Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Computer Science

DIPLOMA THESIS ASSIGNMENT

Student: Moravová Alena

Study programme: Open Informatics
Specialisation: Artificial Intelligence

Title of Diploma Thesis: Exploration strategies for reinforcement learning with function approximation

Guidelines:

Reinforcement learning (RL) with function approximation recently led to a super-human performance in playing video games or the game of go. These methods repetitively solve the problem in simulation and learn from experience. Computational resources and the amount of data required for this training are often huge, but they may be substantially reduced by efficient exploration of the space of possible solutions to the problem. In this work the student will:

1) review existing exploration strategies for RL with both tabular and function approximation representations
2) implement at least two existing strategies within the RL agent for classification with costly features developed at CTU
3) design a new exploration strategy for RL with function approximation, ideally based on ideas from tabular representation or Monte Carlo Tree Search
4)experimentally compare the effectiveness of the existing and new exploration strategies in the task of classification with costly features + one more domain.

Bibliography/Sources:

[1] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. Vol. 1. No. 1. Cambridge: MIT press, 1998.
[2] Osband I, Blundell C, Pritzel A, Van Roy B. Deep exploration via bootstrapped DQN. In Advances in Neural Information Processing Systems 2016 (pp. 4026-4034).
[3] Even-Dar, Eyal, Shie Mannor, and Yishay Mansour. "Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems." Journal of machine learning research 7.Jun (2006): 1079-1105.
[4] Putta SR, Tulabandhula T. Efficient Reinforcement Learning via Initial Pure Exploration. arXiv preprint arXiv:1706.02237. 2017 Jun 7.

Diploma Thesis Supervisor: Mgr. Viliam Lisý

Valid until the end of the winter semester of academic year 2018/2019

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
Z.S.
-4-

prof. Dr. Michal Pěchouček, MSc.
Head of Department

prof. Ing. Pavel Ripka,CSc
Dean

Prague, June 21, 2017

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, January 9th, 2018 ....[illegible]..................

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science

Master Thesis

Exploration strategies for reinforcement learning with function approximation

Alena Moravová

Supervisor: Mgr. Viliam Lisý, MSC., Ph.D.

Study Programme: Open Informatics, Master program

Field of Study: Artificial Intelligence

January 9, 2018

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, January 9th, 2018 ....

Abstract

This thesis investigates exploration strategies in high-dimensional reinforcement learning problems. Three existing exploration approaches were evaluated on two domains – the Classification with costly features task and the Cart-pole problem. Based on the researched strategies a novel method for exploration is proposed: the Uncertainty driven exploration. Our approach combines two other methods in a novel way which allows us to use model's uncertainty as a guidance for exploration. The proposed approach is evaluated together with the other three selected strategies and the baseline ϵ-greedy strategy and is shown to have competitive results.

Abstrakt

Tématem této diplomové práce jsou explorační strategie ve vysokodimenzionálních problémech posilovaného učení. Tři současné explorační strategie byly evaluovány na dvou doménách – na klasifikaci s drahými proměnnými a na problému balancování tyče na pohybujícím se vozíku. Na základě zkoumaných strategií byla navržena nová strategie nazvaná Explorace řízená nejistotou. Náš přístup kombinuje dvě metody, které umožňují použití odhadu nejistoty modelu pro řízení explorace. Všechny strategie byly porovnány se základní ϵ-greedy strategií. Navržená strategie má srovnatelné výsledky s ostatními evaluovanými přístupy.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Reinforcement learning (RL) agents are algorithms which are dealing with decision making in problems with distant rewards. Unlike standard supervised learning algorithms with input-target pairs, RL algorithms don't receive feedback about their decisions at every timestep and thus RL agents have to execute many actions until any feedback is given. In order to find the optimal paths (sequences of actions) reinforcement learning agents have to try out many trails – ideally all the alternatives. However, in many applications of reinforcement learning it is impossible to try out every possible sequence of actions so the focus has to be brought to the learning efficiency of the agent. Learning efficiency in reinforcement learning problems is equivalent to finding the best trade-off between exploration and exploitation of the environment. The problem is present the most in the online training cases (when the agent is learning while being deployed in the real world). But even in the offline training scenarios (learning via simulations, using training samples) the time-efficiency of the training is important as the training can take days or up to several weeks until the final convergence.

Reinforcement learning can be divided into tabular and continuous state-space cases. The traditional RL research is mostly concerned with the tabular / finite state space problems, where the number of states is countable and the states are known. Examples of these problems are some card games (with reductionist representations) or trivial maze problems. In the continuous state-space cases none of those conditions hold. The number of states is either infinite or the least practically uncountable and the states are inherently unknown. Within such problems belong most card games (without simplistic representations), all problems with visual input (such as computer games or robots navigating in the real world) and many others. In general all real world problems which could be of any interest are continuous state-space problems.

The main purpose of exploration is to prevent premature convergence to a local optimum, however, finding an efficient exploration strategy is non-trivial. In the traditional tabular RL tasks the problem of efficient exploration are extensively theoretically and practically studied and for specific cases solutions which are polynomial in the number of states have been found (E3 (MIT) and R-MAX (Brafman and Tennenholtz, 2002)). However, these solutions aren't really applicable in practical problems of even medium size, as even though they are polynomial in the number of states, the solutions are mostly exponential in the number of state variables. More practical approaches to the problems are solutions based on characteristics of the underlying mean value distributions (Thompson sampling (Thompson, 1933)) or based on theoretical approaches from Multi-armed bandits (such as counting of state-visits).

More recently the attention of reinforcement learning research has been shifted towards the high-dimensional state-space problems (for instance playing Atari games based on visual input) where the application of the traditional approaches for tabular MDPs is not straightforward because of the intractable number of states. Most of the successful algorithms for the high-dimensional RL problems base their exploration methods on simple randomization of the actions and new strategies have only recently been added to the algorithms. Selection of the best exploration strategy for high-dimensional state-space environments is therefore still an open problem.

1.2 Thesis contribution

The objective of the thesis was to compare at least two state-of-the-art exploration methods in high-dimensional state-space problems and to propose one new exploration strategy. Three contemporary exploration methods were implemented and evaluated: Parameter-noise, Dropout and Visits-count method. Each of the strategies was selected as a representative of a distinct approach to the problem of exploration in RL.

The main contribution of this thesis is the direct comparison of these strategies and evaluation of their applicability to the selected problem of Classification with costly features (CCF) (Tan, 1993) and the task of pole balancing on a cart (Cart-pole). The research of current approaches exposed the lack of direct comparisons of different strategies. Most of the contemporary methods are only compared with a simple randomization method (ϵ-greedy, see Section 3.1.1).

The second contribution is our proposed exploration method: Uncertainty driven exploration, which is based on two existing approaches: the Optimism in the face of uncertainty, which is an exploration method from the traditional tabular RL algorithms, combined with the Distributional perspective to reinforcement learning (Bellemare et al., 2017).

Our new approach employs the Distributional perspective to RL in high-dimensional state-spaces as a novel way of measuring the model uncertainty, which is then used as a guidance for the exploration. Prior methods based on model uncertainty used more indirect measures such as using variance in the model weights.

Chapter 2

Background

This chapter provides an introduction to the problem of reinforcement learning (RL) and a summary of existing approaches. In Section 2.1 there will be introduced the theoretical foundations of reinforcement learning. Existing solutions to RL problems will be described first in the simpler tabular case (with finite and countable states and actions), later also in the high dimensional case.

Section 2.2 provides a short introduction to the artificial neural networks. First the algorithms behind basic neural networks will be described and then deep neural networks which are currently widely used as function approximators in the high-dimensional case of reinforcement learning problems will be presented.

2.1 Reinforcement learning

All problems concerned with agents learning to operate in unknown environments through interaction fall into the area of reinforcement learning. A RL agent is maximizing total reward in the given environment by trying to select optimal action in each time step. In order to select the best possible action, the agent is learning an inner model of the environment.

The presented algorithms in this section are mostly taken from the Introduction to Reinforcement Learning (Sutton and Barto, 1998).

2.1.1 Tabular reinforcement learning

The tabular case RL problems include problems small enough that they can be represented as arrays or tables. In the very small cases even exact solutions can be obtained by means of Dynamic programming. The basic formulation of the tabular reinforcement learning problems is formulation via finite Markov decision processes (MDPs).

A Markov decision process (MDP) is a 5-tuple (S, A, P, R, γ) where S is a finite set of states, A is a finite set of actions available to the agent, P defined as $P_a(s, s_{t+1}) =$

$\Pr(s_{t+1} \mid s_t = s, a_t = a)$ is the probability of transition from state s to state s_{t+1} given an action a, R (or more concretely $R_a(s, s_{t+1})$) is the reward received after transitioning from state s to state s_{t+1} given an action a and γ is the discount factor with values between $[0, 1]$, which determines the importance ratio of future and present rewards.

In general MDP is a description of the environment where we expect the environment to have a finite number of states and the agent to have a finite number of actions available. If the MDP contains only a single state then the problems are called the Bandit problems. In the multi-state MDPs the solutions to the RL problems can be achieved by applying either Dynamic programming, Monte Carlo methods or Temporal-difference learning.

Given a description of the environment as MDP we can define the problem of reinforcement learning as the maximization of the expected discounted reward

$$\mathrm{E}[r_0 + \gamma \cdot r_1 + \gamma^2 \cdot r_2 + ...] = \mathrm{E}[\sum_{t=0}^{\infty} \gamma^t r_t], \tag{2.1}$$

where r_i is received reward at timestep i and the optimal state value can be calculated using recursive Bellman equations (Bellman, 1957)

$$V^*(s) = \max_a \{R(s, a) + \gamma \sum_{s_{t+1}} P(s_{t+1}|s, a) V^*(s_{t+1})\}, \tag{2.2}$$

where $V^*(s)$ is the value of state s, $R(s, a)$ is the expected reward for action a in state s. In order to follow the optimal policy the agent selects the action that maximizes the value $V^*(s)$ in each of the states.

The optimal values V^* can be obtained from the Bellman equation by backwards induction, either analytically or numerically. For high dimensional state spaces the values can be obtained by approximate dynamic programming using neural networks. The Bellman equation is the foundation for all algorithms in reinforcement learning.

Bandit problems

The Bandit problem which is also called the Multi-armed bandit problem is often compared to an agent choosing between multiple slot machines. More formally it is a scenario where an agent is selecting one among k different options (actions) given exactly one state. The agent receives a reward based on a probability distribution of the rewards given the selected action. The objective of the agent is to maximize the expected total reward over multiple time-steps.

Mathematically we can define the problem as selecting an action a so as to maximize the expected (mean) reward over T steps, the value of the action a is defined as

$$v(a) = \mathrm{E}_{t=1,..T}[R_t | A_t = a], \tag{2.3}$$

where $v(a)$ is the value of an action a, R_t is the reward given action a was selected in timestep t. However, the agent can't know the action value certainly, and thus a value estimate at timestep t. Therefore even in the simple case of multi-armed bandit problems there is the typical exploration vs. exploitation dilemma. Ideally the agent would want to select the optimal action in each time-step given the optimal Q-value, however, the real values of the actions are not known in advance, and they must be discovered by the agent. Thus the agent has to balance between exploiting given knowledge by using the currently best action or exploring also actions with currently worse value estimates. The multi-armed bandit problem serves often as the theoretical baseline the problem by itself is quite simple, yet it still contains the same profound issues as the more complex scenarios of reinforcement learning.

Dynamic programming

Dynamic programming (DP) (Bellman, 1954) is a collection of algorithms that can be used to optimally solve problems in the restricted scenarios of finite MDPs. However, the applicability of the DP algorithms is very limited, as they expect a perfect model of the environment and also are very computationally expensive. They still are of great importance as they build the theoretical foundations for most (all except the Monte Carlo methods) practical reinforcement learning algorithms. The other more practical RL algorithms can be seen as an attempt to achieve the same results as with DP but with smaller computational demands and without the perfect model of the agent's environment.

For calculation of the optimal policy in RL, it is useful to define a new quantity – the state-action value V. The optimal policy can then be obtained through maximization of the state-action value $V(s, a)$ in each step. The optimal value V is defined as:

$$V^{\pi}(s, a) = E[R|s, a, \pi], \tag{2.4}$$

where R stands for the random return of rewards after first taking action a in state s and then following policy π. The two basic approaches for computing the optimal action-value functions are value and policy iteration. Both algorithms are able to find optimal solutions from the sequence of functions $V_k,\ k = 0, 1, 2, \ldots$ which converge in finite number of steps to Q^*. However, the convergence to the optimal V^* involves computing the expected values over the whole state-space, which is impractical for all but the smallest MDPs.

The Value iteration (Bellman, 1957) algorithm is iteratively calculating state values using the Bellman equation. The policy for action selection is then obtained by selecting action leading to state which has the maximal value.

The Policy iteration (Howard, 1960) algorithm has two repetitive steps. First a policy is selected from the current state values, for each state an action leading to the state with maximal value is stored into the policy. In the next step the state values for the new experience are gather from executing the selected policy. The two steps are iterated until the convergence – until the policy doesn't change. In general the value and policy iteration lead to the same results, but the two approaches have slightly different convergence rates.

Monte Carlo methods

A more practical approach is not to evaluate the values exactly, but only to use the estimates of the exact state or state-action values. One such class of approaches are the Monte Carlo (MC) (Metropolis and Ulam, 1949) methods. In the Monte Carlo methods the agent doesn't need a complete knowledge of the environment. The action-value estimates are received by randomly sampling experience trails (of states, actions and rewards), therefore the knowledge of the environment transitions is not needed. We can either learn from simulated (from a model of the environment) or real experience. The agent's value estimates and policies for all visited states are changed based on the completed experience (after one episode).

Temporal difference methods

Another method for estimating the Q-values is the Temporal difference (TD) learning (Sutton, 1988) method. It can be seen as a type of unification of the Monte Carlo and Dynamic programming methods. Similarly to the MC methods the TD methods use sampling of whole experience trails for update and similarly to the DP methods the algorithms update the values based only on other learned estimates, they don't need to wait for the final reward.

The n-step algorithm updates the expected rewards from the next n steps. The TD(λ) (Eligibility traces) algorithm takes into account the length of history parameterized by the λ value. If $\lambda = 0$ only the next step is taken into account and the algorithm is equivalent to the TD algorithm, if $\lambda = 1$ then it is the same as the Monte Carlo methods.

The TD learning is based on the idea, that neighboring states' estimates are often correlated. Thus the algorithm is adjusting predictions about states based on more accurate predictions of states' values in more distant future. The idea is that states which are closer to the end of an episode should have more correct value estimates. This learning procedure is a form of bootstrapping.

One of the first successful TD algorithms in reinforcement learning was the Q-learning algorithm which was developed in 1989 by Watkins (Watkins, 1989). The Q-learning is also the algorithm underneath all tested exploration strategies and therefore it will be explained in more detail. In this algorithm similarly to the TD(0) method only the values from the nearest state are used for the update. The new Q-value estimate is calculated in each timestep as:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \cdot (R(s_t, a_t) + \gamma \cdot \max_{a \in A} Q(s_{t+1}, a) - Q(s_t, a_t)), \tag{2.5}$$

where α is the learning rate and γ is the discount factor as was defined in the MDP definition. The Q-value estimate of the state s_t in timestep t are updated only by the obtained reward $R(s_t, a_t)$ and the difference of the actual Q-value estimate and the estimate of state s_{t+1} – of the next observed state.

The discount factor determines how important the future rewards are for the current state. The higher the discount factor the further the rewards of a state will propagate in time. A discount factor of 0 would mean that an agent only cares about the immediate rewards. The corresponding optimal policy is then defined as:

$$\pi(s_t) = \operatorname*{argmax}_{a' \in A} Q(s_t, a'). \tag{2.6}$$

Despite the fact that the tabular methods are very powerful in the theoretical and small problems they cannot be directly applied to the high dimensional state-space problems. Thus the high-dimensional reinforcement learning problems have to be solved using function approximation methods which will be presented in the following section.

2.1.2 Function approximation reinforcement learning

As already mentioned the traditional reinforcement learning algorithms can work only in discrete state and action spaces. However, most of the real world applications of reinforcement learning require continuous or very high-dimensional state spaces. The problem is not only the memory requirements of the tables which would hold all the possible states, but also the fact that hardly any states will be ever revisited. One of the approaches to handle the continuous or very high-dimensional state spaces is the discretization of its variables. However, this approach still quickly leads to combinatorial explosion or the discretization may lead to loosing too much information and therefore it is not a preferable solution. The solution can be found in the generalization ability of algorithms which would be able to compress the states so that similar states would return similar action values. Generalization is a great advantage of the deep neural networks which also serve as the underlying models in most function approximation methods.

The basic distinction in function approximation methods is between value based methods which are learning model that returns value of provided state-action pairs and policy gradient methods which are learning policy model which maximizes an objective function (typically the expected reward).

Value based methods

The Value based methods are algorithms which directly approximate the Q-function. The most famous application of these approaches is the Deep Q-network (DQN) (Mnih et al., 2015). The DQN was the first reinforcement learning method which was really able to learn action policies directly from the visual input. It was the first algorithm that was able to play Atari 2600 games from the Arcade Learning Environment (Bellemare et al., 2013a) on human level and in some of the games the it even surpassed the human professional players' level. The DQN model is using a deep convolutional neural network to approximate the optimal state-action value function:

$$Q^*(s,a) = max_\pi \, \mathrm{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...|s_t = s, a_t = a, \pi], \tag{2.7}$$

which corresponds to the expected reward given a policy π and taking an action a in a state s.

The underlying model of DQN is a deep convolutional neural network trained by a loss function based on Q-learning. Given the high-dimensional visual input the model returns Q-values for each action and the agent then selects the action maximizing the obtained Q-values. The DQN loss function L is calculated using the original Q-learning function on drawn samples (batches) of experience. Each sample of experience is denoted as (s, a, r, s'), where s is the initial state, $s^{'}$ is the observed state after taking action a and r is the obtained reward. The corresponding loss function is defined as

$$L(\Theta) = \mathrm{E}_D[(r + \gamma \cdot \max_{a' \in A} Q(s', a', \overline{\Theta}) - Q(s, a, \Theta))^2], \tag{2.8}$$

where D is the batch of drawn samples (s, a, r, s'), Θ are the parameters of the model (network) and $\overline{\Theta}$ are the parameters which are used to calculate the target value. The target network parameters $\overline{\Theta}$ are only updated with some frequency from the parameters Θ and kept fixed in between. This removes from the training non-stationary targets and subsequently the convergence of the values can be more smooth.

As the function approximation model of DQN is a neural network the problem of dependence between training samples occurred because one of the assumptions of training of neural networks is that the training samples are i.i.d. (independently and identically distributed). The dependence between samples emerges as the consecutive samples from the agent's trails are naturally correlated.

In order to remove the correlation the authors of the DQN model used a method called Experience replay. The method removes the correlation between the samples by randomly sampling from a saved buffer of experience which breaks their correlation.

The run of the DQN algorithm has two main steps. First the agent collects experience from the environment based on the state-action values returned from the present model. The network's parameters are updated after several episodes. The training batches are sampled from the experience and the loss function $L(\Theta)$ is used to update the model's parameters. The two steps are repeated until convergence.

Since the success of the DQN model in year 2015, there were many subsequent algorithms solving the high-dimensional state space problems. One of those algorithms is the Soft-target network (Lillicrap et al., 2015) which follows the values of the first network with a delay. The parameters of the target network are updated at each step by $\overline{\Theta} = (1-\rho)\overline{\Theta} + \rho\Theta$, where ρ is the target network factor, $\overline{\Theta}$ are the target network's parameters.

Another is a direct follower of the DQN model – the Double DQN algorithm (van Hasselt et al., 2015). The authors have adjusted the algorithm so that the maximization bias induced by max in Q estimate of the original DQN model was reduced. The DDQN method is based upon the target network method. The estimate of the value of the state s' is obtained by combining the estimates of both – the original and the target – networks. The Q-value estimate of the next state s' is obtained by first selecting the maximizing action using the estimate from the original network, and then using this action to obtain Q-value, but the value is taken from the target network. Mathematically the Q-value of the state s' is obtained as $Q(s') = \gamma Q^{\overline{\Theta}}(s', \text{argmax}_{a'} Q^{\Theta}(s', a'))$. The $Q(s')$ estimate is then used instead of the term $max_{a \in A} Q(s', a)$ in the original Q-learning update.

Another adaptation of the DQN model was the Prioritised replay by (Schaul et al., 2015), where the experience used for the update of network was not selected randomly, but was ordered according to a measure of priority (usually proportional to the TD-error).

Policy based methods

The policy based methods or Policy gradient (PG) methods are optimization processes where the algorithm is trying to find the parameters Θ which are maximizing a policy objective function $J(\Theta)$. The Policy gradient algorithms are searching for the local maximum of J by following the gradient w.r.t. the parameters Θ.

The optimized policy given parameters Θ is defined as a weight vector π:

$$\pi_\Theta(a|s) = Pr(a|s, \Theta) \tag{2.9}$$

which is the probability that action a is applied in a state s. We define a performance measure $J(\Theta)$ which is used to update the policy parameters Θ.

The objective function $J(\Theta)$ is defined as the total discounted reward:

$$J(\Theta) = \mathrm{E}_{\pi\ \Theta}[v_1] = \mathrm{E}[r_1 + r_2 \cdot \gamma + r_3 \cdot \gamma^2 + ...|\pi_\Theta)], \tag{2.10}$$

where π_Θ is the policy given model with parameters Θ, r_t is received reward at time-step t while following policy π_Θ and γ is the discount factor. The gradient of J given the parameters Θ is then used to update the model. This basic algorithm for Policy gradients is called Reinforce (Williams).

Very successful Policy gradients methods is also the Actor-critic algorithm. It implements the basic idea of altering steps between an Actor and a Critic. The Actor uses and improves the current policy and the Critic evaluates the actor's policy. The aim of Actor-critic methods is to reduce variance so that the learning is more stable than in standard Policy gradient methods.

