Training Many-to-Many Recurrent Neural Networks with Target Propagation

Peilun Dai(✉) and Sang Chin

Department of Computer Science, Boston University, Boston, MA 02215, USA
{peilun,spchin}@bu.edu

Abstract. Deep neural networks trained with back-propagation have been the driving force for the progress in fields such as computer vision, natural language processing. However, back-propagation has often been criticized for its biological implausibility. More biologically plausible alternatives to backpropagation such as target propagation and feedback alignment have been proposed. But most of these learning algorithms are originally designed and tested for feedforward networks, and their ability for training recurrent networks and arbitrary computation graphs is not fully studied nor understood. In this paper, we propose a learning procedure based on target propagation for training multi-output recurrent networks. It opens doors to extending such biologically plausible models as general learning algorithms for arbitrary graphs.

Keywords: Artificial neural networks · Recurrent neural networks · Biologically plausible learning · Target propagation · Backpropagation

1 Introduction

Our brain has the amazing ability to use past information to set up our expectations for the future and use the actual perceived information to update synaptic weights to build a better model of the world around us. This type of sequential modelling also has been an important tasks for artificial neural networks. An important sequence model is the *Simple Recurrent Model* (SRN) proposed by [6,9] which has been the basis for many of today's successful sequence models. Usually recurrent neural network (RNN) models are trained by *backpropagation through time* (BPTT) [15,17]. However, there are two major challenges for training recurrent networks with BPTT: First, there is the well known vanishing/exploding gradient problem that the error signal received by earlier steps are either too small or to large due to the long paths of applying chain rules under certain conditions [14]. Second, it is usually considered biologically implausible [5] because it requires symmetric weights for the forward and backward passes, which has not been observed biologically. Other more biologically learning algorithms have been proposed, such as target propagation that utilizes auto-encoders for credit assignment [2,10].

I. Farkaš et al. (Eds.): ICANN 2021, LNCS 12894, pp. 433–443, 2021.
https://doi.org/10.1007/978-3-030-86380-7_35

Target propagation was originally proposed for feedforward neural networks, and there has been work to extend it to training RNNs. In [12], a step-wise inverse function is used to propagate the target activations backward in time, which is termed *Target Propagation Through Time* (TPTT). The authors have shown that target propagation is able to back propagate targets instead of error derivatives over longer ranges than backpropagation can, partially addressing the exploding/vanishing gradient problem. However, It is not straightforward to extend TPTT to RNNs with multiple outputs. For models with multiple loss terms, each loss term would have its own credit assignment path. The error back propagation in backpropagation algorithm is a linear operation, thus, we can add the error derivatives from multiple credit assignment paths and propagate the resulting accumulated derivatives only. However, due to the non-linearity in target propagation's backward pass, we cannot add up the targets from different paths directly. In this paper, we propose a method that could merge the targets from multiple loss terms, and as a result, only the merged targets need to be further propagated. This method is able to generalize target propagation to training RNNs with multiple outputs and potentially to arbitrary computation graphs with multiple credit assignment paths.

In the following section, we will give a brief introduction to backpropagation and target propagation.

2 Background

2.1 Backpropagation Through Time (BPTT)

With the help of backpropagation, recurrent neural network (RNN) models have been widely applied to solve many sequence modeling tasks in domains such as audio signal processing, natural language processing and more. The simplest recurrent network model is the Simple Recurrent Network (SRN) [6], and it is considered a precursor to many of today's state-of-the-art RNN models such as Long Short-Term Memory (LSTM) [8] and Gated Recurrent Unit (GRU) [4].

