Interpretable Policies for Reinforcement Learning by Empirical Fuzzy Sets[1]

Jianfeng Huang [a, *], Plamen P. Angelov [b], Chengliang Yin [a]

[a] Shanghai Jiao Tong University, School of Mechanical Engineering, 200240 Shanghai, China

[b] Lancaster University, School of Computing and Communications, LA1 4WA Lancaster, UK

Abstract—This paper proposes a method and an algorithm to implement interpretable fuzzy reinforcement learning (IFRL). It provides alternative solutions to common problems in RL, like function approximation and continuous action space. The learning process resembles that of human beings by clustering the encountered states, developing experiences for each of the typical cases, and making decisions fuzzily. The learned policy can be expressed as human-intelligible IF-THEN rules, which facilitates further investigation and improvement. It adopts the actor-critic architecture whereas being different from mainstream policy gradient methods. The value function is approximated through the fuzzy system AnYa. The state-action space is discretized into a static grid with nodes. Each node is treated as one prototype and corresponds to one fuzzy rule, with the value of the node being the consequent. Values of consequents are updated using the Sarsa(λ) algorithm. Probability distribution of optimal actions regarding different states is estimated through Empirical Data Analytics (EDA), Autonomous Learning Multi-Model Systems (ALMMo), and Empirical Fuzzy Sets (**ε**FS). The fuzzy kernel of IFRL avoids the lack of interpretability in other methods based on neural networks. Simulation results with

Please cite this article as:

J. Huang, P.P. Angelov and C. Yin, Interpretable policies for reinforcement learning by empirical fuzzy sets. Engineering Applications of Artificial Intelligence (2020) 103559, https://doi.org/10.1016/j.engappai.2020.103559.

[*] Corresponding author at: School of Mechanical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China

E-mail address: 515064@sjtu.edu.cn (Jianfeng Huang)

four problems, namely Mountain Car, Continuous Gridworld, Pendulum Position, and Tank Level Control, are presented as a proof of the proposed concept.

Keywords—interpretable fuzzy systems, reinforcement learning, probability distribution learning, autonomous learning systems, AnYa type fuzzy systems, Empirical Fuzzy Sets

I. INTRODUCTION

Reinforcement learning (RL) has attracted extensive research interest in recent years. It is mainly for solving decision- making problems in Markovian processes (Sutton and Barto, 2018). The goal is to find out the mapping from states to actions which yields maximal return. Here, "return" is defined as optionally discounted cumulative rewards within a finite or infinite time horizon. Various algorithms have been developed to solve RL problems. At the early stage, policies are derived through evaluation of actions values, like in the classic tabular Q-learning (Watkins, 1989) and Sarsa (Rummery and Niranjan, 1994). However, it is also possible to make decisions directly through a parameterized function, like in (Silver et al., 2014; Sutton et al., 2000). State-of-the-art researches combine deep learning (DL) (Goodfellow et al., 2016; Lecun et al., 2015) with RL to attain powerful algorithms like Deep Q-Learning Networks (DQN) (Antonoglou et al., 2015; Mnih et al., 2013) which is able to play Atari games at human level and Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015) which can be used for high-dimensional continuous action space.

Although great progress in both theory and applications of RL has been achieved, few researches are observed dealing with improving interpretability of policies produced by existing algorithms. In circumstances where safety is critical, like bio-medicine, the lack of interpretability makes the application of RL unacceptable (Maes et al., 2012; Verma et al., 2018). Introduction of deep neural networks (DNN) to deal with high-dimensional state-action space further deteriorates the problem, since DNN is known to be black-box.

Current attempts in deriving interpretable policies are characterized by a) expressing them as parameterized forms like fuzzy rules (Hein et al., 2017; Mucientes and Casillas, 2007; Samsudin et al., 2011), mathematical formulas (Hein et al., 2018; Maes et al., 2012), domain specific programming language (Verma et al., 2018), and b) using optimization methods like Particle Swarm Optimization (PSO) (Hein et al., 2017), genetic algorithms (GA) (Hein et al., 2018; Samsudin et al., 2011), ant colony optimization (ACO) (Mucientes and Casillas, 2007), or searching algorithms (Maes et al., 2012; Verma et al., 2018) to determine the parameters. They can be further categorized into two groups according to the objective of the optimization/searching problem. In (Hein et al., 2018, 2017), an environment model is established using neural networks (NN), based on historical state-action-reward trajectories from the real environment. Objective/fitness function of the optimization problem is then expressed as the weight-average return of all initial states within the RL framework. Here, the role of RL is solely on providing the objective/fitness/scoring function, rather than updating policy parameters. This kind of methods are typical model-based ones since they require the availability of environment models. Therefore, they are only applicable when the system dynamics are relatively easy to model. Another approach is to firstly learn a high-performance whereas uninterpretable policy (also referred to as "oracle" or nominal policy) through state-of-the-art methods like DQN, and then search for parameters that minimize the differences between the behavior of the nominal policy and the parameterized one (Verma et al., 2018). Inspired by imitation learning (Ross et al., 2010; Schaal, 1999), this method provides policy interpretability by making one in the interpretable form to imitate another uninterpretable. (Hein et al., 2018) compared these two approaches with different objectives in parameter optimization/searching, and found that the one that optimizes the RL return directly actually performs better.

All attempts above fail to learn interpretable policies online, in a per-step manner. Rather, they are all per-batch. This means that policy parameters won't be updated until the end of an episode. Furthermore, for either the return optimization or the policy imitation approaches, the final attained policy is fixed once the offline learning is finished. If the environment changes, the whole set of policy parameters have to be relearned. In other words, these algorithms are non-adaptive. Another disadvantage is that whereas the parameters can be learned automatically, the structure of the policy has to be manually specified a priori, like the number of membership functions (Mucientes and Casillas, 2007), the number of the rules (Hein et al., 2017; Mucientes and Casillas, 2007; Samsudin et al., 2011), the complexity of the mathematical formulas (Hein et al., 2018; Maes et al., 2012), the atoms and operators in the language for policy representation (Verma et al., 2018), etc. Such decisions are problem-specific and are usually hard to made, which implies the necessity of trial and error. These drawbacks are shared by most of existing algorithms for interpretable RL. There are exceptions, though. In (Chia-Feng Juang and Chia-Hung Hsu, 2009), interval type-2 fuzzy sets are used in antecedent parts of the fuzzy rules, which are online generated automatically through a clustering algorithm. The algorithm also partitions the input space to reduce the number of rules. Consequent part of each rule is updated using both Q-learning and ACO, the former of which is per-step whereas the latter is per-episode. Thus, the algorithm is capable of online learning both the structure and the parameters automatically. However, discussions in (Chia-Feng Juang and Chia-Hung Hsu, 2009) are restricted to the problem of wall-following control of a mobile robot, in which the action space is discrete and univariate, and the number of action candidates is finite and small. Furthermore, the policy is solely action-value based, and therefore covers only deterministic cases (one state is mapped to exactly one action). In circumstances where effects of function approximation are significant, stochastic policy may provide better optimality than deterministic ones (Sutton and Barto, 2018).

The term “interpretability”, though widely mentioned in literature on machine learning, has not yet been well defined (Lipton, 2018; Maes et al., 2012). In (Maes et al., 2012), interpretability of the policy is indicated through the Kolmogorov complexity, which is related to the number of symbols used in a certain description language. Generally speaking, it is more a qualitative metric than a quantitative one (Verma et al., 2018). Here, we take a practical perspective: a policy is regarded as “interpretable” if it satisfies:

a) users are able to develop intuitive insights about the interactive process between the agent and the environment. More specifically, this means that:

- the mapping between states/observations and actions should be expressed *explicitly* in a human-readable manner, rather than through black-box representations like NN;
- there should be some inductive procedures in the policy derivation, so as to condense the results and make them more tractable.

b) the form of policy should facilitate integration of priori knowledge as well as modification of the algorithmic results according to expertise or application requirements.

Fuzzy systems, dating back to (Mamdani and Assilian, 1975; Takagi and Sugeno, 1993; Zadeh, 1965), are appropriate candidates for such missions. Fuzzy controllers have been widely used in the past decades and welcomed by engineers, partly due to the fact that the control laws can be conveniently expressed as interpretable IF-THEN rules. Besides, fuzzy systems are universal approximators (Buckley, 1993; Kosko, 1994; Wang and Mendel, 1992) just like NN and therefore can be used for approximation of value functions in RL (Jin, 2000; Nauck and Kruse, 1998).

This paper proposes a method and an algorithm to implement interpretable fuzzy reinforcement learning (IFRL). It adopts the actor-critic architecture and consists of two components, namely the *value function approximator* and the *optimal policy estimator*. The former is based on the recent fuzzy systems AnYa. Value of a certain state-action pair is estimated as fuzzy ensemble of those of the predefined prototypes. The latter is a probability distribution learner within the framework of Autonomous Learning Multi-Model Systems (ALMMo) and Empirical Data Analytics (EDA). Generalization of the learned distribution between different states is achieved through Empirical Fuzzy Sets (**ε**FS). Cases of multivariate action space are handled through a hierarchical learning approach. Compared to other methods for interpretable RL, the proposed IFRL is model-free and learns online in a sample-by-sample or step-by-step manner. As a result, it is able to react to the change of the environment adaptively in real time. This is possible because IFRL does not rely on offline optimization or searching to derive the policy parameters. Rather, they are obtained directly from the learned probability distribution. It is applicable to continuous and multivariate action space, whereas being different from mainstream policy gradient methods. The policy learned is expressed stochastically, which is sometimes more favorable than the deterministic ones under the function approximation setting. Compared to classic tabular or state-of-the-art DNN-based algorithms, the main advantage of IFRL is that it produces policies as human-intelligible IF-THEN rules, which is convenient for integration of priori knowledge as well as further investigation and improvement. Numerical experiments on four RL problems are conducted and the results are presented as a proof of the concept.

The rest of this paper is organized as follows. Section II introduces the proposed IFRL structure. The *value function approximator* based on AnYa is discussed in Section III. The *optimal policy estimator* based on ALMMo, EDA, and **ε**FS is discussed in Section IV. Section V presents simulation results and Section VI gives the conclusion.

II. IFRL Structure

As with other actor-critic algorithms, IFRL is made up of two components, namely the *value function approximator* and the *optimal policy estimator*. However, the mechanism of generating actual behaviors and the learning process of optimal policy in IFRL are different from mainstream policy-gradient methods. Common practice of the latter is to produce actual behaviors directly from a *parameterized policy function* describing the probability of selecting a certain action under a certain state. The value function can be used to aid the learning process of the policy function, but is not required for selecting actions (Sutton and Barto, 2018). The policy function is updated by gradient-based methods to maximize returns.

In the proposed IFRL, actions that are actually carried out come from two different sources. The first are the *advised actions* from the function approximator by comparing values of all candidates. The second are the *inferred actions* from the policy estimator which reflects the distribution of advised actions. The two sources can be combined in different ways, e.g. switching to one with a certain probability. The policy estimator differs from policy gradient methods in that it learns the policy from observed samples empirically, rather than through optimization techniques like gradient descent. Details are to be given in Section IV.

Candidates evaluated by the function approximator are also from two different sources. The first are the randomly selected actions from predefined intervals across the whole range of each dimension ($A_{candidates1}$ in Fig. 1). For example, velocity of vehicles typically ranges from 0 to 200 km/h, which can be divided into 20 intervals with the interval length being 10 km/h. Thus, the first interval will be 0-10 km/h, and the second 10-20 km/h, and so on. At each time step for decision making, within each interval, one sample of velocity is randomly selected for evaluation. This approach is aiming to enable sparse, coarse, and fast exploration of the action space, and identify regions that are worth further investigation. Therefore, the interval length can be set quite large, which facilitates computation and memory reduction. The second source of candidates are actions randomly selected from the neighborhood of the inferred action ($A_{candidates2}$ in Fig. 1). They are to enable finer exploration in the region that is promising for optimal actions. The candidate with the largest estimated value is output as the advised action by the value function approximator.

After the current action A_t is executed, the agent moves to a new state S_{t+1} and receives a reward R_{t+1}. The same procedure is repeated to determine the behavior A_{t+1} for the new state. The Sarsa(λ) algorithm is then used to update the value of the last state-action pair (S_t, A_t). The policy estimator is updated each time the advised action is determined.

Complete block diagram of the proposed IFRL is shown in Fig. 1.

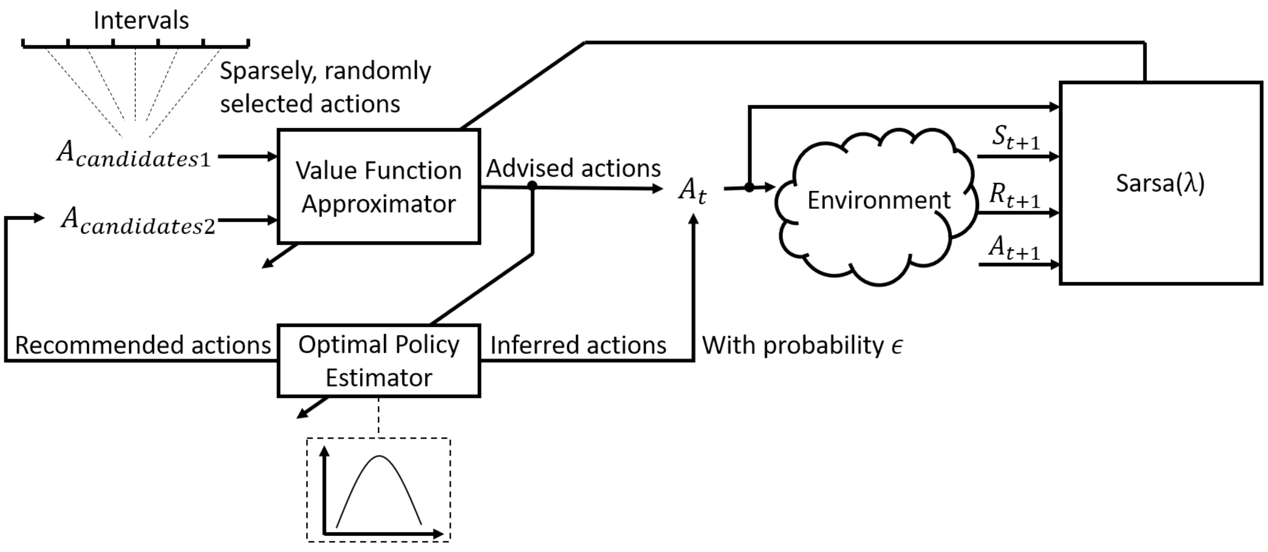


Fig. 1. Block diagram of IFRL.