More formally the run of the algorithm is following. First the actor executes several episodes according to the current policy $\pi(\omega)$ and gathers experience from those episodes. The critic then evaluates the experience and finds the parameters Θ by following the approximate gradient:

$$\nabla_\Theta J(\Theta) \approx \mathrm{E}_{\pi_\Theta}[\nabla_\Theta \log \pi_\Theta(s,a) \cdot Q_w(s,a)] \tag{2.11}$$

The actor subsequently updates its parameters ω using the new Θ:

$$\omega = \omega + \alpha \cdot \Theta \tag{2.12}$$

Those three steps are repeated until convergence. The Actor-critic method combines both of the approaches – the value and policy based methods. The actor employs the Policy gradient idea and the critic the value approximation of the Q-function by evaluating the current policy.

One very successful Actor-critic implementation is the Asynchronous Advantage Actor-Critic (A3C) algorithm (Mnih et al., 2016). It surpassed by far the results of the DQN (Mnih et al., 2015) scores. In the A3C method the critic is evaluating the actor's policy by means of an advantage function which is defined as:

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s), \tag{2.13}$$

where the Q-values are estimated by an n-step sample of rewards starting from state s and action a and then following policy π and the state value function $V(s)$ represents the expected reward given state s and then following policy π.

2.2 Deep networks

Most of the present best reinforcement learning algorithms that solve the problem of function approximation make use of a deep neural network to obtain the estimates of their policy or state-action values. The underlying model used in the DQN (Mnih et al., 2015) algorithm, which is the function approximation model that was used for all examined exploration strategies, is an artificial neural network (neural network). The basics of neural networks will be briefly introduced in the section bellow.

2.2.1 Neural networks

Neural networks (NN) are a class of machine learning models loosely inspired by biological neural systems which are used for classification, clustering, function approximation and other tasks. The NN model can be seen as an interconnected system of neurons. The typical architecture of neural networks is divided into layers of neurons where information is passed from one layer to the next. In this model of NN called fully-connected the basic computational unit of neural networks – the neuron – receives input from all neurons in the previous layers. This input is multiplied by weights that belong to each connection between two neurons and summed up into inner activation of the neuron. The output of the neuron is then obtained by passing its inner activation through a non-linear transfer function called activation function. The weights between neurons are updated based on the error given the provided training data using the chain-rule of backpropagation (Rumelhart et al., 1985).

The simple model of neural network consists of three layers of neurons, the input, hidden and output layer. Neighboring layers are connected through the weights (weighted edges) and they form a fully connected directed graph. Output of a network is received by passing the input through the network in the direction of the edges. The update of the weights is achieved by passing the error of the generated prediction in the opposite direction.

2.2.2 Deep neural networks

Deep neural networks are a specific version of neural networks, where many layers of neural networks are stacked on top of each other. In the recent years neural networks became the leading algorithms for most of machine learning tasks, mainly in computer vision. But also in other fields such as reinforcement learning the recent breakthroughs were also enabled thanks to the use of deep neural networks. The success of the deep networks lies in the ability to efficiently train large number of parameters and therefore to be able to approximate high complexity functions. In general the deep networks can learn better representations of the input data by cascading many layers of neurons with nonlinear activation functions. Subsequently we can build more sophisticated algorithms on top of these representations. Usually networks with more than one hidden layer are called deep networks, but the really deep networks used for instance in the computer vision can have over hundred of layers.

Chapter 3

Exploration in reinforcement learning

The problem of exploration in reinforcement learning is a growing research area as we witness an increase of interest in the RL problems within the computer science field. Exploration strategies are algorithms which decide when to follow the currently best known path or when to try a new one. The exploration of new paths can potentially discover better policies, however, the result is uncertain and too much exploration is costly. Also if the agent keeps exploring too much then its model of the environment might never converge and thus the agent might never learn the optimal strategy.

The general goal of exploration strategies in reinforcement learning is to minimize time and cost of learning. According to (Thrun, 1992) the problem of exploration can be summarized in these four questions: How can learning time be minimized? How can costs be minimized? What impact does the exploration strategy have on the speed and cost of learning? How does one trade off exploration and exploitation?

First an overview of the most fundamental tabular approaches will be given in Section 3.1. Then in Section 3.2 will be described how these approaches have to be adapted to high dimensional state-spaces. Finally, approaches selected for the experimental evaluation in the practical part of the thesis will be reviewed in more detail in the Section 3.3.

3.1 Tabular cases

In the field of traditional tabular reinforcement learning there are many good solutions to exploration problems and several algorithms (for instance (Jaksch et al., 2010)) have provided near optimal solutions guarantee. The basic categories of exploration strategies in tabular RL are the undirected and directed methods. The following methods are taken from (Thrun, 1992) and (Sutton and Barto, 1998).

3.1.1 Undirected methods

Undirected exploration strategies are those methods which aren't based on information about the learning process, the agent uses only the actual state-action utility values. The undirected methods are employing a probability distribution which is used for setting the frequency of selection of random actions.

ϵ-greedy

One of the examples of undirected exploration is the ϵ-greedy strategy which lets the agent take a random action with an ϵ probability. In the other cases the agent selects the best (value maximizing) action.

In order to achieve convergence a learning policy called Greedy in the Limit with Infinite Exploration (GLIE) (Singh et al., 2000) has to be used. A learning policy is GLIE if it satisfies two properties:

1. If a state is selected infinitely often than each of its actions must be also chosen infinitely often,
2. in the limit $t \to \infty$ the learning policy must become greedy.

Boltzmann selection

Another undirected strategy is the Boltzmann selection. It involves probability for action selections but it also takes into account the utility values of the state-actions. The probability of taking an action is proportional to its corresponding utility – the higher the value the higher the probability. The exploration rate is affected by the temperature T, increasing the value of T increases also exploration rate.

At a state s, an action a from the set of available actions A is selected given probability:

$$p(s,a) = \frac{e^{U(s,a)\cdot T^{-1}}}{\sum\limits_{a' \in A} e^{U(s,a')\cdot T^{-1}}}, \tag{3.1}$$

where $U(s,a)$ denotes the current utility (value) of the state-action pair. Boltzmann selection belongs to the undirected exploration methods as it is only keeping the utility values and not any history about the learning process such as visits-count for each state.

3.1.2 Directed methods

The Directed methods are exploration methods based on the progress of the learning process. The information about the learning process is then included in the evaluation of the policies or utility values of the state-action pairs. The distortion of the original values leads subsequently to exploration of different than only maximizing actions.

Count-based exploration

Simple, but efficient strategies are the Count-based exploration methods. The class of strategies is based on counting the number of state-action visits. The visits-count are then used to evaluate all actions in a state. Thus the new utility of an action is obtained as a combination of an exploitation (the original state-action utility) and an exploration term which is a function of the visits-count.

One example of the Count-based exploration methods is the Upper Confidence Bound strategy – the UCB1 (Auer et al., 2002). Using the UCB1 approach the combined utility $\overline{U}(s,a)$ of selecting action a in state s can be obtained as

$$\overline{U}(s,a) = U(s,a) + c \cdot \sqrt{\frac{2 \cdot ln(n(s))}{n(s,a)}}, \tag{3.2}$$

where $U(s,a)$ is the original utility, c is the exploration factor, $n(s)$ is the total count of visits to state s and $n(s,a)$ is the count state-action visits. The algorithm is derived from the Chernoff-Hoeffding Bound (Hoeffding, 1963), (Chernoff, 1952) and achieves regret which grows logarithmically with the number of all explored actions. As the visits-count rises, the agent gradually begins to select only the maximizing because the weight of the exploration term slowly diminishes. The UCB1 is a very popular algorithm in the tabular reinforcement learning also due to its implementation simplicity in the tabular reinforcement learning problems.

Error-based exploration

Another heuristic strategy, the Error-based exploration works on the assumption that if a state's utility has been recently extensively changed than it makes sense to prefer to visit those states as the change in utility might be an indication of even more changes of utility in the future.

Recency-based exploration

Opposite to the previous method the Recency-Based Exploration is a method where the state which hasn't been seen recently will be more preferred which is similar to the idea of Count-based exploration. In the Recency-based exploration an additional value is added to the state (state-action) utility value which incorporates how recently a state has been visited combined with the total frequency of visits to the state. Thus if a state was visited only a few times, but has been observed just recently the probability of the state being selected again will be low.

Optimism in the face of uncertainty

A large class of heuristic approaches are methods called the Optimism in the face of uncertainty. The exploration methods are using the distributions instead of the single scalar values for each state-action. The idea of these methods is that the more uncertain the agent is about an action the more important it is to explore it, because it still has the potential to be the best action.

The UCB1 algorithm (from Section 3.1.2) can also be viewed as a method which utilizes the Optimism in the face of uncertainty, as the count exploration term describes the uncertainty about the value estimate.

Another example of the approach is the Optimistic initialization method where all state-action values are initialized with values that are higher than their true values. Thus until the state-action values are corrected by the experience the less explored actions will get selected more often than the more explored actions.

Thompson sampling

Thompson sampling is another action selection method for reinforcement learning. The agent is choosing an action which maximizes expected reward of randomly drawn samples from the distributions of mean values for each state-action pair.

At the beginning of the run the distribution is created from a few randomly drawn samples, thus the initial distribution might not correspond to the real mean distribution. In each step agent evaluates all available actions by sampling expected (mean) rewards from the current distributions and selects the maximizing action given those samples. In case of not fully converged distributions (distributions with large variance) the sampled rewards might be more optimistic (or pessimistic) than the current mean rewards and thus other than currently best action could be selected. Therefore the agent explores also different actions. The basic idea of the approach is that flat distribution of the mean rewards means that the action was not sufficiently explored and thus the agent is not fully certain about the outcome. As the distributions become more converged exploration is lowered and the agent is more exploiting its knowledge.

PAC–MDP efficient approaches

For the tabular case reinforcement learning we can even obtain the PAC–MDP (Probably approximately correct MDP) efficient exploration strategies. The PAC-MDP learning is such a learning where the agent will learn with high probability a selected function which will have low generalization error (this stands for the "approximately correct"). The agent must be able to learn the given problem with any possible probability and approximation. Examples of these methods are the E3 (MIT) and R-MAX (Brafman and Tennenholtz, 2002) algorithms.

For very simple problems the optimal exploration strategy can be obtained by means of the Bayesian reinforcement learning. The Bayesian RL maintains distributions (beliefs) over e.g. transition or value-functions, policy, and reward functions. Thus the model can deal with uncertainties. The exploration-exploitation dilemma is solved by planning in the belief space. Even though the approach is theoretically optimal, it is not applicable in practice as it is in general computationally intractable.

In general the exploration in reinforcement learning problems with tabular representation is a very well studied problem with many successful strategies. In the function approximation reinforcement learning most of the tabular exploration methods cannot be directly applied. Thus either the strategies have to be adjusted or new have to be developed. The exploration in these domains will be described in the following section.

3.2 Function approximation cases

In the case of high-dimensional state space most of the solutions for the traditional tabular reinforcement learning problems cannot be directly applied and therefore different methods have to be used. For better orientation we can apply the same categorization of directed and undirected in the same manner as in the tabular cases.

3.2.1 Undirected methods

Similarly to exploration in tabular reinforcement learning cases the ϵ-greedy is the basic strategy for dealing with the problem of exploration in the function approximation reinforcement learning problems.

Other methods also employ the idea of action randomization, however, unlike the standard ϵ-greedy strategy these strategies don't apply the randomness to the actions, but noise is added directly to the model parameters. The disturbance of the parameters leads to more random action selection.

One of those approaches is the strategy Noisy Networks for Exploration by (Fortunato et al., 2017), the strategy is adding noise to the weights of the model which estimates the Q-values. The intensity (parameters) of the noise is learned by gradient descent same as other parameters of the network. Thus the noise is also induced during the network update and its intensity changes during training. The authors showed that parameter-based noise can drive efficient exploration.

The authors of Parameter Space Noise for Exploration by (Plappert et al., 2017) apply the same idea of adding noise directly to the weights. However unlike the Noisy Networks approach the intensity of the noise is not trained by gradient descent but it is adjusted according to the variance in the actions so that the variance in selected actions is kept same as in ϵ-greedy exploration. This work has been evaluated in the practical part of the thesis and therefore will be further introduced in Section 3.3.1.

Another undirected method is the Deep Exploration via Bootstrapped DQN (Bootstrapped DQN) by (Osband et al., 2016) which uses the randomized value function approximations for exploration. The randomized value estimates are obtained from keeping more separate models which are running and learning in parallel. Each one of the networks is trained against its own target network and on its own subsample of the data. In each episode the agent is following the policy given by only one of the models which is selected randomly. Thus the action selection is not always the maximizing action as each model is slightly different.

3.2.2 Directed methods

Most of the directed methods for the function approximation methods take inspiration in the exploration strategies from the tabular case reinforcement learning.

State visits-count

One group of approaches inspired by tabular RL is based on the idea of state visits-count. The state visits-count cannot be applied directly because the keeping of an actual state-visit count table is infeasible for the continuous state space problem, thus methods based on count approximation have to be developed.

The authors of Unifying Count-Based Exploration and Intrinsic Motivation by (Bellemare et al., 2016) calculate the state visits-count by using a density model which can be used to measure uncertainty on the given input. The uncertainty is then transformed into pseudo-count of visits. The density model is a successful reconstruction model the PixelCNN (van den Oord et al., 2016) which can also be used to calculate the probability of the given input given by the state of the model. The PixelCNN network is trained online, it is updated after each step of the algorithm, thus the model is estimating the actual current pseudo-count of the provided state-action pair.

Another implementation of the state visits-count is the Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning by (Tang et al., 2017). In this work the problem of high-dimensionality of the state space is solved by mapping the input states to hash codes using autoencoders. The hash codes are then used as indexes to hash tables where the occurrences of the given state is recorded. The strategy by (Tang et al., 2017) has been evaluated in this thesis, thus it will be further introduced in Section 3.3.2.

Model uncertainty

Another option for exploration is to use uncertainty of a model for the selection of interesting states to explore. In (Stadie et al., 2015) the authors used the error of encoding prediction of the state s_{t+1} given the encoding of the state s_t and action a as a measure of how well the given state-action pair is learned. The error is then added to the reward function which forces the agent prefer unknown state-action pairs.

The model uncertainty for exploration cal also be used to guide the exploration rate. The more uncertain the model is the more noisy its output is and thus the model explores also non maximizing actions. Examples of this approach are the next two methods – the Weight Uncertainty in Neural Networks by (Blundell et al., 2015) and the Bayesian Approximation: Representing Model Uncertainty in Deep Learning by (Gal and Ghahramani, 2015).

The authors of Weight Uncertainty in Neural Networks introduced an algorithm for modelling probability distribution on the weights (Bayes by Backprop) which induces regularization on the network similar to Dropout method in neural networks. The weight uncertainty can be subsequently used for weighting exploration-exploitation trade-off in reinforcement learning. Another way to estimate model uncertainty is by using Dropout as presented in Bayesian Approximation: Representing Model Uncertainty in Deep Learning. The Dropout in neural networks is approximating Bayesian inference in deep Gaussian processes and subsequently it can be used to estimate the model's uncertainty. The Dropout method was implemented in this thesis, thus further explanation of the method can be found in Section 3.3.3.

Curiosity based exploration

In the work by (Houthooft et al., 2016) the exploration strategy called VIME (Variational Inference Maximizing Exploration) is based on maximization of information gain about the agent's belief of environment dynamics. A similar approach is the curiosity-driven exploration by (Pathak et al., 2017), where the curiosity is formulated as the error in an agent's ability to predict the consequence of its own actions. This method was only applied to Policy gradient based reinforcement learning, as the error is only known after the action has been selected and the next state has been seen. Thus the error is used to update the current policy of the agent.

3.3 Studied algorithms

In the following section the selected and in the thesis compared approaches to the exploration in function approximation reinforcement learning problems will be more broadly introduced and their theoretical foundations will be explained.

3.3.1 Parameter noise exploration

The strategy by (Plappert et al., 2017) induces exploration by adding noise to the parameters of the model. The method was selected for comparison based on the published results where the Parameter space noise surpassed the results of the ϵ-greedy strategy despite the simplicity of the proposed strategy in many games from the Arcade learning environment ALE (Bellemare et al., 2013b).

The strategy – using random noise to induce exploration – is very similar to the basic ϵ-greedy method. However, the authors of this approach explain that even though the method is undirected, the non maximizing actions are unlike in the ϵ-greedy method less random and more corresponding to the real values of the actions. This is caused by the application of the noise directly to the parameters of the model in contrast to the application of noise to the actions as in the ϵ-greedy strategy.

The intensity of the noise added to each of the parameters of the network is drawn from the Gaussian distribution $(0, \sigma)$. The value of σ is updated based on distance between non-perturbed and perturbed policy. Different distance measures have been selected for each of the examined methods (DQN, DDPG, TRPO) in their work.

For the DQN version of the method a distance measure based on Kullback-Leibler (KL) divergence has been proposed. In order to be able to compare the outputs from a perturbed and non-perturbed model a probabilistic formulation of the policies has to be obtained by application of softmax function over the predicted Q values has been applied. Given two policies $\pi, \overline{\pi} : S \times A \to [0, 1]$ we can measure their distance in action space through:

$$d(\pi, \pi) = D_{KL}(\pi||\overline{\pi}), \tag{3.3}$$

where D_{KL} is the Kullback-Leibler divergence. The distance measure is then used to update the value of σ every K steps using:

$$\sigma_{k+1} = \begin{cases} 1.01 \cdot \sigma_k, & \text{if } d(\pi, \overline{\pi}) < \delta \\ \frac{1}{1.01} \cdot \sigma_k, & \text{otherwise,} \end{cases} \tag{3.4}$$

where δ denotes a threshold value. The choice of δ in the DQN version of the method is derived from the distance measure between a greedy and ϵ-greedy policy, and the threshold is set to $\delta = -\log(1 - \epsilon + \frac{\epsilon}{|A|})$.

By connecting the threshold value to ϵ-greedy strategy the amount of exploration is the same as in a given ϵ-greedy strategy.

In the work by (Plappert et al., 2017) the noise level is kept the same in all layers. This should be not feasible due to the fact that in the deep neural networks each layer could have different weight values, however, the authors of the work showed, that through applying Layer normalization (Ba et al., 2016) the same perturbation scale can be used in all layers without caring for the different sensitivity to noise in different layers.

The model used in the Parameter Space Noise for Exploration is the DQN proposed in (Mnih et al., 2016). Layer normalization (Ba et al., 2016) is applied on each neuron in the fully connected part of the network before the nonlinearity. Differently to the original work by (Mnih et al., 2016) the output of the network has two heads, one for prediction of the direct Q values which are used for action selection and second with softmax applied to the Q values which is used for the noise update.

3.3.2 Visits-count exploration

The next examined strategy was a directed method based on state visits-count. The state visits-count is a widely successful strategy in the table state spaces and thus its implementation in the more practical high-dimensional state spaces is attractive. The basic idea behind the strategy is quite simple in comparison to other to high-dimensional state spaces applicable approaches. However, the application of the visits-count strategy to the high-dimensional state spaces is not straight-forward, but several approaches have shown that after some modifications the idea is still applicable. One of the implementation of visits-count is the work by (Tang et al., 2017). This exact implementation was selected based on its applicability of the approach to non-image inputs. The other (more famous) method of (Bellemare et al., 2016) based its pseudo-count on a density model PixelCNN (van den Oord et al., 2016) which enabled them to use the model's uncertainty about the current input frame (agent's input from the environment) as a proxy for visits-count. The more often the model has seen a particular input the less uncertain it was. As the input vector of the researched environment was not image-based (the features of the vector should be independent) this approach was not applicable.

The method by (Tang et al., 2017) is based on different approach to the counts. The authors also use a reconstruction model (deep learning model called Autoencoder), however, they use it for dimensionality reduction of the high-dimensional input. The reduction is made so that a count table of the visits can be maintained as it is impossible to keep a table directly for the high-dimensional input, since the table would be too large and also most states would occur only once.

Autoencoder learns the feature representation of the input by being forced to compress it into a lower dimension and then reconstruct it back. An Autoencoder has two main parts – one is reducing the input dimensionality towards a bottleneck and second is enlarging from the bottleneck to former size of the input and the model is updated using the error of the reconstruction. In the implementation of (Tang et al., 2017) the obtained bottleneck vector is used for calculating of state visits-count. The obtained vector is first binarized and then further hashed using the Simhash (Charikar, 2002) algorithm. In order to improve the model's ability of distinguishing between different inputs the authors introduce two changes to the model. First a noise is introduced to the reconstruction part of the Autoencoder after the bottleneck and second the loss function of the Autoencoder is extended with a binarization error of the bottleneck. The resulting loss function for input s_n is then defined as

$$L(s_n) = -[log(p(s_n)) - \frac{\lambda}{K} \sum_{i=1,..D} min((1 - b_i(s_n))^2, b_i(s_n)^2)], \qquad (3.5)$$

where $log(p(s_n))$ is cross-entropy term and the rest is a term which forces the bottleneck vector to take on binary values. The second term is scaled by a balancing constant λ which is divided by the size K of the Simhash matrix which determines the final size of the hash.

The input states are mapped to the hash codes which enable to count the occurrence of the input states using a hash table. Using these counts a reward bonus is computed given the theory of the count-based exploration methods as described in Section 3.1.2. The reward bonus is then added to the obtained rewards and given the new rewards the maximizing action is selected. The new reward $\overline{R}(s_m, a_n)$ is calculated as

$$\overline{R}(s_m, a_n) = R(s_m, a_n) + \frac{\beta}{\sqrt{n(\phi(s_m, a_n)}}, \tag{3.6}$$

where $R(s_m, a_n)$ is the original reward given action a_n was selected in state s_m, β is an exploration constant, and $n(\phi(s_m, a_n))$ is the state-action visits-count. As all actions are gradually explored the count reward bonus is diminishing and the agent becomes more greedy (exploiting).

In (Tang et al., 2017) a Policy gradient based method called Trust Region Policy Optimization (TRPO) (Schulman et al., 2015) is used. The newly obtained rewards are thus used for policy update and not as a direct exploration bonus during the decision making of the agent, however, principally it is the same as if the bonus was added to the Q estimates in the DQN model.

3.3.3 Dropout

The Dropout is an exploration strategy which uses the model uncertainty to drive exploration. The uncertainty can help the agent decide when it should exploit and when it should explore the environment. In the function approximation reinforcement learning a neural network is used to approximate the values of possible actions. Given the uncertainty measure and using methods such as Thompson sampling the agent will select the truly best action in states which are well known and in states that it is uncertain about it might select a different action.

