frontiers in
NEUROROBOTICS

Reward modulated Hebbian plasticity as leverage for partially embodied control in compliant robotics

Ken Caluwaerts, Jeroen Burms and Joni Dambre

Journal Name:	Frontiers in Neurorobotics
ISSN:	1662-5218
Article type:	Original Research Article
First received on:	26 Jan 2015
Frontiers website link:	www.frontiersin.org

Frontiers in Neurorobotics

Research Article
201527 January 2015

1

Reward modulated Hebbian plasticity as leverage for partially embodied control in compliant robotics.

K. Caluwaerts[1,2], **J. Burms**[1] **and J. Dambre**[1]

[1] *Computing Systems Lab (Reservoir Team), Electronics and Information Systems Department (ELIS), Ghent University, Ghent, Belgium*
[2] *Intelligent Robotics Group, NASA Ames Research Center/Oak Ridge Associated Universities, Moffett Field CA, United States*

Correspondence*:
Joni Dambre - Joni.Dambre@UGent.be
,

Neural plasticity for rich and uncertain robotic information streams

2 **ABSTRACT**

3 We present a dual contribution to the Hebbian learning field. First, we demonstrate that a
4 simple Hebbian-like learning rule can be used to optimize the connection weights of recurrent
5 neural networks. We then apply this learning rule to compliant Tensegrity robots by optimizing
6 static feedback controllers that directly exploit the dynamics of the robot body. This leads to
7 partially embodied controllers, i.e., hybrid controllers that naturally integrate the computations
8 that are performed by the robot body into a neural network architecture.

9 The goal of this work is also twofold. First, it solves a crucial missing aspect of our recent work
10 in which we showed that static feedback controllers can be used to induce robust locomotion
11 in compliant robots. Previously, the problem of finding the desired motor signals was solved by
12 using Evolutionary Algorithms. This study directly optimizes the feedback controller, bypassing
13 this problem.

14 Secondly, this study strengthens our belief that compliant robots should or can be seen
15 as computational units, instead of dumb hardware that needs a complex controller. This link
16 between compliant robotics and neural networks is also the main reason for our search for
17 simple universal learning rules for both neural networks and robotics.

18 **Keywords: distal reward, tensegrity, compliant robotics, Hebbian plasticity, recurrent neural networks, morphological computation**

1 INTRODUCTION

19 Hebbian theory has been around for over half a century **Hebb** (1949), but it still sparks the interest of
20 today's researchers. Small changes to the basic correlation learning rule result in various well known
21 algorithms, such as principal **Oja** (1982); **Sanger** (1989) or independent component **Clopath et al.**
22 (2008); **Hyvrinen and Oja** (2000) extractor networks. The basic rule is biologically plausible as are some
23 of its variations **Mazzoni et al.** (1991); **Loewenstein and Seung** (2006). Whereas all these approaches
24 belong to the general category of unsupervised learning approaches, *reward modulated Hebbian* (RMH)
25 learning is similar to reinforcement learning in that it can be used to tune a neural system to solve a
26 specific task without the need for the desired output signals at the neural level **Hoerzer et al.** (2012);

Fiete and Seung (2006); **Legenstein et al.** (2010); **Soltoggio and Steil** (2013). When using RMH learning in a robotics context, a reward can, e.g., be computed by comparing the sensory inputs with the desired observations. The use of RMH learning for optimizing robot motor control has several additional advantages. Firstly, the basic learning rule is simple. There is no need for complex mathematical operations and it can therefore be efficiently implemented on various platforms in hardware and software. Secondly, it allows for a distributed implementation: a central unit can be responsible for a global reward, which can then be broadcast to the learning units of local controllers. Finally, RMH learning is an online learning approach. If the reward mechanism remains active, the controller can adapt to changes in the robot morphology or dynamics, e.g., due to wear or damage.

Class one tensegrity structures **Skelton and de Oliveira** (2009); **Caluwaerts et al.** (2014) consist of compression members held together by tension members in such a way that compression members are never directly connected. This results in flexible pin-jointed structures that can make efficient use of materials. Tensegrities have been researched from various perspectives, from architecture and art **Snelson** (1965) over mathematics **Connelly and Back** (1998) to biology **Ingber** (1997).

In previous work (**Caluwaerts et al.** (2012)), we demonstrated that the motor control of a tensegrity robot can be drastically simplified by using its body as a computational resource. This approach originated from the concepts of *physical reservoir computing* **Verstraeten et al.** (2007) and *morphological computation* **Pfeifer and Bongard** (2007), both of which treat the use of physical systems or *bodies* as a computational resource in so-called embodied computation. In **Caluwaerts et al.** (2012), we mainly focused on approximating motor signals through a single layer linear neural network acting as a feedback controller. The flexibility and lack of joints of our tensegrity robot allowed for simple learning rules, as the risk of failure due to mechanical stress or hard constraints is minimal. As a consequence, the feedback weights were learned by applying online supervised learning rules to approximate the target motor signals, among which a supervised version of RMH learning. However, the target motor signals themselves were generated using genetic algorithms.

We now extend this work by incorporating a similar feedback controller into a two-level control hierarchy for end-effector control in a highly compliant class one tensegrity robot. The primary controller, a simple feed forward kinematic controller, generates control signals derived from a very rough static inverse model of the relationship between end effector positions and actuator signals. The secondary embodied controller, consisting only of the robot body and 'neural' linear feedback weights, handles the dynamics, i.e., it tunes the primary control signals to result in smooth and stable trajectories. In this task, only the desired end-effector trajectories are known, not the control signals required to generate them. We use RMH learning to train the feedback weights in the secondary controller, demonstrating that a biologically plausible learning rule can indeed be used to control a compliant system. For this purpose, we first use traditional neural network models to show how RMH learning can be extended to neural systems with partial embodiment. This leads to a version of the learning rule tailored to robotics applications in which the number of observable variables is limited compared to neural networks.

2 METHODS

In this section, we introduce the basic learning rule used throughout the paper, we derive a stabilized version of this rule, and we discuss additional changes to the rule to make it more suitable for the targeted application in partially embodied control of a tensegrity robot.

Throughout this work, we will employ the term *observations* instead of state or network activity, because the reward can be based on a general performance criterion of the neural network state or the robot behavior.

2.1 HEBBIAN LEARNING IN ANALOG RECURRENT NEURAL NETWORKS

Hebbian plasticity is a biologically plausible learning methodology for neural networks. A learning rule is called Hebbian if it modifies the weights between a set of presynaptic neurons $\boldsymbol{x}$ and postsynaptic neurons $\boldsymbol{y}$ as a function of their joint activity. Although, Hebb did not provide a precise mathematical formulation of his postulate, a relatively general form can be written as **Hebb** (1949):

$$\boldsymbol{\Delta W}_{\text{Hebb}} \quad = \quad f(\boldsymbol{X}, \boldsymbol{Y}). \tag{1}$$

Note that we have used capital $\boldsymbol{X}$ and $\boldsymbol{Y}$[1] to indicate that the weight updates in the learning rule can depend on multiple time steps, i.e., the history of the pre- and postsynaptic neuron activations. As we are using fully connnected recurrent neural networks, the sets of presynaptic and postsynaptic neurons are the same, shifted by a single time step, i.e. $\boldsymbol{y}[k] = \boldsymbol{x}[k+1]$.

To apply Hebbian theory in a reinforcement learning setting, we have to introduce the notion of a reward r into the learning rule. Indeed, reinforcement learning aims at making behavior that optimizes the reward more likely to happen. However, learning new behaviors necessitates another tool, namely exploration.

We use noise injected at the postsynaptic neurons for exploration. Noise causes variations $\delta\boldsymbol{y}$ of the postsynaptic neurons, which are essentially drifts from the expected value $\bar{\boldsymbol{y}}$. We therefore introduce the exploration noise variable $\boldsymbol{z} = \delta\boldsymbol{y} = (\boldsymbol{y} - \bar{\boldsymbol{y}})$.

If the exploratory noise causes an improvement in behavior, this will result in a higher reward (and vice versa). A basic learning rule based on this idea is:

$$\boldsymbol{\Delta W} \quad = \quad r\boldsymbol{z}\boldsymbol{x}^T. \tag{2}$$

This rule, however, suffers from a number of basic flaws to be able to solve distal reward problems. First, we need to add some efficient notion of memory of the relationship between exploration noise and the presynaptic neurons. This can be achieved by estimating the covariance between the exploration and the states of the presynaptic neurons. Secondly, we note that in its current form, any significant bias of the reward r will cause unfavorable results. The solution to this is to predict the reward and subtract this from the obtained reward, resulting in a learning rule of the form:

$$\boldsymbol{\Delta W} \quad = \quad \alpha(r - \bar{r})\boldsymbol{Z}^T\boldsymbol{X}, \tag{3}$$

in which $\bar{r}$ is the predicted reward and where we have added a learning rate parameter α.

The predicted reward is sometimes ambiguously referred to as the (short term) average reward. More precisely, it is the average (or expected) reward when noise is present in the system. As we will demonstrate, the average reward is typically highly dependent on the noise level of the system. The learning rule therefore optimizes the expected reward while noise is present in the system (i.e. $\max \mathbb{E}[r|\boldsymbol{z}]$), under the assumption that this also optimizes the performance when the exploration noise is removed (i.e. $\max \mathbb{E}[r|\boldsymbol{z} = \boldsymbol{0}]$).

In this work, we apply the learning rule on a trial-to-trial basis, by evaluating the sample covariance $\boldsymbol{X}^T\boldsymbol{Z}$ of a number of samples and then applying the learning rule once. It can be seen that - in case the reward signal can be computed at every time step - one could apply the algorithm at every time step, similar to the Rare Correlation learning rule of **Soltoggio and Steil** (2013).

2.2 A STABILIZED REWARD MODULATED HEBBIAN RULE

Oja's rule **Oja** (1982) and its extension, the Generalized Hebbian Algorithm or Sanger's rule **Sanger** (1989), provide a single layer neural network implementation to compute principal components. Contrary

[1] We use the notation x to denote a scalar, $\boldsymbol{x}$ for a vector, and $\boldsymbol{X}$ for a matrix. In general $\boldsymbol{x}_i$ is the ith row of $\boldsymbol{X}$ and x_i is the ith element of $\boldsymbol{x}$.

to pure Hebbian plasticity, the learning rules are stable, because they force the norm of the weight vectors to unity. Unlike in the unsupervised learning case, reward modulated rules tend to be stable in practice (i.e. the trained weights remain bounded). However, it can still be useful to control the norm of the weights as this can have practical implications. For example, in a robotics application, this would allow to limit the required feedback gain and thus the required motor power. From a theoretical point of view it is also instructive to see how the learning rules from the previous sections resemble the now classic rule discovered by Sanger over 20 years ago. In this section, we provide a similar derivation for the RMH learning rule we studied in this work.

To simplify the notation, we start by defining a number of variables:

$$\boldsymbol{E} = \boldsymbol{Z}^T \boldsymbol{X} \tag{4}$$

$$r' = r - \bar{r}. \tag{5}$$

The basic learning rule we studied can now be written in element-wise form as:

$$w_{ij}[n+1] = w_{ij}[n] + \alpha r' e_{ij} \tag{6}$$

Oja's rule is a first order approximation to the normalization of the weights at every update step:

$$w_{ij}[n+1] = b_i \frac{w_{ij}[n] + \alpha r' e_{ij}}{\|\boldsymbol{w}_i[n] + \alpha r' \boldsymbol{e}_i\|}, \tag{7}$$

where $\boldsymbol{b}$ contains the desired L_2 norms of the weight vectors.

We now consider the linearization of this rule for small learning rates α. To further simplify the notation, we drop the time index and consider a single output dimension:

$$w_j \approx b \frac{w_j + \alpha r e_j}{\|\boldsymbol{w} + \alpha r \boldsymbol{e}\|}\bigg|_{\alpha=0} + \alpha b \left(\frac{\partial}{\partial \alpha} \frac{w_j + \alpha r e_j}{\|\boldsymbol{w} + \alpha r \boldsymbol{e}\|} \right) \bigg|_{\alpha=0} \tag{8}$$

A straightforward calculation shows that the part inside the parentheses of the second term can be written as:

$$\frac{\partial}{\partial \alpha} \frac{w_j + \alpha r e_j}{\|\boldsymbol{w} + \alpha r \boldsymbol{e}\|}\bigg|_{\alpha=0, \|\boldsymbol{w}\|=b} = \frac{1}{b} r e_j - \left(\frac{r \boldsymbol{w} \cdot \boldsymbol{e}}{b^3} \right) w_j, \tag{9}$$

assuming $||\boldsymbol{w}|| = b$ and $\alpha = 0$.

