Reviving and Improving Recurrent Back-Propagation

Renjie Liao [*1 2 3] **Yuwen Xiong** [*1 2] **Ethan Fetaya** [1 3] **Lisa Zhang** [1 3] **KiJung Yoon** [4 5] **Xaq Pitkow** [4 5]
Raquel Urtasun [1 2 3] **Richard Zemel** [1 3 6]

Abstract

In this paper, we revisit the recurrent back-propagation (RBP) algorithm (Almeida, 1987; Pineda, 1987), discuss the conditions under which it applies as well as how to satisfy them in deep neural networks. We show that RBP can be unstable and propose two variants based on conjugate gradient on the normal equations (CG-RBP) and Neumann series (Neumann-RBP). We further investigate the relationship between Neumann-RBP and back propagation through time (BPTT) and its truncated version (TBPTT). Our Neumann-RBP has the same time complexity as TBPTT but only requires constant memory, whereas TBPTT's memory cost scales linearly with the number of truncation steps. We examine all RBP variants, along with BPTT and TBPTT, in three different application domains: associative memory with continuous Hopfield networks, document classification in citation networks using graph neural networks, and hyperparameter optimization for fully connected networks. All experiments demonstrate that RBPs, especially the Neumann-RBP variant, are efficient and effective for optimizing convergent recurrent neural networks.

1. Introduction

Back-propagation through time (BPTT) (Werbos, 1990) is nowadays the standard approach for training recurrent neural networks (RNNs). However, the computation and memory cost of BPTT scale linearly with the number of steps which makes BPTT impractical for applications where long sequences are common (Sutskever et al., 2014; Goodfellow et al., 2016). Moreover, as the number of unrolling steps increases, the numerical error accumulates which may render the algorithm useless in some applications, e.g., gradient-based hyperparameter optimization (Maclaurin et al., 2015). This issue is often solved in practice by using truncated back-propagation through time (TBPTT) (Williams & Peng, 1990; Sutskever, 2013) which has constant computation and memory cost, is simple to implement, and effective in some applications. However, the quality of the TBPTT approximate gradient is not well understood. A natural question to ask is, can we get better gradient approximations while still using the same computational cost as TBPTT?

Here will show that under certain conditions on the underlying model, the answer is positive. In particular, we consider a class of RNNs whose hidden state converges to a steady state. For this class of RNNs, we can bypass BPTT and compute the exact gradient using an algorithm called recurrent back-propagation (RBP) (Almeida, 1987; Pineda, 1987). The key observation exploited by RBP is that the gradient of the steady state w.r.t. the learnable parameters can be directly computed using the implicit function theorem, alleviating the need to unroll the entire forward pass. The main computational cost of RBP is in solving a linear system which has constant memory and computation time w.r.t. the number of unrolling steps. However, due to the strong assumptions that RBP imposes, TBPTT has become the standard approach used in practice and RBP did not get much attention for many years.

In this paper, we first revisit RBP in the context of modern deep learning. We discuss the original algorithm, the assumptions it imposes and how to satisfy them for deep neural networks. Second, we notice that although the fixed point iteration method used in (Almeida, 1987; Pineda, 1987) is guaranteed to converge if the steady hidden state is achievable, in practice it can fail to do so within a reasonable amount of steps. This may be caused by the fact that there are many fixed points and the algorithm is sensitive to initialization. We try to overcome the instability issue by proposing two variants of RBP based on conjugate gradient on normal equations (CG-RBP) and Neumann series (Neumann-RBP). We show a connection between Neumann-RBP and TBPTT which sheds some new light on the approximation quality of TBPTT. In the experiments, we show

[*]Equal contribution [1]Department of Computer Science, University of Toronto [2]Uber ATG Toronto [3]Vector Institute [4]Department of Electrical and Computer Engineering, Rice University [5]Department of Neuroscience, Baylor College of Medicine [6]Canadian Institute for Advanced Research. Correspondence to: Renjie Liao <rjliao@cs.toronto.edu>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

several important applications which are naturally amenable to RBP. For example, we show how RBP can be used to back propagate thorough the optimization of deep neural networks in order to tune hyperparameters. Throughout our experiments, we found that Neumann-RBP not only inherits the advantages of original RBP but also remains consistently stable across different applications.

2. Related Work

In the context of neural networks, RBP was independently discovered by Almeida (Almeida, 1987) and Pineda (Pineda, 1987) in 1987, which is why this algorithm is sometimes called the Almeida-Pineda algorithm. Back then, RBP was shown to be useful in learning content-addressable memory (CAM) models (Hopfield, 1982; 1984) and other computational neurodynamic models (Lapedes & Farber, 1986; Haykin, 1993; Chauvin & Rumelhart, 1995). These models are special RNNs in a sense that their inference stage is a convergent dynamic system by design. For these systems, one can construct a Lyapunov function for the underlying dynamics which further guarantees the asymptotic stability. We refer readers to chapter 13 of (Haykin, 1993) for more details on neurodynamic models. The goal of learning in these models is to manipulate the attractors, *i.e.* steady states, such that they are close to the input data. Therefore, during inference stage, even if the input data is corrupted, the corresponding correct attractor or “memory“ can still be retrieved. Instead of computing gradient via BPTT, RBP provides an more efficient alternative for manipulating the attractors.

RBP was later applied to learning graph neural networks (GNNs) (Scarselli et al., 2009), which are generalizations of RNNs that handle graph-structured input data. Specifically, the inference of GNNs is essentially a propagation process which spreads information along the graph. One can force the propagation process to converge by either constructing a contraction map explicitly, or by regularizing the Jacobian of the update function. Similarly, the goal is to push the converged inference solution close to the target. RBP is naturally applicable here and demonstrated to save both computation time and memory. A recent investigation (Scellier & Bengio, 2017a) shows that RBP is related to equilibrium propagation (Scellier & Bengio, 2017b) which is motivated from the perspective of biological plausibility. Another recent related work in deep learning is OptNet (Amos & Kolter, 2017) where the gradient of the optimized solution of a quadratic programming problem w.r.t. parameters is obtained by analytically differentiating the KKT system.

