

# Learning Yahtzee Using Deep Reinforcement Learning

Nick Pape

nap626

nickpape@utexas.edu

## Abstract

Write your abstract here. This should be a concise summary of your work, including the problem you’re addressing, your approach, and key results. Keep it to about 150-250 words.

## 1 Introduction

### 1.1 Yahtzee

While on the surface *Yahtzee* appears to be a trivial dice game (Hasbro, Inc., 2022), it is actually a complex stochastic optimization problem with combinatorial complexity. By its nature, it is heavily driven by random chance due to dice rolls, but strategic decision-making is required to maximize expected score over the course of a game. While the state complexity in solitaire play is large (Verhoeff, 1999) but manageable using dynamic programming techniques (Glenn, 2007), adversarial multiplayer *Yahtzee* remains an open problem due to the exponential growth of the state space with additional players Pawlewicz (2011). Even though *Yahtzee* has been a classic testbed for search and DP, it has been less explored as a deep reinforcement learning benchmark. Its varying, but well-defined, problem settings make it a good candidate to serve as a “bridge” between smaller benchmarks like *Lunar Lander* and large-scale games like *Go* or *Poker*.

### 1.2 Research Questions

The primary research questions this paper aims to address are:

- Can the tradeoff between maximizing the single-turn expected score and long-term game performance be quantified?
- Can a deep Reinforcement Learning agent trained only from self-play reach near-DP-

optimal performance in full-game solitaire *Yahtzee*?

- Which design choices (state encoding,  $\gamma$  schedule, entropy regularization, baselines, batch size, etc) most affect sample efficiency and final performance?

### 1.3 Contributions

The main contributions of this paper are:

- A deep RL agent that achieves  $\mathbb{E}[X_t]$  average score, within  $\delta$  points of an exact DP baseline under a strict budget of 1 million training games.
- An ablation study of key architectural and training choices and their impact on performance in *Yahtzee*.
- A failure mode analysis identifying issues in learned policies and their resolution through architectural or training modifications.

## 2 Related Work

### 2.1 Complex Games

Typical board and dice games have extreme state complexity or stochasticity; reinforcement learning methods are a natural fit for these problems. In a classic example, Tesauro (1995) utilized temporal difference learning to achieve superhuman performance in *Backgammon*, another game with a large state space and stochastic elements. Tetris has also been studied extensively; Bertsekas and Ioffe (1996) utilized approximate dynamic programming methods to learn effective policies for the game. Stochastic combinatorial games like Tetris can be effectively tackled using reinforcement learning methods (Gabillon et al., 2013). Adversarial games with extremely large

state spaces, such as Go were solved using Monte-Carlo Tree Search combined with deep value networks (Silver et al., 2016), and later work showed Go could be learned purely through self-play (Silver et al., 2017). Moravčík et al. (2017) demonstrated that *Texas Hold’em*, a game with hidden information and stochasticity, could be also effectively learned. Highly stochastic games can be learned as well, utilizing methods to ensure better exploration (Osband et al., 2016).

## 2.2 Yahtzee

Solitaire *Yahtzee* is a complex game with over  $7 \times 10^{15}$  possible states in its state space manifold, and has a high degree of stoachasticity due to dice rolls being the primary driver of state transitions. Despite this, it has been analytically solved using dynamic programming techniques (Verhoeff, 1999) and baseline metrics for optimal play are available, which calculates an ideal play score of 254.59 points. Later work by Glenn (2007) improved upon this by optimizing the DP approach, utilizing symmetries and limiting analysis only to reachable states, allowing the problem to be solved on limited hardware, as well as describing several benchmark heuristics. Glenn (2006) However, adversarial Yahtzee remains an open problem. While Pawlewicz (2011) showed that DP techniques can be expanded to 2-player adversarial *Yahtzee*, they do not scale to more players due to the exponential growth of the state space. Approximation methods must be utilized for larger player counts.

