Optimisation of Learning-to-Learn in Spiking Neural Circuits

Kristian D'Amato

Supervisors: Dr George Azzopardi &
Prof. Wolfgang Maass

Faculty of ICT

University of Malta

01/08/2018

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Artificial Intelligence

Faculty of ICT

Declaration

I, the undersigned, declare that the dissertation entitled:

Optimisation of Learning-to-Learn in Spiking Neural Circuits

submitted is my work, except where acknowledged and referenced.

Kristian D'Amato

09/08/2018

For Nicole, my wife-to-be, without whom this would not have happened.

Acknowledgements

I offer my sincere gratitude to Dr George Azzopardi for his support, as well as Prof. Wolfgang Maass and his research team in Graz, Austria for introducing me to their wonderful world of computational neuroscience.

Abstract

Situated at the intersection of artificial intelligence and theoretical neuroscience, spiking neural networks (SNNs) have proven valuable for modelling and predicting neural phenomena. SNNs provide rich dynamics that are not replicable by conventional neural networks, exhibiting desirable properties such as self-healing, the ability to exploit noise as a resource, and the ability to solve difficult constraint problems like Sudoku or NP-hard problems such as the Travelling Salesman Problem without training. This work aims to discover initial configurations that lead to more effective and generalisable learning as part of a broader effort to deduce the computational principles that achieve generalisable learning in the human brain. Building on Pecevski & Maass, the work explores density estimation in simulations of SNN winner-take-all (WTA) circuits and similar constructions by using genetic algorithms (GA) and natural evolution strategies (NES) to search for individual optimal configurations that minimise Kullback-Leibler (KL) divergence between multiple estimate and target distributions. It demonstrates that in some tasks optimal network configurations outperform published results even though theoretical zero-delay conditions do not hold, that generalisability can be achieved with a significant but non-fatal impact in both conditional and joint estimation tasks, that biological plausibility can be pushed by introducing a novel architecture with realistic modifications that also achieves competitive performance in conditional estimation tasks, and that there exists a negative relationship between synaptic delay and estimation performance.

Contents

List of Figures

List of Tables

1. Introduction

There are fundamental issues with current approaches to artificial intelligence that are unlikely to be solved by deep learning or other contemporary techniques in machine learning [27]. One focal problem is the lack of learning generalisability—for instance, a neural network trained to play Go can only be used for that purpose, because its internal weights and biases are tuned for that specific task. The study of generalisability or inductive principles is often subsumed under the terms *learning-to-learn* (LTL), transfer learning or multitask learning.[1]

Another desideratum lacking in contemporary machine learning is the model-building ability of human intelligence. Published work mainly deals with pattern recognition—models that take a pattern as input, and output a class or value, also called *discriminative models*. What we are interested in instead are *generative models*—models that build an internal representation of a domain, and that can act in a discriminative capacity as well. Generative models are more relevant to intelligence because an intelligent agent does not simply respond to external stimuli, but also possesses an internal representation of the external world upon which are founded cognitive and psychological processes like intentions, desires and beliefs.

Spiking neural networks (SNNs) are low-level abstractions of brain circuitry, employing temporal dynamics that mimic biological reality. Suitably constructed SNNs have been shown to act in both discriminative and generative capacities,

[1] The terms place slightly different emphases on different aspects of the same problem.

University of Malta
L-Università ta' Malta

depending on whether input is presented to them or not [21, 43], thus fulfilling the second desideratum. When compared with conventional neural networks, numerous properties make SNNs an attractive research topic. Among other things, it has been shown that rather than being hindered by the presence of stochastic noise, SNNs exploit it as a resource [34]; that recurrent SNNs can continuously rewire themselves and compensate for network damage in realistic reinforcement scenarios [22]; that properly constructed SNNs can solve difficult constraint problems like Sudoku or NP-hard problems such as the Travelling Salesman Problem without training [14, 17]; that some SNN structures account well for the Bayesian brain hypothesis [39]; that these SNN architectures implicitly implement Expectation Maximisation for building their generative models [39]; and that neuromorphic hardware based on spiking events is orders of magnitude more energy efficient than conventional hardware of equivalent performance [9].

However, learning remains a challenge in SNNs. Earlier efforts to develop simple supervised learning rules for SNNs analogous to backpropagation in artificial neural networks (ANNs) had limited results, owing to the complexity of SNN dynamics (reviewed in [23]). Later efforts have focused on the use of reward-based learning and Bayesian inference, with significant results being reported [21, 22]. Although successfully convergent, reinforcement in SNNs is highly resource-intensive and does not yet exhibit the generalisability of the first desideratum above. Other work has depended on the use of problem-specific, theoretically informed or hand-crafted models that are not meant to generalise [43].

The present work, as part of a broader effort by a research group at Graz University of Technology at addressing this issue, looks at LTL in generalisations of *winner-take-all* (WTA) spiking circuits, aiming to find optimal network and training algorithm configurations that give best performance when learning probabilistic dependencies between binary variables in conditional and joint density estimation tasks. The work also pushes biological plausibility by introducing a novel architecture that does not rely on WTA-like behaviour, with the objective of performing

the same tasks. Since zero delay is a key assumption in [43], this work explores the impact of delay on estimation performance and, finally, aims to understand the sensitivity of optimal configurations to perturbations in their hyperparameters.

Given the mathematical intractability of complex ensembles of neurons, this work takes the experimental approach by simulating neural networks in biological time using a fixed time step, with estimated distributions determined experimentally from spiking activity. To evaluate estimation performance, a key measure is Kullback-Leibler divergence, used for its effectiveness and consistency with existing research. In achieving these aims, the work solves the hand-crafting problem and demonstrates that these architectures are in principle a feasible construct for these types of estimation task in the brain. It also offers a biologically-inspired alternative to parameter estimation in conventional probabilistic graphical models. Aside from the theoretical step forward that this work represents, it also has practical applications: neuromorphic hardware can far more readily leverage architectures that generalise to multiple problems, as opposed to hand-crafted models with their severe restrictions.

This work draws heavily on research in the neurosciences. Aside from the motivating factors and challenges of the field, the work is founded on the observation that the level of abstraction above biological complexity appropriate for the understanding and replication of human intelligence has not yet been determined. It is therefore sensible to use the brain as a point of departure and to postpone the decision on the proper level of abstraction until it can be made on the basis of evidence.

Following this introduction, Chapter 2 describes the broader context and progress up till the present. Chapters 3 and 4 lay down the two experiments that are to be carried out, underlying both of which are conditional and joint distribution estimation tasks. Results are presented in Chapter 5, interpreted and discussed in Chapter 6, and concluding remarks are given in Chapter 7.

2. Background

The human brain has been a model for artificial intelligence since the birth of computer science, with early research focusing on top-down symbolic manipulation under the assumption that the mind was a symbol-pusher, an algorithmic manipulator of bits of information. When the fruits of this labour failed to materialise, it gave way to connectionism, a bottom-up approach that sought to understand the physical substrate of the brain—Marr's *hardware implementation* level of analysis [35]—informed by the reductionist expectation that human intelligence arose from the complex dynamics of neural networks as an emergent phenomenon [46].

Over the years, the connectionist doctrine has become nuanced by other considerations, but it remains a key assumption of most research into the computational principles of the mind. At the same time, several tangible advances in machine learning under the connectionist paradigm have led to an upsurge in popular and academic interest in neural networks, strengthening belief in the possibility of creating intelligent machines.

Machine learning and computational neuroscience are parallel but distinct fields, maintaining a tenuous link through the connectionist doctrine. Whereas the former is mainly concerned with *reproducing* artificial intelligence, irrespective of its final form, the latter searches for the instrumental principles of intelligence as *embodied* by the biological implementation. Thus, contemporary connectionist models in machine learning tend to be highly reduced versions of their neuroscientific siblings,

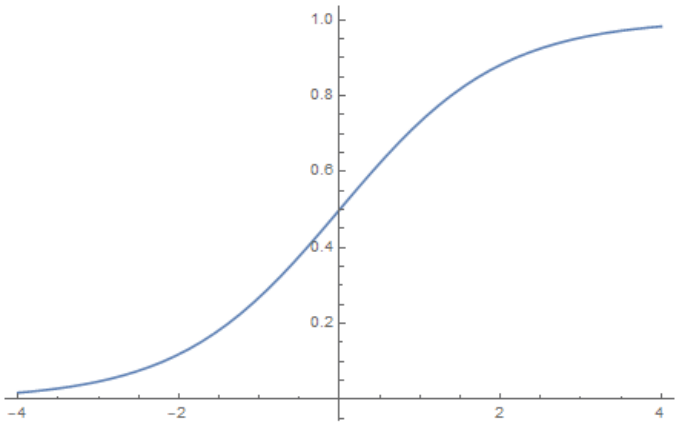

Figure 2.1: Logistic sigmoid function, with output in the range (0, 1). This is a common choice of activation function in second generation neural networks.

used because they give fruitful results, not for their explanatory power.

2.1 Spiking Neural Networks

To highlight the differences, Maass distinguishes between three generations of connectionist models, depending on the type of computational unit[1] that the network uses [32]. The *first generation*, proposed in 1943, was based on the McCulloch-Pitts neuron[2] and was capable of binary output only. *Second generation* neurons introduced continuous static output. In networks that employ them, each neuron calculates a weighted sum of its input and transforms the resulting value by passing it through a real-valued activation function. A typical choice for activation function is shown in Figure 2.1. Note that output is deterministic and time-independent. Second generation networks are still the most popular choice of model in machine learning, owing to their resurgence in deep learning and convolutional networks.

Neurobiological evidence, however, gradually challenged the plausibility of second generation networks as good models of brain circuitry, with several experiments showing that temporal effects were a necessary ingredient of neural communication

[1] Variously called *nodes* or *neurons*, depending on whether the context emphasises AI or neuroscience, respectively.

[2] Also known as *threshold gates*.

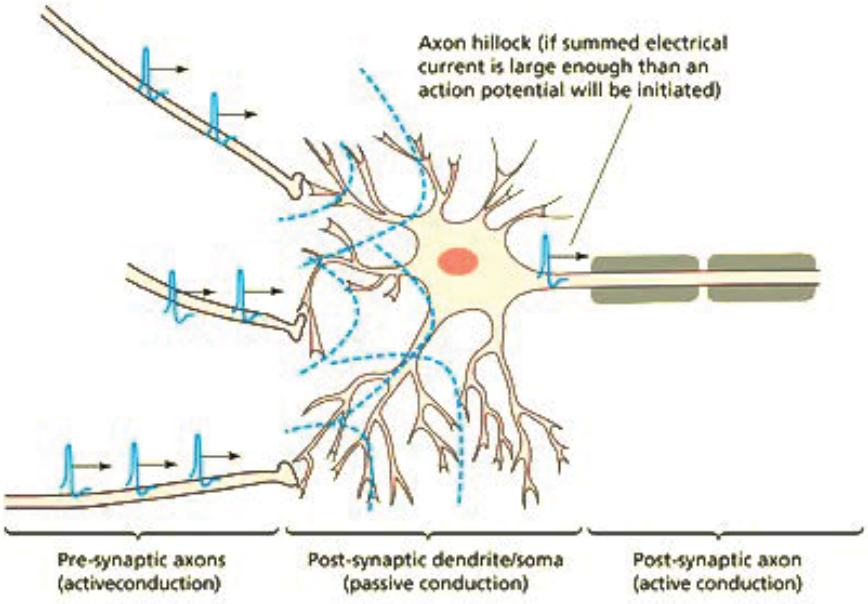

Figure 2.2: A postsynaptic neuron summing up action potentials from multiple presynaptic neurons. Adapted from [53].

[32, 6]. The gap was addressed in the late 90s by the *third generation* of neural networks, which used the spiking neuron as computational unit.

Spiking neurons introduce intrinsic *time-dependence* in their mechanics [32, 12]. Similarly to second generation neurons, spiking neurons act as integrators of input from multiple sources. However, information transfer cannot be modelled by a simple function of the weighted input, and is instead described by a set of partial differential equations that take into account time, internal state, as well as transient input.[3] To maintain the link with neuroscience, this internal state is usually thought of as "membrane potential", although a dimensionless abstract quantity can replace it. Spiking neuron differential equations describe mechanics that generally cause the neuron to emit short, sharp "spikes" or *action potentials*, when it is sufficiently activated. These electrical perturbations emerge from the input or *presynaptic* neuron and reach the *postsynaptic* neuron via connections called *synapses* (see Figure 2.2). At the postsynaptic neuron incident spikes induce *post-*

[3]Some models act as convolving operators rather than differential equations, but many differential equation models are equivalent to specific types of convolution. Convolution models are also called *spike response models*.

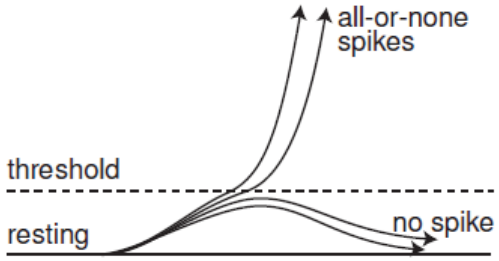

Figure 2.3: All-or-none response of a deterministic neuronal model. From [15].

synaptic potentials (PSPs), transient responses in the neuron membrane potential that can be either excitatory or inhibitory.[4]

In the simplest of third generation neuron models, such as the *leaky integrate-and-fire* (LIF) neuron, if the sum of transient responses exceeds a fixed threshold, a spike is generated which is then relayed to the neuron's subsequent connections in the manner just described. More realistic models like the *AdEx* model[5] introduce adaptability or stochasticity in the threshold mechanism,[6] together with complex internal dynamics that accurately keep track of the membrane potential. Nevertheless, irrespective of the complexity of the model dynamics chosen, their final validation is solely in terms of spike timing—how well does the model reconstruct biological spiking in a neuron? Current neuroscience theory is indeed founded on spikes: to the best of our knowledge, biological neurons communicate via spikes [44, 11]. Subthreshold transients, that is, perturbations that do not lead to action potentials, have little or no influence on connected neurons. Moreover, there is no intermediate output between spiking and not-spiking; a neuron either spikes or it does not. This is termed "all-or-none" behaviour (see Figure 2.3). It is the precise timing of spikes that carries information, not their temporal extent or the potential difference of the spike. For these reasons, spikes can be idealised

[4]That is, tending to increase or decrease membrane potential, respectively.

[5]Or *adaptive exponential integrate-and-fire* model.

[6]Stochasticity is present in biological neurons because ion-channel mechanisms—the mechanisms that determine a neuron's response to incident spikes—open and close in a discrete and stochastic manner.

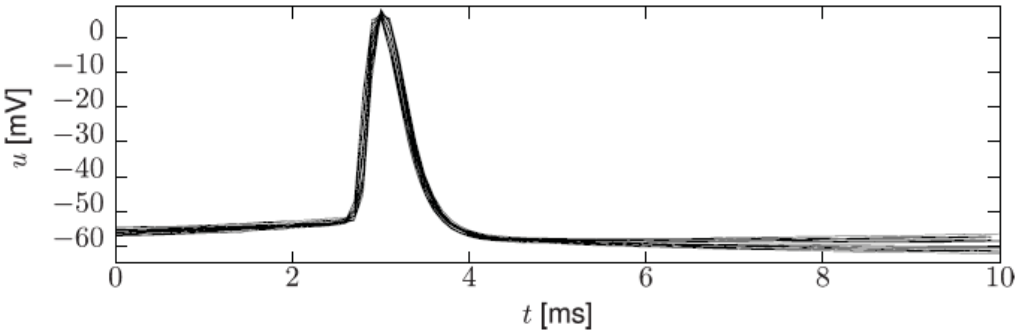

Figure 2.4: Spikes are stereotypical events that follow similar trajectories with little variation, and from an information-theoretic point of view can therefore be thought of as idealised point-events. From [11].

as stereotypical point events, discontinuities in the solutions described by the state equations [44, 11].[7] Thus, a machine that can accurately reproduce the rich dynamics of natural brain spiking together with its specific connectivity patterns would in essence be equivalent to an artificial mind.[8] The stereotypical trajectory of action potentials in biological neurons is shown in Figure 2.4.

Although still a simplification of biological reality, third generation neuron models respect the importance of the spike and provide far richer dynamics than previous generations. Importantly, they realise the possibility of using time as a resource in computation and communication [32], and therefore, unlike second generation models, they are time-dependent and can also exhibit indeterminacy via stochasticity. Partly because of their intrinsic temporal dynamics, and partly through explicit network recurrence, SNNs can integrate information on multiple time scales, unlike their conventional second generation counterparts. For instance, suitably constructed *liquid state machines* (LSMs)—a particular SNN architecture that exploits strong recurrence—were found to compete with and outperform hidden Markov models (HMMs) in classifying audio patterns under the effect of noise [52].

[7] Complex neuron models like the Hodgkin-Huxley model integrate the spiking trajectory into the differential equations themselves, but the equations are not easily tractable for large collections of interacting neurons.

[8] This ignores some of the thorny philosophical issues involved in defining what a mind is.

Learning and adaptability are key to any AI technique, irrespective of the technique's biological plausibility. Unfortunately, while SNNs had been theoretically shown to be computationally more powerful than earlier connectionist models [32],[9] they are not straightforward to train under supervision [23], with most earlier techniques handicapped by severe limitations. Many publications instead relied on hand-crafted models that solved specific problems or explored theoretical limitations. Over the last decade, however, a slew of techniques ranging from the biologically plausible to the improbable have been developed and successfully applied. Crucially, SNNs have demonstrated state-of-the-art performance in certain tasks (e.g. [9, 28]).

Plasticity—the modification of synaptic weights through the co-activation of connected neurons—plays a key role in both biological and simulated networks. Hebb's famous postulate of 1949, often summarised as "neurons that fire together, wire together" [11], offers the basis of many biologically-inspired learning techniques in SNNs. *Spike-timing dependent plasticity* (STDP) is one such Hebbian technique with strong experimental support. In STDP, a synapse is strengthened if presynaptic spiking precedes spiking in the postsynaptic neuron, and weakened if the timing is reversed. Figure 2.5 demonstrates this relationship.

Note, however, that many of these plasticity rules, STDP among them, are local rules that determine changes, if any, blindly from the activity of two connected neurons. They are not counterparts to supervised learning techniques like backpropagation in ANNs, where the target signal is co-present with the input, and where weight changes follow an optimal direction with respect to reducing error. In fact, since unsupervised learning may be characterised by modification of model parameters to input statistics, STDP and several other plasticity rules can be considered unsupervised techniques [11].[10] It is therefore a non-trivial result that, as

[9] In the sense that some input-output mappings can be achieved with fewer spiking neurons than the corresponding number of conventional neurons.

[10] Plausible techniques that involve supervision in the brain are mediated by reward signals, implying that unadulterated supervised learning is replaced by reinforcement.

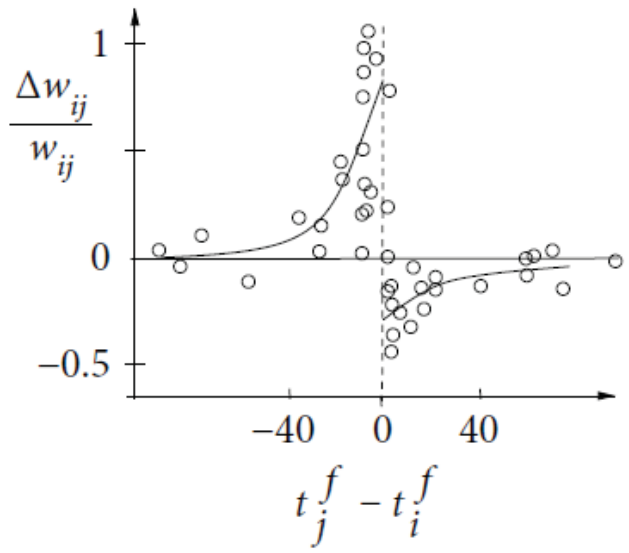

Figure 2.5: STDP window, demonstrating relative weight changes $\frac{\Delta w_{ij}}{w_{ij}}$ between two neurons as a function of the difference between postsynaptic firing time t_i^f and presynaptic firing time t_j^f. Experimental data points are shown in circles, and best-fit exponential curves in solid black. Note that synaptic weight changes tend to zero as the difference increases. From [11].

we shall see below, STDP can install in a neural network the ability to optimise its connectivity for certain tasks.

Recent research on spiking networks has placed much emphasis on probabilistic generative models [7, 30, 18]. Automated learning of probabilistic graphical models (PGMs) is currently an attractive parallel topic of research in conventional machine learning, presenting an opportunity for knowledge discovery in complex datasets without requiring models hand-crafted by experts.[11] This may be transformative in such fields as bioinformatics and medicine, where the ability to consume massive datasets, learn structure and make automated inferences may obviate the need for some types of manual hypothesis building and research. Learning in PGMs is detailed in [25].

[11] *Online* learning of PGM structures and parameters are, to the author's knowledge, very sparsely researched. The present work deals with an online method of training.

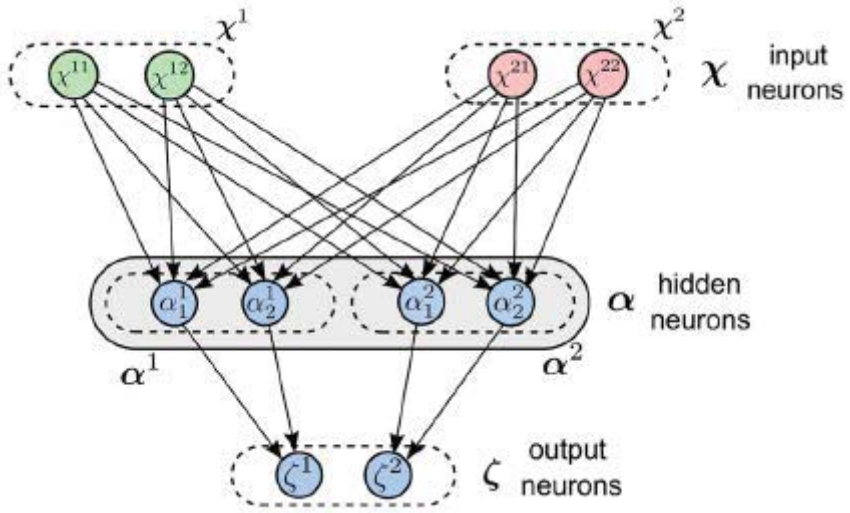

Figure 2.6: Stochastic association module (SAM) for a target probability with two input binary random variables and an output binary random variable. Note that lateral inhibition connections are not shown. From [43].

2.2 Stochastic Association Modules

In SNNs, meanwhile, it has been shown that spiking can be interpreted as sampling from a probability distribution. Furthermore, synaptic plasticity can bestow to a network the ability to perform Bayesian computation [39], the gold standard of inference under uncertainty. Some of this work has focused on the use of a particular stereotypical arrangement of neural assemblies called winner-take-all circuits (e.g. [41]), which were shown to have superior computational properties when compared to threshold gates or sigmoidal units [33]. In WTA circuits, neurons compete for output, with activated neurons inhibiting their counterparts to "take all" of the output activity. Thus, the neuron with strongest activation wins the competition.

Pecevski & Maass [43], the basis of the present work, builds on these previous efforts by suggesting a model that uses spiking neurons to learn probabilistic dependencies between random variables using experimentally observed plasticity rules within an unsupervised learning environment. Although the network is topologically similar to a two-layer feedforward perceptron (see Figure 2.6), the units are stochastic integrate-and-fire neurons (see Section 2.3), which were thoroughly

investigated in [16] and found to fit experimental data well. Lateral inhibition is used between hidden neurons, with spiking hidden neurons activating inhibitory neurons (not shown in Figure 2.6) that relay a depressing signal to competing hidden neurons. This type of recurrent architecture is called a *stochastic association module* (SAM), a generalisation of the WTA circuit. SAM modules can also be combined recursively to learn generic dependencies that are typically modelled using Bayesian networks, such as the visual perception problem in [43] (see Section 4.1).

In the SAM architecture, input neurons χ are divided into subpopulations of neurons χ^i, one subpopulation for one input variable x^i, each of which has a single neuron for every possible value of that random variable (Section 2.5 describes variable encoding). The hidden neurons α are also divided into subpopulations α^i, one for each discrete value of the output variable z. Multiple neurons in each α^i allows the module to better represent multi-modal probability distributions [43]. Finally, there is one neuron ζ^i for each value of z. χ and α neurons are all-to-all connected, while α-ζ connections are made on the basis of subpopulation index (see Figure 2.6). Importantly, all weights are fixed except those that connect χ with α neurons. Bias is similarly fixed, except in hidden neurons (see Sections 2.3 and 2.4 for details on bias). In short, plasticity is only active in hidden neurons and their afferent connections. The output neurons act as integrators of activity in connected α neurons, for which purpose α-ζ connections have large weights, strong enough so that activity in any of the α neurons is certain to trigger sympathetic spiking in the connected ζ neuron.

2.3 Stochastic Integrate-and-Fire Neurons

Pecevski & Maass [43] uses a variant of the *stochastic integrate-and-fire* neuron model, which is retained in the present work. Stochastic integrate-and-fire neurons are a family of models that do not have a fixed threshold. Instead, the mechanism

is replaced by a firing probability density that at any point in time is given by a positive function. A common choice is the exponential function, so that the probability density, also known as a firing intensity, is given at t by

$$\rho(t) = \frac{1}{\tau} \exp(V_m(t)), \tag{2.1}$$

where τ is a time constant and $V_m(t)$ is the membrane potential [43]. Upon firing, the neuron enters a short *refractory* period of duration τ, during which no further spiking can occur. The membrane voltage is given by a weighted sum of the PSPs induced in the neuron by its spike train input,

$$V_m(t) = \sum_i w_i \epsilon_i(t) + b, \tag{2.2}$$

where w_i is the synaptic weight connecting the presynaptic neuron that caused PSP ϵ_i to the postsynaptic neuron and b is the postsynaptic neuron's bias. PSP models abound, but the one used by Pecevski & Maass for computer simulations is α-shaped,[12] a model informed by experimental findings and described by the equation

$$\epsilon(t) = \epsilon_0 \cdot e \cdot (\frac{t}{\tau_\alpha} + t_1) \cdot \exp(-(\frac{t}{\tau_\alpha} + t_1)) - \frac{\epsilon_0}{2} \text{ if } 0 < t < (t_2 - t_1)\tau_\alpha, \tag{2.3}$$

and 0 otherwise. Here, ϵ_0 is a constant, τ_α is the α-kernel time constant, and t_1 and t_2 are the numerical solutions to the equation $e \cdot t \cdot \exp(-t) - 0.5 = 0$. The α-shaped PSP from [43] is shown in Figure 2.7. Pecevski & Maass also uses simple step-shaped PSPs, and all theoretical derivations are based on this more tractable form.

Since computer simulations discretise time, firing probabilities need to be worked out from the firing intensity. In particular, we must avoid multiplying the probability density by the simulation time step Δt since it only gives an approximate

[12] That is, characterised by a sharp increase and a gradual, exponential decay.

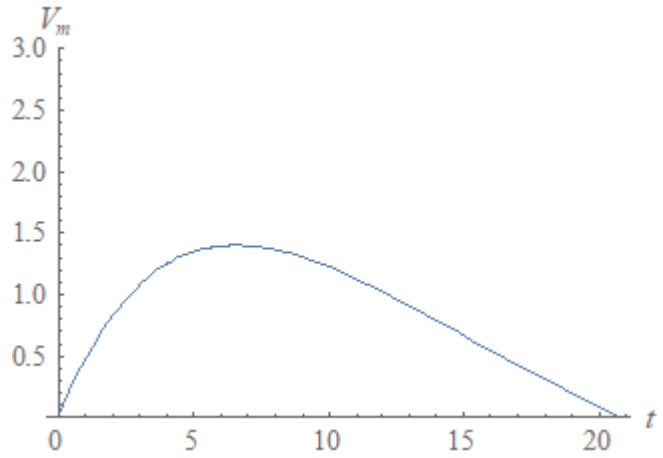

Figure 2.7: The effect of a single α-shaped PSP from Pecevski & Maass on membrane voltage V_m, assuming $\epsilon_0 = 2.8$ and $\tau_\alpha = 8.5$ ms as in the paper. PSPs are additive. Note the rapid rise and slow decline of the voltage perturbation.

probability, and is also bound to overestimate for high firing intensities [11]. Instead, a better approximation is found by integrating the *survival* probability[13] over Δt and subtracting it from 1. This gives

$$P_F(V_m) = \text{Prob}\big(\text{spike in } [t, t+\Delta t] | V_m(t)\big) \approx 1 - \exp(-\Delta t \rho(t)), \qquad (2.4)$$

which retains a bounded probability between zero and one [11].

2.4 Spike-Timing Dependent Plasticity

Although there are many commonly used formulations of STDP, Pecevski & Maass employs a form of STDP modified for tractability. If w is the synaptic weight of a connection from some presynaptic neuron ν_i to a postsynaptic neuron ν_j, then upon every postsynaptic spike at time t the weight is updated, $w \leftarrow w + \eta \Delta w$, where Δw is positive if a presynaptic spike occurred within a time window τ, and

[13] That is, the probability that the neuron does *not* fire.

