Received: 23 February 2023 | Revised: 6 May 2024 | Accepted: 29 June 2024

DOI: 10.1111/coin.12691

ORIGINAL ARTICLE

Computational Intelligence WILEY

Learning multi-modal recurrent neural networks with target propagation

Nikolay Manchev[1] | **Michael Spratling**[1,2]

[1]Department of Informatics, King's College, London, UK

[2]Department of Behavioural and Cognitive Sciences, University of Luxembourg, Esch-sur-Alzette, Luxembourg

Correspondence
Nikolay Manchev, Department of Informatics, King's College, London, WC2B 4BG, UK.
Email: nikolay.manchev@kcl.ac.uk

Abstract

Modelling one-to-many type mappings in problems with a temporal component can be challenging. Backpropagation is not applicable to networks that perform discrete sampling and is also susceptible to gradient instabilities, especially when applied to longer sequences. In this paper, we propose two recurrent neural network architectures that leverage stochastic units and mixture models, and are trained with target propagation. We demonstrate that these networks can model complex conditional probability distributions, outperform backpropagation-trained alternatives, and do not rapidly degrade with increased time horizons. Our main contributions consist of the design and evaluation of the architectures that enable the networks to solve multi-model problems with a temporal dimension. This also includes the extension of the target propagation through time algorithm to handle stochastic neurons. The use of target propagation provides an additional computational advantage, which enables the network to handle time horizons that are substantially longer compared to networks fitted using backpropagation.

KEYWORDS

multi-modal learning, recurrent neural networks, stochastic neural networks, target propagation

1 | INTRODUCTION

A recurrent neural network (RNN) is a class of artificial neural network that use internal state based on temporal indexing to process sequential data. It has been show that RNNs are capable of solving a wide range of problems, including, but not limited to, speech recognition,[1] word prediction,[2] text classification,[3,4] image captioning,[5,6] sentiment analysis,[7] language modelling[8] and others.

A diagram of a simple recurrent network (SRN)[9,10] is shown in Figure 1. The network is fully recurrent and is connected to the external environment via its input node, which receives a sequence of data points $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$. The state of the network at time t is then given by:

$$\mathbf{h}_t = \sigma(\mathbf{W}_{\mathrm{xh}} \cdot \mathbf{x}_t + \mathbf{W}_{\mathrm{hh}} \cdot \mathbf{h}_{t-1} + \mathbf{b}_h) \tag{1}$$

where $\mathbf{W}_{\mathrm{xh}}$ is the matrix of synaptic weights between the input and the hidden layer, $\mathbf{W}_{\mathrm{hh}}$ is the matrix of weights between the hidden layer and itself at adjacent time steps, $\mathbf{b}_h$ is a bias term, and $\sigma(\cdot)$ is a non-linear activation function. In this paper, we focus on a variant of the SRN that outputs a single data point at its final time step. For categorical distributions the output is typically given by:

$$\mathbf{y}_t = \mathrm{softmax}(\mathbf{W}_{\mathrm{hy}} \cdot \mathbf{h}_t + \mathbf{b}_y) \tag{2}$$

where $\mathbf{W}_{\mathrm{hy}}$ is a matrix of weights between the hidden layer and the output, and $\mathbf{b}_y$ is the output bias.

Despite their usefulness in modeling temporal dynamic behavior, RNNs have long been criticized for their training difficulties and biological implausibility. The learning instability and failures of RNNs to find optimal solutions are typically associated with the backpropagation through time (BPTT) algorithm, which is a gradient-based technique for training RNNs.[11,12] The key principle BPTT employs is to "unroll" the network for a fixed number of time steps ($t \in [1, t_{\mathrm{max}}]$), using temporal indexing for the inputs and hidden states but sharing the synaptic weights across all t's. The unfolded network is then trained using standard backpropagation. The challenge with this arrangement is that deeper unrolled networks (i.e., high t_{max} values) lead to vanishing and exploding gradients, which prevent the network from learning.[13] It has been shown by Reference 14 that it is necessary for the spectral radius (operator two-norm) of $\mathbf{W}_{\mathrm{hh}}$

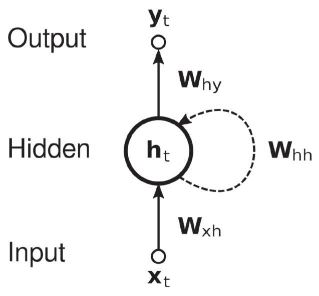


FIGURE 1 A diagram of a simple recurrent network with one input unit, one output unit, and one recurrent hidden unit. The hidden synaptic weights matrix $\mathbf{W}_{\mathrm{hh}}$ allows the network to retain a state that, in theory, can represent information from an arbitrarily long context window.

to be larger than 1 for the gradients to explode, and it is sufficient for it to be smaller than 1 for the gradients to vanish. Various approaches have been suggested to address gradient stability in RNNs, including modifying the network architecture,[15] use of Hessian-free optimization,[16] and a gradient norm clipping strategy.[14] Recently we introduced the target propagation through time algorithm (TPTT), which outperforms BPTT and exhibits stable learning for larger $t_{\max}$.[17]

Backpropagation is also criticized for being biologically implausible, and according to Reference 18 is biologically "unrealistic in almost every respect." This includes mechanisms adopted by BPTT-trained SRNs to solve the credit assignment problem,[19] and the continuous values used for computing the gradients. The latter is in stark contrast with biological neurons that communicate using spikes.[20]

2 | MOTIVATION

A major issue with BPTT-trained RNNs is that because $\sigma(\cdot)$ is deterministic, it models the conditional distribution $p(Y|X)$ using input X and output space Y under a unimodal assumption. There are, however, many cases where the problem that is being modeled mandates a one-to-many mapping (i.e., $p(Y|X)$ is multimodal), and the observed samples come from one of several distinct underlying populations. This is the case with structured prediction problems like modeling facial expressions where the distribution of all possible emotions of a particular individual is multimodal in pixel space.[21] Cross-modal information retrieval, which aims to retrieve interesting content across different modalities, is another task that is still considered an open challenge.[22] Image generation[23] is another real-world task of complex stochastic nature where modeling multiple modalities is necessary. Other examples include image captioning, paper bibliographies, genes and their function, and annotation problems in general.[24] Fuzzy image segmentation, where multiple patterns can have certain ownership over a single pixel,[25] and machine translation[26] fall in the same category.

Solving such problems often involves mixture models in which the model learns multiple components, where each component has a simple parametric form.[27] A drawback of this approach is that the number of network parameters scales at least linearly with the number of mixture components. Another alternative to modeling a multimodal $p(Y|X)$ that doesn't change the architecture of the network is to make the hidden variables stochastic by including stochastic units in its architecture. These stochastic units have noisy binary output with a firing probability given by the sigmoid activation function. Unfortunately, BPTT prevents SRNs with stochastic units from being trained as backpropagation can't compute gradients through discrete units.

The use of stochastic neurons brings additional advantages to generalization performance. The additional noise introduced in the form of stochastic activation acts as a powerful regularization mechanism. It has been shown[28] that adding noise to the network's inputs greatly improves the overall performance of the network and increases the number of hidden units used in the process. Adding multiplicative binary noise to the hidden units leads to even better results as it prevents complex co-adaptations by randomly omitting feature detectors.[29]

The task of modeling biological neurons is another motivator for using non-deterministic activation. The noise associated with neuronal spike trains could be caused by complex electrochemical interactions, which are typically not captured by the idealized model of a neuron. On the other hand, implementing large-scale complex networks using biophysically detailed neuron models is challenging because of the inconvenience around numerical calculations associated with the conductance-based model.[30] Using idealized models with stochastic activation provides an easy way to run simulations and study how such noise may impact the learning in a neural network: an approach that also motivated the Boltzmann machine.[31]

Another use-case is conditional computations, where stochastic units can selectively switch large areas of the network on and off, thus providing the network with a mechanism for executing a hard decision. In this setting, having neurons capable of outputing exactly 0 is essential. This type of hard decisions are also needed in hardware configurations restricted to binary synaptic states like TrueNorth.[32] Moreover, the presence of trainable stochastic neurons in an RNN equips such a network with a capability to take hard stochastic decisions about temporal events at different time scales.[33]

Here, we present two recurrent architectures that leverage stochastic units and demonstrate that the training challenges presented by such networks can be successfully solved by using optimization based on target propagation. Our main contributions consist of the development of two novel architectures that demonstrate that target propagation provides a viable mechanism for addressing multi-model tasks with temporal components. Although the use of mixture models and network architectures with stochastic units are fairly established techniques, they have not been widely applied to recurrent models mostly due to the challenges presented in training networks that handle long temporal dependencies. The current work demonstrates that it is possible to develop and evaluate architecture that are challenging not just from time horizon perspective, but because they also require capturing multi-modal relationships in the modeled tasks.

