

Retrieve Instruction from Manual for Strategy Games

Alison Paredes, Tianyi Gu

Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA

Abstract

In this project, we study the task of helping an automated player win a computer game by reading a strategic user’s guide designed for human players. In complex computer games such as Star Craft, War Craft, and Civilization, finding a winning strategy is challenging even for humans. Therefore, human players typically rely on manuals and guides. Recently, researchers have tried to using such textual information to train an automated player (Branavan, Silver, and Barzilay 2011). Our goal, is to better understand the retrieval model used in their paper in terms of algorithms and metrics from the field of information retrieval. Our results provide some evidence that inverse document frequency out performs recurrent neural networks at assisting human players, and could be used as a baseline for evaluating retrieval models used in playing games like this.

Introduction

In this project, we evaluated ways that relevant text could be retrieved from a strategic user’s guide to aid in automated planning in a turn-based strategy game. In complex computer games such as Star Craft, War Craft, and Civilization, finding a winning strategy is challenging even for humans, therefore, human players may take advantage of strategic manuals to play a winning game. Automated planning in games like these is also challenging because of the number of actions available at each turn. In game such as the one discussed in this paper there can be as many as 10^{21} possible actions at each turn. Recently, some researchers (Branavan, Silver, and Barzilay 2011) have proposed to use a strategic user’s guide designed for human readers to limit automated search for promising actions.

In this paper we compare two retrieval models, a vector space model and a generative language model, and we evaluate them in terms of their ability to satisfy an information need defined by human experts. First, we give an overview of Monte-Carlo Tree Search, a critical component of Branavan, et. al. to identify relevant text automatically.

(Branavan, Silver, and Barzilay 2011) approach this task in a online supervised framework based on a feedback signal from the tree leaves. Because at the beginning steps, the tree search will be guided by an un-trained retrieval model, which certainly could waste the computational resource, we propose two approaches to learn the retrieval model offline. Our

The natural resources available where a population settles affects its ability to produce food and goods. Build your city on a plains or grassland square with a river running through it if possible.

Figure 1: An example sentence from the user manual of the game Civilization II

first approach is to use vector space model as the retrieval model, where we treat each sentence in the manual as a document. We compute the TF-IDF value for all documents of fine. The game state could be described as a query sentence. Then in online tree search, we compute the similarity between the query and all sentences and rank them with the hope that the most relevant sentence is at the top of the ranking. Our second approach is to train a language model offline by recurrent neural network, which would contain all the information of the user manual. Then in online tree search, we use the language model as a generative model, sample from it to generate a sentence to bias the search, using the query as seed words. We evaluate our method using mean average precision (MAP@K=10) and Normalized Discounted Cumulative Gain (NDCG@K=10). Then we show our results could be used as an alternative retrieval model in the tree search framework.

Related Work

Training a Robot to Read

Our work is based on previous research in machine learning (Branavan, Silver, and Barzilay 2011). As they do, our method also operates within the Monte-Carlo tree search framework (Tesauro and Galperin 1996), which has been successfully applied to complex games such as Go (Silver et al. 2016) and Klondike Solitaire (Bjarnason, Fern, and Tadepalli 2009). The game is defined by a large Markov Decision Process $\langle S, A, T, R \rangle$. Here S is the set of possible states which encodes attributes of the games such as players’ territories and military resources. A is the space of legal actions such as arrange some works to get natural resources. $T(s'|s, a)$ is a stochastic state transition function where $s, s' \in S$ and $a \in A$. Finally, a given utility function $R(s) \in R$ captures the likelihood of winning the game from

state s .

The goal of the Monte-Carlo search algorithm is to dynamically select the best action for the current state s_t . This select is based on the results of multiple *roll-outs* which measure the outcome of a sequence of actions in a simulated game. States and actions are evaluated by an *action-value function* $Q(s, a)$, which is an estimate of the expected outcome of action a in state s . As the accuracy of $Q(s, a)$ improves, the quality of action selection improves and vice versa, in a cycle of continual improvements (Sutton and Barto 1998).