The complete learning rule can therefore be written as:

$$w_j = b \frac{w_j}{b} + \alpha b \left(\frac{1}{b} r e_j - \left(\frac{r \boldsymbol{w} \cdot \boldsymbol{e}}{b^3} \right) w_j \right) \tag{10}$$

$$= w_j + \alpha r \left(e_i - \frac{w_j}{b^2} \boldsymbol{w} \cdot \boldsymbol{e} \right). \tag{11}$$

The matrix form of the rule for multiple outputs is given by:

$$\boldsymbol{\Delta W} = \alpha r' (\boldsymbol{E} - \mathrm{diag_v}(\boldsymbol{b})^{-2} \mathrm{diag_v}(\mathrm{diag_m}(\boldsymbol{E}\boldsymbol{W}^T))\boldsymbol{W}), \tag{12}$$

where the $\mathrm{diag_v}$ operator transforms a vector into a diagonal matrix, and the $\mathrm{diag_m}$ operator transforms a matrix into a vector, containing its diagonal elements. The time complexity to this rule is of the order $\mathcal{O}(mn)$, with m the number of outputs and n the number of inputs. The stabilized learning rule therefore takes the same form as Sanger's rule.

2.3 DECORRELATED LEARNING RULE FOR ROBOTICS EXPERIMENTS

In partially embodied control, the dynamics of the robot body are used directly as a computational resource. In our RMH learning setup, this is equivalent to replacing part of the neural network by the robot body, which receives inputs from the remaining neurons and the observations of which are fed back into those neurons. The training procedure is now heavily constrained, as it can only adapt synaptic weights of the remaining neurons, whereas the part of the network that is replaced by the body remains unchanged.

Although this situation is similar to RMH learning in neural networks, it differs in the fact that most of the physical state of the robot remains hidden to the observer and the number of observable signals that can be fed into the trainable neural network is relatively small. Furthermore, the dynamics of the observed variables tend to be highly correlated. For example, stiffening the structure typically causes an increase in all sensor values.

As RMH learning relies on the varying influence of exploration noise on the observed variables. A common influence (increase or decrease) of the noise on all observed variables therefore reduces the effectiveness of the learning rule. A simple approach to overcome this issue is to decorrelate the observations. In (**Caluwaerts et al.**, 2012, Appendix I), we showed that a decorrelation layer that uses Sanger's rule offers a biologically plausible solution for this. In this work, we take a more pragmatic approach and decorrelate on a trial-by-trial basis.

The resulting learning rule is:

$$\boldsymbol{\Delta W}^T \quad = \quad \alpha H(r-\bar{r})(\boldsymbol{X}^T\boldsymbol{X}+\lambda \boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{Z}, \tag{13}$$

where $H(\cdot)$ represents the Heaviside step function. In this variant of the learning rule, weight updates only occur when the observed reward is better than expected. This is not strictly necessary, but we found that it slightly improved our results.

This learning rule is similar to ridge regression, the difference being that the algorithm will try to reproduce the noise $\boldsymbol{Z}$ (instead of a desired output) proportionally to the relative performance with the injected noise w.r.t. the expected reward. We used a large regularization parameter ($\lambda = 1$) which results in only a minor decorrelation of the sensor variables, yet is enough to allow for efficient learning. A high regularization parameter allows for simple covariance estimators, in case a more biologically plausible version of the rule is desired.

3 EXPERIMENTS

3.1 NEURAL NETWORK EXPERIMENTAL SETUP

Before presenting our results in a robotics context, we first study RMH plasticity in discrete time recurrent neural networks with hyperbolic tangent activation functions. They receive input, which we denote $\boldsymbol{U}$ and a readout function provides observations of the network state. Additionally, exploration noise $\boldsymbol{Z}$ is injected into the network. The network update equation is given by:

$$\boldsymbol{x}[k+1] \quad = \quad \tanh(\boldsymbol{W}\boldsymbol{x}[k]+\boldsymbol{W}_{\text{in}}\boldsymbol{u}[k+1]+\boldsymbol{z}[k+1]). \tag{14}$$

For each of our neural network experiments, the network is initialized according to the reservoir computing approach **Verstraeten et al.** (2007). This implies that we initialized the weights randomly (i.i.d. standard normally distributed samples) and then tune the network dynamics in a useful regime. This is achieved by rescaling the weight matrix $\boldsymbol{W}$ such that its spectral radius[2] is such that the learning

[2] The spectral radius is the largest amongst the absolute values of the eigenvalues of $\boldsymbol{W}$.

converges. For the experiments we will describe, we obtained good performance for initial spectral radii in $[0.95,\ 1.2]$, i.e., stable or almost stable networks. This differs from related approaches such as **Legenstein et al.** (2010) and **Hoerzer et al.** (2012), where initially chaotic networks are used. The input weight matrix $\boldsymbol{W}_{\text{in}}$ was sparsely initialized (20% non-zero elements) with i.i.d. normally distributed values with standard deviation 0.05. All networks contained 100 neurons.

Fig. 1 shows our learning setup for neural networks. The neural network to be trained is the central element. A reward is provided after a trial based on the network observations. A trial is defined as a number of time steps which corresponds to a period of time in which the network tries to perform a task of interest. In parallel to the network, a reward prediction system estimates the expected reward based on the network inputs. The RMH learning rule finally combines information from the network state, exploration noise, reward and estimated expected reward to compute an update $\boldsymbol{\Delta W}$ of the network weights.

3.2 NEURAL NETWORK EXPERIMENTS

The networks used for the 3 tasks described below only differ in the way observed outputs are generated and in the subset of weights that can be modified by the learning rule. In the first two networks, two neurons are selected as output-generating neurons, and the output is computed as the sum of their states. The third network has three output-generating neurons and has an output equal to the product of these neurons' states.

In all networks, the weights to and from the observed neurons are fixed and recurrent. This prevents the learning algorithm from generating solutions in which the observation neurons become a pure output layer, which does not influence the state of the rest of the network. In the networks for tasks 1 and 3, all other internal weights are modifiable, i.e., there are 98^2 and 97^2 trainable connections in tasks 1 and 3, respectively. In task two, the training is further restricted by fixing the input weights for half of the remaining neurons, i.e., there are only $49\,(48+49)$ adjustable weights. This means that about half of the network is a random recurrent neural network. In a neural or neurorobotics context, we can see this as a rudimentary model for a trainable network interacting with an untrained dynamical system, such as another brain area or a physical body, e.g., the partially embodied control of a robot arm with a neural network.

Note that having a learning rule that handles delayed rewards is crucial, as the input does not directly, nor immediately define the observations.

In what follows, we describe the three neural network tasks in more detail. We first consider problems with discrete input spaces. More precisely, we solve the 2 bit delayed XOR problem and a 3 bit delayed classification task. Our third example has a continuous input space and a more complex readout function.

3.2.1 Task 1: proof-of-principle

Inputs The input signal for this task represents a single bit stream. A zero bit is coded as the negative half period of a sine wave, a one bit as the positive half (see the top row of Fig. 7). For each trial, we randomly select one of the 4 possible 2-bit input sequences.

Desired output The neural network has to compute the so-called 2-bit delayed XOR task, i.e., the exclusive OR function applied to the last two bits of its input stream, represented as binary values. More concretely, the output of the network should be as close to plus one or minus one during the last half of the second bit.

The XOR task is a common test or benchmark, because the inputs are not linearly separable. A linear network cannot obtain optimal performance for all inputs simultaneously. Therefore, this task is a simple test to verify if the learning rule can exploit the non-linear effects of the network. The task also requires

the network to remember a specific part of the input, while ignoring inputs that occurred more than one bit length in the past.

Reward function Throughout this manuscript, we use different reward functions. The main reasons for this is that some reward functions are more appropriate for a specific task and to show that the learning rule does not depend on a specific reward function (e.g. square loss). For the results presented for the 2-bit XOR task, we used minus the mean squared hinge loss, as the hinge loss is a more appropriate reward function for a binary classification task:

$$r_n^{\text{2bit}} = \frac{-1}{5}\sum_{k=15}^{19}\max\left(0,\, 1 - t\left[k\right]\left(x_0^{\text{o}}[20n+k] + x_1^{\text{o}}[20n+k]\right)\right)^2, \tag{15}$$

where n indicates the number of the current trial, $\boldsymbol{x}^{\mathbf{o}}$ are the neurons that generate the observations and $t[k]$ are the desired observations. However, we have also successfully applied the mean squared error reward function for this task.

Prediction of the expected reward Estimating the expected reward is trivial in the case of a modest number of different inputs. For the results presented here, we averaged the last 50 rewards per input sequence.

3.2.2 Task 2: partially embodied computation

Inputs For this task, the input is the same as for the previous task. For each trial, we now randomly select one of 8 possible 3-bit input sequences.

Desired outputs The network now has to act as a 3-bit digital-to-analog decoder, i.e., it has to produce one of 8 equidistant analog values in the range $[-1, 1]$, corresponding to the decimal interpretation of the last three encoded bits it received. Similar to the previous task, the desired value has to be present on the output during the second half of the third bit.

This task is more complex and nonlinear than the previous one and it requires more memory as well. The target values are indicated by red crosses in the bottom row of Fig. 7.

Reward function For the 3 bit task, the reward value r^{3bit} is defined as minus the mean squared error of the observations during the last five time steps:

$$r_n^{\text{3bit}} = \frac{-1}{5}\sum_{k=25}^{29}\left(t\left[k\right] - x_0^{\text{o}}\left[30n+k\right] - x_1^{\text{o}}[30n+k]\right)^2. \tag{16}$$

Prediction of the expected reward The rewards were estimated in the same way as in the previous task.

3.2.3 Task 3: nonlinear observation function

The task at hand is to reproduce part of the input in reverse after a delay (see Figure 3).

Inputs The network receives two input signals. The first input signal (see example in Figure 3) consists of sequences of 12 time steps per trial. The first 5 of these steps form a linear segment between two values, which are sampled with uniform probability form $[0,\, 1]$ for each trail. The final value is held constant for two more time steps. The final 5 time steps again form a linear segment that is obtained by connecting the last value from the first 5 steps with a third random sample from $[0,\, 1]$.

Because the trials are fed into the system one by one, it is not clear to the network when a trial starts. The second input, a binary signal, is used to inform the network when it has to generate the desired output.

Desired outputs The network must learn to recall the input during the first 5 steps and reproduce them in reverse order during the last 5 steps of the trial. The system must ignore the remaining 7 input samples.

Reward function The reward function used here is minus the mean absolute error of the observations:

$$r_n^{\text{cont}} = \frac{-1}{5}\sum_{k=0}^{4}|u\left[12n+k\right] - \prod_{j=0}^{2} x_j^{\text{o}}\left[12n+11-k\right]|. \tag{17}$$

Neural network structure The complete network structure is shown in Fig. 2(right). The observations are computed by multiplying the states of three internal neurons, which have fixed (i.e., untrained) incoming and outgoing connections.

Prediction of the expected reward The expected reward $\bar{r}$ estimates the performance of the system given the noise level σ. Furthermore, the reward is input dependent, therefore $\bar{r}$ estimates the following quantity:

$$\bar{r} = \mathbb{E}[r|\boldsymbol{u}, \sigma]. \tag{18}$$

Various algorithms can be used to estimate this quantity. We employed the well known Recursive Least Squares algorithm **Kailath et al.** (2000) to learn a simple online estimator of this quantity, which we applied to the input sequences $\boldsymbol{u}$ and the network state $\boldsymbol{x}$ at the end of a trial.

3.3 THE TENSEGRITY ROBOT

The general setup of our tensegrity robot control problem is shown in Figure 4. It is similar to the neural network setup represented in Fig. 1, but the recurrent neural network has been replaced with the tensegrity robot.

The tensegrity structure used for our experiments has 4 struts and is shown in Figure 5. It is based on the standard 3 strut tensegrity prism to which a shorter rod has been added that acts as a compliant end-effector. The bottom 3 nodes of the original prism have been fixed through ball-joints. The resulting structure has 17 $k = 20\,\text{N}\,\text{m}^{-1}$ springs, 14 of which are actuated (the length of the 3 bottom springs is fixed). The controller time step was $50\,\text{ms}$ and gravity was not modeled.

