Feed-Forward Learning: Fast Reinforcement Learning of Controllers[⋆]

Marek Musial and Frank Lemke

Real-Time Systems and Robotics (PDV)
Technische Universität Berlin

Abstract. Reinforcement Learning (RL) approaches are, very often, rendered useless by the statistics of the required sampling process. This paper shows how *very fast* RL is essentially made possible by abandoning the state feedback during training episodes. The resulting new method, *feed-forward learning* (FF learning), employs a return estimator for pairs of a state and a *feed-forward policy's* parameter vector. FF learning is particularly suitable for the learning of controllers, e.g. for robotics applications, and yields learning rates unprecedented in the RL context.

This paper introduces the method formally and proves a lower bound on its performance. Practical results are provided from applying FF learning to several scenarios based on the collision avoidance behavior of a mobile robot.

1 Introduction

Reinforcement learning (*RL*) is a well-known approach that *must* be applied whenever some agent is to learn a behavior and the agent's information is limited to state observations and a reward signal. The general RL background has been elaborately presented e.g. in the excellent book [1] by Sutton and Barto. Numerous algorithms and algorithm variants have been devised within the RL framework, with temporal difference (TD) learning [2] and Q-learning [3] among the best-known thereof. Especially for model-free policy optimization, it is required for existing formal convergence properties [4] that *all state-action pairs [...] be visited an infinite number of times in the limit of an infinite number of episodes* [1]. Speedups have been reported, but mainly result from better utilization of the acquired sampling data, e.g. [5], but not from a simplified sampling process. Moreover, when the action resolution is increased to approach continuous-time modeling, the statistical relevance of the "free" initial action in the state-action-pairs with regard to the expected return vanishes. This leads to the fact that for practical application in non-toy cases, the "infinite number" mentioned in the theorems also constitutes a far-too-good hint at the practically achievable convergence rate.

In order to particularly address the properties of control learning applications, this paper suggests a completely new RL approach. Instead of estimating

[⋆] This work has been conducted within the *NeuRoBot* project, funded by German Research Foundation (DFG).

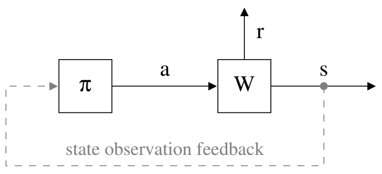

Fig. 1. System diagram of the RL training process: FF learning abandons the state observation feedback

returns for state-action-pairs, as done by classical model-free RL methods like Q-learning, the new method involves a return estimator for *state-policy pairs*. Here, the policies are *deterministic* and *state-independent*, actually parameterized functions over time. The sampling process is completely independent of the policy *currently favored by the agent*, so that all acquired return samples are final and do not require any updates (*backups*) in the course of policy iteration. This results in a dramatic simplification of the sampling statistics, and in turn leads to *very fast* reinforcement learning.

Figure 1 depicts the training process of RL: Some policy π issues actions a to effect the environment (world) W, which in turn yields state observations s and a reward signal r, the use of which is not detailed in this diagram. While traditional RL approaches do use state feedback, depicted through the dashed arrow, to govern the sampling process during training episodes, the new approach eliminates this feedback completely. It is therefore termed *feed-forward learning* (FF learning).

The remainder of this paper is organized as follows: The subsequent section 2 explains the FF learning method formally, section 3 addresses formal properties such as convergence issues and performance bounds, and section 4 gives insight into some relevant implementation details. Result examples from an collision avoidance application for a ground-based mobile robot are presented in section 5 in order to emphasize the applicability and performance of FF learning. Section 6 concludes the paper with an outline of the merits and drawbacks of FF learning.

2 FF Learning Method

This section describes the FF learning algorithm formally. After introducing the basic entities, the description distinguishes between the training phase and the application phase.

