Training Dynamical Binary Neural Networks with Equilibrium Propagation

Jérémie Laydevant[1*], Maxence Ernoult[2†], Damien Querlioz[3], Julie Grollier[1]
[1] Unité mixte de Physique, CNRS-Thales, Université Paris-Saclay, Palaiseau, France
[2] MILA, Montreal, Canada
[3] Université Paris-Saclay, CNRS, Centre de Nanosciences et de Nanotechnologies, Palaiseau, France
{[*]jeremie.laydevant@cnrs-thales.fr; [†]ernoultm@mila.quebec}

Abstract

Equilibrium Propagation (EP) is an algorithm intrinsically adapted to the training of physical networks, thanks to the local updates of weights given by the internal dynamics of the system. However, the construction of such a hardware requires to make the algorithm compatible with existing neuromorphic CMOS technologies, which generally exploit digital communication between neurons and offer a limited amount of local memory. In this work, we demonstrate that EP can train dynamical networks with binary activations and weights. We first train systems with binary weights and full-precision activations, achieving an accuracy equivalent to that of full-precision models trained by standard EP on MNIST, and losing only 1.9% accuracy on CIFAR-10 with equal architecture. We then extend our method to the training of models with binary activations and weights on MNIST, achieving an accuracy within 1% of the full-precision reference for fully connected architectures and reaching the full-precision accuracy for convolutional architectures. Our extension of EP to binary networks opens new solutions for on-chip learning and provides a compact framework for training BNNs end-to-end with the same circuitry as for inference.

1. Introduction

Conventional deep learning models, trained with error backpropagation (BP), have demonstrated outstanding performance at multiple cognitive tasks. But the training process is so energy consuming [32, 33] that it questions the environmental sustainability of AI deployment [21]. These artificial neural networks are actually trained on un-optimized von Neumann hardware with a delocalized memory, such as GPUs or TPUs. Furthermore, they struggle to fully benefit from a hardware which provides local memory access at neuron level as their learning rule, backpropagation, is fundamentally non-local.

Equilibrium Propagation (EP) [30] is a learning framework that leverages the dynamics of energy-based physical systems fed by static inputs to compute weight updates with a learning rule local in space [8] which can also be made local in time [9], and in addition scales to CIFAR-10 [20]. Today, EP is developed on standard hardware (GPU) that 1) does not provide for the low power and the computing efficiency a dedicated hardware implementation might exhibit [34, 23] and 2) prevents EP to scale to large scale datasets such as ImageNet due to the duration of simulations. An EP-dedicated hardware would reduce the energy consumption of training by two orders of magnitude compared to GPUs and accelerate training by several orders of magnitude [24], while being competitive on large scale benchmarks in terms of accuracy since the gradients estimates prescribed by EP are equivalent to those given by BPTT [8].

The main asset of EP is the ability of on-chip learning, especially when the memory and the computational budgets dedicated to training and inference are constrained (e.g. embedded environments). EP also naturally suits for training physical systems intrinsically dynamical whose dynamics are unknown and hardly derivable [19, 36, 24]. EP therefore appears as a solution for on-chip training for embedded systems and dynamical hardware, two cases with which BP is not compatible without major adaptation.

EP is however based on full-precision (64 bits floating point) weights and activations that do not match the current requirements of such hardware systems. Full-precision weights overload the memory capacity of chips when they are stored digitally [11, 34], and are prone to noise and hard to read when stored with emerging synaptic nano-devices [2, 35, 14, 15]. Moreover, analog activation functions are not directly compatible with widely-used digital communications between neurons [34].

In this paper, we address the issue of on-chip learning via EP by training dynamical systems having binary activations and weights.We first leverage the recent progress made in Binary Neural Networks (BNNs) optimization [13], to binarize the synapses in energy-based models trained by EP. The optimization of weights is performed using the inertia of the gradient. This lowers the memory required for training such systems compared to real-valued ("latent") weights optimization, traditionally used for training BNNs. We then binarize the activation functions, yielding an easy way to compute the local gradient while supporting a digital communication between neurons. More precisely, our contributions are:

- We introduce a version of EP that can learn recurrent binary weights assuming full-precision activations (Fig. 1a). For simplicity, we call this version of EP "binarized EP". Our implementation uses a novel weight normalization scheme directly learnable by EP. We are able to maintain an accuracy similar to full-precision models on fully connected and convolutional architectures on MNIST. We extend these results to the CIFAR-10 task, with performance only degraded by 1.9 % from that achieved with the full-precision counterpart trained by EP [20].
- We extend our technique to fully binarized dynamical networks where both weights and neuron activations are binarized (Fig. 1b). We demonstrate successful training on fully connected and convolutional architectures on MNIST with a slight degradation (between 0.2 and 1%) with respect to standard EP. This "fully binarized" version of EP achieves binary communication between neurons while reducing the memory required to compute the local gradient to 1 bit, making the gradient ternary.
- Our code is available at: `https://github.com/jlaydevant/Binary-Equilibrium-Propagation`.

2. Background

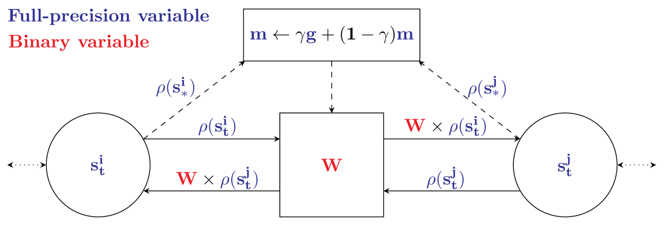
Energy-based models. As emphasized above, our work focuses on dynamical energy-based neural networks as opposed to purely feedforward models. More precisely, denoting s the state of the neurons at a given time, $\rho(s)$ the activation function of the neurons, x an input and $\theta = \{W, b\}$ the parameters of the model, we assume dynamics of the form:

$$\frac{ds}{dt} = -\frac{\partial E}{\partial s}(x, s, \rho(s), \theta), \tag{1}$$

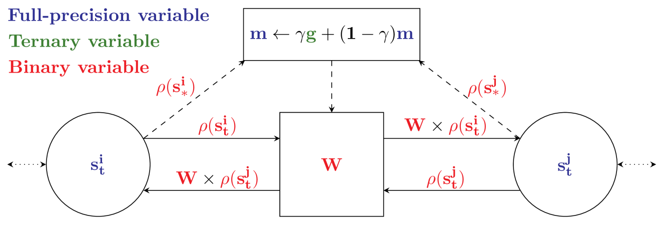
where $E(x, s, \rho(s), \theta)$ denotes an energy function describing the system of interest. Given x and θ, the system evolves according to Eq. (1) until reaching a steady state s_* which minimizes the energy function: this constitutes the first phase. Given a target y for the output layer of the system, the learning objective is to optimize the synaptic weights θ to minimize the loss:

$$\mathcal{L}_* = \ell(s_*, y) \tag{2}$$

where ℓ denotes a cost function that outlines the discrepancy between s_* and y. After learning, the system evolves to steady states of minimal prediction error.


