CS 520 Final Exam Q2

Tej P Shah, NetID: tps75

Question 1: War Games (30 Points)

Observe that the AI agent plays Tic-Tac-Toe against itself for numerous simulations. After many simulations, the AI agent concludes that there is no winning strategy for Tic-Tac-Toe.

It's realistic for the AI agent to conclude that there is no winning strategy for Tic-Tac-Toe if both players play optimally, an assumption of the Minimax Algorithm. Suppose we have 2 players: X and O. Suppose that you score the utility of a game state of a board of 9 squares as $+1$ if X wins, -1 if O wins, and 0 otherwise. Player X's objective is to maximize its score, and Player O's objective is to minimize its score. Observe the entire state space for the classic 9×9 Tic-Tac-Toe game $3^9 = 19,683$ total possible board configurations. Since the state space size is small and Tic-Tac-Toe is deterministic, we can brute force a game tree to all terminal states from a given state of the board, scoring all the intermediate boards along the way. Using the scored boards, we can propagate utilities up the game tree to the current state of the board, so X picks the action with maximum utility, and O picks the action with minimum utility. Since X and O both move optimally, the game always results in a draw.

Observe that if the Minimax Algorithm underpins the AI Agent's logic for Tic-Tac-Toe, it would be completely unnecessary to run multiple simulations of the game to "learn" an optimal strategy with demonstrations since the environment is completely deterministic.

It's realistic to argue that self-play reinforcement systems can learn meaningful game strategies if a game can be formulated as a Markov Decision Process (MDP). Indeed, that is the core idea underpinning the remarkable success behind AlphaGo Zero. As depicted in the movie, though, it'd be a stretch to argue that AlphaGo Zero underpins the logic behind the AI Agent's logic for Tic-Tac-Toe: in the movie, from the initial state, the policy is always optimal, which is highly unrealistic. Further, for a small, deterministic environment, AlphaGo Zero is overkill.

By a similar argument to AlphaGo Zero, it's unrealistic Monte Carlo simulations are used to inform the decision-making of the AI Agent's logic for Tic-Tac-Toe since all the simulations in the movie result in a draw. None result in a win or loss - hence, the actions selected are not random, so the repeated simulations are not helping calculate the utility of a particular state for training. Reasonably, we can infer that the AI Agent is not learning from data in an online fashion. It is most likely that the AI agent learned data in an offline manner. The only way that Monte Carlo could be underpinning the AI Agent's logic in the game is the Monte Carlo simulations were computed before the games running in the movie.

It is realistic to argue that the AI Agent calculated the optimal policy or utilities for each state in the state space since it is completely deterministic, small, and easily formulated as an MDP by solving the Bellman Equations with either Value Iteration or Policy Iteration using dynamic programming, which will be guaranteed to converge for this situation. Hence, every game results in a draw because players X and O always make decisions based on the optimal policy. As with the case with Minimax, running more simulations after the optimal utilities/policy have been determined is unnecessary.

In the movie, we see that increasingly more compute is needed as more simulations are being run. This is unrealistic since the Tic Tac Toe game has a finite horizon, the representation of the game state is likely simple, and there is an upper bound maximum on compute (the longest sequence of actions from X and O that result in a terminal state is fixed). Furthermore, based on the arguments earlier, it is likely that the AI Agent is NOT learning from data in an online fashion. Reasoning from an economic perspective, Minimax has zero fixed cost but a non-zero variable cost; by contrast, Value/Policy Iteration has a single-time fixed cost but a constant time variable cost. Hence, it'd be more likely that Minimax underpins the AI Agent's logic than basic reinforcement learning since it requires more compute than Value/Policy iteration.

After the agent determines the optimal policy for Tic-Tac-Toe, the AI Agent plays self-play games of global thermonuclear war against itself as the US and the USSR. At the end of the simulations, the AI shuts down and concludes that the best policy is not to play the game.

