Training a Spiking Neural Network with Equilibrium Propagation

O'Connor, P.; Gavves, E.; Welling, M.

Publication date
2019

Document Version
Final published version

Published in
Proceedings of Machine Learning Research

License
Other

Citation for published version (APA):
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 continuous values 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 spiking neural network using the update scheme from Equilibrium propagation.

1 Introduction

The human brain, a network of around $10^{11}$ neurons, consumes around 20W [Ling, 2001]. For comparison, a Titan X GPU running real-time object detection with YOLO [Redmon et al., 2016], a network of around $10^7$ neurons, consumes 250W. In the quest for more efficient hardware for deep learning, biology is a not a bad place to start looking.

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. **Biological Neurons do not do Backpropagation**: 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 send signals down a one-way signalling pathway called an **axon**. They appear to lack any secondary signalling mechanism for sending gradient backwards.

2. **Biological Neurons communicate with Spikes**: Neurons in deep learning have continuous, differentiable activation functions. This is necessary in order to propagate useful gradients back through the network. Biological neurons are best understood as dynamical systems, which output streams of all-or-nothing signals 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.

To address the “no biological backprop” problem, Scellier and Bengio [2017] proposed Equilibrium Propagation. This 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
Spiking Equilibrium Propagation

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 ($\frac{\partial}{\partial s_i} \propto -\frac{\partial E}{\partial s_i}$, where $s_i$ 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, letting the network settle again, and then
updating parameters to minimize a contrastive loss be-
tween the original fixed-point state and the perturbed
fixed-point. Their work showed a semi-plausible mech-
nism by which biological neural networks (or artificial
networks implemented as analog circuits) 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. Specifically, we show how a network of
neurons can efficiently minimize an energy function
when neurons are “spiking” - i.e. constrained to only
communicate binary values at each time-step.

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 circuits, but electrical issues such as capaci-
tance, inductance, and cross-talk make it difficult to
faithfully transmit analog values over a circuit. Dig-
tal signals, by comparison, can be transmitted with
ease. The brain appears to use a hybrid approach, with
neurons having analog internal dynamics but commun-
icating 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:

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

Where $s_i$ is the activation of neuron $i$, $w_{ij}$ and $b_i$ are
model parameters, and $\rho$ is a nonlinearity. Scellier and
Bengio [2017] use a hard sigmoid function: $\rho(s) = [s]_a^b$, where $[x]_a^b$ 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:

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

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):

$$s^t_j = [1 - \epsilon] s^{t-1}_j + \epsilon \rho'(s^{t-1}_j) \left( \sum_i w_{ij} \rho(s^{t-1}_i) + b_j \right) \tag{3}$$

Where $[i]_0^1$ indicates clipping to range [0, 1] and $\epsilon \in (0, 1)$
can be seen 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 $\epsilon$ (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 Prop-
agation, 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 be-
tween some target variable $y$ and output activations:

$$C(s_{out}, y)$$

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: $E^2(s) = E(s) + \beta C(s_{out}, y)$ (where $\beta$
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:

$$\Delta w = \eta \beta \left( \frac{\partial E(s^+)}{\partial w} - \frac{\partial E(s^-)}{\partial w} \right) \approx -\frac{\partial C(s_{out}, y)}{\partial w} \tag{4}$$

Where $\eta$ is some learning rate. Scellier and Bengio
[2017] show that for small $\beta$, the resulting parameter
update is proportional to $\frac{\partial \mathcal{L}(x, s, y)}{\partial w}$. 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 optimally converge to the same fixed-points as the continuous-valued dynamical system, under the constraint of binary communication between neurons. In other words, we constrain our neurons to obey the interface:

$$q_j^t, s_j^t, z_j^t = f(q_{ij}^{t-1}, s_{ij}^{t-1}, w_{ij}, b_j, z_j^{t-1})$$

(5)

Where $q_j^t \in \{0, 1\}$ is the binary output of neuron $j$, $q_{ij} \in \{0, 1\}^D$ are the binary signals of other neurons in the network, $s_j^t \in \mathbb{R}$ is the external state associated with a neuron, $w_{ij}, b_j$ are the parameters associated with neuron $j$, and $z_j$ is the internal state of encoders and decoders which we will discuss in the following section. Note that the only values that are communicated between neurons are the binary $q_j$’s. Our goal is to design our neurons so that despite being limited by binary communication, the states $s$ in our network to converge to the same fixed point as they would when following the real-valued dynamics of Equation 6.

