Using Noise to Speed Up Video Classification with Recurrent Backpropagation

Olaoluwa Adigun
Department of Electrical Engineering
University of Southern California
Los Angeles, CA 90089.
E-mail: adigun@usc.edu

Bart Kosko
Department of Electrical Engineering
University of Southern California
Los Angeles, CA 90089.
E-mail: kosko@usc.edu

***Abstract*—Carefully injected noise can speed the convergence and accuracy of video classification with recurrent backpropagation (RBP). This noise-boost uses the recent results that backpropagation is a special case of the generalized expectation maximization (EM) algorithm and that careful noise injection can always speed the average convergence of the EM algorithm to a local maximum of the log-likelihood surface. We extend this result to the time-varying case of recurrent backpropagation and prove sufficient noise-benefit conditions for both classification and regression. Injecting noise that satisfies the noisy-EM positivity condition (NEM noise) speeds up RBP training. The classification simulations used eleven categories of sports videos based on standard UCF YouTube sports-action video clips. Training RBP with NEM noise in just the output neurons led to 60% fewer iterations in training as compared with noiseless training. This corresponded to a 20.6% maximum decrease in training cross entropy. NEM noise injection also outperformed simple blind noise injection: RBP training with NEM noise gave a 15.6% maximum decrease in training cross entropy compared with RBP training with blind noise. Injecting NEM noise also improved the relative classification accuracy by 5% over noiseless RBP training. NEM noise improved the classification accuracy from 81% to 83% on the test set.**

I. Noise Boosting Recurrent Backpropagation

We show how carefully injected noise can speed up the convergence of recurrent backpropagation (RBP) for classifying time-varying patterns such as videos. Simulations used the 11 categories of sampled UCF YouTube sports videos [1], [2]. We prove a related result for speeding up RBP for regression.

These new noise-benefit results turn on the recent result that backpropagation (BP) is a special case of the generalized expectation-maximization (EM) algorithm [3]. BP is the benchmark algorithm for training deep neural networks [4]–[6]. EM is the standard algorithm for finding maximum-likelihood parameters in the case of missing data or hidden parameters [7]. BP finds the network weights and parameters that minimize some error or other performance measure. It equivalently maximizes the network likelihood and this permits the EM connection. The error function is usually cross-entropy for classification and squared error for regression.

The Noisy EM (NEM) Theorem [8]–[10] gives a sufficient condition for noise to speed up the average convergence of the EM algorithm. It holds for additive or multiplicative or any other type of noise injection [10]. The NEM Theorem ensures that NEM noise improves training with the EM algorithm by ensuring that at each iteration NEM takes a larger step on average up the nearest hill of probability or log-likelihood. The BP algorithm is a form of maximum likelihood estimation [11] and a special case of generalized EM algorithm. So NEM noise also improves the convergence of BP [3]. It also tends to improve classification accuracy because the likelihood lower-bounds the accuracy [12]. The NEM noise benefit is a type of of *stochastic resonance* where a small amount of noise improves the performance of a nonlinear system while too much noise harms the performance [13]–[21]. Figure 2 shows that noise-boosted RBP took only 8 iterations to reach the same cross-entropy value that noiseless RBP reached after 34 iterations. The noise benefit became more pronounced when we added more internal memory neurons. Figure 3 shows that the benefit reached diminishing returns with about 200 memory neurons. This 200-memory-neuron case produced a 5% improvement in the relative classification accuracy.

A recurrent neural network (RNN) is a neural network whose directed graph contains cycles or feedback. This feedback allows an RNN to sustain an internal memory state that can hold and reveal information about recent inputs. So the RNN output depends not only on the input but on the recent output and on the internal memory state. This internal dynamical structure makes RNNs suitable for time-varying tasks such as speech processing [5], [22] and video recognition [23], [24]. RBP is the most common method for training RNNs [25], [26]. We apply the NEM-based RBP algorithm below to video recognition. The NEM noise-boost applies in principle to any time-varying task of classification or regression.

The next section reviews the recent result that BP is a special case of generalized EM. Then section III presents the new noisy RBP training theorems for classification and for regression with RNNs. We also present the noisy RBP algorithm for training long short-term memory (LSTM) [26] or recurrent learning from time-series data [27]. Section IV presents the results of NEM-noise training of a LSTM RNN for video classification.

II. Backpropagation as Generalized Expectation Maximization

This section states the recent result that backpropagation is a special case of the generalized Expectation-Maximization

(EM) algorithm and that careful noise injection can always speed up EM convergence. The next section states the companion result that carefully chosen noise can speed convergence of the EM on average and thereby noise-boost BP.

The BP-as-EM theorem casts BP in terms of maximum likelihood. Both classification and regression have the same BP (EM) learning laws but they differ in the type of neurons in their output layer. The output neurons of a classifier network are softmax or Gibbs neurons. The output neurons of a regression network are identity neurons. But the regression case assumes that the output training vectors are the vector population means of a multidimensional normal probability density with an identity population covariance matrix.

We now restate the recent theorem that BP is a special case of the generalized EM (GEM) algorithm [3].

Theorem 1: Backpropagation as Generalized Expectation Maximization

The backpropagation update equation for a differentiable likelihood function $p(\mathbf{y}|\mathbf{x}, \mathbf{\Theta})$ at epoch n

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

equals the GEM update equation at epoch n

$$\Theta^{n+1} = \Theta^{n} + \eta \bigtriangledown_{\Theta} Q(\Theta|\Theta^{n})\Big|_{\Theta=\Theta^{n}} \tag{2}$$

where GEM uses the differentiable Q-function

$$Q(\Theta|\Theta^{n}) = \mathbb{E}_{\mathbf{p(h|y,x},\Theta^{n})}\Big\{ln \; p(\mathbf{h}, \mathbf{y}|\mathbf{X}, \Theta)\Big\}.$$

The next section shows how selective noise injection can speed up the average convergence of the EM algorithm.

A. Noise-boosting Backpropagation via EM

The Noisy EM Theorem shows that noise injection can only speed up the EM algorithm on average if the noise obeys the NEM positivity condition [8], [10]. The Noisy EM Theorem for additive noise states that a noise benefit holds at each iteration n if the following positivity condition holds:

$$\mathbb{E}_{\mathbf{x},\mathbf{h},\mathbf{N}|\mathbf{\Theta}^{*}}\Big[\ln\Big(\frac{p(\mathbf{x}+\mathbf{N}, \mathbf{h}|\Theta^{n})}{p(\mathbf{x}, \mathbf{h}|\Theta^{n})}\Big)\Big] \geq 0 \; . \tag{3}$$

Then the EM noise benefit

$$Q(\Theta^{n}|\Theta^{*}) \leq Q_{N}(\Theta^{n}|\Theta^{*}) \tag{4}$$

holds on average at iteration n:

$$\begin{aligned}&\mathbb{E}_{\mathbf{x},\mathbf{N}|\mathbf{\Theta^{n}}}\Big[Q(\Theta^{n}|\Theta^{*}) - Q_{N}(\Theta^{n}|\Theta^{*})\Big] \\ &\qquad \leq \mathbb{E}_{\mathbf{x}|\mathbf{\Theta^{n}}}\Big[Q(\Theta^{*}|\Theta^{*}) - Q(\Theta^{n}|\Theta^{*})\Big]\end{aligned}$$

where Θ^{*} denotes the maximum-likelihood vector of parameters. The NEM positivity condition (3) has a simple form for Gaussian mixture models [28] and for classification and regression networks [3].

