TDL Value Function Approximations in 2048

Owen Siljander

May 14, 2020

University of Minnesota

Abstract

This paper investigates value function approximations for temporal difference learning in the game *2048*. *2048* is a puzzle game released in 2014 that gained popularity among researchers due to it being a discrete time stochastic environment with a substantial state space. Many classical search approaches have been studied under this environment, as well as reinforcement learning agents, with substantial success over human players. TDL is not normally equipped to handle such a large state space and so this paper investigates linear functions of features and neural networks as value function approximations for TDL. Other solutions, particularly the one this study will closely take after, are n-tuple networks for value function approximation as in [2].

The results of this study are mostly inconclusive due to computing and time constraints. Linear functions of features in experiments would often diverge extremely quickly (in less than 10 learning steps) which could either be caused by a poor design in the linear function or indicate that the true value function is not linearly approximable. Experiments with four three-layered non-linear neural networks, with ReLU as an activation function and stochastic gradient descent as an optimizer, were occasionally better than a simple random agent. It is impossible to make a claim about the reason for this due to the insufficient number of games the networks trained for. There is large room for improvements and experimentation in the application of both these value function approximaters in *2048*.

1 Introduction

1.1 Environment

2048 consists of a 4×4 grid starting out with two tiles in random locations with each tile having a value of 2 or 4 (ex. Figure 1.1). The agent has the available moves of up, down,

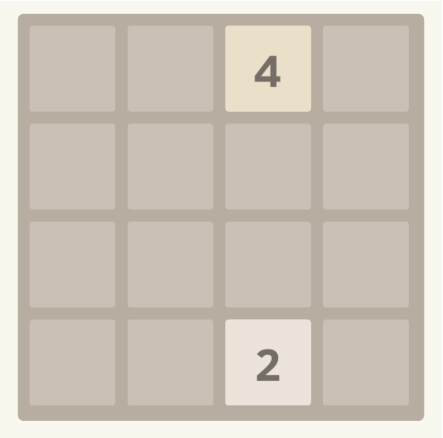

Figure 1.1: Example starting state for *2048*

left, or right. Each of these actions will slide the tiles in the appropriate direction. If a tile collides with or moves into another tile of the same value, they combine into a single tile with a value equal to their sum. A tile which has been produced through such a combination cannot combine with another tile within the same move; that is to say, an agent could not combine a row of 2s into a single tile with a value of 8, it would instead produce two tiles of value 4. If a move would not combine or move any tiles on the grid, it is not a legal move. Each time the agent takes an action, a new tile is generated in a random location having a value of 2 with probability 0.9 and a value of 4 with probability 0.1. The game ends when there is no legal move (i.e. cannot move or combine tiles). For every move, the agent receives a reward equal to the value of all tiles which have been combined (e.g. combining two 2 tiles and two 4 tiles results in a reward of 12). The game is considered to be won when the agent has produced a tile with a value of 2048, though the agent can keep playing to achieve higher scores.

The reason this environment is challenging is not only due to the stochastic nature of the game, but also due to the large state space. Considering each tile's value is a power of two, and the maximum possible tile value (in practice) is 2^{17}, there are roughly $18^{16} \approx 10^{20}$ possible states (a power of 18 to include the zero/empty tile) [2].[1] This state space increases greatly when you consider state-action pairs. This large state space has made sampling algorithms, such as Monte-Carlo tree search (MCTS), particularly attractive solutions.

[1] [2] calculates the number of states to be $16^{18} = 4.7 \times 10^{21}$ but I believe this calculation to be erroneous.

1.2 Requisite Knowledge

In this paper we focus on the application of temporal difference learning (TDL) in a Markovian decision process (MDP). Temporal difference learning is a particular class of model-free reinforcement learning algorithms that use a value function, which calculates the expected utility of a state, as a means for decision making. By determining the value of potential next states, an agent can use the value function to choose the best action to take. TDL works by continually adjusting its estimated value function, $V^\pi(s)$, under some policy π. One of the simplest forms, commonly referred to as TD(0) (the 0 denotes a one step look-ahead), in which the value function is updated as follows:

$$V^\pi(s) \leftarrow V^\pi(s) + \alpha(r + \gamma V^\pi(s') - V^\pi(s))$$

where $\alpha \in [0, 1]$ is some learning rate, γ is the discount factor, s' is the state following an action from s under the policy π, and r is the reward received from such an action.

