Assessing inference methods for various hyperbolically discounted Q and V functions

Alexandru Dumitrescu
Aalto University
Helsinki, Finland
alexandru.dumitrescu@aalto.fi

Askhat Issakov
Aalto University
Helsinki, Finalnd
askhat.issakov@aalto.fi

Mustafa Çelikok
Aalto University
Helsinki, Finalnd
mustafa.çelikok@aalto.fi

***Abstract*—Almost all reinforcement learning problems are inferred by considering exponential discount factors γ of future returns R_{t+k}. In order to diminish the importance of future rewards and make an agent decide on its actions based on a larger or smaller time-horizon, the γ discount can be tuned to account for the problem setting.**

This type of discount has been, however, limiting. By "looking" into the future, an agent decides to take a certain path based on R_{t+k_1}, R_{t+k_2}, and the chosen discount factor γ, and will stick to this decision, no matter how close or far time step t is to $t+k_1$ or $t+k_2$. This has been a modeling limitation, as humans and animals do not act in the same way. A human might change his mind when getting closer to either one of the two reward, R_{t+k_1}, R_{t+k_2}.

Inferring a model that considers discounted rewards through a discount factor other than the exponential γ, is, however, not trivial. We discuss in detail what problems do discount factors other than exponential pose when inferring certain models. We implement two different methods of inferring optimal solutions that have hyperbolically discounted returns. We experiment with multiple environments and assess the performance of each of those.

***Index Terms*—reinforcement learning, hyperbolic discounting, multi-horizon**

I. Background

A Markov Decision Process stands at the core of most of the reinforcement learning problems. It can be defined by its state space $\mathcal{S}$ and state transition function as a consequence of interacting with the environment by taking a certain action from the action space $\mathcal{A}$. Further, transitions from a given state $s_i \in \mathcal{S}$ to another $s_j \in \mathcal{S}$ can be stochastic. Its definition, as given by Richard S. Sutton and Andrew G. Barto [11] is:

$$p(s', r|s, a) = Pr\{S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\}, \tag{1}$$

where r is the reward given to the agent for reaching state s' from the state s, following action a. This function can be either a probability density or a probability mass function, depending on whether the state and action spaces are continuous or discrete.

In most of the reinforcement learning settings, the problems can be formulated as maximizing some reward. This reward, in turn, can depend on the states and the actions taken by the agent in the environment. We again use the definition given by Richard S. Sutton and Andrew G. Barto in [11]:

$$r(s, a) = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p\left(s', r \mid s, a\right), \tag{2}$$

where the state transition probability function is given in equation (1).

Agents act in a state environment S throughout an episode, which can have multiple, potentially infinite time steps. To "solve" a reinforcement learning problem, it is usually desired to maximize the potential cumulated reward of an entire episode. For this, we defined a policy $\pi(a|s)$ which is a function that gives a probability of taking an action $a \in \mathcal{A}$ from a state $s \in \mathcal{S}$. With this, we write down the definitions of a state-value and a state-action functions:

$$v_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right] \tag{3}$$

$$q_\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right] \tag{4}$$

The first definition gives the expected cumulated reward of the states values $s \in \mathcal{S}$, by following a policy π. The second one, gives the expected cumulated reward for following any possible action $a \in \mathcal{A}$ from state s. Throughout this project, we focus on optimizing state-value and state-action functions.

An additional term seen in equations (3) and (4) is the discount factor γ. As described in [6], this term can be thought of as the "probability of dying" of the agent. In a lot of problems, the agent can enter states that determine the end of an episode. Additionally, without this discount factor, infinite MDPs can reach infinite expected returns. Because of these problems, solving MDPs makes use of some type of discounting, which in this case exponentially decreases the influence of the reward based on how far in time it is considered, from he current time step t.

Discount factors other than the exponential discount γ pose a problem. Bellman optimality equations defined in (7) enable an iterative process of inferring an optimal state-value and action-value functions respectively (in terms of maximizing a reward). However, the optimality equations rely on the fact that the expected return of various states can be recursively written as:

$$
\begin{aligned}
G_t &= R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + ... \\
G_t &= R_t + \gamma \left(R_{t+1} + \gamma R_{t+2} + ... \right) \\
G_t &= R_t + \gamma G_{t+1}
\end{aligned} \tag{5}
$$

In equation (5), we see how the return G_t at time t (cumulative discounted-reward throughout the episode) can be written as a function of the immediate reward R_t and the return from time step G_{t+1}. This enables the recursive definition of optimality Bellman equations in (7). Choosing a discount factor other than the exponential γ, like the hyperbolic discounting of the form $\Gamma = \frac{1}{1+kt}$, makes it impossible for the discount G_t to be written in terms of only R_t and G_{t+1}. In turn, Bellman equations optimality equations won't hold.

A. Bellman optimality equations

Bellman equations define values of expected rewards for the states of an MDP. These values take into account both the immediate and future returns through and expected total return G_t, given some followed policy π.

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi \left[G_t \mid S_t = s \right] \\
q_\pi(s,a) &\doteq \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right],
\end{aligned} \tag{6}
$$

where v_π and q_π are value and action-value functions that follow policy π. Optimizing for the maximal return can then be done through learning a state-value or state-action value function that follow a policy which chooses the maximum state s or chooses the best action a s.t. the total future return G_{t+1} is maximized q_π. Maximizing the expected returns is equivalent in with Bellman optimality equations:

$$
\begin{aligned}
v_*(s) &= \max_a \sum_{s'r} p\left(s', r \mid s, a\right) \left[r + \gamma v_* \left(s' \right) \right] \\
q_*(s,a) &= \sum_{s',r} p\left(s', r \mid s, a\right) \left[r + \gamma \max_{a'} q_* \left(s', a' \right) \right]
\end{aligned} \tag{7}
$$

We note here that optimizing a policy through learning a state-action value function q can be done without explicitly knowing the dynamics of the model. In other words, we know what's the best action a to take from state s, without knowing the state s' to which we get by following a. In contrast, learning a state-value function v requires knowledge of what action needs to be taken in order to get to the best state s'.

