Equilibrium Propagation: Bridging the Gap Between Energy-Based Models and Backpropagation

Benjamin Scellier and Yoshua Bengio*
Université de Montréal, Montreal Institute for Learning Algorithms

September 27, 2016

Abstract

We introduce Equilibrium Propagation (e-prop), a learning algorithm for energy-based models. This algorithm involves only one kind of neural computation both for the first phase (when the prediction is made) and the second phase (after the target is revealed) of training. Contrary to backpropagation in feedforward networks, there is no need for special computation in the second phase of our learning algorithm. Equilibrium Propagation combines features of Contrastive Hebbian Learning and Contrastive Divergence while solving the theoretical issues of both algorithms: the algorithm computes the exact gradient of a well defined objective function. Because the objective function is defined in terms of local perturbations, the second phase of e-prop corresponds to only nudging the first-phase fixed point towards a configuration that has lower cost value. In the case of a multi-layer supervised neural network, the output units are slightly nudged towards their target, and the perturbation introduced at the output layer propagates backward in the network. The theory developed in this paper shows that the signal 'back-propagated' during this second phase actually contains information about the error derivatives, which we use to implement a learning rule proved to perform gradient descent with respect to the objective function. Thus, this work makes it more plausible that a mechanism similar to backpropagation could be implemented by brains.

1 Introduction

We introduce a very general and abstract machine learning framework for energy-based models (Section 2). In this framework, both the 'prediction' and the cost function are defined in terms of the energy function. The prediction corresponds to an energy minimum (also called 'fixed point'), and as such it is an implicit function of the data and the parameters of the model. We establish a general learning algorithm, called Equilibrium Propagation, that performs gradient descent on an objective function. We also present a special case of this general framework suited for supervised learning. But we expect that the general framework could be applied to more situations, yet to be discovered.

In section 3, we give a new description of the continuous Hopfield model (Hopfield, 1984) as a special case of our general framework. In the model, inputs are clamped and the network relaxes to a fixed point, corresponding to a local minimum of an energy function. The prediction is then read out on the output units at the fixed-point. This corresponds to the first phase of the algorithm. In the second phase of training, when the target values for output units are observed, the outputs are nudged towards their targets and the network relaxes to a new fixed point which corresponds to smaller prediction error. The learning rule, which is proved to perform gradient descent on the squared error, is a kind of contrastive Hebbian learning rule in which we *learn* the second-phase fixed point by reducing its energy and *unlearn* the first-phase fixed point by increasing its energy. However, our learning rule is not the usual contrastive Hebbian learning rule and it also differs from Boltzmann machine learning rules, as discussed in section 4.1.

During the second phase, the perturbation caused at the outputs propagates across hidden layers in the network. Because the propagation goes from outputs backward in the network, it is better thought of as a 'back-propagation'. It is shown by Bengio and Fischer (2015) that the early change of neural activities in the second phase correspond to the propagation of error derivatives with respect to neural activities. Our contribution in this paper is to go beyond the early change of neural activities and to show that the second phase also implements the (back)-propagation of error derivatives with respect to the synaptic weights.

In section 4, we compare our algorithm to previous existing algorithms. The recurrent back-propagation algorithm introduced by Pineda (1987); Almeida (1987) optimizes the same objective function as ours (formulated differently) but

*Y.B. is also a Senior Fellow of CIFAR

it involves a different kind of neural computation in the second phase of training, which is not satisfying from a biological perspective. The contrastive Hebbian learning rule for continuous Hopfield nets described by Movellan (1990) suffers from theoretical problems: the contrastive objective function that this algorithm optimizes may take negative values, in which case learning may deteriorate. The Contrastive Divergence algorithm (Hinton, 2002) has theoretical issues too: the CD_1 update rule is provably not the gradient of any objective function (Sutskever and Tieleman, 2010). The equivalence of back-propagation and contrastive Hebbian learning was shown by Xie and Seung (2003) but at the cost of extra assumptions: their model requires a layered MLP-like architecture, infinitesimal feedback weights and exponentially growing learning rates for remote layers.

Equilibrium Propagation solves all these issues at once (at least in theory). In our model, leaky integrator neural computation performs both *inference* (in the first phase) and *back-propagation of error derivatives* (in the second phase). The objective function is guaranteed to be always positive because it is defined in terms of infinitesimal local perturbations. e-prop computes the gradient of that objective function. Furthermore, we do not need extra hypotheses such as those required by Xie and Seung (2003). Note that algorithms related to ours based on infinitesimal perturbations at the outputs were also proposed by O'Reilly (1996); Hertz *et al.* (1997). However the theory developed in our paper is much more general than these earlier works and gives new insights on energy-based models.

Finally, we show experimentally in section 5 that our model is trainable. We train recurrent neural networks with 1, 2 and 3 hidden layers on MNIST and we achieve 0.00% training error. The generalization error lies between 2% and 3% depending on the architecture. The code for the model is available[1] for replicating and extending the experiments.

2 Implicit Framework for Machine Learning

We lay down the basis for a new framework for machine learning. In this framework the quantities of interest (the prediction and the objective function) are defined, in terms of the data and the parameters of the model, *implicitly* through an energy function, rather than *explicitly* (like in a feedforward net). This implicit definition involves the minimization of an energy function. This makes applications on digital hardware (such as GPUs) less practical as it needs long inference phases involving numerical optimization. But we expect that this framework could be extremely efficient if implemented by analog circuits, as suggested by Hertz *et al.* (1997).

The framework presented in this section is deterministic, but a natural extension to the stochastic case is presented in the Appendix B.

2.1 Energy Function, Prediction, Cost Function and Objective Function

We denote by s the state of the system and v the state of the external world (i.e. the data). The model is based on an energy function F that drives s toward low-energy configurations. The set of free parameters to be learned is denoted by θ, while β is an additional parameter that controls the level of influence of the external world on the system. The energy function, modeling all interactions within the system as well as the action of the external world on the system, is denoted by $F(\theta, \beta, s, \mathrm{v})$.

For fixed θ, β and v, we denote by $s_{\theta,\mathrm{v}}^{\beta}$ a local minimum of the energy function F, also called fixed point, which corresponds to the 'prediction' from the model:

$$s_{\theta,\mathrm{v}}^{\beta} \in \arg\min_{s} F(\theta, \beta, s, \mathrm{v}). \tag{1}$$

Here we use the notation $\arg\min$ to refer to the set of local minima. The fixed point is also characterized by

$$\frac{\partial F}{\partial s}\left(\theta, \beta, s_{\theta,\mathrm{v}}^{\beta}, \mathrm{v}\right) = 0 \tag{2}$$

and

$$\frac{\partial^2 F}{\partial s^2}\left(\theta, \beta, s_{\theta,\mathrm{v}}^{\beta}, \mathrm{v}\right) \quad \text{is a symmetric definite positive matrix.} \tag{3}$$

The fixed point $s_{\theta,\mathrm{v}}^{\beta}$ is the state of the system when it is in agreement with the state of the world v, the current 'model of the world' associated with θ, and the parameter β.

Let δ be a directional vector in the space of β. Note in particular that $\dim(\beta) = \dim(\delta)$. The values of β and δ being fixed, we define the cost function through

$$C_{\beta}^{\delta}(\theta, s, \mathrm{v}) := \delta^{T} \cdot \frac{\partial F}{\partial \beta}\left(\theta, \beta, s, \mathrm{v}\right). \tag{4}$$

[1] https://github.com/bscellier/Towards-a-Biologically-Plausible-Backprop

This is the cost associated with the state of the system s and the state of the external world v. The cost function is the directional derivative of the function $\beta \mapsto F(\theta, \beta, s, \mathrm{v})$ at the point β in the direction δ, which represents the variation of energy due to a change of β in the direction δ.

Finally, the objective function that we want to optimize is

$$J_\beta^\delta(\theta, \mathrm{v}) := C_\beta^\delta(\theta, s_{\theta,\mathrm{v}}^\beta, \mathrm{v}), \tag{5}$$

which is the cost associated to the fixed point $s_{\theta,\mathrm{v}}^\beta$, which depends on θ and v. Note the distinction between the cost function and the objective function: the cost function is defined for any state s, whereas the objective function is the cost at the fixed point $s_{\theta,\mathrm{v}}^\beta$. For practical use cases, the choice of the form of $F(\theta, \beta, s, \mathrm{v})$ as a function of β, as well as the choice of the values of β and δ, leave us freedom to design the cost function. We will see in the next subsection how to design a cost function for the supervised setting.

2.2 Example: Supervised Setting

In this subsection, we illustrate how to design an objective function for supervised learning.

When addressing the supervised learning scenario where one wants to predict y given x, we choose to split the state s of the system into three states: the 'input' state x, the 'output' state y and the hidden state h. We write $\mathrm{v} = \{\mathrm{x}, \mathrm{y}\}$ for the state of the outside world, and $s = \{x, y, h\}$ for the state of the system. The input state x and output state y aim to replicate x and y respectively. For this purpose, we decompose the energy function F into

$$F(\theta, \beta, s, \mathrm{v}) = E(\theta, s) + A(\beta, s, \mathrm{v}), \tag{6}$$

where $E(\theta, s)$ represents an internal potential energy that models the interactions within the system, and $A(\beta, s, \mathrm{v})$ is an external potential energy that models how the external world influences the system. The parameter β controls the level of influence of the external world on the system. Here, we consider as a simple choice of A the quadratic external potential

$$A(\beta, s, \mathrm{v}) := \frac{1}{2}\beta_x \left\| x - \mathrm{x} \right\|^2 + \frac{1}{2}\beta_y \left\| y - \mathrm{y} \right\|^2, \tag{7}$$

where β_x (resp. β_y) is a scalar that controls whether x (resp. y) is pushed towards x (resp.y) or not, and by how much. Thus, in this simple setting, the vectors β and δ are two-dimensional. We write $\beta = \{\beta_x, \beta_y\}$ and $\delta = \{\delta_x, \delta_y\}$. It remains to choose the values of β and δ, which specify the cost function C_β^δ and the objective function J_β^δ.