III. Value function approximator

The *value function approximator* is responsible for evaluating values of different state-action pairs, and providing samples for the *optimal policy estimator*. We use AnYa for this purpose. This section discusses the framework of AnYa as well as the calculation and update of the firing strengths and consequents.

A. AnYa Framework

The *value function approximator* is based on the fuzzy rule-based system AnYa (Angelov, 2012; Angelov and Yager, 2012, 2011) within the EDA framework (Angelov et al., 2016; P. Angelov et al., 2017; Plamen P Angelov et al., 2017). The approximated value of a certain state-action pair is determined by constructing a set of fuzzy rules, each of which can be written as

$$\text{IF } \begin{pmatrix}\mathbf{S}\\\mathbf{A}\end{pmatrix} \sim \begin{pmatrix}\mathbf{S}_\mathbf{j}^*\\\mathbf{A}_\mathbf{j}^*\end{pmatrix} \text{ Then } \hat{q}(\mathbf{S},\mathbf{A}) = C_j \tag{1}$$

where $\mathbf{S}$ and $\mathbf{A}$ are the state and action variable, respectively. $\begin{pmatrix}\mathbf{S}_\mathbf{j}^*\\\mathbf{A}_\mathbf{j}^*\end{pmatrix}$ denotes the j^{th} prototype, $j = 1,2,\dots,N$ where N denotes the total number of prototypes. A prototype represents a typical case. The "∼" can be interpreted as "being close to". $\hat{q}(\mathbf{S},\mathbf{A})$ is the estimated value of $\begin{pmatrix}\mathbf{S}\\\mathbf{A}\end{pmatrix}$. C_j is the consequent part of the rule. Finally, the estimation that is used is calculated as the weighted average of all consequents

$$\hat{q}(\mathbf{S},\mathbf{A}) = \boldsymbol{\lambda}^{\mathrm{T}}\mathbf{C} \tag{2}$$

where $\boldsymbol{\lambda}$ is the firing strength of each rule

$$\boldsymbol{\lambda} = [\lambda_1 \quad \lambda_2 \quad \dots \quad \lambda_N]^{\mathrm{T}} \tag{3}$$

and $\mathbf{C}$ is the vector of consequents

$$\mathbf{C} = [C_1 \quad C_2 \quad \dots \quad C_N]^{\mathrm{T}} \tag{4}$$

B. State-action Space Discretization

State and action variables may come with multiple dimensions, and may be a hybrid of continuous and discrete components. For example, when describing the status of a vehicle on a straight road, we may consider both, the moving direction and the velocity. The former is discrete with two enumerations: forward or backward. The latter is continuous with the possible range from 0 to 200 km/h. The approach here is to transform each continuous dimension into the discrete one with a certain step, and form a static grid. A typical one is shown in Fig. 2. The smaller the steps, the finer the hybrid/continuous space is approximated, whereas the number of nodes in the grid is larger and more computations are needed.

Each node is treated as one prototype.

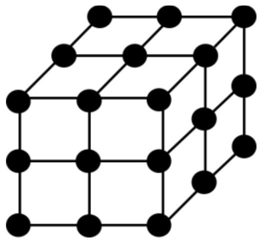


Fig. 2. A typical grid.

C. Firing Strength Calculation

Calculation of the firing strength vector $\boldsymbol{\lambda}$ is based on the EDA framework (P. Angelov et al., 2017). Fig. 3 shows the concept. A data cloud is formed with multiple data samples whereas requiring only two parameters for description, namely the focal point $\boldsymbol{\mu}_\mathbf{j}$ denoting the most representative sample and the standard deviation σ_j denoting the "radius" or range of the j^{th} cloud. It should be noted that the cloud itself can be of any shape. With the grid defined in Section III.B, the state-action space is partitioned into a set of sub-blocks with equal volumes. Each block is treated as a data cloud with its focal point being the corresponding node $\boldsymbol{\mu}_\mathbf{j} = \begin{pmatrix} \mathbf{S}_\mathbf{j}^* \\ \mathbf{A}_\mathbf{j}^* \end{pmatrix}, j = 1,2,\dots,N$. Typicality of a certain sample $\mathbf{x} = \begin{pmatrix} \mathbf{S} \\ \mathbf{A} \end{pmatrix}$ regarding the j^{th} data cloud is measured through the *unimodal discrete density* (Plamen P. Angelov et al., 2017):

$$D_j = \frac{1}{1+\frac{\left\|\mathbf{x}-\boldsymbol{\mu}_\mathbf{j}\right\|^2}{\sigma_j^2}} \tag{5}$$

It comes with the form of a Cauchy function and can also be interpreted as the membership function in conventional fuzzy systems. For a static data cloud (σ_j being fixed), the closer the sample is to the focal point, the larger the typicality. Fig. 3 shows the variation of D_j regarding $\boldsymbol{\mu}_\mathbf{j} = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}$. Note how D_j decays radially.

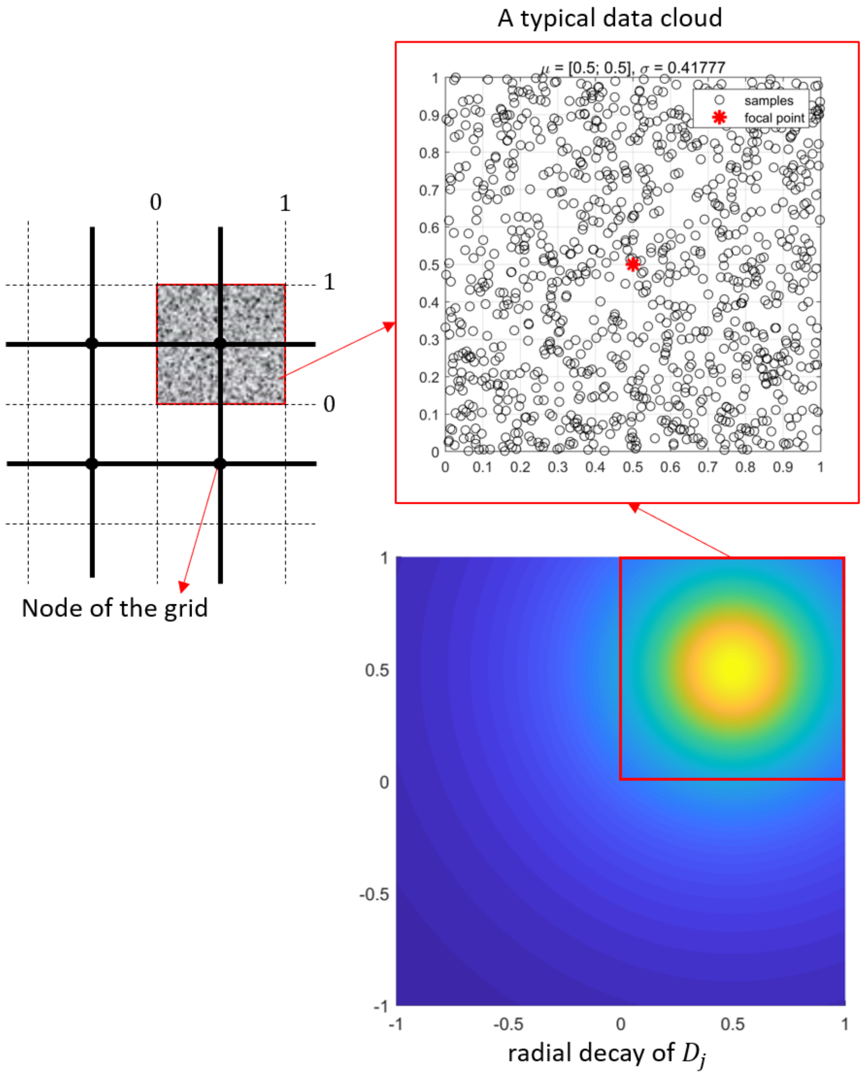


Fig. 3. Concept of data clouds.

To calculate D_j, $\boldsymbol{\mu}_\mathbf{j}$ and σ_j should be known. As has been mentioned before, $\boldsymbol{\mu}_\mathbf{j}$ is the j^{th} node itself. Thus σ_j remains to be determined. As is shown in Fig. 4, variance of samples in the data cloud can be calculated as (assuming that all members are randomly distributed within the cloud)

$$\begin{aligned}\sigma^2 &= \mathrm{E}(\|\mathbf{X}\|^2) - \|\mathrm{E}(\mathbf{X})\|^2 \\ &= \frac{\int_{x_1}^{x_2}\int_{y_1}^{y_2}(x^2+y^2)\mathrm{d}x\mathrm{d}y}{(x_2-x_1)(y_2-y_1)} - \frac{(x_1+x_2)^2}{4} - \frac{(y_1+y_2)^2}{4} \\ &= \frac{(x_2-x_1)^2+(y_2-y_1)^2}{12}\end{aligned} \tag{6}$$

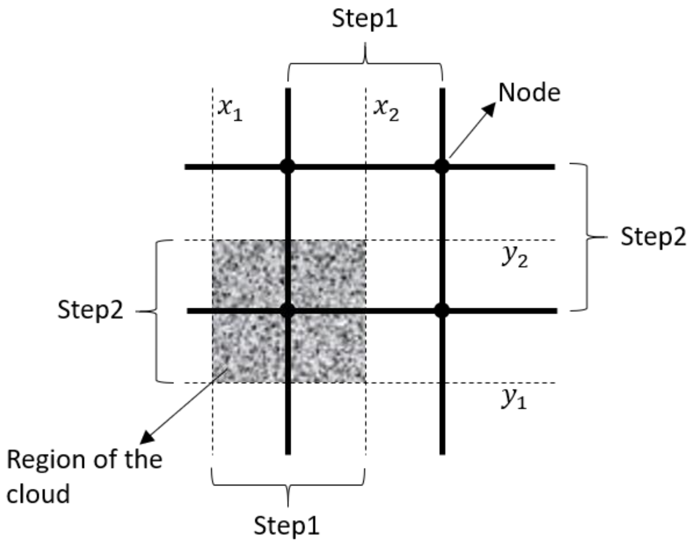


Fig. 4. Calculation of σ.

where x_1, x_2, y_1, and y_2 are coordinates of the borders.

Generally, for the state-action space with n dimensions, the standard deviation of each cloud is

$$\sigma = \sqrt{\frac{1}{12}\sum_{i=1}^{n} step_i{}^2} \tag{7}$$

where $step_i$ is the step size for discretization of the i^{th} dimension. More details on derivation of Eq. (7) is given in the supplementary material. In practice, it is usually favorable to "shrink" the clouds. This is done by adding a factor δ to the step size:

$$\sigma = \sqrt{\frac{1}{12}\sum_{i=1}^{n}(\delta \cdot step_i)^2} \tag{8}$$

For example, $\delta = \frac{1}{2}$ means that size of the cloud in each dimension is half of the original step sizes (Fig. 5).

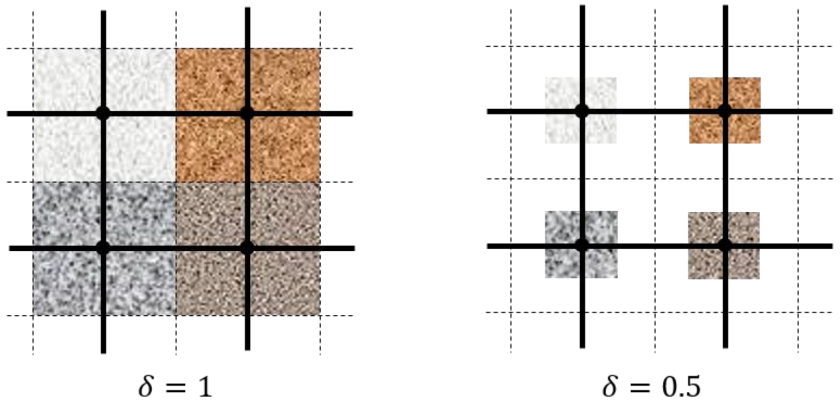


Fig. 5. Effect of δ.

Finally, firing strength of the j^{th} rule is defined as the *normalized unimodal discrete density* (Plamen P. Angelov et al., 2017):

$$\lambda_j = \frac{D_j}{\sum_{l=1}^{N} D_l} \tag{9}$$

D. Consequents update

Consequent of each cloud is updated by the Sarsa(λ) algorithm (Sutton and Barto, 2018). The feature vector and the weight vector in the context of linear function approximation in reinforcement learning corresponds to the firing strength $\boldsymbol{\lambda}$ and the consequents $\mathbf{C}$ in AnYa, respectively. The tailored version of Sarsa(λ) for AnYa is detailed in the supplementary material.

E. Computation complexity

It is obvious that the number of nodes in the grid grows exponentially with the number of dimensions, which results in the *curse of dimension*. To alleviate this problem, other methods for selecting prototypes should be applied. Some options are discussed in the supplementary material.

