Neural Networks 78 (2016) 15–23

Contents lists available at ScienceDirect

Neural Networks

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

2016 Special Issue

Noise-enhanced convolutional neural networks

Kartik Audhkhasi, Osonde Osoba, Bart Kosko *

Signal and Information Processing Institute, Electrical Engineering Department, University of Southern California, Los Angeles, CA, United States

ARTICLE INFO

Article history:
Available online 19 October 2015

Keywords:
Backpropagation
Noise injection
Convolutional neural network
Expectation–maximization algorithm
Stochastic resonance
Sampling from big data sets

ABSTRACT

Injecting carefully chosen noise can speed convergence in the backpropagation training of a convolutional neural network (CNN). The Noisy CNN algorithm speeds training on average because the backpropagation algorithm is a special case of the generalized expectation–maximization (EM) algorithm and because such carefully chosen noise always speeds up the EM algorithm on average. The CNN framework gives a practical way to learn and recognize images because backpropagation scales with training data. It has only linear time complexity in the number of training samples. The Noisy CNN algorithm finds a special separating hyperplane in the network's noise space. The hyperplane arises from the likelihood-based positivity condition that noise-boosts the EM algorithm. The hyperplane cuts through a uniform-noise hypercube or Gaussian ball in the noise space depending on the type of noise used. Noise chosen from above the hyperplane speeds training on average. Noise chosen from below slows it on average. The algorithm can inject noise anywhere in the multilayered network. Adding noise to the output neurons reduced the average per-iteration training-set cross entropy by 39% on a standard MNIST image test set of handwritten digits. It also reduced the average per-iteration training-set classification error by 47%. Adding noise to the hidden layers can also reduce these performance measures. The noise benefit is most pronounced for smaller data sets because the largest EM hill-climbing gains tend to occur in the first few iterations. This noise effect can assist random sampling from large data sets because it allows a smaller random sample to give the same or better performance than a noiseless sample gives.

1. Noise-boosted convolutional neural networks

This paper presents the Noisy Convolutional Neural Network (NCNN) algorithm for speeding up the backpropagation (BP) training of convolutional neural networks (CNNs). Fig. 1 shows the architecture of a CNN with a single hidden convolutional layer and 3 convolutional masks or retinal-like receptive fields. CNNs are standard feedforward neural networks for large-scale image recognition (Cireşan, Meier, Masci, Gambardella, & Schmidhuber, 2011; Krizhevsky, Sutskever, & Hinton, 2012; Lawrence, Giles, Tsoi, & Back, 1997; LeCun et al., 1990; LeCun, Bottou, Bengio, & Haffner, 1998; Matsugu, Mori, Mitari, & Kaneda, 2003; Simard, Steinkraus, & Platt, 2003; Szarvas, Yoshizawa, Yamamoto, & Ogata, 2005). Some deep neural nets use on the order of 20 hidden layers of neurons (LeCun, Bengio, & Hinton, 2015; Simonyan & Zisserman, 2014). The NCNN algorithm can noise-boost a CNN with any number of hidden layers so long as the injected noise lies above the NCNN hyperplane in noise space.

The NCNN algorithm exploits two recent theoretical results—a reduction and a noise boost.

The first result is that the BP algorithm is a special case (Audhkhasi, Osoba, & Kosko, 2013a) of the generalized expectation–maximization (EM) algorithm for iteratively maximizing a likelihood or log-likelihood (Dempster, Laird, & Rubin, 1977). We restate and prove this result below as Theorem 1. BP and generalized EM are both iterative gradient algorithms. The proof shows that their log-likelihood gradients are equal at each training epoch. This embeds BP in the general framework of maximum likelihood estimation. Fig. 2 shows this BP–EM correspondence at the system level. BP's forward step corresponds to the expectation or E-step. Its backward error pass corresponds to the maximization or M-step. The network's hidden-layer parameters correspond to EM's latent variables.

BP that minimizes training-sample squared error equally maximizes a likelihood in the form of the exponential of the negative squared error. Maximizing the log-likelihood also minimizes the squared error. This "classical" squared-error or least-squares BP case (Rumelhart, Hinton, & Williams, 1986) occurs in the likelihood framework when the output neurons have conditionally Gaussian activation functions (Bishop, 2006). We achieved better

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

http://dx.doi.org/10.1016/j.neunet.2015.09.014
0893-6080/

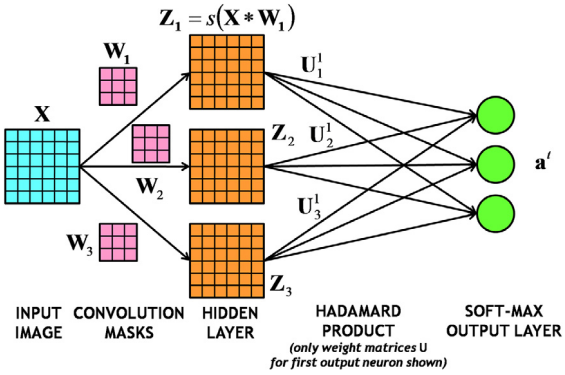

Fig. 1. Convolutional Neural Network (CNN): The diagram shows a CNN with just one hidden convolutional layer. The input image $\mathbf{X}$ convolves with the 3 masks $\mathbf{W}_1$, $\mathbf{W}_2$, and $\mathbf{W}_3$. These masks act as receptive fields in the retina. The resulting images pass pixel-wise through the hidden neurons with logistic sigmoid activations. Then the CNN computes element-wise Hadamard products between the hidden neuron activation matrices $\mathbf{Z}_1$, $\mathbf{Z}_2$, and $\mathbf{Z}_3$ with weight matrices $\mathbf{U}_j^k$ where $j = 1, 2, 3$ and $k = 1, 2, 3$. The output neurons have softmax activations that define a discrete Gibbs probability density function.

CNN recognition performance with BP that minimized the more common training-set cross entropy in (6).

Using cross entropy corresponds to using output neurons that use the softmax or Gibbs activation functions a_k^t (Bishop, 2006) in (5). Softmax neurons have the form of an exponential divided by a sum of exponentials. So the activations a_k^t lie in the unit interval and define probabilities because they sum to one. Each softmax activation itself defines a discrete Gibbs probability density function. But the K output neurons or activations are independent of one another. Independence factors the joint probability or likelihood $p(y|\mathbf{x}, \Theta)$ of the K output neurons where the network's output y depends on the input $\mathbf{x}$ and the network parameters Θ. This factorization produces a categorical probability density function at the output layer: $p(y|\mathbf{x}, \Theta) = \prod_{k=1}^{K} (a_k^t)^{y_k}$ because the exponents y_k are binary and sum to one. This holds because the output target vectors encode patterns as unit binary vectors. The categorical density corresponds to a one-trial multinomial density. Then taking the logarithm of $p(y|\mathbf{x}, \Theta)$ shows that this log-likelihood is just the cross entropy (6).

So minimizing cross entropy maximizes the log-likelihood and leads to a BP form of multinomial regression for multiple categories of images or other patterns. Section 3 further shows that minimizing the cross entropy in this case minimizes the Kullback–Liebler divergence or relative entropy.

The second theoretical result is that carefully chosen and injected noise speeds convergence of the EM algorithm on average as it iteratively climbs the nearest hill of likelihood (Osoba & Kosko, 2016; Osoba, Mitaim, & Kosko, 2011, 2013). Theorem 2 restates this result. Then we show that this guaranteed EM noise boost gives rise to a noise-space hyperplane condition for training CNNs with backpropagation: Noise chosen from above the NCNN hyperplane speeds CNN training on average. Noise chosen from below slows it.

This NCNN-hyperplane result may help explain anecdotal reports that randomly chosen noise sometimes gives a slight boost in training performance. We would expect that on average such blind noise should contain roughly the same number of noise samples from above the NCNN hyperplane as from below it. But some cases should also contain more samples from above the NCNN hyperplane. These cases should enjoy at least a mild noise boost.

The NCNN algorithm also holds promise for big data applications. There are least two main reasons for this.

The first reason is that training BP scales only linearly with sample size. Training BP with n samples incurs only linear $O(n)$ time complexity. Linear time complexity holds because the forward or predictive pass of BP has only $O(1)$ complexity. The more involved backward pass has $O(n)$ complexity. BP's overall linear complexity contrasts with the $O(n^2)$ time complexity of modern support-vector kernel methods (Kung, 2014). The quadratic complexity of such kernel methods arises from the $O(n)$ complexity of their predictive pass. The recent Fastfood kernel algorithm (Le, Sarlós, & Smola, 2013) reduces the $O(n^2)$ kernel complexity to $O(n \log d)$ for n nonlinear basis functions in d dimensions. Fastfood's loglinear complexity appears to be the current lower bound for kernel methods. The computational difference between $O(n)$ and $O(n \log d)$ time complexities in practice can be substantial for large-scale problems of big data.

