LETTER Communicated by Paul F.M.J. Verschure

Closed-Loop Deep Learning: Generating Forward Models With Backpropagation

Sama Daryanavard
s.daryanavard.1@research.gla.ac.uk
Bernd Porr
bernd.porr@glasgow.ac.uk
Biomedical Engineering Division, School of Engineering, University of Glasgow, Glasgow G12 8QQ, U.K.

A reflex is a simple closed-loop control approach that tries to minimize an error but fails to do so because it will always react too late. An adaptive algorithm can use this error to learn a forward model with the help of predictive cues. For example, a driver learns to improve steering by looking ahead to avoid steering in the last minute. In order to process complex cues such as the road ahead, deep learning is a natural choice. However, this is usually achieved only indirectly by employing deep reinforcement learning having a discrete state space. Here, we show how this can be directly achieved by embedding deep learning into a closed-loop system and preserving its continuous processing. We show in z-space specifically how error backpropagation can be achieved and in general how gradient-based approaches can be analyzed in such closed-loop scenarios. The performance of this learning paradigm is demonstrated using a line follower in simulation and on a real robot that shows very fast and continuous learning.

1 Introduction

Reinforcement Learning (Sutton & Barto, 1998) has enjoyed a revival in recent years, significantly surpassing human performance in video games (Deng et al., 2009; Guo, Singh, Lee, Lewis, & Wang, 2014). Its success is owed to a combination of variants of Q learning (Watkins & Dayan, 1992) and deep learning (Rumelhart, Hinton, & Williams, 1986). This approach is powerful because deep learning is able to map large input spaces, such as camera images or pixels of a video game, onto a representation of future rewards or threats, which can then inform an actor to create actions as to maximize such future rewards. However, its speed of learning is still slow, and its discrete state space limits its applicability to robotics.

Classical control, on the other hand operates in continuous time (Phillips & Harbor, 2000), which potentially offers solutions to the problems encountered in discrete action space. Adaptive approaches in control develop

Neural Computation **32, 2122–2144** (2020)
https://doi.org/10.1162/neco_a_01317

forward models where an adaptive controller learns to minimize an error arising from a fixed feedback controller (e.g., proportional integral derivative (PID) controllers) called "reflex" in biology. This has been shown to work for simple networks (Klopf, 1986; Verschure & Coolen, 1991) where the error signal from the feedback loop successfully learns forward models of predictive (reflex) actions. In particular, there is a rich body of work in the area of movement control and experimental psychology where a subject needs to work against a disturbance—for example, pole balancing with hierarchical sensory predictive control (HSPC) (Maffei, Herreros, Sanchez-Fibla, Friston, & Verschure, 2017) or grasping of different objects (Haruno, Wolpert, & Kawato, 2001). One of the earliest models is feedback error learning (FEL), where the error is actually just the control output of the feedback controller itself, which then computes a forward model by using a "distal" or earlier signal such as the impact or a cue (Miyamoto, Kawato, Setoyama, & Suzuki, 1988). However, based on biological anticipatory control, there is mounting evidence that the brain also predicts future perceptual events and monitors its task performance in this way (Maffei et al., 2017; Popa & Ebner, 2018), which we will also honor in this work. Nevertheless, both HSPC (Maffei et al., 2017) and FEL have the drawback that they employ only single-layer networks where an error signal trains neurons with the help of a heterosynaptic learning rule.

In a more technical context, such a network also employing sensor predictions was able to improve the steering actions of a car where a nonoptimal hard-wired steering is then quickly superseded by a forward model based on camera information of the road ahead (Porr & Wörgötter, 2006; Kulvicius, Porr, & Wörgötter, 2007). Such learning is close to one-shot learning in this scenario because at every time step, the error signal from the PID controller is available and adjusts the network (Porr & Wörgötter, 2006). In these learning paradigms the error signal is summed up with the weighted activations of neurons to generate an action command for both the reflex and learning mechanism (Porr & Wörgötter, 2006). This has the advantage that the error signal also has a behavioral meaning, but the immediate summation of the error with the activations results in the loss of information, which means that the system is much more constrained, so it cannot be extended to deeper structures. This is reflected in Kulvicius et al. (2007) in a dedicated chained architecture, which shows that the design of the network topology is constrained because of the merging of the error signal with the activation. Thus, so far, these fast-learning correlation-based networks could not easily be scaled up to arbitrary deeper structures and consequently had limited scope.

A natural step is to employ deep learning (Rumelhart et al., 1986) instead of a shallow network to learn a forward model. If we directly learn a forward model with the deep network mapping sensor inputs to actions, then we no longer need a discrete action space. This will then allow potentially much higher learning rates because the error feedback will be continuous as

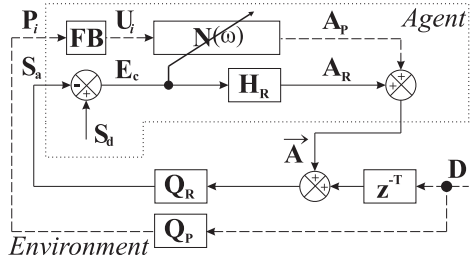

Figure 1: The closed-loop platform consists of an inner reflex loop (solid lines) and an outer learning loop (dashed lines); the learning unit $N(\omega)$ generates a forward model of the environment. U_i are the inputs to the network, generated by filtering the predictive signals, P_i, with a bank of low-pass filters, FB. Given these inputs, the network generates an action A_P that combats the disturbance D on its arrival at the reflex loop. Finally, the closed-loop error E_c gives an instructive feedback to the learning unit on how well A_P protected the system from D.

well. In order to achieve this, we need to define a new cost function for our deep network, which is defined within the closed-loop framework benchmarking the forward model in contrast to a desired output.

In this letter, we present a new approach for direct use of deep learning in a closed-loop context where it learns to replace a fixed feedback controller with a forward model. We follow the line of argumentation by Maffei et al. (2017) that anticipatory actions can be controlled by predictive sensory signals and that these are superior to motor anticipation because they can generate those actions based on predictive sensory cues alone. We show in an analytical way how to use the Laplace/z-space to solve backpropagation in a closed loop system. We then apply the solution first to a simulated line follower and then to a real robot where a deep network learns fast to replace a simple fixed PID controller with a forward model.

2 The Learning Platform

Before we introduce and explore the deep learner N, we need to establish our closed-loop system. The configuration depicted in Figure 1 is the architecture of this learning paradigm, which provides a closed-loop platform for autonomous learning. It consists of an inner reflex loop and an outer predictive loop that contains the learning unit. In the absence of any learning, the reflex loop receives a delayed disturbance Dz^{-T} via the reflex environment Q_R; this leads to the actual state S_a. Given the desired state S_d, the closed-loop error (E_c) is generated as $E_c = S_d - S_a$. This drives the agent to take an appropriate reflex action A_R as to recover to S_d and force E_c to

zero. However, the reflex mechanism, H_R, can react to the disturbance D only after it has perturbed the system.

Hence, the aim of the learning loop is to fend off D before it has disturbed the state of the robot. To that end, this loop receives D via the predictive environment Q_P and in advance of the reflex loop. This provides the learning unit with predictive signals P_i, and given its internal parameters ω, a predictive action is generated as $A_P = N(P_i, \omega)$.

