Online Decision Tree Construction with Deep Reinforcement Learning

Ayman Chaouki
AI Institute, LIX laboratory
The University of Waikato, Ecole Polytechnique
ayman.chaouki@ip-paris.fr

Albert Bifet
AI Institute
The University of Waikato
abifet@waikato.ac.nz

Jesse Read
LIX Laboratory
Ecole Polytechnique
jesse.read@polytechnique.edu

Abstract

Decision Trees are widely used in machine learning and data mining tasks, mainly because they can be easily interpreted; due to their popularity, Decision Trees were adapted for settings with streams of data, commonly in the form of Hoeffding Trees. While these methods are fast and incremental, they are also greedy in the sense that they optimise multiple local criteria (generally based on Entropy or Gini impurity) which makes them prone to suboptimality with respect to a global objective metric. On the other hand, Reinforcement Learning (RL) aims at maximizing a long term objective, and as such, it is a good candidate for alleviating this suboptimality problem of the standard Decision Tree methods. In this work, we show that looking for the most accurate Decision Tree with the lowest depth is equivalent to solving an RL problem, then we implement Deep RL algorithms DQN, Double DQN and Advantage Actor-Critic to seek the optimal Decision Tree, this choice being motivated by the scalability of these methods to problems with large state spaces unlike Q-Learning. We compare these methods with Hoeffding Trees on real-world data sets and show that DQN and Double DQN perform best in general.

1 Introduction

Decision trees (DTs) are useful tools for classification problems, their main strength resides in being easy to interpret. Indeed, a fitted DT extracts simple rules that partition the input space and provides the importance of each attribute for the classification problem at hand. The problem of constructing a shallow optimal DT is NP-complete Laurent and Rivest (1976), thus, the classic algorithms, in the batch setting, like C4.5 Quinlan (2014) and CART Breiman et al. (1984), use a heuristic that constructs a DT in a top-down fashion by greedily maximizing a gain criterion, using Entropy or Gini impurity, at each node to select the best split, unfortunately this approach presents no guarantee of optimality with respect to some global objective metric and is thus likely to yield suboptimal tree structures.

Inspired by these batch algorithms, DTs were adapted to the Data Stream Learning setting, where data arrives incrementally over time; such frameworks are increasingly relevant in modern applications, hence Hoeffding Trees were developed and investigated in dozens of research papers, e.g., in Domingos and Hulten (2000); Bifet Figuerol and Gavaldà Mestre (2009); Manapragada et al. (2018). However, these methods inherit the suboptimality issue of the batch methods, as such, they are invariably deployed in large ensembles constantly renewing trees across the stream; usually proving

16th European Workshop on Reinforcement Learning (EWRL 2023).

costly and inefficient Read (2018). In addition, by using ensembles, we lose the easy interpretability of DTs.

Reinforcement Learning (RL) Sutton and Barto (2018) on the other hand aims at maximising a long-term objective in the form of an expected cumulative reward, hence, carefully modelling the online DT construction problem within an RL framework would allow us to seek the optimal DT with respect to a global objective function which would alleviate the suboptimality issue; furthermore, RL is naturally well suited for online learning. A first work on this topic was carried out by Garlapati et al. (2015), the authors formulated an interesting Markov Decision Process (MDP), however, the link between solving this MDP and optimising some global objective over the space of DTs was not clearly stated, in addition, the authors only used Q-Learning, which fails to scale to large state spaces due to its tabular nature. In this work, we consider the MDP introduced by Garlapati et al. (2015) and show that it can be reduced to a simpler MDP with less degrees of freedom, i.e less hyperparameters, for which the optimal policy corresponds to the optimal DT with respect to a penalised accuracy metric, where the depth of the DT is penalised. We implement Deep RL algorithms DQN, Double DQN and Advantage Actor-Critic (AAC) to address large state spaces, and we discuss the benefits and drawbacks of these methods. In the experimental section, we demonstrate empirically the efficacy of both DQN and Double DQN while also showing the failure of AAC in solving the MDP.

2 Related Work

In their seminal work, Domingos and Hulten (2000) developed the VFDT algorithm that constructs HTs. This approach estimates a gain criterion (typically Entropy or Gini index) for each attribute as the data samples arrive and a statistical test, based on Hoeffding's inequality, is used to decide when to split each leaf and what attribute to use for the split. Because of their simplicity and ease of use, HTs are very popular among the data stream learning community Bifet Figuerol and Gavaldà Mestre (2009); Manapragada et al. (2018), however, as Rutkowski et al. (2012) show, Hoeffding's inequality is inadequate in this context since the gain criterion is not a sum of independent random variables, on the other hand, it satisfies the assumptions of McDiarmid's inequality, and Rutkowski et al. (2012) derived a new McDiarmid based statistical test. Although theoretically sound, The drawback of this method is its dependence on a loose concentration inequality, making the splitting process slow as it requires a large number of samples to make a confident decision. Hence, other tests were considered; Jin and Agrawal (2003) and Rutkowski et al. (2013) use a Normal approximation based with the Multivariate Delta method, which not only reduces the required number of samples, but also aims to optimise the true Entropy (or Gini index) unlike the previous methods that sought to optimise the expectation of the empirical Entropy (or Gini) without considering that this estimator is biased.

