Neurocomputing 559 (2023) 126438

Contents lists available at ScienceDirect

Neurocomputing

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

Noise-boosted recurrent backpropagation

Olaoluwa Adigun[a], Bart Kosko[a,*]

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

ARTICLE INFO

Communicated by Zidong Wang

Keywords:
Recurrent backpropagation
Recurrent neural network
Generalized expectation maximization
Long short-term memory
Gated recurrent unit
NEM noise

ABSTRACT

A statistical formulation of recurrent backpropagation (RBP) allows direct noise boosting for time-varying classification and regression. The noise boost reduces training iterations and improves accuracy. The injected noise is just that noise that makes the current signal more probable. This noise-boost result extends the two 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. The noise-benefit conditions differ for additive and multiplicative noise in RBP. We tested noise-boosted RBP classifiers on 11 classes of sports video clips and tested RBP regressors on predicting the dollar-rupee exchange rate. Injecting noisy-EM (NEM) noise outperformed injecting blind noise or injecting no noise at all. Additive NEM noise usually outperformed multiplicative noise. The best case of NEM noise injection with RBP training of a recurrent neural classification model speeded up its training by 60% and improved its classification accuracy by 9.51% compared with noiseless RBP training and accuracy. The best performance of the NEM noise with the RBP training of a recurrent neural regression model yielded a 38% speed-up in training and also reduced the squared error by 49.3%. The injection of the additive NEM noise in the output and hidden neurons performed best.

1. Noise Boosting Recurrent Backpropagation

We show that a statistical formulation of recurrent neural networks can improve recurrent backpropagation (RBP) through noise injection into the neurons. RBP remains the most popular algorithm for training recurrent neural networks (RNNs) [1–5]. Proper randomization or noise boosting speeds RBP convergence on average for both classification and regression. It also improves classification accuracy. Fig. 1 shows the additive noisy Expectation–Maximization (NEM) noise for a recurrent neural classifier and how it benefits its RBP training.

The injected noise is *not* simple blind white noise or faint independent Gaussian or uniform noise as with earlier neural-net noise-injection schemes [6–8]. The noise is instead just that dependent noise that makes the current signal more probable on average. Such noise can add to or multiply the signals in the output or hidden neurons. The sufficient conditions for an average noise benefit depend on the activation structure of the neurons in a layer. They also depend on whether the network performs classification or regression.

The ubiquitous backpropagation algorithm underlies modern deep learning [1,9–14]. We review its maximum-likelihood reformulation below and state Theorem 1 from [15]. This result shows that backpropagation is a form of the generalized Expectation–Maximization (GEM) algorithm. Theorems 2 and 3 extend these results to the noise-boosting of RBP for classification and regression by injecting noise into the output neurons. Theorems 4 and 5 further extend these results to allow noise injection into a deep network's hidden units.

We tested RBP noise boosting on two types of classifier networks for video recognition. The two types were long short-term memory (LSTM) [3,16–20] and gated-recurrent-unit (GRU) [21,22] recurrent networks. Fig. 2 shows the convolutional structure of the LSTM-RNN for video classification. Fig. 7 shows a typical sampled video that the RNNs classified to one of 11 types of videos from the standard UCF-11 sports-action YouTube video dataset [23,24]. Fig. 8 gives more detail on the convolutional structure of the RNN classifier. Fig. 9 compares noiseless and noise-boosted LSTM-RNN classifiers on the UCF-11 video dataset.

We also tested the noise effect on LSTM and GRU regression networks for time-series prediction of the dollar-rupee exchange rate. The dataset contains the exchange rate from the US dollar to the Indian rupee for 9,697 consecutive days. Fig. 3 shows the RNN regression structure that predicts the dollar-rupee exchange rate given information about the exchange rate on the previous 4 days. Fig. 10 shows the dollar-rupee time-series data over this period of 9,697 days. The RNN regression

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

https://doi.org/10.1016/j.neucom.2023.126438
Received 20 October 2021; Received in revised form 10 May 2023; Accepted 5 June 2023
Available online 30 June 2023

models trained on exchange-rate data from day 0 to day 8,000. We then tested the RNN regression models on exchange-rate data from day 8,001 to day 9,697. Fig. 11 compares these predicted exchange rates with the actual exchange rates. Fig. 12 shows the noise benefits in the noise-boosted LSTM-RNN regression models. Fig. 14 shows the related noise benefits in the GRU-RNN regression models. The GRU-RNNs outperformed the LSTM-RNNs in both speed of convergence and in accuracy.

Simulations show that the noise-boosted recurrent systems converged faster and with better accuracy than did either their noiseless versions or simply injecting blind noise into the neurons. Blind-noise injection did produce a small benefit in some cases as other researchers have found in simpler neural networks [7,8]. This appears to be a “stochastic resonance” or dithering noise benefit often found in signal systems that use threshold-like units [25–37]. Additive noise injection outperformed multiplicative noise injection in most cases. Multiplicative noise boosting did outperform additive noise boosting for the GRU regressor.

We first extend the recent result that the popular backpropagation algorithm [9,10] is a special case of the Expectation–Maximization (EM) algorithm for maximum likelihood with hidden variables or missing data [15,38]. RBP also admits such an EM statistical formulation. We use the recent general result that noise can speed the average convergence of the EM algorithm [39,40]. This produces different types of noise-boosted RBPs that depend on the network architecture and function.

The neural network’s probabilistic structure does require that the network obey what we call *backpropagation invariance*: the gradient of the network’s log-likelihood function L must give back the same BP learning laws. The next section explains BP invariance and shows how it leads to the generalized EM algorithm. Then we can noise-boost RBP by noise-boosting the generalized EM algorithm. This follows from the results [41,42] that show how EM-based noise injection improves some forms of the GEM algorithm.

The next sections have the following form:

- Section 2 reviews the backpropagation algorithm as a form of maximum-likelihood estimation (MLE). It shows in Theorem 1 that backpropagation is an instance of the generalized EM algorithm when backpropagation invariance holds at the neural layers.
- Section 3 presents the noisy Expectation–Maximization (NEM) benefit for the maximum-likelihood estimation. It shows that the noise samples that make the data more probable improve the performance of the GEM algorithm.
- Section 4 presents recurrent backpropagation as a form of GEM. This framework extends the NEM noise benefit to RBP.
- Section 5 states and proves the theorems for noise-boosting RBP. Theorems 2 and 3 apply to noise-boosting RBP for recurrent neural classifiers and for recurrent neural regression models by injecting NEM noise into the output neurons. Theorems 4 and 5 extend the NEM benefit to noise injection into the hidden units of LSTM-RNN and GRU-RNN.
- Section 6 presents the simulation results that show that NEM noise improves the performance of the RBP. The NEM noise injects into the output layer or hidden units or into both. The NEM benefit holds for both additive and multiplicative noise injection in recurrent neural classifiers and regressors.

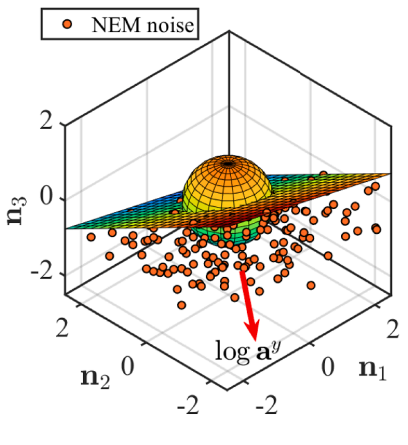
2. Backpropagation Invariance and the EM Algorithm

We first develop the main ideas behind the probabilistic formulation of neural networks and the connection between BP and the EM algorithm.

The key structure in the argument is the gradient identity $\nabla \log p(\mathbf{y}|\mathbf{x}, \Theta^n) = \nabla Q(\Theta^n|\Theta^n)$ in (18)–(19) below. BP invariance refers to the left side of this gradient identity. BP invariance requires that the gradient $\nabla \log p(\mathbf{y}|\mathbf{x}, \Theta^n)$ must equal the *same* BP learning laws for a given network architecture.

BP invariance holds for neural classifiers with softmax output neurons if the network’s output likelihood $p(\mathbf{y}|\mathbf{x}, \Theta^n)$ is a one-shot multinomial probability or categorical distribution. Then the network’s output probability $p(\mathbf{y}|\mathbf{x}, \Theta^n)$ corresponds to the roll of a K-sided die. The negative of the log-likelihood $L = \log p(\mathbf{y}|\mathbf{x}, \Theta^n)$ of the output equals the cross entropy. So minimizing the cross entropy maximizes the output log-likelihood L.

BP invariance holds for a regression model if the output probability $p(\mathbf{y}|\mathbf{x}, \Theta^n)$ is a vector normal density and if the output neurons are linear or identity units. Then the output log-likelihood L is proportional to the negative of the squared error. So minimizing the squared error equals maximizing the output log-likelihood L. The parameter gradient $\nabla_{\Theta^n} L$



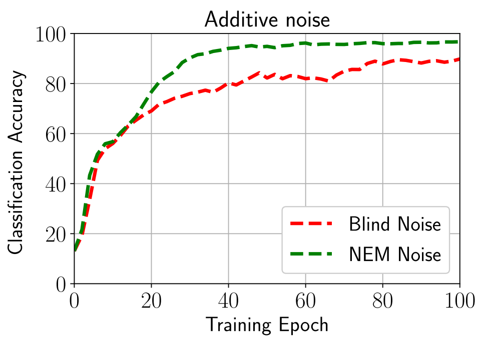


Fig. 1. NEM noise benefit with additive noise injection in the output neurons of a neural network: The NEM positivity condition guarantees a NEM noise benefit with backpropagation training. (a) shows the NEM hyperplane for injecting additive NEM noise in the output neurons of a neural classifier. The noise samples below the NEM hyperplane satisfy the positivity condition and improve accuracy on average. The output activation vector is $\mathbf{a}^y = [0.6, 0.3, 0.1]$ for the target $\mathbf{y} = [1, 0, 0]$. The NEM noise differs from blind or dither noise. The NEM noise comes from picking random samples from a multivariate Gaussian density with mean vector $\mathbf{0}$ and identity covariance $\mathbf{I}$. These normal samples satisfy the positivity condition in (40). The blind noise samples lie inside the sphere centered at $[0,0,0]$. (b) compares the effect of additive noise injection on the performance of LSTM-RNN classifiers trained on the YouTube sports-video UCF-11 test set. Injected NEM noise quickly outperformed injected blind or dither noise as training proceeded.

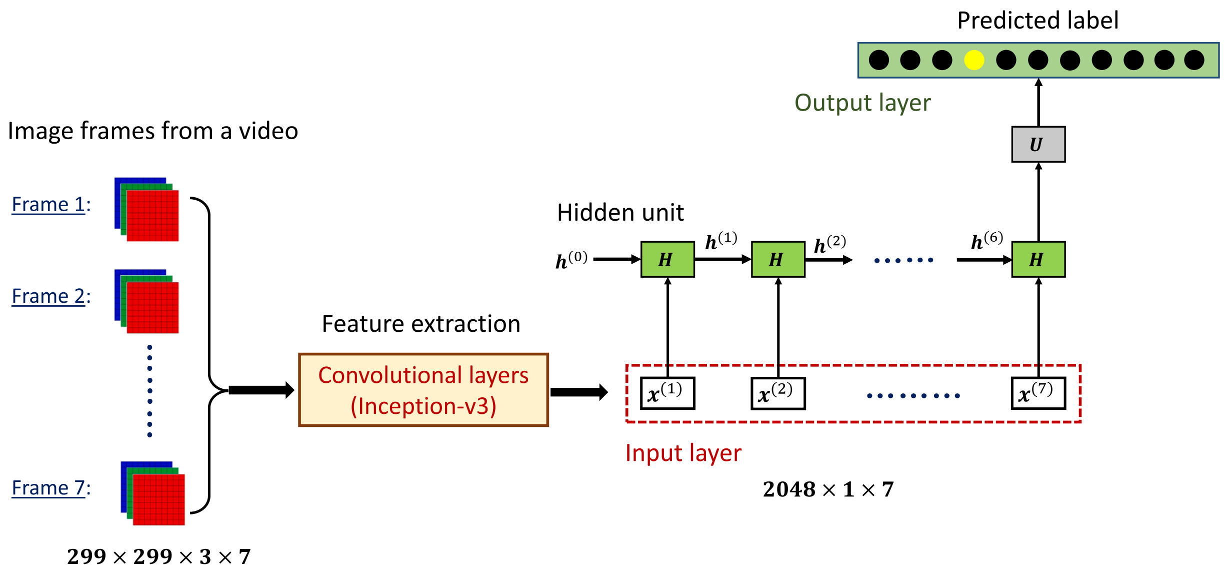


Fig. 2. Architecture of the long-short-term-memory (LSTM) recurrent neural network for the classification of the UCF-11 sports video dataset. We extracted image frames from videos at the rate of 1.4 frames per seconds and sampled 7 image frames from a video over a period of 5s. The convolutional layers of the trained convolutional neural network used the inception-v3 architecture [43]. The convolutional layers extracted features from the image frames and reduced the input space of each frame from $299 \times 299 \times 3$ to 2048×1. The extracted features for the 7 frames per video fed into the input layer of LSTM-RNN. The network used 256 hidden units. Weight matrix U connected the hidden memory to the output layer. The classifiers 11 output softmax neurons corresponded to the 11 pattern classes of videos encoded as unit bit vectors of length 11.

gives back the same BP learning laws for updating the final layer of synaptic weights.

BP invariance must also hold at each of the network's k hidden layers $\mathbf{h}_1, \ldots, \mathbf{h}_k$ for a given input $\mathbf{x}$. A deep neural network has at least two such hidden layers. We sometimes denote all or some of these hidden layers with the single symbol $\mathbf{h}$ for convenience. So the output probability at training iteration n has the form $p(\mathbf{y}|\mathbf{x},\Theta^n) = p(\mathbf{y}|\mathbf{h},\mathbf{x},\Theta^n) = p(\mathbf{y}|\mathbf{h}_k,\ldots,\mathbf{h}_1,\mathbf{x},\Theta^n)$. The layer likelihood of the network's penultimate layer $\mathbf{h}_k$ is $p(\mathbf{h}_k|\mathbf{h}_{k-1},\ldots,\mathbf{h}_1,\mathbf{x},\Theta^n)$. The layer likelihood of the first hidden layer $\mathbf{h}_1$ is just $p(\mathbf{h}_1|\mathbf{x},\Theta^n)$. The input data $\mathbf{x}$ can have its own prior $p(\mathbf{x})$ or hyper-prior. We assume here that $p(\mathbf{x}) = 1$. Then the multiplication theorem of elementary probability gives the total network likelihood $p(\mathbf{y},\mathbf{h}_k,\ldots,\mathbf{h}_1,|\mathbf{x},\Theta^n)$ as the product of the layer likelihoods [44]:

$$p(\mathbf{y},\mathbf{h}_k,\ldots,\mathbf{h}_1|\mathbf{x},\Theta^n) = p(\mathbf{y}|\mathbf{h}_k,\ldots,\mathbf{h}_1,\mathbf{x},\Theta^n) \times p(\mathbf{h}_k|\mathbf{h}_{k-1},\ldots,\mathbf{h}_1,\mathbf{x},\Theta^n) \times \cdots \times p(\mathbf{h}_1|\mathbf{x},\Theta^n). \tag{1}$$

Taking logarithms gives the network's total log-likelihood $L_{\text{total}}(\mathbf{y},\mathbf{h}|\mathbf{x},\Theta^n)$ as the sum of the $k+1$ layer log-likelihoods:

$$L_{\text{total}}(\mathbf{y},\mathbf{h}|\mathbf{x},\Theta^n) = \log p(\mathbf{y},\mathbf{h}_k,\ldots,\mathbf{h}_1|\mathbf{x},\Theta^n) \tag{2}$$

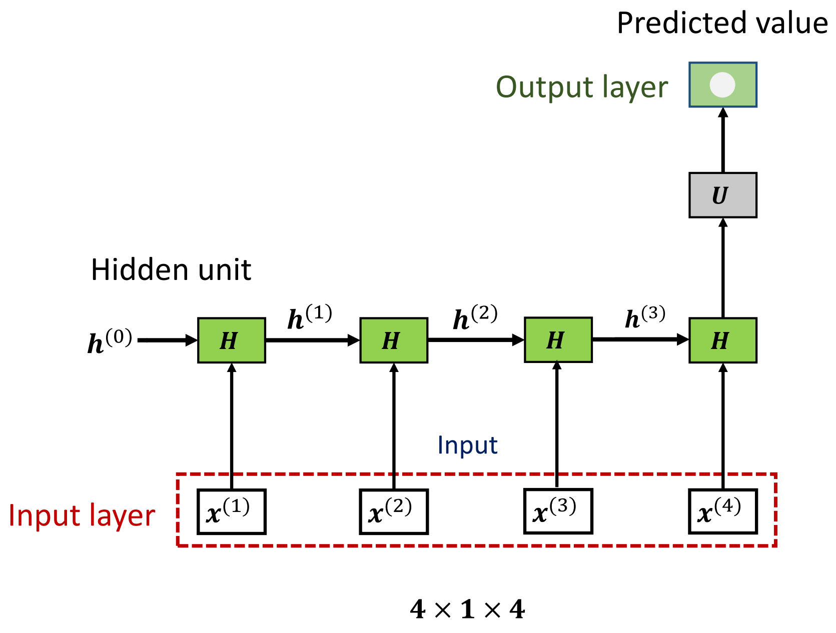


Fig. 3. Network architecture of the RNN regression models for predicting the time-series exchange rate of the US-dollar to the Indian rupee. The window size was $T = 4$ days. The trained LSTM-RNN and GRU-RNN regression models predicted the average exchange rate for a given date given the open price, high price, low price, and the average price for the previous 4 days.

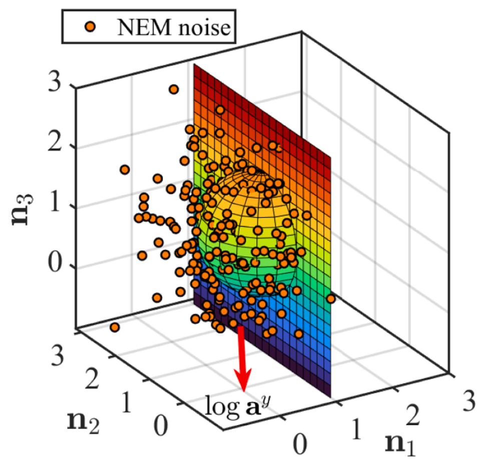


Fig. 4. Multiplicative NEM noise injection into the output softmax neurons of a recurrent neural classifier. The noise samples on the left side of the NEM hyperplane satisfied the NEM positivity condition in (23). The output activation vector was $\mathbf{a}^y = [0.6, 0.3, 0.1]$ with target vector $\mathbf{y} = [1.0, 0.0, 0.0]$. The NEM noise came from picking random samples from a multivariate Gaussian probability density with mean vector $\mathbf{1}$, identity covariance $\mathbf{I}$, and that satisfied the NEM inequality in (41).

$$= L(\mathbf{y}|\mathbf{x}, \Theta^n) + L(\mathbf{h}_k|\mathbf{x}, \Theta^n) + \cdots + L(\mathbf{h}_1|\mathbf{x}, \Theta^n) \tag{3}$$

where a given layer log-likelihood depends on the inputs from all the prior layers. So the shorthand symbol $L(\mathbf{h}_k|\mathbf{x}, \Theta^n)$ stands for $\log p(\mathbf{h}_k|\mathbf{h}_{k-1}, \ldots, \mathbf{h}_1, \mathbf{x}, \Theta^n)$. We often just write L when discussing the log-likelihood at a given layer.

The right side of the gradient identity $\nabla \log p(\mathbf{y}|\mathbf{x}, \Theta^n) = \nabla Q(\Theta^n|\Theta^n)$ in (18)–(19) describes the gradient step of the generalized EM algorithm in its iterative maximum-likelihood estimation of the parameter vector Θ. This result holds at each layer of the network even though for simplicity we will just describe the EM structure from the vantage point of the output layer.

The EM algorithm grows out of a simple rearrangement of the definition of conditional probability. Suppose measurable events A and B have positive probability for some probability measure P. Suppose their intersection or joint occurrence $A \cap B$ does as well: $P(A \cap B) > 0$. Then we can always rearrange the conditional probability $P(B|A)$ from its ratio definition

$$P(B|A) = \frac{P(A \cap B)}{P(A)} \tag{4}$$

as

$$P(A) = \frac{P(A \cap B)}{P(B|A)}. \tag{5}$$

So we can rewrite an arbitrary unconditional probability $P(A)$ in terms of *any* other measurable event B in the probability space's sigma-algebra. This "EM trick" lets us invoke hidden variables at will.

The EM algorithm averages against the hidden posterior $P(B|A)$. The posterior density $P(B|A)$ conditions the hidden or unknown quantity B on the known quantity A. Take the logarithm of (5). Then take the expectation of this difference of logarithms with respect to the hidden posterior $P(B|A)$:

$$\log P(A) = \mathbb{E}_{B|A}[\log P(A \cap B)] - \mathbb{E}_{B|A}[\log P(B|A)] \tag{6}$$

$$= Q(B|A) + H(B|A). \tag{7}$$

The log-likelihood equality (7) follows because $P(A)$ does not depend on B:

$$\mathbb{E}_{B|A}[\log P(A)] = P(B|A)\log P(A) + P(B^c|A)\log P(A) \tag{8}$$

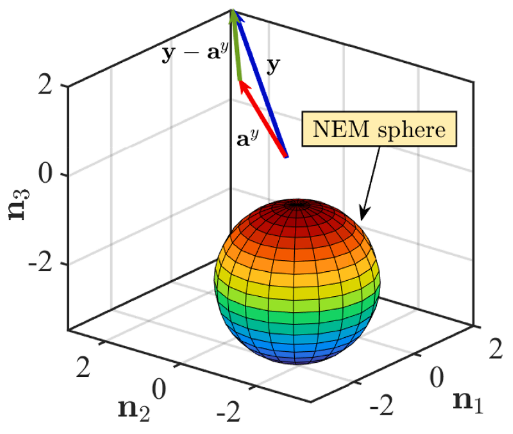


Fig. 5. Additive NEM noise injection into the output identity neurons of a recurrent neural regression model. The output activation vector was $\mathbf{a}^y = [1.0, 2.0, 1.0]$ with target vector $\mathbf{y} = [2.0, 3.0, 2.0]$. The noise samples inside the NEM sphere satisfied the NEM positivity condition. This additive NEM noise came from picking random samples from a multivariate Gaussian probability density with mean vector $\mathbf{0}$, identity covariance $\mathbf{I}$, and that satisfied the inequality in (48).

$$= \log P(A)[P(B|A) + P(B^c|A)] \tag{9}$$

$$= \log P(A) \tag{10}$$

where B^c is the set complement of B. The term $H(B|A) = \mathbb{E}_{B|A}[-\log P(B|A)]$ is just the Shannon entropy of the conditional probability $P(B|A)$.

The Q term $Q(B|A) = \mathbb{E}_{B|A}[\log P(A \cap B)]$ serves as the *surrogate* log-likelihood in iterative versions of EM when we seek to maximize the original log-likelihood $\log P(A)$. The entropy term $H(B|A)$ does not contribute to this local maximization in the so-called EM *ascent property*: Maximizing $Q(B|A)$ maximizes $\log P(A)$. We establish this ascent property below. The Q term averages the joint or *complete* log-likelihood $\log P(A \cap B)$ by conditioning on the hidden posterior $P(B|A)$. The EM algorithm's *E*-step computes or estimates the surrogate likelihood Q as it

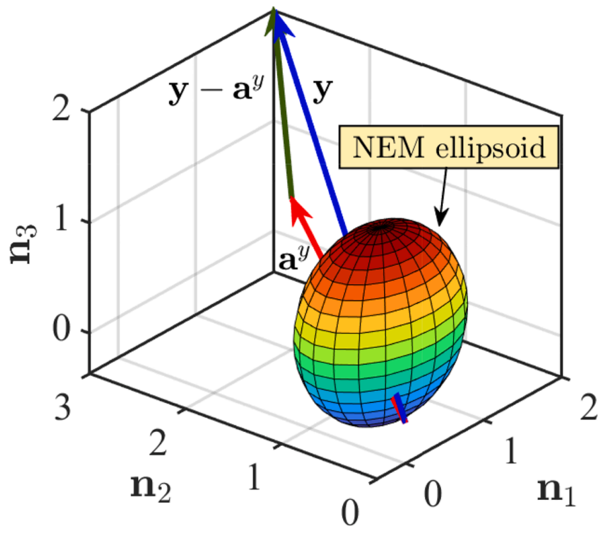


Fig. 6. Multiplicative NEM noise injection into the output identity neurons of a regression recurrent neural network. The output prediction or activation vector was $\mathbf{a}^y = [1.0, 2.0, 1.0]$ with target vector $\mathbf{y} = [2.0, 3.0, 2.0]$. The noise samples inside the NEM ellipsoid satisfied the NEM positivity condition. This additive NEM noise came from picking random samples from a multivariate Gaussian probability density with mean vector $\mathbf{1}$, identity covariance $\mathbf{I}$, and that satisfied the NEM inequality in (49).

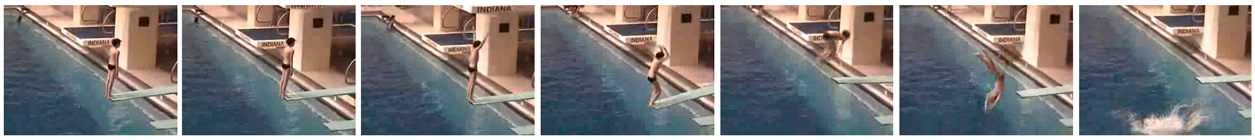
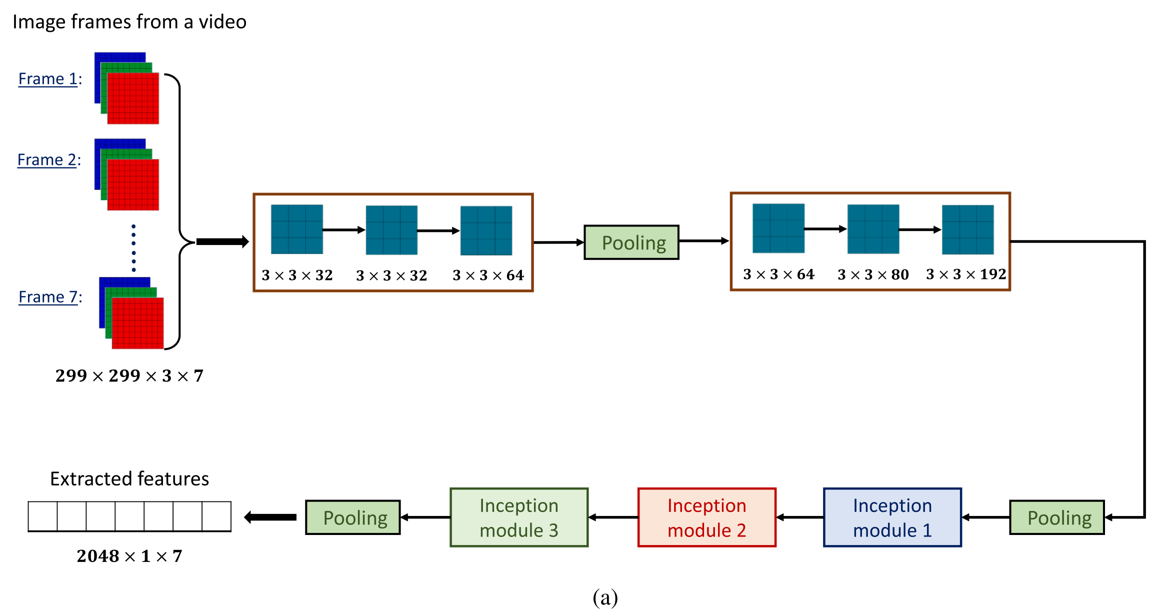
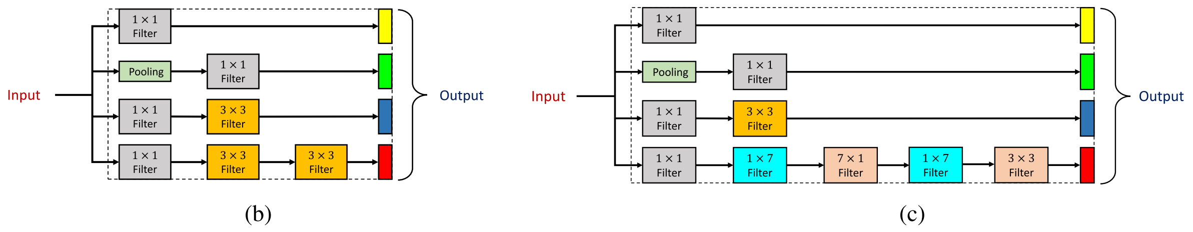


Fig. 7. Sample extracted image frames of a diver from the UCF-11 YouTube sports database. Sampling gave these diving images at a rate of 7 image frames in 5s. The diving videos were from the 3rd of 11 video pattern categories. So the target output vector for this sampled video clip was the 1-in-K-coded bit vector $[0,\ 0,\ 1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0]$.

estimates a parameter vector Θ given the current data as we explain below in the context of a neural network. Then the M-step maximizes $Q(\Theta|\Theta^n)$ over all parameter choices of Θ given the current estimate of Θ^n. Generalized EM weakens the total maximization of $Q(\Theta|\Theta^n)$ to the partial maximization of a gradient step based on $\nabla Q(\Theta|\Theta^n)$.