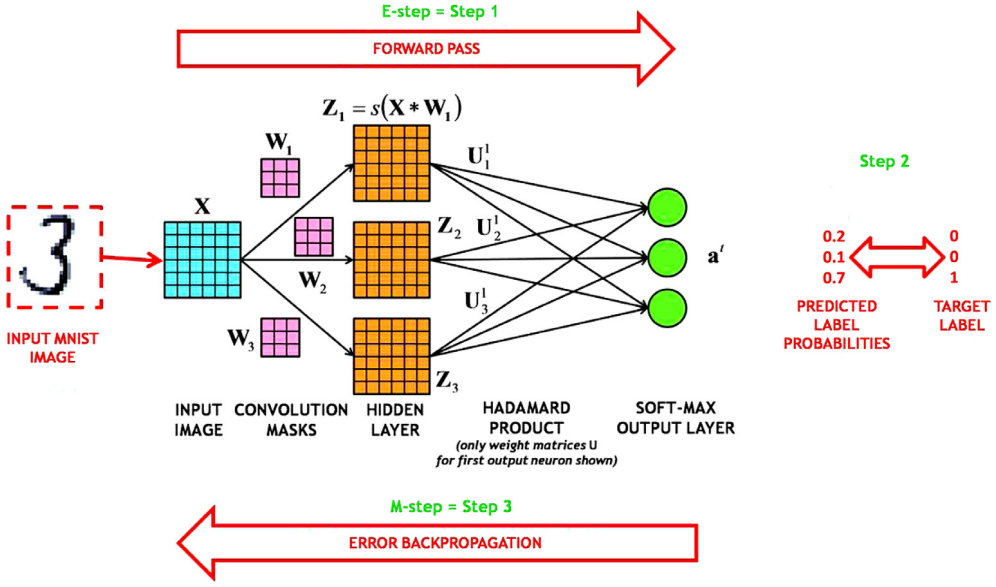

Fig. 2. Backpropagation CNN training as the EM algorithm: BP's forward pass corresponds to the EM algorithm's E-step. Its backward pass corresponds to the M-step. The hidden neurons and parameters correspond to EM's latent variables. The input image is a hand-drawn digit "3" from the MNIST data set. A forward pass through the CNN computes the activations of the hidden and output neurons. The error between the output activation vector $\mathbf{a}^t$ and the true target vector (0, 0, 1) propagates back through the CNN to compute the gradient of the cross entropy for softmax output neurons. Then gradient descent updates the weights of the CNN.

The second reason is that noise-boosting *enhances sampling* from big-data data sets. A natural way to deal with ever bigger data sets is to randomly sample from them. Such sampling throws away or ignores some or even much of the data. But sufficiently large sample sizes can give adequate statistical precision in many cases. The laws of large numbers ensure this when using sample means and variances or covariances based on population data with finite moments. This opens the door to an array of Monte Carlo sampling techniques. Some big-data "sketching" algorithms already use some form of sampling (Slavakis, Giannakis, & Mateos, 2014).

The NCNN algorithm allows the user to take a smaller random sample than in the noiseless case and still achieve a given level of performance. It also allows the user to take the same number of samples for a better level of performance. Fig. 9 shows that NCNN training with only 700 random image samples had on average the same squared error that noiseless BP training of a CNN had with 1000 such random samples. This 300-sample noise benefit decreased to 100 samples as the noiseless training approached 2000 random image samples.

We tested the NCNN algorithm on standard MNIST test images for image recognition. The test images were handwritten digits from zero to nine. We found a substantial reduction in training time when compared with ordinary or noiseless backpropagation: NCNN reduced the average per-iteration training-set cross entropy by 39%.

These simulations achieved this noise boost by adding noise only to the output neurons. The general NCNN algorithm in Section 5 allows the user to add noise to *any* of the neurons in the multilayered network. Adding noise to hidden or throughput neurons entails only slightly increased cost in terms of using a new scaling matrix. But it further speeds up BP training (Audhkhasi, Osoba, & Kosko, in review).

Fig. 3 shows the defining NCNN hyperplane that passes through the origin of the network's output-layer noise space. The hyperplane structure implies that the NCNN hyperplane imposes only a simple linear condition on the noise. The three dimensions of the noise space in this example correspond to the three output neurons in Fig. 2. Adding uniform noise to the output neurons defines the uniform noise cube. Adding Gaussian noise defines a Gaussian noise ball.

Noise vectors above the NCNN hyperplane speed BP training convergence on average because just these noise samples increase the iterative likelihood steps in the corresponding EM algorithm. Noise below the NCNN hyperplane slows BP convergence on average because it decreases the EM's likelihood steps compared with the noiseless case.

The NCNN noise benefit gradually shrinks as the sample size increases. So the noise-benefit boxes or balls will shrink as the noise boost becomes fainter. The NCNN noise acts as a type of synthetic but statistically representative data. Its helpful effect wanes as the system processes ever more actual data. So its greatest benefit to big data may be in helping the user pick smaller but more representative sample sets to keep.

Fig. 4 shows the training-set cross entropy of a CNN that uses standard noiseless BP, BP with blind noise (Blind-BP), and BP with noisy-EM noise (NEM-BP). Blind-BP ignores the NEM sufficient condition. It simply adds all randomly drawn noise samples to the training data. NCNN or Noisy BP reduced the average training-set cross entropy by 39.26% compared with noiseless BP.

Fig. 5 plots the training-set classification error rates as the system trained. The testing-set classification error rate was nearly the same at convergence. NEM-BP gave a 47.43% reduction in training-set error rate averaged over the first 15 iterations compared with noiseless BP. Adding blind noise only slightly improved cross entropy and classification accuracy.

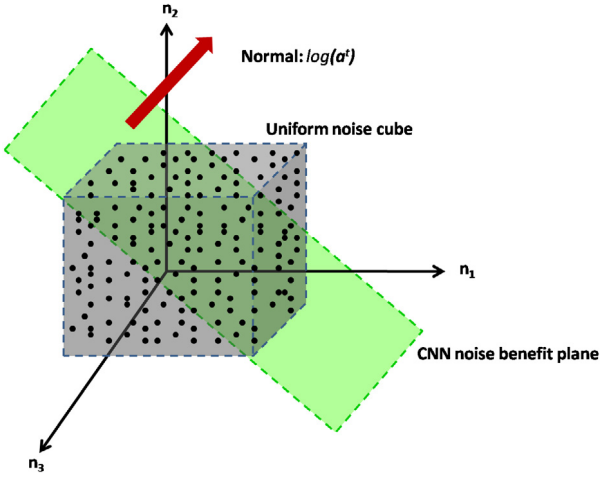

Fig. 3. Noise-benefit region for a CNN with softmax output neurons: Noise speeds up the maximum-likelihood parameter estimation of the CNN with softmax output neurons if the injected noise lies above the NCNN hyperplane. The hyperplane passes through the origin of the noise space. Independent and identically distributed uniform noise lies inside the noise cube. The output layer's activation vector $\mathbf{a}^t$ controls the normal to the NCNN hyperplane. The hyperplane changes as learning proceeds because the parameters and hidden-layer neuron activations change. Adding noise from below the hyperplane slows convergence on average.

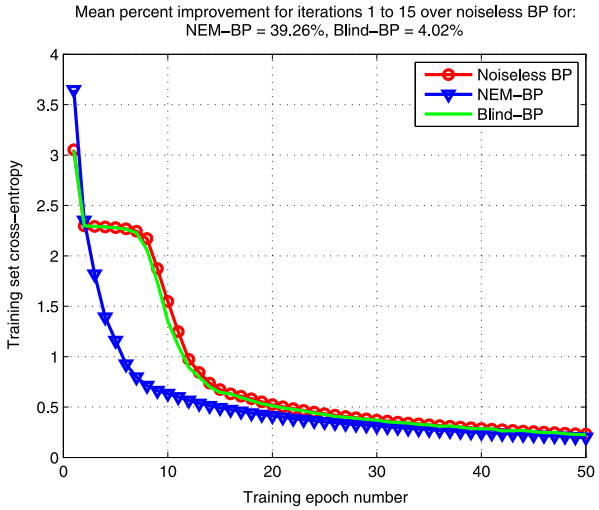

Fig. 4. NEM noise-benefit in BP training of a CNN using MNIST data: NEM-BP training reduced the average training-set cross entropy of the MNIST data set compared with standard noiseless BP training. The NEM-BP algorithm reduced the cross entropy by 39.26% on average compared with standard BP over the first 15 training iterations. Adding blind noise gave only a minor average reduction of 4.02% in cross entropy. Training used 1000 images from the MNIST data for a CNN with one convolution hidden layer. The convolutional layer used three 3×3 masks or filters. Factor-2 downsampling followed the convolutional layer by removing all even index rows and columns of the hidden neuron images. The hidden layer fully connected to the 10 output neurons that predicted the class label of the input digit image. We used uniform noise over $[-0.5/\sqrt{t^5}, 0.5/\sqrt{t^5}]$ where t was the training iteration number for both NEM noise and blind noise.

