Using deep reinforcement learning to solve Mastermind-like games

Bartosz Swedrowski

June 16, 2021

1. Introduction

This project is a term paper for *Reasoning* class. The goal of the project was to try to solve increasingly more difficult Mastermind-like environments using deep reinforcement learning. I introduce the structure of the project briefly in section 2. Then I give an overview of the most successful run in section 3. Section 4 enumerates details about the algorithm, including the architecture of its network and its hyperparameters, and the reward function. Section 5 contains key insights that could help in developing bigger environments solving algorithms. I conclude the project in section 6.

2. The goal and what was used to achieve it

The goal of Mastermind is to guess the code in a few attemps, basing on received feedback about tried combinations. The code consists of k balls, each being in one of n colors. That makes n^k possible variations with repetition (codes).

Following means were used to achieve the goal:

- Deep Deterministic Policy Gradient algorithm (from Keras-rl2),
- Python 3.7.8 (64 bit),
- Tensorflow 2.3.0,
- Keras 2.4.3,
- Keras-rl2 1.0.4,
- Gym 0.17.2,
- Matplotlib 3.3.0.

3. The most successful run

The most successful run has achieved 100% guess rate in average 4,46 guesses in a simplified Mastermind environment (3 balls, 4 colors). There were 4^3 (64) possible codes formulated as three digits long strings, each digit being in a $< 0, 3 >$ range. Limit of guesses in a single epizode was 7. Total training time was two hours. Algorithm's solutions to all the possible cases can be found on the following page.

Listing 1: Algorithm's solution

```
[000]: [[000]]
[001]: [[000], [002], [003], [003], [001]]
[002]: [[000], [002]]
[003]: [[000], [002], [003]]
[010]: [[000], [002], [100], [102], [010]]
[011]: [[000], [011]]
[012]: [[000], [011], [013], [012]]
[013]: [[000], [011], [013]]
[020]: [[000], [002], [100], [102], [020]]
[021]: [[000], [011], [013], [322], [032], [021]]
[022]: [[000], [011], [013], [012], [023], [022]]
[023]: [[000], [011], [013], [012], [023]]
[030]: [[000], [002], [100], [102], [030]]
[031]: [[000], [011], [013], [322], [030], [031]]
[032]: [[000], [011], [013], [102], [022], [032]]
[033]: [[000], [011], [013], [012], [033]]
[100]: [[000], [002], [100]]
[101]: [[000], [011], [130], [310], [331], [101]]
[102]: [[000], [011], [331], [102]]
[103]: [[000], [011], [331], [102], [103]]
[110]: [[000], [011], [130], [110]]
[111]: [[000], [131], [132], [111]]
[112]: [[000], [131], [112]]
[113]: [[000], [131], [132], [113]]
[120]: [[000], [011], [331], [102], [002], [120]]
[121]: [[000], [131], [132], [103], [012], [331], [121]]
[122]: [[000], [131], [112], [123], [122]]
[123]: [[000], [131], [112], [103], [123]]
[130]: [[000], [011], [331], [102], [003], [130]]
[131]: [[000], [131]]
[132]: [[000], [131], [132]]
[133]: [[000], [131], [132], [133]]
[200]: [[000], [002], [100], [012], [320], [200]]
[201]: [[000], [011], [131], [301], [123], [201]]
[202]: [[000], [011], [220], [102], [012], [121], [202]]
[203]: [[000], [011], [220], [102], [203]]
[210]: [[000], [011], [131], [210]]
[211]: [[000], [131], [112], [312], [310], [211]]
[212]: [[000], [131], [212]]
[213]: [[000], [131], [312], [213]]
[220]: [[000], [011], [220]]
[221]: [[000], [131], [112], [111], [203], [032], [221]]
[222]: [[000], [131], [112], [113], [222]]
[223]: [[000], [131], [212], [213], [123], [223]]
[230]: [[000], [011], [220], [302], [002], [021], [230]]
[231]: [[000], [131], [132], [102], [210], [231]]
[232]: [[000], [131], [112], [133], [233], [232]]
[233]: [[000], [131], [112], [233]]
[300]: [[000], [002], [100], [002], [300]]
[301]: [[000], [011], [131], [301]]
[302]: [[000], [011], [220], [302]]
[303]: [[000], [011], [220], [302], [222], [020], [303]]
[310]: [[000], [011], [131], [310]]
[311]: [[000], [131], [132], [311]]
[312]: [[000], [131], [312]]
[313]: [[000], [131], [312], [313]]
[320]: [[000], [011], [220], [302], [002], [320]]
[321]: [[000], [131], [112], [111], [203], [210], [321]]
[322]: [[000], [131], [212], [213], [322]]
[323]: [[000], [131], [212], [233], [023], [013], [323]]
[330]: [[000], [011], [220], [302], [003], [330]]
[331]: [[000], [131], [132], [103], [010], [331]]
[332]: [[000], [131], [112], [133], [223], [332]]
[333]: [[000], [131], [112], [333]]
```