The intuition behind the NEM sufficient condition (3) is that some noise realizations n make a signal x more probable: $f(x+n|\Theta) \geq f(x|\Theta)$. Taking logarithms gives $\ln(\frac{f(x+n|\theta)}{f(x|\theta)}) \geq 0$. Taking expectations gives a NEM-like positivity condition. The actual proof of the NEM Theorem uses Kullback-Liebler divergence to show that the noise-boosted likelihood is closer on average at each iteration to the optimal likelihood function than is the noiseless likelihood [10].

The NEM positivity inequality (3) is not vacuous because the expectation conditions on the converged parameter vector Θ^{*}. Vacuity would result in the usual case of averaging a log-likelihood ratio. Take the expectation of the log-likelihood ratio $\ln \frac{f(x|\Theta)}{g(x|\Theta)}$ with respect to the probability density function $g(x|\Theta)$ to give $E_{g}[\ln \frac{f(x|\Theta)}{g(x|\Theta)}]$. Then Jensen's inequality and the concavity of the logarithm give $E_{g}[\ln \frac{f(x|\Theta)}{g(x|\Theta)}] \leq \ln E_{g}[\frac{f(x|\Theta)}{g(x|\Theta)}] = \ln \int_{X} \frac{f(x|\Theta)}{g(x|\Theta)} g(x|\Theta) \; dx = \ln \int_{X} g(x|\Theta) \; dx = \ln 1 = 0$. So $E_{g}[\ln \frac{f(x|\theta)}{g(x|\theta)}] \leq 0$ and thus in this case strict positivity is impossible [10]. But the expectation in (3) does not in general lead to this cancellation of probability densities because the integrating density in (3) depends on the optimal maximum-likelihood parameter Θ^{*} rather than on just Θ^{n}. So density cancellation occurs only when the NEM algorithm has converged to a local likelihood maximum because then $\Theta^{n} = \Theta^{*}$.

The NEM Theorem simplifies for a classifier network with K softmax output neurons. Then the additive noise must lie above the defining NEM hyperplane [3]. A similar NEM result holds for regression except that the noise-benefit region is a hypersphere. NEM noise can also inject into the hidden-neuron layers in accord with their likelihood functions [12]. The next section extends these results to recurrent BP for classification and regression.

III. Noise-Boosted Recurrent Backpropagation

We next develop the recurrent backpropagation (RBP) algorithm and then show how to noise-boost it using the NEM Theorem. NEM noise can inject in all neurons [12]. This paper focuses only on noise injection into the output neurons.

A. Recurrent Backpropagation

This section presents the learning algorithm for training Long Short-Term Memory (LSTM) with RBP. The LSTM is a deep RNN architecture suitable for problems with long time-lag dependencies [26].

The LSTM network consists of I input neurons, J hidden neurons, and K output neurons. We also have control gates for the input, hidden, and output layers. The input gate controls the input activation. The forget gate controls the hidden activation. And the output gate controls the output activation. These gates each have J neurons. The LSTM network captures the dependency of the input data with respect to time. The time t can take on any value in $\{1, 2, ..., T\}$.

The $J \times I$ matrix $\mathbf{W}^{x}$ connects the hidden neurons to the input neurons. The $K \times J$ matrix $\mathbf{U}^{y}$ connects the output neurons to the hidden neurons. Superscripts x, h, and y denote the respective input, hidden, and output layers. The $J \times I$ matrix $\mathbf{W}^{\gamma}$ connects the input gates to the input neurons and $J \times K$ matrix $\mathbf{V}^{\gamma}$ connects the input gates to the output neurons. The $J \times I$ matrix $\mathbf{W}^{\phi}$ connects the forget gates to the input neurons and $J \times K$ matrix $\mathbf{V}^{\phi}$ connects the forget gate to the output neurons. The $J \times I$ matrix $\mathbf{W}^{\omega}$ connects the output gates to the input neurons and $J \times K$ matrix $\mathbf{V}^{\omega}$ connects the output gates to the output neurons. Superscripts γ, ϕ, and ω denote the respective input, forget, and output gates.

The training involves the forward pass of the input vector and the backward pass of the error. The forward pass propagates the input vector from the input layer to the output layer that has output activation $\mathbf{a}^{y}$. The backward pass propagates the error from the output and hidden layers back to the input layer and updates the network's weights. We alternate these two passes starting with the forward pass until the network converges. RBP seeks those weights that minimize the error function $E(\Theta)$–or maximize the likelihood $L(\Theta)$.

The forward pass initializes the internal memory $h_j^{(0)}$ to 0 for all values of j. The output activation $a_k^{y(0)}$ is 0 for all values of k. We then compute $a_k^{y(t)}$ for $t = 1, 2, ...,$ and T. The input vector $\mathbf{x}^{(t)}$ has I input neuron values. The input neurons have identity activations: $a_i^{x(t)} = x_i^{(t)}$ where $a_i^{x(t)}$ denotes the activation for the j^{th} input neuron due to the input vector $\mathbf{x}^{(t)}$ at time t. The term $o_j^{x(t)}$ denotes the j^{th} value for the transformation from the input space to the hidden state:

$$o_j^{x(t)} = \sum_{i=1}^{I} w_{ji}^{x} a_i^{x(t)} + b_j^{x} \tag{5}$$

where b_j^{x} is the bias for the j^{th} hidden neuron due to the input activation. This transformation is the logistic sigmoidal activation function.

The term $a_j^{x(t)}$ denotes the activation of the j^{th} hidden neuron with respect to the input-space transformation of $\mathbf{x}^{t}$:

$$a_j^{x(t)} = \frac{1}{1 + e^{-o_j^{x(t)}}} \tag{6}$$

The term $o_j^{\gamma(t)}$ is the input to the j^{th} input gate at time t:

$$o_j^{\gamma(t)} = \sum_{i=1}^{I} w_{ji}^{\gamma} a_i^{x(t)} + \sum_{k=1}^{K} v_{jk}^{\gamma} a_k^{y(t-1)} + b_j^{\gamma} \tag{7}$$

where b_j^{γ} is the bias for the j^{th} input gate and $a_k^{y(t-1)}$ is the output activation for time $t - 1$.

