Learn as you go: Training recurrent networks online without backtracking

Yann Ollivier

CNRS & Paris-Saclay University, France

Joint work with Guillaume Charpiat and Corentin Tallec

Alan Turing Institute Workshop on Deep Learning
Edinburgh, November 23–25, 2015

Algorithms to train dynamical systems

$p = \#\text{params}$

Algorithm	Cost per time step	Store past states	Store past data
Kalman filter	$O(p^2)$	No	No

Algorithms to train dynamical systems

$p = \#\text{params}$

Algorithm	Cost per time step	Store past states	Store past data
Kalman filter	$O(p^2)$	No	No
RTRL	$O(p^2)$	No	No

Algorithms to train dynamical systems

$p = \#\text{params}$

Algorithm	Cost per time step	Store past states	Store past data
Kalman filter	$O(p^2)$	No	No
RTRL	$O(p^2)$	No	No
Backprop through time (T-truncated)	$O(p)$ biased	$O(\sqrt{T}p)$	$O(T)$

Algorithms to train dynamical systems

$p = \#\text{params}$

Algorithm	Cost per time step	Store past states	Store past data
Kalman filter	$O(p^2)$	No	No
RTRL	$O(p^2)$	No	No
Backprop through time (T-truncated)	$O(p)$ biased	$O(\sqrt{T}p)$	$O(T)$
?	$O(p)$	No	No

Recurrent networks as dynamical systems

Problem: how to train a dynamical system defined by

$$h(t+1) = F(h(t), x(t), \theta)$$

- $h(t)$: internal state of the system at time t
- $x(t)$: external input signal
- F: transition function, fixed
- θ: parameter to be trained

Recurrent networks as dynamical systems

Problem: how to train a dynamical system defined by

$$h(t+1) = F(h(t), x(t), \theta)$$

- $h(t)$: internal state of the system at time t
- $x(t)$: external input signal
- F: transition function, fixed
- θ: parameter to be trained

Goal: minimize some loss function $\ell_t(h(t))$ along the trajectories of the system.

Recurrent networks as dynamical systems

Problem: how to train a dynamical system defined by

$$h(t+1) = F(h(t), x(t), \theta)$$

- $h(t)$: internal state of the system at time t
- $x(t)$: external input signal
- F: transition function, fixed
- θ: parameter to be trained

Goal: minimize some loss function $\ell_t(h(t))$ along the trajectories of the system.

Example: recurrent network with activation function σ,

$$h_j(t+1) = \sum_i w_{ij}\,\sigma(h_i(t)) + \sum_k r_{kj} x_k(t)$$

with $\theta = (w, r)$.

Simple strategy: online gradient descent over the loss at time t,

$$\theta \leftarrow \theta - \eta \, \frac{\partial \ell_t(h(t))}{\partial \theta}$$

with learning rate η.

Simple strategy: online gradient descent over the loss at time t,

$$\theta \leftarrow \theta - \eta \, \frac{\partial \ell_t(h(t))}{\partial \theta}$$

with learning rate η.

Problem: how to compute the derivative $\dfrac{\partial \ell_t}{\partial \theta}$? Current loss depends on θ via whole past trajectory.

Simple strategy: online gradient descent over the loss at time t,

$$\theta \leftarrow \theta - \eta \frac{\partial \ell_t(h(t))}{\partial \theta}$$

with learning rate η.

Problem: how to compute the derivative $\frac{\partial \ell_t}{\partial \theta}$? Current loss depends on θ via whole past trajectory.

(More complex algorithms like the Kalman filter also rely on $\frac{\partial \ell_t}{\partial \theta}$.)

Simple strategy: online gradient descent over the loss at time t,

$$\theta \leftarrow \theta - \eta \, \frac{\partial \ell_t(h(t))}{\partial \theta}$$

with learning rate η.

Problem: how to compute the derivative $\frac{\partial \ell_t}{\partial \theta}$? Current loss depends on θ via whole past trajectory.

