Dissertation
submitted to the
Combined Faculty of Natural Sciences and Mathematics
of Heidelberg University, Germany
for the degree of
Doctor of Natural Sciences

Put forward by

Christian-Gernot, Pehle

born in: Darmstadt

Oral examination: 10.02.2021

Adjoint Equations of Spiking Neural Networks

Referees:

Dr. Johannes Schemmel
Prof. Christof Wetterich

This dissertation is concerned with gradient-based optimization or "learning" in spiking neural networks and their applications. Based on a method well known in the optimal control literature, I derive a novel algorithm "EventProp," which computes exact gradients for arbitrary loss functions and allows for the optimization of spiking point neural networks. In the special case of time-invariant linear systems with jumps, this suggests an exact integration algorithm. Based on the same starting point, it is also possible to derive approximate online learning rules for spiking point neurons. More broadly, the adjoint method with jumps can be applied to structured neurons and other dynamical nets. In the case of structured neurons, the adjoint equations couple so that they can be interpreted as being associated with the same physical structure. This partially resolves the weight transport problem. Finally, I turn to the question of how stochastic classical systems, such as networks of deterministic spiking neurons stimulated by Poissonian noise, can emulate properties of small quantum systems.

In dieser Dissertation beschäftige ich mit gradienten basierter Optimierung oder "Lernen" in Spikenden Neuronalen Netzen und deren Anwendung. Anfangend mit einer Methode die in der Kontrolltheorie Literatur wohlbekannt ist, leite ich einen neuartigen Algorithmus "EventProp" her, welcher exakte Gradienten für beliebige Zielfunktionen berechnet und es erlaubt Spikende Punkt Neuron Netze zu optimieren. In dem Spezialfall von zeitinvarianten linearen Systemen mit Sprüngen, erhält man so einen exakten Integrationsalgorithmus. Angefangen vom selben Ausgangspunkt kann man auch genäherte Online Lern Algorithmen herleiten. In einem weiteren Kontext lässt sich die selbe Methode auch auf ausgedehnte Neuronen und andere dynamische Netze anwenden. Im Fall von ausgedehnten Neuronen stellt sich heraus, dass die adjoint Dynamik sich mit der selben Struktur assoziieren lässt. Dadurch wird zum Teil das "weight transport" Problem gelöst. Schliesslich wende ich mich der Frage zu wie klassische stochastische Systeme, wie Netzwerke von deterministischen Spikenden Neuronen, welche von Poisson Noise stimuliert werden, Eigenschaften von kleinen Quantensystemen emulieren können.

Acknowledgements

Completion of this thesis would not have been possible without the support and encouragement of a lot of people. I want to thank Johannes Schemmel for advice and encouragement over the years. I want to thank Christof Wetterich for many interesting conversations and great collaboration in the last two years. Most of the numerical experiments presented in this thesis would not have been possible without the organizational skill of Sandra Diaz and the initiative of Wolfgang Maass. I want to thank the fellow hardware developers, Andreas Hartel, Andreas Grübel, Sebastian Billeaudelle, Gerd Kiene, Yannick Stradmann, Vitali Karasenko, Simon Friedmann and Mitja Kleider. It wasn't always fun, but the result is something we can be proud of. I want to thank Eric Müller for the many discussions over the years and for always having an open door. I would like to thank Sebastian Schmitt for advice and encouragement. Special Thanks to Korbinian Schreiber for sharing office space, lots of coffee breaks, listening to my most recent idea, and being a great friend. Thanks to Jens Pedersen for the awesome care-package, the great collaboration in the last 3 years, and for being a great friend. Thanks to Timo Wunderlich for working out with me, the very productive collaboration, introduction to weight lifting, and being a good friend. I want to thank Damir Vodenicarevic and Christopher Bennett for the intense conversations and fun we had in Capo Caccia, Paris and on your visit to Heidelberg.

Before his tragic death, Karlheinz Meier was very supportive of me and gave me a lot of opportunities. He was open and enthusiastic towards ideas I presented to him. His dedication and passion for his work will always be an inspiration to me.

Thanks to parents for giving me life, for teaching me dedication, curiosity for the world and always having my back. Finally, no one observed my progress closer and with more patience than Motek and Carmel, for that I will always be grateful.

Dedicated to Carmel and the Memory of Karlheinz Meier.

Contents

List of Tables

List of Figures

Listings

Chapter 1

Introduction

Understanding computational principles underlying intelligence independent of the biological examples we are familiar with has been a research goal since the advent of universal computers in the first half of the 20th century. Indeed early work by pioneers such as McCulloch and Pitts [110], and von Neumann [157] used language and metaphors based on the nervous system in some of their work on computation. An important first question to answer is then how to judge intelligence independent of the computational realization [153]. Another closely related one is how intelligence can arise from inanimate matter in the first place. Here the basic insight is that any intelligent machine would have "structure" and "dynamics." While the structure would be largely static over the machine's existence, the dynamics would need to implement the learning process. In close analogy to nature, the machine's structure would be encoded by "hereditary" material, and the success of learning determines whether this structure gets replicated into the future. The use of directed evolutionary algorithms to create "infant" learners was already suggested in Alan Turing's work [153] (p. 456)

> "Instead of trying to produce a programme to simulate the adult mind, why not rather try to produce one which simulates the child's? [...]
>
> There is an obvious connection between this process and evolution, by the identifications
>
> - Structure of the child machine = hereditary material
> - Changes of the child machine = mutation,
> - Natural selection = judgment of the experimenter
>
> [...]"

A driving force for this thesis then has been to describe a general way of describing learning or self-optimizing systems. Such a general description should ideally be flexible enough to encompass artificial neural networks, open dynamical systems and hybrid open dynamical systems.

In chapter 2, I introduce the notion of a *Neural Processing Element* (NPE). The basic intuition behind them is to define a notion of a self-optimizing processing element that can be composed with other such processing elements in a

well-defined manner. Importantly they are defined in such a way that they can have a simple physical realization. From a programming language perspective, one can regard such processing elements as terms in a programming language. By introducing lambda abstraction, repetitive substructures can be abstracted and lead to the identification:

"structure of the machine" = "term in a certain lambda calculus".

To generate examples of Neural Processing Elements, I then introduce methods well known in the control theory literature: *adjoint and forward sensitivity analysis*. The basic idea is that it is possible to associate to any open dynamical system or system of equations an "adjoint" system that allows computing gradients with respect to parameters and initial conditions with respect to an arbitrary loss function. This even holds for hybrid systems that undergo discontinuous state transitions at certain hypersurfaces in state-space under certain natural conditions.

I give an introduction to this method and derive explicit equations for the jumps in the adjoint state variables in various special cases. I also define (tentatively) a notion of Neural (Event) Processing Element.

One main result of this thesis is then the derivation of explicit adjoint (and forward) sensitivity equations for neuron models relevant to computational neuroscience and neuromorphic engineering. In contrast to so-called surrogate gradient methods, the computed gradients agree exactly with numeric gradients computed by forward differences. In the particular case of leaky integrate and fire neurons with or without adaptive threshold, the resulting equations suggest an efficiently implementable algorithm *EventProp*, which only propagates error information at spike events. It subsumes several previously proposed algorithms that work with exact spike times.

I demonstrate the algorithm's effectiveness by applying it to several standard machine learning problems and find that it performs either better or equivalently to surrogate gradient based training. Since the algorithm admits a purely event-based implementation, it has memory advantages over surrogate gradients as well: It is only necessary to store data at events local to each computational unit (neuron). Moreover, the events during the backward pass can reuse routing information used during the forward pass. This implies it is possible to implement the EventProp algorithm on (digital) neuromorphic hardware, such as Loihi [44], Spinnaker [60], or Tianjic [125], efficiently. It also means it can be efficiently implemented on a hybrid digital-analog system like BrainScaleS-2.

In the case of linear time-invariant differential equations the adjoint equations with jumps take a particularly simple form, which I discuss in chapter 4. Forward and backward dynamics are coupled at jump times. Since quantum systems with time invariant Hamiltonian are one example of such differential equations this opens up interesting possibilities of studying classical systems coupled to quantum systems at events (measurements, photon detection, ...).

While the adjoint sensitivity method itself allows one to compute exact gradients, it suffers from the fact that it has to be computed in reverse time relative to the original differential equation. While the forward sensitivity equations don't suffer from this problem, a naive implementation scales as a product of dynamic variables and parameters. Recent work in the context of spiking neurons solved

this scalability problem for time discretized systems of equations with smoothed derivatives [15]. In chapter 5 I show that by an analogous process, but without resorting to discretization or smoothing of derivatives, it is possible to derive three-factor learning rules from the forward sensitivity equations derived in the preceding two chapters. Extensions to more general settings based on the forward sensitivity equations and hybrid forward adjoint schemes are left for future work. I also do not present any numerical experiments on the derived method.

Although (adaptive) LIF neuron models and the slightly more general AdEx neuron model are typically implemented to conduct network-level simulations of spiking neural networks, they only crudely approximate biological neurons and their bewildering diversity of signaling pathways and molecular mechanisms. A huge advantage of the adjoint method is the applicability to arbitrary complicated hybrid systems of ordinary differential equations. To illustrate this, I apply in chapter 6 the adjoint method to the multi-compartment neuron model as implemented in the BrainScaleS-2 neuromorphic chip and a model of CA3 hippocampal Pyramidal neurons. I also derive the general form of the adjoint equations for an arbitrary multi-compartment neuron model with any number of ion-channels and a very general synaptse model. The main insight here is that any such multi-compartment model couples the different compartments by a symmetric matrix of conductances. Geometrically it means that errors represented by the adjoint-state variables to the membrane voltage in each compartment can be thought to propagate on the same dendritic tree structure. This resolves in part the "weight transport problem" assuming that there is a way to signal from post- to pre-synaptic synapse. Moreover, at each synaptic density only the adjoint state variables of the pre-synaptic and post-synaptic density couple under natural assumptions on the jump and transition equations, this suggests the study of synaptic plasticity mechanism in this framework. In particular it should be possible to optimize the parameters entering into a plasticity mechanism in this way, leading to a very general notion of metaplasticity.

Roughly speaking, the complex biological processes at each synapse should implement synaptic plasticity and thereby learning in biological neurons. Abstracted from the details of complicated biochemistry so-called plasticity rules, such as STDP (spike time dependent plasticity) and others are used to model the plasticity of synapses in neuron models. In neuromorphic hardware, the possibility to implement such rules ranges fixed-function implementations of specific rules, to rules based on a limited set of micro-operations [44], to fully programmable plasticity as for example in the BrainScaleS-2 and Spinnaker system.

In the BrainScales-2 system, the plasticity processing unit's (PPU) role is to implement any such rule as efficiently as possible. As part of my work, I oversaw the successful scaling of the existing plasticity processing unit to the full sized single-chip system HICANN-X and changed the memory architecture to allow for the execution of instructions from external memory, as well as vector and scalar read and write access to external memory. Both changes taken together enable the implementation of plasticity algorithms that take advantage of large external memory, thereby significantly increasing the modeling capabilities of the system. Moreover, the change is also crucial for the analog inference accelerator mode as it allows software to be written without being constrained by the limited on-chip SRAM. I also implemented a fully parallel random number generator for the plasticity processor's vector unit to enable applications

such as random synaptic rewiring, noise terms in plasticity rules and stochastic rounding. I describe these changes in chapter 7.

One use case of the BrainScaleS-2 chip is stochastic inference with spiking neurons [114, 116, 30, 99, 22]. In chapter 8 I describe an extension to the BrainScaleS-2 chip, implemented by Gerd Kiene and jointly desinged by Gerd Kiene and me, which adds the ability to perform logical operations on the refractory state. I discuss several use cases for this extension and introduce a novel training method based on surrogate gradients [117]. Further experiments and evaluation of the hardware extension, which has been taped out with the single chip HICANN-X system are left for future work.

I then turn to a slightly more abstract question in chapter 9: There is a well known way to analyse stochastic optimal control problems by the so-called Hamilton-Jacobi-Bellmann equation. The key idea now is to regard the weight parameters in a (spiking) neural networks as dynamic quantities as well, which vary at a slower time scale. Changing the weight parameters based on the future expected reward can then be modeled as a control problem. Here I show that the weight parameters' optimal control is given by a gradient of the future expected reward under certain assumptions. As I indicated above models of biological neurons can also be seen as such parameterised dynamical systems. The assumed cost for the parameter update in a biological system (which corresponds to recruitment of addtional ion-channels etc.) most likely does not satisfy the assumptions on the cost of parameter updates. However this approach allows in principle to model these more complicated parameter update costs as well. Based on the seminal work [90, 91], Hamilton-Jacobi-Bellmann equations of the kind I just discussed admit a change of variables that linearize them and a path integral interpretation. Future work could specialise this observation to the case of optimal control of weight parameters.

In the final two chapters I turn to questions related to quantum emulation. In chapter 10 I demonstrate that small artificial neural networks can learn to emulate unitary transformations on quatum density matrices. Moreover, by interpreting the entries of certain real 8×8 as expectation values of classical spins, one can introduce a "quantumness" gate, which implements the constraints of a quantum density matrix on them. Based on the ideas first presented in [159] a natural second step is then to consider instead of just expectation values, expectation values and correlations of classical spins. This is done in chapter 11. I demonstrate numerically that the bit-quantum map [159] is complete for $Q = 2$ qubits and implement an explicit temporal correlation map from a system of spiking neurons to density matrices. Finally I show that the minimal bit-quantum map needs to be extended for $Q > 2$ and suggest situations in which only sparse information of all correlations would be necessary.

I conclude this work with a chapter 12 containing an outlook and conclusions. Many things I did work on in the last five years were not included in this thesis. Some of them because they turned out to be dead ends, some because of a lack of time. Nevertheless I believe there is a lot of exciting work to be done based on the ideas I have presented here. I therefore hope this is not just the end of a chapter but also the beginning of an exciting new one.

Chapter 2

Neural Processing Elements

The nervous system has besides its complicated dynamics a particular structure. The working hypothesis of *connectionism* is that how the primitive components used to model neurons are connected that is its structure already in large part determines the (potential) function. The question then becomes to what degree of fidelity the structure of the biological nervous system needs to be understood and captured in order to derive useful functional models. Arguably the most important characteristic of of biological nervous systems is their capability to "adapt" or "learn" beyond the innate function they had at birth.

In this chapter I describe a framework based on *category theory*, which allows for the description of self-optimizing "machines", which I call "Neural Processing Elements". Those machines can be composed and nested to form larger machines. The category theory perspective on "wiring diagrams" has been advanced in several papers [169, 103, 149]. In some ways the description I give here is just a different perspective on well known results in the literature, which I will point out along the way. Furthermore I will not actually give any formal defintions here. This would require the introduction of far more background material. Rather I include this chapter because the ideas I present here informed my thinking on the problem of self-optimizing systems.

While I want to consider a more general case later, let me first discuss artificial neural networks. In the case of artificial neural networks advances in their computational power significantly derived from structural innovations. While it is well known that a two layer artificial neural network is in principle already capable of approximating a large class of functions, in practice restrictions on connectivity and parameter sharing as in convolutional architectures, residual neural networks [78], as well as transformer and attention architectures [156], demonstrate the large impact that choice of structures have.

Moreover choice of structure or network architecture both has an impact on the feasability of solving a given task, as well as the convergence speed of the training algorithm. A basic insight is that the choice of structure can already ensure that a network can be trained to accomplish a wide range of domain specific tasks (vision, natural language processing, etc.).

The main enabling innovation in the case of artificial neural networks is the backpropagation algorithm [106, 107, 138]. Briefly speaking it answers the question how to *efficiently* compute derivatives of a function composed out of a set of primitive functions with respect to parameters, in the case that the

space of parameters has much higher dimensionality than the co-domain of the function. The basic insight is that this problem can be reduced to a two-phase message passing algorithm: By assumption the function can be computed by a directed acyclic graph whose nodes are primitives ϕ and edges are variables. During the *forward pass* values are propagated along these edges to ultimately compute the function, while simultaneously the input values to each node (or more generally a *context*) is recorded. During the *backward pass* cotangent vectors are propagated and at each node the *pullback* ϕ^* with respect to the given primitive at the input computed during the forward pass is computed. While this algorithm lends itself well to bulk-synchronous parallel implementations (c.f.[4] and references therein) it exhibits inherent forward-backward locking in that the backward phase depends on values computed during the forward phase.

From a programming language perspective a curious feature of artificial neural network models is that their architecture can typically be described very concisely in terms of few repeating and nested primitives, together with task specific parameters. This enables the development of hardware accelerators, which then can focus on implementing this short list of primitives in an efficient manner. Furthermore to support gradient based training one can associate to each primitive a way to compute the *pullback* between the output and input cotangent spaces, which in practice means that the number of supported operations has to be doubled. In other words operations come in pairs.

One such pair is addition and copying

$$+\colon \mathbf{R} \times \mathbf{R} \to \mathbf{R}, (x, y) \mapsto x + y, \tag{2.1}$$

$$\Delta\colon \mathbf{R} \to \mathbf{R} \times \mathbf{R}, dz \mapsto (dz, dz) \tag{2.2}$$

another is multiplication m and an operation m^*

$$m\colon \mathbf{R} \times \mathbf{R} \to \mathbf{R}, (x, y) \mapsto x \cdot y, \tag{2.3}$$

$$m^*\colon (\mathbf{R} \times \mathbf{R}) \times \mathbf{R} \to \mathbf{R} \times \mathbf{R}, ((x, y), dz) \mapsto (y \cdot dz, x \cdot dz). \tag{2.4}$$

Most generally to any smooth map between (pointed) manifolds

$$f\colon (M, p) \to (N, f(p)) \tag{2.5}$$

we can associate the *pullback* map between cotangent spaces

$$f^*\colon T^*_{f(p)}N \to T^*_p M. \tag{2.6}$$

This turns out to be a *contravariant* functor.

We now want to associate to each of the primitives ϕ a *processing element* $e(\phi)$ that computes ϕ and to each of the pullback primitives ϕ^* a *processing element* $e^* = e(\phi^*)$ that computes ϕ^*. What is meant concretely by "processing element" depends on further details. Let us say for example we wanted to implement primitives as digital circuits. To implement an addition primitive $+\colon \mathbf{R} \times \mathbf{R} \to \mathbf{R}$ for example a choice of value representation must be made, since digital circuits can't operate on real numbers and there are then still multiple options to implement the control and datapath of a digital adder. Other primitives like matrix multiplication could in principle be regarded as being composed of elementary operations, but the most efficient implementation of

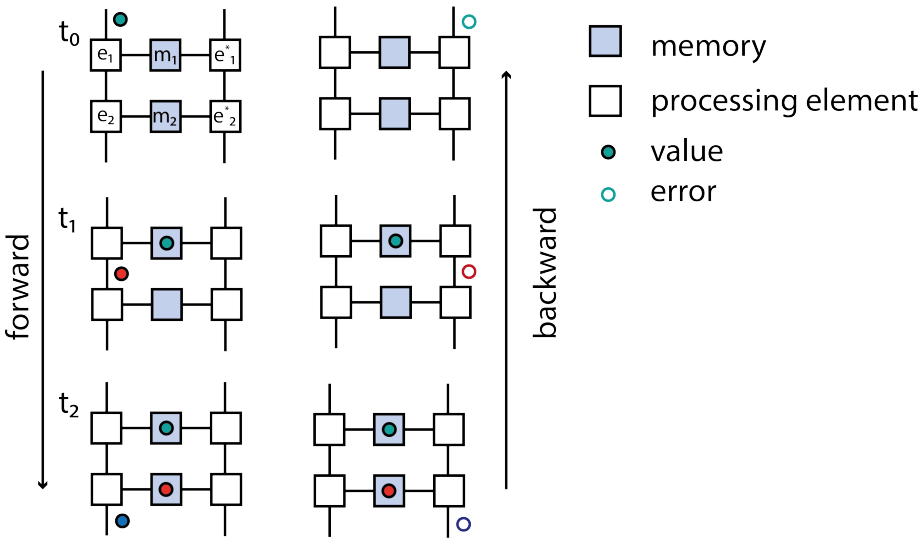


Figure 2.1: Composition of two processing elements e_1, e_2 computing primitives $\phi_1 \colon X \to Y$ and $\phi_2 \colon Y \to Z$ in sequence. Processing happens time steps, $t_0, t_1 = t_0 + \delta t, \ldots$. For simplicity we illustrate the case here, where every processing element takes one time step to compute its result. During the forward pass a message with value $x \in X$ (solid green circle) is passed into e_1 at t_0, in the next timestep t_1 it is stored in the associated memory element m_1 and an input message with value $y = \phi_1(x) \in Y$ (solid red circle) to e_2 is produced, moreover processing element e_1 does not accept new messages until the value in m_1 is consumed. The processing element e_2 stores this message at timestep t_2 in m_2 and produces an output message to the next processing elements with value $z = \phi_2(y) \in Z$ (solid blue circle). During the backward pass, messages are passed in reverse order through the directed acyclic graph. This happens because all processing elements wait for both a valid value in their associated memory element and a valid error message. For example element e_2^*, which computes the pullback ϕ_2^*, waits for an error message $dz \in T_z^*Z$ (open blue circle) and a valid value in the memory element m_2 before it produces an output error message with value $dy = (\phi_2^*)_y(dz) \in T_y^*Y$ (open red circle) passed on to e_1^*, which computes $dx = (\phi_1^*)_x(dy) \in T_x^*X$ (open green circle).

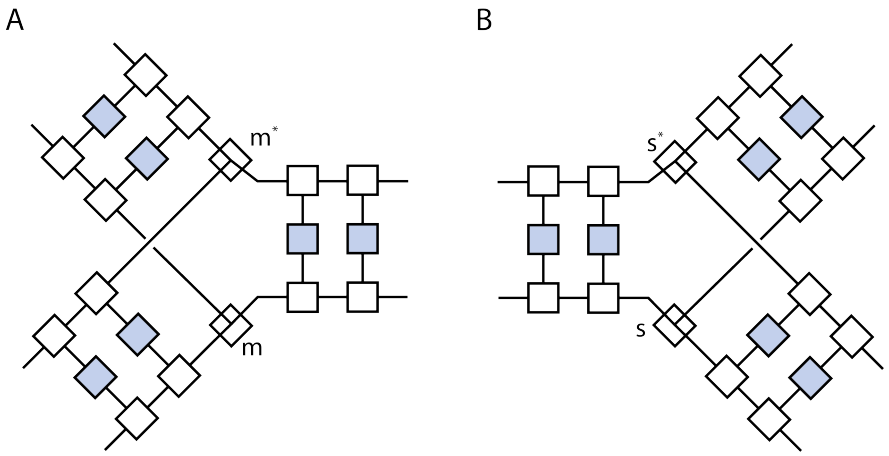


Figure 2.2: Illustration of merging (**A**) and splitting of (**B**) implemented by processing elements. The merge processing element $m_{k,l}$ implements the primitive $\phi\colon \mathbf{R}^k \times \mathbf{R}^l \to \mathbf{R}^{k+l}, (x, y) \mapsto [x_1, \ldots, x_k, y_1, \ldots, y_l]$ and the element $m^*_{l,k}$ implements the pullback $\phi^*\colon \mathbf{R}^{k+l} \to \mathbf{R}^k \times \mathbf{R}^l, [x_1, \ldots, x_k, y_1, \ldots, y_l] \mapsto (x, y)$. Similarly the split processing element $s_{k,l}$ implements the primitive $\psi\colon \mathbf{R}^{k+l} \to \mathbf{R}^k \times \mathbf{R}^l, [x_1, \ldots, x_k, y_1, \ldots, y_l] \mapsto (x, y)$. and the element $s^*_{l,k}$ implements the pullback $\psi^*\colon \mathbf{R}^k \times \mathbf{R}^l \to \mathbf{R}^{k+l}, (x, y) \mapsto [x_1, \ldots, x_k, y_1, \ldots, y_l]$.

the overall operation in realistic settings in many cases is not a composition of implementations of these more primitive operations. In many ways large parts of neumorphic engineering are concerned with coming up with novel ways of implementing matrix multiplication primitives in various physical inncarnations.

Once a choice of implementation is made we want to demand that there is a notion of composition of the implementation of primitives, which is compatible with the composition of the primitives. In the case of artificial neural network primitives this is illustrated in fig. 2.1. Each primitive is computed by a processing element, during the forward pass the input values need to be stored in a memory. Assuming the memory can hold only one value, the processing element has to be blocked until during the backward pass the computation of the pullback consumes the input value. As it turns out an analogous algorithm can be derived when each processing element operates in continuous time and operates either on events or continuous signals. I will explain this in chapter 3.

Informally what we are aiming for is a way of describing the hierachical composition of time-continuous or discrete time processes, which "learn" or self-optimise over time. We want to decompose the problem into two parts: A way to describe the structure of interconnected systems of elements, that is the legal "diagrams" without specifying their "dynamics" and then a way to associate a "function" or "dynamics" to a given diagram. These vague notions have been made precise in several instances by describing the "structure" of the system by a suitable *operad* and the "function" by an *operad algebra* [149]. There the brain is also explicitely mentioned as one example of a complex hierachically composed system. Running the program we sketched so far to completion would therefore mean to first desribe an *operad* of neural processing elements and then define an operad algebra which exhibits "self-optimization" or "learning", where again

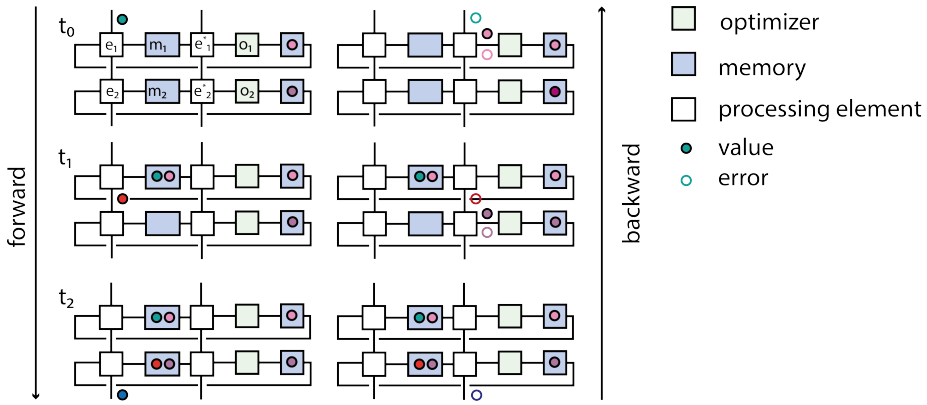


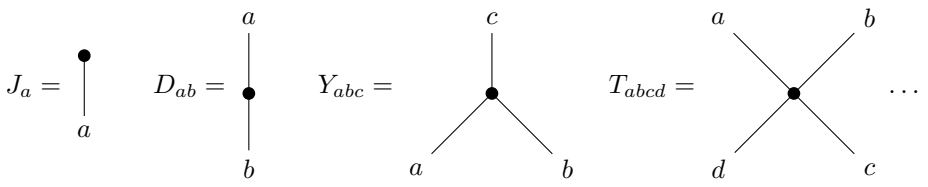
Figure 2.3: Sequential composition of "self-optimizing" or "neural" processing elements. The processing elements e_1, e_2 compute parameterised primitive functions $\phi_1 \colon X \times P_1 \to Y, \phi_2 \colon Y \times P_2 \to Z$. The parameter values p_1, p_2 (solid light and dark pink) are stored in additional parameter memories. During the *forward pass an input message with value $x \in X$ arrives for e_1 at time t_0. The processing element p_1 computes a message with value $y = \phi_1(x, p_1)$ (solid red circle) and stores the input and current parameter value (x, p_1) in memory m_1 at t_1. Similarly at time t_2 the processing element e_2 computes $z = \phi_2(y, p_2)$ and stores (y, p_2) in memory m_2. During the* backward *pass an error signal (open blue circle) arrives for processing element e_2^* at t_2, which implements the pullback ϕ_2^*. It computes $(dy, dp_2) = (\phi_2^*)_{(y,p_2)}(dz)$, a message with value $dy \in T_y^*Y$ (open red circle) is passed to e_1^* and a message with value $(p_2, dp_2) \in P_2 \times T_{p_2}^* P_2$ (solid dark pink and open dark pink circle) is passed to the optimizer o_2 at t_1. Finally at t_0 the optimizer o_2 computes a new value $p_2' = f_{p_2}(dp_2)$ and stores it in the parameter memory and the processing element e_1^* computes $(dx, dp_1) = (\phi_1^*)_{(x,p_1)}(dy)$ (open green and pink circles).*

these terms would need to be defined further.

In the context of artificial neural networks a category theoretic approach has been pursued by [55, 49]. Without an emphasis on learning, various "networks" of components have been described in this framework as well (Event based systems [169] , hybrid systems [103], open dynamical systems [155, 104, 55]). In the context of physics such wiring diagrams were of course first considered by Feynman to describe probability amplitutes in quantum mechanics [53] and later QED. Specifying Feynman rules corresponds to the specification of an operad algebra, indeed this algebraic view in the context of Feynman diagrams is well known [37, 11].

In the case of artificial neural networks we can depict primitive processing elements by diagrams $D_{ab}, Y_{abc}, \ldots$, where the multi-indices $a, b, c, \cdots$ are labels (from a computer science perspective "types", in the context of wiring diagrams this can be formalised by the notion of C-typed finite sets [155], with C some

monoidal category) which need to match for them to be composable.



(2.7)

Any other processing element is then a composition of primtive processing elements, which one can write

$$B_{dca} = Y_{dcb} D_{ba}, \tag{2.8}$$

where repeated indices mean that the corresponding wires are connected.

A concrete specification of the allowed abstract indices or types is dependent on the implementation of the processing elements. One example is for each index to be of the form

$$a(\tau) = ((\tau, +), (\text{req}, +), (\text{ack}, -)) \text{ or } \bar{a}(\tau) = ((\tau, -), (\text{req}, -), (\text{ack}, +)),$$

this would correspond to a situation, where processing elements are implemented by handshake circuits [68]. The signs indicate the directionality of three "wires" in the "bundles" $a(\tau)$ and $\bar{a}(\tau)$ and τ denotes an unspecified "value type". The request (req) and acknowledge (ack) wires implement the control between processing elements.

In such situations it is more convenient to track input and output indices separately by including a bar for output indices (say). The rule is then that two diagrams can be composed on matching input output index pairs

$$B_{\bar{d}ca} = Y_{\bar{d}cb} D_{\bar{b}a}. \tag{2.9}$$

A more complicated diagram like 2.3 could be also written in the abstract index notation, but just like with Feynman diagrams this would quickly become unwieldy, which is why in practice abstractions such as the ones that are present in libraries like PyTorch [122] are used. There are a few interesting things to notice in 2.3 though. First the row consisting of processing elements e_1, e_1^*, memories and optimiser o_1, say, can be abstracted as one diagram

$$N_{a(X)\bar{b}(X)\bar{c}(T^*X)d(T^*Y)} \tag{2.10}$$

which internally contains the optimization loop and parameter memory, where I indicated in parantheses the type of values the wires will carry. The trace which implements the optimisation could be explicitely written in terms of an index contraction over the corresponding primitive diagrams. A similar thing can be said about fig. 2.1. In particular fig. 2.1 is reminiscent of the matrix model double line notation first introduced by t'Hooft [82].

There are two additional conceptional advantages of the separation between structure and implementation, which I will now briefly discuss and elaborate elsewhere. As described for example in [48], based on earlier work by [100] any cartesian closed category has an internal logic based on lambda calculus.

This means that in a programming language which contains diagrams as primitives and allows their composition in the way I sketched above one can also introduce lambda abstraction:

$$\lambda \left(D_{c\bar{d}} \colon \; c \; \text{—} \bigcirc \text{—} \; \bar{d} \right) . Y_{d\bar{e}\bar{f}} D_{c\bar{d}} Z_{ab\bar{c}} \tag{2.11}$$

Concretely they can be thought of as diagrams with holes, into which diagrams with fitting wires can be pasted in. A hierachical structure can then be assembled by abstracting over several of these diagrams with holes. This view from the perspective of functional programming is of a particular practical use. Indeed various (domain specific) languages such as Zélus [26], Modia [20] make use of this compositionality.

The goal of the next chapter will then be to explain how in the case of spiking neural networks and more generally ordinary differential equation with jumps a similar prescription for parameter optimisation as in the case of artificial neural networks can be implemented, which will turn out to be composable in a similar way as indicated in figures 2.1, 2.3.

In chapter 6 I will then explain how this approach can be extended to structured neurons, which again admit composable equations which allow for parameter optimisation of synaptic weights and plasticity. A further step would be to fully regard a biological neuron as many nested biological machines, wich again could be subject to parameter optimisation, that is to model for instance the gene expression and active transport in individual cells as well.

Chapter 3

Adjoint Equations of Spiking Point Neurons

In this chapter I propose a new method that enables parameter optimization of spiking neural network models. By formulating the problem in a way that is well known in the optimal control literature [128, 137, 63, 38] and recently popularized in the machine-learning community [35, 88], I arrive at a set of ordinary differential equations with *jumps* of so-called *adjoint state variables*, which in turn can be used to compute gradients of the parameters with respect to a given loss functions. By implementing the method numerically, I can demonstrate, that it can be used to solve a set of standard machine learning benchmark tasks with comparable or better performance than currently used surrogate gradient methods [51, 171, 17, 147, 117]. Earlier work addressing the challenge of optimizing spiking neural networks or networks with binary activation functions includes [43, 39, 24].

While I evaluate the method here in the case of a simple point neuron model, it is in fact fully general and can in principle be applied to arbitrarily complex multi-compartment neuron models. I will derive explicit equations for a fairly general kind of multi-compartment neuron model in chapter 6. The applicability of the adjoint method to the optimization of non-spiking neuron models was already pointed out in [84], however there the spike discontinuity was avoided by replacing the delta distribution with a sharply peaked smooth function. The work also did not explicitly address reset of the membrane voltage and used different neuron models than we consider here.

In a recent work [34] the adjoint method with jumps was introduced to the machine learning community with several interesting experiments, but without reference to the original work by [137, 63]. Part of the content of this chapter is joint work with Timo Wunderlich [165]. Here I expand on the work reported there in several ways: Based on the derivation in [63], I derive explicit equations for jumps in the adjoint state variables both in the general case and in several special cases that are of interest. This allows me to introduce a general notion of Neural Event Processing Element, which specialises to all point neuron models and synaptic plasticity mechanisms known to me. It also clarifies that the essence of the EventProp algorithm as presented in [165] is that the system of ordinary differential equations under consideration decomposes into many smaller

subsystems (the "neurons") only coupled at events. The EventProp algorithm presents a significant computational advantage over surrogate gradient methods, since in the case of linear systems such as LIF neurons it only requires storage at spike events. Even for non-linear neuron equations the adjoint method is preferable as it does not rely on a discretisation of the dynamics as surrogate gradient methods do and can make use of established methods of numerically solving ODEs.

The experimental evaluation is done on the leaky-integrate and fire neuron model as in [165]. Besides the experiments presented there, I performed additional experiments on machine learning datasets. I find that the proposed method has, given a choice of network architecture and hyperparamters, similar or better performance compared to surrogate gradient backpropagation on the MNIST [102], Fashion-MNIST [166] and CIFAR-10 dataset [97]. I do not achieve state of the art performance in absolute terms, however. I also demonstate that the method can be applied to a simple regression and reinforcement learning task. I conclude with some indication how to extend and apply the adjoint method both in the context of neuromorphic hardware, different neuron models, and outline some further potential theoretical development of the adjoint method in the context of spiking neuron models.

3.1 Adjoint Equations

The general problem that I want to subsume our method of optimizing spiking neural networks under can be stated as follows: Given an action S, that is a functional of a path $x(t)$ and parameters p

$$S(x,p) = \int_0^T l(x,p,t)\mathrm{dt}$$

together with contraints

$$f(x,\dot{x},p,t) = 0, \tag{3.1}$$
$$h(x(0),p) = 0, \tag{3.2}$$

where f and h are differentiable functions, find a set of parameters p, such that S is minimized. In the case I am interested in, the dynamics of the system furthermore is allowed to undergo state transitions, when jump conditions are satisfied: That is we will in addition consider implicitly defined differentiable *jump conditions* $j(x^-,\dot{x}^-,p)$, which lead to a transition specified implicitely by differentiable *transition functions*

$$T(x^+,\dot{x}^+,x^-,\dot{x}^-,p).$$

Here x^- and x^+ denote the state before and after the transition, that is the left and right limit of the state variables in time.

Then the total derivative of the action functional S with respect to the parameters is given by

$$\frac{dS}{dp}(x,p) = \int_0^T \left[\partial_x l \frac{dx}{dp} + \partial_p l\right] \mathrm{dt}, \tag{3.3}$$

here and in the following I denote by $\partial_p x = \frac{\partial x}{\partial p}$ the partial derivative. One can introduce Lagrange multipliers $\lambda(t) \in \mathbf{R}^n, \mu \in \mathbf{R}^n$ to impose the constraints given by 3.1, 3.2 directly

$$\mathcal{S} = \int_0^T [l(x,p,t) + \langle \lambda(t), f(x,\dot{x},p,t)\rangle]\mathrm{dt} + \langle \mu, h(x(0),p)\rangle,$$

where $\langle v, w\rangle = v^T w$ denotes the dot product. Along trajectories $x(t)$, which satisfy the constraints $f = 0, h = 0$, these two actions agree and therefore also their parameter derivatives agree

$$\frac{d\mathcal{S}}{dp} = \frac{dS}{dp}. \tag{3.4}$$

The total derivative of the augmented action with respect to the parameters p is given by

$$\frac{d\mathcal{S}}{dp} = \int_0^T \left[\partial_x l \frac{dx}{dp} + \partial_p l + \langle \lambda, (\partial_x f \frac{dx}{dp} + \partial_{\dot{x}} f \frac{d\dot{x}}{dp} + \partial_p f)\rangle\right] \mathrm{dt} \tag{3.5}$$

$$+ \langle \mu, (\partial_{x(0)} h \frac{dx}{dp}(0) + \partial_p h)\rangle. \tag{3.6}$$

Integration by parts of the terms that contain $\frac{d\dot{x}}{dp}$ yields

$$\frac{d\mathcal{S}}{dp} = \int_0^T \left[\left(\partial_x l + \langle \lambda, (\partial_x f - \frac{d}{dt}\partial_{\dot{x}} f)\rangle - \langle \dot{\lambda}, \partial_{\dot{x}} f\rangle\right)\frac{dx}{dp} + \partial_p l + \langle \lambda, \partial_p f\rangle)\right] \mathrm{dt} \tag{3.7}$$

$$+ \langle \lambda, \partial_{\dot{x}} f\rangle \frac{dx}{dp}|_T + (-\langle \lambda, \partial_{\dot{x}} f\rangle + \langle \mu, \partial_{x(0)} h\rangle)|_0 \frac{dx}{dp}(0) + \langle \mu, \partial_p h\rangle \tag{3.8}$$

We can eliminate two of the boundary terms by imposing the boundary condition $\lambda(T) = 0$ and $\langle \lambda, \partial_{\dot{x}} f\rangle = \langle \mu, \partial_{x(0)} h\rangle|_0 d_p x(0)$. Finally to avoid calculating the forward sensitivities $s_x = \frac{dx}{dp}$ we can demand that

$$\partial_x l + \left\langle \lambda, \left(\partial_x f - \frac{d}{dt}\partial_{\dot{x}} f\right)\right\rangle - \langle \dot{\lambda}, \partial_{\dot{x}} f\rangle = 0. \tag{3.9}$$

Then parameter derivative of the action S with regard to the parameters p is therefore given by

$$\frac{dS}{dp} = \frac{d\mathcal{S}}{dp} = \int_0^T [\partial_p l + \langle \lambda, \partial_p f\rangle]\mathrm{dt} + \langle \lambda, \partial_{\dot{x}} h\rangle|_0 \partial_{x(0)} h^{-1} \partial_p h \tag{3.10}$$

In order to implement these equations numerically several techniques have been developed and implemented.

