Lecture 8

Recurrent Neural Network (RNNs)

Sequential data

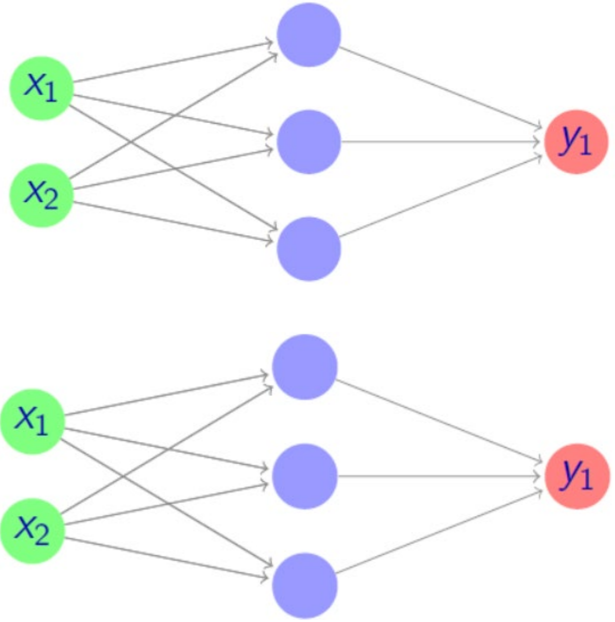
Recurrent neural networks (RNNs) are often used for handling sequential data.

They introduced first in 1986 (Rumelhart et al 1986).

Sequential data usually involves variable length inputs.

Parameter sharing

Parameter sharing makes it possible to extend and apply the model to examples of different lengths and generalize across them.



Recurrent neural network

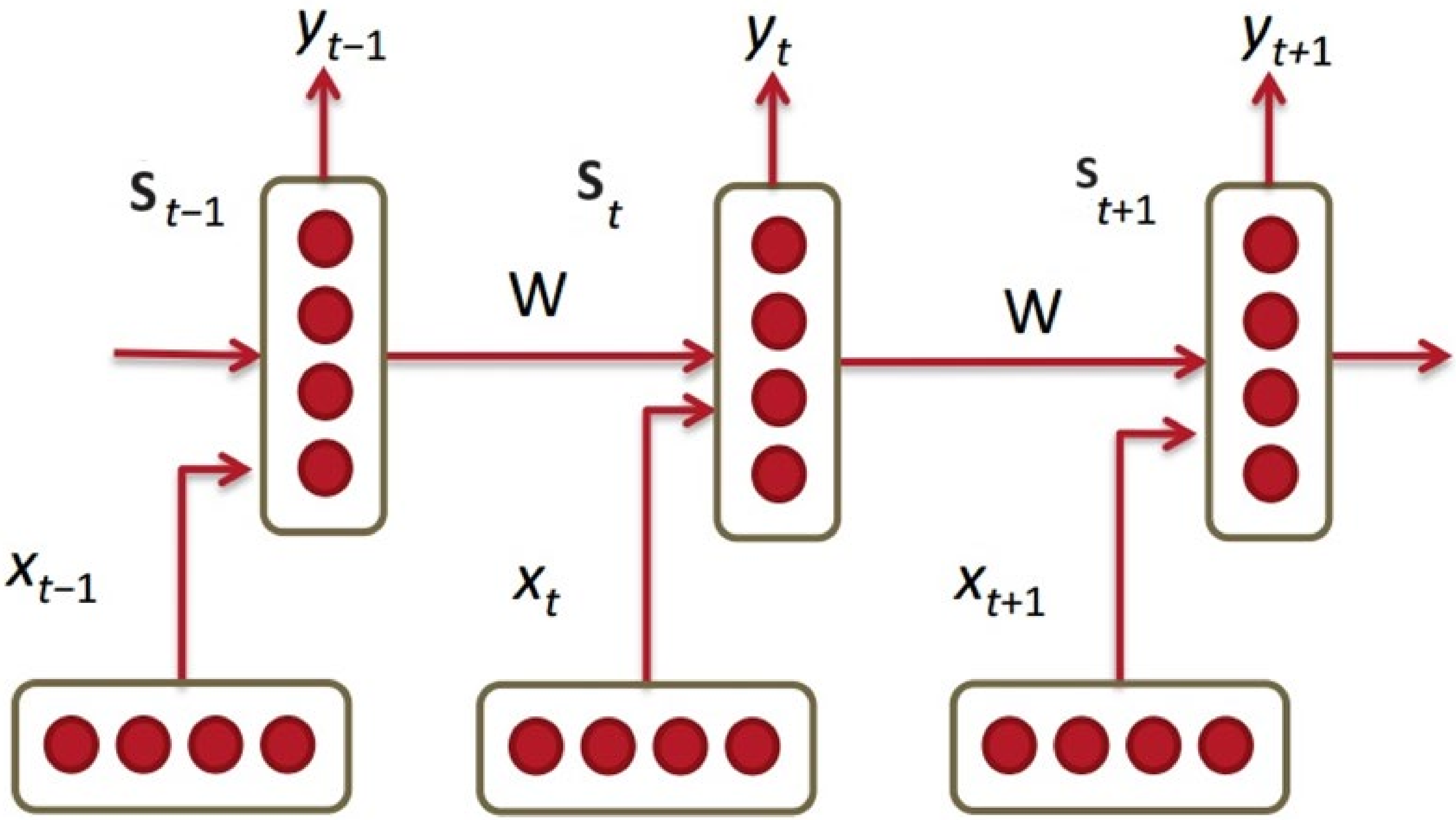
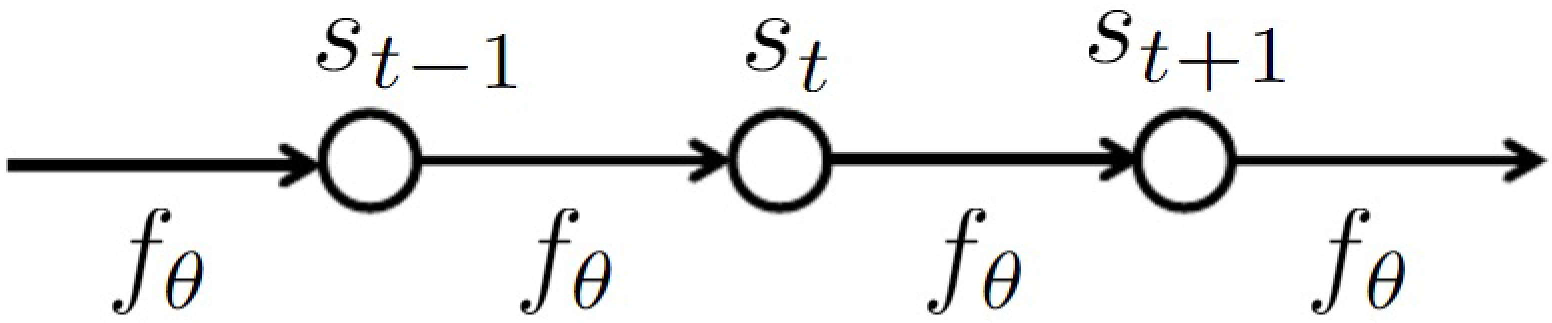


Figure: Richard Socher

Dynamic systems

The classical form of a dynamical system:

