From Heuristics to Neural Nets: Backgammon AI Agents

Mark Shperkin
Department of Computer Science and Engineering
University of South Carolina
Columbia, SC 29208
CSCE 775: Deep Reinforcement Learning
05/05/2025

***Abstract*— Backgammon is one of the world's oldest two-player board games, originating in Mesopotamia over 5000 years ago and later spreading through Persia, Rome, Byzantium, and medieval Europe. Today, it remains deeply embedded in Middle Eastern culture. From an artificial intelligence perspective, Backgammon presents a challenging reinforcement learning problem due to its vast state space, on the order of 10^20 possible configurations, far exceeding that of checkers or chess. Exhaustive table-lookup methods are therefore impractical for full-game play. In this work, we implement and evaluate a spectrum of decision-making algorithms, including minimax search with α–β pruning, Temporal Differences, and Monte Carlo. Through empirical comparison on single-move evaluation tasks, we demonstrate that Temporal Differences with eligibility traces achieves the highest predictive accuracy, outperforming traditional minimax, Monte Carlo, and other temporal-difference approaches. These findings highlight the effectiveness of simulation-based strategies in tackling combinatorially complex games such as Backgammon.**

Keywords*—*Backgammon; Reinforcement Learning; Monte Carlo; Minimax Search; Temporal Differences; Neural Networks

I. Introduction

Backgammon is an ancient, two-player, zero-sum board game that combines deep strategic planning with stochastic elements introduced by dice rolls. From an artificial intelligence standpoint, Backgammon presents a unique challenge: its enormous state space, in the order of 10^20 possible configurations, renders exhaustive search infeasible, and the randomness of dice outcomes demands decision-making under uncertainty.

Programming an agent to play Backgammon at a high level therefore requires balancing long-term strategy with probabilistic evaluation of chance events. Simple brute-force approaches, even when augmented by shallow heuristic functions, struggle to capture the nuanced tactics employed by expert players, even a two-ply lookahead can take minutes to evaluate in practice. This project seeks to develop and compare a suite of algorithms, ranging from minimax search with α–β pruning to Monte Carlo simulation and temporal-difference learning to determine which methods best navigate Backgammon's combinatorial complexity. Achieving strong performance in this domain not only advances the state of reinforcement learning in stochastic environments but also demonstrates the potential for AI to learn sophisticated game strategies that rival human expertise.

II. Related Work

Neurogammon [1] pioneered supervised neural-network approaches by training multiple feedforward networks via backpropagation on large expert game databases. Six distinct networks, each responsible for different phases of play, were taught to compare and rank final board states using a pairwise "comparison" training paradigm. This architecture enabled Neurogammon 1.0 to secure a flawless 5–0 victory at the First Computer Olympiad, demonstrating the power of learned positional evaluation over handcrafted heuristics. However, the reliance on phase-specific networks and expert-labeled data yield a complex ensemble that can be cumbersome to train and maintain.

TD-Gammon [2] advanced the field by introducing a single multilayer perceptron trained entirely by temporal-difference learning (TD(λ)) through self-play. Starting from a raw board encoding, the network autonomously discovered elementary strategies (e.g., hitting, blocking) and, when augmented with Neurogammon's handcrafted features, achieved master-level performance, coming within a few hundredths of a point per game against human grandmasters. Nevertheless, the original TD-Gammon reports omit critical hyperparameter settings, such as the λ decay schedule, learning-rate, state representation methods, forcing extensive experimentation to reproduce and optimize performance.

Building on these foundations, we implemented and trained a TD (0) agent using a multilayer perceptron (MLP). The state representation is fed into the network, which outputs an approximate value function $V(s;\theta)$ for move selection. In parallel, we trained an Monte Carlo (MC) agent under the same architecture and hyperparameter schedule. In exhaustive head-to-head evaluations, both TD (0) and MC not only surpassed all heuristic baselines but outperformed by the TD (λ) variant. Direct replication of the original TD-Gammon results [2] remained infeasible. However, by independently tuning these parameters, our agents achieved win rates comparable to those reported in [2], thereby validating the effectiveness of deep

function approximation for Backgammon value estimation and showing the ability to model Backgammon's stochastic dynamics.