Fig. 6 shows a noise-benefit inverted U-curve for NCNN training of a CNN on the MNIST data set. This inverted U-curve is the signature of a classic nonlinear noise benefit or so-called *stochastic resonance* (Bulsara & Gammaitoni, 1996; Franzke & Kosko, 2011; Gammaitoni, 1995; Kosko, 2006; Mitaim & Kosko, 2014; Patel & Kosko, 2007, 2008, 2009, 2010, 2011; Wilde & Kosko, 2009). The optimal uniform noise scale occurred at 1. A NEM noise swamping effect occurred where the noise hurt or slowed CNN training when the noise scale increased beyond 2.6. This swamping effect

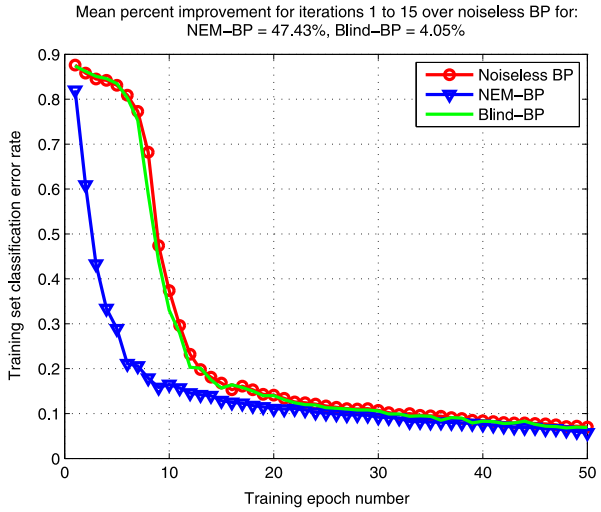

Fig. 5. NEM noise benefit in BP training of a CNN using MNIST data: NEM-BP training reduced the training-set classification error rate of the MNIST data set compared with noiseless BP training. NEM-BP reduced the classification error by 47.43% on average compared with standard BP over the first 15 training iterations. Adding blind noise gave only a minor average reduction of 4.05% in classification error rate. Training used 1000 images from the MNIST data for a CNN with one convolution hidden layer. The convolutional layer used three 3×3 masks or filters. Factor-2 downsampling followed the convolutional layer by removing all even index rows and columns of the hidden neuron images. The hidden layer fully connected to the 10 output neurons that predicted the class label of the input digit. We used uniform noise over $[-0.5/\sqrt{t^5}, 0.5/\sqrt{t^5}]$ where t was the training iteration number for both NEM noise and blind noise.

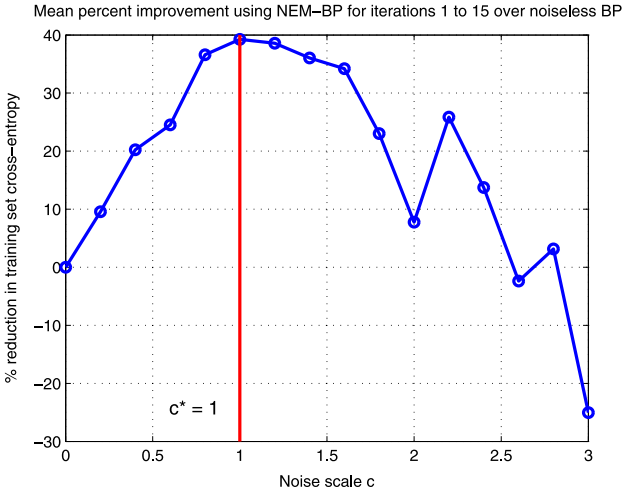

Fig. 6. NEM noise-benefit inverted U-curve for NEM-BP training of a CNN: The figure shows the mean percent reduction in per-iteration training-set cross entropy for NEM-BP training of a CNN with different uniform noise variances. The inverted-U is the signature of a stochastic-resonance noise benefit. We added zero mean uniform $(-0.5\sqrt{c/t^d}, 0.5\sqrt{c/t^d})$ noise where $c = 0, 0.2, \ldots, 2.8, 3$, t was the training epoch, and $d = 5$ was the noise annealing factor. The noise benefit increased when c increased from 0 to 1 and tended to decrease for values greater than 1. The optimal noise scale was $c^* = 1$. Injecting NEM noise hurt the training-set cross entropy when $c \geqslant 2.6$.

appears to arise from the iterative fixed-point structure of the EM algorithm.

The next section presents the working details of a CNN for image recognition. Section 3 presents the BP algorithm for CNN training. Theorem 1 shows that the BP algorithm is a special case of the generalized EM (GEM) algorithm. Section 4 reviews the NEM algorithm for speeding maximum likelihood estimation in the case of missing data or variables. Section 5 presents the NEM-BP algorithm for CNN training. Section 6 summarizes the simulations on the MNIST data set.

2. Convolutional neural networks

A CNN convolves the input data with a set of filters. This operation is a rough analogy to the use of receptive fields in the retina (Hubel & Wiesel, 1959) as in Fukushima's original neocognitron network (Fukushima, 1988).

Consider the CNN in Fig. 1 with one hidden convolutional layer for simplicity. The notation extends directly to allow deep or multiple hidden layers. Let $\mathbf{X}$ denote the input 2-dimensional data of size $M_X \times N_X$ where M_X and N_X are positive integers. The 2D filters $\mathbf{W}_1, \ldots, \mathbf{W}_J$ are each of size $M_W \times N_W$. Then the convolution of $\mathbf{X}$ with the filter $\mathbf{W}_j$ gives the matrix

$$\mathbf{C}_j = \mathbf{X} * \mathbf{W}_j \tag{1}$$

where $*$ denotes 2D convolution.

The 2D data matrix $\mathbf{C}_j$ has size $(M_X + M_W - 1) \times (N_X + N_Y - 1)$ with (m, n)th entry

$$\mathbf{C}_j(m, n) = \sum_{a=1}^{M_W} \sum_{b=1}^{N_W} \mathbf{X}(a - m, b - n)\mathbf{W}_j(a, b). \tag{2}$$

Pad $\mathbf{X}$ with zeros to define it at all points in the above double sum. Then pass the J matrices $\mathbf{C}_1, \ldots, \mathbf{C}_J$ element-wise through logistic sigmoid functions s to give the hidden-neuron activations $\mathbf{Z}_j$:

$$\mathbf{Z}_j(m, n) = s(\mathbf{C}_j(m, n)) \tag{3}$$

$$= \frac{1}{1 + \exp(-\mathbf{C}_j(m, n))}. \tag{4}$$

Suppose the network has K output neurons. A $(M_X + M_W - 1) \times (N_X + N_Y - 1)$ weight matrix $\mathbf{U}_j^k$ multiplies the jth hidden neuron matrix $\mathbf{Z}_j$ element-wise. The softmax or Gibbs activation a_k^t of the kth output neuron is the ratio

$$a_k^t = \frac{\exp\left(\sum_{j=1}^{J} \mathbf{e}^T \mathbf{Z}_j \odot \mathbf{U}_j^k \mathbf{e}\right)}{\sum_{k_1=1}^{K} \exp\left(\sum_{j=1}^{J} \mathbf{e}^T \mathbf{Z}_j \odot \mathbf{U}_j^{k_1} \mathbf{e}\right)} \tag{5}$$

where $\odot$ denotes the element-wise Hadamard product between two matrices. $\mathbf{e}$ is a vector of all 1s of length $(M_X + M_W - 1)(N_X + N_W - 1)$. The JK matrices $\mathbf{U}_j^k$ ($j = 1, \ldots, J$ and $k = 1, \ldots, K$) are the weights of the connections between the hidden neurons and the output neurons. The next section presents the BP and EM algorithms for training a CNN and then proves their equivalence.