For the formal treatment, continuous state and action spaces and continuous time are assumed. However, quantization of some or all of these sets is possible and may be required in an implementation. The learning task is characterized by a set of (observable) *states* $s \in S \subseteq \mathbb{R}^K$ and a set of permissible *actions* $a \in A \subseteq \mathbb{R}^N$. The environment W evolves according to some (unobservable) internal state and the applied action signal $a(t)$, while outputting observable state information $s(t) \in S$ and a *reward* signal $r(t) \in \mathbb{R}$. Furthermore, let $s(t), r(t)$ be piecewise continuous for continuous $a(t)$. There is no assumption whether the process underlying W is a Markov process, or whether it is deterministic.

For return calculation, an infinite horizon, discounted reward problem is generally assumed. However, finite episodes and undiscounted reward can easily be introduced as special cases. An *episode* starts always at $t = 0$, and the *return* $R \in \mathbb{R}$ assigned to time t_0 is defined as

$$R(t_0) = \int_{t=t_0}^{\infty} r(t) \cdot e^{-\gamma(t-t_0)}\, dt \tag{1}$$

with discount rate $\gamma \geq 0$ (with $\gamma = 0$ modeling undiscounted reward). The end of a single episode, either induced by the environment or arbitrarily by the training algorithm, is denoted by $t = T$.

Now, FF learning introduces a set of *feed-forward policies*

$$\pi_p : \mathbb{R}_0^+ \to A \tag{2}$$

defining action trajectories $a(t) = \pi_p(t)$ in dependence of a D-dimensional parameter vector $p \in P \subseteq \mathbb{R}^D$. It is further postulated that the feed-forward policies are parameterized such that the postfix of any of them can be regarded as a new episode start by selecting a suitable parameter vector, i.e. formally:

$$\forall p \in P,\ t_0 \geq 0 \quad \exists p' \in P \quad \forall t \geq 0\ :\ \pi_{p'}(t) = \pi_p(t_0 + t) \tag{3}$$

2.1 Training Phase

All episodes in the *training phase* are executed, without state feedback (see section 1 and Figure 1), by selecting a single parameter vector $p \in P$ of a feed-forward policy. This fully determines the agent's action signal as:

$$a(t) = \pi_p(t) \qquad t \in [0; T] \tag{4}$$

p is selected at the start of the training episode, which may be done randomly or in some way involving the current "application phase" policy (see 2.2), perhaps *ϵ-greedy* as known from classical exploration regimes. But the crucial fact is that p is kept constant throughout every training episode.

The training process on the whole consists in sampling triples

$$(s, p, R) \in S \times P \times \mathbb{R} \tag{5}$$

in order to train a function approximator to estimate a return $\bar{R}(s, p) \in \mathbb{R}$ from a pair of state s and policy parameters p. This is trivially possible for $t_0 = 0$, using $s(0)$, p, and $R(0)$. But due to (3), any number of return triples may be extracted from a single training episode, by using $s(t_0)$, p', and $R(t_0)$ for any instantiation of the variables according to (3). In other words, any episode postfix can be considered and evaluated just as a full episode on its own.

Therefore, the following two facts are met:

1. Samples are obtained as fast as in traditional "backup" algorithms, like Q-learning or TD learning, e.g. one sample per simulation step.

2. All samples obtained are final, i.e. do not depend on the "application policy", and therefore, need not be corrected at any later time. Traditional "backup" algorithms, in contrast to that, must continuously adapt their return estimates together with their policy evolution, which leads to a twofold, convoluted convergence process.

The implementation of the return estimator is irrelevant for the FF learning algorithm as such. Any sort of neural network can be employed, and table-based discretization is also basically possible.

2.2 Application Phase

In the application phase, it is neither desirable nor possible to limit the agent's behavior to one of the feed-forward policies π_p. Instead, the agent uses the *initial* action of the *most promising* feed-forward policy, at any time step. This gives the "application phase" policy $\pi^* : S \to A$ as:

$$\pi^*(s) = \pi_{\arg\max_{p \in P} \bar{R}(s,p)}(0) \tag{6}$$

Here, assume the *arg max* operator works deterministically in a way that keeps π^* piecewise continuous over time, e.g. by always yielding the geometric center of subspaces with equal maximum return.

The implementation of the *arg max* operator needs to be addressed by an actual implementation of FF learning. Clearly, determining this maximum will usually be approximate and may still involve substantial computation cost.