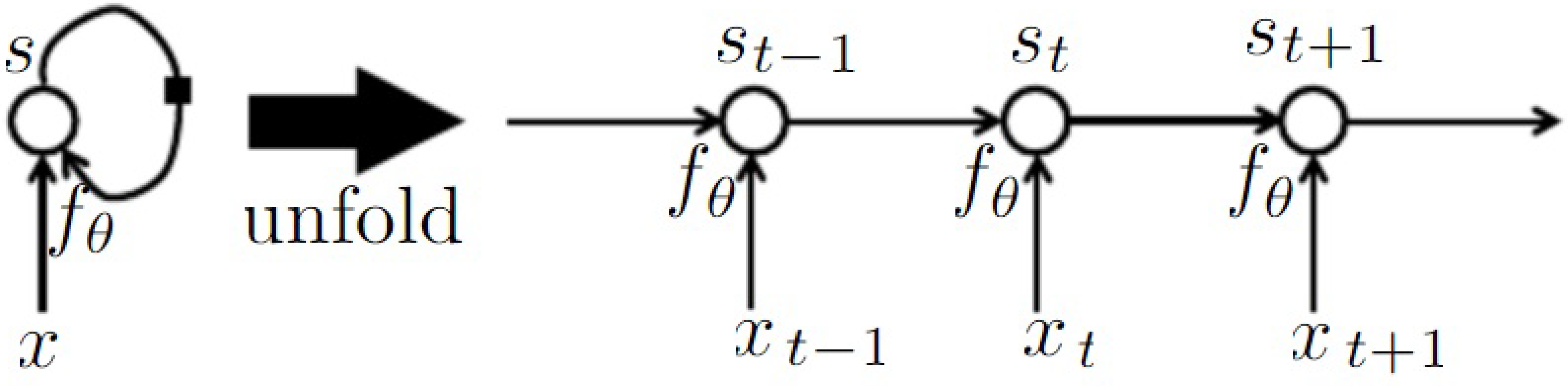
$$s_t = f_\theta(s_{t-1})$$



Dynamic systems

Now consider a dynamic system with an external signal x

$$s_t = f_\theta(s_{t-1}, x_t)$$



The state contains information about the whole past sequence.

$$s_t = g_t(x_t, x_{t-1}, x_{t-s}, \ldots, x_2, x_1)$$

Parameter sharing

We can think of s_t as a summary of the past sequence of inputs up to t.

Parameter sharing

We can think of s_t as a summary of the past sequence of inputs up to t.

If we define a different function g_t for each possible sequence length, we would not get any generalization.

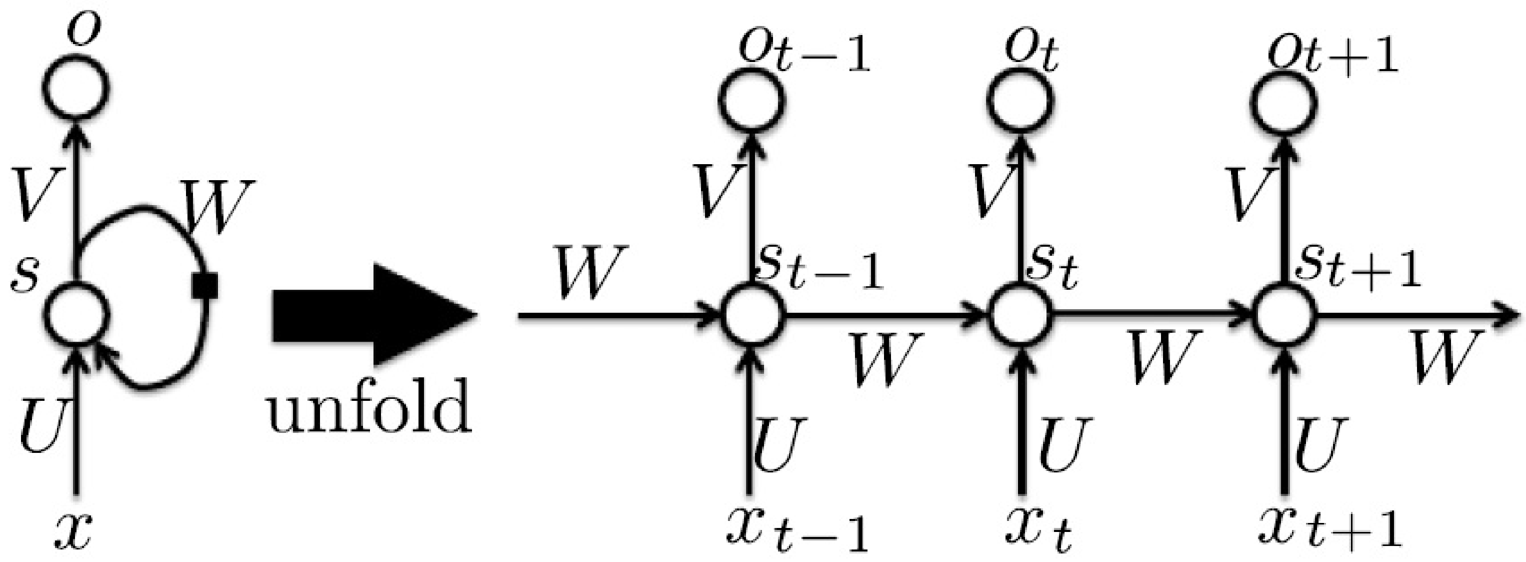
Parameter sharing

We can think of s_t as a summary of the past sequence of inputs up to t.

If we define a different function g_t for each possible sequence length, we would not get any generalization.

If the same parameters are used for any sequence length allowing much better generalization properties.

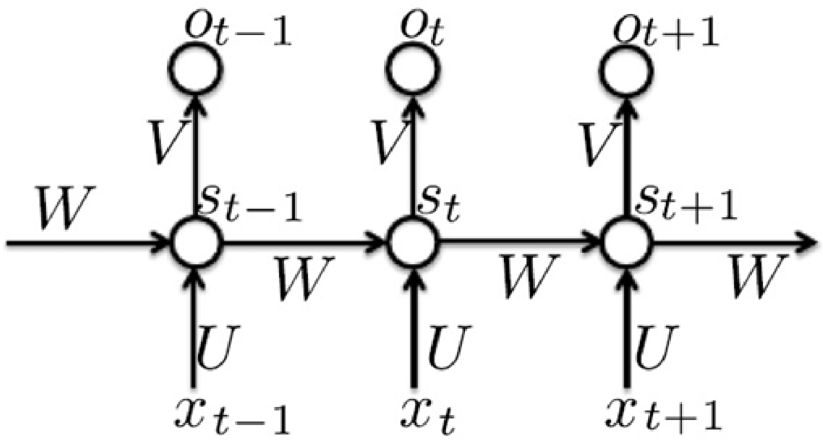
Recurrent Neural Networks



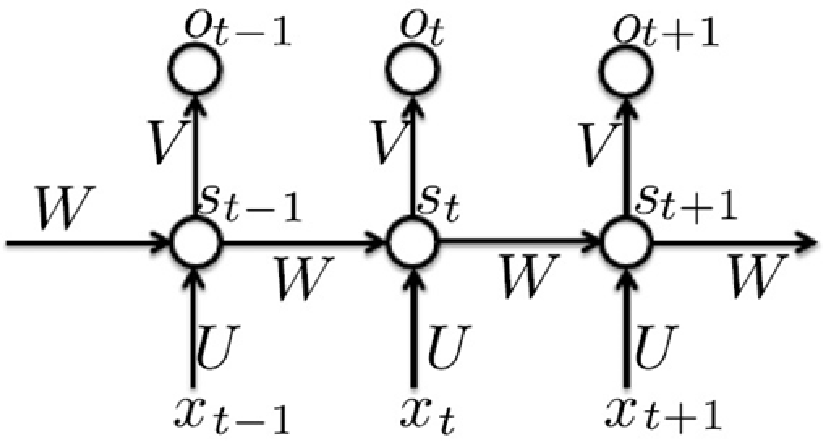
$$
\begin{aligned}
\mathbf{a}_t &= \mathbf{b} + W\mathbf{s}_{t-1} + U\mathbf{x}_t \\
\mathbf{s}_t &= \mathit{tanh}(\mathbf{a}_t) \\
\mathbf{o}_t &= \mathbf{c} + V\mathbf{s}_t \\
\mathbf{p}_t &= \mathit{softmax}(\mathbf{o}_t)
\end{aligned}
$$

Computing the Gradient in a Recurrent Neural Network

Using the generalized back-propagation algorithm one can obtain the so-called Back-Propagation Through Time (BPTT) algorithm.

