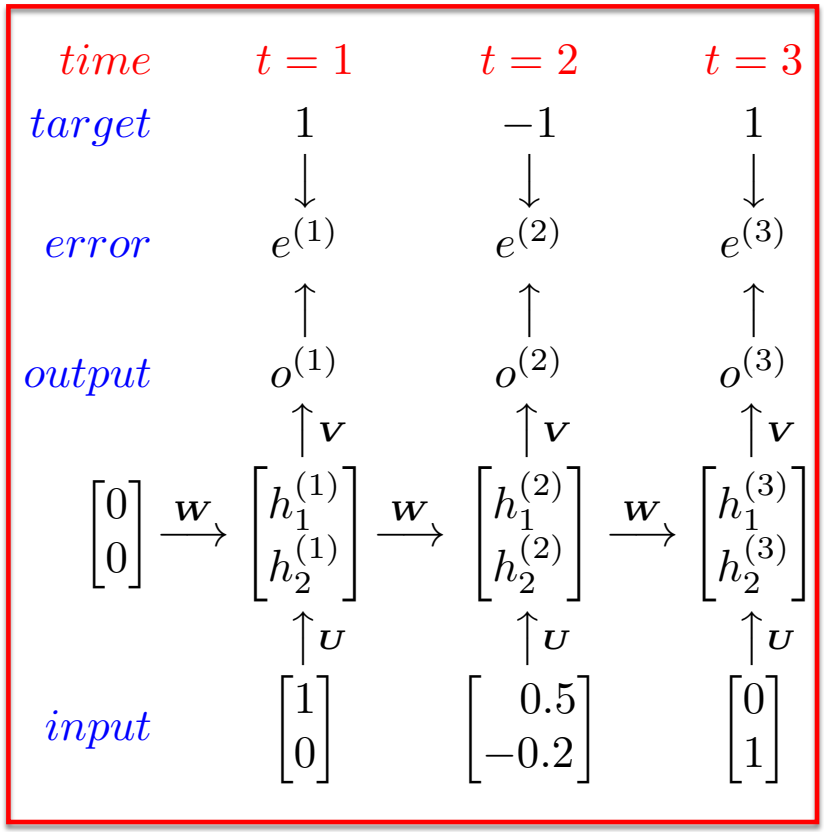
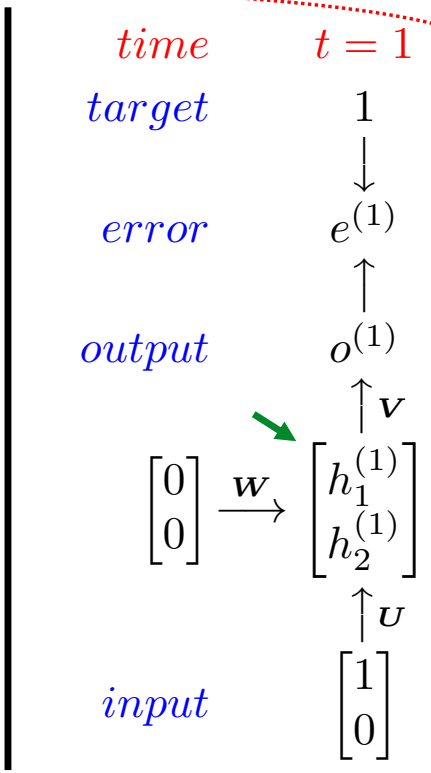
Real-Time Recurrent Learning (RTRL)





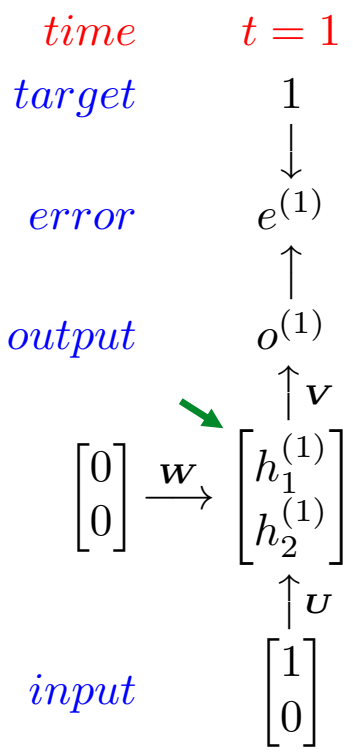
Online Learning:
cannot wait to unroll the recursive network

Solution:
compute the gradients during the forward phase

Real-Time Recurrent Learning (RTRL)

Equations

$$
\begin{aligned}
\boldsymbol{h}^{(0)} &= \mathbf{0}, \\
\boldsymbol{a}_h{}^{(t)} &= \boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}, \\
\boldsymbol{h}^{(t)} &= tanh(\boldsymbol{a}_h{}^{(t)}), \\
a_o^{(t)} &= \boldsymbol{V}\boldsymbol{h}^{(t)} + c, \\
o^{(t)} &= tanh(a_o^{(t)}), \\
L &= \textstyle\sum_t e^{(t)} = \sum_t (y^{(t)} - o^{(t)})^2
\end{aligned}
$$



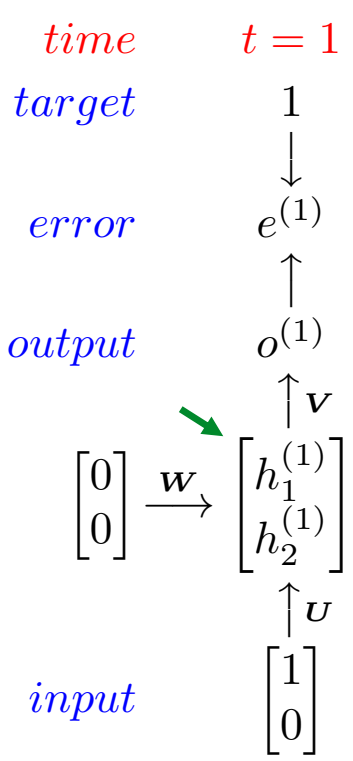
Online Learning:
cannot wait to unroll the recursive network

Solution:
compute the gradients during the forward phase

Real-Time Recurrent Learning (RTRL)

Equations

$$
\begin{aligned}
\boldsymbol{h}^{(0)} &= \mathbf{0}, \\
{\boldsymbol{a}_h}^{(t)} &= \boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}, \\
\boldsymbol{h}^{(t)} &= tanh({\boldsymbol{a}_h}^{(t)}), \\
a_o^{(t)} &= \boldsymbol{V}\boldsymbol{h}^{(t)} + c, \\
o^{(t)} &= tanh(a_o^{(t)}), \\
L &= \textstyle\sum_t e^{(t)} = \sum_t (y^{(t)} - o^{(t)})^2
\end{aligned}
$$



Online Learning:
cannot wait to unroll the recursive network

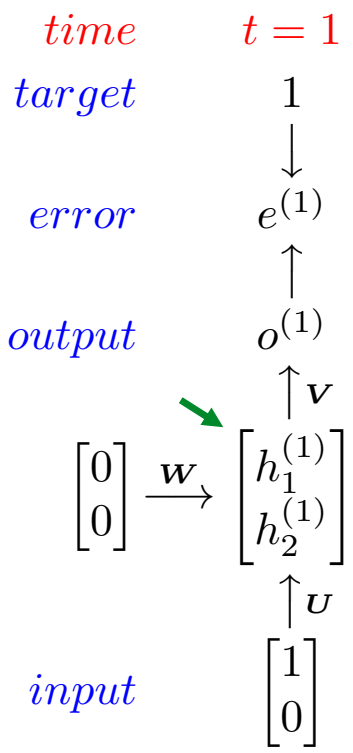
Solution:
compute the gradients during the forward phase

here it is also possibile to compute $\frac{\partial \boldsymbol{h}^{(1)}}{\partial \boldsymbol{U}}$

Real-Time Recurrent Learning (RTRL)

Equations

$$
\begin{aligned}
\boldsymbol{h}^{(0)} &= \mathbf{0}, \\
\boldsymbol{a}_h{}^{(t)} &= \boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}, \\
\boldsymbol{h}^{(t)} &= tanh(\boldsymbol{a}_h{}^{(t)}), \\
a_o^{(t)} &= \boldsymbol{V}\boldsymbol{h}^{(t)} + c, \\
o^{(t)} &= tanh(a_o^{(t)}), \\
L &= \textstyle\sum_t e^{(t)} = \sum_t (y^{(t)} - o^{(t)})^2
\end{aligned}
$$



Online Learning:
cannot wait to unroll the recursive network

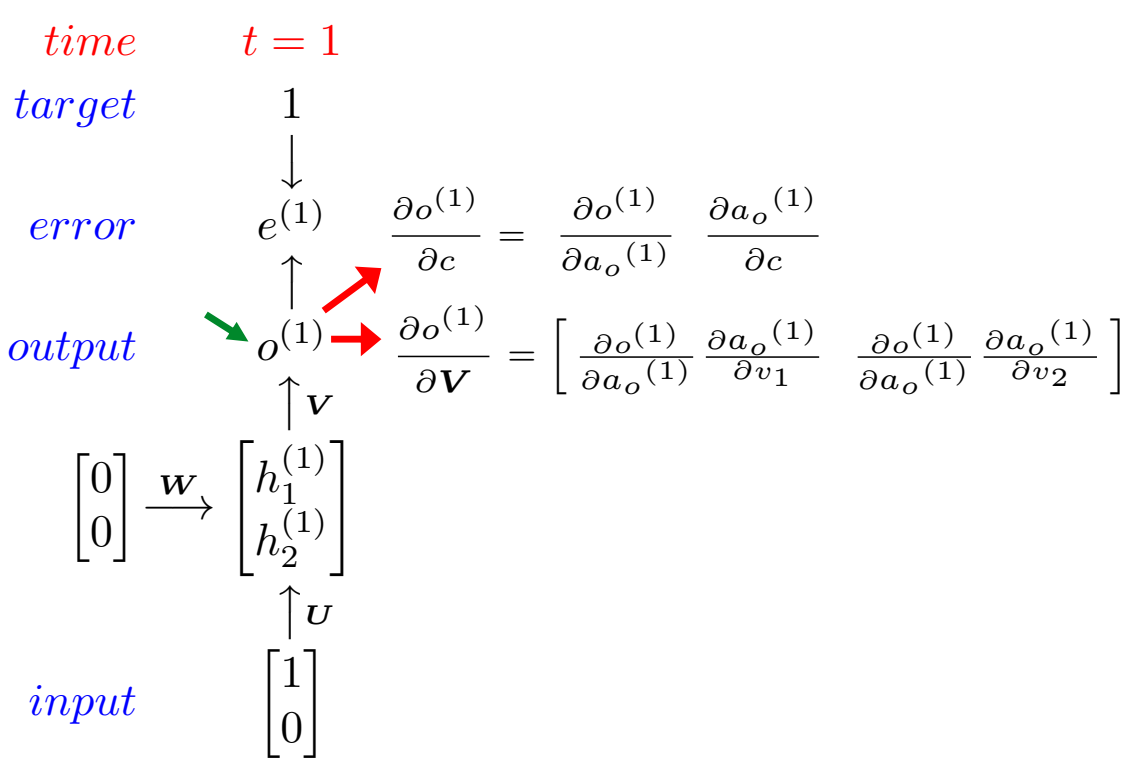
Solution:
compute the gradients during the forward phase

