Using LLMs and Reinforcement Learning to Build an Agreeable Political Ideology

Ben Wilen

October 2023

1 Abstract

The goal of this project is to see if political chatbots can be trained in a certain belief system (either Democratic or Republican in this case), and then using prompt engineering, discuss different political issues. By incorporating a reward system for each of the responses, we attempted to use reinforcement learning to modify the chatbots underlying model and ultimately see if they could agree on certain political issues, or even better, create completely new approaches to political issues.

2 Introduction

There are two main paradigms in machine learning. The first is using past human data to build an AI model to replicates human action or thought. Both supervised learning and unsupervised learning fit this paradigm. This paradigm is great when we want AI to behave like humans. We might want a chatbot that sounds like if it was a person speaking, or we might want autonomous vehicles to interprets street signs like how we interpet them. But, this paradigm has a limit: it will only ever be as accurate or helpful as humans are. It will only restate and reinact what humans have already done.

But what if we want an AI agent that can do better than humans? Or in the specific case of large language models and chatbots, what if we want a chatbot that creates new ideas instead of just repeating previous ideas? The second machine learning paradigm—reinforcement learning—doesn't use past human data. Instead, we place in agent in an environment (often simulated) and allow the agent to explore the environment, giving it rewards for being in certain states. For example, reinforcement learning can be used to train a chess AI player. If we can determine a state that is a win or loss, we can use that to back propagate to previous states and determine the proper actions the agent should take.

So, while a AI chess player trained on human data might be as a good as the best chess players in the world, this reinforcement learning bot will be better. It will make moves that no human has ever tried before, but will mathematically lead to a higher chance of winning. And, this happened with AlphaGo in 2016. AlphaGo, a reinforcement learning trained bot for the Chinese board game Go, beat Lee Sedol, the best Go player in the world. What was unique about the way AlphaGo won, however, was that it tried new moves that humans have never done, and resultingly taught Lee Sedol and other players new approaches to the game.

Let's get away from board games and back to idea generation, as discussed above. We can train chatbots on specific political ideologies are parties, such as a Democratic bot and Republican bot. And we can ask them to debate political issues. But, because the bots are mostly trained on past human responses, the debates will sound very similar to previous human debates in these issues.

So, let's try something new. Let's take a Democratic and Republican bot, and have them speak to each other. But, at each iteration, let's let them assign a reward to the other individuals response. In a way, it will be compromises. Theoretically, to reach a maximum reward, the bots might agree on approaches to political issues that humans have yet to think of. Let's try it.

3 Process/Methods

3.1 Building the Bots

These section will both serve as an overview of the methods I used in my research, but also my process throughout the winter study as well.

The first step was finding a NLP dataset with political orientations. I originally wanted to train 4 chat bots to communicate: a socialist bot, communist bot, capitalist bot, and libertarian bot. However, there were no immediate datasets available with these categories, and for the sake of time management over winter study, I decided that scraping the web for corpi in these areas was too time consuming.

Tweets are very often used to train NLP models. They provide short passages of human text, and exist in billions. However, given I wanted my bots the accurately represent democratic and republican beliefs, I wanted to make sure my dataset contained useful content.

A preexisting Hugging Face model[1] was trained as a NLP classifier to identify whether statements were democratic or republican. To train this classifier, the developer used a training set of tweets annotated by political party[2], perfect for my models. This dataset was also unique that the tweets were already hydrated (scraped from Twitter). Given the accuracy of the existing classifier, I was confident that the tweets contained substantive content and were annotated well. This dataset contained around 40,000 of each democratic and republican tweets.

After the datasets were generated using typical dataframe processing, I then fine-tuned a preexisting LM with my unlabeled tweets. The model I chose was the Writer/palmyra-small model containing 128 million weights. The reason I chose this model was two-fold: 1) it ranked among the best LMs on Hugging Face. 2) The drastically low amount of weights (most other LMs had multiple billions of weights) would slow down fine-tuning the model and then EXPONENTIALLY slow down the reinforcement learning to the point where it would never realistically finish training. In the future, one could explore fine-tuning larger models.

There are two built-in classes for fine-tuning Hugging Face models. If I had more time to work on this project, I would manually fine-tune them with PyTorch to improve the performance of the chatbots by letting me invoke custom modules such as an optimizer. The two classes were a Trainer class and an SFT-Trainer class. The Trainer class is a general purpose training module, typically requiring larger datasets and often used to train language models from scratch. The SFTTrainer (Supervised Fine Tuning) is built specifically for fine-tuning pre-existing models with smaller datasets. Given these specifications, I assumed the SFT trainer would perform better, but I wanted to train both and compare. As a result, I trained 4 bots, two republican and two democratic bots.

