ESANN 2011 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2011, i6doc.com publ., ISBN 978-2-87419-044-5. Available from http://www.i6doc.com/en/livre/?GCOI=28001100817300.

Ensemble Usage for More Reliable Policy Identification in Reinforcement Learning

Alexander Hans[1,2] and Steffen Udluft[2]

1- Ilmenau University of Technology – Neuroinformatics & Cognitive Robotics Lab
P.O. Box 100565, D-98684 Ilmenau – Germany

2- Siemens AG, Corporate Technology – Intelligent Systems & Control
Otto-Hahn-Ring 6, D-81739 Munich – Germany

Abstract. Reinforcement learning (RL) methods employing powerful function approximators like neural networks have become an interesting approach for optimal control. Since they learn a policy from observations, they are also applicable when no analytical description of the system is available. Although impressive results have been reported, their handling in practice is still hard, as they can fail at reliably determining a good policy. In previous work, we used ensembles of policies from independent runs of neural fitted Q-iteration (NFQ) to produce successful policies more reliably. In this paper we evaluate the approach on more problems and propose to form ensembles from successive iterations of a single NFQ run as a computationally cheap alternative to completely independent runs.

1 Introduction

Reinforcement learning (RL) [1] deals with optimal control problems, where often the notion of an agent interacting with an environment is used. In each step, the agent observes the state of the environment and executes an action, thereby causing a transition to a successor state. Along with each transition a reward is given, a scalar value evaluating that transition. The agent's aim is to find a policy, mapping each state to an action, that maximizes the expected sum of future rewards. If the environment can be formulated as a Markov decision process (MDP) $M := (S, A, P, R)$, where S is the set of states, A the set of actions, $P : S \times A \times S \mapsto [0, 1]$ the transition probabilities, and $R : S \times A \times S \mapsto \mathbb{R}$ the reward function, the optimal policy π maximizes the value function

$$V^\pi(s) = \max_a \sum_{s'} P(s'|s,a) \left[R(s,a,s') + \gamma V^\pi(s')\right], \tag{1}$$

where γ is the discount factor. The optimal policy's Q-function is also a solution to the Bellman optimality equation

$$Q^*(s,a) = \sum_{s'} P(s'|s,a) \left[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right]. \tag{2}$$

When the MDP's parameters are known, this equation can be used to iteratively determine Q^* (starting from an arbitrary Q), from which the optimal policy follows as $\pi^*(s) := \arg\max_a Q^*(s,a)$. However, this is only feasible for

ESANN 2011 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2011, i6doc.com publ., ISBN 978-2-87419-044-5. Available from http://www.i6doc.com/en/livre/?GCOI=28001100817300.

environments featuring discrete state and action spaces with a small number of state-action pairs. If the state space is continuous or the number of state-action pairs is large, one has to resort to function approximation to overcome the storage problem and achieve data-efficiency by generalizing to afore unobserved state-action pairs.

Neural fitted Q-iteration (NFQ) [2] is an RL method that achieves data-efficiency by employing neural networks and re-using all observations, i.e., performing batch-mode RL. While [2] reports very positive results, problems have been reported as well [3]: often after a number of iterations a near-optimal policy is found, but after a few more iterations the policy degrades, only to become near-optimal again a few iterations later. This is partly due to effects like *chattering* [4] and systematic overestimation of Q-values [5]. As a result, it is hard to know what iteration's policy to use as final solution.

In previous work we suggested to use ensembles to address this issue [6]. Ensembles have been used successfully in machine learning to improve the prediction quality by combining several individual predictors (e.g., [7]). The smaller the correlation of the errors of the ensemble members, the greater the expected improvement. To introduce diversity and thereby cause errors to be less correlated, methods like bagging [8] and boosting [9] have been suggested. So far only few contributions exist that employ ensembles for RL. Wiering and van Hasselt used ensembles of quite different algorithms trained with the same data for discrete MDPs in an online setting [10]. Each algorithm determines its own policy, the final policy is determined from the individual policies, e.g., using majority voting. Ernst et al. used ensembles of regression trees in a fitted Q-iteration approach [11]. Instead of using only one regression tree to represent the Q-function in each iteration, they used an ensemble of trees (random forest).

