Uncertainty based exploration in Distribution Reinforcement Learning

A Project Report

submitted by

SUJITH VENNA

in partial fulfilment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY

under the guidance of
Dr. Balaraman Ravindran

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, MADRAS.

May 2020

THESIS CERTIFICATE

This is to certify that the thesis entitled **Uncertainty based exploration in Distribution Reinforcement Learning**, submitted by **Sujith Venna**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology**, is a bona fide record of the research work carried out by her under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Balaraman Ravindran
Research Guide
Dept. of Computer Science and Engineering
IIT-Madras, 600 036

Place: Chennai

Date: June 21, 2020

ACKNOWLEDGEMENTS

I Would like to thank my guide Dr. Balaraman Ravindran for guiding me throughout the project.

I Would also like to thank my family and friend for their support in these times.

Sujith

ABSTRACT

KEYWORDS: Reinforcement Learning, Distributional Reinforcement learning , Exploration

On of the fundamental problems in reinforcement learning is handling exploration exploitation tradeoff. Distributional reinforcement learning estimates and maintains a distribution over returns and has shown state of the art performance in may environments. We try to make use of the extra information available in distributional RL for efficient exploration. We leverage on the uncertainty of the unexplored states to direct our agent for efficient exploration. we experimented with environments from openAI gym and exploratory chain task which is considered as an difficult exploratory problem

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

Introduction

Humans or animals in general sense their surroundings and make decisions based on the stimuli received. Reinforcement learning is one of the topics in machine learning inspired by behavioral psychology. Making decision Accumulating information by exploring the environment, exercising this newly available information to make decisions in order to achieve maximum reward is the main objective of reinforcement learning. Trial and error search and delayed reward are the two most important characteristics of reinforcement learning.[14]

One of the major breakthrough in this field was achieved when DeepMind created a reinforcement learning algorithm, AlphaGo[ ] , which defeated the world champion in the game of Go. Considering the scale of the game it was thought to require intuition, strategy to master the game. With only limited information apart from the rules of the game achieving this landmark shows the power of reinforcement learning. Not only games, from finance to robotics to self-driving cars we can see a diverse application of reinforcement learning.

1.1 Leads to the Problem

The Objective of reinforcement learning is to maximize the reward. This can be achieved by taking actions which give high rewards. These actions are unknown and are needed to be found out or explored. There can always be some actions which give higher rewards than other actions which can be found out by exhaustive exploration. Large amounts of exploration leads to missing out on the rewards. There is a clear trade off which is usually called Exploration-Exploitation-Trade-Off. This is one of the Fundamental challenges in RL, with a lot of methods proposed to handle but not completely avoid this problem.

1.2 Problem Statement

[2] Proposed an algorithm, c51, which maintains a distribution over the returns. This algorithm outperformed many other methods in Arcade Learning environment [3]. Unlike the slandered Q-learning we maintain an approximate distribution instead of the expected return. This implies availability of more information. We try to leverage on the this extra information to achieve efficient exploration.

1.3 Structure of the Thesis

Basis of reinforcement learning and major algorithms are explained in Chapter 2. There are may methods to deal with exploration exploitation trade off in RL. Some of the classic methods more sophisticated methods for exploration are described in Chapter 3. Chapter 4 deals with some of the exploration methods in distribution RL setting. We propose an algorithm which deals with exploration in stochastic environment in Chapter 5. The results and comparisons of our algorithm on different environments are stated in Chapter 6. Chapter 7 describes our conclusion and future work that can be done in this directing.

CHAPTER 2

Foundations

2.1 Reinforcement Learning

In Reinforcement learning, the learner is modeled as an 'agent' inside an 'environment'. The agent can perform actions which modify the environment and in turn the agent receives a reward for the action. This can be modeled formally as follows:

The environment is represented by a state $s \in S$. The action of the agent is an $a \in \mathcal{A}(s)$ where $\mathcal{A}(s)$ represents the set of actions that can be performed in environment state s. After performing the action, the environment state changes from s to s' and the agent receives the reward r. Then this cycle repeats.

In general, the reward gained by the agent depends on all the past actions, rewards and states. This is also true for the state transitions as well, the probability of transitioning into a given state is dependent on all previous states and actions. Naturally we would like these relevant information to be stored as succinctly as possible. A simplifying assumption that allows this is the Markov property.