Some prior attempts have been made to apply reinforcement learning to *Yahtzee*. YAMS attemped by Q-learning and SARSA, but was not able to surpass 120 points median (Belaich, 2024). For example, Kang and Schroeder (2018) applied heirarchical MAX-Q, achieving an average score of 129.58 and a 67% win-rate over a 1-turn expectimax agent baseline. Vasseur (2019) explored strategy ladders for multiplayer Yahtzee, to understand how sensitive Deep-Q networks were to the upper-bonus threshold. Later, (Yuan, 2023) used Deep-Q networks and policy ladders to quantify the model’s preference for prioritizing the upper-section bonus. Yahtzotron used heavy supervised pretraining and A2C to achieve an average of 236 points (Häfner, 2021). Although not formally written, Dutschke reports a deep-Q agent achieving a score of  $241.6 \pm 40.7$  after just 8,000 games.

<b>2.3</b>	<b>Long-Horizon Policy Gradient Methods</b>
<b>2.4</b>	<b>Architecture and Optimization Components</b>
<b>3</b>	<b>Problem Formulation</b>
<b>3.1</b>	<b>Game Description</b>
<b>3.2</b>	<b>Single-Turn Optimization Task</b>
<b>3.3</b>	<b>Full-Game Reinforcement Learning Task</b>
<b>4</b>	<b>Methodology</b>
<b>4.1</b>	<b>State Representation &amp; Input Features</b>
<b>4.1.1</b>	<b>Dice Representation</b>
<b>4.1.2</b>	<b>Scorecard Representation</b>
<b>4.1.3</b>	<b>Computed Features</b>
<b>4.2</b>	<b>Neural Network Architecture</b>
<b>4.2.1</b>	<b>Trunk</b>
<b>4.2.2</b>	<b>Policy and Value Heads</b>
<b>4.2.3</b>	<b>Weight Initialization</b>
<b>4.2.4</b>	<b>Optimization &amp; Schedules</b>
<b>4.3</b>	<b>Reinforcement Learning Algorithms</b>
<b>4.3.1</b>	<b>REINFORCE for Single-Turn Optimization</b>
<b>4.3.2</b>	<b>PPO for Full-Game Reinforcement Learning</b>
<b>4.4</b>	<b>Training Regimes</b>
<b>4.4.1</b>	<b>Single-Turn Training</b>
<b>4.4.2</b>	<b>Transfer Learning Setup</b>
<b>4.4.3</b>	<b>Evaluation Protocol</b>
<b>5</b>	<b>Single-Turn Results &amp; Ablations</b>
<b>5.1</b>	<b>Baseline Model Performance</b>
<b>5.2</b>	<b>Representational Ablations</b>
<b>5.3</b>	<b>Architectural Ablations</b>
<b>5.4</b>	<b>Failure Mode Analysis</b>
<b>5.5</b>	<b>Summary</b>
<b>6</b>	<b>Full-Game Results</b>
<b>6.1</b>	<b>From-Scratch Training</b>
<b>6.2</b>	<b>REINFORCE</b>
<b>6.3</b>	<b>PPO</b>
<b>6.4</b>	<b>Transfer Learning</b>
<b>6.5</b>	<b>REINFORCE</b>
<b>6.6</b>	<b>PPO</b>
<b>6.7</b>	<b>Summary</b>
<b>7</b>	<b>Policy Analysis</b>
<b>7.1</b>	<b>Category Usage Distribution</b>
<b>7.2</b>	<b>Score Breakdown</b>
<b>7.3</b>	<b>Strategy Comparison</b>
<b>8</b>	<b>Discussion</b>
<b>9</b>	<b>Conclusion and Future Work</b>

to keep, and score points based on the resulting combinations. While arithmetic may be the minimum skill required to play, playing the game well requires strategic decision-making, probabilistic reasoning, and risk management.

In this paper, we explore the application of various reinforcement learning techniques to optimize gameplay in Yahtzee. We investigate how different reinforcement learning algorithms, model architectures, state and action representations, and reward structures impact the performance of agents in this complex environment.