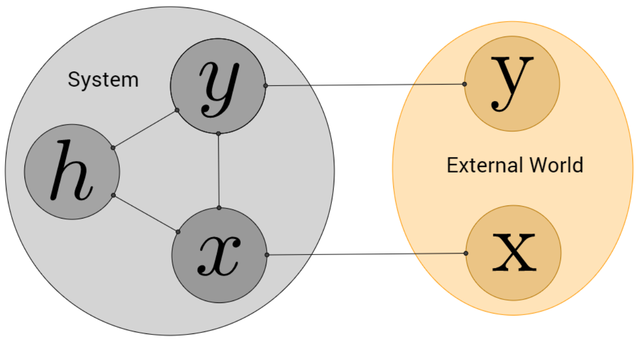


Figure 1: Diagram of the model in the supervised setting.

First we choose the value of β, which implicitly specifies the fixed point $s_{\theta,\mathrm{v}}^\beta$ corresponding to the prediction. We would like the input state to be always clamped, just like in other models such as the conditional Boltzmann machine. To this end, we choose $\beta_x = +\infty$. Indeed, for this choice of β_x, the only value of x that gives a finite energy is $x = \mathrm{x}$. Second, we do not want to choose $\beta_y > 0$ because this would be cheating by revealing the target y to the output y at prediction time, so we choose $\beta_y = 0$.

Then we choose the value of δ. Note that the cost function (Eq. 4) is equal to

$$C_\beta^\delta\left(\theta, s, \mathrm{v}\right) = \delta^T \cdot \frac{\partial F}{\partial \beta}\left(\theta, \beta, s, \mathrm{v}\right) = \delta_x \frac{\partial A}{\partial \beta_x}\left(\beta, s, \mathrm{v}\right) + \delta_y \frac{\partial A}{\partial \beta_y}\left(\beta, s, \mathrm{v}\right) = \frac{1}{2}\delta_x \left\| x - \mathrm{x} \right\|^2 + \frac{1}{2}\delta_y \left\| y - \mathrm{y} \right\|^2. \tag{8}$$

Therefore, what makes more sense for the supervised setting is to choose $\delta_x = 0$ and $\delta_y = 1$. We get

$$C_\beta^\delta\left(\theta, s, \mathrm{v}\right) = \frac{1}{2}\left\| y - \mathrm{y} \right\|^2. \tag{9}$$

Finally, we get for the objective function (Eq. 5)

$$J_\beta^\delta(\theta, \mathrm{v}) = \frac{1}{2} \left\| y_{\theta,\mathrm{v}}^\beta - \mathrm{y} \right\|^2 , \tag{10}$$

which represents the discrepancy between the prediction $y_{\theta,\mathrm{v}}^\beta$ and the target y. Here $y_{\theta,\mathrm{v}}^\beta$ represents the output state at the fixed point $s_{\theta,\mathrm{v}}^\beta = \left\{ x_{\theta,\mathrm{v}}^\beta, y_{\theta,\mathrm{v}}^\beta, h_{\theta,\mathrm{v}}^\beta \right\}$.

We will write $\beta = \{+\infty, 0\}$ and $\delta = \{0, 1\}$ for short.

Note that we could also add a regularization term $\lambda \, \Omega(\theta)$ to the objective function J_β^δ, where $\Omega(\theta)$ could be a L_1 or L_2 norm for example. For this purpose, we just need to add a term $\beta_\theta \, \Omega(\theta)$ to the energy function F, and to choose the corresponding values $\beta_\theta = 0$ and $\delta_\theta = \lambda$.

2.3 The Learning Algorithm: Equilibrium Propagation

The gradient of the objective function with respect to θ is given by the formula

$$\frac{d}{d\theta} J_\beta^\delta(\theta, \mathrm{v}) = \lim_{\xi \to 0} \frac{1}{\xi} \left(\frac{\partial F}{\partial \theta} \left(\theta, \beta + \xi\delta, s_{\theta,\mathrm{v}}^{\beta+\xi\delta}, \mathrm{v} \right) - \frac{\partial F}{\partial \theta} \left(\theta, \beta, s_{\theta,\mathrm{v}}^\beta, \mathrm{v} \right) \right). \tag{11}$$

This will be proved in section 2.5. Let us introduce some terminology. For fixed $\theta, \beta, \delta, \mathrm{v}$, we call $s_{\theta,\mathrm{v}}^{\beta+\xi\delta}$ a ξ-fixed point. Moreover, we call a ξ-phase any procedure that yields a ξ-fixed point by minimizing the energy function $F(\theta, \beta + \xi\delta, s, \mathrm{v})$ with respect to s. The gradient formula (Eq. 11) suggests the following two-phase training procedure. Given a data point v:

1. Run a 0-phase until the system settles to a 0-fixed point $s_{\theta,\mathrm{v}}^\beta$ and collect the statistics $\frac{\partial F}{\partial \theta} \left(\theta, \beta, s_{\theta,\mathrm{v}}^\beta, \mathrm{v} \right)$.
2. Run a ξ-phase for some "small" $\xi \neq 0$ until the system settles to a ξ-fixed point $s_{\theta,\mathrm{v}}^{\beta+\xi\delta}$ and collect the statistics $\frac{\partial F}{\partial \theta} \left(\theta, \beta + \xi\delta, s_{\theta,\mathrm{v}}^{\beta+\xi\delta}, \mathrm{v} \right)$.
3. Update the parameter θ according to

$$\Delta\theta \propto -\frac{1}{\xi} \left(\frac{\partial F}{\partial \theta} \left(\theta, \beta + \xi\delta, s_{\theta,\mathrm{v}}^{\beta+\xi\delta}, \mathrm{v} \right) - \frac{\partial F}{\partial \theta} \left(\theta, \beta, s_{\theta,\mathrm{v}}^\beta, \mathrm{v} \right) \right). \tag{12}$$

The gradient formula suggests that, starting from the 0-fixed point (which corresponds to the 'prediction'), a small change of the parameter β (from the value β to the value $\beta + \xi\delta$) causes slight modifications in the interactions in the system. This small perturbation makes the system settle to a new fixed point (a ξ-fixed point) which is ξ-close to the 0-fixed point. Simultaneously, a kind of contrastive update rule for θ is happening, in which the energy of the 0-fixed point is increased and the energy of the ξ-fixed point is decreased. We call this learning algorithm Equilibrium Propagation.

2.4 Traditional 'Explicit' Framework Vs Proposed 'Implicit' Framework

In the 'traditional' framework for machine learning, the prediction $\widehat{s} = f_\theta(\mathrm{v})$ is an *explicit* function of the data v and the parameter θ. The cost function $C(\theta, s, \mathrm{v})$ is also an explicit function of the parameter θ, the state s and the data v. For example, a cost function for supervised learning could be

$$C_\lambda(\theta, s, \mathrm{v}) = \frac{1}{2} \left\| y - \mathrm{y} \right\|^2 + \lambda \, \Omega(\theta), \tag{13}$$

where $s = \{x, y, h\}$ is the state of the system, $\mathrm{v} = \{\mathrm{x}, \mathrm{y}\}$ is the data, $\Omega(\theta)$ is a regularization term and λ is a hyper-parameter that controls the level of regularization. Thus, the objective function $J(\theta, \mathrm{v})$ is an explicit function of v and θ:

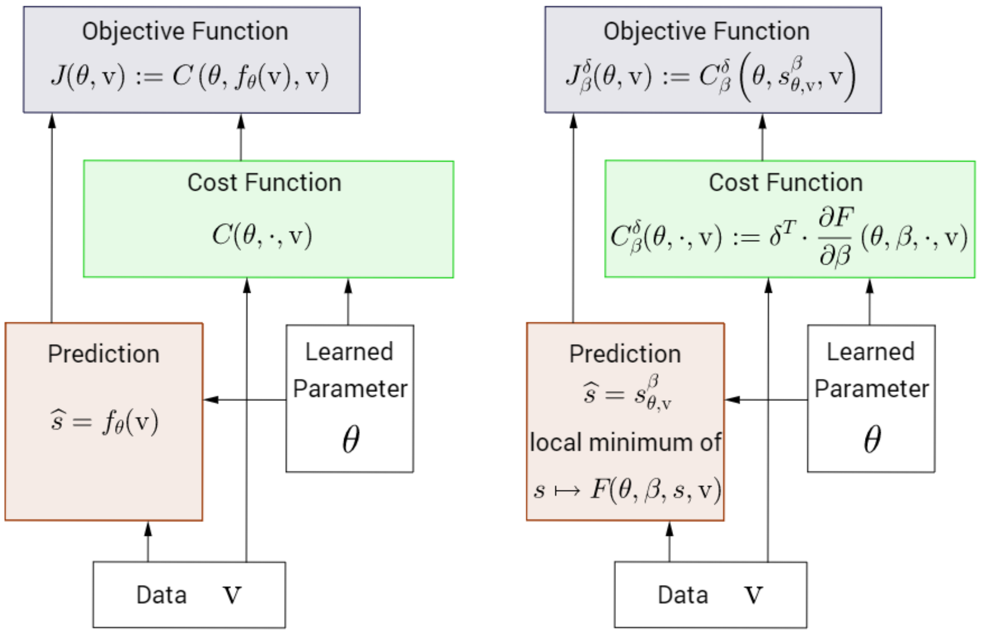
$$J(\theta, \mathrm{v}) := C(\theta, f_\theta(\mathrm{v}), \mathrm{v}), \tag{14}$$

so the parameter update

$$\Delta\theta \propto -\frac{\partial J}{\partial \theta}(\theta, \mathrm{v}) \tag{15}$$

can be done by automatic differentiation (i.e. standard backprop). The learning algorithm in the traditional framework requires two phases, known as forwardprop and backprop.