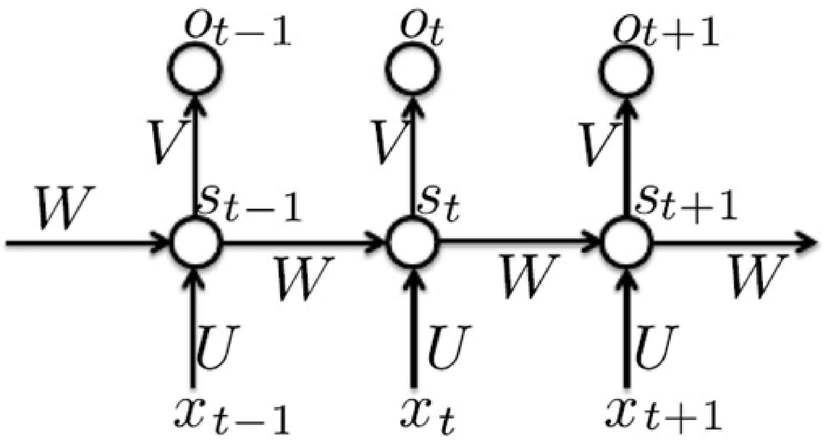


$$L = \sum_t L_t$$



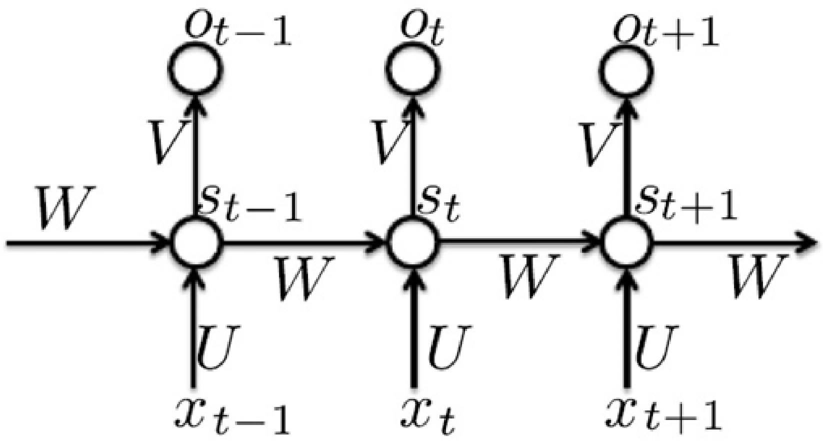
$$L = \sum_t L_t$$

$$\frac{\partial L}{\partial O_t} = \frac{\partial L}{\partial L_t} \cdot \frac{\partial L_t}{\partial O_t}$$



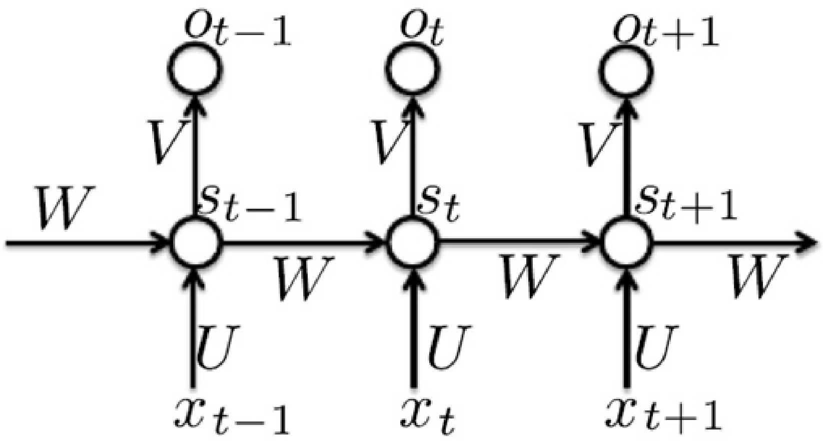
$$L = \sum_t L_t$$

$$\frac{\partial L}{\partial O_t} = \cancelto{\boxed{1}}{\frac{\partial L}{\partial L_t}} \cdot \frac{\partial L_t}{\partial O_t}$$



$$L = \sum_t L_t$$

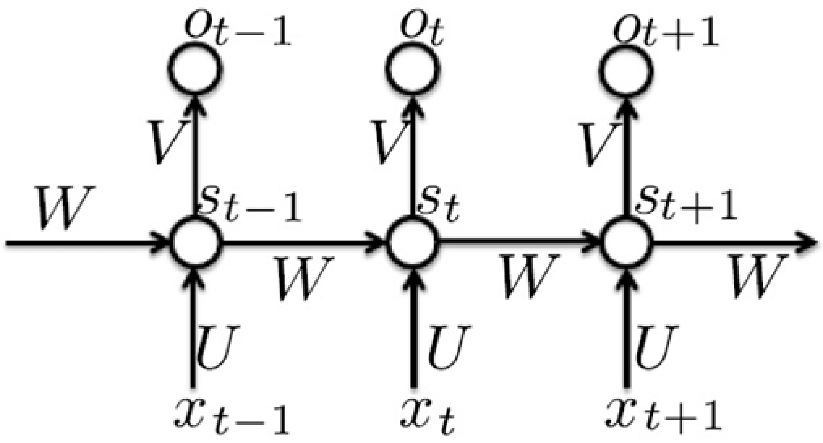
$$\frac{\partial L}{\partial O_t} = \cancelto{\boxed{1}}{\frac{\partial L}{\partial L_t}} \cdot \frac{\partial L_t}{\partial O_t}$$ ✓



$$L = \sum_t L_t$$

$$\frac{\partial L}{\partial O_t} = \cancelto{1}{\frac{\partial L}{\partial L_t}} . \frac{\partial L_t}{\partial O_t}$$ ✓

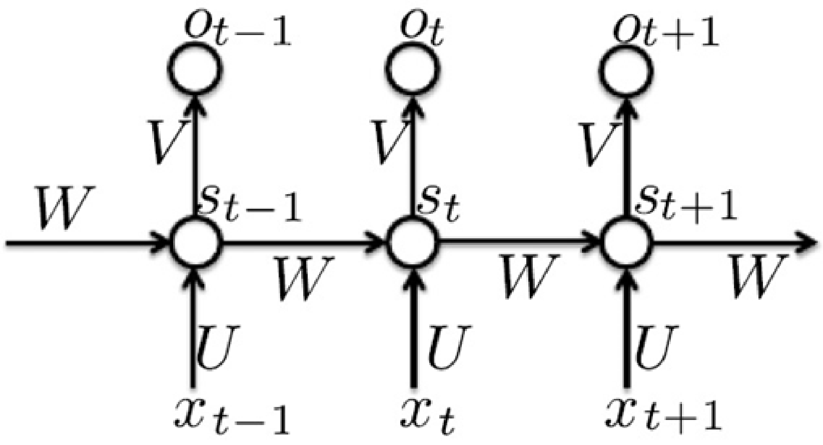
$$\frac{\partial L}{\partial S_T} = \frac{\partial L}{\partial O_T} . \frac{\partial O_T}{\partial S_T}$$



$$L = \sum_t L_t$$

$$\frac{\partial L}{\partial O_t} = \cancelto{1}{\frac{\partial L}{\partial L_t}} \cdot \frac{\partial L_t}{\partial O_t} \checkmark$$

