Spring 2022 Research Report

Nishant Balepur

May 2022

Problem Statement

Many information synthesis [1] tasks (the process of combining information from one or more sources into a single output) have long been viewed as multi-document summarization tasks [3]. However, these approaches mainly focus on the words in the output, rather than trying to mimic the structure of the output. When investigating Wikipedia articles in the computer science domain, it was found that similar sections had mirrored sentence structure. For example, sentences in the "History" subsection frequently start with "In [Year], . . . ". This phenomenon was also noted for long form question answering, another information synthesis task. This semester, I worked on two task-specific pipelines that exploit structure-mirroring (commonly referred to as 'soft templates' in the literature) for Wikipedia content generation and Long Form Question Answering.

My Approaches

The first approach tested to solve this problem was via follow-up questions. The idea was that if we could train a model to find follow-up questions given input text, we could then use long-form question answering to generate more content. However, this approach showed to be ineffective upon initial testing, so we moved onto the mimicking approach described in the problem statement.

Wikipedia Content Generation

From a high-level overview, this task takes a keyword, and section title as input, and outputs content. I also assume we have access to a list of computer science keywords (Springer data mined list). The structure we want to mirror is that of Wikipedia.

To create our data set, I use the Wikipedia API to find all articles in the computer science domain (using the list of computer science keywords) that have the inputted section title. For each of these articles, we break up the text under every heading into a list of instances. An instance is defined as a sequence of sentences that discuss the same topic. Two sentences are in the same instance if they share a non-stopword stemmed token, contain a pronoun that corresponds to a previously referenced idea (this, that, it, etc.), or contains a token that corresponds to a continuing idea (also, additionally, etc.).

At this point, I have a data frame with three columns: keyword, heading, instances, where instances are a list of sentences as described above. I treat every observation that matches the inputted heading in the heading column as a positive observation, and every other one as a negative observation. I randomly sample the dataset to obtain a 50/50 positive-negative split.

I create two classifiers: one to classify the first sentence of an instance and one to classify the last sentence of an instance. The chosen classifier was fastText, because my thought was that a character-based tokenization approach would better preserve the structure of a sentence compared to other classifiers that focus on word tokens. A train/valid/test split was created, and the models' parameters were tuned appropriately.

To generate content, I first obtain a document by querying on Google the keyword followed by the section, in the form '[keyword] [section]' ('machine learning history', 'machine learning examples', etc.) and selecting the first non-Wikipedia article. I run the classifiers to identify which sentences are beginning instances and which sentences are ending instances. If a sentence is identified as both a beginning and ending instances, the one with the higher confidence level, returned by fastText, is selected. Every sentence above a certain confidence threshold is selected in the final output, and these sentences are then formatted appropriately.

Long Form Question Answering

This task is similar to Wikipedia content generation. It takes a question and keyword (what the question is about) as input. For example, the question "Who invented machine learning?" would have "machine learning" as the keyword parameter. Just like before, I also assume we have access to a list of computer science keywords. The structure we want to mirror is that of Google Search Queries. When typing a question into Google, we are prompted with paragraph answers to our question. Initially, I tried using Reddit's ELI5 subreddit as the structure to mirror, but this proved to be ineffective. I believe this is because Reddit posts are constructed less systematically, and the author of the answer will have a different writing style.

The dataset is created by scraping Google, leveraging a tool called Google Chromium. I create a list of queries by replacing the keyword with the computer science-related keywords. For example, the input "Who invented machine learning?" would give us the queries "Who invented NLP?", "Who invented data mining?", and so on. I query each of these and scrape the question-answer pairs in the "people also ask" section of Google. To ensure the information is concise, I treat the sentence that contains bolded text as the answer to the query. I only ran this LFQA pipeline for "What is the history of machine learning?" so I created a fixed set of rules for making positive/negative training observations, based on the tokens in the question.

I create a single fastText classifier to classify the positive and negative observations sentence-wise. Just like before, I make a train/valid/test split and tuned the models appropriately.

To answer the question input, I first select a document by querying the question into google and selecting the first returned document. I then run the classifier on this document and select the top-k sentences above a certain confidence threshold, which is treated as the final output.

Progress

I was able to successfully implement the two approaches described above. I tested the Wikipedia content generation approach on multiple articles and manually evaluated them myself, which I will show in the results section. I also implemented other classifiers used in other Wikipedia content generation literature and showed that our approach has a higher accuracy. For LFQA, I have only tested the pipeline for “What is the history of machine learning?”, which proved to be effective.

Results

Wikipedia Content Generation

Below are sample outputs for “machine learning history”. We compare our approach with BERT Summarization.

