Building a Knowledge Graph-based Dialogue System at the 2nd ConvAI Summer School

Michael Galkin Follow
Jul 16, 2019 · 10 min read

I recently returned back from the 2nd Conversational Intelligence Summer School organized by the Text Machine Lab of UMass Lowell and Neural Networks and Deep Learning lab at MIPT (iPavlov project), Moscow.

The School took place in Lowell, MA, USA, one of the first US industrial cities with remaining spirit of the industrial revolution.

Group photo!

Content-wise the school was pretty packed with fundamentals of NLP as well as bleeding edge of the progress in dialogue systems.

We had 5.5 days of lectures, hands-on tutorials, invited talks, different cuisines, and, of course, team projects! Starting from word embeddings and arriving to variational inference via attention and transformers just within 5 days — as you can see it was pretty packed. Invited speakers shared a lot of fruitful thoughts on the state of the art topics. For more information on the summer school schedule and talks please check out this post.

. . .

Knowledge Graphs and Dialogue Systems

In this article, I'd like to shed some light on the technical project we'd been working on the whole week and how DeepPavlov library could help you achieve this task. The project of our team aimed at improving question answering over knowledge graphs, namely, Wikidata in our case, and support a dialogue over subject entities.

Traditionally, question answering aims at answering over some unstructured context like a text paragraph or Wikipedia page. Datasets have been comprised accordingly, e.g., SQuAD, HotPot-QA, MS-Marco. However, exploratory capabilities of such QA systems are limited to the given context, e.g., having a wiki page about Albert Einstein one could obtain his birth place "Where was Albert Einstein born? — Ulm" but asking more questions about this city (e.g, "How many people live in Ulm?") won't return decent results. In this case, we need some structural background which knowledge graphs are happy to provide us with.

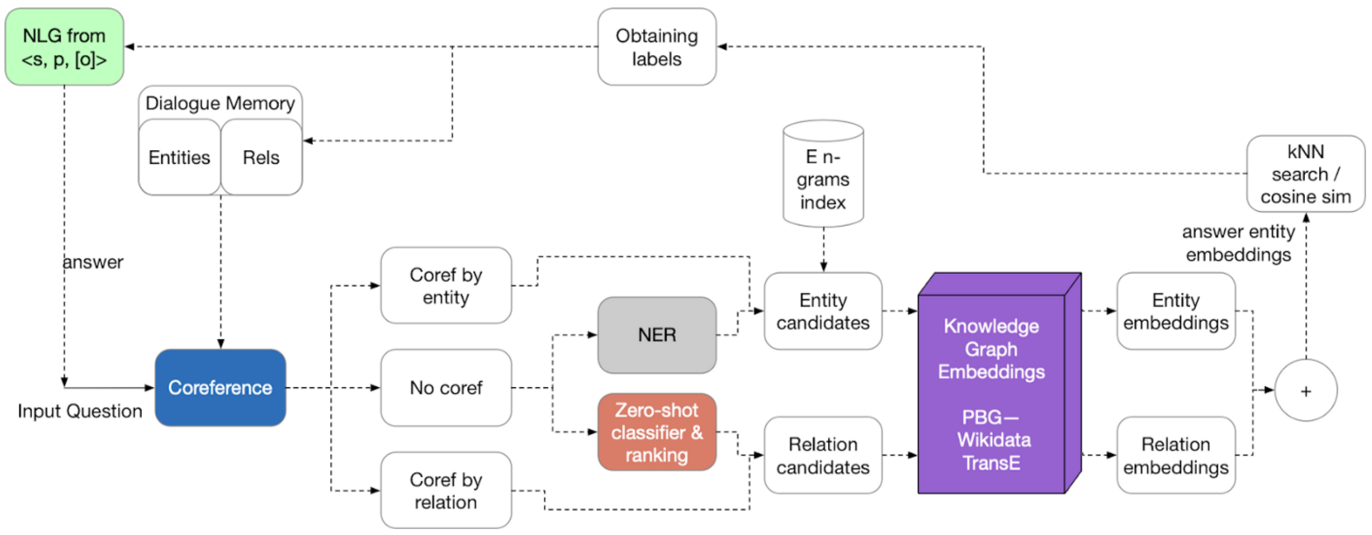
Knowledge graphs possess inherent and explicit semantics (via so-called vocabularies or ontologies that declare classes, properties and various constraints) which allow for sophisticated reasoning current language models are incapable of.