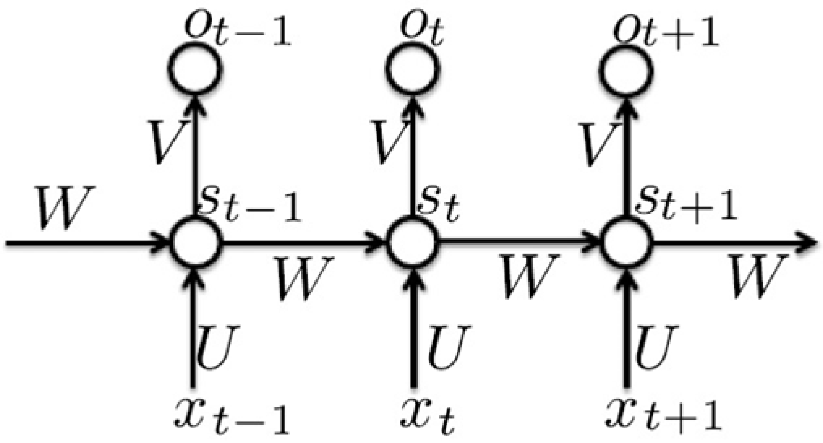
$$\frac{\partial L}{\partial S_T} \checkmark = \frac{\partial L}{\partial O_T} \cdot \frac{\partial O_T}{\partial S_T}$$



$$L = \sum_t L_t$$

$$\frac{\partial L}{\partial O_t} = \cancelto{1}{\frac{\partial L}{\partial L_t}} \cdot \frac{\partial L_t}{\partial O_t} \checkmark$$

$$\frac{\partial L}{\partial S_T} \overset{\checkmark}{=} \frac{\partial L}{\partial O_T} \cdot \cancelto{V}{\frac{\partial O_T}{\partial S_T}}$$

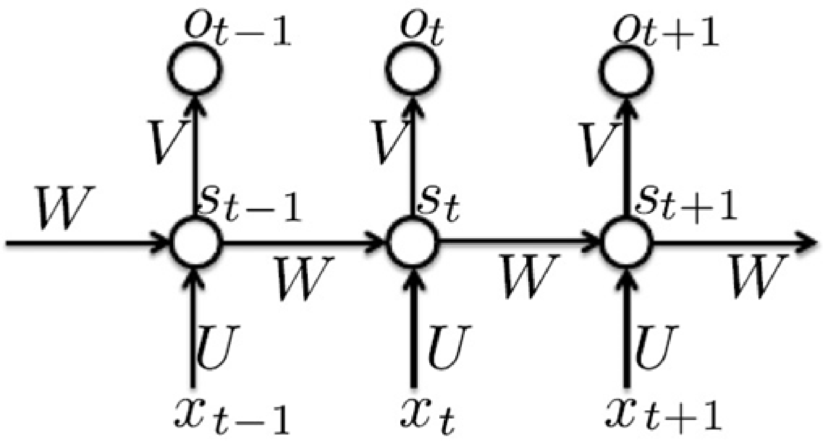


$$L = \sum_t L_t$$

$$\frac{\partial L}{\partial O_t} = \cancelto{1}{\frac{\partial L}{\partial L_t}} \cdot \frac{\partial L_t}{\partial O_t} \checkmark$$

$$\frac{\partial L}{\partial S_T} \checkmark = \frac{\partial L}{\partial O_T} \cdot \cancelto{V}{\frac{\partial O_T}{\partial S_T}}$$

$$\frac{\partial L}{\partial S_t} = \frac{\partial L}{\partial O_t} \cdot \frac{\partial O_t}{\partial S_t} + \frac{\partial L}{\partial S_{t+1}} \cdot \frac{\partial S_{t+1}}{\partial S_t}$$



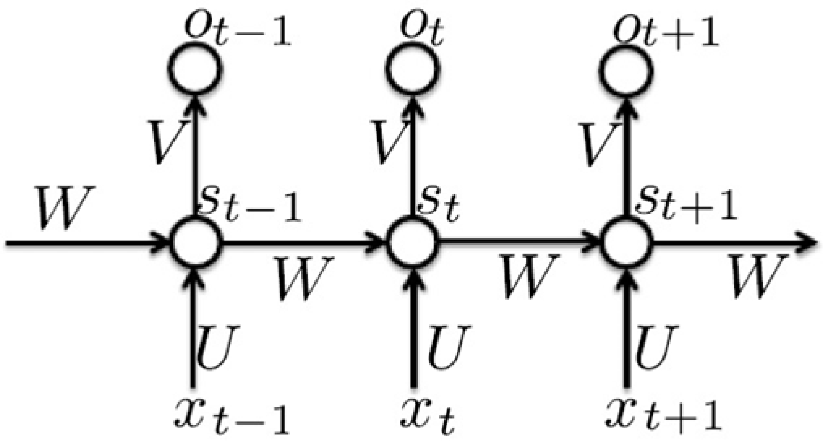
$$L = \sum_t L_t$$

$$\frac{\partial L}{\partial O_t} = \overset{\boxed{1}}{\frac{\partial L}{\partial L_t}} . \frac{\partial L_t}{\partial O_t} \checkmark$$

$$\frac{\partial L}{\partial S_T} = \checkmark \frac{\partial L}{\partial O_T} . \overset{\boxed{V}}{\frac{\partial O_T}{\partial S_T}}$$

$$\frac{\partial L}{\partial S_t} = \frac{\partial L}{\partial O_t} . \frac{\partial O_t}{\partial S_t} + \frac{\partial L}{\partial S_{t+1}} . \frac{\partial S_{t+1}}{\partial S_t}$$

$$\delta_t = \quad .V + \delta_{t+1}.W\left(1 - tanh^2(b + WS_{t-1} + Ux_t)\right)$$



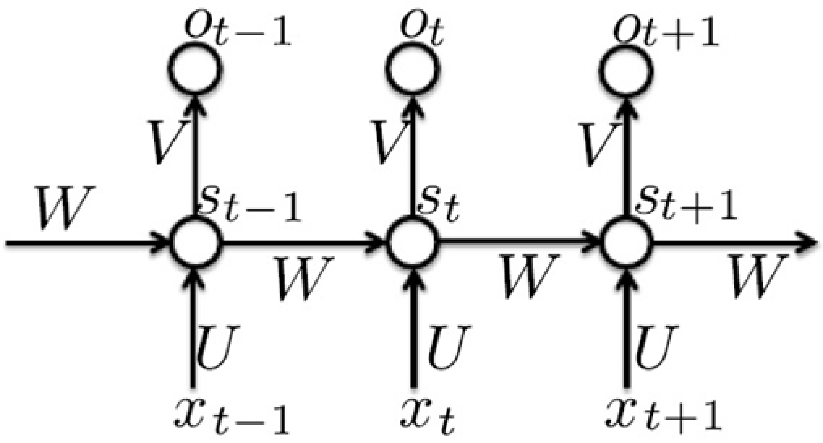
$$L = \sum_t L_t$$

$$\frac{\partial L}{\partial O_t} = \overset{\boxed{1}}{\cancel{\frac{\partial L}{\partial L_t}}}\cdot\frac{\partial L_t}{\partial O_t} \checkmark$$

$$\frac{\partial L}{\partial S_T} = \checkmark \frac{\partial L}{\partial O_T}\cdot\overset{\boxed{V}}{\cancel{\frac{\partial O_T}{\partial S_T}}}$$

$$\frac{\partial L}{\partial S_t} = \frac{\partial L}{\partial O_t}\cdot\frac{\partial O_t}{\partial S_t} + \frac{\partial L}{\partial S_{t+1}}\cdot\frac{\partial S_{t+1}}{\partial S_t}$$

$$\delta_t = \checkmark .V + \delta_{t+1}.W\left(1 - tanh^2(b + WS_{t-1} + Ux_t)\right)$$



