Learning Algorithms for RNNs

JOHANNES KEPLER UNIVERSITY LINZ

- Backpropagation through time (BPTT)
- Truncated BPTT
- Real-Time Recurrent Learning (RTRL)
- Focused Backpropagation

Gradient of the Loss

learning algorithm for the fully recurrent network

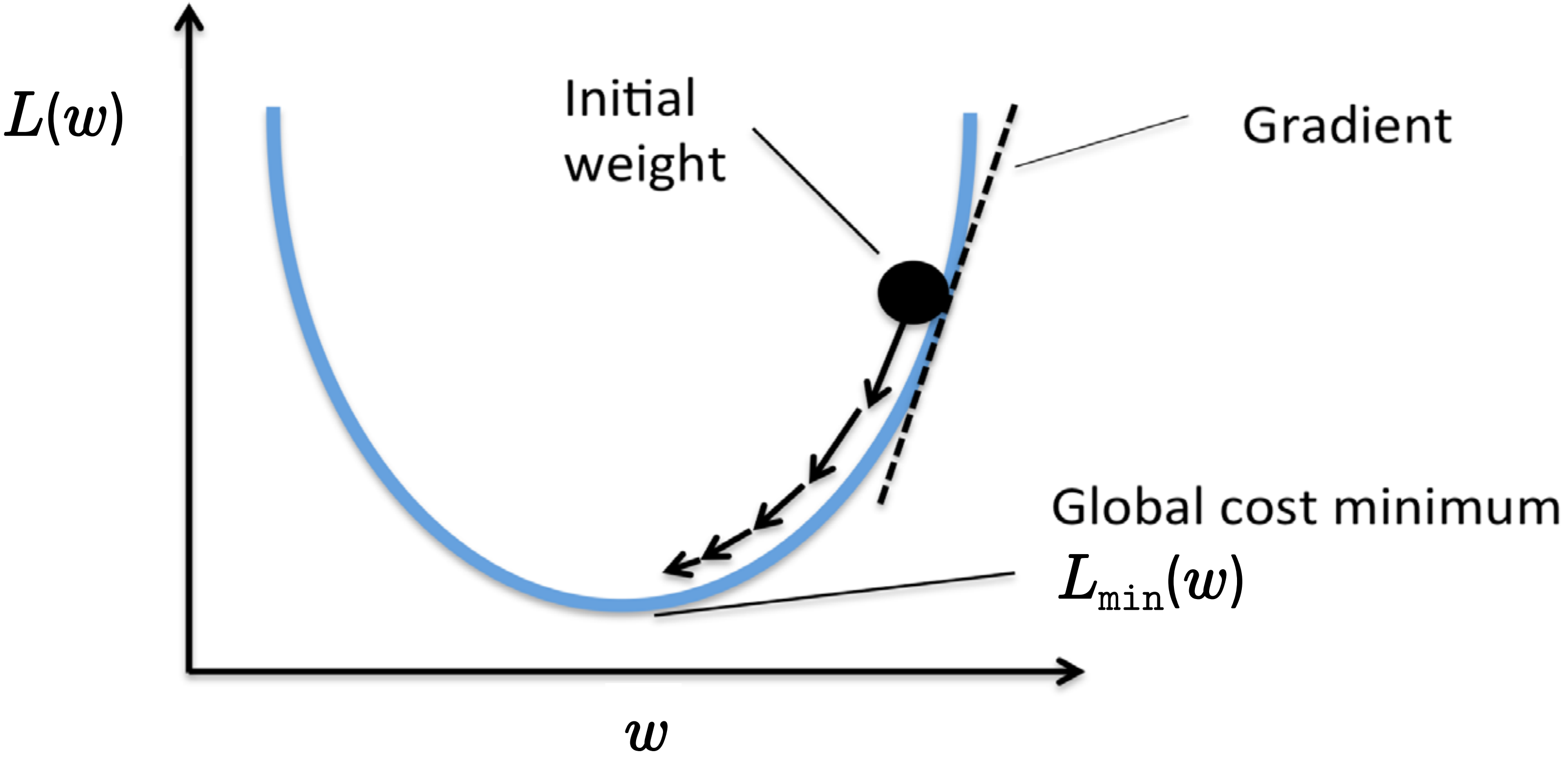
given a loss function L

alter the weights such that the loss becomes smaller

→ gradient descent

Backpropagation: Computing the gradient efficiently

Gradient of the Loss



Gradient of the Loss

How to compute the gradient for a recurrent network?