Our Approach

Until the late 1970s, it was a part of AI’s evolution. Then, it branched off to evolve on its own.

Arthur Samuel of IBM developed a computer program for playing checkers in the 1950s. Since the program had a very small amount of computer memory available, Samuel initiated what is called alpha-beta pruning. The program chooses its next move using a minimax strategy, which eventually evolved into the minimax algorithm.

The perceptron was initially planned as a machine, not a program. The software, originally designed for the IBM 704, was installed in a custom-built machine called the Mark 1 perceptron, which had been constructed for image recognition. Although the perceptron seemed promising, it could not recognize many kinds of visual patterns (such as faces), causing frustration and stalling neural network research. It would be several years before the frustrations of investors and funding agencies faded. Neural network/machine learning research struggled until a resurgence during the 1990s.

In the 1960s, the discovery and use of multilayers opened a new path in neural network research. It was discovered that providing and using two or more layers in the perceptron offered significantly more processing power than a perceptron using one layer.

It describes “the backward propagation of errors,” with an error being processed at the output and then distributed backward through the network’s layers for learning purposes. Backpropagation is now being used to train deep neural networks.

The hidden layers are excellent for finding patterns too complex for a human programmer to detect, meaning a human could not find the pattern and then teach the device to recognize it.

In the late 1970s and early 1980s, artificial intelligence research had focused on using logical, knowledge-based approaches rather than algorithms. Additionally, neural network research was abandoned by computer science and AI researchers. Until then, machine learning had been used as a training program for AI.

Its focus shifted from the approaches inherited from AI research to methods and tactics used in probability theory and statistics. During this time, the ML industry maintained its focus on neural networks and then flourished in the 1990s.

BERT Summarization

Machine learning (ML) is an important tool for the goal of leveraging technologies around artificial intelligence. Because of its learning and decision-making abilities, machine learning is often referred to as AI, though, in reality, it is a subdivision of AI.

Machine learning is a necessary aspect of modern business and research for many organizations today. It uses algorithms and neural network models to assist computer systems in progressively improving their performance.

Machine learning is, in part, based on a model of brain cell interaction. Translating Hebb's concepts to artificial neural networks and artificial neurons, his model can be described as a way of altering the relationships between artificial neurons (also referred to as nodes) and the changes to individual neurons. The relationship between two neurons/nodes strengthens if the two neurons/nodes are activated at the same time and weakens if they are activated separately.

The program chooses its next move using a minimax strategy, which eventually evolved into the minimax algorithm.

Samuel also designed a number of mechanisms allowing his program to become better.

The perceptron was initially planned as a machine, not a program. This made the software and the algorithms transferable and available for other machines. Other versions of neural networks were created after the perceptron opened the door to "layers" in networks, and the variety of neural networks continues to expand.

Backpropagation is now being used to train deep neural networks. Its focus shifted from the approaches inherited from AI research to methods and tactics used in probability theory and statistics. Weak learners are defined as classifiers that are only slightly correlated with the true classification (still better than random guessing).

AdaBoost is a popular machine learning algorithm and historically significant, being the first algorithm capable of working with weak learners. Around the year 2007, long short-term memory started outperforming more traditional speech recognition programs.

In 2014, Facebook developed DeepFace, an algorithm capable ofrecognizing or verifying individuals in photographs with the same accuracy ashumans. ML algorithms combined with new computing technologies promote scalability and improve efficiency.

Combined with business analytics, machine learning can resolve a variety of organizational complexities. Google is currently experimenting with machine learning using an approach called instruction fine-tuning.

Image used under license from Shutterstock.com
You must be logged in to post a comment.
© 2011 – 2022 Dataversity Digital LLC — All Rights Reserved.

Long Form Question Answering

For LFQA, I have only tested the answer to the question "What is the history of machine learning?" The outputs from three different documents are pasted below:

Until the late 1970s, it was a part of AI's evolution. The model was created in 1949 by Donald Hebb in a book titled The Organization of Behavior (PDF).
Arthur Samuel of IBM developed a computer program for playing checkers in the 1950s.
Arthur Samuel first came up with the phrase "machine learning" in 1952.
Described as the first successful neuro-computer, the Mark I perceptron developed some problems with broken expectations.
In 1967, the nearest neighbor algorithm was conceived, which was the beginning of basic pattern recognition.
Additionally, neural network research was abandoned by computer science and AI researchers.
The concept of boosting was first presented in a 1990 paper titled "The Strength of Weak Learnability," by Robert Schapire.

