Neural Networks 120 (2019) 9–31

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

2019 Special Issue

# Noise-boosted bidirectional backpropagation and adversarial learning

Olaoluwa Adigun, Bart Kosko *

*Department of Electrical and Computer Engineering, Signal and Image Processing Institute, University of Southern California, Los Angeles, CA 90089-2564, USA*

ARTICLE INFO

*Article history:*
Available online 17 October 2019

*Keywords:*
Bidirectional backpropagation
Neural networks
Noise benefit
Stochastic resonance
Expectation–Maximization algorithm
Bidirectional associative memory
Noise benefit

ABSTRACT

Bidirectional backpropagation trains a neural network with backpropagation in both the backward and forward directions using the same synaptic weights. Special injected noise can then improve the algorithm's training time and accuracy because backpropagation has a likelihood structure. Training in each direction is a form of generalized expectation–maximization because backpropagation itself is a form of generalized expectation–maximization. This requires backpropagation invariance in each direction: The gradient log-likelihood in each direction must give back the original update equations of the backpropagation algorithm. The special noise makes the current training signal more probable as bidirectional backpropagation climbs the nearest hill of joint probability or log-likelihood. The noise for injection differs for classification and regression even in the same network because of the constraint of backpropagation invariance. The backward pass in a bidirectionally trained classifier estimates the centroid of the input pattern class. So the feedback signal that arrives back at the input layer of a classifier tends to estimate the local pattern-class centroid. Simulations show that noise speeded convergence and improved the accuracy of bidirectional backpropagation on both the MNIST test set of hand-written digits and the CIFAR-10 test set of images. The noise boost further applies to regular and Wasserstein bidirectionally trained adversarial networks. Bidirectionality also greatly reduced the problem of mode collapse in regular adversarial networks.



## 1. Introduction: From adaptive resonance to noise-boosted bidirectional backpropagation and adversarial learning

What is a feedback signal that arrives at the input layer of a neural network?

Grossberg answered this question with his adaptive resonance theory or ART: The feedback signal is an *expectation* (Grossberg, 1976, 1982, 1988). The neural network expects to see this feedback signal or pattern given the current input signal that stimulated the network and given the pattern associations that it has learned. So any synaptic learning should depend on the match or mismatch between the input signal and the feedback expectation (Grossberg, 2017).

Grossberg gave this ART answer in the special case of a 2-layer neural network. The two layers defined the input and output fields of neurons. The two layers can also define two stacked contiguous layers of neurons in a larger network with multiple such stacked layers (Bengio et al., 2009). The topological point is that the neural signals flow bidirectionally between the two layers. There are no hidden or intervening layers. The synapses of the forward flow differ in general from the synapses of the backward flow.

A bidirectional associative memory or BAM results if the two synaptic webs are the same (Kosko, 1987, 1988, 1990, 1991a). A BAM is in this sense a minimal 2-layer neural network. The input signal passes forward through a synaptic matrix $M$. Then the backward signal passes through the transpose $M^T$ of the same matrix $M$. This minimal BAM structure holds for stacked contiguous layers that reverberate through the same weight matrix $M$ (Graves, Mohamed, & Hinton, 2013; Vincent, Larochelle, Lajoie, Bengio, & Manzagol, 2010).

The basic BAM stability theorem results if the network uses the transpose $M^T$ for the backward pass because then the forward and backward network Lyapunov functions are equal. The BAM theorem states that every real rectangular matrix $M$ is bidirectionally stable for threshold or sigmoidal neurons: Any input stimulation quickly leads to an equilibrium or resonating bidirectional fixed point of a fixed input vector and a fixed output vector. This BAM stability holds for the wide class of Cohen-Grossberg neuron nonlinearities (Cohen & Grossberg, 1983; Kosko, 1990, 1991a). It still holds even if the synaptic weights simultaneously change in accord with a Hebbian or competitive learning law (Kosko, 1991a). It also holds for time-lags and for many other recent extensions of the basic neuron models (Ali, Yogambigai, Saravanan, & Elakkia, 2019; Bhatia & Golman, 2019; Maharajan, Raja, Cao, Rajchakit, & Alsaedi, 2018; Wang, Chen, & Liu, 2018).

* Corresponding author.
*E-mail address:* kosko@usc.edu (B. Kosko).

https://doi.org/10.1016/j.neunet.2019.09.016
0893-6080/

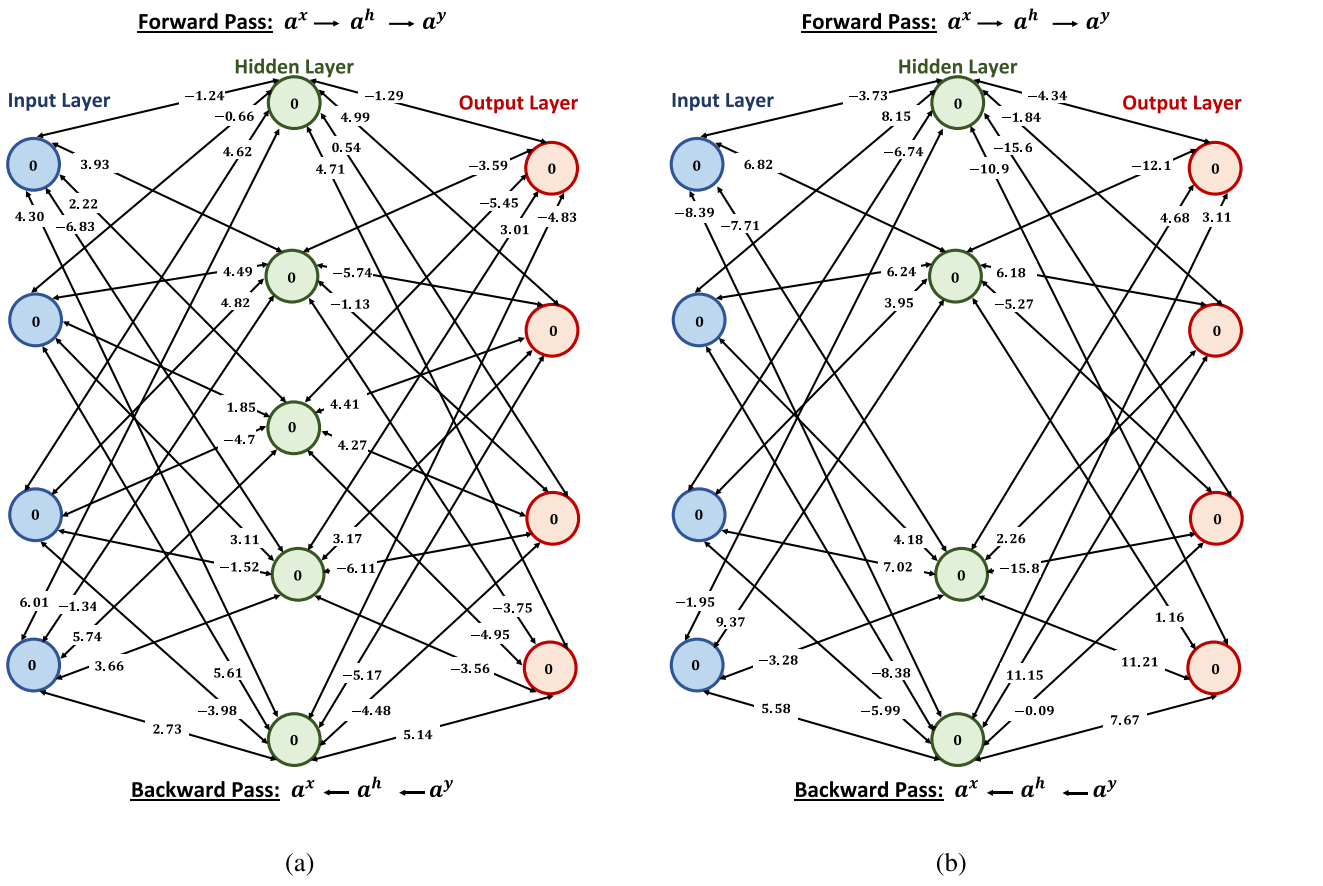



**Fig. 1.** Two bidirectional representations of the 4-bit bipolar permutation function in Table 1. (a) A 3-layer bidirectional associative memory (BAM) exactly represents the permutation function and its inverse. The forward or left-to-right direction encodes the permutation mapping. The backward or right-to-left direction encodes the inverse mapping. Noiseless bidirectional backpropagation trained the BAM network. The best set of weights and thresholds used a single hidden layer with 5 threshold neurons. (b) A *noise-boosted* 3-layer BAM that exactly represents the same permutation function and its inverse. Bidirectional backpropagation with NEM-noise injection trained the network. The NEM noise boost reduced the number of hidden neurons from 5 to 4.

Simple fixed-point stability need not hold in general when the BAM contains one or more hidden layers of nonlinear units. It does hold for the two 3-layer BAM networks in Fig. 1 because they represent the invertible permutation mapping in Table 1. Simulations showed that such multilayer feedback BAMs still often converged to fixed points. Fig. 3 and Tables 2–7 show how this convergence behavior varied with the number of hidden layers and the number of neurons per layer.

### 1.1. BAMs extend to multilayer networks

The ART and BAM concepts apply in the more general case when the network has any number of hidden or intervening neural layers between the input and output layers. This paper considers just such generalized or extended ART–BAM networks for *supervised* learning.

The extended BAM's feedback structure depends on the neural network's *reverse* mapping $N^T : R^K \rightarrow R^n$ from output vectors $\mathbf{y} \in R^K$ back to vectors $\mathbf{x}$ in the input pattern space $R^n$. Suppose that the vector input pattern $\mathbf{x} \in R^n$ stimulates the multilayer neural network $N$. It produces the vector output $\mathbf{y} = N(\mathbf{x}) \in R^K$. Then the feedback signal $\mathbf{x}'$ is the signal $N^T(\mathbf{y})$ that the network $N$ produces when it maps back from the output $\mathbf{y} = N(\mathbf{x})$ through its web of synapses to the input layer: $\mathbf{x}' = N^T(\mathbf{y}) = N^T(N(\mathbf{x}))$.

We use the notation $N^T(\mathbf{y})$ to denote this feedback signal. The backward direction uses only the transpose matrices $M^T$ of the synaptic matrices $M$ that the network $N$ uses in the forward direction. So these deep networks define generalized BAM networks with complete unidirectional sweeps from front to back and vice versa.

**Table 1**
4-bit bipolar permutation function (self-bijection) and its inverse that the 3-layer BAM networks represent in Figs. 1(a) and (b). The inverse simply maps from the output $y$ back to the corresponding input $x$.

| Input $x$ | Output $y$ |
|---|---|
| [− − −−] | [+ + −−] |
| [− − −+] | [+ + +−] |
| [− − +−] | [+ + ++] |
| [− − ++] | [+ + −+] |
| [− + −−] | [+ − −+] |
| [− + −+] | [− + −+] |
| [− + +−] | [+ − −−] |
| [− + ++] | [− + −−] |
| [+ − −−] | [+ − ++] |
| [+ − −+] | [− + ++] |
| [+ − +−] | [+ − +−] |
| [+ − ++] | [− + +−] |
| [+ + −−] | [− − +−] |
| [+ + −+] | [− − −−] |
| [+ + +−] | [− − −+] |
| [+ + ++] | [− − ++] |

The transpose notation $N^T$ also makes clear that the feedback or backward signal $N^T(\mathbf{y})$ differs from the *set-theoretic* inverse or pullback $N^{-1}(\mathbf{y})$. The pullback mapping $N^{-1} : 2^{R^K} \rightarrow 2^{R^n}$ always exists. It maps sets $B$ in the output space to sets $A$ in the input space: $A = N^{-1}(B) = \{\mathbf{x} \in R^n : N(\mathbf{x}) \in B\}$ if $B \subset R^K$. So the

**Table 2**
BAM stability without training. The network used no hidden neurons. The input–output configuration was $n \longleftrightarrow p$. There were $n$ input threshold neurons and $p$ output threshold neurons. We picked the BAM matrix values uniformly from the bipolar interval $[-1, 1]$.

| | Network configuration | % of convergence | Max run to converge | Max period for non-convergence |
|---|---|---|---|---|
| Bipolar | $5 \longleftrightarrow 5$ | 100% | 5 runs | 0 |
| | $10 \longleftrightarrow 10$ | 100% | 7 runs | 0 |
| | $50 \longleftrightarrow 50$ | 100% | 15 runs | 0 |
| | $100 \longleftrightarrow 100$ | 100% | 24 runs | 0 |
| | $500 \longleftrightarrow 500$ | 100% | 60 runs | 0 |
| Binary | $5 \longleftrightarrow 5$ | 100% | 6 runs | 0 |
| | $10 \longleftrightarrow 10$ | 100% | 10 runs | 0 |
| | $50 \longleftrightarrow 50$ | 100% | 13 runs | 0 |
| | $100 \longleftrightarrow 100$ | 100% | 20 runs | 0 |
| | $500 \longleftrightarrow 500$ | 100% | 30 runs | 0 |

**Table 3**
Stability of bidirectional associative memory (BAM) networks without training. We picked the synaptic weights uniformly in the bipolar interval $[-1, 1]$. The networks used either one or two hidden layers of threshold neurons and so did not always yield a BAM fixed point. The network configuration was $n \longleftrightarrow h_1 \longleftrightarrow p$ for networks with one hidden layer and $n \longleftrightarrow h_1 \longleftrightarrow h_2 \longleftrightarrow p$ for networks with two hidden layers. The term $n$ equaled the number of input neurons, $h_1$ was the size of the first hidden layer in terms of its threshold neurons, $h_2$ was the size of the second hidden layer, and $p$ was the number of output neurons. There were 50 hidden neurons in total.

| | Network configuration | % of convergence | Max run to converge | Max run for non-convergence |
|---|---|---|---|---|
| Bipolar | $5 \leftrightarrow 50 \leftrightarrow 5$ | 89.6% | 9 runs | 5 runs |
| | $10 \leftrightarrow 50 \leftrightarrow 10$ | 79.1% | 15 runs | 10 runs |
| | $50 \leftrightarrow 50 \leftrightarrow 50$ | 43.4% | 99 runs | 100 runs |
| | $100 \leftrightarrow 50 \leftrightarrow 100$ | 53.6% | 85 runs | 84 runs |
| | $500 \leftrightarrow 50 \leftrightarrow 500$ | 99.4% | 10 runs | 2 runs |
| Binary | $5 \leftrightarrow 50 \leftrightarrow 5$ | 79.9% | 9 runs | 9 runs |
| | $10 \leftrightarrow 50 \leftrightarrow 10$ | 62.5% | 22 runs | 19 runs |
| | $50 \leftrightarrow 50 \leftrightarrow 50$ | 37.5% | 99 runs | 100 runs |
| | $100 \leftrightarrow 50 \leftrightarrow 100$ | 57.8% | 91 runs | 99 runs |
| | $500 \leftrightarrow 50 \leftrightarrow 500$ | 99.4% | 10 runs | 2 runs |
| Bipolar | $5 \leftrightarrow 25 \leftrightarrow 25 \leftrightarrow 5$ | 81.3% | 8 runs | 99 runs |
| | $10 \leftrightarrow 25 \leftrightarrow 25 \leftrightarrow 10$ | 62.7% | 14 runs | 100 runs |
| | $50 \leftrightarrow 25 \leftrightarrow 25 \leftrightarrow 50$ | 44.6% | 44 runs | 100 runs |
| | $100 \leftrightarrow 25 \leftrightarrow 25 \leftrightarrow 100$ | 51.9% | 30 runs | 100 runs |
| | $500 \leftrightarrow 25 \leftrightarrow 25 \leftrightarrow 500$ | 54.7% | 26 runs | 100 runs |
| Binary | $5 \leftrightarrow 25 \leftrightarrow 25 \leftrightarrow 5$ | 63.6% | 11 runs | 99 runs |
| | $10 \leftrightarrow 25 \leftrightarrow 25 \leftrightarrow 10$ | 43.4% | 34 runs | 100 runs |
| | $50 \leftrightarrow 25 \leftrightarrow 25 \leftrightarrow 50$ | 50.8% | 71 runs | 100 runs |
| | $100 \leftrightarrow 25 \leftrightarrow 25 \leftrightarrow 100$ | 72.1% | 38 runs | 100 runs |
| | $500 \leftrightarrow 25 \leftrightarrow 25 \leftrightarrow 500$ | 99.5% | 12 runs | 100 runs |

set-theoretic inverse $N^{-1}(\mathbf{y}) = N^{-1}(\{\mathbf{y}\}) = A$ partitions the input pattern space $R^n$ into the two sets $A$ and $A^c$. All pattern vectors $\mathbf{x}$ in $A$ map to $\mathbf{y}$. All other inputs map elsewhere.

The *point* inverse $N^{-1} : R^K \to R^n$ exists only in the rare bijective case that the network $N : R^n \to R^K$ is both one-to-one and onto and $n = K$. The 3-layer BAM networks in Fig. 1(a)–(b) are just such rare cases of bijective networks because they exactly represent the permutation mapping in Table 1. The bidirectional learning algorithms below do not require that the networks $N$ have a point inverse $N^{-1}$.

Extending BAMs to multilayer networks entails relaxing BAM fixed-point stability in general. Stability still holds between any two contiguous fields that reverberate in isolation or subject only to fixed or slowly changing inputs from adjoining neural layers. Some of the figures below show how simple fixed-point BAM stability tends to fall off as the number of hidden layers increases. We often take as the network's output its first forward and backward results. We can also let the BAM reverberate in multiple back-and-forth sweeps before we record an equilibrium or quasi-equilibrium output.

### 1.2. The implicit bidirectionality of classifier networks

An important special case is the modern deep classifier network (Bishop, 2006; Jordan & Mitchell, 2015; LeCun, Bengio, & Hinton, 2015; Mohri, Rostamizadeh, & Talwalkar, 2018). These feedforward networks map images or videos or other patterns to $K$ softmax output neurons. The input neurons are most often identity functions because they act as data registers.

Classifier networks map patterns to probability vectors. A softmax output activation has the ratio form of an exponential divided by a sum of $K$ such exponentials. So the output vector $\mathbf{y} = N(\mathbf{x})$ defines a probability vector of length $K$. The network output $\mathbf{y}$ is a point in the $K$-1-dimensional simplex $S^{K-1} \subset R^K$.

Supervised learning for the classifier network uses 1-in-$K$ encoding for the $K$ standard basis or unit bit vectors $\mathbf{e}_1, \ldots, \mathbf{e}_K$. Input pattern $\mathbf{x} \in C_k \subset R^n$ should map to the corresponding $k$th unit bit vector $\mathbf{e}_k \in S^{K-1}$ if $C_k$ is the $k$th input pattern class. An ideal classifier emits $N(\mathbf{x}) = \mathbf{e}_k$ if and only if $\mathbf{x} \in C_k$. Then the $K$ set-theoretic pullbacks $N^{-1}(\mathbf{e}_\mathbf{k})$ of the $K$ basis vectors $\mathbf{e}_k$ partition the input pattern space $R^n$ into the $K$ desired pattern classes $C_k$: $R^n = \cup_{k=1}^n N^{-1}(\mathbf{e}_\mathbf{k}) = \cup_{k=1}^n C_k$.

A classifier network defines a bidirectional feedback system despite its feedforward mapping of patterns to probabilities. Most users simply ignore the backward pass and thus ignore the overall bidirectional structure. The reverse network $N^T : S^{K-1} \to R^n$ can always map probability descriptions $\mathbf{y}$ back to patterns in the input space through the reverse mapping $N^T$ by using the transpose of all synaptic weight matrices. This holds even when the classifier encodes time-varying patterns in simple recurrent loops among their hidden layers (Hochreiter & Schmidhuber, 1997).