The activation $a_j^{\gamma(t)}$ for the j^{th} input gate at time t also uses the logistic sigmoid:

$$a_j^{\gamma(t)} = \frac{1}{1 + e^{-o_j^{\gamma(t)}}} \ . \tag{8}$$

The term $o_j^{h(t)}$ denotes the input to the j^{th} hidden neuron at time t:

$$o_j^{h(t)} = a_j^{x(t)} \circ \ a_j^{\gamma(t)} \tag{9}$$

where $\circ$ denotes *Hadamard* product. The term $o_j^{\phi(t)}$ denotes the input to the j^{th} forget gate. It has the form

$$o_j^{\phi(t)} = \sum_{i=1}^{I} w_{ji}^{\phi} a_i^{x(t)} + \sum_{k=1}^{K} v_{jk}^{\phi} a_k^{y(t-1)} + b_j^{\phi} \tag{10}$$

where b_j^{ϕ} is the bias for the j^{th} forget gate.

The term $s_j^{(t)}$ denotes the state of the j^{th} internal memory at time time t:

$$s_j^{(t)} = o_j^{h(t)} + \left(s_j^{(t-1)} \circ \ a_j^{\gamma(t)}\right) \tag{11}$$

where $s_j^{(t-1)}$ is the state of j^{th} internal memory at time $t - 1$. The activation $a_j^{s(t)}$ of the j^{th} internal memory is also logistic:

$$a_j^{s(t)} = \frac{1}{1 + e^{-s_j^{(t)}}} \tag{12}$$

where superscript s denotes the internal memory. The term $o_j^{\omega(t)}$ denotes the input to the j^{th} output gate. It has the form

$$o_j^{\omega(t)} = \sum_{i=1}^{I} w_{ji}^{\omega} a_i^{x(t)} + \sum_{k=1}^{K} v_{jk}^{\omega} a_k^{y(t-1)} + b_j^{\omega} \tag{13}$$

where b_j^{ω} is the bias for the j^{th} output gate. The term $a_j^{\omega(t)}$ is the activation for the j^{th} output gate at time t. The output gate also uses the logistic sigmoid

$$a_j^{\omega(t)} = \frac{1}{1 + e^{-o_j^{\omega(t)}}} \ . \tag{14}$$

The activation $a_j^{h(t)}$ for the j^{th} output neuron at time t obeys

$$a_j^{h(t)} = a_j^{s(t)} \ \circ \ a_j^{\omega(t)} \tag{15}$$

where $a_j^{s(t)}$ and $a_j^{\omega(t)}$ are from (12) and (14). The term $o_k^{y(t)}$ is the input to the k^{th} output neuron at time t:

$$o_k^{y(t)} = \sum_{j=1}^{J} u_{kj}^{x} a_j^{h(t)} + b_k^{y} \tag{16}$$

where b_k^{y} is the bias for k^{th} output neuron.

The output-layer neurons use softmax activations because this is a classification network [3], [11]. The kth output neuron $a_k^{y(t)}$ for k^{th} has the softmax form

$$a_k^{y(t)} = \frac{\exp(o_k^{y(t)})}{\sum_{l=1}^{K} \exp(o_l^{y(t)})} \ . \tag{17}$$

The backward pass computes the error and its derivative with respect to the weights. The error function $E^{(t)}$ is the cross-entropy because this is a classifier network [11]:

$$\begin{aligned} E(\Theta) &= \sum_{t=1}^{T} E^{(t)} \\ &= \sum_{t=1}^{T} \Big(\sum_{k=1}^{K} y_k^{(t)} ln(a_k^{y(t)}) \Big) \end{aligned} \tag{18}$$

.

The derivative of the error term $E^{(t)}$ with respect to u_{kj}^{y} for $t = T, T - 1, ...,$ and 1 is

$$\begin{aligned} \frac{\partial E^{(t)}}{\partial u_{kj}^{y}} &= \frac{\partial E^{(t)}}{\partial a_k^{y(t)}} \frac{\partial a_k^{y(t)}}{\partial o_k^{y(t)}} \frac{\partial o_k^{y(t)}}{\partial u_{kj}^{y}} \\ &= (a_k^{y(t)} - y_k^{(t)}) a_j^{h(t)} \ . \end{aligned} \tag{19}$$

The partial derivative of the $E^{(t)}$ with respect to the b_k^y is

$$\begin{aligned}\frac{\partial E^{(t)}}{\partial b_k^y} &= \frac{\partial E^{(t)}}{\partial a_k^{y(t)}} \frac{\partial a_k^{y(t)}}{\partial o_k^{y(t)}} \frac{\partial o_k^{y(t)}}{\partial b_k^y} \\ &= (a_k^{y(t)} - y_k^{(t)}) \ . \end{aligned} \tag{20}$$

Let $E^{(t)'}$ denote the derivative of $E^{(t)}$ with respect to $a_j^{h(t)}$. Then

$$\begin{aligned} E^{(t)'} &= \frac{\partial E^{(t)}}{\partial a_j^{h(t)}} \\ &= \Big(\sum_{k=1}^{K} (a_k^{y(t)} - y_k^{(t)}) u_{kj}^y\Big) \ . \end{aligned} \tag{21}$$

We now list the many other partial derivatives in the RPP algorithm.

The derivative of $E^{(t)}$ with respect to w_{ji}^ω is

$$\begin{aligned}\frac{\partial E^{(t)}}{\partial w_{ji}^\omega} &= \frac{\partial E^{(t)}}{\partial a_j^{h(t)}} \frac{\partial a_j^{h(t)}}{\partial a_j^{\omega(t)}} \frac{\partial a_j^{\omega(t)}}{\partial o_j^{\omega(t)}} \frac{\partial o_j^{\omega(t)}}{\partial w_{ji}^\omega} \\ &= E^{(t)'} \ a_j^{s(t)} a_j^{\omega(t)} (1 - a_j^{\omega(t)}) a_i^{x(t)} \ . \end{aligned} \tag{22}$$

The derivative of $E^{(t)}$ with respect to v_{kj}^ω is

$$\begin{aligned}\frac{\partial E^{(t)}}{\partial v_{kj}^\omega} &= \frac{\partial E^{(t)}}{\partial a_j^{h(t)}} \frac{\partial a_j^{h(t)}}{\partial a_j^{\omega(t)}} \frac{\partial a_j^{\omega(t)}}{\partial o_j^{\omega(t)}} \frac{\partial o_j^{\omega(t)}}{\partial v_{kj}^\omega} \\ &= E^{(t)'} \ a_j^{s(t)} a_j^{\omega(t)} (1 - a_j^{\omega(t)}) a_i^{y(t-1)} \ . \end{aligned} \tag{23}$$

