NER TASK

Francesco Peracchia 1906895

Abstract

Name Entity Recognition (NER) is a task composed of locating and classifying named entities $tab.1$, in this paper are proposed and evaluated some possible approaches for this problem. A final model is hence proposed, achieving an f1 score of 0.6442, then a search strategy is used, in order to avoid inconsistency and exploit all the relationships present in the problem description.

1 Data processing

The dataset is first processed to extract all the labels that are present in it, finally is created a dictionary that that converts the correspondent token into index, this index refers to the token position in the dictionary. Hence, because were used different embedding in this phase depending on the experiment conducted different dictionaries were used. In order to represent each token in its vectorial form, 4 different approaches are used. A trainable embedding layer, a frozen layer that embed token into vectors by using Glovo pre-trained weights, a similar solution with Word2vec weights, and finally the last embedding model that combines both results from Glovo and Word2vec.

1.1 Learnable embedding layers

Using a lookup table where are stored both embeddings vectors and a dictionary is possible to build a new representation for each token, as alternative to one-hot-encoding or BAG, in this way each token is then translated into vector. Embedding size for this module was chosen in a range between 50 and 200, however, whatever further classifier was used in the forward step, F1 score never exceeded 0.2. These poor results are mainly caused by the quality of the embedding layer, to increase our results, some alternatives are proposed in the following section.

Labels	
O	-
Corporation	B-CORP
Corporation	I-CORP
Group	B-GRP
Group	I-GRP
Person	B-PER
Person	I-PER
Creative Work	B-CW
Creative Work	I-CW
Product	B-PROD
Product	I-PROD
Location	B-LOC
Location	I-LOC

Table 1: Labels for NER task are divided in 6 semantic areas plus a O:Outside, after each must be Labelled in BIO format for Beginning, Inside, and Outside.

1.2 Pre-trained Weights

Rather than learning from scratch a new model for embedding, some pre-trained solutions could be used, GloVe or Global Vectors for word representation could help us to provide our model with a good input representation. Each token is then converted by using a previously built dictionary over GloVe, this dictionary associates indexes with weights, to handle padding and also words that could be missing the dictionary is augmented with two indexes its weights for $< PAD >$ and $< UNK >$, this solution again present different experiments with different dimensions, from 50 to 300. Alternatively, Word2vec weights are used in a similar solution, tokens are fist converted in an index, and then using a look-up table converted with a vector of dimension 300. Finally, another solution is hence proposed by combining both GloVe300 and Word2vec300, by concatenating the Latent Features extracted in output from LSTMs networks.

Layer	Input dim	Output dim
Embedding*	(32,*)	(32,*,Emb)
LSTM*	(32,*,Emb)	(32,*,2*Emb)
Linear	(32,*,2*Emb)	(32,*,Emb)
Dropout	(32,*,Emb)	(32,*,Emb)
Linear	(32,*,Emb)	(32,*,13)
Softmax	(32,*,13)	(32,*,13)

Table 2: **Model A :** LSTM* is a RNN composes of two layers of LSTMs, with a dropout layer between the LSTMs with p=0.3, Dropout between the Linear layers has probability p=0.3, Embedding* dimensions depends on with pre-trained weights file are used (50,100,200,300).

2 Model

In order to classify each entity in one of the 13 labels $tab.1$, after some experimental trials, it was decided to use a Bidirectional LSTMs rather than a normal LSTMs. Bi-LSTM are RNN capable to exploit dependencies in both directions and dealing with vanishing and exploding gradient problems by using input and forget gates. LSTM's out hidden size is equal to the input size, in this way we are feeding the successive Head, a NER classifier, with an enough large description of the dependencies that are affecting that token, referring to all the elements in the sentence. Initially, a single layer RNN composed of Bi-LSTMs was designed, the latent space that we have in output from the Bi-LSTMs was better exploited by the classifier if the RNN was composed of two layers of Bi-LSTMs with a dropout layer in the middle capable to reduce overfitting and increase the latent feature generalization. The previous structure is the backbone of the model and remains the same for each experimental result after being proposed, of course changing the Embedding dimension will influence also the obtained dimensions for the LSTMs and the successive classifier, this model is fully presented in $tab.2$, further as was already introduced before a combination of Word2vev and GloVe embedding was used, rather than use a unique backbone here are used two different backbone trained individually from Glove300 and Word2vec300, then LSTMs for GloVe300 and LSTMs for GloVe300 outputs are concatenated doubling the dimension of the latent space features used as input for a new Head, this Classifier results in a more Large and Deep network composed by linear and dropout layers $tab.3$

3 Training

3.1 Model A

Once the Dataset is loaded and charged by a Dataloader, using custom dictionaries for each embedding modalities, all the sentences, in each batch, are transformed from an indexed representation to speed up the successive process of embedding. Pre-trained weights and Dictionaries are processed and stored by using numpy utilities, finally accessing the correspondent element each index in converted in the linked vector. Training and Validation Dataloader are finally used to train the Head weights for all the experiments. As Loss is used Cross-Entropy Loss, SGD with lr=0.1 and momentum=0.9 as optimizer. After each epochs, the entire validation dataset is forwarded, in order to have immediate complete feedback on the real model's performance.

3.2 Model B

Differently from the previous model, model B $tab.3$ concatenate the LSTMs latent features, obtained in output from the trained model A in GloVe300 and Word2vec. In this case, the first part of the model should not be trainable, hence only the Classifier weights are changed accordingly with SGD algorithm while the LSTMs for both the embedding are frozen.