3. Backpropagation and EM for training CNNs

The BP algorithm performs gradient ascent on a scalar performance measure. We will use output cross entropy as the performance measure because of its excellent performance in simulations and because the CNN uses softmax output neurons. We could also use squared error as the performance measure. Such a least-squares approach would correspond in the maximum-likelihood (ML) framework to using output neurons that are conditionally Gaussian because the gradient applies to the log-likelihood $\ln p(y|\mathbf{x}, \Theta)$ for input $\mathbf{x}$ and all network parameters Θ (Bishop, 2006). The conditional probability density function (pdf) $p(y|\mathbf{x}, \Theta)$ is the probability of observing the pattern y at the output layer given the input $\mathbf{x}$ of a CNN with parameters Θ.

The BP algorithm performs ML estimation of the J convolution matrices $\mathbf{W}_1, \ldots, \mathbf{W}_J$ and the JK hidden-output weight matrices $\mathbf{U}_j^k$. Let $\mathbf{y}$ denote the 1-in-K binary encoding vector of the target

label for a given input image $\mathbf{X}$. This means that $y_k = 1$ when k corresponds to the correct class and 0 otherwise.

BP computes the *cross entropy* $E(\Theta)$ between the softmax activations $a_1^t, \ldots, a_K^t$ of the output neurons and the binary target vector $\mathbf{y}$:

$$E(\Theta) = -\sum_{k=1}^{K} y_k \ln(a_k^t) \tag{6}$$

where again Θ denotes all parameters of the CNN—the J convolution matrices $\mathbf{W}_1, \ldots, \mathbf{W}_J$ and the weight matrix $\mathbf{U}$.

This BP computation holds because the K output neurons are independent of one another. There are no between-neuron connections at the output layer. So the output likelihood pdf $p(y|\mathbf{x}, \Theta)$ factors. The kth factor $p(y_k|\mathbf{x}, \Theta)$ is just the powered softmax activation $(a_k^t)^{y_k}$ because $y_k = 1$ if k is the correct class label for the input image and $y_k = 0$ otherwise. Then the network's likelihood function $L(\Theta)$ equals the negative cross entropy $-E(\Theta)$:

$$\begin{aligned} L(\Theta) &= \ln p(y|\mathbf{x}, \Theta) && (7)\\ &= \ln \prod_{k=1}^{K} p(y_k|\mathbf{x}, \Theta) && (8)\\ &= \ln \prod_{k=1}^{K} (a_k^t)^{y_k} && (9)\\ &= \sum_{k=1}^{K} y_k \ln(a_k^t) && (10)\\ &= -E(\Theta) && (11) \end{aligned}$$

because again the exponents y_k are binary and sum to one. So BP computes the cross entropy because $p(y|\mathbf{x}, \Theta) = \exp(-E(\Theta))$.

This derivation shows more specifically that

$$L(\Theta) = \ln a_k^t = -E(\Theta) \tag{12}$$

if k is the correct class label for the given input image or pattern. So minimizing the cross entropy $E(\Theta)$ maximizes the likelihood $L(\Theta)$ and vice versa. So the ML estimate Θ_* of Θ is

$$\Theta_* = \arg\max_{\Theta} L(\Theta). \tag{13}$$

The above derivation also shows that the supervised neural network computes a categorical pdf:

$$p(y|\mathbf{x}, \Theta) = \prod_{k=1}^{K} (a_k^t)^{y_k} \tag{14}$$

for target unit-bit-vector y. A categorical pdf is just a one-sample multinomial pdf. The BP network performs multinomial regression in this sense.

We show next that minimizing the cross entropy $E(\Theta)$ is the same as minimizing the Kullback–Leibler divergence (Kullback & Leibler, 1951) between the output softmax activations and the target vector. This holds because the discrete Kullback–Liebler divergence $KL(\mathbf{y} \parallel \mathbf{a}^t)$ expands as

$$\begin{aligned} KL(\mathbf{y} \parallel \mathbf{a}^t) &= \sum_{k=1}^{K} y_k \ln\left(\frac{y_k}{a_k^t}\right)\\ &= \sum_{k=1}^{K} y_k \ln y_k - \sum_{k=1}^{K} y_k \ln a_k^t\\ &= -H(\mathbf{y}) + E(\Theta) && (15) \end{aligned}$$

where $E(\Theta)$ is the cross entropy in (6) and $H(\mathbf{y})$ is the entropy of the target $\mathbf{y}$. But the entropy of the target does *not* depend on the CNN parameters Θ. So minimizing the Kullback–Liebler divergence or the cross-entropy gives the same extremal estimate Θ_* of the CNN parameters. So minimizing the Kullback–Liebler divergence also maximizes the network likelihood $L(\Theta) = \ln p(y|\mathbf{x}, \Theta)$.

We can summarize the above discussion as follows: BP performs gradient ascent on the log-likelihood surface $L(\Theta)$ to iteratively find the ML estimate Θ_* of Θ. This also holds when minimizing squared-error because again BP is then equivalent to ML estimation with a conditional Gaussian pdf (Audhkhasi et al., 2013a; Bishop, 2006). The estimate of Θ at the $(n+1)$th iteration or training epoch is $\Theta^{(n+1)}$:

$$\Theta^{(n+1)} = \Theta^{(n)} - \eta \nabla_{\Theta} E(\Theta)\Big|_{\Theta=\Theta^{(n)}} \tag{16}$$

where η is a positive learning rate. The above argument shows that we could equivalently write this learning law in terms of taking the gradient of the Kullback–Liebler divergence $KL(\mathbf{y} \parallel \mathbf{a}^t)$ or the log-likelihood $\ln p(y|\mathbf{x}, \Theta)$.

A forward pass in BP computes the activations of all hidden and output neurons in the CNN. Back-propagating the output neuron activation errors through the network gives the gradient of the data log-likelihood function with respect to the CNN parameters. The gradient ascent in (16) updates these parameters.

The hidden neuron activations in a CNN are "latent" or unseen variables for the purposes of the EM algorithm. BP here performs ML estimation of a CNN's parameters.

The EM algorithm itself is an iterative method for ML estimation in the case of missing data or latent variables Z (Dempster et al., 1977). The algorithm iteratively maximizes the likelihood pdf $p(y|\mathbf{x}, \Theta)$ by exploiting the pdf identity $p(y|\mathbf{x}, \Theta) = \frac{p(\mathbf{z}, y|\mathbf{x}, \Theta)}{p(\mathbf{z}|y, \mathbf{x}, \Theta)}$. Taking logarithms gives the crucial log-likelihood equation at the heart of both EM and Theorem 1:

$$\ln p(y|\mathbf{x}, \Theta) = \ln p(\mathbf{z}, y|\mathbf{x}, \Theta) - \ln p(\mathbf{z}|y, \mathbf{x}, \Theta). \tag{17}$$

The key idea is that the algorithm conditions on the pdf $p(\mathbf{z}|y, \mathbf{x}, \Theta^{(n)})$ to estimate the missing variable $\mathbf{z}$. So it estimates $\mathbf{z}$ both with the observed data y and $\mathbf{x}$ and with the current parameter estimate $\Theta^{(n)}$.

We can now state the EM algorithm in terms of tuning the CNN parameters. The EM algorithm uses the lower-bound surrogate likelihood function $Q(\Theta|\Theta^{(n)})$ of the log-likelihood function $L(\Theta)$:

$$Q(\Theta|\Theta^{(n)}) = \mathbb{E}_{p(\mathbf{Z}_1, \ldots, \mathbf{Z}_J|\mathbf{y}, \mathbf{X}, \Theta^{(n)})}\{\ln p(\mathbf{Z}_1, \cdots, \mathbf{Z}_J, \mathbf{y}|\mathbf{X}, \Theta)\}. \tag{18}$$

So the J matrices $\mathbf{Z}_1, \ldots, \mathbf{Z}_J$ are the latent variables in the algorithm's expectation (E) step.

Note that $Q(\Theta|\Theta^{(n)})$ is just the expectation with respect to the pdf $p(\mathbf{Z}_1, \ldots, \mathbf{Z}_J|\mathbf{y}, \mathbf{X}, \Theta^{(n)})$ of the first term on the right side of (17). This expectation does not affect the log-likelihood term $\ln p(\mathbf{y}|\mathbf{X}, \Theta^{(n)})$ on the left side of the log-likelihood Eq. (17) since that term does not depend on the latent variables $\mathbf{Z_k}$. EM's hill-climbing or "ascent property" (Dempster et al., 1977) states that any Θ that increases $Q(\Theta|\Theta^{(n)})$ can only increase the log-likelihood difference $\ln p(\mathbf{y}|\mathbf{X}, \Theta) - \ln p(\mathbf{y}|\mathbf{X}, \Theta^{(n)})$. So EM can never decrease the likelihood $p(\mathbf{y}|\mathbf{X}, \Theta^{(n)})$ at an iteration n.

