Abstract—Balancing exploration and exploitation is a key challenge in reinforcement learning. Currently, most approaches use uninformed exploration, where only the greedy choices are changed by the agent experience and learning but exploration does not. Here, we present an alternative approach to exploration where the agent learns the exploration value of states and actions. Unlike other methods, this is done with initialization playing little or no rule, hence depending very little on explicit prior knowledge of the problem. Moreover, we provide a generalization that emphasize the difference of the two problems to which many former methods correspond. We show that our algorithm outperforms standard methods in both bridge-problem and maze-like problems.

I. INTRODUCTION – DEFINING THE PROBLEM

A. Uninformed Exploration

Q-Learning has been a main Reinforcement Learning tool for solving Markov Decision Problems (MDPs) in the last two decades. One of the challenges in this framework is balancing between Exploration and Exploitation – the need to take new actions and visit new states on the one hand, and the need to choose valuable actions that are expected to result with a high reward on the other hand. A widely used method to deal with this problem is the $\epsilon$-greedy schema [9], in which the agent greedily chooses the best action with probability $1 - \epsilon$, and a random action with probability $\epsilon$ to encourage exploration.

It is reasonable to believe that at a certain point the agent is confident about some actions more than about other, for example, when only some routes have been searched extensively. Even at this point greedy exploration chooses a random action, not using the information gained to evaluate which action should be further explored. This phenomenon is a major drawback showing why information gained during training should be considered not only for greedy action choosing, but also for more informed exploration.

Here we aim to deal with this issue, namely how to take advantage of the information gained by the agent during its training to improve exploration.

B. Uninformed Initialization

The $Q$ Learning algorithm allows for any initialization of the $Q$ values, and although some methods that use this fact[12] were suggested, they require prior knowledge regarding the specific problem, and the more commonly used method is to initialize all $Q$ values arbitrarily, usually to 0. Such initialization makes the algorithm sensitive to reward transformations for which we wouldn’t expect any change. Moreover, if the initial value is not 0, the learning process will even be sensitive for positive-scaling transformations ($r \mapsto \alpha r$) which forms an isomorphic MDP (see Appendix). While this doesn’t affect the algorithm asymptotic performance, it could have a significant effect in practice. For example, consider the following toy problem with $S$ as the initial state and $T_1, T_2$ as terminal states. Assume that the first step is randomly chosen to be $a_1$, and consider the following choices in the next episodes. If $R(T_1) = 1$ and $R(T_2) = 2$, after visiting $T_1$, the agent will greedily keep choosing $a_1$, and will only explore $a_2$ with probability $\frac{\epsilon}{2}$. However, if $R(T_1) = -2$ and $R(T_2) = -1$ then the agent will greedily choose $a_2$ as its next action. Clearly $a_2$ is the optimal policy in both problems (which are really the same problem), but it will take longer for the agent to discover it when $R(T_2) = 2$.

We can consider the initial $Q$ values as the agent’s prior knowledge or beliefs about the world[7]. This means higher initial values will drive the agent toward exploration – even as its greedy choices – since it believes better reward exist than what seen so far. Similarly, lower values will drive the agent to less exploratory behavior. While this might seem minuscule in the above toy example, it could prove
to be a problem in bigger problems, given a limited number of train episodes.

C. Bridge Problem as a Case Study

As a case study, we use the following MDP as described in figure 1. The problem consists of two “positive” terminal states separated by a bridge surrounded by “negative” terminal states. The optimal policy is crossing the bridge to the rightmost terminal state. This problem poses a great challenge for ϵ-greedy schema, even though the values for actions leading over the cliff are learned relatively fast.

In order to find the optimal policy, the agent must complete the path to the rightmost terminal state at least once, and then propagate the learned value back by repeating the intermediate states in this path.

For finding the optimal policy (i.e. go over the bridge), the agent must complete at least one path ending at the rightmost terminal state.

Before we have reached this state once, there are no greedy choices of paths/steps that lead to it, because all paths seen either end in the left state (+1) or in the pits (-100). So in order to learn anything about this path, we could analyze the case in which ϵ = 1, where moves are chosen uniformly at random, as an upper bound on the probability of the general case.