IV. Optimal Policy Estimator

The *optimal policy estimator*, is indeed, a probability distribution learner. It reconstructs the cumulative distribution function (CDF) from the online observed samples. Discussions of learning an unknown distribution in this section will focus on continuous cases. For discrete variables, the problem is easier since the distribution can be learned by simply recording the frequency of each enumeration.

A. Univariate Distribution Learning

Consider the circumstance where the agent observes univariate samples generated from an unknown continuous distribution. The aim now is to reconstruct the inherent distribution from the samples. There are many approaches available, the most common of which is to use conventional distribution models like Gaussian or Cauchy ones for description, and learn the parameters (mean, standard deviation, etc.) with optimization techniques. Such practices are popular because they are convenient for mathematical analysis. However, using predefined distribution functions may introduce subjective bias and degrade performances of the system. Moreover, merely one distribution function is usually not descriptive enough and a mixture of them is required, which induces the need for clustering.

To address these problems, ALMMo and EDA are used.

1) ALMMo System

ALMMo forms data clouds dynamically from streaming data in an objective way (Plamen P. Angelov et al., 2017; Angelov and Gu, 2019, 2017a). In the context of machine learning, an ALMMo agent performs online clustering for a certain variable, while avoiding the need of specifying priori configurations like the number of clusters. It is a sophisticated system with components like structure identification, online quality monitoring, parameter identification, online input selection, etc. In the context of IFRL, however, only the first two are used. Structure identification does the fundamental job of forming data clouds and updating corresponding parameters like $\boldsymbol{\mu}$ and σ. Upon arrival of a new data sample $\mathbf{x_{k+1}}$, its *unimodal discrete density* $D_{k+1}(\mathbf{x_{k+1}})$ is calculated using Eq. (5) and then compared with those of the focal points. Based on the result of comparison, either a new cloud is formed or the meta-parameters of a certain existing one are updated. Online quality monitoring is to prune the clouds that are less relevant to the recently observed samples and keep the number of existing clouds from going too large, since more clouds bring about more computations. Flowchart of the ALMMo system's learning process is given in the supplementary material. Interested readers can refer to (Plamen P. Angelov et al., 2017) for further details.

2) EDA Framework

We get a group of autonomously formed clouds and their meta-parameters like mean $\boldsymbol{\mu}_{\mathbf{j}}$, standard deviation σ_j, support S_j (the number of members belonging to a certain cloud), etc. with ALMMo. We then process them with EDA to extract the underlying probabilistic information. Specifically, the quantity *continuous multimodal typicality* is used (Plamen P Angelov et al., 2017). It resembles the probability density function (PDF) whereas differs from it. PDF is predefined, subjective, and considers only spatial relationship of data samples. For example, if observed samples concentrate around a certain focal point $\boldsymbol{\mu}$, then the probability density of a sample far away from $\boldsymbol{\mu}$ will be small and its occurrence will be considered less probable. Comparatively, *continuous multimodal typicality* considers not only the spatial relationship but also the frequency of the samples. It approximates frequentist probability when the number of observed samples is small and automatically transforms into PDF when a lot of samples are observed (P. Angelov et al., 2017). Since the data clouds are formed online automatically by the ALMMo, *continuous multimodal typicality* is totally objective and not based on any priori assumptions of the pattern of data. Denote this quantity with τ and for data sample $\mathbf{x}$ it is (assuming that Euclidean type distance is used)

$$\tau(\mathbf{x}) = \frac{\Gamma\left(\frac{n+1}{2}\right)}{\pi^{\frac{n+1}{2}} S} \sum_{j=1}^{N} \frac{S_j}{\sigma_j^n \left(1 + \frac{\left\|\mathbf{x} - \boldsymbol{\mu}_{\mathbf{j}}\right\|^2}{\sigma_j^2}\right)^{\frac{n+1}{2}}} \tag{10}$$

where S is the number of all the observed data samples, S_j is the number of members belonging to the j^{th} cloud, n is the number of dimensions of $\mathbf{x}$, and N is the number of clouds formed dynamically online. For the univariate case ($n = 1$), Eq. (10) turns into

$$\tau(x) = \frac{1}{\pi S} \sum_{j=1}^{N} \frac{S_j}{\sigma_j \left(1 + \frac{\left(x - \mu_j\right)^2}{\sigma_j^2}\right)} \tag{11}$$

This function can be used as a form of PDF. The corresponding CDF can be derived by integrating Eq. (11):

$$P(x \leq t) = \int_{x=-\infty}^{t} \tau(x)\mathrm{d}x = \frac{\sum_{j=1}^{N} S_j\left(\frac{1}{\pi}\arctan\left(\frac{t-\mu_j}{\sigma_j}\right)+\frac{1}{2}\right)}{S} \tag{12}$$

3) Inverse Transform Sampling

The ultimate goal of learning the optimal policy is to reproduce actions from it. This is done through *Inverse Transform Sampling* (ITS) (Devroye, 1990). Specifically, for the variable X, samples are reproduced by

$$X = F_X^{-1}(U) \tag{13}$$

where $F_X(t) = P(x \leq t)$ is the CDF and U is a uniformly distributed random number in the interval $[0, 1]$. For the CDF in Eq. (12), however, the inverse function is difficult to calculate analytically. Therefore, a 1-D lookup table is used instead. Eq. (12) is evaluated on evenly spaced points and the values are stored in a table. The uniform random number generator is called to produce U and interpolation is carried out to derive the interpolated value of $F_X^{-1}(U)$ at the query point.

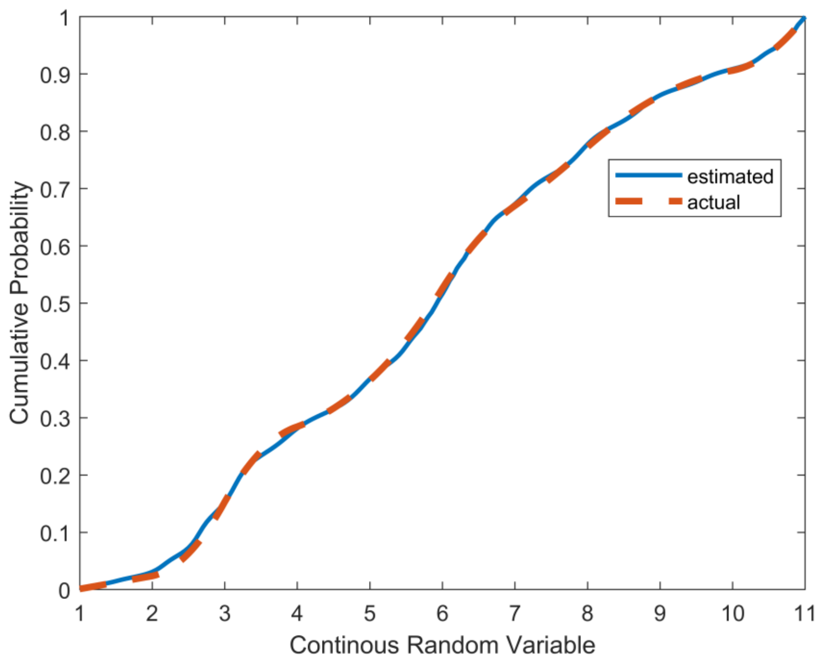
Note that the domain of definition of $F_X(t)$ is $[-\infty, +\infty]$, which may differ from real cases. To fix this problem, the *truncated* version (Kochenderfer et al., 2015) of Eq. (12) should be used:

$$P_{tr}(x \leq t) = \frac{P(x\leq t)-P(x\leq l)}{P(x\leq r)-P(x\leq l)}, l \leq t \leq r \tag{14}$$

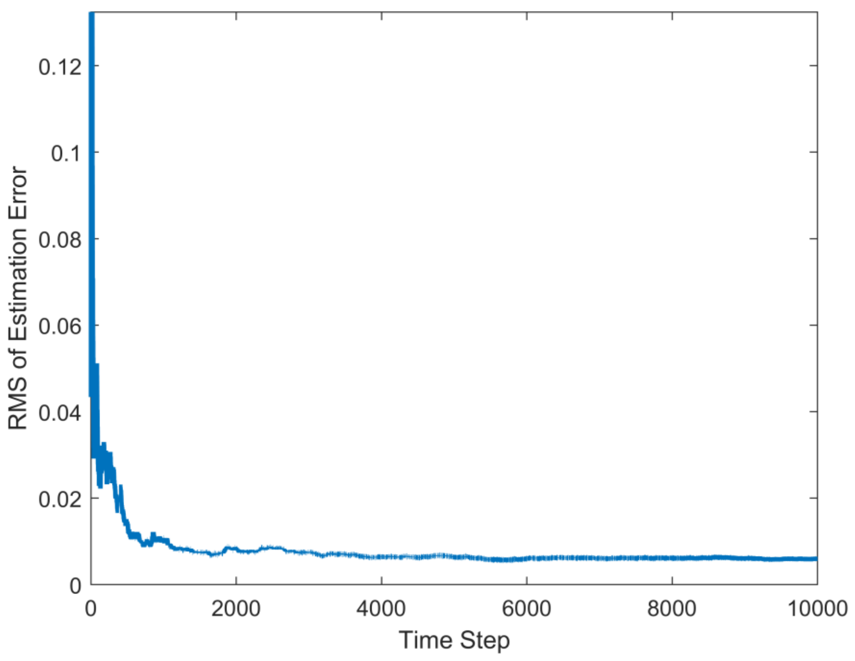
where l and r are the left and right boundary of the interval, respectively.

4) Verification

The univariate distribution learning algorithm is tested through a simple simulation. Firstly, an artificial distribution is defined by a table with two rows specifying the query points and the corresponding cumulative probability. A batch of samples are then generated from it using ITS and interpolation. At each time step, one sample is passed to the proposed algorithm. Totally 10000 samples are used. Comparison between the estimated CDF and the actual one as well as the root-mean-square (RMS) of the estimation error during training is shown in Fig. 6. It can be seen that the estimated CDF corresponds with the actual one quite well and the RMS of estimation error falls to a low level with about 1000 samples.



(a) Estimated vs. actual CDF.



(b) RMS of estimation error.

Fig. 6. Results of learning a univariate distribution.

B. *Fuzzy Generalization Between Different States*

1) *εFS Framework*

The last subsection discusses learning of a continuous univariate distribution. Now consider the extended circumstance where the learned distribution is valid only in a certain state. Moreover, the state variable is also continuous, but not necessarily univariate. The question is how to generalize the estimated distribution between different states.

We propose to use for this purpose the ALMMo and the **ε**FS (Angelov and Gu, 2017b; Rong et al., 2018). They are much like the density and typicality discussed in Section III with differences in that a) data clouds are formed dynamically online, whereas in Section III they are predefined and static; b) variable in the antecedent part is the state variable rather than the state-action pair; c) consequents are the distributions regarding each state, whereas in Section III they are the estimated values of each state-action pair.

Suppose that the action variable of the reinforcement learning problem is one-dimensional. In this case, totally $(N + 1)$ ALMMo agents need to be used, as is illustrated in Fig. 7. Here, N denotes the number of data clouds/prototypes for states. As a result, N agents are needed for estimating the distributions of optimal actions. Apart from them, another ALMMo agent is needed for the state variable. It outputs the firing strength for each of the N rules. For better understanding, recall the process of decision making by human beings. We categorize numerous situations into several typical ones and take corresponding actions for each of them. For example, we get more dressing when we travel north, and less when travelling south (assuming that we live in the Northern Hemisphere). Here, "north" and "south" are typical prototypes of the state variable "latitude", and "more" or "less" are the actions for each prototype. We do not keep a table in our mind with the first row as 0°, 1°, …, etc. and the second row as different levels of dressing, since this will take up too much memory resources. Instead, we just use two simple rules. This is possible because human beings are able to identify typical situations and generalize actions from them. ALMMo and **ε**FS implement these two functionalities, respectively. Prototypes are identified online through autonomously formed data clouds. Generalization is done through the fuzzy ensemble of N sets of policies corresponding to N prototypes.

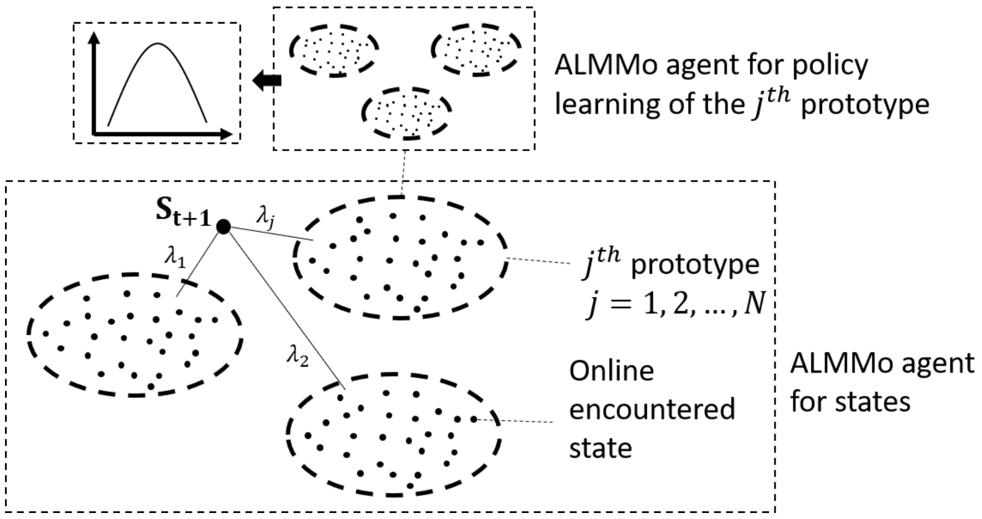

ALMMo agent for policy learning of the j^{th} prototype
$\mathbf{S_{t+1}}$
λ_j
λ_1
λ_2
j^{th} prototype $j = 1, 2, \ldots, N$
Online encountered state
ALMMo agent for states

Fig. 7. Fuzzy generalization of policies between different states.