(More complex algorithms like the Kalman filter also rely on $\frac{\partial \ell_t}{\partial \theta}$.)

Standard approach to compute $\frac{\partial \ell_t}{\partial \theta}$: backpropagation through time (BPTT).
Problem: goes back in time...

Why backpropagation through time?

For instance, let us compute how the current loss ℓ_t depends on the starting point $h(0)$:

$$\frac{\partial \ell_t}{\partial h(0)} =$$

Why backpropagation through time?

For instance, let us compute how the current loss ℓ_t depends on the starting point $h(0)$:

$$\frac{\partial \ell_t}{\partial h(0)} = \frac{\partial \ell_t}{\partial h(t)} \times \frac{\partial h(t)}{\partial h(t-1)} \times \cdots \times \frac{\partial h(1)}{\partial h(t-0)}$$

Why backpropagation through time?

For instance, let us compute how the current loss ℓ_t depends on the starting point $h(0)$:

$$\frac{\partial \ell_t}{\partial h(0)} = \frac{\partial \ell_t}{\partial h(t)} \times \frac{\partial h(t)}{\partial h(t-1)} \times \cdots \times \frac{\partial h(1)}{\partial h(t-0)}$$

$$= \text{vector} \times \text{sparse matrix} \times \cdots \times \text{sparse matrix}$$

Why backpropagation through time?

For instance, let us compute how the current loss ℓ_t depends on the starting point $h(0)$:

$$\frac{\partial \ell_t}{\partial h(0)} = \frac{\partial \ell_t}{\partial h(t)} \times \frac{\partial h(t)}{\partial h(t-1)} \times \cdots \times \frac{\partial h(1)}{\partial h(t-0)}$$

$$= \text{vector} \times \text{sparse matrix} \times \cdots \times \text{sparse matrix}$$

Left-to-right: OK, only vector×sparse matrix multiplications. That's BPTT.

Right-to-left: matrix-matrix multiplications, must store a matrix, and any sparsity is lost.

Why backpropagation through time?

For instance, let us compute how the current loss ℓ_t depends on the starting point $h(0)$:

$$\frac{\partial \ell_t}{\partial h(0)} = \frac{\partial \ell_t}{\partial h(t)} \times \frac{\partial h(t)}{\partial h(t-1)} \times \cdots \times \frac{\partial h(1)}{\partial h(t-0)}$$

$$= \text{vector} \times \text{sparse matrix} \times \cdots \times \text{sparse matrix}$$

Left-to-right: OK, only vector×sparse matrix multiplications. That's BPTT.

Right-to-left: matrix-matrix multiplications, must store a matrix, and any sparsity is lost.
$\Longrightarrow$ only for small networks. Known as real-time recurrent learning (RTRL).

Why backpropagation through time?

For instance, let us compute how the current loss ℓ_t depends on the starting point $h(0)$:

$$\frac{\partial \ell_t}{\partial h(0)} = \frac{\partial \ell_t}{\partial h(t)} \times \frac{\partial h(t)}{\partial h(t-1)} \times \cdots \times \frac{\partial h(1)}{\partial h(t-0)}$$

$$= \text{vector} \times \text{sparse matrix} \times \cdots \times \text{sparse matrix}$$

Left-to-right: OK, only vector×sparse matrix multiplications. That's BPTT.

Right-to-left: matrix-matrix multiplications, must store a matrix, and any sparsity is lost.
$\Longrightarrow$ only for small networks. Known as real-time recurrent learning (RTRL). Same problem with Kalman filters.

Why backpropagation through time?

For instance, let us compute how the current loss ℓ_t depends on the starting point $h(0)$:

$$\frac{\partial \ell_t}{\partial h(0)} = \frac{\partial \ell_t}{\partial h(t)} \times \frac{\partial h(t)}{\partial h(t-1)} \times \cdots \times \frac{\partial h(1)}{\partial h(t-0)}$$

$$= \text{vector} \times \text{sparse matrix} \times \cdots \times \text{sparse matrix}$$

Left-to-right: OK, only vector×sparse matrix multiplications. That's BPTT.

