# Comparing Continuous Reinforcement Learning Algorithms Using OpenAI

Neda Ahmadi Diego Cabo Sudhakaran Jain Ruben Kip

*University of Groningen, KIM.ML09 Machine Learning*

**Abstract**

In this project, a comparison between the performance of two well known reinforcement learning (RL) algorithms is presented. CACLA (Continuous Actor-Critic Learning Automaton) and SPG (Sampled Policy Gradient) are two RL algorithms designed to work with continuous input and action spaces. The OpenAI Gym platform is used to test these two algorithms. OpenAI Gym consists of a set of different environments that aim to provide an easy-to-setup general-intelligence benchmark. Specifically, the *BipedalWalker-v2* [8] environment was selected to conduct the training and testing of both algorithms. The experimental results show better performance for CACLA in comparison to SPG, however the implemented SPG showed difficulties.

## 1 Reinforcement Learning

In the context of AI, reinforcement learning (RL) is one type of dynamic programming that is about learning what to do and how to map situations using a system of reward and punishment to train algorithms. An agent in a RL algorithm learns by interacting with its environment. By performing correctly, the agent receives rewards and there are some penalties for performing incorrectly. The goal is to optimize the agent's behavior based on the rewards that are provided by the environment. By maximizing these rewards and minimizing the penalties the agent learns a new task without intervention from a human. Additionally, the actions of the agent can also affect the environment, and make the searching process for the optimal behavior more complicated. Any actions are affected by the immediate reward and subsequent rewards discounted. [11, 3]. Figure 1 shows the interaction between agent and its environment.

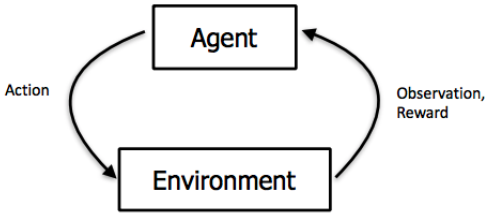



Figure 1: The agent observes the environment, takes an action to interact with the environment, and receives positive or negative reward [1].

### 1.1 Main points in Reinforcement learning

**Input:** An initial state from which the model will start
**Output:** As there are many solutions to a particular problem, there are many possible outputs

| REINFORCEMENT LEARNING | SUPERVISED LEARNING |
|---|---|
| Sequential decision making.<br>The output depends on the state of the current input and the next input depends on the output of the previous input | Decision is made on the initial input |
| Decision is dependent.<br>Labels are given to sequences of dependent decisions<br>Example: Go game | Decisions are independent of each other.<br>Labels are given to each decision.<br>Example: Object recognition |

**Training:** Based on the input, the model will return a state and wait for the response from the environment which can be a reward or punishment. The best solution is decided based on the maximum reward.

There are two types of Reinforcement [2]:

- Positive : When an event occurs as a result of a particular behavior, it increases the strength and the frequency of that behavior. In other words it has a positive effect when they have performed the desired behavior.
- Negative: By stopping or avoiding a negative condition, a behaviour becomes more strong

### 1.2 Exploration/Exploitation Trade off:

The trade off between exploration and exploitation is one of the challenges that arise in RL, and not in other kinds of learning. Balancing exploration and exploitation does not arise in the purest forms of supervised or unsupervised learning.

An agent in RL must prefer actions that it has tried before and found them effective in producing reward. But to discover such actions, it has to try actions that it has not selected before. In order to obtain reward, the agent should exploit what it has already experienced, but to make better action selections in the future, it also needs to explore new actions. The dilemma is that neither exploration nor exploitation can be pursued exclusively without failing at the task. The agent must try different possible actions and progressively favor those that appear to lead to better result. To have a reliable estimation of an action's expected reward, each action should be tried many times, when it is a stochastic task. Although mathematicians studied the explorationexploitation dilemma for many decades, it still remains unresolved. Exploration may use a simple strategy which is epsilon-greedy strategy. Based on that strategy, most of the time the agent takes the best known action (about 80% of the time), but occasionally explore a new, randomly selected direction even if it might be away from known reward. Epsilon is the percent of the time that the agent takes a randomly selected action instead of the action that is most likely to maximize reward given what it knows so far (in this case, 20%). Usually exploration rate is started with high rate (i.e. a higher value for epsilon). As the agent learns more about the environment and the most long-term reward collector actions, epsilon can steadily reduce to 10% or even lower as it settles into exploiting what it knows.