In our project, we went for Wikidata as the main source of factual information for all Wikipedia pages and languages. Wikidata is a huge graph — it consists of more than

50M entities and 4B facts. That is, everything you can find on Wikipedia is already in Wikidata, whereas some Wikidata entries might not yet be on Wikipedia. Moreover, big companies like Google contribute to Wikidata with their own datasets.

If you want to try it out — DeepPavlov provides the KBQA component out of the box ready to install. Currently it works for Russian language, but the general pipeline is pretty similar for any language.

At the summer school we were encouraged to try out some non-trivial approaches to known problems 🤔 . So we decided to go for a KG-based QA and dialogue system based on knowledge graph embeddings, and here is a basic sketch of the overall architecture. As you can see it's quite a modular architecture so DeepPavlov can provide us with a handful of pre-trained components!



A typical pipeline of a knowledge-grounded dialogue system

• • •

Stage 1. Coreference Resolution

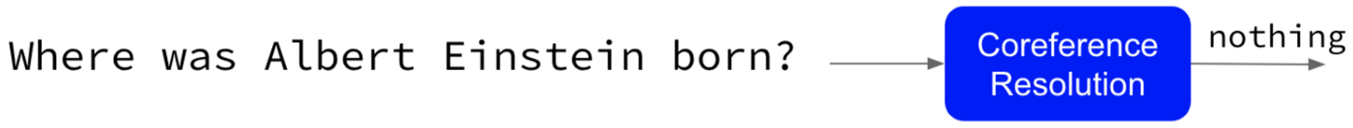
A user input question is processed first via the coreference resolution module. Coreference is pretty much a necessary thing when developing a dialogue system. Moreover, you'd want it working on both entity and relation levels. For a simple question like "Where was Albert Einstein born?" the coref won't return anything, but having a context:

- Where was Albert Einstein born?
- He was born in Ulm
- How many people live in this city?

The coref module should point to Ulm — this is an example of a coref over entities. Within the context

- Where was Albert Einstein born?
- He was born in Ulm
- And what about Max Planck then?

the coref should resolve to a birth place property from the knowledge graph (more details on that below). To alleviate the correct resolution we track history of mentioned entities and properties within a conversation in the dialog manager. Of course, resolving coreferences (especially complex) is still a major NLP topic. However, there are libraries and software components you could plug in as a baseline, e.g., neuralcoref from 🤗 or coref by Allen NLP.



No coreference resolved in this case

. . .

Stage 2. Named Entity Recognition

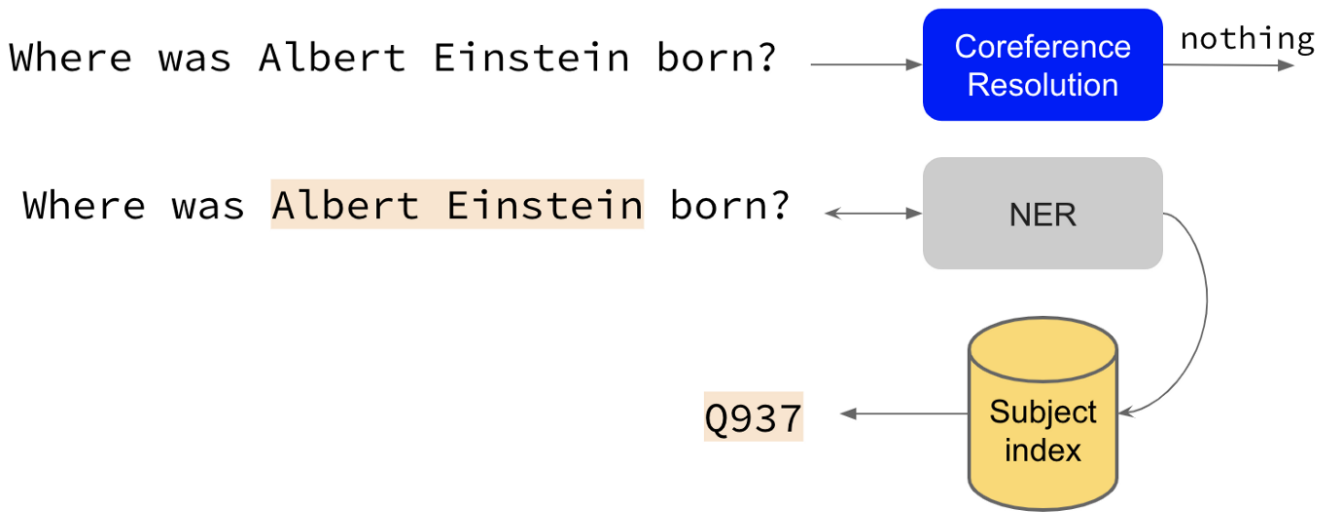
If the coreference resolution component doesn't find anything with high confidence, then it probably means you have some unseen named entities and relations, so it's time to find them out and link to the knowledge graph. Practically, if you give a NER system a question like "Where was Albert Einstein born?" it should return you spans of the given question, e.g., "Albert Einstein", that most probably refer to some named entity you could look up in the graph.

