AdaElectra — Adaptive pooling based approach with Electra for Relation Classification

Aditya Shah

October 2021

1 Introduction

Relation classification is the task of classifying semantic relation between two entity pairs in a given sentence. Identifying such relations can be extremely helpful for downstream tasks such as robust knowledge extraction, social media analytics, etc. In this assignment, I present a noval Adaptive pooling based method on top of Electra model — *AdaElectra* in order to classify relations.

2 Objective

Formally, given a sentence S with tokens: $s_1, s_2, s_3, ...s_n$, which our goal is to classify relation between any pair s_i, s_j where $i, j \in S$. Here these pair of tokens (s_i, s_j) are known as entities. Any such pair of entities may exhibit a relation r that we aim to classify. Further, a given entity pair can have more then 1 relation

3 Data Preprocessing

Given an input sentence S, it is first converted into tokens $s_1, s_2, s_3, ...s_n$. Next I fetch the entity pairs from the tuple file and then assign special tokens $< s1 >$ before the start and $< e1 >$ after the end of entity one in the sentence S. Similarly, I assign token $< s2 >$ and $< e2 >$ before the start and end of entity two respectively. For eg:

Given sentence: *"then the terrorism struck again, this time in the indonesia capital of jakarta."*

Given entity pair: *(indonesia, jakarta)*

Preprocessed sentence: *"then the terrorism struck again, this time in the* $< s1 >$ *indonesia* $< e1 >$ *capital of* $< s2 >$ *jakarta* $< e2 >$*."*

We have been given 29 relations. So we convert the relation to ids and assign a relation id to every entity pair for a sentence.

Negative sampling:
In order to make sure that our model does not predict unnecessary entity pairs we create negative samples for every entity pair which is not present in the sentence.

So, let's consider we have 3 distinct entities (e_1, e_2, e_3) and we have relation r_1 for pair (e_1, e_2) and r_2 for (e_1, e_3). Here, we will also consider all the possible permutations i.e $(e_1, e_2); (e_1, e_3); (e_2, e_1); (e_2, e_3); (e_3, e_1); (e_3, e_2)$ and then assign a dummy relation *other* to the pair of entities that are not given in the input tuple file.

Entity Mask Sequence: Additionally, I compute an entity mask sequence for every sentence. This mask sequence will have token 1 for the span of entity 1, token 2 for the span of entity 2 and 0 for the rest of the text tokens. Padding is denoted by adding an additional token 3. For eg:

Given sentence: *"[CLS]then the terrorism struck again, this time in the* $< s1 >$ *indonesia* $< e1 >$ *capital of* $< s2 >$ *jakarta* $< e2 >$*."*, we will also have a following entity mask sequence:[SEP] (Electra tokenizer adds special tokens CLS] and [SEP])

Entity mask sequence: $< 00000000000111100022200 >$.

For padding, given a max length limit we append token 3 if the length of the sequence is less then the max length.

Finally, the encoded sentence representation with the entity mask sequence is given as an input to our model.

4 Model Architecture

Figure 1 shows the overview of the proposed architecture.

Electra Model: The input encoded sequence is passed through the Electra model. Then, we obtain the last hidden states of the Electra model. Next, we apply *adaptive max pooling* over all the hidden states of the model in order to compute a dense vector which represents the overall sentence encoding.

Entity Embedding Module: Next, the entity mask sequence is passed through an embedding layer to get a embedding for every token pair. Here our vocab size for the embedding will be 4 (as we have 4 tokens - 0, 1, 2 and 3) with token 3 used for padding. After we obtain the embedding, then we apply Adaptive max pooling over all the vectors to jointly form the entity encoding vector. We pass this entity encoding vector through a linear in order to final representation of the entity mask sequence.

