Feedforward Neural Networks in Reinforcement Learning Applied to High-dimensional Motor Control

Rémi Coulom

Laboratoire Leibniz-IMAG, Grenoble, France

Abstract. Local linear function approximators are often preferred to feedforward neural networks to estimate value functions in reinforcement learning. Still, motor tasks usually solved by this kind of methods have a low-dimensional state space. This article demonstrates that feedforward neural networks can be applied successfully to high-dimensional problems. The main difficulties of using backpropagation networks in reinforcement learning are reviewed, and a simple method to perform gradient descent efficiently is proposed. It was tested successfully on an original task of learning to swim by a complex simulated articulated robot, with 4 control variables and 12 independent state variables.

1 Introduction

Reinforcement learning [20, 8] has been successfully applied to numerous motor control problems, either simulated or real [1, 6, 11, 19]. These results are rather spectacular, but successes are restricted to the control of mechanical systems with few degrees of freedom, such as the cart-pole task or the acrobot. Schaal *et. al.* [15, 16] successfully taught robots with many degrees of freedom to perform motor tasks, but the learning consisted only in estimating a model of state dynamics, and no value function was estimated (a linear-quadratic regulator was used to generate controls). As far as I know, no value function with more than 6 input variables has been successfully approximated in non-trivial dynamic motor-control tasks.

The reasons of this restriction is that linear functions approximators are struck by the curse of dimensionality [3]: the cost of approximating a function is exponential with the dimension of the input space. Although some linear approximation schemes such as tile coding [20] can alleviate the curse of dimensionality when their architectures are carefully designed with well-chosen features, they are still very hard to use in practice when there are many input variables, and when there is no good way to guess the right features.

Another way to approximate value functions in reinforcement learning consists in using feedforward neural networks. Tesauro [21] obtained very spectacular successes with these function approximators in his famous backgammon player. An advantage of feedforward neural networks is that they can handle high-dimensional inputs more easily. Barron proved that, to approximate a function

N. Cesa-Bianchi et al. (Eds.): ALT 2002, LNAI 2533, pp. 403–413, 2002.

in a general class, neural networks with sigmoidal units do not suffer from the curse of dimensionality, whereas parametric linear approximators do [2].

Nevertheless, linear function approximators have been often preferred. The main advantage of linear function approximators is that they can have a better locality, that is to say it is possible to adjust the value in a small area of the input space without interfering with values outside this small area. This prevent the approximation scheme from "unlearning" past experience and allows reinforcement learning algorithms to make an efficient incremental use of learning data.

In this paper, we demonstrate that it is worth trying to overcome the difficulties of feedforward neural networks, because they can solve problems that have a higher dimensionality than those previously tackled with linear function approximators. In the first section, the continuous TD(λ) algorithm that was used to train the neural network is presented. The second section deals with issues that have to be solved to perform gradient descent efficiently with feedforward neural networks. The third section presents experimental results obtained on a swimmer with 12 state variables and 4 control variables.

2 Continuous TD(λ)

The algorithm used in experiments reported in this paper is Doya's [6] continuous TD(λ). It is a continuous version of Sutton's discrete algorithm [18] that is adapted to problems in continuous time and space such as motor-control problems.

2.1 Problem Definition

In general, we will suppose that we are to solve motor problems defined by:

- States $\boldsymbol{x} \in S \subset \mathbb{R}^p$ (p real values define the state of the system).
- Controls $\boldsymbol{u} \in U \subset \mathbb{R}^q$ (the system can be controlled via q real values).
- System dynamics $f : S \times U \mapsto \mathbb{R}^p$. This function maps states and actions to derivatives of the state with respect to time. That is to say $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$.
- A reward function $r : S \times U \mapsto \mathbb{R}$. The problem consists in maximizing the cumulative reward as detailed below.
- A shortness factor $s_\gamma \geq 0$. This factor measures the short-sightedness of the optimization. It can be related to the traditional γ in discrete reinforcement learning by $\gamma = e^{-s_\gamma \delta t}$.

