Word Sense Disambiguation

The aim of this project was to tackle the problem of word sense disambiguation, that raises due to the intrinsic ambiguity of the lexicon. Word sense disambiguation is the task of computationally determining the sense of a word among the senses that make it ambiguous, exploiting its context. Among the tens of approaches to WSD I focused myself on the neural approach, despite some challenges, I managed to improve actively my model by performing hyper-parameters tuning, and integration with a knowledge base. I'm going to present you my work in this report: the standard architecture I implemented and the further modifications.

Neural Approach The neural approach to WSD exploits the supervised learning paradigm. We have a training dataset of words mapped to their exact meaning in context, we train a neural model to predict that exact meaning. When the prediction is right or wrong the neural network acts accordingly modifying the weights to improve further predictions. Specifically the neural model we train for WSD is composed by several layers. My neural structure contained: an embedding layer, two LSTM layers, a softmax layer.

Standard architecture The standard version of my implementation has the following structure: (1) The **embedding layer** has the purpose of mapping words in the sentences to the relative embedding (a latent representation of the word) within an embedding matrix. (2) Two stacked **LSTM layers**, specifically a bidirectional LSTM, it is responsible for creating a latent representation of each word in context (either from right or from left). The latent representation of the context coming from the left and the right are concatenated to form the context representation of a word in a batch sentence. We now need to decode the latent representation of the word in context, in order to do that we need another layer. (3) The **softmax layer** has the purpose to decode the context representation of each input word. It is structured essentially in 5 components. (3.1) A matrix of weights that ideally - in an abstract model - connect 2×`HIDDEN_SIZE` neurons (context) to `NUMBER_SENSES` neurons (predictions senses). (3.2) The bias is a vector of weights connected to `NUMBER_SENSES` neurons. (3.3) The matrix multiplication between the context representation and the matrix of weights, plus the contributes of the bias vector, gives the scores vector (one score for each word, for each sentence in the batch). The score in the i-th position of the vector expresses the likelihood that the input word has the i-th sense, stored in the vocabulary. (3.4) Computing the softmax of the scores vector gives us the vector of probabilities, a distribution over the senses. If the k-th cell of the vector of probabilities contains the maximum value in the vector, the k-th sense is the most likely for that input word.

Modification 1 of the standard architecture Moved by the fact the above presented architecture didn't lead to good results (small F1), I needed something different. At first I didn't implement the attention layer in my architecture since as it was presented the attention layer could lead to a small increase in metics, I wanted to first focus on something more substantial. As it was suggested by the professor, I made a powerful change. From the vector of probabilities given by the softmax, instead of simply retrieving the index associated to the maximum probability (= get the sense suggested by the BiLSTM), I firstly isolated the probabilities of the senses that make the input word ambiguous, and then I searched the maximum between them to eventually return the index of that maximum (= get the sense suggested by the neural network among those which make the word ambiguous). This have much more sense and the results are much more satisfying. To get the possible senses of a word, I use the training set. As soon as I meet a new sense for word w I added it in a list of ambiguities associated to w.

Modification 2 of the standard architecture Sometimes with the neural approach we cannot express a prediction for an input word w coming from the development or test set. It may happen either because we don't have an embedding vector for w (encoded `UNK` in the vocabulary), or we didn't encounter the sense of w in the training set (encoded `UNK` in the vocabulary) or the probability correspondent to the prediction is too low with respect to a certain threshold. In all those cares I used the most frequent sense of the word taking it from *BabelNet*. As a threshold I used the mean acceptance probability discovered experimentally, that I used as a constant for my tests, which is roughly: 0.0878.

Modification 3 of the standard architecture For this modification I was inspired by the Sequence-to-Sequence model discussed in the proposed papers. I simply added a supplementary BiLSTM layer to modification 1, to study the performance of such a learner. What I expected is an encoder of context feeding a decoder of context. I evaluated the model, see results in the next pages.

