Learning and Modulating Motor Commands using FORCE-trained Spiking Neural Networks

Carlo Alessi

October 20, 2019

Abstract

Supervised learning in networks of spiking neurons is efficiently carried out through a technique referred to as FORCE learning, at least for a certain range of frequency content in the teaching signals. As long as the teaching signals are limited to specific harmonics, FORCE can be efficiently used for learning and reproducing periodic signals [9], for chunking input streams [2], implementing signal routing [7], etc. The process however seems to break down when the teaching signals contain either low-frequency or very high-frequency components. This is due to the fact that the output of networks in the FORCE scheme is intimately bound to the intrinsic neuronal dynamics of these networks. This in turn currently restricts the applicability of the scheme to some toy examples, and prevents its more widespread use. In order to address these limitations, the following three questions are therefore of interest: 1) Can the FORCE scheme be adapted in order to learn sequences with low-frequency components, especially in the context of behaviourally-relevant motor tasks? 2) Can the FORCE scheme be extended in order to learn sequences and replay them at speeds that differ from the teaching signal? 3) Can the FORCE scheme be linked to Reinforcement Learning to derive an error/reward signal to drive the weight update process (which is more biologically plausible), instead of using explicit target functions (that are not biologically plausible)? One possible path to explore in this framework is to consider whether the requirement of the FORCE scheme to use a chaotic, non-saturated regime as a starting point for the learning process can be relaxed, and what that entails in terms of network structure, readout schemes, temporal constants of neuronal dynamics and learning processes. Some practical solutions can be implemented that go in that direction, and applied to motor control in the HBP Neurorobotics Platform (with a proposed emphasis on control of arm movements).

Contents

Chapter 1

Background

1.1 Introduction

According to the International Federation of Robotics (IFR), robots play an important role in today's manufacturing industry. In the last decade there has been an increasing demand of robots in most of the industrial sectors, especially in the automotive and electronics industry. In particular, collaborative robotics is entering the market as the new frontier of industrial robotics, because it combines the high accuracy, speed and repeatability of robots with the flexibility and cognitive skills of humans [11]. The results are augmented worker productivity, stress and fatigue reduction.

cita summary IFR

Human-robot collaboration Collaborative robots (cobots) enable direct interaction between human operators and robots. To obtain a successful and efficient human-robot interaction, several challenging problems must be solved: (i) motor control; (ii) safety issues; (iii) user interfaces; and (iv) user modeling.

Motor control for trajectory planning can be achieved using either standard methods based on control theory, or via more or less biologically plausible neurocontrollers.

Traditional industrial robots work confined into cages where humans are not allowed to enter. Conversely cobots are designed to collaborate with a human operator on the same area and task; therefore they need to be lightweight and be equipped with sensor avoidance sensors. A safe interaction must be guaranteed to prevent human-robot collisions and the risk of injuries.

Moreover, intuitive user interfaces must be properly designed, so that human operators can easily program and interact with the robot. Robots can be taught using several methods, such as systems based on augmented reality, programming by demonstration, etc.

Finally user modeling is needed to adapt the assistance provided by the cobot in order to maximise the operator's physical and cognitive well-being.

Project objectives In this project the focus in only on motor control by means of a neural network. The aim is to use a single neural network to learn a set of primitives that would allow an industrial

robotic arm to reach several points in a workspace and return to a home position, operating at different velocities (accelerating and decelerating on demand).

advantages of proposed approach (implementation on neuromorphic hardware, see ipad)

Possible real-world applications The results of this project could find a direct application in industrial scenarios where (i) robots are used in assembly lines or (ii) collaborate with a human operator.

For the assembly line, a robot operating by a conveyor belt could adapt its velocity according to the velocity of the conveyor belt. For the human-robot collaboration, the robot could adapt to the human speed.

interpolazione buona per coprire tutta l'area di uno scaffale o di un piano da lavoro.

- interpolazione molto importante perche non devi imparare ogni singola posizione. buona capacita di generalizzazione

Thesis outline The outline of the thesis is as follows.

The remaining of chapter 1 covers the background material. The necessary robotics fundamentals are summarized in section 1.2. Section 1.3 reviews the Izhikevich neuron model, which is one of the main building blocks of the neural network used. Section 1.4 describes the main features and functioning of the FORCE learning scheme, while section 1.5 reviews some applications of FORCE training in learning complex natural signals, as well as some relevant contributions to motor control using neurocontrollers.

In chapter 2 the methodologies adopted in the project are explained in detail. The datasets used in the experiments are described in section 2.1. finisci

The results are reported in chapter 3. finisci

The results are discussed in chapter 4. finisci

chapter 5 concludes with a summary of the main findings and an insight for future work.

1.2 Robotics Foundamentals

links, joints, DH parameters, forward and inverse kinematics, how can the IK be solved, IK challenges, PID control (?), neurocontrollers

1.3 The Izhikevich model

The Izhikevich model [8] is one of the most widely used models of spiking neurons because it combines the biological plausibility of the Hodkin-Huxley model and the computational efficiency of Integrate-and-Fire neurons. In this project it was used a variant of the model proposed by Nicola & Clopath in [9]. The model is composed of two components: (i) a two-dimensional system of ordinary differential equations (1.1), which models the sub-threshold behaviour of the system and generates the upstroke

of the action potential; (ii) an after-spike reset mechanism (1.2), which generates the downstroke of the action potential.

$$C\dot{v} = k(v - v_{rest})(v - v_{threshold}) - u + I \tag{1.1a}$$

$$\dot{u} = a(b(v - v_{rest}) - u) \tag{1.1b}$$

$$if\ v \geq v_{peak}\ then \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases} \tag{1.2}$$

The two variables of the model are the membrane potential $\boldsymbol{v}$ (mV), and the adaption/recovery variable $\boldsymbol{u}$ (pA), which allows for spike frequency adation, and represents the effect of slow current on the spike generation.

The model presents a wide variety of parameters.

- $\boldsymbol{C}$ (pF), membrane capacitance;
- $\boldsymbol{v_{rest}}$ (mV), resting membrane potential;
- $\boldsymbol{v_{threshold}}$ (mV), voltage threshold when $b = 0$ and $I = 0$;
- $\boldsymbol{I}$ (pA), injected or synaptic current;
- $\boldsymbol{v_{peak}}$ (mV), voltage peak (i.e. spike cutoff value);
- $\boldsymbol{k}$ (nS/mV), gain on v;
- $\boldsymbol{a}$ (ms^{-1}), time constant of the adaption current;
- $\boldsymbol{b}$ (nS), sensitivity of the adaption current u to subthreshold fluctuations of the membrane potential v;
- $\boldsymbol{c}$ (mV), post-spike reset value of the membrane potential;
- $\boldsymbol{d}$ (pA), post-spike additive value to the recovery variable u.

cambia il nome del parametro k, perche gia usato per il numero di dimensioni del segnale

The parameter $k > 0$ is a scaling factor that controls the action potential up-swing [5]. It affects the value of $\dot{v}$ and hence the spike width. It is important for determining the half width of the action potential. Smaller values of k give smaller half widths.

The parameter b describes how sensible is u to subthreshold oscillations of v. Greater values of b couple v and u more strongl, resulting in possible subthreshold oscillations and low-threshold spiking dynamics. The sign of b determines whether the effect of u is amplifying $(b < 0)$ or resonant $(b > 0)$.

maybe make a table with parameter-units-description

The voltage reset value c is designed to model the effect of fast high-threshold K^+ conductances. The appropriate choice of this parameter will help to obtain a suitable fast post-hyperpolarization.

what does it mean?

The value d is added to the recovery variable u after a spike occurs. It describes the total amount of outward minus inward currents activated during the spike, which affects the after-spike behaviour. This parameter, together with a, determines how much spike frequency adaption the model exhibits.

Note that both u and I are currents and in (1.1a) they have opposite sign ($-u$ and $+I$). Therefore they can potentially balance each other out. Moreover the negative feedbak that u has on the membrane potential v, combined with the after spike reset ($u \leftarrow u + d$), mean that both the value and sign of d influence the spike frequency adaption.

In particular, when $d < 0$, the variable u tends to become negative. Therefore the current $-u$ will be exitatory. As a result the network could spike at a higher rate (i.e. spike facilitation). Moreover the more it spikes, the more it is easy to produce ulterior spikes. Conversely when $d > 0$, the current u will become inhibitory. In the special case where $d = 0$ the network is neither facilitated or discouraged to spike.

At each time t, the discretized evolution of the system dynamics is given by:

$$v(t+1) = v(t) + \delta t \cdot [(k(v(t) - v_{rest})(v(t) - v_{threshold}) - u(t) + I)]/C \tag{1.3}$$

$$u(t+1) = u(t) + \delta t \cdot a[b(v(t) - v_{rest}) - u(t)] \tag{1.4}$$

The value $\boldsymbol{\delta t}$ is the integration time constant. If at time t a spike occurs, the variables $\boldsymbol{v}$ and $\boldsymbol{u}$ are reset according to (1.2).

1.4 The FORCE method

The FORCE training method originated in the field of reservoir computing, and was first proposed by Sussillo et Al. in [10]. It is used to modify the synaptic strengths of an artificial neural network, either external or withing the model, to change chaotic spontaneous activity into a wide variety of desired patterns. The method is widely applicable because any multi-dimensional dynamical system can be used as a reservoir, guiding its dynamics using and error signal. Therefore FORCE can be used with many network type to solve many task. Unlike other techniques, the target behaviour does not have to be specified as a closed form differential equation for training. All that is required for training is a supervisor to provide an error signal.

A variant of the original FORCE method, proposed by Nicola&Clopath in [9], is described below.

Network architecture The newtork architecture is a Recurrent Neural Network (RNN) composed by a reservoir of Izhikevich neurons, a linear layer of readout units, and a feedback channel that connects the output of the readouts back into the reservoir, see Figure 1.1.

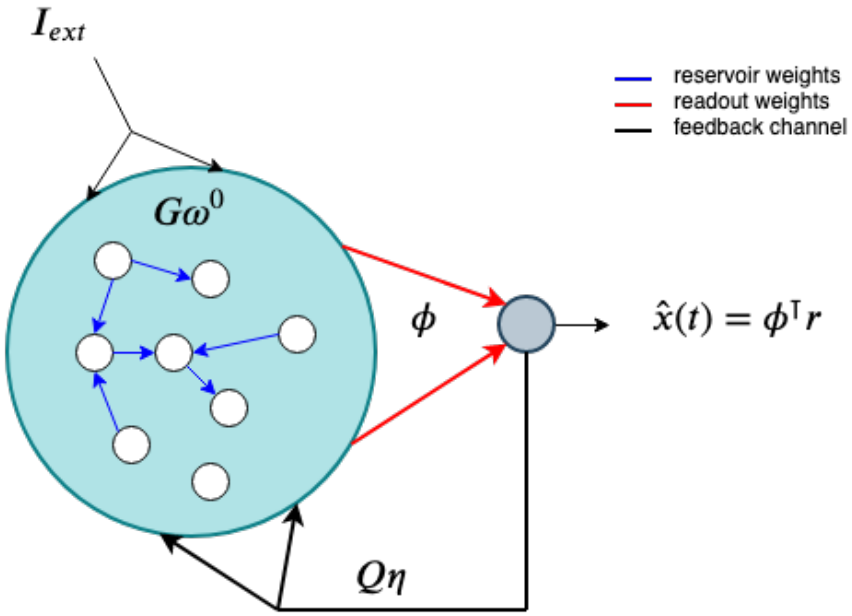


Figure 1.1: Network architecture. Example of a reservoir with $N = 8$ Izhikevich neurons and $k = 1$ readout unit.

Network weights In the FORCE method, the weight matrix ω is decomposed by a static weight component $\boldsymbol{\omega^0}$, a learned decoder $\boldsymbol{\phi}$, and a static matrix $\boldsymbol{\eta}$:

$$\omega = G\omega^0 + Q\eta\phi^\intercal \tag{1.5}$$

The term $\boldsymbol{\omega^0} \in \mathbb{R}^{N\times N}$ is a sparse and static weight matrix that determines the connectivity of the neurons inside the reservoir, and induces chaos in the network. The chaos is balanced by the constant $\boldsymbol{G}$. Each element ω_{ij} is the weight of the synaptic connection between the pre-synaptic neuron j and the post-synaptic neuron i.

The component $\boldsymbol{\phi} \in \mathbb{R}^{N\times k}$ is the weight matrix of the readout layer, which serves as a linear decoder of the network dynamics. The values of $\boldsymbol{\phi}$ are learned using a supervised learning method called Recursive Least Squares (RLS).

The static matrix $\boldsymbol{\eta} \in \mathbb{R}^{N\times k}$ encodes the output of the readout layer back into the reservoir, and defines the tuning preference of the neurons. The constant $\boldsymbol{Q}$ balances the effects of the decoder.

Network output The network output is defined as a weighted sum of the network activities, and determines the task that the network performs. The network activities at time t are specified by a column vector $\boldsymbol{r(t)}$, which correspond to the firing rate of each of the N neurons of the reservoir.

The task of the network is to approximate the dynamics of a k-dimensional teaching signal, $\boldsymbol{x(t)} \in \mathbb{R}^k$, with the following approximant

$$\hat{x}(t) = \phi^\intercal r \tag{1.6}$$

Recursive Least Squares update FORCE learning requires a learning rule that rapidly reduces the magnitude of the error to a small value, and then keeps it small while the readout weight vector converges to a solution that can maintain the error small post training. The authors in [10] reported that the Recursive Least Squares (RLS) rule is a suitable and very powerful method for FORCE learning.

RLS is a weight update rule that minimizes the squared error between the network dynamics $\hat{\boldsymbol{x}}(\boldsymbol{t})$ and the target dynamics $\boldsymbol{x}(\boldsymbol{t})$ during training. The training process is considered successful if the network can well approximate the target dynamics post training. Every Δt, the readout weights are updated with the following rule[1]

$$\phi(t) = \phi(t - \Delta t) - e(t)P(t)r(t) \tag{1.7}$$

where $\boldsymbol{e(t) = \hat{x}(t) - x(t)}$ is the error between the approximant and desired dynamics, and $\boldsymbol{r(t)} \in \mathbb{R}^N$ is the vector of firing rates of the neurons. The matrix $\boldsymbol{P(t)}$ is the network estimate for the inverse of the correlation matrix of the neuron firing rates $\boldsymbol{r}$, plus a regularization term:

$$P(t) = (\sum_t r(t)r(t)^\intercal + \lambda I_N)^{-1} \tag{1.8}$$

The parameter λ acts as a regularization parameter and controls the rate of the error reduction, while $\boldsymbol{I_N}$ is the $N \times N$ identity matrix. The matrix $\boldsymbol{P}$ is also updated at the same time as the weights, according to the rule:

$$P(t) = P(t - \Delta t) - \frac{P(t - \Delta t)r(t)r(t)^\intercal P(t - \Delta t)}{1 + r(t)^\intercal P(t - \Delta t)r(t)} \tag{1.9}$$

The weight update rule (1.7) can be viewed as a standard delta-type rule, but instead of having a single scalar learning rate (e.g. as in Gradient Descent), it uses multiple learning rates given by the matrix-vector product $\boldsymbol{P(t)r(t)}$. The parameter λ, also functions as a learning rate, and should be adjusted depending on the particular target function being learned. Small λ values result in fast learning, however weight changes could be so rapid that learning becomes unstable. Conversely if λ is too large, the FORCE algorithm may not be able to keep the network output close to the desired dynamics for a long enough time, causing the learning to fail. Moreover the author report that for a correct functioning the learning rate should be subject to the constraint $\lambda \ll N$[10]. Values in the interval $[1, 100]$ performs well in practice.

Injected Current FORCE can make the network learn a single target $\boldsymbol{x}$ in the absence of any external input, but in the general case, different outputs depend on different inputs.

The spikes are filtered by a double exponential filter of the form

[1]Note that the time interval that pass from one weight update to the next is greater than the integration time constant, i.e. $\Delta t > \delta_t$.

$$\dot{r}_j = -\frac{r_j}{\tau_d} + h_j \tag{1.10}$$

$$\dot{h}_j = -\frac{h_j}{\tau_r} + \frac{1}{\tau_r \tau_d} \sum_{t_{jk}<t} \delta(t - t_{jk}) \tag{1.11}$$

where $\boldsymbol{\tau_r}$ is the synaptice rise time, $\boldsymbol{\tau_d}$ is the synaptic delay time, and $\boldsymbol{t_{jk}}$ is the time of the k-th spike fired by neuron j.

The synaptic currents are given by the equation:

$$s_i(t) = \sum_{i=1}^{N} \omega_{ij} r_j \tag{1.12}$$

... ...

The current is

$$I = IPSC + \eta \hat{x}(t - \delta_t) + I_{BIAS} \tag{1.13}$$

understand double exponential filter and what is IPSC.

Feedback

some observation about the feedback channel

Error reduction Most learning algorithms repeatedly modify the weights in order to gradually reduce the error, which is quite large at the beginning. This process goes on until the weights converge to a solution where the vector changes minimally, so that the training can be terminated. However, in FORCE training the error magnitudes become small from the first weight update, and the goal is to keep it small throughout the training process.

Weights initialization The static sparse matrix $\boldsymbol{\omega^0}$ is drawn from a normal distribution with mean $\mu = 0$, and standard deviation $\sigma = 1/\sqrt{Np}$, where p is the probability connection of the matrix. The encoding variables, $\boldsymbol{\eta_i} \in \mathbb{R}^k$, are drawn randomly and uniformly from $[-1, 1]^k$, where k is the dimensionality of the teaching signal. The readout weights are initialized as $\boldsymbol{\phi(0) = 0}$. The matrix $\boldsymbol{P}$ is initially estimated with $\boldsymbol{P(0)} = \lambda^{-1}\boldsymbol{I_N}$, where λ is a constant parameter.

1.5 Related Work

1.5.1 FORCE training to learn multiple signals

The FORCE method was first proposed by Sussillo et Al. in [10]. They showed that using FORCE learning it is possible to construct various network architectures that can produce (i) a wide variety of complex output patterns, (ii) input-output transformations that require memory, (iii) multiple outputs that can be switched by control inputs, and (iv) motor patterns matching human motion capture from data.