We propose to design our neurons as follows:

$$u_j^t = \sum_i w_{ij} q_i^{t-1}$$
$$w_j^t, s_{dec}, z_{dec} = dec(u_j^t, z_{dec})$$
$$\epsilon_j^t, z_{anneal} = anneal(\epsilon_j^{t-1}, v_j, z_{anneal})$$
$$s_j^t = [(1 - \epsilon_j^t) s_j^{t-1} + \epsilon_j^t \rho'(s_j^{t-1}) (v_j + b_j)]_0^1$$
$$q_j^t, z_{enc} = enc(\rho(s_j^t), z_{enc})$$

(6)

Where $enc$ and $dec$ are functions for encoding and decoding signals between neurons, $anneal$ is a function of updating the step size $\epsilon$, and the form of internal state variables $z_j = (z_{dec}, z_{anneal}, z_{enc})$ will be defined in the following sections.

In this work we show how various definitions of $enc$, $dec$ and $anneal$ affect the convergence of our discrete dynamics to the true minimum of the energy (Equation 1). 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 $\theta^t$ from a stream of noisy samples $x^t = \theta^t + \zeta^t$, where $\zeta^t$ is some unbiased noise. When $\theta^t$ is not constant in time, we stay the input is nonstationary.

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

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

(7)

If we anneal the step-size (or learning rate) $\epsilon^t$ 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 ($\lim_{t \rightarrow \infty} \hat{\theta}^t = \theta^t$). For stationary problems, when $\theta^t = \theta^* : \forall t$, 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:

$$\epsilon^t = \frac{\theta}{(t)^{\eta}}$$

(8)

With the exponent $\eta \in (\frac{1}{2}, 1)$. This guarantees that as $t \rightarrow \infty$, 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 “true” parameter $\theta$ corresponds to the total pressure exerted on neuron $j$ by the rest of the network: $\rho'(s_j^{t-1}) (\sum_i w_{ij} \rho(s_i^{t-1}) + b_j)$ (from Equation 3). The noise arises from trying to represent real signals with a temporal stream of bits. The nonstationarity 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 $i$. 

O’Connor, Gavves, Welling
Suppose each input neuron $i$ in Equation 6 stochastically outputs bits $q^t_i \sim \text{Bernoulli}(\rho(s^t_i))$, where $\rho(s^t) \in (0, 1)$ is the neuron’s activation. Since $q^t_i$ is an unbiased estimator of $\rho(s^t_i)$, a neuron $j$ receiving this signal should eventually average it out, along with all its other inputs, to achieve a correct estimate of $\rho'(s_t^{i-1}) \left( \sum w_{ij} \rho(s_t^{i-1}) + b_j \right)$, 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 6) as:

\[
q^t = \text{Bern}(\rho(s^t)) \quad \text{Stochastic Encoder (9)}
\]
\[
v^t = u^t \quad \text{Identity Decoder (10)}
\]
\[
\epsilon^t = \frac{1}{(t)^n} \quad \text{Annealer (11)}
\]

\[
\phi^t = \phi^{t-1} + x^t \quad \text{Sigma Delta Encoder (12)}
\]
\[
q^t_i = \begin{cases} 
1 & \text{if } \phi^t > \frac{1}{2} \\
0 & \text{otherwise}
\end{cases}
\]
\[
\epsilon^t = \phi^t - q^t_i
\]

Where $[a > b]$ evaluates to 1 if $a > b$ and 0 otherwise. By expanding Equation 12 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 $O(1/T)$ convergence, compared to the $O(1/\sqrt{T})$ 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” $\phi$, and once that potential crosses a threshold a “spike” ($q^{i(t)} = 1$) 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].

3.3 Better Averaging with Adaptive Step Sizes

In choosing the step-size for stochastic optimization, we face a trade-off. Small step sizes allow us to average out noise, but also cause our current estimates to include outdated values of the time-varying parameter $\theta^t$. It can therefore be advantageous to adaptively adjust our step size according to our estimate of how non-stationary $\theta$ is. George and Powell [2006] review several existing step-size adaptation algorithms and propose one of their own called Optimal Step-Size Adaptation (OSA) which, similarly to a Kalman filter, adjusts its step-size according to the ratio of the estimated drift in the underlying parameter and the noise in the measurement. OSA is “optimal” in the sense that it optimally estimates the parameter $\theta$ if the drift and noise are known. Since they are not - OSA also estimates these quantities, and bases the step-size on these estimates. OSA has only a single parameter, $\nu$, which is the target learning rate for estimating the drift and noise. The full algorithm is included in Appendix A.