EM's maximization (M) step maximizes the Q-function to find the next parameter estimate $\Theta^{(n+1)}$:

$$\Theta^{(n+1)} = \arg\max_{\Theta} Q(\Theta|\Theta^{(n)}). \tag{19}$$

The *generalized* EM (GEM) algorithm merely increases $Q(\Theta|\Theta^{(n)})$ at each iteration. GEM need not always maximize it. GEM performs this partial optimization by stochastic gradient ascent:

$$\Theta^{(n+1)} = \Theta^{(n)} + \eta \nabla_{\Theta} Q(\Theta|\Theta^{(n)})\Big|_{\Theta=\Theta^{(n)}} \tag{20}$$

where again η is a positive learning coefficient or sequence of such coefficients.

We can now state Theorem 1 from Audhkhasi et al. (2013a). This fundamental theorem shows that BP is a special case of the GEM algorithm because their gradient updates coincide at each iteration n. We restate this theorem and give it a more full proof here for completeness since noise-boosting CNNs depends on it.

Theorem 1 (*Backpropagation is a Special Case of the GEM Algorithm*)**.** *The backpropagation update equation for a differentiable likelihood function $p(y|\mathbf{x}, \Theta)$ at epoch n*

$$\Theta^{(n+1)} = \Theta^{(n)} + \eta \nabla_\Theta \ln p(y|\mathbf{x}, \Theta)\Big|_{\Theta=\Theta^{(n)}} \tag{21}$$

equals the GEM update equation at epoch n

$$\Theta^{(n+1)} = \Theta^{(n)} + \eta \nabla_\Theta Q(\Theta|\Theta^{(n)})\Big|_{\Theta=\Theta^{(n)}} \tag{22}$$

where GEM uses the differentiable Q-function $Q(\Theta|\Theta^{(n)})$ in (18).

Proof. Rewrite the difference of log-likelihoods (17) as

$$\ln p(y|\mathbf{x}, \Theta) = \ln p(\mathbf{h}, y|\mathbf{x}, \Theta) - \ln p(\mathbf{h}|\mathbf{x}, y, \Theta) \tag{23}$$

so that now $\mathbf{h}$ denotes the hidden or latent variables. Then take expectations on both sides by integrating against the pdf $p(\mathbf{h}|\mathbf{x}, y, \Theta^n)$:

$$\ln p(y|\mathbf{x}, \Theta) = \int \ln p(\mathbf{h}, y|\mathbf{x}, \Theta)\, dp(\mathbf{h}|\mathbf{x}, y, \Theta^{(n)}) - \int \ln p(\mathbf{h}|\mathbf{x}, y, \Theta)\, dp(\mathbf{h}|\mathbf{x}, y, \Theta^{(n)}) \tag{24}$$

because the log-likelihood on the left does not depend on $\mathbf{h}$. This gives the equality (Bishop, 2006; Oakes, 1999)

$$\ln p(y|\mathbf{x}, \Theta) = Q(\Theta|\Theta^{(n)}) + H(\Theta|\Theta^{(n)}) \tag{25}$$

from (18). The term $H(\Theta|\Theta^{(n)})$ is the continuous cross entropy (Cover & Thomas, 2012):

$$H(\Theta|\Theta^n) = -\int \ln p(\mathbf{h}|\mathbf{x}, y, \Theta)\, dp(\mathbf{h}|\mathbf{x}, y, \Theta^{(n)}). \tag{26}$$

The result now follows by taking gradients if we can show that $\nabla_\Theta H(\Theta|\Theta^{(n)}) = 0$ at $\Theta = \Theta^{(n)}$.

Expand the continuous Kullback–Leibler divergence (Kullback & Leibler, 1951) of the pdfs $p(\mathbf{h}|\mathbf{x}, y, \Theta^{(n)})$ and $p(\mathbf{h}|\mathbf{x}, y, \Theta)$:

$$D_{\mathrm{KL}}(\Theta^{(n)} \parallel \Theta) = \int \ln\left(\frac{p(\mathbf{h}|\mathbf{x}, y, \Theta^{(n)})}{p(\mathbf{h}|\mathbf{x}, y, \Theta)}\right) dp(\mathbf{h}|\mathbf{x}, y, \Theta^{(n)}) \tag{27}$$

$$= \int \ln p(\mathbf{h}|\mathbf{x}, y, \Theta^{(n)})\, dp(\mathbf{h}|\mathbf{x}, y, \Theta^{(n)}) - \int \ln p(\mathbf{h}|\mathbf{x}, y, \Theta)\, dp(\mathbf{h}|\mathbf{x}, y, \Theta^{(n)}) \tag{28}$$

$$= -H(\Theta^{(n)}|\Theta^{(n)}) + H(\Theta|\Theta^{(n)}). \tag{29}$$

Jensen's inequality and the concavity of the logarithm imply that $D_{\mathrm{KL}}(\Theta^{(n)} \parallel \Theta) \geqslant 0$ (Kullback & Leibler, 1951). So $H(\Theta|\Theta^{(n)}) \geqslant H(\Theta^{(n)}|\Theta^{(n)})$ for all Θ. So $\Theta^{(n)}$ *minimizes* $H(\Theta|\Theta^{(n)})$. So $\nabla_\Theta H(\Theta|\Theta^{(n)}) = 0$ at $\Theta = \Theta^{(n)}$. Take gradients on both sides of (25) and apply this null-gradient result. This gives the fundamental gradient equality

$$\nabla_\Theta \log p(y|\mathbf{x}, \Theta)\Big|_{\Theta=\Theta^{(n)}} = \nabla_\Theta Q(\Theta|\Theta^{(n)})\Big|_{\Theta=\Theta^{(n)}}. \tag{30}$$

So the backpropagation and GEM update equations are identical at each iteration or epoch n. ■

This BP–EM equivalency theorem lets us use the noisy EM algorithm to speed up the BP training of a CNN. The next section details the noisy EM algorithm.

4. Noisy Expectation–Maximization (NEM) algorithm

The Noisy Expectation–Maximization (NEM) algorithm (Osoba & Kosko, 2016; Osoba et al., 2011, 2013) speeds up the EM algorithm on average. The NEM algorithm adds noise to the data at each EM iteration. But the noise must satisfy an average positivity condition. Then such noise can only increase the EM algorithm's "ascent property" (Dempster et al., 1977) on average as it climbs a local hill of probability or log-likelihood.

The injected noise decays with the iteration count to ensure convergence to the optimal parameters of the original data model. The additive noise must also satisfy the NEM positivity condition below. The condition ensures that the NEM parameter estimates will climb faster up the likelihood surface on average.

4.1. NEM theorem for additive noise injection

The NEM Theorem (Osoba et al., 2011, 2013) states when additive noise speeds up the EM algorithm's average convergence to a local optimum of the likelihood surface. This sufficient condition for a noise boost is a positivity (non-negativity) condition. This NEM noise-benefit result holds more generally for multiplicative noise or for any other measurable combination of signal and noise (Osoba & Kosko, 2016).

The NEM idea is that noise $\mathbf{n}$ increases the probability of a signal $\mathbf{x}$ when $p(\mathbf{x} + \mathbf{n}|\Theta) \geqslant p(\mathbf{x}|\Theta)$. This pdf inequality is equivalent to the inequality $\ln \frac{p(\mathbf{x}+\mathbf{n}|\Theta)}{p(\mathbf{x}|\Theta)} \geqslant 0$. Then taking averages gives the positivity condition in the NEM Theorem below.

The NEM Theorem uses the noise random variable $\mathbf{N}$ with probability density function (pdf) $p(\mathbf{n}|\mathbf{x})$. So the noise $\mathbf{N}$ depends on the data $\mathbf{x}$ in general. The vector $\mathbf{h}$ denotes the latent or hidden variables in the EM model. The sequence $\{\Theta^{(n)}\}$ is a sequence of EM estimates for Θ. Then Θ_* is the converged EM estimate for Θ: $\Theta_* = \lim_{n\to\infty} \Theta^{(n)}$. Define the *random* noisy Q function $Q_{\mathbf{N}}(\Theta|\Theta^{(n)})$ as the expectation $Q_{\mathbf{N}}(\Theta|\Theta^{(n)}) = \mathbb{E}_{\mathbf{h}|\mathbf{x},\Theta_k}[\ln p(\mathbf{x} + \mathbf{N}, \mathbf{h}|\theta)]$. We assume that all random variables have a finite differential entropy and that the additive noise keeps the data in the support of the likelihood function. Then we can state the general NEM theorem for additive noise injection (Osoba et al., 2011, 2013).