3 | STOCHASTIC TARGET PROPAGATION THROUGH TIME

In this section, we present stochastic target propagation through time (STPTT): a combined architecture based on target propagation through time[17] and stochastic neurons, which can successfully train RNNs and solve multimodal problems.

The key concept behind target propagation[34,35] is that instead of sending an error signal to the upstream layers, the network establishes individual targets for its hidden layers and uses local optimization to adjust the relevant parameters.

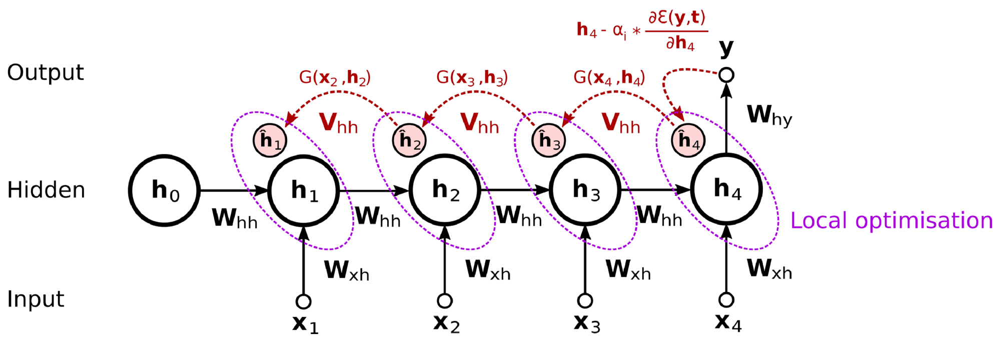
In the case of TPTT the target for $t_{\max}$ is set as:

$$\hat{\mathbf{h}}_{t_{\max}} = \mathbf{h}_{t_{\max}} - \alpha_i * \frac{\partial \mathcal{E}}{\partial \mathbf{h}_{t_{\max}}} \tag{3}$$

Where $\mathcal{E}$ is a global loss for the network, and α_i is a learning rate. The targets for the remaining upstream layers are set by approximating the inverse of (1) using a function $G(\cdot)$, such that:

$$\begin{aligned} \hat{\mathbf{h}}_t &= G(\mathbf{x}_{t+1}, \hat{\mathbf{h}}_{t+1}) \text{ and } \mathbf{h}_{t-1} \approx G(\mathbf{x}_t, \mathbf{h}_t) \\ G(\mathbf{x}_{t+1}, \mathbf{h}_{t+1}) &= \sigma(\mathbf{W}_{\text{xh}} \cdot \mathbf{x}_{t+1} + \mathbf{V}_{\text{hh}} \cdot \mathbf{h}_{t+1} + \mathbf{c}_h) \end{aligned} \tag{4}$$

After all local targets have been established, the network performs a two-stage optimization by first updating the parameters of $G(\cdot)$ based on a local loss (e.g., mean squared error or MSE) between the actual and approximated values of the hidden layers. The second phase updates the feedforward parameters by pushing the hidden layers towards the locally assigned individual targets. The architecture of a shallow, four-step TPTT network is shown in Figure 2 for illustrative purposes. See Reference 17 for further details.



$h_4 - \alpha_i * \frac{\partial\mathcal{E}(\mathbf{y},\mathbf{t})}{\partial \mathbf{h}_4}$
Output
Hidden
Input
G($\mathbf{x}_2$,$\mathbf{h}_2$)
G($\mathbf{x}_3$,$\mathbf{h}_3$)
G($\mathbf{x}_4$,$\mathbf{h}_4$)
$\mathbf{V}_{hh}$
$\hat{\mathbf{h}}_1$ $\hat{\mathbf{h}}_2$ $\hat{\mathbf{h}}_3$ $\hat{\mathbf{h}}_4$
$\mathbf{h}_0$ $\mathbf{h}_1$ $\mathbf{h}_2$ $\mathbf{h}_3$ $\mathbf{h}_4$
$\mathbf{W}_{hh}$
$\mathbf{W}_{xh}$
$\mathbf{W}_{hy}$
$\mathbf{y}$
$\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$ $\mathbf{x}_4$
Local optimisation

FIGURE 2 Target propagation through time: Setting the first and the upstream targets and performing local optimization to bring $\mathbf{h}_t$ closer to $\hat{\mathbf{h}}_t$.

In STPTT stochasticity is introduced in the form of non-deterministic binary hidden units. Here, we look at a single time-step RNN but the extension to multiple time-steps is straightforward. The activation probability of a transition matrix comprised of such neurons at time step t is given by:

$$P(\boldsymbol{h}|\boldsymbol{x}) = P(\boldsymbol{h} = 1|\boldsymbol{x}) = \sigma(\mathbf{W}_{\text{xh}} \cdot \mathbf{x} + \mathbf{b}_h) \tag{5}$$

where $\sigma(\cdot)$ is the sigmoid function. The stochastic state matrix of the network is then defined as a binary random vector drawn from the above distribution. Furthermore, the conditional distribution of the output is then given by:

$$\begin{aligned} P(y|\boldsymbol{x}) = \mathbb{E}_{P(\boldsymbol{h}|\boldsymbol{x})}[P(y|\boldsymbol{h})] &= \mathbb{E}_{P(\boldsymbol{h}|\boldsymbol{x})}[\mathcal{N}(y|\mu_y, \sigma_y^2)] \\ &= \mathbb{E}_{P(\boldsymbol{h}|\boldsymbol{x})}[\mathcal{N}(y|\boldsymbol{W}_{\text{hy}}\boldsymbol{h} + \boldsymbol{b}_y, \sigma_y^2)] \end{aligned} \tag{6}$$

This setting enables the network to represent arbitrary complex conditional probability distributions as a mixture of exponentially many Gaussians.[27] On the contrary, this mandates the summation over an exponential number of configurations of $\boldsymbol{h}$. To address this we compute the expectation with respect to the stochastic neurons by using the following Monte Carlo approximation:

$$\begin{aligned} P(y|\boldsymbol{x}) &\simeq \frac{1}{M}\sum_{m=1}^{M} P(y|\boldsymbol{h}^{(m)}) \\ \boldsymbol{h}^{(m)} &\sim P(\boldsymbol{h}|\boldsymbol{x}) \end{aligned} \tag{7}$$

The sampling steps above introduce discontinuities and prevent the computation of gradients via traditional techniques. This challenge, however, is naturally resolved by the use of target propagation, which employs local optimization and has no dependency on upstream gradient computation.

The selection of M is somewhat problematic. The assumption that $M = 1$ may work well enough doesn't hold. Reference 36 shows that maximizing the expectation of $\log \frac{1}{M}\sum_{m=1}^{M} P(y|\boldsymbol{h}^{(m)})$ with $M = 1$ leads to deterministic behavior (i.e., the network increases the synaptic weights to the sigmoid, turning it into a deterministic step-function). Here we follow Reference 21, who empirically investigate how the chosen number of samples impacts learning and show that $M = 20$ is sufficient for learning good stochastic networks.

