Bowling with Deep Learning

Zizhen Jiang
Department of Electrical Engineering
Stanford University, Stanford, CA, 94305
jiangzz@stanford.edu

Abstract

A deep learning model, which can learn control policies directly from high-dimensional sensory input using reinforcement learning, is explored and improved for Atari Bowling. Extraction of high-level features from raw-sensory data is made possible by the recent advances in deep learning. However, in general, agents take a series of actions to get a reward. In the environments with sparse rewards, reinforcement learning agents struggle to learn. To overcome this challenge, researches investigated multiple approaches, such as Q-learning and double Q-learning. In this project, we analyze the performances of Q-learning and double Q-learning on all Atari games, and further explore and improve the architecture of Q-learning and double Q-learning specifically on Bowling game. A general architecture design guideline for a specific game, i.e. complex games take complex neural network architecture, is suggested. For the learning process, the training and testing dataset is generated when playing the game. The scores of the agent playing each round is evaluated as the results. The learning history is provided to investigate the learning process.

1. Introduction

Human beings are attracted by games, which are usually for enjoyment and sometimes used as an educational tool. One of our favorite games, Rayman, is a platform video game series owned by Ubisoft [Rayman-wiki]. The agent of Rayman can jump, fly, run, or beat monster. The target is to save Electoons locked in all cages and then to save the world (Fig. 1). Complex agent movements and various environments challenge human beings in playing.

To understand how human beings perceive the game and play, researchers are developing methods to learn to control agents directly from high-dimensional sensory input like vision using reinforcement learning (RL). Before most successful RL applications that operated on these domains relied on hand-crafted features combined with linear value functions or policy representations. The quality of the feature representation decided the performance of those systems. Recent advances in deep learning lead to breakthroughs in computer vision, making it possible to extract high-level features from raw sensory data. These methods utilize a range of neural network architectures, which have been demonstrated beneficial for RL with sensory data [Mnih, 2013].

Figure 1. One of our favorite games, Rayman, is a platform video game series owned by Ubisoft.

However, in general, games may involve a series of actions required to get a reward. A human can easily understand how to acquire a reward. It is difficult for a computer to sample billions of random moves and succeed with less than 1% possibility. A reinforcement learning model for extended-time games needs to focus on sparse and delayed rewards.

To deal with sparse and delayed rewards, DeepMind introduces deep Q-learning, which falls under the umbrella of reinforcement learning [Mnih, 2013]. The world is shocked that computers are able to automatically learn to play Atari games (Fig. 2) and beat world champions at Go (Fig. 3) [Silver, 2016]. For the first time, reinforcement learning agents were learning from high-dimensional visual input using convolutional neural networks. With only

Figure 2. Alpha Go on the cover page of Nature. [This image is from Nature]

Figure 3. Atari computer system. [Q-news]

pixels as input and scores as rewards, the agents achieved superhuman performance.

In this work, we analyzes the performance of the reported Q-learning architectures on various Atari games. Instead of providing a general architecture, we explore and improved the Q-learning based architecture specifically on one game - Bowling (Fig. 4). A general architecture design guideline for a specific game is suggested.

2. Background

The tasks are considered as an agent interacts with the Atari emulator, in a sequence of actions, observations and rewards. At each time-step, the agent selects an action a_t from the set of game actions, $A = \{1, 2, ..., K\}$. The emulator takes the action as input, and modifies its internal state and the game score. The agent doesn't observe the internal state of the emulator; instead, the agent observes an image $x_t \in \mathbb{R}^d$ from the emulator, which is a vector of raw pixel values representing the current screen. In addition, the agent receives a reward r_t representing the change in game score. Note that in general the whole prior sequence of actions and observations determines the game score; agents may receive feedbacks about an action after many thousands of time-steps have elapsed (Fig. 5).