Implementation and more ideas The code is quite commented but for more clarifications, I'm available. I'd have liked to play more with *BabelNet* but time didn't allow me to. I wanted to try how doable the following is: for each ambiguous lemma of a sentence (with associated `POS`), take the top k `BabelSynset` objects containing the lemma and extract in a bag of words their description, category and senses. Exploit these bags of words, for example by finding words they have in common across the lemmas, to chose the right senses.

	Pre-trained embeddings	Batch size	Hidden size	Optimizer	Dataset size (♯)	Epochs	Initial avg loss	Final avg loss	Instances	Instances to disambiguate	**Instances well classified**	Initial Recall (%)	Initial Precision (%)	Initial F1 score (%)	Final Recall (%)	Final Precision (%)	**Final F1 score (%)**
Test 1	GLOVE	32	150	GRA 0.1	20	15	48.62	28.54	7110	5046	3012	39.3	55.4	45.9	42.3	59.6	49.5
Test 2	FTEXT	-	-	-	-	-	54.32	33.66	7110	5046	**3027**	36.2	51	42.3	42.6	60	**49.8**
Test 3	G-W2V	-	-	-	-	-	53.27	34.76	7110	5046	3019	37.1	52.4	43.5	42.4	59.8	49.6
Test 4	FTEXT ♠	-	-	GRA 1.0	-	10	43.84	27.23	7095	5038	3001	41.6	58.7	48.7	42.2	59.5	49.5
Test 5	-	-	-	GRA 2.0	-	-	43.97	26.87	7095	5038	**3022**	42.3	58.6	48.9	42.5	60	**49.8**
Test 6	-	-	-	ADA 0.1	-	-	23.41	6.11	7095	5038	2764	39.3	55.3	45.9	38.9	54.8	45.5
Test 7	-	-	-	ADA 1.0	-	-	23.51	7.10	7095	5038	2751	38.8	54.6	45.2	38.8	54.6	45.3
Test 8	-	-	50	GRA 2.0 ♠	-	-	41.21	22.96	7095	5038	**3029**	42.3	59.5	49.5	42.7	60.1	**50**
Test 9	-	-	250	-	-	-	70.12	22.9	7095	5038	2978	41.4	58.3	48.4	42	59.1	49.1
Test 10	-	-	500	-	-	-	108.15	26.21	7095	5038	2964	41.1	57.9	48.1	41.8	58.8	48.8
Test 11	-	4	50 ♠	-	-	-	28.08	9.49	7248	5132	**3147**	41	57.9	48.1	43.4	61.3	**50.8**
Test 12	-	10	-	-	-	-	33.38	16.55	7226	5117	3100	41.6	58.8	48.7	42.9	60.5	50.2
Test 13	-	100	-	-	-	-	49.64	31.79	6691	4747	2813	41.5	58.5	48.5	42	59.3	49.2

Table 1: I executed my BiLSTM model as the modification 1 of the standard architecture, making each hyper-parameter vary in order to find the best configuration. This table shows the hyper-parameter tuning via a grid search approach. The hyper-parameters are on the left of the two parallel vertical bars, the results of the execution are on the right. I didn't consider *Epochs* a real hyper-parameter, since I noticed that increasing epochs improved the results unboundedly during my tests, so in general I just stopped when it seemed a convergence was about to take place. Also the Dataset size is not a real hyper-parameter since the increasing of data can only bring positive results. The training was performed on an excerpt of the training dataset (20 sentences per text) composed of the 21% of words present in the whole training set, specifically with: 350 texts, 7040 sentences, 168541 words of which 47346 instances and 121195 wf's. The decision of the best hyper-parameter value was taken looking at the highest values of F1 score during the executions. **Notice** ♯ = sentences per text; and ♠ = for this hyper-parameter, a decision has been taken: such value has been seen to perform best and it will be used from this test on.

