E^2 RL – Efficient Exploration in Reinforcement Learning

João Abreu, N. 67329, MEIC
Instituto Superior Técnico, Lisbon
joao.abreu@tecnico.ulisboa.pt

ABSTRACT
This thesis addresses the problem of efficient exploration in reinforcement learning (RL) with neural network approximations. However, and in spite of their recent successes, these methods require significant amounts of data.

Efficient exploration strategies — in which the agent actively seeks to visit promising or less-visited portions of the state-action space — have been actively investigated in classical RL domains, significantly improving the learning efficiency of such methods.

This thesis contributes novel active exploration strategies that combine and extend existing approaches for exploration with Deep RL architectures.

The impact of our proposed approaches is tested in several benchmark domains in the RL literature, showcasing the positive impact of active exploration in the learning performance of RL algorithms with neural network approximations.

Author Keywords
Deep Learning, Reinforcement Learning,
Exploration-exploitation trade-off, Intrinsic motivation, Noisy neural networks

INTRODUCTION

Motivation

Deep Learning is a machine learning technique that has recently had several high profile success stories by demonstrating progress in a number of classical computer vision and artificial intelligence problems.

It consists of training neural networks with several layers of hidden units. This is an old idea which recently became viable due to engineering advances and some tweaks to the training algorithms based on backpropagation, making them more appropriate for deep networks.

Some of the most notable results include learning to play classic Atari 2600 video games from screen captures [12] and attaining human-level skill in the board game Go [16].

These applications also made use of reinforcement learning. This is a different machine learning and AI discipline which studies the problem of how artificial agents can learn to interact optimally with an environment which provides them with a reward signal. More specifically, it studies the algorithmic tools for determining policies that maximize expected long-term cumulative reward. These policies are deterministic or stochastic maps from states of the environment to available actions of the agent.

In problems where the state-space is too large or continuous, it is impractical to use traditional approaches to approximate the Q-value functions mapping states and actions to expected reward as well as other related functions. Deep learning can then be used to learn non-linear approximations of these functions which can be used in practice.

In the process of learning these policies there is an implicit trade-off between exploration and exploitation. This involves balancing the immediate benefit of selecting the most rewarding option according to current incomplete knowledge with the potential benefit of discovering better options by acquiring new information through exploration of the state-action space.

Despite showing impressive progress, these approaches still have certain limitations, especially for problems in which rewards are sparse (as is the case for certain Atari video games [8]). This lack of robustness is in part due to the inefficient exploration and learning methodologies.

This thesis explores how these limitations could be addressed by combining reinforcement learning using neural network based approximations with different methods for exploration as well as analyze how they work and their effects in several environments.

Problem statement and hypothesis

This thesis addresses the following research question:

- How can the learning performance of reinforcement learning with neural network function approximation be improved through efficient exploration?

To address the research question above, a novel active exploration approach based on *pseudo-counts* for reinforcement learning using neural networks is implemented and tested. Our hypothesis is that visitation frequencies — captured through pseudo-counts — can be used to augment the reward function with an exploration bonus that fosters efficient exploration and leads to an improved learning performance of the agents.

The proposed approach is general, in that it does not rely on any specific domain knowledge, and can be implemented without a significant impact on the computational effort of the learning algorithms.

Contributions

The main contribution in this thesis is a comparative study of the effects of different strategies to improve the exploration in different environments of reinforcement learning agents with neural network approximations:

- Standard ε-greedy exploration as used in [12];
- Intrinsic motivation — specifically through modifying the reward function to incentivize the agent to visit portions of the state-space that have potentially high information gain. This is done using a pseudo count based exploration bonus [1];
- Using different sources of uncertainty to drive exploration by considering noise in the weights and biases of the neural network itself [3].

The latter two strategies are also combined and tested, since they are complementary, in the sense that the first only changes the reward function and the second the structure of the network and the action selection algorithm. As such, they can both be easily applied simultaneously.

Additionally, a simplified density model is proposed in the pseudo-count bonus case, which still manages to improve the obtained results in several environments.

RELATED WORK

The tabular methods of reinforcement learning presented previously become impractical in cases in which the state-space is large or continuous. The usage of neural network approximations of these functions can partially overcome this problem, since the neural network can exploit the structure of state-space by using an appropriate architecture and can learn to generalize with appropriate training.

Neural Fitted Q Iteration [14] describes one way of doing this training. It is an offline Q-Learning Temporal Difference (TD) model-free algorithm for training a Q-value function represented by a neural network. It improves data efficiency by attempting to constrain the effects of an update on unrelated portions of state-space induced by the global approximation of the neural network. To this end, it stores a batch of transitions and uses all of those during training. Every iteration of the algorithm consists of generating a dataset consisting of the inputs (s, a) and targets or outputs (2). Training is then performed with those values using the quadratic error between estimate and target as a loss function (the authors used the *Rprop* algorithm).

$$\text{target} = r(s,a,s') + \gamma \max_{a'} Q_k(s',a') \tag{1}$$

More recently, the application of deep learning to reinforcement learning problems started to be developed. Some of these recent results are presented here.

Prominent initial results demonstrated the capability of deep learning based agents to learn to play Atari 2600 games [12]. The architecture proposed, called a deep Q-network (DQN), received as input the screen captures from video games of the 4 previous frames, with the output being the Q-values for all possible actions. The architecture consisted of 3 convolutional layers followed by two fully connected layers, with the hidden layers connected by a rectifier nonlinearity. Two main advances were proposed for reducing training instability. During training, stochastic gradient descent is used to optimize the parameters of the network used to compute the Q-values. The target values provided to the loss function are progressively updated based on an expression that involves the estimated Q-values (target $= r + \gamma \max_{a'} Q(s',a';\theta^-)$). In order to reduce correlations which lead to slow or incorrect convergence, a second set of parameters that is only updated periodically is used to compute Q-values for the target (θ^-). Training is done off-line and an experience replay mechanism is used, which stores sets of transitions $e_t = (s_t, a_t, r_t, s_{t+1})$. These transitions are then sampled randomly during training. This had the effect of reducing correlations in the sequence of observations and reducing variations in the data distribution due to changing the policy derived from the Q-values.