The derivative of $E^{(t)}$ with respect to b_j^ω is

$$\begin{aligned}\frac{\partial E^{(t)}}{\partial b_j^\omega} &= \frac{\partial E^{(t)}}{\partial a_j^{h(t)}} \frac{\partial a_j^{h(t)}}{\partial a_j^{\omega(t)}} \frac{\partial a_j^{\omega(t)}}{\partial o_j^{\omega(t)}} \frac{\partial o_j^{\omega(t)}}{\partial b_j^\omega} \\ &= E^{(t)'} a_j^{s(t)} a_j^{\omega(t)} (1 - a_j^{\omega(t)}) \ . \end{aligned} \tag{24}$$

The derivative of $E^{(t)}$ with respect to $s_j^{(t)}$ is

$$\begin{aligned}\frac{\partial E^{(t)}}{\partial s_j^{(t)}} &= \frac{\partial E^{(t)}}{\partial a_j^{h(t)}} \frac{\partial a_j^{h(t)}}{\partial s_j^{(t)}} \\ &= E^{(t)'} \ a_j^{\omega(t)} \circ a_j^{s(t)} (1 - a_j^{s(t)}) \ . \end{aligned} \tag{25}$$

Let $a_j^{\phi(t)'}$ denote the derivative of $a_j^{\phi(t)}$ with respect to $o_j^{\phi(t)}$. The derivative of $a_j^{\phi(t)}$ with respect to $o_j^{\phi(t)}$ is

$$\begin{aligned} a_j^{\phi(t)'} &= \frac{\partial a_j^{\phi(t)}}{\partial o_j^{\phi(t)}} \\ &= a_j^{\phi(t)} (1 - a_j^{\phi(t)}) \ . \end{aligned} \tag{26}$$

The derivative of $E^{(t)}$ with respect to w_{ji}^ϕ is

$$\frac{\partial E^{(t)}}{\partial w_{ji}^\phi} = \sum_{n=1}^{T} \Big(\prod_{m=n+1}^{T} a_j^{\phi(m+1)} \Big) s_j^{(n-1)} a_j^{\phi(n)'} a_i^{x(n)} \ . \tag{27}$$

The derivative of $E^{(t)}$ with respect to v_{jk}^ϕ is

$$\frac{\partial E^{(t)}}{\partial v_{jk}^\phi} = \sum_{n=1}^{T} \Big(\prod_{m=n+1}^{T} a_j^{\phi(m+1)} \Big) s_j^{(n-1)} a_j^{\phi(n)'} a_k^{y(n-1)} \ . \tag{28}$$

The derivative of $E^{(t)}$ with respect to b_j^ϕ is

$$\frac{\partial E^{(t)}}{\partial b_j^\phi} = \sum_{n=1}^{T} \Big(\prod_{m=n+1}^{T} a_j^{\phi(m+1)} \Big) s_j^{(n-1)} a_j^{\phi(n)'} \ . \tag{29}$$

So the derivative of $E^{(t)}$ with respect to $a_j^{\gamma(t)}$ is

$$\begin{aligned}\frac{\partial E^{(t)}}{\partial a_j^{\gamma(t)}} &= \frac{\partial E^{(t)}}{\partial s_j^{(t)}} \frac{\partial s_j^{(t)}}{\partial o_j^{h(t)}} \frac{\partial o_j^{h(t)}}{\partial a_j^{\gamma(t)}} \\ &= \Big(E^{(t)'} \ a_j^{\omega(t)} \circ a_j^{s(t)} (1 - a_j^{s(t)})\Big) a_j^{x(t)} \ . \end{aligned} \tag{30}$$

The derivative of $E^{(t)}$ with respect to w_{ji}^γ is

$$\begin{aligned}\frac{\partial E^{(t)}}{\partial w_{ji}^\gamma} &= \frac{\partial E^{(t)}}{\partial a_j^{\gamma(t)}} \frac{\partial a_j^{\gamma(t)}}{\partial o_j^{\gamma(t)}} \frac{\partial o_j^{\gamma(t)}}{\partial w_{ji}^\gamma} \\ &= \frac{\partial E^{(t)}}{\partial a_j^{\gamma(t)}} \ a_j^{\gamma(t)} (1 - a_j^{\gamma(t)}) a_i^{x(t)} \ . \end{aligned} \tag{31}$$

The derivative of $E^{(t)}$ with respect to v_{jk}^γ is

$$\begin{aligned}\frac{\partial E^{(t)}}{\partial v_{jk}^\gamma} &= \frac{\partial E^{(t)}}{\partial a_j^{\gamma(t)}} \frac{\partial a_j^{\gamma(t)}}{\partial o_j^{\gamma(t)}} \frac{\partial o_j^{\gamma(t)}}{\partial v_{jk}^\gamma} \\ &= \frac{\partial E^{(t)}}{\partial a_j^{\gamma(t)}} \ a_j^{\gamma(t)} (1 - a_j^{\gamma(t)}) a_k^{y(t-1)} \ . \end{aligned} \tag{32}$$

The derivative of $E^{(t)}$ with respect to b_j^γ is

$$\begin{aligned}\frac{\partial E^{(t)}}{\partial b_j^\gamma} &= \frac{\partial E^{(t)}}{\partial a_j^{\gamma(t)}} \frac{\partial a_j^{\gamma(t)}}{\partial o_j^{\gamma(t)}} \frac{\partial o_j^{\gamma(t)}}{\partial b_k^\gamma} \\ &= \frac{\partial E^{(t)}}{\partial a_j^{\gamma(t)}} \ a_j^{\gamma(t)} (1 - a_j^{\gamma(t)}) \ . \end{aligned} \tag{33}$$

The derivative of $E^{(t)}$ with respect to $a_j^{x(t)}$ is

$$\begin{aligned}\frac{\partial E^{(t)}}{\partial a_j^{x(t)}} &= \frac{\partial E^{(t)}}{\partial s_j^{(t)}} \frac{\partial s_j^{(t)}}{\partial o_j^{h(t)}} \frac{\partial o_j^{h(t)}}{\partial a_j^{x(t)}} \\ &= \Big(E^{(t)'} \ a_j^{\omega(t)} \circ a_j^{s(t)} (1 - a_j^{s(t)})\Big) a_j^{\gamma(t)} \ . \end{aligned} \tag{34}$$

The derivative of $E^{(t)}$ with respect to w_{ji}^x is

$$\begin{aligned}\frac{\partial E^{(t)}}{\partial w_{ji}^x} &= \frac{\partial E^{(t)}}{\partial a_j^{x(t)}} \frac{\partial a_j^{x(t)}}{\partial o_j^{x(t)}} \frac{\partial o_j^{x(t)}}{\partial w_{ji}^x} \\ &= \frac{\partial E^{(t)}}{\partial a_j^{x(t)}} \ a_i^{x(t)} (1 - a_j^{\gamma(t)}) a_i^{x(t)} \ . \end{aligned} \tag{35}$$

The derivative of $E^{(t)}$ with respect to the b_j^x is

$$\frac{\partial E^{(t)}}{\partial b_j^x} = \frac{\partial E^{(t)}}{\partial a_j^{x(t)}} \frac{\partial a_j^{x(t)}}{\partial o_j^{x(t)}} \frac{\partial o_j^{x(t)}}{\partial b_j^x}$$
$$= \frac{\partial E^{(t)}}{\partial a_j^{x(t)}} a_i^{x(t)}(1 - a_j^{\gamma(t)}) \ . \tag{36}$$

These partial derivatives form the update rules for training the RNN with the RBP algorithm.