Although useful in some cases, the methods described above present no guarantee of optimality, in addition, as data samples arrive, the induced DTs can keep growing seeking purer leaves, and becoming more prone to overfitting; the standard evaluation metrics, such as prequential accuracy, fail to raise this issue. In this context, RL Sutton and Barto (2018), is a promising framework because of its sequential decision making nature that aims at maximising a long term objective (the expected cumulative reward). RL requires the definition of a Markov Decision Process (MDP), which is a tuple $(\mathcal{S}, \mathcal{A}, p, \gamma, r)$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $p\left(s'|s,a\right)$ the transition probability from state s to state s' when performing action a in state s, $r\left(s,a\right)$ the reward induced by action a in state s, and $0 \leq \gamma \leq 1$ is the discount factor. An agent interacts with its environment starting from an initial state by using a policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ and transitioning from a state to another according to $p\left(s'|s,a\right)$ until it reaches a final state at a time T. The objective is to find the policy π maximising the value function $\mathcal{V}^{\pi}\left(s\right) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^{t} r\left(S_t, \pi\left(S_t\right)\right) | S_0 = s; \pi\right]$ at all states; since $\mathcal{V}^{\pi}\left(s\right)$ satisfies the Bellman equation, it is usually more convenient to work with the Q-function $\mathcal{Q}^{\pi}\left(s,a\right) = r\left(s,a\right) + \gamma \sum_{s' \in \mathcal{S}} p\left(s'|s,a\right) \mathcal{V}^{\pi}\left(s'\right)$. Then, the optimal policy π^{*} is just the greedy policy with respect to the optimal Q-function $\mathcal{Q}^{*}$, i.e $\forall s \in \mathcal{S} : \pi^{*}\left(s\right) = \text{argmax}_{a}\, \mathcal{Q}^{*}\left(s,a\right)$. There have been few attempts at using RL in the context of online DTs, Preda (2007) considered the batch setting with a fixed training set E, and defined the state space $\mathcal{S}$ as the space of all possible partitions of E. However, $\mathcal{S}$ can be very large since $|\mathcal{S}| = 2^{|E|}$, in which case, tabular Q-Learning Watkins and Dayan (1992) cannot be used. Instead, Preda (2007) uses a linear approximation with handcrafted attributes containing different measures of the quality of the split Rokach and Maimon (2005) e.g Information gain, Gini index gain, and the discriminant power function. Unfortunately, it is not trivial

to extend this formulation to an online setting, the state space strongly depends on E and changes completely by adding or removing a data point. Garlapati et al. (2015) formulated a simple MDP that is naturally suited for data streams, and Blake and Ntoutsi (2018) built upon it to address concept drift. This MDP models DTs in an indirect way where a policy can be thought of as a DT. However, the link between solving the MDP and optimising some global metric, such as accuracy, was not clearly stated. We will rework this MDP, and show that solving it is equivalent to looking for the DT maximising a penalised accuracy metric with penalty of the DT depth.

3 Problem Formulation

We consider an Online Supervised Multi-class Classification problem with incrementally observed data from a stream. Each data point consists of an input $X = (X_1, \ldots, X_d)$ of $d \geq 1$ attributes and a predicted class $Y \in \{1, \ldots, l\}$ with $l \geq 2$. In this section, we formulate the MDP of Garlapati et al. (2015) and explain how each policy π is related to a DT T^π. Then we derive the value function $\mathcal{V}^\pi$ and explain how it can be written as an evaluation metric of T^π under certain assumptions.

3.1 MDP Formulation

State space: Given $X = (X_1, \ldots, X_d)$, a state $s \in \mathcal{S}$ is a vector of size d where each entry i, is either the value of X_i or describes that X_i is unobserved. For example, let $X = (1, 3, 0, 2, 5)$, let NaN denote an unobserved attribute and consider the state $s = (\text{NaN}, 3, 0, \text{NaN}, \text{NaN})$. s describes that X_2 and X_3 are observed and their respective values are 3 and 0 while attributes X_1, X_4, X_5 are, on the other hand, unobserved. The empty state consists of NaNs only and is always the initial state of an episode, it describes the fact that we have not yet observed any attribute of X. If each X_i is categorical with k_i possible values, then $|\mathcal{S}| = \prod_{i=1}^{d} (1 + k_i)$, and in the presence of numerical attributes, $\mathcal{S}$ is infinite.
Action space: In a state s, we can either query the value of an unobserved attribute or directly label the data point, these two types of actions are called query actions and report actions respectively. The action space is then defined as $\mathcal{A} = \{1, \ldots d, d+1, \ldots, d+l\}$ where $\{1, \ldots, d\}$ are the query actions and $\{d+1, \ldots, d+l\}$ the report actions.
Transitions: When taking action a in a state s:

- If a is a report action, the episode ends and the data point is classified according to a.
- If a is a query action, the next state observes all the already observed attributes at s in addition to the queried attribute. For example, let $X = (1, 0, 2, 0, 3)$ be a data point, $s = (\text{NaN}, 0, 2, \text{NaN}, 3)$ the current state and a the query action of the 4^{th} attribute, the next state is then $s' = (\text{NaN}, 0, 2, 0, 3)$ by observing the 4^{th} attribute of X.