In the probabilistic graphical models (PGMs) literature, similar techniques to RBP have been developed as well. For example, an efficient gradient-based method to learn the hyperparameters of log-linear models is provided in (Foo et al., 2008) where the core contribution is to use the implicit differentiation trick to compute the gradient of the optimized inference solution w.r.t. the hyperparameters. A similar implicit differentiation technique is used in (Samuel & Tappen, 2009) to optimize the maximum a posterior (MAP) solution of continuous MRFs, since the MAP solution can be regarded as the steady state of the inference process. An implicit-differentiation-based optimization method for generic energy models is proposed in (Domke, 2012) where the gradient of the optimal state (steady state) of the energy w.r.t. the parameters can be efficiently computed given the fast matrix-vector product implementation (Pearlmutter, 1994). If one regards the inference algorithms from aforementioned applications as unrolled RNNs, the implicit differentiation technique is essentially equivalent to RBP.

Other efforts have been made to develop alternatives to BPTT. NoBackTrack (Ollivier et al., 2015) maintains an online estimate of the gradient via the random rank-one reduction technique. ARTBP (Tallec & Ollivier, 2017) introduces a probability distribution over the truncation points in the sequence and compensates the truncated gradient based on the distribution. Both approaches provide an unbiased estimation of the gradient although their variances differ.

3. Revisiting Recurrent Back-Propagation

In this section, we review the original RBP algorithm and discuss its assumptions.

3.1. Recurrent Back-Propagation

We denote the input data and initial hidden state as x and h_0. During inference, the hidden state at time t is computed as follows,

$$h_{t+1} = F(x, w_F, h_t), \tag{1}$$

where F is the update function parameterized by w_F. A typical instantiation of F is an LSTM (Hochreiter & Schmidhuber, 1997) cell function. This RNN formulation differs from the one commonly used in language modeling, as the input is not time-dependent. We restrict our attention to RNNs with fixed inputs for now as it requires fewer assumptions. Assuming the dynamical system, (i.e., the forward pass of the RNN), reaches steady state before time step T, we have the following equation,

$$h^* = F(x, w_F, h^*), \tag{2}$$

where h^* is the steady hidden state. We compute the predicted output y based on the steady hidden state as follows,

$$y = G(x, w_G, h^*), \tag{3}$$

where G is the output function parameterized by w_G. Typically, a loss function $L = l(\bar{y}, y)$ measures the closeness between ground truth $\bar{y}$ and predicted output y. Since the input data x is fixed for all time steps, we can construct a function Ψ of w_F and h as follows,

$$\Psi(w_F, h) = h - F(x, w_F, h). \tag{4}$$

At the fixed point, we have $\Psi(w_F, h^*) = 0$. Assuming some proper conditions on F, e.g., continuous differentiability, we can take the derivative w.r.t. w_F at h^* on both sides. Using the total derivative and the dependence of h^* on w_F we obtain,

$$\begin{aligned} \frac{\partial \Psi(w_F, h^*)}{\partial w_F} &= \frac{\partial h^*}{\partial w_F} - \frac{\mathrm{d}F(x, w_F, h^*)}{\mathrm{d}w_F} \\ &= (I - J_{F,h^*}) \frac{\partial h^*}{\partial w_F} - \frac{\partial F(x, w_F, h^*)}{\partial w_F} \\ &= \mathbf{0}, \end{aligned} \tag{5}$$

where $J_{F,h^*} = \frac{\partial F(x, w_F, h^*)}{\partial h}$ is the Jacobian matrix of F evaluated at h^* and d is the total derivative operator. Assuming that $I - J_{F,h^*}$ is invertible, we rearrange Eq. (5) to get,

$$\frac{\partial h^*}{\partial w_F} = (I - J_{F,h^*})^{-1} \frac{\partial F(x, w_F, h^*)}{\partial w_F}. \tag{6}$$

In fact, Equations (4- 6) are an application of the Implicit Function Theorem (Rudin, 1964), which guarantees the existence and uniqueness of an implicit function ϕ such that $h^* = \phi(w_F)$ if two conditions hold: I, Ψ is continuously differentiable and II, $I - J_{F,h^*}$ is invertible. Although we do not know the analytic expression of the function ϕ, we can still compute its gradient at the fixed point.

Based on Eq. (6), we now turn our attention towards computing the gradient of the loss w.r.t. the parameters of the RNN. By using the total derivative and the chain rule, we have

$$\frac{\partial L}{\partial w_G} = \frac{\partial L}{\partial y} \frac{\partial G(x, w_G, h^*)}{\partial w_G} \tag{7}$$

$$\frac{\partial L}{\partial w_F} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial h^*} (I - J_{F,h^*})^{-1} \frac{\partial F(x, w_F, h^*)}{\partial w_F}. \tag{8}$$

Since the gradient of the loss w.r.t. w_G can be easily obtained by back-propagation, we focus our exposition on the computation of $\frac{\partial L}{\partial w_F}$. The original RBP algorithm (Pineda, 1987; Almeida, 1987) introduces an auxiliary variable z such that,

$$z = \left(I - J_{F,h^*}^{\top}\right)^{-1} \left(\frac{\partial L}{\partial y} \frac{\partial y}{\partial h^*}\right)^{\top}, \tag{9}$$

where z is a column vector. If we managed to compute z, then we can substitute it into Eq. (8) to get the gradient. Note that the Jacobian matrix J_{F,h^*} is nonsymmetric for general RNNs which renders direct solvers of linear system impractical. To compute z, the original RBP algorithm uses fixed point iteration. In particular, we multiply $\left(I - J_{F,h^*}^{\top}\right)$ on the left hand of both sides of Eq. (9) and rearrange the terms as follows,