A system is said to have the Markov property if the state succinctly stores all the relevant information required to compute the rewards and transition probabilities. Thus, given the state and the performed action, we can completely determine the transition probabilities and the associated reward. This is shown by the equation:

$$Pr\{s_{t+1} = s', R_{t+1} = R' | s_0, a_0, s_1, a_1, R_1, \ldots s_t, a_t, R_t\} = Pr\{s_{t+1} = s', R_{t+1} = R' | s_t, a_t\} \tag{2.1}$$

Here s_i, a_i, R_i represent the state, action and reward at time $t = i$. This equation holds true for all variables s_i, a_i, R_i. Any system that satisfies this Markov property is said to be a Markov Decision Process(MDP). MDPs can be succinctly represented by a 4-tuple.

- A set of States- S
- A set of actions- A

- Transition probability from state s to s' under an action a- $P_a(s, s')$
- Reward awarded when transitioning from state s to s' under an action a- $R_a(s, s')$

A practical variant of the above definition considers a 'discount factor' γ, which dictates that rewards at a later point in time are less valuable than in the present.

To determine the optimal behavior of the agent in an environment, we must first characterize what we mean by 'behavior'. The behavior of the agent under an environment can be modeled using a policy π. It defines the action taken under a specific state of environment. It can range from a simple state to action mapping to more complicated cases in which the policy can even be stochastic. After we have seen how to characterize the behavior of an agent, we need to evaluate it.

To estimate the worth of a policy we can define what we call as a value function. It is the expected value of cumulative discounted rewards that can be accumulated from a state by following a certain policy. Similarly, the Action Value function gives the expected reward staring from a given state under a given policy if a given action is taken.2.2

$$Q_\pi(s_t, a_t) = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | s_t, a_t] \tag{2.2}$$

2.1.1 Q-learning

Q-Learning proposed by [18] is used we have only some or no information of the MDP. The agent makes use to the information s_t, a_t, r_t, s_{t+1} sampled form the MPD. From equation 2.2 we arrive at

$$Q_\pi(s, a) = r + \gamma P_\pi(s'|s, a) Q_\pi(s', \pi(s')) \tag{2.3}$$

If we act greedily at every step.

$$Q(s, a) = r + \gamma P_\pi(s'|s, a) \max_{a'} Q(s', a') \tag{2.4}$$

Bellmen operator τ is defined as

$$\tau Q(s, a) = r + \gamma P_\pi(s'|s, a) \max_{a'} Q(s', a') \tag{2.5}$$

It is proven that the bellman operator converges to a unique fixed point where τQ* = Q*.The aim of Q learning is to find this fixed point thus the optimal policy.

This Q function can iteratively be estimated using the samples from the MDP by applying the equation

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \eta(r_t + \gamma \max_{a'} Q_t(s_{t+1}, a') - Q_t(s_t, a_t)) \tag{2.6}$$

2.2 Distributional Reinforcement Learning

In traditional RL we find optimal policy by estimating the expected value function for each state(or state,action pair). Where the value Work has been done on, bellman equations for variance(sobel 1982), Bayesian Uncertainty(Engel 2003) and also for higher moments of the value distribution.
Instead of calculating for individual moments, here we try to estimate the whole value distribution, all at once. The bellman equation for the value distribution will look something like $Z^\pi(x,a) :\overset{\mathrm{D}}{=} R(x,a) + \gamma Z(X', A')$, Where $Z^\pi(x,a)$ is random variable for return in state x with action a, and $Z(X', A')$ is the random variable capturing the randomness of what the next state will be and the next state value distribution. Lets look at the bellman operator T^π, $T^\pi Z(x,a) \overset{\mathrm{D}}{=} R(x,a) + \gamma Z(X', A')$.

The bellman operator for expectation is a contraction under the infinity norm. We can use the same norm and prove that the expectation of the distributions converge for our operator but we cannot say that the distributions always converge . We need to quantify the difference in the probability distributions. There is KL divergence which is usually used for this, but in our case the reward function domain varies a lot i.e. the probability distributions won't have the same support. The KL divergence in such case will be infinite, which is not helpful.

The Wasserstein metric is the L_p distance between the inverse CDF's of the probability distributions. Z is a set of probability distributions, so we take the supremum of the Wasserstein distances. $\|Z_1 - Z_2\|\, w := supx, aw(Z_1(x,a), Z_2(x,a))$ Theoretical bounds are proved with this wasserstine metric , but for practical application we go back kl divergence with fixed support

2.2.1 Algorightm : C51

The proposed algorithm C51 is similar to DQN. In DQNs we try to estimate a value using neural networks, here we try to estimate the distribution by taking the output layer as discrete distributions with 51 possible values distributed uniformly across some interval. So we approximate the random variable Z with a discrete distribution with 51 values(bins). (as number of bins increase the performance increases as less approximation is done, but so will the amount of computation). The steps in the algorithm are as follows:

1) Sample a transition x, a, r, x'.