A single layer of the simple recurrent network is defined by three sets of weights $\{\mathbf{W}_{xh}\}$, $\{\mathbf{W}_h, \mathbf{b}_h\}$ and $\{\mathbf{W}_{hy}, \mathbf{b}_{hy}\}$ and an activation function for the hidden layer, such as $ReLU(\cdot)$ and $tanh(\cdot)$. Depending on the tasks, an RNN may need to produce an output at each step (many-to-many) for tasks such as language modeling and part-of-speech (POS) tagging, or may only produce one output at the last step (many-to-one) for tasks such as customer review sentiment classification. For a typical many-to-many task, the inference step is defined by a forward pass in time through the following equations for $t = 1, 2, \ldots, T$,

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

$$\mathbf{y}_t = \sigma_y(\mathbf{W}_{hy} \mathbf{h}_t + \mathbf{b}_{hy}) \tag{2}$$

where $\sigma_h(\cdot)$ is the hidden activation function and σ_y is the output activation function, such as $softmax(\cdot)$ for outputting a categorical distribution for multi-class classification tasks.

When such a model is trained by backpropagation (more accurately, gradient descent, but hereafter, we will refer to the training process simply as backpropagation), it needs to be "unrolled" in time with shared weights across time steps, and then normal backpropagation can be applied to obtain the gradients of the loss with respect to all trainable weights. This way of back-propagating errors back in time is usually called *backpropagation through time* (BPTT).

BPTT works well for modeling short sequences. However, when it is applied directly to long sequences, due to the long range dependencies between inputs and outputs, it usually fails to propagate gradients across long distances due to the so-called *exploding/vanishing gradient* problem [1,14].

In order to solve this problem, more complex model architectures, such as LSTM [8] and GRU [4], and variants of BPTT, such as Truncated BPTT, gradient clipping and regularization [14] have been proposed. All these methods still use backpropagation at its core, but utilize different ad hoc tricks to make the long range credit assignment work.

Another problem with BPTT is its biological-implausibility. There is little evidence from brain research that supports backpropagation as the learning algorithm for biological learning. The main incompatibilities between backpropagation and our current understanding of biological learning include:

- Backpropagation requires precise knowledge of the non-linearity in the corresponding forward pass. But in biological learning, the feedback paths (if exist) usually consist of a different population of neurons, which makes it hard to match the feedforward counterpart.
- In the backward pass, backpropagation uses the exact symmetric weights of the forward pass (the weight transport problem [11]).
- Current mainstream neural network models use real-valued activations to convey information while most biological neurons use spikes to communicate.
- Backpropagation requires alternating between forward and backward passes.

Other more biologically plausible alternatives have been proposed mainly for feedforward networks. These models include feedback alignment (FA) and its variants [11,13], energy-based models such as equilibrium propagation [7,16] and free energy models [3]. Below, we will give a short introduction to such an algorithm, target propagation [2,10] which uses auto-encoders for credit assignment. In a later section, we will introduce our proposed method to extend target propagation to training multiple-output RNNs.

2.2 Target Propagation

Target propagation is a learning algorithm that uses learned inverse functions between layers to back-propagate activations instead of error derivatives [2,10].

For a multi-layer supervised feed-forward network being trained input $\mathbf{x}$ and label $\mathbf{t}$, we denote the hidden value at the i-th layer as $\mathbf{h}_i$, the feedforward pass sets the activations $\mathbf{h}_i$ for $i = 1, 2, \ldots, M$ where M is the depth of the network and $\mathbf{h}_M$ is the output of the network. The relationships between the activations are defined by

$$\mathbf{h}_i = f_i\left(\mathbf{h}_{i-1}\right) = s_i\left(\mathbf{W}_i \mathbf{h}_{i-1} + \mathbf{b}_i\right), \quad i = 1, \ldots, M \tag{3}$$

where s_i is a non-linear activation function such as $sigmoid(\cdot)$, $ReLU(\cdot)$, W_i and b_i are the parameters for $f_i(\cdot)$, the forward function at layer i, and $\mathbf{h}_0$ and $\mathbf{h}_M$ are the input $\mathbf{x}$ and output of the network respectively. Let's denote the parameters between the i-th layer and j-th layer $(0 \leq i < j \leq M)$ as $\theta^{i,j} = \{(\mathbf{W}_k, \mathbf{b}_k), k = i+1, \ldots, j\}$. Since $\mathbf{h}_j$ is a function of $\mathbf{h}_i$, their relationship can be written as $\mathbf{h}_j = \mathbf{h}_j(\mathbf{h}_i; \theta^{i,j})$. Then a loss function $L(\mathbf{h}_M(\mathbf{x}; \theta^{0,M}), \mathbf{t})$ is defined for the output of the network $\mathbf{h}_M$ with respect to the given label $\mathbf{t}$.