With ϵ = 1, the chance of a path ending in the rightmost terminal, is bounded by 0.25^5 – this is the probability for a path that terminates in 5 steps or less, and every longer path has lower probability of terminating in this state. So, the chance for not completing such path is at least 1 − 0.25^5. For 50 episodes, we have (1 − 0.25^5)^50 = 0.952, so the probability to complete such path in 50 episodes is 1 − 0.952 = 0.048.

Note that the probability to actually find the optimal policy is actually (much) lower, since the agent must complete paths to all of the intermediate states in order to propagate the learned Q values back to the initial state.

II. APPROACH AND METHODS

A. Softmax

To better balance between exploration and exploitation, we might want to let the learned Q values affect which actions should be explored. Actions which have already low values, should be rarely further explored, especially during the later parts of the training.

One way to achieve this is^1 to select actions drawn from a Boltzmann Distribution (also known as Softmax) defined by the Q values, and regulated by a Temperature parameter. Formally, let t be the temperature of the agent, and let [a1, ..., an] be the available actions for state s. We define a distribution over the actions by

$$\forall i \in [n] \quad p_i = \frac{\exp \left( \frac{Q(s,a_i)}{t} \right)}{\sum_{j=1}^{n} \exp \left( \frac{Q(s,a_j)}{t} \right)}$$

Then at each step, the agent choses an action based on this distribution.

Note that when t → ∞ then p_i → \frac{1}{n} and all actions have the same probability to be chosen. Alternatively, when t → 0 then the most valuable option is chosen with probability 1. In this way, we can start with a high temperature and decrease it through the training episodes to encourage more exploratory behavior during early training.

Because it could be guaranteed that each state will be visited infinitely many times, as the number of train episodes approaches infinity, this modified Q Learning algorithm will also converge to the optimal policy[11].

While this approach leads to more informed exploration, based on learning experience, it is still affected by the uninformed initialization with respect to the reward scale. A potential way to overcome this is to initialize Q values not to 0 but closer to the maximum reward of the problem (if known), to further encourage exploration.

B. λ-confidence

Ideally, we would like to estimate for each pair (S,a) not only the expected reward value (as represented and learned by Q(S,a)) but also higher moments such as the variance. With this information we could better select actions, since its possible to evaluate not only the reward from a given action, but also how certain (or uncertain) this reward is, hence how valuable it is to further explore this action. Unfortunately, even for the second moment (variance), calculating a stable estimator will require storing all Q-values seen and updated so far and iterating through them.

Consider the previously discussed Softmax method. One might view its choices as balancing exploration

^1We first thought of this approach while trying to analyze the problem, but of course we weren’t the first. This is a common approach in Q Learning, for example see [2], [9]
and exploitation using only the first moment – the expected reward value. It is obvious how the expected reward estimation is used to regulate exploitation, but how this value is used to regulate exploration is less obvious. Here is where the initial Q values become significant, because the algorithm uses these values as a prior knowledge about the problem. The further the current value of a state-action pair is from the prior, the more confident the agent is in this estimation. This holds for low and high (relative to the prior) values alike, so high valued state-action pairs could be exploited with high confidence in the outcome, and on the other hand low valued pairs should be avoided from being explored, with high confidence that there is no reason to further explore them. This approach could be better than fully random exploration, as in the naive $\epsilon$-greedy scheme, but in order for it to perform well it depends on an appropriate setting of the prior knowledge (Q values initialization), which may prove difficult to achieve. Similar problems arise in other Reinforcement Learning settings[4], [6], and we believe further investigation is required here as well.

To overcome these difficulties, and move from uninformed to informed exploration, we could use different heuristics to evaluate for each state-action pair ($s, a$) how valuable it is to explore it, and take this value into consideration when choosing an action. Formally, we generalize the problem as having a function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ that evaluates the expected reward values of state-action pairs, and a function $E : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ that evaluate the need in exploration for such pairs. Then, action selection is based on some utility function $U : \mathbb{R}^2 \to \mathbb{R}$ combining both functions in some desired way. Simply put, action selection is made (either in a deterministic or probabilistic way) according to $U_\theta (Q(s, a), E(s, a))$ where $\theta$ is a parameter.