During the learning process, A_P combined with A_R and Dz^{-T} travels through the reflex loop, and E_c is generated. This error signal provides the deep learner N with a minimal instructive feedback. Upon learning, A_P fully combats D on its arrival at the reflex loop (i.e., Dz^{-T}); hence, the reflex mechanism is no longer evoked and E_c is kept at zero.

3 Closed-Loop Dynamics

The aim of the learning is to keep the closed-loop error E_c to zero. In Figure 1, this signal is derived as $E_c(z) = S_d(z) - S_a(z)$; expansion of $S_a(z)$ yields[1]

$$E_c = S_d - Q_R(Dz^{-T} + E_c H_R + A_P) = \frac{S_d - Q_R(Dz^{-T} + A_P)}{1 + H_R Q_R}. \tag{3.1}$$

In mathematical terms, *learning* entails the adjustment of the internal parameters of the learning unit ω so that E_c is kept at zero. To that end, the closed-loop cost-function C_c is defined as the square of absolute E_c:

$$C_c := |E_c|^2. \tag{3.2}$$

Introduction of the closed-loop cost function (C_c) translates the learning goal into adjustments of ω so that C_c is minimized. This in turn ensures that E_c is kept at zero.

$$\frac{\partial C_c}{\partial \omega} = 2|E_c| \frac{\partial E_c}{\partial \omega}\bigg|_{\omega_{min}} = 0 \begin{cases} E_c = 0, \text{learning goal} \\ E_c \neq 0, \text{local minima} \end{cases}. \tag{3.3}$$

When analyzing the weight dependency of C_c in the context of closed-loop learning, it is intuitive to separate the dynamics of the closed-loop environment and the inner working of the controller using the chain rule, where A_P serves as the intermediate signal:

$$\frac{\partial C_c}{\partial \omega} = \frac{\partial C_c}{\partial A_P} \frac{\partial A_P}{\partial \omega} = G_C G_N. \tag{3.4}$$

[1] For brevity, we omit the complex frequency variable (z).

The former partial derivative, termed closed-loop gradient G_C, solely relates to the dynamics of the closed-loop platform; this is derived from equations 3.1 and 3.2:

$$G_C := \frac{\partial C_c}{\partial A_P} = 2|E_c| \frac{\partial E_c}{\partial A_P} = 2|E_c| \frac{-Q_R}{1 + H_R Q_R} = 2|E_c| T_R, \tag{3.5}$$

where the resulting fraction $\frac{-Q_R}{1+H_R Q_R}$ is the transfer function of the reflex loop T_R.

4 Toward Closed-Loop Error Backpropagation

To be able to link open-loop backpropagation to our closed-loop learning paradigm, we need to relate our closed-loop error E_c to the standard open-loop error utilized by backpropagation. In conventional open-loop implementations, the open-loop cost function C_o and open-loop error E_o are defined at the action output of the network:

$$C_o := |E_o|^2 := |A_P^d - A_P|^2, \tag{4.1}$$

where A_P^d is the desired predictive action. Minimization of C_o with respect to the internal parameters of the learning unit ω gives

$$\frac{\partial C_o}{\partial \omega} = \frac{\partial C_o}{\partial A_P} \frac{\partial A_P}{\partial \omega} = G_O G_N. \tag{4.2}$$

The former partial derivative is termed open-loop gradient G_O, from equation 4.1:

$$G_O = 2E_o \frac{\partial E_o}{\partial A_p} = -2|A_P^d - A_P|. \tag{4.3}$$

Now we relate the open-loop parameters to their closed-loop counterparts. To that end, E_c can be expressed as

$$\begin{aligned} E_c &= Q_R(Dz^{-T} + E_c H_R + A_P^d) - Q_R(Dz^{-T} + E_c H_R + A_P) \\ &= Q_R(A_P^d - A_P) = Q_R E_o. \end{aligned} \tag{4.4}$$

Given that Q_R is a nonzero transfer function, the open-loop error is kept at zero if and only if the closed-loop error is kept at zero:

$$Q_R \neq 0, \quad \text{therefore:} \quad E_c = 0 \iff E_o = 0. \tag{4.5}$$

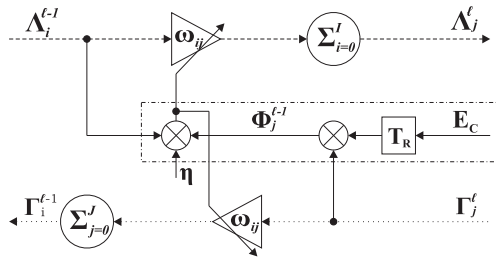

Figure 2: The computational unit shows the structure of the jth neuron in layer ℓ. This shows the forward propagation of the inputs (dashed arrows) and backpropagation (dotted arrows) of the error to the deeper layers, as well as the learning rule (solid arrows). Λ_j^{ℓ} is the activation of this neuron, and $\Lambda_i^{\ell-1}$ is the activation of the ith neuron in the previous layer. Γ_j^{ℓ} is the linking error (Γ) of this neuron, whlist $\Gamma_i^{\ell-1}$ is the linking error of the ith neuron in the previous layer. ω_{ij}^{ℓ} denotes the weight that connects the ith neuron in the previous layer to this neuron. The dash dotted rectangle marks the correlation of the closed-loop error (E_c) with the internal parameters of the neuron highlighting the *update rule*, where T_R is the transfer function of the reflex loop shown in Figure 1.

Having established how the closed-loop error can be fed into an error backpropagation framework, we are now able to present the inner workings of the learning unit.

5 The Inner Workings of the Deep Learner

Having explored the closed-loop dynamics, we now focus on the inner working of the learning unit. The latter partial derivative in equations 3.4 and 4.2, termed the network gradient G_N, is merely based on the inner configuration of the learning unit, which in this work is a deep neural network (DNN) with backpropagation (BP). Given that the network is situated in the closed-loop platform, its dynamics are expressed in z-space.

The forward propagation (FP) entails feeding the filtered predictive inputs U_i and generating the predictive action A_P. This is shown in Figure 2 with dashed arrows and is expressed as

$$\Lambda_j^{\ell} = \Sigma_{i=1}^{I} \omega_{ij}^{\ell} \Lambda_i^{\ell-1} \quad \text{where:} \quad \ell : 2, \ldots, L \quad \text{note that:} \quad \Lambda_j^{1} = \Sigma_{i=1}^{I} \omega_{ij}^{1} U_i, \tag{5.1}$$

where Λ denotes the activation of neurons,[2] L and I denote the total number of hidden layers and the total number of neurons in ℓth layer,

[2] Subscripts refer to the neuron's index, and superscripts refer to the layer containing the neuron or weight.

respectively. ω_{ij}^{ℓ} denotes the weights of the neurons in the z-domain, which are treated as constant since their rate of change is considerably slower in the time domain. We can formulate network gradient (G_N) with respect to specific weights of the network using equation 5.1:

$$G_N = \frac{\partial A_P}{\partial \omega_{ij}^{\ell}} = \frac{\partial A_P}{\partial \Lambda_j^{\ell}} \frac{\partial \Lambda_j^{\ell}}{\partial \omega_{ij}^{\ell}} = \frac{\partial A_P}{\partial \Lambda_j^{\ell}} \Lambda_i^{\ell-1}. \tag{5.2}$$

