Optimal Control with Reinforcement Learning using Reservoir Computing and Gaussian Mixture

István Engedy, Gábor Horváth
Department of Measurement and Information Systems
Budapest University of Technology and Economics
Budapest, Hungary
engedy@mit.bme.hu, horvath@mit.bme.hu

***Abstract*—Optimal control problems could be solved with reinforcement learning. However it is challenging to use it with continuous state and action spaces, not to speak about partially observable environments. In this paper we propose a reinforcement learning system for partially observable environments with continuous state and action spaces. The method utilizes novel machine learning methods, the Echo State Network, and the Incremental Gaussian Mixture Network.**

Keywords-optimal control; reinforcement learning; ESN; IGMN

I. INTRODUCTION

Reinforcement learning (RL) is often used to solve optimal control problems. In these problems the goal is to calculate a control policy that is optimal with respect to a criterion function. Some examples one can consider are the well-known mountain car problem, or the inverted pendulum problem, which are benchmark tasks for optimal control algorithms. The effectiveness of reinforcement learning could be experienced the most in complex nonlinear or even discontinuous control problems, non-stationary control problems, where classical control methods cannot be used.

In this paper we present a reinforcement learning based method that is able to solve the optimal control problem in case of partially observable, continuous state spaces. To do that, a probabilistic function approximator and a special recurrent neural network are used together to store and improve the control policy. Both methods are novel approaches in the field of machine learning, and both have been used in itself successfully with reinforcement learning, however we does not know any published results of using these methods together combined with reinforcement learning. These methods are used in order to improve or fix certain issues with the classical reinforcement learning. The proposed method improves the control policy by perceiving the reward signal while interacting with the environment, which is given in the form of a simulator.

In classical reinforcement learning the problem is defined as a Markov Decision Process (MDP): the agent (or the controller) has to choose between the possible actions at a given state, and based on the chosen action, it perceives the next state from the environment (or the system) and a reward signal. The aim of the agent is to find the optimal policy, which is when followed, maximizes the expected reward of the agent over an infinite time horizon.

Reinforcement learning methods introduce a value function, which gives the expected discounted total reward at a given state for a given action [1]. In the basic reinforcement learning methods the value function is stored in a tabular format. It is very efficient for low dimensional discrete state and action spaces, because it can be stored and updated the easiest way possible.

However most control problems have continuous state spaces and often they have continuous action spaces too. There are a number of ways to deal with continuous spaces in reinforcement learning. The most basic way is to partition the state space into discrete cells, and use one of the tabular methods on these discrete cells. However this often results in so many states, that it would be impractical to visit them all, and to learn a near-optimal policy, they might be needed to be visited multiple times.

To overcome this problem, function approximators, like neural nets or linear function approximators are often used in continuous state spaces. They are practical, not just because they can store a continuous value function, but because they can also interpolate the value function between states, so the agent has some knowledge about never visited states. However there is another problem with this approach. Since these function approximators are iterative algorithms themselves, the two iterative learning algorithms working together simultaneously (the RL algorithm and the function approximator), might cause convergence problems [2]. In continuous action spaces the problem is even more complex, because to find the optimal policy, the global maximum of the value function in respect of the action must be found at each state.

In [3] the authors claim that the use of non-parametric function approximators is more adequate for RL, because it is not possible to guess the needed parameterized family in RL problems. Different non-parametric function approximators were used in RL in the recent years. In [4] and in [5] Gaussian Processes (GP) were used to represent the value function. Using GP in RL has an additional benefit that it also provides the variance of the value function, thus indicating the direction in the action space where exploration is needed the most. In [6]

This work has been supported by the Hungarian Scientific Research Fund. (OTKA), grant number TS-73496, and by the New Hungary Development Plan (Project ID: TÁMOP-4.2.1/B-09/1/KMR-2010-0002).

the authors proposed the fitted Q Iteration algorithm using randomized trees. A Gaussian Mixture Model (GMM) was used for the approximation of the value function in [3]. The advantage of GMM is that it can be trained incrementally and it is a probabilistic representation. It represents the observed samples of the RL algorithm in the joint space of states, actions, and Q-value with a mixture of normal distributions. In this paper we follow the same idea.

