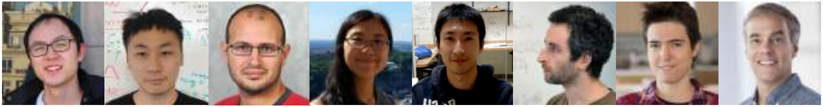
Reviving and Improving Recurrent Back Propagation

Renjie Liao[1,2,3], Yuwen Xiong[1], 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]

[1]University of Toronto, [2]Uber ATG Toronto, [3]Vector Institute, [4]Rice University, [5]Baylor College of Medicine, [6]Canadian Institute for Advanced Research

rjliao@cs.toronto.edu



July 12, 2018

Overview

1. Motivations & Backgrounds
2. Algorithms
3. Experiments

Motivations

Recurrent Back-Propagation (RBP), a.k.a., Almeida-Pineda algorithm, is independently proposed by following papers:

- *Almeida, L.B., 1987. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. IEEE International Conference on Neural Networks, 609-618.*
- *Pineda, F.J., 1987. Generalization of back-propagation to recurrent neural networks. Physical Review Letters, 59(19), p.2229.*

Motivations

Recurrent Back-Propagation (RBP), a.k.a., Almeida-Pineda algorithm, is independently proposed by following papers:

- *Almeida, L.B., 1987. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. IEEE International Conference on Neural Networks, 609-618.*
- *Pineda, F.J., 1987. Generalization of back-propagation to recurrent neural networks. Physical Review Letters, 59(19), p.2229.*

Property

- It exploits implicit function theorem to compute the gradient without back-propagating through time

Motivations

Recurrent Back-Propagation (RBP), a.k.a., Almeida-Pineda algorithm, is independently proposed by following papers:

- *Almeida, L.B., 1987. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. IEEE International Conference on Neural Networks, 609-618.*
- *Pineda, F.J., 1987. Generalization of back-propagation to recurrent neural networks. Physical Review Letters, 59(19), p.2229.*

Property

- It exploits implicit function theorem to compute the gradient without back-propagating through time
- It is efficient with memory and computation

Motivations

Recurrent Back-Propagation (RBP), a.k.a., Almeida-Pineda algorithm, is independently proposed by following papers:

- *Almeida, L.B., 1987. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. IEEE International Conference on Neural Networks, 609-618.*
- *Pineda, F.J., 1987. Generalization of back-propagation to recurrent neural networks. Physical Review Letters, 59(19), p.2229.*

Property

- It exploits implicit function theorem to compute the gradient without back-propagating through time
- It is efficient with memory and computation
- It is successful for limited cases, e.g., Hopfield Networks

Motivations

Recurrent Back-Propagation (RBP), a.k.a., Almeida-Pineda algorithm, is independently proposed by following papers:

- *Almeida, L.B., 1987. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. IEEE International Conference on Neural Networks, 609-618.*
- *Pineda, F.J., 1987. Generalization of back-propagation to recurrent neural networks. Physical Review Letters, 59(19), p.2229.*

Property

- It exploits implicit function theorem to compute the gradient without back-propagating through time
- It is efficient with memory and computation
- It is successful for limited cases, e.g., Hopfield Networks

Similar technique (implicit differentiation) was rediscovered later in PGMs!

Background

Convergent Recurrent Neural Networks

- Dynamics:

$$h^{t+1} = F(x, w, h^t)$$

  where x, w and h^t are data, weight and hidden state.
- Steady/Stationary/Equilibrium State:

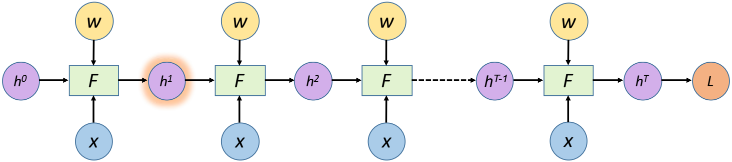
$$h^* = F(x, w, h^*)$$

Special Instances

- Jacobi method
- Gauss–Seidel method
- Fixed-point iteration method

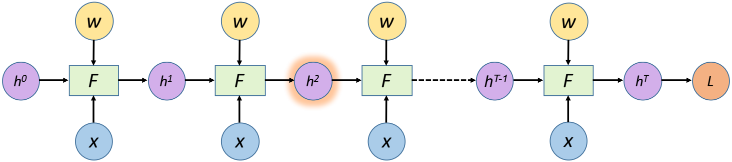
Back-Propagation Through Time

Forward Pass:



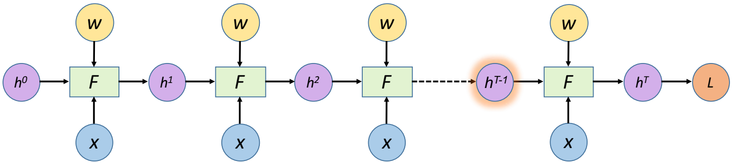
Back-Propagation Through Time

Forward Pass:



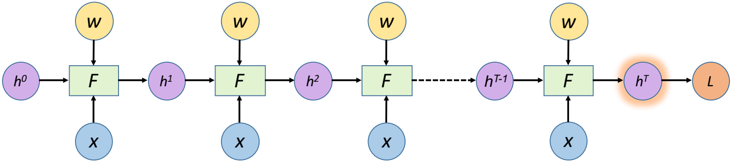
Back-Propagation Through Time

Forward Pass:



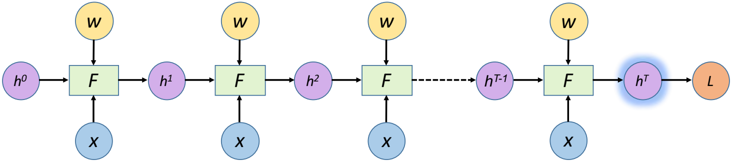
Back-Propagation Through Time

Forward Pass:



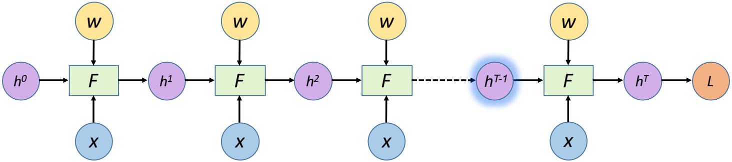
Back-Propagation Through Time

Backward Pass:

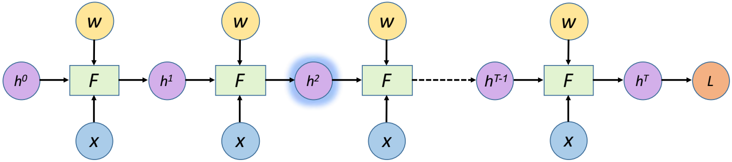


Back-Propagation Through Time

Backward Pass:

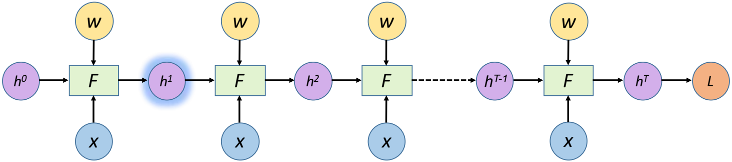


Backward Pass:



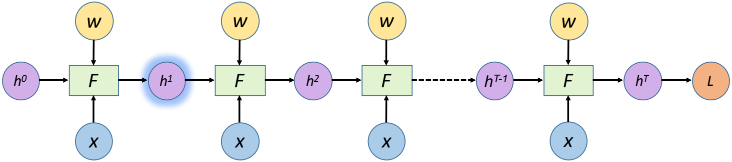
Back-Propagation Through Time

Backward Pass:



Back-Propagation Through Time

Backward Pass:



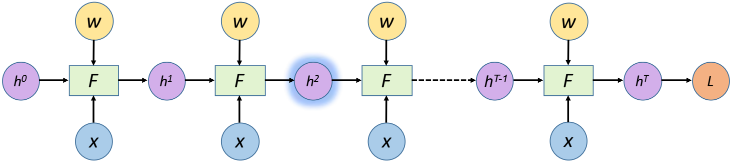
BPTT

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial h^T}\left(\frac{\partial h^T}{\partial w} + \frac{\partial h^T}{\partial h^{T-1}}\frac{\partial h^{T-1}}{\partial w} + \dots\right)$$

$$= \frac{\partial L}{\partial h^T}\sum_{k=1}^{T}\left(\prod_{i=T-k+1}^{T-1} J_{F,h^i}\right)\frac{\partial F(x, w, h^{T-k})}{\partial w}$$

