Training a Network of Spiking Neurons with Equilibrium Propagation

Peter O'Connor [1] **Efstratios Gavves** [1] **Max Welling** [1 2]

Abstract

Backpropagation is almost universally used to train artificial neural networks. However, there are several reasons that backpropagation could not be plausibly implemented by biological neurons. Among these are the facts that (1) biological neurons appear to lack any mechanism for sending gradients backwards across synapses, and (2) biological "spiking" neurons emit binary signals, whereas back-propagation requires that neurons communicate real numbers between one another. Recently (Scellier and Bengio, 2017), demonstrated an alternative to backpropagation, called Equilibrium Propagation, wherein gradients are implicitly computed by the dynamics of the neural network, so that neurons do not need an internal mechanism for backpropagation of gradients. This provides an interesting solution to problem (1). In this paper, we address problem (2) by proposing a way in which Equilibrium Propagation can be implemented with neurons which are constrained to just communicate binary values at each time step. We show that with appropriate step-size annealing, we can converge to the same fixed-point as a real-valued neural network, and that with predictive coding, we can make this convergence much faster. We demonstrate that the resulting model can be used to train a neural network using the update scheme from Equilibrium propagation.

1. Introduction

The "neurons" used in deep learning are so-named because of their loose correspondence to biological neurons. There are however, a number of fundamental differences between the types of neurons used in deep learning and those we observe in biology (Crick, 1989). Among them are :

1. **Gradient Propagation**: Neurons used in deep learning emit two types of signals - an activation on the forward pass, and a gradient on the backward pass. Biological neurons are only known to emit one kind of signal - a forward activation, and do not appear to transmit gradients backwards across synapses.

2. **Activations Functions**: Neurons in deep learning have continuous, differentiable activation functions. This is necessary in order to propagate useful gradients back through the network. Biological neurons instead emit streams of all-or-nothing impulses called "spikes", which are some function of recent inputs to the neuron.

These two characteristics pose a conundrum to those looking to reconcile theories in machine-learning with how the brain might be reasonably expected to operate. The recent successes in deep learning have been based on achieving gradient-descent by propagating error-gradients backwards through a network. But it is not clear at all how biological neurons could achieve this.

Equilibrium Propagation, proposed by (Scellier and Bengio, 2017), showed how one may propagate gradients through a deep network in a setting where neurons only produce one type of signal, the forward activation. The authors use a continuous Hopfield network (Hopfield, 1984) - a symmetrically-weighted neural network whose dynamics are defined according to the gradient of an energy function ($\frac{\partial s}{\partial t} \propto -\frac{\partial E}{\partial s}$, where s is the state of the neurons). Learning is based on allowing the network to converge to a fixed-point conditioned on the input data, then perturbing the output units towards the target, and then updating parameters to minimize a contrastive loss between the fixed-point state and the perturbed state. Their work showed a semi-plausible mechanism by which biological neural may be able to achieve gradient descent.

The original formulation of Equilibrium Propagation, however, still assumes continuous-valued units. In this paper, we constrain neurons to emit binary-valued signals, and look at how neurons can efficiently convey their real-valued activations to other neurons despite this bottleneck.

This line of research may be of interest for designing the next generation of neural network hardware. A continuous-dynamical system can be implemented with an analog cir-

[*]Equal contribution [1]QUVA Lab, University of Amsterdam [2]Qualcomm Netherlands. Correspondence to: Peter O'Connor <peter.ed.oconnor@gmail.com>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

cuits, but electrical issues such as capacitance, inductance, and cross-talk make it difficult to faithfully transmit analog values over longer distances. Digital signals, by comparison, can be transmitted with ease. The brain appears to use a hybrid approach, with neurons having analog internal dynamics but communicating with one another using digital "spikes".