Algorithm 1 : Original RBP

```
Initialization: initial guess $z_0$, e.g., draw uniformly from $[0, 1]$, $i = 0$, threshold $\epsilon$
repeat
    $i = i + 1$
    $z_i = J_{F,h^*}^{\top} z_{i-1} + \left(\frac{\partial L}{\partial y}\frac{\partial y}{\partial h^*}\right)^{\top}$
until $\|z_i - z_{i-1}\| < \epsilon$
$\frac{\partial L}{\partial w_F} = z_i^{\top} \frac{\partial F(x, w_F, h^*)}{\partial w_F}$
Return $\frac{\partial L}{\partial w_F}$
```

$$z = J_{F,h^*}^{\top} z + \left(\frac{\partial L}{\partial y} \frac{\partial y}{\partial h^*}\right)^{\top}. \tag{10}$$

If we view the right hand side of the above equation as a function of z, then applying the fixed point iteration results in the Algorithm 1. Note that the most expensive operation in this algorithm is the matrix-vector product $J_{F,h^*}^{\top} z$, which is the same operator as back-propagation.

3.2. Assumptions of RBP

In this section, we discuss how to satisfy the assumptions of RBP. Recall that in order to apply the implicit function theorem, $\Psi(w_F, h)$ has to satisfy two assumptions: I, Ψ is continuously differentiable. II, $I - J_{F,h^*}$ is invertible. Condition I requires the derivative of F to be continuous, a condition satisfied by many RNNs, like LSTM and GRU (Cho et al., 2014). Condition II is equivalent to requiring the determinant of $I - J_{F,h^*}$ to be nonzero, i.e., $\det(I - J_{F,h^*}) \neq 0$. One sufficient but not necessary condition to ensure this is to force F to be a contraction map, as in Scarselli et al. (2009). Recall that F is a contraction map on Banach space B, i.e., a complete normed vector space, iff, $\forall h_1, h_2 \in B$, $\|F(h_1) - F(h_2)\| \leq \mu \|h_1 - h_2\|$ where $0 \leq \mu < 1$. Banach fixed point theorem guarantees the uniqueness of the fix point of the contraction map F in B. Note that here we drop the dependency of F on w for readability. Based on the first order Taylor approximation, $F(h) = F(h^*) + J_{F,h^*}(h - h^*)$, we have,

$$\frac{\|F(h) - F(h^*)\|}{\|h - h^*\|} = \frac{\|J_{F,h^*}(h - h^*)\|}{\|h - h^*\|}. \tag{11}$$

Note that if we use L_2 vector norm, then the induced matrix norm, a.k.a., operator norm, is,

$$\|J_{F,h^*}\| = \sup\left\{\frac{\|J_{F,h^*} h\|}{\|h\|} : \forall h \neq 0\right\} = \sigma_{\max}(J_{F,h^*}), \tag{12}$$

where $\sigma_{\max}$ is the largest singular value. Therefore, relying on the contraction map definition, we have,

$$\|J_{F,h^*}\| \leq \mu < 1, \tag{13}$$

Moreover, since the minimum singular value of $I - J_{F,h^*}$ is $1 - \sigma_{\max}(J_{F,h^*})$, we have

$$\begin{aligned} |\det(I - J_{F,h^*})| &= \prod_i |\sigma_i(I - J_{F,h^*})| \\ &\geq [1 - \sigma_{\max}(J_{F,h^*})]^d > 0. \end{aligned} \tag{14}$$

Thus our second condition holds following Eq. (14).

Scarselli et al. (2009) use L_1 vector norm which results in a looser inequality since $\|J_{F,h^*}\|_2 \leq \sqrt{d}\|J_{F,h^*}\|_1$. They obtain an easier to compute regularization term $\max_i(\|J_{F,h^*}(:,i)\|_1 - \eta)^2$ where $(:, i)$ denotes the i-th column and $\eta \in (0, 1)$ is the desired contraction constant. We note, however, that this work makes a claim that the contraction map assumption can be achieved by regularizing the local Jacobian J_{F,h^*} of a general neural network. This is problematic because the contraction map property is a global property of F that requires regularizing every h in the space B, not just h^*. Nevertheless, this regularization evaluated at h^* encourages local contraction at the fixed point h^*, which is sufficient for satisfying condition II. Another way to enforce condition II to hold is directly formalizing the Lagrangian of equality constraint $\Psi(w_F, h^*) = 0$. Since all applications we considered in this paper have converged dynamic systems in practice, we leave further discussions of condition II to the appendix.

4. New Recurrent Back-Propagation Variants

In this section, we present our newly proposed variants of RBP, CG-RBP and Neumann-RBP, in detail.

4.1. Recurrent Back-Propagation based on Conjugate Gradient

Facing the system of linear equations like Eq. (9) in the derivation of original RBP, one would naturally think of the most common iterative solver, i.e., conjugate gradient method (Hestenes & Stiefel, 1952). In particular, multiplying $I - J_{F,h^*}^\top$ on both sides, we obtain the following equations,

$$\left(I - J_{F,h^*}^\top\right) z = \left(\frac{\partial L}{\partial y}\frac{\partial y}{\partial h^*}\right)^\top. \tag{15}$$

Unfortunately, for general RNNs, the Jacobian matrix J_{F,h^*} of the update function, e.g., a cell function of LSTM, is non-symmetric in general. This increases the difficulty of solving the system. One simple yet sometimes effective way to approach this problem is to exploit the conjugate gradient method on the normal equations (CGNE) (Golub & Van Loan, 2012). Specifically, we multiply $I - J_{F,h^*}$ on both sides of Eq. (15) which results in,

$$\left(I - J_{F,h^*}\right)\left(I - J_{F,h^*}^\top\right) z = \left(I - J_{F,h^*}\right)\left(\frac{\partial L}{\partial y}\frac{\partial y}{\partial h^*}\right)^\top.$$