Right-to-left: matrix-matrix multiplications, must store a matrix, and any sparsity is lost.
$\Longrightarrow$ only for small networks. Known as real-time recurrent learning (RTRL). Same problem with Kalman filters.

Same forward-backward structure in many problems: hidden Markov models (EM), reinforcement learning and optimal control (Bellman equations)...

If you cannot travel back in time...

Algorithms that go forward in time must maintain the gradient of the current state with respect to the parameters:

$$G_t := \frac{\partial h(t)}{\partial \theta}$$

and then compute the gradient of the loss via the chain rule

$$\frac{\partial \ell_t}{\partial \theta} = \frac{\partial \ell_t}{\partial h(t)} \frac{\partial h(t)}{\partial \theta}$$

If you cannot travel back in time...

Algorithms that go forward in time must maintain the gradient of the current state with respect to the parameters:

$$G_t := \frac{\partial h(t)}{\partial \theta}$$

and then compute the gradient of the loss via the chain rule

$$\frac{\partial \ell_t}{\partial \theta} = \frac{\partial \ell_t}{\partial h(t)} \frac{\partial h(t)}{\partial \theta}$$

G_t is a full object of size dim(state)$\times$dim(param).

If you cannot travel back in time...

Algorithms that go forward in time must maintain the gradient of the current state with respect to the parameters:

$$G_t := \frac{\partial h(t)}{\partial \theta}$$

and then compute the gradient of the loss via the chain rule

$$\frac{\partial \ell_t}{\partial \theta} = \frac{\partial \ell_t}{\partial h(t)} \frac{\partial h(t)}{\partial \theta}$$

G_t is a full object of size dim(state)$\times$dim(param).

Sometimes, cannot even store G_t.

The NoBackTrack strategy

The NoBackTrack strategy

- At each time, maintain a search direction $\bar{w}_t$ in parameter space,

The NoBackTrack strategy

- At each time, maintain a search direction $\bar{w}_t$ in parameter space, together with an estimate $\bar{v}_t$ of the effect of $\bar{w}_t$ on the current state $h(t)$.

The NoBackTrack strategy

- At each time, maintain a search direction $\bar{w}_t$ in parameter space, together with an estimate $\bar{v}_t$ of the effect of $\bar{w}_t$ on the current state $h(t)$.

- Have the search direction $\bar{w}_t$ evolve stochastically, but not fully at random, in a way driven by how the transition function F depends on the parameter θ.

The NoBackTrack strategy

- At each time, maintain a search direction $\bar{w}_t$ in parameter space, together with an estimate $\bar{v}_t$ of the effect of $\bar{w}_t$ on the current state $h(t)$.
- Have the search direction $\bar{w}_t$ evolve stochastically, but not fully at random, in a way driven by how the transition function F depends on the parameter θ.
- Can be arranged so that, at each time, $\bar{v}_t \bar{w}_t^\top$ is an unbiased estimate of $\dfrac{\partial h(t)}{\partial \theta}$:

$$\mathbb{E}\bar{v}_t \bar{w}_t^\top = G_t$$

The NoBackTrack strategy

- At each time, maintain a search direction $\bar{w}_t$ in parameter space, together with an estimate $\bar{v}_t$ of the effect of $\bar{w}_t$ on the current state $h(t)$.
- Have the search direction $\bar{w}_t$ evolve stochastically, but not fully at random, in a way driven by how the transition function F depends on the parameter θ.
- Can be arranged so that, at each time, $\bar{v}_t \bar{w}_t^\top$ is an unbiased estimate of $\frac{\partial h(t)}{\partial \theta}$:

$$\mathbb{E} \bar{v}_t \bar{w}_t^\top = G_t$$

- Unbiased estimate of $G_t \implies$ unbiased estimate of the gradient of the loss function ℓ_t wrt the parameter

The NoBackTrack strategy

- At each time, maintain a search direction $\bar{w}_t$ in parameter space, together with an estimate $\bar{v}_t$ of the effect of $\bar{w}_t$ on the current state $h(t)$.

