1 Preliminaries

The authors use a nice consequent notation. Aside from the usual derivatives and partial derivatives, for discontinuous variables, they adopt the convention that x^- is the value before the jump and x^+ the value after. One can formally define this as $x^- = \lim_{n\to\infty} x(t^* - \frac{1}{n})$ and $x^+ = \lim_{n\to\infty} x(t^* + \frac{1}{n})$, if the jump is at t^*.

Throughout the derivation, things remain contained within these sensible notations without reference to non-existing derivatives at the jump.

Throughout this manuscript, vectors do not have a separate notation, so it is up to the reader to remember, which symbols are vector-valued. The $\cdot$ is the normal scalar product on $\mathbb{R}^n$, i.e. $y \cdot z = \sum_{i=1}^n y_i z_i$.

I am not 100% convinced that the use of normal and partial derivative symbols is always following the most consequent line - we will discuss when we see issues.

The original text is in blue but I have at times corrected a typo or added a detail (very minor changes). The black text is commentary from me about what I think is going on.

1.1 Overall approach

There is a couple of main ideas that make the body of the method.

1. The adjoint method itself. For this Wunderlich and Pehle refer to
   63. Pontryagin, L. S. Mathematical Theory of Optimal Processes (Routledge,1962).
   64. Bradley, A. M. PDE-constrained optimization and the adjoint method (2019).
   I.e. we are looking at gradient descent in constrained optimisation problems. We will see Lagrange multipliers here but I think it is probably a distraction to try to find an intuition what the Lagrange multipliers λ "mean". In this work, the goal is to regroup the calculations for the gradient of the loss function, which involves a huge number of forward derivatives $\frac{\partial x}{\partial p_i}$ (in the general formulation of a dynamical system $x(t)$) into a less computationally expensive backward pass of "adjoint variables". The trick is to indtroduce the Lagrange multipliers and then reformat everything so that they replace the need for tracking those many derivatives.

2. To do the adjoint method for a network of LIF neurons, one needs to work around the discontinuities of the spikes and look at how terms in the loss that depend on the spike time can be handled. The former is done by careful consideration of the jumps at spike times (and breaking everything up into differentiable periods and jump times). The latter involves using the implicit function theorem.

3. Finally, the previous point often takes the form of taking a jump condition, using that the jump occurs at t_k^{post}, which by the implicit function theorem is a differentiable function of the weights, and deriving relationships for $\frac{\partial \cdot}{\partial w_{ji}}$ that have the form of a direct partial derivative with respect to w_{ji} plus a term that comes about through the dependence on t_k^{post} (see for instance (36)).

2 Adjoint method

We apply the adjoint method to a continuous, first order system of ordinary differential equations and refer the reader to [63,64] for a more general setting. Consider an N-dimensional dynamical system $x : t \to x(t) \in \mathbb{R}^N$ with parameters $p \in \mathbb{R}^P$ defined by the system of implicit first order ordinary differential equations

$$\dot{x} - F(x,p) = 0 \tag{15}$$

and constant initial conditions $G(x(0)) = 0$ where F, G are smooth vector-valued functions.

For what follows it might be useful to think of this alternatively in terms of the implicit function theorem where the N-dimensional dynamical system might be defined as

$$\mathcal{F}(\dot{x}, x, p) = 0 \tag{1}$$

with $\mathcal{F}(\dot{x}, x, p) = \dot{x} - F(x,p)$. More about this when we discuss adding $\lambda \cdot \mathcal{F}$ to the loss function $\mathcal{L}$.

We are interested in computing the gradient of a loss that is the integral of a smooth function l over the trajectory of x,

$$\mathcal{L} = \int_0^T l(x,t)dt \tag{16}$$

We have

$$\frac{d\mathcal{L}}{dp_i} = \int_0^T \frac{\partial l}{\partial x} \cdot \frac{\partial x}{\partial p_i} dt, \tag{17}$$

where $\cdot$ is the dot product ...

To unpack this, first, differentiation with respect to p_i can be swapped with the integration:

$$\frac{d}{dp_i} \int_0^T l(x,t)dt = \int_0^T \frac{d}{dp_i} l(x,t)dt \tag{2}$$

because the only dependency on p_i is through x, which depends on p_i through the dynamics (15). In other words the integration limits 0 and T do not depend on p_i.

Then, it is chain rule,

$$\frac{d}{dp_i} l(x,t) = \sum_k \frac{\partial l}{\partial x_k} \frac{\partial x_k}{\partial p_i} \tag{3}$$

which can then be written using the scalar product in vector form,

$$\frac{\partial l}{\partial x} \cdot \frac{\partial x}{\partial p_i} \tag{4}$$

... and the dynamics of the partial derivatives $\frac{\partial x}{\partial p_i}$ are given by applying Gronwalls theorem [61],

$$\frac{d}{dt} \frac{\partial x}{\partial p_i} = \frac{\partial F}{\partial x} \frac{\partial x}{\partial p_i} + \frac{\partial F}{\partial p_i}. \tag{18}$$

I believe Gronwall's theorem boils down to that under sufficient smoothness assumptions, one can swap derivatives if taking the partial derivatives of the dynamics equations (15),

$$\dot{x} = F(x,p) \tag{5}$$

$$\Rightarrow \frac{\partial}{\partial p_i} \frac{dx}{dt} = \frac{\partial}{\partial p_i} F(x,p) \tag{6}$$

$$\Rightarrow \frac{d}{dt} \frac{\partial x}{\partial p_i} = \frac{\partial}{\partial p_i} F(x,p) \tag{7}$$

and then it is chain rule,

$$\frac{\partial}{\partial p_i} F(x,p) = \frac{\partial F}{\partial x} \cdot \frac{\partial x}{\partial p_i} + \frac{\partial F}{\partial p_i} \tag{8}$$

where $\cdot$ is the scalar product (missing in the original text).

Computing $x(t)$ along with $\frac{\partial x}{\partial p_i}(t)$ using Eqs. (15) and (18) allows us to calculate the gradient in Eq. (17) in a single forward pass. However, this procedure can incur prohibitive computational cost. When considering a recurrent neural network with N neurons and $P = N^2$ synaptic weights, computing $\frac{\partial x}{\partial p_i}(t)$ for all parameters requires storing and integrating $PN = N^3$ partial derivatives.

Note that (18) is not needed for the adjoint method below and is only there to make the point that one can in principle do feed-forward gradient calculations.

The adjoint method allows us to avoid computing PN partial derivatives in the forward pass by instead computing N adjoint variables $\lambda(t)$ in an additional backward pass. We add a Lagrange multiplier $\lambda : t \to \lambda(t) \in \mathbb{R}^N$ that constrains the system dynamics as given in Eq. (15),

$$\mathcal{L} = \int_0^T \left[l(x,t) + \lambda \cdot (\dot{x} - F(x,p)) \right] dt \tag{19}$$

Along trajectories where Eq. (15) holds, λ can be chosen arbitrarily without changing $\mathcal{L}$ or its derivative.

This is a key step and I find it difficult to develop the right intuition. I think the motivation for adding the langrange multiplier is the eventual result of the reformulated derivative of the cost function (24). I could not come up with a different motivation or explanation why this is needed or what it is for.

To unpack the last statement a little, one can add the Langrange multiplier without changing $\mathcal{L}$ because $\mathcal{L}$ is meant to only be calculated for valid trajectories and for those $\mathcal{F} = \dot{x} - F(x,p) \equiv 0$ by definition (see (15)).

For the derivative with respect to parameters $\frac{\partial}{\partial p_i}(\dot{x} - F(x,p))$ it may seem less clear but if we think of the trajectories of the dynamical system for two nearby values of p_i, then we know that for both $\mathcal{F} \equiv 0$. So, therefore, there is no change of $\mathcal{F}$ with respect to p_i, and hence the derivative of $\mathcal{F}$ with respect to p_i is also 0.

We get

$$\frac{d\mathcal{L}}{dp_i} = \int_0^T \left[\frac{\partial l}{\partial x} \cdot \frac{\partial x}{\partial p_i} + \lambda \cdot \left(\frac{d}{dt}\frac{\partial x}{\partial p_i} - \frac{\partial F}{\partial x} \cdot \frac{\partial x}{\partial p_i} - \frac{\partial F}{\partial p_i} \right) \right] dt \tag{20}$$

Here, it is again the same argument as above for pulling the derivative into the integral. Then, the term $\frac{\partial l}{\partial x} \cdot \frac{\partial x}{\partial p_i}$ is as in (17) and the term $\lambda \cdot \left(\frac{d}{dt}\frac{\partial x}{\partial p_i} - \frac{\partial F}{\partial x} \cdot \frac{\partial x}{\partial p_i} - \frac{\partial F}{\partial p_i} \right)$ uses the same arguments as previously in (18) (see above).

Using partial integration, we have

$$\int_0^T \lambda \cdot \frac{d}{dt}\frac{\partial x}{\partial p_i} dt = -\int_0^T \dot{\lambda} \cdot \frac{\partial x}{\partial p_i} dt + \left[\lambda \cdot \frac{\partial x}{\partial p_i} \right]_0^T. \tag{21}$$

Just to be sure, partial integration is

$$\int u(t)\dot{v}(t)dt = [u(t)v(t)] - \int \dot{u}(t)v(t)dt \tag{9}$$

with $u(t) = \lambda(t)$, $v(t) = \frac{\partial x}{\partial p_i}$ this yields (21).

By setting $\lambda(T) = 0$, the boundary term vanishes because we chose parameter independent initial conditions ($\frac{\partial x}{\partial p_i} = 0$). The gradient becomes

$$\frac{d\mathcal{L}}{dp_i} = \int_0^T \left[\left(\frac{\partial l}{\partial x} - \dot{\lambda} - \frac{\partial F}{\partial x}\lambda \right) \cdot \frac{\partial x}{\partial p_i} - \lambda \cdot \frac{\partial F}{\partial p_i} \right] dt. \tag{10}$$

I think this reveals the overall design of the method. When starting out with (17) we had to track the numerous and hence inconvenient $\frac{\partial x}{\partial p_i}$. To get rid of that, we use the Langrange multiplier and partial integration so that $\frac{\partial x}{\partial p_i}$ is prefaced with a factor that we can make identically 0 along the whole trajectory with an appropriate choice of $\lambda(t)$.

By choosing λ to fulfill the adjoint differential equation

$$\dot{\lambda} = \frac{\partial l}{\partial x} - \frac{\partial F}{\partial x}\lambda \tag{23}$$

we are left with

$$\frac{d\mathcal{L}}{dp_i} = -\int_0^T \lambda \cdot \frac{\partial F}{\partial p_i} dt. \tag{24}$$

The gradient can therefore be computed using Eq. (24), where the adjoint state variable λ is computed from $t = T$ to $t = 0$ as the solution of the adjoint differential equation Eq. (23) with initial condition $\lambda(T) = 0$. This corresponds to backpropagation through time (BPTT) in discrete time artificial neural networks.

Mostly straightforward except the time inversion, maybe. My understanding is that this is necessary only because we know the “boundary condition” at T rather than at 0: $\lambda(T) = 0$. This stems from the need to get rid of the boundary values during the partial integration. To ensure this boundary condition, it is easiest to make it the initial condition for a backwards integration of the λ dynamics (23).

One way of summarising the method is to say λ and its dynamics backward in time cleverly combine what normally the many $\frac{\partial x}{\partial p_i}$ would have contributed to the gradient of the loss function.

