# Learn to Run: QWOP using Reinforcement Learning

## CIS 731 - Artificial Neural Networks

Report by - *Amar Shrestha and Swatantra kafle*

December 8, 2017

# Contents

# 1 Introduction

Reinforcement learning (RL) [1] is a machine learning problem which involve an agent learn set of actions that maximize some notion of numerical reward. In this learning task, agents are never told which set of actions to take under the given environment state to maximize reward. The agents learns to map a situation to action through successive trials. In other words RL is a closed loop problem as the current system action has some effect of the future system input and action. Figure 1 shows the Agent-Environment interaction in RL process. An Agent observe state of environment and take an action.

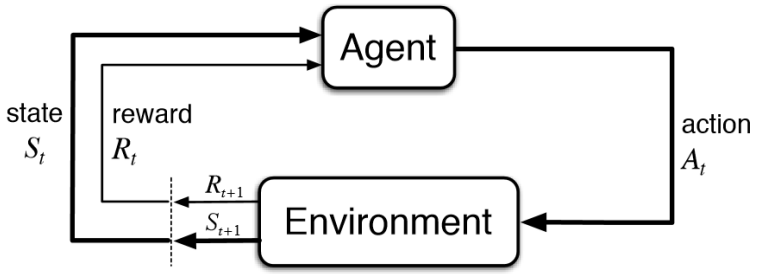



Figure 1: Agent-Environment interaction in RL

Let $\mathcal{A}, \mathcal{S}$ represent the set of actions available to the agent, and the set of states the agent can encounter. Based on a state $s \in \mathcal{S}$, agent take some action $a \in \mathcal{A}$. Based on action $a$, and environment state $s$ agent receives some reward $r \in \mathcal{R}$ from the environment and the state changes to $s' \in \mathcal{S}$. Let $P(s'|s, a)$ be the transition probability to state $s'$ given that environment is in state $s$ and action $a$ is taken. $\pi$ defines the policy of the agent. Policy maps environment state to the action for agent. The goal of the agent to find an optimal policy, $\pi$ that maximizes the expected discounted return. Discounted return is defined as:

$$DR_t = r_{t+1} + \gamma r_{t+2} + \gamma r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1},$$

where $\gamma$ is a discount factor with $0 \leq \gamma \leq 1$. The value of $\gamma$ weighs the importance of immediate rewards with the future rewards. If $\gamma = 0$, RL agent objective is to maximize the immediate reward. This will lead the agent follow greedy approach in maximizing the discount return. Higher the value of $\gamma$, the higher emphasis on the future rewards. Value Functions which are functions of state and action pair, are used as a measure to formalize 'how good' given action-state pair is with respect to policy $\pi$. State-value function for policy $\pi$ is defined as

$$V_\pi(s) = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| \mathcal{S} = s \right]$$

Similarly, action-value function for policy $\pi$, $Q_\pi$ is defined as

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| \mathcal{S} = s, \mathcal{A} = a \right]$$

# 2 Problem

In this section, we first introduce the game and then we define the problem we solve.

## 2.1 Game Introduction:

QWOP [2] is a fairly popular flash game where a player should control the movement of an athlete using Q, W, O, and P keys such that the athlete completes the 100m race. The Q and W keys each drive one of the runners thighs, while the O and P keys work the runners calves. The Q key drives the runners right thigh forward and left thigh backward, and the W key also affects the thighs and does the opposite. The O and P keys work in the same way as the Q and W keys, but with the runners calves. The actual amount of movement of a joint is affected by the resistance due to forces from gravity and inertia placed upon it.

## 2.2 Learning QWOP:

QWOP can be learnt using RL in two different ways, namely

- General AI
- Robot AI

### 2.2.1 General AI:

In this RL problem, an agent learns to play a game, here QWOP, by observing raw pixel informations. This RL learning approach has some similarity with how humans learn to play these computer games. As any of Q, W, O or P key is pressed, the game environment generates new set of new pixels (based on the key pressed) and some reward value is generated. The agent learns to act (i.e. press Q, W, O and P keys) such that the reward is maximized. The popular work of Deep Mind Technologies on Reinforcement learning Atari games is based on this paradigm [3]. The performance of learned agents were encouraging, and these agents were even able to beat human experts in some of the games they trained. We note that raw pixel information is a high dimensional vector and certainly has effect on computational complexity of learning algorithms. Hence, we might actually want to develop some RL algorithms with the input observation feature is of reasonable dimension.