Instead of observing the state of the neurons, we measure the spring forces:

$$x_i = f_i \tag{19}$$

$$= \max(k(l_i - l_i^0), 0). \tag{20}$$

Actuators changing the equilibrium lengths of the springs replace the postsynaptic neurons and motor babbling takes on the role of the exploration noise:

$$\boldsymbol{l}^0 = \boldsymbol{l}^{\text{init}} + \boldsymbol{W}\boldsymbol{x} + \boldsymbol{u} + \boldsymbol{z}. \tag{21}$$

All tensegrity experiments were performed in our tensegrity simulator which is based on an Euler-Lagrange formulation of the tensegrity dynamics (**Skelton and de Oliveira**, 2009, Chapter 5). For more details on the simulation setup, we refer to **Caluwaerts et al.** (2012).

3.4 HIERARCHICAL END-EFFECTOR CONTROL FOR THE TENSEGRITY ROBOT

The task we consider is writing characters with the top node of the rod suspended in the tensegrity structure. More precisely, the node has to trace letters in a horizontal (XY) plane. The characters

were taken from UCI Character Trajectories data set **Bache and Lichman** (2013), integrated and then subsampled and rescaled.

The robot is controlled by combining a feed forward kinematic controller and a learned static linear feedback controller. The kinematic controller provides the input signals $\boldsymbol{u}$ in Equation 21. We sampled 100 random spring lengths to create a set of configurations for the kinematic controller. To write a character, the kinematic controller selects a combination of spring lengths that move the end-effector as close as possible to the desired position when the structure is in equilibrium.

The reward function used for the next experiments tries to bring the end-effector close to the desired trajectory:

$$r^{\text{traj}} = \frac{-1}{s}\sum_{k=0}^{s-1}\max(\|\boldsymbol{n}[k]-\boldsymbol{c}[k]\|-0.01,\,0), \tag{22}$$

where s is the number of steps of the current character, $\boldsymbol{c}[k]$ the vector containing the target position at time k (relative to the beginning of the trial) and $\boldsymbol{n}[k]$ the position in the XY plane of the end-effector at time k. This reward function will cause the learning rule to stop improving a feedback controller w.r.t. a point on the trajectory in case the end-effector is within 1cm of the target position.

3.5 ROBUSTNESS AGAINST FAILURES

To demonstrate the robustness of the controllers as well as a more practical application of the learning rule, we simulated various actuator failures. In this case, we used a more realistic feed forward controller. Again starting from a simple kinematic controller, we now optimized the inputs $\boldsymbol{u}$ in Equation (21) at each time step using a basic exploration method. More precisely, a small amount of noise $\boldsymbol{z}$ is injected at each time step and the change in expected reward is observed. If an improvement of the expected reward is observed after a trial, we reproduce the noise in the feed forward controller $\boldsymbol{u} = \boldsymbol{u}_{kin} + \boldsymbol{u}_{expl}$ by using $\Delta \boldsymbol{U}_{expl} = \boldsymbol{Z}$ when the expected reward (of the trial) improved and $\Delta \boldsymbol{U}_{expl} = 0$ when it did not. In the previous equations, $\boldsymbol{u}_{kin}$ refers to the original kinematic controller discussed in the previous section.

In this setup, we now simulate actuator failures by resetting one or more actuators to the original kinematic controller instead of the optimized ones, i.e. $\boldsymbol{u} = \boldsymbol{u}_{kin}$.

4 RESULTS

4.1 NEURAL NETWORK EXPERIMENTS

We first demonstrate on an academic benchmark task (task 1) that the proposed learning rule can be successfully applied to neural networks. The left panel of Figure 6 shows the evolution of the reward and the spectral radius while the system learns the 2 bit delayed XOR task. The noise level was $\sigma = 0.01$ and the learning rate was $\alpha = 0.05$. We observe that the network indeed learns to solve the task almost perfectly, as the reward converges to its maximal value of 0.

In this experiment, we set the initial spectral radius to 0.95 to show how the spectral radius evolves as the average reward increases. Every run of the algorithm for different initial random weights resulted in a final spectral radius of approximately 1.2, which indicates that the learning rule indeed tunes the memory of network. The right panel of Fig. 6 shows a detailed view of the reward and expected reward for a single input sequence.

For the particular run shown in this Figure, we see a drop in performance after about $125\,000$ trials. This is probably a significant change in network behavior (a bifurcation) due to a minor weight change, a

typical phenomenon in non-linear recurrent neural networks. However, the algorithm continues learning successfully afterwards.

Next, we address task 2, which is similar to task 1 (the classification of sequentially offered bit patterns), but more difficult, as the sequences now consist of 3 bits. However, in this case the learning rule is only allowed to modify half of the internal weights of the recurrent neural network. The network structure can be considered as a model for partially embodied computation, i.e., we replace part of the original trainable network by a fixed one, which acts as a dummy for a physical body.

The results presented here show that, again, the learning rule is successful. Indeed, Figure 7 shows the results of overlaying 50 random orderings of the input sequences. Note that the classification result must only be available at the output during 5 time steps at the end of each trail (indicated by red crosses in the figure). As trails follow each other continuously, the variability of the state trajectories is due to the different initial states. Clearly, the network has learned the correct time window, as the classification result is available at the right time and only depends on the two previous bits and not on older inputs (the random inputs were fed into the system sequentially). We can identify a number of separate trajectories that keep track of the possible outcomes. It can be seen that within a trial, each additional bit that is offered at the input reduces the number of possible states of the system by half. Interestingly, the two neurons that generate the observations have different state trajectories, because the learning rule only quantifies the performance based on the observations, without directly enforcing a specific behavior of the neurons responsible for the observations.

Whereas the network was trained without any noise on the input signals, the resulting behaviour is robust against such noise. In Figure 8, we show the behavior of the network when input noise is present. This plot was generated by first applying k-means clustering on the trajectories and then estimating the variance of each centroid. Shown are the various centroids and the standard deviation of each. We see that the network is robust against high amounts of noise on the input data (σ up to 0.5), as the original trajectories are maintained.

Finally, we move to the last and most elaborate task. Figure 9 visualises the rewards during testing for various state noise levels, using $100\,000$ random input trials per noise level. The noise was added to the internal neurons of the network, but not to the 3 neurons which generate the observations. The network shown was first trained for $1\,000\,000$ time steps with an exploration noise level $\sigma = 0.05$ and a learning rate of 0.005. Each graph in the top panel shows the average rewards of the trained networks, across the whole spectrum of possible input sequences for a given level of input noise (increasing from left to right). In the top left graph, we see that without noise, the average reward remains close to its optimal value of 0 for most input patterns, although some regions of the input pattern space seem to be slightly more difficult. This demonstrates that the learning rule also works, although less perfectly than in the previous cases, when the relation between the internal states in the network and the way they are translated into actions and rewards is highly nonlinear and when the input patterns do not fall into discrete categories.

The bottom panel shows the reward distribution, averaged across the different input trials, for each noise level. It becomes clear that $\bar{r}$ is indeed the expected reward under the influence of noise for the given input sequence. The expected reward shifts to the left as a function of the amount of noise in the network. The learning rule evaluates the performance with respect to the expected reward for the current noise level.

4.2 TENSEGRITY EXPERIMENTS

The left panel of Figure 10 shows how the tensegrity robot performs when drawing characters between 20 and 68 time steps long (1s to 3.4s). A different set of feedback weights was learned for each character, therefore it is easy to predict the expected reward. To clarify, we estimated the expected reward by averaging the rewards obtained during the previous 30 trials. As can be seen from the top row, the initial performance of the system with only the kinematic controller is very low, whereas the combination of both controllers, using our RMH learning rule, performs considerably better.

The plot on the right of Fig. 10 shows the learning curves for each character, indicating that for most characters good results were obtained after 1000 to 1500 trials, which would be equivalent to less than one hour real robot time for most characters. It is possible to accelerate learning by further tuning the learning parameters. We used a conservative level of exploration noise ($\sigma = 5\,\mathrm{mm}$) and a learning rate $\alpha = 1$, which consistently resulted in stable feedback controllers. The learning rule did not achieve the same final reward for all characters (e.g. the "m"). This is due to physical limitations enforced on the motor commands.

Finally, we simulate actuator failures. The results of these experiments are presented in Fig. 11. As could be expected, the performance immediately drops significantly after each failure. By applying the RMH learning rule to the feedback controller, the system is able to recover from the various failures. To investigate the stability of the learning rule, each experiment was performed 30 times, with similar results.

5 DISCUSSION

Hebbian theory is a well-established approach to explain synaptic plasticity between neurons. Over the years, many variations of the basic learning rule have been developed. Each of these had a specific application, ranging from unsupervised feature extraction to reinforcement learning. In this work, we presented Hebbian like learning rules in which the synaptic plasticity is based on the correlation of the presynaptic neurons and an exploratory noise signal. The plasticity was modulated by a distal reward signal, resulting in a learning method that maximizes the expected reward of a trial.

While similar rules have already been presented, we focused on distal reward learning in recurrent neural networks and compliant robots. The rationale for this is our belief that both can be seen as computational resources and can therefore benefit from similar learning techniques. We had already shown this before, but in this work we also presented a method to efficiently train the substrates.

Our work builds upon Legenstein's **Legenstein et al.** (2010), who considered simulated motor control tasks in combination with an instantaneous reward signal in an initially chaotic neural network. One significant difference with respect to our experimental setup is that Legenstein estimated the exploration noise as well as the expected reward. This allows for uncontrolled or unknown noise sources to be used, which adds to the biological plausibility of the method learning **Faisal et al.** (2008). Covariance and noise based rules have a strong biological foundation **Loewenstein and Seung** (2006); **Loewenstein** (2008); **Soltani and Wang** (2006). For example, it is well known that neural networks in biology have intrinsic noise sources **Faisal et al.** (2008), which could be used for learning **Maass** (2014). While this type of noise can sometimes be measured by external means (e.g. voltage clamps), a plasticity rule within the biological substrate cannot generally observe the noise signals, hence the importance of the noise estimator in Legenstein's rule. In this work, we did not consider such a noise estimator, as significant uncontrolled noise sources seem unlikely in robotics. Finally, remark that Legenstein's learning rule extends various earlier techniques with similar mathematical formulations **Fiete and Seung** (2006); **Loewenstein** (2008); **Loewenstein and Seung** (2006).

Our study of RMH learning in neural networks shows that our version of the learning rule, using distal rewards, works for a wide range of conditions (trainability and observation functions) and for very different tasks. In addition, we have interpreted the network configuration of task 2, in which an entire subnetwork was not trainable, as a model for partially embodied computation. An interesting question in this case is whether learning in the trainable part of the neural network was fundamentally necessary in order to recognize the input patterns, or, in contrast, whether the necessary computations were already present in the network dynamics. In this case, as in a traditional reservoir computing setup, the learning only needed to provide a suitable mapping from the internal dynamics to the observation function.

The presented results show that after training, the system state evolves along a fixed number of highly robust trajectories. This phenomenon is not commonly observed in reservoirs without trained feedback weights, indicating that training half of the network at least provides feedback loops in addition to a

suitable observation function. However, whether an actual trainable neural network has added value on top of these functionalities is not clear from the current experimental results. A more detailed analysis of the learning outcomes in assemblies of fixed and trainable substrates, as a model for partially embodied computation, is the subject of ongoing work.

Given the promising results we obtained in our simulations and the limited number of assumptions we had to make to obtain successful results, this work paves the way towards more complex control hierarchies for robot motor control, in which each level refines the output of the previous one. In this context, too, our results raise some interesting questions, for instance, about the exact role of the, very poorly performing, kinematic controller in our experiments. In fact, the main goal of the kinematic controller is not to have the optimal performance, but rather to inject energy into the structure. In our previous work, we showed an example with instantaneous reward function in which we first trained a feedback controller with known target signals using recursive least squares, and then proceeded to learn additional feedback signals using a reward modulated Hebbian rule (**Caluwaerts et al.**, 2012, Section 5.1.3). The reason why an additional energy source is employed in both cases, is that it is hard to consistently learn pure feedback controllers with simple Hebbian like learning rules. A small change in a feedback weight can cause the system dynamics to fade out, which often results in instability. Therefore, an easy and efficient solution is to use an additional input that consistently pumps energy into the system. In principle, this can be accomplished using a feedback controller as in our previous work, a simple feed forward controller as we use here, or another controller such as a Central Pattern Generator. More extensive research is needed to determine how much of the workload can be offloaded to the lowest level, partially embodied feedback controller and how this scales to more complex control tasks, e.g. involving more complex robot bodies.