- loops in the network connections

- weight sharing

→ unfolding / unrolling the recurrent network in time

Gradient of the Loss

$$R_{\mathrm{emp}}\left(\left\{\boldsymbol{y}(1{:}T)^{(n)}, \hat{\boldsymbol{y}}(1{:}T)^{(n)}\right\}_{n=1}^{N}\right) = \frac{1}{N}\sum_{n=1}^{N} L\left(\boldsymbol{y}(1{:}T)^{(n)}, \hat{\boldsymbol{y}}(1{:}T)^{(n)}\right)$$

$$= \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T} L\left(\boldsymbol{y}(t)^{(n)}, \hat{\boldsymbol{y}}(t)^{(n)}\right) = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T} L\left(\boldsymbol{y}(t)^{(n)}, \boldsymbol{g}\left(\boldsymbol{a}(0), \boldsymbol{x}(1{:}t)^{(n)}; \boldsymbol{w}\right)\right)$$

R_{emp}: empirical risk / trainings error / expected loss (over samples)
L: loss / error for one sample
N: number training examples
T: length of the sequences
$\boldsymbol{w}$: parameter vector for the neural network

Gradient of the Loss

$$\boldsymbol{w}^{\text{new}} = \boldsymbol{w}^{\text{old}} - \eta \nabla_{\boldsymbol{w}} L$$ one sample with learning rate $\eta > 0$

$\boldsymbol{x}(t) \in \mathbb{R}^D$

$\boldsymbol{a}(t) \in \mathbb{R}^I$

$\hat{\boldsymbol{y}}(t) \in \mathbb{R}^K$

$\boldsymbol{W} \in \mathbb{R}^{D \times I}$

$\boldsymbol{R} \in \mathbb{R}^{I \times I}$

$\boldsymbol{V} \in \mathbb{R}^{I \times K}$

$t \in \{1, \ldots, T\}$

$\boldsymbol{a}(0) = \boldsymbol{0}$

$(\boldsymbol{y}(t))_{t=1}^{T}$

$$\boldsymbol{s}(t) = \boldsymbol{W}^\top \boldsymbol{x}(t) + \boldsymbol{R}^\top \boldsymbol{a}(t-1)$$

$$\boldsymbol{a}(t) = f(\boldsymbol{s}(t))$$

$$\hat{\boldsymbol{y}}(t) = g\left(\boldsymbol{V}^\top \boldsymbol{a}(t)\right)$$

$$L = \sum_{t=1}^{T} L(\boldsymbol{y}(t), \hat{\boldsymbol{y}}(t))$$

Gradients of V

$$\frac{\partial L}{\partial v_{ik}} = \sum_{t=1}^{T} \frac{\partial L(\boldsymbol{y}(t), \hat{\boldsymbol{y}}(t))}{\partial v_{ik}}$$

$$= \sum_{t=1}^{T} \frac{\partial L(\boldsymbol{y}(t), \hat{\boldsymbol{y}}(t))}{\partial \hat{\boldsymbol{y}}(t)} \frac{\partial \hat{\boldsymbol{y}}(t)}{\partial v_{ik}}$$

$$= \sum_{t=1}^{T} \frac{\partial L(\boldsymbol{y}(t), \hat{\boldsymbol{y}}(t))}{\partial \hat{y}_k(t)} \frac{\partial \hat{y}_k(t)}{\partial v_{ik}}$$

$$\hat{y}_k = g\left(\sum_{i=1}^{I} v_{ik} a_i(t)\right)$$

$$\partial \hat{y}_\ell / \partial v_{ik} = 0$$

$$\ell \neq k$$

Gradient of V

The term $\partial L(\boldsymbol{y}(t), \hat{\boldsymbol{y}}(t)) / \partial \hat{y}_k(t)$ depends on the loss function.