Another extension of this model is the inclusion of both stochastic and deterministic units in the weight matrices. Reference 21 show that a hybrid architecture composed of both deterministic and stochastic units achieves superior performance compared to conditional restricted Boltzmann machines and mixture density networks. This architecture can represent an exponential number of mixture components in output space and has the advantage that all units can be jointly trained using TPTT.

4 | TARGET PROPAGATION FOR MIXTURE DENSITY RECURRENT NETWORKS

This section presents a second extension of the standard TPTT-trained RNN by modifying it to parametrize a Gaussian mixture model or GMM.[37] This network tries to learn a conditional distribution:

$$p(y|\boldsymbol{x}_{1:t_{\max}}) = \sum_{i=1}^{k} \pi_i(\boldsymbol{x}_{1:t_{\max}})\, \mathcal{N}\left(y; \mu_i(\boldsymbol{x}_{1:t_{\max}}), \sigma_i^2(\boldsymbol{x}_{1:t_{\max}})\right) \tag{8}$$

where k is the number of mixture components, and $\pi_i(\boldsymbol{x}_{1:t_{\max}})$, $\mu_i(\boldsymbol{x}_{1:t_{\max}})$, and $\sigma_i^2(\boldsymbol{x}_{1:t_{\max}})$ represent the mixing coefficient, mean, and variance of the i-th GMM component. These parameters are obtained as follows:

$$\boldsymbol{\pi} = \mathrm{softmax}(\mathbf{W}_\pi \cdot \mathbf{h}_{t_{\max}} + \mathbf{b}_\pi) \tag{9}$$

$$\boldsymbol{\mu} = \mathbf{W}_\mu \cdot \mathbf{h}_{t_{\max}} + \mathbf{b}_\mu \tag{10}$$

$$\boldsymbol{\sigma} = \exp(\mathbf{W}_\sigma \cdot \mathbf{h}_{t_{\max}} + \mathbf{b}_\sigma) \tag{11}$$

Note, that the normalized exponential is used in (9) to guarantee that the mixing coefficients sum to unity, and the exponential function is used in (11) to ensure that the standard deviation is non-negative.

In general, assuming that the target data is normally distributed means that the maximum likelihood estimate is the same as the least squares estimate. In such a case, the likelihood of the entire dataset is simply the product of the likelihoods of each of the N data points included in the dataset. Using a mixture model allows us to remove the normality assumption for the conditional distribution of the targets, but we can still use Gaussian kernel functions for the individual mixture components. Therefore, we can construct a likelihood function for the entire mixture model by using the summation of all density functions weighted by the mixing coefficients. We can train the model by maximizing the likelihood or by minimizing the negative logarithm of the likelihood for convenience. Here we do the latter, hence, the global loss that the network optimizes is the negative log-likelihood (NLL), which is defined as:

$$\mathcal{E} = \mathrm{NLL}(Y|X) = -\frac{1}{N}\sum_{n=1}^{N} \log\left(\sum_{i=1}^{k} \pi_i(\boldsymbol{x}_{n_{1:t_{\max}}})\, \mathcal{N}\left(y_n; \mu_i(\boldsymbol{x}_{n_{1:t_{\max}}}), \sigma_i^2(\boldsymbol{x}_{n_{1:t_{\max}}})\right)\right) \tag{12}$$

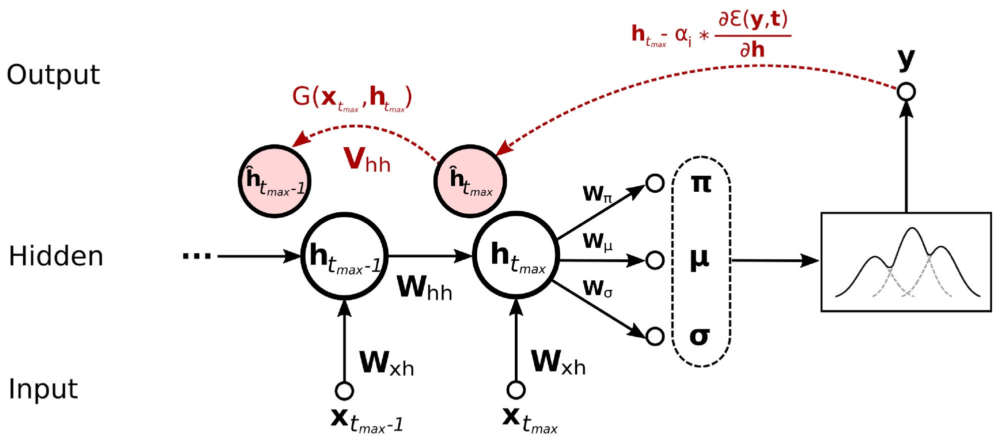


FIGURE 3 A diagram of a TPTT-trained mixture density RNN. Compared to the standard TPTT-trained RNN, the final layer has been modified to output the parameters of a mixture density model (π, μ, and σ).

The modified architecture of the TPTT-trained MDRNN is shown in Figure 3. Setting the targets for the hidden layers is identical to the approach adopted in the standard TPTT-trained RNN: $\hat{\boldsymbol{h}}_{t_{\max}}$ is set using (3) and the targets for the remaining upstream layers are set using the approximate inverse (4). The mixture model parameters are optimized directly using the global loss:

$$\begin{aligned}
\boldsymbol{W}_{\pi} &:= \boldsymbol{W}_{\pi} - \alpha_f \frac{\partial \mathcal{E}}{\partial \boldsymbol{W}_{\pi}} \\
\boldsymbol{W}_{\mu} &:= \boldsymbol{W}_{\mu} - \alpha_f \frac{\partial \mathcal{E}}{\partial \boldsymbol{W}_{\mu}} \\
\boldsymbol{W}_{\sigma} &:= \boldsymbol{W}_{\sigma} - \alpha_f \frac{\partial \mathcal{E}}{\partial \boldsymbol{W}_{\sigma}}
\end{aligned} \tag{13}$$

This extension is useful, because it can model one-to-many mappings for both classification and regression tasks. In Section 5.2, we demonstrate how such a network can successfully solve one-to-many mapping problems featuring a temporal component.

5 | EXPERIMENTS

This section presents a series of experiments conducted with an STPTT and a TPTT-trained MDRNN. The Python code and data used in all experiments is available on GitHub.[1]

5.1 | Synthetic classification task

This task is inspired by Reference 27 and considers a one-to-many mapping between a single input variable x and a single output discrete variable y, which represents 10 separate classes (class labels in $[0, 9]$).

The $x \rightarrow y$ mapping dataset is formed as follows:

$$\begin{aligned} x &= a + 0.3\sin(2\pi a) + \epsilon, \\ y &= [x * 9], \\ &\text{where [] denotes convergent rounding,} \\ &\text{and } a \sim U(0,1),\ \epsilon \sim U(-0.1, 0.1) \end{aligned} \tag{14}$$

The purpose of the rounding function is to discretize the target variable and produce a dataset fit for a classification task. Figure 4 shows a sample of the data and the corresponding mean for each of the individual target classes. The task is further extended and made more challenging by transforming the input variable x into a vector defined as:

$$\boldsymbol{x} = \{x, t_1, t_2, \dots, t_N\} \tag{15}$$

Where the elements $t_1, t_2, \dots, t_N \sim U(0,1)$ act as distractors. This transformation reshapes the inputs x into sequences of length $T = t_N + 1$, which can be fed to the network sequentially, thus introducing a temporal component to the problem. Ultimately, the end goal in this task is for the network to learn to model $p(y|x)$. Therefore, to achieve decent performance, the network also needs to learn to ignore the distractors, making the problem more difficult.

Two hybrid stochastic recurrent neural networks were trained and evaluated on this problem. The first was trained using TPTT and the second using straight-through estimation (STE). The straight-through estimator was proposed by Reference 38, and here it is used to provide baseline results for comparison. This estimator back-propagates through the sampling step by ignoring its derivative, treating the discrete sampling step as it had been the identity function. Reference 36 show that this method achieves the best performance in binary stochastic feedforward networks. The derivatives for a straight-through estimator in the context of binarized RNNs are given in Appendix A.

