CS224N: Patent Citation Prediction with Content and Network Features

John Knowles
Symbolic Systems
Stanford University
Stanford, CA 94305
`jkn0wles@stanford.edu`

Sam Premutico
Computer Science
Stanford University
Stanford, CA 94305
`samprem@stanford.edu`

Abstract

In our project, we offer citation recommendations for newly submitted and existing patents in the US Patent Office. In doing do, we attempt to automate a portion of the job of a patent examiner, who upon receiving a patent application adds additional citations beyond those included by the applicant. We frame the task as both an NLP task and network analysis task, since we have both textual features in the patent applications themselves as well as network features in the existing citation network. We build off of the model proposed in *GraphNet: Recommendation system based on language and network structure* by Ying et al.[4], which provides content recommendations by analyzing both the content of a document and the network structure of the document. We modify the *GraphNet* approach by leveraging textual and network features in distinct phases, rather than using both in a single supervised prediction task. We extend the *GraphNet* NLP feature extraction work done by training an autoencoder whose encoder's state is used as the document representation, rather than mean GloVe vector. Further, in order to overcome performance issues that arise from a large search space, we implement a clustering phase in which we cluster document representations in order to filter candidate documents prior to recommendation before feeding the candidate patents through a prediction network trained on network features to output the final recommendations. We find that our complete cluster-then-predict model outperforms both baseline models and our cluster-only model.

1 Introduction

1.1 Motivation

When a patent application is submitted to the United States Patent Office, the author of the patent includes in the patent application a series of citations which point to prior art (patents) whose work is relevant to the patent at hand. The patent examiner then, in reviewing the patent, adds additional citations which she believes point to additional relevant prior art not already included in the original application. In their paper *Do Applicant Patent Citations Matter?* [1], Stanford Law Professor Mark Lemley and Columbia University Associate Professor Bhaven Sampat study the effect of prior art listed in applications. The authors found that "patent examiners rarely use applicant-submitted art in their rejections to narrow patents, relying almost exclusively on prior art they find themselves. However, this is not because the prior art pointed to is irrelevant, but rather becauase examiners exhibit a sense of "myopia". That is, examiners believe the work they identify is most relevant in the review process.

Since the additional prior art surfaced by examiners largely dictates the fate of a patent application, we aim to provide partly automate the process by surfacing the most relevant patents for an examiner to consider when reviewing a single application. We frame this as a link-prediction task in which we predict the existence of a link (a citation) between a source patent and target patent. To do so, we make use of both network features (existing links) and natural language features (patent document text).

1.2 Task

We formalize our task as follows: given a patent a and a subset C_a of the set of patents S_a that a cites, we attempt to predict the unseen patents which a cites, namely $S_a - C_a$. We visualize the task below, where we predict citations for a, with $S_a = \{b, c, d, e, f\}$ and $C_a = \{c, e, f\}$. This means that to train and test our model, we must hold out a set of edges $S_a - C_a$ which we attempt to predict. In this case $S_a - C_a = \{b, d\}$. We label the patents in the network which a does *not* cite $F_a = \{g, h, i\}$.

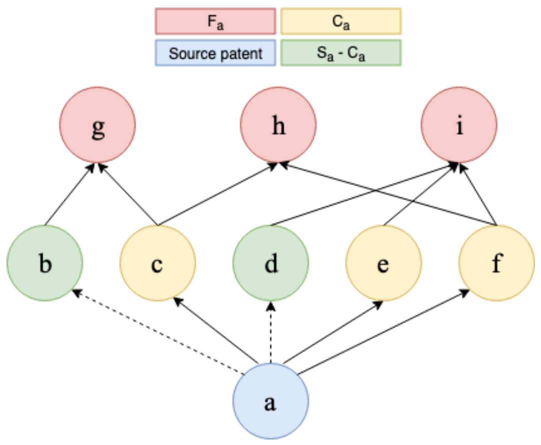

Figure 1: Link-prediction task where we predict held-out citations $\{b, d\}$ for a, given citations $\{c, d, f\}$