Theorem 2 (*Noisy Expectation–Maximization (NEM)*)**.** *The EM estimation noise benefit*

$$Q(\Theta_*|\Theta_*) - Q(\Theta^{(n)}|\Theta_*) \geqslant Q(\Theta_*|\Theta_*) - Q_N(\Theta^{(n)}|\Theta_*) \tag{31}$$

or equivalently

$$Q_N(\Theta^{(n)}|\Theta_*) \geqslant Q(\Theta^{(n)}|\Theta_*) \tag{32}$$

holds on average if the following positivity *condition holds:*

$$\mathbb{E}_{\mathbf{x},\mathbf{h},\mathbf{N}|\Theta_*}\left[\ln\left(\frac{p(\mathbf{x}+\mathbf{N}, \mathbf{h}|\Theta_n)}{p(\mathbf{x}, \mathbf{h}|\Theta_n)}\right)\right] \geqslant 0. \tag{33}$$

Reversing the inequalities in the NEM Theorem gives a dual theorem for noise harm on average (Osoba & Kosko, 2016). Injecting noise from below the hyperplane in Fig. 3 only slows convergence on average because it takes smaller steps up the likelihood surface compared with noiseless EM. It thus reduces the "ascent property" of ordinary EM (Dempster et al., 1977). We state this result as Corollary 1.

Corollary 1 (*Noise Harm in Expectation–Maximization*)**.** *The EM estimation noise harm*

$$Q(\Theta_*|\Theta_*) - Q(\Theta^{(n)}|\Theta_*) \leqslant Q(\Theta_*|\Theta_*) - Q_N(\Theta^{(n)}|\Theta_*) \tag{34}$$

or equivalently

$$Q_N(\Theta^{(n)}|\Theta_*) \leqslant Q(\Theta^{(n)}|\Theta_*) \tag{35}$$

holds on average if the following negativity *condition holds:*

$$\mathbb{E}_{\mathbf{x},\mathbf{h},\mathbf{N}|\Theta_*}\left[\ln\left(\frac{p(\mathbf{x}+\mathbf{N},\mathbf{h}|\Theta_n)}{p(\mathbf{x},\mathbf{h}|\Theta_n)}\right)\right] \leqslant 0. \tag{36}$$

The NEM Theorem states that each iteration of a properly noisy EM algorithm gives higher likelihood estimates on average than do the noiseless EM estimates. So the NEM algorithm converges faster on average than does noiseless EM for a given data model. The faster NEM convergence occurs both because the likelihood function has an upper bound and because the NEM algorithm takes larger average steps up the likelihood surface. The largest gains tend to occur in the first few steps.

NEM also speeds up the training of hidden Markov models (Audhkhasi, Osoba, & Kosko, 2013b) and the k-means clustering algorithm (Osoba & Kosko, 2013) used in big-data processing (Jain, 2010). The NEM positivity condition has a much simpler form in the practical case of a Gaussian or Cauchy mixture model because then the NEM positivity condition reduces to a quadratic inequality (Osoba et al., 2013). Exponential noise leads to a still simpler linear NEM condition (Osoba & Kosko, 2016). Related noise injection speeds up Markov chain Monte Carlo simulations and simulated annealing (Franzke & Kosko, 2015). We show next how to apply the NEM additive-noise boost to a CNN.

5. Noisy backpropagation for CNN training

The next theorem states the NEM-based noise-benefit sufficient condition for softmax or Gibbs-activation *output* neurons used in CNN K-class classification. Our simulations add NEM noise only to the 1-in-K encoding vector $\mathbf{y}$ of the target class labels. The end of this section shows how to add NEM noise to the hidden neurons as well. Monte Carlo importance sampling approximated the expectation in the Q-function.

Theorem 3 (*Hyperplane Noise-Benefit Condition for CNNs*)**.** *The NEM positivity condition holds for ML training of a CNN with softmax or Gibbs activation output neurons if*

$$\mathbb{E}_{\mathbf{y},\mathbf{Z}_1,\ldots,\mathbf{Z}_J,\mathbf{n}|\mathbf{X},\Theta_*}\left\{\mathbf{n}^T \log(\mathbf{a}^t)\right\} \geqslant 0 \tag{37}$$

where the activation of the kth output neuron is

$$a_k^t = \frac{\exp\left(\sum_{j=1}^{J} \mathbf{e}^T\mathbf{Z}_j \odot \mathbf{U}_j^k\mathbf{e}\right)}{\sum_{k_1=1}^{K}\exp\left(\sum_{j=1}^{J} \mathbf{e}^T\mathbf{Z}_j \odot \mathbf{U}_j^{k_1}\mathbf{e}\right)} \tag{38}$$

where $\odot$ denotes the element-wise Hadamard product between two matrices. $\mathbf{e}$ *is a vector of all* 1*s of length* $(M_X+M_W-1)(N_X+N_W-1)$.

Proof. Add noise $\mathbf{n}$ to the target 1-in-K encoding vector $\mathbf{y}$ at the output neurons. Then the likelihood ratio in the NEM sufficient condition becomes

$$\frac{p(\mathbf{y}+\mathbf{n},\mathbf{Z}_1,\ldots,\mathbf{Z}_J|\mathbf{X},\Theta)}{p(\mathbf{y},\mathbf{Z}_1,\ldots,\mathbf{Z}_J|\mathbf{X},\Theta)} = \frac{p(\mathbf{y}+\mathbf{n}|\mathbf{X},\mathbf{Z}_1,\ldots,\mathbf{Z}_J,\Theta)}{p(\mathbf{y}|\mathbf{X},\mathbf{Z}_1,\ldots,\mathbf{Z}_J,\Theta)}. \tag{39}$$

The output softmax activations $\mathbf{a}_k^t$ from (5) simplify the ratio on the right-hand side of the above equation. Then (14) gives

$$\frac{p(\mathbf{y}+\mathbf{n}|\mathbf{X},\mathbf{Z}_1,\ldots,\mathbf{Z}_J,\Theta)}{p(\mathbf{y}|\mathbf{X},\mathbf{Z}_1,\ldots,\mathbf{Z}_J,\Theta)} = \prod_{k=1}^{K}\frac{(a_k^t)^{t_k+n_k}}{(a_k^t)^{t_k}} \tag{40}$$

$$= \prod_{k=1}^{K}(a_k^t)^{n_k}. \tag{41}$$

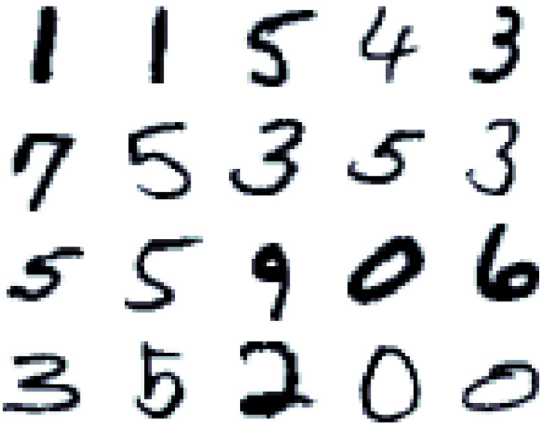

Fig. 7. MNIST Digits: The figure shows 20 sample images from the MNIST data set. Each digit is a 28 × 28 pixel grayscale image.

Put this activation product in the positivity condition of the NEM Theorem to get the inequality

$$\mathbb{E}_{\mathbf{y},\mathbf{Z}_1,\ldots,\mathbf{Z}_J,\mathbf{n}|\mathbf{X},\Theta_*}\left\{\ln\left(\prod_{k=1}^{K}(a_k^t)^{n_k}\right)\right\} \geqslant 0 \tag{42}$$

or

$$\mathbb{E}_{\mathbf{y},\mathbf{Z}_1,\ldots,\mathbf{Z}_J,\mathbf{n}|\mathbf{X},\Theta_*}\left\{\sum_{k=1}^{K} n_k \ln(a_k^t)\right\} \geqslant 0. \tag{43}$$

Then vector notation gives the desired inequality:

$$\mathbb{E}_{\mathbf{y},\mathbf{Z}_1,\ldots,\mathbf{Z}_J,\mathbf{n}|\mathbf{X},\Theta_*}\left\{\mathbf{n}^T \ln(\mathbf{a}^t)\right\} \geqslant 0. \quad \blacksquare \tag{44}$$

Fig. 3 illustrates the sufficient condition in (44) for a CNN with three output neurons. All noise $\mathbf{n}$ above the hyperplane $\{\mathbf{n} : \mathbf{n}^T \log(\mathbf{a}^t) = 0\}$ speeds CNN training on average.

