Homework 1: Word in Context Disambiguation

Simone Rossetti
rossetti.1900592@studenti.uniroma1.it

1 Introduction

In this report we are going to present a possible solution to a problem named Word-in-Context Disambiguation (WiC), which is a different formulation of the Word Sense Disambiguation (WSD) problem into a binary classification task. Given a polysemous lemma we want to discriminate weather it has the same meaning in two different sentences, without relying on a fixed inventory of word senses.

2 Pre-processing

I experimented four different approaches in additional trend.
The first trivial approach leads to simply remove from the sentences special characters, keeping only words longer than a 3 characters size. This results in a total of 200K words corpus with 16K unique words and a word length mean of 3-4 characters.
The second approach consists in building a vocabulary from the dataset and adopting negative frequency sampling of words, this is done in order to avoid the model to rely too much on frequent words and help generalization.
The third approach makes use of some *well known NLP best practices* (Navigli, 2009), that is build a pipeline which tokenizes the sentence (removing special characters and numbers), then, removes (english) stop words, tags Part Of Speach (POS) and lemmatizes the words reducing the morphological variants to their basic form (for this purpose I used NLTK library APIs). This pipeline makes the dataset more consistent w.r.t words standard form and minimizes presence of unknown token when using pre-trained word embedding models. This results in a total of 100K words corpus with 12K unique words and a word length mean of 5-6 characters. These approaches aims also at maximizing the entropy of the dataset. Most frequent words are infact pruned before fitting the model.
The last approach consists in recognizing and labeling the text with Named Entity (NE) tags and using a naive english regular expression grammar parser to extract more relevant parts of the sentence such as Noun Phrases (NP) and Verb Phrases (VP) to get the context phrase of the query lemma.
The last approach, namely, POS and NE tags and grammar rules has been used as *extra submission* to provide additional information to the model as we will see in the experiments I made.

3 Word Embeddings

I decided to use a pre-trained word embedding model, namely one of the most successful ones, GloVe word embeddings trained on Wikipedia and Gigaword 2014 dataset made of 6B tokens, a very little set but very efficient for our purposes. The model contains 400K vocables, represented as 300D embedding vectors. Using this corpus on the training set resulting from the last pre-processing approach described before, we minimize the OOV (out of vocabulary) words to 300 tokens, which is about the 3% of the words in the set, all of them appear just once. I decided to reserve four spacial tokens, namely $\langle pad \rangle$, $\langle unk \rangle$, $\langle drop \rangle$, $\langle sep \rangle$, to keep the input noise proportional I decided to sample the embeddings for those tokens from a normal distribution $\mathcal{U}(\mu_e, \sigma_e)$, where μ_e and σ_e are the mean and standard deviation of the embeddings of the model (Kågebäck and Salomonsson, 2016).

4 Models

As described above the model should take as input two sentences and output a binary value, thus a binary cross entropy loss with sigmoid activation function for the last layer output is used in both cases. The baseline network simply relies on the topology of the word embedding vector, a simple mean is performed for aggregating each

sentence independently. These two new vectors are then concatenated and given to a MLP (multi layer perceptron) with one hidden layer and ReLU activation function, more details in Table 1, using simple regularization methods gave me very good generalization getting 68.7% accuracy on dev set (see Table 1).

The second model instead relies on a more sophisticated reasoning, which is, using a Recurrent Neural Network (RNN) layer on two sentences we can learn how to aggregate word embeddings and produce a context embedding, then use an MLP classifier to discriminate their similarity.

In particular I adopted a Gated Recurrent Unit (GRU) layer (Kyunghyun Cho and Bengio, 2014) which gave me better performances but also because has been shown to perform better for smaller (Gruber and Jockisch, 2020) and less frequent datasets (Junyoung Chung and Bengio, 2014).

Given the two pre-processed sentences s_1 and s_2, the query indices q_1 and q_2 of the query lemmata, and the length of the sequences L_1 and L_2, I took the embeddings $e_1^{s_i}, e_2^{s_i}, \ldots, e_{L_i}^{s_i}$ from the GloVe pretrained model of each word $w_1^{s_i}, w_2^{s_i}, \ldots, w_{L_i}^{s_i}$ of each sentence s_i, concatenated to the one-hot encoding of the query lemmata (1 labels the query lemma, 0 the others), and fed the sequences to 2 stacked mono-directional GRU layers (the full gated version) with 256 hidden size layer and 0.2 dropout in between, which gave me the outputs $x_1^{s_i}, x_2^{s_i}, \ldots, x_{L_i}^{s_i}$, where $i = 1, 2$.

At this point I collected the last output of the sequences so namely, $x_{L_i}^{s_i}$ for each $i = 1, 2$ and concatenated them to get the full vector $[x_{L_i}^{s_1}, x_{L_i}^{s_2}]$.