Instead of back-propagating error derivatives, if for each hidden value $\mathbf{h}_i$, we have a nearby target $\hat{\mathbf{h}}_i$ that would make the loss smaller, that is

$$L(\mathbf{h}_M(\hat{\mathbf{h}}_i; \theta^{i,M}), \mathbf{t}) < L\big(\mathbf{h}_M(\mathbf{h}_i(\mathbf{x}; \theta^{0,i}); \theta^{i,M}), \mathbf{t}\big), \tag{4}$$

then during training, we can simply update the parameters such that the hidden values get closer to the layer-wise local targets, thus decreasing the prediction loss. The local optimization problem can be set up by defining a loss function for each layer $i = 1, 2, \ldots, M$,

$$L_i(\mathbf{h}_i, \hat{\mathbf{h}}_i) = L_i(\mathbf{h}_i(\mathbf{x}; \theta^{0,i}), \hat{\mathbf{h}}_i). \tag{5}$$

We can update the weights at each layer with

$$\mathbf{W}_i \leftarrow \mathbf{W}_i - \eta_i \frac{\partial L_i(\mathbf{h}_i, \hat{\mathbf{h}}_i)}{\partial \mathbf{h}_i} \frac{\partial \mathbf{h}_i}{\mathbf{W}_i} \tag{6}$$

$$\mathbf{b}_i \leftarrow \mathbf{b}_i - \eta_i \frac{\partial L_i(\mathbf{h}_i, \hat{\mathbf{h}}_i)}{\partial \mathbf{h}_i} \frac{\partial \mathbf{h}_i}{\mathbf{b}_i} \tag{7}$$

where η_i is a layer-specific learning rate.

We then needs to define the target at the output layer.

$$\hat{\mathbf{h}}_M = \mathbf{h}_M - \hat{\eta} \frac{\partial L(\mathbf{h}_M, \mathbf{t})}{\partial \mathbf{h}_M} \tag{8}$$

where $\hat{\eta}$ is the learning rate to control how close the target is to the output. Note that although we need the derivative to define the last target, we don't need to use the chain rule as in backpropagation. To define the targets for intermediate layers, we need to use an approximate inverse functions $g_i(\cdot)$ at each layer which satisfies

$$f_i(g_i(\mathbf{h}_i)) \approx \mathbf{h}_i \quad \text{or} \tag{9}$$

$$g_i(f_i(\mathbf{h}_{i-1})) \approx \mathbf{h}_{i-1}. \tag{10}$$

These inverse functions could be obtained by training auto-encoders between adjacent layers. Once $g_i(\cdot)$ is defined for each layer, the target for earlier layers $\hat{\mathbf{h}}_i, i = M-1, M-2, \ldots, 1)$ can be obtained using

$$\hat{\mathbf{h}}_i = g_{i+1}(\hat{\mathbf{h}}_{i+1}). \tag{11}$$

or a linearly corrected version of it,

$$\hat{\mathbf{h}}_i = \mathbf{h}_i + g_{i+1}(\hat{\mathbf{h}}_{i+1}) - g_{i+1}(\mathbf{h}_{i+1}). \tag{12}$$

This linearly corrected version of target propagation is called *Difference Target Propagation* [10]. In this paper, we always use the linearly corrected version if not stated otherwise.

In the next section, we will introduce a generalization of target propagation through time for RNNs with multiple outputs and multiple credit assignment paths.

