Study of Entropy Regularized Reinforcement Learning Algorithms

Veronika Shilova
École Polytechnique
veronika.shilova@polytechnique.edu

Zheng Ge
École Polytechnique
zheng.ge@polytechnique.edu

Abstract

Reinforcement learning (RL) has shown great potential in solving complex decision-making problems. However, there are still major challenges in RL such as the instability of RL algorithm and the exploration-exploitation dilemma. In recent years, entropy regularization has emerged as a promising technique to address these issues. In this project, we discuss a number of entropy regularizated algorithms. We also present experimental results demonstrating the effectiveness of entropy regularization in various classical RL tasks, such as Frozen lake and Cart-pole.

1 Introduction

Reinforcement Learning (RL) is getting more and more popular in recent years. However, despite its present advances, there are still a number of problems that need to be dealt with. For instance, there is still an issue of finding good exploration/exploitation balance and, moreover, RL methods are very unstable even on the simplest tasks.

To tackle these challenges, we can use entropy regularization techniques. Entropy Regularized Reinforcement Learning Algorithms is a class of reinforcement learning algorithms that seek to balance between exploration and exploitation. In traditional reinforcement learning, agents are trained to maximize their cumulative reward over time. However, this can lead to the agent being overly focused on exploiting the actions that have already yielded high rewards, and not exploring enough to discover potentially better actions.

To address this, entropy regularization is introduced into the objective function of the agent. Entropy is a measure of the unpredictability of a random variable, and in reinforcement learning, it can be used as a measure of the exploration of the agent. By adding a term to the objective function that encourages the agent to take actions that have high entropy, the agent is incentivized to explore more.

The entropy regularizated algorithms are introduced in the papers [1], [2] and [3]. The study of them consitutes the main subject of our project.

In this work, we review and compare all previously mentioned algorithms. Furthermore, we implement Politex algorithm from [1], benchmark it on two classical RL environments: Frozen lake and Cart-pole; and compare it to SARSA and Q-learning. We also study an influence of hyperparameters on Politex performance and propose a variant of Politex with Neural Networks.

2 Algorithms

In this section, we present three main algorithm that were studied and compared either theoretically or practically or both during this project.

MAP/INF641 - Reinforcement Learning (2022-2023), Palaiseau, France.

2.1 Politex: Regret Bounds with Expert

In the paper [1], the authors present a novel algorithm called POLicy ITeration with EXpert advice or, in short, Politex. It is a policy iteration variation in which each policy is a Boltzmann distribution over the sum of action-value function estimates of the prior policies. An expert algorithm, in this case, is the exponentially-weighted average forecaster. The authors claim to offer the first regret bound for a completely practical model-free technique that solely scales in terms of feature count rather than MDP size.

To model the interaction between the agent and the environment, consider Markov decison process (MDP). It given by a tuple $(\mathcal{X}, \mathcal{A}, c, P)$, where $\mathcal{X}$ is a finite state space of cardinality S; $\mathcal{A}$ is a finite action space of carinality A, c is cost function, and P is the transition probability distribution that maps each state-action pair to a distribution over the states. The agent receives the state of the environment $x_t \in \mathcal{X}$ and chooses an action $a_t \in A$ and suffer a cost of $c(x_t, a_t)$. Initially, the agent does not know P and c. We also assume that x_1, the initial state, is chosen at random from some unknown distribution.

A policy is a mapping $\pi : \mathcal{X} \to \Delta_{\mathcal{A}}$ from a state to a distribution over actions. Following a policy means that in any time step, upon receiving state x, an action $a \in \mathcal{A}$ is chosen with probability $\pi(a|x)$.

The proposed algorithm, Politex is shown in Algorithm listing 1. In each phase i, POLITEX executes policy π_i and at the end of the phase computes an estimate $\hat{Q}_i$ of Q_{π_i}, the state-action value function of π_i (a policy evaluation step). The next policy is a Boltzmann distribution over the sum of all past state-action value estimates:

$$\pi_{i+1}(a|x) \propto exp\Big(\eta \sum_{j=1}^{i} \hat{Q}_j(x,a)\Big), \tag{1}$$

where $\eta > 0$ is a, so called, softmax temperature.