1.3 Approach

Our model is inspired by the *GraphNet* model proposed by Ying et al. which uses network features and NLP features to predict which posts a Reddit user will engage with. The model extracts NLP features of a user and post by averaging GloVe vectors over the post and user, respectively. The model extracts network features by sampling neighbors of a user and post and extracting and aggregating their NLP features. Finally, *GraphNet* feeds the concatonated NLP and network features through a prediction layer to predict which posts a user interacts with (i.e. comments on). Negative sampling loss is used in the prediction layer, with mean reciprocal rank (MRR) as the authors's final evaluation metric.

Once we have generated NLP features in the first phase of the algorithm, we attempt to separate the use of NLP features and network features into two distinct phases. Additionally, we generate richer document representations than *GraphNet* by using an autencoder rather than averaging GloVe vectors. Once we have generated representations for a each patent, we perform an initial clustering phase in order to reduce the output space and generate candidate recommendations. In our clustering phase, we identify the top K patents whose representations are closest to that of the patent for which we are generating recommendations. Next, in our third phase, we take the K candidate patents filtered through in the clustering phase and pass them through a prediction network, trained on a link-prediction task using existing citations as features. Finally, we recommend these K candidate patents in order of the predicted likelihood of an edge existing between the source and candidate patent.

2 Related Work

Previous work with the dataset was helpful in initial exploration and for problem definition. Work by Cui, Wang and Zhai [2] on a graph-based approach to link prediction on the US Patent network provided an overview to graph-based techniques. The team of Leskovec, Kleinberg, and Faloutsos [3] study the US Patent network using sophisticated, graph-based methods. Work by Ying, Li and Li [4] on a recommendation system for reddit content, utilizing GloVe vectors for langauge features, proved invaluable in structuring our recommendation architecture. There exists a large body of literature on the topic of learning representations from documents. Hinton and Salakhutdinov's [5] work laid the foundation for understanding and motivating the use of autoencoder networks as bottleneck structures for learning fixed length representations in contexts of variable length input. Autoencoders utilize a bottleneck architecture, in which an input sequence is "squeezed" into a fixed sized representation, then expanded back to its original size. Further work around document representations takes several forms. The team of Le and Mikolov [6] used an unsupervised setting, training vector representations for in-document prediction tasks. Le and Dai [7] use a semi-supervised approach with unlabeled sequence data, utilizing a LSTM autoencoder network architecture to achieve strong results across a number of classification tasks. Kaiser and Bengio [8] expand and augment on the traditional autoencoder structure, introducing semantic hashing and other improvements. Finally, work by Sutskever, Vinyals and Le [8] on a Seq2Seq network provided the basis for understanding and development of our autoencoder network.

3 Approach

As previously mentioned, we implement a three-phase algorithm, visualized below in Figure 2. We first generate document representations using an autoencoder trained on our patent corpus. Second, we cluster and filter candidate patents by cosine distance to the source patent representation. Third, we predict the existence of a link between the source patent and each candidate patent using existing citations as network features, outputting our recommendations in order of our model's confidence in the existence of an edge. We now separately describe each phase in detail below.

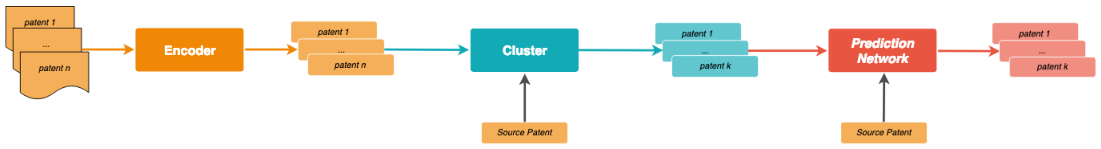

Figure 2: Our three-phase model. First, patent documents are encoded via the encoder of an Autoencoder trained on our patent corpus. Second, these 256-dimension encoder representations are filtered by cosine distance to the encoding of the source patent whose citations are being predicted to identify the top K candidate patents. Third, these K candidate patents are passed into the prediction network which predicts the existence of an edge between the source node and each candidate node and reorders the candidate patents accordingly.