4 Search Strategy

In order to exploit the intrinsic constraints of NER task, i have developed a Search strategy to explore the output distribution received from the model rather than just return the index with max probability. Bi-LSTM already linking very well the dependencies between chosen token's labels, but in order to completely avoid inconsistent solution, is possible to explore the distributions and add some hard constraints. Reversing the order of the words, and then their output distribution, to process the last word before anyone else, is possible to add some fixed constraints on the next word available tag, if a word is hence classified as Inside must be followed by a word classified with Beginning or Inside of the same areas. This exploration is developed by a recursive call that at each step returns the best hypothesis tagging for the sentence, however, this is an exponential problem, and using search strategies over long sentences where for each choice The Search is speed-up by discarding

tags with probability under a certain threshold, and current hypothesis path if already with a heuristic function worst than the current max path hypothesis, this consideration, together with the fact that not all the 13 labels should be explored at each step, due to the hard constraints, limits the computational cost of the Search algorithm. However, the locally obtained results suggest a slight decrease, or no increment, in terms of accuracy and f1 score; compared with the effort necessary to compute the exploration, and with the risk of having of selecting a bad threshold and having even worst results, is possible conclude that it should not be used at least in the modalities that are above proposed. The speed-up process approximation leads the model to fail in improving the accuracy, which decreases the f1-score and accuracy, while only into few cases the corrections are acting positively.

5 Results

Model A with pre-trained weights from GloVe200 achieve over **f1 score 0.6442** and accuracy **accuracy 0.9343** $tab.5$, finally a confusion matrix for this model is below proposed $fig.1$, with the results that are shown divided for each class, the order proposed is the same as in $tab.1$, but with reverse order. Results obtained with GloVe are overcoming the result from Word2vec that seems not performing as well as GloVe for NER task, finally combining both GloVe and Word2vec hidden features are obtain the worst performance of the entire trial.

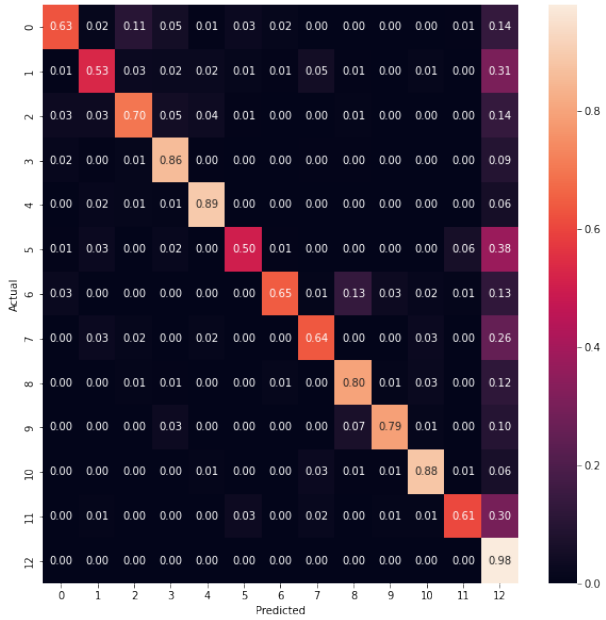

Figure 1: Confusion matrix for Model A with Glove200

6 Tables

Layer	Input dim	Output dim
Glovo300*	(32,*)	(32,*,300)
Word2vec300**	(32,*)	(32,*,300)
LSTMs1*	(32,*,300)	(32,*,600)
LSTMs2**	(32,*,300)	(32,*,600)
Concatenation	(32,*,600),(32,*,600)	(32,*,1200)
Linear	(32,*,1200)	(32,*,600)
Dropout	(32,*,600)	(32,*,600)
Linear	(32,*,600)	(32,*,169
Dropout	(32,*,169)	(32,*,169)
Linear	(32,*,169)	(32,*,13)
Softmax	(32,*,13)	(32,*,13)

Table 3: **Model B :** Vectors from Glovo300 and Word2vec300 are used in frozen LSTMs, previously trained, then both the output are concatenated and used into a new Head Classifier.

Labels	
O	10240/10427
B-CORP	133/111
I-CORP	119/95
B-GRP	192/182
I-GRP	377/355
B-PER	300/301
I-PER	329/324
B-CW	170/146
I-CW	261/226
B-PROD	149/111
I-PROD	87/82
B-LOC	243/254
I-LOC	153/137

Table 4: Labels and Prediction for each class, these results are obtained by using model A and with embedding with Glovo200 without search algorithm.

Embedding	Classifier	Accuracy	F1 score	Accuracy*	F1 score*
Glove50	model A	0.9248	0.5876	0.9131	0.5275
Glove100	model A	0.9271	0.6199	09160	0.5577
Glove200	model A	**0.9343**	**0.6442**	**0.9248**	**0.5812**
Glove300	model A	0.9260	0.6038	0.9124	0.5479
Word2Vec300	model A	0.8861	0.4375	0.9124	0.5479
Glove300+Word2Vec300	model B	0.9078	0.5078	0.8987	0.4792

Table 5: Results from different pre-trained embedding layers are compared, all the experiments were conducted over the same Classifier Head, while only the Backbone was changed, results from different embedding show us a double consideration, * means using search strategy among the distributions. First : Increasing the embedded vector dimension increases both F1 scores and accuracy until GloVe200. Second : Concatenate Glove300 and Word2vec300 embedded vectors and use them to feed a model with the same structure as before does not increase our ability to discriminate and classify chunks in NER task.