Improved results in training for deep reinforcement learning were achieved with the approach presented in [11]. Instead of using an experience replay mechanism in order to stabilize training, several independent actor-learners running in different threads explore the environment simultaneously. Each of these can have different exploration strategies, and as such visit different portions of state-space. This is an on-policy algorithm, in which each of these actor-learners computes the gradient of the error function (using shared parameters of the network) associated with the transitions it experiences. These gradients in relation to the parameters ($d\theta$) are accumulated globally for all the agents ($d\theta \leftarrow d\theta + \frac{\partial (y - Q(s,a;\theta))^2}{\partial \theta}$) and are periodically used to update the parameters of the network asynchronously. The parameters used in computing the objective function (θ^-) are also only updated every N steps.

A3C seems to improve on the scores and speed-up the training on the games in which it was evaluated (including Atari 2600 games).

Exploration-exploitation trade-off

The previously referred trade-off between exploration and exploitation in reinforcement learning is particularly important in the kinds of complex tasks in which reward may be sparse and delayed. Several approaches to this problem have been proposed, which we summarize next.

Active learning

Active learning relates to the choice of strategy in data acquisition in order to be able to maximize performance of the agents. From the perspective of reinforcement learning it closely maps to the concept of exploration of the environment.

One application of active learning to the training of deep neural networks was presented in [17]. The paper describes the use of active learning to improve learning efficiency of a CNN that tries to correctly transcribe CAPTCHAs (Completely Automated Public Turing tests to tell Computers and Humans Apart). The output of the neural network provides a predictive distribution for each of the digits of the CAPTCHA. To make

training faster only CAPTCHAs that have been previously correctly predicted but for which uncertainty η (measured by the sum of the ratios of the second highest to the highest classification probabilities for each of the digits) is the largest are used for retraining.

One possible application of the ideas of active learning is during training and sampling from experience replay mechanisms. The authors in [15] propose training with experience replay where the transitions used are sampled from a distribution that prioritizes those with a larger corresponding TD error. TD error is used as a proxy for how unexpected that transition is. The distribution from which the transitions are sampled from is specificed by:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{2}$$

In this expression, p_i can either be of the form $p_i = |\delta_i| + \varepsilon$ (the sum of the absolute value of TD error (δ_i) in addition to a constant ε, that is a parameter) or $p_i = rank(i)^{-1}$ (in which $rank(i)$ is the rank of the ordered TD errors). α is a parameter that controls the level of determinism in the sampling.

Incorporating additional information from the environment

The neural networks used in deep reinforcement learning are used to approximate either value functions or policies directly. However, the choice of architecture can affect the success in training. It is possible to connect some of the initial layers of the network to additional outputs which represent information that reflects the structure of the problem. This makes those layers better at predicting those potentially informative features, which may help to learn better approximations of these functions.

In [9], the authors apply a Deep Recurrent Q-Network to learn how to play the FPS video game Doom. Since the world is only partially observed due to the limited field of view (making the task a Partially Observable Markov Decision Process (POMDP)), the network estimates $Q(o_t, h_{t-1}, a_t)$, in which h_{t-1} corresponds to the values of an hidden layer that is the output of the Convolutional Neural Network (CNN) at a previous time-step.A recurrent network is used to compute an updated value for this hidden layer $h_t = LSTM(h_{t-1}, o_t)$, which is then used to estimate the Q-values — $Q(h_t, a_t)$. The images from the screen of the video game are passed through two convolutional layers before being fed to a fully connected layer that connects to the LSTM. Additionally this convolutional layer connects to an output that encodes whether different entities (enemies, power-ups, etc.) are present in the frame. This information is obtained from the video game engine and is then provided as an additional output during training. The objective is to allow the parameters of the convolutional network to learn to predict the presence of those objects which, in theory, should improve performance.

Learning structure of state-action space

There has been some success in learning sequences of actions, instead of associations of states to single actions.

In [8], the authors combine temporal abstraction and intrinsic motivation in an approach they call Hierarchical Deep Reinforcement Learning. It combines two levels of control, the meta-controller maximizes extrinsic reward and learns to choose a goal which it provides to the controller. The controller on the other hand maximizes intrinsic reward based on changing relations between objects in the frame. The detection of objects in the frame is hand-tuned to the game and provided as additional inputs to the network in the form of masks with the same resolution as the other input images. It has greater success than DQN in one Atari game with sparse reward, "Montezuma's Revenge".

The authors in [2] propose considering learning sequences of actions that are executed deterministically in addition to atomic actions. This effectively transforms the Markov Decision Process (MDP) process the agent is trained on into an Semi-Markov Decision Process (SMDP) (since each "action" takes a variable amount of time). These macros are determined with an heuristic based on the frequency with which each sequence of actions is executed. For each of these macros there is a corresponding output from the DQN which estimates the associated value of taking the corresponding sequence of actions. These outputs need to be retrained when the macros change. By associating a value function to sequences of actions, the problem of sparse rewards is somewhat alleviated because the value function of a macro depends on the rewards obtained in a larger time period. It shows improved performance in several Atari games.

Intrinsic motivation

By taking inspiration in developmental psychology, intrinsic motivation or curiosity attempts to provide incentive for exploration based on the potential information gain from each choice.

In [1], a reward bonus is provided based on an approximation of visitation counts (pseudo-counts, $\hat{N}_n(s)$) that is obtained from a density model of the frames of the game (pixel-level version of Context Tree Switching, which is a probabilistic sequence technique).

Stochastic value function

Another idea has recently been proposed that does not fit in the previously identified categories of exploration strategies, which consists of directly applying the randomness that is involved in exploration to the learned Q-value functions instead of afterwards, during action selection (like in the case of ε-greedy). In [3], the authors propose modelling the Q-value function as a random variable that is constructed by using a neural network that replaces the weights and biases used in the affine transformations by normal random variables which are each parameterized by a mean and a standard deviation.