By contrast, in the framework for machine learning proposed here, the prediction $\widehat{s} = s_{\theta,\mathrm{v}}^\beta$ (or fixed point) is an *implicit* function of the external world v (or 'data') and the parameter θ, defined through the energy function F (Eq. 1, Eq. 2 and Eq. 3). The cost function $C_\beta^\delta(\theta, s, \mathrm{v})$ is also defined in terms of the energy function F and is an explicit



Objective Function
$J(\theta, \mathrm{v}) := C(\theta, f_\theta(\mathrm{v}), \mathrm{v})$
Cost Function
$C(\theta, \cdot, \mathrm{v})$
Prediction
$\widehat{s} = f_\theta(\mathrm{v})$
Learned Parameter
θ
Data v
Objective Function
$J_\beta^\delta(\theta, \mathrm{v}) := C_\beta^\delta\left(\theta, s_{\theta,\mathrm{v}}^\beta, \mathrm{v}\right)$
Cost Function
$C_\beta^\delta(\theta, \cdot, \mathrm{v}) := \delta^T \cdot \frac{\partial F}{\partial \beta}(\theta, \beta, \cdot, \mathrm{v})$
Prediction
$\widehat{s} = s_{\theta,\mathrm{v}}^\beta$
local minimum of
$s \mapsto F(\theta, \beta, s, \mathrm{v})$
Learned Parameter
θ
Data v

Figure 2: Comparison between the traditional 'explicit' framework for machine learning and our 'implicit' framework. In the red box, a vector $\widehat{s}$ called 'prediction' is built. In the green box, a function $s \mapsto C(\theta, s, \mathrm{v})$ called 'cost function' is built. In the blue box, the prediction and the cost function are combined to form the objective function. **Left.** In the traditional 'explicit' framework, the objective function J is optimized by automatic differentiation (i.e. standard backprop). **Right.** In our 'implicit' framework, the prediction $s_{\theta,\mathrm{v}}^\beta$ and the cost function C_β^δ are both defined in terms of the energy function F. The objective function J_β^δ is optimized with our learning algorithm: Equilibrium Propagation. The values of the parameters β and δ are fixed (not learned).

function of θ, s and v (Eq. 4). All in all, the objective function $J_\beta^\delta(\theta, \mathrm{v})$ is an implicit function of the prediction v and θ (Eq. 5). In particular we have seen how one can design a cost function for supervised learning (section 2.2) which is the same as Eq. 13. Finally the parameter update $\Delta\theta \propto -\frac{d}{d\theta} J_\beta^\delta(\theta, \mathrm{v})$ is done with our learning algorithm: Equilibrium Propagation (section 2.3). This learning algorithm requires two inference phases that we call 0-phase and ξ-phase, the second of which is faster since it requires only a slight perturbation of the system to move from the 0-fixed point to the ξ-fixed point.

An important consequence of this analysis is that, on digital hardware (e.g. a GPU), computations are *efficient* and (almost) *exact* in the traditional 'explicit' framework, whereas they are *inefficient* and *approximate* in the proposed 'implicit' framework (since they require numerical optimization). The experiments in section 5 will show that the inference phase is fairly long in the proposed implicit framework. However, we expect that the full potential of the proposed implicit framework could be exploited on analog hardware, as suggested by Hertz *et al.* (1997).

2.5 Main Theorem and Proof for the Gradient Formula

In this subsection we prove the gradient formula Eq. 11. (Another proof based on constrained optimization is proposed in Appendix A.) We first state and prove a theoretical result that holds for any energy function $F(\theta, \beta, s, \mathrm{v})$. We only assume $F(\theta, \beta, s, \mathrm{v})$ smooth enough so that the implicit function theorem guarantees that the fixed point $s_{\theta,\mathrm{v}}^\beta$ is a continuously differentiable function of (θ, β), for fixed v. Since v does not play any role in the theorem, its dependence is omitted in the notations.

Theorem 1 (Deterministic version)**.** *Let $F(\theta, \beta, s)$ be any energy function and s_θ^β a fixed point characterized by*

$$\frac{\partial F}{\partial s}(\theta, \beta, s_\theta^\beta) = 0. \tag{16}$$

Then we have

$$\left(\frac{d}{d\theta} \frac{\partial F}{\partial \beta}(\theta, \beta, s_\theta^\beta) \right)^T = \frac{d}{d\beta} \frac{\partial F}{\partial \theta}(\theta, \beta, s_\theta^\beta). \tag{17}$$

The notations $\frac{\partial F}{\partial \theta}$ and $\frac{\partial F}{\partial \beta}$ are used to mean the partial derivatives with respect to the first and second arguments of F respectively, whereas $\frac{d}{d\theta}$ and $\frac{d}{d\beta}$ represent the gradients of the entire expression with respect to θ and β respectively. Note that the object on both sides of Eq. 17 is a matrix of size $\dim(\beta) \times \dim(\theta)$. *Also, note that θ and β play symmetric roles.*

Proof. First we differentiate the fixed point equation Eq. 16 with respect to β:

$$\frac{d}{d\beta}\,(16) \qquad \Rightarrow \qquad \frac{\partial^2 F}{\partial s \partial \beta}(\theta, \beta, s_\theta^\beta) + \frac{\partial^2 F}{\partial s^2}(\theta, \beta, s_\theta^\beta) \cdot \frac{\partial s_\theta^\beta}{\partial \beta} = 0. \tag{18}$$

The left-hand side of Eq. 17 can be rewritten

$$\frac{d}{d\theta} \frac{\partial F}{\partial \beta}(\theta, \beta, s_\theta^\beta) = \frac{\partial^2 F}{\partial \theta \partial \beta}(\theta, \beta, s_\theta^\beta) + \left(\frac{\partial s_\theta^\beta}{\partial \theta} \right)^T \cdot \frac{\partial^2 F}{\partial s \partial \beta}(\theta, \beta, s_\theta^\beta) \tag{19}$$

$$= \frac{\partial^2 F}{\partial \theta \partial \beta}(\theta, \beta, s_\theta^\beta) - \left(\frac{\partial s_\theta^\beta}{\partial \theta} \right)^T \cdot \frac{\partial^2 F}{\partial s^2}(\theta, \beta, s_\theta^\beta) \cdot \frac{\partial s_\theta^\beta}{\partial \beta}, \tag{20}$$

where we used Eq. 18. Similarly we differentiate the fixed point equation Eq. 16 with respect to θ:

$$\frac{d}{d\theta}\,(16) \qquad \Rightarrow \qquad \frac{\partial^2 F}{\partial s \partial \theta}(\theta, \beta, s_\theta^\beta) + \frac{\partial^2 F}{\partial s^2}(\theta, \beta, s_\theta^\beta) \cdot \frac{\partial s_\theta^\beta}{\partial \theta} = 0. \tag{21}$$

The right-hand side of Eq. 17 can be rewritten

$$\frac{d}{d\beta} \frac{\partial F}{\partial \theta}(\theta, \beta, s_\theta^\beta) = \frac{\partial^2 F}{\partial \beta \partial \theta}(\theta, \beta, s_\theta^\beta) + \left(\frac{\partial s_\theta^\beta}{\partial \beta} \right)^T \cdot \frac{\partial^2 F}{\partial s \partial \theta}(\theta, \beta, s_\theta^\beta) \tag{22}$$

$$= \frac{\partial^2 F}{\partial \beta \partial \theta}(\theta, \beta, s_\theta^\beta) - \left(\frac{\partial s_\theta^\beta}{\partial \beta} \right)^T \cdot \frac{\partial^2 F}{\partial s^2}(\theta, \beta, s_\theta^\beta) \cdot \frac{\partial s_\theta^\beta}{\partial \theta}, \tag{23}$$

where we used Eq. 21. Since Eq. 20 is the transpose of Eq. 23, we conclude that Eq. 17 is valid. □

We will see shortly that the gradient formula (Eq. 11) for the objective function (Eq. 5) naturally arises from Theorem 1. Multiplying both sides of Eq. 17 by δ^T we get

$$\delta^T \cdot \left(\frac{d}{d\theta} \frac{\partial F}{\partial \beta} \left(\theta, \beta, s_{\theta,\mathrm{v}}^\beta, \mathrm{v}\right) \right)^T = \delta^T \cdot \frac{d}{d\beta} \frac{\partial F}{\partial \theta} \left(\theta, \beta, s_{\theta,\mathrm{v}}^\beta, \mathrm{v}\right). \tag{24}$$

The left-hand side of Eq. 24 represents the gradient of the objective function with respect to θ:

$$\frac{d}{d\theta} J_\beta^\delta(\theta, \mathrm{v}). \tag{25}$$

On the other hand, the right-hand side of Eq. 24 represents the directional derivative of the function

$$\beta \mapsto \frac{\partial F}{\partial \theta} \left(\theta, \beta, s_{\theta,\mathrm{v}}^\beta, \mathrm{v}\right) \tag{26}$$

at the point β in the direction δ. This is also the derivative of the function

$$\xi \mapsto \frac{\partial F}{\partial \theta} \left(\theta, \beta + \xi\delta, s_{\theta,\mathrm{v}}^{\beta+\xi\delta}, \mathrm{v}\right) \tag{27}$$

at $\xi = 0$ (where ξ is a scalar), which can be rewritten

$$\lim_{\xi \to 0} \frac{1}{\xi} \left(\frac{\partial F}{\partial \theta} \left(\theta, \beta + \xi\delta, s_{\theta,\mathrm{v}}^{\beta+\xi\delta}, \mathrm{v}\right) - \frac{\partial F}{\partial \theta} \left(\theta, \beta, s_{\theta,\mathrm{v}}^\beta, \mathrm{v}\right) \right). \tag{28}$$

Therefore, combining Eq.25 and Eq. 28 we get the gradient formula (Eq. 11).

