CS238 Final Project: Predicting NBA Game Outcomes using POMDPs

Authors
Paul Steenkiste pws

Jaak Uudmae juudmae

Abstract

There is a lot of literature focusing on using POMDPs to choose actions in an uncertain state space (indeed, this is their explicit and intended purpose). Here, we attempt to expand the domain of problems that POMDPs can be used to solve by modeling prediction as a POMDP. Using historical data from the National Basketball Association (NBA), we attempt to build a model to predict the final score of a game given only the participating teams. In a POMDP framework, we "observe" who is playing while the eventual winner (or perhaps the true relative skills of the two teams) remains "hidden". While this approach did not yield the desired result, we are optimistic that using POMDPs for prediction is a realm worth exploring.

1 Introduction

POMDPs were created to model actions selection in uncertain environments, and seemingly all work related to them has stuck exclusively with this initial intention. Searching through the literature around POMDPs, we could not find any applications outside of this domain.

One of our goals with this project was to expand the domain of problems to which POMDPs can be applied. Here, we formulate prediction as a decision process, where the "action" is simply the statement of a prediction and the "state" is the current test data point.

We chose as our application the prediction of basketball game outcomes in the NBA. Our initial goal was to predict a single game score given the teams that are playing each other. Afterwards we switched our focus to predicting only the game winner. As our algorithm walks through the data set (here, the state space), the goal is to learn to make better predictions (here, actions) based on the participating teams (here, our observation) by learning the relative strength of all the teams in the league.

The data used in this project is from stats.nba.com, which is a publicly available website for statistics associated with the NBA. In order to scrape the data, we used a Python library called nba_py (Uriegas, E, 2017).

The data used for this project was also used for Jaak Uudmae's CS229 final project. The work described in this paper was exclusively used for this class. The only overlap between the two projects were the Python code written for scraping and creating the data.

All of the code written for this project is available at https://github.com/jaagu/CS238FinalProject .

2 Description of the Models

We implemented and trained two different POMDP models - one for predicting single game scores and one for predicting the winner of a single game. The following sections explain those models in detail.

In both cases, we define a POMDP and using the Partially Observable Monte Carlo Planning online solver (described Silver, D., Veness, J., 2010). We believed this to be a good choice of algorithm for two reasons.

First, it is effective in environments that have large state spaces. Because our state models the score possibilities of each 30 choose 2 team pairings (including information about who is home/away), and in a given game there is a range of roughly 100 points in which either team's score could fall, our state space is rather large (summing to almost 9 million states). The POMCP solver uses Monte-Carlo sampling to break the curse of dimensionality. This is also a natural construction for our model, as our transition function simply takes a random sample from our dataset.

Second, the algorithm does not require explicit probability distributions, but rather only a black box simulator of the POMDP. Since we only have a list of game scores as our input, using a generative POMDP solver was a natural choice.

To run this algorithm, we used the BasicPOMCP.jl class of Julia's POMDPs.jl.

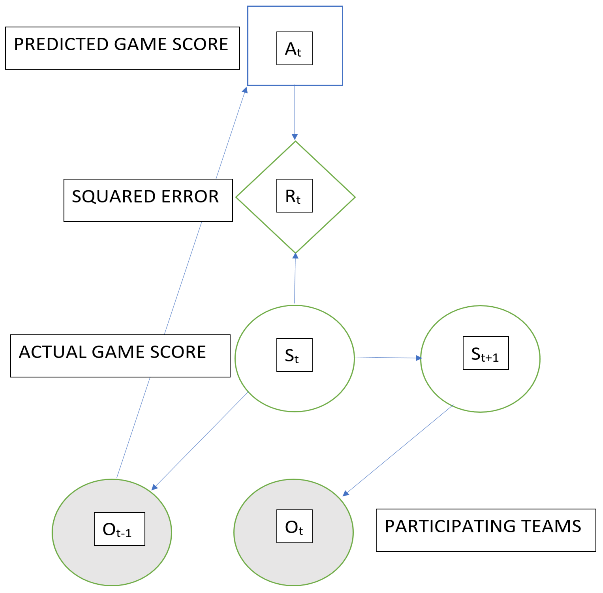

Figure 1: Game Score POMDP

2.1 Model for Game Scores

We first aimed to predict the final score of a game given only the participating teams. Thus, our hidden "state" is the actual final score, while the "observation" is the participating team. As our "action", we output a prediction of the final score. To transition between states, we simply randomly choose another game in the dataset to predict (thus the transition function is independent of the current state, action, and observation, which is uncommon). Formally, we define the following POMDP construction (See also Figure 1):

S: The actual final score of the game t. This is "hidden". The state also encodes which team is playing at "home" and which is "away". Represented as a sparse vector of length 60 and consisting of integers.

A: Our prediction of the final score. Represented as simply two integers.

T: Randomly choose another game in our dataset.

R: Sum of squared error, where the error is the difference between predicted score and actual score for a team.

Ω: The participating teams for the *next* game. We use o_{t-1} to know the state and action spaces for time t. Observation also encodes which team is playing at "home" and which is "away". Represented as a sparse vector of length 60 and consisting of integers.