In summary, our main conclusion is that reward modulated Hebbian plasticity with distal rewards provides a simple, yet effective tool for bridging learning in recurrent neural networks and the exploitation of the own dynamics of compliant robots. This strengthens our belief that both the body and the neural network can be used as computational tools and that they should be combined in a self-organising way into partially embodied hierarchical controllers.

DISCLOSURE/CONFLICT-OF-INTEREST STATEMENT

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

AUTHOR CONTRIBUTIONS

K. Caluwaerts performed the experiments. All authors contributed to the interpretation and structuring of the results and to writing the paper.

ACKNOWLEDGEMENT

We would like to thank Andrea Soltoggio for useful discussions in the preparation of this letter.

Funding: This research was funded by a Ph.D. fellowship of the Research Foundation - Flanders (FWO) and the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreements No 248311 (AMARSi), and No 604102 (Human Brain Project).

REFERENCES

Bache, K. and Lichman, M. (2013), UCI machine learning repository

Caluwaerts, K., Despraz, J., Iscen, A., Sabelhaus, A., Bruce, J., Schrauwen, B., et al. (2014), Design and Control of Compliant Tensegrity Robots through Simulation and Hardware Validation, *Journal of the Royal Society Interface*, 98, 11

Caluwaerts, K., D'Haene, M., Verstraeten, D., and Schrauwen, B. (2012), Locomotion without a brain: Physical reservoir computing in tensegrity structures, *Artificial Life*, 19, 1, 35–66, doi:10.1162/ARTL_a_00080

Clopath, C., Longtin, A., and Gerstner, W. (2008), An online hebbian learning rule that performs independent component analysis, *BMC Neuroscience*, 9, Suppl 1, O13, doi:10.1186/1471-2202-9-S1-O13

Connelly, R. and Back, A. (1998), Mathematics and tensegrity, *American Scientist*, 86, 2, 142–151, doi:10.1511/1998.2.142

Faisal, A. A., Selen, L. P., and Wolpert, D. M. (2008), Noise in the nervous system, *Nature Reviews Neuroscience*, 9, 4, 292–303

Fiete, I. R. and Seung, H. S. (2006), Gradient learning in spiking neural networks by dynamic perturbation of conductances., *Physical Review Letters*, 97, 4, 5

Hebb, D. O. (1949), The organization of behavior (Wiley)

Hoerzer, G. M., Legenstein, R., and Maass, W. (2012), Emergence of complex computational structures from chaotic neural networks through reward-modulated hebbian learning, *Cerebral Cortex*, doi:10.1093/cercor/bhs348

Hyvrinen, A. and Oja, E. (2000), Independent component analysis: algorithms and applications, *Neural Networks*, 13, 45, 411 – 430, doi:http://dx.doi.org/10.1016/S0893-6080(00)00026-5

Ingber, D. E. (1997), Tensegrity: The architectural basis of cellular mechanotransduction., *Annual Review of Physiology*, 59, 1, 575–599

Kailath, T., Sayed, A. H., and Hassibi, B. (2000), Linear estimation, number 3 in Information and System Sciences (Prentice Hall)

Legenstein, R., Chase, S. M., Schwartz, A. B., and Maass, W. (2010), A reward-modulated Hebbian learning rule can explain experimentally observed network reorganization in a brain control task., *Journal of Neuroscience*, 30, 25, 8400–8410

Loewenstein, Y. (2008), Robustness of learning that is based on covariance-driven synaptic plasticity, *PLoS Computational Biology*, 4, 3, 10

Loewenstein, Y. and Seung, H. S. (2006), Operant matching is a generic outcome of synaptic plasticity based on the covariance between reward and neural activity, *Proceedings of the National Academy of Sciences*, 103, 41, 15224–15229, doi:10.1073/pnas.0505220103

Maass, W. (2014), Noise as a Resource for Computation and Learning in Networks of Spiking Neurons, *PROCEEDINGS OF THE IEEE*, 102, 5, SI, 860–880, doi:{10.1109/JPROC.2014.2310593}

Mazzoni, P., Andersen, R. A., and Jordan, M. I. (1991), A More Biologically Plausible Learning Rule for Neural Networks, *Proceedings of the National Academy of Sciences of the United States of America*, 88, 10, 4433–4437

Oja, E. (1982), A simplified neuron model as a principal component analyzer., *Journal of Mathematical Biology*, 15, 3, 267–273

Pfeifer, R. and Bongard, J. (2007), How the body shapes the way we think: A new view of intelligence (MIT Press)

Sanger, T. (1989), Optimal unsupervised learning in a single-layer linear feedforward neural network, *Neural Networks*, 2, 6, 459–473, doi:10.1016/0893-6080(89)90044-0

Skelton, R. E. and de Oliveira, M. C. (2009), Tensegrity systems (Springer)

Snelson, K. (1965), Continuous tension, discontinuous compression structures

Soltani, A. and Wang, X.-J. (2006), A biophysically based neural model of matching law behavior: melioration by stochastic synapses., *Journal of Neuroscience*, 26, 14, 3731–3744

Soltoggio, A. and Steil, J. (2013), Solving the distal reward problem with rare correlations, *Neural Computation*, 25, 978, doi:10.1162/NECO_a_00419

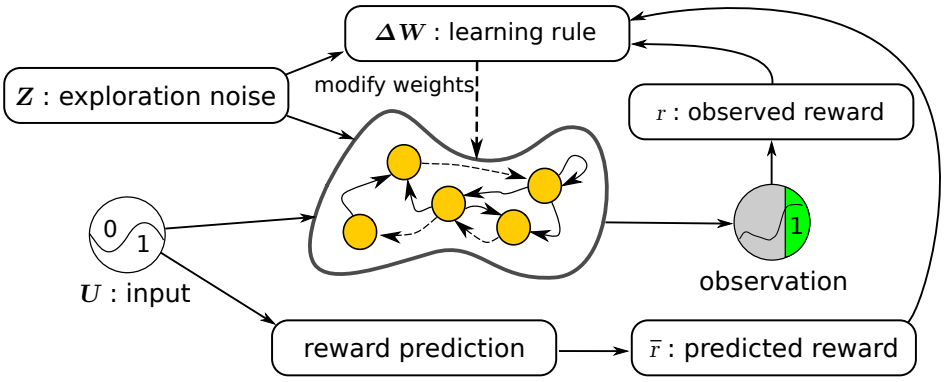


Figure 1. Overview of the learning setup for recurrent neural networks. The initially random recurrent neural network receives the inputs U and the exploration noise Z. The state of the postsynaptic neurons is computed by applying the hyperbolic tangent function to the sum of the inputs, the noise and the weighted sum of the presynaptic neurons. Observations are made of the state of the network and after every trial (fixed number of time steps), a reward is computed, based on the observations made during the last trial. In parallel, a simple reward prediction network predicts the expected reward for the given input. The learning rule then updates the weights between the presynaptic and postsynaptic neurons, by using the reward, the expected reward, the exploration noise and the states of the presynaptic neurons.

Verstraeten, D., Schrauwen, B., D'Haene, M., and Stroobandt, D. (2007), 2007 special issue: An experimental unification of reservoir computing methods, *Neural Networks*, 20, 3, 391–403

FIGURES

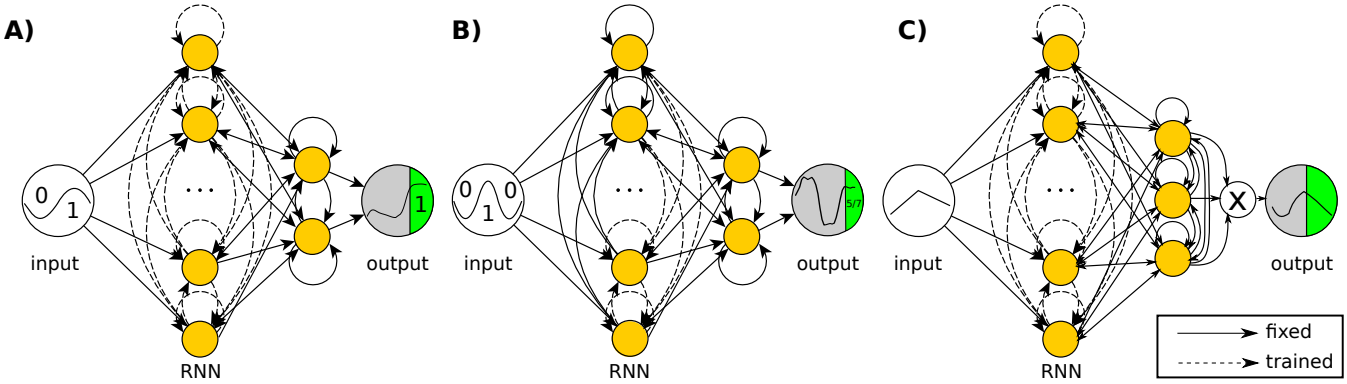


Figure 2. Network structures for the Recurrent Neural Network tasks. The networks are simulated in discrete time with hyperbolic tangent neurons (yellow nodes). Full lines are fixed connections, while dashed lines are trained.The reward is evaluated at the output neuron over the green period of time (one reward per trial).
(left) Task 1 (2-bit delayed XOR): the output equals the sum of two neurons (yellow nodes at the right), but only the internal connections can be modified by the algorithm. The RNN consisted of 100 neurons and the bit period was 10 time steps. (center) Task 2 (3 bit delayed classification): The output equals the sum of two neurons as in the XOR task, but now only half of the internal weights can be modified. The RNN consisted of 100 neurons and the bit period was 10 time steps. (right) continuous input task: the network has to reproduce the reversed input from the first 5 time steps of each trial. A trial consists of 12 time steps in total, of which the last 7 have to be ignored by the network.

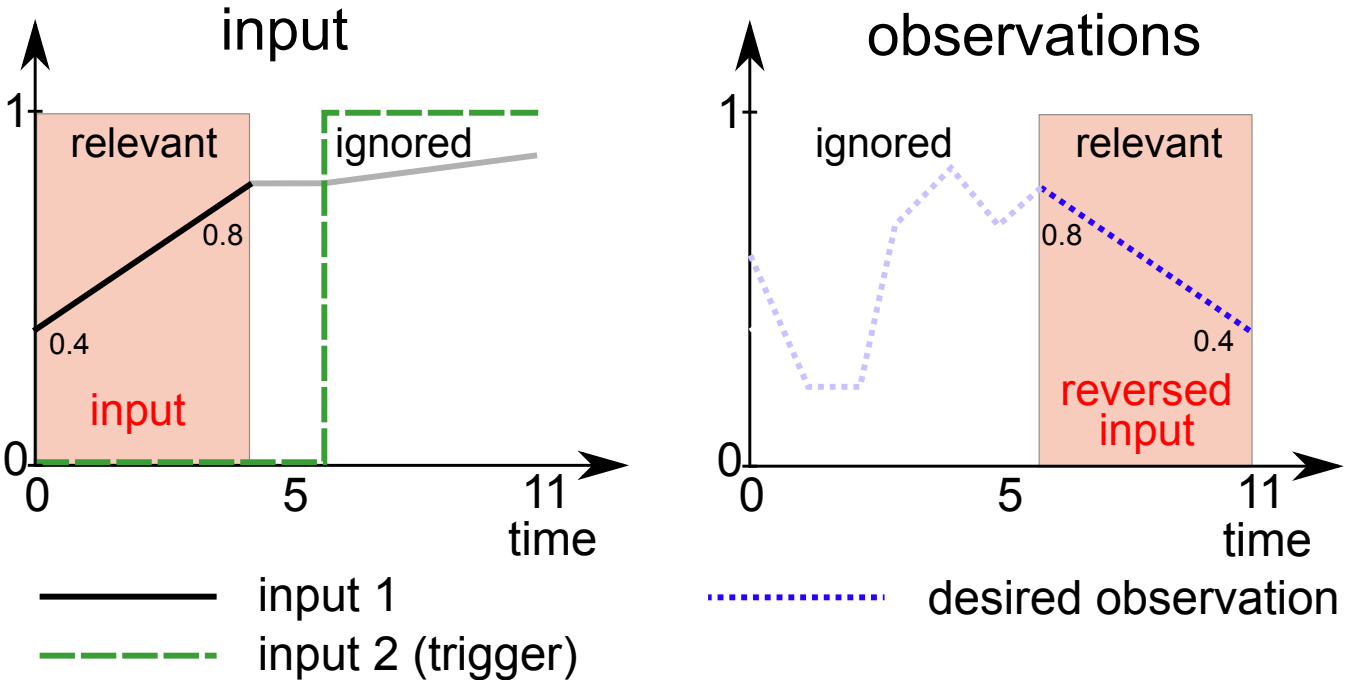


Figure 3. Overview of task 3: the network has to reproduce part of the first input (dashed black line) in reverse at the end of the trial (dashed blue line). More precisely, the first 5 steps of the first input (dashed line) are to be reproduced in reverse at the end of the trial (12 time steps total). The input space consists of all straight line originating and ending in $[0, 1]$. A second input indicates when the network has to start producing the desired observations.

