Deep Deterministic Policy Gradient for Locomotion Task in Continuous Environment

Maxim Shelopugin
University of Central Florida
Orlando, Florida
maxim.shelopugin@knights.ucf.edu

ABSTRACT

This paper tries to explore and evaluate the Deep Deterministic Policy Gradient algorithm in continuous environment on the example of OpenAI Gym Bipedal Walker-v2. Extensive testing and adjustment of hyperparameters are discussed as well.

CCS CONCEPTS

• **Computing methodologies** → *Sequential decision making*; *Learning from critiques*; *Neural networks*;

KEYWORDS

DDPG, Actor-critic, Reinforcement learning

ACM Reference Format:
Maxim Shelopugin. 2018. Deep Deterministic Policy Gradient for Locomotion Task in Continuous Environment. In *Proceedings of ACM Conference (Conference'18).* ACM, New York, NY, USA, 5 pages.

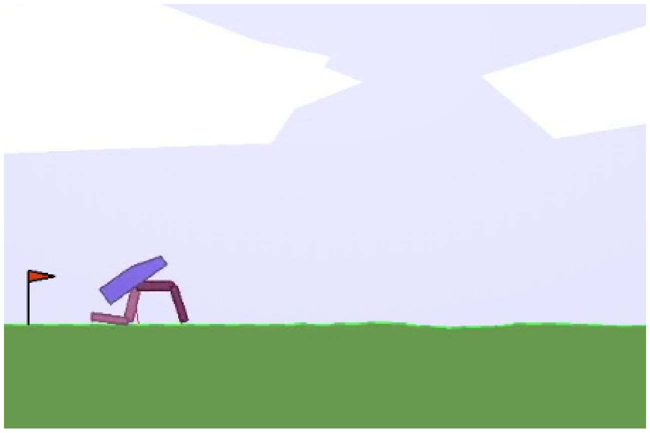
Figure 1: BipedalWalker environment

1 INTRODUCTION

Reinforcement learning is the paradigm in machine learning domain. Its goal is to imitate human intelligent behavior by agents with the help of rewards which are given if the agent reaches desirable state. This is done by the usage of various approximation techniques which measure the *quality* of the state and often the main idea is to distinguish between how bad or good is an action given the observation. The notion of *quality* of the state comes with the cost - agent needs to visit it numerous times to make sure that the judgments about the state are correct. Of course, visiting the same state will not help with the overall knowledge of environment, hence various exploration-exploitation techniques are applied in deciding the next action to be taken. This paper describes and tries to address a specific difficult problem of trying to teach a robot to walk with the help of reinforcement learning algorithm. Specifically an actor-critic DDPG model[2] will be trained and evaluated on the OpenAI Gym environment BipedalWalker-v2 shown on the Figure 1.
Reinforcement learning became more popular recently due to the increase of datasets it can be trained on. Mainly, since the data do not need to have labels nor algorithms need any supervision, rather a reward function, therefore most games which can be simulated on the computer can be used as great playgrounds for reinforcement learning experiments. To apply the methods in practice however, many things need to be addressed and more strict assumptions need to be made. Great success of the usage of non-linear function approximators like deep neural networks had its part in the resent interest as well. Deep Q Networks for example, were popularized by DeepMind[4] and successfully applied to the variety of problems, which attracted the attention of researches around the world. This paper specifically tries to solve the BipedalWalker-v2, which relaxes requirements such as stochasticity and rather focuses on continuous in and output spaces - this is far from real world applications, but nevertheless gives a great complicated example of a problem which solution can be useful.

2 RELATED WORK

Advancements in the field of deep reinforcement learning began with Volodymyr Mnih et al.[4] especially with their deep Q-learning agent, which could play 2600 Atari games without any prior knowledge of any of the environments. The architecture of that model consisted of Deep Convolutional Neural Network which is flattened and connected to dense layers, which try to approximate the expected sum of future discounted rewards starting from a specific state $s \in S$ taking an action $a \in A$. The approximation which is done is defined by the Bellman Update:

$$Q(s,a) \leftarrow (1-\alpha)\cdot Q(s,a) + \alpha \cdot (r + \gamma \cdot \max_{a'} Q(s',a')) \qquad (1)$$

