Tutorial on recurrent neural networks

Jordi Pons

Researcher at Dolby Laboratories
@jordiponsdotme – www.jordipons.me

Master in Sound and Music Computing
Universitat Pompeu Fabra, Winter 2019

Recurrent Neural Networks

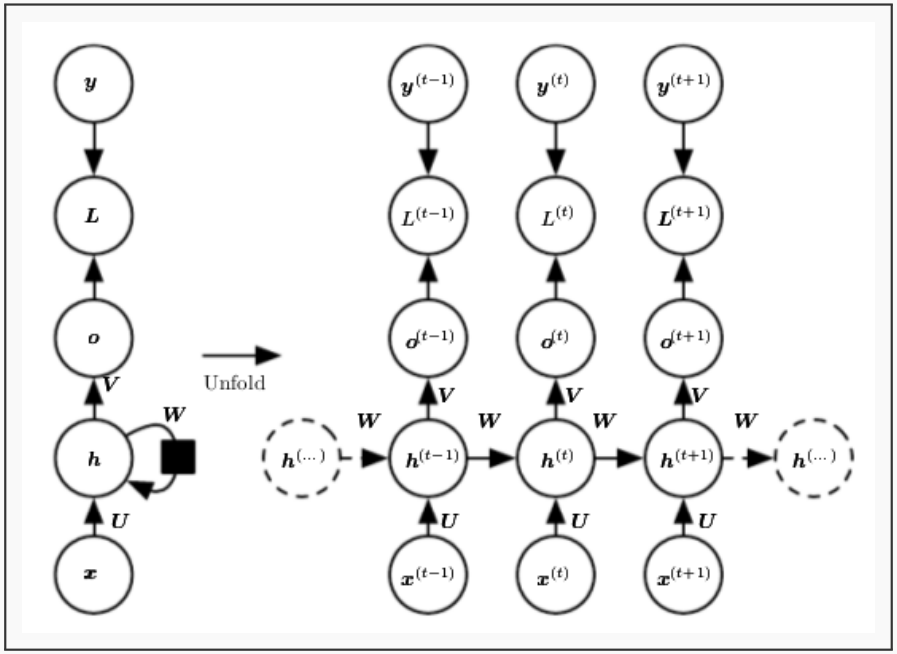


Figure: Unfolded recurrent neural network

$$\mathbf{h}^{(t)} = \sigma(\mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)})$$

$$\mathbf{o}^{(t)} = \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)}$$

Throughout this presentation we assume single layer RNNs!

Recurrent Neural Networks

$$\mathbf{h}^{(t)} = \sigma(\mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)}) \qquad \mathbf{o}^{(t)} = \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)}$$

- $\mathbf{x} \in \mathbb{R}^{d_{in} \times 1}$
- $\mathbf{b} \in \mathbb{R}^{d_h \times 1}$
- $\mathbf{U} \in \mathbb{R}^{d_h \times d_{in}}$
- $\mathbf{V} \in \mathbb{R}^{d_{out} \times d_h}$

$\mathbf{y} \in \mathbb{R}^{d_{out} \times 1}$

$\mathbf{h} \in \mathbb{R}^{d_h \times 1}$

$\mathbf{W} \in \mathbb{R}^{d_h \times d_h}$

σ: element-wise non-linearity.

Where d_{in}, d_h and d_{out} correspond to the dimensions of the input layer, hidden layer and output layer, respectively.