O: We define the conditional probability of seeing an observation given a state to be uniformly distributed across all possible states, as the next state (and therefore the current observation, which reveals the participants in the next state's game) is chosen randomly.

γ: We defined the discount factor to be 1, as we care equally for every point in our data set (and thus equally about each state/prediction pair, no matter when we look at it).

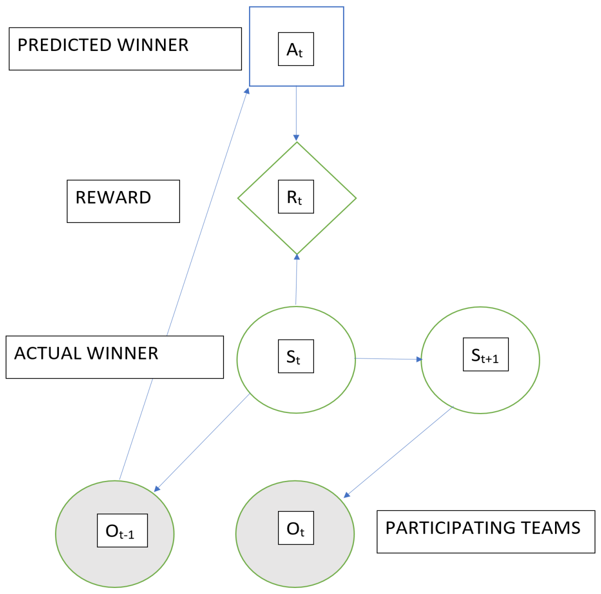

Figure 2: Game Winner POMDP

2.2 Model for Game Winners

After finding little success with the above formulation (described in the results section), we decided to simplify our problem somewhat by only requiring the model to predict the winner of the game, instead of the two scores. The POMDP is largely the same, with the principal difference being that the hidden state now encodes simply the 0/1 winner. Formally (See also Figure 2):

S: The actual winner/loser of the game i. This is "hidden". Represented as a sparse vector of length 60 and consisting of integers.

A: Our prediction of winner. Represented as a single integer, where 1 means the home team won.

T: Randomly choose another game in our dataset.

R: +1 if our prediction is correct, -1 if our prediction is incorrect.

Ω: The participating teams for the *next* game. Represented as a sparse vector of length 60 and consisting of integers.

O: Uniform distribution across all possible states, as the next state is chosen randomly.

γ: We defined the discount factor to be 1.

3 Results

Overall, our results were not what we had hoped for; neither model learned to predict particularly well. Below is an description of the output of both models, followed by a discussion of what might have gone wrong and what can be done in the future.

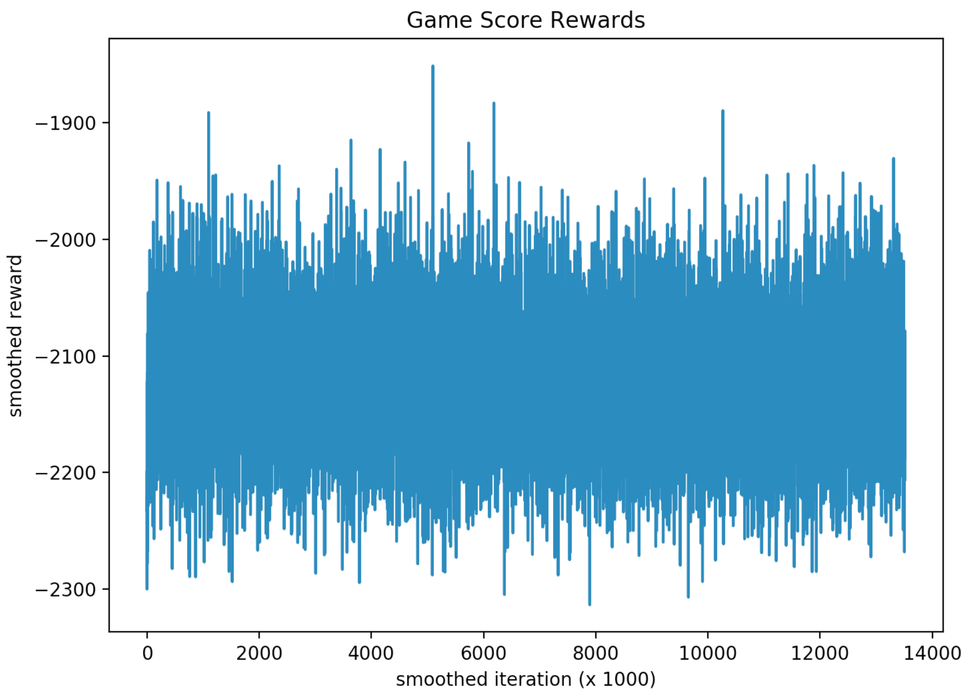

Figure 3: Game Score Rewards

3.1 Results for the Game Score POMDP

Below (Figure 3) is a graph of the rewards over time gained by our POMDP when trying to predict the specific scores of both teams. We ran the code for more than 13 million iterations, and it is evident that there was no upward slope to the rewards, which would have evidenced learning by the algorithm. The sum of squared errors hovers around 2100, which implies that the average difference between predicted an actual scores was around 30 points. This error is of course not negligible.