Upon encounter of a new state $\mathbf{S_{t+1}}$, which may be of multiple dimensions, the vector of firing strength $\boldsymbol{\lambda}(\mathbf{S_{t+1}})$ is calculated as in Section III. Let us denote the policy vector as

$$\mathbf{F}_X = [F_X^1 \quad F_X^2 \quad \ldots \quad F_X^N]^{\mathrm{T}} \tag{15}$$

where $F_X^j(t) = P^j(x \leq t)$ is the estimated CDF of optimal actions for the j^{th} prototype of state. The policy for $\mathbf{S_{t+1}}$ is determined as

$$F_X^{\mathbf{S_{t+1}}} = \boldsymbol{\lambda}^{\mathrm{T}}\mathbf{F}_X \tag{16}$$

The superscript means that the CDF is conditioned on $\mathbf{S_{t+1}}$.

2) Learning of Individual Policy

Whenever an advised action is proposed by the *value function approximator*, the policy corresponding to the most relevant situation is updated. Relevance of the current state regarding each prototype is measured through the firing strength vector $\boldsymbol{\lambda}$. Specifically, it is identified as

$$ind = \operatorname*{argmax}_j \lambda_j \tag{17}$$

The corresponding ALMMo agent for actions is then updated as detailed in (Plamen P. Angelov et al., 2017).

C. Multivariate Actions

Discussions above are restricted to the case of univariate action variable. However, there are problems with multivariate action space. To solve them within the same framework introduced before, a hierarchical method is used.

The idea is to learn distributions of each component in the action vector incrementally. Imagine a clock with two hands. In each round, the player manipulates the two hands and receives a reward as either 0 or 1. The action variable is $\mathbf{A} = [A_1 \quad A_2]$ denoting positions of the two hands, each of which varies continuously. The aim is to find out the distribution of $\mathbf{A}$ with which the player maximizes his rewards. Since the optimal distribution is unknown, the player tries different combinations of $[A_1 \quad A_2]$ randomly, and for each attempt records the action and the corresponding reward. Actions with positive rewards are then picked out. For the first component A_1, a histogram with a certain discretization step can be used to describe the distribution. However, for the second one, since it is dependent on the first one, doing so would be meaningless. In other words, distribution of A_2 is conditioned on A_1. This situation is similar to the one in Section IV.B, where the the univariate action is conditioned on the state. Therefore, the method of fuzzy generalization can be used here. If the clock comes with M hands, firstly the distribution of A_1 is learned, then the conditioned distribution of A_2 on A_1, then A_3 on $[A_1 \quad A_2]$, and so on, up to A_M. Each time we learn the univariate distribution of a certain component in the action vector as in Section IV.A, and condition it on the sub-vector composed of all preceding ones as in Section IV.B.

Now consider a more complex situation where the clock presented to the player in each round is different and attached with a state variable $\mathbf{S}$. This is controlled by the environment and not the player. However, it can still be treated as part of the condition. Therefore, the method discussed before can be applied by simply appending $\mathbf{S}$ to each condition: A_1 on $\mathbf{S}$, A_2 on $[\mathbf{S} \quad A_1]$, A_3 on $[\mathbf{S} \quad A_1 \quad A_2]$, and so on, up to A_M. In this way, the problem of multivariate policy learning is handled. It is hierarchical because the learning of each component is based on the preceding ones.

Decision process of the *inferred action* for $\mathbf{S_{t+1}}$ is as follows. Firstly generate A_1 through ITS of the fuzzily weighted CDF $F_{A_1}^{\mathbf{S_{t+1}}} = \boldsymbol{\lambda}^{\mathrm{T}}(\mathbf{S_{t+1}})\mathbf{F}_{A_1}$ as

$$A_1 = \mathrm{ITS}\left(F_{A_1}^{\mathbf{S_{t+1}}}\right) \tag{18}$$

Then generate A_2 as

$$A_2 = \mathrm{ITS}(\,F_{A_2}^{[\mathbf{S_{t+1}} \quad A_1]}) \tag{19}$$

where $F_{A_2}^{[\mathbf{S_{t+1}} \quad A_1]} = \boldsymbol{\lambda}^{\mathrm{T}}([\mathbf{S_{t+1}} \quad A_1])\mathbf{F}_{A_2}$. And repeat the procedure for A_3, A_4, etc. until all the components are determined.

D. *Computation complexity*

Computation complexity of the *optimal policy estimator* is discussed in this subsection. Suppose that the action space has M dimensions. In this case, the conditions are $\mathbf{S}$, $[\mathbf{S} \quad A_1]$, $[\mathbf{S} \quad A_1 \quad A_2]$, …, $[\mathbf{S} \quad A_1 \quad A_2 \quad \ldots \quad A_{M-1}]$, and the number of them is $1 + (M-1) = M$. For each condition, one ALMMo agent is needed to calculate the firing strength vector $\boldsymbol{\lambda}$. The number of data clouds within each agent is denoted as $N_i, i = 1,2, \ldots, M$. As has been discussed before, each cloud corresponds to one prototype of the condition, and for each prototype there is one policy, which is a univariate distribution learned through an ALMMo agent. Thus, the number of all agents needed is

$$M_{all} = M + \sum_{i=1}^{M} N_i = M + \sum_{i=1}^{M} \bar{N} = M \cdot (1 + \bar{N}) \tag{20}$$

where $\bar{N}$ is the average of N_i. N_i is controlled by online quality monitoring (Plamen P. Angelov et al., 2017) and its variation is small for the same configurations. Therefore, $\bar{N}$ can be regarded as a constant for action space with different dimensions. Consequently, it can be concluded from Eq. (20) that the computation complexity grows linearly, rather than exponentially, with increase of the number of dimensions in the action space. This is one of the advantages of the proposed *optimal policy estimator*.

V. SIMULATION RESULTS

Four problems, namely Mountain Car, Continuous Gridworld, Pendulum Positioning, and Tank Level Control, are used for validation.

A. *Mountain Car*

The Mountain Car problem (Sutton and Barto, 2018) is classic and mainly used for evaluation of function approximation methods. The goal is to drive the car up to the goal position, which is on the end of the upslope. Difficulty of the problem is that the power of the car is not strong enough to propel it along the slope directly. Rather, the car has to be driven in the inverse direction first to build up enough potential energy. Definition of the problem is shown in Table I. An illustration is given in Fig. 8.

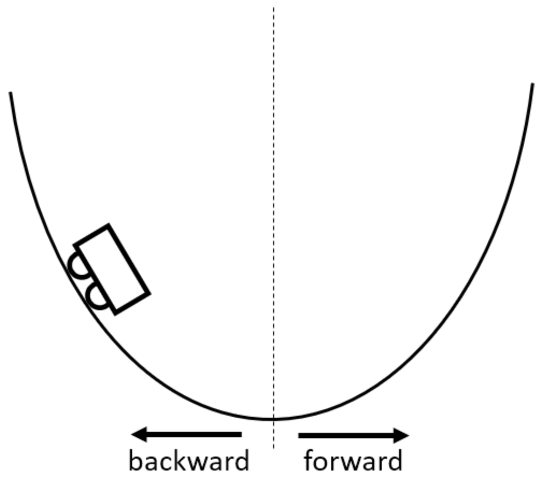


Fig. 8. The mountain car problem.

TABLE I

MOUNTAIN CAR PROBLEM

Item	Definition
State transition	$x_{t+1} = \mathrm{bound}[x_t + \dot{x}_{t+1}]$ $\dot{x}_{t+1} = \mathrm{bound}[\dot{x}_t + 0.001A_t - 0.0025\cos(3x_t)]$ $\dot{x}_t$ is reset to zero when x_t reaches the left bound
State variable	$[x_t \quad \dot{x}_t]$ position and velocity of the car
State space	$-1.2 \le x_t \le 0.5$ $-0.07 \le \dot{x}_t \le 0.07$
Initial state	$x_0 \in (-0.6 \quad -0.4]$

	$\dot{x}_0 = 0$
Terminal state	$x_T = 0.5$
Maximal steps per episode	unrestricted
Discount rate	1
Action variable	A_t can be taken as discrete: $A_t \in \{+1, -1, 0\}$ or continuous: $A_t \in [-1, 1]$ full throttle forward (+1) full throttle backward (-1) zero throttle (0)
Action space	discrete action variable
Reward	-1 on each step until the terminal state is reached

Firstly, AnYa as the value function approximator is compared to *state aggregation*, which produces one estimation for one group. Value approximation in *state aggregation* is based on merely one component of the weight vector **w** (i.e. the consequent vector **C**), whereas in AnYa all the components (or part of them, depending on the implementation) in **C** are utilized. For the Mountain Car problem, performances of the two methods are compared through *Steps per Episode* (SPE), assuming discrete action space. Results are shown in Fig. 9. It is obvious that AnYa comes with great advantages over *state aggregation* regarding the learning rate. Near optimality is obtained after the first several episodes. Comparatively, it takes over 2000 episodes for *state aggregation* to achieve the same result. Difference in the learning rate by two methods partly results from the different numbers of components in the weight vector. In AnYa, only 90 weights need to be learned, whereas in *state aggregation* it is 14280. Using the same number of weights in *state aggregation* as that of AnYa will result in divergence.

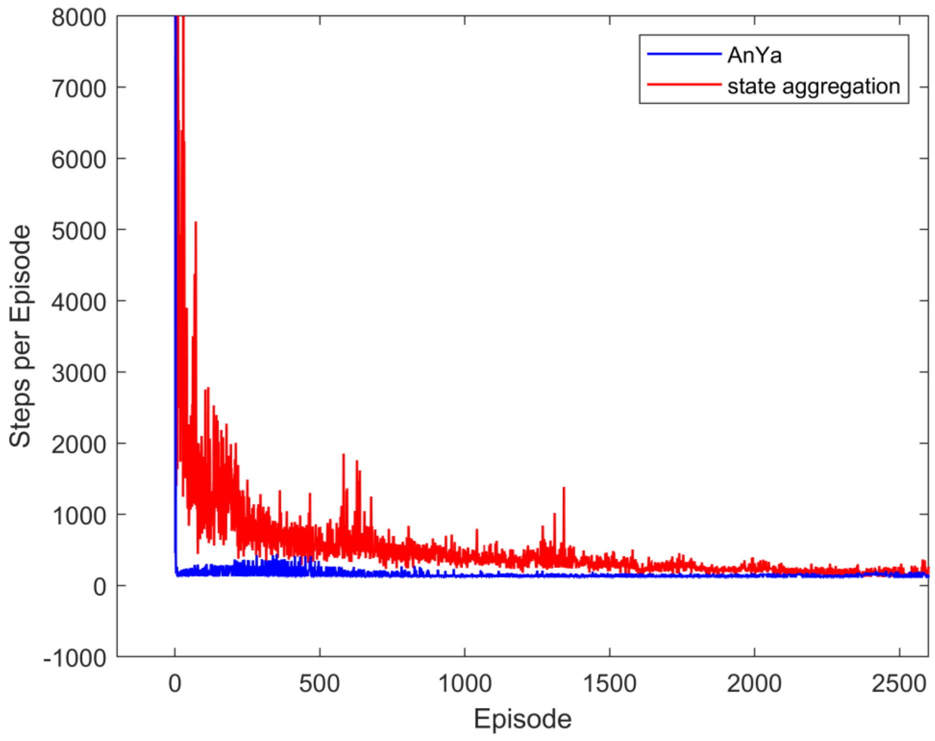
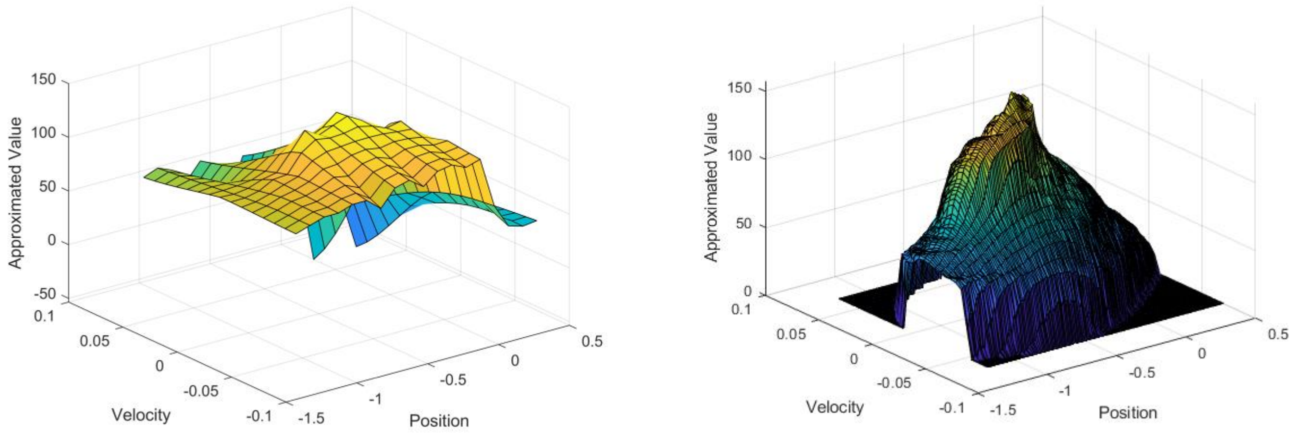


Fig. 9. Steps per episode in Mountain Car problem (AnYa vs. state aggregation).



(a) AnYa

(b) state aggregation

Fig. 10. Cost-to-go function learned by the two methods.

The cost-to-go function learned at the end is shown in Fig. 10. It is obvious that *state aggregation* with more weights produces a better approximation. However, although AnYa approximates the true values coarsely (with only 90 weights), the corresponding policy is near-optimal. This implies that for learning an optimal policy, the exact value of each state-action pair is not important. Rather, the relativity of magnitude is what really matters. By considering values of all the nodes or a group of neighboring ones within the grid, and weighting them fuzzily, AnYa is able to extract this relativity with a small number of weights, which makes extremely fast learning possible.