B. Methods for learning inferring optimal MDP

Value iteration is a simple, dynamic programming method of determining an optimal policy on MDPs that have discrete, finite state spaces. It assigns every unique state (the MDP having a finite number of states) values that are updated based on it's reward and the values of neighboring states.

$$
v_{k+1}(s) \doteq \max_a \mathbb{E} \left[R_{t+1} + \gamma v_k \left(S_{t+1} \right) \mid S_t = s, A_t = a \right] \tag{8}
$$

Monte Carlo had been a widely used method of estimating a state-value function given a policy, in the case of MDPs that have a finite horizon (so episodes must end after a finite number of iterations). This method relies on generating episodes from an MDP using some policy π, retaining at each step all states, actions and rewards it gets along the way: $(S_0, A_0); (S_1, A_1, R_1); ...; (S_n, A_n, R_n)$. After generating it, it loops over all triplets computes a cumulative reward G, which is initialized with 0 and, at each step t, updates as follows:

$$
G \leftarrow \gamma G + R_{t+1} \tag{9}
$$

where $\gamma \in (0, 1)$ is a discount factor and R_{t+1} is the reward initially generated by the model. All G are then retained and mapped to their corresponding states S_t that generated them. MC methods have dropped in popularity, mainly due to their hard constraint on the horizon limit as well as an arguably inefficient method of generating training data: MC cannot start inferring any policy until a full episode has been generated.

Temporal difference (TD) is the preferred method of inferring optimal policies in most of the recent RL algorithms. Comparatively, it does not rely on generating a full episode to start updating it's a state-value function. The update function of TD is:

$$
V\left(S_t\right) \leftarrow V\left(S_t\right) + \alpha \left[R_{t+1} + \gamma V\left(S_{t+1}\right) - V\left(S_t\right) \right] \tag{10}
$$

where V is a state-value function, α is an arbitrarily chosen learning rate and the rest of the notations are the same as in (9).

TD inference can also be used to learn state-action value functions. Notably different, a state-action value function $Q(S_t, A_t)$ can learn independently of any policy and it only needs to have all of it's actions from every state updated enough times for convergence. It's equation is:

$$
\begin{aligned}
Q\left(S_t, A_t\right) \leftarrow & Q\left(S_t, A_t\right) + \alpha [R_{t+1} + \\
& \gamma \max_a Q\left(S_{t+1}, a\right) \\
& - Q\left(S_t, A_t\right)]
\end{aligned} \tag{11}
$$

This equation gives us a way of learning an action-value function for each step we take, as the reward R_{t+1} is observed after taking just one action A_t from state S_t to S_{t+1}.

II. Hyperbolic discounting

Hyperbolic discounting is a form of non-linear discount rate. It has been shown to be more accurate when modeling behaviour of animals and humans [2], [3], due to time inconsistency that animals show [4], [5]. All RL models that use an exponential γ as their discount rate, are consistent between present and future timelines when deciding what state to follow given some rewards. Concretely, looking at two states S and S' in the future with their discounted rewards $\gamma^{d_1} R_{t+d_1}$ and $\gamma^{d_1} R_{t+d_2}$, their discount factors do not depend on how close to these states we are from the current time t. The model is consistent with its decision to go either for the reward R_{t+d_1} or R_{t+d_2} at time t as well as $t+d$, for any d satisfying $d < d_1$ and $d < d_2$.

We give an example that illustrates the behaviour we described above. Offered 1$ million now or 1.1$ the next day,

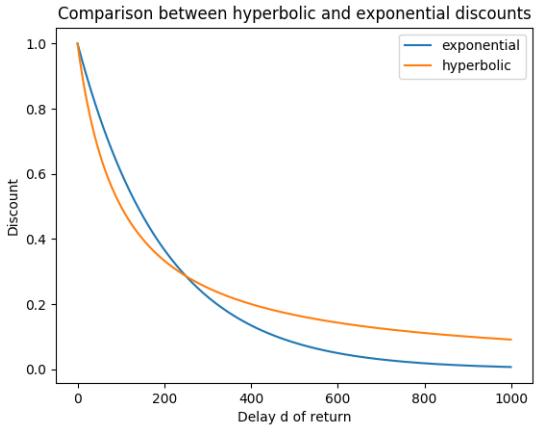

Fig. 1. Exponential and hyperbolic discounts computed for a delay $d \in (0, 1000)$ for $\gamma = 0.995$ and $k = 0.01$

a human usually chooses the 1$ million reward immediately. However if the person is offered 1$ million in 30 days or 1.1$ million in 31 days respectively a person will be more inclined to choose the 1.1 million 31 days later.

For a human, it is important to know how far away the current time t we consider the two rewards, and the decision might differ based on that. Assuming $d_2 > d_1$, an exponentially discounted RL model will chose its reward only based on the $d_2 - d_1$. This form of preferential change cannot therefore be capture by Q or V functions inferred using an exponential discount factor.

Hyperbolic discount factor is suitable for this kind of preferential change based on how far away in the future we consider rewards and can capture time-inconsistent behaviour. It is defined as:

$$d_k(t) = \frac{1}{1 + kt}, \tag{12}$$

with k being an arbitrarily chosen value $k > 0$.

In Figure 1, we see how the hyperbolic discounts drop faster initially (future rewards are taken less into account then when using the exponential discounts), but at some point it begins to flatten.