**Table 4**
Stability of multilayered BAMs without training. We picked the synaptic weight values uniformly in the bipolar interval $[-1, 1]$. The networks used one or two hidden layers of threshold neurons. The configuration was $n \longleftrightarrow h_1 \longleftrightarrow p$ for BAM networks with one hidden layer and $n \longleftrightarrow h_1 \longleftrightarrow h_2 \longleftrightarrow p$ for BAM networks with two hidden layers. The term $n$ equaled the number of input threshold neurons, $h_1$ was the size of the first hidden layer in terms of its threshold neurons, $h_2$ was the size of the second hidden layer, and $p$ was the number of output threshold neurons. There were 200 hidden neurons in total.

| | Network configuration | % of convergence | Max run to converge | Max run for non-convergence |
|---|---|---|---|---|
| Bipolar | $5 \leftrightarrow 200 \leftrightarrow 5$ | 90.4% | 7 runs | 5 runs |
| | $10 \leftrightarrow 200 \leftrightarrow 10$ | 77.4% | 12 runs | 8 runs |
| | $50 \leftrightarrow 200 \leftrightarrow 50$ | 27.0% | 99 runs | 100 runs |
| | $100 \leftrightarrow 200 \leftrightarrow 100$ | 2.0% | 99 runs | 100 runs |
| | $500 \leftrightarrow 200 \leftrightarrow 500$ | 1.9% | 99 runs | 100 runs |
| | $1000 \leftrightarrow 200 \leftrightarrow 1000$ | 54.5% | 99 runs | 100 runs |
| Binary | $5 \leftrightarrow 200 \leftrightarrow 5$ | 90.4% | 8 runs | 6 runs |
| | $10 \leftrightarrow 200 \leftrightarrow 10$ | 61.7% | 25 runs | 12 runs |
| | $50 \leftrightarrow 200 \leftrightarrow 50$ | 3.1% | 98 runs | 100 runs |
| | $100 \leftrightarrow 200 \leftrightarrow 100$ | 0.0% | No convergence | 100 runs |
| | $500 \leftrightarrow 200 \leftrightarrow 500$ | 1.8% | 99 runs | 100 runs |
| | $1000 \leftrightarrow 200 \leftrightarrow 1000$ | 54.5% | 99 runs | 100 runs |
| Bipolar | $5 \leftrightarrow 100 \leftrightarrow 100 \leftrightarrow 5$ | 78.7% | 9 runs | 100 runs |
| | $10 \leftrightarrow 100 \leftrightarrow 100 \leftrightarrow 10$ | 55.2% | 13 runs | 100 runs |
| | $50 \leftrightarrow 100 \leftrightarrow 100 \leftrightarrow 50$ | 2.2% | 90 runs | 100 runs |
| | $100 \leftrightarrow 100 \leftrightarrow 100 \leftrightarrow 100$ | 0.9% | 96 runs | 100 runs |
| | $500 \leftrightarrow 100 \leftrightarrow 100 \leftrightarrow 500$ | 1.5% | 96 runs | 100 runs |
| | $1000 \leftrightarrow 100 \leftrightarrow 100 \leftrightarrow 1000$ | 2.3% | 99 runs | 100 runs |
| Binary | $5 \leftrightarrow 100 \leftrightarrow 100 \leftrightarrow 5$ | 61.3% | 11 runs | 100 runs |
| | $10 \leftrightarrow 100 \leftrightarrow 100 \leftrightarrow 10$ | 30.3% | 30 runs | 100 runs |
| | $50 \leftrightarrow 100 \leftrightarrow 100 \leftrightarrow 50$ | 0.0% | No convergence | 100 runs |
| | $100 \leftrightarrow 100 \leftrightarrow 100 \leftrightarrow 100$ | 0.0% | No convergence | 100 runs |
| | $500 \leftrightarrow 100 \leftrightarrow 100 \leftrightarrow 500$ | 13.3% | 99 runs | 100 runs |
| | $1000 \leftrightarrow 100 \leftrightarrow 100 \leftrightarrow 1000$ | 82.0% | 89 runs | 100 runs |

**Table 5**
Classification accuracies of multilayer BAMs trained on the MNIST dataset of handwritten digits. We computed both the one-shot classification accuracy and the long-shot accuracy over 100 bidirectional sweeps through the trained networks.

| | Hidden layer configuration | One-shot accuracy | Long-shot accuracy | Long-shot squared error |
|---|---|---|---|---|
| Bipolar | 1024 | 98.37% | 98.21% | 0.0 |
| | $512 \longleftrightarrow 512$ | 98.15% | 98.17% | 0.0 |
| | $256 \longleftrightarrow 256 \longleftrightarrow 256 \longleftrightarrow 256$ | 97.67% | 97.35% | 0.0 |
| Binary | 1024 | 98.47% | 98.40% | 0.0 |
| | $512 \longleftrightarrow 512$ | 98.49% | 98.45% | 0.0 |
| | $256 \longleftrightarrow 256 \longleftrightarrow 256 \longleftrightarrow 256$ | 97.90% | 97.85% | 0.0 |

**Table 6**
Classification accuracies of multilayer BAMs trained on the CIFAR-10 dataset. We computed both the one-shot accuracy and the long-shot accuracy over 100 bidirectional sweeps through the trained networks.

| | Hidden layer configuration | One-shot accuracy | Long-shot accuracy | Long-shot squared error |
|---|---|---|---|---|
| Bipolar | 1024 | 55.65% | 54.3% | $2.1 \times 10^{-6}$ |
| | $512 \longleftrightarrow 512$ | 53.54% | 42.27% | $4.6 \times 10^{-6}$ |
| | $256 \longleftrightarrow 256 \longleftrightarrow 256 \longleftrightarrow 256$ | 50.38% | 27.76% | $2.8 \times 10^{-5}$ |
| Binary | 1024 | 55.26% | 52.61% | 0.0 |
| | $512 \longleftrightarrow 512$ | 52.28% | 33.33% | $5.3 \times 10^{-6}$ |
| | $256 \longleftrightarrow 256 \longleftrightarrow 256 \longleftrightarrow 256$ | 48.09% | 26.02% | $1.9 \times 10^{-6}$ |

**Table 7**
Centroid analysis for the backward pass of a multilayered BAM with MNIST handwritten digits dataset. We compared the Euclidean distance $d^{(t)}$ between the backward-pass vector $\mathbf{x} = N^T(\mathbf{y})$ at time $t$ and the sample class centroids over bidirectional sweeps through the trained network. The simulations used $d_{max} = \max_{1 \leq t \leq 100} d^{(t)}$, $d_{min} = \min_{1 \leq t \leq 100} d^{(t)}$, and $\bar{d} = \frac{1}{N} \sum_{t=1}^{100} d^{(t)}$.

| | Hidden layer configuration | Correct classification | | | Misclassification | | |
|---|---|---|---|---|---|---|---|
| | | Min ($d_{min}$) | Max ($d_{max}$) | Mean ($\bar{d}$) | Min ($d_{min}$) | Max ($d_{max}$) | Mean ($\bar{d}$) |
| Bipolar | 1024 | $-1.0 \times 10^{-2}$ | 2.4965 | $5.8 \times 10^{-3}$ | −0.2704 | 4.1620 | 2.3864 |
| | $512 \longleftrightarrow 512$ | $-3.3 \times 10^{-2}$ | 4.0669 | $3.0 \times 10^{-3}$ | −0.4219 | 3.4808 | 0.2280 |
| | $256 \longleftrightarrow 256 \longleftrightarrow 256 \longleftrightarrow 256$ | $-2.1 \times 10^{-2}$ | 0.9085 | $0.0014 \times 10^{-3}$ | −0.6152 | 2.7178 | 0.0856 |
| Binary | 1024 | $-2.1 \times 10^{-2}$ | 3.6311 | $8.3 \times 10^{-3}$ | −3.2937 | 0.4729 | −0.4167 |
| | $512 \longleftrightarrow 512$ | $-8.1 \times 10^{-3}$ | 5.0859 | $4.0 \times 10^{-3}$ | −1.8951 | 2.6759 | −0.0700 |
| | $256 \longleftrightarrow 256 \longleftrightarrow 256 \longleftrightarrow 256$ | $-7.2 \times 10^{-3}$ | 4.3520 | $0.0052 \times 10^{-3}$ | −4.9765 | 1.3177 | −3.1570 |

The reverse pass through $N^T$ can thereby answer *why*-type causal questions: Why did this observed output occur? What type of input pattern caused this result?

Pattern answers to these why-type questions are versions of Grossberg's network feedback expectations. The images in Fig. 5 show such answers or expectations from a deep BAM network

with 7 hidden layers. The bidirectional network trained on CIFAR-10 images. The convolutional classifier in Fig. 6 shows similar reverse-pass images $N^T(\mathbf{y})$ or $N^T(\mathbf{e}_k)$ from a deep convolutional network in BAM mode. Simple unidirectional backpropagation training produced the uninformative reverse-pass images in the (b) panels of both figures. Proper bidirectional training with the B-BP algorithm below produced reverse-pass images that closely match the sample centroids of the pattern classes.

The forward pass through the network $N$ answers comparatively simpler *what-if* questions: What happens if this input stimulates the system? What effect will this input cause? Bidirectional processing can help even here as many of the simulations show. Modern unidirectional classifiers $N$ simply ignore the reverse mapping information housed in the transpose matrices $M^T$ of the reverse pass $N^T$. The equilibrium forward pass through the network $N$ differs in general from the first forward pass through $N$. Modern feedforward classifiers simply take this first forward pass as the network's final output.

This implied BAM feedback structure requires only that output vectors $\mathbf{y} \in S^{K-1}$ pass back through the transpose matrices of all the synaptic matrices that the forward pass used to produce a given output from a given input. This backward pass also requires a related backward pass through any windows or masks in convolutional classifiers (Fukushima, 1980; Krizhevsky, Sutskever, & Hinton, 2012; LeCun, Bengio, et al., 1995). The last section shows how the simulations passed output information backward through the convolutional filters of deep convolutional networks.

Then even a convolutional classifier can produce a feedback signal $\mathbf{x}' = N^T(\mathbf{y})$ at the input layer (and at any hidden layer). The feedback signal $\mathbf{x}'$ looks like a handwritten digit if the classifier network trains on the MNIST digit data. It looks like some other image if it trains on the CIFAR image test set. So we can train the network in the backward direction $N^T$ if we compute some form of error based on the triggering input pattern $\mathbf{x} \in C_k$ and based on what the network expects to see $\mathbf{x}' = N^T(\mathbf{y}) = N^T(N(\mathbf{x}))$ in the Grossberg sense of ART. This insight leads to the bidirectional supervised learning algorithm below.

We also point out that the backward direction in such classifiers performs a form of statistical regression. This holds because the input neurons are identity units. It does not hold if the input neurons have logistic or other nonlinear form. The regression structure leads to an optimal mean-squared structure in the backward direction as we explain below.

The regression structure in the backward direction dictates both a different training regimen and a different noise-injection algorithm from the forward classifier. The backward tendency toward mean-squared estimation implies that such classifiers estimate the $K$ class centroids $\mathbf{c}_k$ in the backward direction. So the feedback signal $\mathbf{x}' = N^T(\mathbf{y})$ shows what the classifier network expects to "see" or perceive at the input field given the current pattern stimulus $\mathbf{x}$: It shows what the network expects the local class centroid to look like. That explains why the backward-pass signals in Figs. 5 and 6 so closely match the sample class centroids in their (a) panels.

### 1.3. Bidirectional backpropagation through BP invariance

The new *bidirectional backpropagation* (B-BP) algorithm extends these ART and BAM concepts to supervised learning in multilayer networks (Adigun & Kosko, 2016, 2019). B-BP trains the neural network in both the forward and backward directions using the *same* weights. Using the same weights entails using the matrix transposes $M^T$ in the backward direction for any synaptic weight matrices $M$ that the network $N$ uses in the forward direction. So this bidirectional processing through the backward-pass network $N^T$ converts feedforward networks $N$ into BAM networks. The B-BP algorithm shows how to extend ordinary unidirectional BP (Rumelhart, Hinton, & Williams, 1986; Werbos, 1974) without overwriting in either direction.

The added computational cost of B-BP is slight compared with ordinary or forward-only BP. This holds because the time complexity for BP in either direction is $O(n)$ for $n$ input–output training samples. The complexity in the B-BP case remains $O(n)$ because $O(n) + O(n) = O(n)$.

Fig. 1 shows two such 3-layer BAM networks after B-BP training. All neurons are threshold on–off neurons with zero threshold. Each feedback network exactly represents the same 4-bit bipolar permutation mapping and its inverse from Table 1. Each network maps a 4-bit vector $\mathbf{x} \in \{-1, 1\}^4$ to a 4-bit vector $\mathbf{y} \in \{-1, 1\}^4$. Each network's backward direction maps $\mathbf{y}$ back to the same input $\mathbf{x}$. So the input $\mathbf{x} = (-1, 1, 1, 1)$ maps to $\mathbf{y} = (-1, 1, -1, -1)$ and conversely. So the vector pair $(\mathbf{x}, \mathbf{y})$ is a BAM fixed point of each network. This holds for all 16 vector pairs in Table 1 for each BAM network. There are more than 20 quadrillion such 4-bit permutation mappings and associated inverse mappings because there are $2^4!$ or 16! ways to permute the 16 input vectors in Table 1.

The two BAM networks differ both in their synaptic values and in their number of hidden neurons. The BAM network in Fig. 1(a) needs 5 hidden threshold neurons to learn the permutation mapping in Table 1 with B-BP. Simulations with 4 or fewer hidden neurons failed to find a representation. The BAM network in Fig. 1(b) needs only 4 hidden neurons because the proper additive noise boosted the B-BP learning as we will show below. The noise obeyed the sufficient condition for a B-BP noise boost in Theorem 4. The neurons were steep logistic functions during B-BP learning. We rounded them off to threshold neurons with zero thresholds after learning.

An earlier theorem showed that a 3-layer threshold BAM network can always exactly represent an $n$-bit permutation and its inverse if the network uses $2^n$ hidden threshold neurons (Adigun & Kosko, 2016, 2019). That architecture would require 16 hidden neurons in this case. So the B-BP algorithm reduced this exponential number of hidden neurons in $n$ to a linear number of hidden neurons.

The main problem with bidirectional learning is overwriting. Learning in one direction tends to overwrite or undo learning in the other direction. This may explain why users have applied BP almost exclusively in the forward direction for classifiers or regressors. Running simple BP in the backward direction will only degrade the prior learning in the forward direction.

B-BP solves the overwriting problem with two performance measures. Each direction gets its own performance measure based on whether that direction performs classification or regression or some other task. B-BP sums the two error or log-likelihood performance measures. The overall forward-and-backward sweeps minimize the network's total log-likelihood.

Fig. 2 shows how the two performance measures overcome the problem of overwriting for the two 3-layer BAMs in Fig. 1. The first two sub-figures in Fig. 2 show that learning in one direction overwrites learning in the other direction with the same performance measure. The third figure shows rapid learning in both directions because it used the sum of the two correct performance measures. The performance measures were correct in the sense that they preserved the BP learning equations for a given choice of output-neuron structure and directional functionality. We call this *backpropagation invariance*.

BP invariance requires that the log-likelihoods must correspond to the functions that the forward and backward passes perform. A deep classifier gives a canonical example. It uses a cross-entropy performance measure for its forward pass of

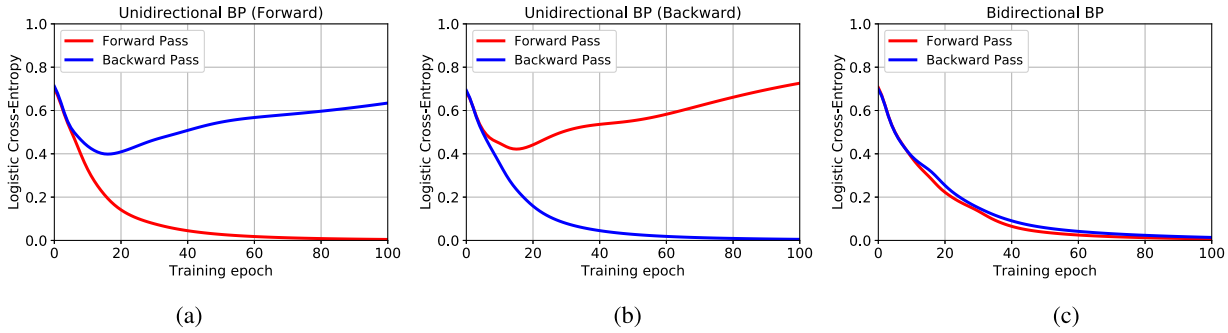



**Fig. 2.** Backpropagation overwrites in one direction as it trains in the other direction if it uses a single performance measure or error function. Bidirectional backpropagation avoids overwriting because it uses the sum of two error functions that match the terminal neuron structure and functionality of each direction. Simulations used NEM noise injection in each direction for the 3-layer BAM network in Fig. 1(b) that learned the 4-bit bipolar permutation in Table 1 and its inverse. The forward and backward performance measures were each a double cross entropy in Eq. (28) because the input and output neurons were logistic (later rounded-off to threshold neurons). (a) Unidirectional BP training on the forward pass reduced the forward error but overwrites the learning of the inverse map in the backward direction. (b) Unidirectional BP training on the backward pass reduced the backward training error but overwrites the learning of the permutation map in the forward direction. (c) NEM noise injection with bidirectional BP trained the network in accord with the logistic NEM condition in Eq. (58) and with the sum of the forward and backward cross entropies. There was no over-writing in either direction. The NEM noise injection speeded convergence in both directions.

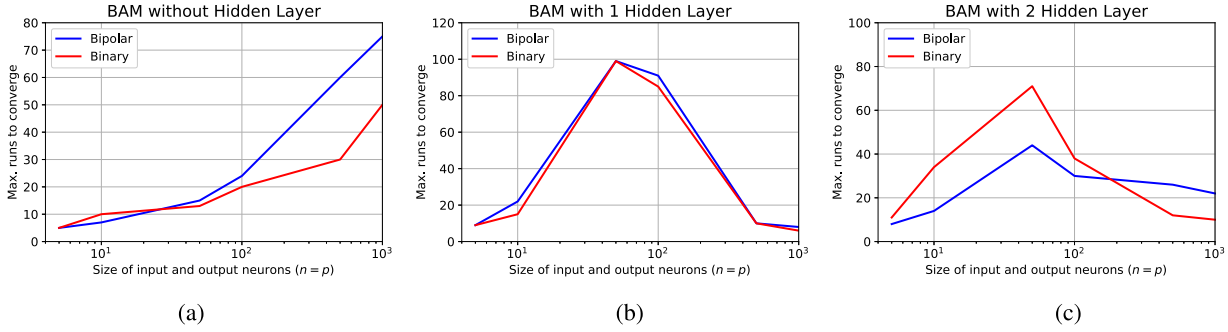



**Fig. 3.** Maximum number of runs or iterations that a BAM needed to converge if it did converge. BAMs with no hidden layers always converged in accord with the basic BAM theorem. We varied the size of the threshold BAM networks and compared bipolar encoding with binary encoding. We picked all synaptic weights uniformly from the bipolar interval $[-1, 1]$. (a) No hidden layer (classical BAM). (b) One hidden layer of threshold neurons. (c). Two hidden layers of threshold neurons.

classification. The classifier implicitly uses a squared-error performance measure for its backward pass of regression. The overwhelming number of classifiers in practice simply ignore this backward-pass structure.

B-BP's sum of performance measures stems from the bidirectional network's total log-likelihood structure. The forward direction has the likelihood function $p(\mathbf{y}|\mathbf{x}, \Theta)$ for input vector $\mathbf{x}$ and output vector $\mathbf{y}$ and all network parameters $\Theta$. The backward direction has the converse likelihood function $p(\mathbf{x}|\mathbf{y}, \Theta)$. This likelihood structure holds in general at the layer level so long as the layer neurons and performance measure obey BP invariance.

The back-*and*-forth structure of bidirectionality gives the joint or total likelihood function as the product $p(\mathbf{y}|\mathbf{x}, \Theta)p(\mathbf{x}|\mathbf{y}, \Theta)$. So the network log-likelihood $L$ is just the sum $L = \log p(\mathbf{y}|\mathbf{x}, \Theta) + \log p(\mathbf{x}|\mathbf{y}, \Theta)$. The negative log-likelihoods define the forward and backward error functions or performance measures. Then the gradient $\nabla L = \nabla \log p(\mathbf{y}|\mathbf{x}, \Theta) + \nabla \log p(\mathbf{x}|\mathbf{y}, \Theta)$ leads to the B-BP algorithm and its noise-boost by way of the generalized EM algorithm.

The forward direction of the classifier uses the cross-entropy between the desired output target distribution and the actual $K$ softmax outputs. The cross-entropy is just the negative of the logarithm of a one-trial multinomial probability density $p(\mathbf{y}|\mathbf{x}, \Theta)$. So the statistical structure of the forward pass corresponds to rolling a $K$-sided die. The backward pass uses the squared-error at the input. This arises from taking the logarithm of a vector normal distribution $p(\mathbf{x}|\mathbf{y}, \Theta)$. It also implies that the backward learning estimates the $k$th local pattern-class centroid because the centroid minimizes the squared error.