Having a symmetric matrix multiplying z on the left hand side, we can now use the conjugate gradient method. The detailed algorithm is easily obtained by instantiating the standard conjugate gradient (CG) template. The most expensive operation used in CGNE is $J_{F,h^*}J_{F,h^*}^\top z$, which can be implemented by successive matrix-vector products similarly for computing the Fisher information product of the natural gradient method (Schraudolph, 2002). Once we solve z via K-step CGNE, we obtain the final gradient by substituting the solution into Eq. (8). Since the condition number of the current system is the square of the original one, the system may be slower to converge in practice. Exploring more advanced and faster convergent numerical methods under this setting, like LSQR (Paige & Saunders, 1982), is left for future work.

4.2. Recurrent Back-Propagation based on Neumann Series

We now develop a new RBP variant called Neumann-RBP, which uses Neumann series from functional analysis and is efficient in terms of computation and memory. We then show its connections to BPTT and TBPTT.

A Neumann series is a mathematical series of the form $\sum_{t=0}^{\infty} A^t$ where A is an operator. In matrix theory, it is also known as the geometric series of a matrix. A convergent Neumann series has the following property,

$$(I - A)^{-1} = \sum_{k=0}^{\infty} A^k. \tag{16}$$

One sufficient condition of convergence is that the spectral radius (i.e., the largest absolute eigenvalue value) of A is less than 1. This convergence criterion applied to $A = J_{F,h^*}$ implies condition II. Other cases where the convergence hold is beyond the scope of this paper. If the Neumann series $\sum_{t=0}^{\infty} J_{F,h^*}^t$ converges, we can use it to replace the term $\left(I - J_{F,h^*}\right)^{-1}$ in E.q. (8). Furthermore, the gradient of RBP can be approximated with the K-th order truncation of Neumann series as below,

$$\frac{\partial L}{\partial w_F} \approx \frac{\partial L}{\partial y}\sum_{k=0}^{K}\frac{\partial y}{\partial h^*}J_{F,h^*}^k\frac{\partial F(x, w_F, h^*)}{\partial w_F}. \tag{17}$$

There is a rich body of literature on how to compute Neumann series efficiently using binary or ternary decomposition (Westreich, 1989; Vassil & Diego, 2017). However,

Algorithm 2 : Neumann-RBP

```
Initialization: $v_0 = g_0 = \left(\frac{\partial L}{\partial y}\frac{\partial y}{\partial h^*}\right)^\top$
for step $t = 1, 2, \dots, K$ do
    $v_t = J^\top v_{t-1}$
    $g_t = g_{t-1} + v_t$
end for
$\frac{\partial L}{\partial w_F} = (g_K)^\top \frac{\partial F(x, w_F, h^*)}{\partial w_F}$
Return $\frac{\partial L}{\partial w_F}$
```

these decomposition based approaches are inapplicable in our context since we cannot compute the Jacobian matrix J_{F,h^*} efficiently for general neural networks. Fortunately, we can instead efficiently compute the matrix-vector product $J_{F,h^*}^\top u$ and $J_{F,h^*} u$ (u is a proper sized vector) by using reverse and forward mode auto-differentiation (Pearlmutter, 1994). Relying on this technique, we summarize the Neumann series based RBP algorithm in Algorithm 2. In practice, we can obtain further memory efficiency by performing updates within the for loops in-place (please refer to the example code in appendix), so that memory usage need not scale with the number of truncation steps. Moreover, since the algorithm does not rely on hidden states except the fixed point h^*, we no longer need to store the hidden states in the forward pass of the RNN. Besides the computational benefit, we now have the following proposition to connect Neumann-RBP to BPTT and TBPTT.

Proposition 1. *Assume that there exists some step K where $0 < K \leq T$ such that for the convergent sequence of hidden states $h^0, h^1, \dots, h^T$ of an RNN, we have $h^* = h^T = h^{T-1} = \cdots = h^{T-K}$ where h^* is the fixed point. If the Neumann series $\sum_{t=0}^{\infty} J_{F,h^*}^t$ converges, then the full and K-step Neumann-RBP are equivalent to BPTT and K-step TBPTT respectively.*

Moreover, the following proposition bounds the error of K-step Neumann-RBP.

Proposition 2. *If the Neumann series $\sum_{t=0}^{\infty} J_{F,h^*}^t$ converges, then the error between K-step and full Neumann series is as follows,*

$$\left\| \sum_{t=0}^{K} J_{F,h^*}^t - (I - J_{F,h^*})^{-1} \right\| \leq \left\| (I - J_{F,h^*})^{-1} \right\| \left\| J_{F,h^*} \right\|^{K+1}$$

We leave all proofs in appendix.

5. Experiments

In this section, we thoroughly study all RBP variants on diverse applications. Our implementation based on PyTorch is publicly available[1]. Note that our Neumann-RBP is very simple to implement using automatic differentiation and we provide a very short example program in the appendix.

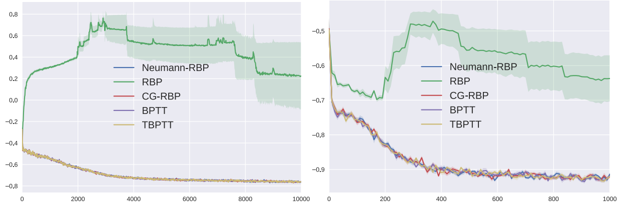


Figure 1. Left and right figures are training and validation curves of the same Hopfield network. y axis is the log scale L1 loss. x axis of (a) and (b) are training and validation step respectively. We do validation every 10 training steps.