- Have the search direction $\bar{w}_t$ evolve stochastically, but not fully at random, in a way driven by how the transition function F depends on the parameter θ.

- Can be arranged so that, at each time, $\bar{v}_t \bar{w}_t^\top$ is an unbiased estimate of $\dfrac{\partial h(t)}{\partial \theta}$:

$$\mathbb{E} \bar{v}_t \bar{w}_t^\top = G_t$$

- Unbiased estimate of $G_t \Longrightarrow$ unbiased estimate of the gradient of the loss function ℓ_t wrt the parameter

- The estimates are noisy but unbiased $\Longrightarrow$ over time the parameter evolves in the correct direction.

To understand how to approximate G_t, let us look at its evolution. The evolution equation is

$$h(t+1) = F(h(t), x(t), \theta)$$

To understand how to approximate G_t, let us look at its evolution. The evolution equation is

$$h(t+1) = F(h(t), x(t), \theta) \qquad\qquad G_t := \frac{\partial h(t)}{\partial \theta}$$

To understand how to approximate G_t, let us look at its evolution. The evolution equation is

$$h(t+1) = F(h(t), x(t), \theta) \qquad\qquad G_t := \frac{\partial h(t)}{\partial \theta}$$

Taking the derivative of the evolution equation wrt θ we get

$$G_{t+1} = \frac{\partial F(h(t), x(t), \theta)}{\partial \theta} +$$

To understand how to approximate G_t, let us look at its evolution. The evolution equation is

$$h(t+1) = F(h(t), x(t), \theta) \qquad\qquad G_t := \frac{\partial h(t)}{\partial \theta}$$

Taking the derivative of the evolution equation wrt θ we get

$$G_{t+1} = \frac{\partial F(h(t), x(t), \theta)}{\partial \theta} + \frac{\partial F(h(t), x(t), \theta)}{\partial h(t)} \cdot G_t$$

To understand how to approximate G_t, let us look at its evolution. The evolution equation is

$$h(t+1) = F(h(t), x(t), \theta) \qquad\qquad G_t := \frac{\partial h(t)}{\partial \theta}$$

Taking the derivative of the evolution equation wrt θ we get

$$G_{t+1} = \frac{\partial F(h(t), x(t), \theta)}{\partial \theta} + \frac{\partial F(h(t), x(t), \theta)}{\partial h(t)} \cdot G_t$$

This equation is affine.

To understand how to approximate G_t, let us look at its evolution. The evolution equation is

$$h(t+1) = F(h(t), x(t), \theta) \qquad\qquad G_t := \frac{\partial h(t)}{\partial \theta}$$

Taking the derivative of the evolution equation wrt θ we get

$$G_{t+1} = \frac{\partial F(h(t), x(t), \theta)}{\partial \theta} + \frac{\partial F(h(t), x(t), \theta)}{\partial h(t)} \cdot G_t$$

This equation is affine.
$\Longrightarrow$ If $\tilde{G}_t$ is an unbiased approximation of G_t, then

To understand how to approximate G_t, let us look at its evolution. The evolution equation is

$$h(t+1) = F(h(t), x(t), \theta) \qquad\qquad G_t := \frac{\partial h(t)}{\partial \theta}$$

Taking the derivative of the evolution equation wrt θ we get

$$G_{t+1} = \frac{\partial F(h(t), x(t), \theta)}{\partial \theta} + \frac{\partial F(h(t), x(t), \theta)}{\partial h(t)} \cdot G_t$$

This equation is affine.
$\Longrightarrow$ If $\tilde{G}_t$ is an unbiased approximation of G_t, then

$$\frac{\partial F}{\partial \theta} + \frac{\partial F}{\partial h} \cdot \tilde{G}_t$$

is an unbiased approximation of G_{t+1}.