Algorithm 1 POLITEX: POLicy ITeration using EXperts

Input: phase length $\tau > 0$, initial state x_0
  Set $\hat{Q}_0(x,a) = 0 \; \forall x, a$
  for $i := 1, 2, ...,$ **do**
    Set $\pi_i(a|x) \propto exp\Big(\eta \sum_{j=0}^{i-1} \hat{Q}_j(x,a)\Big)$
    Execute π_i for τ time steps and collect data

$$\mathcal{Z}_i = \{(x_t, a_t, c_t, x_{t+1})\}_{t=\tau(i-1)+1}^{\tau i}$$

    Compute $\hat{Q}_i$ from $\mathcal{Z}_1, ..., \mathcal{Z}_i$, $\pi_1, ..., \pi_i$
  end for

Politex can be viewed as a "softened" and "averaged" version of policy iteration. Averaging reduces noise, allowing for noisier estimates $(\hat{Q}_j)_j$ and thus switching to a new policy faster, while the exponential weighting increases robustness.

The authors leave the choice of how $\hat{Q}_i$ is estimated to the user (thus, Politex is better viewed as a learning schema).They let the user choose how to estimate the $\hat{Q}_i$ and expect a longer phase lengths τ to lead to better estimates. Although Algorithm 1 suggests that all data should be gathered and reserved, but this is only to clarify which data can be used to estimate the $\hat{Q}_i$. In reality, it is feasible to apply any incremental algorithm (e.g., TD-learning, as in [4]). Similarly, one can also use policy-based methods (thus, restricting the use for estimating $\hat{Q}_i$ to $\hat{Z}_i$), or use an off-policy approach.

2.2 Improved Regret Bound and Experience Replay

In the paper [2], the authors presents another entropy regularized algorithm which improves on the drawbacks of Politex. They study algorithms for learning in infinite-horizon undiscounted Markov

decision processes (MDPs) with function approximation. The paper demonstrate that, under roughly similar assumptions, the regret analysis of the Politex algorithm can be further sharpened.

The authors argue that in terms of memory and computation, the accurate implementation of Politex with neural network function approximation is inefficient. They propose a straightforward and effective solution where they train a single Q-function on a replay buffer with historical experience because. The result of analysis indicates that it must accurately approximate the average of the action-value functions of previous policies. They demonstrate that this approach frequently results in better performance than alternative implementation options.

The work targets the actual application of Politex using neural networks. Each iteration of Politex's policy needs access to the total of all past Q-function estimates. Exact implementation of neural network function approximation necessitates keeping all previous networks in memory and evaluating them at each step, which is wasteful in terms of memory and processing. They suggest an alternative strategy in which a single network is trained on a replay buffer of historical data to approximate the average of all prior Q-functions. Moreover, the research offers a fresh viewpoint on the advantages of experience replay. Experience replay is a commonly utilized method in off-policy techniques like Deep Q-Networks for stabilizing learning in contemporary deep RL. They point out that Politex can be roughly implemented using experience replay and KL-divergence regularization. The conventional argument for experience replay is that it "breaks temporal correlations". Their research also recommends a new goal for transitions between priority-sampling and subsampling in the replay buffer.

This paper's algorithm is comparable to the Politex schema. Every phase k, Politex estimates the action-value function $Q_{\pi k}$ of the current policy π_k, and next step, using the mirror descent updating rule, sets the next policy:

$$\begin{aligned}\pi_{k+1}(\cdot|x) &= \arg\max_{\pi} \hat{Q}_{\pi_k}(x,\pi) - \eta^{-1} D_{KL}(\pi||\pi_k(\cdot|x)) \\ &= \arg\max_{\pi} \sum_{i=1}^{k} \hat{Q}_{\pi_i}(x,\pi) + \eta^{-1}\mathcal{H}(\pi) \\ &\propto exp\Big(\eta \sum_{i=1}^{k} \hat{Q}_{\pi_i}(x,\cdot)\Big),\end{aligned} \tag{2}$$

where the entropy function is $\mathcal{H}(\cdot)$. The aforementioned update can be efficiently accomplished when the functions $(\hat{Q}_{\pi i})_{i=1}^{k}$ are tabular or linear by simply summing all the table entries or weight vectors. Nevertheless, using neural network function approximation, all networks must be kept in memory and evaluated at each step, which quickly degrades storage and processing efficiency. The authors attempt to directly approximate the average of all prior action-value functions

$$Q_k(x,a) := \frac{1}{k} \sum_{i=1}^{k} Q_{\pi_i}(x,a), \tag{3}$$

which is shown in Algorithm listing 2.

In order to do this, they employ one small network and train it to approximate Q_k. Then they get a dataset of tuples $D_k = (x_t, a_t, R_t)$ at each iteration k, where R_t is the empirical return from the state-action pair (x_t, a_t). By reducing the squared error over the union of D_k and the replay buffer R_t, they initialize Q_k to Q_{k-1} and update it. The replay buffer is then updated with all or a portion of the data in D_k.