Dropout is originally a regularization method proposed in (Srivastava et al., 2014) for application in neural networks to achieve better generalization (overcome overfitting) of large fully connected neural networks. During training a neural network with Dropout a random subset of activations in each layer is set to zero. Each unit's output is dropped with a probability p and kept with probability 1-p. This forces the model to be robust so that a change in few units' outputs should not influence the final output. The output of a layer l_k can in a Dropout network be calculated by:

$$l_k = m \cdot a(W \cdot l_{k-1}), \tag{3.7}$$

where W is a weight matrix, l_{k-1} activation of previous layer, a activation function and finally m is a random binary vector with probability of zero being p.

Training of neural networks with Dropout is similar to training of a standard neural network with the difference that the forward pass and backpropagation are only done on the thinned (non-dropped part) of the network.

The idea behind application of Dropout as an exploration strategy is similar to the Bootstrapped DQN (Osband et al.) model. The dropping of different parts of the network forces the model to create more submodels, during the exploitation phase whole network is used for the prediction which corresponds to the ensembling idea from Bootstrapped DQN. The idea of ensembling is that each of the created submodels have some small inherent error, however, by combining all of them we can obtain better results than only if one model was trained. Another view on Dropout is that it also induces some noise in the Q-value predictions, and before the network learns how to deal with this noise the Q-values estimates might be noisy and thus the agent will select variant actions.

The Dropout method was selected based on the work by (Gal and Ghahramani, 2015). In their work the authors showed that Dropout can be used as an approximation of Bayesian inference in deep Gaussian processes which can be used to measure prediction uncertainty of the model. Other works address the problem of model uncertainty as well, however, the benefit of using the Dropout method is that it is much simpler than other uncertainty modeling method.

In the original work the applicability of the idea was shown in several settings (mainly classification tasks). One of the settings was a simple 2D Reinforcement learning environment. The simple RL task was used to show that Dropout can drive exploration in function approximation cases of RL and the convergence on the task was with the Dropout method faster than with the traditional ϵ-greedy strategy.

The underlying model in the work by (Gal and Ghahramani, 2015) was a 2-layer network. The simple 2D environment consisted of an agent with 9 sensors (inputs) and 5 available actions. The task of the agent was to navigate through a maze towards a predefined goal. In this simple settings the proposed method based on Dropout with probability of 0.1 and Thompson sampling surpassed the original ϵ-greedy baseline.

3.3.4 Proposal – Uncertainty driven exploration

We propose a novel exploration method based on the idea that the optimal exploration strategy should balance the exploration vs. exploitation, which is generally an unsolved problem. But one way how to make a progress in this direction is to design such exploration strategy which would be able to select those actions which are promising in terms of expected reward, but have not yet been explored much.

Our proposed solution to this problem builds on these two main assumptions:

- Agent's uncertainty about the values of it's actions can be modeled
- A measure of this uncertainty can be used as a positive bias for selecting actions with beneficial exploration

For satisfying the first assumption we propose using the Distributional Perspective approach (Bellemare et al., 2017). The fundamental idea of the approach in Distributional Perspective is that instead of using the mean of the rewards the authors propose to model their variation. They demonstrate that it results in improved accuracy and performance. The authors stress the importance of keeping value distributions instead of keeping only the expected values on the example of very randomized rewards. If the outcomes can be either very good or very bad then the distributions can better capture this behavior and thus risk aware algorithms can utilize this information. The variation of the random rewards is modelled using value distributions. The authors prove a variant of the original Bellman's equation which enables them to train the model without aggregating the Q-values into averages. The value distribution is modelled as a histogram over possible values with probabilities assigned to each of the value. The authors propose and implement new algorithm for updating the value distributions instead of the mean values.

In the original work, the exploration of the agent is driven by the ϵ-greedy strategy. First all actions are evaluated and the Q-value estimates for comparison are obtained by taking the sum over the range of Q-values weighted by the probabilities provided by the distribution. This estimate is then used to select the maximizing action and random action is selected with probability of ϵ.

We propose to replace this randomized exploration strategy by employing the Optimism in the face of uncertainty paradigm described in Section 3.1.2, which shows that overestimating actions in a certain way can lead towards beneficial exploration.

By combining these two approaches together we obtain a new exploration strategy which is novel in the way how actions are overestimated: the original Q-value is increased by a bonus calculated as the entropy of the state-action value distribution:

$$\overline{Q}(s,a) = Q(s,a) + c \cdot H(P(s,a)), \tag{3.8}$$

where $Q(s,a)$ is the original Q-value, $P(s,a)$ is the value distribution over state s and action a, H its entropy, and c is the exploration constant similarly as in the Visits-count exploration 3.3.2.

In certain domains the proposed strategy could lead to over-exploration as an intrinsic noisiness of the environment would be manifest itself in higher entropy of actions. Thus with wrongly set exploration constant a sub-optimal action would still be selected even if the value distributions have converged to its real shape.

Optimally the selected domains should be deterministic as in those domain the shape of the value distribution corresponds to the uncertainty about the Q estimate, however, with properly select exploration constant the strategy could be applied in noisy domains too. In theory the exploration constant which would prevent over-exploration with converged Q estimates could be set as $c < min_{i,j}(|Q_i - Q_j|)$, however, in practice we will know the optimal Q ranges only very rarely.

Chapter 4

Environments

In this chapter two environments for evaluation of the exploration strategies are introduced. The primary environment is the Classification with costly features (CCF). This task will be introduced in Section 4.1. The secondary evaluation environment is the Cart-pole balancing problem, the results of the evaluation are presented in Section 5.3.

4.1 Classification with Costly Features

Classification with Costly Features (CCF) (Tan, 1993) is a task where an agent has to classify input vectors where the obtaining of each feature has a certain cost. Wrong and correct classifications are also awarded a certain cost. Thus the agent is forced to select the lowest number of features possible so that the reward is maximized.

This task can be viewed as a classic reinforcement learning problem where an agent explores the environment through its actions and observes rewards and new states. The state of the agent is encoded by the observed features – observed features in the sample. The initial state x_0 of the agent has no revealed features thus all values in the feature are set as $x_0(i) = 0, \text{ for } i = 1, 2, ..|x_0|$, the 0 values in the vector represent that all features are unknown. Given the observed state x_t the agent at each timestep t either classifies the feature vector or decides to view another feature. One episode corresponds to the classification of one sample (feature vector) from the given dataset.

The costs of correct and incorrect classifications of a given sample x are constants. The selection of another action (obtaining the value of $x(i)$) is praised with a negative reward of $-\lambda \cdot c(f(i))$. Due to the nature of the task the agent can only select an action $a(i)$ (feature) which has not yet been selected. Input vector to the network is combined with a binary mask. The binary mask contains 1 at the indexes of features that are already known to the agent, thus the agent can differentiate between unknown features and features with 0 value.

The constant λ is a feature cost factor, by changing its value the trade-off is made between precision and average cost. If the λ is higher then the agent will select fewer actions over lower precision and vice versa. The goal of an agent is to find a policy π that determines the sequence of actions (features) and classification of the input sample so as to maximize the expected reward.

The datasets used for evaluation were MiniBooNE and Forest CoverType (Bache and Lichman, 2013) as in (Janisch et al., 2017). Both datasets were split into training, validation and testing sets, thus all methods are compared on the same parts of the datasets. The evaluated datasets don't have assigned explicit costs to their feature therefore uniform cost of 1.0 (multiplied by the feature factor λ) was used for all actions. Each feature in the dataset is normalized by its mean and standard deviation, if the feature is yet unknown a value of 0 is used in place of its real value. The MiniBooNE dataset contains 50 real valued features and 2 classification classes. On the contrary the Forest dataset contains 54 features out of which only 10 have real values and the rest are binary encodings for 2 features, thus the input vector is very sparse, the Forest CoverType dataset classifies into 7 classes. Both datasets create non-deterministic environments as taking an action in a state can lead to different states given different samples.

4.2 OpenAI Cart-pole

The Cart-pole is the task of pole balancing on a cart. The goal is to keep the pole in the vertical position by changing the directions of movement of the cart.

We used the implementation of the task available from the OpenAI gym framework (Brockman et al., 2016) which is corresponding to the Cart-pole problem definition in (Barto et al., 1983). The agent receives reward for each step, when the pole is above a certain threshold. The threshold is defined as the distance and the angle to the starting position. If the pole falls under the threshold the episode ends and the state is reset. The episode also ends after 200 steps. If the agent manages to keep the pole upright for the defined 200 steps, then the problem has been solved.

The Cart-pole domain is deterministic, equal state-action pairs always lead towards the same next state and reward regardless the time at which it was selected. Thus without exploration the agent can never improve its initial strategy.

The input observed by the agent is a real valued vector of size 4. The state encodes the agent's position, speed and direction of movement, and the angle of the pole. Given by the implementation by (Brockman et al., 2016) the initial state is selected randomly from predefined starting intervals. After each reset the agent faces a slightly different situation and thus it can not overfit to a single starting position.

In each state the agent has 2 available actions – to move left or right. In the original implementation a reward of $r = 1$ is awarded for each step. We changed the original setting to a reward of $r = 0.1$ to enable faster experimenting, as the speed of convergence with the original setting was slower due to the utilization of gradient clipping. Gradient clipping was used in all models in order to prevent divergence caused by gradient explosion.

Chapter 5

Experimental evaluation

In this chapter we present the evaluation results of the examined and the proposed methods on both domains. The model and its hyper-parameters used in the experiments will be described in Section 5.1. Description of the experimental setup, implementation details of all the studied exploration strategies, as well as comparison of their performance on the CCF task and on the Cart-pole can be found in Section 5.2 and Section 5.3.

For a better comprehensiveness we don't show all results on the MiniBooNE and Forest datasets as the results are very similar. The omitted results can be found in Appendix A.

5.1 Experimental settings

In this section the details about the experimental settings and model parameters for both evaluated domains will be described.

Model for CCF domain

The underlying model for all studied exploration strategies is a Deep Q-Network (DQN) based on the work by (Janisch et al., 2017) Classification with Costly Features using Deep Reinforcement Learning. All of the training settings are set equally to (Janisch et al., 2017) if not stated otherwise. Similarly as in (Janisch et al., 2017) the DQN baseline model is enriched with several improving methods – Experience replay, Double Deep Q-learning and Soft-target network. For explanation of the methods see Section 2.1.2. The underlying network consists of two fully-connected layers of 128 hidden units each, each hidden unit has ReLU (Nair and Hinton, 2010) activation function, in the output layer no activation function is used.

For performance reasons the simulation was run with multiple agents in parallel. In the CCF task 1000 agents are run simultaneously which generates 1000 samples at once. Sample is a tuple (s, a, r, s') where s is a given state, a is the selected action, r

is obtained reward and s' is observed new state. At each step the model is updated using samples randomly selected from the replay memory buffer of 1000000 samples. The memory buffer is at the beginning of the training filled with samples given by the initial network and exploration strategy. For the Q-value updates a large mini-batchsize of 10000 samples is used. The large mini-batchsize is enabled by the small size of the input vectors and results into shorter training time which is needed as we ran over 1000 experiments. For the training of the neural network we use the optimization method RMSProp (Tieleman and Hinton, 2012) with momentum of 0.95. Learning rate is initially set to 10^{-4} and then exponentially decayed over time. We apply gradient clipping of gradients larger than 1.

The discount factor γ is set in the CCF to 1. We can use such an uncommon value of discount factor, as each episode has predefined (short – max 54) maximal length and thus we can take the whole episode into account for the Q-learning update. Feature factor (cost of each step) λ was set to 0.01. The value was selected based on the experiments with the value of λ in (Janisch et al., 2017). For the $\lambda = 0.01$ the convergence of the number of selected features was visible but not too fast thus the different exploration methods could be well compared.

Three different settings of the rewards for the correct/incorrect classifications were compared. The original values from (Janisch et al., 2017) were set to 0 for correct and -1 for wrong prediction. However due to the Xavier (Glorot and Bengio, 2010) initialization of the model parameters to values close to 0, the starting Q-values are also around 0 which lead to an exploration method called Optimistic initialization as described in Section 5.2.1. Thus two more settings had to be evaluated. One setting with rewards of 1 and 0 for correct and wrong prediction which lead to Pessimistic initialization and second setting with values of 0.28 and -0.72 for correct and wrong prediction lead to Initialization by the priors of classes.

Model for Cart-pole domain

For the Cart-pole domain we simplified the original model from the work by (Janisch et al., 2017) as our input and action dimensionality was smaller than in their task. The number of hidden layers was reduced from 2 to 1 fully-connected layer of 64 units.

Similarly to CCF domain 100 agents are simulated in parallel. The model is updated after each step using a mini-batchsize of 32 samples. The samples are selected from a replay memory buffer of 10000 samples. The discount factor γ is set to 0.99. The size of mini-batchsize, memory buffer and γ are set equally as in the original work (Mnih et al., 2015). We applied the same optimization method (RMSProp) with the same setting as in (Janisch et al., 2017). Initial learning rate was set to 5^{-5} due to significantly smaller mini-batchsize. We also apply gradient clipping of values larger than 1.

5.2 Classification with costly features

In this section we will present the results of the implemented exploration strategies. We will first shortly introduce the experimental settings and results for each strategy. For each strategy we will select the best parameters which will be subsequently used for comparison of results of all strategies. In the main text we only present the results from the evaluation of the strategies on the MiniBooNE dataset as the conclusions were same for both datasets. The omitted results can be found in the Appendix A.

Each run was trained for 10000 steps and presented values are averages over 6 runs with different seed initializations for all random generators. All plots show evaluation of only exploitation (with turned off exploration) agents given the learned models. Evaluation ran every 50 steps and the obtained values at each step has been averaged from runs of 1000 agents.

During training we apply no exploration value annealing as it would lead to exponential growth of combinations of parameters' settings and thus proper experimental evaluation would not be possible. For all strategies we first find the best exploration setting. The best settings for all strategies are subsequently compared between each other.

We evaluated 3 basic properties for each strategy:

- the **Average rewards** which is the optimization criterion,
- the **Average length of trails** which correspond to the number of actions that an agent selects prior the classification
- and the **Correct classification ratio** which state the precision of the classification.

For better understanding of the results we evaluated 2 more properties for each strategy – the Resulting exploration and the Resulting entropy of actions. The Resulting exploration shows the ratio of non-maximizing actions selected by the set of 1000 agents during the search of the environment. The entropy shows the randomness in the selected actions of the 1000 agents averaged over 50 steps, thus consequently the entropy shows us the real randomness of the exploration. If the exploration is high, but the resulting entropy is low then the exploration strategy is often selecting non-optimal actions, however, the actions often repeat. For each property we present the mean and standard deviation of the values over 6 runs each with different seed initializations.

As we were primarily interested in the convergence speed and due to the computational demands of the experiments, we decided to keep the length of evaluation restricted. The strategies are compared given two criteria: the speed of convergence and the top results within the restricted training range. To compare the strategies on limited length of training is useful as in practice one has often only limited time and computational resources.

Provided the experiments with initializations as described in Section 5.2.1 we selected the reward setting of 0 and 1 (Pessimistic initialization) for the evaluation of the selected examined exploration strategies. All strategies were evaluated using the same baseline model as described in Section 5.1. Only in the case of the proposed Optimistic agent strategy the output layer is enlarged as each output is multiplied by the number of bins needed for the Q-value distributions.

5.2.1 ϵ-greedy strategy

The ϵ-greedy strategy was selected as the baseline for comparison of all other methods. During exploration of the environment the agent selects with probability ϵ a random action and with probability $1 - \epsilon$ it selects the maximization action. The random action is selected from all available actions and thus also the maximization action can be selected, therefore the resulting real exploration is a little diminished, but as can be seen in Fig. 5.2d it still corresponds to the ϵ setting. Exploration was applied to the search (sample collection) through the environment. During the Q-learning update only the maximization actions are used for all Q estimates. As already stated earlier in this section no annealing of the ϵ value has been applied.

When running our experiments we realized that the different settings of rewards (penalties) influence the meaning of the initialization of Q-values with respect to their final values.

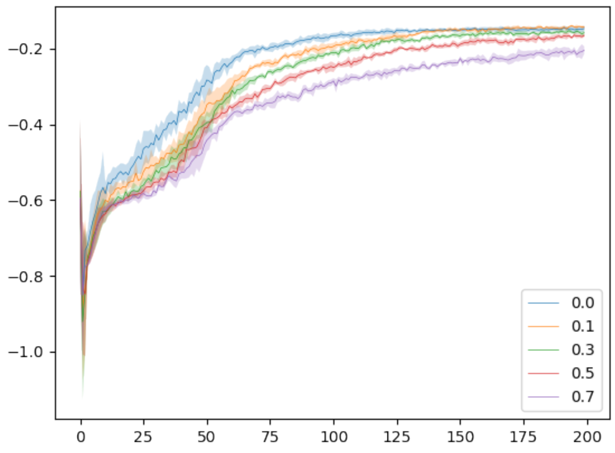
Optimistic initialization

The original settings of rewards – -1 for wrong and 0 for correct classification – from (Janisch et al., 2017) led to surprising behavior of the basic ϵ-greedy exploration. We observed that the strategy with ϵ set to 0 outperformed all other settings. This lead to realization, that this setting of rewards resulted into Optimistic initialization exploration strategy. All actions were initially overvalued and therefore all of them had a large probability of being selected at the beginning. Only after the action values converged close to their real values, the exploration was stopped. In this specific problem, the Optimistic initialization strategy was the most successful and thus we examined also different rewards' settings that could be more challenging and subsequently the different strategies could be better compared. The results can be seen in Fig. 5.1.

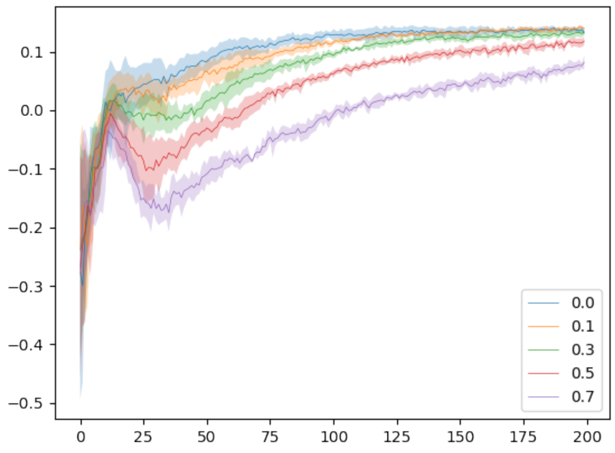
Initialization according to priors

For the MiniBooNE dataset we also compared the settings of rewards which corresponded to initial Q-values being balanced given the prior probabilities of classes. The rewards were set to -0.72 for wrong and 0.28 for correct initialization (priors of the classes were 0.72 and 0.28 respectively). This initialization was already more challenging, however, still the ϵ=0 was the best strategy as quite a lot of perturbance in the actions came directly from the non-determinism of the dataset. The results can be seen in Fig. 5.1.

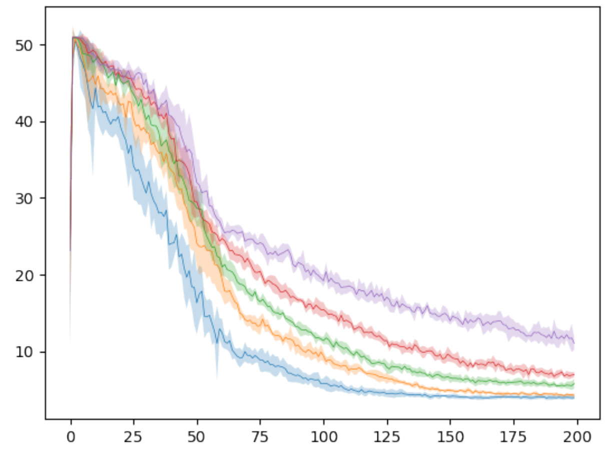
Optimistic init. **Prior init.**



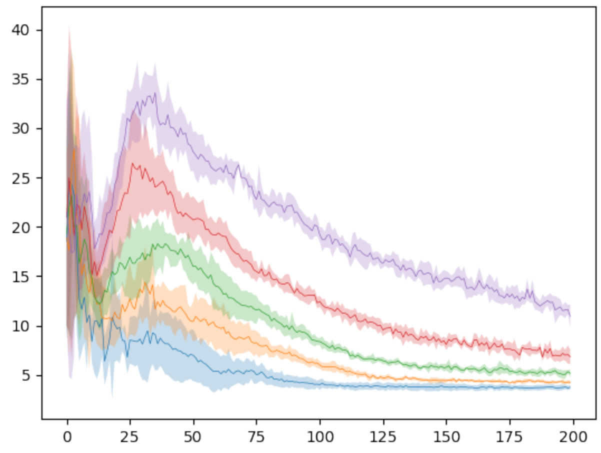
(a) Average rewards



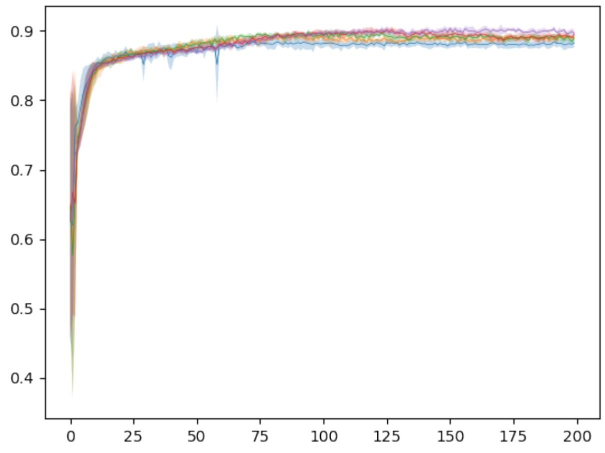
(b) Average rewards



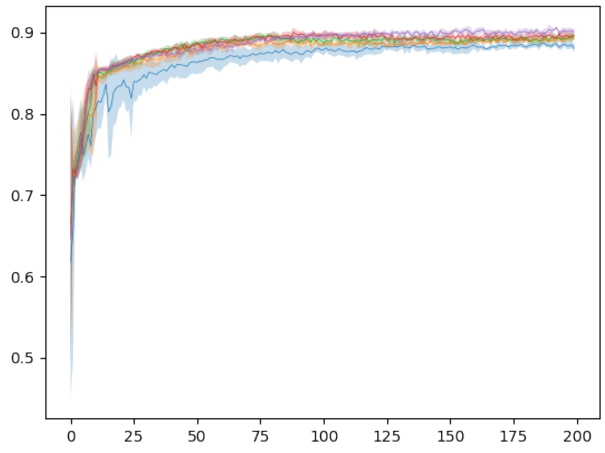
(c) Average length of trails



(d) Average length of trails



(e) Correct classification ratio



(f) Correct classification ratio

Figure 5.1: Results of ϵ-greedy strategy given Optimistic and Prior initialization.