... and $\frac{\partial \boldsymbol{h}^{(1)}}{\partial \boldsymbol{b}}$, which is a tensor in $\mathbb{R}^{2\times 2}$

Real-Time Recurrent Learning (RTRL)

Equations

$$
\begin{aligned}
\boldsymbol{h}^{(0)} &= \mathbf{0}, \\
\boldsymbol{a}_h{}^{(t)} &= \boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}, \\
\boldsymbol{h}^{(t)} &= tanh(\boldsymbol{a}_h{}^{(t)}), \\
a_o^{(t)} &= \boldsymbol{V}\boldsymbol{h}^{(t)} + c, \\
o^{(t)} &= tanh(a_o^{(t)}), \\
L &= \textstyle\sum_t e^{(t)} = \sum_t (y^{(t)} - o^{(t)})^2
\end{aligned}
$$



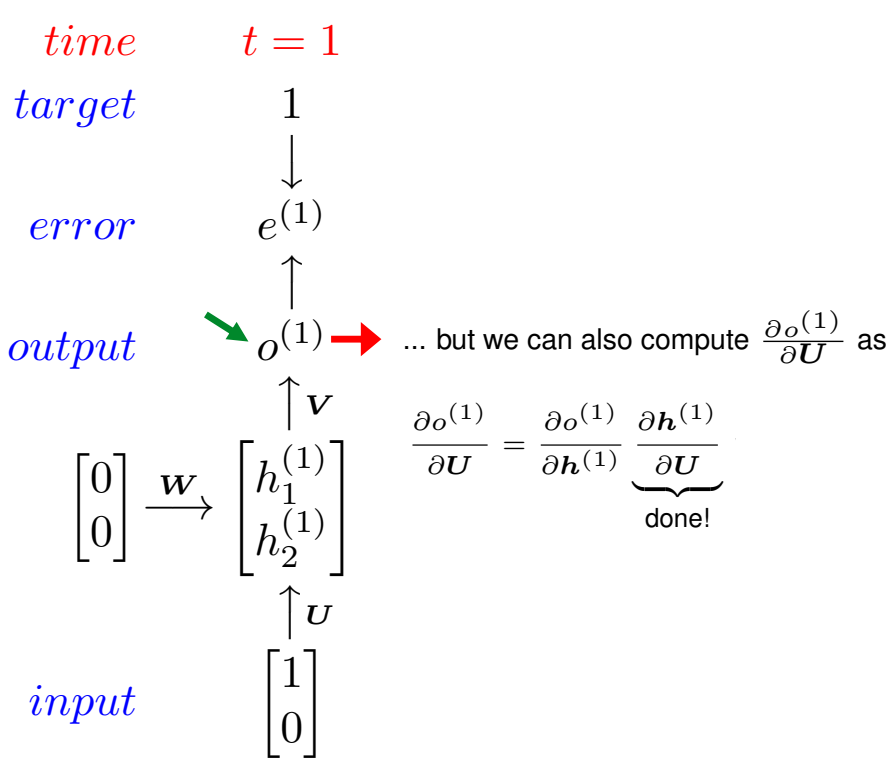
$$\frac{\partial o^{(1)}}{\partial c} = \frac{\partial o^{(1)}}{\partial a_o{}^{(1)}} \frac{\partial a_o{}^{(1)}}{\partial c}$$

$$\frac{\partial o^{(1)}}{\partial \boldsymbol{V}} = \left[\frac{\partial o^{(1)}}{\partial a_o{}^{(1)}} \frac{\partial a_o{}^{(1)}}{\partial v_1} \quad \frac{\partial o^{(1)}}{\partial a_o{}^{(1)}} \frac{\partial a_o{}^{(1)}}{\partial v_2} \right]$$

Real-Time Recurrent Learning (RTRL)

Equations

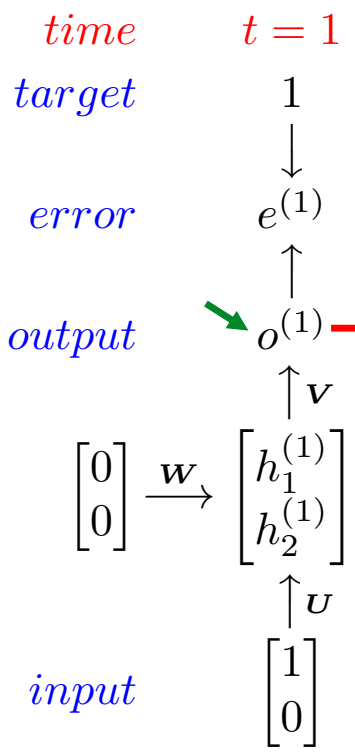
$$
\begin{aligned}
\boldsymbol{h}^{(0)} &= \mathbf{0}, \\
\boldsymbol{a}_h{}^{(t)} &= \boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}, \\
\boldsymbol{h}^{(t)} &= tanh(\boldsymbol{a}_h{}^{(t)}), \\
a_o^{(t)} &= \boldsymbol{V}\boldsymbol{h}^{(t)} + c, \\
o^{(t)} &= tanh(a_o^{(t)}), \\
L &= \textstyle\sum_t e^{(t)} = \sum_t (y^{(t)} - o^{(t)})^2
\end{aligned}
$$



Real-Time Recurrent Learning (RTRL)

Equations

$$
\begin{aligned}
\boldsymbol{h}^{(0)} &= \mathbf{0}, \\
\boldsymbol{a}_h{}^{(t)} &= \boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}, \\
\boldsymbol{h}^{(t)} &= tanh(\boldsymbol{a}_h{}^{(t)}), \\
a_o^{(t)} &= \boldsymbol{V}\boldsymbol{h}^{(t)} + c, \\
o^{(t)} &= tanh(a_o^{(t)}), \\
L &= \textstyle\sum_t e^{(t)} = \sum_t (y^{(t)} - o^{(t)})^2
\end{aligned}
$$

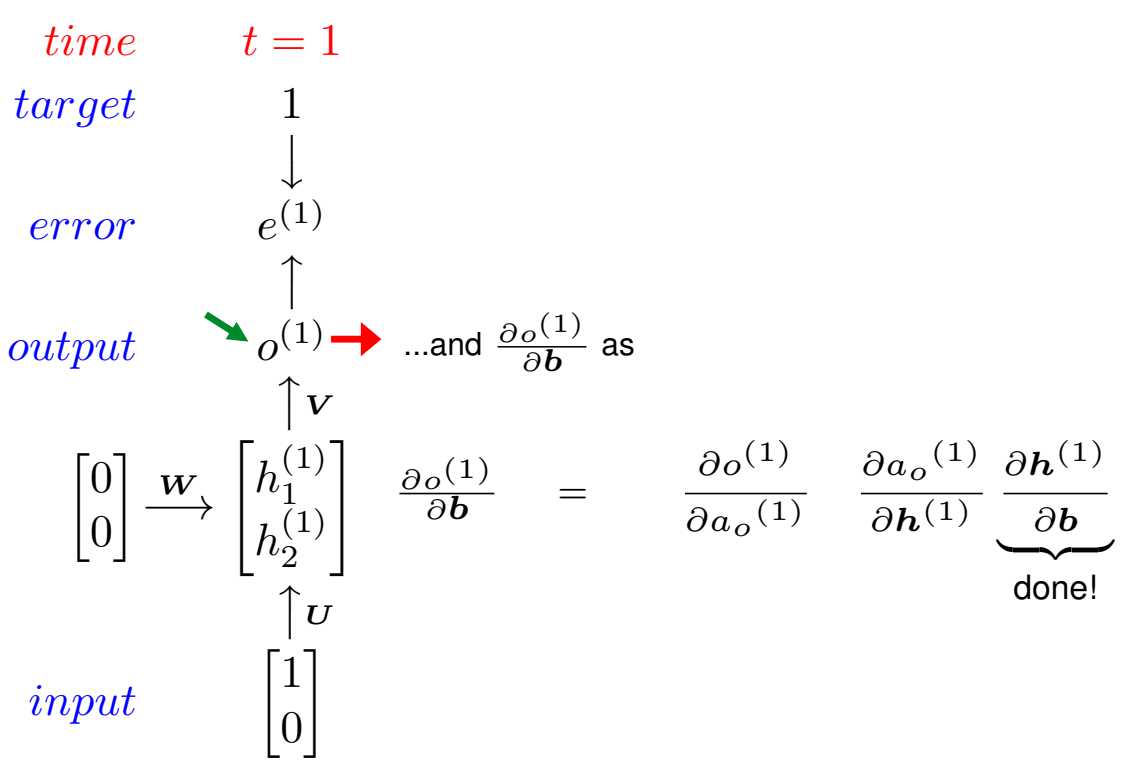


...and we already know that $\frac{\partial o^{(1)}}{\partial \boldsymbol{W}}$ is a zero tensor!

