Laboratório VISGRAF
Instituto de Matemática Pura e Aplicada

Deep Reinforcement Learning for High Level Character Control

Caio Souza
Luiz Velho (supervisor)

Technical Report TR-20-05 Relatório Técnico

March - 2020 - Março

DEEP REINFORCEMENT LEARNING FOR HIGH LEVEL CHARACTER CONTROL

CAIO SOUZA, LUIZ VELHO

Final Version as of March 12, 2020 (`classicthesis`).

CONTENTS

1 Introduction 1
  1.1 Motivation 1
2 Environment and Agent Modeling 2
  2.1 Environment 2
  2.2 Task 3
  2.3 Agent 4
  2.4 Reward 6
3 Training 7
  3.1 Configuration 7
4 Results 9
  4.1 Training Convergence 9
  4.2 Discussion 9
5 Conclusion 12

References 12

1 INTRODUCTION

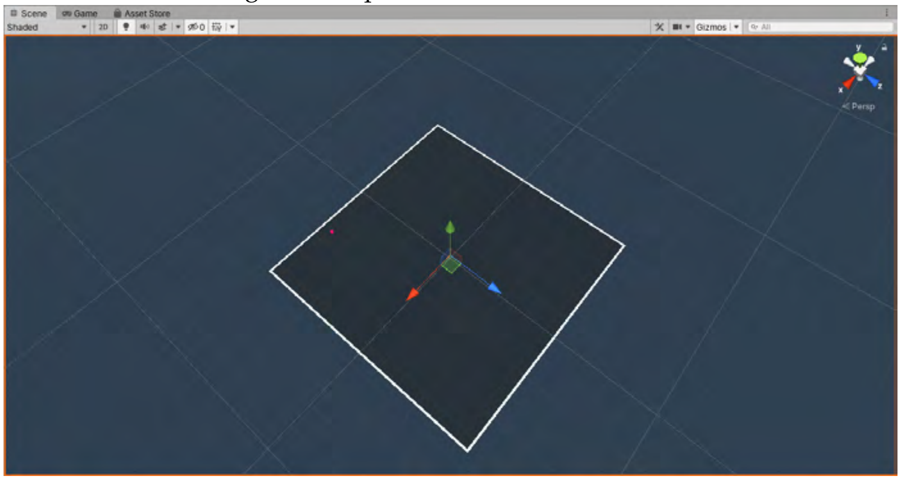
Reinforcement learning have been a trending topic recently with the dissemination of Deep Learning techniques. Various interesting applications have been made, from playing games like Atari[4], Go[6] and StarCraft[9], to controlling actuators (muscles) on full body motion control simulation[3] or a robotic hand solving Rubik's cube[1], those are some examples in a vast range of applications. Here we want to experiment with high level controls of a character in a 3D environment to do simple tasks like collect or fetch an object.

The chosen character is a dog named *DogBot*. It has a set of animations (stand, walk, trot, run, jump, etc) and a heuristic traditional controller. Deep reinforcement learning was applied on it to select between the aforementioned actions at each instant t_i to accomplish a desired task. That said, a later goal of experimenting with reinforcement learning for high level control is to have basic trained behaviors which can be used to interact with players in a virtual environment.

1.1 *Motivation*

While there are plenty of successful cases of low-level training, developing such learning environments and characters demands a lot of additional work compared with the standard animation stack and also requires fine tuning. For such cases, achieving a desired artistic movement/animation can be challenging or not viable at all.

Figure 1: Top View from environment.



The mixed use of heuristic, machine learning and traditional animation simplifies both the learning process and the artistic control over the final result. It gathers the best of each world: the traditional animation and heuristic gives the artistic control over the set of actions whereas the machine learning gives the possibility of the agent generalizing for many situations.

With this plot, our approach differs from the others using high level control such as the 2D Atari games in its fundamental principle: it is not solving a existing game, but building its own game environment and rules, which gives the freedom of balancing between the learning difficult (generalization) and the control over the actions (static heuristic and animations). All that with full access to the underlying environment model[1].