III. Approach

A. Expected Minimax with α–β Pruning:

In this work, we model Backgammon move selection as an expected minimax search problem augmented with α–β pruning to manage the game's stochastic decision tree. The search alternates between MAX nodes where the agent selects the highest-utility successor of agents moves by following

$$V(s)=\max\{a\in Moves(s)\}\ h(s),$$

MIN nodes where the agents select the lowest-utility successor of opponent moves by following

$$V(s)=\min\{a\in Moves(s)\}\ h(s),$$

and CHANCE nodes where the computed weighted average over all distinct dice outcomes by following

$$V(s)=\sum r\ P(r)\ h(s).$$

To reduce computation cost and reduce tree explosion, at each MAX or MIN node, α and β bounds are propagated down the tree branches whose potential values cannot influence the final decision are pruned, significantly reducing the number of evaluated positions without sacrificing optimality.

We employ a handcrafted heuristic function h(s) that approximates strategic value. For each player p, let Dp be the sum of pip-distances remaining to bear off, P¯p the amount of checkers on the bar, Bp the count of borne-off checkers, and Sp the number of locations occupied by two or more checkers. We define

$$hp(s)= Dp + 30P\bar{}p - 30Bp - 3Sp,$$

and the overall value at state s as

$$h(s) = \{h\_black(s) - h\_white(s), \text{ if black to move.}$$

$$\{h\_white(s) - h\_black(s), \text{ if white to move.}$$

This scoring captures both progress toward bearing off and board control, yielding positive values when the current player holds the advantage.

Finally, we restrict the expected minimax search to two plies (one move per player) plus a subsequent CHANCE node. This shallow-depth strategy, together with α–β pruning, reduces per-move computation from tens of minutes toward just a few minutes. As the game advances and the branching factor shrinks, the computation time decreases, producing complete games in roughly 20 minutes. By balancing selective deepening with a robust heuristic and stochastic sampling, our approach achieves a tractable yet strategically informed agent for Backgammon play.

B. Temporal-Difference Learning Variants:

Temporal-difference (TD) methods form a class of model-free reinforcement-learning algorithms that learn value estimates by bootstrapping, that is, by updating predictions based in part on other learned predictions rather than waiting for final outcomes.

To leverage the generalization capacity of function approximation, we parameterize V(s) with a single multilayer perceptron network. Each board state is encoded as a 28-dimensional input vector, where each entry corresponds to a point on the Backgammon board, the signed checker count (positive for White, negative for Black) is normalized by dividing by 15 (the maximum amount of checkers per point). The network comprises a single hidden layer of 80 units and an output layer of four units, all with sigmoid activation function producing a vector

$$[pW\_reg, pW\_gammon, pB\_reg, pB\_gammon],$$

where pW_reg and pW_gammon estimate the probabilities of regular win or gammon (where the player borne off all of his checkers and his opponent did not bore off any) for white, and similarly for black. All output lies in [0,1] via a sigmoid normalization.

During training, self-play generates sequences of state transitions and rewards, which drive the TD updates of the network's weights. Terminal outcomes are encoded as one-hot vectors, for example, a regular White victory is [1,0,0,0]. This vector serves as the target rt+1 at the terminal state.

In our TD (0) implementation, the value network is updated at every step via one-step temporal-difference bootstrapping. During self-play, each transition (st, rt+1, st+1) is evaluated by forwarding st and st+1 through the network to obtain V(st) and V(st+1). We set the immediate reward rt+1 = 0 for nonterminal moves and use the one-hot terminal outcome vector at game end. The temporal-difference error

$$\delta t = rt+1 + \gamma\ V(st+1) - V\ (st),$$

measures the discrepancy between the current estimate and the observed successor value. Network weights θ are then adjusted by stochastic gradient descent, using the update rule

$$\theta \leftarrow \theta + \alpha\ \delta t\ \nabla\ \theta\ V(st;\theta),$$

where α is the learning rate set to 0.001, γ, the discount factor is set to 0.99, and used a ε-greedy policy where ε was set to 0.1 and reduced at episode 4000 to 0.01. Because TD (0) does not employ eligibility traces, each weight update depends solely on the most recent transition. Iterating this process over thousands of self-play episodes drives the network toward accurate value predictions across the Backgammon state space.

In our TD (λ) implementation, we extend the one-step bootstrapping of TD (0) by incorporating eligibility traces to distribute credit across multiple preceding states. At each time step t, we compute the standard TD error

$$\delta t = rt+1 + \gamma\, V(st+1) - V(st),$$

then update each network parameter θ using an accumulating trace et defined by

$$et = \gamma\, \lambda\, et-1 + \nabla\, \theta\, V(st;\theta),$$

where $\lambda \in [0, 1]$ is the trace-decay parameter. The weight update becomes

$$\theta \leftarrow \theta + \alpha\, \delta t\, et.$$

Eligibility traces enable multi-step credit assignment, allowing a single TD error to adjust not only the current state's estimate but also earlier states in proportion to their recent influence. This mechanism accelerates learning by smoothing updates and reducing variance compared to TD (0). We preserve the same learning rate α=0.001 and discount factor γ=0.99 as in our TD (0) experiments and set λ=0.8 to balance bias and variance in the trace accumulation.