Reinforcement learning converges to an optimal solution if the environment has the Markov property. That is when a state signal summarizes past sensations compactly, in such a way that all relevant information is retained, and no additional information could be gained by using past sensations beside the immediate sensations.

However in many control problems the state of the system is only partially observable, thus the environment does not have the Markov property. These environments could be modeled as partially observable Markov decision processes (POMDP). To overcome this problem, various machine learning algorithms are used to somehow remember past state signals.

One approach is to use an Echo State Network (ESN), which is a novel recurrent neural network [7]. This method uses a large untrained recurrent neural network to remember past state signals, and it uses a linear weight matrix to generate the output from the recurrent neural network. In [8] such a network is used in RL with discrete action space. The outputs of the ESN were the value functions of each action. The authors also proved the convergence of their method. In [9] a similar method was presented, but there the input of the ESN was the joint state action space, and the only output was the value function. In this paper we will follow a similar approach.

In the sequel we will present a solution to solve POMDP's in continuous state spaces that utilizes Q-learning with a function approximator called echo state incremental Gaussian mixture network (ESIGMN). The rest of this paper is organized as follows. Section II describes each method in detail. In section III the unified method is presented. In section IV preliminary test results are shown. Section V concludes the paper.

II. Methods in detail

A. Reinforcement Learning

The goal of reinforcement learning is to find an optimal policy π^*, which maximizes the total reward the agent gets from the environment during its lifetime. To do this, most reinforcement learning algorithms use a value function or *Q*-function, which gives the expected discounted total reward at a given state for a given action, following policy π. The policy is determined based on this *Q*-function: in every state the action with the greatest *Q*-value is to be selected. It is proven that the greedy policy – always selecting the best action – over the *Q*-values of the optimal policy is equal to the optimal policy [1]. Thus if the *Q*-values of the optimal policy are known, then the optimal policy itself is known. Because of this, most reinforcement learning algorithms focus on determining the *Q*-values of the optimal policy, by means of learning.

The *Q*-function is estimated during the learning, by making a series of actions and perceiving the reward meanwhile, using a reinforcement learning update rule, like *Q-learning* (1) or *SARSA*. Both of these rules are based on the temporal difference learning, the Bellman equation.

$$\Delta Q(s_t, a_t) = \alpha[R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (1)$$

In this equation *Q(s, a)* is the *Q*-value of action *a* in state s. R_{t+1} is the perceived reward at the next state. The parameter α is the learning rate, γ is the discount factor, a real number in [0, 1]. That is needed to avoid divergence of the total reward in case of non-episodic problems. This learning rule backs up the perceived reward to the *Q*-values of the previous states.

$$\pi(s) = \begin{cases} \text{argmax}_a Q(s, a), & P(\varepsilon) \\ \text{random action}, & P(1-\varepsilon) \end{cases} \quad (2)$$

In order to actually improve the value function, sometimes new random moves must be made, to explore new actions and new states. This is called the exploration versus exploitation problem. Usually it is solved with stochastic policies, which tends to choose the best action, but not deterministically. A very simple, yet effective such policy is the ε-greedy policy (2), which takes the best action with ε probability, and a random action with (1 – ε) probability.

B. Incremental Gaussian Mixture Network

Incremental Gaussian Mixture Network (IGMN) [10] and its predecessors, Incremental Gaussian Mixture Model (IGMM) and Incremental Probabilistic Neural Network (IPNN) [11] are machine learning methods, using a mixture of multivariate normal probabilistic distribution functions. All of these methods are incremental, in the sense that they can learn by presenting data points only once to these algorithms, without the need to store the data points. Both IPNN and IGMN are function approximators, while IGMM is a trainable model, using unsupervised learning. While IPNN considers the probability distribution functions as a neural network, IGMN rather uses probabilistic inference to calculate the output. In the sequel, we will use the IGMN method only. Both IGMN and IPNN were successfully used in reinforcement learning problems as function approximators for the *Q*-value [10], [11].