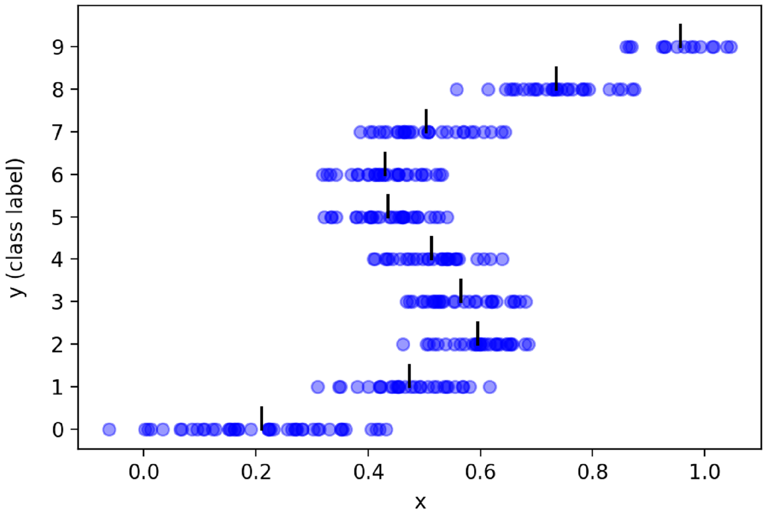


FIGURE 4 A simple one-to-many dataset generated using (14). The black vertical bars denote the mean for each class distribution.

Both networks were trained for 1000 epochs on the same 3000 labelled samples. The networks have an identical number of neurons: 100 in the transition matrix (hidden-to-hidden), 10 neurons in the output layer (corresponding to the number of target classes), and a single neuron for the input (the dimensionality of $\boldsymbol{W}_{\mathrm{xh}}$ is 1×100). The hidden-to-hidden matrix is used in a hybrid configuration with an equal split between stochastic and deterministic units (i.e., 50 binary stochastic neurons and 50 deterministic neurons).

The length of the unrolled network is $T = t_N + 1$, where N represents the number of distractor symbols after x. The networks also use the *sigmoid* as their activation function, **softmax** for the output layer, and cross-entropy as global loss. The networks were optimized using AdaGrad,[39] and the hyperparameter selection was based on a grid search with varying learning rates from $[1.0 \times 10^{-5}, 1.0 \times 10^{-1}]$. Each network was trained in two different settings: a shallow setting ($T =$ 2 or a single distractor symbol) and a deep setting ($T = 20$, i.e., 19 distractors). The performance in terms of cross-entropy loss as measured on the training data is presented in Table 1. The STE network uses $\alpha = 0.1$. The learning rates for the STPTT network are set to $\alpha_i = \alpha_f = 1.0 \times 10^{-1}$ and $\alpha_g = 1.0 \times 10^{-5}$. The number of samples drawn for calculating the loss after the forward phase is set to $M = 20$ for both networks.

After training is completed, each network is sampled across the 10 possible classes ($10 \times 100 =$ 1000 observations) and the samples are plotted against the training data (Figure 5). The mean of the sampled data from each distribution is also calculated and reflected on the plot. It is evident that when the task is shallow (two time steps, single distractor) the STE and the STPTT networks perform equally well. The increase of the sequence length, though, substantially impacts the performance of the STE network. Not only does it fail to learn the correct means, but it completely misses an entire distribution (class 9). The STPTT network on the other hand is less impacted, and although its output gets noisier with the increase of depth, it still approximates the ten distributions reasonably well. This finding is in agreement with the results presented in Reference 17 as straight-through estimation ultimately relies on backpropagation, which is susceptible to gradient instabilities driven by the increase of the sequence length.

5.2 | Synthetic regression task using a mixture density recurrent neural network

The problem in this section is directly taken from Reference 27 and represents a one-to-many mapping defined by $y = x + 0.3\sin(2\pi x) + \epsilon$, where $\epsilon \sim U(-0.1, 0.1)$ and $x \in [0, 1]$. A 3000 samples dataset depicted in Figure 6 is used for training, and a further 1000 samples are used as a hold-out set. This task is further transformed into a temporal problem by adding distractor symbols in a fashion identical to the modification introduced in Section 5.1 (see Equation 15).

Two MDRNNs were trained and evaluated on the synthetic regression problem. One MDRNN was trained with standard BPTT and the other trained using the TPTT scheme outlined above.

TABLE 1 Cross-entropy loss for the straight-trough estimator-trained RNN (STE) and a stochastic target propagation through time-trained RNN (STPTT).

Sequence length	STE	STPTT
$T = 2$	1.44	1.44
$T = 20$	2.27	1.54

Note: Each network is tested in two regimes: shallow (sequence length of 2) and deep (sequence length of 20).

