NLP Homework1-2021: Word-in-Context Disambiguation

Sorokoletova Olga
sorokoletova.1937430@studenti.uniroma1.it

May 2, 2021

1 Introduction

This work is devoted to the word-in-context disambiguation task. Both of the proposed in task assignment approaches (word-level and sequence encoding) have been implemented. The model corresponding to the 1st approach performs best of all and after some preprocessing reaches quite high accuracy, but work was mainly focused on the 2nd approach. In particular, I aimed to make it performing on the decent level: at least close to the level of the 1st approach. The main LSTM-based model and 2 it's modifications are developed by the 2nd approach fine-tuning. A comparative analysis of all four models (together with the first approach baseline model) is carried out.

2 Preprocessing

Without preprocessing even the best configured model is hardly able to outperform a random baseline. Meanwhile, adding it leads to a significant growth of metric values. Therefore, preprocessing is a powerful instrument which should necessarily be a part of a pipeline for a given NLP task.
First of all, from the Figure 1 it can be seen that both training and development datasets are small and perfectly balanced, so no additional measures related to the imbalanced classification are needed. Then, it is worthy to notice, that preprocessing functions for the 1st and the 2nd approach models encounters the most of the things in common, but is not identical.

1. Lowercasing. Required step to normalize representation of a sentence.
2. Punctuation. Without doing anything with punctuation after applying of a sentence split into tokens punctuation signs will be considered as parts of word tokens and such tokens will not match to the targets. Implemented approach is to delete punctuation, only adding a dot as a separation. There exists also an alternative: to separate punctuation signs from the words by spaces and include them into computations, but it is not really helpful in our case.
3. Punctuation. Special cases. Exploration shows that punctuation removal not always works in an expected way, and some manual corrections are needed to provide correct mapping with the vocabulary. Modification are made for the hyphen, slash and dash signs (e.g. word `non-partners` would be transformed into unrecognizable `nonpartners` without corrections for a hyphen).
4. Stop words. Stop words are not meaningful for disambiguation. Removal of them gives a great boost in the performance, but within a more precise view it turns out that some stop words (e.g. `only`) are also targets. Appropriate corrections are done for the 2nd approach where target words are used.
5. Unknown tokens. For the 1st approach unknowns are just ignored, meanwhile for the 2nd – it is more arguable. Basically unknown words are mapped to a special index in the vocabulary. However, it was found out that elimination of unknown tokens performs better except for the cases when these words are our targets (probably, they are too specific to enhance generalization).
6. Numbers removal. Numbers are included into embeddings and might be used, but since we are not in the math domain, presence of numbers seems to influence more as a stop words presence than as something relevant. Thus, numerical tokens are eliminated.
7. Lemmatization. Method helps to find lemmas of words and is reasonable for the 2nd, but not for the 1st approach, where the result is smoothed by averaging.
8. Stemming. Stemming has shown bad results which can be intuitively explained by shortened look-ups not containing anymore some relevant

for usage information. Probably, stemming is more suitable for the larger datasets and not when the goal is focused on the word-in-context disambiguation.

9. POS. One could provide an auxiliary information by POS tags retrieval. In experiments, it seems not to be over-complication, but not useful, so tagging is not applied eventually.

10. Pad token is needed for LSTM-s to provide sequences of an equal length. A set of experiments with padding allowed to check if there is more effective way pad than with zeros. For instance, one of them was to pad with a target word index (inspired by an idea to give more weight to targets). However, there are two issues with this: 1) distribution of a target word is corrupted in an unexpected way because sequences are needed in a different amount of padding, 2) influence of the surrounding context is then less considered.

3 Architecture

Four model architectures are designed: Baseline, Baseline-2, 2a and 2b: correspond to the 1st approach and the 2nd approach model with two it's modifications, respectively (corresponding files are referenced in the Table 1). For all models: 1) `GloVe` pre-trained embeddings with 50 dimensions were used (more dimensions set up during experiments did not lead to improvements, therefore, 50d are chosen for the effective memory consumption), 2) Binary Cross-Entropy is a loss-function suitable for a binary classification.

3.1 Baseline

This model includes computation of an average embedding for each sentence, concatenation of them with an embedding of a dot token, obtaining in the result $3 \times 50d$ feature vector to be given as input to an MLP-classifier with 2 hidden layers.