2.3 Entropy Regularized RL with Cascade-NN

In paper [3], the authors argue that deep reinforcement learning has proven to be extremely effective on high-dimensional problems, but its learning process is unstable even on the simplest tasks. Neural networks are used as function approximators in deep reinforcement learning. The development of the (un)supervised machine learning community was a major inspiration for these neural

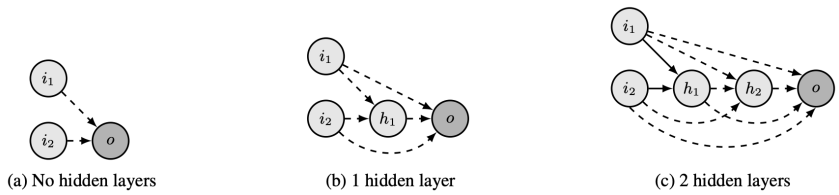
Algorithm 2 Schema for policy iteration with replay

Input: phase length $\tau > 0$, num. phases K, parameter η
  Set $\pi_1(a|x) = 1/|\mathcal{A}|$, empty replay buffer $\mathcal{R}$
  for $k = 1, ..., K$ **do**
    Execute π_k for τ time steps and collect data $\mathcal{D}_k$
    Compute $\hat{Q}_k$, an estimate of $Q_k(x, a) = \frac{1}{k}\sum_{i=1}^{k} Q_{\pi_i}$, from data $\mathcal{D}_k$ and replay $\mathcal{R}$
    Set $\pi_{k+1}(a|x) \propto \exp(\eta k \hat{Q}_k(x, a))$
    Update replay buffer $\mathcal{R}$ with $\mathcal{D}_k$
  end for
Ouput: π_{k+1}

models. The lack of data is one of the main challenges of RL in comparison to these learning frameworks. One approach to dealing with this difficulty. Controlling the rate of change of the strategy per iteration is one way to deal with this difficulty. The authors' work challenges the (un)supervised learning community's common practice of using a fixed neural structure. Instead they build a neural model that grows with each policy update. This enables closed form entropy regularization of policy updates, resulting in better control of the policy change rate in each iteration and assisting in dealing with the non-instantaneous nature of RL. Preliminary results on the classical RL baseline are promising, with significant convergence on some RL tasks when compared to other deep RL baselines. Some RL tasks show significant convergence when compared to other deep RL baselines, while they show limitations.

RL considers the problem of finding the optimal policy in the MDP's unknown environment. One way of doing so is to use policy iteration methods that successively perform i) a policy evaluation step to compute Q_π and ii) a policy improvement step, yielding a new policy π' that picks at every state s an action in $\arg\max_a Q_\pi(s, a)$. Repeatedly performing these two steps will converge in finite time to an optimal policy, irrespective of the choice of the initial policy. However, the policy π' obtained after a policy improvement step does not explore actions that do not maximize the previous Q-function $Q_\pi(s,.)$, for every state s. Hence, to correctly estimate $Q_{\pi'}(s,.)$ for such actions in practice, one would need to introduce another data gathering policy that is more explorative than π'. An alternative is to regularize the policy update step to maintain the stochasticity of π'. This is usually performed by adding a KL-divergence term between π and π', ensuring that exploration does not vanish too quickly.

However, the authors of this paper investigate solving the soft policy update exactly for every state, using an incrementally increasing neural architecture which is similar to Politex. Thus, they show that the entropy regularized policy update can be solved in closed form, and policies have a very simple form, provided we can keep track of all previously estimated Q-functions. However, they will not train independent neural networks (as with Politex) for each $\hat{Q}_{\pi_k}$, but leverage the Cascade-NN architecture and only add a few neurons at every iteration to learn $\delta_k = \hat{Q}_{\pi_k} - \hat{Q}_{\pi_{k-1}}$. Cascade-NN architecture was first introduced in the following paper [5].



(a) No hidden layers (b) 1 hidden layer (c) 2 hidden layers

Figure 1: Incremental growth in Cascade-NN architecture and freezing of weights. Dashed lines show learnable parameters after each addition, while solid lines represent frozen weights. In a Cascade-NN, when new hidden nodes are added, all inputs to older nodes are frozen so as to freeze older features.

They propose MICARL algorithm - a deep RL algorithm that follows the policy iteration scheme, covers Q-function and adapts Cascade-NN model. The Cascade-NN has two heads, one storing the last Q-function-used to compute the targets of the neural fitted Q step in Algorithm listed 3, while

the output Σq accumulates all past weight matrices of node q, and is used by the softmax policy. Note that all nodes in the figure can be multi-dimensional, including hidden nodes/layers.

