Summary

Since the midterm, we have increased the scale of our data since we thought our sample dataset of 5000 Wikipedia introductions was too small. Additionally, we looked into ways of embedding the Wikidata knowledge graph, and our knowledge graph embeddings show great results. We then started working on using deep learning models to approach the task of named entity linking. We thought of two similar approaches to use with some initial results that we plan on improving and experimenting with further. Finally, we are currently planning on building an end-to-end pipeline as our final product.

Upgraded Data

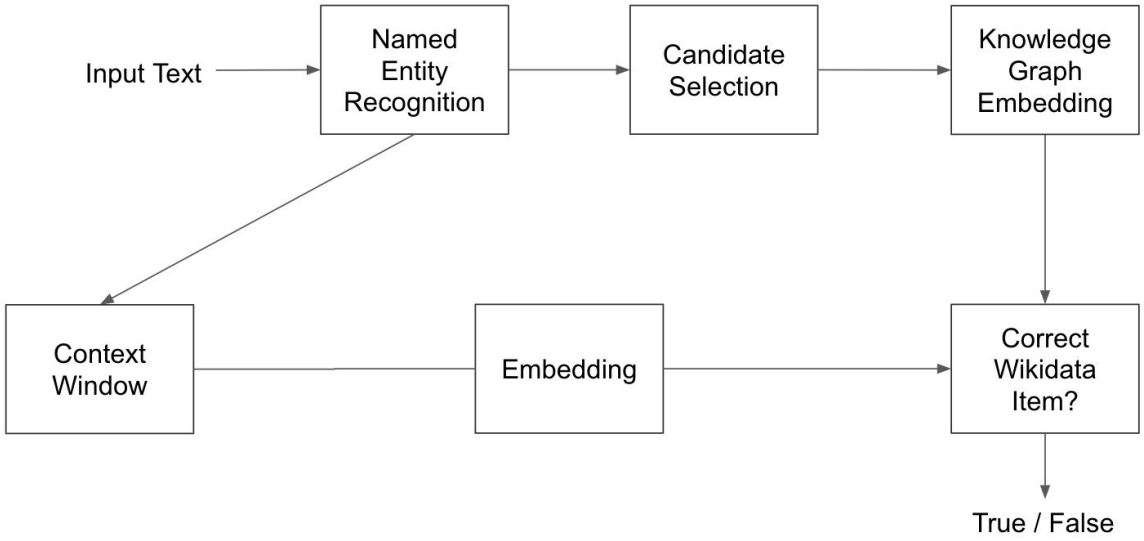
We are currently working on ~1% of the Wikipedia article introductions (385000), up from 5000 previously. We are requesting for more computational resources, but the size of the current dataset seems sufficient for training purposes. In addition, we are training on actual Wikipedia hyperlinks rather than NER identified entities. This allows us to focus on the entity disambiguation stage. As such, all the metrics reported do not account for errors in the NER and candidate selection stages.

Knowledge Graph Embedding

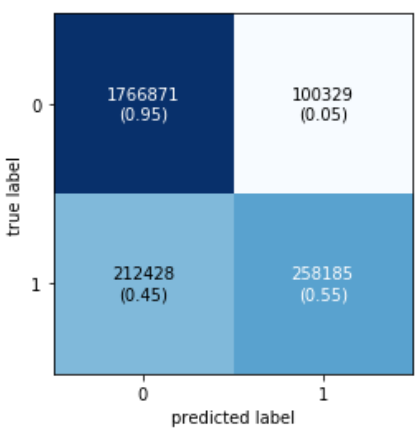
As a way to incorporate the information from the knowledge graph, we train RotatE, state-of-art knowledge graph embedding by Sun et al. (ICLR 2019), on Wikipedia dataset to embed our entities to capture the graph structural information. We get 90% test accuracy on the task of knowledge graph completion - predict head or tail in a (head, relation, tail) triplet. It's not been possible for us to train on the Wikimedia entities since there's about 60 million of them (around 8 million for Wikipedia only), which would require about 80GB of GPU memory.