## 10.1 Background

### 10.1.1 Rules of Yahtzee

Yahtzee is played with five standard six-sided dice and a shared scorecard containing 13 categories. Turns are rotated among players. A turn starts with a player rolling all five dice. They may then choose to keep some dice, and re-roll the remaining ones. This process can be repeated up to two more times, for a total of three rolls. After the final roll, the player must select one of the 13 scoring categories to apply to their current dice. Each category has specific scoring rules, and each can only be used once per game.

### 10.1.2 Mathematical Representation of Yahtzee

The space of all possible dice configurations is:

$$\mathcal{D} \in \{1, 2, 3, 4, 5, 6\}^5$$

and the current state of the dice is represented as:

$$\mathbf{d} \in \mathcal{D}$$

In addition, we can represent the score card as a vector of length 13, where each element corresponds to a scoring category:

$$\mathbf{c} = (c_1, c_2, \dots, c_{13}) \text{ where } c_i \in \mathcal{D}_i \cup \{\emptyset\}$$

and  $\emptyset$  indicates an unused category.

Let us also define a dice face counting function which we can use to simplify score calculations:

$$n_v(\mathbf{d}) = \sum_{i=1}^5 \mathbb{I}(d_i = v), \quad v \in \{1, \dots, 6\}$$

$$\mathbf{n}(\mathbf{d}) = (n_1(\mathbf{d}), \dots, n_6(\mathbf{d}))$$

Let the potential score for each category be defined as follows (where detailed scoring rules can be found in Appendix):

$$\mathbf{f}(\mathbf{d}) = (f_1(\mathbf{d}), f_2(\mathbf{d}), \dots, f_{13}(\mathbf{d}))$$

The domain of possible scores in a score card for each category is then:

$$\mathcal{C}_i = \left\{ i \cdot k : k \in \{0, 1, \dots, 5\} \right\} \forall i \in \{1, \dots, 6\}$$

$$\mathcal{C}_7 = \mathcal{C}_8 = \mathcal{C}_9$$

$$= \{0, 1, 2, \dots, 30\}$$

$$\mathcal{C}_9 = \{0, 25\}$$

$$\mathcal{C}_{10} = \{0, 30\}$$

$$\mathcal{C}_{11} = \{0, 40\}$$

$$\mathcal{C}_{12} = \{0, 50\}$$

The current turn number can be represented as:

$$t \in \{1, 2, \dots, 13\}, \quad t = \sum_{i=1}^{13} \mathbb{I}(c_i \neq \emptyset)$$

A single turn is composed of an initial dice roll, two optional re-rolls, and a final scoring decision. Let  $r = 0$ , with  $r \in \{0, 1, 2\}$  which is the number of rolls taken so far. Prior to the first roll, the dice are randomized:

$$\mathbf{d}_{r=0} \sim U(\mathcal{D})$$

The player must decide which dice to keep and which to re-roll. Let the player define a keep vector:

$$\mathbf{k} \in \{0, 1\}^5$$

where  $\mathbf{k}_i = 1$  indicates that die  $i$  is kept, otherwise it is re-rolled.

We can then define the transition of the dice state after a re-roll as:

$$\begin{aligned} \mathbf{d}' &\sim U(\mathcal{D}), \\ \mathbf{d}_{r+1} &= (\mathbf{1} - \mathbf{k}) \odot \mathbf{d}' + \mathbf{k} \odot \mathbf{d} \end{aligned}$$

When  $r = 2$ , the player must choose a scoring category to apply their current dice to. Define a scoring choice mask as a one-hot vector:

$$\mathbf{s} \in \{0, 1\}^{13}, \quad \|\mathbf{s}\|_1 = 1$$

For the purposes of calculating the final (or current) score, any field that has not been scored yet

can be counted as zero. We can define a mask vector for this:

$$\mathbf{u}(\mathbf{c}) \in \{0, 1\}^{13}$$

$$\mathbf{u}(\mathbf{c})_i = \mathbb{I}(c_i \neq \emptyset), \quad \forall i = \{1, \dots, 13\}$$