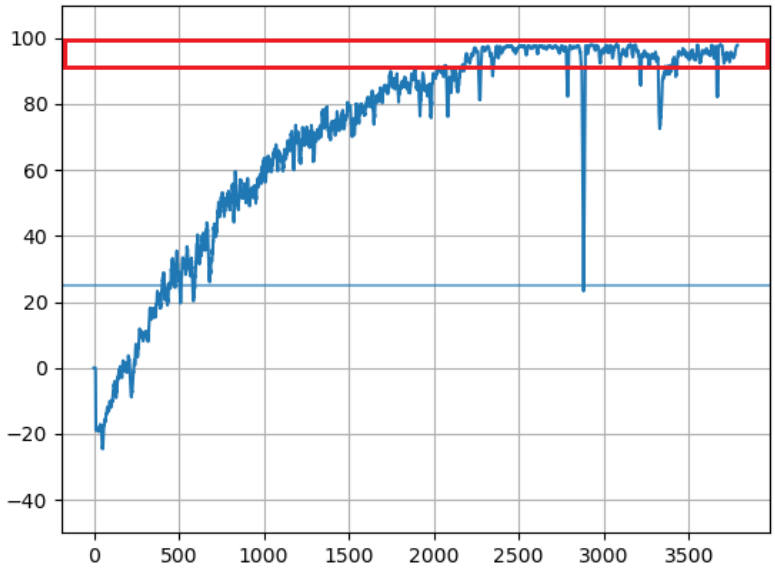

Figure 1: The 100% guess rate run's reward history chart. Y axis is mean reward across previous episodes. X axis values should be read as x*300 steps of training. Horizontal blue line marks 30% guess rate threshold. Red frame marks 100% guess rate.

4. Details

Algorithm's input and output:

- Actor's input: number of the guess ($< 1, 7 >$ range), tried code ($< 000000, 111111 >$ range, binary), feedback pegs ($< 0000 - 1111 >$ range, binary) + 6 alike observations from previous steps,
- actor's output: code ($< 000000, 111111 >$ range, binary),
- critic's input: actor's input + actor's output,
- critic's output: number that is used to modify actor's future behavior, $< -1, 1 >$ range.

Hyperparameters:

- Actor's architecture: 66 (input), 378 (ReLu), 128 (ReLu), 48 (ReLu), 6 (sigmoid),
- critic's architecture: 72 (actor's input + actor's output), 378 (ReLu), 128 (ReLu), 48 (ReLu), 1 (linear),
- gamma (importance of future rewards): 0.99,
- window size (amount of previous steps considered by the algorithm + 1): 7,

- optimizer: Adam (learning rate: 0.0001),
- memory size: 80000,
- warm-up steps: 10000.

Reward function:

- 100 for guessing the code correctly (at the end of the episode),
- -12 for not guessing the code correctly (at the end of the episode),
- -1 for each new started guess,
- -5 if current guess has already happened in this episode.

5. Insights

Main insights from the project:

- Main differences between NN solving different versions of the environment were memory size and layers size (width),
- algorithm remembered every unique case or so it appears,
- the key to better generalisation might be to deepen the network (more layers).
- wider network is able to get 94,92% on 4 balls, 4 colors version (256 codes),
- categorical variables have to be one-hot/binary/[...] encoded.

6. Conclusion

Algorithm has successfully solved multiple, sequentially increasing versions of Mastermind game, including 16, 64, and 256 combinations options. Network width and memory size is what seems to limit the algorithm, since those were key factors in solving bigger environments. Considering solvability of consecutively bigger environments, it is justified to say that regular Mastermind game (4 balls, 6 colors) would be solved with a bigger network and with more time.