The Gaussian mixture model inside IGMN is over the joint space of the input and output of the network. This makes it possible to infer any part of the input or output data from the known data. As the network is trained, new components can be added, and irrelevant or spurious components with low prior probability could be removed from the model. The training of the network is done by maximizing the likelihood of each incoming data point by adjusting the parameters in the model, just like in the Expectation Maximization (EM) algorithm.

Each component in the model has a couple of parameters. C_j is the covariance matrix and μ_j is the mean of the *j*th normal distribution function. *P(j)* is the prior probability, *v(j)* is the age, and sp_j is the accumulator of the *j*th component. The updating of the model is done as follows. First, the likelihood of each component is calculated from the multivariate normal probability distribution function (3):

$$P(x|j) = \frac{1}{(2\pi)^{D/2}\sqrt{|C_j|}} e^{-\frac{(x-\mu_j)^T C_j^{-1}(x-\mu_j)}{2}} \quad (3)$$

After that, the posterior is calculated based on this likelihood, and the prior (4):

$$P(j|x) = \frac{P(x|j)P(j)}{\sum_{i=1}^{M} P(x|i)P(i)} \quad (4)$$

The rest of the training algorithm is done as follows, according to [11]:

$$v_j(t) = v_j(t-1) + 1 \quad (5)$$

$$sp_j(t) = sp_j(t-1) + P(j|x) \quad (6)$$

$$e_j = x - \mu_j \quad (7)$$

$$\omega_j = \frac{P(j|x)}{sp_j} \quad (8)$$

$$\Delta\mu_j = \omega_j e_j \quad (9)$$

$$\mu_j(t) = \mu_j(t-1) + \Delta\mu_j \quad (10)$$

$$C_j(t) = C_j(t-1) + \Delta\mu_j \Delta\mu_j^T + \omega_j\left[e_j e_j^T - C_j(t-1)\right] \quad (11)$$

$$P(j) = \frac{sp_j}{\sum sp_i} \quad (12)$$

Where $v_j(t)$ is the age of the j^{th} Gaussian, which is checked to remove only old enough Gaussians; $sp_j(t)$ is the accumulator, to calculate the P(j) prior probability; and $\mu_j(t)$ and $C_j(t)$ are the mean and covariance of the j^{th} Gaussian respectively.

The training is started with an empty model. A new component is added to the model when the following criterion (13) is met:

$$P(x|j) < \frac{\tau_{nov}}{(2\pi)^{D/2}\sqrt{|C_j|}} \; \forall j \quad (13)$$

Here τ_{nov} is a novelty parameter, which tells how much part of the space is actually represented by that distribution, in the sense that a new data point in the outside area bears new information, thus a new component must be created for it. The only parameter at new component creation is the initial covariance matrix, which could reflect the sizes of the space along each dimension.

Components should also be removed from the model after a certain age, if their prior probability is decreasing over time. That means the specific distribution is never selected as a relevant distribution for any data points.

The recall phase of the IGMN in the function approximation case is done using probabilistic inference. Let the data x be the joint of input vector a and output vector b, $x=[a, b]$. The output is calculated as follows (14):

$$\hat{b} = \sum_{i=1}^{M} P(j|a)\left[\mu_{j,b} + C_{j,ba}C_{j,aa}^{-1}(a - \mu_{j,a})\right] \quad (14)$$

Where $C_{j,ba}$ and $C_{j,aa}$ are submatrices of the covariance matrix.

C. Echo State Network

Echo State Network (ESN) [7] is a novel architecture for recurrent neural networks. As other recurrent neural networks, ESN is capable of modeling nonlinear dynamic systems. Its advantage over other training methods of conventional recurrent neural networks, like backpropagation through time, real time recurrent learning or extended Kalman filtering is that compared to these methods it has fast convergence.