4-bit memory As an example, they designed a network that functions as a 4-bit memory that is robust to input noise. The network has 8 inputs that randomly connect to neurons in the network, and are functionally divided into pairs. The input values are held at zero except for short pulses to

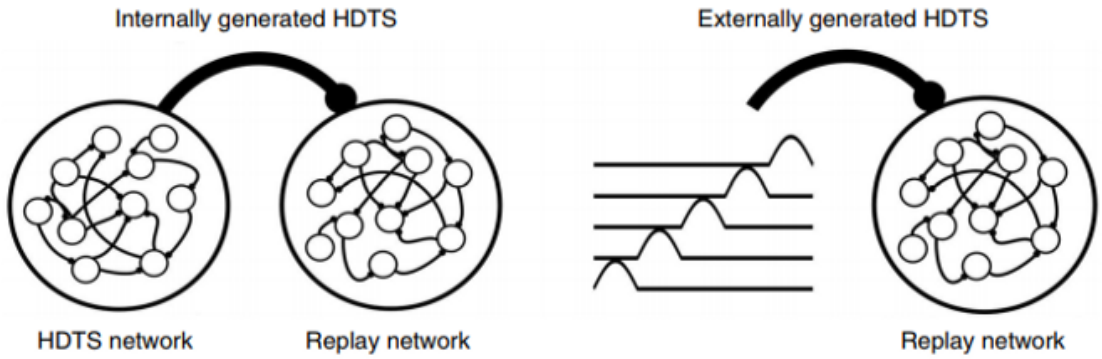


Figure 1.2: Internally and externally generated HDTS. (figure taken from [9])

positive values that act as ON and OFF commands for the four readout units. Post FORCE learning, the inputs are correctly mapped to the corresponding output, with little interference between inputs and inappropriate outputs. In order to have a functioning model, the network is required to have 16 point attractors, one for each of the 4^2 possible combinations of the four outputs. Moreover the network must be able to transition between these attractors in response to a pulse to one of the inputs.

Motion capture The authors also demonstrated that FORCE learning can be used to train a single chaotic recurrent network to generate multiple, high-dimensional, nonperiodic patterns that resemble complex human motions. They used a dataset from the CMU motion capture library, which consists of 95 joint angles trajectories for walking and running activity, measured over hundreds of time steps. The data was obtained from human subjects performing these activities while wearing a suit that permits the measurement of the joint angles. The 95 joint angles were predicted by 95 readouts. To cope with the nonperiodicity of the motions, they introduced static control inputs to initialize the network prior to starting the running or walking motions. The control inputs were also used to switch between the different activities.

1.5.2 High-dimensional temporal signals improve FORCE learning

The FORCE method has been successfully adopted to learn oscillators. In [9] it was shown that FORCE training can be aided to learn more complex behaviours by the so called high dimensional temporal signals (HDTS). The HDTS, inspired by the clock-like input pattern that songbirds use for learning and replay, discretizes time by segregating the neurons into assemblies that fire at specific time intervals.

The HDTS can be (i) *internally generated* by a separate HDTS network, or (ii) *externally generated* and directly fed as input into and encoding and replay network (see Figure 1.2). Nicola & Clopath explored the benefit of HDTS in learning complex signals such as a portion of a song and a movie scene, using either internally or externally generated HDTS.

Learn a song fragment They FORCE trained an HDTS network of Izhikevich neurons to internally generate its own HDTS, while simultaneously also training a replay network to reproduce the first bar of Ode to Joy. The supervisor consisted of 5 notes, plus 16 other components of the HDTS which correspond to a sequence of 16 pulses that partition time. The networks learned both the HDTS and the song, with less training time and greater accuracy than without HDTS. They also succeded in learning the first four bars of the song (corresponding to 16 seconds) with a 64-dimensional HDTS.

Learn a movie scene To investigate if HDTS would help to learn natural high-dimensional signals, they trained a network of Izhikevich neurons to learn a movie scene of 8 seconds. The high-dimensional supervisor consisted in the evolution of the 1920 pixels of the frame. The training of the replay network was successful for both internally and externally generated HDTS.

They observed that the HDTS inputs were necessary both for training and replay. Without HDTS, the network could still replay the individual frames from the movie scene; however the order of scenes was incorrect and appeared chaotic. Moreover, decreasing the HDTS amplitude yield a steady decrease in replay performance. Also, by observing the histogram of spike times in the replay network, they noticed that the HDTS conferred a strong $4Hz$ modulation (the histogram can be interpreted as the mean network activity). Removing the HDTS, the mean network activity displayed a much slower oscillation ($\approx 2Hz$), which corresponded to a sharp decline in replay performance, as the scenes were no longer played in chronological order.

Frequency and amplitude of HDTS They wondered how the frequency and amplitude of HDTS would affect the learning performance of a network. They found that an optimal input frequency was located in the interval $8 - 16Hz$, for large regions of parameter space, which was robust to different neural parameters.

Accelerated and reverse replay Several studies speculated that compressed of reversed replay of an event might be important for memory consolidation . Thus they wondered if networks trained with an HDTS could replay the scene in accelerated time by compressing the HDTS post-training. They compressed the external HDTS after training, and uncovered that the network was able to replay the movie scene in compressed time up to a compression factor of 16x (correlation $r > 0.8$). The accuracy sharply dropped for further compression. The effect of time compression on the mean activity was to introduce higher frequency oscillations. The frequency of these oscillations scaled linearly with the amount of compression. However, with increasing compression frequency, large waves of synchronized activity also emerged in the mean population activity.

(cite studies?)

use a low-pass filter to smooth?

Reverse replay was also successfully obtained by reversing the order in which the HDTS components were presented to the network (correlation $r = 0.9$). The authors reported that the loss in accuracy was due to the fact that the temporal dynamics of the network is not reversed within a segment of the HDTS. Therefore the network could generalize robustly to accelerated and inverse replay, despite not being trained to do so.

how to counter this?

The key findings of [9] regarding HDTS can be summarized as follows:

- HDTS can make FORCE training faster, more accurate, and more robust to learn longer signals.
- HDTS inputs facilitate both learning and spontaneous replay of high-dimensional signals.
- a network trained with HDTS can repeat the signals in accelerated time or in inverse order by compressing or reversing the HDTS.

1.5.3 Robot control via artificial neural networks

In this section some proposed solutions to robot control using artificial neural networks are reviewed.

I. A simple but valuable example was proposed by Duka in [4]. The author investigated the use of a feed-forward neural network to solve the IK problem for a three-link simulated robotic arm. The hypothetical robotic arm was composed by three movable links l_1, l_2, l_3, of the same size $l_1 = l_2 = l_3 = 2$ that operate within a plane. The links are connected by three rotational joints, q_1, q_2, q_3, which have the axes of rotation perpendicular to the plane where the links lie. The joints movements were respectively limited by the intervals

$$q_1 \in [0, \pi], \qquad q_2 \in [-\pi, 0], \qquad q_3 \in [-\frac{\pi}{2}, \frac{\pi}{2}] \tag{1.14}$$

The coordinate position $P_E = (X_E, Y_E)$ of the end-effector, with respect to the reference system located to the base of the robot $O_{X_0Y_0}$, is defined by the following forward kinematics equations

$$\begin{aligned} X_E &= l_1 \cos(q_1) + l_2 \cos(q_1 + q_2) + l_3 \cos(q_1 + q_2 + q_3) \\ Y_E &= l_1 \sin(q_1) + l_2 \sin(q_1 + q_2) + l_3 \sin(q_1 + q_2 + q_3) \end{aligned} \tag{1.15}$$

The orientation Θ_E of the end-effector, described by the angle of rotation of the reference system attached to the end-effector $O_{X_EY_E}$ relative to $O_{X_0Y_0}$, is defined as

$$\Theta_E = q_1 + q_2 + q_3 \tag{1.16}$$

The task of the robotic manipulator was to perform a circular trajectory in its workspace. The points in this trajectory were defined by the equation of a circle of radius r, centred in (x_c, y_c), parameterized by φ

$$\begin{aligned} X_d &= x_c + r \cos \varphi \\ Y_d &= y_c + r \sin \varphi \\ \varphi &\in [0, 2\pi] \end{aligned} \tag{1.17}$$

The point $P_d = (X_d, Y_d)$ corresponds to the desired position of the end-effector. The desired orientation Θ_d of the end-effector is expressed by the angle between the positive x-axis and the line that connects $O_{X_0Y_0}$ to a point on the circle trajectory

$$\Theta_d = \arctan \frac{Y_d}{X_d} \tag{1.18}$$

The training data for the neural network was generated by sampling 1000 random joint angle values uniformly over the intervals defined in (1.14). From each sampled configuration of joint angle values $(q_1, q_2, q_3)^\intercal$, the corresponding end-effector pose $(X_E, Y_E, \Theta_E)^\intercal$ was computed with the forward kinematics equations defined in (1.15). To counter the fact that for the same end-effector pose $(X_E, Y_E, \Theta_E)^\intercal$ the IK problem has multiple solutions $(q_1, q_2, q_3)^\intercal$, the author removed duplicates in order to obtain a unique cartesian-joint space mapping. Before training, a final preprocessing step was applied to scale the entire dataset to the interval $[-1, 1]$.

The neural network was trained to map $(X_E, Y_E, \Theta_E)^\intercal$ to $(q_1, q_2, q_3)^\intercal$ using the Levenberg-Marquardt algorithm. The dataset was divided according to the following partition percentages: 70% training set, 15% validation set, 15% test set. The performance was evaluated computing the mean squared error between the network output and the desired output. The error reduced to the order of $O(10^{-3})$ in circa 10 training epochs. The network architecture consited in 3 input neurons, 3 linear output neurons, and one hidden layer with 100 tanh neurons.

conclude paragragh

II. Another solution based on multilayer perceptrons to solve the inverse kinematics problem was proposed by Aggarwal et Al. in [1], with a focus on the prediction of singularity zones.

They used the PUMA 560 robot manipulator, which has six degrees of freedom. The first three joints determine the end-effector position, while the last three joints determine the end-effector orientation. The Forward Kinematics of the robot was computed using homogeneous transformation matrices ${}^{(i-1)}A_i$, defined by the D-H parameters of the PUMA 560 manipulator. A dataset of 1000 samples was constructed using the values of the joint angles $\theta_1, \ldots, \theta_6$, and the corresponding cartesian coordinates of the end-effector (X_E, Y_E, Z_E), which were computed from ${}^0A_6$ using unique combinations of the joint angle ranges.

As above, the neural network was trained to map cartesian coordinates to joint angles, minimizing the mean square error using the Levenberg-Marquardt learning rule. The network architecture consited in three input neurons, one hidden layer with 30 sigmoid neurons, and six linear output neurons. The dataset was partitioned in 70% training set, 15% validation set, and 15% test set. The test set was used to measure the network performance on cartesian points where the singularity condition was satisfied. In this set, it was obtained a 1.32 MSE. Moreover, it was observed that singularity zones were located throughout the robot workspace. They reported that this was due to the fact that the PUMA 560 manipulator assumes a singularity condition when $\theta_5 = 0$ radians.

III. A slightly different approach for robot control was studied by Bouganis & Shanahan in [3]. They presented a Spiking Neural Network architecture that autonomously learns to control a 4-DoF robotic arm in the three-dimensional space, after an initial period of *motor babbling*[2].

[2] *Motor babbling* is the process of repeatedly performing small random motor movements for a short duration, that tries to mimic human-like cognition and learning. The robot performing motor babbling can autonomously develop an internal model of itself and the environment

Given the current joint configuration $\Theta = [\theta_1, \ldots, \theta_4]$ and joint velocities $\dot{\Theta} = [\dot{\theta_1}, \ldots, \dot{\theta_4}]$, the resulting end-effector spatial velocity $\dot{e} = [\dot{e}_1, \ldots, \dot{e}_3]$ is computed as

$$\dot{e} = J(\Theta)\dot{\Theta} \tag{1.19}$$

where $J(\Theta)$ is the Jacobian at the joint configuration Θ. To compute the joint commands $\dot{\Theta}$ that will move the end-effector in the desired spatial direction $\dot{e}$, (1.19) can be solved with respect to $\dot{\Theta}$ obtaining

$$\dot{\Theta} = J^{+}(\Theta)\dot{e} \tag{1.20}$$

$$J^{+} = (J^{\intercal}J)^{-1}J \tag{1.21}$$

$$\tag{1.22}$$

where J^{+} is the pseudo-inverse of J.

In summary their approach can be summarized as follows: (i) discretize the trajectory of the end-effector into small steps; (ii) compute at each step the spatial direction for the next movement, $\dot{e}$; (iii) compute the Jacobian at the current joint configuration, $J(\Theta)$; (iv) compute the pseudoinverse of the Jacobian, J^{+}; (v) compute the joint commands, $\dot{\Theta}$

The linearity of (1.19) ensures that the linear combination of known solutions will also give a valid solution. The continuity of the joint space along the path is ensured by the fact that the solution is found by computing the small increments in the joint angles.

The manipulator used was the arm of the iCub humanoid robot. They only focused on the end-effector position, ignoring its orientation. The four joints of interest are located at the shoulder (roll, pitch and yaw) and the elbow of the arm, with values taken in the intervals $\theta_1 \in [-75°, -15°]$, $\theta_2 \in [-75°, 15°]$, $\theta_3 \in [-10°, 50°]$, $\theta_4 \in [15°, 75°]$.

In order to mimic the approach described above they used a feed-forward network of Izhikevich neurons. The network consisted seven input layers L_i^{input}, and four output layers L_j^{output}. Each input and output layer was respectively composed of 1200 and 800 spiking neurons. The first four input layers $L_{i=1\ldots4}^{input}$, encode the information given by the joint angles $\theta_1, \ldots, \theta_4$. The last three input layers, $L_{i=5\ldots7}^{input}$, encode the spatial direction of the end-effector, $\dot{e}_1, \ldots, \dot{e}_3$, i.e. the direction that the end-effector should move to at the next time step. The four output layers, $L_{i=1,\ldots,4}^{output}$, represent the motor commands to the joints, $\dot{\theta}_1, \ldots, \dot{\theta}_4$. The input layers are connected all-to-all with the output layers. Each input neuron is connected with an excitatory and inhibitory synapse to each output neuron. All the synapses are plastic, i.e. they can be modified during learning.

The network encodes the joint angles $\theta_1, \ldots, \theta_4$ after discretizing them into bins with 5° resolution. Also the end-effector directions $\dot{e}_1, \ldots, \dot{e}_3$ are discretized using a 45° resolution. As a result the end-effector can move in 26 possible directions from the original position.

For the training phase, a set of robotic arm configurations were selected as "home" positions. During the action-perception cycle, sensory neurons were stimulated by proprioception, which also encoded into the network the current joint configuration. Motor neurons were randomly stimulated

using an Endogenous Random Generator (ERG), which sent random motor commands in the range of $[-5°, 5°]$ at each joint. The resulting joint commands moved the end-effector in a certain spatial direction which was observed and encoded into the network. The spatial position of the end-effector was computed with the forward kinematics equations during training, according to the D-H parameters of the iCub's arm.

Each iteration included four movements of the arm. For each movement, the joint positions Θ, the spatial direction movements $\dot{e}$, and the motor commands $\dot{\Theta}$, were encoded with a firing stimulus of 20 msec in the respective layers. Two consecutive movements were equally separated by 50 msec. In the simulation, a background nise of 3Hz was used in both the input and output layers in order to weaken the synaptic weights between uncorrelated neurons.

Chapter 2

Methods

2.1 Datasets description

The kind of datasets used are divided in two sets. The first set correspond to synthetic datasets of various family of signals, which were used to get familiar with the techniques applied and validate preliminary experiments, as well as to investigate new solutions of speculative nature. The second set consists of the joint trajectories of the robotic arm.

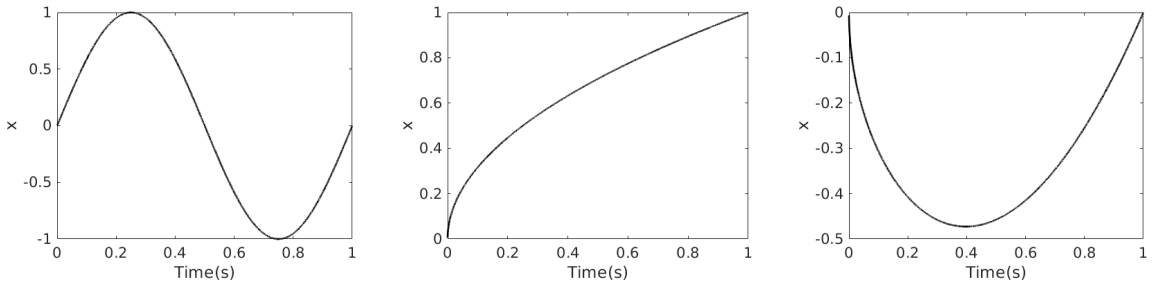
2.1.1 Synthetic datasets

Three different mono-dimensional signals were chosen as synthetic datasets: (i) a family of sinusoid of different frequencies f; (ii) a signal with positive values; (iii) a signal with negative values. The three signals are respectively defined in (2.1), (2.2), (2.3), and shown in Figure 2.1.

$$x_1(t) = \sin 2\pi f t \tag{2.1}$$

$$x_2(t) = \sqrt{t} \tag{2.2}$$

$$x_3(t) = t^2 - \sqrt{t} \tag{2.3}$$



(a) dataset 1 **(b)** dataset 2 **(c)** dataset 3

Figure 2.1: Collection of synthetic datasets. Each signal lasts for 1 second.

2.1.2 Trajectories datasets

The datasets used correspond to four joint trajectories generated with the *Kuka lwr iiwa* robotic arm in the NRP. Each dataset contains the *positions*, *velocities* and *accelerations* of the seven joints of the robotic arm, as well as the *time stamp* when they were sampled during the movement.

For each dataset, the start and end pose of the end-effector, as well as the length of the trajectory and the number of points sampled, are summarized in Table 2.1[1].

Table 2.1: Summary of the trajectories indicating the start and end pose of the end-effector, the length of the trajectory in milliseconds, and the number of samples taken during the simulation in the NRP. The four trajectories were performed from the same initial position and orientation of the end-effector, to four different end-effector positions. The end-effector orientation was maintained fixed at $(\alpha, \beta, \gamma) = (3.0934, -0.0304, 0.0163)$.

	Start Position	End Position	Length Signal (ms)	# Samples
Trajectory 1	(0.51, 0.09, 0.71)	(0.35, -0.2, 0.1)	2496	37
Trajectory 2	(0.51, 0.09, 0.71)	(0.75, -0.2, 0.1)	2590	28
Trajectory 3	(0.51, 0.09, 0.71)	(0.65, 0.33, 0.1)	2329	22
Trajectory 4	(0.51, 0.09, 0.71)	(0.35, 0.33, 0.1)	2301	25

The start position of the end-effector, along with the four end-points given by the final position of the end-effector are shown in Figure 2.2. The end-effector starting position is approximately centred above the polygon spanned by the four end-points.

