Report of the 3rd NLP Homework: Multi-inventory Word Sense Disambiguation

Manuel Prandini ID: 1707827

I. INTRODUCTION

The goal of this homework was to create and train own Word Sense Disambiguation system, choosing own model and trying to improve its performance based on f-score. I tried to draw inspiration from [1], using some models described inside and trying to perform multiple tests by varying my model in terms of training datasets, hyperparameters and layers. The built system is multitask type: it tries to disambiguate the polysemic words, predicting the right Babelnet sense for the *WSD fine-grained task*, while for the *WSD coarse-grained task*, it tries to predicts the right Wordnet Domain and the right Lexname.

II. PREPROCESSING

I used the *WSD Evaluation Framework* that contains datasets for training and testing my model, described in [2]. In particular, I chose to use only *semcor* as corpus for training and for hyperparameters tuning. I used *semeval2007* as a development set during training, while I used the remaining evaluation datasets to predict the model. From the xml files, I created two text files: the train file containing the sentences to give in input to the model and the label file that contains the sentences with polysemic words written in the format *'lemma_babelnetsynset'*. The polysemic words are marked as *'instances'* in the xml file and through the id of the given word, I found the corresponding meaning using the *'id-keysense'* map. I used Wordnet to calculate the corresponding Wordnet synset of the keysense. Then, I converted the Wordnet synset to Babelnet synset through a map and I joined it to the lemma. For the *coarse-grained WSD* task, I converted the Babelnet synsets labels using the maps *"babelnet-wndomains"* and *"babelnet-lexnames"*, to create Wordnet Domains and Lexnames labels files. When the corresponding Babelnet synset is not in the map, I used *'factotum'* for Wordnet domains, because it is the most common domain. Instead for the Lexnames, all the Babelnet synsets are in the corresponding map. Then, I organized data in array of arrays of words. I tried to clean the data in different ways: first, removing only some punctuation (leaving the *'%'* character and other particular characters), and then removing all the punctuation but they produced similar results during the training phase, so I decided to leave the data without cleaning them. Then I created the input vocabulary, the babelnet sense output vocabulary (that contains also the input vocabulary), the wordnet domains and lexnames output vocabularies (both containing only senses). Each output vocabulary has also *'PAD'* for the padding and *'UNK'* for OOV words. I used them to convert the input and the labels splitted into id tokens. I also filtered words that do not appear in the dataset more than *'x'* times. Mainly i did this, to solve out-of-memory problems, but then I discovered that Colab gave me these problems due to some too large samples that the batch-generator loaded during the fit of the model. I decided to padding sentences, using the minimum between the longest sentence in the batch and a certain threshold (i chose 50) as a reference. The sentences longer than this value have been truncated, the shorter ones have been padded. However, I filtered the words in the vocabulary to speed up the time during the training phase. I also trained the models using *'Train-on-Matic'*. I reused the code written for the second Homework to parse it, modifying the method to produce input and labels files. Moreover, I resized the file to 200,000 sentences because it was too large and I had difficulty loading it on Colab.

III. MODELS

The base model uses the Keras Embeddings and has a Dropout Layer. Then is implemented with a *Bidirectional LSTM* and has 3 Dense softmax layers as output, one for each prediction (Babelnet sense, wordnet domains, lexnames). I tried different combinations on this model. First, I added a second Bidirectional LSTM Layer. Then, I removed the second BLSTM and I added an Attention Layer composed by: a Time Distributed Dense with 1 neuron and activation function 'tanh' that takes in input the lstm output; an Activation layer of type softmax that takes in input the previous layer output; a Multiply Layer that takes in input the previous layer output and the BLSTM output. Finally, I tried to add the ELMO pretrained Embeddings to the model with Attention Layer, replacing the Keras Embeddings.