This paper uses the formal definition of an MDP as found in [2], it is repeated here for convenience and coherence. An MDP is a tuple $\langle S, A, R, P \rangle$ where

- S is a set of possible states
- A is a set of actions, $A(s)$ denotes the possible actions at state s
- $R : S \times A \to \mathbb{R}$ is the reward function that maps a state-action pair to real number
- $P : S \times A \times S \to \mathbb{R}$ is a stochastic transition function where $P(s, a, s')$ denotes the probability of transitioning from state s to s' with an action a.

In some cases, it may be possible to express the reward function explicitly, e.g. as a lookup table. In the case of *2048*, the state space makes this solution infeasible. Instead, TDL can be modified to use an approximation of the value function. In some particular cases, it may be more appropriate to enable decision making through a function of state-action pairs (as opposed to just states); this is an extension of TDL known as Q-learning. Q-learning uses a function $Q(s, a) : S \times A \to \mathbb{R}$ that can discriminate more accurately against actions.

1.3 Related Work

More classical approaches have been taken to create agents for this game, namely Minimax, Expectimax, and Monte-Carlo tree search (MCTS). For instance, [**?**] found that averaged depth-limited search (ADLS) and MCTS can exceed average scores of 50,0000 and 120,000 respectively. However, they found that in particular, these agents worked quite well for maintaining high average scores but failed to produce large individual scores. Specifically, an increase in the search limit for ADLS would reduce the number of low-scores, but would also reduce the number of high-scores.

[2] used temporal difference learning (TDL) with an n-tuple network to create an agent for this game that performed exceedingly well. In over 97% of plays, the agent reached the 2048 tile and had an average score of $100,178$ and a maximum of $261,526$. The additional benefit of this technique is that it can make decisions far quicker than the more classical approaches.

In particular, [2] found that both Expectimax and Minimax performed exceedingly slower. The implementation of Expectimax is capable of 6.6 moves per second compared to 330,000 moves per second by TDL. Nonetheless, TDL still suffers from the same issue as the agents in [**?**], it struggles to reach large scores. This is pointed out in [**?**], which proposes an improvement upon the implementation of TDL by [2]. This extension is called multistage temporal difference learning (MS-TD). The authors equate it to hierarchical reinforcement learning methods. In essence, MS-TD breaks up the training process into three distinct stages, each containing their own weights. The intuition here is that each stage presents a stronger (and different) challenge and is not as forgiving with the agent's actions as it would be in the early stages. The intuition indeed turned out to be correct, this extension increased the rate at which the agent achieved higher scores. Further improvements were made upon this to reinforce it with expectimax to yield even higher rates of reaching high-valued tiles.

1.4 Motivation and Objective

2048 can be modeled as a Markovian Decision Process (MDP); this is a framework to facilitate decision making in a (discrete) time stochastic environment. The motivation to select TDL for this study is largely arbitrary but is also due to the fact that it is a model-free reinforcement learning solution to decision making in such an environment.

While there is a fair amount of research on reinforcement learning and TDL for *2048*, very few of these studies with TDL appear to discuss ways in which to combat the large state space. TDL is not normally equipped to handle such large state spaces and so this paper focuses on particular technique known as value function approximation. Of the various techniques for this, this paper will consider linear combinations of features and neural networks. The main objective for studying these techniques is to discover the efficacy of value function approximations other than the n-tuple networks used in [2].

2 Methods

While it is possible to use plain TDL, this paper uses an extension from [2] that enables TDL to behave quite like Q-learning. Q-learning is a variant upon TDL that considers a state-action value function, as opposed to just a state value function in TDL. This should increase the accuracy in which the value function approximations can classify the expected utility of states. For both variants of the value function approximations, there is a distinct approximater for each action (left, right, up, down) denoted V_a for the respective action $a \in A$. This extension comes from [2]; this extension is included in this study to reduce the number of variables which may cause the results to differ, allowing a more direct comparison of results. This extension enables TDL to behave like Q-learning without needing an explicit approximater for a state-action value function.