For least square loss: $\partial L(\boldsymbol{y}(t), \hat{\boldsymbol{y}}(t)) / \partial \hat{y}_k(t) = \hat{y}_k(t) - y_k(t)$

$$e_k(t) := \partial L(\boldsymbol{y}(t), \hat{\boldsymbol{y}}(t)) / \partial \hat{y}_k(t)$$

$$\frac{\partial \hat{y}_k}{\partial v_{ik}} = \sum_{j=1}^{I} \frac{\partial v_{jk} a_j(t)}{\partial v_{ik}} = g'(a_i(t)) a_i(t)$$

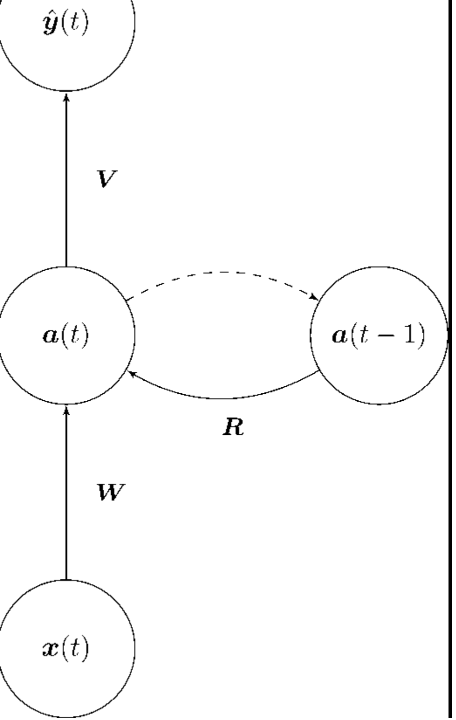
$\hat{y}_k = \sum_{i=1}^{I} v_{ik} a_i(t)$