2) Compute a sample backup $T^{\pi}Z(x, a) := r + \gamma Z(x', a')$ where π is the greedy policy.

3) Project the sample backup onto a fixed support.

4) Calculate the KL divergence between the sample backup and the current distribution, which is the loss.

At every iteration we are trying to move towards the sample backup(loosely,the distribution is moving towards the current sample).The sample comes from the underlying distribution. So in expectation we are moving towards the actual value distributions. We are not using Wasserstein loss as we just have the sample at each step not the actual distribution. If there is a way to calculate the Wasserstein loss between the actual distribution, we can slowly converge to it. As we have only a sample we might get a biased gradient(in expectation we will get the true gradient). Thus we project into a fixed support and calculate the KL divergence.

CHAPTER 3

Exploration

Exploration is one of the main challenges in Reinforcement Learning. Exploration is, taking actions in order to gather information about the environment. The long term objective of reinforcement learning agent is to maximizes the returns and in order to do so it has to find the necessary actions, making exploration an important task. The agent should also exploit by taking *optimal* actions form our experience to maximize the reward. If the environment is stochastic we have to take the same action multiple times to get a good estimate of the reward and dynamics of the environment. This section provides an overview of the strategies which address this exploration-exploitation trade-off.

3.1 Undirected Exploration

3.1.1 $\epsilon-$Greedy

This is one of the basic strategy. The idea is to pick the optimal action $1-\epsilon$ times and pick a random action ϵ times. Basically we explore with ϵ probability and exploit with $1-\epsilon$.

$$\pi(s) = \begin{cases} \text{argmax}_a(Q(s,a)) & \text{with prob } 1-\epsilon \\ \text{random}(\mathcal{A}) & \text{with prob}\epsilon \end{cases}$$

This method can never converge to an optimal policy as there is always a small chance of choosing a random action. To handle this ϵ is usually decayed to zero, stopping exploring. The decay rate usually depends on the environment and amount of exploration we want. Small decay rate can lead to taking many suboptimal actions while large decay rate can lead to less exploration. Find the right rate is usually a difficult task.

3.1.2 Boltzmann-Policy

The Boltzmann-Policy is on the widely used exploratory policy. Unlike ϵ greedy this policy assigns different probabilities to actions based on the Q-values. A temperature parameter β is used to control the amount of dependence on Q-values. At each state action gets sampled based on the probability assigned.

$$\pi(a|s) = \frac{\exp(Q(s,a)/\beta}{\sum_{a' \in \mathcal{A}} \exp(Q(s,a')/\beta)}$$

The dependence of action on Q-values helps in not taking actions which are significantly worse than others. In the initial stages of learning the exploration depends on the initialization of Q-values and may lead to some noise. The temperature parameter is adjusted to control the amount of exploration. Although β is usually same for all state sometimes it can be state dependent based on the amount of rewards of that particular state. Similar to ϵ greedy the β can be decayed to 0 to achieve a greedy policy.

3.2 Directed Exploration

3.2.1 Count based Exploration

In count based exploration the number of visits to each state (or state action) is maintained. This count along with Q-values is used to define the exploratory policy. One of the standard ways is to use the count as an intrinsic reward along with the actual reward to motive visiting states (or take actions) which are less frequently visited. One of the popular algorithm is UCB where the rewards are modified as below.

$$r^{+}(s,a) = r(s,a) + \frac{\beta}{\sqrt{N(s,a)}}$$

Where β is theoretically derived constant [7]. As we visit the state more frequently the intrinsic reward dies off. The drawback of this method is we need to have a table to maintain the counts and thus cannot generalize to continuous state space.

3.2.2 Pseudo counts

Keeping the exact count for large(continuous) state space is neither efficient nor effective exploration method. Instead an estimate of this value can be used, usually called pseudo counts. Few methods have been introduced to estimated these pseudo counts. One of the basic methods is to use hashmap which maps closer states to same bin and this reduced space can be used to keep track of the counts same like tabular method. Auto encoders can be used to learn this hash map or we can use basic hash functions like LSH hash[15]. Density models can be used to compute pseudo counts by estimating a distribution over the state space which accounts for number of times the state is visited. [1] says that this pseudo count $\hat{N}(s,a)$ is similar to actual count $N(s,a)$. $\hat{}(N)(s,a)$ can be obtained by solving simple system of equations.

$$\rho_n(s,a) = \frac{\hat{N}_n}{\hat{n}} \qquad \rho'_n = \frac{\hat{N}_n + 1}{\hat{n} + 1}$$

$$\hat{N}_n(s,a) = \frac{\rho_n(s,a)(1 - \rho'_n(s,a))}{\rho'_n(s,a) - \rho_n(s,a)}$$

The exploration bonus used in [1] is

$$r'(s,a) = \frac{\beta}{\sqrt{\hat{N}(s,a)}}$$

The choice of density model depends on the environment. Neural density models like Pixel Pixel CNN [17] or PixelRNN [11] are suitable for image data.

