
PLAYING YAHTZEE WITH DEEP REINFORCEMENT LEARNING - A SYSTEMATIC COMPARISON OF DIFFERENT APPROACHES

A PREPRINT

Markus T. Dutschke*

Erlangen, Germany
post at markusdutschke dot de

Elias D. Striatum

Department of Electrical Engineering
Mount-Sheikh University
Santa Narimana, Levand
stariate@ee.mount-sheikh.edu

January 15, 2020

ABSTRACT

In this paper we present an open source Q-learning algorithm for the dice game yahtzee. We implemented a variation of the Q-learning algorithm as used by Mnih [?], which he used for playing Atari games. The specific obstacles of yahtzee are thereby to handle two different types of possible actions: 1) choose what dice to re-roll; 2) choose a category on the score board, the significantly larger number possible actions of type 1 compared to an Atari game controller and the randomness in the response of the game to the players actions of type 1. By presenting different implementations of increasing complexity, we give the reader an overview of different concepts to improve the performance of Q-learning for certain situations and evaluate their performance in the specific use case. Among those concepts are different exploration strategies, concepts to handle randomness and a technique for the efficient handling of the two decision types. The most successful implementation achieves superhuman performance within a few thousand training cycles.

Keywords Q-learning · neural networks · exploration strategies · replay memory

1 Introduction

The complete source code of this project is publicly available at

<https://github.com/markusdutschke/yahtzee>

Since Mnih's famous publication 'Playing Atari with deep reinforcement learning' [?] strong research interest has evolved around the possibilities of Q-learning in combination with neural networks. Thereby computer and board games turned out to be an excellent playground for this research, due to their complex character, their easy reproducibility and the clear definition of the systems rules.

ToDo: One should mention some historic achievements with the corresponding machine learning technology here. This should include - gackgammon (ibm, temporal time difference) - go (deepMind, ?) and many more (chesss?, poker?, doom?).

Solving these puzzles often paved the path for more complex applications like ToDo: - thermomix, which evolved out of the solution for fruit ninja (this is false and just an example) - more eamples of this structure

Especially the dice game Yahtzee has a set of interesting properties, which makes it an highly interesting test system for our purpose:

*Use footnote for providing further information about author (webpage, alternative address)—*not* for acknowledging funding agencies.

- Yahtzee is a broadly known game. This makes it easy for many researchers to evaluate a certain decision of the algorithm.
- Even after several hundred games, Yahtzee is still challenging for a human player. It thereby represents a challenge, which goes beyond the development of a few best practice strategies.
- There is a mixture of randomness and strategy involved. This makes it an interesting application which combines the reproducible domain of games with the influence of statistical uncertainty in real-life applications.
- Yahtzee is exactly solvable. The solution is far beyond the human abilities but can be used to evaluate the performance of the Q-learning implementation.

2 The dice game Yahtzee

2.1 Rules

2.2 Implementation

The code is executed by calling *main.py* in the root folder. For an extended functionality, there are several functions implemented, which are however not called. These functions is a good starting point of playing around with the code.

The complete game logic can be found in *lib/yahtzee.py*.

The class *Dice*, thereby encodes a set of zero to five dice. The *roll* method is used to re-roll one or more dice of a given set. The *keep* method reduces the dice in a set to the ones, which shall be kept. This method is used to compactly encode game situations, where certain dice shall be re-rolled and hence their values are irrelevant.

The score board is encoded in *ScoreBoard*. The class-method *get_cat_points* returns the number of points a player gets, when assigning a certain dice combination to a category. *stat_cat_score* calculates the exact expectation values and their standard error for each category based on a set of dice. Using a dice configuration of five dice, is the trivial use-case. For dice configurations with less than five dice, all combinations of the unspecified dice are looped over. The method *add* is used to assign a set of dice to a category on the score board. Open categories are accessed by the methods *mask*, *open_cats_mask* (todo: probably redundant to *mask*) and *open_cats*. The sum of a score board is evaluated by the methods *getUpperSum*, *getLowerSum* and *getSum*.

A game can be played by using the class *Game*. Thereby a player object is given for initialization. A *player*-object supplies two methods: *choose_reroll* and *choose_cat*. The *Game*-class is calling the *autoplay*-method on initialization. The game is started with an empty score board and five dice values. From then on the suitable methods of *player* are called and the resulting course of the game is simulated in alternating order. A protocol of the game is found in the attribute *log*, which can be most easily accessed by a string-cast of *Game*. This represents all dice configurations and the players decisions together with the complete score board. The internal evaluation for the decisions can be accessed by setting debug to 1. This feature is not supported for all players.