The resulting partial derivative, $\frac{\partial A_P}{\partial \Lambda_j^{\ell}}$, correlates with the closed-loop error to generate the internal error and hence is termed linking error Γ and is calculated using backpropagation:

$$\Gamma_j^{\ell} := \frac{\partial A_P}{\partial \Lambda_j^{\ell}} = \Sigma_{k=1}^{K} (w_{jk}^{\ell+1} \Gamma_k^{\ell+1}) \quad \text{where:} \quad \ell : (L-1), \ldots, 1$$
$$\text{note that:} \quad \Gamma_j^{L} = 1, \tag{5.3}$$

where K is the total number of neurons in the $(\ell+1)$th layer. Therefore, the Internal error Φ of the neuron, measuring sensitivity of the closed-loop cost function with respect to its activation, is given as refer to equation 3.5 and 5.3

$$\Phi_j^{\ell} := \frac{\partial C_c}{\partial \Lambda_j^{\ell}} = \frac{\partial C_c}{\partial A_P} \frac{\partial A_P}{\partial \Lambda_j^{\ell}} = 2^{\ell} |E_c| \frac{-Q_R}{1 + H_R Q_R} \Gamma_j. \tag{5.4}$$

The time-domain update rule for a specific weight can be expressed as the correlation of the internal error of the neuron with the input associated with that weight:

$$\Delta \omega_{ij}^{\ell} = \eta \Phi_j^{\ell}(z) \Lambda_i^{\ell-1}(-z), \quad \eta \ll 1. \tag{5.5}$$

The small learning rate η ensures that the time-dependant weight change is small compared to closed-loop dynamics. The gradient of the C_c with respect to an arbitrary weight is given as following, referring to equations 3.5, 5.1, and 5.4:

$$\begin{aligned} \frac{\partial C_c}{\partial \omega_{ij}^{\ell}} &= \frac{\partial C_c}{\partial \Lambda_j^{\ell}} \frac{\partial \Lambda_j^{\ell}}{\partial \omega_{ij}^{\ell}} = \frac{-2|E_c|Q_R}{1 + H_R Q_R} \Sigma_{k=0}^{K} (w_{jk}^{\ell+1} \Gamma_k^{\ell+1}) \Lambda_i^{\ell-1} \\ &= \Sigma_{k=0}^{K} (w_{jk}^{\ell+1} \Phi_k^{\ell+1}) \Lambda_i^{\ell-1}. \end{aligned} \tag{5.6}$$

This shows that the changes in C_c with respect to an arbitrary weight depends on the weighted internal error introduced in the adjacent deeper

layer. This is the propagation of C_c into the deeper layers and shows the backpropagation in the z-domain.

This concludes the derivation and formulation of our closed-loop deep learning (CLDL) paradigm. It is worth noting that CLDL is an online learning platform where the robot learns while driving and navigating through the environment. This is fundamentally different from conventional offline learning, where an agent is trained first and merely recalls the trained information when in use.

6 Results

The performance of our CLDL paradigm is tested using a line follower in simulation and through experiments with a real robot. The learning paradigm was developed into a bespoke low-level C++ external library (Daryanavard & Porr, 2020a). The transfer function of the reflex loop T_R, resulting from equation 3.5, is set to unity for the results.

6.1 Real Robot Experiments

6.1.1 Robot Configuration. The experiments with a real robot were carried out using a Parallax SumoBot as a mechanical test-bed, a Raspberry Pi 3B+ (RPi) for computation, and an Arduino Nano as the motor controller with a sampling rate of 33 Hz (Daryanavard & Porr, 2020b). Figure 3A shows the configuration of the robot. The robot is placed on a white canvas measuring 100 by 120 cm, with the path printed in black. The robot has two separate sets of sensors, where one set feeds in the reflex circuit and the other into our CLDL algorithm. The sensor set for the reflex is equipped with a symmetrical array of six light sensors, $[G]_6$, beneath the chassis and close to the canvas. The sensor set, which feeds into CLDL, is represented by a standard camera mounted on the robot. These are our predictors. Figure 3B shows the camera view and the array of light sensors in relation to one another. The six light sensors measure the intensity of the reflected light from the canvas in the form of 8-bit unsigned integer data, with 255 referring to full brightness and 0 referring to full darkness. As the robot navigates the canvas, if the light sensors align above the black line, their reading drops from a high value to a lower value, generating an error signal and thus indicating that the robot has gone off the path, and then generating a steering command sent to the servo motors adjusting the velocities of the right and left wheels $V_{R,L}$. In short, the sensors read the gray-scale value (GSV) of a small section of the canvas immediately underneath them, generating a corrective reflex reaction and, with that, our error signal for the CLDL algorithm.

6.1.2 Closed-Loop Error. The closed-loop error signal is defined as a weighted sum of the output of the light sensors array, $[G] = G_{R,1,2,3}$ and

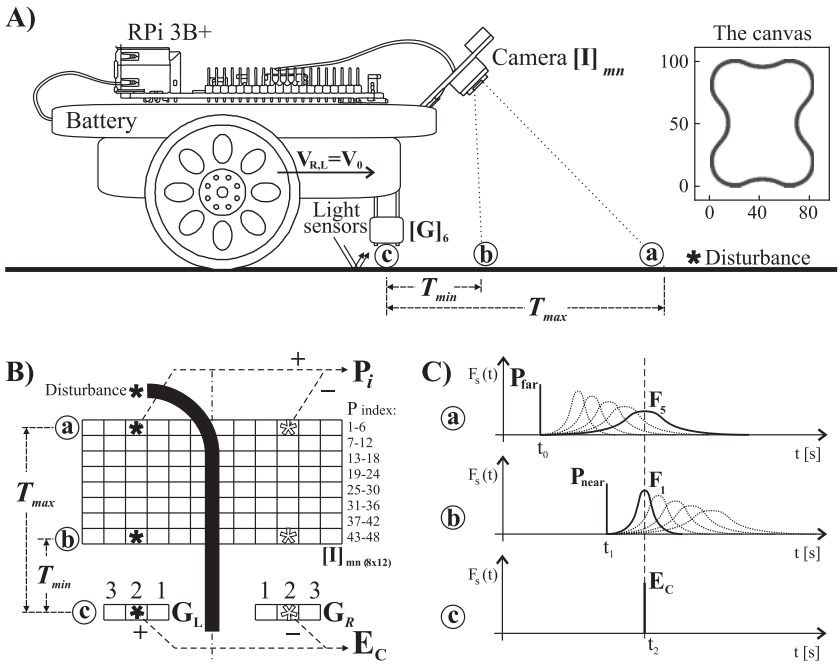

Figure 3: (A) The configuration of the robot and the canvas on which it navigates. A battery bank is placed on the chassis that powers the Raspberry Pi 3B+ (RPi) and provides power to the array of light sensors $[G]_6$ and the motors. The camera provides a view of the path ahead from point ⓐ to ⓑ. The star sign marks a disturbance in the path, such as a bend. (B) The view of the path ahead as a matrix of segments of the vision, $[I]_{mn}$, as well as the array of six light sensors $[G] = G_{R,1,2,3}$ and $G_{L,1,2,3}$. This shows the minimum and maximum time difference between the farthest and nearest predictors and the error signal. (C) The temporal relationship between impulse-shaped disturbances P_{far} and P_{near} and a light sensor E_c. It shows how filtering of the predictors P_{far} and P_{near} by the filter bank $F_1, \ldots, F_s$ causes appropriate delays so as to optimize their correlation with the error signal E_c derived from the light sensors.

