Yahtzee: Reinforcement Learning Techniques for Stochastic Combinatorial Games

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 it's 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, γ 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 ¡X¿ 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. It has been shown that highly stochastic games can be learned reliably, as long as methods to ensure better exploration are used (Osband et al., 2016).

2.2 DP Methods for Yahtzee

Solitaire *Yahtzee* is a complex game with an upper bound of 7×10^{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 (2006) optimizing the DP approach via symmetries to propose an efficient algorithm. Glenn (2007) calculated the reachable state space to be 5.3×10^8, and subsequently implemented this efficint algorithm, which and compared favorably to several benchmark heuristics.

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.

2.3 Reinforcement Learning for Yahtzee

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) applied Deep-Q networks to the adversarial setting. 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 ± 40.7 after just 8,000 games.

2.4 Policy Gradient Methods and Variance Reduction

Temporal Difference (TD) (Sutton, 1988) and REINFORCE (Williams, 1992) methods have long been used to assign credit and calculate training signals in long-horizon tasks. These have spawned an entire heirarchy of policy gradient methods, including asynchronous advantage actor-critic (A3C) (Mnih et al., 2016), proximal policy optimization (PPO) (Schulman et al., 2017), and others (Sutton et al., 2000).

Unfortunately, all reinforcement learning methods struggle with long-horizon credit assignment due to the compounding variance of returns and sparse rewards, leading to high variance updates and unstable training (Bjorck et al., 2022). There are numerous architectural and optimization tricks which can be employed to mitigate these issues. For example, during training, the Adam optimizer (**?**) is widely used due to its adaptive momentum term, which helps to stabilize high variance updates. Performing "warmups" of the learning rate improves RL performance by deprioritizing policy updates during the poorly-performing early training regime (Kalra and Barkeshli, 2024) (Liu et al., 2025). At initialization, Kaiming/He weight initialization (He et al., 2015) has been shown to reduce variance in deep networks by normalizing activation variances across layers. The SiLU and Swish activation functions (Ramachandran et al., 2017) are improvements of GELU (Hendrycks and Gimpel, 2016) that have been shown to be highly effective in RL tasks; their smooth and fully differentiable curves improve gradient flow (Elfwing et al., 2018).

Another common problem with policy gradient methods is policy collapse due to premature convergence to suboptimal policies. ¡ENTROPY¿ ¡TEMPERATURE¿ ¡CLIPPING¿ ¡MONITORING ENTROPY¿

3 Problem Formulation

3.1 Game Description

3.1.1 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.

3.1.2 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:

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

and $\varnothing$ indicates an unused category.

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

$$\begin{aligned} n_v(\mathbf{d}) &= \sum_{i=1}^{5} \mathbb{I}(d_i = v), \quad v \in \{1, \ldots, 6\} \\ \mathbf{n}(\mathbf{d}) &= \big(n_1(\mathbf{d}), \ldots, n_6(\mathbf{d})\big) \end{aligned}$$

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

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

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

$$\begin{aligned} \mathcal{C}_i &= \Big\{ i \cdot k : k \in \{0, 1, \ldots, 5\} \Big\} \forall i \in \{1, \ldots, 6\} \\ \mathcal{C}_7 &= \mathcal{C}_8 = \mathcal{C}_9 \\ &= \{0, 1, 2, \ldots, 30\} \\ \mathcal{C}_9 &= \{0, 25\} \\ \mathcal{C}_{10} &= \{0, 30\} \\ \mathcal{C}_{11} &= \{0, 40\} \\ \mathcal{C}_{12} &= \{0, 50\} \end{aligned}$$

The current turn number can be represented as:

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

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:

$$\begin{aligned} &\mathbf{u}(\mathbf{c}) \in \{0, 1\}^{13} \\ &\mathbf{u}(\mathbf{c})_i = \mathbb{I}\big(c_i \neq \varnothing\big), \quad \forall i = \{1, \ldots 13\} \end{aligned}$$

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:

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

3.2 MDP Formulation

We can formulate Yahtzee as a Markov Decision Process (MDP) defined by the 4-tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$.

3.3 Single-Turn Optimization Task

3.4 Full-Game Reinforcement Learning Task

4 Methodology

1400-1700 words

4.1 State Representation & Input Features

The design of $\phi(\mathbf{s}) \to \mathbf{x}$ is one of the most critical components to the performance of a model (2018).

As such, several different representations were tested to evaluate their impact on learning efficiency and final performance.

4.1.1 Dice Representation

The dice representation can be encoded in several ways, depending on if we want to preserve permutation invariance or not. Preserving ordering information (and implicitly, ranking) gives the model the benefit of being able to directly output actions corresponding to dice indices, however, it comes at the cost of implicitly biasing the model to specific dice orderings; in other words, biasing towards a local optima of always keeping the highest ranking dice. However, eliminating ordering information requires the model to either waste capacity learning permutation invariance or be architecturally supportive of it (e.g. with self-attention). It also requires a different action representation, since actions can no longer correspond to specific dice indices. We attempted 5 different dice representations:

- **One-hot encoding of ordered dice:** Each die is represented as a one-hot vector of length 6, and the 5 dice are concatenated in order.
- **Bin representation:** Here we pass through the dice face count vector $\mathbf{n}(\mathbf{d})$.
- **Combined representation:** Both ordered one-hot and bin representations are concatenated.
- **Learnable encoding of dice:** Each die is represented as a learnable embedding vector of length d, and self-attention is applied.
- **Learnable encoding of dice with positional encodings:** Similar to above, but with sinusoidal positional encodings added to each die embedding.