In their work (Branavan, Silver, and Barzilay 2011), the authors propose a method for learning an action-value function augmented with linguistic features, while simultaneously modeling sentence relevance and predicate structure. They employ a multi-layer neural network where the hidden layers represent sentence relevance and predicate parsing decisions. The hidden layer encodes the action-value function $Q(s, a, d)$, which also depends on the document d . Their relevance decision is modeled as a log-linear distribution over sentences as follows:

$$P(y_i | s_t, a_t, d) \propto e^{\vec{\mu} \Phi(y_i, s_t, a_t, d)}$$

Here $\Phi(y_i, s_t, a_t, d) \in R^n$ is a feature function and $\vec{\mu}$ are the parameters need to estimate online.

Probabilistic models are one of the most important classes of retrieval models in information retrieval making its use in this case not unusual. One of the most popular probabilistic models is the binary independence model which is like the probabilistic model proposed in their paper because not only does it rank documents, in our case sentences, by probability of its relevance given a vector of features of both the query and the sentence after some training but it can also be used untrained, without any judgments as to the relevance of a sentence given a sentence and the query. Since the model is trained as the game is played it may be initially untrained, vector u will be 0, therefore the probability that any sentence will be the most relevant sentence to the current state will be the same across all sentences. At the start of the game any sentence may look like the most relevant sentence. But unlike the authors probabilistic model where all sentences are equally likely to be relevant before the model has been trained, in binary independence model (BIM) some sentences will be more likely than others to be relevant given the query. It is this baseline case that we would like to use to compare any other retrieval model used in game playing because it is a common model, not only is it used in probabilistic models but also vector space models of retrieval.

Vector Space Model

We begin by showing how the vector space model we have chose is related to the probabilistic model noted above. BIM weighs the ratio that a term used in the query appears in a relevant document against the frequency that it appears in a non-relevant document. The idea is that a query term will appear more frequently in relevant documents than non-relevant documents.

$$\sum_{t: x_t=1, q_t=1} \log \frac{p_t(1-u_t)}{u_t(1-p_t)}$$

Of course this requires knowing something about which documents are relevant to the query or not. Instead, if we assume that it is equally likely that a term appears in a relevant document as not,

$$p_t = \frac{1}{2}$$

and that the proportion of times it does not show up in a non relevant document is influenced by the number of sentences it shows up in across the entire corpus,

$$u_t = \frac{n_t}{N} = \frac{df_t}{N}$$

then we we can say that without any training terms that show up rarely in other sentences throughout the user's guide but do show up in the sentence in question make this sentence more likely to be the most relevant sentence than other sentences.

$$\sum_{t: x_t=1, q_t=1} \log \frac{N - df_t}{df_t}$$

This weight is the similar to a weight used in vector space retrieval models, inverse document frequency (IDF) (Christopher D. Manning and Schtze 2008).

$$\log \frac{N}{df_t}$$

This will serve as the baseline for comparison for other retrieval models discussed in this paper given its popularity and its simplicity. While it is better than a uniform distribution over relevant sentences it is nothing fancy. It weights some terms more than others in a way that models the frequency of words in language using limited computational effort. If we can assume that we can read the entire user's guide before it is ever queried by the player then we can score each term in our user's guide's vocabulary in a way that can be leveraged at query time in constant time, giving us a rough but quick score of relevant documents which we can use to compare to more sophisticated methods.

Language Model

Our another approach is to use language model to predict a sentence that relevant to the query. The Recurrent Neural Network (RNN) is neural sequence model that achieves state of the art performance on important tasks that include language modeling (Mikolov et al. 2010), speech recognition (Graves 2013), and machine translation (Kalchbrenner and Blunsom 2013). The RNNs address the issue that it is unclear how a traditional neural network could use its reasoning about previous words in the documents to inform later ones. Essential to these success is the use of Long Short Term Memory networks (LSTMs), a very special kind of recurrent neural network, for many tasks, much better than the standard version. They were introduced by (Hochreiter

and Schmidhuber 1997), and were refined and popularized by many people in following work.

One of the appeals of RNNs is the idea that they are able to connect previous information to the present task, such as using previous paragraph inform the understanding of the present sentence.