### 2.2.2 Robot AI

Consider a situation where we want a robot learn to walk, run in a real environment. Here, robot takes account of all the sensory information from its sensors (state of the robot) and takes some action based on these sensory observations. We consider a situation where robot has access to low dimensional sensory measurements and should learn to walk/run which is more challenging problem to address.

As the agent takes either pixels information or sensory information, it is required to take some actions. The agent chooses an action from an action space which can either be a continuous space or a discrete space. By default QWOP has discrete action space where keys Q, W, O, and P are the set of actions that an agent can choose from. However, the game can be modified for continuous action space. An example of continuous action space can be considered as the current values for motor control for joints. QWOP game is considered to be easy and hard if the game is designed such that the torque in joints are constant or the speed of the athlete is constant.

In this project, we consider low dimensional sensory measurements(from the athlete) as observations, and train the agent to play QWOP, both easy and difficult version, where the action space is discrete. We train agent to learn both hard and easy QWOP game and study the learning behavior. We note that training the agent to play QWOP with continuous action space is a difficult learning task and hence, consider it as an extension of our current work.

It might seem strange that why it might be even be remotely important trying to teach a machine to learn QWOP. But it is intuitive that Robot based AI for QWOP learning can be used to learn bipedal walking [4]. People have tried to solve the problem QWOP using reinforcement learning [5].

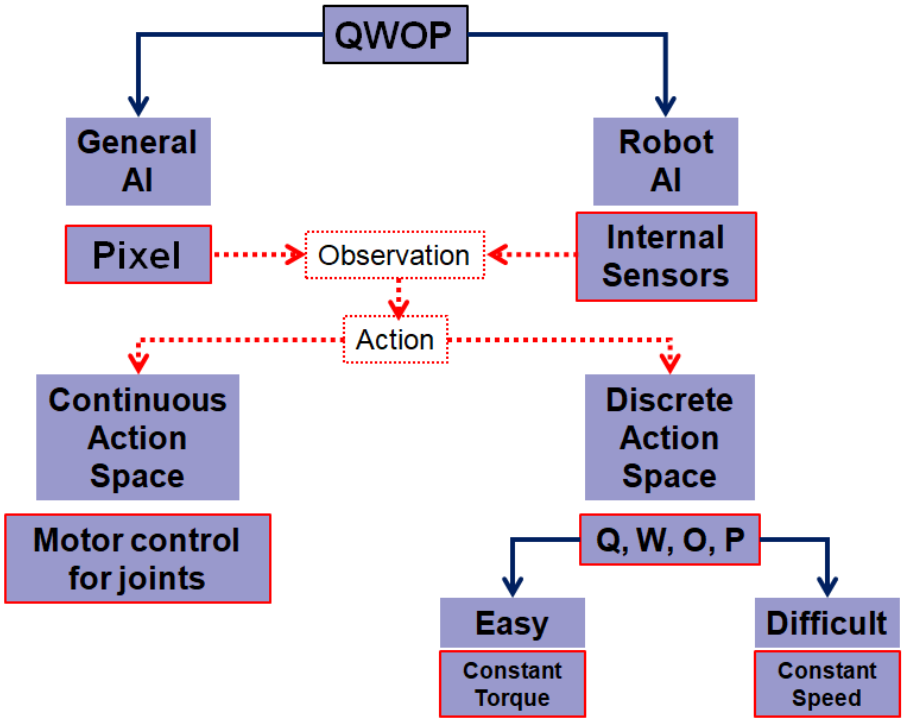


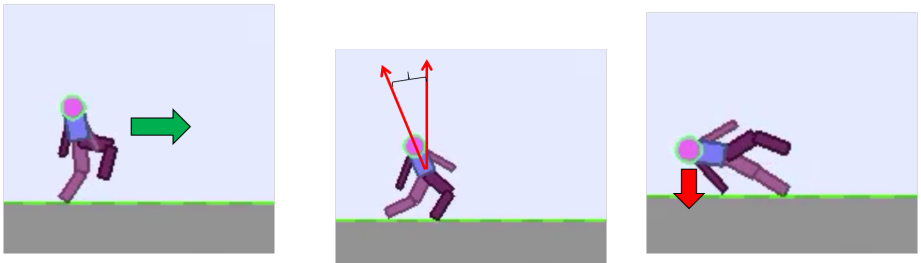

Figure 2: QWOP-RL scope

## 2.3 Reward Mechanism

An agent learns a task at hand by maximizing the expected reward it receives through out the testing process. Thus learning performance of an agent depends a lot on the design of reward mechanism. If reward mechanism reflect the learning objective at hand, the agent is expected to learn better. In this work, we have designed three different types of reward such that agent can only improve or cumulate more reward if it runs large distance with good body posture as quick as possible. We define reward in terms of distance traveled per action, angle of the body per action (stability and posture), and whether the subject falls or not. These rewards are summarized below:

- **Distance traveled Reward(DTR)**: This is a positive reward and larger reward is assigned to an agent if it travel a longer distance. This is a positive reinforcement reward.

$$\text{DTR} = + \text{Factor}_1 \times \text{distance}$$



(a) Distance Travel Reward (b) Posture Reward (c) Standing/Falling Reward

Figure 3: Design of Reward Signal.