3.1 Baselines

We implement several baselines models, utilizing different document vector representations.

1. **TF-IDF Baseline**
   First, we implemented a TF-IDF algorithm which extracts the TF-IDF scores for each word in each document and then predicts patents in order of their TF-IDF vector cosine-similarity distance. We used clustering methods with a cosine-similarity distance measure, returning the top K documents based on distance.
2. **GloVe NN Baselines**
   We implemented several text-only approaches utilizing GloVe representations as document encodings. We generate our own GloVe embeddings trained over our patent corpus in order to better capture word meaning the the patent-domain. We trained using a learning rate of .05 over 50 epochs to generate 256-dimensional embeddings. As patents use highly technical

and legal language, we thought it best to train our own embeddings. We used average, element-max and element-min over the GloVe vectors for each word in our documents. We used clustering methods with a cosine-similarity distance measure, returning the top K documents based on distance.

3.2 Network Architecture

Patent citation presents two main issues. First, the high average document size of a patent means that a system is attempting to extract features from large bodies of text, and the second, the size of the network increases predictive difficulty. Our method employed a variant on Seq2Seq, an encoder-decoder network, which makes use of the autoencoder structure to produce vectorized representations of documents, using the encoder side of the network. Our autoencoder network uses a recurrent structure, utilizing RNN cells to compute hidden states over all inputs. The task of recommending citations from this representation uses both unsupervised and supervised learning, utilizing clustering methods for output space reduction and a supervised prediction network. Each candidate pair of patent and potential citation from the document cluster is fed to a predictive network and given a link prediction score. We return recommended documents according to their sorted score.

3.2.1 Autoencoder Network

Our autoencoder network consists of two parts, an $encoder$ and $decoder$. We utilize an RNN cell to read over the input sequence, one word at a time, to obtain a fixed dimensional vector representation. This $encoder$ part of our network is responsible for encoding the input sequence into a 256 dimensional vectorized form. The $decoder$ side of our network also utilized an RNN cell, acting as a language model attempting to reconstruct the input sequence. LSTM cells capture the optimal representation of long range dependencies, however, due to large input sequences, we settled on an RNN cell to balance capturing feature information and training time. GRU, RNN and LSTM cells were tested. However, as will be addressed in our conclusion, RNN cells were chosen as a training time consideration. The update equation for RNN cells is presented below:

$$h_t = Wf(h_{t-1}) + W^{(h_x)}x_t$$
$$\hat{y} = W^{(S)}f(h_t)$$

3.2.2 Clustering

We take the definition of a document cluster to be the top K documents based on the defined distance metric, cosine-similarity. We see that we are using a distance measure to attain clusters, which is an inherently bidirectional task. Our data, however, has unidirectional links. thus, in order to constrain our predictions, we only take citations from patents filed before the predicted patent. We calculate the cosine-similarity over all documents, and take a cluster of the the top K candidates. The formula for cosine-similarity is presented below:

$$\textit{cos-sim}(x, y) = \frac{x \cdot y}{||x|| \cdot ||y||}$$

3.2.3 Prediction Network

Our prediction network takes in a candidate pair, represented as 256 dimensional vectors, and runs convolutions over both inputs before concatenating their representations. This concatenation is convolved and then fed through a linear layer and a sigmoid output layer. We use Binary Cross-Entropy loss. We train this network on the task of link prediction based on a candidate pair, and output a binary classification on the presence of a link. We generate candidate pairs representations for this network using the $encoder$ described earlier.