Real-Time Recurrent Learning (RTRL)

Equations

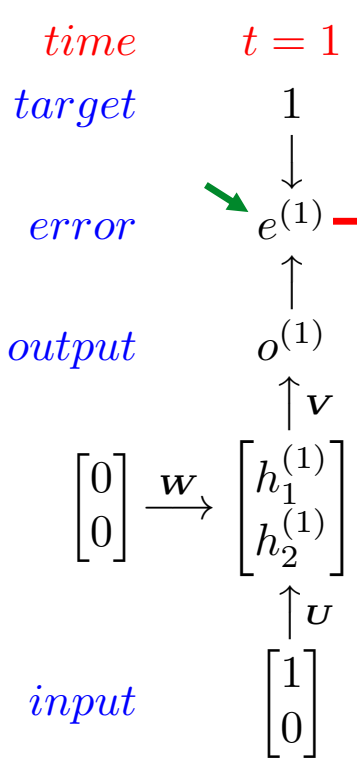
$$
\begin{aligned}
\boldsymbol{h}^{(0)} &= \mathbf{0}, \\
{\boldsymbol{a}_h}^{(t)} &= \boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}, \\
\boldsymbol{h}^{(t)} &= tanh({\boldsymbol{a}_h}^{(t)}), \\
a_o^{(t)} &= \boldsymbol{V}\boldsymbol{h}^{(t)} + c, \\
o^{(t)} &= tanh(a_o^{(t)}), \\
L &= \textstyle\sum_t e^{(t)} = \sum_t (y^{(t)} - o^{(t)})^2
\end{aligned}
$$



Real-Time Recurrent Learning (RTRL)

Equations

$$
\begin{aligned}
\boldsymbol{h}^{(0)} &= \mathbf{0}, \\
\boldsymbol{a}_h{}^{(t)} &= \boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}, \\
\boldsymbol{h}^{(t)} &= tanh(\boldsymbol{a}_h{}^{(t)}), \\
a_o^{(t)} &= \boldsymbol{V}\boldsymbol{h}^{(t)} + c, \\
o^{(t)} &= tanh(a_o^{(t)}), \\
L &= \textstyle\sum_t e^{(t)} = \sum_t (y^{(t)} - o^{(t)})^2
\end{aligned}
$$

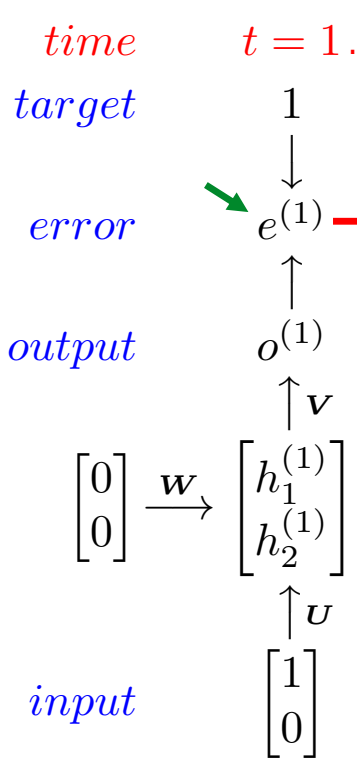


$$
\begin{aligned}
\frac{\partial e^{(1)}}{\partial o^{(1)}} &= -2(1 - o^{(1)}) \\
\frac{\partial e^{(1)}}{\partial \boldsymbol{V}} &= \frac{\partial e^{(1)}}{\partial o^{(1)}} \frac{\partial o^{(1)}}{\partial \boldsymbol{V}} \leftarrow \text{done!} \\
\frac{\partial e^{(1)}}{\partial c} &= \frac{\partial e^{(1)}}{\partial o^{(1)}} \frac{\partial o^{(1)}}{\partial c} \leftarrow \text{done!} \\
\frac{\partial e^{(1)}}{\partial \boldsymbol{W}} &= \frac{\partial e^{(1)}}{\partial o^{(1)}} \frac{\partial o^{(1)}}{\partial \boldsymbol{W}} \leftarrow \text{done!} \\
\frac{\partial e^{(1)}}{\partial \boldsymbol{U}} &= \frac{\partial e^{(1)}}{\partial o^{(1)}} \frac{\partial o^{(1)}}{\partial \boldsymbol{U}} \leftarrow \text{done!} \\
\frac{\partial e^{(1)}}{\partial \boldsymbol{b}} &= \frac{\partial e^{(1)}}{\partial o^{(1)}} \frac{\partial o^{(1)}}{\partial \boldsymbol{b}} \leftarrow \text{done!}
\end{aligned}
$$

Real-Time Recurrent Learning (RTRL)

Equations

$$
\begin{aligned}
\boldsymbol{h}^{(0)} &= \mathbf{0}, \\
\boldsymbol{a}_h{}^{(t)} &= \boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}, \\
\boldsymbol{h}^{(t)} &= tanh(\boldsymbol{a}_h{}^{(t)}), \\
a_o^{(t)} &= \boldsymbol{V}\boldsymbol{h}^{(t)} + c, \\
o^{(t)} &= tanh(a_o^{(t)}), \\
L &= \textstyle\sum_t e^{(t)} = \sum_t (y^{(t)} - o^{(t)})^2
\end{aligned}
$$



$$
\begin{aligned}
\frac{\partial e^{(1)}}{\partial o^{(1)}} &= -2(1 - o^{(1)}) \\
\frac{\partial e^{(1)}}{\partial \boldsymbol{V}} &= \frac{\partial e^{(1)}}{\partial o^{(1)}} \frac{\partial o^{(1)}}{\partial \boldsymbol{V}} \leftarrow \text{done!} \\
\frac{\partial e^{(1)}}{\partial c} &= \frac{\partial e^{(1)}}{\partial o^{(1)}} \frac{\partial o^{(1)}}{\partial c} \leftarrow \text{done!} \\
\frac{\partial e^{(1)}}{\partial \boldsymbol{W}} &= \frac{\partial e^{(1)}}{\partial o^{(1)}} \frac{\partial o^{(1)}}{\partial \boldsymbol{W}} \leftarrow \text{done!} \\
\frac{\partial e^{(1)}}{\partial \boldsymbol{U}} &= \frac{\partial e^{(1)}}{\partial o^{(1)}} \frac{\partial o^{(1)}}{\partial \boldsymbol{U}} \leftarrow \text{done!} \\
\frac{\partial e^{(1)}}{\partial \boldsymbol{b}} &= \frac{\partial e^{(1)}}{\partial o^{(1)}} \frac{\partial o^{(1)}}{\partial \boldsymbol{b}} \leftarrow \text{done!}
\end{aligned}
$$

BPTT vs RTRL

- Both **BPTT** and **RTRL** compute the same quantities (gradients), but in different ways

- Computational complexity

	Space	*Time*
BPTT	$O(NT)$	$O(N^2T)$
RTRL	$O(N^3)$	$O(N^4)$

T: time steps
N: number of units

- Whatever the computation is performed, there is a serious problem...vanishing/exploding of gradient!

Gradient Vanishing/Exploding Problem
how to cope with it
(part 1)

Learning Long-term Dependencies is difficult

What is a long-term dependency ?

- when the *desidered* output at time t depends on the input at time $t - \tau$, with $t > \tau \gg 1$

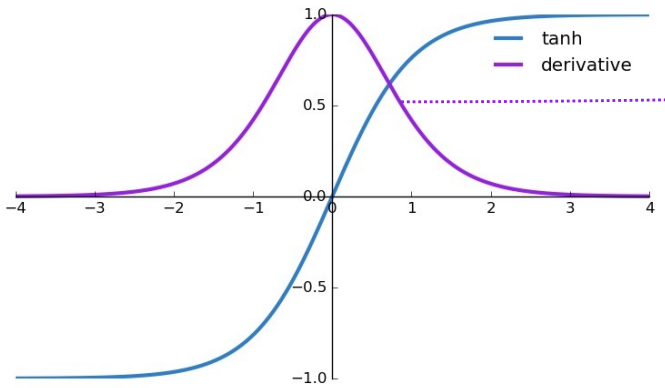
$$\boldsymbol{x}^{(t-\tau)} \rightarrow \boldsymbol{y}^{(t)}$$

e.g.

$$\boldsymbol{x}^{(t-100)} \rightarrow \boldsymbol{y}^{(t)}$$

- this means that, for the Recurrent Neural Network to output the correct *desidered* $\boldsymbol{y}^{(t)}$, it has:
  - *to recognize* its dependency on $\boldsymbol{x}^{(t-\tau)}$, and
  - *to use* $\boldsymbol{x}^{(t-\tau)}$ in the generation of $\boldsymbol{y}^{(t)}$

Learning Long-term Dependencies is difficult



$$\begin{bmatrix} \frac{\partial h_1^{(t)}}{\partial \boldsymbol{W}} \\ \frac{\partial h_2^{(t)}}{\partial \boldsymbol{W}} \end{bmatrix} = \begin{matrix} \underline{(1-(h_1^{(t)})^2)}\left(\begin{bmatrix} h_1^{(t-1)} & h_2^{(t-1)} \\ 0 & 0 \end{bmatrix} + w_{11}\frac{\partial h_1^{(t-1)}}{\partial \boldsymbol{W}} + w_{12}\frac{\partial h_2^{(t-1)}}{\partial \boldsymbol{W}} \right) \\ \underline{(1-(h_2^{(t)})^2)}\left(\begin{bmatrix} 0 & 0 \\ h_1^{(t-1)} & h_2^{(t-1)} \end{bmatrix} + w_{21}\frac{\partial h_1^{(t-1)}}{\partial \boldsymbol{W}} + w_{22}\frac{\partial h_2^{(t-1)}}{\partial \boldsymbol{W}} \right) \end{matrix}$$

t=3 $\Rightarrow$

$$\frac{\partial h_1^{(3)}}{\partial \boldsymbol{W}} = \underline{(1-(h_1^{(3)})^2)}\left(\begin{bmatrix} h_1^{(2)} & h_2^{(2)} \\ 0 & 0 \end{bmatrix} + w_{11}\underline{(1-(h_1^{(2)})^2)} \begin{bmatrix} h_1^{(1)} & h_2^{(1)} \\ 0 & 0 \end{bmatrix} + w_{12}\underline{(1-(h_2^{(2)})^2)} \begin{bmatrix} 0 & 0 \\ h_1^{(1)} & h_2^{(1)} \end{bmatrix} \right)$$

$$\frac{\partial h_2^{(3)}}{\partial \boldsymbol{W}} = \underline{(1-(h_2^{(3)})^2)}\left(\begin{bmatrix} 0 & 0 \\ h_1^{(2)} & h_2^{(2)} \end{bmatrix} + w_{21}\underline{(1-(h_1^{(2)})^2)} \begin{bmatrix} h_1^{(1)} & h_2^{(1)} \\ 0 & 0 \end{bmatrix} + w_{22}\underline{(1-(h_2^{(2)})^2)} \begin{bmatrix} 0 & 0 \\ h_1^{(1)} & h_2^{(1)} \end{bmatrix} \right)$$

Learning Long-term Dependencies is difficult

- if $\|\boldsymbol{W}\|$ is large enough, then $\|\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-\tau)}}\|$ will increase for increasing values of τ (*exploding gradient* $\Rightarrow$ hard to decide the "right" descent step size)
- however, in order to robustly store past information, the dynamics of the network must exhibit attractors:

spectral radius $\rho(\boldsymbol{W}) < 1$, i.e. $\boldsymbol{W}$ is *contractive*

- In the presence of attractors, *gradients vanish* going backward in time:

$$\lim_{t-\tau \to \infty} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(\tau)}} = \mathbf{0}$$

$\Rightarrow$ no learning with gradient descent...