With these models trained and saved, I first manually evaluated these models by generating text with sample political issues, such as "What is your opinion on abortion?" I quickly learned I needed to increase the training epochs. The current state of this project is training these higher-order epoch models. However, during these training periods, I continued to work ahead into the reinforcement learning section of this project.

3.2 Reinforcement Learning

The reinforcement learning component is the heart of this project and required the most exploration and creativity. However, the paradigm I decided to move forward with uses preexisting models and infrastructure, combining them in a unique way I have previously not seen models used. Using existing models allowed me to use highly performant and easily distributed models.

[1] https://huggingface.co/m-newhauser/distilbert-political-tweets
[2] https://huggingface.co/datasets/m-newhauser/senator-tweets

The general architecture of my paradigm is two agents being used in each other's environment to provide a reward and next observation. In a traditional reinforcement learning paradigm, one agent samples from an environment, is given a reward for that action, and that reward is used to update the policy (the underlying neural network).

In this scenario, the first bot (the initial bot being republican or democratic does not matter) is given an observation which is a prompt randomly selected from a list of starter prompts. These consist of political questions, such as "What are your thoughts on ISSUE X?" The first bot generates a response. This is the action taken. The second bot rates the first response on a scale of 1-100 given the following prompt: "How much do you agree with this text on a scale of 1-100: RESPONSE FROM BOT 1?" This is the reward given to the action. The second bot then replies to the response. This is both the observation for the first bot and the action for the second bot. The first bot then rates the response as the reward for the second bot, and the conversation continues with this pattern.

There are two potential convergence metrics I explored for determining when to stop training. The first is to stop training after the bots have rated each others response a 45-55 a certain number of times in a row. I believe this criteria is superficial. However, as I will discuss in the implementation later, it does work well with the library being used to implement training.

The second convergence metric is the actual weights of the bots' policies. For example, we can say the bots have converged if their weights are under a certain euclidean distance apart. I believe this method dives deeper into the underlying mechanism of the policies, but is less interpretable. It is unclear with such large neural networks what weights actually correspond to. Knowing the bots are giving neutral rewards is much clearer than trying to interpret the bots weights. However, the goal of this project is to have the bots converge to a new and unique ideology. As a result, comparing the bots anatomy, their weights, might be a better indication that they are becoming a single bot.

To implement this paradigm, I used Ray's RLlib package for reinforcement learning. Ray specializes in production level reinforcement learning through distributed training. However, the largest motivator for me using this library was that I had used it recently in the past and was familiar with it.

I first had to build the environment using Gymnasium, an environment package. A Gymnasium environment requires a initializer, reset, and step function. This code follows the paradigm above in a very straightforward manner.

I then had to build a custom algorithm. The algorithm implements the logic above (the two bots communicating in each training iteration). I additionally had to customize the policy to pull from the saved PyTorch model I had fine-tuned before. I lastly pass this algorithm into a PPO trainer. I again chose PPO not as much for its performance but because I was familiar with it from previous work. Future exploration could include exploring other algorithms such as DQN.

4 Ongoing Work

First, although the codebase is almost complete, the training is nowhere close to done. Future exploration should explore training these models on more GPUs/TPUs or computers specifically built for machine learning. I do believe that this work is possible to be completed given significant resources as LLMs such as ChatGPT do use reinforcement learning after their initial training. A challenge will be, however, allowing these models to generate text at each training iteration without slowing down training.

At a higher-level, I also believe we can improve on how the bots are prompted to respond to each other. My naive approach, "What's your response to (previous response)?" might lead to non-substantive discussion after a while. I believe that for this paradigm to work, the discussion needs to be adversarial, prompting strong responses on controversial subjects. I also believe that for the responses to align with the goals of

my project, we could also preprocess prompts to include bits such as "Do you have a compromise?", "What would you do instead?", "Do you see a solution that you both will agree on?", etc. Giving the bots leading prompts like this might provoke more substantial changes to their weights.

Although winter study is complete, I plan to continue training the bots and build out the custom algorithm for training. I would also love to train more bots on different political party data as I believe it could lead to more intriguing convergences. And, if I can complete these tasks, I believe there is substantial software engineering work to be completed to improve the training time and infrastructure of this system.