Different selections of $E$ and $U$ were suggested. Perhaps the most straightforward is counting[8] – for each such pair ($s, a$), store a counter $c(s, a)$ that indicates how many times the agent performed action $a$ at state $s$ so far. This way, an exploration bonus could be given to state-action pairs with low $c(s, a)$ values. Other options to evaluate the need in exploration is with recency measure[8] – where $r(s, a)$ indicates how many steps the agent performed since last performing action $a$ at state $s$. Intuitively, pairs that were last observed a long time ago might require re-visiting for better evaluation, especially if their last visit was in the early part of the training. Finally, we could use a value difference (or error) measure[8], [9], based on the magnitude of the last update to $Q(s, a)$ in the learning process – for state-actions pairs for which we changed our utility estimation significantly, the current estimated value
might be still far from the actual value, hence more exploration is required.

All of the above methods might provide improvement from a naive \(\epsilon\)-greedy approach, but they still suffer from a major drawback, namely, they only consider the “exploration value” of the immediate, temporal action (or more precisely, state-action pair)\(^2\). This “exploration value” stands for the amount of information gained by taking a temporal action. But in real problems, the desired exploration value of a temporal action depends not only on the immediate amount of information it yields, but also on the expected amount of information (potentially, the discounted amount) that the agent will gain from this action to the end of the episode.

Following this analysis, we notice that the function \(E\) should approximate a function \(E^* : S \times A \rightarrow \mathbb{R}\) of the general form:

\[
E^* (s, a) = \inf_{s'} (s, a) + \gamma \cdot \max_{a'} \mathbb{E} [E^* (s', a') | s, a]
\]

So, in fact, we have two separate and complementary problems – the exploitation problem of finding an optimal policy, implemented by learning \(Q^*\), and the exploration problem of finding most information about the world, which we would like to implement by learning \(E^*\). Moreover, the definition of both functions is of the same general form, which naturally suggests using \(Q\) Learning in order to learn \(E\) itself. However, one major difference must be taken into consideration, as the exploration problem does not have the Markov property: the “reward”, or in this case the new information we gain by visiting a state, depends on the history so far. Obviously visiting a state after we already visited it many times results with much less “information reward” than visiting it for the first time.

Trying to overcome this problem, we suggest constructing another MDP to account for the exploration problem. The new problem is identical to the original problem’s dynamic (the probability to transfer from state \(s\) to state \(s'\) by performing action \(a\)), and it is not obvious how to generalize this for a model-free context of \(Q\) Learning. And even here, only one step ahead is considered.

\(\text{Algorithm 1} \lambda \text{ confidence}\)

1) Parameters: \(\alpha, \alpha_\varepsilon, \gamma, \gamma_\varepsilon, \lambda\)
2) Initialize \(Q (s, a) = 0, E (s, a) = 1\) for all \(s, a\)
3) For each episode:
   a) Set \(s\) as a start state
   b) While \(s\) is not terminal:
      i) \(a = \arg \max_a (\lambda Q (s, a) + (1 - \lambda) E (s, a))\)
      ii) Perform \(a\), observe reward \(r\) and next state \(s'\)
      iii) Update:
      \[
      \Delta Q (s, a) = \alpha \left( r + \gamma \max_{a'} Q (a', s') - Q (s, a) \right)
      \]
      \[
      \Delta E (s, a) = \alpha_\varepsilon \left( \gamma_\varepsilon \max_{a'} E (a', s') - E (s, a) \right)
      \]
   c) Decay \(\alpha, \alpha_\varepsilon\), and increase \(\lambda\)