2 ENVIRONMENT AND AGENT MODELING

The modeling of the virtual environment and agent was made with Unity 2019.3 using its physics engine simulation.

2.1 *Environment*

The environment is a Unity scene where the character can act and interact with objects. It consists of a square gray plane 110×110m with a white border of 1m diameter, which visually delimits the area of interest. While it is possible to walk past this area, if the agent go past it a final state (game over) is reached leading to a reset or restart. The figure 1 shows the top view of the environment in the Unity editor.

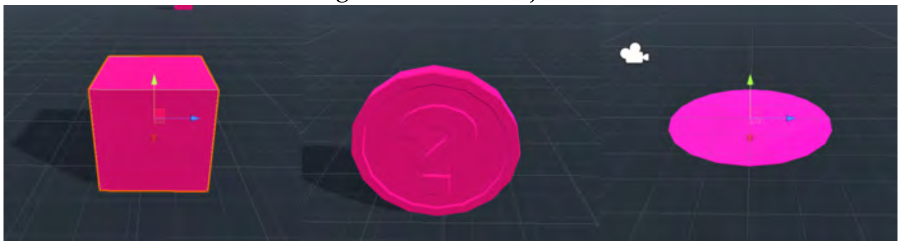
Inside the delimited area the following objects can be found (figure 2):

- Collectibles
  - (Simple Geometry) Cube with 1m edge
  - (Complex Geometry) Coin with 1.5m diameter
- Standing Pad
  - Circular region with 5m diameter

The choice of objects is arbitrary, but this is an advantage of virtual environments: What is seen by the agent don't need to be the same which is seen by a player, hence the freedom of using simple objects and rendering for the

[1] Full access to the environment model gives the possibility of shaping the reward process easily and possibly avoiding or softening the fundamental problem of reinforcement learning: The credit assignment[7].

Figure 2: Scene objects.



agent, albeit a complex scene may be displayed to the player. Another important point is the size of the objects, with later usage of visual information at low resolution (84×84) rises the need of big objects.

Environment Observability

An important conceptual distinction of environments is whether it is completely or partially observable. Although it's described as an environment property here, it is directly related to the agent's perception of its world.

- Completely Observable - All information of the environment state is observable.
- Partially Observable - Some of the information of the environment state is observable.

One example of a completely observable environment is:
Assuming the previous environment area without obstacles and with a single collectible object as goal. If the observations are set as the agent position and direction, the direction to target, target position and border distance and position. It is a completely observable configuration in the sense that independent of the agent state (position and direction) it always have complete information of itself and the target to complete its goal[2]. Indeed one could even write a heuristic to solve this simple task.

In the other hand, following the same example, if only visual information is used, (i.e, a 2D image from the agent's vision cone), depending of the agent position it may or may not "see" the target, neither know its exact position. In this case the sensing of the agent varies depending of its state and it is always a partial view of the environment which may or not contain relevant information.

This differentiation of how the environment is perceived and how much information is available per observation is an important component of learning performance.

2.2 *Task*

The agent's goal can be put as a general mobility task, given a stimulus it needs to reach the target point. Inside this general task three specialized tasks are implemented:

- Collecting an object: reach the object position, when the agent collides with it the object is removed from the scene, i.e., collected.
- Reach and stay: reach the standing pad and stay inside it.
- Fetch: reach the object and go back to its initial position. It also can be thought as reaching two objects, for example, the stick and then who threw it.

[2] "All information" is relative to the important information to complete a given task. While it is not possible to draw the complete scene from the given observations, it is sufficient to complete the task at any state/time without the need for information such as object shape, color, etc.

Figure 3: Agent abstraction as a black box with input/output.