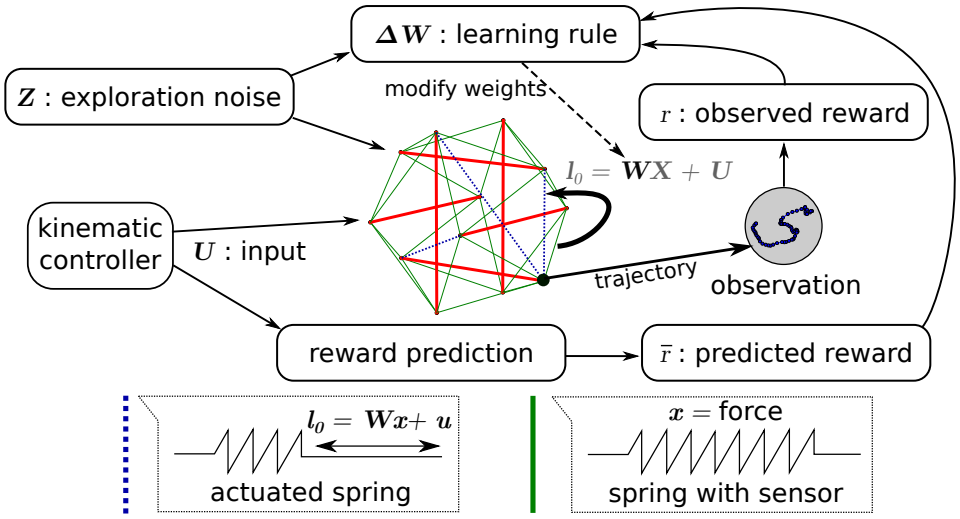


Figure 4. Overview of the way the learning rule is applied to compliant Tensegrity structures. The setup is similar to the recurrent neural network setup of Fig. 1. The neural network has been replaced by the combination of the compliant robot body and the neural linear feedback weights. It now receives input from the kinematic controller. Force sensors on the springs act as presynaptic neurons for the trained weights and the actuator signals correspond to the postsynaptic neurons. The learning rule adapts the feedback weights from the force sensors to the motor signals. The observations used for reward computation are based on the trajectories of an end-effector.

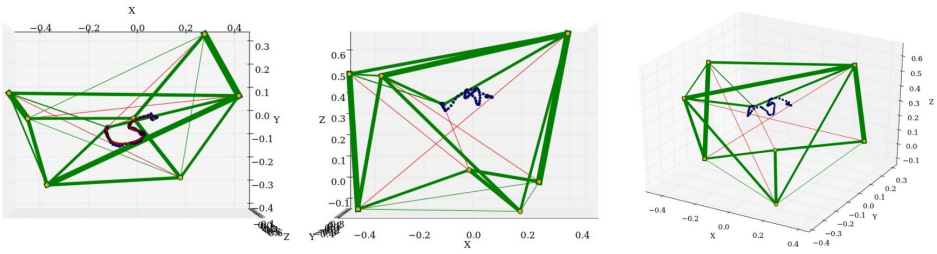


Figure 5. Tensegrity structure used for the experiments. The top node of the center rod is used as an end-effector to draw in the XY plane. In this example, the robot draws an ”S” as can be seen on the left. The center and right figures show other perspectives to demonstrate that the reward does not depend on the vertical position.

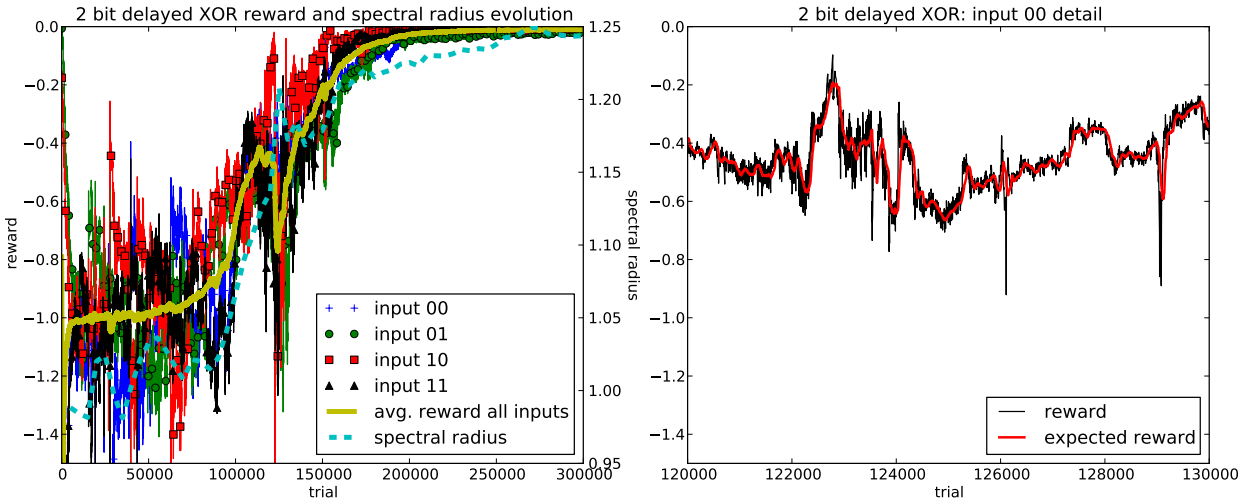


Figure 6.(Left) Evolution of the reward and the spectral radius for the 2 bit delayed XOR task and (Right) detail of the reward and expected reward for a single input sequence.

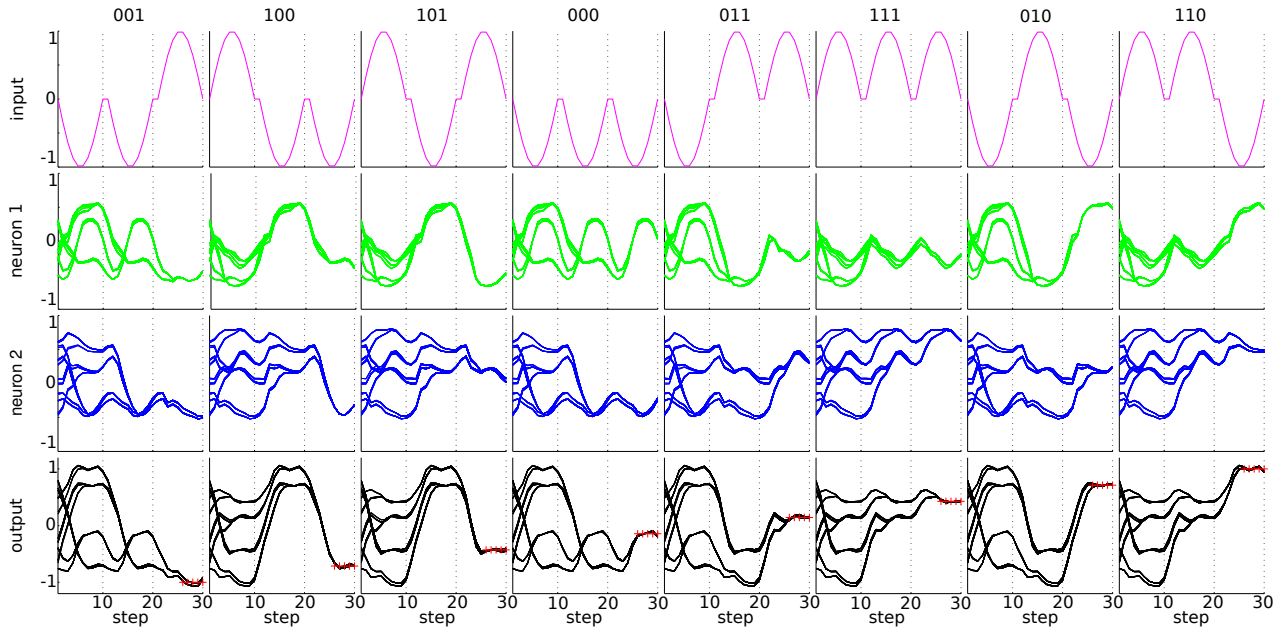


Figure 7. State trajectories for the 3 bit classification task. The top row shows the 8 input sequences. The next two rows show the state trajectories of the 2 neurons which are summed to compute the observations (see Fig. 2(center)) for the given input sequence of the top row. The bottom row shows the observations (the sum of the two middle rows). The desired observation at the end of the trial is indicated by red crosses. It can be seen that each additional bit reduces the number of possible states by half. Interestingly, the two neurons that generate the observations have different state trajectories, because the learning rule only quantifies the performance based on the observations, without directly enforcing a specific behavior of the neurons responsible for the observations. The plots were generated by overlaying 50 random orderings of the input sequences.

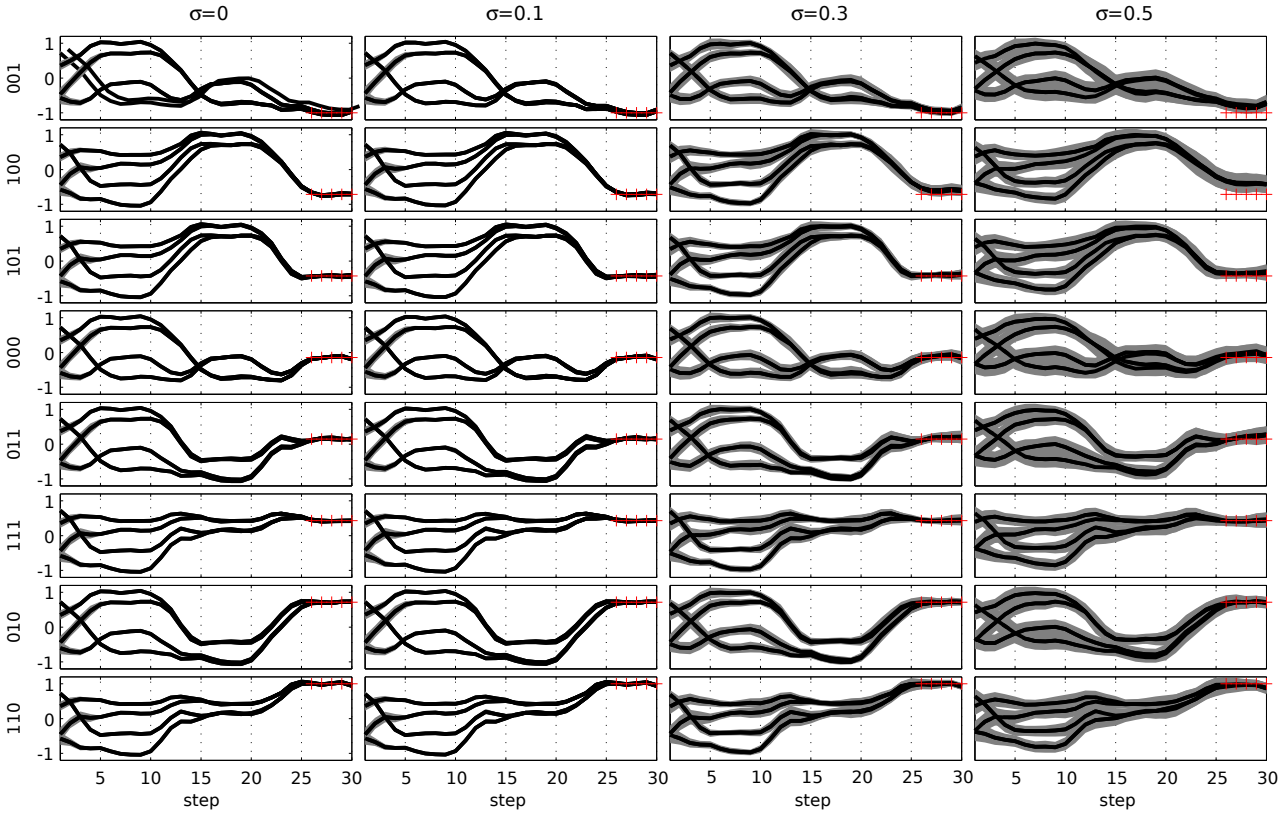


Figure 8. Evaluation of a network trained for the 3 bit classification task under the influence of input noise. As exploration noise drives the learning rule, the trained networks are generally robust against both input and state noise. The noise amplitude increased from left to right. The 8 possible inputs are shown from top to bottom. The gray area indicates one standard deviation around the observations for each of the different state trajectories (thick lines). The red crosses indicate the target observations at the end of the trial.