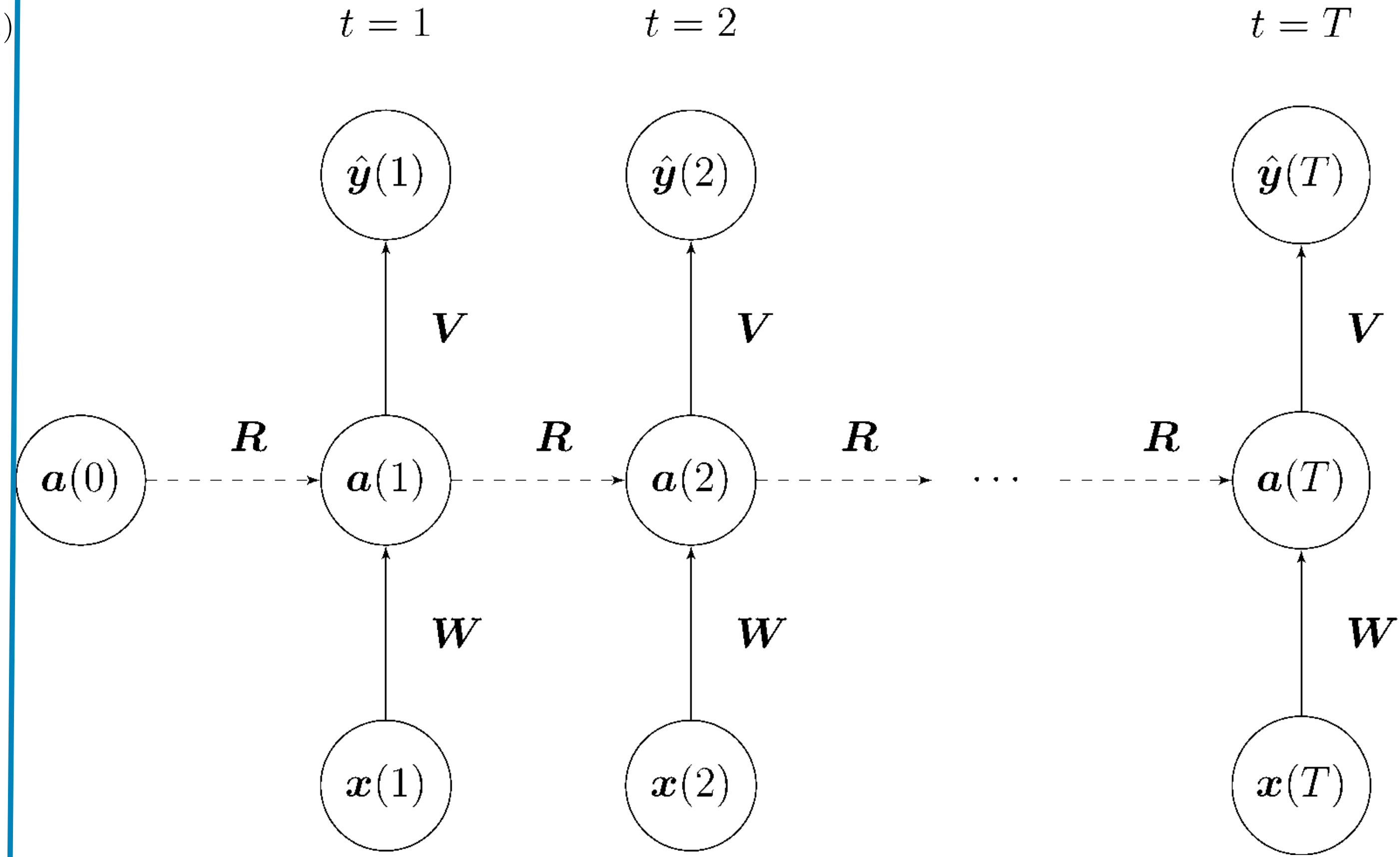
$$\frac{\partial L}{\partial \boldsymbol{V}} = \sum_{t=1}^{T} \boldsymbol{e}(t) \operatorname{diag}(\boldsymbol{g}'(\boldsymbol{V}^\top \boldsymbol{a}(t))) \boldsymbol{a}(t)^\top$$

$$\frac{\partial L}{\partial \boldsymbol{V}} = \sum_{t=1}^{T} \boldsymbol{e}(t) \operatorname{diag}(\boldsymbol{g}'(\boldsymbol{V}^\top \boldsymbol{a}(t))) \boldsymbol{a}(t)^\top$$