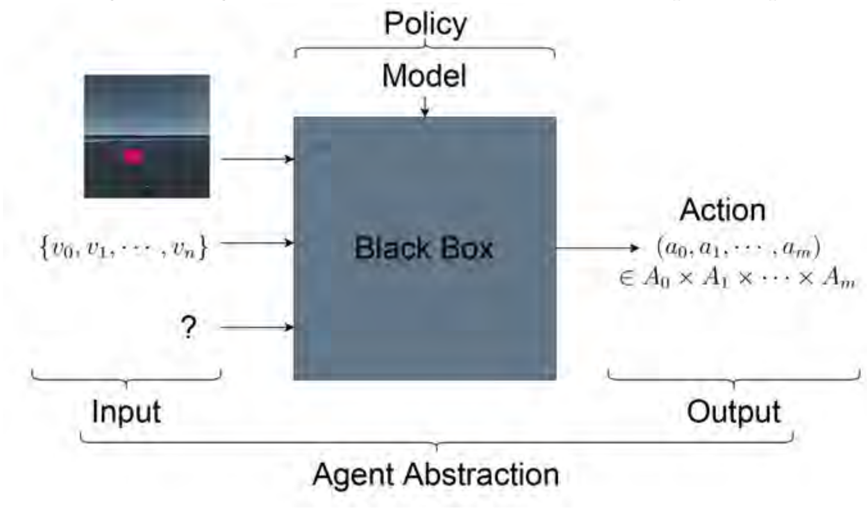
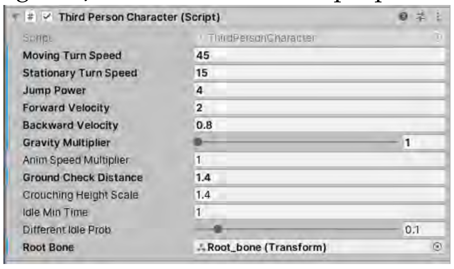

Policy
Model
Black Box
$\{v_0, v_1, \cdots, v_n\}$
?
Action
$(a_0, a_1, \cdots, a_m)$
$\in A_0 \times A_1 \times \cdots \times A_m$
Input
Output
Agent Abstraction

Figure 4: Character intrinsic properties.



Third Person Character (Script)

Script	ThirdPersonCharacter
Moving Turn Speed	45
Stationary Turn Speed	15
Jump Power	4
Forward Velocity	2
Backward Velocity	0.8
Gravity Multiplier	1
Anim Speed Multiplier	1
Ground Check Distance	1.4
Crouching Height Scale	1.4
Idle Min Time	1
Different Idle Prob	0.1
Root Bone	Root_bone (Transform)

While these can all be cast as essentially the same task, their difference comes of how they are modeled inside the virtual environment and which information is available to the agent to complete them. That said, they are practical examples of mobility tasks.

2.3 *Agent*

The agent is an entity abstraction which is itself a *behavior policy*. Nevertheless, when referred without a specific policy it can be imagined as an entity with sensors collecting observations and actuators interacting, where a policy can be plugged in linking an observation with a given action[3] as show in the figure 3.

The important points in a agent are what it observes to take actions, how both the observations and actions are encoded and how the associated rewards are distributed.

Character Controller

The Unity character controller is the lower level hierarchy of agent's actuators. It can be considered intrinsic to the agent and controls its velocity, turn speed, gravity and other effects of the character physical properties. The detailed list of the parameters used in the training are in the results section 3.1, while the figure 4 shows it inside Unity editor.