The task is somewhat tricky as out-of-the-box NER components usually support different number of named entity types (or tags) depending on the training set. There is a good selection of NER components from spaCy, Allen NLP, and DeepPavlov. If you go for the last one, you'd enjoy BERT-based NER with zero-shot transfer capabilities for hundreds of languages (yes, most probably it does support your native language) and state-of-the-art performance (F1 is 88.8% on OntoNotes compared to 85.85% of spaCy).

Once you extracted named entities you need to link them to the knowledge graph (or find out that there is no such named entity there). Well, KGs are huge — Wikidata contains over 50M entities with labels and synonyms in hundreds of languages.

Here we come into a grey area: even Wikidata does not embrace the infinite space of domain-specific named entities, say, from underwater archaeology or aircraft design. For such cases there are domain-specific knowledge graphs, so if you want to build a domain-specific dialogue system you'd need a graph with known named entities there.

Since we were aimed at general-knowledge system and had limited time and computational power we sampled and indexed Top-100 most popular pages from 10 categories using The Open Wikipedia Ranking and all entities within 2-hop subgraphs around each seed entity which is about 160K entities overall. Good number for starters! So we're now able to map named entities to their Wikidata URIs, for instance, "Albert Einstein" is described in Wikidata as https://www.wikidata.org/wiki/Q937.


Where was Albert Einstein born?
Coreference Resolution
nothing
Where was Albert Einstein born?
NER
Subject index
Q937

NER and subject detector identify a Wikidata entity corresponding to Albert Einstein

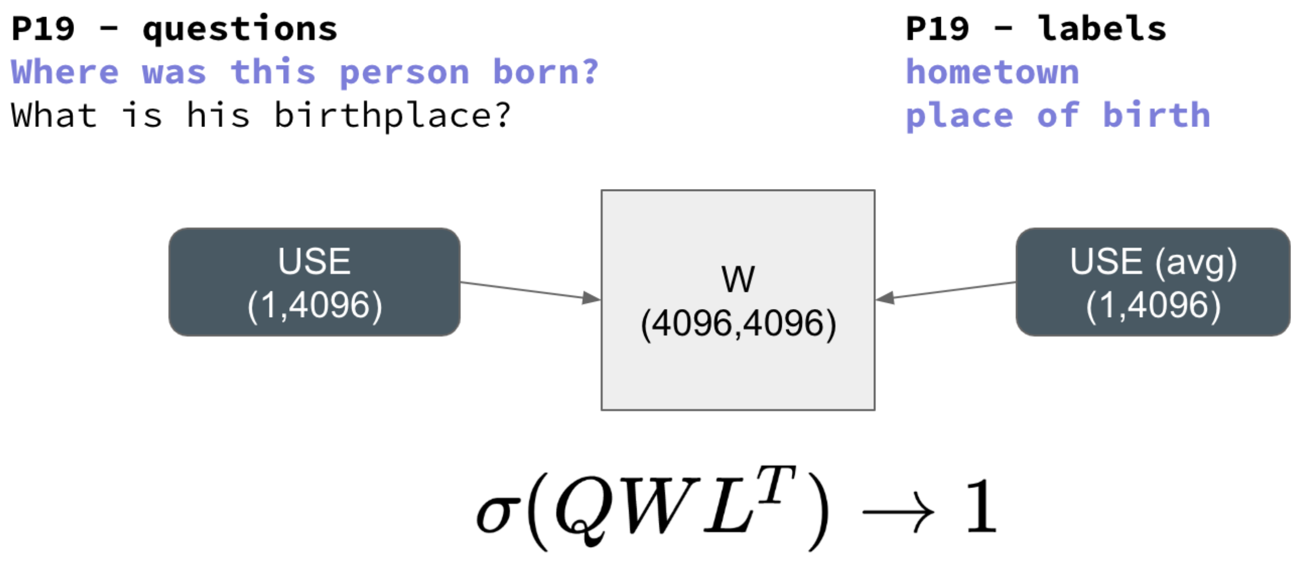
Note that we didn't tackle here some sophisticated context-based NER, e.g., there are several entities named Berlin in Wikidata — capital of Germany, town in Maryland, even American synth-pop band. So you might need to infer which Berlin you're dealing with from the conversation context, and that's still an on-going research topic with a manifold of papers at top conferences how to do it efficiently.