The key point is that BP invariance must hold for proper bidirectional learning: The network can perform classification or regression or any other function that leaves the backpropagation likelihood structure invariant. The bidirectional network itself can perform different functions in different directions. Its directional likelihood structure then allows noise boosting in those different directions.

### 1.4. NEM noise-boosted bidirectional backpropagation

We show that injecting carefully chosen noise (not blind noise) into the input and output layers both speeds the convergence of B-BP and improves its accuracy. This special noise is just that noise that makes the signal more probable. It is NEM or Noisy EM noise because the ordinary unidirectional backpropagation algorithm turns out to be a special case of the generalized expectation–maximization algorithm (Audhkhasi, Osoba, & Kosko, 2016) and because this NEM noise always speeds the EM algorithm's ascent up the nearest hill of probability (Osoba & Kosko, 2016). We can also inject NEM noise into hidden units. This may involve more involved matrix transformations.

We develop below the NEM noise inequalities that apply to B-BP training. These theoretical conditions follow the joint forward-and-backward log-likelihood structure of B-BP itself. The NEM noise is again just that noise $\mathbf{n}$ that makes the current signal $\mathbf{y}$ more probable: $p(\mathbf{y}|\mathbf{n}) \geqslant p(\mathbf{y})$ on average. This simple inequality leads to an average likelihood ratio that suffices for an average noise-boost. Boosting B-BP injects this noise into the output (or hidden) neurons at each likelihood-gradient step for the joint or

bidirectional log-likelihood $L = \log p(\mathbf{y}|\mathbf{x}, \Theta) + \log p(\mathbf{x}|\mathbf{y}, \Theta)$. This summed log-likelihood $L$ also gives a general proof strategy for B-BP results: Prove the result in each direction and then combine directions. So we will review and extend the earlier unidirectional results for BP (Audhkhasi et al., 2016; Kosko, Audkhasi, & Osoba, 2019) and then apply them as lemmas to B-BP.

We show last how to apply B-BP to generative adversarial networks (GANs) and how to boost their performance with bidirectional NEM noise. Adversarial networks combine at least two networks that train while they try to deceive each other. The generator network tends to generate ever better fake patterns from training patterns while the discriminator network tries to detect the fake patterns. We test these bidirectional results on the standard vanilla and Wasserstein GANs using the MNIST and CIFAR-10 image datasets. Bidirectional BP largely removed the problem of mode collapse in the standard GANs. Mode collapse is a type of training stutter where the network keeps emitting the same pattern. Wasserstein GANs tend to avoid the problem of mode collapse but at the cost of increased computation. NEM noise further improved GAN performance.

The overall insight is that neural feedback matters. Feedback can help a neural system as well as hurt it. The same is true of noise. They both require careful analysis if a user wants to apply them to a given nonlinear neural system.

## 2. Backpropagation and noise injection

The BP algorithm trains a neural network to approximate some mapping from the input space $X$ to the output space $Y$ (Bishop, 2006; Hinton, Rumelhart, & Williams, 1986; Jordan & Mitchell, 2015; LeCun et al., 2015; Werbos, 1974). The mapping is a simple function for an ideal classifier. The BP algorithm is itself a special case of generalized expectation–maximization (GEM) for maximum-likelihood estimation with latent or hidden parameters (Audhkhasi et al., 2016).

The reduction of GEM to BP follows from the key gradient identity $\nabla \log p(\mathbf{y}|\mathbf{x}, \Theta) = \nabla Q(\Theta|\Theta^n)$ that we re-derive below in (9). The BP noise benefit follows in turn from the EM-noise-boost results in Osoba and Kosko (2016). These EM results show that injecting noise helps the maximum-likelihood system bound faster up the nearest hill of probability on average if the noise satisfies a positivity condition that involves a likelihood ratio. We next review these results and then extend them to the bidirectional case.

### 2.1. Backpropagation invariance and expectation–maximization

BP training has a probabilistic structure based on maximum-likelihood estimation. BP trains a network by iteratively minimizing some error or performance measure $E(\Theta)$ that depends on the difference between the network's actual or observed output value $\mathbf{y} = N(\mathbf{x})$ and its desired or target $\mathbf{t}$. The parameter vector $\Theta$ describes the network's current configuration of synaptic weights and neuronal coefficients. There is some probability $p(\mathbf{y}|\mathbf{x}, \Theta)$ that a neural network $N$ with parameters $\Theta$ will emit output $\mathbf{y}$ given input $\mathbf{x}$. The probability $p(\mathbf{y}|\mathbf{x}, \Theta)$ defines the network's output-layer likelihood function.

*BP invariance* requires that the network error $E(\Theta)$ equals the negative of the network's log-likelihood function $L$: $E(\Theta) = -L = -\log p(\mathbf{y}|\mathbf{x}, \Theta)$. Then minimizing the network error $E(\Theta)$ maximizes the network log-likelihood $\log p(\mathbf{y}|\mathbf{x}, \Theta)$ and vice versa. So BP performs maximum-likelihood estimation (Bishop, 2006):

$$\Theta^* = \arg\min_{\Theta} E(\Theta) = \arg\max_{\Theta} \log\, p(\mathbf{y}|\mathbf{x}, \Theta). \tag{1}$$

BP invariance helps explain why BP is a special case of generalized expectation–maximization. EM itself generalizes maximum likelihood to the case of hidden variables or missing data (Dempster, Laird, & Rubin, 1977). EM iteratively climbs the nearest hill of probability or log-likelihood as it alternates between a forward expectation step and a backward maximization step. The forward step computes or estimates an expectation with respect to the posterior density of the hidden variables given the current data and parameter estimates. The expectation defines a surrogate likelihood function $Q$. The backward step maximizes this surrogate likelihood given the current data and given the current estimate of the parameters. Generalized EM replaces the complete maximization of the surrogate with a gradient estimate or partial maximization. So both BP and generalized EM are back-and-forth gradient algorithms that find local maximum-likelihood parameters. The formal argument below that BP equals generalized EM shows how entropy minimization achieves this result if BP invariance holds.

BP invariance holds in particular for the two common cases of classification and regression. Classification requires both that the error function $E(\Theta)$ be cross entropy and that the output neurons have a softmax or other nonlinear form that produces a one-shot multinomial likelihood $p(\mathbf{y}|\mathbf{x}, \Theta)$. Regression requires instead that $E(\Theta)$ be squared error and that the output neurons are linear or identity neurons. Then the likelihood $p(\mathbf{y}|\mathbf{x}, \Theta)$ must equal a multidimensional Gaussian density. So its negative log-likelihood $-L$ just equals the squared error $E(\Theta)$. Then both the classifier and regression networks have the *same* BP learning laws.

BP is a special case of the GEM algorithm because the gradient of the network (layer) likelihood $\log p(\mathbf{y}|\mathbf{x}, \Theta)$ equals the gradient of EM's surrogate likelihood function $Q(\Theta|\Theta^n)$ (Audhkhasi et al., 2016): $\nabla_\Theta \log p(\mathbf{y}|\mathbf{x}, \Theta^{(i)}) = \nabla_\Theta Q(\Theta^{(i)}|\Theta^{(i)})$ at each iteration $i$ for the network's weight parameters $\Theta^{(i)}$ and input $\mathbf{x}$.

We now restate the recent BP-as-GEM theorem (Audhkhasi et al., 2016) and then sketch its proof. The theorem states that the backpropagation update equation for a differentiable likelihood function $p(\mathbf{y}|\mathbf{x}, \Theta)$ at epoch $i$:

$$\Theta^{(i+1)} = \Theta^{(i)} + \eta \nabla_\Theta \log\, p(\mathbf{y}|\mathbf{x}, \Theta)\Big|_{\Theta=\Theta^{(i)}} \tag{2}$$

equals the GEM update equation at epoch $i$

$$\Theta^{(i+1)} = \Theta^{(i)} + \eta \nabla_\Theta Q(\Theta|\Theta^{(i)})\Big|_{\Theta=\Theta^{(i)}} \tag{3}$$

if GEM uses the differentiable Q-function

$$Q(\Theta|\Theta^i) = \mathbb{E}_{\mathbf{h}|\mathbf{y},\mathbf{x},\Theta^i}\Big[\log\, p(\mathbf{h}, \mathbf{y}|\mathbf{x}, \Theta)\Big]. \tag{4}$$

The EM algorithm takes the expectation of $\log p(\mathbf{y}|\mathbf{x}, \Theta)$ with respect to the hidden posterior $p(\mathbf{h}|\mathbf{y}, \mathbf{x}, \Theta^{(i)})$. The "EM trick" rewrites the conditional probability $p(\mathbf{h}|\mathbf{y}, \mathbf{x}, \Theta) = \frac{p(\mathbf{h},\mathbf{y}|\mathbf{x},\Theta)}{p(\mathbf{y}|\mathbf{x},\Theta)}$ as $p(\mathbf{y}|\mathbf{x}, \Theta) = \frac{p(\mathbf{h},\mathbf{y}|\mathbf{x},\Theta)}{p(\mathbf{h}|\mathbf{y},\mathbf{x},\Theta)}$. Then taking hidden-posterior expectations of the log-likelihood $\log p(\mathbf{y}|\mathbf{x}, \Theta)$ gives

$$\log p(\mathbf{y}|\mathbf{x}, \Theta) = \mathbb{E}_{\mathbf{h}|\mathbf{y},\mathbf{x},\Theta^{(i)}}\left[\log\, p(\mathbf{y}|\mathbf{x}, \Theta)\right] \tag{5}$$

$$= \mathbb{E}_{\mathbf{h}|\mathbf{y},\mathbf{x},\Theta^{(i)}}\left[\log \frac{p(\mathbf{h}, \mathbf{y}|\mathbf{x}, \Theta)}{p(\mathbf{h}|\mathbf{y}, \mathbf{x}, \Theta)}\right] \tag{6}$$

$$= Q(\Theta|\Theta^{(i)}) + H(\Theta|\Theta^{(i)}) \tag{7}$$

if $Q(\Theta|\Theta^{(i)})$ is the EM surrogate likelihood with cross entropy $H(\Theta|\Theta^{(i)}) = -\mathbb{E}_{\mathbf{h}|\mathbf{y},\mathbf{x},\Theta^{(i)}}[\log p(\mathbf{h}|\mathbf{y}, \mathbf{x}, \Theta)]$. Taking gradients gives

$$\nabla \log p(\mathbf{y}|\mathbf{x}, \Theta) = \nabla Q\left(\Theta|\Theta^{(i)}\right) + \nabla H\left(\Theta|\Theta^{(i)}\right). \tag{8}$$

The entropy inequality $H(\Theta^{(i)}|\Theta^{(i)}) \leqslant H(\Theta|\Theta^{(i)})$ holds for all $\Theta$ because Jensen's inequality and the concavity of the logarithm imply that Shannon entropy $H(\Theta^{(i)}|\Theta^{(i)})$ minimizes the cross entropy $H(\Theta|\Theta^{(i)})$. Hence $\nabla H(\Theta^{(i)}|\Theta^{(i)}) = \mathbf{0}$. This gives the master

equation:

$$\nabla \log p(\mathbf{y}|\mathbf{x}, \Theta^{(i)}) = \nabla Q\left(\Theta^{(i)}|\Theta^{(i)}\right) . \tag{9}$$

So the BP and GEM gradients are identical at each iteration $i$.

We remark that the master gradient equation (9) holds at each hidden layer of the neural network. This allows NEM-noise injection in hidden layers as well as in output layers (Kosko et al., 2019). We focus on noise injection in output layers and just summarize how the layer-level equation applies. Suppose the network has $k$ hidden layers $\mathbf{h}_k, \dots, \mathbf{h}_1$. The numbering starts in the forward direction with the first hidden layer $\mathbf{h}_1$ after the input (identity) layer $\mathbf{x}$. Then the total network likelihood is the probability density $p(\mathbf{y}, \mathbf{h}_k, \dots, \mathbf{h}_1, |\mathbf{x}, \Theta^{(i)})$. The "chain rule" or multiplication theorem of probability factors this likelihood into a product of layer likelihoods:

$$p(\mathbf{y}, \mathbf{h}_k, \dots, \mathbf{h}_1, |\mathbf{x}, \Theta^{(i)}) = p(\mathbf{y}|\mathbf{h}_k, \dots, \mathbf{h}_1, \mathbf{x}, \Theta^{(i)})\times$$
$$p(\mathbf{h}_k|\mathbf{h}_{k-1}, \dots, \mathbf{h}_1, \mathbf{x}, \Theta^{(i)}) \cdots p(\mathbf{h}_2|\mathbf{h}_1, \mathbf{x}, \Theta^{(i)})p(\mathbf{h}_1|\mathbf{x}, \Theta^{(i)}) \tag{10}$$

where we assume the input-data prior $p(\mathbf{x}) = 1$ for simplicity. Taking logarithms at iteration $i$ gives the total network log-likelihood $L(\mathbf{x})$ as the sum of the layer log-likelihoods:

$$L(\mathbf{x}) = L(\mathbf{y}|\mathbf{x}) + L(\mathbf{h}_k|\mathbf{x}) + \cdots + L(\mathbf{h}_1|\mathbf{x}) \tag{11}$$

where $L(\mathbf{h}_k|\mathbf{x}) = \log p(\mathbf{h}_k|\mathbf{h}_{k-1}, \dots, \mathbf{h}_1, \mathbf{x}, \Theta^{(i)})$. Then the above BP-as-GEM argument (5)–(9) applies at layer $k$ if BP invariance holds at that layer.

We next present the likelihood structure for three different neural networks. The networks differ in the structure of their terminal-layer neurons and in the function that these networks perform. We consider three different likelihood structures based on the output activation of the networks. The two main neural models are classifiers and regressors. The third type of network is the logistic network.

A regression network maps the input vector space $\mathbb{R}^I$ to the output space $\mathbb{R}^K$. $I$ is the number of input neurons. $K$ is the number of output neurons. A regression network uses identity activation functions at the output layer. The target vector $\mathbf{t}$ is a Gaussian random vector. It has a mean vector $\mathbf{a}^t$ and has an identity or white covariance matrix $\mathbf{I}$ (which can generalize to a nonwhite covariance matrix):

$$\mathbf{t} \sim \mathcal{N}(\mathbf{t}|\mathbf{a}^t, \mathbf{I}). \tag{12}$$

The likelihood $p_{reg}(\mathbf{t}|\mathbf{x}, \Theta)$ of a regression network is

$$p_{reg}(\mathbf{t}|\mathbf{x}, \Theta) = \frac{1}{(2\pi)^{K/2}} \exp\left\{-\frac{\|\mathbf{t} - \mathbf{a}^t\|^2}{2}\right\} \tag{13}$$

where $\|\cdot\|$ is the Euclidean norm. Then the log-likelihood $L_{reg}$ of the regression network is a constant plus the squared error:

$$L_{reg} = \log p_{reg}(\mathbf{t}|\mathbf{x}, \Theta) \tag{14}$$
$$= \log(2\pi)^{-\frac{K}{2}} - \frac{\|\mathbf{t} - \mathbf{a}^t\|^2}{2} \tag{15}$$
$$= \log(2\pi)^{-\frac{K}{2}} - \frac{1}{2}\sum_{k=1}^{K}\left(t_k - a_k^t\right)^2. \tag{16}$$

So maximizing $L_{reg}$ with respect to $\Theta$ minimizes the squared-error $E_{reg}$ of regression:

$$E_{reg} = \frac{1}{2}\sum_{k=1}^{K}\left(t_k - a_k^t\right)^2. \tag{17}$$

A classifier network maps the input space $\mathbb{R}^I$ to a length-$K$ probability vector in $[0, 1]^K$. The output neurons of a classification network use a softmax or Gibbs activation function. So an output neuron has the form of an exponential divided by $K$ exponentials. The target vector $\mathbf{t}$ is a unit bit vector. It has a 1 in the $k$th slot and 0s elsewhere. The probability that the $k$th scalar component $t_k$ equals 1 is

$$p(t_k = 1|\mathbf{x}, \Theta) = a_k^y. \tag{18}$$

The target $\mathbf{t}$ defines a one-shot multinomial or categorical random variable. So the (output-layer) likelihood $p_{class}(\mathbf{t}|\mathbf{x}, \Theta)$ of a classifier network is the multinomial product

$$p_{class}(\mathbf{t}|\mathbf{x}, \Theta) = \prod_{k=1}^{K}\left(a_k^t\right)^{t_k}. \tag{19}$$

The log-likelihood $L_{class}$ is the negative cross-entropy:

$$L_{class} = \log\ p_{class}(\mathbf{t}|\mathbf{x}, \Theta) \tag{20}$$
$$= \log \prod_{k=1}^{K}\left(a_k^t\right)^{t_k} \tag{21}$$
$$= \sum_{k=1}^{K} t_k \log\left(a_k^t\right). \tag{22}$$

Maximizing the log-likelihood $L_{class}$ with respect to $\Theta$ minimizes the output cross-entropy $E_{class}$:

$$E_{class} = -\sum_{k=1}^{K} t_k \log\left(a_k^t\right). \tag{23}$$

A logistic network $N$ maps the input space $\mathbb{R}^I$ to the unit hypercube $[0, 1]^K$ if $K$ is the number of output logistic neurons. Bipolar logistic output neurons map inputs to the bipolar hypercube $[-1, 1]^K$ by shifting and scaling the logistic activation functions. The target vector $\mathbf{t}$ consists of $K$ independent Bernoulli variables. The probability that the $k$th output neuron $t_k$ equals 1 is just the ordinary Bernoulli probability of getting heads after one flip of a coin:

$$p_{log}(t_k = 1|\mathbf{x}, \Theta) = (a_k^t)^{t_k}(1 - a_k^t)^{1-t_k}. \tag{24}$$

The likelihood $p_{log}(\mathbf{t}|\mathbf{x}, \Theta)$ of a logistic network equals the product of the $K$ independent Bernoulli variables. So it equals the probability of flipping $K$ independent coins:

$$p_{log}(\mathbf{t}|\mathbf{x}, \Theta) = \prod_{k=1}^{K}(a_k^t)^{t_k}(1 - a_k^t)^{1-t_k}. \tag{25}$$

Then the log-likelihood $L_{log}$ of a logistic network equals the negative of the *double cross entropy*:

$$L_{log} = \log\ p_{log}(\mathbf{y}|\mathbf{x}, \Theta) \tag{26}$$
$$= \sum_{k=1}^{K} t_k \log\,(a_k^t) + \sum_{k=1}^{K}(1 - t_k)\log\,(1 - a_k^t). \tag{27}$$

So maximizing the log-likelihood $L_{log}$ minimizes the error function $E_{log}$ or double cross entropy:

$$E_{log} = -\sum_{k=1}^{K} t_k \log\,(a_k^t) - \sum_{k=1}^{K}(1 - t_k)\log\,(1 - a_k^t). \tag{28}$$

The BP algorithm uses gradient descent or its variants to update a network's weights and other parameters. The BP learning laws remain invariant for all three networks (regression, classification, and logistic) because taking the gradient of their network likelihood gives the same partial derivatives for updating their weights (Kosko et al., 2019).

Suppose that the neural network has just one hidden layer. All results apply to any finite number of hidden layers. The weight

matrix **W** connects the input layer to the hidden layer. The weight matrix **U** connects the hidden layer to the output layer. The learning laws are the same for $L_{reg}$, $L_{class}$, and $L_{log}$ (Kosko et al., 2019). Then the chain rule expands the partial derivative of the log-likelihood $L$ with respect to the synaptic weight $u_{kj}$ as

$$\frac{\partial L}{\partial u_{kj}} = \frac{\partial L}{\partial o_k^t}\frac{\partial o_k^t}{\partial u_{kj}} \tag{29}$$

$$= \frac{\partial L}{\partial a_k^t}\frac{\partial a_k^t}{\partial o_k^t}\frac{\partial o_k^t}{\partial u_{kj}} \tag{30}$$

$$= (t_k - a_k^t)a_j^h \tag{31}$$

if weight $u_{kj}$ connects the $k$th output neuron to the $j$th hidden neuron. This holds because $t_k = 1$ and $t_j = 0$ for $j \neq k$ in 1-in-K encoding. The term $o_k^t$ is the input or argument of the $k$th output neuron. The term $a_k^t$ is the activation of the $k$th output neuron. The term $a_j^h$ is the activation of the $j$th hidden neuron.

