DQN Analysis with 2048

1. Introduction

Deep Reinforcement Learning has proved useful in solving tasks bounded by an environment/rules. Q-learning, a popular method of RL, has been shown to solve wide varieties of diverse environments. Due to the vigorous study/use of Q-learning, engineers and researchers have developed various add-ons and improvements to this algorithm. Using the once popular game 2048 as our environment, we will analyze these improvements to investigate sample efficiency and find the optimal Q-learning policy.

2. Problem Definition and Algorithm

2.1 Task Definition

The environment of 2048 consists of very simple dynamics. In the context of reinforcement learning given some state s_i, an agent can take some action a_i and transition to state s_{i+1}. The challenge in solving this transition is determining the future reward value of these given state-action pairs. If we can successfully learn these values, acting about an optimal policy will yield a high-performance algorithm.

2.2 Algorithm Definition

There is an inherent problem with determining an optimal policy for a game such as 2048. Given some state s_i and set of actions $\{ a_0 a_1 \dots a_i \}$, which action is the "best" action? Let us define the "best" action in a set of actions to be a_j which maximizes future reward. Because of

this definition, we cannot simply choose actions based on the immediate reward they yield. We use Q-learning and the Bellman equation to combat this dilemma.

The Bellman equation is used to generate our Q-values (the value of a given state-action pair). This value is the sum of immediate reward at step t_i and a discounted future Q-value, which is chosen by taking the max Q-value from the set of Q-values at step t_{i+1}. Here is what the equation looks like:

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a') \middle| s,a\right]$$

In order to obtain this max Q-value from the next timestep, we use our neural network. Using this Q*(s, a) as a target value for training, we will slowly but surely approximate the true value of a given state action pair. Using the neural network output as part of our training labels may seem very strange, but this method has been proven to approximate sufficiently.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

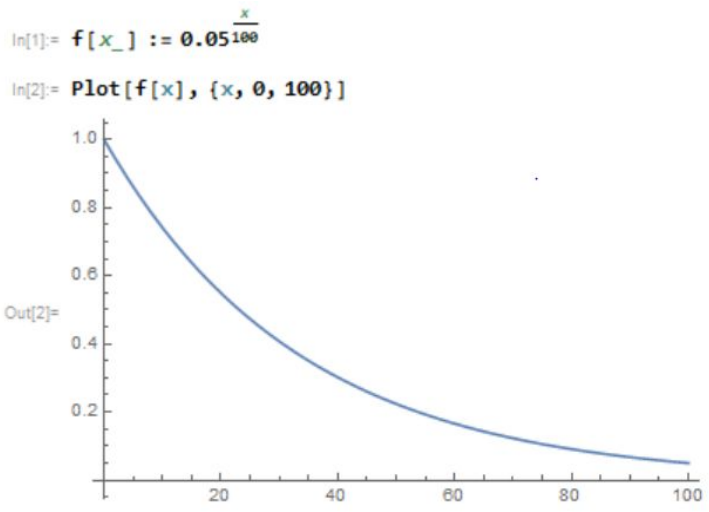
Deep Q-Learning Pseudocode: *Playing Atari with Deep Reinforcement Learning, DeepMind Technologies*

This instance of the vanilla DQN will act as our baseline of performance and learning. Our goal is to achieve an increased sample efficiency and better policy through training. In order to achieve this goal, we have implemented various add-ons to our vanilla DQN baseline.

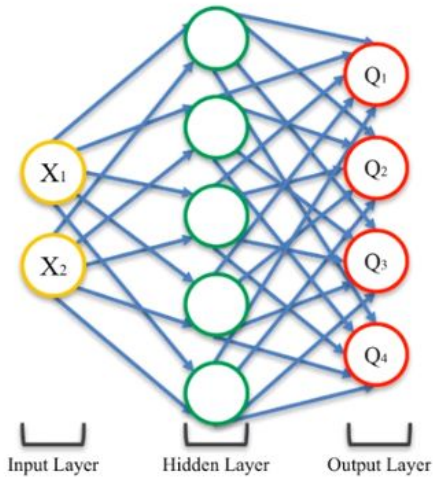
We hypothesized that adding a target network, a copy of our value network, to our base model will yield better results (*Deep Reinforcement Learning with Double Q-learning Hasselt et al 2015*). This DeepMind paper mentions the overestimation bias associated with a single network architecture when taking our forward pass to get target values. They go on to mention Hasselt's 2010 paper which proved to solve this problem with Double Q-learning (in a tabular setting) and show results of the DDQN (with NN's, not tabular) also solving this issue.