Recurrent Neural Networks

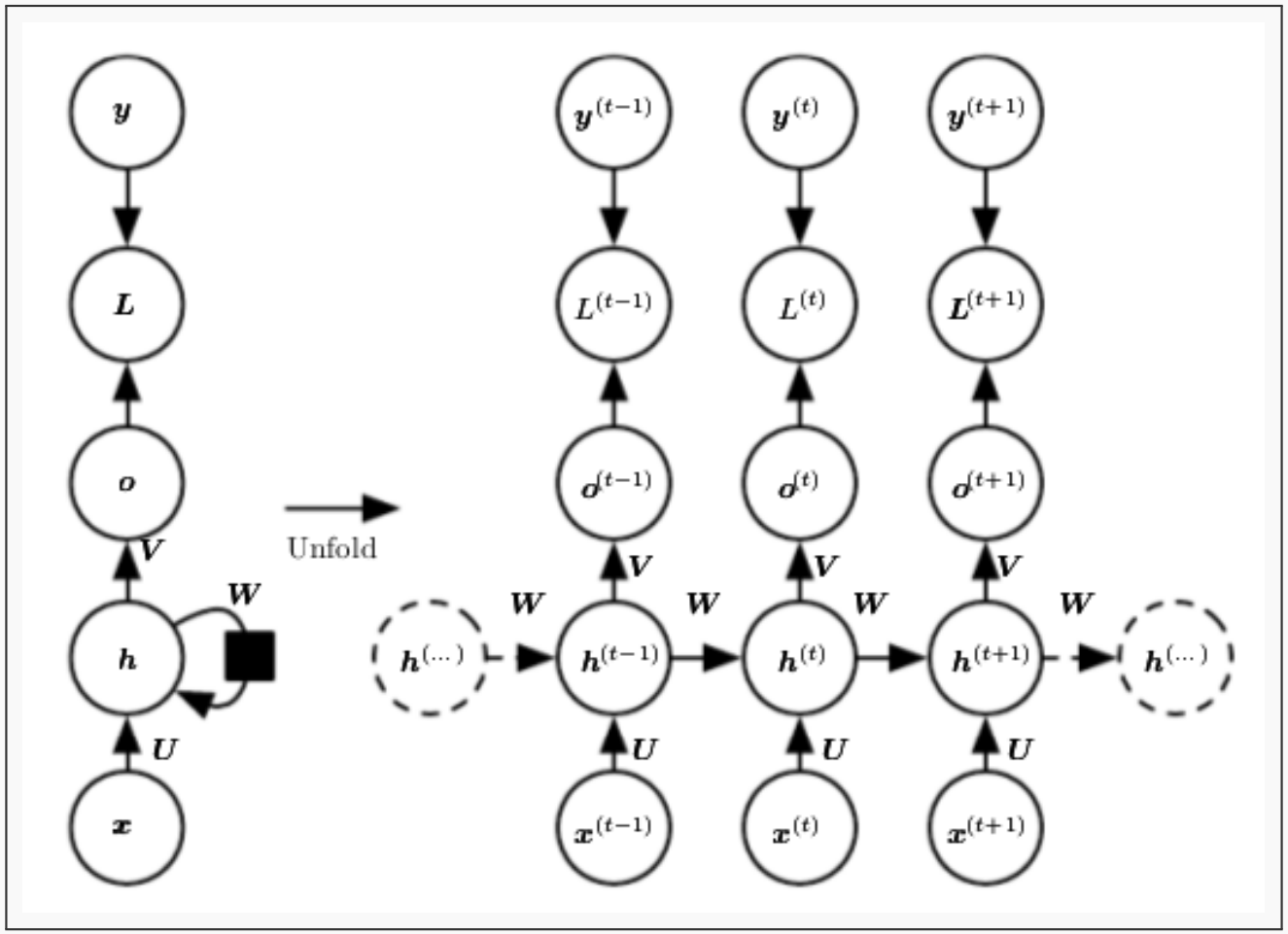


Figure: Unfolded recurrent neural network