$G_{L,1,2,3}$. The positioning and configuration of these sensors are shown in Figures 3A and 3B. The error signal is calculated as

$$E_c = 2 \cdot (G_{L1} - G_{R1}) + 3 \cdot (G_{L2} - G_{R2}) + 5 \cdot (G_{L3} - G_{R3}) \quad \text{[GSV]}. \quad (6.1)$$

The greater the deviation of the robot from the path (i.e., a nonzero $(G_{L3} - G_{R3})$), the greater the error signal, depending on the weighting of the sensor pairs. This results in a more informative error signal that is fed to the network for learning, and a smoother steering action is generated in return.

6.1.3 Predictors. The camera provides a view of the path ahead, which is divided into a matrix of 8 by 12 segments, $[I]_{m,n}$, as shown in Figures 3A and 3B. Predictive signals, P_i, are extracted from this matrix as

$$P_i = I_{mn} - I_{mn^*}, \quad \text{where } n^* \text{ is the sensor index symmetrical to } j. \tag{6.2}$$

This results in 48 predictive signals that are used for learning.

6.1.4 Filter-Bank. The predictive signals are filtered so as to cause the correct delay for optimum correlation with the closed-loop error signal. The specifications of these filters depend on environmental parameters and are obtained through a simple experiment. The robot is placed on a straight path with a disturbance ahead (a bend), which is shown as a star sign in Figures 3A and 3B. The robot is switched on and moves forward with a constant velocity of $V_0 = 5[\frac{cm}{s}]$ with the steering ability deactivated. The disturbance first appear, at position ⓐ at time t_0 and is sensed by the predictor farthest from the robot, P_{far}. The disturbance next appears at position ⓑ and is picked up by a the predictor nearest the robot, P_{near}, at time t_1. Finally, in position, ⓒ the disturbance is sensed by the light sensors, which generate an error signal E_c at time t_2. These signals, P_{far}, P_{near}, and E_c are shown in a timeline in Figure 3C. In order to cause an optimum correlation between the predictors and the error signal a maximum delay of $T_{\max} = t_2 - t_0$ and a minimum delay of $T_{\min} = t_1 - t_0$ is needed. These time delays are determined by the number of samples between the events, and given the sampling rate of 33 Hz we find that $T_{\max} = 0.4[s]$ and $T_{\min} = 0.2[s]$, as shown in Figure 3C. Thus, a bank of five second-order low-pass filters, FB, is designed with damping coefficients of $Q = 0.51$ and impulse responses with peaks from 0.2 to 0.4 seconds.

6.1.5 CLDL Algorithm. Figure 4 shows the configuration of our deep neural network used for experiments. This is a feedforward network composed of fully connected layers that performs backpropagation. The filtering stage of the 48 predictors is also illustrated in this figure, resulting in 240 filtered inputs U_i to the network. Thus, the network consists of an input layer with 240 neurons, as well as 11 hidden layers with 11 neurons in each, and finally an output layer with 3 neurons, giving a total of 364 neurons in the network.

Although the line-following task may not use the power of a deep neural network, this serves purely to benchmark the practicality and flexibility of our CLDL algorithm for use in both shallow (as will follow in the simulations section) and deep neural networks. An increase in the number of hidden layers is often associated with vanishing and exploding gradients that hinder the learning and adversely affect the performance of the network

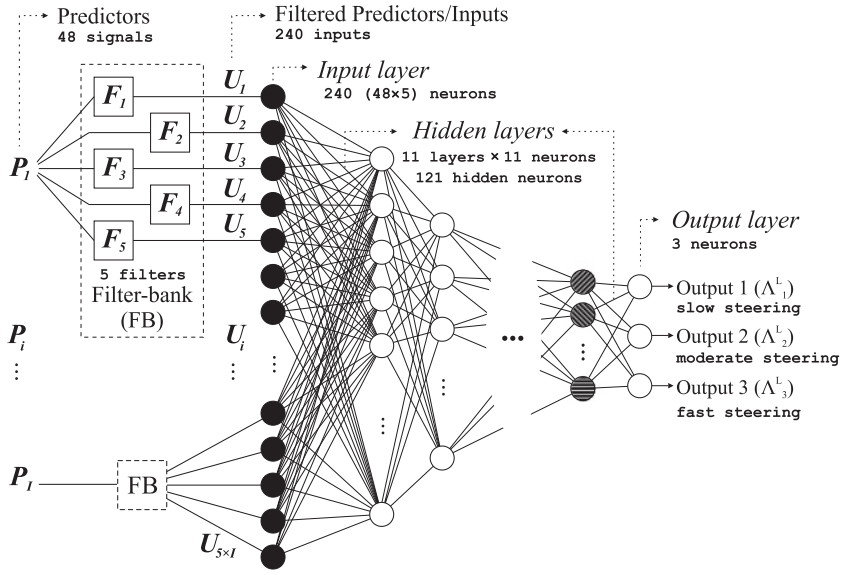

Figure 4: Architecture of the neural network: a feedforward network composed of fully connected layers. This shows the filtering stage of the predictors with a filter-bank FB resulting in delayed inputs U_i to the network. There are 240 neurons in the input layer, 11 neurons in each of the 11 hidden layers, and 3 neurons in the output layer. Λ^L_1, Λ^L_2, and Λ^L_3 are the outputs of the network allowing for slow, moderate, and fast steering.

(Pascanu, Mikolov, & Bengio, 2013; Bengio, Simard, & Frasconi, 1994; Bengio, Frasconi, & Simard, 1993). The issues that emerge from these gradients are often resolved by deliberate normalization of the weights and inputs, as well as careful manipulation of the computational units (neurons) (Pascanu et al., 2013). In this work we aim to present a convincing and authentic benchmark without reliance on such manipulation. Thus, we have experimentally arrived at a square-like structure for the deep network (containing 11 neurons in each 11 hidden layer) that combats the effect of vanishing and exploding gradients with no internal tuning of the weights.

6.1.6 Steering. The navigation of the robot is facilitated through the adjustment of velocities of the right and left wheels. This is done using the closed-loop error signal and the output of the network,

$$V_R = V_0 + (E_c + A_P) \quad \text{and:} \quad V_L = V_0 - (E_c + A_P), \tag{6.3}$$

where A_P is the predictive action previously shown in Figure 1 and is calculated here as a weighted sum of the three outputs of the network as shown in Figure 4

$$A_P = 1 \cdot \Lambda_1^L + 3 \cdot \Lambda_2^L + 5 \cdot \Lambda_3^L, \tag{6.4}$$

where Λ_i^L is the activation of the ith output neuron, which was formulated in equation 5.1. The weighting of the outputs means that each output neuron can learn to generate one of fast, moderate, or slow steering commands, resulting in a smooth navigation of the robot.

6.1.7 Reflex Trial. To be able to evaluate the performance of our algorithm, we need to first test the baseline performance without learning. We ran pure reflex trials where the robot navigated using its reflex mechanism only and in the absence of any learning. More specifically, this is when A_P is set to zero in equation 6.3. Figure 5A shows an example of such trial. It can be seen that the error signal is very persistent in its occurrence and amplitude. In this setting, the robot can only generate an appropriate steering command retrospectively, after an error has occurred. This sets a benchmark for evaluation of the deep learner.