[1] quaternion_xyzw = [-0.0242 -0.9995 0.0085 0.0150]; euler_xyz = quat2eul(quaternion_xyzw, 'XYZ');

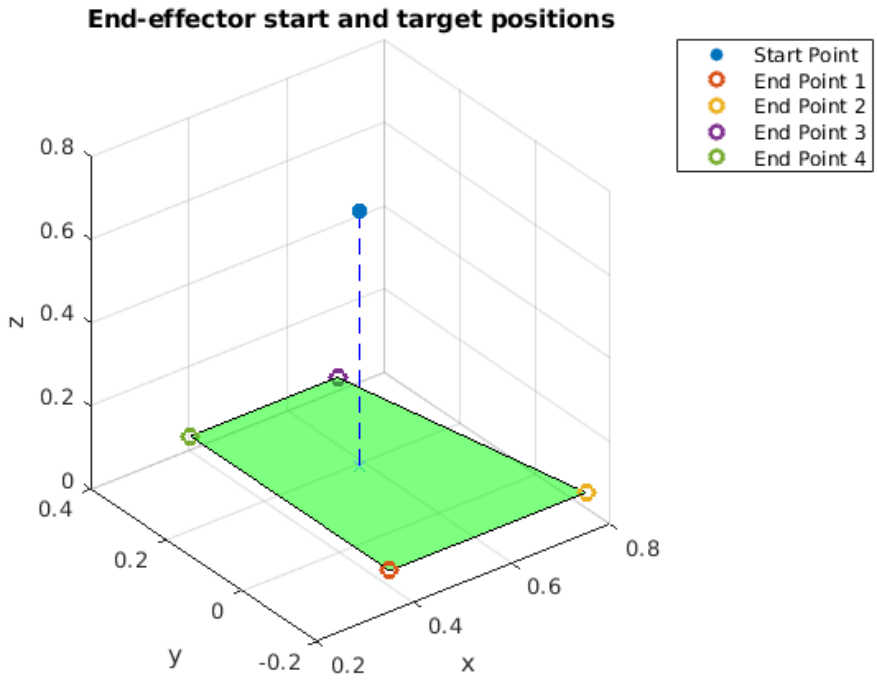
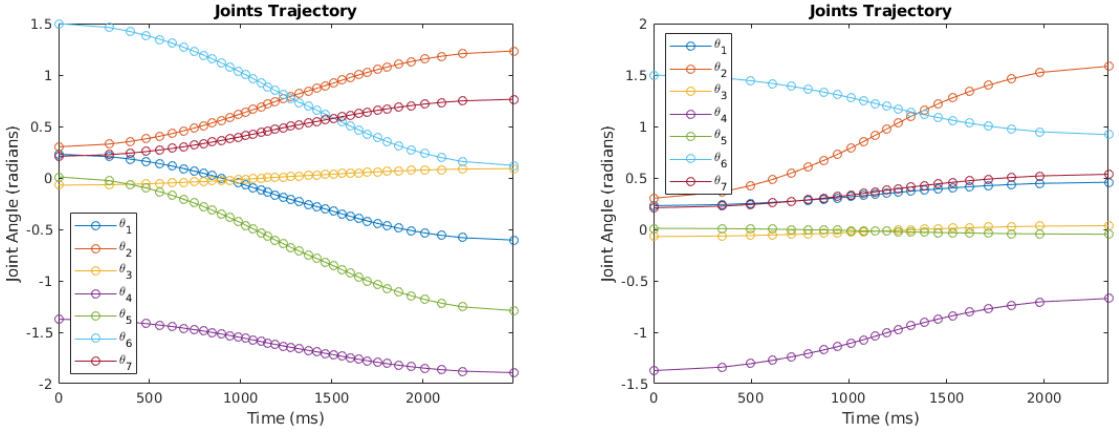


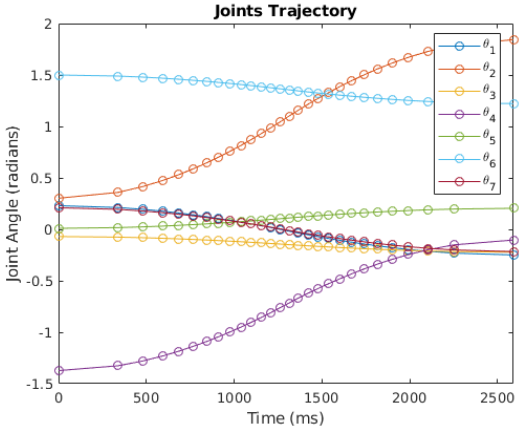
Figure 2.2: Start and end positions of the end-effector for each trajectory. The four end points lie in the same x-y plane. The end-effector start position is approximately above the centre of the rectangle spanned by the four end points.

From the time series of the joint positions are shown in Figure 2.3, it can be seen that the trajectories are quite related to each other. Indeed they correspond to similar movements.

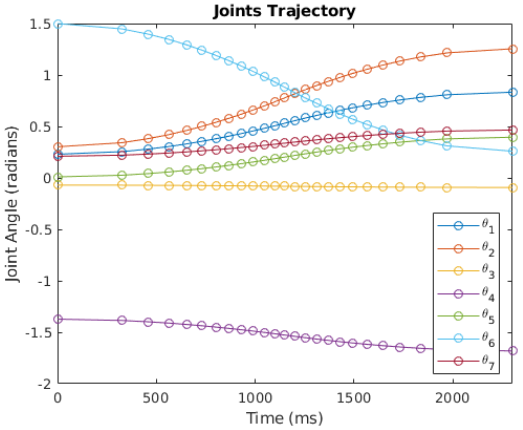


(a) trajectory 1

(b) trajectory 2



(c) trajectory 3



(d) trajectory 4

Figure 2.3: Four datasets corresponding to four different trajectories of the 7 joint positions of the robotic arm.

Dataset usage In order to use the datasets with the reservoir, only the *positions* and *time stamps* where needed. Let T_i be the length in milliseconds of the i-th trajectory. For each dataset, a third-degree polynomial was used to fit the trajectory positions at each time stamp. The resulting polynomial was then used to resample the trajectory points so that each dataset was composed of $\frac{T_i}{\delta t}$ equally spaced samples, where δt is the integration time constant.

In order to perform the interpolation between any given trajectories, those needed to have the same length. This was obtained by reshaping all the trajectories to have the same length, $T_{max} = \max_i T_i$, and then fit polynomials to obtain new datasets of $\frac{T_{max}}{\delta t}$ equally spaced samples[2].

[2]The target signal interpolation was only needed to compute the error with the output of the network.

2.2 Settling the network in an oscillatory regime prior to training

It was explored if the FORCE learning assumptions that the network must be in a chaotic regime prior to training could be relaxed. The aim was to make the network settle in a oscillatory/bursting regime, similar to what happens in the hyppocampus, before learning starts. In order to obtain a synchronized chattering behaviour, the following three stategies were explored

a) injection of an inhibitory current;

b) injection of a sinusoidal wave;

c) by means of Short Term Depression (STD)[3].

2.2.1 Injection of an inhibitory current

The first strategy used to generate oscillations in the network activity was to inject an inhibitory current to all or a fraction of the reservoir's neurons.

General network activity A measure of the general network activity (GNA) can be described by the following first-order differential equation implementing a low-pass filter

$$\tau_a \dot{y} = x - y \tag{2.4}$$

The spike distribution, $x(t)$, counts the number of spikes occurred in the network at time t

$$x(t) = \sum_{i=1}^{N} \delta(t - t^{(i)}) \tag{2.5}$$

where $t^{(i)}$ is the time of the latest spike of the i-th neuron, and the δ function is defined as follows

$$\delta(z) = \begin{cases} 1\ , & z = 0 \\ 0\ , & otherwise \end{cases} \tag{2.6}$$

The parameter τ_a defines how much the spikes are smoothed by the filter, and should also impact the duration and/or frequency of the bursts.

By rearranging the terms in equation (2.4) it is possible to derive a discretized measure of the GNA

$$y(t + \delta t) = \left(1 - \frac{\delta t}{\tau_a}\right) \cdot y(t) + \frac{\delta t}{\tau_a} \cdot x(t) \tag{2.7}$$

[3]Also known as *synaptic fatigue*.

Inhibitory term The GNA variable y can be used to implement an external inhibitory current, at each time step t, of the form $-\gamma y(t)$. The constant γ is a parameter to tune in accordance with the time constant τ_a. High values of γ would kill the network activity. The inhitory term was injected to a fraction $\rho \in [0, 1]$ of the neurons (usually 10% of the neurons), in order to break the symmetry of the neurons activities.

High-gain network The inhibitory term was injected to a network with high "gain" [6]. This was obtained by initializing the static weight matrix $\boldsymbol{\omega^0}$ from a Gaussian distribution with mean $\mu = 0$ and standard deviation $\sigma = g/\sqrt{pN}$, with connection probability $p = 0.1$. The parameter g is the gain of the network and was set to $g = 1.6$ (high-gain), which initializes the network on the edge of chaos.

rifai esperimenti

2.2.2 External sinusoidal wave

The second method used to generate oscillations in the network activity was to inject an external sinusoidal wave, to all or a fraction of the reservoir's neurons.

$$A \cdot \sin(\omega_{osc} \cdot t + \varphi_{osc}) \tag{2.8}$$

where A is the amplitude of the oscillations, ω_{osc} is the angular frequency, and φ_{osc} is the phase of the oscillations.

To replicate the oscillations that happen in the Hyppocampus, the sinuoidal wave must resemble the so called theta oscillations, which frequency takes value in the interval $[4, 10]$ Hz.

citazione?

ripeti esperimenti

2.2.3 Short-term depression

formalizza, ripeti esperimenti

2.3 High-dimensional temporal signals

In this section it is formalized how to construct and use HDTS signals as input to the reservoir during the training and replay phase, illustrating the different types of pulses used. Moreover it is explained how to construct various compressed version of the same HDTS in order to speedup the replay process, and how to manipulate given HDTS signals to achieve inverse replay. It is finally shown how to linearly combine different HDTS to obtain an interpolation and extrapolation of the learned target signals.

HDTS construction Let T be the duration of the target signal x in milliseconds, and $l = T/\delta t$ the corresponding length in simulation time steps. The HDTS is a matrix $H \in \mathbb{R}^{m \times l}$, where m is the dimensionality of the temporal signal, and l is its length. Each of the m rows of the matrix H is discretized into m sub-intervals of equal length T/m milliseconds (or l/m simulation time steps), defined as

$$I_j = [T \cdot \frac{j-1}{m}, T \cdot \frac{j}{m}] \qquad j = 1, \ldots, m \tag{2.9}$$

The HDTS is constructed in such a way that the j-th row of matrix H has a pulse of length T/m, with amplitude 1, centred at the sub-interval I_j. See Figure 2.4 for an example of HDTS matrix.

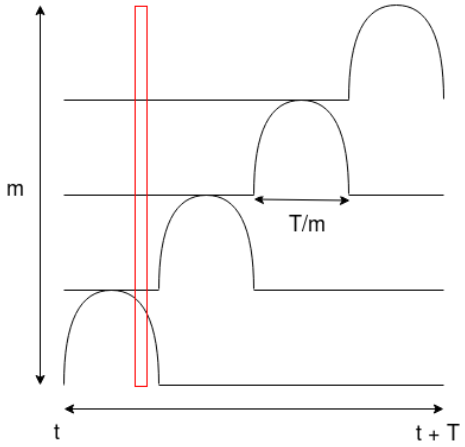
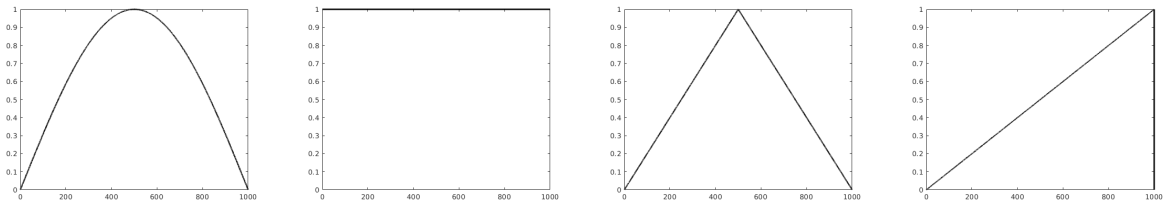


Figure 2.4: Representation of the HDTS matrix. In the example the matrix H has $m = 4$ rows, which correspond to m pulses of length T/m milliseconds centred in the sub-interval I_i for $i = 1, \ldots, 4$. At each time step, a column of H is selected (shown in red) and multiplied by a static weight matrix to construct the input current to the replay network.

HDTS usage The HDTS matrix H is used to create an additional input to the reservoir during both the training and replay phase. For a correct functioning of the learning process, the timings of the presentation of the target signal and the HDTS input must completely overlap. Let $i \in [1, l]$ be the index that iterates through the target signal. At each time step, a column of the matrix H is selected, $H_{:,i} \in \mathbb{R}^m$, and multiplied by the matrix $\eta_h \in \mathbb{R}^{N \times m}$, which components are uniformly drawned at random. The resulting vector, $I_{HDTS} = \eta_h H_{:,i} \in \mathbb{R}^N$, is used as external input current to the reservoir.

HDTS pulses The HDTS can be constructed using several types of pulses. When learning multiple target signals, each signal should be coupled with a different pulse type. In this project a neural network is trained to learn four joint trajectories, so four pulse types were used. The four types of pulses are shown in Figure 2.5.



(a) upper sinusoid **(b)** rectangular **(c)** isosceles triangle **(d)** rectangular triangle

Figure 2.5: The four types of pulses used to construct the HDTS matrices. Each pulse has a maximum amplitude of 1.

More formally, the upper sinuosoid and the rectangular pulse types for the i-th row of H, can be respectively defined as in (2.10) and (2.11) below

$$H_{i,:} = \begin{cases} |\sin(\frac{\pi m t}{T})| & , t \in I_i \\ 0 & , t \notin I_i \end{cases} \tag{2.10}$$

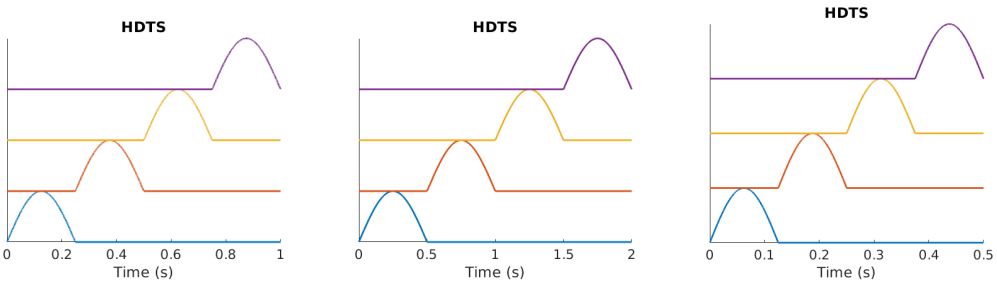
$$H_{i,:} = \begin{cases} 1 & , t \in I_i \\ 0 & , t \notin I_i \end{cases} \tag{2.11}$$

HTDS for accelerated replay Once the learning is finished, in order to replay the learned signal at different velocities, it is necessary to slightly modify the definitions of time sub-interval (2.9) and HDTS (2.10). A new factor $\rho > 0$ is introduced, that is used to compress or dilate time by multiplying the target signal length, obtaining $T \leftarrow T/\rho$. The new definitions of sub-interval (2.12) and HDTS pulse (2.13) become as below

$$I_i = [\frac{T}{\rho} \cdot \frac{i-1}{m}, \frac{T}{\rho} \cdot \frac{i}{m}] \quad , i = 1, \ldots, m \tag{2.12}$$

$$H_{i,:} = \begin{cases} |\sin(\frac{\pi m t}{T/\rho})| & , t \in I_i \\ 0 & , otherwise \end{cases} \tag{2.13}$$

In particular, with $\rho > 1$ the HDTS is compressed into a shorter time interval, while preserving the number of pulses m. As a result, each pulse is presented at a higher frequency and the network will replay the signal in accelerated velocity. Conversely with $\rho < 1$ the HDTS is dilated into a longer time interval. Each pulse has a lower frequency, and the target signal will be replayed at a slower pace. An example of HDTS compression and dilation is shown in Figure 2.6.



(a) $\rho = 1$ **(b)** $\rho = 0.5$ **(c)** $\rho = 2$

Figure 2.6: Representation of an HDTS with $m = 4$ pulses, and its compressed/dilated variants using compression factor ρ. 2.6a With $\rho = 1$ it is obtained the original HDTS signal, which has length $T = 1000$ milliseconds. 2.6b Using $\rho = 0.5$ the HDTS is dilated into an interval of double length $T = 2000$ milliseconds. 2.6c Using $\rho = 2$ the HDTS is compressed into an interval of half length $T = 500$.

HTDS for inverse replay In order to replay the learned signal in inverse order, the columns of matrix H should be flipped horizontally, to obtain a new matrix $H_{reverse}$ as illustrated below

$$H = \Big[H_{:,1}|H_{:,2}|\dots|H_{:,l}\Big] \Longrightarrow H_{reverse} = \Big[H_{:,l}|H_{:,l-1}|\dots|H_{:,1}\Big] \tag{2.14}$$

HDTS for the interpolation/extrapolation of multiple signals Multiple HDTS signals can be used to learn multiple target functions. Post training it is possible to replay a signal by reinjecting the corresping HDTS pulses that were used during training, and switch to another activity by injecting a different pulse.

Moreover, it is possible to linearly combine different HDTS pulse in order to create a new HDTS matrix that can be used as input during the replay phase. Let $H_{(j)}$ be the HDTS matrix associated with trajectory $x_{(j)}$. A new HDTS matrix H_{comb} is constructed as a linear combination of the known HDTS matrices

$$H_{comb} = \sum_{j=1}^{n} \alpha_j H_{(j)} \tag{2.15}$$

where α_j are the coefficients of the linear combination. In order to obtain an interpolation between the trajectories, it is necessary to use a convex combination, i.e. $\sum_{j=1}^{n} \alpha_j = 1$, otherwise any α coefficient can be used.

Chapter 3

Results

In the preliminary experiments it was choosen to use a family of sinusoidal waves as target signal with frequency f Hz and phase φ. The reservoir had $N = 2000$ neurons with $p = 0.1$ connection probability.

The simulation time $[t_0, t_{end}]$ is generally divided in three intervals. During $[t_0, t_{min}]$ the network stabilizes into a specific regime. The training takes place in during (t_{min}, t_{crit}) , where the readout weights are updated by means of RLS. The interval $[t_{crit}, t_{end})$ is used assess the quality of the learning scheme by measuring various metrics such as average firing rate and loss.

$G = 5 \times 10^3$, $Q \in \{0, 5 \times 10^3\}$. Integration time step $\delta t = 0.04$ ms.

The simulation performs $i_s = \lceil 1000/\delta t \rceil$ iterations per second. Therefore the period of a target signal with frequency f lasts $i_p = \lceil i_s/f \rceil$ iterations per period.

3.1 Pre-training

The first experiment consisted in performing a qualitative evaluation of the effect of applying an external sinusoid to generate oscillations in the network activity. To this end it was played with the amplitude A and the frequency f of the oscillations. Figure 3.1 shows a sample of 5 neurons membrane potential, the population activity and GNA variable for different values of $A \in \{0, 10, 50, 100, 200, 500\}$ and fixed frequency $f = 4Hz$. The same information is shown in Figure 3.2 for a fixed amplitude $A = 500$, varying the frequency $f \in \{4, 7, 10\}$.