Gradient vanish/explode: forward path

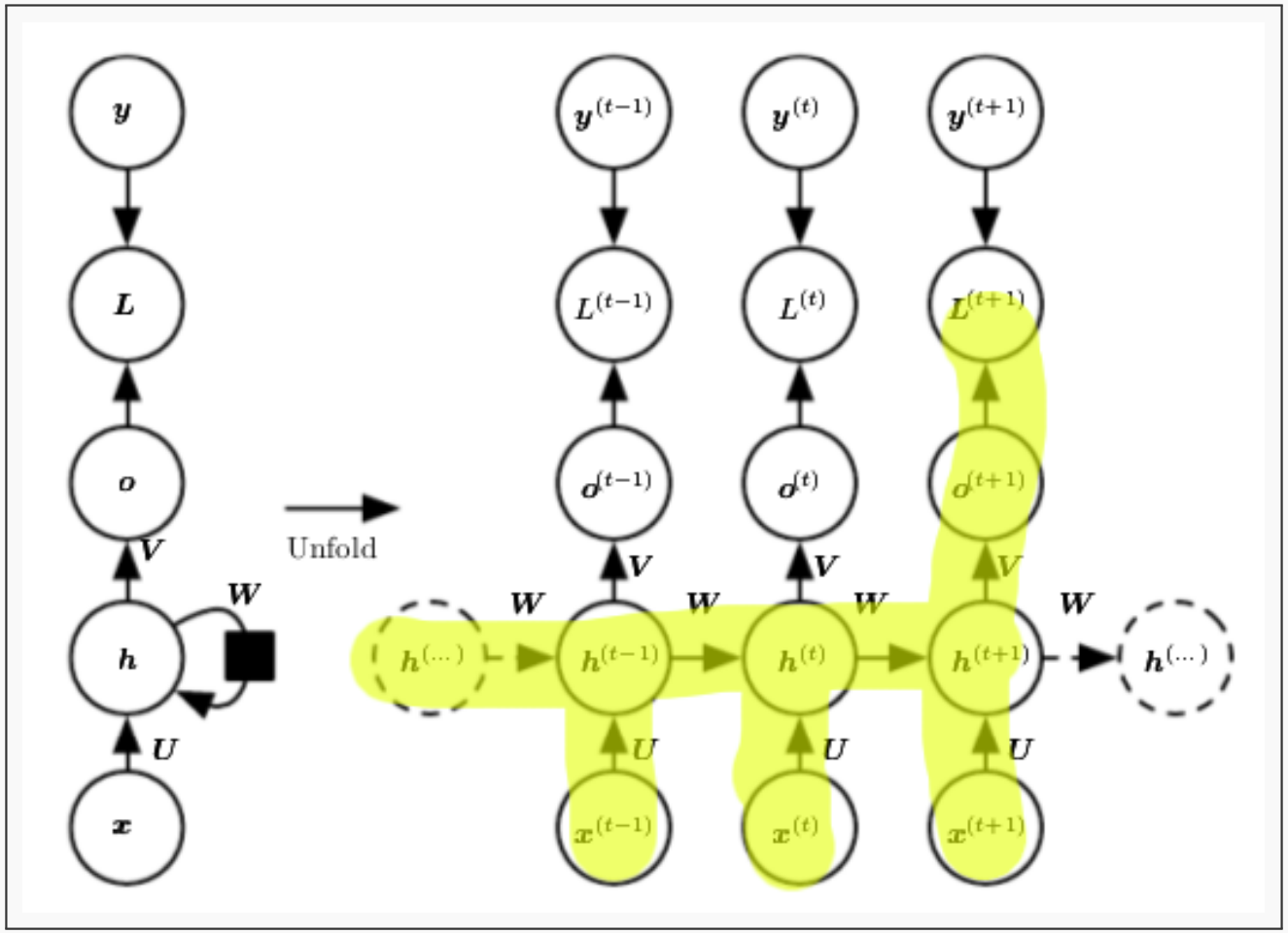


Figure: Forward and backward path for time-step $t + 1$ (yellow).