2.6 More Insight on the Learning Rule

In the supervised setting introduced in section 2.2, the 0-phase is with $\beta = \{+\infty, 0\}$, meaning that the input state is clamped and the output state is free. As for the ξ-phase, we have $\beta + \xi\delta = \{+\infty, \xi\}$, which means that a novel term $\frac{1}{2}\xi \left\| y - \mathrm{y} \right\|^2$ is introduced in the total energy F. If $\xi > 0$, the novel term slightly attracts the output state y to the target y; if $\xi < 0$, the novel term slightly repels y from y. Clearly, if $\xi > 0$ (resp. $\xi < 0$), the ξ-fixed point is better (resp. worse) than the 0-fixed point in terms of prediction error. The following proposition generalizes this property to the general setting.

Proposition 2 (Deterministic version)**.** *The derivative of the function*

$$\xi \mapsto C_\beta^\delta(\theta, s_{\theta,\mathrm{v}}^{\beta+\xi\delta}, \mathrm{v}) \tag{29}$$

at $\xi = 0$ *is non-positive.*

Proof. Like in section 2.5 we omit to write the dependence on v in the notations. Multiplying both sides of Eq. 18 on the left by $\delta^T \cdot \left(\frac{\partial s_\theta^\beta}{\partial \beta} \right)^T$ and on the right by δ, we get

$$\delta^T \cdot \left(\frac{\partial s_\theta^\beta}{\partial \beta} \right)^T \cdot \frac{\partial^2 F}{\partial s \partial \beta}(\theta, \beta, s_\theta^\beta) \cdot \delta = -\delta^T \cdot \left(\frac{\partial s_\theta^\beta}{\partial \beta} \right)^T \cdot \frac{\partial^2 F}{\partial s^2}(\theta, \beta, s_\theta^\beta) \cdot \frac{\partial s_\theta^\beta}{\partial \beta} \cdot \delta \leq 0. \tag{30}$$

The inequality holds because $\frac{\partial^2 F}{\partial s^2}(\theta, \beta, s_\theta^\beta)$ is symmetric positive (as s_θ^β is a local minimum of F). The left-hand side represents the derivative of Eq. 29 at $\xi = 0$. ☐

Proposition 2 shows that, unless $s_{\theta,\mathrm{v}}^\beta$ is already optimal in terms of prediction error, for $\xi > 0$ small enough, the ξ-fixed point achieves lower prediction error than the 0-fixed point. Thus, a small perturbation due to a change of β in the direction δ would nudge the system towards a state that reduces prediction error. This property sheds light on the update rule Eq. 12, which can be seen as a kind of contrastive learning rule (somehow similar to the Boltzmann machine learning rule) where we *learn* the slightly better state $s_{\theta,\mathrm{v}}^{\beta+\xi\delta}$ by reducing its energy and *unlearn* the slightly worse state $s_{\theta,\mathrm{v}}^\beta$ by increasing its energy.

However, our learning rule is different from the Boltzmann machine learning rule and the contrastive Hebbian learning rule. The differences between these algorithms will be discussed in section 4.1.

3 The Continuous Hopfield Model Revisited: Equilibrium Prop as a More Biologically Plausible Backprop

To illustrate our implicit framework for machine learning, we propose in this section an interpretation of e-prop as a new kind of 'backprop'. Equilibrium Prop has some features that make more sense, from a physical point of view, than the standard backprop implemented by feedforward nets. However, still several points need to be elucidated from a biological perspective. Perhaps the most important of them is that our model (like many other models) requires symmetric weights.

The setting considered here is the supervised setting presented in section 2.2. In the context of neural networks, the state $s = \{x, y, h\}$ is the vector of the states of the units and $\theta = (W, b)$ represents the set of free parameters to be learned, which includes the synaptic weights W_{ij} and the neuron biases b_i. The units are continuous-valued and would correspond to averaged voltage potential across time, spikes, and possibly neurons in the same minicolumn. Finally, ρ is a nonlinear activation function such that $\rho(s_i)$ represents the firing rate of unit i. The network is recurrently connected with symmetric connections. The algorithm presented here is applicable to any architecture, even a fully connected network. However, to make the connection to backpropagation more obvious, we will consider more specifically a layered architecture with no skip-connections and no lateral connections within a layer (Figure 3).

3.1 A Kind of Hopfield Energy

Recall that in the supervised setting introduced in section 2.2, the energy function F is divided into an internal potential E and an external potential A (Eq. 6). Moreover the external potential A is the quadratic potential (Eq. 7). As an internal potential E, we consider here a kind of Hopfield energy, first studied by Bengio and Fischer (2015); Bengio *et al.* (2015a,b):

$$E(\theta, s) := \frac{1}{2} \sum_i s_i^2 - \frac{1}{2} \sum_{i \neq j} W_{ij} \rho(s_i) \rho(s_j) - \sum_i b_i \rho(s_i). \tag{31}$$

The connections are supposed to be symmetric, that is $W_{ij} = W_{ji}$. Note that

$$\frac{\partial E}{\partial W_{ij}}(\theta, s) = -\rho(s_i)\rho(s_j). \tag{32}$$

Thus, the learning rule Eq. 11 can be rewritten

$$\Delta W_{ij} \propto \lim_{\xi \to 0} \frac{1}{\xi}\left(\rho\left(s_i^{\xi}\right)\rho\left(s_j^{\xi}\right) - \rho\left(s_i^{0}\right)\rho\left(s_j^{0}\right)\right) \tag{33}$$

where $s^{\xi} := s_{\theta,\mathrm{v}}^{\beta+\xi\delta}$ and $s^{0} := s_{\theta,\mathrm{v}}^{\beta}$ represent the ξ-fixed point and the 0-fixed point. Thus, the learning rule Eq. 33 is a kind of contrastive Hebbian learning rule, somewhat similar to the one studied by Movellan (1990) and the Boltzmann machine learning rule. The differences with these algorithms will be discussed in section 4.1.

3.2 The Neuronal Dynamics

It has been hypothesized numerous times (Hinton and Sejnowski, 1986; Friston and Stephan, 2007; Berkes *et al.*, 2011) that, given a state of sensory information (current and past inputs), neurons are collectively performing inference, i.e., moving towards configurations that better 'explain' the observed sensory data. We can think of the neurons' configuration as an 'explanation' (or 'interpretation') for the observed sensory data, In the energy-based model presented here, that means that the units of the network gradually move towards lower energy configurations that are more probable, given the sensory input and according to the current "model of the world" associated with the parameters of the model.

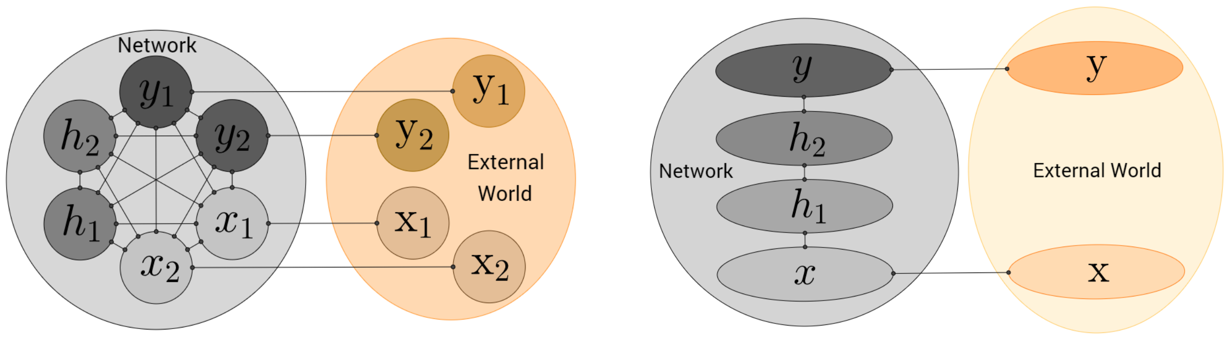


Figure 3: **Left.** Equilibrium Propagation applies to any architecture, even a fully connected network. **Right.** The connection with Backpropagation is more obvious when the network has a layered architecture.

