QuackTD

Study on Implementation of Backgammon-playing Systems

Abstract

In this project, we develop a backgammon playing system trained by self-play using the TD(0) algorithm. The system is inspired by TD-Gammon, originally developed by Gerald Tesauro in the 1990's.

We experiment with varying how the backgammon board is represented to the neural network and varying the learning rate used in training the neural network.

We find that there is a noticeable difference between the performance for the board representations tested. The truncated unary TD-Gammon-based board representation performed better than all variations of the counting-based Quack representations.

Group

Anders Brunsgaard Ladefoged
<anders@bladefoged.com>

Christoffer Müller Madsen
<christoffer@guava.space>

Alexander Munch-Hansen
<alex@pwnh.io>

Supervisor

Panagiotis Karras
<panos@cs.au.dk>

Board representations

We have evaluated multiple board representations. They can be partitioned into two families:

The Quack board representations are based on integer checker counting. Each point on the backgammon board is represented by a single integer denoting the colour and amount of checkers on a point by the sign and magnitude of the integer. This leads to a compact board representation.

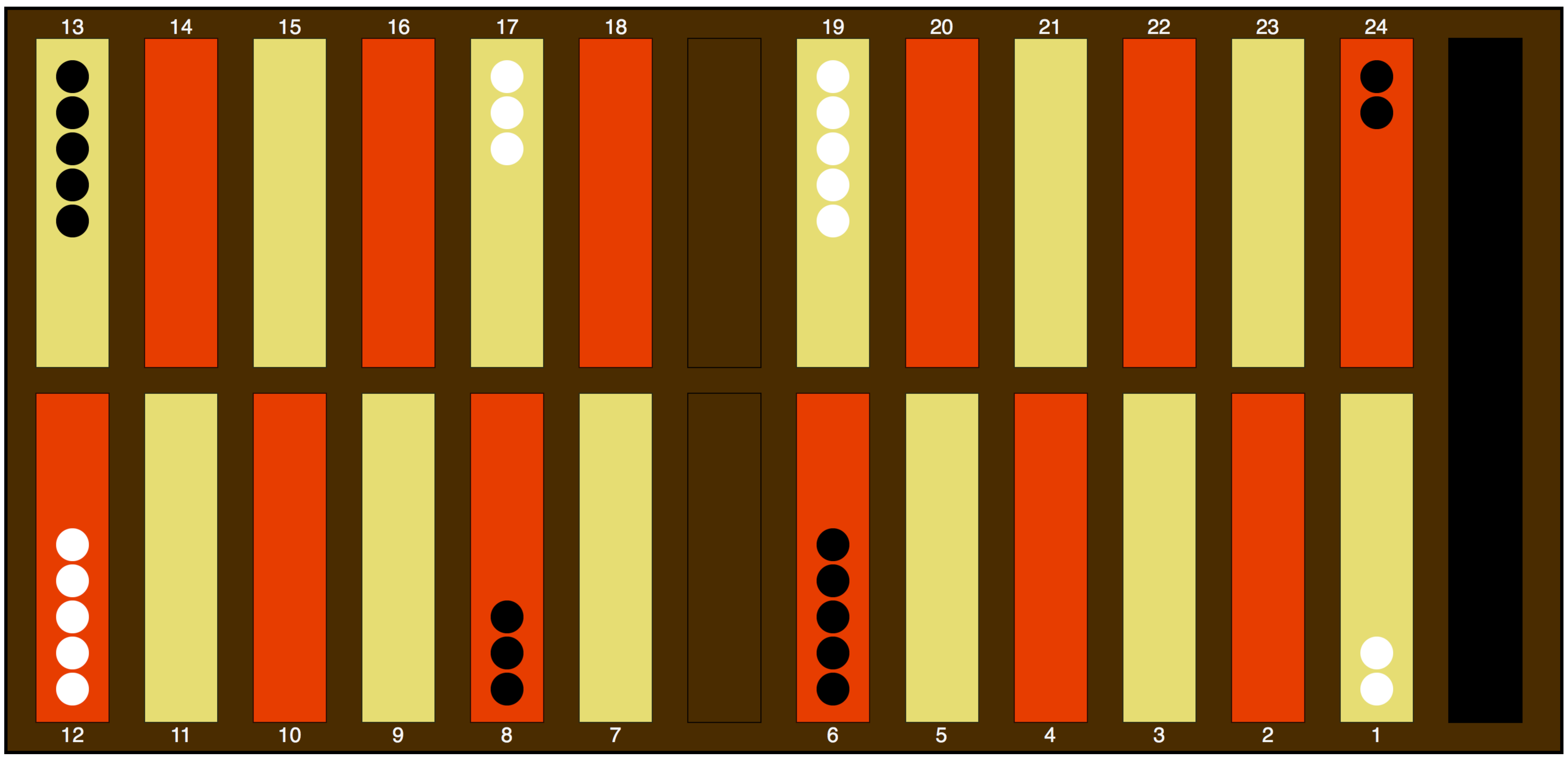
The TDG board representations are based on the representation used in TD-Gammon. In contrast to the Quack board representations, each point is represented by multiple values. Every point is transformed for using the transformation function shown below. This is done for both players, and the results are concatenated into a single sequence.