ESN belongs to the family of reservoir computation. Its architecture is shown in Figure 1. The network consists of an input layer, a large recurrent hidden layer of neurons with sigmoid nonlinearity, called the reservoir, and a linear output layer. Only the output layer is trained, the recurrent hidden layer is only initialized with randomized weights that fit a certain criterion. The spectral radius of the hidden layer weight matrix must be less than one, so the network will satisfy the Echo State Property, which is that the reservoir must forget all past information asymptotically.

The linear output layer of the network is trained with a simple linear regression. First the reservoir dynamics is simulated with the input time series. After that the time series of the reservoir states is used with the target time series to calculate the output weights. To achieve smoother solutions in the meaning of generalization capability, either noise is added to the input time series, or ridge regression is used instead of the linear regression.

The equations to calculate the output of the ESN are the following (15, 16):

$$x(t) = f(W_{in}u(t) + Wx(t-1)) \quad (15)$$

$$y(t) = W_{out}x(t) \quad (16)$$

Where W_{in} is the input weight matrix, W is the reservoir weight matrix and W_{out} is the output weight matrix, *f()* is the sigmoid nonlinearity function.

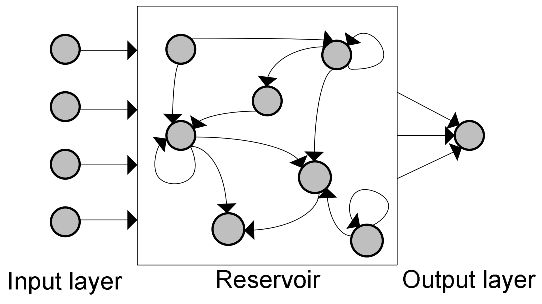

Figure 1. Echo State Network architecture

ESN was successfully used in many partially observable reinforcement learning problems, like in [8]. It should be also noted that in reinforcement learning tasks usually not the previously discussed regression methods are used to find the weights of the output layer, because they need batch training, which is not an option in RL.

III. THE PROPOSED SYSTEM

As we discussed it in the introduction, our aim is to make an intelligent controller for higher level control tasks using machine learning algorithms. Reinforcement learning is a greatly studied area of research, and many results show that it is an adequate general framework for modeling operation in partially observable stochastic environments, thus to solve optimal control tasks. The core framework of our proposed system is reinforcement learning, with the *Q*-learning rule. We assume that a simulator is available for the system, which can be used to generate sample runs, and actually the RL can be trained on it.

To make the system able to cope with partially observable control tasks, we use an ESN to estimate the hidden state variables. Others used ESN for such purpose with success [8]. To do that, we feed the values of the observable state variables and also the action values step by step to a sufficiently sized ESN. Instead of calculating the weights of the output layer, we simply use the outputs of the reservoir to extend the observable state variable. If the reservoir is correctly sized, it is likely that the hidden state variables could be retrieved from the reservoir signals by means of linear combination.

We use an IGMN to store the *Q*-values of the actions for each state. This way we could take the assumption, that both the extended state and action spaces are continuous, however action selection in continuous action spaces are not trivial, so in all of our tests we used only discrete actions. The extended state signal consists of the observable state variables, and the signals of the reservoir. The normal probabilistic distribution functions of the IGMN are taken in the joint space of the extended state, actions and *Q*-values. The *Q*-values are calculated from the network according to (14).

During training we used a constant weight of 0.25 to modify the Gaussians, instead of the calculated value in (8), which is based on the posterior probability of the Gaussian. We did that to achieve a forgetting effect, because during the reinforcement learning, at first, inaccurate Q values are used to update other Q values. The effect of these inaccurate values would fade away asymptotically, as new, accurate Q values are used, but this tends to be slower than just using a forgetting effect.

