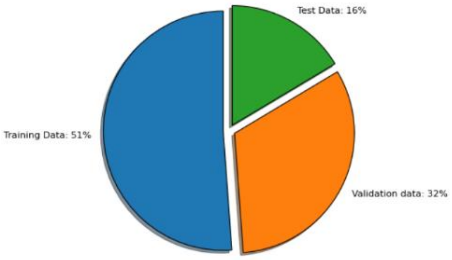

Figure 1. Data splitting

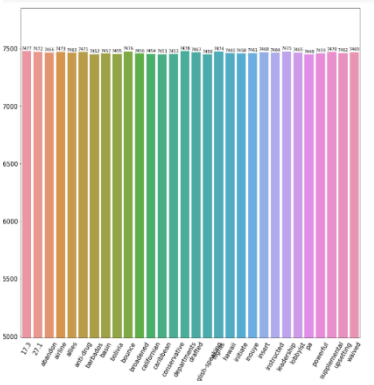

Figure 2. Frequency Distribution of words

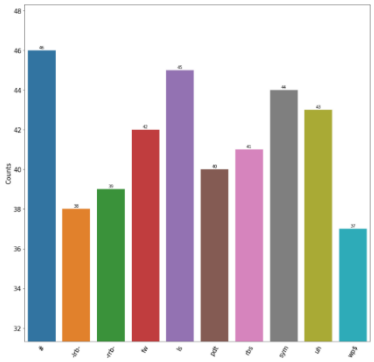

Figure 3. Frequency Distribution of tags

2. Building Vocabularies and Embedding models

Before building the vocabularies, first we created the embedding model with GloVe ('Global Vectors for Word Representation'). We decided to go for pre-trained GloVe ones with dimension of 100. GloVe allows us to take a corpus of text, and intuitively transform each word in that corpus into a position in a high-dimensional space. This means that similar words will be placed together. To be specific we used a pre-trained word embedding: the pre-trained Wikipedia Glove Embedding. From here, the steps proceed as follows:

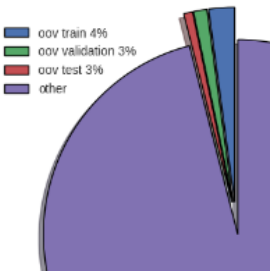

Figure 4. OOV Distribution

1. Building separate vocabularies ('*build_Vocabulary()*' function)
2. Combining the previous vocabulary with the new one & checking for OOV terms ('*combine_Vocabularies()*', function)
3. Converting tokens and tags into categorical ('*to_categorical()*')
4. Inverting the vocabulary for the tags ('*inverting ()*' function)
5. Creating a co-occurrence matrix to handle the OOV terms ('*co_occurance_count()*' function)
6. Creating an embedding model by handling the OOV words ('*build_embedding_matrix()*' function)
7. Embedding the model with the unknown & padding pad vectors and obtaining the final embedding matrix (done with NumPy concatenation)

All these steps are repeated for the training, validation, and test set. Furthermore, we explored the distribution of OOV terms by calculating the percentage with respect to the previous pre-trained vocabulary [*Figure 4*].

3. Neural Model

Recurrent Neural Networks (RNNs) process the input information sequentially and they perform the same calculation for each token in the sequence considering the previous results. We handled our dataset using the DataLoaders which implements the batch creation, padding and shuffling. We implemented our models using the parent class 'RNN' and we implemented the child classes:

1. BiLSTM_BiLSTM → Two Bidirectional LSTM layer + FC softmax
2. BiLSTM → Bidirectional LSTM + FC softmax
3. BiGRU → Bidirectional GRU + FC softmax
4. GRU → GRU + FC softmax

Case	Batch_size	Learning_rate	max_epochs
a)	64	0.01	100
b)	128	0.01	100

Table1. Tuning of batch_size hyperparameter

It implements the input layer, embedding layer, loss function, weights, and metrics. The input and embedding layers are shared among all the models. The hyperparameters used are '*batch_size*', *max_epochs*, '*learning_rate*', '*latent_dim*' (the number of nodes used as input of the generator) and the model applied. After analysing We considered followings as fixed: *latent_dim* = 64, *max_epochs* = 100 and *sequence* = 128. We tuned the *batch_size* with the values in the *Table1* (row highlighted in green results in the best obtained model). The loss we use for these models is sparse_*categorical_crossentropy* masked to exclude paddings and punctuations. The training loop implements early stopping. We computed the validation on every 5th epoch and saved the model when optimum epoch was obtained.

4. Results for Training and Validation set

For both cases [*Table 1*], the loss, the accuracy and the f1 curves during training are well performed by all the four models. Therefore, we analysed the results for the validation set. For the **case a)** we plotted our results [*Figure 5*] and, according all the movements curves w.r.t the epochs, we decided that the best models are BiGRU and BiLSTM. Instead for **case b),** we observed that the best models to evaluate are BiGRU and BiLSTM_BiLSTM [*Figure 6*].