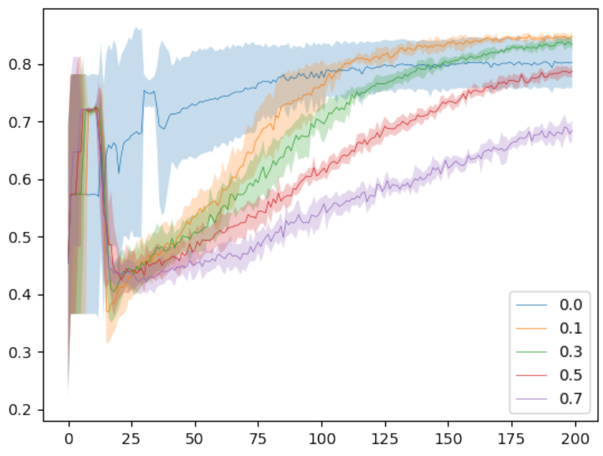
Pessimistic initialization

The Pessimistic initialization was achieved by setting the rewards to 0 for wrong and 1 for correct initialization. For this setting the $\epsilon = 0$ was outperformed by higher ϵ as the noise of the network wasn't enough to search through sufficient number of actions in order to find the optimal sequence of actions which would lead to the best trade-off between precision and number of steps.

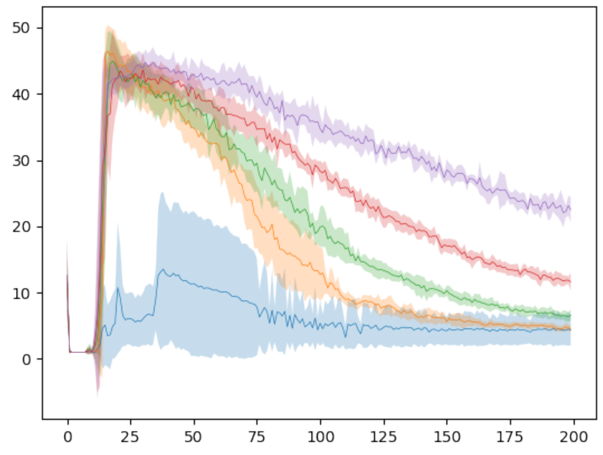
As can be seen in Fig. 5.2 all settings of ϵ are converging, however, with larger values the convergence is significantly slower. As optimistic initialization is not present the $\epsilon = 0$ doesn't find the best optimum, the average rewards remain around prior value of the larger class as the model classifies only into one class. The ϵ-greedy strategy result in small variance within runs, even though each run has different seed for both the walk through the data as well as the selection of random actions. The setting of $\epsilon = 0.1$ had the fastest and best convergence, thus it was selected as the baseline for comparison with other exploration strategies.

Exploration and entropy as can be seen in Fig 5.2d and 5.2e correspond to the setting of ϵ parameter. All settings reach the same level of maximum entropy, however, smaller values allowed better convergence of Q-values and thus the level of entropy stabilized (decreased) over the run of the training.

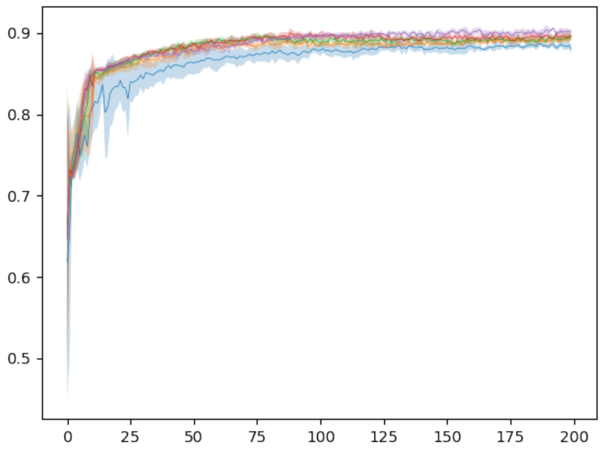
In Fig. 5.2b and Fig. 5.2a can be seen a typical development of the convergence. First the model learns only the priors and thus it utilizes a few actions to classify. During the run the exploration outweighs the prior as taking more actions results in more correct classifications and thus better average reward. During the peak of entropy the agent is utilizing nearly all actions. While examining all actions agent slowly removes uninformative actions, as each step has a predefined cost and shorter and correct classification has better reward. Towards the end of the training the agent utilizes only about 10 actions before the classification.



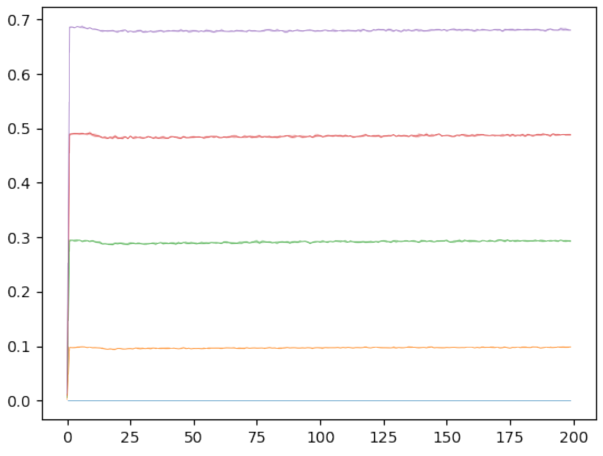
(a) Average rewards



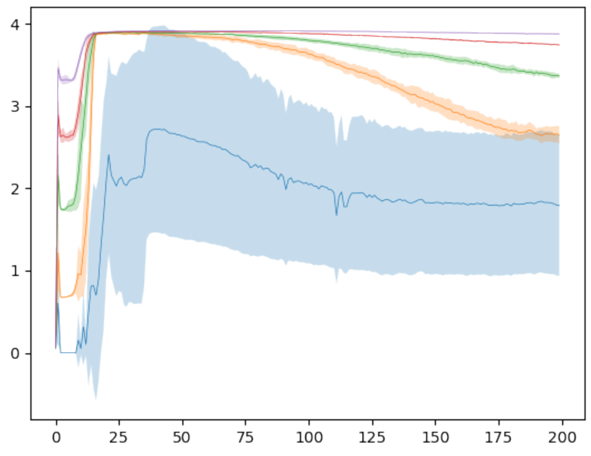
(b) Average length of trails



(c) Correct classification ratio



(d) Resulting exploration



(e) Resulting entropy

Figure 5.2: Results of ϵ-greedy strategy given Pessimistic initialization.

5.2.2 Parameter-noise exploration

The implementation of Parameter-noise was very similar to the original work of (Plappert et al., 2017). However, a few differences resulted from the settings of our DQN model and from the decision not to apply exploration parameter annealing. According to the original work, the noise level in the parameters determines the exploration of the agent.

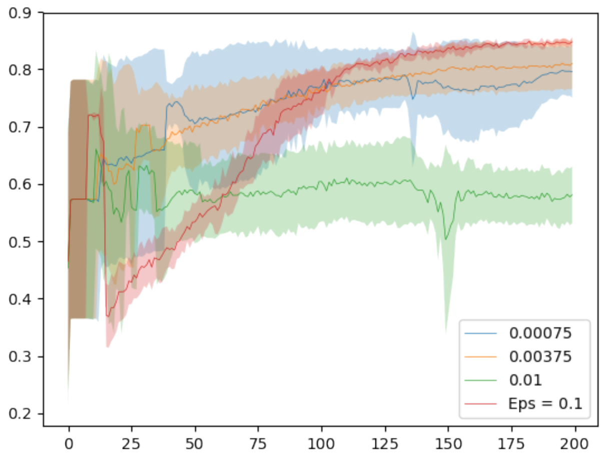
We also didn't implement the KL-divergence update of noise level and we didn't use Layer normalization as our network consisted of only 2 layers, for the same reasons we kept the level of noise the same in both layers.

Noise was added to all weights and biases in the baseline DQN model. For each sample from batch new noise matrix is added to the weights, thus each agent could search through different actions and more experience could be gathered over the same time. Having different noise matrix for each input corresponds to the ϵ-greedy baseline, where each agent can select different action. The added noise was generated from Gaussian distribution $N(0, p)$ where p is the level of noise. Unlike the original implementation we don't keep the noise same during the whole episode of the agent. As we apply Experience replay the update of the network is made from random samples from different episodes and states of the network thus the influence of different noise during one episode is minimal.

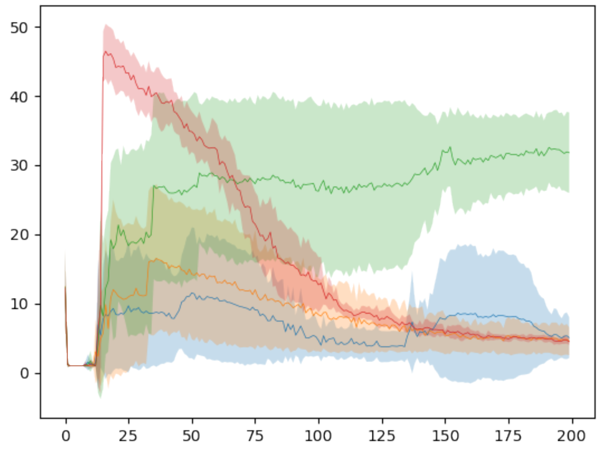
The Parameter-noise exploration as can be seen in Fig. 5.3 resulted into faster convergence towards worse average rewards than the baseline ϵ-greedy strategy. There are several possible explanations. First we can conclude that the non-annealing of the noise hurts the performance of the Parameter-noise exploration more than in the case of ϵ-greedy strategy as the perturbation is absolute and thus it highly depends on the values of weights. Large values of noise bring too much exploration to the end of the training and small values of noise don't bring enough exploration at the beginning of the training and thus the optimal strategy for selection of actions cannot be found. The small values of noise can lead towards poor local optima brought by no exploration during the run. Through detection and removing (as can be seen in Fig. 5.9) of such an outlier we obtain a a fast average convergence towards similar optima as in the ϵ-greedy strategy.

We have also observed that the Parameter-noise strategy result into less entropy than in the ϵ-greedy strategy with larger induced exploration. This naturally leads to worse convergence as it results in less real exploration but often non-optimal action selection. The large value of exploration doesn't allow for the optimal (real) Q-values of state-action pairs to be discovered as they cannot converge to the correct values.

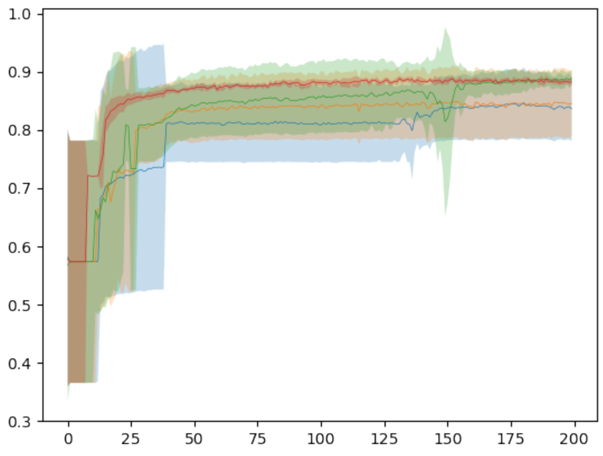
We can also observe that the Parameter-noise strategy brings significantly more variance than the baseline ϵ-greedy approach. We can conclude that adding noise directly to the parameters of the model can sometimes lead to faster but not guaranteed convergence. Given our setting the best value for the noise level has been found to be $p = 0.00375$.



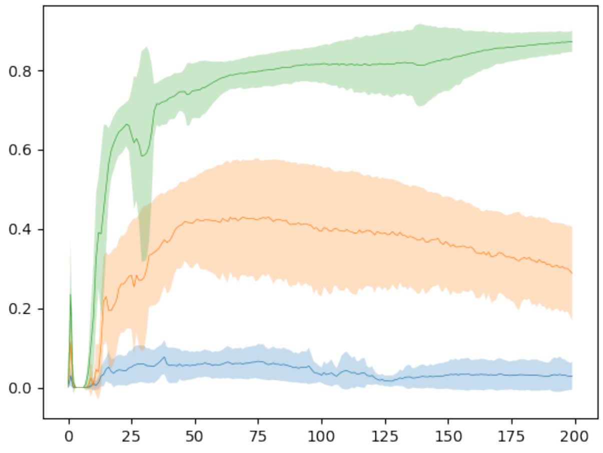
(a) Average rewards



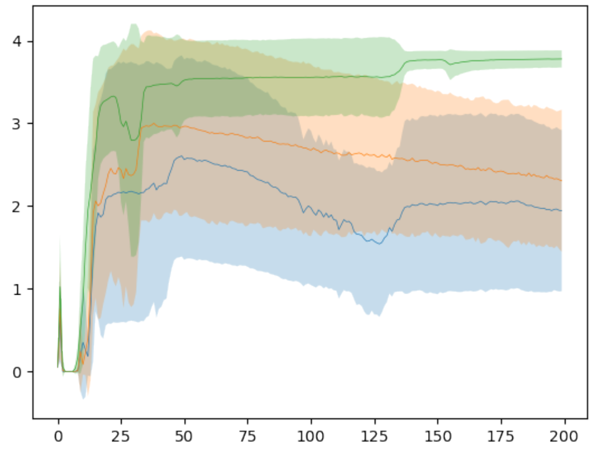
(b) Average length of trails



(c) Correct classification ratio

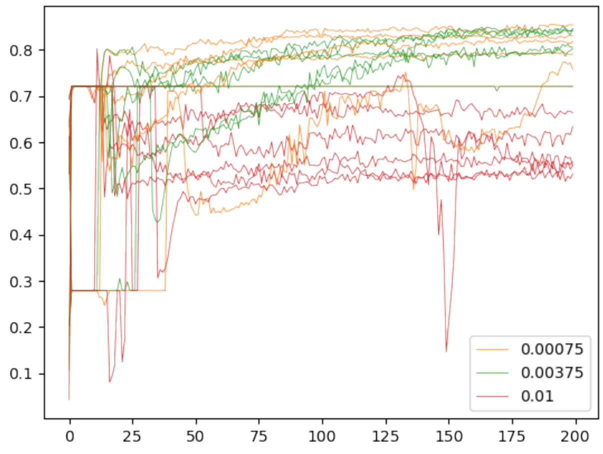


(d) Resulting exploration

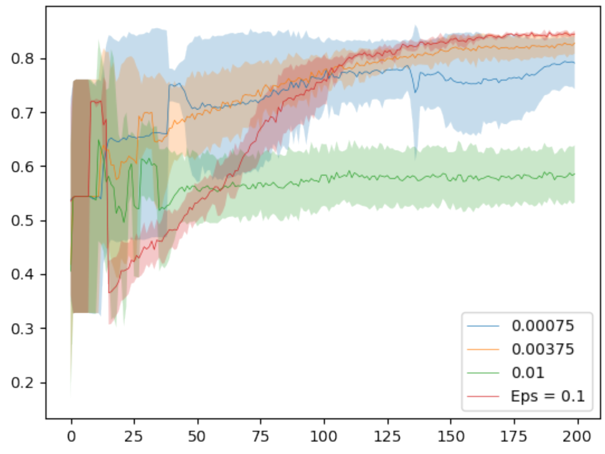


(e) Resulting entropy

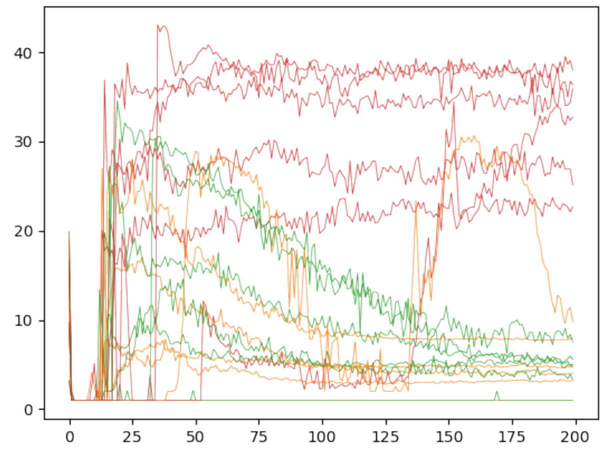
Figure 5.3: Results of Parameter-noise strategy.



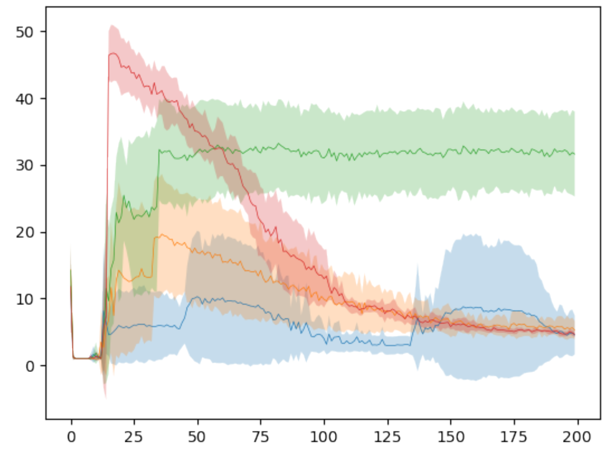
(a) Rewards of separate runs



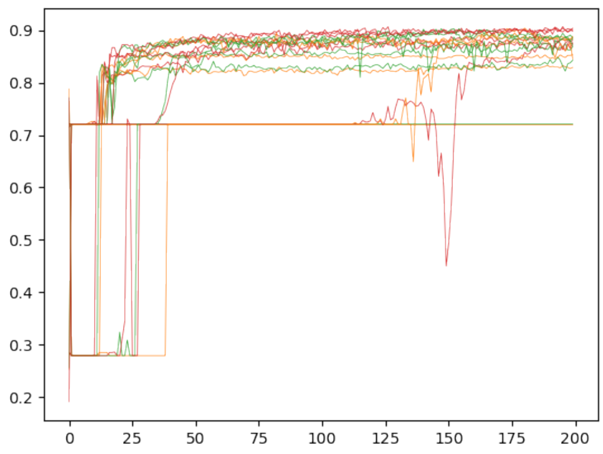
(b) Average rewards without outliers



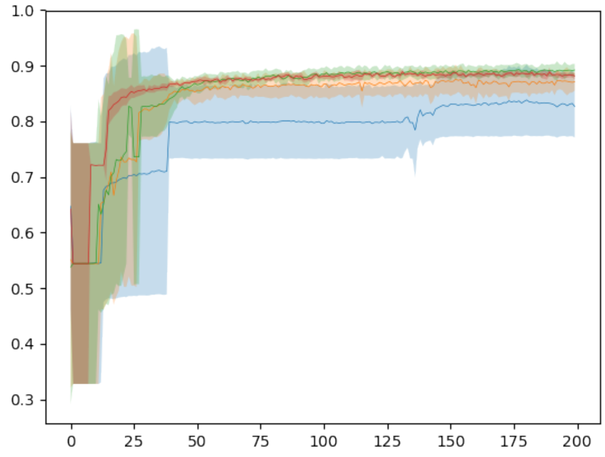
(c) Length of trails by separate runs



(d) Average length of trails without outliers



(e) Correct classification ratios by separate runs



(f) Correct classification ratio without outliers

Figure 5.4: Results of Parameter-noise strategy with removed outliers.

5.2.3 Dropout based exploration

Dropout based exploration is described in Section 3.3. The exploration is induced by the noise from random dropping of a subset of neurons in the hidden layers given the probability p. Dropout was implemented using original PyTorch (Paszke et al., 2017) algorithm. In each sample of the mini-batch different set of neurons are dropped.

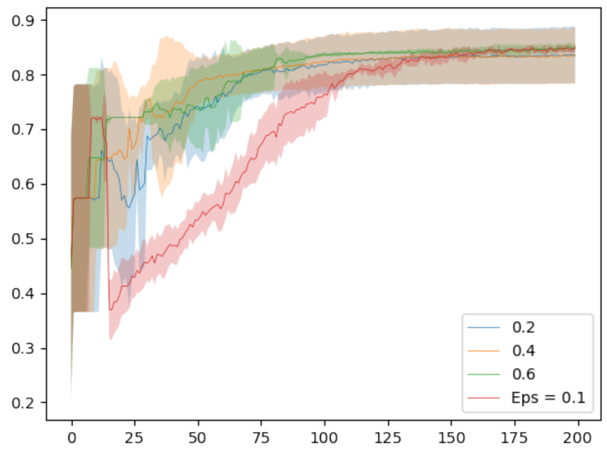
The effect of Dropout is two-fold

- The exploration is induced by the noise from the Dropout of neurons as the noise can result in non-maximizing action selection.
- As the Dropout method is initially a regularization technique thus the Q-value estimates will be more robust.
- As the model adapts to the dropping of the neurons the exploration will very likely decay.

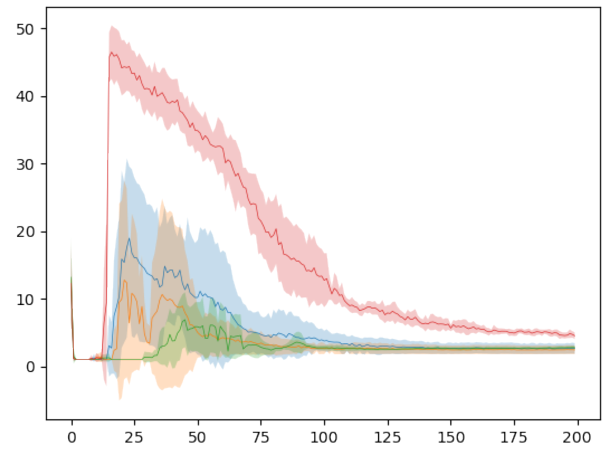
Dropout is applied during exploration of the environment, however, as corresponding to the fundamental Dropout idea, Dropout is also applied during the Q-value update. In order to stabilize the training, we apply Dropout only to the network during the estimation and learning of Q-values and we turn off Dropout for the target Q-values estimate. Thus the target network's Q-value estimates correspond to the current optimal estimates. This change brings more stable convergence as the dropped network is learning to estimate the correct estimates.