6.1.8 Learning Trials. In a learning trial, the robot navigates using both the reflex and the predictive action of the network, as formulated in equation 6.3. In the context of learning, "success" refers to a condition where the closed-loop error shows a minimum of 75% reduction from its average value during reflex trials, for three consecutive seconds (or 100 samples). Figure 5B shows the error signal during one learning trial where $(\eta = 2 \cdot 10^{-1})$; this shows a strong reduction of the error signal over the first 50 seconds, where the learning is achieved rapidly. The closed-loop error acts as a minimal instructive feedback for the deep learner.

Figure 5C shows the final distribution of the weights in the first layer assigning different strength to different predictor signals. This is an 11 by 240 matrix of weights in the first layer showing the input index U_i on the x-axis and the neuron index on the y-axis (refer to Figure 4 for the configuration of the input layer). The inputs that are generated from each row of predictors are organized into blocks separated by vertical lines (refer to Figure 3B for the location of these predictors). The six predictors in each row are filtered by a bank of five filters, which results in 30 inputs, and a total of 330 weights in each block. It can be seen that the weight distribution closely follows the positioning of predictors with weights assigned to the outermost column of predictors, $P_{6,12,\ldots,42,48}$, having high values (black) and weights assigned to the innermost column of predictors, $P_{1,7,\ldots,37,43}$, having small values (white) to allow for a combination of abrupt and subtle steering, respectively.

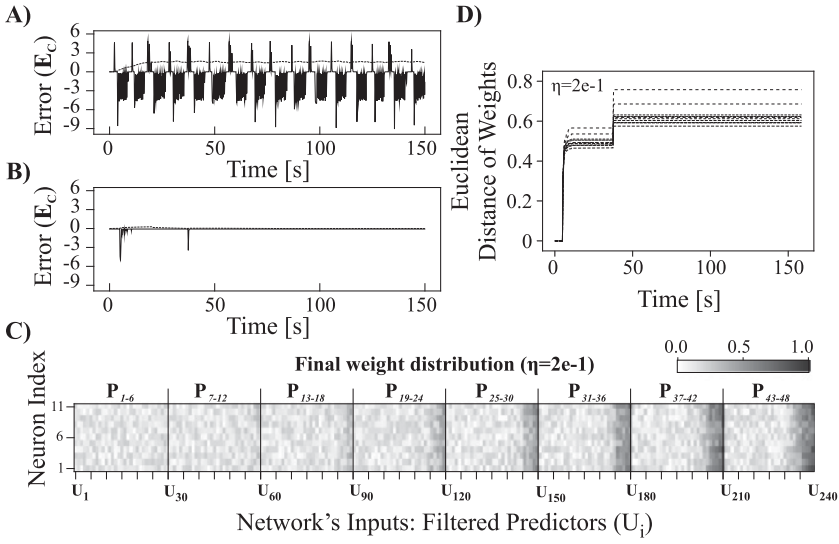

Figure 5: Real robot results for learning rate of $\eta = 2 \cdot 10^{-1}$. (A) The closed-loop error when robot navigates with reflex mechanism only. This sets a benchmark for evaluating the performance of the learning. Note the high amplitude and persistence of this signal. (B) The closed-loop error when the learning mechanism governs the navigation of the robot. Note the significant reduction of the error signal compared to the reflex data showing fast learning. (C) A greyscale map of the weight distribution in the first layer after the learning is completed. The x-axis shows the filtered inputs, and the y-axis shows the index of neuron in the first layer (refer to Figure 4). From the gradient, it can be seen that the farther the predictor is from the center line, the greater is the steering action (refer to Figure 3B). (D) The Euclidean distance of the weights in each layer during learning. It shows a stable convergence of all hidden layers with no vanishing or exploding gradients.

Figure 5D shows the weight change in each hidden layer. The purpose of this is to closely inspect the contribution of each hidden layer to the overall stability and convergence of the network. All layers show a stable increase in their weight change before they converge to their final value. The weight distance changes noticeably over the first 50 seconds, dictated by the closed-loop error, but arrives at a stable plateau as the error signal remains at zero.

Figure 6 shows another example of a learning trial similar to that in Figure 5 but with a smaller learning rate, $\eta = 2 \cdot 10^{-3}$. Figure 6A shows the predictive action for this trial. This is the contribution of the deep learner to the resultant differential speed of the robot (refer to equation 6.3). This quantity is initially small and inaccurate at the start of the trial, where the

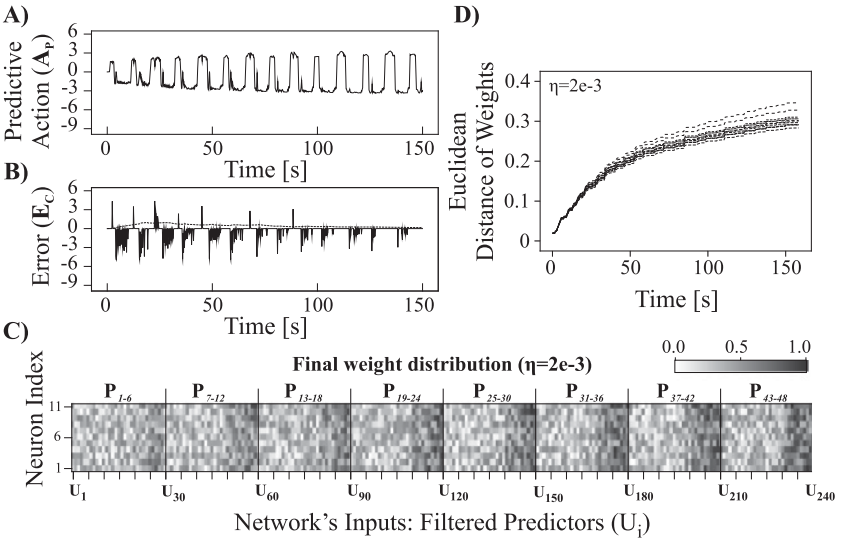

Figure 6: Real robot results for learning rate of $\eta = 2 \cdot 10^{-3}$. (A) The action of the network A_P. This is the contribution of the learning to the steering of the robot in anticipation of a disturbance (turn in the path). Note that as the learning improves, the amplitude of the steering increases and becomes more precise. (B) The closed-loop error when the learning mechanism governs the navigation of the robot. Note that the error is continuously reduced over time as the learning progresses. (C) A greyscale map of the weight distribution in the first layer after the learning is completed, as explained in Figure 5C, though with a cruder distribution. (D) The Euclidean distance of the weights in each layer during learning. All layers show a stable convergence, though with a more gradual convergence compared to that of Figure 5D.

reflex mechanism governs the navigation of the robot; however, the contribution of the learner grows larger and more precise over time as the learner begins to dominate the navigation. This transition from reflex-dominated to learning-dominated navigation is also seen in Figure 6B, where the error signal E_c decreases gradually toward a successful learning. Figure 6C shows the final distribution of the weights in the first layer, showing a similar trend but a cruder distribution compared to that of Figure 5C. Figure 6D shows the weight change in each hidden layer during this learning trial. The weight distance in all layers increases stably; however, they show a more gradual change compared to the learning trial with $\eta = 2 \cdot 10^{-1}$ shown in Figure 5D.