The application phase *does* employ state observation feedback, according to Figure 1.

3 Formal Properties

This section examines the formal performance properties of FF learning. As a part of this, the characteristics of the environment and of the feed-forward policies must be investigated, in relationship to one another.

First of all, it is immediately clear that (6) does *not* denote an optimal policy with respect to (1) with any generality. This simply results from the fact that the look-ahead conducted in action selection according to (6), even in case of a perfect estimator $\bar{R}$, is limited to the family of feed-forward policies used in training and will thus ignore non-representable action trajectories with potentially higher return.

On the other hand, the positive statement is conveyed in the following theorem:

Theorem 1. *Assuming that the process underlying W is Markov*[1] *and provided that the estimator $\bar{R}(s,p)$ in (6) is perfect (i.e. equals the expected return of the*

[1] For the formal proof, the Markov property is required, which is common for the formal examination of RL algorithms. This should *not* be misconceived as restricting the application of FF learning to Markov processes.

feed-forward policy for any p applied from state s onward), then $\bar{R}$ delivers a lower bound on the return of π^*:

$$E\{R(t) \mid s(t), \pi^*\} \;\geq\; \max_{p \in P} \bar{R}(s(t), p) \tag{7}$$

Proof Sketch: Starting with a policy that permanently applies π_p for the *arg-max* p from the right-hand side of (7) and delivers an expected return of $\bar{R}(s(t), p)$, consider one adds a new point τ in the episode time at which p is replaced by the "new" *arg max* according to (6), say p'. Denote the "old" expected return starting from τ with $E\{R(\tau)\}$ and the "new" one after adding the additional parameter lookup at τ with $E\{R'(\tau)\}$. Now, it is

$$E\{R'(\tau)\} \geq E\{R(\tau)\} \tag{8}$$

as π^* picks, for any state $s(\tau)$, the p' with the highest expected return, always considering to keep the "old" feed-forward policy (described by p at the starting time t) because of (3). By applying this argument repeatedly for e.g. $\tau, 2\tau, 3\tau, \ldots$ and using an arbitrarily small τ, the expected return of the so constructed policy converges to the left-hand side of (7) without ever decreasing. $\square$

While Theorem 1 does not provide a particular good quantitative estimate of the return to be expected from π^*, looking at the best-possible performance of the feed-forward policies covered by π_p is usually quite easy and intuitive. By all means, Theorem 1 will be the primary reference when considering how many parameters (degrees of freedom, DOF) D need to be used in the π_p family of feed-forward policies in order to meet the requirements of any specific application. In the limit of infinitely many DOF in π_p, the right-hand side of (7) converges to the optimal value, which then makes π^* an optimal policy according to Theorem 1.

4 Implementation Details

This section explains some details of the example implementation of FF learning that has been used to generate the results presented in section 5. These details are *not* inherent to FF learning, but could be addressed in different ways as well.

4.1 $\bar{R}$ Approximator

As the approximator of the $\bar{R}$ estimate, a standard multilayer perceptron (MLP) with backpropagation training has been used. For its implementation, the *Fast Artificial Neural Network* library (FANN, [6]) has been employed. This library is available in source and provides the standard network structures and training algorithms with very high computation performance. During training, the samples according to (5) are collected in a single training set, which is occasionally used for training the estimator network. In training, a fraction (30%) of this set is randomly selected as a test set for cross-validation and early stopping to avoid overfitting. The network uses two hidden layers with sigmoid activation functions and a single linear output unit for the estimated return. As the actual training algorithm, FANN's *iRPROP* has been used (see [6] for details).

4.2 *arg max* Operator

The implementation of the *arg max* operator in (6), especially for increasing dimensionality of p, is a central issue of FF learning. Due to the potential non-continuity of the *arg max* operator, it does not make sense to try to train an estimator to directly obtain the most promising p vector. Instead, an on-line search has been implemented, in the form of a two step scheme:

1. A global maximum search is performed, starting at the best p vector from the last time step, by checking a number of fixed displacements along all axes. The displacements used, for $P = [-1; +1]^D$, are $\pm 0.01, \pm 0.02, \pm 0.04, \pm 0.1, \pm 0.2, \pm 0.3, \pm 0.5, \pm 1, \pm 1.5$.
2. The best p vector from step 1 is used as the starting point of a line search in the direction of the estimate's gradient, which can be determined based on the network's δ-terms (see [6,7])[2].