- **Posture Reward (PR)**: This is a negative reward. This reward discourage an agent to travel horizontal distance without maintaining a good body posture. Here we measure PR through absolute value of angle made by the body with respect to vertical. This reward forces agent to run forward being as vertical as possible.

  PR= - $\text{Factor}_2\times$ absolute body angle

- **Fall Reward (FR)**: This is also a negative reward. This reward discourages subject/athlete to fall, which is basically end of game. We define FR reward as

  FR = - $\text{Factor}_3\times$ 100,

Factor1, Factor2 and Factor3 are constants which are adapted such that desired training performance is achieved The total reward acheived per episode by the agent is the sum of these individual rewards, i.e.,

- **Total Reward** = DTR + PR + FR

Final reward is rescaled between 1 and -1.

# 3 RL Algorithms

We consider two different types of RL algorithms, namely on-policy and off-policy algorithms, and try to learn QWOP based on these algorithms. Following are the two algorithms we consider in this project:

- SARSA [1]
- Deul DQN [6]

We briefly introduce these algorithms.

## 3.1 SARSA

It is an on-policy algorithm. The convergence of the algorithm requires all of the state are visited infinitely often. Hence, the estimation policy has some stochasticity in it and

hence there is variance in SARSA update as the action are not chosen deterministically. The update of SARSA algorithm is given by

$$Q(s,a) \leftarrow Q(s,a) + \alpha\Big[r + \gamma Q(s',a') - Q(s,a)\Big]$$

This update rule uses every elements that make up a transition from one state-action pair to the another state-action pair.

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize $s$
  Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
  Repeat (for each step of episode):
    Take action $a$, observe $r$, $s'$
    Choose $a'$ from $s'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    $Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma Q(s',a') - Q(s,a)\big]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until $s$ is terminal

Figure 4: SARSA- Algorithm

## 3.2 Deul DQN

Before introducing Duel DQN algorithm, we first introduce DQN algorithm. DQN is an off-policy algorithm. The update of $Q(s,a)$ in DQN is given by

$$Q(s,a) \leftarrow Q(s,a) + \alpha\Big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\Big]$$

In this algorithm, the learned values of $Q$ is guaranteed to estimate optimal $Q*$, optimal action value function independent of policy followed. The policy has only effect in the determination of state-pair visited and updated.

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize $s$
  Repeat (for each step of episode):
    Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Take action $a$, observe $r$, $s'$
    $Q(s, a) \leftarrow Q(s, a) + \alpha\left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$
    $s \leftarrow s'$;
  until $s$ is terminal

Figure 5: DQN-Algorithm

Duel DQN splits the DQN network into two channels as shown in the Figure 5 below. The network estimates Value function and advantage function separately which are combined in the outer stage to compute state-action values, i.e. $Q$ values.

$$Q(s, a) = V(s) + A(s, a)$$

where $V(s)$, and $A(s, a)$ are the value function and the advantage function, respectively.