3 Gradient of the loss function of an SNN

We apply the adjoint method (see previous methods subsection) to the case of a spiking neural network (i.e., a hybrid, discontinuous system with parameter dependent state transitions). The following derivation is specific to the model given in Table 1. A fully general treatment of (adjoint) sensitivity analysis in hybrid systems can be found in [8] or [10].

The differential equations defining the free dynamics in implicit form are

$$f_V \equiv \tau_{\mathrm{mem}} \dot{V} + V - I = 0, \tag{25a}$$
$$f_I \equiv \tau_{\mathrm{syn}} \dot{I} + I = 0, \tag{25b}$$

where f_V, f_I are again vectors of size N.

These are standard equations for N leaky integrate and fire (LIF) neurons and current-based synapses with exponential decay. The corresponding transitions during a spike are described in the earlier table 1, when $V_n - \vartheta = 0$ and $\dot{V}_n \neq 0$,

$$V_n^+ = 0 \tag{11}$$
$$I^+ = I^- + W \cdot e_n \tag{12}$$

where e_n is the unit vector with a 1 in dimension n and 0 otherwise. I.e., the voltage reset is to 0 and post-synaptic input currents are incremented by the sum of incoming weights from the spiking neuron.

We now split up the loss integral in Eq. (1) at the spike times t^{post} and use vectors of Lagrange multipliers λ_V, λ_I that fix the system dynamics f_V, f_I between transitions.

$$\frac{d\mathcal{L}}{dw_{ji}} = \frac{d}{dw_{ji}} \left[l_p(t^{\mathrm{post}}) + \sum_{k=0}^{N_{\mathrm{post}}} \int_{t_k^{\mathrm{post}}}^{t_{k+1}^{\mathrm{post}}} [l_V(V,t) + \lambda_V \cdot f_V + \lambda_I \cdot f_I] dt \right], \tag{26}$$

where we set $t_0^{\mathrm{post}} = 0$ and $t_{N_{\mathrm{post}}+1}^{\mathrm{post}} = T$ and $x \cdot y$ is the dot product of two vectors x, y. Note that because f_V, f_I vanish along all considered trajectories, λV and λI can be chosen arbitrarily without changing $\mathcal{L}$ or its derivative.

For reference, Eq. (1) was:

$$\mathcal{L} = l_p(t^{\mathrm{post}}) + \int_0^T l_V(V(t),t) dt \tag{1}$$

This is essentially the same approach as in the previous section except that there are now two dynamics equations. The argument, why the Lagrange multipliers can be added, are, however, exactly the same. Another two notes would be (1) the l_p term depends on all post-synaptic spike times, i.e. t^{post} without lower index is the vector of all post-synaptic spike times, and (2) the splitting of the integral for now does nothing but anticipates that later jumps occur in some variables at the spike times and splitting the integral into bits that are unproblematic inside helps dealing with these discrete events.

Using Eq. (25) we have, as per Gronwalls theorem [61],

$$\frac{\partial f_V}{\partial w_{ji}} = \tau_{\mathrm{mem}} \frac{d}{dt} \frac{\partial V}{\partial w_{ji}} + \frac{\partial V}{\partial w_{ji}} - \frac{\partial I}{\partial w_{ji}}, \tag{27a}$$
$$\frac{\partial f_I}{\partial w_{ji}} = \tau_{\mathrm{syn}} \frac{d}{dt} \frac{\partial I}{\partial w_{ji}} + \frac{\partial I}{\partial w_{ji}}, \tag{27b}$$

where we have used the fact that the derivatives commute, $\frac{\partial}{\partial w_{ji}} \frac{d}{dt} = \frac{d}{dt} \frac{\partial}{\partial w_{ji}}$ (the weights are fixed and have no time dependence).

I think the commuting of the derivatives is the essence of Gronwall's theorem but it is commendable that they have considered that this only works if w_{ji} do not depend on t. Other than the swapping of derivatives nothing special is happening here.

The gradient then becomes, by application of the Leibniz integral rule,

$$\begin{aligned} \frac{d\mathcal{L}}{dw_{ji}} = \sum_{k=0}^{N_{\mathrm{post}}} \Bigg[\int_{t_k^{\mathrm{post}}}^{t_{k+1}^{\mathrm{post}}} \left[\frac{\partial l_V}{\partial V} \cdot \frac{\partial V}{\partial w_{ji}} + \lambda_V \cdot \left(\tau_{\mathrm{mem}} \frac{d}{dt} \frac{\partial V}{\partial w_{ji}} + \frac{\partial V}{\partial w_{ji}} - \frac{\partial I}{\partial w_{ji}} \right) + \lambda_I \cdot \left(\tau_{\mathrm{syn}} \frac{d}{dt} \frac{\partial I}{\partial w_{ji}} + \frac{\partial I}{\partial w_{ji}} \right) \right] dt \\ + \frac{\partial l_p}{\partial t_k^{\mathrm{post}}} \frac{dt_k^{\mathrm{post}}}{dw_{ji}} + l_{V,k+1}^- \frac{dt_{k+1}^{\mathrm{post}}}{dw_{ji}} - l_{V,k}^+ \frac{dt_k^{\mathrm{post}}}{dw_{ji}} \Bigg], \end{aligned} \tag{28}$$

where $l^{\pm}_{V,k}$ is the voltage-dependent loss evaluated before (-) or after (+) the transition and we have used that $f_V = f_I = 0$ along all considered trajectories.

There is a lot to unpack here. The first thing to note is that the integral boundaries t^{post}_k and t^{post}_{k+1} depend on w_{ji} and hence the invocation of the Leibniz integral rule. It is an interesting case where we have something of the type

$$\frac{d}{dx}\int_{f(x)}^{g(x)} h(x,t)dt = \frac{d}{dx}\left(H(x,g(x)) - H(x,f(x))\right) \tag{13}$$

$$= \frac{\partial H}{\partial g}\frac{dg}{dx}\Big|_{x,g(x)} + \frac{\partial H}{\partial x}\Big|_{x,g(x)} - \frac{\partial H}{\partial f}\frac{df}{dx}\Big|_{x,f(x)} - \frac{\partial H}{\partial x}\Big|_{x,f(x)} \tag{14}$$

$$= h(x,g(x))\frac{dg(x)}{dx} - h(x,f(x))\frac{df(x)}{dx} + \int_{f(x)}^{g(x)}\frac{\partial h}{\partial x}dt \tag{15}$$

So, the first row of (28) is a straight-forward derivative of the integrand with application of the chain rule, as well as using (27a) and (27b). That leaves the derivative of the $l_p(t^{\text{post}})$ term and the terms caused by the w_{ji}-dependent integration bounds. The former is a straightforward application of the chain rule giving the term $\frac{\partial l_p}{\partial t^{\text{post}}_k}\frac{dt^{\text{post}}_k}{dw_{ji}}$. For the latter we get

$$\left(l^-_{V,k+1} + \lambda_V\cdot f_V(V^-_{k+1}) + \lambda_I\cdot f_I(I^-_{k+1})\right)\frac{dt^{\text{post}}_{k+1}}{dw_{ji}} - \left(l^+_{V,k} + f_V(V^+_k) + \lambda_I\cdot f_I(I^+_k)\right)\frac{dt^{\text{post}}_k}{dw_{ji}}. \tag{16}$$

However, all the terms with containing f_V and f_I are zero.

Using partial integration, we have

$$\int_{t^{\text{post}}_k}^{t^{\text{post}}_{k+1}} \lambda_V\cdot\frac{d}{dt}\frac{\partial V}{\partial w_{ji}}dt = -\int_{t^{\text{post}}_k}^{t^{\text{post}}_{k+1}}\dot{\lambda}_V\cdot\frac{\partial V}{\partial w_{ji}}dt + \left[\lambda_V\cdot\frac{\partial V}{\partial w_{ji}}\right]_{t^{\text{post}}_k}^{t^{\text{post}}_{k+1}}, \tag{29}$$

$$\int_{t^{\text{post}}_k}^{t^{\text{post}}_{k+1}} \lambda_I\cdot\frac{d}{dt}\frac{\partial I}{\partial w_{ji}}dt = -\int_{t^{\text{post}}_k}^{t^{\text{post}}_{k+1}}\dot{\lambda}_I\cdot\frac{\partial I}{\partial w_{ji}}dt + \left[\lambda_I\cdot\frac{\partial I}{\partial w_{ji}}\right]_{t^{\text{post}}_k}^{t^{\text{post}}_{k+1}}. \tag{30}$$

Very similar as in the more general derivation of the adjoint method above except that here we cannot get rid of all the integrated terms at t^{post}_k and t^{post}_{k+1} arguing that either the λ's or derivatives are 0.

Collecting terms in $\frac{\partial V}{\partial w_{ji}}$, $\frac{\partial I}{\partial w_{ji}}$, we have

$$\frac{d\mathcal{L}}{dw_{ji}} = \sum_{k=0}^{N_{\text{post}}}\left[\int_{t^{\text{post}}_k}^{t^{\text{post}}_{k+1}}\left[\left(\frac{\partial l_V}{\partial V} - \tau_{\text{mem}}\dot{\lambda}_V + \lambda_V\right)\cdot\frac{\partial V}{\partial w_{ji}} + \left(-\tau_{\text{syn}}\dot{\lambda}_I + \lambda_I - \lambda_V\right)\cdot\frac{\partial I}{\partial w_{ji}}\right]dt\right.$$
$$\left. + \frac{\partial l_p}{\partial t^{\text{post}}_k}\frac{dt^{\text{post}}_k}{dw_{ji}} + \tau_{\text{mem}}\left[\lambda_V\cdot\frac{\partial V}{\partial w_{ji}}\right]_{t^{\text{post}}_k}^{t^{\text{post}}_{k+1}} + \tau_{\text{syn}}\left[\lambda_I\cdot\frac{\partial I}{\partial w_{ji}}\right]_{t^{\text{post}}_k}^{t^{\text{post}}_{k+1}} + l^-_{V,k+1}\frac{dt^{\text{post}}_{k+1}}{dw_{ji}} - l^+_{V,k}\frac{dt^{\text{post}}_k}{dw_{ji}}\right]. \tag{31}$$

This is just straightforward term collections.

Since the Lagrange multipliers $\lambda_V(t)$, $\lambda_I(t)$ can be chosen arbitrarily, this form allows us to set the dynamics of the adjoint variables between transitions. Since the integration of the adjoint variables is done from $t = T$ to $t = 0$ in practice (i.e., reverse in time), it is practical to transform the time derivative as $\frac{d}{dt} \to -\frac{d}{dt}$. Denoting the new time derivative by $'$, we have

$$\tau_{\text{mem}}\lambda'_V = -\lambda_V - \frac{\partial l_V}{\partial V}, \tag{32a}$$

$$\tau_{\text{syn}}\lambda'_I = -\lambda_I + \lambda_V. \tag{32b}$$

The integrand in Eq. (31) therefore vanishes along the trajectory and we are left with a sum over the transitions.

Exactly the same as previously but note how the signs have been flipped to do the backward derivatives.