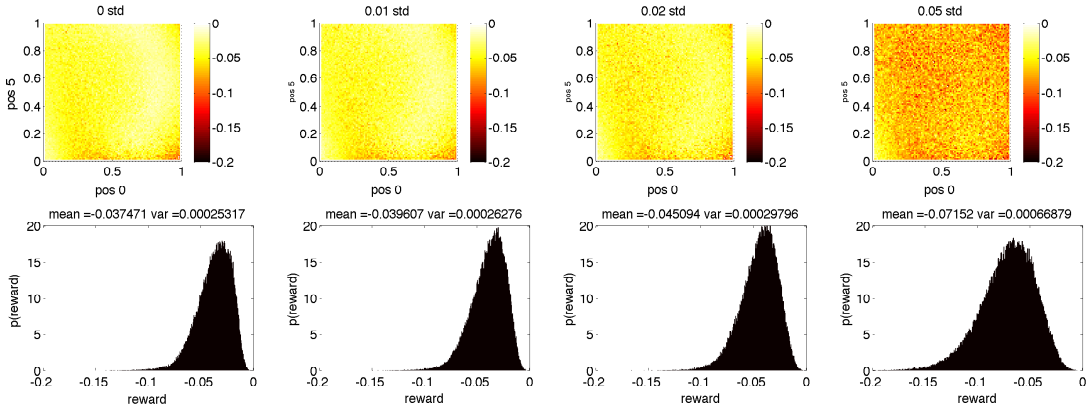


Figure 9. Reward distribution for Task 3 during testing with different state noise levels: (Top) average reward (negative mean absolute error) for the whole range of input combinations. The horizontal and vertical axes of each plot indicate the initial and final values of the linear segments that need to be reproduced, in reverse order, at the output (Bottom) sample distribution of the rewards for all inputs. The amount of injected state noise increases from left to right. We can see the average reward level shift to the left (lower rewards).

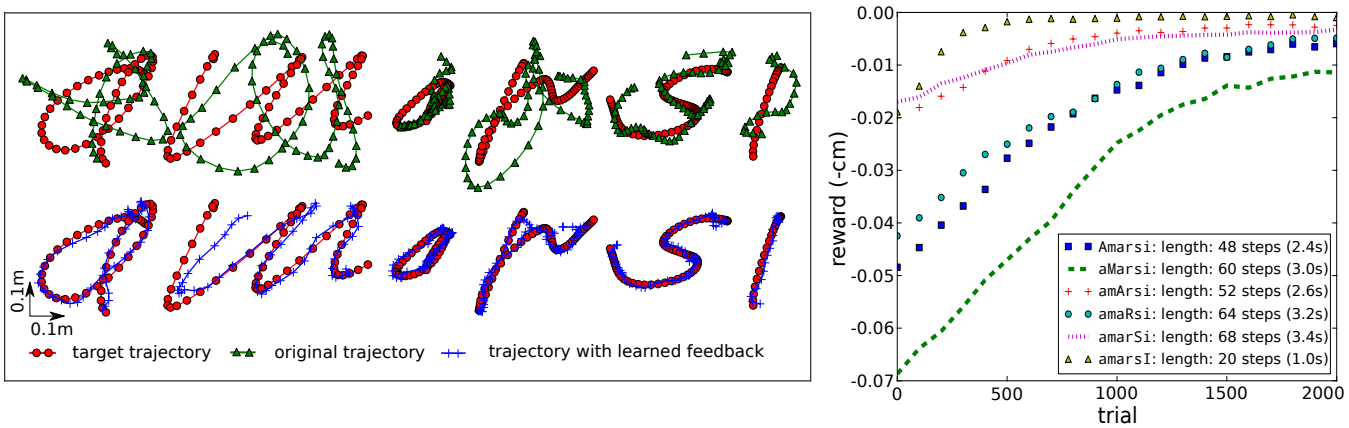


Figure 10. Writing characters with a tensegrity end-effector. Top left: characters drawn with only the kinematic feed forward controller active. Bottom left: characters drawn with the kinematic feed forward controller and the learned feedback controller active. Right: learning curves for the different characters. The legend indicates the length of a trial.

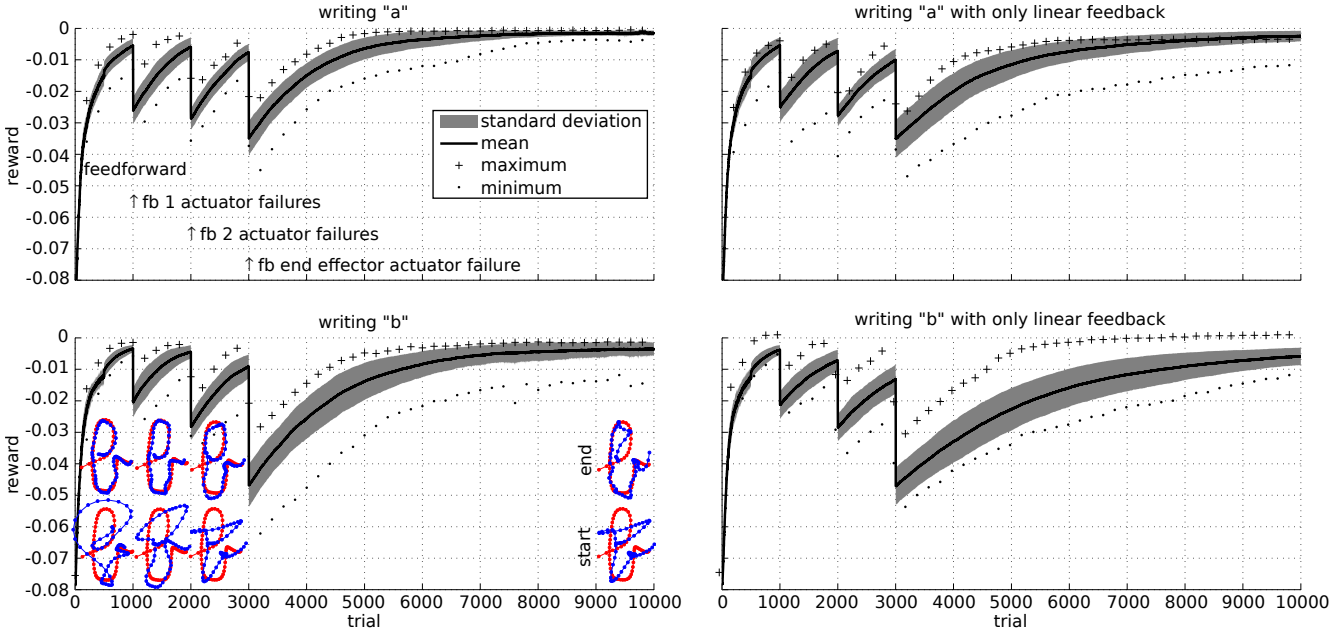


Figure 11. Robustness of the learning rule for the writing task. Initially a feed forward controller is optimized using gradient ascent. We then simulate a failure by making a single actuator follow its initial trajectory from trial 1000 onwards. At the same time, the learning rule starts learning a set of feedback weights to compensate for the actuator failure. Similarly, we simulate a 2 actuator failure after trial 2000. At time 3000, we simulate a failure of an actuator directly attached to the end-effector. The top row shows the results for writing an ”a” character, while the bottom row shows the results for a ”b”. The left column show the results when the feedback includes the spring forces and the square of the spring forces, while the right column only includes the spring forces.