Approach

Vector Space Model

One simplifying assumption we made was to forego any language processing in constructing the vocabulary. We did not remove any stop words such as "a" or "the" either in the vocabulary or in our query construction. Instead we relied on the model to emphasize ubiquitous and therefore less valuable words like "a" and "the" in distinguishing once sentence from another. In a small corpus such as ours, this assumption may not have been as helpful as we would have liked since the corpus is so small even words like "the" which we would expect to occur in every document if they were much larger did not, and while they were weighted low as expected, their scores were close to zero, other less valuable words that occurred in our query such as "how" might have had a bigger impact then we would have liked. Nonetheless, as our results show, IDF still out performed RNN despite the influence of stop words.

$$tf = 1 \\ (1 + \log(tf))(\log \frac{N}{df_t}) = \log \frac{N}{df_t}$$

We also opted not to normalize scores by the length of the sentence in which they occurred as is more commonly practiced in using the vector space model to retrieve documents from a large corpus. We made this decision when we initially ran a normalized version of this scoring function against the user's guide and discovered a characteristic of the user's guide that would be expected of a single document but not necessarily of a corpus of documents. Since our user's guide is a collection of sentences and each sentence, to be consistent with the task defined by Brannavan et. al, is treated as a single document. We found that some sentences within the user's guide act more as subtitles than sentences. They are consequently much shorter than the average sentence and therefore will score higher than the average sentence when they contain the same set of terms. Because subtitles however are not structured the same way as a sentence with an object and a predicate, predicate being the more interesting piece of the sentence for our longer term objective, to test the ability of information from the user's guide to influence action selection and since actions can be found in the predicate of a sentence, we opted to simplify the scoring model. This introduced a new assumption about which sentences would be more useful to a player, longer more complex sentences than shorter collections of words.

Finally, we opted to weight query terms only by their existence, using a binary model. We assumed all query terms would be equally weighted and we let the occurrence of the query term in a sentence drive the value of that term. The

score of a particular query and sentence was therefore the sum of TF-IDF scores for each term in the query.

Another reason why we were excited to be able to compare Branavan et. al's probabilistic model to a vector space model was because it enables us to query the user's guide in constant time. Ideally we would have liked to be able to compare the performance of both our baseline and our generative retrieval model using a different source of feedback other than our human annotators. We believe it is a good baseline to measure retrieval models using judgments provided by human players, given that the game is designed for human players and the user's guide is designed for human readers. We are implicitly assuming that encapsulated in our annotator judgments is their assessment that the sentence is relevant to playing a winning game. The authors Branavan et al propose another way to evaluate the relevance of a sentence. They propose to use its performance in a simulated game as a proxy. This could be considered a kind of relevance feedback which they use to adjust their retrieval model. For our purposes we would have liked to use the result of a simulated game as a proxy simply for determining relevance. This would have however required simulating a game which in order to be feasible would have required our implementation of querying the user's guide to be fast, constant time ideally.

Consequently our system indexes the user's guide offline, at the time calculating term frequency (TF) for each term-document pair in the user's guide's vocabulary. The time to construct this index is O in the number of terms in the user's guide. We expect to read every word in the corpus and either add the word and sentence to our index and increment the total number of sentences in the user's guide, add the sentence to our index and increment document frequency, or increment the count of the word in our index to support TF. We defer calculating TF-IDF until query time rather than revisit every term-sentence pair to calculate and store TF-IDF. At query time all we have to do is look up the number of sentences we read and the term where we can access the term's document frequency and its posting list, where for each posting we can access the term-frequency in the sentence, and at that time calculate TF-IDF. Because calculating TF-IDF for a single term-sentence pair involves directly accessing these stored values we can consider calculating TF-IDF takes constant time at query time. Ideally we can characterize the calculation of TF-IDF for all the terms in a query as constant time if we assume that the query size is considerably smaller than the size of our vocabulary. For the queries we tested in the project this is true, however it is possible that the number of terms used in a query initiated by an automated player may have considerable more terms. But for now we can say queries are executed in constant time.

The size of our implementation is also O in the size of our vocabulary and the number of sentences in the user's guide. Given that we expect some words to occur in almost every document and many more words to occur in very few documents, we estimate the size of implementation to be O in the size of the vocabulary.

