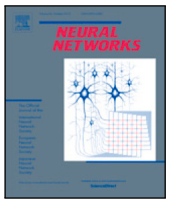
Neural Networks 184 (2025) 107034

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet



Full Length Article

Neurocontrol for fixed-length trajectories in environments with soft barriers

Michael Fairbank [a,*], Danil Prokhorov [b], David Barragan-Alcantar [a], Spyridon Samothrakis [a], Shuhui Li [c]

[a] School of Computer Science and Electronic Engineering, University of Essex, Colchester, UK
[b] Toyota Research Institute NA, Ann Arbor, MI, US
[c] Department of Electrical and Computer Engineering, University of Alabama, Tuscaloosa, 35401, AL, US

ARTICLE INFO

Keywords:
Neurocontrol
Adaptive dynamic programming
Reinforcement learning
Exploding gradients
Soft barriers
Back-propagation through time
Analytic policy gradient

ABSTRACT

In this paper we present three neurocontrol problems where the analytic policy gradient via back-propagation through time is used to train a simulated agent to maximise a polynomial reward function in a simulated environment. If the environment includes terminal barriers (e.g. solid walls) which terminate the episode whenever the agent touches them, then we show learning can get stuck in oscillating limit cycles, or local minima. Hence we propose to use fixed-length trajectories, and change these barriers into soft barriers, which the agent may pass through, while incurring a significant penalty cost. We demonstrate that the presence of soft barriers can have the drawback of causing exploding learning gradients. Furthermore, the strongest learning gradients often appear at inappropriate parts of the trajectory, where control of the system has already been lost. When combined with modern adaptive optimisers, this combination of exploding gradients and inappropriate learning often causes learning to grind to a halt. We propose ways to avoid these difficulties; either by careful gradient clipping, or by smoothly truncating the gradients of the soft barriers' polynomial cost functions. We argue that this enables the learning algorithm to avoid exploding gradients, and also to concentrate on the most important parts of the trajectory, as opposed to parts of the trajectory where control has already been irreversibly lost.

1. Introduction

In neurocontrol, the aim is to train a neural network to control an agent such that it maximises a total reward function in a given environment (Fairbank, Prokhorov, & Alonso, 2014; Kremer & Kolen, 2001; Prokhorov, Santiago, & Wunsch, 1995; Sarangapani, 2018; Werbos, 2018). In this sense, neurocontrol is an umbrella term which includes specific forms of Reinforcement Learning (RL) and Adaptive Dynamic Programming (ADP) (Prokhorov & Wunsch, 1997; Sutton & Barto, 1998; Wang, Zhang, & Liu, 2009). Compared to RL, which considers the environment to be a black box, in neurocontrol we may assume the environment is known and differentiable, which allows us to learn by considering simulated trajectories of the agent; and to perform gradient ascent on the total trajectory reward (i.e. to perform policy-gradient learning (PGL) analytically, for example by using backpropagation through time (BPTT) to calculate that gradient (Werbos, 2018)). In the discrete-time neurocontrol problem, at each time step t, the agent has state vector $\vec{x}_t \in \mathbb{S}$, and uses a neural network (the "action network" or "policy") function A to choose an action $\vec{u}_t = A(\vec{x}_t, \vec{w}) \in \mathbb{A}$ to take at that time step. At each time step the agent receives a reward, or step utility, U_t. Hence the total discounted reward, accumulated along the entire trajectory, is given by

$$R = \sum_{t=0}^{T_F-1} \gamma^t U_t, \tag{1}$$

where $\gamma \in (0, 1]$ is a discount factor, and T_F is the final time step of the trajectory.

In a sense, neurocontrol by PGL is the simplest and most elegant way to address the neurocontrol problem: it is simply gradient ascent on R with respect to the weight vector of the neural network, $\vec{w}$. Compared to other ADP/RL methods, which might involve multiple neural networks that interact with each other during training (e.g. an actor and a critic, each with their own function-approximation limitations, potentially causing divergence (Fairbank & Alonso, 2012a)), PGL has relatively strong convergence guarantees that come associated with gradient ascent methods; assuming smoothness of the policy-gradient function (Fairbank, Alonso, & Prokhorov, 2013). However in practice, we find that PGL often gets stuck. This is because the policy-gradient function is usually not smooth (e.g. see the Oscillating

* Corresponding author.
E-mail address: m.fairbank@essex.ac.uk (M. Fairbank).

https://doi.org/10.1016/j.neunet.2024.107034
Received 4 April 2024; Received in revised form 2 September 2024; Accepted 6 December 2024
Available online 12 December 2024

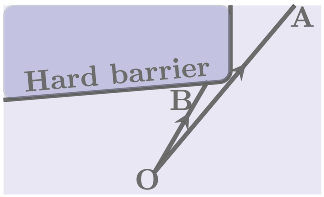


Fig. 1. The "Oscillating Corner-Avoidance Problem" in a concave environment. Here the agent has a fixed start point O, and plans to find the shortest straight-line path which moves north past the barrier. Analytic PGL methods will often get stuck in a limit cycle here, with the trajectory alternating between the path OA and the path OB. In this common situation, the policy gradient is not smooth; and changes discontinuously as the trajectory shifts from OA to OB.

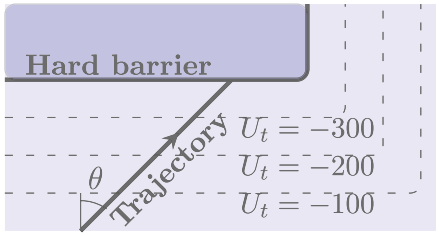


Fig. 2. Example of "Deliberate Early Suicide" where the agent is incentivised to truncate the accumulated negative trajectory reward, by deliberately crashing as early as possible. Here, to address the Oscillating Corner-Avoidance Problem shown in Fig. 1, we have introduced penalty terms (shown as dotted lines here) to discourage the agent from getting close to the barrier. However in this example, reducing the action chosen, θ, would mean that the trajectory traverses less distance through the dotted contour lines of negative utility, and hence that $\partial R/\partial\theta$ is negative, indicating that the agent wants to steer left, i.e. to deliberately crash into the barrier sooner.

Corner-Avoidance Problem in Fig. 1), it might have many local maxima, or because there are exploding or vanishing gradients occurring somewhere. We look at these cases in this paper, expand on them, and propose solutions.

One approach to address the oscillating corner-avoidance problem might be to introduce penalty terms for just getting *close* to the barrier, but this can lead to unexpected consequences, such as a kind of local maximum which we call "Deliberate Early Suicide" (see Fig. 2). Other solutions might be to add stochastic terms to the environment or agent's behaviour, so that there is always a certain probability that the agent clips the barrier, and thus the learning algorithm never forgets about the barrier. In this paper we explore another solution, where, while in the simulation stage of learning, we change the barriers into "soft barriers",[1] which the simulated agent can move straight through at the expense of a penalty, and we make the trajectories have fixed length (thus preventing early suicide).

Hence in this paper, we only consider situations where the trajectory duration T_F (in (1)) is constant; regardless of how far the agent may penetrate through a soft barrier. Ultimately though, the agent must still learn to avoid going through soft barriers, because after training, the agent could potentially be acting in the real world, where crashing into barriers is costly. Hence during training, in the simulated environment, there must be significant penalty costs (i.e. significantly negative rewards U_t) associated with going through soft barriers; and the training process must learn to avoid these penalties. Such a polynomial penalty function might be given by $U_t = -C(x_t^{\text{penetration}})$, where $x_t^{\text{penetration}} > 0$ is the perpendicular distance which the agent has penetrated the barrier at time t, where the barrier is positioned at $x^{\text{penetration}} = 0$, and $C(x)$ is

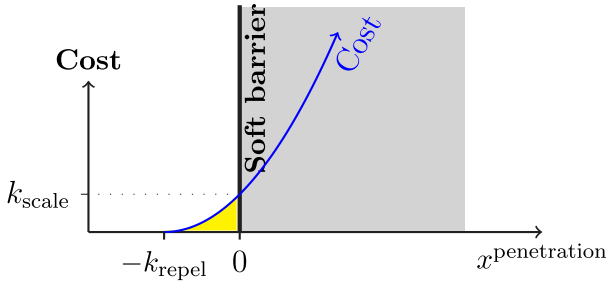


Fig. 3. An illustration of the simple polynomial cost function (2) forming a soft barrier. For this function, the cost starts to rise in the yellow region, before the agent even penetrates the barrier. This aims to *repel* the agent before hitting the barrier. In the region $x^{\text{penetration}} < -k_{\text{repel}}$, the cost function is flat and zero (as enforced by the max function in (2)); giving the agent total freedom to navigate as it sees best in this flat region of cost. The blue curve penetrates the intended soft barrier, indicating that the cost is not infinite. But this curve can be made as steep as required by increasing the constants k_{scale} or k_{pow}.

a positive cost function defined by

$$C(x) = k_{\text{scale}} \left(\max \left[\frac{x}{k_{\text{repel}}} + 1, 0 \right] \right)^{k_{\text{pow}}}. \tag{2}$$

Here $k_{\text{pow}} \geq 1$ is the power of the polynomial penalty, and k_{repel} is an argument specifying how wide we want the repulsion zone of this barrier to be, and $k_{\text{scale}} > 0$ is vertical stretch factor. See Fig. 3 for a visualisation.

A drawback of using soft barriers, and fixed trajectory durations T_F, is that the agent can travel arbitrarily far through the soft boundaries imposed by (2), potentially generating hugely negative R values. This is especially likely at the start of learning when the neural network's weights have just been randomised, and trajectories are therefore randomised paths.[2] On consideration of (2) with $k_{\text{pow}} > 1$, we know that the gradients $\partial C/\partial x \to \infty$ when the penetration distance x is large. We find that these large gradients cause two severe problems to this approach:

1. They cause exploding policy gradients, which prevents learning.
2. They cause the largest gradients to appear at a point in the trajectory where the agent has potentially already lost control (i.e. at the point where $x^{\text{penetration}}$ is largest), and where remedial actions are impossible. This is illustrated most clearly later in the paper, in Fig. 5.

It might be expected that high gradients such as those described above produce overly-fast learning, and the associated problems that come with that (e.g. non-monotonic progress of improvement in R). To the contrary, with modern acceleration algorithms (e.g. Adam and RMSProp), we generally see extremely slow learning occurring, with a puzzling effect of trajectories appearing to freeze up. An explanation for this is that Adam and RMSProp both use an adaptive learning rate whose denominator is proportional to the exponential moving average of the squared gradient components. Therefore, when large gradients are encountered, both algorithms develop a huge denominator which slows down learning to a halt.