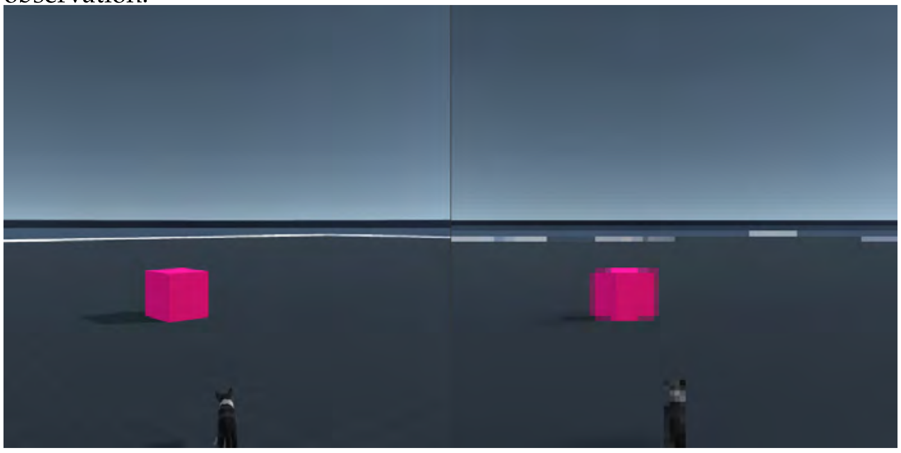
[3] In programming it would be equivalent to an interface, where the implementation itself would be the policy.

Agent Observation

An observation is any kind of sensing made from the agent itself or the environment state. Those need to be encoded by numbers that serves as input to the behavior policy. Here, two types of observations are employed:

- Vector observation: complete observable, hand-crafted features.
  - Normalized direction to target: $d_{\text{target}} = (x, y, z)$, $\|d_{\text{target}}\|_2 = 1$
  - Normalized distance to border: $d_{\text{border}} = (x, y)$, $\|d_{\text{border}}\|_\infty < 1 \rightarrow$ inside, $\|d_{\text{border}}\|_\infty \geq 1 \rightarrow$ outside
  - Linear velocity: $v_{\text{linear}} = (x, y, z)\text{m/s}$
  - Angular velocity: $v_{\text{angular}} = (x, y, z)\text{rad/s}$
  - Normalized agent forward direction: $d_{\text{forward}} = (x, y, z)$, $\|d_{\text{forward}}\|_2 = 1$
  - Normalized agent up direction: $d_{\text{up}} = (x, y, z)$, $\|d_{\text{up}}\|_2 = 1$
  - Agent local position (agent's position referent to its transformation matrix): $p_{\text{local}} = (x, y, z)$
- Visual observation: partially observable, down-sampled from the original rendered image, 3rd-person-like camera aligned with agent forward direction.
  - 2D image: matrix $I_{84\times84\times3}(r, g, b)$

Figure 5: (Left)Character third-person camera. (Right)Downsampled visual observation.



The first encoding takes a total of 20 numbers (floats) as observation. These were calculated from the agent/environment state and are much like as the observations taken by Puppo, the Corgi[8], which is a demo made by Unity.

There are some differences to fit our modeling. First, the Normalized distance to the border is added, because it is relevant in an unbounded environment to keep the agent inside the desired area. Last all the joint angles and torque information were removed. Puppo works on a low-level control of the joint angles and torque while this new agent works on a high-level control with animations. Those differences are related to how the agent senses itself, while how it senses the rest of the environment and its target remained the same.

Next, the visual encoding consists of 84×84×3=21168 numbers (floats), in a 2D RGB image. This image is the direct rendering of the agent's view (figure 5). This kind of observation is interesting for many real world applications. yet it is much more complex and less complete than the first encoding in 20 numbers because only partial information can be inferred from it. In spite of that, it can be more general in some aspects. For example if a variable number of collectibles is admitted in the environment, the first encoding would not be able to handle that variation, but using a 2D image the agent could deal with many objects (because the image is already a partial view of the environment).

Observation Constraint

One differentiation about those observations is with respect to its constraints. They can be:

- Self-contained - Depends only on the agent state and sensors.
- Model-dependent - Depends on the environment underlying model.

With those criteria, despite the 2D image drawbacks, it is an agent self-contained sensing while the vector observation needs access of the underlying environment model to be processed and fed to the agent (i.e. target position). This is directly related to the applicability and generalization of the agent. The disadvantage of a model-dependent agent is: it cannot be placed in a different environment where it does not have access to the underlying model.

Agent Action

The DogBot's actuator(s) is a character inside Unity. It is composed of various animations and a controller (with blend trees and alikes) which receives four parameters controlling the X-Axis velocity, the Y-Axis rotation and booleans jump/crouch.