To understand how to approximate G_t, let us look at its evolution. The evolution equation is

$$h(t+1) = F(h(t), x(t), \theta) \qquad\qquad G_t := \frac{\partial h(t)}{\partial \theta}$$

Taking the derivative of the evolution equation wrt θ we get

$$G_{t+1} = \frac{\partial F(h(t), x(t), \theta)}{\partial \theta} + \frac{\partial F(h(t), x(t), \theta)}{\partial h(t)} \cdot G_t$$

This equation is affine.
$\Longrightarrow$ If $\tilde{G}_t$ is an unbiased approximation of G_t, then

$$\frac{\partial F}{\partial \theta} + \frac{\partial F}{\partial h} \cdot \tilde{G}_t$$

is an unbiased approximation of G_{t+1}. Use with $\tilde{G}_t = \bar{v}\bar{w}^\top$.

To understand how to approximate G_t, let us look at its evolution. The evolution equation is

$$h(t+1) = F(h(t), x(t), \theta) \qquad\qquad G_t := \frac{\partial h(t)}{\partial \theta}$$

Taking the derivative of the evolution equation wrt θ we get

$$G_{t+1} = \frac{\partial F(h(t), x(t), \theta)}{\partial \theta} + \frac{\partial F(h(t), x(t), \theta)}{\partial h(t)} \cdot G_t$$

This equation is affine.
$\Longrightarrow$ If $\tilde{G}_t$ is an unbiased approximation of G_t, then

$$\frac{\partial F}{\partial \theta} + \frac{\partial F}{\partial h} \cdot \tilde{G}_t$$

is an unbiased approximation of G_{t+1}. Use with $\tilde{G}_t = \bar{v}\bar{w}^\top$.

Problem: Even if $\tilde{G}_t = \bar{v}\bar{w}^\top$ is rank-one, $\tilde{G}_{t+1}$ is full-rank again.

The rank-one trick

Proposition

Let A be a matrix and decompose A as a sum of rank-one terms,

$$A = \sum_i v_i w_i^\top$$

The rank-one trick

Proposition

Let A be a matrix and decompose A as a sum of rank-one terms,

$$A = \sum_i v_i w_i^\top$$

Let ε_i be independent random ± 1 signs. Let

$$\bar{v} := \sum_i \varepsilon_i v_i \qquad \bar{w} := \sum_i \varepsilon_i w_i$$

The rank-one trick

Proposition

Let A be a matrix and decompose A as a sum of rank-one terms,

$$A = \sum_i v_i w_i^\top$$

Let ε_i be independent random ± 1 signs. Let

$$\bar{v} := \sum_i \varepsilon_i v_i \qquad \bar{w} := \sum_i \varepsilon_i w_i$$

Then $\bar{v}\bar{w}^\top$ is an unbiased, rank-one estimate of A:

$$\mathbb{E}\,\bar{v}\bar{w}^\top = A$$

Proof: expand, $\mathbb{E}\varepsilon_i^2 = 1$ and $\mathbb{E}\varepsilon_i\varepsilon_j = 0$.

The rank-one trick (2)

- Easy to compute

The rank-one trick (2)

- Easy to compute
- Extends to tensors of arbitrary order (use complex roots of unity instead of ± 1).

The rank-one trick (2)

- Easy to compute
- Extends to tensors of arbitrary order (use complex roots of unity instead of ± 1).
- Can reduce variance a lot by first rescaling v_i and w_i:

$$v_i w_i^\top = (\lambda_i v_i)\,(w_i^\top / \lambda_i)$$

The rank-one trick (2)

- Easy to compute
- Extends to tensors of arbitrary order (use complex roots of unity instead of ± 1).
- Can reduce variance a lot by first rescaling v_i and w_i:

$$v_i w_i^\top = (\lambda_i v_i)\,(w_i^\top / \lambda_i)$$

Optimal choice of λ_i: first equalize the norms of v_i and w_i.