Furthermore, when ordinary gradient descent is used (with a constant learning rate), then huge gradients mean that divergence can occur resulting in infinities or NaN errors. Hence for ordinary gradient descent, it is necessary to use a tiny learning rate. So again, learning ceases up.

In this paper we propose two methods to address the above two points:

[1] This usually requires a software simulation of the environment to be available.

[2] Remember, with only soft barriers present, the trajectories are often completely unconstrained to meander anywhere.

1. We can use gradient clipping inside of the backwards propagation of value gradients ($\partial R/\partial \vec{x}$) within the BPTT algorithm. (Note that this is different from ordinary gradient clipping, which acts on $\partial R/\partial \vec{w}$).
2. Alternatively, we propose to smoothly truncate the gradients of the steep-sided cost functions appearing in (2), using what we call a "tanh wrapper", so that they never exhibit infinite gradients in the limit as $x^{\text{penetration}} \to \infty$.

Seeing as the choice to use soft barriers and fixed trajectory lengths raises the problems described above, it might seem better instead to use hard barriers, such that the trajectory terminates instantly when a barrier is breached. However terminating the trajectory early like this brings its own problems:

1. If the hard barrier is breached part-way through a time-step, then it can distort the direction of the learning gradient, and a careful correction involving clipping of the final time step needs making, as described by Fairbank, Prokhorov, and Alonso (2014).
2. With hard barriers, the T_F appearing in (1) depends upon the policy, i.e. upon $\vec{w}$, but many learning algorithms which aim to do analytic gradient ascent on R do not take this dependency into account, especially when T_F is a discrete quantity. Hence they either are not performing true gradient ascent on R; or the surface of R with respect to $\vec{w}$ is not smooth, i.e. not suited to gradient ascent, resulting in the problems exemplified in Figs. 1 & 2.

The central aim of this paper is to make analytic PGL via BPTT more robust, and applicable to a wider range of environments. The contributions of this paper are as follows:

1. We explain how Oscillating Corner-Avoidance and Deliberate Early Suicide can adversely affect neurocontrol in hard-barrier environments.
2. We define useful soft-barrier functions, introduced to address the above two problems. The soft-barrier functions proposed allow the agent to explore in a flat zero-cost region, but provide steep differentiable walls.
3. We explain a common cause of why BPTT algorithms can seem to stall with their learning, when soft-barrier, fixed-length trajectories are used; our answer is that exploding value gradients ($dR/d\vec{x}$) occur at *inappropriate* time steps of the trajectory.
4. We propose solutions to this problem by clipping the back-propagated gradient or by smoothly truncating the gradients of soft-barrier cost functions, with what we call a "tanh wrapper".

In the rest of this paper, in Section 2 we list related work. In Section 3 we give the main contributions of this paper, which include methods to smoothly truncate learning gradients to enable more efficient learning in neurocontrol. In Section 4, we define a useful flat-bottomed soft barrier function, and apply it in three experiments that all show the benefit of using the methods proposed in this paper to smoothly truncate gradients. These include a car-driver problem (Section 4.1), which becomes solvable only when the tanh-wrapper or clipped gradients are used; the classic pole-balancing problem (Section 4.2), where we show learning becomes much more robust when the tanh wrapper method is used; and an electrical grid-connected converter tracking problem (in Section 4.3), which becomes solvable for quadratic and quartic cost functions using the tanh-wrapper or clipped gradients, which was not previously possible. Finally, in Section 5, we give conclusions.

2. Related work

The use of gradient ascent to train a neural network to perform neurocontrol, where the gradient is found analytically (i.e. using automatic differentiation) dates back to Werbos (1990, 2018). Analytic gradient-based learning methods for optimising trajectories are still an active research area, (e.g. de Avila Belbute-Peres, Smith, Allen, Tenenbaum, & Kolter, 2018; Toussaint, Allen, Smith, & Tenenbaum, 2018); and physics engines which provide differentiable environments for the sole purposes of ADP/RL research are being actively developed (de Avila Belbute-Peres et al., 2018; Freeman et al., 2021; Hu et al., 2019).

Gradient ascent on the total trajectory reward R is known as Analytic Policy Gradients by Wiedemann et al. (2023) and Freeman et al. (2021), which in this paper we refer to as BPTT. Despite gradient ascent offering convergence guarantees under smoothness assumptions and using appropriate learning rates, it is noted by Freeman et al. (2021) that gradient-based learning methods such as BPTT can frequently get stuck in local minima, and/or suffer from vanishing or exploding gradients, compared to model-free RL methods. In this paper we attempt identify and rectify a class of those obstacles.

In our work, we propose smoothly truncating the gradients of the utility function. The benefits of this are to prevent exploding gradients, and to make the steepest gradients appear at appropriate locations of the trajectory. This is related to some of the ideas used by Mnih et al. (2015) when they modified the reward-signal received when learning to play Atari games. In their case, after any game action was taken that led to a change of game score, regardless of the magnitude of the game-score change, their RL system received a truncated reward of $+1$ if the game score went up, and -1 if the game score went down. A stated motivation for doing this was so that they could use the same learning rates over a wide range of Atari games. But a secondary benefit of doing this would have been to ensure that the learning gradient magnitudes were never too large or too small, and thus less likely to explode.

Another trick used by Mnih et al. (2015) was that they truncated the temporal-difference error used in their Q-network's weight update, if it exceeded a certain amount. This is also reminiscent of the clipped value-gradients we propose in this paper. In both cases though, the methods used for the Atari-learning work was for discrete state-spaces and scalar-based RL algorithms; whereas the work in this paper is applicable to continuous-valued state-spaces and gradient-based neurocontrol learning algorithms

Barrier functions are already known about and used in optimisation theory. They can be added to the optimisation objective function to act as a penalty term which tends to infinity when the solution enters into a forbidden region. Similarly, in control theory, control barrier functions can be used to assure that a dynamical system will never enter into an undesirable region (Ames et al., 2019).

Exploding gradients have been studied in neural-network training in a non-control setting, for deep and recurrent neural networks (Bengio, Simard, & Frasconi, 1994). Resolutions have included gradient clipping (Pascanu, Mikolov, & Bengio, 2013), careful weight initialisations (Glorot & Bengio, 2010; Yilmaz & Poli, 2022), target-space methods (Fairbank, Samothrakis, & Citi, 2022; Rohwer, 1990), and memory gates (Hochreiter & Schmidhuber, 1997).

In this paper we use the term "gradient clipping" to specify that learning gradients over a certain magnitude are reduced. In previous work we discussed "Clipping in Neurocontrol" (Fairbank, Prokhorov, & Alonso, 2014) which has a different meaning, namely referring to truncating the final time step of a trajectory into a fractional quantity; so that it exactly reflects the fraction of the final time step in which a hard barrier was breached. It turns out that it is necessary to performing this fractional calculation carefully, as omitting it can radically distort the learning gradients, and prevent the neurocontrol objective from being achieved. A related "time-of-impact" consideration was identified and solved by Hu et al. (2019) that affects the differentiability of physics models during collisions against barriers.

Another way to truncate exploding gradients in BPTT for control is to only backpropagate the value gradients for a fixed number (h) of trajectory time steps, as in the algorithm BPTT(h). This is related to the gradient clipping method we use. BPTT(h) was shown to be particularly

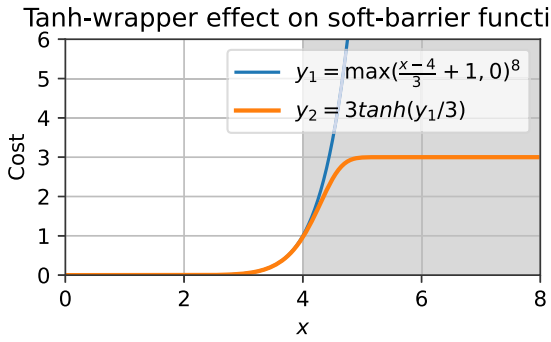


Fig. 4. Effect of tahh wrapper with $k_{\max} = 3$ on the soft barrier function defined in (2) with $y = C(x-4)$, and $k_{\text{repel}} = 3$, $k_{\text{pow}} = 8$ and $k_{\text{scale}} = 1$. In this case, the argument $x-4$ means barrier is defined to be at $x \geq 4$ (shown here in grey). The curve smoothly saturates at a maximum of $y = k_{\max}$. Gradients $dy_2/dx \to 0$ as $x \to \infty$.

effective compared to the full BPTT method (see, e.g. Puskorius & Feldkamp, 1994; Williams & Peng, 1990). However choosing the value of h is sometimes non-trivial, and we address that in this paper through the introduction of soft-barrier fixed-length trajectories.

3. Proposed methods

We describe the method of smoothly truncating the gradients of soft-barrier cost functions in Section 3.1. We explain how to apply gradient-clipping to value gradients in Sections 3.2 and 3.3, where we explain how to incorporate it into the back-propagation through time algorithm.

3.1. Tanh-wrapper around cost functions

Two problems with soft-barriers are the infinite gradients associated with them, and the fact that their gradients are largest at points where the agent has often already irrecoverably lost control. We can avoid these infinite gradients by smoothly truncating the magnitudes of the steep-sided cost function. Assume the utility function includes a negative pure cost function, like $U_t = -C(x^{\text{penetration}})$, with $C(x) > 0$ as in (2). Then, we can modify the cost function to be smoothly truncated by:

$$\mathbb{C}_t := k_{\max} \tanh\left(\frac{1}{k_{\max}} C_t\right) \tag{3}$$

We refer to (3) as a "tanh wrapper with constant $k_{\max}$" around a step-cost function, and refer to $\mathbb{C}$ and C as the wrapped and unwrapped step-cost functions, respectively. In this paper, we find that using this tanh-wrapper can greatly help learning. The tanh wrapper changes a typical polynomial cost function from being an unbound function into a bound one, since (3) ensures that $\mathbb{C}_t < k_{\max}$; and one where the unbound gradients are removed. In the region $x \leq 0$, the original step-cost function is largely unaffected. But the function $\mathbb{C}$ smoothly saturates at the asymptote $y = k_{\max}$ (and with zero gradient) as $C_t \to \infty$. Fig. 4 illustrates this behaviour, for a soft-barrier function, with tanh-wrapper constant $k_{\max} = 3$.

The choice of tanh in (3) is probably not the only option available to achieve what we want; however, the tanh function was chosen for the following desirable properties:

- It is smoothly bound above by an asymptote along the line $y = 1$.
- Its derivative at $x = 0$ is 1. This means that if $0 \leq C_t \ll k_{\max}$ in (3), then we have $\mathbb{C}_t \approx C_t$.
- Because we assume $C_t \geq 0$, we only need consider the positive domain of the tanh function.
- The function tanh and its derivatives are readily available in neural-network software libraries.