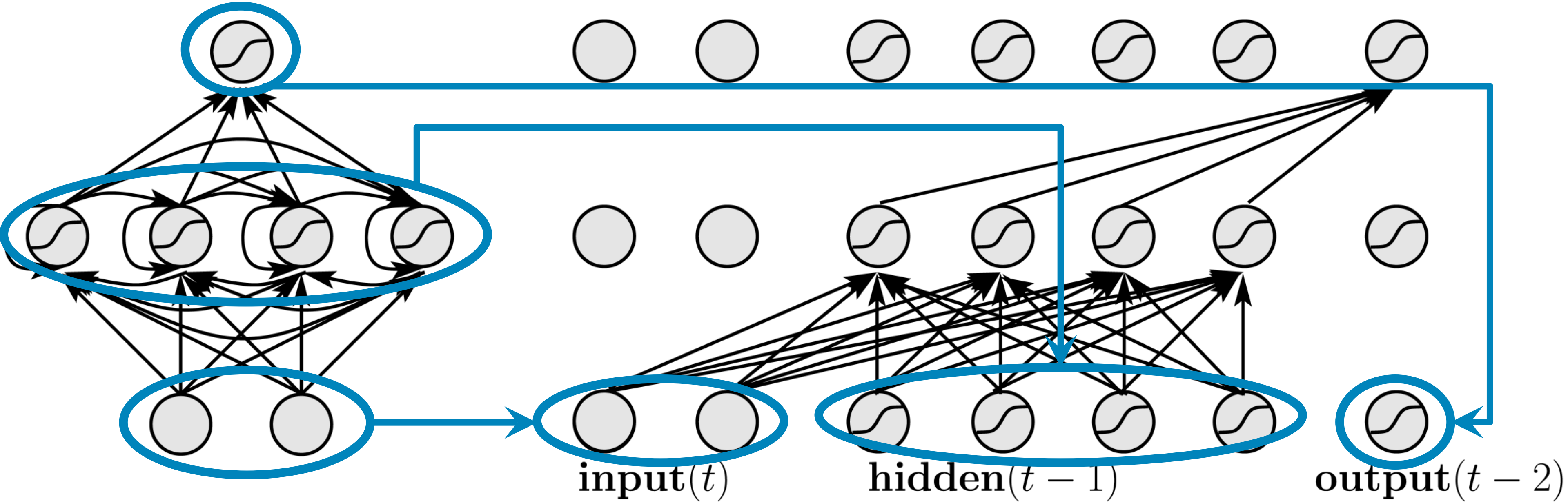
Backpropagation through time

$$\boldsymbol{s}(t) = \boldsymbol{W}^\top \boldsymbol{x}(t) + \boldsymbol{R}^\top \boldsymbol{a}(t-1)$$
$$\boldsymbol{a}(t) = f(\boldsymbol{s}(t))$$
$$\hat{\boldsymbol{y}}(t) = g(\boldsymbol{V}^\top \boldsymbol{a}(t))$$