3.2 Baseline-2

Baseline-2 is a subject of a main focus due to the fact that approach, which takes into consideration sequential nature of a text should perform better, but it does not, and therefore, a way to make it working so that to take an advantage of it's potential capabilities was possible, should be developed. Final best configuration is following: preprocessed tokenized sentences a concatenated with a dot token (different strategies of concatenation were applied but this simplest one seems to be the best), and then list of corresponding indices is padded and given to an RNN-classifier constructed by embedding, recurrent and classification layers. Depending on the number of an experiment dropout/batch normalization layers are also present.

3.2.1 Approach 2. Model 2a

So far no priority to the target words was given and information about them was not really used other than during preprocessing, meanwhile this information is relevant. So, in model 2a, information about target words locations is stored and kept through the processing and many-to-2 architecture is implemented instead of many-to-1 on the intermediate level of recurrent layer: the whole sentence is fed but then outputs corresponding to the target words are extracted instead of only one final output. To organize correct training double-sized gold answers vector is provided.

3.2.2 Approach 2. Model 2b

Notebook with this model is a combination of Baseline-2 and 2a and is not provided in order to avoid pieces of code duplication. Underlined idea is to combine all available information and implement a standard supervised learning where feature space is represented by the 4 global features: LSTM-output for the sentence 1, output corresponding to the target word in the sentence 1 and the same for the sentence 2. Input for the classification layer has a dimension $4 \times 50d$, two LSTM-s are trained in a parallel, but no double-sizing of a label vector is needed.

4 Experiments

Except for the experiments with preprocessing and model design, hyperparameters settings and overfitting handling have been investigated. Comparative table of the best performances (Table 2), hyperparameters final set-ups (Table 3), accuracy and losses plots for the Baseline, Baseline-2, 2a models (Figures 2, 3, 7) and results of the regularization measures applied to the Baseline-2 model (Figures 4, 5, 6) with the corresponding conclusions as well as diagram of what is considered to be my extras (Figure 8) can be found on the subsequent pages.

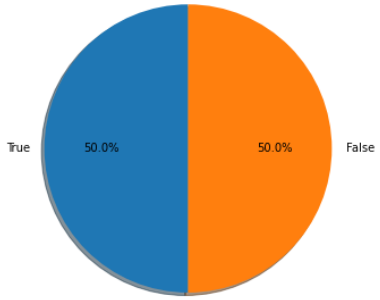


Figure 1:
Train dataset: 'False': 4000, 'True': 4000,
Dev dataset: 'False': 500, 'True': 500

Model	Files

Baseline
Baseline-2
2a
	`implementation.py, baseline.ipynb`
	`baseline2.ipynb`
	`2a.ipynb`

Table 1: Mapping of the described models to the files where their implementation can be found

Model	Best accuracy
Baseline	0.7236
Baseline-2	0.6681
2a	0.5776
2b	0.5773

Table 2: Best overall performance achieved by the model through the set of independent runs.
Baseline: the result is stably achieved around 30-40 epoch of training, corresponding best weights can be found in the model folder, no regularization techniques are applied.
Baseline-2: the result is obtained due to continuous tuning of the model and is a 0.4 points more than stable best result (about 0.62), achieved at the 60-70 epoch of training, corresponding best weights can be found in the model folder.
2a and 2b: results are very close to each other, but in the first case they are permanently achieved over independent runs, meanwhile in the second – training process is represented by fluctuations of a significant amplitude.

Hyperparameter	Approach 1	Approach 2
Epochs	50	70
ES patience	7	7
ES threshold	0.009	0.01
Batch Size	64	256
Embedding dim	50	50
N features	150	50
N hidden units	128	128
N hidden layers / LSTM-layers	2	1
Hidden MLP activation	ReLU	ReLU
Optimizer	Adam	Adam
Learning Rate	10^{-4}	10^{-4}
Weight Decay	–	10^{-6}
Dropout rate	0.0	0.0

Table 3: Hyperparameters corresponding to the best performed models.
Epochs: is regulated by Early Stop. Patience of it and it's threshold varies during experiments.
Bath size: larger batch sizes perform better for the 2nd approach, but should not be too large, considering the size of dataset.
N hidden units: smaller values perform worse, larger – do not lead to improvements.
N hidden layers / LSTM-layers: more LSTM-layers allows to apply dropout to LSTM-cell, but this does not seem to enhance generalization ability.
Hidden MLP activation: Tanh and Leaky ReLU (0.2) have been tried, but ReLU is chosen as the best one.
Optimizer: SGD (+momentum) and RMSprop have been tried, but Adam is chosen as the best one.
Weight Decay: L2-regularization applied against overfitting.
Other: Bidirectional architecture and standard RNN/GRU units are tried when aiming to achieve better performance for the 2nd approach, but LSTM works better than other types of cells, meanwhile unidirectional architecture produces less overfitting.