3.2 Results for the Game Winner POMDP

Below (Figure 4) is a graph of the rewards over time gained by our POMDP when trying to predict the winner of a given game. We ran the code for more than 10 million iterations, and again it is clear that the rewards did not enjoy the telltale upward slope of learning. Because the reward could be either +1 or -1, the average reward hovered around 0, with a very thin band around it (roughly 0.05).

4 Conclusions

These unsatisfying results can likely be attributed to two possible explanations.

The first is the most obvious: using POMDPs for prediction rather than sequential decision making has never been attempted before (to the best of our knowledge), and it is perhaps asking too much for this model framework to be extended to a domain for which is was not designed. It was exciting to

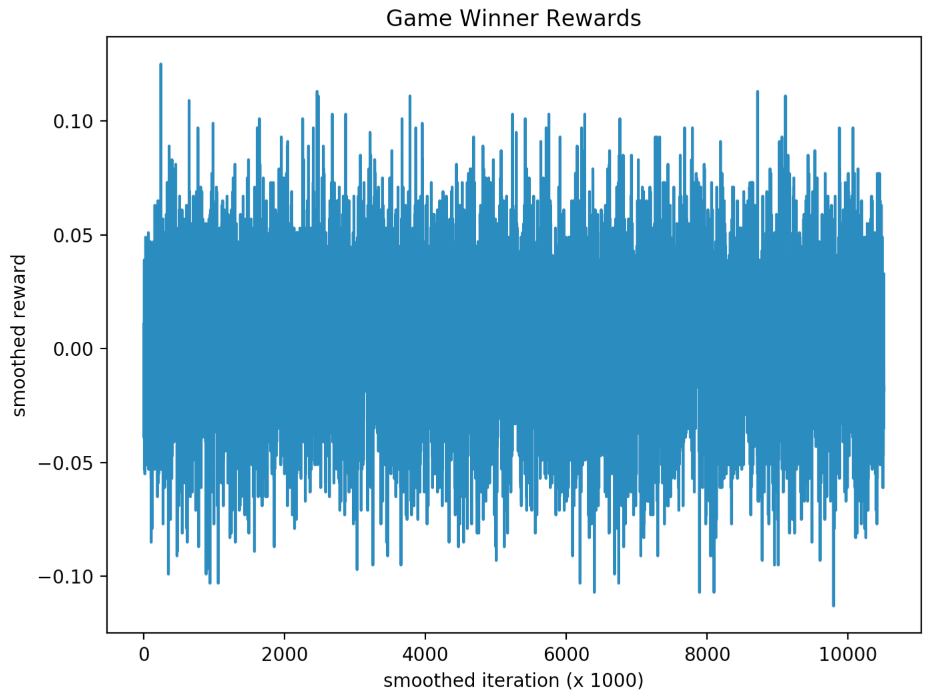

Figure 4: Game Winner Rewards

attempt to expand the horizon of possible applications, but it is possible that it was intrinsically an ill-fated venture.

A second explanation might be a lack of data. We used games between 2013-2017 - a total of 5258 games. Given our state space for the game scores problem, which is in the millions, the number of data samples we have is definitely not enough. We chose to only go back to 2013 because we wanted the data to be indicative of the current strength of the teams; the landscape of the NBA can change rapidly as players and coaches change teams and retire, and so games from the distant past of course can't be used to predict games in the present.

5 Future Work

The problem at hand remains unsolved using POMDPs. For future work, it would be interesting to see if it is possible to reformulate the model in a way that yields successful results. As the POMDP library for Julia is expanded, too, the possibilities for different solutions also increases.

The National Basketball Association collects data actively, and so the corpus of testing data grows daily. There is also a lot of other data available for making predictions related to the NBA games besides just game scores: for example, team overall statistics or individual player statistics. These statistics could be used to model a new POMDP and predict something different than we did (e.g. how many points a given player will score in a game). The more basic problems (like predicting a game winner) should be tackled first, however, before moving onto a these more challenging problems.

Overall, however, regardless of our initial results, this project started a process for expanding the use cases for POMDPs which we find very exciting.

References

[1] Silver, D., & Veness, J. (2010). Monte-Carlo Planning in Large POMDPs. In Advances in neural information processing systems (pp. 2164–2172). Retrieved from http://discovery.ucl.ac.uk/1347369/

[2] Egorov, M., Sunberg, Z., Balaban, E., Wheeler, T., Gupta, J., & Kochenderfer, M. (2017). POMDPs.jl: A Framework for Sequential Decision Making under Uncertainty. In Journal of Machine Learning Research (18.26.1-5, 2017). Retrieved from http://jmlr.org/papers/v18/16-300.html

[3] Uriegas, E (2017). nba_py. Retrieved from https://github.com/seemethere/nba_py