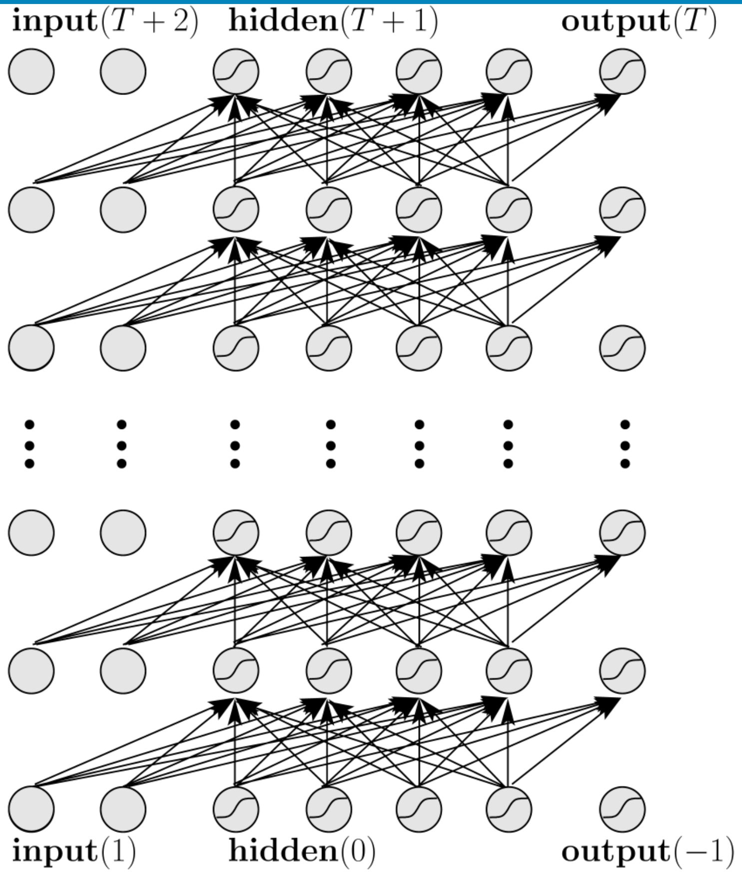


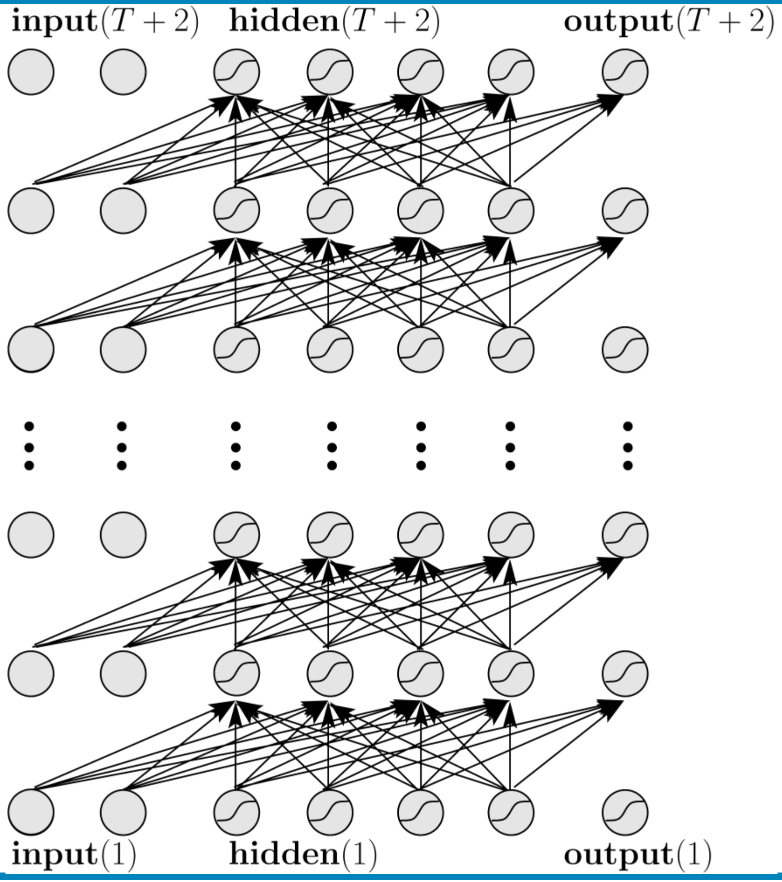


Backpropagation through time



Backpropagation through time





Delta Propagation: Recursion

$\boldsymbol{s}(t) = \boldsymbol{W}^\top \boldsymbol{x}(t) + \boldsymbol{R}^\top \boldsymbol{a}(t-1)$

$\boldsymbol{a}(t) = f(\boldsymbol{s}(t))$

$\hat{\boldsymbol{y}}(t) = g(\boldsymbol{V}^\top \boldsymbol{a}(t))$

Backpropagation is also called -propagation

are derivatives of L w.r.t. pre-activations:

$$\boxed{\boldsymbol{\delta}(t)^\top = \frac{\partial L}{\partial \boldsymbol{s}(t)}} = \frac{\partial L}{\partial \boldsymbol{a}(t)} \frac{\partial \boldsymbol{a}(t)}{\partial \boldsymbol{s}(t)}$$

$$= \left(\frac{\partial L(\boldsymbol{y}(t), \hat{\boldsymbol{y}}(t))}{\partial \boldsymbol{a}(t)} + \frac{\partial L}{\partial \boldsymbol{s}(t+1)} \frac{\partial \boldsymbol{s}(t+1)}{\partial \boldsymbol{a}(t)} \right) \frac{\partial \boldsymbol{a}(t)}{\partial \boldsymbol{s}(t)}$$

$$= \left(\frac{\partial L}{\partial \hat{\boldsymbol{y}}(t)} \frac{\partial \hat{\boldsymbol{y}}(t)}{\partial \boldsymbol{a}(t)} + \frac{\partial L}{\partial \boldsymbol{s}(t+1)} \frac{\partial \boldsymbol{s}(t+1)}{\partial \boldsymbol{a}(t)} \right) \frac{\partial \boldsymbol{a}(t)}{\partial \boldsymbol{s}(t)}$$