Where α is learning rate, γ is a discount factor, r is a reward returned by the action and s', a' are observed next state and action taken from the observed state respectively. The Equation 1 shows the model to update a single state-action pair by the summation of rewards received and the prediction of the value of the next state. This in a sense means that the update is recursive - the accuracy of the model depends on itself, and as it is seen, the model assumes MDP (Markov Decision Process) property of the system, which

Conference'18, December 2018, Orlando, FL, USA
2018.

namely means that the state can be defined by a single previous state. The resultant network learns a non-linear mapping of state-action pairs to values without explicitly learning a policy since it can be derived from the learned values themselves. This has proven to be effective, however has a couple of obstacles. Since the relationship between the state action pairs and expected rewards is complicated, the resultant network might take a rather long time to learn. In fact, there is no guarantee that it will converge at all, which is a consequence of another major problem with the model - it does not look at actions in isolation, so a specific action with a bad influence on the environment may be smoothed out by other actions in the action sequence. Hence there is no guarantee that the network will detect a particularly bad action and would keep propagating it throughout the training. When training a model, iid - independent and identically distributed data is assumed, however in game environment states are highly correlated due to their temporally dependent nature. To counter this, authors used a *replay buffer* - a data container which stores experiences. Experience is represented by a tuple (s,a,r,s'), and a random batch of such tuples is sampled each time for training purposes. Since the system is assumed to be Markovian, only the information about current and next states are enough in the tuple.
Although DQN proved itself to be very effective in continuous input spaces, it fails to work in continuous action spaces since the DQN learns the state action values explicitly, and in continuous action spaces it needs to output potentially infinite amount of values. This is a problem, since many game and real world environments require determining actions in continuous spaces, which is addressed by Timothy Lillicrap et al. in their expansion of DQN which they call DDPG[2]. DDPG, or Deep Deterministic Policy Gradient agent summarizes the value learning model of DQN and uses it to learn the mapping of spaces to actions. It uses a distinct model to approximate the policy π and improves the accuracy of the policy by the feedback of its value-based counterpart. In other words, q-learning network does not need to consider every possible action now, it already receives the action dictated by the policy. DQN is used just for the evaluation of the action. This family of algorithms is called Actor-Critic algorithms, where the actor learns the policy by applying the policy update:

$$\nabla_{\theta^{\pi}} J = \frac{1}{N}\sum_{i=0}^{n} \nabla_a Q(s,a|\theta^Q)|_{s=s_i,a=\pi(s_i)} \nabla_{\theta^{\pi}} \pi(s|\theta^{\pi})|_{s_i} \quad (2)$$

where π is a policy, and $\nabla_{\theta^{\pi}} J$ is a gradient of the policy, which the critic improves. Equation 2 describes the error of the specific policy as an average error of each state-action pair value given the action dictated by the policy. As it is seen, now the model does not suffer from overestimation of quality of specific action in isolation, because it considers each action on its own. Since now the model needs to fit two non-linear functions independently, some improvements need to be made to increase the chances of convergence. The prediction of the q-values is done by the usage of target policy $\tilde{\pi}$, which is a separate policy model. The idea behind the usage of separate models for the update increases stability because otherwise the update of policy would be dependent on the policy itself. To increase the accuracy even further, Critic has a separate target model ̃Q too. To keep the target models accurate, they need to be updated throughout the execution, however the degree of the update needs to be small - this keeps target values more stable. Authors use the idea of *soft update* for the target networks: the weights of both networks are pushed by a parameter $\tau \in (0,1)$ towards the weights of critic and actor. This results in these update equations:

$$\theta^{\tilde{\pi}} = \tau\theta^{\pi} + (1-\tau)\theta^{\tilde{\pi}}$$

$$\theta^{\tilde{Q}} = \tau\theta^{Q} + (1-\tau)\theta^{\tilde{Q}}$$

This results in gradual improvement of the policy but does nothing to explore the unknown states. To alleviate the usage of greedy exploration-exploitation decision making, authors introduce noise $\mathcal{N}$ to all actions. $\mathcal{N}$ is chosen via Ornstein-Uhlenbeck process and is added to all the actions during the decision making of the policy. This results in an agent that learns optimal policy in a stochastic environment, however for testing purposes the noise is removed and agent acts in deterministic environment. The final optimization that they made to the algorithm is the usage of *batch normalization*. Since the algorithm is supposed to work in various continuous environments, it is expected that the ranges of each specific input space would be significantly different, hence normalizing all the inputs to the same range would greatly improve the stability and performance across all testing environments.
To continue the success of DDPG Arun Kumar et al. implemented it in the realistic setting of bipedal walker[1] in Gazebo simulation. Techniques for the actor-critic algorithms mirror the techniques of Timothy Lillicrap et al.: the concepts of noise, batch normalization and replay memory were used as well. They payed an explicit attention to the "realisticity" of walking and compared the joint rotations of humans to the ones of the simulated bipedal robot and tried to minimize the difference. As expected, the simulation was successful and within 41 hours of training time was able to mimic the human movement.
Some notable mentions in the field of continuous control are Xue Bin Peng et al., with their task of locomotion in more sophisticated environments[5] and Volodymyr Mnih et al. with the usage of Asynchronous Actor Critic[3]. Xue Bin introduced the concept of spatial similarity while sampling the memory batches - instead of random sampling, use the N closest spaces together to display the varying actions. It was successful enough to enable the 21-degree of freedom dog robot to traverse through the highly variable landscape. Mnih et al. had an improvement on a previously advised memory batch as well. Instead of storing each transition in the memory, they used a varying number of agents playing asynchronously in the same environment. Since it is very probable that the agents are exploring different state spaces due to the random initialization, this results in truly independent data samples, and leads to the faster convergence even while using a single CPU as a hardware.

3 METHOD DESCRIPTION

Simulated continuous and stochastic environments often model the real-world situations with high precisions, and it is believed that agents that do well in simulations will do well outside of it. Recent advancements in reinforcement learning are related to the diverse simulations and games that are released on a regular basis, and hence introduce great variety of playgrounds for testing the reinforcement learning algorithms. This paper focuses on the ability of Deep Deterministic Policy Gradient algorithm to solve a high dimensional continuous environment from OpenAI Gym - BipedalWalker-v2. The design of the algorithm will be addressed in this section, however it is necessary to understand its counterparts first: Q Learning and Policy Gradients.

3.1 Deep Q Learning

One of the earliest and most popular reinforcement learning algorithms is tabular Q-learning. The algorithm tries to approximate the sum of discounted future rewards given the state and action which is precisely described in the Equation 1. Tabularity in its case just signifies the discretization of input state and output action spaces. Tabular Q Learning algorithm creates a table where each entry represents a specific state and action which can be taken from this state, and by repeatedly applying Bellman Equation to it, it quantifies the *quality* of the action given the space. Obviously if the input state space is continuous, it is impossible to construct the table due to it being infinite. To alleviate the problem of continuity, due to the most problems having continuous state spaces, a more versatile approach needs to be taken. This resulted in a great success of Deep Q Learning[4]. Instead of creating a table with state-action values, it is possible to approximate the non-linear function which would map any real valued state to a value of each of the actions. This would result again in quantification of the *quality* of each of the actions. To depict such a complicated relationship, neural networks are used. Since the function is obviously nontrivial, and the Bellman Update assumes recursion of the approximation, this model is never guaranteed to converge. However, it still produces surprising results and solves many discrete action space environments given enough training time and a proper set of parameters. Increasing the probability of convergence of such a complicated model might be difficult, however many attempts to improve the learning stability were made. One of them was the introduction of the independent target network. This results in the DQN agent which is updated on its target counterpart. Target network although needs to be updated as well, does not need to be updated as often. This results in DQN agent steadily getting closer to the target network, and when the target is updated, the chase starts over. Even with this major improvement, there is an architectural disadvantage of Q learning networks - the estimation of the cumulative rewards beginning with a state. This means that the DQN learns the "chain" of actions and is only concentrated on the "big picture". Even if the chain would have a single worst possible action in it, which is outweighed by many good actions in the chain, the network will never see that bad action. The result is a strategy which might be good, but surely not the best one.

3.2 Policy Gradients

Disadvantages of Q-learning motivate researches to approach the problem from the different perspective. Rather than learning a complicated function which would tell us how one action differs quantitatively from other actions given the state, just learn the best action to make. This might sound counter-intuitive, since this is what DQN does anyway, however when the action space is continuous DQN would have to learn the mapping from a single state to all possible actions - which are continuous and hence infinite. And since we do not really need to know the "value" of each of the actions anyway and would rather know which action is the best, it is much easier to map a state to the single best action. During evaluation process we can decide if the action should be changed with some exploration-exploitation policy and if the new action is better than the old one, we update the model with new action distribution. This results in an agent that can map continuous spaces to continuous actions. This approach is called Policy Gradient. Since every action is considered in isolation, it does not have the "big picture" problem, and if it detects the particularly bad action, it updates it resulting in a better policy. All these improvements might sound exciting, but policy-based methods have a major disadvantage as well. Since each action is considered in isolation, it might take a very long time to consider each one of them, hence convergence takes a long time. But the convergence in this scenario is guaranteed. To help the policy converge faster, a more sophisticated method needs to be applied.