The EM algorithm replaces the event A with a neural-network conditional probability density $p(\mathbf{y}|\mathbf{x},\Theta)$ for network output $\mathbf{y}$ and input $\mathbf{x}$. The parameter vector Θ describes all network parameters. This includes





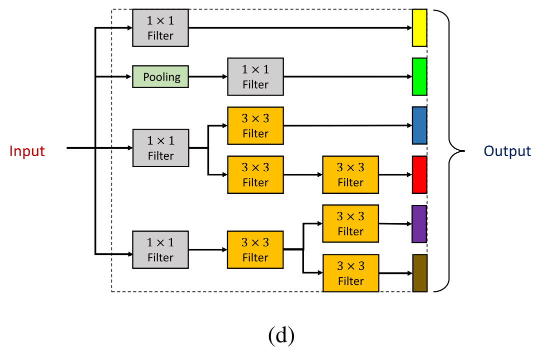


Fig. 8. Feature extraction with the convolutional layers of a trained inception-v3 network [43]: The convolutional layers consisted of convolutional masks, pooling layers, and inception modules. Inception modules arranged the convolutional masks and pooling layers to extract features from images. (a) signal flow chart for using the convolutional layers of inception-v3 network to extract features and to reduce the dimension of images. The process converted an input image of size 299 × 229 × 3 to the reduced size 2048 × 1. (b) shows the architecture of the inception module 1. (c) shows the architecture of the inception module 3. (d) shows the architecture of the inception module 2.

all synaptic weights between and among all layers of neurons. Then the EM trick brings the network's hidden variable $\mathbf{h}$ into the network probability $p(\mathbf{y}|\mathbf{x},\Theta)$:

$$p(\mathbf{y}|\mathbf{x},\Theta) = \frac{p(\mathbf{h},\mathbf{y}|\mathbf{x},\Theta)}{p(\mathbf{h}|\mathbf{y},\mathbf{x},\Theta)}. \tag{11}$$

The hidden variable $\mathbf{h}$ collects all the information about the hidden units or neurons. Then taking logarithms in (11) gives the basic log-likelihood structure of *any* neural network or any other input–output system that depends on parameters and hidden variables:

$$\log p(\mathbf{y}|\mathbf{x},\Theta) = \log p(\mathbf{h},\mathbf{y}|\mathbf{x},\Theta) - \log p(\mathbf{h}|\mathbf{y},\mathbf{x},\Theta). \tag{12}$$

EM works with a current estimate Θ^n of the network parameters at time or iteration n. So it conditions current probabilistic knowledge of the hidden units $\mathbf{h}$ on the current parameter estimate Θ^n and on the known input $\mathbf{x}$ and the output $\mathbf{y}$. The hidden posterior density $p(\mathbf{h}|\mathbf{y},\mathbf{x},\Theta^n)$ captures just this information. Taking the expectation of (12) with respect to this EM density $p(\mathbf{h}|\mathbf{y},\mathbf{x},\Theta^n)$ leads to the fundamental equality between the log-likelihood $\log p(\mathbf{y}|\mathbf{x},\Theta)$ and the surrogate likelihood $Q(\Theta|\Theta^n)$ and the entropy $H(\Theta|\Theta^n)$:

$$\log p(\mathbf{y}|\mathbf{x},\Theta) = \mathbb{E}_{\mathbf{h}|\mathbf{y},\mathbf{x},\Theta^n}[\log p(\mathbf{y}|\mathbf{x},\Theta)] \tag{13}$$

$$= \mathbb{E}_{\mathbf{h}|\mathbf{y},\mathbf{x},\Theta^n}[\log p(\mathbf{h},\mathbf{y}|\mathbf{x},\Theta)] - \mathbb{E}_{\mathbf{h}|\mathbf{y},\mathbf{x},\Theta^n}[\log p(\mathbf{h}|\mathbf{y},\mathbf{x},\Theta)] \tag{14}$$

$$= Q(\Theta|\Theta^n) + H(\Theta|\Theta^n). \tag{15}$$

The E-step of EM computes or estimates the surrogate likelihood $Q(\Theta|\Theta^n)$ at iteration n. This computation can involve sample-average Monte Carlo or other approximation techniques because $Q(\Theta|\Theta^n)$ is an ensemble expectation. Then the M-step maximizes $Q(\Theta|\Theta^n)$ over all Θ values to give the next parameter estimate $\Theta^{n+1}: \Theta^{n+1} = \text{argmax}_\Theta Q(\Theta|\Theta^n)$. These two steps repeat until the algorithm converges at or near the peak of the nearest hill of probability or log-likelihood. Users often run several EM simulations from different random starting points on the log-likelihood surface and then keep the solution that corresponds to the highest probability peak.

EM's ascent property ensures that the sequence of parameter estimates $\Theta_0, \Theta_1, \Theta_2, \ldots$ will converge to a local maximum-likelihood parameter estimate Θ^*. This convergence occurs even though the algorithm updates only the surrogate likelihood function $Q(\Theta|\Theta^n)$ and not the likelihood function itself [45]. We now sketch the proof of the ascent property both for its own sake and because of its close relationship to the reduction of backpropagation to generalized EM in Theorem 1.

The EM ascent property depends on the fact that Shannon entropy minimizes cross entropy: $H(\Theta^n|\Theta^n) \leqslant H(\Theta|\Theta^n)$ for all parameter vectors Θ. This entropy inequality follows in turn from Jensen's inequality and the concavity of the logarithm. Then the entropy inequality and (13)–(15) imply that

$$\log p(\mathbf{y}|\mathbf{x},\Theta) - \log p(\mathbf{y}|\mathbf{x},\Theta^n) \geqslant Q(\Theta|\Theta^n) - Q(\Theta^n|\Theta^n) \geqslant 0 \tag{16}$$

for the Q-maximizing choice $\Theta = \Theta^{n+1}$. This gives the EM ascent property because the log-likelihood can only increase at each iteration:

$$\log p(\mathbf{y}|\mathbf{x},\Theta^{n+1}) \geqslant \log p(\mathbf{y}|\mathbf{x},\Theta^n) \tag{17}$$

when maximizing the surrogate likelihood: $Q(\Theta^{n+1}|\Theta^n) \geqslant Q(\Theta|\Theta^n)$.

The same entropy-based argument shows why backpropagation is a special case of *generalized* EM or GEM. GEM replaces the complete maximization in the M-step with the *partial* maximization of taking a gradient step up the log-likelihood surface. But the gradient of the entropy H must be null at the Shannon-entropy minimum for a differentiable entropy $H: \nabla H(\Theta^n|\Theta^n) = \mathbf{0}$. This gives the master equation $\nabla \log p = \nabla Q$ for the backpropagation noise-boosting that follows:

$$\nabla \log p(\mathbf{y}|\mathbf{x},\Theta^n) = \nabla Q(\Theta^n|\Theta^n) + \nabla H(\Theta^n|\Theta^n) \tag{18}$$

$$= \nabla Q(\Theta^n|\Theta^n) \tag{19}$$

since Shannon entropy minimizes cross entropy. We here restate this recent result [15] as Theorem 1. We point out again that BP invariance requires that this gradient log-likelihood identity hold at the output layer and at each hidden layer.

Theorem 1. Backpropagation as Generalized Expectation Maximization

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

$$\Theta^{n+1} = \Theta^n + \eta \nabla_\Theta \log p(\mathbf{y}|\mathbf{x},\Theta)\big|_{\Theta=\Theta^n} \tag{20}$$

equals the GEM update equation at epoch n

$$\Theta^{n+1} = \Theta^n + \eta \nabla_\Theta Q(\Theta|\Theta^n)\big|_{\Theta=\Theta^n} \tag{21}$$

where GEM uses the differentiable Q-function

$$Q(\Theta|\Theta^n) = \mathbb{E}_{p(\mathbf{h}|\mathbf{y},\mathbf{x},\Theta^n)}[\log p(\mathbf{h},\mathbf{y}|\mathbf{x},\Theta)] \tag{22}$$

and η is the learning rate. The master gradient equation $\nabla \log p(\mathbf{y}|\mathbf{x},\Theta^n) = \nabla Q(\Theta^n|\Theta^n)$ does require comment to see the connection to backpropagation. The right-hand side $\nabla Q(\Theta^n|\Theta^n)$ describes the gradient step of the GEM algorithm. The gradient $\nabla \log p(\mathbf{y}|\mathbf{x},\Theta^n)$ on the left-hand-side depends on the network probability density $p(\mathbf{y}|\mathbf{x},\Theta^n)$. It thus depends on the network structure as we discussed above. This paper focuses on the two important cases of classification and regression. Classification picks the network probability $p(\mathbf{y}|\mathbf{x},\Theta^n)$ as a one-shot multinomial or categorical distribution for K output patterns or categories. So a pass through the network corresponds to the roll of a K-sided die. This structure arises in part from the 1-in-K encoding of the K target patterns or videos as K unit bit vectors. It also arises from the use of softmax output neurons and

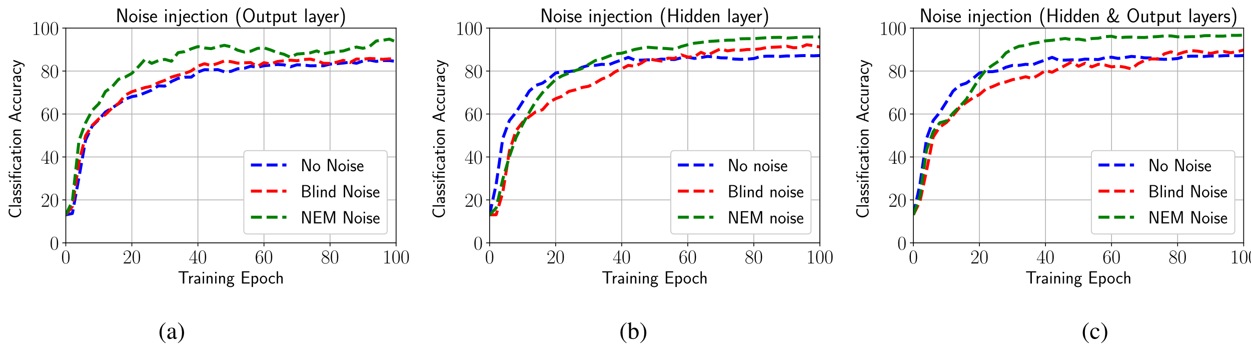


Fig. 9. Noise-boosted accuracy performance of LSTM-RNN classifiers trained on the UCF-11 sports-action YouTube video dataset. The best accuracy resulted from additive NEM-noise injection in both the output and hidden neurons. (a) Noise injection in the output layer. (b) Noise injection in the hidden units. (c) Noise injection in both the output neurons and hidden neurons.

a cross-entropy performance measure. Taking the logarithm of the one-roll multinomial probabilities gives the log-likelihood as the negative cross-entropy. So minimizing the cross-entropy maximizes the network likelihood. A direct calculation shows that the gradient of this cross-entropy gives back precisely the backpropagation learning laws [38].

The *same* backpropagation learning laws result for a regression network if the network probability $p(\mathbf{y}|\mathbf{x},\Theta^n)$ is a multidimensional Gaussian probability and the output neurons have identity activation functions. Therefore its log-likelihood is proportional to the negative summed squared error if the covariance matrix is diagonal. Then minimizing the summed squared error again maximizes the network likelihood. Different network structures will give back the same invariant backpropagation learning laws if the user picks the appropriate output activation functions and performance measures.

We show next how to noise-boost backpropagation by way of noise-boosting the EM algorithm.

3. Noise-Boosting the EM Algorithm

The Noisy EM Theorem shows that noise injection will speed up the EM algorithm on average if the noise obeys the NEM positivity condition [39,40]. The Noisy EM Theorem for additive noise states that a NEM noise benefit holds on average at each iteration n if the following positivity condition holds:

$$\mathbb{E}_{\mathbf{x},\mathbf{h},\mathbf{N}|\Theta^*}\left[\log\left(\frac{p(\mathbf{x}+\mathbf{N},\mathbf{h}|\Theta^n)}{p(\mathbf{x},\mathbf{h}|\Theta^n)}\right)\right]\geqslant 0. \tag{23}$$

Then the surrogate-likelihood EM noise benefit

$$Q(\Theta^n|\Theta^*)\leqslant Q_N(\Theta^n|\Theta^*) \tag{24}$$

holds on average at iteration n:

$$\mathbb{E}_{\mathbf{x},\mathbf{N}|\Theta^n}[Q(\Theta^n|\Theta^*)-Q_N(\Theta^n|\Theta^*)]\leqslant\mathbb{E}_{\mathbf{x}|\Theta^n}[Q(\Theta^*|\Theta^*)-Q(\Theta^n|\Theta^*)] \tag{25}$$

where Θ^* denotes the maximum-likelihood vector of parameters. The NEM positivity condition (23) has a simple form for Gaussian mixture models [46] and for classification and regression networks [15,38]. The intuition behind the NEM sufficient condition (23) is that some noise realizations $\mathbf{n}$ make a signal $\mathbf{x}$ more probable:

$$f(\mathbf{x}+\mathbf{n}|\Theta)\geqslant f(\mathbf{x}|\Theta). \tag{26}$$

Taking logarithms gives the key log-likelihood ratio:

$$\log\left(\frac{f(\mathbf{x}+\mathbf{n}|\Theta)}{f(\mathbf{x}|\Theta)}\right)\geqslant 0. \tag{27}$$

Taking expectations gives a NEM-like positivity condition. The proof of the NEM Theorem uses Kullback–Leibler 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 [40]. The proofs of Theorems 4 and 5 below appeal to the average noise-based inequality (26).

The pointwise NEM inequality (26) also helps explain the observed NEM-based accuracy improvement in the simulated recurrent neural networks below. Consider the case of a classifier neural network $N:\mathbb{R}^n\rightarrow[0,1]^K$ with the usual 1-in-K encoding for the network's K output neurons with softmax or Gibbs activation. We assign each output training vector $\mathbf{t}$ to exactly one of the K unit-bit basis vectors $[1,0,0,\dots,0]$, $[0,1,0,\dots,0]$, …, $[0,0,\dots,1]$ based on the class membership of the corresponding input pattern vector $\mathbf{x}$. So we assign the input vector $\mathbf{x}$ from the k^{th} pattern class to the k^{th} output basis vector that has a 1 in the k^{th} slot and has 0s elsewhere. The K softmax output neurons in (38) produce a discrete length-K probability vector $\mathbf{a}^y\in[0,1]^K$. The softmax output neurons give rise to an output probability $p(\mathbf{y}|\mathbf{x},\Theta)=\prod_{j=1}^{K}p_j(y_j|\mathbf{x},\Theta)$ as a one-shot multinomial probability density in accord with BP invariance. Then (26) gives on average a NEM accuracy boost:

$$p_j(y_j+n_j|\mathbf{x},\Theta)\geqslant p_j(y_j|\mathbf{x},\Theta). \tag{28}$$

We measure the classifier's accuracy on a test set of T samples by dividing the number of correct classifications (true positives) by the number T of trials. But an input $\mathbf{x}$ from category j leads to a correct classification if and only if the corresponding output activation is such that $a_j^y\geqslant a_i^y$ for all i where again the non-negative softmax activation a_k^y sum to unity for each input vector $\mathbf{x}$. So the NEM-boost (28) from a probability-increasing noise realization $N_j=n_j$ can only increase the chance that the output activation $a_k^{(y)NEM}$ wins the output competition for input $\mathbf{x}$ on average. The normalization effect of the softmax units likewise can only decrease the other $K-1$ activations on average. A NEM-based accuracy argument also follows from the likelihood structure of the NEM sufficient condition in [38].

We also observe that the NEM positivity inequality (23) is not vacuous. This holds because the expectation in (23) conditions on the converged parameter vector Θ^* rather than on the current estimated parameter vector Θ^n. Vacuity would result in the usual case of averaging a log-likelihood ratio. Take the expectation of the log-likelihood ratio $\log\frac{f(x|\Theta)}{g(x|\Theta)}$ with respect to the probability density function $g(x|\Theta)$ to give $\mathbb{E}_g\left[\log\frac{f(x|\Theta)}{g(x|\Theta)}\right]$. Then Jensen's inequality and the concavity of the logarithm imply that $\mathbb{E}_g\left[\log\frac{f(x|\Theta)}{g(x|\Theta)}\right]\leqslant\log\mathbb{E}_g\left[\frac{f(x|\Theta)}{g(x|\Theta)}\right]=\log\int_x\frac{f(x|\Theta)}{g(x|\Theta)}g(x|\Theta)dx=\log\int_x f(x|\Theta)dx=\log 1=0$. So $\mathbb{E}_g\left[\log\frac{f(x|\theta)}{g(x|\theta)}\right]\leqslant 0$ holds. But the expectation in (23) does not in general lead to this cancellation of probability densities because the integrating density in (23) depends on the optimal maximum-likelihood parameter Θ^* rather than on just Θ^n [40]. 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 [15]. A similar NEM result holds for regression except that then the noise-benefit region defines a hypersphere. NEM noise can also inject into the hidden neurons. Theorems 2–5 extend these NEM results for recurrent classifiers and regression models.

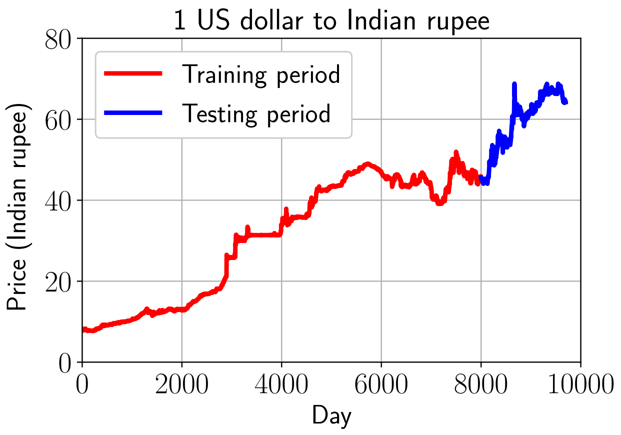


Fig. 10. Time-series data for the average daily exchange rate of the US dollar to the Indian rupee over the 9,697 business days from January 1980 to August 2017. The recurrent neural networks trained on the data from day 0 to day 8,000–from January 1980 to February 2011. We tested the RNN models on the exchange-rate data from day 8,001 to day 9,697–from March 2011 to August 2017.

4. Backpropagation Training of Recurrent Neural Networks

A recurrent neural network (RNN) has a form of controlled *feedback* in its throughput structure. This internal feedback structure endows the

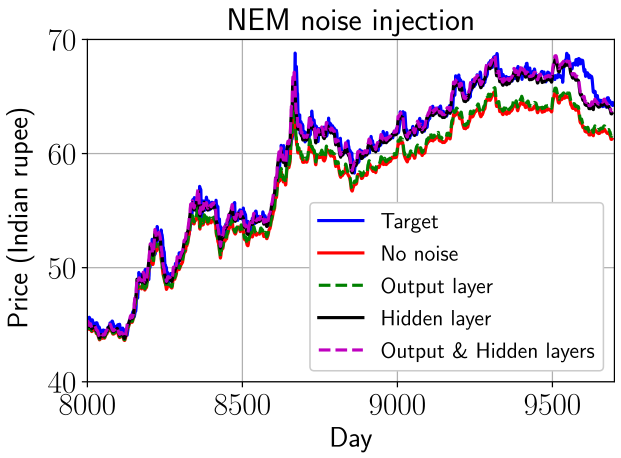


Fig. 11. LSTM-RNN predictions of the dollar-rupee exchange rate for additive NEM-noise injection in the output neurons, in the hidden neurons, and in both output and hidden neurons.

otherwise feedforward network with a dynamical structure. The feedback does not extend to the network output feeding back to become an input to the network. So the network's overall structure is still feedforward.

An RNN's internal feedback lets it process sampled time-varying data. RNNs are suitable for data processing with long time-lag dependencies [3] such as speech and large-language-model processing, video classification, and stock price prediction [10]. An RNN takes an input over a discretized time window *T*. The popular Long Short-Term Memory (LSTM) network is a deep RNN that can process time-varying data [3]. LSTM-RNNs have outperformed many other methods in areas such as natural language processing, speech recognition, and handwriting recognition [20,47,48]. A single LSTM unit consists of *I* input neurons, *J* hidden units, and *K* output neurons. LSTM uses control gates for the input, hidden, and output layers. An input gate controls the input. A forget gate controls the hidden cells. An output gate controls the output [3].

An alternative RNN architecture is the Gated Recurrent Unit (GRU) network. A GRU network resembles an LSTM network because it also uses a gating mechanism. But a GRU network does so with fewer parameters than does an LSTM network [21]. This makes GRU training faster than LSTM training. A GRU network merges the output and forget gates into a single gate. It also merges the hidden cell and memory unit into one unit. GRU performance is similar to that of LSTM in areas such as music modeling and speech recognition [22]. We develop noise-boosted versions of both LSTM and GRU-RNNs. The simplest cases inject NEM noise additively or multiplicatively into only the output neurons. More complex versions inject NEM noise into the hidden units as well as into the output neurons.

4.1. Long Short-Term Memory (LSTM) Recurrent Neural Networks

This section develops the RBP learning algorithm for an LSTM recurrent network [3]. An LSTM network consists of *I* input neurons, *J* hidden neurons, and *K* output neurons. It also uses control *gates* for the input, hidden, and output layers. The input gate controls the input activation, the forget gate controls the hidden unit, and the output gate controls the output activation. These gates each have *J* neurons. The LSTM network captures the time dependency of the input data where the time index *t* takes values in {*1, 2, ….., T*}.

The backpropagation training of an LSTM network iterates 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. The output layer has output activation $\mathbf{a}^y$. The backward pass propagates the error from the output layer back to the input layer and updates the network's weights. We alternate these two directional passes until the network's weights or parameters converge. The RBP seeks those weights that maximize the system performance or minimize the error function $E(\Theta)$.

Appendix B presents RBP training algorithm for the LSTM-RNN. Appendix B.1 shows the forward pass across an LSTM-RNN. The forward pass propagates the input $\{\mathbf{x}^{(t)}\}_{t=1}^{T}$ over the network from the input layer through the gates and hidden units to the output layer. The choice of output activation depends on the network architecture of the output layer. The output neurons uses *softmax* activation if the network is a classifier while the output activation for a regression model is an *identity* function [49,50].

Appendix B.2 presents the backward pass of RBP training with an LSTM-RNN. The backward pass shows the partial derivatives for updating the parameters of the network. The update rule for the network parameter Θ after *n* training epochs follows from the following equation:

$$\Theta^{n+1} = \Theta^{n} - \eta \nabla_{\Theta} E(\Theta)|_{\Theta=\Theta^{n}} \quad (29)$$

where η is the learning rate and Θ can be any of the network parameters. Eqs. (B.15), (B.22), (B.23), (B.26)–(B.62), (B.64)–(B.67), (B.69)–(B.72), (B.75)–(B.79), (B.81)–(B.84), (B.87)–(B.90), (B.92), (B.93) give the partial derivatives of the LSTM-RNN parameters over the backward pass. The partial derivatives form the update or learning rules for the RBP training of an LSTM-RNN. The update or learning rules for the RBP training is the same for both classifier and regression models because of BP invariance.

4.2. Gated Recurrent Unit (GRU) Recurrent Neural Networks

Gated-recurrent-unit (GRU) models offer a recent alternative to the LSTM recurrent networks [21]. GRU networks resemble LSTM because

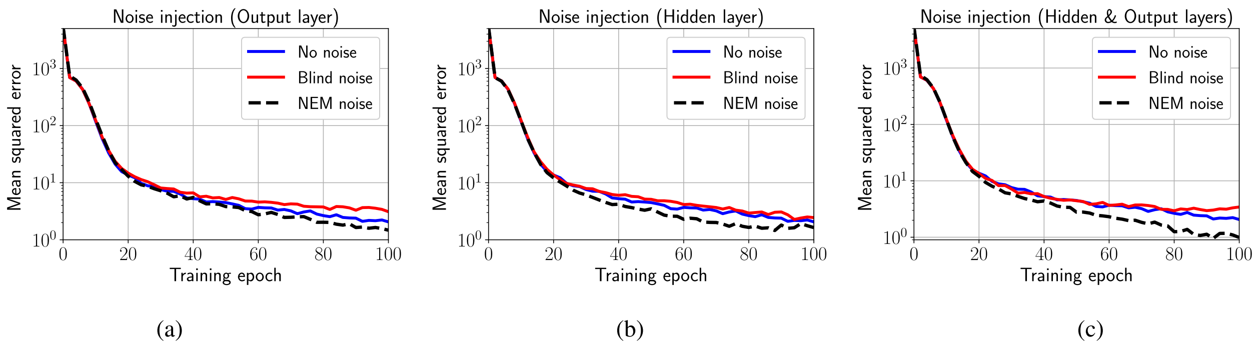


Fig. 12. Noise-boosted prediction performance of LSTM-RNN regression models trained on the dollar-rupee exchange-rate dataset. The models used *additive* noise samples. The best results used NEM-noise injection in both the output layer and hidden units. (a) Noise injection in only the output layer. (b) Noise injection only in the hidden units. (c) Noise injection in both the the output neurons and hidden neurons.

