Project Report

Andrea Bonaiuti mat. 1618825

1 Introduction

This report is about the project of Deep Learning exam regarding the Relation Extraction task.

It is structured in a brief definition of the task/problem, then some analysis of the assigned dataset and a section for each baseline method developed.

2 Relation Extraction task

As stated in the project assignment, Relation Extraction is the task of predicting attributes and relations for entities in a sentence. For example, given the sentence "Barack Obama was born in Honolulu, Hawaii.", a relation classifier aims at predicting the relation of "bornInCity".

Relation Extraction is the key component for building relation knowledge graphs. It is crucial to natural language processing applications such as structured search, sentiment analysis, question answering, and summarization.

After the study of some paper about this topic, I found that the standardized output for RE (achronym for Relation Extraction from now on) is a "triplet", composed by: Entity (subject), Relation (most frequently a verb), Entity (object or attribute).

As seen in the composition of the output, RE firstly relies on the identification of entities in a text, a task called NER (achronym of Named Entity Recognition from now on); and only after in the composition of the triplets with these predicted entities, plus a relation extracted from a set of predefined relations. Given this last detail, the task can be defined as a classification one.

3 The WebNGL Dataset

3.1 The Dataset

As stated in the project assignment, The WebNLG corpus comprises sets of triplets describing facts (entities and relations between them) and the corresponding facts in the form of natural language text. The corpus contains sets with up to 7 triplets each, along with one or more reference texts for each set. The test set is split into two parts: seen, containing inputs created for entities and relations belonging to DBpedia categories seen in the training data, and unseen, containing inputs extracted for entities and relations belonging to 5 unseen categories. Initially, the dataset was used for the WebNLG natural language generation challenge, which consists of mapping the sets of triplets to text, including referring expression generation, aggregation, lexicalization, surface realization, and sentence segmentation. The corpus is also used for a reverse task of triplets extraction.

3.2 The selected version

For a more fair comparison between the developed models and the papers which inspired this project (in particular the SOTA indicated in the assignment), and also due to the limits of computational resources of the used hosted notebook services (Colab and Kaggle), I choose to use the version given by the SOTA paper [3], also used in other studies like [2].

A more in-depth look at the train split showed me that some classes where missing and the others were very unbalanced, as it can be seen in Figure 1.

Whit this information I started doing some inference on the SOTA project, and after several tries its obvious that this model has not learned to classify all the relations written in the paper and in the relations file from the data folder.

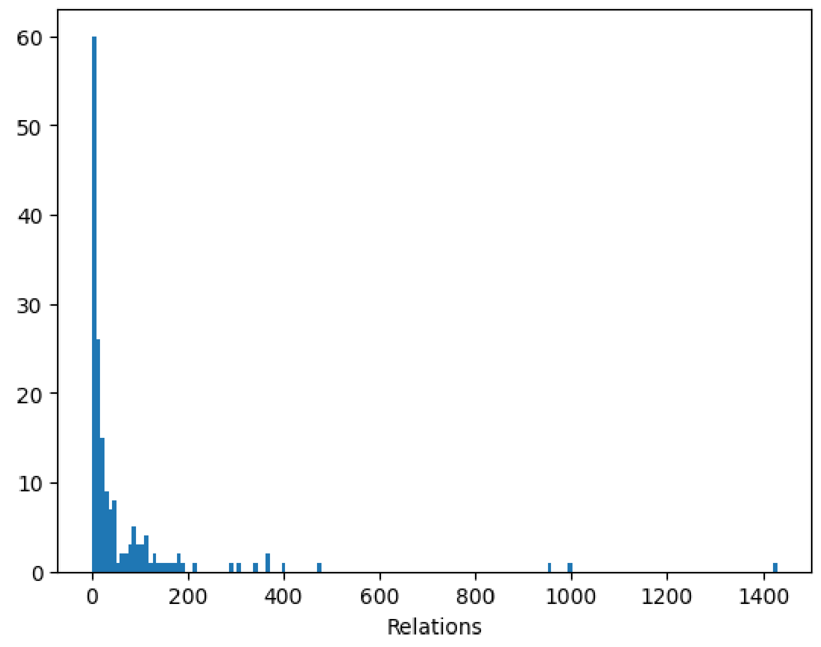

Figure 1: The plotted distribution of the relations(classes) in the dataset from the SOTA paper.

Furthermore, it results in being very case sensitive: using ”rome” instead of ”Rome” won’t be recognized as an entity and therefore any relation involving the city won’t be recognized as well, but this property extends also with some common nouns that usually don’t require the Upper case. The very good results indicates that also the validation and test set contains the same amount and distribution of relations.

4 SOTA Baseline, PFN

The PFN method is a sort of encoder-decoder in which NER and RE are done jointly.

It firstly encode the test sentence using an embedder, bert-base-cased in this scenario, to convert each token (scalar representing a word or just a part of it) in a feature vector and then implements a particular gating technique to divide each token’s embedding features in three partitions: an entity-related one that is fed into the NER task unit, a relation-related one that is fed into the RE unit, and a shared partition that will be fed in both. Then the tasks are done in a table-filling way, through some Linear units fed with the associated specific partition and a task specific global representation.