C. Monte Carlo Learning:

Monte Carlo (MC) methods estimate the state-value function V(s) by averaging full-episode returns rather than bootstrapping from intermediate predictions. We reuse the same multilayer perceptron architecture and hyperparameters described in Section III B. During self-play, for an episode of length T, we record all states and reward (st, Gt). let {r1,r2,…,rT} denote the sequence of reward vectors, where the terminal outcome is the one hot encoding reward is ek

$$rt = \{0, \text{for } t = 0, 1, \dots, T - 1,$$

$$\{ek, \text{for} \quad t = T,$$

Then the return is simply GT = rT = ek at time T, and GT−1 = γ rT = γ ek at time T-1.

This gives us

$$Gt = \sum(\text{from } i = t+1 \text{ to } T)\ \gamma\text{^}(i - (t + 1))\ ri$$

And the update rule

$$\theta \leftarrow \theta + \alpha\, [Gt - V(sT;\theta)]\, \nabla\, \theta\, V(sT;\theta).$$

We perform a single backward pass per episode, the network's outputs for all visited states (st, Gt) in the recorded episode are regressed toward their true returns. By deferring weight updates until episode end, MC learning eliminates the bootstrap noise inherent in the TD (0) method, resulting in more stable gradient signals.

IV. Experimental Results

Our experimental evaluation followed a round-robin format, in which each agent played 1000 games against every other agent. This design ensures broad coverage of opponent behaviors and mitigates the impact of stochastic outcomes on any single pairing.

We implemented three baseline methods for comparison and better evaluation of agents learning delicate strategies of Backgammon**.**

- Random agent: selects a uniformly random action a in state s.
- Furthest-First agent: selects action a in state s that has the largest distance from bearing off.
- Closest-First agent: selects action a in state s that has the smallest distance from bearing off.

We found that Furthest-First agent outperformed all the other baseline agents and Closest first places last, giving us a good variety of baseline agents, with different levels of ply as shown in Table 1.

	Vs. Random	Vs. CFA	Vs. FFA	Wins	Rank
Random		393	431	0	3rd
CFA	607		368	1	2nd
FFA	659	632		2	1st

Table 1 shows 1000 games of each baseline agent against other baseline agents.

In our experiments, we evaluated a range of hyperparameter configurations by varying both the number of training episodes and the network depth. Model identifiers follow the convention

MvNe_E,

where

- M ∈ {MC, TD0, TDL} denotes the learning algorithm
- v ∈ {v1, v1} indicates the number of hidden-layers architecture
- e signifies the use of an ε–greedy exploration policy
- E specifies the total number of episodes, complete self-play games used for training.

For example, TD0v1e_10000 corresponds to a TD (0) agent with one hidden layer trained under an ε–greedy policy for 10000 full games.

For our reinforcement-learning agents, we tested two multilayer perceptron architectures. The primary network (Section III B.) uses a single hidden layer, with an input of 80 units while the alternate design employs two hidden layers of 60 and 30 units, respectively. Both architectures were trained using TD (0). As shown in Table 2, the single-hidden layer network outperformed the deeper variant.

V1 vs. V2	Wins	Rank
V1	4565	1st
V2	4435	2nd

Table 2 shows 1000 round robin V1 versus V2 architectures over all sum wins over three different variants of TD (0).

To evaluate the effect of training duration, each algorithmic variant was trained for 4000, 10000, and 35000 self-play episodes (complete games). This range was selected to determine whether extended exposure to Backgammon's enormous state space, exceeding 10^20 unique configurations, yields consistent policy improvement. After each training, we measured the variant's win rate in round-robin matches against all other variants. The resulting performance trends are shown in tables 3-5.

TD (0) v1 e Evaluation	Vs. FFA	Vs. 4000	Vs. 10000	Vs. 35000	Wins	Rank
FFA		477	489	349	0	4th
E = 4000	523		504	343	2	2nd
E = 10000	511	496		372	1	3rd
E = 35000	651	657	628		3	1st

Table 3. 1000 games for each TD (0) variant, each trained with a different number of episodes.

TD (λ = 0.8) v1 e Evaluation	Vs. FFA	Vs. 4000	Vs. 10000	Vs. 35000	Wins	Rank
FFA		338	397	376	0	4th
E = 4000	662		546	476	2	2nd
E = 10000	603	454		427	1	3rd
E = 35000	624	524	573		3	1st

Table 4. 1000 games for each TD (λ) variant, each trained with a different number of episodes.