Furthermore we propose an alteration to the common epsilon greedy exploration strategy. In the instance manifesting an agent inside the 2048 environment, decrementing epsilon slowly will have insignificant effects on performance due to the average length of the episode in this game. We need the agent to take optimal series actions as soon as possible in order to even get to the desired late game states. In order to achieve this behavior, we have formulated an

exponentially decaying epsilon: $\varepsilon = \varepsilon_{min}^{\,i\,/\,iterations}$, where *i* is in range (0, iterations). Let us observe the behavior of epsilon in a 100 iteration and ε_{min} of 0.05 training example.
As seen in this graph below of our new epsilon decay, the value of epsilon decreases fast for lower values of x and slowly approaches our minimum value. This function's behavior exhibits what we desired. This rate of decay for our random action probability parameter will hopefully enable training in more difficult/advanced states and therefore elicit higher results/rewards.



For all training, we will be using a neural network with the following architecture:



NN architecture:
** All layers Densely connected **

Input Layer: size 16 - relu activation

Layer 1: size 256 - relu activation
Layer 2: size 256- relu activation
Layer 3: size 256 - relu activation

Output Layer: size 4 - linear activation

3. Experimental Evaluation

3.1 Methodology

Due to the nature of the 2048 environment, we have a very easy metric to analyze the performance of our agent: score. Throughout training, we are playing many games at every epoch. Thus, the most naive/obvious way to evaluate the performance of our model throughout training would include a plot of score vs epoch.

This method of performance analytics may seem satisfactory, although our exploration strategy (epsilon greedy) may interfere. In order to sufficiently explore our environment, our agent makes random moves at a probability of 1.0 decreasing toward 0.0 as we train. This results in poor scores due to random moves even in instances where our model has the potential to play much better. To combat this uncertainty, we also track our agents performance without an epsilon greedy strategy for a specified number of tests per epoch. We plot this"greedy" performance on top of our training performance to analyze model performance during training.

Our final metric of evaluation is utilized once training is complete. This evaluation acts as the final score of our agent. The function for this final test will iterate through *n* games where *n* is usually a very large number (n = 250 or 1000 or etc.). We allow our agent to act completely greedily to test what has been learned through training. We average the scores of *n* games, where our theoretical "score" or performance of a given agent can be determined as *n* approaches infinity.

As an extension to this final metric, we also take advantage of analysing standard deviation of scores. This deviation metric tells us how precise an agent is able to obtain a specific average score. We can use this metric in conjunction with average score to not only determine how "good" an agent is, but also how consistently it plays at a given level.

3.2 Results

It has been made apparent for some time now that the extension of a target network (DDQN) on the vanilla DQN baseline yields increased agent performance on a variety of environments. This has been confirmed by our experiments and outlined in the figure below. However, the extension of exponential epsilon decay, which was made up specifically for this unique environment, yields an increase in results on top of both vanilla DQN and DDQN based agents. These results are also outlined in the figure below.

	Average	Standard Deviation
DQN - Linear ε Decay	***2512.8***	***1150.4***
DQN - Exponential ε Decay	***2542.6***	***1144.8***
DDQN - Linear ε Decay	***2708.2***	***1091.6***
DDQN - Exponential ε Decay	***2857.8***	***1024***

3.3 Discussion

With the addition of DDQN and or exponential epsilon decay to the vanilla model, we see an increase in average score and decrease in standard deviation. These results are promising but in order for us to conclude these results as statistically significant, many more trials are needed.

4. Related Work

Rainbow: Combining Improvements in Deep Reinforcement Learning. Hessel et al. The folks at Google DeepMind devised an algorithm that combines many improvements onto the vanilla DQN that we have explored thus far. This paper presents gameplay on a variety of Atari games with and without specific improvements to the vanilla DQN and the associated results. The work at DeepMind differs from the 2048 DQN due to game data representation. In our system, we use the data from the 2048 board as inputs for our neural network. In the Rainbow paper, raw pixels of the game are used as network input. 2048 is very easily converted to numerical data, but the classic atari games require much more interpretation for this conversion.

5. Conclusion

Deep reinforcement learning proves to be an interesting, sophisticated, and fast evolving sub field of machine learning. The improvements that have been made on games like Go, Chess, Atari, etc. are happening years quicker than experts ever imagined thanks to these RL algorithms. The agents that we develop may only exist virtually or in our scripted "environments" for now, but all over the world agents are being developed using similar techniques to tackle some of the hardest real world problems.

Although our models/environments lack the complexity of those on the cutting edge of RL research, the thought experiment of how we can improve agent performance for a given set of environments is crucial to the development of this subfield of ML. It would be an interesting experiment implementing this exponential epsilon decay on environments similar to 2048. We can define "similar to 2048" as an environment where playing well results in longer games and different state spaces in these later-game situations. With this being clearly defined, we can make the hypothesis that using an exponentially decaying epsilon would yield higher scores than a linearly decaying epsilon.

Code and Dataset

https://github.com/jvoynow/DQN-analysis-with-2048

Bibliography

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. A. 2013. Playing atari with deep reinforcement learning. CoRR abs/1312.5602.

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2018). Rainbow: Combining Improvements in Deep Reinforcement Learning. In the AAAI Conference on Artificial Intelligence (AAAI).

van Hasselt, Hado, Guez, Arthur, and Silver, David. Deep Reinforcement Learning with Double Qlearning. In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, 2016. URL