Truncated Back-Propagation Through Time

Backward Pass:



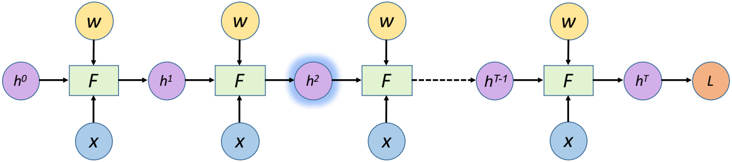
TBPTT

Truncated at K-th step:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial h^T}\left(\frac{\partial h^T}{\partial w} + \frac{\partial h^T}{\partial h^{T-1}}\frac{\partial h^{T-1}}{\partial w} + \dots\right)$$

$$= \frac{\partial L}{\partial h^T}\sum_{k=1}^{K}\left(\prod_{i=T-k+1}^{T-1} J_{F,h^i}\right)\frac{\partial F(x, w, h^{T-k})}{\partial w}$$

Recurrent Back-Propagation



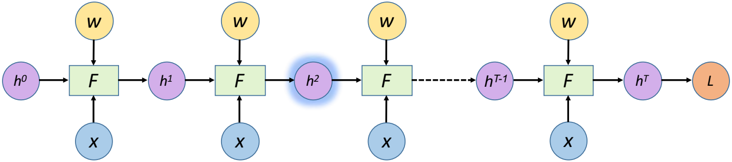
Implicit Function Theorem

Let $\Psi(x, w, h) = h - F(x, w, h)$, at steady state h^*, we have $\Psi(x, w, h^*) = 0$

Implicit Function Theorem is applicable if two conditions hold:

- Ψ is continuously differentiable
- $I - J_{F,h^*}$ is invertible

Recurrent Back-Propagation



Implicit Function Theorem

Let $\Psi(x, w, h) = h - F(x, w, h)$, at steady state h^*, we have $\Psi(x, w, h^*) = 0$

Implicit Function Theorem is applicable if two conditions hold:

- Ψ is continuously differentiable (LSTM, GRU)
- $I - J_{F,h^*}$ is invertible

Recurrent Back-Propagation

Contraction Mapping on Banach Space

F is a contraction mapping on Banach (completed and normed) space B, iff there exists some $0 \leq \mu < 1$ such that $\forall x, y \in B$

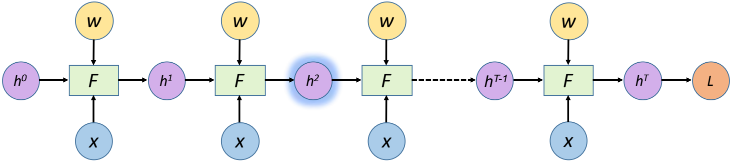
$$\|F(x) - F(y)\| \leq \mu \|x - y\|$$

One Sufficient Condition

- Contraction Mapping $\Longrightarrow \sigma_{\max}(J_{F,h^*}) \leq \mu < 1$
- We then have,

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

- $I - J_{F,h^*}$ is invertible



Implicit Function Theorem

$$
\begin{aligned}
\frac{\partial \Psi(x, w, h^*)}{\partial w} &= \frac{\partial h^*}{\partial w} - \frac{\nabla F(x, w, h^*)}{\nabla w} \\
&= (I - J_{F,h^*}) \frac{\partial h^*}{\partial w} - \frac{\partial F(x, w, h^*)}{\partial w} = \mathbf{0}
\end{aligned} \tag{1}
$$

The desired gradient is:

$$
\frac{\partial L}{\partial w} = \frac{\partial L}{\partial h^*} (I - J_{F,h^*})^{-1} \frac{\partial F(x, w, h^*)}{\partial w}
$$

Recurrent Back-Propagation

Derivation of Original RBP

- Gradient:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial h^*} \left(I - J_{F,h^*}\right)^{-1} \frac{\partial F(x, w, h^*)}{\partial w}$$

- Introduce $z^\top = \frac{\partial L}{\partial h^*} \left(I - J_{F,h^*}\right)^{-1}$ which defines an adjoint linear system,

$$\left(I - J_{F,h^*}^\top\right) z = \left(\frac{\partial L}{\partial h^*}\right)^\top \tag{2}$$