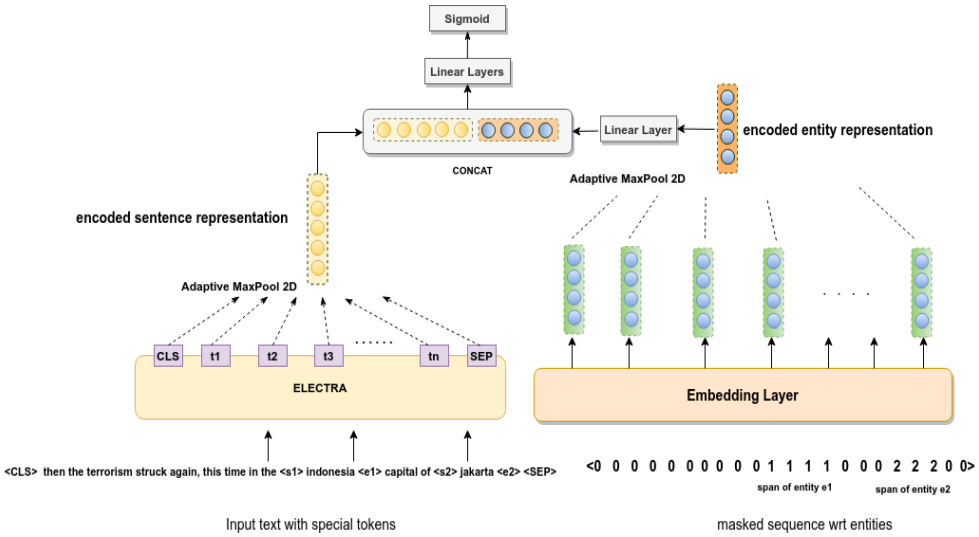

Figure 1: Overview of our proposed model

Concatenation: Next, we concatenate the entity encoding vector along with the sentence encoding vector obtained from Electra model and then pass it through dense layers. Finally we apply sigmoid over the last linear layer in order to get the probabilities for every relation class.

5 Results and Analysis

Given table shows the performance of the proposed model on the training, validation and test data.

Table 1: Results

Mode	Precision	Recall	F1-score
Train	0.949	0.920	0.934
Val	0.95	0.952	0.954
Test	0.956	0.829	0.888

Since we use special tokens for transformer model, so the model is aware of the relative positions of the entities in the sentence. Additionally using entity mask sequence helps the entity embeddings to learn position aware representations with respect to the input sentence. Thus, concatenating these vectors results in efficiently classifying the relations as the model is able to learn from both position aware entity encoding vector as well as the contextualized sentence vector from Electra.

Table 2 show the classification report generated by the model on the test data:

Table 2: Classification Report on Test dataset

Class	Precision	recall	f1-score	support
/location/administrative_division/country	1.00	0.99	0.99	380
/location/administrative_division/country	1.00	0.99	0.99	380
/location/country/capital	0.99	1.00	0.99	485
/location/country/administrative_divisions	0.94	0.97	0.95	380
/location/neighborhood/neighborhood_of	0.98	0.97	0.98	65
/location/location/contains	1.00	0.73	0.84	2518
/people/person/nationality	0.99	0.99	0.99	668
/people/person/place_lived	0.88	0.93	0.90	427
/people/deceased_person/place_of_death	0.63	0.58	0.61	62
/business/person/company	0.99	0.99	0.99	285
/location/us_state/capital	0.00	0.00	0.00	38
/people/person/place_of_birth	0.78	0.32	0.46	146
/people/person/children	1.00	0.73	0.84	26
/business/company/founders	0.89	0.72	0.80	90
/business/company/place_founded	0.93	0.70	0.80	20
/sports/sports_team/location	1.00	1.00	1.00	10
/people/person/ethnicity	1.00	1.00	1.00	12
/people/ethnicity/geographic_distribution	1.00	1.00	1.00	123
/people/person/religion	1.00	1.00	1.00	5
/business/company/major_shareholders	1.00	0.23	0.37	44
/location/province/capital	0.00	0.00	0.00	11
/location/br_state/capital	0.00	0.00	0.00	2
/business/company/advisors	0.00	0.00	0.00	8
/film/film_location/featured_in_films	0.00	0.00	0.00	2
/film/film/featured_film_locations	0.00	0.00	0.00	2
/location/us_county/county_seat	0.93	0.70	0.80	20
/time/event/locations	0.00	0.00	0.00	4
/people/deceased_person/place_of_burial	0.00	0.00	0.00	8
/people/place_of_interment/interred_here	0.00	0.00	0.00	8
/business/company_advisor/companies_advised	0.00	0.00	0.00	8
micro avg	0.96	0.83	0.89	5857
macro avg	0.59	0.51	0.54	5857
weighted avg	0.96	0.82	0.87	5857
samples avg	0.93	0.87	0.89	5857

We observe a high precision, f1 and recall score for classes like */location/country/capital*, */location/administrativedivision/country* as they are common relations and are present in majority in the dataset. On the other hand, we can see that for some classes like */time/event/locations* or *location/province/capital* which had low samples, the model was unable to predict any of them. This is usually a

challenge in multi label classification where we have unbalanced dataset. The model usually gives high scores for labels which are in majority and does not generalize well to labels that have low samples.

To see the effectiveness of adaptive pooling, I tried an ablation experiment by replacing the adaptive pooling module with [CLS] token output. So instead of applying adaptive pooling, I simply use the [CLS] token output as the sentence representation.

Table 3 shows the results on the test data for both the models. We get better score for the approach with adaptive pooling as compared to when simply taking the [CLS] token.

Table 3: Ablation study on Test data

Mode	F1-score	Precision	Recall
with [CLS]	0.872	0.963	0.801
with Adapt. pooling	0.888	0.956	0.829

One reason could be that max pooling would allow only most important semantic words (high vector values) to be considered for the computation of output sentence vector. Thus, this might be able to represent the overall sentence vector in a more constraint manner where most important words are given a preference. So it might lead to an improve in the performance.

In essence, the proposed model — *AdaElectra* performs good with about **0.888 F1** on the test dataset.