Selection criteria

The last two approaches based on a pseudo-count bonus and using noise at the level of the value function were selected and will be explained in more detail in the next chapter. This choice was made based on the fact they have distinct and interesting theoretical underpinnings and are relatively independent in practical terms and their implementations can easily be combined to produce a hybrid solution.

In the first case, a pseudo-count bonus relies on providing an additional bias towards towards exploration by making it more rewarding to the agent to select actions leading to portions of state space that have been less visited in the past.

Whereas the usage of noisy networks involves recasting the problem of exploration in an entirely different way because the noise parameters (the standard deviations of the weights) become variables that should be tuned by the optimization algorithm used.

Implementation-wise, this second approach requires changing the structure of network, sampling random values before action selection and training. The pseudo-count strategy only requires modifying the reward function and updating the density model online, as the agent acquires information.

The Deep Q-Network (DQN) algorithm with ε-greedy action selection was used as a baseline, because it can achieve respectable performance and proved simpler in terms of implementation than Asynchronous Advantage Actor Critic (A3C) due to the latter algorithm's usage of the asynchronous nature of multithreaded execution to overcome the problem of correlated values in online training.

EXPLORATION IN DEEP REINFORCEMENT LEARNING

The proposed solution tries to evaluate several different strategies identified in the literature, such that their relative effects on performance of the agent can be tested.

To recap, in order to apply reinforcement learning to continuous or high-dimensional state spaces, the value functions considered have to be approximated. In the work presented here, this approximation is obtained through the use of neural networks. Since the action spaces are discrete the networks will be used to model a Q-value function that receives the state representation at its input and has a different output for each possible action corresponding to the associated $Q(s,a)$ value.

Network architecture

In the cases considered, the states are continuous and of varying dimension. As such, a fully-connected feedforward neural network was used. Each input node receives a continuous value that corresponds to a different dimension of the state space.

Fully-connected Neural Network

A fully-connected network was used which had an input size equal to the dimensionality of the states of the environment, with the following layers:

- Hidden layer with 512 hidden units.
- Hidden layer with 256 hidden units.

The Rectified Linear Unit (ReLU) non-linearity was used between layers.

The output layer is a fully-connected linear layer of size equal to the number of available actions.

Initialization of the networks

The weights were initialized from independent truncated normal distributions with mean 0 and a standard deviation of 0.1 and the biases were initialized with a value of 0.1. This had the effect of producing values at the output close to zero, which was useful in terms of providing an optimistic initialization in tasks with non-positive rewards, which is further explained ahead.

DQN with Experience Replay

The main algorithm that was considered was DQN, that was implemented in [12] and had previously been applied to Atari 2600 games. Its main innovations are the use of experience replay and a separate target network which help stabilize training.

Experience replay consists in keeping a long-term memory storage of transitions, from which sampling occurs during the training periods which alternate with action selection. The choice in capacity of the memory represents a trade-off between the stabilizing effect of using old values in optimization and the capacity to learn quickly.

The separate target network is used to keep frozen copies of the weights of the main network, which are only periodically updated. This leads to decreased correlations between the value function that is being subjected to optimization and the target values, which decreases the tendency of the process to diverge. This divergence would happen due to positive feedback if the target function and the current value function were being updated simultaneously.

A modification of DQN is used called Double DQN [4], that selects the action used to compute the target from the value network that is being optimized in the non-terminal case (Equation 3).

$$y_j = r_j + \gamma \hat{Q}(\phi_{j+1}, \underset{a'}{\operatorname{argmax}}\, Q(\phi_{j+1}, a'; \theta); \theta^-) \qquad (3)$$

The authors argue that this modification can partially correct the previously found bias in Q-Learning that leads to overestimation of the value function as a result of the use of the max operator in its update equation [4] which tends to select overestimated values for value iteration.

Optimization algorithm

Instead of the algorithm RMSProp, used in the original DQN paper, the Adam algorithm [7] was used instead. This algorithm can be seen as having simultaneously the capability of working with sparse gradients and being well adapted to non-stationary and online settings (similar to the case of RMSProp). It is characterized by computing what amounts to adaptive learning rates for each parameter individually, based on exponential moving averages of the gradient (m_t) and squared gradient (v_t).

The ratio $\hat{m}_t / sqrt(\hat{v}_t)$ generally tends towards zero as $f(\theta_t)$ gets closer to the minimum. This can be seen seen as a form of automatic annealing. This means that the usually necessary but ad-hoc step of using a schedule to reduce the step-size of the optimization algorithm could be avoided.

These properties mean that Adam generally has good performance, without requiring extensive tuning of its parameters.

Adam is used in place of the RMSProp gradient descent step in DQN, by applying one iteration of the presented algorithm's main loop for each gradient descent step with transitions from the experience replay memory.

Exploration approaches

This section describes the exploration approaches investigated in the thesis.

ε-greedy exploration

It is one of the simplest, more traditional approaches to action selection in reinforcement learning. A certain probability ε is divided equally among the remaining actions while the remainder of the probability is assigned to greedy action that maximizes the value function (Equation 4) [18].

$$P(a|s) = \begin{cases} 1-\varepsilon+\frac{\varepsilon}{|\mathscr{A}(s)|} & \text{if } a = \text{argmax}_{a'} Q(s,a') \\ \frac{\varepsilon}{|\mathscr{A}(s)|} & \text{otherwise} \end{cases} \quad (4)$$

Usually a small value of ε is selected to ensure that there is progression while simultaneously exploring in non greedy directions. It is also common to have a certain decaying schedule that initially favors exploration and becomes progressively more greedy.

Pseudo-count bonus

This generalization of count-based exploration, described in [1], consists in providing incentive to exploration by using a reward bonus. In a traditional tabular setting there are ways that this can be achieved by keeping track of the visitation counts to each state. In the high-dimensional case, this becomes impractical.

