ESANN'2006 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 26-28 April 2006, d-side publi., ISBN 2-930307-06-4.

Reducing Policy Degradation in Neuro-Dynamic Programming

Thomas Gabel and Martin Riedmiller
Neuroinformatics Group
University of Osnabrück, 49069 Osnabrück, Germany

Abstract. We focus on neuro-dynamic programming methods to learn state-action value functions and outline some of the inherent problems to be faced, when performing reinforcement learning in combination with function approximation. In an attempt to overcome some of these problems, we develop a reinforcement learning method that monitors the learning process, enables the learner to reflect whether it is better to cease learning, and thus obtains more stable learning results.

1 Introduction

The basic idea behind reinforcement learning (RL) is to let (software) agents acquire a control policy on their own on the basis of trial and error by repeated interaction within their environment. Aiming at the application of RL to complex, real-world problems with large and continuous state spaces, it becomes indispensable to make use of a value function approximation mechanism. In this work, we will focus on neuro-dynamic approaches (NDP) for that task and employ neural networks as approximation architecture [2]. Though most theoretical results regarding the convergence behavior of RL algorithms do not generally hold in the presence of value function approximation, impressive results could be obtained in the past, e.g. Tesauro's TD-Gammon.

However, the sequences of control policies found during learning with function approximation generally do not converge. A typical phenomenon in the NDP context, as reported in a number of publications, is that the approximated value function, from which the control policy is induced, improves during the initial iterations and, after having reached the vicinity of the optimum, tends to oscillate. Therefore, we introduce a reinforcement learning method that confers a better control over the learning process, enables the agent to know if it is better to stop learning, and thus clearly reduces the oscillations of the agent's performance obtained.

2 Approximate Neuro-Dynamic Programming

RL problems can be modelled as Markov Decision Processes (MDP). An MDP consists of sets of states S and actions A, a stochastic transition function $p(s,a,s')=p_{sas'}$ that tells how likely it is to end up in state s', when taking action a in state s, as well as a function of immediate costs $c(s,a)$ that arise when taking action a in state s. If the actions available depend on the current state the system is in, the set of actions is usually denoted as $A(s)$. The goal of learning within an MDP is to find a policy $\pi : S \rightarrow A$ that minimizes the expected accumulated costs. Q learning [9] is a suitable RL method to learn a value function, if there is no explicit model of the environment available. It updates directly the estimates for the values of state-action pairs according to $Q(s,a) :=$

$(1-\gamma)Q(s,a) + \gamma(c(s,a) + min_{b\in A(s')}Q(s',b)$ where the successor state s' and the immediate cost $c(s,a)$ are generated by simulation or by interaction with a real process. For the case of finite state/action spaces where the Q function can be represented using a look-up table, Q learning is guaranteed to converge to the optimal value function $Q^\star$ under mild assumptions. Given convergence to $Q^\star$, the optimal policy $\pi^\star$ can be induced by greedy exploitation of Q according to $\pi^\star(s) = \arg\min_{a\in A(s)} Q^\star(s,a)$.

Working with high-dimensional, continuous state spaces, we need to utilize a function approximator to represent Q. Then, most of the convergence results regarding Q learning no longer hold. Concerning function approximation we pursue a neuro-dynamic approach (NDP), representing the value function with a multi-layer perceptron neural network. We focus on networks of this type since they are universal approximators and feature good generalization capabilities. With function approximation, no direct assignment of Q values is possible: Instead, the approximator used must be fitted to the Q function. If r denotes the network's weights, a common way to approximate the optimal value function by a function $Q^r(s,a)$ is based on minimizing the error in Bellman's equation, i.e. $\min_r \sum_{(s,a)\in C}(Q^r(s,a) - \sum_{s'}(p_{sas'}(c(s,a) + \min_{b\in A(s')} Q^r(s',b)))^2$ (with C as set of representative sample state-action pairs [2] and $p_{sas'}$ and $c(s,a)$ being estimated from data).

2.1 Troubles in Converging

Does Bellman error minimization in conjunction with function approximation imply convergence to a (near-)optimal policy? In general, it is expected that convergence cannot be achieved and a phenomenon called "chattering" occurs. The space $\mathbb{Q}$ of all Q functions can be divided into *greedy regions* [2] where a constant policy (given by greedy Q exploitation) is followed. Each such region corresponds to a different greedy policy and has its own *greedy point*—the point in $\mathbb{Q}$ to which $Q^r(\cdot)$ moves in the course of learning. If that greedy point does not lie in its greedy region, the policy learnt may fluctuate between two or more greedy policies sharing the same boundary in $\mathbb{Q}$ [5].