An obvious way to reach the fixed point $s_{\theta,\mathrm{v}}^{\beta}$ is to perform gradient descent on $F(\theta, \beta, s, \mathrm{v})$ with respect to s: we assume that the time evolution of the units is governed by the gradient system

$$\frac{ds}{dt} = -\frac{\partial F}{\partial s}(\theta, \beta, s, \mathrm{v}). \tag{34}$$

Unlike more conventional artificial neural networks, the model that we study here is a continuous-time dynamical system described by the differential equation of motion Eq. 34. The total energy of the system decreases as time progresses (θ, β and v being fixed) since

$$\frac{dF}{dt} = \frac{\partial F}{\partial s}(\theta, \beta, s, \mathrm{v}) \cdot \frac{ds}{dt} = -\left\|\frac{ds}{dt}\right\|^2 \leq 0. \tag{35}$$

The energy stops decreasing when the network has reached a fixed point:

$$\frac{dF}{dt} = 0 \quad \Leftrightarrow \quad \frac{ds}{dt} = 0 \quad \Leftrightarrow \quad \frac{\partial F}{\partial s}(\theta, \beta, s, \mathrm{v}) = 0 \quad \Leftrightarrow \quad s = s_{\theta,\mathrm{v}}^{\beta}. \tag{36}$$

The differential equation of motion Eq. 34 can be seen as a sum of two 'forces' that act on the temporal derivative of s:

$$\frac{ds}{dt} = -\frac{\partial E}{\partial s}(\theta, s) - \frac{\partial A}{\partial s}(\beta, s, \mathrm{v}). \tag{37}$$

The 'internal force' induced by the internal potential (Eq. 31) on the i-th unit is

$$-\frac{\partial E}{\partial s_i}(\theta, s) = \rho'(s_i)\left(\sum_{j\neq i} W_{ij}\rho(s_j) + b_i\right) - s_i, \tag{38}$$

while the 'external force' induced by the external potential (Eq. 7) on x_i, y_i and h_i is respectively

$$-\frac{\partial A}{\partial x_i}(\beta, s, \mathrm{v}) = \beta_x(\mathrm{x}_i - x_i), \qquad -\frac{\partial A}{\partial y_i}(\beta, s, \mathrm{v}) = \beta_y(\mathrm{y}_i - y_i) \qquad \text{and} \qquad -\frac{\partial A}{\partial h_i}(\beta, s, \mathrm{v}) = 0. \tag{39}$$

The form of Eq. 38 recalls a kind of leaky integrator neuron model, in which neurons are seen as performing leaky temporal integration of their past inputs. Note that the hypothesis of symmetric connections ($W_{ij} = W_{ji}$) was used to derive Eq. 38. As discussed in Bengio and Fischer (2015), the factor $\rho'(s_i)$ would suggest that when a neuron is saturated (firing at the maximal or minimal rate so that $\rho'(s_i) \approx 0$), its state is not sensitive to external inputs, while the leaky term drives it out of the saturation regime, towards its resting value $s_i = 0$.

The form of Eq. 39 suggests that when $\beta_x > 0$, the 'external force' drives the input unit x_i towards the state of the world x_i. In the limit $\beta_x \to +\infty$, the input unit x_i moves infinitely fast towards x_i, so x_i is immediately clamped to x_i and is no longer sensitive to the 'internal force'. The same reasoning holds for output units.

Finally, a more likely dynamics would include some form of noise. In the Appendix B, we present a stochastic framework that naturally extends the analysis here.

3.3 Interpretation of the ξ-Phase as Performing Backpropagation of Errors

Recall that in the supervised setting, we have $\beta = \{+\infty, 0\}$ and $\delta = \{0, 1\}$, so that the 0-phase is with $\beta = \{+\infty, 0\}$ and the ξ-phase is with $\beta + \xi\delta = \{+\infty, \xi\}$. Although Equilibrium Prop works in principle with any infinitesimal $\xi \neq 0$, here we take ξ *positive*.

At the beginning of the ξ-phase, starting from the 0-fixed point, the novel term $\frac{1}{2}\xi \left\|y - \mathrm{y}\right\|^2$ in the external potential induces a new 'external force' that acts on the output units:

$$-\frac{\partial A}{\partial y_i}(\beta + \xi\delta, s, \mathrm{v}) = \xi(\mathrm{y}_i - y_i). \tag{40}$$

This force models the observation of y: it drives the output units from their 0-fixed point value in the direction of their target. Since this force only acts on the output units, the hidden units are initially at equilibrium at the beginning of the ξ-phase, but the perturbation caused at the output units will propagate in the hidden units as time progresses. When the architecture is a multi-layered net (Figure 3. Right), the perturbation at the output layer propagates backwards across the hidden layers of the network. This propagation is thus better thought of as a 'back-propagation'.

But the ξ-phase wouldn't deserve the name of 'back-propagation' if the (back-)propagated perturbation did not correspond to error signals. It was first shown by Bengio and Fischer (2015) that, starting from the 0-fixed point, the early changes of neural activities during the ξ-phase (caused by the output units moving towards their target) approximate some kind of error derivatives. They considered a regular multi-layer neural network with no skip connections and no lateral connections within a layer.

In this paper, the update rule Eq. 33 shows that the ξ-phase also implements the (back)-propagation of error derivatives with respect to the synaptic weights: starting from the 0-fixed point, the combination of a ξ-phase with the update rule Eq. 33 gives rise to stochastic gradient descent on the objective function. Moreover, this result holds for any architecture and not just a layered architecture (Figure 3) like the one considered by Bengio and Fischer (2015).

In particular, leaky integrator neural computation as described in section 3.2, performs both *inference* (in the 0-phase) and *error back-propagation* (in the ξ-phase).

3.4 Connection to Spike-Timing Dependent Plasticity

Spike-Timing Dependent Plasticity (STDP) is believed to be a prominent form of synaptic change in neurons (Markram and Sakmann, 1995; Gerstner *et al.*, 1996). It relates the expected change in synaptic weights to the timing difference between postsynaptic spikes and presynaptic spikes. It is the result of experimental observations in biological neurons, but its role as part of a learning algorithm remains a topic where more exploration is needed. Here is an attempt in this direction.

Experimental results by Bengio *et al.* (2015a) show that if the temporal derivative of the synaptic weight W_{ij} satisfies

$$\frac{dW_{ij}}{dt} \propto \rho(s_i)\frac{ds_j}{dt}, \tag{41}$$

then one recovers the experimental observations by Bi and Poo (2001) in biological neurons.

Here we change the learning rule Eq. 41 into

$$\frac{dW_{ij}}{dt} \propto \rho(s_i) \frac{d\rho(s_j)}{dt}, \tag{42}$$

which is also the one studied by Xie and Seung (2000). The two rules Eq. 41 and Eq. 42 are the same up to a factor $\rho'(s_j)$. An advantage of Eq. 42 is that it leads to a more natural view of the update rule in the case of tied weights studied here ($W_{ij} = W_{ji}$). Indeed, the update should take into account the pressures from both the i to j and j to i synapses, so that the total update under constraint is

$$\frac{dW_{ij}}{dt} \propto \rho(s_i) \frac{d\rho(s_j)}{dt} + \rho(s_j) \frac{d\rho(s_i)}{dt} = \frac{d}{dt} \rho(s_i)\rho(s_j). \tag{43}$$

By integrating Eq. 43 on the path from $s^0 := s^{\beta}_{\theta,\mathrm{v}}$ to $s^{\xi} := s^{\beta+\xi\delta}_{\theta,\mathrm{v}}$ during the ξ-phase, we get

$$\Delta W_{ij} \propto \int \frac{dW_{ij}}{dt} dt = \int \frac{d}{dt} \rho(s_i)\rho(s_j) dt = \int d\left(\rho(s_i)\rho(s_j)\right) = \rho\left(s_i^{\xi}\right)\rho\left(s_j^{\xi}\right) - \rho\left(s_i^{0}\right)\rho\left(s_j^{0}\right), \tag{44}$$

which is the same rule as Eq. 33 up to a factor $1/\xi$. Therefore the update rule Eq. 33 can be interpreted as a continuous-time integration of the learning rule Eq. 42, in the case of symmetric weights, on the path from s^0 to s^{ξ} during the ξ-phase.

4 Related Work and Discussion

Our learning algorithm, Equilibrium Propagation, gives a new perspective on the relationship between back-propagation in feedforward nets and contrastive Hebbian learning in energy-based models.

Learning Algorithm	Backprop	Equilibrium Prop	Contrastive Hebbian Learning	Almeida-Pineida
First Phase	Forward Pass	$0-$ Phase	Negative Phase (or Free Phase)	Free Phase
Second Phase	Backward Pass	$\xi-$ Phase	Positive Phase (or Clamped Phase)	Recurrent Backprop

Table 1: Correspondence of the phases for different learning algorithms: Back-propagation, Equilibrium Propagation (our algorithm), Contrastive Hebbian Learning (in the Hopfield Model and the Boltzmann Machine) and Almeida-Pineida's Recurrent Back-Propagation

Previous work on the back-propagation interpretation of contrastive Hebbian learning was done by Xie and Seung (2003). Under certain assumptions, they show that the contrastive Hebbian learning rule is equivalent to back-propagation. However, the equivalence shown in their work relies on extra hypotheses. Their result only holds for a layered MLP-like architecture, the feedback weights are assumed to be tiny compared to the feedforward weights, and each layer has its own learning rate whose value grows exponentially with depth (to compensate for the tiny feedback weights).

Our algorithm, Equilibrium Propagation, works for any architecture (Figure 3. Left) and does not need extra assumptions.

4.1 Differences with Contrastive Hebbian Learning and Boltzmann Machine Learning

Despite the similarity between our learning rule (Eq. 11) and the contrastive Hebbian learning rule (CHL) for the continuous Hopfield model and the Boltzmann machine, there are important differences.

First, recall that our objective function is defined as

$$J^{\delta}_{\beta}(\theta, \mathrm{v}) = \delta^T \cdot \frac{\partial F}{\partial \beta}\left(\theta, \beta, s^{\beta}_{\theta,\mathrm{v}}, \mathrm{v}\right), \tag{45}$$

where δ represents the direction in which we slightly change the parameter β around the value β. By contrast, with our notations, the contrastive objective function (which the contrastive Hebbian learning rule for the continuous Hopfield model optimizes) can be written:

$$J_{\mathrm{contrastive}}(\theta, \mathrm{v}) := F(\theta, \beta^+, s^{\beta^+}_{\theta,\mathrm{v}}, \mathrm{v}) - F(\theta, \beta^-, s^{\beta^-}_{\theta,\mathrm{v}}, \mathrm{v}), \tag{46}$$

where β^+ and β^- are two different values of β corresponding to when the target is clamped or not. The contrastive objective function has theoretical problems: it may be negative if the clamped phase and free phase stabilize in different modes of the energy function, in which case learning usually deteriorates, as pointed out by Movellan (1990). Our objective function does not suffer from this problem, because it is defined in terms of infinitesimal local perturbations, so that the implicit function theorem guarantees that the ξ-fixed point will be in the same mode as the 0-fixed point, and ξ-close to it.