The basic Quack representation consists of 26 integer valued fields representing all points and the two bars of the board. QuackNorm is an augmentation of Quack with values denoting the amount of checkers borne off the board for each player. QuackNorm is QuackAug with normalised values.

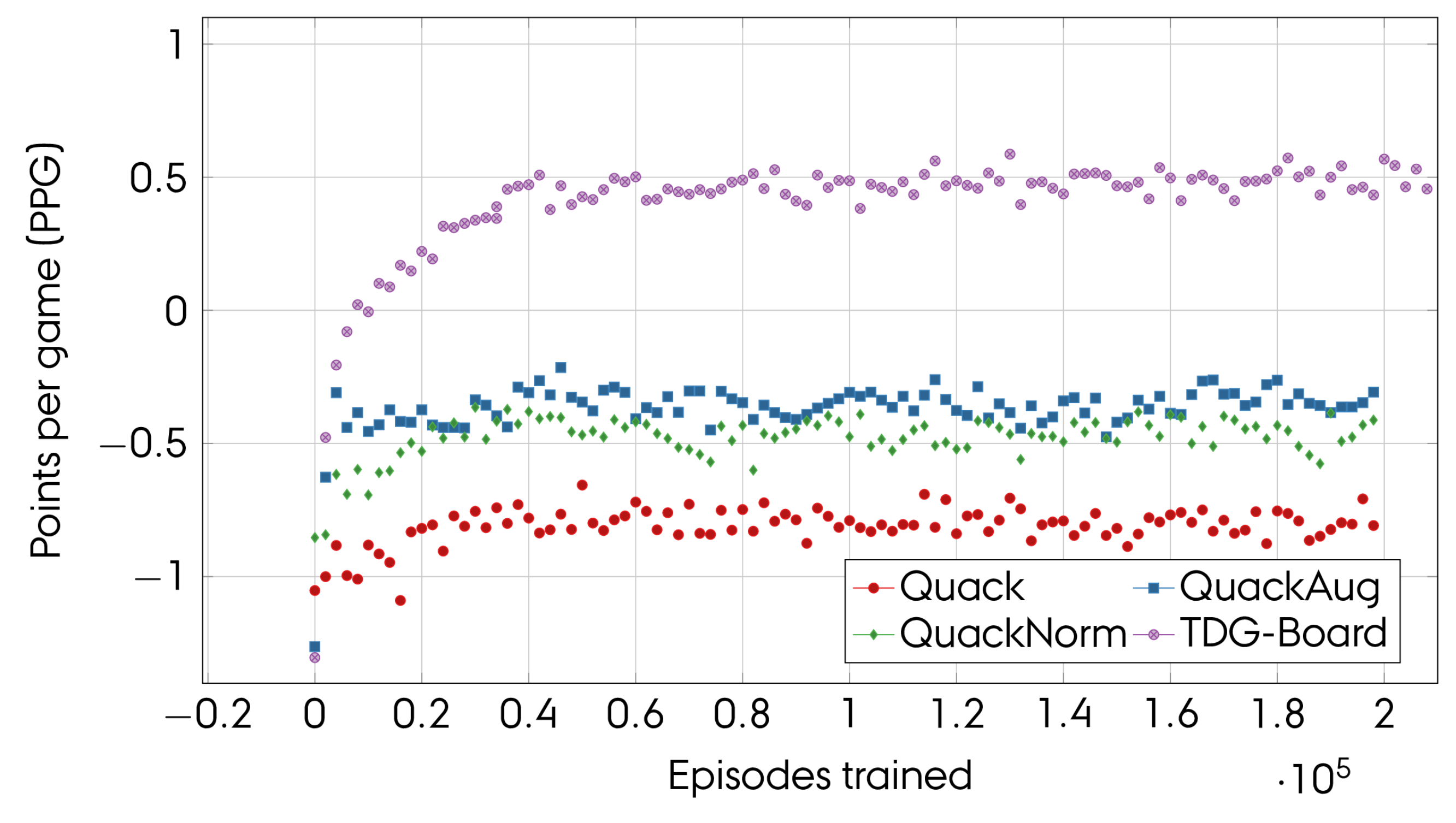
TDG-BoardAug uses a board representation truncated at index 10 instead of the index of 3 used in TDG-Board.