Full-precision variable
Binary variable
$\mathbf{m} \leftarrow \gamma\mathbf{g} + (1-\gamma)\mathbf{m}$
$\rho(\mathbf{s}_*^{\mathbf{i}})$
$\rho(\mathbf{s}_*^{\mathbf{j}})$
$\rho(\mathbf{s}_t^{\mathbf{i}})$
$\mathbf{W} \times \rho(\mathbf{s}_t^{\mathbf{i}})$
$\mathbf{s}_t^{\mathbf{i}}$
$\mathbf{W}$
$\mathbf{s}_t^{\mathbf{j}}$
$\mathbf{W} \times \rho(\mathbf{s}_t^{\mathbf{j}})$
$\rho(\mathbf{s}_t^{\mathbf{j}})$

(a) Binary weights (Section 3)


Full-precision variable
Ternary variable
Binary variable
$\mathbf{m} \leftarrow \gamma\mathbf{g} + (1-\gamma)\mathbf{m}$
$\rho(\mathbf{s}_*^{\mathbf{i}})$
$\rho(\mathbf{s}_*^{\mathbf{j}})$
$\rho(\mathbf{s}_t^{\mathbf{i}})$
$\mathbf{W} \times \rho(\mathbf{s}_t^{\mathbf{i}})$
$\mathbf{s}_t^{\mathbf{i}}$
$\mathbf{W}$
$\mathbf{s}_t^{\mathbf{j}}$
$\mathbf{W} \times \rho(\mathbf{s}_t^{\mathbf{j}})$
$\rho(\mathbf{s}_t^{\mathbf{j}})$

(b) Binary weights and activations (Section 4)

Figure 1: Building blocks of binarized EP: two neurons communicate bidirectionally through a binary synapse. The color code highlights the precision of each variable: the synaptic weight (bold red) is binary and the internal state of the neurons (s_t^k), as well as the momentum (m) averaging the gradient (bold blue), are full-precision. The activations ($\rho(s_t^k)$) and the equilibrium activations ($\rho(s_*^k)$) are full-precision variables (blue) in Section 3 but binary (red) in Section 4. The gradient estimate (g) prescribed by EP (bold blue) is a full-precision variable in Section 3 but is ternary (bold green) in Section 4. Every neuron also has a bias which is full-precision and does not appear on the figure for clarity.

Equilibrium Propagation (EP). While the learning objective could be optimized by backpropagating the prediction error backward in time (BPTT), EP instead proceeds with a second phase where the dynamics of Eq. (1) is changed into:

$$\frac{ds}{dt} = -\frac{\partial E}{\partial s} - \beta\frac{\partial \ell}{\partial s}, \tag{3}$$

where β denotes a scalar nudging parameter. In this way, the system evolves along Eq. (3) towards decreasing the cost function ℓ until reaching a second steady state s_*^β. In their foundational paper, Scellier & Bengio [30] proved that $\mathcal{L}_*$ could be minimized using the local gradient estimate:

$$\mathbf{g}_\theta = \frac{1}{\beta}\left(\frac{\partial E}{\partial \theta}(x, s_*^\beta, \theta) - \frac{\partial E}{\partial \theta}(x, s_*, \theta)\right), \tag{4}$$

which typically translates, for the weights of a fully connected layer, to [30]:

$$\Delta W_{ij} = \frac{1}{\beta}\left(\rho(s_{i,*}^\beta)\rho(s_{j,*}^\beta) - \rho(s_{i,*})\rho(s_{j,*})\right), \tag{5}$$

where ρ denotes an activation function. EP has extremely attractive features for neuromorphic chip design: the same dynamics sustain both inference (Eq. (1)) and error propagation (Eq. (3)), and the learning rule is local (Eq. (5)).

Binary Neural Networks (BNNs). BNNs were first introduced by Courbariaux *et al.* [6] to reduce the memory footprint and the cost of operations in feedforward neural networks at inference time, later scaled to hard visual tasks [29]. In BNNs, the weights and activations are constrained to the binary values $\{-1, +1\}$. During BNN training, each binary weight is paired with a full-precision "latent" weight which undergoes weight updates. Binary weights are taken equal to the sign of the latent weights and are used for the forward and the backward passes. After training, the latent weights are discarded.

Binary Optimizer without latent weights (BOP). Although latent weights in BNNs accumulate weight updates, Helwegen *et al.* [13] suggested that they were not weights in the strictest sense (they are not used at run time) but were only meant to convey inertia for the optimization of the binary weights. Based on this insight, Helwegen *et al.* [13] proposed a Binary Optimizer (BOP) which flips the binary weights solely based on the value of their associated momentum (without latent weights *per se*): if the momentum is large enough and crosses a threshold from below, the binary weight is switched. By using one full precision variable instead of two per synapse, BOP is of definite interest to reduce the memory footprint of BNN training, which is why our work heavily relies on this technique (see Section 3). BOP has two hyperparameters: the value of the flipping decision threshold τ, and the adaptativity rate γ. The larger τ, the less frequent the binary weight flips and the slower the learning. On the other hand the larger γ, the more sensitive the momentum to a new gradient signal, the more likely a binary weight flips. The BOP algorithm is summarized in Alg. 1.