Since the initial conditions of V and I are assumed to be parameter independent, we have $\frac{\partial V}{\partial w_{ji}} = \frac{\partial I}{\partial w_{ji}} = 0$ at $t = 0$. We set the initial condition for the adjoint variables to be $\lambda_V(T) = \lambda_I(T) = 0$ to eliminate the boundary term for $t = T$. We are, therefore, left with a sum over transitions ξ_k evaluated

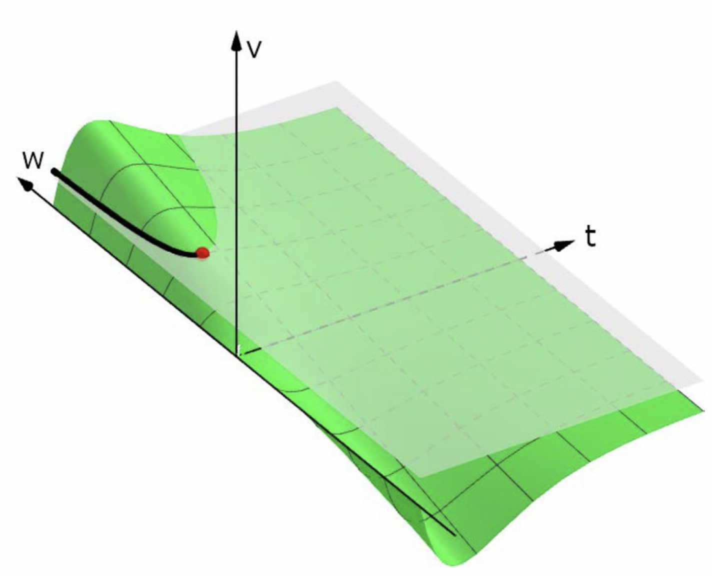

Figure 1: In this sketch, the relation $V(t,W) - \vartheta = 0$ defines an implicit function (black line along which $dV = 0$). The critical point where the gradient diverges is shown in red

at the transition times t_K^{post},

$$\frac{d\mathcal{L}}{dw_{ji}} = \sum_{k=1}^{N_{\text{post}}} \xi_k \tag{33}$$

with the definition

$$\begin{aligned}\xi_k \equiv{}& \frac{\partial l_p}{\partial t_k^{\text{post}}} \frac{dt_k^{\text{post}}}{dw_{ji}} + l_{V,k}^{-} \frac{dt_k^{\text{post}}}{dw_{ji}} - l_{V,k}^{+} \frac{dt_k^{\text{post}}}{dw_{ji}} \\ &+ \left[\tau_{\text{mem}} \left(\lambda_V^- \cdot \frac{\partial V^-}{\partial w_{ji}} - \lambda_V^+ \cdot \frac{\partial V^+}{\partial w_{ji}} \right) + \tau_{\text{syn}} \left(\lambda_I^- \cdot \frac{\partial I^-}{\partial w_{ji}} - \lambda_I^+ \cdot \frac{\partial I^+}{\partial w_{ji}} \right) \right] \Bigg|_{t_k^{\text{post}}}. \end{aligned} \tag{34}$$

There is not much new here but they have used their zero boundary conditions to niftily regroup the terms so that they all now relate to t_k^{post}, i.e. combining the right term of previous step k with the left term of step $k+1$ gives the new step $k+1$.

We proceed by deriving the relationship between the adjoint variables before and after each transition. Since the computation of the adjoint variables happens in reverse time in practice, we provide λ_- in terms of λ_+.

In other words, the adjoint method trick only has defined the necessary λ dynamics for in between spikes. We can still choose discontinuos jumps for the λ's at the spike times if we want. The strategy now is to try to get rid of the terms $\frac{\partial V^\pm}{\partial w_{ji}}$ and $\frac{\partial I^\pm}{\partial w_{ji}}$ as these are not easily available/ annoyingly many (this is the same strategy as before but now for the discrete bits).

To get rid of these terms, first relationships between + and − versions are derived, reducing the number of terms by half when substituting the relationships into the gradient equation (34). These relationships are all based on the original jump conditions in the original hybrid system dynamics (see (11), (12)). Then λ jumps are defined to eliminate the remaining ones (see at the end).

Consider a spike caused by the nth neuron, with all other neurons $m \neq n$ remaining silent. We start by first deriving the relationships between $\frac{\partial V^+}{\partial w_{ji}}$, $\frac{\partial V^-}{\partial w_{ji}}$ and $\frac{\partial I^+}{\partial w_{ji}}$, $\frac{\partial I^-}{\partial w_{ji}}$.

3.1 Membrane potential transition

By considering the relations between V^+ , V^- and $\dot{V}^+$, $\dot{V}^-$, we can derive the relation between $\frac{\partial V^+}{\partial w_{ji}}$ and $\frac{\partial V^-}{\partial w_{ji}}$ at each spike. Each spike at t^{post} is triggered by a neuron's membrane potential crossing the threshold. We therefore have, at t^{post},

$$(V^-)_n - \vartheta = 0. \tag{35}$$

This relation defines t^{post} as a differentiable function of w_{ji} via the implicit function theorem (illustrated in Fig. 5, see also [65]), under the condition that $(\dot{V})_n \neq 0$. Differentiation of this relation yields

$$\left(\frac{V^-}{d_{w_{ji}}}\right)_n + \left(\dot{V}^-\right)_n \frac{dt^{\text{post}}}{dw_{ji}} = 0. \tag{36}$$

The figure makes it quite intuitive that (35) defines a function $t^{\text{post}}(w)$ where (35) is true. The implicit function theorem gives the conditions on smoothness etc for which this implicitly defined function is differentiable. Generally, (35) might not be expicitly solvable for $t^{\text{post}}(w)$ but one can always take the derivative of (35) to find

$$\frac{\partial}{\partial w_{ji}}\left[(V^-(t^{\text{post}}, W))_n - \vartheta\right] \tag{17}$$

$$= \left(\frac{\partial V^-}{\partial w_{ji}}\right)_n + \left(\frac{\partial V^-}{\partial t^{\text{post}}}\right)_n \frac{dt^{\text{post}}}{dw_{ji}} = 0 \tag{18}$$

which leads to (36) as $\frac{\partial V^-}{\partial t^{\text{post}}} = \dot{V}^-$.

Since we only allow transitions for $(\dot{V})_n \neq 0$, we have

$$\frac{dt^{\text{post}}}{dw_{ji}} = -\frac{1}{\left(\dot{V}^-\right)_n}\left(\frac{\partial V^-}{\partial w_{ji}}\right)_n. \tag{37}$$

Note that corresponding relations were previously used to derive gradient-based learning rules for spiking neuron models [20 22,26,66]; in contrast to the suggestion in [20], Eq. (37) is not an approximation but rather an exact relation at all non-critical parameters and invalid at all critical parameters. Because the spiking neurons membrane potential is reset to zero, we have

$$\left(V^+\right)_n = 0. \tag{38}$$

This implies by differentiation

$$\left(\frac{\partial V^+}{\partial w_{ji}}\right)_n + (\dot{V}^+)_n \frac{dt^{\text{post}}}{dw_{ji}} = 0. \tag{39}$$

I find the steps for V^+ slightly less intuitive but I think the way to think about it is that while t^{post} is defined via the threshold condition on V_-, V^+ still is tied to t^{post} and hence depends on t^{post}. Maybe it is actually easier to think about this backwards in time. If integrated backwards, V^+ must reach 0 at t^{post}. That implies (39) with all minus signs ... giving the same equation.

Using Eq. (37), this allows us to relate the partial derivative after the spike to the partial derivative before the spike,

$$\left(\frac{\partial V^+}{\partial w_{ji}}\right)_n = \frac{\left(\dot{V}^+\right)_n}{\left(\dot{V}^-\right)_n}\left(\frac{\partial V^-}{\partial w_{ji}}\right)_n. \tag{40}$$

This derives directly from substituting (39) into (37).

Since we have $(V^+)_m = (V^-)_m$ for all other, non-spiking neurons $m \neq n$, it holds that

$$\left(\frac{\partial V^+}{\partial w_{ji}}\right)_m + \left(\dot{V}^+\right)_m \frac{dt^{\text{post}}}{dw_{ji}} = \left(\frac{\partial V^+}{\partial w_{ji}}\right)_m + \left(\dot{V}^-\right)_m \frac{dt^{\text{post}}}{dw_{ji}} \tag{41}$$

I suppose this just means that you can of course also ask what is the derivative with respect to w_{ji} of all the other $(V)_m$ at t^{post} ... just by asking this question at t^{post} makes $(V)_m$ a function of t^{post}.

Because the spiking neuron n causes the synaptic current of all neurons $m \neq n$ to jump by w_{mn} , we have

$$\tau_{\text{mem}}\left(\dot{V}^+\right)_m = \tau_{\text{mem}}\left(\dot{V}^-\right)_m + w_{mn} \tag{42}$$

and therefore get with Eq. (36)

$$\left(\frac{\partial V^+}{\partial w_{ji}}\right)_m = \left(\frac{\partial V^-}{\partial w_{ji}}\right)_m - \tau_{\text{mem}}^{-1} w_{mn} \frac{dt^{\text{post}}}{dw_i j} \tag{43}$$

This equality results from reformatting (42) into

$$\left(\dot{V}^-\right)_m - \left(\dot{V}^+\right)_m = -\tau_{\text{mem}}^{-1} w_{mn} \tag{19}$$

and inserting into (41) reformatted as

$$\left(\frac{\partial V^+}{\partial w_{ji}}\right)_m = \left(\frac{\partial V^+}{\partial w_{ji}}\right)_m + \left(\left(\dot{V}^-\right)_m - \left(\dot{V}^+\right)_m\right) \frac{dt^{\text{post}}}{dw_{ji}} \tag{20}$$

$$= \left(\frac{\partial V^-}{\partial w_{ji}}\right)_m + \frac{1}{\tau_{\text{mem}}(\dot{V}^-)_n} w_{mn} \left(\frac{\partial V^-}{\partial w_{ji}}\right)_n \tag{44}$$

This is based on (37) inserted into (43) ... not (36).

3.2 Synaptic current transition

The spiking neuron n causes the synaptic current of all neurons $m \neq n$ to jump by the corresponding weight w_{mn} . We therefore have

$$(I^+)_m = (I^-)_m + w_{mn}. \tag{45}$$

By differentiation, this relation implies the consistency equations for the partial derivatives $\frac{\partial I}{\partial w_{ji}}$ with respect to the considered weight wji,

Essentially, $I^\pm$ is also a function of t^{post} in the same way as the other variables as they are tagged to the spike time. So taking the same kind of derivative makes sense.

$$\left(\frac{\partial I^+}{\partial w_{ji}}\right)_m + \left(\dot{I}^+\right)_m \frac{dt^{\text{post}}}{dw_{ji}} = \left(\frac{\partial I^-}{\partial w_{ji}}\right)_m + \left(\dot{I}^-\right)_m \frac{dt^{\text{post}}}{dw_{ji}} + \delta_{in}\delta_{jm}, \tag{46}$$

where δ_{ji} is the Kronecker delta.

Here the δ's arise because if $i = n$ and $j = m$, then $\frac{\partial w_{mn}}{\partial w_{ji}} = 1$ and it's 0 in all other cases.