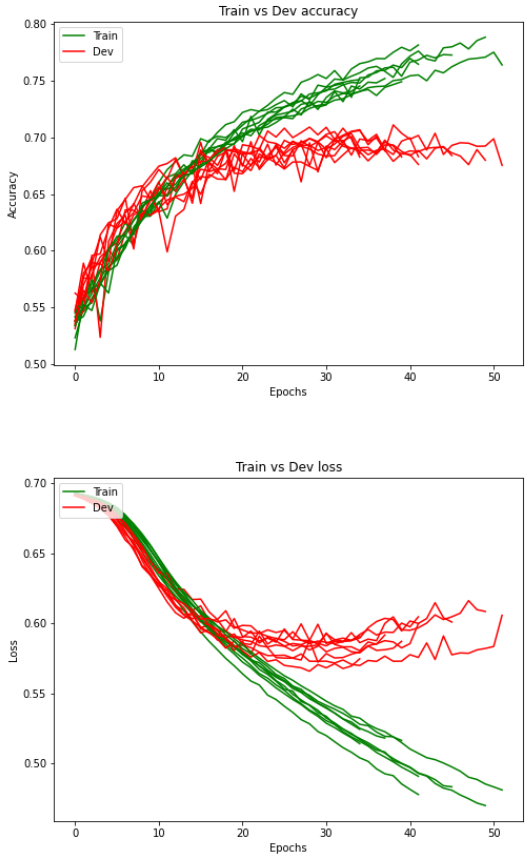


Figure 2: **Baseline model.** Training and validation accuracy and loss over 10 independent runs. Training process got stabilized around 0.70 value of accuracy that is close to the best one and should be stopped about 30-40 epoch, after which overfitting effect is clearly seen.

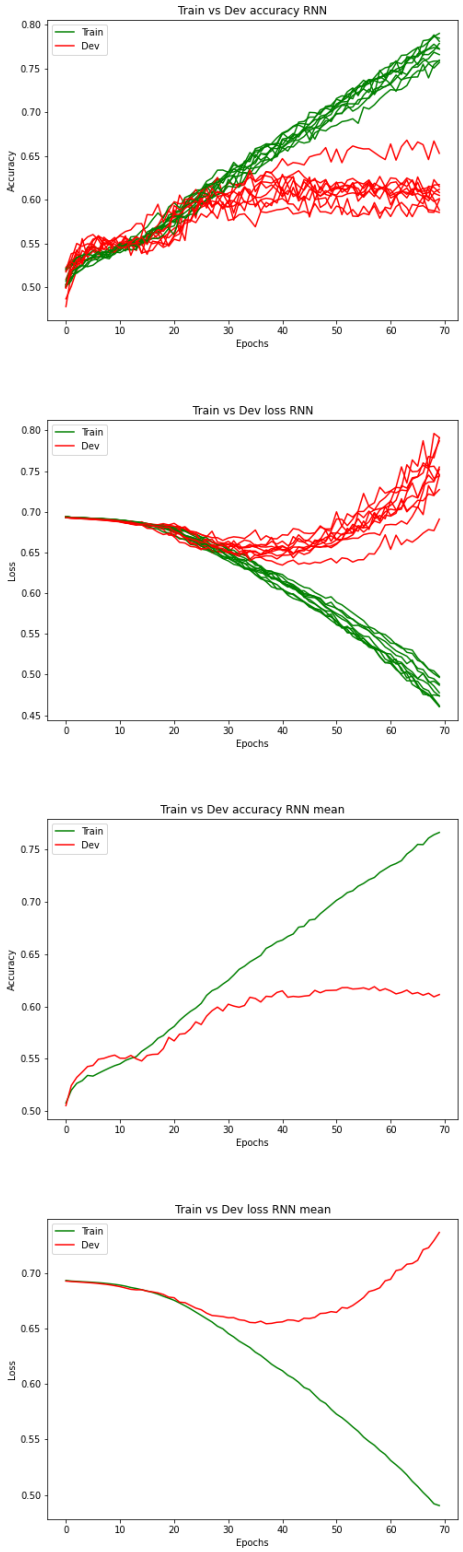
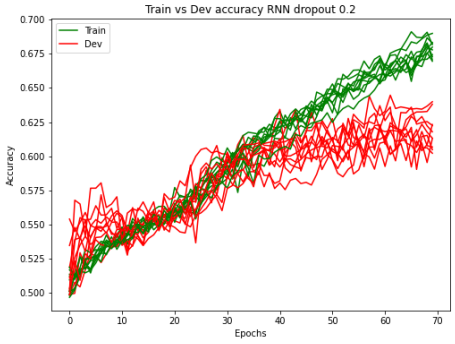
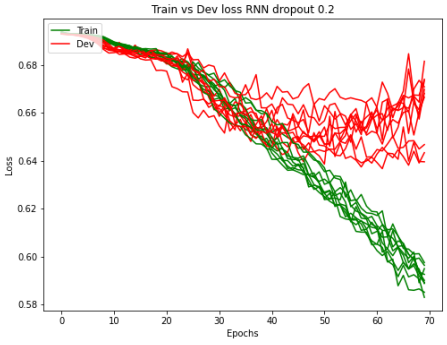
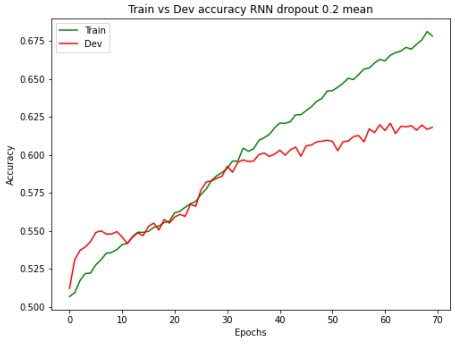