Reward: The agent is rewarded positively when it reports the right class and negatively otherwise. For the policy to use a fewer number of attributes, a query action is rewarded negatively. We denote R_+ the positive reward of a right report action, R_- the negative reward of a wrong report action and λ the negative reward of any query action.
Episode: An episode starts upon the arrival of a new data point, the initial state is then initialized to the empty state and the agent's policy is run until a report action is taken.

3.2 Linking policies to DTs

Upon the arrival of a new data point X, the agent starts an episode from the empty state, then according to its policy π, it makes a sequence of query actions followed by a report action ending the episode. This process can be viewed as a descent down a DT T^π where each node represents a state and each query action represents a split of a node with respect to an attribute, the leaves of T^π are states where π takes a report action, we define the prediction of T^π at a leaf l as the report action taken by π at state (leaf) l. By construction, the mapping $\pi \mapsto T^\pi$ is bijective between the set of policies and the set of DTs. See Figure 1 for an illustration of the correspondence between π and T^π.

The following result provides an expression of $\mathcal{V}^\pi(\emptyset)$ in terms of the hyperparameters $\gamma, \lambda, R_+, R_-$, where $\emptyset$ is the empty state and γ the discount factor.

Theorem 3.1. *For the MDP defined in section 3.1 we have:*

$$\mathcal{V}^\pi(\emptyset) = A(\gamma, \lambda) + B(\gamma, R_+) + C(\gamma, R_-)$$

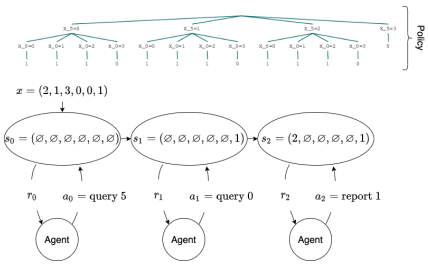

Figure 1: Agent's policy interpreted as a DT.

Where:

$$A(\gamma, \lambda) = \lambda \sum_l p(l) \sum_{j=0}^{|l|-1} \gamma^j$$

$$B(\gamma, R_+) = R_+ \sum_l \gamma^{|l|} \mathbb{P}[X \in l] \, \mathbb{P}[T^\pi(X) = Y | X \in l]$$

$$C(\gamma, R_-) = R_- \sum_l \gamma^{|l|} \mathbb{P}[X \in l] \, \mathbb{P}[T^\pi(X) \neq Y | X \in l]$$

$|l|$ *denotes the depth of leaf* l *with the root having depth* 0*.*

Proof. Given an input X, let $l^\pi(X)$ denote the leaf in T^π such that $X \in l^\pi(X)$, then we can write $\mathcal{V}^\pi(\emptyset)$ as follows:

$$\mathcal{V}^\pi(\emptyset) = \mathbb{E}\left[\sum_{j=0}^{|l^\pi(X)|-1} \gamma^j \lambda + \gamma^{|l^\pi(X)|} R_+ \mathbb{1}\{T^\pi(X) = Y\} + \gamma^{|l^\pi(X)|} R_- \mathbb{1}\{T^\pi(X) \neq Y\} \right]$$

Let's denote $A(\gamma, \lambda) = \mathbb{E}\left[\sum_{j=0}^{|l^\pi(X)|-1} \gamma^j \lambda\right]$, $B(\gamma, R_+) = \mathbb{E}\left[\gamma^{|l^\pi(X)|} R_+ \mathbb{1}\{T^\pi(X) = Y\}\right]$, $C(\gamma, R_-) = \mathbb{E}\left[\gamma^{|l^\pi(X)|} R_- \mathbb{1}\{T^\pi(X) \neq Y\}\right]$.