- Original RBP uses fixed-point iteration method,

$$z = J_{F,h^*}^\top z + \left(\frac{\partial L}{\partial h^*}\right)^\top = f(z) \tag{3}$$

Recurrent Back-Propagation

Algorithm 1: : Original RBP

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

Pros & Cons

- Memory cost scales constantly w.r.t. # time steps whereas BPTT scales linearly
- It is often faster than BPTT for many-step RNNs
- It may converge slowly and sometimes numerically unstable

Conjugate Gradient based RBP

Observations

- Our core problem is $\left(I - J_{F,h^*}^\top\right) z = \left(\frac{\partial L}{\partial h^*}\right)^\top$, simplified as $Az = b$
- CG is better than fixed-point iteration if A is PSD
- A is often asymmetric for RNNs

Conjugate Gradient on the Normal Equations (CGNE)

- Multiple $(I - J_{F,h^*})$ on both sides,

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

- Apply CG to solve z

Caveat: the condition number of the new system is squared!

Neumann Series based RBP

Neumann Series

- It's a mathematical series of the form $\sum_{t=0}^{\infty} A^t$ where A is an operator, a.k.a., matrix geometric series in matrix terminology
- A convergent Neumann series has the property:

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

Neumann-RBP

- Recall auxiliary variable z in RBP:

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

- Replace A with $J_{F,h^*}^{\top}$ and truncate it at K-th power

Neumann-RBP

Algorithm 2: : Neumann-RBP

1: **Initialization:** $v_0 = g_0 = \left(\frac{\partial L}{\partial h^*}\right)^\top$
2: **for** $t = 1, 2, \dots, K$ **do**
3: $\quad v_t = J_{F,h^*}^\top v_{t-1}$
4: $\quad g_t = g_{t-1} + v_t$
5: **end for**
6: Return $\frac{\partial L}{\partial w} = (g_K)^\top \frac{\partial F(x,w,h^*)}{\partial w}$

We show Neumann-RBP is related to BPTT and TBPTT:

Proposition 1

Assume that we have a convergent RNN which satisfies the implicit function theorem conditions. If the Neumann series $\sum_{t=0}^{\infty} J_{F,h^*}^t$ converges, then the full Neumann-RBP is equivalent to BPTT.

Proposition 2

For the above RNN, let us denote its convergent sequence of hidden states as $h^0, h^1, \dots, h^T$ where $h^* = h^T$ is the steady state. If we further assume that there exists some step K where $0 < K \leq T$ such that $h^* = h^T = h^{T-1} = \cdots = h^{T-K}$, then K-step Neumann-RBP is equivalent to K-step TBPTT.

Neumann-RBP

Proposition 3

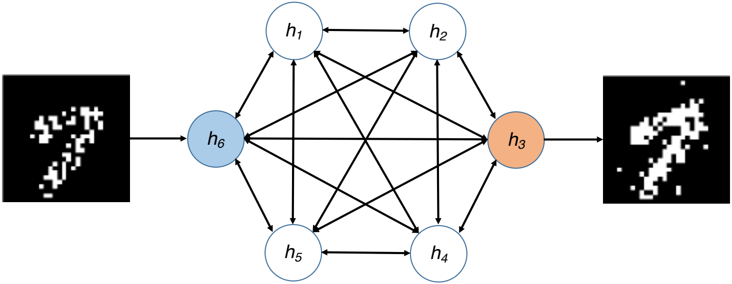
If the Neumann series $\sum_{t=0}^{\infty} J_{F,h^*}^t$ converges, then the error between K-step and full Neumann series is as follows,

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

Pros & Cons

- CG-RBP requires fewer # updates but may be slower in run time and is sometimes problematic due to the squared condition number
- Neumann-RBP is stable and has same time & memory complexity

Continuous Hopfield Networks



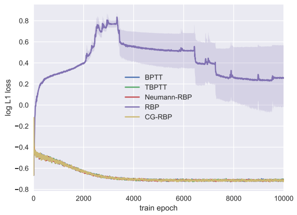
Model

- Inference:

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

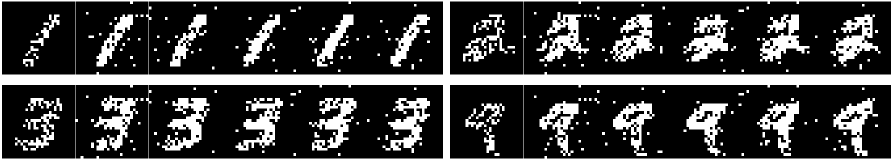
- Learning: $\min_w \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} \|\phi(b \cdot h_i) - I_i\|_1$

Continuous Hopfield Networks



Truncation Step	10	20	30
TBPTT	100%	100%	100%
RBP	1%	4%	99%
CG-RBP	100%	100%	100%
Neumann-RBP	100%	100%	100%

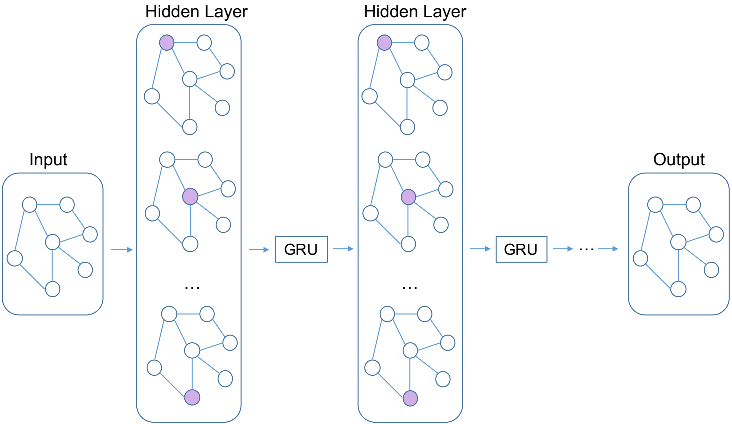
Table: Success (final loss $\leq$ 50% initial loss) rate.



(a) (b) (c) (d) (e) (f) (a) (b) (c) (d) (e) (f)

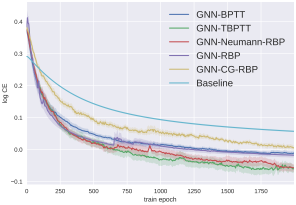
Figure: Visualization of associative memory. (a) Corrupted input image; (b)-(f) are retrieved images by BPTT, TBPTT, RBP, CG-RBP, Neumann-RBP respectively.

Gated Graph Neural Networks

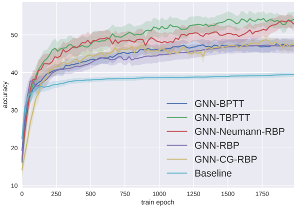


Gated Graph Neural Networks

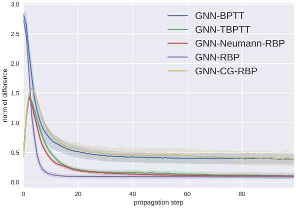
- Semi-supervised document classification in citation networks



(a)



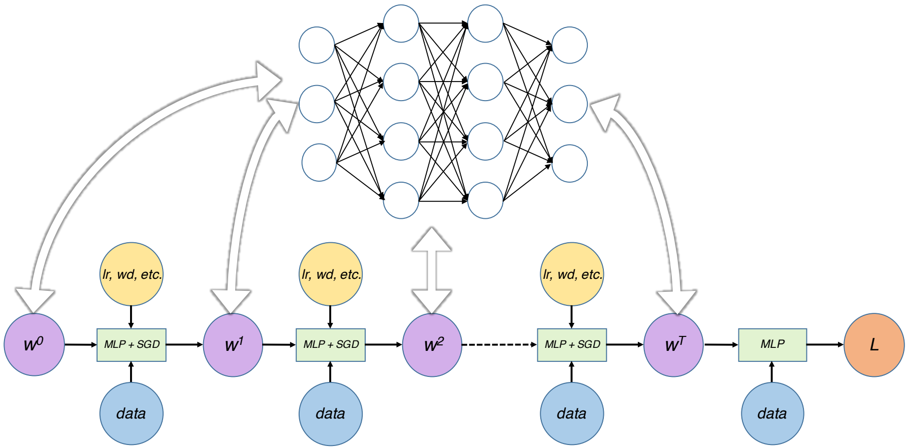
(b)



(c)

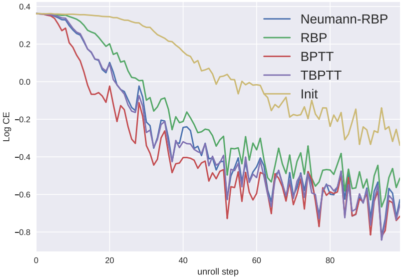
Figure: (a) Training loss; (b) Validation accuracy. (c) Difference between consecutive hidden states.