Algorithm 1 BOP [13].

```
Input: $g$, $m$, $\boldsymbol{W}$, $\gamma$, $\tau$.
Output: $m$, $\boldsymbol{W}$.
  $m \leftarrow \gamma g + (1-\gamma) m$
  for $i \in [1, d]$ do
    if $|m_i| > \tau$ and sign$(m_i)$ = sign$(\boldsymbol{W_i})$ then
      $\boldsymbol{W_i} \leftarrow -\boldsymbol{W_i}$
    end if
  end for
```

Related work. Spiking neural networks (SNNs) are models that compress the communication between neurons to one bit. They are thus compatible with digital and energy efficient hardware [25, 10, 1, 7]. Most existing hardware implementations of SNNs on neuromorphic platforms use Spike Timing Dependent Plasticity (STDP) as a learning rule [28]. Despite its low accuracy on complex tasks, the locality of STDP indeed enables compact circuits for on-chip training. This shows the importance of making EP compatible with digital hardware: its local learning rule can be implemented with compact circuits and the accuracy greatly improved compared to STDP as EP optimizes a global objective junction.

Other studies have investigated how much synapses and (or) neural activations of energy-based models could be compressed when trained by EP. Mesnard *et al.* [26], O'Connor *et al.* [27] and Martin *et al.* [24] showed that EP can train networks where neighboring neurons communicate with spikes only. However all these techniques require full precision weights, as well as an analysis of the spike trains in order to determine the firing rates giving the gradients and are only demonstrated on MNIST or non-linear toy problems.

Ji & Gross [18] have studied the effect of weight and gradient quantization of an energy-based model trained by EP, showing that at least 12 or 14 bits are required to achieve less than 10% test error on MNIST. Here we show that the weights (at all time) and the neural activations (at read time) can be compressed down to 1 bit only, yielding ternary gradients (at read time) and binary communication between neurons in the system. We discuss how the full-precision pre-activations and accumulated weight momentum can be handled in a neuromorphic chip. Finally, our work is the first to demonstrate energy-based model compression with EP on CIFAR-10.

3. EP Learning of Recurrent Binary Weights with Full Precision Neural Activations

In this section, we show that we can train dynamical systems with binary weights and full precision activations by EP with a performance on MNIST and CIFAR-10 close to the one achieved by their full-precision counterparts trained by EP [8, 20]. Our technique relies on the combined use of BOP described in Alg. 1 and of a proper weight normalization to avoid vanishing gradients. Therefore, we first describe how EP can be embedded into BOP (Alg. 2). Then, we propose two weight normalization schemes: one with a fixed scaling factor taken from [29], another one with a dynamical scaling factor directly learned by EP. We show that the use of the learnt weight normalization, which naturally fits into the EP framework, considerably improves model fitting and training speed on MNIST and CIFAR-10.

3.1. Feeding EP weight updates into BOP

Working principle. We explain here how to use BOP to optimize the binary synapses given the gradient computed with EP. At each training iteration, the first steps of our technique are the same as standard EP: the first phase and the second phase are performed as usual and the EP gradient estimate g is obtained from the steady states s_* and s_*^β for each synaptic weight. Thereafter, g is directly fed into the

BOP algorithm (Alg. 1): for each synapse connecting neuron j to neuron i, the EP gradient estimate g_{ij} conveys inertia to the synaptic momentum m_{ij}, and the binary synaptic weight W_{ij} is flipped or not, depending on the value of m_{ij}. Finally, as usual in BNNs [6], the biases are full-precision and are updated with standard Stochastic Gradient Descent (SGD). We summarize all those steps in Alg. 2, where we have highlighted binarized variables in bold red for clarity. With this procedure, we have a system in which the synapses are binarized at all time. In this section we use a full-precision activation function for the neurons, the hardsigmoid, and full-precision gradients. The binarization of activation functions and ternarization of gradients is addressed in Section 4.

Algorithm 2 EP learning of dynamical binary weights (with simplified notations). Binarized variables are in bold red. When the neural pre-activations are binarized (Section 4), the EP gradient estimate g is ternarized (in bold green), otherwise full precision (Section 3).

```
Input: x, y, $s$, $\beta$, $\theta = \{\mathbf{W}, b\}$, $\eta$, $m$, $\gamma$, $\tau$.
Output: $\theta = \{\mathbf{W}, b\}$, $m$.
  Free phase:
  for $t \in [1, T]$ do
    $s \leftarrow s - dt \times \frac{\partial E(\mathrm{x}, s, \mathbf{W}, b)}{\partial s}$
  end for
  $s_* \leftarrow s$
  Nudged phase:
  for $t \in [1, K]$ do
    $s \leftarrow s - dt \times \frac{\partial E(\mathrm{x}, s, \mathbf{W}, b)}{\partial s} - \beta \times \frac{\partial \ell(\mathrm{y}, y)}{\partial s}$
  end for
  $s_*^\beta \leftarrow s$
  Compute EP gradient with $s_*^\beta$ and $s_*$:
    $\mathbf{g} \leftarrow -\frac{1}{\beta}\left(\frac{\partial E}{\partial \theta}(\mathrm{x}, s_*^\beta, \mathbf{W}, b) - \frac{\partial E}{\partial \theta}(\mathrm{x}, s_*, \mathbf{W}, b)\right)$
  Apply BOP (Alg. (1)):
    $m$, $\mathbf{W}$ = BOP$(g, m, \mathbf{W}, \gamma, \tau)$
  Update biases with SGD:
    $b \leftarrow b + \eta \times g$
```

Hyperparameter tuning. Similarly to [13], we monitor the number of weight flips per epoch and layer-wise in order to tune the hyperparameters of BOP, using the metric:

$$\pi_{\text{epoch}}^{layer} = \log\left(\frac{\text{Number of flipped weights}}{\text{Total number of weights}} + e^{-9}\right) \qquad (6)$$

Heuristically, $\pi_{\text{epoch}}^{layer}$ reflects a trade-off between learning speed (high $\pi_{\text{epoch}}^{layer}$) and stability (low $\pi_{\text{epoch}}^{layer}$). We measure $\pi_{\text{epoch}}^{layer}$ in the regions of γ and τ where learning performs well, and use this value of $\pi_{\text{epoch}}^{layer}$ in return as a criterion to tune γ and τ on new models.