The effect of the tanh-wrapper on a soft-barrier cost function in a 2D navigable environment is illustrated in Fig. 5. This figure highlights how the tanh-wrapper makes the learning algorithm concentrate on the most important parts of the trajectory for successful navigation.

3.2. Back-propagation through time algorithm for control

To solve the neurocontrol problem, we aim to maximise R in (1) with respect to $\vec{w}$ in the action network. We can achieve this by an iterative gradient-ascent weight update:

$$\Delta\vec{w} = \alpha \frac{\partial R(\vec{x}_0, \vec{w})}{\partial \vec{w}} \tag{4}$$

for some small positive learning rate α. Gradient ascent will naturally find local maxima of $R(\vec{x}_0, \vec{w})$, and has good convergence properties when the surface $R(\vec{x}_0, \vec{w})$ is smooth with respect to $\vec{w}$ and has an upper bound.

At each time step, when the agent chooses action $\vec{u}_t$ from state $\vec{x}_t$, we assume the environment applies a known differentiable function f, which takes the agent to the next state according to

$$\vec{x}_{t+1} = f(\vec{x}_t, \vec{u}_t), \tag{5}$$

and gives it an immediate step reward, or *utility*, U_t, given by the function $U_t = U(\vec{x}_t, \vec{u}_t)$. The agent keeps moving, forming a trajectory of states $(\vec{x}_0, \vec{x}_1, \ldots, \vec{x}_{T_F})$, until the final time step $t = T_F$ is reached.

For our purposes, we assume that the functions $f(\vec{x}, \vec{u})$, $U(\vec{x}, \vec{u})$ and $A(\vec{x}, \vec{w})$ are known and differentiable, so that the gradient $\partial R/\partial \vec{w}$ can be computed analytically. The assumption on differentiable functions is not a strong limitation, as methods exist to allow effective non-differentiable training for neurocontrol (e.g. Prokhorov, 2006).

The back-propagation through time equations (Werbos, 1990), which are used to calculate $\partial R/\partial \vec{w}$, can be implemented easily with modern automatic-differentiation packages. However we derive them explicitly here.

The total trajectory reward $R(\vec{x}_0, \vec{w}) = \sum_{t=0}^{T_F-1} \gamma^t U_t$ can be written recursively as

$$R(\vec{x}, \vec{w}) = U(\vec{x}, A(\vec{x}, \vec{w})) + \gamma R(f(\vec{x}, A(\vec{x}, \vec{w})), \vec{w}) \tag{6}$$

with $R(\vec{x}_{T_F}, \vec{w}) = 0$ at the final time step.

To calculate the gradient of (6) with respect to $\vec{w}$, we differentiate using the chain rule:

$$\left(\frac{\partial R}{\partial \vec{w}}\right)_t = \left(\frac{\partial}{\partial \vec{w}}(U(\vec{x}, A(\vec{x}, \vec{w})) + \gamma R(f(\vec{x}, A(\vec{x}, \vec{w})), \vec{w}))\right)_t$$
$$= \left(\frac{\partial A}{\partial \vec{w}}\right)_t \left(\left(\frac{\partial U}{\partial \vec{u}}\right)_t + \gamma \left(\frac{\partial f}{\partial \vec{u}}\right)_t \left(\frac{\partial R}{\partial \vec{x}}\right)_{t+1}\right) + \gamma \left(\frac{\partial R}{\partial \vec{w}}\right)_{t+1}$$

In the notation used in the equation above, parentheses with a subscripted t indicates that the partial derivatives are all evaluated at the quantities $\vec{x}_t$ and $\vec{u}_t$ associated with the time step t. When a vector is differentiated with respect to another vector, e.g. $\partial f/\partial \vec{u}$, this denotes the transpose of the usual Jacobian notation.

Expanding this recursion gives,

$$\left(\frac{\partial R}{\partial \vec{w}}\right)_0 = \sum_{t=0}^{T_F-1} \gamma^t \left(\frac{\partial A}{\partial \vec{w}}\right)_t \left(\left(\frac{\partial U}{\partial \vec{u}}\right)_t + \left(\frac{\partial f}{\partial \vec{u}}\right)_t \left(\frac{\partial R}{\partial \vec{x}}\right)_{t+1}\right) \tag{7}$$

This equation refers to the quantity $\partial R/\partial \vec{x}$ which can be found recursively by differentiating (6) and using the chain rule, giving

$$\left(\frac{\partial R}{\partial \vec{x}}\right)_t = \begin{cases} \left(\frac{DU}{D\vec{x}}\right)_t + \gamma \left(\frac{Df}{D\vec{x}}\right)_t \left(\frac{\partial R}{\partial \vec{x}}\right)_{t+1} & \text{if } t < T_F \\ \vec{0} & \text{if } t = T_F \end{cases} \tag{8}$$

where $\frac{D}{D\vec{x}}$ is shorthand for

$$\frac{D}{D\vec{x}} \equiv \frac{\partial}{\partial \vec{x}} + \frac{\partial A}{\partial \vec{x}} \frac{\partial}{\partial \vec{u}}. \tag{9}$$

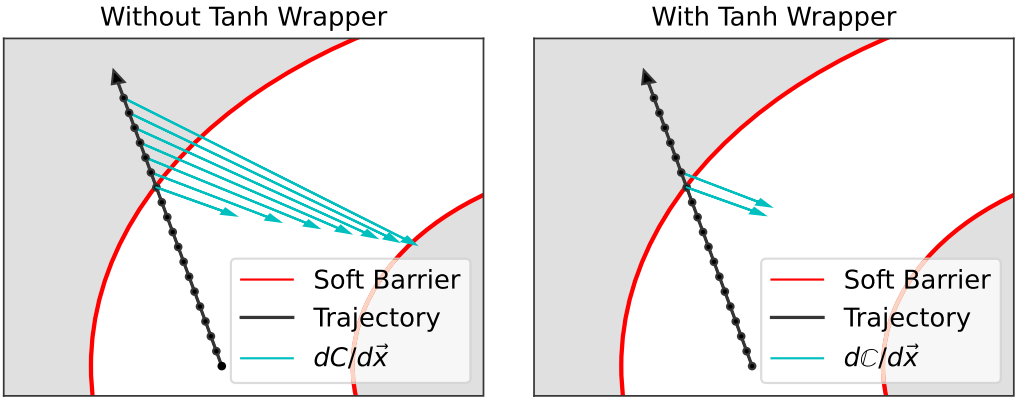


Fig. 5. How the cost-function's gradients are affected by the tanh wrapper in a navigable environment. In these examples, an agent with state $\vec{x} = (x, y)$ explores a 2D plane. A soft-barrier function $C(x^{\text{penetration}})$ is used following (2), where $x^{\text{penetration}}$ is the perpendicular distance that the agent $\vec{x}$ penetrates beyond the red sides of the white curved road, and $k_{\text{repel}} = (\text{road width})/4$, $k_{\text{pow}} = 8$ and $k_{\text{scale}} = 1$. The cyan arrows give the direction of $\partial C/\partial\vec{x}$. These arrows are drawn with magnitudes $\log(|\partial C/\partial\vec{x}| + 1)$. Note that all arrows with negligible magnitude, including all arrows on the white road, are not drawn. In the left figure, without the tanh wrapper, the gradient arrows are huge in inappropriate locations of the trajectory, making the learning algorithm concentrate most on those time steps with the largest arrows. This is analogous to trying to stop a car from falling off a steep-sided mountain road, by concentrating most on correcting those steering actions taken *after* the car has already departed the road; which is obviously a terrible strategy for improvement. Due to the log scale used, the largest arrow drawn dwarfs all other arrows in true magnitude. In the right-hand diagram, with the tanh wrapper, $\mathbb{C}$ is used in place of C, and the gradient arrows are largest, and give the most significant learning, at the most relevant time steps for steering; and also avoid exploding gradient magnitudes.

Eq. (8) can be understood to be backpropagating the quantity $\left(\frac{\partial R}{\partial \vec{x}}\right)_{t+1}$ through the action network, model and cost functions to obtain $\left(\frac{\partial R}{\partial \vec{x}}\right)_t$, and giving the BPTT algorithm its name.

3.3. Using gradient clipping with back-propagation through time

Suppose we have a clipping function $\text{clip}_\epsilon(\vec{a})$ which clips the magnitude (or largest components) of a vector $\vec{a}$ to be bounded above by ϵ. Although there are two possibilities here for implementing this function (clip-by-magnitude or clip-by-component), we only concentrate on the former in this paper:

$$\text{clip}_\epsilon(\vec{a}) = \begin{cases} \vec{a} & \text{if } |\vec{a}| < \epsilon \\ \epsilon\vec{a}/|\vec{a}| & \text{Otherwise} \end{cases} \tag{10}$$

In the most obvious approach, we could apply gradient clipping to the accumulated trajectory weight update, by modifying the learning Eq. (4) into:

$$\Delta\vec{w} = \alpha\text{clip}_\epsilon\left(\frac{\partial R(\vec{x}_0, \vec{w})}{\partial \vec{w}}\right) \tag{11}$$

However, an alternative, which is more targeted for BPTT, is to put the clipping around the $\frac{\partial R}{\partial \vec{x}}$ recursion, by changing (8) to:

$$\left(\frac{\partial R}{\partial \vec{x}}\right)_t = \text{clip}_\epsilon\left(\begin{cases} \left(\frac{DU}{D\vec{x}}\right)_t + \gamma\left(\frac{Df}{D\vec{x}}\right)_t\left(\frac{\partial R}{\partial \vec{x}}\right)_{t+1} & \text{if } t < T_F \\ \vec{0} & \text{if } t = T_F. \end{cases}\right). \tag{12}$$

If training batches were being used, then (12) would be applied separately to each trajectory within the batch.

We find that clipping by (12) shows superior performance compared to (11) in the experiments of Section 4. A reason for this improvement is because if the cost function produces the largest gradients at inappropriate time steps of the trajectory (such as time steps when the control of the agent has already been irrecoverably lost), then when Eq. (11) is used, those time steps with largest gradients will dominate the learning gradient after magnitude clipping; and the gradients at the time-steps which are important will be dwarfed to almost zero. Hence the irrecoverable time steps will dominate in the calculation of $\partial R/\partial\vec{w}$. In contrast, the clipping method (12) will still back-propagate gradients from those irrecoverable time-steps, but it will never allow them to dominate the whole gradient calculation of $\partial R/\partial\vec{w}$.

It should be noted that gradient clipping via (12) is actually a corruption of the true BPTT computation given by (7) and (8); hence this form of clipping does *not* yield true ascent on R. This is a theoretical drawback of this method, although in practice the method works well in the experiments of Section 4. This theoretical drawback is in contrast to the tanh-wrapper method, which does produce true gradient ascent. This can be understood because the tanh-wrapper method is merely replacing Eq. (1) by something like $R = \sum_t \gamma^t \tanh(-C_t)$; and then gradient ascent takes place on that quantity by (4) using the exact derivative calculations.