$$= \left(\boldsymbol{e}(t)^\top \operatorname{diag}(g'(\boldsymbol{V}^\top \boldsymbol{a}(t))) \boldsymbol{V}^\top + \boldsymbol{\delta}(t+1)^\top \boldsymbol{R}^\top \right) \operatorname{diag}(f'(\boldsymbol{s}(t)))$$

recursion starting at time $t{=}T$ and going backward to $t{=}0$.

Gradient with respect to R

We equip the weights with a time index to track them during the forward pass: $\boldsymbol{W}(1) = \cdots = \boldsymbol{W}(T)$ and $\boldsymbol{R}(1) = \cdots = \boldsymbol{R}(T)$

$$\boldsymbol{s}(t) = \boldsymbol{W}(t)^{\top} \boldsymbol{x}(t) + \boldsymbol{R}(t)^{\top} \boldsymbol{a}(t-1)$$

The gradient with respect to $\boldsymbol{R}$ is:

$$\frac{\partial L}{\partial r_{ij}} = \sum_{t=1}^{T} \frac{\partial L}{\partial r_{ij}(t)} = \sum_{t=1}^{T} \frac{\partial L}{\partial \boldsymbol{s}(t)} \frac{\partial \boldsymbol{s}(t)}{\partial r_{ij}(t)} = \sum_{t=1}^{T} \frac{\partial L}{\partial s_i(t)} \frac{\partial s_i(t)}{\partial r_{ij}(t)} = \sum_{t=1}^{T} \delta_i(t) a_j(t-1)$$

Gradient with respect to W

Analog we derive the gradient with respect to $\boldsymbol{W}$ by BPTT:

$$\frac{\partial L}{\partial w_{ij}} = \sum_{t=1}^{T} \delta_i(t) x_j(t)$$

All Gradients for BPTT

$$\frac{\partial L}{\partial \boldsymbol{W}} = \sum_{t=1}^{T} \boldsymbol{\delta}(t) \boldsymbol{x}(t)^{\top}$$

$$\frac{\partial L}{\partial \boldsymbol{R}} = \sum_{t=1}^{T} \boldsymbol{\delta}(t) \boldsymbol{a}(t)^{\top}$$

$$\frac{\partial L}{\partial \boldsymbol{V}} = \sum_{t=1}^{T} \boldsymbol{e}(t)) \operatorname{diag}(\boldsymbol{g}'(\boldsymbol{V}^{\top} \boldsymbol{a}(t))) \boldsymbol{a}(t)^{\top}$$

Backpropagation through time

BPTT requires to compute the gradients for every network output $\hat{\boldsymbol{y}}(t)$ with respect to all past time steps.

→ storage of all activations at all time steps

→ complexity of BPTT is $\mathcal{O}\left(TI\left(I+K+D\right)\right)$

→ BPTT is local in space (complexity per time step and weight is independent of the number of weights).

Truncated BPTT

Truncated backpropagation through time introduces a maximum number of τ time steps for the backward pass.

→ choice of τ determines how far on looks back

→ complexity of truncated BPTT is $\mathcal{O}\left(\tau I \left(I + K + D\right)\right)$

Truncated BPTT only approximates the gradient since functional dependencies from the forward pass are cut.
→ Training is not guaranteed to converge.

Real-Time Recurrent Learning

Real-time recurrent learning (RTRL) computes all contributions to the gradients during the forward pass.
The derivative of each unit with respect to each weight is tracked.

Thus, activations need not be stored along the whole sequence.
→RTRL is local in time (independent of the sequence length T).
→RTRL is an alternative for very long sequences.
→RTRL has at least complexity of $\mathcal{O}(I^4)$.

$\partial \boldsymbol{s}(t) / \partial \boldsymbol{R}$ has I^3 entries to be memorized and updated.
The update requires a sum over I terms.