The partial derivative of the log-likelihood $L$ with respect to the synaptic weight $w_{ji}$ expands as

$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial L}{\partial a_j^h}\frac{\partial a_j^h}{\partial o_j^h}\frac{\partial o_j^h}{\partial w_{ji}} \tag{32}$$

$$= \left(\sum_{k=1}^{K}\frac{\partial L}{\partial o_k^t}\frac{\partial o_k^t}{\partial a_j^h}\right)\frac{\partial a_j^h}{\partial o_j^h}\frac{\partial o_j^h}{\partial w_{ji}} \tag{33}$$

$$= \left(\sum_{k=1}^{K}(t_k - a_k^t)u_{kj}\right)a_j^h(1 - a_j^h)x_i \tag{34}$$

if weight $w_{ji}$ connects the $j$th hidden neuron to the $i$th input neuron. The term $o_j^h$ is the input of the $j$th hidden neuron. The term $a_j^h$ is the logistic activation of the $j$th hidden neuron. The activation $x_i$ is the (identity) activation of the $i$th input neuron.

### 2.2. Noisy expectation–maximization

The Noisy Expectation–Maximization (NEM) theorem states the general sufficient condition for a noise benefit in the EM algorithm. The theorem shows that noise injection can only shorten the EM algorithm's walk up the nearest hill of log-likelihood on average if the noise satisfies the NEM positivity condition. It holds for additive or multiplicative or any other type of measurable noise injection (Osoba & Kosko, 2016). The noise is just the noise that makes the current signal more probable.

The basic NEM theorem for additive noise states that a noise benefit holds on average at each iteration $i$ if the following positivity condition holds (Osoba, Mitaim, & Kosko, 2013):

$$\mathbb{E}_{\mathbf{x},\mathbf{h},\mathbf{n}|\Theta^*}\left[\log\left(\frac{p(\mathbf{x}+\mathbf{n},\mathbf{h}|\Theta^{(i)})}{p(\mathbf{x},\mathbf{h}|\Theta^{(i)})}\right)\right] \geqslant 0\ . \tag{35}$$

Then the EM noise benefit

$$Q(\Theta^{(i)}|\Theta^*) \leqslant Q_N(\Theta^{(i)}|\Theta^*) \tag{36}$$

holds on average at iteration $i$:

$$\mathbb{E}_{\mathbf{x},\mathbf{N}|\Theta^{(i)}}\left[Q(\Theta^*|\Theta^*) - Q_N(\Theta^{(i)}|\Theta^*)\right]$$
$$\leqslant \mathbb{E}_{\mathbf{x}|\Theta^n}\left[Q(\Theta^*|\Theta^*) - Q(\Theta^{(i)}|\Theta^*)\right] \tag{37}$$

where $\Theta^*$ denotes the maximum-likelihood vector of parameters, $Q_N(\Theta^{(i)}|\Theta^*) = \mathbb{E}_{\mathbf{h}|\mathbf{y},\mathbf{n},\Theta^*}[\log p(\mathbf{x}+\mathbf{n},\mathbf{h}|\Theta^{(i)})]$, and $Q(\Theta^{(i)}|\Theta^*) = \mathbb{E}_{\mathbf{h}|\mathbf{y},\Theta^*}[\log p(\mathbf{x},\mathbf{h}|\Theta^{(i)})]$.

The intuition behind the NEM sufficient condition is that NEM noise is just that added noise **n** that makes the current signal **x** more probable on average: $p(\mathbf{x}+\mathbf{n}|\Theta) \geqslant p(\mathbf{x}|\Theta)$. Rearranging and taking logarithms and expectations gives the NEM sufficient condition (35). The noise-boosted likelihood is closer on average at each iteration to the maximum-likelihood outcome than is the noiseless likelihood (Adigun & Kosko, 2018; Osoba & Kosko, 2016). Kullback–Leibler divergence measures the closeness.

### 2.3. NEM noise benefits in backpropagation

We now present the basic NEM theorems for backpropagation. These results inject noise only into the output layer of a neural network. They extend to noise injection in the hidden layers as well (Kosko et al., 2019). The first three noise-boost theorems appear in Kosko et al. (2019). A version of the classification noise-boost theorem also appeared in Audhkhasi et al. (2016). We restate and briefly reprove these basic noise results. We then extend these noise-boost results to the new and more general case of bidirectional backpropagation. We then further extend these results to the main types of adversarial networks.

The additive NEM-noise sufficient condition for BP training is the average positive inequality

$$\mathbb{E}_{\mathbf{t},\mathbf{h},\mathbf{n}|\mathbf{x},\Theta^*}\left[\log\frac{p(\mathbf{t}+\mathbf{n},\mathbf{h}|\mathbf{x},\Theta^{(i)})}{p(\mathbf{t},\mathbf{h}|\mathbf{x},\Theta^{(i)})}\right] \geqslant 0 \tag{38}$$

at each training iteration $i$ where **t** is the output target, **x** is the input vector, and **n** is the noise.

The next results instantiate the expectation in (38) based on the network function and the demands of BP invariance. Then we extend these unidirectional results to the bidirectional case to obtain the corresponding NEM noise boosts for B-BP.

We start with the BP NEM-noise boost in a regression network. BP invariance requires that the output neurons have identity activations and that the network's output performance measure is squared error. The squared-error constraint corresponds to an output likelihood structure that is a multidimensional normal probability density.

**Theorem 1** (*NEM Noise Benefit for a Regression Network Kosko et al., 2019*)**.** *A backpropagation NEM noise benefit holds for a regression network at iteration $i$ with Gaussian target vector $\mathbf{t} \sim \mathcal{N}(\mathbf{a}^t, \mathbf{I})$ if the injected noise **n** satisfies the inequality*

$$\mathbb{E}_{\mathbf{t},\mathbf{h},\mathbf{n}|\mathbf{x},\Theta^*}\left[\mathbf{n}^T\left(2\mathbf{t} - 2\mathbf{a}^t + \mathbf{n}\right)\right] \leqslant 0 \tag{39}$$

*where $\mathbf{a}^t$ is the output activation vector.*

**Proof.** A NEM noise benefit holds on average if the following positivity condition from (38) holds at training iteration $i$:

$$\mathbb{E}_{\mathbf{t},\mathbf{h},\mathbf{n}|\mathbf{x},\Theta^*}\left[\log\frac{p(\mathbf{t}+\mathbf{n},\mathbf{h}|\mathbf{x},\Theta^{(i)})}{p(\mathbf{t},\mathbf{h}|\mathbf{x},\Theta^{(i)})}\right] \geqslant 0.$$

Rewrite the ratio of probability density functions as

$$\frac{p(\mathbf{t}+\mathbf{n},\mathbf{h}|\mathbf{x},\Theta^{(i)})}{p(\mathbf{t},\mathbf{h}|\mathbf{x},\Theta^{(i)})} = \frac{p(\mathbf{t}+\mathbf{n}|\mathbf{h},\mathbf{x},\Theta^{(i)})p(\mathbf{h}|\mathbf{x},\Theta^{(i)})}{p(\mathbf{t}|\mathbf{h},\mathbf{x},\Theta^{(i)})p(\mathbf{h}|\mathbf{x},\Theta^{(i)})} \tag{40}$$

$$= \frac{p(\mathbf{t}+\mathbf{n}|\mathbf{h},\mathbf{x},\Theta^{(i)})}{p(\mathbf{t}|\mathbf{h},\mathbf{x},\Theta^{(i)})}. \tag{41}$$

Taking logarithms gives

$$\log\frac{p(\mathbf{t}+\mathbf{n},\mathbf{h}|\mathbf{x},\Theta^{(i)})}{p(\mathbf{t},\mathbf{h}|\mathbf{x},\Theta^{(i)})} = \log\frac{p(\mathbf{t}+\mathbf{n}|\mathbf{h},\mathbf{x},\Theta^{(i)})}{p(\mathbf{t}|\mathbf{h},\mathbf{x},\Theta^{(i)})} \tag{42}$$

$$= \log p(\mathbf{t}+\mathbf{n}|\mathbf{h},\mathbf{x},\Theta^{(i)}) - \log p(\mathbf{t}|\mathbf{h},\mathbf{x},\Theta^{(i)}) \tag{43}$$

$$= \log\left[2\pi^{-\frac{K}{2}}\exp\left\{-\frac{\|\mathbf{t}+\mathbf{n}-\mathbf{a}^t\|^2}{2}\right\}\right]$$

$$- \log\left[2\pi^{-\frac{K}{2}} \exp\left\{-\frac{\|\mathbf{t}-\mathbf{a}^t\|^2}{2}\right\}\right] \tag{44}$$

$$= \frac{\|\mathbf{t}-\mathbf{a}^t\|^2}{2} - \frac{\|\mathbf{t}+\mathbf{n}-\mathbf{a}^t\|^2}{2} \tag{45}$$

$$= \frac{1}{2}\Big( \|\mathbf{t}\|^2 + \|\mathbf{a}^t\|^2 - 2\mathbf{t}^T\mathbf{a}^t - \|\mathbf{t}\|^2 - \|\mathbf{a}^t\|^2 - \|\mathbf{n}\|^2 - 2\mathbf{n}^T\mathbf{t} + 2\mathbf{t}^T\mathbf{a}^t + 2\mathbf{n}^T\mathbf{a}^t \Big) \tag{46}$$

$$= \frac{\mathbf{n}^T}{2}\left(2\mathbf{a}^t - 2\mathbf{t} - \mathbf{n}\right). \tag{47}$$

So the NEM sufficient condition becomes

$$\mathbb{E}_{\mathbf{t},\mathbf{h},\mathbf{n}|\mathbf{x},\Theta^*}\left[\frac{\mathbf{n}^T}{2}\left(2\mathbf{a}^t - 2\mathbf{t} - \mathbf{n}\right)\right] \geqslant 0. \tag{48}$$

The inequality has the equivalent form

$$\mathbb{E}_{\mathbf{t},\mathbf{h},\mathbf{n}|\mathbf{x},\Theta^*}\left[\mathbf{n}^T\left(2\mathbf{t} - 2\mathbf{a}^t + \mathbf{n}\right)\right] \leqslant 0. \quad \blacksquare \tag{49}$$

The next BP NEM-noise boost applies to classifier networks with softmax output neurons. BP invariance requires that the output performance measure is the network cross entropy with implied 1-in-$K$ encoding of labeled patterns as unit bit vectors. The cross-entropy constraint corresponds to an output likelihood structure of a categorical or one-shot multinomial probability density.

**Theorem 2** (*NEM Noise Benefit for a Classification Network (Audhkhasi et al., 2016)*)**.** *A backpropagation NEM noise benefit with maximum-likelihood training (BP) holds for a softmax classification network for a multinomial target vector* $\mathbf{t} \sim \text{Multinoulli}(p_k = a_k^t)$ *for* $k \in \{1, 2, \ldots, K\}$ *at iteration* $i$ *if a hyperplane inequality holds on average:*

$$\mathbb{E}_{\mathbf{t},\mathbf{h},\mathbf{n}|\mathbf{x},\Theta^*}\left[\mathbf{n}^T \log\ \mathbf{a}^t\right] \geqslant 0 \tag{50}$$

*where* $\mathbf{a}^t$ *is the output activation vector and where* $p_k$ *is the probability that a given pattern belongs to the kth class.*

**Proof.** A NEM noise benefit holds on average at training iteration $i$ if

$$\mathbb{E}_{\mathbf{t},\mathbf{h},\mathbf{n}|\mathbf{x},\Theta^*}\left[\log\frac{p\left(\mathbf{t}+\mathbf{n},\mathbf{h}|\mathbf{x},\Theta^{(i)}\right)}{p\left(\mathbf{t},\mathbf{h}|\mathbf{x},\Theta^{(i)}\right)}\right] \geqslant 0.$$

Rewrite the ratio of the probability density functions as in (40)–(43). Then the softmax/multinomial structure gives the NEM-noise hyperplane condition:

$$\log\frac{p\left(\mathbf{t}+\mathbf{n},\mathbf{h}|\mathbf{x},\Theta^{(i)}\right)}{p\left(\mathbf{t},\mathbf{h}|\mathbf{x},\Theta^{(i)}\right)} = \log\frac{p\left(\mathbf{t}+\mathbf{n}|\mathbf{h},\mathbf{x},\Theta^{(i)}\right)}{p\left(\mathbf{t}|\mathbf{h},\mathbf{x},\Theta^{(i)}\right)} \tag{51}$$

$$= \log p\left(\mathbf{t}+\mathbf{n}|\mathbf{h},\mathbf{x},\Theta^{(i)}\right) - \log p\left(\mathbf{t}|\mathbf{h},\mathbf{x},\Theta^{(i)}\right) \tag{52}$$

$$= \log\prod_{k=1}^{K}\left(a_k^t\right)^{t_k+n_k} - \log\prod_{k=1}^{K}\left(a_k^t\right)^{t_k} \tag{53}$$

$$= \sum_{k=1}^{K}\left(t_k+n_k\right)\log\left(a_k^t\right) - \sum_{k=1}^{K} t_k\log\left(a_k^t\right) \tag{54}$$

$$= \sum_{k=1}^{K} n_k\log\left(a_k^t\right) \tag{55}$$

$$= \mathbf{n}^T\log\ \mathbf{a}^t. \tag{56}$$

Taking a NEM-based expectation gives the final average form of the hyperplane inequality:

$$\mathbb{E}_{\mathbf{t},\mathbf{h},\mathbf{n}|\mathbf{x},\Theta^*}\left[\mathbf{n}^T\log\ \mathbf{a}^t\right] \geqslant 0. \quad \blacksquare \tag{57}$$

The next BP NEM-noise boost applies to logistic networks. These networks are important for bidirectional processing as in the threshold networks of Figs. 1(a) and (b). BP invariance requires that a layer of logistic neurons have a performance measure that we call double cross entropy (28). It corresponds to a likelihood structure that is a product of Bernoulli probabilities. The output $N(\mathbf{x})$ of a logistic network is not a discrete probability density as in the case of the softmax classifier. The output $N(\mathbf{x})$ is instead a discrete fuzzy set (Kosko, 2018).

**Theorem 3** (*NEM Noise Benefit for a Logistic Network (Kosko et al., 2019)*)**.** *A backpropagation NEM noise benefit holds for a logistic network with* $K$ *independent Bernoulli target neurons* $\mathbf{t}_k \sim \text{Bernoulli}(p_k = a_k^t)$ *at iteration* $i$ *if the following inequality holds:*

$$\mathbb{E}_{\mathbf{t},\mathbf{h},\mathbf{n}|\mathbf{x},\Theta^*}\left[\mathbf{n}^T\left(\log\left(\mathbf{a}^t\right) - \log\left(1-\mathbf{a}^t\right)\right)\right] \geqslant 0 \tag{58}$$

*where* $\mathbf{a}^t$ *is the output activation and where* $p_k$ *is the success probability that the kth output neuron equals 1.*

**Proof.** A NEM noise benefit holds on average if the following positivity condition holds at each training iteration $i$:

$$\mathbb{E}_{\mathbf{t},\mathbf{h},\mathbf{n}|\mathbf{x},\Theta^*}\left[\log\frac{p\left(\mathbf{t}+\mathbf{n},\mathbf{h}|\mathbf{x},\Theta^{(i)}\right)}{p\left(\mathbf{t},\mathbf{h}|\mathbf{x},\Theta^{(i)}\right)}\right] \geqslant 0\ .$$

Then rewriting the ratio of the probability density functions as in (40)–(43) and using the product-Bernoulli (double-cross-entropy) structure gives a noise hyperplane condition:

$$\log\frac{p\left(\mathbf{t}+\mathbf{n},\mathbf{h}|\mathbf{x},\Theta^{(i)}\right)}{p\left(\mathbf{t},\mathbf{h}|\mathbf{x},\Theta^{(i)}\right)} = \log\frac{p\left(\mathbf{t}+\mathbf{n}|\mathbf{h},\mathbf{x},\Theta^{(i)}\right)}{p\left(\mathbf{t}|\mathbf{h},\mathbf{x},\Theta^{(i)}\right)} \tag{59}$$

$$= \log p\left(\mathbf{t}+\mathbf{n}|\mathbf{h},\mathbf{x},\Theta^{(i)}\right) - \log p\left(\mathbf{t}|\mathbf{h},\mathbf{x},\Theta^{(i)}\right) \tag{60}$$

$$= \log\prod_{k=1}^{K}\left(a_k^t\right)^{t_k+n_k}\left(1-a_k^t\right)^{1-t_k-n_k} - \log\prod_{k=1}^{K}\left(a_k^t\right)^{t_k}\left(1-a_k^t\right)^{1-t_k} \tag{61}$$

$$= \sum_{k=1}^{K}\left(t_k+n_k\right)\log\left(a_k^t\right) + \left(1-t_k-n_k\right)\log\left(1-a_k^t\right) - \sum_{k=1}^{K} t_k\log\left(a_k^t\right) + \left(1-t_k\right)\log\left(1-a^{t_k}\right) \tag{62}$$

$$= \sum_{k=1}^{K} n_k\log\left(a_k^t\right) - n_k\log\left(1-a_k^t\right) \tag{63}$$

$$= \mathbf{n}^T\left(\log\ \mathbf{a}^t - \log\left(1-\mathbf{a}^t\right)\right). \tag{64}$$

Taking NEM expectations gives the final form as

$$\mathbb{E}_{\mathbf{t},\mathbf{n}|\mathbf{x},\Theta^*}\left[\mathbf{n}^T\left(\log\ \mathbf{a}^t - \log\left(1-\mathbf{a}^t\right)\right)\right] \geqslant 0. \quad \blacksquare$$

Similar *unidirectional* NEM-noise benefits hold for recurrent backpropagation when training recurrent neural networks with time-varying patterns (Adigun & Kosko, 2017).

The next section extends the above unidirectional-BP results to the more general case of bidirectionality. This culminates in the general Theorem 4 that shows how we can derive families of bidirectional NEM-noise benefits from network architectures that obey BP invariance. These noise benefits include B-BP versions of the unidirectional Theorems 1–3 among many others. We omit their bidirectional proofs for simplicity because they directly apply Theorem 4 to the proofs above.

## 3. Bidirectional backpropagation (B-BP)

The B-BP algorithm trains a multilayered neural network with backpropagation in both directions over the same web of synaptic connections (Adigun & Kosko, 2016, 2019). Algorithm 1 shows the steps in the B-BP algorithm in the more general case if we inject NEM noise into the input and output layers of a classifier network. Similar steps can inject noise in hidden layers and in all the layers of bidirectional regression networks.

B-BP is also a form of maximum likelihood estimation. The forward pass maps the input $\mathbf{x}$ to the output $N(\mathbf{x})$. The backward pass maps the output $\mathbf{y}$ back through the network to $N^T(\mathbf{y})$ given the current network parameters $\Theta$. The algorithm jointly maximizes the forward likelihood $p_f(\mathbf{y}|\mathbf{x}, \Theta)$ *and* the backward likelihood $p_b(\mathbf{x}|\mathbf{y}, \Theta)$. So B-BP finds the weight or parameter vector $\Theta^*$ that maximizes the joint network likelihood:

$$\Theta^* = \arg\max_{\Theta} p_f(\mathbf{y}|\mathbf{x}, \Theta) p_b(\mathbf{x}|\mathbf{y}, \Theta). \tag{65}$$

This likelihood structure holds more generally in terms of the forward and backward *layer* likelihoods per (10).

This joint optimization is the same as the joint optimization of the directional (and layer) log-likelihoods:

$$\Theta^* = \arg\max_{\Theta} \log p_f(\mathbf{y}|\mathbf{x}, \Theta) + \log p_b(\mathbf{x}|\mathbf{y}, \Theta) \tag{66}$$

because the logarithm is monotone increasing. The B-BP algorithm also uses gradient descent or any of its variants to iteratively update the parameters.

### 3.1. B-BP likelihood functions

We next present four different B-BP structures based on the corresponding likelihood structure of the network. The four structures are double regression, double classification, double logistic, and the mixed case of classification and regression. Their function and network structure dictates their log-likelihoods in accord with BP invariance. BP invariance ensures that all the different network structures still use the same BP updates on a given directional training pass.