6.1.9 Tracking. The OptiTrack Infrared (IR) motion capture system was used to track the robot position in real time. It consists of 18 IR cameras and

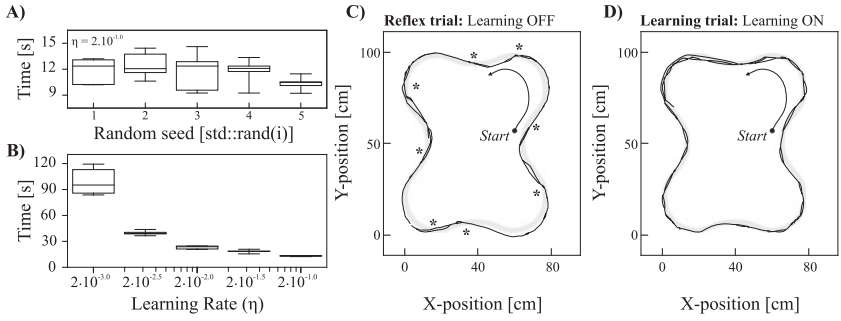

Figure 7: Real robot data. (A) The time taken until the success condition is met for five different random seed for weight initialization. Note that the random initialization of weights plays no significant role in the learning and success time. (B) The effect of learning rate on the time taken until the success condition is met. The data show a significant exponential decrease in the time taken before a successful learning is achieved. In other words, the learning is significantly faster for higher learning rates as it varies from 2.10^{-3} to 2.10^{-1}. (C) The trajectory of the robot for a reflex trial. The robot mostly resides off the path with crossovers marked with a star sign. (D) The robot's trajectory for a learning trial which shows that the robot mostly remains on and aligned with the path.

provides millimeter tracking resolution. Figure 7C shows the trajectory of the robot for a reflex trial, and Figure 7D shows this data for a learning trial. These are two independent trials with the reflex showing the navigation of the robot in the absence of learning and the learning trial showing the trajectory of the robot in the presence of learning from the start to the end of this trial. Figure 7C shows that when the learning is off, the path taken by the robot almost always remains outside the track, with multiple crossover points indicated by a star, whereas Figure 7D shows that with learning ($\eta = 2 \cdot 10^{-1}$), the trace of robot is aligned with the track.

6.1.10 Statistics and Reproducibility. The performance of the deep learner was repeated with five different random weight initializations using different random seeds $srand(i)$ where $i = \{0, 1, 2, 3, 4\}$. The learning rate was kept constant for these trials, $\eta = 2 \cdot 10^{-1}$. Figure 7A shows that different random initialization of the weights makes no significant difference to the time that it takes for the learner to meet the success condition. The learning trial was repeated with five learning rates $\eta : \{2 \cdot 10^{-3}, 2 \cdot 10^{-2.5}, 2 \cdot 10^{-2}, 2 \cdot 10^{-1.5}, 2 \cdot 10^{-1}\}$; each experiment was repeated five times for reproducibility. Figure 7B shows the time taken for the robot to meet the success condition for these trials. This data show an exponential decay of the success time as the learning rate is increased.

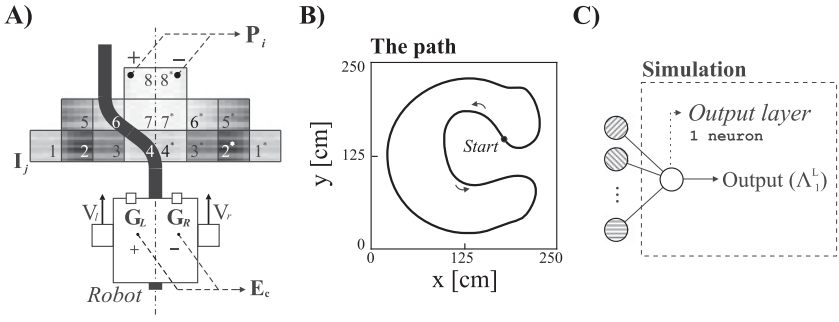

Figure 8: (A) Schematic of the virtual robot and its environment. The robot is composed of a body with two wheels with speeds of V_r and V_l and two ground sensors G_r and G_l, from which the closed-loop error E_c is generated. The robot is placed on a track and has a view of the path ahead through 16 symmetrical ground light sensors I_j from which the predictors P_i are obtained. (B) The C-shaped path on which the robot navigates beginning from the star point and in a loop. (C) The shows the output layer of the network used for simulations that consists of one neuron only. The rest of the network is the same as the one used for the real robot experiments shown in Figure 4.

6.2 Simulations with Virtual Robot. A virtual robot was designed using a simulation environment developed using QT5 and coded in C++ (Porr & Daryanavard, 2020). This allowed rapid verification of a variety of algorithm parameters, which show that in the simulated noise-free environment a shallow network is sufficient. Most important, a virtual robot allowed us to statistically infer the success of the learning paradigm through a large number of runs, which would have been impractical using the real robot.

6.2.1 The Virtual Robot. Figure 8A shows the configuration of the robot for the simulations that is placed on a track, as shown in Figure 8B. The robot is equipped with two light sensors, which function in the same way as the real robot experiments, measuring the GSV of the track underneath. The robot also has an array of light sensors placed in front and ahead of the body to obtain predictive signals. Note that the pyramidal shape of these sensors is simply set up to prevent spurious correlations arising from the adjacent path going in the opposite direction. Given that the network is purely correlation based, such high-level information cannot be learned by the network, and even in a correlation-based framework, the temporal credit assignment is far too long. The navigation of the robot is facilitated through the right and left velocities.

6.2.2 Reflex Error. The closed-loop error E_c is calculated using the right and left ground sensors shown in Figure 8A as

$$E_c = G_L - G_R. \tag{6.5}$$

This is the instructive signal that is used for learning.

6.2.3 Predictors. For the purpose of learning, the predictors P_i are generated using an array of 16 light sensors placed ahead of the robot in a pyramid structure as shown in Figure 8A. This results in eight predictive signals:

$$P_i = I_j - I_{j^*}, \quad \text{where } j^* \text{ is the sensor index symmetrical to } j. \tag{6.6}$$

6.2.4 Filter-Bank. The predictors are then filtered using a bank of five second-order low-pass filters (F_s), with damping coefficients of $Q = 0.51$ and impulse responses with appropriate delays, with their peaks occurring at 0.1 to 0.3 seconds (3 to 10 samples with sampling rate of 33 Hz), so as to cause the maximum correlation between predictors and the error signal. The specifications of these filters were obtained using a simple experiment as described for the real robot experiments.

6.2.5 The Shallow Learner. A feedforward network composed of fully connected layers was used in the same way as Figure 4, with only two hidden layers and with the output layer consisting of one output neuron, as shown in Figure 8C. The eight predictors are filtered as shown in Figure 4, resulting in 40 inputs to the network. Therefore, the network is configured with 40 input neurons, 2 hidden layers with 12 and 6 neurons, respectively, and a neuron in the output layer, giving a total of 59 neurons. The performance of the algorithm for deep neural networks was demonstrated in section 6.1; in this section, we show that this algorithm can also be applied to shallow networks; in particular, it is sufficient in noise-free simulation environments.

6.2.6 Steering. The steering of the robot is facilitated through adjustments of the left and right wheel velocities (see Figure 8A). The predictive action of the network is simply

$$A_P = \Lambda_1^L, \tag{6.7}$$

where Λ_1^L is the activation of the output neuron shown in Figure 8C and was formulated in equation 5.1. The navigation of the robot is dictated by the closed-loop error and the predictive action

$$V_R = V_0 + (\alpha E_c + \beta A_P) \quad \text{and:} \quad V_L = V_0 - (\alpha E_c + \beta A_P), \tag{6.8}$$

where V_0, α, and β are experimental tuning parameters set to 40 $[\frac{m}{s}]$, 200, and 100 respectively.