Different player types are defined in *lib/bot.py*. They all descend from the class *AbstractPlayer*, which abstractly defines the two mandatory methods *choose_reroll* and *choose_cat*. Further a method *benchmark* is implemented, which calculates the mean and standard deviation of playing *nGames* (usually 100) with this player. For reproducibility the seed for random numbers is usually fixed (constant *BENCHMARK_SEED* in *main.py*).

One subclass, which is important to mention here, is *TemporaryPlayer*. This class represents a player, which is initiated by custom re-roll and choose category functions. This class fulfills mainly the purpose of clean code and is used to simulate a game, when just the two functions are available.

The other subclasses of *AbstractPlayer* are implementation of specific players/strategies as described in section 4.

2.3 Solutions and heuristics

3 Q-learning

This chapter contains the theoretical background and explains different techniques to achieve a faster convergence and more successful game strategies.

3.1 Background

3.2 Handling the two decision types

3.3 Encoding

TODO: general remarks about encoding

The current dice configuration is always stored and handled in a sorted manner. This reduces the complexity of the setup and supports the convergence of the regressors.

In the following, we explain a set of possible encodings for the different regressors. We will refer to these in section 4, when discussing the specific implementation of the different self-learning players. If the number of features or the encoding are not specified, this means, that there are different implementations used.

For the chose-a-category regressor, we only need to forecast the rest score of the game, based only on the available categories on the score board and the upper sum (relevant for the bonus).

- category availability: the availability of each category on the score board; 13 features encoded as 0 or 1
- bonus: current sum of the upper half of the score board; 1 feature

This encoding is equivalent for all regressors of that kind.

For the re-roll decision regressor the following additional encodings turned out to be worth considering:

- attempt: first or second re-roll attempt; 1 feature encoded as 0 or 1
- dice: the dice, which are held fixed (i.e. not being re-rolled); 5 features
- dice histogram: number of dice showing ones, twos, ...; 6 features
- sum of dice: sum of the values of all dice as entered in the category chance, 1 feature encoded $\text{int} \in [5, 30]$.
- n of a kind: number of different values, which are presented by exactly n dice; 1 feature for each $n \in \{1, 2, 3, 4, 5, 6\}$
- helper small straight: some heuristic, intended to represent the chances of a small straight
- helper large straight: some heuristic, intended to represent the chances of a large straight
- statistical forecast: exact or heuristic forecast of the expected score in each category based on the currently kept dice configuration; 13 features encoded as expectation value and as 0 if not available

3.4 Exploration

3.5 Concepts to handle a stochastic system response

4 Implementation of classic and ai players

In the following we describe the implementation of different players in *lib/bot.py*.

4.1 Naive Implementations

- random implementation - greedy implementation with and without re-roll

4.2 Remarks about AI players

The AI players have a set of common / similar methods, which shall be explained here quickly.

The *eval_options_cat/reroll* method evaluates all possible actions (either choosing a category or choosing dice to re-roll) and returns an array consisting of the (action, evaluated benefit) tuples for each option. The method *train* is used to train/configure the regressors based on a nGames training games. To ensure reproducibility, the random seed is set to the total number of training games for each training cycle. *save* and *load* are used to save and load a previously trained AI player.

There also exists a set of different methods *add_to...RgrMem* and *to_repMem*, which store the training game experience in the replay memory. This has some size limitation, which is defined as an argument of *__init__*. Based on this, the replay memory is also truncated at a point (either in *truncate...* or at the end of *to_repMem*).

As discussed in ??, the encoding of the current state is of crucial importance. For this reason each regressor has a corresponding method *encode_...*, which just encodes the state of the current game situation (score board and dice) in the most compact and processible way. The corresponding property *nFeat_...* is a constant, used in the methods *encode* and *train* and defines the dimensionality (number of features) of the regressor input.

4.3 AI player Version 0

This player reaches a maximum average score of 200.1 after 1053 training games (compare commit d68fc302a929101602ff95c609298ffcf6f86612).

We use two multi layer perceptron regressors (*scrRgr* and *rrRgr*). Both predict the rest score of the game (i.e. the final score minus the current score). This concept is updated in version 1.

The encoding for the re-roll decision

4.4 AI player version 1

4.5 AI player version 2

5 Benchmark

In this chapter the benefit of different Q-learning concepts are quantitatively benchmarked. The implementation of these benchmarks can be found in the functions *bench...* in *main.py* with player implementations in *botBench.py*

5.1 Direct comparison of the AI players

Performance plots for version 0-2

5.2 Information encoding

Different encodings. Not yet sure, what to compare. Maybe: *rgrSC* with - 13 inputs (-1 for empty, otherwise score) - 26 inputs (first 13: 0 for empty, second 13: 0 or 1 for empty and used) - a good encoding (check maybe player v2)

5.3 Exploration

- epsilon greedy - softmax - minMaxRat

5.4 Concepts to handle a stochastic system response

- implicitly in MLP regressor (v0) - explicitly in *mlprgr* with pretraining and benchmarking (this is v1) - exactly by lookup table (v2)

6 Conclusion

Collection of key facts, whatever turned out to bring the most significant improvement.