These counts of state visitations are generalized in the form of pseudo-counts ($\hat{N}(x)$) which are calculated from a density model of the states ($\rho(x)$, Equation 5) and the pseudo-count total ($\hat{n}$).

$$\hat{N}(x) = \rho(x)\hat{n}(x) \quad (5)$$

This density model assigns a probability to observing a given state and is trained on-line (Equation 6) on the series of states observed by the agent ($x_{1:n}$).

$$\rho_n(x) := \rho(x; x_{1:n}) \quad (6)$$

This pseudo-count total can be derived from the *recoding probability* of the state $\rho'_n(x)$, which corresponds to the probability that the density model assigns to observing the state x after observing it immediately beforehand (Equation 7).

$$\rho'_n(x) := \rho(x; x_{1:n}x) \quad (7)$$

Since it should be the case that observing a state should increase the pseudo-count in one unit, it is postulated that:

$$\rho_n(x) = \frac{\hat{N}_n(x)}{\hat{n}}, \quad \rho'_n(x) = \frac{\hat{N}_n(x)+1}{\hat{n}+1} \quad (8)$$

Through algebraic manipulation, this leads to the following definition of the pseudo-count:

$$\hat{N}_n(x) = \frac{\rho_n(x)(1-\rho'_n(x))}{\rho'_n(x)-\rho_n(x)} \quad (9)$$

By definition, the prediction gain (PG) of ρ is:

$$PG_n(x) = \log \rho'_n(x) - \log \rho_n(x) \quad (10)$$

Using this definition of the prediction gain, an approximation of the pseudo-count can be achieved from:

$$\hat{N}_n(x) \approx \frac{1}{(\exp^{PG_n(x)}-1)} \quad (11)$$

This pseudo-count can then be used to provide an exploration bonus, in form of an additional term to add to the external reward from the environment.

$$r^+(x) := \frac{1}{\sqrt{\hat{N}_n(x)}} \quad (12)$$

$$r_t^{total} := r_t^{env} + r_t^+ \quad (13)$$

In order for the density model to produce the desired effect on learning to solve the Reinforcement Learning (RL) problem, it should be learning positive. That is, observing a state should increase its assigned probability: $\rho'_n(x) \geq \rho_n(x)$. This also ensures that the prediction gain is always positive.

The implementation of this model presupposes a density model that can be updated online at each iteration of the DQN algorithm, from the sequence of observed transitions by the agent. As such, to compute the reward bonus, the density model is queried with the current observed state (obtaining the probability $\rho(x)$), the model is updated with that observation and queried again with the same state (which results in the recoding probability $\rho'(x)$). With these two values the prediction gain (Equation 10), pseudo-count (Equation 11) and, finally, reward bonus (Equation 12) can be obtained.

Density models

A simple density model was considered for the classical control problems in continuous space. The solution is an approximation that nevertheless should scale up well in terms of running time and had good empirical results (as will be shown in the next section).

The continuous space was discretized and the probability distribution was factorized and approximated in the following way:

$$\boldsymbol{s} = [s_1, \ldots, s_N] \tag{14}$$

$$P(\boldsymbol{s}) = P_1(s_1)\prod_{i=2}^{N} P_i(s_i|s_1,\ldots,s_{i-1}) \simeq P_1(s_1)\prod_{i=2}^{N} P_i(s_i|s_{i-1}) \tag{15}$$

Each of the factored terms is a simple counter that is kept updated on-line as soon as every state is observed. This set of observed states is designated D. In order, to avoid a null pseudo-count, the initial counts in the model were initialized to 1, resulting in the following approximation:

$$P_i(s_i = x|s_{i-1} = y) \simeq \frac{\#(\{\boldsymbol{s}^t \in D|s_i^t = x \wedge s_{i-1}^t = y\}) + 1}{\#(\{\boldsymbol{s}^t \in D|s_{i-1}^t = y\}) + 1} \tag{16}$$

An adaptation of the CTS (Context Tree Switching) model [1] proposed in the original paper was also used that modelled each component of the state vector independently when conditioned on the previous components.

Noisy networks

This strategy, proposed in [3], is to introduce some amount of randomness to the behavior of the agents to benefit exploration by considering noise in the weights of the neural network instead of using randomness after computing the neural network's values (like in the case of ε-greedy).

Shifting the randomness to the weights instead of the action space makes the value function a random variable and allows for some theoretical improvements. The action selection during training can be kept consistent this definition of the distribution of the value function, allowing for correlated actions, as opposed to the case of ε-greedy action selection.

The standard deviation associated with the noise is also a parameter that can be optimized by the same procedure that affects the means of each weight.

To this end, each of the network paramers (weights or biases) in the fully-connected portion of the neural network is turned into a random variable (Equation 17), which is characterized by a different set of parameters. In the following expressions, the $\odot$ symbol corresponds to the element-wise multiplication of the involved vectors or matrices and ε represents a matrix (or vector) of random variables of the same size as the deterministic networks weights (or biases) from independent standard normal distributions. This makes μ and σ the corresponding means and standard deviations, respectively.

$$\theta \stackrel{def}{=} \mu + \Sigma \odot \varepsilon,\ \varepsilon \sim \mathcal{N}(0,1) \tag{17}$$

The loss during the training then becomes an expectation over the random variables θ (Equation 18).

$$\zeta \stackrel{def}{=} (\mu, \Sigma),\ \bar{L}(\zeta) \stackrel{def}{=} \mathbb{E}[L(\theta)] \tag{18}$$

As such, the formula used to calculate the values in a fully-connected layer, before the non-linearity becomes Equation 19.

$$y \stackrel{def}{=} (\mu^w + \sigma^w \odot \varepsilon^w)x + \mu^b + \sigma^b \odot \varepsilon^b \tag{19}$$

The gradients of the loss function can also be calculated by using a one-sample Monte Carlo approximation of the expectation of the gradients (Equation 20).

$$\nabla\bar{L}(\zeta) = \nabla\mathbb{E}[L(\theta)] = \mathbb{E}[\nabla_{\mu,\Sigma}L(\mu + \Sigma \odot \varepsilon)] \approx \nabla_{\mu,\Sigma}L(\mu + \Sigma \odot \varepsilon) \tag{20}$$

Or, more specifically in the case of the loss used in DQN:

$$\bar{L}(\zeta) = \mathbb{E}[\mathbb{E}_{(x,a,r,y)\sim D}[r + \gamma \max_{b \in A} Q(y, b, \varepsilon'; \zeta^-) - Q(x, a, \varepsilon; \zeta)]^2] \tag{21}$$

The ε-greedy action selection is replaced by sampling values from the parameters distributions of the action selection network, followed by greedy action selection. Another sampling step is applied to both action selection and target networks, before the training step during the experience replay mechanism.