While step 1 is intended to allow switching between different local maxima of the return estimate toward the global optimum, step 2 yields a good approximation of the true local (and hopefully, global) optimum at low computation cost.

Clearly, this is a heuristic approach that in no way guarantees to determine the true global maximum. On the other hand, with a small time step size, it is practically sufficient to reach the "correct" local optimum after (or within) a couple of time steps. The application results obtained so far indicate that this heuristic optimum search does perform well enough in practice.

4.3 Sampling Control

Basically, the samples according to (5) need *not* be taken IID according to the probability distribution related to any specific intermediate policy, because the feed-forward policy π_p fully defines the statistics of their emergence. Nevertheless, the least-square-error approximation performed by the approximator's learning process *does* depend on the distribution of the samples according to (5). The sampling process conducted by FF learning, based on the family of feed-forward policies, accounts for rather low mean returns as compared to the final π^* policy. Therefore, it is highly reasonable to exert a special *sampling control* regime to minimize the effect of this low mean on the training of the approximator network.

For this purpose, two special rules have been employed in the sample implementation of FF learning:

1. For any new (s, p) pair to be inserted into the training set, its minimum Euclidean distance $\delta_{\min}$ from any element already in the training is computed. The pair is discarded if it is too close (similar) to some prior one (in the examples: if $\delta_{\min} < 0.2$).

[2] For this purpose, the FANN library has been slightly modified to allow direct access to the δ-terms of the network.

2. The range of possible returns of feed-forward policies is divided into a number B of equally sized bins, each of which does not receive more than $1/B$ of the total maximum number of training samples. That is to stop sampling for frequent low returns earlier than for rare better returns (in the examples: $B = 5$ with a maximum of 20000 samples per bin).

Both measures together aim at enforcing the best-possible approximation of an equal distribution of samples over $S \times P$. This reduces crosstalk from bad to good returns in the approximator's training.

5 Result Examples

This section presents a group of sample applications with varying degree of freedom (DOF) and the learning results obtained therein. All examples pertain to a simulated ground-based mobile robot the movement of which is determined by a translation speed v and a rotation rate ω that are in principle independent. That is, the robot moves, unless $\omega = 0$ or $v = 0$, along a circular arc with radius v/ω. Physically, robots with two independently driven wheels plus one or more support wheels exhibit this kind of kinematics. In the experiments, v is sometimes kept constant, leaving only $N = 1$ DOF in the action $a = \omega$, and sometimes controlled by the policy, which results in the $N = 2$ DOF action $a = (v, \omega)$. In the former case, $v = 50\text{mm/s}$, $-70°/\text{s} \leq \omega \leq 70°/\text{s}$ have been used; in the latter one, $0 \leq v \leq 50\text{mm/s}$, $-30°/\text{s} \leq \omega \leq 30°/\text{s}$.

The (non-Markovian) state input to the return estimator consists of 21 simulated distance sensors, each of which covers a single view sector with an opening angle between 4° and 16°, resulting in a total view angle of 164°. Figure 2 depicts the arrangement of the view sectors. Every component s_i of the state vector $s \in S = [0; 1]^{21}$ indicates the distance r_i between the robot's center and the closest obstacle point in the corresponding view sector relative to the robot's radius $R = 32\text{mm}$ via $s_i = R/r_i$. Using r_i as the denominator ensures higher resolution in the more relevant region of small distances and provides a convenient range for the s_i.