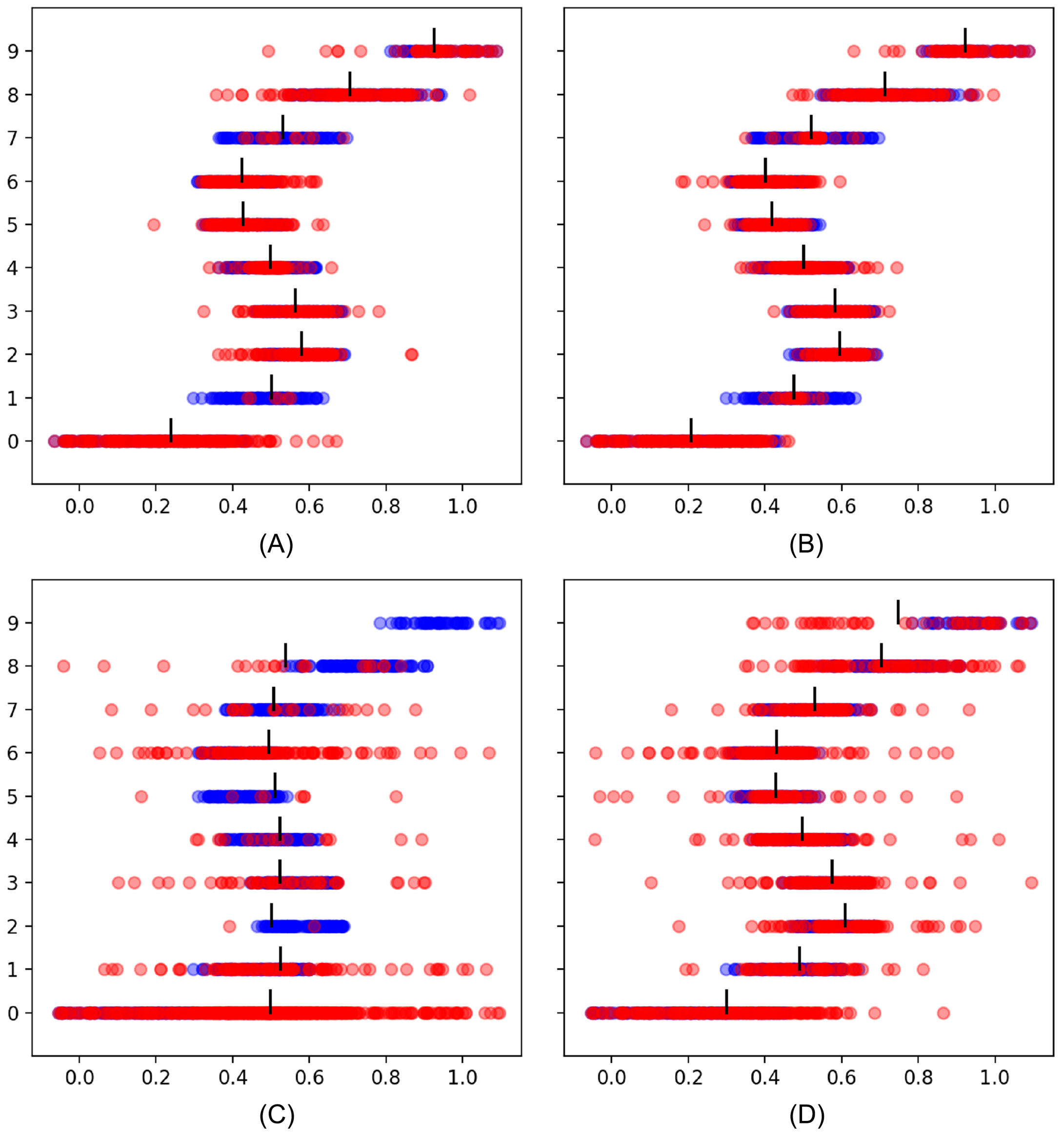


FIGURE 5 Comparison of the estimation of multiple overlapping distributions as defined by the synthetic classification task (Section 5.1). The training data is given in blue, and the samples from the trained network are plotted in red. The vertical bars denote the mean of each distribution, calculated using samples from each class obtained from the trained networks. For relatively shallow networks ($T = 2$) STE (A) and TPTT (B) optimization approximate the mean of the distributions equally well. When the network depth is increased ($T = 20$), the samples from the TPTT-trained network (D) get noisier, but it still approximates the means fairly well and retains the characteristic S shape of the mean values. The performance of the STE-trained network (C), however, rapidly degrades—the means are way off, and it completely misses an entire distribution.

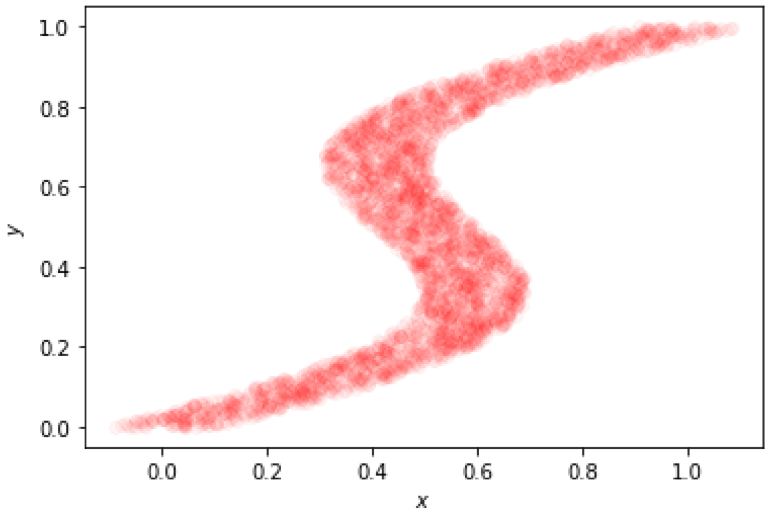


FIGURE 6 A 3000 observations sample for modeling a continuous one-to-many relationship.

TABLE 2 Lowest NLL achieved for a BPTT and TPTT-trained MDRNN.

Sequence length	BPTT	TPTT
$T = 5$	−0.98	−0.98
$T = 30$	0.54	−0.94

Note: Each network is tested using a shallow (sequence length set to 5) and a deep (sequence length set to 30) variant of the synthetic regression task. For short sequences the two networks perform equally well, but increasing the depth of the network substantially degrades the BPTT performance, while the impact on the TPTT-trained MDRNN is negligible.

Both networks had the number of mixture components set to $k = 3$, used the AdaGrad optimizer for tuning their parameters, and were trained for 4000 iterations. A cartesian grid search was performed for finding optimal learning rates for the two networks. As a result the BPTT network uses $\alpha = 0.01$, and the learning rates for the TPTT network are $\alpha_i = 1.0$, $\alpha_f = 0.01$, and $\alpha_g = 0.001$. The lowest NLL achieved using these optimized training hyper-parameters is reported in Table 2. The results demonstrate that BPTT optimization rapidly degrades with increased sequence lengths, while the performance of the TPTT-trained MDRNN is minimally impacted. This is also evident when the trained networks are randomly sampled to inspect the learned distributions. Figure 7 shows the plots of 1000 points sampled from each of the networks under consideration. We see that both the BPTT and TPTT-trained MDRNNs capture the characteristic shape of the one-to-many relationship when the temporal dependency is relatively short (Figure 7A,B). The performance of the TPTT network remains unaffected with the increase of the sequence length, while the BPTT-trained network completely fails at capturing the correct relationship.

5.3 | Polyphonic music task

These experiments directly leverage the recurrent stochastic extension of a TPTT-trained network as outlined in Section 3. The experiments were conducted using a dataset of classical piano midi

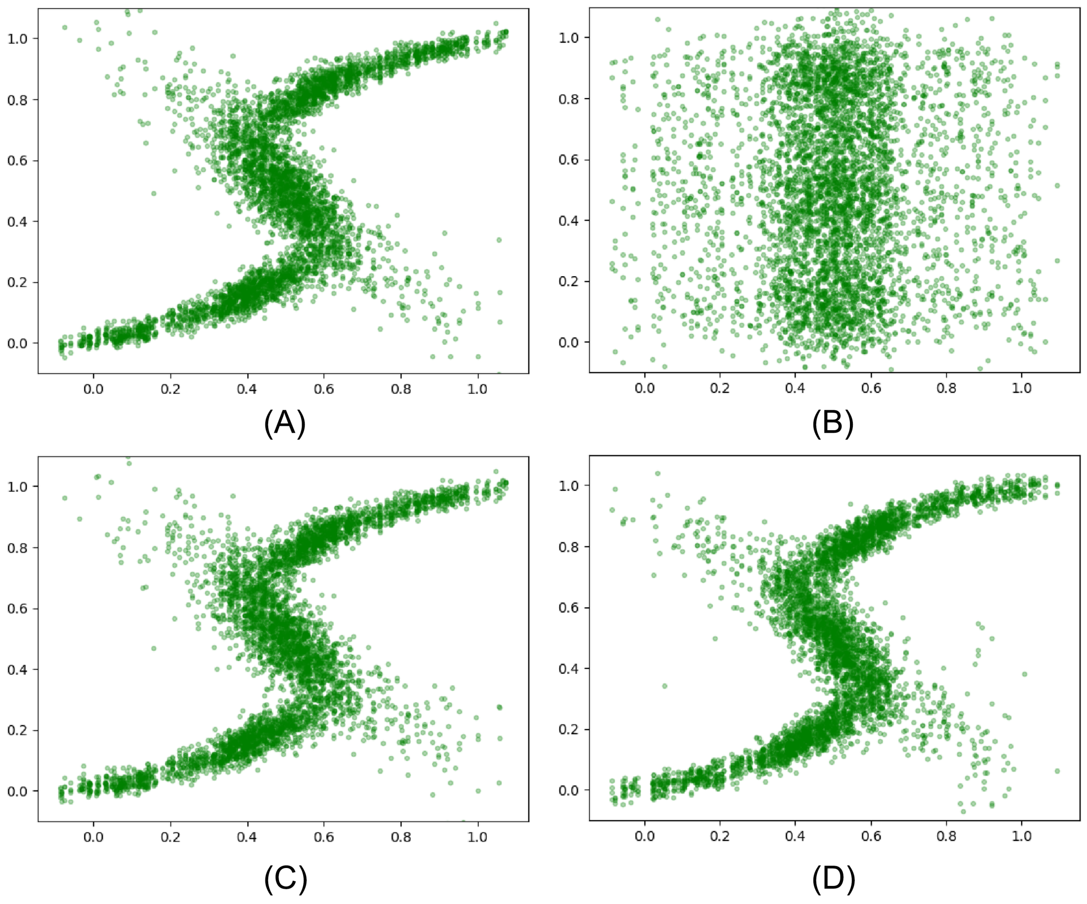


FIGURE 7 Comparison of the performance of BPTT versus TPTT-trained MDN on the synthetic regression task. (A) A shallow BPTT-trained network, $T = 5$; (B) A deep BPTT-trained network, $T = 30$; (C) A shallow TPTT-trained network, $T = 5$; (D) A deep TPTT-trained network, $T = 30$.

music (Piano-midi.de), which was also used in References 40,41 and 42. The data was downloaded from http://www.piano-midi.de and split into training, validation, and test sets following.[43] The training set comprises over 6 h of music and the test set, which is used for measuring the performance of the network, spans slightly over 1 h and 27 min. We used pre-processed training, validation, and tests sets as provided by Reference 44. All pieces were transposed to C4 and sampled every eighth note (quarter note for Johann Sebastian Bach's chorales) following the beat information given in the MIDI files. Each sequence in the data set was loaded as a list of time steps, where a time step contains a list of the non-zero elements in the piano-roll at this instant.