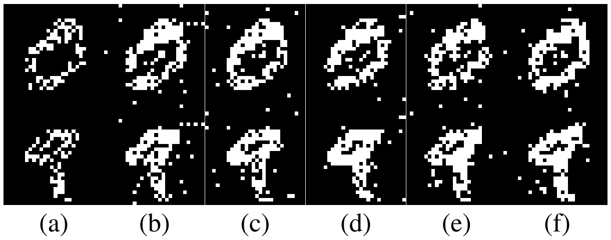


Figure 2. Visualization of associative memory. (a) Corrupted input image; (b)-(f) are retrieved images by BPTT, TBPTT, RBP, CG-RBP, Neumann-RBP respectively.

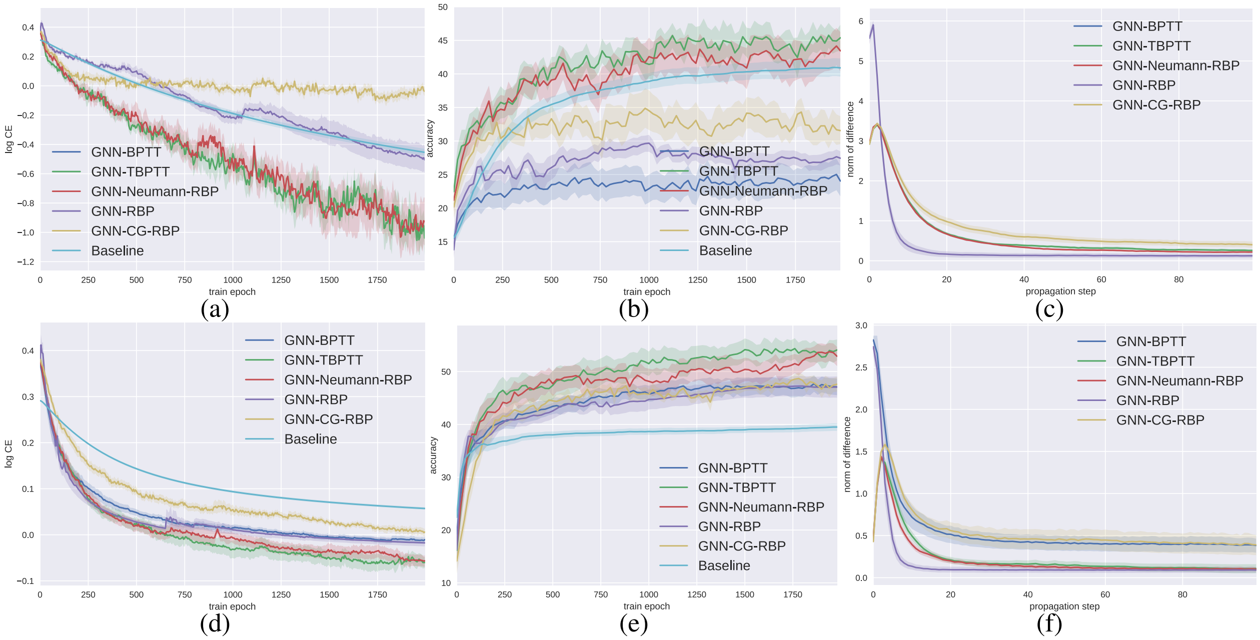
5.1. Associative Memory

The classical testbed for RBP is the associative memory (Hopfield, 1982). Several images or patterns are presented to the neural network which learns to store or memorize the images. After the learning process, the network is subsequently presented with a corrupted or noisy version of the original image. The task is then to retrieve the corresponding original image. We consider a simplified continuous Hopfield network as described in (Haykin, 1993). Specifically, the system of nonlinear first-order differential equations is,

$$\frac{d}{dt} h_i(t) = -\frac{h_i(t)}{a} + \sum_{j=1}^{N} w_{ij} \phi(b \cdot h_j(t)) + I_i, \qquad (18)$$

where subscript i denotes the index of the neuron. w_{ij} is the learnable weight between a pair of neurons. h_i is the hidden state of the i-th neuron. ϕ is a nonlinear activate function which is a sigmoid function in our experiments. a, b are positive constants and are set to 1 and 0.5. The set of neurons consists of three parts: observed, hidden and output neurons, of size 784, 1024 and 784 respectively. For observed neuron, the state h_i is clamped to observed pixel value I_i. For hidden and output neurons, the observed pixel value $I_i = 0$ and their states h_i are updated according to Eq. (18). During inference, the output neurons return $x_i = \phi(b \cdot h_i)$ and we further binarize it for visualization. An important property of continuous Hopfield networks is

[1] https://github.com/lrjconan/RBP



(a) (b) (c)

(d) (e) (f)

Figure 3. The first and second rows are the results on Cora and Pubmed respectively. (a) to (c), (d) to (f) are curves of training loss, validation accuracy and difference norm of the same GNN with different optimization methods.

that by updating the states according to Eq. (18) until convergence (which corresponds to the forward pass of RNNs), we are guaranteed to minimize the following (Lyapunov) energy function.

$$E = \sum_{i=1}^{N} \left(\frac{1}{a} \int_{0}^{x_i} \phi^{-1}(x) dx - I_i x_i \right) - \sum_{i=1}^{N} \sum_{j=1}^{N} \frac{w_{ij} x_i x_j}{2},$$

where we drop the dependency on time t for simplicity. Instead of adopting the Hebbian learning rule as in (Hopfield, 1982), we directly formulate the learning objective as minimizing $\sum_{i \in \mathcal{I}} \|x_i - I_i\|_1$ where $\mathcal{I}$ is the set of observed neurons. In our experiments, we train and test on 10 MNIST images. In training we feed clean data, and during testing we randomly corrupt 50% of the non-zero pixel values to zero. The number of updates for one inference pass is 50.

