Acadmic Paper Recommender

Lucas laughlin

Introduction

Nobody knows definitively how much academic research is published annually. Rough estimates have put it at about 2,000,000 papers a year, split across 30,000 journals. And this is growing. Both numbers increase by roughly 4% every year. The total number of academic papers in existence was estimated to be 50,000,000 in 2015. The staggering amount of research being produced is unprecedented in human history. Not having the time to read all relevant papers means researchers may miss the most applicable prior research, or even worse, duplicate already published work.

The problem is not one sided though. For every researcher not being able to find the right paper, there exists a paper that nobody finds. It is estimated that 50% of papers are only read by those directly involved in its production. This means the author, advisor and peer reviewers. This can negatively impact the careers of academics who often require high h-index scores for job positions and tenure. The h-index is based on total citation count of ones papers.

So, researchers want to find the most relative research, but also to have their paper distributed to other researchers. In effect, a recommendation system to match readers to papers would hopefully decrease the inefficiency in the current system. Such a system, in a properly distributed application, could do for academic publishing what Netlfix did for television or Spotify did for music. While deliberate searching of papers is fine in targeted spurts, general recommendations on a day to day basis could serve academics well in an effort to stay up to date with development in their field.

Recommendation systems broadly fall into 3 categories: content-based filtering, collaborative filtering, and graph based approaches [1, 2]. Collaborative filtering focuses on previous user-item interactions to try and recommend new items based on what similar users have also interacted with. In such an approach, no feature engineering is necessary. Embeddings for the user and items are learned implicitly solely based off their interactions with eachother. The data can be further augmented by applying strength of interactions such as a 1 through 5 star rating. In simple terms these systems seek to group users together based on their preferences. This is the approach used by Netflix as well as Amazon's shopping recommendations. The main issue with this approach is the cold-start problem. If you have no interactions or very few, then there is a lack of data to derive meaningful insight from. This is applicable in several cases. First, new systems lack any and all interactions meaning there is no way to derive insight from 0 interactions. Second, when new items are added to a system they lack any interaction data to derive an embedding. Third, a new user is added to the system and experiences the same problem.

In contrast, the Content-based filtering approach tries to find items similar to one another based on some notion of distance between features. These can be based off a users given initial preferences or based on thier interactions with items. The difference, is they are not reliant on other peoples interactions, only their own. The benefit of this method is that it avoids cold start. New items can be immediately recommended based off of their features, and new users can get recommendations after a single interaction.

Graph based systems are mainly comprise of 2 steps. A graph construction step and a recommendation step. Graph construction relies on some inherent network found in relation to users or items. In paper recommenders, this could be a citation network, co-author network, etc. Once a graph has been constructed it simply becomes a random walk to find nearby nodes for recommendation.

All of these systems can be combined in different ways to form hybrid systems. For example, graph recommenders can be further augmented by providing some sort of weighting to edges such as cosine similarity between the embeddings of neighboring documents. In this way we are incorporating some form of content based filtering.

The focus of this paper is to consruct good embeddings for documents which can in isolation be used for content based filtering but can also be applied to both collaborative filtering and graph based systems in a hybrid approach. I will be deriving document embeddings from TF-IDF as a baseline and then trying to improve upon them with Doc2Vec, RoBERTa and GPT-2 derived embeddings.

Dataset

The dataset I used was a snapshot of the arXiv dataset found on Kaggle [3, 4]. This is an open source dataset containing 2,057,949 articles in the fields of physics, mathematics, computer science, quantitative biology, quantitative finance, statistics, electrical engineering and systems science, and economics. The dataset is curated by Cornell.

```
Example Document:
({'abstract': '  A fully differential calculation in perturbative quantum chromodynamics is\npresented for
  'authors': "C. Bal\\'azs, E. L. Berger, P. M. Nadolsky, C.-P. Yuan",
  'authors_parsed': [['Balázs', 'C.', ''],
   ['Berger', 'E. L.', ''],
   ['Nadolsky', 'P. M.', ''],
   ['Yuan', 'C. -P.', '']],
  'categories': 'hep-ph',
  'comments': '37 pages, 15 figures; published version',
  'doi': '10.1103/PhysRevD.76.013009',
  'id': '0704.0001',
  'journal-ref': 'Phys.Rev.D76:013009,2007',
  'license': None,
  'report-no': 'ANL-HEP-PR-07-12',
  'submitter': 'Pavel Nadolsky',
  'title': 'Calculation of prompt diphoton production cross sections at Tevatron and\n  LHC energies',
  'update_date': '2008-11-26',
  'versions': [{'created': 'Mon, 2 Apr 2007 19:18:42 GMT', 'version': 'v1'},
   {'created': 'Tue, 24 Jul 2007 20:10:27 GMT', 'version': 'v2'}]},)
```

Each entry contains the metadata described above. Of particular interest to this project, was the abstract, titles, authors, doi, id, and categories.