2. Background

2.1. A Neural Network as a Dynamical System

Suppose we have a network of recurrently connected neurons with symmetric weights ($w_{ij} = w_{ji}$). This is known as a continuous Hopfield Network. (Hopfield, 1984) proposed an energy-function for such a network, which can be defined as:

$$E(s) = \frac{1}{2}\sum_u s_i^2 - \sum_{i \neq j} w_{ij}\rho(s_i)\rho(s_j) - \sum_i b_i \rho(s_i) \quad (1)$$

Where s_i is the activation of neuron i, w_{ij} and b_i are model parameters, and ρ is a nonlinearity ((Scellier and Bengio, 2017) use a hard sigmoid function: $\rho(s) = [s]_0^1$, where $[\cdot]_a^b$ indicates that values outside the range of a and b are clipped to these limits). Given this energy function, we can define the temporal dynamics that minimize this energy with respect to activations:

$$\frac{\partial s_j}{\partial t} = -\frac{\partial E(s_j)}{\partial s_j} = -s_j + \rho'(s_j)\left(\sum_i w_{ij}\rho(s_i) + b_j\right) \quad (2)$$

Where $\rho'(s_j)$ is the derivative of the activation function about s_j. For implementation in discrete time, this can be expressed as a difference-equation (this is known as the *Forward Euler Method*):

$$s_j^t = \left[(1-\epsilon)s_j^{t-1} + \epsilon\rho'(s_j^{t-1})\left(\sum_i w_{ij}\rho(s_i^{t-1}) + b_j\right)\right]_0^1 \quad (3)$$

Where $\epsilon \in (0, 1)$ can be seen either as the size of the time-step or as the learning-rate of the activations. This update will converge to the optimum for a sufficiently small ϵ (e.g. $\epsilon = \frac{1}{2}$).

2.2. Equilibrium Propagation

(Scellier and Bengio, 2017) proposed a method for using a continuous Hopfield Network to implement gradient descent on a loss defined over a subset of units in the network. They propose a two-phase learning algorithm called *Equilibrium Propagation*. In Equilibrium Propagation, units are partitioned into *input*, *hidden*, and *output* neurons, whose states we denote s_{in}, s_{hid}, and s_{out}, respectively. They define some loss function between some target variable y and output activations: $\mathcal{L} = C(s_{out}, y)$

In the "Negative Phase", we are given an input vector x, and clamp the corresponding input units to that value: ($s_{in} = x$). The remaining units (s_{hid} and s_{out}) are allowed to settle to an energy minimum s^- according to Equation 3.

In the "Positive Phase", the output units are "weakly clamped" by the loss function. The "weak clamping" is done by adding the loss to the energy function from Equation 1: $E^\beta(s) = E(s) + \beta C(s_{out}, y)$ (where β is some small scalar), and allowing the network to briefly settle to a state s^+. Finally, network parameters are updated according to the difference between these states:

$$\Delta w = \frac{\eta}{\beta}\left(\frac{\partial E(s^+)}{\partial w} - \frac{\partial E(s^-)}{\partial w}\right) \propto -\frac{\partial \mathcal{L}}{\partial w} \quad (4)$$

Where η is some learning rate. (Scellier and Bengio, 2017) show that for small β, the resulting parameter update is proportional to $\frac{\partial \mathcal{L}}{\partial w}$. Intuitively, the idea works by pulling the minima of $E(s)$ closer to the minima of $E^\beta(s)$ (when $s_{in} = x$) so that the network will gradually learn to naturally minimize the output loss.

3. Binary Communication

Suppose we now operate under the constraint that neurons can only output binary values at each time-step. Our objective is to still converge to the same fixed-points as the continuous-valued dynamical system. We modify Equation 3 to describe the dynamics of a binary "spiking" neuron as follows:

$$\begin{aligned} u_j^t &= \sum_i w_{ij} q_i^{t-1} \\ v_j^t, \psi_j^t &= dec(u_j^t, \psi_j^{t-1}) \\ \epsilon_j^t, \xi_j^t &= anneal(\epsilon_j^{t-1}, v_j^t, \xi_j^{t-1}) \\ s_j^t &= [(1-\epsilon_j^t)s_j^{t-1} + \epsilon_j^t \rho'(s_j^{t-1})\left(v_j^t + b_j\right)]_0^1 \\ q_j^t, \phi_j^t &= enc(\rho(s_j^t), \phi_j^{t-1}) \end{aligned} \quad (5)$$

Where $q_j^t \in \{0, 1\}$ is the binary signal from neuron j at time t. enc and dec are functions for encoding and decoding signals between neurons, $anneal$ is a function of updating the step size ϵ, and internal state variables ϕ, ψ, ξ of functions enc, dec and $anneal$ can take any form.