#### 3.1.1. B-BP double regression

A bidirectional network is a double regression network if it performs regression in both directions. So the input and output layers both use identity activations. The output $\mathbf{y}$ is a Gaussian random vector with mean $\mathbf{a}^y$ and with identity or white covariance matrix $\mathbf{I}$. The input $\mathbf{x}$ is a also Gaussian random vector with mean $\mathbf{a}^x$ and identity or white covariance matrix $\mathbf{I}$:

$$p_f(\mathbf{y}|\mathbf{x}, \Theta) = \frac{1}{(2\pi)^{K/2}} \exp\left\{-\frac{\|\mathbf{y} - \mathbf{a}^y\|^2}{2}\right\} \tag{67}$$

and

$$p_b(\mathbf{x}|\mathbf{y}, \Theta) = \frac{1}{(2\pi)^{I/2}} \exp\left\{-\frac{\|\mathbf{x} - \mathbf{a}^x\|^2}{2}\right\} \tag{68}$$

where $I$ is the dimension of the input layer and $K$ is the dimension of the output layer. The forward log-likelihood $L_f(\Theta)$ and backward log-likelihood $L_b(\Theta)$ are

$$L_f(\Theta) = \log p(\mathbf{y}|\mathbf{x}, \Theta) = \log(2\pi)^{-\frac{K}{2}} - \frac{\|\mathbf{y} - \mathbf{a}^y\|^2}{2} \tag{69}$$

and

$$L_b(\Theta) = \log p(\mathbf{x}|\mathbf{y}, \Theta) = \log(2\pi)^{-\frac{I}{2}} - \frac{\|\mathbf{x} - \mathbf{a}^x\|^2}{2}. \tag{70}$$

The error functions are the squared-error $E_f(\Theta)$ for the forward pass and squared error $E_b(\Theta)$ for the backward pass. B-BP minimizes the joint error $E(\Theta)$:

$$E(\Theta) = E_f(\Theta) + E_b(\Theta) \tag{71}$$

$$= \frac{1}{2}\sum_{k=1}^{K}\left(y_k - a_k^y\right)^2 + \frac{1}{2}\sum_{i=1}^{I}\left(x_i - a_i^x\right)^2. \tag{72}$$

Then Adigun and Kosko (2019) give the detailed B-BP updates rules for double regression.

#### 3.1.2. B-BP double classification

A double classifier is a neural network that acts as a classifier in both the forward and backward directions. So both the input and output layers use softmax or Gibbs activations.

The output $\mathbf{y}$ with activation $a_k^y$ defines the multinomial probability $p_f(y_k = 1|\mathbf{x}, \Theta)$. The input $\mathbf{x}$ with $a_i^x$ defines the dual multinomial probability $p_b(x_i = 1|\mathbf{y}, \Theta)$:

$$p_f(\mathbf{y}|\mathbf{x}, \Theta) = \prod_{k=1}^{K}\left(a_k^y\right)^{y_k} \tag{73}$$

and

$$p_b(\mathbf{x}|\mathbf{y}, \Theta) = \prod_{i=1}^{I}\left(a_i^x\right)^{x_i}. \tag{74}$$

Then the log-likelihoods $L_f$ and $L_b$ are negative cross entropies:

$$L_f(\Theta) = \log p_f(\mathbf{y}|\mathbf{x}, \Theta) = \sum_{k=1}^{K} y_k \log\left(a_k^y\right) \tag{75}$$

and

$$L_b(\Theta) = \log p_f(\mathbf{x}|\mathbf{y}, \Theta) = \sum_{i=1}^{I} x_i \log\left(a_i^x\right). \tag{76}$$

The forward error $E_f(\Theta)$ is the cross-entropy $E_f(\Theta)$. The backward error $E_b(\Theta)$ is likewise the cross-entropy $E_b(\Theta)$. B-BP for double classification minimizes this joint error $E(\Theta)$:

$$E(\Theta) = E_f(\Theta) + E_b(\Theta) \tag{77}$$

$$= -\sum_{k=1}^{K} y_k \log\left(a_k^y\right) - \sum_{i=1}^{I} x_i \log\left(a_i^x\right). \tag{78}$$

Adigun and Kosko (2019) give the update rules for B-BP with double classification. Training in each direction applies ordinary BP updates because of BP invariance. Training occurs in one direction at a time.

#### 3.1.3. B-BP double logistic networks

A double logistic network has logistic neurons at both the input and output layers. Figs. 1(a) and (b) show such double logistic networks where the logistic sigmoids are so steep as to give threshold neurons.

The probability of the logistic-vector output $\mathbf{y}$ is a product of $K$ independent Bernoulli probabilities. The $k$th output activation $a_k^y$ is the probability $p_f(y_k = 1|\mathbf{x}, \Theta)$. The probability of the logistic-vector input $\mathbf{x}$ is a product of $I$ independent Bernoulli probabilities. So the $i$th input activation $a_i^x$ is the probability $p_b(x_i = 1|\mathbf{y}, \Theta)$. Logistic neurons can also map to the bipolar interval $[-1, 1]$ by simple scaling and translation of the usual binary logistic function. We trained the bidirectional threshold networks in Figs. 1(a) and (b) as double-logistic networks with steep logistic sigmoid activations. Then we replaced the steep logistics after B-BP training with threshold functions.

The forward likelihood $p_f(\mathbf{y}|\mathbf{x}, \Theta)$ and the backward likelihood $p_b(\mathbf{x}|\mathbf{y}, \Theta)$ have the product-Bernoulli form

$$p_f(\mathbf{y}|\mathbf{x}, \Theta) = \prod_{k=1}^{K} \left(a_k^y\right)^{y_k} \left(1 - a_k^y\right)^{1-y_k} \tag{79}$$

and

$$p_b(\mathbf{x}|\mathbf{y}, \Theta) = \prod_{i=1}^{I} \left(a_i^x\right)^{x_i} \left(1 - a_i^x\right)^{1-x_i}. \tag{80}$$

The log-likelihoods $L_f(\Theta)$ and $L_b(\Theta)$ define *double* cross entropies:

$$L_f(\Theta) = \log p_f(\mathbf{y}|\mathbf{x}, \Theta) \tag{81}$$

$$= \sum_{k=1}^{K} y_k \log\left(a_k^y\right) + (1 - y_k) \log\left(1 - a_k^y\right) \tag{82}$$

and

$$L_b(\Theta) = \log p_f(\mathbf{x}|\mathbf{y}, \Theta) \tag{83}$$

$$= \sum_{i=1}^{I} x_i \log\left(a_i^x\right) + (1 - x_i) \log\left(1 - a_i^x\right). \tag{84}$$

Then the joint error function $E(\Theta)$ for a double logistic network is the sum

$$E(\Theta) = E_f(\Theta) + E_b(\Theta) \tag{85}$$

$$= -\sum_{k=1}^{K} y_k \log\left(a_k^y\right) + (1 - y_k) \log\left(1 - a_k^y\right) - \sum_{i=1}^{I} x_i \log\left(a_i^x\right) + (1 - x_i) \log\left(1 - a_i^x\right). \tag{86}$$

The BP update rules for double-logistic training are the same as for double classification because of BP invariance.

#### 3.1.4. Mixed case: B-BP classification and regression

The mixed bidirectional case occurs when the forward pass is a classifier and the backward pass is a regressor. This is the implied structure of the modern neural classifier used only in the forward direction because it maps identity neurons to output softmax neurons. The forward pass through the network $N$ estimates the class membership of a given input pattern $\mathbf{x}$. So the $K$ output neurons have softmax or Gibbs activations. The backward pass through $N^T$ estimates the class centroid (in the common least-squares case) given an output class-label unit bit vector or other length-$K$ probability vector $\mathbf{y}$. So the input neurons use identity or linear activations.

The forward likelihood is a multinomial random variable as in the case of double classification. The backward likelihood is a Gaussian random vector as in the case of double regression. BP invariance holds for these likelihoods given the corresponding softmax output neurons and identity input neurons.

The total mixed error $E(\Theta)$ sums the forward cross-entropy $E_f(\Theta)$ and the backward squared-error $E_b(\Theta)$:

$$E(\Theta) = E_f(\Theta) + E_b(\Theta) \tag{87}$$

$$= -\sum_{k=1}^{K} y_k \log\left(a_k^y\right) + \frac{1}{2} \sum_{i=1}^{I} \left(x_i - a_i^x\right)^2. \tag{88}$$

The original B-BP sources (Adigun & Kosko, 2016, 2019) give the gradient update rules for B-BP in the mixed case as well for double regression and double classification. BP invariance greatly simplifies the forward and backward updates because then all network types use the same BP updates.

### 3.2. B-BP algorithm for classifier networks

We first extend the B-BP algorithm to classifier networks for bidirectional representation. The mapping from $Y$ to $X$ or vice versa is not a well-defined point function in most cases. So the $k$th unit basis vector $\mathbf{y}$ need not map back to a unique value. It maps instead to a pre-image set $f^{-1}(\mathbf{y})$ as discussed above. The reverse pass through $N^T$ gives a point $\mathbf{x}'$ in the input pattern space $R^n$. Then Adigun and Kosko (2019) shows that the regression backward pass over the network $N$ tends to map to the centroid of the pre-image in a classification-regression bidirectional network.

The backward pass of a classifier network maps the class labels or output $K$-length probability vectors back to the input pattern space. The output space $Y$ is the $K$-1 simplex $S^{K-1}$ of $K$-length probability vectors for a classifier network. This holds because the network maps vector pattern inputs to $K$ softmax neurons vectors and uses 1-in-$K$ encoding.

So the backward pass for bidirectional representation maps any given $\mathbf{y}$ to a centroidal measure of its class. Let the $m$th sample $(\mathbf{x}^{(m)}, \mathbf{t}^{(m)})$ be a pair of sample input and target vectors that belongs to the $k$th class $C_k$. B-BP trains the network along the forward pass $N(\mathbf{x}^{(m)})$ with $\mathbf{t}^{(m)}$ as the target. The forward error $E_f$ is the cross entropy between $N(\mathbf{x}^{(m)})$ and $\mathbf{t}^{(m)}$.

Backward-pass training can in principle use any $\mathbf{x}'$. The simplest case uses the backward-emitted vector $\mathbf{x}' = N^T(\mathbf{y})$ after the input $\mathbf{x}$ has emitted the output vector $\mathbf{y}$ in the forward direction: $\mathbf{y} = N(\mathbf{x})$. Simulations show that somewhat better performance tends to result if the backward pass replaces $\mathbf{y}$ with the supervised class label $k$ or the unit vector $\mathbf{e}_k$ if the stimulating input vector $\mathbf{x}$ belongs to the $k$th input class.

The backward pass can also use the $k$th class population centroid $\mathbf{c}_k$ of input pattern class $C_k$ or its sample-mean estimate $\hat{\mathbf{c}}_k$:

$$\hat{\mathbf{c}}_k = \frac{1}{N_k} \sum_{\mathbf{x}^{(m)} \in C_k} \mathbf{x}^{(m)} \tag{89}$$

where $N_k$ is the number of input sample patterns $\mathbf{x}^{(m)}$ from class $C_k$. Then the weak and strong laws of large numbers state that the sample centroid $\hat{\mathbf{c}}_k$ converges to the population or true class centroid $\mathbf{c}_k$ in probability or with probability one if the sample vectors are random samples (independent and identically distributed) with respective finite covariances or finite means (Hogg, Mckean, & Craig, 2012). This *ergodic* result still holds in the mean-squared sense when the sample vectors are correlated if the random vectors are wide-sense stationary and if the finite scalar covariances are asymptotically zero (Gubner, 2006). This correlated result also holds in probability since convergence in mean-square implies convergence in probability.

Using the centroid estimate $\hat{\mathbf{c}}_k$ makes sense in the backward direction because locally the population centroid $\mathbf{c}_k$ minimizes the squared error of regression. It likewise minimizes the total mean-squared error of vector quantization (Kosko, 1991b).

The $K$-means clustering algorithm is the standard way to estimate $K$ centroids from training data (Jain, 2010). This algorithm is a form of competitive learning and sometimes called a self-organizing map (Kohonen, 1990). $K$-means clustering enjoys a NEM noise-boost because the algorithm is itself a special case of the EM algorithm for tuning a Gaussian mixture model if the class membership functions are binary indicator functions (Osoba & Kosko, 2013).

The projection $\tilde{\mathbf{x}}_k$ of the $k$th class centroid $\hat{c}_k$ to the space of training samples in $C_k$ obeys

$$\tilde{\mathbf{x}}_k = \arg\min_{\mathbf{x}^{(m)} \in C_k} \|\mathbf{x}^{(m)} - \hat{c}_k\|^2 \tag{90}$$

where $\tilde{\mathbf{x}}_k \in C_k$. The sample centroid need not lie in the pattern class $C_k$. We can compute the backward vector directly if it does by taking the difference between the backward pass $\mathbf{x}'$ and the sample class centroid. We otherwise take the difference between $\mathbf{x}'$ and that training vector in $C_k$ that lies closest to the sample class centroid. Then the backward error $E_b$ equals the squared error between $\mathbf{x}' = N^T(\mathbf{y}^{(m)})$ and the projection $\tilde{\mathbf{x}}_k$.

## 4. NEM noise-boosted B-BP learning

This section shows how NEM noise boosts the B-BP algorithm. B-BP seeks to jointly maximize the log-likelihood of the network's forward pass and its backward pass. So B-BP is also a form of maximum-likelihood estimation.

The above BP-as-GEM theorem states that the gradient of the network log-likelihood equals the gradient of the EM surrogate likelihood $Q(\Theta|\Theta^n)$: $\nabla \log p = \nabla Q$. This gradient equality gives in the bidirectional case

$$\nabla_\Theta \log p(\mathbf{y}|\mathbf{x}, \Theta) + \nabla_\Theta \log p(\mathbf{x}|\mathbf{y}, \Theta) = \nabla_\Theta Q_f(\Theta|\Theta^{(i)}) + \nabla_\Theta Q_b(\Theta|\Theta^{(i)}) \quad (91)$$

if $Q_f(\Theta|\Theta^{(i)})$ and $Q_b(\Theta|\Theta^{(i)})$ are the surrogate likelihoods for the respective forward pass and backward pass. So B-BP is also a special case of generalized EM. These results also hold at the layer level so long as the layer likelihood obeys BP invariance in each direction.

The above gradient result also holds in general for any number $M$ of directions and for any neural-network structure so long as the structure maintains BP invariance in each direction. We consider next the special case of classification in the forward direction because it is by far the most common neural network in practice.

### 4.1. NEM-boosted B-BP for classifier networks

B-BP is a special case of GEM in each direction. So the injection of noise that satisfies the NEM positivity condition improves the average performance of B-BP. The noise can be additive or multiplicative or can have any other measurable form. The beneficial NEM noise is just that noise $\mathbf{n}_x$ and $\mathbf{n}_y$ that makes the data more probable on average.

The forward likelihood $p(\mathbf{y}|\mathbf{x}, \Theta)$ for a classifier network is the one-shot multinomial probability density. So the classifier uses softmax output neurons and a cross-entropy performance measure. The backward likelihood $p(\mathbf{x}|\mathbf{y}, \Theta)$ is an $I$-dimensional vector Gaussian density. So the implied backward regression network uses identity input neurons (output neurons in the backward direction) and a squared-error performance measure. Then BP invariance holds: both passes use the same BP updates.

The NEM positivity condition holds for B-BP training of a classifier network with noise injection $\mathbf{n}_x$ in the input neurons and with $\mathbf{n}_y$ in the output neurons if the following NEM positivity conditions hold:

$$\mathbb{E}_{\mathbf{y},\mathbf{h},\mathbf{n}|\mathbf{x},\Theta^*}\left[\mathbf{n}_y^T \log \mathbf{a}^y\right] \geqslant 0 \quad (92)$$

and

$$\mathbb{E}_{\mathbf{x},\mathbf{h},\mathbf{N}|\mathbf{y},\Theta^*}\left[\mathbf{n}_x^T\left(2\tilde{\mathbf{x}} - 2\mathbf{a}^x + \mathbf{n}_x\right)\right] \leqslant 0 \quad (93)$$

if $\tilde{\mathbf{x}}$ is the sample class centroid, $\mathbf{a}^y$ is the forward pass $N(\mathbf{x})$, and $\mathbf{a}^x$ is the backward pass $N^T(\mathbf{y})$ through the classifier network. Injecting the noise pair $(\mathbf{n}_x, \mathbf{n}_y)$ that satisfies this condition at the input and output layers produces the NEM noise benefit in the B-BP training of the classifier (classifier-regression) network.

**Data**: $\{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^N$, batch size $M$, learning rate $\alpha$, number of epochs $L$, number of iterations $T$, annealing factor $\eta$, noise variance $\sigma$, and other hyperparameters for optimization.
**Result**: Trained weight $\Theta$.
**Compute**: The class centroids $c_k$ for all the $K$ classes:

$$c_k = \frac{1}{N_k}\sum_{j=1}^{N_k} \mathbf{x}^{k(j)}$$

**Compute**: The projection $\tilde{\mathbf{x}}_k$ of class centroids $c_k$ onto the class sample set:

$$\tilde{\mathbf{x}}_k = \underset{\mathbf{x}^{(i)} \in C_k}{\operatorname{argmin}} ||\mathbf{x}^{(i)} - c_k||^2$$

**while** *epoch* $l : 1 \to L$ **do**
  **while** *iteration* $t : 1 \to T$ **do**
    • Select a batch of $M$ samples $\{\mathbf{x}^{(m)}, \mathbf{y}^{(m)}\}_{m=1}^M$
    • *Forward Pass* : Compute the $K$-dimensional output softmax activation vector $\mathbf{a}^{y(m)}$:

$$\mathbf{a}^{y(m)} = N(\mathbf{x}^{(m)})$$

    • *Backward Pass* : Compute the $I$-dimensional input identity activation vector $\mathbf{a}^{x(t)}$:

$$\mathbf{a}^{x(m)} = N^T(\mathbf{y}^{(m)})$$

    • Compute the projection $\tilde{\mathbf{x}}^{(m)}$ for sample $\mathbf{x}^{(m)}$

$$\tilde{\mathbf{x}}_k^{(m)} = \underset{\tilde{\mathbf{x}}_k}{\operatorname{argmin}} ||\tilde{\mathbf{x}}_k - \mathbf{x}^{(m)}||^2$$

    • Generate noise $\mathbf{n}_{y^{(m)}}$ for the output layer:
    **if** $\mathbf{n}_{y^{(m)}}^T \log \mathbf{a}^{y(m)} \geqslant 0$ **then**

$$\mathbf{y}^{(m)} \leftarrow \mathbf{y}^{(m)} + \mathbf{n}_{y^{(m)}}$$

    **end**
    • Generate noise $\mathbf{n}_{x^{(m)}}$ for the input layer:
    **if** $\mathbf{n}_{x^{(m)}}^T\left(2\tilde{\mathbf{x}}_k^{(m)} - 2\mathbf{a}^{x(m)} + \mathbf{n}_{x^{(m)}}\right) \leqslant 0$ **then**

$$\tilde{\mathbf{x}}_k^{(m)} \leftarrow \tilde{\mathbf{x}}_k^{(m)} + \mathbf{n}_{x^{(m)}}$$

    **end**
    • Compute the cross-entropy $E_f(\Theta)$:

$$E_f(\Theta) = -\frac{1}{M}\sum_{m=1}^{M} \mathbf{y}^{(m)T} \log\ \mathbf{a}^{y(m)}$$

    • Compute the squared-error $E_b(\Theta)$:

$$E_b(\Theta) = \frac{1}{M}\sum_{m=1}^{M} \left|\left|\mathbf{a}^{x(m)} - \tilde{\mathbf{x}}_k^{(m)}\right|\right|^2$$

    • Back-propagate the error functions and update the weights:

$$\Theta^{(t+1)} = \Theta^{(t)} - \alpha\nabla_\Theta E_f(\Theta) - \alpha\nabla_\Theta E_b(\Theta)$$

  **end**
**end**

**Algorithm 1:** B-BP algorithm for NEM noise injection into the output and input layers of a classifier neural network trained with bidirectional backpropagation.

Algorithm 1 gives the update rule for the $I$th epoch of NEM-boosted B-BP training of such a classifier network.

### 4.2. B-BP NEM theorem

We now summarize the above noise-boosting results as the B-BP NEM theorem. A joint NEM noise results if we inject noise that satisfies the average positivity condition on both the forward and backward passes of B-BP. This includes B-BP versions of the unidirectional Theorems 1–3 above as special cases.