3 Generalizing Target Propagation for RNNs with Multiple Outputs

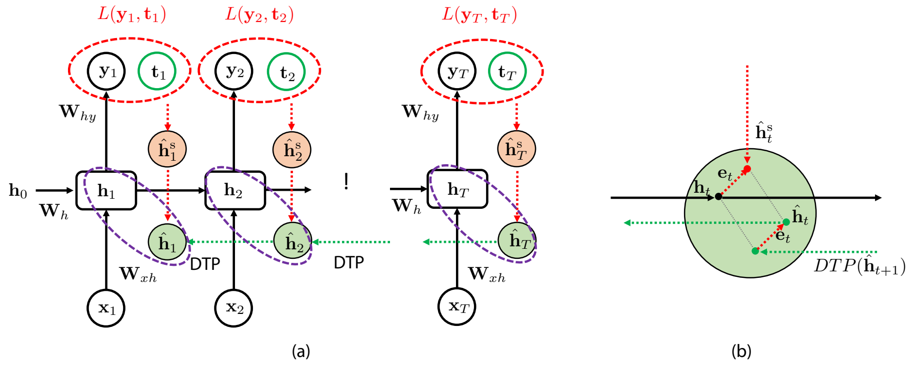

Fig. 1. (a) Propagation of merged targets back in time. For step t, $\mathbf{h}_t$ is the forward activation $\hat{\mathbf{h}}^s_t$ is the step target set with respect to the step loss $L(\mathbf{y}_t, \mathbf{t}_t)$ and $\hat{\mathbf{h}}_t$ is the updated step target used for training obtained by merging the step target and back-propagated target from future time steps. (b) Merging step target and back propagating target linearly. This merging step ensures that only one target is propagated back in time, similar to backpropagation in which the step gradients are accumulated when being back propagated.

We first need to define a simple RNN model by specifying its forward path,

$$\begin{aligned} \mathbf{h}_t &= F\left(\mathbf{x}_t, \mathbf{h}_{t-1}\right) & (13)\\ &= \sigma\left(\mathbf{W}_{xh} \cdot \mathbf{x}_t + \mathbf{W}_h \cdot \mathbf{h}_{t-1} + \mathbf{b}_h\right) & (14)\\ \mathbf{y}_t &= \text{softmax}(\mathbf{W}_{hy} \cdot \mathbf{h}_t + \mathbf{b}_y) & (15) \end{aligned}$$

where $\mathbf{W}_{xh}, \mathbf{W}_h, \mathbf{b}_h, \mathbf{W}_{hy}$ and $\mathbf{b}_y$ are the model parameters. Similar to TPTT [12], we also define a step-wise approximate inverse function,

$$\begin{aligned} \mathbf{h}_{t-1} &\approx G\left(\mathbf{x}_t, \mathbf{h}_t\right) & (16)\\ &= \sigma(\mathbf{V}_h \cdot \mathbf{h}_t + \mathbf{W}_{xh} \cdot \mathbf{x}_t + \mathbf{c}_h). & (17) \end{aligned}$$

This approximate inverse function can be trained as the decoder of an auto-encoder, i.e. trained to reconstruct $\mathbf{h}_{t-1}$ from $\mathbf{h}_t$. In order for the proposed method to work well, and more generally for target propagation to work well, the inverse functions need to be good enough with the linear correction. It can be trained as a denoising auto-encoder [10] so that it works well in the neighborhood of the forward activations. In addition, we can also use a more complex function such as a two-layer neural network to approximate the inverse function, but in training, this requires the use of chain rule. In our derivation above, we choose the inverse function to be in the same form as the forward step function.