Figure 1.JPEG

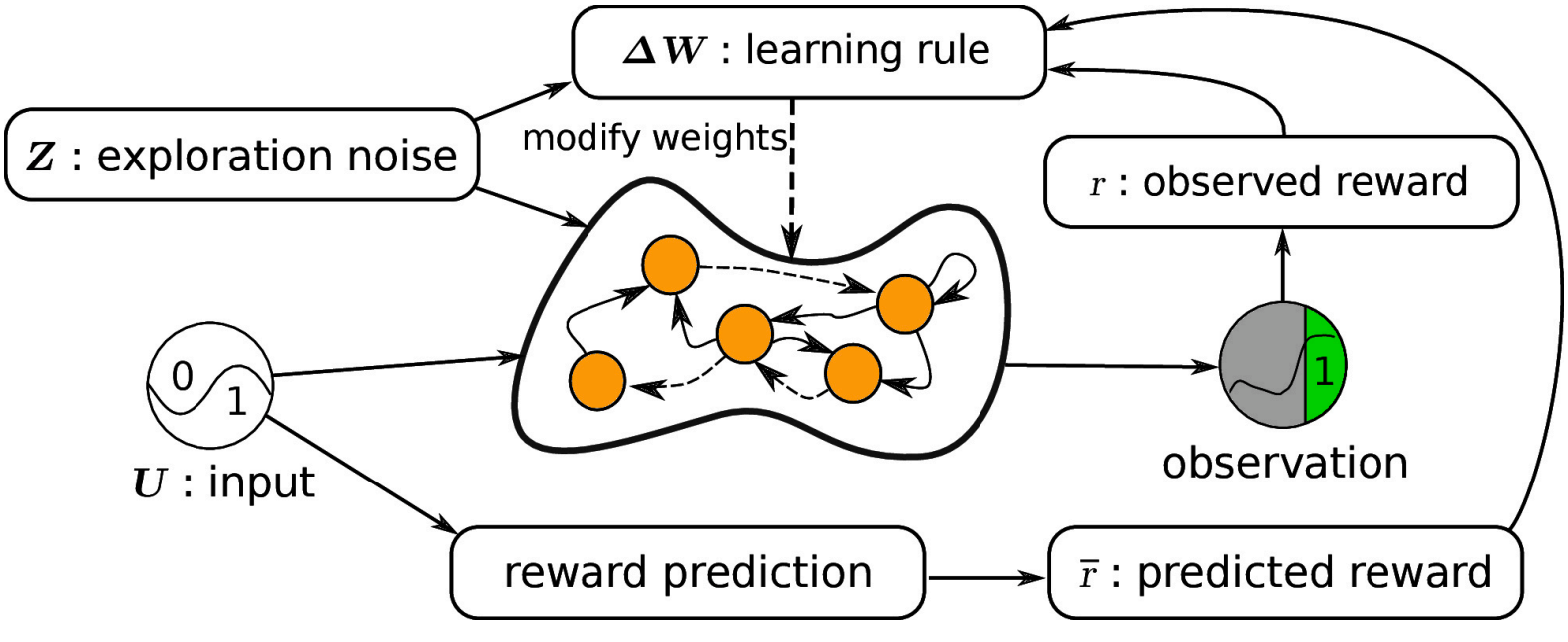


Figure 2.JPEG

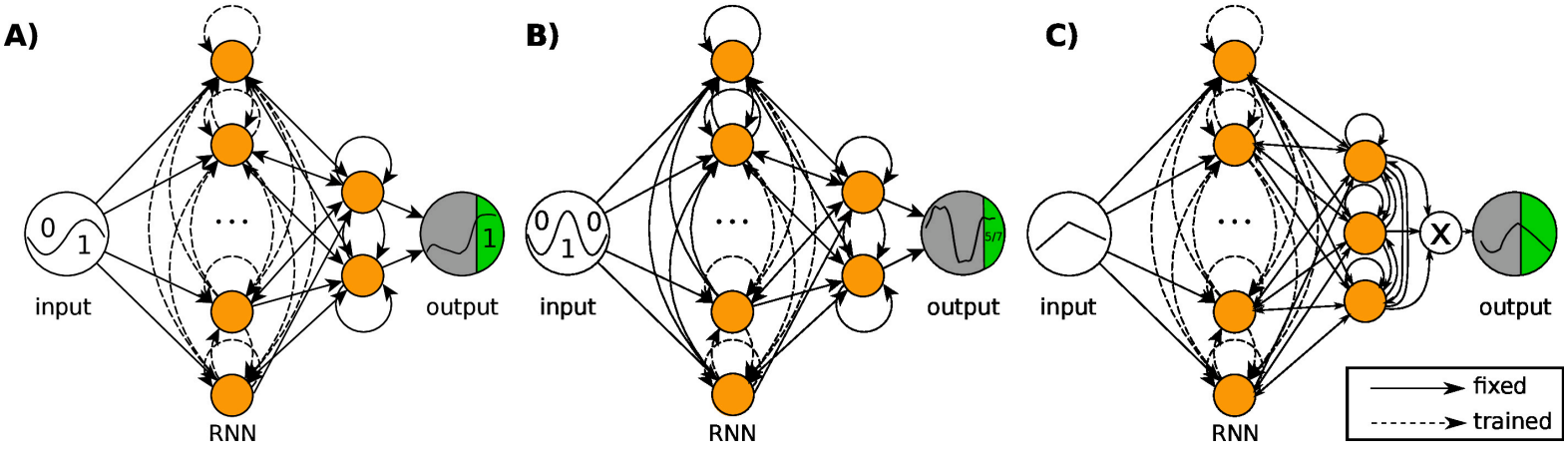


Figure 3.JPEG

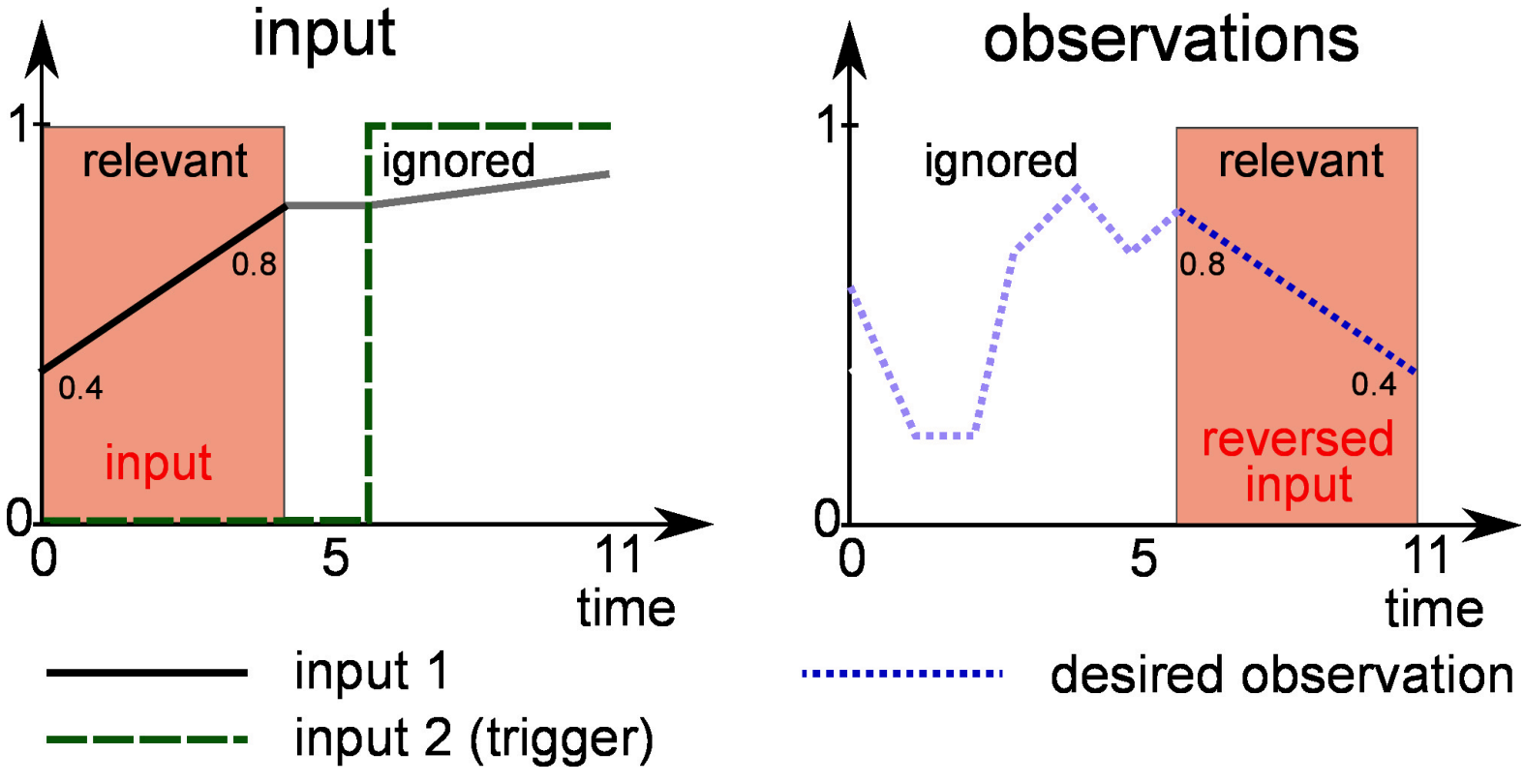


Figure 4.JPEG

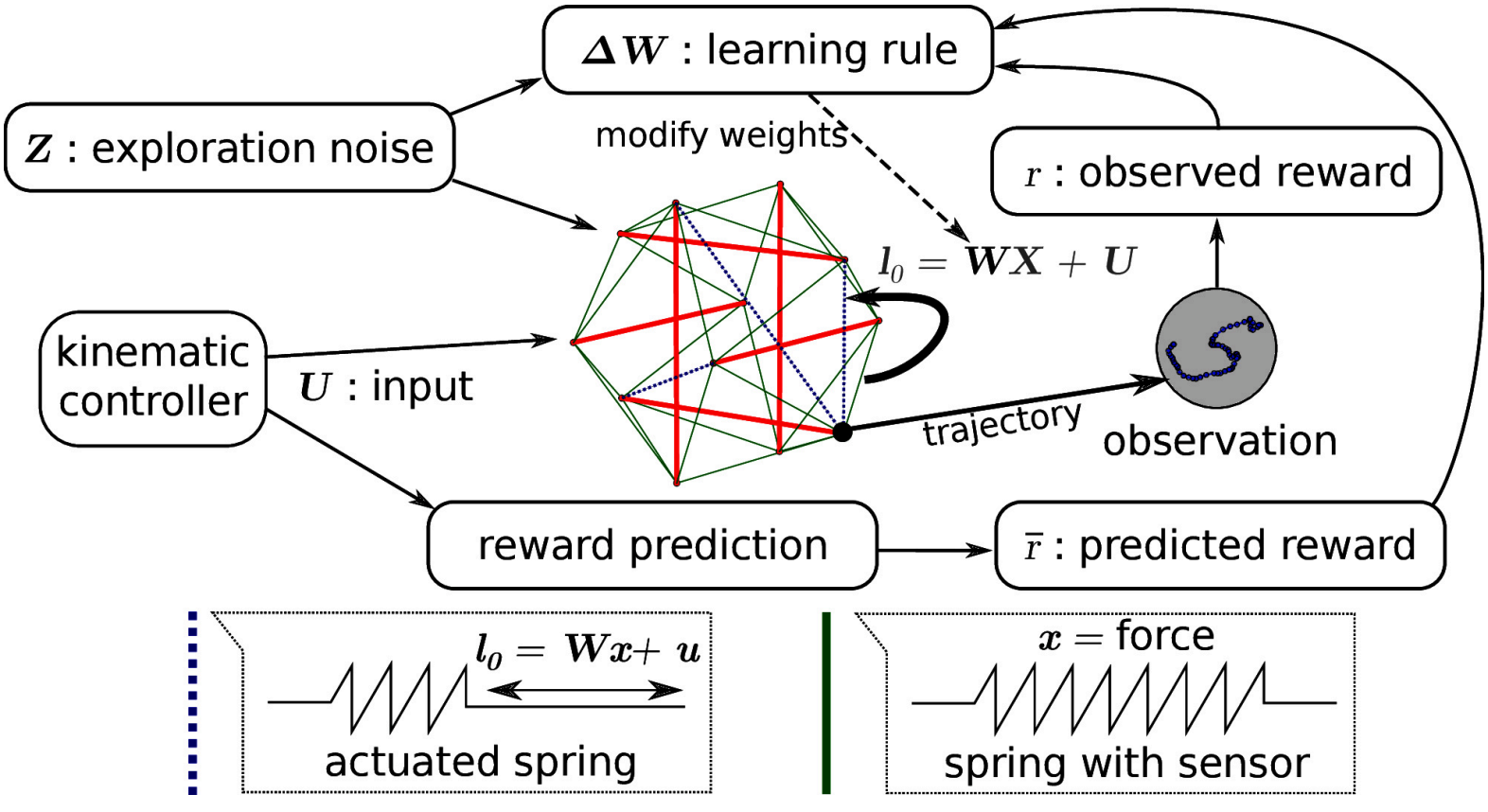


Figure 5.JPEG

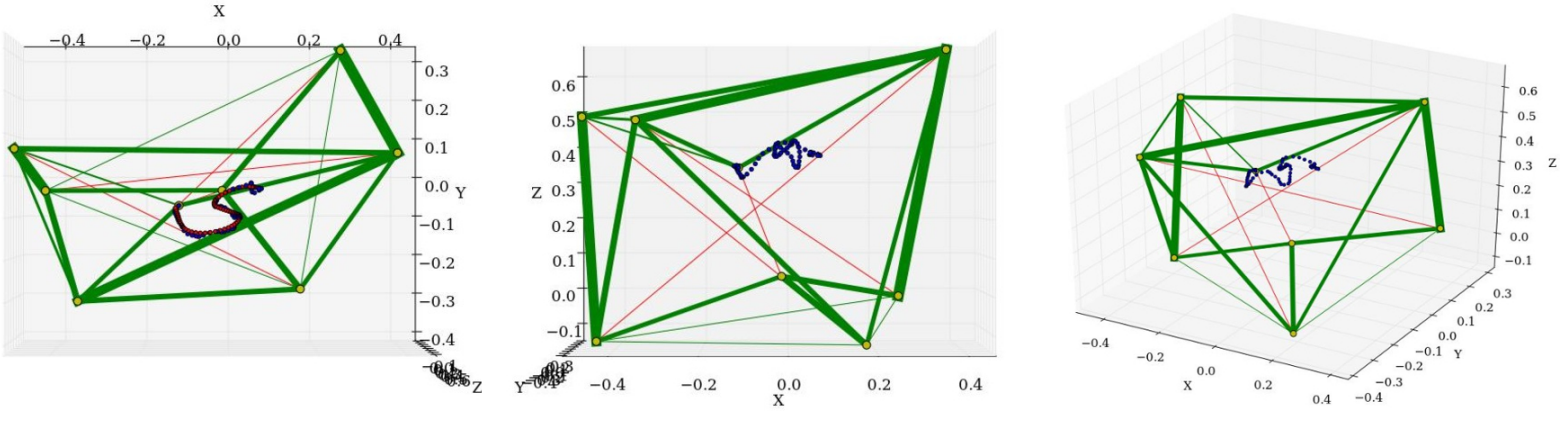


Figure 6.JPEG