A *strategy* or *policy* is a function $\pi : S \mapsto U$ that maps states to controls. Applying a policy from a starting state $\boldsymbol{x}_0$ at time t_0 produces a trajectory $\boldsymbol{x}(t)$ defined by the ordinary differential equation

$$\forall t \geq t_0 \quad \dot{\boldsymbol{x}} = f\big(\boldsymbol{x}, \pi(\boldsymbol{x})\big) \ ,$$
$$\boldsymbol{x}(t_0) = \boldsymbol{x}_0 \ .$$

The value function of π is defined by

$$V^{\pi}(\boldsymbol{x}_0) = \int_{t=t_0}^{\infty} e^{-s_\gamma (t-t_0)} r\Big(\boldsymbol{x}(t), \pi\big(\boldsymbol{x}(t)\big)\Big) dt \ . \tag{1}$$

The goal is to find a policy that maximizes the total amount of reward over time, whatever the starting state $\boldsymbol{x}_0$. More formally, problem consists in finding π^* so that

$$\forall \boldsymbol{x}_0 \in S \quad V^{\pi^*}(\boldsymbol{x}_0) = \max_{\pi : S \mapsto U} V^{\pi}(\boldsymbol{x}_0) \ .$$

V^{π^*} does not depend on π^* and is denoted V^*.

2.2 Learning Algorithm

We will suppose we are to approximate the optimal value function V^* with a parametric function approximator $V_{\boldsymbol{w}}$, where $\boldsymbol{w}$ is the vector of weights (parameters). The continuous TD(λ) algorithm consists in integrating an ordinary differential equation:

$$\begin{cases} \dot{\boldsymbol{w}} = \eta \mathcal{H} \boldsymbol{e} \ , \\ \dot{\boldsymbol{e}} = -(s_\gamma + s_\lambda)\boldsymbol{e} + \dfrac{\partial V_{\boldsymbol{w}}(\boldsymbol{x})}{\partial \boldsymbol{w}} \ , \\ \dot{\boldsymbol{x}} = f\big(\boldsymbol{x}, \pi(\boldsymbol{x})\big) \ , \end{cases} \tag{2}$$

with

$$\mathcal{H} = r\big(\boldsymbol{x}, \pi(\boldsymbol{x})\big) - s_\gamma V_{\boldsymbol{w}}(\boldsymbol{x}) + \frac{\partial V_{\boldsymbol{w}}}{\partial \boldsymbol{x}} \cdot f\big(\boldsymbol{x}, \pi(\boldsymbol{x})\big) \ .$$

$\mathcal{H}$ is the Hamiltonian and is a continuous equivalent of Bellman's residual. $\mathcal{H} > 0$ indicates a "good surprise" and causes and increase in the past values, whereas $\mathcal{H} < 0$ is a "bad surprise" and causes a decrease in the past values. The magnitude of this change is controlled by the learning rate η, and its time extent in the past is defined by the parameter s_λ. s_λ can be related to the traditional λ parameter in the discrete algorithm by $\lambda = e^{-s_\lambda \delta t}$. $\boldsymbol{e}$ is the vector of eligibility traces. Learning is decomposed into several episodes, each starting from a random initial state, thus insuring exploration of the whole state space. During these episodes, the policy π is chosen to be greedy with respect to the current value estimate $V_{\boldsymbol{w}}$.

3 Efficient Gradient Descent

The basic continuous TD(λ) algorithm presented in the previous section trains the parametric function estimator by applying the steepest-descent method. This kind of method usually does not work efficiently with feedforward neural networks, because it can be struck hard by ill-conditioning. Ill-conditioning means that the output of the value function may be much more sensitive to some weights

than to some others. As a consequence, a global learning coefficient that would be good for one weight is likely to be too high for some and too low for others.

Many numerical methods to deal with ill-conditioning have been proposed in the framework of supervised learning [5]. Most of the classical advanced methods that are able to deal with this difficulty are batch algorithms (scaled conjugate gradient [10], Levenberg Marquardt, RPROP [14], QuickProp [7], ...). TD(λ) is incremental by nature since it handles a continuous infinite flow of learning data, so these batch algorithms are not well adapted.

It is possible, however, to use second order ideas in on-line algorithms [13, 17]. Le Cun *et al.* [9] recommend a "stochastic diagonal Levenberg Marquardt" method for supervised classification tasks that have a large and redundant training set. TD(λ) is not very far from this situation, but using this method is not easy because of the special nature of the gradient descent used in the TD(λ) algorithm. Evaluating the diagonal terms of the Hessian matrix would mean differentiating with respect to each weight the total error gradient over one trial, which is equal to

$$\int_{t_0}^{t_f} -\mathcal{H}(t)\boldsymbol{e}(t)dt \ .$$

This is not impossible, but still a bit complicated. In particular, $\mathcal{H}$ depends on $\boldsymbol{w}$ in a very complex way, since it depends on the gradient of the output of the network with respect to its input.