Spectral Radius and Power Sequence of a Matrix

The *spectral radius* $\rho(\boldsymbol{W})$ of a matrix $\boldsymbol{W} \in \mathbb{C}^{n \times n}$ is defined as:

$$\rho(\boldsymbol{W}) = \max\left\{|\lambda_1|, \ldots, |\lambda_n|\right\},$$

where $\lambda_1, \ldots, \lambda_n$ are the (real or complex) eigenvalues of $\boldsymbol{W}$.

The *spectral radius* is closely related to the behaviour of the *convergence* of the power sequence of a matrix $\boldsymbol{W}$

Theorem
Let $\boldsymbol{W} \in \mathbb{C}^{n \times n}$ with spectral radius $\rho(\boldsymbol{W})$. Then $\rho(\boldsymbol{W}) < 1$ *if and only if*

$$\lim_{k \to \infty} \boldsymbol{W}^k = \mathbf{0}.$$

On the other hand, if $\rho(\boldsymbol{W}) > 1$,

$$\lim_{k \to \infty} \|\boldsymbol{W}^k\| = \infty.$$

The statement holds for any choice of matrix norm on $\mathbb{C}^{n \times n}$.

Vanishing/Exploding of Gradients: some remedies

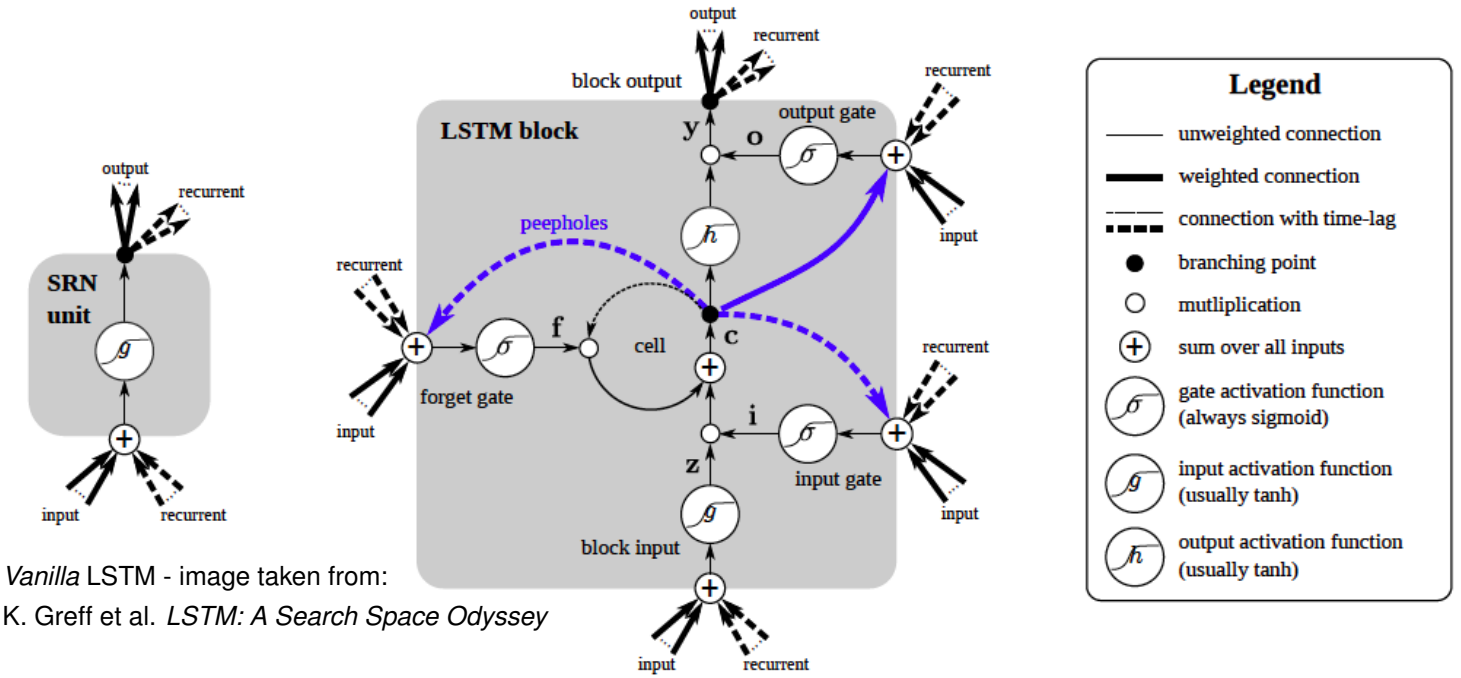
Here are some approaches to try to *reduce* the vanishing/exploding gradients problem

- Architectural
  - Long Short-Term Memory or Gated Recurrent units
  - Reservoir Computing: Echo State Networks and Liquid State Machines
- Algorithmic
  - *Clipping gradients (avoids exploding gradients)*
  - *Hessian Free Optimization*
  - Smart Initialization: pre-training techniques

Long Short-Term Memory

Extension of RNN that can deal with long-term temporal dependencies

Mechanism that allows the networks to "remember" relevant information for a long period of time



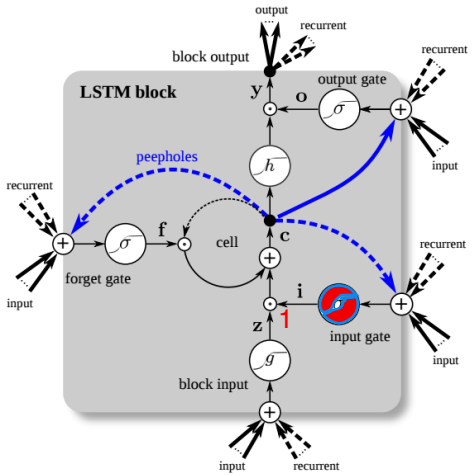
Vanilla LSTM - image taken from:
K. Greff et al. *LSTM: A Search Space Odyssey*

S. Hochreiter & J. Schmidhuber, *Neural Computation*, 1997

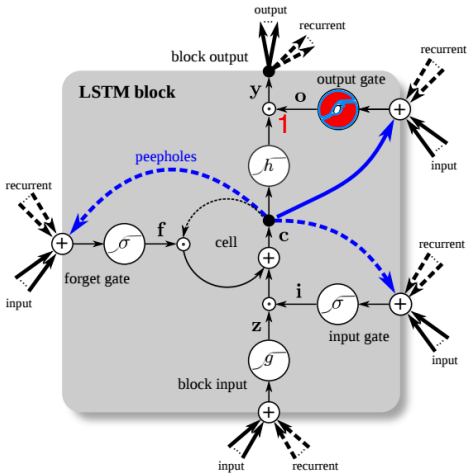
Long Short-Term Memory (*Vanilla* LSTM)

- Exploits a **linear** memory cell (*state*) that integrates input information through time
  - memory obtained by self-loop
  - gradient not down-sized by Jacobian of sigmoidal function $\Rightarrow$ no vanishing gradient
- 3 gate units (with sigmoid: soft version of a 0/1 switch) control the information flow via multiplicative connections
  - input gate "on": let input to flow in the memory cell
  - output gate "on": let the current value stored in the memory cell to be read in output
  - forget gate "off": let the current value stored in the memory cell to be reset to 0

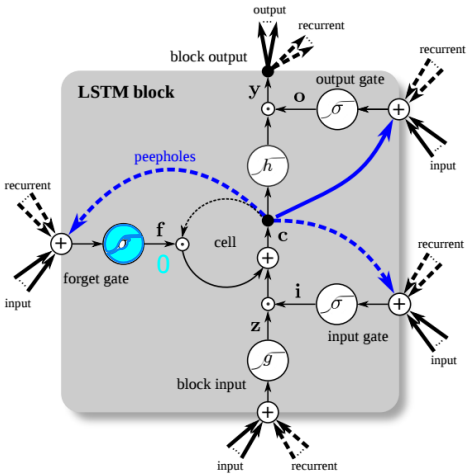
input gate on



output gate on

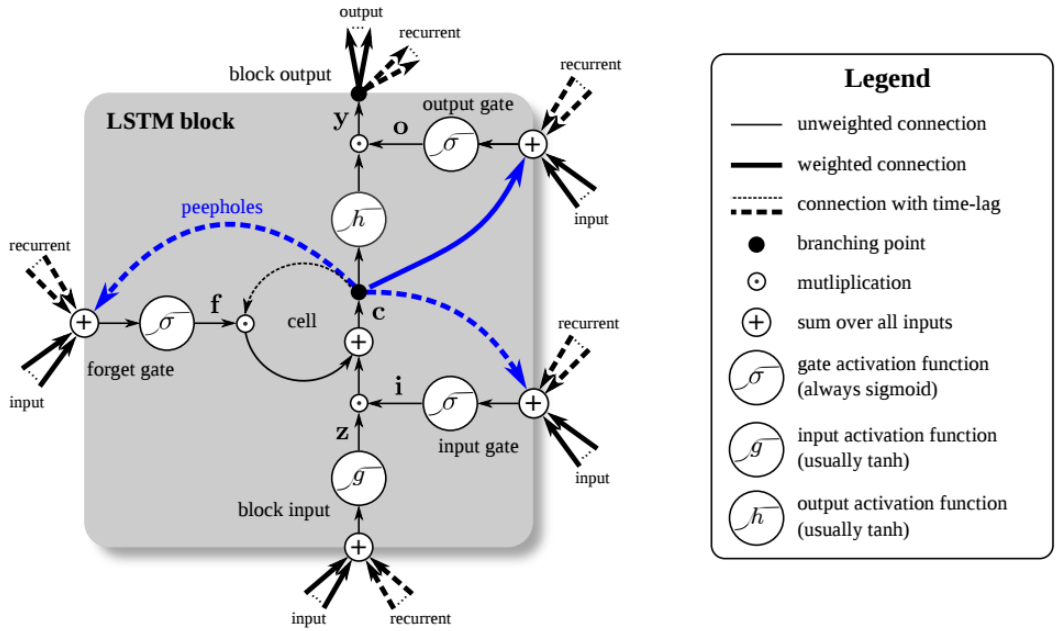


forget gate off



Long Short-Term Memory (*Vanilla* LSTM)

- peepholes connections allow to directly control all gates to allow for easier learning of precise timings
- full back-propagation through time (BPTT) training introduced only in 2005