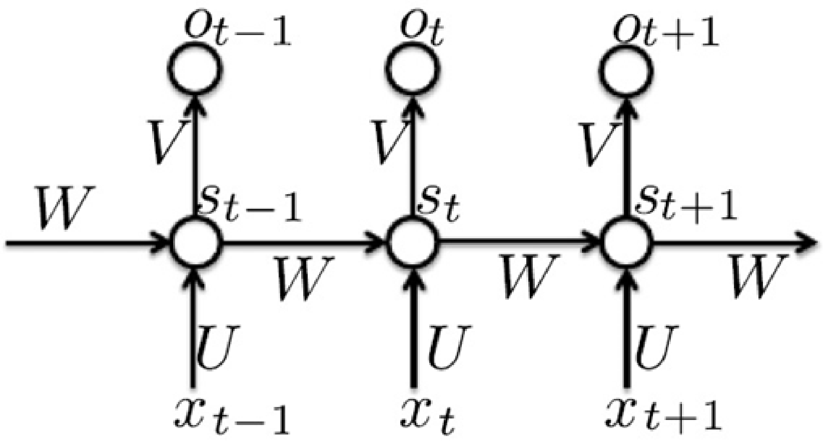
$$L = \sum_t L_t$$

$$\frac{\partial L}{\partial O_t} = \cancelto{\boxed{1}}{\frac{\partial L}{\partial L_t}} . \frac{\partial L_t}{\partial O_t} \checkmark$$

$$\frac{\partial L}{\partial S_T} \overset{\checkmark}{=} \frac{\partial L}{\partial O_T} . \cancelto{\boxed{V}}{\frac{\partial O_T}{\partial S_T}}$$

$$\frac{\partial L}{\partial S_t} = \frac{\partial L}{\partial O_t} . \frac{\partial O_t}{\partial S_t} + \frac{\partial L}{\partial S_{t+1}} . \frac{\partial S_{t+1}}{\partial S_t}$$

$$\delta_t = \overset{\checkmark}{\bigcirc} . V + \delta_{t+1} . W \left(1 - \underbrace{tanh^2(b + WS_{t-1} + Ux_t)}_{S_t^2}\right)$$



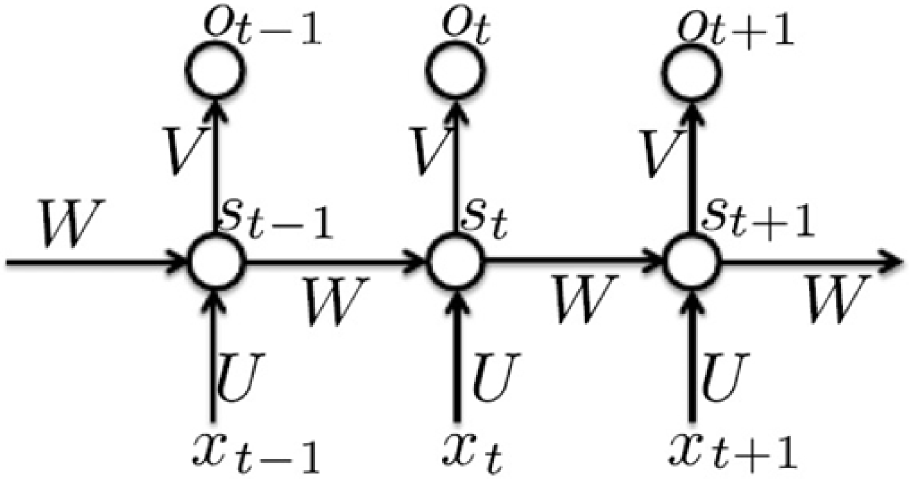
$$L = \sum_t L_t$$

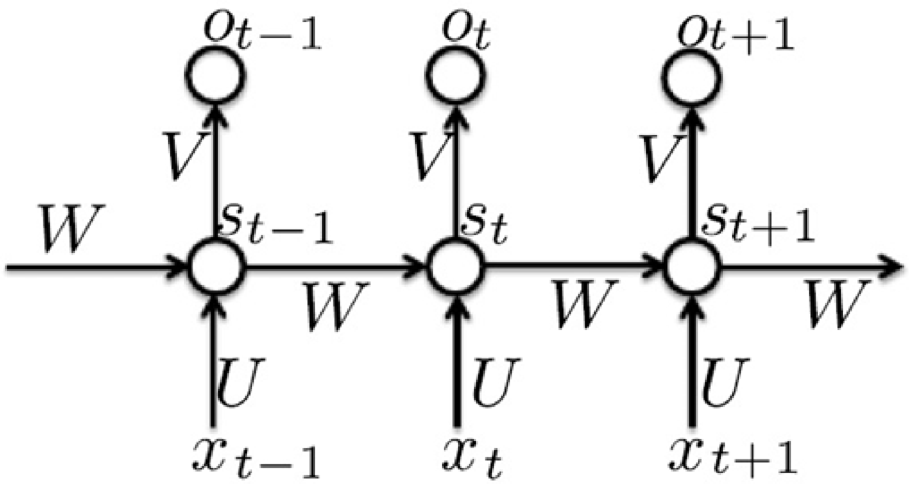
$$\frac{\partial L}{\partial O_t} = \cancelto{\boxed{1}}{\frac{\partial L}{\partial L_t}} \cdot \frac{\partial L_t}{\partial O_t} \checkmark$$

$$\frac{\partial L}{\partial S_T} \overset{\checkmark}{=} \frac{\partial L}{\partial O_T} \cdot \cancelto{\boxed{V}}{\frac{\partial O_T}{\partial S_T}}$$

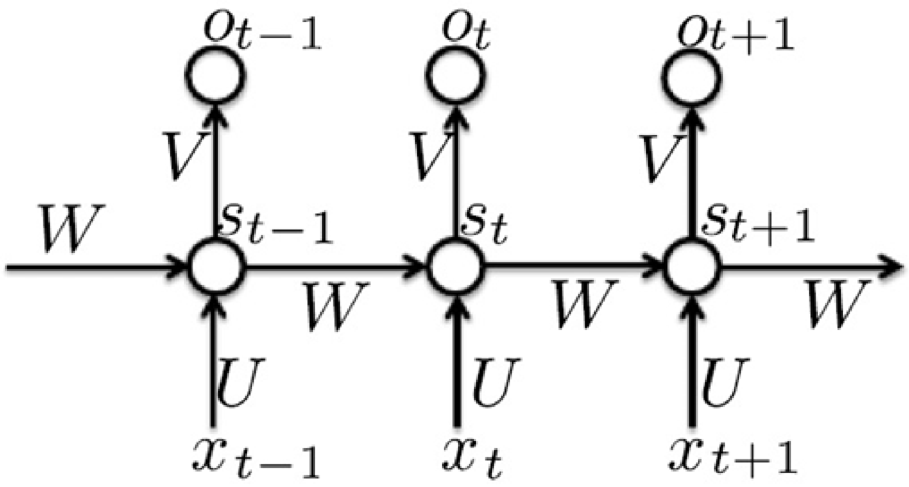
$$\frac{\partial L}{\partial S_t} = \frac{\partial L}{\partial O_t} \cdot \frac{\partial O_t}{\partial S_t} + \frac{\partial L}{\partial S_{t+1}} \cdot \frac{\partial S_{t+1}}{\partial S_t}$$