$$
t(x) = \begin{cases} (0,0,0,0) & \text{if} \quad |x| = 0 \\ (1,0,0,0) & \text{if} \quad |x| = 1 \\ (1,1,0,0) & \text{if} \quad |x| = 2 \\ (1,1,1,0) & \text{if} \quad |x| = 3 \\ (1,1,1,\frac{x-3}{2}) & \text{if} \quad |x| \geq 4 \end{cases}
$$

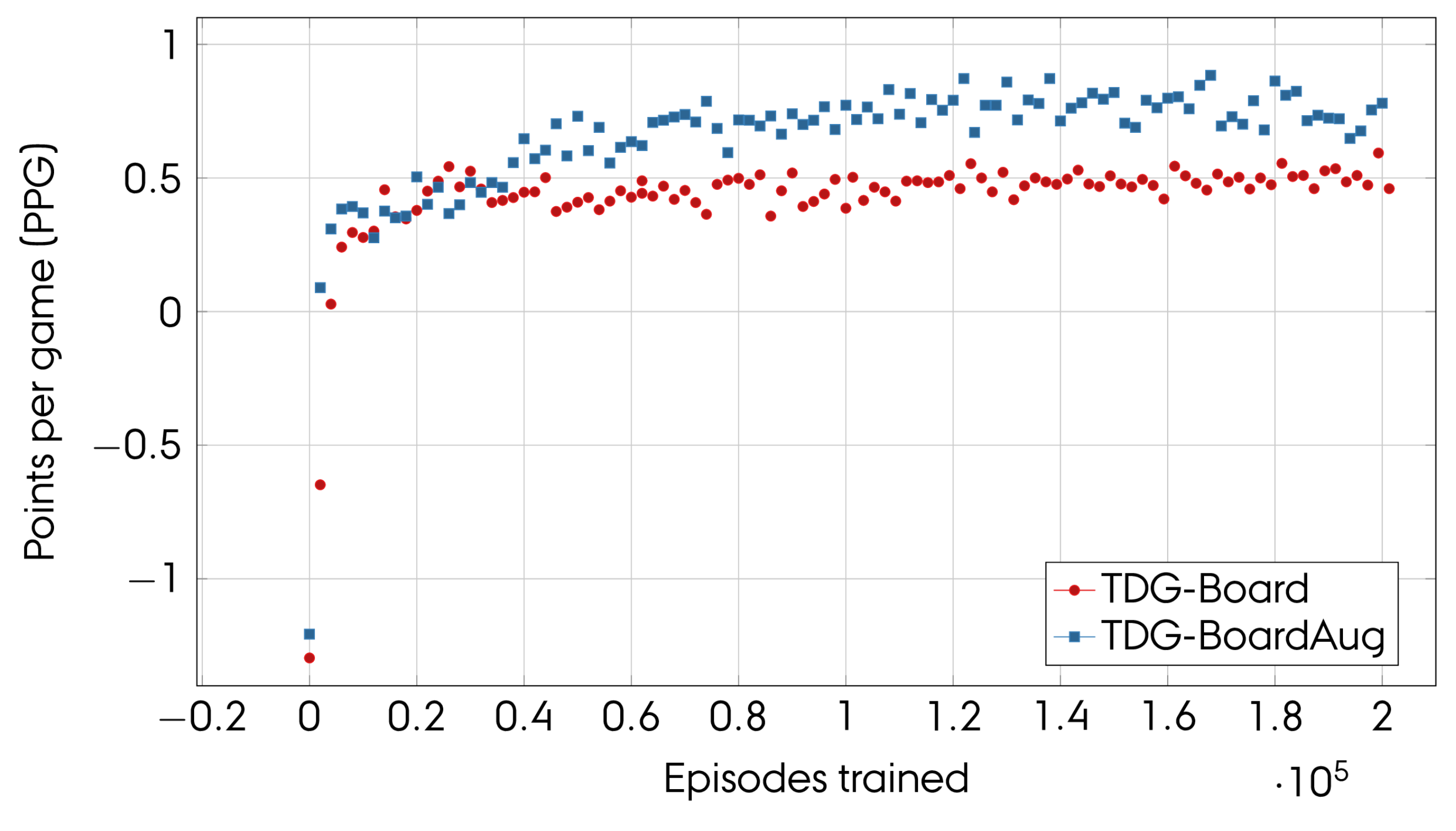
The transformation function used in the TDG board representations. Each point (x) is transformed using this function.