Suppose we start from at time $t_0 = 0$ and consider an exponential discount factor γ. Looking at the exponentially discounted rewards ration of two potential rewards at different time-steps, $\gamma^{t+d_1} R_{t+d_1}$ and $\gamma^{t+d_2} R_{t+d_2}$, $t + d_1$ and $t + d_2$ away from t_0, we have $f = \frac{\gamma^{t+d_1} R_{t+d_1}}{\gamma^{t+d_2} R_{t+d_2}} = \gamma^{d_1 - d_2} \frac{R_{t+d_1}}{R_{t+d_2}}$. This means that models decide only between the relative ratio of delays (and their rewards) d_1 and d_2, without accounting for t. Comparatively, considering a hyperbolic discount scheme with some $k > 0$ we have $f' = \frac{R_{t+d_1}/(1+k(t+d_1))}{R_{t+d_2}/(1+k(t+d_2))} = \frac{R_{t+d_1}(1+kt+kd_2)}{R_{t+d_2}(1+kt+kd_1)}$. In this latter case, hyperbolic discounting is clearly not independent of how far in the future we consider the two rewards and reward preferences can change based on t.

The methods described so far in Chapter I-B are only directly applicable when a discount rate other than the exponential one is considered. The cumulative expected return G_t out of an episode can be maximized using the Bellman optimality equations (6), only when the cumulative return can be recursively expressed as in (5). This does not apply for non-linear discount rates like the hyperbolic discounting we use here, as $d_k(t + 1) \neq d_k(1) \cdot d_k(t + 1)$ with d_k being the discount rate (12). We continue by describing two methods for learning approximations of hyperbolic discounted value and action-value functions that we also use in our experiments.

III. Methods of inferring hyperbolic-discounted models

We present two methods for inferring hyperbolically discounted RL models. Our first approach makes use of the fact that the hyperbolic function can be expressed by the integral of an exponential function (15). Through appropriate weighting, a hyperbolically discounted function can be approximated by various exponentially discounted ones, inferred on different horizons [6]. The second method uses the transformed reward function $\mathrm{g}(R)$ (Equation 17) to approximate a hyperbolically-discounted function while using the standard exponential discount factor in environments with sparse rewards [7].

A. Approximating hyperbolic discounted models using exponentially discounted models on multiple horizons

The first Lemma in [6], proves the fact that if a functions a function $w : [0, 1] \to \mathbb{R}$ such that:

$$d(t) = \int_0^1 w(\gamma)\gamma^t d\gamma \tag{13}$$

then, we can express a state-action function that uses any discount factor d, through the factors w and exponentially discounted state-action functions with $\gamma \in (0, 1)$, by:

$$Q_\pi^{\mathcal{H},d}(s, a) = \int_0^1 w(\gamma) Q_\pi^{\mathcal{H},\gamma}(s, a) d\gamma, \tag{14}$$

where $\mathcal{H}, d$ are the hazard of the MDP and the general discount function $d(t)$, that is given in (12) in the case of hyperbolic discounting and $d(t) = \gamma^t, \gamma \in (0, 1)$ in the case of exponential discounting.

The hazard $\mathcal{H}$ is a way of re-formulating the idea of an agent having a $1 - \gamma$ chance of dying when using an exponential discounting with a factor γ [8].

In particular for hyperbolic discounting, it can be shown that:

$$\frac{1}{k} \int_{\gamma=0}^1 \gamma^{1/k+t-1} d\gamma = \frac{1}{1 + kt} \tag{15}$$

So, for hyperbolic discounting, the function w is taken from $f(\gamma, t) = \frac{1}{k}\gamma^{1/k+t-1} = w(\gamma)\gamma^t$. Using the results from (14), the form of the function w from equation (15) and discretizing the integral, we get the following:

$$Q_\pi^\Gamma(s,a) \approx \sum_{\gamma_i \in \mathcal{G}} (\gamma_{i+1} - \gamma_i)\, w(\gamma_i)\, Q_\pi^{\gamma_i}(s,a) \qquad (16)$$

Which finally gives us an approximation of a hyperbolically discounted action-value function, for which we need to simultaneously learn exponential action-value functions $Q_\pi^{\gamma_i}$ as auxiliary tasks, with $\gamma_i \in (0,1), \gamma_i < \gamma_{i+1}$.

B. Learning through modifying the reward function

Another method for learning hyperbolically-discounted functions was introduced in [7]. The authors discussed a general class of non-linear Bellman equations and their benefits. The classical formulation of Bellman equations assumes a linear form of cumulative return function from Equation 9. The linear formulation limits the modelling power of Bellman equations to exponentially-discounted cumulative return. On the other hand, non-linear Bellman equations have two benefits: modelling a wider range of natural phenomena and allowing for more flexibility in design choices for learning algorithms. We are most interested in the first benefit because it allows us to model hyperbolic discounting by modifying the reward function.

There are three ways we can introduce non-linearity to Equation 9. We can transform:

- the reward

$$f(R,G) = \mathrm{g}(R) + \gamma G,$$

- the value

$$f(R,G) = R + \mathrm{g}(G),$$

- the target as a whole

$$f(R,G) = h(R + \mathrm{g}(G)),$$

where g and h are scalar functions.

We are using reward transform $\mathrm{g}(R)$ (Equation 17) to approximate a hyperbolic discount factor while using the standard exponentially discounted value (Equation 18).

$$\mathrm{g}(R) = re^{\eta(R/r-1)}, \qquad (17)$$

which gives us

$$G_t = \sum_{n=0}^{\infty} \gamma^n \mathrm{g}(R_{t+n+1}) = \mathrm{g}(R_{t+1}) + \gamma G_{t+1}. \qquad (18)$$

The proof that this reward transform can approximate the hyperbolic return is presented in Theorem 1 from [7]. Note that $\mathrm{g}(R)$ works only for environments with sparse rewards.