A similar noise-benefit result also holds for additive noise injection into the *hidden* neurons in a CNN. The hidden neuron activations become visible data during the forward pass of neural-network training. They behave as output neurons for earlier layers (Audhkhasi et al., in review). Then the NEM noise-benefit condition becomes the generalized hyperplane inequality

$$(\mathbf{U}^T\mathbf{n})^T \log(\mathbf{a}^t) \geqslant (N^T n)^T \log(\mathbf{1}-a^t) \tag{45}$$

where $\mathbf{U}$ is the synaptic weight matrix that connects the hidden and output layers. But now $\mathbf{a}^t$ is the vector of hidden-layer activations. The special case $\mathbf{U} = \mathbf{I}$ holds in Theorem 3 when $\mathbf{I}$ is the identity matrix that corresponds to the output layer of neurons with output activations $\mathbf{a}^t$.

We also state Corollary 2 as a noise-harm result dual to Corollary 1. It also follows from reversing the inequalities in Theorem 3 and in its proof.

Corollary 2. *The NEM negativity condition holds for ML training of a CNN with Gibbs activation output neurons if*

$$\mathbb{E}_{\mathbf{y},\mathbf{Z}_1,\ldots,\mathbf{Z}_J,\mathbf{n}|\mathbf{X},\Theta_*}\left\{\mathbf{n}^T \log(\mathbf{a}^t)\right\} \leqslant 0 \tag{46}$$

where (38) gives the activation of the kth output neuron.

6. Noise-enhanced CNN simulation results

All simulations used the MNIST data set of handwritten digits. The MNIST data set contains 28 × 28 gray-scale pixel images with pixel intensities between 0 and 1. Fig. 7 shows 20 sample images

Data: T input images $\{\mathbf{X}_1, \ldots, \mathbf{X}_T\}$, T target label 1-in-K vectors $\{\mathbf{y}_1, \ldots, \mathbf{y}_T\}$, number J of convolution masks, size $M_W \times N_W$ of each convolution mask, number of BP epochs R
Result: Trained CNN weight matrices
while *epoch* $r : 1 \rightarrow R$ **do**
  while *training image number* $t : 1 \rightarrow T$ **do**
    • Compute the J hidden activation matrices $\mathbf{Z}_1, \ldots, \mathbf{Z}_J$ using (2) and (3);
    • Downsample the J hidden activation matrices $\mathbf{Z}_1, \ldots, \mathbf{Z}_J$ by a factor of 2.
    • Compute the K-D output softmax activation vector $\mathbf{a}$ using (5);
    • Generate noise vector $\mathbf{n}$;
    if $\mathbf{n}^T \log(\mathbf{a}) \geq 0$ **then**
      • Add NEM noise: $\mathbf{y}_t \leftarrow \mathbf{y}_t + \mathbf{n}$;
    else
      • Do nothing
    end
    • Compute error $\mathbf{y}_t - \mathbf{a}$;
    • Back-propagate error to compute cross entropy gradient $\nabla_\Theta E(\Theta)$;
    • Update network parameters Θ using gradient descent in (16);
  end
end

Algorithm 1: The NEM-BP Algorithm for a CNN: The Noisy Convolutional Neural Network Algorithm. A similar algorithm exists for injecting NEM noise into the hidden layers of a CNN as the paper discusses. A MATLAB implementation of the above algorithm is available at http://sail.usc.edu/~audhkhas/software/NCNN.zip

from this data set. Fig. 1 shows a schematic diagram of the BP training of a CNN using images from the MNIST data set.

The simulations used at least 1000 images from the MNIST training set. We modified an open-source Matlab toolbox (Palm, 2014) to add noise during CNN training. The CNN contained one convolution layer with three 3×3 pixel masks each. We followed the convolution layer with factor-2 down-sampling to increase system robustness and to reduce the number of CNN parameters (LeCun et al., 1998). A full non-convolution connection matrix $\mathbf{U}$ connected the neurons of the hidden layer to the output layer.

The output-layer neurons used the softmax or Gibbs activation function for 10-way classification. All hidden neurons used the logistic sigmoid function. We used uniform noise over $(-0.5\sqrt{c/t^d}, 0.5\sqrt{c/t^d})$ where $c = 0, 0.2, \ldots, 3, d = 1, 2, \ldots, 5$, and t was the training epoch. So the noise variance decreased to 0 as the training epochs proceeded.

Fig. 4 shows the training-set cross entropy of a CNN for three algorithms: standard noiseless BP, BP with blind noise (Blind-BP), and BP with NEM noise (NEM-BP or NCNN). NEM-BP reduced the training-set cross entropy by 39.26% on average over the first 15 iterations as compared with noiseless BP.

Fig. 5 plots the training-set classification error rates as the CNN learned. NEM-BP reduced the training-set error by 47.43% averaged over the first 15 iterations as compared with noiseless BP. This significant reduction in cross-entropy and training-set error occurred because NEM-BP took bigger steps on average towards the maximum likelihood CNN parameters. Adding blind noise (Blind-BP) gave only a comparatively minor improvement of 4.05%.

We next plotted the relative average reduction in cross entropy for NEM-BP as the noise scale c varied from 0 to 3 in steps of 0.2. Fig. 6 shows the resulting characteristic noise-benefit inverted U-curve of stochastic resonance. The optimal uniform noise scale occurred at $c^* = 1$ and NEM-BP gave a 39.26% improvement in average cross entropy. NEM noise hurt CNN training when the noise scale increased beyond 2.6. A very large noise variance hurt convergence because EM is a fixed-point algorithm. So too much noise tends to shadow or swamp the clean data. The noise benefit decreased to zero as the noise variance decreased because then the NEM algorithm became the standard EM algorithm.

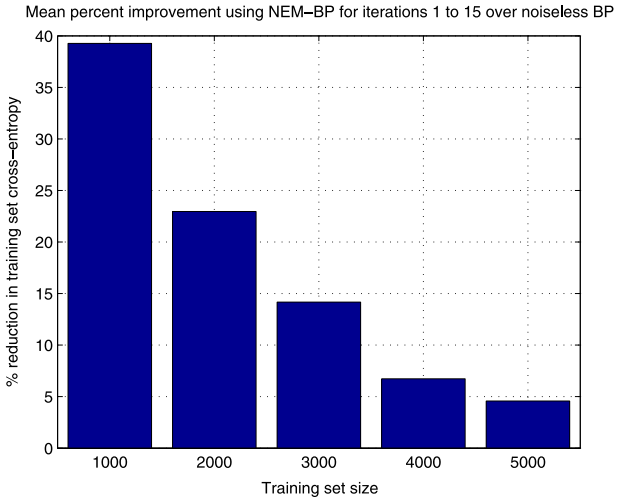

Fig. 8. The NCNN noise boost decreased as the training-set sample size increased: The bar chart shows the relative average reduction in training-set cross entropy for NEM-BP as the training-set size increased. The noise benefit was largest for smaller training-data set sizes.

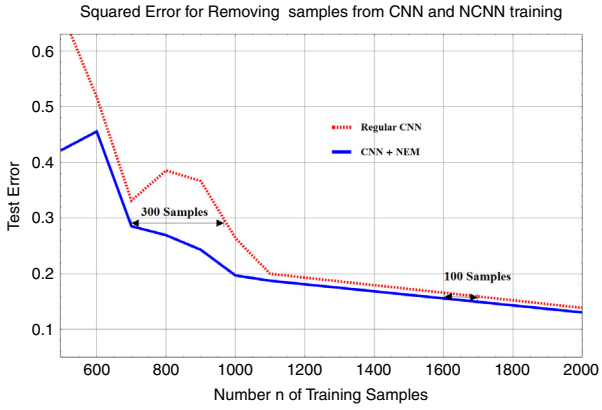

Fig. 9. Random-sampling noise boost: The two curves show the relative noise-benefit effects for different CNN training-set sample sizes using the NMIST test images. Noise training improved the performance of the CNN at any given training sample size. NCNN training with only 700 random image samples had on average the same squared error that noiseless BP training of a CNN had with 1000 random image samples. This 300-sample noise benefit decreased to 100 samples as the noiseless training approached 2000 random image samples. The dashed line shows the average test-set squared error for the CNN at different training set sizes. The solid line shows the average test-set squared error for the NCNN at different training set sizes. Each plotted error value averaged 20 error measurements. (Please see the web version of this article for interpretation of the references to color in this figure legend.)