The goal in this task is to predict the next note or combination of simultaneously played notes. This is a challenging problem because of the high dimensionality of the data and the complex temporal dependencies involved. For example, a network could easily learn a number of scales (patterns of notes ordered by pitch) but piano pieces often feature multiple voices in the same hand using overlapping piano scales. In this case, we model the output as a zero-initialized 88-dimensional binary vector representing the A0–C7 range. A one indicates that the note in the specified location is being played at the current time-step. This set up enables us to model jointly played notes and chords by having multiple ones in the label vector.

TABLE 3 NLL for a BPTT and TPTT-trained RNN with 50 stochastic and 50 deterministic neurons.

Optimizer	STE-BPTT	STPTT
AdaGrad	13.34	**9.69**
Nesterov	13.39	12.44
SGD	13.39	12.44
RMS	13.17	9.86

Note: Sequence length was set to $T = 50$.

TABLE 4 NLL for a fully deterministic, STE-optimized RNN (STE(D)-BPTT), an stochastic baseline RNN optimized with (STE-BPTT) and TPTT-trained stochastic RNN (STPTT) with 300 neurons in a hybrid 50:50 configuration.

STE(D)-BPTT	STE-BPTT	STPTT
11.22	13.25	**9.45**

Note: Sequence length was set to $T = 100$.

Two hybrid recurrent networks were fitted using this dataset: a baseline straight-through estimator trained via BPTT (STE-BPTT) and a stochastic TPTT-trained network (STPTT). Both networks were configured with 100 neurons in their hidden layers. The split between stochastic and deterministic neurons is evenly balanced (a 50:50 ratio), as suggested by Reference 21.

A grid search for determining the optimal learning rates for both networks was performed over $[1, 1 \times 10^{-4}]$. In addition, to rule out any optimizer induced impact on the final results, four different optimizers were treated as a parameter of the grid search (AdaGrad, Nesterov, SGD, and RMS). The networks were trained over sequences of length $T = 50$, and the lowest NLL values achieved for each optimizer are reported in Table 3. The results demonstrate that the STPTT network outperforms the baseline regardless of which optimizer was used for adjusting the synaptic weights.

To further evaluate the impact of network depth on the behavior of the stochastic network, we ran three experiments using the optimal parameters selected by the grid search but with a time horizon of 100 time steps. We also increased the networks complexity by using 300 neurons in a hybrid configuration, with an identical 50:50 split between deterministic and stochastic. The networks were trained for 300 epochs, which equates to 20,132,880 training sequences. Once again the STPTT network significantly outperformed the STE baseline (see Table 4). We also conducted a separate experiment with STE using fully deterministic neurons: STE(D)-BPTT. Interestingly, this network performed better than its hybrid STE counterpart. Nevertheless, it was also significantly outperformed by its stochastic TPTT-trained competitor.

6 | DISCUSSION AND FUTURE WORK

In this paper, we introduced two extensions to the standard target propagation trained recurrent neural network as outlined in Reference 17. We proposed a mechanism for training recurrent neural networks with stochastic units (both fully stochastic or in a hybrid configuration, where stochastic units are used in conjunction with traditional fully deterministic neurons). We also showed an extension of the standard TPTT-trained recurrent neural network to a mixture density RNN, which can successfully solve one-to-many mappings for regression problems in a temporal setting. We believe that this work provides a valuable perspective to learning multi-modal RNNs and could be of interest in a wide range of related applications.

We empirically demonstrated that the proposed variants outperform our baseline comparisons (STE in the case of stochastic units and backpropagation for the MDRNN) in three distinctive problems, which require many-to-one mappings—a synthetic classification task, a synthetic regression task, and a polyphonic music classification task. We speculate that the better performance of these two architectures can be attributed to the overall better stability that target propagation exhibits over extended time horizons.[17] On the other hand, there are certain computational disadvantages stemming from the use of stochastic units and the use of target propagation in general. The pairing of stochastic units with a Markov chain Monte Carlo sampling is computationally demanding. For the STPTT network that we use in our experiments the value of $M = 20$ leads to a 20-fold increase of the forward passes needed to train the network. Target propagation also brings an increased difficulty in hyperparameter tuning as it employs three different learning rates, thus substantially increasing the search space. This can, of course, be mitigated by clamping the α_i, α_g, and α_f to the same value if it is deemed that model performance can be sacrificed to reduce the time needed for hyperparameter optimization. It is important to recognize that although we performed the evaluation on a selected class of problems, these network architectures are fairly general and can be applied to a wide range of problems like word prediction, text classification, image captioning, sentiment analysis, language modeling, and many others.[2] All of these problems can occur in a setting where one-to-many mapping is integral to finding good solutions. We, however, defer the aspect of evaluation on other, more practical tasks to future investigations.

In Reference 17, we also demonstrated that injection of random Gaussian noise in the process of learning $G(\cdot)$ can be beneficial as a form of regularization. This ensures that the inverse function is not over-fitted to the training data, and in certain cases can lead to an increase of accuracy of over 10%. We did not investigate the impact of noise injection in any of the proposed architectures and leave this to future work. Similarly, the choice of M is based on empirical results[21] obtained for different architectures and there is a possibility that better M values exists for the experiments and network variants presented in this paper. It is also worth noting that there is no inherent need to clamp the value of M during the learning and inference phases of the network. For example, Reference 45 use $M = 1$ when training a feedforward stochastic neural network, but switch to $M = 100$ when performing inference. Further investigation on the selection of optimal M is also left to future work.

For all experiments in this paper we used transition matrices comprising 100 neurons. This is identical to the setup used in Reference 17 and matches what[41] used for their synthetic evaluations. It must be noted that we did not experiment with other transition matrix sizes but, as we showed in Reference 17, varying the size of $\boldsymbol{W}_{\text{hh}}$ in TPTT-trained networks can have a substantial impact on the model performance. Therefore, this is a potential area for future research. In terms of the split between stochastic and deterministic units, we decided to maintain an equal ratio of 50:50 as suggested by Reference 21. This is not a hard requirement and a future analysis of how varying the stochastic to deterministic split impacts the network performance could be potentially beneficial.

While not explicitly investigated here, the ideas outlined in this paper can easily be extended to computationally more powerful architectures like LSTM.[15] This, alongside the architecture optimization of the presented networks (e.g., number of elements in the transition matrix, number of mixture components etc.) is left to future work.

CONFLICT OF INTEREST STATEMENT

The authors declare no potential conflict of interests.

DATA AVAILABILITY STATEMENT

The data that supports the findings of this study are available in the supplementary material of this article.

ENDNOTES

[1]See https://github.com/nmanchev/MM-TPTT-RNN.
[2]Relevant references are provided in Section 1.