Discussion From Figure 3.1 it can be seen that increasing A corresponds to wider bursts, more evident synchronization, sharper GNA oscillations and higher slightly higher GNA values. In particular there were no oscillations generated with $A \in \{0, 10\}$. With $A = 500$ there are perfectly synchronized, wide and clear bursts. However these could be entirely driven by the external signal given the high amplitude of the sinusoidal vawe. Instead with $A = 200$ there could be more interesting hidden neuron interactions as the bursts are not as synchronized.

From Figure 3.2 it is clear to see that higher values of f_{osc} correspond to more frequent oscillations per second, as expected.

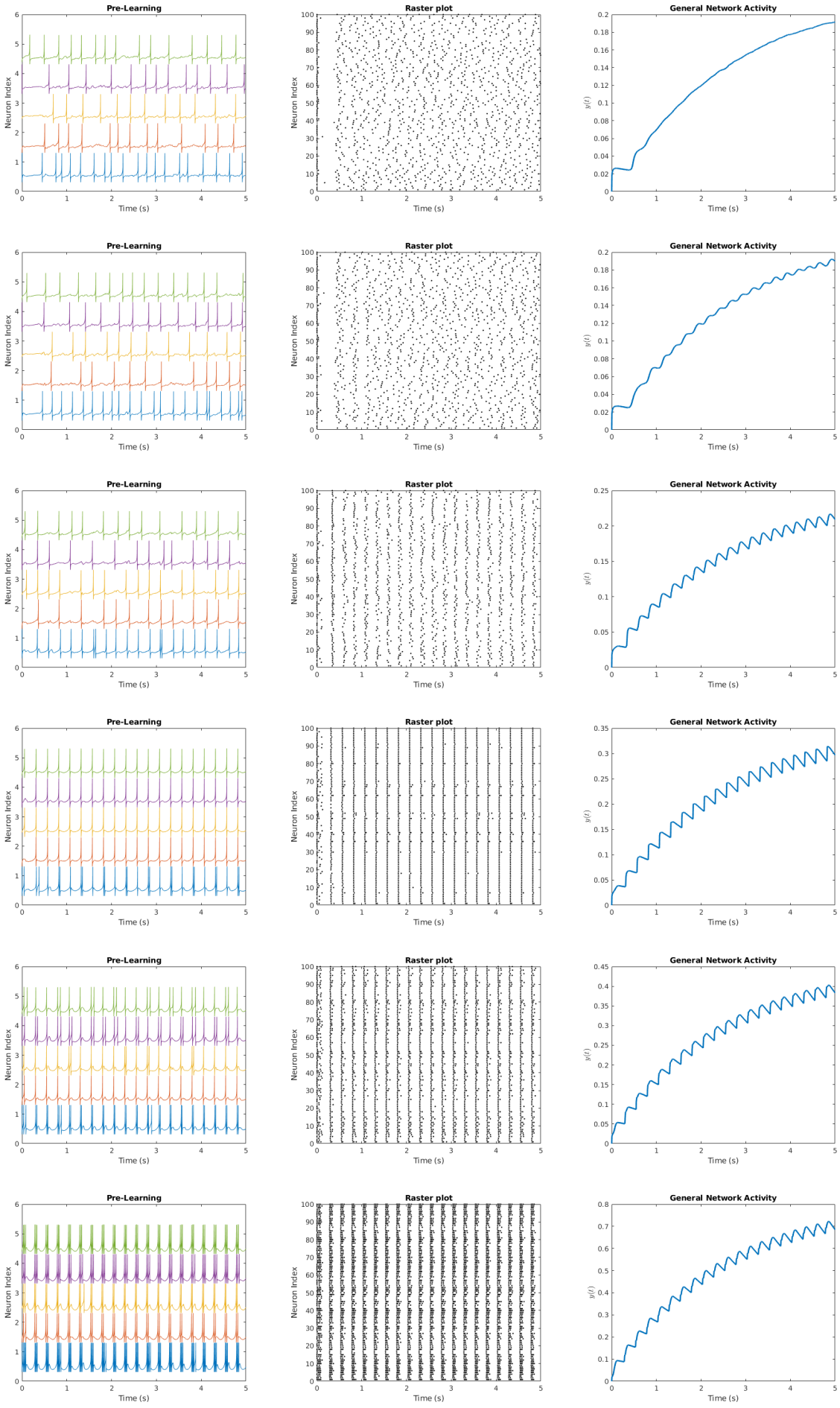


Figure 3.1: Pre-learning activity chaging amplitude of external oscillations, $A \in \{0, 10, 50, 100, 200, 500\}$.

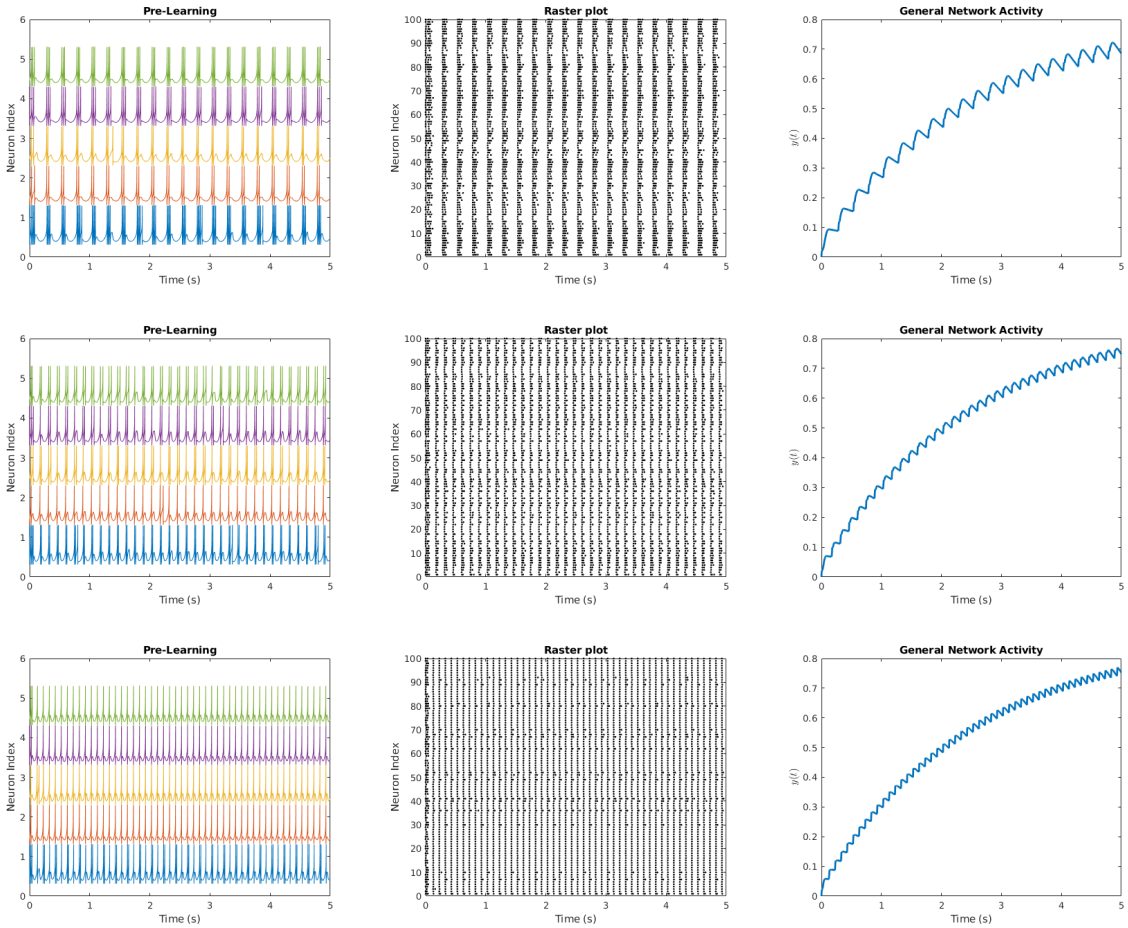


Figure 3.2: Pre-learning activity changing frequency of external oscillations $f \in \{4, 7, 10\}$ Hz.

3.2 Training

We refer to *continuous training* when the weight update is performed every $\Delta t = 20 \times \delta t = 0.8$ ms. We refer to *phase training* when the weights are updated at a specific point of the phase of the target signal, only once per cycle.

3.2.1 Changing the phase φ_{sin} of the teaching signal

Determine whether the phase of the teaching signal $\varphi_{sin} \in \{0\pi, \frac{1}{2}\pi\}$ has an impact on the performance. The network is first stabilized for 5 seconds of simulation. The network is then trained for an amount of time corresponding to 20 cycles of the target signal. This choice was made to better compare the performance obtained with target signals of different frequencies $f \in \{0.3, 0.5, 0.8, 1, 5, 7, 12\}$ Hz. The frequency of the external oscillations was fixed at $f = 4$ Hz, whereas it was varied the amplitude $A \in \{200, 500\}$.

When FORCE training was turned off, the simulation was carried out for additional 10 target signal cycles. For each setting, it is computed the Average Firing Rate and the Average Squared Error between the target signal and the approximant, over the 10 post training cycles. Each result is

obtained as the average of 10 repetitions. The Results are summarized in Table 3.1.

Discussion Our approach performs slightly better than the Clopath settings for low frequency target signals. However for low-frequency target signals the Clopath settings perform considerably better. The pattern of best results do not depend on the phase of the target signal φ_{sin}.

Table 3.1: Comparison of Average Firing Rate and Average Error obtained by Our approach and the Clopath settings. The external sinuoid amplitude was $A \in \{200, 500\}$. The teaching signal phase was $\varphi_{sin} \in \{0\pi, \frac{1}{2}\pi\}$. Training lasted for 20 cycles of the teaching signal. The results were averaged over 10 cycles after training, for 10 trials, plus/minus standard deviation shown. Best results shown in bold.

$\boldsymbol{\varphi_{sin} = 0\pi}$	Average Firing Rate			Average Error		
f_{sin}	$A = 200$	$A = 500$	Clopath	$A = 200$	$A = 500$	Clopath
0.3	24.61 (1.17)	26.53 (0.67)	29.00 (2.10)	**0.87 (0.12)**	0.91 (0.10)	0.97 (0.17)
0.5	28.09 (1.11)	28.29 (0.80)	29.78 (0.62)	1.06 (0.14)	**0.83 (0.10)**	0.99 (0.21)
0.8	27.48 (0.99)	27.87 (0.84)	28.27 (0.90)	0.98 (0.25)	**0.82 (0.22)**	0.84 (0.47)
1	27.62 (0.93)	28.71 (0.69)	28.81 (0.97)	**0.26 (0.35)**	0.31 (0.38)	0.66 (0.54)
5	27.39 (1.46)	26.93(1.92)	36.52 (0.85)	0.50 (0.34)	0.71 (0.27)	**0.05 (0.03)**
7	29.96 (1.83)	26.92 (1.15)	36.64 (1.00)	0.48 (0.23)	0.64 (0.14)	**0.05 (0.04)**
12	25.50 (3.03)	21.46 (1.42)	37.33 (0.67)	0.32 (0.15)	0.29 (0.06)	**0.02 (0.02)**

$\boldsymbol{\varphi_{sin} = \frac{1}{2}\pi}$	Average Firing Rate			Average Error		
f_{sin}	$A = 200$	$A = 500$	Clopath	$A = 200$	$A = 500$	Clopath
0.3	24.31 (0.84)	26.45 (0.96)	28.63 (1.41)	**0.86 (0.12)**	0.91 (0.11)	0.94 (0.12)
0.5	27.48 (1.30)	28.11 (0.79)	29.08 (1.04)	1.02 (0.13)	**0.90 (0.13)**	0.93 (0.11)
0.8	26.66 (1.30)	28.77 (1.12)	28.01 (1.09)	0.99 (0.15)	**0.93 (0.38)**	1.03 (0.25)
1	27.82 (0.66)	29.18 (0.65)	28.36 (0.89)	**0.28 (0.36)**	0.68 (0.62)	0.53 (0.42)
5	28.67 (0.85)	26.22 (1.39)	37.12 (0.74)	0.66 (0.39)	0.76 (0.28)	**0.02 (0.03)**
7	30.64 (2.34)	26.34 (2.14)	37.62 (0.71)	0.60 (0.36)	0.64 (0.17)	**0.02 (0.02)**
12	25.84 (2.07)	22.61 (0.90)	37.81 (0.82)	0.38 (0.16)	0.26 (0.01)	**0.01 (0.00)**

3.2.2 "Computing derivatives"

The standard Clopath settings were modified in order to make the network to learn to approximate first order derivatives with respect to an arbitrary period $t_{past} < t$.

The simulation interval $[t_{crit}, t_{end}]$ was split in two sub-intervals, or stages: $[t_{crit}, t_{crit_2})$ and $[t_{crit_2}, t_{end}]$. In addition, during the interval (t_0, t_{crit_2}) a smoothed version of the original target signal was injected as input current (see (3.2) below).

Let $A \in \mathbb{R}$ be the amplitude of the injected target signal, and p be the percentage of neurons that receive the additional current. The binary vector $I_{TR} \in \{0, 1\}^N$ represents the input target recipients, where each entry is 1 with probability p and 0 otherwise. Let $I_{TIC} \in \mathbb{R}^N$ be the input target indiv.

coefficient, where each entry is drawn from a gaussian distribution, $\mathcal{N}(\mu, \sigma)$, with mean μ and standard deviation σ. The input target coefficient $I_{TC} \in \mathbb{R}^N$ that is injected as current is defined as

$$I_{TC} = A \cdot I_{TR} \odot I_{TIC} \tag{3.1}$$

where $\odot$ is the component-wise product. The term (3.1) is used to augment the input current as below

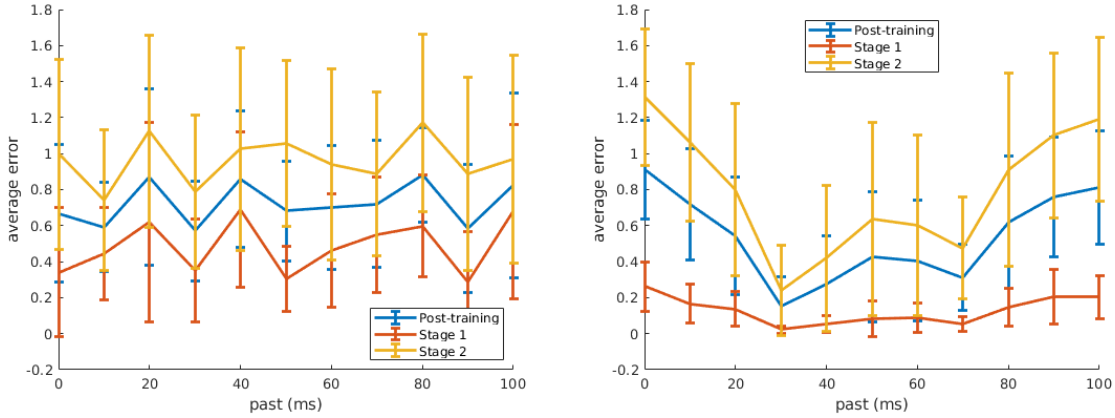
$$I = \begin{cases} IPSC + \eta\hat{x}(t - \delta_t) + BIAS + x(t) \odot I_{TC} & , \quad t \in (t_0, t_{crit_2}) \\ IPSC + \eta\hat{x}(t - \delta_t) + BIAS & , \quad otherwise \end{cases} \tag{3.2}$$

The average error was computed both for the stages $[t_{crit}, t_{crit_2})$ and $[t_{crit_2}, t_{end}]$, as well as across the entire post-training interval $[t_{crit}, t_{end}]$, with respect to previous observations of the target signal as

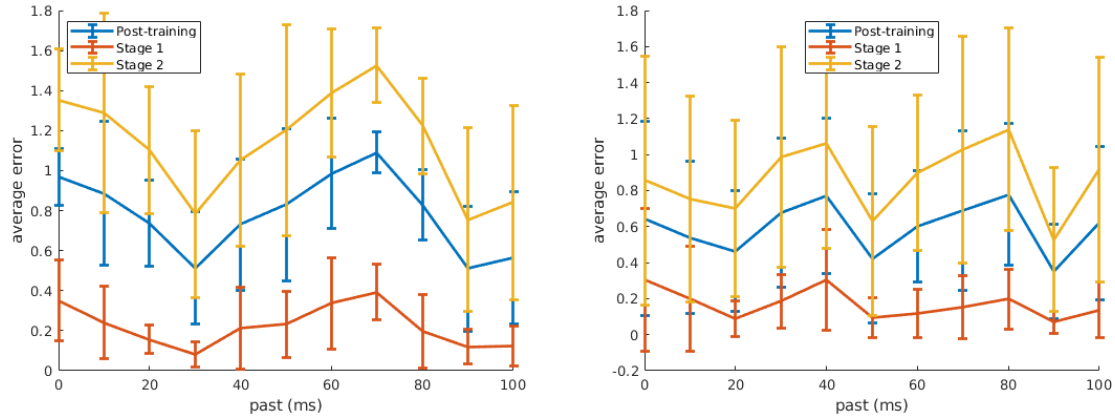
$$e = \hat{x}(t) - x(t - t_{past}) \tag{3.3}$$

The training settings are as follows. $t_{min} = 3$ seconds, $t_{crit} = t_{min} + \max\{5s, 20 \cdot i_p\}$, where i_p is the number of simulation iterations per period. $t_{crit_2} = t_{crit} + \max\{5s, 10 \cdot i_p\}$, $t_{end} = t_{crit_2} + \max\{8s, 10 \cdot i_p\}$. $A = 30$, $p = 0.2$. $\mu = 0.5$, $\sigma = 0.5$.

The results are shown in Figure 3.3 for $t_{past} \in [0, 100]$ ms and target signal frequency $f \in \{1, 5, 7, 12\}$.



(a) $f = 1$ Hz **(b)** $f = 5$ Hz



(c) $f = 7$ Hz **(d)** $f = 12$ Hz

Figure 3.3: Squared error vs $t_{past} \in [0, 100]$ computed across the entire post-training period $[t_{crit}, t_{end}]$, stage one $[t_{crit}, t_{crit_2})$, and stage two $[t_{crit_2}, t_{end}]$. Target signal frequency $f \in \{1, 5, 7, 12\}$.

3.2.3 Discretization of the target signal

Starting from the standard Clopath settings, it was experimented with target discretization and smoothing. Let $x(t) \in \mathbb{R}$ be the original unidimensional target signal. Let $x_{01}(t) \in \{0, 1\}^m$ be the one-hot representation of the target signal after its discretization into m uniform bins [1]. Each bin b_i, for $i = 1, \ldots, m$, accounts for a certain contiguous subset of the target signal codomain, entirely defined by the interval $[b_i^L, b_i^H]$, which correspond to the lower and upper extreams of the signal values for that bin. Let $x_s(t) \in \mathbb{R}^m$ be a smoothed version of the one-hot encoded target obtained after the application of a gaussian filter.