Figure 3: **Baseline-2 model.** Training and validation accuracy and loss over 10 independent runs and their averaged over runs versions. Training process got stabilized around 0.60 value of accuracy. The first two plots show that the best performed model is not a stably achieved result, but result of a lucky initialization. About 50 epoch training should be stopped to avoid overfitting effect or additional regularization measures are required.







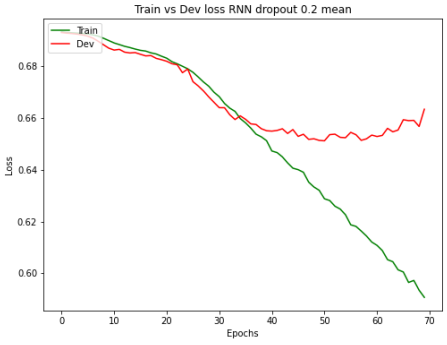
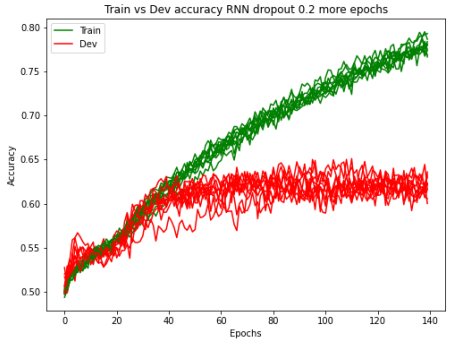
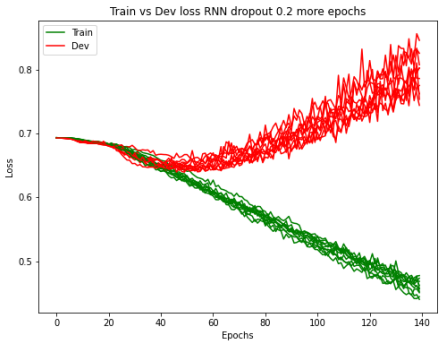
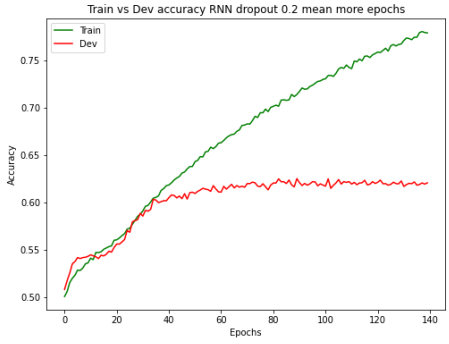


Figure 4: **Baseline-2 + dropout, 70 epochs.** Training and validation accuracy and loss over 10 independent runs and their averaged versions. Additionally to the already applied before L2-regularization, idea to reduce overfitting by dropout layers, applied after embedding, recurrent layers and inside LSTM (num of LSTM-layers is set to 2) with dropout rate 0.2 is implemented. Stabilizing occurs around 0.62 value which is higher than before. It is not obvious, if validation accuracy is going to go down after epoch 70, since accuracy curves do not degrade or stuck yet, but loss plots contain hints about it.