Our learning algorithm is also more flexible, because the cost function (Eq. 4) involved in the definition of the objective function (Eq. 5) can be any given function. To see this, note that for any given function $C(\theta, s, \mathrm{v})$, by choosing

$$F(\theta, \beta, s, \mathrm{v}) = E(\theta, s, \mathrm{v}) + \beta\, C(\theta, s, \mathrm{v}) \tag{47}$$

as an energy function, where β is a scalar, $C(\theta, s, \mathrm{v})$ is the cost function associated to the values $\beta = 0$ and $\delta = 1$. Indeed

$$C_0^1(\theta, s, \mathrm{v}) = 1 \times \frac{\partial F}{\partial \beta}(\theta, 0, s, \mathrm{v}) = C(\theta, s, \mathrm{v}). \tag{48}$$

On the other hand, the contrastive function and the log-likelihood that CHL and the Boltzmann machine optimize are determined by the Hopfield energy.

The ξ-phase of e-prop (going from the 0-fixed point to the ξ-fixed point) can be seen as a brief 'backprop' phase with weakly clamped target outputs (section 3.3). By contrast, in the positive phase of the Boltzmann machine, the target is fully clamped, so the (correct version of the) Boltzmann machine learning rule requires two independent phases, making an analogy with backprop less obvious.

The latter argument tends to show that our algorithm would be similar to the CD algorithm (Contrastive Divergence) for Boltzmann machines. Indeed, in our model, we start from a 0-fixed point (which requires a long 0-phase relaxation) and then we run a short ξ-phase. In the CD algorithm, one starts from a positive equilibrium sample with the visible units clamped (which requires a long positive phase Markov chain in the case of a general Boltzmann machine) and then one runs a short negative phase. But there is an important difference: our algorithm computes the *correct* gradient on our objective function, whereas the CD algorithm computes an *approximation* to the gradient of the log-likelihood. The CD_1 update rule is provably not the gradient of any objective function (Sutskever and Tieleman, 2010).

Finally, in the supervised setting, a more subtle difference with the Boltzmann machine is that the 'input' and 'output' states x and y in our model are best thought of as being part of the latent state s. If we were to make a correspondence with the Boltzmann machine, the visible units of the Boltzmann machine would be $\mathrm{v} = \{\mathrm{x}, \mathrm{y}\}$, while the hidden units would be $s = \{x, y, h\}$. In the Boltzmann machine, the state of the external world is inferred directly on the visible units, whereas in our model we make the choice to integrate in s special latent variables x and y that aim to reconstruct the external world $\mathrm{v} = \{\mathrm{x}, \mathrm{y}\}$.

4.2 Link to Recurrent Back-Propagation

Directly connected to the analysis presented here is the work by Pineda (1987); Almeida (1987) on recurrent back-propagation. They consider the same objective function as ours, but formulate the problem as a constrained optimization problem. In Appendix A we derive another proof for the learning rule Eq. 11 with the formalism of constrained optimization problems. The beginning of this proof is in essence the same as the one proposed by Pineda (1987); Almeida (1987), but there is a major difference when it comes to solving Eq. 61 for the costate variable λ^*. The method proposed by Pineda (1987); Almeida (1987) is to use Eq. 61 to compute λ^* by a fixed point iteration in a linearized form of the recurrent network. The computation of λ^* corresponds to their second phase, which they call *recurrent back-propagation*. However, this second phase does not follow the same kind of dynamics as the first phase (called 0-phase in this paper) because it uses a linearization of the neural activation rather than the fully non-linear activation. From a biological plausibility point of view, having to use a different kind of hardware and computation for the two phases is not satisfying.

By contrast, like the continuous Hopfield net and the Boltzmann machine, our model involves only one kind of neural computations for both phases.

4.3 Discussion

From a biological perspective, a troubling issue is the requirement of symmetric weights between the units. This issue is almost omnipresent in energy-based models. Even in feedforward nets, the backward pass requires the feedback weights to be symmetric to the feedforward weights. Note that the units in our model need not correspond exactly to actual neurons in the brain (it could be groups of neurons in a cortical microcircuit, for example). It remains to be shown how a form of symmetry could arise from the learning procedure itself (for example from autoencoder-like unsupervised learning) or if a different formulation could eliminate the symmetry requirement. Encouraging cues come from the observation that denoising autoencoders without tied weights often end up learning symmetric weights (Vincent *et al.*, 2010). Another encouraging piece of evidence, also linked to autoencoders, is the theoretical result from Arora *et al.* (2015), showing that the symmetric solution minimizes the autoencoder reconstruction error between two successive layers of rectifying (ReLU) units. Also, recent work by Lillicrap *et al.* (2014) shows that the backpropagation algorithm for feedforward nets also works when the feedback weights are random.

Another practical issue is that we would like to reduce the negative impact of a lengthy relaxation to a fixed point in the 0-phase. A possibility is explored by Bengio *et al.* (2016) and was initially discussed by Salakhutdinov and Hinton (2009) in the context of a stack of RBMs: by making each layer a good autoencoder, it is possible to make this iterative inference converge quickly after an initial feedforward phase, because the feedback paths "agree" with the states already computed in the feedforward phase.

5 Implementation of the Model and Experimental Results

In this section, we provide experimental evidence that our model is trainable. Recall that our model is a recurrently connected neural network with symmetric connections. Here, we train multi-layered networks with 1, 2 and 3 hidden layers, with no skip-layer connections and no lateral connections within layers, to classify the MNIST digits. Although we believe that analog hardware would be more suited for our model, here we propose an implementation on digital hardware (a GPU). We achieve 0.00% training error. The generalization error lies between 2% and 3% depending on the architecture.

For each training example (x, y) in the dataset, training proceeds as follows:

1. Clamp x to the input units x.
2. Run a 0-phase until the hidden and output units settle to a 0-fixed point s^0, and collect the statistics $\frac{\partial E}{\partial \theta}\left(\theta, s^0\right)$.
3. Run a ξ-phase for some "small" $\xi > 0$ until the hidden and output units settle to a ξ-fixed point s^ξ, and collect the statistics $\frac{\partial E}{\partial \theta}\left(\theta, s^\xi\right)$.
4. Update each synapse W_{ij} according to

$$\Delta W_{ij} \propto \frac{1}{\xi}\left(\rho\left(s_i^\xi\right)\rho\left(s_j^\xi\right) - \rho\left(s_i^0\right)\rho\left(s_j^0\right)\right) \tag{49}$$

The prediction is made at the 0-fixed point s^0 at the end of the 0-phase relaxation. The predicted value y_{pred} is the index of the output unit whose activation is maximal among the 10 output units, that is

$$y_{\mathrm{pred}} := \arg\max_i y_i^0. \tag{50}$$

Note that no constraint is imposed on the activations of the units of the output layer in our model, unlike more traditional neural networks where a softmax output layer is used to constrain them to sum up to 1. Recall that the objective function that we minimize is the square of the difference between our prediction and the one-hot encoding of the target value:

$$J_\beta^\delta(\theta, \mathrm{v}) = \left\|\mathrm{y} - y^0\right\|^2. \tag{51}$$

5.1 Finite Differences

Implementation of the differential equation of motion. First we clamp x on the input units x. Then the obvious way to implement Eq. 34 is to discretize time into short time lapses of duration ϵ and to update each hidden and output unit s_i according to

$$s_i \leftarrow s_i - \epsilon \frac{\partial F}{\partial s_i}(\theta, \beta, s, \mathrm{v}). \tag{52}$$

This is simply one step of gradient descent on the total energy F, with step size ϵ.

For our experiments, we choose the hard sigmoid activation function $\rho(s_i) = 0 \vee s_i \wedge 1$, where $\vee$ denotes the max and $\wedge$ the min. For this choice of ρ, since $\rho'(s_i) = 0$ for $s_i < 0$, it follows from Eq. 38 and Eq. 39 that if $h_i < 0$ then $\frac{\partial F}{\partial h_i}(\theta, \beta, s, \mathrm{v}) = -h_i > 0$. This force prevents the hidden unit h_i from going in the range of negative values. The same is true for the output units. Similarly, s_i cannot reach values above 1. As a consequence s_i must remain in the domain $0 \leq s_i \leq 1$. Therefore, rather than the standard gradient descent (Eq. 52), we will use a slightly modified version:

$$s_i \leftarrow 0 \vee \left(s_i - \epsilon \frac{\partial F}{\partial s_i}(\theta, \beta, s, \mathrm{v})\right) \wedge 1. \tag{53}$$

Choice of the step size ϵ. We find experimentally that the choice of ϵ has little influence as long as $0 < \epsilon < 1$. What matters more is the *total duration of the relaxation* $\Delta t = n_{\mathrm{iter}} \times \epsilon$ (where n_{iter} is the number of iterations). In our experiments we choose $\epsilon = 0.5$ to keep $n_{\mathrm{iter}} = \Delta t / \epsilon$ as small as possible so as to avoid extra unnecessary computations.

Duration of the 0-phase relaxation. We find experimentally that the number of iterations required for the 0-phase relaxation is large and grows fast as the number of layers increases (Table 2), which considerably slows down training. More experimental and theoretical investigation would be needed to analyze the number of iterations required, but we leave that for future work.

Duration of the ξ-phase. During the ξ-phase we only initiate the movement of the units. Notice that the time constant of the integration process in the leaky integrator equation Eq. 38 is $\tau = 1$. This time constant represents the time needed for a signal to propagate from a layer to the next one with "significant amplitude". So the time needed for the error signals to back-propagate in the network is $N\tau = N$, where N is the number of hidden and output layers of the network. So we perform N/ϵ iterations with step size $\epsilon = 0.5$.