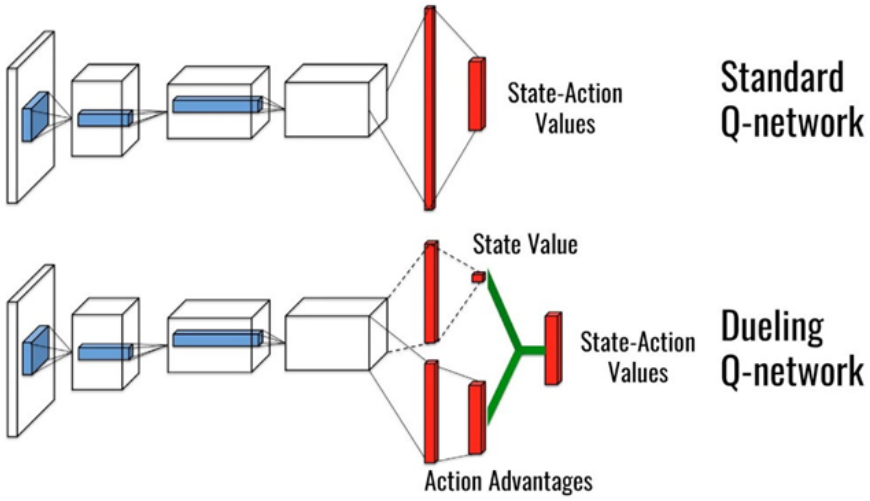


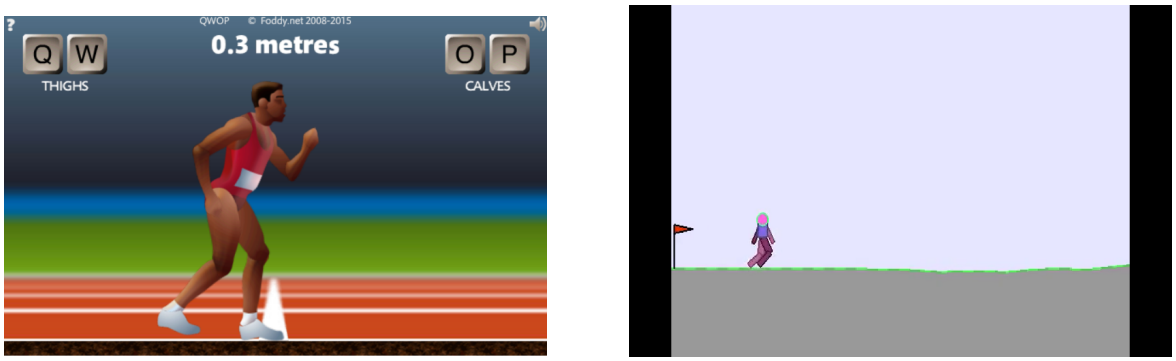

Figure 6: A single stream Q-network (top) vs the dueling Q-network (bottom)

Here we provide experiments where we plot Mean Q-values, Mean Rewards, Mean Absolute Error, and Loss, against Training episodes. Mean of Q-values, Reward, Absolute Error is taken over the training in an episode.

# 4 Experiments

In this section we discuss the experiments that we carried out in evaluating the learning performance of the agent. Frist, we note that the developers of the actual QWOP game were reluctant to provide us their code. Hence, we end up developing our own version

of QWOP. Figure 7 shows the snap shots of the actual QWOP games and our version of QWOP game. We emphasize that, in contrast of the actual QWOP game, our version has uneven running surface. This makes the learning task more realistic and challenging. The game was developed using python.

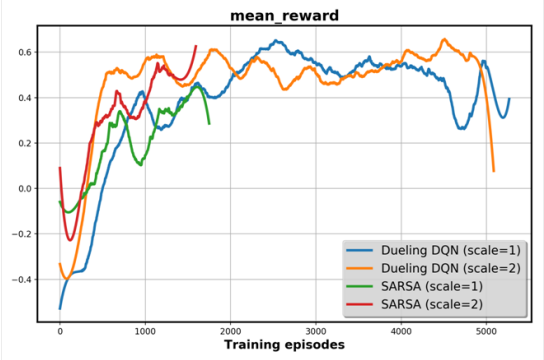



(a) QWOP (b) QWOP-ourVersion

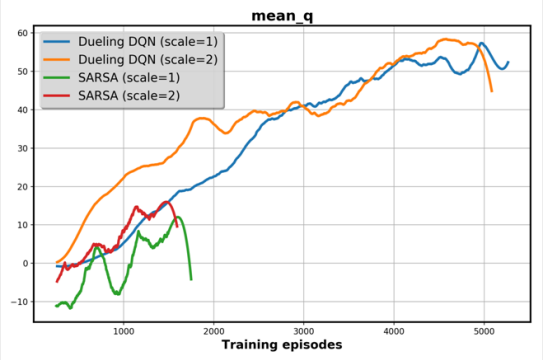
Figure 7: Snap shot of QWOP vs QWOP-ourVersion