The continuous reward signal $r(t)$ is meant to encourage distance coverage compared to rapid turning in small circles:

$$r(t) = -0.003 \frac{\text{s}}{\text{deg}} \cdot \|\omega(t)\| \; + \; 0.015 \frac{\text{s}}{\text{mm}} \cdot \frac{\|x(T) - x(t)\|}{T - t} \tag{9}$$

The second summand denotes the mean "topological" velocity over the rest of the episode and $x(t)$ the robot's position in the world. A collision with a wall or obstacle terminates the episode and is penalized with an immediate reward impulse of $\int r(t)dt = -0.75$. The discount rate has been set at $\gamma = 0.1\text{s}^{-1}$.

The experiments use either *constant* action feed-forward policies for each action component, $a_i(t) = A$, with $M = 1$ and parameter $p = A$, or *exponentially* varying feed-forward policies for each action component, $a_i(t) = A + B \cdot e^{-kt}$, with $M = 2$ and parameters $P = (A, B)$. In the latter case, $k = 0.3s^{-1}$ has been used. Thus, the total DOF of the feed-forward policy π_p according to (2) is always $D = N \cdot M$.

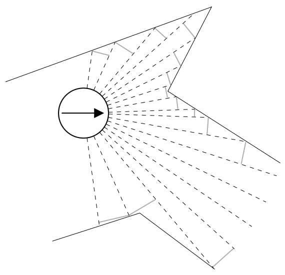

Fig. 2. Obstacle sensor model, comprised of 21 virtual view sectors

In all experiments, *training episodes* have been performed according to section 2.1 and *test episodes* according to section 2.2. The latter ones only serve for monitoring the training progress. After each group of 90 training and 10 test episodes, the return estimator is newly trained from the full training set (see 4.1). Training episodes are terminated after 50s of simulation, test episodes after 500s, unless a collision has occurred earlier.

5.1 "Track" Example

The first examined environment is roughly similar to a race track and shown in Figure 3. The track width is about 125mm. The grey line in Figure 3 depicts the first "good" (i.e. collision-free) test episode for the most primitive $N = M = 1$ case, which occurred after no more than 90 training episodes and involved only 528 training samples according to (5), with only 70% of them actually used as the training set. Please note that the inferred controller is mostly oscillation-free, which is extraordinary after so few trials, and that the plotted trajectory

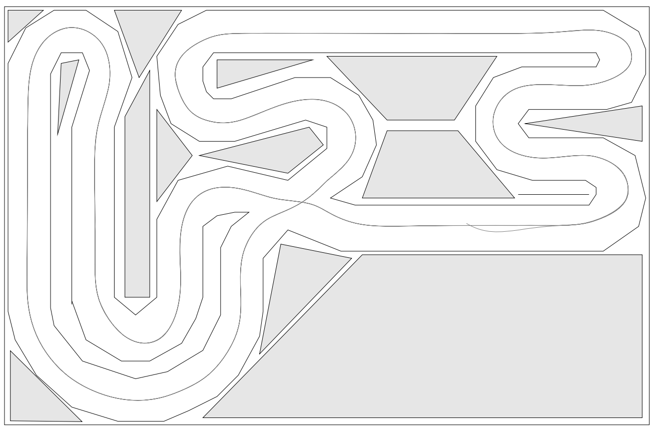

Fig. 3. Track example with $N = 1$, $M = 1$. Collision-free navigation without oscillation after only 90 training episodes.

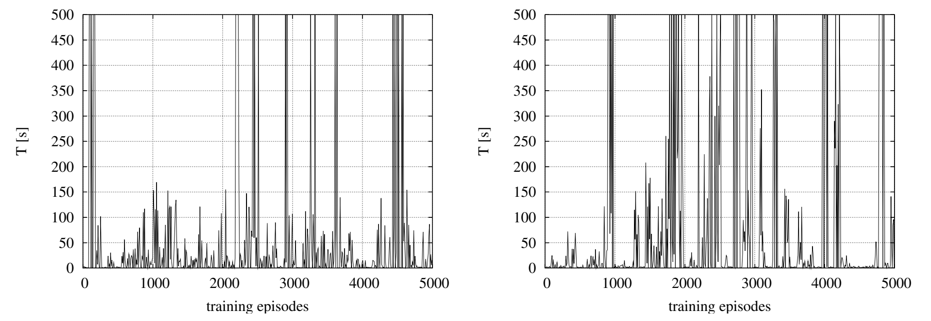

Fig. 4. Learning progress for track example, $N = M = 1$ (left) and $N = 1, M = 2$ (right)

actually comprises 3 "laps" around the track with very high repeat accuracy. With exponential FF policy ($N = 1, M = 2$), the first successful round-track test requires 900 training episodes; with additional v-control ($N = M = 2$) this worsens to 3240 training episodes. Thus, adding more DOF tends to increase learning time, probably because the additional DOF are redundant in this case.