## 2 Compared Algorithms

As stated above, we chose to compare the performances of two well know reinforcement algorithms namely CACLA and SPG. For both these chosen algorithms, we used a technique named 'Experience Replay' [4, 9] to improve their performances. In experience replay every transition tuple ($s_t$, $a_t$, $r_t$, $s_{t+1}$) is stored in a buffer instead of directly being fed to the neural network for training. When the size of this buffer reaches its maximum capacity, the oldest transitions in it are replaced. $N$ random transitions from the buffer are extracted to train the network for every training step. For each of these transitions the target for ($s_t$, $a_t$) is calculated and then the network is trained on this mini-batch. Apart from this, we also made sure that all the state values are normalized before they are fed into the networks. We may further explore how these algorithms work in detail.

## 2.1 Continues Actor Critic Learning Automaton (CACLA)

CACLA[3] is an actor-critic algorithm and has both an actor and a critic. The actor is a multi-layer perceptron (MLP) that takes state representation as an input and outputs actions. The critic for CACLA is an MLP as well and also has the state representation as an input. The output of the critic is one value which indicates the total expected reward for that input state. Therefore this critic does not predict a Q-value of a state-action combination, it predicts how good a specific state $s_t$ is. The formula to train this critic on a transition ($s_t$ , $a_t$ , $r_t$ , $s_{t+1}$) using backpropagation is:

$$Target^{critic}(s_t) = r_t + \gamma \cdot V(s_{t+1})$$

In this formula $V(s)$ refers to the prediction of the CACLA critic of state $s$ and is trained with temporal difference learning [10]. Unlike most actor-critic systems, CACLA uses the sign of the temperoral difference error $\delta$ to decide if the actor will be updated or not, not the value. $\delta$ is calculated as follows:

$$\delta = r_t + \gamma V_t(s_{t+1}) - V_t(s_t))$$

If $\delta$ is positive, then the action was better than expected, and CACLA will increase the probability of picking that action again. If $\delta$ is negative, we do not train the actor but only the critic.

## 2.2 Sampled Policy Gradient (SPG)

Here, instead of changing the weights of the actor deterministically through backpropagation, SPG[12] samples actions from the action space for a specific state. These actions are set to occur more often if they perform better than the actions which are predicted by the actor for that state. The critic in SPG is a Q-value network. The critic is trained to predict Q-values of all the possible actions of every state. To predict the Q-value for an action in a state the critic is trained through backpropagation. The target for backpropagation for a state-action tuple $s_t$, $a_t$ of a transition ($s_t$ , $a_t$ , $r_t$ , $s_{t+1}$) is:

$$Target(s_t, a_t) = r_t + \gamma \cdot maxQ(s_{t+1}, a)$$

The actor network architecture is the same as in CACLA. CACLA is considered as a on-policy algorithm as it is trained on the actions taken in the transitions in the experience replay buffer, whereas SPG uses offline exploration to find better actions. Therefore, we can consider SPG as a off-policy version of CACLA. SPG has a unique way to explore actions from the action space and move towards them if they are good.