. . .

Stage 3. Relation Linking

Okay, so we have an entity from the knowledge graph now. What we need next is to obtain a relevant property from the graph a user is asking about, e.g., "place of birth" in Wikidata. Currently, there are more than 6000 properties in the graph, each of them has a label and maybe some synonyms. "Let's build a classifier!" — one would suggest. Well, we have a small problem here — we don't have enough training data to cover all of them. So far most of QA over KG datasets (LC-QuAD 2.0, SimpleQuestions, CSQA) employ just a subset of all existing ones, so how do you scale up for the relations not in the training set?

In our project, we decided to go for a zero-shot learning approach for relation linking. Here is a short schema how we did it:

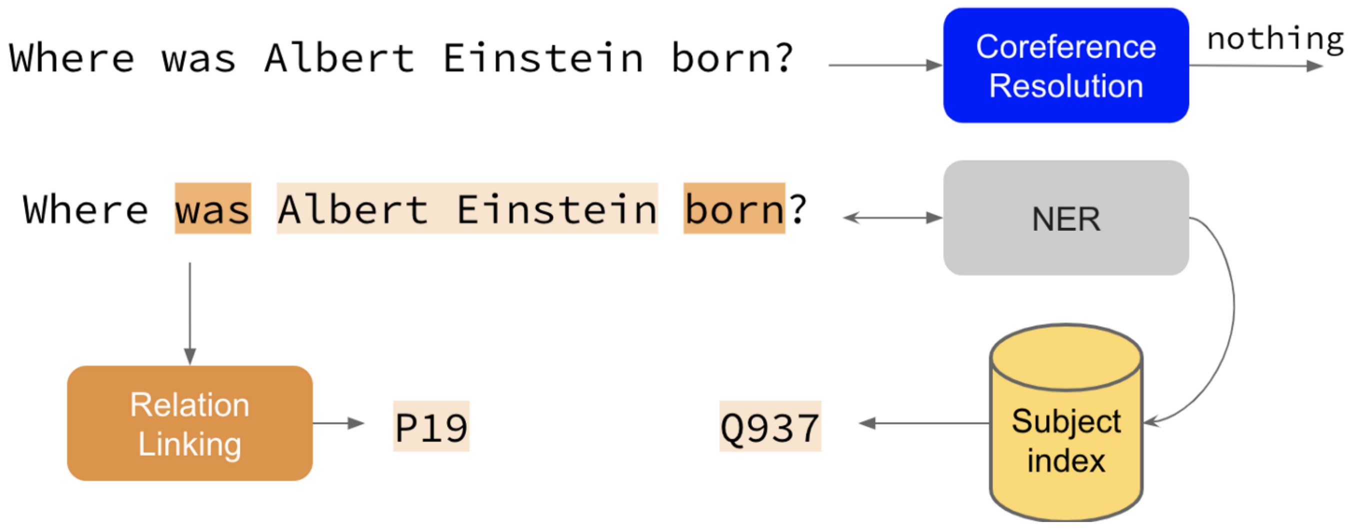


$$\sigma(QWL^T) \to 1$$

Given the questions from the above-mentioned datasets and labels+synonyms (e.g., “place of birth”, “hometown”) of Wikidata properties, we pass the training question through a sentence encoder, pass labels and synonyms through the sentence encoder (there are some pretty looong labels actually, so simple word embeddings might not work well) with a subsequent averaging of the embeddings, and train a simple classifier with a bilinear matrix to maximize the similarity score between a question and averaged labels. As to the formula above, `Q` denotes a question, `L` denotes averaged labels and synonyms, `W` denotes the optimized matrix.

We also sample a negative example, i.e., wrong relation, for each question to equalize the training set, so the similarity score should be close to 1 for positive samples and close to 0 for negative.

What can you use for sentence encoding? Universal Sentence Encoder for Tensorflow connoisseurs, InferSent for PyTorch aficionados, or pre-trained DeepPavlov components for a general Python public that just wants to import the model and it works :)