B. Noisy EM Theorems for RBP

We can now state and prove the new NEM RBP theorems for a classifier RNN and for a regression RNN. We start with the classifier result.

Theorem 2: RBP Noise Benefit for RNN Classification
The NEM positivity condition (3) holds for the maximum likelihood training of a classifier recurrent neural network with noise injection in the output softmax neurons if the following hyperplane condition holds :

$$\mathbb{E}_{\mathbf{y},\mathbf{h},\mathbf{n}|\mathbf{x},\mathbf{\Theta}^*}\Big\{\sum_{t=1}^{T} \mathbf{n^{(t)}}^T ln\ \mathbf{a}^{y(t)}\Big\} \geq 0\ . \tag{37}$$

Proof : We first show that the cross entropy $E(\Theta)$ equals the sum of the negative log-likelihood functions over time $t = 1, 2, ...,$ and T. The error $E(\Theta)$ is the cross entropy for a classifier network:

$$E(\Theta) = \sum_{t=1}^{T} E^{(t)} \tag{38}$$
$$= -\sum_{t=1}^{T}\sum_{k=1}^{K} y_k^{(t)} \ln\ a_k^{y(t)} \tag{39}$$
$$= -\sum_{t=1}^{T} \ln\Big[\prod_{k=1}^{K} (a_k^{y(t)})^{y_k^{(t)}}\Big] \tag{40}$$
$$= -\sum_{t=1}^{T} \ln\Big[\prod_{k=1}^{K} p_k(y^{(t)} = y_k^{(t)}|\mathbf{x}, \Theta)\Big] \tag{41}$$
$$= -\sum_{t=1}^{T} \ln\ p(y^{(t)}|\mathbf{x}, \Theta) \tag{42}$$
$$= -\ln \prod_{t=1}^{T} p(y^{(t)}|\mathbf{x}, \Theta) \tag{43}$$
$$= -L(\Theta) \tag{44}$$

So minimizing the cross entropy $E(\Theta)$ maximizes the log-likehood function $L(\Theta)$: $p(\mathbf{y}|\mathbf{x}, \mathbf{\Theta}) = \mathbf{e}^{-\mathbf{E}(\mathbf{\Theta})}$.

The NEM sufficient condition (3) simplifies to the product of exponentiated output activations over time t and each with K output neurons. Then additive noise injection gives

$$\frac{p(\mathbf{n}+\mathbf{y}, \mathbf{h}|\mathbf{x}, \Theta)}{p(\mathbf{y}, \mathbf{h}|\mathbf{x}, \Theta)} = \prod_{t=1}^{T} \frac{p(\mathbf{n^{(t)}} + \mathbf{y^{(t)}}, \mathbf{h}|\mathbf{x}, \Theta)\ p(\mathbf{h}|\mathbf{x}, \Theta)}{p(\mathbf{h}|\mathbf{x}, \Theta)\ p(\mathbf{y^{(t)}}, \mathbf{h}|\mathbf{x}, \Theta)}$$
$$= \prod_{t=1}^{T} \frac{p(\mathbf{n^{(t)}} + \mathbf{y^{(t)}}|\mathbf{h}, \mathbf{x}, \Theta)}{p(\mathbf{y^{(t)}}|\mathbf{h}, \mathbf{x}, \Theta)} \tag{45}$$
$$= \prod_{t=1}^{T} \Big[\prod_{k=1}^{K} \frac{(a_k^{y(t)})^{n_k^{(t)} + y_k^{(t)}}}{(a_k^{y(t)})^{y_k^{(t)}}}\Big] \tag{46}$$
$$= \prod_{t=1}^{T} \Big[\prod_{k=1}^{K} (a_k^{y(t)})^{n_k^{(t)}}\Big]\ . \tag{47}$$

Then the NEM positivity condition in (3) becomes

$$\mathbb{E}_{\mathbf{y},\mathbf{h},\mathbf{N}|\mathbf{x},\mathbf{\Theta}^*}\Big\{\ln \prod_{t=1}^{T}\Big[\prod_{k=1}^{K} (a_k^{y(t)})^{n_k^{(t)}}\Big]\Big\} \geq 0\ . \tag{48}$$

This simplifies to the inequality

$$\mathbb{E}_{\mathbf{y},\mathbf{h},\mathbf{N}|\mathbf{x},\mathbf{\Theta}^*}\Big\{\sum_{t=1}^{T}\Big[\sum_{k=1}^{K} n_k^{(t)}\ \ln(a_k^{y(t)})\Big]\Big\} \geq 0\ . \tag{49}$$

The inner summation represents the inner-product of $\mathbf{n}^{(t)}$ with ln $\mathbf{a}^{y(t)}$. Then we can rewrite (49) as the hyperplane inequality

$$\mathbb{E}_{\mathbf{y},\mathbf{h},\mathbf{N}|\mathbf{x},\mathbf{\Theta}^*}\Big\{\sum_{t=1}^{T} \mathbf{n^{(t)}}^T ln\ \mathbf{a}^{y(t)}\Big\} \geq 0\ . \tag{50}$$

■

We next state and prove the NEM RBP theorem for regression networks.

Theorem 3: RBP Noise Benefit for RNN Regression
The NEM positivity condition (3) holds for the maximum likelihood training of a regression RNN with Gaussian target vector $\mathbf{y} \sim \mathcal{N}(\mathbf{y}|\mathbf{a^y}, \mathbf{I})$ *if the following inequality condition holds:*

$$\mathbb{E}_{\mathbf{y},\mathbf{h},\mathbf{N}|\mathbf{x},\mathbf{\Theta}^*}\Big\{\sum_{t=1}^{T} ||(\mathbf{y}^{(t)} + \mathbf{n}^{(t)} - \mathbf{a}^{y(t)}||^2\Big\}$$
$$- \mathbb{E}_{\mathbf{y},\mathbf{h},\mathbf{N}|\mathbf{x},\mathbf{\Theta}^*}\Big\{\sum_{t=1}^{T} ||\mathbf{y}^{(t)} - \mathbf{a}^{y(t)}||^2\Big\} \leq 0\ . \tag{51}$$