3.3 Deep Deterministic Policy Gradient

Since the DQN tends to converge faster although not to the true maximum and policy-based methods learn the true optimal policy although slower, it is possible to use the advantages of both. This results in a complicated actor-critic algorithm called DDPG[2]. This method creates two agents, one which tries to decide on the action - policy based, and the other that takes the made decision alongside with the current state and tries to evaluate its value - value based model. Result is a double neural network architecture that learns not only the single best action, but also depicts how good is the action in the global scenario. Actor - policy based agent, gets updated by the critic - value based agent, and the changes of the critic are propagated back to the actor.

To improve the stability of the said algorithm a couple of methods described earlier were applied in the implementation. Since the environment is highly correlated - an action from the specific state results in a very similar state and the assumption of independence of data is violated, hence the replay buffer is used to sample the independent experiences. To increase the stability of learning, the soft update scheme is used as well - now target critic and target actor are separate neural networks. To make the distinction between the exploration and exploitation, each action predicted by the actor was adjusted by some noise, so that the actor learns to play in stochastic environment. Once it is done training in stochastic environment with noise, it performs very well in deterministic environment.

Policy and value both are approximated by neural networks. They need to be large to represent the complicated relationships even between the simple BipedalWalker-v2 environment. Actor is a neural network with three hidden layers. Input space is 24-dimensional and is propagated through 256, 128, 64 wide layers to the four-dimensional action output space. All activations are ReLu to combat the curse of dimensionality, and output activations are tanh - since the motor commands are continuous in range (-1,1). Critic is a neural network with three hidden layers as well, but since it needs to consider both action and state, it starts as 2 separate networks which merge together in the third hidden layer. It then outputs the expected Q-value of the state action pair which is used for the critique of the said action. All activations are again ReLu, except for the output node - which is linear due to the output constraints. Backpropagation is done as described earlier and the loss is computed by the mean squared error between the critic and target critic networks.

4 TESTING ENVIRONMENT AND EVALUATION

As it was said above, the DDPG is run on BipedalWalker-v2 OpenAI Gym environment. The inputs are continuous 24-dimensional vectors and outputs are 4-dimensional actions of velocity applied to each part of the joints. Reward function penalizes the walker by -100 points for a fall, has a small penalty for each of the motor torque applications and reward by 300 if the agent reaches the end of the track. To force the agent to finish the race as soon as possible, γ was set to be 0.999 - this results in very high appreciation of future rewards even from the starting state. To keep the target networks as stable as possible, very low τ of 0.001 was used - this results in very slow oscillation of target values resulting in very stable, however very slow learning. Memory buffer has a constant length of 100.000 and newer experiences are overwriting the older ones. Batch size is kept at 128, and network is initialized with output weights having the mean of 0 and bounded by (-0.003, 0.003). Test environments were run on Intel Core I7 4790k machine with GeForce GTX 1080. All linear computations were run on the GPU with CUDA 9.0. Overall testing was done in epochs, each epoch consisting on 10 runs. The tests were conducted after each epoch with noise = 0. Results presented here are smoothed over 10 consecutive runs to produce more smooth representations. Overall each experiment took around 8 hours of training. To study the behavior of the learning, various learning rates were used throughout the study. Performance of the relatively high α of 0.001 and very low α of 0.00001 are compared in this paper. As it is seen from the Figure 2, lower rates of alpha tend to learn in spikes. First the walker tries not to fall at all, then the performance falls when it tries to learn to finish the track. Once it finishes the track, it is seen that the performance is dropped again due to the attempt to optimize the motor movements. While attempting said optimization it fails to move with stability, and hence falls often. Overall, although it solves the environment, the variability of performance shows that the walker is very unreliable - even when it can solve the environment in one run, it can fail drastically in the next. To address the stability of learning, α was lowered to 0.00001 in the Figure 3. As it is seen the performance is incredible. Walker does not even try to finish the race unless it is confident about its movements. But once it learns the overall patterns of movements, it immediately solves the environment and keeps the optimal performance throughout the next 300 epochs of execution.