Generally speaking the solution is structured in three steps. First the original ordinary differential equation specified by f is solved from 0 to T. Then the adjoint equation (3.9) is solved from T to 0. Finally the parameter sensitivity of the action is computed by carrying out the integral (3.10). The second and third step can be combined into one numerical procedure. It should be noted that it can be advantageous to use the discrete adjoint equations associated to the discretization scheme used during forward integration (e.g. [174, 173]).

In the case that the loss function l depends on the values of x at discrete time points one can take those into account by considering the following loss function

$$l(x,p,t) = \sum_i \bar{l}(x,\tilde{x}_i)\delta(t-t_i) + \tilde{l}(x,p,t), \tag{3.11}$$

where t_i are chosen points in time independent of the parameters p. Such a loss introduces discontinuities into the implicit equation (3.9). The relevant term is

$$\partial_x l(x,p,t) = \sum_i \partial_x \bar{l}(x,\tilde{x}_i)\delta(t-t_i) + \partial_x \tilde{l}(x,p,t). \tag{3.12}$$

In the case that the ODE is in explicit form $f(x,\dot{x},p,t) = \dot{x} - \tilde{f}(x,p,t)$, this contribution to (3.9) introduces jumps by $\partial_x g(x,\tilde{x}_i)$ at times t_i. For example a loss function of the form

$$l(x,p,t) = \frac{1}{2}\sum_i |x-\tilde{x}_i|^2 \delta(t-t_i) \tag{3.13}$$

would demand that the state variable x is close to $\tilde{x}_i$ at time t_i. Resulting in a jump of λ by $|x-\tilde{x}_i|$ at time t_i.

3.2 Adjoint Equations of Spiking Neurons

So far we've assumed that the dynamics specified by $f(x,\dot{x},p,t)$ is continuous. I now want to consider a situation where instead the dynamics of the system can change based on events (externally induced state changes) or state events and in which the system in response either switches to a different mode or the state variables jump. Any classical physical system can of course not suddenly change its state in this way, so it is clear that such a situation is always an approximation to a situation where a continuous transition happens at a faster timescale (e.g. a bouncing ball). Nevertheless such a viewpoint is very useful for studying event based neuromorphic hardware and circuits, as there this approximation is also employed and one does explicitely not want to model the digital event propagation as a physical process. Similarly in neurobiology action potentials and neurotransmitter release are typically modelled by events.

To the best of our knowledge the derivation of forward sensitivities in this situation was first studied (in the case of explicitely specified transition and jump equations) in [137]. There equations for the *forward sensitivities* under discontinuous jumps were derived for explicit ordinary differential equations and time, velocity indepedent transitions. A recent survey with examples and setting that includes differential algebraic equations and memory is [38]. Detailed proofs of a generalisation of the approach in [137] can be found in [63]. I will adopt the notation of the latter here.

We want to study a situation in which a parametrised dynamical system can enter various modes $[S_1, S_2, \ldots, S_j, \ldots, S_{n_j}]$ at times $t_f^{(1)}, \ldots, t_f^{(n_j-1)}$. The time and the ordering of the occurence of the modes depends on the parameters. Each mode S_j is specified by explicit ordinary differential equations

$$\dot{x}^{(j)} = f^{(j)}(x^{(j)},p,t), f\colon \mathbf{R}^{N^{(j)}} \times \mathbf{R}^M \times \mathbf{R} \to \mathbf{R}^{N^{(j)}} \tag{3.14}$$

when the state trajectory $x^{(j)}(t,p)$ satisfies a jump condition

$$j(x^{(j)}, \dot{x}^{(j)}, p, t_f^{(j)}) = 0 \tag{3.15}$$

at time t_f the *mode* of the dynamics changes, that is the dynamics is now determined by

$$\dot{x}^{(j+1)} = f^{(j+1)}(x^{(j+1)}, p, t_0^{(j+1)}), f \colon \mathbf{R}^{N^{(j+1)}} \times \mathbf{R}^M \times \mathbf{R} \to \mathbf{R}^{N^{(j+1)}}$$

where the state and velocities before the transition $x^{(j)}, \dot{x}^{(j)}$ are related to the state and velocities after the transition $x^{(j+1)}, \dot{x}^{(j+1)}$ by a transition function

$$T^{(j)}(x^{(j)}, \dot{x}^{(j)}, x^{(j+1)}, \dot{x}^{(j+1)}, p, t_f^{(j)}) = 0. \tag{3.16}$$

Then the equation for the parameter sensitivity of the action 3.3 has to be modified by splitting up the integral at the transition times

$$d_p S = \sum_{j=1}^{n_j} \int_{t_0^{(j)}}^{t_f^{(j)}} \left[\partial_x l d_p x^{(j)} + \partial_p l\right] \mathrm{dt} + l(x^{(j)}, p, t_f^{(j)}) \frac{\mathrm{dt}^{(j)}}{dp} - l(x^{(j)}, p, t_0^{(j)}) \frac{dt^{(j-1)}}{dp} \tag{3.17}$$

Similarly the adjoint action can be split up at the transition times and the total derivative with respect to the parameters yields after integration by parts and use of the equation (3.9)

$$d_p \mathcal{S} = \sum_{j=1}^{n_j} \int_{t_0^{(j)}}^{t_f^{(j)}} \left[\partial_p l + \langle \lambda, \partial_p F^{(j)} \rangle\right] \mathrm{dt} \tag{3.18}$$

$$+ \sum_{j=1}^{n_j} \langle \lambda^{(j)}(t_f^{(j)}), \partial_{\dot{x}} F^{(j)}(t_f^{(j)}) \rangle d_p x^{(j)}(t_f) - \langle \lambda^{(j)}(t_0^{(j)}), \partial_{\dot{x}^{(j)}} F(t_0^{(j)}) \rangle d_p x^{(j)}(t_0^{(j)})$$

$$= \sum_{j=1}^{n_j} \int_{t_0^{(j)}}^{t_f^{(j)}} [\partial_p l + \lambda^T \partial_p F^{(j)}] \mathrm{dt} - \langle \lambda(t_f^{(n_j)}), d_p x(t_f^{(n_j)}) \rangle + \langle \lambda(t_0^{(0)}), d_p x(t_0^{(0)}) \rangle \tag{3.19}$$

$$+ \sum_{j=1}^{n_j - 1} (l^{(j)}(t_f^{(j)}) - l^{(j+1)}(t_f^{(j)})) \tau(t_f^{(j)})$$

$$+ \sum_{j=1}^{n_j - 1} \langle \lambda^{(j)}(t_f^{(j)}), \partial_{\dot{x}^{(j)}} F(t_f^{(j)}) \rangle d_p x^{(j)}(t_f^{(j)})$$

$$- \sum_{j=1}^{n_j - 1} \langle \lambda^{(j+1)}(t_f^{(j)}), \partial_{\dot{x}^{(j+1)}} F^{(j+1)}(t_f^{(j)}) \rangle d_p x^{(j+1)}(t_f^{(j)}), \tag{3.20}$$

where I've denoted by $F^{(j)} = \dot{x}^{(j)} - f^{(j)}(x^{(j)}, p, t)$. Just as before the goal is now to eliminate the dependence on the forward sensitivities $d_p x$. The following theorem gives sufficient conditions under which the forward sensitivities before and after a transition can be related.

Theorem 3.2.1 (Existence and Uniqueness of Sensitivity Functions [63])**.** *Assume that*

1. *For all* $t \in [t_0^{(j)}, t_f^{(j)}]$ *the partial derivatives*

$$\partial_x f^{(j)}, \partial_p f^{(j)} \tag{3.21}$$

exist and are continuous in a neighborhood of the solution $x^{(j)}(p,t)$.

2. *For all transition times*

$$t_f^{(j)}, j = 0, \ldots, n_j - 1$$

the following four equations define a continuosly differentiable map of an open set

$$h \colon U \subset \mathbf{R}^{2N^{(j)}+2N^{(j+1)}+1+M} \to \mathbf{R}^{2N^{(j)}+2N^{(j+1)}+1}$$

with

$$h(\dot{x}^-, x^-, \dot{x}^+, x^+, t_f^{(j)}; p) = 0$$

where h *is defined as*

$$x^-(t_f) - x^-(t_0) - \int_{t_0}^{t_f} f^-(x^-, p, t)\mathrm{dt} = 0 \tag{3.22}$$

$$j(\dot{x}^-, x^-, p, t_f) = 0 \tag{3.23}$$

$$T(\dot{x}^-, x^-, \dot{x}^+, x^+, p, t_f^{(j)}) = 0 \tag{3.24}$$

$$\dot{x}^+ - f^+(x^+, p, t_f^{(j)}) = 0 \tag{3.25}$$

with the abbreviations $x^- = x^{(j)}(t_f^{(j)}), x^+ = x^{(j+1)}(t_f^{(j)}), f^+ = f^{(j+1)}, f^- = f^{(j)}$, *etc. And assume that the submatrix formed by the columns of the Jacobian matrix corresponding to* $\dot{x}^-, x^-, x^+, t_f^{(j)}$ *is invertible.*

Then the partial derivatives $\partial_p x^{(j)}$ *exist for all* $j = 0, \ldots, n_j$ *are continuous and obey the differential equation*

$$\partial_t(\partial_p x_i^{(j)}) = \partial_{x^{(j)}} f_i^{(j)} \partial_p x^{(j)} + \partial_p f_i^{(j)} \tag{3.26}$$

in $(t_0^{(j)}, t_f^{(j)})$.

The paramter sensitivity of the transition time is given by

$$\frac{\mathrm{dt}}{\mathrm{dp}} = -\frac{\partial_{\dot{x}^-} j \partial_p f^- + \partial_x j d_p x^- + \partial_p j}{\partial_{\dot{x}^-} j \partial_t f^- + \partial_x j \dot{x}^- + \partial_t j} \tag{3.27}$$

and sensitivities before and after the jump are related by:

$$\begin{aligned} d_p x^+ = &- \left[f^+ + (\partial_{x^+} T)^{-1} \left(\partial_{\dot{x}^-} T \partial_t f^- + \partial_{x^-} T f^- + \partial_{\dot{x}^+} T \partial_t f^+ + \partial_t T \right) \right] \frac{\mathrm{dt}}{\mathrm{dp}} \\ &- (\partial_{x^+} T)^{-1} \left(\partial_{\dot{x}^-} T \partial_p f^- + \partial_{x^-} T \partial_p x^- + \partial_{\dot{x}^+} T \partial_p f^+ + \partial_p T \right) \end{aligned} \tag{3.28}$$

Proof. This follows essentially by the implicit function theorem and Gronwalls theorem [72], for details see [63]. □

Theorem (3.2.1) also allows us to relate the adjoint state variables as follows.

Theorem 3.2.2. *Given the same assumptions as 3.2.1, the jumps in the adjoint state variables at a given transition time $t_f^{(j)}$ are given by*

$$\lambda^- = \lambda^+ \left[CA - (\partial_{x^+}T)^{-1}\partial_{x^-}T\right] - (l^- - l^+)A \tag{3.29}$$

and the gradient contribution of the transition is given by

$$\xi_p = (l^- - l^+)B - \lambda^+ CB + \lambda^+[\partial_{\dot{x}^-}T\partial_p f^- + \partial_{\dot{x}^+}T\partial_p f + \partial_p T] \tag{3.30}$$

with

$$A = -\frac{\partial_{x^-}j}{\partial_{\dot{x}^-}j\partial_t f^- + \partial_{x^-}j\dot{x}^- + \partial_t j} \tag{3.31}$$

$$B = -\frac{\partial_{\dot{x}^-}j\partial_p f^- + \partial_p j}{\partial_{\dot{x}^-}j\partial_t f^- + \partial_{x^-}j\dot{x}^- + \partial_t j} \tag{3.32}$$

$$C = -\left[f^+ + (\partial_{x^+}T)^{-1}\left(\partial_{\dot{x}^-}T\partial_t f^- + \partial_{x^-}Tf^- + \partial_{\dot{x}^+}T\partial_t f^+ + \partial_t T\right)\right] \tag{3.33}$$

$$\tag{3.34}$$

Proof. We can focus on one term in the sum (3.20):

$$\xi = (l^- - l^+)\frac{dt}{dp} + \lambda^- \cdot d_p x^- - \lambda^+ \cdot d_p x^+ \tag{3.35}$$

we will use eq. 3.28 and 3.27 to collect terms $d_p x^-$. We can write

$$\begin{aligned}\frac{dt}{dp} = Ad_p x^- + B = &-\frac{\partial_{x^-}j}{\partial_{\dot{x}^-}j\partial_t f^- + \partial_{x^-}j\dot{x}^- + \partial_t j}d_p x^- \\ &- \frac{\partial_{\dot{x}^-}j\partial_p f^- + \partial_p j}{\partial_{\dot{x}^-}j\partial_t f^- + \partial_{x^-}j\dot{x}^- + \partial_t j}\end{aligned} \tag{3.36}$$

and

$$\begin{aligned}d_p x^+ = &-\left[f^+ + \ldots\right]Ad_p x^- - \left[f^+ + \ldots\right]B \\ &- (\partial_{x^+}T)^{-1}\partial_{x^-}Td_p x^- - (\partial_{x^+}T)^{-1}\left(\partial_{\dot{x}^-}T\partial_p f^- + \partial_{\dot{x}^+}T\partial_p f^+ + \partial_p T\right)\end{aligned} \tag{3.37}$$

inserting into eq. 3.35 we get

$$\begin{aligned}\xi = &\left[(l^- - l^+)A + \lambda^- + \lambda^+[f^+ + \ldots]Ad_p x^- + \lambda^+[(\partial_{x^+}T)^{-1}\partial_{x^-}T]\right]d_p x^- \\ &+ \lambda^+[f^+ + \ldots]B + \lambda^+[\partial_{\dot{x}^-}T\partial_p f^- + \partial_{\dot{x}^+}T\partial_p f + \partial_p T] + (l^- - l^+)B\end{aligned} \tag{3.38}$$

Therefore we derive for the jumps in the adjoint variables

$$\begin{aligned}\lambda^- = \lambda^+ &\left[\left[f^+ + \ldots\right]\frac{\partial_{x^-}j}{\partial_{\dot{x}^-}j\partial_t f^- + \partial_{x^-}j\dot{x}^- + \partial_t j} - (\partial_{x^+}T)^{-1}\partial_{x^-}T\right] \\ &+ (l^- - l^+)\frac{\partial_{x^-}j}{\partial_{\dot{x}^-}j\partial_t f^- + \partial_{x^-}j\dot{x}^- + \partial_t j}\end{aligned} \tag{3.39}$$

and the gradient contribution

$$\xi_p = (l^- - l^+)B + \lambda^+[f^+ + \ldots]B + \lambda^+[\partial_{\dot{x}^-}T\partial_p f^- + \partial_{\dot{x}^+}T\partial_p f + \partial_p T] \tag{3.40}$$

□

These equations simplify further if we make assumptions on the transition equations. If we assume that the transition equations do not explicitely depend on time or velocities

$$T(x^+, x^-, p) = x^+ - \theta(x^-, p) \tag{3.41}$$

it follows that

$$\partial_{x^+} T(x^+, p, t) = I, \partial_{\dot{x}^-} T = 0, \partial_{\dot{x}^+} T = 0, \partial_t T = 0 \tag{3.42}$$

and the transition equations of the adjoint state variables simplify to

$$\begin{aligned}\lambda^- = \lambda^+ \left[\left[f^+ - \partial_{x^-}\theta f^- \right] \frac{\partial_{x^-} j}{\partial_{\dot{x}^-} j \partial_t f^- + \partial_{x^-} j \dot{x}^- + \partial_t j} + \partial_{x^-}\theta \right] \\ + (l^- - l^+) \frac{\partial_{x^-} j}{\partial_{\dot{x}^-} j \partial_t f^- + \partial_{x^-} j \dot{x}^- + \partial_t j}\end{aligned} \tag{3.43}$$

Moreover the contribution to the gradient simplifies to

$$\xi_p = (l^- - l^+) B + \lambda^+ \left[f^+ - \partial_{x^-}\theta f^- \right] B - \lambda^+ \partial_p \theta \tag{3.44}$$

In the cases we are interested in there are typically two distinct situations: One where the jump condition j depends only on time, this is the case for external input and one where the jump condition j only depends on the state of the system. We also don't want to consider jump conditions j that depend on the state velocity. In the first case we have

$$\partial_{x^-} j = 0, \partial_{\dot{x}^-} j = 0 \tag{3.45}$$

and in the second case

$$\partial_t j = 0, \partial_{\dot{x}^-} j = 0 \tag{3.46}$$

and therefore we have in the first case (external events)

$$\lambda^- = \lambda^+ \left[\partial_{x^-}\theta \right] \tag{3.47}$$

$$\xi_p = -\left[(l^- - l^+) + \lambda^+ \left[f^+ - \partial_{x^-}\theta f^- \right] \right] \frac{\partial_p j}{\partial_t j} - \lambda^+ \partial_p \theta \tag{3.48}$$

and in the second case

$$\lambda^- = \lambda^+ \left[\left[f^+ - \partial_{x^-}\theta f^- \right] \frac{\partial_{x^-} j}{\partial_{x^-} j \dot{x}^-} + \partial_{x^-}\theta \right] + (l^- - l^+) \frac{\partial_{x^-} j}{\partial_{x^-} j \dot{x}^-} \tag{3.49}$$

$$\xi_p = -\left[(l^- - l^+) + \lambda^+ \left[f^+ - \partial_{x^-}\theta f^- \right] \right] \frac{\partial_p j}{\partial_{x^-} j \dot{x}^-} - \lambda^+ \partial_p \theta \tag{3.50}$$

Now it should be observed that in the case of spiking neuron models the transition equations are are almost always translations in a subspace of the state space. Moreover there is are finite sets of transition and jump conditions associated to each neuron independently. That is we can partition the state space $\mathbf{R}^N$ into K copies of L state variables and then there are typically a small set of jump conditions $j_k^{(l)}, k = 1, \ldots, K$ associated to each neuron. One of them is distinguished as the "spike" or "fire" jump condition, which determines spike propagation to other neurons. Another example would be the end of the absolute refractory period. The transition equations associated to the "spike" jump condition then implements spike propagation. In simple neuron models the spike jump conditions can be thought of as codimension one hyperplanes in the membrane-voltage state space. This leads to the following defintion

Definition 3.2.3 (Neural Event Processing Element (NEPE))**.** *A Neural Event Processing Element is given by a differentiable function*

$$f\colon S \times P \times \mathbf{R} \to TS \tag{3.51}$$

together with a set of differentiable state jump conditions

$$j_k\colon S \times P \to \mathbf{R} \tag{3.52}$$

with associated differentiable state transition functions

$$\theta_k\colon S \times P \to S \tag{3.53}$$

where each of the jump conditions j_k only depends on a (small) subset of the state variables, that is there is a projection $\pi_k\colon S \to S_k$ to some smaller space S_k and $j_k = \bar{j}_k \circ (\pi_k \times id_P)$. Here S, P could be manifolds in which case TS would be the tangent space to S. The dynamics, state jump conditions and transition functions need to satisfy the assumptions in (3.2.1)*.*

It should be noted that both f, j_k, θ_k can be neural networks, that is compositions of parametetrised differentiable functions such as convolutions, linear maps and non-linearities such as sigmoids or rectifier linear units. The adjoint dynamics together with the transition equations for the adjoint variables are sufficient for employing standard machine learning optimization techniques in such a situation. More precisely for systems of equations of this kind an efficient *event* based analogue of the Backpropagation algorithm can be implemented, which uses integrates during the forward dynamics the differential equations specified by f, together with jumps and transitions given by j_k, θ_k and uses the jump equations of the adjoint state variables and adjoint equations to compute gradients. What is less obvious but nevertheless true is the fact that one can easily compose several of such Neural Processing Elements, in a way that also the adjoint equations compose.

I will now discuss two simple examples of these kinds of systems: Leaky Integrate and Fire Neurons and adaptive Leaky Integrate and Fire Neurons. Both are special cases of linear time invariant ordinary differential equations with jumps, which I will discuss in chapter 4.

Leaky Integrate and Fire Neuron

We can apply the procedure outlined above to the case of a leaky integrate and fire neuron with exponential synapses. For simplicity of the derivation we only track time constants and not membrane capacitances or conductances. The state variables are $x = (V, I)$, namely the membrane voltage and synaptic input currents and the parameters are the synaptic weights $p = W$. The state variables follow the differential equations

$$\tau_v \dot{V} = (V_{\text{leak}} - V) + I \tag{3.54}$$

$$\tau_s \dot{I} = -I, \tag{3.55}$$

here V_{leak} denotes the leak potential, τ_v the membrane and τ_s the synaptic time constants. Both V and I are N-dimensional vectors, whose components we denote by $V_n, I_m, n, m = 1 \dots N$.

The membrane voltage V_n of neuron n jumps and input currents I jump, when the threshhold voltage $(V_{\text{th}})_n$ is reached. The corresponding jump condition is given by

$$j_n(V^-) = ((V^-)_n - (V_{\text{th}})_n). \tag{3.56}$$

One subtlety to consider is that in a numerical integration scheme multiple of these jump conditions can be satisfied simultaneously, within one timestep this can lead to potentially expensive backtracking and root finding. Moreover an efficient numerical implementation also needs to track the N jump conditions simultaneously. In the case we are considering the resulting jumps commute as they are simply translations of the synaptic input currents I, therefore even the analytical possibility that several jump conditions are satisfied simulataneously poses no problem.

Assume that the jump condition for neuron n is satisfied, then the state variables change as follows

$$V_n^+ = V_m^- - ((V_{\text{th}})_n - (V_{\text{reset}})_n) = V_m^- - \vartheta_n \tag{3.57}$$
$$V_m^+ = V_m^- \tag{3.58}$$
$$I_m^+ = I_m^- + w_{nm}. \tag{3.59}$$

In other words the states changes by a euclidean translation $x^+ = T_n^{\text{rec}}(x^-)$. From a numerical perspective it is much cheaper to implement this translation compared to a matrix multiplication. Similarly external spike input results in jumps

$$V_m^+ = V_m^- \tag{3.60}$$
$$I_m^+ = I_m^- + w_{km}^{\text{in}}, \tag{3.61}$$

which again is a euclidean translation $x^+ = T_k^{\text{in}}(x^-)$.

Based on the equations (3.54) when no jump conditions are satisfied, it is easy to derive the adjoint equations from (3.9):

$$\dot{\lambda}_V = \lambda_V + \partial_V l \tag{3.62}$$
$$\dot{\lambda}_I = \lambda_I - \lambda_V + \partial_I l \tag{3.63}$$

in practice since the boundary condition is given at T, the equation should be formulated with $s = T - t$, which leads to

$$\lambda_V' = -\lambda_V - \partial_V l \tag{3.64}$$
$$\lambda_I' = -\lambda_I + \lambda_V - \partial_I l, \tag{3.65}$$

where we have denoted the total derivative $\frac{df}{ds}$ by f'.

We can use the jump condition and transition equations to derive relationships between the forward sensitivities before and after the jumps. Taking the total derivative of the jump condition we get at the spike time t^{post}

$$(s_V^-)_n + \dot{V}_n^- \frac{dt^{\text{post}}}{dw_{ij}} = 0 \tag{3.66}$$

and therefore the transition time sensitivity τ^{post} is given by

$$\tau^{\text{post}} = \frac{dt^{\text{post}}}{dw_{ij}} = -\frac{1}{\dot{V}_n^-}\left(\frac{\partial V^-}{\partial w_{ij}}\right) = -\frac{(s_V^-)_n}{\dot{V}_n^-} \tag{3.67}$$

This relation has been previously derived in the context of gradient based learning in spiking neural networks in [13, 24, 25, 167, 54]. It can also directly be derived from the general formulas (3.27) [63]. Note that the initial work in the context of spiking neural networks [24] asserted incorrectly, that it is an approximation. A proof using the implicit function theorem appeared in [168]. The forward sensitivity equations are given by

$$\tau_v \dot{s}_V = -s_V + s_I \tag{3.68}$$

$$\tau_s \dot{s}_I = -s_I \tag{3.69}$$

We can similarly derive from the transition equations (3.57, 3.59, 3.58) the corresponding relationships for the forward sensitivities

$$(s_V^+)_n = (s_V^-)_n + \frac{1}{\tau_v}(\vartheta + w_{nn})\frac{(s_V^-)_n}{(\dot{V}^-)_n} \tag{3.70}$$

$$(s_V^+)_m = (s_V^-)_m + \frac{1}{\tau_v} w_{nm}\frac{(s_V^-)_n}{(\dot{V}^-)_n}, \forall m \neq n \tag{3.71}$$

$$(s_I^+)_m = (s_I^-)_m - \frac{1}{\tau_s} w_{nm}\frac{(s_V^-)_n}{(\dot{V}^-)_n} + A_{nm}\delta_{in}\delta_{jm}, \tag{3.72}$$

here I used A_{nm} to denote the adjacency matrix of the neuron connectivity.

Taken together these equations give relationships between $s_v^+, s_v^-, s_I^+, s_I^-$ that can be used to derive relations for $\lambda_V^+, \lambda_V^-, \lambda_I^+, \lambda_I^-$, by inserting in equation (3.18). Focussing on one term in the sum, we find

$$\begin{aligned}\xi_k = &\left[l^- - l^+ + \frac{\partial l_p}{\partial t_k^{\text{post}}}\right]\tau_k^{\text{post}} \\ &+ \left[\tau_v\left(\lambda_V^+ \cdot s_V^- - \lambda_V^+ \cdot s_V^+\right) + \tau_s\left(\lambda_I^- \cdot s_I^- - \lambda_I^+ \cdot s_I^+\right)\right]|_{t_k^{\text{post}}}.\end{aligned} \tag{3.73}$$

Inserting the jump relations for the forward sensitivities (3.70, 3.71, 3.72), the sensitivity of the transition time (3.67) and collecting terms in s_V^-, s_I^- (we denote by $n(k)$ the index of the neuron that has spiked)

$$\begin{aligned}\xi_k = &\sum_{m\neq n(k)}\left[\tau_v(\lambda_V^- - \lambda_V^+)_m(s_V^-)_m + \tau_s(\lambda_I^- - \lambda_I^+)_m(s_I^-)_m - \tau_s A_{nm}\delta_{in(k)}\delta_{jm}(\lambda_I^+)_m\right] \\ &+ \tau_v\left[(\lambda_V^-)_{n(k)} - \left((\lambda_V^+)_{n(k)} + \frac{1}{\tau_v \dot{V}_{n(k)}^-}\left[\vartheta(\lambda_V^+)_{n(k)}\right.\right.\right. \\ &\left.\left.\left. + \sum_{m\neq n(k)} w_{n(k)m}(\lambda_V^+ - \lambda_I^+)_m + \frac{\partial l_p}{\partial t_k^{\text{post}}} + l^- - l^+\right]\right)\right](s_V^-)_{n(k)} \\ &+ \tau_s(\lambda_I^- - \lambda_I^+)_{n(k)}(s_I^-)_{n(k)}.\end{aligned} \tag{3.74}$$

to eliminate the dependency on the forward sensitivity, we therefore should demand the following jumps in the adjoint state variables at the spike time

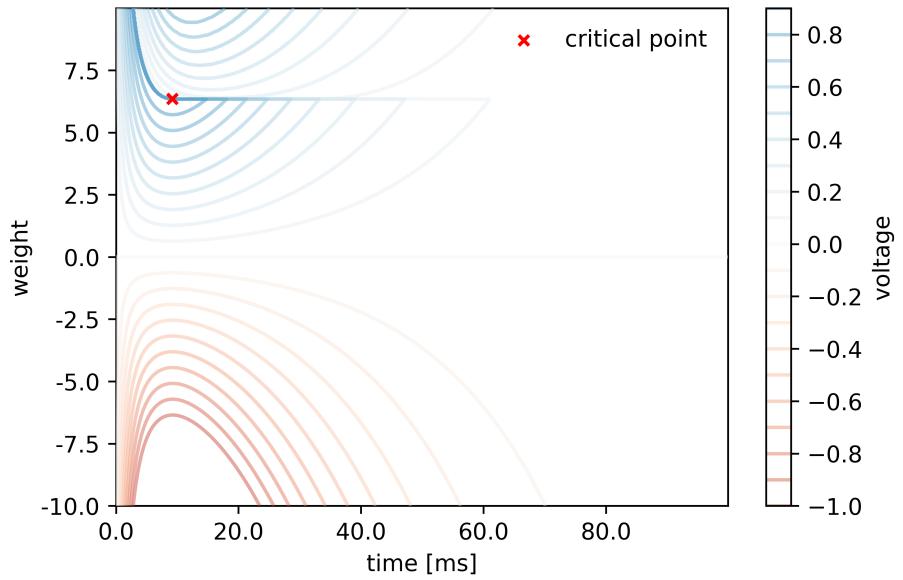


Figure 3.1: Contourplot of mebrane voltage in response to a single spike input to a synapse with time constants $\tau_{\text{syn}} = 5\,\text{ms}$ and $\tau_{\text{mem}} = 20\,\text{ms}$. Close to a jump, the spike time t^{post} is a smooth function of the weight. The jump condition $v(t, w) - \theta = 0$ defines an implicit function (the dark blue line marks $dv = 0$). The gradient diverges at the red cross, where $\dot{v}^- = 0$. In weight space this defines a hyperplane $w^{\text{crit}} = 0$, where the total number of spikes of this neuron increases by one (see also 3.2)

t_k^{post}:

$$\lambda_I^- = \lambda_I^+ \tag{3.75}$$

$$(\lambda_V^-)_m = (\lambda_V^+)_m, m \neq n(k) \tag{3.76}$$

$$(\lambda_V^-)_{n(k)} = (\lambda_V^+)_{n(k)} + \frac{1}{\tau_v(\dot{V}^-)_{n(k)}} \left[\vartheta(\lambda_V^+)_{n(k)} + (W^T(\lambda_V^+ - \lambda_I))_{n(k)} + \frac{\partial l_p}{\partial t_k^{\text{post}}} + l_V^- - l_V^+ \right]. \tag{3.77}$$

Note that naturally we want to relate variables backward in time and that the state of the forward equations only enters for the neuron that spiked at that time during forward integration. This means that this state only needs to be maintained locally. Moreover the adjoint state variables are also coupled only at spike times and are otherwise local to the neurons.

Using these relations the expression for the total derivative of the action with respect to the parameters (3.18) reduces to

$$d_p\mathcal{S} = d_{w_{ij}}\mathcal{S} = \int_0^T \partial_p l dt - \tau_s \sum_k A_{n(k)j}\delta_{in(k)}(\lambda_I^+)_j. \tag{3.78}$$

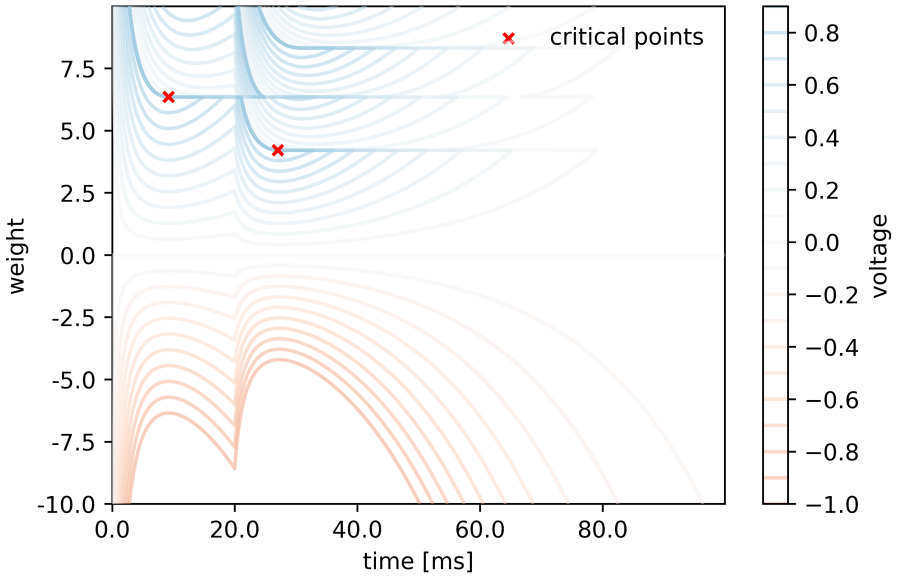


Figure 3.2: Contourplot of mebrane voltage in response to two spikes at $0\,\text{ms}, 20\,\text{ms}$ to a synapse with time constants $\tau_{\text{syn}} = 5\,\text{ms}$ and $\tau_{\text{mem}} = 20\,\text{ms}$. We indicate two out of four total critical points at which additional spikes occur. There is a well defined spike count function $n(w)$, which can be read of from the blue horizontal lines in the contourplot at $t = T$. This observation was used in the seminal work [76] to derive a learning rule.

The equations of the jumps in the adjoint variables (3.75), together with the adjoint equations (3.64), can be numerically implemented. We did so, as well as derived and implemented the corresponding adjoint equations and jumps for a number of other point neuron models in a PyTorch [122] based library, as well as in a custom event-based simulator implemented by Timo Wunderlich.

LIF Neuron with Adaptive Threshold

A slight generalization of the LIF neuron model we considered in the section before is a neuron model with adaptive threshhold. The continuous equations are given by

$$\tau_v \dot{V} = -V + I \tag{3.79}$$
$$\tau_s \dot{I} = -I \tag{3.80}$$
$$\tau_b \dot{B} = -B \tag{3.81}$$

The jump conditions for the n-th neuron is given by

$$j_n(V^-, B^-) = V_n^- - V_{\text{th}} + B_n^-. \tag{3.82}$$

In other words the threshhold of each of the N neurons is modulated independently by the state variable B_n. The transition equations are given by

$$V_n^+ = V_n^- - \vartheta \tag{3.83}$$
$$B_n^+ = B_n^+ + \beta \tag{3.84}$$
$$I^+ = I^- + We_n \tag{3.85}$$

where ϑ, β parametrize the jump in the membrane voltage and adaption variable B respectively, W is the synaptic weight matrix and e_n denotes the n-th standard basis vector.

The general idea behind this simple modification is to introduce another longer time constant $\tau_b > \tau_v$, which enables the neurons to have a "memory" of recent spike activity at the time scale τ_b. Experimental results [17] show that this suffices to significantly increase the computational capabilities on time-dependent tasks. We can now go through the same derivation as in the case of a LIF neuron. Since it is highly similar we will for the most part just state results. The first observation is that now the jump condition of the n-th neuron now depends linearly on the two state variables V_n, B_n. We find that the parameter sensitivity of a transition time t^{post} is given by

$$\tau^{\text{post}} = -\frac{(s_V^- + s_B^-)_n}{(\dot{V}^- + \dot{B}^-)_n} \tag{3.86}$$

and we can relate the *forward sensitivities* $s_V^{\pm}, s_I^{\pm}, s_B^{\pm}$ as follows:

$$(s_V^+)_n = (s_V^-)_n + \frac{1}{\tau_v}(\vartheta + w_{nn})\frac{(s_V^- + s_B^-)_n}{(\dot{V}^- + \dot{B}^-)_n} \tag{3.87}$$

$$(s_V^+)_m = (s_V^-)_m + \frac{1}{\tau_v} w_{nm}\frac{(s_V^- + s_B^-)_n}{(\dot{V}^- + \dot{B}^-)_n}, \forall m \neq n \tag{3.88}$$

$$(s_I^+)_m = (s_I^-)_m - \frac{1}{\tau_s} w_{nm}\frac{(s_V^- + s_B^-)_n}{(\dot{V}^- + \dot{B}^-)_n} + A_{nm}\delta_{im}\delta_{in} \tag{3.89}$$

$$(s_B^+)_n = (s_B^-)_n - \frac{1}{\tau_b}\beta\frac{(s_V^- + s_B^-)_n}{(\dot{V}^- + \dot{B}^-)_n} \tag{3.90}$$

$$(s_B^+)_m = (s_B^-)_m, \forall m \neq n \tag{3.91}$$

here I used A_{nm} to denote the adjacency matrix of the neuron connectivity. The forward sensitivity equations outside of jumps are given by

$$\tau_v \dot{s}_V = -s_V + s_I \tag{3.92}$$

$$\tau_s \dot{s}_I = -s_I \tag{3.93}$$

$$\tau_b \dot{s}_B = -s_B \tag{3.94}$$

Using these equations we can compute the gradient w.r.t. the loss as

$$\begin{aligned}\frac{\mathrm{d}L}{\mathrm{d}w_{ij}} = &\sum_{k=0}^{N_{\text{post}}}\left[\int_{t_k^{\text{post}}}^{t_{k+1}^{\text{post}}}\left[\frac{\partial l}{\partial V}\cdot s_V + \frac{\partial l}{\partial I}\cdot s_I + \frac{\partial l}{\partial B}\cdot s_B + \frac{\partial l}{\partial w_{ij}}\right]\right] \\ &- \sum_{k=0}^{N_{\text{post}}}\left[\frac{\partial l_p}{\partial t_k} + l^+ - l^-\right]\frac{(s_V^- + s_B^-)_{n(k)}}{(\dot{V}^- + \dot{B}^-)_{n(k)}}\end{aligned} \tag{3.95}$$

where we've used $n(k)$ to denote the index of the neuron emitting the k-th spike. The free dynamics of the adjoint variables is given by

$$\tau_v \lambda_V' = -\lambda_V - \partial_V l \tag{3.96}$$

$$\tau_b \lambda_B' = -\lambda_B - \partial_B l \tag{3.97}$$

$$\tau_s \lambda_I' = -\lambda_I + \lambda_V - \partial_I l \tag{3.98}$$

Therefore the dynamics of the adaptation adjoint variable is fully decoupled from that of the current and membrane voltage adjoint variables. We can again restrict our attention to one term ξ_k in the sum over transitions

$$\begin{aligned}\xi_k = &\left[l^- - l^+ + \frac{\partial l_p}{\partial t_k^{\text{post}}}\right]\tau \\ &+ \left[\tau_v\left(\lambda_V^+\cdot s_V^- - \lambda_V^+\cdot s_V^+\right) + \tau_s\left(\lambda_I^-\cdot s_I^- - \lambda_I^+\cdot s_I^+\right) + \tau_b\left(\lambda_B^-\cdot s_B^- - \lambda_B^+\cdot s_B^+\right)\right]|_{t_k^{\text{post}}}\end{aligned} \tag{3.99}$$

Inserting the relations obtained for the forward sensitivities (3.87-3.91), as well

as (3.86) and reorganizing yields:

$$\begin{aligned}
\xi_k = & \sum_{m\neq n(k)} \left[\tau_v(\lambda_V^- - \lambda_V^+)_m (s_V^-)_m + \tau_b(\lambda_B^- - \lambda_B^+)_m (s_B^-)_m\right] \\
& + \sum_{m\neq n(k)} \left[\tau_s(\lambda_I^- - \lambda_I^+)_m (s_I^-)_m - \tau_s A_{nm}\delta_{in(k)}\delta_{jm}(\lambda_I^+)_m\right] \\
& + \tau_v \left[(\lambda_V^-)_{n(k)} - \left((\lambda_V^+)_{n(k)} + \frac{1}{\tau_v(\dot{V}^-_{n(k)} + \dot{B}^-_{n(k)})}\left[\vartheta(\lambda_V^+)_{n(k)} - \beta(\lambda_B^+)_{n(k)}\right.\right.\right. \\
& \left.\left.\left. + \sum_{m\neq n(k)} w_{n(k)m}(\lambda_V^+ - \lambda_I^+)_m + \frac{\partial l_p}{\partial t_k^{\text{post}}} + l^- - l^+\right)\right)\right] (s_V^-)_{n(k)} \\
& + \tau_b \left[(\lambda_B^-)_{n(k)} - \left((\lambda_B^+)_{n(k)} + \frac{1}{\tau_b(\dot{V}^-_{n(k)} + \dot{B}^-_{n(k)})}\left[\vartheta(\lambda_V^+)_{n(k)} - \beta(\lambda_B^+)_{n(k)}\right.\right.\right. \\
& \left.\left.\left. + \sum_{m\neq n(k)} w_{n(k)m}(\lambda_V^+ - \lambda_I^+)_m + \frac{\partial l_p}{\partial t_k^{\text{post}}} + l^- - l^+\right)\right)\right] (s_B^-)_{n(k)} \\
& + \tau_s(\lambda_I^- - \lambda_I^+)_{n(k)}(s_I^-)_{n(k)}.
\end{aligned} \tag{3.100}$$

Therefore the transition equations of the adjoint variables are given by

$$\lambda_I^- = \lambda_I^+ \tag{3.101}$$

$$\lambda_B^- = \lambda_B^+, m \neq n(k) \tag{3.102}$$

$$\begin{aligned}
(\lambda_B^-)_{n(k)} = & (\lambda_B^+)_{n(k)} \\
& + \frac{1}{\tau_b(\dot{V}^- + \dot{B}^-)_{n(k)}}\left[\vartheta(\lambda_V^+)_{n(k)} - \beta(\lambda_B^+)_{n(k)} + (W^T(\lambda_V^+ - \lambda_I))_{n(k)} + \frac{\partial l_p}{\partial t_k^{\text{post}}} + l_V^- - l_V^+\right]
\end{aligned} \tag{3.103}$$

$$(\lambda_V^-)_m = (\lambda_V^+)_m, m \neq n(k) \tag{3.104}$$

$$\begin{aligned}
(\lambda_V^-)_{n(k)} = & (\lambda_V^+)_{n(k)} \\
& + \frac{1}{\tau_v(\dot{V}^- + \dot{B}^-)_{n(k)}}\left[\vartheta(\lambda_V^+)_{n(k)} - \beta(\lambda_B^+)_{n(k)} + (W^T(\lambda_V^+ - \lambda_I))_{n(k)} + \frac{\partial l_p}{\partial t_k^{\text{post}}} + l_V^- - l_V^+\right].
\end{aligned} \tag{3.105}$$

and the gradient of the loss function is again

$$d_p\mathcal{S} = d_{w_{ij}}\mathcal{S} = \int_0^T \partial_p l dt - \tau_s \sum_k A_{n(k)j}\delta_{in(k)}(\lambda_I^+)_j. \tag{3.106}$$

One interesting observation is that the issue of critical weight parameters is better behaved for adaptive LIF neurons, assuming that the adaptation variable B is initialised with a non-zero value. For large time constants τ_b the value of $\dot{B}^-$ will be sufficiently far from zero for jumps in the λ_V and λ_B not to diverge, as $\dot{V}_k^- \geq 0$ for a neuron that spiked. In other words there will be no point in weight space where $\dot{V}^- + \dot{B}^- = 0$ in contrast to the situation of LIF neurons.