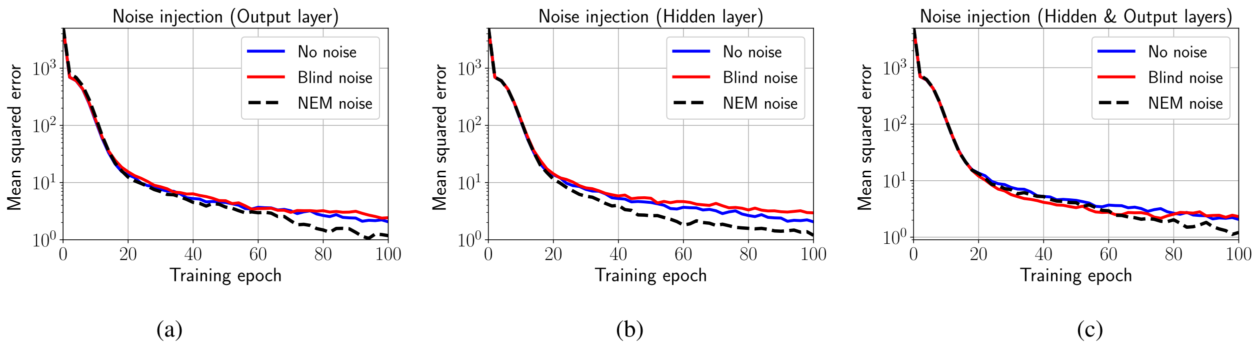


Fig. 13. Noise-boosted prediction performance of LSTM-RNN regression models trained on the dollar-rupee exchange-rate dataset. The models used *multiplicative* noise samples. The best results used NEM-noise injection in both the output layer and hidden units. (a) Noise injection in only the output layer. (b) Noise injection only in the hidden units. (c) Noise injection in both the the output neurons and hidden neurons.

they both use gating mechanisms. But GRU networks use comparatively fewer gates. GRU networks merge the hidden cell and the hidden memory into a single unit. This reduces both the number of parameters and the computational cost. GRU networks also use an update gate and a reset gate. So there is no forget gate. GRU networks combine the internal cell and hidden memory into a single unit. This further reduces the number of network parameters.

The RBP training of a GRU-RNN iterates the forward pass of the input vector and the backward pass of the error over time index t. The forward pass propagates the input vector from the input layer to the output layer over the time index. The output layer has output activation $\mathbf{a}^{y(t)}$ at time t. The backward pass propagates the error from the output layer back to the input layer and updates the network's weights. We alternate these two directional passes until the network's weights or parameters converge. RBP seeks those weights that maximize the system performance or minimize the error function $E(\Theta)$.

Appendix C gives the RBP algorithm for the GRU-RNN. Appendix C.1 gives the forward pass across a GRU-RNN. The forward pass feeds the input $\{\mathbf{x}^{(t)}\}_{t=1}^{T}$ and propagates the input through the hidden units and gates to the output layer. Similarly the output activation of the network depends on the structure of the architecture of the output layer. A classifier uses *softmax* activation and a regression model uses an *identity* activation. Appendix C.2 gives the backward pass across a GRU-RNN. The backward pass presents the partial derivatives and the update rules for training the parameters of the GRU-RNN. These partial derivatives form the update or learning rules for the RBP training of a GRU-RNN. These partial derivatives form the update rules for training a GRU-RNN with the RBP algorithm.

We turn next to noise-boosting RBP by way of BP invariance and the BP-GEM gradient identity $\nabla \log p(\mathbf{y}|\mathbf{x},\Theta^n) = \nabla Q(\Theta^n|\Theta^n)$ from (18)–(19).

5. NEM Noise Injection in Recurrent Backpropagation

We now show how to inject NEM noise into the RBP training of a recurrent neural network. The NEM noise can inject into both the output neurons and any of the hidden neurons. Extensive simulations show that classifiers tend to get the most benefit from noise injection into their output neurons. Regression models tend to get more benefit from noise injection into their hidden neurons. The sufficient condition for a NEM-noise benefit differs for classifier and regression model networks because these two types of networks have different output log-likelihoods that arise from BP invariance. The conditions are similar for additive and multiplicative noise injection.

The RBP trains RNNs so as to iteratively minimize some error function $E(\Theta)$. BP invariance implies that this is the same as iteratively maximizing the network log-likelihood L. The time-sampled version of the error function $E(\Theta)$ sums the error function's corresponding time slices

$$E(\Theta) = \sum_{t=t_0}^{T} E^{(t)} \tag{30}$$

where $E^{(t)}$ is the output error at time t and $1 \leqslant t_0 \leqslant T$. RBP uses gradient descent or any of its variants to iteratively minimize $E(\Theta)$ with respect to the weights or parameters of the RNN [42]. The output-layer time-slice error $E^{(t)}$ equals the cross entropy for a classifier network [50] and equals the squared error for a regression network at time t. The network's total output log-likelihood $L(\Theta)$ takes the logarithm of the output conditional probability $p(\mathbf{y}|\mathbf{x},\Theta)$ that itself factors in terms of both the time slices and the K output neurons where $t_0 \in \{1, \ldots., T\}$. The error function simplifies as follows:

$$E(\Theta) = \sum_{t=t_0}^{T} E^{(t)} \tag{31}$$

$$= -\sum_{t=t_0}^{T} \sum_{k=1}^{K} y_k^{(t)} \log\, a_k^{y(t)} \tag{32}$$

$$= -\sum_{t=t_0}^{T} \log\left(\prod_{k=1}^{K} p_k(y^{(t)} = y_k^{(t)}|\mathbf{x},\Theta) \right) \tag{33}$$

$$= -\sum_{t=t_0}^{T} \log\, p(\mathbf{y}^{(t)}|\mathbf{x},\Theta) \tag{34}$$

$$= -\log \prod_{t=t_0}^{T} p(\mathbf{y}^{(t)}|\mathbf{x},\Theta) \tag{35}$$

$$= -\log\, p(\mathbf{y}|\mathbf{x},\Theta) \tag{36}$$

$$= -L(\Theta). \tag{37}$$

So minimizing the error $E(\Theta)$ maximizes the log-likelihood $L(\Theta)$ for classification or regression for any other type of recurrent network whose global probability structure obeys BP invariance.

BP invariance ensures that the gradient $\nabla_\Theta L$ gives back the same BP learning laws at a given layer. The output-layer probability density function $p(\mathbf{y}^{(t)}|\mathbf{x},\Theta)$ for classification is a categorical or one-shot multinomial probability density: $\mathbf{y}^{(t)} \sim \textit{Multinomial}(\mathbf{a}^{y(t)})$. So the recurrent classifier's total conditional probability at the output layer is $p(\mathbf{y}|\mathbf{x},\Theta) =$

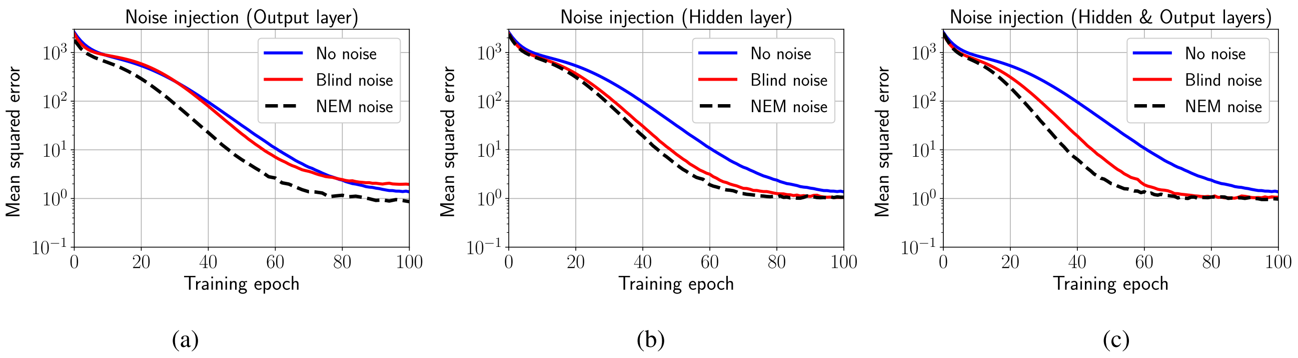


Fig. 14. Noise-boosted prediction performance of GRU-RNN regression models trained on the dollar-rupee exchange-rate dataset. The best predictions in terms of least average squared error resulted from the *Additive* NEM-noise injection in both the output and hidden neurons. (a) Additive noise injection in only the output layer. (b) Noise injection in only the hidden units. (c) Noise injection in both the output neurons and hidden neurons.

$\prod_{t=t_0}^{T}\prod_{k=1}^{K}\left(a_k^{y(t)}\right)^{y_k^{(t)}}$. The probability density function of $\mathbf{y}^{(t)}$ for regression is the vector normal density: $\mathbf{y}^{(t)} \sim \mathcal{N}(\mathbf{y}^{(t)}|\mathbf{a}^{y(t)}, \mathbf{I})$ with population mean vector $\mathbf{a}^{y(t)}$ and identity covariance matrix $\mathbf{I}$. So the recurrent regression model's total output conditional probability at the output layer is $p(\mathbf{y}|\mathbf{x},\Theta) = \prod_{t=t_0}^{T}\mathcal{N}(\mathbf{y}^{(t)}|\mathbf{a}^{y(t)}, \mathbf{I})$. This layer likelihood can also approximate the likelihood of a hidden layer of rectified-linear-unit or ReLU neurons because such activations have the truncated-identity form $\max(0,x)$. The likelihood function of a layer of logistic-sigmoid neurons has the from of a double cross entropy.

5.1. NEM Noise Injection in Recurrent Network's Output Neurons

The sufficient condition for a NEM-noise benefit in a recurrent network depends on the output log-likelihood L because of BP invariance. So it depends both on the error function and the structure of the output neurons. Noise injection in hidden layers treats such layers as modified output layers in the recursive training [38]. So the NEM sufficient condition for a given layer also depends on that layer's log-likelihood and thus in turn on that layer's error function and the activation structure of the neurons in that layer in accord with (2)–(3).

We first derive the NEM noise-benefit conditions for noise injection into the output neurons of a recurrent classifier network and a recurrent regression network. The benefit condition for additive noise injection leads at once to a similar condition for multiplicative noise injection. This result generalizes to *any* measurable combination of the signal and noise because of the corresponding general result for noise-boosting [40].

5.1.1. Noise-boosting an RBP Classifier

Recurrent neural classifiers also use Gibbs or softmax activation functions for their K output neurons:

$$a_k^{y(t)} = \frac{\exp\left(o_k^{y(t)}\right)}{\sum_{l=1}^{K}\exp\left(o_l^{y(t)}\right)} \tag{38}$$

where $a_k^{y(t)}$ is the activation of the kth output neuron and $o_k^{y(t)}$ is the input to the kth output neuron at time t. The classifier scheme also uses 1-in-K encoding for the time-varying patterns from the K pattern classes. So the ideal classifier maps a time-varying pattern from the kth pattern class to the kth basis vector $\mathbf{e}_k$ where the unit bit vector $\mathbf{e}_k$ has a 1 in the kth slot and has 0s elsewhere.

We represent a time-varying signal or pattern $\mathbf{x}$ with its T-many ordered samples or time slices. Then the recurrent classifier's error function $E(\Theta)$ sums the output cross entropy over the T time slices for a given input $\mathbf{x}$:

$$E(\Theta) = -\sum_{t=t_0}^{T}\sum_{k=1}^{K} y_k^{(t)}\log a_k^{y(t)} \tag{39}$$

where $y_k^{(t)} \in \{0,1\}$. BP invariance requires that the recurrent classifier's output conditional probability $p(\mathbf{y}|\mathbf{x},\Theta)$ is a time-factored one-shot multinomial probability $p(\mathbf{y}|\mathbf{x},\Theta) = \prod_{t=t_0}^{T}\prod_{k=1}^{K}\left(a_k^{y(t)}\right)^{y_k^{(t)}}$. Then the corresponding output log-likelihood L is just the negative of the cross entropy $E(\Theta)$ from (31)–(37) and thus $L(\Theta) = -E(\Theta)$. So RBP iteratively maximizes the log-likelihood function for a RNN classifier. We now restate the NEM theorem for an RNN classifier [42]. The theorem gives a sufficient condition for injecting beneficial additive noise in the classifier's output neurons.

Theorem 2. NEM Noise Benefit for an RNN Classifier with Additive Noise Injection into the Output Neurons

The NEM noise-benefit positivity condition (23) holds for the maximum-likelihood training of a classifier recurrent neural network with additive noise injection into its K output softmax neurons if the following hyperplane condition holds:

$$\mathbb{E}_{\mathbf{y},\mathbf{h},\mathbf{n}|\mathbf{x},\Theta^*}\left[\sum_{t=t_o}^{T}\left(\mathbf{n}^{(t)}\right)^T\log\, \mathbf{a}^{y(t)}\right] \geqslant 0 \tag{40}$$

where the additive noise $\mathbf{n}^{(t)}$ injects into the output neurons and $\mathbf{a}^{y(t)}$ is the output activation. Appendix A.1 shows the proof for Theorem 2. Fig. 1a shows the additive noise samples that inject in the output neurons of a classifier. The noise samples satisfy the inequality in (40). The noise samples below the NEM hyperplane satisfies the NEM positivity condition in (23) with respect to $\mathbf{a}^y = [0.6, 0.3, 0.1]$ and $\mathbf{y} = [1.0, 0.0, 0.0]$. The additive NEM noise samples in this case are the random samples from a multivariate Gaussian probability density function with mean vector $\mathbf{0}$, identity covariance $\mathbf{I}$, and satisfy the corresponding inequality condition in (40) with respect to $\mathbf{a}^y$ and $\mathbf{y}$. The additive NEM noise condition is not limited to Gaussian noise samples.

NEM noise can also inject multiplicatively into the output neurons of the recurrent classifier. Theorem 2 simplifies to the next corollary for multiplicative NEM noise injected into the output neurons of recurrent neural classifiers.

Corollary 1. Multiplicative NEM-Noise Injection into the Output Neurons of a Classifier

The NEM positivity condition (23) holds for the maximum-likelihood *training of a recurrent classifier neural network with multiplicative noise injection into the output softmax neurons if the following condition holds:*

$$\mathbb{E}_{\mathbf{y},\mathbf{h},\mathbf{n}|\mathbf{x},\Theta^*}\left[\sum_{t=t_o}^{T}((\mathbf{n}^{(t)}-\mathbf{1})\circ\mathbf{y}^{(t)})^T \log \mathbf{a}^{y(t)}\right]\geqslant 0 \tag{41}$$

where $\mathbf{n}^{(t)}$ is the NEM noise that multiplies the output neuron with the output activation $\mathbf{a}^{y(t)}$ and $\circ$ is the Hadamard product. The proof in this multiplicative case tracks the proof for additive NEM noise in a classifier's output neurons from Appendix A.1. The proof replaces $p(\mathbf{y}+\mathbf{n},\mathbf{h}|\mathbf{x},\Theta)$ with $p(\mathbf{y}\circ\mathbf{n},\mathbf{h}|\mathbf{x},\Theta)$ in (A.2)–(A.6) where $\mathbf{y}\circ\mathbf{n}$ denotes the pairwise multiplication of the signal vector $\mathbf{y}$ and the noise vector $\mathbf{n}$. The result gives the equivalent NEM positivity condition in (41).

Fig. 4 shows the multiplicative noise samples that inject into the output neurons of a recurrent neural classifier. The noise samples on left side of the NEM hyperplane satisfy the NEM positivity condition in (23) with respect to $\mathbf{a}^y=[0.6,0.3,0.1]$ and $\mathbf{y}=[1.0,0.0,0.0]$. The multiplicative NEM noise samples in this case are the random samples from a multivariate Gaussian density with mean vector $\mathbf{1}$, identity covariance matrix $\mathbf{I}$, and satisfy the corresponding inequality condition in (41) with respect to $\mathbf{a}^y$ and $\mathbf{y}$. The multiplicative NEM condition is not limited to Gaussian noise samples.

5.1.2. NEM Noise Injection in a Recurrent Regression Network

Regression networks approximate functions that take values in real vector spaces. A time-varying pattern defines a curve in such a space. Its sampled version defines a point in a finite product of such vector spaces.

The regression approximation entails that the network's output need not define a length-K probability vector in general. Each output neuron should be free to equal any real number. So regression networks use identity activation for output neurons. The output activations can also be linear or affine. The output activation $\mathbf{a}^{y(t)}$ for a regression network is the identity activation because it equals its own input:

$$a_k^{y(t)}=o_k^{y(t)}. \tag{42}$$

The error function is the squared error $E(\Theta)$ over all the time slices:

$$E(\Theta)=\sum_{t=t_o}^{T}||\mathbf{y}^{(t)}-\mathbf{a}^{y(t)}||^2. \tag{43}$$

The output log-likelihood L of the RNN regressor is

$$L(\Theta)=\log\prod_{t=t_o}^{T}p(\mathbf{y}^{(t)}|\mathbf{x},\Theta) \tag{44}$$

$$=\log\prod_{t=t_o}^{T}\frac{1}{(2\pi)^{\frac{K}{2}}}\exp\left(-\frac{||\mathbf{y}^{(t)}-\mathbf{a}^{y(t)}||^2}{2}\right) \tag{45}$$

$$=\log\prod_{t=t_o}^{T}\prod_{k=1}^{K}\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{|y_k^{(t)}-a_k^{y(t)}|^2}{2}\right) \tag{46}$$

$$=-\sum_{t=t_o}^{T}\left(\frac{K}{2}\log 2\pi+\frac{1}{2}||\mathbf{y}^{(t)}-a^{y(t)}||^2\right). \tag{47}$$

RBP trains the RNN regression model by iteratively maximizing the log-likelihood $L(\Theta)$ in (47). We point out again that BP invariance requires that this likelihood maximization occur successively at each hidden layer as well as at the output layer. We show next how to inject beneficial additive NEM noise into the output neurons of a RNN regression model. We then extend this hyper-spherical result to NEM noise injection into the hidden neurons.

Theorem 3. RBP Noise Benefit for a Regression RNN with Additive Noise Injection into the Output Neurons.

The NEM positivity condition (23) holds for the maximum-likelihood training of a regression recurrent neural network with Gaussian target vector $\mathbf{y}^{(t)}\sim\mathcal{N}(\mathbf{y}^{(t)}|\mathbf{a}^{y(t)},\ \mathbf{I})$ and with additive noise injection into the output identity neurons if the following hyper-spherical inequality condition holds:

$$\mathbb{E}_{\mathbf{y},\mathbf{h},\mathbf{n}|\mathbf{x},\Theta^*}\left[\sum_{t=t_0}^{T}||\mathbf{y}^{(t)}+\mathbf{n}^{(t)}-\mathbf{a}^{y(t)}||^2\right]-\mathbb{E}_{\mathbf{y},\mathbf{h},\mathbf{n}|\mathbf{x},\Theta^*}\left[\sum_{t=t_0}^{T}||\mathbf{y}^{(t)}-\mathbf{a}^{y(t)}||^2\right]\leqslant 0 \tag{48}$$

where the noise $\mathbf{n}^{(t)}$ injects additively into the output neurons and where $\mathbf{a}^{y(t)}$ is the output activation.

Appendix A.1 gives the proof of Theorem 3 and derives the hyper-spherical NEM condition.

Fig. 5 shows the additive noise samples that inject in the output neurons of a neural regression network and obey the hyper-spherical NEM-noise condition of Theorem 3. The noise samples inside the NEM sphere satisfy the NEM positivity condition in (23) with respect to $\mathbf{a}^y=[1.0,2.0,1.0]$ and $\mathbf{y}=[2.0,3.0,2.0]$. The additive NEM noise samples in this case come from random samples from a multivariate Gaussian density with mean vector $\mathbf{0}$, identity covariance $\mathbf{I}$, and satisfy the corresponding inequality condition in (48) with respect to $\mathbf{a}^y$ and $\mathbf{y}$. We point out again that the additive NEM condition is not limited to Gaussian noise samples.

Multiplicative NEM noise can also inject into the output neurons of the recurrent regression model. Theorem 3 implies the next corollary that shows how to inject multiplicative NEM noise into the output neurons of a recurrent regression network.

Corollary 2. Multiplicative NEM Noise Injection into the Output Neurons of a Regression RNN.

The NEM positivity condition (23) holds for the maximum-likelihood training of a regression recurrent neural network with Gaussian target vector $\mathbf{y}^{(t)}\sim\mathcal{N}(\mathbf{y}^{(t)}|\mathbf{a}^{y(t)},\ \mathbf{I})$ and multiplicative noise injection into the output neurons if the following inequality condition holds:

$$\mathbb{E}_{\mathbf{y},\mathbf{h},\mathbf{n}|\mathbf{x},\Theta^*}\left[\sum_{t=t_0}^{T}(2\mathbf{y}^{(t)}+2\mathbf{n}^{(t)}-\mathbf{a}^{y(t)})^T\mathbf{n}^{(t)}\right]\leqslant 0 \tag{49}$$

where $\mathbf{n}^{(t)}$ is the multiplicative noise injected into the output neurons and $\mathbf{a}^{y(t)}$ is the output activation at time t. The proof tracks the proof for injecting additive NEM noise in a regression network from Appendix A.1. We just replace $p(\mathbf{y}+\mathbf{n},\mathbf{h}|\mathbf{x},\Theta)$ with $p(\mathbf{y}\circ\mathbf{n},\mathbf{h}|\mathbf{x},\Theta)$ in (A.16)–(A.20). The result reduces the NEM positivity condition to the inequality in (49). Fig. 6 shows the multiplicative noise samples that inject into the output neurons of a regression recurrent neural network. The noise samples inside the NEM ellipsoid satisfies the inequality condition in (49) with respect to $\mathbf{a}^y=[1.0,2.0,1.0]$ and $\mathbf{y}=[2.0,\ 3.0,\ 2.0]$. The multiplicative NEM noise samples come from picking random samples from a multivariate Gaussian density with mean vector $\mathbf{0}$, identity covariance $\mathbf{I}$, and satisfy the corresponding inequality in (49) with respect to $\mathbf{a}^y$ and $\mathbf{y}$. The definition of the multiplicative NEM condition is not restricted to Gaussian distribution samples.

5.2. NEM Noise Injection into a RNN's Hidden Layers

We show next how to inject beneficial NEM noise into the hidden units of a RNN. This noise injection applies to both the LSTM and GRU-RNNs. The proof technique extends that in [38]. The RBP algorithm trains RNNs by iteratively maximizing the log-likelihood function $L(\Theta)$. Theorem 1 above states that the backpropagation is a form of generalized expectation maximization. Then maximizing the log-likelihood $\log p(\mathbf{y}|\mathbf{x},\Theta)$ maximizes the differentiable Q-function

$$Q(\Theta|\Theta^n)=\mathbb{E}_{p(\mathbf{z}|\mathbf{y},\mathbf{x},\Theta^n)}[\log p(\mathbf{z},\mathbf{y}|\mathbf{x},\Theta)] \tag{50}$$

where $\mathbf{z}$ denotes the hidden or latent variables.

RBP depends on the time index t. The RBP log-likelihood $L(\Theta)$ equals the product of time-indexed log-likelihood functions. Then (37) shows that the log-likelihood $L(\Theta)$ for RBP and the update rule for the n^{th} training epoch with RBP and learning rate η is

$$\Theta^{n+1} = \Theta^n + \eta \nabla_\Theta L(\Theta)|_{\Theta=\Theta^n} \tag{51}$$

$$= \Theta^n + \eta \nabla_\Theta \sum_{t=t_0}^{T} L(\Theta)^{(t)}|_{\Theta=\Theta^n} \tag{52}$$

$$= \Theta^n + \eta \sum_{t=t_0}^{T} \nabla_\Theta \log p(\mathbf{y}^{(t)}|\mathbf{x},\Theta)|_{\Theta=\Theta^n} \tag{53}$$

$$= \Theta^n + \eta \sum_{t=t_0}^{T} \nabla_\Theta Q(\Theta|\Theta^n)^{(t)}|_{\Theta=\Theta^n} \tag{54}$$

because Theorem 1 shows that

$$\nabla_\Theta \log p(\mathbf{y}^{(t)}|\mathbf{x},\Theta)|_{\Theta=\Theta^n} = \nabla_\Theta Q(\Theta|\Theta^n)^{(t)}|_{\Theta=\Theta^n}. \tag{55}$$

The surrogate likelihood Q-function for time t has the form

$$Q(\Theta|\Theta^n)^{(t)} = \sum_{\mathbf{z}^{(t)}} p(\mathbf{z}^{(t)}|\mathbf{y}^{(t)},\mathbf{x},\Theta^n)\log p(\mathbf{z}^{(t)},\mathbf{y}^{(t)}|\mathbf{x},\Theta) \tag{56}$$

where $\mathbf{z}^{(t)}$ is the latent variable for the hidden units at time t. The computational cost of computing $p(\mathbf{z}^{(t)}|\mathbf{y}^{(t)},\mathbf{x},\Theta^n)$ is high because it requires $T \times 2^J$ samples over the entire time window T. We can use Monte Carlo importance sampling to approximate $Q(\Theta|\Theta^n)^{(t)}$ as in [38]. This approximation assumes Bernoulli random variable for latent variable $\mathbf{z}^{(t)}$. This means that $z_j^{(t)}$ can take up either 0 or 1 and the conditional probabilities are

$$p(z_j^{(t)} = 1|\mathbf{x},\Theta) = a_j^{z(t)} \tag{57}$$

$$p(z_j^{(t)} = 0|\mathbf{x},\Theta) = 1 - a_j^{z(t)} \tag{58}$$

where $a_j^{z(t)}$ is the activation of the j^{th} neuron in the hidden unit $\mathbf{z}^{(t)}$. The neurons in each single unit are independent so the joint probability density function factors into product of marginals

$$p(\mathbf{z}^{(t)}|\mathbf{x},\Theta^n) = \prod_{j=1}^{J} p(z_j^{(t)}|\mathbf{x},\Theta^n) \tag{59}$$

$$= \prod_{j=1}^{J} \left(a_j^{z(t)}\right)^{z_j^{(t)}} \left(1 - a_j^{z(t)}\right)^{1-z(t)_j} \tag{60}$$

where $z_j^{(t)} \in \{0,1\}$. The conditional probability $p(\mathbf{z}^{(t)}|\mathbf{y}^{(t)},x,\Theta^n)$ for the hidden unit $\mathbf{z}^{(t)}$ is as follows:

$$p(\mathbf{z}^{(t)}|\mathbf{y}^{(t)},\mathbf{x},\Theta^n) = \frac{p(\mathbf{z}^{(t)},\mathbf{y}^{(t)}|\mathbf{x},\Theta^n)}{p(\mathbf{y}^{(t)}|\mathbf{x},\Theta^n)} \tag{61}$$

$$= \frac{p(\mathbf{y}^{(t)}|\mathbf{z}^{(t)},\mathbf{x},\Theta^n)p(\mathbf{z}^{(t)}|\mathbf{x},\Theta^n)}{\sum_{\mathbf{z}^{(t)}} p(\mathbf{y}^{(t)}|\mathbf{z}^{(t)},\mathbf{x},\Theta^n)p(\mathbf{z}^{(t)}|\mathbf{x},\Theta^n)}. \tag{62}$$