ORCID

Nikolay Manchev https://orcid.org/0000-0003-2987-9234

REFERENCES

1. Graves A. Sequence transduction with recurrent neural networks. CoRR; abs/1211.3711. 2012.
2. Mikolov T. Efficient estimation of word representations in vector space. CoRR; abs/1301.3781. 2013.
3. Liu P. Recurrent neural network for text classification with multi-task learning. Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016. 2016;2873-2879.
4. Yogatama D. Generative and discriminative text classification with recurrent neural networks. ArXiv e-Prints. 2017.
5. Karpathy A, Fei-Fei L. Deep visual-semantic alignments for generating image descriptions. *IEEE Trans Pattern Anal Mach Intell*. 2017;39(4):664-676.
6. Xu K. Show, attend and tell: neural image caption generation with visual attention. CoRR; abs/1502.03044. 2015.
7. Timmaraju A, Khanna V. Sentiment Analysis on Movie Reviews using Recursive and Recurrent Neural Network Architectures. 2015.
8. De Mulder W. A survey on the application of recurrent neural networks to statistical language Modeling. *Comput Speech Lang*. 2015;30(1):61-98.
9. Elman JL. Finding structure in time. *Cognit Sci*. 1990;14(2):179-211.
10. Jordan MI. Serial order: a parallel, distributed processing approach. In: Elman JL, Rumelhart DE, eds. *Advances in Connectionist Theory: Speech*. Erlbaum; 1989.
11. Mozer M. A Focused Backpropagation Algorithm for Temporal Pattern Recognition. 1995.
12. Werbos PJ. Generalization of backpropagation with application to a recurrent gas market model. *Neural Netw*. 1988;1(4):339-356.
13. Bengio Y. Learning long-term dependencies with gradient descent is difficult. *Trans Neur Netw*. 1994;5(2):157-166.
14. Pascanu R. Understanding the exploding gradient problem. CoRR; abs/1211.5063. 2012.
15. Hochreiter S, Schmidhuber J. Long short-term memory. *Neural Comput*. 1997;9(8):1735-1780.
16. Sutskever I. Generating text with recurrent neural networks. Icml. 2011;1017-1024.
17. Manchev N, Spratling M. Target propagation in recurrent neural networks. *J Mach Learn Res*. 2020;21(7):1-33.
18. Crick F. The recent excitement about neural networks. *Nature*. 1989;337(1):129-132.
19. Lillicrap TP. Random synaptic feedback weights support error backpropagation for deep learning. *Nat Commun*. 2016;7:1-10.
20. Bengio Y. Towards biologically plausible deep learning. CoRR; abs/1502.04156. 2015.
21. Tang Y, Salakhutdinov R. Learning stochastic feedforward neural networks. Proceedings of the 26th International Conference on Neural Information Processing Systems-Volume 1. 2013;530-538.

22. Kaur P. Comparative analysis on cross-modal information retrieval: a review. *Comput Sci Rev*. 2021;39:100336.
23. Goodfellow IJ. Generative Adversarial Networks. 2014.
24. Blei DM, Jordan MI. Modeling Annotated Data. 2003;127-134.
25. Shen JJ. A stochastic-variational model for soft mumford-shah segmentation. *Int J Biomed Imag*. 2016;2006(1):092329.
26. Shen T. Mixture models for diverse machine translation: tricks of the trade. In: Chaudhuri K, Salakhutdinov R, eds. Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA. 2019;5719-5728.
27. Bishop CM. *Mixture Density Networks*. Working paper. Aston University; 1994.
28. Sietsma J, Dow RJF. Creating Artificial Neural Networks That Generalize. Volume 4 1991.
29. Hinton GE. Improving neural networks by preventing co-adaptation of feature detectors. CoRR; abs/1207.0580. 2012.
30. Wang T. Predicting spike features of Hodgkin-Huxley-type neurons with simple artificial neural network. *Front Comput Neurosci*. 2022;15:800875.
31. Hinton GE. *Boltzmann Machines: Constraint Satisfaction Networks that Learn*. Carnegie-Mellon University, Department of Computer Science; 1984.
32. Esser SK. Cognitive computing systems: algorithms and applications for networks of neurosynaptic cores. The 2013 International Joint Conference on Neural Networks (IJCNN). 2013;1-10.
33. Bengio Y. Estimating or propagating gradients through stochastic neurons for conditional computation. CoRR; abs/1308.3432. 2013.
34. Bengio Y. How auto-encoders could provide credit assignment in deep networks via target propagation. CoRR; abs/1407.7906. 2014.
35. LeCun Y. Learning process in an asymmetric threshold network. In: Bienenstock E, ed. *Disordered Systems and Biological Organization*. Springer Berlin Heidelberg; 1986:233-240.
36. Raiko T. Techniques for learning binary stochastic feedforward neural networks. In: Bengio Y, LeCun Y, eds. *ICLR (Poster)*. OpenReview.net; 2015.
37. McLachlan GJ, Basford KE. *Mixture Models: Inference and Applications to Clustering*. Marcel Dekker; 1988.
38. Hinton G. Neural networks for machine learning. 2012.
39. Duchi JC. Adaptive subgradient methods for online learning and stochastic optimization. *J Mach Learn Res*. 2011;12:2121-2159.
40. Bengio Y. Advances in optimizing recurrent networks. ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing-Proceedings. 2013;8624-8628.
41. Pascanu R. On the difficulty of training recurrent neural networks. Proceedings of the 30th International Conference on Machine Learning, edited by Sanjoy Dasgupta and David McAllester. 2013;1310-1318.
42. Bayer J, Osendorfer C. Learning Stochastic Recurrent Networks. 2015.
43. Poliner GE, Ellis DPW. A discriminative model for polyphonic piano transcription. *EURASIP J Adv Signal Process*. 2007;2007(1):154.
44. Boulanger-Lewandowski N. Modeling temporal dependencies in high-dimensional sequences: application to polyphonic music generation and transcription. Proceedings of the 29th International Coference on International Conference on Machine Learning. 2012;1881-1888.
45. Lee DH. Difference target propagation. *Lecture Notes Comput Sci (Including Subseries Lecture Notes Artif Intell Lecture Notes Bioinform)*. 2015;9284:498-515.

How to cite this article: Manchev N, Spratling M. Learning multi-modal recurrent neural networks with target propagation. *Computational Intelligence*. 2024;40(4):e12691. doi: 10.1111/coin.12691

APPENDIX A. DERIVATION OF STRAIGHT-THROUGH GRADIENT OPTIMIZATION FOR A SIMPLE RECURRENT NEURAL NETWORK

Let's define a binarized RNN as follows:

$$\begin{aligned} ha_t &= W_{\text{hh}} \cdot hs_{t-1} + W_{\text{xh}} \cdot x_t + b_h \\ h_t &= \sigma(ha_t) \\ hs_t &= \text{samp}(h_t) \end{aligned}$$

where samp$(\cdot)$ is our sampling function defined as $\text{samp}(h_i, z_i) = 1_{z_i > h_i}$ where $z_i \sim U[0,1]$. For the output at the final time step ($t_{\max}$) we have:

$$\begin{aligned} y_a &= W_{\text{hy}} \cdot h_{t_{\max}} + b_y \\ y &= \text{softmax}(y_a) \\ \mathcal{E} &= \text{NLL}(y, t) = -\sum_{i=1}^{M} t_i \times \log(y_i), \text{where } M \text{ is the length of } y \text{ and } t \end{aligned}$$

Next, we derive the gradients for the synaptic weights. Note, that hs_t is completely ignored, as in the context of straight-through optimization samp$(\cdot)$ is considered to be the identity function, and the derivative of the identity function is 1.