3.2.3 Curiosity Driven Exploration

Identifying states that are not yet explored and directing our agent towards those states is the main aim of curiosity driven exploration techniques.

[6] Tries to model the environment dynamics with a model $p(s_{t+1}|s_t, a_t, \theta)$. By maintain this model they try to direct the agent towards states where most information about the environment can be achieved. In other words the agent takes actions which reduce our uncertainty of our understanding of the environment. This is done by using incentric

rewards

$$r'(s_t, a_t, s_{t+1}) = r(s_t, a_t) + \eta D_{KL}[p(\theta|\xi_t, a_t, s_{t+1})||p(\theta|\xi_t)]$$

Where ξ_t is history of samples obtained till time t.

Similar to the above method [13] models the dynamics of the environment. They make use of the predictions error of the model as an incentric reward.

$$r^i = \frac{\eta}{2} \left\| \hat{\phi}(s_{t+1}) - \phi(s_{t+1}) \right\|^2$$

Where $\phi(\hat{s_{t+1}} = f(\phi(s_t), a_t; \theta)$ is the feature representation of the state s_{t+1} as predicted by over model and $\phi(s_{t+1})$ is the feature of the state actually reached taking action a_t at s_t. The idea is, if we did not visit the state or we are uncertain about that transaction, it will result in more error and driving us towards that state.

CHAPTER 4

Literature Survey

[8] Used the variance of the return distribution as an incentric reward. The variance can be caused by parametric uncertainty or model uncertainty. To only consider parametric uncertainty the variance is decayed over time. Further to avoid model uncertainty only The Right Truncated Variance 4.1 is used as an incentric reward.

$$r' = \frac{1}{2N} \sum_{i=N/2}^{N} (\theta_{N/2} - \theta_i)^2 \tag{4.1}$$

Where θ_i is the i/N^{th} quantile of the return distribution.

[19] Used hierarchical reinforcement learning for exploration in a Distributional RL setting. Leveraging on the uncertainty of the environment exploration by op tins framework is done. In Distributional RL we usually act on the expected mean. In this paper different options are created where each option corresponds to acting on different quantiles of the return distribution. This helps in exploration in different directions. Further this method is extended to continuous action space.

[16] assumed a distribution over the model parameters. This distribution is approximated to be consistent with the sample data. Formally $\mathrm{KL}[\hat{Z}||Z_\theta]$ should be minimum where θ are the parameters. $q_\phi(\theta)$ is an assumed approximate distribution over the parameters. Our objective function will be to minimize $\mathrm{KL}[q_\phi(\theta)||p(\theta|X)]$, where X are the samples. By assuming a improper uniform prior this loss function will be equivalent to 4.2. The first term corresponds to minimizing the error with respect to the data. The second term, which is the entropy encourages uncertainty of θ thus ensuring exploration.

$$\min_{\phi} E_{\theta \sim q_\phi}[-\sum_{i=0}^{N} \log Z_\theta(x_i)] - H(q_\phi(\theta)) \tag{4.2}$$

4.1 Double Uncertain Value Network

[9] Leverages on uncertainty to achieve directed exploration.They make use of two sources of uncertainty. Parametric uncertainty by assuming the parameters are not fixed. Return uncertainty by maintaining a distribution over the returns. [10] is an extend this work by using categorical distribution for returns and achieved some interesting results.

Parametric uncertainty: We consider the parameters as random variables and maintain a distribution. Thomsan sampling in multi arm bandits works in a similar way by maintaining a distribution and finding our posterior at every step until convergence. Bootstrap DQN [12] approximates parametric uncertainty in neural networks using non-parametric bootstrap, in RL. Instead of having multiple networks, in this paper a bayesian neural network is used.

Return uncertainty: In deterministic environments the stochasticity is only induced by the policy. For a deterministic policy if the return distribution of some state,action has some variance it implies it did not converge implying a need to explore that state,action. One way to address this is by using variance as an exploration bonus[8]. But intrinsic rewards may lead to unstable behaviour of the model while learning. In this paper authors sample a value from the return distribution and use it for selecting action. Usually in ϵ-greedy exploration method for distributional RL, at each state we select actions which have the highest expected return with epsilon randomness. Here instead of using the expected return we use a sample. The intuition is if the distribution for that particular state,action converged then the sample will be the same as expected return(as deterministic), if not converged this the expected mean may not be the true return and thus taking a sample will add some randomness to our action selection not compromising on the actual value of the return. Also usually the neural networks assume a uniform distribution at initialization. Thus more exploration will take place initially which keeps on decaying as the model converges.

