Language Models as Knowledge bases? (P5)

Mattia Paravisi 08395A
Instructor: Dott.ssa Darya Shlyk

November 24, 2023

1 Introduction

As result of recent advancements in natural language processing new language models with incredible capacities to comprehend, summarize and produce human language has emerged. This project examines the possibility of exploiting the internal state of these kind of models to create a different source of factual knowledge with respect to traditionally crafted knowledge bases.

2 Language models and knowledge encoding

Language models are trained on large amount of linguistic data to predict the next word given the previous context; this kind of training process exposes the models to a broad range of linguistic patterns and structures allowing them to generate contextually appropriate and coherent texts. In addition to learning the linguistic properties of the language to which the language model is exposed, it appears that it also learns factual knowledge during the learning process.

For example a language model can learn the fact "Dante was born in Florence" from the textual information it has been trained on.

The crucial question is whether or not this knowledge, which is embedded within the model's parameters, can be extracted efficiently and used as knowledge base.

3 Traditional knowledge base construction

Construction of traditional knowledge bases involves a multi-step approach which relies on natural language processing techniques: entity recognition, relation extraction, co-reference resolution and entity linking. The problem with this approach is that the quality of extracted knowledge depends on the accuracy of the individual components: errors in pipeline's early stages can propagate and result in inaccurate or incomplete knowledge base.

4 The dataset

The recommended dataset for this project is the Google Relation Extraction Corpus which can be divided in sub-datasets by grouping the examples using the relation predicates that are: *place of birth, place of death, education degree, institution, place of birth.*

Each example in the dataset is composed by four main elements: *object*, *subject*, *relation predicate*, *proposition.* Given as example the proposition "Stephen Hawking graduated from Oxford" is possible to extract:

- Object: Oxford
- Subject: Stephen Hawking
- Relation predicate: Graduated

For each quadruple $(subject, object, proposition, relation\ predicate)$ are also available five human judgments each telling whether or not, for a specific reviewer, the quadruple is correct.

Object and subject are represented using the Freebase MIDs while relation predicate is represented using Freebase properties.

For example the previous triple will be represented as:

- Object: "/m/07tgn"
- Subject: "/m/01tdnyh"
- Relation predicate: "/education/education/institution"

For this project the considered datasets are: *place of birth* and *date of birth.*

5 Chosen model

There are a lot of language models one can choose between. In paper [1] , for example, BERT is identified as the most suitable model to build a knowledge base. BERT comes in cased and uncased version: the first one takes into account the differences between upper cases and lower cases characters, the latter makes no differences. For the purposes of this project the differences created by upper or lower cases letters are irrelevant, for this reason I selected **BERT uncased** as model.

As suggested in paper [1] to access the knowledge stored in a language model one can craft a query and complete it with **mask token** where needed; for example given the quadruple from section 4 one possible query can be:

"Stephen Hawking graduated from [MASK]".

BERT, with its encoder architecture, is well-suited for masked language modeling. Furthermore BERT was trained on two specific datasets: Wikipedia and book corpus, the propositions contained in Google RE datasets are taken from Wikipedia, this guarantee consistency between crafted queries and the embedded model's knowledge.

6 Experiments with BERT

I wanted to test if a pretrained language mode, without any further tuning process, can be used as a knowledge base for certain types of relation predicates.

In [1] authors decided to use (in addition to others) Google RE *birth date, birth place and death place* sub-datasets to study if the querying method exposed in chapter 5 could effectively serve as a knowledge base for person's year of birth, place of birth and place of death.

I decided to take into account just the sub-datasets of Google RE in which the relation predicate regarded birth date and birth place for a main reason: in [1] the performances obtained by the model on place of birth and place of death are better than performances obtained by other models on the same sub-dataset and are also very similar; this fact was not surprising to me: the tokens the model has to predict in both relations are "similar" in the sense they are related to a specific geographic place. This is different if one thinks about the relation birth date:

- The kind of tokens the model has to predict are not concrete as places but are abstract.
- The performances on this relation predicate are incredibly lower than birth date or birth place:
  - P@1 place of birth: 14.9
  - P@1 date of birth: 1.5

I was intrigued by this difference in performances between date of birth and place of birth: was it due to the kind of tokens? Was it due to the model itself?

To find it out i tried to directly query the model and check the precision performances on birth date relation. It is not possible to use the sub-datasets as they are because object and subject encodings are not suited to directly build queries, a preprocessing step is needed: one can use Wikidata's API to retrieve from MIDs the relative string.