A.1 W_{hy}, b_y

$$\frac{\partial \mathcal{E}}{\partial W_{\text{hy}}} = \frac{\partial \mathcal{E}}{\partial y} \frac{\partial y}{\partial y_a} \frac{\partial y_a}{\partial W_{\text{hh}}}$$

The first two terms are straightforward:

$$\frac{\partial \mathcal{E}}{\partial y} \frac{\partial y}{\partial y_a} = -(t - y) = y - t \tag{A1}$$

We then have:

$$\frac{\partial \mathcal{E}}{\partial W_{\text{hy}}} = (y - t) \frac{\partial y_a}{\partial W_{\text{hh}}} = (y - t) h_{t_{\max}}$$

and for the bias we have:

$$\frac{\partial \mathcal{E}}{\partial b_y} = (y - t) \frac{\partial y_a}{\partial b_y} = (y - t) * 1 = y - t$$

A.2 W_{hh}, b_h

For the deepest layer of the network (at $h_{t_{\max}}$) we have:

$$\frac{\partial \mathcal{E}}{\partial W_{\text{hh}}} = \frac{\partial \mathcal{E}}{\partial y} \frac{\partial y}{\partial y_a} \frac{\partial y_a}{\partial h_{t_{\max}}} \frac{\partial h_{t_{\max}}}{\partial ha_{t_{\max}}} \frac{\partial ha_{t_{\max}}}{\partial W_{\text{hh}}}$$

Because the hidden state h_t partially influences h_{t+1}, we need to backprop through the upstream layers, so $\frac{\partial \mathcal{E}}{\partial W_{\text{hh}}}$ becomes:

$$\frac{\partial \mathcal{E}}{\partial W_{\text{hh}}} = \sum_{t=1}^{t_{\max}-1} \frac{\partial \mathcal{E}}{\partial y} \frac{\partial y}{\partial y_a} \frac{\partial y_a}{\partial h_{t_{\max}}} \frac{\partial h_{t_{\max}}}{\partial ha_{t_{\max}}} \frac{\partial ha_{t_{\max}}}{h_t} \frac{\partial h_t}{\partial ha_t} \frac{\partial ha_t}{\partial W_{\text{hh}}}$$

$$\frac{\partial \mathcal{E}}{\partial W_{\text{hh}}} = \frac{\partial \mathcal{E}}{\partial y} \frac{\partial y}{\partial y_a} \frac{\partial y_a}{\partial h_{t_{\max}}} \frac{\partial h_{t_{\max}}}{\partial ha_{t_{\max}}} \sum_{t=1}^{t_{\max}-1} \frac{\partial ha_{t_{\max}}}{h_t} \frac{\partial h_t}{\partial ha_t} \frac{\partial ha_t}{\partial W_{\text{hh}}}$$

The first two terms in the summation are:

$$\frac{\partial ha_{t_{\max}}}{h_t} \frac{\partial h_t}{\partial ha_t} = \prod_{i=t+1}^{t_{\max}} \frac{\partial ha_i}{\partial h_{i-1}} \frac{\partial h_{i-1}}{\partial ha_{i-1}} \tag{A2}$$

So we finally get:

$$\frac{\partial \mathcal{E}}{\partial W_{\text{hh}}} = \frac{\partial \mathcal{E}}{\partial y} \frac{\partial y}{\partial y_a} \frac{\partial y_a}{\partial h_{t_{\max}}} \frac{\partial h_{t_{\max}}}{\partial ha_{t_{\max}}} \sum_{t=1}^{t_{\max}} \left[\prod_{i=t+1}^{t_{\max}} \frac{\partial ha_i}{\partial h_{i-1}} \frac{\partial h_{i-1}}{\partial ha_{i-1}} \right] \frac{\partial ha_t}{\partial W_{\text{hh}}} \tag{A3}$$

Looking at the first four terms above we can use (A1) to get:

$$\frac{\partial \mathcal{E}}{\partial y} \frac{\partial y}{\partial y_a} \frac{\partial y_a}{\partial h_{t_{\max}}} \frac{\partial h_{t_{\max}}}{\partial ha_{t_{\max}}} = (y-t) \frac{\partial y_a}{\partial h_{t_{\max}}} \frac{\partial h_{t_{\max}}}{\partial ha_{t_{\max}}} \frac{\partial ha_{t_{\max}}}{\partial W_{\text{hh}}}$$

$\frac{\partial y_a}{\partial h_{t_{\max}}}$ is simply W_{hy}. We also use $\sigma'(x) = \sigma(x)(1-\sigma(x))$ to get:

$$\frac{\partial \mathcal{E}}{\partial y} \frac{\partial y}{\partial y_a} \frac{\partial y_a}{\partial h_{t_{\max}}} \frac{\partial h_{t_{\max}}}{\partial ha_{t_{\max}}} = (y-t) \cdot W_{\text{hy}} \cdot \sigma(ha_{t_{\max}}) \cdot (1-\sigma(ha_{t_{\max}})) \tag{A4}$$

For (A2) we use the derivative of the sigmoid to get:

$$\prod_{i=t+1}^{t_{\max}} \frac{\partial ha_i}{\partial h_{i-1}} \frac{\partial h_{i-1}}{\partial ha_{i-1}} = \prod_{i=t+1}^{t_{\max}} W_{\text{hh}} \cdot \sigma(ha_{i-1}) \cdot (1-\sigma(ha_{i-1})) \tag{A5}$$

Substituting (A4) and (A5) in (A3) and using $\frac{\partial ha_t}{\partial W_{\text{hh}}} = h_{t-1}$

$$\frac{\partial \mathcal{E}}{\partial W_{\text{hh}}} = (y-t) \cdot W_{\text{hy}} \cdot \sigma(ha_{t_{\max}}) \cdot (1-\sigma(ha_{t_{\max}}))$$
$$\sum_{t=1}^{t_{\max}} \left[\prod_{i=t+1}^{t_{\max}} W_{\text{hh}} \cdot \sigma(ha_{i-1}) \cdot (1-\sigma(ha_{i-1})) \right] \cdot h_{t-1}$$

The derivation of $\frac{\partial \mathcal{E}}{\partial b_h}$ is identical to (A3). The only difference is the final term, which is $\frac{\partial ha_t}{\partial b_h}$ (equal to 1). Therefore for b_h we have:

$$\frac{\partial \mathcal{E}}{\partial b_h} = \frac{\partial \mathcal{E}}{\partial y}\frac{\partial y}{\partial y_a}\frac{\partial y_a}{\partial h_{t_{\max}}}\frac{\partial h_{t_{\max}}}{\partial ha_{t_{\max}}}\sum_{t=1}^{t_{\max}}\left[\prod_{i=t+1}^{t_{\max}}\frac{\partial ha_i}{\partial h_{i-1}}\frac{\partial h_{i-1}}{\partial ha_{i-1}}\right]\frac{\partial ha_t}{\partial b_h}$$

$$\frac{\partial \mathcal{E}}{\partial b_h} = (y-t)\cdot W_{\mathrm{hy}}\cdot\sigma(ha_{t_{\max}})\cdot(1-\sigma(ha_{t_{\max}}))\sum_{t=1}^{t_{\max}}\left[\prod_{i=t+1}^{t_{\max}}W_{\mathrm{hh}}\cdot\sigma(ha_{i-1})\cdot(1-\sigma(ha_{i-1}))\right]$$

A.3 $\boldsymbol{W}_{\mathbf{xh}}$

This is again identical to (A3) except for the last term:

$$\frac{\partial \mathcal{E}}{\partial W_{\mathrm{xh}}} = \frac{\partial \mathcal{E}}{\partial y}\frac{\partial y}{\partial y_a}\frac{\partial y_a}{\partial h_{t_{\max}}}\frac{\partial h_{t_{\max}}}{\partial ha_{t_{\max}}}\sum_{t=1}^{t_{\max}}\left[\prod_{i=t+1}^{t_{\max}}\frac{\partial ha_i}{\partial h_{i-1}}\frac{\partial h_{i-1}}{\partial ha_{i-1}}\right]\frac{\partial ha_t}{\partial W_{\mathrm{xh}}}$$

$$\frac{\partial \mathcal{E}}{\partial W_{\mathrm{xh}}} = (y-t)\cdot W_{\mathrm{hy}}\cdot\sigma(ha_{t_{\max}})\cdot(1-\sigma(ha_{t_{\max}}))\sum_{t=1}^{t_{\max}}\left[\prod_{i=t+1}^{t_{\max}}W_{\mathrm{hh}}\cdot\sigma(ha_{i-1})\cdot(1-\sigma(ha_{i-1}))\right]x_t$$