Fig. 1 shows the training and validation curves of continuous Hopfield network with different optimization methods. Here truncation steps for TBPTT, RBP, CG-RBP and Neumann-RBP are all set to 20. From the figure, we can see that CG-RBP and Neumann-RBP match BPTT under this setting which verifies that their gradients are accurate. Nevertheless, we can see that training curve of the original RBP blows up which validates its instability issue. The hidden state of Hopfield network becomes steady within 10 steps. However, we notice that if we set the truncation step to 10, original RBP exhibits behaviors which fails to converge. We also show some visualizations of retrieved images of the Hopfield network under different optimization methods in Fig. 2. More visual results are provided in the appendix.

5.2. Semi-supervised Document Classification

We investigate RBPs on semi-supervised document classification with citation networks. A node of a network represents a document associated with a bag-of-words feature. Nodes are connected based on the citation links. Given a portion of nodes labeled with subject categories, e.g., science, history, the task is to predict the categories for unlabeled nodes within the same network. We use two citation networks from (Yang et al., 2016), i.e., Cora, Pubmed, of which the statistics are summarized in the appendix. We adopt graph neural networks (GNNs) (Scarselli et al., 2009) model and employ the GRU as the update function similarly as (Li et al., 2016). We refer to (Li et al., 2016; Liao et al., 2018) for more details. We compare different optimization methods with the same GNN. We also add a logistic regression model as a baseline which is applied to every node independently. The labeled documents are randomly split into 1%, 49% and 50% for training, validation and testing. We run all experiments with 10 different random seeds and report the average results. The training, validation and difference norm curves of BPTT, TBPTT and all RBPs are shown in Fig. 3. We can see that the hidden states of GNNs with different optimization methods become steady during inference from Fig. 3 (c). As shown in Fig. 3 (a) and (b), Neumann-RBP is on par with TBPTT on both datasets. This matches our analysis in proposition 1 since the changes of successive hidden states of TBPTT and Neumann-RBP are almost zero as shown in Fig. 3 (c). Moreover, they outperform other variants and the baseline model. On the other hand, BPTT on both datasets encounter issues in learning which may be attributable to the accumulation of errors in

Test Acc.	Cora	Pubmed
Baseline	39.96 ± 3.4	40.41 ± 3.1
BPTT	24.48 ± 6.6	47.05 ± 3.1
TBPTT	46.55 ± 6.4	53.41 ± 6.7
RBP	29.25 ± 3.3	48.55 ± 3.4
CG-RBP	39.26 ± 6.5	49.12 ± 2.9
Neumann-RBP	$\mathbf{46.63} \pm 8.3$	$\mathbf{53.56} \pm 5.3$

Table 1. Test accuracy of different methods on citation networks.

Truncate Step	10	50	100
Run Time	×3.02	×2.87	×2.68
Memory	×4.35	×4.25	×4.11

Table 2. Run time and memory comparison. We show the ratio of BPTT's cost divided by Neumann-RBP's.

the many steps of unrolling. Note that CG-RBP sometimes performs significantly worse than Neumann-RBP, e.g., on Cora. This may be caused by the fact that the underlying linear system of CG-RBP is ill-conditioned in some applications as the condition number is squared in CGNE. The test accuracy of different methods are summarized in Table 1. It generally matches the behavior in the validation curves.

5.3. Hyperparameter Optimization

In our next experiment, we test the abilities of RBP to perform hyperparameter optimization. In this experiment, we view the optimization process as a RNN. When training a neural network, the model parameters, e.g., weights and bias, are regarded as the hidden states of the RNN. Hyperparameters such as learning rate and momentum are learnable parameters of this 'meta-learning' RNN. Here we focus on the gradient based hyperparameter optimization rather than the gradient-free one (Snoek et al., 2012). We adopt the same experiment setting as in (Maclaurin et al., 2015), using an initial learning rate of $\exp(-1)$ and momentum 0.5. The optimization is on a fully connected network with 4 layers, of sizes 784, 50, 50, and 50. For each layer, we associate one learning rate and one momentum with weight and bias respectively which results in 16 hyperparameters in total. We use tanh non-linearities and train on $10,000$ examples on MNIST. At each forward step of the RNN, i.e., at each optimization step, a different mini-batch of images is fed to the model. This is different from the previous setting where input data is fixed. However, since the mini-batches are assumed to be i.i.d., the sequential input data can be viewed as sampled from a stationary distribution. We can thus safely apply RBP as the steady state holds in expectation. In terms of implementation, we just need to average the meta gradient returned by RBPs or TBPTT across multiple mini-batches at the end of one meta step. We use Adam (Kingma & Ba, 2014) as the meta optimizer and set the learning rate to 0.05. The initialization of the fully connected network at each meta step is controlled to be the same.

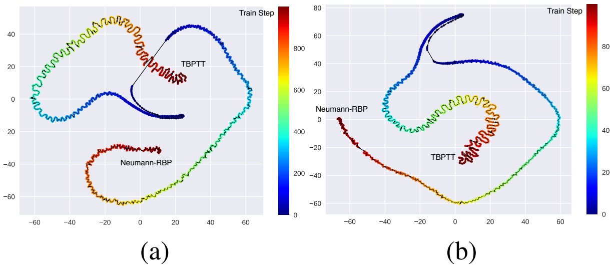


Figure 4. t-SNE visualization of trajectories of hidden states for TBPTT and Neumann-RBP on hyperparameter optimization. Different methods are annotated at the convergence point. (a) and (b) are snapshots of hidden states at meta step 20 and 40. Time is encoded as the color map for better illustration.