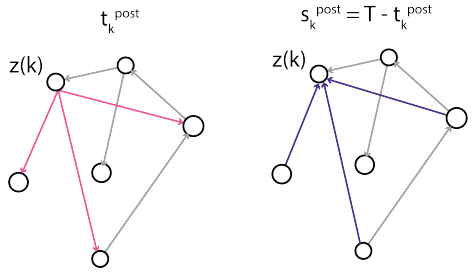


Figure 3.3: Illustration of the scatter and gather pattern of spike communication (pink) and adjoint error communication (violet) of a given neuron the spikes at time t and receives error information at $s = T - t$ respectively. Alternatively each neuron can keep track of the time and source of any spike input it receives and transmit error information during the adjoint integration accordingly.

3.3 EventProp Algorithm

The forward and adjoint equations with jumps of the LIF and adaptive LIF neuron respectively suggest an algorithm to compute the gradients (3.78, 3.106) with respect to the parameters efficiently (see also [165], where this algorithm is explained in the case of the LIF neuron). The main observation is that both during forward integration and during integration of the adjoint state dynamics, the state variables (V, I) or (V, I, B) are only coupled at spike times and so are the adjoint variables (λ_V, λ_I) or $(\lambda_V, \lambda_I, \lambda_B)$. Moreover while during forward integration a spike at time t results in a broadcast of a message to all neurons the spiking neuron is a presynaptic partner to, during adjoint integration this results at time $s = T - t$ in the gathering of errors according to (3.75, 3.101) at this neuron. This is illustrated in figure 3.3. Such a communication pattern lends itself well to a distributed implementation both in event-based simulators, as well as neuromorphic hardware. Finally in the case of the more general Neural Event Processing elements it suggests an end-to-end training approach for potentially large systems of Neural ODEs that are coupled through event based communication.

In the case of digital-analog neuromorphic hardware, such as the BrainScaleS-2 system the method could be implemented on-chip with the use of the plasticity processing unit or in-the-loop learning [40], I will discuss those options in chapter 7.

In order to evaluate the method I conducted a number of experiments to verify its performance and evaluate its behaviour relative to previously established methods of training spiking neural networks. Two different implementations were used: A purely event based simulators that uses exact solutions of the LIF neuron equations and root bracketing to implement arbitrary feed-forward networks with efficient event communication and high time precision. In a similar fashion the adjoint dynamics can be solved exactly based on information stored at event times during forward integration. This implementation was done by Timo Wunderlich [165]. I implemented several neuron models besides the LIF and adaptive LIF neuron model in a PyTorch [121] based library. In that implementation I replaced the term $\lambda_V^+ - \lambda_I$ by λ_I^- in in eqs. (3.77), (3.105), (3.103).

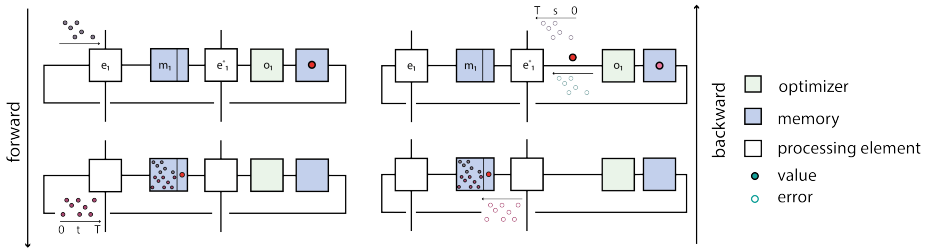


Figure 3.4: Schematic depiction of the EventProp algorithm implemented by a Neural (Event) Processing Element. Given a spiking neural network, which can be implemented by a processing element e_1 (for example a neuromorphic chip), the EventProp algorithm defines an adjoint spiking neural network, whose differential equations are implemented by a processing element e_1^*. During forward integration the processing element e_1 only needs to store the pre-spike times t_k^{pre} and $(t_l^{\text{post}}, \dot{v}^-)$ at the post spike times t_l^{post} in the memory m_1. During backward integration the processing element e_1^* receives error information at the post-spike times t_l^{post} and uses the knowledge of the pre-spike times t^{pre} to pass on errors to other processing elements. The optimizer o_1 integrates weight changes and applies the weight update at $s = 0$.

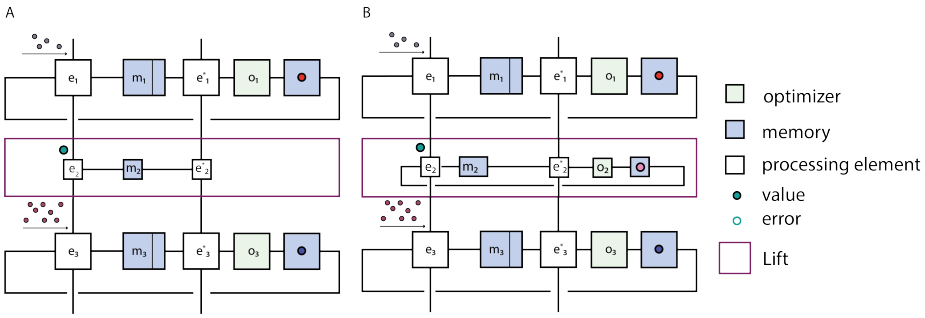


Figure 3.5: Illustration of the compatibility of Neural Event Processing Elements and other Neural Processing Elements. (**A**) illustrates the case of a pointwise (in time) application of a primitive ϕ to the event output of a processing element e_1, which then gets passed to a processing element e_3. (**B**) illustrates the same situation but for a parametrised processing element e_2. It can be seen as an illustration of the PyTorch code listing 3.1, with the correspondence $e_1 =$ LIFFeedForwardLayer, $e_2 =$ torch.nn.Conv2d, $e_3 =$ LIFFeedForwardLayer.

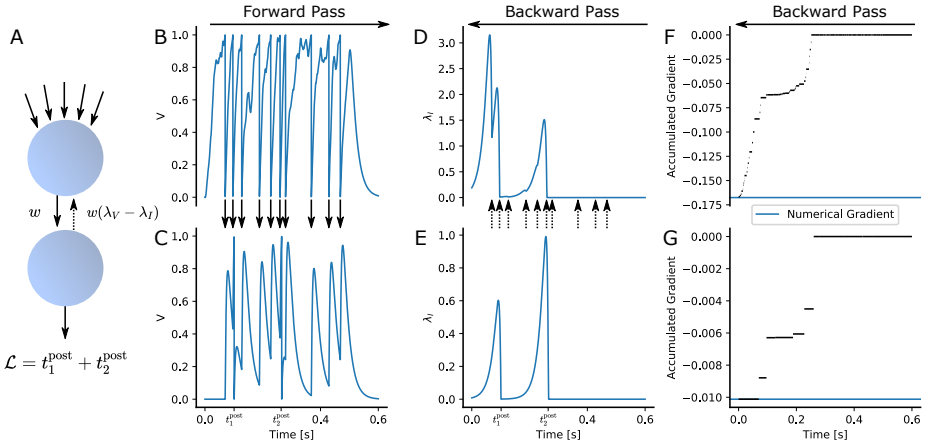


Figure 3.6: Two leaky integrate and fire neurons are connected by a weight w in a feed-forward fashion (A). The first neuron receives poissonian input from 100 independent Poisson sources with frequency 200 Hz weighted by randomly initialised static weights. The loss is given by the sum over the spike times of the second neuron $\mathcal{L} = \sum_i t_i$. We show the membrane voltage traces for one particular trial in (B,C). In the particular trial the upper neuron spikes at times indicated by the black arrows and the lower neuron spikes twice at times $t_1^{\text{post}}, t_2^{\text{post}}$. Accordingly the loss has two terms $t_1^{\text{post}} + t_2^{\text{post}}$ resulting in jumps of the adjoint state variable λ_V, which influences the adjoint state variable λ_I of the lower neuron at these times (E). At spike times of the upper neuron its adjoint state variable λ_v jumps by $w(\lambda_V - \lambda_I)$ (G), since integration happens backwards in time only spikes (dashed arrows) that happen before the last spike of the lower neuron lead to contributions (D). The gradients get accumulated at presynaptic spike times at the weights, we show one of the accumulated gradients of the upper neuron in (F) and the gradients accumulated for w in (G), both agree with the gradient computed by central differences with an error of less than 10^{-7}. Figure adapted from [165].

A PyTorch based implementation has the main advantage that it is fully compatible with standard machine learning tools. However implementing differential equations efficiently is an art in itself, so in the long run integration into either a dedicated event based simulator or a general purpose solver suite such as [132, 131, 133] seems desirable.

An example how the library can be used to compose models out of PyTorch modules that implement the feedforward and adjoint equations is shown in 3.1. It also indicates how ordinary PyTorch modules can be "lifted" to sequentially compose with the spiking neuron models. The "Lift" module is a higher order module which applies a module pointwise in time.

Listing 3.1: Use of the PyTorch based library implementing the EventProp algorithm. The example also illustrates the composability of the method.

```
import torch

data = torch.randn(T, B, IC, W, H)

module = torch.nn.Sequential(
    LIFFeedForwardLayer()
    Lift(torch.nn.Conv2d(IC, OC, F, S)),
    LIFFeedForwardLayer(),
)

output, _ = module(data)
```

In order to be able to compare to existing surrogate gradient methods, I also implemented corresponding PyTorch modules of the same neuron models. The implementation of the forward pass can be shared, with the exception of which information is saved for the backward integration. In the case of surrogate gradient training this is done automatically. I used a factor of $\alpha = 100$ in the super-spike surrogate gradient [171] throughout, recent work [172] suggests that the method is relatively stable with respect to this hyperparamter, but a more thourough analysis would need to compare performance across this hyperparamter as well.

3.4 Experiments

3.4.1 Single Neuron Tasks

A simple task to consider is a single neuron stimulated at different times by k fixed poisson distributed spike trains, with synaptic weights distributed according to a gaussian distribution. The goal is for the neuron to respond to these fixed spike trains with a certain number of spikes n_{target} within a time T. The loss in this case is given

$$l = -n_{\text{target}}/T + \sum_i \delta(t - t_i(p)),$$

here the sum is over the times $t_i(p)$ at which the neuron fired. The total loss (action) therefore evaluates to

$$S = \int_0^T (-n_{\text{target}}/T + \sum_i \delta(t - t_i(p)))dt = n_{\text{actual}} - n_{\text{target}}.$$

As can be seen in figure 3.7 the total loss remains constant for multiple epochs, but nevertheless the neuron eventually spikes the prescribed number of times. This is due to the fact that the time course of λ_i is non-zero and therefore leads to weight changes at presynaptic spike times.

The second single neuron task I consider demonstrates that it is also possible to incorporate loss functions dependent on voltage information at pre-defined points in time. This illustrates equation 3.13. The target is for the membrane voltage to reach the spike threshhold v_{th} at times t_0, t_1. This can be expressed by a loss function

$$l = |v - v_{\text{th}}|^2 \delta(t - t_0) + |v - v_{\text{th}}|^2 \delta(t - t_1). \tag{3.107}$$

One should note that a priori there is no guarantee that this will result in spikes only at times t_0, t_1. This can be mitigated by introducing a regularisation term

$$l_{\text{reg}} = \int \Theta(v - v_{\text{th}} + \epsilon)|v - v_{\text{th}}|^2 \tag{3.108}$$

Figure 3.8 illustates the opimization progress on a specific example. As in the first task a single neuron is stimulated with fixed poisson input. In contrast to the first task the total loss continuosly decreases. This is intuitively clear because we are optimizing for the square distance of the voltage state at given points in time. The adjoint state variable λ_v on the other hand exhibits jumps at times t_0, t_1.

3.4.2 Ying-Yang Dataset

The experiment described in this section, as well as the implementation of the event-based simulator was done by Timo Wunderlich [165]. We trained a two-layer spiking neural networks using an event-based simulator on a two-dimensional dataset first described in [71]. We also used the same time-to-first spike loss as in [71]

$$\mathcal{L}(t^{\text{post}}, l) = -\log\left[\frac{\exp\left(-t_l^{\text{post}}/(\xi\tau_{\text{syn}})\right)}{\sum_k \exp\left(-t_k^{\text{post}}/(\xi\tau_{\text{syn}})\right)}\right] + \alpha\left[\exp\left(\frac{t_l^{\text{post}}}{\beta\tau_{\text{syn}}}\right) - 1\right]. \tag{3.109}$$

The loss is given as a sum of a cross-entropy term, where t_k^{post} is the first spike time of neuron k and l is the index of the neuron correponding to the correct label and a regularization term. Simulation parameters can be found in appendix A (A.1). In contrast to [71] our method applies to arbitrary ratios of synaptic and membrane time-constant, as no explicit solution of the spike times depending on weights is needed. Over 300 training epochs we achieve $(95.9 \pm 0.5)\%$ (mean and standard deviation over 10 runs) accuracy on the validation dataset, which is comparable to the results reported in [71] $(95.9 \pm 0.7)\%$ on a comparable network architecture.

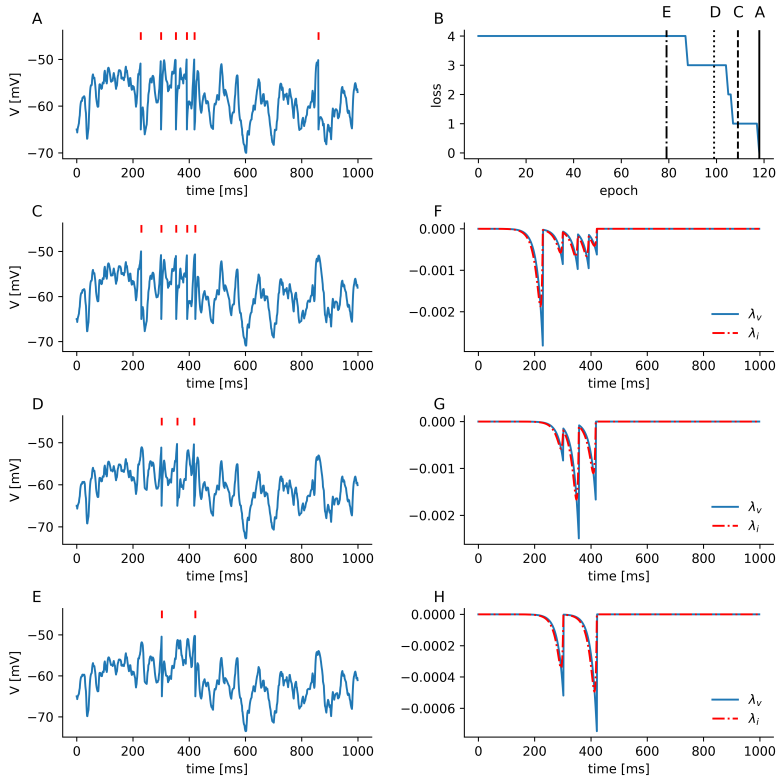


Figure 3.7: A single LIF neuron receives input from 100 poisson sources. The optimization goal is for the neuron to spike a certain number of times ($n = 6$ in the figure) within the integration window. I show training progress (**B**) and final neuron dynamics (**A**). The voltage dynamics (**C**-**E**) belong to different epochs marked in (**B**). The associated adjoint-state dynamics can be seen in (**F**-**H**). While the loss, which in this case is an integral over the elicited spikes remains constant for multiple epochs, the gradient of each weight is computed from the adjoint dynamics of λ_I and the occurence of pre-synaptic spikes (not-shown) according to eq. 3.78. The jumps in the λ_V occur due to contributions of the spike times.

3.4.3 MNIST and Fashion-MNIST

To validate the approach I consider an easy machine learning benchmark: Classifying handwritten digits. I compare with another approach to training spiking neural networks directly, namely surrogate gradients. More specifically I use the surrogate gradient introduced in [171]. I evaluate two network architectures: A recurrent LIF network with leaky-integrator readout neurons and a four-layer feed-forward convolutional network. I trained the recurrent neural network with a sequence length $T = 32\,\text{ms}$ and batch size $B = 32$. I used the ADAM optimizer

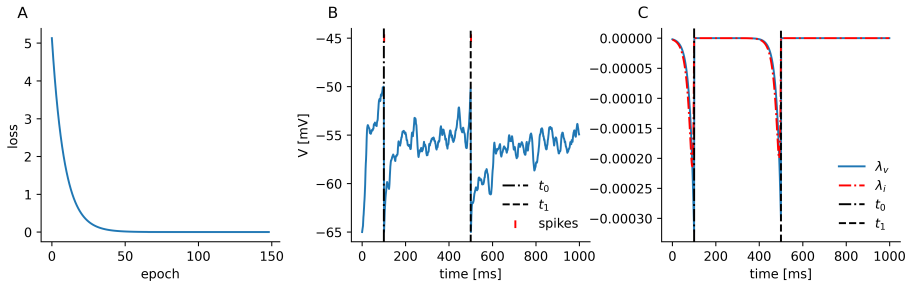


Figure 3.8: A single LIF neuron receives input from 100 poisson sources. The optimization goal is for the neuron to spike at randomly chosen spike times t_0, t_1 and not at any other times. This is realised by a distributional loss $L = \frac{1}{2}\sum_{i\in\{0,1\}}|v - v_{\text{th}}|^2\delta(t - t_i) + l_{\text{reg}}$, the regularization term $l_{\text{reg}} = \int \Theta(v - v_{\text{th}} + \epsilon)|v - v_{\text{th}}|^2$ enforces that no spikes at other spike times occur. I show the loss (**A**), final voltage trace (**B**) and adjoint variable dynamics (**C**) of one sample run of this task. The two black dashed lines in (**B**, **C**) mark the target spike times t_0, t_1. The adjoint state dynamics leads to increases of weights that received input close to the desired target spike times (with an exponential decay), the regularisation term in turn decreases any weights that received input close unwanted spike.

with parameters ADAM(lr $= 0.002, \beta = (0.9, 0.999), \epsilon = 10^{-8}$). To compute the loss I consider the voltage traces $(V_1(t), \ldots, V_{10}(t))$ of the ten readout leaky integrators (each belong to a class). The maximum accross time of each of the voltage traces is computed

$$m = (\max(V_1), \ldots, \max(V_{10})), \tag{3.110}$$

and then the cross entropy loss over these voltage peaks is computed

$$\text{loss}(m, c) = -\log\left(\frac{\exp(m_c)}{\sum_j \exp(m_j)}\right). \tag{3.111}$$

In other words this loss considers the maximal voltage of each of the readout neurons to be the class label.

The convolutional neural network was trained with a sequence length $T = 32\,\text{ms}$, batch size $B = 32$. We used the ADAM optimizer with parameters ADAM(lr $= 0.001, \beta = (0.9, 0.999), \epsilon = 10^{-8}$). The resulting accuracies on the validation data set are visualized in figure 3.10, with the same loss as for the recurrent network models. The architecture of the convolutional neural network is given in listing (A.1).

3.4.4 CIFAR-10

I also evaluated our method on a larger image recognition machine learning dataset CIFAR-10 [98]. The resulting accuracies of five training runs each are displayed in figure 3.12. I chose to evaluate the network on a sequence length $S = 32\,\text{ms}$, batch size $B = 256$ and used the ADAM optimizer with parameters

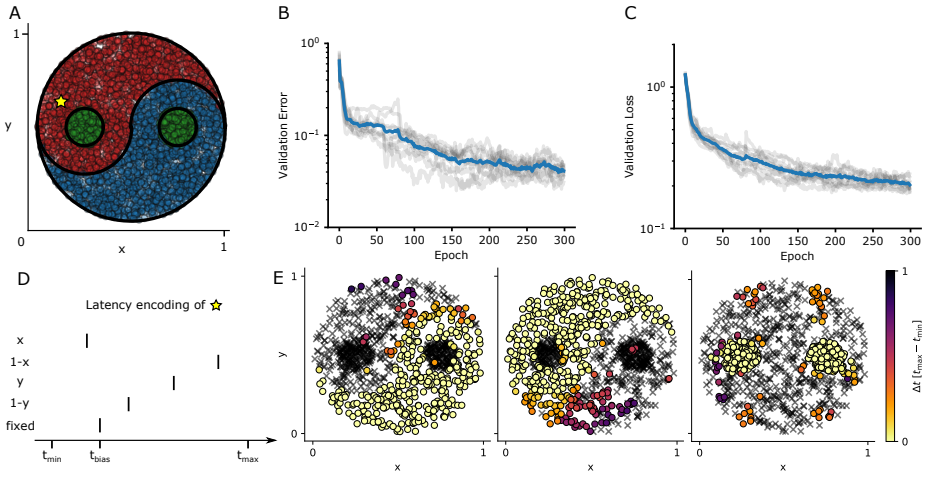


Figure 3.9: We train a two-layer spiking neural network on the two-dimensional Ying-Yang dataset, with a time to first-spike loss (Dataset and task adopted from [71]). The three classes of points (red, green, blue) are illustrated in (**A**), each point (x, y) is translated in a tuple of spike times $(Tx, T(1-x), Ty, T(1-y), t_{\text{bias}})$ (**D**), scaled by the experiment time $T = t_{\max} - t_{\min}$, with t_{bias} being one fixed additional "bias" spike. Both the validation loss and validation error decrease rapidly over 300 training epochs (**D,E**). We show 10 training runs with different random initialisation in grey and their mean in blue. The network performance after training on the validation set is illustrated in (**E**): Colored dots encode the time difference $\Delta t = t_{\text{spike}} - t_{\min}$ when one of the three label neurons spiked for a given input t_{spike} relative to the first spike any of the label neurons produced for this input $t_{\min}$, crosses indicate no spike. Figure adapted from [165].

ADAM(lr, $\beta = (0.9, 0.999), \epsilon = 10^{-8}$). I evaluate on a set of learning rates $lr = \{0.0001, 0.0005, 0.0006, 0.002\}$.

As can be seen from figure (3.12), performance of surrogate gradient and adjoint learning is comparable on the evalutated range of learning rates. To achieve results closer to the state of the art therefore most likely depends on better choices of model architecture, encoding, decoding to spikes and loss function. In particular deeper spiking neural networks are known to yield better results [51, 145]. Since the method I presented here can accomodate loss function that depend on arbitrary parameters it enables the same kind of regularisation techniques currently used to make deep spiking neural networks competitive.

3.4.5 Sum of Sines Regression Task

Timo Wunderlich contributed code to this Task. To demonstrate the ability of the method to work on a long temporal sequence I consider a simple regression task. A layer of hidden neurons ($N_{\text{hidden}} = 100$) receives poisson input with frequency of 100 Hz a single leaky-integrator's voltage trace $v(t)$ should approximate the given target function $s(t)$. The loss is given by the mean squared error

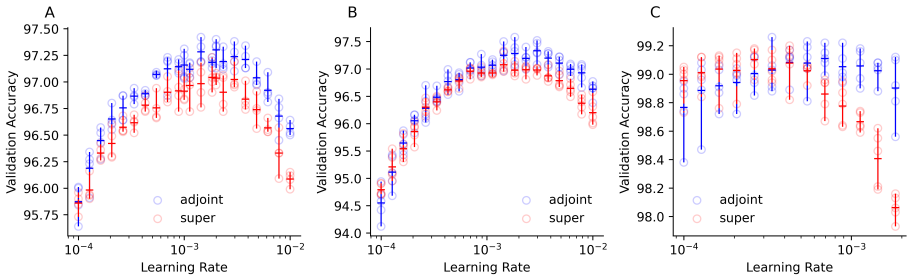


Figure 3.10: Comparison of validation accuracy on the MNIST dataset of three different networks after training for 20 epochs. In (**A,B**) I compare recurrent spiking neural networks with 100 hidden neurons with 10 leaky integrators as readout units. The learning rate of the ADAM optimizer is swept over a range $10^{-4} \dots 10^{-2}$ in logarithmically equally spaced steps and 5 models are trained for each method. In (A) the greyscale images are encoded by first transforming them to have zero mean and variance of one and then treating their pixel value as a constant input current to a leaky integrator. In (**B**) images are encoded by transforming the pixel value into a spike latency. Finally the model used in (C) is a convolutional neural network, that receives inputs encoded according to the same method used in (**A**). As can be seen in all cases (**A**-**C**) the adjoint method either outperforms or has equal performance to surrogate gradient training.

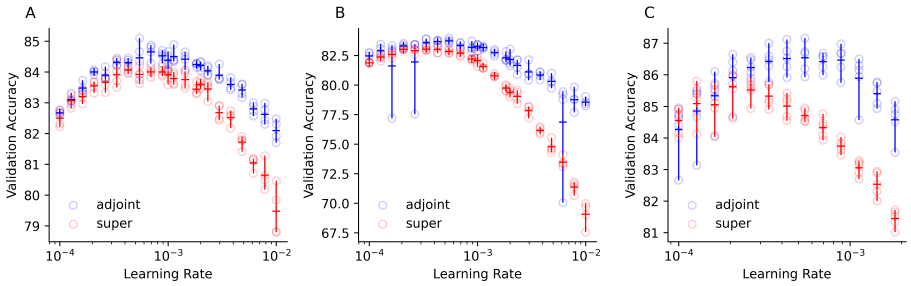


Figure 3.11: Comparison of the validation accuracy on the Fashion-MNIST dataset for three different networks after training for 20 epochs. Data was encoded by normalising it to have mean zero and variance of one and then treating the pixel values as a constant input current to a leaky integrator. In (**A**, **B**) I consider 1 and 2 layers of recurrently connected neural LIF neurons with 100 and $100, 50$ neurons per layer respectively. Readout is done in all cases by 10 Leaky-Integrators. In (**C**) the same feed-forward convolutional architecture as for the MNIST task was used. The learning rate of the ADAM optimizer is again swept over a range $10^{-4} \dots 10^{-2}$ in logarithmically equally spaced steps and 5 models are trained for each method. As can be seen in all cases (**A**-**C**) the adjoint method either outperforms or has equal performance to surrogate gradient training.

over the output voltage trace $\frac{1}{T}\int_0^T |v(t) - s(t)|^2 \mathrm{dt}$. This illustrates an example where the loss enters the adjoint dynamics (3.64). Training was done using ADAM$(0.002, \beta = (0.9, 0.999), \epsilon = 10^{-8})$, results are shown in Fig. 3.13.

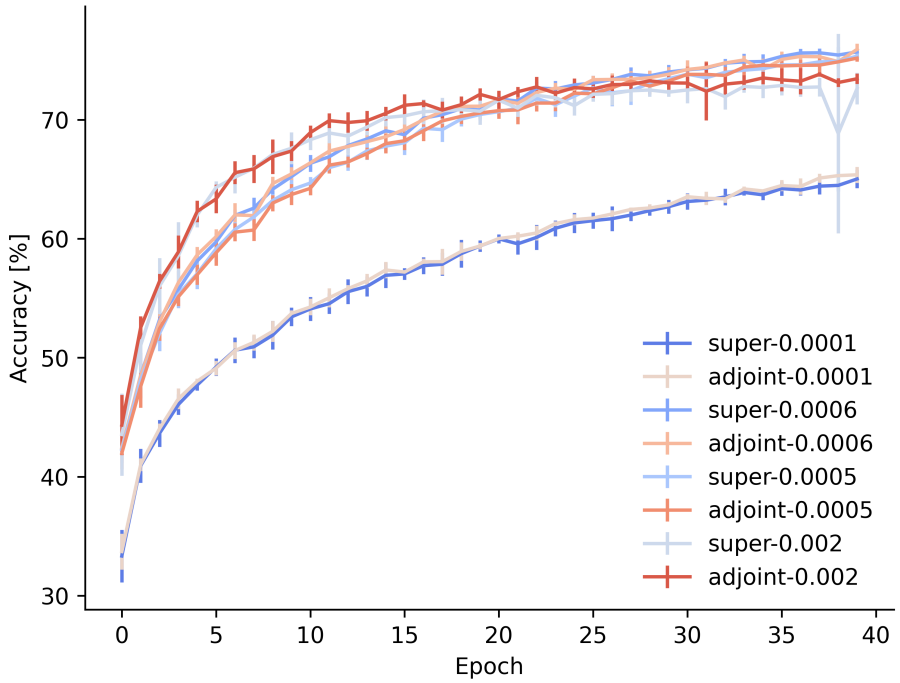


Figure 3.12: Comparison of validation accuracy obtained over 40 training epochs on the CIFAR-10 dataset using a feed forward convolutional neural network of LIF neurons. Encoding was done by normalising the RGB input data to mean 0 and standard deviation of 1 and encoding each channel subsequently using a 6 channels per pixel indexed by sign and color of the input, input values where then treated as constant input currents to two leaky integrate and fire neuron with a threshhold of 0.7. Details of the convolutional archticture are given in listing (A.2). Training was done using the ADAM(lr, $\beta = (0.9, 0.999), \epsilon = 10^{-8}$) with learning rates $\{0.0001, 0.0005, 0.0006, 0.0003\}$, overall maximum validation accuracy of 76.43 and 76.24 percent was achieved with a learning rate lr $= 0.0005$ for adjoint and super-spike training respectively.

3.4.6 Cartpole - Reinforcement Learning Task

I also evaluated the method on a reinforcement task: The task is to balance a pole connected by a joint to a cart, which moves on a frictionless track (see fig. 3.14 B). This problem was first proposed as a task in [12]. Here I use OpenAI gym mplementation of the task [28].

The four state variables are the carts position x, velocity v, the poles angle θ (deviation from vertical orientation) and angular velocity ω. For each trial they are assigned a random uniform value in $[-0.005, 0.005]$. The agent can accelerate (push) the car in either positive or negative x direction at each timestep t. For each timestep the pole has not tipped over, which is defined as the angle increasing beyond $\pm\frac{1}{15}\pi$ and the cart's position has remained in $x \in [-2.4, 2.4]$ the agent receives a reward of $r_t = 1$. The episode ends when one of these two conditions is not satisfied or the pole has been balanced for $N = 200$ timesteps.

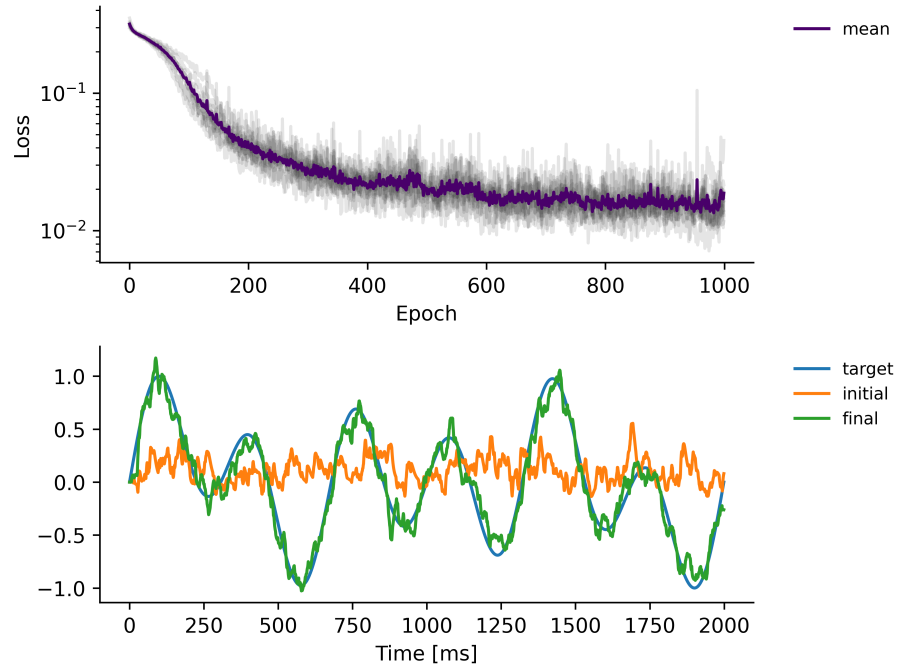


Figure 3.13: Regression loss of sum of sine task (A), shown are the mean over ten runs (violet), as well as the individual loss curves (grey). Voltage trace of the readout leaky integrator (B) on the sum of sine regression task, shown are performance before and after training.

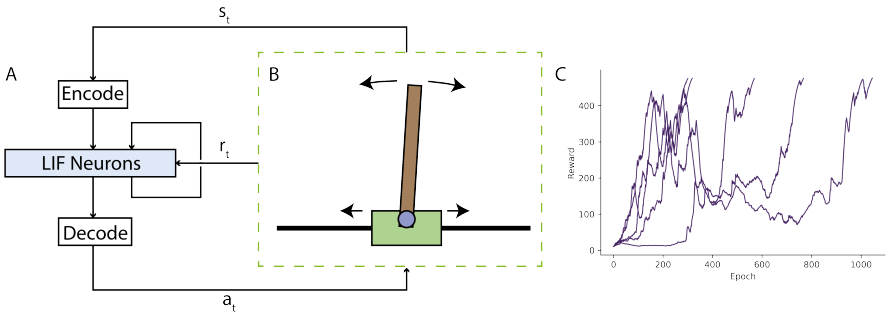


Figure 3.14: Illustration of the cartpole reinforcement learning task: A cart on a frictionless track is connected to a pole via a hinge (B). The goal is to keep the pole from tipping over. The angle θ, angular velocity ω, velocity v and position x are the input to a spiking neural network (A). It produces control output that pushes the cart. (C) shows reward over episodes of 5 different runs.

The task is considered solved if the agent has received an average return of 495.0 or higher over 100 consecutive trials.

This task is an example of a sequential decision problem, where at each timestep $t \in 0, 1, \ldots$ the agent is confronted with a state $s_t \in S$, chooses an action $a_t \in A$ and receives a real-valued reward $r_t \in \mathbf{R}$. In order to solve it with

a spiking neural network I used a modified version of the REINFORCE policy gradient algorithm [163] as implemented in [8]. I encode the 4 state variables as 8 input currents

$$(\text{relu}(x), -\text{relu}(x), \text{relu}(v), -\text{relu}(v), \text{relu}(\theta), -\text{relu}(\theta), \text{relu}(\omega), -\text{relu}(\omega)) \tag{3.112}$$

to LIF neurons for 40 ms, these encoding neurons are connected to 100 recurrently connected LIF neurons, which in turn are connected to 2 readout Leaky Integrators (fig. 3.14 A). The connection from recurrent neurons to readout neurons is subject to dropout per timestep with probability $p = 0.2$. Action selection is done by sampling from the probability distribution

$$\left(\frac{\exp(m_0)}{\sum_j \exp(m_j)}, \frac{\exp(m_1)}{\sum_j \exp(m_j)} \right) \tag{3.113}$$

where $m_j = \max_t(v_j(t)), j = 0, 1$ are the maximum of the voltage of the two readout neurons during the integration window of 40 ms. Fig. 3.14 C shows for 5 runs the reward after a given number of Epochs (trials).

3.5 Conclusions

In this chapter I have introduced a method well-known in the optimal control literature to the study of learning in spiking neural networks. In contrast to recent deep-learning inspired methods the resulting learning dynamics can easily be formulated in continuous time and depends only on the state variables at spike times in the case of linear neuron dynamics such as the leaky integrate and fire neuron. Moreover the dynamics of the adjoint-state variables follows a similar time course to the forward differential equations, with jumps in the adjoint-state at spike times in the forward direction.

More generally it suggests the introduction of the notion of Neural Event Processing Element, which are systems of ordinary differential equations of small dimension, which are only coupled at events. Because of the general nature of the method it is possible to treat a whole class of neuron models on an equal footing. It should be possible to implement them generically while still retained the advantages of the sparse and event based coupling of gradient computation, which follow from the restrictions imposed in the definition. These properties should also make it relatively easy to incorporate the method in event based simulators such as NEST [67] or arbor [5]. Moreover in the context of neuromorphic hardware such as [60, 44, 2] it should enable efficient in-the-loop [40, 142] or even on-chip optimization, because only sparse information about the state needs to be stored per-neuron.

Another particular attractive avenue of research is then to consider systems where the equations are partially fixed known from physical considerations up to free adjustable parameters (for example memristive crossbar arrays, quenched josephson junctions, spin transfer nano-oscillators [135]) and then consider parametrised and event based coupling of such systems. See [109] for a recent review of such approaches. In particular it would be interesting to find physical systems that naturally accomodate the adjoint equations.