5.2 Implementation Details and Experimental Results

Architecture	Iterations $(0-\text{phase})$	Iterations $(\xi-\text{phase})$	ϵ	ξ	α_1	α_2	α_3	α_4
$784-500-10$	20	4	0.5	1.0	0.1	0.05		
$784-500-500-10$	100	6	0.5	1.0	0.4	0.1	0.01	
$784-500-500-500-10$	500	8	0.5	1.0	0.128	0.032	0.008	0.002

Table 2: Hyperparameters. ϵ is the learning rate used for iterative inference (multiplies the energy gradient wrt states). ξ scales the amount of change of β_y. α_k is the learning rate for updating the parameters in layer k.

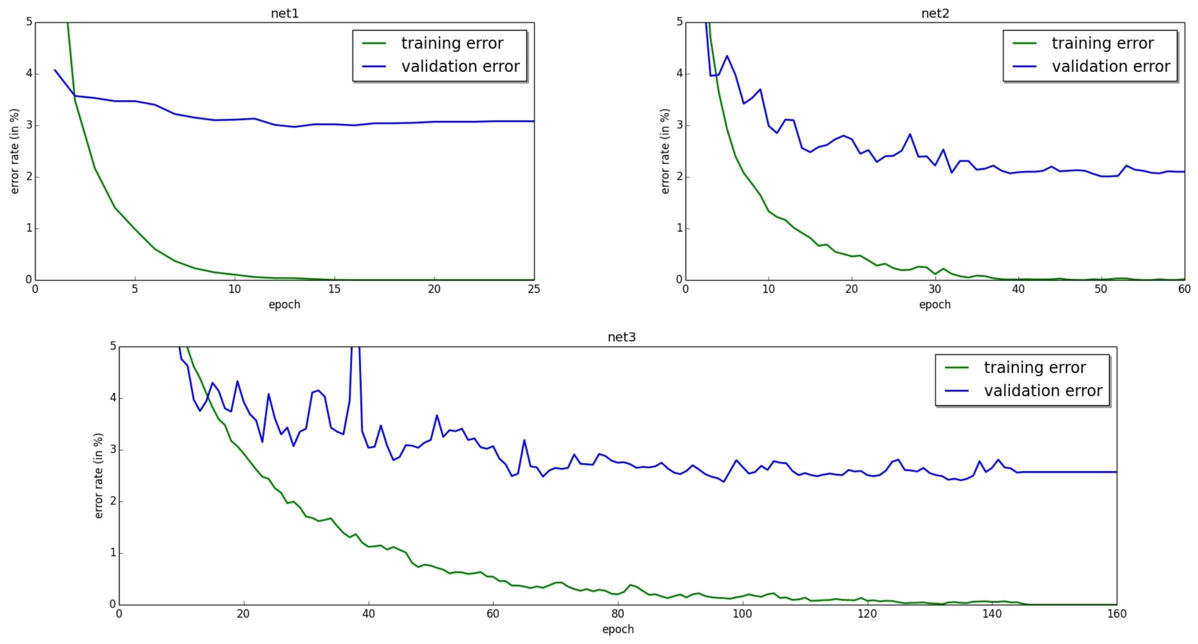


Figure 4: Training and validation error for neural networks with 1 hidden layer of 500 units (top left), 2 hidden layers of 500 units (top right), and 3 hidden layers of 500 units (bottom). The training error eventually decreases to 0.00% in all three cases.

To tackle the problem of the long 0-phase relaxation and speed-up the simulations, we use 'persistent particles' for the latent variables to re-use the previous fixed point configuration for a particular example as a starting point for the next 0-phase relaxation on that example. This means that for each training example in the dataset, we store the state of the hidden layers at the end of the negative phase, and we use this to initialize the state of the network at the next epoch. This method is similar in spirit to the PCD algorithm (Persistent Contrastive Divergence) for sampling from other energy-based models like the Boltzmann machine (Tieleman, 2008).

For reasons that are unclear yet, we find that it is important to choose different learning rates for the weight matrices of different layers, although the theory presented above guarantees that the algorithm should work with a unique learning rate for all synaptic weights. Let us denote by $L_0, L_1, \cdots, L_N$ the layers of the network (where $L_0 = x$ and $L_N = y$) and by W_k the weight matrix between the layers L_{k-1} and L_k. We choose the learning rate α_k for W_k so that the quantities $\frac{\|\Delta W_k\|}{\|W_k\|}$ for $k = 1, \cdots, N$ are approximately the same in average, where $\|\Delta W_k\|$ represents the weight change of W_k after seeing a minibatch.

The hyperparameters chosen for each model are shown in Table 2 and the results are shown in Figure 4. For efficiency of the experiments, we use minibatches of 20 training examples. Finally we initialize the weights according to the Glorot-Bengio initialization (Glorot and Bengio, 2010).

Acknowledgments

The authors would like to thank Akram Erraqabi, Alex Lamb, Alexandre Thiery, Mihir Mongia, Samira Shabanian and Asja Fischer for feedback and discussions, as well as NSERC, CIFAR, Samsung and Canada Research Chairs for funding, and Compute Canada for computing resources. We would also like to thank the developers of Theano [2], for developing such a powerful tool for scientific computing.

References

Almeida, L. B. (1987). A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In M. Caudill and C. Butler, editors, *IEEE International Conference on Neural Networks*, volume 2, pages 609–618, San Diego 1987. IEEE, New York.

Arora, S., Liang, Y., and Ma, T. (2015). Why are deep nets reversible: a simple theory, with implications for training. Technical report, arXiv:1511.05653.

Bengio, Y. and Fischer, A. (2015). Early inference in energy-based models approximates back-propagation. Technical Report arXiv:1510.02777, Universite de Montreal.

Bengio, Y., Mesnard, T., Fischer, A., Zhang, S., and Wu, Y. (2015a). STDP as presynaptic activity times rate of change of postsynaptic activity. arXiv:1509.05936.

Bengio, Y., Lee, D.-H., Bornschein, J., and Lin, Z. (2015b). Towards biologically plausible deep learning. arXiv:1502.04156.

Bengio, Y., Scellier, B., Bilaniuk, O., Sacramento, J., and Senn, W. (2016). Feedforward initialization for fast inference of deep generative networks is biologically plausible. *arXiv preprint arXiv:1606.01651*.

Berkes, P., Orban, G., Lengyel, M., and Fiser, J. (2011). Spontaneous cortical activity reveals hallmarks of an optimal internal model of the environment. *Science*, **331**, 83—87.

Bi, G. and Poo, M. (2001). Synaptic modification by correlated activity: Hebb's postulate revisited. *Annu. Rev. Neurosci.*, **24**, 139—166.

Friston, K. J. and Stephan, K. E. (2007). Free-energy and the brain. *Synthese*, **159**, 417—458.

Gerstner, W., Kempter, R., van Hemmen, J., and Wagner, H. (1996). A neuronal learning rule for sub-millisecond temporal coding. *Nature*, **386**, 76–78.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *AISTATS'2010*.

Hertz, J. A., Krogh, A., Lautrup, B., and Lehmann, T. (1997). Nonlinear backpropagation: doing backpropagation without derivatives of the activation function. *IEEE Transactions on neural networks*, **8**(6), 1321–1327.

[2]http://deeplearning.net/software/theano/

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, **14**, 1771–1800.

Hinton, G. E. and Sejnowski, T. J. (1986). Learning and releaming in boltzmann machines. *Parallel distributed processing: Explorations in the microstructure of cognition*, **1**, 282–317.

Hopfield, J. J. (1984). Neurons with graded responses have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences, USA*, **81**.

Lillicrap, T. P., Cownden, D., Tweed, D. B., and Akerman, C. J. (2014). Random feedback weights support learning in deep neural networks. arXiv:1411.0247.

Markram, H. and Sakmann, B. (1995). Action potentials propagating back into dendrites triggers changes in efficacy. *Soc. Neurosci. Abs*, **21**.

Movellan, J. R. (1990). Contrastive Hebbian learning in the continuous Hopfield model. In *Proc. 1990 Connectionist Models Summer School*.

O'Reilly, R. C. (1996). Biologically plausible error-driven learning using local activation differences: The generalized recirculation algorithm. *Neural Computation*, **8**(5), 895–938.

Pineda, F. J. (1987). Generalization of back-propagation to recurrent neural networks. *Pattern Recognition Letters*, **59**, 2229–2232.

Salakhutdinov, R. and Hinton, G. E. (2009). Deep Boltzmann machines. In *AISTATS'2009*, pages 448–455.

Sutskever, I. and Tieleman, T. (2010). On the Convergence Properties of Contrastive Divergence. In Y. W. Teh and M. Titterington, editors, *Proc. of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 9, pages 789–795.

Tieleman, T. (2008). Training restricted boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th international conference on Machine learning*, pages 1064–1071. ACM.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Machine Learning Res.*, **11**.

Xie, X. and Seung, H. S. (2000). Spike-based learning rules and stabilization of persistent neural activity. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 199–208. MIT Press.

Xie, X. and Seung, H. S. (2003). Equivalence of backpropagation and contrastive Hebbian learning in a layered network. *Neural Computation*.

Appendix

A/ Reformulation of the Theorem as a Constrained Optimization Problem

Here we give another proof for the gradient formula Eq. 11. Considering β, δ and v as fixed, and regarding θ and s as the free parameters, we can frame the training objective as the following constrained optimization problem:

$$\text{find} \quad \arg\min_{\theta,s} C_\beta^\delta(\theta,s,\mathrm{v}) \tag{54}$$

$$\text{subject to} \quad \frac{\partial F}{\partial s}(\theta,\beta,s,\mathrm{v}) = 0, \tag{55}$$

where the cost function $C_\beta^\delta(\theta,s,\mathrm{v})$ is given by Eq. 4. Note that in more conventional machine learning algorithms, one only optimizes θ, since the prediction and the cost function are *explicit* functions of θ. Here on the contrary, in the context of constrained optimization, the state s is regarded as belonging to the set of free parameters that we optimize, while the prediction and the cost function are *implicit* functions of θ through the constraint Eq. 55.