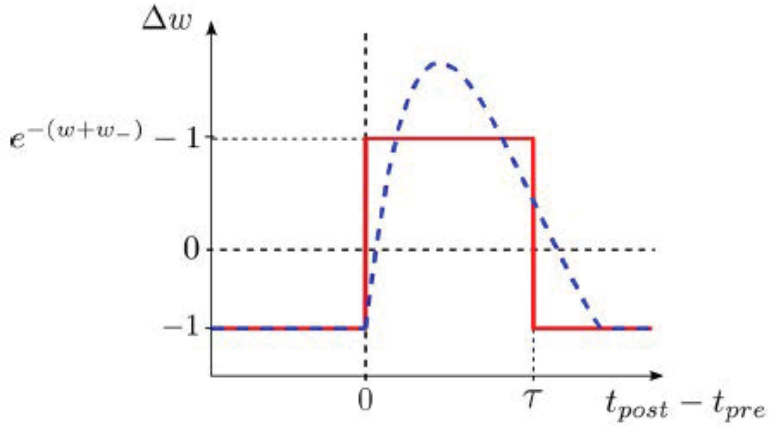

Figure 2.8: Weight update Δw as a function of the time delta $t_{post} - t_{pre}$. In red is the profile described by Equation 2.5. The dashed blue curve shows a smoother α-shaped STDP profile, used in some simulations in Pecevski & Maass. From [43].

negative otherwise. Specifically, this can be stated as

$$\Delta w = \begin{cases} e^{-(w+w_-)} - 1, & \text{if } \nu_i \text{ fired in } [t-\tau, t] \\ -1 & \text{if } \nu_i \text{ did not fire in } [t-\tau, t]. \end{cases} \tag{2.5}$$

w_- is a constant baseline parameter and η is a learning rate [43]. Figure 2.8 shows the step profile described by this equation.

Neuron bias—also referred to as *intrinsic plasticity*, an adaptive mechanism within the neuron that adjusts excitability depending on recent activity—is similarly updated upon spiking by the assignment $b \leftarrow b + \eta' \Delta b$, with the bias update given by

$$\Delta b = \tau e^{-(b+b_-)}, \tag{2.6}$$

where b_- is a constant baseline parameter [43]. Note that the bias learning rate η' is not equal to the weight learning rate η. Aside from the discrete updates, bias decays continuously according to

$$\frac{db}{dt} = -\eta'. \tag{2.7}$$

2.5 Neural Coding and Sampling

The topic of neural coding is subject to lively debate and controversy: how are variables encoded by neurons or ensembles of neurons? Pure *rate coding*—where the average spike rate is the sole carrier of information—cannot apply throughout the brain, for instance, for the simple reason that humans and animals respond to certain stimuli on time scales that are too short for computation of rates [6]. Biological reality is complex, and it is likely that the brain uses a heterogeneous mix of neural codings. Be that as it may, published research tends to explore a variety of neural codes.

Population coding is one candidate for the neural code that is commonly found in literature dealing with probabilistic SNN models. Under this scheme, variables are encoded by the joint activity of a pool of neurons, with one or more neurons sensitive to a particular component of the input. In Pecevski & Maass [43] specifically, each neuron in the input and output layers represents a specific discrete value of one random variable. During training, samples $\langle \mathbf{x}, z \rangle$ drawn from a target joint distribution $p(\mathbf{x}, z)$ are presented to the network (with $\mathbf{x}$, a vector quantity, encoded by the input layer, and z by the output layer), which causes the neurons corresponding to the assigned values to become disinhibited,[14] allowing plasticity to proceed and strengthening certain connections. Slowly, the network's dynamics and plasticity rules (a simplified form of STDP) modify its connectivity so that the target joint distribution is better approximated by the probability distribution embodied by the network itself, minimizing the Kullback-Leibler (KL) divergence between the internal model and the target distribution. When incomplete input is presented—i.e. samples $\langle \mathbf{x} \rangle$ missing z values—the module approximates the conditional probability $p(z|\mathbf{x})$ by the spiking activity in its output layer. In this sense, SAM modules embody a generative model of the target distribution, but can also be used for predictive inference.[15] Moreover, modules can be combined recursively to

[14] The other neurons in the hidden layer are artificially inhibited by a negative current.

[15] Deep belief networks are non-spiking generative models, but they require separate unsuper-

represent generic dependencies with the flexibility of probabilistic graphical models [43], in which case the network becomes a full generative model of the target joint distribution via its spontaneous spiking activity.

Pecevski & Maass obtains certain theoretical results on the spiking behaviour of SAM modules based on a mixed collection of implicit and explicit assumptions about synaptic delay and the neuron model. In particular, the square form assumption of the STDP profile has already been stated (see Section 2.4 and Figure 2.8). In deriving an analytic formula for the activity of output neurons, additionally, an implicit assumption is that PSPs are also square-shaped, unlike the α-shaped PSP described in Equation 2.3, which is only used for simulation purposes. Another pillar is the assumption of zero synaptic delay. This is not stated explicitly, but can be implied from the mathematical derivations and by other material in supporting publications by the same author, such as [42]. This last publication accompanies a neural simulator created by Pecevski, Kappel and Jonke, called NEVESIM, which was specifically developed to allow zero-delay simulations. NEVESIM was also used for the computer simulations in Pecevski & Maass [43], further reinforcing the claim that the simulations used no synaptic delay.[16]

Zero delay is of particular significance since biological networks violate that assumption. Electrical perturbations in real neurons undergo more than one form of transmission delay: membrane activity takes time to propagate down dendrites and axons—*cable theory* is a subject that deals with the mechanics of electrical propagation, introduced in the Scholarpedia article at [40]; moreover, synapses are not instantaneous in their action—chemical synapses typically take a few milliseconds for transmitters to make the leap across the gap, at least 0.3 ms [20]. Small, microsecond delays may be necessary in hardware implementations, and it is certainly possible that conduction delay plays a key role in determining neural connectivity patterns [2], and therefore, in determining the computational properties of a network.

vised and supervised training steps to be used for discriminative purposes.

[16]The primary author of the article was not responsive to email questions on the subject.

The zero delay assumption is also significant because any violation is bound to introduce discrepancies between theoretical and simulated (as well as biological) network properties. In [42] Pecevski et al. state that "allowing small non-zero delay in the synaptic connections also leads to functional neural sampling models in most cases", but this is not explored in the later publication ([43]) that forms the basis of the present work. Indeed, the exploration of discrepancies between theory and simulation is a secondary motivating factor of this dissertation.

Be that as it may, under these assumptions Equations 2.1 and 2.2 can be combined to describe the firing intensity of output neuron ζ^l at time t by

$$\rho^l(t) = \sum_{j=1}^{J^l} \frac{1}{\tau} \exp\Big(b_j^l + \sum_{i=1}^{l} \sum_{m=1}^{M(x^i)} w_{im,j}^l x^{im}(t)\Big), \tag{2.8}$$

where j indexes over the number of hidden neurons connected to the output neuron ζ^l, b_j^l is the bias of the j^{th} α neuron of the l^{th} subpopulation, and m indexes over the number of possible values the random variable x^i can take. Here, $x^{im}(t)$ is a binary random variable that assumes the value 1 if and only if $x^i = m$ at that point in time, or equivalently, if χ^{im} fired in the time interval $[t-\tau, t]$ [43]. Thus, the inner two summations effectively add up the weights of synapses that connect activated input neurons with the α neuron currently indexed by the outer summation.

This equation can in turn be used to derive an analytic expression for the generative model embodied by the network connectivity and weights:

$$p(\mathbf{a}, z, \mathbf{x}; \theta) = \frac{1}{A(\theta)} p(z|\mathbf{a}) \exp\Big(\sum_{i,m} \sum_{l,j} \hat{w}_{im,j}^l x^{im} a_j^l + \sum_{l,j} \hat{b}_j^l a_j^l\Big), \tag{2.9}$$

where $A(\theta)$ is a normalisation constant and θ is the parameter vector made up of the hidden layer weights and biases. This distribution introduces the random variable $\mathbf{a}$, which stands in for the activity in the α layer, such that $a_j^l = 1$ if and only if α_j^l is active. Since activity in the α layer is competitive and the refractory period prevents further activity within a time window τ, there can only be one

active neuron at any time t, and therefore only one coordinate of the vector $\mathbf{a}$ can be non-zero, for a minimum duration of τ.[17] The hatted weights and biases above are simply shifted for convenience by the baseline constants with respect to unhatted weights and biases in Equation 2.8:

$$\hat{w}^l_{im,j} = w^l_{im,j} + w_- \tag{2.10}$$

$$\hat{b}^l_j = b^l_j + b_-. \tag{2.11}$$

Moreover, there is a simple deterministic relationship between z and $\mathbf{a}$, such that $p(z = l_1|\mathbf{a}) = 1$ when $a^{l_2}_j = 1$ and $l_1 = l_2$ for any j, and 0 otherwise.[18] This analytic formula can easily have $\mathbf{a}$ marginalised out to derive the theoretical joint distribution $p(z, \mathbf{x}; \theta)$ embodied by the network [43]. It is made use of extensively for fitness measurement.

2.6 Kullback-Leibler Divergence

Kullback-Leibler (KL) divergence[19] is a widely-used measure of the difference between an approximate and target probability distributions. Defined as

$$D_{KL}(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}, \tag{2.12}$$

where P and Q are discrete probability distributions, it encodes the loss of information[20] that occurs when a target distribution Q is approximated by a second distribution P [8]. Although widely referred to as a "distance", it is strictly speaking non-symmetric and therefore does not satisfy the mathematical requirements of a distance metric. Thus, $D_{KL}(P||Q) \neq D_{KL}(Q||P)$. In our case, as a SAM module

[17] As in the input layer, a spike at t in α^l_j sets the random variable a^l_j to 1 until $t + \tau$.

[18] This is a result of the deterministic relationship between spiking activity in the alpha layer and connected output neurons mentioned in Section 2.2.

[19] Also known as *relative entropy*.

[20] In the information-theoretic sense.

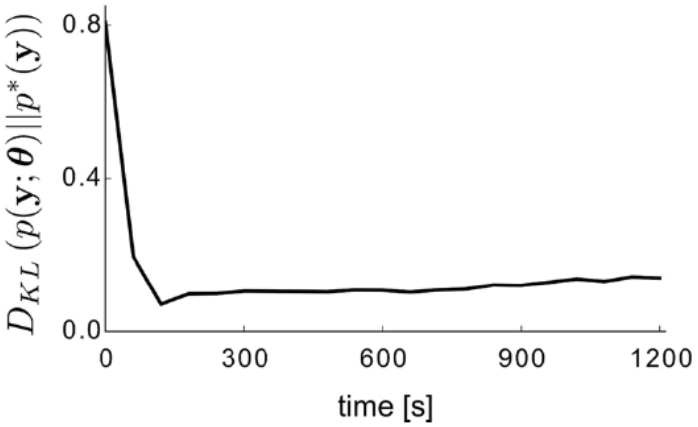

Figure 2.9: KL divergence between the internal model of a SAM network and its target distribution, as a function of simulated biological time. Note the slight increase of divergence after an initial sharp drop-off. This results from the use of realistic PSP shapes (whereas theoretically optimal shapes are unrealistic). From [43].

learns its target distribution, KL divergence decreases (but see Figure 2.9).

2.7 Learning-to-Learn

The technique presented by Pecevski & Maass and reinforcement techniques such as [22] offer good approximation of target distributions, but they are extremely computationally demanding, typically running on supercomputers or limited to very simple learning tasks or network configurations. Moreover, the approach in Pecevski & Maass was not meant to generalise to other distributions. It is the main motivating factor of the present work to tackle this issue.

In light of the above, *learning-to-learn* (LTL) can be seen as an effort to solve the efficiency problem, as well as to enable qualitatively new learning processes. LTL in SNNs is an embryonic paradigm currently being explored by the Graz research team, whose objectives are three-fold:[21]

1. To discover initial configurations and hyperparameters that lead to more

[21]These objectives were communicated via email correspondence with Prof. Maass' research team. See Chapter 8.

effective and generalisable learning

2. To abstract from these experiments computational principles that enable learning generalisability and phenomena like one-shot learning and multi-task capabilities

3. To implement these principles at an appropriate level of abstraction in simulated networks

Although no research papers on LTL in SNNs have yet been published by the Graz team,[22] related ideas have been applied with some success in deep learning. See [45]. Nevertheless, note that the present work extensively cites publications from the Graz research team, building on research concepts that have already been peer-reviewed or are in pre-print.

Fulfilment of the second and third LTL objectives require a good understanding of the first, and are therefore beyond current research. The first objective, however, can be approached with existing knowledge. It is essentially an *optimisation problem*.

2.8 Evolutionary Algorithms

Optimisation is an important subfield of mathematics and computer science with a long history. Typically, optimisation problems involve finding the parameters of a function, model or algorithm that correspond to a global maximum or minimum of some fitness or error landscape. Fitness (or error) is measured by a metric that is convenient to the domain in question—mean squared error and the L^2 norm being common choices. Pecevski & Maass [43] uses KL divergence, a natural choice in this scenario. The present work retains the measure.

As such, a solution to an optimisation problem is "best" with respect to the criterion set up by the fitness or error function. Methods are many and varied,

[22] One paper is in preprint at the time of writing: Bellec et al. [4].

and a thorough discussion would take us far afield. See [51] for a brief overview of techniques.[23] In general, the choice of optimisation method depends partly on what is known about the fitness landscape; e.g. is the general shape well-understood? can the derivative be approximated in some local region about any point? is the problem constrained? is it deterministic or stochastic? It also depends on other properties of the domain, such as whether it is a multi-objective problem and whether the hyperparameters are discrete or continuous, among other things.

Our problem is stochastic, constrained and single-objective: which initial neural configuration gives the best mean performance on a family of related tasks? It is stochastic because of the noisy nature of our neuron model, which can cause the same configuration to give different learning performances during different runs. It is constrained because there are reasonable bounds on most of the parameters. Finally, it is single-objective because we are only looking to optimise a single, scalar value. Little else is known about the fitness landscape, except that it is probably complex: whether it has one maximum, for instance, or several local maxima, is unknown.

These properties impose few restrictions on the choice of optimisation technique. Nevertheless, local optimisation techniques are typically susceptible to noise and may fall prey to local maxima, especially in complex landscapes [51]. Global optimisation techniques like genetic algorithms or evolutionary strategies appear to be reasonable alternatives in this domain. These are techniques inspired by natural evolution, involving repeated application of competitive selection, crossover and mutation, processes referred to as *operators*. When applied cyclically over multiple generations, these operations tend to "breed" populations of "better" individuals.[24] Goldberg [13] gives an excellent introduction.

Optimisation techniques under the umbrella of evolutionary algorithms borrow their terminology from biological evolution. Indeed, *genetic algorithms* (GAs) rep-

[23] There are few good paper-length reviews of so vast a subject. Each family of optimisation methods can easily be expanded to fill a volume.

[24] Better in the sense of higher mean fitness (or less mean error).

resent each "individual" (or solution) by a "chromosome" consisting of the "genes" (parameters) that define the individual. In GAs, chromosomes are typically bit strings that encode the value of the parameters, concatenated together, with the operators performing transformations at the bit level. A "generation" consisting of a "population" of several individuals is evaluated and the fittest individuals are "selected" for "reproduction", which involves two "parents" and normally leads to a pair of "offspring". The offspring are usually a genetic combination of their parents, although not necessarily. Moreover, "mutation" may randomly alter chromosomes to explore genetic possibilities that are harder to find through simple recombination of parent genes. The offspring of one population form the next generation, and the algorithm repeats.

Algorithmic variations are many, but the three key steps are:

1. **Selection:** The purpose of the *selection* operator is to—deterministically or otherwise—pick the best fitness individuals for the next operation. In *elitism*, for instance, a proportion of the fittest individuals are retained unchanged in the next generation, ensuring that "best" individuals are not discarded.[25] In *roulette-wheel selection*, an individual is probabilistically chosen in proportion to its fitness value, normalised by the sum of all fitnesses. In *tournament selection*, the fittest individuals in a random subset of the population are retained, with the process repeated until all individuals in the next generation are picked.

2. **Crossover:** *Crossover* or *recombination* operators explore and exploit the solution space, combining the genetic code of the best candidates selected by the last operator to generate fitter individuals. *One-point crossover* recombines the genetic code—the bit strings—of two parents by picking a crossover point in the representation and joining the code up to that point from one parent with that from the second. *Two-point crossover* picks two random points and swaps the genetic code in between, and *uniform crossover* swaps

[25] That is, recombination is not applied.

each coding bit randomly, typically with a probability of 0.5. Figure 2.10 demonstrates these three common types of crossover using bit strings as representation. Although the subject is fraught with difficulties, Goldberg [13] gives a thorough discussion of why GAs implemented with at least a selection and recombination operator tend to arrive at good solutions of the optimisation problem based on the *building block hypothesis*. However, it is also pointed out that like many other optimisation techniques, GAs suffer in domains where the fitness landscape has remote optima, that is, when the solution is akin to finding a "needle in a haystack" [13].

3. **Mutation:** *Mutation* is an operator that randomly modifies genetic representations for better exploration of the solution space. As in its biological counterpart, mutation modifies genetic sequences in ways that can be impossible to achieve through reproduction, and that might be beneficial to the individual. Thus, it avoids scenarios where recombination on its own converges onto local optima. With bit string representations, the mutation operator flips bits at random positions, albeit with a small, domain-dependent probability (typically 0.01 or less) [38].

There is some overlap between *evolutionary strategies* (ESs) and GAs, and in fact evolutionary strategies also involve the use of selection, crossover and mutation operators. However, in evolutionary strategies chromosomes normally represent genes by their real values, rather than bit strings, and manipulation occurs not at the level of bits, but through arithmetic operations on the real values. ESs tends to better model the natural representation of the problem: small changes in individuals leading to small changes in fitness.

A key difference between ES and GA is that in generic ES an individual comprises the *object parameter vector*—the parameters that we intend to optimise—as well as a set of *strategy parameters*, as opposed to only the object parameter vector in GA. Strategy parameters encode the rate of mutation of the parameter vector,

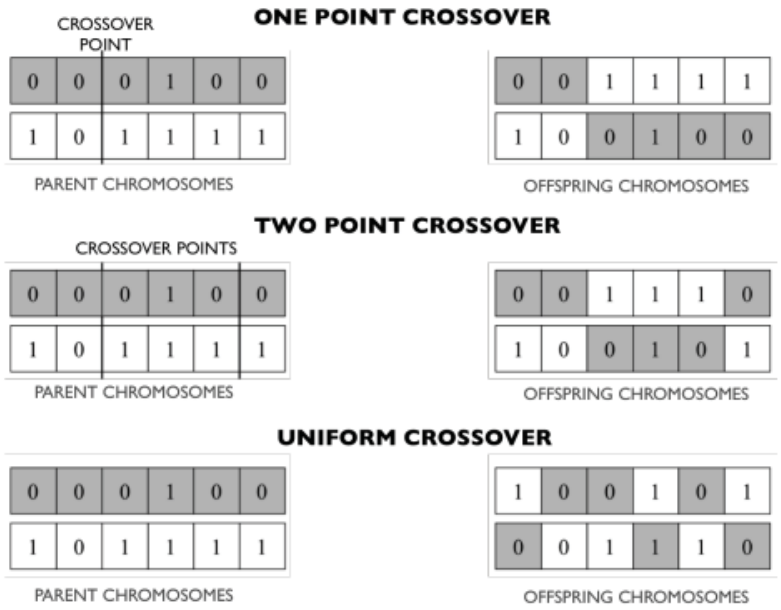

Figure 2.10: An illustration of one-point crossover, two-point crossover, and uniform crossover, recombining the genetic code from two parents to create two children with complementary chromosomes. From [36].

and are themselves mutated in each cycle of operator application.[26] This makes the algorithm self-adaptable. The peer-reviewed ES article on Scholarpedia gives a good review of the topic [5].

ES methods are diverse, but a common use case is unconstrained search in real-valued spaces $\mathbb{R}^n$, for which *self-adaptation-ES* is often applied. Under this scheme, the strategy and object parameters of the l^{th} offspring are recombined and mutated according to the arithmetic operations defined by

$$\sigma_l \leftarrow \langle\sigma\rangle e^{\tau N_l(0,1)}, \tag{2.13}$$

$$\mathbf{y}_l \leftarrow \langle\mathbf{y}\rangle + \sigma_l \mathbf{N}_l(\mathbf{0}, \mathbf{I}), \tag{2.14}$$

respectively. Here, $\langle\cdot\rangle$ denotes the arithmetic mean, σ is the strategy (mutation) scalar, $\mathbf{y}$ is the object parameter vector, and $N(0,1)$ and $\mathbf{N}(\mathbf{0}, \mathbf{I})$ are scalar and vector forms of the normal sampling function respectively, with centre 0 and standard

[26] The strategy parameters are mutated *before* the object parameter vector.

deviation 1. σ_l acts as the standard deviation of the normal components in the second assignment. τ is a learning parameter typically proportional to $1/\sqrt{n}$. The arithmetic means are calculated using the fittest individuals that result from the selection process. The effect of these operations is to add random noise to the mean solution, the extent to which noise is added itself determined by a noisy sampling of the previous mean mutation rate. Self-adaptation allows *self-tuning* of mutation rates without external control, which is not possible in GAs. Note also that recombination and mutation occur as a combined process, as opposed to the discrete steps in GAs. A popular and highly successful variant of ES is called *covariance matrix adaptation-ES* (CMA-ES), which replaces scalar mutation by a covariance matrix, and is therefore able to adapt the shape of the mutation distribution to better fit the fitness landscape [5].

Natural evolution strategies (NESs) are a subtype of ESs suggested by Wiesrtra et al. [54] that iteratively update a *search distribution* of individuals, as opposed to maintaining a population of individuals directly. NES represents the state-of-the-art in ES [54]. In NES, the search distribution is used to generate individuals for fitness evaluations, from which the algorithm estimates a parameter step in the direction of better expected fitness while taking into consideration uncertainty so as to avoid detrimental effects such as oscillation or premature convergence. The algorithm assumes a parametric form of the search distribution with a known derivative—commonly the multinormal distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

The search gradient and parameter updates are found by computing the derivative of the expected fitness in the search parameter space. For a sample of λ individuals, the derivatives of the expected fitness J along the multinormal parameter directions are given by

$$\nabla_{\boldsymbol{\mu}} J \leftarrow \frac{1}{\lambda} \sum_{k=1}^{\lambda} \nabla_{\boldsymbol{\mu}} \log \pi(\boldsymbol{z}_k|\theta) \cdot f(\boldsymbol{z}_k) \tag{2.15}$$

$$\nabla_{\boldsymbol{\Sigma}} J \leftarrow \frac{1}{\lambda} \sum_{k=1}^{\lambda} \nabla_{\boldsymbol{\Sigma}} \log \pi(\boldsymbol{z}_k|\theta) \cdot f(\boldsymbol{z}_k), \tag{2.16}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are the mean and covariance matrix[27] of the distribution respectively, $\pi(\boldsymbol{z}_k|\theta)$ is the likelihood of individual $\boldsymbol{z}_k$ conditioned on the parameters θ and $f(\cdot)$ is the fitness. The updates to the search distribution parameters are then simply

$$\mu \leftarrow \mu + \eta \cdot \nabla_{\boldsymbol{\mu}} J \tag{2.17}$$

$$\Sigma \leftarrow \Sigma + \eta \cdot \nabla_{\boldsymbol{\Sigma}} J, \tag{2.18}$$

where η is a small learning rate [54]. For robustness, NES modifies the updates in Equations 2.17 and 2.18 by pre-multiplying the update step ∇J with the inverse of the Fisher information matrix $\mathbf{F^{-1}}$ [54]. This takes into account uncertainty and renders the update direction independent of the search distribution parametrisation.

Evolutionary algorithms have been successfully used in *neuroevolution*, a family of generic techniques that evolves ANN architectures to improve their performance [31]. Stanley et al. [48] was particularly successful in mimicking nature by exploiting scale invariant connectivity motifs to reduce the dimensionality of the problem. An article in preprint [49] demonstrates that very simple GAs can rival popular methods based on backpropagation such as Q-learning and policy gradients when applied to large scale deep neural networks (DNNs) with over four million free parameters. Another article in preprint[47] follows a very similar methodology to demonstrate that ESs are also good alternatives to Q-learning and policy gradient techniques. Finally, it is worth noting that connectivity and other parameters of brain networks were similarly optimised through a long evolutionary process for efficient learning. The present work aims to reconstruct this process, at least on a conceptual level.

[27] Also called a *mutation* matrix within the context of NES, since it determines the spread of candidate solutions around the generation mean, the latter of which should be the optimal solution—or close to it—of that particular generation.

3. Design and Implementation

3.1 Objectives

The core objective of the present work falls under the first LTL directive in Section 2.7: optimising the configuration of an SNN in order to accelerate and improve learning ability. Specifically, the work builds on Pecevski & Maass [43] and poses the optimisation problem, **which neural configuration gives the best mean performance on two families of related density estimation tasks after a suitable training run, as measured by the KL divergence between the learned and target distributions?**

A secondary objective of this thesis is to push biological plausibility by introducing realistic inhibition and other changes to the neural architecture. Pecevski & Maass [43] uses strong WTA inhibition which, in the idealised form embodied by the SAM architecture it proposes, would be implausible in brain circuitry. In addition, biological synapses exhibit small connection delays that are not modelled in Pecevski & Maass. Both of these deficiencies are taken up by proposing a novel architecture: the sparse probabilistic inference (SPI) module (see Section 3.3).

A third, minor objective is to explore the effect of deviation from theoretical assumptions. Section 2.5 describes how an underlying assumption in [43] and other work is that synapses propagate spikes with no delay. This was used to derive theoretical results for idealised architectures that are not found in nature. Thus,

part of the analysis will be devoted to experimenting with incremental increases in synaptic delay, to deduce any trends in performance impact.

A final objective is to understand the sensitivity of optimal configurations to perturbations in their hyperparameters.[1]

The work will test the performance of two architectures (SAM & SPI) on two density estimation task families: a conditional distribution with one independent and two dependent random variables, and a joint probability distribution that models a visual perception problem with four random variables.

3.2 Overview

Below, the term *inner-loop optimiser* refers to the unsupervised SNN algorithm that learns the target distribution—that is, the effect of STDP on the connections between the input and α-layer of the SAM or SPI module. *Outer-loop optimiser* refers to the evolutionary algorithm that suggests parameter configurations and optimises them for performance.

This terminology arises from the fact that STDP can bestow to a network the ability to optimise its connectivity for the purposes of density estimation, and one therefore needs to carefully distinguish between this type of optimisation occurring internally—a biologically plausible one that happens during simulated biological time—and the evolutionary search which, although having conceptual parallels with the long-term biological process of evolution, is not explicitly intended to be plausible and happens outside simulation. Also, it is crucial to point out that outer-loop optimisation does not affect the learning process, beyond setting up the network and inner-loop algorithm parameters. Instead, learning happens during simulations in an online manner.

Logically, the work can be described as a sequence of steps:[2]

[1] Note that synaptic delay—a key parameter—is not one of the optimisable hyperparameters.
[2] In practice, actual work did not proceed linearly.

1. Understanding which hyperparameters are to be optimised by the outer-loop optimiser
2. Defining a suitable encoding of these parameters
3. Defining families of tasks of different complexity that can be carried out by the inner-loop optimiser
4. Designing and implementing the overall simulation that runs outer-loop optimisation on the inner-loop algorithm
5. Running the combined algorithm on the tasks and evaluating each architecture-configuration combination on each family of tasks
6. (Supplementary) Exploring these results for commonalities that might have implications on the second LTL directive

SAM has already been described in Section 2.2; thus we shall proceed directly to the novel SPI architecture. Following that, subsequent subsections roughly map onto and flesh out steps 1–6 above.

3.3 Sparse Probabilistic Inference Modules

A novel architecture is suggested that generalises the SAM architecture proposed by Pecevski & Maass [43] by introducing multiple neurons per encoded value, *sparseness* (i.e. a non-unitary probability of connection between neuron pairs), recurrence within neuron pools, and random synaptic delays uniformly spread over a range of delays. We shall call the architecture a *sparse probabilistic inference* (SPI) module. All of these changes are introduced on the basis of biological plausibility (see, for example, [18, 29]).

Analogous to the SAM module, the purpose of this architecture is to perform density estimation of conditional distributions, i.e. each module can be used to estimate the dependence of one variable y^i on a set of other variables,

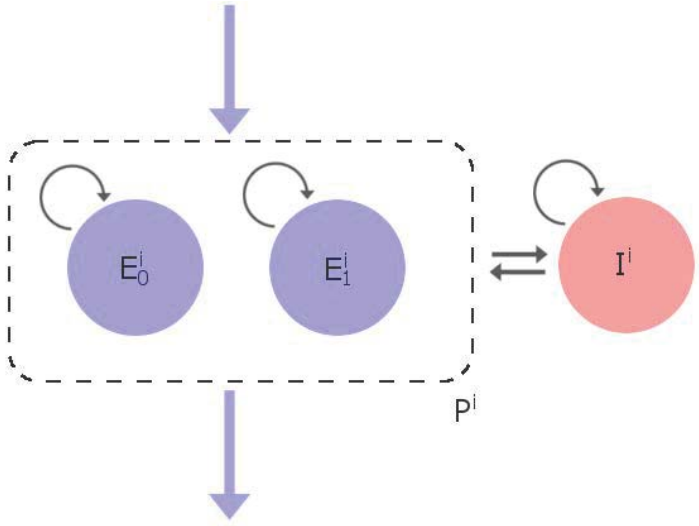

Figure 3.1: SPI module that represents variable y^i, consisting of a population of excitatory neurons P^i and an inhibitory pool of neurons I^i. The excitatory pool has subpopulations E_j^i, each of which codes for one value of variable y^i. In the case displayed, the excitatory pool represents a binary random variable y^i. Connections between pools are represented by straight black arrows, and recurrent connections within pools are represented by circular black arrows. Note that the activity of population P^i can be input into other modules representing other variables. Cross-module connections of this type are shown by a vertical blue arrow. Although only one input arrow is shown, a module can have as many inputs as necessary to represent all dependencies. All connections are sparse.

or $p(y^i|y^1, ..., y^{i-1}, y^{i+1}, ..., y^m)$. SPI modules can also be combined in recursive networks like SAM modules to estimate joint distributions as alluded to in Sections 2.1–2.2 (a detailed description will be provided in Section 4.1). Figure 3.1 gives a schematic of each module's architecture.