Once the dataset has elements in the form:

- Object: objString
- Subject: subjString
- Relation predicate: relpredString

one can easily create a fixed query structure:

```
if relpredString == '/people/person/date_of_birth':
    text = subjString + " (born [MASK])."
elif relpredString == '/people/person/place_of_birth':
    text = subjString + " was born in [MASK]."
```

and then query the model:

```
preds = mask_filler(text)
```

where mask_filler object is simply created using a pipeline:

```
from transformers import pipeline

model_checkpoint = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_checkpoint,
                                          use_fast=False)
model = AutoModelForMaskedLM.from_pretrained(model_checkpoint)
mask_filler = pipeline(
    "fill-mask", model=model, tokenizer=tokenizer
)
```

The structure to run the test is pretty straightforward:

1. For each pair (*subjString*, *objString*) create an appropriate query based on *relpredString* where *objString* is replaced by $[MASK]$. Call the query q.
2. Query the model using q. Get an array of prediction A.
3. A is sorted using the probability for every token as sorting key.
4. Check is $A[0] ==$ *objString* or if any object in A is equal to *objString*.

Starting with, as previously mentioned, relation *date of birth*:

```
count_pred_dob = 0
for (sub, pred) in zip(df['subj'], df['pred']):
  if pred == '/people/person/date_of_birth':
    text = sub + " (born [MASK])."
    ...
```

in this particular cases I checked if the most probable **numeric token** was the right one. The result is:

count: 38, total cases number: 2034, precision: 0.018682399213372666

This result is good: (also considered the paper's results) the precision I have obtained on 2034 examples, an higher number with respect to the 1825 examples in paper, is slightly higher than the one reported (1.5%). This is probably due to the different preprocessing steps I made with respect to the paper whose include/exclude different examples; the results are, in the end, comparable.

I made some experiments with other queries as shown in table 1. I found some of this query structures looking at Wikipedia dataset BERT was trained on: every time it includes a proposition regarding a specific person and his/her date of birth the structure is more or less "**person (birth place, birth date - death place, death date)**". Is easy to see how the query schema affect the model performances. The query has to specifically suite as well as possible the linguistic structures the model has learnt. An interesting case is related the first query structure reported in table 1: one expects that the model predicts a numerical token because of "year" in the query but this is not true: most of

the times the model's prediction are non-numerical tokens as "nine", "seven", "approx"; this behavior is due to the fact that the model interprets "year" as "age" and not as "calendar year".

Query	Total number of cases	Correct cases	Percentage
sub + " born in year [MASK]."	2034	0	0.00 %
sub + " born ca. [MASK]."	2034	0	0.00 %
sub + " (born c. [MASK])."	2034	22	1.0 %
sub + " (born ca. [MASK])."	2034	24	1.1 %

Table 1: P@1 for different queries schemas

The second part of this test regarded the dataset *place of birth*. The code is the same as before, the query differs:

```
count_pred_pob = 0
for (sub, pred) in zip(df_2['subj'], df_2['pred']):
  if pred == '/people/person/place_of_birth':
    text = sub + " was born in [MASK]."
    ...
```

count: 352, total cases number: 3418, precision: 0.10298420128730251

In this particular case the model is far more precise, in 10% of cases it founds exactly the right object. This is slightly less than the result reported in the paper (14%), but also here the number of example is different: in the dataset I used there are 3418 examples, the paper refers to a dataset with 2937 examples; as in the previous case the results are comparable, considered also the different preprocessing steps.

I wanted to, in general, comprehend the possible errors made by the model when it was predicting a wrong token. This analysis was made by comparing the predicted object for a query with the real object. For example i considered the query:

giovanni preziosi was born in [MASK].

the model proposed as most probable token "milan" and as second most probable "venice", the right answer was "torella dei lombardi". What i suppose is that in this case "torella dei lombardi" is a rare token and the model barely saw it, for this reason is very unlike that the model will provide it as most probable token. Another interesting example is the following:

antonio ciacca was born in [MASK].

the two most probable tokens for the model are "rome" and "naples" but the correct token is "germany". In this case the problem is similar to the previous one: "antonio" is a typical italian name and "ciacca" as a surname is widespread more in southern Italy. Here, in my opinion, the problem is not related to the fact that the model is unable to recognize the context (because the model

recognizes it) but, on the contrary, is related to the fact that it has to infer the context using just the subject. If it is a very specific subject, for example Dante Alighieri, then the model has no difficulties in finding the correct token. But if the subject is generic then is very difficult for the model, but also for humans, to find the correct token.

7 Model's precision

In this section are reported the precision results achieved by the model on both the considered datasets. Table 2 is a summary of the number of total cases in a specific dataset and the number of correct tokens the model found for all the queries at k. For example the results in column "Correct at 1" are equal to the results previously reported because I considered just the most probable token to do the comparison with the correct one. Is easy to see that if one consider the k most probable tokens the number of times the model is right increases.