Because

$$\tau_{\text{syn}} \left(\dot{I}^+\right)_m = \tau_{syn} \left(\dot{I}^-\right)_m - w_{mn}, \tag{47}$$

This comes from combining (45) and the original current dynamics (25b):

$$\tau_{\text{syn}} \dot{I}^+ - \tau_{\text{syn}} \dot{I}^- = -(I^+ - I^-) \tag{21}$$

by (25b) and

$$(I^+)_m - (I^-)_m = w_{mn} \tag{22}$$

by (45).

we get with Eq. (36)

$$\left(\frac{\partial I^+}{\partial w_{ji}}\right)_m = \left(\frac{\partial I^-}{\partial w_{ji}}\right)_m + \tau_{\text{syn}^{-1}} w_{mn} \frac{dt^{\text{post}}}{dw_{ji}} + \delta_{in}\delta_{jm}, \tag{48}$$

This is based on (46) and (47), where in (46) one shifts the $\left(\dot{I}^+\right)_m \frac{dt^{\text{post}}}{dw_{ji}}$ term to the right and then replaces $\left(\dot{I}^-\right)_m - \left(\dot{I}^+\right)_m$ with $\tau_{\text{syn}^{-1}} w_{mn}$ based on (47).

$$= \left(\frac{\partial I^-}{\partial w_{ji}}\right)_m - \frac{1}{\tau_{syn}\left(\dot{V}^-\right)_n} w_{mn} \left(\frac{\partial V^-}{\partial w_{ji}}\right)_n + \delta_{in}\delta_{jm}, \tag{49}$$

This comes about by using (36) to replace $\frac{dt^{\text{post}}}{dw_{ji}}$.

With $(I^+)_n = (I^-)_n$ and $\left(\dot{I}^+\right)_n = \left(\dot{I}^-\right)_n$, we have

$$\left(\frac{\partial I^+}{\partial w_{ji}}\right)_n = \left(\frac{\partial I^-}{\partial w_{ji}}\right)_n . \tag{50}$$

For the spiking neuron n, there are no self-synapses and hence the synaptic current for this neuron does not change, and neither its derivative. The equation (50) then comes about by considering the derivative of $(I^+)_n = (I^-)_n$,

$$\left(\frac{\partial I^+}{\partial w_{ji}}\right)_n + \left(\dot{I}^+\right)_n \frac{dt^{\text{post}}}{dw_{ji}} = \left(\frac{\partial I^-}{\partial w_{ji}}\right)_n + \left(\dot{I}^-\right)_n \frac{dt^{\text{post}}}{dw_{ji}}, \tag{23}$$

which together with $\left(\dot{I}^+\right)_n = \left(\dot{I}^-\right)_n$ gives the result.

3.3 Putting it all together

Using the relations of the partial derivatives from Eqs. (37), (40), (44), (49) and (50) in the transition equation Eq. (34), we now derive relations between the adjoint variables. Collecting terms in the partial derivatives and writing the index of the spiking neuron for the kth spike as $n(k)$, we have

$$\begin{aligned}\xi_k = \Bigg[&\sum_{m \neq n(k)} \left[\tau_{\text{mem}} (\lambda_V^- - \lambda_V^+)_m \left(\frac{\partial V^-}{\partial w_{ji}}\right)_m + \tau_{\text{syn}} (\lambda_I^- - \lambda_I^+)_m \left(\frac{\partial I^-}{\partial w_{ji}}\right)_m - \tau_{\text{syn}} \delta_{in(k)} \delta_{jm} (\lambda_I^+)_m \right] \\ &+ \left(\frac{\partial V^-}{\partial w_{ji}}\right)_{n(k)} \left[\tau_{\text{mem}} \left(\lambda_V^- - \frac{(\dot{V}^+)_{n(k)}}{(\dot{V}^-)_{n(k)}} \lambda_V^+ \right)_{n(k)} + \frac{1}{(\dot{V}^-)_{n(k)}} \left(\sum_{m \neq n(k)} w_{mn(k)} (\lambda_I^+ - \lambda_V^+)_m - \frac{\partial l_p}{\partial t_k^{\text{post}}} + l_V^+ - l_V^- \right) \right] \\ &+ \tau_{\text{syn}} (\lambda_I^- - \lambda_I^+)_{n(k)} \left(\frac{\partial I^-}{\partial w_{ji}}\right)_{n(k)} \Bigg] \Bigg|_{t_k^{\text{post}}} . \end{aligned} \tag{51}$$

This is a grand collection and reorganisation of terms from (34) and equations as listed. It is important to keep in mind that in (34) there are still scalar products of vectors $\lambda_V^- \cdot \frac{\partial V^-}{\partial w_{ji}}$ and so on, that in (51) are broken up into their components and separately into the $n(k)$ component of the spiking neuron and the $m \neq n(k)$ components of the non-spiking neurons. This is because the spiking and non-spiking components have different relationships between $\left(\frac{\partial V^+}{\partial w_{ji}}\right)_x$ and the "minus version" $\left(\frac{\partial V^-}{\partial w_{ji}}\right)_x$. Note that there is a typo in the manuscript that I have corrected above: It's $w_{mn(k)}$, not $w_{n(k)m}$.

This form dictates the jumps of the adjoint variables for the spiking neuron n and all other, silent neurons m,

$$(\lambda_V^-)_n = \frac{\left(\dot{V}^+\right)_n}{\left(\dot{V}^-\right)_n} (\lambda_V^+)_n + \frac{1}{\tau_{\text{mem}} \left(\dot{V}^-\right)_n} \left[\sum_{m \neq n} w_{mn} (\lambda_V^+ - \lambda_I^+)_m + \frac{\partial l_P}{\partial t_k^{\text{post}}} + l_V^- - l_V^+ \right], \tag{52a}$$

$$(\lambda_V^-)_m = (\lambda_V^+)_m \tag{52b}$$

$$\lambda_I^- = \lambda_I^+ \tag{52c}$$

With these jumps, the gradient reduces to

$$\frac{d\mathcal{L}}{dw_{ji}} = -\tau_{\text{syn}} \sum_{k=1}^{N_{\text{post}}} \delta_{in(k)} (\lambda_I)_j \tag{53}$$

$$= -\tau_{\text{syn}} \sum_{t \in \text{spikes from i}} (\lambda_I)_j(t). \tag{54}$$

These choices are now made to remove all the cumbersome terms from (51). I have checked and the math works out that everything is removed except the term with the δ's.

3.4 Summary

The free adjoint dynamics between spikes are given by Eq. (32) while spikes cause jumps given by Eq. (52). The gradient for a given weight samples the post-synaptic neuron's λ_I when spikes are transmitted across the corresponding synapse [Eq. (53)]. Since we can identify, with $\left(\dot{V}^+\right)_n - \left(\dot{V}^-\right)_n = \tau_{\text{mem}}^{-1}\vartheta$,

$$\frac{\left(\dot{V}^+\right)_n}{\left(\dot{V}^-\right)_n} = \frac{\left(\dot{V}^+\right)_n - \left(\dot{V}^-\right)_n}{\left(\dot{V}^-\right)_n} + 1 = \frac{\vartheta}{\tau_{\text{mem}}\left(\dot{V}^-\right)_n} + 1 \tag{55}$$

the derived solution is equivalent to Eq. (2) and Table 2.

3.5 Fixed Input Spikes

If a given neuron j is subjected to a fixed pre-synaptic spike train across a synapse with weight $w_{j,\text{input}}$, the transition times are fixed and the adjoint variables do not experience jumps. The gradient simply samples the neurons λ_I at the times of spike arrival,

$$\frac{d\mathcal{L}}{dw_{j,\text{input}}} = -\tau_{\text{syn}} \sum_{t \in \text{input spikes}} (\lambda_I^+)_j(t). \tag{56}$$

This is confusing and I am not sure entirely right. Jumps in λ variables are caused by the integration boundaries of the piece-wise integral in the loss function, broken at spike times. The jumps come about if those spike times depend on the weight wrt which we are taking a partial derivative. In this case, the spikes of the receiving neuron j depend on $w_{j,\text{input}}$ and hence should experience jumps as normal when they spike. And their contribution to the gradient is as described. The input spikes themselves do not cause jumps – but they also don't really have a neuron model attached or any λ adjoint variables.

3.6 Coincident spikes

The derivation above assumes that only a single neuron of the recurrent network spikes at a given t_k^{post}. In general, coincident spikes may occur. If neurons a and b spike at the same time and the times of their respective threshold crossing vary independently as function of w_{ji} , the derivation above still holds, with both neurons λ_V experiencing a jump as in Eq. (52a).

I am not quite sure what it means for the two neurons' threshold crossings to vary independently as a function of w_{ji}. But presumably, even if this isn't always strictly true, it's probably so rare and inconsequential that we don't need to worry about it in practice.

4 Code availability

Code to reproduce the shown results will be made available at https://github.com/eventprop.

5 Summary

The method is hence given as in tables 1 and 2:

Table 1:

Free dynamics	Transition condition	Jumps at transition
$\tau_{\text{mem}}\frac{d}{dt}V = -V + I$	$(V)_n - \vartheta = 0,\ \left(\dot{V}\right)_n \neq 0$	$(V^+)_n = 0$
$\tau_{\text{syn}}\frac{d}{dt}I = -I$	for any n	$I^+ = I^- + We_n$

Table 2:

Free dynamics	Transition cond.	Jumps at transition
$\tau_{\text{mem}}\lambda_V' = -\lambda_V - \frac{\partial l_V}{\partial V}$	$t - t_k^{\text{post}} = 0$	$(\lambda_V^-)_{n(k)} = (\lambda_V^+)_{n(k)} + \frac{1}{\tau_{\text{mem}}(\dot{V}^-)_{n(k)}}\Big[\vartheta(\lambda_V^+)_{n(k)}$
$\tau_{\text{syn}}\lambda_I' = -\lambda_I + \lambda_V$	for any k	$+ \left(W^T(\lambda_V^+ - \lambda_I^+)\right)_{n(k)} + \frac{\partial l_p}{\partial t_k^{\text{post}}} + l_V^- - l_V^+\Big]$

The gradient is then calculated as

$$\frac{d\mathcal{L}}{dw_{ji}} = -\tau_{\text{syn}} \sum_{t \in \text{spikes of neuron } i} (\lambda_I^+)_j(t) \tag{24}$$

Similar for input spikes (see (56)).