Variable y^i is encoded in SPI module i by a pool P^i that consists of n excitatory subpopulations E_j^i, each of which codes for one value of y^i in $\{1, ..., n\}$. Unlike SAM modules, therefore, each value of y^i is encoded by the group activity of a collection of neurons, rather than a single neuron. In fact, each subpopulation ideally has several neurons to avoid small number effects. The 0-value is encoded, as in [43], by inactivity.[3]

[3]The 0-value is a special case that is not given much significance in Pecevski & Maass, because under theoretical assumptions the neural network can be made to avoid inactivity most of the time.

For purposes of continuity and comparison, SPI modules use a stochastic integrate-and-fire neuron model with rectangular or α-shaped PSPs as in the SAM architecture (see Section 2.3). Synapses also retain the same dynamics from Pecevski & Maass [43], and input-excitatory synapses are similarly subject to the same simplified STDP. Unless specified, variables that specify model dynamics are retained from that publication. However, the output layer ζ present in the SAM architecture is eliminated. More importantly, since multiple neurons can be simultaneously active, network states are determined in a different manner: instead of making the assumption that only a single neuron can fire within a short time (as in a WTA circuit, and zero-delay SAM), network states are determined by smoothing activity using a temporal kernel. Details of network interpretation are given later in Section 4.2.5. Inhibition in SPI modules is by means of a pool of inhibitory neurons I^i that is activated by spiking in excitatory neurons of P^i.

This opens up the opportunity to use "softer" inhibition that does not make the rigorous, biologically unrealistic demands that are assumed in [43]. Indeed, soft or *divisive inhibition* has been explored recently in Jonke et al. [18] and Legenstein et al. [29], where it was shown that it can account for a broader range of natural phenomena than hard WTA-like inhibition, such as the self-organising ability of neuron populations to become sensitive to select features by forming receptive fields.

Anecdotal tests run during an exploratory phase confirmed that well-tuned SPI modules can exhibit non-exclusive competitive inhibition of excitatory subpopulations, while also demonstrating stochastic state-switching.[4] These are important properties, the former because competition is integral to neural computation (see [33] for the benefits of WTA-like competition, [18] for that of softer, divisive inhibition), the latter because state-switching is necessary to represent intermediate probabilities in $(0, 1)$. See Figure 3.2 for a spike raster plot of an SPI module's

[4]That is, for the same input a module can from time to time switch the subpopulation with highest activity.

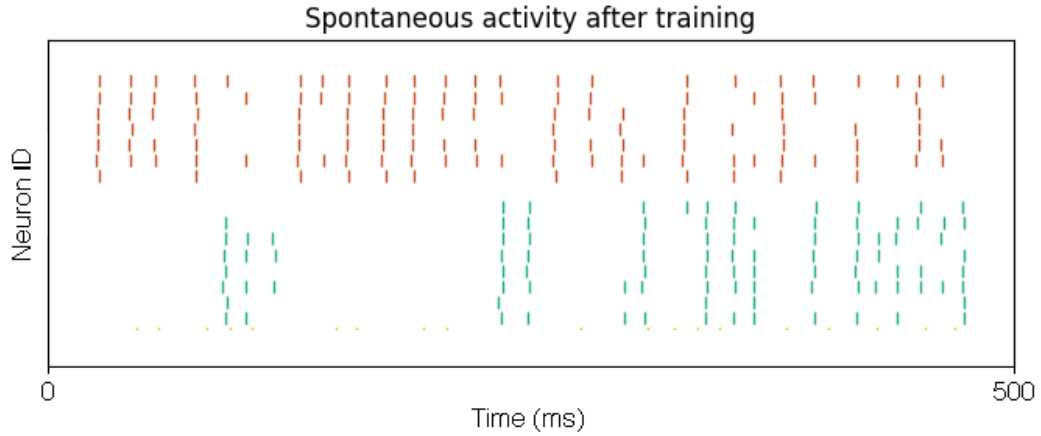

Figure 3.2: A spike raster plot showing spontaneous activity in an SPI network over a duration of 500 ms. The two colour groupings indicate subpopulations that code for different values of the same variable. Note that while activity is dominated by the orange subpopulation, there are clear short intervals during which the green subpopulation has higher activity. This represents a stochastic switch of the variable value encoded by the population. Additionally, although activity is competitive and nearly exclusive to one or the other subpopulation, there are also periods during which both subpopulations are simultaneously active.

excitatory neurons that exhibits these phenomena.[5]

There are five connection types in this architecture:

1. P^i–P^j-type excitatory connections between excitatory pools of different modules. These connections have random synaptic delays, and are subject to STDP as in Pecevski & Maass [43].

2. E^i_k–E^i_k-type recurrent excitatory connections,[6] with fixed delays and fixed weights.

3. P^i–I^i-type excitatory connections, with fixed delays and fixed weights.

4. I^i–I^i-type recurrent inhibitory connections, with fixed delays and fixed weights.

5. I^i–P^i-type inhibitory connections, with fixed delays and fixed weights.

[5] A spike raster plot is a temporal plot where neuron activity is represented by small vertical bars coinciding with the time of spiking.

[6] In this context, "recurrent" refers to pairwise connections between neurons of the same pool.

This categorisation achieves a 5-fold division of synaptic connections on the basis of function, each of which are separately parametrised for the purpose of optimisation. Weights (or weight ranges) and connection probabilities are in this manner separately specialised for every functional group. Note that there are no excitatory connections between excitatory subpopulations of the same module.

SPI is a relevant alternative to the architecture proposed by Pecevski & Maass because:

1. SPI uses soft inhibition, as opposed to WTA-like inhibition
2. SPI has sparse rather than all-to-all connections
3. SPI allows recurrent connectivity within subpopulations
4. SPI uses random synaptic delay between input and excitatory neurons, as opposed to the zero delay assumed for the theoretical results in Pecevski & Maass

All of these properties are biologically plausible desirables not present in the SAM architecture suggested by Pecevski & Maass. Moreover, the first three have demonstrably important consequences on the computational properties of spiking neural networks [18, 29].

3.4 Hyperparameters

A tenet of this work is that there should be a clear distinction between the domains of outer-loop and inner-loop optimisation. Whereas inner-loop optimisation deals with the learning aspect of the algorithm, outer-loop optimisation should solely be dedicated to the search problem—finding ideal inner-loop learning parameters. The choice of hyperparameters evolved by the outer-loop optimiser is thus informed by this distinction: the evolutionary algorithm must not optimise parameters that are too granular, e.g. individual synaptic weights or biases, nor, ideally, should

there be ad hoc variables unless there is a reasonable basis for them. All else being equal, a parsimonious model with fewer parameters is preferable and biologically more relevant.

To avoid combinatorial explosion, the number of hyperparameters must be kept manageable. To this end, most of the variables selected for outer-loop optimisation are global parameters that describe the broad structure of the network or that control inner-loop learning. An interesting approach, however, is to divide the neural network into functionally similar groupings, each with a unique set of parameters. Following on Section 3.3, some parameters may be related to connection probabilities or weights between subpopulations. These subpopulations are distinguished on the basis of functional role, and having distinct variables for each is justified on evidence from neurobiological data [3]. Indeed, recent publications on similar architectures use different connection parameters between subpopulations with different functional roles [18, 29]. Aside from subpopulation parameters, both SAM and SPI modules in the complex joint estimation task require that input be balanced on a per-module basis, demanding specialised variables for each (bias baselines & maximum afferent weights; see Chapter 5). Although this was also done in [43], it remains an open question as to how this specialisation could be implemented in brain circuitry.

Both SAM and SPI models have several parameters that are evolvable. Owing to unknown interactions between variables, a reasonably wide range of values over which to optimise was decided upon in an initial testing phase after consulting existing literature that proposes similar architectures (e.g. Jonke et al. [18] and Pecevski & Maass [43]). Because of limited computational resources, however, some parameter choices were made on the basis of compromise.[7]

SAM hyperparameters, along with the ranges over which their values could evolve during outer-loop optimisation, are given in Table 3.1. All of the parameters have the same meaning as in Pecevski & Maass, except where new variables are

[7]Such as the number of neurons per subpopulation in SPI networks.

introduced:

- *Bias baseline* b_-, a constant used in Equation 2.11 to shift the dynamic intrinsic plasticity b.

- *Weight baseline* w_-, a constant used in Equation 2.10 to shift the dynamic synaptic plasticity w.

- T: Pecevski & Maass [43] introduces a further scaling factor T to Equations 2.5 and 2.6 for the purposes of the simulation only. In particular, the potentiation part of the weight update becomes

$$\Delta w = e^{-T(w+w_-)} - 1, \tag{3.1}$$

  and bias updates become

$$\Delta b = \tau e^{-T(b+b_-)}. \tag{3.2}$$

- *Relative bias update rate*, R, which scales the learning rate of the bias update upon spiking relative to the rate of continuous spike-independent bias decay (Equation 2.7). Thus, Equation 3.2 is modified once more to become

$$\Delta b = R\tau e^{-T(b+b_-)}. \tag{3.3}$$

  A non-unitary value demonstrated better convergence in preliminary optimisation tests.[8]

- *First bias rate* and *second bias rate*, η'_0 and η'_1 respectively. Pecevski & Maass allows intrinsic plasticity to proceed at different rates, depending on biological time since simulation start. In [43], the first bias rate is fixed for a training duration of 600000 ms of biological time, at which point it discontinuously jumps to the second bias rate. The training duration is therefore another

[8] Relative bias update rate was introduced by the present author.

relevant parameter, but this is fixed for all simulations in the present work. Some simulations, however, only have one bias rate.[9]

- *Initial STDP rate* and *final STDP rate*, η_0 and η_1 respectively. These variables are the extrinsic plasticity analogues to the first and second bias rates, and are similarly affected by the training duration. Unlike intrinsic plasticity, however, the learning rate governing weight updates changes linearly from the initial to the final rate.[10] After the training duration—immediately after the final STDP rate is reached—extrinsic learning is stopped (that is, STDP learning rate is set to 0), whereas bias learning continues at the second rate).

- *Firing rate scaling coefficients* c_1 and c_2. Pecevski & Maass assumed the firing intensity given by Equation 2.1. These new coefficients, on the other hand, introduce scaling variables so that the intensity equation becomes

$$\rho(t) = c_1 \exp(c_2 \cdot V_m(t)). \tag{3.4}$$

  c_1 scales all probability calculations equally, while c_2 effectively determines the sharpness of the kink in the exponential curve at $V_m = 0$. Note that c_1 replaces the static $1/\tau$ in Equation 2.1 to offer more control to the optimisation algorithm.

Table 3.2 gives a list of outer-loop optimisable parameters of the SPI module and their ranges. Most hyperparameters are reused from the SAM architecture. The rest mainly deal with connectivity:[11]

- *Connection probabilities* ρ_{PP}, ρ_E, ρ_{PI}, ρ_I and ρ_{IP}. Recall that an SPI module for a variable y_i consists of an excitatory pool of neurons P_i and an inhibitory

[9] Those that do are specified.

[10] Hence the usage of "initial" and "final", as opposed to "first" and "second".

[11] Note that there is only one bias rate. Because of increased complexity, SPI simulations take much longer than simulations of SAM networks, and training durations have to be cut. However, this can be done in a way that minimises impact, as shall be described below. The upshot is that there is only one bias rate for the entire training duration.

Table 3.1: Evolvable hyperparameters of the SAM module

Hyperparameter	**Minimum**	**Maximum**
b_-	-40	0
w_-	-10	0
T	0	1
R	0.00001	1
η'_0	0	0.1
η'_1	0	0.1
η_0	0	0.01
η_1	0	0.01
c_1	0	1
c_2	0	5

pool I_i. The excitatory pool P^i, in turn, contains multiple subpopulations E^i_k, with one k for each possible value of the variable y^i (see Figure 3.1). ρ_{PP} is then the probability of forming P^i–P^j-type excitatory connections, ρ_E the probability of forming recurrent E^i_k–E^i_k-type excitatory connections, ρ_{PI} the probability of forming P^i–I^i-type excitatory connections, ρ_I the probability of forming recurrent I^i–I^i-type inhibitory connections and P_{IP} the probability of forming I^i–P^i-type connections (see Section 3.3 for details about connection types). Note that connections are directed, so that we need reciprocal variables P_{PI} and P_{IP} if we are to describe circular connectivity between the two pools. Each probability variable P_{XY} describes the pairwise probability of establishing a connection between any two neurons picked randomly from X and Y respectively. Thus, if X and Y both have 10 neurons and $P_{XY} = 0.9$, for instance, we expect 90 connections from X to Y.

- *Maximum afferent connection weight* w_{MAX}, which determines the maximum weight of an input connection into the excitatory pool of an SPI module. In the more complex tasks, this variable is specialised for each module to help balance activity from a variable number of input modules. Note that this weight corresponds to P^i–P^j-type connections, which are dynamic throughout the learning phase. Therefore, only the maximum possible weight is fixed

Table 3.2: Evolvable hyperparameters of the SPI module

Hyperparameter	**Minimum**	**Maximum**
b_-	-50	0
w_-	-10	0
T	0	1
R	0.00001	1
η'	0	0.1
η_0	0	0.01
η_1	0	0.01
c_1	0	2
c_2	0	4
ρ_{PP}	0	1
ρ_E	0	1
ρ_{PI}	0	1
ρ_I	0	1
ρ_{IP}	0	1
w_{MAX}	0.01	8
w_E	0	14
w_{PI}	0	14
w_I	-14	0
w_{IP}	-14	0

as a hyperparameter.

- *Fixed connection weights* w_E, w_{PI}, w_I and w_{IP}. These weights correspond to the remaining four connection types, which are initialised by the outer-loop optimisation algorithm and stay fixed throughout the simulation. Note that inhibitory connections are represented by negative weight values.

There are also fixed parameters which, while not optimisable, are relevant because outer-loop optimisation proceeds on the assumption of their given values. These parameters are mainly reused from Pecevski & Maass, with the same semantics and values for both SAM and SPI modules. Other fixed constants are introduced to manage the increased complexity of SPI modules:

- *PSP amplitude*, which determines the size of the effect on membrane voltage that each spike has on connected postsynaptic neurons.

- *Number of inhibitors* [SAM only]. Each α-neuron in the SAM architecture is connected to a fixed number of inhibitory neurons to achieve WTA-like competition.

- *Excitatory bias minimum* b_{MIN} and *excitatory bias maximum* b_{MAX}, the minimum and maximum bias values that can be achieved in dynamic α-layer neurons (SAM) or excitatory pool neurons (SPI).

- *Weight maximum* [SAM only]. Dynamic synapses in SAM modules (i.e. χ–α connections) have a maximum weight, which helps in balancing input. This value depends on the task (see Section 4.2).

- *Learning time*, the duration during which training samples are drawn from the target distribution and presented to the module as spiking input. During this time, STDP learning rate changes linearly from the initial to the final learning rate. Bias, on the other hand, stays fixed throughout at the first rate value, and jumps discontinuously to the second rate at the end of the training duration.

- *Neurons per subpopulation* [SPI only], the number of neurons per SPI subpopulation.[12] Note that an excitatory pool has 1 or more excitatory subpopulations, but the inhibitory pool consists of only one subpopulation.

- *Synaptic delay* [SAM only]. This defines the fixed synaptic delay—that is, the time it takes for a presynaptic spike to arrive at a postsynaptic neuron—between each pair of connected neurons in the SAM architecture.

- *Minimum* and *maximum input synaptic delay* [SPI only]. Recall that input synapses in SPI modules—that is, synapses impinging on excitatory pools from other modules—have fixed random synaptic delays spread over a range of time values. The minimum and maximum define this range.

[12] This number is highly constrained by execution time.

- *Fixed synaptic delay* [SPI only]. Other synapses in SPI modules have a uniform fixed delay defined by this variable.

- *Initial excitatory bias mean* and *standard deviation*. Neurons in the α-layer (SAM) or excitatory pools (SPI) are initialised with a random bias drawn from a normal distribution with these parameters.

- *Initial χ–α weight mean* and *standard deviation* [SAM only]. Synapses connecting the input and α neurons in the SAM architecture are initialised with a random weight drawn from a normal distribution with these parameters.

- *α–inhibitors weight* and *α–ζ weight* [SAM only]. These are large fixed weights that trigger postsynaptic activity with near-certainty.

- *Inhibitors–α weight* [SAM only]. This determines the inhibitory strength—hence the negative sign—of connections from inhibitors to α-neurons.

- *Initial P–P-type weight mean* and standard deviation [SPI only]. Inter-module connections in SPI networks are initialised with a random weight drawn from a normal distribution with these parameters.

- *Inhibitory neuron bias*, which determines the fixed bias of all inhibitory neurons.

- *Time constant τ*, a value that is used in several contexts. It determines the time window over which network states are measured, the refractory period during which neurons are not allowed to fire, the duration of rectangular PSPs, the firing intensity (Equation 2.1), the STDP time window (Equation 2.5), and bias updates (Equation 2.6).

- *Injected inhibitory current (α/other)*. As described in Section 4.2, during training a strong inhibitory current is injected into some populations to prevent spiking (and to stop extrinsic plasticity changes in α-neurons).

Table 3.3: Fixed parameters of the SAM module

Parameter	**Value**
PSP amplitude	2 mV
Num. inhibitors	5
Exc. bias minimum	-30
Exc. bias maximum	5
Weight maximum	Various
Learning time	600 s (Task A), 300 s (Task B)
Synaptic delay	Various
Initial exc. bias mean	5
Initial exc. bias std.	0.1
Initial χ–α weight mean	1.333 (Task B), 3.0 (Task A)
Initial χ–α weight std.	0.1
α–inhibitors weight	80.0
α–ζ weight	20.0
Inhibitors–α weight	-7.0
Inh. neuron bias	-10
Time constant τ	15 ms
Injected inh. current (α)	-80 pA
Injected inh. current (other)	-30 pA
Injected exc. current (other)	30 pA
Sample presentation time T	100 ms
Activity collection time D_s	2 s (Task A), 20 s (Task B)

- *Injected excitatory current (other)* [SAM only]. A positive external current forces input neurons to fire during training.

- *Sample presentation time*, the duration (in biological time) for which each independent sample from the target distribution is presented to the network during the learning phase.

- *Activity collection time*. After training, network input is clamped (Task A; see Sections 4.1 and 4.2) or entirely removed (Task B). Spiking activity in the output layers (SAM) or excitatory pools (SPI) is then collected for this duration to reconstruct the estimated conditional (Task A) or joint (Task B) distributions.

Tables 3.3 and 3.4 give the fixed parameters of the SAM and SPI modules re-

Table 3.4: Fixed parameters of the SPI module

Parameter	**Value**
PSP amplitude	2 mV
Exc. bias minimum	-30
Exc. bias maximum	5
Learning time	300 s
Neurons per subpopulation	8 (Task B), 10 (Task A)
Minimum input synaptic delay	Various
Maximum input synaptic delay	Various
Fixed synaptic delay	Various
Initial exc. bias mean	5
Initial exc. bias std.	0.1
Initial P–P-type weight mean	Various
Initial P–P-type weight std.	0.1
Inh. neuron bias	-10
Time constant τ	10 ms
Injected inh. current	-20 pA
Sample presentation time T	100 ms
Activity collection time D_s	2 s (Task A), 20 s (Task B)

spectively. Some of these parameters are equally applicable to both architectures, with a few changes and additions. Note also that synaptic delay, while not optimisable by the outer-loop and fixed during any one run,[13] takes various values (as described in Section 4.2), in order to examine the effect of increasing delay. Similarly for the other parameters whose value is given as "various"; these values are specified later.

Variable encoding is highly dependent on the specific form of evolutionary algorithm. In the case of GAs, it has been found that very simple techniques can achieve highly competitive results when applied to neuroevolution problems. In Such et al. [49], for instance, a simple combination of selection by means of truncation and elitism, and mutation via addition of random normal noise—with no crossover operation at all—resulted in performances that rival those of popular reinforcement techniques in deep learning.

For this reason, it was decided to apply a relatively simple form of GA, de-

[13] Attempts to optimise delay gave too much variability in execution time of different individuals, and sometimes resulted in stability issues.

scribed generally in Section 2.8 and more specifically in Section 4.2. Aside from the operations described below, this GA processes bit string representations of individuals, requiring suitable conversion between floating type hyperparameters and binary strings. Preliminary tests using a straightforward conversion from binary floating-point format ruled this method out on the basis of poor convergence: this is expected because floating types are typically represented using 1 sign bit, 8 exponent bits and 23 fraction bits,[14] which has the consequence that flipping a bit in the exponent part of a representation has a starkly different effect than flipping a bit in the fraction part. Additionally, floating point values have higher resolution near 0. Assuming a common approach to both parts of the representation, as required by the GA technique described in Section 4.2, is therefore unsatisfactory.

The problem is avoided if floating type variables are represented by converting to a suitable integral value. This can be achieved by discretising the hyperparameter space using the ranges in Tables 3.1 and 3.2. Specifically, when required to perform bit operations on floating type hyperparameters, the floating values are first mapped to the nearest discrete step by the transformation

$$h_n = \text{round}\left(\text{UINT\_MAX} \cdot \frac{h_f - h_{\text{MIN}}}{h_{\text{MAX}} - h_{\text{MIN}}}\right), \tag{3.5}$$

where h_n is the integral representation, UINT_MAX is the maximum value representable by unsigned integers, h_f the original floating point variable, and h_{MIN} and h_{MAX} the minimum and maximum value that the hyperparameter can take. This transformation scales all hyperparameters into the range [0, UINT_MAX]. The bit string representation of h_n is then used in subsequent bit string operations. To recover the floating value of a bit string, the inverse of Equation 3.5 can be used.

This has the advantages of

1. Harmonising the effect of bit operations on all parts of the bit string and giving unpreferential coverage of the search space.

[14] Or 1, 11 and 52 bits respectively in the case of double floating types. Kahan's lecture on the IEEE standard 754 for floating point representations gives a good overview [19].

2. Reducing the explorable space to the region defined in Tables 3.1 and 3.2. This improves convergence if the optimal solution lies in that space.

3. Avoiding issues with spliced floating point binary representations that are meaningless (NaNs), which would require special treatment. Indeed, every bitwise recombination of two unsigned integers is also an unsigned integer.

4. Experiments and Evaluation

4.1 Families of Tasks

Pecevski & Maass [43] describes two different tasks, for each of which a model with hand-crafted weights is built and simulated. The current work does away with the hand-crafted design and separately optimises the architecture of the SAM module and the novel SPI architecture for best estimation performance on both tasks. The tasks themselves, however, have the same structure as the published examples in [43].

The tasks can be considered two specific instances of a generic density estimation problem, referred to as Task A and B below:

4.1.1 Task A

Task A is a conditional probability density estimation task with one dependent and two independent variables. All variables are binary. The target density can be written as $p(z|x^1, x^2)$. As in [43], since each SAM module estimates one conditional density, a single module suffices for the estimation. Similarly, only one SPI module is required. This is the easier of the two tasks, requiring fewer hyperparameters and therefore a smaller search space. Pecevski & Maass gives one specific instance of this density, its source joint distribution given in Table 4.1 and visualised in

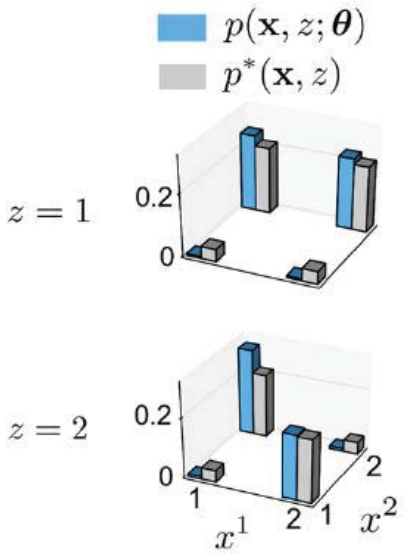

Figure 4.1: Target joint distribution (grey) and generative model approximation (cyan), for a z variable associated with two x variables. $p(z = 1, \mathbf{x})$ and $p(z = 2, \mathbf{x})$ are plotted separately for clarity. From [43].

Table 4.1: Sample target distribution for Task A

	$p^*(x^1, x^2, z = 1)$	$p^*(x^1, x^2, z = 2)$
$x^1 = 1, x^2 = 1$	0.04	0.04
$x^1 = 1, x^2 = 2$	0.21	0.21
$x^1 = 2, x^2 = 1$	0.04	0.21
$x^1 = 2, x^2 = 2$	0.21	0.04

Figure 4.1,[1] which it then uses to test SAM performance. Note that given a target joint distribution $p^*(\mathbf{x}, z)$, it is easy to calculate the conditional distribution $p^*(z|\mathbf{x})$ using basic manipulations:

$$p^*(z|\mathbf{x}) = \frac{p^*(\mathbf{x}, z)}{p^*(\mathbf{x})} = \frac{p^*(\mathbf{x}, z)}{\sum_z p^*(\mathbf{x}, z)} \tag{4.1}$$

Below, Section 4.2 will explain how the same distribution is also used in our evaluation, for continuity with [43].

[1] In being bimodal, the specific distribution given in Pecevski & Maass is not trivial. Indeed, the paper describes how bimodal mixtures are more difficult to learn than unimodal distributions, therefore requiring more neurons in the hidden layer.

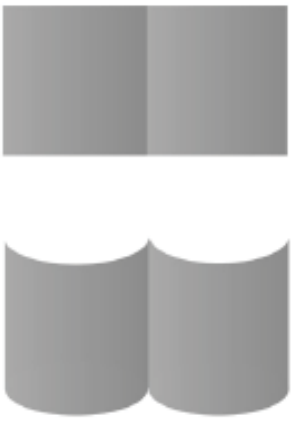

Figure 4.2: Stimuli from the visual task experiment. Both panels have identical shading profiles in the horizontal direction, but the top panel is perceived as two flat halves with the area immediately left of the centre line appearing to have a lower reflectance than the area immediately to the right. The bottom panel is perceived as two cylindrical surfaces with equal reflectance. The 3D shape effectively "explains away" the competing cause: different surface reflectances. Adapted from [43].

4.1.2 Task B

Task B is a joint probability density estimation task based on what Pecevski & Maass refers to as a "visual explaining away problem". The task models an experiment from the field of visual perception, first described in Knill & Kersten [24]. Figure 4.2 gives the two stimuli that set up the experiment. The problem can be modelled by a Bayesian network with four variables, where two variables y^1 and y^2, representing relative reflectance and 3D shape reflectively, are independent; y^3, shading, dependent on both y^1 and y^2; and y^4, contour, dependent solely on y^2. The full joint distribution can then be formally written as

$$p^*(y^1, y^2, y^3, y^4) \equiv p^*(y^1) \cdot p^*(y^2) \cdot p^*(y^3|y^1, y^2) \cdot p^*(y^4|y^2), \tag{4.2}$$

and represented by the Bayesian network shown in Figure 4.3. One SAM module—or one SPI module—can represent one variable's conditional dependence on other variables. Every conditional dependence needs to be accounted for. Pecevski & Maass [43] explains how for any variable y^i, the entire set of variables $\{y^j : j \neq i\}$

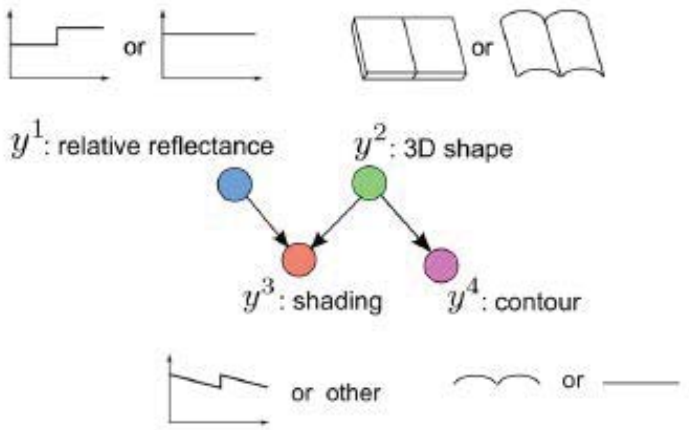

Figure 4.3: Bayesian network that models the visual explaining away problem. SAM and SPI modules can be combined recursively in a structure analogous to that of the Bayesian network in order to learn dependencies. From [43].

Table 4.2: Sample target probability distribution for Task B

	$p^*(y^3 = 2 \mid y^1 = 1, y^2)$	$p^*(y^3 = 2 \mid y^1 = 2, y^2)$	$p^*(y^4 \mid y^2)$
$y^2 = 1$	0.13	0.87	0.13
$y^2 = 2$	0.87	0.13	0.87

can be used as the dependency set of y^i; technically, a SAM module can in this case still approximate the target conditional dependencies, albeit inefficiently. Knowledge of a problem's Bayesian network representation gives us an advantage, however. In this case, each variable can be conditioned on its Markov blanket only, which proves far more efficient for learning purposes [43].[2] This gives a structure of network connections analogous to the Bayesian network, with additional dependencies from the Markov blankets as illustrated in Figure 4.4. Task B, then, requires a recursive combination of SAM or SPI modules. [43] uses a specific target joint probability distribution of this form, given in Table 4.2. Note that in this instance, $p^*(y^1) = p^*(y^2) = 0.5$. We will also use this specific target distribution in our evaluation.

Aside from the specific instances of the distributions from Task A and B, the current work will also generate target distributions of the same algebraic form,

[2]For any node in a Bayesian network, the Markov blanket is defined as: its parents, children, and the other parents of its children.

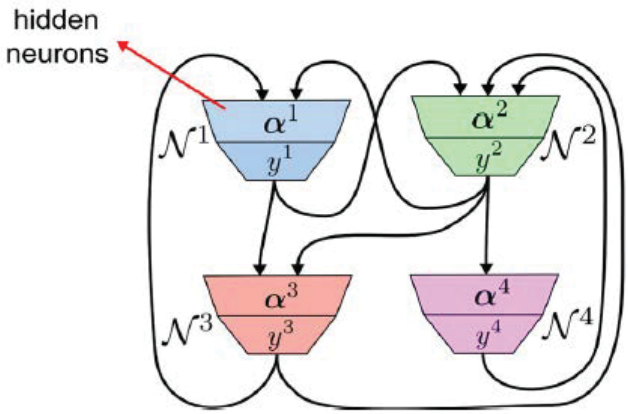

Figure 4.4: SAM or SPI modules need to be arranged recursively into this structure. Each module is fed input from the Markov blanket of the variable it represents. Note the structural similarity with the Bayesian network in Figure 4.3. Reproduced from [43].

consisting of mixtures of discrete uni-modal components with randomly drawn parameters. These will be used in generalisation tests.