1950 — Alan Turing creates the "Turing Test" to determine if a computer has real intelligence.
1952 — Arthur Samuel wrote the first computer learning program. The program was the game of checkers, and the IBM computer improved at the game the more it played, studying which moves made up winning strategies and incorporating those moves into its program.
1957 — Frank Rosenblatt designed the first neural network for computers (the perceptron), which simulate the thought processes of the human brain. 1967 — The "nearest neighbor" algorithm was written, allowing computers to begin using very basic pattern recognition.
1997 — IBM's Deep Blue beats the world champion at chess.
2011 — IBM's Watson beats its human competitors at Jeopardy.
The AlphaGo algorithm developed by Google DeepMind managed to win five games out of five in the Go competition.

Who was the first person to create the concept of a self-learning machine? The test was designed to determine whether a computer has human-like intelligence.

1952 – Arthur Samuels, the American pioneer in the field of artificial intelligence and computer gaming, wrote the very first computer learning program.
1967 – The nearest neighbor algorithm was written for the first time this year. The 1990s – during the 1990s, the work in machine learning shifted from the knowledge-driven approach to the data-driven approach.
For starters, IBM's Watson managed to beat human competitors at Jeopardy.
2016 – this was the year when Google's artificial intelligence algorithms managed to beat a professional player at the Chinese board game Go. Natural language processing (NLP).

Assessment

Wikipedia Content Generation

Overall, I believe that the outputs are pretty good. They are factual and contain rich information but do sacrifice some readability in the process. Through human evaluation, it was clear that our approach outperformed traditional summarization. I evaluated the outputs using Grammar, Redundancy, Referential Clarity, Focus, and Structure and Coherence, which is commonly used for evaluating the linguistical quality of text [3]. I found that on average and in fact, in most cases, our approach outperformed BERT summarization. As described earlier, I also compared our approach with other Wikipedia generation approaches that use classification. The table below displays the accuracy of these approaches, which were tested using 5 different topics and 3 different headers:

Section	Model	Train Accuracy	Test Accuracy
History	KNN	0.789	0.568
	RandomForest	1.000	0.612
	ALBERT	0.735	0.760
	fastText-sent	0.992	0.842
	fastText-beg	1.000	**0.860**
	fastText-end	1.000	0.806
	fastText-multi	0.987	0.705
Applications	KNN	0.800	0.620
	RandomForest	1.000	0.624
	ALBERT	0.542	0.560
	fastText-sent	1.000	0.784
	fastText-beg	0.978	**0.796**
	fastText-end	0.997	0.777
	fastText-multi	0.997	0.680
Definition	KNN	0.782	0.576
	RandomForest	1.000	0.608
	ALBERT	0.559	0.520
	fastText-sent	1.000	0.708
	fastText-beg	0.909	**0.726**
	fastText-end	1.000	0.639
	fastText-multi	0.997	0.573

Long Form Question Answering

I have only tested on the single example, but the output for this is quite good. It reads more like a statement of facts, but I believe that it has very high factual accuracy and depth of information. The outputs can convey a lot of information with only a short amount of text.

I was unable to complete any meaningful assessment compared to other LFQA approaches at this point.

Reflection

This past semester I had a great research experience. I felt like I learned a lot more about natural language processing and information retrieval, and I believe I have become much better at reading and understanding research papers. I also gained some insights into the research and writing process, and I hope to apply what I have learned in the future. I thought our collaboration went well, and I am grateful for your guidance and frequent meetings throughout the semester.

Future Plan

Overall, I was quite satisfied with these results but there is still a lot of future work to be done. To begin, I would like to test long form question answering more and compare it to the current state of the art approaches. More importantly, however, I would like to try and make this approach less task specific. I believe we can create a single neural network that can both identify positive and negative observations as well as classify the text. By doing this, our pipeline can be more generalized and be used for various tasks.

I found related work which has explored this 'soft template' approach for abstractive summarization [2]. They have observed the same things as us, focusing on the structure rather than the lexical content of the words. However, I think that our task is still novel because we are considering the extractive case rather than the abstractive case.

One issue I found with their abstractive approach was factual accuracy, but extractive approaches are not as subject to inaccuracies because it takes sentences word-for-word. Also, I thought that we could introduce some joint learning into this approach, using fastText embeddings to both identify positive/negative observations as well as classify them, at the same time.

References

[1] Enrique Amigo, Julio Gonzalo, Victor Peinado, Anselmo Penas, and Felisa Verdejo. An empirical study of information synthesis task. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, pages 207–214, 2004.

[2] Chenxin An, Ming Zhong, Zhichao Geng, Jianqiang Yang, and Xipeng Qiu. Retrievalsum: A retrieval enhanced framework for abstractive summarization, 2021.

[3] Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences, 2018.