5.1 Ying-Yang experiment

For the Ying-Yang experiment, Pehle et al. use the loss function:

$$\mathcal{L} = -\frac{1}{N_{\text{batch}}}\left[\sum_{i=1}^{N_{\text{batch}}} \log\left[\frac{\exp\left(-t^{\text{post}}_{i,l(i)}/\tau_0\right)}{\sum_{k=1}^{3}\exp\left(-t^{\text{post}}_{i,k}/\tau_0\right)}\right] - \alpha\left[\exp\left(t^{\text{post}}_{i,l(i)}/\tau_1\right) - 1\right]\right] \tag{25}$$

This is a loss function that is of the form $\sum l_p(t_k)$. For the backwards jumps, we need

$$\frac{\partial\mathcal{L}}{\partial t_{i,l(i)}} = -\frac{1}{N_{\text{batch}}}\left[\frac{\sum_{k=1}^{3}\exp\left(-t^{\text{post}}_{i,k}/\tau_0\right)}{\exp\left(-t^{\text{post}}_{i,l(i)}/\tau_0\right)}\left(\frac{-\frac{1}{\tau_0}\exp\left(-t^{\text{post}}_{i,l(i)}/\tau_0\right)}{\sum_{k=1}^{3}\exp\left(-t^{\text{post}}_{i,k}/\tau_0\right)}\right.\right. \tag{26}$$

$$\left.\left.-\frac{\exp\left(-t^{\text{post}}_{i,l(i)}/\tau_0\right)}{\left(\sum_{k=1}^{3}\exp\left(-t^{\text{post}}_{i,k}/\tau_0\right)\right)^2}\left(-\frac{1}{\tau_0}\exp\left(-t^{\text{post}}_{i,l(i)}/\tau_0\right)\right)\right) - \alpha\frac{1}{\tau_1}\exp\left(t^{\text{post}}_{i,l(i)}/\tau_1\right)\right] \tag{27}$$

$$= -\frac{1}{N_{\text{batch}}}\left[-\frac{1}{\tau_0} + \frac{1}{\tau_0}\frac{\exp\left(-t^{\text{post}}_{i,l(i)}/\tau_0\right)}{\sum_{k=1}^{3}\exp\left(-t^{\text{post}}_{i,k}/\tau_0\right)} - \frac{\alpha}{\tau_1}\exp\left(t^{\text{post}}_{i,l(i)}/\tau_1\right)\right] \tag{28}$$

$$= \frac{1}{N_{\text{batch}}}\left[\frac{1}{\tau_0}\left(1 - \frac{\exp\left(-t^{\text{post}}_{i,l(i)}/\tau_0\right)}{\sum_{k=1}^{3}\exp\left(-t^{\text{post}}_{i,k}/\tau_0\right)}\right) + \frac{\alpha}{\tau_1}\exp\left(t^{\text{post}}_{i,l(i)}/\tau_1\right)\right] \tag{29}$$

$$\frac{\partial\mathcal{L}}{\partial t_{i,j}} = -\frac{1}{N_{\text{batch}}}\left[\frac{\sum_{k=1}^{3}\exp\left(-t^{\text{post}}_{i,k}/\tau_0\right)}{\exp\left(-t^{\text{post}}_{i,l(i)}/\tau_0\right)}\left(-\frac{\exp\left(-t^{\text{post}}_{i,l(i)}/\tau_0\right)}{\left(\sum_{k=1}^{3}\exp\left(-t^{\text{post}}_{i,k}/\tau_0\right)\right)^2}\right)\left(-\frac{1}{\tau_0}\exp\left(-t^{\text{post}}_{i,j}/\tau_0\right)\right)\right] \tag{30}$$

$$= \frac{1}{N_{\text{batch}}}\left[-\frac{1}{\tau_0}\frac{\exp\left(-t^{\text{post}}_{i,j}/\tau_0\right)}{\sum_{k=1}^{3}\exp\left(-t^{\text{post}}_{i,k}/\tau_0\right)}\right] \tag{31}$$

We will use (29) and (31) to update λ_V at jump points of output neurons.

6 MNIST example

Here the authors define the cost function

$$\mathcal{L} = -\frac{1}{N_{\text{batch}}}\sum_{i=1}^{N_{\text{batch}}}\log\left[\frac{\exp\left(\max_t V_{l(i)}(t)\right)}{\sum_{k=1}^{10}\exp\left(\max_t V_k(t)\right)}\right] \tag{32}$$

where $V_k(t)$ is the voltage trace of the kth readout neuron, $l(i)$ is the index of the correct label for the ith sample and N_{batch} is the number of samples ina given batch.

They then argue: " Note that we can write the maximum voltage as $\max_t V_k(t) = \int V_k(t)\delta(t - t_{\max}dt$ with the time of the maximum $t_{\max}$ and the Dirac delta δ, allowing us to apply the chain rule to find the jump of λ_{V_k} (cf Table 2) at time $t_{\max}$ (terms containing the distributional derivative of δ are always zero)."

The first observation is that this does not conform to their general expression for L,

$$\mathcal{L} = l_p(t^{\text{post}}) + \int_0^T l_V(V(t), t)dt \tag{33}$$

so the normal formula do not apply. However, I believe the argument is that if one did the derivation for this different form of cost function, using the represnetation of $\max_t V_k(t)$ with the Dirac delta as explained above, one would arrive at a similar set of update equations where there si a jump in λ_{V_k} at $t_{max,k}$

This seems supported by the helpful comment in the earlier discussion: The loss l_V may depend on the voltage at a discrete time t_i using the Dirac delta, $l_V(V(t), t) = V(t)\delta(t_i t)$, causing a jump of λ_V of magnitude τ_{mem}^{-1} at time t_i. Note that in many practical scenarios as found in deep learning, the loss l_V

depends only on the state of a constant number of neurons, irrespective of network size. If l_V depends on the voltage of non-firing readout neurons, we have $l_V^+ = l_V$ and the corresponding term in the jump given in Table 2 vanishes.

***THIS IS NOT CORRECT *** SEE NEW ATTEMPT AT THE BOTTOM ***

In order to unpick the details of what's needed, let us define $M_k = \max_t V_k(t)$ such that

$$\mathcal{L} = -\frac{1}{N_{\text{batch}}} \sum_{i=1}^{N_{\text{batch}}} \log \left[\frac{\exp\left(M_{l(i)}\right)}{\sum_{k=1}^{10} \exp\left(M_k\right)} \right] \tag{34}$$

$$= -\frac{1}{N_{\text{batch}}} \sum_{i=1}^{N_{\text{batch}}} M_{l(i)} - \log \left(\sum_{k=1}^{10} \exp\left(M_k\right) \right) \tag{35}$$

and the partial derivatives with respect to V_r and $V_{l(i)}$ would be obtained by the chain rule, for $V_r(t)$ in presentation i and $r \neq l(i)$,

$$\frac{\partial}{\partial V_r(t)} \mathcal{L} = -\frac{1}{N_{\text{batch}}} \left(-\frac{\exp\left(M_r\right)}{\sum_{k=1}^{10} \exp\left(M_k\right)} \right) \frac{\partial}{\partial V_r} M_r \tag{36}$$

where

$$\frac{\partial}{\partial V_r(t)} M_r = \frac{\partial}{\partial V_r(t)} \int V_r(t') \delta(t' - t_{\text{max}}) dt' \tag{37}$$

$$= \int \delta(t' - t) \delta(t' - t_{\text{max}}) dt' \tag{38}$$

$$= \delta(t - t_{\text{max}}) \tag{39}$$

which, if inserted into the reverse differential equation for λ_V in Table 2, would lead to the described jump behaviour. In this case the jump would be

$$-\frac{1}{\tau_{\text{mem}}} \frac{1}{N_{\text{batch}}} \frac{\exp\left(M_r\right)}{\sum_{k=1}^{10} \exp\left(M_k\right)} \tag{40}$$

at time $t_{\text{max,r}}$ (the minus sign is from the λ_V equation in Table 2: $-\frac{\partial l_V}{\partial V}$).

For the voltage of the correct output neuron, $V_{l(i)}$ the derivative would be

$$\frac{\partial}{\partial V_{l(i)}(t)} \mathcal{L} = -\frac{1}{N_{\text{batch}}} \left(1 - \frac{\exp\left(M_{l(i)}\right)}{\sum_{k=1}^{10} \exp\left(M_k\right)} \right) \frac{\partial}{\partial V_{l(i)}} M_{l(i)} \tag{41}$$

and hence the jump,

$$\frac{1}{\tau_{\text{mem}}} \frac{1}{N_{\text{batch}}} \left(1 - \frac{\exp\left(M_{l(i)}\right)}{\sum_{k=1}^{10} \exp\left(M_k\right)} \right) \tag{42}$$

at time $t_{\text{max,r}}$ (including the minus sign from table 2, as above).

So, in essence, in the MNIST example with non-spiking output neurons, there are no spike times to jump at. Furthermore, l_V does not jump at other neurons' spike times, so there is no contribution $l_V^- - l_V^+$ to other neurons' jumps. However, in addition to their normal exponential decay λ_V equation, output neurons jump at their t_{max} by the amounts noted above. t_{max} can obviously easily be recorded in the forward pass.

The same technology can be applied to the spiking Heidelberg digits dataset. Though (as of 2022-02-15) not with much success.

7 Exact integration of forward and backward dynamics

Because of the simplicity of the dynamic equations (at least in the absence of the $\frac{\partial l_V}{\partial V}$ term) one can calculate the exact solution for teh λ dynamics over a timestep from t_0 to t.

We have

$$\tau_{\text{mem}} \lambda_V' = -\lambda_V \tag{43}$$

$$\tau_{\text{syn}} \lambda_I' = \lambda_V - \lambda_I. \tag{44}$$

For the λ_V eqaution we know by heart that this is simply exponential decay (or otherwise there is the usual derivation with $\frac{f'}{f}$ integrating to $\log f$):

$$\lambda_V(t) = \lambda_V(t_0) \exp\left(-\frac{t-t_0}{\tau_{\text{mem}}}\right) \tag{45}$$

A quick check shows:

$$\begin{aligned} \tau_{\text{mem}}\lambda_V' &= \tau_{\text{mem}}\lambda_v(t_0)(-1)\exp\left(-\frac{t-t_0}{\tau_{\text{mem}}}\right) && (46)\\ &= -\lambda_V(t_0)\exp\left(-\frac{t-t_0}{\tau_{\text{mem}}}\right) = -\lambda_V && (47) \end{aligned}$$

and

$$\lambda_V(t_0) = \lambda_V(t_0) \tag{48}$$

as it should be. Then, the homogeneous equation for λ_I is also simple exponential decay, so that

$$\lambda_I(t) = c(t)\exp\left(-\frac{t-t_0}{\tau_{\text{syn}}}\right) \tag{49}$$

Therefore,

$$\begin{aligned} & \tau_{\text{syn}}\lambda_I'(t) = \tau_{\text{syn}}c'(t)\exp\left(-\frac{t-t_0}{\tau_{\text{syn}}}\right) - c(t)\exp\left(-\frac{t-t_0}{\tau_{\text{syn}}}\right) \overset{!}{=} \lambda_V(t) - \lambda_I(t) && (50)\\ \Rightarrow\quad & \tau_{\text{syn}}c'(t)\exp\left(-\frac{t-t_0}{\tau_{\text{syn}}}\right) = \lambda_V(t) = \lambda_V(t_0)\exp\left(-\frac{t-t_0}{\tau_{\text{mem}}}\right) && (51)\\ \Rightarrow\quad & c'(t) = \frac{\lambda_V(t_0)}{\tau_{\text{syn}}}\exp\left(-\left(\frac{1}{\tau_{\text{mem}}}-\frac{1}{\tau_{\text{syn}}}\right)(t-t_0)\right) && (52)\\ \Rightarrow\quad & c(t) = -\frac{\lambda_V(t_0)}{\tau_{\text{syn}}}\frac{1}{\frac{1}{\tau_{\text{mem}}}-\frac{1}{\tau_{\text{syn}}}}\exp\left(-\left(\frac{1}{\tau_{\text{mem}}}-\frac{1}{\tau_{\text{syn}}}\right)(t-t_0)\right) + c && (53)\\ \Rightarrow\quad & \lambda_I(t) = -\frac{\lambda_V(t_0)}{\frac{\tau_{\text{syn}}}{\tau_{\text{mem}}}-1}\exp\left(-\frac{1}{\tau_{\text{mem}}}(t-t_0)\right) + c\exp\left(-\frac{t-t_0}{\tau_{\text{syn}}}\right) && (54) \end{aligned}$$

Taking the initial condition into account, we have

$$\begin{aligned} \lambda_I(t_0) &= -\frac{\lambda_V(t_0)}{\frac{\tau_{\text{syn}}}{\tau_{\text{mem}}}-1} + c && (55)\\ \Rightarrow\quad c &= \lambda_I(t_0) + \frac{\lambda_V(t_0)}{\frac{\tau_{\text{syn}}}{\tau_{\text{mem}}}-1} && (56) \end{aligned}$$

Hence we get altogether

$$\begin{aligned} \lambda_I(t) &= -\frac{\lambda_V(t_0)}{\frac{\tau_{\text{syn}}}{\tau_{\text{mem}}}-1}\exp\left(-\frac{t-t_0}{\tau_{\text{mem}}}\right) + \lambda_I(t_0)\exp\left(-\frac{t-t_0}{\tau_{\text{syn}}}\right) + \frac{\lambda_V(t_0)}{\frac{\tau_{\text{syn}}}{\tau_{\text{mem}}}-1}\exp\left(-\frac{t-t_0}{\tau_{\text{syn}}}\right) && (57)\\ &= \frac{\lambda_V(t_0)}{\frac{\tau_{\text{syn}}}{\tau_{\text{mem}}}-1}\left(\exp\left(-\frac{t-t_0}{\tau_{\text{syn}}}\right) - \exp\left(-\frac{t-t_0}{\tau_{\text{mem}}}\right)\right) + \lambda_I(t_0)\exp\left(-\frac{t-t_0}{\tau_{\text{syn}}}\right) && (58) \end{aligned}$$

As the forward equations are identical for V (i.e. like the λ_I equation), the equivalent equation can be used for V in the forward pass.