$$\begin{aligned}
A(\gamma, \lambda) &= \lambda \mathbb{E}\left[\sum_{j=0}^{|l^\pi(X)|-1} \gamma^j \right] \\
&= \lambda \mathbb{E}\left[\sum_l \sum_{j=0}^{|l|-1} \gamma^j \mathbb{1}\{X \in l\} \right] \\
&= \lambda \sum_l p(l) \sum_{j=0}^{|l|-1} \gamma^j \\
B(\gamma, R_+) &= R_+ \mathbb{E}\left[\gamma^{|l^\pi(X)|} \mathbb{1}\{T^\pi(X) = Y\} \right] \\
&= R_+ \mathbb{E}\left[\sum_l \gamma^{|l|} \mathbb{1}\{T^\pi(X) = Y\} \mathbb{1}\{X \in l\} \right] \\
&= R_+ \sum_l \gamma^{|l|} \mathbb{E}\left[\mathbb{1}\{X \in l\} \mathbb{E}\left[\mathbb{1}\{T^\pi(X) = Y\} | X \right] \right] \\
&= R_+ \sum_l \gamma^{|l|} \mathbb{E}\left[\mathbb{1}\{X \in l\} \mathbb{E}\left[\mathbb{1}\{T^\pi(l) = Y\} | X \right] \right] \\
&= R_+ \sum_l \gamma^{|l|} \mathbb{P}[X \in l] \, \mathbb{P}[T^\pi(X) = Y | X \in l] \\
C(\gamma, R_-) &= R_- \sum_l \gamma^{|l|} \mathbb{P}[X \in l] \, \mathbb{P}[T^\pi(X) \neq Y | X \in l]
\end{aligned}$$

$T^\pi(X)$ is the same for all X in the same leaf, hence why we used the notation $T^\pi(l)$ for $T^\pi(X)$ when $X \in l$. To derive $C(\gamma, R_-)$, we followed the same steps as for $B(\gamma, R_+)$.

□

Recall that $0 \leq \gamma \leq 1, \lambda \leq 0, R_- \leq 0, R_+ \geq 0$, and notice that if $R_- < 0$, the term $C(\gamma, R_-)$ can penalise low depth DTs more than higher depth ones due to the terms $\gamma^{|l|}$ being larger for smaller depths $|l|$. We seek accurate DTs with the lowest depths possible because of their interpretability and lower risk of overfitting, thus we choose $R_- = 0$. Since the MDP is episodic, we can set $\gamma = 1$ without worrying about the divergence of the cumulative reward, then by also setting $R_+ = 1$ we derive the following corollary that reduces the MDP to one hyperparameter only λ instead of the four $\gamma, R_+, R_-, \lambda$, this result is also important because it links π and T^π by stating that $\mathcal{V}^\pi(\emptyset)$ is equal to a penalised accuracy metric of T^π, which renders solving the MDP equivalent to finding the optimal DT with respect to a global objective.

Corollary 3.2 (Linking π and T^π)**.** *Consider the MDP defined in section 3.1 such that $\gamma = 1, R_- = 0, R_+ = 1$, then we have:*

$$\mathcal{V}^\pi(\emptyset) = \mathbb{P}\left[T^\pi(X) = Y\right] - \lambda \sum_{l \in \mathcal{L}(T^\pi)} p(l)\,|l| \tag{1}$$

Where $\mathcal{L}(T^\pi)$ is the set of leaves of T^π and $p(l) = \mathbb{P}[X \in l]$.

Since we seek accurate and interpretable DTs, we want solutions of low depth, more specifically, we want DTs where the most likely leaves l (with high $p(l)$) have low depth $|l|$. As such, it is natural to define the complexity of a DT T^π as its expected depth $\mathbb{E}[|l^\pi(X)|]$, and seek the most accurate T^π with lowest complexity. Then, according to Corollary 3.2, solving the MDP can be viewed as a constrained optimisation problem over the space of DTs where the complexity defined above is penalised since $\sum_{l \in \mathcal{L}(T^\pi)} p(l)\,|l| = \mathbb{E}[|l^\pi(X)|]$. If we are only interested in the accuracy of the DTs with no regard for their complexities, we choose $\lambda = 0$, which renders $\mathcal{V}^\pi(\emptyset) = \mathbb{P}\left[T^\pi(X) = Y\right]$.

4 Solving the MDP

As the number of attributes and their categories (for the categorical ones) increases, the state space becomes quickly too large for Q-Learning to be practical, and this issue is even worse in the presence of numerical attributes since the state space becomes infinite. Therefore, we turn to Approximate RL algorithms as they are scalable. In this work, we implement DQN Mnih et al. (2013), Double DQN (DDQN) Van Hasselt et al. (2016) and Advantage Actor-Critic (AAC) Konda and Tsitsiklis (2000) to solve the RL problem. We discuss the benefits and drawback of these methods in our context and compare them empirically to the Hoeffding Trees (HTs) spanned by the VFDT algorithm.