IV. PREDICT FUNCTIONS

Each prediction method creates its own vocabulary, defines the model with training parameters, and loads the weights. Input consists of array of tokens if the model is trained with Keras Embeddings, else it consists of arrays containing non-split sentences when the model is trained with ELMO Embeddings. The model prediction returns a matrix where first dimension indicates the task prediction (Babelnet, WordnetDomain, Lexname), second dimension indicates the word of the sentence predicted and the last dimension indicates the vector of probabilities produced. The output file, that has a predicted sense for each id instance, is compared with the corresponding gold file to calculate its f-score. I converted the gold file provided, replacings the sense-key with the corresponding sense to predict (Babelnet, Domain, Lexname).

A. Most Frequent Sense

I used Wordnet database to retrieve Most Frequent Sense of a lemma that isn't present in the babelnet sense output vocabulary. Wordnet returns a list of corresponding synsets, passing a lemma in input. Initially, I took only the first result of the list without checking the Part of Speech. Then, I tried to pass the POS in input to Wordnet and this has increased the f-score during the tests, because the first element of the returned synsets list is most accurate. I used MFS both when I converted the gold files, and in the various sense predictions.

B. Predict Babelnet

I retrieved from the evaluation dataset only the instances with their lemma, POS and the position in the sentence. For each instance of the sentence, I passed the input lemma to the output babelnet sense vocabulary to return the list of related senses and I inserted in a list all the corresponding id tokens. Then, I used the list of id tokens to take only the corresponding values from the probability vector, that i retrieved in the predict matrix, using the position of word in sentence. Then, I did the argmax on them and I got back the synset, through a reverse 'token-babelnet synsets' map, and then I associated it to the id instance. Finally, I wrote the result on output file. I used MFS when a lemma was not present in the babelnet sense output vocabulary and no synset returned.

C. Predict Wordnet Domains

First, I retrieved the babelnet senses from the babelnet sense output vocabulary and then, I converted them via the map, into corresponding domains. If the lemma did not return synset, I used MFS and I converted it to the right domain. If MFS is not present, I used 'factotum', because it is a common domain. From the retrieved domains, i toke the corresponding id tokens, and i did argmax from the softmax probability vector of a instance, only with them. Initially I did argmax on the whole vector for each instance but this always gave me 'factotum' for each instance because was the most likely class.

D. Predict Lexnames

I predicted the lexnames in a similar way to domains, using the right mapping from babelnet sense to lexnames and when a lemma returns no synsets and hasn't MFS , I attributed 'adj.all' to an instance id.