Monte Carlo can approximate $p(\mathbf{z}^{(t)}|\mathbf{x},\Theta^n)$ with M independent and identically distributed Bernoulli samples. Then the approximation converges almost surely to the desired probability density function $p(\mathbf{z}^{(t)}|\mathbf{x},\Theta^n)$ in accord with the strong law of large numbers because the approximation is just a sample mean or a random sample [51]:

$$p(\mathbf{z}^{(t)}|\mathbf{x},\Theta^n) \approx \frac{1}{M}\sum_{m=1}^{M} \delta_K(\mathbf{z}^{(t)} - \mathbf{z}^{m(t)}) \tag{63}$$

where δ_K is the J-dimensional Kronecker delta and where $\mathbf{z}^{m(t)}$ is the m^{th} sample at time t. We approximate $p(\mathbf{z}^{(t)}|\mathbf{y}^{(t)},\mathbf{x},\Theta^n)$ by using the Monte Carlo estimator of $p(\mathbf{z}^{(t)}|\mathbf{x},\Theta^n)$ in (62). The importance-sampling approximation for the conditional probability density function is

$$p(\mathbf{z}^{(t)}|\mathbf{y}^{(t)},\mathbf{x},\Theta^n) \approx \frac{\sum_{m=1}^{M} \delta_K(\mathbf{z}^{(t)} - \mathbf{z}^{m(t)})p(\mathbf{y}^{(t)}|\mathbf{z}^{(t)},\mathbf{x},\Theta^n)}{\sum_{m_1=1}^{M} p(\mathbf{y}^{(t)}|\mathbf{z}^{m_1(t)},\mathbf{x},\Theta^n)} \tag{64}$$

$$= \sum_{m=1}^{M} \delta_K(\mathbf{z}^{(t)} - \mathbf{z}^{m(t)})\gamma^{m(t)} \tag{65}$$

where the parameter $\gamma^{m(t)}$ is

$$\gamma^{m(t)} = \frac{p(\mathbf{y}^{(t)}|\mathbf{z}^{m(t)},\mathbf{x},\Theta^n)}{\sum_{m=1}^{M} p(\mathbf{y}^{(t)}|\mathbf{z}^{m(t)},\mathbf{x},\Theta^n)} \tag{66}$$

and $\mathbf{z}^{m(t)}$ is the m^{th} samples for latent variable $\mathbf{z}$ at time t. The term $\gamma^{m(t)}$ measures the relative importance of sample $\mathbf{z}^{m(t)}$ with respect to other samples. The Monte Carlo approximation for $Q(\Theta|\Theta^n)^{(t)}$ is

$$Q(\Theta|\Theta^n)^{(t)} \approx \sum_{\mathbf{z}^{(t)}}\sum_{m=1}^{M}\left(\gamma^{m(t)}\delta_K(\mathbf{z}^{(t)} - \mathbf{z}^{m(t)}) \times \left[\log p(\mathbf{z}^{(t)}|\mathbf{x},\Theta) + \log p(\mathbf{y}^{(t)}|\mathbf{z}^{(t)},\mathbf{x},\Theta)\right]\right) \tag{67}$$

$$\approx \sum_{m=1}^{M} \gamma^{m(t)}\left[\log p(\mathbf{z}^{m(t)}|\mathbf{x},\Theta) + \log p(\mathbf{y}^{(t)}|\mathbf{z}^{m(t)},\mathbf{x},\Theta)\right]. \tag{68}$$

The Monte Carlo approximation for the noisy surrogate likelihood $Q_N(\Theta|\Theta^n)$ is

$$Q_N(\Theta|\Theta^n)^{(t)} \approx \sum_{\mathbf{z}^{(t)}}\sum_{m=1}^{M}\left(\gamma^{m(t)}\delta_K(\mathbf{z}^{(t)} - \mathbf{z}^{m(t)}) \times \left[\log p(\mathbf{y}^{(t)}|\mathbf{z}^{(t)},\mathbf{x},\Theta) + \log p(\mathbf{z}^{(t)} + \mathbf{n}^{(t)}|\mathbf{x},\Theta)\right]\right) \tag{69}$$

$$\approx \sum_{m=1}^{M} \gamma^{m(t)}\left[\log p(\mathbf{z}^{m(t)} + \mathbf{n}^{(t)}|\mathbf{x},\Theta) + \log p(\mathbf{y}^{(t)}|\mathbf{z}^{m(t)},\mathbf{x},\Theta)\right]. \tag{70}$$

We saw above that the log-likelihood factors as follows: $\log p(\mathbf{y}|\mathbf{x},\Theta) = Q(\Theta|\Theta^n) + H(\Theta|\Theta^n)$ for the entropy term $H(\Theta|\Theta^n) = -\mathbb{E}_{\mathbf{z}|\mathbf{y},\mathbf{x},\Theta^n}[\log p(\mathbf{z}|\mathbf{y},\mathbf{x},\Theta)]$. Then the log-likelihood $L(\Theta)$ for the RNN is

$$L(\Theta) = \sum_{t=t_0}^{T} \log p(\mathbf{y}^{(t)}|\mathbf{x},\Theta) \tag{71}$$

$$= \sum_{t=t_0}^{T}\left(Q(\Theta|\Theta^n)^{(t)} + H(\Theta|\Theta^n)^{(t)}\right) \tag{72}$$

$$= Q(\Theta|\Theta^n) + H(\Theta|\Theta^n) \tag{73}$$

where $Q(\Theta|\Theta^n)$ sums the time-indexed surrogate likelihoods $Q(\Theta|\Theta^n)^{(t)}$ over $1 \leqslant t_0 \leqslant T$. We next derive sufficient conditions for NEM noise injection in the hidden units of LSTM and GRU recurrent networks.

5.2.1. *NEM Noise Injection into the Hidden Units of a Long Short-Term Memory (LSTM) RNN*

The next result shows how to inject NEM noise into the hidden units of an LSTM network. An LSTM-RNN uses three sigmoidal gates: the input, forget, and output gates. The hidden variable $\mathbf{z}^{(t)}$ denotes the gates. The terms $\mathbf{i}^{(t)}$, $\mathbf{f}^{(t)}$, and $\mathbf{o}^{(t)}$ represent the respective activations for input, forget and output gates. These gates are statistically independent. The terms $\mathbf{z}_i^{(t)}$, $\mathbf{z}_f^{(t)}$, and $\mathbf{z}_o^{(t)}$ are the respective latent variables for the input, forget, and output gates. The conditional probability density function $p(\mathbf{z}^{(t)}|\mathbf{x},\Theta)$ is

$$p(\mathbf{z}^{(t)}|\mathbf{x},\Theta) = p(\mathbf{z}_i^{(t)}, \mathbf{z}_f^{(t)}, \mathbf{z}_o^{(t)}|\mathbf{x},\Theta) \tag{74}$$

$$= p(\mathbf{z}_i^{(t)}|\mathbf{x},\Theta)p(\mathbf{z}_f^{(t)}|\mathbf{x},\Theta)p(\mathbf{z}_o^{(t)}|\mathbf{x},\Theta) \tag{75}$$

because of the statistical independence. The terms $\mathbf{n}_i^{(t)}, \mathbf{n}_f^{(t)}$, and $\mathbf{n}_o^{(t)}$ are the respective noise injections into the input, forget, and output gates. Then the noise-injected likelihood has the form

$$p(\mathbf{z}^{(t)} + \mathbf{n}^{(t)}|\mathbf{x},\Theta) = p(\mathbf{z}_i^{(t)} + \mathbf{n}_i^{(t)},\ \mathbf{z}_f^{(t)} + \mathbf{n}_i^{(t)},\ \mathbf{z}_o^{(t)} + \mathbf{n}_o^{(t)}|\mathbf{x},\Theta) \tag{76}$$

$$= p(\mathbf{z}_i^{(t)} + \mathbf{n}_i^{(t)}|\mathbf{x},\Theta)p(\mathbf{z}_f^{(t)} + \mathbf{n}_f^{(t)}|\mathbf{x},\Theta) \times p(\mathbf{z}_o^{(t)} + \mathbf{n}_o^{(t)}|\mathbf{x},\Theta). \tag{77}$$

Putting (56) in (51) gives the surrogate likelihood $Q(\Theta|\Theta^n)^{(t)}$ for an LSTM-RNN. Putting (57) in (52) gives the noisy surrogate likelihood $Q_N(\Theta|\Theta^n)^{(t)}$ for an LSTM-RNN. These results lead to the next theorem.

Theorem 4. Injecting NEM Noise into the Hidden Neurons of an LSTM-RNN.

A NEM noise benefit holds for the iterative maximum-likelihood training of an LSTM-RNN with additive noise injection into the hidden units (gates) if the following positivity condition holds:

$$\mathbb{E}_{\mathbf{z},\mathbf{n}|\mathbf{y},\mathbf{x},\Theta^*}\left[\log\left(\frac{p(\mathbf{z}+\mathbf{n}|\mathbf{y},\mathbf{x},\Theta)}{p(\mathbf{z}|\mathbf{y},\mathbf{x},\Theta)}\right)\right] \geqslant 0. \tag{78}$$

This inequality reduces to

$$\mathbb{E}_{\mathbf{z},\mathbf{n}|\mathbf{y},\mathbf{x},\Theta^*}\left[\sum_{t=t_0}^{T}\left(\left(\mathbf{n}_i^{(t)}\right)^T \log\frac{\mathbf{i}^{(t)}}{1-\mathbf{i}^{(t)}} + \left(\mathbf{n}_f^{(t)}\right)^T \log\frac{\mathbf{f}^{(t)}}{1-\mathbf{f}^{(t)}} + \left(\mathbf{n}_o^{(t)}\right)^T \log\frac{\mathbf{o}^{(t)}}{1-\mathbf{o}^{(t)}}\right)\right] \geqslant 0 \tag{79}$$

where the noises $\mathbf{n}_i^{(t)}, \mathbf{n}_f^{(t)}$, and $\mathbf{n}_o^{(t)}$ inject additively into the respective input, forget, and output gates at time t. The terms $\mathbf{i}^{(t)}, \mathbf{f}^{(t)}$, and $\mathbf{o}^{(t)}$ denote the activations for the respective input, forget, and output gates. Appendix A.2 presents the proof for Theorem 4. The NEM noise can also inject multiplicatively into the hidden units (gates) of the LSTM-RNN. Theorem 4 simplifies to the following corollary with multiplicative NEM noise into the gates of the LSTM-RNN.

Corollary 3. Multiplicative NEM Noise Injection into the Hidden Neurons of an LSTM-RNN.

A NEM noise benefit holds for iterative maximum-likelihood training of an LSTM-RNN with multiplicative noise injection into the hidden units (gates) if the following positivity condition holds:

$$\mathbb{E}_{\mathbf{z},\mathbf{n}|\mathbf{y},\mathbf{x},\Theta^*}\left[\log\left(\frac{p(\mathbf{z}\circ\mathbf{n}|\mathbf{y},\mathbf{x},\Theta)}{p(\mathbf{z}|\mathbf{y},\mathbf{x},\Theta)}\right)\right] \geqslant 0. \tag{80}$$

This inequality reduces to

$$\mathbb{E}_{\mathbf{z},\mathbf{n}|\mathbf{y},\mathbf{x},\Theta^*}\left[\sum_{t=t_0}^{T}\left(\left(\mathbf{z}_i^{(t)}\circ\mathbf{n}_i^{(t)} - \mathbf{z}_i^{(t)}\right)^T \log\frac{\mathbf{i}^{(t)}}{1-\mathbf{i}^{(t)}} + \left(\mathbf{z}_f^{(t)}\circ\mathbf{n}_f^{(t)} - \mathbf{z}_f^{(t)}\right)^T \log\frac{\mathbf{f}^{(t)}}{1-\mathbf{f}^{(t)}} + \left(\mathbf{z}_o^{(t)}\circ\mathbf{n}_o^{(t)} - \mathbf{z}_o^{(t)}\right)^T \log\frac{\mathbf{o}^{(t)}}{1-\mathbf{o}^{(t)}}\right)\right] \geqslant 0 \tag{81}$$

where $\circ$ is the Hadamard product, the noises $\mathbf{n}_i^{(t)}, \mathbf{n}_f^{(t)}$, and $\mathbf{n}_o^{(t)}$ inject multiplicatively into the respective input, forget, output gates at time t. The terms $\mathbf{i}^{(t)}, \mathbf{f}^{(t)}$, and $\mathbf{o}^{(t)}$ are the activations for the respective input, forget, and output gates. The proof for Corollary 3 follows from replacing the additive noise with a multiplicative version in Appendix A.2. The proof replaces $p(\mathbf{z}+\mathbf{n},\mathbf{h}|\mathbf{x},\Theta)$ with $p(\mathbf{z}\circ\mathbf{n},\mathbf{h}|\mathbf{x},\Theta)$ in (A.26)–(A.32). The result gives the positivity condition in (81).

5.2.2. NEM Noise Injection into the Hidden Units of a Gated-Recurrent-Unit GRU-RNN

Noise injection into the hidden units of a GRU-RNN has the same form as in the case of an LSTM-RNN. The GRU gating mechanism differs from that of the LSTM because the GRU uses two gates (update and reset) instead of three gates. The terms $\mathbf{d}^{(t)}$ and $\mathbf{r}^{(t)}$, and represent the respective activations for update and reset gates. The terms $\mathbf{z}_d^{(t)}$, and $\mathbf{z}_r^{(t)}$ are the respective latent variables for the update and reset gates. These gates are also statistically independent. Then the likelihood $p(\mathbf{z}^{(t)}|\mathbf{y},\mathbf{x},\Theta)$ has the factored form

$$p(\mathbf{z}^{(t)}|\mathbf{y},\mathbf{x},\Theta) = p(\mathbf{z}_d^{(t)},\ \mathbf{z}_r^{(t)}|\mathbf{y},\mathbf{x},\Theta) \tag{82}$$

$$= p(\mathbf{z}_d^{(t)}|\mathbf{y},\mathbf{x},\Theta)p(\mathbf{z}_r^{(t)}|\mathbf{y},\mathbf{x},\Theta). \tag{83}$$

The terms $\mathbf{n}_d^{(t)}$ and $\mathbf{n}_r^{(t)}$ are the respective noise injections into the update and reset gates. The noise likelihood $p(\mathbf{z}^{(t)} + \mathbf{n}^{(t)}|\mathbf{y},\mathbf{x},\Theta)$ has the related factored form

$$p(\mathbf{z}^{(t)} + \mathbf{n}^{(t)}|\mathbf{y},\mathbf{x},\Theta) = p(\mathbf{z}_d^{(t)} + \mathbf{n}_d^{(t)},\ \mathbf{z}_r^{(t)} + \mathbf{n}_r^{(t)}|\mathbf{y},\mathbf{x},\Theta) \tag{84}$$

$$= p(\mathbf{z}_d^{(t)} + \mathbf{n}_d^{(t)}|\mathbf{y},\mathbf{x},\Theta)p(\mathbf{z}_r^{(t)} + \mathbf{n}_r^{(t)}|\mathbf{y},\mathbf{x},\Theta). \tag{85}$$

Theorem 5. Injecting NEM Noise into the Hidden Neurons a GRU-RNN.

A NEM noise benefit holds for the iterative maximum-likelihood training of a GRU-RNN with noise injection into the hidden units (gates) if the following positivity condition holds:

$$\mathbb{E}_{\mathbf{z},\mathbf{n}|\mathbf{y},\mathbf{x},\Theta^*}\left[\log\left(\frac{p(\mathbf{z}+\mathbf{n}|\mathbf{y},\mathbf{x},\Theta)}{p(\mathbf{z}|\mathbf{y},\mathbf{x},\Theta)}\right)\right] \geqslant 0. \tag{86}$$

The inequality reduces to

$$\mathbb{E}_{\mathbf{z},\mathbf{n}|\mathbf{y},\mathbf{x},\Theta^*}\left[\sum_{t=t_0}^{T}\left(\left(\mathbf{n}_d^{(t)}\right)^T \log\frac{\mathbf{d}^{(t)}}{1-\mathbf{d}^{(t)}} + \left(\mathbf{n}_r^{(t)}\right)^T \log\frac{\mathbf{r}^{(t)}}{1-\mathbf{r}^{(t)}}\right)\right] \geqslant 0 \tag{87}$$

where the noises $\mathbf{n}_d^{(t)}$ and $\mathbf{n}_r^{(t)}$ inject into the respective update and reset gates. The terms $\mathbf{d}^{(t)}$ and $\mathbf{r}^{(t)}$ are the activations for the respective update and reset gates at time t. Appendix A.2 presents the proof for Theorem 5. The NEM noise can also inject multiplicatively into the hidden units (gates) of a GRU-RNN. Theorem 4 simplifies to the following corollary with multiplicative NEM noise into the gates of a GRU-RNN.

Corollary 4. Multiplicative NEM Noise Injection into the Hidden Neurons of a GRU-RNN.

A NEM noise benefit holds for the iterative maximum-likelihood training of a GRU-RNN with multiplicative noise injection into the hidden units (gates) if the following positivity condition holds:

$$\mathbb{E}_{\mathbf{z},\mathbf{n}|\mathbf{y},\mathbf{x},\Theta^*}\left[\log\left(\frac{p(\mathbf{z}\circ\mathbf{n}|\mathbf{y},\mathbf{x},\Theta)}{p(\mathbf{z}|\mathbf{y},\mathbf{x},\Theta)}\right)\right] \geqslant 0 \tag{88}$$

and this simplifies to

$$\mathbb{E}_{\mathbf{z},\mathbf{n}|\mathbf{y},\mathbf{x},\Theta^*}\left[\sum_{t=t_0}^{T}\left(\left(\mathbf{z}_d^{(t)}\circ\mathbf{n}_d^{(t)} - \mathbf{z}_d^{(t)}\right)^T \log\frac{\mathbf{d}^{(t)}}{1-\mathbf{d}^{(t)}} + \left(\mathbf{z}_r^{(t)}\circ\mathbf{n}_r^{(t)} - \mathbf{z}_r^{(t)}\right)^T \log\frac{\mathbf{r}^{(t)}}{1-\mathbf{r}^{(t)}}\right)\right] \geqslant 0 \tag{89}$$

where $\circ$ is the Hadamard product, $\mathbf{n}_d^{(t)}$ and $\mathbf{n}_r^{(t)}$ are the respective multiplicative noise injections into the update and reset gates at time t. The terms $\mathbf{d}^{(t)}$ and $\mathbf{r}^{(t)}$ are the respective activations for update and reset gates. The proof for Corollary 4 follows from replacing the additive noise with a multiplicative version in Appendix A.2. The proof replaces $p(\mathbf{z}+\mathbf{n},\mathbf{h}|\mathbf{x},\Theta)$ with

$p(\mathbf{z} \circ \mathbf{n}, \mathbf{h} | \mathbf{x}, \Theta)$ in (A.26)–(A.32). The result gives the positivity condition in (89). Algorithm 1 summarizes the process of injecting the additive NEM noise into the hidden and output neurons of a GRU classifier. This also extends to other modifications such as GRU regression model, multiplicative noise, and LSTM-RNNs with the appropriate modifications.

Algorithm 1: The NEM-RBP algorithm for a GRU classifier with additive NEM noise injection in the hidden and output neurons.

```
Data: $M$ time-dependent input vectors $\{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$, the corresponding $M$ target label
  1-in-$K$ vectors $\{\mathbf{y}_1, \ldots, \mathbf{y}_M\}$, number of training epochs $R$, time offset $t_0$, and the time
  window $T$
Result: Trained GRU classifier weights $\Theta$
While epoch $r : 1 \rightarrow R$ do
  While training data number $m : 1 \rightarrow M$ do
    • Compute the forward pass of the input data $\mathbf{x}_m = \{\mathbf{x}_m^{(t)}\}_{t=1}^{T}$ forward through the
    GRU classifier using (C.1)–(C.8)
    • Compute the corresponding output softmax activation vector $\mathbf{a}_m^y = \{\mathbf{a}_m^{y(t_0)}, \ldots,
    \mathbf{a}_m^{y(T)}\}$ using (38)
    • Noise injection in the output neurons
    For t=$t_0$,…,T do
      • Generate output noise vector $\mathbf{n}_m^{(t)}$
      if $\mathbf{n}_m^{(t)^T} \log(\mathbf{a}_m^{y(t)}) \geqslant 0$ then
        • Add NEM noise: $\mathbf{y}_m^{(t)} \leftarrow \mathbf{y}_m^{(t)} + \mathbf{n}_m^{(t)}$;
      else
        • Do nothing
    • Compute the cross entropy between $\mathbf{a}_m^y$ and $\mathbf{y}_m = \{\mathbf{y}_m^{(t)}, \ldots, \mathbf{y}_m^{(T)}\}$ using (39)
    • Back-propagate the error
    • Noise injection in the input and output gates
    For t=1,…,T do
      • Generate hidden noise vectors $\mathbf{n}_d^{m(t)}$ and $\mathbf{n}_r^{m(t)}$
      If $\mathbf{n}_d^{m(t)^T} \log \frac{\mathbf{d}_m^{(t)}}{1 - \mathbf{d}_m^{(t)}} \geqslant 0$ then
        • Add NEM noise: $\mathbf{d}_m^{(t)} \leftarrow \mathbf{d}_m^{(t)} + \mathbf{n}_d^{m(t)}$
      else
        • Do nothing
      If $\mathbf{n}_r^{(t)^T} \log \frac{\mathbf{r}_m^{(t)}}{1 - \mathbf{r}_m^{(t)}} \geqslant 0$ then
        • Add NEM noise: $\mathbf{r}_m^{(t)} \leftarrow \mathbf{r}_m^{(t)} + \mathbf{n}_r^{m(t)}$
      else
        • Do nothing
    • Update the network parameter $\Theta$ using (29)
```

6. Simulation Results

We simulated the NEM noise algorithms for RBP training on recurrent classifiers and regression models. The recurrent classifiers had either a long-short-term-memory or gated-recurrent-unit structure. The same held for the recurrent regression models. We first present the simulation results for the recurrent classifiers.

We trained recurrent neural classifiers on the standard UCF-11 sports-action YouTube video dataset [23,24]. The dataset consists of 11 categories of videos. The simulated RNNs had both long-short-term-memory (LSTM) and gated-recurrent-unit (GRU) architectures. We injected both additive and multiplicative NEM noise during the RBP training of the RNN video classifiers. Fig. 2 shows the architecture of the RNN classifiers.

We extracted 30,000 training samples from the UCF-11 videos that consist of the following 11 categories: basketball shooting, cycling, diving, golf-swing, horse riding, soccer juggling, swinging, tennis swinging, trampoline jumping, volleyball spiking, and walking. We used 5:1 data splits for training and testing. So we used 6,000 video-clip samples for testing the network after training. Each training instance had 7 image frames from a sports-action video at the rate of 1.4 frames *per* second. Each image frame in the dataset had $299 \times 299 \times 3$ pixels. So the size of each input was $299 \times 299 \times 3 \times 7$.

Convolutional filters in the RNN extracted features from the images and reduced the dimension of the input pattern space. Fig. 8 shows the structure of the convolutional layers. The convolutional layers of a trained inception-v3 network [43] extracted the features. It reduced the dimension of each image to 2048×1. This reduced the dimension of each input to $2048 \times 1 \times 7$. We fed the extracted features into the input neurons of the RNN classifier and tried different numbers of hidden neurons.

The hidden gates and memory all had the same number of neurons for each trial. The output layer had 11 softmax neurons that coded for the 11 categories of sports actions in the dataset. Fig. 7 shows 7 sampled consecutive frames from a video in the diving category.

We compared RBP training without noise, RBP training with blind noise, and RBP training with NEM noise. We also compared noise injection in the output layer only, in the hidden layers only, and in both the output and hidden neurons. We found the best performance with NEM noise injection in both the hidden and output units.

6.1. Noise-boosted LSTM-RNN classifiers

We trained noise-boosted recurrent LSTM classifiers on the UCF-11 sports-video dataset [23,24]. Table 1 shows the classification accuracy for different noise-boosted RBP training regimens.

Injecting additive blind noise had little effect on the RNN's classification accuracy. It slightly increased the classification accuracy in some cases and it reduced it in others. Total blind-noise injection in the hidden and output neurons decreased the classification accuracy by 1.15%. This accuracy change used noiseless RBP as the baseline. The noiseless case had a classification accuracy of 87.21% with LSTM-RNN.

Fig. 9 shows that additive NEM noise improved classification accuracy in all cases compared with noiseless RBP and blind-noise injection. Additive NEM noise also outperformed multiplicative NEM noise in all cases. The best performance occurred with total additive NEM-noise injection in both the output neurons and in all hidden units. This gave 96.70% classification accuracy. This was an increase of 9.51% over the noiseless classification accuracy. The extra computation involved in the NEM-noise injection was slight.

Table 2 shows that additive NEM noise injection speeded up training more than did multiplicative noise. Both gave far more pronounced speed-ups than did blind-noise injection. The best speed-up occurred with total additive NEM-noise injection in both the output neurons and in all hidden units. This total NEM-noise injection took 60 fewer training epochs to reach the baseline value than did noiseless RBP. Total multiplicative NEM-noise injection took nearly 18 fewer epochs to reach the same value.

6.2. Noise-boosted Gated-Recurrent-Unit Classifiers

We also trained noise-boosted recurrent GRU classifiers on the UCF-11 sports-video dataset [23,24]. Table 3 shows the classification accuracy for different RBP training regimens. Additive injection of NEM noise in both the hidden and output neurons increased the classification accuracy by 3.02%. Multiplicative NEM-noise injection in both the hidden and output neurons increased the accuracy by 2.25%. Injecting blind noise or dither only slightly improved the classification accuracy.

Table 4 shows that NEM noise injection also speeded up the training of the GRU RBP classifier. The best speed-up came from additive NEM-noise injection in both the hidden and output neurons. It took 28% fewer epochs than the noiseless RBP to reach the baseline accuracy value of 92.10%. The baseline value was the cross entropy value that noiseless RBP achieved on the test set after training for 100 epochs. Multiplicative NEM-noise injection in the same neurons took 21% fewer epochs than the noiseless RBP to reach the baseline value.

6.3. Noise-boosted RNN Regression Model

We tested how well RNN regression models predicted a known time series of currency exchange rates [52]. The RNN regressors trained and tested on the dollar-rupee exchange rates from January 1980 to August

Table 1
Classification accuracy of LSTM on the UCF-11 sports-action YouTube video dataset with injected blind noise and NEM noise with RBP training. The baseline accuracy of noiseless RBP was 87.21% and all training was for 100 epochs. Additive NEM noise outperformed multiplicative NEM noise. Both outperformed injecting blind noise. The best NEM-noise injection increased the accuracy by 9.51% over noiseless RBP.

Training Algorithm	Additive Noise			Multiplicative Noise		
	Output	Hidden	Output & Hidden	Output	Hidden	Output & Hidden
RBP with blind noise	85.50%	91.17%	89.61%	85.43%	86.93%	86.98%
RBP with NEM noise	**93.55%**	**95.66%**	**96.70%**	**87.30%**	**87.26%**	**87.78%**

Table 2
Training speed-up of recurrent LSTM classifiers trained on the UCF-11 sports-action YouTube video dataset for injected blind noise and NEM noise with RBP training. All training was for 100 epochs where the baseline was the noiseless RBP cross-entropy value after 100 epochs. Additive NEM noise outperformed multiplicative NEM noise. Both outperformed injecting blind noise. Additive NEM-noise injection into all output and hidden neurons gave a relative 60% speed-up in training.

Training Algorithm	Additive Noise			Multiplicative Noise		
	Output	Hidden	Output & Hidden	Output	Hidden	Output & Hidden
RBP with blind noise	−8%	26%	20%	−6%	3%	9%
RBP with NEM noise	**40%**	**56%**	**60%**	**15%**	**13%**	**18%**

2017. The dataset gives the daily average exchange rate for the US dollar to the Indian rupee over 9,697 business days. The exchange-rate data came from the currency website: https://www.investing.com/currencies/usd-inr-historical-data.