3.2. Normalizing the Binary Weights with a fixed scaling factor

When binarizing synaptic weights to ± 1, neural activities may easily saturate to regions of flat activation function, resulting in vanishing gradients. It is especially true with the hardsigmoid activation function often used with EP. Batch-Normalization [17] used by Courbariaux *et al.* [6] and Hubara *et al.* [16] helps with this issue by recentering and renormalizing activations by computing the batch statistics. Batch-Normalization has been introduced in recurrent neural networks such as LSTMs to process sequence tasks [22] but it does not translate directly to energy-based models. The normalization scheme should indeed itself derive from an energy function in order to be learnable, which restricts the choice of candidate normalizations. However, the goal in convergent dynamical systems processing static inputs is not to center neural activations at every time step, but rather at their steady state. Moreover, using batch-based weight normalization schemes is far from straightforward from a hardware prospective. For this purpose, we first normalize the binary weights with a static scaling factor.

Static XNOR-Net weight scaling factor. In the design of their XNOR-Net model, Rastegari *et al.* [29] introduced a scaling factor to minimize the difference between the binary synapses and the corresponding set of full-precision "latent" weights at each layer. This scaling factor is updated at each training iteration and depends on the size of two adjacent layers and on the magnitude of these latent weights. The scaling factor reads in our context:

$$\alpha_{n,n+1} = \frac{||w_{n,n+1}^{\text{init}}||_1}{\dim(w_{n,n+1}^{\text{init}})} \qquad (7)$$

where n is the index of a layer, $w_{n,n+1}^{\text{init}}$ are the full-precision random weights used to initialize the binary weights. Using this scaling factor, we initialize each binary weights, layer by layer, as $W_{n,n+1} = \pm\alpha_{n,n+1}$. Contrarily to XNOR-Net where the scaling factor is updated at each forward pass, we first keep the scaling factor fixed to its initial value throughout training. We show in Section D that this scaling factor is crucial to train recurrent binary weights by EP.

Results. We investigate fully connected architectures (with one and two hidden layers) on MNIST and convolutional architectures (with two and four convolutional layers) on the MNIST and CIFAR-10 datasets. We employ prototypical models to speed up training as in [8]. Our results (Table 1 - "EP - Binary Synapses") are benchmarked against those of full precision models (Table 1 - "EP - Benchmark") and those obtained by BPTT+BOP (Table 1 - "BPTT - Binary Synapses"). Note that for a given architecture, the number

Table 1: Error of EP and BPTT on networks having binary synapses with a fixed or a dynamical scaling factor - Results are reported as the mean over 5 trials $\pm$ 1 standard deviation - Benchmark performances are taken from [8, 20]

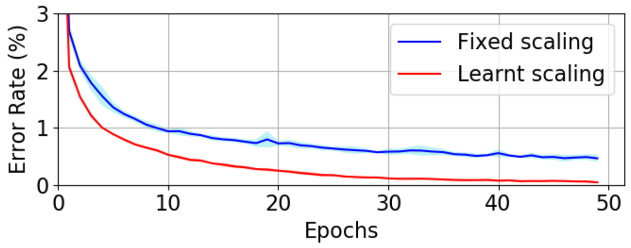
		EP - Binary Synapses				**EP**	**BPTT**
		Fixed α		Dynamical α		Benchmark	Binary Synapses
Dataset	Model	Test	Train	Test	Train	Test	Test
MNIST	(1fc)	2.07 (0.02)	0.77	**1.7 (0.04)**	**0**	2.00	2.14 (0.06)
MNIST	(2fc)	2.48 (0.08)	0.29	**2.28 (0.13)**	**0**	1.95	2.38 (0.07)
MNIST	(conv)	0.85(0.11)	0.46	$\mathbf{0.88}(\mathbf{0.06})$	$\mathbf{0.05}$	1.05	0.97 (0.03)
CIFAR-10	(conv)	16.8(0.3)	6.9	$\mathbf{15.66}(\mathbf{0.28})$	$\mathbf{5.54}$	13.78	14.45 (0.12)

of neurons we used per layer may not be the same as in reference architectures – see 3.4 and Appendix F for details.

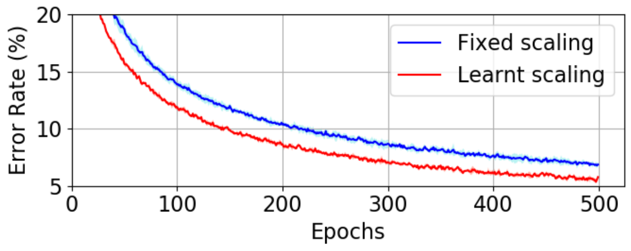
Overall, Table 1 shows that the normalization of weights with a fixed scaling factor allows EP with binary synapses to perform comparably to full-precision models trained across different fully connected and convolutional architectures, on MNIST and CIFAR-10. The fully connected architecture which has one hidden layer trained on MNIST shows no statistically significant loss of performance compared to full-precision counterpart trained by EP. This architecture with binary synapses reaches the same accuracy if trained by (EP+BOP) or by (BPTT+BOP). The fully connected architecture having two hidden layers trained on MNIST with fixed scaling factors shows 0.5% performance degradation compared to full-precision models trained by EP. For the convolutional architecture trained on MNIST, we can even observe a slightly better training and testing (-0.2%) errors on model with binary synapses compared to full precision models trained by EP. We explain this improvement by the cumulative use of the randomization of β and of the regularization effect induced by the binarized architecture itself [6]. Furthemore, the training framework (EP+BOP) achieves a similar accuracy as the framework (BPTT+BOP). Finally, the performance of our convolutional model trained on CIFAR-10 is $\sim 3\%$ less than the one of Laborieux *et al.* [20], using the same architecture. Also, the network trained by (EP+BOP) shows only 2.5% degradation of the accuracy compared to the same network trained by (BPTT+BOP).