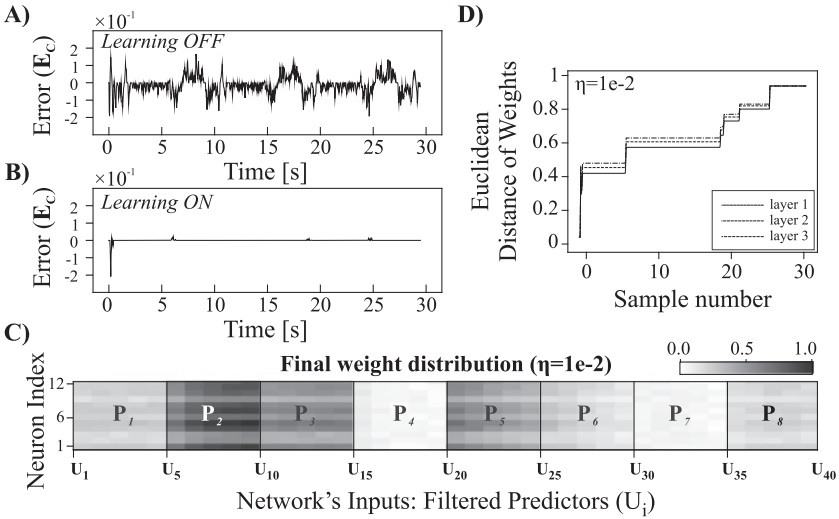

Figure 9: Simulation results with a learning rate of $\eta = 10^{-2}$. (A) The closed-loop error signal when navigating by reflex mechanism only. Note the high amplitude with RMS = 0.09 (refer to Figure 10A) and frequent occurrence of the error while the learning is off. (B) The closed-loop error signal when navigating by reflex and learning mechanism. Note that with learning, both the amplitude and the occurrence of the error have reduced significantly (RMS = 0.02) compared to that of reflex only. (C) A greyscale map of the weight distribution in the first layer after the learning is completed. The x-axis shows the filtered inputs, and the y-axis shows the index of neuron in the first layer (see Figure 4). Note that the weight distribution closely follows the location of predictors (see Figure 8A). (D) The Euclidean distance of the weights in each layer during this learning trial and a stable convergence of the network with no vanishing or exploding gradients.

6.2.7 Reflex Trial. The simulation environment is shown in Figure 8B, where the robot follows the C-shaped path, beginning from the start point, in a loop for 30 seconds (1000 iterations). Figure 9A shows the closed-loop error when the learning is off ($\eta = 0$). This is when the robot navigates with no learning, using only its fixed feedback controller (reflex); this serves as a benchmark and a visual aid to better comprehend and appreciate the performance of the fast online learner.

6.2.8 Learning Trial. For evaluation of the deep learner, a learning trial is designed to be one where the robot navigated using the predictive action, beginning from the start point and for the same number of samples as the reflex trials (30 seconds). Figure 9B, which illustrates the performance

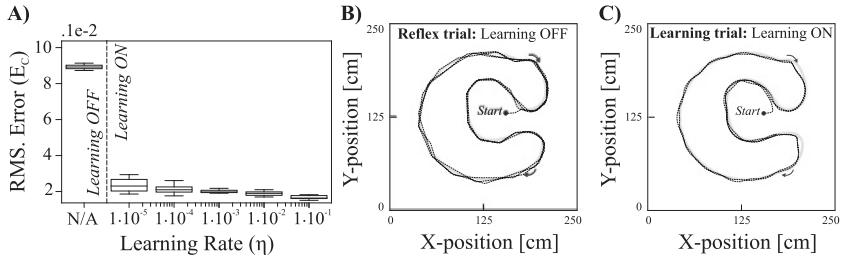

Figure 10: Virtual robot data. (A) The effect of learning rate on RMS value of closed-loop error E_c. Note the significant reduction of the closed-loop error in the presence of learning compared to that of reflex only, as well as the gradual improvement of learning (faster learning) with an exponential increase of the learning rate η from 10^{-5} to 10^{-1}. Examples of these trials are shown in Figures 9A and 9B for reflex and learning with $\eta = 10^{-2}$, respectively. (B) The trajectory of the robot for a reflex trial, showing a poor, uneven trace. (C) The trajectory of the robot for a learning trial showing a smooth and even trace.

of the deep learner, shows the closed-loop error when the learning is on ($\eta = 1 \cdot 10^{-2}$). This is when the robot learns while navigating. The robot exhibits very fast learning ($\approx$2 seconds), where the error signal is kept at, or close to, zero. Figure 9D shows the Euclidean distance of the weights in each layer from their initial random value. This shows a gradual increase from zero to its maximum during the course of one trial. Since the error signal is propagated as a weighted sum of the internal errors, all layers show a similar weight change.

Moreover, Figure 9C shows the final distribution of the first layer's weights in the form of a normalized grayscale map on completion of the learning, as shown Figure 5C. This shows a 12 (neurons) by 40 (inputs) matrix of weights in the first layer. The inputs generated to form each predictor sensor are grouped in blocks separated by vertical lines (see Figure 8A for the positioning of these predictors). Each predictor was filtered with a bank of five filters resulting in five inputs on the x-axis and 60 weights in the area of each block. The weights show an organized distribution, with greater weights (black) assigned to the outer predictors, $P_{2,5}$, and smaller weights (white) assigned to the inner predictors, $P_{4,7,8}$. This facilitates a sharper steering for the outer predictors, ensuring a smooth steering.

6.2.9 Tracking. The x- and y-coordinates of the robot were recorded during trials. Figure 10B show the trajectory of the robot over the course of a reflex trial, and Figure 10C shows data during a learning trial. These figures show that in the presence of learning, the steering is of an anticipatory nature and exhibits a smooth trajectory, whereas in the absence of learning, the steering is reactive and hence generates an abrupt response.

6.2.10 Statistics and Reproducibility. A set of simulations was carried out with five learning rates: $\eta : \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$. Each of the scenarios was repeated 10 times. Figure 10A shows the root mean square (RMS) of the error signal for each learning trial, as well as that of the reflex trials for comparison. All learning scenarios show a significantly smaller RMS error when compared to the reflex behavior; the error is reduced from around $9 \cdot 10^{-2}$ to around $2 \cdot 10^{-2}$ and lower. There is a gradual decrease in this value as the learning rate is increased. Smaller values of RMS error indicate both the reduction in the amplitude and the recurrence of the error signal.

7 Discussion

In this letter we have presented a learning algorithm that creates a forward model of a reflex employing a multilayered network. Previous work in this area used shallow (Kulvicius et al., 2007), usually single-layer, networks to learn a forward model (Nakanishi & Schaal, 2004; Porr & Wörgötter, 2006), and it was not possible to employ deeper structures. Model-free RL has been using more complex network structures such as deep learning by combining it with Q-learning, where the network learns to estimate an expected reward (Guo et al., 2014; Bansal, Akametalu, Jiang, Laine, & Tomlin, 2016). On first sight, this looks like two competing approaches because both use deep networks with error back-propagation. However, they serve different purposes, as discussed in Dolan and Dayan (2013) and Botvinick and Weinstein (2014), which lead to the idea of hierarchical RL, where RL provides a prediction error for an actor, which can then develop forward models.