Language Model

Language modeling is a task of learning a probability distribution $P(w_1, \dots, w_n)$ over a set of all possible word sequences. The goal is to learn such probability distribution P that real sentences will have much higher probability compared to random sets of words. Once such probability distribution is learned, we can use it as a generative model and sampling from it to generate new text. In this section, we will talk about a language model approach that predict a sentence given several query words as sampling seed.

To predict a sentence, sometimes, we only need to look at recent information to perform the present task. For example, consider a language model trying to predict the next word based on the previous ones. If we trying to predict the last word in “the clouds are in the sky”, we don’t need any further context. In such cases, where the gap between the relevant information and the place that it’s needed is small. There are also cases where we need more context. Consider trying to predict the last word in the text “I grew up in France...I speak fluent *French*”. Recent information suggests that the next word is probably the name of a language, but if we want to predict which language, we need the context of France, from a sentence might be several paragraph back. It’s entirely possible for the gap between the relevant information and the point where it is needed to become very large. Recurrent neural network is a neural network with a recurrent connection. That is, unlike conventional network, it considers it’s previous state in addition to the current input. Because of this modification RNNs are natural model of choice for modeling sequential data.

So we have an idea about what RNNs are, why they are super exciting. We’ll now ground this in our task: We’ll train RNN language models. An obvious way to achieve this is to train a word based language model. However, sampling with query words which are not exist in the manual is likely to be a problem. This problem can be solved by train character based language model. That is, we’ll give the RNN the huge chunk of text (the manual) and ask it to model the probability distribution of the next character in the sequence given a sequence of previous characters. This will then allow us to generate new sentence one character at a time.

```
Distinct terms: 32
Seed: autoworker
Sample: !fs?jzso ?llsjs<eos>r.o,mh<eos>vvtghl a,e,byopukbhfh cacal zbs

Epoch: 1 Learning rate: 0.050
Epoch: 1 Train Perplexity: 17.205
Epoch: 1 Valid Perplexity: 8.915
Seed: autoworker
Sample: dy the eissoiv afef the osich to cealv biove theuml fae

Epoch: 19 Learning rate: 0.029
Epoch: 19 Train Perplexity: 3.492
Epoch: 19 Valid Perplexity: 3.007
Seed: autoworker
Sample: is its support , module detail listed .<eos>if however ,

Epoch: 38 Learning rate: 0.002
Epoch: 38 Train Perplexity: 2.928
Epoch: 38 Valid Perplexity: 2.527
Seed: autoworker
Sample: s are ogner pollution , you have ship pollution short l
```

Figure 2: Example sentences generated by RNN

Here, we just follow Tensorflow’s tutorial. Instead of us-

ing the data set from the original paper (Zaremba, Sutskever, and Vinyals 2014), we use the user manual of the game Civilization II. My seed phrase is: “autoworker” which is found in the manual. As shown in figure 2, before any training, we get a totally random sequence of characters. As expected, this sentence doesn’t make sense, meaning that our (currently random) probability distribution over word sequences isn’t particularly useful. Amazingly, just after the first epoch, the model seems to have learned sentences are split by space. After 19 epoch the model have learned to to spell English words. While the phrase itself does not make sense, the word in it are undoubtedly English with a typo or two. After epoch 38, we get a sentence that try to summarize the mean of “autoworker”. As shown in figure 3, after epoch 82, the perplexity of both train set and test set converge to about 2.5.