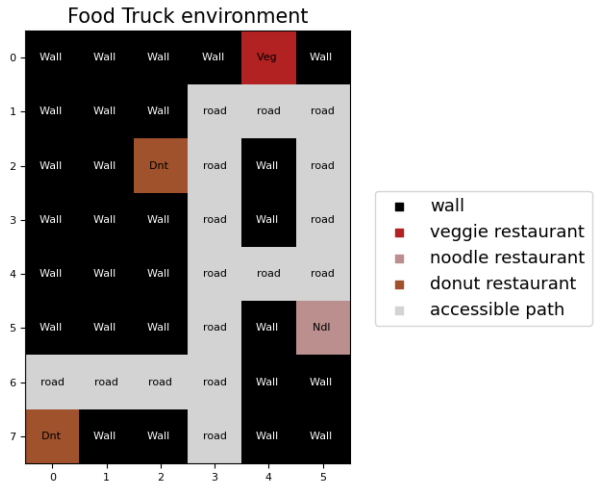

Fig. 2. Food truck environment.

IV. Environments

A. Food truck

In order to test the ability of hyperbolically discounted functions to model a decision that would be typical for a human, but impossible to achieve for an exponentially discounted RL model, we use a simple, discrete environment and infer a value function for it, using value iteration.

In Figure 2 we illustrate the 8×6 Food truck discrete environment for which we will infer an optimal policy according to a hyperbolically-discounted value function. For this environment, rewards are given at the time the agent exits a given state.

The environment is a finite MDP with end states located in each restaurant: in the veggie restaurant, donut stores ("dnt" in the image) or noodle ("ndl" in the image). We use two different states and rewards for the restaurants, meaning that after the first arrival in a given restaurant, the agent waits one more time step in the same location (it is unable to move from there) and then receives the delayed reward.

For both of the donut stores, the immediate reward is $r_i = 10$, while the delayed one is negative $r_d = -10$. The noodle retaurant has equal delayed and immediate rewards $r_i = r_d = 0$ and lastly, the veggie restaurant has the highest total gain, with a negative immediate reward $r_i = -10$, but a higher delayed positive reward $r_d = 20$.

B. Cart Pole

We test the hyperbolic-discounted inference using multiple exponentially discounted action-value functions on a continuous, infinite MDP. We use a continuous state-space MDP in which the agent can take one of two possible discrete actions.

We use the Cart Pole environment model provided by OpenAI [9], in which a pole is attached to a cart in an unactuated joint, as illustrated in Figure 3.

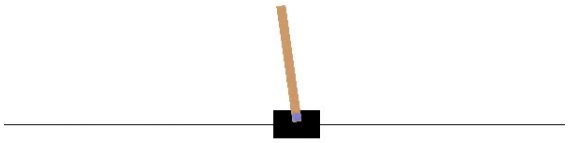

Fig. 3. Cart Pole environment.

The model's state space is small, but (notably different than Food Truck) continuous, with $S \in \mathcal{R}^4$. The 4 numbers describing each state in the environment (x, v, θ, ω) are given by the cart position x (on the uni-dimensional x-axis), the cart velocity v, the pole angle θ and the angular velocity of the pole ω.

The action space is discrete, with $A \in \{-1, 1\}$, meaning the agent may only apply some force on the cart to move it towards left or right.

The goal of this environment is to stabilize the pole in a vertical line ($\theta \approx 0$). The MDP is infinite, meaning that the Cart Pole may run forever, as long as the pole is stabilized. An episode ends if either the cart is moved too far $x < -2.4$ or $x > 2.4$ from the center position $x = 0$, or if the pole is inclined too much $\theta < -0.418$ rad or $\theta > 0.418$ rad.

C. Frozen lake

Frozen lake represents a simple grid-based environment with 16 discrete states (8×8 grid). The agent controls the movement of a character on a frozen lake, which has to navigate on frozen ice patches around unsafe ice holes. The states on Figure 4 are defined as follows:

- Start - starting point, safe
- Frozen - frozen surface, safe
- Hole - ice hole, unsafe
- Goal - goal state, safe

The action space is discrete and allows movement in four directions on the grid. Additionally, there is an uncertainty in the movement, which leads to the mismatch between the intention and the actual move.

The agent receives a reward of 1 when reaching the goal state and 0 otherwise. The episode terminates when the agent reaches the goal or falls into an ice hole.

D. Mountain car

Mountain car represents a one-dimensional car track with two mountains, where the goal is to climb one of them (Figure 5). However, the car cannot climb it in a single pass and needs to build up the momentum by moving back and forth.

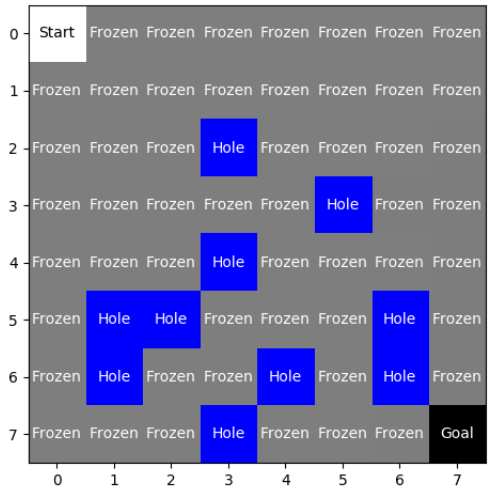

Fig. 4. Frozen lake environment

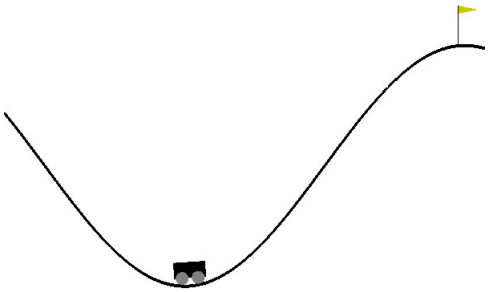

Fig. 5. Mountain car environment

The state space is continuous and consists of two components: the position of the car x in the horizontal axis on the range $[1.2, 0.6]$ and its velocity v on the range $[0.07, 0.07]$. The action space is discrete and consists of 3 possible actions: 0 - accelerate to the left, 1 - do not accelerate and 2 - accelerate to the right.