CHAPTER 5

Contributions

In RL we try to maximize the expected return. C51 empirically showed that maintaining a distribution improved the learning ability of the agent. Though the C51 algorithm maintains a distribution over returns, the greedy policy used is based on maximizing expected return, with random exploration. If we apply the C51 algorithm for deterministic MDPs the final return distribution we obtain will be a dirac delta function. We initialize the model parameters randomly and thus the return distributions for the state,action pairs take some "random(mostly uniform)" distributions. As we keep on learning they keep moving towards the true return distribution of the policy. The policy we follow in duvn is, at every state we act greedily on a sample not on the expectation i.e for every action we sample a probable return for that state,action pair form the distribution given by our model. The action we choose is the one having the highest value of the sample. This way during the initial stages of learning we do not always go with the optimal action thus encouraging exploration. Once the model converges to the dirac delta distributions(the true distribution) we follow the optimal policy.

For stochastic MDPs this is not the case as the model doesn't converge to the dirac delta distributions but rather to a distribution with some variance. If we follow the duvn policy then even after convergence there is a good probability of the policy being not optimal. Even for some deterministic MDPs like cartpole (6.3) with large state space, convergence to the true distribution is very slow. Even though the model doesn't converge to the true distributions the greedy policy on expected returns would still be an optimal (close to optimal) policy. Refer Section 6.3.1 for more details.

5.1 Algorithm

One way to handle this is by taking multiple samples. For every state action pair we can take multiple samples and act greedily on the average. The number of samples is something that should be carefully chosen. If we take too many samples then the average will just

be the expectation resulting in less exploration. If we take only a few samples then we won't choose the optimal action at every state. The number of samples should be increased slowly such that it encourages initial exploration and later exploitation. It should depend on the confidence of our distribution and this can be captured by the number of times the state,action pair is visited.

Algorithm 1: Multi DUVN

Input: $\mathcal{M} : (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P})$
Output: $\mathcal{Q}^*(\mathcal{S}, \mathcal{A})$

Intialize $p_\Phi(\phi)$ parameters Φ with random values.;
Intialize $\tilde{N}(s, a) = 0$;
Intialize Buffer = [ ];

for $e \leftarrow 0$ **to** E **do**
  $s_0 \sim \mathcal{M}$;
  set t = 0 **set** done = 0;
  sample $\phi \sim p_\Phi(\phi)$;

  while ***not*** *done* **do**
    for $a \in \mathcal{A}$ **do**
      $n = \tilde{N}(s_t, a)$;
      sample $\{r_1, r_2 ... r_n\} \sim p_\phi(Z|s_t, a)$;
      $R[a] = avg(\{r_1, r_2 ... r_n\})$;
    end
    get $a_t \in \text{argmax}_{a'}\ R$;
    observe s_{t+1}, r_t, done $= \mathcal{M}(a_t)$;
    store Buffer $\leftarrow (s_t, a_t, r_t, s_{t+1})$;
    update $\tilde{N}(s_t, a_t) = \tilde{N}(s_t, a_t) + 1$;
  end

  Sample batch $\sim$ Buffer;
  Train $\Phi \leftarrow$ batch;
end

The parameters Φ are trained using the c51 algorithm. [2].For parametric uncertainty similar to duvn[9] we use the Bayesian inference method proposed in [5]. The idea of taking multiple samples is that as we learn about the environment our predictions will be more precise. As the goal is to maximize the expected returns we want to act greedily on the expected returns. In stochastic environments we need to visit states multiple times to learn either the distributing or the expected value due to the uncertainty.

The central limit theorem states that the sample mean of any random vairable irrespective of the undleying distribution tends towards a normal distribution $N(\mu, \sigma^2/n)$ where n

is the number of samples, μ, σ are the mean and variance of the underlying distribution. This shows the stability of our method and connection with thompson sampling in RL. This theorem can be used to try and prove some theoretical bounds for this method

5.1.1 Working with continues state spaces

Keeping track of number of visits to states for discrete state space can be done by maintaining a table. For continues state spaces, due to the sparsity this methods doesn't work. We can

One of the basic methods we can use to handle this is by discretizing the state space with limited range. This can be done for environments with small feature vector. Also limits for feature values should be know. Environments where states are represented with complex feature vector like images this cannot be directly applied. Though this can be applied feature representation of the image, learnt by the network, the range of values and size makes it difficult.