Hyperparameter Optimization

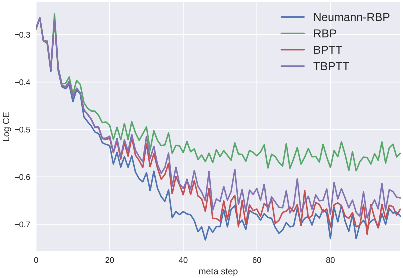


Hyperparameter Optimization

Optimize 100 steps, truncate 50 steps:



(a)

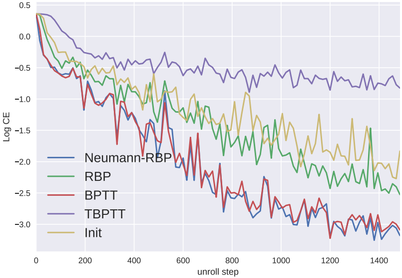


(b)

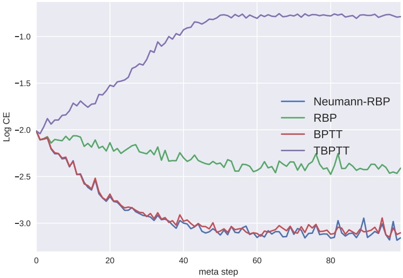
Figure: (a) Training loss at last meta step; (b) Meta training loss.

Hyperparameter Optimization

Optimize 1500 steps, truncate 50 steps:



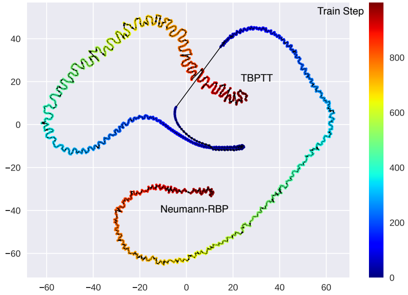
(a)



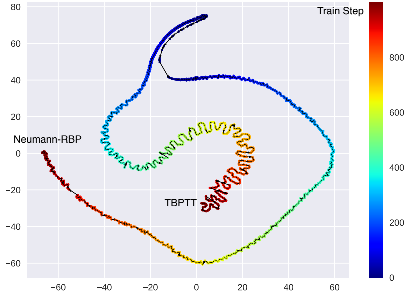
(b)

Figure: (a) Training loss at last meta step; (b) Meta training loss.

Hyperparameter Optimization



(a) Meta step 20



(b) Meta step 40.

Truncation Step	10	50	100		10	50	100
Run Time	$\times 3.02$	$\times 2.87$	$\times 2.68$	Memory	$\times 4.35$	$\times 4.25$	$\times 4.11$

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

Source Code

Our code is released at https://github.com/lrjconan/RBP

```python
def neumann_rbp(weight, hidden_state, loss, rbp_step)
  # get the gradient of last hidden state
  grad_h = autograd.grad(loss, hidden_state[-1], retain_graph=True)

  # set v, g to grad_h
  neumann_v = grad_h.clone()
  neumann_g = grad_h.clone()

  for i in range(rbp_step):
    # set last hidden_state's gradient to neumann_v[prev]
    # and get the gradient of last second hidden state
    neumann_v = autograd.grad(
                    hidden_state[-1], hidden_state[-2],
                    grad_outputs=neumann_v,
                    retain_graph=True)

    neumann_g += neumann_v

  # set last hidden_state's gradient to neumann_g
  # and return the gradient of weight
  return autograd.grad(hidden_state[-1], weight, grad_outputs=neumann_g)
```

Take Home Messages

- RBP is very efficient for learning convergent RNNs
- Neumann-RBP is stable, often faster than BPTT and takes constant memory
- Neumann-RBP is simple to implement

Welcome to our poster #178 tonight!

Thank You