Clearly the method as presented here can't be taken as a biologically plausible model of learning. One obvious obstacle is that the time integration of

the adjoint-variables happens backwards in time with respect to the neuron dynamics. One promising research direction is therefore to combine the method presented here with methods of breaking the forward-backward dependence in (spiking) deep-neural networks, such as differentiable neural interfaces [86] or other recent work in that direction such as [16, 14] (based on surrogate gradients) and dendritic learning rules [154]. I present a derivation of a simple online learning rule for (adaptive) LIF neurons based on the forward sensitivity equations with jumps in chapter 5. Another promising direction is to use the adjoint equations in a Meta-Learning or Learning to Learn setting, we can then optimize biologically plausible plasticity rules, which typically also depend on discontinuities state evolution, using gradients derived from the adjoint dynamics in the outer loop. I will discuss this briefly in chapters 6 and 7.

In a theoretical direction several other developments are plausible: The method as presented here can be related to surrogate gradient based approaches via the well known concept of a mollifier. Smoothing out the jumps in the forward direction with such mollifiers, yields differential equations to which the adjoint method without jumps can be applied, the resulting adjoint-equations depend on derivatives of the mollifier, it should then be possible to show that solutions converge in the sense of distributions to solutions of the equations presented here.

The numerical experiments on machine-learning datasets, especially on the larger CIFAR-10 dataset, don't reach state of the art performance. Since we see comparable performance between surrogate gradient and adjoint learning, I believe this is most likely due to choices in network architecture, as well as how I encode to and from spikes. Nevertheless further work is needed to achieve state of the art results in such benchmarks.

Chapter 4

Linear Adjoint Dynamics with Jumps

Both the adaptive LIF neuron and LIF neuron dynamics are examples of time-invariant linear ordinary differential equations. Such equations have explicit solutions, which can be used to derive exact integration schemes. In the context of spiking neural networks this was pointed out in [136]. As it turns out the adjoint dynamics of a linear time invariant system can also be solved analytically. If one introduces jumps as is necessary for the treatment of the LIF and adaptive LIF neurons this still hold with the exception that now the adjoint dynamics jumps at jump times computed during the forward integration. I arrive at explicit formulas for the gradients of linear time-invariant ordinary differential equations with jumps with respect to arbitrary loss functions and arbitrarily specified jump and transition functions. These generalise the explicit formulas presented in chapter 3.

4.1 Background

Consider a linear ordinary differential equation given by

$$\dot{y} = Ay + x, \tag{4.1}$$

with A an $n \times n$ dimensional matrix, $y(t) \in \mathbf{R}^n$ and $x(t) \in \mathbf{R}^n$ some time dependent input vector then an explicit solution is given by

$$y(t') = e^{A(t'-t)}y(t) + \int_t^{t'} e^{A(t'-\tau)}x(\tau)\mathrm{d}\tau \tag{4.2}$$

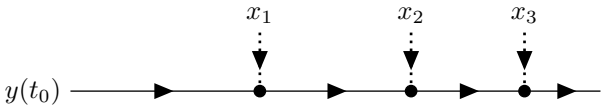
Now assume that the input to the system is given as a (weighted) sum of dirac delta distributions (weighted spikes)

$$x(t) = \sum_k x_k \delta(t - t_k) \tag{4.3}$$

then

$$y(t') = e^{A(t'-t)} + \sum_k e^{A(t'-t_k)}x_k \tag{4.4}$$

This situation can be illustrated by the following diagram



(4.5)

That is we can divide the integration into segments interrupted by external events, since the ODE is linear and time indepedent it can be restarted at each external event.

If we assume that jump conditions are given by functions

$$j(y^-, p, t^f) = 0 \tag{4.6}$$

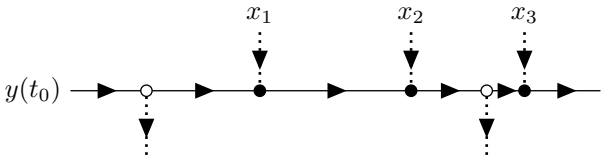
resulting in a transition

$$y^+ = \theta(y^-, p, t^f) \tag{4.7}$$

we can split up the integration at the transition times $t_j^f, j = 1 \dots N$ and get a recursive formula for the state at time t'

$$y(t') = e^{A(t'-t_j^f)}\theta(y_j^-, p, t_j^f) + \sum_{k, t_k > t_j^f} e^{A(t'-t_k^f)} x_k, t' < t_{j+1}^f \tag{4.8}$$

These internal jumps can be incorporated in a diagram like (4.5) as follows:



(4.9)

the open dots indicate the times t_i^f, when one of the jump conditions is satisfied. I will now turn to the adjoint dynamics of such a system.

4.2 Adjoint Equations

The adjoint equation of the system (4.1) are given by (with $s = T - t$) and $f' = \frac{df}{ds}$:

$$\lambda' = A^T\lambda - \partial_y l, \tag{4.10}$$

which again has the exact solution

$$y(s') = e^{A^T(s'-s)}y(s) - \int_s^{s'} e^{A^T(s'-\sigma)}\partial_y l \mathrm{d}\sigma \tag{4.11}$$

Now the general formula for the jumps in the adjoint variables that I derived in chapter 3 and gradients can be specialised to the case at hand:

$$\begin{aligned}(\lambda^-)^T = (\lambda^+)^T &\left[\left[Ay^+ - \partial_{y^-}\theta Ay^-\right]\frac{\partial_{y^-} j}{\partial_{y^-} jAy^-} + \partial_{y^-}\theta\right] \\ &+ (l^- - l^+ + \partial_{t_k} l_p)\frac{\partial_{y^-} j}{\partial_{y^-} jAy^-}\end{aligned} \tag{4.12}$$

$$\xi_p = -\left[(l^- - l^+ + \partial_{t_k} l_p) + \lambda^+\left[Ay^+ - \partial_{y^-}\theta Ay^-\right]\right]\frac{\partial_p j}{\partial_{x^-} j\dot{x}^-} - \lambda^+\partial_p\theta \tag{4.13}$$

Therefore we can write

$$(\lambda^-)^T = \theta^*(\lambda^+, y^+, y^-, A, p) \tag{4.14}$$

and we can compute in the backward pass the gradient contributions efficiently by the following recursive formula

$$\lambda(s') = e^{A^T(s'-s^f_{j+1})}\theta^*(\lambda^+_{j+1}, y^+_{j+1}, y^-_{j+1}, A, p) - \int_{s^f_{j+1}}^{s'} e^{A^T(s'-\sigma)}\partial_y l \mathrm{d}\sigma \tag{4.15}$$

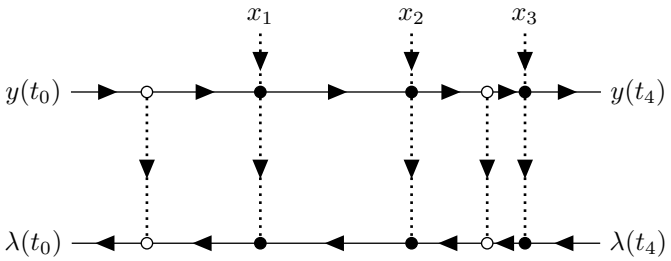
with $s^f_j = T - t^f_j$. Since the jump times are known from the forward integration we only need to compute

$$\lambda^-(s^f_j) = e^{A^T(s^f_j-s^f_{j+1})}\theta^*(\lambda^+_{j+1}, y^+_{j+1}, y^-_{j+1}, A, p) - \int_{s^f_{j+1}}^{s^f_j} e^{A^T(s^f_j-\sigma)}\partial_y l \mathrm{d}\sigma \tag{4.16}$$

and the gradient of the loss with respect to the parameters is given by

$$\begin{aligned} d_p L = &\int_0^T \partial_p l - \langle \lambda, \partial_p A(p) y\rangle] \mathrm{dt} \\ &+ \sum_{t_k} - \left[(l^- - l^+ + \partial_{t_k} l_p) + \lambda^+ \left[Ay^+ - \partial_{y^-}\theta A y^-\right]\right] \frac{\partial_p j}{\partial_{x^-} j\dot{x}^-} - \lambda^+ \partial_p \theta \end{aligned} \tag{4.17}$$

Including the adjoint dynamics the diagram 4.9 becomes



(4.18)

The state dynamics of the adjoint variable is "backwards" in time and is non-linear at events in the forward direction. In the spiking neuron models I discussed in the previous chapter the equations *only* coupled at the jumps indicated by dotted lines.

Now assuming that the transition is an affine transformation

$$y^+ = By^- + c \tag{4.19}$$

and the jump condition is given by some other affine map

$$j(y^-, D) = Dy^- + e \tag{4.20}$$

then

$$(\lambda^-)^T = (\lambda^+)^T \left[\left[Ay^+ - BAy^-\right] \frac{D}{DAy^-} + B\right] + (l^- - l^+ + \partial_{t_k} l_p)\frac{D}{DAy^-} \tag{4.21}$$

$$\xi_p = -\left[(l^- - l^+ + \partial_{t_k} l_p) + (\lambda^+)^T \left[Ay^+ - DAy^-\right]\right] \frac{\partial_p j}{\partial_{x^-} j\dot{x}^-} - \lambda^+ \partial_p \theta \tag{4.22}$$

and the gradient of the loss with respect to the parameters is given by

$$d_p L = \int_0^T \partial_p l - \langle \lambda, \partial_p A(p) y \rangle] \mathrm{dt} + \sum_{t_k} - \left[(l^- - l^+ + \partial_{t_k} l_p) + (\lambda^+)^T \left[A y^+ - D A y^- \right] \right] \frac{\partial_p j}{\partial_{x^-} j \dot{x}^-} - \lambda^+ \partial_p \theta \tag{4.23}$$

More generally we could both parametrise the affine transformation and the jump condition by an arbitrary artificial neural network while retaining the simple form of the exact integration above.

4.3 Conclusions

The results presented in this chapter hopefully clarify the somewhat uninspired calculations necessary to arrive at the results for the LIF and adaptive LIF neurons derived in chapter 3. It also makes clear under what circumstances an efficient implementation of the dynamics using exact integration is possible. In particular the adjoint dynamics can be iteratively determined by equation (4.15). Moreover the time evolution of a quantum system with time independent Hamiltonian is also an example of this kind of ordinary differential equation

$$i\hbar \partial_t \psi = H\psi \tag{4.24}$$

and so is the von-Neumann equation

$$i\hbar \partial_t \rho = [H, \rho]. \tag{4.25}$$

This suggests the study quantum systems coupled via events to classical systems. Measurements lead to some projection P_i of the density matrix

$$\rho_i' = \frac{P_i \rho P_i}{\mathrm{tr}\left[\rho P_i\right]}. \tag{4.26}$$

which can be considered to a transition of the kind considered above. The use of analog quantum systems to implement neuromorphic computing was also recently discussed in [109].

Chapter 5

Online Learning for Spiking Point Neurons

In the previous chapter I derived an exact method to compute gradients and optimize point neuron models. During the course of the derivation we also arrived at equations for forward sensitivity equations and their jumps. While the adjoint sensitivity method itself allows one to compute exact gradients, it suffers from the fact that it has to be computed in reverse time relative to the original differential equation. While the forward sensitivity equations don't suffer from this problem, a naive implementation scales as a product of dynamic variables and parameters. This is prohibitive for large numbers of parameters.

Recent work in the context of spiking neurons solved this scalability problem for time discretized systems of equations with smoothed derivatives [15]. This algorithm called "e-prop 1" is a truncation of the real-time recurrent learning algorithm (RTRL) [134, 164]. This same truncation had been used before to train LSTM (c.f. Figure 2 and eq. (10,11,14,15) in [80]), as well as in subsequent papers [64], where it also was pointed out that the approximation is local in space and time. The forward sensitivity equations can be seen as a continuous time analogue of the RTRL equations. The goal in this chapter then is to derive a truncation similar to the one done in [15] but in continuous time and without resorting to approximations of spike derivatives. The crucial insight is that spiking neural network models only couple through events and therefore the sensitivity equations largely develop independently. In point neuron models the number of dynamical variables K_i per neuron $i = 1 \dots N$ is small. The method I will derive has $K_i \times N_i$ additional dynamical variables, where N_i is the number of presynaptic partners of neuron i. Since spiking neural networks are sparsely connected, this is a *dramatic* reduction of the N^3 additional dynamical variables that the full forward sensitivity equations track. Without resorting to discretization or smoothing of derivatives, it is possible to derive three-factor learning rules from the forward sensitivity equations derived in the preceeding two chapers.

I will begin with the two explicit models I derived in chapter 3 the LIF neuron model and the ALIF neuron model. Recall that the gradient contributions for

the LIF neuron were given by

$$\frac{\mathrm{d}L}{\mathrm{d}w_{ij}} = \sum_{k=0}^{N_{\text{post}}} \left[\int_{t_k^{\text{post}}}^{t_{k+1}^{\text{post}}} \left[\frac{\partial l}{\partial V} \cdot s_V + \frac{\partial l}{\partial I} \cdot s_I + \frac{\partial l}{\partial w_{ij}} \right] \right] \tag{5.1}$$

$$- \sum_{k=0}^{N_{\text{post}}} \left[\frac{\partial l_p}{\partial t_k} + l^+ - l^- \right] \frac{(s_V^-)_{n(k)}}{\dot{V}_{n(k)}^-} \tag{5.2}$$

where the sum runs over the spike times t_k and $n(k)$ is the neuron that spiked at time t_k. In order for that part of the expression to be locally computable we conclude that

$$(s_V^-)_{n(k)} = ((s_V^-)_{n(k)})_{ij} \approx \begin{cases} 0 & \text{if } i \neq n(k), j \notin N_{\text{pre}}(n(k)) \\ ((s_V^-)_{n(k)})_{n(k)j} & \forall j \in N_{\text{pre}}(n(k)) \end{cases} \tag{5.3}$$

where $N_{\text{pre}}(n(k))$ is the set of all presynaptic neurons of neuron $n(k)$. Then only gradients of weights $w_{n(k)j}$ are modified on a spike of neuron $n(k)$. We can then define projections

$$(e_V)_n = P_n((s_V)) = P_n((s_V)_{ijk}) = \begin{cases} 0 & \text{if } k \neq n, i \neq n, j \notin N_{\text{pre}}(n(k)) \\ (s_V)_{njn} & \forall j \in N_{\text{pre}}(n) \cup \{n\} \end{cases} \tag{5.4}$$

and $P = \bigoplus_n P_n$. Inspired by [15] I call these variables *eligibility traces*. This projection defines a new set of equations based on the forward sensitivity equations

$$\tau_v \dot{e}_V = \tau_v P(\dot{s}_V) = -P(s_V) + P(s_I) = -e_V + e_I \tag{5.5}$$

$$\tau_s \dot{e}_I = \tau_s P(\dot{s}_I) = -P(s_I) = -e_I \tag{5.6}$$

Each LIF neuron has $K_i = 2$ dynamical variables and therefore we get a total of

$$K \sum_i (N_i^{\text{pre}} + 1) < 2(N+1)N \tag{5.7}$$

additional dynamical variables to track. In sparse networks this bound is very loose but in any case much better than N^3.

Since the projections P are linear, we can also easily apply them to the transition equations

$$P_n((s_V^+)_n) = P_n\left(P_n((s_V^-)_n) + \frac{1}{\tau_v}(\vartheta + w_{nn}) P_n\left(\frac{(s_V^-)_n}{\dot{V}_n^-} \right) \right) \tag{5.8}$$

$$P_m((s_V^+)_m) = P_m\left((P_m((s_V^-)_m) + \frac{1}{\tau_v} w_{nm} P_n\left(\frac{(s_V^-)_n}{\dot{V}_n^-} \right) \right), \forall m \neq n \tag{5.9}$$

$$P_m((s_I^+)_m) = P_m\left(P_m((s_I^-)_m) - \frac{1}{\tau_s} w_{nm} P_n\left(\frac{(s_V^-)_n}{\dot{V}_n^-} \right) + P_m(A_{nm}\delta_{in}\delta_{jm}) \right), \tag{5.10}$$

where A_{nm} denotes the (directed) adjacency matrix. Note that the terms

$$\frac{1}{\tau_v} w_{nm} P_n\left(\frac{(s_V^-)_n}{\dot{V}_n^-} \right), -\frac{1}{\tau_s} w_{nm} P_n\left(\frac{(s_V^-)_n}{\dot{V}_n^-} \right) \tag{5.11}$$

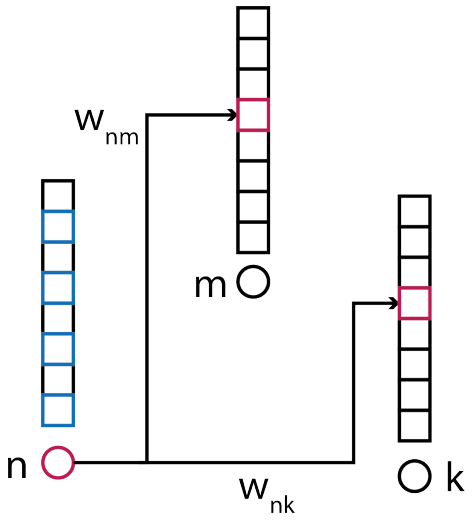


Figure 5.1: When a neuron n (indicated by a red circle) fires, the current eligibility traces s_I in all post-synaptic neurons are affected (red squares) and jump by one. At the same time the voltage eligibility traces s_V of the neuron n that fired are changed (blue squares) according to eq. (5.16).

get projected to zero. We are therefore left with

$$(e_V^+)_n = (e_V^-)_n + \frac{1}{\tau_v}(\vartheta + w_{nn})\frac{(e_V^-)_n}{\dot{V}_n^-} \tag{5.12}$$

$$(e_V^+)_m = (e_V^-)_m, \forall m \neq n \tag{5.13}$$

$$(e_I^+)_n = (e_I^-)_n - \frac{1}{\tau_s} w_{nn}\frac{(e_V^-)_n}{\dot{V}_n^-} + \delta_{in}\delta_{jn} \tag{5.14}$$

$$(e_I^+)_m = (e_I^-)_m + A_{nm}\delta_{in}\delta_{jm}, \forall m \neq n \tag{5.15}$$

If the neurons have no autapses $w_{nn} = 0, A_{nn} = 0$, the somatic voltage and current eligibilities will remain zero if they were zero initially and the equations simplify even further

$$(e_V^+)_n = (e_V^-)_n + \frac{1}{\tau_v}\vartheta\frac{(e_V^-)_n}{\dot{V}_n^-} \tag{5.16}$$

$$(e_V^+)_m = (e_V^-)_m, \forall m \neq n \tag{5.17}$$

$$(e_I^+)_m = (e_I^-)_m + A_{nm}\delta_{in}\delta_{jm}, \tag{5.18}$$

This situation is illustrated in figure (5.1). We therefore arrive at the following approximation of the gradient

$$\frac{\mathrm{d}L}{\mathrm{d}w_{ij}} = \sum_{k=0}^{N_{\text{post}}} \left[\int_{t_k^{\text{post}}}^{t_{k+1}^{\text{post}}} \left[\frac{\partial l}{\partial V} \cdot e_V + \frac{\partial l}{\partial I} \cdot e_I + \frac{\partial l}{\partial w_{ij}} \right] \right] \tag{5.19}$$

$$- \sum_{k=0}^{N_{\text{post}}} \left[\frac{\partial l_p}{\partial t_k} + l^+ - l^- \right] \frac{(e_V^-)_{n(k)}}{\dot{V}_{n(k)}^-} \tag{5.20}$$

Starting from the equation for the gradient we can apply the same reasoning to the adaptive LIF neuron

$$\frac{\mathrm{d}L}{\mathrm{d}w_{ij}} = \sum_{k=0}^{N_{\text{post}}} \left[\int_{t_k^{\text{post}}}^{t_{k+1}^{\text{post}}} \left[\frac{\partial l}{\partial V} \cdot s_V + \frac{\partial l}{\partial I} \cdot s_I + \frac{\partial l}{\partial B} \cdot s_B + \frac{\partial l}{\partial w_{ij}} \right] \right] \tag{5.21}$$

$$- \sum_{k=0}^{N_{\text{post}}} \left[\frac{\partial l_p}{\partial t_k} + l^+ - l^- \right] \frac{(s_V^- + s_B^-)_{n(k)}}{(\dot{V}^- + \dot{B}^-)_{n(k)}} \tag{5.22}$$

this suggests to again define

$$e_V = P(s_V) \tag{5.23}$$
$$e_I = P(s_I) \tag{5.24}$$
$$e_B = P(s_B) \tag{5.25}$$
$$\tag{5.26}$$

and we immediately obtain from eqs. (3.92), (3.93), (3.94):

$$\tau_v \dot{e}_V = -e_V + e_I \tag{5.27}$$
$$\tau_s \dot{e}_I = -e_I \tag{5.28}$$
$$\tau_b \dot{e}_B = -e_B \tag{5.29}$$

and by the same rationale as before (assuming $w_{nn} = 0, A_{nn} = 0$), we get

$$(e_V^+)_n = (e_V^-)_n + \frac{1}{\tau_v} \vartheta \frac{(e_V^- + e_B^-)_n}{(\dot{V}^- + \dot{B}^-)_n} \tag{5.30}$$
$$(e_V^+)_m = (e_V^-)_m, \forall m \neq n \tag{5.31}$$
$$(e_I^+)_m = (e_I^-)_m + A_{nm}\delta_{im}\delta_{in} \tag{5.32}$$
$$(e_B^+)_n = (e_B^-)_n - \frac{1}{\tau_b} \beta \frac{(e_V^- + e_B^-)_n}{(\dot{V}^- + \dot{B}^-)_n} \tag{5.33}$$
$$(e_B^+)_m = (e_B^-)_m, \forall m \neq n \tag{5.34}$$

Using these equations we can compute the gradient w.r.t. the loss approximately as

$$\frac{\mathrm{d}L}{\mathrm{d}w_{ij}} = \sum_{k=0}^{N_{\text{post}}} \left[\int_{t_k^{\text{post}}}^{t_{k+1}^{\text{post}}} \left[\frac{\partial l}{\partial V} \cdot e_V + \frac{\partial l}{\partial I} \cdot e_I + \frac{\partial l}{\partial B} \cdot e_B + \frac{\partial l}{\partial w_{ij}} \right] \right] \tag{5.35}$$

$$- \sum_{k=0}^{N_{\text{post}}} \left[\frac{\partial l_p}{\partial t_k} + l^+ - l^- \right] \frac{(e_V^- + e_B^-)_{n(k)}}{(\dot{V}^- + \dot{B}^-)_{n(k)}} \tag{5.36}$$

Both derivations have in common that in order to derive the approximation the equations had to only couple at the transition times. In structured neurons as I will discuss in the next chapter 6 this still holds true for the variables of the pre- and post-synaptic density. The equations as I have derived them here have the advantage that they rely only on spike times and synaptic parameter gradients that are computed online. This makes them candidates for an (approximate) implementation in neuromorphic hardware.

The eligibility equations as I derived them here have the disadvantage that error signals only can propagate forward, which necessitates recurrence. One way of sidestepping this issue and still retain the desirable online-learning capabilities is to proceed in two steps. First we can observe that during the derivation of the adjoint equations, we could have chosen to only perform partial integration on some of the variables in (3.5). This leaves us with some variables whose sensitivities are integrated forward in time and others for which adjoint sensitivities are computed. In a second step we can then attempt to locally predict the future value of the adjoint variable and thereby again decouple the forward and backward dynamics.

Chapter 6

Adjoint Equations of Structured Neurons

Although the point neuron models I considered in the previous chapter are what is typically implemented in order to conduct network level simulations of spiking neural networks, they only crudely approximate biological neurons and their bewildering diversity of signalling pathways and molecular mechanisms. A huge advantage of the adjoint method is the applicability to arbitrary complicated hybrid systems of ordinary differential equations and even partial differential equations. To illustrate this I turn in this chapter to multicompartment neuron models, which are a typical way of modelling more biologically realistic neuron models [66].

I indicate how to generalise adjoint equations to arbitrary multi-compartment neuron models. This allows the computation of parameter gradients of all parameters appearing in a multi-compartment model according to arbitrary loss functions. In particular I make only very weak assumptions on the coupling of the dynamics in the pre- and post-synaptic density. That way the adjoint dynamics and parameter gradients I derive are applicable to a very general class of synaptic transmission models, including models incorporating plasticity. As one would expect coupling of the adjoint dynamics at jumps is in the reverse direction that is from post- to pre-synaptic density and very weak assumptions on the jump condition in the forward integration. Using the approach presented here one can therefore study meta-plasticity or optimise synapse models based on experimentally observed data.

While point neuron models model the membrane potential by a single capacitor, multi-compartment neuron models take into account the morphology of the cell and assign varying conductances to segments of the cell. Each segment is taken to be a cylindrical shape with constant trans-membrane conductance with the exeption of potentially present voltage-dependent Ion channels.

The main insight is that any such multi-compartment model couples the different compartments by a symmetric matrix of conductances. Geometrically it means that errors represented by the adjoint-state variables to the membrane voltage in each compartment can be thought to propagate along the same dendritic tree structure. This resolves in part the "weight transport problem" assuming that there is a way to signal from post- to pre-synaptic synapse. I

illustrate the approach on the multi-compartment model implemented in the BrainScaleS-2 neuromorphic chip and a model of CA3 hippocampal Pyramidal neurons.

6.1 From Neuron Morphology to Equations

A discretised cable equation is formulated by considering a subdivision of the dendritic tree into N sections [66]. The currents flowing between sections are determined by Kirchoff's law. A sparse $N \times N$ matrix g of conductances determines the coupling between the sections:

$$C_k \frac{dV_k}{dt} = \sum_{i \neq k} g_{ki}(V_i - V_k) - I_k^{\text{ion}} - I^{\text{syn}}(V_k). \tag{6.1}$$

Here C_k is the sections capacitance (summation over the capacitance is never implied), which just as the conductances g_{ij} are determined by the properties of this section of the neuron (for the most part its geometry).

The current $I_k^{\text{ion}}(V)$ is a sum of contributions over the different Ion channels and current contributions from the synapses at this neuron section. Dendritic trees have only branch points with at most three branching sections. So for a given neuron segment k the row g_{ki} has $1, 2$ or 3 non-zero entries.

In order to simplify the differential equation for the voltage above we can introduce the following symmetric matrix

$$G_{ij} = g_{ij} - \delta_{ij} \sum_k g_{ik}, \tag{6.2}$$

then we can write the equation above more compactly as

$$C_k \frac{dV_k}{dt} = G_{ik} V_i - I^{\text{ion}}(V_k) - I^{\text{syn}}(V_k). \tag{6.3}$$

Ion channels are typically modelled in the following way: The Ion current is given by

$$I_c = \overline{g}_c p(x)(V - e_c) \tag{6.4}$$

where $p(x)$ describes the fraction of open channels of this type in the given segment, depending on state variables $x = (x_1, \ldots, x_n)$. Each of the state variables evolves according to a differential equation

$$\dot{x} = \alpha_x(V, S) \cdot (1 - x) - \beta_x(V, S) \cdot x. \tag{6.5}$$

The gating functions α and β depend on the voltage V of the segment and potentially on concentrations S_i of other Ions or molecules. The functions α and β are chosen in such a way, that the fraction

$$x_\infty(V, S) = \frac{\alpha_x(V, S)}{\alpha_x(V, S) + \beta_x(V, S)} \tag{6.6}$$

lies between 0 and 1. That way the fraction of open channels is typically expressed as a monomial

$$p(x_1, \ldots, x_n) = x_1^{k_1} \ldots x_n^{k_n}. \tag{6.7}$$

The synapse currents are assumed to be of the form

$$I^{\text{syn}} = g^{\text{syn}}(V^{\text{syn}}, A)(V^{\text{syn}} - e^{\text{syn}}) \tag{6.8}$$

with some synapse type specific ordinary differential equations

$$\dot{A} = f(V, A, p) \tag{6.9}$$

here A is a vector of synapse state variables and the function f might depend on the post- and pre-synaptic voltage. Before we discuss the adjoint equations of such a model, we will give a concrete example in the next section.

6.2 Model of a Pyramidal Neuron

To ground the discussion we will now consider a concrete model first put forward by [152] and recently used in modified form in [69]. Here the Ion currents at the somatic compartment (where action potentials are invoked) are given by

$$I^{\text{ion}}(V) = \overline{g_{\text{Na}}} h m^2 (V - e_{\text{Na}}) + \overline{g_K} n (V - e_K) + \overline{g_L}(V - e_L) \tag{6.10}$$

with gating variables

$$\dot{m} = \alpha_m(V)(1 - m) - \beta_m(V)m \tag{6.11}$$
$$\dot{h} = \alpha_h(V)(1 - h) - \beta_h(V)h \tag{6.12}$$
$$\dot{n} = \alpha_n(V)(1 - n) - \beta_n(V)n \tag{6.13}$$

If one defines the characteristic time τ_x and equilibrium quantities x_∞ by

$$\tau_x(v) = \frac{1}{\alpha_x(v) + \beta_x(v)} \tag{6.14}$$
$$x_\infty(v) = \frac{\alpha_x(v)}{\alpha_x(v) + \beta_x(v)} \tag{6.15}$$

the three equations can alternatively be written as

$$\dot{m} = (m_\infty(v) - m)/\tau_m(v) \tag{6.16}$$
$$\dot{h} = (h_\infty(v) - h)/\tau_h(v) \tag{6.17}$$
$$\dot{n} = (n_\infty(v) - n)/\tau_n(v). \tag{6.18}$$

The shape of the gating functions α_x, β_x is experimentally determined. Conceptually each of them models a particular Ion channel. At the soma [69] uses a subset of the numerical values presented in [152], we summarize them in table 6.1. The gating variables n, m, h are used to determine the conductances and ion currents at the soma

$$g_{\text{Na}} = \overline{g_{\text{Na}}} h m^2 \qquad I_{\text{Na}} = g_{\text{Na}}(v - e_{\text{Na}}) \tag{6.19}$$
$$g_K = \overline{g_K} n \qquad I_K = g_{\text{K}}(v - e_K) \tag{6.20}$$
$$I_L = \overline{g_L}(v - e_L) \tag{6.21}$$

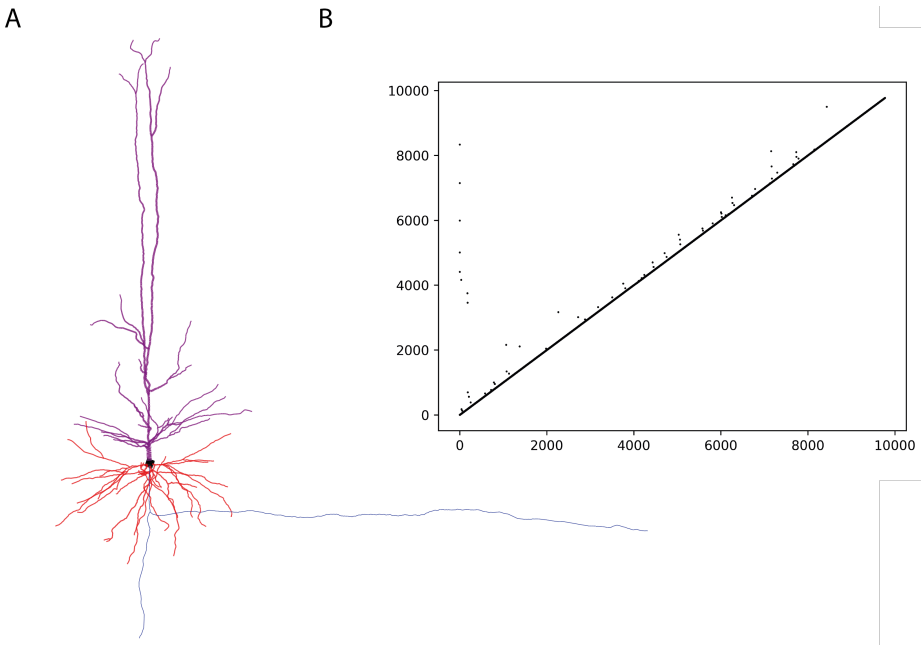


Figure 6.1: Morphology of a human pyramidial neuron in the neocortex (A). Proximal dendrites are colored in red, distal dedrites in violet, the axon in blue and the soma is indicated by a black dot. Data obtained from [9, 10], based on original data published by the Allen Institute [95]. The morphology is stored in discretised form, in total this particular neuron morphology has 9500 sections. Their connectivity results in adjacency matrix of a directed acyclic graph, which is visualised in (B). Most of the sections are directionally connected to their nearest neighbor, this explains the diagonal entries. Each column has at most two non-zero entries. The columns that contain two entries correspond to branch points.

Channel / Variable	Forward ($\alpha_x(v)$)	Backward ($\beta_x(v)$)
g_{Na}/m	$\frac{13.1-vt}{\exp((13.1-vt)/4)-1}$	$0.28\frac{vt-40.1}{\exp((vt-40.1)/5)-1}$
g_{Na}/h	$0.128\exp((17-vt)/18)$	$\frac{4}{(1+\exp((40-vt)/5)}$
g_K/n	$0.016\frac{35.1-vt}{\exp((35.1-vt)/5)-1}$	$1/4\exp((20-vt)/40$

Table 6.1: Numerical values for the voltage dependence of the Ion channel gating variables. Table adapted from [152].

Ion Channel (x)	Conductance $g_x[\mathrm{mS}/\mathrm{cm}^2]$	Reversal Potential $e_x[\mathrm{mV}]$
Na	30	90
K	15	-80
L	0.14	-62

Table 6.2: Summary of Ion Channel reversal potentials and conductances, adapted from [69].

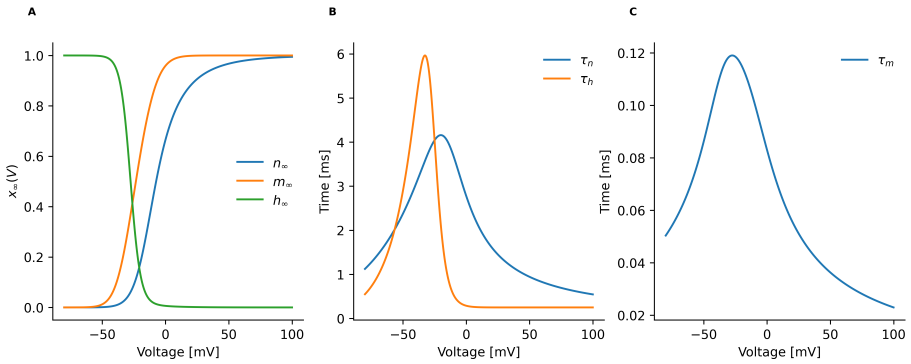


Figure 6.2: Equilibrium values x_∞ and time constants τ_x or the three Ion channel state variables used in the model.

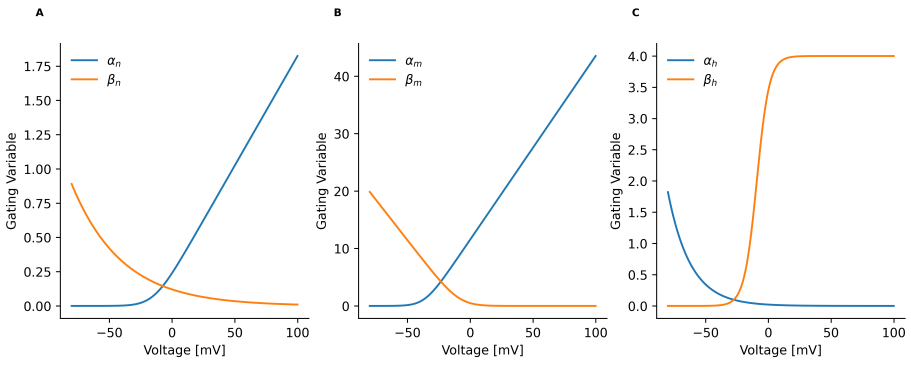


Figure 6.3: Gating variables α_x, β_x used in the model.

The total currents and conductance at the somatic segment are therefore given by

$$I = I_L + I_K + I_{\mathrm{Na}} \tag{6.22}$$
$$g = g_{\mathrm{Na}} + g_K + \overline{g_L} \tag{6.23}$$

The model takes into account three kinds of chemical synapses: AMPA, NMDA and GABA. All three types of synapses are modelled with two state variables A, B, we omit the voltage dependent homeostatic term in [69].

These determine voltage dependent conductances and synaptic currents given by

$$\alpha_s(V) = \frac{1}{1 + (\mathrm{mg}/\mathrm{mgdep})\exp(-\gamma V)} \tag{6.24}$$
$$g_s = \overline{g_s}(B - A)\alpha_s(V) \tag{6.25}$$
$$I_s = g_s(V)(V - e_s). \tag{6.26}$$

The constants appearing in α_s are specific to each of the types of chemical synapses, as are the values of $\overline{g_s}$ and the value of the reversal potential e_s. Numerical values for them are summarized in table 6.3.

The dimensionless state variables A, B undergo exponential decays with two different time constants τ_f, τ_s, which are also specific to each synapse type

$$\dot{A} = -A/\tau_f \tag{6.27}$$
$$\dot{B} = -B/\tau_s \tag{6.28}$$

with initial conditions

$$A(0) = 0 \tag{6.29}$$
$$B(0) = 0. \tag{6.30}$$

Whenever a presynaptic spike arrives the state variables jump

$$A = A + wf(\tau_f, \tau_s) \tag{6.31}$$
$$B = B + wf(\tau_f, \tau_s) \tag{6.32}$$

here w is the synaptic weight and the factor $f(\tau_f, \tau_s)$ is given by

$$t_p = (\tau_f \tau_s)/(\tau_s - \tau_f \log(\tau_s/\tau_f)) \tag{6.33}$$
$$f(\tau_f, \tau_s) = \frac{1}{\exp(-t_p/\tau_s) - \exp(-t_p/\tau_f)}, \tag{6.34}$$

it normalises the maximal amplitude of $B - A$ to w.

6.3 Adjoint Equations

As already pointed out in the introduction of this chapter the main observation is that for multi-compartment models the structure of the adjoint equations is determined by the morphology of the neuron just as the forward dynamics is. More precisely to each compartmental membrane voltage V_k the adjoint method

x	$\overline{g_x}$[nS]	τ_f [ms]	τ_s [ms]	mg [mM]	e [mV]	γ [1/mV]	mgdep [mM]
AMPA	0.7	0.3	1.8	0	0	0	0
NMDA	1.3	8	35	1	0	0.077	3.57
$GABA_1$	0.5	0.5	5	0	−75	0	0
$GABA_2$	0.5	2	23	0	−75	0	0

Table 6.3: Numerical constants that determine the synapse dynamics. AMPA, NMDA are excitatory synapse types, GABA an inhibitory synapse types. Data adapted from [69]. The NMDA channel is voltage gated.

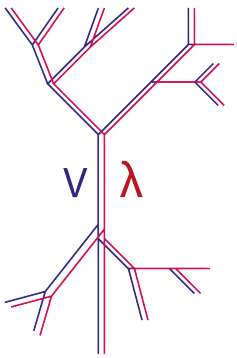


Figure 6.4: Illustration of the adjoint dynamics in a structured neuron, voltage adjoint state variables λ_V are coupled by the same conductance matrix and section state variables couple only locally. We therefore can think of the adjoint equations as being implemented in an overlayed identical structure.

introduces an adjoint variable λ_{V_k}, whose coupling structure is determined by the same adjacency matrix.

Another observation is that depending on where the structure of the loss function error signals can be injected at different places in the neuron morphology and the adjoint equation determines how that error signal is propagated along the neuron morphology. In biological neurons a working hypothesis for how error assignment happens are backpropagating action potentials based on calcium currents. The idea is that errors are communicated to proximal dendrites and backpropagated to the distal dendrites via backpropagating action potentials and potentially other signalling mechanisms based on the endo-plasmatic reticulum (see e.g. [69] and references therein). The adjoint state dynamics predicts precise dynamics based on errors injected at these proximal dendrites. This suggests an association between the adjoint state variables and experimentally observed mechanisms.