4.1.2 Scorecard Representation

There are two important pieces of information ϕ must encode about the scorecard: whether a category is open or closed, and some form of progress towards the upper bonus. For the category availaboility, we simply pass through $\mathbf{c}$. Here, several different representations were tested:

- **Raw upper score:** $\sum_{i\in\mathbf{c}} c_i$
- **Distance to bonus:** $63 - \sum_{i\in\mathbf{c}} c_i$
- **Norm. upper score:** $\min(\frac{1}{63}\sum_{i\in\mathbf{c}} c_i, 1)$
- **Distance to bonus:** $\mathbb{I}(\sum_{i\in\mathbf{c}} c_i \geq 63)$
- **”Golf” score:**

4.1.3 Computed Features

4.2 Neural Network Architecture

4.2.1 Trunk

4.2.2 Policy and Value Heads

4.2.3 Weight Initialization

4.2.4 Optimization & Schedules

4.2.5 Training Metrics

4.3 Reinforcement Learning Algorithms

4.3.1 REINFORCE for Single-Turn Optimization

4.3.2 PPO for Full-Game Reinforcement Learning

4.4 Training Regimes

4.4.1 Single-Turn Training

4.4.2 Transfer Learning Setup

4.4.3 Evaluation Protocol

5 Results

5.1 Generalization of Single-Turn Agent to Full Game

200-400 words

5.2 Single-Turn Results & Ablations

600-700 words

5.2.1 Baseline Model Performance

5.2.2 Representational Ablations

5.2.3 Architectural Ablations

5.2.4 Failure Mode Analysis

5.2.5 Summary

5.3 Full-Game Results

600-700 words

5.3.1 From-Scratch Training

5.3.2 REINFORCE

5.3.3 PPO

5.3.4 Transfer Learning

5.3.5 REINFORCE

5.3.6 PPO

5.3.7 Summary

5.4 Policy Analysis

300-500 words

5.4.1 Category Usage Distribution

5.4.2 Score Breakdown

5.4.3 Strategy Comparison

6 Discussion

400-600 words

7 Conclusion and Future Work

200-300 words

7.0.1 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 $\mathbf{d}$ encoded one-hot, the roll count r encoded one-hot, the bin count vector $\mathbf{n}(\mathbf{d})$ encoded one-hot, and a score mask $\mathbf{u}(\mathbf{c})$. 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 $|\mathbf{S}|$, there are classical results (Horne and Hush, 1994) showing that a recurrent neural network can represent any $|\mathbf{S}|$-state finite state machine with a number of neurons that grows only logarithmically with $|\mathbf{S}|$. This gives us an upper bound of $O(\sqrt{|\mathbf{S}|}) \approx 8.4 \times 10^7$ 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 a example, assume Yahtzee is non-convex, and we represent the dice state $\mathbf{d}$ and r as one-hot encodings and utilize the score mask $\mathbf{u}(\mathbf{c})$, 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

8 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.

References

Alae Belaich. 2024. 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.

Johan Bjorck, Carla P. Gomes, and Kilian Q. Weinberger. 2022. Is high variance unavoidable in rl? a case study in continuous control. *arXiv preprint arXiv:2110.11222*.

Markus Dutschke. A yahtzee/kniffel simulation making use of machine learning techniques. GitHub repository; accessed: 2025-11-16.

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. 2018. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11.

Victor Gabillon, 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). 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.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034.

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*.

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.

Dayal Singh Kalra and Maissam Barkeshli. 2024. Why warmup the learning rate? underlying mechanisms and improvements. *arXiv preprint arXiv:2406.09405*.

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

Yuxing Liu, Yuze Ge, Rui Pan, An Kang, and Tong Zhang. 2025. Theoretical analysis on how learning rate warmup accelerates convergence. *arXiv preprint arXiv:2509.07972*.

Volodymyr Mnih, Adria Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*.

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.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. 2017. Searching for activation functions. *arXiv preprint arXiv:1710.05941*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

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.

Richard S. Sutton. 1988. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44.

Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*, 2nd edition. MIT Press, Cambridge, MA.

Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12 (NIPS 1999)*, pages 1057–1063.

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`.

Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256.

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:

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

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

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

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

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

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

The potential score for each category can then be defined as:

$$\begin{aligned}
f_j(\mathbf{d}) &= j \cdot n_j(\mathbf{d}), \qquad j \in \{1, \dots, 6\} \\
f_7(\mathbf{d}) &= \mathbf{1}^\top \mathbf{d} \cdot \mathbb{I}_{\text{3k}}(\mathbf{d}) \\
f_8(\mathbf{d}) &= \mathbf{1}^\top \mathbf{d} \cdot \mathbb{I}_{\text{4k}}(\mathbf{d}) \\
f_9(\mathbf{d}) &= 25 \cdot \mathbb{I}_{\text{full}}(\mathbf{d}) \\
f_{10}(\mathbf{d}) &= 30 \cdot \mathbb{I}_{\text{ss}}(\mathbf{d}) \\
f_{11}(\mathbf{d}) &= 40 \cdot \mathbb{I}_{\text{ls}}(\mathbf{d}) \\
f_{12}(\mathbf{d}) &= 50 \cdot \mathbb{I}_{\text{yahtzee}}(\mathbf{d}) \\
f_{13}(\mathbf{d}) &= \mathbf{1}^\top \cdot \mathbf{d} \\
\mathbf{f}(\mathbf{d}) &= \big(f_1(\mathbf{d}), f_2(\mathbf{d}), \dots, f_{13}(\mathbf{d})\big)
\end{aligned}$$