There are a few properties of this algorithm worth noticing. In terms of both time and space complexity, the algorithm behaves asymptotically as the original \(Q\) Learning algorithm. In terms of its concrete specifications, observe first that although we defined learning \(E\) as a \(Q\)-Learning scheme, it is possible to use other Reinforcement Learning algorithms, such as SARSA or TD Learning as well, with only minor adjustments. In addition, following the general framework presented earlier, other choices for \(U\) function could be easily implemented for differently combining \(E\) and \(Q\) values. The main advantage for the suggested \(U_\lambda\) function is its scalability tolerance – with the correct choices of \(\lambda\) and its decaying rate, we could neutralize the sensitivity for irrelevant reward transformations\([5]\) without explicit prior knowledge about the problem. Finally, action selection could be made in a probabilistic way according to \(U\) values, for example using Softmax rather than taking argmax. However, this might be redundant as the current deterministic choice is not purely greedy or exploratory, but greedy according to the weighted combination of both problems. Therefore, compensating greediness with probabilistic action choices might be unnecessary in practice. Still, stochastic
action selection might be desired in order to grant asymptotic convergence, by guaranteeing the visit of any state infinitely many times – this is sufficient since with infinitely many episodes, \( \lambda \to 1 \) and \( E(s,a) \to 0 \) so we get a standard Q-Learning behavior.

We now turn to present empirical results, comparing behavior and performance of the different algorithms discussed above, and demonstrating their properties in practice.

## III. Experiments and Results

### A. Method

We wished to prove that in practice our proposed methods do indeed solve hard exploration problems and outperform \( \epsilon \)-greedy, and for that purpose we use the aforementioned bridge problem. A second set of experiments were conducted to make sure our solutions were not an overfit solution to the bridge problem but a real improvement in general case complex MDP, for that we chose two Pacman mazes.

In all of these experiments, the learning rates and discount factors were the same in all of the algorithms and did not change during training.

In our experiments and simulations, we used the Berkeley AI Pacman Projects framework\[3\] providing the underlying engine for Gridworld and Pacman.

### B. GridWorld

We used the same MDP as described in figure 1. For these series of tests, we used \( \alpha = 0.5, \gamma = 0.9 \) as constants for all of the algorithms and for \( \lambda \)-confidence they were used for the exploration problem as well. The hyperparameters were chosen by hand, and because of the high performance were actually not that important. For \( \lambda \)-confidence, we set initial \( \lambda = 0.01 \) and increase by \( \lambda_{n+1} = 1.01\lambda_n \). For Softmax, we set initial \( t = \frac{1}{0.05} \) and decrease by \( t_{n+1} = \frac{1}{1.005} t_n \). The examples show characteristic results of the algorithms performance – for 50 episodes of training (figure 2), and until convergence (figure 3).

As demonstrated by the results, \( \lambda \)-confidence outperforms both \( \epsilon \)-greedy and Softmax in terms of required training episodes before convergence. Also note that since for all algorithms the same learning rate was used, the accuracy of estimated values for terminal states indicates the number of times these states were visited during training. For example, \( \lambda \)-confidence only visited each of the “pit states” twice, while the other algorithms continued to further explore those states.

1) \textit{Initialization and its Effects}: In I-B, we discussed the effect of the initial \( Q \) values, representing a “prior knowledge” about the problem, on the learning process. Figure 4 demonstrates this principle in two ways. First, note that higher initial \( Q \) values – optimistic prior knowledge or belief – drives to a more exploratory behavior, which in this problem leads to faster convergence to finding the optimal policy (figure 4a). On the other hand, lower initial \( Q \) values – pessimistic prior knowledge or belief – drives to a more cautious behavior, which in this problem ends in catastrophic results (figure 4b).

Note that it is not always the case that this exploratory behavior is optimal, for example in an \( n \)-armed bandit problem where the number of allowed episodes is less than \( n \) [10]. Moreover, in other problems, exploratory behavior might be just avoiding terminal states, thus gaining more training data from each episode. Although this might show better results with limited number of episodes, it might be undesirable when training times and not training episodes is the main cost. This factor does not seem to be well documented but might prove to be very important in practice.

### C. Pacman

1) \textit{Small}: We saw that the suggested \( \lambda \)-confidence schema do indeed outperform \( \epsilon \)-greedy on the bridge problem, suggesting at least an improvement in certain aspects of exploration. We also conducted a series of experiments showing that at a general, more complex MDPs this schema works at least as well as \( \epsilon \)-greedy.