Another method, the Vario-η algorithm [12], was used instead. It is well adapted for the continuous TD(λ) algorithm, and it is both virtually costless in terms of CPU time and extremely easy to implement.

3.1 Principle

Figure 1 shows the typical variations of two weights during learning. In this figure, the basic algorithm (derived from (2)) was applied to a simple control problem and no special care was taken to deal with ill-conditioning. Obviously, the error function was much more sensitive to w_1 than to w_2. w_2 varied very slowly, whereas w_1 converged rapidly. This phenomenon is typical of ill-conditioning.

Another effect of these different sensitivities is that w_1 looks much more noisy than w_2. The key idea of Vario-η consists in measuring this noise to estimate the sensitivity of the error with respect to each weight, and scale individual learning rates appropriately. That is to say, instead of measuring ill-conditioning of the Hessian matrix, which is the traditional approach of efficient gradient-descent algorithms, ill-conditioning is measured on the covariance matrix.

3.2 Algorithm

In theory, it would be possible to obtain a perfect conditioning by performing a principal component analysis with the covariance matrix. This approach is not

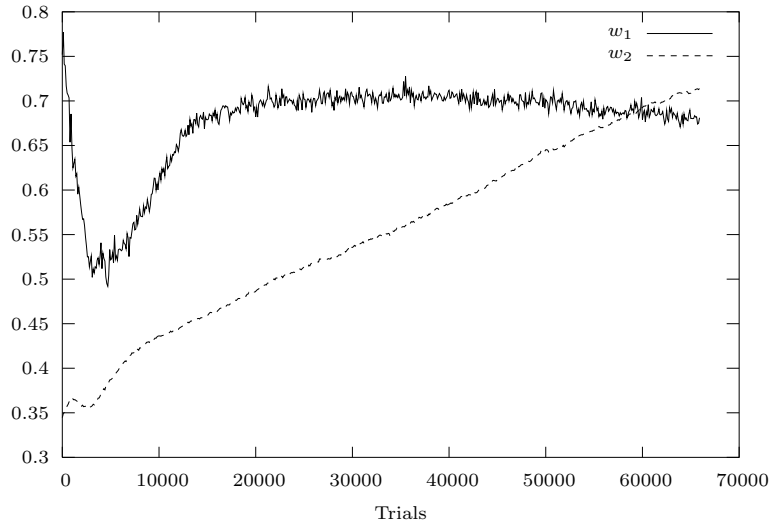

Fig. 1. Variations of two weights when applying the basic TD(λ). A value is plotted every 100 trials. The function approximator used is a 66-weight feedforward neural network.

practical because of its computational cost, so a simple analysis of the diagonal is performed:

$$v_i(k+1) = (1-\beta)v_i(k) + \beta\left(\frac{w_i(k+1) - w_i(k)}{\eta_i(k)}\right)^2 ,$$

$$\eta_i(k) = \frac{\eta}{\sqrt{v_i(k)} + \varepsilon} .$$

k is the trial number. $v_i(0)$ is a large enough value. β is the variance decay coefficient. A typical choice is 1/100. ε is a small constant to prevent division by zero. $\eta_i(k)$ is the learning rate for weight w_i. This formula assumes that the standard deviation of the gradient is large in comparison to its mean, which was shown to be true empirically in experiments.

3.3 Results

Experiments were run with fully-connected cascade feedforward networks, with a linear output unit, and sigmoidal hidden units. Observations during reinforcement learning indicated that the variances of weights on connections to the linear output unit were usually n times larger than those on internal connections, n being the total number of neurons. The variances of internal connections are all of the same order of magnitude (we have yet to find a good theoretical explanation for this). This means that good conditioning can be simply obtained by scaling the learning rate of the output unit by $1/\sqrt{n}$. This allows to use a global learning rate that is $\sqrt{n}$ times larger and provides a speed-up of about $\sqrt{n}$. The network used in experiments had 60 neurons, so this was a very significant acceleration.

3.4 Comparison with Second-Order Methods

An advantage of this method is that it does not rely on the assumption that the error surface can be approximated by a positive quadratic form. In particular, dealing with negative curvatures is a problem for many second-order methods. There is no such problem when measuring the variance.