Long Short-Term Memory

Many variations of the *Vanilla* LSTM architecture have been studied, as

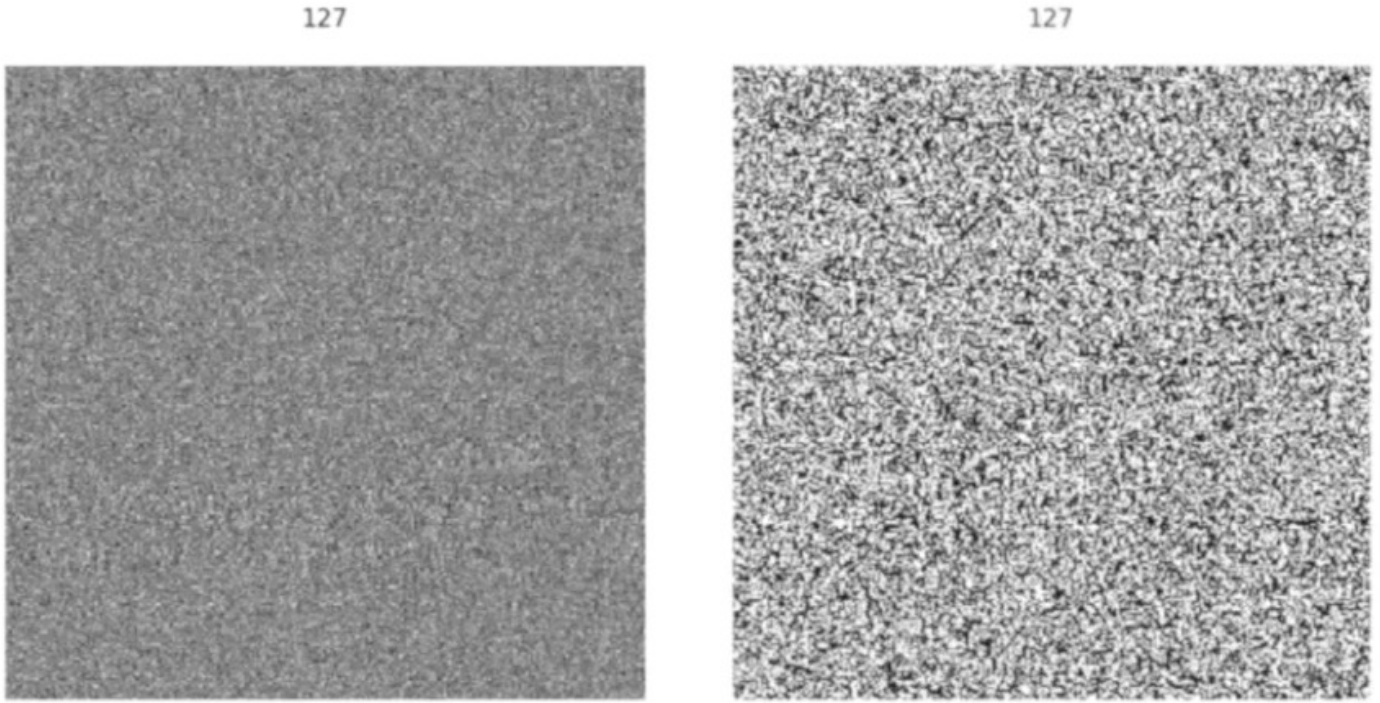
- No Input Gate (NIG)
- No Forget Gate (NFG)
- No Output Gate (NOG)
- No Input Activation Function (NIAF)
- No Output Activation Function (NOAF)
- No Peepholes (NP)

However:

- *Vanilla* LSTM performs reasonably well in general and variations do not significantly improve the performance
- the forget gate is crucial for LSTM performance
- if the cell state is unbounded the output activation function is needed

Long Short-Term Memory

gradient as an image



Simplifying LSTM: Gated Recurrent Units

Is it possibile to simplify LSTM units ?

- Gated Recurrent Units (GRU) do that by using a single gating unit that simultaneously controls the forgetting factor and the decision to update the state unit

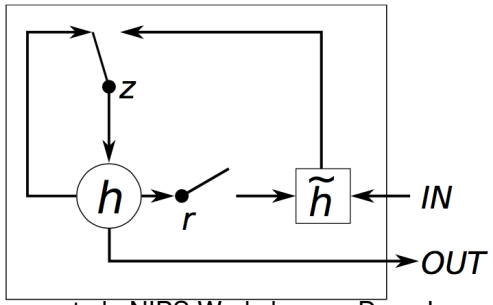


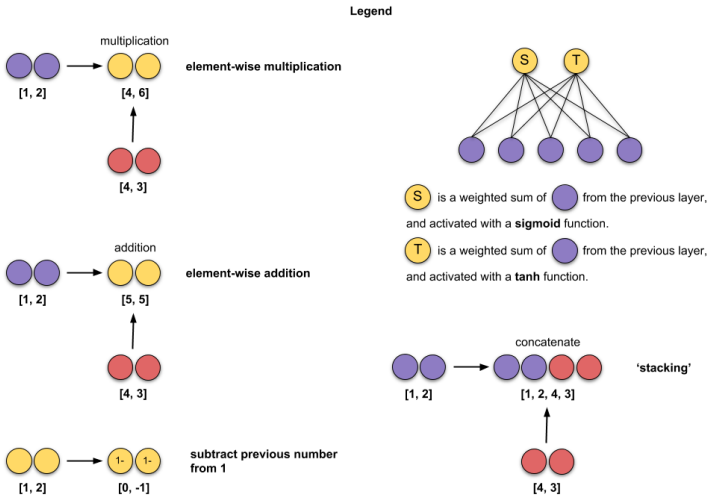
image taken from Chung et al., NIPS Workshop on Deep Learning, 2014

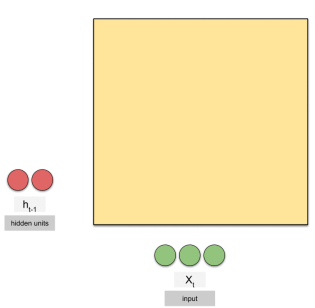
$$\boldsymbol{h}^{(t)} = \boldsymbol{z}^{(t)} \odot \boldsymbol{h}^{(t-1)} + (\mathbf{1} - \boldsymbol{z}^{(t)}) \odot \underbrace{\sigma\left(\boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{W}(\boldsymbol{r}^{(t)} \odot \boldsymbol{h}^{(t-1)})\right)}_{\tilde{\boldsymbol{h}}^{(t)}}$$

- the *update* gate z selects whether the hidden state is to be updated with a new hidden state $\tilde{h}$
- the *reset* gate r decides whether the previous hidden state is ignored

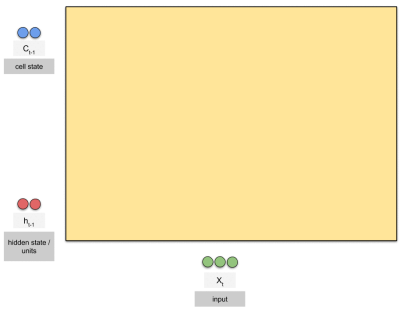
Cho et al., *EMNLP*, 2014

Summing Up

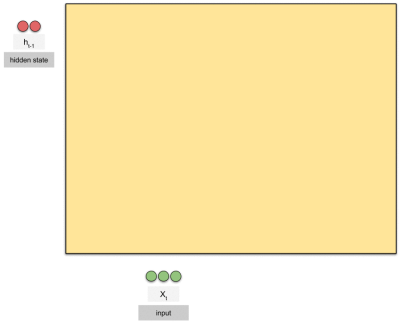




Recurrent Unit



LSTM Unit



GRU Unit

Recap: Vanishing/Exploding of Gradients: some remedies

Here are some approaches to try to *reduce* the vanishing/exploding gradients problem

- Architectural
  - Long Short-Term Memory or Gated Recurrent units
  - Reservoir Computing: Echo State Networks and Liquid State Machines
- Algorithmic
  - *Clipping gradients (avoids exploding gradients)*
  - *Hessian Free Optimization*
  - Smart Initialization: pre-training techniques

Done!

Reservoir Computing: Echo State Nets and Liquid State Machines

Idea: fix the input-to-hidden and hidden-to-hidden connections at random values and only learn the output units connections

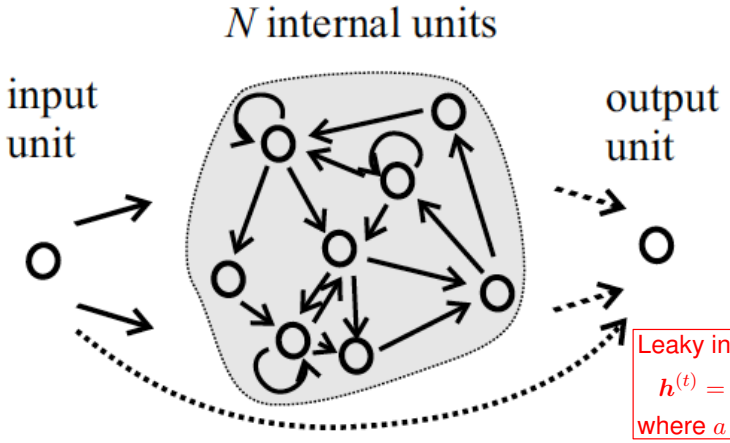


image taken from:
H. Jaeger, *NIPS 2002*

Leaky integrator unit:
$\boldsymbol{h}^{(t)} = (1-a)\boldsymbol{h}^{(t-1)} + \sigma(\boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{W}\boldsymbol{h}^{(t-1)})$,
where a is the *leaky decay rate*

- Echo State Networks (ESN): standard recurrent neurons (+ leaky integrators)
  Jaeger, Technical Report GMD Report 148, 2001
- Liquid State Machines (LSM): spiking integrate-and-fire neurons and dynamic synaptic connection models
  Maass et al., *Neural Computation*, 2002

Reservoir Computing: additional details

1. A recurrent neural network with no output (the *reservoir*) is randomly created and remains unchanged during training
   - it is passively excited by the input signal $\boldsymbol{x}^{(t)}$
   - the hidden state $\boldsymbol{h}^{(t)}$ maintains a nonlinear version of input history
   - in order to produce a "rich" set of dynamics, the reservoir should
     - be big (hundreds to thousands units)
     - be sparsely (hidden weight matrix $\boldsymbol{W}$ up to 20% possibile connections) and randomly (uniform distribution symmetric around zero) connected
     - satisfy the *echo state property*, i.e. the effect of the current state $\boldsymbol{h}^{(t)}$ and the current input $\boldsymbol{x}^{(t)}$ on a future state $\boldsymbol{h}^{(t+\tau)}$ should vanish gradually as time passes ($\tau \to \infty$), in practice:

       spectral radius $\rho(\boldsymbol{W}) < 1$, i.e. $\boldsymbol{W}$ is *contractive*
   - on the contrary, the input ($\boldsymbol{U}$) and *optional* output feedback weight matrices are dense (still random with uniform distribution)
2. The output is computed as a linear combination of the input-excited reservoir, and the linear combination is obtained by linear regression