If the target at step t is $\hat{\mathbf{h}}_t$, then we can use difference target propagation (DTP) and the approximate inverse function $G(\cdot)$ to define the target at step $t-1$ as

$$\hat{\mathbf{h}}_{t-1} = \mathrm{DTP}(\hat{\mathbf{h}}_t) \tag{18}$$

$$= G(\mathbf{x}_t, \hat{\mathbf{h}}_t) + (\mathbf{h}_{t-1} - G(\mathbf{x}_t, \mathbf{h}_t)) \tag{19}$$

This linear correction is used to stabilize the training [10]. We have defined how the targets could be back propagated in time. Next, we will define the local step targets generated by local step-wise losses. Let the loss at step t be $\mathcal{E}_t = L(\mathbf{y}_t, \mathbf{t}_t)$ where $\mathbf{y}_t$ is the output and $\mathbf{t}_t$ the correct label. Then the local step target could be defined as

$$\hat{\mathbf{h}}_t^{\mathrm{s}} = \mathbf{h}_t - \alpha_i \cdot \frac{\partial \mathcal{E}_t}{\partial \mathbf{h}_t} \tag{20}$$

where α_i is defined as the initial step size, controlling how far the local target is from the corresponding activation (Fig. 1(a)). This step cannot be to large because the inverse functions can only work well in the neighborhood of the forward activations. Once the local step-wise targets have been defined, we then need a way to merge these local targets while the targets are being back propagated, i.e., defining the relationship between $\hat{\mathbf{h}}_t$ and $\hat{\mathbf{h}}_t^{\mathrm{s}}$.

Unlike backpropagation, in target propagation, the backward pass, like the forward pass, are non-linear, so it is hard to define a merging strategy that would make target propagation approximate backpropagation exactly. Thus, we simplify this target merging process by making it linear (Fig. 1(b)),

$$\mathbf{e}_t = \hat{\mathbf{h}}_t^{\mathrm{s}} - \mathbf{h}_t \tag{21}$$

$$\hat{h}_t = \mathrm{DTP}(\hat{h}_{t+1}) + \mathbf{e}_t \tag{22}$$

The above target merging rule has an intuitive interpretation: as the back propagating target passes through the backward path, it accumulates local errors similar to how backpropagation accumulates gradients during backward pass, but in a non-linear way. Once the merged step targets $\hat{\mathbf{h}}_t$ are defined for each step after the backward target propagation pass, we can define a local loss term (e.g., using MSE$(\cdot)$) for updating the weights $\mathbf{W}_{xh}, \mathbf{W}_h$ and $\mathbf{b}_h$ respectively,

$$\mathbf{W}_h \leftarrow \mathbf{W}_h - \alpha_f \sum_{t=1}^{T} \frac{\partial \,\mathrm{MSE}\left(F\left(\mathbf{x}_t, \mathbf{h}_{t-1}\right), \hat{\mathbf{h}}_t\right)}{\partial \mathbf{W}_{hh}} \tag{23}$$

$$\mathbf{b}_h \leftarrow \mathbf{b}_h - \alpha_f \sum_{t=1}^{T} \frac{\partial \,\mathrm{MSE}\left(F\left(\mathbf{x}_t, \mathbf{h}_{t-1}\right), \hat{\mathbf{h}}_t\right)}{\partial \mathbf{b}_h}. \tag{24}$$

Since the weights are shared across time steps, the updates applied to the weights are the sum of the updates cross all time steps. Note that this optimization problem is local and doesn't require chain rule as in backpropagation.

4 Experiments

4.1 Tasks

Copy Memory Task. In order to test the capability of the proposed method in training recurrent neural networks and understand its limitations, we first used the copy memory task commonly used to test the models' ability to retain information over long ranges. The first 10 elements in the input sequence are drawn uniformly randomly from k symbols, and it is then followed by $T - 1$ blank symbol. A special symbol is followed to indicate that the model should start to recall the first 10 elements in the input, which is followed by another 10 blank symbols. For the correct output, the first $(T + 10)$ elements should be blank symbols. When it receives the start recall symbol from input at the $(T + 10)$-th position, it will output the first 10 elements of the input as its last 10 elements. If, instead of 10 elements, we only consider memorizing the first 3 elements as an example, with a delay $T = 3$, the input and corresponding correct output will be $\{a, c, b, \_, \_, *, \_, \_, \_\}$ and $\{\_, \_, \_, \_, \_, \_, a, c, b\}$ where "$*$" is the start recalling signal symbol and "_" is the blank symbol. In this task, the information of the first 10 elements of the input should be kept in the model for at least T steps. Thus, as we increase the delay T, it is harder and harder for the network to recall the first 10 elements.