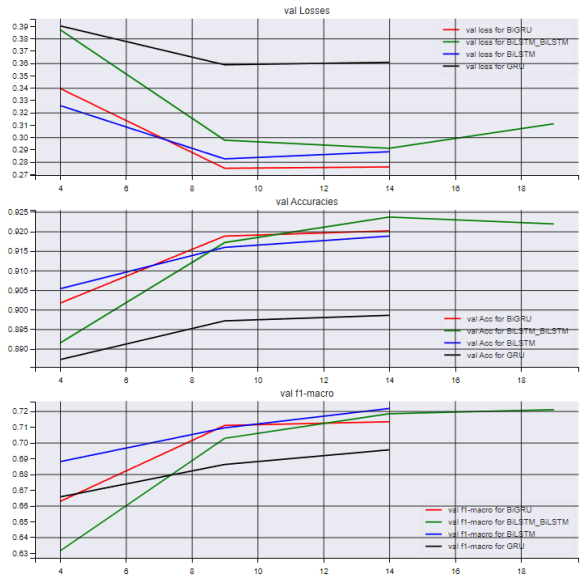

Figure 5. Validation subplots for losses, accuracies and f1-macro considering case a)

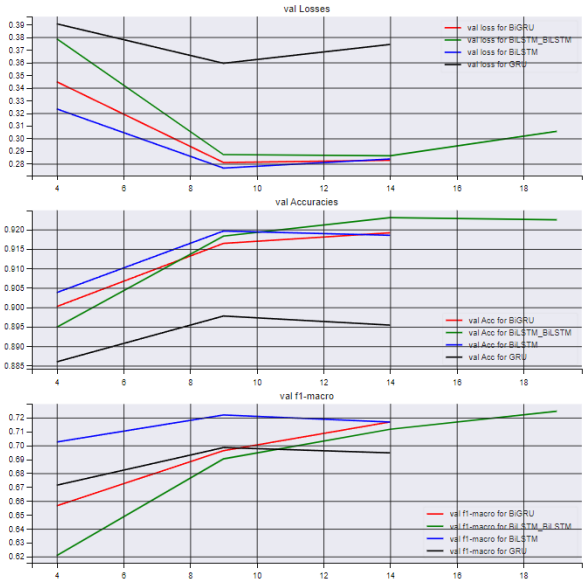

Figure 6. Validation subplots for losses, accuracies and f1-macro considering case b)

5. Best two Model evaluation for case a) and b)

We implemented the test models with the best ones of case a) and case b), we plotted the corresponding confusion matrices and analysed the values that contribute the most to the '*False Positive*' calculation [*Table2*]. In fact, in the table we indicated the values greater or equal than 10 and they are given in round brackets for each entry label, which is highlighted in black bold.

Case a)		**Case b)**	
BiGRU (Loss=0.2521, acc=0.9293, f1-macro=0.6794)	**BiLSTM (Loss=0.2482, acc= 0.9282, f1-macro=0.6816)**	**BiGRU (Loss =0.259, acc=0.9238, fi-macro=0.6958)**	**BiLSTM_BiLSTM (Loss=0.262, acc=0.9263, f1-macro=0.702)**
'nnp' gets confused with 'nns'(12) 'jj' (45) and 'nn' (106)	**'nnp'** gets confused with 'jj' (37) and 'nn' (85)	**'nnp'** gets confused with 'jj' (33) and 'nn' (82)	**'nnp'** gets confused with 'jj' (48) and 'nn' (105)
'nns' , with 'nn' (17)	**'nns'** with 'jj'(10) and 'nn' (17)	**'nns'** , with 'nn' (45)	**'nns'** , with 'nn' (15)
'jj', with 'nn' (38), 'vbn'(11) and 'rb'(12)	**'jj'** with 'nn' (38) , 'vbn' (12) and 'rb' (8)	**'jj'**, with 'nn' (64)	**'jj'**, with 'nn' (32) , 'rb' (11)
'nn', with 'vbg' (18)	**'nn'**, with 'vbg' (10)	**'nn'**, 'in' (11) with 'vbg' (14)	**'nn'**, with 'vbg' (10)
'in' with 'rb' (18) and 'rp' (16)	**'in'** with 'rb' (29) and 'rp' (14)	**'in'** with 'rb' (25) and 'rp' (19)	**in'** with 'rb' (32) and 'rp' (19)
'vbd', with 'vbn' (12)	**'vbd'** with 'vbn' (41)	**'vbd'**, with 'vbn' (17)	**'vbd'**, with 'vbn' (32)

Table2. Analysis of Confusion Matrices

We can conclude that best case is b) based on f1-macro and the best models found are BiLSTM_BiLSTM & BiGRU and we fixed hyperparameters as below Batch_size – 128, Learning_rate – 0.01 ,max_epochs- 100. We do not make further changes to the model hyperparameter. Therefore, increasing the *batch_size* to 128 helped us efficiently to optimise the model for our use case where the losses reduced & the accuracy got enhanced

References

- Most of the code blocks were taken from a notebook as part of an assignment for the NLP course [Tutorial 1 and 2 from 'Virtuale' platform]
- *Using Recurrent Neural Networks for Part-of-Speech Tagging and Subject and Predicate Classification in a Sentence*, https://www.atlantis-press.com/journals/ijcis/125941274/view
- *A Survey on Recent Advances in SequenceLabeling from Deep Learning Models*, https://arxiv.org/pdf/2011.06727.pdf
- *Part-of-Speech Tagging with Bidirectional Long Short-Term MemoryRecurrent Neural Network*, https://arxiv.org/pdf/1510.06168v1.pdf
- *Building RNN, LSTM, and GRU for time series using PyTorch*, https://towardsdatascience.com/building-rnn-lstm-and-gru-for-time-series-using-pytorch-a46e5b094e7b
- *Confusion Matrix for Your Multi-Class Machine Learning Model,* https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826