If a player achieves a total score of 63 or more in the upper section (categories 1-6), they receive a bonus of 35 points:

$$B(\mathbf{c}) = \begin{cases} 0, & 63 \leq \sum_{i=1}^6 \mathbf{u}(\mathbf{c})_i \cdot \mathbf{c}_i \\ 35, & \text{otherwise} \end{cases}$$

The player's score can thus be calculated as:

$$S(\mathbf{c}) = B(\mathbf{c}) + \langle \mathbf{u}(\mathbf{c}), \mathbf{c} \rangle$$

## 10.2 Complexity Analysis

### 10.2.1 State Space Analysis

The unfiltered state space for a solitaire game of Yahtzee can be described by the tuple of the current dice configuration, the scorecard status, and the roll count within the turn:

$$\mathbf{S} = (\mathbf{d}, \mathbf{c}, r)$$

The cardinality of the state space is primarily driven by the score card,  $\mathcal{C}$ :

$$|\mathcal{C}| = 7^6 \cdot 32^3 \cdot 3^4 \approx 3.12 \times 10^{11}$$

However the dice configuration space is also significant:

$$|\mathcal{D}| = 6^5 = 7776$$

And the roll count contributes a factor of 3:

$$|r \in \{0, 1, 2\}| = 3$$

Thus the unfiltered state space has a cardinality of:

$$\begin{aligned} |\mathbf{S}| &= |\mathcal{D}| \cdot |\mathcal{C}| \cdot |r| \\ &= 6^5 \cdot 7^6 \cdot 32^3 \cdot 3^4 \cdot 3 \\ &\approx 7.28 \times 10^{15} \end{aligned}$$

However,

### 10.2.2 Infeasibility of Tabular Methods

If this space were to be solved with tabular dynamic programming techniques using 16-bit half-precision floats, it would require nearly 1.5 petabytes of memory to store. While there have been prior attempts to use tabular methods for Yahtzee, these required significant optimizations and clustered computing resources to solve directly ()�.

For a local-play setting, this makes traditional tabular methods infeasible for Yahtzee, necessitating the use of function approximation methods such as deep neural networks.

### 10.2.3 Neural Network Size Bounds

One challenge in applying neural networks to approximate the value or policy function for Yahtzee is in determining an appropriate network size. We will quickly consider theoretical upper and lower bounds for the number of neurons required using both the full state space and a reduced representation.

Assume for simplicity our reduced representation consists of the dice state  $d$  encoded one-hot, the roll count  $r$  encoded one-hot, the bin count vector  $n(d)$  encoded one-hot, and a score mask  $u(c)$ . This gives input dimensionality of  $30 + 3 + 30 + 13 = 52$  and cardinality of  $2^{52} \approx 4.5 \times 10^{15}$ .

For a discrete control problem with a finite state size of  $|S|$ , there are classical results (Horne and Hush, 1994) showing that a recurrent neural network can represent any  $|S|$ -state finite state machine with a number of neurons that grows only logarithmically with  $|S|$ . This gives us an upper bound of  $O(\sqrt{|S|}) \approx 8.4 \times 10^7$  neurons. However, this is a prohibitively large number of neurons for practical training and inference.

For a lower bound, Hanin (2017) shows any continuous, convex function can in principle be approximated by a network whose hidden layer width has a lower bound of  $\Omega(d + 1)$ , and a non-convex function can be approximated by a network with a hidden layer width of at least  $\Omega(d+3)$ , where  $d$  is the input dimension. For example, assume Yahtzee is non-convex, and we represent the dice state  $d$  and  $r$  as one-hot encodings and utilize the score mask  $u(c)$ , then  $x = 6 \cdot 5 + 3 + 13 = 46$  means our *minimum* hidden layer width is 49 neurons.