V. TRAINING AND TESTING

I did several training sessions on Colab, where each training epoch required about 10 minutes to 30 minutes. I reported only the main tests which improved f-score on *'ALL'* dataset. I tried to do a few tests with clean data, on models with randomly chosen parameters, but the f-score was lower than 2.0% compared to tests with dirty data. The hiperparameters used are described in the TABLE I. I started with base model, using the input vocabulary filtered with only words that appear at least 5 times or more in the input vocabulary, all the Babelnet senses vocabulary, 200 of embedding size, hidden size 128, input dropout 0.4, LSTM dropout 0.4, the learning rate at 0.015, 128 of batch size and 5 epochs and this produced an f-score of 51.6% for Babelnet sense, 80% on Wordnet domains and 66.7% on Lexnames. I then tried to make the network deeper, because the model was underfitting and still had to learn. So I set the hidden size to 256 and I had an increase of the f-score on the Babelnet sense of 0.6%. Then, I changed the learning rate (which is the first parameter that radically changes the performance of the training) by setting it to 0.0015, 64 of batch and I filtered the sense vocabulary by putting only the terms that appeared 5 or more times and the f-score it had significant increase of 6.0%, reaching 58.2%, Fig 2. This performance increase is due to the filtering of vocabularies rather than the hyperparameters tuning. I did other tests with this model trying to set other parameters observing the graph performances, but this not improved the f-score, so i reset last parameters. Then, I tested the double BLSTM with 6 epochs, trying to avoid overfitting, but this produced a f-score of 56,8%, worsening by 1,4%. The double BLSTM was not very efficient, so I added the Attention Layer to the single BLSTM. I tried with same hyperparameters but this has increased the f-score only by 0.4%. So I set epochs to 8, 32 of batch size, dropout input to 0.6, lstm dropout to 0.2, 0.2 of learning rate (the loss grows up using ADAM optimizer with higher values of learning rate), and I not filtered the babelnet sense vocabulary, but this has little increased the f-score by 0.2%. After several tests I found the right parameters that increased f-score up to 59.3%. To obtain this, I increased embedding size up to 400, I decreased epochs to 5 and I set the input dropout to 0.5. I tryed to set less epochs but I got a minus 0.1%. Instead, I got a minus 0.7% when i set the batch at 64 and I filtered the words of the input vocabulary and babelnet sense, using only words that appear 3 or more times. I got 63.5% predicting only MFS for each instance, so I had to improve the model that until now, given me lower f-scores. So I tried pretrained ELMO Embeddings with Attention layer. I did less training because each epoch took 20 minutes to half an hour, due to the preprocessing on the whole sentences passed in input. Moreover, it isn't possible to do long training sessions, because Colab aborts them. With this change, I got 64.4% on babelnet, 84.8% on domains and 77.6% on lexnames using the same parameters used in the best model with Keras Embeddings and the Attention layer, Fig 3. I tried to perform the hyperparameters without success. So I tried to train this model with TOM dataset using only 50,000 phrases from the file that contains 200,000 sentences. I noticed that the validation steps were much less than the training steps, so I decided to add more. Furthermore, I changed *'early stopping'*, setting it on the Babelnet loss validation, instead of generic loss, because this last one is influenced by the overfitted results obtained by lexnames and wordnet domains. Using only TOM, I got 61,4%. The portion of the dataset used had less Babelnet sense than all semcor. Finally, I did also a final test with this model using both semcor and TOM. I decided to filter the babelnet sense dictionary less by entering only the senses that appear 3 or more times instead of 5. I lowered the learning rate to 0.0008. This produced the same result for the evaluation dataset *'ALL'*, ie 64.4%. The TABLE II, III, IV summarizes the values obtained from the best hyperparameters configuration of each model for all the predicts.

VI. CONCLUSIONS

I noticed that the models that I trained, easily went into overfitting, in particular with general loss metric. Perhaps this depends on the influence of performance on the training of domains and lexnames: they have less and unbalanced number of output classes, compared to babelnet senses. It's difficult to find parameters that balance the training of babelnet sense, domains and lexnames, trying not to go overfitting on the one hand and not going underfitting on the other. Moreover, the f-score change is affected by other reasons, compared to the parameter tuning: it is mainly given by how the preprocessing data is made, by the layers used in the model, by how MFS is used in the predict function and also how much padding is given to the samples, because sentences longer than the chosen threshold are truncated and information is lost. The tests in the tables below, show that only the model that uses pretrained embeddings (ELMO) exceeds the f-scores of the predictions with only MFS. The f-scores obtained in the prediction are influenced by a percentage of MFS. For example, using only words that appear 5 or more times in the input and babelnet sense vocabularies, I have about 21% (1540/7253) of MFS over the total instances to predict. The TABLE II shows that in Babelnet sense prediction the fourth model and the last, have same results on 'ALL' and 'SE3'. Fourth model gets a good result on the prediction of *SE07*, although it is difficult to obtain, because in the other tests I always got lower results, so i chose it how best model for this prediction. Instead, the best result on Wordnet domains is obtained by the last model, as shown in the TABLE III, and it exceeds the model trained only with semcor of 0.1% on ALL. Finally, in TABLE IV , can see that, best results for Lexnames, have been achieved by the fifth model, while the model trained with semcor obtains the second best results. Also in these tests, the model trained only with semcor is exceeded by 0.1% on ALL. Observing the results carefully and considering the little importance of model overfitting, I chose the model with attention layer and ELMO as best, Fig 1, where hyperparameters and metrics are described in TABLE V. It would be interesting to carry out longer training sessions with more powerful hardware resources to be able to use more training data, given Colab's limits.