Proof: The error function $E(\Theta)$ is the squared error for a regression network [3]. We first show that minimizing the squared error $E(\Theta)$ is the same as maximizing the likelihood function $L(\Theta)$:

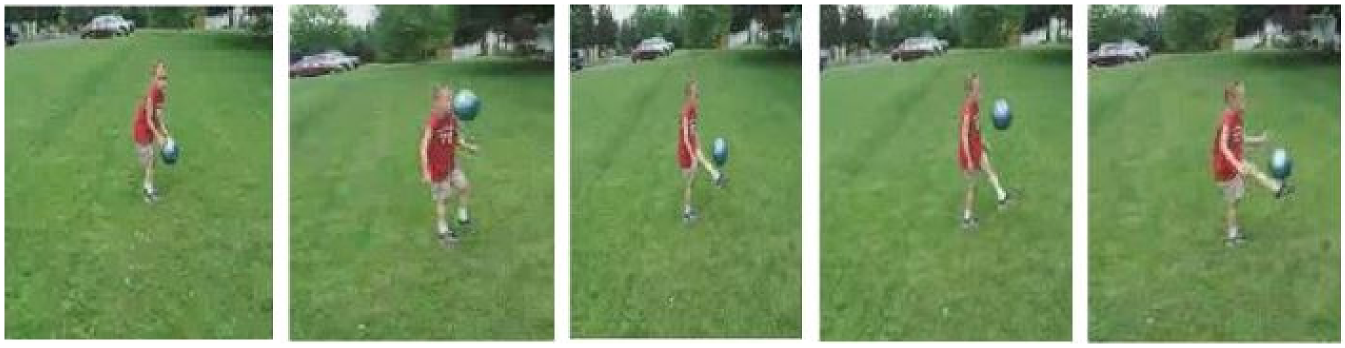

Fig. 1: *The first 5 frames from a UCF YouTube video sample of juggling a soccer ball. Such soccer videos were from the 6th of 11 categories in the classification experiment. So the target output for this sampled video clip was the 1-in-K-coded bit vector* $[0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0]$.

$$E(\Theta) = \sum_{t=1}^{T} E^{(t)} \tag{52}$$
$$= -\sum_{t=1}^{T} -\frac{||\mathbf{y}^{(t)} - \mathbf{a}^{y(t)}||^2}{2} \tag{53}$$
$$= -\sum_{t=1}^{T} \ln\Big[\exp\Big\{-\frac{||\mathbf{y}^{(t)} - \mathbf{a}^{y(t)}||^2}{2}\Big\}\Big] \tag{54}$$
$$- T\ \ln(2\pi)^{-\frac{K}{2}} + T\ \ln(2\pi)^{-\frac{K}{2}} \tag{55}$$
$$= -\ln\Big[\prod_{t=1}^{T} p(\mathbf{y} = \mathbf{y}^{(t)}|\mathbf{x},\Theta)\Big] + T\ \ln(2\pi)^{-\frac{K}{2}}$$
$$= -L(\Theta) + T\ \ln(2\pi)^{-\frac{K}{2}} \tag{56}$$

where the network's output probability density function $p(\mathbf{y} = \mathbf{y}^{(t)}|\mathbf{x},\Theta)$ is the vector normal density $\mathcal{N}(\mathbf{y^{(t)}}|\mathbf{a}^{(t)},\Theta)$. So minimizing the squared error $E(\Theta)$ maximizes the likelihood $L(\Theta)$ because the last term in (56) does not depend on Θ.

Injecting additive noise into the output neurons gives

$$\frac{p(\mathbf{n}+\mathbf{y},\mathbf{h}|\mathbf{x},\Theta)}{p(\mathbf{y},\mathbf{h}|\mathbf{x},\Theta)} = \prod_{t=1}^{T} \frac{p(\mathbf{n^{(t)}}+\mathbf{y^{(t)}},\mathbf{h}|\mathbf{x},\Theta)\ p(\mathbf{h}|\mathbf{x},\Theta)}{p(\mathbf{h}|\mathbf{x},\Theta)\ p(\mathbf{y^{(t)}},\mathbf{h}|\mathbf{x},\Theta)}$$
$$= \prod_{t=1}^{T} \frac{p(\mathbf{n^{(t)}}+\mathbf{y^{(t)}}|\mathbf{h},\mathbf{x},\Theta)}{p(\mathbf{y^{(t)}}|\mathbf{h},\mathbf{x},\Theta)} \tag{57}$$
$$= \prod_{t=1}^{T} \frac{\exp\Big(\frac{1}{2}||(\mathbf{y}^{(t)} - \mathbf{a}^{y(t)}||^2\Big)}{\exp\Big(\frac{1}{2}||(\mathbf{y}^{(t)} + \mathbf{n}^{(t)} - \mathbf{a}^{y(t)}||^2\Big)}\ . \tag{58}$$

Then the NEM positivity condition in (3) simplifies to

$$\mathbb{E}_{\mathbf{y},\mathbf{h},\mathbf{N}|\mathbf{x},\mathbf{\Theta}^*}\Big\{\ln\prod_{t=1}^{T}\frac{\exp(\frac{1}{2}||(\mathbf{y}^{(t)} - \mathbf{a}^{y(t)}||^2)}{\exp(\frac{1}{2}||(\mathbf{y}^{(t)} + \mathbf{n}^{(t)} - \mathbf{a}^{y(t)}||^2)}\Big\} \geq 0\ . \tag{59}$$

This positivity condition simplifies to the hyperspherical inequality

$$\mathbb{E}_{\mathbf{y},\mathbf{h},\mathbf{N}|\mathbf{x},\mathbf{\Theta}^*}\Big\{\sum_{t=1}^{T}||(\mathbf{y}^{(t)} + \mathbf{n}^{(t)} - \mathbf{a}^{y(t)}||^2\Big\}$$
$$- \mathbb{E}_{\mathbf{y},\mathbf{h},\mathbf{N}|\mathbf{x},\mathbf{\Theta}^*}\Big\{\sum_{t=1}^{T}||\mathbf{y}^{(t)} - \mathbf{a}^{y(t)}||^2\Big\} \leq 0\ . \tag{60}$$

■

These NEM theorems extend directly to other RNN architectures such as those with gated recurrent units and Elman RNNs [29].

Equation (61) below gives the update rule for training a RNN with BP through time. It applies to both classification and regression networks. The update rule for the n^{th} training iteration is

$$\Theta^{(n+1)} = \Theta^{(n)} + \eta \bigtriangledown_{\Theta} L(\Theta)\Big|_{\Theta=\Theta^{(n)}} \tag{61}$$
$$= \Theta^{(n)} - \eta \bigtriangledown_{\Theta} E(\Theta)\Big|_{\Theta=\Theta^{(n)}} \tag{62}$$

for $\Theta \in \{u^y_{kj}, b^k_y, w^\omega_{ji}, v^\omega_{jk}, b^\omega_j, w^\phi_{ji}, v^\phi_{jk}, b^\phi_j, w^\gamma_{ji}, v^\gamma_{jk}, b^\gamma_j, w^x_{ji}, b^x_j\}$ and η is the learning rate.