Recall that adjoint equation was given by

$$(\lambda')^T = \lambda^T \partial_x f - \partial_x l. \tag{6.35}$$

Here $\partial_x f$ denotes the Jacobian matrix $\mathbf{J}$ given by

$$\mathbf{J} = \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial x_1} & \cdots & \frac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}. \tag{6.36}$$

Therefore the adjoint equations of multi-compartment neuron model will have the general form

$$\begin{aligned} C_k \lambda'_{V_k} &= \lambda_{V_k} \left[G_{ik} - \partial_{V_k} (I^{\text{ion}}(V_k) + I^{\text{syn}}(V_k)) \right] && (6.37)\\ &+ \sum_l \lambda_{x_{lk}} \partial_{V_k} \left[\alpha_{lk}(V_k, S)(1 - x_{lk}) - \beta_{lk}(V, S) x_{lk} \right] && (6.38)\\ &+ \sum_l \lambda_{A_{lk}} \partial_{V_k} f_{lk}(V, A, p) && (6.39)\\ &+ \partial_{V_k} L && (6.40)\\ &= \lambda_{V_k} \left[G_{ik} - \sum \overline{g}_c p(x) - \sum g^{\text{syn}}(V^{\text{syn}}, A) \right] && (6.41)\\ &+ \lambda_{V_k} \partial_{V_k} g^{\text{syn}}(V^{\text{syn}}, A) \left[V^{\text{syn}} - e^{\text{syn}} \right] && (6.42)\\ &+ \sum_l \lambda_{x_{lk}} \partial_{V_k} \left[\alpha_{lk}(V_k, S)(1 - x_{lk}) - \beta_{lk}(V, S) x_{lk} \right] && (6.43)\\ &+ \sum_l \lambda_{A_{lk}} \partial_{V_k} f_{lk}(V, A, p) && (6.44)\\ &- \partial_{V_k} l && (6.45) \end{aligned}$$

The partial derivative $\partial_{V_k} I_k^{\text{ion}}(V)$ captures the dependence on the state variables of Ion channels. Since the matrix G_{ik} is symmetric by construction, the adjoint state dynamics is coupled in the same way as the voltage state variables are. This indicates that error information within a neuron might be propagated according to a version of the adjoint state dynamics.

$$\begin{aligned} \tau_{lk} &= \alpha_{lk}(V, S) + \beta_{lk}(V, S) && (6.46)\\ \lambda'_{x_{lk}} &= -\lambda_{x_{lk}} / \tau_{lk} - \lambda_{V_k} / C_k \partial_{x_{lk}} \sum_c \overline{g}_c^{\text{ion}} p_c(x)(V_k - e_c^{\text{ion}}) && (6.47)\\ \lambda'_{A_{lk}} &= \lambda_{A_{lk}} \partial_{A_{lk}} f(V, A, p) - \lambda_{V_k} / C_k \partial_{A_{lk}} g^{\text{syn}}(V^{\text{syn}}, A)(V_k^{\text{syn}} - e_{lk}^{\text{syn}}) && (6.48) \end{aligned}$$

Now let me assume that synaptic transmission is triggered by a jump condition that only depends on variables in the presynaptic compartment (voltage, etc.) in other words

$$j(V_{\text{pre}}, A_{\text{pre}}, p) = 0 \tag{6.49}$$

and the transition equation similarly only affect pre- and post-synaptic variables

$$\begin{aligned} V_{\text{pre}}^+ &= \theta_{V_{\text{pre}}}(V_{\text{pre}}^-, A_{\text{pre}}^-, p) && (6.50)\\ A_{\text{pre}}^+ &= \theta_{A_{\text{pre}}}(V_{\text{pre}}^-, A_{\text{pre}}^-, p) && (6.51)\\ V_{\text{post}}^+ &= \theta_{V_{\text{post}}}(V_{\text{post}}^-, A_{\text{post}}^-, p) && (6.52)\\ A_{\text{post}}^+ &= \theta_{A_{\text{post}}}(V_{\text{post}}^-, A_{\text{post}}^-, p) && (6.53) \end{aligned}$$

where I dropped the synaptic and neuron compartment indices for clarity. Recall the general equation for the jumps in the adjoint state variables (eq. (3.49)):

$$(\lambda^-)^T = (\lambda^+)^T \left[\left[f^+ - \partial_{x^-}\theta f^- \right] \frac{\partial_{x^-} j}{\partial_{x^-} j \dot{x}^-} + \partial_{x^-}\theta \right] + (l^- - l^+) \frac{\partial_{x^-} j}{\partial_{x^-} j \dot{x}^-} \quad (6.54)$$

then

$$\partial_{x^-}\theta = \begin{bmatrix} \frac{\partial \theta_{V_{\text{pre}}}}{\partial V^-_{\text{pre}}} & \frac{\partial \theta_{V_{\text{pre}}}}{\partial A^-_{\text{pre}}} & 0 & 0 \\ \frac{\partial \theta_{A_{\text{pre}}}}{\partial V^-_{\text{pre}}} & \frac{\partial \theta_{A_{\text{pre}}}}{\partial A^-_{\text{pre}}} & 0 & 0 \\ 0 & 0 & \frac{\partial \theta_{V_{\text{post}}}}{\partial V^-_{\text{post}}} & \frac{\partial \theta_{V_{\text{post}}}}{\partial A^-_{\text{post}}} \\ 0 & 0 & \frac{\partial \theta_{A_{\text{post}}}}{\partial V^-_{\text{post}}} & \frac{\partial \theta_{A_{\text{post}}}}{\partial A^-_{\text{post}}} \end{bmatrix}. \quad (6.55)$$

and

$$\partial_{x^-} j = \left[\frac{\partial j}{\partial V^-_{\text{pre}}}, \frac{\partial j}{\partial A^-_{\text{pre}}}, 0, 0 \right]. \quad (6.56)$$

Therefore we arrive at the equations for the jumps in the adjoint state variables

$$\begin{aligned} \lambda_{V^-_{\text{pre}}} = & \Big(\lambda_{V^+_{\text{pre}}} [\dot{V}^+_{\text{pre}} - (\partial_{V^-_{\text{pre}}} \Theta_{V_{\text{pre}}} \dot{V}^-_{\text{pre}} + \partial_{A^-_{\text{pre}}} \Theta_{V_{\text{pre}}} \dot{A}^-_{\text{pre}})] \\ & + \lambda^T_{A^+_{\text{pre}}} [\dot{A}^+_{\text{pre}} - (\partial_{V^-_{\text{pre}}} \Theta_{A_{\text{pre}}} \dot{V}^-_{\text{pre}} + \partial_{A^-_{\text{pre}}} \Theta_{A_{\text{pre}}} \dot{A}^-_{\text{pre}})] \\ & + \lambda_{V^+_{\text{post}}} [\dot{V}^+_{\text{post}} - (\partial_{V^-_{\text{post}}} \Theta_{V_{\text{post}}} \dot{V}^-_{\text{post}} + \partial_{A^-_{\text{post}}} \Theta_{V_{\text{post}}} \dot{A}^-_{\text{post}})] \\ & + \lambda^T_{A^+_{\text{post}}} [\dot{A}^+_{\text{post}} - (\partial_{V^-_{\text{post}}} \Theta_{A_{\text{post}}} \dot{V}^-_{\text{post}} + \partial_{A^-_{\text{post}}} \Theta_{A_{\text{post}}} \dot{A}^-_{\text{post}})] \Big) \frac{\partial_{V^-_{\text{pre}}} j}{\partial_{V^-_{\text{pre}}} j \dot{V}^-_{\text{pre}} + \partial_{A^-_{\text{pre}}} j \dot{A}^-_{\text{pre}}} \\ & + \lambda^T_{V^+_{\text{pre}}} \partial_{V^-_{\text{pre}}} \Theta_{V_{\text{pre}}} + \lambda^T_{A^+_{\text{pre}}} \partial_{V^-_{\text{pre}}} \Theta_{A_{\text{pre}}} \\ & + (l^- - l^+) \frac{\partial_{V^-_{\text{pre}}} j}{\partial_{V^-_{\text{pre}}} j \dot{V}^-_{\text{pre}} + \partial_{A^-_{\text{pre}}} j \dot{A}^-_{\text{pre}}} \end{aligned} \quad (6.57)$$

$$\begin{aligned} \lambda^T_{A^-_{\text{pre}}} = & \Big(\lambda_{V^+_{\text{pre}}} [\dot{V}^+_{\text{pre}} - (\partial_{V^-_{\text{pre}}} \Theta_{V_{\text{pre}}} \dot{V}^-_{\text{pre}} + \partial_{A^-_{\text{pre}}} \Theta_{V_{\text{pre}}} \dot{A}^-_{\text{pre}})] \\ & + \lambda^T_{A^+_{\text{pre}}} [\dot{A}^+_{\text{pre}} - (\partial_{V^-_{\text{pre}}} \Theta_{A_{\text{pre}}} \dot{V}^-_{\text{pre}} + \partial_{A^-_{\text{pre}}} \Theta_{A_{\text{pre}}} \dot{A}^-_{\text{pre}})] \\ & + \lambda_{V^+_{\text{post}}} [\dot{V}^+_{\text{post}} - (\partial_{V^-_{\text{post}}} \Theta_{V_{\text{post}}} \dot{V}^-_{\text{post}} + \partial_{A^-_{\text{post}}} \Theta_{V_{\text{post}}} \dot{A}^-_{\text{post}})] \\ & + \lambda^T_{A^+_{\text{post}}} [\dot{A}^+_{\text{post}} - (\partial_{V^-_{\text{post}}} \Theta_{A_{\text{post}}} \dot{V}^-_{\text{post}} + \partial_{A^-_{\text{post}}} \Theta_{A_{\text{post}}} \dot{A}^-_{\text{post}})] \Big) \frac{\partial_{A^-_{\text{pre}}} j}{\partial_{V^-_{\text{pre}}} j \dot{V}^-_{\text{pre}} + \partial_{A^-_{\text{pre}}} j \dot{A}^-_{\text{pre}}} \\ & + \lambda^T_{V^+_{\text{pre}}} \partial_{A^-_{\text{pre}}} \Theta_{V_{\text{pre}}} + \lambda^T_{A^+_{\text{pre}}} \partial_{A^-_{\text{pre}}} \Theta_{A_{\text{pre}}} \\ & + (l^- - l^+) \frac{\partial_{A^-_{\text{pre}}} j}{\partial_{V^-_{\text{pre}}} j \dot{V}^-_{\text{pre}} + \partial_{A^-_{\text{pre}}} j \dot{A}^-_{\text{pre}}} \end{aligned} \quad (6.58)$$

$$\lambda_{V^-_{\text{post}}} = \lambda_{V^+_{\text{post}}} \partial_{V^-_{\text{post}}} \Theta_{V_{\text{post}}} + \lambda^T_{A^+_{\text{post}}} \partial_{V^-_{\text{post}}} \Theta_{A_{\text{post}}} \quad (6.59)$$

$$\lambda^T_{A^-_{\text{post}}} = \lambda_{V^+_{\text{post}}} \partial_{A^-_{\text{post}}} \Theta_{V_{\text{post}}} + \lambda^T_{A_{\text{post}}} \partial_{A^-_{\text{post}}} \Theta_{A_{\text{post}}} \quad (6.60)$$

We conclude that the post-synaptic adjoint state variables influence the presynaptic adjoint state variables of a synapse on synaptic transmission. Note that

the equations simplify further if for example the transition equations do not depend on the pre-synaptic voltage. The form of the equations is general enough to capture a wide variety of common synaptic plasticity and transmission models. Together with the corresponding equations for the parameter gradients, which can be derived starting from eq. (3.50) in a similar way this allows for applications such as metaplasticity and parameter estimation of synaptic transmission mechanism based on observations of membrane dynamics.

6.3.1 Traub Model

We can specialise the calculation we just did to the Traub model. Then we find for the voltage adjoint state variable

$$\lambda'_{V_k} = \lambda_{V_k}(G_{kj} - (\overline{g_{\mathrm{Na}}}h_k \cdot m_k^2 + \overline{g_K}n_k + \overline{g_L}) - \partial_{V_k} I_k^{\mathrm{syn}}(V)) \tag{6.61}$$

$$+ \sum_{x\in\{n,m,h\}} \lambda_{x_k}[\alpha'_x(V_k)(1-x_k) - \beta'_x(V_k)x_k] \tag{6.62}$$

$$- \partial_{V_k} l \tag{6.63}$$

since the Ion channel equations only depend on the voltage of the neuron segment they are in. The adjoint variables to the Ion channel state variables follow the equations

$$\lambda'_{m_k} = -\lambda_m/\tau_m + \lambda_{V_k}(2\overline{g_{\mathrm{Na}}}hm(V - e_{\mathrm{Na}}) + \overline{g_K}(V - e_K)) \tag{6.64}$$

$$\lambda'_{h_k} = -\lambda_h/\tau_h + \overline{g_{\mathrm{Na}}}\lambda_{V_k}m^2(V - e_{\mathrm{Na}}) \tag{6.65}$$

$$\lambda'_{n_k} = -\lambda_{n_k}/\tau_n + \overline{g_K}\lambda_{V_k}(V - e_K). \tag{6.66}$$

Similarly the adjoint state variables of the synapses follow the equations

$$\lambda'_{A_k} = -\lambda_{A_k}/\tau_f - \overline{g_s}\lambda_{V_k}\alpha_s(V)(V_k - e_s) \tag{6.67}$$

$$\lambda'_{B_k} = -\lambda_{B_k}/\tau_s + \overline{g_s}\lambda_{V_k}\alpha_s(V)(V_k - e_s). \tag{6.68}$$

We can therefore specialise eqs. (6.57), (6.58), (6.59) (6.60) to

$$\lambda_{V^-_{\mathrm{pre}}} = \left(\lambda_{V^+_{\mathrm{post}}}[\dot V^+_{\mathrm{post}} - \dot V^-_{\mathrm{post}}] + \lambda^T_{A^+_{\mathrm{post}}}[\dot A^+_{\mathrm{post}} - \dot A^-_{\mathrm{post}}]\right)\frac{\partial_{V^-_{\mathrm{pre}}} j}{\partial_{V^-_{\mathrm{pre}}} j\dot V^-_{\mathrm{pre}} + \partial_{A^-_{\mathrm{pre}}} j\dot A^-_{\mathrm{pre}}}$$

$$+ \lambda_{V^+_{\mathrm{pre}}} + (l^- - l^+)\frac{\partial_{V^-_{\mathrm{pre}}} j}{\partial_{V^-_{\mathrm{pre}}} j\dot V^-_{\mathrm{pre}} + \partial_{A^-_{\mathrm{pre}}} j\dot A^-_{\mathrm{pre}}} \tag{6.69}$$

$$\lambda^T_{A^-_{\mathrm{pre}}} = \left(\lambda_{V^+_{\mathrm{post}}}[\dot V^+_{\mathrm{post}} - \dot V^-_{\mathrm{post}}] + \lambda^T_{A^+_{\mathrm{post}}}[\dot A^+_{\mathrm{post}} - \dot A^-_{\mathrm{post}}]\right)\frac{\partial_{A^-_{\mathrm{pre}}} j}{\partial_{V^-_{\mathrm{pre}}} j\dot V^-_{\mathrm{pre}} + \partial_{A^-_{\mathrm{pre}}} j\dot A^-_{\mathrm{pre}}}$$

$$+ \lambda^T_{A^+_{\mathrm{pre}}} + (l^- - l^+)\frac{\partial_{A^-_{\mathrm{pre}}} j}{\partial_{V^-_{\mathrm{pre}}} j\dot V^-_{\mathrm{pre}} + \partial_{A^-_{\mathrm{pre}}} j\dot A^-_{\mathrm{pre}}} \tag{6.70}$$

$$\lambda_{V^-_{\mathrm{post}}} = \lambda_{V^+_{\mathrm{post}}} \tag{6.71}$$

$$\lambda^T_{A^-_{\mathrm{post}}} = \lambda^T_{A^+_{\mathrm{post}}} \tag{6.72}$$

Here $\lambda_{A^\pm_{\mathrm{post}}} = (\lambda_{A^\pm}, \lambda_{B^\pm})$ and we could use (6.31), (6.32) and (6.24) to further simplify the expression.

6.3.2 Model on BrainScaleS-2

The multi-compartment neuron model implemented on the BrainScaleS-2 neuromorphic chip features current based synapses and three operating modes of each of the neuron compartment meant to model NMDA, Ca^{2+} and Na spikes. Compared to the neuron model I described in the previous sections, most of the non-trivial dynamics is implemented by *switching* of the dynamics, whenever one of the neuron compartments reaches a certain threshhold. Depending on the compartment type this results in either a reset to a reset potential or a transition to a plateau potential. This mode is maintained for a digitally set period of time. A conceptual motivation and implementation details can be found in [140]. Here I want to focus on a derivation of the adjoint sensitivity equations of the implemented circuitry. Since it also involves switching, it is well suited to the method presented in the previous chapter. When one of the neuron compartments crosses a configurable threshold voltage V_T a digital switch triggers a transition from the configurable leak conductance g_L to a reset (maximal) conductance g_R or a "NMDA" conductance g_N as well as a reset E_R or "NMDA" potential E_N. The multi-compartment neuron equations are given by

$$C_k \dot{V}_k = G_{lk} V_l + g_x (E_x - V)_k + I_k \tag{6.73}$$
$$\tau_r \dot{R}_k = -\delta_{kl} \tag{6.74}$$
$$\tau_s \dot{I}_k = -I_k \tag{6.75}$$

the jump conditions are

$$j_k(V) = V - V_T \tag{6.76}$$

and

$$q_k(R) = -R_k \tag{6.77}$$

with transition equations

$$V_l = V_l \tag{6.78}$$
$$I_l = I_l + W_{kl} \tag{6.79}$$
$$R_l = R_l + \delta_{kl} \tag{6.80}$$

In reality spike transport takes time, so the transition in the current variables is temporally shifted compared to when a neuron crosses a threshold. It is relatively easy to take delays into account, but I will not do so here. The adjoint equations are

$$C_k (\lambda_V')_k = (\lambda_V)_l G_{lk} - g_x (\lambda_V)_k - \partial_{V_k} l \tag{6.81}$$
$$\lambda_R' = 0 \tag{6.82}$$
$$(\lambda_I')_k = -(\lambda_I)_k / \tau_s + \lambda_V / C_k \tag{6.83}$$

The current flow before and after neuron compartment n crossed the threshhold are

$$(\dot{V}^+)_k = 1/C_k [G_{lk} V_l + g_- (E_- - V)_k + I_k^-] \tag{6.84}$$
$$(\dot{V}^-)_k = 1/C_k [G_{lk} V_l + g_+ (E_+ - V)_k + I_k^+] \tag{6.85}$$

where I've denoted by $g_+ \in \{g_\mathrm{N}, g_\mathrm{R}\}$ the conductance and by $E_+ \in \{V_R, V_N\}$ the potential after the switch. A treshold crossing of neuron compartment n results in the following transition equations for the adjoint variables

$$(\lambda_V^-)_l = (\lambda_V^+)_k \left[\left[\frac{1}{C_k}(g_+ E_+ - g_- E_-)_k \delta_{kn} + W_{kn} \right] \frac{\delta_{nl}}{\dot{V}_n} + \delta_{kl} \right] + (\lambda_R^+)_k \left[\frac{1}{\tau_r} \delta_{kn} \frac{\delta_{nl}}{\dot{V}_n} \right] + (\lambda_I^+)_k \left[-W_{kn} \frac{\delta_{nl}}{\dot{V}_n} \right] + (l^- - l^+) \frac{\delta_{nl}}{\dot{V}_n} \tag{6.86}$$

$$(\lambda_I^-)_l = (\lambda_I^+)_l \tag{6.87}$$

$$(\lambda_R^-)_l = (\lambda_R^+)_l \tag{6.88}$$

(summation over n not implied).

At the end of a refractory period of neuron compartment n ends the conductance g_- changes to $g_+ = g_L$ and the potential E_- changes to $E_+ = E_L$

$$(f_V^+)_k = 1/C_k [G_{lk} V_l + g_+ (E_+ - V)_k + I_k] \tag{6.89}$$

$$(f_V^-)_k = 1/C_k [G_{lk} V_l + g_- (E_- - V)_k + I_k] \tag{6.90}$$

and this leads to corresponding changes

$$(\lambda_V^-)_l = (\lambda_V^+)_l \tag{6.91}$$

$$(\lambda_I^-)_l = (\lambda_I^+)_l \tag{6.92}$$

$$(\lambda_R^-)_l = (\lambda_R^+)_k \delta_{kl} + (\lambda_V^+)_k \left[\frac{1}{C_k}(g_+ E_+ - g_- E_-)_k \delta_{kn} \right] \frac{\delta_{nl}}{\dot{R}_n} \tag{6.93}$$

(summation over n not implied). Starting from eq. eq. (3.50) we could also derive parameter gradients.

6.4 Conclusions

In this chapter I've derived explicit formulas for the adjoint dynamics and therefore gradients of structured neurons with respect to arbitrary loss functions.

The next immediate step would be to use these equations in some task. Since the ordinary differential equations that appear here are harder to solve this would need the use of an ODE solver library such as [132, 131, 133]. The striking result is that the adjoint dynamics uses the same dendritic structure (as encoded by the conductance matrix) to propagate the error information and that coupling of the adjoint equation between structured neurons happens only locally at each synapse. This presents a different perspective on what is usually understood as the "weight transport" problem (see e.g. [41, 73] and [6] for a recent attack on the problem), namely that the Backpropagation algorithm needs weight symmetry between the connections in the forward and backward pass, which is not biologically plausible. The symmetry of the weight matrices for the forward and backward pass is naturally encoded by the dendritic tree. Among the remaining mysteries are rather the reversal of the time direction between adjoint and forward dynamics and that there is only weak evidence for a bi-directional coupling at synapses.

Nevertheless various known mechanisms of backpropagating action-potentials, retrograde transmission [119] and retrograde transport in the endo-plasmatic

reticulum could be analysed as approximations to the exact adjoint dynamics. One thing to investigate would be the chemical reaction networks that determine synaptic plasticity and transmission. Ultimately biological neurons employ of course many mechanisms that aren't cleanly captured by simple ordinary differential equations. Among them is the continuous expression and production of proteins by the cell nucleus and their transport to and from synaptic densities. For example axonal transport [75] involves active transport of precursors for synaptic vesicles, mitochondria and other structures along micotubules. A fascinating question then becomes how to incorporate gene-expression networks and substructures and dynamics of biological neurons, such as the endoplasmatic reticulum and microtubule transport into functional models.

An alternative view on structured neurons abandons the discretisation into neuron segments and instead uses a 1-dimensional cable equation to model the voltage dynamics. Even more generally one could consider a two-dimensional version of the cable equation, the shape of the neuron would then be captured by an appropriate rieammanian metric and the observation that the conductance matrix is symmetric corresponds to the fact that the laplace beltrami-operator is self-adjoint.

In the context of the mutli-compartment neuron circuitry as implemented on BrainScaleS-2 the results derived in this chapter are of immediate use for training multi-compartment neuron circuits in an hardware-in-the-loop setting.

Chapter 7

Modifications to Plasticity Processing Unit

Roughly speaking the complex biological processes at each synapse should implement synaptic plasticity and thereby learning in biological neurons. Abstracted from the details of complicated bio-chemistry so called plasticity rules, such as STDP (spike time dependent plasticity) and others are used to model the plasticity of synapses in neuron models. In neuromorphic hardware the possibility to implement such rules ranges fixed function implementations of specific rules, to rules based on a limited set of micro-operations [44], to fully programmable plasticity as for example in the BrainScaleS-2 and Spinnaker system.

In the BrainScales-2 system it is the role of the plasticity processing unit (PPU) to implement any such rule in an efficient fashion as possible [57]. As part of my work I oversaw the successful scaling of the existing plasticity processing unit to the full sized single chip system HICANN-X and changed the memory architecture to allow for the execution of instructions from external memory, as well as vector and scalar read and write access to external memory. Both changes taken together enable the implementation of plasticity algorithms that take advantage of large external memory, thereby significantly increasing the modelling capabilities of the system. Moreover the change is also crucial for the analog inference accelerator mode as it allows software to be written without being constrained by the on-chip memory. Finally it enables use of the same software and hardware abstraction layers, because memory limitations are not an issue. I also implemented a fully parallel random number generator for the vector unit of the plasticity processor to enable applications such as random synaptic rewiring, noise terms in plasiticity rules and stochastic rounding.

The rest of the chapter is structured as follows: I will begin with a brief description of the implemented additional hardware features. Then I discuss results that have already obtained and enabled by these features. Finally I will discuss some future experiments that are made possible by these features in particular with reference to the Eventprop algorithm for both the structured and unstructured neuron configuration possible on HICANN-X.

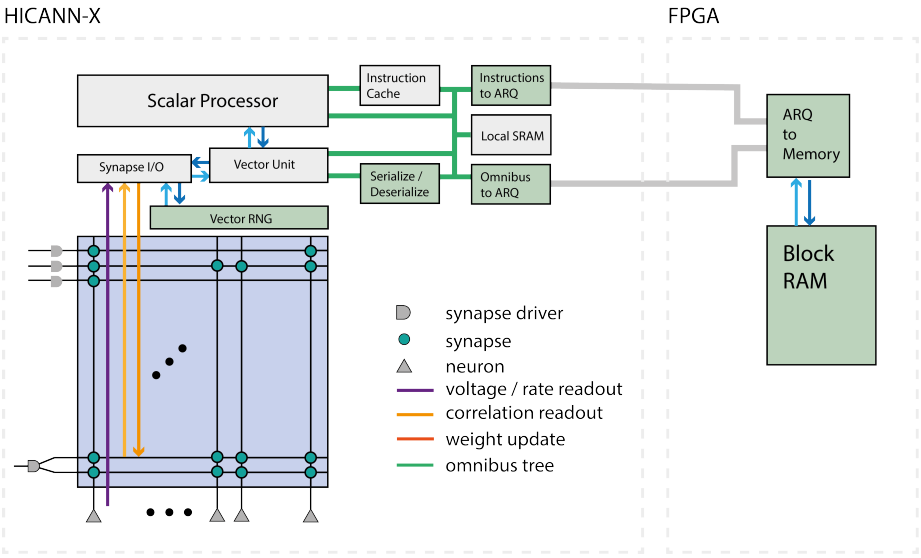


Figure 7.1: Schematic diagram of parts of the HICANN-X neuromorphic processor. Additions to the plasticity processor and FPGA carried out in this thesis are highlighed in light green. The synapse I/O unit and vector unit was scaled to accomodate the larger number of 256 synapse columns. A random number generator was added which generates 128 8 bit random numbers at a time. Parts of the omnibus tree (in green) was modified in such a way that it is 128 bit wide, such that loading and storing data from the vector unit is sped up by a factor of 4. In addition both the scalar, as well as the vector unit are now ARQ clients via the omnibus tree, enabling communication to the external FPGA. Separately the instruction cache can now load data from either the local SRAM or external FPGA connected memory (block or DRAM). On the FPGA a simple ARQ client was implemented that arbitrates instruction and data requests and connects to Block RAM memory.

7.1 Implementation

In this setion I will describe in more detail the different additional components I implemented for the HICANN-X single chip system. A schematic summary is given in figure 7.1. I will describe in turn the parallel random number generator, extensions to the memory datapath and improvements to the verification process. The plasticity processing unit itself is described in [57].

Briefly the processor implements a subset of the 32 bit POWER instruction set architecture, together with a custom vector extension that operators on 8 and 16-bit vectors of implementation defined length. The processor is structured into a frontend, which handles instruction fetching, decoding and branch-control and a backend with several functional units. Instructions are dispatched to the functional units in order but can retire out of order, with data dependencies being tracked by scoreboards. The vector-unit is treated as an additional functional unit by the frontend, but instructions are executed by the vector-unit asynchronously, with an implementation defined number of possible outstanding instructions. Within the vector-unit, instructions are again dispatched to several functional units (load-store, arithmetic and parallel-load-store), with each being organized into slices. Execution happens again in order, but instructions can retire out of order which is particularly important for parallel-load-store instructions that can take more than 40 cycles to complete. With out-of-order retirement, arithmetic instructions (such as plasticity calculations) can execute while the parallel-load-store unit waits for a result of for example a CADC (column ADC) measurement. Dependencies are again tracked by a scoreboard mechanism. Synchronisation between the scalar unit and the vector-unit happens explicitely by a dedicated ”sync” instruction. The implementation used for the HICANN-X single chip system uses 8 vector slices, with 16 wide 8-bit or 8-wide 16-bit vectors each. This results in $1 : 2$ ratio of vector entries to synapse columns. Up to 32 instructions to the vector unit can be enqueued. The parallel-load-store unit of the vector unit handles load and store instructions that operate on full-width vectors at a time. In the implementation used in the HICANN-X chip this unit interfaces with the synapse-access module (called synapse i/o in figure 7.1), which in turn handles different kinds of transactions based on the memory region that is accessed. It is also connected to the Omnibus [57] tree, which we we will describe in more detail below.

The possible transactions handled by the synapse access unit are

1. Synapse Array raw load / store
2. Synapse Array address load / store
3. Synapse Array weight load / store
4. Synapse Array config load / store
5. CADC causal/acausal read
6. Correlation causal/acausal reset
7. Random Number Generator (RNG) load / store

the first four of these transaction types concern access to the digital configuration, weight and address bits stored in each synapse. Access is done column-wise

in an even-odd pattern with respect to the slices. The CADC read operation triggers an analog to digital conversion of a full row of causal- or anti-causal state variables in the synapse array or of the membrane voltage of all 256 neurons, the result is buffered and can be read back consecutively into two vector registers. Finally the Random Number Generator load transaction triggers an update of the accessed linear-feedback shift registers (LFSR) state and returns a vector of 8-bit random numbers. A Random Number Generator write transaction sets the lower 8 bits of the 16-bit state of the accessed random number generator columns.

7.1.1 Random Number Generator Array

The random number generator module is implemented as 256 independent 16-bit linear-feedback shift registers (LFSR), this implementation was chosen because it is simple to implement and (relatively) resource efficient. A n-bit LFSR is specified by a polynomial of degree $n-1$ in $\mathbf{F}_2[x]$. For a LFSR to have maximal cycle length 2^n the polynomial needs to be *primitive*. Two such examples for $n = 16$ are

$$p_1 = x^{15} + x^4 + x^2 + x^1 + 1 \tag{7.1}$$

$$p_2 = x^{15} + x^4 + x^3 + x^2 + 1. \tag{7.2}$$

In total there are 2048 primitive polynomials of degree 15. In order to implement the 256 linear-feedback shift registers of each random number generator arrays on the HICANN-X chip, I chose 256 different of these *primitive* polynomials (from [96]). While it is well known that LSFR don't provide high quality random numbers, the samples associated with different primitives polynomials should be independent. One limitation of the design is that only the lower 8-bit of each of the LSFR states are returned during a read transaction and only the lower 8-bit can be initialised as well. This was done because the primary envisioned use case of the random number output is its use in plasticity algorithms, which we assume to mostly employ 8-bit arithmetic and operations on synapse columns. We turn to a description of such potential applications in the discussion section 7.3.

7.1.2 Memory Interface

I now turn to the modifications made to the memory architecture of the plasticity processing unit. I begin with a description of the architecture before the modifications were made. The plasticity processor has access to a local memory via three datapaths:

The instruction cache loads instructions from a local memory, the scalar read-write unit loads and writes data to local memory and finally the vector unit has a load-store unit that can read and write local memory. All of these use 32-bit wide addresses and byte-wise addressing.

The main goal of the modifications to the memory path I made were to enable access via the ARQ data transport layer [92] to the FPGA. To that end I implemented two ARQ clients, one for instruction and one for data access. The interface consists of a data and command queue and a response queue. The protocol guarantees in-order reliable delivery so that minimal logic is needed

for interfacing it with omnibus transactions. The instruction cache is connected via an omnibus split on the highest bit to the internal SRAM and the new ARQ client. That way any instruction fetch traffic can't block data read-write operations or vice versa.

Both the parallel-load store unit and the load-store unit of the vector unit can access the data ARQ client. This client operates on 128 bit wide words, which is the width of one slice of the vector unit. A serialisation and deserialisation module implements the conversion from 1024 bit used by the parallel-load store unit to 8×128 bit words and back. By interfacing the parallel-load store unit directly with external memory, the vector unit can be reused as an asynchronously operating data-mover, while the scalar unit is executing other operations. Finally load-store unit of the Scalar unit is also connected via the Omnibus tree to the data ARQ client.

On the FPGA the ARQ transport is again terminated in several queues which correspond to the ones instatiated on the ASIC side. To verify initial functionality both in simulation and initial FPGA bringup I implemented a simple interface of the two sets of ARQ queues to a synthesisable Block Ram memory module.

7.1.3 Verification

Verification of the functionality was done in several steps: A Universal Verification Methodology (UVM) [85] based verification workflow was used to verify correct functionality of the processor scale-up and modifications to the Omnibus hierachy. For this purpose I implemented PPU specific UVM sequences and a large set of instruction level unit tests. This setup also allowed for arbitrary C programs to be executed. After the first tapeout I contributed to a different verification flow based on a DPI-C interface. Based on this approach described in [74], end to end integration tests could be run, including use of the full C++ software stack [115]. Of particular importance was the ability to also simulate the backannotated physical design, as this revealed a critical bug in the vector unit register file.

7.2 Results

In this section I will discuss technical results directly related to the memory interface implementation and then some experiments that are enabled by it.

7.2.1 Performance Measurements

Using the software integration performance measurements of the memory interface could be made before tape-out. On current silicon an integration test implemented by Philip Spilger yields results summarized in table (7.1). Similarly the performance of the external instruction access could be quantified by running a standard benchmark that tests performance of embedded processors Coremark. This was done pre-tapeout by Eric Müller to quantify the impact of the size of the instruction fetch queue. With instructions and data in local SRAM the HXv2 core achieved in simulation a Coremark score of 335.91 and with data in local SRAM and instruction fetching from external memory a

Operation	Cycles	± Min/Max
Memory Load	344	35
Memory Store	38	4
Memory Store ×2	175	18
Memory Store ×4	306	31
Memory Store ×8	684	69
Memory Store ×16	1408	141

Table 7.1: Summary of external memory load store performance for 1024 $[bit]$ load and $1, 2, 4, 8, 16$ consecutive stores to and from the vector unit. Measurements are in performance counter cycles, second column gives min/max deviation.

Coremark score of 321.22 (with an instruction queue fetch depth of 16). Further increasing the instruction queue fetch depth to $32, 64$ would yield Coreamark scores of $324.32, 326.05$.

7.2.2 SuperSpike Experiment

An experiment that already has made use of the memory extension presented here is [40]. Generally speaking any use of the plasticity processing unit that makes use of non-trivial programs and has large memory storage requirements is enabled by the extension presented in this chapter.

7.3 Outlook and Discussion

A number of further experiments making use of the memory extension and random number generators can be envisioned. The approach presented in [23] of parametrising plasticity rules by artificial neural networks can be extended to use instead of training by evolutionary strategies, training by the adjoint method. This represents a general approach to metaplasticity in particular if recurrent neural networks are allowed. Such a more complicated plasticity artificial neural network can only feasably be implemented given the larger memory available.

In a similar fashion as [40] one could also attempt to implement the EventProp training algorithm in the loop. The main advantage would be that only the voltage trace around a spike needs to be stored. Since the method is also compatible with the multi-compartment neuron model implemented on BrainScaleS-2 as explained in chapter 6.

Finally the online learning algorithm presented in chapter 5 is a promising candidate for implementation using the correlation sensors. There synaptic rewiring as in [21] could use random number generator array to learn sparse feature maps online.

Chapter 8

Stochastic Inference with a Neuromorphic System

With the end of Moore's law and recent successes in deep learning [101, 141] novel domain specific computer architectures have come to the forefront of attention [47, 79]. Just like deep learning model architectures are sometimes partially inspired by what neuroscientist have learned about the brain [77], neuromorphic computing [112, 61] draws inspiration from physical aspects of the human nervous system. Going back to the beginning of digital computing, parallels were drawn between logical circuits and wiring of neurons in the nervous system [158, 111]. With the advent of neurophysiological measurements of the electrical properties of neurons [81], attempts where made to solve the resulting phenomenological differential equations on contemporary analog computers. Neuromorphic systems continue this tradition in that they typically involve an analog core [129, 130, 19, 139] or at least a low power digital implementation of neuron and synapse dynamics [45, 113, 62]. These cores are then digitally connected by routing spike event packets thereby avoiding the bad scalability of purely analog implementations with current technology. Applications of such systems range from (low-power) inference [51, 50, 142] to the emulation of biological brain-circuits [7]. The BrainScaleS 2 systems stand out in that they are the first to combine fully-programmable hybrid plasticity with emulation of spiking neural networks with time constants 10^3 faster than biological time. Other systems, such as SpiNNaker, allow even more flexible plasticity rules but can realistically only simulate spiking networks in real-time or, like Loihi, can carry out only a limited number of algebraic manipulations on a per synapse level.

Recent interest in *stochastic computing* has been motivated, among other reasons, by the prospects of using novel devices for computation (see e.g. [151, 127] and references therein). Such nano-devices have unreliable or noisy behaviour, which needs to be mitigated and ideally harnessed. Using unreliable logic circuits for computation was already of interest in the early time of modern computing [158]. Already then parallels to the operation of the nervous system were drawn. Among recent approaches to stochastic computing are dedicated machines for approximate bayesian inference [52]. It has also been established recently [56] that MULLER-C elements can be used for Bayesian in-

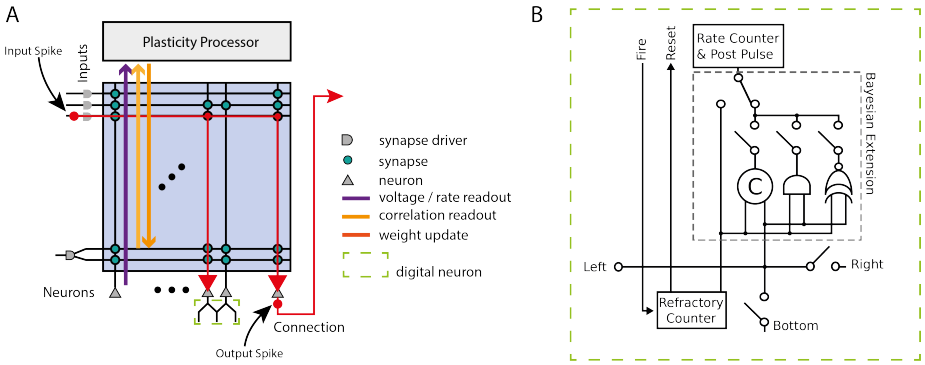


Figure 8.1: Overview of the BrainScaleS-2 neuromorphic core (A). It features a synaptic crossbar, each synapse stores an event address it listens to (indicated by colors here) and a synaptic strength. Input events are broadcast row-wise and integrated in columns that have matching synapse addresses. Each synapse column connects to a dedicated neuron circuit, which has an analog and digital part. The digital part of each of the neuron circuits were modified to allow for the stochastic extension we discuss in this chapter (B). When the configured neuron threshold is crossed in the analog neuron circuit a digital fire signal is emitted. After a neuron has fired it is refractory for a certain period, this boolean refractory signal is implemented by a refractory counter. The output of this refractory counter is used to generate the signal termed *post-pulse*, which gets converted into an event *and* is used as input to the correlation circuit at each synapse. The Bayesian extension gates the generation of the post-pulse based on a binary logic gate (MULLER-C, Xor, And) the left or right neuron's refractory state. In that way neighboring neurons both can be connected together to perform logical operations on their refractory state and influence plasticity algorithms.

ference. While several special purpose stochastic computing systems have been build, conventional digital systems have prevailed so far.