3.3. Normalizing the Binary Weights with a learnt scaling factor

Dynamical weight scaling factor learned by EP. Using fixed scaling factors gives high, yet sub-optimal accuracies (see Fig. 4 in the Appendix). Bulat & Tzimiropoulos [4] show that the scaling factor can be learnt by backpropagation to extend XNOR-Nets. Here we derive a learning rule for the scaling factor with the help of the theorem of Scellier & Bengio [30] to ensure that it provides a gradient estimate of the loss $\mathcal{L}_*$ defined in Eq. (2). The reasoning to derive this learning rule is the following. We split the binary weights in two parts: $W_{n,n+1} \leftarrow \alpha_{n,n+1} \times w^{bin}_{n,n+1}$ where $w^{bin}_{n,n+1}$



(a) MNIST



(b) CIFAR-10

Figure 2: Average training error as a function of the number of epochs for a convolutional architecture with binary synapses trained on MNIST with a static (blue curve) or a dynamical (red curve) scaling factor. Curves averaged over 5 trials ± 1 standard deviation.

are the binary weights scaled to ± 1 and the scaling factors $\alpha_{n,n+1}$ are initialized as when they are fixed. The resulting dynamics still derives from an energy function, and one can derive a learning rule for $\alpha_{n,n+1}$ which reads as:

$$\Delta\alpha_{n,n+1}(\beta) = -\frac{1}{\beta}\left(\frac{\partial E}{\partial \alpha_{n,n+1}}(s^{\beta}_{*}) - \frac{\partial E}{\partial \alpha_{n,n+1}}(s_{*})\right), \quad (8)$$

so that $\alpha_{n,n+1}$ is learned like any other network parameter, and $\lim_{\beta\to 0}\Delta\alpha_{n,n+1}(\beta) = -\frac{\partial \mathcal{L}_*}{\partial \alpha_{n,n+1}}$. In Section D the learning rules for fully connected and convolutional architectures are derived in all the settings of EP.

Results. Fig. 2 illustrates on a convolutional architecture the gain in performance obtained by learning the scaling factor. Globally, this technique systematically results in faster learning and better model fitting across all the models, and almost always in better generalization as observed in Table 1. Learning the scaling factor by EP is thus a powerful alternative to Batch-normalization in convergent dynamical systems as highlighted by Fig. 2.

3.4. Hardware implementation

Memory gain at run time. As often observed in binarized architectures [6, 16], we achieve accuracy similar to the one of full-precision models at the price of having 8 times more hidden neurons in fully connected architectures (see Fig. 8 of Appendix F.1 for a more detailed analysis). In convolutional architectures, we have used at most the same number of output feature maps than their full precision counterparts for computational efficiency. After training, our models use 2 and 7.5 less memory for the synapses for fully connected architectures on MNIST (for two and one hidden layers respectively), 9 and 54 less for the convolutional architectures used on MNIST and CIFAR-10 respectively. In hardware, these binary weights can be stored in digital memories [34] or using nanoscale memristors [15].

Memory requirements at train time. The memory requirements for training should be subject to a more careful treatment. Inertia-based optimization (BOP) requires a single full-precision variable: the momentum, compared to the latent-weight counterpart often trained by elaborate optimization techniques such as (SGD + Momentum) which uses at least 2 full-precision variables: the latent weight and the momentum. Inertia-based optimization thus reduces by at least a factor 2 the required memory for training. Furthermore, the memory required for storing the momentum of BOP could be implemented by non-standard memories. In fact, the discrete time update of the momentum (Alg. 1) can be rewritten into a continuous time update rule which reads:

$$\frac{dm}{dt} + \gamma \cdot m(t) = \gamma \cdot g(t) \tag{9}$$

which naturally appears as the differential equation describing the evolution of the voltage of a capacitor. Capacitors are CMOS-compatible, and highly linear which makes them well suited for storing full-precision variables [2]. They can thus be used to store the inertia, thereby lowering the memory requirement to one capacitor per binary weight and more globally lowering the memory required for training.

4. EP Learning of Recurrent Binary Weights with Binary Neural Activations

The techniques presented in the previous section use full-precision neural activations. However, it is highly preferable to rely on binary activation values in hardware. Binary read and write errors can indeed be accommodated without too much circuit overhead in neuromorphic systems [14] and binary values are easier to pass between spatially distant hardware neurons [34]. In this section, we show that we can train dynamical system with binary weights and binary activations by EP, resulting in a performance on MNIST again approaching full-precision models on fully connected and convolutional architectures. Our implementation relies on two main components: the choice of a proper activation function to binarize neural pre-activations, and output layer augmentation. Combining these two techniques, we can design dynamical systems which are sensitive to error signals despite threshold effects and can compute ternary gradients in return. The corresponding pseudo-algorithm is the same as Alg. 2, except that the gradient estimate g is now ternarized.

4.1. Convergent neural networks with binary activations.

Ternarizing EP gradients. We can note from Eq. (5) that the precision of the gradient g estimate provided by EP is typically determined by the choice of the activation function ρ. For instance, if ρ outputs binary values $\{0, 1\}$, we immediately see from Eq. (5) that, for the parameters $\theta = \{W, b\}$, the gradients has values:

$$\mathbf{g}_\theta \in \left\{-\frac{2}{\beta}, 0, \frac{2}{\beta}\right\}. \tag{10}$$

In practice $\beta = 2$ works well, resulting in $\times 40$ gradient compression compared to 64-floating point resolution.

However, binarizing neural activations comes at several costs for the dynamics of the neurons. The energy function of the system subsequently outputs a semi-discrete variable which affects the dynamics of each neuron non trivially: if the updates of neuron activations are simultaneous, the dynamics of the system may not converge [5]. In particular, this precludes the use of prototypical models [8], that can be employed to speed up training as we did in the previous section. Therefore, we must use standard energy-based models as in [30] so that binary activations are updated only when the full-precision pre-activations reach the threshold of the activation function, thus non simultaneously. We next detail some empirical properties the binary activation of the neurons should have in order to define convergent dynamics.