4.2 Simulations

Step 4 in Section 3.2 is where core simulation and optimisation work occurs. The simulations are a suite of experiments that optimise the performance of our neural networks against Task A or B, using different evolutionary techniques as described in Section 2.8. The experiments are also concerned with evaluating the effect of increased synaptic delay on the quality of density estimation. All experiments, one way or another, evaluate some aspect of the use of these SNNs as generative or discriminative models, and can therefore be said to have the general form described in Algorithm 1. Evaluation is explained in depth in Section 4.3.

Simulations are implemented in interoperable Python and C++, extending third-party tool NEST [26]—a "highly scalable simulator for networks of point or few-compartment spiking neuron models". NEST is cited extensively by computational neuroscience publications, and provides in its core DLL fast CPU-based simulation of popular neuron and synapse models, easily extendible with third-

Algorithm 1 Simulation Overview

$\Theta \leftarrow$ initial random configurations
performances $\leftarrow \emptyset$
while end condition not met **do**
  F $\leftarrow \emptyset$
  for $\theta \in \Theta$ **do**
    Either, evaluation type 1: discriminative performance p_θ
    Or, evaluation type 2: generative performance p_θ
    performances $\leftarrow$ *performances* $\cup \{p_\theta\}$
    Calculate fitness $f(\theta) = f(p_\theta)$
    F $\leftarrow$ F $\cup\{f(\theta)\}$
  $\Theta \leftarrow$ Evolve(Θ, F)

party models and permitting the use of distributed computing resources for reduced execution time. It also comes with Python bindings as PyNEST, which allows the researcher to build and manipulate networks and to connect signal generators and voltmeters in a straightforward manner mimicking electrophysiological experiments.

The neuron and synapse models used in this thesis required extension of core NEST functionality: the stochastic integrate-and-fire neuron with α-shaped PSP and intrinsic plasticity (Section 2.3), and the tractable variant of STDP synapses (Section 2.4). Both SAM and SPI architectures, moreover, were crafted in Python as a modular layer on top of PyNEST functionality extended with the newly added models in core NEST. The experiments, evolutionary algorithms, and evaluation techniques were extensions to a learning-to-learn package called LTL used internally by the Graz research team, which handles the optimisation loop and data management. Figure 4.5 illustrates the nested architecture of the implementation.

Except where noted, all simulations are run with a temporal resolution of 0.1 ms. EA specifics and experiments are described below in separate subsections.

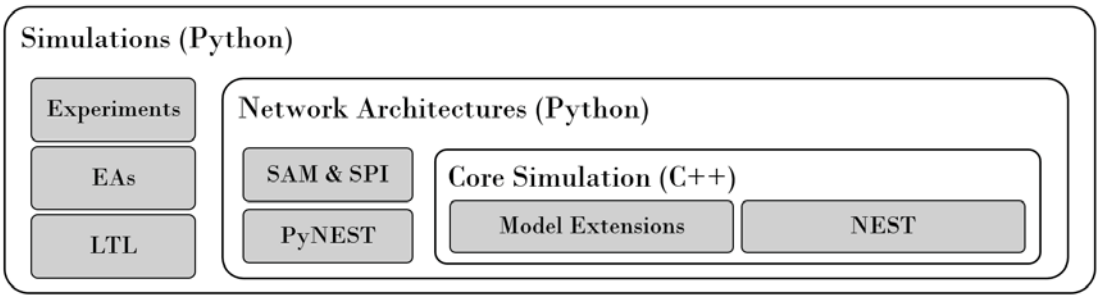

Figure 4.5: Schematic of the simulation architecture. NEST, PyNEST and LTL are third-party tools or packages; the other units are extensions implemented by the author.

4.2.1 Evolutionary algorithm specifics

This work applies two evolutionary algorithms separately for comparative purposes: genetic algorithms (GA) and natural evolutionary strategies (NES).

Specifically, GA applies all three operators (unlike [49], which skips recombination), described below:

- *Tournament selection.* From a *population size* of 200 individuals, a subset of 20 individuals is selected randomly for a tournament, from each of which the single fittest individual is picked. Tournament selection is repeated until all parents for crossover are chosen.

- *Two-point crossover.* Selected individuals are split mechanically into two halves, and every subsequent pair from the respective halves is mated. During mating, there is a *crossover probability* of 0.5. If crossover does occur, the chromosomes of the two parents are spliced together as described in Section 2.8, with two randomly chosen bits between which the code of one parent is swapped with that from the second parent, and vice-versa. Otherwise, the parents are cloned unmodified as offspring.

- *Bit-flip mutation.* Every bit in the bit-string representation of the individual is independently flipped with a *mutation probability* of 0.05. For the specific

combination of SAM & Task A, a mutation probability of 0.01 applies.[3]

The three operators are applied over 20 *generations*, giving a total of 4000 individuals per run.

NES as proposed by Wierstra et al. [54] has the advantage of requiring very few parameters. It is mainly self-administering. It does, however, require the *learning rate* η and *population size* λ along with the *number of generations*. Aside from these parameters, it also requires the initial *search distribution parameters*—$\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$—which in our case are simply set to the vector of midpoints of the hyperparameter ranges, and half of the ranges respectively (see Section 3.4 for the tables).[4]

The specific NES parameters are listed in the experimental results section, because they vary between experiments. Two points, however, influenced the choice of parameters:

1. Since NES exploits the fitness of distributions in parameter space, rather than specific individuals, it is intrinsically better suited for stochastic measurements of fitness than other optimisation techniques (including GA). Thus, motivated also by resource limitations, this obviated the need for repeated fitness measurements to give more efficient optimisation runs. The flip side to this is that although NES can very well handle noisy individual measurements and still manage to improve distribution fitness over generations, individual solutions *per se* are still subject to noise. After optimisation runs, therefore, to present a fairer comparison with the results from GA (which, in contrast, use repeated measurements), the best mean generation fitness is also reported. The mean fitness averages out stochastic effects and is closer to what one expects of individuals sampled from close to the best individual.

2. NES takes relatively longer to converge.[5] This would have eliminated NES

[3]The lower mutation rate gave better convergence during preliminary runs.

[4]We assume a multinormal distribution.

[5]This was confirmed in preliminary runs with small learning rates. Multidimensional benchmarks and neuroevolution applications in the literature routinely report thousands or tens of thousands of iterations [50, 54].

on financial and practical grounds, but high learning rates between 0.5 and 1.0 proved useful here.

These parameters and operator choices were decided upon after preliminary testing.[6] Among other GA operators tested were *one-point crossover* and *random crossover*, also described in Section 2.8. Mutation probabilities ranged between 0.001 and 0.05 during testing. However, it must be noted that since evaluation requires simulation of complex dynamics, it is very compute intensive—demanding up to several minutes for each individual—and owing to time restrictions resulted in limited exploration of different EA parameter combinations. Population and generation sizes are similarly limited because of experiment execution time.

4.2.2 Baseline experiment

In the baseline experiment, the sample distributions from Pecevski & Maass [43]—Tables 4.1 and 4.2—are used as target densities in Tasks A and B, respectively. Only the SAM architecture from the publication is optimised, but both GA and NES are separately applied for comparative reasons. The purpose of this experiment is to measure the optimal KL divergences that can be achieved at different synaptic delays and therefore, to deduce the effect of incrementally increased delay on performance.[7] A second purpose of the experiment is to compare these optimal performances with the performances reported in [43]. Keeping in mind that the models in Pecevski & Maass were hand-crafted, as opposed to optimised by evolutionary algorithm, it is possible that even with non-zero delay SAM model performance exceeds that reported in [43]. A third, minor, motivating factor is to understand whether optimal hyperparameter values differ greatly between models optimised at different synaptic delays. In fulfilling these purposes, this experiment tackles the third objective given in Section 3.1.

Unfortunately, NEST does not allow 0-delay synapses, which precludes perfect

[6] And consultation of general heuristics in the field.

[7] For consistency with existing research we shall use KL divergence as the main metric.

replication of the experiment in Pecevski & Maass. With its ideal connectivity, Pecevski & Maass achieves a performance in line with theoretical assumptions (see Section 2.5).[8] However, NEST does allow a user-specified non-zero delay, irrespective of how small it is.

For the purposes of Task A, we measure the performance of a single SAM module at reconstructing the conditional target distribution from which samples are drawn. Each evaluation consists of generating a SAM module with the specified parameters, running the simulation for a fixed biological period, measuring the module's learned distribution from its activity, and calculating the KL divergence between the estimated and target distribution.

Algorithm 2 gives the general evaluation sequence in pseudocode, which also applies to Task A.[9] During network generation, α-neurons are initialised with a bias drawn randomly from a normal distribution with the parameters given in Table 3.4.[10] Training samples $\langle x^1, x^2, z\rangle$ from the target distribution are exposed to the network via a population code as described in Section 2.5: specifically, input neurons that do not encode for any value of x^1 or x^2 are artificially inhibited by a negative external current (given in Table 3.4), along with α-neurons that do not code for the right value of z. Each sample is presented for a short fixed period of time (100 ms, in line with Pecevski & Maass), during which plasticity is allowed to proceed in accordance with Equations 2.7, 3.1 and 3.3. Maximum weight allowable is 5.0. During the training period, which is reduced from 1200 seconds in [43] to 600 seconds for practical reasons, the synaptic plasticity learning rate changes linearly from the initial to the final STDP rate (these are hyperparameters; see Section 3.4), reaching the final rate halfway through the training period (at 300 s), after which it is halted. Intrinsic plasticity (bias) rate, on the other hand, stays fixed at the first rate until the halfway mark, at which point it switches to the second

[8] More on this later, too.

[9] In the case of Task A, we measure the discriminative performance of the network.

[10] Different runs can therefore generate networks with slightly different connectivity, even if the hyperparameters are identical.

rate.[11] At the end of the full training period, all plasticity is halted and, for every possible combination of inputs $\langle x^1, x^2 \rangle$, ζ-layer activity is collected while the input neurons are clamped to the current that encodes the input combination. At this point, activity is freely allowed in the α-layer without external inhibition—WTA-like competition arises out of the structure of the network, whenever an α-neuron fires. The estimated distribution is then reconstructed from the collected activity and used to estimate the network's performance p_θ. This evaluation sequence is more or less common to all experiments, with some variations that are noted.

Algorithm 2 Evaluation

```
Generate network N from configuration θ
Assign target distribution p* externally
X ← all possible input combinations
t ← 0
while t < learning time T do
    s ← ⟨x, z⟩ sampled from p*
    Reset currents with population code of s
    Simulate for sample presentation time Δt
    Update learning rates
    t ← current simulation time
Halt plasticity
if evaluation type discriminative then
    for combination x of input values in X do
        Reset currents with population code of input x
        Collect spiking activity over duration D_s
else if evaluation type generative then
    Reset currents to 0
    Collect spiking activity over duration D_s
Calculate estimated distribution p from activity
p_θ ← D_KL(p||p*)
```

Task B requires the careful creation of interconnections between modules as shown in Figure 4.4, each of which models the dependency of one variable on its Markov blanket. The maximum allowable weights depend on the module, with α synapses in modules encoding variables y^1, y^2 and y^3 permitted to reach a value of 4, while those encoding variable y^4 ranging up to 2. These choices were made

[11] These idiosyncrasies of learning follow from Pecevski & Maass.

to balance input, and also follow from [43]. The simulation proceeds in a fairly similar way to that of Task A above, and thus follows the outline in Algorithm 2—the major difference being that this experiment measures the *generative* performance of the network, rather than discriminative performance. For this purpose, the network's spontaneous activity is allowed to proceed uninhibited without external forcing currents for a relatively long duration D_s (20000 ms), and the joint distribution, not the conditional distribution, is measured and compared with the target. Another difference is that the second half of the training period—during which only intrinsic plasticity applies in Task A—is discarded, reducing the total run to 300 s of biological time in Task B. Note, however, that as shown in Chapter 5, this has negligible effect on results, because the strongest adjustments occur early during training.

Since both tasks are stochastic in nature, every individual is evaluated 10 times (Task A) or 5 times (Task B)[12] and the mean KL divergence assumed to be the final fitness. The synaptic delays tested in this experiment are in logarithmic increments ranging between 0.1 ms and 2.0 ms, for both Tasks A and B.[13] For the GA run, there are two tasks, five delay values, 10 (or 5, in case of Task B) evaluations per individual, and 4000 individuals per optimisation run, giving a total of 300000 evaluations. By contrast, NES required fewer evaluations per run (see Section 5.1 for the experimental NES parameters), with a total of 76800 evaluations. This experiment runs up to a grand total of 376800 evaluations. Given that on a modern cloud CPU core Task A evaluations run for about 30-40 seconds each and that Task B evaluations run up to about 2.5 minutes each, this gives a total serial execution time of about 8500 hours, or 355 days.[14] This excludes overhead.

As far as allowable by the simulator, fixed parameters are preserved from Pecevski & Maass [43] in order to reconstruct the original density estimation algo-

[12]The number of trials are reduced in Task B owing to the increased duration of simulation runs of that task.

[13]There are five delay values.

[14]Parallel computing rescued this infeasible outcome.

rithm in the inner-loop in both tasks.

4.2.3 Generalisability experiment

This experiment tests the limits of generalisability of both architectures, SAM and SPI. The sample distributions from Pecevski & Maass [43] are done away with and replaced by randomly parametrised distributions of the same form. For this purpose, a real number is drawn randomly from $[0, 1]$ for every value combination,[15] and then normalised by the sum of all draws. Instead of repeated evaluations of one individual on the target distribution (as in the baseline experiment above), the individual is evaluated on the same number of differently parametrised distributions. Note, however, that although randomly parametrised, the distributions are preserved between individuals, so that each individual can be tested on the same targets.

Specifically, the prime purpose of this experiment is to test whether SAM and SPI architectures can suitably estimate multiple distributions, in both Task A and B variants, assuming one set of parameters but separate training runs on each distribution. A secondary purpose is to understand which of the architectures, after optimisation through evolution, generalises better to unseen distributions. A third purpose is to test whether SAM or SPI networks optimised only on the sample distributions from Pecevski & Maass (given in Tables 4.1 and 4.2) achieve worse performance—as expected—once they are tested on unseen distributions, compared to networks that have been trained on multiple distributions. In evaluating SPI, aside from SAM, this group of experiments fulfils the first two objectives given in Section 3.1. First, it solves the optimisation problem (both SAM and SPI); second, it pushes biological plausibility by testing the SPI architecture explained in Section 3.3 and evaluating to what extent, if any, these changes affect the performance of the network when compared to SAM.

Algorithms 1 & 2 apply. The experimental outline described in Section 4.2.2

[15] With four binary variables, for example, there are 16 combinations.

is also still applicable for both tasks, except that the target distributions are now randomly parametrised. In addition, however, the generalisability test requires an additional process: once the optimal parameters for both SAM and SPI are established, each optimal configuration is tested on a set of unseen distributions to determine generalisability. Algorithm 3 illustrates the optimisation and generalisability test in pseudocode, collecting the generalisation performance as the metric $\hat{p}_{\theta_m}$: the arithmetic average of the KL divergences on the unseen distributions.[16]

Algorithm 3 Optimisation & Generalisability

Run Algorithm 1, assuming n random distributions
Optimal $\theta_m \leftarrow \theta \in \Theta$ s.t. $f(\theta) = \max(F)$
$P_{\theta_m} \leftarrow \emptyset$
for i $\in \{1, 2, \ldots, n\}$ **do**
    Generate unseen distribution p^*
    $p_{\theta_m} \leftarrow$ Evaluate(θ_m, p^*) (Algorithm 2)
    $P_{\theta_m} \leftarrow P_{\theta_m} \cup \{p_{\theta_m}\}$
$\hat{p}_{\theta_m} \leftarrow \text{avg}(P_{\theta_m})$

For this experiment, we use the following fixed parameters:[17]

- *Synaptic delay* [SAM] & *fixed synaptic delay* [SPI]—0.2 ms
- *Weight maximum* [SAM]—5 (Task A); 4, 4, 4, and 2 for modules encoding variables y^1, y^2, y^3 and y^4 respectively (as in Section 4.2.2)
- *Minimum input synaptic delay* & *maximum input synaptic delay* [SPI]—0.1 ms & 0.3 ms. Recall that SPI networks have a uniformly distributed input (P–P-type connection) synaptic delay value. Thus, the expected delay value is midway between extremes, which makes 0.1 and 0.3 ms a fair choice for comparison with SAM architectures with a fixed delay of 0.2 ms.
- *Initial P–P-type weight mean* [SPI]—w_{MAX}, varying between individuals

[16] Note that this procedure is equivalent to having a training set of n distributions and a test set of another n distributions.
[17] Which are unspecified in Tables 3.3 and 3.4.

All SPI evaluations used a training period of 300 s. There are two evolutionary techniques, two architectures, two tasks, 10 evaluations per individual (Task A) or 5 (Task B). GA had 4000 individuals per run, and the NES parameters varied between tasks and architectures (see Section 5.2 for details). Number of evaluations during optimisation runs up to 427200, and another 60 evaluations for the generalisability tests. Testing SAM and SPI networks trained only on the sample distributions from Pecevski & Maass [43] for generalisability requires a few more evaluations. In all, the serial execution time ignoring overhead runs up to about the same number as in the baseline experiment above, 8500 hours or ∼355 days.

4.2.4 Sensitivity to perturbations

A minor objective of this work is to derive the *sensitivity* of the optimal configuration to perturbations in its parameters. A quick and dirty way to achieve this without running further experiments is to use the results from NES runs. NES iteratively updates a search distribution $\pi(\boldsymbol{z}|\theta)$, as described in Section 2.8. The *mean* $\boldsymbol{\mu}$ of π—assuming a multinormal distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$—indicates the optimal center according to NES, and is therefore of relevance mainly to the other experiments above. The *covariance matrix* $\boldsymbol{\Sigma}$, on the other hand, is of key importance here: it quantifies the search spread within parameter space. A low-variance spread in some parameter direction around the optimum indicates a parameter sensitive to small perturbations, whereas high variance indicates low sensitivity. Additionally, covariance between different parameters can be useful to expose search distributions that are not parallel to variable axes, but the variant of NES used in this thesis retains only the diagonal elements of the covariance matrix. Thus, only the variances of the multinormal search distribution can be used.

4.2.5 Network state interpretation

Section 2.5 describes some of the theoretical considerations behind network state interpretation. For the purpose of mapping network states onto variable values, we need to distinguish between interpreting a network's activity for its discriminative performance (Task A) on the one hand, and interpreting a network's activity for its generative performance (Task B) on the other hand.

It is straightforward to determine a single module's discriminative performance: we follow the example of Pecevski & Maass, which clamps the network inputs to the correct combination of values and simply counts output spikes in the output layer (SAM) or excitatory subpopulations (SPI). The relative frequency of firing in the neurons or subpopulations coding for different values directly determines the relative probability of those different output values, conditioned on the specific input. This is done exhaustively for all the input combinations (e.g. in the case of two binary predictors, there will be 4 input combinations).

With generative performance, however, the task is less straightforward. In this case, several modules are connected together (as explained in Section 4.1) and the spike counting method above cannot be used. This is because the input to any module can change from one instant to another, as determined by the activity of all the other modules. One must, therefore, determine the network state at regular intervals by splitting the activity into short segments of time and extracting the instantaneous network state from the resulting windows of activity.

The assumption in Pecevski et. al., upon which this work builds, is that in the SAM architecture variables are encoded using population coding in such a way that each neuron in the input layer corresponds to exactly one value of one variable. Moreover, the design of the network—with WTA inhibition built into the hidden layer and the refractory period τ coinciding with the duration of PSP effects—implies that in the ideal scenario explored in [43], only one neuron in the hidden layer fires at any one time, and the next neuron to fire is forced to wait for at least τ ms. This ensures that in the output layer only one neuron can fire within

the time window, which is very convenient since it preserves the simple population code in the output layer, in addition to the input layer.

Practically speaking, however, non-zero synaptic delay introduces a complication in the SAM architecture: as shall be described in Section 6.1, finite delay allows multiple neurons to fire within a time window τ, violating the assumption of a clean population code in the output layer that conforms to the at-most-one behaviour of the input layer. Although these states may be thought of as "invalid" within the specific paradigm of population coding explored in Pecevski & Maass, these are likely to be found in biological neural circuitry, where synaptic delay and their connective structure prevent ideal, WTA-like competition. As has been shown through experiment, it is probable that areas downstream from sensory regions of the brain can react to certain stimuli after only a single spike is received [6], which implies that little information, if any, is ignored. Although two spikes from different regions may arrive within close proximity, their relative timing may indicate relative degrees of activation in the upstream areas and is therefore informative: regions that are strongly stimulated tend to activate quicker than weakly stimulated regions. An option, therefore, is to assume that in the case of multiple spikes from competing neurons the *first* neuron to fire is the relevant one. Nevertheless, this permits another category of state that does not arise under theoretical conditions: the *zero state*, where none of the neurons representing a variable fires for a long duration. Valid and zero states alike need to be captured for analysis.

In our case of SAM with non-zero delay, network states are interpreted in this manner: the spiking activity of the module's output layer is split into consecutive time windows of duration τ. If only one spike happened during some time window, the value of that module's variable z is assigned to the value represented by the neuron that spiked for the duration τ—in effect, neuron ζ^k sets variable $z = k$ for τ ms when it spikes. If no output neuron spiked, that state is considered a "zero state", and if more than one neuron spiked, only the first one to do so is

retained.[18] See Algorithm 4. After iterating through all modules and all time windows, the dictionary S holds the state counts for all variable combinations that occurred during the collection run. The estimated distribution can then simply be reconstructed by normalising the counts by the total count of time windows.

Algorithm 4 State interpretation - SAM

$t \leftarrow 0$
Zero state count $s_0 \leftarrow 0$
Dictionary of state:count pairs $S \leftarrow \{\}$
for spiking activity in $[t, t+\tau]$ **do**
  for variable y^i in set of variables **do**
    $N^i \leftarrow$ neurons that encode for y^i
    if none of N^i spiked **then**
      $s_0 \leftarrow s_0 + 1$
      $y \leftarrow 0$
    else
      $k \leftarrow$ index s.t. N^i_k is the first neuron to spike
      $y^i \leftarrow k$
  Build tuple s of variable values $\{y^j : j \text{ in set of indices}\}$
  $S[s] \leftarrow S[s] + 1$
  $t \leftarrow t + \tau$
Normalise each value in S by $|S|$

Unless specified, wherever reference is made to "performance" or "fitness", it specifically means the average (negative) KL divergence from the estimate $p(\mathbf{y}; \theta)$ to the target $p^*(\mathbf{y})$,[19] i.e. $D_{KL}(p^*||p)$, where the estimate distribution also includes the zero states referred to above. These zero states do not exist in the target distribution, hence reducing performance whenever present in the estimate. Pecevski & Maass and other publications, however, often ignore zero states by simply counting firing rates or using temporal kernels to smooth activity over time, effectively removing the possibility of zero states—it is fairer, then, to use the estimated distribution with its zero states truncated and renormalised in certain cases. The KL divergence from this truncated and renormalised estimate to the target shall be re-

[18] In the marginal case where multiple neurons spike at the same instant—artificially possible because of discretisation in the simulation—the choice is made arbitrarily.

[19] Or their conditional counterparts.

ferred to as *valid-state divergence*. If we need to distinguish this from performance as originally stated, we shall refer to the latter as *whole-state divergence*.

The SPI architecture introduces stronger deviations from the population code seen in [43]. WTA-like competition is replaced by "soft" inhibition as reported in Jonke et al. [18] and Legenstein et al. [29]. In this scenario, it is not sensible to interpret states with the rigid formalism used with SAM above. In particular, because of the larger population count and softer inhibition, multiple neurons are expected to spike within close proximity of each other; moreover, neurons from *different* excitatory subpopulations can be simultaneously active (Figure 3.2 gives a raster plot of activity which illustrates these dynamics). Instead, we shall interpret the network activity by smoothing spike activity and setting variables to the index of the subpopulation with *strongest activity*, i.e. the subpopulation with the highest smoothed firing rate. It is noteworthy that Pecevski & Maass interprets network states similarly when dealing with spontaneous activity in the visual perception example (Task B in Section 4.1); specifically, it uses smoothed firing rates in place of the variable assignment scheme described previously. It achieves this by first convolving the discrete spike trains coding for each value with an α-shaped kernel, $t/\tau \exp(-t/\tau)$, and then using the resulting trace to determine the instantaneous network state. As in Pecevski & Maass, we shall use a rise/decay constant $\tau = 100$ ms. This is outlined in Algorithm 5.

Note that zero states are highly unlikely, under this interpretation, and can for all intents and purposes be ignored.[20] Invalid states are likewise highly unlikely, happening only in the remote possibility that two spike trains coding for different values of the same variable have a long sequence of identical activity stretching over hundreds of milliseconds. Thus, they can also be safely ignored.

Table 4.3 summarises the different interpretation methods for clarity and convenience.

[20] They are only possible at the very beginning before the first spike occurs, or during prolonged (longer than a second) periods with no activity.

Algorithm 5 State interpretation - SPI

Split activity into variable-value spike trains S_j^i
$T_j^i \leftarrow$ result of convolving S_j^i with $t/\tau \exp(-t/\tau)$
$t \leftarrow 0$
Zero state count $s_0 \leftarrow 0$
Dictionary of state:count pairs $S \leftarrow \{\}$
while $t < t_f$ **do**
  for variable y^i in set of variables **do**
    $k \leftarrow$ index s.t. $T_k^i(t)$ has highest activity
    $y^i \leftarrow k$
  Build tuple s of variable values $\{y^j : j$ in set of indices$\}$
  $S[s] \leftarrow S[s] + 1$
  $t \leftarrow t + \tau$
Normalise each value in S by $|S|$

Table 4.3: Network interpretation methods

	SAM	SPI
Task A	spike counting	spike counting
Task B	segmentation (Alg. 4)	smoothing & segmentation (Alg. 5)

4.3 Evaluation

This work has multiple evaluative strategies, depending on the objective being assessed. The metric at the heart of all evaluations, however, is the KL divergence (see Section 2.6), used as fitness for the purpose of optimisation:[21]

- At a fundamental level, the work is successful if optimisation of SAM configurations demonstrates that learning performance can be improved over the baseline reported in Pecevski & Maass [43] on the density estimation tasks described therein (Tasks A & B), when the networks are only trained on the sample distributions. This requires running the simulations described in Section 4.2.2. Final performance at the end of the simulated period is of key relevance.

- At a higher level, the work is successful if it can demonstrate generalisability over a wider range of probabilistic inference or generative modelling tasks

[21] Technically, optimisation *maximises* fitness, which is defined as *negative* divergence.

(Section 4.2.3), either by finding a common configuration that performs well on different distributions, or by discovering common principles that circumscribe ideal architectures. For instance it might be possible, using a suitably scaled-up neural network, to approximate a wide range of simple probability distributions. Then this architecture could be perceived as a basic generative building block in the neural apparatus. Furthermore, it would make experimentally testable predictions about the existence of such building blocks. Success can be evaluated in terms of performance on a range of inference tasks (Task A) carried out by the common architecture, or by a recursive combination of modules (Task B) as in the visual perception experiment in Pecevski & Maass. In this case, networks are trained on multiple target distributions, and then tested on another set of unseen distributions. Generalisability is evaluated in terms of mean final KL divergence on the unseen distributions. Another assessment related to generalisability depends on describing the effect of applying the neural configurations optimised only for performance on the sample distributions to unseen distributions; it is expected that they perform poorly compared to the models optimised for general performance.

- The work is also successful if it can integrate the realistic biological properties discussed in Section 3.3. The architecture proposed for this purpose is the SPI module. Biological plausibility is relevant because it demonstrates that in principle the brain can implement probabilistic inference and generative modelling in neural microcircuitry in the manner described.[22] This can be evaluated by considering the performance impact of introducing these biological phenomena, in terms of final KL divergence and speed of convergence, keeping in mind that a small negative impact does not necessarily translate into failure.

- Short-delay configurations are relevant to the field of hardware. Recently,

[22]Whether it actually does can be experimentally determined.

neuromorphic implementations of SNNs have started to pick up (e.g. SpiNNaker [10] and IBM's TrueNorth [37], with research supported by DARPA [9]). Unlike biological circuitry, these implementations require non-zero but small synaptic delays, on the order of microseconds. Thus, good performance at low synaptic delays is another success criterion.

- A final success criterion is achieved if we can successfully deduce and describe any overarching trends in performance as problem parameters change. This is especially important with respect to the effect of synaptic delay. It is possible, for instance, that performance degrades more gracefully as synaptic delay is increased in the case of SPI, as opposed to SAM. Demonstrating this would have biological implications.

It is possible to undertake some work in the direction of the second LTL directive (step 6 in Section 3.2). Since the simulation can be repeated for several different combinations of task parameters and configurations, it may provide an opportunity to spot commonalities or trends. An interesting possibility, for instance, is that some optimal hyperparameters remain invariant to problem structure or size, or that hyperparameters and problem parameters (such as delay) covary at known rates.

4.3.1 Datasets

All evaluation is against synthetic target distributions, with parameters generated randomly (see Section 4.2.3) or matched with those found in Pecevski & Maass (Section 4.2.2). The procedure for generating these distributions is detailed in the respective experiment sections (see Section 4.1), with key target distributions from Pecevski & Maass documented in Tables 4.1 and 4.2.