Training Hyperparameters	
EMBEDDING SIZE	200 — 300 — 400
HIDDEN SIZE	128 — 256 — 512
INPUT DROPOUT	0.2 — 0.4 — 0.5 — 0.6
LSTM DROPOUT	0.15 — 0.2 — 0.4 — 0.6
BATCH SIZE	32 — 64 — 128
LEARNING RATE	0.015 — 0.001 — 0.0015 — 0.0008
EPOCHS	3 — 4 — 5 — 6 — 8
FILTER INPUT VOCAB	0 — 5
FILTER BN VOCAB	0 — 3 — 5

TABLE I: Different values of hyperparameters used during training phase. The embedding size is used only with Keras Embeddings. The filter in the two vocabs indicates to take only words that appear at least as much as the indicated value. I used Hidden size to 512 only few times, because gave me 'oom' problems on Colab.

Babelnet F-score results						
MODELS	**SE07**	**SE13**	**SE15**	**SE2**	**SE3**	**ALL**
blstm (one layer)	52.7	56.6	54.1	61.8	58.8	58.2
blstm (2 layers)	52.5	56.3	53.2	59.9	56.5	56.8
blstm (Attention Layer)	52.5	56.8	55.8	62.8	61.0	59.3
blstm (with att and ELMO)	**58.0**	61.7	59.6	**68.4**	**66.1**	**64.4**
blstm (with att, ELMO, TOM)	52.1	61.6	58.0	63.9	62.2	61.4
blstm (with att,ELMO, concat)	56.7	**61.9**	**60.7**	67.9	**66.1**	**64.4**
MFS	54.9	62.7	61.0	65.1	65.6	63.5

TABLE II: Best F-score results obtained for Babelnet senses from each different model. with *concat* i mean the dataset concatenation with semcor and TOM. The first three models use Keras Embeddings. The first four models use semcor as training dataset. Best results obtained for each dataset are shown in bold.

Wordnet Domains F-score results						
MODELS	**SE07**	**SE13**	**SE15**	**SE2**	**SE3**	**ALL**
blstm (one layer)	84.6	73.2	77.4	88.1	84.8	82.1
blstm (2 layers)	86.6	72.7	78.9	89.6	84.2	82.7
blstm (Attention Layer)	87.5	73.2	81.2	89.6	84.6	83.3
blstm (with att and ELMO)	88.1	77.2	80.1	**90.9**	85.8	84.8
blstm (with att, ELMO, TOM)	85.9	74.2	80.7	89.1	**86.2**	83.6
blstm (with att,ELMO, concat)	**88.4**	**77.6**	**82.4**	90.3	85.4	**84.9**
MFS	88.4	76.5	82.2	90.2	88.2	85.3

TABLE III: Best F-score results obtained for Wordnet Domains from each different model. with *concat* i mean the dataset concatenation with semcor and TOM. The first three models use Keras Embeddings. The first four models use semcor as training dataset. Best results obtained for each dataset are shown in bold.