For the actions encoding two schemes were used continuous and discrete:

- Continuous action space:
  - Forward and backward movement: $\in [-1, 1]$
  - Steering left and right: $\in [-1, 1]$
  - Jump: $j \in [-1, 1]$,$j > j_0$
  - Crouch: $c \in [-1, 1]$, $c > c_0$
- Discrete action space:
  - Forward and backward movement: {backward, none, walk, trot, run}
  - Steering: {left, none, right}
  - Jump: {true, false}
  - Crouch: {true, false}

Each bullet is a action branch, and can be choose simultaneously. For the case of Jumping/Crouching, despite being separated branches, priority is given to Jump action over the Crouch as it is not physically possible to do both at the same time. While the encoding for actions are arbitrary, they reflect the ranges of the Unity character controller such as max velocity and turn speed, which are configurable but from the agent's perspective are intrinsic to its actuator.

2.4 *Reward*

The entire universe of reinforcement learning is based on encouraging the best behavior through rewards (much like it is done when teaching a trick to a pet), in other words, rewarding good actions according. It can also be posed as a optimization problem of maximizing the total reward[7].

In this scenario two perspectives are brought up: developing a good reward signal is the key to being able to solve this optimization problem, yet most of times it is not as easy to qualify a given action and state pair individually, but only the final outcome of a sequence of actions and states. In theory, even for the cases where only the final outcome is rewarded, in the limit after many (infinite) experiences it would be possible to learn an optimal behavior policy. Sadly in the real world, limited resources are available be it time, number of experiences, you name it. Hence, the art in learning good policies lies in modeling good rewards and good algorithms[4] as much as possible.

The DogBot agent experiments with two types of reward:

- Per action reward:
  - Positive reward is assigned if the agents approximates of the target, negative reward is assigned if it distances itself from it. The formula used in this case is $r = 0.01(v_{\text{linear}} \cdot d_{\text{target}})$
- Sparse reward:
  - Positive reward $+1.0$ is given when the agent reach its destination, i.e., the collectible.

In all cases, leaving the training area leads to a negative reward of -1.0 ending the episode. Also a small negative reward -0.0005 is given at each time step t_i. It was chosen to be close to $-\frac{1}{\text{\# steps}}$ so the accumulated penalty would not saturate the total reward signal. This is widely used as a time penalty to stimulate the completion of a task in the shortest time.

It's also important to classify these rewards in another way: The first reward needs a broader knowledge of the environment to be calculated depending both of the agent and environment underlying state. In the other hand, the second could be completely assigned with the agent's sensing, but brings in the credit assignment problem.

This differentiation is interesting because it resembles the agent self-contained observation property and have implications in the learning performance. Yet, after the learning process it does not prevent the agent to be used in a new unknown environment.

3 TRAINING

The tools used for training were Unity and its machine learning framework ML-Agents[2] in its version v0.13.1. All experiments use the Proximal Policy Optimization (PPO)[5] algorithm. Each experiment take several hours to run, hence the ones presented were run only once. While the results are expected to be similar between runs, when analyzing the results some variability should be taken in account.

The ML-Agents framework exposes some parameters related to it's built-in models (network architecture), they will be specified inside the section 3.1. Other specific details of the network itself are omitted because they are the default of ML-Agents and can be found in their documentation. This choice was made because the focus is on the development of the agent, environment and reward signal instead of network architecture.

The baseline for comparison is a implementation that resembles the Unity Puppo, the Corgi where vector observation with information such as, speed, direction, direction to target and others are used as observation. The similarities stop there as the actions are high level decisions which are passed to the character animator/controller instead of low level torque control.

3.1 *Configuration*

The parameters used both for the character controller and and ML-Agents are presented in the tables 1 and 2 respectively.

[4] Here good performance means reduced time and sample complexity.

Table 1: Character controller parameters.