The partial derivatives in the gradient of (62) come from (19), (20), (22)−(24), (27)−(29), (31)−(33), (35), and (36). Algorithm 1 states the details of the Noisy RBP algorithm with a LSTM architecture and for softmax output activation. A related algorithm injects noise into the hidden units as well [3]. This produces further speed-ups in general.

IV. Video Classification Simulation Results

We injected additive NEM noise during the RBP training of a LSTM-RNN video classifier. We used 80:20 splits for training and test sets. We extracted 26,000 training instances from the standard UCF sports action YouTube videos [1], [2]. We used 6,700 video-clip samples for testing the network after training. Each training instance had 12 image frames sampled uniformly from a sports-action video at the rate of 6 frames per second. Each image frame in the dataset had 72×96 pixels. Each pixel value was between 0 and 1.

We fed the pixel values into the input neurons of the LSTM-RNN and tried different numbers of hidden neurons. The

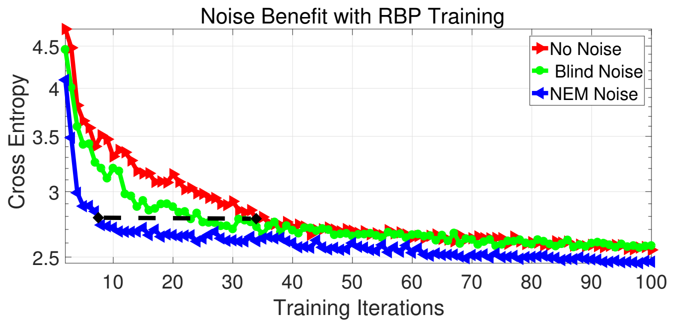

Fig. 2: *Noise benefit comparisons. NEM Noise benefit from noise injection into the output layer of a LSTM RNN. Adding NEM noise to the output neurons of the LSTM RNN improved the performance of RBP training. Training RBP with NEM noise in just the 11 output softmax neurons led to 60% fewer iterations to reach a common low value of cross entropy. Noise-boosted RBP took 8 iterations to reach the cross-entropy value 2.7 while ordinary noiseless RBP took 34 iterations to reach the same cross entropy value. There was a 20.6 % maximum decrease in cross-entropy. NEM noise can also inject into the hidden neurons. Blind noise gave little improvement.*

```
Data: $N$ input sets $\{X_1, \dots, X_N\}$ where each input set $X_n = \{\mathbf{x}_n^{(1)}, \dots, \mathbf{x}_n^{(T)}\}$. $N$ target sets $\{Y_1, \dots, Y_N\}$ where target
    set $Y_n = \{\mathbf{y}_n^{(1)}, \dots, \mathbf{y}_n^{(T)}\}$. Number of epochs for running backpropagation through-time $R$.
Result: Trained RNN weights.
Initialize the network weights $\Theta$.
while: epoch $r : 1 \rightarrow R$ do
    while training input set $n : 1 \rightarrow N$ do
        FEEDFORWARD
        while $t : 1 \rightarrow T$ do
            • Compute the activation of input, forget, and output gates from $\mathbf{x}_n^{(t)}$ and $\mathbf{y}_n^{(t-1)}$.
            • Compute the memory cell activation $\mathbf{s}_n^{(t)}$ and output activation $\mathbf{a}_n^{y(t)}$.
        end
        ADD NOISE
        while $t : 1 \rightarrow T$ do
            • Generate noise vector $\mathbf{n}$:
            if $\left(\mathbf{n}^T \log\left(\mathbf{a}_n^{y(t)}\right) \geq 0\right)$
                • Add the noise: $\mathbf{y}_n^{(t)} \leftarrow \mathbf{y}_n^{(t)} + \mathbf{n}$
            else
                • Do nothing
            end
        end
        BACKPROPAGATE ERROR THOUGH TIME
        Initialize: $\nabla_\Theta E(\Theta) = 0$
        while $t : T \rightarrow 1$ do
            • Compute the cross-entropy $E^t(\Theta)$.
            • Compute cross-entropy gradient $\nabla_\Theta E^t(\Theta)$
            • Update the gradient $\nabla_\Theta E(\Theta) = \nabla_\Theta E(\Theta) + \nabla_\Theta E^t(\Theta)$
        end
        • Update the parameters $\Theta$ using gradient descent:
          $\Theta = \Theta - \eta \nabla_\Theta E(\Theta)$
    end
end
```

Algorithm 1: *Noisy RBP algorithm for training a classifier RNN with an LSTM architecture.*

input, forget, and output gates all had the same number of neurons. The output layer had 11 neurons that represented the 11 categories of sports actions in the dataset. These sport actions were basketball shooting, biking, diving, golf swinging, horseback riding, soccer juggling, swinging, tennis swinging, trampoline jumping, volleyball spiking, and walking. Figure 1 shows 5 sampled consecutive frames from a video in the soccer-juggling category. We fed the frames into the LTSM network as a sequence of data over time.

The RNN used 11 softmax neurons at the output layer because the LSTM-RNN was a classification network. The other layers and the gates used logistic neurons as discussed above. Then the noise benefit tapered off. We used uniform noise from $U(0, 0.1)$ with annealing factor 1. This scaled down the noise by n^{-1} as the iteration number n grew. The Adaptive Moment Estimation optimizer picked the learning rate as n grew. The initial learning rate was $\eta = 0.001$.

We compared the training performance over 200 iterations and found that the best noise benefit occurred with 200 internal memory neurons. Figure 2 shows the cross entropy for different RBP training regimens and the resulting NEM-RBP speed-up of convergence. Simply injecting faint blind (uniform) noise into the output neurons performed slightly better than no noise injection but not nearly as well as NEM noise injection. The figure also shows that noiseless RBP took 26 more iterations to reach the same cross-entropy value of 2.7. So NEM-boosted RBP needed 60% fewer training iterations to converge to the same low cross-entropy value as did noiseless RBP.