4. Experiments

We describe three simulated environments: car driving, pole balancing and a grid-connected converter control task. Since in all cases, trajectory length T_F is fixed and finite, we use discount factor $\gamma = 1$.

In all our experiments, the action network used is a multi-layer perceptron (MLP) with tanh activation functions on every layer (including the output layer). While the number of inputs and outputs the MLP has is problem dependent, in all experiments there were two hidden layers of 6 nodes each (with all short-cut connections present). The tanh activation function on the final layer restricts the magnitude of each component of the action vector $\vec{u}$ to be less than 1.

Shading in neural-network training graphs represents the 95% confidence intervals for the mean of multiple training trials.

4.1. Car driving experiment

In this experiment we propose a simplified driving-agent physics model. This environment is purposefully as simple as possible, with a perfectly circular track, which is sufficient to demonstrate the learning difficulties addressed by this paper. The car moves on the x-y plane, with state vector $\vec{x} := (x, y, \theta)$. The car moves in the direction θ, and constant speed $v_{\text{car}} = 1$. There is no accelerator control for the car; this is a fixed-speed driving problem.

At each time step, the agent chooses a steering action $\vec{u} = a \in [-1, 1]$. The environment's model function (5) is described by the update:

$$\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \leftarrow \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} \sin(\theta) \\ \cos(\theta) \\ ak_{\text{steer}} \end{pmatrix} v_{\text{car}}\Delta T \tag{13}$$

which is applied at every discrete time step t. Here $k_{\text{steer}} = 1$ is the steering rate of the car (which indicates that the radius of the smallest turning circle of the car is approximately $1/k_{\text{steer}}$), and $\Delta T = 0.25$ s

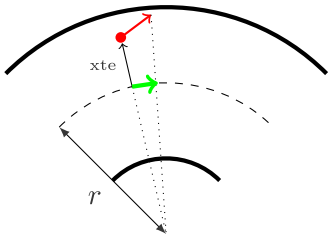


Fig. 6. Car on road, with cross-track error xte. The car is the red dot, moving in the direction of the red arrow, with length $v_{\text{car}}\Delta T$, and angle θ. The road centreline (dashed arc) has radius of curvature r, and direction θ_{road} given by the green arrow. The car's position on the road has lateral displacement (cross-track error) xte, with its positive direction defined as shown. The amount of centreline swept out by the car's displacement is given by the green solid arrow, of length $v_{\text{car}}\Delta T\cos(\theta_{\text{road}}-\theta)\frac{r}{r+\text{xte}}$.

is the time-interval used for integrating the motion. The car starts at state $(-10, 0, 0)$, i.e. initially facing in the direction of the positive y-axis, and the trajectory is expanded for $T_F = 240$ time steps before terminating. The road track used is a fixed circle of radius $r = 10$, with fixed "road half-width" rhw $= 3$. The two sides of the track are shown by the two thick black circles in Fig. 8(a), which also shows some sample trajectories.

An action network MLP with 3 inputs, 2 hidden layers, and 1 output was used. The input to the action network is an observation vector $g(\vec{x})$ defined by

$$g(\vec{x}) := \left(\sin(\theta_{\text{road}}-\theta), \cos(\theta_{\text{road}}-\theta), \frac{\text{xte}}{\text{rhw}}\right), \tag{14}$$

where xte is the "cross track error", which is defined to be the perpendicular component of the car's displacement vector from the centreline of the road (see Fig. 6). θ_{road} is the orientation of the road at the car's current position. Hence the action network operates as follows:

$$\vec{u}_t = A(g(\vec{x}_t), \vec{w}) \tag{15}$$

so that the output of the action network is the car's steering angle $a \in (-1, 1)$.

The use of the observation vector (14) provides a car-centric set of input variables to the action network (analogous to the information a human driver receives when viewing the road), making the driving decision much easier than if the input to (15) were $\vec{x}$.

For convenience, we define a "flat-bottomed soft barrier" (FBSB) function by:

$$f_{\text{FBSB}}(x, k_{\text{width}}, k_{\text{pow}}) := \max\left(\frac{|x|}{0.5k_{\text{width}}} - 1, 0\right)^{k_{\text{pow}}} \tag{16}$$

This FBSB function is illustrated in Fig. 7. It creates two soft barriers: one at $x > k_{\text{width}}$ and the other at $x < -k_{\text{width}}$ (shown in light grey in Fig. 7). There is a flat zone in between the two barriers, in the range $[-k_{\text{width}}/2, k_{\text{width}}/2]$ of zero cost. This "flat bottom" lets the agent explore freely within this area, without any cost. If the term k_{pow} is fairly high, then in practice the flat bottom approximately widens right up to the soft barriers (as shown in Fig. 7).

The FBSB function is a special case of the previous barrier function (2), since $f_{\text{FBSB}}(x, k_{\text{width}}, k_{\text{pow}}) \equiv C(|x| - k_{\text{width}})$ with $k_{\text{repel}} = k_{\text{width}}/2$ and $k_{\text{scale}} = 1$. The FBSB function is useful, e.g. for keeping an agent with coordinate x constrained within $-k_{\text{width}} < x < k_{\text{width}}$; for example to keep a car between the two sides of a road.

The utility function used in this car-driving experiment (with a tanh wrapper included here) is:

$$U_t = 20\tanh\left[\frac{1}{20}\left(-f_{\text{FBSB}}(|\text{xte}|, \text{rhw}, 8) - f_{\text{FBSB}}(|\theta_{\text{road}}-\theta|, \pi/2, 8)\right)\right]\Delta T + v_{\text{car}}\Delta T\cos(\theta_{\text{road}}-\theta)\min\left(\frac{r}{r+\text{xte}}, \frac{|r|}{|r|-\text{rhw}}\right) \tag{17}$$

where f_{FBSB} is defined by (16).

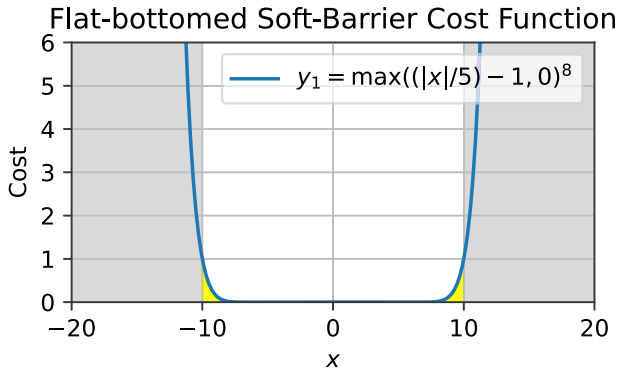


Fig. 7. Illustration of the flat-bottomed soft-barrier (FBSB) function (16), with $k_{\text{width}} = 10$ and $k_{\text{pow}} = 8$. The soft-barrier is defined to be at $|x| \geq k_{\text{width}}$, shown here in grey. Between the grey barriers, the cost function is zero in the region $|x| \leq 5$ and non-zero in the regions $5 < |x| < 10$. The latter regions are shown in yellow here, forming a repelling incentive for the agent, with $k_{\text{repel}} = k_{\text{width}}/2$.

The first f_{FBSB} penalty term in (17) aims to stop the car going off the sides of the road. This flat-bottomed penalty term allows the car to drive laterally across the road freely, but to encounter a steep penalty, rising rapidly once the road sides are approached.

The second f_{FBSB} aims to stop the car driving backwards (anti-clockwise) around the track. The constant $\pi/2$ allows the car to freely drive up to 90 degrees in either direction from θ_{road}. But then the FBSB penalty function for car orientation will penalise the car heavily if it starts facing the wrong way around the track. This term assumes that θ and θ_{road} do not reset to zero when the car completes a lap of the track, but keep accumulating smoothly indefinitely.[3]

The final term in (17) rewards the distance driven (projected onto the road's centreline) by the car, in each time step. This term and projection is explained in Fig. 6. This progress reward is defined to be positive when the car drives forwards (clockwise) around the track, and negative if it drives backwards. Because the projection is made onto the road's centreline, it encourages the car to sweep out as much arc-length as possible, which is achieved most when the car hugs the inside lane. This encourages the car to choose a shortest-path route, aiming to drive as many laps as possible within the given T_F time steps. Without the factor $r/(r + \text{xte})$, this would not happen. The min term in (17) is to prevent the car from gaining any unfair benefit by taking a shortcut across the central barrier of the road (e.g. driving across the point (0,0), which would instantly sweep out a massive road arc length).

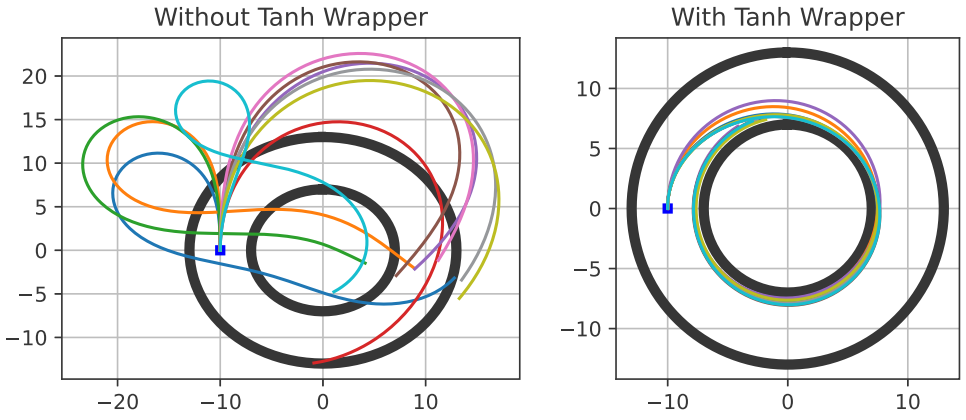
In each experiment below, we trained the action network in (15) on a single trajectory, in 10 independent trials, for 800 training iterations. For each training iteration, BPTT was used to compute the gradient $\partial R/\partial\vec{w}$ on the whole trajectory, and then this gradient was applied as a weight update (and accelerated as appropriate) by Adam (with learning rate 0.001).