There are various publications reporting on negative results using RL approaches with function approximation, many of which head into a specific direction: The policy the learning agent acquires quickly reaches a remarkable quality, but in the further course of learning a significant policy degradation can be observed. For instance, Bertsekas and Tsitsiklis [2] report on attempts to learn a policy for playing the game of Tetris and point out the paradoxical observation that high performance is achieved after relatively few policy iterations, but then performance drops. Similar observations are reported by Gaskett [4] working with the cart pole benchmark, Bertsekas et al. [1] in the context of a missile-defense scenario, and other authors. We encountered comparable problems (see Section 4 and [6]): Applying Q learning in a Bellman residual minimizing manner yields excellent policies, but only at some specific stages of learning before a loss in policy quality appears, whereas the Bellman error is being decreased continuously. We need to stress that these issues cannot be (at least not solely) related to effects of overfitting the network to the training data: Oscillations in and degradation of policy performance can also be observed, when the learned policy is applied to situations that were actually covered by the training set on the basis of which the net had been trained.

3 Monitoring Q Iteration

A simple approach to select high-quality policies despite of oscillations during learning is to let the policies generated undergo an additional evaluation based on simulation. Bertsekas et al. [1] call this process of final policy selection *screening*. However, this approach is extremely time-consuming, in particular for optimistic policy iteration or Q learning settings, where a large number of potential policies are created: Each time an update to the current policy has been made (step from policy π_k to π_{k+1}), a large number of test episodes must be run in order to assess the new policy's true quality.

In the following, we develop a different approach to circumvent effects of policy degradation during advanced stages of learning. The basic idea of our approach to monitored Q iteration (MQI) is to (a) define an auxiliary error metric that more directly relates to a policy's actual performance, (b) create the conditions that this error measure can be effectively calculated, and (c) continuously monitor the learning process, remember top-performing policies, and also facilitate an early stopping of learning. So, we will be able to avoid an extensive simulated evaluation of generated policies (screening) and thus save computational resources, while being able to figure out a nearly optimal policy. In its current version, MQI is applicable to environments where $A(s)$ is finite for all $s \in S$.

MQI can be considered as a representative of the class of fitted Q iteration algorithms [3], which compute an approximation of the optimal policy from a finite set (batch) of four-tuples $\mathbb{F} = \{(s_i, a_i, c_i, s'_i) | i = 1, \ldots, p\}$ that correspond to single "experience units" collected by the agent within its environment.

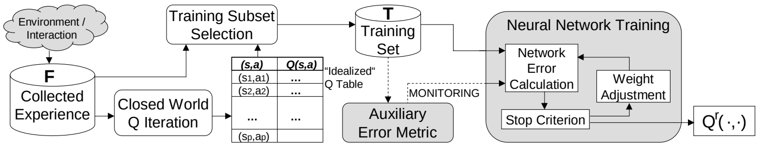

Fig. 1: Building Blocks of MQI

3.1 Closed World Q Iteration

In a preliminary phase, the RL agent interacts with the environment and collects the tuple set $\mathbb{F}$. These training tuples are fed into a procedure we call closed world Q iteration (CWQI). This name tributes to the fact that CWQI abstracts from the real system dynamics and considers the information in $\mathbb{F}$ only. It works like standard Q learning on the finite set of points in state-action space provided by $\mathbb{F}$ and is thus able to compute a value function that can be stored in a look-up table Q^F. The only precondition CWQI requires is that for each state s that is part of a tuple in $\mathbb{F}$, there must exist (at least) one successor state s' to be found in another training tuple in $\mathbb{F}$ (or s' is a goal state corresponding to an episode end). This requirement is fulfilled when the training data is gathered along trajectories. The result is an "idealized" value table Q^F which reflects the cost structure and transition probabilities contained in the training batch $\mathbb{F}$.

3.2 Auxiliary Error Metric

Of course, the idealized Q table returned by CWQI is imperfect with respect to the real system—since it stems from a finite batch of interaction experience only. If $Q^\star$ denotes the optimal value function and $a^\star_s$ a corresponding optimal action in a state s, then there are most likely states s_i for which it holds that $\arg\min_{a\in A(s_i)} Q^F(s_i, a) \neq a^\star_{s_i}$, since there has been no information on the actual value of taking action $a^\star_{s_i}$ in state s_i in the set of training tuples. Thus, for training the neural net, we select only those tuples $(s_i, c_i, r_i, s'_i) \in \mathbb{F}$ for which a substantial share of available actions $A(s_i)$ has been tried (selection of training subset $\mathbb{T}$). We argue that with the idealized Q table's entries, some (near-)optimal policy can be obtained. Particularly, we assume that: If $s \in S$ is a state covered by the training set $\mathbb{T}$, then it holds $Q^F(s, a_i) \leq Q^F(s, a_j) \Leftrightarrow Q^\star(s, a_i) \leq Q^\star(s, a_j)$ for all $a_i, a_j \in A(s)$. So, for each $s \in S$ covered within $\mathbb{T}$, we can determine the optimal action and, moreover, a ranking for all actions $a \in A(s)$, which is identical to the order of actions when a total ordering with respect to $Q^\star(s, a)$ would be defined.