We need to consider the sequences of actions and observations, $s_t = x_1, a_1, x_2, ..., a_{t-1}, x_t$ and the game strategies, which depends on the sequences, are learned through playing. Because it is impossible to fully understand the current situation from only the current screen $_xt$, of which the agent only observes images, when the task is partially observed. With the assumption that all sequences in the emulator terminate in a finite number of time-steps, this formalism gives rise to a large but finite Markov decision process (MDP) in which each sequence is a distinct state. As a result, standard reinforcement learning methods for MDPs can be applied, simply by using the complete sequence s_t as the state representation at time

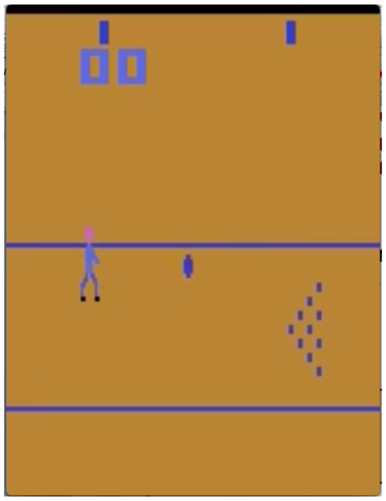
Figure 4. Bowling is chosen as an example of 'easier games'.

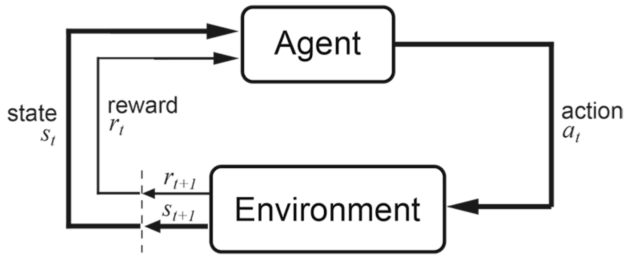

Figure 5. Learning flow of reinforcement learning. [Q-news]

t. The goal of the agent is to interact with the emulator by selecting actions in a way that maximizes future rewards.

Multiple approaches have been developed. DeepMind first presented a model using deep Q-network (DQN) [Mnih, 2013; Mnih, 2015; Wang, 2015; Kulkarni, 2016; van Hasselt, 2016]. Conbining Q-learning with a flexible deep neural network, DQN was tested on a varied and large set of deterministic Atari 2600 games, reaching human-level performance on many games [Mnih, 2013; Mnih, 2015]. Several variants have been later demonstrated. The double Q-learning (Double DQN) is investigated to deal with the overestimation caused by insufficiently flexible function approximation and noise [van Hasselt, 2016]. The dueling network represents two separate estimators: one for the state value function and one for the state-dependent action advantage function, generalizing learning across actions without imposing any change to the underlying reinforcement learning algorithms [Wang, 2015]. Focusing on the specific 'harder'games, such as Montezumas Revenge, Kulkarni proposed hierarchical-DQN (h-DQN) [Kulkarni, 2016]. However, still few work analyzes the differences between Atari games and investigate the architecture design from the 'easier' games.

In this project, we analyze the difference between Atari games, investigate the reported performances of DQN and double DQN on various Atari games, and explore the architecture design for a 'easier' game - Bowling. A general architecture design guideline for a specific game is finally provided.

3. Technical Approach

We first implement deep Q-network (DQN) presented by DeepMind [Q-wiki; Mnih, 2015]. Under the assumption that future rewards are discounted by a factor of per time-step, we define the future discounted return at time t as $R_t = \sum_{t'=t}^{T} \gamma^{t'} t_{\gamma_{t'}}$, where T is the time-step at which the game terminates. The optimal action-value function $Q^*(s,a)$ is defined as the maximum expected return achievable by following any strategy, after seeing some sequences and then taking some action a, $Q^*(s,a) = max_\pi \mathbb{E}[R_t|s_t = s, a_t = a, \pi]$, where π is a policy mapping sequences to actions (or distributions over actions).

At the beginning, $Q(s,a)$ returns an (arbitrary) fixed value, randomly generated by the network. Then, each time the agent selects an action, and observes a reward and a new state $Q(s,a)$ is updated. The new state may depend on both the previous state and the selected action. A simple value iteration update is the core of the algorithm, assuming that the old value is close to the correct and makes a correction based on the new information.

$$Q(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{old\ value} + \alpha_t \big(\overbrace{r_{t+1} + \gamma \underbrace{max_a Q(s_{t+1},a)}_{optimal\ future\ value}}^{learned\ value} - \underbrace{Q(s_t,a_t)}_{old\ value} \big) \quad (1)$$

where α_t is the learning rate, r_{t+1} is the reward, and γ is the discount factor.