5. Results

This section follows the outline of experiments described in Section 4.2, with a view to satisfying the evaluative criteria listed in Section 4.3.

5.1 Baseline Experiment

Zero transmission delay is necessary in some neural sampling scenarios [42], with several published theoretical models built on this assumption. Although it is not explicitly stated, the paper upon which this work is based makes this assumption as well (see Section 2.5 and Section 4.2.5). It is therefore an additional challenge to replicate or improve on the results from Pecevski & Maass [43] without the assumption of zero delay. This is achieved with some limitations discussed below.

5.1.1 Task A

Task A uses a single SAM module as inner-loop optimiser to learn the probabilistic dependence of one binary variable on two other binary variables. The single target probability distribution learned by the module is given in Table 4.1.

Optimisable hyperparameters and most fixed parameters are given in Section 3.4. Experiment-specific fixed parameters are given in Section 4.2.2. Most parameters of the algorithm are matched as closely as possible to the parameters in Pecevski & Maass, with a notable exception being total learning time, reduced

Table 5.1: Optimal performances: baseline, Task A

	Delay (ms)					
	0.05	0.1	0.2	0.5	1.0	2.0
GA	**0.0032**	0.0044	0.0168	0.0219	0.0203	0.0259
NES^a	0.0010	**0.0009**	0.0017	0.0015	0.0029	0.0056
NES^b	0.0045	**0.0044**	0.0103	0.0095	0.0203	0.0274

from 1200 seconds to 600 seconds,[1] and delay, which is set to various values. PSPs are additive, with a rectangular profile having a width of τ=15 ms.

Optimised via GA runs of 20 generations with 200 individuals each, and NES runs with a population size of 192, a learning rate of 1.0, and 80 generations (the rest of the parameters are listed in Section 4.2.1), the best performances at each of the delay values are given in Table 5.1. Note that performance is in terms of mean KL divergence of 10 repeated evaluations against the same sample distribution found in Pecevski & Maass [43]; that is, we are comparing the estimated conditional density $p(z|x^1, x^2; \theta)$ with the target density $p^*(z|x^1, x^2)$. Since KL divergence measures "mismatch" between target and estimated distributions, smaller KL values indicate better performance (see Section 4.2.5 for details on performance-related nomenclature). In the table, bold numbers mark the best performances of each optimisation algorithm. Also note that in addition to the list of five delay values listed in Section 4.2.2, another delay value of 0.05 ms is tested for comparative purposes.[2] Finally, note that NES^a reports the best *individual* fitness at the end of the run, while NES^b is the best *mean* fitness—the mean fitness of the best generation. For comparison with GA performance, NES^b is a fairer value, as explained in Section 4.2.1.

The slight reduction in GA-optimal performance from 0.5 ms to 1.0 ms notwithstanding, there is an overall negative relationship between delay and performance for both GA and NES results: as expected, longer delay introduces increasing

[1] This follows the fact that according to [43] most learning is done by 300 s.

[2] A simulation time step of 0.05 ms was used for this delay value only. Temporal resolution can affect the accuracy of spike timing, and therefore, by extension, final performance. However, differences are typically small, all else being equal.

Table 5.2: Optimal hyperparameters: baseline, Task A, GA

	Delay (ms)					
	0.05	0.1	0.2	0.5	1.0	2.0
b_-	-5.9868	-6.1431	-30.4391	-11.8871	-11.9322	-9.3895
w_-	-3.0747	-3.0748	-2.4757	-2.4491	-2.4541	-2.4784
T	0.4857	0.4236	0.7461	0.4867	0.4949	0.4994
R	0.5657	0.5872	0.1658	0.0746	0.0765	0.5902
η_0'	0.0397	0.0428	0.0294	0.0050	0.0299	0.0300
η_1'	0.0890	0.0445	0.0873	0.0478	0.0283	0.0286
η_0	0.0063	0.0088	0.0095	0.0086	0.0085	0.0086
η_1	0.0006	0.0081	0.0092	0.0048	0.0022	0.0050
c_1	0.7307	0.7015	0.1657	0.5308	0.1384	0.0466
c_2	1.4933	1.4872	0.8302	1.0102	1.1139	0.9504

deviation from ideal WTA behaviour, which tends to reduce performance. This confirms the relationship with respect to discriminative inference in single SAM modules. It is also evident that GA and NES^b performances are comparable, with a slight advantage for NES at the middle delay values (0.2 ms & 0.5 ms). The overall performances, especially at low delay settings, are excellent.

The best solutions corresponding to the performances in Table 5.1 are given in Tables 5.2 and 5.3. Note that in the case of GA optima, most parameters vary by significant amounts between delay settings, but that weight baseline w_- is relatively stable at around -2.4 to -2.5 at higher delays. This is substantially higher than the figure used in Pecevski & Maass (-4.02). So is T at around 0.5, which is just lower than the value used in Pecevski & Maass (0.58). Initial STDP rate is the stablest among all parameters, mainly varying between 0.008 and 0.01 at different delay settings. Pecevski & Maass used an initial learning rate of 0.002, lower than the optimal values achieved here. Note also that some optimal hyperparameters at 0.05 ms and 0.1 ms delay are nearly identical, especially w_-, R, c_1 and c_2.

NES optimisation gave few stable parameters across delay values, with the exception of first bias rate and initial STDP rates, η_0' and η_0 respectively, which were both fixed to extremal values in their respective ranges (see Section 3.4); the scaling parameter c_2 and weight baseline w_- were to some extent stable, as well.

Table 5.3: Optimal hyperparameters: baseline, Task A, NES

	Delay (ms)					
	0.05	0.1	0.2	0.5	1.0	2.0
b_-	-14.9634	-16.7923	0.0000	0.0000	-19.7628	0.0000
w_-	-3.7650	-4.5163	-3.5971	-5.3062	-3.7558	-3.7537
T	0.6049	0.4660	0.6320	0.3417	0.5022	0.4110
R	0.4518	0.9585	1.0000	0.2752	0.8140	0.5354
η'_0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
η'_1	0.0499	0.0691	0.0000	0.0989	0.0351	0.1000
η_0	0.0100	0.0100	0.0100	0.0100	0.0100	0.0100
η_1	0.0000	0.0000	0.0048	0.0000	0.0100	0.0091
c_1	0.5418	0.5727	0.2168	0.2387	0.1549	0.1635
c_2	0.8120	0.7079	0.7337	0.6431	0.6209	0.5793

However, some values such as the bias baseline b_- along with the second bias rate η'_1 and final STDP rate η_1 oscillate between extremal possibilities, and indicate that these parameters may be less relevant to fitness. This will be examined in the sensitivity analysis (Section 5.3). Note that the optimal hyperparameters found by NES do not coincide with GA hyperparameters. Of particular interest is the relationship of the scaling coefficient c_1 with synaptic delay: both GA and NES show a generally decreasing trend. This will be examined again in Section 6.1.

KL divergence values collected from an arbitrary training run of the GA-optimal individual (assuming a delay of 0.1 ms) are plotted in Figure 5.1 against time in s. The process of collecting spike activity for the purpose of density reconstruction is subject to stochasticity, aside from the stochasticity in the learning process itself, which varies from one run to another. This justifies the use of multiple measurements at the end of the process as final loss value.

Figure 5.1 illustrates the evolution of three KL divergence values during training. The orange and blue traces were calculated from the analytic expression in Equation 2.9;[3] the green trace is derived from experimental collection of output spike activity when the input neurons are clamped as outlined in Section 4.2.2, measured at regular intervals of 100 s. Experimental divergence was used as fit-

[3]The analytic conditional distribution is a simple application of Equation 4.1.

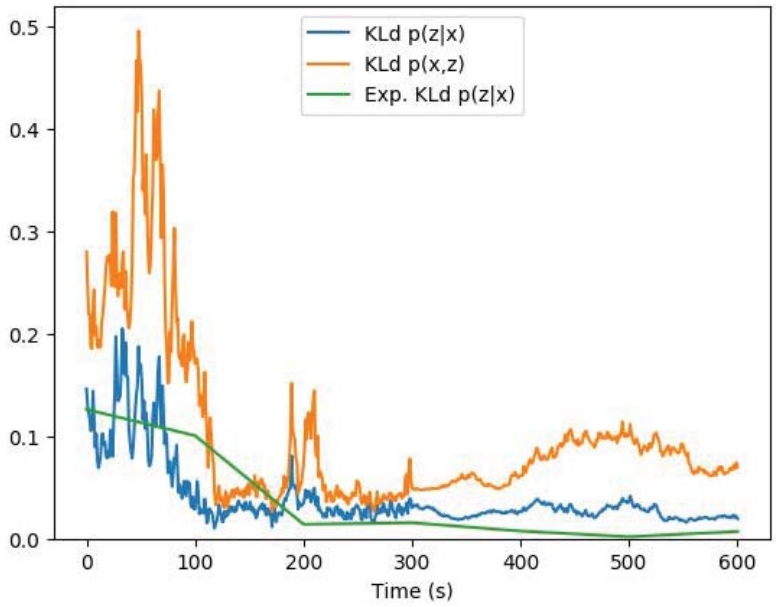

Figure 5.1: KL divergence values against time during a sample training run of the best GA individual at a synaptic delay setting of 0.1 ms. The orange trace gives the divergence between the target joint distribution $p^*(x^1, x^2, z)$ and the analytically-derived estimated $p(x^1, x^2, z)$, and the blue trace gives the divergence between the target conditional distribution $p^*(z|x^1, x^2)$ and its analytically-derived estimate $p(z|x^1, x^2)$. Equation 5 from [43] was used to calculate the analytical distribution. The green trace gives the divergence between the target conditional distribution and the experimental conditional distribution as measured by the spiking activity at regular intervals. The latter is used as fitness.

ness.

The traces provide evidence that *(1)* experimental and analytic conditional divergence follow similar profiles, but are not identical; *(2)* the analytic divergence is highly erratic with large deviations at the beginning, and exhibits movement even after STDP plasticity is halted, whereas the experimental divergence is smoother and settles down after the midpoint; *(3)* learning nearly converges onto its minimal value by about 200–300 s into the process. The dissociation of experimental divergence from theoretical values is a direct result of the violation of zero delay assumptions, as well as noise during measurement. Other sample training runs show similar profiles. See Figure 5.2.

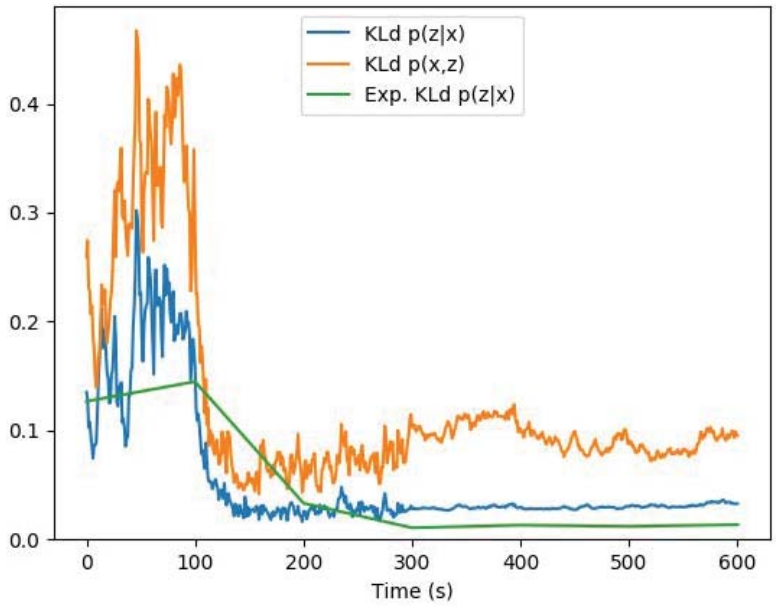

Figure 5.2: A second training run of the same individual as in Figure 5.1, showing a similar profile with an initial increase in analytic divergence that dissociates from the experimental divergence.

Pecevski & Maass does not give numeric results for Task A, but it does provide sample reconstructions of the target distributions. Figure 5.3, in particular, is a figure adapted from Pecevski & Maass that illustrates the learned and target conditional distributions. For comparative purposes, Figure 5.4 shows a similar diagram constructed after an arbitrary run of the optimal SAM used in Figures 5.1 and 5.2. Note that performance, in this case, is slightly better.

Finally, Figures 5.5 and 5.6 illustrate the evolution of fitness while running GA and NES on this task at a delay setting of 0.1 ms, respectively. Note that in the case of GA, mean and best fitness both improve until the end, suggesting that a longer optimisation run might have benefited the experiment. With NES, on the contrary, mean and individual fitnesses achieve their best values after about 30–40 generations with little improvement afterwards.[4]

[4]The number of generations in GA runs was limited by execution time—each GA fitness measurement required multiple simulations.

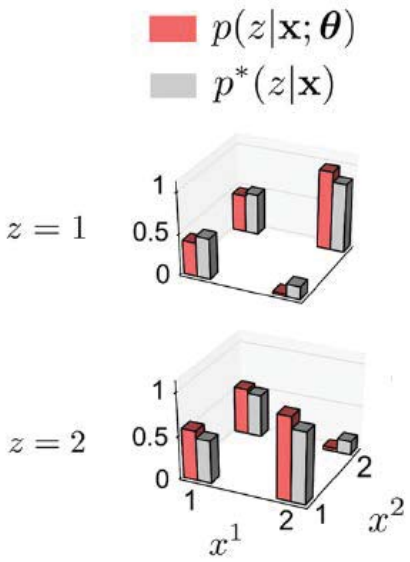

Figure 5.3: Conditional target (grey) and estimated (red) distributions after training on Task A. Adapted from [43].

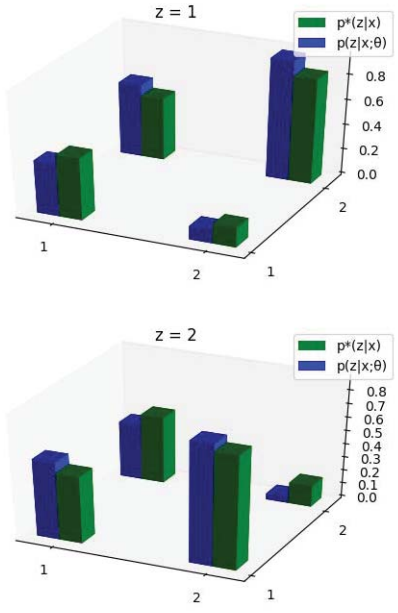

Figure 5.4: Conditional target (green) and estimated (blue) distributions after a training run of 600 s of the GA optimal SAM module in Figures 5.1 and 5.2.

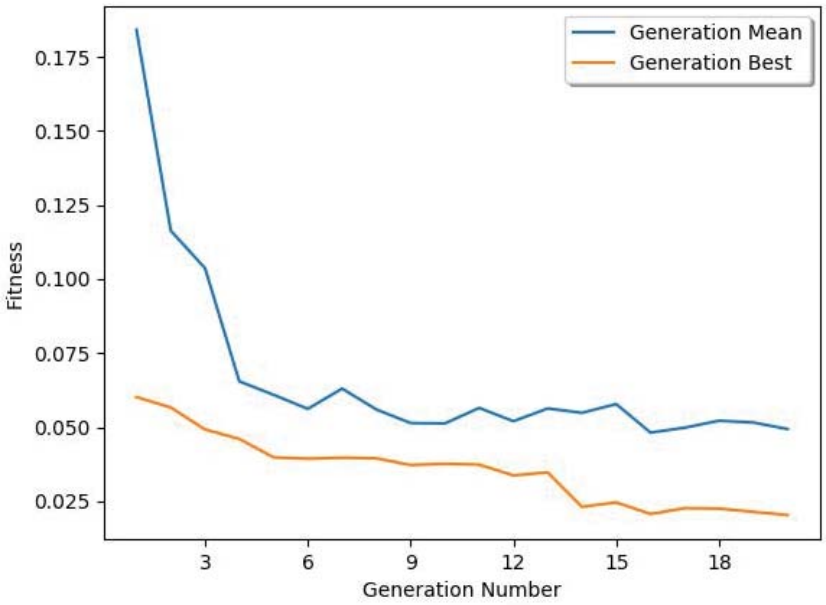

Figure 5.5: Mean generation fitness (blue) and best generation fitness (orange) as a function of generation number while running GA on Task A with delay set to 0.1 ms.

5.1.2 Task B

Four SAM modules are combined recursively for this experiment, following the second example in [43]. The structure and internal connectivity is illustrated in Figure 4.4. In this case, the network is used to learn the visual perception task given by the Bayesian network in Figure 4.3. The target joint probability distribution $p^*(y^1, y^2, y^3, y^4)$ is decomposed into the conditional probabilities given in Table 4.2. Note that $p^*(y^1) = p^*(y^2) = 0.5$.

As in Task A, optimisable hyperparameters and most fixed parameters are given earlier in Section 3.4, and experimental specifics noted in Section 4.2.2. Learning time is further reduced to 300 seconds, owing to the longer execution time of Task B experiments (several hours per optimisation run). However, instead of halting STDP halfway through, it is allowed to proceed throughout the entire 300 seconds, to compensate for the shorter training process. The second bias rate becomes

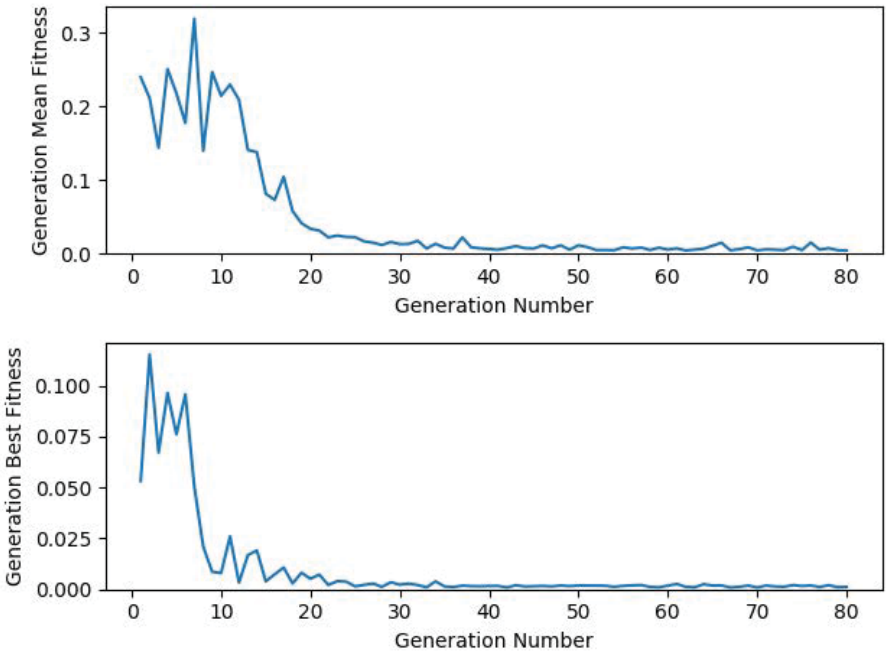

Figure 5.6: Two plots showing the evolution of generation mean fitness and best fitness, while running NES on Task A with a delay value of 0.1 ms.

irrelevant in this process. PSPs are additive[5] with a rectangular profile of width 15 ms, as in Task A. Again, due consideration is given to follow the work in Pecevski & Maass, with most parameters picked from the published paper.

Fitness is evaluated by collecting spontaneous activity of the network while no extraneous activity impinges upon it, which is then used to reconstruct the estimated joint distribution $p(\mathbf{y};\theta)$ and compare it with the target $p^*(\mathbf{y})$. Optimal performances at each delay value are given in Table 5.4. Performance is again in terms of mean KL divergence, this time measured in 5, rather than 10, repeated evaluations against the sample distribution in Pecevski & Maass [43], in the case of GA. In the case of NES, a single evaluation is used for fitness, for the same reasons that apply to Task A above (see Section 4.2.1 for justification). In addition, NES uses a generation size of 192 individuals, along with 40 generations and a learning

[5]As opposed to renewed, which is an option sometimes seen in published literature on the subject.

Table 5.4: Optimal performances: baseline, Task B

	Delay (ms)				
	0.1	0.2	0.5	1.0	2.0
GA	**0.3487**	0.5901	0.7076	1.0641	1.3017
NES^a	**0.5095**	0.7136	1.0163	1.0634	1.2992
NES^b	**0.5523**	0.7510	1.0861	1.1195	1.3540

rate of 0.5.[6]

Note that GA gives substantially better optimal performances at the lowest three delay settings, and marginally better performance at the highest two delay settings. As in Task A above, fair comparison is between GA and the best generation fitness reported by NES (i.e. the NES^b row in Table 5.4), since that takes into account noisy measurements, like the reported GA optima. Best performances are again indicated by bold figures.

During optimisation, one parameter is allowed to evolve independently in each module: bias baseline, $b\_$. This calibrates the bias update response on a per-module basis, allowing each module to respond effectively to a varying number of inputs (the modules representing y^1 and y^3 have two inputs, but those representing y^2 and y^4 have three and one, respectively). Being global, other parameters are not allowed to evolve separately (see Section 3.4). As noted earlier, when running GA the bit flip probability is increased from 0.01 to 0.05.[7]

Again, there is a strong relationship between synaptic delay and optimal performance. Both GA and NES exhibit this relationship. The optimal individuals are given in Tables 5.5 and 5.6. Only the scaling factor c_2, which governs the probability of firing at different membrane voltages, and $b\_^4$, the bias baseline for the fourth module, are relatively stable between delay values in the case of GA. None of the GA hyperparameters are closely related to their NES counterparts. Furthermore,

[6]The number of individuals, 192, was picked to make efficient use of the cores on the machine used for optimisation, which had 96 cores. When running the GA tasks, time had not been an issue yet.

[7]This higher probability gave a worse mean population fitness, but better top individuals. Thus, this problem requires the balance to be tipped slightly in favour of *exploration* rather than *exploitation*.

Table 5.5: Optimal hyperparameters: baseline, Task B, GA

	Delay (ms)				
	0.1	0.2	0.5	1.0	2.0
b^1_-	-4.4011	-5.6809	-8.6548	-11.8183	-10.1801
b^2_-	-0.0455	-5.2997	-21.8785	-34.6482	-30.2591
b^3_-	-9.0472	-6.4183	-9.2903	-10.6125	-20.7805
b^4_-	-33.6021	-39.0513	-34.7303	-20.4372	-35.6151
w_-	-1.6977	-2.2001	-7.2027	-0.4510	-0.6147
T	0.5670	0.5794	0.0814	0.4944	0.4292
R	0.1433	0.7490	0.6058	0.3941	0.4056
η'_0	0.0045	0.0324	0.0187	0.0728	0.0765
η_0	0.0059	0.0068	0.0046	0.0003	0.0018
η_1	0.0006	0.0061	0.0064	0.0021	0.0035
c_1	0.0218	0.2133	0.1830	0.2598	0.4465
c_2	1.5578	1.1971	1.1789	1.7364	1.5200

some of the hyperparameters in the case of NES appear to be highly variable, suggesting that either *(1)* they are irrelevant to fitness, or *(2)* the algorithm did not converge well. More on this in Section 5.3.

The optimal performances (whole-state divergences) above are supplemented by the valid-state divergences of the optimal individuals in Table 5.7. To calculate valid-state divergences, 5 trials are made with the optimal individuals at each delay setting. Note that the positive relationship with delay also extends to valid-state divergence.

In all cases optimisation use whole-state divergence as fitness.[8] Although whole-state divergences are not competitive with respect to the results reported in Pecevski & Maass, valid-state divergences are far superior. This result suggests that optimising the SAM architecture against non-zero delay forces it to simultaneously use a lower probability of firing and to maintain an approximation of WTA-like competition between hidden neurons. High probability of firing leads to simultaneously active neurons in the hidden layer, destroying competitiveness and negatively impacting learning, as will be seen in Section 6.1. Low probability of firing, on the other hand, allows zero states to occur by introducing long gaps in activity. These

[8]Attempts to use valid-state divergence failed to converge.

Table 5.6: Optimal hyperparameters: baseline, Task B, NES

	Delay (ms)				
	0.1	0.2	0.5	1.0	2.0
b_-^1	-26.5761	-9.7322	-36.8226	0.0000	-11.0110
b_-^2	-0.6335	0.0000	-13.8751	-17.0267	-26.3419
b_-^3	-20.9019	-22.4464	-10.6245	-28.7049	-26.7565
b_-^4	-33.9196	-36.8068	-20.7066	-17.8488	-39.1249
w_-	-1.8440	-1.9799	0.0000	0.0000	0.0000
T	0.5709	0.5390	0.5908	0.0000	0.3521
R	1.0000	0.1872	0.0000	1.0000	1.0000
η_0'	0.0019	0.0069	0.0000	0.0000	0.0000
η_0	0.0100	0.0046	0.0078	0.0100	0.0100
η_1	0.0000	0.0072	0.0000	0.0094	0.0000
c_1	0.0418	0.0751	0.7132	1.0000	1.0000
c_2	1.3001	1.2967	1.6953	1.5350	1.3950

Table 5.7: Valid-state divergences: baseline, Task B

	Delay (ms)				
	0.1	0.2	0.5	1.0	2.0
GA	**0.0284**	**0.0534**	0.0887	0.5981	0.6147
NES	**0.0704**	0.0873	0.7283	0.6566	0.6147

opposing factors thus lead to a trade-off between too many zero states and ineffective training.

Figure 5.8 draws the estimated and target distributions of Task B on the same plot, for an arbitrary run of the best GA individual at a delay value of 0.1 ms. The two distributions are very similar. For comparison, see Figure 5.9, which is adapted from Pecevski & Maass. Only graphical plots of performance are provided in Pecevski & Maass (Figure 5.7, reproduced from Section 2.6), but our results indicate that after truncating zero states, the optimal individual's performance improves upon that illustrated in [43], even though non-zero delay is used. Figure 5.7, which is adapted from Pecevski & Maass, illustrates the temporal evolution of KL divergence of the hand-crafted model used therein. Note that it reaches a minimum (best) divergence of about 0.08–0.1, while the valid-state divergences achieved here at delay values of 0.5 ms or lower give better or comparable performance. Valid-

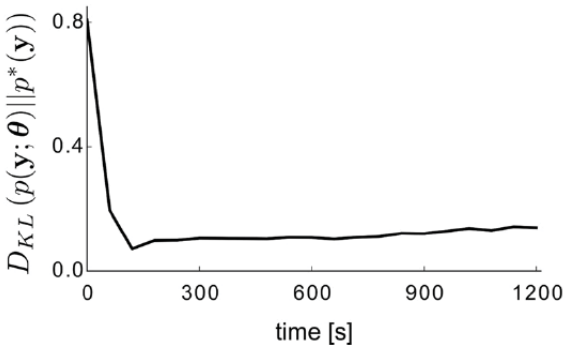

Figure 5.7: KL divergence in Task B as a function of simulated biological time, reported by Pecevski & Maass [43].

state divergences that clearly improve upon that reported in [43] (i.e., 0.08–0.1) are displayed in bold type in Table 5.7.

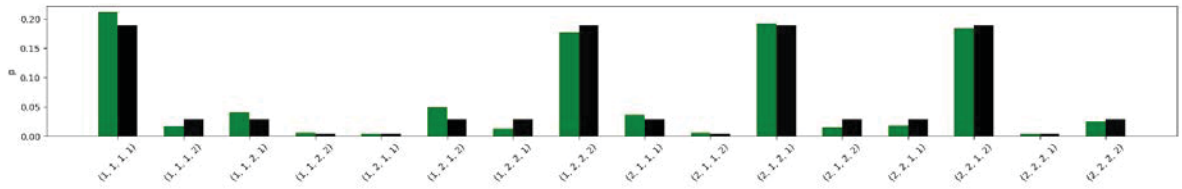

Figure 5.8: Target distribution (black) and estimated distribution (green) for Task B, with the estimated distribution reconstructed from the spontaneous spiking activity of the optimal GA individual at a delay setting of 0.1 ms. Zero states were truncated and the distribution renormalised.

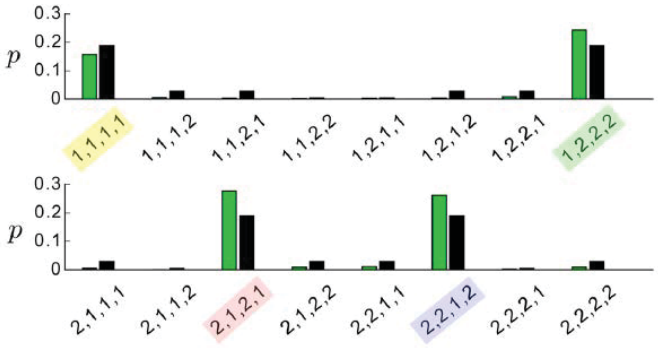

Figure 5.9: Target distribution (black) and estimated distribution (green), as learned by the hand-crafted model in Pecevski & Maass for Task B. The coloured labels correspond to the most prevalent states. Adapted from [43].