4.1 DQN

DQN approximates $\mathcal{Q}^*$ with a neural network $\mathcal{Q}^\theta$, where $\theta \in \mathbb{R}^p$. We use a multi-layer perceptron (MLP) for $\mathcal{Q}^\theta$ along with a target network $\mathcal{Q}^{\theta'}$, of similar architecture, for more consistent target values over time, this has the benefit of stabilising training. Given a batch of transitions[1] $(s_i, a_i, r_i, s'_i)_{i=1}^N$, the loss function of $\mathcal{Q}^\theta$ is:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(r + \gamma \max_{a' \in \mathcal{A}} \mathcal{Q}^{\theta'}(s', a') - \mathcal{Q}^\theta(s, a) \right)^2$$

For a more stable training, DDQN decouples the evaluation and selection process of actions in the loss definition:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left[r + \gamma \mathcal{Q}^{\theta'} \left(s', \operatorname*{argmax}_{a' \in \mathcal{A}} \mathcal{Q}^\theta(s', a') \right) - \mathcal{Q}^\theta(s, a) \right]^2$$

[1] A transition refers to a 4-tuple (state, action, reward, next state)

Algorithm 1 DQN for Online Classification with DTs

K the number of episodes, N the batch size, $0 < \gamma < 1$ the discount factor
Initialize replay buffers `memory_queries` and `memory_reports` by filling them in a pre-training phase
Initialize $\mathcal{Q}^{\theta}$, $\mathcal{Q}^{\theta'}$ as per Glorot and Bengio (2010)
$\tau = 0$
for $k = 1$ to K **do**
  Data point x arrives with its label y
  Initialize s to the empty state.
  while s has an unobserved attribute **do**
    $\tau \leftarrow \tau + 1$
    Randomly sample a report action a
    Take action a and receive reward r
    Store (s, a, r, s) in `memory_reports`
    Sample query action a with an ϵ-greedy policy
    Take action a, receive reward r and next state s'
    Store (s, a, r, s') in `memory_queries`
    Sample $\{(s_i, a_i, r_i, s'_i)\}$ from `memory_queries`
    Compute the loss on this batch of size $N/2$

$$\frac{1}{n}\sum_{i=1}^{n}\left(r_i + \gamma \max_{a' \in \mathcal{A}} \mathcal{Q}^{\theta'}\left(s'_i, a'\right) - \mathcal{Q}^{\theta}\left(s_i, a_i\right)\right)^2$$

    Backpropagate the loss to update θ
    Sample $\{(s_i, a_i, r_i, s'_i)\}$ from `memory_reports`
    Compute the loss on this batch of size $N/2$

$$\frac{1}{n}\sum_{i=1}^{n}\left(r_i - \mathcal{Q}^{\theta}\left(s_i, a_i\right)\right)^2$$

    Backpropagate the loss to update θ
  end while
  Sample a from the report actions
  Take action a and receive reward r (end of episode)
  Store (s, a, r, s) in `memory_reports`
  if $\tau = \tau_{\text{max}}$ **then**
    $\theta' \leftarrow \theta$
    $\tau \leftarrow 0$
  end if
end for

In our context, it is interesting to compare both algorithms empirically to see if there is any merit to using Double DQN instead of standard DQN.

We sample the transitions $(s_i, a_i, r_i, s'_i)_{i=1}^{N}$ from a replay buffer that is pre-filled in a pretraining phase. However, since an episode ends when taking a report action, a naive ϵ-greedy policy makes states with many observed attributes (which correspond to deeper nodes in a DT T^{π}) less likely to be visited; this runs the risk of prematurely ending episodes and leading the agent to be stuck in a suboptimal subspace of parameter θ. This problem has been addressed in Garlapati et al. (2015) with a strategy called truncated exploration. In DQN, we combine this idea with the replay buffer as follows, we explore the environment by only taking query actions (according to an ϵ-greedy policy) until all attributes are observed, then at each state of this sampled trajectory of transitions, we evaluate a random report action, and we store the induced transitions with query actions in a memory buffer `memory_queries`, and transitions with report actions in another buffer `memory_reports`, then, during the learning phase, we sample half of the batch $(s, a, r, s')_{i=1}^{N}$ from `memory_queries` and the remaining half from `memory_reports`. This sampling strategy, along with relatively big

batch sizes, 512 transitions, has the benefit of sampling diversified transitions which in turn estimates the gradient more accurately for backpropagation, see Algorithm 1.

In all of our experiments, for both $\mathcal{Q}^{\theta}$ and $\mathcal{Q}^{\theta'}$, we use a shallow MLP with two fully connected hidden layers of 32 nodes with a ReLU activation function and an output layer with the size of the action space $d + l$ (see section 3 for the notation). We encode the input states with One-Hot encoding where, each categorical attribute X_i, with k_i possible values, is encoded with a dummy variable of dimension $k_i + 1$ in which each entry $1 \leq j \leq k_i$ is the indicator variable $\mathbb{1}\{X_i = j\}$ and the entry $k_i + 1$ is the indicator variable indicating whether X_i is observed or not.

4.2 Advantage Actor-Critic (AAC)

Unlike DQN, AAC is an on-policy algorithm, which means that it can only learn from samples simulated by the current policy. This characteristic is interesting because, in concept drift situations where the joint distribution of X, Y changes with time, sampling transitions from the replay buffer may reflect old distributions, which is obviously unwanted. AAC is an Actor-Critic algorithm Konda and Tsitsiklis (2000), it uses two estimators, the Actor for the policy $\pi_\theta(s)$ and the Critic for the value $\mathcal{V}^{\pi_\theta}(s)$, where s is a state. AAC updates the Actor parameters θ with the policy gradient theorem Sutton et al. (2000), and the critic parameters are updated by using a return estimate sampled with the current policy.