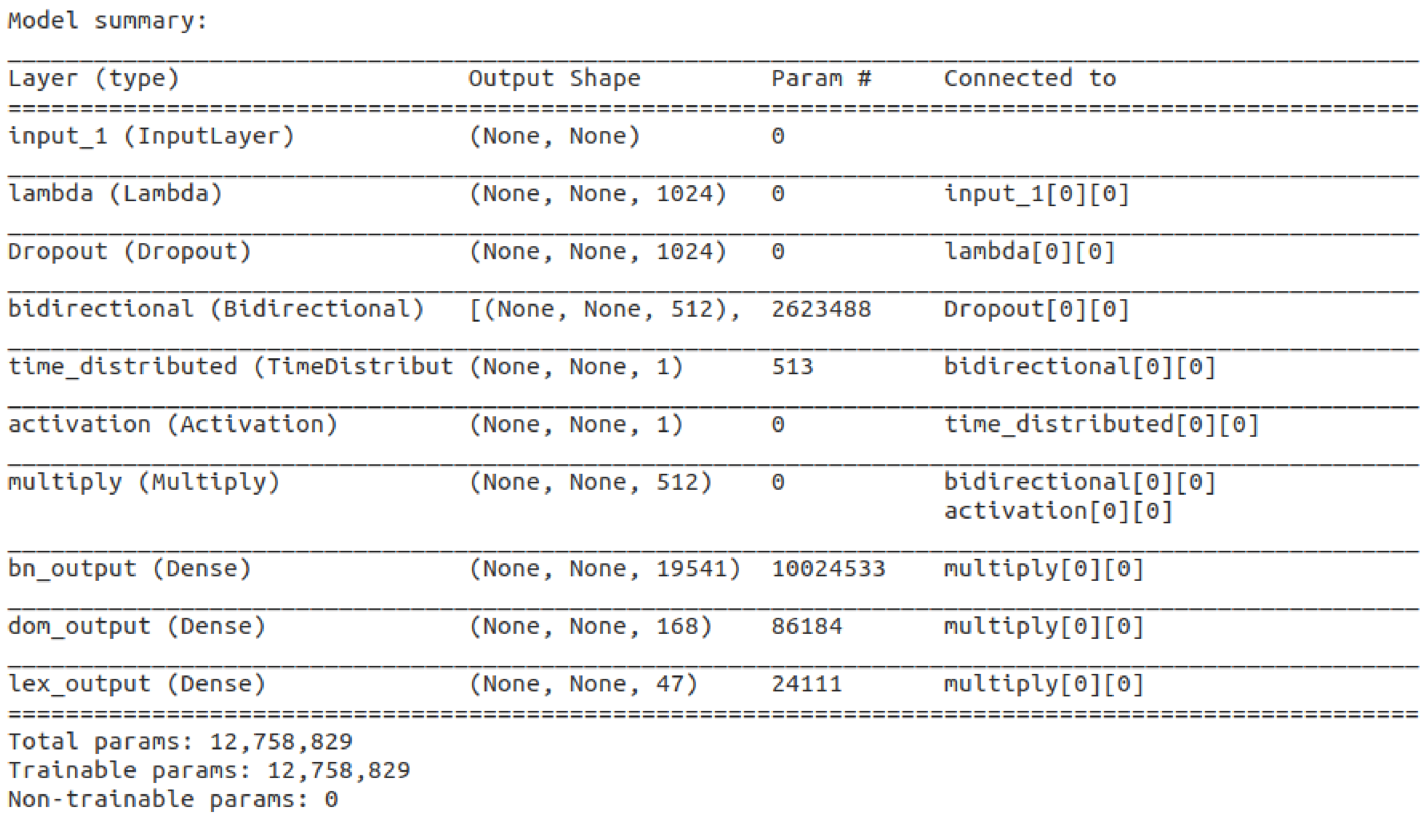
Lexnames F-score results						
MODELS	**SE07**	**SE13**	**SE15**	**SE2**	**SE3**	**ALL**
blstm (one layer)	68.8	65.8	70.0	78.6	72.9	72.4
blstm (2 layers)	62.4	65.0	66.7	77.3	71.3	70.6
blstm (Attention Layer)	68.6	66.9	71.1	78.8	73.8	73.1
blstm (with att and ELMO)	**73.8**	72.8	75.2	**82.7**	77.6	77.6
blstm (with att, ELMO, TOM)	68.6	**75.4**	75.0	81.2	**79.0**	**77.7**
blstm (with att,ELMO, concat)	69.7	72.6	**75.8**	82.5	76.5	77.0
MFS	71.0	74.9	77.2	82.5	81.0	78.9

TABLE IV: Best F-score results obtained for Lexnames from each different model. with *concat* i mean the dataset concatenation with semcor and TOM. The first three models use Keras Embeddings. The first four models use semcor as training dataset. Best results obtained for each dataset are shown in bold.