$$\delta_t = \checkmark . V + \delta_{t+1} . W \overbrace{\left(1 - \underbrace{tanh^2(b + WS_{t-1} + Ux_t)}_{S_t^2}\right)}^{diag(1 - S_t^2)}$$


o_{t-1}
o_t
o_{t+1}
V
V
V
W
s_{t-1}
s_t
s_{t+1}
W
W
W
U
U
U
x_{t-1}
x_t
x_{t+1}

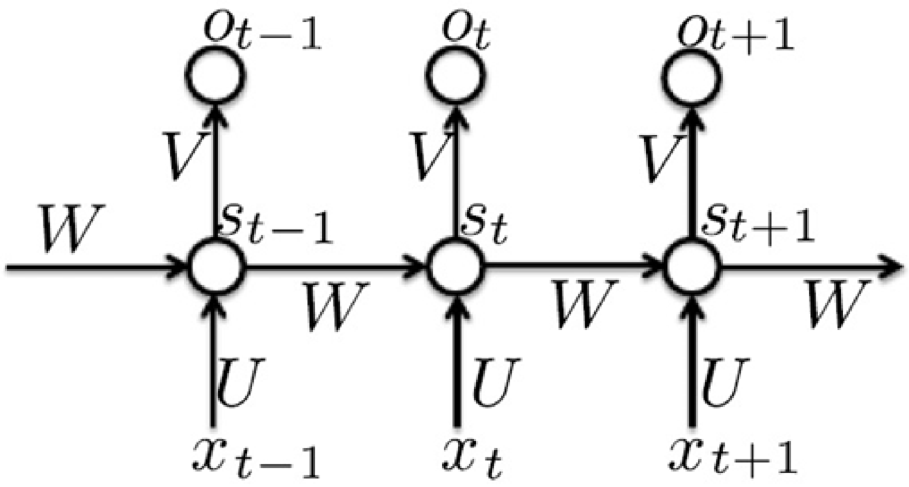


$$\frac{\partial L}{\partial V} = \sum_t \frac{\partial L}{\partial O_t} \cdot \frac{\partial O_t}{\partial V}$$



$$\frac{\partial L}{\partial V} = \sum_t \frac{\partial L}{\partial O_t} \cdot \frac{\partial O_t}{\partial V}$$

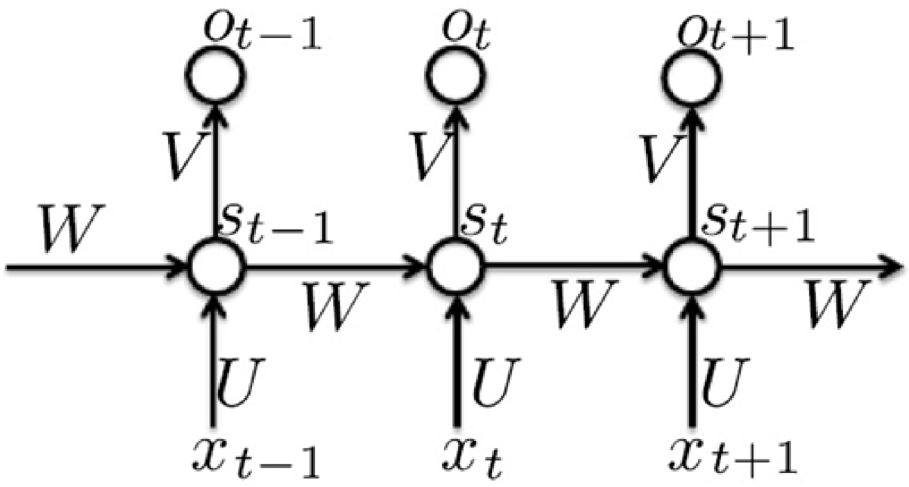
$$= \sum_t \checkmark S_t$$



$$\frac{\partial L}{\partial V} = \sum_t \frac{\partial L}{\partial O_t} . \frac{\partial O_t}{\partial V}$$

$$= \sum_t \checkmark S_t$$

$$\frac{\partial L}{\partial W} = \sum_t \frac{\partial L}{\partial S_t} . \frac{\partial S_t}{\partial W}$$



$$\frac{\partial L}{\partial V} = \sum_t \frac{\partial L}{\partial O_t} . \frac{\partial O_t}{\partial V}$$

$$= \sum_t \checkmark S_t$$

$$\frac{\partial L}{\partial W} = \sum_t \frac{\partial L}{\partial S_t} . \frac{\partial S_t}{\partial W}$$

$$= \sum_t \checkmark diag(1 - S^2) S_{t-1}$$

Facing the challenge

Gradients propagated over many stages tend to either vanish (most of the time) or explode.

Exploding or Vanishing Product of Jacobians

In recurrent nets (also in very deep nets), the final output is the composition of a large number of non-linear transformations.

Even if each of these non-linear transformations is smooth. Their composition might not be.

The derivatives through the whole composition will tend to be either very small or very large.

Exploding or Vanishing Product of Jacobians

The Jacobian (matrix of derivatives) of a composition is the product of the Jacobians of each stage.
If

$$f = f_T \circ f_{T-1} \circ \ldots, f_2 \circ f_1$$

where $(f \circ g)(x) = f\,(g(x))$
$(f \circ g)'(x) = (f' \circ g)(x) \cdot g'(x) = f'(g(x))g'(x)$

Exploding or Vanishing Product of Jacobians

The Jacobian matrix of $f(\mathbf{x})$ derivatives of with respect to its input vector $\mathbf{x}$ is

$$f' = f'_T f'_{T-1} \ldots, f'_2 f_1$$

where

$$f' = \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}$$

and

$$f'_t = \frac{\partial f_t(\mathbf{a}_t)}{\partial \mathbf{a}_t},$$

where $a_t = f_{t-1}(f_{t-2}(\ldots, f_2(f_1(\mathbf{x})))).$

Exploding or Vanishing Product of Jacobians

Simple example

Suppose: all the numbers in the product are scalar and have the same value α.

multiplying many numbers together tends to be either very large or very small.

If T goes to ∞, then

α^T goes to ∞ if $\alpha > 1$

α^T goes to 0 if $\alpha < 1$

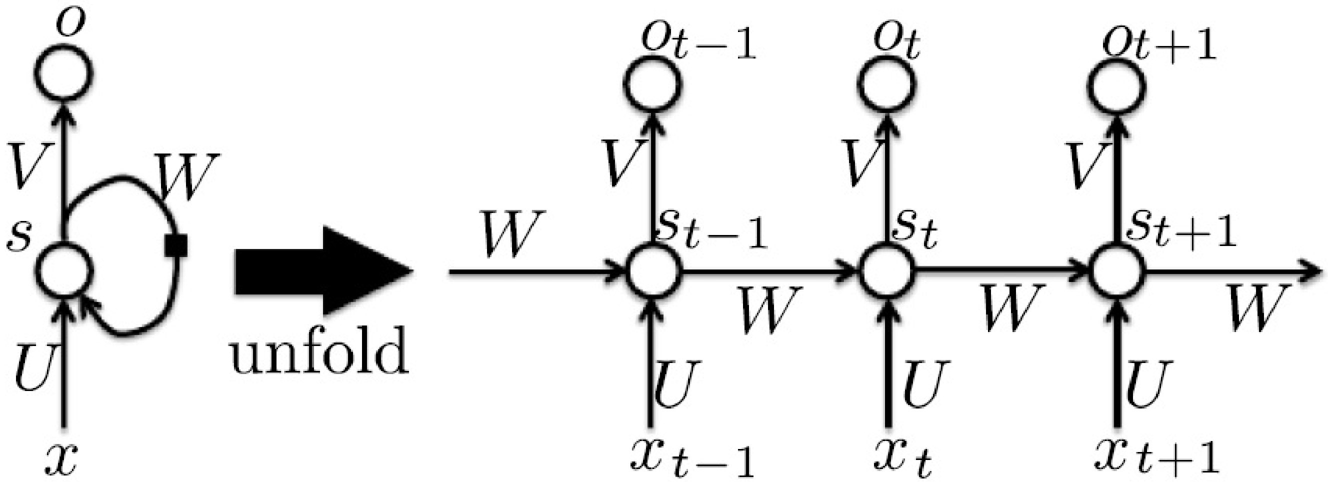
Facing the challenge

Gradients propagated over many stages tend to either vanish (most of the time) or explode.