In this work we show how various definitions of enc, dec and $anneal$ affect the convergence of our discrete dynam-

ics to the true minimum of the energy (Equation 1). Our objective is then to settle as fast a possible to a fixed point under the constraint that neurons can only communicate binary signals at each time step. In the following sections we propose a quantization method that allows our neurons to efficiently settle towards this fixed point.

3.1. Stochastic Approximation

One approach we could take is to look at this as a Stochastic Approximation problem from the perspective of each neuron. The task of Stochastic Approximation is to keep an online estimate $\hat{\theta}^t$ of a time-varying parameter θ^t from a stream of noisy samples $x^t \sim \theta^t + \zeta^t$, where ζ^t is some unbiased noise. When θ^t is not constant in time, we stay the input is *nonstationary*.

(Robbins and Monro, 1951) showed that if the nonstationarity is transient (θ^t converges to a final value over time), we can sequentially average out the noisy samples to form estimates:

$$\hat{\theta}^t = (1 - \epsilon^t)\hat{\theta}^{t-1} + \epsilon^t x^t \tag{6}$$

If we anneal the step-size (or learning rate) ϵ^t in such a way that $\sum_{t=0}^{\infty} \epsilon^t = \infty$ and $\sum_{t=0}^{\infty} (\epsilon^t)^2 < \infty$, then our estimator eventually converges to the true parameter values ($\lim_{t\to\infty} \hat{\theta}^t = \theta$). For stationary problems, when $\theta^t = \theta^0 : \forall t$, the optimal annealing schedule is $\epsilon^t = \frac{1}{t}$, which corresponds to a simple moving average. For *nonstationary* signals (e.g. the activations in our network, which undergo some transient dynamics before settling), we can converge faster by forgetting early samples, so that the average is not corrupted by stale values. There are a number of ways to do this (George and Powell, 2006). A simple one is to schedule the step-size as:

$$\epsilon^t = \frac{\epsilon_0}{(t)^\eta} \tag{7}$$

With the exponent $\eta \in (\frac{1}{2}, 1)$. This guarantees that as $t \to \infty$, the inputs at $t = 0$ diminish to have zero weight relative to the most recent inputs, but the average still smooths over an ever-growing number of samples.

3.2. A Naive Approach: Stochastic Rounding and the Robinson-Munroe Annealing

In our case, the parameter we want to estimate is the total input received from all other neurons at the energy minimum: $\sum_i w_{ij}\rho(s_i^-)$. The noise arises from trying to represent real signals with a temporal stream of bits. The non-stationarity arises from the fact that the rest of the network has not yet settled to the fixed point. Note that our estimate itself affects future inputs: Neurons are connected recurrently in a network and the estimator in neuron i affects the estimator in neuron j which in turn affects the estimators in neuron i.

Suppose each input neuron i in Equation 5 stochastically outputs bits $q_i^t \sim \text{Bernoulli}(\rho(s_i))$, where $\rho(s_i) \in (0, 1)$ is the neuron's activation. Since q_i^t is an unbiased estimator of $\rho(s_i)$, a neuron j receiving this signal i should eventually average it out, along with all its other inputs, to achieve a correct estimate of $\sum_i w_{ij}\rho(s_i^-)$, provided that its input neurons do indeed converge to the correct fixed point s_i^-. A simple communication scheme can then be described (with reference to the variables in Equation 5) as:

$$q^t = \text{Bern}(\rho(s^t)) \qquad \text{Stochastic Encoder} \tag{8}$$

$$v^t = u^t \qquad \text{Identity Decoder} \tag{9}$$

$$\epsilon^t = \frac{1}{(t)^\eta} \qquad \text{Annealer} \tag{10}$$

3.3. Faster Convergence with Sigma-Delta Modulation

There are more efficient ways to communicate a time-varying real value than to send random bits centered around that value. A simple method from signal processing for encoding time-varying signals is Sigma-Delta modulation (Candy and Temes, 1962). Suppose we have a time-varying input signal $x_1, ...x_t$ where $x_\tau \in (0, 1)\forall\tau$. We then quantize x_t into q_t according to:

$$\begin{aligned} \phi' &= \phi^{t-1} + x^t \\ q^t &= \left[\phi' > \frac{1}{2}\right] \\ \phi^t &= \phi' - q^t \end{aligned} \qquad \text{Sigma Delta Encoder} \tag{11}$$

Where $[a > b]$ evaluates to 1 if $a > b$ and 0 otherwise. By expanding Equation 11 recursively, we can verify that if $x^t \in (0, 1)$ and $\phi_0 = 0$, the mean quantization error is bounded: $\frac{1}{T}\left|\sum_{t=0}^{T}(x^t - q^t)\right| \leq \frac{1}{2T}$. So we have $\mathcal{O}(1/T)$ convergence, compared to the $\mathcal{O}(1/\sqrt{T})$ convergence that we would get from averaging out a stochastic estimator.