In the present work we evaluate the ensemble usage proposed in [6] for further environments. In addition to ensembles composed of policies from individual NFQ runs, we propose and evaluate the usage of ensembles composed of policies from different iterations of a single NFQ run.

2 Neural Fitted Q-Iteration

NFQ [2] is an instance of the fitted Q-iteration (FQI) [11] approach. FQI iteratively finds a solution to the Bellman optimality equation (2) just like dynamic programming, but instead of storing the Q-function tabularly, a function approximator is used. See Alg. 1 for a summary of FQI. In NFQ, the function approximator is a neural network (multi-layer perceptron). Due to the excellent generalization properties of neural networks, it is possible to produce near-optimal policies very data-efficiently [2].

For our NFQ implementation we use a neural network with an input layer, two hidden layers, and an output layer. For all neurons in the hidden and output layer we use the hyperbolic tangent as activation function, the input layer neurons use the identity. We initialize the Q-function as $\forall(s,a) \in S \times A : Q^0(s,a) := 0$, i.e., in the first step we learn the rewards. The inputs are scaled

Algorithm 1: The basic FQI algorithm. $\hat{Q}^k$ is a function approximator, $\hat{Q}^k(s,a)$ gives the value of $\hat{Q}^k$ evaluated with input (s,a).

```
Input: observation tuples $(s_i, a_i, r_i, s_i')$, $\gamma$
Result: $\hat{Q}^*$
begin
    $\hat{Q}^0 := 0$
    $i := 0$
    while the desired precision is not reached do
        $\text{input}_j := (s_j, a_j)$
        $\text{target}_j := r_j + \gamma \max_a \hat{Q}^i({s_j}', a)$
        $\hat{Q}^{i+1} := \text{train}(\text{input}, \text{target})$
        $i := i + 1$
    return $\hat{Q}^i$
```

to have a mean of zero and a standard deviation of one, the targets are scaled using $\hat{t}_i := t_i/(1.05 \cdot t_{\max})$, where $t_{\max}$ is the maximum absolute target. The hyperbolic tangent limits the output of the network to lie within $[-1, 1]$. This helps to avoid the "rising Q problem" [5], while scaling with $1.05 \cdot t_{\max}$ allows the network to produce the desired output without having to use very large weights in the connector between the second hidden and the output layer.

3 Improving Reliability through Ensembles

To address the problem of policy degradation, in [6] we combined several completely individually learned Q-functions to an ensemble policy using majority voting and Q-averaging. It was shown that ensemble policies are more often successful than individual ones, especially when using majority voting. However, for an improvement of reliability it might not even be necessary to do completely independent NFQ runs. Instead, one could use an ensemble of policies from a number of final iterations. E.g., assuming the Q-function converges after 100 iterations, one could use Q^{100}, Q^{100+1}, Q^{100+2}, ..., Q^{100+n} to form an ensemble. In theory, those Q-functions should be identical. In practice with NFQ, however, they differ slightly, causing occasionally inferior policies.

We will evaluate the following methods:

1. Ensembles of successive iterations of converged NFQ runs.
2. Calculating ensembles from independent NFQ runs.

4 Experiments

We conducted experiments using the pole balancing, cart-pole, and wet-chicken benchmarks. For all domains we used random exploration to generate the observations.

ESANN 2011 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2011, i6doc.com publ., ISBN 978-2-87419-044-5. Available from http://www.i6doc.com/en/livre/?GCOI=28001100817300.

4.1 Benchmark Domains

Pole Balancing In the pole balancing benchmark a pole attached to a cart must be kept in upright position by applying forces to the cart. Starting from an upright position, in each time step the agent can choose to apply -50 N, 0 N, or $+50$ N to the cart. The actions are corrupted by uniformly distributed noise $n \in [-10, 10]$ N. Therefore, the trivial policy of always applying 0 N does not lead to success, even though initially the pole is in upright position. The two-dimensional state space consists of the pole's angle θ and the angular velocity $\dot{\theta}$. A reward of 0 is given if $\theta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$. If the pole leaves this area a reward of -1 is given and the episode ends. The time constant used is $\Delta t = 0.1$ s, the discount factor is set to $\gamma = 0.95$. A policy is considered successful if it is able to repeatedly balance the pole for at least 3,000 steps.