We also explored how the training-data set size affected NEM performance. We varied the MNIST training-set size over 1000, 2000, ..., 5000 and computed the relative average reduction in training cross entropy for NEM-BP using the optimal noise variance. Fig. 8 shows the resulting decreasing bar chart: The NEM-BP noise boost fell as the number of training data samples increased. This shows that NEM-BP is especially useful when the number of training data samples is small relative to the number of estimated CNN parameters.

We also simulated how the NCNN algorithm favored *subset sampling* with CNN image recognition. Fig. 9 summarizes the results: BP training of CNN with 1000 randomly sampled test

images corresponds in squared error to NCNN training with only 700 samples. So the noise benefit was roughly 300 samples. This noise benefit fell to 100 samples as the noiseless samples approached 2000.

The simulations first trained the CNN on a random selection of 1000 MNIST sample images from the full 60 000 sample training set. We ran 20 separate training runs at the same sample size and recorded the final squared error on the test set for each run. The next step repeated the same simulation setup but with 5% fewer samples for training. We repeated the experiment by reducing the training set by 5% on each simulation epoch.

The simulation ended with the 500-sample training-set case. The dashed red curve in Fig. 9 shows the average test-set squared error at each training sample size. Each point averaged 20 test-set squared-error values. The solid blue curve in the figure arose from a similar experiment that used NEM noise in the CNN training procedure and thus that ran the NCNN algorithm.

7. Conclusions

Careful noise injection speeds up the backpropagation training of a convolutional neural network (CNN). This result follows because the BP algorithm is a special case of the generalized EM algorithm and because the recent noisy EM theorem gives a sufficient condition for noise to speed up the average convergence of the EM algorithm. The Noisy CNN (NCNN) algorithm uses this noisy-EM result to produce a hyperplane in noise space that separates helpful noise from harmful noise. NCNN noise-injection experiments on the MNIST image data set show substantial reduction in training-set cross entropy and in classification error rate as compared with the noiseless BP algorithm. Blind noise gave only a small noise benefit. Simulations showed that the NEM noise benefit was largest for smaller data sets. This suggests exploiting these noise benefits in random sampling from large data sets. Future work should also explore noise injection in different combinations of hidden layers in deep networks.

References

Audhkhasi, K., Osoba, O., & Kosko, B. (2013a). Noise benefits in backpropagation and deep bidirectional pre-training. In *The international joint conference on neural networks, IJCNN* (pp. 1–8). IEEE.

Audhkhasi, K., Osoba, O., & Kosko, B. (2013b). Noisy hidden Markov models for speech recognition. In *Proc. IJCNN* (pp. 1–8).

Audhkhasi, K., Osoba, O., & Kosko, B. (2016). Noise can speed backpropagation learning and deep bidirectional pre-training (in review).

Bishop, C. M. (2006). *Pattern recognition and machine learning. Vol. 1*. Springer.

Bulsara, A. R., & Gammaitoni, L. (1996). Tuning in to noise. *Physics Today*, 39–45.

Cireşan, D. C., Meier, U., Masci, J., Gambardella, L. M., & Schmidhuber, J. (2011). Flexible, high performance convolutional neural networks for image classification. In *Proceedings of the twenty-second international joint conference on artificial intelligence. Vol. 2* (pp. 1237–1242). AAAI Press.

Cover, Thomas M., & Thomas, Joy A. (2012). *Elements of information theory*. Wiley-Interscience.

Dempster, Arthur P., Laird, Nan M., & Rubin, Donald B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B. Statistical Methodology*, 1–38.

Franzke, B., & Kosko, B. (2011). Noise can speed convergence in Markov chains. *Physical Review E*, *84*(4), 041112.

Franzke, Brandon, & Kosko, Bart (2015). Using noise to speed up Markov chain Monte Carlo estimation. *Procedia Computer Science*, *53*, 113–120.

Fukushima, K. (1988). Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks*, *1*(2), 119–130.

Gammaitoni, L. (1995). Stochastic resonance in multi-threshold systems. *Physics Letters A*, *208*, 315–322.

Hubel, D. H., & Wiesel, T. N. (1959). Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, *148*(3), 574–591.

Jain, A. K. (2010). Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*, *31*(8), 651–666.

Kosko, B. (2006). *Noise*. Viking.

Krizhevsky, A., Sutskever, I., & Hinton, G.E. (2012). Imagenet classification with deep convolutional neural networks. In *NIPS, Vol. 1* (p. 4).

Kullback, Solomon, & Leibler, Richard A. (1951). On information and sufficiency. *The Annals of Mathematical Statistics*, *22*(1), 79–86.

Kung, S. Y. (2014). *Kernel methods and machine learning*. Cambridge University Press.

Lawrence, S., Giles, C. L., Tsoi, A. C., & Back, A. D. (1997). Face recognition: A convolutional neural-network approach. *IEEE Transactions on Neural Networks*, *8*(1), 98–113.

Le, Q., Sarlós, T., & Smola, A. (2013). Fastfood—approximating kernel expansions in loglinear time. In *Proc. ICML*.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*, 436–444.

LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., & Hubbard, W. et al. (1990). Handwritten digit recognition with a back-propagation network. In *Proc. NIPS*.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Matsugu, M., Mori, K., Mitari, Y., & Kaneda, Y. (2003). Subject independent facial expression recognition with robust face detection using a convolutional neural network. *Neural Networks*, *16*(5), 555–559.

Mitaim, Sanya, & Kosko, Bart (2014). Noise-benefit forbidden-interval theorems for threshold signal detectors based on cross correlations. *Physical Review E*, *90*(5), 052124.

Oakes, David (1999). Direct calculation of the information matrix via the EM. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, *61*(2), 479–482.

Osoba, O., & Kosko, B. (2013). Noise-enhanced clustering and competitive learning algorithms. *Neural Networks*,.

Osoba, O., & Kosko, B. (2016). The noisy expectation–maximization algorithm for multiplicative noise injection. *Fluctuation and Noise Letters*,.

Osoba, O., Mitaim, S., & Kosko, B. (2011). Noise benefits in the expectation–maximization algorithm: NEM theorems and models. In *The international joint conference on neural networks (IJCNN)* (pp. 3178–3183). IEEE.

Osoba, O., Mitaim, S., & Kosko, B. (2013). The noisy expectation–maximization algorithm. *Fluctuation and Noise Letters*, *12*(03).

Palm, Rasmus Berg (2014). DeepLearn Toolbox. https://github.com/rasmusbergpalm/DeepLearnToolbox.

Patel, A., & Kosko, B. (2007). Levy noise benefits in neural signal detection. In *IEEE international conference on acoustics, speech and signal processing, 2007. ICASSP 2007, Vol. 3* (pp. III–1413–III–1416).

Patel, A., & Kosko, B. (2008). Stochastic resonance in continuous and spiking neurons with Levy noise. *IEEE Transactions on Neural Networks*, *19*(12), 1993–2008.

Patel, A., & Kosko, B. (2009). Error-probability noise benefits in threshold neural signal detection. *Neural Networks*, *22*(5), 697–706.

Patel, A., & Kosko, B. (2010). Optimal mean-square noise benefits in quantizer-array linear estimation. *IEEE Signal Processing Letters*, *17*(12), 1005–1009.

Patel, A., & Kosko, B. (2011). Noise benefits in quantizer-array correlation detection and watermark decoding. *IEEE Transactions on Signal Processing*, *59*(2), 488–505.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536.

Simard, P., Steinkraus, D., & Platt, J.C. (2003). Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR, Vol. 3* (pp. 958–962).

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. ArXiv Preprint arXiv:1409.1556.

Slavakis, K., Giannakis, G., & Mateos, G. (2014). Modeling and optimization for big data analytics: (statistical) learning tools for our era of data deluge. *IEEE Signal Processing Magazine*, *31*(5), 18–31.

Szarvas, M., Yoshizawa, A., Yamamoto, M., & Ogata, J. (2005). Pedestrian detection with convolutional neural networks. In *Proceedings of the IEEE intelligent vehicles symposium* (pp. 224–229). IEEE.

Wilde, M., & Kosko, B. (2009). Quantum forbidden-interval theorems for stochastic resonance. *Journal of Physics A: Mathematical and Theoretical*, *42*(46), 465309.