It is worth noting, however, that the Gauss-Newton approximation of the Hessian suggested by Le Cun *et al.* is always positive. Besides, this approximation is formally very close to the variance of the gradient (I thank Yann Le Cun for pointing this out to me): the Gauss-Newton approximation of the second order derivative of the error with respect to one weight is

$$\frac{\partial^2 E}{\partial w_{ij}^2} \approx \frac{\partial^2 E}{\partial a_i^2} y_j^2 \ .$$

This is very close to

$$\Big(\frac{\partial E}{\partial w_{ij}}\Big)^2 = \Big(\frac{\partial E}{\partial a_i}\Big)^2 y_j^2 \ .$$

A major difference between the two approaches is that there is a risk that the variance goes to zero as weights approach their optimal value, whereas the estimate of the second order derivative of the error would stay positive. That is to say, the learning rate increases as the gradient of the error decreases, which may be a cause of instability, especially if the error becomes zero. This was not a problem at all in the reinforcement learning experiments that were run, because the variance actually *increased* during learning, and was never close to zero.

4 Experiments with Swimmers

Experiments where run with a mechanical system made of 5 articulated segments, controlled with 4 torques, and moving in a two-dimensional fluid. This system has 12 state variables and 4 control variables. Full details about the simulation can be found in Appendix A.

A 60-neuron feedforward neural network was used, with a fully connected cascade architecture (that is to say, neuron number i was connected to all neurons $j > i$). Inputs were the 5 angles of the 5 segments, their 5 derivatives with respect to time, and the velocity of the center of mass (so, 12 independent variables). The learning rate was $\eta = 0.01$ and it was scaled down in the output unit as explained in Section 3.3. The learning algorithm was integrated with the Euler method, using a time step of 0.01 seconds.

Figure 2 shows how learning progressed. It took about 2.5 million trials of 5 seconds to obtain top performance. This simulation took about one week of CPU time on a 800 MHz PC. The fact that it took such a long time to learn is one of the limitations of this methods: it requires a lot of training data.

After some more training time, performance collapses. The reasons of this drop in performance are not very well understood yet, and some investigations remain to be done. This lack of stability is another limitation of this method.

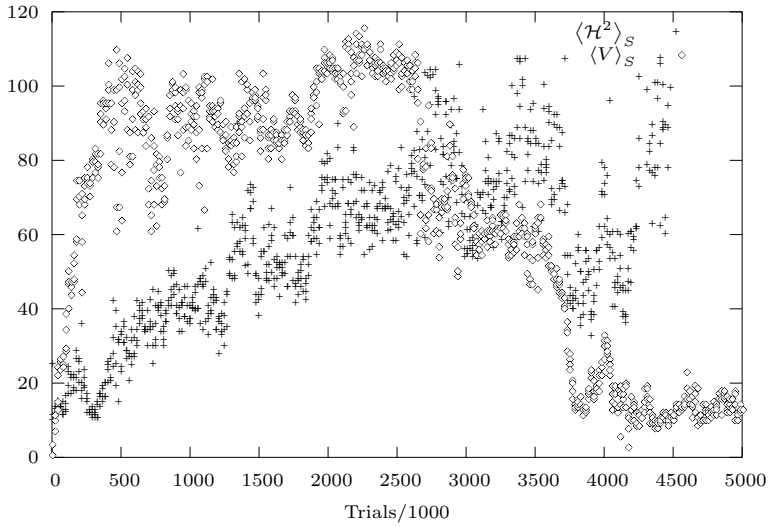

Fig. 2. Progress of learning. $\langle \mathcal{H}^2 \rangle_S$ is the mean squared Hamiltonian estimated over the state space. $\langle V \rangle_S$ is the mean value over the state space.

The swimming technique obtained after 2.5 million trials is presented on Fig. 3. Visualizing the swimmer movements on this figure might not be very easy. Animations can be downloaded from the author's web page (`http://remi.coulom.free.fr/Thesis/`), and can give a better feeling of how the swimmer moves.

5 Conclusion

In this paper, a method for the efficient use of feedforward neural networks has been presented, and tested successfully on a difficult motor-control task. This result indicates that continuous model-based reinforcement learning with feedforward neural networks can handle motor control problems that are significantly more complex than those that have been solved with linear function approximators so far.

The main limits of this method are that it requires a lot of training data, and convergence is not guaranteed. Refining the learning algorithm to deal with these problems seems to be an interesting direction for further research. Weaver *et al's* [22] method to make feedforward neural networks local might be a possibility, but its efficiency on real problems remains to be established. Another interesting possibility to make a more efficient use of learning data consists in "recycling trajectories" [4], that is to say reusing past trajectories several times.