Fig. 5 shows the meta training losses under different training and truncation steps. For better understanding, one can consider the training step as the unrolling step of the RNN. Truncation step is the the number of steps that TBPTT and RBPs execute. From the figure, we can see that as the number of training steps increases (e.g., from (a) to (d)), the meta loss becomes smoother. This makes sense since more steps make the training per meta step closer to convergence. Another surprising phenomenon we found is the meta loss of TBPTT becomes worse when the training step increases. One possible explanation is that the initial meta training loss of small training steps (e.g., (a)) is still very high as you can see from the log y-axis whereas the one with large training step, e.g., (d) is much lower. The probability of using incorrect gradients to decrease the meta loss in case (a) is most likely higher than that of (d) since it is farther from convergence. On the other hand, our Neumann-RBP performs consistently better than the original RBP and TBPTT which empirically validates that Neumann-RBP provides better estimation of the gradient in this case. The potential reason why RBP performs poorly is that the stochasticity of mini-batches worsen the instability issue. Training losses under similar settings at the last meta step are also provided in Fig. 6. We can see that at the end of hyperparameter optimization, our Neumann-RBP generally matches the performance of BPTT and outperforms the other methods. Fig. 4 depicts the trajectories of hidden states in 2D space via t-SNE (Maaten & Hinton, 2008). From the figure, we can see that as the meta training goes on, TBPTT tends to oscillate whereas Neumann-RBP converges, which matches the finding in the train loss curves in Fig. 6.

We also compare the running time and memory cost of our unoptimized Neumann-RBP implementation with the standard BPTT, i.e., using autograd of PyTorch. With 1000 training steps, one meta step of BPTT cost 310.4s and 4061MB GPU memory in average. We take BPTT as the reference cost and report the ratio BPTT's cost divided by Neumann-

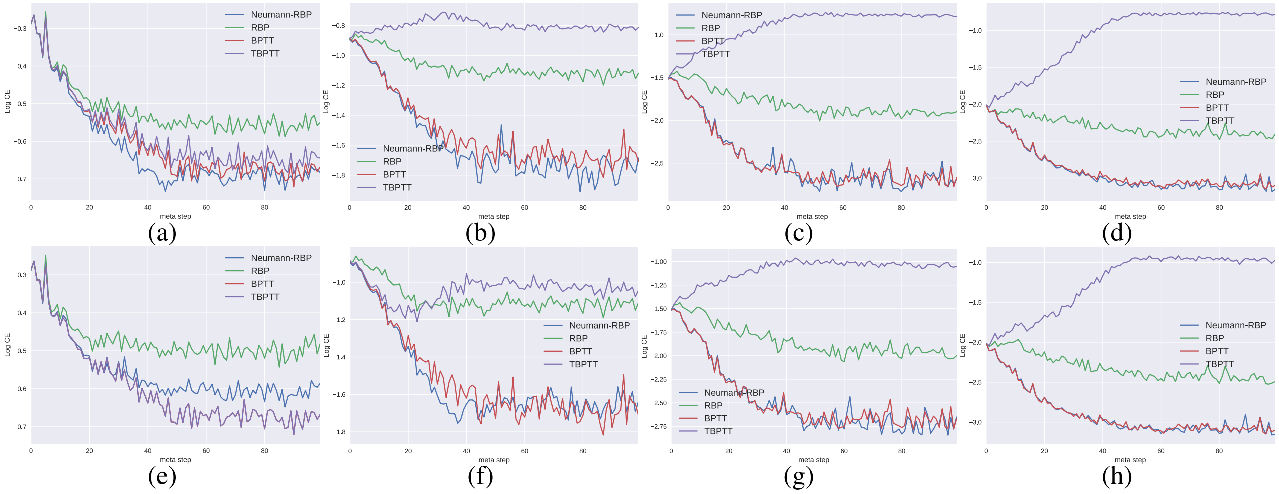


Figure 5. Meta training loss. Training and truncate steps per meta step are (a) (100, 50); (b) (500, 50); (c) (1000, 50); (d) (1500, 50); (e) (100, 100); (f) (500, 100); (g) (1500, 100); (h) (1500, 100).

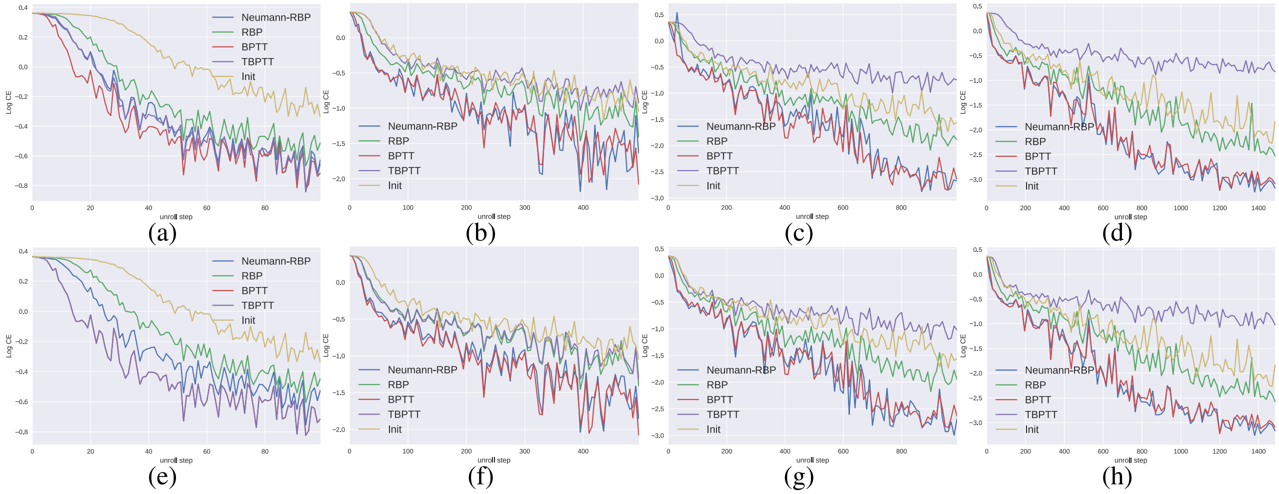


Figure 6. Training loss at last meta step. Training and truncate steps per meta step are (a) (100, 50); (b) (500, 50); (c) (1000, 50); (d) (1500, 50); (e) (100, 100); (f) (500, 100); (g) (1500, 100); (h) (1500, 100).