4.1.1. Car-driver results (without gradient clipping or utility truncation)

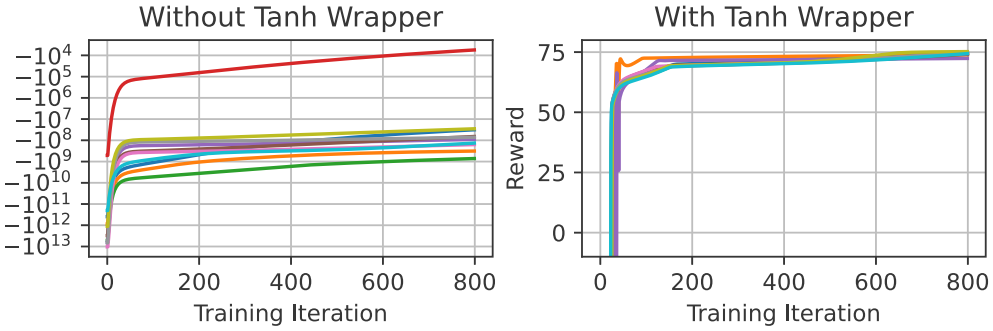
In this initial experiment, we omitted the tanh function from (17) (i.e. replaced the tanh in (17) by the identity function), and also omitted any gradient clipping. The purpose of this initial experiment, for this very-simple looking car driving problem, is to show the problems that can occur when fixed-length trajectories and soft-barriers are combined.

The results are shown in the left-hand diagrams of Fig. 8. In Fig. 8(a), each coloured line starting at the blue square at $(-10, 0)$

[3] This detail is obviously relevant to the construction of (17), in that if the car does a 360° spin, then it will be forever penalised by (17) until the car can figure out how to unspin itself.



(a) Graphs show learned individual trajectories (top-down view onto x-y plane), in 10 trials each, after 800 training iterations. Without the tanh wrapper (left image), the car gets stuck after driving off the road. Some trajectories take a looping detour, which cannot be un-knotted. Some trajectories mysteriously get very close to staying on the road, but get stuck, as if glued to the road sides. With the tanh-wrapper (right image), in every independent trial, the car has learned to drive round the track as efficiently as possible (i.e. it takes the inside lane to maximise its progress).



(b) Reward progress versus training iteration for learning the above trajectories. Each coloured curve here corresponds to the trajectory shown above with the matching colour.

Fig. 8. Results on car-driver problem, with and without the tanh-wrapper. Without the tanh wrapper, each car has driven off the road, but learning only makes tediously slow progress. Learning seems to get jammed up after huge gradients explode the denominator in Adam. The trajectories seem to get stuck in a largely un-recoverable position, with huge negative rewards. With tanh wrapper, the car learns to drive round the track as efficiently as possible (i.e. it takes the inside lane to maximise its progress). When tanh is used, the car has learned to hug the inside lane perfectly (upper diagram), and learning is stable and robust.

shows a trajectory produced by an individual trial. The corresponding coloured curves in Fig. 8(b) show the learning progress as measured by R versus training iteration, for that particular trial.

From this we can see that, without the tanh-wrapper, no individual trial solves the task. In each case, the total rewards obtained are hugely negative, and learning progress is minimal (according to Fig. 8(b) left); but in each case, the corresponding trajectories shown (in Fig. 8(a) left) show the car either driving off the road, or getting stuck to the sides of the road.

For the trajectories in Fig. 8(a) (left) that do *not* show tangled loops, it is at first very puzzling why those trajectories have not been by bent by continuous deformation (and monotonic upward progress on R) into trajectories that hug the inner lane tighter, as this is exactly what the gradient-ascent on R (via the BPTT + Adam iterative weight update) is defined to do. It seemed as if these trajectories are stuck to the outside of the track by a mysterious "glue". The explanation for this puzzling behaviour is because the large gradients encountered earlier on in learning have caused the denominator in Adam to explode and thus learning to grind to a virtual halt.

4.1.2. Car driver results (with tanh-wrapper)

In this experiment we show what happens when the tanh-wrapper in (17) is included. In this case, results are much better, and are shown in the right-hand diagrams of Fig. 8.

This shows all 10 trials working well. In each case the reward increases monotonically, upwards to roughly the same level. Also, in each case, the car has learned to hug the inside lane to make maximum track progression.

Comparison between the rewards attained in Fig. 8(b) between the left and right graphs is not directly possible, because the cost function used is different between them (i.e. one has a tanh in it and the other does not).

Hence, Fig. 9 is provided to verify the effect of the tanh-wrapper using a more objective measurement, i.e. how far the car has driven around the track in each case. This shows that the tanh wrapper enables the problem to be solved consistently, and without it, it is virtually impossible.

We also show results in Fig. 9 with a different optimiser, namely RPROP by Riedmiller and Braun (1993). RPROP does not have a denominator like Adam does which grows hugely when gradients exploded. Instead, RPROP's acceleration rate is sensitive to the rapidity with which individual components of $\partial R/\partial \vec{w}$ flip signs, over each different training iteration.

The results show that RPROP also benefits from using the tanh-wrapper. This indicates that the problem is not merely the large denominators that were growing in Adam. A key explanation for this is that the locations of the largest value-gradients are in unhelpful places (as illustrated in Fig. 5), and these dominate the learning gradients.

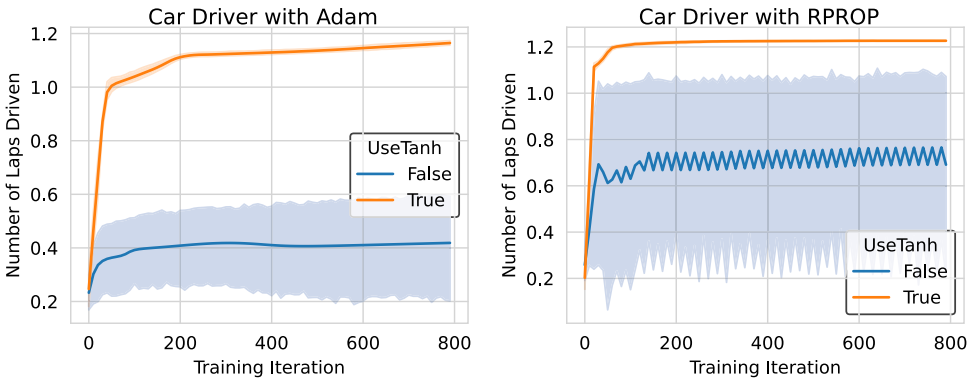


Fig. 9. Car driving problem: Training the network with and without the tanh wrapper around the cost functions, with two different optimisers (Adam and RPROP). Results show that in both cases, the tanh wrapper seems to help a lot.

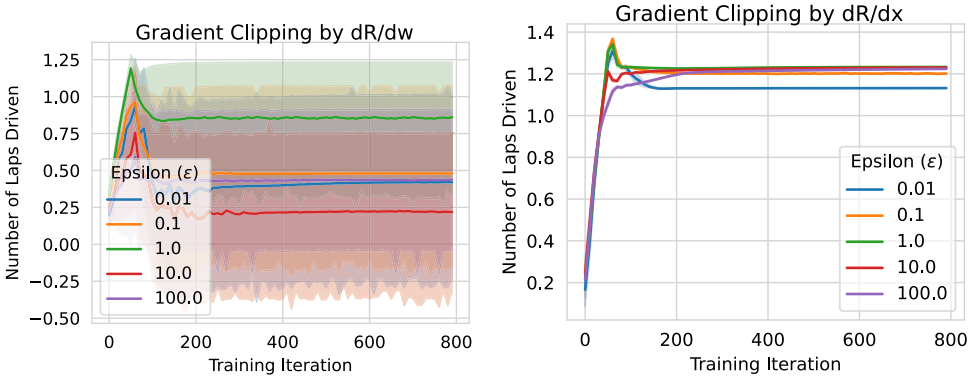


Fig. 10. A comparison of clipping with $\partial R/\partial \vec{w}$ versus clipping with $\partial R/\partial \vec{x}$, in the car-driver problem. These graphs show that the latter clipping method is much better, i.e. it makes learning feasible; whereas the other clipping method (11) does not.

4.1.3. Car driver with gradient clipping

The results for using gradient clipping are shown in Fig. 10, which compares the effectiveness of clipping on the backpropagated value gradient $\partial R/\partial \vec{x}$ via (12), versus the effectiveness of clipping on $\partial R/\partial \vec{w}$ by (11). The results show that the former method is robust over a wide range of clipping parameters $\epsilon \in \{0.01, 0.1, 1, 10, 100\}$, but the latter method performs significantly worse in all cases of ϵ. This is consistent with the explanation given in the penultimate paragraph of Section 3.3.

4.1.4. Magnitudes of learning gradients during training

Fig. 11 shows how the magnitude of the $\partial R/\partial \vec{w}$ vector is reduced during training when either the tanh-wrapper, or the $\partial R/\partial \vec{x}$ gradient-clipping method (with $\epsilon = 0.1$), is used. This figure shows the huge learning gradients that can arise with soft-barriers, and how these are reduced by many orders of magnitude by the tanh-wrapper and gradient clipping.

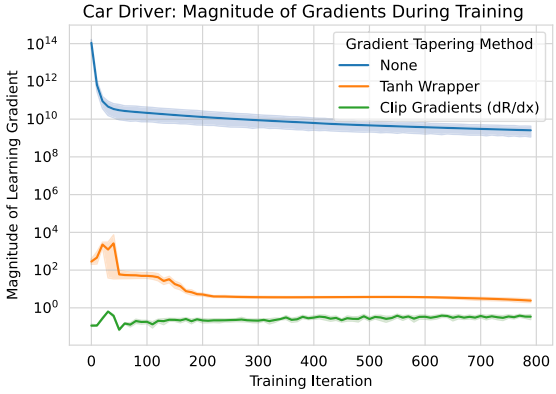


Fig. 11. Gradient magnitudes during learning for the car-driving problem.

4.1.5. Car driver results on non-circular track

To illustrate a more complicated track shape, we consider a track with centreline given by the polar equation:

$$r' = 8 + 4cos(2\theta') \tag{18}$$

and a road half-width rhw $= 1$, where (r', θ') are standard polar coordinates. See Fig. 12 for an illustration of this track. In this problem the car starts from position $(-12, 0)$, and it needs to learn to swap sides of the track during the chicane bend, in order to maximise the distance driven within the 240 time steps.

The neural input vector given by (14) was augmented to include an extra input, $\tanh(6/r)$, where r is the signed radius of curvature of the track (defined to be positive when the road bends to the right; and negative when it bends to the left) taken at the position of the car, and the $\tanh(6/r)$ function is used to rescale this input into the range $[-1, 1]$. This extra input lets the neural network know which bend of the track the car is currently at, so that it can change lanes as appropriate. The hidden layers and outputs of the neural network, and all other experimental parameters, remained unchanged from the previous experiments.

We trained a neural network with gradient clipping by (12), with $\epsilon = 1$, and also with the tanh wrapper, with the gradient clipping method working significantly better than the tanh wrapper method. Results of 10 independent trials are shown in Fig. 12.