As usual for constrained optimization problems, we introduce the Lagrangian

$$L(\theta,s,\lambda) := \delta \cdot \frac{\partial F}{\partial \beta}(\theta,\beta,s,\mathrm{v}) + \lambda \cdot \frac{\partial F}{\partial s}(\theta,\beta,s,\mathrm{v}), \tag{56}$$

where λ is the vector of Lagrange multipliers. We have omitted the explicit dependence on β, δ and v in the notation $L(\theta, s, \lambda)$, since these variables are considered fixed. Starting from the current parameter θ, we first find s^* and λ^* such that

$$\frac{\partial L}{\partial \lambda}(\theta, s^*, \lambda^*) = 0 \tag{57}$$

and

$$\frac{\partial L}{\partial s}(\theta, s^*, \lambda^*) = 0, \tag{58}$$

and then we do one step of gradient descent on L with respect to θ, that is

$$\Delta\theta \propto -\frac{\partial L}{\partial \theta}(\theta, s^*, \lambda^*). \tag{59}$$

The first condition (Eq. 57) gives

$$\frac{\partial F}{\partial s}(\theta, \beta, s^*, \mathrm{v}) = 0 \quad \Rightarrow \quad s^* = s_{\theta,\mathrm{v}}^{\beta}. \tag{60}$$

Thus s^* is the 0-fixed point. Injecting this into the second condition (Eq. 58) we get

$$\delta \cdot \frac{\partial^2 F}{\partial \beta \partial s}\left(\theta, \beta, s_{\theta,\mathrm{v}}^{\beta}, \mathrm{v}\right) + \lambda^* \cdot \frac{\partial^2 F}{\partial s^2}\left(\theta, \beta, s_{\theta,\mathrm{v}}^{\beta}, \mathrm{v}\right) = 0. \tag{61}$$

Comparing Eq. 61 and Eq. 18, we conclude that

$$\lambda^* = \delta \cdot \frac{\partial s_{\theta,\mathrm{v}}^{\beta}}{\partial \beta}, \tag{62}$$

which is the directional derivative of the fixed point (as a function of β) at the point β in the direction δ. Injecting the values of s^* and λ^* in Eq. 56, we see that the Lagrangian $L(\theta, s^*, \lambda^*)$ represents the directional derivative of the energy function

$$\beta \mapsto F\left(\theta, \beta, s_{\theta,\mathrm{v}}^{\beta}, \mathrm{v}\right) \tag{63}$$

at the point β in the direction δ. Similarly, $\frac{\partial L}{\partial \theta}(\theta, s^*, \lambda^*)$ is the directional derivative of the function

$$\beta \mapsto \frac{\partial F}{\partial \theta}\left(\theta, \beta, s_{\theta,\mathrm{v}}^{\beta}, \mathrm{v}\right) \tag{64}$$

at the point β in the direction δ. Therefore Eq. 59 can be rewritten

$$\Delta\theta \propto -\lim_{\xi \to 0} \frac{1}{\xi}\left(\frac{\partial F}{\partial \theta}\left(\theta, \beta + \xi\delta, s_{\theta,\mathrm{v}}^{\beta+\xi\delta}, \mathrm{v}\right) - \frac{\partial F}{\partial \theta}\left(\theta, \beta, s_{\theta,\mathrm{v}}^{\beta}, \mathrm{v}\right)\right) \tag{65}$$

B/ Stochastic Framework

Rather than the deterministic dynamical system Eq. 34, a more likely dynamics would include some form of noise. As suggested by Bengio and Fischer (2015), injecting Gaussian noise in the gradient system Eq. 34 leads to a Langevin dynamics, which we write as the following stochastic differential equation:

$$ds = -\frac{\partial F}{\partial s}(\theta, \beta, s, \mathrm{v})dt + \sigma dB(t), \tag{66}$$

where $B(t)$ is a standard Brownian motion of dimension $\dim(s)$. In addition to the force $-\frac{\partial F}{\partial s}(\theta, \beta, s, \mathrm{v})dt$, the Brownian term $\sigma dB(t)$ models some form of noise in the network. For fixed θ, β and v, the Langevin dynamics Eq. 66 is known to converge to the Boltzmann distribution with temperature $T = \frac{1}{2}\sigma^2$ (consequence of the Fokker-Planck equation). For simplicity, here we assume that $\sigma = \sqrt{2}$, so that $T = 1$.

Let us denote by $p_{\theta,\mathrm{v}}^{\beta}$ the Boltzmann distribution corresponding to the energy function F. It is characterized by

$$p_{\theta,\mathrm{v}}^{\beta}(s) := \frac{e^{-F(\theta,\beta,s,\mathrm{v})}}{Z_{\theta,\mathrm{v}}^{\beta}}, \tag{67}$$

where $Z_{\theta,\mathrm{v}}^{\beta}$ is the partition function

$$Z_{\theta,\mathrm{v}}^{\beta}(s) := \int e^{-F(\theta,\beta,s,\mathrm{v})}. \tag{68}$$

Let us write $\mathbb{E}_{\theta,\mathrm{v}}^{\beta}$ the expectation over $s \sim p_{\theta,\mathrm{v}}^{\beta}(s)$. Similarly to the deterministic case, we define the objective function as

$$\widetilde{J}_{\beta}^{\delta}(\theta, \mathrm{v}) := \mathbb{E}_{\theta,\mathrm{v}}^{\beta}\left[C_{\beta}^{\delta}(\theta, s, \mathrm{v})\right], \tag{69}$$

where C_β^δ is the cost function (Eq. 4). As for the gradient on the objective function, the learning rule takes the form

$$\frac{d}{d\theta}\widetilde{J}_\beta^\delta\left(\theta, \mathrm{v}\right) = \lim_{\xi\to 0}\frac{1}{\xi}\left(\mathbb{E}_{\theta,\mathrm{v}}^{\beta+\xi\delta}\left[\frac{\partial F}{\partial \theta}\left(\theta, \beta+\xi\delta, s, \mathrm{v}\right)\right] - \mathbb{E}_{\theta,\mathrm{v}}^{\beta}\left[\frac{\partial F}{\partial \theta}\left(\theta, \beta, s, \mathrm{v}\right)\right]\right), \tag{70}$$

as a consequence of Theorem 3 below, which generalizes Theorem 1 to the stochastic framework.

Theorem 3 (Stochastic version)**.** *Let $F(\theta,\beta,s)$ be any energy function and p_θ^β the Boltzmann distribution defined by*

$$p_\theta^\beta(s) := \frac{e^{-F(\theta,\beta,s)}}{Z_\theta^\beta}, \tag{71}$$

where Z_θ^β is the partition function

$$Z_\theta^\beta(s) := \int e^{-F(\theta,\beta,s)}, \tag{72}$$

and $\mathbb{E}_\theta^\beta$ the expectation over $s \sim p_\theta^\beta(s)$. Then we have

$$\left(\frac{d}{d\theta}\mathbb{E}_\theta^\beta\left[\frac{\partial F}{\partial \beta}(\theta,\beta,s)\right]\right)^T = \frac{d}{d\beta}\mathbb{E}_\theta^\beta\left[\frac{\partial F}{\partial \theta}(\theta,\beta,s)\right]. \tag{73}$$

Just like in Theorem 1, the object on both sides of Eq. 73 is a matrix of size $\dim(\beta)\times\dim(\theta)$.

Proof. The differentials of the log partition function are equal to

$$\frac{d}{d\beta}\ln\left(Z_\theta^\beta\right) = -\mathbb{E}_\theta^\beta\left[\frac{\partial F}{\partial \beta}\left(\theta,\beta,s\right)\right] \tag{74}$$

and

$$\frac{d}{d\theta}\ln\left(Z_\theta^\beta\right) = -\mathbb{E}_\theta^\beta\left[\frac{\partial F}{\partial \theta}\left(\theta,\beta,s\right)\right]. \tag{75}$$

Therefore

$$\left(\frac{d}{d\theta}\mathbb{E}_\theta^\beta\left[\frac{\partial F}{\partial \beta}\left(\theta,\beta,s\right)\right]\right)^T = -\left(\frac{d}{d\theta}\frac{d}{d\beta}\ln\left(Z_\theta^\beta\right)\right)^T = -\frac{d}{d\beta}\frac{d}{d\theta}\ln\left(Z_\theta^\beta\right) = \frac{d}{d\beta}\mathbb{E}_\theta^\beta\left[\frac{\partial F}{\partial \theta}\left(\theta,\beta,s\right)\right]. \tag{76}$$

□

Finally we state a result similar to Proposition 2 in the stochastic framework.

Proposition 4 (Stochastic version)**.** *The derivative of the function*

$$\xi \mapsto \mathbb{E}_{\theta,\mathrm{v}}^{\beta+\xi\delta}\left[C_\beta^\delta(\theta,s,\mathrm{v})\right] \tag{77}$$

at $\xi = 0$ is non-positive.

Proof. The derivative of Eq. 77 at $\xi = 0$ is

$$-\mathbb{E}_{\theta,\mathrm{v}}^\beta\left[\left(C_\beta^\delta(\theta,s,\mathrm{v})\right)^2\right] + \left(\mathbb{E}_{\theta,\mathrm{v}}^\beta\left[C_\beta^\delta(\theta,s,\mathrm{v})\right]\right)^2 = -\mathrm{Var}_{\theta,\mathrm{v}}^\beta\left[C_\beta^\delta(\theta,s,\mathrm{v})\right] \leq 0, \tag{78}$$

where $\mathrm{Var}_{\theta,\mathrm{v}}^\beta$ represents the variance over $s \sim p_{\theta,\mathrm{v}}^\beta(s)$. □