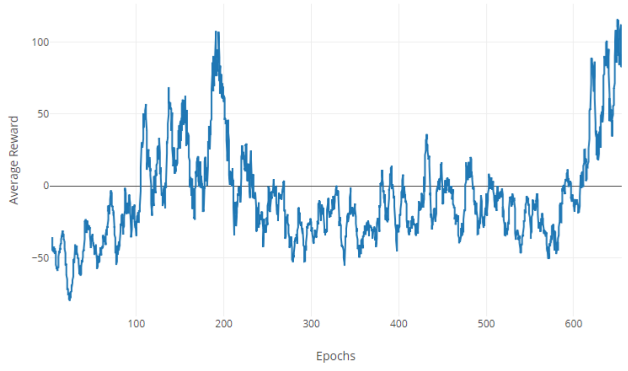

Figure 2: Performance with α = 0.001.

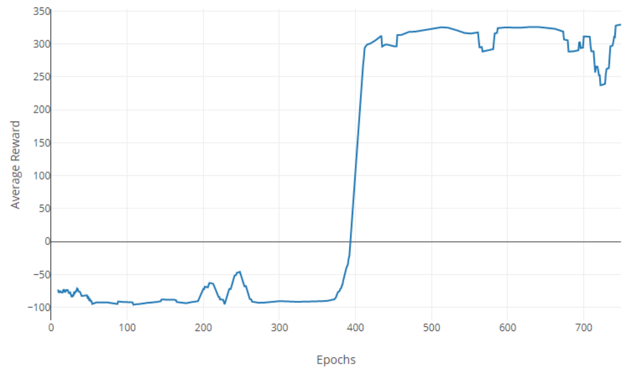

Figure 3: Performance with α = 0.00001.

5 DISCUSSION

As it is seen on the graphs on the Figures 2 and 3, the bipedal walker is easily solvable with the DDPG. To increase the stability a lower learning rates and softer updates were used and although the execution time was increased by a big margin, the results are very satisfactory. To test the robustness, some extreme sets of hyperparameters were used as well and the results are interesting: the DDPG can solve the BipedalWalker-v2 in less than a 1000 runs, which is under an hour of training time. However, such solutions are very unstable, and might make the walker fall even within the first couple of frames. The reward function prioritizes agents that walk "optimally" with as few movement commands as possible, and as it was demonstrated lower learning rates learn such a policy from the start without the intermediate "clumsy" state, while higher learning rates tend to solve the environment no matter how disorganized the movements are. The only optimization which was not applied here was the batch normalization, which in principle can significantly improve the performance, but was not applied due to the specificity of the domain. Since the environment throughout the experiment does not change there is no need to generalize the algorithm and the network can be specifically tuned to perform well on this instance.

6 CONCLUSION

Although the more traditional reinforcement learning techniques are still useful and can be applied to the wide variety of environments, deep neural networks show the advantage of approximation of non-linear functions even in this domain. The resultant architectures can learn hard relationships between arbitrary states and optimal actions, which sounds very impressive and looks even more impressive in practice. Actor critic networks, being the state-of-the-art reinforcement learning agents are proving themselves to be great in learning stable and optimal policies without any prior knowledge of the system, and with some architectural improvements can increase the domain of their usability. In future it is desirable to improve the architecture of the network further and apply the improvements for even harder environment like BipedalWalkerHardcore-v2.

REFERENCES

[1] Arun Kumar, Navneet Paul, and S N Omkar. 2018. Bipedal Walking Robot using Deep Deterministic Policy Gradient. (07 2018).

[2] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *CoRR* abs/1509.02971 (2015).

[3] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48 (ICML'16)*. JMLR.org, 1928–1937. http://dl.acm.org/citation.cfm?id=3045390.3045594

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari With Deep Reinforcement Learning. In *NIPS Deep Learning Workshop*.

[5] Xue Bin Peng, Glen Berseth, and Michiel van de Panne. 2015. Dynamic Terrain Traversal Skills Using Reinforcement Learning. *ACM Trans. Graph.* 34, 4, Article 80 (July 2015), 11 pages. https://doi.org/10.1145/2766910