Reservoir Computing: additional details

- alternative topologies for the reservoir, e.g. small-world, scale-free, and topologies inspired by spatial growth, were compared with no significant improvement over simple random networks
- on the other side, very simple topologies can be very effective

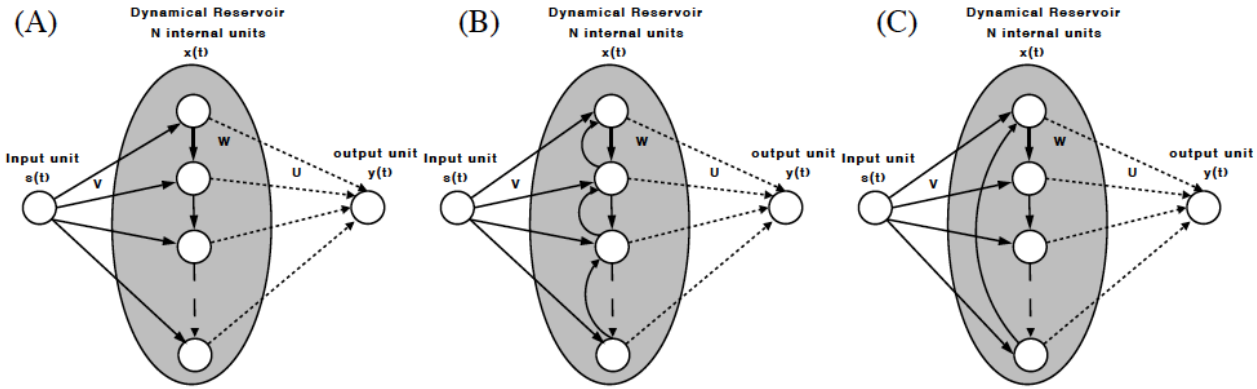


Fig. 2. (A) Delay Line Reservoir (DLR). (B) Delay Line Reservoir with feedback connections (DLRB). (C) Simple Cycle Reservoir (SCR).

from Rodan and Tino, *IEEE TNN*, 2011

- a simple cycle reservoir topology obtains performances comparable to those of ESN
- competitive reservoirs can be constructed deterministically
- the memory capacity of simple linear cyclic reservoirs can be made to be arbitrarily close to the proved optimal memory capacity value

Reservoir Computing: additional details

- *Intrinsic Plasticity* (IP) has attracted a wide attention in the reservoir computing community
- IP is a computationally efficient online learning rule to adjust threshold and gain of sigmoid reservoir neurons
  - it drives the neurons' output activities to approximate exponential distributions
  - the exponential distribution maximizes the entropy of a non-negative random variable with a fixed mean, thus enabling the neurons to transmit maximal information

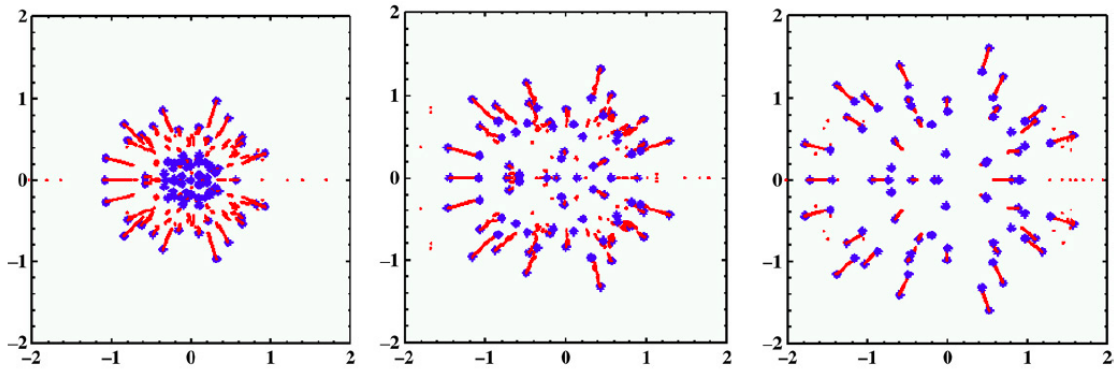


Fig. 6. Changes in the eigenvalue distribution introduced by IP for epochs 0–4, 4–9, 9–14 of a 100 neuron network learning a one-step-ahead prediction of the Mackey–Glass attractor (as detailed in Section 4). Larger dots denote the positions of weight matrix eigenvalues at the beginning and end of the 5-epoch intervals, small dotted lines indicate the path of their movements away from the center.

from Steil, *Neural Networks*, 2007

Deep Echo State Networks

Deep layered organization of RC models have been investigated in terms of

- occurrence of multiple time-scale
- increasing of richness of the dynamics

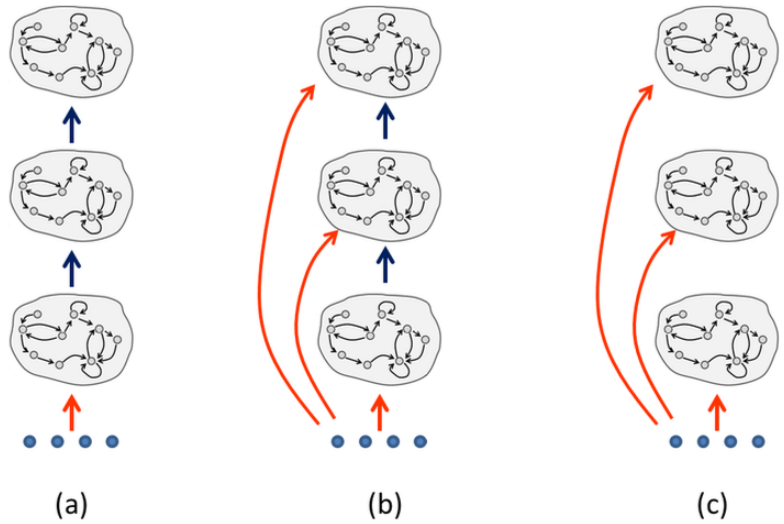


Figure 1: Deep RC architectures: **(a)** deepESN, **(b)** deepESN-IA, **(c)** groupedESN.
from Gallicchio and Micheli, *Neurocomputing*, 2017

Deep Echo State Networks: Richness of Dynamics

Entropy of output distribution $f_i(x)$ of reservoir unit i: $-\int f_i(x) \log f_i(x) dx$

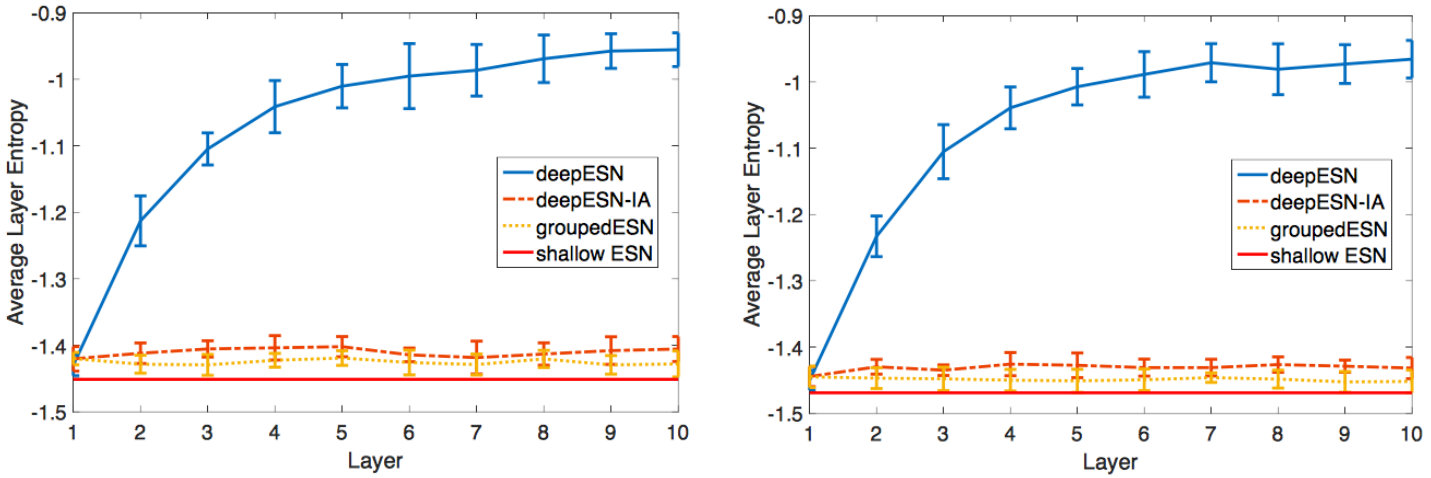


Figure 6: Layer-wise averaged entropy of reservoir states on the Artificial dataset **(a)** and on the Wikipedia dataset **(b)**, computed for deepESN, groupedESN and deepESN-IA, with $a = 0.55$ and $\rho = 0.9$ for every layer (or sub-reservoir) and IP learning. For groupedESN the results refer to sub-reservoirs. The average entropy of the shallow ESN counterpart is reported as a continuous red line across each plot.

from Gallicchio and Micheli, *Neurocomputing*, 2017

Deep Echo State Networks: Memory Capacity

Task: reconstruct the input with increasing delay

- target $\boldsymbol{y}_k^{(t)} = \boldsymbol{x}^{(t-k)}, \forall k \in [0, \ldots, \infty]$
- Memory Capacity (MC):

$$\sum_{k=0}^{\infty} r^2(\boldsymbol{x}^{(t-k)}, \boldsymbol{o}_k^{(t)})$$

where $r^2(\boldsymbol{x}^{(t-k)}, \boldsymbol{o}_k^{(t)})$ is the squared correlation coefficient between:

- the input $\boldsymbol{x}^{(t-k)}$ with delay k
- the corresponding output $\boldsymbol{o}_k^{(t)}$ generated by the network at time t for delay k