Binarizing neural activations into $\{0, 1\}$. While we binarize weights to opposite signs, we found that using the sign activation function to binarize neural activations into $\{-1, 1\}$, as usually done with BNNs to implement MAC operations with XNOR gates, entails non-convergent dynamics. This confirms previous findings on EP which emphasized the importance of bounding the neural activations between 0 and 1 to help dynamics convergence using the hardsigmoid activation function $\rho(s) = \max(0, \min(s, 1))$ [30]. Therefore, our proposal here is to use the Heavyside step function shifted by 0.5:

$$\rho(s) = \mathrm{H}\left(s - \frac{1}{2}\right) \tag{11}$$

where $H(x) = 1$ if $x \geq 0$, 0 otherwise. However, the energy based dynamics of our models requires to gate $\frac{\partial E}{\partial \rho}$ by the derivative of ρ as Eq. 1 rewrites as:

$$\frac{ds}{dt} = -\frac{\partial E}{\partial s}(x, s, \theta) - \frac{\partial \rho(s)}{\partial s}\frac{\partial E}{\partial \rho(s)}(x, \rho(s), \theta). \tag{12}$$

Noting that Eq. 11 is obtained by asymptotically sharpening the narrowed hardsigmoid around $\frac{1}{2}$ within $[\frac{1}{2} - \sigma, \frac{1}{2} + \sigma]$ denoted $\hat{\rho}$, namely:

$$\rho(s) = \lim_{\sigma \to 0} \hat{\rho}(s, \sigma) = \mathrm{H}\left(s - \frac{1}{2}\right) \tag{13}$$

we propose to substitute the derivative of ρ as:

$$\frac{\partial \hat{\rho}(s, \sigma)}{\partial s} \approx \frac{\partial \boldsymbol{\rho}(s)}{\partial s} = \begin{cases} \frac{1}{2\sigma} & \text{if } \left|s - \frac{1}{2}\right| \leq \sigma \\ 0 & \text{else} \end{cases} \tag{14}$$

where σ is a parameter discussed in Appendix F. Therefore, denoting $\hat{\rho}' = \partial_s \hat{\rho}$ for simplicity, the free dynamics of s can be approximated as:

$$\frac{ds}{dt} \approx -\frac{\partial E}{\partial s} - \hat{\rho}'(s)\frac{\partial E}{\partial \rho} \tag{15}$$

4.2. Augmenting the Error Signal to Nudge Neurons with Binary Activations

Binarization of activations can prevent the propagation of errors. As the system sits at rest at the end of the first phase of EP, upon nudging the output layer by the prediction error, the motion of the system during the second phase of EP encodes error signals [31, 8]. Therefore during the second phase, a given neuron i needs to have its activation function change from $\rho(s_{*,i})$ to a distinct $\rho(s^{\beta}_{*,i})$ to compute the error gradient locally and transmit it backward to upstream layers. However, when using a discontinuous activation function like defined in Eq. (13), we may have $\rho(s_{*,i}) = \rho(s^{\beta}_{*,i})$ if the pre-activation s_i of the neuron moves less than the value of the activation threshold of ρ, thus zeroing the error signal, or equivalently vanishing gradients. Consequently, we need to ensure that for a sufficient number of neurons i:

$$\Delta s_i = |s^{\beta}_{*,i} - s_{*,i}| > \frac{1}{2} \tag{16}$$

In order to satisfy Eq. (16) for a sufficient number of neurons, we propose to increase the error signal by augmenting the output layer so that each prediction neuron is replaced by N_{perclass} neurons per class, inflating the output layer from N_{classes} to $N_{\text{classes}} \times N_{\text{perclass}}$. We choose N_{perclass} in such a way that the number of output neurons matches approximately the number of neurons in the penultimate hidden layer: $N_{\text{perclasses}} \approx \frac{N_{\text{penultimate}}}{N_{\text{classes}}}$. In this way, the output layer delivers a large and redundant initial error signal that can push neurons beyond the activation threshold of ρ and propagate across the whole architecture. Our solution is reminiscent of the the use of auxiliary output neurons in [3], albeit with a very different motivation.

4.3. Results

We investigate here fully connected (1 and 2 hidden layers) and convolutional architectures on MNIST. The first layer receives full-precision inputs from the input layer and binary inputs from the next layer. For a given architecture, the number of neurons used per layer is different for both situations: for the fully connected architectures we use 8192 neurons per hidden layer and the two convolutional layers of the convolutional architecture have respectively 256 and 512 channels - see Appendix F for more details. We use a randomized sign for β as prescribed by Laborieux *et al.* [20] to improve the gradient estimate given by EP for all simulations except for the fully connected architecture with two hidden layers where we only use $\beta > 0$.

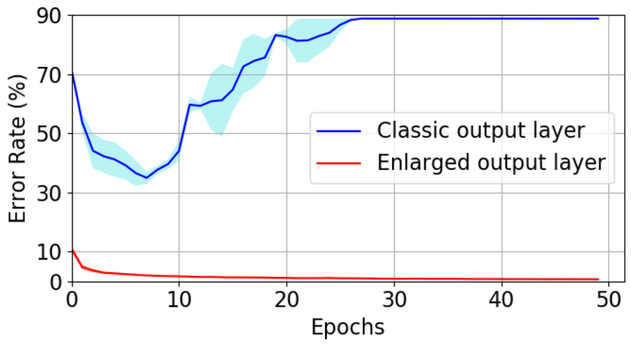


Figure 3: Average training error as a function of the number of epochs for a convolutional architecture with binary synapses and binary activations trained on MNIST with a classic output layer (10 output neurons - blue curve) or an enlarged output layer (700 output neurons - red curve). Blue curves are averaged over 2 trials ± 1 standard deviation - Red curves are averaged over 5 trials ± 1 standard deviation.

Fig 3 shows for the convolutional architecture a trend observed for all models: when using 10 neurons in the output layer, training fails (blue curve) while it succeeds upon augmenting the output layer. It is here augmented by a factor 70 (red curve) which is required for the number of neurons in the output layer to match the number of input neurons that the last convolutional layer receives from the penultimate convolutional layer: we multiply the number of channels in the penultimate convolutional layer (256) by the kernel size (5×5) and divide the result by the max pooling kernel size (3×3) which gives ~ 700 output neurons).

Table 2: Error achieved by EP with binary synapses & activations, and fixed scaling factors α - Results are reported as the mean over 5 trials $\pm$ 1 standard deviation.