In practice, the ideal network size is not driven by the cardinality of the state space, but rather by the

### 10.3 Intra-turn Decision Complexity

Test.

### 10.4 Problem Statement

Clearly state the problem you are addressing.

### 10.5 Research Objectives

Outline your research objectives and what you aim to achieve.

## 11 Research and Methods

Describe your approach and methods here.

### 11.1 Methodology Overview

Provide an overview of your research methodology and approach.

### 11.2 Model Architecture

Describe your model or approach in detail.

### 11.3 Experimental Design

Describe your experimental design and evaluation metrics.

## 12 Materials and Data Sources

Describe the dataset and materials you are using.

### 12.1 Dataset Description

Provide detailed information about your dataset, including size, characteristics, and source.

### 12.2 Data Collection and Preprocessing

Explain how data was collected and any preprocessing steps performed.

### 12.3 Tools and Technologies

List the tools, frameworks, and technologies used in your research.

## 13 Results

Present your experimental results here.

Method	Metric 1	Metric 2
Baseline	XX.XX	XX.XX
Your Method	XX.XX	XX.XX

Table 1: Results comparison

## 14 Discussion and Conclusion

### 14.1 Discussion of Results

Discuss your findings, their implications, and how they relate to existing literature.

### 14.2 Limitations

Acknowledge the limitations of your study and approach.

### 14.3 Future Work

Discuss potential directions for future research.

### 14.4 Conclusion

Summarize your work and its contributions to the field.

## 15 AI Usage Disclosure

This paper utilized artificial intelligence tools in the following ways:

- **GitHub Copilot** was used for initial typesetting assistance with the LaTeX document structure and formatting.
- **ChatGPT GPT-5** was used for brainstorming ideas for reinforcement learning applications in games, helping to narrow down to a particular game that was both non-trivial and interesting for this research.

All other content, including research methodology, analysis, results interpretation, and conclusions, represents original work by the author. The AI tools were used only for initial ideation and technical formatting support, not for generating substantive content or analysis.

GitHub Copilot was also used to write this section.

## 16 Acknowledgments

Acknowledge any help or resources you used.

## References

Alae Belaich. 2024. YAMS: Reinforcement Learning Project. <https://github.com/abelaich/YAMS-Reinforcement-Learning-Project>. GitHub repository, accessed 2025-11-16.

Dimitri P. Bertsekas and Sergey Ioffe. 1996. Temporal differences-based policy iteration and applications in neuro-dynamic programming. Technical Report LIDS-P-2349, Laboratory for Information and Decision Systems, MIT.

Markus Dutschke. A yahtzee/kniffel simulation making use of machine learning techniques. <https://github.com/markusdutschke/yahtzee>. GitHub repository; accessed: 2025-11-16.

Victor Gabilon, Mohammad Ghavamzadeh, Alessandro Lazaric, and Bruno Scherrer. 2013. Approximate dynamic programming finally performs well in the game of tetris. In *Advances in Neural Information Processing Systems*, volume 26.

Jeffrey R. Glenn. 2006. An optimal strategy for yahtzee. Technical Report CS-TR-0002, Loyola College in Maryland, Department of Computer Science.

Jeffrey R. Glenn. 2007. Computer strategies for solitaire yahtzee. In *2007 IEEE Symposium on Computational Intelligence and Games (CIG)*, pages 132–139.

Dion Häfner. 2021. Learning to play yahtzee with advantage actor-critic (a2c). <https://dionhaefner.github.io/2021/04/yahtzotron-learning-to-play-yahtzee-with-advan>. Accessed: 2025-11-16.

Boris Hanin. 2017. Universal function approximation by deep neural nets with bounded width and relu activations.

Hasbro, Inc. 2022. YAHTZEE Game: Instructions. Official rules and instructions.

Bill G. Horne and Don R. Hush. 1994. Bounds on the complexity of recurrent neural network implementations of finite state machines. In *Advances in Neural Information Processing Systems 6*, pages 359–366, San Francisco, CA. Morgan Kaufmann.