	Pre-trained embeddings	Batch size	Hidden size	Optimizer	Dataset size (♯)	Epochs	Initial avg loss	Final avg loss	Instances	Instances to disambiguate	**Instances well classified**	Initial Recall (%)	Initial Precision (%)	Initial F1 score (%)	Final Recall (%)	Final Precision (%)	**Final F1 score (%)**
Test 14	FTEXT	4	50	GRA 2.0	WHOLE	10	107.75	25.62	7248	6049	3438	48.1	57.6	52.4	47.3	56.8	51.7
Test 15	FTEXT	10	50	GRA 2.0	WHOLE	10	80.2	8.23	7226	6030	**3590**	47.8	57.2	52.1	49.7	59.5	**54.2**

Table 2: I executed my BiLSTM model as the modification 1 of the standard architecture, on the *whole* training dataset with the best hyper-parameters configuration discovered (expressed in Table 1), then I tested it on the *whole* development set. The results show clearly that the best configuration discovered in Table 1 aren't extraordinary when more data is added for training. This confirmed me that sometimes an exhaustive combination of multiple hyper-parameters is necessary in order to find the best configuration, a sequential discovery may not be enough. So I decided to make the *Batch size* slightly bigger in another test (Test 2), to see if somehow due to the increasing in the dataset size a low value could negatively influence the performance, that was actually true. Test 2 is indeed the best configuration and the model I used for final evaluation.

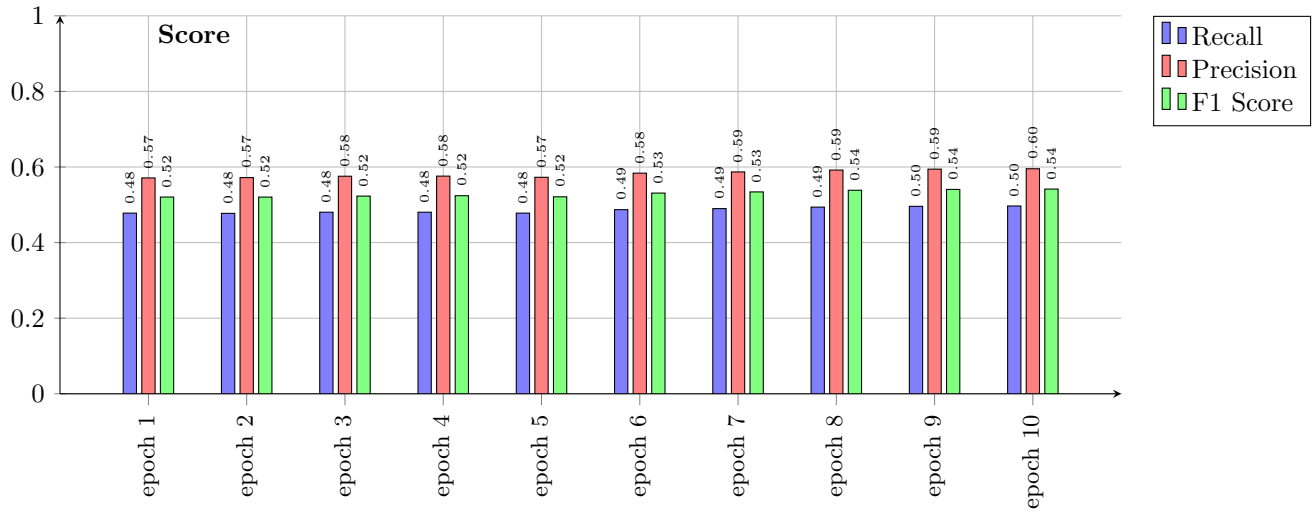

Figure 1: The execution of the best configuration discussed above (Test 15) of the modification 1 of the standard architecture, produced the following Recall, Precision and F1 Score per epoch. As the plot accurately shows the score sometimes goes up and down, due to a possible overfitting behaviour, and furthermore the increase in scores is very small at each epoch. I stopped this test even though the score was increasing, and I didn't reach a convergence since it took more than 5 hours. **Note**: I computed the *recall* as the ratio between the correct predictions of ambiguous words over the total number of ambiguous words. I computed the *precision* as the ratio between the correct predictions of ambiguous words over the total number of ambiguous words I could predict (no UNK word, no UNK meaning, see comments in the code). I computed the *F1 score* as the ratio between twice the precision times the recall, and the sum of the precision and the recall.