Gradient vanish/explode: problematic path

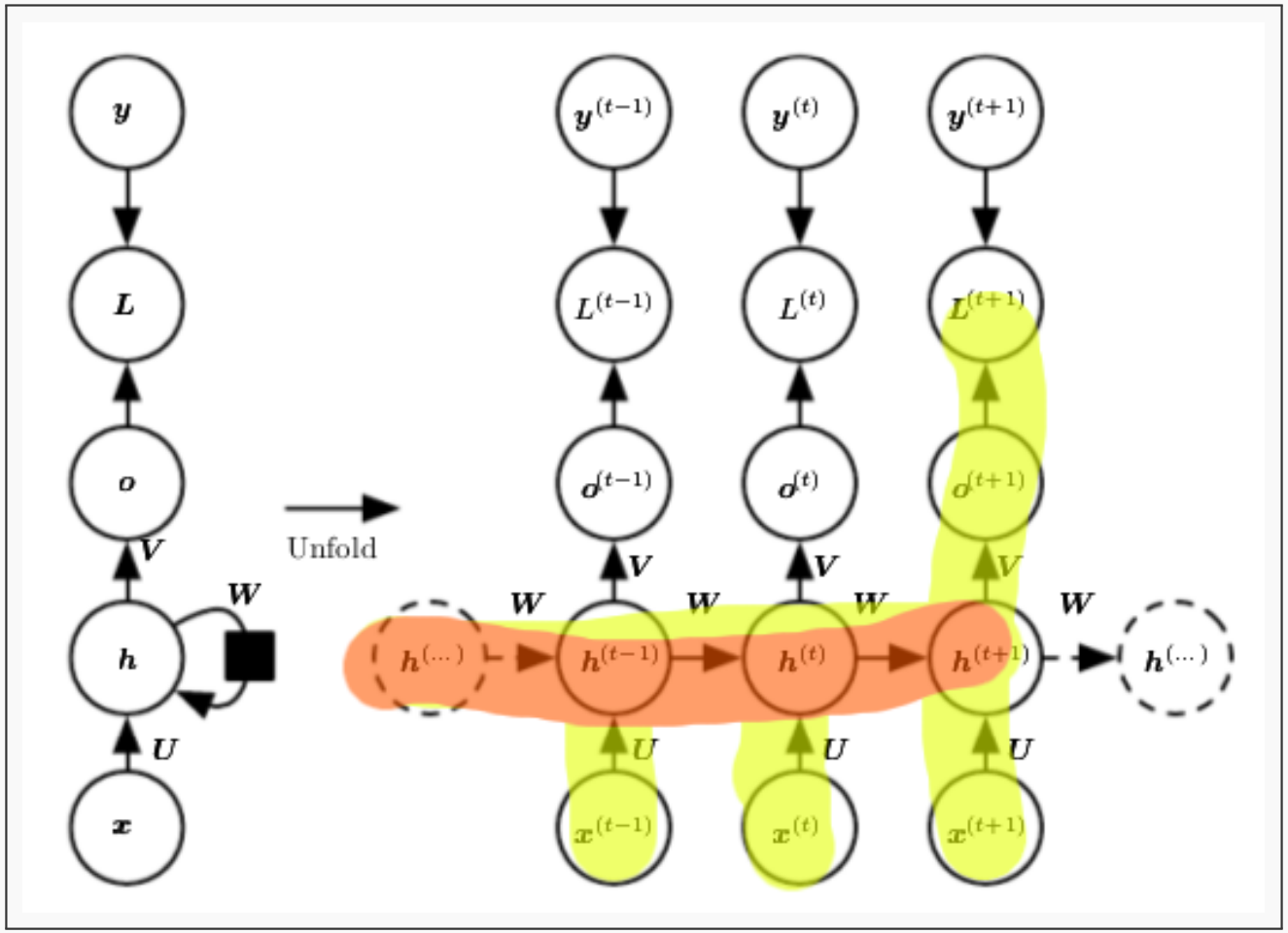
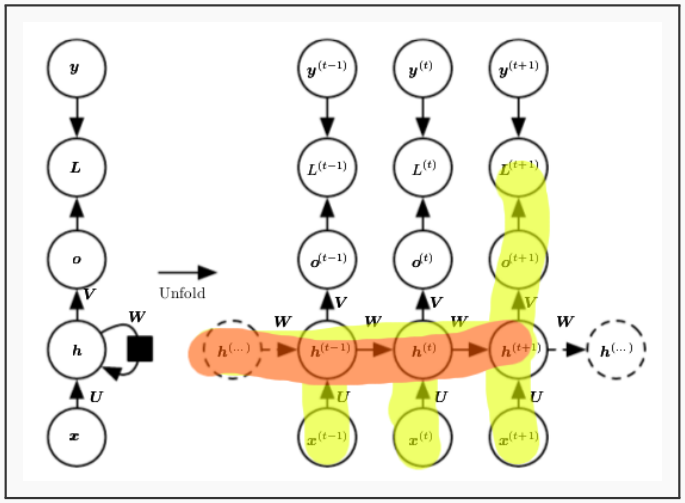


Figure: Problematic path (red).