Though that assumption will be sometimes violated in practice, we define and make use of the following auxiliary error measure: Let Q^r denote the value function currently represented by the net and let $rank_{Q^r} : S \times A \to \mathbb{N}$ with $(s, a) \mapsto |\{b \in A(s) | Q^r(s, b) < Q^r(s, a)\}|$ define a ranking on the finite number of actions available in state s (action with highest Q value has first rank). Then, we compute the *index error* $IE_{Q^r}(\mathbb{T})$ for a given training set $\mathbb{T}$ and value function Q^r as

$$IE_{Q^r}(\mathbb{T}) := \sum\ _{((s,a),Q^F(s,a))\in\mathbb{T}} \frac{|rank_{Q^F}(s, a) - rank_{Q^r}(s, a)|}{|\mathbb{T}|}$$

The idea behind the index error is to have an indicator that states to which extent Q^r matches the intended ranking of actions for one specific state. If we have achieved finding a Q^r for which $IE_{Q^r}(\mathbb{T}) = 0$, then the corresponding policy π_{Q^r} will behave optimally w.r.t. Q^F and so optimally w.r.t. the experience collected in the environment.

3.3 Neural Network Training

Adjusting the network weights with respect to the error on the training set $\mathbb{T}$, we make use of backpropagation. The error to be minimized is $e = \frac{1}{|\mathbb{T}|} \sum_{((s,a),Q^F(s,a))\in\mathbb{T}} (Q^F(s, a) - Q^r_k(s, a))^2$ with Q^r_k as the value function currently represented by the net. It is known that excessive minimization of the mean squared error on some finite training set may worsen the net's generalization capabilities (overfitting). Apart from that, we have seen (cf. Section 2.1) that a minimized Bellman error does not necessarily correspond to maximized policy performance. In particular, when considering the behavior on the training set only, i.e. disregarding the desire to generalize, Bellman error minimization on the training data may lead to a policy which less often chooses the optimal action, even on the states that are covered in the training set. With the index error, however, we have a measure that captures the correctness of a policy's action choice and helps us in finding (near-)optimal policies more reliably. We use it as an alternative error measure, while iterating in the neural network training building block of MQI (learning by epoch) and hence minimizing the mean squared (Bellman) error: The value of $IE_{Q^r_k}(\mathbb{T})$ is monitored continuously, the iteration μ, at which the minimal index error could be achieved, is remembered, and Q^r_μ (represented by the neural net) is returned.

4 Empirical Evaluation

To evaluate MQI we chose the application domain of reactive job shop scheduling [7]. The learner's task is to autonomously acquire dispatching policies to assign jobs to a limited number of resources, where each job consists of a certain number of operations to be processed on specific resources. For a detailed description of the RL job shop scheduling scenario we refer to [8]. The basic idea of this alternative approach to scheduling, however, is to model the environment as an MDP[1] and have a learning agent at some of the resources, that decides which job to process next based on its local view on the entire plant. During learning it shall adapt its behavior so that production cost minimization, i.e. minimal overall tardiness (job due date violations), is achieved.

All experiments involved 3 cooperative resources one of which was equipped with a learning (Q or MQI) agent and 2 stationary ones working according to some simple dispatching rule, preferring jobs with earliest due date (EDD), minimal slack (MS), or shortest/longest processing time (SPT/LPT). Each experiment is divided into a train and test phase: A random set S_A of training scheduling scenarios and an independent set S_B of testing scenarios is generated (all differing in the properties and numbers of jobs to be processed). During training, the scenarios in S_A are processed repeatedly where the learning agent picks random actions (explores) and that way collects experience, used to learn the state-action value function (represented by a three-layer net). Testing, the S_B scenarios are processed, with the adaptive agent behaving greedily w.r.t. its current Q function, and the overall tardiness (as optimization goal of the plant) is measured.

Degrading Policies: Applying Q learning, the Bellman error on the training set is minimized continuously (see Figure 2, top). In the beginning, policy performance (in terms of tardiness) is improving, too. In the further course of learning, however, the average tardiness increases again. Intuitively, this is the effect one may expect and blame on overfitting. Yet more interesting, in this figure the summed tardiness on the training set (the scheduling scenarios for which the agent's policy is actually being trained) is also shown. It reveals that, while the Bellman error is dropping concurrently, the agent's performance on the training set is worsening (after an initial phase of improvement); the learned policy degrades. Only in one of our experiments (comprising an EDD, an LPT, and the learning resource; not sketched) these effects were negligible.