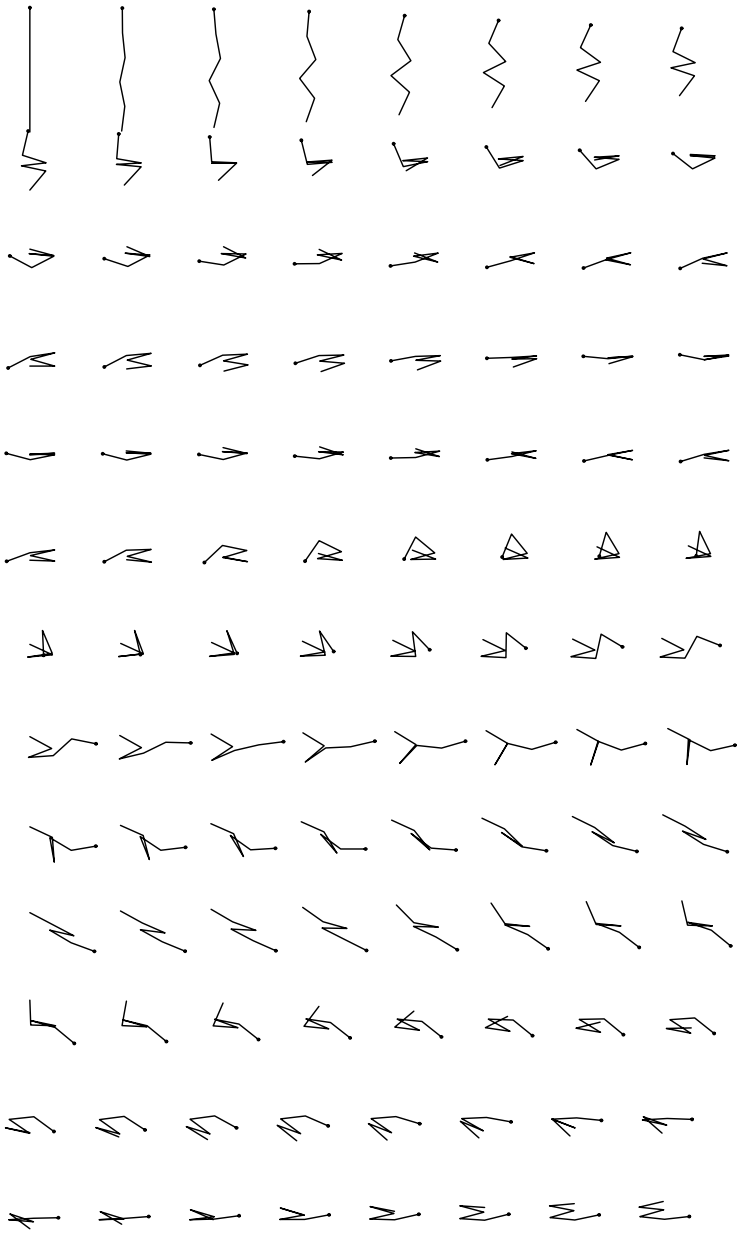

Fig. 3. A 5-segment swimmer trained with a 60-neuron network. In the first 5 lines of this animation, the target direction is to the right. In the last 8, it is reversed to the left. Swimmers are plotted every 0.1 seconds, reading from left to right and from top to bottom.

References

1. Charles W. Anderson. Strategy learning with multilayer connectionist representations. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 103–114, Irvine, CA, 1987. Morgan Kaufmann.
2. Andrew R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945, May 1993.
3. Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, New Jersey, 1957.
4. Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.
5. Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.
6. Kenji Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12:243–269, 2000.
7. Scott E. Fahlman. An empirical study of learning speed in back-propagation networks. Technical Report CMU-CS-88-162, Carnegie-Mellon University, 1988.
8. Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.
9. Yann Le Cun, Leon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient BackProp. In Genevieve B. Orr and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade*. Springer, 1998.
10. Martin F. Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6:525–533, 1993.
11. Jun Morimoto and Kenji Doya. Hierarchical reinforcement learning of low-dimensional subgoals and high-dimensional trajectories. In *Proceedings of the Fifth International Conference on Neural Information Processing*, pages 850–853, 1998.
12. Ralph Neuneier and Hans-Georg Zimmermann. How to train neural networks. In Genevieve B. Orr and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade*. Springer, 1998.
13. Genevieve B. Orr and Todd K. Leen. Using curvature information for fast stochastic search. In *Advances in Neural Information Processing Systems 9*. MIT Press, 1997.
14. Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proceedings of the IEEE International Conference on Neural Networks*, 1993.
15. Stefan Schaal and Christopher G. Atkeson. Robot juggling: An implementation of memory-based learning. *Control Systems Magazine*, 14:57–71, 1994.
16. Stefan Schaal, Christopher G. Atkeson, and Sethu Vijayakumar. Real-time robot learning with locally weighted statistical learning. In *International Conference on Robotics and Automation (ICRA2000)*, 2000.
17. Nicol N. Schraudolph. Local gain adaptation in stochastic gradient descent. In *Proceedings of the 9th International Conference on Artificial Neural Networks*, London, 1999. IEE.
18. Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.
19. Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8*, pages 1038–1044. MIT Press, 1996.

20. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.
21. Gerald Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, March 1995.
22. Scott E. Weaver, Leemon C. Baird, and Marios M. Polycarpou. Preventing unlearning during on-line training of feedforward networks. In *Proceedings of the International Symposium of Intelligent Control*, 1998.

A Swimmer Model

This section gives a description of the Swimmer model. The C++ source code of the simulator can be obtained from the author on request.

A.1 Variables and Parameters

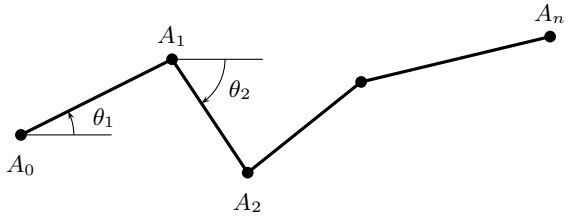

Fig. 4. Swimmer model

State variables:

- A_0: position of first point
- θ_i: angle of part i with respect to the x axis
- $\dot{A}_0, \dot{\theta}_i$: their derivatives with respect to time

Control variables:

- $(u_i)_{i\in\{1\ldots n-1\}}$, torques applied between body parts, constrained by $|u_i| < U_i$

Problem parameters:

- n: number of body parts
- m_i: mass of part i ($i \in \{1...n\}$)
- l_i: length of part i ($i \in \{1...n\}$)
- k: viscous-friction coefficient

A.2 Model of Viscous Friction

The model of viscous friction used consists in supposing that a small line of length $d\lambda$ with a normal vector $\boldsymbol{n}$ and moving at velocity $\boldsymbol{v}$ receives a small force of value $d\boldsymbol{F} = -k(\boldsymbol{v} \cdot \boldsymbol{n})\boldsymbol{n}d\lambda$. The total force applied to part i is equal to $\boldsymbol{F}_i = -kl_i(\dot{G}_i \cdot \boldsymbol{n}_i)\boldsymbol{n}_i$, with $G_i = (A_{i-1} + A_i)/2$. The Total moment at G_i is $\mathcal{M}_i = -k\dot{\theta}_i l_i^3/12$.

A.3 System Dynamics

Let $\boldsymbol{f}_i$ be the force applied by part $i+1$ to part i.

$$\forall i \in \{1, \ldots, n\} \quad -\boldsymbol{f}_{i+1} + \boldsymbol{f}_i + \boldsymbol{F}_i = m_i \ddot{G}_i$$

These equations allow to express the $\boldsymbol{f}_i$'s as a function of state variables:

$$\boldsymbol{f}_0 = \boldsymbol{0}$$
$$\forall i \in \{1, \ldots, n\} \quad \boldsymbol{f}_i = \boldsymbol{f}_{i-1} - \boldsymbol{F}_i + m_i \ddot{G}_i$$

We end up with a set of $n+2$ linear equations with $n+2$ unknowns:

$$\begin{cases} \boldsymbol{f}_n = \boldsymbol{0}, \\ m_i \dfrac{l_i}{12} \ddot{\theta}_i = \det(\overrightarrow{G_i A_i}, \boldsymbol{f}_i + \boldsymbol{f}_{i-1}) + \mathcal{M}_i - u_i + u_{i-1}. \end{cases}$$

Unknowns are $\ddot{G}_0$ (or any $\ddot{G}_i$) and all the $\ddot{\theta}_i$'s. Solving this set of equations gives state dynamics.

A.4 Reward

$r(\boldsymbol{x}, \boldsymbol{u}) = \dot{G}_x$ (G_x is the abscissa of the center of mass)

A.5 Numerical Values (SI Units)

$m_i = 1$, $l_i = 1$, $k = 10$, $U_i = 5$