Intuition from the forward linear case

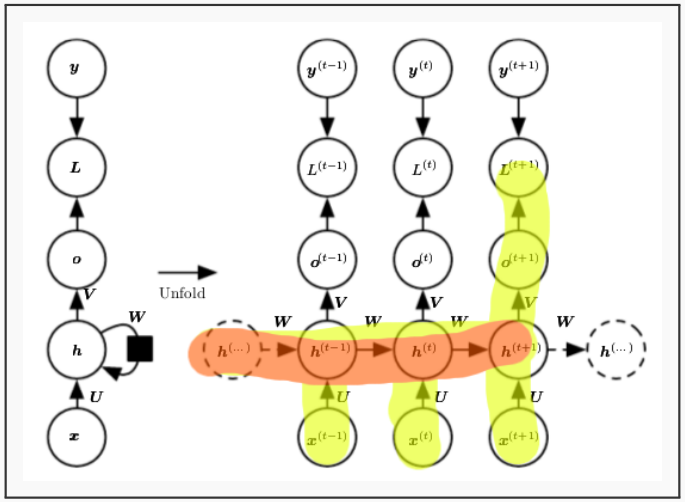
Intuition from the forward linear case



- Study case: **forward problematic path** for the **linear case**.
  $\mathbf{h}^{(t)} = \sigma(\mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)}) \rightarrow \mathbf{h}^{(t)} = \mathbf{W}\mathbf{h}^{(t-1)}$

- After t steps, this is equivalent to multiply $\mathbf{W}^t$.

Intuition from the forward linear case



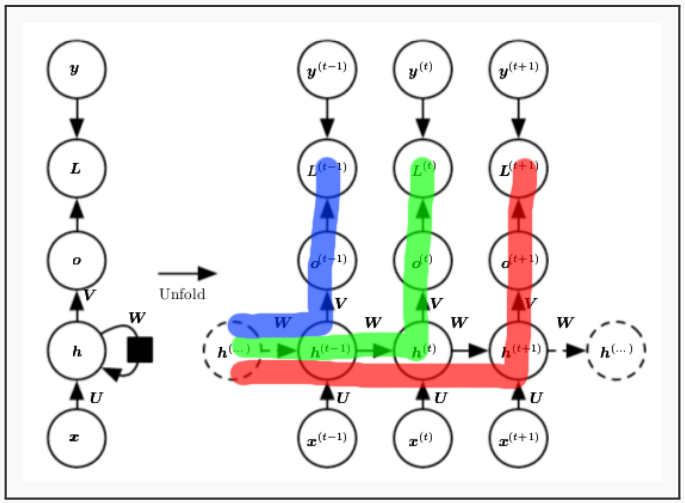
$$\mathbf{h}^{(t)} = \mathbf{W}\,\mathbf{h}^{(t-1)} \quad \rightarrow \quad \mathbf{h}^{(t)} = \mathbf{W}^t\,\mathbf{h}^{(0)}$$

if $\boldsymbol{W} < 1$: $\mathbf{h}^{(t)}$ will tend to 0 (will "**vanish**").
if $\boldsymbol{W} > 1$: $\mathbf{h}^{(t)}$ will tend to ∞ (will "**explode**").

Gradient vanish/explode: back-propagation with non-linearities

Gradient vanish/explode: back-propagation with non-linearities

→ Consider the **backward pass** when **non-linearities** are present.
→ This will allow to explicitly describe the **gradients** issue.



$$\frac{\partial \mathbf{L}}{\partial \mathbf{W}} = \frac{\partial L\left(\mathbf{o}^{(t+1)}, \mathbf{y}^{(t+1)}\right)}{\partial \mathbf{W}} + \frac{\partial L\left(\mathbf{o}^{(t)}, \mathbf{y}^{(t)}\right)}{\partial \mathbf{W}} + \frac{\partial L\left(\mathbf{o}^{(t-1)}, \mathbf{y}^{(t-1)}\right)}{\partial \mathbf{W}} =$$