Best Model Hyperparameters	
filter input vocab	5
filter bn vocab	5
hidden size	256
input dropout	0.5
lstm dropout	0.2
batch size	32
learning rate	0.001
epochs	5

Best Model Performance		
Metric	**Training**	**Validation**
bn loss	from 1.388 to 0.176	from 0.610 to 0.370
dom loss	from 0.240 to 0.140	from 0.234 to 0.221
lex loss	from 0.254 to 0.166	from 0.294 to 0.273
general loss	from 1.883 to 0.483	from 1.139 to 0.865
bn acc	from 0.820 to 0.948	from 0.904 to 0.930
dom acc	from 0.941 to 0.960	from 0.956 to 0.958
lex acc	from 0.929 to 0.950	from 0.947 to 0.949

TABLE V: In the left table are shown the best hyperparameters used to train the BLSTM with Attention Layer and ELMO pretrained Embeddings, chosen as best model. Moreover, in the right table are shown the metrics obtained starting from the first epoch up to the fifth epoch.



```
Model summary:
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, None)         0

__________________________________________________________________________________________________
lambda (Lambda)                 (None, None, 1024)   0           input_1[0][0]
__________________________________________________________________________________________________
Dropout (Dropout)               (None, None, 1024)   0           lambda[0][0]
__________________________________________________________________________________________________
bidirectional (Bidirectional)   [(None, None, 512),  2623488     Dropout[0][0]
__________________________________________________________________________________________________
time_distributed (TimeDistribut (None, None, 1)      513         bidirectional[0][0]
__________________________________________________________________________________________________
activation (Activation)         (None, None, 1)      0           time_distributed[0][0]
__________________________________________________________________________________________________
multiply (Multiply)             (None, None, 512)    0           bidirectional[0][0]
                                                                 activation[0][0]
__________________________________________________________________________________________________
bn_output (Dense)               (None, None, 19541)  10024533    multiply[0][0]
__________________________________________________________________________________________________
dom_output (Dense)              (None, None, 168)    86184       multiply[0][0]
__________________________________________________________________________________________________
lex_output (Dense)              (None, None, 47)     24111       multiply[0][0]
==================================================================================================
Total params: 12,758,829
Trainable params: 12,758,829
Non-trainable params: 0
```

Fig. 1: This figure summarized the architecture of BLSTM with Attention Layer and ELMO pretrained Embeddings, chosen as best model. *Lambda* indicates the ELMO pretrained Embeddings Layer; *time_distributed* indicates the first Dense Layer of the Attention Layer with *tanh* as activation function; *activation* is the Activation Layer that uses *softmax* function.

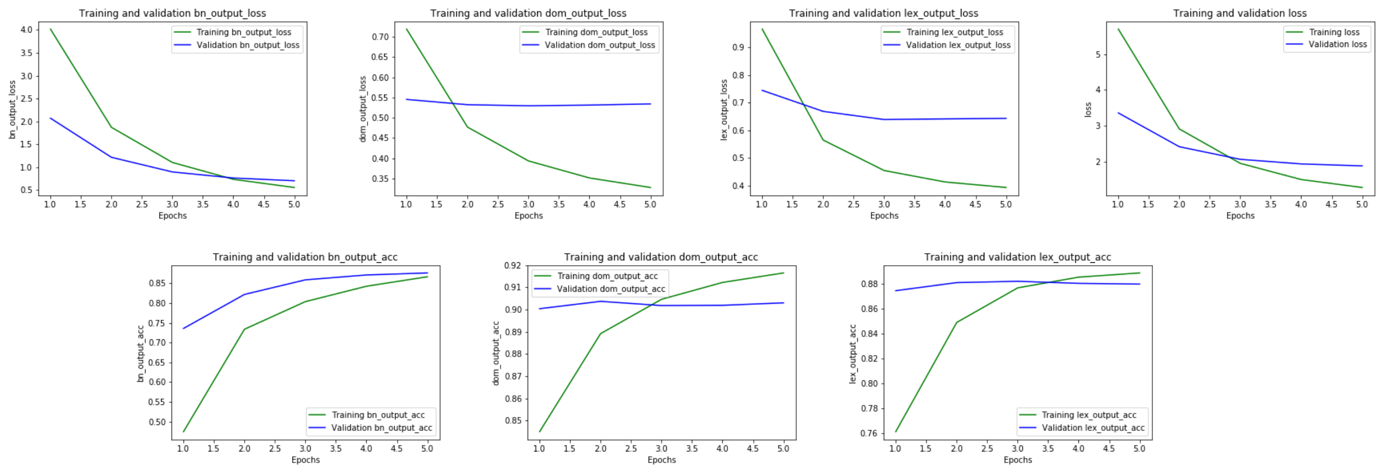


Fig. 2: In these figures are shown the training and validation metrics, respectively from top left to bottom right *bn_loss*, *dom_loss*, *lex_loss*, *general_loss*, *bn_acc*, *dom_acc*, *lex_acc*, obtained from the best hyperparameters configuration of BLSTM base model trained with Keras Embeddings. From the graphs, it is possible to see how the training and validation trends are uniform only for bn loss, while for domains and lexnames the loss are completely in overfitting, and them influence the overall loss.