Algorithm 3 Pseudo code of the MicaRL Algorithm

Set $\hat{Q}_{\pi_0}$ to the zero function
for Iteration i in $\{1, ..., NB\_ITER\}$ **do**
  Collect NB_SAMP transitions of type (s, a, r, s', a') from the environment following policy $\pi_i(s) \propto exp\Big(\eta \sum_{k=0}^{i-1} \hat{Q}_{\pi_k}(s, \cdot)\Big)$
  Add n neurons to the current Cascade-NN
  Set δ^0 to the zero function
  for Epoch e in $\{1, ..., E\}$ **do**
    Compute target $r + \gamma(\hat{Q}_{\pi_{k-1}} + \delta^{e-1})(s', a') - \hat{Q}_{\pi_{k-1}}(s, a)$ for every transition (s, a, r, s', a')
    Obtain δ^e by fitting the target using stochastic gradient descent
  end for
  Set $\hat{Q}_{\pi_k} = \hat{Q}_{\pi_{k-1}} + \delta^E$
end for

3 Benchmarks

In this section, we present two environments from OpenAI Gym which we use in the following section to benchmark Politex algorithm.

3.1 Frozen Lake

Frozen Lake is a popular environment in OpenAI Gym, which simulates a gridworld game where the agent navigates a frozen lake to reach a goal while avoiding holes in the ice. The environment is represented as a 4×4 grid of tiles, each of which can be one of four types: start, frozen, hole, or goal.

At the start of each episode, the agent begins at the start tile, and the goal is to reach the goal tile without falling into any holes. The agent can take one of four actions at each time step: move up, down, left, or right. However, the ice is slippery, so the agent may not always move in the intended direction. The transition probabilities of the environment are stochastic, with a probability of 0.33 that the agent moves in the intended direction and a probability of 0.33 that the agent moves 90 degrees to the left or right of the intended direction.

The agent receives a reward of +1 if it reaches the goal tile and a reward of 0 for all other actions. If the agent falls into a hole, it receives a reward of -1 and the episode ends. The episode also ends if the agent reaches the goal tile.

The observation space in Frozen Lake is an integer representing the current tile that the agent is on. The action space is a discrete set of four actions: 0 for moving up, 1 for moving down, 2 for moving left, and 3 for moving right. Fugure 2 gives a visual description of this environment.

3.2 Cart Pole

The Cart-Pole environment is a classic control problem used in the field of reinforcement learning to test and benchmark algorithms. The environment consists of a cart that moves along a horizontal track, and a pole that is attached to the cart with a joint. The goal of the agent is to balance the pole on the cart by applying forces to the cart. The episode ends when the pole falls beyond a certain angle or the cart moves outside a certain range.

The state of the environment is represented by four continuous variables: the position and velocity of the cart, and the angle and angular velocity of the pole. The agent can take two discrete actions: move the cart left or right. The agent receives a reward of +1 for every time step the pole is balanced, and the episode ends when the pole falls beyond a certain angle or the cart moves outside a certain range. The maximum reward that an agent can achieve per episode is 500.

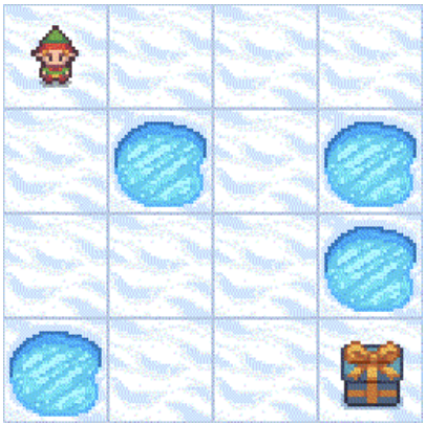


Figure 2: Frozen Lake environment.

The Cart-Pole environment is a simple yet challenging problem that requires the agent to balance a system with nonlinear dynamics. It is often used as a benchmark for testing and comparing reinforcement learning algorithms, and has been used in many research studies to evaluate the performance of different algorithms. Figure 3 gives a glimpse of how this environment looks like and table 1 contains possible values for observation space of each element in Cart-pole.

Observation	Min	Max
Cart Position	-4.8	4.8
Cart Velocity	$-\infty$	∞
Pole Angle	-.418 (radians)	.418 (radians)
Pole Angular Velocity	$-\infty$	∞

Table 1: Possible values for observation space of each element in Cart-pole

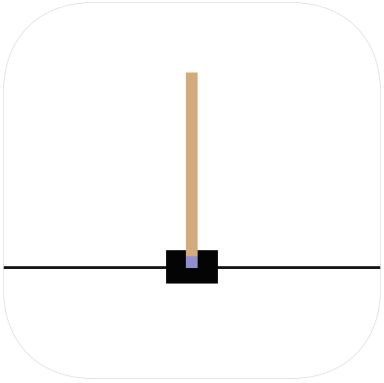


Figure 3: Cart Pole environment.