The rank-one trick (2)

- Easy to compute
- Extends to tensors of arbitrary order (use complex roots of unity instead of ± 1).
- Can reduce variance a lot by first rescaling v_i and w_i:

$$v_i w_i^\top = (\lambda_i v_i)\,(w_i^\top / \lambda_i)$$

Optimal choice of λ_i: first equalize the norms of v_i and w_i.
Very important in practice.

Corollary

Apply the rank-one trick at each step. Then $\bar{v}_t \bar{w}_t^\top$ is an unbiased estimate of G_t at every step if

$$\bar{w}_{t+1} = \bar{w}_t + \sum_i \varepsilon_i \, \frac{\partial F_i}{\partial \theta}$$

$$\bar{v}_{t+1} = \frac{\partial F}{\partial h} \cdot \bar{v}_t + \sum_i \varepsilon_i \, e_i$$

where e_i is the i-th basis vector in state space, and where the ε_i are random ± 1 signs.

Corollary

Apply the rank-one trick at each step. Then $\bar{v}_t \bar{w}_t^\top$ is an unbiased estimate of G_t at every step if

$$\bar{w}_{t+1} = \bar{w}_t + \sum_i \varepsilon_i \frac{\partial F_i}{\partial \theta}$$

$$\bar{v}_{t+1} = \frac{\partial F}{\partial h} \cdot \bar{v}_t + \sum_i \varepsilon_i e_i$$

where e_i is the i-th basis vector in state space, and where the ε_i are random ± 1 signs.

(Scaling by λ_i omitted for clarity.)

Corollary

Apply the rank-one trick at each step. Then $\bar{v}_t \bar{w}_t^\top$ is an unbiased estimate of G_t at every step if

$$\bar{w}_{t+1} = \bar{w}_t + \sum_i \varepsilon_i \, \frac{\partial F_i}{\partial \theta}$$

$$\bar{v}_{t+1} = \frac{\partial F}{\partial h} \cdot \bar{v}_t + \sum_i \varepsilon_i \, e_i$$

where e_i is the i-th basis vector in state space, and where the ε_i are random ± 1 signs.

(Scaling by λ_i omitted for clarity.)

For RNNs: same computational cost as running the RNN itself.

Corollary

Apply the rank-one trick at each step. Then $\bar{v}_t \bar{w}_t^\top$ is an unbiased estimate of G_t at every step if

$$\bar{w}_{t+1} = \bar{w}_t + \sum_i \varepsilon_i \, \frac{\partial F_i}{\partial \theta}$$

$$\bar{v}_{t+1} = \frac{\partial F}{\partial h} \cdot \bar{v}_t + \sum_i \varepsilon_i \, e_i$$

where e_i is the i-th basis vector in state space, and where the ε_i are random ± 1 signs.

(Scaling by λ_i omitted for clarity.)

For RNNs: same computational cost as running the RNN itself.

In RNNs, $\dfrac{\partial F_i}{\partial \theta}$ is sparse since $h_i(t+1)$ depends on only a small subset of parameters.

NoBackTrack: variants

This was Euclidean NoBackTrack.

NoBackTrack: variants

This was Euclidean NoBackTrack.

Kalman NoBackTrack is obtained by feeding this gradient estimate to a Kalman filter (with diagonal or block-diagonal covariance matrix).

NoBackTrack: variants

This was Euclidean NoBackTrack.

Kalman NoBackTrack is obtained by feeding this gradient estimate to a Kalman filter (with diagonal or block-diagonal covariance matrix). Related to using a natural gradient on θ instead of the Euclidean gradient.

NoBackTrack: variants

This was Euclidean NoBackTrack.

Kalman NoBackTrack is obtained by feeding this gradient estimate to a Kalman filter (with diagonal or block-diagonal covariance matrix). Related to using a natural gradient on θ instead of the Euclidean gradient.

In practice, Kalman NoBackTrack reduces the noise in sensitive directions.

NoBackTrack: variants

This was Euclidean NoBackTrack.