Name	Value	Description
Moving turn speed	45.0 deg/s	Turn speed when not stationary
Stationary turn speed	30.0 deg/s	Turn speed when stationary
Jump power	5.0 m/s	Vertical velocity applied when jumping
Forward Velocity	9.0 m/s	Maximum forward velocity
Backward Velocity	2.0 m/s	Maximum backward velocity
Gravity multiplier	1.0	Multiplier for gravity simulation
Anim speed multiplier	1.0	Multiplier for animation time scale

Table 2: ML-Agents training parameters.

Name	Value	Description
batch size (Continuous)	4096	Number of samples used for each optimization step for continuous action space.
batch size (Discrete)	256	Number of samples used for each optimization step for discrete action space.
buffer size	40960	Number of samples collected for each policy update.
hidden units	512	Number of neurons per hidden layer.
num layers	2	Number of hidden layers used for the model.
learning rate	3.0×10^{-4}	Initial learning rate for training.
max steps	2×10^{7} m/s	Number of total simulation steps (actions) taken for training.
num epochs	5	Number of times each collected observation is used for training.
time horizon	1000	Horizon for learning, it represents how far in time steps one action can influence a past reward.
gamma	0.995	Discount factor, it represents how much of a n-future reward (R_n) is assigned to a present action in the form $\gamma_n R_n$.
Curiosity strength	0.1	Strength of the curiosity intrinsic reward signal.
Curiosity gamma	0.99	Discount factor for the curiosity reward.
Visual encoding type	nature_cnn	Type of architecture used for the convolutional layers for the visual observation encoding.

Table 3: Results for the various models on the standardized test scene. In order the columns are: Experiment number(Exp), observation type(Obs. Type), action type(Act. Type), number of collectibles in the training environment(Train Env.), Reward Type(Reward Type), number of collectibles in the test environment (Test Env.), Score(Score) and Reset(Reset).

Exp	Obs. Type	Act. Type	Train Env.	Reward Type	Test Env.	Score	Reset
1	Vector	Discrete	1, box	Per action	1, box	1027	61
2	Vector	Continuous	1, box	Per action	1, box	2324	82
3	Vector	Discrete	1, box	Sparse	1, box	4	670
4	Vector	Continuous	1, box	Sparse	1, box	0	0
5	Visual	Discrete	1, box	Per action	1, box	1370	7
6	Visual	Continuous	1, box	Per action	1, box	2883	0
7	Visual	Continuous	24, boxes	Sparse	1, boxes	12	0
8	Visual	Continuous	24, boxes	Sparse	24, boxes	7119	3
9	Visual	Continuous	1, box	Per action	24, boxes	6163	0

Table 4: Results for the scaled down environment on the standardized test scene. In order the columns are: Experiment number(Exp), observation type(Obs. Type), action type(Act. Type), number of collectibles in the training environment(Train Env.), Reward Type(Reward Type), number of collectibles in the test environment (Test Env.), Score(Score) and Reset(Reset).

Exp	Obs. Type	Act. Type	Train Env.	Reward Type	Test Env.	Score	Reset
10	Visual	Continuous	1, box	Per action	1, box	4606	599
11	Visual	Continuous	24, boxes	Sparse	1, box	1502	241

4 RESULTS

The table 3 and 4 contains the result of various trained models on the test scene containing n collectibles which randomly re-spawn when collected. The evaluation metric is the Score, (the number of objects collected over all episodes) and Reset, (the number of times the agent was reseted due to leaving the training area). It ran for 200 episodes or 10^6 steps, (each episode is 5000 steps long).