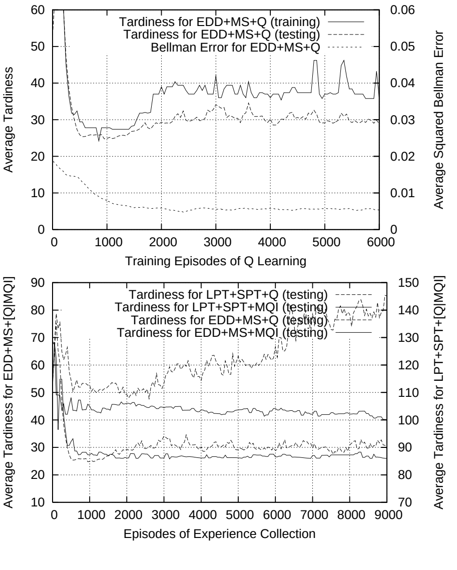

Fig. 2: Degrading Policy (Q Learning, top) and Performance of MQI (bottom)

Stabilization of the Learning Process: The x-axes in Fig.2 (bottom) correspond to episodes of expeience collection: For Q learning this is the time-dependent course of

[1] A state is represented by the situation a resource is in (i.e. properties of the set of waiting jobs), actions correspond to the decision for processing a specific job next, and costs arise due to due date violations.

learning, while MQI first collects experience interacting with its environment and then starts fitting the value function. So, for MQI each data point along the x-axis represents a single learning process using all the experience collected in as many episodes.

Clearly, MQI's results are much less dependent on the amount of episodes of experience collection used for learning. Moreover, in both scenarios sketched the policies learned by MQI are better than the ones found by Q learning, regardless of the size of the four-tuples set $\mathbb{F}$. On S_b, standard Q learning achieves better scheduling performance only in the EDD+ MS+[Q|MQI] scenario (Table 1), but only during the first few hundred training episodes. Yet, this good result would have been only detected by exhaustive policy screening.

Tardi-	EDD+LPT+		EDD+MS+		LPT+SPT+		MS+SPT+	
ness	Q	MQI	Q	MQI	Q	MQI	Q	MQI
Best	88.5	**85.6**	**24.8**	25.4	107.7	**99.5**	58.7	**57.3**
Avg	93.2	**92.1**	31.1	**26.3**	130.9	**103.8**	63.2	**61.0**
Worst	100.5	**95.3**	36.9	**28.3**	146.7	**115.9**	67.4	**64.4**
StdDev	2.50	**1.48**	1.89	**0.51**	7.37	**3.05**	1.34	1.32

Table 1: Considering learning processes of varying length (1000 to 40000 scheduling episodes), this table opposes average, best/worst case tardinesses, and learning result fluctuations of the learnt policy for Q learning and MQI and for different environments, i.e. different heuristically deciding agents.

5 Conclusion

In approximate RL where the state-action value function is represented by a function approximator, the learned policy may degrade after an initial phase of improvement. As a way to combat this problem and to circumvent the need for an additional policy screening process to evaluate intermediate policies, we have proposed an RL method that reliably learns a near-optimal state-action value function from a batch of experience. MQI facilitates the definition of an auxiliary error metric by which the learning process can be monitored and eventually ceased to prevent the learner from policy degradation and overfitting. Empirically, we have investigated MQI's capability to stabilize the learning process in the application field of reactive production scheduling.

Acknowledgements: This research has been supported by the German Research Foundation (DFG) under grant number Ri 923/2-1.

References

[1] D. Bertsekas, M. Homer, D. Logan, S. Patek, and N. Sandell. Missile Defense and Interceptor Allocation by NDP. In *IEEE Transactions on Systems, Man, and Cybernetics*, pages 42–51, 2000.

[2] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro Dynamic Programming*. Athena Scientific, Belmont, 1996.

[3] D. Ernst, P. Geurts, and L. Wehenkel. Tree-Based Batch Mode Reinforcement Learning. *Journal of Machine Learning Research*, (6):504–556, 2005.

[4] C. Gaskett. *Q-Learning for Robot Control*. Ph.D. Thesis, Australian National University, 2002.

[5] G. Gordon. Stable fitted reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 8, pages 1052–1058. The MIT Press, 1996.

[6] W. Hunger and M. Riedmiller. Scheduling with adaptive agents - an empirical evaluation. In *Proceedings of EWRL-5, European Workshop on Reinforcement Learning*, 2001.

[7] Michael Pinedo. *Scheduling. Theory, Algorithms, and Systems*. Prentice Hall, USA, 2002.

[8] S. Riedmiller and M. Riedmiller. A Neural Reinforcement Learning Approach to Learn Local Dispatching Policies in Production Scheduling. In *IJCAI99*, pages 764–771, Stockholm, Sweden, 1999.

[9] C. Watkins and P. Dayan. Q-Learning. *Machine Learning*, 8:279–292, 1992.