The RNNs trained on the exchange-rate data from day 0 until day 8,000—from January 1980 to February 2011. We tested the RNN regression models on the exchange-rate data from day 8,001 until day 9,697—from March 2011 to August 2017. The trained RNN regression models predicted the dollar-rupee exchange rate for the i^{th} day $\mathscr{D}^{(i)}$ given data from the 4 previous days. The inputs were the open price, high price, low price, and average price from the 4 previous days $\mathscr{D}^{(i-1)}$, $\mathscr{D}^{(i-2)}$, $\mathscr{D}^{(i-3)}$, and $\mathscr{D}^{(i-4)}$. Each RNN regression models trained over 100 epochs of RBP learning with or without noise injection. The RNNs used 50 neurons each for the hidden units and the gates.

6.3.1. Noise-boosted LSTM-RNN Regression

Simulations showed that NEM-noise boosted LSTM RBP with NEM noise outperformed ordinary noiseless RBP and outperformed RBP with injected blind noise. Fig. 12 and Table 5 show these squared-error results for the additive noise injection. The same relationships held for additive and multiplicative noise injection. Fig. 13 shows the performance for the multiplicative noise injection. The baseline squared error was the average squared error of the noiseless LSTM-RNN that had trained for 100 epochs.

Blind noise also reduced the error but not as much as NEM-noise injection did. The best exchange-rate prediction in terms of the averaged squared error came from injecting additive NEM noise in both the hidden and output neurons. This also gave the fastest convergence in training. It took fewer training epochs for the NEM-noise-boosted RNN to reach the baseline squared-error value of 2.087. Table 6 shows the training speed-up from noise injection.

RBP training with blind noise outperformed RBP without noise. This result held for noise injection in the output layer only, in the hidden layer only, and in both the hidden and output layers. Fig. 11 shows that the best result was for additive NEM injection in both the hidden and output layers. This blind-noise benefit may reflect some form of a stochastic-resonance noise benefit for threshold-like systems [25,28,31]. Tables 5 and 6 show that blind noise injection helped slightly but not as much as NEM-noise injection helped. The tables also show that the best LSTM RBP performance was with NEM noise injection in both the hidden and output layers. This held both for reducing the average squared

Table 4
Training speed-up due to noise injection in recurrent GRU classifiers trained on the UCF-11 sports-video YouTube dataset. All training was for 100 epochs. The baseline cross entropy was the cross entropy of noiseless RBP after 100 epochs.

Training Algorithm	Additive Noise			Multiplicative Noise		
	Output	Hidden	Output & Hidden	Output	Hidden	Output & Hidden
RBP with blind noise	26%	11%	16%	10 %	−10 %	8%
RBP with NEM noise	**26%**	**15%**	**28%**	**20%**	**18%**	**21%**

Table 5
Predictive accuracy of noise-injected LSTM-RNN regression models that trained on the dollar-rupee exchange-rate dataset. Each entry shows the average squared error after training for 100 epochs. The baseline squared-error value of 2.087 was the squared error of noiseless RBP after 100 training epochs. Additive NEM-noise injection in both hidden and output neurons gave the best performance.

Training Algorithm	Additive Noise			Multiplicative Noise		
	Output	Hidden	Output & Hidden	Output	Hidden	Output & Hidden
Blind noise	1.689	2.465	2.988	2.310	1.756	2.359
NEM noise	**1.490**	**1.636**	**1.059**	**1.170**	**1.191**	**1.147**

Table 3
Classification accuracy of GRU on the UCF-11 YouTube sports-video dataset for injected blind noise and NEM noise with RBP training. The baseline accuracy of noiseless RBP was 92.10%. Additive NEM noise outperformed multiplicative NEM noise. Both outperformed injecting multiplicative blind noise. The best NEM-noise result increased the accuracy by 3.02% over noiseless RBP. The models trained over 100 epochs.

Training Algorithm	Additive Noise			Multiplicative Noise		
	Output	Hidden	Output & Hidden	Output	Hidden	Output & Hidden
RBP with blind noise	93.83%	93.62%	94.93%	93.03%	90.55%	92.17%
RBP with NEM noise	**94.92%**	**94.52%**	**95.12%**	**94.23%**	**94.10%**	**94.35%**

Table 6
Training speed-up due to noise injection in LSTM-RNN regression models trained on the dollar-rupee exchange-rate dataset. All training was for 100 epochs. Additive NEM-noise injected into the hidden and output neurons gave the best performance. It reached the baseline squared-error value 2.087 in 38% fewer steps than did noiseless RBP.

Training Algorithm	Additive Noise			Multiplicative Noise		
	Output	Hidden	Output & Hidden	Output	Hidden	Output & Hidden
Blind noise	−24%	-12%	−24%	−11%	−24%	−11%
NEM noise	**20%**	**36%**	**38%**	**31%**	**30%**	**29%**

Table 7
Predictive accuracy of noise-injected GRU-RNN regression models that trained on the dollar-rupee exchange-rate dataset. Each entry shows the average squared error after training for 100 epochs. The baseline squared-error value of 1.508 was the squared error of noiseless RBP after 100 training epochs. Additive NEM-noise injection in both hidden and output neurons gave the best performance.

Training Algorithm	Additive Noise			Multiplicative Noise		
	Output	Hidden	Output & Hidden	Output	Hidden	Output & Hidden
Blind noise	1.566	1.463	1.461	1.558	1.483	1.461
NEM noise	**1.425**	**1.362**	**1.280**	**1.422**	**1.161**	**1.151**

Table 8
Training speed-up due to noise injection in GRU-RNN regression models trained on the dollar-rupee exchange-rate dataset. All training was for 100 epochs. Multiplicative NEM-noise injected into the hidden and output neurons gave the best performance. It speeded up training by 37% over noiseless RBP while additive NEM-noise injection speeded training by 32%.

Training Algorithm	Additive Noise			Multiplicative Noise		
	Output	Hidden	Output & Hidden	Output	Hidden	Output & Hidden
Blind noise	−10%	10%	4%	−10 %	18 %	22%
NEM noise	**28%**	**27%**	**32%**	**18%**	**36%**	**37%**

error and for reducing the number of training epochs that the system took to reach the baseline noiseless squared-error value.

6.3.2. *Noise-boosted GRU-RNN Regression*

This final section shows how injecting NEM noise improved the predictive accuracy of GRU-RNN regression models. Fig. 14 shows that injecting NEM noise in the GRU-RNN for regression outperformed injecting blind noise and outperformed noiseless RBP. The RBP with blind noise outperformed RBP without noise.

Fig. 14 shows that NEM-noise injection in both the GRU-RNN's hidden and output neurons outperformed injecting NEM noise in either its hidden neurons or output neurons separately. Injecting NEM noise outperformed injecting blind noise. But injecting blind noise did substantially better than injecting no noise at all. Tables 7 and 8 give the resulting squared-error values of the predictions. Table 7 shows the noise benefits in terms of the average squared errors of the predictions. Additively injecting NEM noise in both the hidden and output neurons gave the best squared-error performance for prediction. But multiplicative NEM-noise injection gave the best training speed-up to the baseline squared-error value of 1.508 of noiseless RBP after 100 training epochs. This was the only case where multiplicative NEM-noise injection outperformed additive injection.

7. Conclusion

Careful noise injection improved the convergence and the accuracy of recurrent backpropagation for both classification and regression with LSTM and GRU recurrent neural networks. Simulations confirmed the noise benefits that the corresponding noise-benefit theorems predicted will hold on average. These results extend the recent noise-boosting of the simple backpropagation algorithm. The noise-boosting itself depends on the facts that the backpropagation algorithm is a special case of the generalized Expectation–Maximization algorithm and that noise can always boost the EM algorithm on average. This NEM formulation of recurrent backpropagation confirms that the algorithm is a statistical model.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Proof of Theorems

This section presents the proofs for the theorems that state NEM noise benefit condition for noise injection into the output and hidden units of RNNs with recurrent backpropagation training.

A.1. *NEM Noise Injection into Output Neurons.*

Theorem 2. *NEM Noise Benefit for an RNN Classifier with Additive Noise Injection into the Output Neurons.*
The NEM noise-benefit positivity condition (23) holds for the maximum-likelihood training of a classifier recurrent neural network with additive noise injection into its K output softmax neurons if the following hyperplane condition holds:

$$\mathbb{E}_{\mathbf{y},\mathbf{h},\mathbf{n}|\mathbf{x},\Theta^*}\left[\sum_{t=t_o}^{T}(\mathbf{n}^{(t)})^T\log\,\mathbf{a}^{y(t)}\right]\geqslant 0 \tag{A.1}$$

where the additive noise $\mathbf{n}^{(t)}$ injects into the output neurons and $\mathbf{a}^{y(t)}$ is the output activation.

Proof. Eqs. (31) and (37) show that the RNN classifier's output cross entropy $E(\Theta)$ equals the sum of the negative log-likelihood functions over time $t\in\{t_0,\ t_0+1,\ldots,T\}$ where $t_0\in\{1,\ldots.,T\}$. So $p(\mathbf{y}|\mathbf{x},\Theta)=\exp(-E(\Theta))$. This shows again that minimizing the cross entropy $E(\Theta)$ maximizes the log-likelihood $L(\Theta)$.

The NEM sufficient condition (23) simplifies to the product of exponentiated output activations over time t and each with K output neurons. Then additive noise injection gives

$$\frac{p(\mathbf{y}+\mathbf{n},\mathbf{h}|\mathbf{x},\Theta)}{p(\mathbf{y},\mathbf{h}|\mathbf{x},\Theta)} = \frac{p(\mathbf{y}+\mathbf{n},\mathbf{h}|\mathbf{x},\Theta)\ p(\mathbf{h}|\mathbf{x},\Theta)}{p(\mathbf{h}|\mathbf{x},\Theta)\ p(\mathbf{y},\mathbf{h}|\mathbf{x},\Theta)} \tag{A.2}$$

$$= \frac{p(\mathbf{y}+\mathbf{n}|\mathbf{h},\mathbf{x},\Theta)}{p(\mathbf{y}|\mathbf{h},\mathbf{x},\Theta)} \tag{A.3}$$

$$= \frac{\prod_{t=t_0}^{T} p(\mathbf{y}^{(t)}+\mathbf{n}^{(t)}|\mathbf{h},\mathbf{x},\Theta)}{\prod_{t=t_0}^{T} p(\mathbf{y}^{(t)}|\mathbf{h},\mathbf{x},\Theta)} \tag{A.4}$$

$$= \prod_{t=t_0}^{T}\left(\frac{\prod_{k=1}^{K}(a_k^{y(t)})^{n_k^{(t)}+y_k^{(t)}}}{\prod_{k=1}^{K}(a_k^{y(t)})^{y_k^{(t)}}}\right) \tag{A.5}$$

$$= \prod_{t=t_0}^{T}\left(\prod_{k=1}^{K}(a_k^{y(t)})^{n_k^{(t)}}\right). \tag{A.6}$$

So the NEM positivity condition in (23) becomes

$$\mathbb{E}_{\mathbf{y},\mathbf{h},\mathbf{n}|\mathbf{x},\Theta^*}\left[\log\prod_{t=t_0}^{T}\left(\prod_{k=1}^{K}(a_k^{y(t)})^{n_k^{(t)}}\right)\right]\geqslant 0. \tag{A.7}$$

This simplifies to the inequality

$$\mathbb{E}_{\mathbf{y},\mathbf{h},\mathbf{n}|\mathbf{x},\Theta^*}\left[\sum_{t=t_0}^{T}\left(\sum_{k=1}^{K} n_k^{(t)}\ \log(a_k^{y(t)})\right)\right]\geqslant 0. \tag{A.8}$$

The inner sum is the inner product of $\mathbf{n}^{(t)}$ with $\log\ \mathbf{a}^{y(t)}$. Then we can rewrite (A.8) as the hyperplane inequality

$$\mathbb{E}_{\mathbf{y},\mathbf{h},\mathbf{n}|\mathbf{x},\Theta^*}\left[\sum_{t=t_0}^{T}\mathbf{n}^{(t)^T}\log\ \mathbf{a}^{y(t)}\right]\geqslant 0. \tag{A.9}$$

□

Theorem 3. *RBP Noise Benefit for a Regression RNN with Additive Noise Injection into the Output Neurons.*

The NEM positivity condition (23) holds for the maximum-likelihood training of a regression recurrent neural network with Gaussian target vector $\mathbf{y}^{(t)} \sim \mathcal{N}(\mathbf{y}^{(t)}|\mathbf{a}^{y(t)},\ \mathbf{I})$ and with additive noise injection into the output identity neurons if the following inequality condition holds:

$$\mathbb{E}_{\mathbf{y},\mathbf{h},\mathbf{n}|\mathbf{x},\Theta^*}\left[\sum_{t=t_0}^{T}||\mathbf{y}^{(t)}+\mathbf{n}^{(t)}-\mathbf{a}^{y(t)}||^2\right] - \mathbb{E}_{\mathbf{y},\mathbf{h},\mathbf{n}|\mathbf{x},\Theta^*}\left[\sum_{t=t_0}^{T}||\mathbf{y}^{(t)}-\mathbf{a}^{y(t)}||^2\right]\leqslant 0 \tag{A.10}$$

where the noise $\mathbf{n}^{(t)}$ injects additively into the output neurons and where $\mathbf{a}^{y(t)}$ is the output activation.

Proof. The error function $E(\Theta)$ is the squared error for a regression network [15]. We first show that minimizing the squared error $E(\Theta)$is the same as maximizing the likelihood function $L(\Theta)$:

$$E(\Theta) = \sum_{t=t_0}^{T} E^{(t)} \tag{A.11}$$

$$= -\sum_{t=t_0}^{T} -\frac{||\mathbf{y}^{(t)}-\mathbf{a}^{y(t)}||^2}{2} \tag{A.12}$$

$$= -\sum_{t=t_0}^{T}\log\left[\exp\left(-\frac{||\mathbf{y}^{(t)}-\mathbf{a}^{y(t)}||^2}{2}\right)\right] \tag{A.13}$$

$$= -(T-t_0)\ \log(2\pi)^{-\frac{K}{2}} + (T-t_0)\ \log(2\pi)^{-\frac{K}{2}} - \sum_{t=t_0}^{T}\log\left[\exp\left(-\frac{||\mathbf{y}^{(t)}-\mathbf{a}^{y(t)}||^2}{2}\right)\right] \tag{A.14}$$

$$= -L(\Theta) + (T-t_0)\ \log(2\pi)^{-\frac{K}{2}} \tag{A.15}$$

where the network's output probability density function $p(\mathbf{y}=\mathbf{y}^{(t)}|\mathbf{x},\Theta)$ is the vector normal density $\mathcal{N}(\mathbf{y}^{(\mathbf{t})}|\mathbf{a}^{(\mathbf{t})},\mathbf{I})$. So minimizing $E(\Theta)$ is the same as maximizing $L(\Theta)$ with respect to Θ. Injecting additive noise into the output neurons gives

$$\frac{p(\mathbf{y}+\mathbf{n},\mathbf{h}|\mathbf{x},\Theta)}{p(\mathbf{y},\mathbf{h}|\mathbf{x},\Theta)} = \frac{p(\mathbf{y}+\mathbf{n},\mathbf{h}|\mathbf{x},\Theta)\ p(\mathbf{h}|\mathbf{x},\Theta)}{p(\mathbf{h}|\mathbf{x},\Theta)\ p(\mathbf{y},\mathbf{h}|\mathbf{x},\Theta)} \tag{A.16}$$

$$= \frac{p(\mathbf{y}+\mathbf{n}|\mathbf{h},\mathbf{x},\Theta)}{p(\mathbf{y}|\mathbf{h},\mathbf{x},\Theta)} \tag{A.17}$$

$$= \frac{\prod_{t=t_0}^{T} p(\mathbf{y}^{(t)}+\mathbf{n}^{(t)}|\mathbf{h},\mathbf{x},\Theta)}{\prod_{t=t_0}^{T} p(\mathbf{y}^{(t)}|\mathbf{h},\mathbf{x},\Theta)} \tag{A.18}$$

$$= \prod_{t=t_0}^{T} \frac{p(\mathbf{y}^{(t)}+\mathbf{n}^{(t)}|\mathbf{h},\mathbf{x},\Theta)}{p(\mathbf{y}^{(t)}|\mathbf{h},\mathbf{x},\Theta)} \tag{A.19}$$

$$= \prod_{t=t_0}^{T} \frac{\exp\left(\frac{1}{2}||\left(\mathbf{y}^{(t)}-\mathbf{a}^{y(t)}||^2\right)}{\exp\left(\frac{1}{2}||\left(\mathbf{y}^{(t)}+\mathbf{n}^{(t)}-\mathbf{a}^{y(t)}||^2\right)}. \tag{A.20}$$

Then the NEM positivity condition in (23) reduces to

$$\mathbb{E}_{\mathbf{y},\mathbf{h},\mathbf{n}|\mathbf{x},\Theta^*}\left[\log\prod_{t=t_0}^{T}\frac{\exp(\frac{1}{2}||\mathbf{y}^{(t)}-\mathbf{a}^{y(t)}||^2)}{\exp(\frac{1}{2}||\mathbf{y}^{(t)}+\mathbf{n}^{(t)}-\mathbf{a}^{y(t)}||^2)}\right]\geqslant 0. \tag{A.21}$$

This positivity condition reduces to the following hyperspherical inequality:

$$\mathbb{E}_{\mathbf{y},\mathbf{h},\mathbf{n}|\mathbf{x},\Theta^*}\left[\sum_{t=1}^{T}||\mathbf{y}^{(t)}+\mathbf{n}^{(t)}-\mathbf{a}^{y(t)}||^2\right] - \mathbb{E}_{\mathbf{y},\mathbf{h},\mathbf{n}|\mathbf{x},\Theta^*}\left[\sum_{t=1}^{T}||\mathbf{y}^{(t)}-\mathbf{a}^{y(t)}||^2\right]\leqslant 0. \tag{A.22}$$

□

A.2. NEM Noise Injection into Hidden Units of Recurrent Neural Networks.

Theorem 4. *Injecting NEM Noise into the Hidden Neurons of LSTM-RNN.*

A NEM noise benefit holds for the iterative maximum-likelihood training of an LSTM-RNN with additive noise injection into the hidden units (gates) if the following positivity condition holds:

$$\mathbb{E}_{\mathbf{z},\mathbf{n}|\mathbf{y},\mathbf{x},\Theta^*}\left[\log\left(\frac{p(\mathbf{z}+\mathbf{n}|\mathbf{y},\mathbf{x},\Theta)}{p(\mathbf{z}|\mathbf{y},\mathbf{x},\Theta)}\right)\right]\geqslant 0. \tag{A.23}$$

This inequality reduces to

$$\mathbb{E}_{\mathbf{z},\mathbf{n}|\mathbf{y},\mathbf{x},\Theta^*}\left[\sum_{t=t_0}^{T}\left(\left(\mathbf{n}_i^{(t)}\right)^T\log\frac{\mathbf{i}^{(t)}}{1-\mathbf{i}^{(t)}}+\left(\mathbf{n}_f^{(t)}\right)^T\log\frac{\mathbf{f}^{(t)}}{1-\mathbf{f}^{(t)}}+\left(\mathbf{n}_o^{(t)}\right)^T\log\frac{\mathbf{o}^{(t)}}{1-\mathbf{o}^{(t)}}\right)\right]\geqslant 0 \tag{A.24}$$

where the noises $\mathbf{n}_i^{(t)}$, $\mathbf{n}_f^{(t)}$, and $\mathbf{n}_o^{(t)}$ inject additively into the respective input, forget, and output gates at time t. The terms $\mathbf{i}^{(t)}$, $\mathbf{f}^{(t)}$, and $\mathbf{o}^{(t)}$ are the activations for the respective input, forget, and output gates.

Proof. The inequality (A.23) comes from the Noisy EM Theorem [39,40] and implies that the following holds on average:

$$p(\mathbf{z}^{(t)}+\mathbf{n}^{(t)}|\mathbf{y},\mathbf{x},\Theta)\geqslant p(\mathbf{z}^{(t)}|\mathbf{y},\mathbf{x},\Theta) \tag{A.25}$$

where the noises $\mathbf{n}_i^{(t)}$, $\mathbf{n}_f^{(t)}$, and $\mathbf{n}_o^{(t)}$ inject into the respective input, forget, and output gates at time t. The conditional probability function $p(\mathbf{z}^{(t)}+\mathbf{n}^{(t)}|\mathbf{y},\mathbf{x},\Theta)$ factors in (77) just as $p(\mathbf{z}^{(t)}|\mathbf{y},\mathbf{x},\Theta)$ factors in (75). Then dividing by the noiseless term and taking logarithms gives

$$\log\left(\frac{p(\mathbf{z}^{(t)}+\mathbf{n}^{(t)}|\mathbf{y},\mathbf{x},\Theta)}{p(\mathbf{z}^{(t)}|\mathbf{y},\mathbf{x},\Theta)}\right) = \log\left(\frac{p(\mathbf{z}_i^{(t)}+\mathbf{n}_i^{(t)}|\mathbf{y},\mathbf{x},\Theta)}{p(\mathbf{z}_i^{(t)}|\mathbf{y},\mathbf{x},\Theta)}\right)+\log\left(\frac{p(\mathbf{z}_f^{(t)}+\mathbf{n}_f^{(t)}|\mathbf{y},\mathbf{x},\Theta)}{p(\mathbf{z}_f^{(t)}|\mathbf{y},\mathbf{x},\Theta)}\right)+\log\left(\frac{p(\mathbf{z}_o^{(t)}+\mathbf{n}_o^{(t)}|\mathbf{y},\mathbf{x},\Theta)}{p(\mathbf{z}_o^{(t)}|\mathbf{y},\mathbf{x},\Theta)}\right). \tag{A.26}$$

LSTM uses sigmoid activations for the gates. The Monte Carlo approximation above assumes that a Bernoulli distribution describes the latent variables $\mathbf{z}_i^{(t)}$, $\mathbf{z}_f^{(t)}$, and $\mathbf{z}_o^{(t)}$. This gives

$$\log\left(\frac{p(\mathbf{z}_i^{(t)}+\mathbf{n}_i^{(t)}|\mathbf{y},\mathbf{x},\Theta)}{p(\mathbf{z}_i^{(t)}|\mathbf{y},\mathbf{x},\Theta)}\right) = \log\prod_{t=t_0}^{T}\left(\prod_{j=1}^{J}\frac{\left(i_j^{(t)}\right)^{z_{i_j}^{(t)}+n_{i_j}^{(t)}}\left(1-i_j^{(t)}\right)^{1-z_{i_j}^{(t)}-n_{i_j}^{(t)}}}{\left(i_j^{(t)}\right)^{z_{i_j}^{(t)}}\left(1-i_j^{(t)}\right)^{1-z_{i_j}^{(t)}}}\right) \tag{A.27}$$

$$= \log \prod_{t=t_0}^{T} \left(\prod_{j=1}^{J} \frac{(i_j^{(t)})^{n_{i_j}^{(t)}}}{(1 - i_j^{(t)})^{n_j^{i(t)}}} \right) \tag{A.28}$$

$$= \sum_{t=t_o}^{T} \sum_{j=1}^{J} n_{i_j}^{(t)} \left(\log(i_j^{(t)}) - \log(1 - i_j^{(t)}) \right) \tag{A.29}$$

$$= \sum_{t=t_0}^{T} \left(\mathbf{n}_i^{(t)}\right)^T \log \frac{\mathbf{i}^{(t)}}{1 - \mathbf{i}^{(t)}} \tag{A.30}$$

for the input gates. The same approach also extends to forget and output gates and we have the following:

$$\log\left(\frac{p(\mathbf{z}_f^{(t)} + \mathbf{n}_f^{(t)} | \mathbf{y}, \mathbf{x}, \Theta)}{p(\mathbf{z}_f^{(t)} | \mathbf{y}, \mathbf{x}, \Theta)} \right) = \sum_{t=t_0}^{T} \left(\mathbf{n}_f^{(t)}\right)^T \log \frac{\mathbf{f}^{(t)}}{1 - \mathbf{f}^{(t)}} \tag{A.31}$$

$$\log\left(\frac{p(\mathbf{z}_o^{(t)} + \mathbf{n}_o^{(t)} | \mathbf{y}, \mathbf{x}, \Theta)}{p(\mathbf{z}_o^{(t)} | \mathbf{y}, \mathbf{x}, \Theta)} \right) = \sum_{t=t_0}^{T} \left(\mathbf{n}_o^{(t)}\right)^T \log \frac{\mathbf{o}^{(t)}}{1 - \mathbf{o}^{(t)}} \tag{A.32}$$

because they also use sigmoid activations. Taking the appropriate NEM averages gives

$$\mathbb{E}_{\mathbf{z},\mathbf{n}|\mathbf{y},\mathbf{x},\Theta^*}\left[\log\left(\frac{p(\mathbf{z} + \mathbf{n} | \mathbf{y}, \mathbf{x}, \Theta)}{p(\mathbf{z} | \mathbf{y}, \mathbf{x}, \Theta)} \right) \right] = \mathbb{E}_{\mathbf{z},\mathbf{n}|\mathbf{y},\mathbf{x},\Theta^*}\left[\sum_{t=t_0}^{T} \left(\left(\mathbf{n}_i^{(t)}\right)^T \log \frac{\mathbf{i}^{(t)}}{1 - \mathbf{i}^{(t)}} + \left(\mathbf{n}_f^{(t)}\right)^T \log \frac{\mathbf{f}^{(t)}}{1 - \mathbf{f}^{(t)}} + \left(\mathbf{n}_o^{(t)}\right)^T \log \frac{\mathbf{o}^{(t)}}{1 - \mathbf{o}^{(t)}} \right) \right] \geqslant 0 \tag{A.33}$$

from (A.23), (A.26), (A.30), (A.31), and (A.32). □

Theorem 5. *Injecting NEM Noise into the Hidden Neurons a GRU-RNN.*

A NEM noise benefit holds for the iterative maximum-likelihood training of a GRU-RNN with noise injection into the hidden units (gates) if the following positivity condition holds:

$$\mathbb{E}_{\mathbf{z},\mathbf{n}|\mathbf{y},\mathbf{x},\Theta^*}\left[\log\left(\frac{p(\mathbf{z} + \mathbf{n} | \mathbf{y}, \mathbf{x}, \Theta)}{p(\mathbf{z} | \mathbf{y}, \mathbf{x}, \Theta)} \right) \right] \geqslant 0. \tag{A.34}$$

The inequality reduces to

$$\mathbb{E}_{\mathbf{z},\mathbf{n}|\mathbf{y},\mathbf{x},\Theta^*}\left[\sum_{t=t_0}^{T} \left(\left(\mathbf{n}_d^{(t)}\right)^T \log \frac{\mathbf{d}^{(t)}}{1 - \mathbf{d}^{(t)}} + \left(\mathbf{n}_r^{(t)}\right)^T \log \frac{\mathbf{r}^{(t)}}{1 - \mathbf{r}^{(t)}} \right) \right] \geqslant 0 \tag{A.35}$$

where the noises $\mathbf{n}_d^{(t)}$ and $\mathbf{n}_r^{(t)}$ inject into the respective update and reset gates. The terms $\mathbf{d}^{(t)}$ and $\mathbf{r}^{(t)}$ are the activations for the respective update and reset gates at time t.