In this chapter we describe an extension to the BrainScaleS-2 system (8.1), which enables it to carry out some of the stochastic computations in those specialised systems in the context of a neuromorphic chip (8.1 B). It works on the same interpretation of the refractory state of leaky-integrate and fire neurons as a binary random variable (illustrated in 8.3 B) that has been successfully used for stochastic inference [114, 116, 30]. In particular spike-based bayesian inference using the high conductance state has been already demonstrated in the chip architecture the modifications discussed in this chapter were made for (see fig. 3 and section 3b in [22]). Beyond that the extension also enables simple temporal computations like coincidence detection. Moreover this extension can be used in conjunction with the correlation circuitry, which enables novel plasticity rules.

The rest of the chapter is organized as follows: We begin with a brief review of spike-based sampling. Leaky integrate and fire neurons are subjected to weighted noise sources, which turns their state variables into stochastic processes. In turn their refractory state can then be interpreted as a binary random variable. Beyond the typical contrastive divergence learning, we propose here that the such a spiking neural network can be trained directly through either surrogate gradient or adjoint learning. In particular it is possible to define surrogate gradients that make it possible to differentiate through the logical operations introduced here.

We then present the newly implemented additions to the digital neuron backend carried out by Gerd Kiene in section 8.3 in the context of the BrainScaleS-2 single chip system. These modifications enable the new additional operating modes complementary to the existing features. The remainder of this chapter outlines four such use cases.

One of the key distinguishing features of the BrainScaleS 2 system is its fully programmable implementation of hybrid plasticity. The newly implemented digital functionality allows for interesting interactions with the correlation and rate counter functionality in the synapse crossbar and neuron array respectively. We present a way of using digital coincidence detection to implemented a gated form of STDP in section 8.4.1 and propose a circuit for implementing online error backpropagation in section 8.4.2.

8.1 Background

In this section I will recall some background on neural sampling as pioneered by [114, 116, 30] and recently implemented in neuromorphic hardware [99, 22]. We want to think of the membrane potential of a neuron as implementing a stochastic process $v_\theta(t)$ parametrised by the weights θ of its synapses. Since the circuit dynamics of both the synapses and neurons we want to consider, with the exception of thermal noise, is deterministic in itself, this is only possible if the circuit's input are samples from some random process. To an observer the membrane voltage process itself typically not observable itself, instead it can observe for each neuron a binary random variable derived from this process. This binary random variable is one, when the neuron is active (refractory) and zero, when it is quiet.

In the case of conductance based leaky-integrate and fire neurons it was

demonstrated in [114] that certain parametrisations of such neurons allow an interpretation of their refractory state as samples from a Boltzmann distribution

$$p(z, W, b) = \frac{1}{Z} \exp(-z^T W z - b^T z), \tag{8.1}$$

there also an explicit translation to and from the parameters θ of the corresponding LIF neuron equations was given. This allows optimization of the Boltzmann distributions parameters W and b independent of the substrate and subsequent transfer to a neuromorphic hardware parameters for fast and efficient sampling [99, 22]. However it also has a number of drawbacks: The correspondence holds only approximately and it is well known that the training method typically used to train Boltzmann machines called Contrastive Divergence (CD) scales poorly for deep restricted Boltzmann machines.

8.2 Neural Sampling by Surrogate Gradients

I therefore want to advance here a different perspective, which contains Boltzmann machines as a special case, but has not been discussed in this form in the context of neuromorphic computing to the best of my knowledge.

The concrete model I want to consider in the following is the leaky-integrate and fire model with current based synapses and an absolute refractory period. There are three n-dimensional state variables $v, i, r \in \mathbf{R}^n$, the membrane voltage v, synaptic input current i and refractory state r. The continuous part of the time evolution is given by

$$\dot{v} = 1/\tau_{\text{mem}}(1 - \Theta(r)) \cdot (v_{\text{leak}} - v + i) \tag{8.2}$$

$$\dot{i} = -1/\tau_{\text{syn}} i \tag{8.3}$$

$$\dot{r} = -1/\tau_{\text{refrac}} \Theta(r), \tag{8.4}$$

with $\tau_{\text{mem}}, \tau_{\text{syn}}, \tau_{\text{refrac}}$ the membrane, synaptic and refractory time constants respectively. The Heaviside function Θ ensures that neuron membrane voltages remain clamped to the reset potential during the refractory period. The k-th neuron spikes if the jump condition

$$j(v^-) = v_k^- - (v_{\text{th}})_k \tag{8.5}$$

is satisfied. The transition equations resulting from such a jump condition can be summarised by

$$v^+ = (1 - z) \cdot v^- + z \cdot v_{\text{reset}} \tag{8.6}$$

$$i^+ = i^- + w_{\text{in}} z_{\text{in}} + w_{\text{rec}} z_{\text{rec}} \tag{8.7}$$

$$r^+ = (1 - z) \cdot r^- + z \cdot r_{\text{reset}}^- \tag{8.8}$$

where z_{rec} and z_{in} are the recurrent and input spikes respectively. Given these equations the binary random variable associated to each LIF neuron can therefore be defined by

$$z_k = \Theta(r_k), \quad k = 1, \ldots, n \tag{8.9}$$

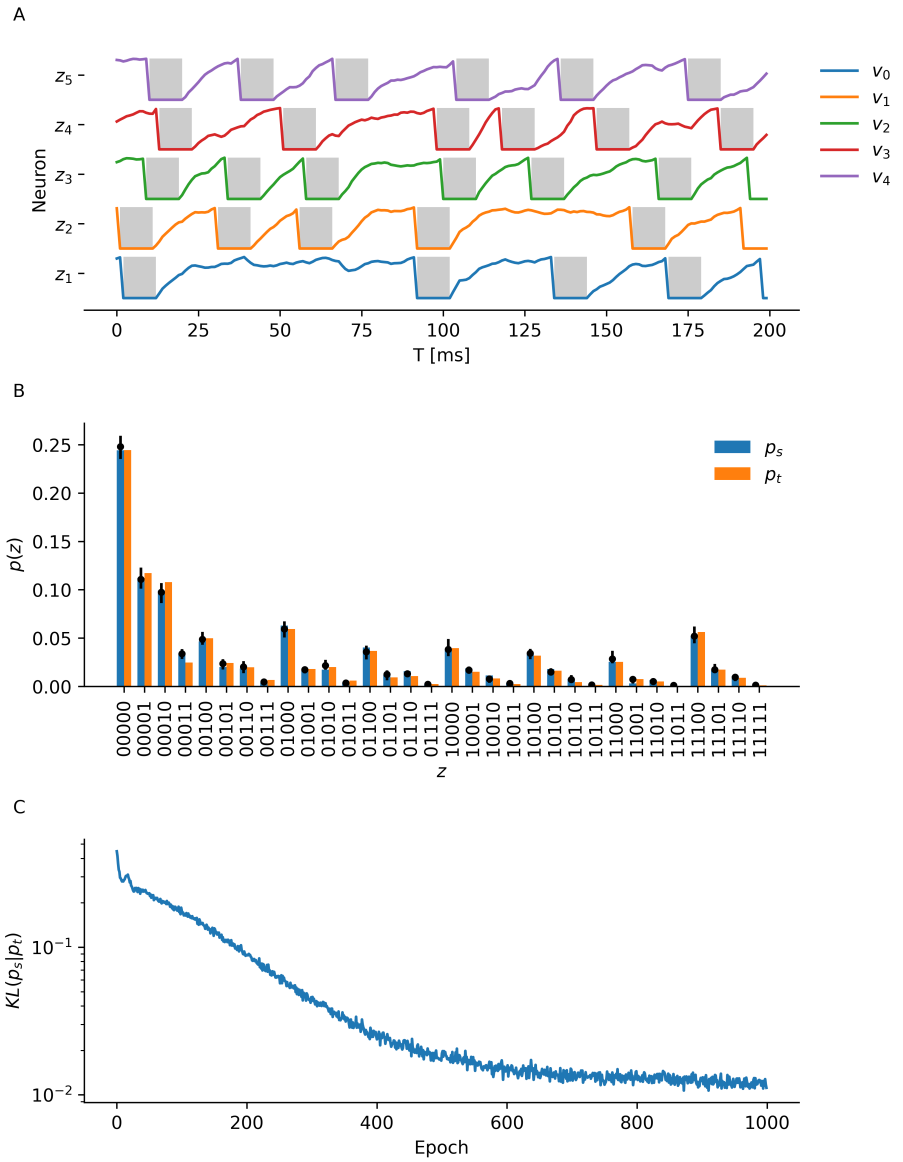


Figure 8.2: Five leaky-integrate and fire neurons with current based synapses and refractory periods of 10 ms are recurrently connected with initial weights $w_{\text{rec}} \sim N(1/\sqrt{5})$ sampled from a normal distribution. They receive poissonian input (300 Hz) from $K = 256$ independent noise sources weighted by a input matrix with initial weights $w_{\text{in}} \sim N(1/\sqrt{256})$ sampled from a normal distribution. As explained in the main text, the refractory state of each of the neurons is interpreted as a binary random variable $z_i, i = 1 \dots 5$. Training uses mini-batch stochastic gradient descent in batches of $B = 16$ using the ADAM($lr = 10^{-3}, \beta = (0.9, 0.999), \varepsilon = 10^{-8}$) optimizer [94], with a sampling time of $T = 10^4$ ms and the batch-averaged Kullback-Leibler divergence $\text{KL}(p_s|p_t)$ as loss. I indicate the state $z_i = 1$ by grey boxes in (A), together with the voltage traces during a time window of 200 ms. (B) compares sampling distribution $p_s(z)$ and target distribution $p_t(z)$, black bars indicate minimum and maximum over one batch of 16 sample inputs. Training converges over 1000 epochs to a Kulback-Leibler divergence $\text{KL}(p_s|p_t)$ of close to 10^{-2} (C), this is consistent with the performance reported in [114] for a sampling period of 10^4 ms (cf. fig. 3 C).

$z_0(t)$	$z_1(t)$	$c(t)$
0	0	0
1	0	$c(t-1)$
0	1	$c(t-1)$
1	1	1

Table 8.1: MULLER-C element state table.

This definition also immediately suggests equations to compute the AND and XOR gates

$$\mathrm{AND}(z_k, z_l) = z_k z_l = \Theta(r_k)\Theta(r_l) \tag{8.10}$$
$$\mathrm{XOR}(z_k, z_l) = |z_k - z_l| = |\Theta(r_k) - \Theta(r_l)|. \tag{8.11}$$

The MULLER-C gate involves an additional state variable c with two states $0, 1$, there are two jump conditions for c given by

$$\begin{aligned} j_1(c^-, r_k, r_l) &= ((1 - \Theta(r_k)) + (1 - \Theta(r_l)))c^- \\ j_2(c^-, r_k, r_l) &= (\Theta(r_k) + \Theta(r_l))c^- \end{aligned} \tag{8.12}$$

and transition equations

$$c_1^+ = c_1^- + 1 \tag{8.13}$$
$$c_2^+ = c_2^- - 1 \tag{8.14}$$

A simulation of the LIF neuron equations and the computation of the logic gates on refractory states is shown in Figure (8.3 B). This corresponds to the usual state table of the MULLER-C element in discrete time (see 8.1).

For a given integration period T (for example $T = 10^4\,\mathrm{ms}$) we can then define the joint probability distribution defined by sampling the binary random variables $z_1, \ldots, z_n$ as

$$p_s(z; w) = \frac{1}{T}\int_0^T \bigotimes_{i=1}^{n} [1 - \Theta(r_i), \Theta(r_i)]\mathrm{dt}, \tag{8.15}$$

the tensor product compactly expresses the 2^n different entries. Given a target probability distribution $p_t(z)$, we can then pose the optimization problem

$$\min_w \mathrm{KL}(p_s|p_t), \tag{8.16}$$

in other words we can ask for sampling distribution p_s to be close to the target distribution p_t.

The Kullback-Leibler divergence in this case is defined as

$$\mathrm{KL}(p_s|p_t) = -\sum_z p_s(z; w)\log p_t(z) + \sum_z p_s(z; w)\log p_s(z; w), \tag{8.17}$$

that is it is the difference between the cross entropy of p_s and p_t and the entropy of p_s

$$\mathrm{KL}(p_s|p_t) = H(p_s, p_t) - H(p_s). \tag{8.18}$$

Now to solve this optimisation problem I want to employ gradient descent with respect to the paremters w on the Kullback-Leibler divergence. The fundamental obstacle to this is again that the Heaviside function does only have a distributional derivative. Instead of the strategy used in previous chapters, I want to introduce here an alternative approach, based on a relaxation of the *gradient* computation. The basic idea is to interpret the Heaviside function as the cumulative distribution of the point mass probability density given by the Dirac delta distribution. Then the concept of Friedrichs Mollifier [59] provides a rigorous way of talking about smoothed solutions and convergence to weak solutions of equations involving Heaviside step functions and delta distributions. From a physics perspective the idea can be explained in terms of a temperature limit $T \to 0$ as follows: Consider a two-state system of distinguishable "atoms" with energy levels E_0, E_1. The partition function of such a system is

$$Z = \exp(-\beta E_0) + \exp(-\beta E_1) = \exp(-\beta E_1)[1 + \exp(-\beta E)], \beta = \frac{1}{k_B T} \quad (8.19)$$

here k_B is the Boltzmann constant and T the temperature. The probability to encounter the system at equilibrium in either of the states is

$$p_0 = \frac{\exp(-\beta E)}{1 + \exp(-\beta E)} \quad (8.20)$$

$$p_1 = \frac{1}{1 + \exp(-\beta E)}. \quad (8.21)$$

In the case under consideration the membrane and threshold potential define two energy states $E_0 = qv$ and $E_1 = qv_{\text{th}}$, with q the charge of the "ion" mediating the switching (actual biological situations are more complicated and will be briefly addressed in another chapter). Therefore

$$E = q(v - v_{\text{th}}) \quad (8.22)$$

and

$$p_1(v, \beta) = \frac{1}{1 + \exp(-\beta q(v - v_{\text{th}}))} \quad (8.23)$$

In the limit $\beta \to \infty$ or $T \to 0$, the deterministic situation is recovered

$$p_1(v, \infty) = \Theta(v - v_{\text{th}}), p_0(v) = (1 - \Theta(v - v_{\text{th}})) \quad (8.24)$$

In the context of spiking neural networks surrogate gradient training as reviewed in [117] replaces the derivative of Heaviside function, which only exists in a distributional sense, by a scaled positive symmetric mollifier ϕ', for example

$$\phi'(v, \beta) = \alpha p_1'(v, \beta) \quad (8.25)$$

with α some normalisation factor. Choice of $\beta \sim 1/T$ then allows one to adjust how much gradient information is propagated backward in time, when the threshold isn't reached.

In fig. 8.2 I show training results on the first task used in [114] to demonstrate neural sampling using the surrogate gradient function first described in [171]. As can be seen, a similar performance can be achieved without explicitely enforcing weight symmetry or conditions on the input matrix. In the experiment I set

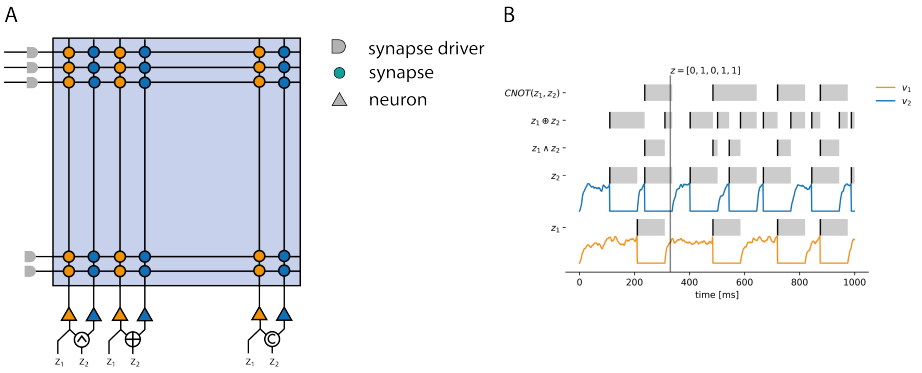


Figure 8.3: Schematic diagram of the analog core with specific choices of digital connections between neighboring neuron circuits (A). From left to right neighboring neurons are connected by AND, XOR and MULLER-C gates. Figure (B) shows a computer *simulated* response to excitatory poisson input. Each neuron has a refractory period set to 100 ms, after it has fired during which the neuron is clamped to the reset potential. The logic gates operate on the refractory signal, I indicate where their output is one as grey boxes and show in color the analog voltage traces of the corresponding neurons. The output spikes are indicated by black lines. At a given point in time the refractory output defines a binary vector, I indicate one such value at the grey vertical line.

the leak potential of the neurons identical to the threshold $v_{\text{leak}} = v_{\text{th}} = 1.0$. Combined with the expressions for the MULLER-C (8.12), AND (8.10) and XOR gate (8.11) by Heaviside functions, surrogate gradient training can also be used to train networks that use the additional digital gates I've introduced in this section.

8.3 Implementation

On the BrainScaleS 2 system the neuron digital backend is used to distinguish and control the different operating modes of the neuron, in particular in order to allow it both to function as a dendritic compartment and as a standalone neuron. Gerd Kiene extended the digital backend with the functionality needed to implement the Bayesian inference primitives explained in the previous section. The conceptual placement of the circuitry within the digital neuron was done jointly.

The refractory time of each neuron is controlled by an asynchronous 7 bit counter, during the refractory period one of a few different alternatives happen, depending on which mode the neuron is in. A 1-bit event register is set to one at the beginning of the refractory time and cleared by the priority encoder circuit once it has been serviced. As indicated in Fig. 8.1 (B) the post-pulse and event counter start signal generation was modified. The post-pulse and event-counter start signals can now be generated in one of the following ways (selectable by digital switches): Either by the neuron fire signal, which is generated by an analog comparator with the neuron's threshhold voltage or as a boolean function f of the neuron's refractory and the refractory pulse of neighboring neurons.

The boolean function f can be either AND, XOR or MULLER-C gate between the two refractory pulses. The refractory pulse used in this computation can be routed to the neuron from left or right and can bypass neurons for routing the pulse. In this new mode the event output and counter are only activated, when the function f evaluates to 1. Similarly the synapses corresponding to the neuron only receive a post-pulse affecting the correlation circuitry [58], if the function evaluates to 1.

8.4 Learning Rules

The BrainScaleS 2 system includes a "plasticity processing unit" (PPU) [58]. Relevant to biological learning rules are the synapse weights, which have 63 levels of strength and can be either inhibitory or excitatory, the rate counters of each neuron and finally local correlation sensors in each synapse, which are meant to enable the implementation of STDP like rules [58].

The correlation circuit in each synapse measures the exponentially weighted time a_+ and a_- between the most recent pre-post or post-pre pair of events. On the chip this is implemented by the "post pulse" signal line from each neuron digital part to its synapse column.

The inclusion of a boolean circuit between the digital part of two adjacent neuron circuits allows in principle a dissociation of the post pulse of the individual neurons with the spike event of the neurons themselves. Instead post pulse generation of one digital compartment can be made dependent on the spike generation of the other digital compartment. This means that the content of the rate counters of one neuron can now depend on the spike activity of a neighboring neuron, similarly the pre-post correlation measurements are now influenced by the spike activity of a neighboring neuron. On a high level the and-gate computes a coincidence of spikes within the refractory period of a neighboring neuron. Since the correlation post-pulse is gated as well, the correlation sensors also only detect correlation when spikes coincide within the refractory window.

The purpose of the following two sections then is to discuss how this could be used in learning rules implementable on the PPU.

8.4.1 Digital Dendrites

Another application of the additional circuitry described above is in the realization of a limited form of multi-compartment neuron, here neighbouring neuron compartments can be thought of as being digitally connected into a simple multi-compartment neuron. Thinking of individual neuron-compartments as performing an overall logical operation was for example suggested in [108]. Because of the purely digital realization, we will refer to this kind of configuration as *digital dendrites*.

Introducing such a logical relation between neighbouring neuron circuits in the case of the neuromorphic hardware under discussion has the additional advantage of obtaining more state variables for a single neuron. As a concrete application we will discuss the case that two neighbouring neuron circuits are connected by an "and"-gate. Together with the on-chip plasticity processing unit [58] this allows for a gated variant of spike time dependent plasticity STDP [65] to be implemented, which enables association of spatio-temporal patterns

to a label. For a detailed description of the on-chip plasticity processing unit and the specific correlation anti-correlation sensors present in each synapse, we refer to [58].

The idea for this use case is very simple: Consider two neuron compartments A and B. Neuron compartment A is will be considered to be "label" compartment and neuron compartment B will be considered the "classification" compartment. The output and post-pulse generation of neuron compartment B is gated via an "and"-gate by the output of neuron compartment A. This means that only while the label neuron compartment A is refractory a spike of the classification neuron compartment B can result in an increase in the correlation trace of its synapses. Application of the standard STDP update rule

$$dw = \alpha(a_+ - a_-),$$

where a_+ and a_- are the correlation traces of a given synapse and α some factor, will therefore result in an increase of those synaptic weights that receive input during the desired labelled input pattern and a decrease for those whose input is not correlated with it. Because the synaptic weights are quantized stochastic rounding has to be used, so that the weight updates are unbiased:

$$d\bar{w} = \lfloor dw \rfloor + \begin{cases} 1 & \text{with probability } p = dw - \lfloor dw \rfloor \\ 0 & \text{with probability } 1 - p \end{cases}$$

8.4.2 Error Backpropagation

The considerations of the last section can be extended to the case of a multilayer network. The new circuitry allows for an implementation of an approximate form of error backpropagation as follows: Consider again neighbouring neuron compartments in pairs, one for the forward direction and one for the backward direction. For simplicity we will consider a simple feed forward network with N layers. Denote by W_i the weights of layer $i = 1 \dots N$. The response of a LIF neuron to an input current I is given by [46, 70]:

$$f(I) = \begin{cases} \left[\tau \log\left(\frac{E+RI-V_r}{E+RI-V_{th}}\right) + t_{\text{ref}}\right]^{-1} & \text{, if } E + RI > V_{\text{th}} \\ 0 & \text{, if } E + RI \leq V_{\text{th}} \end{cases}$$

here V_{th} is the threshhold potential, V_{rest} is the resting potential, τ is the membrane time constant, t_{ref} is the refractory time, R is the membrane resistance, E the membrane potential and I the input current. Close to the onset of firing this response can be approximated by a so called rectified linear unit (see e.g. [70] for this rationale): That is the response of a LIF-neuron to stimulation with an average rate vector x is roughly

$$f(x) = \max(x \cdot w - \beta(\theta), 0),$$

where β is a monotone increasing function of the threshhold θ, and w is the weight vector of the neuron. The derivative of this approximate response function is a heavyside step function

$$f'(x) = \Theta(x \cdot w - \beta(\theta))$$

in other words the derivative is 1 if the input reached the threshhold or zero otherwise. The backpropagated error is then

$$dx = f'(x)dy = \Theta(x \cdot w - \beta(\theta))dy. \tag{8.26}$$

Consider now a specific layer i of the feed forward network. The backpropagated error needs to be determined for both the weights $dW^{(i)}$, as well as the error backpropagated to the inputs $dx^{(i)}$ and optionally to the threshhold $d\theta^{(i)}$.

To compute the error $dx^{(i)}$ backpropagated to the inputs $x^{(i)}$ we use the following prescription: As we said above we associate to each neuron compartment A in the forward direction in layer i, a neurite B in the backward direction. The output of neurite B is gated by an "and"-gate with the output of neuron compartment A. This gating mechanism approximates the multiplication with the derivative of the response of neurite A in the forward direction, because the derivative of the forward activation function is approximately a step function as explained above.

For this to fully conform to the backpropagation algorithm, the weights of neurite B need to be the (transpose) of the synaptic weights with which neurite A connects to layer i+1.

To compute the error $dW^{(i)}$ the average spike rate $r_B^{(i)}$ for each of the backward neurites B in layer i is measured over a time T, this will again be proportional to the derivative of the response function of neurite A, by the rationale given above. The weight update after a time T is then given by

$$dW^{(i)} = -\alpha r_B^{(i)} \cdot \text{sign}(W^{(i)}) \cdot (a_+^{(i)} - a_-^{(i)}),$$

here as above $a_+^{(i)}$ and $a_-^{(i)}$ are the correlation and anti-correlation measurements of all synapses of layer i after the given time period T, which serve as an approximation to the overall input received at this synapse, α is a scaling factor and $\text{sign}(W^{(i)})$ computes whether the synapse is inhibitory or excitatory. The relationship between negative STDP and gradient descent in spiking neural networks was suggested in [18] and in lectures by Hinton.

The output layer of the feed-forward neural network needs to be treated differently: Again consider pairwise neurites A and B, B is again the "label" neurite and A is the "output" neurite. By connecting the output neurite with a "xor"-Gate to the label the output of the label neurite, it will only produce an output if the label neurites activity differs from that of the output neurite. Indeed for correlated input bitstreams X, Y the xor gate computes

$$Z = |X - Y| \tag{8.27}$$

This is precisely the desired behaviour for producing an error signal, as it is the derivative of the l_2 distance.

Because the synapse array of the BrainScaleS 2 system only has quantized weights, stochastic rounding can again be used to implement this weight update rule. That is the update rule will be

$$d\bar{W}^{(i)} = \lfloor dW^{(i)} \rfloor + \begin{cases} 1 & \text{with probability } p = dW^{(i)} - \lfloor dW^{(i)} \rfloor \\ 0 & \text{with probability } 1 - p \end{cases}$$

To facilitate this kind of stochastic operation the plasticity processing unit has a vectorized pseudo-random number generator.

8.5 Conclusions

In this chapter I've introduced an extension to the BrainScaleS-2 architecture jointly developed with Gerd Kiene, which allows one to combine primitives used for stochastic computing directly with the neuron and correlation circuitry. Separately I also introduced a novel way of training deterministic spiking neurons to model binary probability distributions. Directly training on the output of the spiking neural network via either surrogate gradient training (as done in this chapter) or the adjoint method. This is a conceptual advantage since it does not presuppose an underlying probability distribution but instead allows one to directly approximate stochastic processes with deterministic neurons and simple noise sources. And interesting theoretical consideration would be to find results on which stochastic processes can be approximated in this way. The case where the stochastic process has a stable distribution is then a special case. The circuits presented in this chapter have not been experimentally used so far, but I hope the suggestions made in this chapter, together with the training method I presented can lead to interesting results.

Chapter 9

Learning as Optimal Control

In previous chapters I have discussed several approaches to optimisation in spiking neural networks, all centered around the hypothesis that the problem can be posed as the minimisation of some loss over time. With the exception of the online learning approximations I presented in chapter 5 the optimisation, while also being governed by a dynamical system (the adjoint equations), can't readily be interpreted as "biologically" or physically plausible since the equations are solved in reverse time and the separation of learning in separate "forward" and "backward" phases is not observed in this fashion. Moreover it is unclear why some form of gradient descent should happen in the first place and how to decide when parameters should be updated. In this chapter I want to partially address these issues by formulating the "parameter update" as an optimal control problem. That is instead of considering the parameters to be static, they are now considered to be part of the dynamics and instead there are temporally varying controls that influence them. I find that under the assumption of linear controls (but non-linear) dynamic and a quadratic contribution of the control to the cost, the optimal weight dynamics can be derived explicitely.

9.1 Background

This section follows closely the exposition presented in [91]. We want to understand the learning dynamics in terms of a stochastic optimal control problem of the following form:

$$dx = b(x(t), u(t), t)dt + d\xi. \tag{9.1}$$

Here x, b, $d\xi$ are n-dimensional vectors and u is an m-dimensional control vector. The noise term $d\xi$ is a Wiener process with correlation $\langle d\xi_k d\xi_l \rangle = \nu_{kl}(u, x, t)dt$. We are given an initial state $x(t_0) = x_0$ and are looking for a solution until some final time t_f.

The Hamilton-Jacobi Bellman equations are a way to find optimal controls $u(t)$ such that the cost

$$C(x, t) = \left\langle \phi(x(t_f)) + \int_{t_0}^{t_f} l(x, u, t)\mathrm{dt} \right\rangle_{x(t)} \tag{9.2}$$

is minimized, here the expectation value is formed over all realisations $x(t)$ of the stochastic process. Define the optimal cost to go function

$$J(x,t) = \min_{u(t\to t_f)} \left\langle \phi(x(t_f)) + \int_t^{t_f} l(x,u,t)\mathrm{dt} \right\rangle_{x(t)} \tag{9.3}$$

Then the Hamilton-Jacobi Bellman equation for J is given by [91]

$$-\partial_t J(x,t) = \min_u \left(l(x,u,t) + b(x,u,t)^T \partial_x J(x,t) + \frac{1}{2}\mathrm{Tr}\left[\nu(x,u,t)\partial_x^2 J(x,t)\right] \right), \forall t, x \tag{9.4}$$

with boundary condition

$$J(x,t_f) = \phi(x). \tag{9.5}$$

Now assume that the control is linear, that is

$$dx = b(x(t),t)dt + Bu(t)dt + d\xi. \tag{9.6}$$

and the cost $l(x,u,t)$ quadratic in u:

$$l(x,u,t) = \frac{1}{2}(u^T R u) + V(x,t) \tag{9.7}$$

then the stochastic Hamilton-Jacobi Bellmann equations simplify to [91]

$$-\partial_t J = \min_u \left(u^T R u + V + (b+Bu)^T \partial_x J + \frac{1}{2}\mathrm{Tr}\left[\nu \partial_x^2 J\right] \right) \tag{9.8}$$

The minimization over u can be carried out [91]

$$u = -R^{-1} B^T \partial_x J(x,t). \tag{9.9}$$

9.2 Learning

I now want to distinguish two kinds of variables in our stochastic model: Parameters $w \in \mathbf{R}^l$ and state variables $y \in \mathbf{R}^k$. The idea is to find optimal changes of the parameters based on a linear control u. Therefore assume $B = P_w^T$, where $P_w \colon \mathbf{R}^{k\times l} \to \mathbf{R}^l$ projects to the parameter space. In other words the control can only affect the parameters of the system. Inserting into equation we find that

$$dw = P_w b(x(t),t)dt - R^{-1}\partial_w J(x,t)dt + d\xi \tag{9.10}$$

if we furthermore assume that $P_w b = 0$, that is w has no other deterministic dynamics, this implies that the parameters w should change according to the gradient of the cost to go function with respect to the parameters

$$dw = -\mathrm{grad}_w J(x,t)dt + d\xi, \tag{9.11}$$

where I've identified R^{-1} as the inverse of a (constant) Riemannian metric on the parameter space $\mathbf{R}^l$. This demonstrates that under the assumptions I've made the optimal learning strategy is indeed gradient descent according to the future cost-to-go function. The problem then becomes how to estimate this gradient.

9.3 Conclusion

The perspective of learning as optimal control can be fruitfully expanded beyond the admittedly limited observation I have made here. In particular [91, 90] goes on to linearise equation (9.8) and derive a pathintegral representation. Stochastic weight update equations like (9.11) were also used in [89], the perspective from the Hamilton-Jacobi-Bellmann equation might shed further light on "optimal" synaptic rewiring.

Chapter 10

Emulating quantum computation with ANN

10.1 Introduction

This chapter is based on [124]. Most of the preceding chapters were motivated by the question how physical systems (such as the biological nervous system or neuromorphic hardware) can *learn* to implement certain classical computations (classification, regression, markov decision processes). In this chapter I will consider how classical artificial neural networks (ANN) can learn to emulate quantum operations. This is done in order to prepare for the more challenging question of how a neuromorphic computing system could implement quantum operations, which I will discuss in the next chapter. Both questions are motivated by a change of perspective on quatum systems as being realised in certain classical statistical systems [162] obeying constraints. Quantum states in this case are represented by probabilistic information on macroscopic observables. Quantum gates then correspond to transformations of the probabilistic information at "layer" t to $t + \epsilon$ in a certain way. Here t can correspond to a layer in an artificial neural network, time or space.

Implementation of the necessary constraints on the classical system take different forms. Here I implement aspects of quantum computation by small artificial neural networks. They can perform arbitrary sequences of unitary transformations on the density matrix of two entangled qubits, by composing a small a set of trained gates (Hadamard, CNOT and $\pi/8$-rotation). The necessary constrains are then that the complex structure, hermiticity of the transformation are implemented by the ANN, as well as proper normalisation and positivity of the output matrix. Finally the network also needs to implement the unitary transformation (see fig. 10.1, 10.2). Once a basic set of gates has been trained, non-computing sequences of gates can be implemented by composition (see fig. 10.3).

For a given network structure the learnable parameters represented the remaining degrees of freedom that have to be adjusted to implement the task. In the networks that are typically considered, one curious feature is that they are *overparametrised* relative to the number of training samples that are available. One view on the results in presented in this chapter is as a demonstration, how

such an overparametrisation can help to implement constraints in the transformations imposed by the data. The space of $N \times N$ density matrices, that is positive hermitean matrices ρ with trace one $\mathrm{tr}\,\rho = 1$, can be seen as a submanifold in the space of all $2N \times 2N$ real matrices, once a choice of complex structure and hermitean inner product has been made, as I will explain in more detail below. I will denote the image of a given density matrix ρ under this embedding by $\bar{\rho}$. While the artificial neural network operates on the space of all $2N \times 2N$ matrices, only on the subspace of density matrices (the "quantum subsystem") [162] the transformation by the ANN implements the unitary transformation

$$\rho(t+\epsilon) = U(t)\rho(t)U^{\dagger}(t), \tag{10.1}$$

that is relates the density matrix at layer t to the density matrix at layer $t+\epsilon$.

The real valued entries of $\bar{\rho}$ can be considered to be (a function of) the expectation value of some macroscopic probabilistic observable, which could also include correlations, that is expectation values of products of "basis observables". One choice is to use two-level observables $s = \pm 1$, which could for example correspond to a "active" or "refractory" state of a spiking neuron [31, 126]. This kind of two-state system was also discussed in chapter 8.1 and will be explicitely used in the following chapter 11. Since the publication of the preprint [124] this chapter is based on, low-dimensional quantum states were also realised on neuromorphic hardware using a similar interpretation of the refractory state of a LIF neuron [42].

A quantum system with Q quantum spins or qubits has a density matrix ρ that is defined by $2^{2Q} - 1$ independent real numbers. Positivity implies further "quantum constraints" on these real numbers, which imply the uncertainty relation [162]. Clearly already for moderate numbers of qubits (say $Q = 20$) it becomes impractical to use a representation which uses a one-to-one correspondence to classical expectation values, since this would require $2^{2Q} - 1$ independent classical basis observables.

This changes, if instead the elements of ρ can be represented as a function of correlations of basis observables. A minimal example is to use the expectation values and correlations of $3Q$ classical Ising spins to describe a density matrix for Q qubits [162]. In this approach one uses $3Q$ Ising spins $s_k^{(i)}$, with $i = 1, \ldots, Q$ running over the different qubits and $k = 1 \ldots 3$ are the cartesian directions of the quantum spins. One then considers up to Q-point correlation functions, denoted by

$$\chi_{\mu_1 \mu_2 \ldots \mu_Q} = \langle s_{\mu_1}^{(1)} s_{\mu_2}^{(2)} \ldots s_{\mu_Q}^{(Q)} \rangle, \tag{10.2}$$

with $\mu_l = 0, \ldots, 3$ and $s_0^{(i)} = 1$. The minimal correlation map then defines the density matrix as

$$\rho = 2^{-Q} \chi_{\mu_1 \mu_2 \ldots \mu_Q} L_{\mu_1 \mu_2 \ldots \mu_Q}, \tag{10.3}$$

here $L_{\mu_1 \mu_2 \ldots \mu_Q}$ are the generators of the unitary group $\mathrm{SU}(2^Q)$ given by

$$L_{\mu_1 \mu_2 \ldots \mu_Q} = \tau_{\mu_1} \otimes \tau_{\mu_2} \otimes \ldots \otimes \tau_{\mu_Q}, \tag{10.4}$$

with the Pauli-matrices τ_μ explicitely given by

$$\tau_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \tau_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \tau_2 = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \tau_3 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{10.5}$$

By definition ρ is normalized $\mathrm{tr}(\rho) = 1$ and hermitean $\rho^\dagger = \rho$. This minimal correlation map is complete if every positive quantum density matrix ρ can be represented by classical correlation functions (10.2) via (10.3). Denote this map by

$$q\colon \langle s^{(1)}_{\mu_1} s^{(2)}_{\mu_2} \dots s^{(Q)}_{\mu_Q} \rangle \mapsto \rho. \tag{10.6}$$

and the correlation map by

$$c\colon (s^{(1)}_{\mu_1}, s^{(2)}_{\mu_2}, \dots, s^{(Q)}_{\mu_Q}) \mapsto \langle s^{(1)}_{\mu_1} s^{(2)}_{\mu_2} \dots s^{(Q)}_{\mu_Q} \rangle. \tag{10.7}$$

A given unitary transformation U acts on density matrices by $\mathrm{ad}_U(\rho) = U\rho U^\dagger$. If q were complete then for a given U it might be possible to find a function f_U, such that

$$q \circ c \circ f_U = \mathrm{ad}_U \circ q \circ c \tag{10.8}$$

In other words the unitary transformation might be realised by a change of the classical probability distribution, which realises the change of expectation values (10.2).

10.2 Implementation

In order to implement the transformation (10.1) of a given density matrix $\rho(t)$ by a unitary matrix U, one can start in the case of 2 qubits with a real 8×8 matrix $A(t)$ as input to a network that produces an 8×8 matrix $B(t + \epsilon)$ as output. The matrix elements of $A(t)$ correspond therefore to 64 input "neurons" and the matrix output elements $B(t + \epsilon)$ to 64 output "neurons". The overall network architecture has a "bottleneck" layer of m hidden neurons, all realised as real linear units. The overall network architecture is therefore $64 - m - 64$. Since a density matrix is specified by only 15 real numbers, the network has to learn that only those degrees of freedom are relevant, the remaining $64-15 = 49$ dimension can be considered to be the "probabilistic environment".