We first define the relevant surrogate likelihood functions for generalized EM. The surrogate log-likelihood $Q_f(\Theta|\Theta^*)$ for the forward pass of B-BP without noise injection is

$$Q_f(\Theta|\Theta^*) = \mathbb{E}_{\mathbf{h}|\mathbf{y},\mathbf{x},\Theta^*}\Big[\log\ p(\mathbf{y}, \mathbf{h}|\mathbf{x},\Theta)\Big]. \tag{94}$$

Then the surrogate log-likelihood $Q_f^N(\Theta|\Theta^*)$ for the forward pass of B-BP with noise injection is

$$Q_f^N(\Theta|\Theta^*) = \mathbb{E}_{\mathbf{h}|\mathbf{y},\mathbf{n_y},\mathbf{x},\Theta^*}\Big[\log\ p(\mathbf{y}+\mathbf{n}_y, \mathbf{h}|\mathbf{x}, \Theta)\Big] \tag{95}$$

if $\mathbf{n}_y$ is the injected noise into the output layer. The surrogate log-likelihood $Q_b(\Theta|\Theta^*)$ for the backward pass of B-BP without noise injection is

$$Q_b(\Theta|\Theta^*) = \mathbb{E}_{\mathbf{h}|\mathbf{y},\mathbf{x},\Theta^*}\Big[\log\ p(\mathbf{x}, \mathbf{h}|\mathbf{y},\Theta)\Big]. \tag{96}$$

The surrogate log-likelihood $Q_b^N(\Theta|\Theta^*)$ for the backward pass of B-BP with noise injection is

$$Q_b^N(\Theta|\Theta^*) = \mathbb{E}_{\mathbf{h}|\mathbf{y},\mathbf{n_x},\mathbf{x},\Theta^*}\Big[\log\ p(\mathbf{x}+\mathbf{n}_x, \mathbf{h}|\mathbf{y},\Theta)\Big] \tag{97}$$

if $\mathbf{n}_x$ is the injected noise into the input layer. We can now state and prove the general noise-benefit theorem for B-BP.

**Theorem 4** (*NEM Noise Benefit with B-BP*)**.** *Suppose the forward positivity condition holds on average at iteration i :*

$$\mathbb{E}_{\mathbf{y},\mathbf{h},\mathbf{n}_y|\boldsymbol{\Theta}^*}\left[\log\frac{p_f(\mathbf{y}+\mathbf{n}_y, \mathbf{h}|\mathbf{x},\ \boldsymbol{\Theta})}{p_f(\mathbf{y}, \mathbf{h}|\mathbf{x},\ \boldsymbol{\Theta})}\right] \geqslant 0. \tag{98}$$

*Suppose further that the backward positivity condition also holds on average at iteration i:*

$$\mathbb{E}_{\mathbf{x},\mathbf{h},\mathbf{n}_x|\boldsymbol{\Theta}^*}\left[\log\frac{p_b(\mathbf{x}+\mathbf{n}_x, \mathbf{h}|\mathbf{y},\ \boldsymbol{\Theta})}{p_b(\mathbf{x}, \mathbf{h}|\mathbf{y},\ \boldsymbol{\Theta})}\right] \geqslant 0. \tag{99}$$

*Then the bidirectional EM noise benefits*

$$Q_f\left(\Theta^{(i)}|\Theta^*\right) \leqslant Q_f^N\left(\Theta^{(i)}|\Theta^*\right) \tag{100}$$

*and*

$$Q_b\left(\Theta^{(i)}|\Theta^*\right) \leqslant Q_b^N\left(\Theta^{(i)}|\Theta^*\right) \tag{101}$$

*hold on average at each iteration i:*

$$\mathbb{E}_{\mathbf{y},\mathbf{n_y}|\boldsymbol{\Theta}^{(i)}}\Big[Q_f\left(\Theta^*|\Theta^*\right) - Q_f^N(\Theta^{(i)}|\Theta^*)\Big] \leqslant \mathbb{E}_{\mathbf{y}|\boldsymbol{\Theta}^{(i)}}\Big[Q_f(\Theta^*|\Theta^*) - Q_f(\Theta^{(i)}|\Theta^*)\Big] \tag{102}$$

$$\mathbb{E}_{\mathbf{x},\mathbf{n_x}|\boldsymbol{\Theta}^{(i)}}\Big[Q_b(\Theta^*|\Theta^*) - Q_b^N(\Theta^{(i)}|\Theta^*)\Big] \leqslant \mathbb{E}_{\mathbf{x}|\boldsymbol{\Theta}^{(i)}}\Big[Q_b(\Theta^*|\Theta^*) - Q_b^N(\Theta^{(i)}|\Theta^*)\Big]. \tag{103}$$

*The joint EM noise benefit also holds on average at iteration i:*

$$\mathbb{E}_{\mathbf{y},\mathbf{n_y}|\boldsymbol{\Theta}^{(i)}}\Big[Q_f^N(\Theta^{(i)}|\Theta^*)\Big] + \mathbb{E}_{\mathbf{x},\mathbf{n_x}|\boldsymbol{\Theta}^{(i)}}\Big[Q_b^N(\Theta^{(i)}|\Theta^*)\Big] \geqslant \mathbb{E}_{\mathbf{y}|\boldsymbol{\Theta}^{(i)}}\Big[Q_f(\Theta^{(i)}|\Theta^*)\Big] + \mathbb{E}_{\mathbf{x}|\boldsymbol{\Theta}^{(i)}}\Big[Q_b(\Theta^{(i)}|\Theta^*)\Big]. \tag{104}$$

**Proof.** We show that B-BP training maximizes the sum of the surrogate log-likelihoods $Q_f(\Theta|\Theta^{(n)})$ and $Q_b(\Theta|\Theta^{(i)})$.

The bidirectional version of the master gradient equation (9) shows that the bidirectional log-likelihood equals the sum of the surrogate log-likelihoods and entropies. So the forward log-likelihood equals

$$\log p_f(\mathbf{y}|\mathbf{x},\ \Theta^{(i)}) = Q_f(\Theta^{(i)}|\Theta^*) + H(\Theta^{(i)}|\Theta^*). \tag{105}$$

The backward log-likelihood likewise equals

$$\log p_b(\mathbf{x}|\mathbf{y},\ \Theta^{(i)}) = Q_b(\Theta^{(i)}|\Theta^*) + H(\Theta^{(i)}|\Theta^*). \tag{106}$$

The unidirectional BP-as-GEM theorem (Audhkhasi et al., 2016) states that the gradient of the network log-likelihood equals the gradient of the EM surrogate likelihood $Q(\Theta|\Theta^{(n)})$. Then the gradient of the sum of the directional likelihoods equals

$$\nabla_\Theta \log p_f(\mathbf{y}|\mathbf{x},\ \Theta) + \nabla_\Theta \log p_b(\mathbf{x}|\mathbf{y},\ \Theta) = \nabla_\Theta Q_f(\Theta|\Theta^{(i)}) + \nabla_\Theta Q_b(\Theta|\Theta^{(i)}) \tag{107}$$

where $Q_f(\Theta|\Theta^{(i)})$ and $Q_b(\Theta|\Theta^{(i)})$ are the surrogate log-likelihoods for the respective forward pass and backward pass. This holds because the above entropy inequality implies that both entropy gradients are null. Then the gradient update rule for B-BP at the $i$th iteration

$$\Theta^{(i+1)} = \Theta^{(i)} + \eta\nabla_\Theta\Big(\log\ p_f(\mathbf{y}|\mathbf{x},\ \Theta) + \log p_b(\mathbf{x}|\mathbf{y},\ \Theta)\Big)\Big|_{\Theta=\Theta^{(i)}} \tag{108}$$

equals the GEM update equation

$$\Theta^{(i+1)} = \Theta^{(i)} + \eta\nabla_\Theta\left(Q_f(\Theta|\Theta^{(i)}) + Q_b(\Theta|\Theta^{(i)})\right)\Big|_{\Theta=\Theta^{(i)}}. \tag{109}$$

So B-BP is also a special case of GEM. It then inherits GEM's NEM noise boost in each direction. □

## 5. Generative adversarial networks and bidirectional training

An adversarial network consists of two or more neural networks that try to trick each other. They use feedback among the neural networks and sometimes within neural networks.

The standard generative adversarial network (GAN) consists of two competing neural networks. One network generates patterns to trick or fool the other network that tries to tell whether a generated pattern is real or fake. The generator network $G$ acts as a type of art forger while the discriminator network $D$ acts as a type of art expert or detective.

GAN training helps the discriminator $D$ get better at detecting real data from fake or synthetic data. It trains the generator $G$ so that $D$ finds it harder to distinguish the real data from $G$'s fake or generated data.

Goodfellow et al. (2014) showed that some forms of this adversarial process reduce to a two-person minimax game between $D$ and $G$ in terms of a value function $V(G, D)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x\sim p_r(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{110}$$

if $p_r$ is the probability density of the real data $x$ and if $p_z$ is the probability density of the latent or hidden variable $z$.

GAN training alternates between training $D$ and $G$. It trains $D$ for a constant $G$ and then trains $G$ for a constant $D$. The probability $p_g$ is the probability density of the generated data $G(z)$ that $G$ produces. Optimizing the value function in (110) is the same as minimizing the Jensen–Shannon (JS) divergence between $p_r$ and $p_g$ (Goodfellow et al., 2014). This is the same as maximizing the product of the likelihood functions $p(y = 1|x)$ and $p(y = 0|G(z))$ with respect to $D$ and minimizing the likelihood function $p(y = 0|G(z))$ with respect to $G$. The likelihood functions are Bernoulli probability densities.

Minimizing the *JS* divergence can lead to a vanishing gradient as the discriminator $D$ gets better at distinguishing real data from fake data (Arjovsky & Bottou, 2017). The JS divergence can also correlate poorly with the quality of the generated samples (Arjovsky, Chintala, & Bottou, 2017). It also often leads to *mode collapse*: Training converges to the generation of a single image (Metz, Poole, D. Pfau, & Sohl-Dickstein, 2016). We use the term *vanilla* GAN to refer to a GAN trained to minimize the *JS* distance. We found that bidirectionality greatly reduced the instances and severity of mode collapse in regular vanilla GANs.

Arjovsky et al. (2017) proposed the *Wasserstein* GAN (WGAN) that minimizes the Earth-Mover distance between $p_r$ and $p_g$. The Kantorovich–Rubinstein duality (Villani, 2009) shows that we can define the Earth-Mover distance as

$$W(p_r,\ p_g) = \sup_{\|f\|_{L\leqslant 1}} \mathbb{E}_{x\sim p_r}[f(x)] - \mathbb{E}_{\tilde{x}\sim p_g}[f(\tilde{x})] \tag{111}$$

where the supremum runs over the set of all 1-Lipschitz functions $f : \chi \rightarrow \mathbb{R}$. Then Gulrajani, Ahmed, Arjovsky, V. Dumoulin, and Courville (2017) proposed adding a gradient penalty term that enforces the Lipschitz constraint. This gives the modified value function $L(p_g, p_r)$ as

$$L(p_r,\ p_g) = \mathbb{E}_{x\sim p_r}[D(x)] - \mathbb{E}_{z\sim p_z}[D(G(z))] + \lambda\ \mathbb{E}_{\hat{x}\sim p_{\hat{x}}}[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2] \tag{112}$$

where $\hat{x}$ is a convex sum of $x \sim p_r$ and $G(z)$ with $z \sim p_z$. This approach performed better than using weight clipping (Arjovsky & Bottou, 2017) to enforce the Lipschitz constraint. Salimans, Goodfellow, Zaremba, Cheung, and Radford (2016) proposed heuristics such as feature matching, minibatch discrimination, and batch normalization to improve the performance and convergence of GAN training. Algorithm 3 uses (112) for the B-BP training of a WGAN.

### 5.1. Bidirectional GAN training

Bidirectional GAN training uses a new backward training pass for the discriminator $D$ network. The bidirectional training uses the appropriate joint forward–backward performance measure so that training in one direction does not overwrite training in the reverse direction. Such training does not apply to the usual unidirectional backpropagation training of the generator $G$ as we explain below. The forward pass of the discriminator $D$ tries to distinguish the pattern $x$ from the generated pattern $G(z)$. The backward pass helps the set-inverse map $D^T$ better distinguish the two.

We cast the backward pass as multidimensional *regression*. Then the backward-pass training becomes maximum-likelihood estimation for a conditional Gaussian probability density. This holds because maximizing the likelihood $L_b$ of the backward pass minimizes the backward error $E_b$ when $E_b$ is squared error (Adigun & Kosko, 2019; Audhkhasi et al., 2016; Osoba & Kosko, 2016).

B-BP did not train the generator $G$ because we found that B-BP did not outperform unidirectional BP for this task. We show briefly that this holds because of a constraint inequality: Training the generator $G$ on the backward pass to maximize a likelihood function $L_b^G$ can only harm the GAN's performance on average.

The argument for this negative result is that B-BP constrains the training of $G$ more than unidirectional BP does. GAN training seeks solutions $G^*$ and $D^*$ that optimize the value function $V(D, G)$ that measures the similarity between the real samples and the fake or generated samples:

$$\min_G \max_D V(D, G)\ . \tag{113}$$

Consider the optimal solution $D_G^*$ for a given G:

$$D_G^* = \arg\max_D V(D, G) \tag{114}$$

and define $C(G) = V(D_G^*,\ G)$.

The best generator $G^*$ for the value function obeys

$$G^* = \arg\min_G C(G). \tag{115}$$

The definition of arg min implies that

$$C(G^*) \leqslant C(G) \tag{116}$$

for all $G$. Let $G_b^*$ be that constrained $G$ that further partially or totally maximizes the backward-pass likelihood $L_b^G$. Then

$$C(G^*) \leqslant C(G_b^*) \tag{117}$$

since the additional B-BP constraint only shrinks the search space for the minimum.

### 5.2. Bidirectional training of a vanilla GAN

Each bidirectional training iteration of a vanilla GAN involves three steps. The first step trains the discriminator $D$ to maximize the backward-pass likelihood $L_b^D$. This likelihood measures the mismatch between the real samples and the generated samples on the backward pass. The likelihood was a multivariate Gaussian probability density function.

The backward error $E_b^D$ measures the mismatch between $M$ generated samples $\{G(\mathbf{z}^{(m)})\}_{m=1}^M$ and $M$ real or authentic samples $\{\mathbf{x}^{(m)}\}_{m=1}^M$:

$$E_b^D = \sum_{m=1}^{M}\Big(\big|\big|G(\mathbf{z}^{(m)}) - D^T(\mathbf{a}_r^{(m)})\big|\big|^2 + \big|\big|\mathbf{x}^{(m)} - D^T(\mathbf{a}_g^{(m)})\big|\big|^2\Big) \tag{118}$$

where $\mathbf{a}_r^{(m)} = D(\mathbf{x}^{(m)})$ and $\mathbf{a}_g^{(m)} = D(G(\mathbf{z}^{(m)}))$. The term $D^T$ denotes the backward pass over the discriminator network D.

The second step maximizes the forward-pass likelihood $L_f^D$ for the discriminator $D$. This maximization is the same as minimizing the forward-pass error $E_f^D$ because the error just equals the negative log-likelihood:

$$E_f^D = -\sum_{m=1}^{M}\Big(\log\ D(\mathbf{x}^{(m)}) + \log\ \big(1 - G(D(\mathbf{z}^{(m)}))\big)\Big). \tag{119}$$

The forward-pass error is the cross entropy because the network acts as binary classifier with a Bernoulli probability density since the samples are either real or fake.

The third step trains the generator network $G$. The goal is to make $G(z)$ resemble $x$. So we train $G$ to minimize the error function $E^G$:

$$E^G = -\sum_{m=1}^{M}\log\ D(G(\mathbf{z}^{(m)}))\ . \tag{120}$$

$E^G$ measures the mismatch between the real and generated samples on the forward pass. Algorithm 2 lists the steps for the B-BP training of a vanilla GAN.

### 5.3. Bidirectional training of a Wasserstein GAN

We next show how B-BP can train a Wasserstein GAN. The measure of similarity is now the earth-mover distance between $p_r$ and $p_g$ in (112). The algorithm involves three steps for each training iteration. The first two steps train $D$ with B-BP. The third step trains $G$ with unidirectional BP. Algorithm 3 lists the steps for the B-BP training of a Wasserstein GAN.

**Data**: $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$ learning rate $\alpha$, batch size $M$, starting point for backward training $n^*$, number of epochs $L$, number of iterations $T$, and other hyperparameters.
**Result**: Trained weights $\Theta_d$ for discriminator $D$ and $\Theta_g$ for generator $G$.
**while** *epoch* $l : 1 \to L$ **do**
  **while** *iteration* $t : 1 \to T$ **do**
    • Select $M$ noise samples $\{\mathbf{z}^{(m)}\}_{m=1}^{M}$ from $p_z(\mathbf{z})$;
    • Select $M$ real samples $\{\mathbf{x}^{(m)}\}_{m=1}^{M}$ from $p_r(\mathbf{x})$;
    **if** $n^* \leqslant l$ **then**
      • Generate noise $\mathbf{n}_{x^{(m)}}$ for the input layer of the discriminator:
      **if** $\mathbf{n}_{x^{(m)}}^T\left(2\mathbf{x}^{(m)} - 2\mathbf{a}^{x^{(m)}} + \mathbf{n}_{x^{(m)}}\right) \leqslant 0$ **then**
        $\mathbf{x}_n^{(m)} \leftarrow \mathbf{x}^{(m)} + \mathbf{n}_{x^{(m)}}$
      **else**
        $\mathbf{x}_n^{(m)} \leftarrow \mathbf{x}^{(m)}$
      **end**
      • Update $\Theta_d$ for the backward-pass training of $D$:

$$\Theta_d \Leftarrow -\nabla_{\Theta_d} \sum_{m=1}^{M} \left( \left|\left| G\left(\mathbf{z}^{(m)}\right) - D^T\left(\mathbf{a}_r^{(m)}\right) \right|\right|^2 + \left|\left| \mathbf{x}_n^{(m)} - D^T\left(\mathbf{a}_g^{(m)}\right) \right|\right|^2 \right)$$

    **end**
    • Generate the respective noise $\mathbf{n}_{r^{(m)}}$ and $\mathbf{n}_{g^{(m)}}$ for the real and synthetic data at the output layer of the discriminator $D$:
    **if** $\mathbf{n}_{r^{(m)}}^T \log \frac{D(\mathbf{x}^{(m)})}{1-D(\mathbf{x}^{(m)})} \leqslant 0$ **then**
      $\mathbf{n}_{r^{(m)}} \leftarrow 0$
    **end**
    **if** $\mathbf{n}_{g^{(m)}}^T \log \frac{D(G(\mathbf{x}^{(m)}))}{1-D(G(\mathbf{x}^{(m)}))} \leqslant 0$ **then**
      $\mathbf{n}_{g^{(m)}} \leftarrow 0$
    **end**
    • Update $\Theta_d$ for the forward-pass training of $D$:

$$\Theta_d \Leftarrow -\nabla_{\Theta_d} \sum_{m=1}^{M} \left( \log \frac{D(\mathbf{x}^{(m)})}{1 - D(G(\mathbf{z}^{(m)}))} + \mathbf{n}_{r^{(m)}}^T \log \frac{D(\mathbf{x}^{(m)})}{1 - D(\mathbf{x}^{(m)})} + \mathbf{n}_{g^{(m)}}^T \log \frac{D(G(\mathbf{z}^{(m)}))}{1 - D(G(\mathbf{z}^{(m)}))} \right)$$

    • Update $\Theta_g$ for the training of $G$:

$$\Theta_g \Leftarrow -\nabla_{\Theta_g} \sum_{m=1}^{M} \left( \log\ D\left(G(\mathbf{z}^{(m)})\right) + \mathbf{n}_{g^{(m)}}^T \log \frac{D(G(\mathbf{z}^{(m)}))}{1 - D(G(\mathbf{z}^{(m)}))} \right)$$

  **end**
**end**

**Algorithm 2:** B-BP algorithm for training a vanilla generative adversarial network (GAN) with NEM noise injection. The algorithm injects NEM noise into both the input and output layers of the discriminator $D$.