At this point I adopted a MLP classifier, similiar to the one used above with 256 hidden size layer, to learn the distance function among the feature vectors produced by the two GRU layers.

5 Experiments

The first simple approach, i.e. pre-processing + word embedding mean reduction + 2 MLP layers, gave me a solid baseline of 68.7% accuracy (Table 1). For the second approach I started testing on a simple one-layer mono-directional LSTM followed by the same MLP layers as before (Table 2). Once I reached consistent limitations of overfitting I considered different architectures such as, BiLSTM, GRU and BiGRU (Table 3). The only architecture which could really make the difference was a 2 layer mono-directional GRU and 0.2 dropout, with which I could achieve 69.7% accuracy on *dev* set (Table 5). I considered several options to overcome the limited data and fast overfitting issues (Table 4), that is:

- **Negative frequency sampling**: it avoids biasing of the model towards more frequent words which do not contribute too much to the context. Once the vocabulary with the respective frequencies is built (the Bag of Words model) I sampled words with probability $0.8 * (1 - \frac{fw_i}{f_{max}}) + 0.2$. This consistently improved my results.
- **Windowing**: the training samples are cut of a prefixed size (i.e. 10-20 tokens) around the query lemma, this is done in order to keep only word-context relevant information.
- **Dropword**: avoid the RNN to rely too much on specific tokens, highly improve generalization for RNN architectures (Iyyer et al., 2015), before feeding to the embedding layer, words are substituted by a $\langle drop \rangle$ token with uniform probability distribution $\mathcal{U}(0, 1)$ and 0.2 threshold.
- **Gaussian Noise**: another common technique in NLP and RNN architectures is to add noise at embedding level, in my case I had very good generalization for $\mathcal{N}(0, 1.5\sigma_i)$ where σ_i is the standard deviation of the embeddings.
- **Dropout and batch normalization**: used on the MLP classifier has the well known property to reduce overfitting.

BiGRU and BiLSTM experiments in Table 3 has been done considering embedded inputs $x_{1^{+/-}}^{s_i}, x_{2^{+/-}}^{s_i}, \ldots, x_{L_i^{+/-}}^{s_i}$ $x_1^{s_i}, x_2^{s_i}, \ldots, x_{L_i}^{s_i}$, where $i = 1, 2$ and $^{+/-}$ the forward and backward directions. I collected the last output of the sequences surrounding the query lemmas at positions q_i, so namely, $x_{q_i-1^+}^{s_i}$ and $x_{q_i+1^-}^{s_i}$ for each $i = 1, 2$ and concatenated them to get the full vector $[x_{q_i-1^+}^{s_1}, x_{q_i+1-}^{s_2}]$, but this did not improve the results.

The results of some experiments on grammar based data preprocessing and POS+NE tagging one-hot encoding are shown in Table 6. In particular I experimented the idea that given some grammar sequences and named entity labels the model could also use this knowledge to learn by itself to give

more importance to the relevant part of the sentences, such as noun and verbs.

6 Results

Trainings are stopped 20 epochs after accuracy stops increasing. Bold lines are configurations chosen for subsequent experiments. Dots mean no changes.

Model	Emb	Aggr	Acc	F1
MLP 128	Random	Sum	.527	.470
-	-	Mean	**.532**	**.471**
-	FastText	-	.622	.649
-	GloVe	-	**.649**	**.660**
+ BN, 0.7 D	-	-	**.681**	**.704**

Table 1: MLP classifier with input size of 600 (concatenation of aggregated word embedding vectors of the two sentences) one hidden layer of size 128 and ReLU activation function. All trainings here are performed using SGD optimization, 0.1 learning rate and 0.9 momentum on 128 batch size.

Optimizer	Params	Epoch	Acc	F1
SGD	0.1/0.0	32	.608	.613
-	-/0.5	30	.612	0.657
-	-/0.9	X	X	X
-	0.01/0.9	27	.615	.624
Adam	0.01	X	X	X
-	0.001	**10**	**.620**	**.625**
Adadelta	1	11	.599	.631
-	0.1	20	.609	.658

Table 2: Comparison between different types of optimization. All tests are performed on a model made by one layer LSTM with 256 hidden size aggregation and best MLP classifier previously defined in Table 1. No further regularization technique is applied here. X means 'no success'.

RNNs	Levels	Acc	F1
LSTM 256	1	.620	.625
GRU 256	-	.624	.656
BiLSTM 256	-	.578	.574
BiGRU 256	-	.582	.571
LSTM 256	2	.633	.660
GRU 256	-	**.643**	**.658**

Table 3: Each RNNs have 256 hidden size layer. Aggregation types are trained on the best MLP classifier in Table 1, same concatenation is used. No further regularization technique is applied here.