Proof. The NEM condition (A.34) entails that on average

$$p(\mathbf{z}^{(t)} + \mathbf{n}^{(t)} | \mathbf{y}, \mathbf{x}, \Theta) \geqslant p(\mathbf{z}^{(t)} | \mathbf{y}, \mathbf{x}, \Theta). \tag{A.36}$$

This above likelihood factorizations give

$$p(\mathbf{z}_d^{(t)} + \mathbf{n}_d^{(t)} | \mathbf{x}, \Theta)\ p(\mathbf{z}_r^{(t)} + \mathbf{n}_r^{(t)} | \mathbf{x}, \Theta) \geqslant p(\mathbf{z}_d^{(t)} | \mathbf{x}, \Theta)\ p(\mathbf{z}_r^{(t)} | \mathbf{x}, \Theta). \tag{A.37}$$

The proof now proceeds as in the above case of the LSTM-RNN since the sigmoid activations correspond to a Bernoulli probability structure. The corresponding expressions for the reset and update gates follow from extending the simplification under (A.27)–(A.30) to these gates. Then the log-likelihood ratio becomes

$$\log\left(\frac{p(\mathbf{z}^{(t)} + \mathbf{n}^{(t)} | \mathbf{y}, \mathbf{x}, \Theta)}{p(\mathbf{z}^{(t)} | \mathbf{y}, \mathbf{x}, \Theta)} \right) = \log\left(\frac{p(\mathbf{z}_d^{(t)} + \mathbf{n}_d^{(t)} | \mathbf{y}, \mathbf{x}, \Theta)}{p(\mathbf{z}_d^{(t)} | \mathbf{y}, \mathbf{x}, \Theta)} \right) + \log\left(\frac{p(\mathbf{z}_r^{(t)} + \mathbf{n}_r^{(t)} | \mathbf{y}, \mathbf{x}, \Theta)}{p(\mathbf{z}_r^{(t)} | \mathbf{y}, \mathbf{x}, \Theta)} \right) \tag{A.38}$$

$$= \sum_{t=t_0}^{T} \left(\left(\mathbf{n}_d^{(t)}\right)^T \log \frac{\mathbf{d}^{(t)}}{1 - \mathbf{d}^{(t)}} + \left(\mathbf{n}_r^{(t)}\right)^T \log \frac{\mathbf{r}^{(t)}}{1 - \mathbf{r}^{(t)}} \right). \tag{A.39}$$

So the NEM positivity condition reduces to

$$\mathbb{E}_{\mathbf{z},\mathbf{n}|\mathbf{y},\mathbf{x},\Theta^*}\left[\log\left(\frac{p(\mathbf{z} + \mathbf{n} | \mathbf{y}, \mathbf{x}, \Theta)}{p(\mathbf{z} | \mathbf{y}, \mathbf{x}, \Theta)} \right) \right] = \mathbb{E}_{\mathbf{z},\mathbf{n}|\mathbf{y},\mathbf{x},\Theta^*}\left[\sum_{t=t_0}^{T} \left(\left(\mathbf{n}_d^{(t)}\right)^T \log \frac{\mathbf{d}^{(t)}}{1 - \mathbf{d}^{(t)}} + \left(\mathbf{n}_r^{(t)}\right)^T \log \frac{\mathbf{r}^{(t)}}{1 - \mathbf{r}^{(t)}} \right) \right] \geqslant 0 \tag{A.40}$$

from (A.34) and (A.39). □

Appendix B. Recurrent Backpropagation Training with Long Short-Term Memory (LSTM)

We now present the RBP learning algorithm for training an LSTM recurrent network [3]. These learning laws are applicable to both regression and classification networks with the LSTM architecture because of BP invariance.

The parameters for an LSTM network are as follows. The $J \times I$ matrix $\mathbf{W}^x$ connects the input unit $\mathbf{x}^{(t)}$ to the hidden unit $\mathbf{o}^{c(t)}$. The $J \times J$ matrix $\mathbf{V}^h$ connects the hidden state $\mathbf{a}^{h(t-1)}$ to $\mathbf{o}^{c(t)}$ and $\mathbf{b}^x$ is the bias for $\mathbf{o}^{c(t)}$. The $J \times I$ matrices $\mathbf{W}^\gamma, \mathbf{W}^\phi$, and $\mathbf{W}^\omega$ connect the input unit $\mathbf{x}^{(t)}$ to the respective input, forget, and output gates. The $J \times J$ matrices $\mathbf{V}^\gamma, \mathbf{V}^\phi$, and $\mathbf{V}^\omega$ connect the previous hidden state $\mathbf{a}^{h(t-1)}$ to the respective input, forget, and output gates at time t. The terms $\mathbf{b}^\gamma, \mathbf{b}^\phi$, and $\mathbf{b}^\omega$ are the bias for the respective input, hidden, and output gates. The $K \times J$ matrix $\mathbf{U}^y$ connects the output unit $\mathbf{o}^{y(t)}$ to the hidden state $\mathbf{a}^{h(t)}$. The term $\mathbf{b}^y$ is the bias of the output unit. We initialize the hidden state $\mathbf{a}^{h(0)}$ to $\mathbf{0}$. We also initialize the internal cell $\mathbf{s}^{(0)}$ to $\mathbf{0}$.

B.1. Forward Pass

Forward pass over an LSTM-RNN is the propagation of input $\mathbf{x}^{(t)}$ from the input layer to the output layer through the hidden units. The input $o_j^{c(t)}$ to the jth hidden unit of an LSTM unit at time t is

$$o_j^{c(t)} = \sum_{i=1}^{I} w_{ji}^x a_i^{x(t)} + \sum_{l=1}^{J} v_{jl}^h a_j^{h(t-1)} + b_j^x \tag{B.1}$$

where the input unit uses identity activation $a_i^{x(t)} = x_i^{(t)}$. The hidden unit $a_j^{c(t)}$ uses hyperbolic tangent activation so we have

$$a_j^{c(t)} = \tanh\left(o_j^{c(t)}\right) = \frac{\exp^{o_j^{c(t)}} - \exp^{-o_j^{c(t)}}}{\exp^{o_j^{c(t)}} + \exp^{-o_j^{c(t)}}}. \tag{B.2}$$

The input $o_j^{\gamma(t)}$ of the jth input gate at time t is as follows:

$$o_j^{\gamma(t)} = \sum_{i=1}^{I} w_{ji}^\gamma a_i^{x(t)} + \sum_{l=1}^{J} v_{jl}^\gamma a_j^{h(t-1)} + b_j^\gamma \tag{B.3}$$

and the corresponding activation is the logistic sigmoid function so we have

$$a_j^{\gamma(t)} = \frac{1}{1 + \exp^{-o_j^{\gamma(t)}}}. \tag{B.4}$$

The input $o_j^{\phi(t)}$ of the jth forget gate at time t is as follows:

$$o_j^{\phi(t)} = \sum_{i=1}^{I} w_{ji}^\phi a_i^{x(t)} + \sum_{l=1}^{J} v_{jl}^\phi a_j^{h(t-1)} + b_j^\phi \tag{B.5}$$

and the corresponding activation is

$$a_j^{\phi(t)} = \frac{1}{1 + \exp^{-o_j^{\phi(t)}}}. \tag{B.6}$$

The input $o_j^{\omega(t)}$ of the jth output gate at time t is

$$o_j^{\omega(t)} = \sum_{i=1}^{I} w_{ji}^\omega a_i^{x(t)} + \sum_{l=1}^{J} v_{jl}^\omega a_j^{h(t-1)} + b_j^\omega \tag{B.7}$$

and the corresponding activation is

$$a_j^{\omega(t)} = \frac{1}{1 + \exp^{-o_j^{\omega(t)}}}. \tag{B.8}$$

The term $s_j^{(t)}$ represents the input of the jth internal cell at time t. We have the following:

$$s_j^{(t)} = \left(a_j^{c(t)} a_j^{\gamma(t)}\right) + \left(s_j^{(t-1)} a_j^{\phi(t)}\right) \tag{B.9}$$

and the corresponding activation is

$$a_j^{s(t)} = \tanh\left(s_j^{(t)}\right) = \frac{\exp^{s_j^{(t)}} - \exp^{-s_j^{(t)}}}{\exp^{s_j^{(t)}} + \exp^{-s_j^{(t)}}}. \tag{B.10}$$

The input $a_j^{h(t)}$ of the j^{th} hidden state at time t is

$$a_j^{h(t)} = a_j^{s(t)} a_j^{\omega(t)}. \tag{B.11}$$

The input $o_k^{y(t)}$ of the k^{th} output neuron at time t is

$$o_k^{y(t)} = \sum_{j=1}^{J} u_{kj}^{y} a_j^{h(t)} + b_k^{y}. \tag{B.12}$$

The output activation depends on the structure of the network. The output neurons use softmax activations for a classification network [50,15]. So the k^{th} output neuron $a_k^{y(t)}$at time t has the softmax form

$$a_k^{y(t)} = \frac{\exp\left(o_k^{y(t)}\right)}{\sum_{l=1}^{K} \exp\left(o_l^{y(t)}\right)}. \tag{B.13}$$

The softmax ratio structure produces a length-K output probability vector. So the softmax output's normalization structure enforces a winner-take-all structure $a_k^{y(t)} = 1$ if any output neuron achieves a maximal activation. The output layer uses identity activation for a regression network. We have

$$a_k^{y(t)} = o_k^{y(t)}. \tag{B.14}$$

B.2. Backward Pass

The network probability $p(\mathbf{y}|\mathbf{x}, \Theta)$ for a classifier defines a one-shot multinomial or categorical probability density function because of (B.13) and because we use 1-in-K encoding. Taking logarithms gives the network's output log-likelihood L as the sum of probabilities times the logarithm of probabilities. This sum is precisely the cross entropy $E^{(t)} = -\sum_{k=1}^{K} y_k^{(t)} \log a_k^{y(t)}$ [15]. Then summing over the time slices gives the system error or performance measure $E(\Theta)$ as the sum of cross-entropies

$$E(\Theta) = \sum_{t=t_0}^{T} E^{(t)} = -\sum_{t=t_0}^{T} \sum_{k=1}^{K} y_k^{(t)} \log a_k^{y(t)}. \tag{B.15}$$

The chain rule gives the partial derivative of the error $E^{(t)}$ with respect to the weight u_{kj}^{y} is as follows:

$$\frac{\partial E^{(t)}}{\partial u_{kj}^{y}} = \frac{\partial E^{(t)}}{\partial o_k^{y(t)}} \frac{\partial o_k^{y(t)}}{\partial u_{kj}^{y}} \tag{B.16}$$

$$= \left(\sum_{l=1}^{K} \frac{\partial E^{(t)}}{\partial a_l^{y(t)}} \frac{\partial a_l^{y(t)}}{\partial o_k^{y(t)}} \right) \frac{\partial o_k^{y(t)}}{\partial u_{kj}^{y}} \tag{B.17}$$

$$= -\left(\frac{y_k^{(t)}}{a_k^{y(t)}} \frac{\partial a_k^{y(t)}}{\partial o_k^{y(t)}} + \sum_{l \neq k}^{K} \frac{y_l^{(t)}}{a_l^{y(t)}} \frac{\partial a_l^{y(t)}}{\partial o_k^{y(t)}} \right) \frac{\partial o_k^{y(t)}}{\partial u_{kj}^{y}} \tag{B.18}$$

$$= -\left(\frac{y_k^{(t)}}{a_k^{y(t)}} a_k^{y(t)} \left(1 - a_k^{y(t)}\right) + \sum_{l \neq k}^{K} \frac{y_l^{(t)}}{a_l^{y(t)}} a_l^{y(t)} a_k^{y(t)} \right) \frac{\partial o_k^{y(t)}}{\partial u_{kj}^{y}} \tag{B.19}$$

$$= -\left(y_k^{(t)} - a_k^{y(t)} \sum_{l=1}^{K} y_l^{(t)} \right) \frac{\partial o_k^{y(t)}}{\partial u_{kj}^{y}} \tag{B.20}$$

$$= -\left(y_k^{(t)} - a_k^{y(t)} \right) \frac{\partial o_k^{y(t)}}{\partial u_{kj}^{y}} \tag{B.21}$$

$$= -\left(y_k^{(t)} - a_k^{y(t)} \right) a_j^{h(t)}. \tag{B.22}$$

The chain rule likewise gives the partial derivative of $E^{(t)}$ with respect to the bias b_k^{y} as

$$\frac{\partial E^{(t)}}{\partial b_k^{y}} = \left(\sum_{l=1}^{K} \frac{\partial E^{(t)}}{\partial a_l^{y(t)}} \frac{\partial a_l^{y(t)}}{\partial o_k^{y(t)}} \right) \frac{\partial o_k^{y(t)}}{\partial b_k^{y}} = -\left(y_k^{(t)} - a_k^{y(t)} \right). \tag{B.23}$$

Let $E^{(t)'}$ denote the partial derivative of $E^{(t)}$ with respect to the hidden-unit activation $a_j^{h(t)}$. Then the partial derivative has the form

$$E^{(t)'} = \frac{\partial E^{(t)}}{\partial a_j^{h(t)}} \tag{B.24}$$

$$= \sum_{k=1}^{K}\left(\sum_{l=1}^{K}\frac{\partial E^{(t)}}{\partial a_l^{y(t)}}\frac{\partial a_l^{y(t)}}{\partial o_k^{y(t)}}\right)\frac{\partial o_k^{y(t)}}{\partial a_j^{h(t)}} \tag{B.25}$$

$$= -\sum_{k=1}^{K}(y_k^{(t)} - a_k^{y(t)})u_{kj}^{y}. \tag{B.26}$$

A regression network uses squared error as its cost function. Therefore the error function for a regression network with the LSTM architecture is

$$E^{(t)} = \sum_{t=t_0}^{T}\sum_{k=1}^{K}\left(y_k^{(t)} - a_k^{y(t)}\right)^2. \tag{B.27}$$

The partial derivatives in (B.22)–(B.26) stays the same if we replace the sum of cross-entropies (B.15) with the sum of squared errors (B.27) because of the BP invariance.

The term $a_j^{c(t)'}$ denotes the partial derivative of $a_j^{c(t)}$ with respect to the input $o_j^{c(t)}$. We have

$$a_j^{c(t)'} = \frac{\partial a_j^{c(t)}}{\partial o_j^{c(t)}} = 1 - \left(a_j^{c(t)}\right)^2 \tag{B.28}$$

and the partial derivative $a_j^{s(t)'}$ of the cell state is as follows:

$$a_j^{s(t)'} = \frac{\partial a_j^{s(t)}}{\partial s_j^{(t)}} = 1 - \left(a_j^{s(t)}\right)^2. \tag{B.29}$$

The partial derivative $a_j^{\gamma(t)'}$ of the input gate activation with respect to its input is

$$a_j^{\gamma(t)'} = \frac{\partial a_j^{\gamma(t)}}{\partial o_j^{\gamma(t)}} = a_j^{\gamma(t)}\left(1 - a_j^{\gamma(t)}\right) \tag{B.30}$$

and the corresponding derivative $a_j^{\phi(t)'}$ of the forget gate activation with respect to its input is

$$a_j^{\phi(t)'} = \frac{\partial a_j^{\phi(t)}}{\partial o_j^{\phi(t)}} = a_j^{\phi(t)}\left(1 - a_j^{\phi(t)}\right). \tag{B.31}$$

The partial derivative $a_j^{\omega(t)'}$ of the output gate activation is as follows:

$$a_j^{\omega(t)'} = \frac{\partial a_j^{\omega(t)}}{\partial o_j^{\omega(t)}} = a_j^{\omega(t)}\left(1 - a_j^{\omega(t)}\right). \tag{B.32}$$

We now present the derivatives with respect to the weights that connect the input gate units to the input and hidden units. The partial derivative of $a_l^{h(t)}$ with respect to weight w_{ji}^{γ} is as follows:

$$\frac{\partial a_l^{h(t)}}{\partial w_{ji}^{\gamma}} = a_l^{\omega(t)}\frac{\partial a_l^{s(t)}}{\partial w_{ji}^{\gamma}} + a_l^{s(t)}\frac{\partial a_l^{\omega(t)}}{\partial o_l^{\omega(t)}}\frac{\partial o_l^{\omega(t)}}{\partial w_{ji}^{\gamma}} \tag{B.33}$$

$$= a_l^{\omega(t)}a_l^{s(t)'}\frac{\partial s_l^{(t)}}{\partial w_{ji}^{\gamma}} + a_l^{s(t)}a_l^{\omega(t)'}\sum_{m=1}^{J}v_{lm}^{\omega}\frac{\partial a_m^{h(t-1)}}{\partial w_{ji}^{\gamma}} \tag{B.34}$$

and

$$\frac{\partial s_l^{(t)}}{\partial w_{ji}^{\gamma}} = a_l^{c(t)}\frac{\partial a_l^{\gamma(t)}}{\partial w_{ji}^{\gamma}} + a_l^{\gamma(t)}\frac{\partial a_l^{c(t)}}{\partial w_{ji}^{\gamma}} + a_l^{\phi(t)}\frac{\partial s_l^{(t-1)}}{\partial w_{ji}^{\gamma}} + s_l^{(t-1)}\frac{\partial a_l^{\phi(t)}}{\partial w_{ji}^{\gamma}} \tag{B.35}$$

$$= a_l^{c(t)}a_l^{\gamma(t)'}\left(a_i^{x(t)} + \sum_{m=1}^{J}v_{lm}^{\gamma}\frac{\partial a_m^{h(t-1)}}{\partial w_{ji}^{\gamma}}\right) + a_j^{\gamma(t)}a_j^{c(t)'}\sum_{m=1}^{J}v_{lm}^{h}\frac{\partial a_m^{h(t-1)}}{\partial w_{ji}^{\gamma}} + a_j^{\phi(t)}\frac{\partial s_j^{(t-1)}}{\partial w_{ji}^{\gamma}} + s_l^{(t-1)}a_l^{\phi(t)'}\sum_{m=1}^{J}v_{lm}^{\phi}\frac{\partial a_m^{h(t-1)}}{\partial w_{ji}^{\gamma}}. \tag{B.36}$$

This gives a recursive expression for computing the derivative from $t = 0$ to $t = T$. The initial condition is

$$\frac{\partial s_l^{(0)}}{\partial w_{ji}^{\gamma}} = \frac{\partial a_l^{h(0)}}{\partial w_{ji}^{\gamma}} = 0 \tag{B.37}$$

where $l \in \{1,\ldots,J\}$. The derivative of $s_l^{(t)}$ with respect to w_{ji}^{γ} for $t \in \{1,\ldots,T\}$ follows from applying (B.34)–(B.37). Therefore the recursive expression for the derivative of $E^{(t)}$ with respect to w_{ji}^{γ} follows from (B.26) and (B.34)–(B.38)

$$\frac{\partial E^{(t)}}{\partial w_{ji}^{\gamma}} = \sum_{l=1}^{J} \frac{\partial E^{(t)}}{\partial a_l^{h(t)}} \frac{\partial a_l^{h(t)}}{\partial w_{ji}^{\gamma}}. \tag{B.38}$$

The derivative of $E^{(t)}$ with respect to the bias b_j^{γ} of the input gate follows from replacing $a_i^{x(t)}$ with 1 and w_{ji}^{γ} with b_j^{γ} in (B.33)–(B.38). So we have:

$$\frac{\partial a_l^{h(t)}}{\partial b_j^{\gamma}} = a_l^{\omega(t)} a_l^{s(t)'} \frac{\partial s_l^{(t)}}{\partial b_j^{\gamma}} + a_l^{s(t)} a_l^{\omega(t)'} \sum_{m=1}^{J} v_{lm}^{\omega} \frac{\partial a_m^{h(t-1)}}{\partial b_j^{\gamma}} \tag{B.39}$$

and

$$\frac{\partial s_l^{(t)}}{\partial b_j^{\gamma}} = a_l^{c(t)} a_l^{\gamma(t)'} \left(1 + \sum_{m=1}^{J} v_{lm}^{\gamma} \frac{\partial a_m^{h(t-1)}}{\partial b_j^{\gamma}}\right) + a_l^{\gamma(t)} a_l^{c(t)'} \sum_{m=1}^{J} v_{lm}^{h} \frac{\partial a_m^{h(t-1)}}{\partial b_j^{\gamma}} + a_l^{\phi(t)} \frac{\partial s_l^{(t-1)}}{\partial b_j^{\gamma}} + s_l^{(t-1)} a_l^{\phi(t)'} \sum_{m=1}^{J} v_{lm}^{\phi} \frac{\partial a_m^{h(t-1)}}{\partial b_j^{\gamma}}. \tag{B.40}$$

Therefore we have the following initial condition:

$$\frac{\partial s_l^{(0)}}{\partial b_j^{\gamma}} = \frac{\partial a_l^{h(0)}}{\partial b_j^{\gamma}} = 0. \tag{B.41}$$

So the recursive expression for the partial derivative of $E^{(t)}$ with respect to b_j^{γ} follows from (B.39)–(B.42)

$$\frac{\partial E^{(t)}}{\partial b_j^{\gamma}} = \sum_{l=1}^{J} \frac{\partial E^{(t)}}{\partial a_l^{h(t)}} \frac{\partial a_l^{h(t)}}{\partial b_j^{\gamma}}. \tag{B.42}$$

The partial derivative of the hidden activation $a_m^{h(t)}$ with respect to weight v_{jl}^{γ} follows from replacing every occurrence of subscript l with m and w_{ji}^{γ} with v_{jl}^{γ} in (B.33). The derivative of $s_m^{(t)}$ also follows from replacing subscript l with m and w_{ji}^{γ} with v_{jl}^{γ} in (B.35). These substitutions give the following:

$$\frac{\partial a_m^{h(t)}}{\partial v_{jl}^{\gamma}} = a_m^{\omega(t)} a_m^{s(t)'} \frac{\partial s_m^{(t)}}{\partial v_{jl}^{\gamma}} + a_m^{s(t)} a_m^{\omega(t)'} \sum_{p=1}^{J} v_{jp}^{\omega} \frac{\partial a_p^{h(t-1)}}{\partial v_{jl}^{\gamma}} \tag{B.43}$$

and

$$\frac{\partial s_m^{(t)}}{\partial v_{jl}^{\gamma}} = a_m^{c(t)} a_m^{\gamma(t)'} \left(a_l^{h(t-1)} + \sum_{p=1}^{J} v_{mp}^{\gamma} \frac{\partial a_p^{h(t-1)}}{\partial v_{jl}^{\gamma}}\right) + a_m^{\gamma(t)} a_m^{c(t)'} \sum_{p=1}^{J} v_{mp}^{h} \frac{\partial a_p^{h(t-1)}}{\partial v_{jl}^{\gamma}} + a_m^{\phi(t)} \frac{\partial s_m^{(t-1)}}{\partial v_{jl}^{\gamma}} + s_m^{(t-1)} a_m^{\phi(t)'} \left(\sum_{p=1}^{J} v_{mp}^{\gamma} \frac{\partial a_m^{h(t-1)}}{\partial v_{jl}^{\gamma}}\right). \tag{B.44}$$

Therefore we have the following initial conditions for the derivatives

$$\frac{\partial s_m^{(0)}}{\partial v_{jl}^{\gamma}} = \frac{\partial s_m^{(1)}}{\partial v_{jl}^{\gamma}} = 0 \tag{B.45}$$

$$\frac{\partial a_m^{h(0)}}{\partial v_{jl}^{\gamma}} = \frac{\partial a_m^{h(1)}}{\partial v_{jl}^{\gamma}} = 0 \tag{B.46}$$

where $m \in \{1,\ldots,J\}$. This is so because $\mathbf{a}^{h(0)}$ is equal to $\mathbf{0}$ and $\mathbf{s}^{(0)}$ is equal to $\mathbf{0}$. So the recursive expression for the derivative of $E^{(t)}$ with respect to v_{jl}^{γ} follows from (B.26) and (B.43)–(B.47)

$$\frac{\partial E^{(t)}}{\partial v_{jl}^{\gamma}} = \sum_{m=1}^{J} \frac{\partial E^{(t)}}{\partial a_m^{h(t)}} \frac{\partial a_m^{h(t)}}{\partial v_{jl}^{\gamma}}. \tag{B.47}$$

We now present the derivatives for the weights that connect the forget gate units to the input and output. This partial derivative of $a_l^{h(t)}$ with respect to w_{ji}^{ϕ} follows from replacing w_{ji}^{γ} with w_{ji}^{ϕ} in (B.33). This gives the following:

$$\frac{\partial a_l^{h(t)}}{\partial w_{ji}^{\phi}} = a_l^{\omega(t)} a_l^{s(t)'} \frac{\partial s_l^{(t)}}{\partial w_{ji}^{\phi}} + a_l^{s(t)} a_l^{\omega(t)'} \sum_{m=1}^{J} v_{lm}^{\omega} \frac{\partial a_m^{h(t-1)}}{\partial w_{ji}^{\phi}}. \tag{B.48}$$

The derivative of $s_l^{(t)}$ with respect to w_{ji}^{ϕ} follows from replacing w_{ji}^{γ} with w_{ji}^{ϕ} in (B.35). This gives the following:

$$\frac{\partial s_l^{(t)}}{\partial w_{ji}^{\phi}} = a_l^{c(t)} a_l^{\gamma(t)'} \sum_{m=1}^{J} v_{lm}^{\gamma} \frac{\partial a_m^{h(t-1)}}{\partial w_{ji}^{\phi}} + a_l^{\gamma(t)} a_l^{c(t)'} \sum_{m=1}^{J} v_{lm}^{h} \frac{\partial a_m^{h(t-1)}}{\partial w_{ji}^{\phi}} + s_l^{(t-1)} a_l^{\phi(t)'} \left(\delta(j-l) a_i^{x(t)} + \sum_{m=1}^{J} v_{lm}^{\phi} \frac{\partial a_m^{h(t-1)}}{\partial w_{ji}^{\phi}} \right) + a_l^{\phi(t)} \frac{\partial s_l^{(t-1)}}{\partial w_{ji}^{\phi}} \tag{B.49}$$

where δ represents the Kronecker delta function. The initial conditions are

$$\frac{\partial s_l^{(0)}}{\partial w_{ji}^{\phi}} = \frac{\partial s_l^{(1)}}{\partial w_{ji}^{\phi}} = 0 \tag{B.50}$$

$$\frac{\partial a_l^{h(0)}}{\partial w_{ji}^{\phi}} = \frac{\partial a_l^{h(1)}}{\partial w_{ji}^{\phi}} = 0 \tag{B.51}$$

because $\mathbf{a}^{h(0)}$ is equal to $\mathbf{0}$ and $\mathbf{s}^{(0)}$ is equal to $\mathbf{0}$. So the recursive expression for the derivative of $E^{(t)}$ with respect to w_{ji}^{ϕ} follows from (B.26) and (B.48)–(B.52)

$$\frac{\partial E^{(t)}}{\partial w_{ji}^{\phi}} = \sum_{l=1}^{J} \frac{\partial E^{(t)}}{\partial a_l^{h(t)}} \frac{\partial a_l^{h(t)}}{\partial w_{ji}^{\phi}}. \tag{B.52}$$