To generate observations of the MDP, we used episodes of random exploration. When applying actions randomly, the pole falls (and therefore the episode ends) after approximately six steps. We used data sets of 25 episodes (≈ 150 observations/transitions). We generated 50 data sets and used those to generate policies with the different methods. To assess a policy's quality, it was run 100 times for at most 3,000 steps.

Cart-Pole The cart-pole problem is an extension of the pole balancing benchmark. In addition to the balancing task, the agent must keep the cart within a certain range, while the highest reward is achieved when keeping the cart in the middle of the track. Therefore, the state space is extended by the cart's position x and its velocity $\dot{x}$. We used the same settings as were used in [2] ($\Delta t = 0.02$, $\gamma = 0.95$), except for the reward function. We used a distance based reward measure with added bonus for the target region and punishment for states leading to failure:

$$r_t := \begin{cases} -|x_t| - |\theta_t| + 1 - 2x_t^2/0.05^2 + x_t^4/0.05^4 & \text{if } |x_t| < 0.05 \wedge |\theta_t| < 0.15, \\ -|x_t| - |\theta_t| - 10(|\theta_t| - 0.2) & \text{if } |\theta_t| > 0.2, \\ -|x_t| - |\theta_t| & \text{otherwise.} \end{cases}$$

Observations were generated starting from random positions with zero velocities and $x \in [-2.3, 2.3]$, $\theta \in [-0.2, -0.2]$. For evaluation, the position was randomly initialized with $x \in [-1, 1]$. An episode was stopped when $|x| > 2.4$ or $|\theta| > 0.25$. A policy's quality is reported as average immediate reward from 100 episodes with a maximum length of 3,000 steps. The results reported are averages of 40 completely independent trials, i.e., using 40 distinct data sets.

Wet-Chicken In the wet-chicken benchmark [12] a canoeist paddles on a one-dimensional river with length $l = 20$ and flow velocity $v = 1$. At position $x = l$ of the river there is a waterfall. Starting at position $x = 0$, the canoeist has to try to get as near as possible to the waterfall without falling down. If he falls down, he has to restart at position $x = 0$. The reward increases linearly with the proximity to the waterfall and is given by $r = x$. The canoeist has the

ESANN 2011 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2011, i6doc.com publ., ISBN 978-2-87419-044-5. Available from http://www.i6doc.com/en/livre/?GCOI=28001100817300.

		Pole	Cart-Pole		Wet-Chicken
		25	10,000	30,000	500
single		24	-0.35 ± 0.07	-0.140 ± 0.040	12.9 ± 0.2
successive iterations	2	24	-0.33 ± 0.07	-0.191 ± 0.062	12.7 ± 0.2
	5	25	-0.32 ± 0.08	-0.089 ± 0.029	13.4 ± 0.2
	10	31	-0.21 ± 0.04	-0.146 ± 0.043	13.3 ± 0.2
independent runs	5	30	-0.11 ± 0.02	-0.030 ± 0.005	13.3 ± 0.2
	10	34	-0.07 ± 0.02	-0.021 ± 0.002	13.3 ± 0.2
combined	5×2	32	-0.09 ± 0.02	-0.026 ± 0.003	13.4 ± 0.2
	5×5	31	-0.07 ± 0.01	-0.022 ± 0.002	13.5 ± 0.2
	5×10	34	-0.05 ± 0.01	-0.019 ± 0.002	13.4 ± 0.2
	10×2	32	-0.07 ± 0.01	-0.020 ± 0.002	13.4 ± 0.2
	10×5	32	-0.06 ± 0.01	-0.019 ± 0.001	13.4 ± 0.2
	10×10	33	-0.05 ± 0.01	-0.017 ± 0.001	13.4 ± 0.1

Table 1: Experimental results using policies from a single NFQ run and ensemble policies from a number of final successive iterations from single NFQ runs, completely independent runs, and combinations of both (the first number denotes the number of independent runs, the second the number of final successive iterations). For the pole benchmark the numbers of policies able to balance for 3000 steps are reported (out of 50 trials), for cart-pole and wet-chicken the average immediate reward is shown (from 40 and 100 trials, respectively).

possibility to drift, to hold the position, or to paddle back. River turbulences of size $s = 2.5$ cause the state transitions to be stochastic. Thus, after having applied the canoeist's action to his position (also considering the flow of the river), the new position is finally given by $x' = x + n$, where $n \in [-s, s]$ is a uniformly distributed random value. For our experiments, the discount factor was set to $\gamma = 0.975$.