		Fully binarized EP	
Dataset	Model	Test	Train
MNIST	(1fc)	2.83 (0.06)	0.2
MNIST	(2fc)	3.03(0.03)	0.84(0.17)
MNIST	conv	1.14(0.08)	0.67(0.04)

Performance. The results obtained on MNIST with fixed scaling factors are summarized in Table 2. On the fully connected architectures, the accuracy approaches those obtained with binary synapses and full-precision activations, with a slight degradation of 0.8% for one hidden layer and 0.6% for two hidden layers (see Table 1). The degradation is slightly enhanced when we compare with the full-precision counterpart trained by EP where the performance is degraded by 0.8% for one hidden layer but 1% for two hidden layers. We account the degraded performance of the architecture which has two hidden layers by the fact that we use $\beta > 0$ which makes the estimation of the gradient less accurate than when estimated with the sign of β random. We used $\beta > 0$ because we found that reaching the second equilibrium point with $\beta < 0$ is possible but very long to get in practice with a classic nudge. For the convolutional architecture trained on MNIST, we also report a performance only 0.2% below the system which has binary synapses and full-precision activations but within the error bars of the one achieved by full-precision models as reported by [8]. We think that optimizing the nudging strategy could improve the error obtained with two or more hidden layers and will be key in the future for scaling to CIFAR-10.

4.4. Gains for hardware

When binarizing the activation in addition to the synapses, we had to increase the number of neurons in each layer compared to full-precision models: by 16 for the fully connected layers resulting in 8192 neurons per hidden layer and by 8 for the convolutional architecture which has 256 and 512 channels per respective layer, to get accuracy approaching reported results with full-precision architectures. But considerable gains in terms of memory and computing are achieved due to the way the gradient is computed.

The gradient estimate $\mathbf{g_{ij}}$ is indeed now ternary (Eq. 10), and can be easily computed with the subtraction of 2 AND operations. With the notations of Eq. 5 it decomposes as: one AND operation between $s_{i,*}$ and $s_{j,*}$ and another one between $s^{\beta}_{i,*}$ and $s^{\beta}_{j,*}$, which amounts to only 5 elementary operations including the subtraction. In terms of memory, neurons only have to store 1 bit as the first equilibrium state. That way, the communication between neurons is not only binarized, but in addition, compared to previous works on EP achieving binary communication through spikes [26, 27, 24], our method drastically reduces the memory to compute the gradient. Indeed, spikes need to be stored for several time steps to get an estimation of the firing rate of each neuron, resulting in heavy memory requirements.

The computation of the MAC operation is also simple. It cannot be obtained as in standard BNNs with a single XNOR gate and popcount because this operation does not match our choice of binary activations (0/1) and binary weights (-1/1). However, it only requires the subtraction of the popcount of 2 AND gates, using simpler logical gates, and only doubling their total number compared to usual BNNs.

Despite the fact that we need to enlarge the output layer depending on the architecture, we show in Appendix E.3 that probing the state of one single neuron per class in the output layer is sufficient to obtain almost the same accuracy than when measuring the states of all the output neurons, which is beneficial for lowering the energy consumption of hardware (Figs. 6-7).

In addition, contrarily to BP performed in conventional BNNs where the input of each layer is stored between the forward and the backward passes in order to compute the full-precision gradient, here we only need to store the 1 bit activation after each phase in order to compute the gradient, which drastically reduces the memory requirements of the model for training by a factor 40. The current implementation of binary EP on GPUs, however, still relies on full-precision neuron state $s(t)$ variables. In future implementation of binarized EP on dedicated hardware, these neuron states can be implicitly encoded through the dynamics of nano-devices, thus solving this issue [2].

5. Conclusion

As a conclusion, we provide here a binarized version of EP that exhibits only a slight degradation of accuracy compared to full-precision models. This version of EP offers the possibility of training on-chip BNNs with compact circuitry because the hardware required for training is the same as for inference, whereas current BNNs are trained on conventional hardware, before being transferred to compact, low-energy chips. Finally, the version of EP with binary synapses and full precision activations is of major interest for future fast, low power hardware built on emerging devices. Joint development of EP and hardware will be critical for adapting EP to larger data sets.

6. Acknowledgments

This work was supported by the European Research Council ERC under Grant No. bioSPINspired 682955 and Grant No. NANOINFER 715872.

References

[1] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(10):1537–1557, 2015. 3

[2] Stefano Ambrogio, Pritish Narayanan, Hsinyu Tsai, Robert Shelby, Irem Boybat, Carmelo Nolfo, Severin Sidler, Massimo Giordano, Martina Bodini, Nathan Farinha, Benjamin Killeen, Christina Cheng, Yassine Jaoudi, and Geoffrey Burr. Equivalent-accuracy accelerated neural-network training using analogue memory. *Nature*, 558, 06 2018. 1, 6, 8

[3] Sergey Bartunov, Adam Santoro, Blake A. Richards, Luke Marris, Geoffrey E. Hinton, and Timothy P. Lillicrap. Assessing the scalability of biologically-motivated deep learning algorithms and architectures. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 9390–9400, Red Hook, NY, USA, 2018. Curran Associates Inc. 7, 21

[4] Adrian Bulat and Georgios Tzimiropoulos. XNOR-Net++: Improved Binary Neural Networks. *arXiv:1909.13863 [cs, eess]*, Sept. 2019. arXiv: 1909.13863. 5

[5] Kwan F. Cheung, Les E. Atlas, and Robert J. Marks. Synchronous vs asynchronous behavior of hopfield's cam neural net. *Appl. Opt.*, 26(22):4808–4813, Nov 1987. 6

[6] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. BinaryConnect: Training Deep Neural Networks with binary weights during propagations. *Advances in Neural Information Processing Systems*, 28:3123–3131, 2015. 3, 4, 5, 6

[7] M. Davies, N. Srinivasa, T. Lin, G. Chinya, Y. Cao, S. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. Weng, A. Wild, Y. Yang, and H. Wang. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(01):82–99, jan 2018. 3

[8] Maxence Ernoult, Julie Grollier, Damien Querlioz, Yoshua Bengio, and Benjamin Scellier. Updates of equilibrium prop match gradients of backprop through time in an rnn with static input. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 7081–7091. Curran Associates, Inc., 2019. 1, 3, 4, 5, 6, 7, 8, 12, 22, 23, 25, 27

[9] Maxence Ernoult, Julie Grollier, Damien Querlioz, Yoshua Bengio, and Benjamin Scellier. Equilibrium propagation with continual weight updates. *arXiv preprint arXiv:2005.04168*, 2020. 1

[10] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana. The spinnaker project. *Proceedings of the IEEE*, 102(5):652–665, 2014. 3

[11] Roshan Gopalakrishnan, Yansong Chua, Pengfei Sun, Ashish Jith Sreejith Kumar, and Arindam Basu. Hfnet: A cnn architecture co-designed for neuromorphic hardware with a crossbar array of synapses. *Frontiers in Neuroscience*, 14:907, 2020. 1