One-hot input, continuous m-dimensional output (WRONG) The network takes as input an m-dimensional one-hot encoded signal $x_{01}(t) \in \{0, 1\}^m$ and outputs an m-dimensional continuous vector $\hat{x}(t) \in \mathbb{R}^m$. The loss is computed the sum of squared errors: $\mathcal{L}(t) = \sum_{i=1}^{m} (x_{01_i}(t) - \hat{x}_i(t))^2$. (see Table 3.2)

[1] The target discretization was achieved by means of the Matlab *discretize()* built-in function.

	$m = 10$	
f_{sin}	Average Firing Rate	Average Error
0.3	48.02 (3.26)	2.46 (0.34)
0.5	46.16 (3.73)	2.15 (0.29)
0.8	42.72 (4.09)	1.74 (0.24)
1	40.53 (3.82)	1.58 (0.28)
5	40.88 (1.80)	1.29 (0.20)
7	38.38 (3.29)	1.36 (0.13)
12	23.31 (1.69)	1.05 (0.04)

Table 3.2: Target discretization, $m = 10$ bins. One-hot input, continuous m-dimensional continuous output

One-hot input, one hot target, multi-class misclassification error The network takes as input the one-hot encoded signal $x_{01}(t) \in \mathbb{R}^m$ and outputs an m-dimensional continuous vector $\hat{x}(t) \in \mathbb{R}^m$. The prediction is then converted to a one-hot vector, $\hat{x}(t) \to \hat{x}_{01}(t) \in \{0,1\}^m$. The loss is computed as the misclassification error: $\mathcal{L}(t) = \mathbf{1}(x_{01}(t) == \hat{x}_{01}(t))$, where $\mathbf{1}(\cdot)$ is the indicator function. (see Table 3.3)

	$m = 10$	
f_{sin}	Average Firing Rate	Average misclassification Error
0.3	46.07 (3.21)	0.93 (0.03)
0.5	44.05 (9.05)	0.86 (0.04)
0.8	41.89 (4.40)	0.87 (0.03)
1	38.61 (2.05)	0.90 (0.03)
5	40.63 (1.93)	0.76 (0.19)
7	38.43 (1.90)	0.88 (0.05)
12	22.54 (0.71)	0.92 (0.02)

Table 3.3: Target discretization, $m = 10$ bins. One-hot input, one hot target - multi-class misclassification error

One-hot input, maxout target It was tried to reconstruct the monodimensional target $x(t)$ from the m readouts. The network takes as input the one-hot encoded signal $x_{01}(t) \in \mathbb{R}^m$ and outputs an m-dimensional continuous vector $\hat{x}(t) \in \mathbb{R}^m$. The loss is computed as the squared error between the original target and the maximum of the readouts: $\mathcal{L}(t) = (x(t) - \max_i \hat{x}_i(t))^2$. (see Table 3.4)

	$m = 10$	
f_{sin}	Average Firing Rate	Average Error
0.3	46.07 (3.21)	1.46 (0.31)
0.5	44.05 (9.05)	1.45 (0.39)
0.8	41.89 (4.40)	1.26 (0.33)
1	38.61 (2.05)	0.92 (0.09)
5	40.59 (1.87)	1.07 (0.06)
7	38.43 (1.90)	1.03 (0.08)
12	22.54 (0.71)	0.60 (0.01)

Table 3.4: Target discretization, $m = 10$ bins. One-hot input, one hot target - max-out

One-hot + smoothing input, maxout target The network takes as input a smoothed version of the one-hot encoded signal $x_{01}(t) \in \mathbb{R}^m$, which is constructed as follows. Let $x_0 = [\mu_1, \dots, \mu_m]^\intercal$ be the vector of bin centres, where $\mu_i = \frac{b_i^L + b_i^R}{2}$. It is constructed an m-dimensional vector $\vec{x}(t)$ which elements are m replicas of the value of the monodimensional target signal $x(t)$, $\vec{x}(t) = [x(t), \dots, x(t)]^\intercal \in \mathbb{R}^m$. Then the smoothed m-dimensional target signal $x_s(t)$ is constructed by applying a gaussian kernel of the form:

$$x_s(t) = e^{-\frac{(\vec{x}(t) - x_0)^2}{\sigma_s}} \qquad \forall t \tag{3.4}$$

where σ_s is the standard deviation of the smoothing kernel. The network outputs an m-dimensional continuous vector $\hat{x}_s(t) \in \mathbb{R}^m$. The loss is computed as the squared error between the maximum of the readouts and the original target signal: $\mathcal{L}(t) = (x(t) - \max_i \hat{x}_{s_i}(t))^2$. (see Table 3.5 and Table 3.6)

	$m = 10$	
f_{sin}	Average Firing Rate	Average Error
0.3	150.24 (62.55)	55.37 (67.87)
0.5	37.93 (30.34)	1.51 (1.16)
0.8	17.34 (8.53)	0.71 (0.18)
1	38.55 (24.00)	1.03 (0.36)
5	80.56 (1.62)	1.49 (0.01)
7	80.62 (1.70)	1.49 (0.01)
12	77.46 (8.52)	1.43 (0.17)

Table 3.5: Target discretization, $m = 10$ bins. one-hot + smoothing input ($\sigma_s = 1$), maxout target

	$m = 10$	
f_{sin}	Average Firing Rate	Average Error
0.3	19.95 (3.49)	0.56 (0.19)
0.5	17.75 (9.50)	0.55 (0.04)
0.8	1.95 (6.18)	0.51 (0.03)
1	10.64 (9.19)	0.53 (0.03)
5	12.31 (0.19)	0.51 (0.00)
7	11.46 (0.28)	0.51 (0.00)
12	11.02 (0.52)	0.51 (0.00)

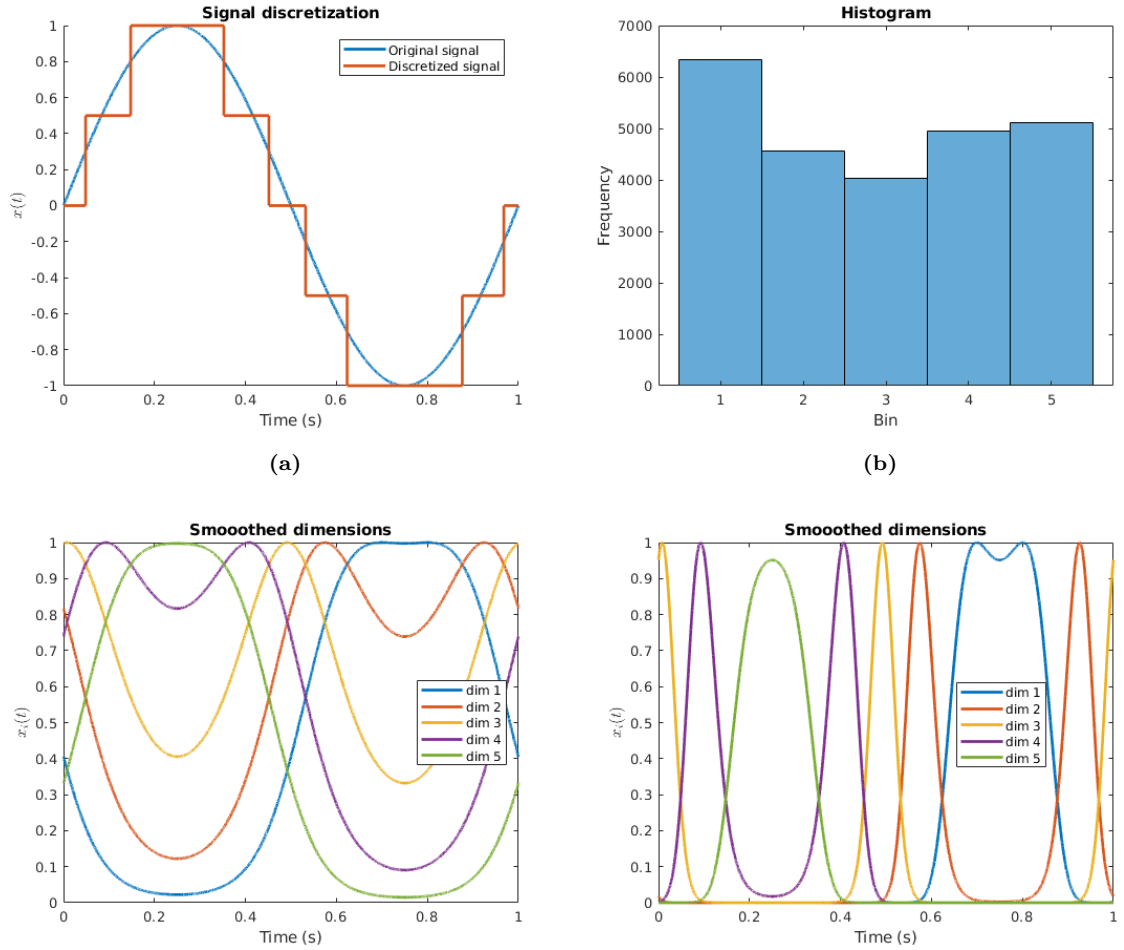
Table 3.6: Target discretization, $m = 10$ bins. one-hot + smoothing input ($\sigma_s = 0.001$), maxout target

One-hot + smoothing input, maxout target, selective feedback The experiment described in the previous paragraph is augmented by modifying the feedback term of the FORCE training. In the Clopath settings, at each time step t a feedback term is reinjected as input current, defined as $\eta\hat{x}_s(t - \delta t) \in \mathbb{R}^N$. In this experiment the feedback term was modified by substituting the matrix η with η', so that each dimension of the output $\hat{x}_s(t-\delta t)$ is reinjected only to a subset of N/m neurons. In this way different neurons will learn to fire when the signal belongs to different bins. Figure 3.4 illustrates the procedure for creating the matrix η'.

$$\eta\hat{x}_s = \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \\ * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \begin{bmatrix} \hat{x}_{s_1} \\ \hat{x}_{s_2} \\ \hat{x}_{s_3} \end{bmatrix} \longrightarrow \eta'\hat{x}_s = \left[\begin{array}{ccc} * & 0 & 0 \\ * & 0 & 0 \\ \hline 0 & * & 0 \\ 0 & * & 0 \\ \hline 0 & 0 & * \\ 0 & 0 & * \end{array}\right] \begin{bmatrix} \hat{x}_{s_1} \\ \hat{x}_{s_2} \\ \hat{x}_{s_3} \end{bmatrix}$$

Figure 3.4: Example of the construction of the selective feedback matrix η' from η, using $N = 6$ neurons and $m = 3$ bins. The rows of matrix η' are virtually divided in m blocks. In the i-th block, all the values are set to 0 except for the i-th row subset, so that each dimension $\hat{x}_{s_i}$ only influences one cluster of neurons.

do some experiments, simple data manipulation



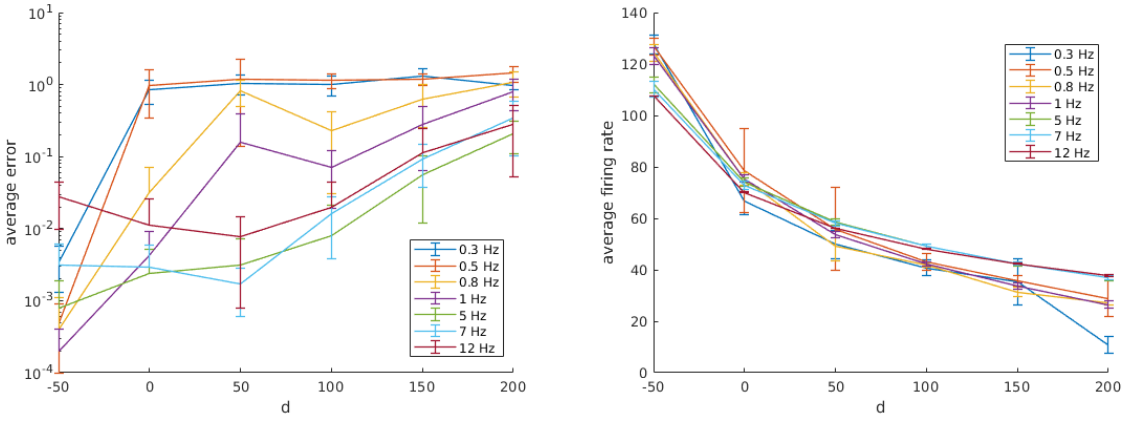
(a) **(b)**

(c) $\sigma = 1$ **(d)** $\sigma = 0.05$

Figure 3.5: Example of target discretization in $m = 5$ bins. An m-dimensional one-hot encoded target signal is created and then smoothed with gaussian pulses centred at in the middle of each sub-interval. For a given sample, the value in each dimension represents the membership value to a particular bin. The standard deviation σ of the gaussians determine how much the intervals overlap.

3.3 Control the replay velocity changing the spike frequency adaption

Original Clopath settings, change $d \in \{-50, 0, 50, 100, 150, 200\}$. See Figure 3.6.



(a) **(b)**

Figure 3.6: Average squared error and average firing rate as a function of parameter $d \in \{-50, 0, 50, 100, 150, 200\}$, for target sinusoids of frequency $f \in \{0.3, 0.5, 0.8, 1, 5, 7, 12\}$.

$N = 100$ because it is faster and qualitatively you get the same results, $f = 1$ Hz, $\delta_t = 0.2$ so it is faster, start with $d = -50$ then change it every 10 seconds post-learning during the interval $[t_{crit}, t_{end}]$ every 10 seconds change parameter, $t_{min} = 3$ seconds, $t_{crit} = 10$ seconds, $t_{crit_2} = 15$ seconds, $t_{end} = 135$ seconds. (i.e 2 minutes of parameter change)

In Figure 3.7 are shown the target approximant and spectrogram for three cases: 3.7a $d \leftarrow d + 20$ for 2 minutes, 3.7b $d \leftarrow d - 5$ for 2 minutes, 3.7c $d \leftarrow d + 20$ for the fist minute and $d \leftarrow d - 20$ in the second minute.

In Figure 3.7 3.8a $a \leftarrow a + 0.005$ for 2 minutes, 3.8b $a \leftarrow a - 0.00025$ for 2 minutes, 3.8c $a \leftarrow a + 0.005$ for the fist minute and $a \leftarrow a - 0.005$ in the second minute.

put raster plots, voltage and recovery traces, and maybe phase portraits with d=-50, d=0, d= 50/200

describe in a more abstract way

does the network still start in a chaotic regime when using d=0 or negative?

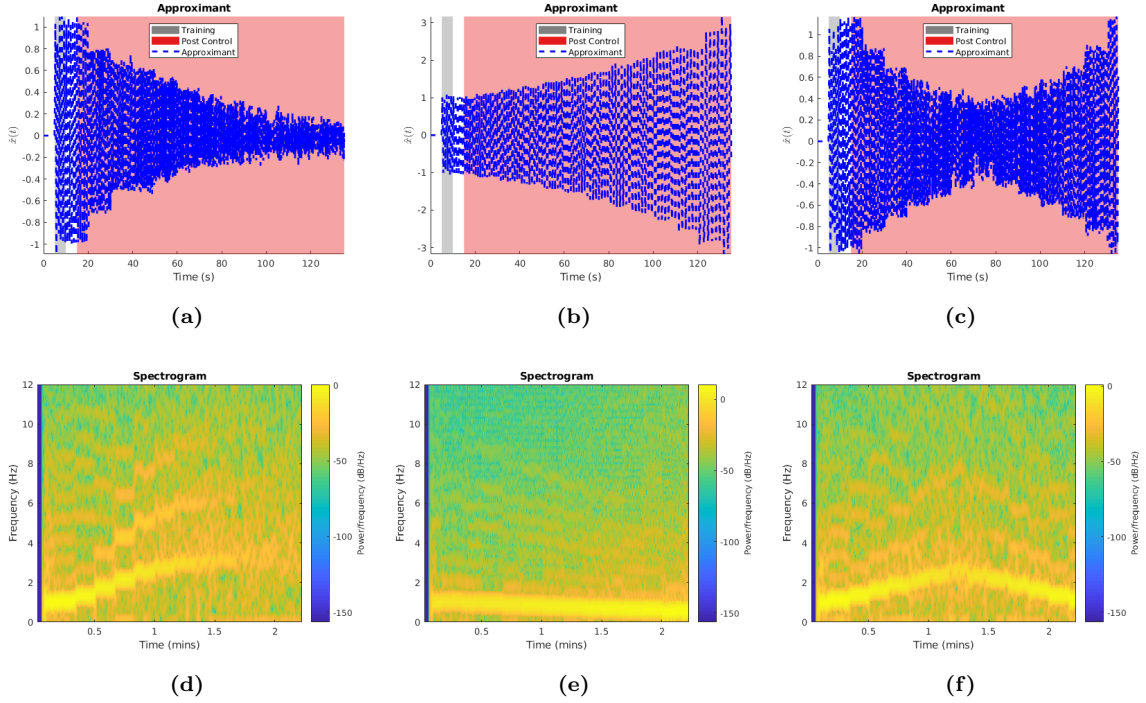


Figure 3.7: Changing the spiking frequency adaptation (parameter d) post training. The network is set onto a chaotic attractor during the first 5 seconds of simulation (first white region). Training takes place for 5 seconds (gray region). Post training, the network reproduces the learned signal for additional 5 seconds (second white region). The parameter d is linearly changed every 10 seconds for 2 minutes (red region): 3.7a d increased; 3.7b d decreased; 3.7c d increased then decreased.