By minimizing a sequence of loss function $L_i(\theta_i)$, the Q-network can be trained at each iteration i,

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(.)}[(y_i - Q(s,a;\theta_i))^2] \quad (2)$$

where $y_i = \mathbb{E}_{s'\sim\varepsilon}[r + \gamma max_{a'} Q(s',a';\theta_{i-1})|s,a]$ is the target for iteration i and $\rho(s,a)$ is a probability distribution over sequences s and actions a that we refer to as the behavior distribution. We fix the parameters from the previous iteration θ_{i-1} awhen optimising the loss function $L_i(\theta_i)$. The following gradient is given by differentiating the loss function with respect to the weights ,

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(.);s'\sim\varepsilon}[(r + \gamma max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i))\nabla_{\theta_i} Q(s,a;\theta_i)] \quad (3)$$

Stochastic gradient descent is used to optimise the loss function, the weights are updated after every time-step, and the expectations are replaced by single samples from the behavior distribution ρ and the emulator ε receptively.

Further we implement double deep Q-network (Double DQN) [van Hasselt, 2016]. The key difference between DQN and double DQN is that double DQN decouple the selection from the evaluation.

4. Analysis

We analyze the reported performance of DQN and double DQN on various Atari games [van Hasselt, 2010; van Hasselt, 2016]. Five key features of the reported Atari games are proposed: a) moving agent, whether the agent can move in the observation image; b) whether the reward target can move in the observation image; c) moving monster, whether the monster can move to eat the agent; d) moving background, whether the game has a background that confuses the perception of the agent; e) complex movement, whether the agent's contour in the image can influence the next action. The summary of those features is provided at Table 1.

We compare the reported scores [van Hasselt, 2016] and find that most of the games with smaller number of features benefit from the double DQN, i.e. Double DQN tends to improve the agent performance. It may be indicator that the reported architecture with DQN overestimate the action values for games with smaller number of features, which we may call those games "easier games". The reported architecture used three convolution layers and two fully-connected layers, besides all those layers are separated by rectifier liner units (ReLu) [van Hasselt, 2016]. The results may indicate that this network may be too complex for the easier games.

One other outstanding observation is that when games involve complex agent movement and multiple background settings, it is difficult to utilize DQN and double DQN to improve agents' performances. As an example, the agent in Montezuma's Revenge can run, jump, and move up and down in multiple environments, which results in specific architecture developments. We perform the traditional DQN and double DQN on Montezuma's Revenge. As expected, the agent either dies soon or stuck at somewhere in the image (Fig. 6-7).

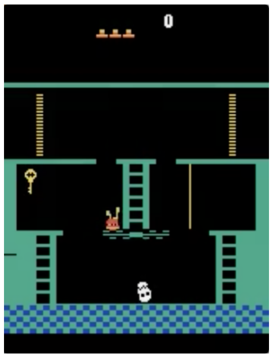
Figure 6. The agent in Montezuma's Revenge dies soon after training.

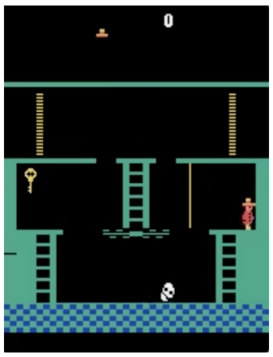
Figure 7. The agent in Montezuma's Revenge gets stuck after training.

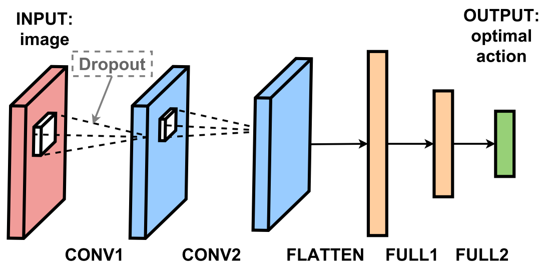

Figure 8. Simplified neural network.

For this project, we are focusing on the easier games. Without losing generosity, we are using the game Bowling.

5. Experiment

We simplify the architecture by using two layers of convolution layers and two fully-connected layers with Relu as the separation layer between adjacent layers (Fig. 8). The work is implemented based on the codes provided by cs234 [Base] and implemented in Tensorflow [TF]. The first convolution layer convolves the input with 16 filters of size 8 (stride 4), and the second convolution layer has 32 layers of size 4 (stride 2). This is followed by a fully-connected hidden layer of 512 units. We may utilize dropout in-between the first and second layers. Our architecture has a smaller number parameters of the network.