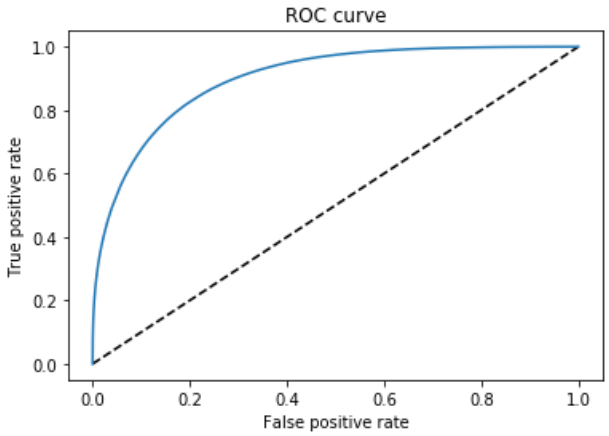
Model 1

For our first deep learning model, we approach the model as a classification problem. For a candidate in the candidate's list for an identified entity in a text, the model predicts whether this candidate is the true entity for the identified entity. Specifically, given the knowledge graph embedding of the candidate and the local context embedding of the text as inputs, the model predicts true if the candidate is the correct entity and false otherwise. The local context is just a window around the identified entity (ex. 10 words before the identified entity and 10 words after), in which we then feed into an embedding layer, which we train as part of the model. We feed the embedding into a LSTM, which we concatenate with the knowledge graph embedding, and this is fed into a neural network. With this model, we basically are trying to find a mapping because the knowledge graph embedding and local context embedding.



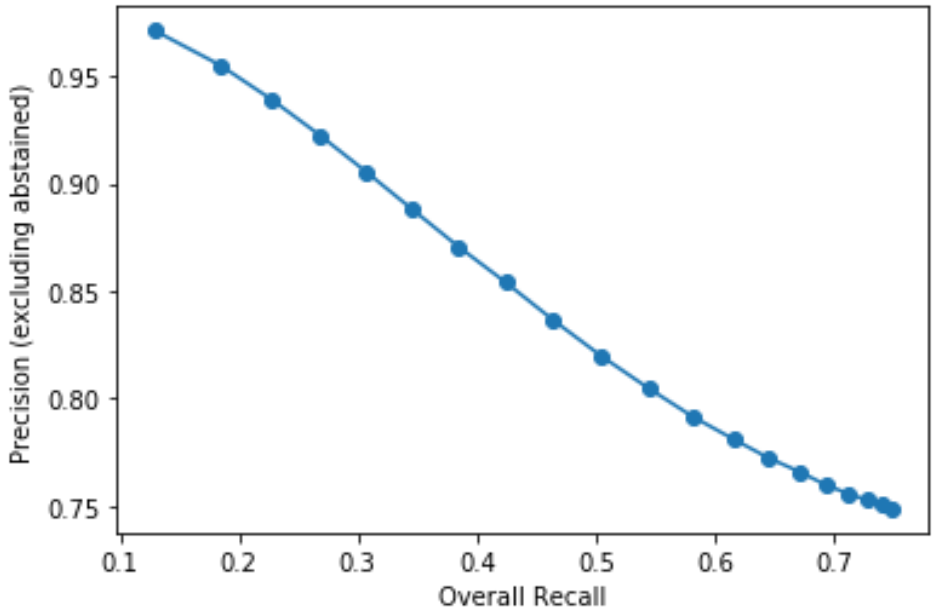
We make training and test sets, in which for each entity in our Wikipedia introductions we put the true entity with label true and 10 false entities from candidate selection with labels false if possible. In total, we had a total of 11701294 rows, with 20.11% positive and 79.89% negative labels. We split the data into a train/test set, ensuring that the candidates for the same identified entity are in the same set to prevent leakage. We achieved a test accuracy of 0.8662 and AUC of 0.8992. This seems like a promising result that we plan on improving.





We also evaluate the performance of the model on the candidate list from candidate selection, where the candidate with the highest probability is picked. Note that since we're only interested in evaluating NED for now, we always include the true reference in the candidate list in the case it's missing. The test accuracy is 75%. In addition, we experimented with the option of abstaining the prediction since in reality a candidate list may not contain the true candidate. This is done by abstaining when the highest probability is below a certain threshold. Here, we can

see a plot of precision (excluding the abstained predictions) vs overall recall tradeoff at different thresholds, where the distance from one point to the next is 0.05 change in the threshold.