Relation Linker identified a Wikidata property ”place of birth”

Alright, so after this stage we have now Wikidata URIs for a subject and property, e.g., Albert Einstein and place of birth.

Note that separate Entity and Relation linking will still be working for some meaningless questions like “How many children does Berlin Hauptbahnhof have?”. If you want to know how the research community deals with such cases (for instance, joint entity and relation linking, or some graph constraints methods) — welcome to the field of QA over KGs 😉

. . .

Stage 4. Knowledge Graph Embeddings

So we have Wikidata URIs of candidate entities and relations, those resemble slots for a query, right? So why don't you just write a SPARQL query to Wikidata to fetch an answer?

At the summer school we wanted to try out something new, that is, knowledge graph embeddings of the whole Wikidata recently released by PyTorch-BigGraph. What are knowledge graph embeddings? They encode triples (subject-property-object) into a vector space. Different algorithms use different distance functions and are optimized against different loss functions, I can only encourage you to check out quite massive KG embeddings literature ;)

The BigGraph embeddings were trained using a translation operator (something like TransE) and L2 distance minimization between (subject + property) and (object) vectors.

$$\mathcal{L} = \sum_{(h,\ell,t)\in S} \sum_{(h',\ell,t')\in S'_{(h,\ell,t)}} \left[\gamma + d(\boldsymbol{h}+\boldsymbol{\ell},\boldsymbol{t}) - d(\boldsymbol{h'}+\boldsymbol{\ell},\boldsymbol{t'})\right]_+$$

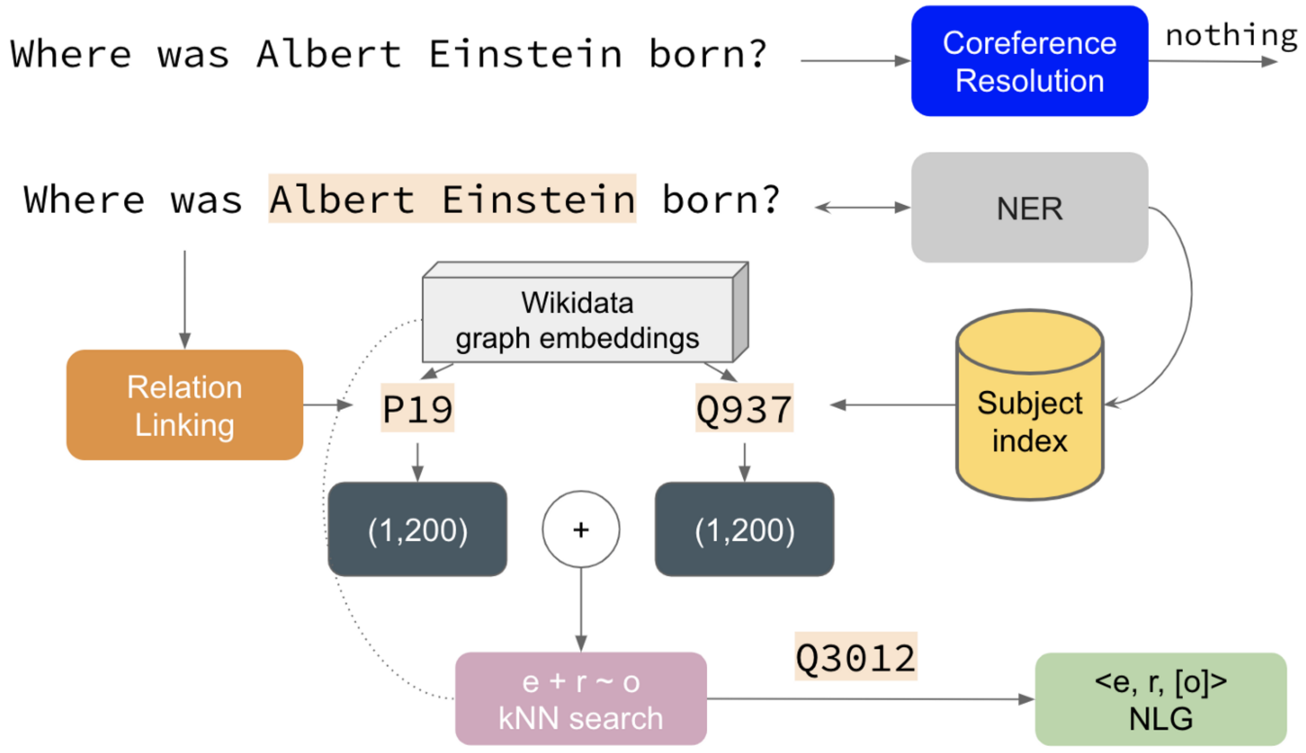
In other words, the training objective resembles a typical SPARQL query, so that when you add a subject vector to a property vector you would get a vector, close to the object vector (or a neighbourhood of them if there are many values).