MC v1 e Evaluation	Vs. FFA	Vs. 4000	Vs. 10000	Vs. 35000	Wins	Rank
FFA		515	530	456	2	2nd
E = 4000	485		460	348	0	4th
E = 10000	470	540		432	1	3rd
E = 35000	544	652	568		3	1st

Table 5. 1000 games for each MC variant, each trained with a different number of episodes.

Based on Tables 3–5, extended self-play yields clear gains, but with some algorithm-specific twists. For all three learning methods, the 35000-episode variant consistently ranks first, each secures all three possible head-to-head "wins" and posts the highest raw match counts. By contrast, shorter runs show mixed results, in TD (0), the 10000-episode model actually underperforms the 4000-episode version, hinting at transient instability during mid-training. In TD (λ), a similar dip occurs at 10000 episodes before recovering at 35000. Similarly, the Monte Carlo learner must run the full 35000 games simply to eclipse the Furthest-First heuristic. These trends confirm that more episodes generally refine policy quality, driving higher win rates and stronger rankings, but also underscore that the shape of the learning curve (and the point of diminishing returns) varies with each algorithm's use of bootstrapping, eligibility traces, and exploration–exploitation balance.

After extensive evaluation, we selected the 35,000-episode variants from each algorithmic family and evaluated them head-to-head. As shown in Table 6, the TD (λ) agent outperforms both its TD (0) and Monte Carlo counterparts, demonstrating superior capacity to navigate Backgammon's enormous state space and capture its subtle strategic nuances.

TD (0) vs. TD (λ) Vs. MC	TD (0)	TD (λ)	MC	Wins	Rank
TD (0)		455	614	1	2nd
TD (λ)	545		616	2	1st
Mc	386	384		0	3rd

Table 6. 1000 games for each variant, each trained with 35000 episodes.

V. Conclusion

In this study, we have implemented and systematically compared a suite of Backgammon AI agents, ranging from heuristic baselines and shallow minimax search to deep-learning methods including Monte Carlo (MC), TD (0), and TD(λ), all realized with a unified multilayer perceptron architecture. Our experimental evaluation, conducted under a round-robin regime of 1000 games per pairing and across multiple network depths and training durations (4000, 10000, and 35000 episodes), demonstrates that TD (λ) consistently achieves the highest win rates. Specifically, TD (λ) outperforms both TD (0) and MC learners as well as all heuristic baselines, confirming that eligibility traces provide an effective tradeoff between bias and variance in value estimation.

Although direct replication of the original TD-Gammon results [2] was infeasible due to omitted hyperparameter details, our independent tuning yielded TD (λ) performance on par with the landmark study, thereby validating the robustness of deep function approximation for stochastic board games. Moreover, the competitive showing of MC and TD (0) underlines that simpler return-based updates can produce strong policies when paired with appropriately tuned architectures.

Because of the tight experimental timeframe, we omitted full minimax evaluation from our empirical results. At the two-ply search depth with α–β pruning, our minimax evaluator required on the order of 20 minutes per game, rendering a 1000-game round-robin prohibitively time-consuming. As a result, we could not perform a comprehensive head-to-head assessment of the handcrafted heuristic function under minimax play against our learned policies.

Because our limited time, we were unable to explore a deep Q-network (DQN) formulation, which offer a fundamentally different paradigm from value-only methods by directly approximating the action-value function $Q(s,a;\theta)$ rather than the state-value function $V(s;\theta)$. Variants such as Double

DQN, Dueling DQN, and more have demonstrated improvements in stability and sample efficiency. Implementing and rigorously evaluating these variants in the Backgammon context could offer deeper insights into the comparative merits of model-free Q-learning versus return-based and temporal-difference approaches for large, stochastic state spaces.

Looking forward, extending this work to richer network topologies, such as convolutional or attention-based models, may further accelerate learning across Backgammon's 10^20-state space. This study also provides a comprehensive framework for applying diverse AI and reinforcement-learning methodologies to challenging, high-dimensional decision-making problems.

VI. References

[1] Tesauro, Gerald. "Neurogammon: A neural-network Backgammon program." 1990 IJCNN international joint conference on neural networks. IEEE, 1990.

[2] Tesauro, Gerald. "Temporal difference learning and TD-Gammon." Communications of the ACM 38.3 (1995): 58-68.

[3] GitHub https://github.com/markshperkin/BackgammonAI-backend