In deep RL (Guo et al., 2014) and in our algorithm, we employ error backpropagation, a mathematical trick where an error/cost function is expanded with the help of partial derivatives (Rumelhart et al., 1986). This approach is appropriate for open-loop scenarios but for closed-loop approaches, one needs to take into account the endless recursion caused by the closed loop. In order to solve this problem, we have switched to the z-domain in which the recursion turns into simple algebra. A different approach has been taken by long short-term memory (LSTM) networks, where the recursion is unrolled and backpropagation in time is used to calculate the weights (Hochreiter & Schmidhuber, 1997), which is done offline, whereas in our algorithm the weights are calculated while the agent acts in its environment.

Deep learning is generally a slow-learning algorithm, and deep RL tends to be even slower because of the sparsity of the discrete rewards. Purely continuous or sampled continuous systems can be very fast because they have continuous error feedback so that in terms of behavior nearly one-shot learning can be achieved (Porr & Wörgötter, 2006). However, this comes at the price of forward models being learned from simple reflex behaviors wherein sophisticated planning can be achieved. For that reason, combining

the model-free deep RL with model-based learning to have a slow and a fast system has been suggested (Botvinick et al., 2019).

Still, our new approach is a deep architecture, and though we have demonstrated it through a line follower robot, it inherits all advantages from standard deep learning, such as convolutional layers and the development of high-level features (Deng et al., 2009), such as receptive fields. These features can then be used to create much more specific anticipatory actions than simple single-layer networks used in motor control to date (Maffei et al., 2017).

Forward models play an important role in robotic and biological motor control (Wolpert & Kawato, 1998; Wolpert, Ghahramani, & Flanagan, 2001; Haruno et al., 2001; Nakanishi & Schaal, 2004), where forward models guarantee an optimal trajectory after learning. With our approach, this offers opportunities to learn more complex forward models with the help of deep networks and then combine them with traditional Q-learning to planning those movements.

In the context of forward models, we should note that our model, like the ones by Miyamoto et al. (1988), Porr and Wörgötter (2006), and Maffei et al. (2017) learn the forward model for only one situation but would fail when different forward models were required, for example, being able to manipulate different objects. This has been addressed by the MOSAIC Model by Haruno et al. (2001) where multiple pairs of forward and inverse controllers were learned. However, this is beyond the scope of this work.

Acknowledgments

We offer our gratitude to Jarez Patel for his considerable technical and intellectual input to this work, Dave Anderson for aiding with the motion-capture system, and Bruno Manganelli for his initial contribution to the graphical user interface framework of the physical robot.

References

Bansal, S., Akametalu, A. K., Jiang, F. J., Laine, F., & Tomlin, C. J. (2016). Learning quadrotor dynamics using neural network for flight control. In *Proceedings of the 2016 IEEE 55th Conference on Decision and Control* (pp. 4653–4660). Piscataway, NJ: IEEE.

Bengio, Y., Frasconi, P., & Simard, P. (1993). The problem of learning long-term dependencies in recurrent networks. In *Proceedings of the IEEE International Conference on Neural Networks* (pp. 1183–1188). Piscataway, NJ: IEEE.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166.

Botvinick, M., Ritter, S., Wang, J. X., Kurth-Nelson, Z., Blundell, C., & Hassabis, D. (2019). Reinforcement learning, fast and slow. *Trends in Cognitive Sciences*, *23*(5), 408–422.

Botvinick, M., & Weinstein, A. (2014). Model-based hierarchical reinforcement learning and human action control. *Philosophical Transactions of the Royal Society B: Biological Sciences*, *369*(1655), 20130480.

Daryanavard, S., & Porr, B. (2020a). Sama-Darya/CLDL: Flexible closed-loop deep learning. https://zenodo.org/record/3922922#.Xz7dz3VKg_A

Daryanavard, S., & Porr, B. (2020b). Sama-Darya/lineFollowerRobot: Physical line-follower robot with deep learning in a closed-loop platform. https://zenodo.org/account/settings/github/repository/Sama-Darya/lineFollowerRobot

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition* (pp. 248–255). Piscataway, NJ: IEEE.

Dolan, R. J., & Dayan, P. (2013). Goals and habits in the brain. *Neuron*, *80*(2), 312–325.

Guo, X., Singh, S., Lee, H., Lewis, R. L., & Wang, X. (2014). Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning. In Z. Gharamani, M. Welling, C. Cortes, N. D. Lawrence, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems*, *27* (pp. 3338–3346). Red Hook, NY: Curran.

Haruno, M., Wolpert, D. M., & Kawato, M. (2001). Mosaic model for sensorimotor learning and control. *Neural Computation*, *13*(10), 2201–2220.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Klopf, A. H. (1986). A drive-reinforcement model of single neuron function: An alternative to the Hebbian neuronal model. In *AIP Conference Proceedings* (vol. 151, pp. 265–270). College Park, MD: American Institute of Physics.

Kulvicius, T., Porr, B., & Wörgötter, F. (2007). Chained learning architectures in a simple closed-loop behavioural context. *Biological Cybernetics*, *97*(5–6), 363–378.

Maffei, G., Herreros, I., Sanchez-Fibla, M., Friston, K. J., & Verschure, P. F. (2017). The perceptual shaping of anticipatory actions. *Proceedings of the Royal Society B: Biological Sciences*, *284*(1869), 20171780.

Miyamoto, H., Kawato, M., Setoyama, T., & Suzuki, R. (1988). Feedback-error-learning neural network for trajectory control of a robotic manipulator. *Neural Networks*, *1*(3), 251–265.

Nakanishi, J., & Schaal, S. (2004). Feedback error learning and nonlinear adaptive control. *Neural Networks*, *17*(10), 1453–1465.

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *Proceedings of the International Conference on Machine Learning* (pp. 1310–1318).

Phillips, C. L., & Harbor, R. D. (2000). *Feedback control systems*. Upper Saddle River, NJ: Prentice Hall.

Popa, L., & Ebner, T. (2018). Cerebellum, predictions and errors. *Frontiers in Cellular Neuroscience*, *12*, 524.

Porr, B., & Daryanavard, S. (2020). *Sama-Darya/enkiSimulator: Virtual line-follower robot with deep learning in a closed-loop platform.* https://zenodo.org/account/settings/github/repository/Sama-Darya/enkiSimulator

Porr, B., & Wörgötter, F. (2006). Strongly improved stability and faster convergence of temporal sequence learning by using input correlations only. *Neural Computation*, *18*(6), 1380–1412.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536.

Sutton, R., & Barto, A. (1998). *Reinforcement learning: An introduction.* Cambridge, MA: MIT Press.

Verschure, P. F., & Coolen, A. C. (1991). Adaptive fields: Distributed representations of classically conditioned associations. *Network: Computation in Neural Systems*, *2*(2), 189–206.

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*(3–4), 279–292.

Wolpert, D. M., Ghahramani, Z., & Flanagan, J. R. (2001). Perspectives and problems in motor learning. *Trends in Cognitive Sciences*, *5*(11), 487–494.

Wolpert, D. M., & Kawato, M. (1998). Multiple paired forward and inverse models for motor control. *Neural Networks*, *11*(7–8), 1317–1329.

Received January 30, 2020; accepted June 15, 2020.