Echo State Networks

set the recurrent and input weights such that the recurrent hidden units do a good job of capturing the history of past inputs, and only learn the output weights.

$$\mathbf{s}_t = \sigma(W\mathbf{s}_{t-1} + U\mathbf{x}_t)$$



Echo State Networks

If a change Δs in the state at t is aligned with an eigenvector v of jacobian J with eigenvalue $\lambda > 1$, then the small change Δs becomes $\lambda \Delta s$ after one time step, and $\lambda^t \Delta s$ after t time steps.

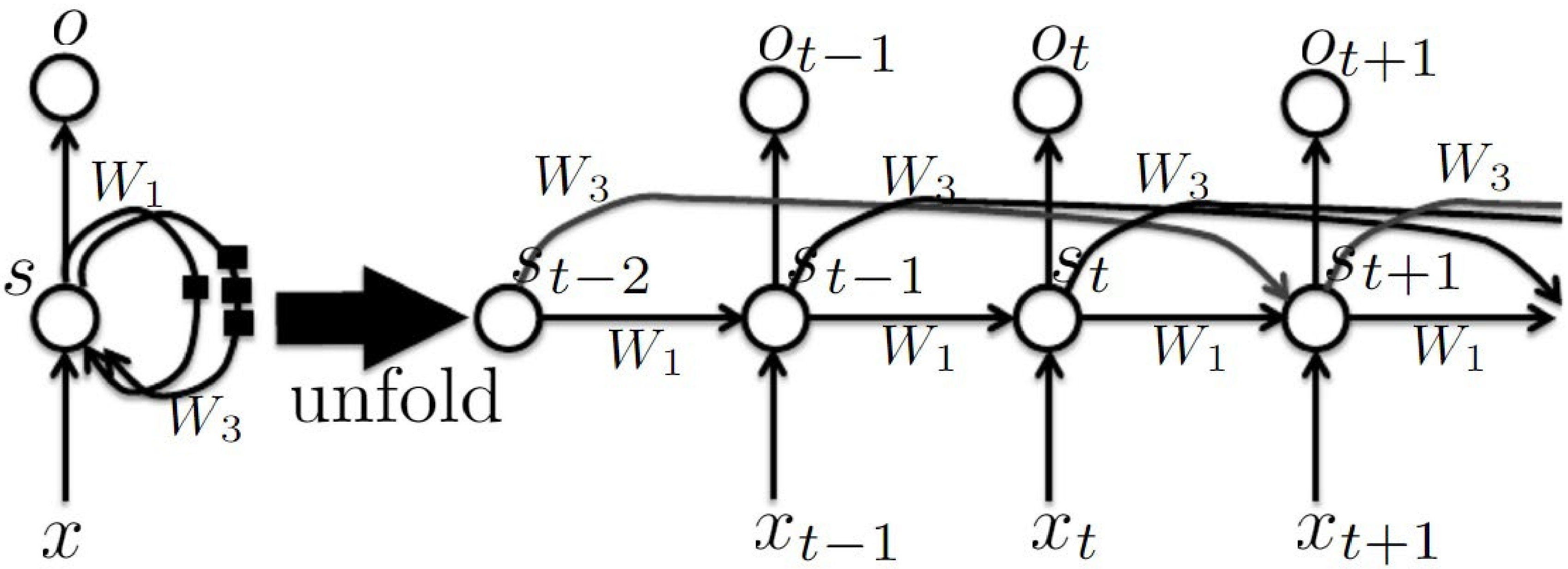
If the largest eigenvalue$\lambda < 1$, the map from t to $t + 1$ is contractive.

The network forgetting information about the long-term past.

Set the weights to make the Jacobians *slightly contractive*.

Long delays

Use recurrent connections with long delays.



Leaky Units

Recall that

$$\mathbf{s}_t = \sigma(W\mathbf{s}_{t-1} + U\mathbf{x}_t)$$

Consider

$$\mathbf{s}_{t,i} = (1 - \frac{1}{\tau_i})\mathbf{s}_{t-1} + \frac{1}{\tau_i}\sigma(W\mathbf{s}_{t-1} + U\mathbf{x}_t)$$

$1 \leq \tau_i \leq \infty$

$\tau_i = 1$, Ordinary RNN

$\tau_i > 1$, gradients propagate more easily.

$\tau_i >> 1$, the state changes very slowly, integrating the past values associated with the input sequence.

Gated RNNs

It might be useful for the neural network to forget the old state in some cases.

Example: $a\ a\ b\ b\ b\ a\ a\ a\ a\ b\ a\ b$

It might be useful to keep the memory of the past.

Example:

Instead of manually deciding when to clear the state, we want the neural network to learn to decide when to do it.

Gated RNNs, the Long-Short-Term-Memory

The Long-Short-Term-Memory (LSTM) algorithm was proposed in 1997 (Hochreiter and Schmidhuber, 1997).

Several variants of the LSTM are found in the literature:

Hochreiter and Schmidhuber 1997

Graves, 2012

Graves et al., 2013

Sutskever et al., 2014

the principle is always to have a linear self-loop through which gradients can flow for long duration.

Gated Recurrent Units (GRU)