LSH hash is a hash function which maps vectors which are close to each other to the same key. In 5.1 x is a vector sampled from a normal distribution. w, b are parameters to be decided. This seems better than the above method but applying this to cartpole problem did not give good results.

$$h^{x,b}(\vec{v}) = \lfloor \frac{\vec{v}.\vec{x} + b}{w} \rfloor \tag{5.1}$$

Similar to Section 3.2.2 we can use pseudo counts for complex state spaces. The drawback is increase in time complexity. This method was not implemented but it is something that should be tried to evaluate the performance.

CHAPTER 6

Experiments and Results

6.1 Chain

Chain is a basic MDP with $\mathcal{S} = \{1, 2. \ldots N\}$ state space and sparse rewards. We always start from state 1. At each state there are two actions possible 0, 1. One action corresponds to going right and the other action leads to termination. Reaching the last state leads to termination with a reward of 1. Rewards for other states is 0. There are two chain types, ordered and unordered. In an ordered chain, taking action 0 will lead to termination and taking action 1 leads to moving to the next state. In the unordered one the actions corresponding to 0, 1 are selected randomly at initialization of the environment.

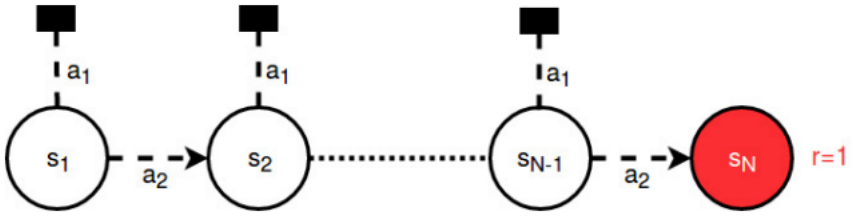

Figure 6.1: Illustration of chain environmenta_1, a_2 at each state can be either 0,1 or 1,0 . Image taken from [10]

Reaching the final state with undirected exploration methods like ϵ-greedy is very difficult. Even with an ϵ of 0.5 the probability of reaching the final state is $\frac{1}{2^N}$ which is quite low. Thus this environment is a good test for exploratory methods. The results can be seen in figure 6.3. Epsilon-greedy could not reach the last state for even $N = 25$ chain. [10] Used ucb exporation for duvn on chain environment. m-duvn shows on par performance on chain-25, on chain-50 it reaches the last state a few time steps after duvn. The main distavantage of ucb is instability of the rewards which can be seen in the graphs.

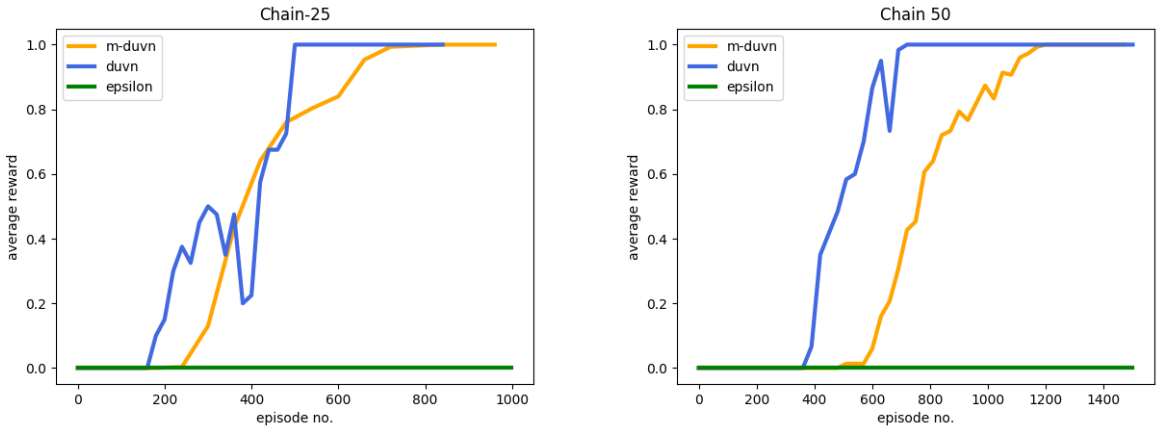

Figure 6.2: result averaged over 5 runs, ucb exploration is used for duvn as proposed in [10]

6.2 Frozen Lake

Frozen Lake is a gridworld environment with one goal state $\mathcal{S} = \{1, 2 \ldots N \times N\}$. We start at the top left corner and the goal state is situated at the bottom right corner. The reward for reaching goal state is 1 and the episode gets terminated. At any state we have 4 actions 0,1,2,4 indication left, down, right, up. Invalid action(corner states) leads to no change in state. Apart from the actions to goal state any action has a reward of 0. The environment also contains some 'hole' states which upon reaching terminates with a reward of 0. Also the environment terminates after 2(N-1) steps. There are Deterministic and stochastic versions of this environment. In the stochastic version, action leads to the correct state (i.e. left action leading to going left) with p probability and other three states with $\frac{1-p}{3}$ probability.