I first implement a "quantumness gate" which maps a 8×8 real matrix $A(t)$ with entries in the interval $[-1, 1]$ (thought to be the expectation values of two-level system), to a density matrix $\rho(t)$. This is done in two steps, first define

$$\tilde{A} = -IAI, \quad I = \begin{pmatrix} 0 & -1_4 \\ 1_4 & 0 \end{pmatrix} \tag{10.9}$$

with 1_4 denoting the 4×4 unit matrix. Then define

$$\bar{A} = \frac{1}{2}(A + \tilde{A}). \tag{10.10}$$

The entries of $\bar{A}$ can by construction be written as

$$\bar{A} = \begin{pmatrix} C_R & -C_I \\ C_I & C_R \end{pmatrix} \tag{10.11}$$

with C_R and C_I the real and imaginary part of a complex matrix $C = C_R + iC_I$. This map establishes an $\mathbf{R}$ algebra isomorphism between 8×8 real matrices of this form and 4×4 complex matrices. In a second step one can obtain a density matrix by

$$\rho = \frac{CC^\dagger}{\mathrm{tr}\, CC^\dagger}, \tag{10.12}$$

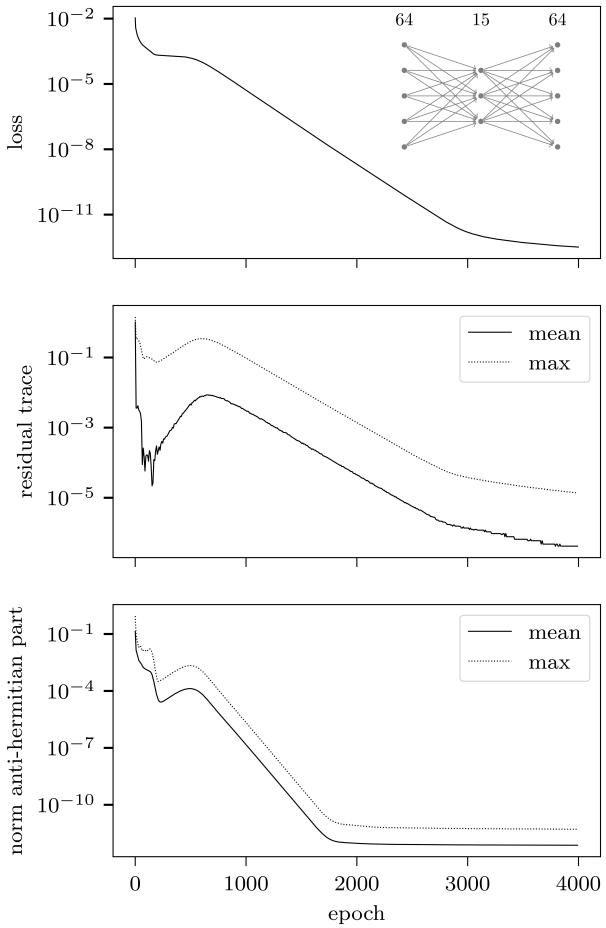


Figure 10.1: From top to bottom: Loss function, residual trace ($|\mathrm{tr}(\rho) - 1|$) and norm of the anti-hermitian part of the output matrix as a function of the training epoch for supervised training on the CNOT gate with bottleneck dimension $m = 15$. Figure adapted from [124].

which is by definition hermitian, has norm one and is positive. The combination of these steps maps a matrix real 8×8 matrix $A(t)$ to a density matrix $\rho(t + \epsilon)$. This "quantumness gate" has been realised so far with a mean-squared error loss of 10^{-5} by training a small ANN with rectify linear non-lineraties with increasing batch sizes of $32, 64, ..., 512$ on 10^6 training samples. Training samples were generated by uniformly sampling entries of $A(t)$ from $[-1, 1]$. The AdaDelta [170] optimizer was used for training. All neural networks were implemented in TensorFlow [3] and used the Keras library [36] for training.

A complex valued 4×4 density matrix for a 2-qubit system can be in turn be regarded as a real 8×8 matrix

$$\bar{\rho}(t) = \begin{pmatrix} \rho_R & -\rho_I \\ \rho_I & \rho_R \end{pmatrix}. \tag{10.13}$$

Such matrices are used as input to the training of an artificial neural network to perform a unitary transformation. Similarly the mean squared error loss is imposed on the real representation of the output density matrix $\bar{\rho}(t+\epsilon)$, that is the loss is given by

$$L = \frac{1}{N}\sum_{i=0}^{N}|\bar{\rho}_i(t+\epsilon) - B_i(t+\epsilon)|^2 \tag{10.14}$$

where $i = 0, \ldots, N$ runs over one mini-batch of training samples, $\bar{\rho}_i(t+\epsilon)$ is the real representation of $\rho_i(t+\epsilon)$ in eq. (10.1) for a given unitary transformation $U(t)$ (for example the CNOT-gate) and $B_i(t+\epsilon)$ is the output of the ANN on $A_i(t) = \bar{\rho}_i(t)$. The norm in (10.14) is the Frobenius norm on matrices. Training was implemented by generating 10^5 input density matrices and performing mini-batch stochastic gradient descent with batch-size of 10^3 using the AdaDelta [170] optimizer. Before training the ANN produces for a given real representation of a density matrix an arbitrary real matrix $B(t+\epsilon)$, during training it subsequently learns to implement the complex structure, hermiticity and norm condition, as well as the desired unitary transformation on $\bar{\rho}(t)$. More precisely the matrix $B(t+\epsilon)$ can be decomposed into 4×4 block matrices

$$B(t+\epsilon) = \begin{pmatrix} B_R(t+\epsilon) & -B_I(t+\epsilon) \\ B'_I(t+\epsilon) & B'_R(t+\epsilon) \end{pmatrix}. \tag{10.15}$$

implementation of the complex structure means that $|B_I - B'_I| \ll 1$ and $|B_R - B'_R| \ll 1$. That is it can approximately represent a complex matrix $F(t+\epsilon)$. Moreover this complex matrix is hermitian, has trace one $\operatorname{tr} F(t+\epsilon) = \operatorname{tr} B(t+\epsilon) = 1$ and $F(t+\epsilon)$ is positive (all eigenvalues of F are positive).

Once training has completed the parameters of the ANN have adapted in such a way that a real representation of an arbitrary density matrix $\bar{\rho}(t)$ is transformed to real representation $\bar{\rho}(t+\epsilon)$ of the unitary transformed density matrix $\rho(t+\epsilon)$ with high precision.

10.3 Results

In Fig. 10.1 I show training results of an ANN with bottleneck dimension $m = 15$ (number of intermediate neurons) on the CNOT gate. The loss (10.14) decreases over 4000 epochs to a value below 10^{-11}. Furthermore the residual trace $|\operatorname{tr} F - 1|$ and the anti-hermitean part $|F(t+\epsilon)F^\dagger(t+\epsilon)|$ are shown. Already after a brief initial training period these two quantities decrease rapidly, implying that the output of the ANN while not faithfully reproducing the unitary transformation, already maps to the submanifold of hermitean, normalized matrices with good precision. This is both true for individual training examples (I indicate the maximal values across the training set for a given epoch), as well as on average over 10^5 independent samples.

I then evaluate the training performance as a function of the bottelneck dimension m (see fig. 10.2). For $m > 16$ training converges already after training for 1000 epochs. Training with $m = 15$ takes considerably longer to converge and completely breaks down for $m < 15$. This corresponds to the fact that a normalized hermitian 4×4 matrix (and therefore a 2-qubit density matrix) has 15 independent real degrees of freedom.

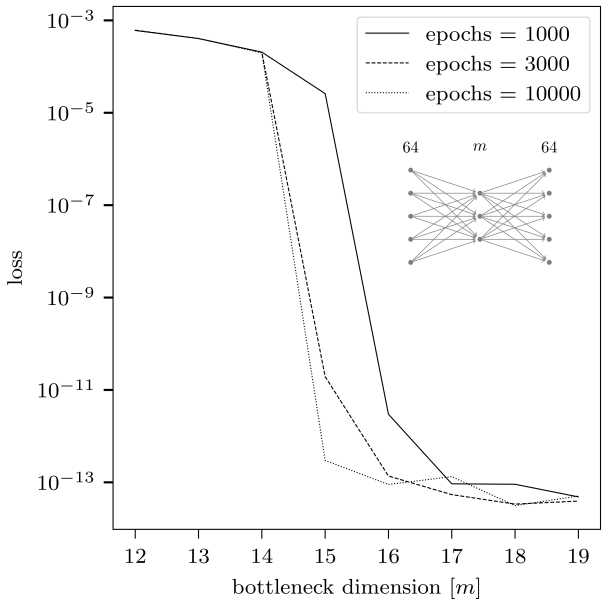


Figure 10.2: Loss values for training on the CNOT gate as a function the number of intermediate neurons m. Shown are the values for 1000, 3000 and 10000 epochs. The training converges for $m \geq 15$, which corresponds to the real dimension of the space of 2-qubit density matrices. Figure adapted from [124].

Varying the bottleneck dimension therefore gives one a way to determine the minimal number of necessary degrees of freedom. This could generalise to situations where this dimension is less obvious than in the present case. If the input to the network is not a real representation of a density matrix $r\bar{h}o(t)$, but an an arbitrary 8×8 matrix $A(t)$, the resulting matrix will not be hermitean or normalised, exept for $m = 15$.

To evaluate the composition of several trained ANNs, I first train one ANN on the CNOT gate U_C and a second one on an application of the Hadamard on the first spin and a rotation by $\pi/8$ on the second spin $U_{HR} = U_H \otimes_{\mathbf{C}} U_R$ (see also inset in fig. 10.3). The gates are given explicitely by

$$U_C = \begin{pmatrix} 1 & 0 \\ 0 & \tau_1 \end{pmatrix}, \quad U_H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad U_R = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/8} \end{pmatrix} \tag{10.16}$$

Given an initial density matrix $\rho(t)$ and its corresponding real representation $r\bar{h}o(t)$ the composition $U(t) = U_{HR}U_C$ is applied n-times to ρ, yielding a final density matrix $\rho(t + n\epsilon)$, that is

$$\rho(t + n\epsilon) = U(t + n\epsilon)\rho(t)U^\dagger(t + n\epsilon), \quad U(t + n\epsilon) = (U_{HR}U_C)^n \tag{10.17}$$

Similarly the two corresponding trained ANN can be applied in sequence n-times yielding a matrix $B(t + n\epsilon)$. As can be seen in fig. 10.3 the resulting matrix $B(t + n\epsilon)$ approximates the density matrix $\bar{\rho}(t + n\epsilon)$ with a mean-squared error of less than 10^{-6} for $n \leq 2^{15}$. Since composition of U_{HR} and U_C densely

approximate arbitrary unitary transformations for varying n this also demonstrates the feasibility of implementing arbitrary unitary transformations with only two trained networks. Similarly non-commutativity of the gates can be demonstrated by changing the order in the sequence of composition of the networks.

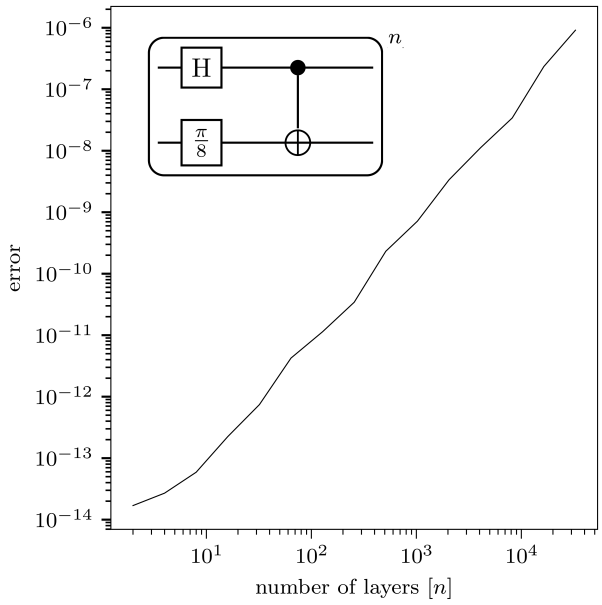


Figure 10.3: Sequential composition ANNs trained on $H \otimes R_{\pi/8}$ and the $CNOT$-gate. Shown is the error, that is the mean-squared Frobenius norm between the real representation $\bar{\rho}(t+n\epsilon)$ of the density matrix and output $B(t+n\epsilon)$ obtained by composing the ANNs, for different numbers of layers n. Figure adapted from [124].

10.4 Summary and Outlook

In this chapter I discussed emulation of quantum operations by artificial neural networks, without any explicit use of the probabilistic interpretation introduced in section 10.1. In the next chapter I will explain how the refractory states of deterministic spiking neurons connected in a recurrent network and subjected to noise can serve as a classical two-state observable, whose time-averaged expectation value and correlation can be used to compute (10.2) and the correlation map (10.3). Just like explained here the coupling parameters of the network adjust to represent the quantum state. In scaling beyond a small number of qubits it will be an interesting challenge to find efficient representations for restricted problems based on a subset of the correlations, analogous to the autoregressive models pioneered in [146], where the state entering the optimisation can only depend on a restricted subset of the correlations.

Chapter 11

Neuromorphic Quantum Computing

11.1 Introduction

This chapter corresponds to the results reported in [123]. Given the fact that spiking neural networks can approximately sample binary probability distributions by associating their quiet and active state to binary random variables as discussed in chapter 8.1 the question arises whether it would be possible to realise the correlation map as introduced in [162] and explained in 10 by a spiking neural network. This question wass raised in [162, 124]. In the context of artificial neural networks recent progress has already demonstrated important aspects of quantum computation with artificial neural networks [32, 150, 105]. See also [29] and references therein for a comprehensive review of computational approaches using artificial neural networks on quantum problems. The time continuous nature and natural event processing capabilites of spiking neural networks make them compelling candidates for exploring both quantum state approximation by sampling and hybrid neuromorphic quantum systems (a recent survey of approaches to "quantum neuromorphic computing" is [109]). This is in particular interesting in the context of the adjoint dynamics discussed in chapter 4. In this chapter however I focus on four steps which continue the approach formulated in [162, 124]

1. Using a parametrised network of spiking neurons and given an arbitrary 2-qubit density matrix ρ, parameters W can be found such that the expectation values and correlations of the binary refractory state variables interpreted as Ising spins, produce under the correlation map the desired density matrix ρ.

2. We demonstrate that the minimal correlation map [162] is complete in the case of 2-qubits by numerically implementing a procedure which allows me to find for a given density matrix ρ a probability distribution p, which maps to ρ through the correlation map.

3. We show how an arbitrary unitary transformation on density matrices ρ results in a corresponding change of probability distributions p.

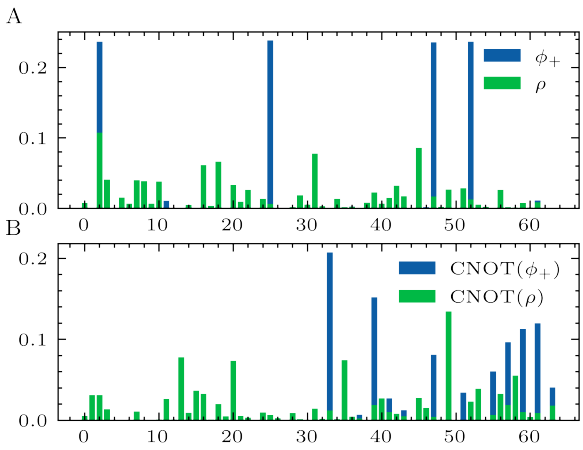


Figure 11.1: Probabilities corresponding to the density matrices of the pure state $\psi_+ = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ (blue) and a randomly generated density matrix ρ (A) (green). We also indicate their transformation under a CNOT gate (B). The labels $0 \ldots 63$ correspond to the states of 6 classical spins $s_k^{(i)}, i = 1, 2, k = 1 \ldots 3$. For example the label 3 corresponds to the spin state $[-1, -1, -1, -1, 1, 1]$. Figure adapted from [123].

4. Finally we show that the minimal correlation map is not complete for $Q > 2$, while numerical experiments suggests that it is capable of producing random density matrices ρ it fails on density matrices associated with maximally entangled pure states. An updated version of [123] will suggest an extended correlation map.

11.2 Correlation map for two qubits

Recall from the previous chapter 10, that the minimal correlation map for Q-qubits is given by

$$\rho = 2^{-Q} \chi_{\mu_1 \mu_2 \ldots \mu_Q} L_{\mu_1 \mu_2 \ldots \mu_Q}, \tag{11.1}$$

here $L_{\mu_1 \mu_2 \ldots \mu_Q}$ are the generators of the unitary group $\mathrm{SU}(2^Q)$ given by

$$L_{\mu_1 \mu_2 \ldots \mu_Q} = \tau_{\mu_1} \otimes \tau_{\mu_2} \otimes \ldots \otimes \tau_{\mu_Q}, \tag{11.2}$$

with the Pauli-matrices τ_μ explicitely given by

$$\tau_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \tau_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \tau_2 = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \tau_3 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{11.3}$$

in the case of two qubits, this simplifies to

$$\rho = \frac{1}{4} \chi_{\mu_1 \mu_2} L_{\mu_1 \mu_2}, \tag{11.4}$$

with

$$L_{\mu_1 \mu_2} = \tau_{\mu_1} \otimes \tau_{\mu_2}. \tag{11.5}$$

That is for two qubits the correlation map is given by a function

$$f\colon \mathbf{R}^{4\times 4} \to \mathbf{C}^{4\times 4}, \chi_{\mu_1\mu_2} \mapsto \frac{1}{4}\chi_{\mu_1\mu_2} L_{\mu_1\mu_2} \tag{11.6}$$

The coefficients $\chi_{\mu_1\mu_2}$ are then

$$\chi_{00} = 1, \chi_{0k} = \langle s_k^{(1)} \rangle, \chi_{l0} = \langle s_l^{(2)} \rangle, \chi_{kl} = \langle s_k^{(1)} s_l^{(2)} \rangle. \tag{11.7}$$

The classical spins $s_k^{(1)}, s_k^{(2)}, k = 1, 2, 3$ can take on values ± 1, overall the $64 = 2^6$ can therefore be assigned a probability distribution p_τ with τ running over all 64 different states. The expectation values and correlations in (11.7) can be obtained in the standard way by summing overall state values and multiplying by the probability p_τ. That is

$$\chi_{k0} = \sigma_\tau^{(k0)} p_\tau, \quad \chi_{0k} = \sigma_\tau^{(0k)} p_\tau, \quad \chi_{kl} = \sigma_\tau^{(kl)} p_\tau, \tag{11.8}$$

where the signs $\sigma_\tau^{(a)} = \pm 1$ are given by

$$\sigma_\tau^{(k0)} = (-1)^{1+\mathrm{bin}(\tau)[k]}, \tag{11.9}$$

$$\sigma_\tau^{(0k)} = (-1)^{1+\mathrm{bin}(\tau)[3+k]}, \tag{11.10}$$

$$\sigma_\tau^{(kl)} = (-1)^{1+\mathrm{bin}(\tau)[k]} (-1)^{1+\mathrm{bin}(\tau)[3+l]}. \tag{11.11}$$

Here I denote by $\mathrm{bin}(\tau)$ the binary representation of the state For example $\mathrm{bin}((-1,-1,-1,1,1,-1)) = (0,0,0,1,1,0)$ and $\mathrm{bin}(\tau)[k]$ takes the k-th entry, for example $\mathrm{bin}((-1,-1,-1,1,1,-1))[3] = 1$. This defines a map

$$g\colon \mathbf{R}^{64} \to \mathbf{R}^{4\times 4}, \quad p \mapsto \chi. \tag{11.12}$$

The *bit-quantum* map b is then a map from the classical binary probability distribution p to the quantum density matrix ρ

$$b = f \circ g\colon \mathbf{R}^{64} \to \mathbf{C}^{4\times 4}, \quad p \mapsto \rho. \tag{11.13}$$

It maps certain classical expectation values and correlations (11.8) to the expectation values and correlations of quantum spins in the cartesian directions. This construction is in some ways analogous to the reconstruction of the density matrix of photons from appropriate correlations [87, 120, 93]. In fig. **??** I show examples of classical probability distributions associated with certain quantum density matrices and in fig. 11.2 the corresponding expectation values and correlations. The goal of the next section is to explain how these can be optained by a simple optimization procedure.

11.3 Completeness of the correlation map for $Q = 2$

In order to demonstrate the completeness of the bit quantum map (11.13), I performed two numerical experiments. The aim is to show that to every postive hermintian normalized quantum density matrix ρ, there exists a classical binary

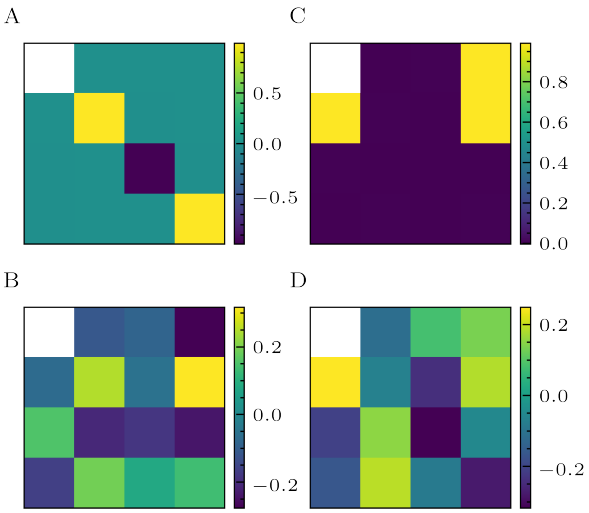


Figure 11.2: We show the $15 = 2 \times 3 + 9$ spin expectation values $\langle s_k^{(1)} \rangle$ and correlations $\langle s_k^{(1)} s_l^{(2)} \rangle$ corresponding to ψ_+ (A) and ρ (B). The first row contains the three expectation values $\langle s_k^{(1)} \rangle$, the first column the three expectation values $\langle s_k^{(2)} \rangle$. The remaining 3×3 entries are made up of the correlations $\langle s_k^{(1)} s_l^{(2)} \rangle$. The transformed spin expectation values and correlations related to the density matrices $\mathrm{CNOT}(\psi_+)$ and $\mathrm{CNOT}(\rho)$ are shown in (C) and (D) respectively. Note the different color scale between (A)(B) and (C)(D). Figure adapted from [123].

probability distribution p, such that ρ is the image of p under equation (11.13). We introduce a "classical wave function" q_τ, which is related to p_τ according to

$$h \colon \mathbf{R}^{64} \to \mathbf{R}^{64} q_\tau \mapsto p_\tau = \frac{q_\tau^2}{\sum_\tau q_\tau^2}. \tag{11.14}$$

The goal is therefore to find for a given density matrix $\rho \in \mathbf{C}^{4\times 4}$ a vector $q \in \mathbf{R}^{64}$, such that $\rho = (b \circ h)(q)$. Clearly such a q is not unique, as changing the sign of any of the components of q doesn't change the image. To find one such q we introduce the loss function

$$l_\rho(q) = |(b \circ h)(q) - \rho|_2^2. \tag{11.15}$$

We can then perform gradient descent with respect q according to this loss. In order to verify the effectiveness of this method of finding q we start with a randomly chosen density matrix ρ_0 and apply a sequence of random combinations of CNOT, Hadamard and rotations by $\pi/8$ to it, in order to obtain density matrices ρ_i. For growing i the sequence of unitary transformations densely covers the space of all possible unitary transformations. We then perform gradient descent optimization according to (11.15) to find q_i (and therefore p_i) associated to the density matrices $\rho_i = (b \circ h)(q_i)$. As can be seen in fig. 11.3 the optimization succeeds to find appropriate q_i after few iterations in all recorded cases. We generated many initial density matrices ρ_0 as well. This lends numerical support to the conclusion that for $Q = 2$ the bit quantum is complete.

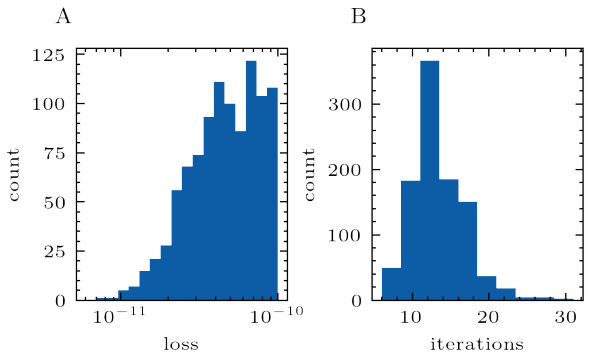


Figure 11.3: Optimization results for finding suitable "classical wave functions" $q_i \in \mathbf{R}^{64}$ for given density matrices ρ_i. We generate 10^3 samples by performing sequences of unitary transformations as described in the main text starting from a random initial density matrix ρ_0. Optimization is stopped once the loss $l_\rho(q)$ falls below 10^{-10}. (A) shows a histograma of final loss values and (B) a histogram of the required number of iterations. Figure adapted from [123].

We then ask a related question: How to find the transformation of q associated to a given transformation of $\rho = (b \circ h)(q)$. In other words we ask how a classical probabilistic object is transformed to implement a quantum operation on the associated density matrix. More precisely the goal is to find for a given vector $q \in \mathbf{R}^{64}$ associated to a density matrix $\rho = (b \circ h)(q)$ and a given unitary transformation U a matrix $M \in \mathbf{R}^{64\times 64}$, such that the transformed vector Mq is related to the transformed density matrix $\rho' = U\rho U^\dagger = (b \circ h)(Mq)$. This can again be regarded as a minimization problem, with loss given by

$$l_U(M) = |\mathrm{ad}_U((b \circ h)(q)) - (b \circ h)(Mq)|_2^2 \tag{11.16}$$

and the adjoint action given by $\mathrm{ad}_U(\rho) = U\rho U^\dagger$. The minimization of this loss can be again carried out by gradient descent, now with respect to M. Example results of this procedure are shown in fig. 11.1 as well as the resulting expectation values and correlations before and after the transformation in fig. 11.2. In this case we chose as the unitary transformation $U = \mathrm{CNOT}$ and investigated both the density matrix associated to a maximally entangled Bell-state $\psi_+ = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, as well as several randomly generated density matrices ρ. It is important to realise that M found in this way depends on q, that is it is a non-linear map $Mq = M(q)q$ [159].

11.4 Quantum computing with spiking neurons

So far we explicitely determined given a density matrix ρ a classical probability distribution p_τ which is mapped to ρ via the bit quantum map. We now want to turn to an implementation based on neuromorphic computing which directly realises classical Ising spin state variables, whose expectation values and correlations are given by temporal averages and can be used to implement the correlation map (11.4) directly. I already described the general idea of how to associate to a deterministic spiking neuron with refractory period a binary random variable in chapter 8.1. A neuron is considered active (or refractory) after

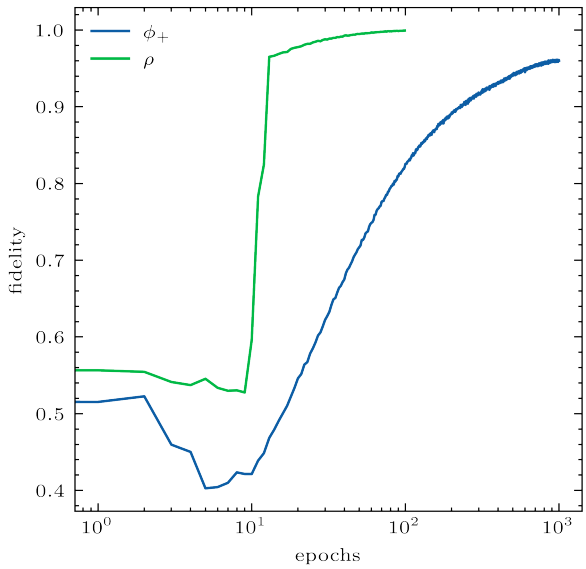


Figure 11.4: Fidelity as a function of training epoch of the representation of the density matrix of the Bell-state $\psi_+ = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ and a random density matrix ρ by a recurrent spiking neural network via the correlation map (11.4) Figure adapted from [123].

it has spiked for a certain refractory period t_{refrac} and silent otherwise. Using this framework pairwise correlations in biological neurons (e.g. [143, 118]) and relationships to spin-glass models [83] have been considered in the literature.

We consider a network of leaky-integrate and fire neurons with an absolute refractory period of t_{refrac}. The dynamical variables are the membrane voltage V, synaptic input current I and the refractory state variable R

$$\dot{V} = (1 - \Theta(R))(V_{\text{l}} - V + I) \tag{11.17}$$

$$\dot{I} = -I + I_{\text{in}} \tag{11.18}$$

$$\dot{R} = -\frac{1}{t_{\text{refrac}}}\Theta(R). \tag{11.19}$$

Here V_l is the leak potential and the input current is given by

$$(I_{\text{in}})_j = \sum_l (W_{\text{in}})_{j,q_l}\delta(t - t_l) \quad q_l \in 1, \ldots, m. \tag{11.20}$$

Just as in previous chapters the neuron dynamic of neuron i jumps if the jump condition

$$V_i - (V_{\text{th}})_i = 0 \tag{11.21}$$

is satisfied. The transition equations are then given by

$$V_j^+ = V_j^- + (V_r - V_{\text{th}})\delta_{ij}, \tag{11.22}$$

$$I_j^+ = I_j^- + (W_{\text{rec}})_{ij}, \tag{11.23}$$

$$R_j^+ = R_j^- + \delta_{ij}, \tag{11.24}$$

We can then define the refractory state as

$$z_i(t) = \Theta(R_i(t)), \tag{11.25}$$

where Θ denotes the Heaviside function and the classical spins are given by

$$s_i(t) = 2z_i(t) - 1. \tag{11.26}$$

We can then select 6 of the n spin variables to be the classical Ising spins we want to consider. For example we can take

$$\pi \colon \mathbf{R}^n \to \mathbf{R}^6, (s_1, \ldots, s_n) \mapsto (s_1^{(1)}, \ldots, s_3^{(2)}). \tag{11.27}$$

The time averages entering into (11.4) can then be computed by

$$\langle s_k^{(i)} \rangle = \frac{1}{T} \int_0^T s_k^{(i)}(t) \mathrm{dt}, \tag{11.28}$$

$$\langle s_k^{(1)} s_l^{(2)} \rangle = \frac{1}{T} \int_0^T s_k^{(1)}(t) s_l^{(2)}(t) \mathrm{dt}. \tag{11.29}$$

Given a density matrix ρ, we can then define the loss

$$l_\rho(W_{\mathrm{rec}}, W_{\mathrm{in}}) = |\rho - f(\chi(\pi(s)))|_2^2, \tag{11.30}$$

which defines an optimization problem for the recurrent and input weights $W_{\mathrm{rec}}, W_{\mathrm{in}}$. This can again be solved by gradient descent by either using a surrogate gradient as explained in chapter 8 or by using the adjoint method. Here we choose to use the surrogate gradient introduced in [171] with smoothing parameter $\alpha = 100$.

We show the results of the optimization process for the spin expectation and correlation matrices in figures 11.4, 11.5. The resulting recurrent weights restricted to the 6 Ising spins entering the correlation maps are shown in 11.6.

We considered a network with $n = 64$ recurrently connected neurons and 128 poisson input sources. We set the leak potential and threshhold to $V_l = 0$ and $V_{\mathrm{th}} = 1$. We use simple forward euler integration with time step $dt = 0.001$ and also set the refractory time to 1 ms, which eliminates the refractory variable from the integration. The numerical implementation is done in JAX [27]. The poisson input spike frequency is chosen to be 700 Hz and the gradient descent starts with a learning rate of 10 and decreases with a decay constant of 1/100.

We choose to evaluate the method both on a density matrix of a maximally entangled pure state $\psi_+ = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ and randomly generated density matrices ρ. In figure 11.4 the fidelity

$$F(\rho, \sigma) = (\mathrm{tr}\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}})^2 \tag{11.31}$$

is shown as a function of the training iterations. By the Fuchs–van de Graaf inequalities the trace norm bounds the Fidelity in the following way

$$1 - \sqrt{F(\rho, \sigma)} \leq \frac{1}{2}|\rho - \sigma|_1 \leq \sqrt{1 - F(\rho, \sigma)}\,. \tag{11.32}$$

We therefore expect the fidelity of the representation to increase as the loss decreases. As can be seen from the figure the fidelity increases somewhat slower for the pure state but nevertheless reaches acceptable levels after a modest amount of iterations.

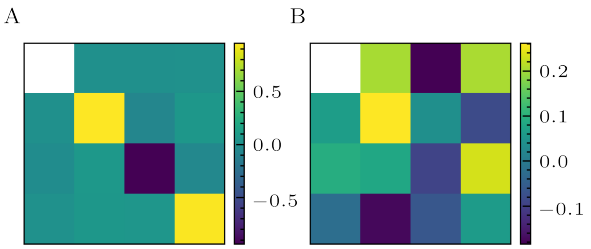


Figure 11.5: Spin expectation and correlation matrices χ corresponding to the pure state ψ_+ (A) and ρ (B) respectively. Figure adapted from [123].

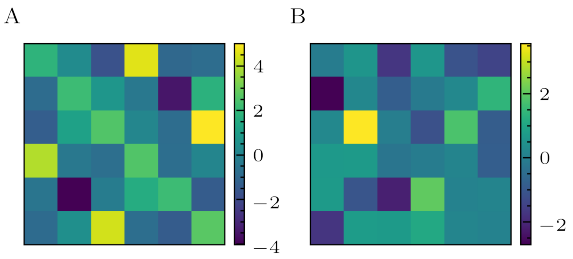


Figure 11.6: Final recurrent weight matrices W_{rec} or the 6 recurrently connected neurons, whose refractory state corresponds to the spins $s_k^{(i)}$ of the pure state ψ_+ (A) and ρ (B) respectively. Figure adapted from [123].

11.5 Generalisations to many qubits

A natural question is whether the minimal bit-quantum map with the correlation map given by eq. (11.4) remains for $Q > 2$. In this section I demonstrate that this is not true in general. This can be demonstrated by a simple experiment. Consider the one-parameter family of density matrices

$$\rho(p) = p\rho_{\text{ghz}} + (1-p)\bar{\rho}, p \in [0,1], \tag{11.33}$$

with ρ_{ghz} the density matrix of the GHZ state

$$\psi = \frac{1}{\sqrt{2}}(|++\rangle + |--\rangle). \tag{11.34}$$

Then by the same procedure as described in section 11.3 we can hope to find for given density matrix $\rho \in \mathbf{C}^{8\times 8}$ a vector $q \in \mathbf{R}^{512}$ such that it is mapped to ρ under a generalisation of the bit quantum map $\rho = (b \circ h)(q)$. Then using the same loss as in section 11.3, we perform gradient descent w.r.t. q.

In figure 11.7 and 11.8 we plot the final loss, fidelity and relative entropy as a function of p, as can be seen the final fidelity of the bit-quantum map after 10^4 optimization epochs is rather low for the density matrix ρ_{ghz}. We conclude that the minimal bit quantum map is not complete for $Q > 2$. An extended quantum map will be proposed in a revised version of [123].

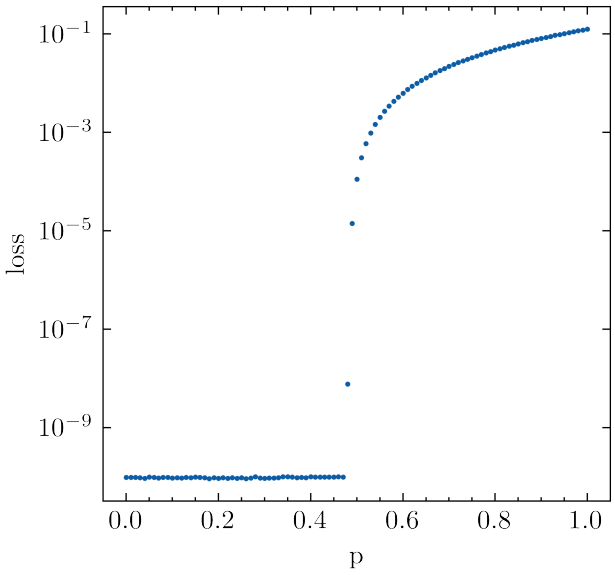


Figure 11.7: Final loss after training up to 10^4 epochs to approximate $\rho(p) = p\rho_{\text{ghz}} + (1-p)\bar{\rho}$, where ρ is a randomly chosen density matrix and ρ_{ghz} is the density matrix of the GHZ-state in dimension 3. Figure adapted from [123].

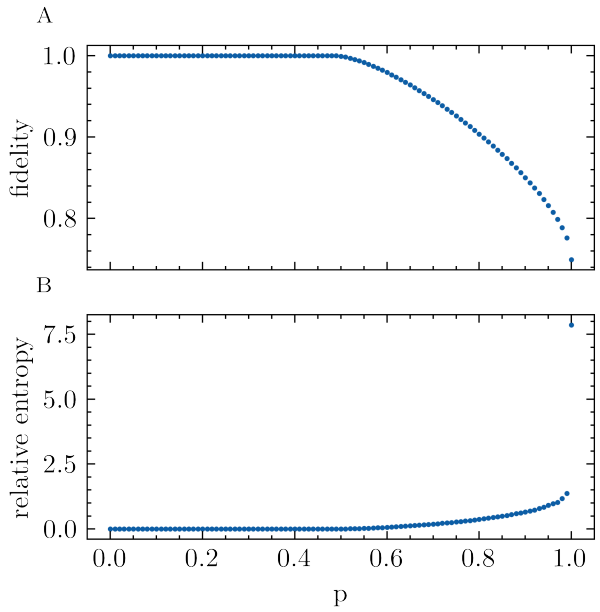


Figure 11.8: Final fidelity (A) and relative entropy (B) after training up to 10^4 epochs to approximate $\rho(p) = p\rho_{\text{ghz}} + (1-p)\bar{\rho}$, where ρ is a randomly chosen density matrix and ρ_{ghz} is the density matrix of the GHZ-state in dimension 3. Figure adapted from [123].

11.6 Incomplete probabilistic information

The use of a (spiking) neural network allows us to define correlation maps which are defined through multiple layers and the specific connectivity between the layers.

First recall the minimal correlation map [162] (already introduced in the previous chapter). Define the generators

$$L_{\mu_1\dots\mu_n} = \bigotimes_{i=1}^{n} \tau_{\mu_i} \quad \mu_i = 0\dots 3, \tag{11.35}$$

then a 2^n dimensional density matrix can be written as

$$\rho = \frac{1}{2^n}\chi_{\mu_1\dots\mu_n} L_{\mu_1\dots\mu_n}. \tag{11.36}$$

The coefficients $\chi_{\mu_1\dots\mu_n}$ are determined through expectation values and correlations of $3n$ spins as follows: Let

$$\sigma_{\mu_1\dots\mu_n}(t) = s^{(1)}_{\mu_1}(t)\dots s^{(n)}_{\mu_n}(t), \tag{11.37}$$

where $s^{(i)}_{\mu}(t) = (1, s^{(i)}_{k}(t))$ with $i = 1..n$, $k = 1..3$ and $\mu = 0\dots 3$. Then the coefficients are given by time-averaged expectation values:

$$\chi_{\mu_1\dots\mu_n} = \frac{1}{T}\int_0^T \sigma_{\mu_1\dots\mu_n}(t)\mathrm{dt}. \tag{11.38}$$

Even if this minimal correlation map were complete, the number of correlations to be computed would still increase rapidly as 2^{2n} with the number of qubits n. In many instances it seems unlikely that one needs to use the full information contained in the quantum density matrix ρ to do useful computations.

For example finding the ground state energy of a Hamiltonian typically involves only sparse information (see for example [33, 146]). In the case of the one-dimensional quantum Ising model with Hamiltonian

$$H = -J\sum_{i=1,\dots,n} \tau_3^{(i)}\tau_3^{(i+1)} - h\sum_i \tau_1^{(i)}, \tag{11.39}$$

the trace $\mathrm{tr}(H\rho)$ will only make use of few entries in eq. (11.36). Namely those two point functions $\chi_{0\dots 0\mu_k\mu_{k+1}\dots 0}$, $\mu_k = \mu_{k+1} = 3$ and the expectation values $\chi_{0\dots 0\mu_k\dots 0}, \mu_k = 1$. It might be the case that already the minimal correlation map can be used to represent this ground state with good fidelity.

In analogy to [146] one could also consider multiple layers of *spiking* neurons whose expectation values and correlations then enter based on a more general prescription than (11.37). That is some of the entries of $\chi_{\mu_1\dots\mu_n}$ with more than one non-zero μ_k, would be represented by the expectation value of a single classical spin, others as correlations. It remains to be seen whether this would result in successful implementation.