The performance comparison of the DeepMind architecture and our architecture is summarized in Table 2. Simpler architecture performs better on the simpler Bowling game using the same Q-learning update strategy. Our architecture beats the performance of DQN reported and has a potential to beat double DQN if tuning the learning rate more carefully. Reference learning process plot of our network is shown in Fig. 9.

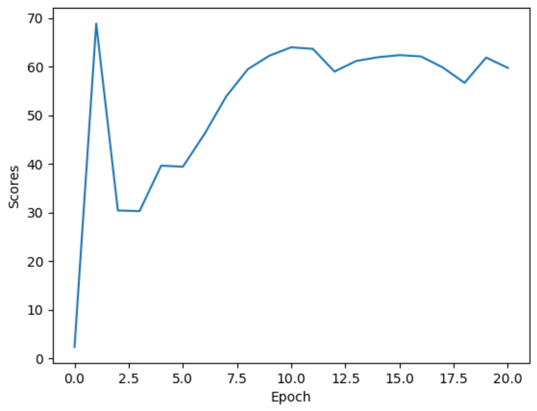

Figure 9. Reference learning process plot of our network with DQN.

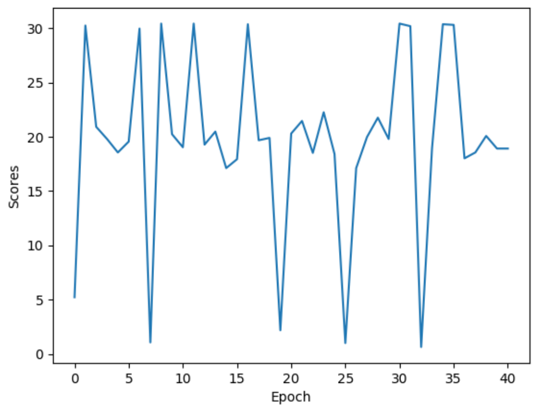

Figure 10. Reference learning process plot of our network with Double DQN.

We further compare the performance of DQN and double DQN using our architecture. However, double DQN do not improve the performance further. It may be caused by the enough simplicity of the network without overestimation or the large learning rate, which makes the scores change back and forth (Fig. 10).

Another approach using dropout [Dropout] is also investigated. We utilize dropout with a keep possibility of 0.8. The result is shown in Fig. 11. The performance may be improved if providing more training time and tuning the parameters more.

The performance of our architecture with DQN, Double

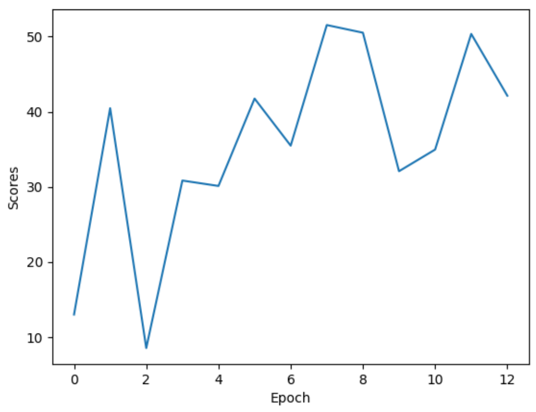

Figure 11. Reference learning process plot of our network with dropout.

DQN and dropout is summarized in Table 2.

6. Conclusion

This work analyzes the key features of various Atari games and suggests that simpler games typically require simpler neural network (less number of parameters) and complex games require complex neural network (larger number of parameters). Based on the analysis, we simplify the architecture into two convolution layers and two fully-connected layers and beat the performance reported by DeepMind with DQN. We demonstrate a concept to analyze the features of the games and then explore and improve the neural network architecture.

7. Future Work

A layer of batch normalization [Ioffe, 2015] may be added on the current simplified network to verify if there is any over-fitting and to improve the robustness of the network. The analysis may provide a method to improve the architecture from bottom to up. Researchers may perform the feature analysis over all Atari games, later explore and improve the network on specific games first, and then generalize the architecture for all atari games.