The derivatives of $a_l^{h(t)}$ and $s_l^{(t)}$ with respect to the forget bias b_j^{ϕ} follow from replacing $a_i^{x(t)}$ with 1 and w_{ji}^{ϕ} in (B.48)–(B.52). So we have

$$\frac{\partial a_l^{h(t)}}{\partial b_j^{\phi}} = a_l^{\omega(t)} a_l^{s(t)'} \frac{\partial s_l^{(t)}}{\partial b_j^{\phi}} + a_l^{s(t)} a_l^{\omega(t)'} \sum_{m=1}^{J} v_{lm}^{\omega} \frac{\partial a_m^{h(t-1)}}{\partial b_j^{\phi}} \tag{B.53}$$

and

$$\frac{\partial s_l^{(t)}}{\partial b_j^{\phi}} = a_l^{c(t)} a_l^{\gamma(t)'} \sum_{m=1}^{J} v_{lm}^{\gamma} \frac{\partial a_m^{h(t-1)}}{\partial b_j^{\phi}} + a_l^{\gamma(t)} a_l^{c(t)'} \sum_{m=1}^{J} v_{lm}^{h} \frac{\partial a_m^{h(t-1)}}{\partial b_j^{\phi}} + a_l^{\phi(t)} \frac{\partial s_l^{(t-1)}}{\partial b_j^{\phi}} + s_l^{(t-1)} a_l^{\phi(t)'} \left(1 + \sum_{m=1}^{J} v_{lm}^{\phi} \frac{\partial a_m^{h(t-1)}}{\partial b_j^{\phi}} \right). \tag{B.54}$$

Therefore we have the following initial condition:

$$\frac{\partial s_l^{(0)}}{\partial b_j^{\phi}} = \frac{\partial s_l^{(1)}}{\partial b_j^{\phi}} = \frac{\partial a_l^{h(0)}}{\partial b_j^{\phi}} = \frac{\partial a_l^{h(1)}}{\partial b_j^{\phi}} = 0. \tag{B.55}$$

So the recursive expression for the partial derivative of $E^{(t)}$ with respect to b_j^{ϕ} follows from (B.53)–(B.56)

$$\frac{\partial E^{(t)}}{\partial b_j^{\phi}} = \sum_{l=1}^{J} \frac{\partial E^{(t)}}{\partial a_l^{h(t)}} \frac{\partial a_l^{h(t)}}{\partial b_j^{\phi}}. \tag{B.56}$$

The partial derivative of the hidden activation $a_m^{h(t)}$ with respect to weight v_{jl}^{ϕ} follows from replacing subscript l with m and w_{ji}^{γ} with v_{jl}^{ϕ} in (B.33). This gives the following:

$$\frac{\partial a_m^{h(t)}}{\partial v_{jl}^{\phi}} = a_m^{\omega(t)} a_m^{s(t)'} \frac{\partial s_m^{(t)}}{\partial v_{jl}^{\phi}} + a_m^{s(t)} a_m^{\omega(t)'} \sum_{p=1}^{J} v_{mp}^{\omega} \frac{\partial a_p^{h(t-1)}}{\partial v_{jl}^{\phi}} \tag{B.57}$$

and the derivative of $s_m^{(t)}$ follows from replacing subscript l with m and w_{ji}^{γ} with v_{jl}^{γ} in (B.35)

$$\frac{\partial s_m^{(t)}}{\partial v_{jl}^{\phi}} = a_m^{c(t)} a_m^{\gamma(t)'} \left(a_l^{h(t-1)} + \sum_{p=1}^{J} v_{mp}^{\gamma} \frac{\partial a_p^{h(t-1)}}{\partial v_{jl}^{\phi}} \right) + a_m^{\phi(t)} \frac{\partial s_m^{(t-1)}}{\partial v_{jl}^{\phi}} + s_m^{(t-1)} a_m^{\phi(t)'} \left(\sum_{p=1}^{J} v_{mp}^{\phi} \frac{\partial a_m^{h(t-1)}}{\partial v_{jl}^{\phi}} \right) + a_m^{\gamma(t)} a_m^{c(t)'} \sum_{p=1}^{J} v_{mp}^{h} \frac{\partial a_p^{h(t-1)}}{\partial v_{jl}^{\phi}}. \tag{B.58}$$

Therefore we have the following initial condition:

$$\frac{\partial s_m^{(0)}}{\partial v_{jl}^{\phi}} = \frac{\partial s_m^{(1)}}{\partial v_{jl}^{\phi}} = \frac{\partial a_m^{h(0)}}{\partial v_{jl}^{\phi}} = \frac{\partial a_m^{h(1)}}{\partial v_{jl}^{\phi}} = 0 \tag{B.59}$$

where $m \in \{1, \ldots, J\}$. The recursive expression for the derivative of $E^{(t)}$ with respect to v_{jl}^{ϕ} follows from (B.26) and (B.57)–(B.60)

$$\frac{\partial E^{(t)}}{\partial v_{jl}^{\phi}} = \sum_{m=1}^{J} \frac{\partial E^{(t)}}{\partial a_m^{h(t)}} \frac{\partial a_m^{h(t)}}{\partial v_{jl}^{\phi}}. \tag{B.60}$$

We now present the derivatives for the weights that connect the output gate units to the input and output. The partial derivative of $a_l^{h(t)}$ with respect to the weight w_{ji}^{ω} follows from replacing w_{ji}^{γ} with w_{ji}^{ω} in (B.33)

$$\frac{\partial a_l^{h(t)}}{\partial w_{ji}^{\omega}} = a_l^{\omega(t)} a_l^{s(t)'} \frac{\partial s_l^{(t)}}{\partial w_{ji}^{\omega}} + a_l^{s(t)} a_l^{\omega(t)'} \left(a_i^{x(t)} \sum_{m=1}^{J} v_{lm}^{\omega} \frac{\partial a_m^{h(t-1)}}{\partial w_{ji}^{\omega}} \right) \tag{B.61}$$

and the derivative of $s_l^{(t)}$ with respect to w_{ji}^{ω} comes from replacing w_{ji}^{γ} with w_{ji}^{ω} in (B.35)

$$\frac{\partial s_l^{(t)}}{\partial w_{ji}^{\omega}} = a_l^{c(t)} a_l^{\gamma(t)'} \sum_{m=1}^{J} v_{lm}^{\gamma} \frac{\partial a_m^{h(t-1)}}{\partial w_{ji}^{\omega}} + a_l^{\gamma(t)} a_l^{c(t)'} \sum_{m=1}^{J} v_{lm}^{h} \frac{\partial a_m^{h(t-1)}}{\partial w_{ji}^{\omega}} + a_l^{\phi(t)} \frac{\partial s_j^{(t-1)}}{\partial w_{ji}^{\omega}} + s_l^{(t-1)} a_l^{\phi(t)'} \sum_{m=1}^{J} v_{lm}^{\phi} \frac{\partial a_m^{h(t-1)}}{\partial w_{ji}^{\omega}}. \tag{B.62}$$

Therefore the initial condition is

$$\frac{\partial s_l^{(0)}}{\partial w_{ji}^{\omega}} = \frac{\partial s_l^{(1)}}{\partial w_{ji}^{\omega}} = \frac{\partial a_l^{h(0)}}{\partial w_{ji}^{\omega}} = 0 \tag{B.63}$$

$$\frac{\partial a_l^{h(1)}}{\partial w_{ji}^{\omega}} = a_l^{s(1)} a_l^{\omega(1)'} a_i^{x(1)} \tag{B.64}$$

because $\mathbf{a}^{h(0)} = \mathbf{0}$ and $\mathbf{s}^{(0)} = \mathbf{0}$. Therefore the recursive expression for the derivative of $E^{(t)}$ with respect to w_{ji}^{ω} follows from (B.26) and (B.61)–(B.65)

$$\frac{\partial E^{(t)}}{\partial w_{ji}^{\omega}} = \sum_{l=1}^{J} \frac{\partial E^{(t)}}{\partial a_l^{h(t)}} \frac{\partial a_l^{h(t)}}{\partial w_{ji}^{\omega}}. \tag{B.65}$$

The derivatives of $a_l^{h(t)}$ and $s_l^{(t)}$ with respect to the output gate bias b_j^{ω} follow from replacing $a_i^{x(t)}$ with 1 and w_{ji}^{ϕ} in (B.48)–(B.52). Therefore we have

$$\frac{\partial a_l^{h(t)}}{\partial b_j^{\omega}} = a_l^{\omega(t)} a_l^{s(t)'} \frac{\partial s_l^{(t)}}{\partial b_j^{\omega}} + a_l^{s(t)} a_l^{\omega(t)'} \left(1 + \sum_{m=1}^{J} v_{lm}^{\omega} \frac{\partial a_m^{h(t-1)}}{\partial b_j^{\omega}} \right) \tag{B.66}$$

$$\frac{\partial s_l^{(t)}}{\partial b_j^{\omega}} = a_l^{c(t)} a_l^{\gamma(t)'} \sum_{m=1}^{J} v_{lm}^{\gamma} \frac{\partial a_m^{h(t-1)}}{\partial b_j^{\omega}} + a_l^{\gamma(t)} a_l^{c(t)'} \sum_{m=1}^{J} v_{lm}^{h} \frac{\partial a_m^{h(t-1)}}{\partial b_j^{\omega}} + a_l^{\phi(t)} \frac{\partial s_l^{(t-1)}}{\partial b_j^{\omega}} + s_l^{(t-1)} a_l^{\phi(t)'} \sum_{m=1}^{J} v_{lm}^{\phi} \frac{\partial a_m^{h(t-1)}}{\partial b_j^{\omega}}. \tag{B.67}$$

Therefore we have the following initial conditions:

$$\frac{\partial s_l^{(0)}}{\partial b_j^{\omega}} = \frac{\partial s_l^{(1)}}{\partial b_j^{\omega}} = \frac{\partial a_l^{h(0)}}{\partial b_j^{\omega}} = 0 \tag{B.68}$$

$$\frac{\partial a_l^{h(1)}}{\partial b_j^{\omega}} = a_l^{s(1)} a_l^{\omega(1)'} \tag{B.69}$$

because $\mathbf{a}^{h(0)} = \mathbf{0}$and $\mathbf{s}^{(0)} = \mathbf{0}$. The recursive expression for the derivative of $E^{(t)}$ with respect to b_j^{ω} follows from (B.26) and (B.66)–(B.70)

$$\frac{\partial E^{(t)}}{\partial b_j^{\omega}} = \sum_{l=1}^{J} \frac{\partial E^{(t)}}{\partial a_l^{h(t)}} \frac{\partial a_l^{h(t)}}{\partial b_j^{\omega}}. \tag{B.70}$$

The partial derivative of the hidden activation $a_m^{h(t)}$ with respect to weight v_{jl}^{ω} follows from replacing every occurrence of subscript l with m and w_{ji}^{γ} with v_{jl}^{ω} in (B.33)

$$\frac{\partial a_m^{h(t)}}{\partial v_{jl}^{\omega}} = a_m^{\omega(t)} a_m^{s(t)'} \frac{\partial s_m^{(t)}}{\partial v_{jl}^{\omega}} + a_m^{s(t)} a_m^{\omega(t)'} \left(a_m^{h(t-1)} + \sum_{p=1}^{J} v_{mp}^{\omega} \frac{\partial a_p^{h(t-1)}}{\partial v_{jl}^{\omega}} \right) \tag{B.71}$$

and the derivative of $s_m^{(t)}$ follows from replacing the subscript l with m and w_{ji}^{γ} with v_{jl}^{ω} in (B.35)

$$\frac{\partial s_m^{(t)}}{\partial v_{jl}^{\omega}} = a_m^{c(t)} a_m^{\gamma(t)'} \left(\sum_{p=1}^{J} v_{jp}^{\gamma} \frac{\partial a_p^{h(t-1)}}{\partial v_{jl}^{\omega}} \right) + a_m^{\gamma(t)} a_m^{c(t)'} \sum_{p=1}^{J} v_{jp}^{h} \frac{\partial a_p^{h(t-1)}}{\partial v_{jl}^{\omega}} + a_m^{\phi(t)} \frac{\partial s_m^{(t-1)}}{\partial v_{jl}^{\omega}} + s_m^{(t-1)} a_m^{\phi(t)'} \left(\sum_{p=1}^{J} v_{jp}^{\phi} \frac{\partial a_m^{h(t-1)}}{\partial v_{jl}^{\omega}} \right). \tag{B.72}$$

Therefore we have the following initial conditions:

$$\frac{\partial s_m^{(0)}}{\partial v_{jl}^{\omega}} = \frac{\partial s_m^{(1)}}{\partial v_{jl}^{\omega}} = \frac{\partial s_m^{(2)}}{\partial v_{jl}^{\omega}} = 0 \tag{B.73}$$

$$\frac{\partial a_m^{h(0)}}{\partial v_{jl}^{\omega}} = \frac{\partial a_m^{h(1)}}{\partial v_{jl}^{\omega}} = 0 \tag{B.74}$$

$$\frac{\partial a_m^{h(2)}}{\partial v_{jl}^{\omega}} = a_m^{s(1)} a_m^{\omega(1)'} a_l^{h(1)}. \tag{B.75}$$

The recursive expression for the derivative of $E^{(t)}$ with respect to v_{jl}^{ϕ} follows from Eqs. (B.26) and (B.71)–(B.76)

$$\frac{\partial E^{(t)}}{\partial v_{jl}^{\phi}} = \sum_{m=1}^{J} \frac{\partial E^{(t)}}{\partial a_m^{h(t)}} \frac{\partial a_m^{h(t)}}{\partial v_{jl}^{\phi}}. \tag{B.76}$$

The partial derivative for the weights connecting the input units to the hidden units follows from replacing w_{ji}^{γ} with w_{ji}^{x} in (B.33). Therefore the partial derivative of $a_l^{h(t)}$ with respect to the weight w_{ji}^{x} is as follows:

$$\frac{\partial a_l^{h(t)}}{\partial w_{ji}^{x}} = a_l^{\omega(t)} a_l^{s(t)'} \frac{\partial s_l^{(t)}}{\partial w_{ji}^{x}} + a_l^{s(t)} a_l^{\omega(t)'} \sum_{m=1}^{J} v_{lm}^{\gamma} \frac{\partial a_m^{h(t-1)}}{\partial w_{ji}^{x}} \tag{B.77}$$

and the derivative of $s_l^{(t)}$ with respect to w_{ji}^{x} comes from replacing w_{ji}^{γ} with w_{ji}^{x} is

$$\begin{gathered}\frac{\partial s_l^{(t)}}{\partial w_{ji}^{x}} = a_l^{c(t)} a_l^{\gamma(t)'} \sum_{m=1}^{J} v_{lm}^{\gamma} \frac{\partial a_m^{h(t-1)}}{\partial w_{ji}^{x}} + \\ a_l^{\gamma(t)} a_l^{c(t)'} \left(a_i^{x(t)} \delta(j-l) + \sum_{m=1}^{J} v_{lm}^{h} \frac{\partial a_m^{h(t-1)}}{\partial w_{ji}^{x}} \right) + \\ a_l^{\phi(t)} \frac{\partial s_l^{(t-1)}}{\partial w_{ji}^{x}} + s_l^{(t-1)} a_l^{\phi(t)'} \sum_{m=1}^{J} v_{lm}^{\phi} \frac{\partial a_m^{h(t-1)}}{\partial w_{ji}^{x}}.\end{gathered} \tag{B.78}$$

Therefore we have the following initial conditions:

$$\frac{\partial s_l^{(0)}}{\partial w_{ji}^{x}} = \frac{\partial a_l^{h(0)}}{\partial w_{ji}^{x}} = 0 \tag{B.79}$$

$$\frac{\partial s_l^{(1)}}{\partial w_{ji}^{x}} = a_j^{\gamma(1)}\ a_j^{c(1)'}\ a_i^{x(1)} \tag{B.80}$$

$$\frac{\partial a_l^{h(1)}}{\partial w_{ji}^{x}} = a_l^{\omega(1)}\ a_j^{s(1)'}\ a_j^{\gamma(1)}\ a_j^{c(1)'}\ a_i^{x(1)}. \tag{B.81}$$

The recursive expression for the derivative of $E^{(t)}$ with respect to w_{ji}^{x} follows from (B.26) and (B.77)–(B.82)

$$\frac{\partial E^{(t)}}{\partial w_{ji}^{x}} = \sum_{l=1}^{J} \frac{\partial E^{(t)}}{\partial a_l^{h(t)}} \frac{\partial a_l^{h(t)}}{\partial w_{ji}^{x}}. \tag{B.82}$$

The derivative of $a_l^{h(t)}$ and $s_l^{(t)}$ with respect to the forget bias b_j^{x} follows from replacing $a_i^{x(t)}$ with 1 and w_{ji}^{x} with b_j^{x} in (B.77)–(B.81). We have

$$\frac{\partial a_l^{h(t)}}{\partial b_j^{x}} = a_l^{\omega(t)} a_l^{s(t)'} \frac{\partial s_l^{(t)}}{\partial b_j^{x}} + a_l^{s(t)} a_l^{\omega(t)'} \sum_{m=1}^{J} v_{lm}^{\omega} \frac{\partial a_m^{h(t-1)}}{\partial b_j^{x}} \tag{B.83}$$

$$\frac{\partial s_l^{(t)}}{\partial b_j^{x}} = a_l^{c(t)} a_l^{\gamma(t)'} \sum_{m=1}^{J} v_{lm}^{\gamma} \frac{\partial a_m^{h(t-1)}}{\partial b_j^{x}} + a_l^{\gamma(t)} a_l^{c(t)'} \left(1 + \sum_{m=1}^{J} v_{lm}^{h} \frac{\partial a_m^{h(t-1)}}{\partial b_j^{x}} \right) + a_l^{\phi(t)} \frac{\partial s_l^{(t-1)}}{\partial b_j^{x}} + s_l^{(t-1)} a_l^{\phi(t)'} \sum_{m=1}^{J} v_{lm}^{\phi} \frac{\partial a_m^{h(t-1)}}{\partial b_j^{x}}. \tag{B.84}$$

So we have the following initial conditions:

$$\frac{\partial s_l^{(0)}}{\partial b_j^{\gamma}} = \frac{\partial a_l^{h(0)}}{\partial b_j^{\gamma}} = 0 \tag{B.85}$$

$$\frac{\partial s_l^{(1)}}{\partial w_{ji}^{x}} = a_j^{\gamma(1)}\ a_j^{c(1)'} \tag{B.86}$$

$$\frac{\partial a_l^{h(1)}}{\partial w_{ji}^{x}} = a_l^{\omega(1)}\ a_j^{s(1)'}\ a_j^{\gamma(1)}\ a_j^{c(1)'}. \tag{B.87}$$

The recursive expression for the partial derivative of $E^{(t)}$ with respect to b_j^{x} follows from (B.83)–(B.88)

$$\frac{\partial E^{(t)}}{\partial b_j^{x}} = \sum_{l=1}^{J} \frac{\partial E^{(t)}}{\partial a_l^{h(t)}} \frac{\partial a_l^{h(t)}}{\partial b_j^{x}}. \tag{B.88}$$

The partial derivative of the hidden activation $a_m^{h(t)}$ with respect to weight v_{jl}^{h} follows from replacing subscript l with m and w_{ji}^{γ} with v_{jl}^{h} in (B.33). This gives the following:

$$\frac{\partial a_l^{h(t)}}{\partial v_{jl}^h} = a_l^{\omega(t)} a_l^{s(t)'} \frac{\partial s_l^{(t)}}{\partial v_{jl}^h} + a_l^{s(t)} a_l^{\omega(t)'} \sum_{m=1}^{J} v_{lm}^{\omega} \frac{\partial a_m^{h(t-1)}}{\partial v_{jl}^h} \tag{B.89}$$

and the derivative of $s_m^{(t)}$ with respect v_{jl}^h is the following:

$$\frac{\partial s_m^{(t)}}{\partial v_{jl}^h} = a_m^{c(t)} a_m^{\gamma(t)'} \sum_{p=1}^{J} v_{mp}^{\gamma} \frac{\partial a_p^{h(t-1)}}{\partial v_{jl}^h} + a_m^{\gamma(t)} a_m^{c(t)'} \sum_{p=1}^{J} \left(a_p^{h(t-1)} \delta(m-p) + v_{mp}^h \frac{\partial a_p^{h(t-1)}}{\partial v_{jl}^h} \right) + a_m^{\phi(t)} \frac{\partial s_m^{(t-1)}}{\partial v_{jl}^h} + s_m^{(t-1)} a_m^{\phi(t)'} \left(\sum_{p=1}^{J} v_{mp}^{\phi} \frac{\partial a_m^{h(t-1)}}{\partial v_{jl}^h} \right). \tag{B.90}$$

Therefore we have the following initial conditions:

$$\frac{\partial s_m^{(0)}}{\partial v_{jl}^h} = \frac{\partial s_m^{(1)}}{\partial v_{jl}^h} = 0 \tag{B.91}$$

$$\frac{\partial a_m^{h(0)}}{\partial v_{jl}^h} = \frac{\partial a_m^{h(1)}}{\partial v_{jl}^h} = 0. \tag{B.92}$$

The recursive expression for the derivative of $E^{(t)}$ with respect to v_{jl}^h follows from (B.26) and (B.89)–(B.93)

$$\frac{\partial E^{(t)}}{\partial v_{jl}^h} = \sum_{m=1}^{J} \frac{\partial E^{(t)}}{\partial a_m^{h(t)}} \frac{\partial a_m^{h(t)}}{\partial v_{jl}^h}. \tag{B.93}$$

RBP uses gradient descent or any of its variants for training. The update rule for weight $\theta \in \{w_{ji}^x, v_{jl}^h, b_j^x, w_{ji}^{\phi}, v_{jl}^{\phi}, b_j^{\phi}, w_{ji}^{\gamma}, v_{jl}^{\gamma}, b_j^{\gamma}, w_{ji}^{\omega}, v_{jl}^{\omega}, b_j^{\omega}, u_{ji}^y, b_j^y\}$ is as follows:

$$\theta^{n+1} = \theta^n - \eta \frac{\partial E}{\partial \theta}\Big|_{\theta=\theta^n} = \theta^n - \eta \sum_{t=t_0}^{T} \frac{\partial E^{(t)}}{\partial \theta}\Big|_{\theta=\theta^n} \tag{B.94}$$

where η is the learning rate, θ^n is the weight after n training epochs, and $t_0 \in \{1, 2, \ldots, T\}$. The above partial derivatives form the update or learning rules and the RBP training algorithm for LSTM-RNN.

Appendix C. Recurrent Backpropagation Training with Gated Recurrent Unit (GRU)

This section develops the RBP learning algorithm for a GRU recurrent networks [21]. The GRU network consists of I input neurons, J hidden neurons, and K output neurons. The GRU network captures the time dependency of the input data where the time index t takes values in $\{1, 2, \ldots.., T\}$. We now define the parameters for a GRU network.

The $J \times I$ matrices $\mathbf{W}^z$ and $\mathbf{W}^r$ connect the input unit $\mathbf{a}^{x(t)}$ to the respective update and reset gates. The $J \times J$ matrices $\mathbf{V}^z$ and $\mathbf{V}^r$ connect the hidden unit $\mathbf{a}^{h(t-1)}$ to the respective update and reset gates. The $J \times I$ matrix $\mathbf{W}^x$ connects $\mathbf{a}^{x(t)}$ to the input to the hidden unit $\mathbf{o}^{c(t)}$ and $\mathbf{b}^x$ is the bias to for the hidden unit. The $J \times J$ matrix $\mathbf{V}^h$ connects the previous hidden unit $\mathbf{a}^{h(t-1)}$ to $\mathbf{o}^{c(t)}$. The $K \times J$ matrix $\mathbf{U}^y$ connects the hidden unit $\mathbf{a}^{h(t)}$ to the output layer $\mathbf{a}^{y(t)}$ and $\mathbf{b}^y$ is the bias of the output layer. The initial value for the hidden state $\mathbf{a}^{h(0)}$ is $\mathbf{0}$.

C.1. Forward Pass

Forward pass over an GRU-RNN is the propagation of input $\mathbf{x}^{(t)}$ from the input layer to the output layer through the hidden units. The input uses an identity activation so $a_i^{x(t)} = x_i^{(t)}$ where $x_i^{(t)}$ is the input of the ith input neuron at time t. The input $o_j^{z(t)}$ to the jth update gate unit is

$$o_j^{z(t)} = \sum_{i=1}^{I} w_{ji}^z a_i^{x(t)} + \sum_{l=1}^{J} v_{jl}^z a_l^{h(t-1)} + b_j^z \tag{C.1}$$

where b_j^z is the bias of the update gate. The corresponding activation $a_j^{z(t)}$ of the update gate unit is the logistic sigmoid function

$$a_j^{z(t)} = \frac{1}{1 + \exp^{-o_j^{z(t)}}}. \tag{C.2}$$

The input $o_j^{r(t)}$ to the jth reset gate unit is

$$o_j^{r(t)} = \sum_{i=1}^{I} w_{ji}^r a_i^{x(t)} + \sum_{l=1}^{J} v_{jl}^r a_l^{h(t-1)} + b_j^r \tag{C.3}$$

where b_j^r is the bias of the jth reset gate unit. The activation $a_j^{r(t)}$ of the reset gate unit is

$$a_j^{r(t)} = \frac{1}{1 + \exp^{-o_j^{r(t)}}}. \tag{C.4}$$

The input $o_j^{c(t)}$ to the j^{th} hidden unit is

$$o_j^{c(t)} = \sum_{i=1}^{I} w_{ji}^x a_i^{x(t)} + \sum_{l=1}^{J} v_{jl}^h \left(a_l^{h(t-1)} a_j^{r(t)}\right) + b_j^x \tag{C.5}$$

where b_j^x is the bias of the j^{th} hidden unit. The corresponding activation $a_j^{c(t)}$ is

$$a_j^{c(t)} = \tanh\left(o_j^{c(t)}\right) = \frac{\exp^{o_j^{c(t)}} - \exp^{-o_j^{c(t)}}}{\exp^{o_j^{c(t)}} + \exp^{-o_j^{c(t)}}}. \tag{C.6}$$

The hidden memory $a_j^{h(t)}$ at time t is

$$a_j^{h(t)} = \left(a_j^{c(t)} a_j^{z(t)}\right) + \left(\left(1 - a_j^{z(t)}\right) a_j^{h(t-1)}\right). \tag{C.7}$$

The input $o_k^{y(t)}$ to the k^{th} output neuron is

$$o_k^{y(t)} = \sum_{j=1}^{J} u_{kj}^y a_j^{y(t)} + b_k^y. \tag{C.8}$$

The corresponding output activation is the same as Eq. (B.13) for a classifier network. Regression network uses identity activation at the output layer from (B.14).

C.2. Backward Pass

We now present the update for RBP with GRU-RNN. The error function for a classifier is a cross-entropy and this follows from Eq. (B.15). The error for a regression network with GRU architecture is a sum of squared error and this follows from (B.27).