TEST ENVIRONMENTS

Several environments provided by the OpenAI Gym package were used to test out the previously presented approaches to exploration in reinforcement learning.

Several simpler classical control problems in the reinforcement learning literature were considered for testing.

Cartpole

The objective is to learn to keep the pole balanced by applying either a left or right lateral force to the base at each time instant.

This was the simplest case considered, with only the existence of two failure conditions poses a difficulty to learning a policy.

The optimal controller for the pole is able to keep it upright and centered in the play field indefinitely. According to this loss function, that means that agent always receives a reward of $+1$, which means that for states and actions consistent with staying in the non-terminal condition:

$$Q^{\pi^*}(s,a) = \sum_{t=0}^{\infty} \gamma^t = 1/(1-\gamma) \tag{22}$$

Mountain Car

The goal is to drive a car to the finish line on the right by choosing to do nothing or to apply a lateral force to the left or to the right.

The two dimensional state space makes it easy to visualize the value function in this case.

For the agent to receive a signal from the reward, such that it is able to learn, it needs to reach the terminal condition.

The fact that it is necessary to balance back and forth along the slope of the mountain, which requires consistent actions (accelerations), means that epsilon-greedy exploration with random actions should have poor performance in this task.

Acrobot

The aim is to elevate the extremity of a two link mechanism above a certain height.

The actions available are to do nothing or to apply a clockwise or counter-clockwise torque to the second joint.

The reward function is similar to the *Mountaincar* environment.

RESULTS AND DISCUSSION

Experiment description

The results obtained are presented in the form of plots that were constructed by averaging the results of 20 simulations, per implemented exploration strategy, as specified in the previous chapter.

Each of these simulations, consisted of 1000 sequential episodes, in which the agent was trained in the environment and which, depending on the environment, ended when a final condition was reached (Acrobot and Mountaincar) or a necessary condition was not fulfilled (Cartpole). There was also a time limit enforced in all the environments if the final condition was not reached in the alloted timesteps.

Due to the high level of variance still present, each of these plots was also subjected to a moving average with a kernel of size corresponding to $1/100^{th}$ of the total number of points.

Parameter selection

The hyperparameters used for the Adam optimization algorithm are the default recommended values, which seemed to produce reasonable results as expected from the fact that the algorithm was designed to be used in non-stationary optimization problems, like those involved in reinforcement learning. Adam was also designed to be relatively insensitive to parameter tuning, which is one of its main advantages. The remaining parameters were selected empirically through tests and while most were somewhat robust to variations, the ε-value and the discount factor had to be at least that low and high, respectively, for the baseline DQN algorithm to be able to solve the Mountaincar task. This is consistent with the fact that this task requires a somewhat consistent sequence of actions in order for the car to acquire enough speed to climb the slope and reach the goal. A sequence of actions like this is improbable with an high value of ε, which means that there is a higher percentage of completely random actions.

This task also required the optimistic initialization provided by setting the weights of the network so that its output was initially close to zero. This was optimistic as a result of the rewards being negative in the Mountain Car task at all times except in the goal condition (this was also the case for the Acrobot environment). This optimistic initialization is a well known heuristic in reinforcement learning [18]. This incentive for exploration happens since the Q-value function initially has an higher value for all states and actions than what it reaches upon convergence. As an agent begins to learn the value function, explored state-action pairs get assigned a lower value than those that have not been visited yet, which encourages new states to be visited.

Results

Results are presented concerning the performance of the reinforcement learning agents and the evolution of the reward bonus based on the proposed density model.

Reinforcement Learning performance

Two measures of performance of the reinforcement learning agents will be presented for each the test environments.

The basic evaluation of performance in the tasks can be observed from a graph of the sum of the extrinsic rewards (without the reward bonus) obtained in each episode as training occurs. This is the natural choice considering the fact that maximizing cumulative reward is the objective of reinforcement learning agents.

As a simple sanity check, to ensure that the agent is learning, it is also possible to observe the evolution of the Q-values of the actions that were picked by the agent in each non-random choice, that is, these greedy choices correspond to maximum Q-value for the observed state at that timestep.

Cartpole

In terms of the performance in the Cartpole task (Figure 1), it is difficult to select a clear winner, but the network with noise seems to obtain an early advantage and also have made more progress at the end of the episode limit. Combining noisy networks with a reward bonus from a simple density function improves results while the one based on the CTS model seems to make the results worse.

As explained in the previous chapter, the obtained Q-values in the Cartpole environment should tend towards the value of $\frac{1}{1-\gamma} = 100$ (with $\gamma = 0.99$), if the timespan of the simulation were unbounded.

It can be observed that for all algorithms there is a continuous increasing tendency of the Q-values, which is consistent with learning due to an increased number of steps upright. However, the predicted value has not been reached in the simulation time-span.

Mountain Car

In the case of the Mountain Car environment (Figure 2), the networks with pseudo-count exploration seem to improve on the task much earlier and maintain the best performance throughout. The noisy network doesn't seem to be able to learn in this case, and the combination with a reward bonus can only match the ε-greedy baseline, with the bonus based on the simple model being slightly better. The proposed less accurate density model by itself seems to improve faster but perform slightly worse than the CTS model by the final episodes.