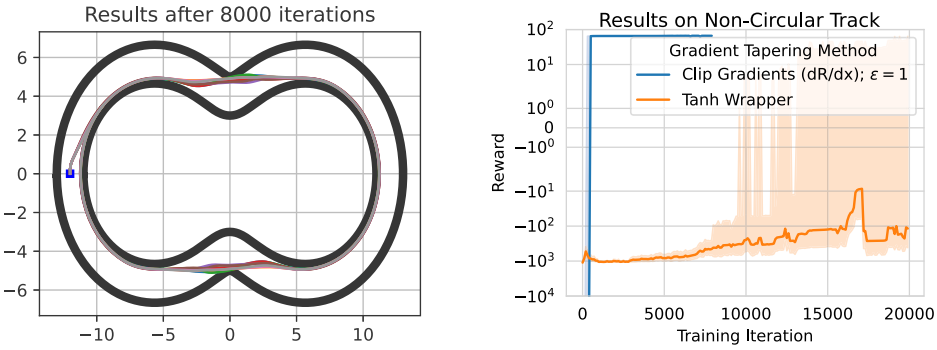


Fig. 12. Results for a non-circular track. Left diagram shows trajectories from 10 trials, trained with gradient clipping with $\epsilon = 1$. Each trajectory learns to find an approximation to the shortest path around the track and through the chicane bends. Right diagram shows that the results for gradient clipping were consistent, but the results with the tanh wrapper were not as good.

4.2. Pole-balancing experiment

We consider the classic cart–pole problem, defined by Florian (2007). This problem has been extensively studied in the control-theory and reinforcement learning literature (Barto, Sutton, & Anderson, 1983; Sutton & Barto, 1998). Prior solutions using BPTT/ADP were provided by Fairbank, Prokhorov, and Alonso (2014), Lendaris and Paintz (1997), Schaefer, Udluft, and Zimmermann (2007) and Wiedemann et al. (2023).

In this classic problem, the cart's position is x and its pole angle is θ. The system has Markovian state vector $\vec{x} = (x, \dot{x}, \theta, \dot{\theta})$. In the conventional version of this problem, with hard-barriers, the control continues until either the final time-step $T_F = 300$ is reached (indicating success), or until the pole topples such that $|\theta| > \pi/15$, or the cart runs off the end of its track such that $|x| > 2$. However, in our experiment, since soft-barriers are required, we remove these two latter constraints during training, such that the trajectory is run for $T_F = 300$ time steps whatever happens; and allow θ and x to grow arbitrarily large in magnitude.[4] To provide a fair comparison to hard barriers in our results, we report on how long the trajectory *would have* lasted before violating $|x| > 2$ or $|\theta| > \pi/15$ or $T_F = 300$.

For the cart–pole differentiable physics, we follow the equations given by Florian (2007). The physics is integrated with the Euler method with time-step duration $\Delta T = 0.002$s. The neural-network controller has 4 inputs, corresponding to the 4 components of $\vec{x}$ (without any rescaling), and 1 output with a tanh activation function to ensure that the control action $\vec{u} \in [-1, 1]$. The control action is rescaled by a factor of 10, giving a maximum force magnitude on the cart of 10 Newtons. For each weight update of BPTT, we averaged the weight update over 4 full trajectories from a set of 4 fixed random starting positions.

For the cart–pole problem, much of the existing RL literature uses a discounted total reward, with each step reward given by:

$$U_t = \begin{cases} 0 & \text{if } |\theta_t| < \pi/15 \text{ and } |x_t| < 2 \\ -1 & \text{Otherwise} \end{cases} \tag{19}$$

The above step reward amounts to a total trajectory reward of $R = -\gamma^T$, where T is the total balancing duration, and $\gamma \in (0, 1)$ is the discount factor. In RL and critic-based ADP algorithms, it is a critic which closes the loop between the approximate reward and the controller since (19) is not differentiable to enable application of BPTT.

[4] Note that this allows the pole to swing below the horizontal. Also, we let the angle θ accumulate without bound (i.e. with $|\theta| > \pi$ if the pole swings beyond one revolution).

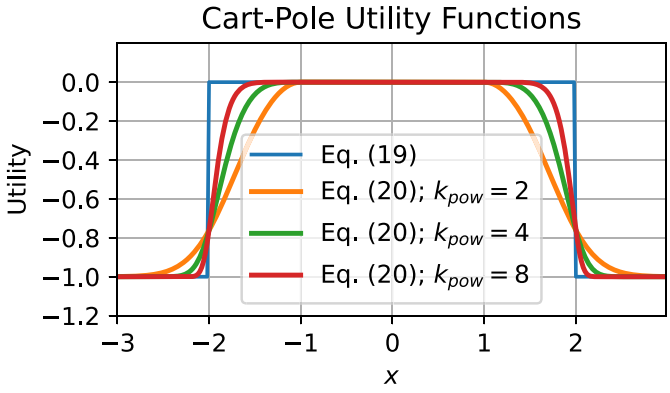


Fig. 13. Variations in cart–pole utility functions, with $k_{\max} = 1$. Only the dependence on x is shown here, with fixed $\theta = 0$ for this graph.

To side-step this problem to allow BPTT to work with cart–pole, prior publications have used a quadratic cost function of the form $U_t = -(k_1\theta_t + k_2 x_t)^2$, (Lendaris & Paintz, 1997; Schaefer et al., 2007; Wiedemann et al., 2023). Likewise, we also use a polynomial cost function, however the utility function we use at time t is:

$$U_t = -k_{\max} \tanh\left(\frac{f_{\text{FBSB}}(|x_t|, 2, k_{\text{pow}}) + f_{\text{FBSB}}\left(|\theta_t|, \frac{\pi}{15}, k_{\text{pow}}\right)}{k_{\max}} \right) \tag{20}$$

The above two FBSB terms in (20) are defined in (16), and are an almost-literal translation of the traditional cart–pole termination conditions (i.e. $|x| < 2$ and $|\theta| < \pi/15$) into FBSB functions. Also, choosing $k_{\max} = 1$ means that Eq. (20) can be viewed as a smoothed version of the traditional RL cart–pole reward function, i.e. (19), as can be seen in Fig. 13. This figure also illustrates how increasing k_{pow} just makes the utility function more like the square wave of (20).

Results are shown in Fig. 14. These results show that with the tanh wrapper, or with gradient-clipping via (12), convergence to a valid balancing solution is consistently and significantly stronger than without the tanh-wrapper or gradient clipping, over a range of k_{pow} values; with the gradient-clipping method being the strongest. The results are generally better for the lower values of k_{pow} used, which is presumably explained by the curves shown in Fig. 13 with lower k_{pow} values having gentler gradients more suited for gradient ascent.

Even though soft-barriers were used during training, i.e. allowing temporary violation of $|\theta| < \pi/15$ and $|x| < 2$, the results reported in Fig. 14 for those trials which show a balancing duration of 300 indicate

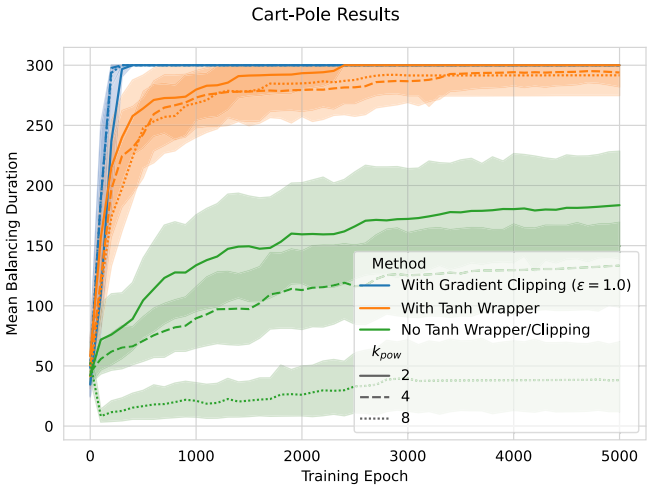


Fig. 14. Results on the cart–pole problem, using step-utility function given (20), with $k_{\max} = 1$ and various k_{pow} (with the tanh-wrapper removed in two of the curves plotted here). Despite the soft-barriers, the "balancing duration" reported here is defined to be the first time step t at which $|\theta_t| > \pi/15$ or $|x_t| > 2$ or $t = T_F$, i.e. is equivalent to a hard-barrier criterion.

that those final trained neural networks would be classified as a success under the original hard-barrier criteria.

4.3. Controller for a grid-connected converter

In this experiment we use the neurocontrol of a simulated electrical grid-connected converter (GCC) control problem. At any time t, the state of the GCC system is $\vec{x}_t = (d_t, q_t)$, where d and q are the GGC's d-axis and q-axis currents. The objective is to make the system solve a tracking problem, i.e. to maximise a reward function R given by:

$$R(\vec{x}_0, \vec{w}) := -\sum_{t=0}^{T_F-1} \left|\vec{x}_t - \vec{x}_t^*\right|^{k_{\text{pow}}} \tag{21}$$

where $\vec{x}_t$ is the actual state of the system at time t, and $\vec{x}_t^* = (d_t^*, q_t^*)$ is the reference (target) state of the system at time t. As before, k_{pow} defines the severity of the polynomial cost.

After the simulated physics of the system has been discretised (with a sampling period of 1 ms), the system state $\vec{x}(t)$ evolution equation is:

$$\vec{x}_{t+1} = \begin{pmatrix} 0.9242 & 0.3659 \\ -0.3659 & 0.9242 \end{pmatrix} \vec{x}_t + \begin{pmatrix} -357.7 & -68.17 \\ 68.17 & -357.7 \end{pmatrix} \vec{u}_t + \begin{pmatrix} 335.9 \\ -64.01 \end{pmatrix} \tag{22}$$

This state-evolution equation is equivalent to Equation 4 by Fairbank, Li, Fu, Alonso, and Wunsch (2014), to 4 significant figures. $\vec{u}_t$ is the control action at time t (a length-2 vector, with each component u^i restricted to $|u^i| \le 1$).

The action network has 4 inputs $(\tanh(\vec{x}_t/1000), \tanh((\vec{x}_t^* - \vec{x}_t)/100)$ and 2 outputs $(\vec{u}_t)$. This allows the neural network to directly control the GCC system. All training results in this section refer to having completed 3000 epochs of training with Adam with default learning rate 0.001.

The trajectory duration used for training was $T_F = 100$ time steps (i.e. 0.1 s of real time). When training the neural network, a fixed reference sequence $\vec{x}_t^*$ was used, for $t = 0, 1, \ldots, T_F - 1$, where $\vec{x}_t^*$ was changed every 20 time steps, as shown in Fig. 15.