RBP's in Table 2. All results are reported as the average of 10 runs. Even without optimizing the code, the practical runtime and memory footprint advantages of Neumann-RBP over BPTT is still significant.

6. Conclusion

In this paper, we revisit the RBP algorithm and discuss its assumptions and how to satisfy them for deep learning. Moreover, we propose two variants of RBP based on conjugate gradient on normal equations and Neumann series. Connections between Neumann-RBP and TBPTT are established which sheds some light on analyzing the approximation quality of the gradient of TBPTT. Experimental results on diverse tasks demonstrate that Neumann-RBP is a stable and efficient alternative to original RBP and is promising for several practical problems. In the future, we would like to explore RBP on hyperparameter optimization with large scale deep neural networks.

Acknowledgements

We thank Barak Pearlmutter for the enlightening discussion and anonymous ICML reviewers for valuable comments. R.L. was supported by Connaught International Scholarships. R.L., E.F., L.Z., K.Y., X.P., R.U. and R.Z. were supported in part by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior/Interior Business Center (DoI/IBC) contract number D16PC00003. K.Y. and X.P. were supported in part by BRAIN Initiative grant NIH 5U01NS094368. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: the views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/IBC, or the U.S. Government.

References

Almeida, L. B. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *IEEE International Conference on Neural Networks*, pp. 609–618, 1987.

Amos, B. and Kolter, J. Z. Optnet: Differentiable optimization as a layer in neural networks. In *ICML*, 2017.

Chauvin, Y. and Rumelhart, D. E. *Backpropagation: theory, architectures, and applications*. Psychology Press, 1995.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *EMNLP*, 2014.

Domke, J. Generic methods for optimization-based modeling. In *AISTATS*, pp. 318–326, 2012.

Foo, C.-s., Do, C. B., and Ng, A. Y. Efficient multiple hyperparameter learning for log-linear models. In *NIPS*, pp. 377–384, 2008.

Golub, G. H. and Van Loan, C. F. *Matrix computations*, volume 3. JHU Press, 2012.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Haykin, S. *Neural Networks and Learning Machines*. Prentice Hall, 1993.

Hestenes, M. R. and Stiefel, E. *Methods of conjugate gradients for solving linear systems*, volume 49. NBS Washington, DC, 1952.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *PNAS*, 79(8): 2554–2558, 1982.

Hopfield, J. J. Neurons with graded response have collective computational properties like those of two-state neurons. *PNAS*, 81(10):3088–3092, 1984.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Lapedes, A. and Farber, R. A self-optimizing, nonsymmetrical neural net for content addressable memory and pattern recognition. *Physica D: Nonlinear Phenomena*, 22(1-3): 247–259, 1986.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. Gated graph sequence neural networks. *ICLR*, 2016.

Liao, R., Brockschmidt, M., Tarlow, D., Gaunt, A., Urtasun, R., and Zemel, R. Graph partition neural networks for semi-supervised classification. In *ICLR Workshop*, 2018.

Maaten, L. v. d. and Hinton, G. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov): 2579–2605, 2008.

Maclaurin, D., Duvenaud, D., and Adams, R. Gradient-based hyperparameter optimization through reversible learning. In *ICML*, pp. 2113–2122, 2015.

Ollivier, Y., Tallec, C., and Charpiat, G. Training recurrent networks online without backtracking. *arXiv preprint arXiv:1507.07680*, 2015.

Paige, C. C. and Saunders, M. A. Lsqr: An algorithm for sparse linear equations and sparse least squares. *ACM transactions on mathematical software*, 8(1):43–71, 1982.

Pearlmutter, B. A. Fast exact multiplication by the hessian. *Neural Computation*, 6(1):147–160, 1994.

Pineda, F. J. Generalization of back-propagation to recurrent neural networks. *Physical review letters*, 59(19):2229, 1987.

Rudin, W. *Principles of mathematical analysis*, volume 3. McGraw-hill New York, 1964.

Samuel, K. G. and Tappen, M. F. Learning optimized map estimates in continuously-valued mrf models. In *CVPR*, pp. 477–484. IEEE, 2009.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

Scellier, B. and Bengio, Y. Equivalence of equilibrium propagation and recurrent backpropagation. *arXiv preprint arXiv:1711.08416*, 2017a.

Scellier, B. and Bengio, Y. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in Computational Neuroscience*, 11:24, 2017b.

Schraudolph, N. N. Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14(7):1723–1738, 2002.

Snoek, J., Larochelle, H., and Adams, R. P. Practical bayesian optimization of machine learning algorithms. In *NIPS*, pp. 2951–2959, 2012.

Sutskever, I. Training recurrent neural networks. *PhD thesis, University of Toronto*, 2013.

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In *NIPS*, pp. 3104–3112, 2014.

Tallec, C. and Ollivier, Y. Unbiasing truncated backpropagation through time. *arXiv preprint arXiv:1705.08209*, 2017.

Vassil, S. D. and Diego, F. G. C. On the computation of neumann series. *arXiv preprint arXiv:1707.05846*, 2017.

Werbos, P. J. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

Westreich, D. Evaluating the matrix polynomial i+a+. . .+a n-1. *IEEE Transactions on Circuits and Systems*, 36(1): 162–164, Jan 1989.

Williams, R. J. and Peng, J. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, 2(4):490–501, 1990.

Yang, Z., Cohen, W. W., and Salakhutdinov, R. Revisiting semi-supervised learning with graph embeddings. In *ICML*, 2016.