Backgammon board



Comparison between evaluation scores of the board representations Quack, QuackAug, QuackNorm, and TDG-Board.



Comparison between evaluation scores of using TDG-Board and TDG-BoardAug.

TD learning

Monte Carlo and TD methods can be used for learning even if knowledge of the environment is incomplete. These methods learn only from experience which consists of sample sequences of states, actions, and rewards. This enables learning using only transitions generated from the simulations based on the model as opposed to the complete probability distribution of all possible transitions.

In the game of backgammon (and many other cases) it is infeasible to obtain the complete distributions in explicit form due to the incomplete knowledge of the environment—in backgammon the incomplete knowledge is the sequence of dice rolls and the subsequent states.

Monte Carlo methods sample and average returns for state-action pairs. The value estimation in these methods is based on the observation that the value of a state is its expected return. Thus Monte Carlo methods predict by averaging the returns observed after encountered states. With the number of observations increasing the average will converge to the expected value. Monte Carlo methods update their estimate of the value function only after the terminal time step.

In contrast to MC methods, TD methods will update their value estimate after every time step of a game using the observed reward and the value estimate.

The advantage of TD methods compared to MC methods is the fully incremental nature of the update. The incremental nature yields smaller memory requirements and lower peak computation. Another advantage of TD methods is that they have shown to converge faster than MC methods in practice.

Evaluation framework

We implemented the game of backgammon in Python3 in order to provide an easy-to-use backgammon interface for training and evaluation purposes.

The neural network is implemented using TensorFlow, which is a library providing optimised functions for typical neural network operations.

Training

The training process consists solely of the network playing against itself in multiple episodes. Each episode proceeds as follows:

1. Decide randomly which player gets the initial turn.
2. The neural network evaluates all possible states resulting from the player and dice roll, and the best move is chosen:
   a) Let the neural network evaluate the value estimate of all possible states.
   b) Pick the state with the maximum value estimate if playing as the white player; otherwise pick the state with the minimum value estimate.
3. The TD error is backpropagated to adjust the weights.
4. If a termination state has been reached, do final backpropagation and start a new episode. Otherwise, return to step 2.

Results

We have experimented with different board representations to see the performance impact resulting from these. We found that the truncated unary TD-Gammon-based board representation outperformed all variations of the counting-based Quack representations. Furthermore, we found that an extension of the original TD-Gammon-based board representation with additional fields had a positive impact on performance against the evaluation opponent.

Additionally, we have experimented with different learning rates for the training of the neural networks in our algorithm. We found that there is a positive correlation between learning rate and evaluation performance (with a limit). Higher learning rates, however, introduce an erratic behaviour in the evaluation scores with no positive impact. This erratic behaviour can be tamed by using an exponentially decaying learning rate without sacrificing game-play performance.