The reward is 0 if the agent reaches the flag (position = 0.5) on top of the mountain and the reward is -1 if the position of the agent is less than 0.5. The episode terminates when the car position is more than 0.5 or the episode length is greater than 200.

V. Implementation and Results

A. Food Truck

The important aspect of this environment is that solving its optimality through an exponentially discounted value function or action-value function will always result in an agent arriving at the bottom-left donut store or at the veggie restaurant. Depending on how small or large the exponential factor γ is chosen, the agent will always prefer the closest donut store of the an immediate reward (when γ is low and the agent is myopic), or the vegetarian store, when γ is large enough.

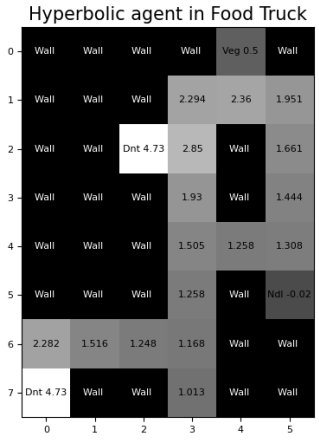

Fig. 6. Hyperbolic discounted value function inferred through multiple exponentially discounted functions.

Comparatively, the hyperbolic discounted model is able to learn a policy that takes the agent to the northern donut store. In Figure 6, 1000 value functions are simultaneously learned, each with a different γ exponential discount.

Each of the value functions V_i is learned using (10), with the discounted next state return being computed with its corresponding exponential discount factor and value function $\gamma_i V(S_{t+1})$ while the final, hyperbolic discounted approximation is taking the weighted sum of all value functions, similar to how action-value functions are aggregated in (16).

The resulted value function, plotted in the 8×6 grid, shows how a policy that chooses to follow states with the corresponding maximal value in the value function will lead an agent to the northern donut store (at position $(2, 2)$).

Intuitively, a human that would pre-emptively decide for himself to eat at a vegetarian restaurant (which in this scenario, is healthier and produces a high delayed reward), could be deterred while passing by the donut restaurant, which has a short term benefit, but the delayed reward makes it an overall worse decision. This kind of behaviour is similar to the decision taken on whether to take the 1 million dollars now or in the future. Humans and animals tend to change their behaviour based on how far in time two relative events occur, and this kind of behaviour is not captured by exponentially discounted models.

B. Cart Pole

We make further experiments with the hyperbolic model approximation from (16). In this experiment, we attempt to learn multiple action-value function approximations, learned through Deep Q-learning, that will reach 200 time-steps in the Cart Pole environment described in Chapter IV-B.

We train simultaneously 50 action-value functions $Q_i(s, a))$, with their own corresponding discounting factor γ_i. As the method described in [6], we construct a model that will be common across all action-value functions $Q_i(s, a))$, with only the last layer (the head) being separate for each $Q_i(s, a))$.

Implementation-wise, we train two linear layers, with their corresponding matrix dimensions $\mathcal{L}_1 \in \mathbb{R}^{4\times 48}$ and another $\mathcal{L}_2 \in \mathbb{R}^{48\times 100}$. The first layer $\mathcal{L}_1$ is responsible for mapping the input space $S \in \mathbb{R}^4$ to some common feature space of $\mathbb{R}^4 8$, that will be used by all 50 action-value functions. The second layer is an equivalent and more efficient way of computing 50 different activations from 50 different linear layers $\mathcal{L}_i \in \mathbb{R}^{48\times 2}$ (appropriately extracting the outputs from our output vector $o \in \mathbb{R}^{100}$ will give use 2 actions - push the cart left or right - for each of the 50 models).

In order to make the training more stable, we use a target target action-value function, that is updated every 100 episodes. This target network mimics our original model defined by its $\mathcal{L}_1$ and $\mathcal{L}_2$ layers, but it remains frozen (no gradients flow through it). It is used to compute the next state-action values and its parameters are copied from $\mathcal{L}_1$ and $\mathcal{L}_2$ every 100 finished episodes.

We additionally control the exploration the model undergoes with a parameter ϵ. We let the model explore less (and exploit more the state-actions it found to be returning higher rewards), through gradual annealing of this ϵ parameter. Concretely, we use $\epsilon = \max(0.1, \frac{\tau}{\tau+e})$, where τ is a parameter we tune and e is the number of epochs elapsed until that point. When choosing the next action during training, we either choose randomly an action with probability $p = \epsilon$, or select the action the model predicts as the best action with probability $p = 1 - \epsilon$. Notable how the probability of random actions slowly decreases as the training advances, making the model exploit more those states it found to be good.

We train the models with a batch size of 64, a learning rate of 10^{-4} using RMSprop optimizer, gradient clipping, and different τ parameters. Additionally, we implement gradient clipping, which has been shown to sometimes help mitigate the problem of catastrophic forgetting [10].

We illustrate how much the amount of exploration affects these models in Figures 7 8. In the first figure, although after a point the model suffers from catastrophic failure, it is easily observable how its training is quite stable in the first half of the training with $\tau = 1000$. In comparison, the second plot shows how the training is oscillating a lot all throughout the training when the model has more freedom in choosing its training samples (it has more chances of deciding the next states to go to) for $\tau = 300$.

We repeat the same experiment using the same hyperparameters for a single, exponentially discounted model with a discount factor $\gamma = 0.98$. Its test accuracy over training is shown in Figure 9.

Comparatively, we see how this model learns relatively quickly, but it is also much more prone to catastrophic forgetting. Even though mechanisms like gradient clipping are implemented for both, selecting the model that reached the highest cumulative reward (similar to early stopping) is probably necessary.