The update rules for some weights are similar with GRU-RNN and LSTM-RNN. The update rules for u_{kj}^y and b_k^y follow from (B.22) and (B.23). The derivatives $E^{(t)'}$ and $a_j^{c(t)'}$ follow from the definitions in (B.26) and (B.28). The derivative of the activations for the update and reset gates are

$$a_j^{z(t)'} = \frac{\partial a_j^{z(t)}}{\partial o_j^{z(t)}} = a_j^{z(t)} \left(1 - a_j^{z(t)}\right) \tag{C.9}$$

$$a_j^{r(t)'} = \frac{\partial a_j^{r(t)}}{\partial o_j^{r(t)}} = a_j^{r(t)} \left(1 - a_j^{r(t)}\right). \tag{C.10}$$

The derivative of activation $a_l^{h(t)}$ with respect to the weight w_{ji}^x connecting the input unit and hidden memory is

$$\frac{\partial a_l^{h(t)}}{\partial w_{ji}^x} = \left(1 - a_l^{z(t)}\right) \frac{\partial a_l^{h(t-1)}}{\partial w_{ji}^x} - a_l^{h(t-1)} \frac{\partial a_l^{z(t)}}{\partial w_{ji}^x} + a_l^{z(t)} \frac{\partial a_l^{c(t)}}{\partial w_{ji}^x} + a_l^{c(t)} \frac{\partial a_l^{z(t)}}{\partial w_{ji}^x} \tag{C.11}$$

$$= \left(1 - a_l^{z(t)}\right) \frac{\partial a_l^{h(t-1)}}{\partial w_{ji}^x} - \left(a_l^{h(t-1)} - a_l^{c(t)}\right) a_l^{z(t)'} \sum_{m=1}^{J} v_{lm}^z \frac{\partial a_m^{h(t-1)}}{\partial w_{ji}^x} + a_l^{z(t)} a_l^{c(t)'} \left[a_i^{x(t)} + \sum_{m=1}^{J} v_{lm}^h \left(a_l^{r(t)} \frac{\partial a_m^{h(t-1)}}{\partial w_{ji}^x} + a_m^{h(t-1)} a_l^{r(t)'} \sum_{p=1}^{J} v_{lp}^r \frac{\partial a_p^{h(t-1)}}{\partial w_{ji}^x}\right)\right]. \tag{C.12}$$

Therefore we have the following initial conditions:

$$\frac{\partial a_l^{h(0)}}{\partial w_{ji}^x} = 0 \tag{C.13}$$

$$\frac{\partial a_l^{h(1)}}{\partial w_{ji}^x} = a_l^{z(1)} \, a_l^{c(1)'} \, a_l^{x(1)}. \tag{C.14}$$

The recursive expression for the partial derivative of $E^{(t)}$ with respect to w_{ji}^x follows from (C.12)–(C.15)

$$\frac{\partial E^{(t)}}{\partial w_{ji}^x} = \sum_{l=1}^{J} \frac{\partial E^{(t)}}{\partial a_l^{h(t)}} \frac{\partial a_l^{h(t)}}{\partial w_{ji}^x}. \tag{C.15}$$

The derivative of $a_l^{h(t)}$ with respect to the bias b_j^x follows from replacing $a_i^{x(t)}$ with 1 and w_{ji}^x with b_j^x in (C.12)–(C.15). Therefore we have

$$\frac{\partial a_l^{h(t)}}{\partial b_j^x} = \left(1 - a_l^{z(t)}\right)\frac{\partial a_l^{h(t-1)}}{\partial b_j^x} - \left(a_l^{h(t-1)} - a_l^{c(t)}\right)a_l^{z(t)'}\sum_{m=1}^{J} v_{lm}^z \frac{\partial a_m^{h(t-1)}}{\partial b_j^x} + a_l^{z(t)}a_l^{c(t)'}\left[1 + \sum_{m=1}^{J} v_{lm}^h\left(a_l^{r(t)}\frac{\partial a_m^{h(t-1)}}{\partial b_j^x} + a_m^{h(t-1)}a_l^{r(t)'}\sum_{p=1}^{J} v_{lp}^r \frac{\partial a_p^{h(t-1)}}{\partial b_j^x}\right)\right]. \tag{C.16}$$

We have the following initial conditions:

$$\frac{\partial a_l^{h(0)}}{\partial b_j^x} = 0 \tag{C.17}$$

$$\frac{\partial a_l^{h(1)}}{\partial b_j^x} = a_l^{z(1)}\ a_l^{c(1)'}. \tag{C.18}$$

The recursive expression for the partial derivative of $E^{(t)}$ with respect to b_j^x follows from (C.16)–(C.19)

$$\frac{\partial E^{(t)}}{\partial w_{ji}^x} = \sum_{l=1}^{J}\frac{\partial E^{(t)}}{\partial a_l^{h(t)}}\ \frac{\partial a_l^{h(t)}}{\partial w_{ji}^x}. \tag{C.19}$$

The derivative of $a_m^{h(t)}$ with respect to v_{jl}^h is as follows:

$$\frac{\partial a_m^{h(t)}}{\partial v_{jl}^h} = \left(1 - a_m^{z(t)}\right)\frac{\partial a_m^{h(t-1)}}{\partial v_{jl}^h} - a_m^{h(t-1)}\frac{\partial a_m^{z(t)}}{\partial v_{jl}^h} + a_m^{z(t)}\frac{\partial a_m^{c(t)}}{\partial v_{jl}^h} + a_m^{c(t)}\frac{\partial a_m^{z(t)}}{\partial v_{jl}^h} \tag{C.20}$$

$$= \left(1 - a_m^{z(t)}\right)\frac{\partial a_m^{h(t-1)}}{\partial v_{jl}^h} - \left(a_m^{h(t-1)} - a_m^{c(t)}\right)a_m^{z(t)'}\sum_{p=1}^{J} v_{mp}^z \frac{\partial a_p^{h(t-1)}}{\partial v_{jl}^h} + a_m^{z(t)}a_m^{c(t)'}\left[a_m^{h(t-1)}a_m^{r(t)} + \sum_{p=1}^{J} v_{mp}^h\left(a_p^{r(t)}\frac{\partial a_p^{h(t-1)}}{\partial v_{jl}^h} + a_p^{h(t-1)}a_p^{r(t)'}\sum_{q=1}^{J} v_{mq}^r \frac{\partial a_q^{h(t-1)}}{\partial v_{jl}^h}\right)\right]. \tag{C.21}$$

Therefore we have the following initial conditions:

$$\frac{\partial a_m^{h(0)}}{\partial v_{jl}^h} = \frac{\partial a_m^{h(1)}}{\partial v_{jl}^h} = 0 \tag{C.22}$$

$$\frac{\partial a_m^{h(2)}}{\partial v_{jl}^h} = a_m^{z(2)}\ a_m^{c(1)'}\ a_l^{h(1)}\ a_j^{r(2)} \tag{C.23}$$

and the recursive expression for computing the derivative of $E^{(t)}$ with respect to v_{jl}^h for $t \in \{1, \ldots, T\}$ follows from (C.21)–(C.24)

$$\frac{\partial E^{(t)}}{\partial v_{jl}^h} = \frac{\partial E^{(t)}}{\partial a_j^{h(t)}}\ \frac{\partial a_j^{h(t)}}{\partial v_{jl}^h}. \tag{C.24}$$

The derivative of activation $a_l^{h(t)}$ with respect to weight w_{ji}^r connecting the reset gate to the input unit follows from replacing w_{ji}^x in Eq. (C.11) with w_{ji}^r. This gives the following:

$$\frac{\partial a_l^{h(t)}}{\partial w_{ji}^r} = \left(1 - a_l^{z(t)}\right)\frac{\partial a_l^{h(t-1)}}{\partial w_{ji}^r} - \left(a_l^{h(t-1)} - a_l^{c(t)}\right)a_l^{z(t)'}\sum_{m=1}^{J} v_{lm}^z \frac{\partial a_m^{h(t-1)}}{\partial w_{ji}^r} + a_l^{z(t)}a_l^{c(t)'}\left[\sum_{m=1}^{J} v_{lm}^h\left(a_l^{r(t)}\frac{\partial a_m^{h(t-1)}}{\partial w_{ji}^r} + a_m^{h(t-1)}a_l^{r(t)'}\sum_{p=1}^{J} v_{lp}^r \frac{\partial a_p^{h(t-1)}}{\partial w_{ji}^r}\right)\right]. \tag{C.25}$$

Therefore we have the following initial condition:

$$\frac{\partial a_l^{h(0)}}{\partial w_{ji}^r} = 0. \tag{C.26}$$

The recursive expression for the partial derivative of $E^{(t)}$ with respect to w_{ji}^x follows from (C.25)–(C.27)

$$\frac{\partial E^{(t)}}{\partial w_{ji}^r} = \sum_{l=1}^{J}\frac{\partial E^{(t)}}{\partial a_l^{h(t)}}\ \frac{\partial a_l^{h(t)}}{\partial w_{ji}^r}. \tag{C.27}$$

The derivative of activation $a_l^{h(t)}$ with respect to b_j^r follows from replacing w_{ji}^r with b_j^r and $a_i^{x(t)}$ with 1 in (C.25). We have

$$\frac{\partial a_l^{h(t)}}{\partial b_j^r} = \left(1 - a_l^{z(t)}\right)\frac{\partial a_l^{h(t-1)}}{\partial b_j^r} - \left(a_l^{h(t-1)} - a_l^{c(t)}\right)a_l^{z(t)'}\sum_{m=1}^{J} v_{lm}^z \frac{\partial a_m^{h(t-1)}}{\partial b_j^r} + a_l^{z(t)}a_l^{c(t)'}\left[\sum_{m=1}^{J} v_{lm}^h\left(a_l^{r(t)}\frac{\partial a_m^{h(t-1)}}{\partial b_j^r} + a_m^{h(t-1)}a_l^{r(t)'}\sum_{p=1}^{J} v_{lp}^r \frac{\partial a_p^{h(t-1)}}{\partial b_j^r}\right)\right]. \tag{C.28}$$

The initial condition is

$$\frac{\partial a_l^{h(0)}}{\partial b_j^r} = 0. \tag{C.29}$$

The recursive expression for computing the derivative of $E^{(t)}$ with respect to the bias b_j^r follows from (C.28)–(C.30)

$$\frac{\partial E^{(t)}}{\partial b_l^r} = \sum_{l=1}^{J} \frac{\partial E^{(t)}}{\partial a_l^{h(t)}} \frac{\partial a_l^{h(t)}}{\partial b_j^r}. \tag{C.30}$$

The derivative of $a_m^{h(t)}$ with respect to v_{jl}^r follows from replacing w_{ji}^x with v_{jl}^r and $a_l^{h(t)}$ with $a_m^{h(t)}$ in (C.11)

$$\frac{\partial a_m^{h(t)}}{\partial v_{jl}^r} = \left(1 - a_m^{z(t)}\right) \frac{\partial a_m^{h(t-1)}}{\partial v_{jl}^r} - \left(a_m^{h(t-1)} - a_m^{c(t)}\right) a_m^{z(t)'} \sum_{p=1}^{J} v_{mp}^z \frac{\partial a_p^{h(t-1)}}{\partial v_{jl}^r} + a_m^{z(t)} a_m^{c(t)'} \left[\sum_{p=1}^{J} v_{mp}^h \left(a_m^{r(t)} \frac{\partial a_p^{h(t-1)}}{\partial v_{jl}^r} + a_p^{h(t-1)} a_m^{r(t)'} \left(a_l^{h(t-1)} + \sum_{q=1}^{J} v_{mq}^h \frac{\partial a_q^{h(t-1)}}{\partial v_{jl}^r} \right) \right) \right]. \tag{C.31}$$

This gives the following:

$$\frac{\partial a_j^{h(0)}}{\partial v_{jl}^r} = 0. \tag{C.32}$$

The recursive expression for computing the derivative of $E^{(t)}$ with respect to the weight v_{jl}^r follows from (C.32)–(C.33)

$$\frac{\partial E^{(t)}}{\partial v_{jl}^r} = \sum_{j=1}^{J} \frac{\partial E^{(t)}}{\partial a_j^{h(t)}} \frac{\partial a_j^{h(t)}}{\partial v_{jl}^r}. \tag{C.33}$$

The derivative of activation $a_l^{h(t)}$ with respect to w_{ji}^z follows from replacing w_{ji}^x in (C.11) with w_{ji}^z. We have the following:

$$\frac{\partial a_l^{h(t)}}{\partial w_{ji}^z} = \left(1 - a_l^{z(t)}\right) \frac{\partial a_l^{h(t-1)}}{\partial w_{ji}^z} - \left(a_l^{h(t-1)} - a_l^{c(t)}\right) a_l^{z(t)'} \left(a_i^{x(t)} + \sum_{m=1}^{J} v_{jm}^z \frac{\partial a_m^{h(t-1)}}{\partial w_{ji}^z} \right) + a_l^{z(t)} a_l^{c(t)'} \left(\sum_{m=1}^{J} v_{lm}^h a_m^{r(t)} \frac{\partial a_m^{h(t-1)}}{\partial w_{ji}^z} \right). \tag{C.34}$$

This gives the following:

$$\frac{\partial a_l^{h(0)}}{\partial w_{ji}^z} = 0 \tag{C.35}$$

$$\frac{\partial a_l^{h(0)}}{\partial w_{ji}^z} = a_l^{c(1)}\ a_l^{z(1)'}\ a_i^{x(1)}. \tag{C.36}$$

The recursive expression for computing the derivative of $E^{(t)}$ with respect to the weight w_{ji}^z follows from (C.34)–(C.37)

$$\frac{\partial E^{(t)}}{\partial w_{ji}^z} = \sum_{j=1}^{J} \frac{\partial E^{(t)}}{\partial a_j^{h(t)}} \frac{\partial a_j^{h(t)}}{\partial w_{ji}^z}. \tag{C.37}$$

The derivative of $a_l^{h(t)}$ with respect to b_j^z follows from replacing w_{ji}^z with b_j^z and $a_i^{x(t)}$ with 1 in Eq. (C.34). We have

$$\frac{\partial a_l^{h(t)}}{\partial b_j^z} = \left(1 - a_l^{z(t)}\right) \frac{\partial a_l^{h(t-1)}}{\partial b_j^z} - \left(a_l^{h(t-1)} - a_l^{c(t)}\right) a_l^{z(t)'} \left(1 + \sum_{m=1}^{J} v_{jm}^z \frac{\partial a_m^{h(t-1)}}{\partial b_j^z} \right) + a_l^{z(t)} a_l^{c(t)'} \left(\sum_{m=1}^{J} v_{lm}^h a_m^{r(t)} \frac{\partial a_m^{h(t-1)}}{\partial b_j^z} \right). \tag{C.38}$$

We have the following:

$$\frac{\partial a_l^{h(0)}}{\partial b_j^z} = \frac{\partial a_l^{h(1)}}{\partial b_j^z} = 0. \tag{C.39}$$

The recursive expression for computing the derivative of $E^{(t)}$ with respect to the weight w_{ji}^z follows from (C.39)–(C.40)

$$\frac{\partial E^{(t)}}{\partial b_j^z} = \sum_{j=1}^{J} \frac{\partial E^{(t)}}{\partial a_l^{h(t)}} \frac{\partial a_l^{h(t)}}{\partial b_j^z}. \tag{C.40}$$

The derivative of activation $a_m^{h(t)}$ with respect to v_{jl}^z follows from replacing v_{jl}^z with v_{jl}^z in (C.11). We have the following:

$$\frac{\partial a_m^{h(t)}}{\partial v_{jl}^z} = \left(1 - a_m^{z(t)}\right) \frac{\partial a_m^{h(t-1)}}{\partial v_{jl}^z} - \left(a_m^{h(t-1)} - a_m^{c(t)}\right) a_m^{z(t)'} \left(a_m^{h(t-1)} + \sum_{p=1}^{J} v_{mp}^z \frac{\partial a_p^{h(t-1)}}{\partial v_{jl}^z} \right) + a_m^{z(t)} a_m^{c(t)'} \left(\sum_{p=1}^{J} v_{mp}^h a_p^{r(t)} \frac{\partial a_p^{h(t-1)}}{\partial v_{jl}^z} \right). \tag{C.41}$$

The initial condition is

$$\frac{\partial a_m^{h(0)}}{\partial v_{jl}^z} = \frac{\partial a_m^{h(1)}}{\partial v_{jl}^z} = 0. \tag{C.42}$$

The recursive expression for computing the derivative of $E^{(t)}$ with respect ot the weight v^z_{jl} follows from (C.41)–(C.43)

$$\frac{\partial E^{(t)}}{\partial v^z_{jl}} = \sum_{m=1}^{J} \frac{\partial E^{(t)}}{\partial a^{h(t)}_m} \frac{\partial a^{h(t)}_m}{\partial v^z_{jl}}. \tag{C.43}$$

RBP uses gradient descent or any of its variants for training. The update rule for the following network parameters $\theta \in \{w^x_{ji}, v^h_{jl}, b^x_j, w^z_{ji}, v^z_{jl}, b^z_j, w^r_{ji}, v^r_{jl}, b^r_j, u^y_{ji}, b^y_j\}$ is as follows:

$$\theta^{n+1} = \theta^n - \eta \frac{\partial E}{\partial \theta}\Big|_{\theta=\theta^n} = \theta^{(n)} - \eta \sum_{t=t_0}^{T} \frac{\partial E^{(t)}}{\partial \theta}\Big|_{\theta=\theta^n} \tag{C.44}$$

where η is the learning rate, θ^n is the weight after n training epochs, and $t_0 \in \{1, 2, \ldots, T\}$. These partial derivatives form the update rules for training a RNN with a GRU architecture using the RBP algorithm.

References

[1] P.J. Werbos, Backpropagation through time: what it does and how to do it, Proc. IEEE 78 (10) (1990) 1550–1560, https://doi.org/10.1109/5.58337.
[2] P.J. Werbos, Generalization of backpropagation with application to a recurrent gas market model, Neural Networks 1 (4) (1988) 339–356, https://doi.org/10.1016/0893-6080(88)90007-X.
[3] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Computation 9 (8) (1997) 1735–1780, https://doi.org/10.1162/neco.1997.9.8.1735.
[4] M.C. Mozer, A focused back-propagation algorithm for temporal pattern recognition, Complex systems 3 (4) (1989) 349–381.
[5] M.C. Mozer, A focused backpropagation algorithm for temporal, Backpropagation: Theory, architectures, and applications 137.
[6] R. Reed, S. Oh, R. Marks, et al., Regularization using jittered training data, in: International joint conference on neural networks, Vol. 3, 1992, pp. 147–152. doi:10.1109/IJCNN.1992.227178.
[7] C.M. Bishop, Training with noise is equivalent to Tikhonov regularization, Neural computation 7 (1) (1995) 108–116, https://doi.org/10.1162/neco.1995.7.1.108.
[8] R. Reed, R. Marks, S. Oh, Similarities of error regularization, sigmoid gain scaling, target smoothing, and training with jitter, IEEE Trans. Neural Networks 6 (3) (1995) 529–538, https://doi.org/10.1109/72.377960.
[9] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back-propagating errors, Nature 323 (6088) (1986) 533–536, https://doi.org/10.1038/323533a0.
[10] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444, https://doi.org/10.1038/nature14539.
[11] M. Jordan, T. Mitchell, Machine learning: trends, perspectives, and prospects, Science 349 (2015) 255–260, https://doi.org/10.1126/science.aaa8415.
[12] G. Hinton, Deep learning–a technology with the potential to transform health care, Journal of the American Medical Association 320 (11) (2018) 1101–1102, https://doi.org/10.1001/jama.2018.11100.
[13] J. Schmidhuber, Deep learning in neural networks: An overview, Neural Networks 61 (2015) 85–117, https://doi.org/10.1016/j.neunet.2014.09.003.
[14] K.P. Murphy, Machine learning: a probabilistic perspective, Adaptive computation and machine learning series, MIT press, 2012.
[15] K. Audhkhasi, O. Osoba, B. Kosko, Noise-enhanced convolutional neural networks, Neural Networks 78 (2016) 15–23.
[16] F.A. Gers, J. Schmidhuber, F. Cummins, Learning to forget: Continual prediction with LSTM, Neural Computation 12 (10) (2000) 2451–2471, https://doi.org/10.1162/089976600300015015.
[17] K. Greff, R.K. Srivastava, J. Koutník, B.R. Steunebrink, J. Schmidhuber, LSTM: A search space odyssey, IEEE transactions on neural networks and learning systems 28 (10) (2016) 2222–2232, https://doi.org/10.1109/TNNLS.2016.2582924.
[18] S. Hochreiter, J. Schmidhuber, LSTM can solve hard long time lag problems, Advances in neural information processing systems (1996) 473–479.
[19] L. Arras, J. Arjona-Medina, M. Widrich, G. Montavon, M. Gillhofer, K.-R. Müller, S. Hochreiter, W. Samek, Explaining and interpreting lstms, in: Explainable ai: Interpreting, explaining and visualizing deep learning, Vol. 11700, Springer, 2019, pp. 211–238, https://doi.org/10.1007/978-3-030-28954-6_11.
[20] A. Graves, A.-R. Mohamed, 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, https://doi.org/10.1109/ICASSP.2013.6638947.
[21] C. Junyoung, G. Caglar, C. Kyunghyun, B. Yoshua, Gated feedback recurrent neural networks, in: Proceedings of the 32nd International Conference on Machine Learning, Vol. 37 of JMLR Workshop and Conference Proceedings, JMLR.org, 2015, pp. 2067–2075.
[22] J. Chung, C. Gulcehre, K. Cho, Y. Bengio, Empirical evaluation of gated recurrent neural networks on sequence modeling, arXiv preprint arXiv:1412.3555.
[23] D.M. Rodriguez, J. Ahmed, 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, IEEE Computer Society, 2008. doi:10.1109/CVPR.2008.4587727.
[24] K. Soomro, A.R. Zamir, Action recognition in realistic sports videos, in: Computer vision in sports, Springer, 2015, pp. 181–208.
[25] B. Kosko, Noise, Penguin, 2006.
[26] S. Mitaim, B. Kosko, Adaptive stochastic resonance, Proceedings of the IEEE 86 (11) (1998) 2152–2183, https://doi.org/10.1109/5.726785.
[27] A. Patel, B. Kosko, Stochastic Resonance in Continuous and Spiking Neurons with Levy Noise, IEEE Transactions on Neural Networks 19 (12) (2008) 1993–2008, https://doi.org/10.1109/TNN.2008.2005610.
[28] M. McDonnell, N. Stocks, C. Pearce, D. Abbott, Stochastic resonance: from suprathreshold stochastic resonance to stochastic signal quantization, Cambridge University Press, 2008.
[29] M. Wilde, B. Kosko, Quantum forbidden-interval theorems for stochastic resonance, Journal of Physical A: Mathematical Theory 42 (46). doi:10.1088/1751-8113/42/46/465309.
[30] A. Patel, B. Kosko, Error-probability noise benefits in threshold neural signal detection, Neural Networks 22 (5) (2009) 697–706, https://doi.org/10.1016/j.neunet.2009.06.044.
[31] A. Patel, B. Kosko, Optimal Mean-Square Noise Benefits in Quantizer-Array Linear Estimation, IEEE Signal Processing Letters 17 (12) (2010) 1005–1009, https://doi.org/10.1109/LSP.2010.2059376.
[32] A. Patel, B. Kosko, Noise Benefits in Quantizer-Array Correlation Detection and Watermark Decoding, IEEE Transactions on Signal Processing 59 (2) (2011) 488–505, https://doi.org/10.1109/TSP.2010.2091409.
[33] B. Franzke, B. Kosko, Noise Can Speed Convergence in Markov Chains, Physical Review E 84 (4) (2011), 041112 , https://doi.org/10.1103/PhysRevE.84.041112.
[34] A.R. Bulsara, R.D. Boss, E.W. Jacobs, Noise Effects in an Electronic Model of a Single Neuron, Biological Cybernetics 61 (1989) 211–222, https://doi.org/10.1007/BF00198768.
[35] L. Gammaitoni, P. Hänggi, P. Jung, F. Marchesoni, Stochastic Resonance, Reviews of Modern Physics 70 (1) (1998) 223–287, https://doi.org/10.1103/RevModPhys.70.223.
[36] F. Chapeau-Blondeau, X. Godivier, Theory of Stochastic Resonance in Signal Transmission by Static Nonlinear System, Physical Review E 55 (2) (1997) 1478–1495, https://doi.org/10.1103/PhysRevE.55.1478.
[37] S. Mitaim, B. Kosko, Noise-benefit forbidden-interval theorems for threshold signal detectors based on cross correlations, Physical Review E 90 (5) (2014), 052124 , https://doi.org/10.1103/PhysRevE.90.052124.
[38] B. Kosko, K. Audhkhasi, O. Osoba, Noise can speed backpropagation learning and deep bidirectional pretraining, Neural Networks 129 (2020) 359–384, https://doi.org/10.1016/j.neunet.2020.04.004.
[39] O. Osoba, S. Mitaim, B. Kosko, The noisy Expectation–Maximization algorithm, Fluctuation and Noise Letters 12 (3) (2013) 1350012–1–1350012–30. doi:10.1142/S0219477513500120.
[40] O. Osoba, B. Kosko, The noisy Expectation-Maximization algorithm for multiplicative noise injection, Fluctuation and Noise Letters 15 (01) (2016) 1650007, https://doi.org/10.1142/S0219477516500073.
[41] O. Adigun, B. Kosko, Noise-boosted bidirectional backpropagation and adversarial learning, Neural Networks 120 (2019) 9–31, https://doi.org/10.1016/j.neunet.2019.09.016.
[42] O. Adigun, B. Kosko, Using noise to speed up video classification with recurrent backpropagation, in: International Joint Conference on Neural Networks, IEEE, 2017, pp. 108–115, https://doi.org/10.1109/IJCNN.2017.7965843.
[43] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the inception architecture for computer vision, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 2818–2826. doi:10.1109/CVPR.2016.308.
[44] O. Adigun, B. Kosko, High capacity neural block classifiers with logistic neurons and random coding, in: 2020 International Joint Conference on Neural Networks (IJCNN), IEEE, 2020, pp. 1–9, https://doi.org/10.1109/IJCNN48605.2020.9207218.
[45] A.P. Dempster, N.M. Laird, D.B. Rubin, Maximum likelihood from incomplete data via the EM algorithm, Journal of the royal statistical society. Series B (methodological) (1977) 1–38. URL:http://www.jstor.org/stable/2984875.
[46] K. Audhkhasi, O. Osoba, B. Kosko, Noisy hidden Markov models for speech recognition, in: Neural Networks (IJCNN), The 2013 International Joint Conference on, IEEE, 2013, pp. 1–6, https://doi.org/10.1109/IJCNN.2013.6707088.

[47] G. Alex, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, J. Schmidhuber, A novel connectionist system for unconstrained handwriting recognition, IEEE Transactions on Pattern Analysis and Machine Intelligence (2009) 855–868.
[48] G. Alex, M. Liwicki, S. Fernandez, H. Bunke, J. Schmidhuber, A novel connectionist system for unconstrained handwriting recognition, Proceedings of the 20th International Conference on Neural Information Processing System (2007) 577–584.
[49] R.O. Duda, P.E. Hart, D.G. Stork (Eds.), Pattern Classification, Vol. 2nd, 2000.
[50] C.M. Bishop, Pattern recognition and machine learning, springer, 2006.
[51] R.V. Hogg, J. McKean, A.T. Craig, Introduction to Mathematical Statistics, (7th edition),, Pearson, 2013.
[52] Y. Perwej, A. Perwej, Forecasting of Indian Rupee/US Dollar currency exchange rate using artificial neural networks, International Journal of Computer Science, Engineering and Applications 2 (2).

Dr. Olaoluwa (Oliver) Adigun is a lecturer at the Department of Electrical and Computer Engineering at the University of Southern California. He received a Bachelor of Science degree in electronic and electrical engineering from Obafemi Awolowo University, Ife, Nigeria. He received a Master of Science degree and a Ph.D. degree in Electrical Engineering from the Department of Electrical and Computer Engineering, Signal and Image Processing Institute, University of Southern California, Los Angeles, CA, USA.

Dr. Bart Kosko is a professor of Electrical and Computer Engineering, and Law, at the University of Southern California. He is a Fellow of the IEEE and of the International Neural Network Society (INNS), and received the INNS Hebb Award for neural learning. He received degrees in philosophy, economics, applied mathematics, electrical engineering, and law. He is a past Director of USC's Signal and Image Processing Institute and a licensed attorney. He has published the textbooks Neural Networks and Fuzzy Systems, and Fuzzy Engineering, the trade books Fuzzy Thinking, Heaven in a Chip, and Noise, the edited volume Neural Networks and Signal Processing, the co-edited volume Intelligent Signal Processing, and the novels Nanotime and Cool Earth.