Kalman NoBackTrack is obtained by feeding this gradient estimate to a Kalman filter (with diagonal or block-diagonal covariance matrix). Related to using a natural gradient on θ instead of the Euclidean gradient.

In practice, Kalman NoBackTrack reduces the noise in sensitive directions.

Can also do rank-k instead of rank-one.

NoBackTrack: variants

This was Euclidean NoBackTrack.

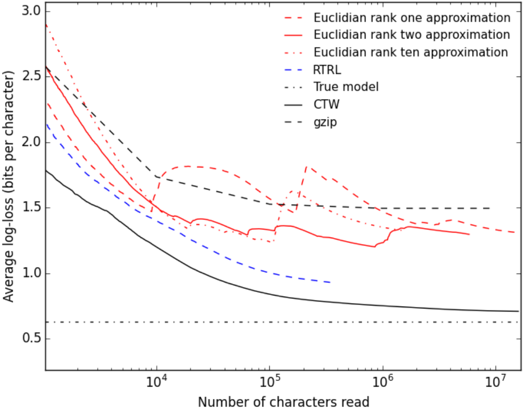
Kalman NoBackTrack is obtained by feeding this gradient estimate to a Kalman filter (with diagonal or block-diagonal covariance matrix). Related to using a natural gradient on θ instead of the Euclidean gradient.

In practice, Kalman NoBackTrack reduces the noise in sensitive directions.

Can also do rank-k instead of rank-one.

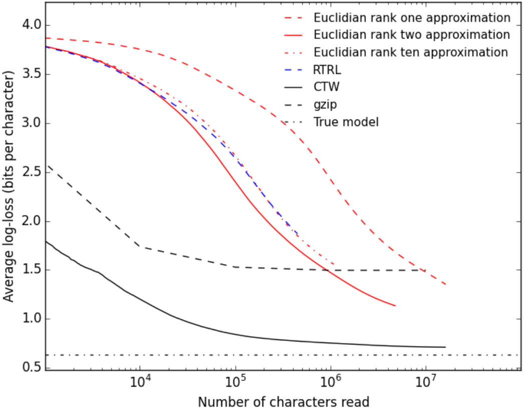
Does it work?

Large learning rate, non-Kalman: noise is clearly visible.



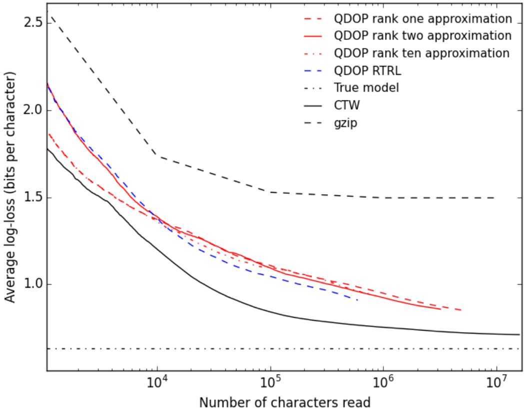
Compression rate (bits per characters) as a function of the number of characters read, for predicting the next character of a synthetic music notation model.

Small learning rate, non-Kalman: tracks the real gradient accurately.



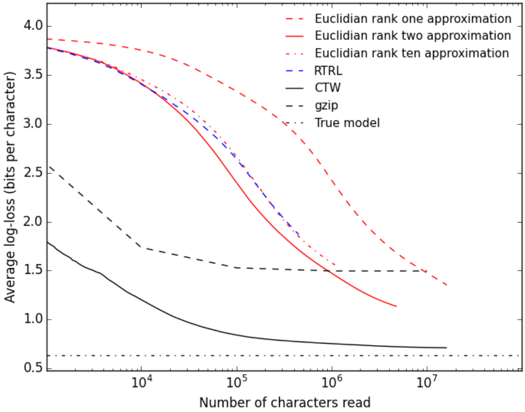
Compression rate (bits per characters) as a function of the number of characters read, for predicting the next character of a synthetic music notation model.

Large learning rate, Kalman: noise is barely visible.