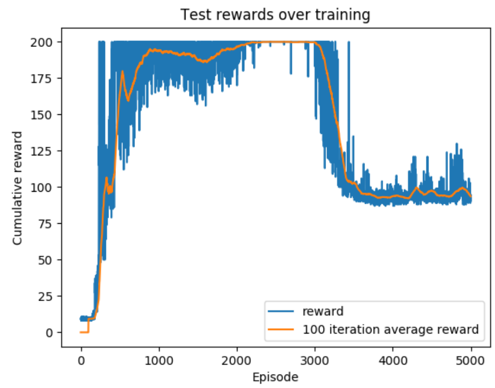

Fig. 7. Hyperbolic discounted value function inferred through multiple exponentially discounted functions.

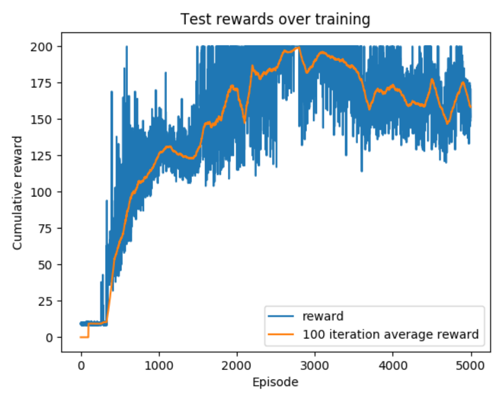

Fig. 8. Hyperbolic discounted value function inferred through multiple exponentially discounted functions.

Fig. 9. Exponentially discounted Q function with $\mathcal{L}_1 \in \mathbb{R}^{4\times 48}$ and $\mathcal{L}_2 \in \mathbb{R}^{[illegible]}$.

We clearly see how the hyperbolic model is much more stable in Figure 7. We also note that its test curve plotted over training is quite consistent, and many runs yielded very similar-looking results. We additionally note how, even though the model learns well, it seems to need a lot of exploration for good convergence. The same model trained with $\tau = 1000$ in Figure 7 is much more consistently increasing than the one trained with $\tau = 300$ Figure 8, possibly because the hyperbolic model is not yielding good training samples, even though it is able to reach the 200 time-steps goal.

C. *Frozen lake*

Since Frozen Lake is a grid-based environment, we use the value iteration algorithm [11] to solve for the optimal path from Start to Goal. We loop through each state of the environment updating state values until a small threshold of 10^{-6} is reached. Figure 10 illustrates the result of running the standard exponentially-discounted value iteration algorithm with γ equal to 0.99. We can see that the algorithm converged to the correct solution. Testing the solution 100 times results in 85 successful runs. Even through the stochastic nature of the environment prevents the agent from reaching the perfect result, we can conclude that the agent learned the optimal policy.

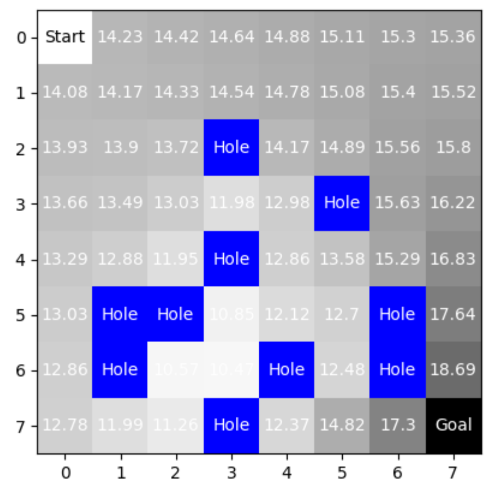

Fig. 10. State values after running the value iteration algorithm with the threshold $\theta = 10^{-6}$ and discount factor $\gamma = 0.99$.

Running the same value iteration algorithm with modified reward function $g(R)$ according to Equation 17, we receive similar results (Figure 11). However, one observation we can make is that increasing the value of k results in better approximation of exponentially-discounted return. For example, note how the state value of cell $(6, 7)$ changes with increasing k. The state value is equal to 22.54 with $k = 1$, while for $k = 1000$ the state value is 18.69, which is equivalent to the state value from exponentially-discounted function (Figure 10).

Transformed reward functions achieve the following results on 100 test episodes:

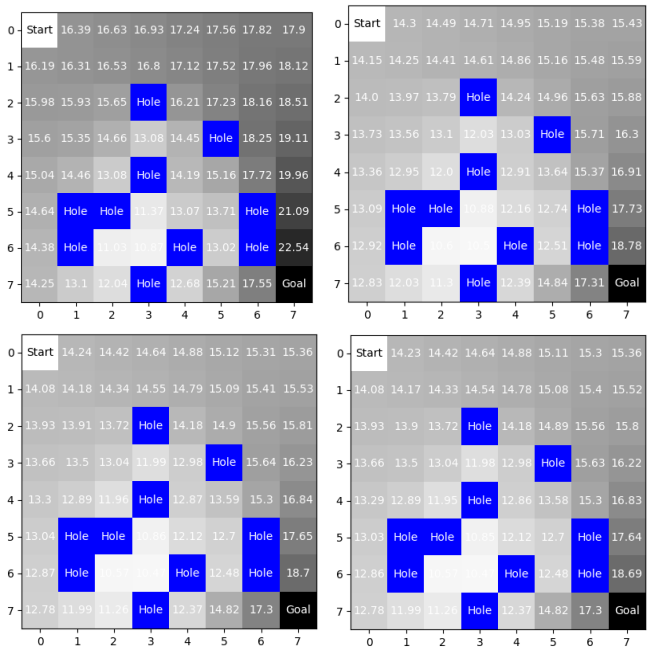

Fig. 11. The figure shows the state values after training the modified reward value iteration algorithm with the threshold $\theta = 10^{-6}$ and discount factor $\gamma = 0.99$. The values of the hyperbolic discount parameter k are the following: $k = 1$ (top left), $k = 10$ (top right), $k = 100$ (bottom left), $k = 1000$ (bottom right).