The injection of NEM noise also improved the classification accuracy on the video-clip test set. This apparently occurred because the network likelihood lower-bounds the classification accuracy as we have shown elsewhere [12]. We compared the classification accuracy of NEM-noise RBP training with the accuracy for noiseless RBP training. The accuracy measure counted all true positives. NEM noise produced a 2% relative gain in classification accuracy. The noiseless RBP training gave 81% classification accuracy on the test set. The injection of NEM noise gave 83% classification accuracy on the test set. This represents 134 more correct classifications than we found for noiseless RBP training. We ran the simulation on a

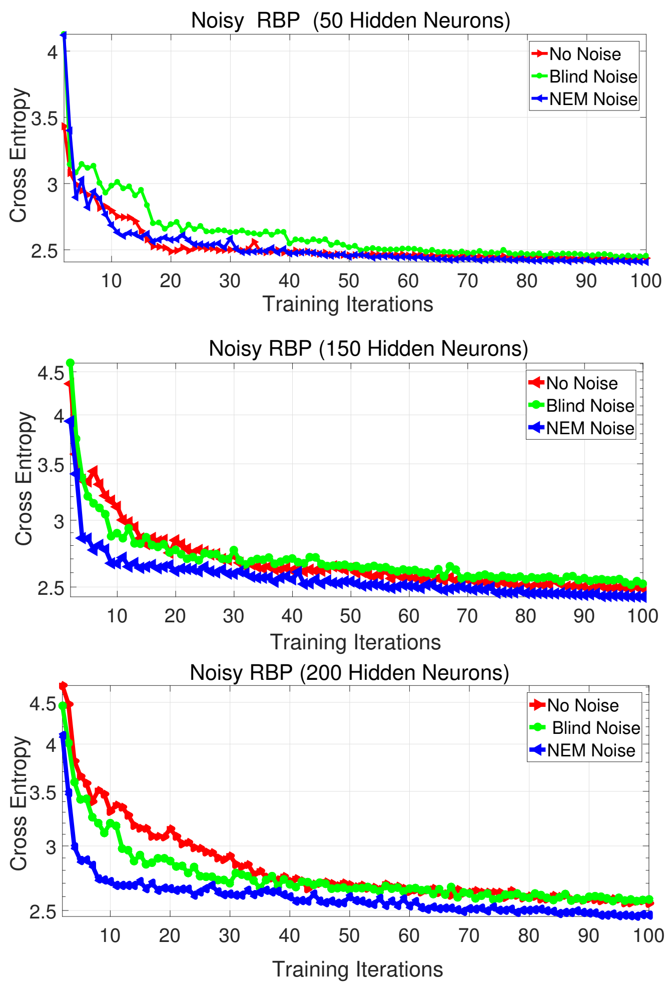

Fig. 3: *Hidden neuron memory-size effects on injecting noise into the output neurons of the RNN. The NEM noise benefit became more pronounced as the number of hidden neurons increased. The best noise effect occurred with 200 memory neurons and tapered off thereafter.*

Google cloud instance with 8 vCPUs, 30GB RAM, and 50GB system disk.

V. Conclusion

Careful noise injection speeded up convergence of the recurrent backpropagation (RBP) algorithm for video samples. The key insight in the noise boost is that ordinary BP is a special case of the generalized EM algorithm. Then noise-boosting EM leads to noise-boosting BP and its recurrent versions for both classification and regression. Simulations showed that noise-boosted RBP training substantially outperformed noiseless RBP for classifying the UCF sports action Youtube video clips. Further noise benefits should result for noise injection into the hidden neurons and the use of convolutional masks [3].

References

[1] J. A. Mikel D. Rodriguez and M. Shah, "Action mach: A spatio-temporal maximum average correlation height filter for action recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[2] K. Soomro and A. R. Zamir, "Action recognition in realistic sports videos," in *Computer Vision in Sports*. Springer, 2014, pp. 181–208.

[3] K. Audhkhasi, O. Osoba, and B. Kosko, "Noise-enhanced convolutional neural networks," *Neural Networks*, vol. 78, pp. 15–23, 2016.

[4] D. Rumelhart, G. Hinton, and R. Williams, "Learning representations by back-propagating errors," *Nature*, pp. 323–533, 1986.

[5] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015.

[6] M. Jordan and T. Mitchell, "Machine learning: trends, perspectives, and prospects," *Science*, vol. 349, pp. 255–260, 2015.

[7] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the royal statistical society. Series B (methodological)*, pp. 1–38, 1977.

[8] O. Osoba, S. Mitaim, and B. Kosko, "The noisy expectation–maximization algorithm," *Fluctuation and Noise Letters*, vol. 12, no. 03, p. 1350012, 2013.

[9] O. Osoba and B. Kosko, "Noise-Enhanced Clustering and Competitive Learning Algorithms," *Neural Networks*, Jan. 2013.

[10] ——, "The noisy expectation-maximization algorithm for multiplicative noise injection," *Fluctuation and Noise Letters*, p. 1650007, 2016.

[11] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[12] B. Kosko, K. Audhkhasi, and O. Osoba, "Noise can speed backpropagation learning and deep bidirectional pretraining," in review.

[13] B. Kosko, *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*. Prentice Hall, 1991.

[14] P. Hänggi, "Stochastic resonance in biology," *ChemPhysChem*, vol. 3, no. 3, pp. 285–290, 2002.

[15] B. Kosko, *Noise*. Penguin, 2006.

[16] A. Patel and B. Kosko, "Stochastic resonance in continuous and spiking neuron models with Levy noise," *IEEE Transactions on Neural Networks*, vol. 19, no. 12, pp. 1993–2008, 2008.

[17] M. McDonnell, N. Stocks, C. Pearce, and D. Abbott, *Stochastic resonance: from suprathreshold stochastic resonance to stochastic signal quantization*. Cambridge University Press, 2008.

[18] M. Wilde and B. Kosko, "Quantum forbidden-interval theorems for stochastic resonance," *Journal of Physical A: Mathematical Theory*, vol. 42, no. 46, 2009.

[19] A. Patel and B. Kosko, "Error-probability noise benefits in threshold neural signal detection," *Neural Networks*, vol. 22, no. 5, pp. 697–706, 2009.

[20] B. Franzke and B. Kosko, "Using noise to speed up Markov chain Monte Carlo estimation," *Procedia Computer Science*, vol. 53, pp. 113–120, 2015.

[21] L. Gammaitoni, P. Hänggi, P. Jung, and F. Marchesoni, "Stochastic Resonance," *Reviews of Modern Physics*, vol. 70, no. 1, pp. 223–287, January 1998.

[22] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 6645–6649.

[23] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 2625–2634.

[24] J. Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici, "Beyond short snippets: Deep networks for video classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 4694–4702.

[25] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[26] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[27] A.-r. M. Alex Graves and G. Hinton, "Speech recognition with deep recurrent neural networks," *Proceedings of International Conference on Acoustic, Speech and Signal Processing*, pp. 6645–6649, 2013.

[28] K. Audhkhasi, O. Osoba, and B. Kosko, "Noisy hidden Markov models for speech recognition," in *Proceedings of the 2013 International Joint Conference on Neural Networks*. IEEE, 2013, pp. 1–6.

[29] P. Rodriguez, J. Wiles, and J. L. Elman, "A recurrent neural network that learns to count," *Connection Science*, vol. 11, no. 1, pp. 5–40, 1999.