The Mountain Car problem is then solved again by treating the action variable as continuous. Results are shown in Fig. 11. IFRL automatically forms 9 rules online. Note how the data clouds are positioned along the optimal phase trajectory. To reach the terminal state, the car has to dangle back and forth to accumulate energy, which is obvious from the plot.

The first four of algorithmically learned rules are listed in Table II. These rules allow human users to gain insights from them. Firstly, although the action space is continuous, actual behaviors are concentrated on the two ends, which indicates that the car is going either forward or backward with full throttle most of the time. Thus, using continuous actions makes little improvement in optimality and is unnecessary. Secondly, intuition on the optimal policy can be developed by observing PDFs of several prototypes. For example, PDFs of both the 1st and the 2nd prototype come with peaks on the two ends of the action space. This can be translated into human-intelligible rules as “IF the car is in the middle of the valley and with low velocity, THEN it should go either forward or backward with full throttle to build up the potential energy”. If the car is on the downhill with negative velocity, the firing strength of the 3rd rule will be dominant and accordingly the car should keep going backward with full throttle, which corresponds to the phase trajectory. On the other hand, if the car is on the downhill with large velocity (the 4th prototype), then it should go forward with full throttle to reach the goal directly.

Interpretability through the fuzzy kernel is one of the main advantages of the newly proposed IFRL. This is possible because:

a) IFRL is able to cluster encountered states online;

b) it is integrated with an *optimal policy estimator*, which learns the probability distribution of optimal actions.

Note also that the automatically formed data clouds only cover states on the optimal trajectory, which makes the method memory-efficient.

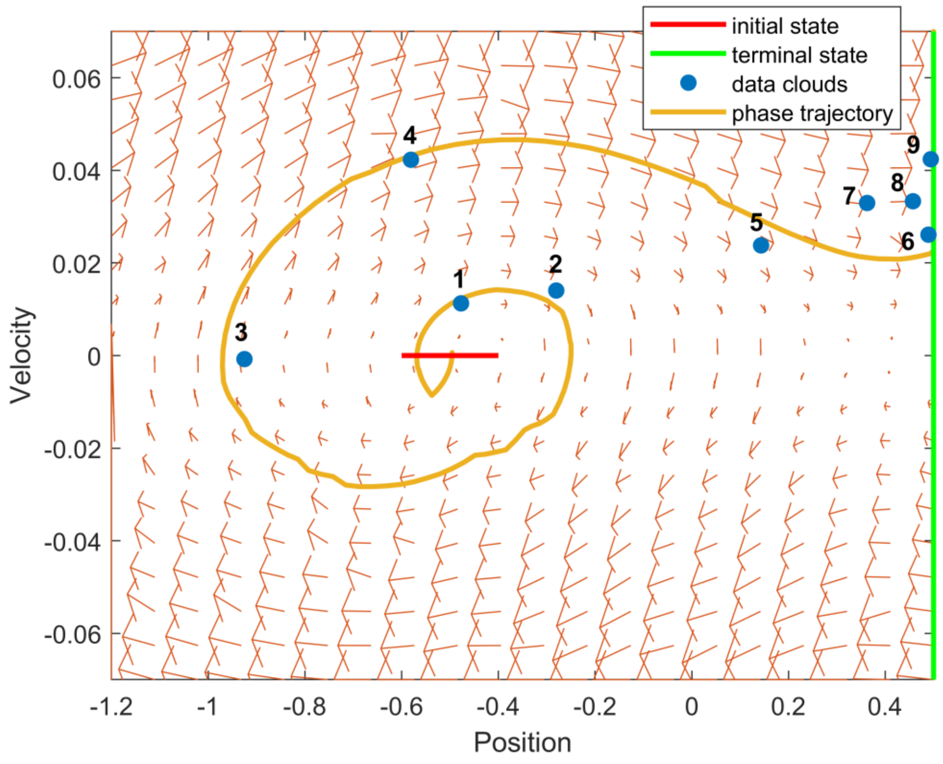
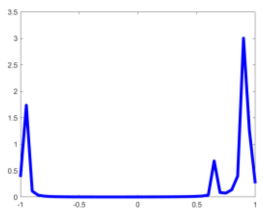
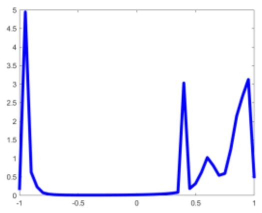
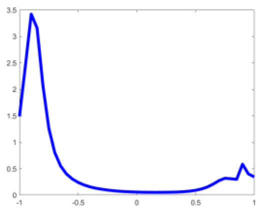
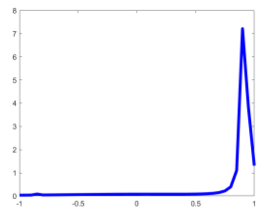


Fig. 11. Results for the Mountain Car Problem with continuous action space.

TABLE II

ONLINE LEARNED RULES

ID of data cloud in Fig. 11	IF state is close to	Then PDF of action is
1	$[-0.4772 \quad 0.0113]$	

2	$[-0.2803 \quad 0.0140]$	
3	$[-0.9251 \quad -0.0007]$	
4	$[-0.5808 \quad 0.0423]$	

B. Continuous Gridworld

A typical gridworld is shown in Fig. 12. The agent starts at the initial position $[5 \quad 5]$ and tries to move to the target position $[10 \quad 10]$. The classic version treats both the state and action as discrete variables. The agent chooses one of the four actions in each step: left, right, up, and down. In the adapted version used here, however, the state and action variables are both continuous, as are shown in Table III. What's more, the action variable is now of two dimensions. This means that the agent can move diagonally. Introduction of the 2-D action space is to validate the method of hierarchical learning in Section IV.C.

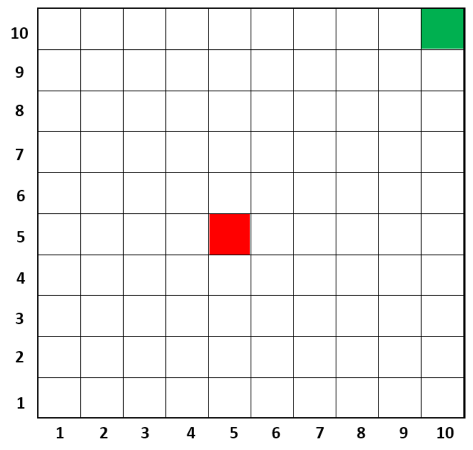


Fig. 12. A typical gridworld.

Results of SPE are shown in Fig. 13. The theoretical minimal SPE is 6. The agent achieves near-optimality after 1500 episodes. From Fig. 13 it can be concluded that the method of hierarchical learning is effective for problems with multivariate action space.

TABLE III

CONTINUOUS GRIDWORLD

Item	Definition
State transition	$h_{t+1} = h_t + \dot{h}_t$ $v_{t+1} = v_t + \dot{v}_t$
State variable	$[h_t \quad v_t]$ horizontal and vertical position
State space	$0 \le h_t \le 10$ $0 \le v_t \le 10$
Initial state	$h_0 = 5$ $v_0 = 5$
Terminal state	$h_T = 10$ $v_T = 10$
Maximal steps per episode	unrestricted

Discount rate	1
Action variable	$[\dot{h}_t \quad \dot{v}_t]$ $-1 < \dot{h}_t < 1$ $-1 < \dot{v}_t < 1$
Action space	continuous action variable
Reward	-1 on each step until the terminal state is reached

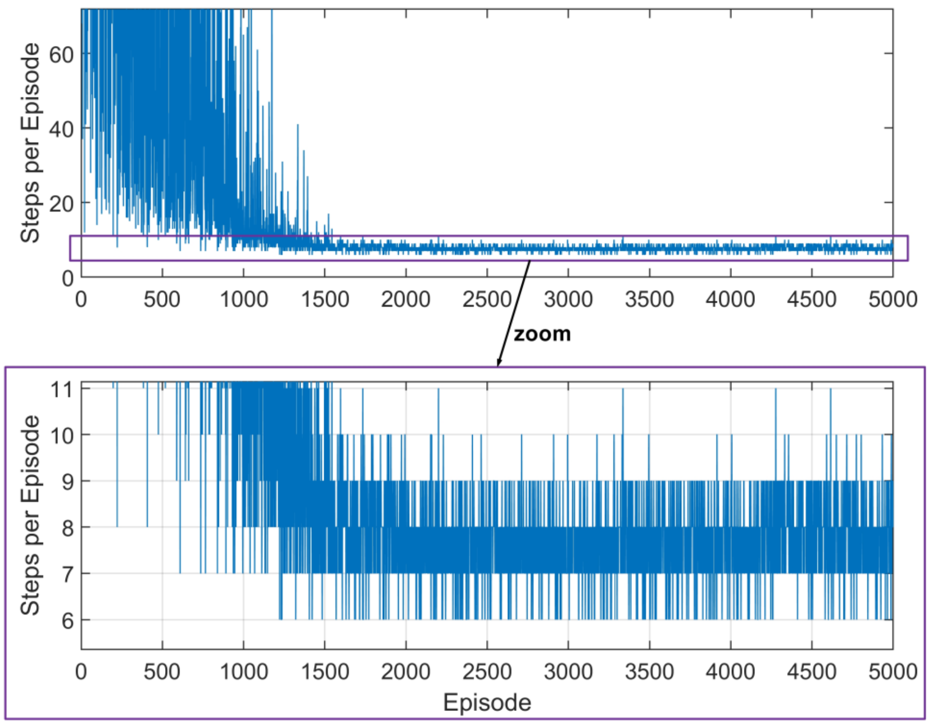


Fig. 13. Steps per episode for the Continuous Gridworld problem with 2-D action space.

C. Pendulum Positioning

The Pendulum Positioning problem is described in (Sheen, 2016). The pendulum is initially positioned straight down. A torque is exerted on the pendulum to drive it to the upright position. If the torque is too small, the pendulum fails to go up. On the other hand, if it is too large, the pendulum simply crosses the target position and goes to the downside again. The aim is to drive the pendulum uprightly with as few steps as possible. An extra bonus is provided if the pendulum is successfully positioned to the target, as is shown in

Fig. 14 and Table IV.

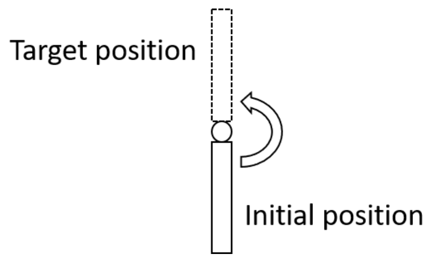


Fig. 14. The pendulum positioning problem.

TABLE IV

PENDULUM POSITIONING PROBLEM

Item	Definition
State transition	refer to (Sheen, 2016) for details
State variable	$[x_t \quad \dot{x}_t]$ angle and angular speed of the pendulum
State space	$-\pi \leq x_t \leq \pi$ $-\pi \leq \dot{x}_t \leq \pi$
Initial state	$x_0 = \pi$ $\dot{x}_0 = 0$
Terminal state	$x_T = 0$ $\dot{x}_T = 0$
Maximal steps per episode	1500
Discount rate	0.9
Action variable	A_t A_t can be taken as discrete: $A_t \in \{-1, 0, 1\}$ or continuous: $A_t \in [-1, 1]$
Action space	discrete/continuous action variable

Reward	IF $\sqrt{x_{t+1}^2 + \dot{x}_{t+1}^2} < 0.01$
	 $R(t+1) = -x_{t+1}^2 - 0.25\dot{x}_{t+1}^2 + 100$
	 ELSE
	 $R(t+1) = -x_{t+1}^2 - 0.25\dot{x}_{t+1}^2$

Phase trajectory of the pendulum is shown in Fig. 15. The action variable is considered to be continuous. The pendulum starts at the initial state $[\pi, 0]$ and travels to the target $[0, 0]$ successfully. Note that $[\pi, 0]$ and $[-\pi, 0]$ are actually the same state.

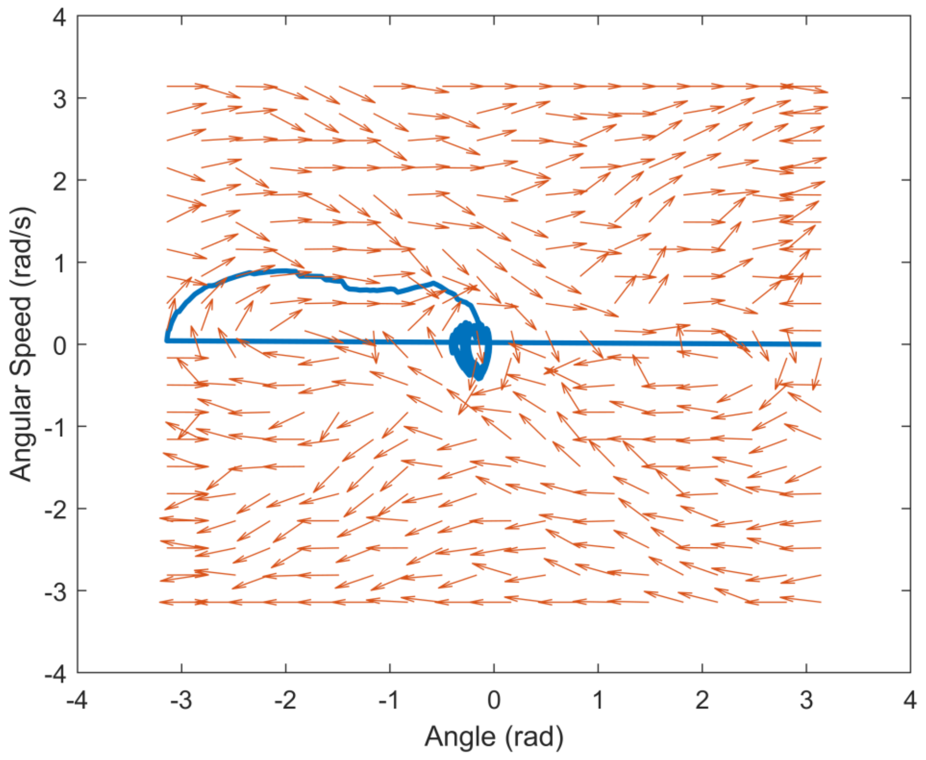


Fig. 15. Phase trajectory of the pendulum.