Being on-policy comes with the challenge of poor exploration resulting in noisy gradient estimates that can hinder training. Indeed, truncated exploration cannot be performed in this case, and the absence of a replay buffer severely limits the variety of the transitions we learn from. To help with exploration, an additional entropy loss is generally considered in the literature, its purpose being to prevent the stochastic policy π_θ from becoming too concentrated. However, this is not in general enough to solve the noisy gradients problem and we will show how the algorithm's performance suffers from this poor exploration in the experiments section.

In practice, we use one neural network with parameter $\theta \in \mathbb{R}^p$ with two output heads, the Actor $\pi_\theta(.|s)$ and the Critic $\mathcal{V}^{\theta}(s)$. Similarly to DQN, we consider a shallow MLP with two fully connected hidden layers of 32 nodes and ReLU activation function. The Actor output layer is of size $d + l$ and the critic output layer is of size 1.

5 Experiments

In this section, we carry experiments on real-world datasets taken from the UCI repository. We first describe the experimental setup, then we present and discuss our results.

5.1 Experimental setup

We evaluate DQN, Double DQN (DDQN), AAC and Hoeffding Trees with a prequential accuracy, i.e with the arrival of a new training data point (x, y), we evaluate the prediction of our current model on (x, y) (before learning from it) and we keep a moving average of the accuracy on the last 1000 data points seen. We consider the following datasets: Adult, Nursery, Mushroom, Skin, Poker from the UCI repository. These datasets are frequently used in the literature of data stream learning and they have different types of attributes; Mushroom, Poker and Nursery datasets have only categorical attributes while the Skin dataset has numerical attributes and The Adult dataset has mixed categorical-numerical attributes; in fact, even though the Skin dataset has numerical attributes, since these take integer values from 0 to 255, we will also carry an alternative experiment where these attributes are considered categorical and states are encoded as such (resulting in very high dimensional input vectors for our neural networks). When dealing with numerical attributes, we first preprocess the data by centering it and normalizing it as this helps in stabilizing training, indeed some attributes of the Adult dataset for example, take very high values and can induce high magnitude gradients that could overshoot the optimal parameter θ. Without this preprocessing, we have noticed large spikes in the loss function of all our RL methods when applied to the Adult dataset, these spikes signal points of no return in training where the agent's performance can no longer recover because the parameter θ likely ended up in a flat surface of $\mathbb{R}^p$ with respect to the loss, which leads to a stagnation and suboptimal convergence.

Algorithm 2 AAC for Online Classification with DTs

K the number of episodes.
$\lambda \in [0, 1]$ the exploration hyperparameter
n_step the number of iterations between two updates
Initialize the Actor-Critic network with two output heads, one for π_θ with a dimension equal to the number of actions and the other is uni-dimensional for $\mathcal{V}^\theta$
for $k = 1$ to K **do**
  Data point x arrives with its label y.
  $t \leftarrow 1$
  $t_{\text{start}} \leftarrow t$
  Initialize s_t to the empty state.
  while episode is unfinished **do**
    Perform action $a_t \sim \pi_\theta(.|s_t)$ and receive reward r_t and next state s_{t+1}
    $t \leftarrow t + 1$
    if $t - t_{\text{start}} =$ n_step or s_t is terminal **then**
      $R \leftarrow 0$ if s_t is terminal and $\mathcal{V}^\theta(s_{t+1})$ otherwise.
      for $i \in \{t-1, \ldots, t_{\text{start}}\}$ **do**
        $\mathcal{A}^{(i)}(\theta) \leftarrow r_i + \gamma R - \mathcal{V}^\theta(s_i)$
        $\mathcal{L}^{(i)}_{\text{critic}}(\theta) \leftarrow \frac{1}{2}\left(\mathcal{A}^{(i)}(\theta)\right)^2$
        $\mathcal{L}^{(i)}_{\text{actor}}(\theta) \leftarrow -\log \pi_\theta(a_i|s_i)\,\mathcal{A}^{(i)}(\theta')$
        $\mathcal{L}^{(i)}_{\text{entropy}}(\theta) \leftarrow \sum_a \pi_\theta(a|s_i) \log \pi_\theta(a|s_i)$
        $R \leftarrow r_i + \gamma R$
      end for
      Construct the global loss:

$$\mathcal{L}(\theta) = \frac{1}{t - t_{\text{start}}} \sum_{i=t_{\text{start}}}^{t-1} \left(\mathcal{L}^{(i)}_{\text{critic}} + \mathcal{L}^{(i)}_{\text{actor}} + \lambda \mathcal{L}^{(i)}_{\text{entropy}}\right)$$

      Update $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$
    end if
    $t_{\text{start}} \leftarrow t$
  end while
end for

5.2 Results

Figure 2 and table 1 present our experimental results. We first notice that DQN and DDQN do not yield a poor performance in any experiment unlike AAC and HT, especially on the Poker dataset, where AAC and HT result in surprisingly low accuracies. This success of DQN and double DQN is likely due to their stable training induced by the replay buffer and the adaptation of truncated exploration in the learning phase, this stability is depicted in figure 3 by a steady decrease in the losses, notice that for the Poker dataset, DDQN yields a better decrease in the loss compared to DQN, and this translates directly into a superior performance (see table 1). However, other than the Poker dataset, DDQN does not seem to provide a significant benefit over standard DQN, thus we would recommend to use DDQN only in a situation where an unstable DQN training is noticed.