2.1 Linear Function of Features

In direct similarity to [2], we update the value function approximation according to Algorithm 1.

ALGORITHM 1: Update for Value Function Approximation

`Learn-Evaluation`(s, a, r, s', s'')
$v_{next} \leftarrow \arg\max_{a' \in A(s'')} V_{a'}(s'')$
$V_a(s) \leftarrow V_a(s) + \alpha(r + v_{next} - V_a(s))$

The state succeeding an action is separated into two distinct states: s' and s''. s' denotes the state after an action has taken place but before a tile is randomly generated. s'' denotes the state after an action and a random tile has been generated. This design comes from [2], though it is not utilized here.

The value function approximater takes form as

$$V_a(s) = \sum_i w_{ai}^T f_i(s)$$

where $\boldsymbol{w}_a$ is a weight vector for the value function approximater for action a. The feature vector, $\Phi(s) = \langle f_1(s), f_2(s), \ldots, f_n(s) \rangle$, is simply the value of each game tile (e.g. $f_1(s)$ would be the value of the first tile). The motivation for selecting this is that a higher weight would be given to high value tiles being held in the corner and in adjacent tiles. That is, keeping large valued tiles outside of a corner or small valued tiles in a corner, would contribute little and would be biased against by the value function approximation. This is a common heuristic used to evaluate the quality of states in search based algorithms.

2.2 Neural Networks

Just like in the previous section, four neural networks are used, one for each possible action. For reasons that will be apparent later, the activation function chosen for the neural network is ReLU. The optimizer is stochastic gradient descent and each network has three layers: a 16 neuron input layer, a middle layer with 4 neurons, and an output layer with one neuron. The choice for the number of neurons in the middle layer comes from two factors: symmetry of the game board, and a common game heuristic. A common strategy for *2048* is to keep the highest valued tile in a corner. The symmetry of the game board means it does not matter which corner this high valued tile is in. For this reason, four neurons in the middle layer should pick up on an organization of tile values within the four four-tiled corners of the game board.

3 Results

Results from the experiment were largely inconclusive, and as such, there is very little data worth being presented here. The linear combination of features performed quite poorly and diverged quickly. Several trials were conducted, both with the weight vectors initialized randomly and initialized to zero. In all trials, The functions associated with the left and up moves quickly approached the maximum integer value of 2^{32} and ended up causing an integer overflow. Interestingly, this type of weighting coincides with a common strategy of keeping the high value tile in a corner; which is achievable by repeatedly moving up and

left and occasionally right or down as needed. There is not too much to be claimed about this particular result though it suggests a more clever feature combination may reveal such a strategy is learned by TDL. The reason for inconclusiveness is that it is not guaranteed that the linear combination of features is even able to approximate the true value function [1]. It is most probably an erroneous implementation that caused these results to be inconclusive and divergent.

For the neural networks, these performed quite poorly as well but not much can be claimed since the number of games used in training was less than a thousand. ReLU was chosen as the activation function for the neural networks to introduce non-linearity into the function approximation. This was largely motivated by the results for the linear combination of features being quite poor. Training the neural networks adequately would have taken on the order of weeks and was thus not feasible given time constraints. However, testing the neural networks decision making speeds proved somewhat illuminating. In comparison to [2], which found a game could be completed in 23ms, the neural networks often took 20+s to finish a single game. Unfortunately, the hardware differences are unknown and thus it is impossible to make a claim about the efficacy of neural networks either.

4 Conclusion and Future Work

The data from experiments was very minimal and not meaningful due to lack of training. Not much can be claimed about the results from the linear functions, despite the experiments running successfully. The same holds true for the implementation of neural networks as function approximaters, though it is difficult to make claims about the efficacy of this solution due to potential hardware differences.

For future improvements, it would be important to take a closer look at linear function approximaters since the results here suggest something of interest, though it is not concrete.

References

[1] J Zico Kolter and Andrew Y Ng. Regularization and feature selection in least-squares temporal difference learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 521–528, 2009.

[2] Marcin Szubert and Wojciech Jaśkowski. Temporal difference learning of n-tuple networks for the game 2048. In *2014 IEEE Conference on Computational Intelligence and Games*, pages 1–8. IEEE, 2014.