During the experiments we were evaluating the influence of Dropout probability p on the exploration. We observed that changes in the probability p corresponded to the changes in resulting exploration. As can be seen in Fig. 5.5d larger values of p induced at the beginning of the training larger exploration. However, during the training process the model has stabilized and all values of probability p lead to similar exploration towards the end of the training as the models parameters have converged.

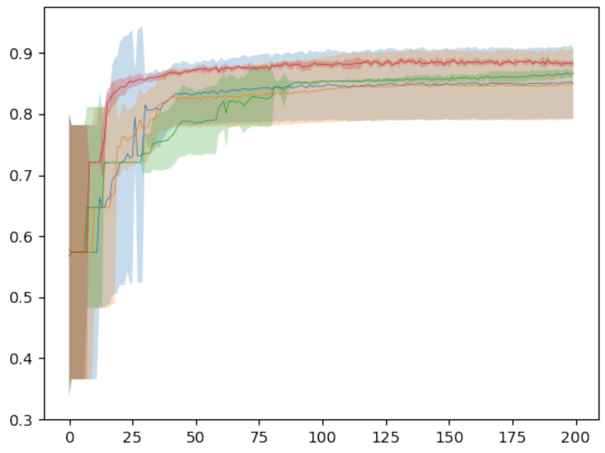
Similarly to the Noise exploration Dropout exploration strategy had more variance with different seeds than using the ϵ-greedy baseline strategy. The larger variance is natural, as the Dropout strategy had significantly smaller Average length (see Fig. 5.5b) compared to the baseline during the whole training process. As the agent has searched through fewer actions than the baseline strategy there is larger probability that the optimal action sequence might not be found. Thus some of the runs get stuck in local optima as can be seen in Fig. 5.6. Similarly to the Parameter-noise method local optima can be detected by zero exploration of there runs. The average rewards with the removed outliers can be seen in Fig. 5.6b We can also observe that in general the exploration resulted in slightly lower entropy (see Fig. 5.5e) than the ϵ-greedy which also resulted in the smaller Average length. Both of the observations correspond to the property of Dropout that it forces the network to learn a simpler hypothesis than it would learn with a full network.



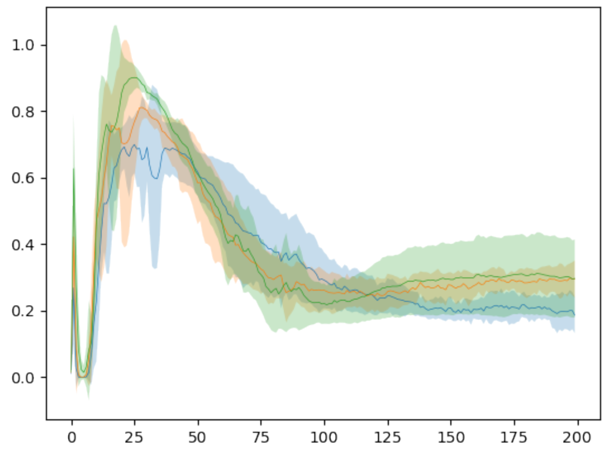
(a) Average rewards



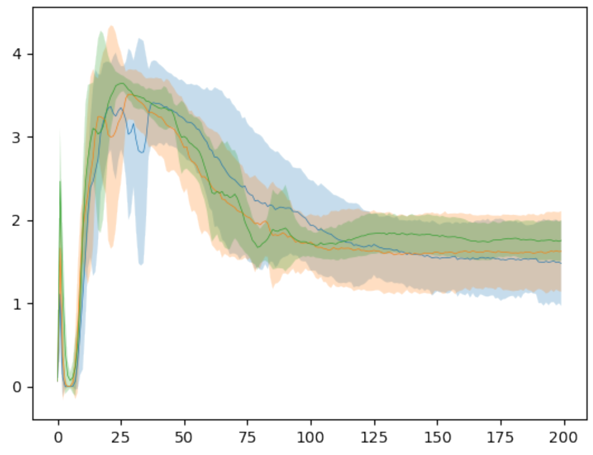
(b) Average length of trails



(c) Correct classification ratio

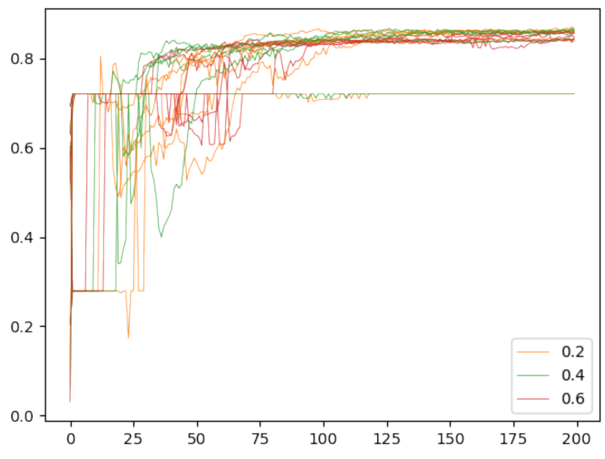


(d) Resulting exploration

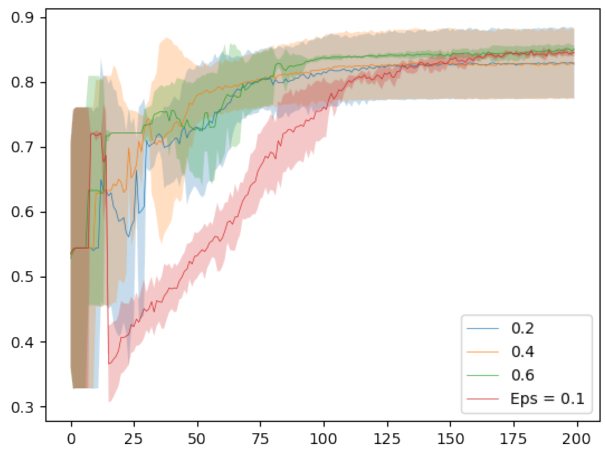


(e) Resulting entropy

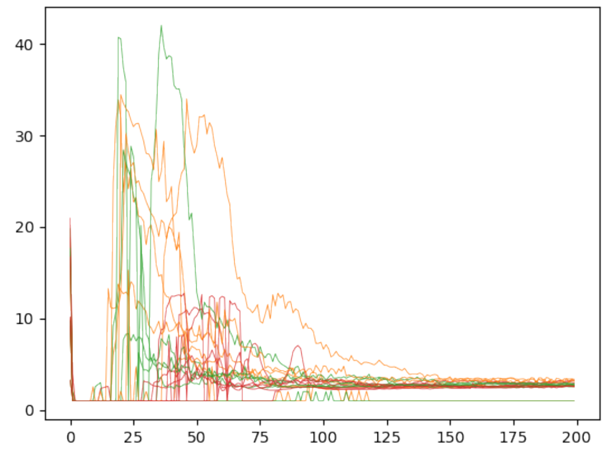
Figure 5.5: Results of the Dropout strategy.



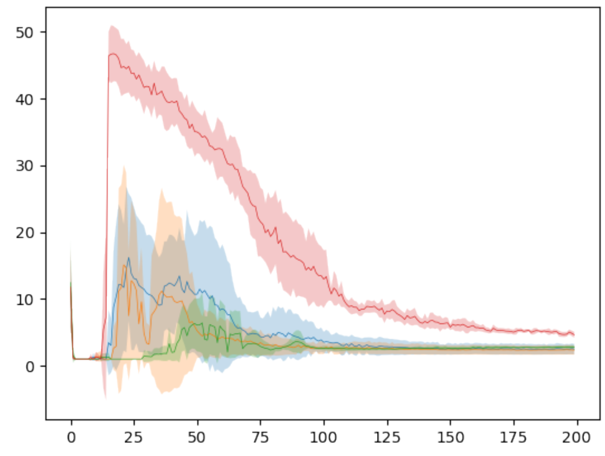
(a) Rewards of separate runs



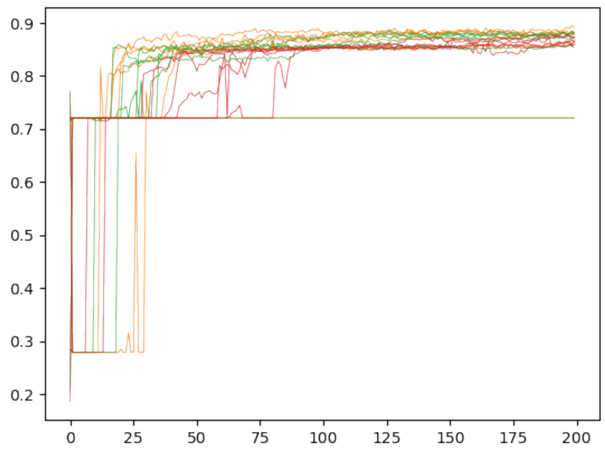
(b) Average rewards without outliers



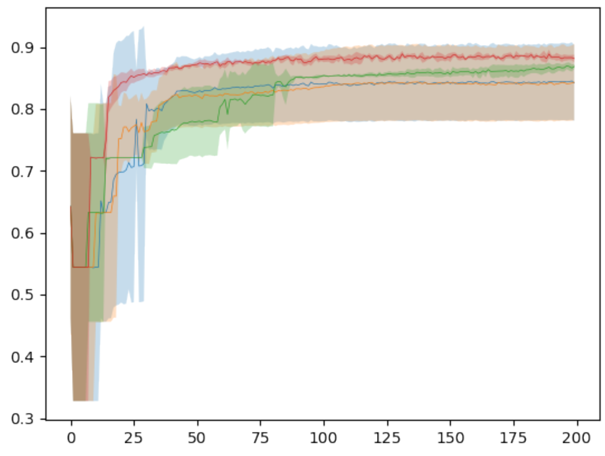
(c) Length of trails by separate runs



(d) Average length of trails without outliers



(e) Correct classification ratios by separate runs



(f) Correct classification ratio without outliers

Figure 5.6: Results of Dropout strategy with removed outliers.

As can be seen in Fig. 5.5b and Fig. 5.5c the Dropout found different Average length to Correct classification ratio (precision) trade-off than the ϵ-greedy strategy. Even though the Dropout strategy achieved faster and better convergence in the optimized criterion of Average rewards than the baseline, the result was achieved with worse precision. Thus we conclude that Dropout would need even smaller cost of step and probably larger cost for incorrect prediction to enforce better precision. Smaller precision of the Dropout strategy also follows from the larger variance of the method. We can observe that the randomness in the dropped models result into larger randomness in the final models, as leaving out of some sets of neurons can probably hurt the performance significantly more than of other set neurons. The optimal setting of Dropout probability was found at $p = 0.4$.

5.2.4 Visits-count based exploration

The Visits-count exploration is an exploration where the Q-value estimate is updated by an exploration term based on the count of visits in the given state. Our implementation was based on the work by (Tang et al., 2017), but adaptations were made to the original algorithm, as our testing environment was quite different from the environment in (Tang et al., 2017). We also wanted to evaluate the method in a more elementary version thus we simplified the proposed algorithm and divided their approach into two different approaches.

We first tested the approach by only using the SimHash algorithm for obtaining indexes to the count tables. We removed the dimensionality reduction via an Autoencoder and directly hashed the given state. We were able to hash directly the input state as our state was smaller than in (Tang et al., 2017) and thus the preprocessing wasn't necessary. The resulting hash was than used as an index in the count table. The hash determined the row in the table and each action had its counts saved in the corresponding column. The results for the Simhash based Visits-count approach can be seen in Fig. 5.7.

We also replaced the count term from (Tang et al., 2017) by the more typical UCB1 value. An interesting property arises from the count bonus term that the model has a natural annealing of the exploration (as can be seen in Fig. 5.7d) even though we keep the exploration value c constant. The annealing of the exploration is caused by the flattening effect of the square root term in the count formula.

We compared several settings for the exploration constant and three settings for the SimHash matrix size: 1, 2 and 4, which correspond to the reduction size of $s = 2^1$, 2^4 and 2^6, respectively. During testing of our approach we realized that the size of the reduced dimensionality didn't influence the final result as even for small sizes ($s = 1$) we were able to find a corresponding exploration constant that brought the same results as with the larger sizes (4 and 6). With all settings a constant can be found which outweights the effect of the exploration annealing that is brought by the

UCB1 count term. This is caused mainly by the fact that has been already observed in other methods that the CCF domain requires in general less exploration. Thus for finding a good optimum it is sufficient, if the exploration is mostly at the beginning of the training. For the similarity of the results we only present the hash size 6 in Fig. 5.7 for which we found the best exploration constant.

Based on the findings we omitted the evaluation of the method using an Autoencoder as in our domain the dimensionality reduction wasn't essential for the performance.

The results show that the exploration constant has a large effect on the convergence. With higher values the agent doesn't find the optimal sequence of actions in terms of length of the path. With smaller exploration the agent can not learn the correct classification. With exploration constant c set around 1 the Visits-count approach has convergence similar to the ϵ-greedy baseline.

5.2.5 Uncertainty driven exploration

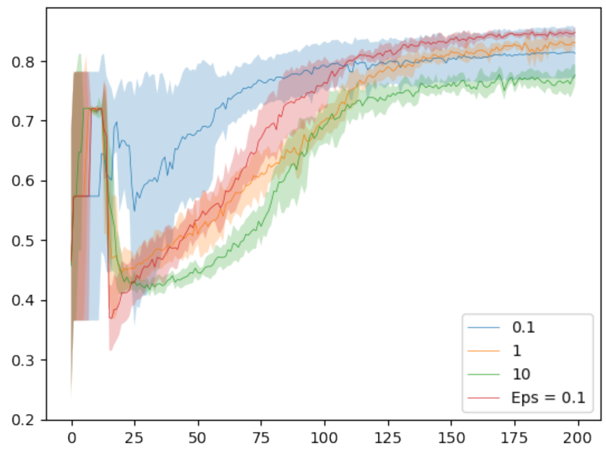
The underlying model of the Uncertainty driven exploration is a reimplementation of the Distributional Perspective on RL (Bellemare et al., 2017). The basic model is enlarged in the output layer so that each Q-value is described by its distribution. The distributions are approximated by discrete bins where the value in each bin corresponds to the probability of an action having Q-value from the corresponding interval.

Similarly to the work by (Bellemare et al., 2017) we restrain the interval of the Q-values to a predefined range. For the CCF datasets we use an interval of $[-1, 1]$ which corresponds to the minimum and maximum optimal Q-values for all initializations. Even though only half of the interval is utilized in each of the settings we keep the interval balanced around zero so that the initial values of the resulting Q-values is around 0 which corresponds to the initial values in all other settings. We use bins of size 0.1 which brought a good trade-off between precision of the distribution approximation and speed of convergence. As stated in (Bellemare et al., 2017) this parameter has significant influence on the training.

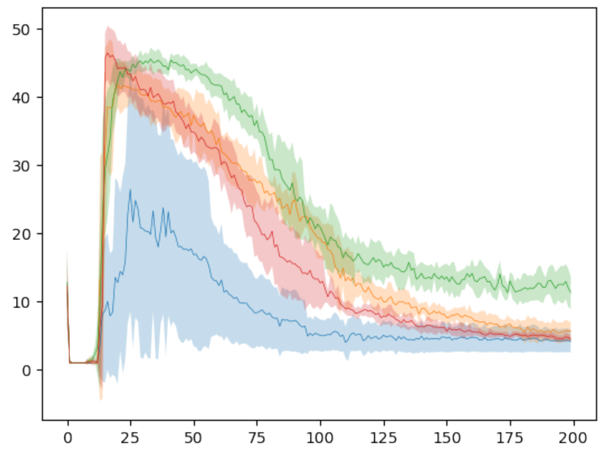
By the entropy updated $\overline{Q}$ estimates are calculated given the definition in Section 3.3.4 and are only applied during the exploration of the environment. The update of the probability distribution is done using the clean estimates without the exploration bonus. As the probability distributions converge the exploration decreases. Thus even though no exploration value annealing is explicitly applied it happens naturally by itself.

Due to the extension of the output layer the model is significantly larger than in other cases. However, the difference is only in the output layer and thus should not excessively increase the capacity of the model. Still it could lead to different speeds of convergence as more weights in the output layer have to be trained and the gradients are calculated with respect to more outputs. In Fig. 5.8 we also provide results of the model with no exploration which show similar results to the original model with $\epsilon = 0$

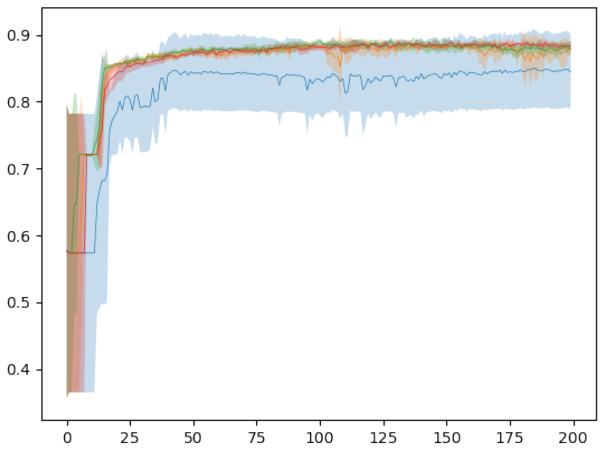
thus the enlarged size of the model has not significant influence on the performance of the model.



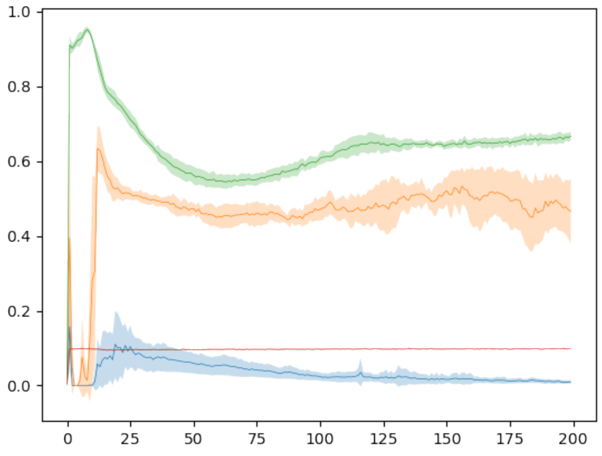
(a) Average rewards



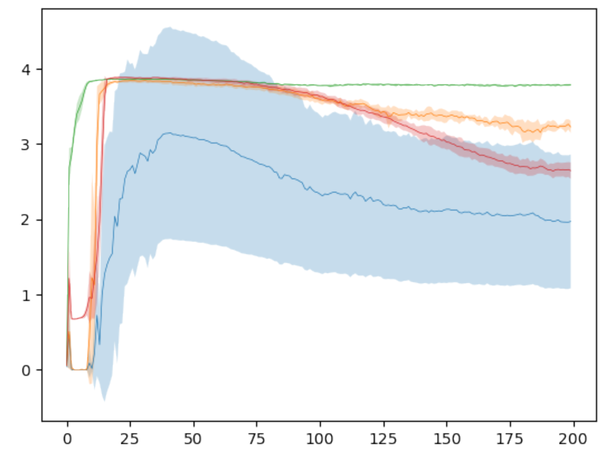
(b) Average length of trails



(c) Correct classification ratio

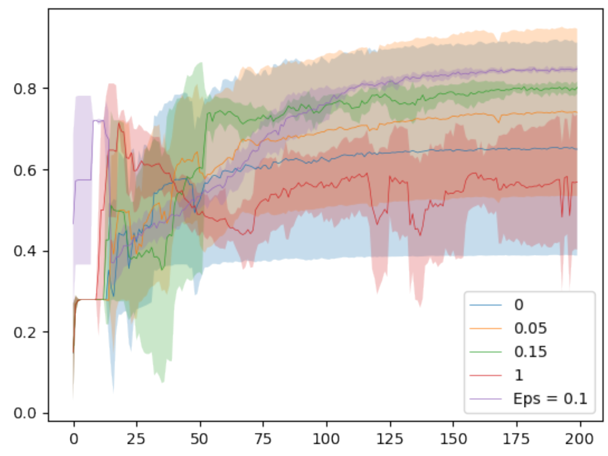


(d) Resulting exploration

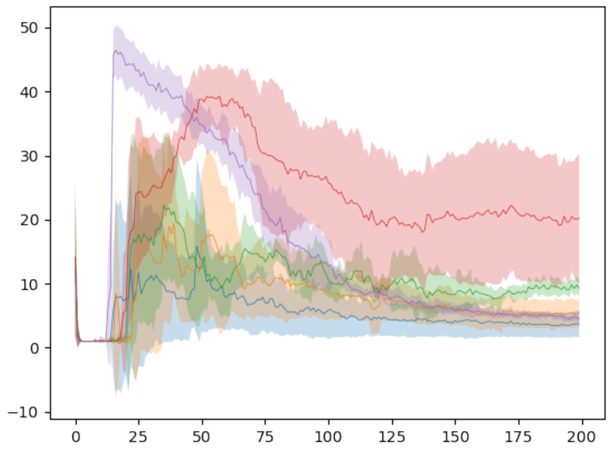


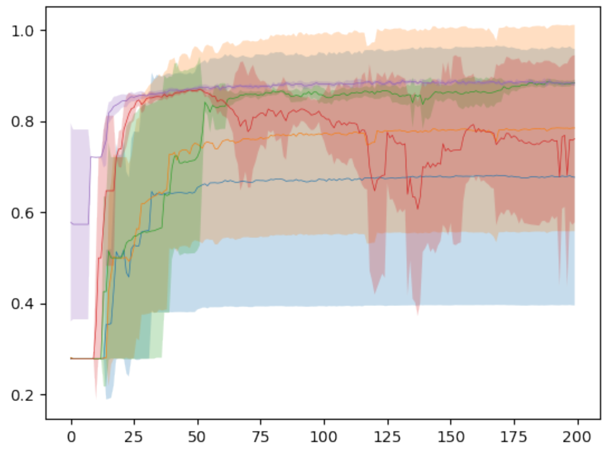
(e) Resulting entropy

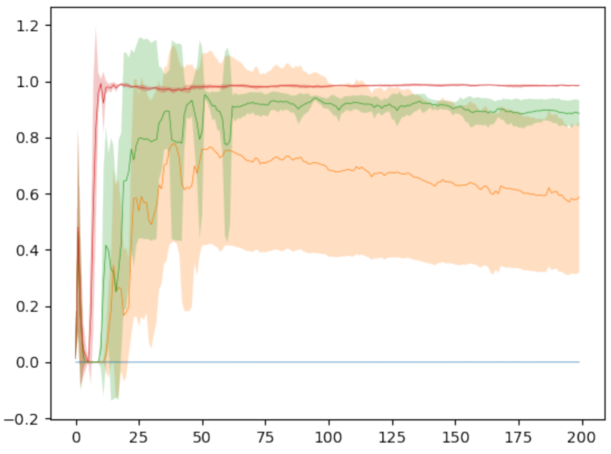
Figure 5.7: Results of the Visits-count Simhash strategy.