The divergence between target and analytical distributions embodied by each

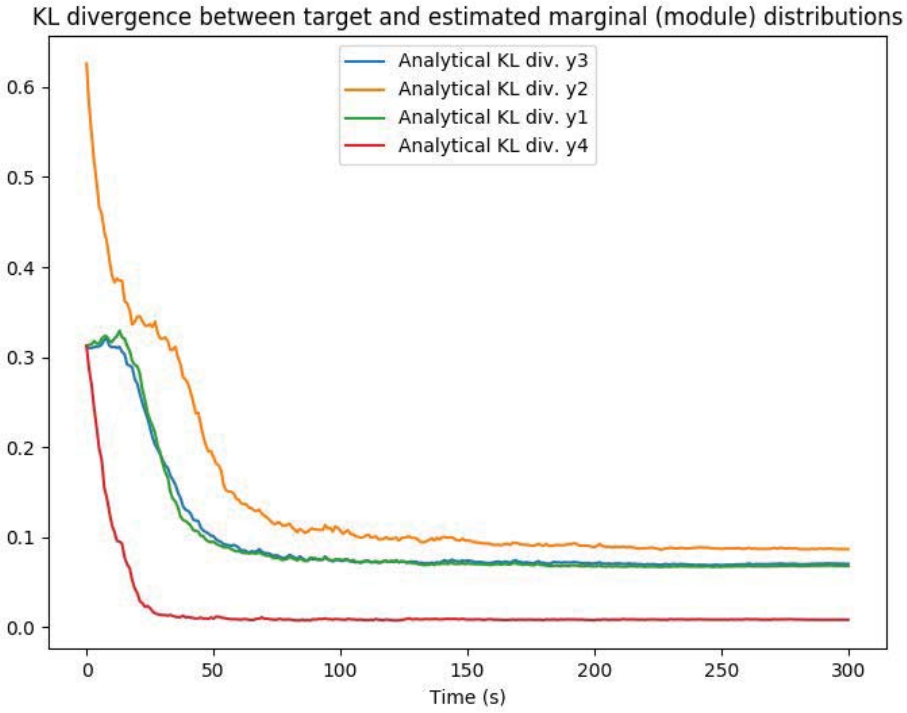

Figure 5.10: KL divergence between the appropriate marginalised distribution and the estimated analytical marginal distribution encoded by each module, as a function of time during the training process, for the optimal GA individual at 0.1 ms delay.

module in the network during a typical training run of the best GA individual at 0.1 ms delay are plotted in Figure 5.10. Figure 5.11 reproduces the divergence profile reported in Pecevski & Maass for comparison. Note that in our case, the four modules' analytical distributions do not all converge to zero. The discrepancy result arises from violations of theoretical conditions: the statistics of spontaneous network activity dissociate from the analytical distribution embodied by the network under the assumption of zero delay.

Figure 5.12 reproduces the evolution of experimental divergence determined by the spiking activity of the whole recurrent SAM network during a typical training run on the best individual (GA, 0.1 ms delay). As described in Section 4.2.2, to measure the distribution of network states the network was simulated for 20 seconds of biological time with the biases and connection weights achieved at regular points

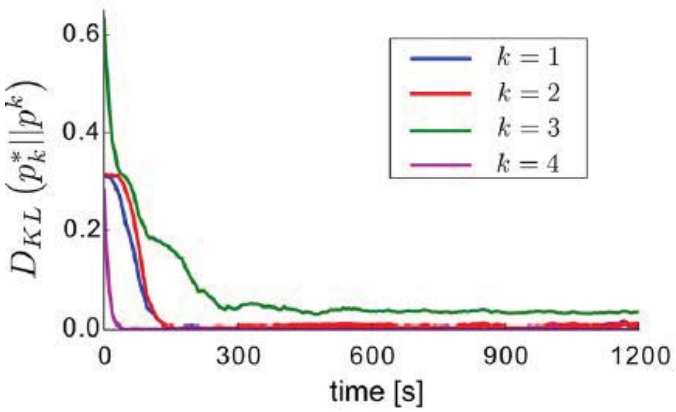

Figure 5.11: KL divergence plot from Pecevski & Maass ([43]) corresponding to Figure 5.10, reproduced here for convenience.

during the training process, and with no external input. The interval was set to 100 seconds.[9] The figure clarifies the effect on KL divergence of truncating zero states: the resulting divergence drops to a low value, traced by the orange curve.

The evolution of fitness is illustrated in Figures 5.13 and 5.14 for GA and NES respectively, for the specific case of 0.1 ms delay. Note that both experiments might have benefited from longer runs.

5.2 Generalisability Experiment

The purpose of this experiment is to understand the generalisability properties of the SAM and SPI architectures (see Section 4.2.3), while comparing them against each other and using different evolutionary techniques. For this experiment, delay is constant at 0.2 ms (SAM) or fixed to a range 0.1 ms–0.3 ms (SPI). General parameters are listed in Section 3.4 and experiment-specific ones in Section 4.2.3. Instead of repeated fitness measurements on the same target distribution from Pecevski & Maass, individuals are measured against different randomly parametrised target distributions, with the optimal individual from each optimisation run further tested on an unseen set of target distributions for the final evaluation. For fairness, each

[9]More frequent evaluation of performance would have proved very costly in terms of computation time.

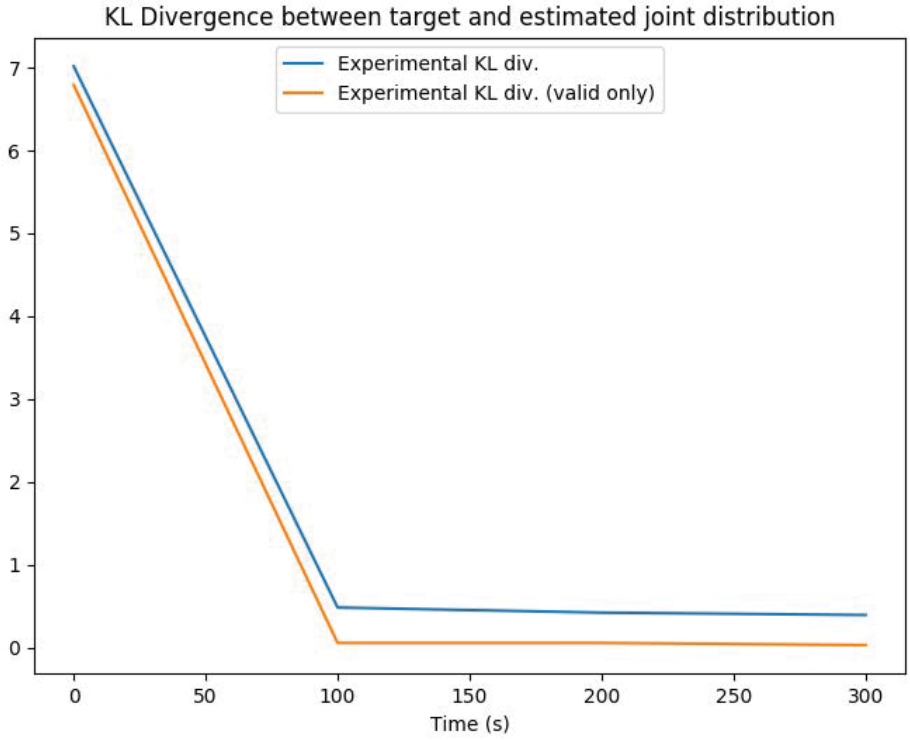

Figure 5.12: KL divergence between the target distribution $p^*(\mathbf{y})$ and the estimated distribution $p(\mathbf{y};\theta)$ determined from the network's spontaneous spiking activity, as a function of time during the training process. The blue trace shows the whole-state divergence (see Section 4.2.5), which includes zero states among the estimated distribution, while the orange trace shows the valid-state divergence. Only valid states of the target distribution $p^*(\mathbf{y})$ are retained in the valid-state estimate distribution, renormalised to 1. An arbitrary run of the optimal Task B GA individual at a delay of 0.1 ms was used.

task is run against a fixed number of target distributions (10, Task A; 5, Task B), and the distributions are reused between individuals and architectures. Only best fitness is reported for NES optimisation. This is because both GA and NES experiments have to repeat fitness measurements on the same number of distributions, for fairness, as opposed to the baseline experiment, where it was permissible to eliminate repeat measurements for efficiency if the evolutionary algorithm allowed it.

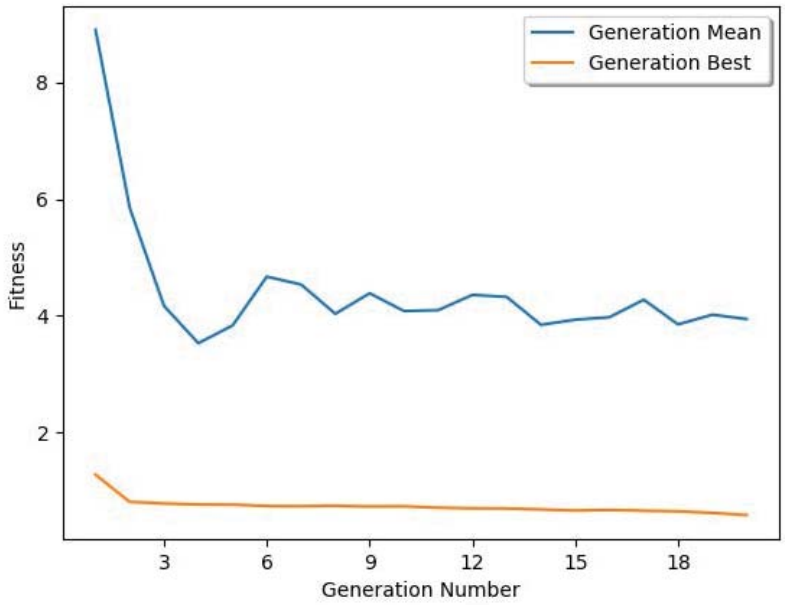

Figure 5.13: Mean generation fitness (blue) and best generation fitness (orange), as a function of generation number while running GA on Task B with delay set to 0.1 ms.

5.2.1 Task A

Outer-loop optimisation by means of GA and NES are kept as far as practicable in line with that described in Section 5.1.1. Indeed, GA runs are identical to the Task A baseline experiment runs, but for NES runs the additional repeated measurements for fair comparison require a reduction of generation size from 192 to 96. The number of generations is set to the same value of 80, and learning rate is set to 1 in SAM runs, and 0.5 in SPI runs.[10]

Table 5.8 gives the training performances of the two architectures, calculated as the mean KL divergence on the set of 10 training distributions. Clearly, overall performances are worse than those reported for the baseline experiment at 0.2 ms synaptic delay. This is due to the introduction of multiple targets, as opposed to the baseline, which trained each setup against one target only. Results show that

[10]Greedy learning rates tended to lead to bad convergence in SPI runs.

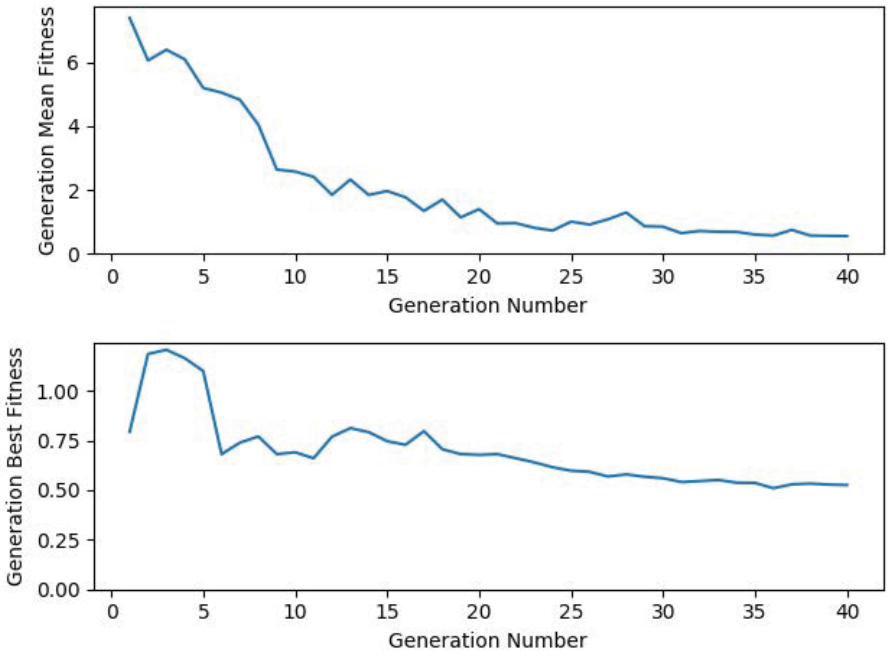

Figure 5.14: Two plots showing the evolution of generation mean fitness and best fitness while running NES on Task B with a delay value of 0.1 ms.

Table 5.8: Optimal training performances: generalisability, Task A

	Architecture	
	SAM	SPI
GA	**0.0188**	0.0271
NES	0.0297	0.0235

the optimal SAM individual ($p_\theta = 0.0188$) outperforms the optimal SPI individual ($p_\theta = 0.0235$) on the optimisation set, and NES optimisation outperforms GA on SPI. The latter relationship is reversed on SAM.

Generalisability is given in Table 5.9. Overall, SPI outperforms SAM on generalisability tests (0.0374 vs 0.0379), although the margin is not significantly large, and may be due to random effects.[11] Best performances on training and test data are indicated in bold type.

To compare the generalisation of networks optimised for performance on multiple distributions against those optimised on a single distribution, here we report

[11]Statistical analysis was ruled out by time constraints. See Discussion, Chapter 6.

Table 5.9: Optimal test performances: generalisability, Task A

	Architecture	
	SAM	SPI
GA	0.0379	0.0730
NES	0.0510	**0.0374**

Table 5.10: Optimal hyperparameters: generalisability, Task A, SAM

	GA	NES
b_-	-7.4902	-7.0260
w_-	-4.1583	-3.1791
T	0.5055	0.6367
R	0.8535	0.5516
η'_0	0.0263	0.0587
η'_1	0.0003	0.0132
η_0	0.0099	0.0100
η_1	0.0063	0.0100
c_1	0.9941	0.4559
c_2	0.5941	0.9793

the performances of the optimal GA and NES individuals from the baseline experiment when run on the unseen distributions which we test with in this section.[12] Respectively, the performances are 0.0569 and 0.0465. These values are inferior to those reported in this section—0.0379 and 0.0374—indicating that configurations optimised against multiple distributions generalise better, as expected.

SAM and SPI optimal individuals are reported in Tables 5.10 and 5.11 respectively. Although most parameters differ significantly between GA and NES, some are within close proximity of each other, suggesting that the optimal solution is sensitive to those parameters. See Section 5.3 for detailed analysis. In particular, most connection probabilities in SPI—the parameters ρ—turn out to be within 3–4% of each other. Interestingly, the recurrent connection weights (w_E and w_I) are both 0 in the optimal SPI individual, suggesting that recurrent pool connections are not necessary for performance in Task A.

Figures 5.15 and 5.16 plot the evolution of fitness as a function of generation

[12]For fairness, we pick the GA and NES individuals optimised at a synaptic delay of 0.2 ms.

Table 5.11: Optimal hyperparameters: generalisability, Task A, SPI

	GA	NES
b_-	-17.9025	-24.0672
w_-	-3.0948	-2.7015
T	0.7070	0.6057
R	0.3436	1.0000
η'	0.0904	0.0590
η_0	0.0063	0.0052
η_1	0.0064	0.0081
c_1	1.6629	1.4438
c_2	1.2612	0.8278
ρ_{PP}	0.5365	0.5167
ρ_E	0.3305	0.3699
ρ_{PI}	0.3144	0.4300
ρ_I	0.9644	1.0000
ρ_{IP}	0.9728	1.0000
w_{MAX}	1.2850	1.8790
w_E	0.1740	0.0000
w_{PI}	11.8626	5.7034
w_I	-10.2197	0.0000
w_{IP}	-1.9281	-2.2727

number for SAM under GA and NES optimisation respectively. Note that the NES run shows a suspiciously quick improvement in best fitness that rebounds to a near-constant level which never recovers the best value. The minimum occurs while the mean generation fitness has not yet settled, suggesting that the algorithm overshot a narrow "ridge" or optimum that was surrounded by relatively low fitness solutions—the typical "needle in a haystack" problem with optimisation algorithms (see also the Discussion, Chapter 6).

Figure 5.17 reproduces the evolution of fitness during the GA run on SPI.[13] In particular, note that the best fitness has an improving trend that does not slow down by the last generation, suggesting that the experiment could have shown further improvement if the run were extended.

The experimental and analytical KL divergence during an arbitrary training

[13]The NES run evolution is not shown, to conserve space, but it shows no particularly interesting features.

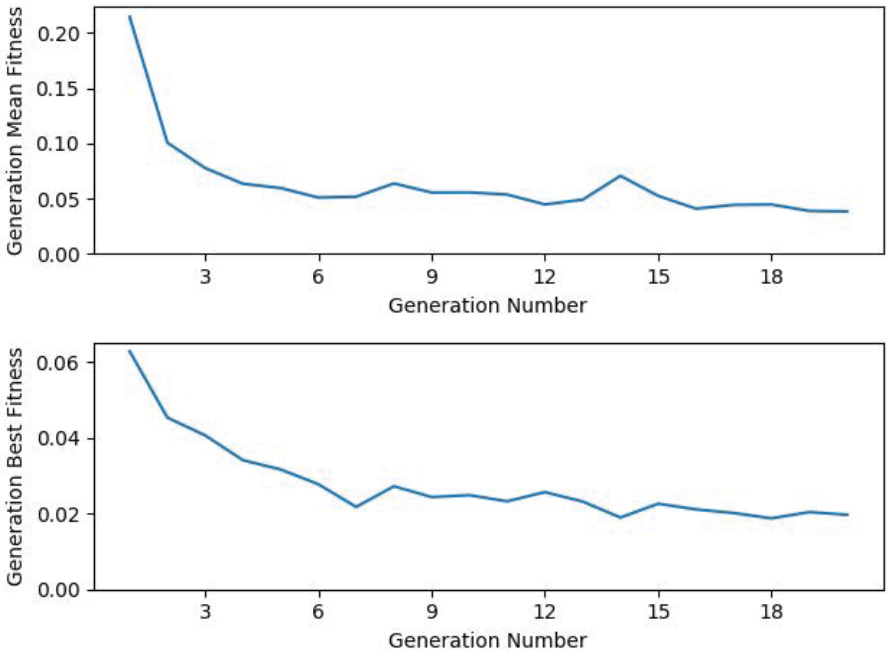

Figure 5.15: Mean generation fitness and best generation fitness as a function of generation number while running GA on SAM in Task A/Generalisation Experiment with delay set to 0.2 ms.

run of the 0.2 ms delay GA-optimal SAM individual are plotted in Figure 5.18. This individual is given in the first column of Table 5.10, and corresponds to the best SAM performance reported in this section. As in Section 5.1.1, the analytical and experimental traces diverge, for the same reasons. The estimated and target distributions that correspond to this training run are given in Figure 5.19—note the close similarity.

Moving on to SPI, Figure 5.20 shows a plot of divergence against time for the NES-optimal SPI individual against one of the random target distributions in the test set. Figure 5.21 reconstructs the target distribution from Figure 5.20 and its experimental estimate after 300 seconds on the same individual. The close similarity is evidence that the SPI module is indeed learning its target distribution. Note that an analytic derivation of the estimated distribution analogous to the one for SAM (Equation 2.9) is not available, rendering it impossible to create divergence trace plots similar to Figures 5.1 and 5.10—only the experimental distribution

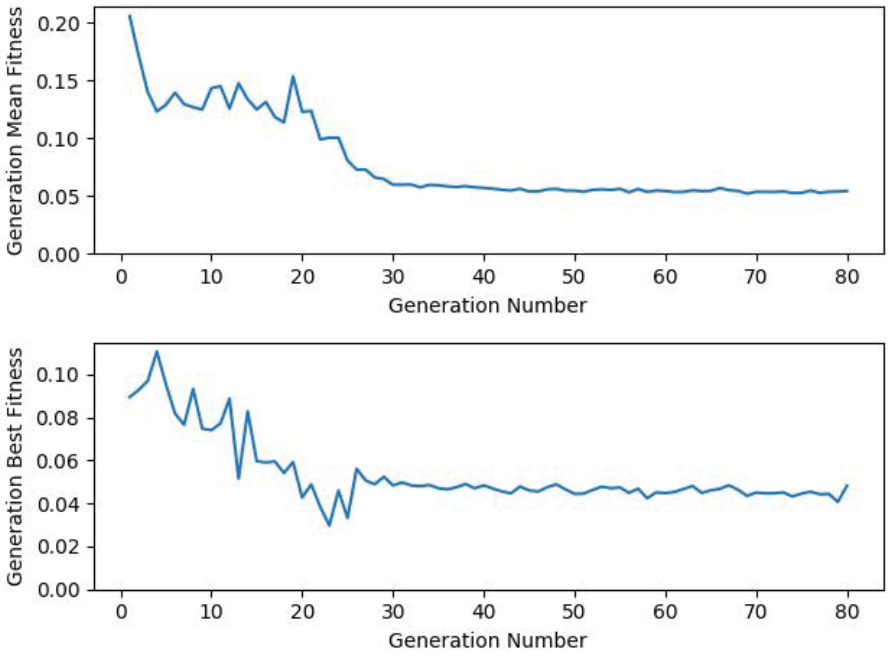

Figure 5.16: Mean fitness and best fitness while running NES on SAM in Task A/Generalisation Experiment with delay set to 0.2 ms.

derived from spiking activity can be measured.

Figure 5.22 shows the typical activity that results from forcing input into the SPI module that corresponds to one incomplete sample drawn from the target distribution in Figure 5.21. Note that activity very well reflects the difference in conditional probability between $p^*(y^3 = 1|y^1 = 1, y^2 = 2)$ and $p^*(y^3 = 2|y^1 = 1, y^2 = 2)$. Figures 5.20, 5.21 and 5.22 use an SPI module with the same configuration.

5.2.2 Task B

For Task B, the evolutionary algorithm parameters were identical to those in Task A above, Section 5.2.1, except that the number of generations in NES runs is halved to 40, and the learning rates of both SAM and SPI tests are set to 0.5. A set of 5 training distributions was used in this case.

Optimisation set performances of SAM and SPI are reported in Table 5.12. Table 5.13 gives generalisation performances. In the case of SAM, values in ital-

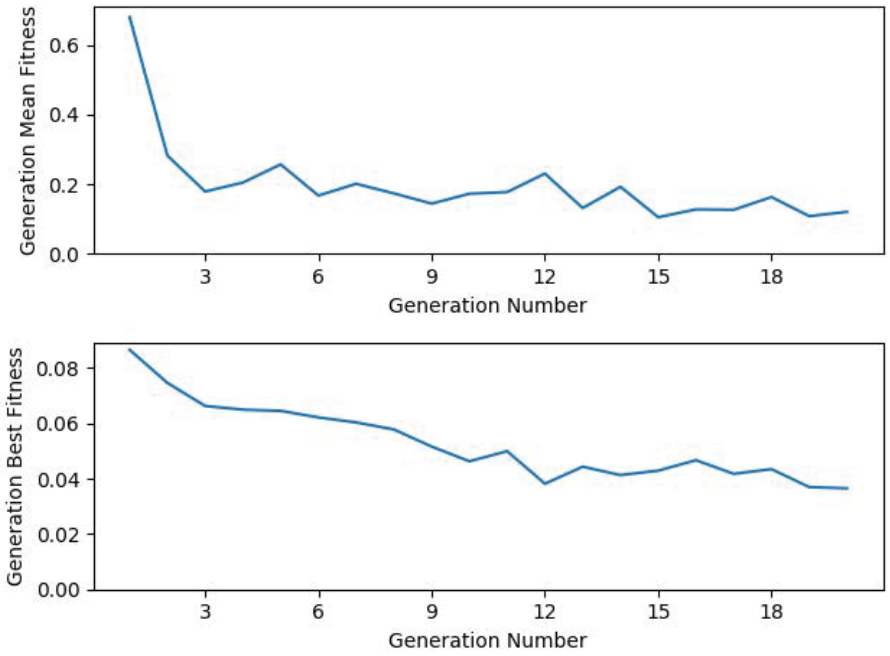

Figure 5.17: Mean generation fitness and best generation fitness as a function of generation number while running GA on SPI in Task A/Generalisation Experiment with random delay set to 0.1 ms–0.3 ms.

Table 5.12: Optimal training performances: generalisability, Task B

	Architecture	
	SAM	SPI
GA	0.7297 ***[0.0849]***	0.1173
NES	0.8040 *[0.1345]*	0.1627

ics indicate valid-state divergence, a fairer metric to compare with SPI than the whole-state divergence (or "performance"). Bold figures indicate best performance. Results show that SAM performs and generalises better than SPI at a delay of 0.2 ms.

Simulating the optimal GA SAM individual from the baseline experiment[14] at a delay of 0.2 ms against the unseen distributions gives a performance of 0.9031 and a valid-state divergence of 0.2139. As expected, this is inferior to the optimal SAM test performances reported in Table 5.13, indicating that optimising against

[14]That is, the individual in the second column of Table 5.5.

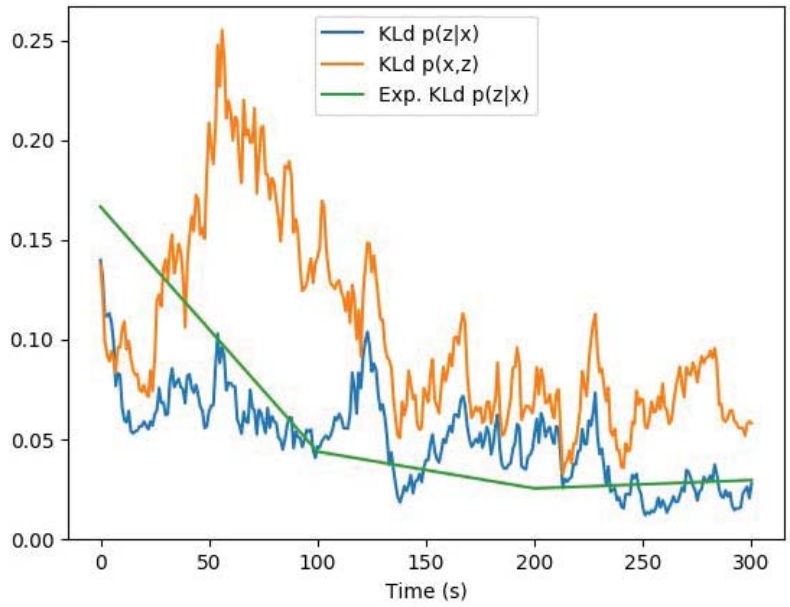

Figure 5.18: Analytical (joint in orange; conditional in blue) and experimental (green) KL divergence plots of the GA-optimal individual as a function of time (in s) during an arbitrary training run on the same target distribution as in Figure 5.19. Generalisation Experiment/Task A

Table 5.13: Optimal test performances: generalisability, Task B

	Architecture	
	SAM	SPI
GA	0.7716 ***[0.1102]***	0.2505
NES	0.8720 *[0.1936]*	0.3245

a number of different distributions leads to better generalisation. Incidentally, the generalisability (in terms of valid-state divergence) of the best SAM individual compares well to the results obtained in Pecevski & Maass (∼0.1; see Figure 5.7) on the sample distribution there, with the added benefit that these reports are indicative of general performance, and that they are achieved under non-zero synaptic delay.[15] SPI, on the other hand, shows poor generalisability.

SAM and SPI optimal individuals are reported in Tables 5.14 and 5.15 respectively. In both instances, NES appears to have converged to a spurious result: SAM

[15]Moreover, using a synaptic delay of 0.1 ms would have further improved generalisability.

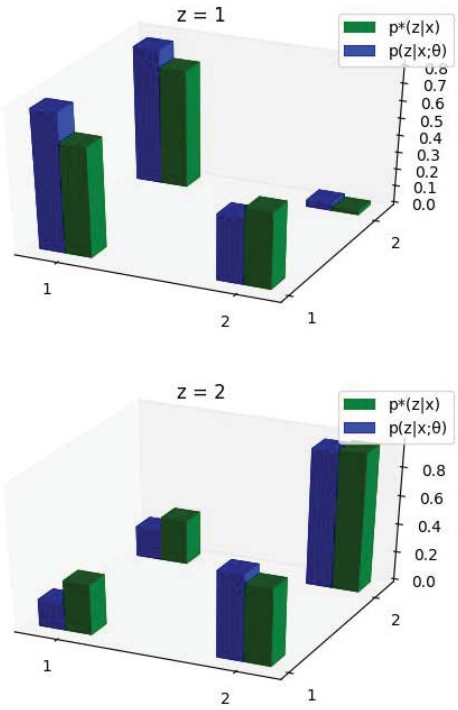

Figure 5.19: Reconstruction of the target (green) and estimated (blue) distributions after an arbitrary training run of the GA-optimal SAM individual in the Generalisation Experiment/Task A.

and SPI alike have optimal individuals with implausible values of T—implying no weight updates could occur during training. Moreover, c_2 being 0 in the optimal NES SPI individual implies a firing rate independent of membrane potential, again an implausible configuration.

Evolution of fitness is shown in Figures 5.23 and 5.24 for SAM under GA and NES respectively, both showing only a moderate improvement in performance over generations. The NES fitness evolution of SPI is plotted in Figure 5.25. The lack of overall improvement in best fitness coinciding with an improvement of mean fitness suggests that generation fitness is dominated by the effect of low fitness individuals, such that improving the mean does not necessarily improve the best fitness.