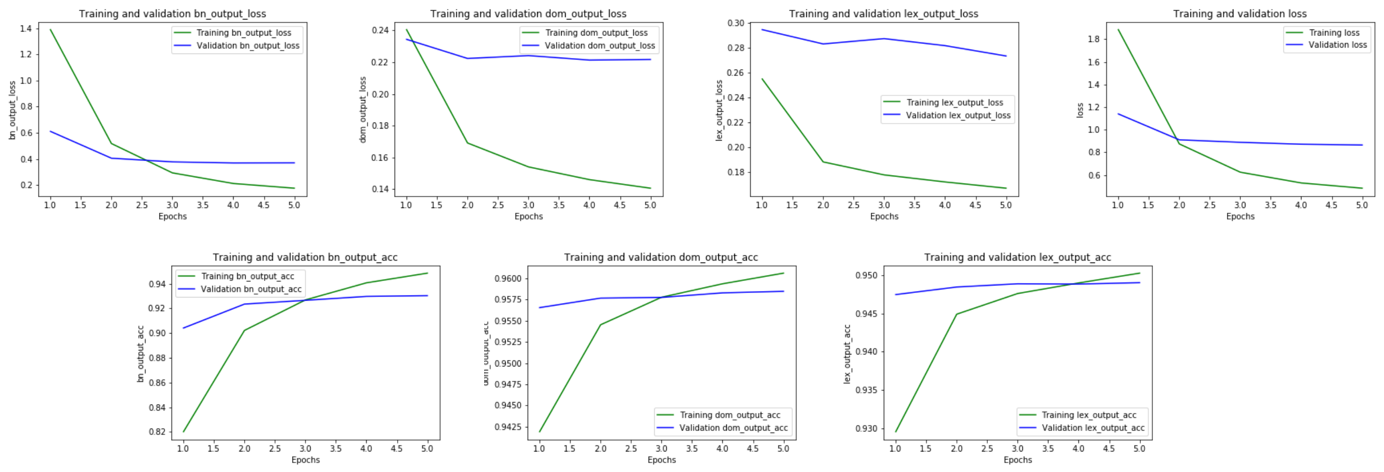


Fig. 3: In these figures are shown the training and validation metrics, respectively from top left to bottom right *bn_loss*, *dom_loss*, *lex_loss*, *general_loss*, *bn_acc*, *dom_acc*, *lex_acc*, obtained from the test of best model chosen, BLSTM with ELMO pretrained embeddings and Attention Layer. From the graphs it is possible to see that at least for the bn loss the training and validation trends are uniform even if the model is overfitted after the 3rd epoch. Domain and loss are completely overfit and this affects the overall loss. Compared to the Fig. 2, the val bn loss has gone from 0.70 to 0.37, having a f-score improvement of 6.2% on ALL, the val loss of domains goes from 0.53 to 0.22, improving only by 2.7% the f- score, and the lexnames val loss, increased from 0.64 to 0.27 which increased the f-score by 5.2%.

References

[1] A. Raganato, C. Delli Bovi and R. Navigli, Neural Sequence Learning Models for Word Sense Disambiguation, Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, sep, 2017, Copenhagen, Denmark, Association for Computational Linguistics, https://www.aclweb.org/anthology/D17-1120, 10.18653/v1/D17-1120, 1156–1167.

[2] A. Raganato, J. Camacho-Collados, and R. Navigli, Word sense disambiguation: A unified evaluation framework and empirical comparison, in Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers, (Valencia, Spain), pp. 99110, Association for Computational Linguistics, Apr. 2017.