(a) Average rewards


(b) Average length of trails


(c) Correct classification ratio


(d) Resulting exploration

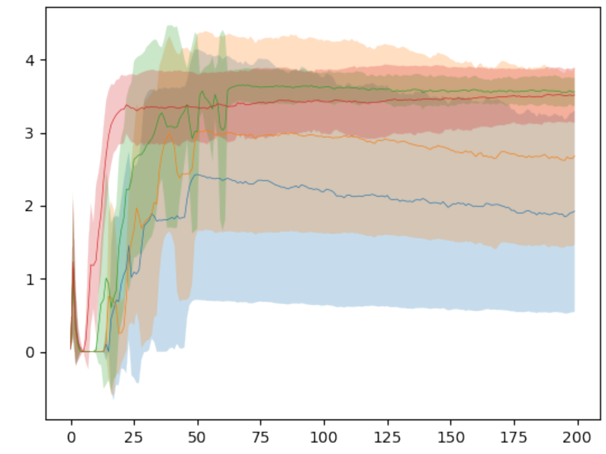
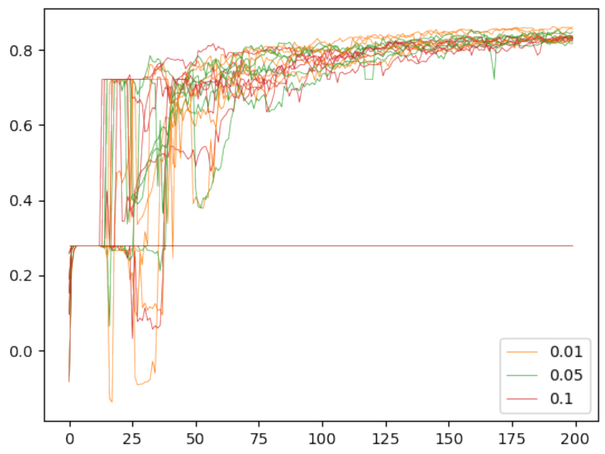
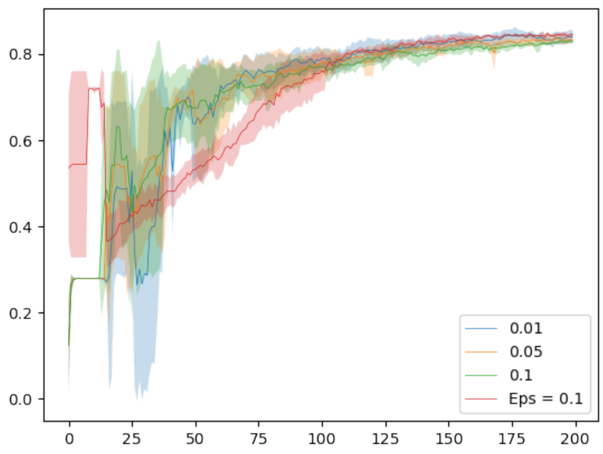

(e) Resulting entropy

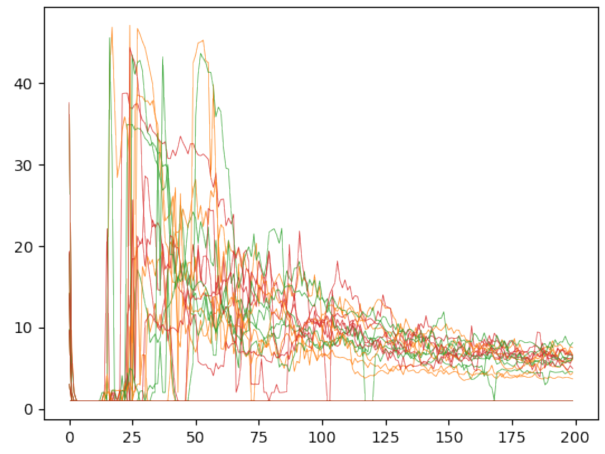
Figure 5.8: Results of the Uncertainty-driven exploration method.



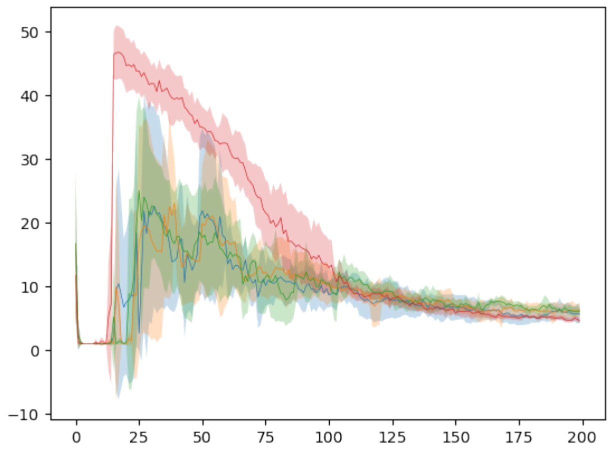
(a) Rewards of separate runs



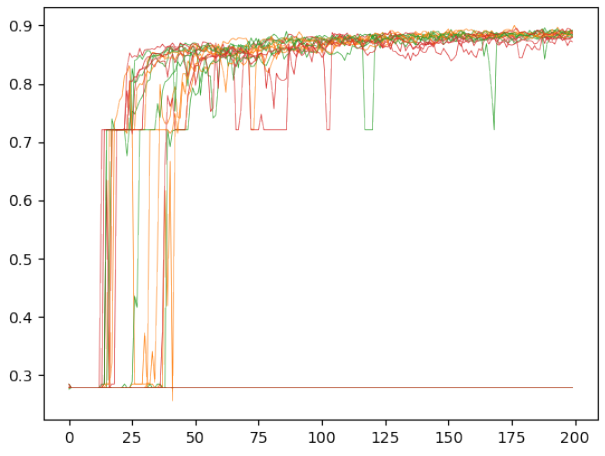
(b) Average rewards without outliers



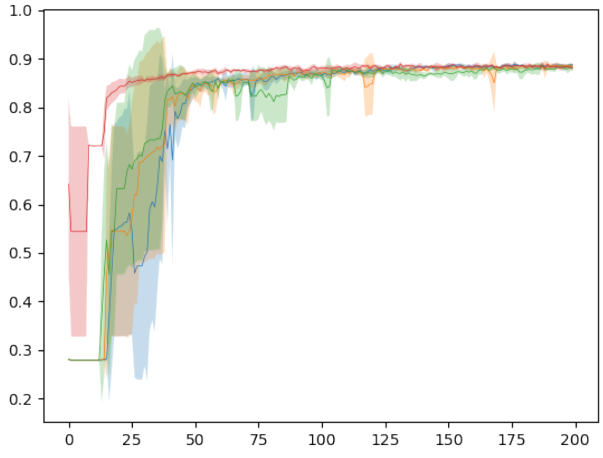
(c) Length of trails by separate runs



(d) Average length of trails without outliers



(e) Correct classification ratios by separate runs



(f) Correct classification ratio without outliers

Figure 5.9: Results of the Uncertainty-driven exploration strategy with removed outliers.

In Fig. 5.8 we present the results of our experiments. We evaluated several settings for the exploration constant c which determines the balance between the Q estimates and the exploration bonus. As can be seen the exploration constant strongly influences the resulting exploration. The large values result in over-exploration towards the end of the training and smaller values of c can sometimes result in bad local optima. The bad local optima arise from the partial non-determinism of the domain. Given an unfavourable random sample from the experience replay buffer the network can get stuck in a local optima which with the small exploration constant cannot be overcome.

Finding an optimal exploration constant setting is a difficult task as running multiple sessions with different values is computationally intensive. For the final comparison the best setting $c = 0.15$ has been selected.

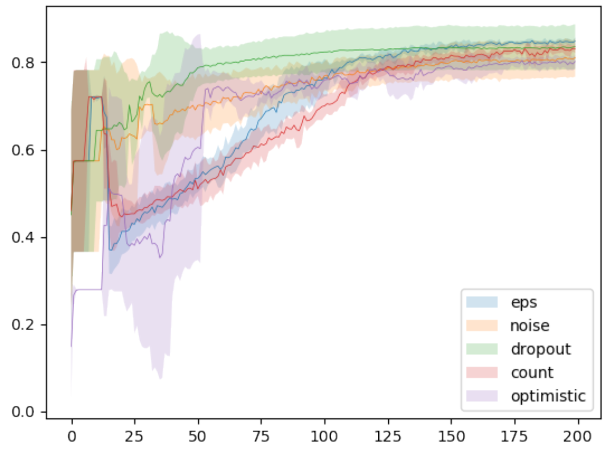
5.2.6 Comparison of all strategies

After the selection of best settings for each strategy we compared all methods against each other. The resulting comparison on the MiniBooNE and the Forest datasets can be seen in Fig. 5.10 and in Fig. 5.11 respectively.

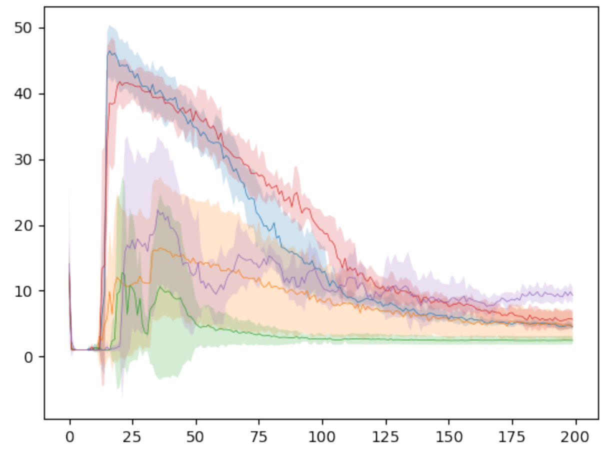
Given our defined performance measure we observe that the Dropout strategy had the fastest and best convergence on both datasets. The Dropout method has a large variance in its results. As was presented in 5.9 the variance is brought by outliers which don't overcome the local optima given by classification according to the class prior. We found that the initial peak of exploration in the Dropout method is brought mainly by the experience replay memory buffer random samples, but overall it doesn't bring enough exploration, as the regularization property of the method then quickly lowers the exploration.

Besides the Visits-count strategy all compared strategies had faster convergence than the baseline ϵ-greedy. The Visits-count method results in a convergence very similar to the ϵ-greedy method. This corresponds to the fact that the ideas behind these strategies are similar as in general both methods should result in uniform search through the environment. The Parameter-noise strategy had faster convergence but it didn't reach as good maximal average reward as the baseline. The strategy also suffers from large variance in results given by the outlier runs which were stuck in local optima. The Uncertainty driven agent also underperformed the baseline as its overall result was strongly disrupted by outliers which remained in very bad local optima.

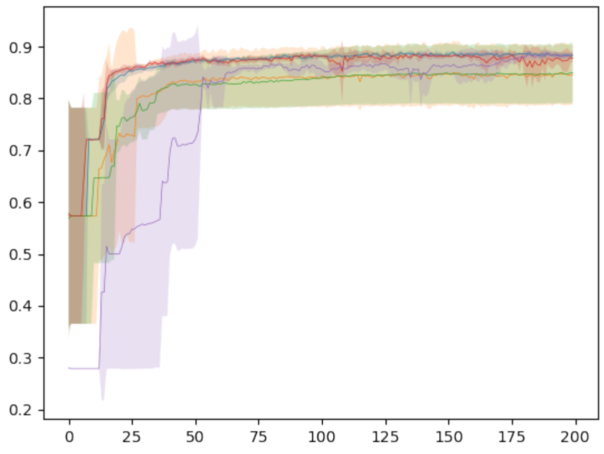
Three out of our examined strategies were prone to stay in local optima which resulted in great variance in their final results. All of the strategies base the exploration on the network parameters and thus the instability of the neural network training is reflected in the instability of the exploration methods. However, in all strategies the states can be easily discovered through as they correspond with exploration throughout the training.



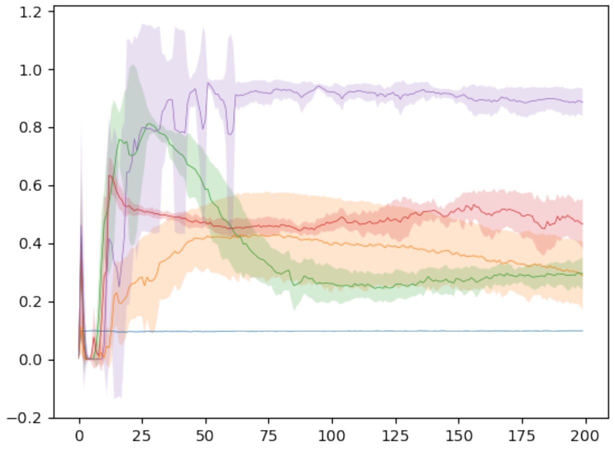
(a) Average rewards



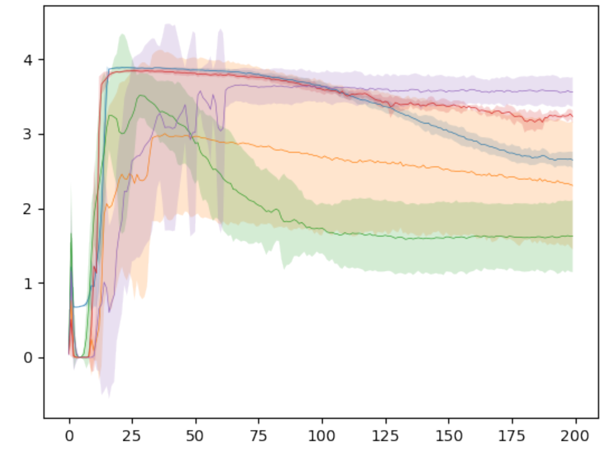
(b) Average length of trails



(c) Correct classification ratio

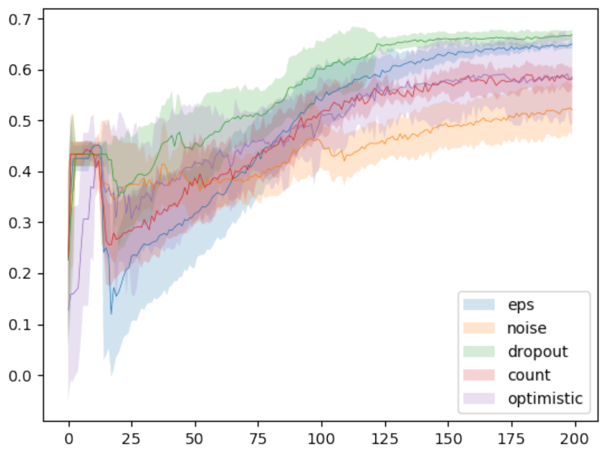


(d) Resulting exploration

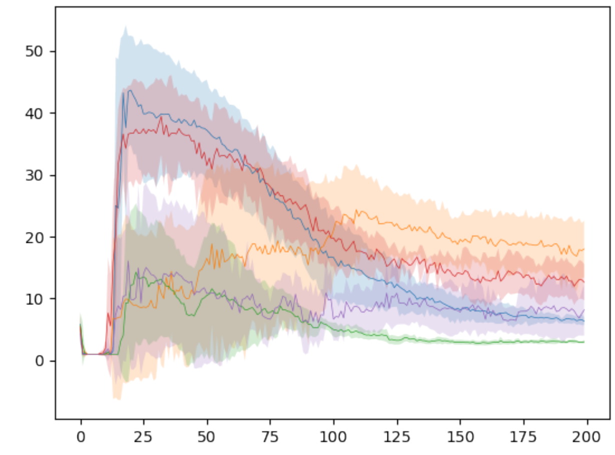


(e) Resulting entropy

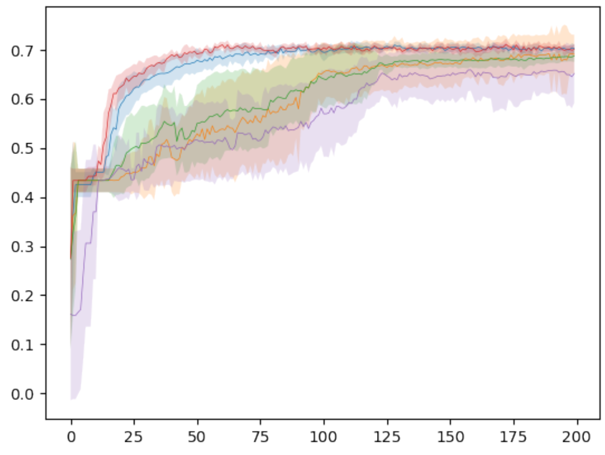
Figure 5.10: Comparison of all examined strategies on the MiniBooNE dataset with settings of $\epsilon = 0.1$, noise randomly selected from $N(0, 0.38)$, probability $p = 0.4$ of dropout, count exploration value $c = 1$ and exploration constant $c = 0.15$ for the uncertainty method.



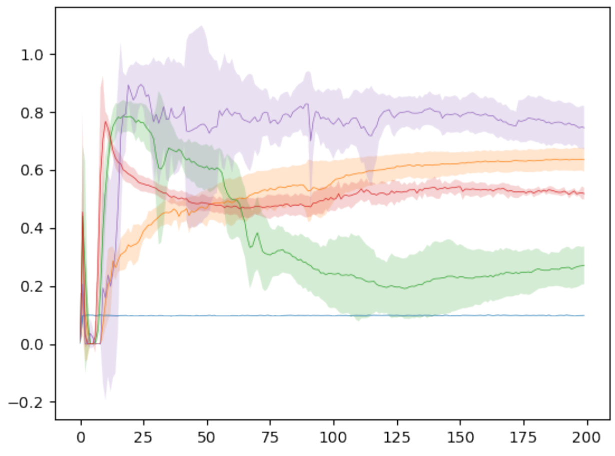
(a) Average rewards



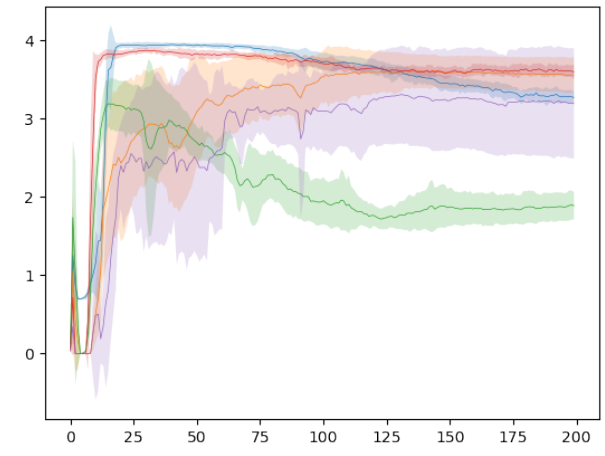
(b) Average length of trails



(c) Correct classification ratio



(d) Resulting exploration



(e) Resulting entropy

Figure 5.11: Comparison of all examined strategies on the Forest dataset with settings of $\epsilon = 0.1$, noise randomly selected from $N(0, 0.005)$, probability $p = 0.1$ of dropout, count exploration value $c = 1$ and exploration constant $c = 0.075$ for the uncertainty method.

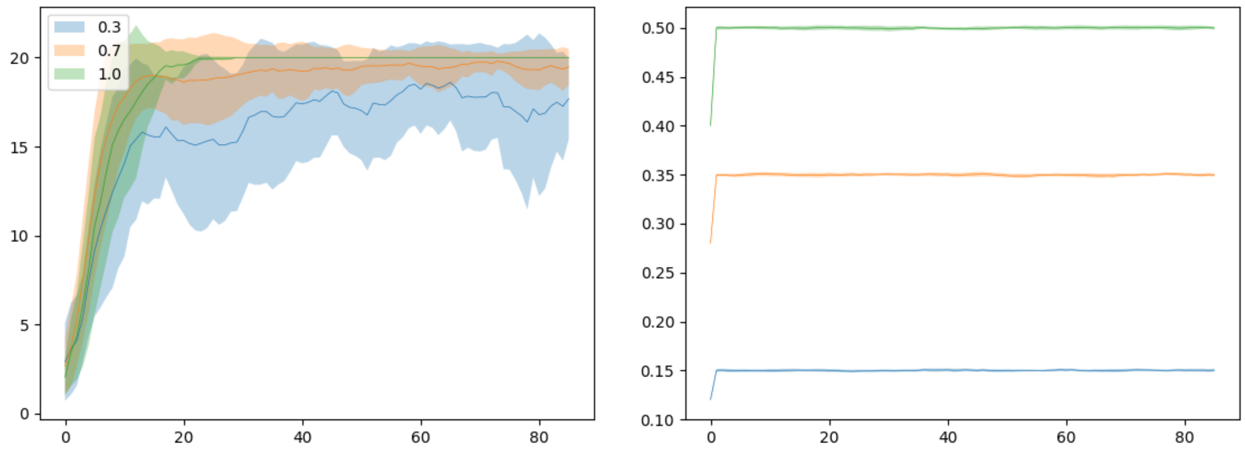
5.3 OpenAI Cart-pole

Second evaluated domain was the task of pole balancing on a cart. There are two main reasons why we selected the Cart-pole task. First, as already explained in Section 4.2, the determinism of the domain enables the comparison of the selected strategies on a task which has only small amount of inherent noisiness (given by the small variations in the starting state). Given the determinism of the domain the exploration is more important because without exploration the agent is very unlikely to discover new states. Our second reason was the simplicity of the task which enabled fast training and thus easier experimenting on the problem. The speed of training was very important as we evaluated multiple strategies with different settings over multiple runs (corresponding to different random seeds).