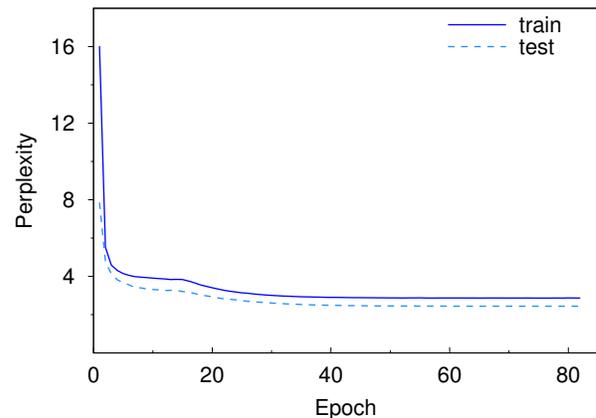


Figure 3: Perplexity of our prediction method

Evaluation

Data set

The data set we used for this project was the same data set used in Branavan, et al. We were able to get a copy of it from the authors. The file, civ_II_game_manual.txt was a text file which differed somewhat what one would have expected a player would have seen. This version appeared to have been processed although minimally. We did not need to distinguish sentences from each other. Most importantly the authors had organized the user’s guide to put each sentence on its own line in the file, delimited by end of line characters. We were thus able to read the file line by line and consider each new line a new sentence. Sentence IDs therefore corresponded to line numbers in the file making it easy to check our parser.

Our decision to interpret each sentence in the user’s guide as a single document was motivated by the retrieval model used in Branavan et al. to retrieve the single most relevant sentence from the user’s guide to evaluate in simulation. Secondly, this interpretation enabled us to compare Branavan et al’s retrieval model to well established information retrieval models. There is a significant difference however between the size of a sentence and the size of a document. Our

benchmark retrieval model is designed to be used in traditional information retrieval settings where the unit of information is the document, a body of text made up of a number of sentences. A corpus should be made up of a large number of documents. In our situation we have significantly reduced the size of both the unit of information and the corpus. On average the size of our unit of information, the sentence, is only 16 words long and the size of our corpus is only 2,084 sentences. Our vocabulary is considerably smaller than a traditional vocabulary might be as well. Instead of millions of terms we have only 3552 terms, including stop words such as "the". The distribution of our vocabulary however is shaped similarly to what we would expect of a traditional corpus. A few words have an IDF close to zero indicating that they occur frequently in many sentences in the corpus while the majority of words are close to the highest IDF indicating that they occur rarely. Nonetheless a sentence may still be too small a unit of information to benefit fully from IDF's ability to minimize the influence of common words. For example the word "the" should be ubiquitous but it appears in only 1426 sentences of our 2084 sentence corpus, resulting in an IDF of 0.4. With the highest IDFs at about 3.4, 0.4 is still about 12% of the weight of a unique term. In a long query such as the queries we used our experiment, a sentence made up of a handful of infrequent words could compete with one made of much rarer words and return a poor result set.

Measures

This project is in part a response to Branavan et al which took an approach to evaluating their retrieval model used in machine learning, evaluating the result returned by their retrieval model for accuracy. They took the single most relevant sentence returned by their retrieval model and determined if it was accurate or not. They were able to distinguish an accurate result from an inaccurate result by adding sentences they considered inaccurate into the user's guide. They argued that because of the large variety of states a player could be in when it queries the user's guide that it would be impossible to manually annotate enough sentence-state pairs to adequately evaluate their model using other methods. We have however attempted this alternative method of evaluating a retrieval model with statistically significant results.

In our method we used a tiny subset of the large number of possible information needs a player could have had at the outset of a new game. Like Branavan et al we focused our experiment on the beginning phase of the game which they propose is better addressed by the user's guide. This information needs was then represented as a query for each system. We then queried each system returning a list of sentences from each ranked in order of most relevant to least relevant. In RNN we made the assumption that earlier epochs would be less relevant than more current epochs. A copy of the results, the information need represented by a question and a saved copy of the game state at the time information need was defined were given to each of our two expert judges to annotate as either relevant to the information need or not. We attempted to control for variation in judges' assessments by selecting judges with similar experience with the game.

They both had significant experience playing a more recent version of the game so they were familiar with some aspects of the game. But neither had much experience with this much older version of the game and so might have a genuine information need for strategic advice about how to win this version of the game. In the next section we compare how our queries performed in aggregate according to our judges using several metrics from information retrieval bolstered by a measure of our judges' judgments.

Mean Average Precision (MAP) measures a retrieval model by aggregating across queries the average precision of each query. Average precision is the average of the precision at each increase in recall up to some k number of results. Starting with the most relevant sentence we calculate recall, which is the number of relevant sentences in the sentences so far divided by the total number of relevant sentences in the result set. If recall increases then measure precision at this ranking. Precision is the total number of relevant sentences so far divided by the total number of sentences so far. Once all k results have been considered then we take the arithmetic mean of all precision measurements for the result set. This is the average precision of the query. The mean of the average precision of each query across all queries using this retrieval model is the MAP measurement of the retrieval model.