We trained RL agent using two different RL-algorithms, namely SARSA and Duel DQN. We consider two neural networks that learns a mapping from state to action. The input to this network was the feature vectors. We implemented $\epsilon$-greedy algorithm which control the exploration vs exploitation rate. Initially $\epsilon$ was chosen to be large so that algorithm explore more to find better action for all the possible state, i.e., learn better policy. With time we decrease this exploration rate and greedly stick to the policy which give the better return of reward. In this project, we considered RL problem in Robot AI setup. Here we consider 24 dimensional state vector. Each element of the vectors were some sensory measurements like horizontal speed, vertical speed, hull angle speed, angular speed, leg contacts with ground, etc. First neural network has three hidden layers with 32, 16 and 8 nodes, respectively. The output layers has 4 nodes. In the experimental results we refer this network with 'Scale = 1' legend. The second Q-network consist of input layers and output layers with the same number of nodes and same number of hidden layers compared to network 1. But the number of nodes in each hidden layer is twice compared to that of network 1. We refer this network in the experimental results with legend 'Scale=2'. We note that we used Keras Wrapper [7] with Tensorflow [8] as its back end to design and create our networks and algorithms. The performance of RL algorithms is evaluated in GYM OpenAI reinforcement learning environment [9].

Figure 8 represents the Mean Q-values and Mean Reward for Dueling DQN algorithm and SARSA algorithms with scaling 1 and 2. As we implemented $\epsilon$-greedy algorithm, we let algorithms to explore first so that it can find the better policy. From the graph we can see that for initial 2000 training episodes there is significant improvement of the reward. This state represents the exploration states of the RL algorithms. After around 2000 training episodes, we see that there is not so much change in the reward values (in average sense). We can see the fluctuations in the reward values. This is because the best policy was not learned yet. However, we can say the agent has learned reasonable policies as the mean reward value is kind of constant after around 2000 training episodes. From the videos of the agent playing the game we presented in the class, we can say that agent had learned to play game. If we look into the mean-Q values, we can see that there is increase in Q values till the end of the training episodes. This implies

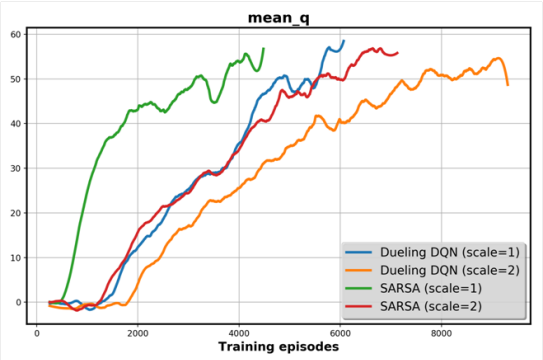



(a) Mean-Reward vs Tranining Episodes: Easy QWOP




(b) Mean Q values vs Tranining Episodes: Easy QWOP


(c) Mean Reward vs Tranining Episodes Hard QWOP




(d) Mean Q values vs Tranining Episodes: Hard QWOP

Figure 8: Mean Reward and Mean Q values of for Deuling DQN and SARSA Algorithms for Hard and Easy QWOP game

that the networks had not conversed to the optimal Q-values. So, if we had trained the networks for more training episodes, the agent would have better learning performance. The performance curves does not differentiate much between the hard QWOP and easy QWOP. However, the agent playing hard QWOP travelled short distance than the agent playing easy QWOP game.

We also implemented the recurrent version of Duel DQN and SARSA algorithms. But, the initial results from these algorithms were not encouraging. Further, the computational requirement of these networks was prohibitively high. Hence we decided to stick with Duel DQN and SARSA algorithms, only.

# 5 Discussion and Conclusion

In this course project, we developed a Robot based AI that learns to play our version of QWOP games, both easy and difficult versions. The input feature was composed of the sensory measurements. We considered discrete action space, i.e., keys Q, W, O and P were the only actions that were required to be learned by the AI agent. We accomplished in the objective of making the computer agent learn to play QWOP game. However, the experimental results show that the reward obtained by the agent per episode has not saturated. This implies that the agent was still learning the game. Had we trained longer, the performance of the agent could have been better. The RL problem with continuous valued action space is a more difficult problem and will be considered as a future work.

# References


[1] R. S. Sutton and A. G. Barto, *Introduction to reinforcement learning*. MIT Press Cambridge, 1998, vol. 135.

[2] B. Foddy, "Qwop [flash game](2008)," Retrieved 11-6-13 from http://www. foddy. net/Athletics. html, Tech. Rep., 2013.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[4] J. Morimoto, G. Cheng, C. G. Atkeson, and G. Zeglin, "A simple reinforcement learning algorithm for biped walking," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 3. IEEE, 2004, pp. 3030–3035.

[5] G. Brodman, "Qwop learning gustav brodman ryan voldstad."

[6] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.

[7] F. Chollet *et al.*, "Keras," 2015.

[8] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[9] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.