[12] K. He, X. Zhang, S. Ren, and J. Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, Dec. 2015. ISSN: 2380-7504. 16

[13] Koen Helwegen, James Widdicombe, Lukas Geiger, Zechun Liu, Kwang-Ting Cheng, and Roeland Nusselder. Latent weights do not exist: Rethinking binarized neural network optimization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 7533–7544. Curran Associates, Inc., 2019. 1, 3, 4, 22

[14] T Hirtzlin, Marc Bocquet, M Ernoult, J-O Klein, E Nowak, E Vianello, J-M Portal, and D Querlioz. Hybrid analog-digital learning with differential rram synapses. In *2019 IEEE International Electron Devices Meeting (IEDM)*, pages 22–6. IEEE, 2019. 1, 6

[15] Tifenn Hirtzlin, Marc Bocquet, Bogdan Penkovsky, Jacques-Olivier Klein, Etienne Nowak, Elisa Vianello, Jean-Michel Portal, and Damien Querlioz. Digital biologically plausible implementation of binarized neural networks with differential hafnium oxide resistive memory arrays. *Frontiers in Neuroscience*, 13:1383, 2020. 1, 6

[16] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4107–4115. Curran Associates, Inc., 2016. 4, 6

[17] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*, pages 448–456. PMLR, June 2015. ISSN: 1938-7228. 4

[18] Zhengyun Ji and Warren Gross. Towards efficient on-chip learning using equilibrium propagation. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2020. 3

[19] Jack Kendall, Ross Pantone, Kalpana Manickavasagam, Yoshua Bengio, and Benjamin Scellier. Training End-to-End Analog Neural Networks with Equilibrium Propagation. *arXiv:2006.01981*, June 2020. arXiv: 2006.01981. 1

[20] Axel Laborieux, Maxence Ernoult, Benjamin Scellier, Yoshua Bengio, Julie Grollier, and Damien Querlioz. Scaling equilibrium propagation to deep convnets by drastically reducing its gradient estimator bias, 2020. arXiv: 2006.03824. 1, 2, 3, 5, 7, 21, 25, 26, 27

[21] Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*, 2019. 1

[22] César Laurent, Gabriel Pereyra, Philémon Brakel, Ying Zhang, and Yoshua Bengio. Batch normalized recurrent neural networks. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2657–2661. IEEE, 2016. 4

[23] Danijela Marković, Alice Mizrahi, Damien Querlioz, and Julie Grollier. Physics for neuromorphic computing. *Nature Reviews Physics*, pages 1–12, July 2020. Publisher: Nature Publishing Group. 1

[24] Erwann Martin, Maxence Ernoult, Jérémie Laydevant, Shuai Li, Damien Querlioz, Teodora Petrisor, and Julie Grollier. EqSpike: Spike-driven Equilibrium Propagation for Neuromorphic Implementations. *arXiv:2010.07859 [cs]*, Jan. 2021. arXiv: 2010.07859. 1, 3, 8

[25] Paul A. Merolla, John V. Arthur, Rodrigo Alvarez-Icaza, Andrew S. Cassidy, Jun Sawada, Filipp Akopyan, Bryan L. Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, Bernard Brezzo, Ivan Vo, Steven K. Esser, Rathinakumar Appuswamy, Brian Taba, Arnon Amir, Myron D. Flickner, William P. Risk, Rajit Manohar, and Dharmendra S. Modha. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014. 3

[26] Thomas Mesnard, Wulfram Gerstner, and Johanni Brea. Towards deep learning with spiking neurons in energy based models with contrastive Hebbian plasticity. *arXiv:1612.03214 [cs, q-bio]*, Dec. 2016. arXiv: 1612.03214. 3, 8

[27] Peter O'Connor, Efstratios Gavves, and Max Welling. Training a spiking neural network with equilibrium propagation. In Kamalika Chaudhuri and Masashi Sugiyama, editors, *Proceedings of Machine Learning Research*, volume 89 of *Proceedings of Machine Learning Research*, pages 1516–1523. PMLR, 16–18 Apr 2019. 3, 8

[28] Michael Pfeiffer and Thomas Pfeil. Deep learning with spiking neurons: Opportunities and challenges. *Frontiers in neuroscience*, 12:774, 2018. 3

[29] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, Lecture Notes in Computer Science, pages 525–542, Cham, 2016. Springer International Publishing. 3, 4, 16

[30] Benjamin Scellier and Yoshua Bengio. Equilibrium Propagation: Bridging the Gap between Energy-Based Models and Backpropagation. *Frontiers in Computational Neuroscience*, 11, 2017. Publisher: Frontiers. 1, 2, 5, 6, 11, 12, 21, 27

[31] Benjamin Scellier and Yoshua Bengio. Equivalence of equilibrium propagation and recurrent backpropagation. *Neural computation*, 31(2):312–329, 2019. 7

[32] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. Green ai. *Commun. ACM*, 63(12):54–63, Nov. 2020. 1

[33] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for modern deep learning research. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(09):13693–13696, Apr. 2020. 1

[34] Chetan Singh Thakur, Jamal Lottier Molin, Gert Cauwenberghs, Giacomo Indiveri, Kundan Kumar, Ning Qiao, Johannes Schemmel, Runchun Wang, Elisabetta Chicca, Jennifer Olson Hasler, Jae-sun Seo, Shimeng Yu, Yu Cao, André van Schaik, and Ralph Etienne-Cummings. Large-Scale Neuromorphic Spiking Array Processors: A Quest to Mimic the Brain. *Frontiers in Neuroscience*, 12, 2018. Publisher: Frontiers. 1, 6

[35] Wenqiang Zhang, Bin Gao, Jianshi Tang, Peng Yao, Shimeng Yu, Meng-Fan Chang, Hoi-Jun Yoo, He Qian, and Huaqiang Wu. Neuro-inspired computing chips. *Nature Electronics*, 3(7):371–382, July 2020. Number: 7 Publisher: Nature Publishing Group. 1

[36] Gianluca Zoppo, Francesco Marrone, and Fernando Corinto. Equilibrium propagation for memristor-based recurrent neural networks. *Frontiers in neuroscience*, 14:240, 2020. 1