Real-Time Recurrent Learning

We drop the time index t from the network parameters and consider only the n-th neuron:

$$\boldsymbol{s}(t) = \boldsymbol{W}(t)^\top \boldsymbol{x}(t) + \boldsymbol{R}(t)^\top \boldsymbol{a}(t-1)$$

$$s_n(t) = \sum_{l=1}^{D} w_{ln} x_l(t) + \sum_{k=1}^{I} r_{kn} a_k(t-1)$$

Dropping the time index means that we consider the weights as shared in time and consequently we have to incorporate the fact that $\boldsymbol{s}(t-1)$ also depends on $\boldsymbol{W}$ and $\boldsymbol{R}$.

Real-Time Recurrent Learning

JOHANNES KEPLER UNIVERSITY LINZ

recursive form of the derivative w.r.t. the network parameters:

$$\frac{\partial s_n(t)}{\partial r_{ij}} = \sum_{k=1}^{I} \frac{\partial r_{kn}}{\partial r_{ij}} a_k(t-1) + \sum_{k=1}^{I} r_{kn} \frac{\partial a_k(t-1)}{\partial r_{ij}}$$

$$= a_i(t-1)[n=j] + \sum_{k=1}^{I} r_{kn} f'(s_k(t-1)) \frac{\partial s_k(t-1)}{\partial r_{ij}}$$

$$\frac{\partial s_n(t)}{\partial w_{dj}} = \sum_{l=1}^{D} \frac{\partial w_{ln}}{\partial w_{dj}} x_l(t) + \sum_{k=1}^{I} r_{kn} \frac{\partial a_k(t-1)}{\partial w_{dj}}$$

$$= x_d(t)[n=j] + \sum_{k=1}^{I} r_{kn} f'(s_k(t-1)) \frac{\partial s_k(t-1)}{\partial w_{dj}}$$

$[n=j]$ is the Iverson bracket for logical expressions

Real-Time Recurrent Learning

Initialization of the recursions:

$$s_k(0) = 0 \qquad \frac{\partial s_k(0)}{\partial w_{dj}} = \frac{\partial s_k(0)}{\partial r_{ij}} = 0$$

The derivatives of the activations w.r.t. the parameters are

$$\frac{\partial \boldsymbol{a}(t)}{\partial \boldsymbol{R}} = \mathrm{diag}(f'(\boldsymbol{s}(t)))\frac{\partial \boldsymbol{s}(t)}{\partial \boldsymbol{R}}$$

At every time step t, RTRL must store the derivatives for computing the gradients for the next time step $t+1$.
→independent of the sequence length T

Real-Time Recurrent Learning

Gradient of the loss function:

$$\frac{\partial L(\boldsymbol{y}(t),\hat{\boldsymbol{y}}(t))}{\partial \boldsymbol{R}} = \sum_k \frac{\partial L(\boldsymbol{y}(t),\hat{\boldsymbol{y}}(t))}{\partial \hat{y}_k(t)} \frac{\partial \hat{y}_k(t)}{\partial \boldsymbol{R}} = \sum_k e_k(t) \frac{\partial a_k(t)}{\partial \boldsymbol{R}}$$

$$e_k(t) := \frac{\partial L(\boldsymbol{y}(t),\hat{\boldsymbol{y}}(t))}{\partial \hat{y}_k(t)} \quad , \qquad a_k(t) = \hat{y}_k(t)$$

Schmidhuber's Approach

Jürgen Schmidhuber's algorithm has complexity of $\mathcal{O}(I^3)$
→ is exact and not truncated

Idea:

- divide the sequence in block of length I.
- within each block BPTT is performed: $\mathcal{O}(I^3)$
- after each block RTRL is performed to collect all gradients

→ Complexity per time step for each block with RTRL is $\mathcal{O}(I^3)$
We have T/I such blocks → complexity of BPTT

Only the activations of sequences of length I have to be stored.