Deep Echo State Networks

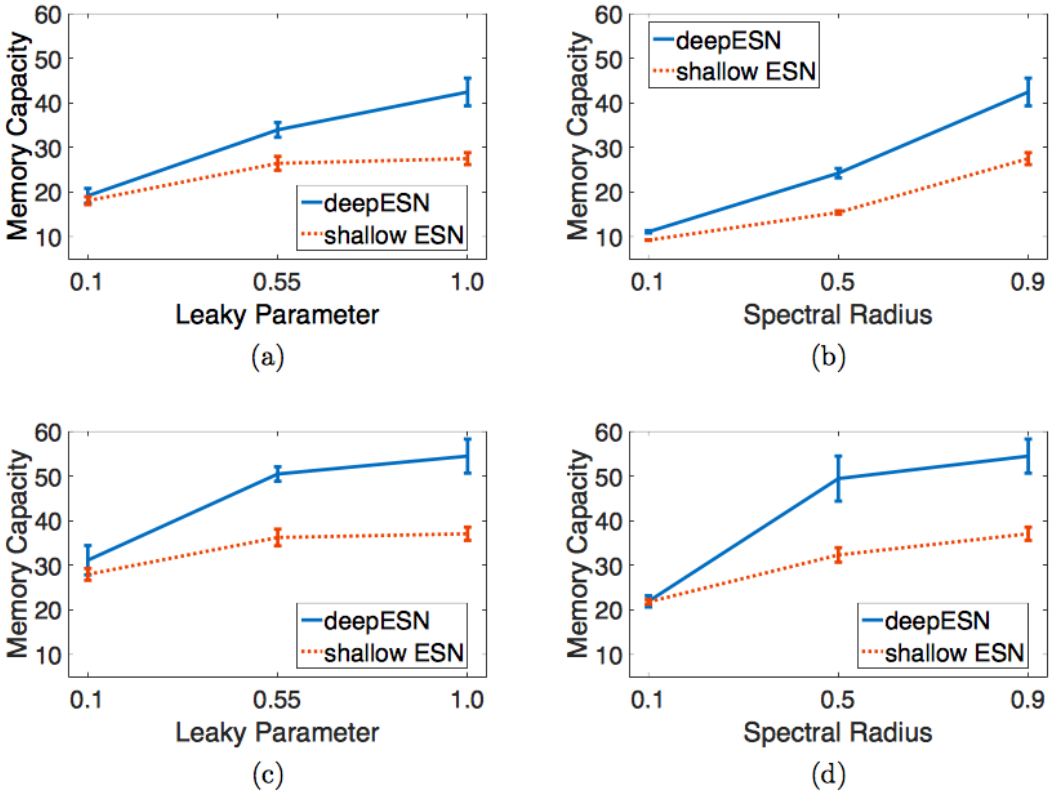
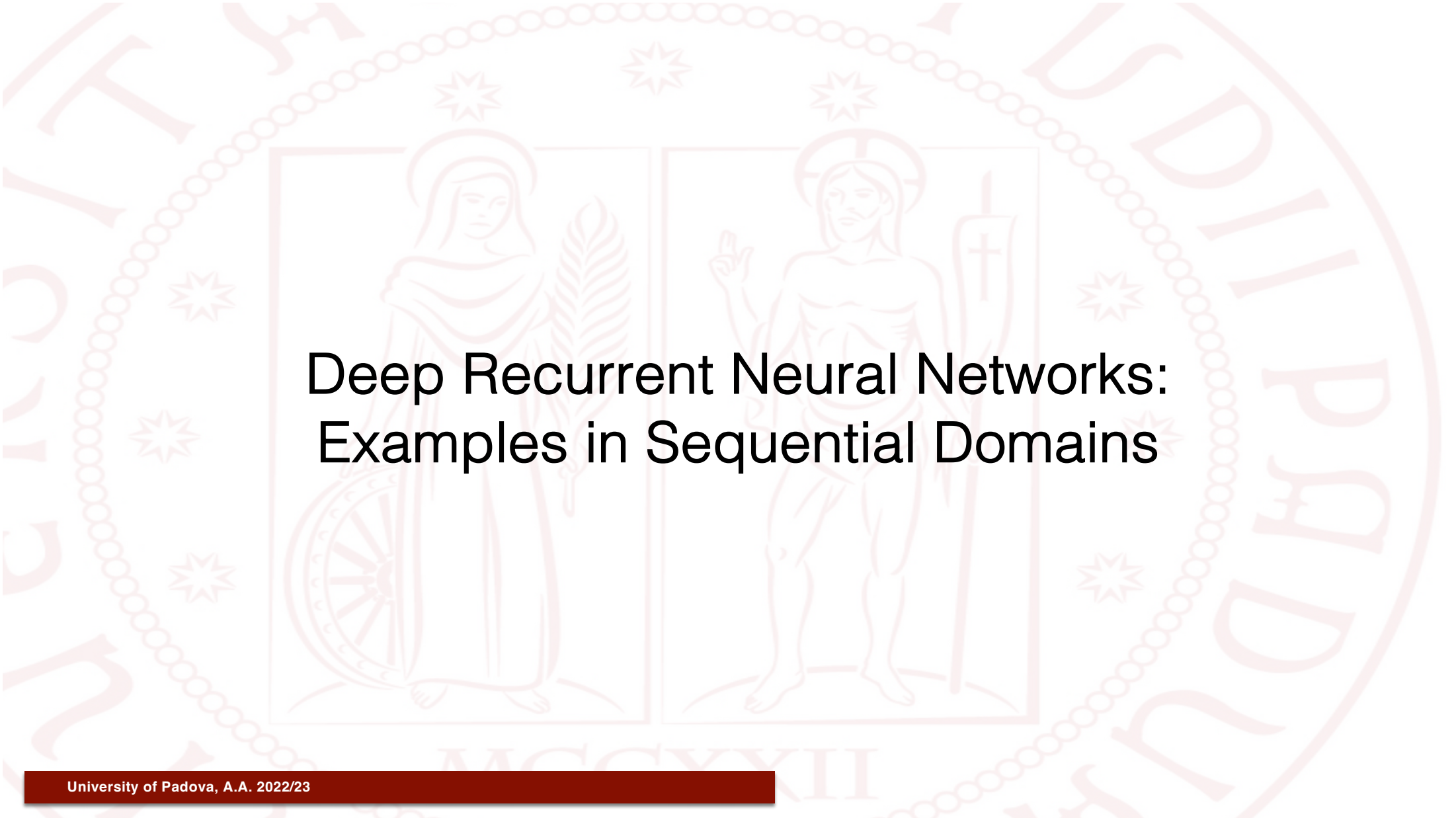


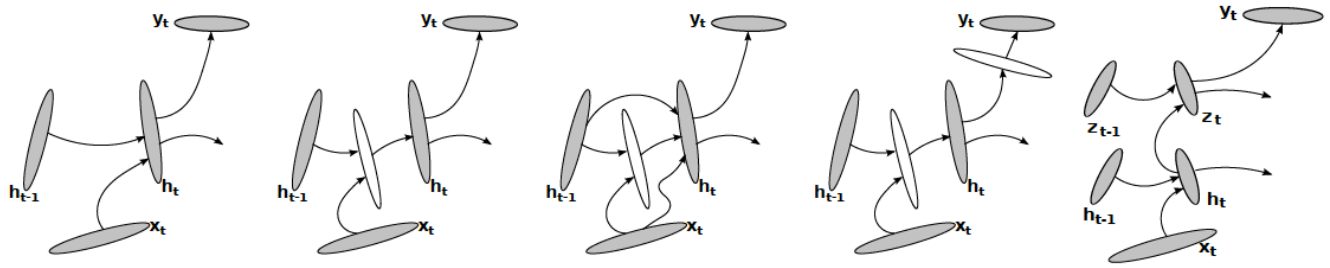
Figure 7: MC results of deepESN and shallow ESN for different values of the spectral radius ρ and of the leaky parameter a. **(a)**: different values of a, without IP, **(b)**: different values of ρ, without IP, **(c)**: different values of a, with IP, **(d)**: different values of ρ, with IP. For deepESN each reservoir layer has the same hyper-parametrization.

from Gallicchio and Micheli, *Neurocomputing*, 2017



Deep Recurrent Neural Networks: Examples in Sequential Domains

How to Construct Deep Recurrent Neural Networks



(a) RNN (b) DT-RNN (b*) DT(S)-RNN (c) DOT-RNN (d) Stacked RNN

Figure 2: Illustrations of four different recurrent neural networks (RNN). (a) A conventional RNN. (b) Deep Transition (DT) RNN. (b*) DT-RNN with shortcut connections (c) Deep Transition, Deep Output (DOT) RNN. (d) Stacked RNN
image taken from: Pascanu et al., arXiv:1312.6026v5

Empirical experiments on polyphonic music prediction and language modeling showed that RNN benefits from having a deeper architecture

Pascanu et al., arXiv:1312.6026v5, 2014

Examples of Deep Architectures

- Novel deep RNN architectures/mechanisms have been designed for specific applications
- Examples of architectures using
  - Encoder-Decoder
  - Alignment/Attention
- image to text applications

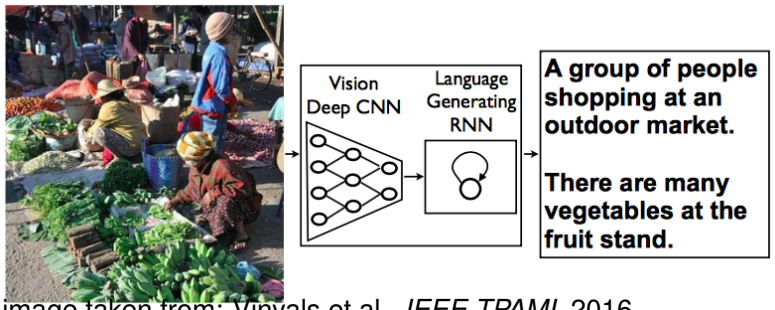


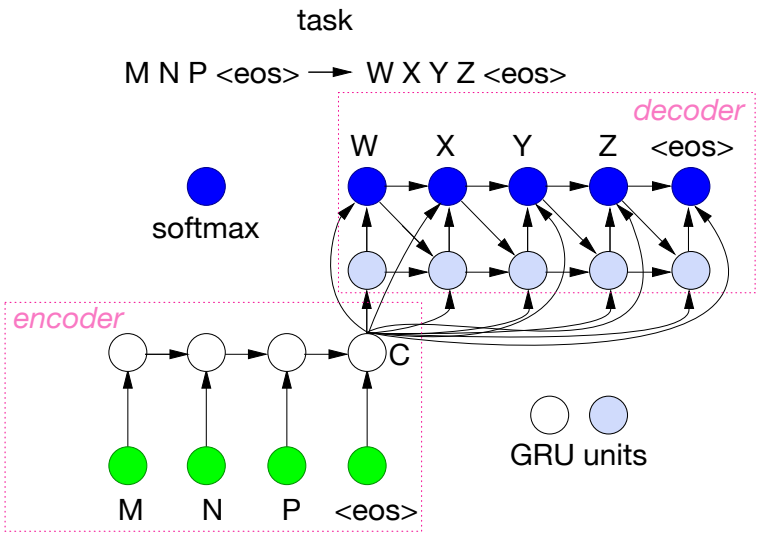
image taken from: Vinyals et al., *IEEE TPAMI*, 2016