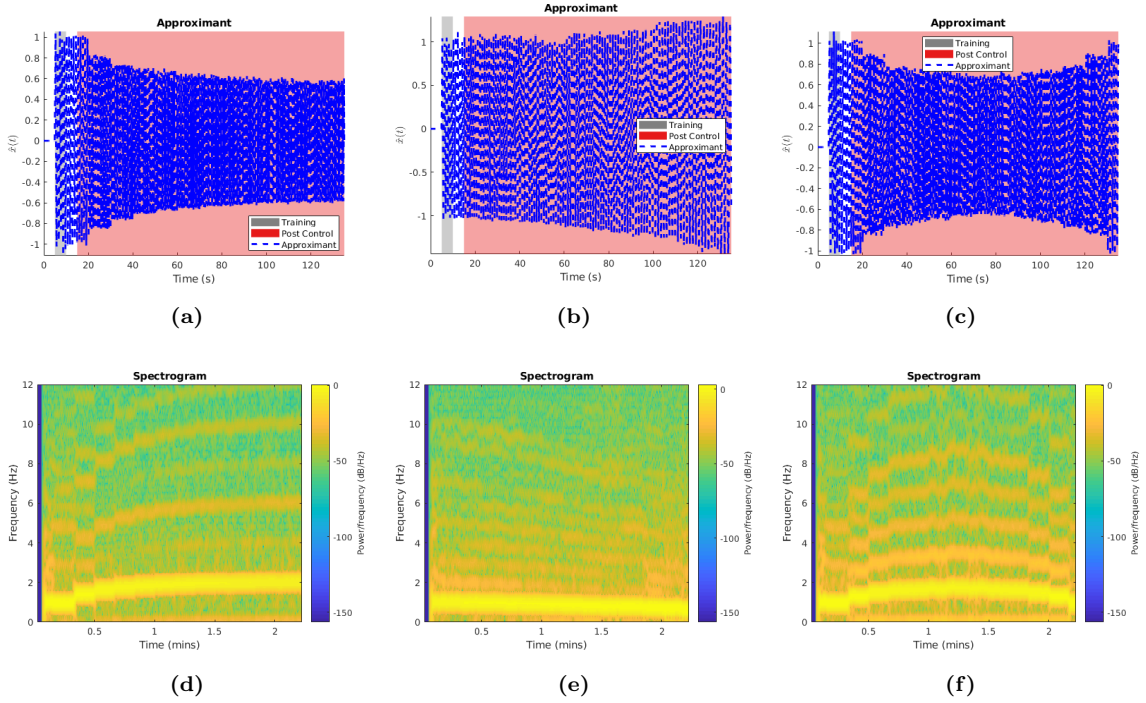


Figure 3.8: Changing the spiking frequency adaptation (parameter a) post training. The network is set onto a chaotic attractor during the first 5 seconds of simulation (first white region). Training takes place for 5 seconds (gray region). Post training, the network reproduces the learned signal for additional 5 seconds (second white region). The parameter a is linearly changed every 10 seconds for 2 minutes (red region): 3.8a a increased; 3.8b a decreased; 3.8c a increased then decreased.

3.4 Using high-dimensional temporal signals

This section reports the results of the experiments performed on the synthetic datasets, with the goal of getting familiar with using the HDTS techniques discussed in section 2.3, uncovering their potential and observing possible limitations.

3.4.1 HDTS speeds up the replay

To verify that the compression and dilation of the HDTS correctly accelerates and decelerates the replay phase, the network was trained to learn one period of a sinuoid. The choice of this supervisor was made because it is clear to observe the change of the replay frequency from the spectrogram of the network output. Post training the network successfully replayed the signal. After few repetitions, the HDTS was first compressed then dilated, resulting in a correct acceleration and deceleration of the signal replay, see Figure 3.9.

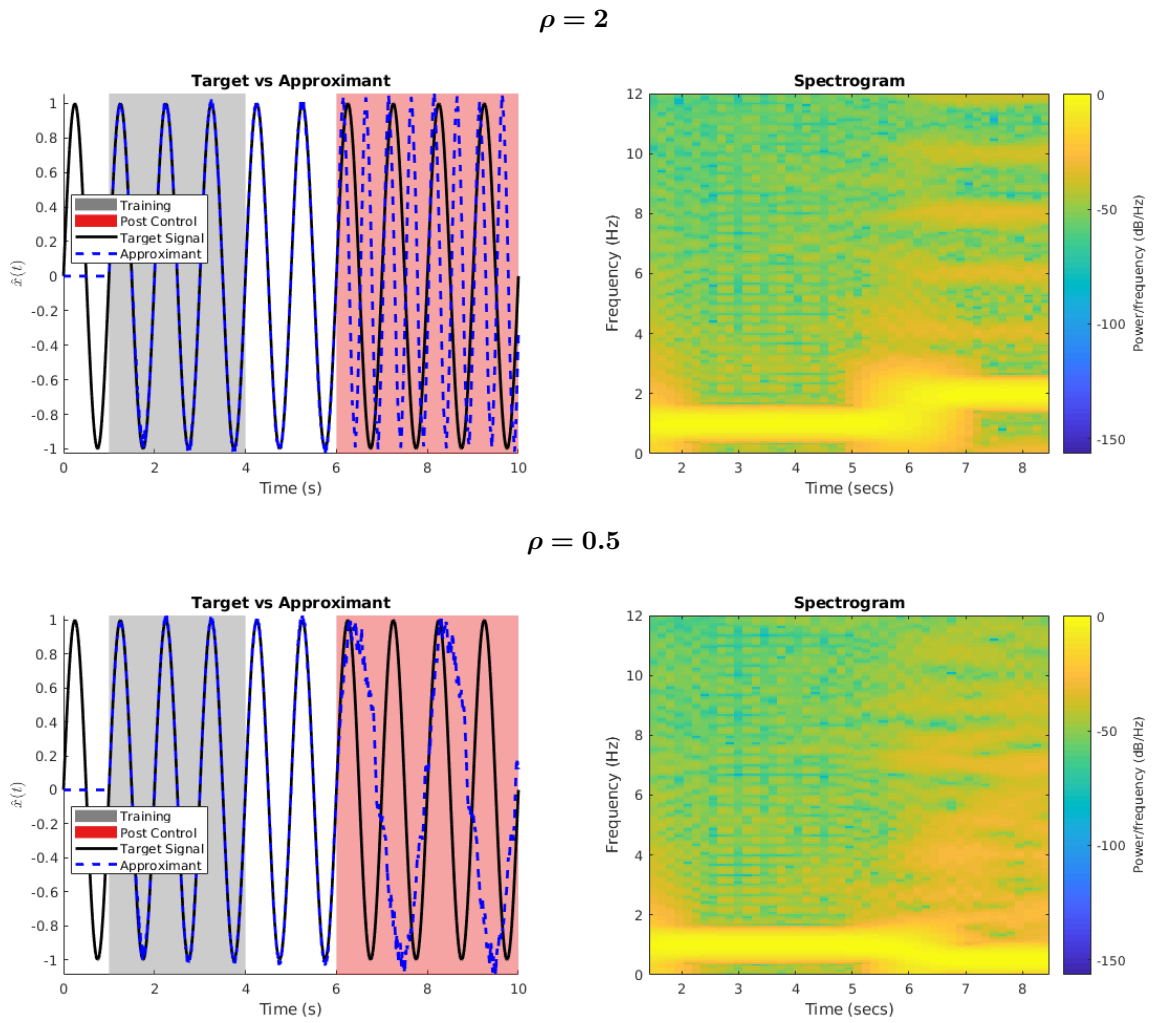


Figure 3.9: Acceleration ($\rho = 2$) and deceleration ($\rho = 0.5$) of the replay of a sinuoid with original frequency $1Hz$. Despite the noise and amplitude errors, the network properly adjusts the velocity of the replay. The original signal is shown with a black line also during the acceleration/deceleration phase, to better see the difference with the network output.

The trend of the network output visibly resembles the target signal, although it presents some high-frequency oscillations. This does not constitute a major limitation, because the high-frequency oscillations could be removed applying a low-pass filter. The output also presents a marginal amplitude error. The same behavior was also verified with non-periodic target signals (not shown).

3.4.2 Inverse replay

It was verified that the FORCE method combined with HDTS allows the inverse replay of a signal. This could be useful for a robotic arm that wants to perform an activity in a certain position and then go back to the original position.

It was trained a network for three repetitions of the signal. Post training the network replayed the learned signal for one repetition. After that, it was injected the reversed version of the HDTS, $H_{reverse}$. As is shown in Figure 3.10, the network inverted the replay successfully, although with some loss in accuracy as reported in [9].

redo picture with colored icrit2 part

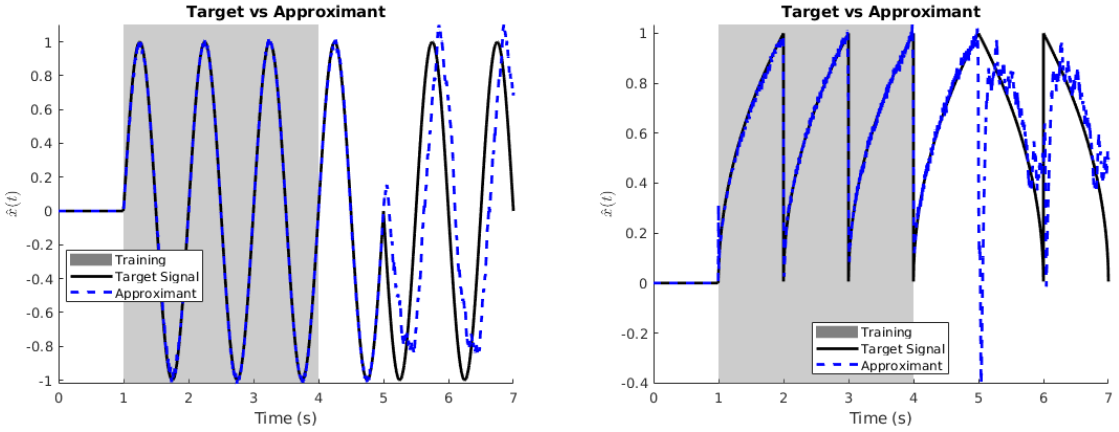
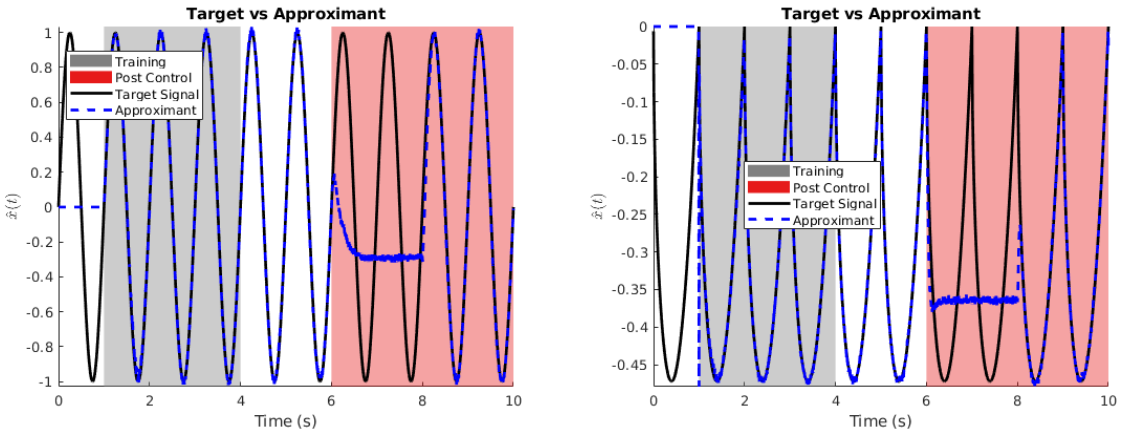


Figure 3.10: The inverse application of the HDTS makes the network replay the signal backwards.

3.4.3 HDTS triggers the replay

For the purpose of this project, it was interesting to asses whether it is possible to trigger the replay of the target signal by turning on/off the HDTS when desired. In order to obtain this effect, the HDTS was turned turned off for 2 seconds post training. As a result the network immediately stopped the replay of the target signal. When the HDTS was reinjected again, the network successfully restarted the replay, see Figure 3.11. The experiment was performed for two supervisors.

if you set BIAS=0 when HDTS=0 the network stops spiking.



(a) dataset 1 **(b)** dataset 3

Figure 3.11: HDTS triggers the replay of the signal. At 6 seconds the HDTS is removed and the network immediately stops reproducing the signal. The HDTS is reinjected after 2 seconds triggering the immediate replay of the singal.

3.4.4 Multiple HDTS to learn multiple signals

In the previous section it was shown that turning on and off the injection of the HDTS triggers the replay of the target signal. The next question to answer was whether it is possible to use multiple HDTS to learn multiple outputs, and make the network switch between activities when desired. The

experiment was done for two supervisors, hence two different HDTS pulse types were used (positive sinusoid and rectangular pulse) and combined into an HDTS train, see Figure 3.12a. During training the two supervisors were presented to the network a couple of times intermittently, along with the corresponding HDTS pulse type. Post training both the HDTS were removed, and then reinjected separately at different times to make the network replay the different target signals, see Figure 3.12b. The network successfully learned the two signals as a function of the two HDTS and was able to repeat them when triggered with the proper HDTS. However the network output presented a strange upward excursion when the second HDTS was turned off. Changing the order in which the HDTS were proposed or using different pulse types resulted in analogous behaviours.

It was also wondered what would happen using only one HDTS pulse type to learn two different signals. It was speculated that the network could remember the last signal presented during training. But as expected, using the same type of HDTS to learn two different signals completely destroyed the learning process, see Figure 3.13.

If you set BIAS=0 when HDTS=0 the network does not completely stop spiking.

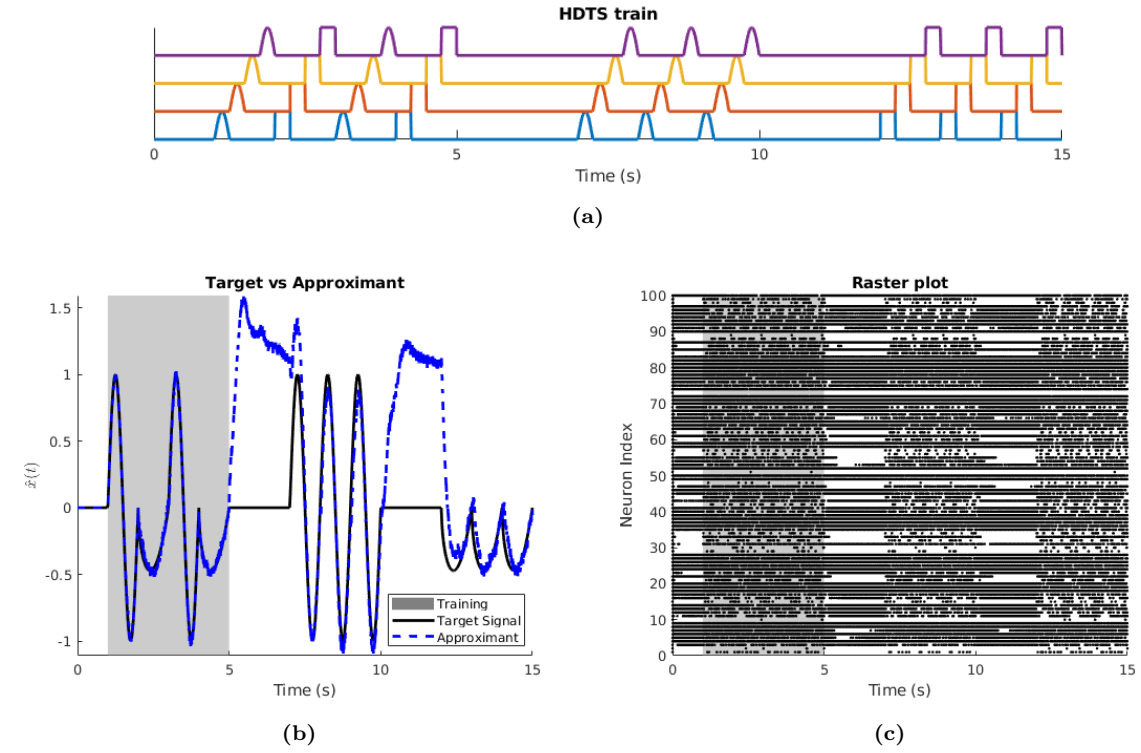


Figure 3.12: Learning two different target signals. 3.12a Example of HDTS train using sinusoidal and rectangular pulses to learn two different objective signals. Flat line segments indicate the absence of HDTS. Post training, changing the type of HDTS the network changes activity replaying a different signal. The removal of the HDTS at time 5 seconds and 10 seconds results in a strange excursion of the network output.

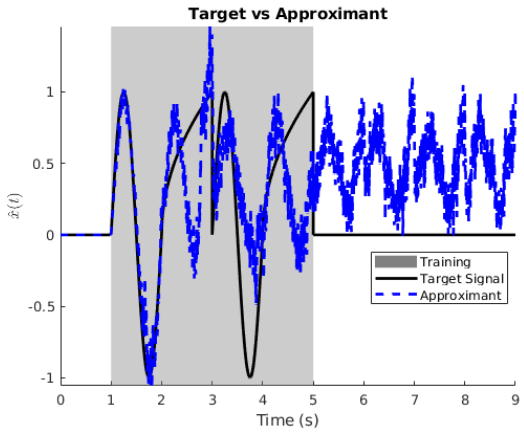


Figure 3.13: Using only one HDTS pulse type to learn two different signals completely destroys the learning process. For convenience, the target signal is shown as a flat black line post training.

not clear what happens when we remove the HDTS.

3.4.5 Signal interpolation and extrapolation

Since the network could learn the many-to-many mapping from different HDTS to different target signals, it was wondered what would happen when a linear combination of two HDTS was used post training. The expectation was that the network would mimic the combination of HDTS, and output a combination of the previously learned signals. The distinction between *interpolation* (i.e. using a convex combination) and *extrapolation* was of particular interest for our scope.

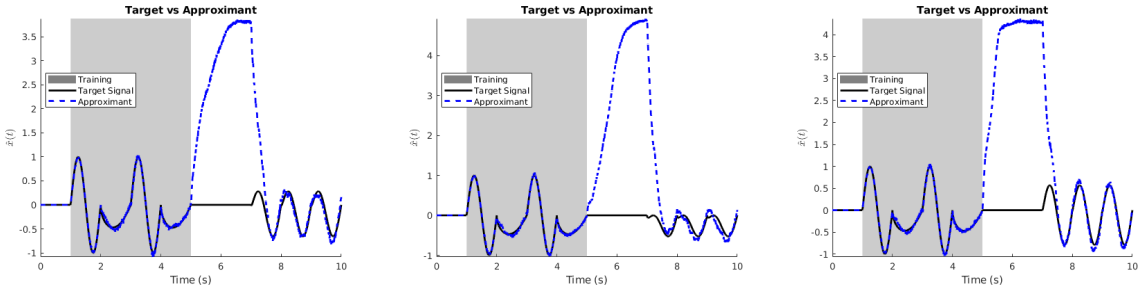
how to define rest? keep robot in a steady position, no spikes?

The network was trained to learn two target signals x_1 and x_2, guided by two different HDTS, H_1 and H_2. Post training it was injected a linear combination of the two HDTS, $H_{comb} = \alpha_1 H_1 + \alpha_2 H_2$. The network output was monitored and compared against the linear combination of the target signals, $x_{comb} = \alpha_1 x_1 + \alpha_2 x_2$. The results for both signal interpolation and extrapolation are shown in Figure 3.14.

It is clear to see that in the interpolation case, the network output well resembled the convex combination of the target signals x_{comb}. Conversely the extrapolation task was harder to solve.