8. Reference

[cs231n] https://cs231n.stanford.edu

[cs234] https://cs234.stanford.edu

[Base] Base code from cs234 assignment 2, http://web.stanford.edu/class/cs234/assignment2/index.html. I do not take cs234. The github codes I found online require some different versions of packages and cannot work easily. Later I found the package provided by cs234 suitable for the implementation of this project.

[Dropout] https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-7-action-selection-strategies-for-exploration-d3a97b7cceaf

[Ioffe, 2015] S. Ioffe, et al., ”Batch normalization: Accelerating deep network training by reducing internal covariate shift.” arXiv preprint arXiv:1502.03167 (2015).

[Kulkarni, 2016] T. D. Kulkarni, et al., ”Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation.” Advances in Neural Information Processing Systems. 2016.

[Mnih, 2015] V. Mnih, et al., Human-Level Control through Deep Reinforcement Learning, Nature 518, 519-533

[Mnih, 2013] V. Mnih, et al., Playing Atari with Deep Reinforcement Learning, NIPS, 2013

[Rayman-wiki] https://en.wikipedia.org/wiki/Rayman

[Silver, 2016] D. Silver, et al. ”Mastering the game of Go with deep neural networks and tree search.” Nature 529.7587 (2016): 484-489.

[TF] Tensorflow, https://www.tensorflow.org/tutorials/

[Q-news] https://deepsense.io/playing-atari-on-ram-with-deep-q-learning/

[Q-wiki] https://en.wikipedia.org/wiki/Q-learning

[van Hasselt, 2010] H. Van Hasselt, ”Double Q-learning.” Advances in Neural Information Processing Systems. 2010.

[van Hasselt, 2016] H. van Hasselt, et al., Deep Reinforcement Learning with Double Q-Learning, AAAI, 2016

[Wang, 2015] Z. Wang, et al., ”Dueling network architectures for deep reinforcement learning.” arXiv preprint arXiv:1511.06581 (2015)

Table 1. Summary of five features for reported Atari games. ’y’ indicates that the game has the feature; ’n’ indicates not having this feature.

Game	Moving Agent	Moving Target	Moving Monster	Moving Background	Complex Movement
Alien	y	n	y	n	y
Amidar	y	n	y	n	n
Assault	y	y	y	n	n
Asterix	y	y	n	n	n
Asteroids	y	y	y	n	y
Atlantis	n	y	y	n	n
Bank Heist	y	n	n	n	n
Battle Zone	n	y	y	y	n
Beam Rider	y	y	y	n	n
Bowling	y	n	n	n	n
Boxing	y	y	y	n	n
Breakout	y	n	n	n	n
Centipede	y	y	y	n	n
Chopper Command	y	y	y	y	n
Crazy Climber	y	n	y	n	n
Demon Attack	y	y	y	n	n
Double Dunk	y	y	y	n	y
Enduro	y	n	y	y	y
Fishing Derby	y	y	y	n	n
Freeway	y	n	y	n	y
Frostbite	y	y	y	n	y
Gopher	y	n	y	n	n
Gravitar	y	y	y	y	y
Ice Hockey	y	y	y	n	y
James Bond	y	y	y	n	n
Kangaroo	y	n	y	n	n
Krull	y	n	y	n	n
Kung-Fu Master	y	y	y	n	n
Montezuma’s Revenge	y	y	y	y	y
Ms. Pacman	y	n	y	n	n
Name This Game	y	y	y	n	n
Pong	y	y	n	n	n
Private Eye	y	y	y	y	n
Q*Bert	y	n	y	n	n
River Raid	y	y	y	y	n
Road Runner	y	y	y	y	n
Robotank	n	y	y	y	n
Seaquest	y	y	y	n	y
Space Invaders	y	y	y	n	n
Star Gunner	y	y	y	y	n
Tennis	y	y	n	n	y
Time Pilot	y	y	y	y	n
Tutankham	y	y	y	y	y
Up and Down	y	y	y	y	n
Venture	y	y	y	n	n
Video Pinball	y	y	n	n	n
Wizard of Wor	y	y	y	n	y
Zaxxon	y	y	y	y	n

Table 2. Performance comparison of Deepmind architecture and our architecture

Game	Random	Human	DQN	Double DQN	My DQN w/o Dropout	My DQN w/o Dropout Peak	My Double DQN	My DQN w/ Dropout
Bowling	23.10	154.80	42.40	70.50	59.80	68.92	18.92	42.12