The abstracts were used to derive document embeddings. The id was used to construct url links to the original arxiv hosted page. The title and authors were used for labeling displayed papers and corroborating results.

The doi was used in conjunction with the Open Citations API [5] to calculate the metrics of efficacy of our derived embeddings.

Around half of all the papers did not contain DOIs. These are all pre prints. While I believe the models would not have trouble processing them, I could not use them with the Open Citation API so they were omitted. Of those I further diminished the sample size to those papers about high energy physics, denoted by "hep-th" in the category column. This resulted in a total dataset of 115,458 papers. I chose this category because it had the most papers out of any other category in the full data set.

Methods

All of the embedding methods performed need a metric for comparison. Normalized number of Co-citaions [6] can be used to derive an objective metric of how well the embeddings were learned. This measures the average number of times the paper was cited along with one of its recommendations. The caveat here is that popular (read highly cited) papers will score higher than others. However, I believe that this is irrelevant as it will drive up the score across embedding types. We can think of this metric as counting the number of recommendation vertices that point toward the same vertex as the paper from which the recommendations are generated in a citation network.

To further augment this metric I will also be using normalized reference and citation scores. These will measure the average number of times that the paper was either referencing or was cited by its recommendations. This can be though of the number of adjacent vertices in a reference or citation network.

Normalized number of co-citations $= nCoP_i = \frac{1}{N}\sum_{j=1}^{N} c_{ij}$, where N = # of Recommendations

Normalized number of references $= nRP_i = \frac{1}{N}\sum_{j=1}^{N} r_{ij}$

Normalized number of citations $= nCiP_i = \frac{1}{N}\sum_{j=1}^{N} (ci)_{ij}$

TF-IDF

I utilized Scikit Learn's TF-IDF vectorizer to get TF-IDF embeddings. I limited the embedding length to 512..

Term Frequency Inverse Document Frequency is defined as:

$tfidf(t, d, D) = tf(t, d) \times idf(t, D)$

Term Frequency is defined as follows:

$$tf(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} = \frac{term\ occurences\ in\ document}{terms\ in\ document}$$

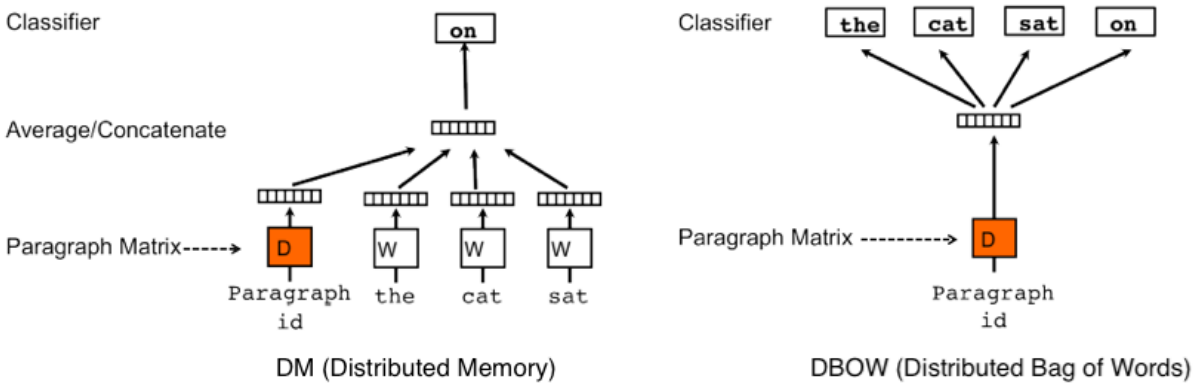
Inverse Document Frequency is defined as:

$$idf(t, D) = \frac{N}{|d \in D : t \in d|} = \frac{number\ of\ documents\ in\ the\ corpus}{number\ of\ documents\ containing\ term}$$

An increase in term occurences in a document increases term frequency which increases TF-IDF. An increase in the number of documents containing the term decreases the inverse term frequency which decreases TF-IDF. In this way it balances the importance of a term to a document with the uniqueness of that term compared with the entire corpus.

Doc2Vec

The Doc2Vec model [8] I used was the one implemented by Gensim [9] which has 2 implementations. The first model known as Distributed Memory modifies the Skip-gram word embedding network to also incorporate a document embedding. The model tries to predict a word based on the contextual words as well as the document embedding. The second implementation modifies the CBOW model and trains by trying to predict the words based off the document embedding.



For this project I chose the DM version. I trained for 10 epochs with a window size of 5 and an output embeddings of length 512. I kept the minimum count at 1.

RoBERTa

RoBERTa[11] is an enhanced version of the language model BERT [12] that increases the total size of the training set from 16 to 160 GB and replaces the next sequence prediction task while implementing dynamic masking so that each epoch the masks are different. I fine-tuned RoBERTa in a similar fashion, masking 15% of tokens and training on the arXiv abstracts for high energy physics.