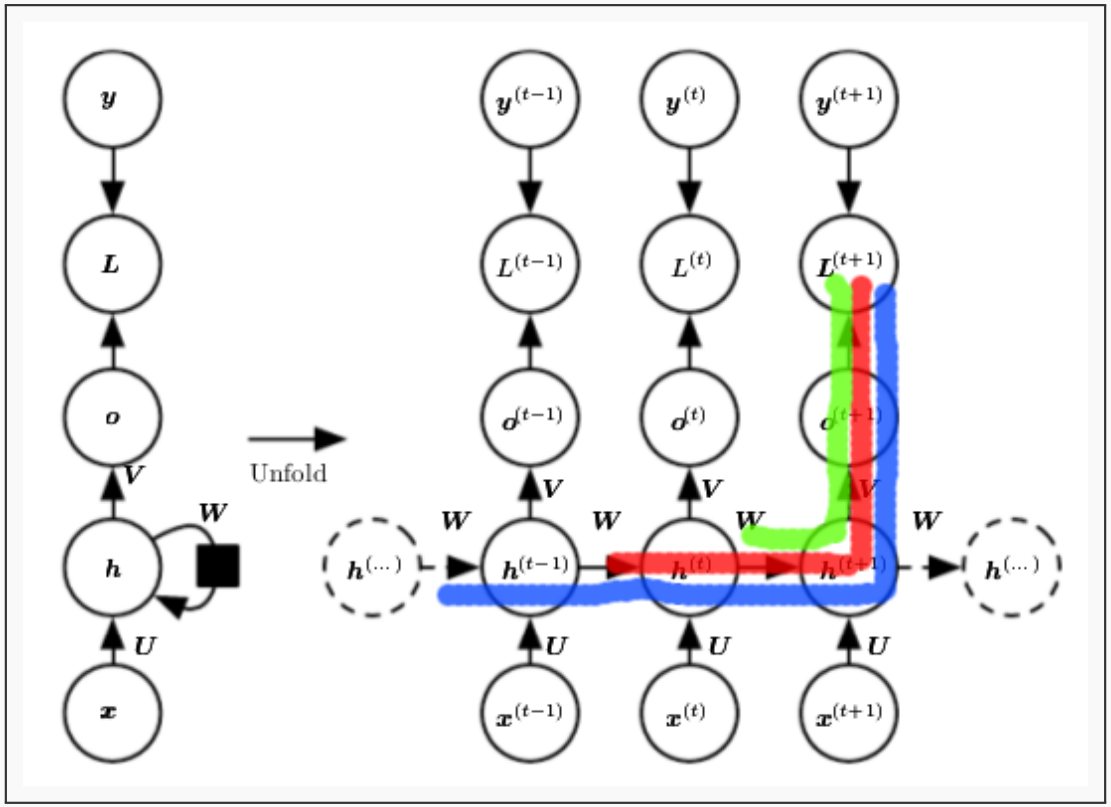
$$= \sum_{S=t_{min}}^{t_{max}} \frac{\partial L\left(\mathbf{o}^{(t-k)}, \mathbf{y}^{(t-k)}\right)}{\partial \mathbf{W}}$$

where $S \in [t_{min}, t_{max}] \rightarrow$ S being the horizon of the BPTT alghoritm.

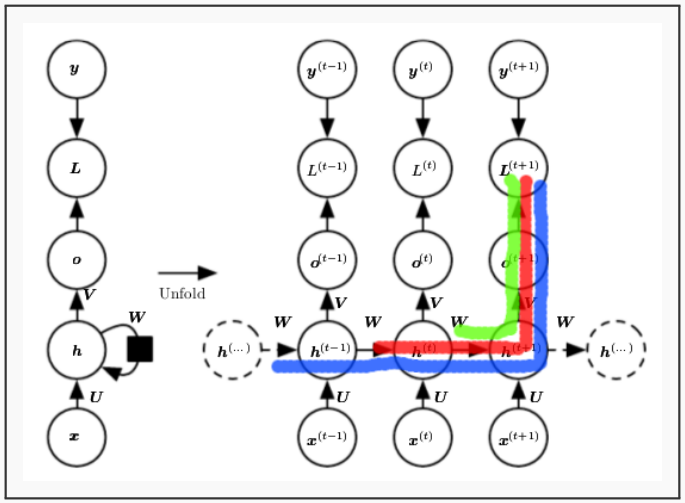
Gradient vanish/explode: back-propagation with non-linearities

Let's now focus on time-step $t+1$:

$$\frac{\partial L\left(\mathbf{o}^{(t+1)}, \mathbf{y}^{(t+1)}\right)}{\partial \mathbf{W}}$$