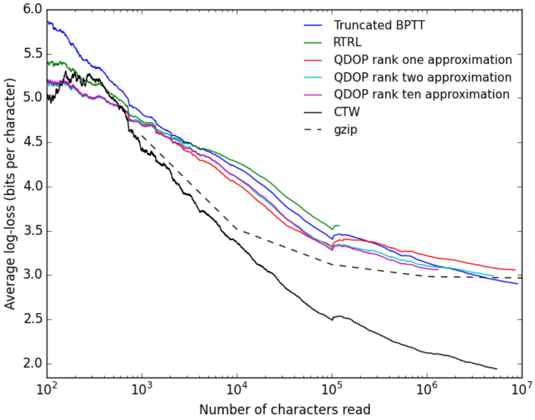
Compression rate (bits per characters) as a function of the number of characters read, for predicting the next character of a synthetic music notation model.

Small learning rate, non-Kalman: tracks the real gradient accurately.



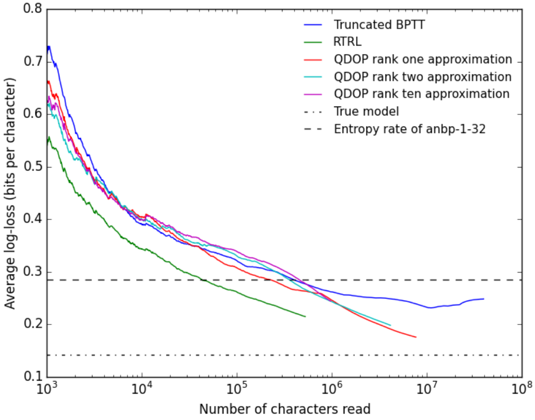
Compression rate (bits per characters) as a function of the number of characters read, for predicting the next character of a synthetic music notation model.

On Shakespeare's collected works, does no better, no worse than truncated backprop through time.



Compression rate (bits per characters) as a function of the number of characters read, for predicting the next character in Shakespeare's complete works.

On the $a^n b^n$ problem, clearly does better than truncated backprop through time when the span of time dependencies is longer than the truncation length for BPTT.



Compression rate (bits per characters) as a function of the number of characters read, for predicting the next character of $a^n b^n$ sequences using a leaky RNN model.

Open problems and future work

Open problems and future work

- Continuous time systems: OK in principle, $\delta t \to 0$ limit.

Open problems and future work

- Continuous time systems: OK in principle, $\delta t \to 0$ limit.
- Extends to other similar situations, such as online EM for Hidden Markov models

Open problems and future work

- Continuous time systems: OK in principle, $\delta t \to 0$ limit.
- Extends to other similar situations, such as online EM for Hidden Markov models
- Do the same with the Kalman covariance matrix/Fisher information matrix?

Open problems and future work

- Continuous time systems: OK in principle, $\delta t \to 0$ limit.
- Extends to other similar situations, such as online EM for Hidden Markov models
- Do the same with the Kalman covariance matrix/Fisher information matrix?
- How to deal with parameters that have a non-continuous influence on the trajectory, such as probabilities to make certain discrete choices?

Open problems and future work

- Continuous time systems: OK in principle, $\delta t \to 0$ limit.
- Extends to other similar situations, such as online EM for Hidden Markov models
- Do the same with the Kalman covariance matrix/Fisher information matrix?
- How to deal with parameters that have a non-continuous influence on the trajectory, such as probabilities to make certain discrete choices?
- Real physical systems: response of the environment depends on the actions of the system (RL), but no gradient available for this.

Open problems and future work

- Continuous time systems: OK in principle, $\delta t \to 0$ limit.
- Extends to other similar situations, such as online EM for Hidden Markov models
- Do the same with the Kalman covariance matrix/Fisher information matrix?
- How to deal with parameters that have a non-continuous influence on the trajectory, such as probabilities to make certain discrete choices?
- Real physical systems: response of the environment depends on the actions of the system (RL), but no gradient available for this.

Thank you!