Since the computation of the expectation values $\chi_{\mu_1\dots\mu_n}$ is performed purely classically no special degree of isolation or low temperature is needed to implement the quantum properties. From the perspective of biological systems the realisation of correlations that (approximately) obey the quantum constraints

across already a moderate number of neurons, could present computational advantages, since such correlated states then allow for transformations which preserve information. The results presented in this chapter are an example how quantum evolution can be implemented in classical probabilistic systems [160, 162, 161].

Chapter 12

Outlook and Discussion

The results presented in this thesis leave open a mutlitude of avenues for future research. Broadly speaking my interest is to better understand ”learning”, ”intelligence” or ”self-adaptation” under the constraints of physical systems as opposed to simply algorithmically. Biological brains are examples of physical systems which exhibit these admittedly somewhat ill-defined properties. Conceptionally one would hope however that the principles underlying this capability are substrate and model independent. The discussion of ”Neural Processing Elements” in chapter 2 is an incomplete attempt of finding a way to conceptionally describe self-optimising hierachically composed networks of machines. Further work is needed to both fully work out the mathematical formalism along the lines outlined there and then to find practical ways of implementing it.

I restricted myself in this work for the most part to the simpler problem of gradient based parameter optimisation with respect to arbitrary loss function. By varying the loss function and the task this can nevertheless result in excellent performance on a wide variety of tasks as recent successes in Deep Learning have convincingly demonstrated. While the adjoint method itself cannot be regarded as a biologically plausible mechanism of parameter optimisation in itself, it can both serve as the starting point for approximations (as done in chapter 5) and as the outer loop in a meta-learning task with biologically plausible synaptic plasticity in the inner loop. By restricting oneself to plasticity rules based on ”realistic” chemical reaction networks and using loss functions that compare to experimental stimulation protocols on real neurons one might be able to fit plasticity rules from biological data.

In neuromorphic hardware the EventProp algorithm has a clear advantage over surrogate gradient methods in that it only requires very sparse information about the dynamics in the forward pass to compute the gradients and adjoint dynamics in the backwards pass. This should allow for an efficient implementation in particular in digital neuromorphic hardware. Similarly it should be possible to implement the algorithm in digital-analog neuromorphic hardware by using an in-the-loop training approach [40]. Since the adjoint method extends to structured neuron models it also allows one to train structured neuron models such as the one realised on BrainScaleS-2 by gradient based methods. It therefore is also a natural candidate for inclusion in neuron simulators of structured neurons such as arbor [5].

Looking beyond spiking neural networks, the adjoint dynamics with jumps

can also be applied to other dynamical nets. In particular approaches to novel computing based on often unreliable nano-devices can make use of it. This then opens the question under which circumstances a physical system can naturally accomodate the adjoint dynamics. I think this could be of particular interest in the context of "quantum neuromorphic computing" as reviewed in [109].

Finally I think there are still many interesting ways to explore the relationship between classical stochastic systems and quantum systems as discussed in chapter 11.

Co-authored Papers

- Christian Pehle and Christof Wetterich. *Neuromorphic quantum computing*. 2020. arXiv: `2005.01533 [cond-mat.dis-nn]` is the basis of chapter 11.

- Timo C Wunderlich and Christian Pehle. "EventProp: Backpropagation for Exact Gradients in Spiking Neural Networks". In: *arXiv preprint arXiv:2009.08378* (2020) results are incorporated into 3.

- Benjamin Cramer, Sebastian Billaudelle, Simeon Kanya, Aron Leibfried, Andreas Grübl, Vitali Karasenko, Christian Pehle, Korbinian Schreiber, Yannik Stradmann, Johannes Weis, Johannes Schemmel, and Friedemann Zenke. *Training spiking multi-layer networks with surrogate gradients on an analog neuromorphic substrate*. 2020. arXiv: `2006.07239 [cs.NE]`

- Philipp Spilger, Eric Müller, Arne Emmel, Aron Leibfried, Christian Mauch, Christian Pehle, Johannes Weis, Oliver Breitwieser, Sebastian Billaudelle, Sebastian Schmitt, Timo C. Wunderlich, Yannik Stradmann, and Johannes Schemmel. *hxtorch: PyTorch for BrainScaleS-2 – Perceptrons on Analog Neuromorphic Hardware*. 2020. arXiv: `2006.13138 [cs.NE]`

- K Schreiber, TC Wunderlich, C Pehle, MA Petrovici, J Schemmel, and K Meier. "Closed-loop experiments on the BrainScaleS-2 architecture". In: *arXiv preprint arXiv:2004.14829* (2020)

- Sebastian Billaudelle, Yannik Stradmann, Korbinian Schreiber, Benjamin Cramer, Andreas Baumbach, Dominik Dold, Julian Göltz, Akos F Kungl, Timo C Wunderlich, Andreas Hartel, et al. "Versatile emulation of spiking neural networks on an accelerated neuromorphic substrate". In: *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2020, pp. 1–5

- Thomas Bohnstingl, Franz Scherr, Christian Pehle, Karlheinz Meier, and Wolfgang Maass. "Neuromorphic hardware learns to learn". In: *Frontiers in neuroscience* 13 (2019), p. 483

- Christian Pehle, Karlheinz Meier, Markus Oberthaler, and Christof Wetterich. "Emulating quantum computation with artificial neural networks". In: *arXiv preprint arXiv:1810.10335* (2018) is the basis of chapter 10.

- Syed Ahmed Aamir, Yannik Stradmann, Paul Müller, Christian Pehle, Andreas Hartel, Andreas Grübl, Johannes Schemmel, and Karlheinz Meier.

"An accelerated lif neuronal network array for a large-scale mixed-signal neuromorphic architecture". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 65.12 (2018), pp. 4299–4312

- SA Aamir, Y Stradmann, P Müller, Christian Pehle, A Hartel, A Grübl, J Schemmel, and K Meier. "An accelerated LIF neuronal network array for a large scale neuromorphic system". In: *IEEE Transactions of Circuits and Systems-I* (2017)

Bibliography

[1] SA Aamir et al. "An accelerated LIF neuronal network array for a large scale neuromorphic system". In: *IEEE Transactions of Circuits and Systems-I* (2017).

[2] Syed Ahmed Aamir et al. "An accelerated lif neuronal network array for a large-scale mixed-signal neuromorphic architecture". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 65.12 (2018), pp. 4299–4312.

[3] Martín Abadi et al. "Tensorflow: A system for large-scale machine learning". In: *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 2016, pp. 265–283.

[4] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[5] N. A. Akar et al. "Arbor — A Morphologically-Detailed Neural Network Simulation Library for Contemporary High-Performance Computing Architectures". In: *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. 2019, pp. 274–282. DOI: `10.1109/EMPDP.2019.8671560`.

[6] Mohamed Akrout et al. "Deep learning without weight transport". In: *Advances in neural information processing systems*. 2019, pp. 976–984.

[7] Sacha Jennifer van Albada et al. "Performance comparison of the digital neuromorphic hardware SpiNNaker and the neural network simulation software NEST for a full-scale cortical microcircuit model". In: *Frontiers in neuroscience* 12 (2018), p. 291.

[8] Kai Arulkumaran et al. *PyTorch Examples*. `https://github.com/pytorch/examples`. 2020.

[9] Giorgio A Ascoli, Duncan E Donohue, and Maryam Halavi. "NeuroMorpho. Org: a central resource for neuronal morphologies". In: *Journal of Neuroscience* 27.35 (2007), pp. 9247–9251.

[10] Giorgio A Ascoli, Duncan E Donohue, and Maryam Halavi. *Neuromorpho.org*. `http://neuromorpho.org`. Accessed: 2020-11-01.

[11] John C Baez and Aaron Lauda. "A prehistory of n-categorical physics". In: *Deep Beauty: Understanding the Quantum World Through Mathematical Innovation* (2011), pp. 13–128.

[12] Andrew G Barto, Richard S Sutton, and Charles W Anderson. "Neuronlike adaptive elements that can solve difficult learning control problems". In: *IEEE transactions on systems, man, and cybernetics* 5 (1983), pp. 834–846.

[13] Anthony J. Bell and Lucas C. Parra. "Maximising Sensitivity in a Spiking Network". In: *Advances in Neural Information Processing Systems 17*. Ed. by L. K. Saul, Y. Weiss, and L. Bottou. MIT Press, 2005, pp. 121–128. URL: http://papers.nips.cc/paper/2674-maximising-sensitivity-in-a-spiking-network.pdf.

[14] Guillaume Bellec et al. "A solution to the learning dilemma for recurrent networks of spiking neurons". In: *bioRxiv* (2020), p. 738385.

[15] Guillaume Bellec et al. "A solution to the learning dilemma for recurrent networks of spiking neurons". In: *Nature Communications* 11.1 (2020), p. 3625. URL: https://doi.org/10.1038/s41467-020-17236-y.

[16] Guillaume Bellec et al. "Biologically inspired alternatives to backpropagation through time for learning in recurrent neural nets". In: *arXiv preprint arXiv:1901.09049* (2019).

[17] Guillaume Bellec et al. "Long short-term memory and learning-to-learn in networks of spiking neurons". In: *Advances in Neural Information Processing Systems*. 2018, pp. 787–797.

[18] Yoshua Bengio et al. "Towards Biologically Plausible Deep Learning". In: *CoRR* abs/1502.04156 (2015). arXiv: 1502.04156. URL: http://arxiv.org/abs/1502.04156.

[19] Ben Varkey Benjamin et al. "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations". In: *Proceedings of the IEEE* 102.5 (2014), pp. 699–716.

[20] Albert Benveniste et al. "Multi-Mode DAE Models - Challenges, Theory and Implementation." In: *Computing and Software Science: state of the Art and Perspectives*. Ed. by Woeginger G. Steffen B. Vol. 10000. Lecture Notes in Computer Science. Springer, 2019.

[21] Sebastian Billaudelle et al. *Structural plasticity on an accelerated analog neuromorphic hardware system*. 2020. arXiv: 1912.12047 [q-bio.NC].

[22] Sebastian Billaudelle et al. "Versatile emulation of spiking neural networks on an accelerated neuromorphic substrate". In: *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2020, pp. 1–5.

[23] Thomas Bohnstingl et al. "Neuromorphic hardware learns to learn". In: *Frontiers in neuroscience* 13 (2019), p. 483.

[24] Sander M Bohte, Joost N Kok, and Johannes A La Poutré. "SpikeProp: backpropagation for networks of spiking neurons." In: *ESANN*. Vol. 48. 2000, pp. 17–37.

[25] Olaf Booij and Hieu tat Nguyen. "A gradient descent rule for spiking neurons emitting multiple spikes". In: *Information Processing Letters* 95.6 (2005). Applications of Spiking Neural Networks, pp. 552 –558. ISSN: 0020-0190. DOI: https://doi.org/10.1016/j.ipl.2005.05.023. URL: http://www.sciencedirect.com/science/article/pii/S0020019005001560.

[26] Timothy Bourke and Marc Pouzet. "ZÃ©lus: A Synchronous Language with ODEs". In: *16th International Conference on Hybrid Systems: Computation and Control (HSCC'13)*. Philadelphia, USA, Mar. 2013, pp. 113–118. URL: http://www.di.ens.fr/~pouzet/bib/hscc13.pdf.

[27] James Bradbury et al. *JAX: composable transformations of Python+NumPy programs*. Version 0.1.55. 2018. URL: http://github.com/google/jax.

[28] Greg Brockman et al. "Openai gym". In: *arXiv preprint arXiv:1606.01540* (2016).

[29] Michael Broughton et al. *TensorFlow Quantum: A Software Framework for Quantum Machine Learning*. 2020. arXiv: 2003.02989 [quant-ph].

[30] Lars Buesing et al. "Neural dynamics as sampling: a model for stochastic computation in recurrent networks of spiking neurons". In: *PLoS computational biology* 7.11 (2011), e1002211.

[31] Lars Buesing et al. "Neural Dynamics as Sampling: A Model for Stochastic Computation in Recurrent Networks of Spiking Neurons". In: *PLOS Computational Biology* 7.11 (Nov. 2011), pp. 1–22. DOI: 10.1371/journal.pcbi.1002211. URL: https://doi.org/10.1371/journal.pcbi.1002211.

[32] G. Carleo and M. Troyer. "Solving the quantum many-body problem with artificial neural networks". In: *Science* 355 (Feb. 2017), pp. 602–606. DOI: 10.1126/science.aag2302. arXiv: 1606.02318 [cond-mat.dis-nn].

[33] G. Carleo and M. Troyer. *Solving the quantum many-body problem with artificial neural networks*. Feb. 2017. arXiv: 1606.02318 [cond-mat.dis-nn].

[34] Ricky T. Q. Chen, Brandon Amos, and Maximilian Nickel. *Learning Neural Event Functions for Ordinary Differential Equations*. 2020. arXiv: 2011.03902 [cs.LG].

[35] Tian Qi Chen et al. "Neural ordinary differential equations". In: *Advances in neural information processing systems*. 2018, pp. 6571–6583.

[36] François Chollet et al. *Keras*. https://keras.io. 2015.

[37] Alain Connes and Dirk Kreimer. "Hopf algebras, renormalization and noncommutative geometry". In: *Quantum field theory: perspective and prospective*. Springer, 1999, pp. 59–109.

[38] Sebastien Corner, Adrian Sandu, and Corina Sandu. "Adjoint sensitivity analysis of hybrid multibody dynamical systems". In: *Multibody System Dynamics* (2020), pp. 1–26.

[39] Matthieu Courbariaux et al. "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1". In: *arXiv preprint arXiv:1602.02830* (2016).

[40] Benjamin Cramer et al. *Training spiking multi-layer networks with surrogate gradients on an analog neuromorphic substrate*. 2020. arXiv: 2006.07239 [cs.NE].

[41] Francis Crick. "The recent excitement about neural networks". In: *Nature* 337.6203 (1989), pp. 129–132.

[42] Stefanie Czischek et al. *Spiking neuromorphic chip learns entangled quantum states*. 2020. arXiv: 2008.01039 [cs.ET].

[43] Max van Daalen, John Shawe-Taylor, and Jieyu Zhao. "Learning in Stochastic Bit Stream Neural Networks". In: *Neural networks : the official journal of the International Neural Network Society* 9.6 (1996), pp. 991–998. ISSN: 0893-6080. DOI: 10.1016/0893-6080(96)00025-1. URL: https://doi.org/10.1016/0893-6080(96)00025-1.

[44] M. Davies et al. "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning". In: *IEEE Micro* 38.1 (2018), pp. 82–99.

[45] Mike Davies et al. "Loihi: A neuromorphic manycore processor with on-chip learning". In: *IEEE Micro* 38.1 (2018), pp. 82–99.

[46] Peter Dayan and Laurence F Abbott. *Theoretical neuroscience: computational and mathematical modeling of neural systems*. Computational Neuroscience Series, 2001.

[47] J. Dean, D. Patterson, and C. Young. "A New Golden Age in Computer Architecture: Empowering the Machine-Learning Revolution". In: *IEEE Micro* 38.2 (2018), pp. 21–29. ISSN: 0272-1732. DOI: 10.1109/MM.2018.112130030.

[48] Conal Elliott. "Compiling to Categories". In: *Proc. ACM Program. Lang.* 1.ICFP (Aug. 2017), 27:1–27:27. ISSN: 2475-1421. DOI: 10.1145/3110271. URL: http://doi.acm.org/10.1145/3110271.

[49] Conal Elliott. "The simple essence of automatic differentiation". In: *Proceedings of the ACM on Programming Languages* 2.ICFP (2018), p. 70.

[50] Steve K Esser et al. "Backpropagation for energy-efficient neuromorphic computing". In: *Advances in Neural Information Processing Systems*. 2015, pp. 1117–1125.

[51] Steven K. Esser et al. "Convolutional networks for fast, energy-efficient neuromorphic computing". In: *Proceedings of the National Academy of Sciences* 113.41 (2016), pp. 11441–11446. ISSN: 0027-8424. DOI: 10.1073/pnas.1604850113. eprint: https://www.pnas.org/content/113/41/11441.full.pdf. URL: https://www.pnas.org/content/113/41/11441.

[52] M. Faix et al. "Design of Stochastic Machines Dedicated to Approximate Bayesian inferences". In: *IEEE Transactions on Emerging Topics in Computing* (2018), pp. 1–1. ISSN: 2168-6750. DOI: 10.1109/TETC.2016.2609926.

[53] Richard Phillips Feynman. "Space-time approach to quantum electrodynamics". In: *Physical Review* 76.6 (1949), p. 769.

[54] Răzvan V. Florian. "The Chronotron: A Neuron That Learns to Fire Temporally Precise Spike Patterns". In: *PLOS ONE* 7.8 (Aug. 2012), pp. 1–27. DOI: 10.1371/journal.pone.0040233. URL: https://doi.org/10.1371/journal.pone.0040233.

[55] Brendan Fong. "The algebra of open and interconnected systems". In: *arXiv preprint arXiv:1609.05382* (2016).

[56] J. S. Friedman et al. "Bayesian Inference With Muller C-Elements". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 63.6 (2016), pp. 895–904. ISSN: 1549-8328. DOI: 10.1109/TCSI.2016.2546064.

[57] Simon Friedmann. "A new approach to learning in neuromorphic hardware". PhD thesis. Heidelberg University, 2013.

[58] Simon Friedmann et al. "Demonstrating hybrid learning in a flexible neuromorphic hardware system". In: *IEEE transactions on biomedical circuits and systems* 11.1 (2017), pp. 128–142.

[59] Kurt Otto Friedrichs. "The identity of weak and strong extensions of differential operators". In: *Transactions of the American Mathematical Society* 55.1 (1944), pp. 132–151.

[60] S. B. Furber et al. "The SpiNNaker Project". In: *Proceedings of the IEEE* 102.5 (2014), pp. 652–665.

[61] Steve Furber. "Large-scale neuromorphic computing systems". In: *Journal of neural engineering* 13.5 (2016), p. 051001.

[62] Steve B Furber et al. "The spinnaker project". In: *Proceedings of the IEEE* 102.5 (2014), pp. 652–665.

[63] Santos Galán, William F. Feehery, and Paul I. Barton. "Parametric sensitivity functions for hybrid discrete/continuous systems". In: *Applied Numerical Mathematics* 31.1 (1999), pp. 17 –47. ISSN: 0168-9274. DOI: https://doi.org/10.1016/S0168-9274(98)00125-1. URL: http://www.sciencedirect.com/science/article/pii/S0168927498001251.

[64] Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber. "Learning precise timing with LSTM recurrent networks". In: *Journal of machine learning research* 3.Aug (2002), pp. 115–143.

[65] W. Gerstner et al. "A neuronal learning rule for sub-millisecond temporal coding". In: *Nature* 383.6595 (1996). article, pp. 76–78.

[66] Wulfram Gerstner et al. *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.

[67] Marc-Oliver Gewaltig and Markus Diesmann. "NEST (NEural Simulation Tool)". In: *Scholarpedia* 2.4 (2007), p. 1430.

[68] Dan R Ghica. "Geometry of synthesis: a structured approach to VLSI design". In: *Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 2007, pp. 363–375.

[69] Albert Gidon et al. "Dendritic action potentials and computation in human layer 2/3 cortical neurons". In: *Science* 367.6473 (2020), pp. 83–87.

[70] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, pp. 315–323.

[71] Julian Göltz et al. "Fast and deep neuromorphic learning with time-to-first-spike coding". In: *arXiv preprint arXiv:1912.11443* (2019).

[72] T. H. Gronwall. "Note on the Derivatives with Respect to a Parameter of the Solutions of a System of Differential Equations". In: *Annals of Mathematics* 20.4 (1919), pp. 292–296. ISSN: 0003486X. URL: http://www.jstor.org/stable/1967124.

[73] Stephen Grossberg. "Competitive learning: From interactive activation to adaptive resonance". In: *Cognitive science* 11.1 (1987), pp. 23–63.

[74] Andreas Grübl et al. "Verification and Design Methods for the BrainScaleS Neuromorphic Hardware System". In: *Journal of Signal Processing Systems* 92.11 (2020), pp. 1277–1292. DOI: 10.1007/s11265-020-01558-7. URL: https://doi.org/10.1007/s11265-020-01558-7.

[75] Pedro Guedes-Dias and Erika LF Holzbaur. "Axonal transport: Driving synaptic function". In: *Science* 366.6462 (2019), eaaw9997.

[76] Robert Gütig. "Spiking neurons can discover predictive features by aggregate-label learning". In: *Science* 351.6277 (2016).

[77] Demis Hassabis et al. "Neuroscience-inspired artificial intelligence". In: *Neuron* 95.2 (2017), pp. 245–258.

[78] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[79] John Hennessy and David Patterson. "A new golden age for computer architecture: Domain-specific hardware/software co-design, enhanced security, open instruction sets, and agile chip development". In: *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 2018, pp. 27–29. DOI: 10.1109/ISCA.2018.00011.

[80] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[81] A. L. Hodgkin and A. F. Huxley. "A quantitative description of membrane current and its application to conduction and excitation in nerve". In: *The Journal of Physiology* 117.4 (1952), pp. 500–544. DOI: 10.1113/jphysiol.1952.sp004764.

[82] Gerard 't Hooft. "A planar diagram theory for strong interactions". In: *The Large N Expansion In Quantum Field Theory And Statistical Physics: From Spin Systems to 2-Dimensional Gravity*. World Scientific, 1993, pp. 80–92.

[83] John J Hopfield. "Neural networks and physical systems with emergent collective computational abilities". In: *Proceedings of the national academy of sciences* 79.8 (1982), pp. 2554–2558.

[84] Dongsung Huh and Terrence J Sejnowski. "Gradient Descent for Spiking Neural Networks". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio et al. Curran Associates, Inc., 2018, pp. 1433–1443. URL: http://papers.nips.cc/paper/7417-gradient-descent-for-spiking-neural-networks.pdf.

[85] "IEEE Standard for Universal Verification Methodology Language Reference Manual". In: *IEEE Std 1800.2-2017* (2017), pp. 1–472. DOI: 10.1109/IEEESTD.2017.7932212.

[86] Max Jaderberg et al. "Decoupled neural interfaces using synthetic gradients". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1627–1635.

[87] Daniel F. V. James et al. "Measurement of qubits". In: *Phys. Rev. A* 64 (5 2001), p. 052312. DOI: 10.1103/PhysRevA.64.052312. URL: https://link.aps.org/doi/10.1103/PhysRevA.64.052312.

[88] Junteng Jia and Austin R Benson. "Neural jump stochastic differential equations". In: *Advances in Neural Information Processing Systems*. 2019, pp. 9843–9854.

[89] David Kappel et al. "Synaptic sampling: a bayesian approach to neural network plasticity and rewiring". In: *Advances in Neural Information Processing Systems* 28 (2015), pp. 370–378.

[90] Hilbert J Kappen. "Linear theory for control of nonlinear stochastic systems". In: *Physical review letters* 95.20 (2005), p. 200201.

[91] Hilbert J Kappen. "Path integrals and symmetry breaking for optimal control theory". In: *Journal of statistical mechanics: theory and experiment* 2005.11 (2005), P11011.

[92] Vitali Karasenko. *Von Neumann bottlenecks in non-von Neumann computing architectures*. 2020.

[93] Nikolai Kiesel. "Experiments on multiphoton entanglement". PhD thesis. lmu, 2007.

[94] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[95] Christof Koch and Allan Jones. "Big science, team science, and open science for neuroscience". In: *Neuron* 92.3 (2016), pp. 612–616.

[96] Philip Koopman. *Maximal Length LFSR Feedback Terms*. http://users.ece.cmu.edu/~koopman/lfsr/index.html. Accessed: 2020-09-30.

[97] Alex Krizhevsky, Geoffrey Hinton, et al. "Learning multiple layers of features from tiny images". In: (2009).

[98] Alex Krizhevsky, Geoffrey Hinton, et al. *Learning multiple layers of features from tiny images*. 2009.

[99] Akos F Kungl et al. "Accelerated physical emulation of Bayesian inference in spiking neural networks". In: *Frontiers in neuroscience* 13 (2019), p. 1201.

[100] Joachim Lambek. "Cartesian Closed Categories and Typed Lambda-calculi." In: May 1985, pp. 136–175. DOI: 10.1007/3-540-17184-3_44.

[101] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444. DOI: 10.1038/nature14539. URL: https://doi.org/10.1038/nature14539.

[102] Yann LeCun and Corinna Cortes. *MNIST handwritten digit database*. http://yann.lecun.com/exdb/mnist/. 2010. URL: http://yann.lecun.com/exdb/mnist/.

[103] Eugene Lerman and James Schmidt. "Networks of hybrid open systems". In: *Journal of Geometry and Physics* 149 (2020), p. 103582.

[104] Eugene Lerman and David I Spivak. "An algebra of open continuous time dynamical systems and networks". In: *arXiv* (2016), arXiv–1602.

[105] Yoav Levine et al. "Quantum entanglement in deep learning architectures". In: *Physical review letters* 122.6 (2019), p. 065301.

[106] S. Linnainmaa. "The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors". MA thesis. Univ. Helsinki, 1970.

[107] Seppo Linnainmaa. "Taylor expansion of the accumulated rounding error". English. In: *BIT Numerical Mathematics* 16.2 (1976), pp. 146–160.

[108] Michael London and Michael Häusser. "Dendritic computation". In: *Annu. Rev. Neurosci.* 28 (2005), pp. 503–532.

[109] Danijela Marković and Julie Grollier. "Quantum neuromorphic computing". In: *Applied Physics Letters* 117.15 (2020), p. 150501. ISSN: 1077-3118. DOI: 10.1063/5.0020014. URL: http://dx.doi.org/10.1063/5.0020014.

[110] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[111] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[112] Carver Mead. "Neuromorphic electronic systems". In: *Proceedings of the IEEE* 78.10 (1990), pp. 1629–1636.

[113] Paul A Merolla et al. "A million spiking-neuron integrated circuit with a scalable communication network and interface". In: *Science* 345.6197 (2014), pp. 668–673.

[114] Mihai A. Petrovici, Johannes Bill, Ilja Bytschok, Johannes Schemmel and Karlheinz Meier. *Stochastic inference with deterministic spiking neurons.* 2013.

[115] Eric Müller et al. *Extending BrainScaleS OS for BrainScaleS-2.* 2020. arXiv: 2003.13750 [cs.NE].

[116] Emre Neftci et al. "Event-Driven Contrastive Divergence for Spiking Neuromorphic Systems". In: *CoRR* abs/1311.0966 (2013). arXiv: 1311.0966. URL: http://arxiv.org/abs/1311.0966.

[117] Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. "Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks". In: *IEEE Signal Processing Magazine* 36.6 (2019), pp. 51–63.

[118] Ifije E Ohiorhenuan et al. "Sparse coding and high-order correlations in fine-scale cortical networks". In: *Nature* 466.7306 (2010), pp. 617–621.

[119] Takako Ohno-Shosaku. "Retrograde Messenger". In: *Encyclopedia of Neuroscience*. Ed. by Marc D. Binder, Nobutaka Hirokawa, and Uwe Windhorst. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 3529–3533. ISBN: 978-3-540-29678-2. DOI: 10.1007/978-3-540-29678-2_5123. URL: https://doi.org/10.1007/978-3-540-29678-2_5123.

[120] Matteo Paris and Jaroslav Rehacek. *Quantum state estimation*. Vol. 649. Springer Science & Business Media, 2004.

[121] Adam Paszke et al. "Automatic differentiation in PyTorch". In: *NIPS-W*. 2017.

[122] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[123] Christian Pehle and Christof Wetterich. *Neuromorphic quantum computing*. 2020. arXiv: 2005.01533 [cond-mat.dis-nn].

[124] Christian Pehle et al. "Emulating quantum computation with artificial neural networks". In: *arXiv preprint arXiv:1810.10335* (2018).

[125] Jing Pei et al. "Towards artificial general intelligence with hybrid Tianjic chip architecture". In: *Nature* 572.7767 (2019), pp. 106–111.

[126] Mihai A. Petrovici et al. "Stochastic inference with spiking neurons in the high-conductance state". In: *Phys. Rev. E* 94 (4 2016), p. 042312. DOI: 10.1103/PhysRevE.94.042312. URL: https://link.aps.org/doi/10.1103/PhysRevE.94.042312.

[127] D. Pinna et al. "Skyrmion Gas Manipulation for Probabilistic Computing". In: *Physical Review Applied* 9, 064018 (June 2018), p. 064018. arXiv: 1701.07750 [cond-mat.mes-hall].

[128] Lev Semenovich Pontryagin. *Mathematical theory of optimal processes*. Routledge, 1962.

[129] Ning Qiao and Giacomo Indiveri. "Scaling mixed-signal neuromorphic processors to 28 nm FD-SOI technologies". In: *Biomedical Circuits and Systems Conference (BioCAS), 2016 IEEE*. IEEE. 2016, pp. 552–555.

[130] Ning Qiao et al. "A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses". In: *Frontiers in neuroscience* 9 (2015), p. 141.

[131] Chris Rackauckas et al. "Diffeqflux.jl-A julia library for neural differential equations". In: *arXiv preprint arXiv:1902.02376* (2019).

[132] Christopher Rackauckas et al. "A comparison of automatic differentiation and continuous sensitivity analysis for derivatives of differential equation solutions". In: *arXiv preprint arXiv:1812.01892* (2018).

[133] Christopher Rackauckas et al. "Universal Differential Equations for Scientific Machine Learning". In: *arXiv preprint arXiv:2001.04385* (2020).

[134] AJ Robinson and Frank Fallside. *The utility driven dynamic error propagation network*. University of Cambridge Department of Engineering Cambridge, MA, 1987.

[135] Miguel Romera et al. “Vowel recognition with four coupled spin-torque nano-oscillators”. In: *Nature* 563.7730 (2018), pp. 230–234.

[136] Stefan Rotter and Markus Diesmann. “Exact digital simulation of time-invariant linear systems with applications to neuronal modeling”. In: *Biological cybernetics* 81.5-6 (1999), pp. 381–402.

[137] EN Rozenvasser. “General sensitivity equations of discontinuous systems”. In: *Automatika i telemekhanika* 3 (1967), pp. 52–56.

[138] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. “Learning Internal Representations by Error Propagation”. In: *Parallel Distributed Processing*. Ed. by D. E. Rumelhart and J. L. McClelland. Vol. 1. MIT Press, 1986, pp. 318–362.

[139] J. Schemmel et al. “A Wafer-Scale Neuromorphic Hardware System for Large-Scale Neural Modeling”. In: *Proceedings of the 2010 IEEE International Symposium on Circuits and Systems (ISCAS”10)* (2010), pp. 1947–1950.

[140] Johannes Schemmel et al. “An accelerated analog neuromorphic hardware system emulating NMDA-and calcium-based non-linear dendrites”. In: *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2017, pp. 2217–2226.

[141] Jürgen Schmidhuber. “Deep learning in neural networks: An overview”. In: *Neural networks* 61 (2015), pp. 85–117.

[142] Sebastian Schmitt et al. “Neuromorphic hardware in the loop: Training a deep spiking network on the brainscales wafer-scale system”. In: *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2017, pp. 2227–2234.

[143] Elad Schneidman et al. “Weak pairwise correlations imply strongly correlated network states in a neural population”. In: *Nature* 440.7087 (2006), pp. 1007–1012.

[144] K Schreiber et al. “Closed-loop experiments on the BrainScaleS-2 architecture”. In: *arXiv preprint arXiv:2004.14829* (2020).

[145] Abhronil Sengupta et al. “Going deeper in spiking neural networks: Vgg and residual architectures”. In: *Frontiers in neuroscience* 13 (2019), p. 95.

[146] Or Sharir et al. “Deep Autoregressive Models for the Efficient Variational Simulation of Many-Body Quantum Systems”. In: *Phys. Rev. Lett.* 124 (2 2020), p. 020503. DOI: 10.1103/PhysRevLett.124.020503. URL: https://link.aps.org/doi/10.1103/PhysRevLett.124.020503.

[147] Sumit Bam Shrestha and Garrick Orchard. “Slayer: Spike layer error reassignment in time”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 1412–1421.

[148] Philipp Spilger et al. *hxtorch: PyTorch for BrainScaleS-2 – Perceptrons on Analog Neuromorphic Hardware*. 2020. arXiv: 2006.13138 [cs.NE].

[149] David I. Spivak. “The operad of wiring diagrams: formalizing a graphical language for databases, recursion, and plug-and-play circuits”. In: *CoRR* abs/1305.0297 (2013). arXiv: 1305.0297. URL: http://arxiv.org/abs/1305.0297.

[150] Giacomo Torlai et al. "Neural-network quantum state tomography". In: *Nature Physics* 14.5 (2018), pp. 447–450. DOI: 10.1038/s41567-018-0048-5. URL: https://doi.org/10.1038/s41567-018-0048-5.

[151] Jacob Torrejon et al. "Neuromorphic computing with nanoscale spintronic oscillators". In: *Nature* 547.7664 (2017), p. 428.

[152] Roger D Traub et al. "A model of a CA3 hippocampal pyramidal neuron incorporating voltage-clamp data on intrinsic conductances". In: *Journal of neurophysiology* 66.2 (1991), pp. 635–650.

[153] AM Turing. "Computing Machinery and Intelligence". In: (1950).

[154] Robert Urbanczik and Walter Senn. "Learning by the dendritic prediction of somatic spiking". In: *Neuron* 81.3 (2014), pp. 521–528.

[155] Dmitry Vagner, David I Spivak, and Eugene Lerman. "Algebras of open dynamical systems on the operad of wiring diagrams". In: *arXiv preprint arXiv:1408.1598* (2014).

[156] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017), pp. 5998–6008.

[157] John Von Neumann. "Probabilistic logics and the synthesis of reliable organisms from unreliable components". In: *Automata studies* 34 (1956), pp. 43–98.

[158] John Von Neumann. "Probabilistic logics and the synthesis of reliable organisms from unreliable components". In: *Automata studies* 34 (1956), pp. 43–98.

[159] C. Wetterich. "Quantum computing with classical bits". In: *Nucl. Phys. B* 948 (2019), p. 114776. DOI: 10.1016/j.nuclphysb.2019.114776. arXiv: 1806.05960 [quant-ph].

[160] C Wetterich. "Quantum formalism for classical statistics". In: *Annals of Physics* 393 (2018), pp. 1–70. arXiv: 1706.01772.

[161] C. Wetterich. *The probabilistic world*. 2020. arXiv: 2011.02867 [quant-ph].

[162] Christof Wetterich. "Quantum computing with classical bits". In: *Nuclear Physics B* 948 (2019), p. 114776.

[163] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8.3-4 (1992), pp. 229–256.

[164] Ronald J Williams and David Zipser. "A learning algorithm for continually running fully recurrent neural networks". In: *Neural computation* 1.2 (1989), pp. 270–280.

[165] Timo C Wunderlich and Christian Pehle. "EventProp: Backpropagation for Exact Gradients in Spiking Neural Networks". In: *arXiv preprint arXiv:2009.08378* (2020).

[166] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. Aug. 28, 2017. arXiv: cs.LG/1708.07747 [cs.LG].

[167] Yan Xu et al. "A supervised multi-spike learning algorithm based on gradient descent for spiking neural networks". In: *Neural networks : the official journal of the International Neural Network Society* 43 (2013), pp. 99—113. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2013.02.003. URL: https://doi.org/10.1016/j.neunet.2013.02.003.

[168] Wenyu Yang, Dakun Yang, and Yetian Fan. "A Proof of a Key Formula in the Error-Backpropagation Learning Algorithm for Multiple Spiking Neural Networks". In: *Advances in Neural Networks – ISNN 2014*. Ed. by Zhigang Zeng, Yangmin Li, and Irwin King. Cham: Springer International Publishing, 2014, pp. 19–26. ISBN: 978-3-319-12436-0.

[169] Gioele Zardini et al. "A Compositional Sheaf-Theoretic Framework for Event-Based Systems". In: *arXiv preprint arXiv:2005.04715* (2020).

[170] Matthew D. Zeiler. "ADADELTA: An Adaptive Learning Rate Method". In: *CoRR* abs/1212.5701 (2012). arXiv: 1212.5701. URL: http://arxiv.org/abs/1212.5701.

[171] Friedemann Zenke and Surya Ganguli. "Superspike: Supervised learning in multilayer spiking neural networks". In: *Neural computation* 30.6 (2018), pp. 1514–1541.

[172] Friedemann Zenke and Tim P. Vogels. "The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks". In: *bioRxiv* (2020). DOI: 10.1101/2020.06.29.176925. eprint: https://www.biorxiv.org/content/early/2020/06/29/2020.06.29.176925.full.pdf. URL: https://www.biorxiv.org/content/early/2020/06/29/2020.06.29.176925.

[173] Hong Zhang, Emil M Constantinescu, and Barry F Smith. "PETSc TSAdjoint: a discrete adjoint ODE solver for first-order and second-order sensitivity analysis". In: *arXiv preprint arXiv:1912.07696* (2019).

[174] Hong Zhang et al. "Discrete adjoint sensitivity analysis of hybrid dynamical systems with switching". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 64.5 (2017), pp. 1247–1259.

Appendix A

Supplementary Material: Adjoint Equations of Point Neurons

Listing A.1: Convolutional architecture used in the Fashin-MNIST and MNIST task. Input dimensionality of one batch of examples is $T \times B \times 1 \times 28 \times 28$, with T the sequence length and B the batch size. Here Sequential, Conv2d, Flatten, MaxPool2d are standard PyTorch modules and LILayer, LIFFeedForwardLayer implement either the adjoint method or a surrogate gradient method for Leaky Integrators and feedforward LIF neurons respectively. Lift is applies a module pointwise accross time.

```
Sequential([
    Lift(Conv2d(1, 20, 5, 1)),
    LIFFeedForwardLayer(),
    Lift(MaxPool2d(2,2)),
    Lift(Conv2d(20, 50, 5, 1)),
    LIFFeedForwardLayer(),
        Lift(MaxPool2d(2,2)),
    Lift(Flatten()),
    Lift(Linear(800, 500)),
    LIFFeedForwardLayer(),
    LILayer(500, 10),
])
```

Listing A.2: Convolutional architecture used for the CIFAR-10 task.

```
Sequential([
    Lift(Conv2d(6, 32, 5, 1)),
    LIFFeedForwardLayer(),
    Lift(MaxPool2d(2,2)),
    Lift(Conv2d(32, 64, 5, 1)),
    LIFFeedForwardLayer(),
    Lift(MaxPool2d(2,2)),
```

```
    Lift(Flatten()),
    Lift(Linear(1600, 1024)),
    LIFFeedForwardLayer(),
    LILayer(1024, 10),
])
```

Symbol	Description	Value
τ_{mem}	Membrane Time Constant	20 ms
τ_{syn}	Synaptic Time Constant	5 ms
ϑ	Threshold	1
	Input Size	5
	Hidden Size	200
	Output Size	3
t_{bias}	Bias Time	20 ms
t_{min}	Minimum Time	10 ms
t_{max}	Maximum Time	40 ms
	Hidden Weights Mean	2
	Hidden Weights Standard Deviation	1
	Output Weights Mean	0.4
	Output Weights Standard Deviation	0.4
	Minibatch Size	200
	Optimizer	Adam
β_1	Adam Parameter	0.9
β_2	Adam Parameter	0.999
ϵ	Adam Parameter	1×10^{-8}
η	Learning Rate	1×10^{-3}
	Allowed Ratio of Missing Spikes in Hidden Layer	0.15
	Allowed Ratio of Missing Spikes in Output Layer	0
Δw_{bump}	Weight Bump Value	1×10^{-4}
α	Regularization Factor	1×10^{-2}
ξ	First Time Constant Factor	0.4
β	Second Time Constant Factor	2

Table A.1: Simulation parameters used for the Ying-Yang time to first spike experiment, table adapted from [165].