	Cases number	Correct at 1	Correct at 2	Correct at 3	Correct at 4	Correct at 5
Date of birth	2034	38	63	82	104	134
Place of birth	3418	352	480	568	651	716

Table 2: Total number of cases for each dataset and corresponding correct predictions at k

	P@1	P@2	P@3	P@4	P@5
Date of birth	1.8	3.0	4.0	5.1	6.5
Place of birth	10.2	14.0	16.6	19.0	20.9

Table 3: Precision at k (as percentage) for BERT base

8 Adding context to the queries

The queries used in paper [1] are constructed as mentioned above:

subject + some sentence + [MASK].

I therefore thought of modifying the method proposed by the authors to understand if it was possible to obtain better performances or better understanding of model's behavior. Considering the fact that I am just building a query, in order to avoid dramatically changing the whole experiment's architecture, I decided to add more **contextual information** to the query itself.

Lets consider as an example the query:

jack curtner was born in [MASK].

and the most probable predictions:

london, chicago, toronto, england, philadelphia

those are all wrong predictions because the right object is "greenville" which is a city in South Carolina. As previously mentioned i decided to add contextual information in the query schema to, maybe, enhance the method performances. For place of birth relation predicate a contextual information can be to add the place of birth of another person: I tried to modify the query schema as follows

subjectA was born in objectA. subject was born in [MASK].

for example:

bill gates was born in seattle. jack curtner was born in [MASK].

the most probable prediction for Jack Curtner's place of birth changed drastically:

seattle, portland, washington, chicago, vancouver

Is easy to see that the predicted tokens are biased by the given context. To confirm this intuition I tried to put in the first part of the query a person which was born geographically distant from the object token I was considering:

friedrich hegel was born in stuttgart. jack curtner was born in [MASK].

as results I obtained:

stuttgart, berlin, zurich, munich, vienna

this confirms my thesis: adding context on a query bias the query results to be closer to the added context. I also run the same experiment as before to confirm that in general performances were worse; I also try to add more facts for a specific relation predicate as shown in table 4.

	P@1	number of right tokens
Place of birth	10	352
Place of birth with 1 fact	0.8	28
Place of birth with 2 fact	5.6	194
Place of birth with 3 fact	7.7	265

Table 4: Precision at 1 (as percentage) for BERT referring to place of birth queries with one or more specified facts.

Same thing happens for birth date:

gary sykes (born [MASK]). → 1966, 1965, 1974, 1960, 1958

bill gates (born 1955). gary sykes (born [MASK]). → 1957, 1956, 1958, 1959, 1960

A different scenario comes out when considering the (not yet studied) relation *place of death* which has the following query structure:

	P@1	number of right tokens
Date of birth	1.8	38
Date of birth with 1 fact	1.2	25
Date of birth with 2 fact	1.1	24
Date of birth with 3 fact	1.1	23

Table 5: Precision at 1 (as percentage) for BERT referring to date of birth queries with one or more specified facts.

subject + died in + [MASK].

Consider the following query:

steve jobs died in [MASK].

and the relative results:

office, 2010, london, 2008, 2009

If we change the query as:

louis armstrong died in new york. steve jobs died in [MASK].

The results are:

california, london, chicago, paris, boston

And if we consider the most probable token it is the correct one.

	P@1	number of right tokens	numer of total facts
Place of death	6.8	81	1188
Place of death with 1 fact	5.6	67	1188
Place of death with 2 fact	4.3	52	1188
Place of death with 3 fact	7.8	93	1188

Table 6: Precision at 1 (as percentage) for BERT referring to place of death queries with one or more specified facts.

In this case the bias created by the additional facts is useful: we use it to deviate the model from predicting certain kind of tokens.

9 Conclusions

In conclusion, the experiments on the Google Relation Extraction dataset have substantiated the findings presented in [1]: it is clear that a language model used as a knowledge base is better with respect to other methodologies; however, the attained accuracy results leave room for improvement. The suboptimal performance can be attributed to the model's knowledge graph which lacks on the total coverage of all examples within the dataset.

The proposed method of adding contextual information to the queries can be used to enhance performances on specific relational predicates while leaving unchanged, or slightly worsening, the performances for the other predicates. This show a possible way for addressing a part of the existing model's limitations. To further optimize performance, I recommend a fine-tuning approach on the model using the specific dataset employed for relation extraction.

In summary the project confirm the viability of language models as knowledge bases and underscores the potential for refinement proposing a new methodology to query the model.

References

[1] Fabio Petroni, Tim Rocktäschel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, Alexander H Miller, and Sebastian Riedel. Language models as knowledge bases? *arXiv preprint arXiv:1909.01066*, 2019.