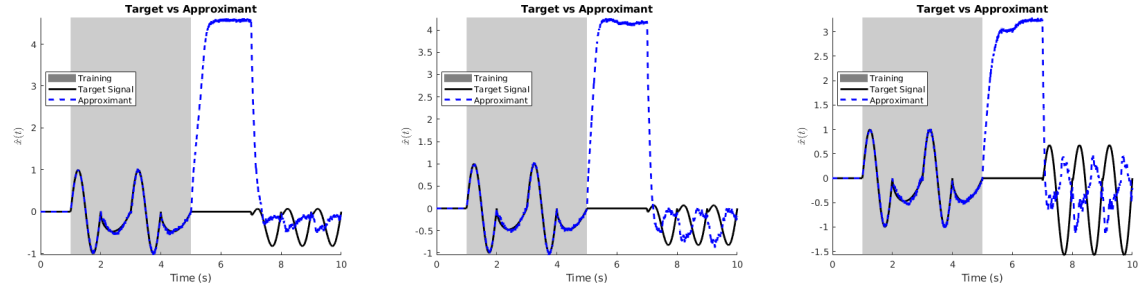
for interpolation the amplitude of the HDTS remains one as the original two. But with extrapolation the amplitude changes, maybe

Interpolation



(a) $\alpha_1 = 0.5$ $\alpha_2 = 0.5$ **(b)** $\alpha_1 = 0.3$ $\alpha_2 = 0.7$ **(c)** $\alpha_1 = 0.7$ $\alpha_2 = 0.3$

Extrapolation



(d) $\alpha_1 = 1$ $\alpha_2 = 0.5$ **(e)** $\alpha_1 = 0.5$ $\alpha_2 = 1$ **(f)** $\alpha_1 = 1.2$ $\alpha_2 = 1.2$

Figure 3.14: Linear combination of two HDTS. The network successfully interpolates between two HDTS, but fails the extrapolation.

Finally, the interpolation task was studied in more detail to see if it was possible to interpolate between three or four signals. The network was trained to learn three different target signals x_1, x_2 and x_3, guided by three different HDTS types respectively. The network was successfully able to reproduce the single signals post training. However the interpolation failed as the network was not even able to perform a linear combination of any two signals, see Figure 3.15. Probably because the target signals were completely unrelated to each other, the network confused itself.

which kind of signals can be interpolated? which kind can be extrapolated?

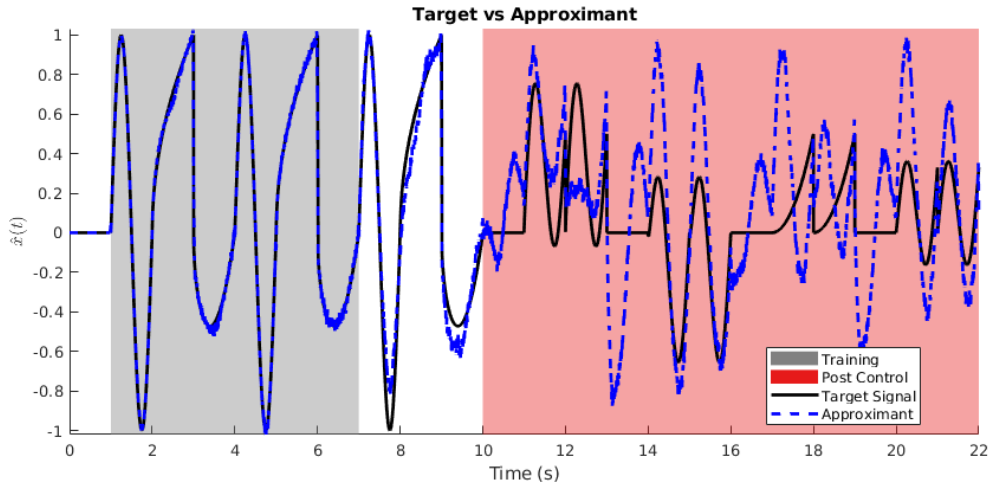


Figure 3.15: The network learns three different target signals guided by three HDTS, but fails to reproduce a linear combination of signals.

3.5 Learning joint trajectories

This section reports the results of the experiments performed on the trajectory datasets, using the same techniques of section 3.4.

N= 1000, a=0.002, b=0, vt=-40, $Q = 4 * 10^2$, d=0

pulses: abs(sin), rectangular, triangular (up then down), triangular (up)

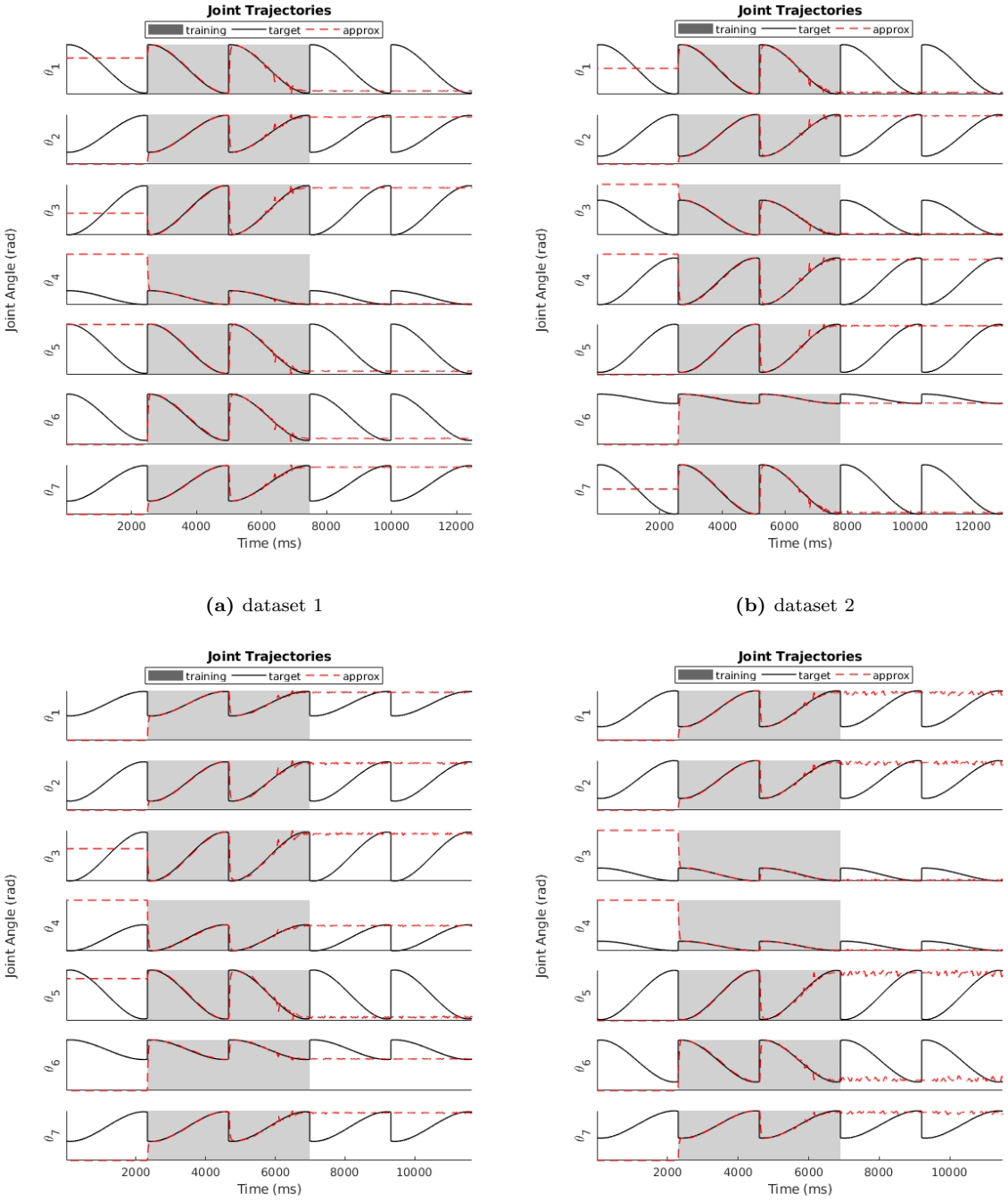
in all figures, the output was smoothed offline (post simulation) with a low-pass filter, to remove the noise and obtain a clearer plot, using filter_output with tau_smooth=1/dt (own function).

3.5.1 HDTS helps the training

In Figure 3.16 it is shown that, without using the HDTS input, the network was still able to learn the trajectories of the 7 joints. However after the training phase was over, the network coud not replay them.

Instead, training using HDTS befenitted the learning process, as the network was able to replay the trajectories post learning, see Figure 3.17.

imin = T, icrit = imin + 2T, iend = icrit + 2T



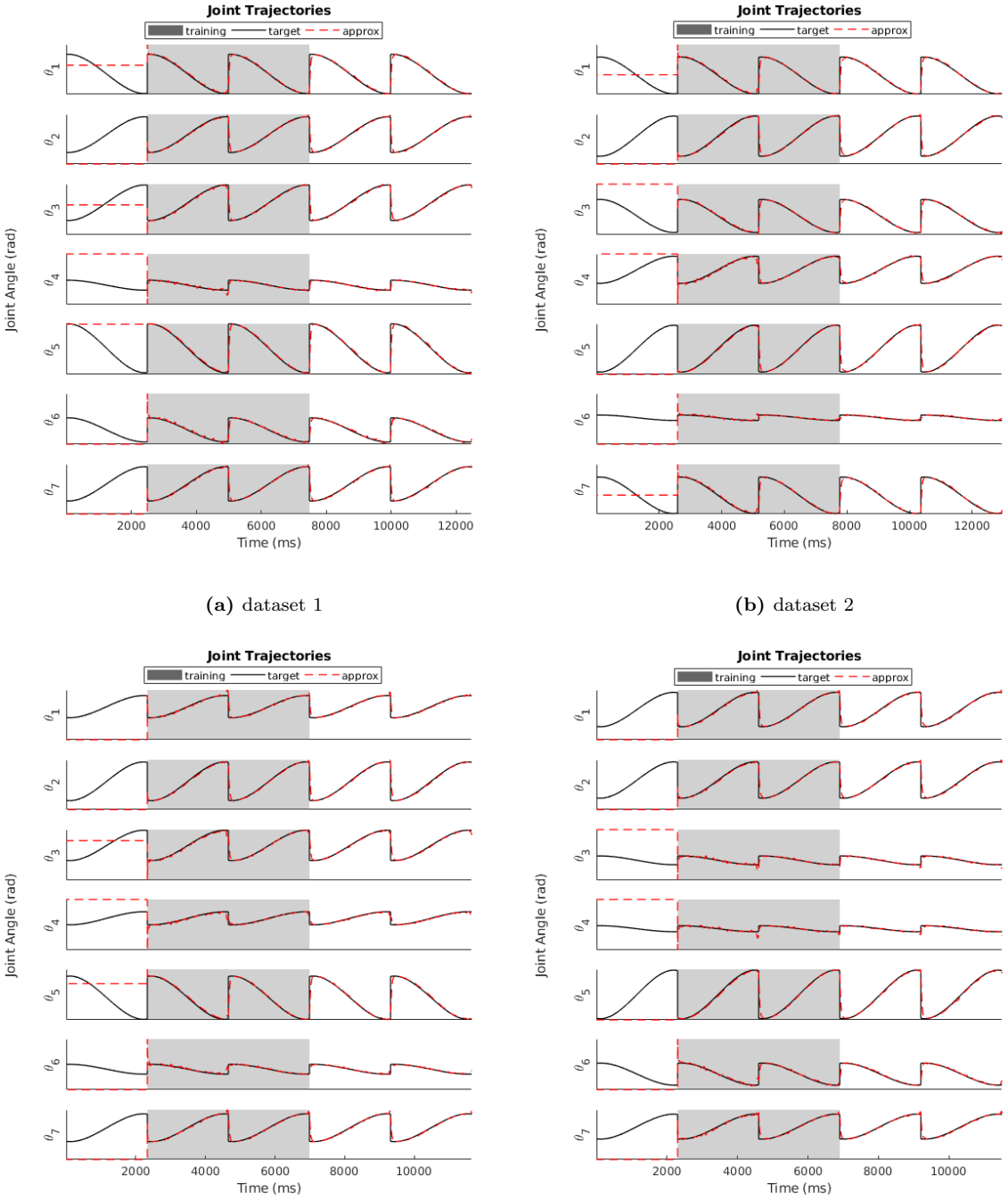
(a) dataset 1

(b) dataset 2

(c) dataset 3

(d) dataset 4

Figure 3.16: Learning the 4 trajectories fails without the aid of HDTS.



(a) dataset 1

(b) dataset 2

(c) dataset 3

(d) dataset 4

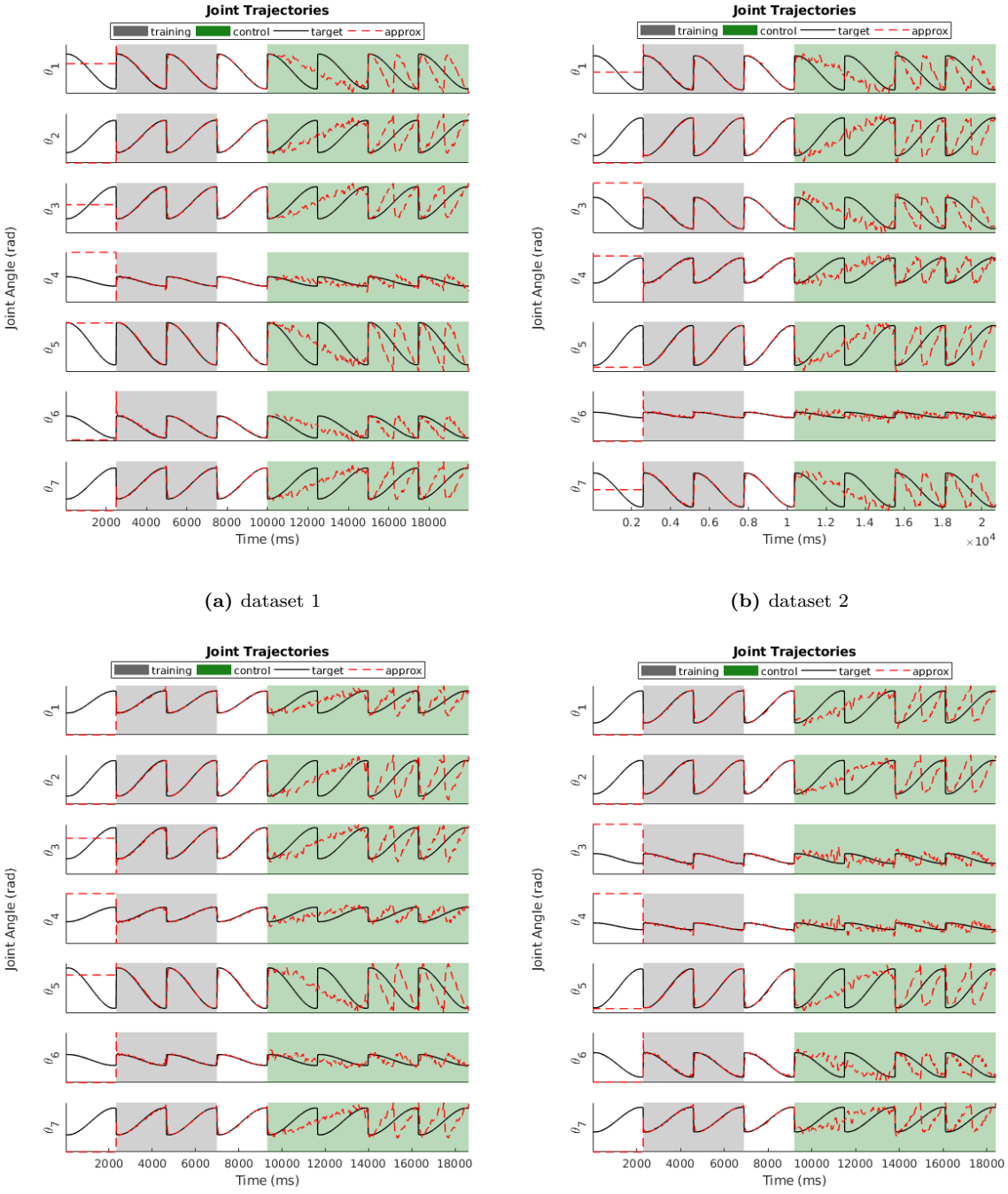
Figure 3.17: Learning the 4 trajectories with the aid of HDTS.

3.5.2 HDTS accelerate and decelerate replay

The acceleration and deceleration of the trajectory replay is shown in Figure 3.18. The acceleration phase consisted in replaying the trajectory with halved velocity, $\rho = 0.5$; the deceleration phase replayed the trajectory at double velocity, $\rho = 2$.

imin=T, icrit= imin + 2T, icrit2=icrit+t, iend=icrit2+4T (2T con rho=0.5 e 2T con rho=2)

Accelerate and decelerate replay



(a) dataset 1

(b) dataset 2

(c) dataset 3

(d) dataset 4

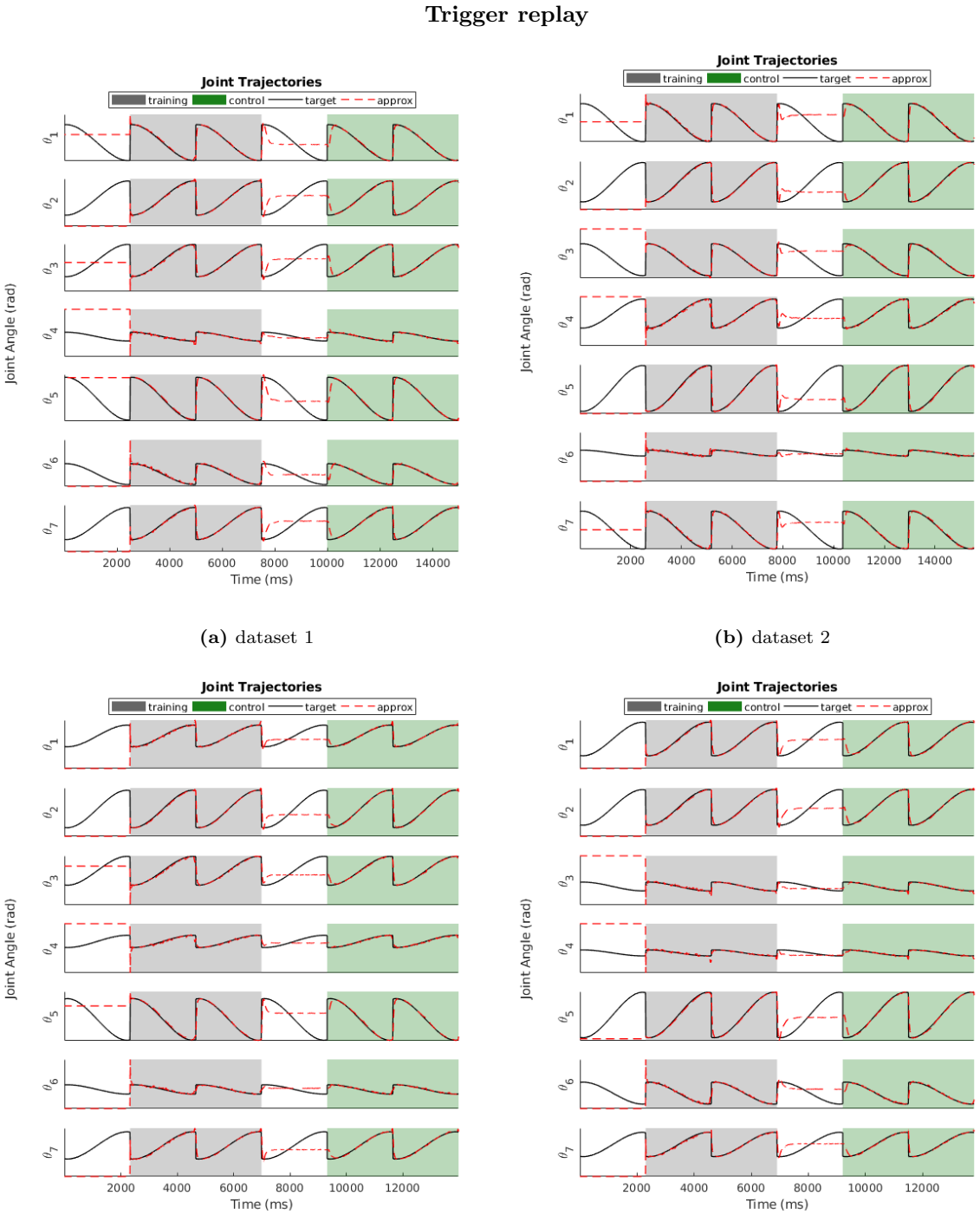
Figure 3.18: Replaying the trajectories at different velocities. Deceleration, $\rho = 0.5$, then acceleration $\rho = 2$. Original trajectory is shown in black.