We define the *mean absolute settling error* (MASE) as the average value of $\frac{1}{2}\left(|d_t - d_t^*| + |q_t - q_t^*|\right)$ over all time-steps lying within the second half of any of the reference plateaus along the whole trajectory shown in Fig. 15 (i.e. over all time steps t such that $10 \le (t \mod 20) \le 19$). Under this metric, the tracking solution shown in Fig. 15 has a MASE of 1.3.

For this experiment it was necessary to modify the tanh wrapper illustrated in Fig. 4 so that it did not have an entirely horizontal plateau. This was because in the GCC problem, when the system state $\vec{x}$ is far from the tracking target $\vec{x}^*$, the horizontal plateaus of the tanh-wrapper lose all useful gradient information. This can cause learning to cease up in any tracking problem where even far-distant tracking objectives still need actively seeking. This was not a problem for the previous two experiments, where soft barriers were used to represent hard barriers, beyond which control of the system is already irrecoverably lost.

To address this, we use a modified tanh-wrapper system shown in Fig. 16, which has inclined tangent half-line arms.

The straight inclined arms in Fig. 16 are implemented simply as extra piece-wise functions, analogous to a Huber loss function (Huber, 1992). These straight arms are attached a user-chosen points A and B, such that they are tangent half-lines to the central curve, according to:

$$\mathbb{C}(x) = \begin{cases} k_{\max} \tanh\left(\frac{|x|^{k_{\text{pow}}}}{k_{\max}}\right) & \text{if } |x| < x_B \\ m|x - x_B| + y_B & \text{if } |x| \ge x_B \end{cases} \tag{23}$$

with $x = |\vec{x}_t - \vec{x}_t^*|$, the tracking error; $y_B = k_{\max} k_{\text{hr}}$ and $k_{\text{hr}} \in (0, 1]$ is a user-chosen parameter.

Since the straight half-lines are defined to be tangents to the central curve, the variables m, x_B and y_B in (23) are uniquely determined by the user-choice of the single variable k_{hr}. We refer to this single variable choice as a height ratio. In the example curve in Fig. 16, we chose $k_{\text{hr}} = 0.99$, so the angled straight lines start when the tanh curve has reached 99% of its vertical way up towards the old asymptote. This gave a gradient $m = 0.112$ and $x_B = 2.82$, to 3 s.f.

The action network was trained with and without the tanh-wrapper given by (23), for various values of $k_{\text{pow}} \in \{1, 2, 4\}$, and with $k_{\max} = 200$. The tanh-wrapper with angled sides was used with various values of k_{hr}.

Results are shown in Fig. 17. Firstly, this figure shows the problem does not appear solvable for $k_{\text{pow}} > 1$ with a reasonable MASE, unless an appropriate tanh-wrapper is used. When no tanh-wrapper is used, the MASE shown in Fig. 17 for $k_{\text{pow}} = 4$ is approximately 75. This is why prior published work (Fairbank, Li, et al., 2014) only used $k_{\text{pow}} = 1$. However the use of an appropriate tanh-wrapper allows the solution for $k_{\text{pow}} \in \{2, 4\}$ (the MASE then drops to approximately 1.0 for $k_{\text{pow}} = 4$).

When the tanh-wrapper with angled sides is used, the values of k_{hr} value used are $\{0.9, 0.95, 0.99\}$; and all work well with all powers k_{pow}. Without the angled sides though, the tanh-wrapper does not perform well (see Fig. 17). This is because for an initially-randomised neural network, the trajectory generated is far away from the reference sequence shown in Fig. 15, and thus for the majority of time steps the cost function is on the horizontal asymptote of the tanh function with no gradient information. Having the sloping sides to the tanh function as in Fig. 16 addresses this problem.

Having just a raw polynomial cost function (of the form of (21)), with no tanh-wrapper at all and $k_{\text{pow}} = 4$, produces gradients that are too steep and explode/focus learning on the wrong parts of the trajectory. This is also shown in Fig. 17, where the final MASE is approximately 2 orders of magnitudes larger than compared to when an appropriate tanh wrapper is used.

Gradient clipping also works well and allows the MASE to be successfully made small in this problem, over the full range of k_{pow} values; as shown in Fig. 18. The graph shows the results working over a range of ϵ values, with best results for the two smallest values of ϵ used.

5. Conclusions

In this paper we have introduced and motivated soft-barrier functions, and ways to smoothly truncate their associated potentially-infinite gradients, to ease the learning task within neurocontrol. We have shown that soft barriers have some advantages over hard-barriers,

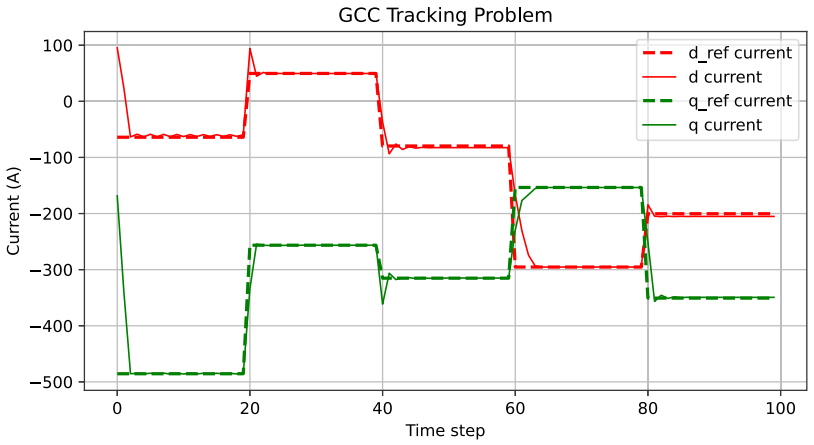


Fig. 15. The two thick dashed curves show the reference-current trajectories $(d^*, q^*)_t$ used in GCC tracking problem. The solid curves are the trained neuro-controller behaviour. As this is a tracking problem, the aim is for the solid curves to match the dashed curves as closely as possible (which is successfully performed here). This was trained using $k_{\text{pow}} = 1$ and a tanh-wrapper with $k_{\text{max}} = 200$ and $k_{\text{hr}} = 0.9$.

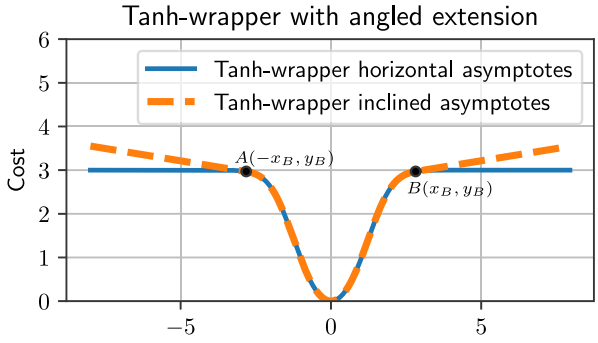


Fig. 16. Tanh-wrapper with angled extension. In this example, a simple quadratic function $C(x) = x^2$ is used with a tanh wrapper with $k_{\text{max}} = 3$. The function is defined piece-wise to artificially attach the angled sloping sides (according to (23)).

in that the Oscillating Corner-Avoidance problem (Fig. 1) and Deliberate Early Suicide problem (Fig. 2) are avoided with soft barriers. Soft barriers and fixed trajectory lengths also ensure that the parameter T_F appearing in (1) is constant, and so simplifies the derivative of $R(\vec{x}_0, \vec{w})$ in (1).

We have shown that without a tanh wrapper, or appropriate gradient clipping via (12), simple soft-barrier environments can become surprisingly troublesome for gradient ascent methods. A key reason to explain why soft-barriers are troublesome is the huge learning gradients, which causes learning to focus on the wrong steps of a trajectory as illustrated in Fig. 5.

We have demonstrated the effectiveness of the two methods proposed in this paper on the three different neurocontrol tasks. We find that out of the two methods, clipping of value-gradients seems to work the most robustly, over quite a large range of clipping magnitudes. However the gradient clipping loses the convergence guarantees of true gradient ascent. The second method, of truncating the magnitudes of steep-sided cost functions with a tanh-wrapper, also works effectively, and maintains true gradient ascent, but seems to work over a narrower range of its hyper parameter, and it struggled with the non-circular track experiment.

The ideas presented in this paper are applicable to environments which naturally have soft barriers (such as a tracking problem), or where a simulation is available of the environment which can be adjusted to include soft barriers. The introduction of soft-barriers can potentially benefit all ADP/RL and Neurocontrol algorithms, as all learning algorithms could otherwise suffer from the oscillating corner-avoidance or deliberate early-suicide problems. It is anticipated that the smooth gradient-truncation methods proposed in this paper would mostly benefit learning algorithms that explicitly manipulate gradients based on $\partial R/\partial \vec{x}$, such as BPTT, Dual Heuristic Programming (Prokhorov et al., 1995), and Value-Gradient Learning (Fairbank & Alonso, 2012b) since for scalar-based value function methods (such as Q-learning methods or TD-learning methods or HDP) simpler reward truncation schemes are possible (Mnih et al., 2015).

For many situations, soft barriers are only possible in simulated environments. It is usually not possible for an agent to travel straight through a physical barrier, but in two of the simulations considered in this paper, we have allowed that to happen. This modification is allowed if training an agent off-line, in a simulated environment. In some noteworthy environments through, such as a tracking problem (like the one in Section 4.3), soft barriers are the only natural choice.

When considering future limitations of using soft-barriers, it seems that soft-barriers may cause problems if they allow an agent to take a shortcut through a soft-barrier wall, for example in a maze environment. In our experiments we made some effort to prevent this kind of behaviour by letting the angles θ in the cart–pole and car-driver be measured cumulatively, and increase beyond 2π. This meant the agents were permanently penalised if the car or pole did unwanted "shortcuts" through disallowed regions of the state space, e.g. no credit would be given if the car's trajectory included an off-road spin followed some good road-following behaviour.

BPTT is supposedly one of the truly convergent ADP/RL algorithms, yet prior work has indicated that getting convergence can be troublesome. For example, (Freeman et al., 2021) wrote about Analytic Policy Gradients (i.e. BPTT) that it "does not currently produce locomotive gaits, and instead seems prone to being trapped in local minima on the environments we provide". Wiedemann et al. (2023) wrote that BPTT is "is prone to get trapped in local minima". Lendaris and Paintz (1997) noted that getting convergence for cart pole "was tedious" (but they used a critic-based ADP method which adds extra difficulty).

We have tried to make learning more robust and less dependent on the exact design choice of the reward functions used. Other works have used ad-hoc input vector rescalings, and reward functions to enhance convergence. We have used the same neural network architecture for all experiments, with minimal state-space rescalings for the input vector of the action network. Previous work has covered "tricks of the trade" to enable RL to work more robustly (Duell, Udluft, & Sterzing, 2012), and we are confident that the tricks proposed in this paper will be useful for the community.