7 END OF ARTICLE - NOW FOLLOWS THE LATEX TEMPLATE

8 Headings: first level

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetur adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

See Section 8.

8.1 Headings: second level

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetur.

$$\xi_{ij}(t) = P(x_t = i, x_{t+1} = j | y, v, w; \theta) = \frac{\alpha_i(t) a_{ij}^{w_t} \beta_j(t+1) b_j^{v_{t+1}}(y_{t+1})}{\sum_{i=1}^N \sum_{j=1}^N \alpha_i(t) a_{ij}^{w_t} \beta_j(t+1) b_j^{v_{t+1}}(y_{t+1})} \quad (1)$$

8.1.1 Headings: third level

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Paragraph Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

9 Examples of citations, figures, tables, references

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat sapien aliquet odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetur at, consectetur sed, eleifend ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed porttitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui.

[1, 2] and see [3].

The documentation for natbib may be found at

<http://mirrors.ctan.org/macros/latex/contrib/natbib/natnotes.pdf>

Of note is the command `\citet`, which produces citations appropriate for use in inline text. For example,

```
\citet{hasselmo} investigated\dots
```

produces

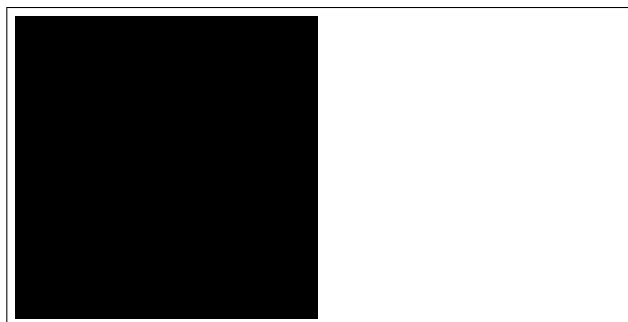


Figure 1: Sample figure caption.

Table 1: Sample table title

Part		
Name	Description	Size (μm)
Dendrite	Input terminal	~ 100
Axon	Output terminal	~ 10
Soma	Cell body	up to 10^6

Hasselmo, et al. (1995) investigated...

<https://www.ctan.org/pkg/booktabs>

9.1 Figures

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetur odio sem sed wisi.

See Figure 1. Here is how you add footnotes.² Sed feugiat. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Ut pellentesque augue sed urna. Vestibulum diam eros, fringilla et, consectetur eu, nonummy id, sapien. Nullam at lectus. In sagittis ultrices mauris. Curabitur malesuada erat sit amet massa. Fusce blandit. Aliquam erat volutpat. Aliquam euismod. Aenean vel lectus. Nunc imperdiet justo nec dolor.

9.2 Tables

Etiam euismod. Fusce facilisis lacinia dui. Suspendisse potenti. In mi erat, cursus id, nonummy sed, ullamcorper eget, sapien. Praesent pretium, magna in eleifend egestas, pede pede pretium lorem, quis consectetur tortor sapien facilisis magna. Mauris quis magna varius nulla scelerisque imperdiet. Aliquam non quam. Aliquam porttitor quam a lacus. Praesent vel arcu ut tortor cursus volutpat. In vitae pede quis diam bibendum placerat. Fusce elementum convallis neque. Sed dolor orci, scelerisque ac, dapibus nec, ultricies ut, mi. Duis nec dui quis leo sagittis commodo.

See awesome Table 1.

9.3 Lists

- Lorem ipsum dolor sit amet
- consectetur adipiscing elit.
- Aliquam dignissim blandit est, in dictum tortor gravida eget. In ac rutrum magna.

²Sample of the first footnote.

References

- [1] George Kour and Raid Saabne. Real-time segmentation of on-line handwritten arabic script. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pages 417–422. IEEE, 2014.
- [2] George Kour and Raid Saabne. Fast classification of handwritten on-line arabic characters. In *Soft Computing and Pattern Recognition (SoCPaR), 2014 6th International Conference of*, pages 312–318. IEEE, 2014.
- [3] Guy Hadash, Einat Kermany, Boaz Carmeli, Ofer Lavi, George Kour, and Alon Jacovi. Estimate and replace: A novel approach to integrating deep neural networks with existing applications. *arXiv preprint arXiv:1804.09028*, 2018.