The model we use is a simple recurrent neural network model with 128 hidden units and $tanh(\cdot)$ activation function. We use $k = 8$ symbols for the 10 elements to be memorized and use another two special symbols as the start recalling signal symbol and blank symbol. All symbols are one-hot encoded.

In this task, a memory-less baseline method is to output blank spaces until receiving the start recalling symbol, and then output 10 random symbols. For this strategy, the expected cross-entropy loss can be calculated as $\frac{10\ln(8)}{T+20}$, which will be used as a baseline when comparing the training losses by different optimization methods. Performance at this baseline means that the recalled sequence is random and information of the first 10 elements in the input sequence has been lost.

At each delay T, we perform a hyper-parameter search to find the best learning rates for backpropagation and the proposed target propagation respectively. With a mini-batch size of 20, we generate training batches on the fly for a total of 25000 batches in total, and use the same validation set of 500 batches for validation. After training for 25000 batches in total, we report the best validation loss and accuracy.

Table 1. Results for the copy memory task using proposed target propagation method and backpropagation. The baseline is calculated using $\frac{10\ln(8)}{20+T}$. The accuracy is the percentage of correctly predicted sequences. For example, with a delay of 5 (total sequence length = 25), a baseline model that outputs random symbols in the last 10 steps and blank elsewhere will have an accuracy of $\frac{1}{8^{10}} \approx 9.31 \times 10^{-10}$ in expectation. It can be shown that our proposed target propagation managed to get a much lower loss than baseline with high recovery accuracy up to a delay of 25 steps. Backpropagation can only get loss close to the baseline, not able to propagate information for long ranges. These two methods use the same weight initialization method.

Delay (T)	Target prop		Backprop		Baseline
	Loss	Accuracy	Loss	Accuracy	Baseline loss
5	3.59×10^{-3}	100%	0.6899	0%	0.8318
10	1.30×10^{-3}	100%	0.5809	0%	0.6931
15	1.34×10^{-4}	99.99%	0.5182	0%	0.5941
20	5.70×10^{-3}	98.32%	0.5205	0%	0.5199
25	1.36×10^{-2}	92.88%	0.4625	0%	0.4621

Table 2. Results for the sequence expansion task using proposed target propagation method and backpropagation.

Sequence length (T)	Target prop		Backprop	
	Loss	Accuracy	Loss	Accuracy
10	0.0005	100.00%	0.0001	100%
20	0.0018	99.34%	0.0005	100%
30	0.3233	3.62%	0.0007	100%
40	1.0771	0%	0.0012	99.49%

The result is shown in Table 1. The accuracy is the percentage of correct sequence predictions in the validation set. As we can see, our proposed target propagation method managed to beat the baseline and achieve a high prediction accuracy until a delay of at least 25 steps. Backpropagation can only achieve a loss at a similar level as the memory-less baseline, and with an accuracy of 0%, it cannot output a single correct sequence out of the 10000 validation samples.