**Data**: $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$ learning rate $\alpha$, batch size $M$, starting point for backward training $n^*$, number of epochs $L$, number of iterations $T$, gradient penalty coefficient $\lambda$, and other hyperparameters for the optimization.
**Result**: Trained weights $\Theta_d$ for discriminator $D$ and weights $\Theta_g$ for generator $G$.
**while** *epoch* $l : 1 \to L$ **do**
  **while** *iteration* $t : 1 \to T$ **do**
    • Select $M$ noise samples $\{\mathbf{z}^{(m)}\}_{m=1}^{M}$ from $p_z(\mathbf{z})$ ;
    • Select $M$ real samples $\{\mathbf{x}^{(m)}\}_{m=1}^{M}$ from $p_r(\mathbf{x})$ ;
    • Select a random number $\epsilon \sim U[0, 1]$;
    • Compute $\{\hat{\mathbf{x}}^{(m)}\}_{m=1}^{M}$ as follows:

$$\hat{\mathbf{x}}^{(m)} = \epsilon \mathbf{x}^{(m)} + (1 - \epsilon) G\left(\mathbf{z}^{(m)}\right)$$

    **if** $n^* \leqslant l$ **then**
      • Generate noise $\mathbf{n}_{x^{(m)}}$ for the input layer of the discriminator:
      **if** $\mathbf{n}_{x^{(m)}}^T\left(2\mathbf{x}^{(m)} - 2\mathbf{a}^{x^{(m)}} + \mathbf{n}_{x^{(m)}}\right) \leqslant 0$ **then**
        $\mathbf{x}_n^{(m)} \leftarrow \mathbf{x}^{(m)} + \mathbf{n}_{x^{(m)}}$
      **else**
        $\mathbf{x}_n^{(m)} \leftarrow \mathbf{x}^{(m)}$
      **end**
      • Update $\Theta_d$ for the backward-pass training of $D$ as follows:

$$\Theta_d \Leftarrow -\nabla_{\Theta_d} \sum_{m=1}^{M} \left( \left|\left| G\left(\mathbf{z}^{(m)}\right) - D^T\left(\mathbf{a}_r^{(m)}\right) \right|\right|^2 + \left|\left| \mathbf{x}_n^{(m)} - D^T\left(\mathbf{a}_g^{(m)}\right) \right|\right|^2 \right)$$

    • Update $\Theta_d$ for the forward-pass training of $D$ as follows:

$$\Theta_d \Leftarrow -\nabla_{\Theta_d} \sum_{m=1}^{M} \left( \left( D\left(\mathbf{x}^{(m)}\right) - D\left(G(\mathbf{z}^{(m)})\right) \right) + \lambda \left( \left|\left| \nabla_{\hat{x}} D\left(\hat{\mathbf{x}}^{(m)}\right) \right|\right|_2 - 1 \right)^2 \right)$$

    • Update $\Theta_g$ for the training of $G$:

$$\Theta_g \Leftarrow -\nabla_{\Theta_g} \sum_{m=1}^{M} D\left(G(\mathbf{z}^{(m)})\right)$$

  **end**
**end**

**Algorithm 3:** B-BP algorithm for training a Wasserstein generative adversarial network (WGAN) with NEM noise injection. Unidirectional BP updates the generator network. Bidirectional BP updates the discriminator network. NEM noise injects into the input layer of the discriminator.

## 6. Bidirectional simulation results

We simulated the performance of NEM-noise-boosted B-BP for different bidirectional network structures on the MNIST and CIFAR-10 image data sets. We also simulated multilayer BAMs to test the extent to which they converged or stabilized to BAM fixed points. Stability tended to fall off with an increased number of hidden layers. A general finding was that bipolar coding tended to outperform binary in most cases (Tables 2–6). This appears to

reflect the correlation-coefficient inequalities from the appendix to the original BAM paper (Kosko, 1988).

The classifier simulations used CIFAR-10 images (Krizhevsky & Hinton, 2009) for multilayer-perceptron (MLP) BAMs and for convolutional neural network (CNN) BAMs. The BAM structure of each network $N : R^n \rightarrow R^K$ arose from using the transpose matrices $M^T$ on the backward pass through the reverse network $N^T : R^K \rightarrow R^n$. B-BP trained the BAM models. We compared the effects of injecting NEM noise into both the input and output neurons with noiseless unidirectional BP and noiseless B-BP. We trained each model with 50,000 samples and tested the models on 10,000 test samples. The dataset used the following $K = 10$ pattern classes $C_k$: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

The forward pass through $N$ mapped an image pattern $\mathbf{x} \in C_k$ to an output probability vector $\mathbf{y} = N(\mathbf{x})$. The error compared this probability vector $\mathbf{y}$ with its class-label or unit bit-vector basis $\mathbf{e}_k$. The BAM classifiers used 10 output neurons with softmax activations. So the neural output vectors $\mathbf{y}$ always defined a probability vector of length 10.

The B-BP trained models jointly minimized the forward and backward error functions. The forward pass used a one-shot multinomial probability density for its likelihood function. So the forward error $E_f$ was cross-entropy. The backward pass mapped the class-label or unit basis vector $\mathbf{e}_k$ to the pattern vector $\mathbf{x}' = N^T(\mathbf{e}_k)$. Some experiments mapped the raw output $\mathbf{y} = N(\mathbf{x})$ back through $N^T$ to produce $\mathbf{x}' = N^T(\mathbf{e}_k)$. We computed the error by comparing this backward-passed vector $\mathbf{x}'$ with the nearest class-$C_k$ vector to the sample centroid of class $C_k$ (sample centroids need not always lie in $C_k$). The $K$ input neurons involved in the backward pass used an $I$-dimensional vector normal probability density as its likelihood function. That gave the negative log-likelihood as the squared error. So the backward error $E_b$ was just this squared error.

We next describe the NEM-boosted performance of the multilayer and convolutional BAM classifiers. Then we describe the NEM-boosted performance of B-BP applied to vanilla and Wasserstein GANs.

### 6.1. Deep bidirectional neural classifiers and NEM B-BP

Deep bidirectional classifiers $N$ are multilayer BAMs that run forward through the matrix-based synaptic webs and run backward through the matrix transposes in $N^T$. B-BP trains such classifiers with cross entropy in the forward direction and squared error in the backward direction as discussed above. Let $I$ denote the number of neurons at the input layer. The multilayer BAMs used 3072 input neurons for the CIFAR-10 dataset because each sample image had dimension $32 \times 32 \times 3$.

We tested BAM classifiers with 1, 3, 5, and 7 hidden layers of rectified-linear unit (ReLU) activations. The BAM models used 512 neurons at each hidden layer. We used the dropout pruning method (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014) to prune hidden neurons. A dropout value of 0.8 for the hidden layers reduced overfitting. So there was a 20% probability of pruning a given hidden neuron.

We compared B-BP with NEM noise injection with noiseless B-BP and with B-BP that injects only blind or dithering noise. We injected the noise only into the output and input layers of the classifier networks. Injecting NEM noise into hidden layers involves a more complicated algorithm that depends on the type of hidden neurons and the structure of the corresponding layer likelihood as in (10) (Kosko et al., 2019). Injecting NEM noise into hidden layers also tends to have less benefit than injecting it into the output layer of a classifier network.

The simulation results showed that B-BP NEM noise injection outperformed noiseless B-BP and blind-noise B-BP both in training speed-ups and classification accuracy. Table 8 shows that NEM noise injection performed best with respect to classification accuracy. Blind noise achieved better classification accuracy than noiseless B-BP in some cases. It performed worse in others. NEM noise injection outperformed blind noise injection in all cases.

Fig. 4 shows the highest and lowest benefit in classification accuracy with NEM noise injection. Table 8 shows that NEM noise improved classification accuracy compared with noiseless B-BP and with noiseless unidirectional BP. A BAM classifier with 7 hidden layers had a B-BP NEM-noise-boosted classification accuracy of 59.8% compared with 56.9% for noiseless B-BP.

Table 9 shows that NEM noise injection always speeded up B-BP training on the CIFAR-10 dataset. Injected blind noise also slowed down B-BP training. The speed-up in training measured how many fewer epochs it took the trained network to reach the final point when trained with noiseless B-BP. The benchmark was the output cross entropy of noiseless B-BP after 100 training epochs. A negative speed-up meant that training took more iterations to reach the noiseless benchmark.

NEM noise injection always produced a positive speed-up in training compared with noiseless B-BP. The best NEM noise speed-up was 68% in the BAM classifier with 1 hidden layer. Blind noise produced a speed-up in some cases and a slow-down in others. Blind-noise benefits tended to be slight. Table 9 shows that blind noise only slowed down training.

### 6.2. Deep convolutional neural classifiers and NEM B-BP

We trained bidirectional convolutional neural network (CNN) classifiers using the CIFAR-10 dataset. The CNNs used 4 convolutional layers and 2 fully connected layers. B-BP trained the convolutional masks for the bidirectional mapping. The forward pass ran a convolution (time-reversed correlation) with masks. The backward pass ran the transpose convolution (Dumoulin & Visin, 2016) with masks. Figs. 5 and 6 show that the backward passes in these deep networks estimate pattern-class centroids.

The forward pass across convolutional layers used downsampling with the pooling operations. The backward pass with the convolutional mask used upsampling. The transpose convolutional operation used the stride parameter to upsample along the backward pass. A convolutional mask was a bidirectional filter with the B-BP algorithm.

The size of all the convolutional masks was $3 \times 3$. The CNN models used 32 masks at the first convolutional layer and used 64 masks at the second convolutional layer. The third and fourth layers used 128 convolutional masks each. The first fully connected layer $fc1$ connected the output of the convolutional layers to the second fully connected layer $fc2$. The $fc1$ layer used 2048 neurons and the $fc2$ layer used 1024 neurons. The $fc2$ layer used 10 neurons to connect to the output layer.

The forward pass used average pooling after the second, third, and fourth convolutional layers to downsample the features. We used a pooling factor of 2. The backward pass used the stride factor of the transpose convolution operation at the second, third, and fourth convolutional layers to upsample the features. The stride factor was 2.

The forward-pass performance measure or error $E_f$ was cross entropy. The backward error $E_b$ was squared error since these were classification-regression bidirectional networks. Different modes of B-BP trained the CNNs.

Table 10 shows that there was a slight boost in classification accuracy with NEM noise injection in B-BP. B-BP with NEM noise injection outperformed the other modes of B-BP. It also performed better than unidirectional BP.

Fig. 6 compares the backward pass of the class-label or unit-bit-vector basis vectors $\mathbf{e}_k$ through the reverse network $N^T$. The

**Table 8**
Classification accuracy of multilayered BAM networks trained with backpropagation on the CIFAR10 dataset. The deep bidirectional networks used 1, 3, 5, and 7 hidden layers with 512 ReLU neurons for each hidden layer. Blind or NEM noise injected into the output and input neurons if at all.

| Network configuration | 1 hidden layer | 3 - hidden layer | 5 hidden layer | 7 hidden layer |
|---|---|---|---|---|
| Unidirectional BP | 53.89% | 57.43% | 57.83% | 57.12% |
| B-BP without noise | 54.26% | 58.74% | 58.18% | 56.90% |
| B-BP with blind noise | 53.86% | 57.54% | 58.09% | 55.49% |
| B-BP with NEM noise | 55.10% | 59.43% | 59.64% | 59.80% |

**Table 9**
Speed-ups and slow-downs in bidirectional training due to noise injection in the output softmax neurons of multilayer classifier BAMs trained on the CIFAR-10 dataset. The percentages are changes in the output cross entropy after 100 training epochs compared with noiseless B-BP.

| | B-BP with blind noise | B-BP with NEM noise |
|---|---|---|
| 1- hidden layer | −6.0% | 30.0% |
| 3- hidden layers | −26.0% | 40.0% |
| 5- hidden layers | −8.0% | 28.0% |
| 7- hidden layers | −16.0% | 32.0% |

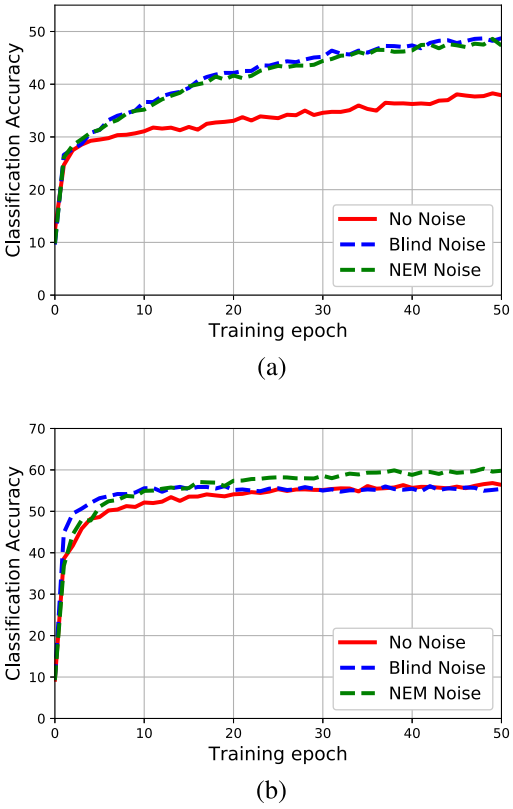



**Fig. 4.** NEM-noise benefits in classification accuracy for multilayered BAMs and B-BP training using the CIFAR-10 dataset. Simulations compared NEM-boosted B-BP with noiseless B-BP and with B-BP with injected blind noise. (a) Multilayer BAM classifier with 1 hidden layer and 1024 hidden ReLU neurons. NEM noise and blind noise injection outperformed noiseless B-BP . (b) Multilayer BAM with 7 hidden layers with 512 ReLU neurons at each hidden layer. NEM-boosted B-BP with NEM outperformed both noiseless B-BP and B-BP with blind noise injection.

backward-pass patterns on the B-BP trained CNNs were similar to the targets $\tilde{\mathbf{x}}$. The backward-pass patterns that arrived at the input of the unidirectional-BP trained network did not visually resemble the target images. Table 11 shows the speed-up in training due to noise injection. Injecting NEM noise in B-BP training always outperformed injecting blind noise.

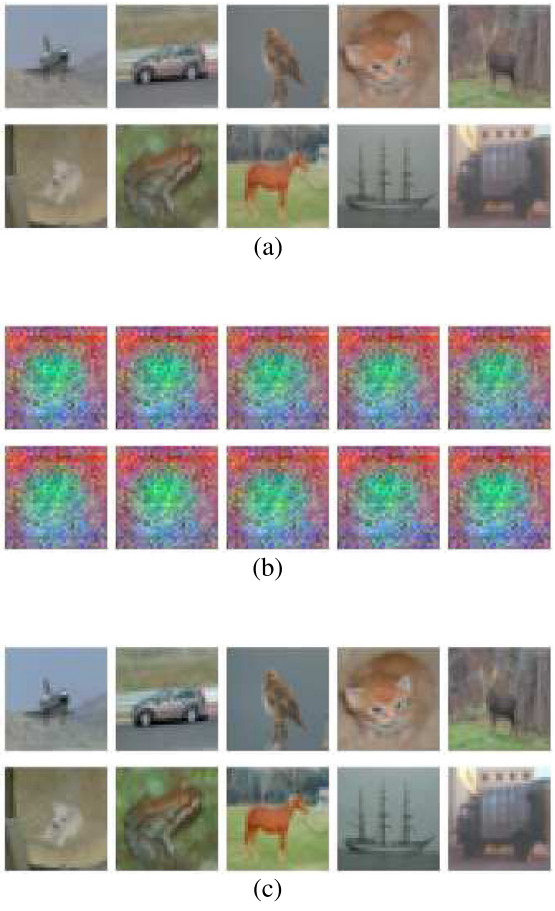



**Fig. 5.** Backward pass of the class-label unit-bit-vector output basis vectors $\mathbf{e}_k$ for a multilayered BAM trained on the CIFAR-10 dataset. The multilayered BAM used 7 hidden layers and swept forward and backward through those layers. (a) The sample class centroids of the 10 image pattern classes. (b) Backward-pass prediction of the class-label output vectors $\mathbf{e}_k$ with unidirectional BP: The network failed to produce meaningful backward-pass feedback signals. (c) Backward-pass prediction of the class-label output vectors $\mathbf{e}_k$ with B-BP and NEM noise injection: The feedback signals closely matched the corresponding sample class centroids.

**Table 10**
Classification accuracy of convolutional neural networks trained on CIFAR-10 images.

| | Accuracy |
|---|---|
| Unidirectional BP | 76.06% |
| B-BP without noise | 76.44% |
| B-BP with blind noise | 77.13% |
| B-BP with NEM noise | 78.64% |

### 6.3. NEM B-BP and adversarial networks

The adversarial simulations used the MNIST digit data (LeCun, Bottou, Bengio, & Haffner, 1998) for both vanilla GANs and

**Table 11**
Speed-up in training due to noise injection in B-BP trained CNN models on the CIFAR10 dataset. The reference is B-BP without noise injection.

| | Speed-Up |
|---|---|
| B-BP with blind noise | 16.0% |
| B-BP with NEM noise | 24.0% |

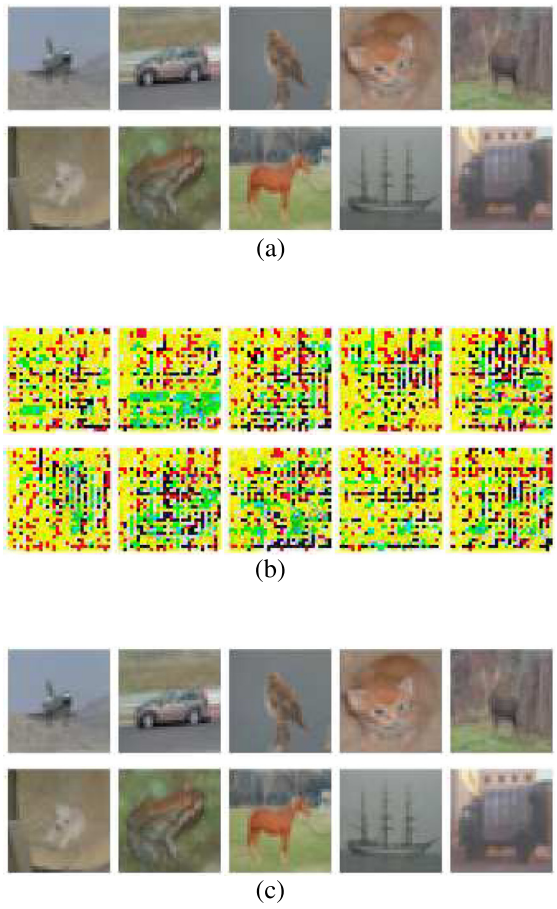



**Fig. 6.** Backward pass of the class-label unit-bit-vector output basis vectors $\mathbf{e}_k$ for a multilayered *convolutional* BAM CNN trained on the CIFAR-10 dataset. (a) The sample class centroids of the 10 image pattern classes $C_k$. (b) Backward-pass prediction of the class-label output vectors $\mathbf{e}_k$ with unidirectional BP only: The network failed to produce meaningful backward-pass feedback signals. (c) Backward-pass prediction of the class-label output vectors $\mathbf{e}_k$ with B-BP and NEM noise injection in the CNN: The feedback signals closely matched the corresponding sample class centroids on the CIFAR-10 dataset.