8 Insight of the day 2022-02-16

Do *not* taper the learning rate aggressively (`p["ETA_DECAY"]= 0.95`) while at the same time using an Adam optimizer. Should have known this. Removing the decay, with not too unusual settings otherwise, now reached almost 60% training correct. Though now some over-fitting has become evident in lower evaluation set scores. This fits the narrative from teh eprop work.

TODO: Re-investigate different regularisations ...

9 GeNN Implementation Notes (OUTDATED)

10 Derivation of exact solution for λ_V and λ_I within a timestep

$$\tau_m \lambda_V' = -\lambda_V \tag{59}$$
$$\tau_{\text{syn}} \lambda_I' = -\lambda_I + \lambda_V \tag{60}$$

From equation (59) we easily get

$$\lambda_V(t+\Delta t - t') = \lambda_V(t+\Delta t)\exp\left(-\frac{t'}{\tau_m}\right). \tag{61}$$

Then we have

$$\lambda_I' = \frac{1}{\tau_{\text{syn}}}\left(-\lambda_I + \lambda_V(t+\Delta t)\exp\left(-\frac{t'}{\tau_m}\right)\right) \tag{62}$$

The solution to the autonomous equation is

$$\lambda_I = \tilde{\lambda}_I \exp\left(-\frac{t'}{\tau_{\text{syn}}}\right) \tag{63}$$

and with the usual trick $\tilde{\lambda}_I \to \tilde{\lambda}_I(t)$ we get

$$\lambda_I' = \tilde{\lambda}_I' \exp\left(-\frac{t'}{\tau_{\text{syn}}}\right) - \frac{1}{\tau_{\text{syn}}}\tilde{\lambda}_I \exp\left(-\frac{t'}{\tau_{\text{syn}}}\right) = -\frac{1}{\tau_{\text{syn}}}\lambda_I + \tilde{\lambda}_I' \exp\left(-\frac{t'}{\tau_{\text{syn}}}\right) \tag{64}$$

Hence, we can identify

$$\tilde{\lambda}_I' \exp\left(-\frac{t'}{\tau_{\text{syn}}}\right) = \frac{1}{\tau_{\text{syn}}}\lambda_V = \frac{1}{\tau_{\text{syn}}}\lambda_V(t+\Delta t)\exp\left(-\frac{t'}{\tau_m}\right) \tag{65}$$
$$\Rightarrow \quad \tilde{\lambda}_I' = \frac{1}{\tau_{\text{syn}}}\lambda_V(t+\Delta t)\exp\left(-\left(\frac{1}{\tau_m} - \frac{1}{\tau_{\text{syn}}}\right)t'\right) \tag{66}$$
$$\Rightarrow \quad \tilde{\lambda}_I = -\frac{1}{\frac{1}{\tau_m} - \frac{1}{\tau_{\text{syn}}}}\frac{1}{\tau_{\text{syn}}}\lambda_V(t+\Delta t)\exp\left(-\left(\frac{1}{\tau_m} - \frac{1}{\tau_{\text{syn}}}\right)t'\right) + C \tag{67}$$
$$\Rightarrow \quad \lambda_I = -\frac{\tau_m}{\tau_{\text{syn}} - \tau_m}\lambda_V(t+\Delta t)\exp\left(-\left(\frac{1}{\tau_m} - \frac{1}{\tau_{\text{syn}}}\right)t' - \frac{t'}{\tau_{\text{syn}}}\right) + C\exp\left(-\frac{t'}{\tau_{\text{syn}}}\right) \tag{68}$$
$$= -\frac{\tau_m}{\tau_{\text{syn}} - \tau_m}\lambda_V(t+\Delta t)\exp\left(-\frac{t'}{\tau_m}\right) + C\exp\left(-\frac{t'}{\tau_{\text{syn}}}\right) \tag{69}$$

At $t' = 0$,

$$-\frac{\tau_m}{\tau_{\text{syn}} - \tau_m}\lambda_V(t+\Delta t) + C = \lambda_I(t+\Delta t) \tag{70}$$
$$\Rightarrow \quad C = \lambda_I(t+\Delta t) + \frac{\tau_m}{\tau_{\text{syn}} - \tau_m}\lambda_V(t+\Delta t) \tag{71}$$

and hence

$$\lambda_I = \frac{\tau_m}{\tau_{\text{syn}} - \tau_m}\lambda_V(t+\Delta t)\left(\exp\left(-\frac{t'}{\tau_{\text{syn}}}\right) - \exp\left(-\frac{t'}{\tau_m}\right)\right) + \lambda_I(t+\Delta t)\exp\left(-\frac{t'}{\tau_{\text{syn}}}\right) \tag{72}$$

And accordingly for $t' = \Delta t$,

$$\lambda_V(t) = \lambda_V(t+\Delta t)\exp\left(-\frac{\Delta t}{\tau_m}\right) \tag{73}$$
$$\lambda_I(t) = \frac{\tau_m}{\tau_{\text{syn}} - \tau_m}\lambda_V(t+\Delta t)\left(\exp\left(-\frac{\Delta t}{\tau_{\text{syn}}}\right) - \exp\left(-\frac{\Delta t}{\tau_m}\right)\right) + \lambda_I(t+\Delta t)\exp\left(-\frac{\Delta t}{\tau_{\text{syn}}}\right) \tag{74}$$

As the forward equations are identical for V (i.e. like the λ_I equation), the equivalent equation can be used for V in the forward pass.

11 GeNN Implementation Notes (Outdated?)

- In the normal forward pass we need to save:
  - spike times (could be the normal spike time recording bit array)
  - $(-V+I)_{n(k)}(t_k^{\text{post}})$ i.e. the input current minus voltage (to represent $\tau_{\text{mem}}(\dot{V}^-)_{n(k)}$) at each spiking neuron when a spike is detected (before reset). This could be done as part of the reset code but we need a new data structure for it. A per neuron ring buffer to hold data for an estimated maximum number of spikes?
  - We also need $l_V^- - l_V^+$ at times t_k^{post}, which I guess is relevant when the spiking neuron's V contributes to the voltage-dependent loss. At other spike times the relevant V's should be continuous and the difference 0. This data is one number for all neurons. One would probably save it for all spike times. In a egp ring buffer?
- I imagine a separate "backward pass", probably to be added as an additional feature to GeNN, in which
  - The λ continuos dynamics is integrated backward with Euler time steps as usual. This could be entered by users as a "backward_sim_code"
  - Spikes are retrieved from the spike recording buffer into the current spikes buffer (in an inversion of the usual recording code)
  - There would be a "backward_reset_code" where all spiking neurons get the jumps applied. This will access the $-V+I$ buffer and step back the counter on it.
  - All synapses execute their "backward_sim_code" which adds the $-\tau_{\text{syn}}\lambda_{I,\text{post}}$ value to their "gradient" variable that respresents $\frac{d\mathcal{L}}{dw_{ji}}$.
- At the end of the "backward pass", all synapse weights get updated with $-\eta\cdot$ gradient.

We could also try to hack it where, depending on a flag, either forward or backward things happen within normal sim_code etc structures. But a) what do we do about the additional data structures, especially the per-neuron one? and b) how would we handle the retrieving of spikes into the current spike array efficiently? Also, would we need to do something awkward about the time variable?

12 Loss functions

The desired loss functions for MNIST and SHD are one of the following:

$$\mathcal{L}_{\max} = -\frac{1}{N_{\text{batch}}}\sum_{i=1}^{N_{\text{batch}}} \log \frac{\exp\left(\max_{t\in[0,T]} V_{l(i)}(t)\right)}{\sum_{k=1}^{N_{\text{out}}}\exp\left(\max_{t\in[0,T]} V_k(t)\right)} \tag{75}$$

and

$$\mathcal{L}_{\text{sum}} = -\frac{1}{N_{\text{batch}}}\sum_{i=1}^{N_{\text{batch}}} \log \frac{\exp\left(\int_0^T V_{l(i)}(t)dt\right)}{\sum_{k=1}^{N_{\text{out}}}\exp\left(\int_0^T V_k(t)dt\right)}. \tag{76}$$

Unfortunately, neither of them have the prescribed functional form (1). this means that the original derivation cannot be applied to these loss functions. Repeated attempts to do so have not led to a successful translation.

Pehle and Wunderlich argue that

$$\max_{t\in[0,T]} V_k(t) = \int_0^T V_k(t)\delta(t-t_{\max})dt \tag{77}$$

and that one can deduce jumps for λ_V at $t_{\max}$ that are equivalent to the jumps normally encountered at spike times. If we follow that logic, we need to inject into $\lambda_{V,n}$ the term $\frac{\partial l_{\text{m}}(V(t))}{\partial V_n(t)}$ where

$$l_{\text{m}}(t_{\max}^j) \tag{78}$$

$$= -\frac{1}{N_{\text{batch}}}\sum_{i=1}^{N_{\text{batch}}} \log \frac{\exp\left(\int_0^T V_{l(i)}(t)\delta(t-t_{\max}^{l(i)})dt\right)}{\sum_{k=1}^{N_{\text{out}}}\exp\left(\int_0^T V_k(t)\delta(t-t_{\max}^k)\right)} \tag{79}$$

This only contributes at the times t^n_{max}, when we'd inject

$$\begin{cases} \delta(0) - \frac{\exp(V^{\mathrm{max}}_{l(i)})}{\sum_{k=1}^{N_{\mathrm{out}}} \exp(V^{\mathrm{max}}_k)} \delta(0) & \mathrm{if} n = l(i) \\ -\frac{\exp(V^{\mathrm{max}}_n)}{\sum_{k=1}^{N_{\mathrm{out}}} \exp(V^{\mathrm{max}}_k)} \delta(0) & \mathrm{otherwise} \end{cases} \tag{80}$$

This seems to suggest jumps of

$$\begin{cases} \lambda_{l(i)} += 1 - \frac{\exp(V^{\mathrm{max}}_{l(i)})}{\sum_{k=1}^{N_{\mathrm{out}}} \exp(V^{\mathrm{max}}_{l(i)})} & \mathrm{at} t = t^{l(i)}_{\mathrm{max}} \\ \lambda_n += -\frac{\exp(V^{\mathrm{max}}_n)}{\sum_{k=1}^{N_{\mathrm{out}}} \exp(V^{\mathrm{max}}_k)} & \mathrm{at} t = t^n_{\mathrm{max}} \mathrm{for} n \neq l(i) \end{cases} \tag{81}$$

However, this arises from pretending that l_m was a function l_V that was integrated over time to obtain the loss:

$$\mathcal{L} = \int_0^T l_V(t)dt \tag{82}$$

which it is not. I am not sure this is mathematically correct ... though the MNIST example appeared to learn.