Once you have a summed vector you can employ a library like faiss to perform kNN clustering in the embedding space to find the most similar entities in the whole embedding matrix.

Since the original matrix is just a 100 GB large tsv file with 70M lines (👀) we had to filter it with the 160K entities sampled for the NER stage and all 6K relations (thanks Perl). Numpy could't load the original file even onto 256 GB RAM server, so you might need to prepare a cluster for parsing and in-memory storage.

. . .

Stage 5. Natural Language Generation



Generating a natural language response based on Wikidata entities

Okay, so you obtained a Wikidata object URI either via SPARQL or graph embeddings. For our toy example case, it is Ulm. Now you have a tuple `(Albert Einstein, birth place, Ulm)` which needs to be verbalized to a human-friendly form. An "old school" approach is to have templates for each predicate like `"X was born in Y"` and then replacing variables with your subject and object label. However, when it comes to more sophisticated answers consisting of several entities or relations, your template base will likely exponentially explode. However, it produces predictable results and can be manually curated — that's why most voice assistants still resort to templates (and armies of maintainers).

But the progress in NLP gave us so many datasets (WebNLG, T-Rex) and powerful generative models (like this)! 😍 So that you can input a graph of various complexities and get coherent verbalisations sometimes even in a few sentences.

. . .

Conclusion

So that was just the first turn of the dialogue. Then, you can ask more questions in the subject, property or object contexts, for instance, "How many people live in this city?" (well, hoping that coref would resolve "this city" to "Ulm" in our toy example).

We didn't have much time to tackle other issues, some of them are: a chit-chat non-goal oriented conversation (the research area is rapidly growing as well!), agent personification (like in the ConvAI2 challenge), different intents treatment, complex questions with sophisticated graph patterns. It is really a great moment to embark on the Conversational AI ship! 🚢

I am really grateful to my team members, Evgeniia Razumovskaia and Arina Maltseva, for hard-working days and nights after the lecturing program 😍

. . .

Materials

All the keynotes, slides, videos and code of the summer school are available in the github repo. Be sure to check them out!

Many thanks to the organizing team for a great week full of super-interesting talks, delicious food and broad networking opportunities!

See you next time 😉

From organizers

We hope this was helpful and you'll be eager to use the DeepPavlov library😃 . And don't forget DeepPavlov has a forum — just ask us anything concerning the framework and the models here, we'll reach out to you back ASAP. Thank you for reading!

Thanks to Darya Moroz.

Machine Learning Dialogue Systems NLP Deep Learning Community

About Help Legal