Regularization	Acc	F1
-	.643	.658
RNN Dropout	.656	.666
Dropword	.666	.672
Gaussian noise	.677	.684
Gradient clip norm	.680	.699
Adadelta optimizer	.691	.701
Final tuning	**.697**	**.698**

Table 4: Successful regularization technique applied on the final model: **2 GRU 256 + 2 MLP 256**. All trainings before Adadelta are performed using Adam optimization with 0.001 learning rate and 128 batch size.

Params	Value
Architecture	2 GRU + 2 MLP
Hidden sizes	256
MLP dropout	0.2/0.2
GRU dropout	0.2
Dropword	0.5
Gaussian noise	$\mathcal{N}(0, 1.5\sigma_e)$
Optimizer	Adadelta
Learning rate	0.3
Batch size	128
Gradient clip norm	5

Table 5: Final model parameters.

Model F1	Aggr	Acc
Final Model	**.697**	**.698**
+ POS tags	.677	.681
+ NE tags	.690	.689
+ NP/VP/CLAUSE only	.673	.669

Table 6: Results are given by concatenating to the embedding layer the one-hot encoding of different labels.

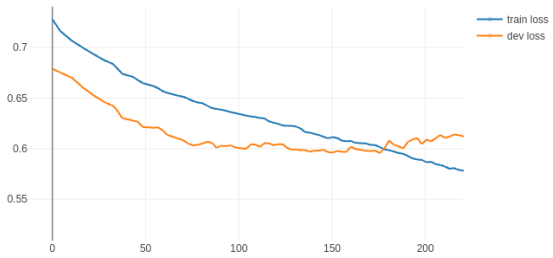

Figure 1: Final model train and dev loss during training.

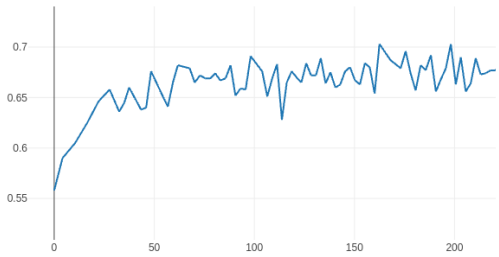

Figure 2: Final model accuracy on dev set during training.

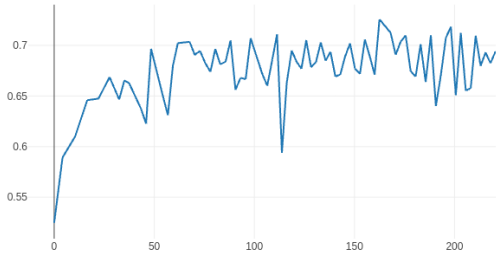

Figure 3: Final model F1 score on dev set during training.

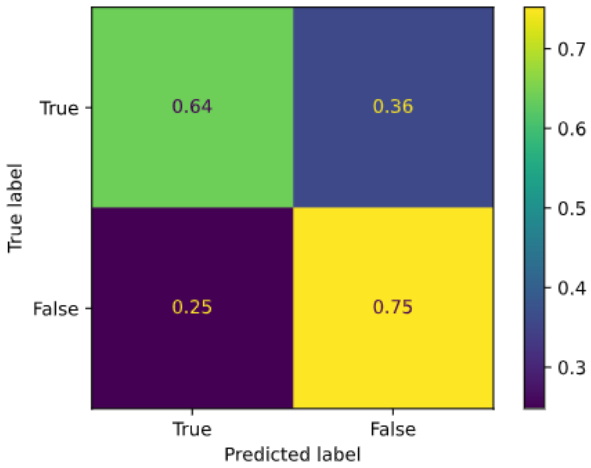

Figure 4: Final model confusion matrix.
Precision: 0.6988, Recall: 0.6970, F1: 0.6963, Accuracy: 0.6970.

References

Nicole Gruber and Alfred Jockisch. 2020. Are gru cells more specific and lstm cells more sensitive in motive classification of text? *Frontiers in Artificial Intelligence*.

Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. 2015. Deep unordered composition rivals syntactic methods for text classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1681–1691, Beijing, China. Association for Computational Linguistics.

KyungHyun Cho Junyoung Chung, Caglar Gulcehre and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *Computing Research Repository*, arXiv:1412.3555. Version 1.

Mikael Kågebäck and Hans Salomonsson. 2016. Word sense disambiguation using a bidirectional LSTM. In *Proceedings of the 5th Workshop on Cognitive Aspects of the Lexicon (CogALex - V)*, pages 51–56, Osaka, Japan. The COLING 2016 Organizing Committee.

Caglar Gulcehre Dzmitry Bahdanau Fethi Bougares Holger Schwenk Kyunghyun Cho, Bart van Merrienboer and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *Computing Research Repository*, arXiv:1406.1078. Version 3.

Roberto Navigli. 2009. Word sense disambiguation: A survey. *ACM Comput. Surv.*, 41(2).