4 Experiments

In this section, we discuss different experiments performed for Politex algorithm on two distinct environments presented earlier: Frozen Lake and Cart-pole. Furthermore, we compare Politex performance with SARSA and Q-learning ones in the same settings.

In all experiments, we consider Politex algorithm in which each Q_{π_i} function is estimated using TD-learning [4] with SARSA update. We decided to choose this specific type of estimates for the

initial experiments, since it is easier to debug the code in this way. We will discuss the possibility to use Neural Networks as estimates for Q_{π_i} functions in the last subsection.

4.1 Frozen Lake

We start by benchmarking Politex, SARSA and Q-learning on a small (state-wise) and rather simple environment Frozen lake. Figures 4 and 5 show the learning and evaluation reward curves for all three algorithms. To get these results, we ran each algorithm for 10 times and ploted average reward curves, as well as the standard deviation for each one. As for the algorithms parameters, for Politex the set-up is as follows: $\eta = 1$ (softmax temperature), $\tau = 1000$ (phase length), $\gamma = 0.995$ (discount factor), $\alpha = 0.1$ (learning rate). For SARSA and Q-learning, we take $\alpha = 0.5$, $\gamma = 0.9$, $\epsilon = 1.0$ ("amount of randomness" in the action selection).

Moreover, for SARSA and Q-learning, we change ϵ parameter over time to find a reasonable trade-off between exploitation and exploration. We do this , since, at the beginning of the training, we want to explore the environment as much as possible. However, exploration becomes less and less interesting, as the agent already knows every possible state-action pairs. This parameter represents the amount of randomness in the action selection. We can decrease the value of ϵ at the end of each phase (episode) by a fixed amount (linear decay), which is in our case is equal to 0.001. Unfortunately, without this trick, the performance of both SARSA and Q-learning in this task is quite poor.

To choose all the parameters, we ran the experiments multiple times and picked those giving maximum reward values.

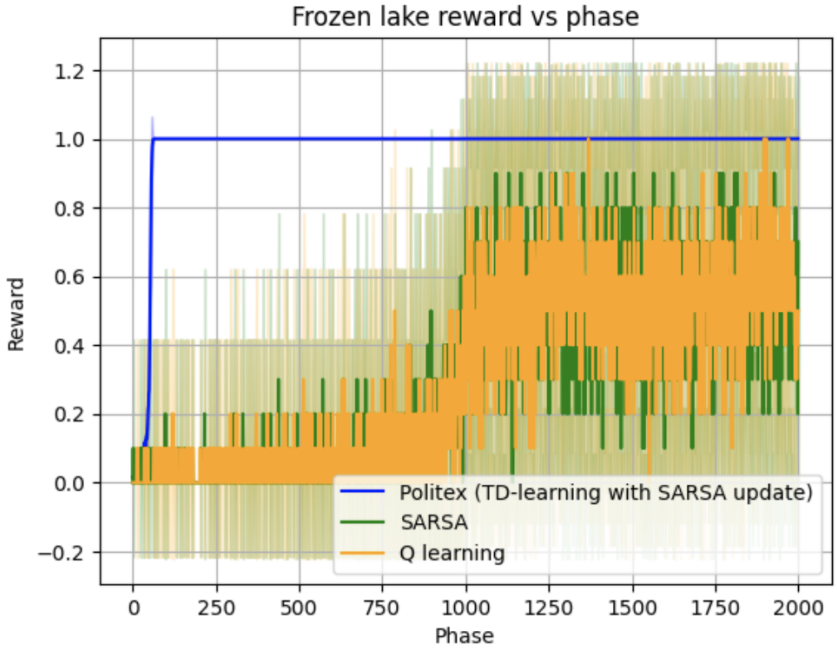


Figure 4: Experiments on the Frozen lake environment. Learning curves for Politex, SARSA and Q-learning.

From the figures 4 and 5, we notice that Politex algorithm does not need many learning phases to achive the highest score in the game: it does so at around 50 epochs. Whereas SARSA and Q-learning do not reach the goal in every phase even after 2000 phases of learning. On average, they reach it in 6 out of 10 episodes.

Furthermore, if we observe the standard deviation for all learning and evaluation reward curves, Politex has smoother curves or 0 error almost everywhere, compared to quite noise reward curves of SARSA and Q-learning.