This results shows that target propagation is able to propagate error information over long range while backpropagation cannot. In order to see if this is due to the exploding/vanishing gradient problem, we computed the spectral radius

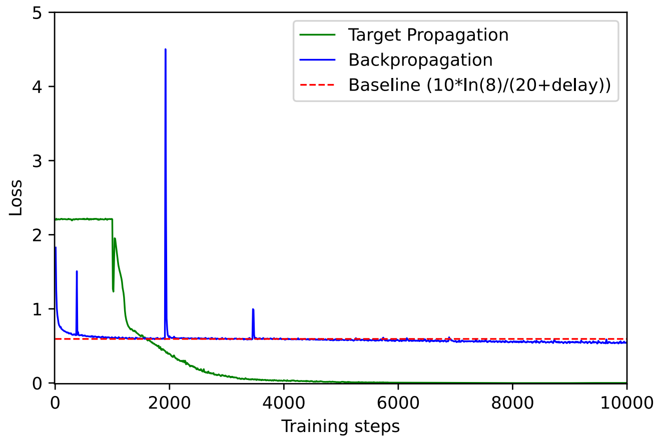

Fig. 2. The training loss of backpropagation and the proposed target propagation method for the copy memory task with a delay of 15 steps. Each step is a single weight update with a mini-batch of training examples. The baseline loss is from a memory-less strategy: outputting blanks until receiving the start-recall symbol, followed by a random sequence of 10 symbols. In target propagation, the first 1000 steps are used to train inverse function only, thus the loss doesn't change.

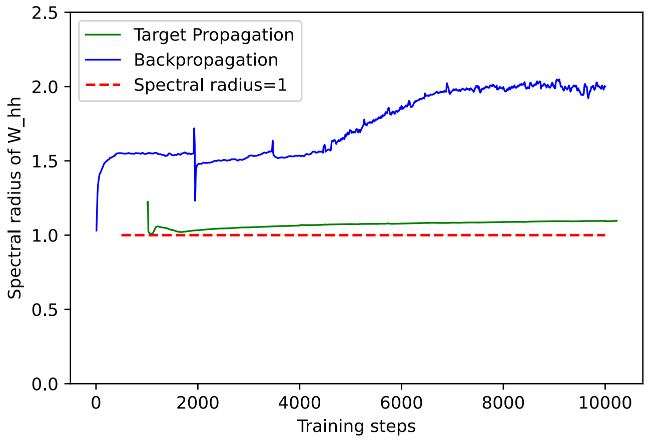

Fig. 3. The spectral radius of the transition weight matrix $\mathbf{W}_h$ during learning for proposed target propagation method and backpropagation.

of the hidden-to-hidden transition weight matrix $\mathbf{W}_h$ defined as the magnitude of the largest eigenvalue. From Fig.3, we can see that the spectral radius of the transition weight matrix $\mathbf{W}_h$ stays relatively stable around value 1.0 for target propagation. However, the spectral radius of the matrix during backpropagation is not stable and deviates significantly from 1.0. Although this figure only shows the spectral radius for one trial, this applies to other trials as well. In addition, we observed that a stable spectral radius around 1.0 during training is correlated with better training results.

Sequence Expansion Task. The copy memory task tests the ability of the learning algorithm to propagate information across long ranges. One drawback of the copy memory task is that regardless of the learning algorithm we use, it will converge quickly to the memory-less baseline solution, and the only challenge remaining is to propagate the error information from the end to the beginning of the sequence without much interference from intermediate positions along the sequence, similar to training a many-to-one RNN model. Thus, we designed a second synthetic task, sequence expansion task, in which each step will generate useful error information during the training process.

In the second task, sequence expansion task, the input is a sequence of length T. The first $T/2$ elements are drawn uniformly randomly from a set of k symbols, and the second half are all blanks represented by a special blank symbol. The task is to duplicate each element of the first half exactly once. For example, the correct output of the input sequence $\{a, c, b, *, *, *\}$ will be $\{a, a, c, c, b, b\}$. The output will have the same length as the input but without blanks. As the sequence length T increases, in order to make correct prediction, the model needs to memorize the $(T/2)$-th element in the input for at least $(T/2-1)$ steps. In contrast to the copy memory task, there is no simple memory-less baseline method and the error information at each step will be useful in training.