To calculate the backpropagation target for the actor, SPG compares the evaluation of the action of a transition $Q(s_t, a_t)$ to the evaluation of the action predicted by the actor $Q(s_t, (s_t))$, where $(s_t)$ denotes the actor prediction for state $s_t$:

$$\text{If } Q(s_t, a_t) > Q(s_t, (s_t)) : Target(s_t) = a_t$$

Here, any search algorithm can be used to find sampled actions with a better evaluation than $Q(s_t, (s_t))$. SPG can be extended further with offline Gaussian exploration (SPG-OffGE). SPG-OffGE creates a new sampled action $S$ times by applying Gaussian noise to the best action found so far. If the evaluation of this new sampled action $Q(s_t$, sampled action) is better than the evaluation of the best action found so far $Q(s_t$, best action so far), then this sampled action becomes the new best action. Initially, the best action represents the action predicted by the actor $(s_t)$ and the first sampled action is the action $a_t$ of the transition $i$. If the evaluation of the best action turns out better than the evaluation of the action that the actor would take currently, then that best action is made more likely to be predicted by the actor for state $s_t$. During training the standard deviation of the Gaussian noise decreases exponentially approaching a value close to zero.

The performance of SPG heavily depends on the accuracy of critic. The critic in SPG has to be accurate in the region of action space around the action that the actor currently predicts, $(s_t)$, to calculate a correct policy gradient. Due to the Gaussian exploration around $(s_t)$, the action space region is generally well explored, which makes the critic accurate in this region. One main advantage of SPG is that the sampling of SPG can be non-local from $(s_t)$, allowing it to escape local optima. To achieve this, SPG requires extensive exploration and a good critic to maximize its performance.

# 3 OpenAI and Bipedal Walker

For the environment and simulation we use the Bipedalwalker V2 [8] from the OpenAI gym. ”OpenAI is a non-profit AI research company, discovering and enacting the path to safe artificial general intelligence.” -OpenAI [7], offering free and payed simulated environments for reinforcement learning.

The Bipedalwalkver v2 is a continuous gym environment with a 2 legged walking robot. The state vector contains 24 parameters that can be checked in the environment manual[6]. The action vector consists of 4 normalized values between -1 and 1 that control the torque applied to each motor.
The reward system works as follows:

- Rewards are gained for moving forward. Roughly 300 for reaching the finish line.
- The robot gets -100 for falling down.
- Moving the motors generates a small amount of negative reward, so more efficient ”walking” techniques will get better results.

# 4 Results

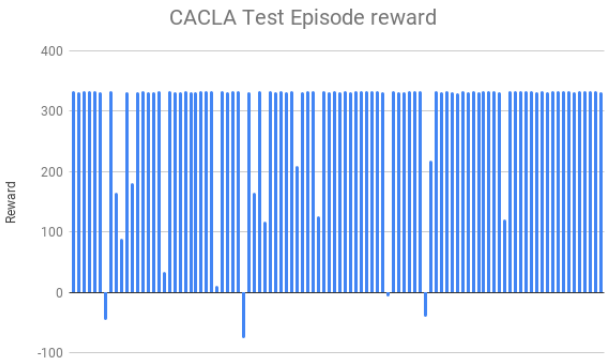
Both algorithms were trained for 20.000 epochs. The final episode reward, as well as the position in the X coordinate at the end of the episode were recorded during the training. After the neural networks for the critic and the actor were trained, both algorithms were tested based on:

- **Episode reward**: Total reward obtained up until the end of the episode.
- **Position X**: Horizontal location of the robot at the end of the episode.
- $\tau$ : Number of timesteps before the episode finished.

This test code was run a 100 epochs for each algorithm to obtain the final comparison graphs. In each test run, the results of course slightly differ, however mean, median and variance do not significantly change in each test run.