Minhyung Kang and Luca Schroeder. 2018. Reinforcement learning for solving yahtzee. AA228: Decision Making under Uncertainty, Stanford University, class project report.

Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisý, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. 2017. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. 2016. Deep exploration via bootstrapped DQN. In *Advances in Neural Information Processing Systems*, volume 29.

Jakub Pawlewicz. 2011. Nearly optimal computer play in multi-player yahtzee. In *Computers and Games*, pages 250–262, Berlin, Heidelberg. Springer Berlin Heidelberg.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering

the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359.

Gerald Tesauro. 1995. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68.

Philip Vasseur. 2019. Using deep q-learning to compare strategy ladders of yahtzee. Source code available at <https://github.com/philvasseur/Yahtzee-DQN-Thesis>.

Tom Verhoeff. 1999. Optimal solitaire yahtzee strategies (slides). <https://www-set.win.tue.nl/~wstomv/misc/yahtzee/slides-2up.pdf>.

Max Yuan. 2023. Using deep q-learning to play two-player yahtzee. Senior essay, Computer Science and Economics.

## A Full Yahtzee Scoring Rules

Next we define the indicator functions for each of the scoring categories:

$$\mathbb{I}_{3k}(\mathbf{d}) = \mathbb{I}\left\{\max_v n_v(\mathbf{d}) \geq 3\right\}$$

$$\mathbb{I}_{4k}(\mathbf{d}) = \mathbb{I}\left\{\max_v n_v(\mathbf{d}) \geq 4\right\}$$

$$\mathbb{I}_{\text{full}}(\mathbf{d}) = \mathbb{I}\left\{\exists i, j \in \{1, \dots, 6\} \text{ with } n_i(\mathbf{d}) = 3 \wedge n_j(\mathbf{d}) = 2\right\}$$

$$\mathbb{I}_{\text{ss}}(\mathbf{d}) = \mathbb{I}\left\{\exists k \in \{1, 2, 3\} \text{ with } \sum_{v=k}^{k+3} \mathbb{I}\{n_v(\mathbf{d}) > 0\} = 4\right\}$$

$$\mathbb{I}_{\text{ls}}(\mathbf{d}) = \mathbb{I}\left\{\exists k \in \{1, 2\} \text{ with } \sum_{v=k}^{k+4} \mathbb{I}\{n_v(\mathbf{d}) > 0\} = 5\right\}$$

$$\mathbb{I}_{\text{yahtzee}}(\mathbf{d}) = \mathbb{I}\left\{\max_v n_v(\mathbf{d}) = 5\right\}$$

The potential score for each category can then

be defined as:

$$f_j(\mathbf{d}) = j \cdot n_j(\mathbf{d}), \quad j \in \{1, \dots, 6\}$$

$$f_7(\mathbf{d}) = \mathbf{1}^\top \mathbf{d} \cdot \mathbb{I}_{3k}(\mathbf{d})$$

$$f_8(\mathbf{d}) = \mathbf{1}^\top \mathbf{d} \cdot \mathbb{I}_{4k}(\mathbf{d})$$

$$f_9(\mathbf{d}) = 25 \cdot \mathbb{I}_{\text{full}}(\mathbf{d})$$

$$f_{10}(\mathbf{d}) = 30 \cdot \mathbb{I}_{\text{ss}}(\mathbf{d})$$

$$f_{11}(\mathbf{d}) = 40 \cdot \mathbb{I}_{\text{ls}}(\mathbf{d})$$

$$f_{12}(\mathbf{d}) = 50 \cdot \mathbb{I}_{\text{yahtzee}}(\mathbf{d})$$

$$f_{13}(\mathbf{d}) = \mathbf{1}^\top \cdot \mathbf{d}$$

$$\mathbf{f}(\mathbf{d}) = (f_1(\mathbf{d}), f_2(\mathbf{d}), \dots, f_{13}(\mathbf{d}))$$