D. *Tank Level Control*

The Tank Level Control problem is described in (Noel and Pandian, 2014). There are two tanks with different liquid levels. The goal is to maintain the first one at a desired setpoint. This benchmark is to evaluate the potential of applying IFRL to control problems of nonlinear systems with continuous states and inputs, which is common in practical engineering circumstances. Definition of the problem is shown in Table V. Fig.

16 shows the optimal trajectories of tank levels obtained by dynamic programming (DP) and IFRL, respectively. Results from DP are calculated offline and can be treated as theoretically best. It can be observed that the trajectory from IFRL successfully achieves the control target, though with some chattering. This is due to the inherent probabilistic characteristics of IFRL.

TABLE V

TANK LEVEL CONTROL PROBLEM

Item	Definition
State transition	$h_1 \geq h_2 \begin{cases} \dot{h}_1 = \dfrac{q_1 - r_1\sqrt{h_1} - r_3\sqrt{h_1 - h_2}}{A_1} \\ \dot{h}_2 = \dfrac{q_2 - r_2\sqrt{h_2} - r_3\sqrt{h_1 - h_2}}{A_2} \end{cases}$ $h_1 \leq h_2 \begin{cases} \dot{h}_1 = \dfrac{q_1 - r_1\sqrt{h_1} - r_3\sqrt{h_2 - h_1}}{A_1} \\ \dot{h}_2 = \dfrac{q_2 - r_2\sqrt{h_2} - r_3\sqrt{h_2 - h_1}}{A_2} \end{cases}$ $h_1(t+1) = h_1(t) + 0.1\dot{h}_1(t)$ $h_2(t+1) = h_2(t) + 0.1\dot{h}_2(t)$
State variable	$[h_1(t) \quad h_2(t)]$ liquid levels of the two tanks
State space	$0 \leq h_1(t) \leq 10$ $0 \leq h_2(t) \leq 10$
Initial state	$h_1(0) = 1$ $h_2(0) = 0$
Terminal state	$h_1(T) = 7$
Maximal steps per episode	1500
Discount rate	0.99

Action variable	$A_t \in [0, 20]$
Action space	continuous action variable
Reward	$R(t+1) = -\lvert h_1(t+1) - h_1(T)\rvert$

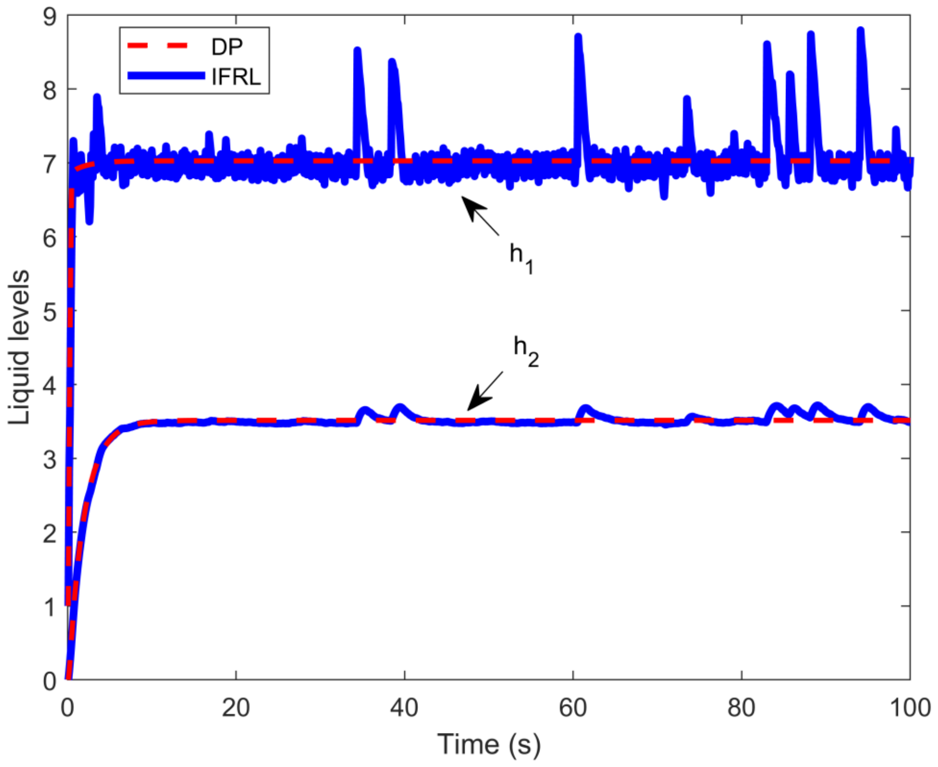


Fig. 16. Optimal trajectories of tank levels (DP vs. IFRL).

VI. Conclusion

In this paper, a new method and an algorithm are proposed to implement interpretable fuzzy reinforcement learning (IFRL). The method is able to produce human-intelligible rules online, which facilitates further investigation and improvement of the policy. It is applicable to problems with continuous and multivariate action space, which is a great advantage over the classic tabular approaches. Different from mainstream policy-gradient methods, the learning process of IFRL is much like that of a human. Various actions are tried and the outcomes are evaluated. Favorable ones are memorized to form a policy. The recent fuzzy system AnYa is used to approximate values of state-action pairs and acts as an evaluator. The classic

Sarsa(λ) algorithm is used to update consequents of the fuzzy rules. ALMMo with EDA is used for learning the univariate probability distribution. Inferred actions from the learned CDF are produced through ITS. Generalization between different states is implemented by **ε**FS. Hierarchical learning is adopted to deal with multivariate action space. Solution of the Mountain Car problem shows that the newly proposed method requires orders of magnitude less parameters and provides low-error solution with orders of magnitude faster convergence, in addition to its transparency from the human-intelligible fuzzy rules. Effectiveness of hierarchical learning is validated by solutions of the Continuous Gridworld problem. Potential of applying IFRL on control problems of nonlinear systems in engineering is evaluated through the Tank Level Control problem.

Future research directions are:

a) Improvement of the *value function approximator*. In the current version, a static grid is used. This can be replaced with a dynamic grid to achieve balance between performances and the number of nodes.

b) Improvement of the *optimal policy estimator*. The current version clusters states/conditions encountered with ALMMo. However, a more proper setting is to cluster the variable consisting both the condition and the action. This is expected to reduced the numbers of rules learned.

c) Addition of a component for learning of the environment model.

d) Addition of the ability to perform planning using algorithms like tree search.

e) Improvement on the way of generating recommended actions from the *optimal policy estimator,* so as to reduce chattering in the output. This can be achieved by, for example, using the mean of the distribution (rather than the sampled one from the CDF) as the inferred action.

f) Treatment of hidden states.

References

Angelov, P., 2012. Autonomous Learning Systems. John Wiley & Sons, Ltd, Chichester, UK. https://doi.org/10.1002/9781118481769

Angelov, P., Gu, X., Kangin, D., 2017. Empirical Data Analytics. Int. J. Intell. Syst. 32, 1261–1284. https://doi.org/10.1002/int.21899

Angelov, P., Gu, X., Kangin, D., Principe, J., 2016. Empirical data analysis: A new tool for data analytics, in: 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC). IEEE, pp. 000052–000059. https://doi.org/10.1109/SMC.2016.7844219

Angelov, P., Yager, R., 2012. A new type of simplified fuzzy rule-based system. Int. J. Gen. Syst. 41, 163–185. https://doi.org/10.1080/03081079.2011.634807

Angelov, P., Yager, R., 2011. Simplified fuzzy rule-based systems using non-parametric antecedents and relative data density, in: 2011 IEEE Workshop on Evolving and Adaptive Intelligent Systems (EAIS). IEEE, pp. 62–69. https://doi.org/10.1109/EAIS.2011.5945926

Angelov, P.P., Gu, X., 2019. Empirical approach to machine learning. Springer.

Angelov, P.P., Gu, X., 2017a. Autonomous learning multi-model classifier of 0-order (ALMMo-0). IEEE Conf. Evol. Adapt. Intell. Syst. 1–7. https://doi.org/10.1109/EAIS.2017.7954832

Angelov, P.P., Gu, X., 2017b. Empirical Fuzzy Sets. Int. J. Intell. Syst. 00, 1–34. https://doi.org/10.1002/int.21935

Angelov, Plamen P, Gu, X., Principe, J., 2017. A generalized methodology for data analysis. IEEE Trans. Cybern. 1, DOI: 10.1109/TCYB.2017.2753880. https://doi.org/10.1109/TCYB.2017.2753880

Angelov, Plamen P., Gu, X., Principe, J.C., 2017. Autonomous learning multi-model systems from data streams. IEEE Trans. Fuzzy Syst. 6706, 1–12. https://doi.org/10.1109/TFUZZ.2017.2769039

Antonoglou, I., Fidjeland, A.K., Wierstra, D., King, H., Bellemare, M.G., Legg, S., Petersen, S., Riedmiller, M., Beattie, C., Graves, A., Sadik, A., Kavukcuoglu, K., Ostrovski, G., Veness, J., Rusu, A.A., Silver, D., Hassabis, D., Kumaran, D., Mnih, V., 2015. Human-level control through deep reinforcement learning. Nature 518, 529–533. https://doi.org/10.1038/nature14236

Buckley, J.J., 1993. Sugeno type controllers are universal controllers. Fuzzy sets Syst. 53, 299–303. https://doi.org/10.1016/0165-0114(93)90401-3

Chia-Feng Juang, Chia-Hung Hsu, 2009. Reinforcement Ant Optimized Fuzzy Controller for Mobile-Robot Wall-Following Control. IEEE Trans. Ind. Electron. 56, 3931–3940. https://doi.org/10.1109/TIE.2009.2017557

Devroye, L., 1990. Non-Uniform Random Variate Generation, Proceedings of COMPSTAT 2010 - 19th International Conference on Computational Statistics, Keynote, Invited and Contributed Papers. https://doi.org/10.1007/978-3-7908-2604-3-1

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep learning. MIT press.

Hein, D., Hentschel, A., Runkler, T., Udluft, S., 2017. Particle swarm optimization for generating interpretable fuzzy reinforcement learning policies. Eng. Appl. Artif. Intell. 65, 87–98. https://doi.org/10.1016/j.engappai.2017.07.005

Hein, D., Udluft, S., Runkler, T.A., 2018. Interpretable policies for reinforcement learning by genetic programming. Eng. Appl. Artif. Intell. 76, 158–169. https://doi.org/10.1016/j.engappai.2018.09.007

Jin, Y., 2000. Fuzzy modeling of high-dimensional systems: Complexity reduction and interpretability improvement. IEEE Trans. Fuzzy Syst. 8, 212–221. https://doi.org/10.1109/91.842154

Kochenderfer, M.J., Amato, C., Chowdhary, G., How, J.P., Reynolds, H.J.D., Thornton, J.R., Torres-Carrasquillo, P.A., Üre, N.K., Vian, J., 2015. Decision Making Under Uncertainty: Theory and Application.

Kosko, B., 1994. Fuzzy systems as universal approximators. IEEE Trans. Comput. 43, 1329–1333.

Lecun, Y., Bengio, Y., Hinton, G., 2015. Deep learning. Nature 521, 436–444. https://doi.org/10.1038/nature14539

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D., 2015. Continuous control with deep reinforcement learning. https://doi.org/10.1561/2200000006

Lipton, Z.C., 2018. The mythos of model interpretability. Commun. ACM 61, 36–43. https://doi.org/10.1145/3233231

Maes, F., Fonteneau, R., Wehenkel, L., Ernst, D., 2012. Policy search in a space of simple closed-form formulas: Towards interpretability of reinforcement learning. Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics) 7569 LNAI, 37–51. https://doi.org/10.1007/978-3-642-33492-4_6

Mamdani, E.H., Assilian, S., 1975. An experiment in linguistic synthesis with a fuzzy logic controller. Int. J. Man. Mach. Stud. 7, 1–13. https://doi.org/10.1016/S0020-7373(75)80002-2

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M., 2013. Playing Atari with Deep Reinforcement Learning 1–9. https://doi.org/10.1038/nature14236

Mucientes, M., Casillas, J., 2007. Quick design of fuzzy controllers with good interpretability in mobile robotics. IEEE Trans. Fuzzy Syst. 15, 636–651. https://doi.org/10.1109/TFUZZ.2006.889889

Nauck, D., Kruse, R., 1998. A neuro-fuzzy approach to obtain interpretable fuzzy systems for function approximation, in: 1998 IEEE International Conference on Fuzzy Systems Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98CH36228). IEEE, pp. 1106–1111. https://doi.org/10.1109/FUZZY.1998.686273

Noel, M.M., Pandian, B.J., 2014. Control of a nonlinear liquid level system using a new artificial neural network based reinforcement learning approach. Appl. Soft Comput. J. 23, 444–451. https://doi.org/10.1016/j.asoc.2014.06.037

Rong, H., Angelov, P.P., Gu, X., Bai, J., 2018. Stability of Evolving Fuzzy Systems Based on Data Clouds. IEEE Trans. Fuzzy Syst. 26, 2774–2784. https://doi.org/10.1109/TFUZZ.2018.2793258

Ross, S., Gordon, G.J., Bagnell, J.A., 2010. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In AISTATS 15, 627–635. http://arxiv.org/abs/1011.0686

Rummery, G., Niranjan, M., 1994. On-line Q-learning using connectionist systems (Technical report). University of Cambridge, Department of Engineering Cambridge, England.

