PROJECT MILESTONE

Jonathan Schory & Jonathan Schwartz

MARCH 29th, 2020

Abstract

The MuZero algorithm which combines a tree-based search (MCTS) with a learned neural model has proved to yield superhuman results in a variety of domains such as Chess, Go, Atari. In addition, a fundamental feature of the MuZero algorithm is that it does not require prior knowledge of domain dynamics, game rules, or pre-determined strategy. The course exercise, which implements AlphaX over an NP-Complete problem can therefore be enhanced by applying the MuZero methodology. With Traveling Salesperson Problem (TSP) as an evaluation domain at a manageable scale, we seek to demonstrate state-of-the-art performance with MuZero.

The following report will seek to demonstrate progress and analyze future steps as part of the course capstone project. In this analysis, we will examine the methodology, resources, and approaches used in the initial steps from formulating the Traveling Salesperson Problem as a single player CSP game to the eventual enhancement of AlphaGo framework leading to the MuZero algorithm on the TSP domain.

Introduction

It is a well known fact in the field of computer science that look-ahead prediction search can yield astonishing success and accuracy in the realm of artificial intelligence. Nonetheless, many if not most of these state-of-the-art planning frameworks require prior, elaborate familiarity with environment-specific details such as game rules, valid actions, potential strategies, etc. Making use of avant-garde reinforcement learning mechanisms, novel frameworks seek to first train a model on the environment and its dynamics for planning algorithms governed by the learned model. Specifically, the MuZero framework (Schrittwieser et. al) which extends the AlphaZero methodology (Silver et. al) uses a learned model to predict the key details for future planning.

Problem, Goals, and Formulation

The original course exercise requires to implement AlphaX where X is any NP-complete problem over a graph. The NP-complete problem chosen for the exercise as well as the project is the Traveling Salesperson problem "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?" The goals and formulation are as follows:

- Formulate the problem as a single-player game and use MCTS to solve a small random instance of the problem with 10 nodes. Actions in this case are limited to node manipulation (adding or removing nodes to the graph). User may be prompted to directly determine how and when actions are taken (limited to computationally feasible actions). Node features will be xy coordinates and graph edges represent distances.
- Add a Graph Neural Network representation to speed-up the MCTS, then implementing AlphaZero with self-play.
- The AlphaZeroTSP will then be on a small random instance of the problem with 10 nodes. Then to be enhanced to MuZero TSP and evalated in a similar fashion on small scale dataset.
- Extend the dataset to a maximum of 20 nodes and thoroughly evaluate performance and limitations of MuZero implementation on an NP-Complete problem through plotting and probability analysis.
- Thoroughly compare and examine the performance difference between AlphaZeroTSP and MuZeroTSP considering a metrics, namely the amount of random instances solved within a close factor of the optimal Traveling Salesperson Problem solution (e.g. factor of 1.1 from the brute-force algorithm).

Data, Approaches, and Algorithms

The data and algorithms used in our first approach were sourced from the provided libraries recommended in the exercise description, namely the OpenSpiel Library. The OpenSpiel framework provides an array of environments and algorithms for research and analysis in the realm of reinforcement learning and search/planning in AI games.

The choice for OpenSpiel was based, among other reasons, for its dynamic nature and the library's ability to support n-player zero-sum, cooperative and general-sum, one-shot, sequential, strictly turn-taking, perfect and imperfect information games. A main advantage of the OpenSpiel library is the variety of example games which implement reinforcement search algorithms and deep learning frameworks. As such, our methodology sought to build upon a pre-existing game, in this case the OpenSpiel Tic Tac Toe example game. The existing Tic Tac Toe code implements the OpenSpiel API and utilizes critical functions for the TSP games such as legal actions, calculation of rewards, returns, etc. While we were able succesfully implement the random_game using the OpenSpiel framework, we reached a roadblock when attempting to implement Monte Carlo Tree Search using the OpenSpiel API. At this point, we transitioned to a new approach: The Alpha Zero General repository provided in the course website. The AlphaZero General framework is flexible and dynamic reinforcement learning environment for self-play based on the AlphaGo Zero paper (Silver et al). Similar to the OpenSpiel, the AlphaZero repository provids a framework for reinforcement games that can be manipulated for the user's needs, in our case: the formulation of TSP as a single player game and implementation of MCTS algorithm for the tour optimization process. A key advantage in the AlphaZero framework is the ability to directly manipulate pre-existing .py files which execute the needed functions for TSP game incorporating the Monte Carlo Tree Search. Namely, the Game.py and MCTS.py files are user-friendly and easily maliable Python code segments which requires relatively little alteration to run properly on our TSP single player game. Both Game.py and MCTS.py outline and provide the necessary functions and algorithms required to carry out the game, which in turn allowed us to remove and manipulate particular aspects of the code and execute the game successfully.

Evaluation Criteria

The evaluation criteria for the TSP need be split into the two steps of the project formulation: AlphaZero and MuZero, with both cases being compared with the brute force optimal solution algorithm. In this particular case, the brute force algorithm makes uses of examining all possible permutations of nodes in the "tour" yielding an optimal solution at exponential time complexity.

The evaluation for the AlphaZeroTSP single player game is mainly based on the exercise evaluation which calls for plotting the percentage of random instances which MCTS solves within a 1.1 factor of the optimal (brute force) solution of the tour. On a similar train of thought, evaluation criteria for the initial part of the project will also include analysis of performance of the GNN enhancement to the AlphaTSP code. In the same fashion of MCTS analysis, the investigation of GNN AlphaTSP too include plotting the percentage of random instances solved within a 1.1 factor of the brute force calculated solution as a function of training samples and epochs.