In this task, the fact that the reward only changes when the goal state is reached, combined with the need to build up balance by executing an appropriate sequence of actions, makes it

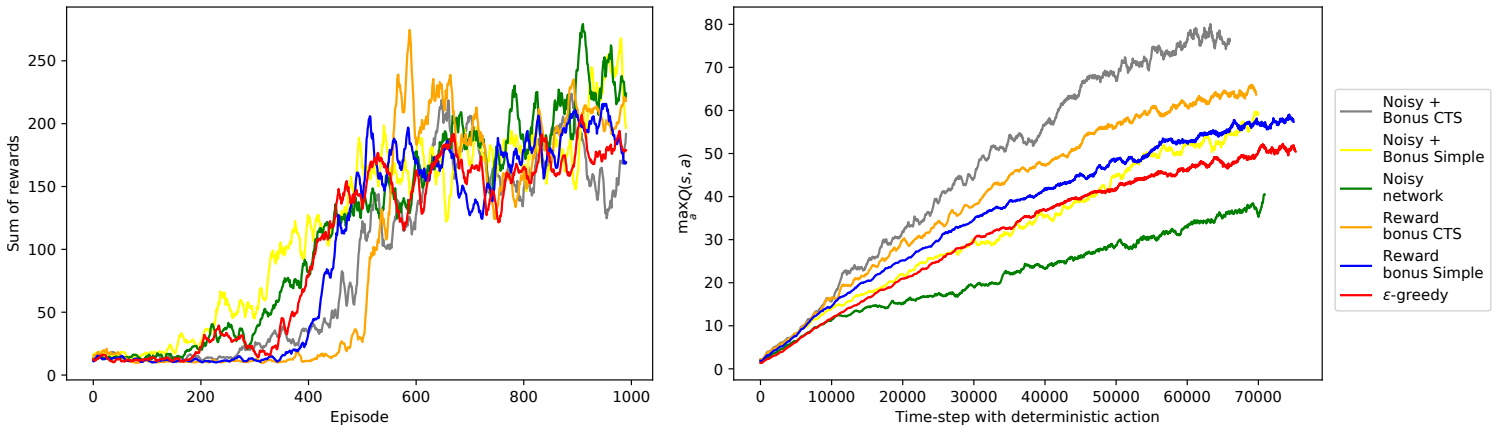

Figure 1: Reinforcement learning results for the Cartpole environment

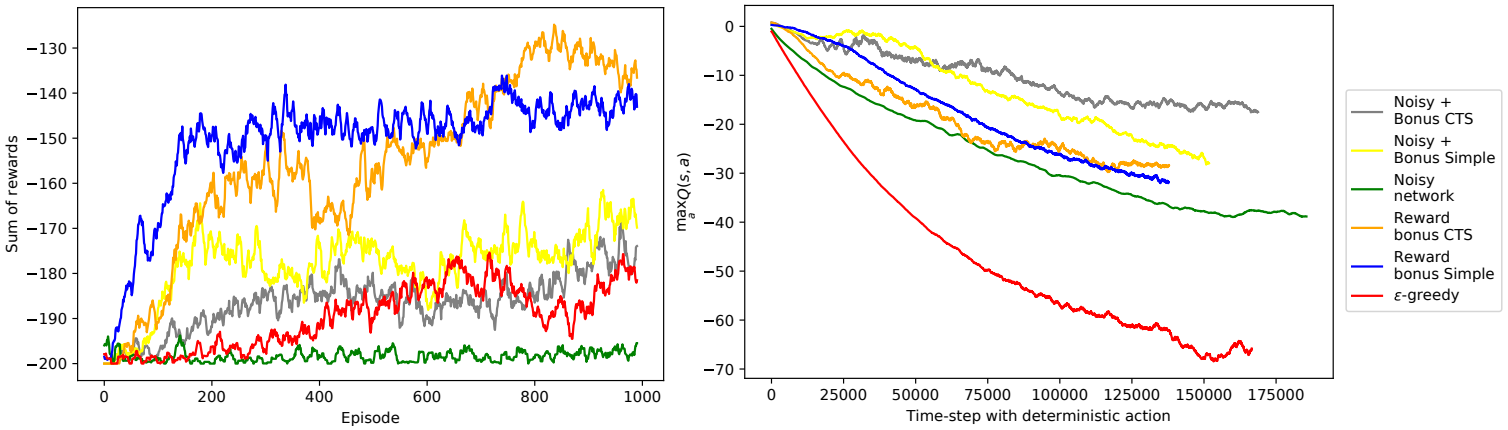

Figure 2: Reinforcement learning results for the Mountain Car environment

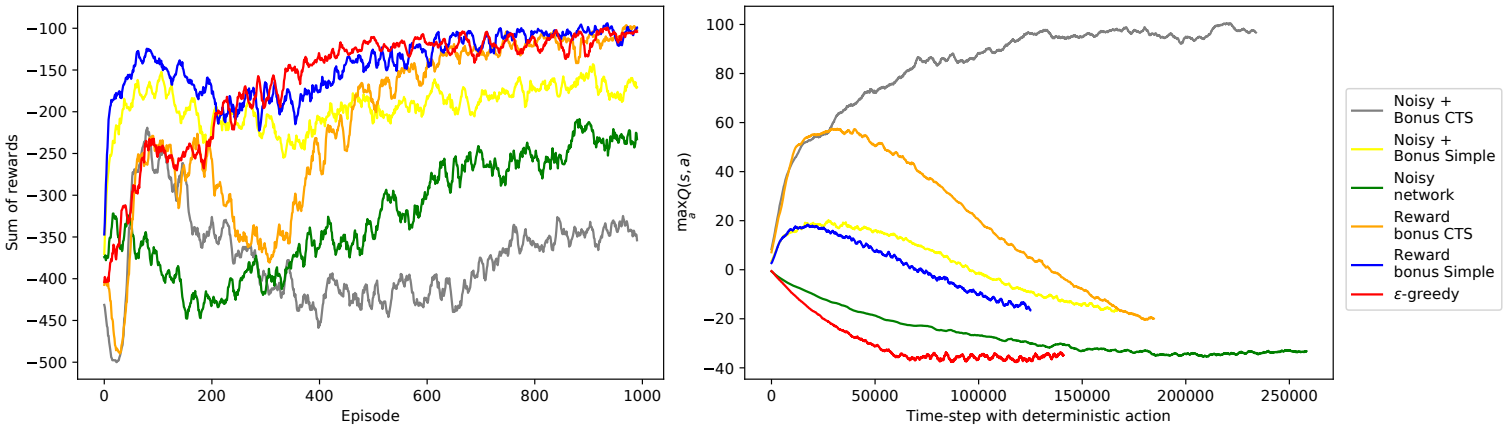

Figure 3: Reinforcement learning results for the Acrobot environment

more difficult for standard ε-greedy action selection, where spurious actions at the wrong time can perturb the momentum of the car excessively. The additional reward signal from the pseudo-count approach seemed to counteract the difficulty in learning to reach the goal, by encouraging the agent to explore alternatives after initially being unable to do it from the extrinsic reward exclusively. On the other, the randomness imposed by the noisy network approach proved too excessive for this problem.

In terms of the Q-values, the expectation in the baseline case is that there would be an initial decay due to the negative rewards and near zero initialization of the networks. The Q-values should then stabilize or increase to a still negative value as the goal condition is reached with a zero reward and the negative reward stops being consecutively acquired at each timestep. The particular final value will depend on how quickly the goal condition can be reached (Equation 23).

$$\text{Sum of rewards} = \sum_{t=0}^{N_t-1} -1 \cdot \gamma^t + 0 = -\frac{1-\gamma^{N_t}}{1-\gamma} < -1, 0 < \gamma < 1 \tag{23}$$

In terms of Q-values, the networks with reward bonus tend to have larger magnitude, as expected. While the noisy network with CTS bonus has the highest Q-values, the trend for progressive decay still happens, which is consistent with the agent still managing to solve the task. However, the noisy network also shows a trend similar to the expected, despite never actually learning the task.

Acrobot

In the case of the Acrobot task (Figure 3), pseudo-count exploration achieves an early lead in terms of performance, but the baseline ε-greedy later reaches approximately the same kinds of results. The bonus based on the simple model improves results particularly quickly. The noisy networks perform worse, with the bonus from the simple density function helping reduce this discrepancy and the CTS bonus increasing it, making this last case the worst performing strategy.

For this environment, it is observed that for the noisy network and ε-greedy exploration, there is initially a decrease in Q-values, followed by a slight increase, consistent with learning (reaching the state in which reward is zero). However, in the case of pseudo-count based exploration, there is a very different shape, due to the effect of the positive reward bonus, which counterbalances the negative extrinsic reward. As the density stabilizes and time progresses, the bonus decays and a behavior similar to the other techniques is observed.