Recent work on gated RNNs, Gated Recurrent Units (GRU) was proposed in 2014

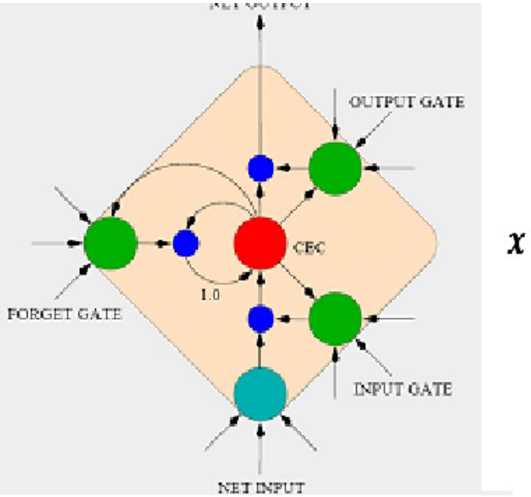
Cho et al., 2014

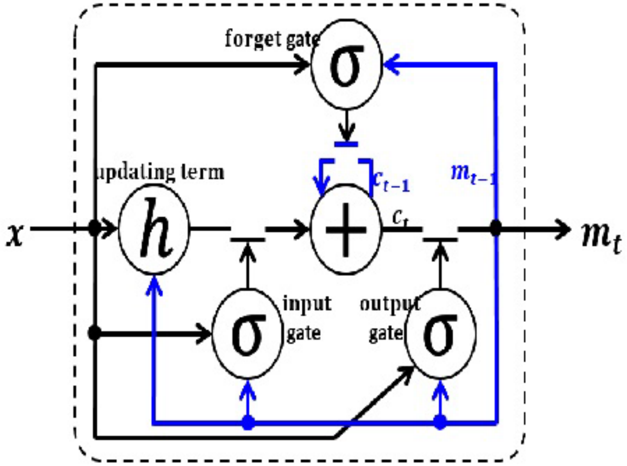
Chung et al., 2014, 2015

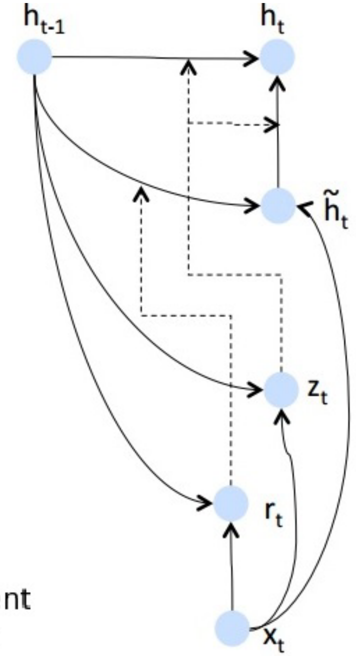
Jozefowicz et al., 2015

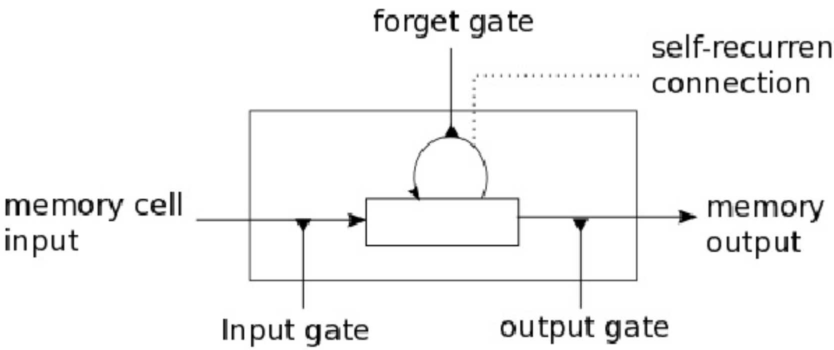
Chrupala et al., 2015

Gated RNNs


NET OUTPUT
OUTPUT GATE
CEC
1.0
FORGET GATE
INPUT GATE
NET INPUT


forget gate
σ
updating term
c_{t-1}
m_{t-1}
x
h
+
c_t
m_t
σ
input gate
output gate
σ


h_{t-1}
h_t
$\tilde{h}_t$
z_t
r_t
x_t


forget gate
self-recurrent connection
memory cell input
memory cell output
Input gate
output gate

Gated Recurrent Units (GRU)

Standard RNN computes hidden layer at next time step directly:

$$S_t = \sigma(Ws_{t-1} + Ux_t)$$

GRU first computes an update Gate (another layer) based on current input vector and hidden state

$$z_t = \sigma(U^{(z)}x_t + W^{(z)}s_{t-1})$$

compute reset gate similarly but with different weights

$$r_t = \sigma(U^{(r)}x_t + W^{(r)}s_{t-1})$$

Gated Recurrent Units (GRU)

Update gate: $z_t = \sigma(U^{(z)}x_t + W^{(z)}s_{t-1})$
Reset gate: $r_t = \sigma(U^{(r)}x_t + W^{(r)}s_{t-1})$
New memory content :

$$\tilde{s}_t = tanh(Ux_t + r_t \circ Ws_{t-1})$$

if reset gate is 0, then this ignores previous memory and only stores the new information
Final memory at time step combines current and previous time steps :

$$s_t = z_t \circ s_{t-1} + (1 - z_t) \circ \tilde{s}_t$$

Update gate:

$$z_t = \sigma(U^{(z)}x_t + W^{(z)}s_{t-1})$$

Reset gate:

$$r_t = \sigma(U^{(r)}x_t + W^{(r)}s_{t-1})$$

New memory content:

$$\tilde{s}_t = tanh(Ux_t + r_t \circ Ws_{t-1})$$

$$s_t = z_t \circ s_{t-1} + (1 - z_t) \circ \tilde{s}_t$$

Gated Recurrent Units (GRU)

$$
\begin{aligned}
z_t &= \sigma(U^{(z)}x_t + W^{(z)}s_{t-1}) \\
r_t &= \sigma(U^{(r)}x_t + W^{(r)}s_{t-1}) \\
\tilde{s}_t &= tanh(Ux_t + r_t \circ Ws_{t-1}) \\
s_t &= z_t \circ s_{t-1} + (1 - z_t) \circ \tilde{s}_t
\end{aligned}
$$

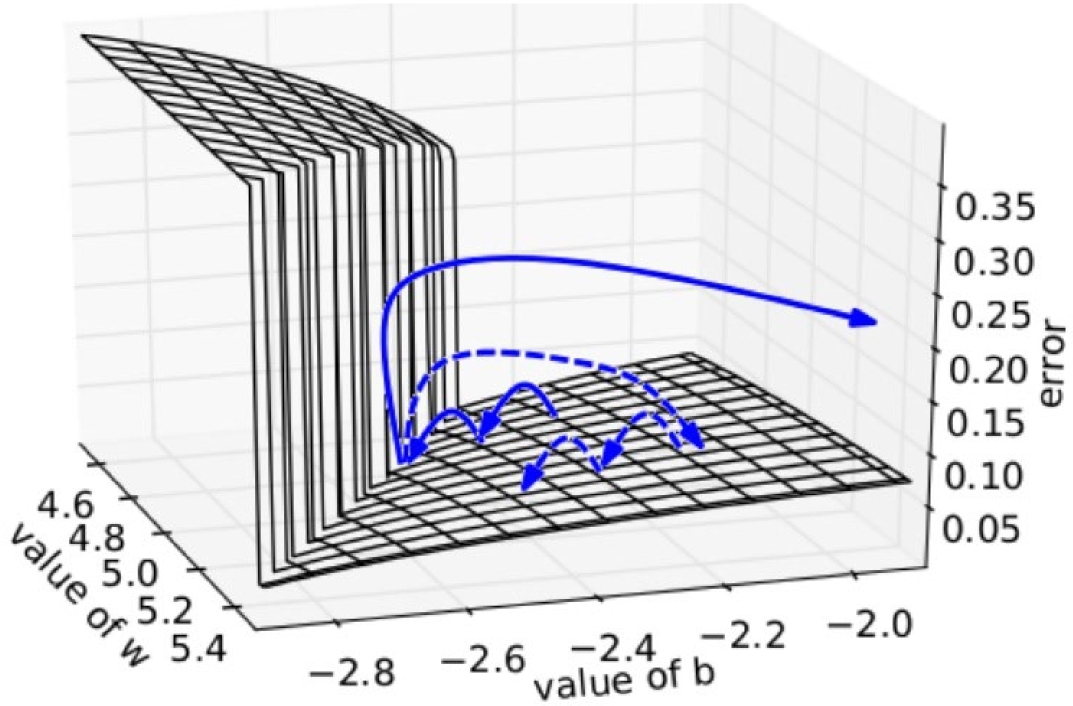
If reset is close to 0, ignore previous hidden state → Allow model to drop information that is irrelevant

Update gate z controls how much of past state should matter now.

If z close to 1, then we can copy information in that unit through many time steps.

Units with short term dependencies often have reset gates very active.

Clipping Gradients



Strongly non-linear functions tend to have derivatives that can be either very large or very small in magnitude.

Clipping Gradients

Simple solution for clipping the gradient. (Mikolov, 2012; Pascanu et al., 2013):
Clip the parameter gradient from a mini batch element-wise (Mikolov, 2012) just before the parameter update.
Clip the norm g of the gradient g (Pascanu et al., 2013a) just before the parameter update.

$$g' = \min\left(1, \frac{c}{\|g\|}\right) g$$