We evaluated 4 strategies on the task: the ϵ-greedy, Parameter-noise, Dropout and Uncertainty driven exploration. We omitted the Visits-count as in the selected domain the results would not be interesting as the input vector's size was too small, thus no dimensionality reduction would be needed as quantization of the input would suffice. All strategies were evaluated given the same implementation as in the CCF task. For all methods we present the mean and standard deviation of the achieved rewards of the exploiting agent and exploration rate during the training. Each 6 runs, 50k steps, evaluation each 500 steps, runs smoothed with a window of size 5.

5.3.1 ϵ-greedy

For the ϵ-greedy strategy the fastest convergence is brought by the largest exploration value of $\epsilon = 1$. Theoretically such a large exploration value should over-explore. However, the resulting exploration in the actions was significantly smaller. As only two actions are available the ϵ-greedy strategy results in only half of the defined exploration. Large exploration doesn't decrease the resulting performance because there is quite a high degree of freedom in how this problem can be solved - all sequences of actions which manage to keep the pole above the threshold are rewarded equally as no penalty for the number of actions used is given. From this point of view the problem is much easier than the CCF task.



(a) Average reward (b) Average real exploration

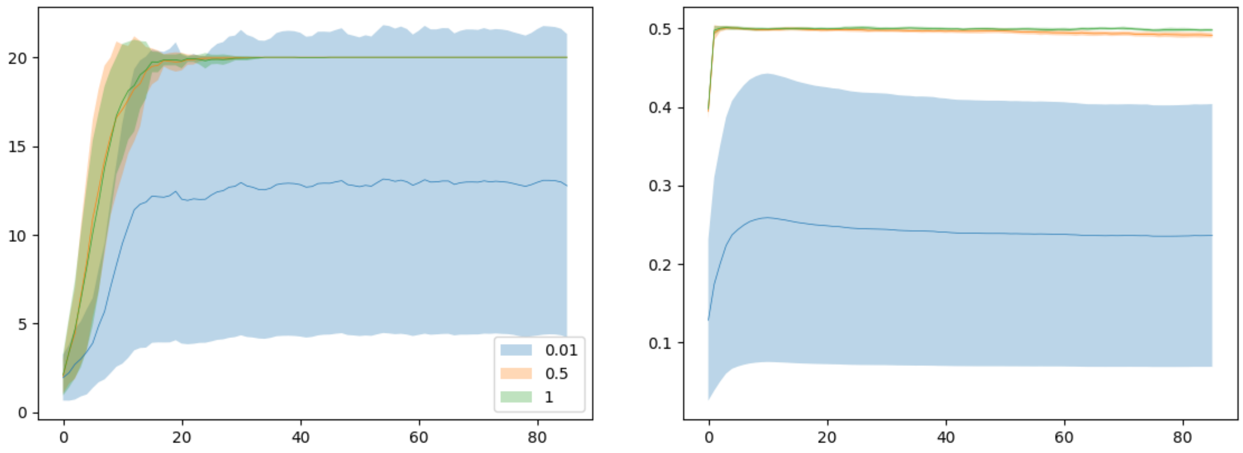
Figure 5.12: Results of ϵ-greedy method on the Cart-pole task.

5.3.2 Parameter-noise

Even though the Parameter-noise method induces noise directly to the weights and thus the resulting exploration is proportional to the absolute Q estimates we observed very fast and smooth convergence. The Parameter-noise exploration outperformed all other strategies on the Cart-pole task with many settings of the value p determining the noise strength in the weights.

The optimal strategy for the agent in the Cart-pole task is to select an opposite action to the direction of the cart. The agent has to learn to frequently change its actions. Thus the optimal learning progress is brought by exploration strategies which result in balanced changes of the network, parameters corresponding to both actions have to be updated. As in the Parameter-noise method different noise is generated for each sample, both actions are selected equally often and their Q estimates are similar over the whole process of training. Thus the noise value p results in similar level of exploration over the whole process of training which enables the agent to discover the optimal action strategy.

Thanks to this behavior the performance of the strategy is mostly independent of the noise value p. For comparison with other strategies the value of $p = 1$. Only extremely low value such as $p = 0.01$ leads to under-exploration as it doesn't overcome the initial small differences in Q estimates.



(a) Average reward (b) Average real exploration

Figure 5.13: Results of Parameter-noise method on the Cart-pole task.

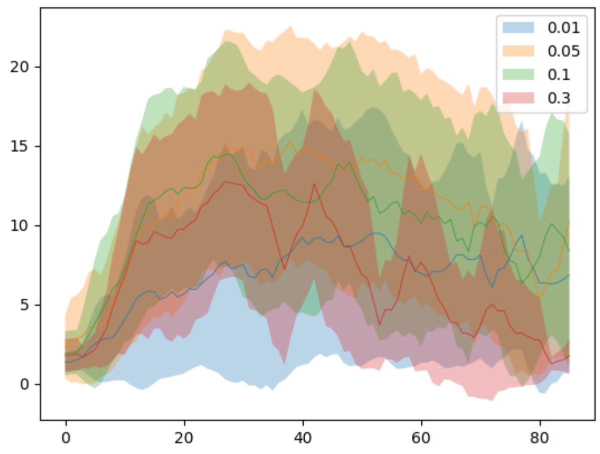
5.3.3 Dropout

The Dropout method was not able to solve the Cart-pole problem as the amount of exploration was not sufficient for the task. As was already shown in the method's results on the CCF the Dropout method induces less exploration then the other strategies due to its regularizing effect. As the regularization is quite fast, we can see in Fig. 5.14b that the exploration quickly drops. The larger the Dropout probability p, the larger the initial peak of exploration, but also the stronger regularization induced in the network. Thus the final exploration has same values as with smaller value of p. The best result of the strategy is brought by $p = 0.05$.

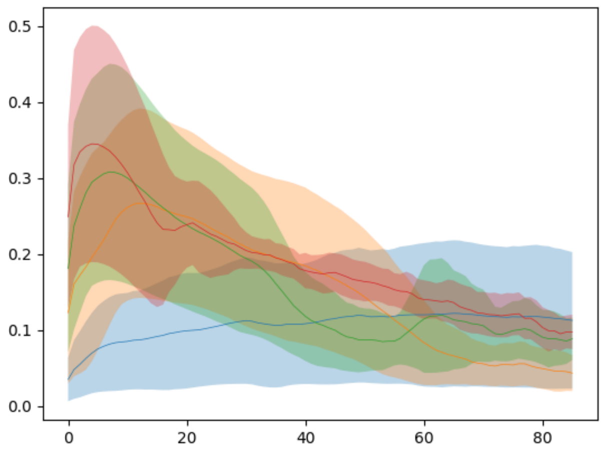
Unlike the Classification of costly features task in the Cart-pole domain the Q-values estimates can gain values from a much wider interval. We can conclude that the difference in the Q-value estimates for the two possible actions quickly become larger than the potential difference between the estimates provided by the dropping of units in the hidden layer. The Q-value difference results from unbalanced selection of actions which emerges from the initial state of the network and the dropout method is not able to surpass it.

The Dropout method is nearly able to reach the optimal value as can be seen in Fig. 5.14a. However, the solution is forgotten by the network as the distances between the two Q-value estimates are enlarged and the exploration falls. Subsequently the network is only trained by one action which only deepens the differences and thus the agent learns to select only one action which leads to sub-optimal results.

On the grounds from our experiments we evaluated a modification of the Dropout method where the dropping of network units is applied only during exploration and not during the update of the network parameters. The noise induced by the different sub-networks is sufficient to solve the problem. The results can be seen in Fig. 5.15. Thus we can conclude that the regularization property of the Dropout method effectively works against exploration as the network adapts to the noise resulting from the various dropped sub-networks.

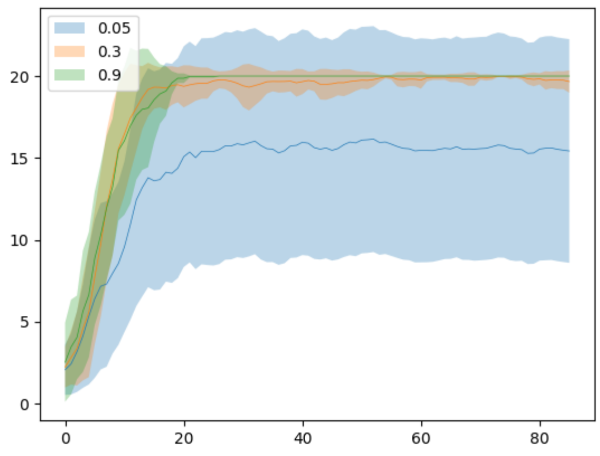


(a) Average reward

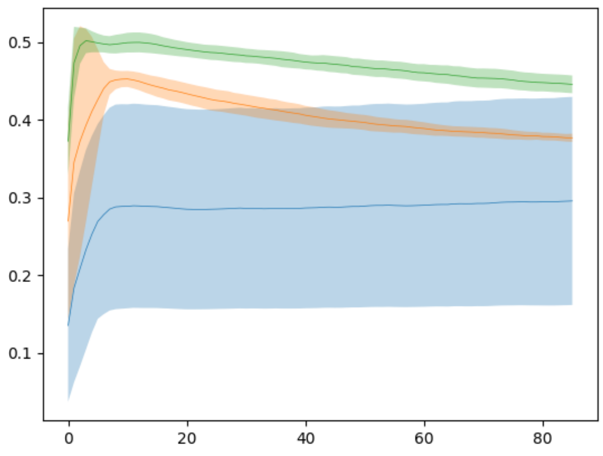


(b) Average real exploration

Figure 5.14: Results of Dropout method on the Cart-pole task.



(a) Average reward



(b) Average real exploration

Figure 5.15: Results of Dropout applied only in action selection on the Cart-pole task.

5.3.4 Uncertainty driven exploration

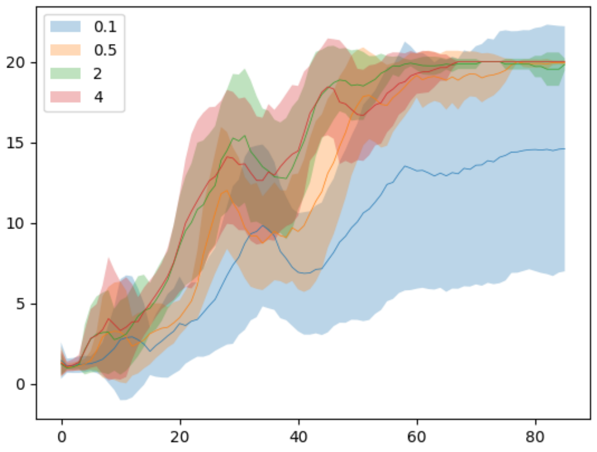
We evaluated the proposed Uncertainty driven exploration method on the Cart-pole task. For the evaluation we utilized same model as in the CCF task with only a few changes. We enlarged the interval for possible Q-values to $[-20, 20]$ which correspond to the original setting from (Bellemare et al., 2017), the size of bins was set to 0.1 which corresponded to reward for 1 step in the Cart-pole task.

Similarly to the Dropout method, the Uncertainty driven exploration performs worse than the Parameter-noise and the ϵ-greedy methods. Worse performance can be explained by different development of exploration.

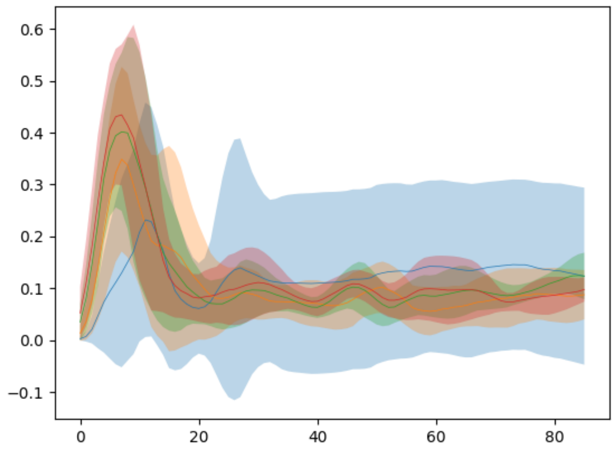
Ac can be seen in Fig. 5.16b the overall exploration level induced by the method is lower. The Cart-pole task has only small variation in the initial states and the initial state of the network brings practically same entropy bonus to both actions, thus practically no exploration is present during the filling of the memory buffer. As the network updates are at start calculated from samples mainly from the initialization of the buffer the method starts with lower exploration.

Next the exploration quickly rises due to large differences in the entropy of the Q-value distributions. The exploration then again decreases as both distributions converge and thus the influence of the exploration bonus diminishes. Afterwards the exploration remains stable till the end of the training.

In Fig. 5.16a we present the resulting rewards obtained by the exploitation agent. We can see that even with the widely different exploration progress the method is able to solve the task, however, the convergence is significantly slower. In order to achieve better results on the deterministic domains an additional randomization of the actions during the initialization of the memory buffer would be needed. For comparison with other methods the exploration constant has been set to $c = 2$.



(a) Average reward

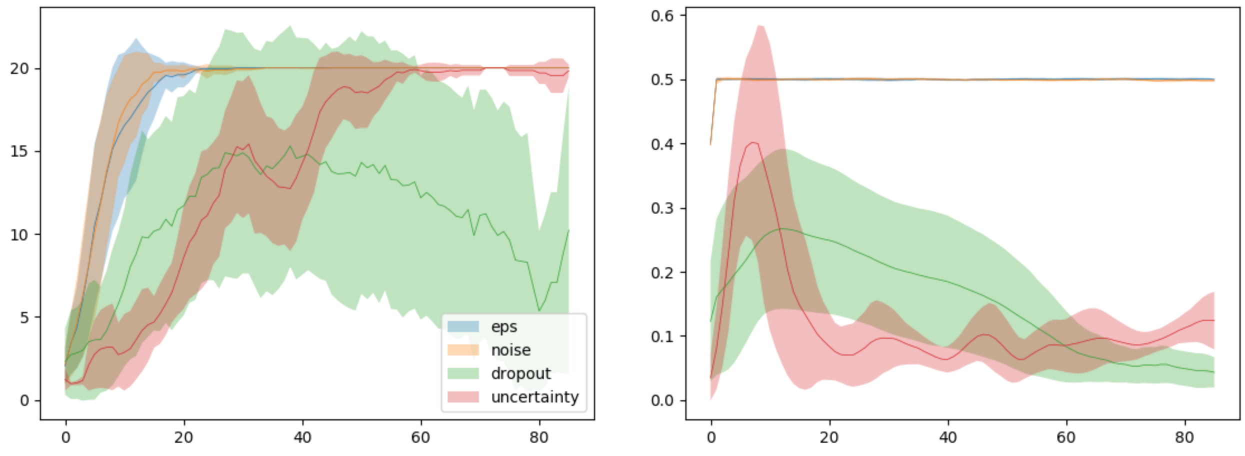


(b) Average real exploration

Figure 5.16: Results of Uncertainty driven method on the Cart-pole task.

5.3.5 Comparison of all strategies

In Fig. 5.17 we present the comparison of the ϵ-greedy, Parameter-noise, Dropout and Uncertainty driven exploration. In the Cart-pole task the Parameter-noise and the ϵ-greedy methods outperformed the two remaining methods. The Dropout method has high rate of convergence too, however, the regularization corrupts the optimal values and the method diverges. Similarly to the Dropout method, the Uncertainty driven exploration performs worse than the Parameter-noise and even the ϵ-greedy baseline method. Unlike the Dropout method it converges slower than the other methods, however, its convergence is more stable and the network doesn't diverge from the optimum.



(a) Average reward (b) Average real exploration

Figure 5.17: Comparison of all examined strategies on the Cart-pole task with settings of $\epsilon = 1$, noise randomly selected from $N(0, 1)$, probability $p = 0.05$ of dropout and exploration constant $c = 2$ for the uncertainty method.

Chapter 6

Conclusion

In the theoretical part of the thesis existing exploration strategies are presented for the tabular (Section 3.1) and the function approximation reinforcement learning (Section 3.2). Next we explain selected strategies in more detail in Section 3.3.

Based on the research of existing approaches three strategies were selected and evaluated in the practical part. First we selected a randomization method the Parameter space noise by (Plappert et al., 2017) which induces exploration by adding noise to the parameters of the estimation model. Next we evaluated the Dropout method by (Srivastava et al., 2014) which is inducing exploration by utilizing the uncertainty of the model. At last the Visits-count exploration method by (Tang et al., 2017) has been selected as it is an implementation of the well-known tabular exploration method in the high-dimensional RL problems.

We proposed a novel exploration approach called Uncertainty driven exploration which brings exploration by adjusting the value estimates used for action selection by a bonus proportional to their uncertainty. The method combines a novel approach to Q-value estimation from the Distributional perspective to Reinforcement learning (Bellemare et al., 2017) with a tabular exploration heuristics called the Optimism in the face of uncertainty.

The strategies have been evaluated on 2 domains – the Classification with costly features (CCF) task and the Cart-pole problem. We showed that all of the strategies except for the Dropout strategy are applicable in both domains.

For the CCF task we demonstrated that the Dropout method surpassed the performance of the ϵ-greedy baseline. The Parameter-noise method had faster convergence, but in the end lower average rewards than the baseline. The Visits-count method converged similarly to the ϵ-greedy method. Our proposed strategy has also been evaluated and showed results comparable in terms of the speed of convergence to the best strategies but with slightly worse average rewards.

On the Cart-pole task we evaluated three of the four implemented strategies. We omitted the Visits-count method because even though the method is applicable in principle the dimensionality reduction doesn't bring any benefit for a problem with such a small input space. Unlike the CCF domain the Cart-pole task needs stronger exploration for best results. All of the strategies were able to at least partially solve the problem. Both Parameter-noise and the Uncertainty driven strategy were able to solve the problem completely. The Uncertainty driven strategy had slower convergence as the strategy brought lower amount of randomness during the initial memory buffer filling. The Dropout strategy wasn't able to fully solve the task due to its regularization behavior which is effectively working against exploration.

We found that in the less exploration demanding problems the Dropout method outperformed the remaining strategies due to its natural annealing of exploration level. Similarly the proposed Uncertainty driven exploration was observed to have an inherent exploration annealing. However, for the same reason both these methods performed worse on the Cart-pole task.

The main problem of most of the strategies has been the high variance of the results. We found that the exploration strategies which are based on the state of the network suffer from the occasional instability of the function approximation methods. Subsequently the exploration cannot overcome such a situation and the reinforcement learning agent ends in poor local optima.

We identified two main directions for the future work. It would be interesting to evaluate the studied strategies on more complex domains such as the game environments of Atari (Bellemare et al., 2013a), which couldn't be done in this thesis due to their higher computational demands. In the Uncertainty driven strategy we encountered the problem of balancing between over- and under-exploration. Further work thus might focus on the adaptation of the exploration value c during the training with the goal of increasing the chance of escaping local optima, while at the same time not exploring too much.