Other interesting architectures that we do not have time to look at:

- Adaptive Computation Time, “External” memory, Pointers

Encoding-Decoding Networks

$$\text{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$



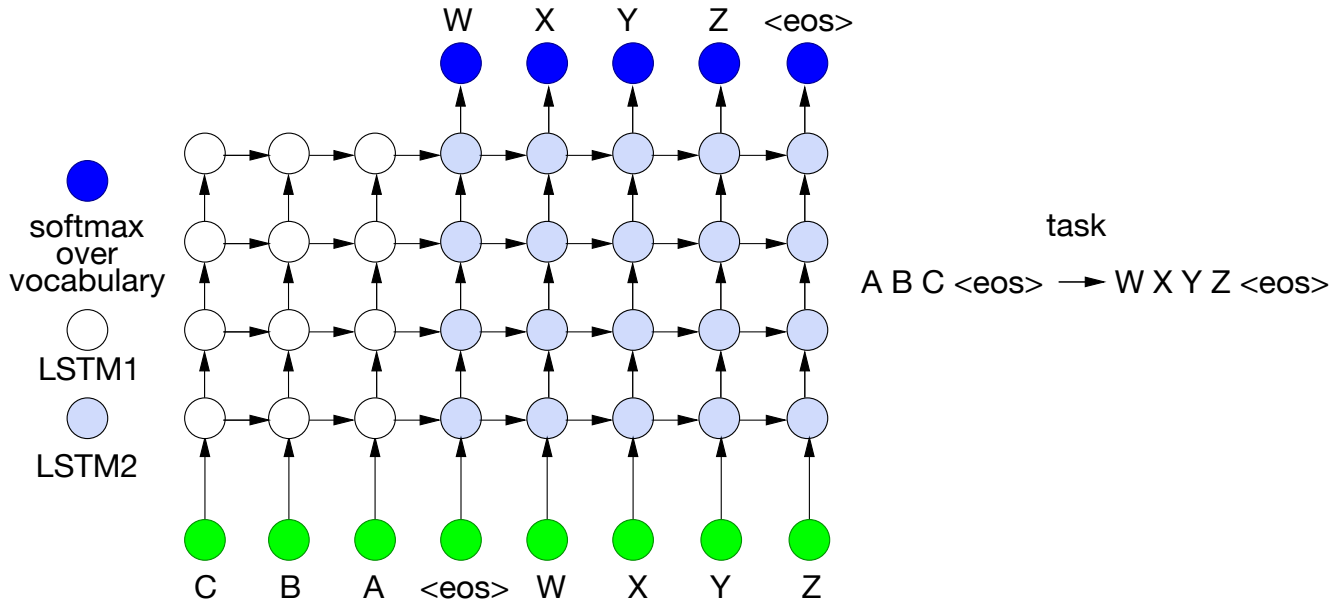
End-to-end learning of $(\max_\theta \frac{1}{N} \sum_{n=1}^{N} \log P_\theta(\boldsymbol{s}_n^d | \boldsymbol{s}_n^i))$

$$P(\boldsymbol{o}_t | \boldsymbol{o}_{t-1}, \boldsymbol{o}_{t-2}, \ldots, \boldsymbol{o}_1, \boldsymbol{c}) \quad = \quad g(\boldsymbol{h}_t, \boldsymbol{o}_{t-1}, \boldsymbol{c})$$

where $\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{o}_{t-1}, \boldsymbol{c})$

Cho et al., *EMNLP 2014*

Sequence to Sequence Networks



End-to-end learning of

$$P(\boldsymbol{o}_1, \boldsymbol{o}_2, \ldots, \boldsymbol{o}_{T'} | \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T) = \prod_{t=1}^{T'} P(\boldsymbol{o}_t | \boldsymbol{v}, \boldsymbol{o}_{t-1}, \boldsymbol{o}_{t-2}, ..., \boldsymbol{o}_1)$$

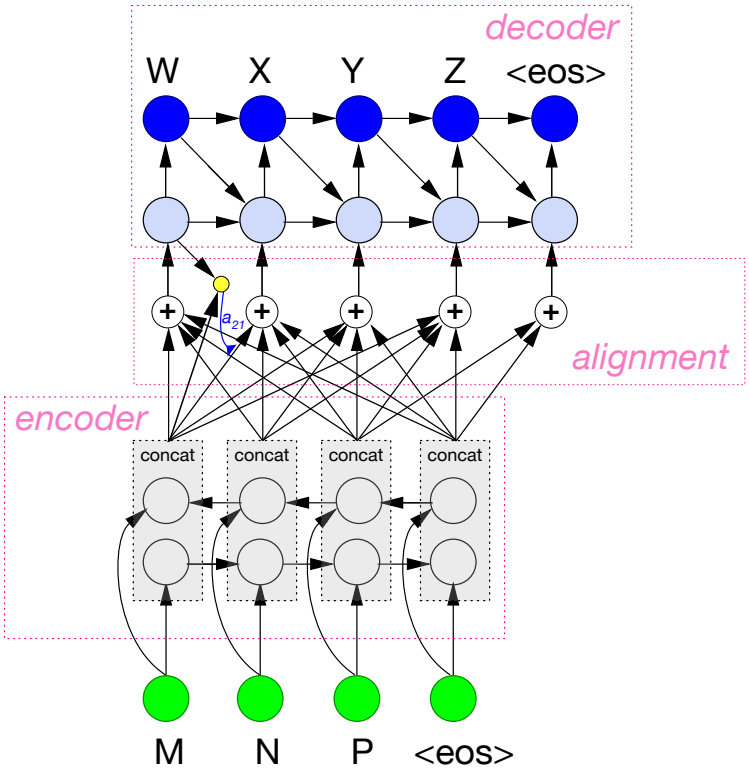
where $\boldsymbol{v} = \boldsymbol{h}_{LSTM1}(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T)$

Sutskever et al., *NIPS 2014*

Encoding-Decoding Networks with Alignment

task

M N P <eos> → W X Y Z <eos>



Bahdanau et al., *ICLR 2015*

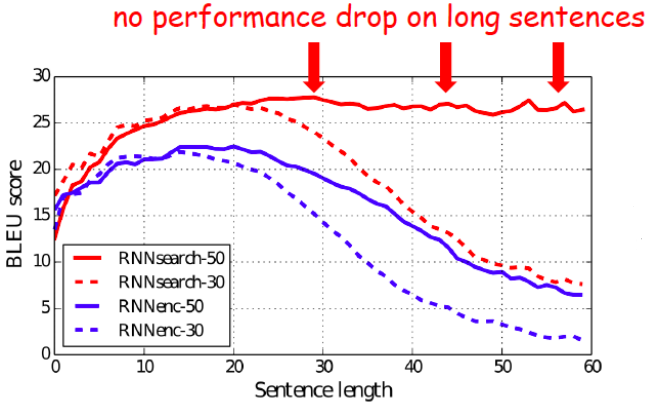
$$P(\boldsymbol{o}_t|\boldsymbol{o}_{t-1},\ldots,\boldsymbol{o}_1,\boldsymbol{s}^i) = g(\boldsymbol{o}_{t-1},\boldsymbol{h}_t^{dec},\boldsymbol{c}_t)$$

$$\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}^{dec},\boldsymbol{o}_{t-1},\boldsymbol{c}_t)$$

$$\boldsymbol{c}_t = \sum_{j=1}^{T} \alpha_{ij}\boldsymbol{h}_j^{enc}$$

$$\alpha_{ij} = softmax_j(e_{ij})$$

$$e_{ij} = a(\boldsymbol{h}_{i-1}^{dec},\boldsymbol{h}_j^{enc})$$



One example of Image-Text Alignment (Karpathy & Fei-Fei, IEEE TPAMI, 2017)

Task

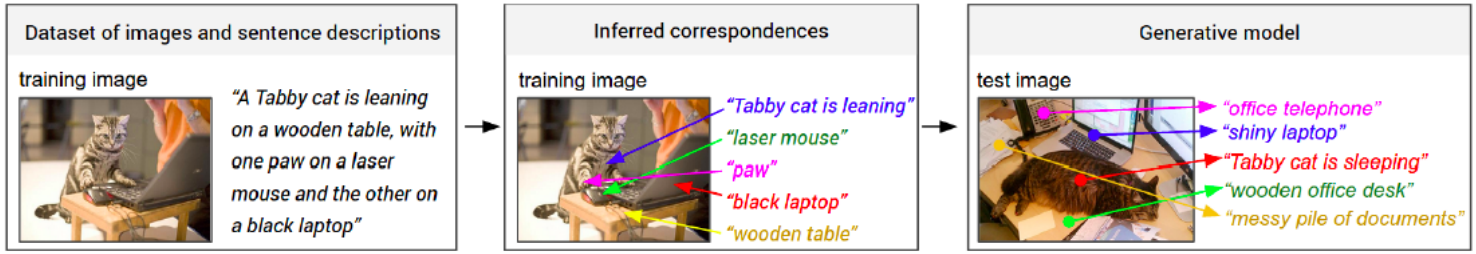
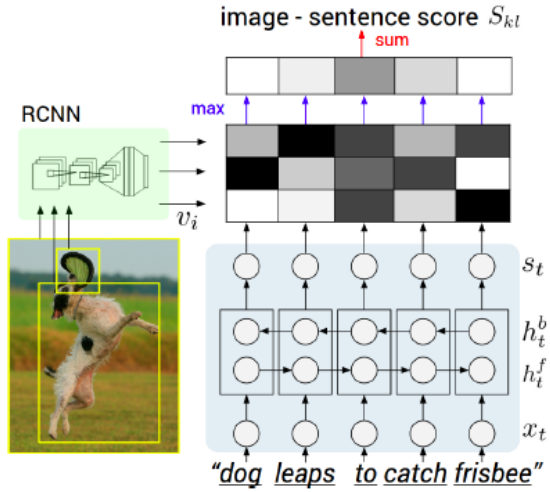


Figure 2. Overview of our approach. A dataset of images and their sentence descriptions is the input to our model (left). Our model first infers the correspondences (middle, Section 3.1) and then learns to generate novel descriptions (right, Section 3.2).



Multimodal Recurrent Neural Network architecture that uses the inferred alignments to learn to generate novel descriptions of image regions