In the case when the noisy network is combined with a CTS-based reward bonus the Q-values never seem to reach the expected negative values, this is consistent with a lack of learning as a result of possible excessive bonus reward due to a density that did not stabilize due to the excessive added noise to the network. The failure of only the CTS model could also be attributed to the higher number of dimensions of the state space in this environment, and the fact that the density model

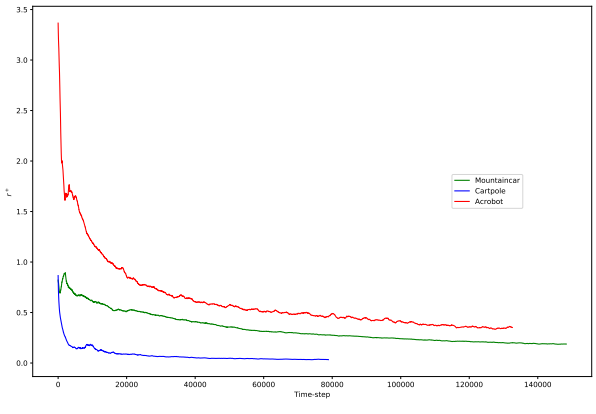

Figure 4: Pseudo-reward evolution in the classical control environments.

does not use the approximation of modelling each component conditioned on only the previous one, resulting in slower convergence of the density model.

Evolution of reward bonus based on the simple density model

For the proposed simple density model, it is also interesting to observe how the resulting reward bonus evolves as a consequence of the predicted convergence of the density model. When the density model converges, the values of the current probability and recoding probability assigned by it to the queried state should be approximately the same, leading to a small prediction gain and therefore a high pseudo-count and small bonus reward value.

It can be observed in all three classical control environments (Figure 4) that the reward bonus for the simple model is initially high but gradually decreases and stabilizes, as would be expected.

The higher magnitude of the initial reward bonus in the Acrobot environment (6 dimensional), can be explained by the higher number of dimensions of its state space, meaning that the density model experiences larger variations initially, as a consequence of the higher number of probability terms according to the models' definition. This higher magnitude also explained the greater effect on the learned Q-values shown for the Acrobot environment, which became initially positive. This trend is also kept for the Cartpole (4 dimensions) and Mountain Car (2 dimensions) environments, although the reward bonus quickly increases in the latter case and takes longer to stabilize, which is consistent with the problem being harder in terms of its exploration requirements.

In general, the reward bonus exploration seems to produce better results in the tests performed, with a general improvement in most cases (and comparable performance in the Cartpole environment). The simple density model proposed had better results in general than the more exact CTS-based model. This is interesting since it suggests that simple heuristics can occasionally be better and that the fidelity of the density model

is not the most important factor when applying pseudo-count based exploration.