3.5.3 HDTS triggers the replay

In Figure 3.19 is shown that the network stops the replay of the trajectories as soon the HDTS input is removed, and the reinjection of the HDTS triggers the replay of the trajectories.

imin=T, icrit=imin+2T, icrit2=icrit+T with no HDTS injected, iend=icrit2+2T with HDTS injected

again



(a) dataset 1

(b) dataset 2

(c) dataset 3

(d) dataset 4

Figure 3.19: The replay of the trajectory can be turned on or off by injecting or removing the HDTS input.

3.5.4 Learn four trajectories

In Figure 3.20 is shown the learning and replaying of all the four trajectories.

imin = T1; icrit = imin + 3*(T1 + T2 + T3 + T4); icrit2 = icrit + T1; nt = icrit2 + 2*(T1 + T2 + T3 + T4);

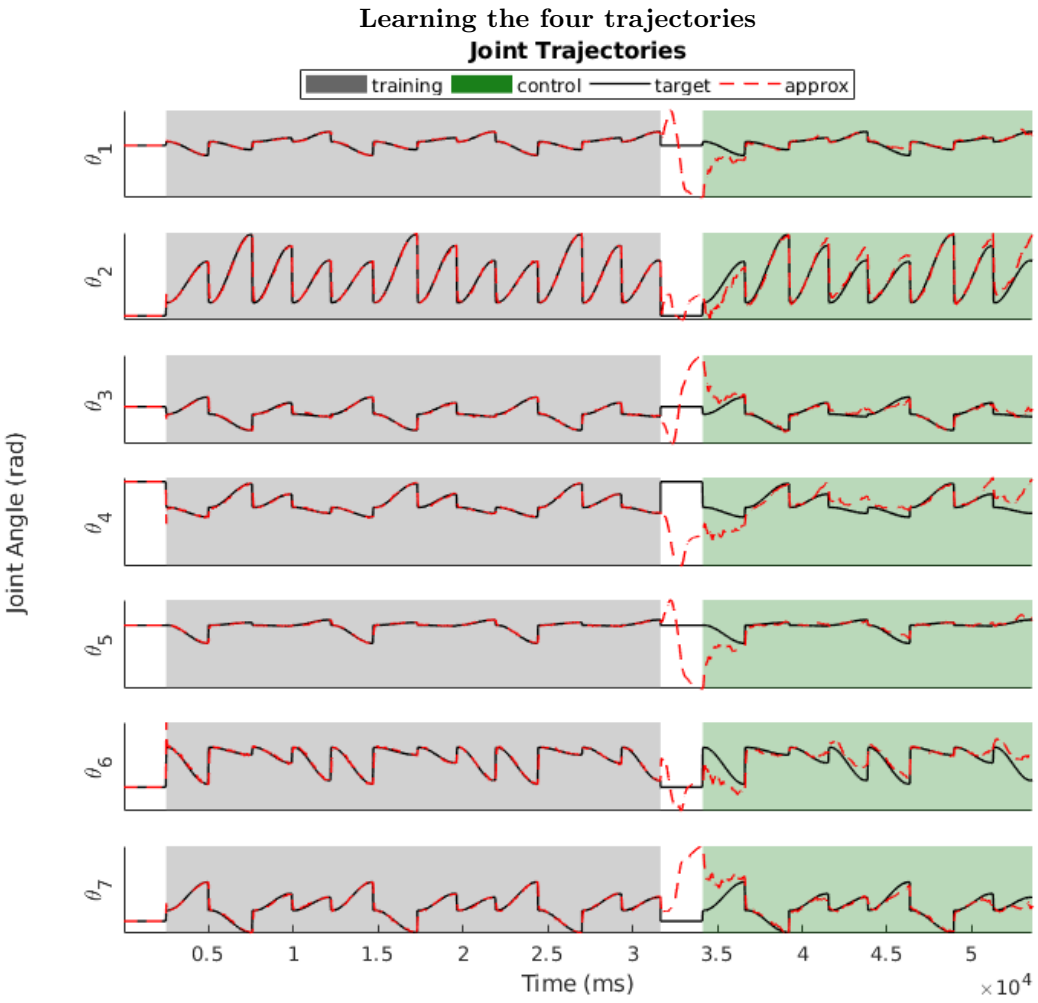


Figure 3.20: Learning the four trajectories.

3.5.5 Trajectory interpolation

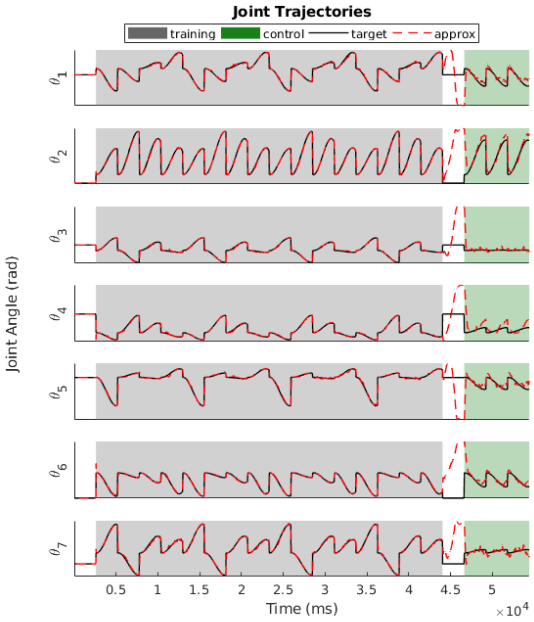
After learning the four trajectories, it was tried to linearly combine the HDTS inputs in order to interpolate between the trajectories. In Figure 3.21 is shown the training and interpolation of the trajectories using the following combinations of HDTS matrixes

$$H_{comb_{12}} = 0.5H_1 + 0.5H_2 \tag{3.5}$$

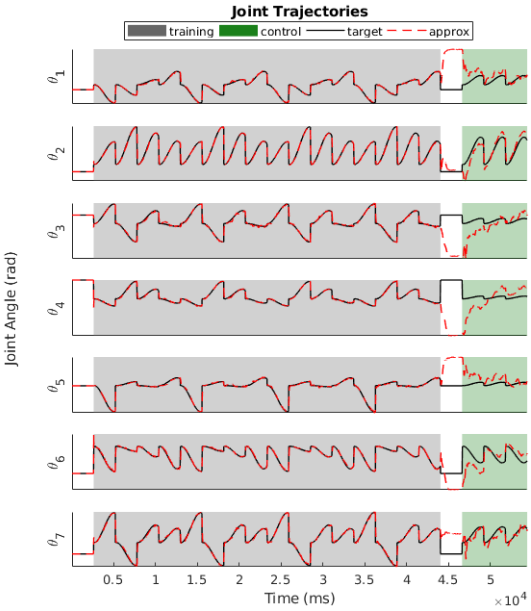
$$H_{comb_{34}} = 0.5H_3 + 0.5H_4 \tag{3.6}$$

$$H_{comb_{123}} = 0.34H_1 + 0.33H_2 + 0.33H_3 \tag{3.7}$$

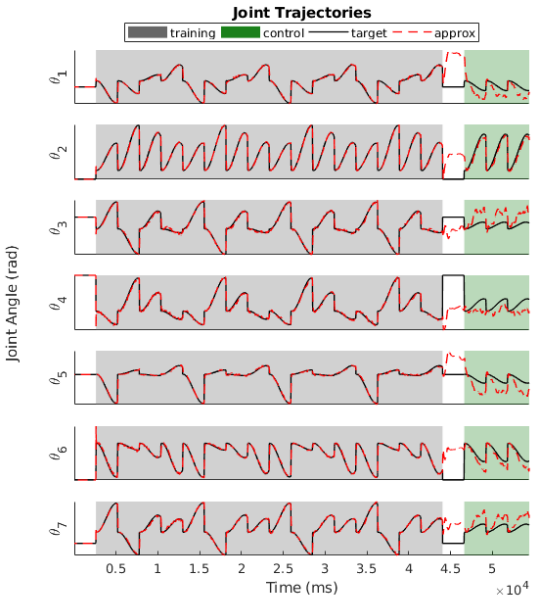
$$H_{comb_{1234}} = 0.25H_1 + 0.25H_2 + 0.25H_3 + 0.25H_4 \tag{3.8}$$



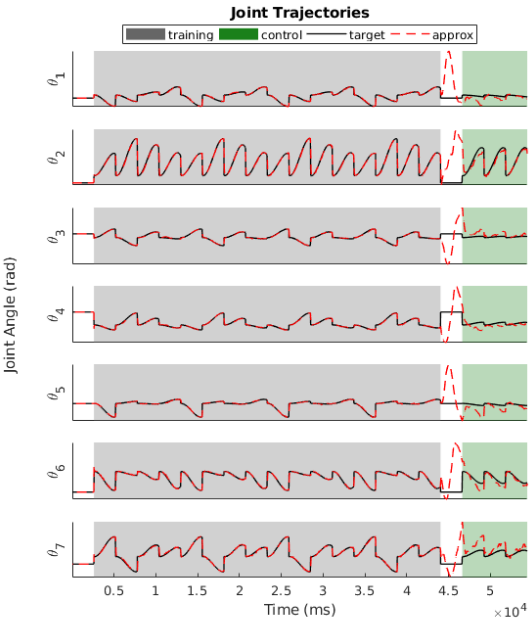
(a) 1-2



(b) 3-4



(c) 1-2-3



(d) 1-2-3-4

Figure 3.21: Trajectory interpolation between (i) two pair of targets 1-2 and 3-4, (ii) a triple of targets 1-2-3, and (iii) a quadruple of targets 1-2-3-4.

T = target length T = max(T1,T2,T3,T4) imin = T, icrit=imin+ 4(4T) present each target 4 times, icrit2= icrit + T, nt=icrit2 + 3T

3.6 Experimental procedures

In this section are reported the values of the parameters used to perform the experiments, and to create the tables and figures.

Parameter	Value	Units
$\boldsymbol{C}$	250	(pF)
$\boldsymbol{v_{rest}}$	-60	(mV)
$\boldsymbol{v_{threshold}}$	$v_{rest} + 40 - b/k$	(mV)
$\boldsymbol{I}$	see eq	(pA)
$\boldsymbol{v_{peak}}$	30	(mV)
$\boldsymbol{k}$	2.5	(nS/mV)
$\boldsymbol{a}$	0.01	(ms^{-1})
$\boldsymbol{b}$	-2	(nS)
$\boldsymbol{c}$	-65	(mV)
$\boldsymbol{d}$	200	(pA)

Table 3.7: Default parameter settings, as specified in [9].

maybe put eq also for v_threshold

put also default table of FORCE method? Q, G, BIAS, tau_d, tau_r

For Figure 3.6 Original Clopath settings, change $d \in \{-50, 0, 50, 100, 150, 200\}$.

For Figure 3.7 $N = 100$ because it is faster and qualitatively you get the same results, $\delta_t = 0.2$ so it is faster, $f = 1$ Hz, start with $d = -50$ then change it every 10 seconds post-learning during the interval $[t_{crit}, t_{end}]$ every 10 seconds change parameter, $t_{min} = 3$ seconds, $t_{crit} = 10$ seconds, $t_{crit_2} = 15$ seconds, $t_{end} = 135$ seconds. (i.e 2 minutes of parameter change) 3.7a $d \leftarrow d + 20$ for 2 minutes, 3.7b $d \leftarrow d - 5$ for 2 minutes, 3.7c $d \leftarrow d + 20$ for the fist minute and $d \leftarrow d - 20$ in the second minute.

For Figure 3.8 As for Figure 3.7. 3.8a $a \leftarrow a + 0.005$ for 2 minutes, 3.8b $a \leftarrow a - 0.00025$ for 2 minutes, 3.8c $a \leftarrow a + 0.005$ for the fist minute and $a \leftarrow a - 0.005$ in the second minute.

3.6.1 HDTS with synthetic datasets

The experiments with the HDTS were carried out using a network of $N = 1000$ neurons, $a = 0.002$, $b = 0$, $d = 0$, $v_{threshold} = -40$, $Q = 4 \cdot 10^2$.

The target signals used were the synthetic datasets defined in section 3.4. All supervisors are long $T = 1000$ milliseconds.

The HDTS matrices were generated using $m = 32$ gaussian pulses. Each HDTS matrix H was multiplied by a static matrix $\eta_h \in \mathbb{R}^{N \times m}$ (same for all the pulses), where each component was drawn

uniformly from $[-4 \cdot 10^3, 4 \cdot 10^3]$. The pulse used with the three target signals were respectively the upper sinusoid, rectangular and isosceles triangle pulse.

For Figure 3.9 The network is set to a chaotic attractor for T milliseconds. Training starts at $t_{min} = T$ and lasts till $t_{crit} = t_{min} + 3 \cdot T$, corresponding to three presentations of the target signal. The learned signal is replayed two times until $t_{crit_2} = t_{crit} + 2 \cdot T$. At time t_{crit_2} it was injected a different version of the HDTS with compression factor $\rho \in \{0.5, 2\}$, resulting in an acceleration or deceleration of the replay.

decide if tmin, tcrit ecc are expressed in seconds, milliseconds or time steps or all interchangeably.

For Figure 3.10 imin = 1T, train for 4 rep, icrit= imin + 4T. Post training replay the signal forward once and twice backward. Supervisor: sinusoid and square root.

For Figure 3.11 Idem as above, except that at time t_{crit_2} the application of the HDTS is removed for 2 seconds. As a result the network stopped the replay of the signal. The network started replaying the signal as soon the HDTS was reinjected till t_{end}. Experiment repeated for two supervisors.

is it chaotic?

For Figure 3.12 imin=1, Trained for 4 rep (s1,s2,s1,s2). So icrit = imin + 4T. After training 2 seconds without HDTS, then 3 rep of sup1, then 2 seconds without HDTS then 3 rep of sup2.

For Figure 3.13 $t_{min} = T$, $t_{crit} = t_{min} + 4T$, $t_{end} = t_{crit} + 4T$. During training two supervisors were presented intermittantly, but using the same HDTS.

For Figure 3.14 trainining as above. After training 2 seconds without HDTS then 3*T with linear combinations of the HDTS. for interpolation using the following (α, β) pairs: $[(0.5, 0.5)\ (0.3, 0.7)\ (0.7, 0.3)]$, for extrapolation using the following (α, β) pairs: $[(1, 0.5)\ (0.5, 1)\ (1.2, 1.2)]$

For Figure 3.15 $t_{min} = T$, $t_{crit} = t_{min} + 6T$ train (s1,s2,s3,s1,s2,s3) with (h1,h2,h3,h1,h2,h3), $t_{crit_2} = t_{crit} + 3T$ replay (s1,s2,s3) with (h1,h2,h3) $t_{end} = t_{crit_2} + 12T$ replay (-, s12, s12, -, s13, s13, -, s23, s23, -, s123, s123) with (-, h12, h12, -, h13, h13, -, h23, h23, -, h123, h123)

3.6.2 HDTS with trajectory datasets

For Figure 3.16

For Figure 3.17

For Figure 3.19

For Figure 3.20

For Figure 3.21

Chapter 4

Discussion

Chapter 5

Conclusions

Bibliography

[1] Luv Aggarwal, Kush Aggarwal, and Ruth Jill Urbanic. Use of artificial neural networks for the development of an inverse kinematic solution and visual identification of singularity zone (s). *Procedia Cirp*, 17:812–817, 2014.

[2] Toshitake Asabuki, Naoki Hiratani, and Tomoki Fukai. Interactive reservoir computing for chunking information streams. *PLoS computational biology*, 14(10):e1006400, 2018.

[3] Alexandros Bouganis and Murray Shanahan. Training a spiking neural network to control a 4-dof robotic arm based on spike timing-dependent plasticity. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2010.

[4] Adrian-Vasile Duka. Neural network based inverse kinematics solution for trajectory tracking of a robotic arm. *Procedia Technology*, 12:20–27, 2014.

[5] Muhammad Dur-e Ahmad, Wilten Nicola, Sue Ann Campbell, and Frances K Skinner. Network bursting using experimentally constrained single compartment ca3 hippocampal neuron models with adaptation. *Journal of computational neuroscience*, 33(1):21–40, 2012.

[6] Vishwa Goudar and Dean V Buonomano. Encoding sensory and motor patterns as time-invariant trajectories in recurrent neural networks. *Elife*, 7:e31134, 2018.

[7] Gregor M Hoerzer, Robert Legenstein, and Wolfgang Maass. Emergence of complex computational structures from chaotic neural networks through reward-modulated hebbian learning. *Cerebral cortex*, 24(3):677–690, 2012.

[8] Eugene M Izhikevich. Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572, 2003.

[9] Wilten Nicola and Claudia Clopath. Supervised learning in spiking neural networks with force training. *Nature communications*, 8(1):2208, 2017.

[10] David Sussillo and Larry F Abbott. Generating coherent patterns of activity from chaotic neural networks. *Neuron*, 63(4):544–557, 2009.

[11] Valeria Villani, Fabio Pini, Francesco Leali, and Cristian Secchi. Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications. *Mechatronics*, 55:248–266, 2018.