3.4 Better Encoding 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 $x_t \in (0, 1) \forall t$. We then quantize $x_t$ into $q_t$ according to:

\[
\phi^t = \phi^{t-1} + x^t
\]
\[
q^t_i = \begin{cases} 
1 & \text{if } \phi^t > \frac{1}{2} \\
0 & \text{otherwise}
\end{cases}
\]
\[
\epsilon^t = \phi^t - q^t_i
\]

3.5 Better Bit-Economy with 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], [Bharicioke and Chiklovskii, 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 write our encoder (with reference to the variables in Equation 6) as:

\[
\phi^t = \phi^{t-1} + x^t
\]
\[
q^t_i = \begin{cases} 
1 & \text{if } \phi^t > \frac{1}{2} \\
0 & \text{otherwise}
\end{cases}
\]
\[
\epsilon^t = \phi^t - q^t_i
\]
\[ a^t = \frac{1}{\lambda} (\rho(s^t) - (1 - \lambda)\rho(s^{t-1})) \]

\[ q^t = Q(a^t) \]

Where \( Q \) is some (possibly stateful) quantization procedure, such as Sigma-Delta modulation (Equation 12) or Stochastic Rounding (Equation 9). On the decoding side, we sum up the weighted quantized inputs and invert the encoding function:

\[ u^t_j = \sum_i w_{ij}q^t_i \]

\[ v^t_j = (1 - \lambda)v^{t-1}_j + \lambda u^t_j \]

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

### 3.6 Lambda-Annealing

As was the case with \( \epsilon \), it is also possible to anneal the prediction-factor \( \lambda \). Intuitively, we would like to start the convergence process with a very short memory (\( \lambda \) 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 (\( \lambda \) close to 0) and use our bits to communicate increasingly fine increments to the state.

### 3.7 The Resulting Model

Combining Sigma-Delta encoding from Equation 12 with the predictive encoder/decoder of Equations 13 and 14 by plugging them all into Equation 6 results in a biologically-plausible model that applies double-exponential smoothing to inputs and produces output spikes with an integrate-and-fire mechanism:

\[ v^t_j = (1 - \lambda^t)v^{t-1}_j + \lambda^t \sum_i w_{ij}q^{t-1}_i \]

\[ s^t_j = [(1 - \epsilon^t)s^{t-1}_j + \epsilon^t\rho(s^{t-1}_j) (v^t_j + b_j)]_0 \]

\[ a^t = \frac{1}{\lambda^t} (\rho(s^t) - (1 - \lambda^t)\rho(s^{t-1})) \]

\[ q^t_j = [\phi^t_j - a^t_j > \frac{1}{2}] \]

\[ \phi^t_j = \phi^{t-1}_j + a^t_j - q^t_j \]

Figure 1 shows some example dynamics from our model.

### Figure 1

- **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 postsynaptic neuron.
- **Row 3:** The step size \( \epsilon \) and predictive-coding parameter \( \lambda \) as they anneal.
- **Bottom:** A comparison of the value of the postsynaptic neuron under the continuous dynamics (Equation 3, red curve) and our binary dynamics (Equation 15, black curve).

### 4 Experiments

We explore several combinations of the hyperparameters \( \epsilon', \lambda' \), 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 simulate the two-phase learning of Equilibrium Propagation by running the network for a fixed number of steps.
Spiking Equilibrium Propagation

Figure 2: A network is presented with a constant input at \( t = 0 \), and allowed to settle. Then at \( t = 100 \) the output layer is perturbed, and the network is allowed to settle to the new fixed point. The \( y \) axis indicates the log-mean-error between the true fixed-point \( s_{\text{fixed}} \), and the current state of \( s \) of the quantized network. \( s_{\text{fixed}} \) is calculated by running a continuous-valued network (with the same parameters and input) to convergence. Note that \( s_{\text{fixed}} \) is different for \( t < 100 \) and \( t \geq 100 \) due to the external perturbation at \( t = 100 \). Shaded regions indicate the standard deviation over 20 runs with randomly initialized networks.

We also ran our model on a deeper network with 3 hidden layers [784-500-500-500-10], and found that our network slightly underperformed the continuous-valued network, achieving (3.65% test, 3.02% training) error vs (2.42% test/ 0.27% training) error for the continuous-valued network. The discrepancy is likely due to the quantized network needing longer negative/positive convergence phases. The learning curves and details on the parametrizations of these experiments can be found in Appendix C.

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] for a network with [784-500-10] units in each layer. 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.