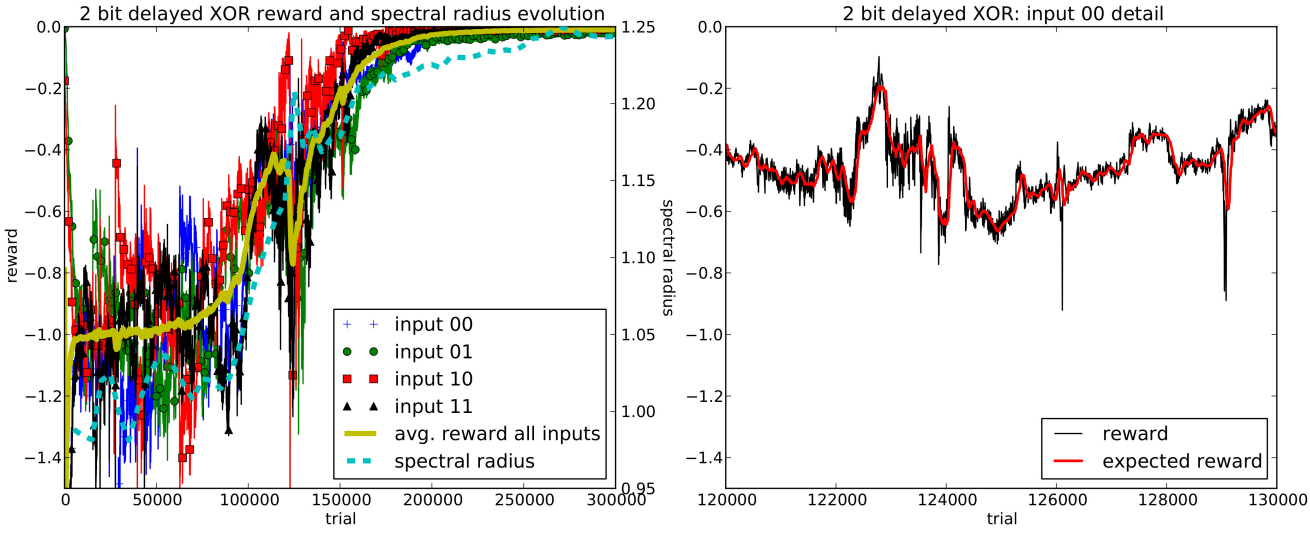


Figure 7.JPEG

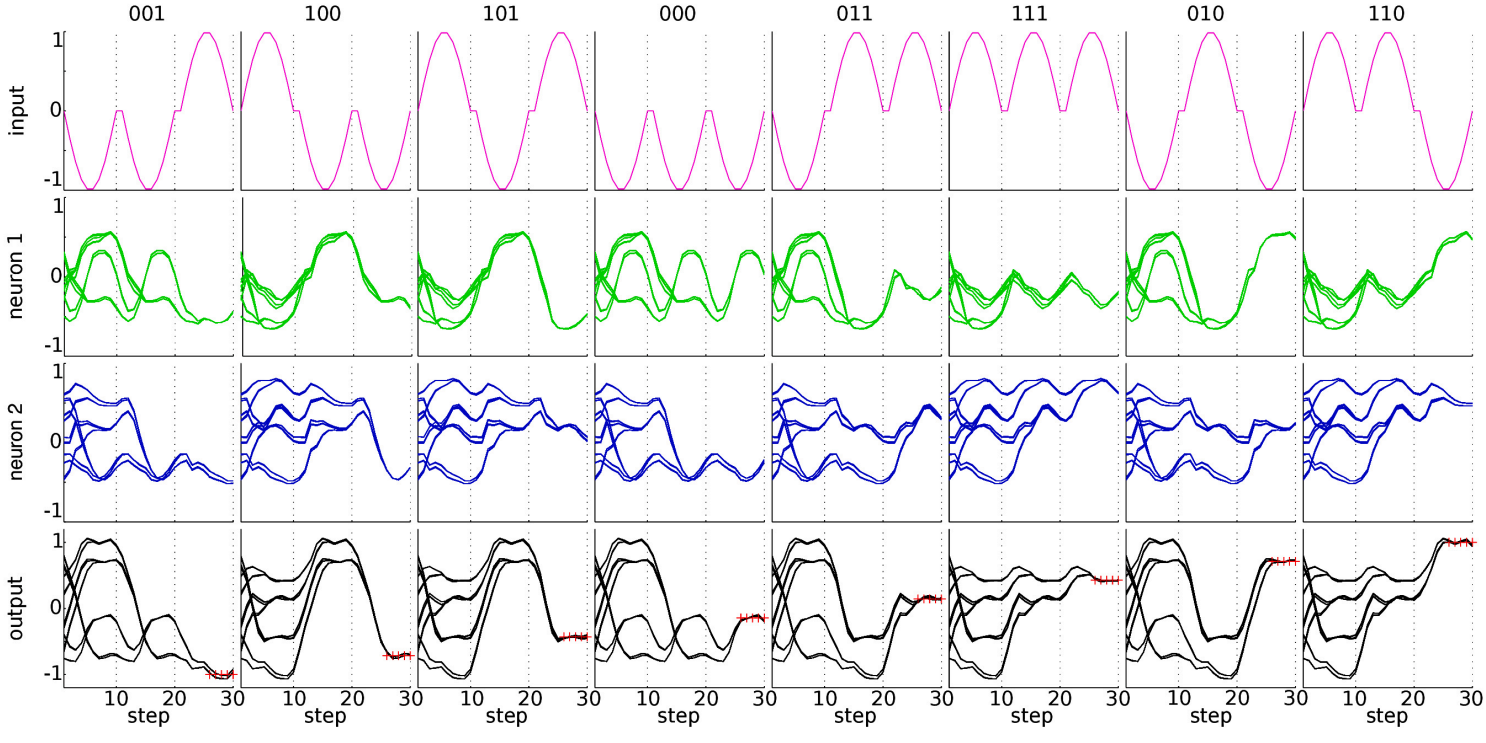


Figure 8.JPEG

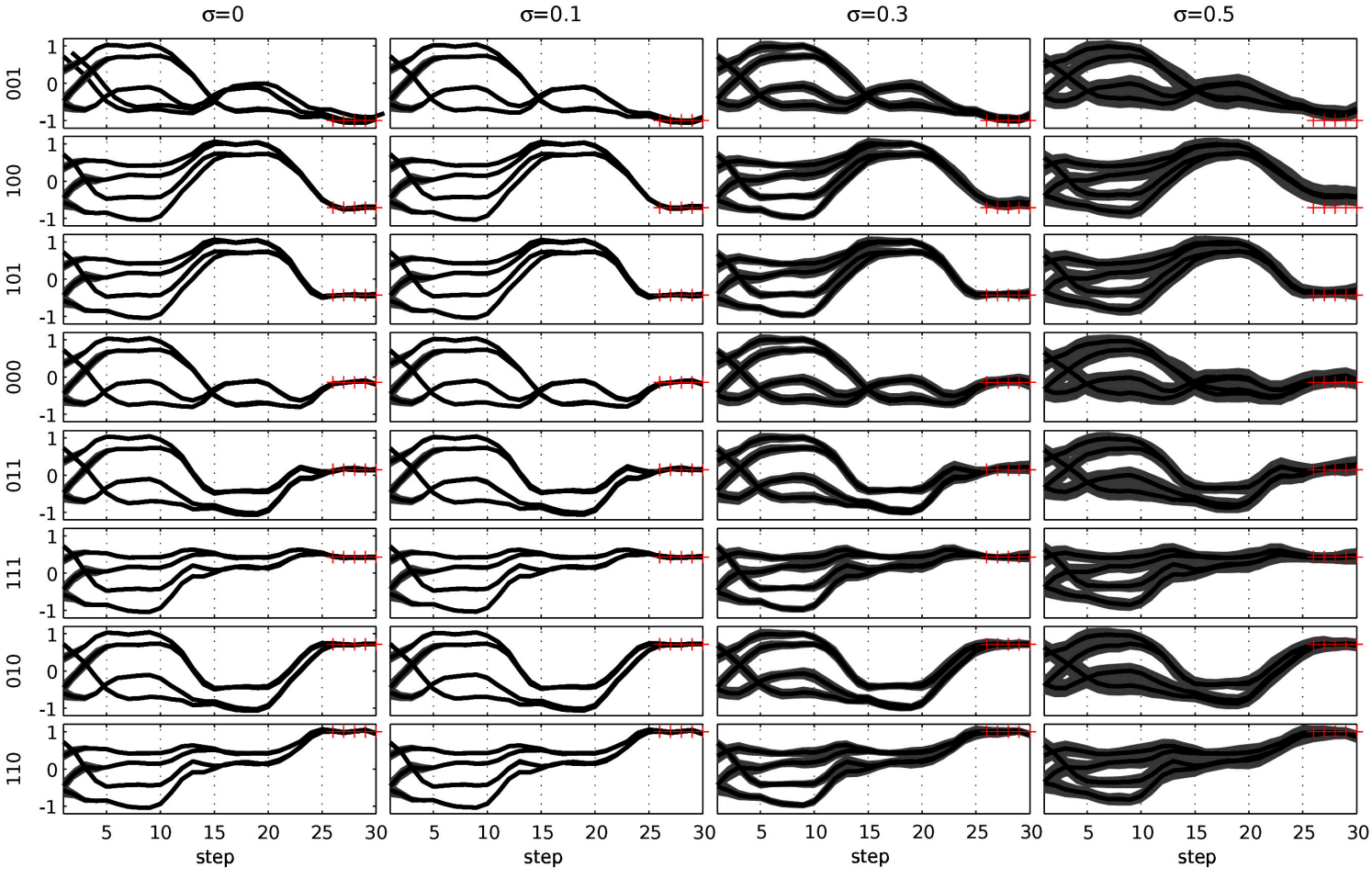


Figure 9.JPEG

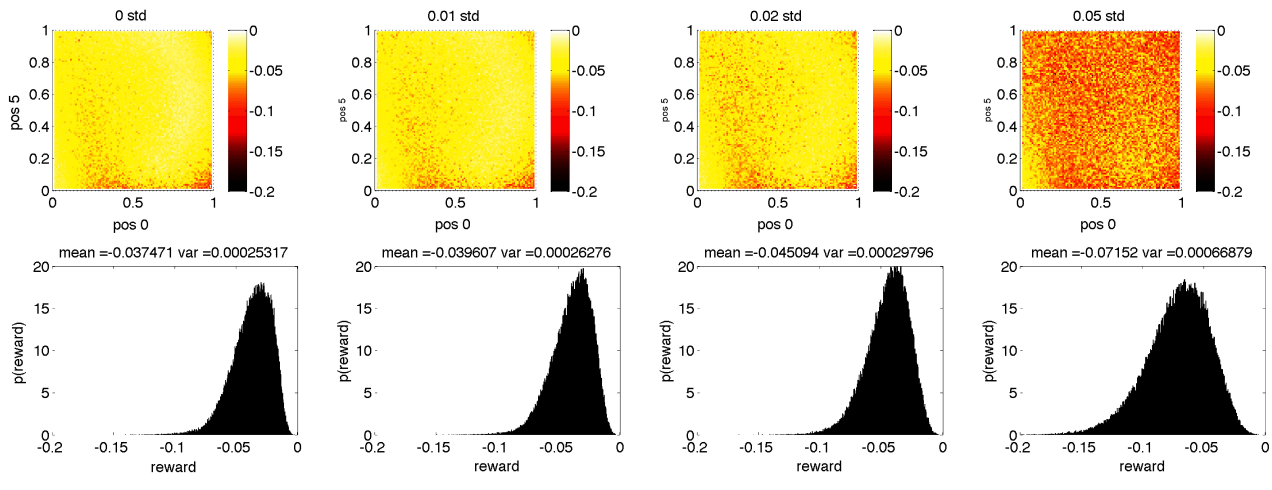
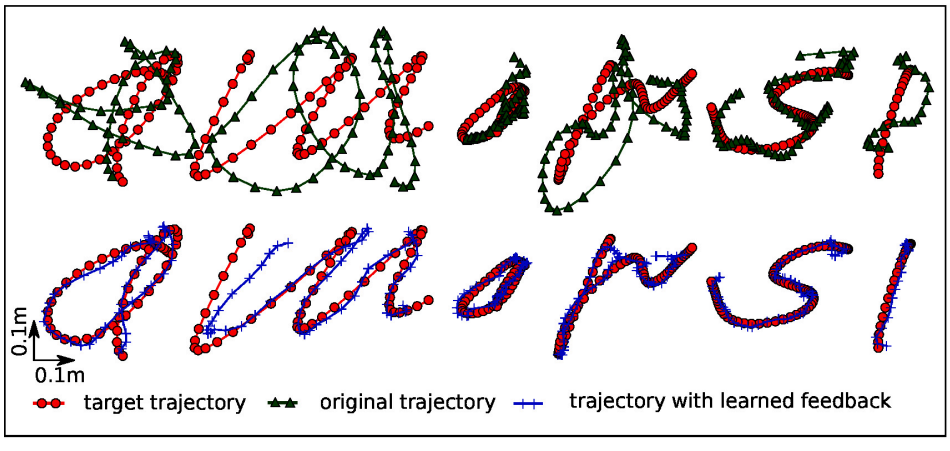


Figure 10.JPEG



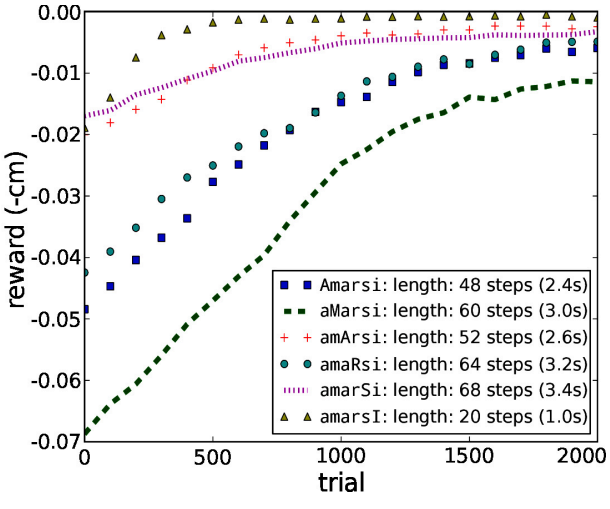


Figure 11.JPEG