## 4.1 Test results

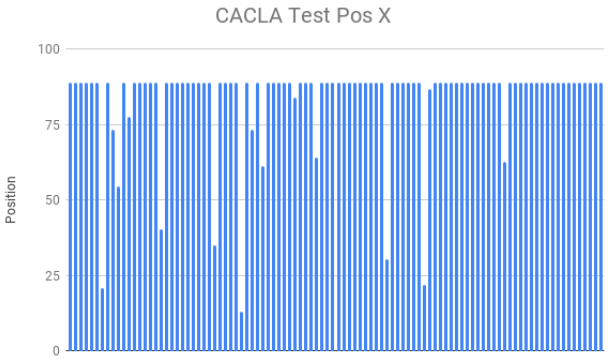
The reward for each epoch can be seen at Fig.2. We can see that the robot still falls sometimes and thus has some outliers in rewards going to 0 and negative. However, it is usually consistent having rewards around 330 points.




| | |
|---|---|
| Mean | 295.493 |
| Median | 332.664 |
| Std. | 96.151 |
| Max | 333.543 |

Figure 2: Episode rewards - CACLA

For the x-distance in Figure 3, we can clearly see the correlation with the reward where the robot falls. Again CACLA performs pretty well and seems to reach the finish line almost always or very close to it, with a median of $\tau = 843.6$




| | |
|---|---|
| Mean | 83.397 |
| Median | 88.709 |
| Std. | 15.709 |
| Max | 88.777 |

Figure 3: CACLA X-distance

## 4.2 Training Results

This graph shows the position of the robot for the training episodes. It can be seen that with SPG, the robot reaches the finish line (around 88) but then it forgets how to do it.

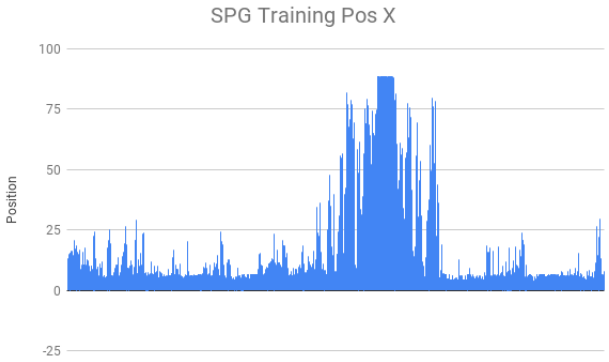



Figure 4: SPG-training-posX

Here we present cumulative reward over all the training episodes.

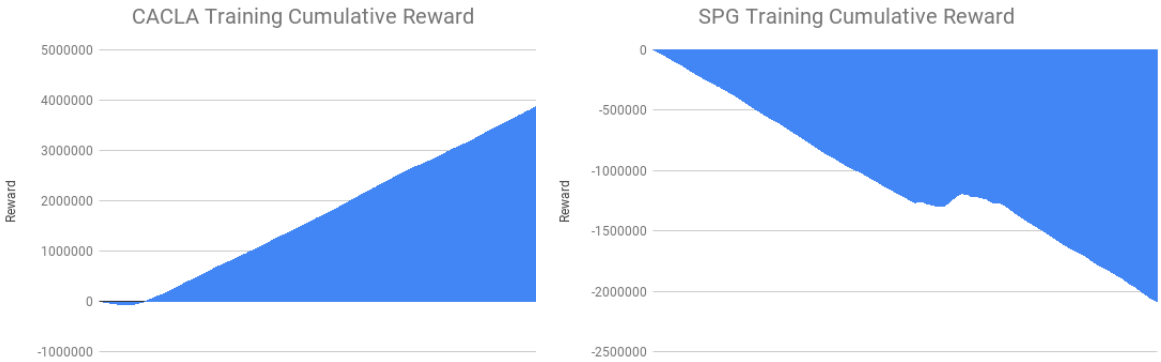



Figure 5: CACLA vs SPG cumulative reward

# 5 Discussions and Conclusion

We can see that CACLA performs extremely well in this environment. Starting to reach the finish line around the episode 1600-1700. The robot reaches the finish line in a very high percentage of the test trials and it takes a bit over 800 steps to reach the goal. Visually we can see the bipedal walker has learned to jump small gaps and even let itself fall a bit so it gets some starting speed without moving joints.