13 Debugging the average cross-entropy results

As an alternative to the loss functions (75) and (76) which are technically not compliant to the eventProp algorithm I decided to work with the average cross-entropy loss

$$\mathcal{L} = \int_0^T -\frac{1}{N_{\mathrm{batch}}} \sum_{i=1}^{N_{\mathrm{batch}}} \log \frac{\exp(V_{l(i)}(t)}{\sum_{k=1}^{N_{\mathrm{class}}} \exp(V_k(t))} \tag{83}$$

Using this average cross-entropy, the networks do not learn the SHD task. Learning success is almost 0 (successful prediction at chance level of 5%) and we note that there is a tendency to drive the hidden layer to silence. While this can be counter-acted by stronger regularisation, I have also discovered that neurons that drive the correct output neuron through strong positive weights appear to get the most weakened input synapses (on average) and eventually fall silent. This could be due to the local gradient reflecting that spacing out input spikes (to the output layer) into regular intervals and across the entire trial is beneficial for reducing loss (as additional, coincident input spikes with positive weight to the correct output neuron have diminishing return due to the non-linearity of the cross-entropy.)

An interesting minimal illustration of this hypothesis is captured in `single_neuron_analysis.ipynb`. Here, the effect of a pattern of intput spikes ontot the correct output neuron is analysed. One can see how the first spike tends to lead to a positive $\lambda_V - \lambda_I$, i.e. potentiation, while the last spike (first in backward pass) akways leads to negative $\lambda_V - \lambda_I$, i.e. tending to depression – moving the first spike earlier and the last one later ...

This is especially true where there is a silence after the last spike until trial end. Note, however, that this is slightly simplistic as it does not take into account the sequential dependency of the λ_V in the hidden layer on it's own past.

Another elucidating experiment is the following set of parameters in `train_SHD.py`:

```
{"NAME": "test_axe1", "DATASET": "SHD", "DEBUG": false, "DEBUG_HIDDEN_N"
   : true, "OUT_DIR": ".", "DT_MS": 1, "BUILD": true, "TIMING": true, "
   TRAIN_DATA_SEED": 372, "TEST_DATA_SEED": 814, "MODEL_SEED": 135, "
   TRIAL_MS": 1400, "N_MAX_SPIKE": 1500, "N_BATCH": 32, "SUPER_BATCH": 1
   , "N_TRAIN": 256, "N_VALIDATE": 32, "N_EPOCH": 10, "SHUFFLE": true, "
   N_TEST": 10000, "NUM_HIDDEN": 256, "RECURRENT": false, "TAU_SYN": 5.0
   , "TAU_MEM": 20.0, "V_THRESH": 1.0, "V_RESET": 0.0, "
   INPUT_HIDDEN_MEAN": 0.02, "INPUT_HIDDEN_STD": 0.01, "
   HIDDEN_OUTPUT_MEAN": 0.0, "HIDDEN_OUTPUT_STD": 0.3, "
   HIDDEN_HIDDEN_MEAN": 0.0, "HIDDEN_HIDDEN_STD": 0.02, "PDROP_INPUT": 0
   .1, "PDROP_HIDDEN": 0.0, "REG_TYPE": "simple", "LBD_UPPER": 0, "
   LBD_LOWER": 2e-08, "NU_UPPER": 15, "NU_LOWER": 5, "RHO_UPPER": 10000.
   0, "GLB_UPPER": 1e-08, "ETA": 0.002, "ADAM_BETA1": 0.9, "ADAM_BETA2":
```

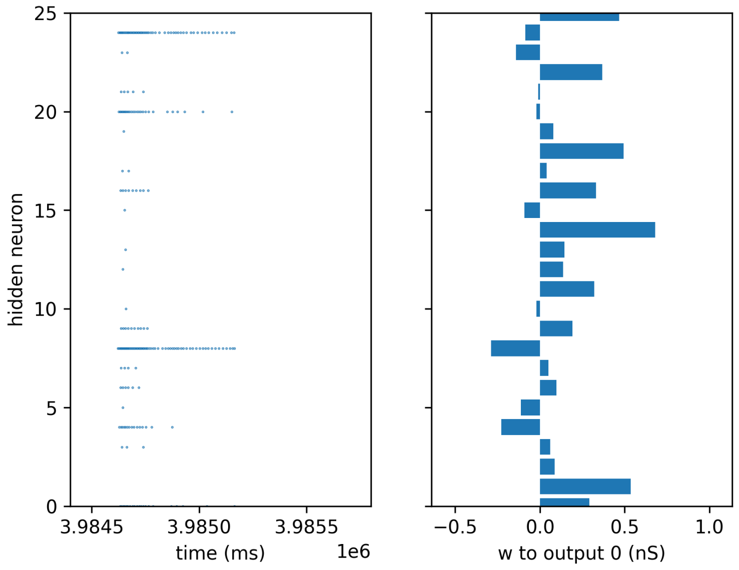

Figure 2: Illustration of the anti-correlation of spikes in hidden neurons and positive weights to the correct output neuron (neuron 0). Left, spike pattern of hidden neurons for one example input. Right, Weights of the hidden neurons onto the correct output neuron. It appears that hidden neurons that have strong positive weights onto the correct output neuron have little or no spikes.

```
 0.999, "ADAM_EPS": 1e-08, "ETA_DECAY": 1.0, "ETA_FIDDELING": false,
"ETA_REDUCE": 0.5, "ETA_REDUCE_PERIOD": 50, "W_OUTPUT_EPOCH_TRIAL": [
[0, 0], [0, 1], [0, 2], [0, 3], [0, 4], [0, 5], [0, 6], [0, 7], [9, 0
], [9, 1], [9, 2], [9, 3], [9, 4], [9, 5], [9, 6], [9, 7]], "
SPK_REC_STEPS": 1400, "REC_SPIKES_EPOCH_TRIAL": [[0, 0], [0, 1], [0,
2], [0, 3], [0, 4], [0, 5], [0, 6], [0, 7], [9, 0], [9, 1], [9, 2], [
9, 3], [9, 4], [9, 5], [9, 6], [9, 7]], "REC_SPIKES": ["input", "
hidden"], "REC_NEURONS_EPOCH_TRIAL": [[0, 0], [0, 1], [0, 2], [0, 3],
 [0, 4], [0, 5], [0, 6], [0, 7], [9, 0], [9, 1], [9, 2], [9, 3], [9,
4], [9, 5], [9, 6], [9, 7]], "REC_NEURONS": [["output", "V"], ["
output", "sum_V"], ["output", "lambda_V"], ["output", "lambda_I"], ["
hidden", "lambda_V"]], "REC_SYNAPSES_EPOCH_TRIAL": [], "REC_SYNAPSES"
: [], "WRITE_TO_DISK": true, "LOAD_LAST": false, "LOSS_TYPE": "
avg_xentropy", "EVALUATION": "speaker", "CUDA_VISIBLE_DEVICES": false
, "AVG_SNSUM": true, "REDUCED_CLASSES": [0], "W_REPORT_INTERVAL": 110
00, "REWIRE_SILENT": false, "AUGMENTATION": {}}
```

This runs the input 0 only and with the normal trial duration of 1400 ms. We observe that the network unlearns if anything unless strong enough regularisation from below is included, the number of spikes in the hidden layer goes to 0. If regularisation is included (as shown above, `lambda_lower` = 2e-08), we can prevent silencing of the middle layer, but the system still does not learn. We can see that high output weights onto the correct output neuron (neuron 0) seem to coincide with low spiking in the corresponding hidden neuron (see fig 2.

If we, however, use the same set of parameters but

```
    {"TRIAL_MS": 400}
```

then the network learns the task perfectly, with rapidly decreasing loss. The number of spikes in the hidden layer is increasing if anything. This illustrates how (I think) the silent time at the end of the trial (which carries considerable loss as all output voltages are equal 0) is affecting the gradient in such a way that hidden neurons' spikes are moved to later times but then inadvertantly deleted and the hidden layer falls silent and eventually the network fails completely. Figures 5-**??** show the outcome after 10 epochs analogously to figure 2-**??**. Now the correlations between weights and hidden unit activity are positive which makes sense for this 1-class artificial problem.

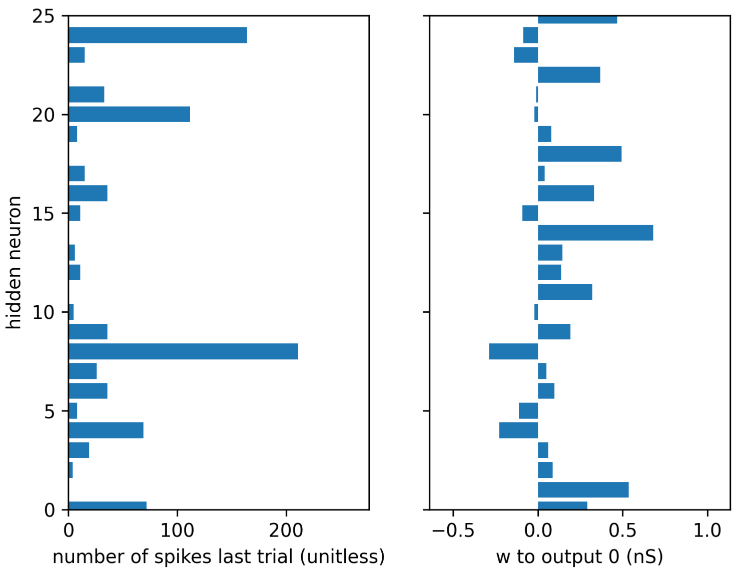

Figure 3: Illustration of the anti-correlation of spikes in hidden neurons and positive weights to the correct output neuron (neuron 0) as in figure 2 but displaying the number of spikes of hidden neurons during the last mini-batch on the left.