For this game of global thermonuclear war, the state space appears to be uncountably infinite, as a missile can strike any coordinate on the map. Hence, it is not likely that Minimax or simple Value/Policy iteration will work in this scenario. The most feasible algorithm to work in this scenario would either be AlphaGo Zero or using Monte Carlo Simulations to determine good actions due to the large state space. But, it is unlikely in the movie that the system is "learning" since the policy or strategy does not seem to visibly improve over successive iterations, as would be the case with AlphaGo Zero or Monte Carlo Simulations.

Suppose that there is some algorithm that is "learning" or intelligently updating its decision-making based on the simulations of the game of global thermonuclear war. It is a stretch to say that the AI Agent will output that "the only winning move is not to play" since that itself is an action. In other words, "not playing" should be a potential action within the action space for the AI Agent to output that result. But, judging from the reactions in the movie, it is likely that not playing the simulation is surprising, so it is unlikely that "not playing" is a valid action in the action space.

Question 2.1: Hallucinations in LLMs (5 Points)

In the last few years, researchers have scaled up the breakthrough transformer architecture components with large amounts of compute and data to achieve impressive results in various domains. In this new foundation model paradigm, Large Language Models (LLMs) such as Chat-GPT and Galactica show remarkable capabilities to generate plausible-sounding text. Unfortunately, these LLMs hallucinate confident responses to questions, regardless of whether the information they provide is true. There are a couple reasons for this hallucination.

(1) Training Objective: Generally, LLMs "don't know any better" or have any preference to generate a true statement versus a false statement: simply, the LLM will predict the statement with the highest probability, irrespective of the statement's veracity. Already, by saying LLMs "don't know any better", we implicitly and unintentionally imbue LLMs with human characteristics. Until further research is conducted, we need to be wary not to anthropomorphize LLMs. Why? At it's very core, LLMs are next-token prediction machines: a statistical autoregressive model to predict the maximum likelihood of the next token given a context. Formally, given words $w_1, w_2, \ldots, w_{t-1}$, a LLM learns to maximize the probability distribution $p(w_t \mid w_1, w_2, \ldots, w_{t-1})$ with a standard loss function $\mathcal{L}$ like cross-entropy loss. Nowhere in this training objective is the LLM incentivized to not make stuff up. Simply, the training objective pushes the LLM towards generating plausible completions given a context.

Recent efforts by OpenAI, most notably ChatGPT, to align LLMs in the GPT series with Reinforcement Learning from Human Feedback (RLHF) make some progress towards mitigating LLM hallucinations. At its core, the RLHF approach is three-pronged: (1) a base LLM is fine-tuned with a dataset of desired completions given some prompt from human contractors; (2) the fine-tuned model in (1) is used to generate several outputs, which are ranked in preference order by human contractors to train a reward model; (3) using the generations from (1) and the reward model from (2), a standard reinforcement learning algorithm like Proximal Policy Optimization (PPO) is used to train a policy for generating outputs to maximize rewards. Optimistically, if humans are incentivized to generate outputs they *believe* to be true, then we'll see LLM generations that *humans are less likely to make up*. But, crucially, observe that humans often have commonly-held beliefs that are not true! Hence, LLMs modified with the RLHF approach will fail in the same way that humans do, making stuff up in the same way that humans do. Clearly, further research is required to create a robust solution to mitigate LLM hallucinations as LLMs become more ubiquitous.

(2) Training Dataset: The training objective argument answers why a model doesn't have a rational preference for a truthful response versus a non-truthful response. The training dataset argument explains why LLMs plausibly include non-truthful responses in their outputs at all. LLMs have remarkable autoregressive generation capability as the dataset size scales, even gaining some semblance of in-context learning. But, to collect a dataset of such magnitude is non-trivial. At a very high level, many researchers scrape vast amounts of publicly available data, both truthful and untruthful, to make LLMs work at scale. Unsurprisingly, LLMs hallucinate non-factual responses at inference time if the LLM has seen non-factual responses during training. In short, LLMs are only as good as the data that it's trained on.