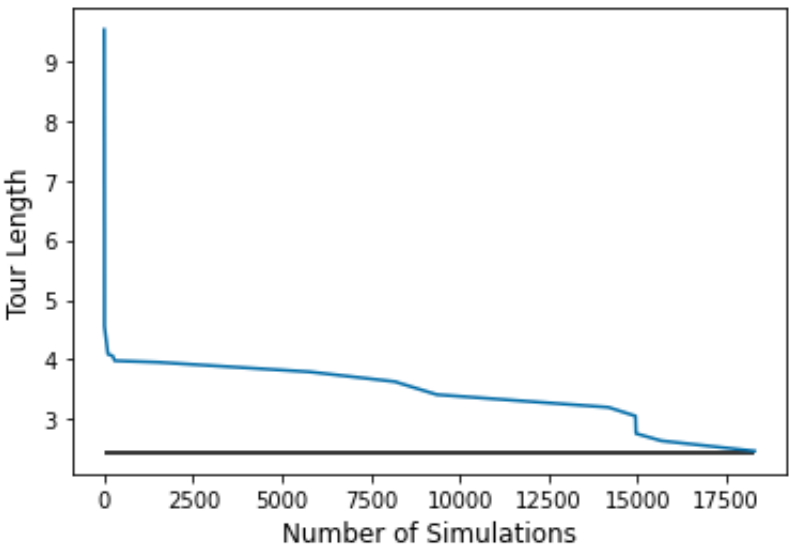

Figure 1: Graph Illustrating MCTS Progress After X Simulations

As noticable from the graph, we observe that over time (e.g. as the number of simulations increases) the tour length approaches the optimal tour length. Specifically, we note that the blue curve (representing the returned reward fromt the Monte Carlo Tree Search algorithm) converges towards the shortest tour length (computed using brute force calculation on possible permutations). Currently, our data shows that many of our sample MCTS runs yield an ultimate result that is within a 1.1 factor of the optimal solution. This is to be both plotted precisely in next progression of the exercise, as well as increase in accuracy and efficiency once introducing a trained model as part of the search foundation.

The evaluation criteria for the MuZero implementation of TSP will likely be based on a similar analysis of plots and comparison to brute force solution as mentioned in the AlphaTSP framework. Specifically, the MuZero

evaluation may include a plot analysis of the MCTS random instances as well as the GNN enhanced version in comparison to the 'hardcoded' optimal solution applying brute force calculation.

Naturally, evaluation criteria for all frameworks (both AlphaTSP and MuZeroTSP) will rely directly on its performance in comparison to the brute force solution and will be modified accordingly.

Current Assessment and Evaluation of Progress

Currently, the majority of progress and status with regards to the capstone project is related to the initial foundation: completion of the exercise. Since the basis of our project relies on successful implementation of the exercise itself and its sub components, the main focus of the team was to implement the exercise completely.

As of right now, the implementation of the implementation of the AlphaZeroTSP framework appears to be running smoothly with proper execution of the Monte Carlo Tree Search algorithm. As such, the team has been to establish a functional environment for running the TSP single player game using 10 nodes. Moreover, we have been able to gain a deep understanding of the underlying processes which govern the AlphaZero methodology. Needless to say, the progress with regards to the MuZero implementation awaits actualization. Yet, as part of the exploration of Github repositories and hands-on familiarity with AlphaZero, it is likely that progress will gain momentum after absolute completion of the exercise and evaluation metrics of AlphaZeroTSP. As of right now, the team is making progress with regards to the implementation of a neural network to accompany and enhance the existing TSP code and the successful MCTS algorithm.

Conclusions, Problems, and Future Steps

The main focus in future steps regarding capstone project progress would be to transition the successful implementation of the AlphaZeroTSP game to the successful MuZeroTSP. As such, we intend to utilize the MuZero General repository, an online implementation of MuZero based on the Google DeepMind paper and the associated pseudocode. As with AlphaZero general, MuZero General is designed to be easily adaptable for every reinforcement learning environments. As part of our implementation of MuZero we intend to modify the provided .py files in the repository, namely the the abstract_game.py file. This file, provides a generalized and adaptable framework which can be modified to enhance the TSP game from AlphaZero to MuZero with little alteration. Moreover, we intend to make use of the existing MuZero Tic Tac Toe example file which may be of further assistance to better understand the intricate details of MuZero implementation on a game.

In further detail, our MuZero framework will be able to learn a model that when iteratively applied can predict and evaluate the quantities most pertaining to search and planning, namely the reward, the action policy, and a value function. Specifically, according to the MuZero reference paper our TSP model will consists of three components for prediction, dynamics and representation. These will be dependent on a previous hidden state s as well as a candidate action a, the dynamics function then yield both an immediate reward r and a new hidden state. Accordingly, the value function v and policy p are computed through the prediction function f. As such, the representation function, dynamics function, and prediction function combine to generate a model that is able to (hypothetically) search over the entire future trajectory space. In turn, the MuZero algorithm or "agent" will be able to internally invent the Traveling Salesperson Problem dynamics that are able to direct accurate, and high-reward planning. In this case, MuZeroTSP should be able to expedite and increase the efficiency of optimal tour search over the search space of all possible routes in the graph.

References and Previous Work

1. https://github.com/jonSc8/deep_learning_final_project
2. https://github.com/deepmind/open_spiel
3. https://github.com/suragnair/alpha-zero-general/blob/master/MCTS.py
4. TA provided tutorials on CourseWorks
5. https://deepmind.com/blog/article/alphago-zero-starting-scratch
6. https://github.com/werner-duvaud/muzero-general
7. Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model: arxiv.org/abs/1911.08265