Gradient vanish/explode: back-propagation with non-linearities



$$\frac{\partial L\left(\mathbf{o}^{(t+1)}, \mathbf{y}^{(t+1)}\right)}{\partial \mathbf{W}} = \frac{\partial L\left(\mathbf{o}^{(t+1)}, \mathbf{y}^{(t+1)}\right)}{\partial \mathbf{o}^{(t+1)}} \frac{\partial \mathbf{o}^{(t+1)}}{\partial \mathbf{h}^{(t+1)}} \frac{\partial \mathbf{h}^{(t+1)}}{\partial \mathbf{W}} +$$

$$+ \frac{\partial L\left(\mathbf{o}^{(t+1)}, \mathbf{y}^{(t+1)}\right)}{\partial \mathbf{o}^{(t+1)}} \frac{\partial \mathbf{o}^{(t+1)}}{\partial \mathbf{h}^{(t+1)}} \frac{\partial \mathbf{h}^{(t+1)}}{\partial \mathbf{h}^{(t)}} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{W}} +$$

$$+ \frac{\partial L\left(\mathbf{o}^{(t+1)}, \mathbf{y}^{(t+1)}\right)}{\partial \mathbf{o}^{(t+1)}} \frac{\partial \mathbf{o}^{(t+1)}}{\partial \mathbf{h}^{(t+1)}} \frac{\partial \mathbf{h}^{(t+1)}}{\partial \mathbf{h}^{(t)}} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} \frac{\partial \mathbf{h}^{(t-1)}}{\partial \mathbf{W}} + \ldots$$

Gradient vanish/explode: back-propagation with non-linearities

Previous equation can be summarized as follows:

$$\frac{\partial L\left(\mathbf{o}^{(t+1)}, \mathbf{y}^{(t+1)}\right)}{\partial \mathbf{W}} =$$

$$= \sum_{k=t_{min}}^{1} \frac{\partial L\left(\mathbf{o}^{(t+1)}, \mathbf{y}^{(t+1)}\right)}{\partial \mathbf{o}^{(t+1)}} \frac{\partial \mathbf{o}^{(t+1)}}{\partial \mathbf{h}^{(t+1)}} \frac{\partial \mathbf{h}^{(t+1)}}{\partial \mathbf{h}^{(t+k)}} \frac{\partial \mathbf{h}^{(t+k)}}{\partial \mathbf{W}}$$

where:

$$\frac{\partial \mathbf{h}^{(t+1)}}{\partial \mathbf{h}^{(t+k)}} = \prod_{s=k+1}^{1} \frac{\partial \mathbf{h}^{(t+s)}}{\partial \mathbf{h}^{(t+s-1)}} = \prod_{s=k+1}^{1} \mathbf{W}^T \, diag\left[\sigma'\left(\mathbf{b}+\mathbf{W}\,\mathbf{h}^{(t+s-1)}+\mathbf{U}\mathbf{x}^{(t+s)}\right)\right]$$

$$\mathbf{h}^{(t+s)} = \sigma(\mathbf{b} + \mathbf{W}\,\mathbf{h}^{(t+s-1)} + \mathbf{U}\mathbf{x}^{(t+s)})$$

$$f(x) = h(g(x)) \rightarrow f'(x) = h'(g(x)) \cdot g'(x)$$

Gradient vanish/explode: back-propagation with non-linearities

$$\frac{\partial \mathbf{h}^{(t+1)}}{\partial \mathbf{h}^{(t+k)}} = \prod_{s=k+1}^{1} \frac{\partial \mathbf{h}^{(t+s)}}{\partial \mathbf{h}^{(t+s-1)}} = \prod_{s=k+1}^{1} \mathbf{W}^T \, diag\left[\, \sigma'\left(\mathbf{b} + \mathbf{W}\, \mathbf{h}^{(t+s-1)} + \mathbf{U}\mathbf{x}^{(t+s)}\right)\right]$$

the L2 norm defines an upper bound for the **jacobians**:

$$\left\| \frac{\partial \mathbf{h}^{(t+s)}}{\partial \mathbf{h}^{(t+s-1)}} \right\|_2 \leq \left\| \mathbf{W}^T \right\|_2 \left\| diag\left[\, \sigma'(\cdot)\right] \right\|_2 \equiv \gamma_w \gamma_\sigma$$

$\rightarrow \gamma_w \equiv$ L2 norm of a matrix. $\equiv$ highest eigenvalue. $\equiv$ spectral radius.