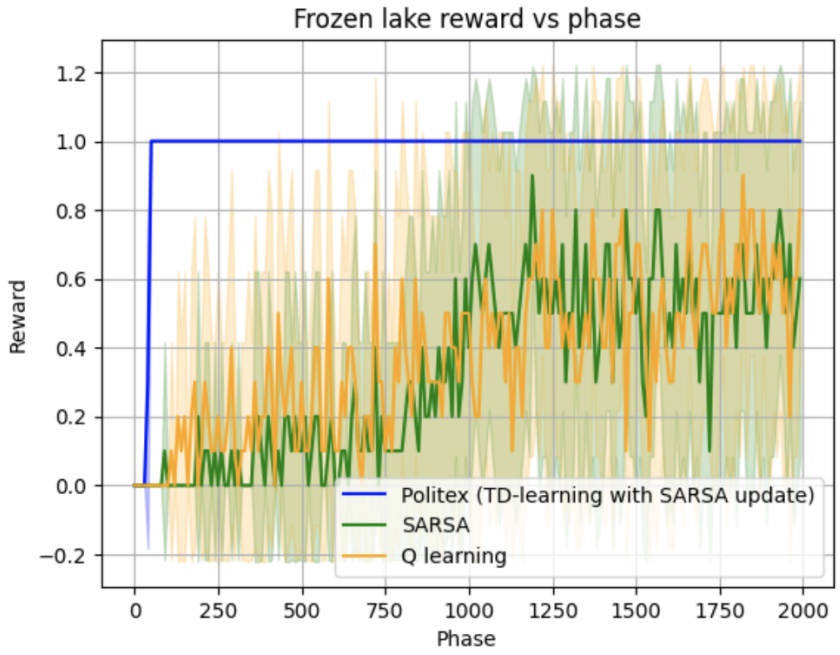


Figure 5: Experiments on the Frozen lake environment. Evaluation curves for Politex, SARSA and Q-learning.

4.2 Cart-pole

We proceed to evaluation and comparison of three algorithm for more complex environment which is called Cart-pole intriduced earlier. This environment has more states than Frozen lake. In fact, it has continuous number of states. Therefore, in order to be able to evaluate the performance of all algorithm which work with only tabular data, we discretize each of four states (cart position, velocity; pole angle, angular velocity) to 30 values in their respective ranges.

In figures 6 and 7, learning and evaluation reward curves are shown for Politex, SARSA and Q-learning. We set $\eta = 1$, $\tau = 6000$ (phase length), $\alpha = 0.1$ (learning rate for TD learning), $\gamma = 0.995$ (discount factor), for Politex in the depicted experiments. We ran each algorithm for 10 times and ploted average reward curves, as well as the standard deviation for each one. For SARSA, learning rate is $\alpha = 0.25$, and $\alpha = 0.1$ for Q-learning; $\gamma = 0.995$ for both of the algorithms.

To choose mentioned parameters, we ran the experiments multiple times and picked those giving maximum reward values.

As we observe from the graphs, Politex surpasses both SARSA and Q-learning in its performance. During the learning process, Politex scores higher rewards in terms of values from the very beginning, while both SARSA and Q-learning struggle to pass higher than 100 in reward for more than 1000 phases (or episodes).

During evaluation Politex also scores 200-250 on average starting from around 50th phase, while SARSA and Q-learning do not achieve the same reward values in average even after 2000 phases. Hence, Politex learns faster than SARSA and Q-learning.

4.3 Influence of softmax temperature η on POLITEX

It is worth discussing the fact that Politex algorithm is highly sensitive from the choice of its softmax temperature η. Whereas, it was not obvious for us in the experiments for small Frozen Lake environment, in case of Cart-pole, the difference grows bigger.