AAC is the worst performing method in our experiments. This is mainly due to a poor exploration coming from a lack of variety in the transitions used to estimate the Actor and Critic gradients as is explained in section 4. Unlike DQN and DDQN, AAC has a very unstable training, which is clearly observed in Figure 4, the evolution of its loss is either very noisy or gets prematurely stuck in a local optimum. In order to help AAC explore states with many observed attributes, which are rarely seen since we cannot implement Truncated Exploration with an on-policy method, we tried to constrain the agent to perform a minimum number of query actions before being able to take a report action, this heuristic sets a minimum episode length. However, despite using this strategy, the agent still poorly explores high depth states as it only took a maximum of one or two query actions before taking a report action. This poor exploration is a severe limitation of AAC in particular and on-policy methods in general.

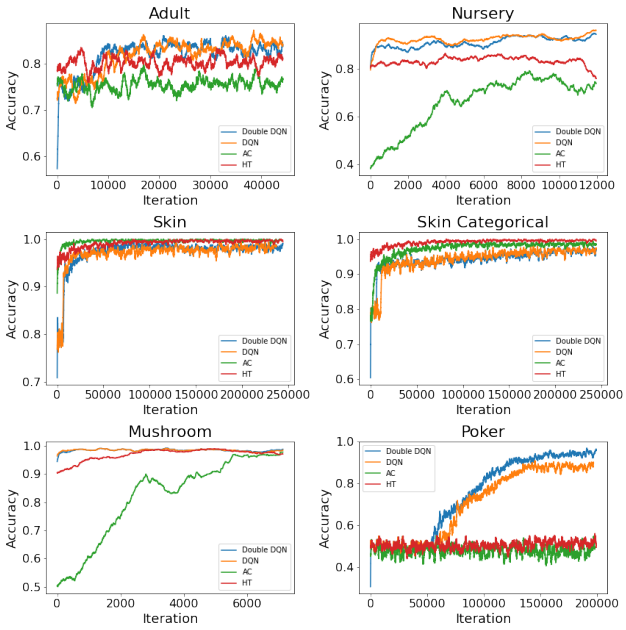

Figure 2: Prequential accuracy of each algorithm on the different datasets described above

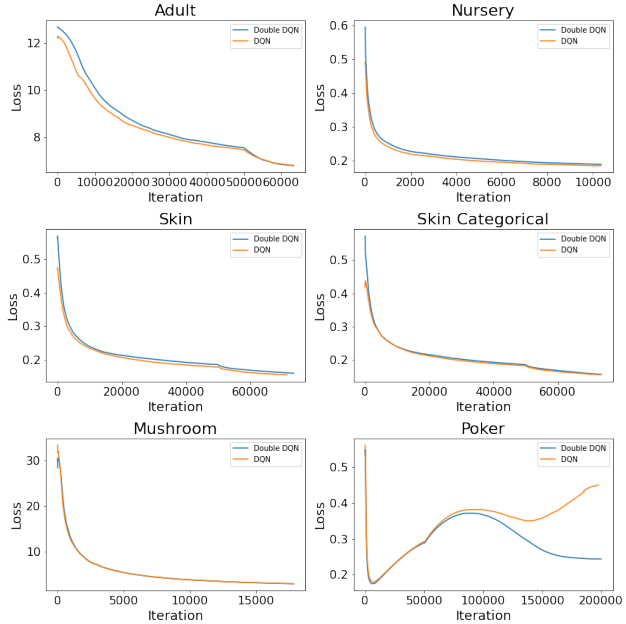

Figure 3: Evolution of the loss for DQN and Double DQN during training.

The only experiment where HT outperformed the RL algorithms is the one carried on the Skin dataset with its attributes considered categorical. Notice however, that this version of the dataset hurts the performance of our algorithms, this is mainly due to the high dimension of the encoded input states that may require larger neural network structures. Another interesting result regarding the performance of HT is its very poor accuracy on the Poker dataset compared to DQN and Double DQN, this depicts how these greedy methods can get stuck in locally optimal tree structures unlike the alternative RL methods.