Bibliography

P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002. 3.1.2

L. J. Ba, R. Kiros, and G. E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016. URL `http://arxiv.org/abs/1607.06450`. 3.3.1

K. Bache and M. Lichman. Uci machine learning repository. 2013. 4.1

A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983. 4.2

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.(JAIR)*, 47: 253–279, 2013a. 2.1.2, 6

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Int. Res.*, 47(1):253–279, May 2013b. ISSN 1076-9757. URL `http://dl.acm.org/citation.cfm?id=2566972.2566979`. 3.3.1

M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying Count-Based Exploration and Intrinsic Motivation. jun 2016. URL `http://arxiv.org/abs/1606.01868`. 3.2.2, 3.3.2

M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. *arXiv preprint arXiv:1707.06887*, 2017. 1.2, 3.3.4, 5.2.5, 5.3.4, 6

R. Bellman. The theory of dynamic programming. Technical report, RAND CORP SANTA MONICA CA, 1954. 2.1.1

R. Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, 6 (5):679–684, 1957. ISSN 00959057, 19435274. URL `http://www.jstor.org/stable/24900506`. 2.1.1, 2.1.1

C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight Uncertainty in Neural Networks. *Icml*, 37:1613–1622, may 2015. URL `http://arxiv.org/abs/1505.05424http://arxiv.org/abs/1505.05424{%}5Cnhttp://www.arxiv.org/pdf/1505.05424.pdf`. 3.2.2

R. I. Brafman and M. Tennenholtz. R-MAX - A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning. *Journal of Machine Learning Research*, 3(Oct):213–231, 2002. ISSN ISSN 1533-7928. URL `http://www.jmlr.org/papers/v3/brafman02a.html`. 1.1, 3.1.2

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016. URL `http://arxiv.org/abs/1606.01540`. 4.2

M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388. ACM, 2002. 3.3.2

H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, pages 493–507, 1952. 3.1.2

M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg. Noisy Networks for Exploration. jun 2017. URL `http://arxiv.org/abs/1706.10295`. 3.2.1

Y. Gal and Z. Ghahramani. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. jun 2015. URL `http://arxiv.org/abs/1506.02142`. 3.2.2, 3.3.3

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010. 5.1

W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963. 3.1.2

R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel. VIME: Variational Information Maximizing Exploration. may 2016. URL `http://arxiv.org/abs/1605.09674`. 3.2.2

R. A. Howard. Dynamic programming and markov processes. 1960. 2.1.1

T. Jaksch, R. Ortner, and P. Auer. Near-optimal Regret Bounds for Reinforcement Learning *. *Journal of Machine Learning Research*, 11:1563–1600, 2010. URL `http://www.jmlr.org/papers/volume11/jaksch10a/jaksch10a.pdf`. 3.1

J. Janisch, T. Pevnỳ, and V. Lisỳ. Classification with costly features using deep reinforcement learning. *arXiv preprint arXiv:1711.07364*, 2017. 4.1, 5.1, 5.1, 5.2.1

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015. 2.1.2

N. Metropolis and S. Ulam. The monte carlo method. *Journal of the American statistical association*, 44(247):335–341, 1949. 2.1.1

MIT. Explicit Explore or Exploit (E3) Algorithm. URL `https://ocw.mit.edu/courses/mechanical-engineering/2-997-decision-making-in-large-scale-systems-spring-2004/lecture-notes/lec{_}9{_}v1.pdf`. 1.1, 3.1.2

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. 2.1.2, 2.1.2, 2.2, 5.1

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016. 2.1.2, 3.3.1

V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010. 5.1

I. Osband, B. V. Roy, B. Edu, Z. Wen, and Z. Com. Generalization and Exploration via Randomized Value Functions. URL `http://proceedings.mlr.press/v48/osband16.pdf`. 3.3.3

I. Osband, C. Blundell, A. Pritzel, and B. V. Roy. Deep Exploration via Bootstrapped DQN. *arXiv*, 1602.04621:1–18, 2016. ISSN 10495258. doi: 10.1145/2661829.2661935. URL `http://papers.nips.cc/paper/6501-deep-exploration-via-bootstrapped-dqn.pdfhttp://arxiv.org/abs/1602.04621`. 3.2.1

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017. 5.2.3

D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven Exploration by Self-supervised Prediction. *Icml*, pages 2778–2787, jul 2017. ISSN 1938-7228. doi: 10.1109/CVPRW.2017.70. URL `http://proceedings.mlr.press/v70/pathak17a.html`. 3.2.2

M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*, 2017. 3.2.1, 3.3.1, 3.3.1, 5.2.2, 6

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985. 2.2.1

T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized Experience Replay. nov 2015. URL `http://arxiv.org/abs/1511.05952`. 2.1.2

J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1889–1897, 2015. 3.3.2

S. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine learning*, 38(3):287–308, 2000. 3.1.1

N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014. 3.3.3, 6

B. C. Stadie, S. Levine, and P. Abbeel. Incentivizing Exploration In Reinforcement Learning With Deep Predictive Models. *arXiv*, pages 1–11, 2015. URL `https://arxiv.org/pdf/1507.00814.pdfhttp://arxiv.org/abs/1507.00814`. 3.2.2

R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988. 2.1.1

R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981. 2.1, 3.1

M. Tan. Cost-sensitive learning of classification knowledge and its applications in robotics. *Machine Learning*, 13(1):7–33, 1993. 1.2, 4.1

H. Tang, R. Houthooft, D. Foote, A. Stooke, O. X. Chen, Y. Duan, J. Schulman, F. DeTurck, and P. Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2750–2759, 2017. 3.2.2, 3.3.2, 3.3.2, 5.2.4, 6

W. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933. 1.1

S. B. Thrun. Efficient exploration in reinforcement learning. Technical report, Pittsburgh, PA, USA, 1992. 3, 3.1

T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012. 5.1

A. van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems*, pages 4790–4798, 2016. 3.2.2, 3.3.2

H. van Hasselt, A. Guez, and D. Silver. Deep Reinforcement Learning with Double Q-learning. sep 2015. URL `http://arxiv.org/abs/1509.06461`. 2.1.2

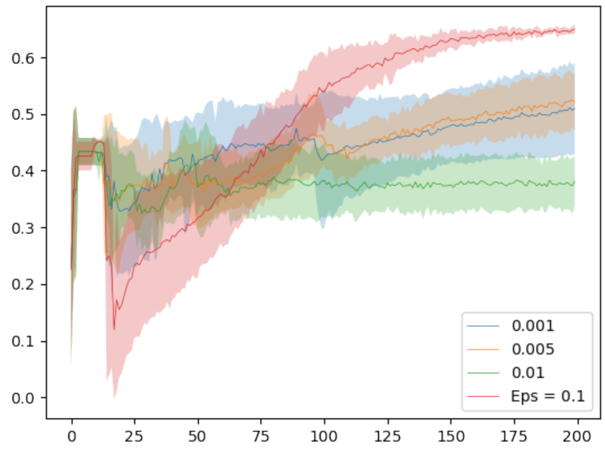
C. J. C. H. Watkins. *Learning from delayed rewards.* PhD thesis, King's College, Cambridge, 1989. 2.1.1

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. 2.1.2

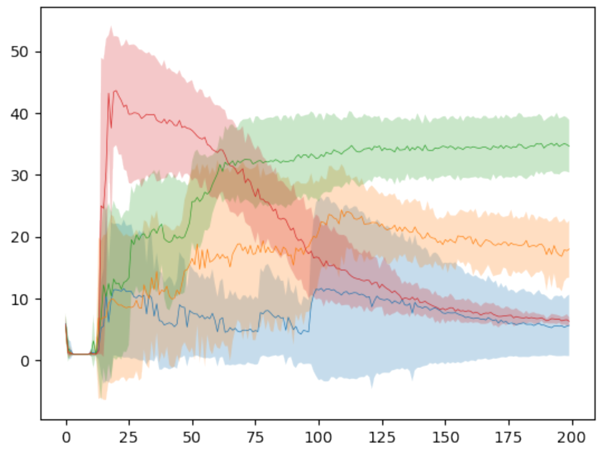
Appendix A

Further results

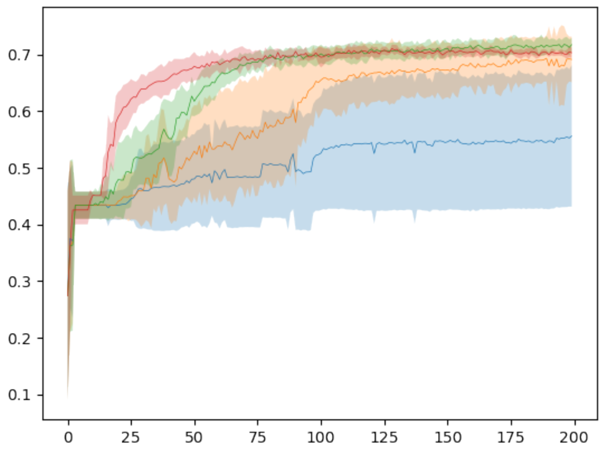
For completeness we will present here the results showing the influence of the exploration constants on the Forest dataset which was omitted in the main text for comprehensiveness of the thesis and also due to the fact that the resulting conclusions from the Forest dataset are similar to the results on the MiniBooNE dataset.



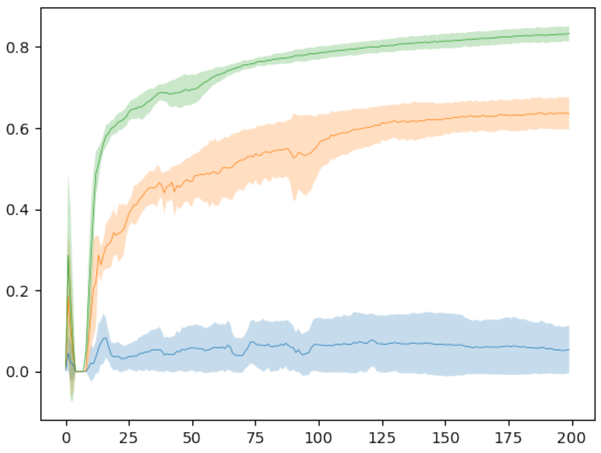
(a) Average rewards



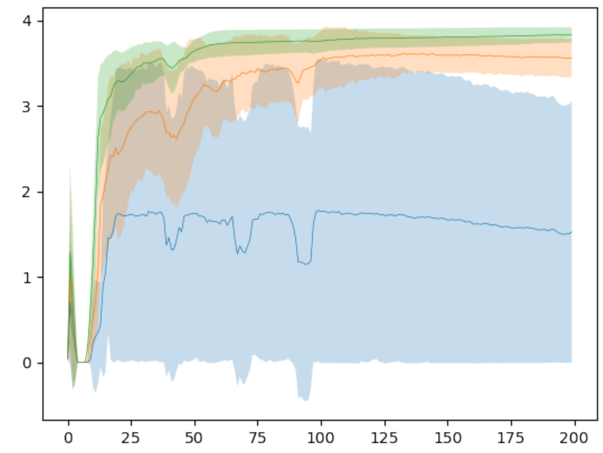
(b) Average length of trails



(c) Correct classification ratio

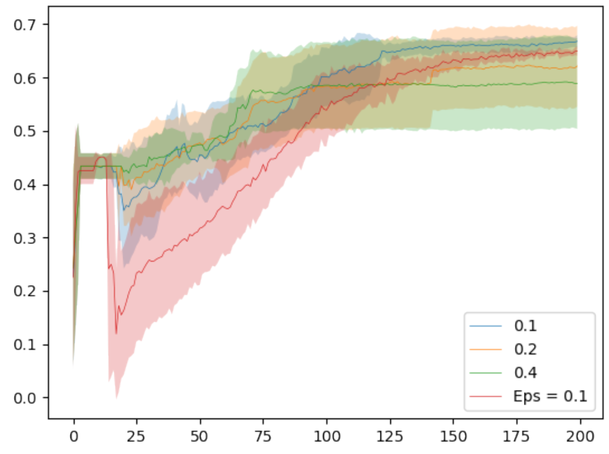


(d) Resulting exploration

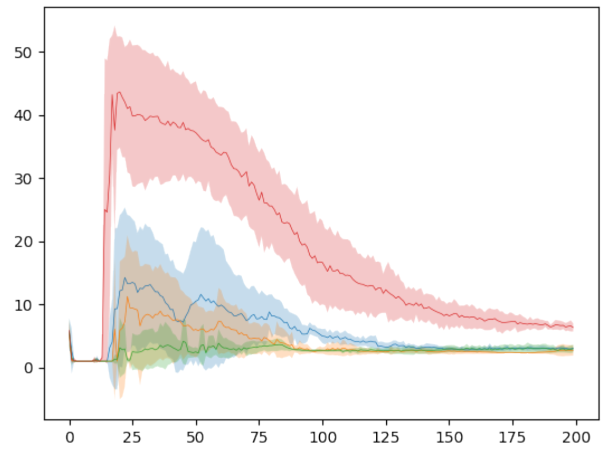


(e) Resulting entropy

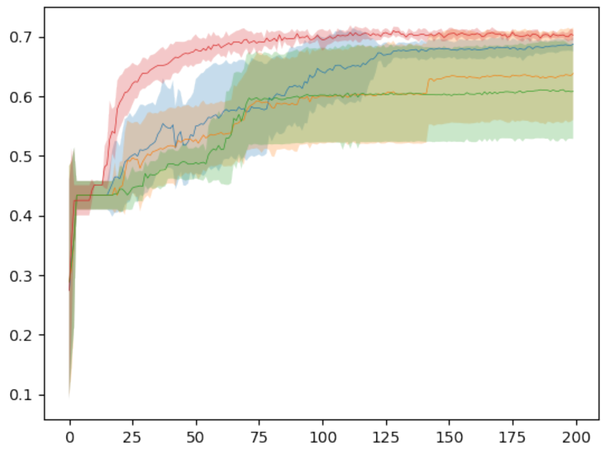
Figure A.1: Results of Parameter-noise strategy on the Forest dataset.



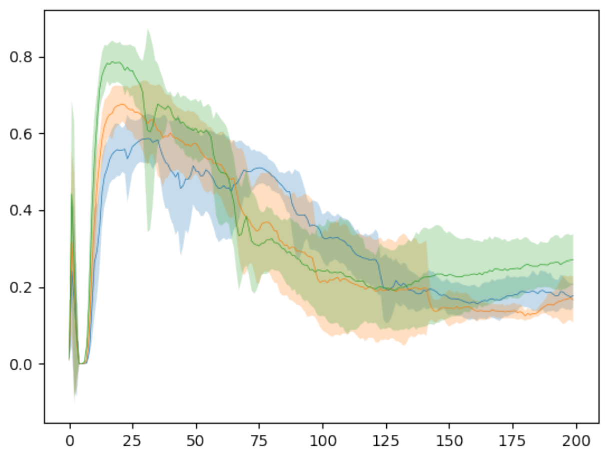
(a) Average rewards



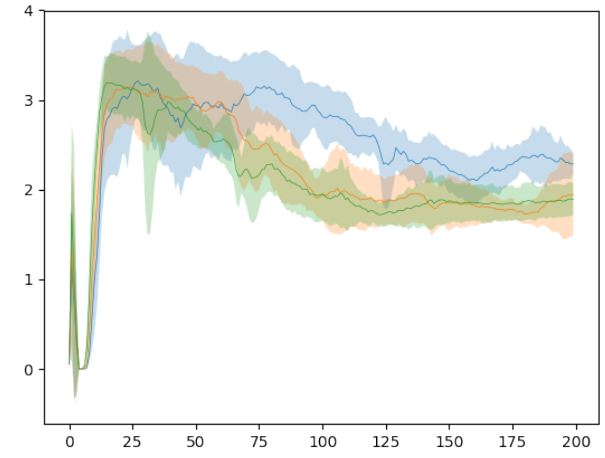
(b) Average length of trails



(c) Correct classification ratio

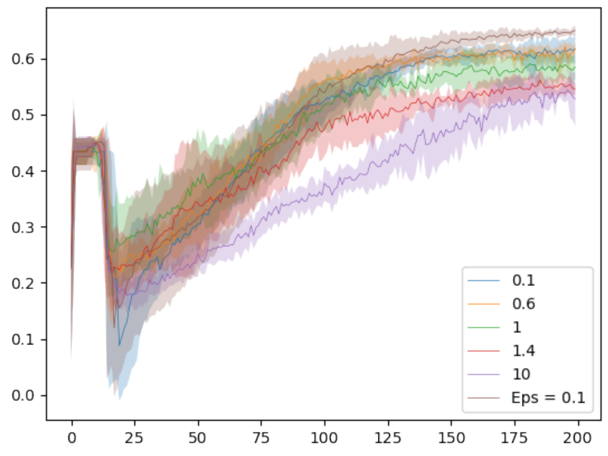


(d) Resulting exploration

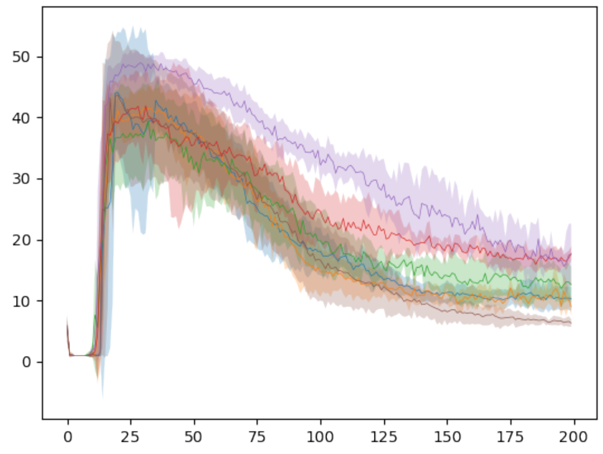


(e) Resulting entropy

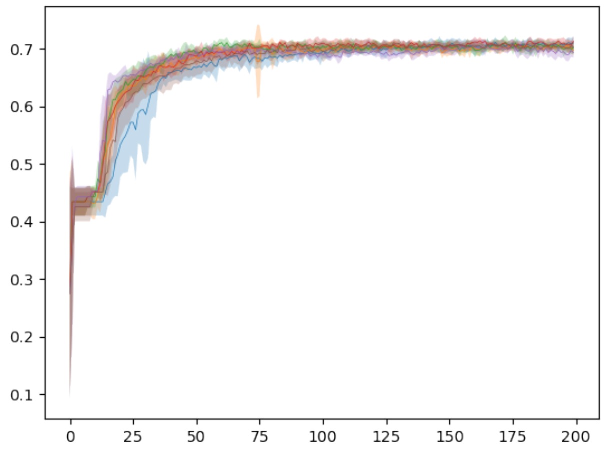
Figure A.2: Results of the Dropout strategy on the Forest dataset.



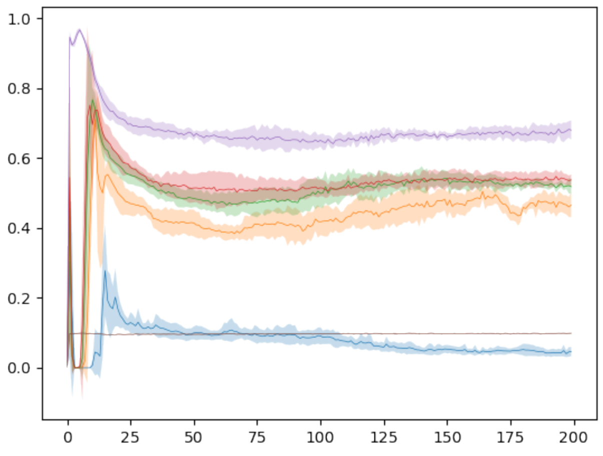
(a) Average rewards



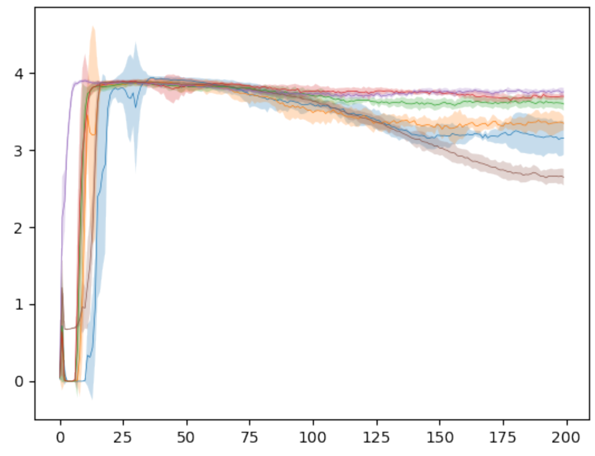
(b) Average length of trails



(c) Correct classification ratio

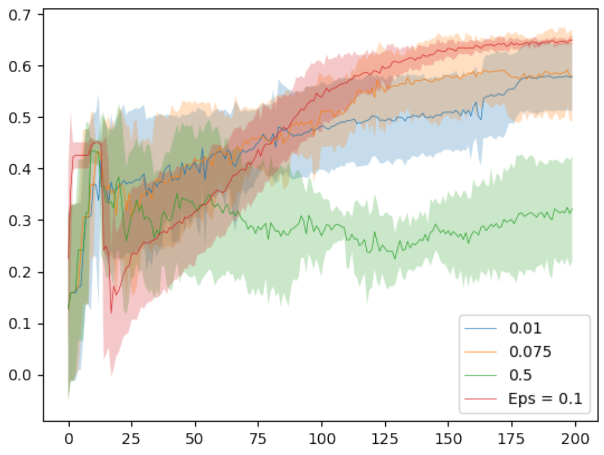


(d) Resulting exploration

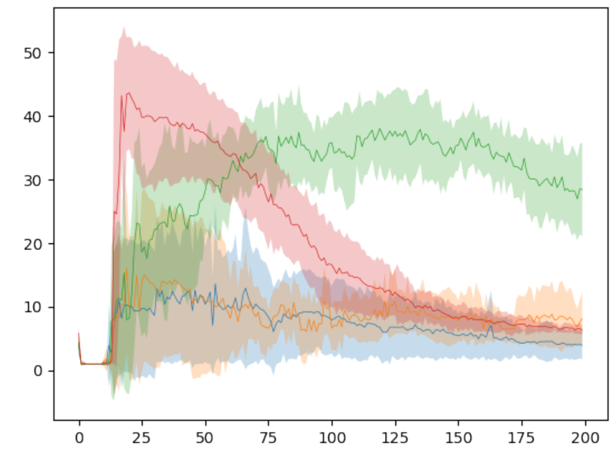


(e) Resulting entropy

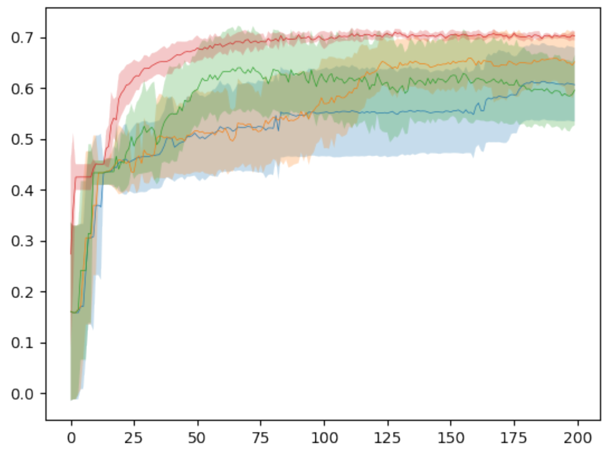
Figure A.3: Results of the Visits-count strategy with SimHash size $s = 6$ on the Forest dataset.



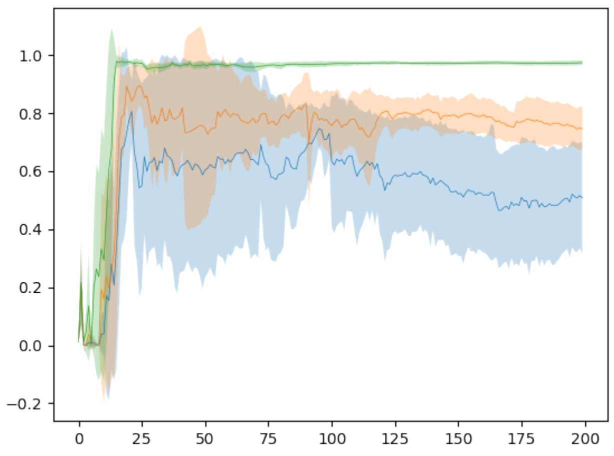
(a) Average rewards



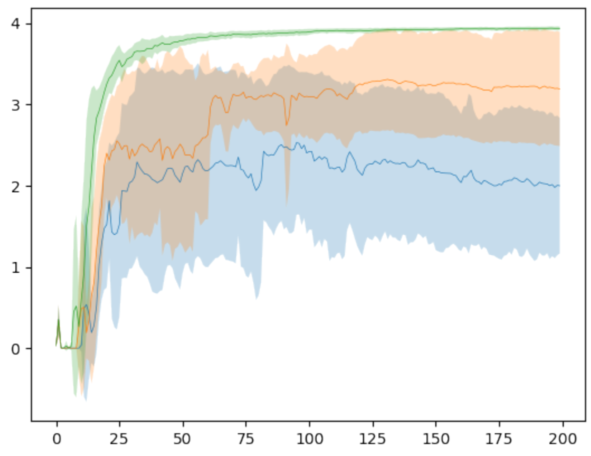
(b) Average length of trails



(c) Correct classification ratio



(d) Resulting exploration



(e) Resulting entropy

Figure A.4: Results of the Uncertainty-driven exploration on the Forest dataset.