As the results shown in Table 2, backpropagation can solve this task easily, while target propagation could not when sequence length grows to greater than 30. The likely explanation is that backpropagation is better at combining error gradients when they are being back propagated in time since the operation is linear. While for target propagation, when there is useful error information at each step, it has difficulty merging them due to the non-linearity in the target back propagation process.

When comparing these two tasks, we can see that backpropagation usually suffers from vanishing/exploding gradient problem when the credit assignment path is too long. Target propagation is better at propagating error information across long ranges as long as there are not many merging targets along the credit assignment paths.

5 Conclusions

In this paper, we generalized target propagation to training recurrent neural networks with multiple outputs and showed that it is better at propagating long range error information compared to backpropagation, which usually suffers from the exploding/vanishing gradient problem. However, when the tasks involve many credit assignment paths that needs to be merged, such as the case in the sequence expansion task, backpropagation generally performs better. This might be due to the linear nature of backpropagation when the derivatives could be simply added along the backward pass. A potential future improvement for target propagation for RNN could be truncating the propagating targets at every a few steps, similar to truncated BPTT. The linear merging operation in the proposed method could potentially be replaced by a single many-to-one inverse function that only produce one output without the need for merging the targets explicitly. In principle, the proposed method could potentially be applied to learning in arbitrary computation graphs when there might be intersecting credit assignment paths.

References

1. Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE Trans. Neural Netw. **5**(2), 157–166 (1994)
2. Bengio, Y.: How Auto-Encoders could provide credit assignment in deep networks via target propagation. ArXiv (July 2014)
3. Bogacz, R.: A tutorial on the free-energy framework for modelling perception and learning. J. Math. Psychol. **76**(Pt B), 198–211 (2017)
4. Chung, J., Gulcehre, C., Cho, K., Bengio, Y.: Empirical evaluation of gated recurrent neural networks on sequence modeling. ArXiv (December 2014)
5. Crick, F.: The recent excitement about neural networks. Nature **337**(6203), 129–132 (1989)
6. Elman, J.L.: Finding structure in time. Cogn. Sci. **14**(2), 179–211 (1990)
7. Ernoult, M., Grollier, J., Querlioz, D., Bengio, Y., Scellier, B.: Equilibrium propagation with continual weight updates. ArXiv (April 2020)
8. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
9. Jordan, M.I.: Chapter 25 - serial order: A parallel distributed processing approach. In: Donahoe, J.W., Packard Dorsel, V. (eds.) Advances in Psychology, vol. 121, pp. 471–495. North-Holland (January 1997)
10. Lee, D.-H., Zhang, S., Fischer, A., Bengio, Y.: Difference target propagation. In: Appice, A., Rodrigues, P.P., Santos Costa, V., Soares, C., Gama, J., Jorge, A. (eds.) ECML PKDD 2015. LNCS (LNAI), vol. 9284, pp. 498–515. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23528-8_31
11. Lillicrap, T.P., Cownden, D., Tweed, D.B., Akerman, C.J.: Random feedback weights support learning in deep neural networks. ArXiv (November 2014)
12. Manchev, N., Spratling, M.W.: Target propagation in recurrent neural networks. J. Mach. Learn. Res. **21**(7), 1–33 (2020)
13. Nøkland, A.: Direct feedback alignment provides learning in deep neural networks. In: Advances in Neural Information Processing Systems, pp. 1037–1045 (2016)
14. Pascanu, R., Mikolov, T., Bengio, Y.: Understanding the exploding gradient problem. ArXiv (2012)
15. Rumelhart, D.E., McClelland, J.L.: Learning internal representations by error propagation. In: Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations, pp. 318–362. MIT Press (1987)
16. Scellier, B., Bengio, Y.: Equilibrium propagation: bridging the gap between energy-based models and backpropagation. Front. Comput. Neurosci. **11**, 24 (2017)
17. Werbos, P.J.: Backpropagation through time: what it does and how to do it. Proc. IEEE **78**(10), 1550–1560 (1990)