Note that this corresponds to an "integrate-and-fire" quantization - inputs are added to a "potential" ϕ, and once that potential crosses a threshold a "spike" ($q^{(t)} = 1$) is sent out, and subtracted from the potential. Sigma-Delta modulation has previously been used as a model of the neural spiking mechanism: (Yoon, 2016), (Zambrano and Bohte, 2016), (O'Connor et al., 2017).

3.4. Predictive Coding

When the signal is time-varying, it seems like a poor use of bandwidth to simply communicate a stream of bits that

averages out to the current signal value. Instead, we can use an encoding scheme wherein neurons primarily send temporal *changes* in the signal value to downstream neurons, and downstream neurons integrate these changes. This is an instance of *Predictive Coding*, a widely used concept in the Signal Processing literature. Predictive Coding has in the past been proposed as a possible mechanism in neural communication. (Srinivasan et al., 1982), (Shin, 2001), (Tewksbury and Hallock, 1978), (Bharioke and Chklovskii, 2015).

Lossy Predictive Coding is a method for efficiently encoding a real-valued signal as a bitstream, and decoding it again on the other end of a communication channel. At each time-step, a *predictor* attempts to predict the current signal from past signal values, and the prediction is subtracted from the signal before quantization. On the receiving end, the same predictor is used to reconstruct the signal from the stream of bits. In the case where the predictor is a linear function of past inputs, we can exploit the commutativity of the weight-multiplication and decoding operations (O'Connor et al., 2017) to sandwich a weight matrix between the encoders and decoders. Here, we formulate an extremely simple predictor $\text{Pred}(x^{t-1}, ...x^0) = (1-\lambda)x^{t-1}$ where $\lambda \in (0, 1)$. We formulate this system (with reference to the variables in Equation 5) as:

$$\begin{aligned} a^t &= \frac{1}{\lambda}\big(\rho(s^t) - (1-\lambda)\rho(s^{t-1})\big) \quad \text{Predictive Encoder} \\ q^t &= Q(a^t) \end{aligned} \tag{12}$$

Where Q is some (possibly stateful) quantization procedure, such Sigma-Delta modulation (Equation 11) or Stochastic Rounding (Equation 8). On the decoding side, we sum up the weighted quantized inputs and invert the encoding function:

$$\begin{aligned} u_j^t &= \sum_i w_{ij} q_i^{t-1} \\ v_j^t &= (1-\lambda)v_j^{t-1} + \lambda u_j^t \quad \text{Predictive Decoder} \end{aligned} \tag{13}$$

When λ is close to 0, we have a system that only sends *changes* in state, and accumulates these change in a running sum. When λ is 1, we recover the case with no predictive coding.

3.5. Lambda-Annealing

As was the case with ϵ, it is also possible to anneal the prediction-factor λ. Intuitively, we would like to start the convergence process with a very short memory (λ close to 1), primarily using bits to communicate the rapidly changing current state. Later, as we approach a fixed point, we would like to lengthen the memory (λ close to 0) and use our bits to communicate increasingly fine increments to the state.

3.6. The Resulting Model

Combining Sigma-Delta encoding from Equation 11 with the predictive encoder/decoder of Equations 12 and 13 by plugging them all into Equation 5 results in a biologically-plausible model that applies double-exponential smoothing to inputs and produces output spikes with an integrate-and-fire mechanism:

$$\begin{aligned} v_j^t &= (1-\lambda^t)v_j^{t-1} + \lambda^t \sum_i w_{ij} q_i^{t-1} \\ s_j^t &= [(1-\epsilon^t)s_j^{t-1} + \epsilon^t \rho'(s_j^{t-1})\left(v_j^t + b_j\right)]_0^1 \\ a^t &= \frac{1}{\lambda^t}\big(\rho(s^t) - (1-\lambda^t)\rho(s^{t-1})\big) \\ q_j^t &= [\phi_j^{t-1} + a_j^t > \frac{1}{2}] \\ \phi_j^t &= \phi_j^{t-1} + a_j^t - q_j^t \end{aligned} \tag{14}$$

Figure 1 shows some example dynamics from our model.