![Figure 6: Pacman Mazes](image)

For that we chose a Pacman maze (figure 6a), where each state is given by the position of the agent
Figure 2: Comparison of learning algorithms after 50 episodes of training
(a) \( \lambda \)-confidence after 50 episodes training
(b) Softmax after 50 episodes training
(c) \( \epsilon \)-greedy after 50 episodes

Figure 3: Comparison of learning algorithms convergence
(a) \( \lambda \)-confidence after 30 episodes training
(b) Softmax after 100 episodes training
(c) \( \epsilon \)-greedy after 2500 episodes

Numbers indicate the estimated \( V(s) \) based on current \( Q \)-values. Arrows indicate current policy. Best view in color

Figure 4: Softmax algorithm with different initialization (prior knowledge)
(a) \( Q_0(s,a) = 9 \)
(b) \( Q_0(s,a) = -50 \)

(Pacman), the position of the ghost(s) and the position of remaining food. The reward for eating is 10, and for solving the maze (eating all food points) is 500. In addition, being eaten by a ghost is a terminal state and has a reward of \(-500\). The ghost is using a random function to choose which way to go, making the MDP probabilistic, each \((s,a)\) might lead to between 1 and 4 states. Finally, there is a constant “living penalty” of \(-1\) for each non-terminal state.

As with the GridWorld problem, we’ve used learning rate and discount factor \( \alpha = \alpha_r = 0.5, \gamma = \gamma_r = 0.9 \). The other parameters were also left unchanged (For \( \lambda \)-confidence, we set initial \( \lambda = 0.01 \) and increase by \( \lambda_{n+1} \leftarrow 1.01 \lambda_n \). For Softmax, we set initial \( \beta = \frac{1}{0.05} \) and decrease by \( \beta_{n+1} \leftarrow \frac{1}{1005} \cdot \beta_n \)). After the train episodes, learning is stopped and the agent execute the optimal policy based on the learned \( Q \) values. It is noteworthy that it was not due to lack of thought that the hyperparameters were left unchanged, we wanted to see how these parameters, not even tailored well for the Bridge problem, generalize to other problems, or in other words how hard it is to choose satisfying
parameters and how stable they are.

As demonstrated in figure 5, both Softmax and the deterministic $\lambda$-confidence algorithms outperformed $\epsilon$-greedy in this problem\(^3\). In particular, this shows that the suggested schema is not overfitted to “pure exploration” problems such as the bridge problem discussed earlier.

2) Medium: After finding out that the methods manage to learn the small Pacman maze after relatively small number of train episodes, we tested them with a more challenging maze, to which $\epsilon$-greedy is far from learning the optimal policy. On that maze (figure 6b) we used training consists of 3000 episodes and tested the winning rate out of 50 test episodes in which only the optimal policy was used.

This time, we introduced a stochastic version of the $\lambda$-confidence algorithm, where action selection is drawn from Softmax distribution instead of taking $\arg\max$ deterministically. The algorithm uses $\lambda$ parameter to regulate the combination between $Q$ and $U$ values and $\beta$ parameter (temperature) to regulate the Softmax distribution. Note that for $\lambda = 1$ the algorithm reduces down to a regular Softmax, and similarly for $t \to 0$ the algorithm reduces down to the deterministic $\lambda$-confidence schema.

We aimed to maximize the algorithms performance, so we chose the hyperparameters more carefully by searching through the parameters space. Our hypothesis was that given a good choice of the hyperparameters for the deterministic $\lambda$-confidence algorithm, adding stochastic actions selection with Softmax should not have a major impact.

However, our results show that the stochastic softmax-$\lambda$ combination does outperform both Softmax and $\lambda$-confidence in this problem. Moreover, the deterministic $\lambda$-confidence outperforms Softmax algorithm, and all three methods gained major improvement over $\epsilon$-greedy. Further work is still needed in order to better understand the contribution of stochastic action selection in the $\lambda$-confidence schema. Albeit, some initial conclusions could still be drawn based on figure 7, for example it seems that Softmax and Soft-$\lambda$ are more sensitive to hyperparameter choices, or have higher variance in performance.