SPG did not perform as hoped for, learning to walk to the finish line sometimes but also learning worse actions in cases. The bipedal walker using SPG learns to walk to the finish line but then it seems to over-train and forget how to walk again (Fig.4 and Fig.5). This means the robot learns a good policy, that allows it to reach the finish line, but then it forgets it and the policy worsens. We now present several theories as to why this implementation of SPG did not yield the expected results.

One of the reasons SPG could not be performing well, is because we created our own experience replay buffer. This buffer uses random sampling to select the transition tuples. As an improvement to the classical experience replay, Schaul et al. [9] developed prioritized experience replay (PER). Applying PER implies that transitions which the value network is bad at predicting will be replayed more often, which was shown to lead to faster learning and better final performance in [9]. This was used in [12] and it could be one of the reasons why our implementation of SPG was not succesful.

Another well known technique to improve the stability in RL combined with artificial neural networks, is the usage of a target network [5]. As the training of Q-learning maximizes over all possible actions taken in the next state, the combination of this training method with function approximators can lead to the deadly triad [11]. This deadly triad leads to a high probability of the Q-function to diverge from the true value function over the course of training through a positive feedback loop. We strongly believe this is the main reason why SPG forgets how to walk after having reached the finish line.

Due to technical difficulties when implementing these 2 algorithms, we did not have much time to adjust the hyperparameters. As a result, we hope the performance of not only SPG, but CACLA as well, can be improved further by tweaking these parameters.

# References


[1] John Schulman Pieter Abbeel. Deep reinforcement learning. Available at `http://rll.berkeley.edu/deeprlcourse-fa15/`.

[2] R. A. Griggs, editor. *Psychology: A concise introduction (2nd ed.)*. Worth Publishers, New York, USA, 2009.

[3] H van Hasselt and Marco A Wiering. Reinforcement learning in continuous action spaces. 2007.

[4] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. machine learning. 1992.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[6] OpenAI. *Bipedalwalker v2 github page*, (accessed February 12, 2019). Available at `https://github.com/openai/gym/wiki/BipedalWalker-v2`.

[7] OpenAI. About openai, (accessed February 4, 2019). Available at `https://openai.com/about`.

[8] OpenAI. Bipedalwalker v2, (accessed February 4, 2019). Available at `http://gym.openai.com/envs/BipedalWalker-v2`.

[9] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[10] Richard S. Sutton. Learning to predict by the methods of temporal differences. machine learning. 1988.

[11] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[12] Anton Orell Wiehe, Nil Stolt Ansó, Madalina M Drugan, and Marco A Wiering. Sampled policy gradient for learning to play the game agar.io. *arXiv preprint arXiv:1809.05763*, 2018.

# A Hyperparameter values

| **HYPERPARAMETER** | **CACLA** | **SPG** |
|---|---|---|
| Reset environment after (n. steps) | 1600 | 1600 |
| Discount rate | 0.99 | 0.99 |
| Total training episodes | 20000 | 20000 |
| Experience replay capacity | 4000 | 4000 |
| Training batch size | 32 | 32 |
| N. iterations | 100 | 100 |
| Optimizer | Adam | Adam |
| Loss Function | mse | mse |
| Activation Function Hidden Units | ReLU | ReLU |
| Number of hidden layers | 2 | 2 |
| Number of units 1st hidden layer | 400 | 400 |
| Number of units 2nd hidden layer | 300 | 300 |
| Critic Activation Function Output | linear | linear |
| Actor Activation Function Output | tanh | tanh |
| Critic learning rate | 1e-3 | 5e-4 |
| Actor learning rate | 1e-4 | 5e-4 |
| Sigma(gaussian noise) | 1.0 | 1.0 |
| Sigma min | 0.1 | 0.1 |
| Sigma decay | 0.9995 | 0.9998 |
| N. sampled actions | N/A | 3 |

# B GITHUB repository

Link to the code and a GIF of the final result: `https://github.com/RUKip/MachineLearningProject`