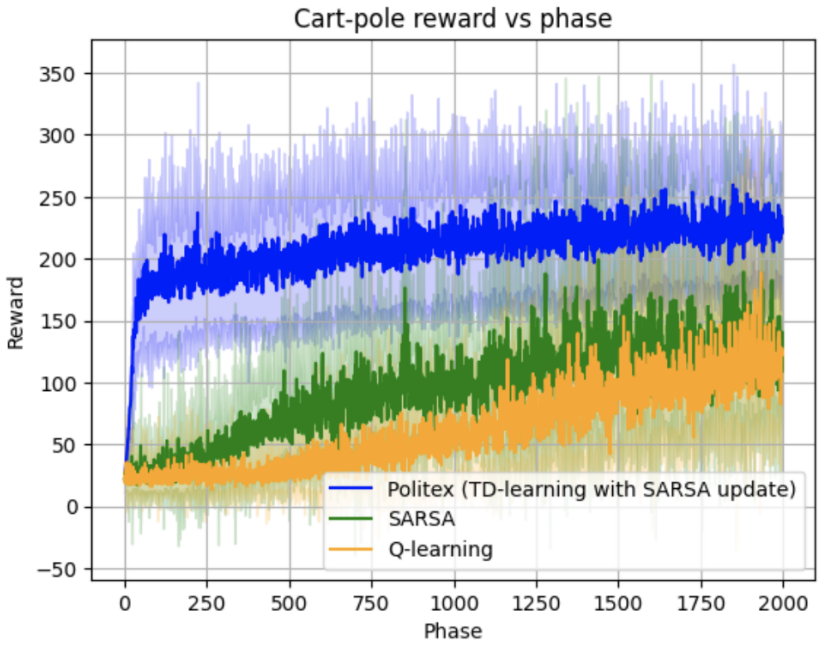


Figure 6: Experiments on the Cart-pole environment. Learning curves for Politex, SARSA and Q-learning.

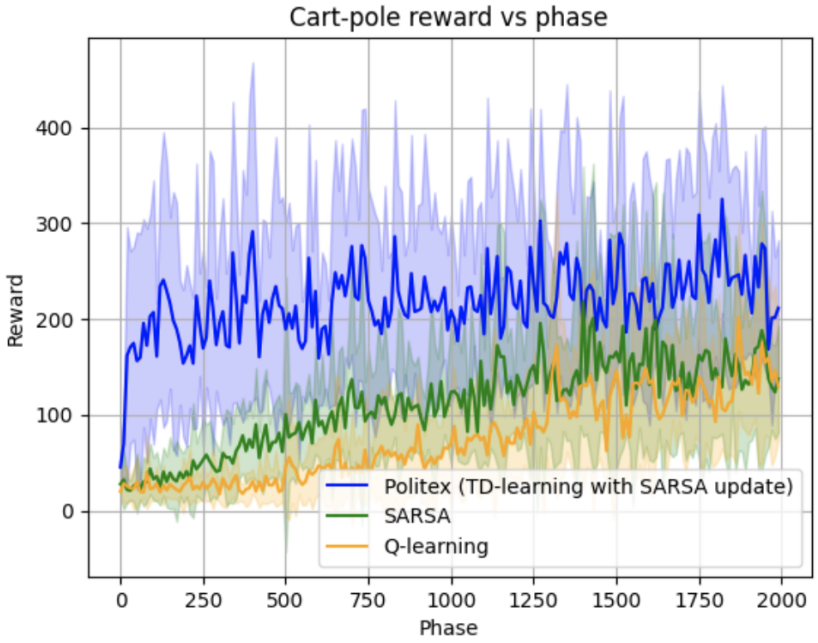


Figure 7: Experiments on the Cart-pole environment. Evaluation curves for Politex, SARSA and Q-learning.

To demonstrate this influence, we perform the same experiments for Politex as in the previous section, but for η in range $\{1, 5, 10, 20\}$. We plot average evaluation curve for each chosen η. For each η, Politex algorithm was launched 10 times.

In the figure 8, we show the results of these experiments. We notice that for $\eta = 1$ and $\eta = 5$, Politex gives the best average evaluation reward among 4 chosen η values, however, for $\eta = 1$ evaluation

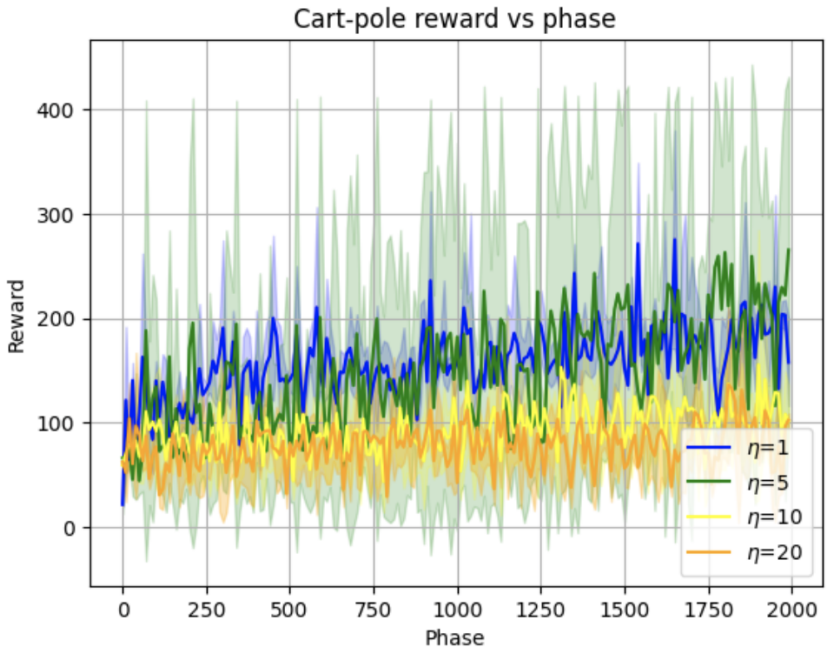


Figure 8: Experiments on the Cart-pole environment. Evaluation curves for Politex, SARSA and Q-learning.