The noisy network approach achieved worse results in Acrobot and was unable to learn in the Mountain Car environment. These failings were partially recovered by combining it with the reward bonus (except in the case of the Acrobot task with the CTS bonus, which performed worse). The additional variance due to the noisy network approach proved excessive and compromised performance in these cases. In order to be able to recover from this limitation it would be possible to apply an annealing process in order to decay the magnitude of the injected noise in the network.

CONCLUSION

A comparative study of different approaches for exploration in reinforcement learning algorithms based on neural network function approximations was performed when applied to several benchmark problems from the RL literature. These approaches consisted in providing an intrinsic motivation signal to the agent in the form of a reward bonus that encourages exploration of less visited regions of state-space and applying randomness to weights of the neural network instead of directly to the action selection. These approaches were also combined in a hybrid strategy and a simplified density model was proposed for calculating pseudo-count based reward bonus.

The pseudo-count reward bonus proved more successful in the tests performed than the baseline and the noisy network based approaches, which generally performed worse. The combination of these techniques produced intermediate results between those corresponding to the individually applied strategies. The simplified model also performed better than the more complete CTS-based model.

The studied strategies try to deal with reward sparsity by encouraging exploration in different ways.

The intrinsic motivation approach changes the reward function in order to increase exploration directed at under visited portions of state space. These changes tend to disappear as the density model converges, leading to a progressive decay of the prediction gain and therefore of the reward bonus.

The noisy network approach recasts the problem of exploration by making the non-deterministic behavior of the agent a part of problem in the form of learnable parameters of the network. However the introduced noise can be too strong in certain cases where a more stable policy needs to enforced so that learning can occur (like in the case of the Mountain Car environment).

Future directions and improvements

The approaches discussed could be implemented in more complex domains like in several Atari 2600 games (such as Pong), in which they should have more significant effects due to the increased difficulty of the exploration problems.

As an extension of the reward bonus approach proposed here, the density model could be extended to an approximation of a transition model if the present and next states were concatenated and used as inputs to the same density models discussed previously. This would be closer to the proposal in [10].

Exploration-exploitation in the types of problems considered here is an active topic of research and several more ideas could be tried, including some more recent proposals.

The usage of general unsupervised auxiliary tasks that try to induce interesting behavior in terms of control of the environment and predicting rewards could be an interesting complementary strategy [6].

The authors of [13], propose an alternative for an intrinsic motivation based reward bonus that, instead of being directly based on a numerical measure extracted from the state-space (images), uses an auxiliary network to construct a feature space that is trained to predict the actions of the agent from the state representations. The objective is to use this feature space to produce a measure of surprise that is based only on properties of the environment that the agent can control through its actions.

The usage of prioritized experience replay, where the recorded transitions would be sampled based on the magnitude of difference to the predictions of the neural network could also be interesting [15].

Another proposal is to penalize the reward function based on the a distance measure of the current policy relative to a limited set of past policies, which should encourage a higher degree of exploration [5].

ACRONYMS

A3C Asynchronous Advantage Actor Critic

CNN Convolutional Neural Network

MDP Markov Decision Process

POMDP Partially Observable Markov Decision Process

SMDP Semi-Markov Decision Process

ReLU Rectified Linear Unit

TD Temporal Difference

DQN Deep Q-Network

CAPTCHA Completely Automated Public Turing test to tell Computers and Humans Apart

RL Reinforcement Learning

REFERENCES

1. Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. 2016. Unifying Count-Based Exploration and Intrinsic Motivation. In *Advances in Neural Information Processing Systems 29*. 1471–1479.

2. Ishan P Durugkar, Clemens Rosenbaum, Stefan Dernbach, and Sridhar Mahadevan. 2016. Deep Reinforcement Learning With Macro-Actions. *arXiv preprint arXiv:1606.04615* (2016).

3. Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Matteo Hessel, Ian Osband, Alex Graves, Volodymyr Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. 2018. Noisy Networks For Exploration. In *Proceedings of the International Conference on Learning Representations*.
4. Hado van Hasselt, Arthur Guez, and David Silver. 2016. Deep Reinforcement Learning with Double Q-Learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. 2094–2100.
5. Zhang-Wei Hong, Tzu-Yun Shann, Shih-Yang Su, Yi-Hsiang Chang, and Chun-Yi Lee. 2018. Diversity-Driven Exploration Strategy for Deep Reinforcement Learning. *arXiv preprint arXiv:1802.04564* (2018).
6. Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. 2017. Reinforcement learning with unsupervised auxiliary tasks. *Proceedings of the International Conference on Learning Representations* (2017).
7. D. P. Kingma and J. Ba. 2015. Adam: A Method for Stochastic Optimization. In *Proceedings of the International Conference on Learning Representations*.
8. Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. 2016. Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation. In *Advances in Neural Information Processing Systems 29*. 3675–3683.
9. Guillaume Lample and Devendra Singh Chaplot. 2016. Playing FPS games with deep reinforcement learning. *arXiv preprint arXiv:1609.05521* (2016).
10. Manuel Lopes, Tobias Lang, Marc Toussaint, and Pierre-Yves Oudeyer. 2012. Exploration in model-based reinforcement learning by empirically estimating learning progress. In *Advances in Neural Information Processing Systems*. 206–214.
11. Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*.
12. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei a Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.
13. Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. 2017. Curiosity-driven Exploration by Self-supervised Prediction. In *Proceedings of the 34th International Conference on Machine Learning*, Vol. 70. 2778–2787.
14. Martin Riedmiller. 2005. Neural fitted Q iteration - First experiences with a data efficient neural Reinforcement Learning method. *Lecture Notes in Artificial Intelligence* 3720 (2005), 317–328.
15. Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2016. Prioritized Experience Replay. In *Proceedings of the 5th International Conference on Learning Representations*.
16. David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.
17. Fabian Stark, Rudolph Triebel, and Daniel Cremers. 2015. CAPTCHA Recognition with Active Deep Learning. In *Proceedings of the 37th German Conference on Pattern Recognition*.
18. R Sutton and A Barto. 1998. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, Massachusetts. 51–85,133–151 pages.