	Senseval 2	Senseval 3	Senseval 2007	Senseval 2013	Senseval 2015
Recall (%)	52.5	53.6	46.6	43.2	47.9
Precision (%)	63.1	60.9	53.8	55	58.2
F1 Score (%)	57.4	57.01	49.9	48.4	52.6

Table 3: The execution of the best configuration discussed above (Test 15) of the modification 1 of the standard architecture, produced the following Recall, Precision and F1 Score for each of the sentences in the 5 different corpora.

Pre-trained Embeddings	Total vectors	Vocabulary hits	Percentage hits (%)
`Google W2V`	3,000,000	26,457	75,6
`Glove`	400,000	21,946	62,7
`FastText`	999,995	28,800	82,4

Table 4: This table shows the quantity of words that are either in the vocabulary of the whole training set (34,996 distinct words), or in the pre-trained word embedding matrix. For each matrix a high *Percentage hit* value implies a low quantity of `UNK` words in the sentences encoded with the vocabulary, so more valuable data to train on. The best pre-trained embedding matrix seems to be the FastText one, it is composed of 1 million word vectors trained on Wikipedia 2017. The FastText embeddings creation treats each word as composed of character n-grams. So the vector for a word is made of the sum of the character n-grams. Eventually this matrix was chosen as the best configuration.

	Senseval 2	Senseval 3	Senseval 2007	Senseval 2013	Senseval 2015
Recall (%)	-	-	-	-	-
Precision (%)	-	-	-	-	-
F1 Score (%)	64.2	60.1	52.3	55.4	56.6

Table 5: The execution of the best configuration discussed at Test 15 of the modification 2(A) of the standard architecture, produced the following Recall, Precision and F1 Score for each of the sentences in the 5 different corpora. In this case the model simply predicted a meaning even when the BiLSTM wasn't able to do it (due to UNK words or UNK meanings), the systems simply predicts the most frequent sense of that word, gathering informations from Babelnet. **Note**: Either here or in the table below recall and precision are the same since all the words could be disambiguated with the most frequent sense at least.

	Senseval 2	Senseval 3	Senseval 2007	Senseval 2013	Senseval 2015
Recall (%)	-	-	-	-	-
Precision (%)	-	-	-	-	-
F1 Score (%)	66.2	65.2	54.3	61.4	60.1

Table 6: The execution of the best configuration discussed at Test 15 of the modification 2(B) of the standard architecture, produced the following Recall, Precision and F1 Score for each of the sentences in the 5 different corpora. In this case the model predicted a meaning in the following way: if the prediction from the BiLSTM (same as modification 1) was made with probability higher than the threshold (mean acceptance probability discovered experimentally = 0.0878) then use that prediction, otherwise use the most frequent sense. The system also predicts the most frequent sense when the BiLSTM wasn't able to predict a sense (due to UNK word or UNK meaning).

	Pre-trained embeddings	Batch size	Hidden size	Optimizer	Dataset size (‡)	Epochs	Initial avg loss	Final avg loss	Instances	Instances to disambiguate	**Instances well classified**	Initial Recall (%)	Initial Precision (%)	Initial F1 score (%)	Final Recall (%)	Final Precision (%)	**Final F1 score (%)**
Test 16	FTEXT	10	50	GRA 2.0	WHOLE	2	123.2	78.33	7226	6030	3336	46.2	55.3	50.3	48.5	58.1	52.9

Table 7: I executed my BiLSTM model as the modification 3 of the standard architecture on the best model of the modification 1. The results are still good, and I could only test for 2 epochs, the impression is that the rate on increase in F1 is much higher, but I'd need more time to experiment this model.