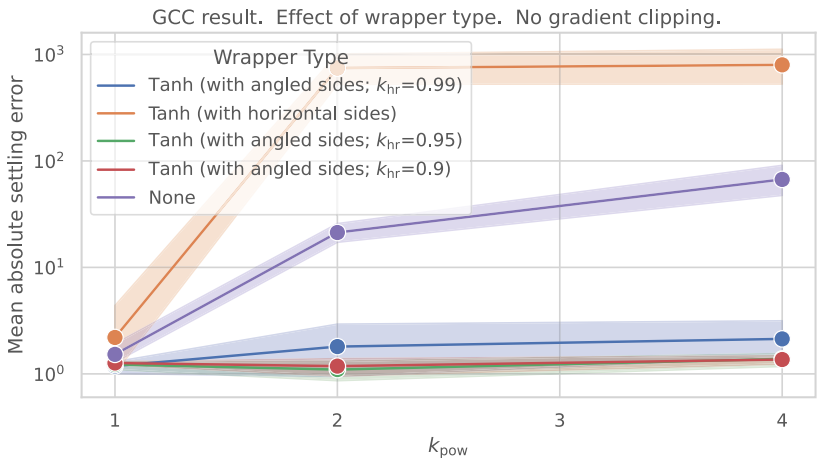


Fig. 17. Effect of using the Tanh Function on the GCC problem with different values of $k_{pow} \in \{1, 2, 4\}$; results showing the MASE after 3000 training iterations (with Adam). These results show that without any wrapper applied to the polynomial cost function, or when the tanh-wrapper with horizontal sides is used, the problem becomes very hard to solve with a reasonable settling error when $k_{pow} \geq 2$. But when the tanh wrapper is used with angled sides, the problem is solved over the full range of k_{pow} values.

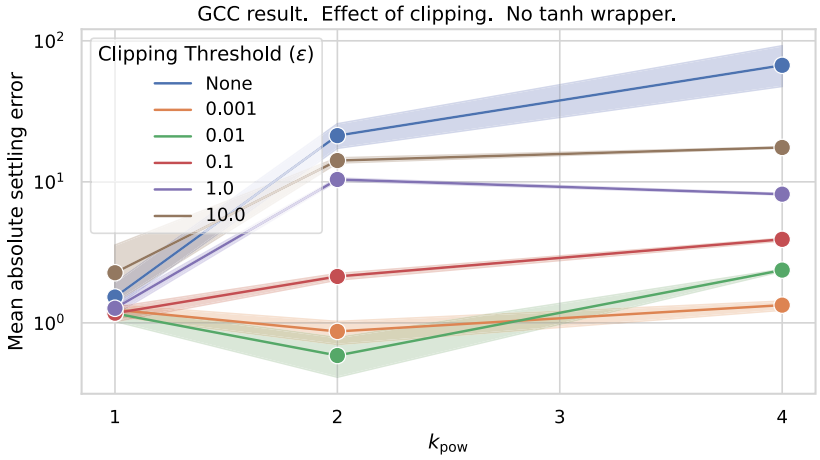


Fig. 18. Effect of using the Gradient clipping (via (12)) on the GCC problem with different values of $k_{pow} \in \{1, 2, 4\}$; results showing the MASE after 3000 training iterations (with Adam). The clipping threshold is the value of ϵ in (10). The results show that gradient clipping enables the problem to be solved well for the values of $\epsilon \leq 0.01$.

CRediT authorship contribution statement

Michael Fairbank: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Danil Prokhorov:** Writing – review & editing, Validation, Methodology, Investigation, Conceptualization. **David Barragan-Alcantar:** Investigation, Software, Investigation, Conceptualization, Writing – review & editing. **Spyridon Samothrakis:** Validation, Conceptualization. **Shuhui Li:** Validation, Resources, Investigation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

Spyridon Samothrakis acknowledges the support of the Business and Local Government Data Research Centre BLG DRC (ES/S007156/1) funded by the Economic and Social Research Council (ESRC).

Data availability

Data will be made available on request.

References

Ames, A. D., Coogan, S., Egerstedt, M., Notomista, G., Sreenath, K., & Tabuada, P. (2019). Control barrier functions: Theory and applications. In *2019 18th European control conference* (pp. 3420–3431). http://dx.doi.org/10.23919/ECC.2019.8796030.

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics*, *13*, 834–846.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*, 157–166.

de Avila Belbute-Peres, F., Smith, K., Allen, K., Tenenbaum, J., & Kolter, J. Z. (2018). End-to-end differentiable physics for learning and control. *Advances in Neural Information Processing Systems*, *31*.

Duell, S., Udluft, S., & Sterzing, V. (2012). Solving partially observable reinforcement learning problems with recurrent neural networks. In *Neural networks: tricks of the trade: second edition* (pp. 709–733). Springer.

Fairbank, M., & Alonso, E. (2012a). The divergence of reinforcement learning algorithms with value-iteration and function approximation. In *Proceedings of the IEEE international joint conference on neural networks 2012* (pp. 3070–3077). IEEE Press.

Fairbank, M., & Alonso, E. (2012b). Value-gradient learning. In *Proceedings of the IEEE international joint conference on neural networks 2012, xctitle=IJCNN'12)* (pp. 3062–3069). IEEE Press.

Fairbank, M., Alonso, E., & Prokhorov, D. (2013). An equivalence between adaptive dynamic programming with a critic and backpropagation through time. *IEEE Transactions on Neural Networks and Learning Systems*, *24*, 2088–2100.

Fairbank, M., Li, S., Fu, X., Alonso, E., & Wunsch, D. (2014). An adaptive recurrent neural-network controller using a stabilization matrix and predictive inputs to solve a tracking problem under disturbances. *Neural Networks*, *49*, 74–86. http://dx.doi.org/10.1016/j.neunet.2013.09.010, URL http://www.sciencedirect.com/science/article/pii/S0893608013002438.

Fairbank, M., Prokhorov, D., & Alonso, E. (2014). Clipping in neurocontrol by adaptive dynamic programming. *IEEE Transactions on Neural Networks and Learning Systems*, *25*, 1909–1920. http://dx.doi.org/10.1109/TNNLS.2014.2297991.

Fairbank, M., Samothrakis, S., & Citi, L. (2022). Deep learning in target space. *Journal of Machine Learning Research*, *23*, 297–342.

Florian, R. V. (2007). *Correct equations for the dynamics of the cart–pole system: Technical report*, Str. Saturn 24 400504 Cluj-Napoca, Romania: Center for Cognitive and Neural Studies (Coneural).

Freeman, C. D., Frey, E., Raichuk, A., Girgin, S., Mordatch, I., & Bachem, O. (2021). Brax–a differentiable physics engine for large scale rigid body simulation. arXiv preprint arXiv:2106.13281.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feed-forward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 249–256).

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*, 1735–1780.

Hu, Y., Anderson, L., Li, T.-M., Sun, Q., Carr, N., Ragan-Kelley, J., et al. (2019). Difftaichi: Differentiable programming for physical simulation. arXiv preprint arXiv:1910.00935.

Huber, P. J. (1992). Robust estimation of a location parameter. In *Breakthroughs in statistics: methodology and distribution* (pp. 492–518). Springer.

Kremer, S. C., & Kolen, J. F. (2001). *Field guide to dynamical recurrent networks*. Wiley-IEEE Press.

Lendaris, G. G., & Paintz, C. (1997). Training strategies for critic and action neural networks in dual heuristic programming method. In *Proceedings of international conference on neural networks*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*, 529.

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In D. McAllester S. Dasgupta (Ed.), *Proceedings of machine learning research, PMLR, Atlanta, Georgia, USA: vol. 28*, *Proceedings of the 30th international conference on machine learning* (pp. 1310–1318). URL https://proceedings.mlr.press/v28/pascanu13.html.

Prokhorov, D. V. (2006). Training recurrent neurocontrollers for robustness with derivative-free kalman filter. *IEEE Transactions on Neural Networks*, *17*, 1606–1616. http://dx.doi.org/10.1109/TNN.2006.880580.

Prokhorov, D. V., Santiago, R. A., & Wunsch, D. C. (1995). Adaptive critic designs: A case study for neurocontrol. *Neural Networks*, *8*, 1367–1372.

Prokhorov, D., & Wunsch, D. (1997). Adaptive critic designs. *IEEE Transactions on Neural Networks*, *8*, 997–1007.

Puskorius, G. V., & Feldkamp, L. A. (1994). Truncated backpropagation through time and kalman filter training for neurocontrol. In *Proceedings of 1994 IEEE international conference on neural networks, vol. 4* (pp. 2488–2493).

Riedmiller, M., & Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proc. of the IEEE intl. conf. on neural networks* (pp. 586–591).

Rohwer, R. (1990). The moving targets training algorithm. In *Advances in neural information processing systems* (pp. 558–565).

Sarangapani, J. (2018). *Neural network control of nonlinear discrete-time systems*. CRC Press.

Schaefer, A. M., Udluft, S., & Zimmermann, H.-G. (2007). A recurrent control neural network for data efficient reinforcement learning. In *2007 IEEE international symposium on approximate dynamic programming and reinforcement learning* (pp. 151–157). IEEE.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: an introduction*. Cambridge, Massachussetts, USA: The MIT Press.

Toussaint, M. A., Allen, K. R., Smith, K. A., & Tenenbaum, J. B. (2018). Differentiable physics and stable modes for tool-use and manipulation planning.

Wang, F.-Y., Zhang, H., & Liu, D. (2009). Adaptive dynamic programming: An introduction. *IEEE Computational Intelligence Magazine*, *4*, 39–47.

Werbos, P. J. (1990). Backpropagation through time: What it does and how to do it. In *Proceedings of the IEEE, volume 78, no. 10* (pp. 1550–1560).

Werbos, P. J. (2018). From backpropagation to neurocontrol. *Intelligent Systems*, 2.1–2.11.

Wiedemann, N., Wüest, V., Loquercio, A., Müller, M., Floreano, D., & Scaramuzza, D. (2023). Training efficient controllers via analytic policy gradient. In *2023 IEEE international conference on robotics and automation* (pp. 1349–1356). IEEE.

Williams, R., & Peng, J. (1990). An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, *2*, 490–501.

Yilmaz, A., & Poli, R. (2022). Successfully and efficiently training deep multi-layer perceptrons with logistic activation function simply requires initializing the weights with an appropriate negative mean. *Neural Networks*, *153*, 87–103. http://dx.doi.org/10.1016/j.neunet.2022.05.030, URL https://www.sciencedirect.com/science/article/pii/S0893608022002040.