4. Experiments

We explore several combinations of the hyperparameters ϵ^t, λ^t and the quantizer, introduced in Section 3. First in Section 4.1, we compare the rate at which these various hyperparameter settings converge to the fixed point for randomly initialized networks. Then in Section 4.2 we apply the more promising settings to train a neural network on the MNIST dataset.

4.1. Convergence

To understand how our encoding/decoding parameters affect the rate of convergence, we use a randomly initialized network with 3 layers of [500-500-10] units, where the first is considered the "input" layer and is clamped to a random input vector ($s_{in} = x$). We find the true fixed point s^* by running a continuous-valued network (Equation 3) for a long time, then evaluate the rate at which our binarized network activations s^t converge to the true fixed point: $error(t) = \|s^* - s^t\|$. We find that the best convergence is obtained by annealing both λ (the predictive coding term), and ϵ (the step size). Using a random search with 5000 trials, we looked for an optimal annealing schedule (in terms of the lowest mean error over the convergence process) in the form of Equation 7, and obtained $\epsilon^t = \frac{0.84}{t^{0.092}}$, $\lambda^t = \frac{0.83}{t^{0.58}}$. This seems to indicate that annealing λ is helpful. Figure 2 plots the results.

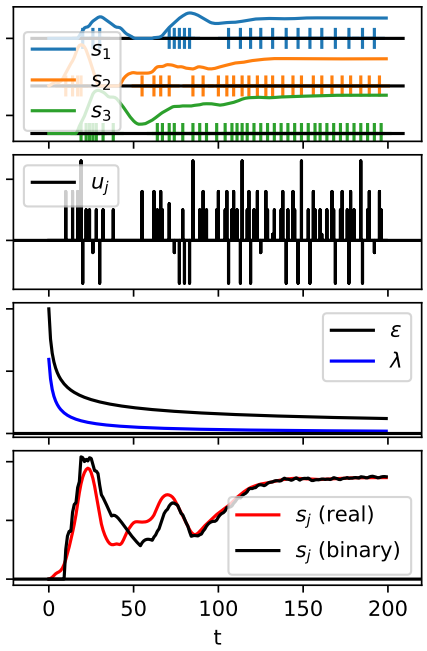

Figure 1. An illustration of the evolution of a neuron in response to converging inputs. **Top:** The values of three input neurons as they converge towards a fixed point. Tick marks indicate the times where the encoders of those neurons output a 1. **Row 2:** The total weighted input from the input neurons to a post-synaptic neuron. **Row 3:** the step size ϵ as it anneals from an initial value of 1. **Bottom:** A comparison of the value of the post-synaptic neuron under the continuous dynamics (Equation 3, red curve) and our binary dynamics (Equation 14, black curve).

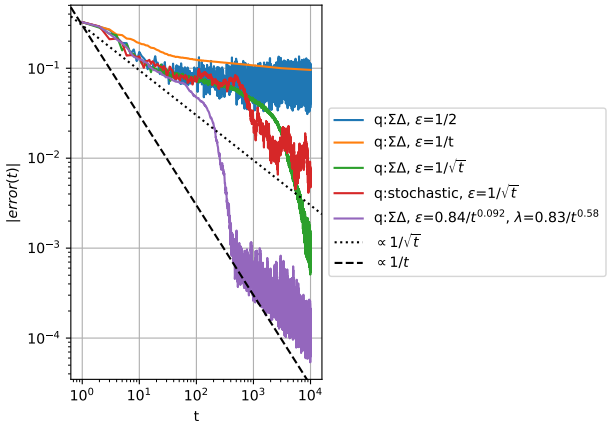

Figure 2. The convergence of the state of our binary network to the true Energy minimum (see Section 4.1). Note the log-log axis. $q = stochastic$ indicates stochastic rounding (Equation 8) and $q = \Sigma\Delta$ indicates Sigma-Delta quantization (Equation 11). The best results (purple line) were obtained by random search for ϵ and λ schedules in the form of Equation 7.