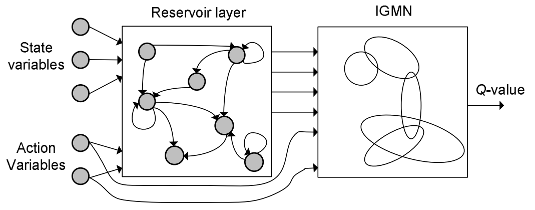

Figure 2. Architecture of the proposed system with the reinforcement

The signals of the reservoir extend the dimensionality of the modeled space inside the IGMN. Because of this, the computational power needs of the algorithm grows exponentially with the size of the ESN, as the number of required Gaussians to cover major part of the extended state space is growing exponentially with the dimensionality, and in each step, the likelihood function of all Gaussians are needed to be calculated. Moreover, the time of the calculation of the likelihood function for each Gaussian scales with the dimensionality in a power of 3, because a matrix inversion is needed to be calculated during it.

To overcome these problems, we introduced several tricks, to reduce the computational complexity of the proposed system. First, we chose the obvious solution to always have the smallest possible echo state network. Beyond this we also used a space partitioning algorithm called XTree [12], which is an extension of the R*-tree algorithm. This is used to divide the space into intervals, and build a tree from these. The leaf nodes of the tree are the axis-aligned bounding boxes of those parts of the Gaussians, where they are considered relevant, based on the τ_{nov} novelty parameter in (13). The parent nodes are always the minimal bounding boxes of their children. This way we can find the relevant Gaussians for a given state, action and Q-value, by expanding those branches of the tree only, that contains the specified point. This algorithm is especially efficient in high dimensional spaces, and its computational complexity scales with the logarithm of the stored Gaussians.

This way we were able to calculate the likelihood of only those Gaussians, that appeared to be relevant based on the XTree algorithm. We considered the likelihood of all the other Gaussians zero.

We also utilized a matrix computational trick for the calculation of the inverse of the covariance matrix of the Gaussians during the computation of the likelihood. Instead of calculating the inverse of the matrix, we store not just the covariance matrix, but also its inverse. When the covariance matrix is modified when the Gaussian in question is updated, we also modify the stored inverse to match. This way we have to use twice as much memory, but the complexity of this modification scales with the dimension only in power of 2, instead of 3 for the direct inverse calculation.

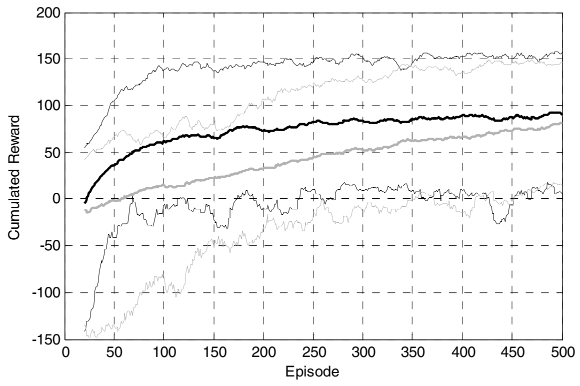

Figure 3. Total reward per episode during the training. Min, max and average values. Grey is a tabular solution,black is the ESIGMN solution

IV. Test results

We have tested the proposed system on a limited torque inverted pendulum task, where only the angle variable was observable, the angular velocity was hidden. There were only two actions, to apply the same amount of torque to the pendulum from left or right. The parameters of the task were selected such way, that the pendulum must make multiple left-right transitions in order to get to the instable equilibrium position.

The angular velocity and the angle were calculated as follows:

$$\ddot{\varphi} = (lmg \sin \varphi + T - \mu\dot{\varphi})/(l^2 m) \tag{17}$$

Where φ is the angle, l=1 is the length, m=1 is the mass of the pendulum, g=1 is the gravity, T=±0.5 is the torque, and μ=0.03 is a friction parameter.

The reward function was the cosine of the angle of the pendulum, so when the pendulum was in the instable equilibrium point, the reward was 1 in each time step. The system was trained over 500 episodes, 200 time steps each episode. In each episode the pendulum was started from a random state.

The training was performed 20 times, and the minimal, maximal and average cumulated rewards were calculated at the end of each episode. These values are shown in black in Figure 3, filtered with a moving average of 20 episodes. To compare these results, the same values of a tabular method are also shown in grey. The tabular method used a 100 by 100 table to store the Q-values of each state (angle-angular velocity pair).