Question 2.2: LLM KB Interface Using RLHF (15 Points)

Suppose $\mathbf{x}$ is your natural language prompt. We generate a text prompt $\mathbf{y} = f(\mathbf{x})$, with a learned function f that maps a raw prompt $\mathbf{x}$ to a better prompt $\mathbf{y}$. The better prompt $\mathbf{y}$ is input as the prompt to Google to achieve good, targetted results from Google for your raw prompt $\mathbf{x}$. As a brief aside, note that this is very similar to the idea of "prompt engineering" and there has been very recent interest in converting raw user prompts to better prompts for foundation models like Stable Diffusion or DALL-E 2 and other image generation models. In this scenario, Google is the knowledge base and we are prompt engineering the LLM that is querying the Google knowledge base with f. A system to learn f is detailed below:

Method: The goal is to engineer "good" prompts $\mathbf{y}$ given raw text $\mathbf{x}$ for "better" Google results while preserving meaning and intent of the original prompt. There is ambiguity in the "good" prompt or "targetted" result. Since this is a matter of preference, we will take a similar approach to ChatGPT's RLHF approach outlined previously in **Question 2.1**.

(1) Dataset Collection: Manually collect $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{m}$. For example, $\mathbf{x}_1 =$ "the apple" and $\mathbf{y}_1 =$ "the apple and its nutritious facts." A contracted labeler generates a sample input $\mathbf{x}_i$ and has another contracted labeler generate a better sample output $\mathbf{y}_i$, that is more detailed than $\mathbf{x}_i$, preserves semantics of the original prompts, and qualitatively appears to have more relevant Google results than the original prompt (as defined by the contractor).

(2) Representation: Using a GPT-2 tokenizer from HuggingFace, generate a representation of the prompt $\mathbf{x}$ as a sequence of tokens $\mathbf{x} = [x_1, x_2, ..., x_n]$. Similarly, generate a representation of the better prompt $\mathbf{y}$ as a sequence of tokens $\mathbf{y} = [y_1, y_2, ..., y_m]$. After the text is pre-processed and tokenized, vector embeddings $\mathbf{x}$ and $\mathbf{y}$ for each token in the sequence is generated as the sum of the positional encoding and the token embedding. This representation captures meaningful information about the location of the words in the sequence as well as the semantics of each word within the sequence, which is useful for seq-to-seq learning.

(3) RLHF: We initialize the parameters θ of $\hat{f}_\theta$ with the trained model of GPT-2 on HuggingFace. Leveraging the power of transfer learning, we fine-tune $\hat{f}_\theta$ on $\mathcal{D}$ with supervised learning to develop a supervised policy π_s. With the fine-tuned policy π_s, we now sample k prompts $\mathbf{x}_i$ from $\mathcal{D}$, and for each prompt $\mathbf{x}_i$, generate s sampled outputs using $\hat{f}_\theta(\mathbf{x}_i)$. Using contracted labelers to rank the outputs of the s outputs for each of the k prompts $\mathbf{x}_i$, we train a Reward Model $R(\mathbf{x}_i)$ to measure the goodness of a generated output from π_s . Using the deep reinforcement learning method of Proximal Policy Optimization, initializing $\pi_{ppo} = \pi_s$, we train π_{ppo} to generate outputs $\hat{y} = \hat{f}_\theta(\mathbf{x}_i)$ that maximizes $R(\mathbf{x}_i)$. This RLHF approach enables us to get better "prompts" from the raw input that aligns with what the human contractors believed to be better prompts. This achieves good, targeted results when inputted into Google search, while preserving the meaning and intent of the original prompt.

(4) Inference: $\hat{f}_\theta(\mathbf{x}_i) = \pi_{ppo}(\mathbf{x}_i) = \hat{\mathbf{y}}_i \approx \mathbf{y}_i$. We take $\mathbf{x}_i$, pass it through $\hat{f}_\theta$, and generate $\hat{\mathbf{y}}_i$ decoded from an embedding to a word as the "better" engineered prompt put into Google. The decoding scheme uses a similar representation to the encoding scheme with GPT-2 tokenizer.

References

- openai.com/blog/chatgpt/
- youtube.com/watch?v=s93KC4AGKnY