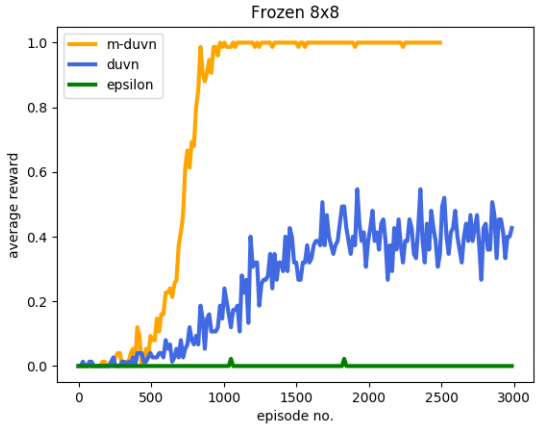

Figure 6.3: Deterministic actions. Result averaged over 5 runs

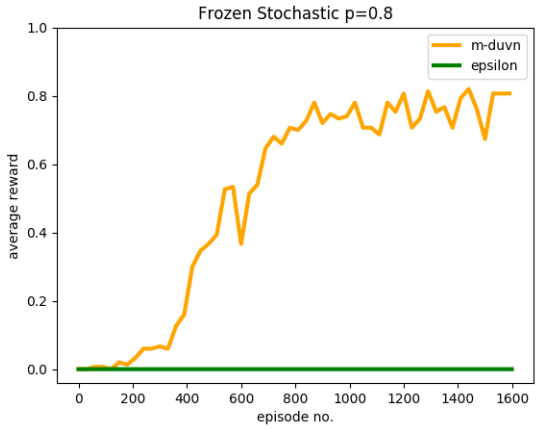

Figure 6.4: Stochastic actions with correct direction probability = 0.8. Result averaged over 5 runs

Duvn has reached the goal state quite early but it will take some time for the return distribution to become dirac delta. [10] Showed duvn reaches the optimal policy after 40000 steps. m-duvn performed much better in this environment than duvn. Once enough exploration is done only optimal policy is followed unlike duvn which keeps on exploring even though we are confident that some action gives zero reward.

Figure 6.4 shows results on stochastic frozen environment with p = 0.8, the probability that we move in the action we take. Duvn is only for deterministic environments thus has not been experimented with this environment. We can see that m-duvn policy has a expected return 0.8 which should be close to the actual expected return of the optimal policy.

6.3 Cartpole

Cartpole is a more complex control task with continuous state space

$$\mathcal{S} = [(-2.4, 2, 4), (-inf, inf), (-0.24, 0.24), (-inf, inf)]$$

. Cartpole, also called an inverted pendulum problem where a pole is attached by an un-actuated joint to a cart ,figure 6.6. The objective is to keep the pole upright through controlling the cart. The two actions $\{0, 1\}$ possible at any state moves the cart to either right or left respectively. The current state is represented by cart position, cart velocity, pole angel, pole angular velocity. If the cart position or pole angles exceeds the threshold the episode terminates. The reward for surviving every is 1 for every time step. So the longer the episode runs, more the net reward. If the episode length exceeds 200 the episode terminates.

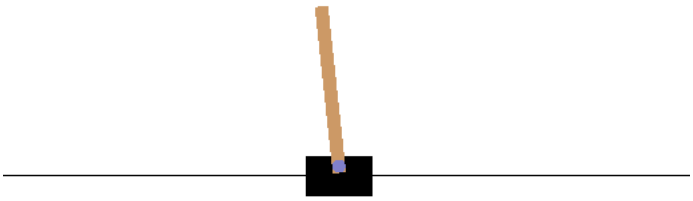

Figure 6.5: Screenshort from [4]

m-duvn showed good performance on cartpole.It is on par with ϵ greedy. Discretization of state space was done to maintain counts As cartpole has dense rewards it is not a difficult to explore environment and thus ϵ greedy works fine. . [10] Did not show performance of duvn on cartpole. Our implementation showed a bad performance on this environment.