4.1 *Training Convergence*

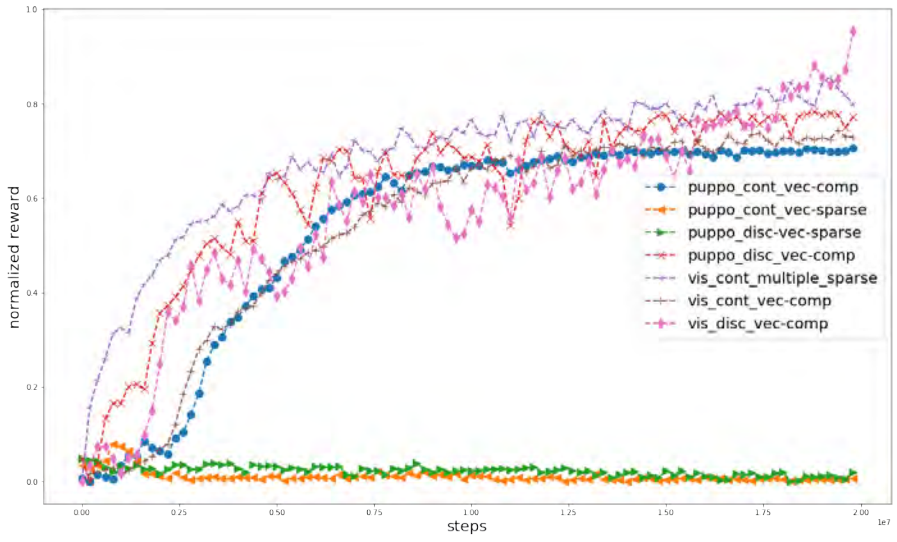
The figure 6 shows the evolution of the training from various configurations. The reward was normalized because each environment had a different reward system and can't be compared directly. Also this figure intends to show the convergence and evolution of the training procedure, while the results section presents specific result for each configuration in a standardized test scene.

4.2 *Discussion*

Action branches and time interval

One difficult when training with all action branches active at the same time was that it would not learn anything. Starting from a random policy, the agent would be stuck alternating between forward, backward, left and right. This problem with the Y-Axis movement is probably due to the interval between the actions being short and not allowing the agent to commit to a give action for enough time until a strong reward signal were obtained.

Figure 6: Reward evolution during training. The naming scheme is the following: *"observation type"_"action type"_"reward type"*, where *puppo* stands for vector observation (like the puppo example), *vis* stands for visual observation, *cont* stands for continuous actions, *disc* stands for discrete actions, *vec-comp* stands for vector per action reward (complete), *vec-sparse* stands for vector sparse reward (partial), *multiple* stands for a scene with multiple collectibles.



Nevertheless increasing the action interval would lead to unwanted delays in the reaction time of the agent. Four solutions worked well:

- Giving a slightly reward for moving forward.
- Adding a bias in the actions favoring the forward movement.
- Training the action's branch separated.
- Restricting the Y-Axis to forward movements only.

The last one was chosen, for being the simplest.

Observation

Visual observation achieved better results than vector observation when using the same reward system. It also could learn faster in the environment with many collectibles, where vector observation can't be applied. Despite it requiring more computational power, it shows that having a end-to-end learning may achieve better performance than having hand crafted features.

Action space

While the continuous actions space seems to be better than the discrete action space in all results, it can't be conclusive. One point is the batch size for continuous and discrete action space were different, and they were not extensively searched for the best result. The use of smaller batch size for discrete actions space is justified because the larger the batch size more samples are averaged in the training process and the in-between or average of discrete (categorical) actions may not make sense at all. Hence it is advised to use small batch sizes for these cases.

In the other hand, for continuous actions the average between samples makes sense (exists). Another way to think about it is how their sampling is done, the final value is sampled from a distribution with a given mean and variance, hence the need to approximate it from various simulation steps in the training.

Reward

Using a per-action reward strategy worked in all cases, while sparse reward only worked in the scene with multiple objects. It is clear that even with the drawback of the agent possible exploiting the reward instead of the true objective a per-action strategy is more reliable.

Nevertheless the sparse reward, given only when an object is collected, can't be exploited, but makes the learning process entirely dependent of the environment's difficult. For those cases a good approach is using a curriculum, starting with very easy tasks and increasing the difficult according with the agent performance. An example of such curriculum could be starting the environment with many objects and decreasing its amount according as the agent learns.