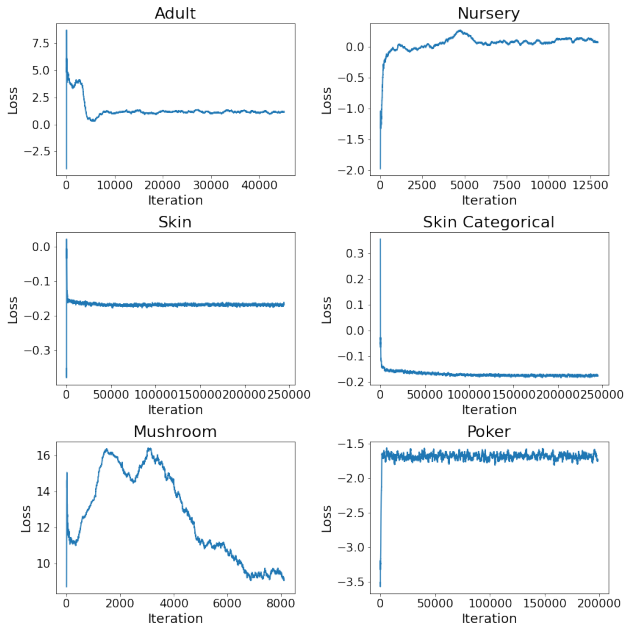

Figure 4: Evolution of the loss for AAC during training.

Table 1: Accuracy of the final model of each algorithm on the different datasets.

	DQN	DDQN[2]	AAC	HT
Adult	**84.4%**	83.7%	78/1%	82.7%
Nursery	**95.7%**	92.1%	84.4%	79.1%
Mushroom	**98.6%**	**98.6%**	**98.6%**	98.2%
Skin	97.7%	98.2%	**99.9%**	99.5%
Skin Categorical	96.4%	96.3%	98.7%	**99.5%**
Poker	86.7%	**94.4%**	41.8%	51%

6 Conclusion and Future Work

We formulated the problem of finding the most accurate DT, in an online setting, as solving an MDP, and we implemented Deep RL methods DQN, Double DQN and Advantage Actor-Critic (AAC), then we tested our methods empirically on real world data sets and compared them with the VFDT algorithm by evaluating their prequential accuracies, these experiments showcased that DQN and Double DQN outperform AAC and HT and we justified this result by the coupling of the replay buffer with truncated exploration, allowing for more accurate gradient estimates; AAC on the other hand suffers greatly from this issue, the impossibility of coupling it with truncated exploration drives the agent algorithm down the path of premature suboptimal convergence.

In a future work, we aim to use prioritised experience replay Schaul et al. (2015) with DQN as a way of addressing the issue of concept drift; and for AAC, a promising idea would be to use the off-policy policy gradient theorem Degris et al. (2012) with a handcrafted exploratory policy along with deterministic Actor policies as it was shown to be successful in many applications Silver et al. (2014), Lillicrap et al. (2015).

References

Bifet Figuerol, A. C. and Gavaldà Mestre, R. (2009). Adaptive parameter-free learning from evolving data streams.

Blake, C. and Ntoutsi, E. (2018). Reinforcement learning based decision tree induction over data streams with concept drifts. In *2018 IEEE International Conference on Big Knowledge (ICBK)*, pages 328–335. IEEE.

Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. (1984). *Classification and regression trees*. CRC press.

Degris, T., White, M., and Sutton, R. S. (2012). Off-policy actor-critic. *arXiv preprint arXiv:1205.4839*.

Domingos, P. and Hulten, G. (2000). Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80.

Garlapati, A., Raghunathan, A., Nagarajan, V., and Ravindran, B. (2015). A reinforcement learning approach to online learning of decision trees. *arXiv preprint arXiv:1507.06923*.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.

Jin, R. and Agrawal, G. (2003). Efficient decision tree construction on streaming data. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 571–576.

Konda, V. R. and Tsitsiklis, J. N. (2000). Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014.

Laurent, H. and Rivest, R. L. (1976). Constructing optimal binary decision trees is np-complete. *Information processing letters*, 5(1):15–17.

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

Manapragada, C., Webb, G. I., and Salehi, M. (2018). Extremely fast decision tree. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1953–1962.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Preda, M. (2007). Adaptive building of decision trees by reinforcement learning. In *Proceedings of the 7th Conference on 7th WSEAS International Conference on Applied Informatics and Communications*, volume 7, pages 34–39. Citeseer.

Quinlan, J. (2014). *C4. 5: programs for machine learning*. Elsevier.

Read, J. (2018). Concept-drifting data streams are time series; the case for continuous adaptation. *arXiv preprint arXiv:1810.02266*.

Rokach, L. and Maimon, O. (2005). Top-down induction of decision trees classifiers-a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 35(4):476–487.

Rutkowski, L., Jaworski, M., Pietruczuk, L., and Duda, P. (2013). Decision trees for mining data streams based on the gaussian approximation. *IEEE Transactions on Knowledge and Data Engineering*, 26(1):108–119.

Rutkowski, L., Pietruczuk, L., Duda, P., and Jaworski, M. (2012). Decision trees for mining data streams based on the mcdiarmid's bound. *IEEE Transactions on Knowledge and Data Engineering*, 25(6):1272–1279.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063.

Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.