Samsudin, K., Ahmad, F.A., Mashohor, S., 2011. A highly interpretable fuzzy rule base using ordinal structure for obstacle avoidance of mobile robot. Appl. Soft Comput. 11, 1631–1637. https://doi.org/10.1016/j.asoc.2010.05.002

Schaal, S., 1999. Is imitation learning the route to humanoid robots? Trends Cogn. Sci. 3, 233–242. https://doi.org/10.1016/S1364-6613(99)01327-3

Sheen, M., n.d. Reinforcement Learning Example - Pendulum Controller w/ Animation [WWW Document]. URL https://ww2.mathworks.cn/matlabcentral/fileexchange/57882-reinforcement-learning-example-pendulum-controller-w-animation (accessed 3.31.19).

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M., 2014. Deterministic policy gradient algorithms, in: 31st International Conference on Machine Learning, ICML 2014. pp. 605–619.

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

Sutton, R.S., McAllester, D.A., Singh, S.P., Mansour, Y., 2000. Policy gradient methods for reinforcement learning with function approximation, in: Advances in Neural Information Processing Systems. pp. 1057–1063.

Takagi, T., Sugeno, M., 1985. Fuzzy Identification of Systems and Its Applications to Modeling and Control, in: IEEE Transactions on Systems, Man and Cybernetics. Elsevier, pp. 116–132. https://doi.org/10.1109/TSMC.1985.6313399

Verma, A., Murali, V., Singh, R., Kohli, P., Chaudhuri, S., 2018. Programmatically interpretable reinforcement learning. 35th Int. Conf. Mach. Learn. ICML 2018 11, 8024–8033.

Wang, L.X., Mendel, J.M., 1992. Fuzzy Basis Functions, Universal Approximation, and Orthogonal Least-Squares Learning. IEEE Trans. Neural Networks 3, 807–814. https://doi.org/10.1109/72.159070

Watkins, C.J.C.H., 1989. Learning from delayed rewards. King’s College, Cambridge.

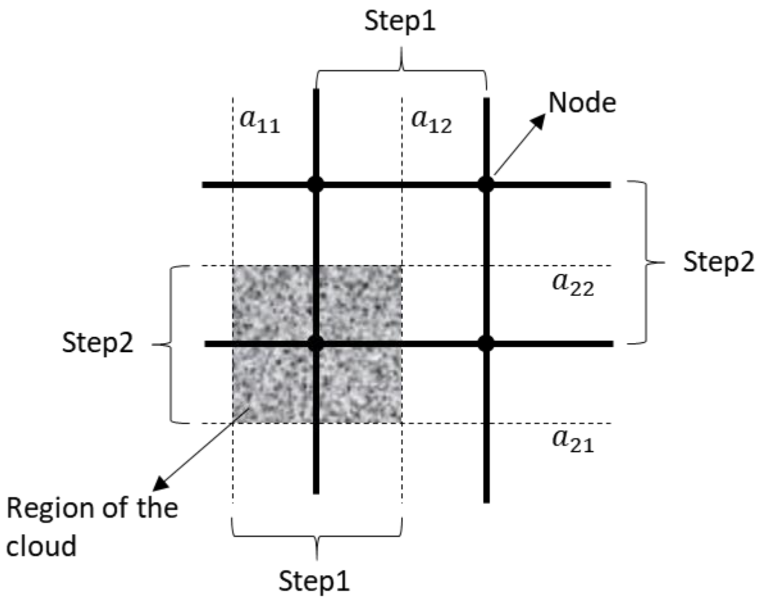
Zadeh, L.A., 1965. Fuzzy sets. Inf. Control 8, 338–353. https://doi.org/10.1016/S0019-9958(65)90241-X

Figure Captions List

Table Captions List

1. Derivation of Eq. (7):

The coordinate variable for the i^{th} dimension is denoted with a_i (recall the role of $x, y, z, \ldots$). a_{i1} and a_{i2} are the two coordinates ($a_{i2} > a_{i1}$) of the border on the rectangular data cloud.



Suppose that the data samples are randomly distributed within the cloud, the variance is

$$\sigma^2 = \mathrm{E}(\|\mathbf{X}\|^2) - \|\mathrm{E}(\mathbf{X})\|^2$$
$$= \frac{\int_{a_{11}}^{a_{12}} \int_{a_{21}}^{a_{22}} \ldots \int_{a_{n1}}^{a_{n2}} (a_1^2 + a_2^2 + \cdots + a_n^2) \mathrm{d}a_1 \mathrm{d}a_2 \ldots \mathrm{d}a_n}{(a_{12} - a_{11})(a_{22} - a_{21}) \ldots (a_{n2} - a_{n1})}$$
$$- \left[\left(\frac{a_{11} + a_{12}}{2} \right)^2 + \left(\frac{a_{21} + a_{22}}{2} \right)^2 + \cdots \right.$$
$$\left. + \left(\frac{a_{n1} + a_{n2}}{2} \right)^2 \right] \quad (1)$$
$$= \frac{1}{3} \cdot \frac{1}{\prod_{i=1}^{n} (a_{i2} - a_{i1})} \cdot \sum_{i=1}^{n} \left((a_{i2}^3 - a_{i1}^3) \prod_{j=1, j \neq i}^{n} (a_{j2} - a_{j1}) \right)$$
$$- \frac{1}{4} \sum_{i=1}^{n} (a_{i1} + a_{i2})^2$$

Note that $a_{i2}^3 - a_{i1}^3$ can be expanded to $(a_{i2} - a_{i1})(a_{i2}^2 + a_{i1}^2 + a_{i2}a_{i1})$, thus Eq. (1)

is

$$
\begin{aligned}
\sigma^2 &= \frac{1}{3} \cdot \frac{1}{\prod_{i=1}^{n}(a_{i2} - a_{i1})} \\
&\qquad \cdot \sum_{i=1}^{n} \left((a_{i2}^2 + a_{i1}^2 + a_{i2}a_{i1}) \prod_{j=1}^{n} (a_{j2} - a_{j1}) \right) \\
&\qquad - \frac{1}{4} \sum_{i=1}^{n} (a_{i1} + a_{i2})^2 \\
&= \frac{1}{3} \sum_{i=1}^{n} (a_{i2}^2 + a_{i2}^2 + a_{i1}a_{i2}) - \frac{1}{4} \sum_{i=1}^{n} (a_{i1} + a_{i2})^2 \\
&= \frac{1}{12} \sum_{i=1}^{n} (a_{i2} - a_{i1})^2 \\
&= \frac{1}{12} \sum_{i=1}^{n} step_i^2
\end{aligned}
\tag{2}
$$

The highlighted parts in Eq. (2) are cancelled.

2. The Sarsa(λ) algorithm with fuzzy features and linear function approximation using AnYa:

Note: Please refer to Section 12.7, page 303 of the textbook for reinforcement learning by Prof. Sutton and Prof. Barto, which can be downloaded via http://incompleteideas.net/book/RLbook2018.pdf.

```
Algorithm parameters:
    step size (learning rate)  $\alpha > 0$,
    trace decay rate  $\lambda_{tr} \in [0,1]$,
    discount rate  $\gamma \in [0,1]$
    probability of taking a random action  $\varepsilon \in [0,1]$

Initialize:  $\mathbf{C} = \mathbf{0}$

Loop for each episode:
    Initialize  $\mathbf{S}$
    Choose  $\mathbf{A}$  near greedily from  $\mathbf{S}$  using  $\mathbf{C}$
    Calculate  $\boldsymbol{\lambda}(\mathbf{S},\mathbf{A}) = [\lambda_1 \quad \lambda_2 \quad \dots \quad \lambda_N]$, with  $\lambda_j = \frac{D_j}{\sum_{l=1}^{N} D_l}$,  $D_j = \frac{1}{1+\frac{\|\mathbf{x}-\boldsymbol{\mu}_\mathbf{j}\|^2}{\sigma_j^2}}$,
        $\mathbf{x} = \begin{pmatrix}\mathbf{S}\\ \mathbf{A}\end{pmatrix}$,  $\boldsymbol{\mu}_\mathbf{j} = \begin{pmatrix}\mathbf{S}_\mathbf{j}^*\\ \mathbf{A}_\mathbf{j}^*\end{pmatrix}$,  $\sigma = \sqrt{\frac{1}{12}\sum_{i=1}^{n} step_i^2}$
    $\mathbf{z} \leftarrow \mathbf{0}$
    Loop for each step of episode:
    |   Take action  $\mathbf{A}$, observe  $R$,  $\mathbf{S}'$
    |   $\hat{q}(\mathbf{S},\mathbf{A}) \leftarrow \boldsymbol{\lambda}^{\mathrm{T}}\mathbf{C}$
    |   $\delta \leftarrow R - \hat{q}(\mathbf{S},\mathbf{A})$
    |   $\mathbf{z} \leftarrow \mathbf{z} + \boldsymbol{\lambda}$
    |   If  $\mathbf{S}'$  is terminal then:
            $\mathbf{C} \leftarrow \mathbf{C} + \alpha\delta\mathbf{z}$
            Go to next episode
    |   Choose  $\mathbf{A}'$  near greedily from  $\mathbf{S}'$  using  $\mathbf{C}$
    |   Calculate  $\boldsymbol{\lambda}'(\mathbf{S}',\mathbf{A}')$  with  $\mathbf{S}'$  and  $\mathbf{A}'$
    |   $\hat{q}(\mathbf{S}',\mathbf{A}') \leftarrow \boldsymbol{\lambda}'^{\mathrm{T}}\mathbf{C}$
    |   $\delta \leftarrow \delta + \gamma\hat{q}(\mathbf{S}',\mathbf{A}')$
    |   $\mathbf{C} \leftarrow \mathbf{C} + \alpha\delta\mathbf{z}$
    |   $\mathbf{z} \leftarrow \gamma\lambda_{tr}\mathbf{z}$
    |   $\mathbf{S} \leftarrow \mathbf{S}'$,  $\mathbf{A} \leftarrow \mathbf{A}'$,  $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda}'$
```

3. Forming the grid in other ways

Suppose that the number of segments/intervals in each dimension is s_i, $i = 1,2,\dots,n$. The total number of nodes is

$$s = \prod_{i=1}^{n} s_i \tag{1}$$

If s_i is the same for every dimension, Eq. (1) turns into

$$s = \bar{s}^n \tag{2}$$

where $\bar{s}$ is the unified number of segments in each dimension. This implies that the number of nodes in the grid grows exponentially with the number of dimensions.

To alleviate it, other methods for selecting prototypes should be applied. Specially, new methods should be able to decouple the number of nodes and the number of dimensions. The simplest way is to randomly select s samples in the state-action space, and use them as nodes of the grid. In this case, calculation of σ should be reformulated. Denote the volume of the state-action space as V, and each sample is allocated with

$$V_i = \frac{V}{s} \tag{3}$$

Suppose that V_i is occupied by a hypersphere of n dimensions. If sampling randomly within the hypersphere, the variance is

$$\sigma^2 = \int_0^R r^2 \mathrm{d}r = \frac{R^3}{3} \tag{4}$$

where R is the radius of the hypersphere. Eq. (4) implies that the standard deviation σ can be expressed as the function of R:

$$\sigma = \frac{R^{\frac{3}{2}}}{\sqrt{3}} \tag{5}$$

Volume of a n-dimensional hypersphere is

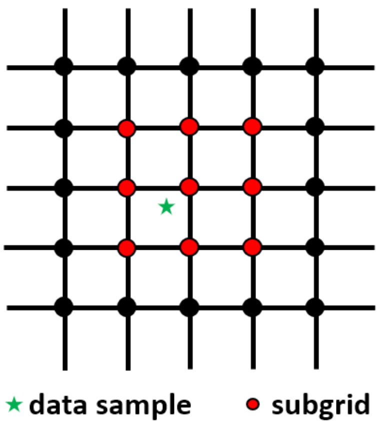
$$V_i = \frac{\pi^{n/2} R^n}{\Gamma(1+\frac{n}{2})} \tag{6}$$

Therefore, σ is

$$\sigma = \frac{1}{\sqrt{3}} \left(\Gamma(1+\frac{n}{2}) \cdot \frac{V}{s} \cdot \frac{1}{\pi^{n/2}} \right)^{\frac{3}{2n}} \tag{7}$$