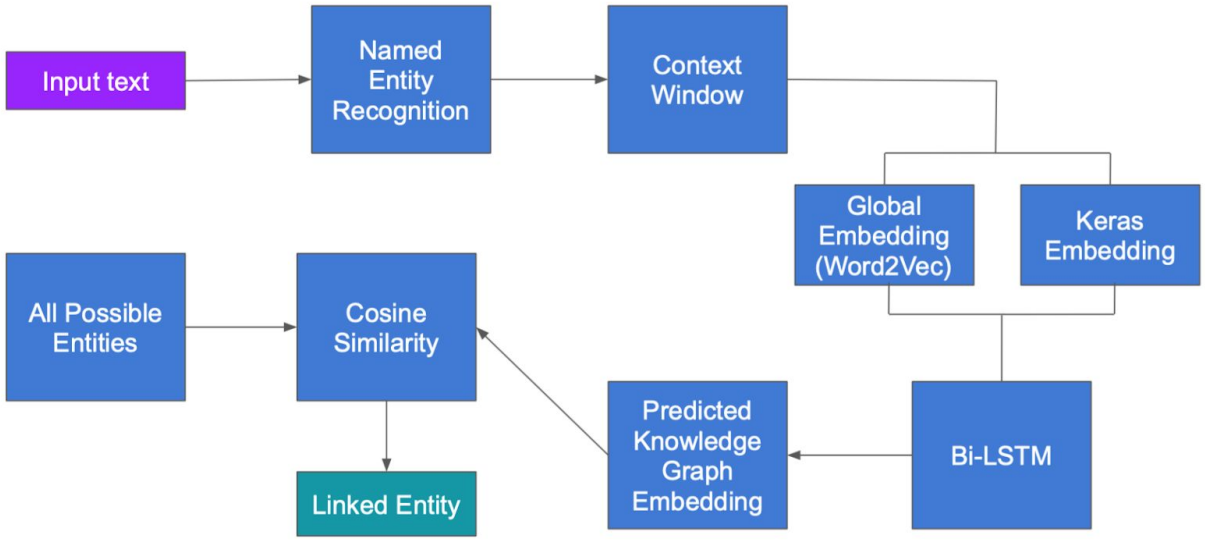


For future work, we plan on trying out BERT and ELMo to get word embeddings for the local context to see if we can further improve this model. ELMo and BERT are much more powerful in that they are context dependent, generating different embeddings for the same word under different context. ELMo is a character based model where the learned representations are words, whereas BERT is a subword based model where the learned representations are subwords as well.

Model 2

Our second model is quite similar to the first model; the major difference is that now we want the model to learn/predict the knowledge graph embeddings that correspond to each entity. For example, given the context window “to drive my **Jaguar** around the city”, the aim of this model is to predict the correct knowledge graph embedding for the car, Jaguar.

For each context window, we create two embeddings: the globally trained Word2Vec vectors for each word, and the embeddings trained using a Keras embedding layer. We concatenate the two embeddings at a word level, meaning each word is now a vector consisting of the concatenation of the two embeddings. The sequence of vectors for each context window is then fed into a Bi-LSTM, followed by an output Dense layer to predict the knowledge graph embedding for each context window. We can then choose the entity that has the largest cosine similarity between its embedding and the model output embedding. Below is a diagram showing this pipeline.



Unfortunately, this model currently is failing to learn the knowledge graph embeddings. It mostly predicts embeddings that have values near 0, which is near the mean of all the knowledge graph embeddings. One possible solution is to input the average embedding of the top *k* candidates for each context window. Therefore, the model only needs to learn the variation from the averaged embedding, rather than learning a new embedding from scratch.

Pipeline

We intend to eventually produce a full information extraction pipeline (entity recognition, candidate selection, and entity disambiguation). The first two stages will be standard readily available models, i.e. SpaCy or Stanford NLTK for the entity recognition stage. The last stage will be our trained entity disambiguation model boosted with knowledge graphs. The final product will be a package that accepts a text document and returns the same document with hyperlink annotations to the corresponding Wikipedia articles. Depending on how Kensho intends to use the pipeline, we are also open to creating a visualization interface, etc. Please let us know if you have any specific suggestions for the final product.