- $k = 1$ - 87 successes
- $k = 10$ - 89 successes
- $k = 100$ - 82 successes
- $k = 1000$ - 83 successes

The test results are approximate because of built-in randomness in Frozen Lake environment.

The Frozen Lake results are less interpretable compared to the Food Truck environment in terms of agent preferences because there is only one destination state. Additionally, there is no delayed reward for the final state, which does not change the state value calculations for the approximated hyperbolic discount function. However, the results of the training demonstrate that the modified reward function approximates exponentially decaying state values very well.

D. Mountain car

We used the Q-learning algorithm [11] to learn the optimal policy in the Mountain Car environment. We discretized the continous state space into 36 discrete states for each component, which resulted in 36×36 grid. The agent trained for 10000 episodes and tested the policy on 10 episodes. The policy was learned using GLIE (Greedy in the Limit with Infinite Exploration) policy improvement $\epsilon = \frac{a}{a+e}$, where $a = 0.1$ and $e = 10000$ (number of training episodes). Figure 12 shows the result state values after training the standard exponentially-discounted Q-learning algorithm using discount factor $\gamma = 0.99$. Learned state values tend to concentrate around the center slightly shifted towards negative velocities, which means that the agent prefers maneuvers at the center in order to build the momentum. Figure 13 shows the episode lengths and 500 episode average during the training. The agent fails to reach the top of the mountain at the beginning but starting around episode 3000 the agent is able to complete the episode earlier than the maximum episode length of 200. The average episode length at the end of the training is approximately 180.

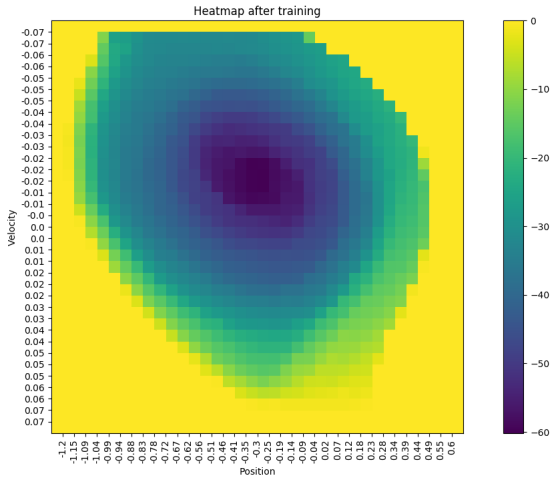

Fig. 12. The figure shows Mountain Car state values after training the Q-learning algorithm for 10000 episodes using the discount factor $\gamma = 0.99$ and the adaptive ϵ (GLIE).

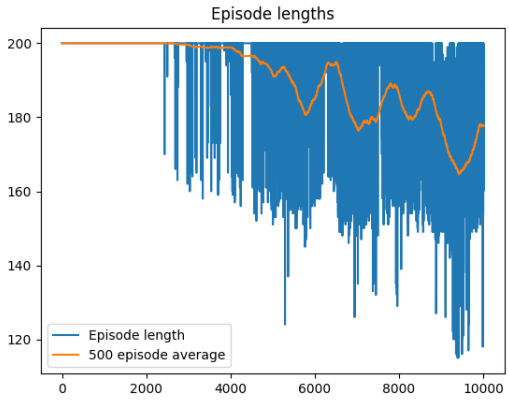

Fig. 13. The figure shows Mountain Car episode lengths for Q-learning algorithm using the discount factor $\gamma = 0.99$ and the adaptive ϵ (GLIE).

The modified reward function $g(R)$ from Equation 17 produces results similar to the standard Q-learning algorithm (Figures 14 and 15). However, as with the results for Frozen Lake environment, we observe that increasing the value of k results in better approximation of exponentially-discounted return. We can see that both from the state values and episode lengths. State values are more concentrated for larger k and the algorithm converges better.

As with the Frozen Lake environment, Mountain Car results are not easily interpretable in terms of agent preferences. There is only one destination state and no delayed reward for it.

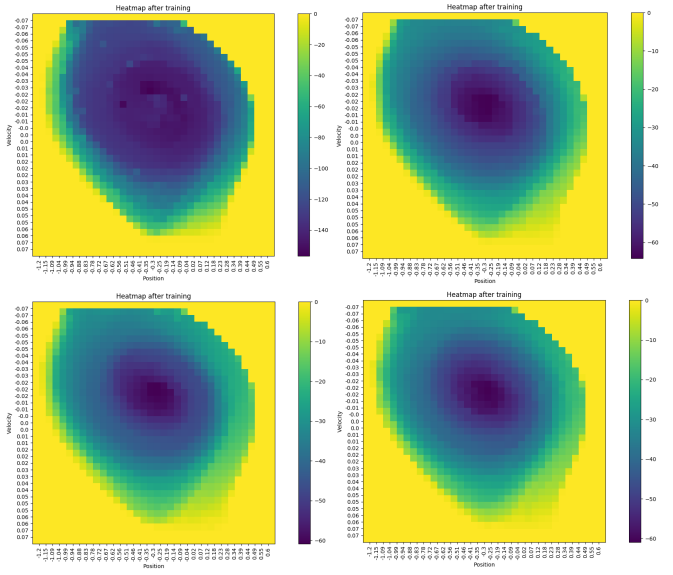

Fig. 14. The figure shows Mountain Car state values after training the modified reward Q-learning algorithm for 10000 episodes using the discount factor $\gamma = 0.99$ and the adaptive ϵ (GLIE). The values of the hyperbolic discount parameter k are the following: $k = 1$ (top left), $k = 10$ (top right), $k = 100$ (bottom left), $k = 1000$ (bottom right).