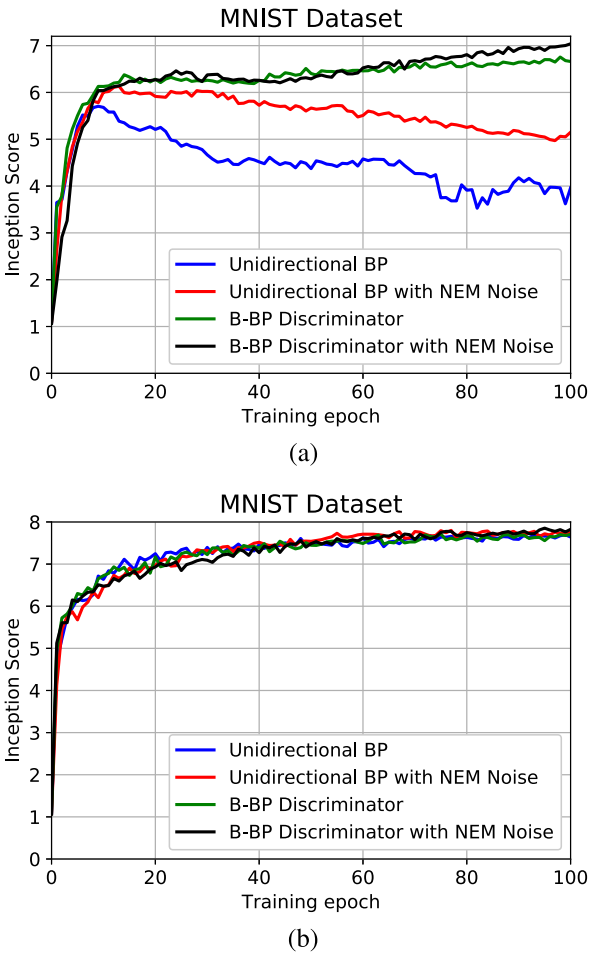



**Fig. 7.** Performance of the vanilla and Wasserstein GANs on the MNIST dataset. (a) Inception scores for regular vanilla GANs trained on the MNIST dataset. Training with bidirectional BP and NEM noise injection gave the best performance in terms of the highest inception score (the exponentiated Kullback–Leibler entropic distance in Eq. (121)). All GANs trained with unidirectional BP suffered from mode collapse: They often produced the same image. The B-BP trained vanilla GANs did not suffer from mode collapse. NEM noise injection also improved the quality of the generated images for both unidirectional BP and bidirectional BP. (b) Inception scores for the Wasserstein GANs trained on the MNIST dataset. Training with B-BP and NEM noise gave the best inception scores. B-BP and NEM noise injection only slightly improved the WGAN performance and generated images.

Wasserstein GANs. Separate simulations used the CIFAR-10 images (Krizhevsky & Hinton, 2009) for the vanilla and Wasserstein GANs. The image data did require that we extend the vanilla GAN to a deep convolutional network.

*6.3.1. Training a vanilla GAN with B-BP*

We now describe the simulation results for the B-BP training of vanilla GANs. The GAN's generator $G$ and discriminator $D$ were multilayered neural networks. The generator network $G$ mapped the space of the latent variable $z$ to the space of generated samples. The discriminator network $D$ mapped these samples to a single output logistic neuron. We trained the vanilla GANs only on the MNIST dataset.

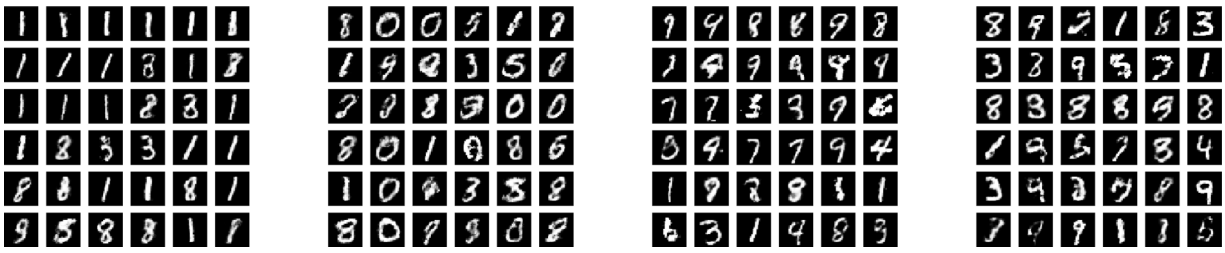
We modeled the latent variable $\boldsymbol{z}$ as a bipolar uniform random variable: $z \sim U[-1, 1]$. We used Goodfellow's GAN *inception score* $IS(G)$ (Salimans et al., 2016) based on the exponentiated average Kullback–Leibler divergence:

$$IS(G) = \exp\left(\mathbb{E}_{x\sim p_g}[D_{KL}\left(p(y|x) \parallel p(y)\right)]\right) \tag{121}$$

for the generator $G$. The Kullback–Leibler divergence or relative entropy $D_{KL}\left(p(y|x) \parallel p(y)\right)$ measures the pseudo-distance between the conditional probability density $p(y|x)$ and the corresponding unconditional marginal density $p(y)$. A higher inception score $IS(G)$ tends to describe generated images of better quality. So a larger inception score corresponds to better GAN performance. Table 12 shows the inception scores for the vanilla GANs trained on the MNIST dataset. Table 13 shows the inception scores for the Wasserstein GANs trained on the same dataset.

The generator network $G$ used 100 input neurons with identity activations. It had a single hidden layer with 128 logistic sigmoid neurons. The output layer contained 784 logistic neurons.

The discriminator network $D$ used 784 input neurons with identity activations. It had a single hidden layer of 128 logistic neurons. The output layer had a single logistic neuron.



(a) (b) (c) (d)

**Fig. 8.** Sample generated digit images from vanilla GANs trained with unidirectional BP and B-BP on the MNIST data set. These generated samples came from the trained GANs after 100 training epochs. (a) Unidirectional BP. (b) Bidirectional BP without noise. (c) Unidirectional BP with NEM noise. (d) Bidirectional BP with NEM noise.

(a) (b) (c) (d)

**Fig. 9.** Sample generated images from deep-convolutional GANs trained with unidirectional BP and B-BP on the CIFAR-10 data set. We generated these samples after training the GANs over 100 epochs. (a) Unidirectional BP. (b) Bidirectional BP without noise. (c) Unidirectional BP with NEM noise. (d) Bidirectional BP with NEM noise.

NEM-boosted B-BP gave the best performance for both vanilla and Wasserstein GANs. NEM noise gave more pronounced benefits for vanilla GANs. We presume this was because the Wasserstein GANs involved more processing and had less room for improvement. Fig. 7 compares the performance of the vanilla and Wasserstein GANs trained on the MNIST dataset of handwritten digits. Fig. 8 shows sample MNIST-like images that the vanilla GANs generated. Fig. 11 shows sample MNIST-like images that the Wasserstein GANs generated. Fig. 10 compares the performance of the deep-convolutional and Wasserstein GANs trained on the CIFAR-10 image dataset. Fig. 9 shows sample CIFAR-10-like images that the deep-convolutional GANs generated. Fig. 12 shows sample CIFAR-10-like images that the Wasserstein GANs generated.

We also trained a deep-convolutional GAN (DCGAN) on the CIFAR-10 image set. The DCGAN used the same error function as did the vanilla GAN for the MNIST dataset. The DCGAN had

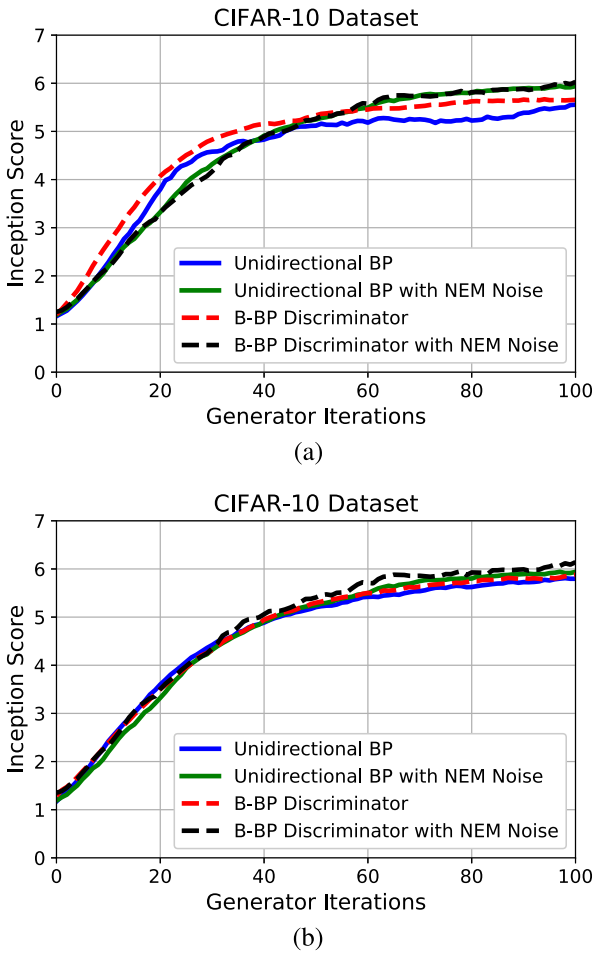



**Fig. 10.** Performance of deep convolutional GANs (DCGANs) and Wasserstein GANs on the CIFAR10 dataset. (a) DCGAN inception scores: NEM noise outperformed noiseless unidirectional BP and noiseless B-BP in the discriminator. (b) Wasserstein GAN inception scores: NEM noise also performed best but with less marked effects than in (a).

3 convolutional layers and no max-pooling layer (Radford, Metz, & Chintala, 2015). The forward pass convolved the input data with masks $W_i^*$. The backward pass used the transposed masks $W_i^*$ (in true bidirectional fashion) to convolve signals. The reverse convolution was the transpose convolution operation (Dumoulin & Visin, 2016). There was no pooling at the convolutional layers for either the forward or backward pass.

The discriminator's 3 convolutional layers used leaky rectified linear (ReLU) activations and a fully connected layer between the input and output layers. The generator $G$ also used 3 convolutional layers with rectified linear activations and a fully connected layer between the input and output layer. The generator $G$ used 128 input neurons with identity activations.

**Table 12**
Inception scores for vanilla GANs trained on the MNIST dataset.

| | Inception score |
|---|---|
| Unidirectional BP | 3.96 ± 0.07 |
| Unidirectional BP with NEM noise | 5.15 ± 0.10 |
| B-BP discriminator | 6.66 ± 0.10 |
| **B-BP discriminator with NEM noise** | **7.04.** ± **0.11** |

**Table 13**
Inception scores for Wasserstein GANs trained on the MNIST dataset.

| | Inception score |
|---|---|
| Unidirectional BP | 7.67 ± 0.10 |
| Unidirectional BP with NEM noise | 7.71 ± 0.11 |
| B-BP discriminator | 7.75 ± 0.12 |
| **B-BP discriminator with NEM noise** | **7.80** ± **0.12** |

**Table 14**
Inception scores for deep-convolutional GANs trained on the CIFAR-10 dataset.

| | Inception score |
|---|---|
| Unidirectional BP | 5.55 ± 0.20 |
| Unidirectional BP with NEM noise | 5.96 ± 0.15 |
| B-BP discriminator | 5.66 ± 0.16 |
| **B-BP discriminator with NEM noise** | **6.03.** ± **0.17** |

**Table 15**
Inception scores for Wasserstein GANs trained on the CIFAR-10 dataset.

| | Inception score |
|---|---|
| Unidirectional BP | 5.80 ± 0.46 |
| Unidirectional BP with NEM noise | 5.98 ± 0.29 |
| B-BP discriminator | 5.96 ± 0.23 |
| **B-BP discriminator with NEM noise** | **6.10.** ± **0.25** |

Table 14 shows that NEM-boosted B-BP gave the best inception-score performance for both the vanilla GAN trained on the MNIST data set and the DCGAN trained on the CIFAR-10 image set. The inception score $IS(G)$ in (121) uses an exponentiated Kullback–Leibler entropic distance to measure the image quality. A higher inception score corresponds to better GAN performance. Table 15 shows likewise that Wasserstein GANs performed best with NEM noise and a bidirectional discriminator.

## 7. Conclusions

NEM noise injection improved the performance of bidirectional backpropagation on classifiers both in terms of increased accuracy and shorter training time. It often required fewer hidden

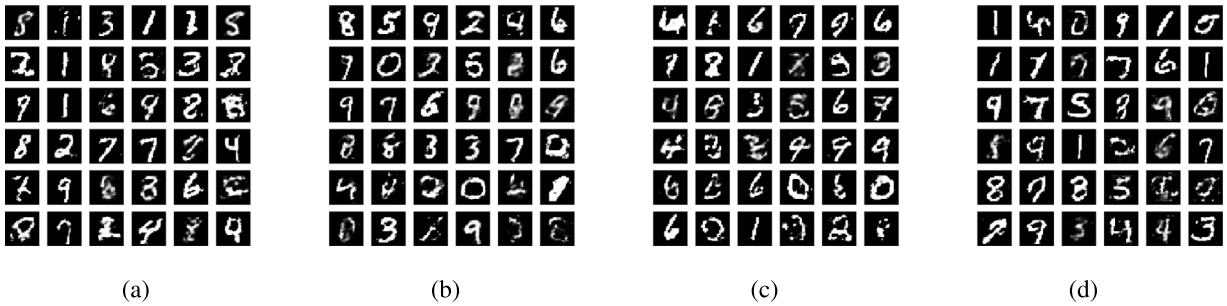



**Fig. 11.** Generated digit images from Wasserstein GANs trained with unidirectional BP and B-BP on the MNIST data set. These generated samples came training the trained GANs after 100 training epochs. (a) Unidirectional BP. (b) Bidirectional BP without noise. (c) Unidirectional BP with NEM noise. (d) Bidirectional BP with NEM noise.


**Fig. 12.** Generated images from Wasserstein GANs trained with unidirectional BP and B-BP on the CIFAR-10 image data set. We generated these samples after training the GANs over 100 epochs. (a) Unidirectional BP. (b) Bidirectional BP without noise. (c) Unidirectional BP with NEM noise. (d) Bidirectional BP with NEM noise.

neurons or layers to achieve the same performance as noiseless B-BP. The NEM noise benefits were even more pronounced for convolutional BAM classifiers. NEM noise injects just that noise that makes the current signal more probable. It almost always outperformed simply injecting blind noise. B-BP also outperformed ordinary unidirectional BP. Injecting NEM noise into a vanilla generative adversarial network improved its inception score and greatly reduced the problem of mode collapse. It also improved the performance of Wasserstein GANs but the benefit was less pronounced. NEM noise can also inject into the hidden-layer neurons of these multilayer BAM networks.

## References


Adigun, O., & Kosko, B. (2016). Bidirectional representation and backpropagation learning. In *International joint conference on advances in big data analytics* (pp. 3–9).

Adigun, O., & Kosko, B. (2017). Using noise to speed up video classification with recurrent backpropagation. In *International joint conference on neural networks* (pp. 108–115). IEEE.

Adigun, O., & Kosko, B. (2018). Training generative adversarial networks with bidirectional backpropagation. In *2018 17th IEEE international conference on machine learning and applications* (pp. 1178–1185). IEEE.

Adigun, O., & Kosko, B. (2019). Bidirectional backpropagation. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*.

Ali, M., Yogambigai, J., Saravanan, S., & Elakkia, S. (2019). Stochastic stability of neutral-type markovian-jumping BAM neural networks with time varying delays. *Journal of Computational and Applied Mathematics*, *349*, 142–156.

Arjovsky, M., & Bottou, L. (2017). Towards principled methods for training generative adversarial networks. In *International conference on learning representations*.

Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein GAN. arXiv preprint arXiv:1701.07875.

Audhkhasi, K., Osoba, O., & Kosko, B. (2016). Noise-enhanced convolutional neural networks. *Neural Networks*, *78*, 15–23.

Bengio, Y., et al. (2009). Learning deep architectures for ai. *Foundations and trends in Machine Learning*, *2*(1), 1–127.

Bhatia, S., & Golman, R. (2019). Bidirectional constraint satisfaction in rational strategic decision making. *Journal of Mathematical Psychology*, *88*, 48–57.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

Cohen, M. A., & Grossberg, S. (1983). Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. *IEEE Transactions on Systems, Man, and Cybernetics*, (5), 815–826.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B. Statistical Methodology*, *39*(1), 1–22.

Dumoulin, V., & Visin, F. (2016). A guide to convolution arithmetic for deep learning. ArXiv e-prints arXiv:1603.07285.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, *36*(4), 193–202.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., et al. (2014). Generative adversarial nets. In *Advances in neural information processing systems* (pp. 2672–2680). NIPS.

Graves, A., Mohamed, A.-r., & Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing* (pp. 6645–6649). IEEE.

Grossberg, S. (1976). Adaptive pattern classification and universal recoding: II. Feedback, expectation, olfaction, illusions. *Biological Cybernetics*, *23*(4), 187–202.
Grossberg, S. (1982). How does a brain build a cognitive code? In *Studies of mind and brain* (pp. 1–52). Springer.
Grossberg, S. (1988). Nonlinear neural networks: Principles, mechanisms, and architectures. *Neural Networks*, *1*(1), 17–61.
Grossberg, S. (2017). Towards solving the hard problem of consciousness: The varieties of brain resonances and the conscious experiences that they support. *Neural Networks*, *87*, 38–95.
Gubner, J. A. (2006). *Probability and random processes for electrical and computer engineers*. Cambridge University Press.
Gulrajani, I., Ahmed, F., Arjovsky, M., V. Dumoulin, V., & Courville, A. (2017). Improved training of wasserstein GANs. arXiv preprint arXiv:1704.00028.
Hinton, G. E., Rumelhart, D., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(9), 533–536.
Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.
Hogg, R., Mckean, J., & Craig, A. (2012). Introduction to mathematical statistics. (7th ed.). *Pearson Education*.
Jain, A. K. (2010). Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, *31*(8), 651–666.
Jordan, M., & Mitchell, T. (2015). Machine learning: trends, perspectives, and prospects. *Science*, *349*, 255–260.
Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE*, *78*(9), 1464–1480.
Kosko, B. (1987). Adaptive bidirectional associative memories. *Applied Optics*, *26*(23), 4947–4960.
Kosko, B. (1988). Bidirectional associative memories. *IEEE Transactions on Systems, Man and Cybernetics*, *18*(1), 49–60.
Kosko, B. (1990). Unsupervised learning in noise. *Neural Networks, IEEE Transactions on*, *1*(1), 44–57.
Kosko, B. (1991a). *Neural networks and fuzzy systems: A dynamical systems approach to machine intelligence*. Prentice Hall.
Kosko, B. (1991b). Stochastic competitive learning. *IEEE Transactions on Neural Networks*, *2*(5), 522–529.
Kosko, B. (2018). Additive fuzzy systems: From generalized mixtures to rule continua. *International Journal of Intelligent Systems*, *33*(8), 1573–1623.
Kosko, B., Audkhasi, K., & Osoba, O. (2019). *Noise Can Speed Backpropagation Learning and Deep Bidirectional Pretraining*. Elsevier, submitted for publication.
Krizhevsky, A., & Hinton, G. (2009). *Learning multiple layers of features from tiny images*. Technical report, University of Toronto.
Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097–1105).
LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*, 436–444.
LeCun, Y., Bengio, Y., et al. (1995). Convolutional networks for images, speech, and time series. In *The handbook of brain theory and neural networks*: *Vol. 3361*, (10), (p. 1995).
LeCun, Y., Bottou, L., Bengio, Y., & Haffner (1998). Gradient-based learning applied to document recognition. In *Proceedings of the IEEE: Vol. 86*, (11) (pp. 2278–2324). IEEE.
Maharajan, C., Raja, R., Cao, J., Rajchakit, G., & Alsaedi, A. (2018). Impulsive Cohen–Grossberg BAM neural networks with mixed time-delays: an exponential stability analysis issue. *Neurocomputing*, *275*, 2588–2602.
Metz, L., Poole, B., D. Pfau, D., & Sohl-Dickstein, J. (2016). Unrolled generative adversarial networks. arXiv preprint arXiv:1611.02163.
Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2018). *Foundations of machine learning*. MIT press.
Osoba, O., & Kosko, B. (2013). Noise-enhanced clustering and competitive learning algorithms. *Neural Networks*, *37*, 132–140.
Osoba, O., & Kosko, B. (2016). The noisy expectation-maximization algorithm for multiplicative noise injection. *Fluctuation and Noise Letters*, 1650007.
Osoba, O., Mitaim, S., & Kosko, B. (2013). The noisy expectation–maximization algorithm. *Fluctuation and Noise Letters*, *12*(3), 1350012–1–1350012–30.
Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434.
Rumelhart, D., Hinton, G., & Williams, R. (1986). Learning representationsby back-propagating errors. *Nature*, 323–533.
Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., & Radford, A. (2016). Improved techniques for training GANs. arXiv preprint arXiv:1606.0349.
Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research (JMLR)*, *15*(1), 1929–1958.
Villani, C. (2009). *Optimal transport: Old and new*. Springer, Berlin,
Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., & Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research (JMLR)*, *11*(Dec), 3371–3408.
Wang, F., Chen, Y., & Liu, M. (2018). pth moment exponential stability of stochastic memristor-based bidirectional associative memory (BAM) neural networks with time delays. *Neural Networks*, *98*, 192–202.
Werbos, P. J. (1974). *Beyond regression: New tools for prediction and analysis in the behavioral sciences* (Doctoral Dissertation), MA: Applied Mathematics, Harvard University.