We used 100 data sets containing 500 observations each. To evaluate a policy's performance, we executed it 100 times for 1000 steps each and determined the average immediate reward. The results are averages of 100 trials using the 100 distinct observation sets.

4.2 Results

Table 1 shows the results of the techniques applied to the three benchmarks. For the pole balancing problem using an ensemble of successive iterations helps to increase policy quality. Although the results here do not indicate a significant performance gain when using policies from completely independent NFQ runs, we believe that in general this leads to better results, as policies from independent runs are less correlated than those from successive iterations of the same run. Moreover, the best result reported in [6] (39/50 successful trials using 40 ensemble members) cannot be matched here even with much larger ensembles, because those contain highly correlated policies.

In both cart-pole settings using ensembles of successive iterations of a single

ESANN 2011 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2011, i6doc.com publ., ISBN 978-2-87419-044-5. Available from http://www.i6doc.com/en/livre/?GCOI=28001100817300.

run does not lead to a significant improvement of performance. On the other hand, combining policies from different runs does give a significant performance gain. Combining policies from successive iterations of independent NFQ runs improves the performance even further.

The improvement for the wet-chicken benchmark is not that obvious. Nonetheless, the combination of successive iterations of individual NFQ runs tends to improve the policy performance.

5 Conclusion

In this paper we extended previous work [6] by further evaluating the usage of ensembles in an NFQ context. Additionally, we proposed and evaluated the usage of policies from successive iterations from a single NFQ run to form an ensemble. In summary, the results from all experiments presented here as well as in [6] indicate that RL with NFQ does always benefit from ensemble usage. When performing multiple independent NFQ runs is too expensive, one can as well use a number of final iterations from single NFQ runs, which gives an ensemble "for free". In recent years, multi-core processors have become standard for commodity hardware and this trend continues. Ensembles constitute an extremely simple way of exploiting the potential of such platforms, since executing multiple NFQ runs in parallel is trivial. Instead of hand-tuning the algorithm for the problem at hand, ensembles allow for more robust and reliable policy identification—we therefore believe that they are an important step on the way toward autonomous RL, needing only little or no intervention of a human expert.

References

[1] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

[2] M. Riedmiller. Neural fitted Q-iteration–first experiences with a data efficient neural reinforcement learning method. In *Proc. of the 16th European Conf. on Machine Learning*, 2005.

[3] T. Gabel and M. Riedmiller. Reducing policy degradation in neuro-dynamic programming. In *Proc. of the European Symposium on Artificial Neural Networks*, 2006.

[4] G.J. Gordon. Reinforcement learning with function approximation converges to a region. *Advances in neural information processing systems*, 2001.

[5] S. Thrun and A. Schwartz. Issues in using function approximation for reinforcement learning. In *Proc. of the 1993 Connectionist Models Summer School*, 1993.

[6] A. Hans and S. Udluft. Ensembles of neural networks for robust reinforcement learning. In *Proc. of the 9th Int'l Conf. on Machine Learning and Applications*, to appear.

[7] T. Dietterich. Ensemble methods in machine learning. *Multiple classifier systems*, 2000.

[8] L. Breiman. Bagging predictors. *Machine learning*, 24(2), 1996.

[9] Y. Freund, R. Schapire, and N. Abe. A short introduction to boosting. *Journal of the Japanese Society for Artificial Intelligence*, 14, 1999.

[10] M.A. Wiering and H. van Hasselt. Ensemble algorithms in reinforcement learning. *IEEE transactions on systems, man, and cybernetics*, 38(4), 2008.

[11] D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6, 2005.

[12] V. Tresp. The wet game of chicken. *Siemens AG, CT IC 4, Technical Report*, 1994.