4.2. Equilibrium Propagation on MNIST

We applied our quantization methods to train our binary-valued network on the MNIST dataset using Equilibrium Propagation. We compare to our implementation of continuous-valued Equilibrium-Propagation by (Scellier and Bengio, 2017). Unlike the author's implementation, we did not use the trick of keeping persistent activations per training sample between epochs. This trick would have improved the performance of both the continuous and binary network but would not be useful in drawing conclusion about the performance of the binary network relative to the continuous one.

Somewhat surprisingly, our binary-network performed on-par with continuous-network (see Figure 3). Our binary network achieved a 2.34% test / 0.066% training error, compared to the continuous network's 2.57% test / 0.036% training error.

We found the performance to be quite sensitive to the choice of ϵ and λ hyperparameters. For many parameter settings, the training procedure completely failed to converge and network's prediction scores remained at chance levels. We suspect that the reason for these failures is that the network had not sufficiently converged by the end of the negative phase. Thus the difference between the positive-phase state s^+ and negative-phase state s^- in Equation 4 not only reflected the effect of the target-perturbation, but also direction that the network would have settled anyway (without the perturbation). This can push units into a useless, saturated regime.

5. Discussion

Our study is preliminary and there is much to explore. Much of the work in the stochastic approximation literature is about how to adapt step-sizes according to the statistics of the incoming sample stream (Chau and Fu, 2015), (George and Powell, 2006). We expect that step-size adaptation will lead to better performance, and intend to explore these methods in future work. It is also unclear whether we can simply transfer adaptive step-size algorithms to the similar task of choosing optimal predictive coding coefficients (λ). (Bharioke and Chklovskii, 2015) suggested that including a nonlinearity in the predictor can have the effect of rapid online adaptation of prediction coefficients. Adaptive step sizes will allow us to perform inference on nonstationary data. We can imagine constructing a network which settles to a precise fixed-point when data is unchanging but then adapts itself to take larger, coarser steps when it sees the input has started changing rapidly (e.g. during a *saccade*).

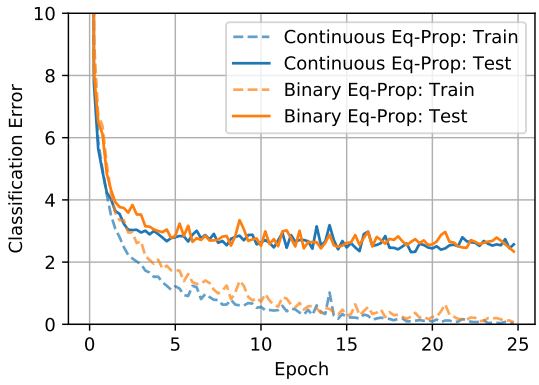

Figure 3. Learning curves on the MNIST dataset, on an equilibrium propagation network with one hidden layer. The **Blue Curves**: Training/Test set Learning curves of our implementation of "Real-Valued" Equilibrium propagation by (Scellier and Bengio, 2017). **Orange Curve:** Scores from our Binary-Values implementation with the best-performing quantization hyperparameters from Figure 2.

There are still several obstacles to doing truly biologically plausible deep learning. One subtle issue is that we rely on "capturing" a negative state s^- and a positive state s^+ in order to do the parameter update (Equation 4). This requires holding onto two states at once - something which biological neurons seem unlikely to do. If we instead take the approach of using the rate of change of the postsynaptic neuron at the beginning of the positive phase, as suggested in (Bengio et al., 2015), (Scellier and Bengio, 2017), (i.e. $\Delta w_{ij} \propto \rho(s_i)\frac{\partial s_j}{\partial t}$) we have the problem that we get very noisy updates due to the quantization. If we run the update for several time-steps so that the noise cancels, we face yet another problem - the trajectory of s_j^+ is not just being moved by the introduction of the target signal, but also the fact that w_{ij} has changed. This can cause a positive feedback loop. Empirically, we found that this prevented effective learning.