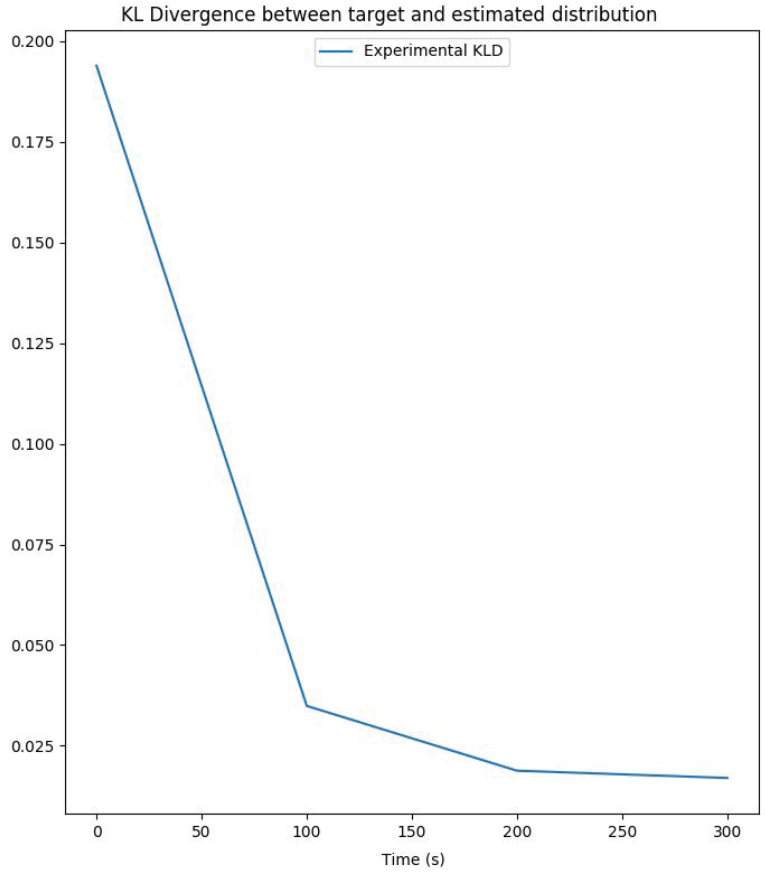

Figure 5.20: KL divergence between estimate and target as training proceeds over 300 seconds, for an arbitrary distribution in the test set of 10 distributions, using the NES-optimal SPI individual from this section. Generalisation Experiment/Task A

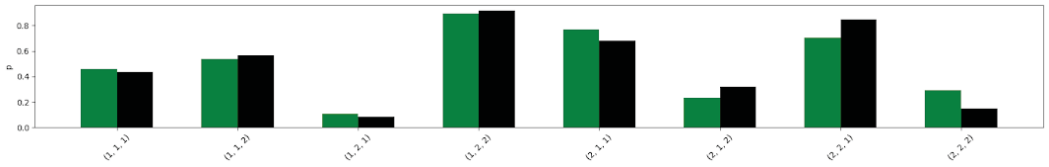

Figure 5.21: Target (black) conditional distribution and its corresponding estimate (green) distribution, after 300 seconds of exposure of the optimal SPI module to samples drawn from the target distribution. The target distribution is the same as in Figure 5.20. Generalisation Experiment/Task A

In Figure 5.26 we reconstruct the target and estimated joint distribution after a run of the GA-optimal SAM individual on an arbitrary target distribution in

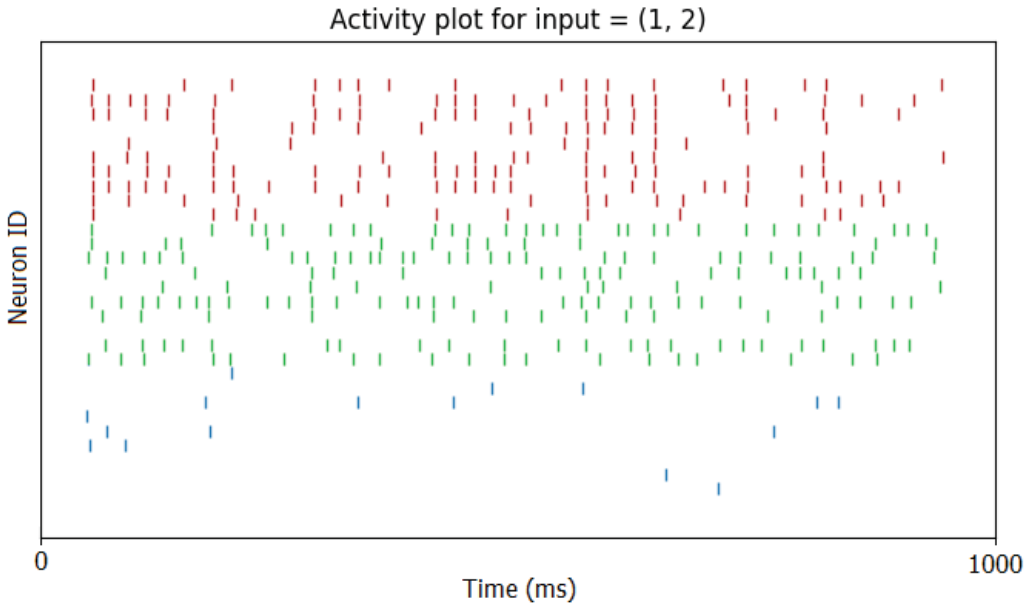

Figure 5.22: Activity over 1000 ms, after the optimal SPI module has been trained on the distribution from Figure 5.21, with input forced to encode the incomplete sample $(1, 2, *)$. Excitatory neurons 1–10 (blue) encode output $y^3 = 1$; excitatory neurons 11–20 (green) encode output $y^3 = 2$; neurons 21–30 (red) are inhibitory neurons connected to both excitatory subpopulations. Note that in correspondence with Figure 5.21, the module exhibits significantly higher activity in the subpopulation that encodes $y^3 = 2$.

the test set.[16] The visual similarity between target and estimated distributions is striking.[17] Compare this with Figure 5.27, which shows the reconstruction by the optimal SPI individual of the same target distribution as in Figure 5.26. Clearly, the SPI reconstruction is inferior to the SAM estimate, reflecting the performance results reported in Table 5.13.

Figure 5.28 traces the analytic divergence in time during a training run of the GA-optimal SAM individual in this section.[18] This is analogous to the plots in Figure 5.10. Figure 5.28 can be compared with Figure 5.29, which shows the experimental divergence measured by capturing the spontaneous activity of the network with no forcing input. Although the analytic divergences of the module

[16] Which corresponds to the best SAM individual given in the first column of Table 5.14.

[17] The other distributions show more or less similar discrepancies between targets and estimates.

[18] The divergence plot of the optimal SPI individual is less relevant, since it does not exhibit good performance.

Table 5.14: Optimal hyperparameters: generalisability, Task B, SAM

	GA	NES
b_{-}^{1}	-25.9407	-40.0000
b_{-}^{2}	-24.8484	0.0000
b_{-}^{3}	-8.9220	-40.0000
b_{-}^{4}	-9.4224	0.0000
w_{-}	-1.0419	-3.7591
T	0.8691	0.0000
R	0.1013	1.0000
η_0'	0.0349	0.0064
η_0	0.0006	0.0099
η_1	0.0015	0.0035
c_1	0.5471	0.2416
c_2	1.1154	1.9900

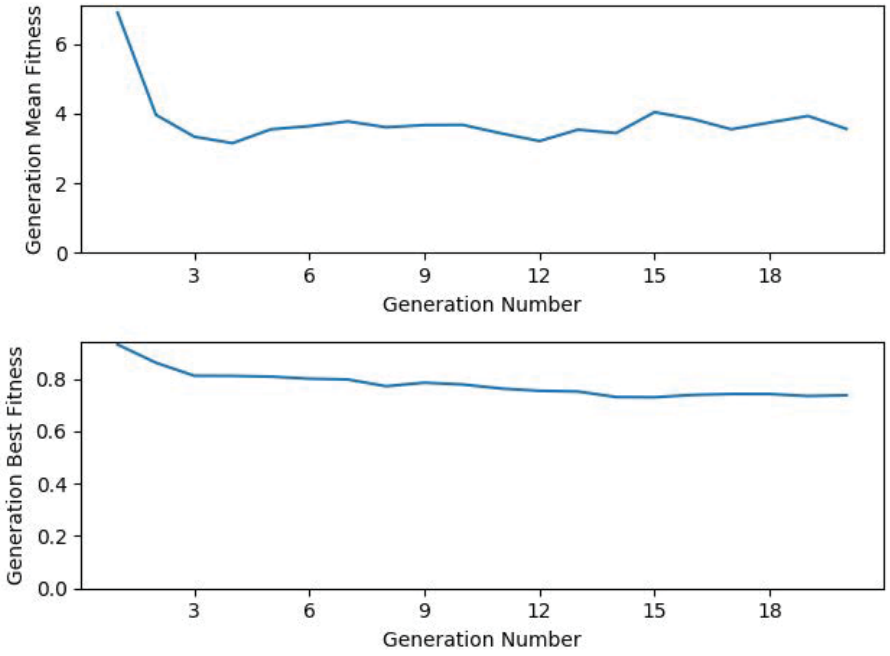

Figure 5.23: Mean generation fitness and best generation fitness as a function of generation number while running GA on SAM in Task B/Generalisation Experiment with delay set to 0.2 ms.

do not all converge to zero—and in fact y^1 shows a slight increase, while y^2 settles on a high value—the experimental divergence, specifically the valid-state divergence, converges to a very low value as quantified by Table 5.13.

Table 5.15: Optimal hyperparameters: generalisability, Task B, SPI

	GA	NES
b_{-}^{1}	-12.3966	0.0000
b_{-}^{2}	-1.7398	-26.9311
b_{-}^{3}	-12.3222	-10.3797
b_{-}^{4}	-16.4097	-27.6312
w_{-}	-1.4751	-9.6932
T	0.2742	0.0000
R	0.8719	0.0000
η'	0.0746	0.0412
η_0	0.0039	0.0056
η_1	0.0097	0.0100
c_1	1.7052	2.0000
c_2	0.1093	0.0000
ρ_{PP}	0.7687	1.0000
ρ_E	0.1126	0.3734
ρ_{PI}	0.4741	0.3317
ρ_I	0.5839	0.4324
ρ_{IP}	0.7498	0.2150
w_{MAX}^{1}	5.9462	1.2601
w_{MAX}^{2}	6.0427	2.7647
w_{MAX}^{3}	0.6634	0.0100
w_{MAX}^{4}	3.6381	4.3172
w_E	10.8536	13.5785
w_{PI}	7.5167	8.6378
w_I	-0.8473	-1.2586
w_{IP}	-2.4896	-2.5703

5.3 Sensitivity Analysis

By means of the search distribution's covariance matrix, NES provides an insight into the sensitivity of the fitness landscape to perturbations in the network hyperparameters. This is not a direct measurement of sensitivity—it would be far too expensive to run a reasonably detailed analysis by manually perturbing each parameter in turn and observing results. Instead, the search distribution under the NES hood provides an indirect tool for quantifying sensitivity: in particular, NES adapts the width of its search distribution to the fitness landscape, widening it in flatter landscapes and narrowing around clear, sharp optima [54] (the website at

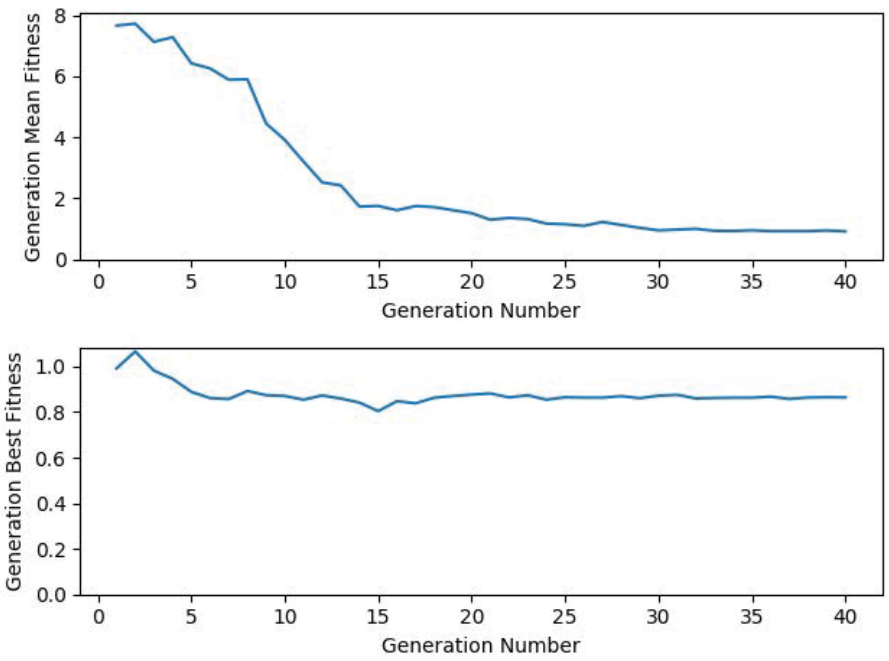

Figure 5.24: Mean fitness and best fitness while running NES on SAM in Task B/Generalisation Experiment with delay set to 0.2 ms.

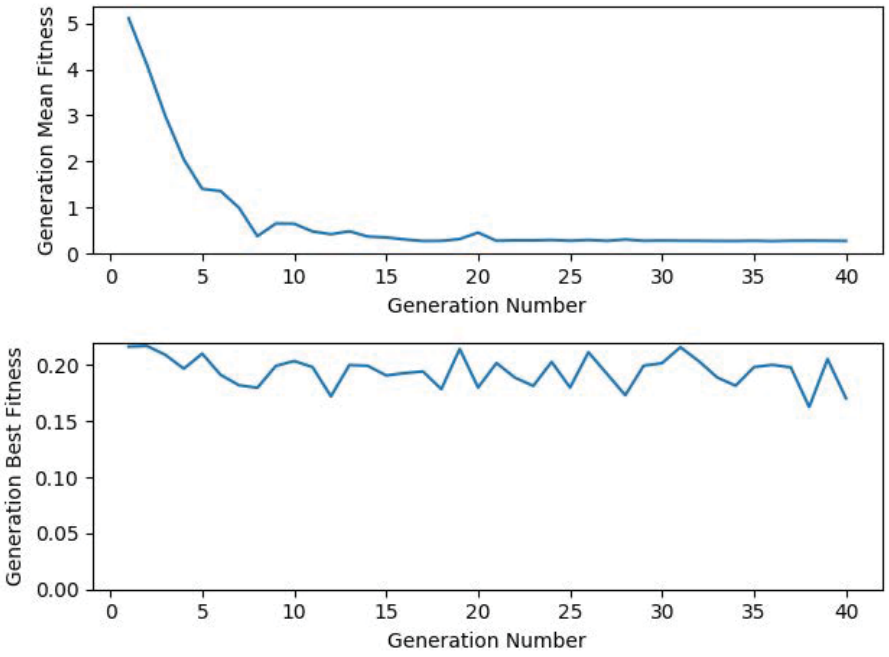

Figure 5.25: Mean fitness and best fitness while running NES on SPI in Task B/Generalisation Experiment with random delay set to 0.1–0.3 ms.

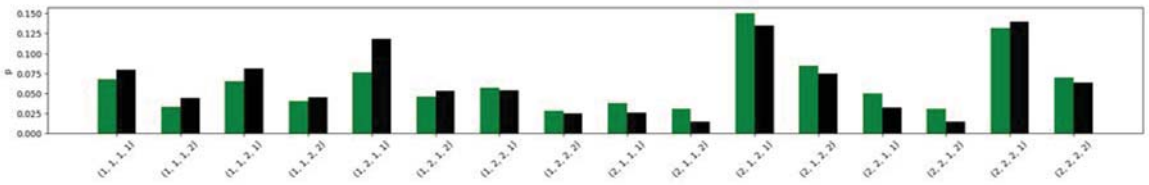

Figure 5.26: Target (black) joint distribution and its estimate (green), after 300 seconds of exposure of the GA-optimal SAM network to samples drawn from the target distribution. Generalisation Experiment/Task B

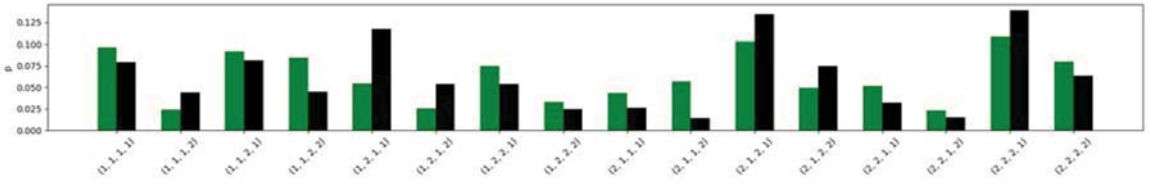

Figure 5.27: Target (black) joint distribution and its estimate (green), after 300 seconds of exposure of the GA-optimal SPI network to samples drawn from the target distribution. The target distribution here is the same as in Figure 5.26. Generalisation Experiment/Task B

[1] gives a good visual overview of the process).

Tables 5.16–5.20 give the mean and standard deviation of the best generation's search distribution for each of the main experiments, at low synaptic delays.[19]

Several of the hyperparameters exhibit a high tolerance, compared to the distribution mean and the ranges specified earlier in Section 3.4. Specifically, some parameters have means outside the optimisation search space, which together with narrow standard deviations give ranges that do not overlap with the optimisable ranges. Good examples are the first bias rate η' in all SAM experiments except Generalisability/Task A, and the self-connection probability ρ_E and excitatory self-weight w_E in the last table. Other parameters—such as the bias baseline b_- in Task A SAM experiments—have means inside the specified ranges, but wide standard deviations; others yet have means outside bounds but standard deviations that overlap into the ranges, such as b_-^2, R, and η_1 in the Task B SAM baseline experiment. Some of these ranges represent impossible, non-physical values, such as the

[19]Task B on SPI is excluded since it resulted in low performance and is therefore uninteresting.

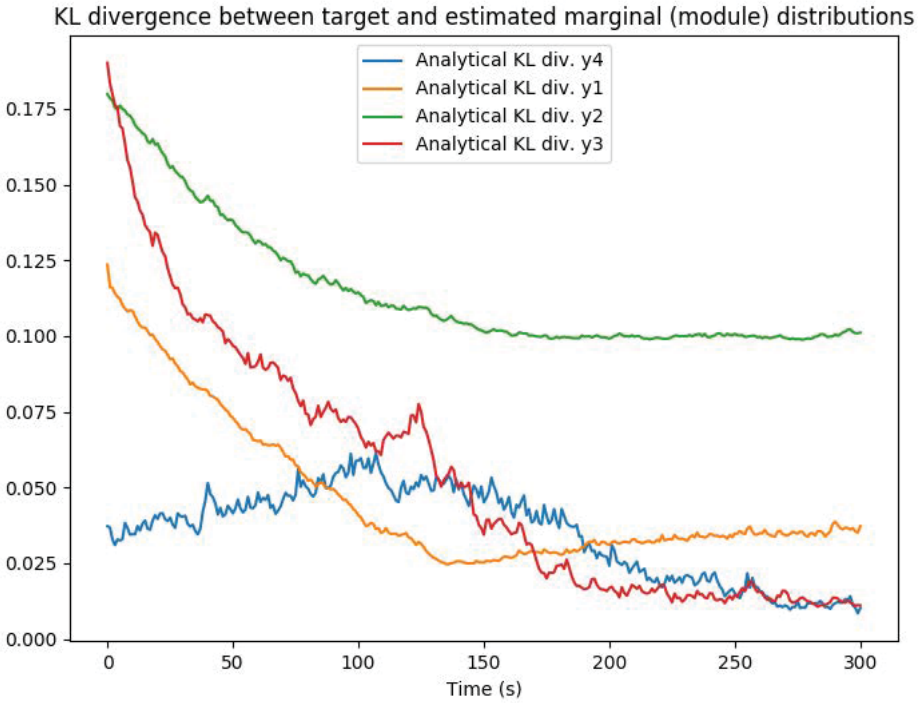

Figure 5.28: Analytic divergence against time (in s) of the marginal distributions represented by the four modules making up the GA-optimal SAM individual at 0.2 ms delay, during a training run on an arbitrary distribution from the test set. Note that the analytic divergence does not approach zero, because the analytic derivation (Equation 2.9) assumes zero delay. Nevertheless, the experimental divergence converges closely to zero (See Figure 5.29). Generalisation Experiment/Task B

negative probability ρ_E, while others, like the high STDP learning rates η_0 and η_1 are simply outside the range of "reasonable" possibilities that were so determined through tests and the literature.

The high tolerance parameters explain the "clamped" values that resulted in some NES runs (see Tables 5.3, 5.6 and 5.14). According to NES, these values are unimportant in determining general fitness slopes in the vicinity of the search distribution.

Aside from the firing intensity parameters c_1 and c_2, which surprisingly have low tolerance in all of the experiments reported in this section,[20] other low-tolerance parameters include T in Table 5.16, b^3_-, w_- and T in Table 5.18, and w_-, T, η', ρ_{PP}

[20]With the exception of c_1 in the generalisability SPI Task A experiment.

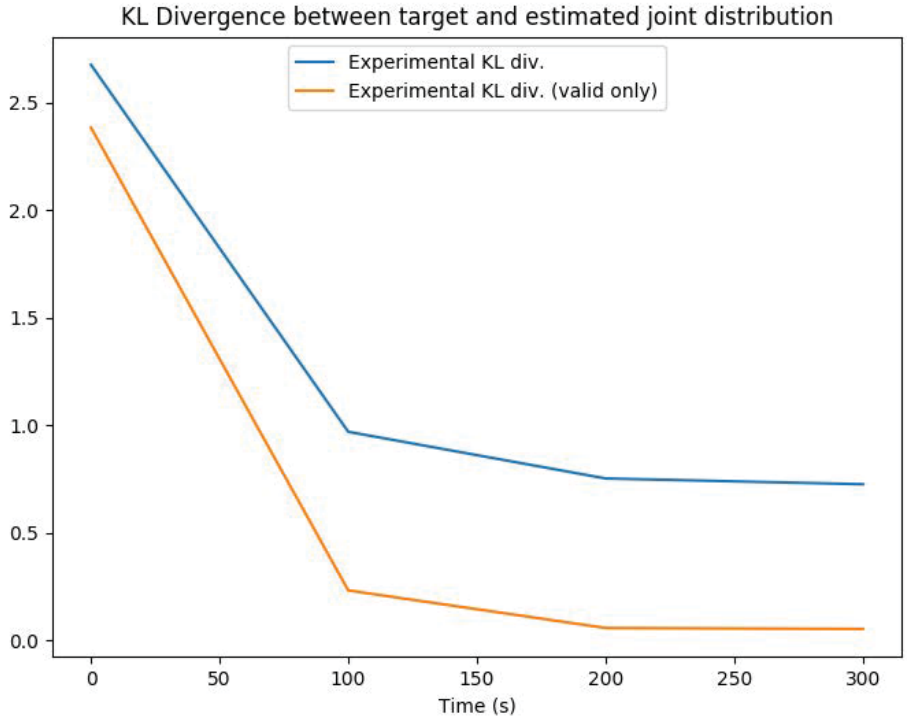

Figure 5.29: Experimental divergence of the SAM network whose analytic trace is shown in Figure 5.28—the valid-state divergence converges very nearly to 0. Experimental divergence could only be measured at well-spaced intervals, since it proved computationally inefficient to do otherwise. Generalisation Experiment/Task B

and w_{PI} in Table 5.19. All show tolerances orders of magnitude smaller than the initial search ranges given in Section 3.4. The fitness landscape is thus sensitive to perturbations of these hyperparameters.

Table 5.16: Sensitivity: baseline, Task A, SAM, 0.1 ms

	Mean	St.d.
b_-	-33.678	30.290
w_-	-4.535	0.030
T	0.476	0.003
R	1.503	0.819
η'_0	-0.182	0.075
η'_1	0.185	0.072
η_0	0.039	0.007
η_1	-0.014	0.006
c_1	0.411	0.045
c_2	0.695	0.017

Table 5.17: Sensitivity: generalisability, Task A, SAM, 0.1 ms

	Mean	St.d.
b_-	-46.089	91.648
w_-	7.827	8.149
T	1.165	0.466
R	0.420	0.365
η'_0	-0.057	0.015
η'_1	0.138	0.006
η_0	0.004	0.006
η_1	0.013	0.001
c_1	0.363	0.051
c_2	1.701	0.088

Table 5.18: Sensitivity: baseline, Task B, SAM, 0.1 ms

	Mean	St.d.
b^1_-	-8.287	13.549
b^2_-	11.516	16.582
b^3_-	-21.748	4.831
b^4_-	-11.497	25.590
w_-	-1.874	0.069
T	0.512	0.084
R	-1.179	1.438
η'_0	-0.037	1.585
η_0	0.022	0.008
η_1	-0.001	0.002
c_1	0.047	0.004
c_2	1.274	0.012

Table 5.19: Sensitivity: generalisability, Task B, SAM, 0.1 ms

	Mean	St.d.
b_-^1	-28.788	22.910
b_-^2	8.363	20.158
b_-^3	-21.351	15.598
b_-^4	-34.870	12.439
w_-	7.322	2.655
T	1.168	0.530
R	0.581	0.447
η_0'	-0.060	0.021
η_0	0.007	0.003
η_1	0.008	0.005
c_1	0.780	0.078
c_2	1.234	0.019

Table 5.20: Sensitivity: generalisability, Task A, SPI, 0.1–0.3 ms

	Mean	St.d.
b_-	-23.827	21.784
w_-	-3.316	0.333
T	0.607	0.038
R	1.239	0.247
η'	0.062	0.015
η_0	0.006	0.004
η_1	0.010	0.002
c_1	2.162	0.539
c_2	0.931	0.098
ρ_{PP}	0.516	0.026
ρ_E	-0.622	0.383
ρ_{PI}	0.584	0.167
ρ_I	0.751	0.641
ρ_{IP}	2.162	0.539
w_{MAX}	7.391	4.221
w_E	-5.876	5.240
w_{PI}	5.812	0.784
w_I	-2.703	8.219
w_{IP}	-7.917	2.876

6. Discussion

We can now interpret results in the light of our objectives and the evaluative strategies set out in Chapter 3. To this end, below is a summary of results that closely follows the structure of Section 4.3:

1. *Low delay SAM configurations outperform published results.* Section 5.1 shows how the baseline experiment in this work achieves results in both Task A and B that compete with, and in low delay configurations indeed outperform, the results published in Pecevski & Maass [43], even though the assumption of zero delay is not upheld in our work. Pecevski & Maass does not give numeric results, instead drawing a histogram of the learned distribution for Task A (Figure 5.3—compare with Figure 5.4, which reports our results), together with a plot of joint KL divergence for Task B (Figure 5.7) and a reconstruction of the sample Task B distribution (Figure 5.9—compare with Figure 5.8). For comparative purposes, our work also provides a table of valid-state divergences for Task B (Table 5.7), as well as an experimental plot of joint KL divergence that shows both whole-state divergence (performance) and valid-state divergence (Figure 5.12). In particular, the bold figures in Table 5.4 illustrate the configuration-delay combinations that are superior to those published in Pecevski & Maass. This positive result underscores the value of optimisation—as opposed to hand-crafting a model—even when operating at the disadvantage of deviation from theory.

2. *Both SAM and SPI generalise well on conditional distribution estimation tasks.* The second experiment tested the generalisability of both SAM and SPI architectures to unseen distributions of the same form at a fixed synaptic delay of 0.2 ms. SAM and SPI perform nearly equally on Task A problems—i.e., estimation of conditional distributions. Test performances are reported in 5.9. These figures are worse than those reported in the baseline experiment, but one needs to acknowledge that in that case the architectures were optimised for performance on a single target distribution, and as such, there was no test group of unseen distributions. In the generalisability experiment, optimisation is run on ten distributions, and testing on a further ten *unseen* distributions. Indeed, although the impact of moving to unseen distributions is evident (compare Table 5.8 and Table 5.9), on average the architectures perform well enough to demonstrate that they are in principle a feasible construct for Task A-type estimations using brain neuron models. Visually, the reconstructions of arbitrary target distributions confirm this (Figures 5.19 and 5.21).

3. *SAM generalises well on joint distribution estimation tasks, but SPI shows poor performance.* Generalisability on Task B-type problems—i.e. joint distribution estimation—is harder, since it involves multiple modules operating in tandem and therefore, a higher number of free variables together with simplifying global variables that have further reach. In this case, SAM performance is superior to SPI (see Table 5.13). SAM performs surprisingly well, with a best valid-state divergence ($\sim$0.11) on the test distributions that is only slightly less competitive than the hand-crafted model's baseline performance in Pecevski & Maass [43] ($\sim$0.1; this is reproduced in Figure 5.7). SPI's performance, although reasonably good on the optimisation distributions, does not generalise well to unseen distributions. Figures 5.26 and 5.27 compare the distribution estimates of SAM and SPI respectively on an arbitrary unseen target.

4. *Networks optimised for performance against single distributions do not generalise well.* As expected, networks from the baseline experiment do not generalise as well as networks optimised against multiple distributions, when tested on unseen distributions. This suggests that the hand-crafted model in Pecevski & Maass [43] cannot be expected to generalise to different distributions as well as general purpose, optimised networks.

5. *Biologically plausible properties can be integrated with little impact on conditional distribution estimation, but joint distribution estimation suffers.* SPI performs just as well as SAM on Task A-type problems (estimation of conditional distributions), as confirmed above. Thus, the biological properties mentioned in Section 3.3—soft inhibition, sparseness, recurrent connectivity and random synaptic delay—can all be in principle introduced in the manner described so that the resulting architecture is competitive in estimating unseen conditional distributions. However, the same cannot be said about joint distribution estimates: SPI performs poorly in Task B-type problems.

6. *SAM architectures exhibit a negative relationship between synaptic delay and estimation performance.* Tables 5.1 and 5.7 establish this interesting relationship: as synaptic delay increases, optimal KL divergence between target and estimate decreases. The result holds for both conditional (Task A) and joint (Task B) estimation tasks. Indeed, at low synaptic delays, optimal Task B valid-state divergence is very low, suggesting that hardware implementations can be built on its principles for low consumption applications to density estimation. The lowest delay tested in this work is 0.1 ms, safely above the order of delay necessary for hardware implementations.[1]

7. *Learning rates are not vitally important in determining fitness landscapes.* According to the NES search distributions, the final solutions are not very sensitive to changes in the learning rates, as long as learning occurs during the

[1] Which is on the order of microseconds.

training period. This may be partly a result of the highly restricted ranges of learning rates that are allowed in the beginning. However, solutions are very sensitive to firing intensities. See Section 5.3.

6.1 Effects of Non-Zero Synaptic Delay in SAM

Practically speaking, the results in Tables 5.1 and 5.7 clearly demonstrate that SAM estimates improve as synaptic delay decreases. The trend is visualised in Figure 6.1, which shows the optimal performances and valid-state divergences achieved in Tasks A and B respectively. Specifically, the figure shows that the *observed optimal* individuals exhibit the performance trend—whether the true optima express the same relationship is a different question. Notwithstanding that the observed trend might be a result of increased difficulty in finding the fitness optima as delays increase, theoretical reasoning suggests otherwise.