Figure 4 shows the test episode durations over the number of training episodes for the two $N = 1$ cases. Clearly, FF learning does not show a smooth, exponential-decay-type convergence process. This is mainly due to the potentially non-continuous nature of the *arg max* operator in (6), which is very sensitive to "noise" in the estimator training . On the other hand, this coincides with an extremely steep initial learning curve. In practice, one should appoint some additional global criterion for selecting the "desired" agent (e.g. the fraction of "good" test episodes in this case).

5.2 "Corridor" Example

This experiment addresses the learning of non-scalar actions ($N = 2$). The corridor shown in Figure 5 strictly requires reduced speed for turning. With exponential FF policy ($M = 2$), a close-to-optimal behavior occurs for the first time after 1620 training episodes. With constant-action FF policy ($M = 1$), the utility of reduced-speed turns cannot be predicted by the learning algorithm, and FF learning does not effectively succeed before 54810 episodes of training – a success that can well be labeled random. Thus, FF learning does benefit from increased DOF in the FF-policy if the task is not suitably addressed otherwise.

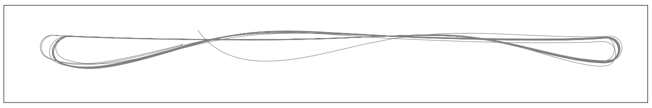

Fig. 5. Corridor example with $N = 2$, $M = 2$. Collision-free test run evolved after 1620 training episodes; turning speed is below 14.4 mm/s, average speed is 38.6 mm/s.

5.3 Results Summary

Table 1 summarizes the learning results for all examined combinations of environment and DOF. For all examples with velocity control, the average speed is shown, indicating that the reward for fast traveling is well honored in all cases.

Table 1. Summary of learning results

World	N	M	**episodes to 1st good**	**best average speed [mm/s]**	**episodes to best speed**
Track	1	1	90	–	–
Track	1	2	900	–	–
Track	2	2	3240	44.0	4230
Corridor	2	1	40410	41.5	54810
Corridor	2	2	1620	40.2	4050

6 Conclusion

FF learning has been shown capable of solving near-real-world control learning problems with very low learning cost. It may even to suitable for tasks with no environment simulation available. Its main drawback is the convolution between the FF policies chosen in training and the characteristics of the environment and task. However, Theorem 1 provides a suitable guideline for selecting appropriate FF policies.

"Fast" biological control learning is sometimes based on an RL situation as well, i.e. solely on exploration and some kind of reward. Thus, FF learning can be considered to close the performance gap between biological and artificial RL to some extent, although there is no methodological foundation from biology involved.

References

1. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA (1998)
2. Sutton, R.S.: Learning to predict by the methods of temporal differences. Machine Learning **3** (1988) 9–44
3. Watkins, C.J.C.H.: Learning from Delayed Rewards. PhD thesis, King's College, University of Cambridge, Cambridge, UK (1989)
4. Bertsekas, D.P.: Dynamic Programming and Optimal Control. second edn. Volume two. Athena Scientific, Belmont, MA (2001)
5. Boyan, J.A.: Least-squares temporal difference learning. In: Proc. 16th International Conf. on Machine Learning, San Francisco, CA, Morgan Kaufmann (1999) 49–56
6. Nissen, S., Nemerson, E.: Fast Artificial Neural Network Library, Version 1.2.0 Reference Manual. http://fann.sourceforge.net (2004)
7. Hertz, J., Krogh, A., Palmer, R.G.: An Introduction to the Theory of Neural Computation. Lecture Notes Volume I. Addison Wesley (1991)