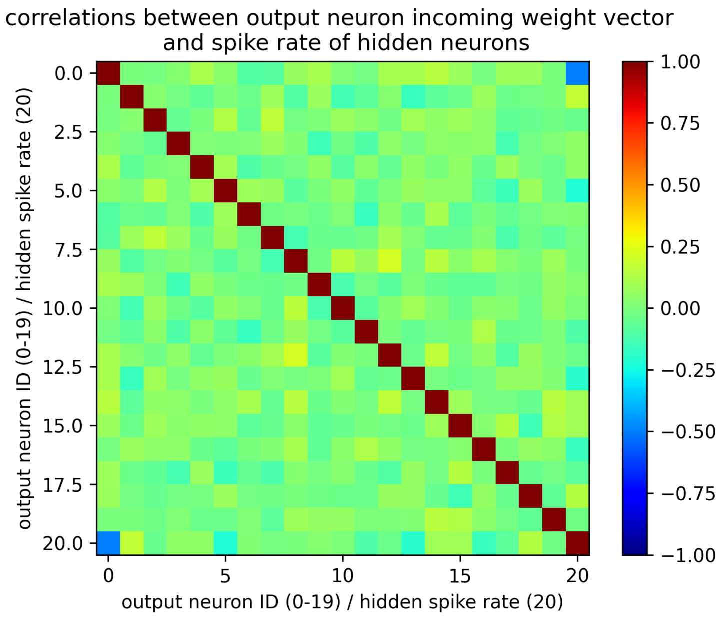

Figure 4: Correlation matrix between the weight vectors of the 20 output neurons and between each weight vector and the vector of number of spikes during the last mini-batch in the hidden neurons. Self-correlations are obviously (diagonal), correlations between weight vectors are low, and correlations between weight vectors and spike numbers are low amplitude except for the correct output neuron (corners) where there is a strong negative correlation of the weight vector with the spike number vector. This quantifies the effect illustrated for examples in the previous figures.

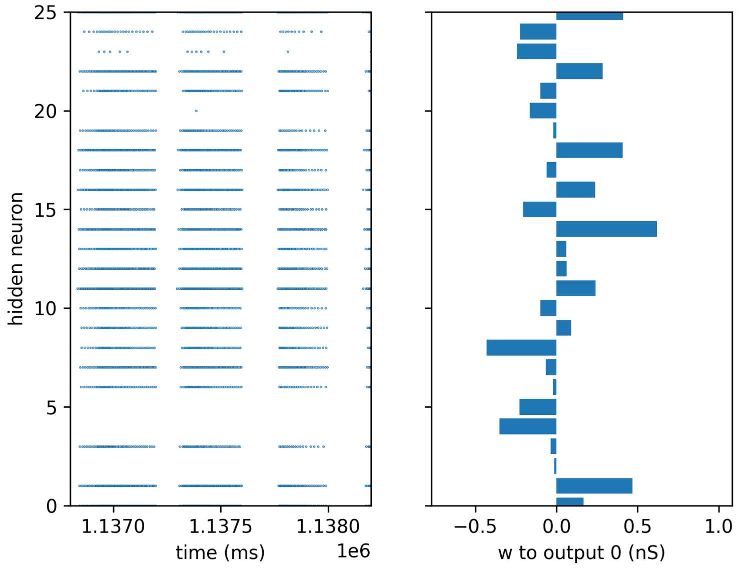

Figure 5: As figure 2 but for TRIAL_MS = 400 ms.

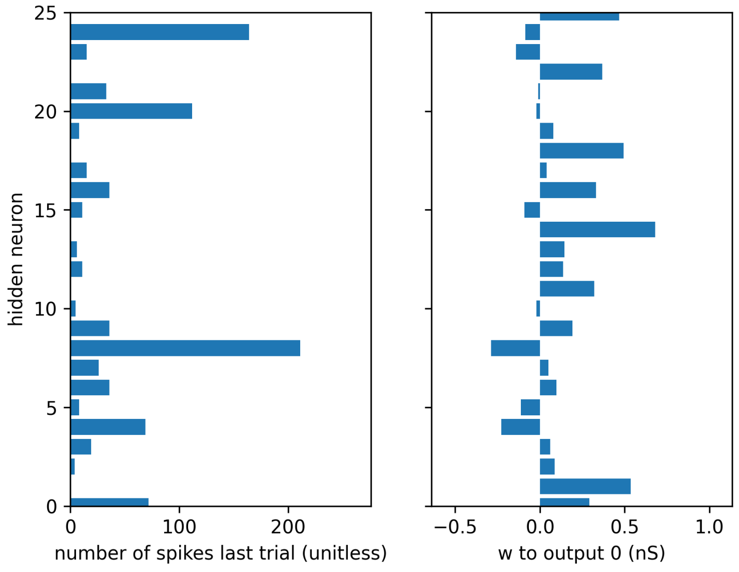

Figure 6: As figure 3 but for TRIAL_MS = 400 ms.

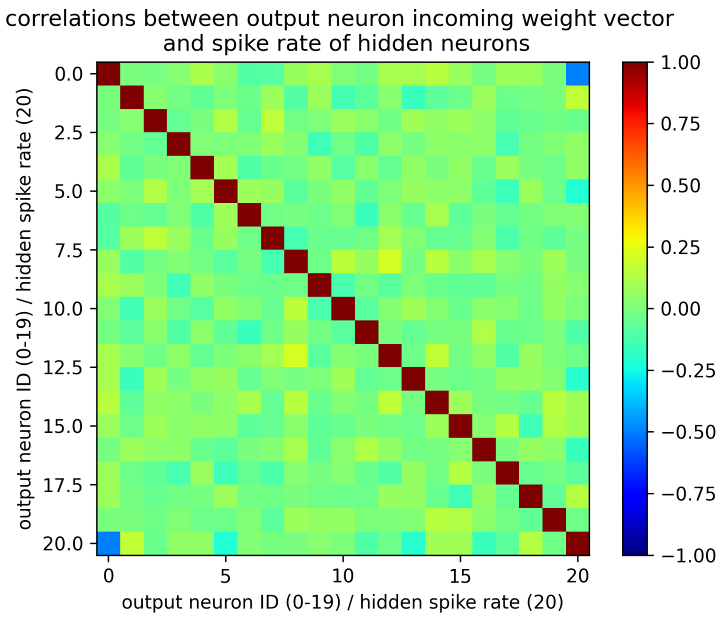

Figure 7: As figure 4 but for TRIAL_MS = 400 ms.

A detailed understanding of this effect in the context of the full system with 20 classes and batched training on many examples is difficult but the simplistic experiments described above suggest that the local exact gradient for the average cross-entropy loss inadvertently tends to switch off the hidden neurons best placed to provide information about a given class and so negates global optimisation (for the normal 1400 ms duration). It is worth noting in this context that (I believe) the local gradient "cannot know" about spike deletion once the input gets too weak. If moving a spike time to a later point in time is reducing the loss, following the local gradient will lead to reduced weights into the neuron in question (which will delay the spike but might also delete it!).

Running the network with a trial time of 400 ms (`train_SHD_avg_xentropy.py`),

```
{"NAME": "test_axe3", "DATASET": "SHD", "DEBUG": false, "DEBUG_HIDDEN_N"
    : true, "OUT_DIR": ".", "DT_MS": 1, "BUILD": true, "TIMING": true, "
    TRAIN_DATA_SEED": 372, "TEST_DATA_SEED": 814, "MODEL_SEED": 135, "
    TRIAL_MS": 400, "N_MAX_SPIKE": 1500, "N_BATCH": 32, "SUPER_BATCH": 1,
     "N_TRAIN": 7900, "N_VALIDATE": 512, "N_EPOCH": 2000, "SHUFFLE": true
    , "N_TEST": 10000, "NUM_HIDDEN": 256, "RECURRENT": false, "TAU_SYN":
    5.0, "TAU_MEM": 20.0, "V_THRESH": 1.0, "V_RESET": 0.0, "
    INPUT_HIDDEN_MEAN": 0.02, "INPUT_HIDDEN_STD": 0.01, "
    HIDDEN_OUTPUT_MEAN": 0.0, "HIDDEN_OUTPUT_STD": 0.3, "
    HIDDEN_HIDDEN_MEAN": 0.0, "HIDDEN_HIDDEN_STD": 0.02, "PDROP_INPUT": 0
    .1, "PDROP_HIDDEN": 0.0, "REG_TYPE": "simple", "LBD_UPPER": 0, "
    LBD_LOWER": 2e-08, "NU_UPPER": 15, "NU_LOWER": 5, "RHO_UPPER": 10000.
    0, "GLB_UPPER": 1e-08, "ETA": 0.002, "ADAM_BETA1": 0.9, "ADAM_BETA2":
     0.999, "ADAM_EPS": 1e-08, "ETA_DECAY": 1.0, "ETA_FIDDELING": false,
    "ETA_REDUCE": 0.5, "ETA_REDUCE_PERIOD": 50, "W_OUTPUT_EPOCH_TRIAL": [
    [0, 0], [0, 1], [0, 2], [0, 3], [0, 4], [0, 5], [0, 6], [0, 7], [9, 0
    ], [9, 1], [9, 2], [9, 3], [9, 4], [9, 5], [9, 6], [9, 7]], "
    SPK_REC_STEPS": 400, "REC_SPIKES_EPOCH_TRIAL": [[0, 0], [0, 1], [0, 2
    ], [0, 3], [0, 4], [0, 5], [0, 6], [0, 7], [9, 0], [9, 1], [9, 2], [9
    , 3], [9, 4], [9, 5], [9, 6], [9, 7]], "REC_SPIKES": ["input", "
    hidden"], "REC_NEURONS_EPOCH_TRIAL": [[0, 0], [0, 1], [0, 2], [0, 3],
     [0, 4], [0, 5], [0, 6], [0, 7], [9, 0], [9, 1], [9, 2], [9, 3], [9,
    4], [9, 5], [9, 6], [9, 7]], "REC_NEURONS": [["output", "V"], ["
    output", "sum_V"], ["output", "lambda_V"], ["output", "lambda_I"], ["
    hidden", "lambda_V"]], "REC_SYNAPSES_EPOCH_TRIAL": [], "REC_SYNAPSES"
    : [], "WRITE_TO_DISK": true, "LOAD_LAST": false, "LOSS_TYPE": "
```

```
   avg_xentropy", "EVALUATION": "speaker", "CUDA_VISIBLE_DEVICES": false
   , "AVG_SNSUM": true, "REDUCED_CLASSES": null, "W_REPORT_INTERVAL": 11
   000, "REWIRE_SILENT": false, "AUGMENTATION": {}}
```

we observe some learning, but it is lack-luster ... around 40% successful for training and 20% for evaluation after very long training time. In the process of testing this we discovered that on occasion output voltages can drift to all being negative and to such an extent that the mechanism of subtracting the maximum voltage to keep exponentials within numerical range failed as it started with the assumption that the maximum would be larger than 0. We now start off the calculation with `-FLT_MAX` instead.

14 Some notes about the incorrect "constant cross-entropy" attempt

I did notice previously that regularisation needed to be a lot weaker in the hidden layer with this model and eventprop as opposed to Jamie's e-prop network. Subsequently it became clear that due to the constant cross-entropy term that was driving λ_V in the output neurons, the transmitted $\lambda_V - \lambda_I$ was extremely small (just stemming from λ_I lagging in the exponential rise/fall to (driven) steady state from starting at 0. Weight changes in the hidden layer were therefore most delicate and easily destroyed by too strong regularisation. This small dirve also explains the slow learning observed compared to e-prop, Jamie typically trains 50 epochs and has convergence easily while here, 200 or more were advised (and that already included fairly high global learning rate).

So, in summary, the incorrect constant cross-entropy approach was clearly not ideal and is not sensible for this reason alone. It also appears quite coarse as no temoral dynamics of the neurons is actually considered. A more fine-grained loss function should be more successful (though with the full toolbox of augmentation etc the performance was not horrible - but it took a long while to achieve it).