As the tests shown, the proposed method was able to improve the performance of the policy. It outperformed the tabular method, and eventually after about 70-100 episodes of training the inverted pendulum spent most of the time near the goal position. The average number of Gaussian distributions learned by the IGMN part was 179. We used 5 neurons in the reservoir for this test, which proved to be enough, based on the fact that the method was able to eventually get the pendulum to the instable equilibrium point.

V. Conclusion

In this paper we have presented an optimal control method that utilizes several novel machine learning techniques. It is able to solve the optimal control problems from the aspect of reinforcement learning. This approach gives a clear framework to define the dynamics of the system, the goals of the control, and the sensors and actuators, in the form of a simulator, reward function, observable variables, and possible actions. Moreover our method is able to deal with unobservable state variables by implicitly predicting them using an echo state network.

The most important element of the method is the IGMN, which stores the learned information in the form of a mixture of Gaussian distributions. The optimal action for each state is extracted from these. We have also presented how a space partitioning algorithm can be used to decrease the computational complexity of IGMN, allowing using it for complex, high dimensional problems.

As our test results shown, the method outperforms the simple tabular reinforcement learning methods. This means that the function approximation is not just an alternative way to tables, to store the Q-values; it actually gives the advantage of being able to generalize the knowledge over the state-action values in a probabilistic manner, thus requiring less training episodes to achieve the same results.

The probabilistic approach of the function approximator has further, yet unexploited potential. Now only the expected value of the Q-value is used in the reinforcement learning, however the variance could also be used during reinforcement learning. This is the main direction of our research for the future.

References

[1] Sutton, R., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA (1998)

[2] S. Thrun and A. Schwartz: Issues in using approximation for reinforcement learning. In Proceedings of the Fourth Connectionist Models Summer School, Hillsdale, NJ, 1993. Lawrence Erlbaum Publisher.

[3] A. Agostini and E. Celaya. Reinforcement learning with a Gaussian mixture model. In Proc. Int. Joint Conf. on Neural Networks (IJCNN'10). (Barcelona, Spain), pages 3485–3492, 2010.

[4] Y. Engel, S. Mannor, and R. Meir, "Reinforcement learning with Gaussian processes," in ICML '05: Proceedings of the 22nd international conference on Machine learning. New York, NY, USA: ACM, 2005, pp. 201–208.

[5] M. Diesenroth, C. Rasmussen, and J. Peters, "Gaussian process dynamic programming," Neurocomputing, vol. 72, no. 7-9, pp. 1508–1524, 2009.

[6] D. Ormoneit and S. Sen, "Kernel-based reinforcement learning," Machine Learning, vol. 49, no. 2-3, pp. 161–178, 2002.

[7] Jaeger, H.: Tutorial on training recurrent neural networks, covering BPTT, RTRL,EKF and the 'echo state network' approach. Technical Report GMD Report 159,German National Research Center for Information Technology (2002)

[8] Szita, I., Gyenes, V., Lőrincz, A.: Reinforcement learning with echo state networks. In: Artificial Neural Networks ICANN 2006. Lecture Notes in Computer Science, vol. 4131/2006, pp. 830–839. Springer Berlin / Heidelberg (2006)

[9] K. Bush. "An Echo State Model of non-Markovian Reinforcement Learning". Ph.D. thesis, Colorado State University, Fort Collins, CO, 2008

[10] Heinen, M.R. "A Connectionist Approach for Incremental Function Approximation and On-line Tasks". Porto Alegre, RS. Ph.D. Thesis. Universida-de Federal do Rio Grande do Sul – UFRGS, 172 p. (2011)

[11] Heinen, M.R., Engel, P.M. "An incremental probabilistic neural network for regression and reinforcement learning tasks". In: K. DIA-MANTARAS; W. DUCH; L.S. ILIADIS, Artificial Neural Networks – ICANN 2010. Berlin, Springer-Verlag, p. 170-179.

[12] S. Berchtold, D. A. Keim, H.-P. Kriegel: The X-tree : An Index Structure for High-Dimensional Data. VLDB 1996: 28-39