In order to reach similar performance to the continuous-valued network, we had to extend the positive and negative phases to (100, 50) steps respectively, compared to the (20, 4) steps used in real-valued equilibrium propagation. We found no significant difference between the networks trained with OSA and with annealing schedules (after finding the optimal annealing-parameters for each model with a Gaussian-Process parameter search - see Appendix C). In the remainder of this section, we report the results for the OSA model with a constant \( \lambda = 0.275 \) predictive-coding coefficient. Our binary-network performed similarly on the test set to the continuous-valued-network (see Figure 3), achieving a (2.37% test / 0.15% training) error, compared to the continuous network’s (2.51% test / 0.25% training) error.
Another lingering biological-implausibility that is not  
we rely on “capturing” a negative state  
There are still several obstacles to doing truly biologi-
when it sees the input has started changing rapidly (e.g.
we can transfer adaptive step-size algorithms to the
2015], [George and Powell, 2006]. It is unclear whether
synapse  
an additional communication channel to synchronize
an energy minimum. Even if we could, we face the prob-
problem of Catastrophic Forgetting, wherein deep networks
tend to forget old training data when presented with a
nonsynaptic stream of training samples (though Kirk-
patrick et al. [2017] have done some work addressing
this). Finally the question of how to learn temporal se-
quencies without doing backpropagation-through-time
remains an open one, though recent works such as Ol-
livier et al. [2015] and Tallec and Ollivier [2017] have
begun to address this.

6 Conclusion

We demonstrate that we can train a network with Equi-
lbrium Propagation even when neurons are constrained
to only communicate binary values. To achieve efficient
communication between neurons, we use ideas from
Stochastic Approximation and Predictive Coding.

We believe that this work is relevant to designing of
the next generation of neural computing hardware. In
modern computers, most of the energy cost is not spent
on computation (in terms of adds and multiples), but
in moving data around Horowitz [2014]. It seems likely
that future neural computing hardware will consist of
neurons implemented as physical circuits, with com-
putation co-located with memory, so that parameter
values never need to be moved. The main energetic
bottleneck will then be the communication between
neurons. In the brain, Attwell and Laughlin [2001] esti-
mate that 81% of metabolic energy is spent on
sending signals between neurons. It makes sense then,
that neurons should have evolved to use the minimum
number of spikes to communicate what they need to
communicate. If the brain is doing something similar
to Equilibrium Propagation, then neurons compute by
collectively trying to find the fixed point of a dynam-
ical system. Our work addresses the question of how
we can efficiently find this fixed point when there is a
communication bottleneck between neurons.

Code is available at https://github.com/quva-lab/
spiking-eqprop

5 Discussion and Related Work

Much of the work in the stochastic approximation liter-
are about how to adapt step-sizes according to the
statistics of the incoming sample stream [Chau and Fu,
2015], [George and Powell, 2006]. It is unclear whether
we can transfer adaptive step-size algorithms to the
similar task of choosing optimal predictive coding co-
efficients (λ). The difficulty is that if the encoder of
neuron i has a different predictive coefficient λ than
the decoder of neuron j, the signal will not be correctly
transmitted, but the binary-communication constraint
prohibits us from directly transmitting predictive cod-

ing coefficients between neurons. It may be possible to
define an adaptive predictive coding rule where the pre-
dictive coefficients of different neurons tend to converge
so that the network reaches the correct fixed point in
the end, but this needs more work. [Bharioke and
Chiolkovskii, 2015] suggested that including a nonlinear-
ity in the predictor can have the effect of rapid online
adaptation of prediction coefficients. Adaptive predic-
tive coding could allow us to perform efficient inference
on nonsynaptic data. We can imagine constructing a
network where neurons use their bits to communicate
fine changes in state when the network is near a fixed
point but then adapt itself to take larger, coarser steps
when it sees the input has started changing rapidly (e.g.
during a saccade).

Mesnard et al. [2016] also implemented Equilibrium
Propagation with spiking neurons. Their model was
primarily built to mimic the leaky-integrate-and-fire
dynamics of biological neurons. There was no annealing
and their neurons do not converge to a fixed point when
presented with a constant input. They demonstrate
that the model can train on a toy dataset, but it is not
clear if this approach would scale to a more standard
machine learning task.

There are still several obstacles to doing truly biologi-
cally 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 (Equa-
tion 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. Δwij ∝ ρ(s1, δs2)) we
have the problem that we get very noisy updates due
to the quantization.

Another lingering biological-implausibility that is not
specific to our algorithm is that we still use symmet-
ric 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 Lilli-
crap 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 any-
way. 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 prob-
lem of Catastrophic Forgetting, wherein deep networks
tend to forget old training data when presented with a
nonsynaptic stream of training samples (though Kirk-
patrick et al. [2017] have done some work addressing
this). Finally the question of how to learn temporal se-
quencies without doing backpropagation-through-time
remains an open one, though recent works such as Ol-
livier et al. [2015] and Tallec and Ollivier [2017] have
begun to address this.
Acknowledgements

This work was funded by Qualcomm.

References


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