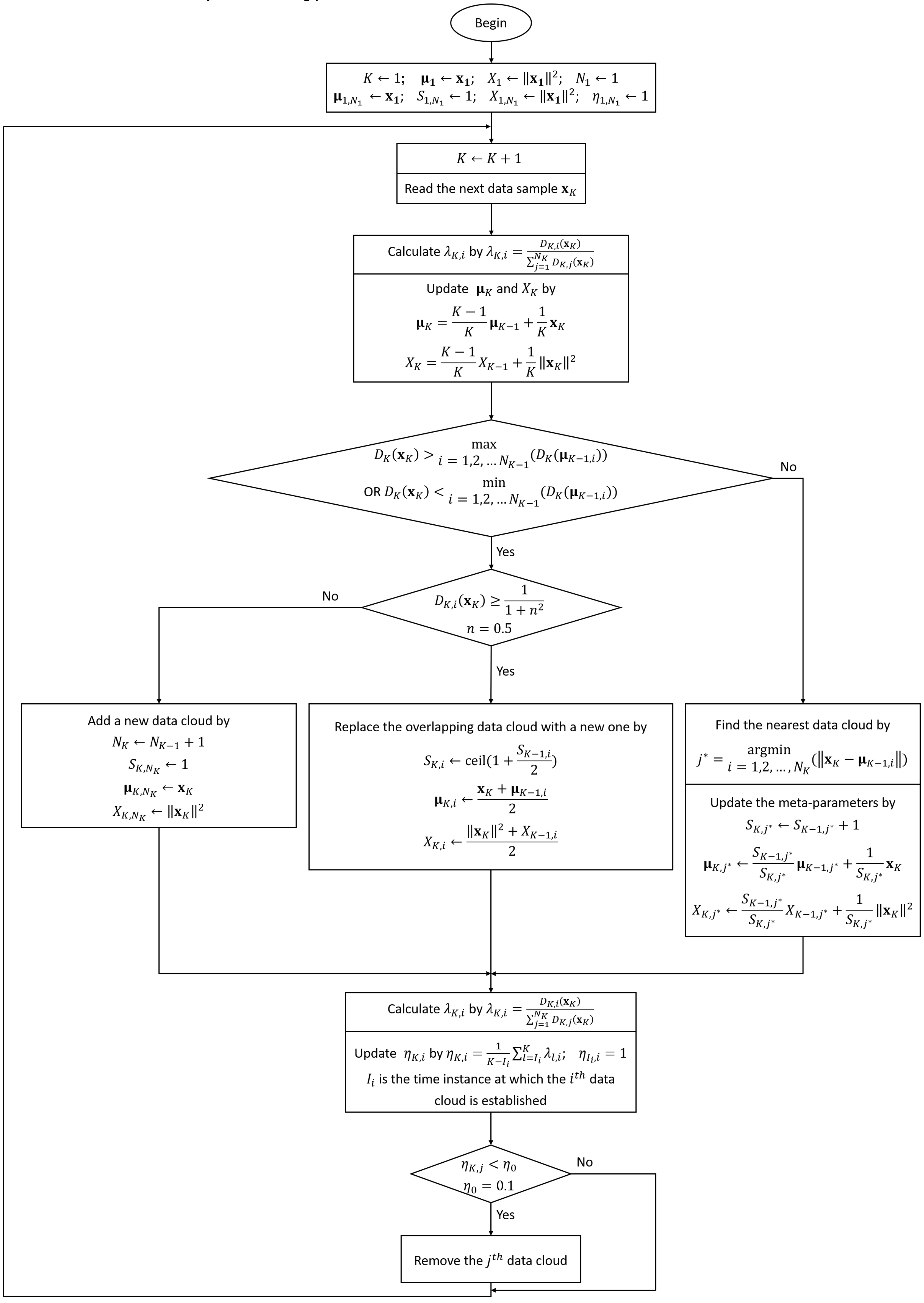
In this way, the number s is subject to specifics of the problem and needs not to be exponential in the number of dimensions.

For some problems, randomly determined prototypes may not produce satisfactory results. This is simply because the resolution of the grid is not large enough to distinguish between good or bad action candidates if the number of nodes in the grid is too small. This is the law of nature: higher-dimensional space contains more information and requires more resources for representation. In these cases, dimensionality reduction techniques in both the state and action space, like changing the way of state representation (e.g. using relative quantities rather than absolute ones as the state variable), removing irrelevant components in the action variable (those make little difference to the reward), will be helpful. However, if dimensionality reduction is impossible, methods which reduce resources consumption only on computation but not memory should be applied. Specifically, note that most components of $\boldsymbol{\lambda}$ are approximately zero, especially those corresponding with nodes far away from the current data sample. For high-dimensional state-action space, the number of nodes can be quite large. It is unnecessary to perform value evaluation and consequents update with the whole $\boldsymbol{\lambda}$ vector. Rather, only parts of it have to be used. Upon each time step, a subgrid near the encountered data sample is used for calculation, rather than the full one. This reduces computations significantly. Memory resources required can be reduced by other techniques like feature hashing[1], which is out of the scope of this paper.



[1] Weinberger, K.Q., Dasgupta, A., Langford, J., Smola, A.J., Attenberg, J., 2009. Feature hashing for large scale multitask learning, in: International Conference on Machine Learning.

4. Flowchart of the ALMMo system's learning process[2]



Begin
$K \leftarrow 1$; $\boldsymbol{\mu}_1 \leftarrow \mathbf{x}_1$; $X_1 \leftarrow \|\mathbf{x}_1\|^2$; $N_1 \leftarrow 1$
$\boldsymbol{\mu}_{1,N_1} \leftarrow \mathbf{x}_1$; $S_{1,N_1} \leftarrow 1$; $X_{1,N_1} \leftarrow \|\mathbf{x}_1\|^2$; $\eta_{1,N_1} \leftarrow 1$
$K \leftarrow K+1$
Read the next data sample $\mathbf{x}_K$
Calculate $\lambda_{K,i}$ by $\lambda_{K,i} = \frac{D_{K,i}(\mathbf{x}_K)}{\sum_{j=1}^{N_K} D_{K,j}(\mathbf{x}_K)}$
Update $\boldsymbol{\mu}_K$ and X_K by
$\boldsymbol{\mu}_K = \frac{K-1}{K}\boldsymbol{\mu}_{K-1} + \frac{1}{K}\mathbf{x}_K$
$X_K = \frac{K-1}{K}X_{K-1} + \frac{1}{K}\|\mathbf{x}_K\|^2$
$D_K(\mathbf{x}_K) > \max_{i=1,2,\dots N_{K-1}}(D_K(\boldsymbol{\mu}_{K-1,i}))$
OR $D_K(\mathbf{x}_K) < \min_{i=1,2,\dots N_{K-1}}(D_K(\boldsymbol{\mu}_{K-1,i}))$
No
Yes
No
$D_{K,i}(\mathbf{x}_K) \geq \frac{1}{1+n^2}$
$n = 0.5$
Yes
Add a new data cloud by
$N_K \leftarrow N_{K-1} + 1$
$S_{K,N_K} \leftarrow 1$
$\boldsymbol{\mu}_{K,N_K} \leftarrow \mathbf{x}_K$
$X_{K,N_K} \leftarrow \|\mathbf{x}_K\|^2$
Replace the overlapping data cloud with a new one by
$S_{K,i} \leftarrow \mathrm{ceil}(1 + \frac{S_{K-1,i}}{2})$
$\boldsymbol{\mu}_{K,i} \leftarrow \frac{\mathbf{x}_K + \boldsymbol{\mu}_{K-1,i}}{2}$
$X_{K,i} \leftarrow \frac{\|\mathbf{x}_K\|^2 + X_{K-1,i}}{2}$
Find the nearest data cloud by
$j^* = \operatorname{argmin}_{i=1,2,\dots,N_K}(\|\mathbf{x}_K - \boldsymbol{\mu}_{K-1,i}\|)$
Update the meta-parameters by
$S_{K,j^*} \leftarrow S_{K-1,j^*} + 1$
$\boldsymbol{\mu}_{K,j^*} \leftarrow \frac{S_{K-1,j^*}}{S_{K,j^*}}\boldsymbol{\mu}_{K-1,j^*} + \frac{1}{S_{K,j^*}}\mathbf{x}_K$
$X_{K,j^*} \leftarrow \frac{S_{K-1,j^*}}{S_{K,j^*}}X_{K-1,j^*} + \frac{1}{S_{K,j^*}}\|\mathbf{x}_K\|^2$
Calculate $\lambda_{K,i}$ by $\lambda_{K,i} = \frac{D_{K,i}(\mathbf{x}_K)}{\sum_{j=1}^{N_K} D_{K,j}(\mathbf{x}_K)}$
Update $\eta_{K,i}$ by $\eta_{K,i} = \frac{1}{K-I_i}\sum_{l=I_i}^{K}\lambda_{l,i}$; $\eta_{I_i,i} = 1$
I_i is the time instance at which the i^{th} data cloud is established
$\eta_{K,j} < \eta_0$
$\eta_0 = 0.1$
No
Yes
Remove the j^{th} data cloud

[2] Reproduced from: Angelov, P.P., Gu, X., Principe, J.C., 2017. Autonomous learning multi-model systems from data streams. IEEE Trans. Fuzzy Syst. 6706, 1–12. https://doi.org/10.1109/TFUZZ.2017.2769039

Variables and meanings:

K: time instance, at each time instance only one data sample is observed
$\mathbf{x}_K$: data sample observed at the K^{th} time instance
$\boldsymbol{\mu}_K$: mean of **all** the data samples observed till the K^{th} time instance
X_K: average scalar product of **all** the data samples observed till the K^{th} time instance

$$X_K = \sum_{l=1}^{K} \frac{1}{K} \|\mathbf{x}_K\|^2$$

$\sigma_K^2 = X_K - \|\boldsymbol{\mu}_K\|^2$
N_K: number of data clouds at the K^{th} time instance

$\boldsymbol{\mu}_{K,j}$: focal point of the j^{th} data cloud at the K^{th} time instance

$S_{K,j}$: number of members of the j^{th} data cloud at the K^{th} time instance

$X_{K,j}$: average scalar product of **members of the j^{th} data cloud** at the K^{th} time instance
$D_K(\mathbf{x}_K)$: unimodal discrete density of $\mathbf{x}_K$ calculated using **all** the data samples observed till the K^{th} time instance

$$D_K(\mathbf{x}_K) = \frac{1}{1 + \frac{\|\mathbf{x}_K - \boldsymbol{\mu}_K\|^2}{\sigma_K^2}}$$

$D_{K,i}(\mathbf{x}_K)$: unimodal discrete density of $\mathbf{x}_K$ calculated using **members of the i^{th} data cloud**

$$D_{K,i}(\mathbf{x}_K) = \frac{1}{1 + \frac{S_{K-1,i}^2 \left\|\mathbf{x}_K - \boldsymbol{\mu}_{K-1,\mathbf{i}}\right\|^2}{(S_{K-1,i}+1)(S_{K-1,i}X_{K-1,i} + \|\mathbf{x}_K\|^2) - \left\|\mathbf{x}_K + S_{K-1,i}\boldsymbol{\mu}_{K-1,i}\right\|^2}}$$

$\lambda_{K,j}$: firing strength of $\mathbf{x}_K$ regarding the j^{th} rule, or activation level of $\mathbf{x}_K$ regarding the j^{th} data cloud

$$\lambda_{K,i} = \frac{D_{K,i}(\mathbf{x}_K)}{\sum_{j=1}^{N_K} D_{K,j}(\mathbf{x}_K)}$$

$\eta_{K,j}$: utility of the j^{th} data cloud at the K^{th} time instance

Utility is defined as averaged $\lambda_{l,j}$ since the establishment of the j^{th} data cloud. If the j^{th} data cloud is formulated at the I_j time instance, then $\eta_{I_j,j}$ is defined to be 1. For other time instances after the birth of the data cloud, $\eta_{K,j} = \frac{1}{K-I_j}\sum_{l=I_j}^{K} \lambda_{l,j}$.

5. Using other kinds of orderings in Section 4.3

In Section 4.3, it is stated the learning problem of multivariate action space with M dimensions can be solved by conditioning each component of the action variable over a series of artificial states: A_1 on $\mathbf{S}$, A_2 on $[\mathbf{S}\quad A_1]$, A_3 on $[\mathbf{S}\quad A_1\quad A_2]$, and so on, up to A_M. It is also possible to use the opposite ordering: A_M on $\mathbf{S}$, A_{M-1} on $[\mathbf{S}\quad A_M]$, A_{M-2} on $[\mathbf{S}\quad A_M\quad A_{M-1}]$, and so on, up to A_1.

In fact, $A_1, A_2, \ldots, A_M$ and $\mathbf{S}$ can be combined in any order. For example, if $\mathbf{A} = [A_1\quad A_2\quad A_3\quad A_4]^{\mathrm{T}}$, then the conditions can be $\mathbf{S}$, $[\mathbf{S}\quad A_3]$, $[\mathbf{S}\quad A_3\quad A_1]$, $[\mathbf{S}\quad A_3\quad A_1\quad A_2]$. Or they can be $\mathbf{S}$, $[\mathbf{S}\quad A_4]$, $[\mathbf{S}\quad A_4\quad A_2]$, $[\mathbf{S}\quad A_4\quad A_2\quad A_3]$.

This can be proved by definitions and theorems on conditional probability:

a) The definition of conditional probability is

$$P(Y|X) = \frac{P(XY)}{P(X)} \tag{1}$$

b) The Multiplication Rule:

$$\begin{aligned} &P(A_1A_2 \ldots A_M\mathbf{S}) \\ &\qquad = P(\mathbf{S}) \cdot P(A_1|\mathbf{S}) \cdot P(A_2|\mathbf{S}A_1) \cdot P(A_3|\mathbf{S}A_1A_2) \cdot \ldots \\ &\qquad \cdot P(A_M|\mathbf{S}A_1A_2 \ldots A_{M-1}) \end{aligned} \tag{2}$$

According to Eq. (1) and Eq. (2) above,

$$\begin{aligned} P(A_1A_2 \ldots A_M|\mathbf{S}) &= \frac{P(A_1A_2 \ldots A_M\mathbf{S})}{P(\mathbf{S})} \\ &= P(A_1|\mathbf{S}) \cdot P(A_2|\mathbf{S}A_1) \cdot P(A_3|\mathbf{S}A_1A_2) \cdot \ldots \\ &\cdot P(A_M|\mathbf{S}A_1A_2 \ldots A_{M-1}) \end{aligned} \tag{3}$$

Note that the left side of Eq. (3) is the policy for state $\mathbf{S}$.

Since the ordering and combination of $A_1, A_2, \ldots, A_M$ on the right side of Eq. (2) do not matter, they can appear in any order on the right side of Eq. (3).

6. Pseudocode and source code of IFRL

Both will be provided here (details to be supplemented later):

https://ww2.mathworks.cn/matlabcentral/fileexchange/73473-interpretable-fuzzy-reinforcement-learning