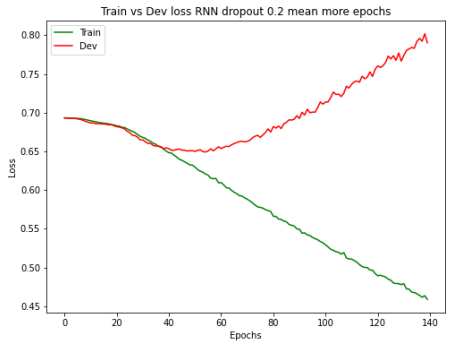


Figure 5: **Baseline-2 + dropout, 140 epochs.** Training and validation accuracy and loss over 10 independent runs and their averaged versions. Training time is doubled. Assumptions, made during previous (short) runs, are confirmed.

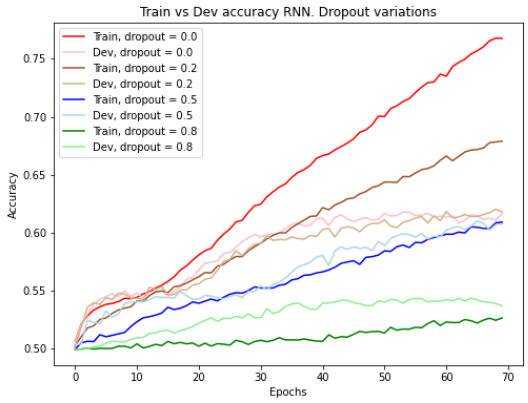


Figure 6: **Baseline-2 + dropout variations.** Training and validation accuracy averaged over 10 independent runs. Check the hypothesis if further increasing of a dropout rate is able to gain better performance. Accuracy curves represented on the graph correspond to the 0.0, 0.2, 0.5 and 0.8 dropout rates. It can be concluded that the larger dropout rate is, the slower training process goes, meanwhile the best performances are achieved at the low values of a dropout rate and when dropout rate is too large, model is losing it's ability to learn anything. Therefore, further improvements should be done through the other techniques.

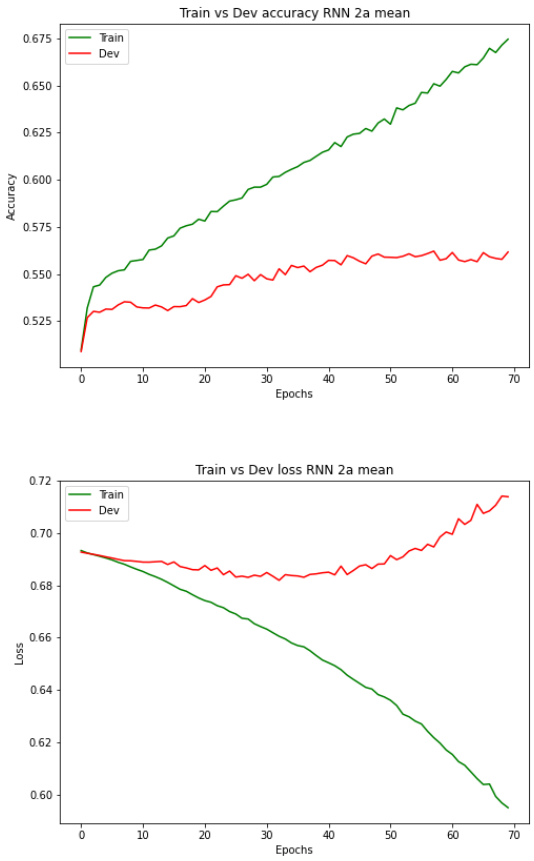


Figure 7: **2a model.** Training and validation accuracy and loss averaged over 10 independent runs. Check the hypothesis if changing of the model architecture in a proposed alternative way is able to gain better performance. Various set-ups for the model were made, did not make it perform better. Again, a lack of ability to generalize is present, but it can be noticed that even before overfitting, training was not progressing significantly, which makes a clue that either chosen architecture does not fit to a given task or it should be further fine tuned with more sophisticated approaches.

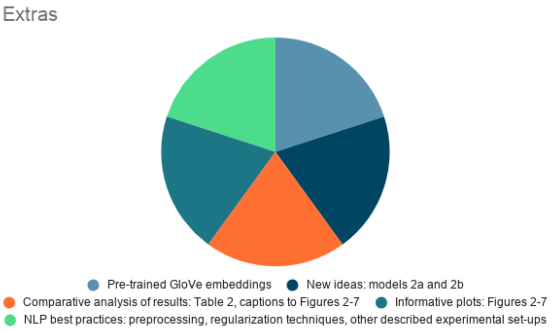


Figure 8: **Extras by categories:** attempts to implement all categories of extras have been made