curve is smoother or the standard deviation is smaller, than for $\eta = 5$. For $\eta = 10$ and $\eta = 20$, Politex gives small average reward on evaluation.

For this reason, in the previous experiments, we set up η to be 1.

4.4 Politex with Neural Networks

As it was mentioned before, Politex is a learning scheme. Hence, it is up to user to chose an appropriate estimate for Q_{π_i}.

It is possible to estimate Q_{π_i} by $\hat{Q}_{\theta_i}$ which is represented by Neural Network (NN). The target of this NN is $\hat{Q}^{target}_{\theta_i^-}$ which is defined as

$$\hat{Q}^{target}_{\theta_i^-}(c_t, s_{t+1}) = c_t + \gamma \sum_{a_{t+1} \in \mathcal{A}} \pi_i(a_{t+1}|s_{t+1}) q_{\theta_i}(s_{t+1}, a_{t+1}), \tag{4}$$

where θ_i^-, is an old version of θ_i that we have to keep fixed for a certain number of updates, like in DQN.

The objective to optimize is mean squared error (MSE) and is given as follows

$$\min_{\theta_i} \sum_{(s_t, a_t, c_t, s_{t+1})} \left(\hat{Q}_{\theta_i}(s_t, a_t) - \hat{Q}^{target}_{\theta_i^-}(c_t, s_{t+1}) \right)^2. \tag{5}$$

Unfortunately, we were not able to finish debugging the implementation of Politex with NNs to the present moment. This will remain to be our next step.

5 Conclusions

In this project, we studied three entropy regularized Reinforcement Learning algorithms, which are based on common sheme of Politex. We discussed that the algorithms presented in [2] and

[3] improve upon the classical Politex algorithm presented in [1], introducing more compact and computationally efficient ways to compute $\hat{Q}_{\pi_i}$ estimates.

Furthermore, we implemented Politex algorithm from paper [1] using TD-learning with SARSA update to estimate $\hat{Q}_{\pi_i}$. We showed that for such classical environments as Frozen lake and Cart-pole this algorithm surpasses other tabular techniques, specifically SARSA and Q-learning. We also investigated and proved by the experiments the sensitivity of Politex algorithm from softmax temperature η.

As potential future steps, we intend to implement Politex algorithm using Neural Networks as $\hat{Q}_{\pi_i}$ estimates, and also to compare it with imporved algorithms from papers [2] and [3].

In conclusion, entropy regularization has emerged as a promising technique to address the exploration-exploitation dilemma in reinforcement learning. By adding an entropy regularization term to the objective function of the RL algorithm, agents are incentivized to explore more, leading to better learning performance. Entropy regularization has been applied to various RL algorithms, and experimental results have demonstrated its effectiveness in various tasks. However, there are still challenges and future directions for research, including how to balance exploration and exploitation in more complex and dynamic environments, and how to optimize the hyperparameters of the regularization term. Overall, entropy regularization has the potential to improve the efficiency and effectiveness of RL algorithms, and is an active area of research in the machine learning and AI communities.

References

[1] Yasin Abbasi-Yadkori, Peter Bartlett, Kush Bhatia, Nevena Lazic, Csaba Szepesvari, and Gellért Weisz. Politex: Regret bounds for policy iteration using expert prediction. In *International Conference on Machine Learning*, pages 3692–3702. PMLR, 2019.

[2] Nevena Lazic, Dong Yin, Yasin Abbasi-Yadkori, and Csaba Szepesvari. Improved regret bound and experience replay in regularized policy iteration. In *International Conference on Machine Learning*, pages 6032–6042. PMLR, 2021.

[3] Riccardo Della Vecchia, Alena Shilova, Philippe Preux, and Riad Akrour. Entropy regularized reinforcement learning with cascading networks. *arXiv preprint arXiv:2210.08503*, 2022.

[4] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3:9–44, 1988.

[5] Scott Fahlman and Christian Lebiere. The cascade-correlation learning architecture. *Advances in neural information processing systems*, 2, 1989.