6.3.1 Uncertaninty in cartpole

The bad performance on cartpole by duvn can be explained by the slow convergence of the return distribution. In simple terms duvn explores when it's time to exploit. Figure 6.7 is

the return distribution of state $[0.1, 0.1, 0.18, 1]$ for action 0 to the left 1 to the right. The optimal action here is 1. An agent action on expectation takes the optimal action 1. But duvn has a high probability of taking action 0 since the sample from the return distribution of action 0 has non negligent probability

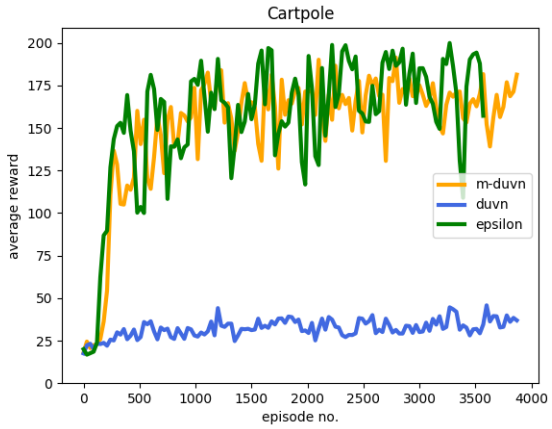

Figure 6.6: Results averaged over 2 runs

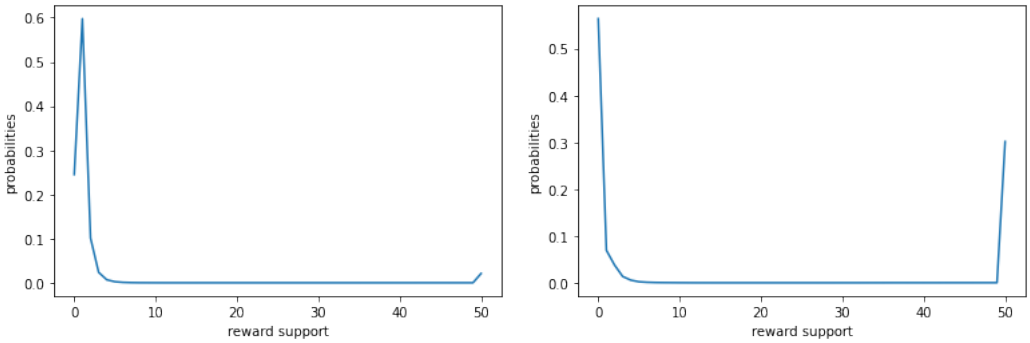

Figure 6.7: Return distributions

CHAPTER 7

Conclusion & Future work

In this thesis we looked at various exploration methods and their basic functionality. Some methods which made use of the extra information in Distributional RL for efficient exploration were introduced. Algorithms which made use of the uncertainty in Distributional RL were explored. An algorithm which make use of the uncertainty and also strikes a good balance between exploration and exploitation was proposed. Some of the problems with this approach and handling them was discussed. Finally, experiments on different environments empirically showed efficiency of this algorithm.

7.1 Future work

Implementation and testing of this algorithm for more environments with continuous and complex feature representation should be done. Only a small amount of information available in Distributional RL setting is being used. Approaches like information directed exploration should be experimented with. Prioritized experience replay show good promise in RL. it Should be implemented of better learning. Although good empirical results can be seen theoretical bounds for converges atleast for linear parameterization is ought to be studied.

REFERENCES

[1] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in neural information processing systems*, pages 1471–1479, 2016.

[2] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 449–458. JMLR. org, 2017.

[3] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[5] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *arXiv preprint arXiv:1506.02142*, 2015.

[6] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*, pages 1109–1117, 2016.

[7] J Zico Kolter and Andrew Y Ng. Near-bayesian exploration in polynomial time. In *Proceedings of the 26th annual international conference on machine learning*, pages 513–520, 2009.

[8] Borislav Mavrin, Shangtong Zhang, Hengshuai Yao, Linglong Kong, Kaiwen Wu, and Yaoliang Yu. Distributional reinforcement learning for efficient exploration, 2019.

[9] Thomas M Moerland, Joost Broekens, and Catholijn M Jonker. Efficient exploration with double uncertain value networks. *arXiv preprint arXiv:1711.10789*, 2017.

[10] Thomas M Moerland, Joost Broekens, and Catholijn M Jonker. The potential of the return distribution for exploration in rl. *arXiv preprint arXiv:1806.04242*, 2018.

[11] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016.

[12] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn, 2016.

[13] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 16–17, 2017.

[14] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. 2011.

[15] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in neural information processing systems*, pages 2753–2762, 2017.

[16] Yunhao Tang and Shipra Agrawal. Exploration by distributional reinforcement learning. *arXiv preprint arXiv:1805.01907*, 2018.

[17] Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in neural information processing systems*, pages 4790–4798, 2016.

[18] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[19] Shangtong Zhang and Hengshuai Yao. Quota: The quantile option architecture for reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5797–5804, 2019.