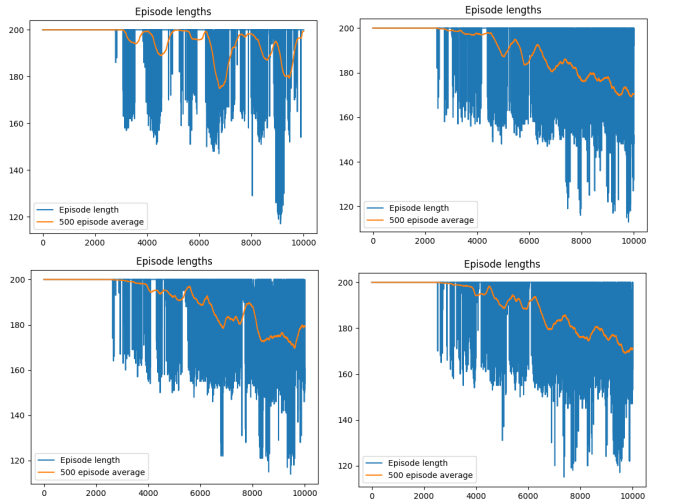

Fig. 15. The figure shows Mountain Car episode lengths for modified reward Q-learning algorithm using the discount factor $\gamma = 0.99$ and the adaptive ϵ (GLIE). The values of the hyperbolic discount parameter k are the following: $k = 1$ (top left), $k = 10$ (top right), $k = 100$ (bottom left), $k = 1000$ (bottom right).

Nevertheless, the training results allow us to make a similar conclusion that the modified reward function approximates exponentially decaying state-action values very well.

VI. Conclusion (and future work)

We have seen how hyperbolic discounting can be used to model human behaviour in the food truck environment. We also noted how the multi-horizon inference method produces arguably better results on a continuous state-space, infinite-horizon environment, in CartPole.

The relatively better result (compared to the exponential training curve) can most likely be accounted to how the hyperbolic-training in this setting acts as an ensemble method, and is therefore more stable. We also note that multi-horizon hyperbolic training method requires a very close attention to the exploration and network size parameters. Gradient clipping and early stopping, as suggested in [10] might still be necessary, as catastrophic forgetting seems to occur regardless of the improved stability we illustrated in the hyperbolic-cartpole training.

We also conclude that the modified reward function approximates exponentially-discounted state and state-action value functions very good especially for larger k values. However, proving the feasibility of the reward transform function for approximating hyperbolic discounting requires a more careful selection or design of environments. An environment has to have several competing goals and corresponding policies as well as delayed rewards in order to evaluate the effectiveness of hyperbolic discount approximation.

To summarize, we conclude that the multi-horizon technique is capable of successfully modelling hyperbolically discounted state-action functions with very few restrictions, inferring functions that achieve good and stable results even on infinite MDPs with continuous state spaces. On the other hand, the multi horizon technique can be computationally expensive, and in the right settings, the reward function modification might be the better approach.

Although we did not experiment with many of recent advents of reinforcement learning, both the multi-horizon and reward modification techniques should be suitable with more powerful and reliable techniques, like Rainbow [12] or double Q learning [13], which would potentially enable inferring models on much more complex MDPs.

References

[1] G. Eason, B. Noble, and I. N. Sneddon, “Reinforcement Learning, second edition”.

[2] Ainslie, G. (1975). “Specious reward: A behavioral theory of impulsiveness and impulse control. Psychological Bulletin“, 82(4), p 463–496.

[3] Frederick, Shane and Loewenstein, George and O’Donoghue, Ted 2002 “Time Discounting and Time Preference: A Critical Review“, Journal of Economic Literature, pages 351-401

[4] Richard Thaler, “Some empirical evidence on dynamic inconsistency,”, 1981, p. 201-207

[5] Green, Leonard and Myerson, Joel and Lichtman, David and Rosen, Suzanne and Fry, Astrid. “Temporal Discounting in Choice between Delayed Rewards: The Role of Age and Income. Psychology and aging. 11.” (1996) p. 79-84].

[6] William Fedus and Carles Gelada and Yoshua Bengio and Marc G. Bellemare and Hugo Larochelle “Hyperbolic Discounting and Learning over Multiple Horizons“, 2019

[7] Hado van Hasselt, John Quan, Matteo Hessel, Zhongwen Xu, Diana Borsa, Andre Barreto, “General non-linear Bellman equations“, DeepMind, London, UK, 2019

[8] Tor Lattimore and Marcus Hutter. “Time consistent discounting.“ In International Conference on Algorithmic Learning Theory, pp. 383–397. Springer, 2011.

[9] Greg Brockman and Vicki Cheung and Ludwig Pettersson and Jonas Schneider and John Schulman and Jie Tang and Wojciech Zaremba, “OpenAI Gym“, 2016

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. “Human-level control through deep reinforcement learning. Nature, “, 2015
[11] Richard S. Sutton and Andrew G. Barto “Reinforcement Learning: An Introduction.“
[12] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, David Silver “Rainbow: Combining Improvements in Deep Reinforcement Learning“
[13] Hado van Hasselt, Arthur Guez, and David Silver “Deep Reinforcement Learning with Double Q-learning“

VII. Code and data

The value-iteration and multihorizon hyperbolic discount implementation for Food Truck and Cart Pole can be found here: https://github.com/Dumitrescu-Alexandru/Research-project. The implementation of approximate hyperbolic discount factor through reward transforms for Frozen Lake and Mountain Car environments can be found here: https://github.com/askhissak/rp_project. The Cart Pole, Frozen Lake and Mountain Car environments can be obtained through pip/conda installation of OpenAI-gym library (https://gym.openai.com/).