By the balance of gradual bias decay through intrinsic plasticity when an alpha neuron is inactive and discrete updates when it spikes, a SAM network can learn a basic marginal density $p(z = l; \theta)$ [43]—more frequently sampled values of z simply result in more frequent positive bias spikes in the corresponding alpha population, thereby causing a higher firing intensity via Equation 3.4. Together with the conditional density $p(\mathbf{x}|z; \theta)$ learned by STDP synapses (through extrinsic plasticity), the marginal density forms the generative distribution $p(\mathbf{x}, z; \theta) = p(\mathbf{x}|z; \theta) \cdot p(z; \theta)$, an estimate of the target distribution $p^*(\mathbf{x}, z)$. Thus, disabling or hampering either of the two mechanisms, both of which are affected by non-zero delay, is bound to impact learning ability.

The violation of zero delay has a significant effect because synaptic delay acts as a decoupling mechanism. In particular, if delay is set to a non-zero value d ms, it takes $2d$ ms for other neurons in the same alpha layer to receive the inhibiting signal after any alpha neuron fires,[2] and within that time period it may very well

[2]SAM is designed in such a way that any firing of an alpha neuron causes with high probability a firing of an inhibitory neuron, but it takes twice the delay period for the effect to make the

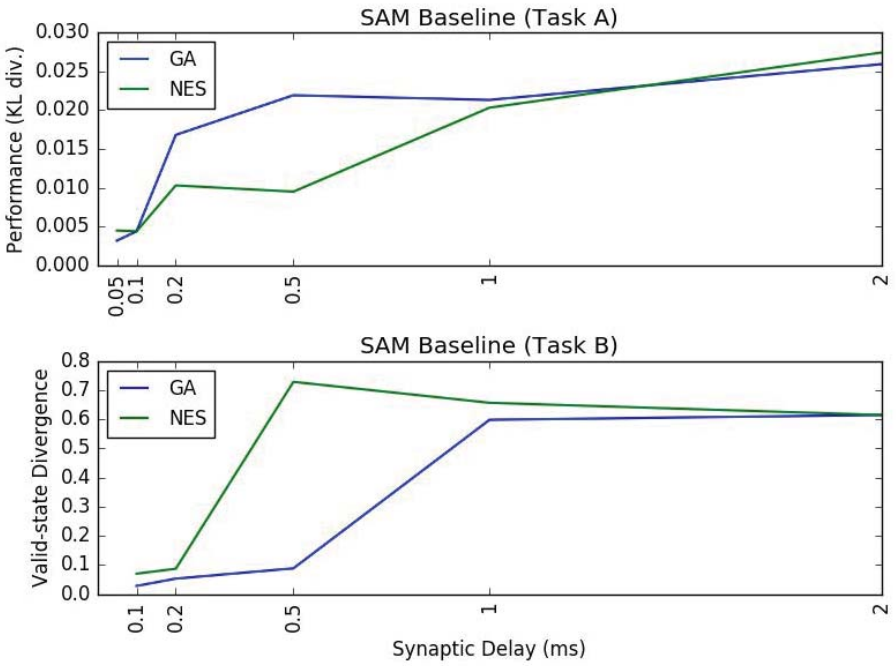

Figure 6.1: Relationship between synaptic delay and observed optimal performances in Task A (top) or valid-state divergences in Task B (bottom), for both GA and NES optima. There is a clear decrease in performance as synaptic delay increases.

happen that a second alpha neuron with a high spike probability fires before being inhibited by the first.

As such, a non-zero delay creates an isolating "horizon" within which a neuron's dynamics can proceed unaffected by activity within a short time window into the past of length $2d$ elsewhere within the network. This mechanism tends to generate simultaneous activity in alpha neurons, especially given the high initial firing intensities designed into the SAM architecture [43]. On the contrary, zero delay means that a firing alpha neuron directly causes inhibitory neurons to fire, which in turn cause immediate inhibition in co-members of the same alpha layer. The theoretical assumptions in Pecevski & Maass are then upheld, with STDP halting in neurons that are not encoding members of the right populations [43], while intrinsic bias decay proceeds unhindered. In the zero-delay scenario, therefore, ideal

round trip from one firing alpha neuron to the rest of the layer.

WTA behaviour in the alpha layer is entirely unaffected by changes in firing intensity, whether systematic—for example through changes in c_1—or temporal—as through increased synaptic input.

In more practical situations with non-zero delay, the firing intensity—and anything that affects it, systematically or otherwise—becomes relevant. According to [11], the probability of a single neuron firing within a time window $[t, t + \Delta t]$ is approximately given by

$$P_F(V_m) \approx 1 - \exp(-\Delta t \rho(V_m(t))), \tag{6.1}$$

where $\rho(t)$ is the firing intensity at t, given by Equation 3.4. Thus, the probability that *none* of n neurons fire within a time window defined by the round trip lag of $2d$, assuming an identical firing intensity ρ_0, is given by

$$S_n(\rho_0) \approx \exp(-2d\rho_0))^n. \tag{6.2}$$

At initial conditions, the probability of firing in SAM networks is designed to be high [43]. Table 3.3 gives an initial mean bias of 5, an initial mean weight of 1.333 (in Task B), and a PSP amplitude of 2 mV. Thus, given a SAM architecture with four alpha neurons, each with two active connections to the input layer, and assuming the scaling parameters from Pecevski & Maass [43] (i.e. $c_1 = 1/\tau$ and $c_2 = 1$), we can substitute these values in the membrane equation (Eqn. 2.2) and into the updated firing intensity equation (Eqn. 3.4), to give the probability that an excited neuron not in its refractory period survives a time window of $2d$ without firing:[3]

$$S(\rho_0) \approx \exp(-2d \cdot \frac{1}{\tau} \exp(2 \cdot 1.333 \cdot 2 + 5)). \tag{6.3}$$

For a synaptic delay of 0.1 ms, this gives a probability of ~ 0.664; the probability that none of the three supposedly inactive alpha neurons fire is therefore 0.29.

[3]Here we assume that each neuron is in its initial state, and that it receives spikes from two active input neurons.

This goes down to 0.002 at a delay of 0.5 ms, an unlikely possibility. Assuming the model in Pecevski & Maass with a small finite delay, therefore, it is exceedingly likely that some if not *all* neurons in the same alpha layer exhibit firing within close proximity. Since exclusive at-most-one behaviour is a core ingredient of the WTA-based theory behind the model in [43], simultaneous firings that subvert this competitive ideal render the learning algorithm inefficient at high firing intensities.[4]

Because of this not-quite-ideal WTA mechanism, in response to increased synaptic delay the mean firing intensity has to be decreased to reduce simultaneous activity and to retain some learning efficiency. However, decreasing the firing density introduces the possibility of long gaps in neuronal activity and generates zero states, as described in Section 4.2.5 and visualised below.

Figure 6.2 reproduces the spontaneous spiking activity from an example run of 500 ms of the best GA SAM individual after training on the visual perception task (Task B in the baseline experiment). It is clear from this activity that although most states are valid, it also exhibits two stereotypical states that are not:

- *Zero states*, corresponding to gaps longer than τ in the activity of output neurons from the same module—i.e., periods during which the value of some variable is not defined. In the spike plot, these are visible at the very beginning, for both variable y^1 (neuron indices 0, 1) and y^3 (neuron indices 4, 5).

- *"Invalid" states*, corresponding to time windows of τ within which there is more than one neuron from the pair that represents one variable firing. These are evident in the spike plot wherever two neurons from the same module fire within a very short time period—e.g. the second spike of neuron 0 in the raster plot nearly coincides with another spike from neuron 1. The theoretical model assumes that a spiking neuron sets the random variable it represents to a specific value for a time period τ, so these states are invalid

[4] Conversely, note that at zero delay, the probability of survival without firing is 1; this is the instantaneous probability of any process with discrete events in continuous time.

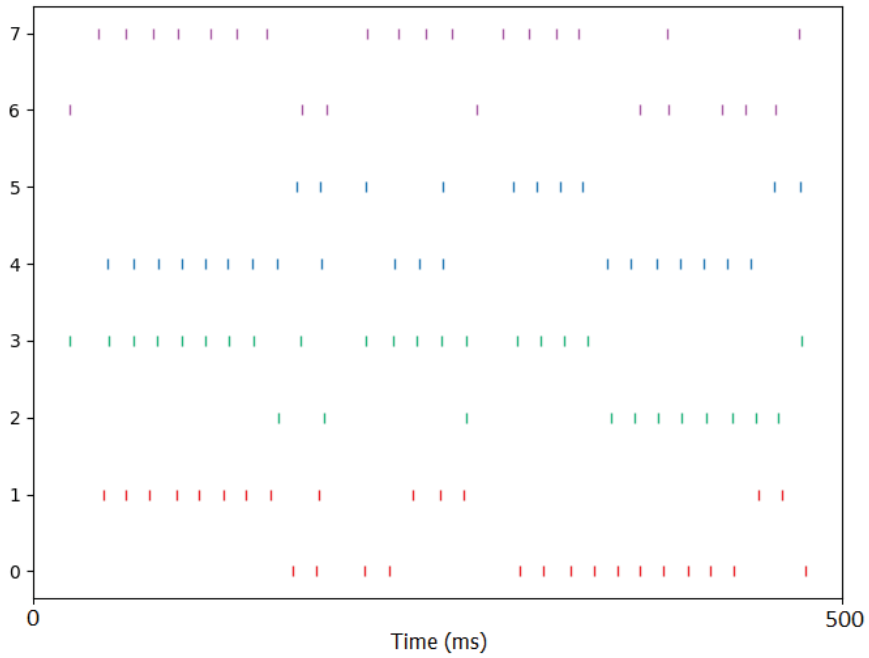

Figure 6.2: Spontaneous activity in the eight output neurons of the optimal SAM individual trained on the perception task (Task B in the baseline experiment). Neurons representing y^1 have indices 0 and 1 (red), those representing y^2 have indices 2 and 3 (green), and so forth.

under that interpretation because assignments cannot overlap. However, as we have seen in Section 4.2.5, the population coding interpretation can be reasonably relaxed so that whenever a time window with simultaneous activity occurs, the neuron to fire *first* is assumed to take precedence. This is also informed by biological findings that relative timing is informative about relative excitability.

Invalid and zero states, then, directly result from the introduction of non-zero delay, through balancing mechanisms.

Optimising for overall KL divergence—what we have also called "performance" or "whole-state divergence"; i.e., including the zero states in the estimated distribution—leads to a minimised KL divergence between the target distribution and the valid-state distribution, as can be seen in Figure 5.12. In this light, optimisation can be interpreted as a trade-off between the two mechanisms above: it balances

the mean firing density so that the number of zero states is not excessive, and at the same time it keeps simultaneous firings to a minimum, so that there is a happy medium within which valid states can be generated and learning maintained at an efficient level. Lowering the firing intensity so that there are too many zero states will directly result in a poorer whole-state divergence, since that metric uses all of the experimental states, valid or not. On the other hand, leaning towards too many simultaneous firings results in poor learning performance, and therefore a mismatch between the valid states of the estimate and target. This problem is avoided with zero delay, because in that case inhibition is instantaneous and increasing the firing probability—to eliminate zero states—cannot generate invalid states.

Naturally, it is going to be harder for the outer-loop optimiser to strike a good balance when delay values increase beyond a certain margin. At some point zero states will dominate, and very low firing intensities will harm learning efficiency in other ways. Further experimental evidence can be gleaned from the relationship of c_1 with delay, as found in the optimal parameters of baseline experiment Task A (Tables 5.2 and 5.3). c_1 governs the firing probability intensity in Equation 3.4. Results show that it decreases with increasing delay, and as it decreases, so does the firing probability in any given time interval. This counteracts simultaneous firing, attempting to find the right balance between too many spikes in close proximity and too many gaps in activity. Unfortunately, this evidence is only partial—Task B results show the *reverse* relationship, but this may be due to counterbalancing effects attributable to large interconnected networks.

6.2 Issues, Limitations and Comments

6.2.1 Simulator limitations

As we have seen, control over synaptic delay is an important aspect of our work. A basic limitation of NEST is that while it allows any finite delay to be used, it does not support zero delay. This makes it impossible to replicate the experiments

in Pecevski & Maass [43] exactly, in turn rendering it difficult to precisely evaluate the effect of introducing non-zero delay. However, the relationship between finite delay and optimal performance is well established in the work above, supporting the view that closer approaches towards theoretical conditions reap strong benefits.

6.2.2 Interpretation of network states

As discussed in Section 2.5, neural coding is not a solved problem, nor even expected to be uniform throughout the natural brain. Thus, various encoding schemes have been studied, with the population coding implemented in Pecevski & Maass [43] being merely one among many. To complicate matters further, there is significant leeway in the specifics that might appear confusing to the novice: Pecevski & Maass [43], for instance, suggests at one point using the most recent spike to assign variable values, apparently in contradiction with the injunction to assign variables for periods of a fixed length. In our work, we have used the latter scheme since it appears to be supported by the bulk of the text, with modifications as described in Section 4.2.5 to make this scheme viable in the finite delay context.

6.2.3 Use of valid-state divergence

At several points in this work, valid-state divergence as opposed to whole-state divergence is used to compare estimate quality,[5] while the latter is used as fitness for optimisation. As explained, this is necessary because of the low firing intensities that are introduced when using non-zero delay (see Section 6.1). Note that valid-state divergence is obtained by simple truncation of zero states in the estimated distribution—which are meaningless in any case—before calculating the KL value. This is a consistent, systematic procedure, and not in any way arbitrary or selective. In addition to the better results it gives (see Sections 5.1.2 and 5.2.2), it may also be sensible from a neuroscientific perspective because biological brains lack WTA-like

[5] See definition in Section 4.2.5.

rigidity.

6.2.4 GA vs NES

Although GA and NES share the evolutionary nomenclature, there are conceptual differences that are relevant to the results of this study. GA optimises individual fitness, and in so doing it can easily be fooled by stochastic measurements: in GA, a good individual tends to be propagated to the next generation with high probability, so that if fitness measurements are subject to noise, "unlikely but good" measurements of "bad solutions" will dominate the gene pool of subsequent generations, resulting in poor exploitation of the search space and a variation on the textbook case of getting stuck in a local minimum—in this case, the algorithm gets stuck in an unrepresentative stochastic minimum. This is somewhat mitigated by running multiple evaluations of fitness on each individual, at a price in terms of computation time. Nevertheless, we are chiefly interested in mean performance, which makes this a reasonable venture.[6] In contrast, NES optimises the fitness of a *distribution* of solutions, paying less attention to individual fitnesses and therefore successfully dealing with stochastic outliers. However, this could cause it to overlook optima in "narrow" fitness landscapes,[7] where the distribution might be dominated by bad fitnesses of individuals further from the optimum. These considerations explain some of the discrepancies between GA and NES results.

6.2.5 Training time vs accuracy trade-off

The experiments employed 96-core machines on Google's Cloud Platform, running for hours or days on end.[8] This left little time for adjustment, and few high-level parameter combinations could be tested within a comfortable time frame. A

[6]The *spread* of performance results is also of interest, naturally. However, it has not been studied here.

[7]For fairness' sake, it must be noted that all optimisation techniques suffer from the "needle-in-a-haystack" problem, including GA. See [13].

[8]See Section 4.2 for single core running times.

more exhaustive approach would have demanded a significantly increased budget for computing resources, or more time. Moreover, as already explained in the results, some experiments would have benefited from longer iterations. This would have been infeasible given the present time and resource limitations. Additionally, measurements with a larger number of repetitions could have provided statistically stronger comparisons, given that many of the measured metrics are subject to stochastic variability. These limitations explain the high learning rates used in NES (0.5 and 1): during preliminary runs it was observed that low learning rates (0.01–0.1) resulted in very slow convergence, compared to GA. This is supported by the literature (e.g. [54] and [50]), which reports several thousand iterations on typical high-dimensional problems.

6.2.6 Supervised vs unsupervised learning

A question that deserves some comment is whether this work deals with supervised, rather than unsupervised learning. The appearance of "labelled" output during training might give this misleading impression. However, a short argument shows that this suggestion is mistaken: in supervised classification, output is labelled with the ground truth—what we finally wish to predict with the model. In our case, the output labels that we provide are not what we wish to predict. We wish to predict their *distribution*: if our model is learning the distribution $p^*(z|\mathbf{x})$, for instance, we provide it with output samples $z \sim p^*(z|\mathbf{x} = \mathbf{X})$ for some specific input $\mathbf{X}$, and not $p^*(z|\mathbf{x} = \mathbf{X})$ itself. The fundamental task here is density estimation, which is in any case generally accepted to be an unsupervised type of machine learning.

6.2.7 Plausibility of parameter tuning

It is a philosophy of this work that the number of hyperparameters be kept to a minimum (see Section 3.4), and to use global or functional-group parameters as opposed to fine-grained neuron-by-neuron optimisation. Nevertheless, along

with Pecevski & Maass [43] it tunes some parameters to compensate for target statistics, such as module-specific weight baselines. To the author's knowledge, a tuning mechanism that could effect this in biological circuitry is as yet unknown, and remains an open question.

6.2.8 Scientific value of neural models

Aside from the benefit of exploring biologically plausible neural networks on the field of AI, neural models are also of relevance to neuroscience. Indeed, the subject straddles the two fields, with the consequence that it is sometimes less than clear where the significance of results lies: the reader often has to stop and ask whether some observation, assumption or conclusion is about biological fact or some machine learning model thereof. Reported results may also be of relevance only with respect to the model itself—a model with so-and-so assumptions exhibits such-and-such behaviour—and it may yet be unknown what the consequences are on either of the fields. Nevertheless, this work explores a clear machine learning domain—density estimation—within a biologically-inspired background. As such, the work presented offers an alternative to conventional density estimation techniques such as parameter estimation in Bayesian networks, detailed in [25], aside from pushing the boundaries of biological plausibility.

7. Conclusion and Future Work

Broadly speaking, this work improved upon the results in Pecevski & Maass [43], both those specific to the tasks that were demonstrated therein, as well as conceptually by proposing SPI, a novel architecture that integrates properties well-understood to be present in biological implementations with partial success vis-à-vis its conditional estimation performance. Although the new architecture showed poor generalisability on joint distribution problems, its performance on conditional distributions was competitive with SAM's. In the case of SAM, it was demonstrated that even though theoretical assumptions were not met by the simulation conditions,[1] practical performance after optimisation at low synaptic delays matched and in fact exceeded that of the hand-crafted model in Pecevski & Maass [43]. Generalisable SAM performance on unseen distributions, while slightly inferior to that on training distributions, was also competitive with the *single* distribution tests in Pecevski & Maass [43], which were performed in a far more favourable environment and are therefore a tough standard to compare with. These results satisfy the chief objectives of this work.

This work also explored an important aspect of deviation from theory—the relationship between synaptic delay and estimation performance. It was found that in SAM, performance declines with increasing delay, because delay introduces a decoupling mechanism that prevents inhibition from being completely exclusive to

[1] Indeed, *could not* be met, because of NEST limitations.

further activity in SAM hidden layers. This could be interpreted as further evidence that in biological circuits, which typically have delays exceeding the lowest such values tested in this work [20], a different type of inhibitory mechanism is employed, such as the divisive mechanism recently explored in work including [18] and [29]. In any case, non-zero delay has been found to be beneficial in structuring neural circuitry [2], which makes a strong case for understanding the effects of delay.

One broad limitation of this work is that SPI was not thoroughly explored. Future work could focus on the relationship of its performance with synaptic delay. During anecdotal runs, it was observed that SPI does not suffer as severe a hit as SAM when delay is increased. This would obviously have to be established by thorough experimentation, but its veracity would have consequences: as synaptic delay increased, SPI would increasingly outperform SAM, in turn implying that SPI might be a better model of some brain computations than SAM. A reason that possibly explains this changed relationship with synaptic delay is that the model of inhibition in SPI has a different dynamics than SAM's WTA-like inhibition, which is reliant on exclusive competition. Unfortunately, this could not be closely examined due to time restrictions.

Future work can look to firmly place the generalisability of neural networks in an analytic framework that directly involves delay as an integral component, since this work only examined the experimental aspect of the relationship with delay, giving heuristic justification of observed deviations based on theory.

A close sibling to this work's domain of contribution, the general task in the field of conventional PGMs referred to as structure learning aims to discover network *structure* itself, rather than the parameters only. Since this work was limited to two structural variations with different parameters, future approaches could examine the general structure learning task in biological circuitry, if it is found that there are neural correlates to this problem.

Finally, future experiments that can exploit the power of bigger computing clusters would benefit from finer learning rates and extensive outer-loop parameter

exploration. Some of the decisions made in this work were a practical compromise forced by resource limitations, and some possibilities were left unexamined. Shorter experimental turnover times would greatly benefit the researcher, who could then quickly fine-tune optimiser configurations and follow more promising directions.

8. Supervision

This work was supervised by Dr George Azzopardi from the University of Malta and Prof. Wolfgang Maass from Graz University of Technology, Graz, Austria.

References

[1] A visual guide to evolution strategies. `http://blog.otoro.net/2017/10/29/visual-evolution-strategies/`. Accessed: 2018-07-10.

[2] M. M. Asl, A. Valizadeh, and P. A. Tass. Dendritic and axonal propagation delays determine emergent structures of neuronal networks with plastic synapses. *Scientific Reports*, 7:39682, 2017.

[3] M. Avermann, C. Tomm, C. Mateo, W. Gerstner, and C. C. Petersen. Microcircuits of excitatory and inhibitory neurons in layer 2/3 of mouse barrel cortex. *Journal of Neurophysiology*, 107(11):3116–3134, 2012.

[4] G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass. Long short-term memory and learning-to-learn in networks of spiking neurons. *arXiv preprint arXiv:1803.09574*, 2018.

[5] H. Beyer. Evolution strategies. *Scholarpedia*, 2(8):1965, 2007. revision #130731.

[6] S. M. Bohte. The evidence for neural information processing with precise spike-times: A survey. *Natural Computing*, 3(2):195–206, 2004.

[7] L. Buesing, J. Bill, B. Nessler, and W. Maass. Neural dynamics as sampling: a model for stochastic computation in recurrent networks of spiking neurons. *PLoS Computational Biology*, 7(11):e1002211, 2011.

[8] M. B. Christopher. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, 2016.

[9] S. K. Esser, P. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. Barch, C. di Nolfo, P. Datta, A. Amir, B. Taba, M. Flickner, and D. S. Modha. Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the National Academy of Sciences of the United States of America*, 113 41:11441–11446, 2016.

[10] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana. The spinnaker project. *Proceedings of the IEEE*, 102(5):652–665, 2014.

[11] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski. *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press, 2014.

[12] S. Ghosh-Dastidar and H. Adeli. Spiking neural networks. *International Journal of Neural Systems*, 19(04):295–308, 2009.

[13] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[14] S. Habenschuss, Z. Jonke, and W. Maass. Stochastic computations in cortical microcircuit models. *PLoS Computational Biology*, 9(11):e1003311, 2013.

[15] E. M. Izhikevich. *Dynamical Systems in Neuroscience*. MIT Press, 2007.

[16] R. Jolivet, A. Rauch, H.-R. Lüscher, and W. Gerstner. Predicting spike timing of neocortical pyramidal neurons by simple threshold models. *Journal of Computational Neuroscience*, 21(1):35–49, 2006.

[17] Z. Jonke, S. Habenschuss, and W. Maass. Solving constraint satisfaction problems with networks of spiking neurons. *Frontiers in neuroscience*, 10:118, 2016.

[18] Z. Jonke, R. Legenstein, S. Habenschuss, and W. Maass. Feedback inhibition shapes emergent computational properties of cortical microcircuit motifs. *Journal of Neuroscience*, 37(35):8511–8523, 2017.

[19] W. Kahan. Ieee standard 754 for binary floating-point arithmetic. *Lecture Notes on the Status of IEEE*, 754(94720-1776):11, 1996.

[20] E. R. Kandel, J. H. Schwartz, T. M. Jessell, S. A. Siegelbaum, A. J. Hudspeth, et al. *Principles of Neural Science*, volume 5. McGraw-Hill New York, 2013.

[21] D. Kappel, S. Habenschuss, R. Legenstein, and W. Maass. Network plasticity as bayesian inference. *PLoS Computational Biology*, 11(11):e1004485, 2015.

[22] D. Kappel, R. Legenstein, S. Habenschuss, M. Hsieh, and W. Maass. A dynamic connectome supports the emergence of stable computational function of neural circuits through reward-based learning. *eNeuro*, 5(2):ENEURO–0301, 2018.

[23] A. Kasiński and F. Ponulak. Comparison of supervised learning methods for spike time coding in spiking neural networks. *International Journal of Applied Mathematics and Computer Science*, 16:101–113, 2006.

[24] D. C. Knill and D. Kersten. Apparent surface curvature affects lightness perception. *Nature*, 351(6323):228, 1991.

[25] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT press, 2009.

[26] S. Kunkel et al. Nest 2.12.0. zenodo. 10.5281/zenodo.259534, Mar. 2017.

[27] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, pages 1–101, 2016.

[28] J. H. Lee, T. Delbruck, and M. Pfeiffer. Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience*, 10, 2016.

[29] R. Legenstein, Z. Jonke, S. Habenschuss, and W. Maass. A probabilistic model for learning in cortical microcircuit motifs with data-based divisive inhibition. *arXiv preprint arXiv:1707.05182*, 2017.

[30] R. Legenstein and W. Maass. Ensembles of spiking neurons with noise support optimal probabilistic inference in a dynamically changing environment. *PLoS Computational Biology*, 10(10):e1003859, 2014.

[31] J. Lehman and R. Miikkulainen. Neuroevolution. *Scholarpedia*, 8(6):30977, 2013. revision #133684.

[32] W. Maass. Networks of spiking neurons: the third generation of neural network models. *Neural Networks*, 10(9):1659–1671, 1997.

[33] W. Maass. On the computational power of winner-take-all. *Neural computation*, 12(11):2519–2535, 2000.

[34] W. Maass. Noise as a resource for computation and learning in networks of spiking neurons. *Proceedings of the IEEE*, 102(5):860–880, 2014.

[35] D. Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. WH Freeman, 1982.

[36] U. Mehboob, J. Qadir, S. Ali, and A. Vasilakos. Genetic algorithms in wireless networking: techniques, applications, and issues. *Soft Computing*, 20(6):2467–2501, 2016.

[37] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[38] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT press, 1998.

[39] B. Nessler, M. Pfeiffer, L. Buesing, and W. Maass. Bayesian computation emerges in generic cortical microcircuits through spike-timing-dependent plasticity. *PLoS Computational Biology*, 9(4):e1003037, 2013.

[40] E. Niebur. Neuronal cable theory. *Scholarpedia*, 3(5):2674, 2008. revision #184781.

[41] M. Oster, R. Douglas, and S.-C. Liu. Computation with spikes in a winner-take-all network. *Neural Computation*, 21(9):2437–2465, 2009.

[42] D. Pecevski, D. Kappel, and Z. Jonke. Nevesim: event-driven neural simulation framework with a python interface. *Frontiers in Neuroinformatics*, 8, 2014.

[43] D. Pecevski and W. Maass. Learning probabilistic inference through spike-timing-dependent plasticity. *eNeuro*, 3(2):ENEURO–0048, 2016.

[44] F. Rieke, D. Warland, and v. S. de Ruyter. *Spikes: Exploring the Neural Code*. MIT Press, Cambridge, Massachusetts, 1997.

[45] S. Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

[46] S. J. Russell and P. Norvig. *Artificial intelligence - A Modern Approach, 3rd Edition*. Prentice Hall, 2016.

[47] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[48] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212, 2009.

[49] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, 2017.

[50] Y. Sun, D. Wierstra, T. Schaul, and J. Schmidhuber. Efficient natural evolution strategies. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, pages 539–546. ACM, 2009.

[51] G. Venter. Review of optimization techniques. *Encyclopedia of Aerospace Engineering*, 2010.

[52] D. Verstraeten, B. Schrauwen, D. Stroobandt, and J. Van Campenhout. Isolated word recognition with the liquid state machine: a case study. *Information Processing Letters*, 95(6):521–528, 2005.

[53] J. Ward. *The Student's Guide to Cognitive Neuroscience*. Psychology Press, 2015.

[54] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber. Natural evolution strategies. *Journal of Machine Learning Research*, 15(1):949–980, 2014.

A. Code Repositories

Source code can be found at the author's GitHub profile, `http://github.com/krisdamato/`:

- Extensions to NEST can be found forked at `http://github.com/krisdamato/nest-simulator` from the official NEST repository at `http://github.com/nest/nest-simulator`
- Evolutionary algorithm extensions to the LTL package can be found forked at `http://github.com/krisdamato/LTL` from the original repository at `http://github.com/IGITUgraz/LTL`
- Simulation and result processing scripts can be found at `http://github.com/krisdamato/LTL-SAM`

A ready-to-run Docker image, complete with all of the simulation architecture and Python 3.5, can be found at `https://hub.docker.com/r/krisdamato/spikes/`.

B. Acronyms

ANN: Artificial Neural Network

CMA-ES: Covariance Matrix Adaptation-Evolutionary Strategy

DNN: Deep Neural Network

EA: Evolutionary Algorithm

ES: Evolutionary Strategy

GA: Genetic Algorithm

HMM: Hidden Markov Model

KL: Kullback-Leibler

LIF: Leaky Integrate-and-Fire

LSM: Liquid State Machine

LTL: Learning-To-Learn

NES: Natural Evolution Strategy

PGM: Probabilistic Graphical Model

PSP: Post-synaptic Potential

SAM: Stochastic Association Module

SNN: Spiking Neural Network

SPI: Sparse Probabilistic Inference

STDP: Spike-Timing-Dependent Plasticity

WTA: Winner-Take-All