RoBERTa is designed to be a base laguage model on top of which you would apply a head to perform a downstream task. To get a document embeddings out of the network I passed a batch of a single tokenized abstract into the model. Each hidden layer of the network produces an output of dimensions (b, 512, 768) where b is batch size. So for each abstract I would get a batch size of (1, 512, 768) or 512 feature maps with 768 features each. On earlier layers in the network, these 512 feature maps should more directly represent token embeddings. From there I complete 4 separate calculations on the outputs of the hidden layers to get 4 different document embeddings.

1. Averaging each token embedding to get an embedding of dimensions (1, 512, 1).
2. Averaging the 768 feature maps to get a single feature map of dimensions (1, 1, 768).
3. Concatenating the output of the last 4 hidden layer and then averaging each feature map to get a tensor of dimensions (1, 512, 1).

4. Concatentating the output of the last 4 hidden layers to create a tensor of dimensions (1, 512, 3072). I then applied a 1 dimensional pooling of kernel and stride size 4 before averaging each feature map to get a final embedding of dimensions (1, 1, 768).

All of these were then flattened into a single dimension to act as document embeddings. The method of concatenating the last 4 hidden layers' outputs comes directly from the BERT paper where they found it to be the best method for extracting features.

GPT-2

GPT-2 was fine tuned using a causal modelling task on the same set of abstracts. I kept the batch size at 2 due to the size of Google Colab's GPU memory. To produce document embeddings, I took the output of the first hidden layer and took the mean of all feature maps to get a single document embedding.

Results

Upon sampling 30 papers and then findig their 30 closest neighbors, I got the following scores:

	TF-IDF	Doc2Vec	RoBERTa 1	RoBERTa 2	RoBERTa 3	RoBERTa 4	GPT-2
Co-citations	7.4	3.83	0	1.13	0	0.2	0.63
Citations	0.267	0.2	0	0	0	0	0.33
References	0.233	0.033	0	0.03	0	0	0

TF-IDF is a tough baseline to improve upon. However the language models and Doc2Vec still provide relevant results based on our metrics of co-citations. When looking at the individual co-citation scores I can see that one of papers in the TF-IDF run provided 108 co-citations. If this outlier was removed we are left with a co-citation score of 3.93 which is very similar to the Doc2Vec model.

Of the 4 methods for creating document embeddings from the RoBERTa model, the 2nd one was strongest. This was the one that involved taking the mean tensor of the output of the first hidden layer. It seems that averaging each tensor into a single number (methods 1 and 3) was not a good method for extracting document embeddings. This would make sense in that it is similar to just concatenating a list of token embeddings with the added benefit of attention from the transformers. It is interesting to me that the output of the first hidden layer does better than the output of the last 4 layers concatenated which differs from the findings of the BERT paper. I think these scores could be improved with significantly longer training times.

To take this further into the realm of application, I took the TF-IDF embeddings which were objectively the strongest and used them to build a small test application. I built it using React, Flask and MongoDB and deployed it to the AWS ecosystem as a proof of concept. The link can be found in the github.

Further work

Some effort should be made to improve these embedding models. Possible options would be to train the language models (GPT-2 and RoBERTa) on all 2,000,000 papers for a larger amount of time or changing

the operations performed on the hidden layer output toproduce document embeddings. These may be fine on their own or we could then fine tune them again for a specific category. This would be similar to Don't Stop Pretraining [15].

Another possible future approach would be to train category vectors in the same way we trained document vectors using Doc2Vec. This could be an interesting way to try and promote collaboration amongst researchers in different fields of study.

References

[1] Dhoha Almazro, Ghadeer Shahatah, Lamia Albdulkarim, Mona Kherees, Romy Martinez, andWilliam Nzoukou. A survey paper on recommender systems. ArXiv, abs/1006.5278, 2010.

[2] Xiaomei Bai, Mengyang Wang, I. Lee, Z. Yang, Xiangjie Kong, and Feng Xia. Scientific paper recommendation: A survey. IEEE Access, 7:9324–9339, 2019.

[3] https://www.kaggle.com/Cornell-University/arxiv.

[4] https://arxiv.org/

[5] https://opencitations.net/api/v1

[6] Milind Shyani. An ArXiv Paper Recommender.

[7] https://scikit-learn.org/stable/index.html

[8] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. ArXiv, abs/1405.4053, 2014

[9] https://radimrehurek.com/gensim/

[10] Tomas Mikolov, Kai Chen, G. S. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In ICLR, 2013.

[11] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer and Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv:1907.11692, 2019

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805, 2018

[13] https://huggingface.co/

[14] Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, Noah A. Smith, Don't Stop Pretraining: Adapt Language Models to Domains and Tasks. arXiv:2004.10964, 2020