We chose to measure MAP for K=10, since we intended to present our results to human annotators, which we can assume expect some results and have some tolerance for sorting through multiple responses. However because our ultimate objective is to help evaluate the performance of a retrieval model for an automated player which has the potential to be able to sort through many more results than the usual player and an architecture for training a retrieval model that has very little tolerance for anything but the most relevant sentence we thought it would should also evaluate our query results using NDCG which rewards queries that place relevant results higher in the ranked result set. Ideally a good retrieval model should return the most relevant result in the first result in the result set. While a K=1 would have evaluated how well the highest ranked sentences performed, each system returned a relevant result in the top ranked position so infrequently that we would not have been able to make a statistically significant comparison using this metric. NDCG however allows us to get a sense of this metric with some forgiveness for a rough retrieval model, which our benchmark was intended to be.

NDCG is a ratio of a weighted sum of relevance judgments, where relevant sentences ranked higher in the result set get a higher score compared to relevant sentences ranked lower in the result set, compared to an ideal ranking of the same result set.

$$z_{kj} = \sum_{m=1}^k \frac{2^{k(jm)} - 1}{\log(1 - m)}$$

And like MAP this score is aggregated across all queries in the test suite using the arithmetic mean. As for MAP, we chose K=10 for NDCG as well.

Results

Results of MAP@K=10 and NDCG@K=10 for 10 queries are shown in Figure 4. Since there was no interaction between judge and retrieval model, the single-variable model for the retrieval model seemed to best describe this data.

For MAP, the difference between retrieval models is (IDF-J1+IDF-J2) - (RNN-J1 +RNN-J2). The mean difference is 0.4688. The standard error of the mean difference is 0.1992. A correlated t-test of the difference in scores finds the mean difference to be statistically significant at the 0.05 confidence level ($t=0.4687609/0.1991941= 2.3533$ and Degrees of Freedom=9, two-tailed $p=0.0431$). The power of this test however is low (0.350). Note because the data were positively skewed we chose a positive transformation. The scores shown below represent the square root of the measurement of the query in each condition.

For NDCG, the difference between retrieval models is (IDF-J1+IDF-J2) - (RNN-J1 +RNN-J2). The mean difference is 0.5222. The standard error of the mean difference is 0.2480. A correlated t-test of the difference in scores failed to find the mean difference to be statistically significant at the 0.05 confidence level ($t=0.5222/0.2480= 2.1056$ and Degrees of Freedom=9, two-tailed $p=0.0645$). The power of this test however is low (0.350). Note because the data were positively skewed we chose a positive transformation. The scores shown below represent the square root of the measurement of the query in each condition.

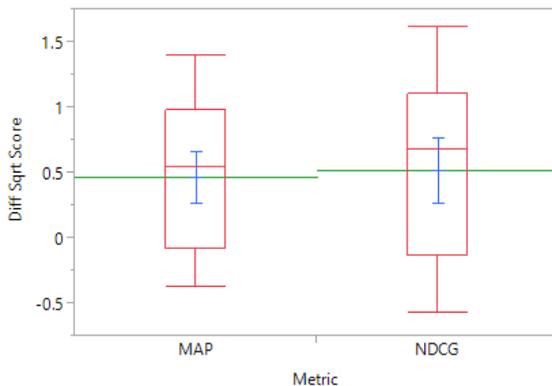


Figure 4: Mean difference in MAP and NDCG

Conclusion

In conclusion, while there is some evidence that the precision of the IDF retrieval model is better than RNN, a larger sample of queries would have made for stronger evidence. Still a difference in MAP of 0.05 percentage points, transformed back to raw average precision, is not a scientifically significant difference. Neither IDF nor RNN performed very well on the given information needs using this traditional approach—a good MAP score should have been higher. That does not however preclude the possibility that these retrieval models could support automated game play. But if one were to pursue this question one should increase the number of

queries in each condition from 10 to at least 40 for better power.