Another lingering biological-implausibility that is not specific to our algorithm is that we still use symmetric weights. Learning with symmetric weights implies an additional communication channel to synchronize synapse w_{ij} of neuron j to synapse w_{ji} of neuron i. However recent work by (Scellier et al., 2018) and (Lillicrap et al., 2014) seems to suggest that this may not be a problem, because weights tend to align themselves to be approximately symmetrical through learning anyway. Another obstacle is that biological networks do not appear to have distinct forward/backward passes, or negative/positive phases (except perhaps on the very slow timescale of the sleep/wake cycle). It is still not clear how one could do learning in a setting where new data is continuously coming in and we do not have the luxury of pausing our sensory input while we wait for our brains to do a forward/backward pass or settle to an energy minimum. Even if we could, we face the problem of Catastrophic Forgetting, wherein deep networks tend to forget old training data when presented with a nonstationary stream of training samples (though (Kirkpatrick et al., 2017) have done some work addressing this). Finally the question of how to learn temporal sequences without doing backpropagation-through-time remains an open one, though recent works such as (Ollivier et al., 2015) and (Tallec and Ollivier, 2017) have begun to address this.

We speculate that this work is relevant to the design of asynchronous neural computing hardware. One can imagine future neural-computers wherein neurons are implemented as loosely-coupled physical circuits that operate asynchronously. These neurons may internally perform analog computations, and asynchronously send digital "spikes" whenever a neuron's internal dynamics crosses some threshold. Such an architecture, with no need for a centralized clock and with memory closely coupled to processing, could allow for much more scalable neural computing machines.

Code is available at `https://github.com/QUVA-Lab/spiking-equilibrium-prop`

References

Yoshua Bengio, Thomas Mesnard, Asja Fischer, Saizheng Zhang, and Yuhuai Wu. Stdp as presynaptic activity times rate of change of postsynaptic activity. *arXiv preprint arXiv:1509.05936*, 2015.

Arjun Bharioke and Dmitri B Chklovskii. Automatic adaptation to fast input changes in a time-invariant neural circuit. *PLoS computational biology*, 11(8):e1004315, 2015.

James C Candy and Gabor C Temes. *Oversampling delta-sigma data converters: theory, design, and simulation*. University of Texas Press, 1962.

Marie Chau and Michael C Fu. An overview of stochastic approximation. In *Handbook of Simulation Optimization*, pages 149–178. Springer, 2015.

Francis Crick. The recent excitement about neural networks. *Nature*, 337(6203):129–132, 1989.

Abraham P George and Warren B Powell. Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming. *Machine learning*, 65(1): 167–198, 2006.

John J Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the national academy of sciences*, 81(10):3088–3092, 1984.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, page 201611835, 2017.

Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random feedback weights support learning in deep neural networks. *arXiv preprint arXiv:1411.0247*, 2014.

Peter O'Connor, Efstratios Gavves, and Max Welling. Temporally efficient deep learning with spikes. *arXiv preprint arXiv:1706.04159*, 2017.

Yann Ollivier, Corentin Tallec, and Guillaume Charpiat. Training recurrent networks online without backtracking. *arXiv preprint arXiv:1507.07680*, 2015.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

Benjamin Scellier and Yoshua Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in computational neuroscience*, 11:24, 2017.

Benjamin Scellier, Anirudh Goyal, Jonathan Binas, Thomas Mesnard, and Yoshua Bengio. Extending the framework of equilibrium propagation to general dynamics, 2018. URL https://openreview.net/forum?id=SJTB5GZCb.

Jonghan Shin. Adaptive noise shaping neural spike encoding and decoding. *Neurocomputing*, 38:369–381, 2001.

Mandyam V Srinivasan, Simon B Laughlin, and Andreas Dubs. Predictive coding: a fresh view of inhibition in the retina. *Proceedings of the Royal Society of London B: Biological Sciences*, 216(1205):427–459, 1982.

Corentin Tallec and Yann Ollivier. Unbiased online recurrent optimization. *arXiv preprint arXiv:1702.05043*, 2017.

S Tewksbury and RW Hallock. Oversampled, linear predictive and noise-shaping coders of order n> 1. *IEEE Transactions on Circuits and Systems*, 25(7):436–447, 1978.

Young C Yoon. Lif and simplified srm neurons encode signals into spikes via a form of asynchronous pulse sigma-delta modulation. 2016.

Davide Zambrano and Sander M Bohte. Fast and efficient asynchronous neural computation with adapting spiking neural networks. *arXiv preprint arXiv:1609.02053*, 2016.