Multiplicity of collectibles

Having multiple objects in the scene completely changed the result of using sparse reward, from unable to learn anything to achieving the best result. Giving the agent an environment where it can often achieve a goal even with a random policy is the key point to use sparse reward. It is a good strategy to be used with curriculum learning, in this case, the difficult could be easily controlled by the amount of collectibles. While it was not tested here, one can believe that if the agent was trained with such curriculum it would perform extremely well. Especially in the test case where it was trained with many objects but tested on a single object and performed poor when the object was too far away.

Environment variation

Two variations where tested, changing the size of the environment and the number of collectibles. Those variations shown interesting behaviors.

First the agent trained with multiple collectibles which was not able to perform in the single collectible environment, had a good performance when the size of the environment was reduced. From visual inspection, what was happening was: the agent had a short sight. Again if the proposed curriculum was applied this behavior would probably be solved.

Then, the agent trained with a single object performed very well with multiple collectibles, but not as good as the one trained with many objects. Also from visual inspection, the agent would only turn to the right, which indeed works but is not the optimal behavior with multiple objects. In the case of a single object, which most of times was far away, it would not change much, but in the case of multiple objects it was a limiting factor.

These behaviors reinforces the idea of the need for variation in the training and probably the use of a curriculum of increasing difficult and variability. It also shows the importance of the details of the environment as a key factor for achieving the desired behavior and avoiding side-effects.

Learning generalization

Although the agent could somewhat generalize well to multiple and single objects it wasn't as good as the performance of the trained environment. When running on a smaller area it could not cope with the delimited marks and the agent left the are much more than when using the original size.

For achieving good generalization it must experience great variation of the environment and goal, which was not provided. Yet, this exposition need to be done without a steep difficult variation, or the agent may not be able to learn anything.

The key feature to develop is certainly the reward system. Making the agent to be able to have feedback even with a completely random behavior is a must, be it through a per action reward or a easier environment where rewards are not much sparse. Even so, it may get stuck if the action space is complex or if there is not enough time to commit to each individual action. This was exemplified by the lack of convergence when using the full action space despite having a per action reward. Nevertheless, with sparse reward and increased amount of objects in the scene leaded to convergence.

The type of observation used also shown some less intuitive behavior: while the vector observation had all the data needed to heuristically solve the problem, the visual observation had a better result. It may be due to encoding which is compact and human understandable, but may not be the best for learning, since it relies on common human knowledge and concepts. Letting the model extract the features from visual observation is computationally expensive for training, but it achieved better policies and is easily extensible to new environments.

Modeling and training agents to complete tasks, (in the way one would expect a human being to do so), is a complex problem. Good policies were learned, although side effects and not so much intuitive results were obtained. This explicits how an agent can exploit unnoticed details of its environment in unpredictable ways for good, bad and ugly things. Although the agent may solve the given task, expecting human like (in our case dog like) behavior may mislead the modeling and comprehension of the results.

Here, nor the neural network architecture/parameters neither the environment graphics complexity were evaluated, but the modeling of the environment, agent and reward. Certainly these aspects are important, but much has been done in that area already and could be easily "plugged in" or replaced in DogBot agent, for example a more complex visual system, i.e. a segmentation/detection neural network when using photo-realistic rendering.

Last, there is no formula to achieve good results. Part of it is the art of modeling the system and part is building insight from past failures and adapting, which itself is much the idea of (meta) reinforcement learning.

REFERENCES

[1] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[2] Arthur Juliani, Vincent-Pierre Berges, Esh Vckay, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2018.

[3] Seunghwan Lee, Moonseok Park, Kyoungmin Lee, and Jehee Lee. Scalable muscle-actuated human simulation and control. *ACM Transactions on Graphics (TOG)*, 38(4):1–13, 2019.

[4] 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.

[5] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[6] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai,

Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

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

[8] Unity3D. Puppo, the corgi: Cuteness overload with the unity ml-agents toolkit, 2018.

[9] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.