In summary, we set out to evaluate retrieval models for use in querying a single document, a strategic user's guide, for strategic advice that an automated player could use to play a winning game. We began first by characterizing the retrieval model used in Branavan et. al. and comparing it to similar models from the field of information retrieval. We looked at a comparable vector space model (IDF), which became our baseline, and a promising generative language model (RNN), and trained these models offline using only the strategic user's guide. While Branvan et. al. evaluated their retrieval model's performance by comparing how well their automated player performed with and without help from the user's guide, our idea was to better understand the retrieval model in comparison to other retrieval models and to use metrics from information retrieval to evaluate them. Our results showed that IDF performed better than a generative language model for human players but we would have liked to try these models on an automated player. Given that an automated player using Monte-Carlo Tree Search may try many different actions in simulation at each turn before committing to its next action, it is possible that an automated player may judge different sentences to be relevant than our human experts. If the actions implied by the sentence result in a promising $Q(s,a)$ value then the automated player should judge the sentence's advice to be relevant. While we did not have the opportunity to implement a Monte-Carlo Tree Search to get this kind of feedback, we think it would have produced a more nuanced understanding of the performance of these two types of retrieval models in the context of playing turn-based strategy games because we believe while our human experts scored RNN results poorly, an automated player might have made better use of these sentences.

Future work along this line of thinking should consider including more words in the calculation of IDF for each sentence, perhaps by clustering sentences to better distinguish between sentences. One should give RNN more time to train its model to get the best possible results, perhaps leveraging GPU. And finally one should consider caching offline indexes. If we were to attempt to use game scores instead of expert judgments we would need ideal performance at query time to execute 200 queries per roll-out as outlined in Branavan, et al and calculate the expected value of these games thereby giving us a game score that we can use as feedback, a proxy for the relevance of a sentence to the given game state described by the query and measure of the automated player's happiness with the results.

Individual Contributions

The authors, Tianyi Gu and Alison Paredes, contributed equally to designing and executing this project. Alison Paredes coded the indexer and query function for the vector space model. Tianyi installed and ran the recurrent neural network. Both authors facilitated annotations, and each wrote sections of this paper. Alison Paredes performed the data analysis of the the results. Tianyi laid out the final paper and created all of the equations in LaTeX.

The authors would like to thank Professor Laura Dietz, Professor Wheeler Ruml, Bryan Zhang and other members of the UNH AI Group and CS880 for their insightful comments. We also thank Igor Kozlov and Joseph Sichelstiel for being our human experts.

References

- [Bjarnason, Fern, and Tadepalli 2009] Bjarnason, R.; Fern, A.; and Tadepalli, P. 2009. Lower bounding klondike solitaire with monte-carlo planning. In *ICAPS*.
- [Branavan, Silver, and Barzilay 2011] Branavan, S.; Silver, D.; and Barzilay, R. 2011. Learning to win by reading manuals in a monte-carlo framework. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, 268–277. Association for Computational Linguistics.
- [Christopher D. Manning and Schtze 2008] Christopher D. Manning, P. R., and Schtze, H. 2008. *Introduction to Information Retrieval*. Cambridge University Press.
- [Graves 2013] Graves, A. 2013. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.
- [Hochreiter and Schmidhuber 1997] Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.
- [Kalchbrenner and Blunsom 2013] Kalchbrenner, N., and Blunsom, P. 2013. Recurrent continuous translation models. In *EMNLP*, volume 3, 413.
- [Mikolov et al. 2010] Mikolov, T.; Karafiát, M.; Burget, L.; Cernocký, J.; and Khudanpur, S. 2010. Recurrent neural network based language model. In *Interspeech*, volume 2, 3.
- [Silver et al. 2016] Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489.
- [Sutton and Barto 1998] Sutton, R. S., and Barto, A. G. 1998. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.
- [Tesauro and Galperin 1996] Tesauro, G., and Galperin, G. R. 1996. On-line policy improvement using monte-carlo search. In *NIPS*, volume 96, 1068–1074.
- [Zaremba, Sutskever, and Vinyals 2014] Zaremba, W.; Sutskever, I.; and Vinyals, O. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.