The development started reconstructing this method from a very approximated way, using just the gating technique (the most relevant part of this paper), and adding more elements of the SOTA after trials and trainings to reach at least similar metrics results. The best improvement was to include the embedder in the training, but I did this only as one of the last steps, so I have the feelings that some layernorm and some initialization could be removed without a significant loss in the metric results; unfortunately, due to time and resource limitations I didn’t experimented it yet.

To save some time in the training phase, a needed move due to limited resources again, the first trick was to tokenize the sentences before training.

Other tricks used during the development were 1) using a smaller embedder like tiny-bert, but results were too low for that, at least at the first epochs, 2) not fine-tuning the embedder, as it would have saved computation and time, but this is also compromised results.

One important modification was to remove the prediction of entity type during the NER phase, as there is no such property in the selected dataset. This led to a removed dimension in the NER unit and also, of course, in the NER labels.

The training behaviour of the SOTA paper consists in a very fast learning of the NER task and after some epochs the RE one starts to learn too, with a slower velocity. In my experiments this behaviours were not reproduced, so I decided to freeze the RE task learning for some epochs training just the NER unit (until a f1 score of 97%), but the start of learning of this task rapidly worsened the NER unit that didn't recover in acceptable time.

So my last trick was to weight the RE loss proportionally with the number of epochs, starting underweighting and finishing overweighting it.

These changes led to a 90% of F1 score during the validation step in just 50 epochs, that is when a decided to stop it as it reached an acceptable value.

The test run confirms the 90% of F1 score.

5 2nd Baseline, GRU

The second method was a sort of ablation study, to try to understand the effectiveness of the partition filter technique.

The development of this method was done in parallel with the SOTA one, so the difficulties, tricks and alternative ideas were the same.

Instead of the peculiar partition filter encoder used in the SOTA, this method used a simple Gated Recurrent Unit from pytorch, used in bidirectional mode, that outputs a tensor that is then splitted in 3 parts to mimic the SOTA partitions.

The rest of the method is the same of the SOTA one to be able to analyze this change.

The training and validation results were faster then the SOTA, reaching a 91% of F1 score in just 37 epochs.

The test run ends with a 90% on F1 score.

6 3rd Baseline, LSTM

The last method was also the first implemented.

After a lot of experiments to study how to deal with this task, what technique to use, thinking about convolutional ways, simple linear units or simple recurrent ones, I finally decided to use two bidirectinal LSTM.

For this task I decided to do just the RE task, for reasons that vary from the use of less elements in the model to time and resource reasonings.

The idea behind this model is inspired by [1], a modification of [2], that treats in different steps subjects and objects, and uses additional embeddings (of relations in this case) to focus on some elements.

I used this idea for skipping the NER task and to put the focus on a single entity; after having tried repeating the desired entity (or entities) at the and of the sentence and/or inserting a special tag before and/or after the desired entity.

First thing to do was to extract the entities in the dataset, iterating on the subjects and objects of each triplet. This explains the 2 different behaviours in the dataset's getitem() method.

The model tries to predict if each entity is a subject (first LSTM) and an object (second LSTM) for some relations, starting from the hypothesis that subjects and objects, are differently recognizable given their different position in the sentence in relation to the elements that identify a relation. In order to do so, the given tokenized entities are first embedded and then reduced to a single vector of features (an entity can be of more than just one token). This vector is then summed over the whole sentence and then passed to the LSTMs, that will output the probabilities that this entity is a subject/object for each relation.

Labels for this method are divided in subject and object labels, and only in the final metrics the triplets are composed, putting together subjects and objects that share the same relation.

The same logic is used in the metrics for the first two methods, were NER and RE results are analyzed together in order to validate the triplets.

The training was stopped at the 50th epoch with an 89% of F1 score, and the test run confirms the result.

7 Comments on inference and dataset

Inference experiments on trained models confirm the doubts about some incorrectness in data.

From the triplets predicted it seems like the models also memorize some entity that maybe appears frequently in the train samples. For example, in the sentence ”Andrea is a pilot of Apollo 12, Alan Shepard was born in USA” all the three models predict ”Shepard, pilot, 12”, that of course is incorrect as ”Andrea” is the subject of this sentence.

This example was done intentionally as the Alan Shepard sentences are one of the first in the data splits and repeated very often.

The overall feeling is that this dataset comprises too many examples of the same shape and kind, sentences are very similar, with the same subjects and object, not just the relation and this can result in models that learn also the repeated entities.

Maybe that’s because this dataset is just a little fragment of the original WebNLG, which is a lot bigger and should contain sentences of more different topics.

After little analysis I did into the original one with some experiment with dummy models, I found out that it should be written better, lot of entities with different writing styles, entities written with underscores instead of spaces etc.

All of that could give confusion in the trained models, or maybe be a good thing as the model could be able to generalize more the entity recognition.

References

[1] Yang J. et al. Li X. Li Y. “A relation aware embedding mechanism for relation extraction.” In: *Appl Intell 52*. 2022, pp. 10022–10031. URL: https://doi.org/10.1007/s10489-021-02699-3.

[2] Zhepei Wei et al. “A Novel Cascade Binary Tagging Framework for Relational Triple Extraction”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2020, pp. 1476–1488.

[3] Zhiheng Yan et al. “A Partition Filter Network for Joint Entity and Relation Extraction”. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, 2021, pp. 185–197. URL: https://aclanthology.org/2021.emnlp-main.17.