3.3 Implementation References

To gather our data, we wrote our own scraper to scrape patent data from the USPTO (United States Patent and Trademark Office), and store it in a local database. We referenced a PyTorch implementation of the Seq2Seq network for our model, editing it according to our specific task. For our baselines, we trained a Python implementation of GloVe. Other code, such as our MRR metric, clustering techniques, and prediction network, was self-implemented, but relied in part on assignment code.

4 Experiments

4.1 Data

The dataset that we use is sourced from Google Patent data. We scraped .txt files on weekly US Patent acceptances from 1976 - 2001, aggregating text content from Abstract, Description and Figure sections, and link content from listed citations. This dataset is then stored as a collection of numpy dictionaries stored locally, queryable by patent id for training. Our dataset consists of a total of roughly 3 million patents. We only used a subset of the data of approximately 85,000 patents in order to speed up experimentation and ensure that the target nodes were all present. The average out-degree of a given node in our subset is 14, meaning that the average node cites 14 patents. This data is then split into a 90/5/5 training/test/dev sets, which was held constant across multiple training settings.

4.2 Evaluation method

The evaluation metric we use is mean reciprocal rank (MRR), because our task is one of recommending a list of citations. In patent citations, labels consist of the patent and corresponding citations. We take the MRR score over labeled links, testing the quality of the recommended citations a given patent was assigned. MRR over |Q| queries is calculated as:

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i}$$

where $rank_i$ is the position of the first ground truth patent in the list of recommended patents. If no true link appears in the recommendations, $\frac{1}{rank_i} = 0$. Note that patent citations are directed edges, and we consider only those of the relevant direction as true citations.

4.3 Experimental details

Given our link prediction task, to prepare our data for training we need to remove a random subset of the links in the training data. For our training, we tried several rates of removal, at 10, 20, and 30 percent link removal, using a rate of 20 percent as our final training split. This removed link content was used as the target output. In order to impose a training/testing split, we used data from different time periods. This ensured that the patent content would be unseen. Due to space limitations, we trained over a month of patent data from July 1999, and used the first week of August 1999 as a test set.

Note that we could not simply take all of the patents from a single month as a training set, since the patents that are cited by a single patent are almost never processed in the same month as the citing patent. This meant that in addition to taking all of the patents within a time frame to generate a split, we have to traverse the edges of those in the time-window and add their neighbors to the set. This meant that the labeled nodes which we try to predict are actually present in the split. Our training set ultimately had 75,000 patents and our evaluation and test sets had 5,000 each. In order to constrain the size of our text content, document size was limited to the final 1000 words. The last section of patent content includes general and figure descriptions, which provide suitable document summaries. Experiments with smaller document sizes yielded poor clustering results.

Our final The network was trained for 10 hours on a NVIDIA Tesla M60 GPU using Microsoft Azure. For loss, our current implementation uses the PyTorch implementation of Cross Entropy loss with Logits and for our optimization, we used the SGD optimizer with learning rates .005 with a decay of 1e^{-6}.

4.4 Results

Table 1 reports results on our evaluation set of 5,000 patents using values of $K = 50, 100, 300$, and 500. *TF-IDF Baseline* and *GloVe Baseline* correspond to the baselines outlines above. *Cluster Distance* uses only the output of the clustering phase without the prediction network. *Cluster + Predict* uses the entire network outlined above.

Table 1: **MRR Results Across K**

Model	**K=50**	**K=100**	**k=300**	**k=500**
TF-IDF Baseline	0.003	0.011	0.008	0.014
GloVe Baseline	0.014	0.016	0.016	0.018
Cluster Distance	**0.172**	**0.172**	0.174	0.175
Cluster + Predict	0.164	0.169	**0.185**	**0.185**

Table 2: Experiment results.

We provide analysis of results below.