The performance details of each algorithm is listed in table I. Note that due to different number of hyperparameters, each algorithm was tested different number of times.

![Figure 5: Average winning rate on 100 test episodes as a function of train episodes](image)

\(^3\)The max number of train episodes was 500, which was sufficient for the $\epsilon$-greedy algorithm to gain near-perfect performance. Note that in the original exercise, we were requested to train the agent for 2000 episodes – by this time the expected reward for a train episode is around 500 (maximum score). This, however, is misleading as the optimal policy is learned before that, as displayed in the figure.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Best</th>
<th>Top 5% average</th>
<th>Total runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>random</td>
<td>0.035</td>
<td>-</td>
<td>4</td>
</tr>
<tr>
<td>$\epsilon$-greedy</td>
<td>0.255</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>Softmax</td>
<td>0.595</td>
<td>0.58</td>
<td>61</td>
</tr>
<tr>
<td>$\lambda$-confidence</td>
<td>0.74</td>
<td>0.635</td>
<td>56</td>
</tr>
<tr>
<td>Soft-$\lambda$-confidence</td>
<td>0.84</td>
<td>0.68</td>
<td>1637</td>
</tr>
</tbody>
</table>

Table I: Comparison of the different algorithms on Pacman medium-grid
IV. FURTHER WORK

We introduced several methods to enable informed exploration in reinforcement learning problem. However, this is just the beginning and further analysis is required. One aspect of this analysis would be better understanding of the learning parameters for the exploration problem – learning rate and discount factor. Because the reward is fixed, we can control the decrease rate of $E$, meaning for how many explorations are required for a state to have no exploration value. Another factor we can control is how much far away states will contribute to the exploration value. This might be done using analytic methods, if we could estimate number of the problem’s properties, such as the number of states and expected episode length.

In addition, regarding the stochastic soft-λ algorithm, better understanding of the contribution each of its components – Softmax and λ-confidence – provide is still required. For example, our results suggest that λ should be raised slowly, regardless of its initial value, for better learning.

Another important aspect that still requires further investigation is reward-shaping methods[1], their connection to our generalization and their similarities and differences from λ-confidence suggested here.

Finally, other approximations or learning schemas for $E^*$ could be tested, for example learning $E$ with TD algorithm rather than $Q$-learning.

APPENDIX

Lemma 1. Let $\mathcal{M}_1$ be an MDP for which the reward is given by (maybe stochastic) function $r_1 (s,a)$, and a copy of this MDP when we only change the reward $r_2 = \alpha r_1$. Then we have $Q^*_2 (s,a) = \alpha Q^*_2 (s,a)$.

Proof: By definition,

$$Q^*_1 (s,a) = \mathbb{E} [r_1 (s,a)] + \gamma \max_{a'} \mathbb{E}_{s'} [Q^*_1 (s',a')]$$

$$Q^*_2 (s,a) = \mathbb{E} [r_2 (s,a)] + \gamma \max_{a'} \mathbb{E}_{s'} [Q^*_2 (s',a')]$$

$$= \alpha \mathbb{E} [r_1 (s,a)] + \gamma \max_{a'} \mathbb{E}_{s'} [Q^*_2 (s',a')]$$

$$= \alpha \mathbb{E} [r_1 (s,a)] + \gamma \max_{a'} \mathbb{E}_{s'} [Q^*_2 (s',a')]$$

$$= \gamma \left( \alpha \mathbb{E} [r_1 (s',a')] + \gamma \max_{a''} \mathbb{E}_{s''} [Q^*_2 (s'',a'')] \right)$$

$$= \alpha (Q^*_1 (s,a))$$
Since $\pi^*(s) = \arg \max_a Q^*(s, a)$, it follows from lemma 1 that for any $\alpha > 0$, a reward transformation of scaling by $\alpha$ preserves the optimal policy, and moreover preserve the order relation between $Q$-values and $V(s)$ values. In other words, such transformations create an isomorphic copy of the original problem.

REFERENCES