5 Analysis

We see that TF-IDF vector cosine distance performs quite poorly. Note that MRR reports 0 for instances where there is no true patent citation among those returned, which explains why we see improvements as K increases. Thus the worst-performing models, namely TF-IDF, experiences solid improvement as K increases because we have fewer instances in which no true citation is present in the output predictions. Looking at TF-IDF, we see that the majority of outputs do not contain a true citation in the top 50 returned, explaining it's large increase across experiments.

We see that Cluster Distance initially slightly outperforms the prediction model, meaning that the initial ordering of documents by cosine distance is more accurate than the confidence that the prediction network has in the existence of an edge. *However*, as we increase K the full prediction network actually begins to outperform the clustering model. Intuitively, this means that the prediction network is able to identify true citations which previously were not in the, say, top 100 returned results based on clustering but were in the top 300 and then accurately predict a link to it with greater confidence than any of those previously in the top 100. As we do not see this phenomenon occur further at $K = 500$, this perhaps means that patents at a distance of 100-300 often contain true citations that the prediction network can identify most confidently.

Qualitatively, we found that patents that relied on more frequently used terms performed worse in the recommendation task. We take this to be a standard result, as more unique documents had more unique representations. Given our clustering task, we returned higher quality across top documents, but lower overall quality clusters over these more unique documents, as the clustering algorithm struggled to find similar documents. Given the standardized nature of patent submissions, we have a set of commonly used terms throughout all patents. For example, words like 'application', 'continuation' and 'disclosure' are surprisingly common across patents. Given the presence of these terms and the fact that patents follow a standard, we find it harder to differentiate among patents using standard forms. In future work, we could leverage this fact by clustering $> K$ candidate

Additionally, in an analysis of the patent recommendations with lower MRR, we saw that a large number of these seed patents had a relatively low degree. In contrast, the patents with large numbers of citations performed better, especially at high values of K. This intuitively makes sense, as a patent that cites 30 other patents has more true citations to surface than does a patent which cites 15. Future experimentation using an evaluation metric which is not biased towards higher-degree patents, unlike MRR, might be worth pursuing.

6 Conclusion

In conclusion, this project has shown the ability of a deep learning system to find optimal representations of text documents in vector space. Several techniques were explored in order to accommodate the large search space. Through coupling of multiple architectures, an autoencoder network, clustering algorithm and a predictive network, we were able to take advantage of optimizing over multiple contexts, as improvements to one system yielded improvement across the entire architecture. However, the length of our documents and the size of our network proved to be a formidable hurdle to success in our project. Looking forward, there is much room to improve on our current architecture. In our autoencoder network, explorations around network architecture, including the use of an LSTM cell to improve long range information capture, and increasing the hidden layer size of the cell, may result in more optimal document representations. our network chose a representation size of 256, but experimentation around different sized representations may result greater ability to encode document information. Additionally, with an established training/testing pipeline, an attention model could help focus the representations on the most important parts of the input, though refinements to the attention model would have be made in order to optimize speed of training.

7 Additional information

Custom Project TA: Amita Kamath

References

[1] C. Cotropia, M. Lemley, B. Sampat. Do Applicant Patent Citations Matter? 2013.
[2] J. Cui, F. Wang, J. Zhai. Citation Networks as a Multi-layer Graph: Link Prediction and Importance Ranking. 2010. .
[3] J. Leskovec, J. Kleinberg, C. Faloutsos. Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations. 2005.
[4] R. Ying, Y. Li, X Li. GraphNet: Recommendation system based on language and network structure. 2017.
[5] G.Hinton, R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. 2006.
[6] Q. Le, T. Mikolov. Distributed Representations of Sentences and Documents. 2014.
[7] A. Dai, Q. Le. Semi-supervised Sequence Learning. 2015. arXiv:1511.01432
[8] L. Kaiser, S. Bengio. Discrete Autoencoders for Sequence Models. 2018. arXiv:1801.09797
[9] I. Sutskever, O. Vinyals, Q. Le. Sequence to Sequence Learning with Neural Networks. 2014.