$\rightarrow \gamma_\sigma \in [0, 1]$.

and therefore:

$$\left\| \frac{\partial \mathbf{h}^{(t+1)}}{\partial \mathbf{h}^{(t+k)}} \right\|_2 \leq (\gamma_w \gamma_\sigma)^{|k-1|}$$

Gradient vanish/explode: back-propagation with non-linearities

then, consider: $$\left\| \frac{\partial \mathbf{h}^{(t+1)}}{\partial \mathbf{h}^{(t+k)}} \right\|_2 \leq (\gamma_w \gamma_\sigma)^{|k-1|}$$

$\gamma_w \gamma_\sigma \gg 1$: $$\left\| \frac{\partial \mathbf{h}_{(t)}^{(1)}}{\partial \mathbf{h}_{(k)}^{(1)}} \right\|_2 \rightarrow \frac{\partial L\left(\mathbf{o}^{(t+1)}, \mathbf{y}^{(t+1)}\right)}{\partial \mathbf{W}} \rightarrow \frac{\partial \mathbf{L}}{\partial \mathbf{W}}$$ explodes!

$\gamma_w \gamma_\sigma \ll 1$: $$\left\| \frac{\partial \mathbf{h}_{(t)}^{(1)}}{\partial \mathbf{h}_{(k)}^{(1)}} \right\|_2 \rightarrow \frac{\partial L\left(\mathbf{o}^{(t+1)}, \mathbf{y}^{(t+1)}\right)}{\partial \mathbf{W}} \rightarrow \frac{\partial \mathbf{L}}{\partial \mathbf{W}}$$ vanishes!

$\gamma_w \gamma_\sigma \approx 1$: gradients should propagate well until the past

Gradient vanish/explode: take away message

$\rightarrow$ **Vanishing** gradients make it difficult to know which direction the parameters should move to improve the cost function.

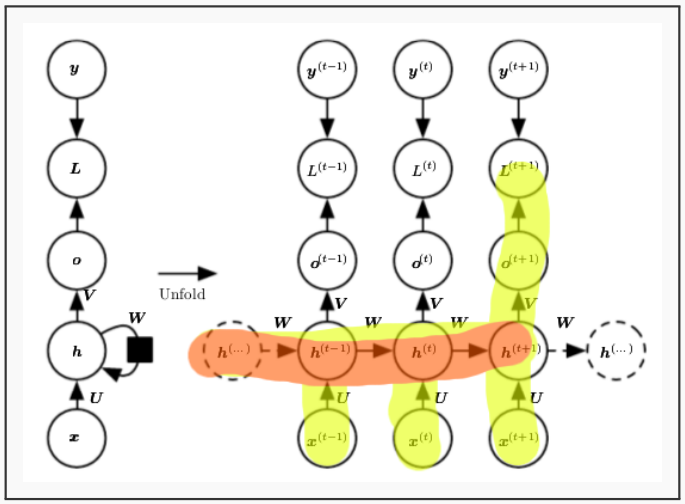
$\rightarrow$ While **exploding** gradients can make learning unstable.

$\rightarrow$ The **vanishing and exploding gradient** problem refers to the fact that gradients are **scaled** according to:

$$\left\| \frac{\partial \mathbf{h}^{(t+1)}}{\partial \mathbf{h}^{(t+k)}} \right\|_2 \leq (\gamma_w \gamma_\sigma)^{|k-1|} \qquad \textit{with, tipically} \ : \ \gamma_\sigma \in [0, 1]$$

Gradient vanish/explode in (regular) deep neural networks?

Gradient vanish/explode in (regular) deep neural networks?



- Recurrent networks use the same matrix **W** at each time step.
- Feedforward networks do not use the same matrix **W**:
  → **Very deep feedforward networks can avoid the vanishing and exploding gradient problem**.
  → if an appropriate scaling for **W**'s variance is chosen.

Gated units: intuition

1. **Accumulating is just a bad memory?**

   $\mathbf{W} = \mathbf{I}$ and **linear** $\cong$ $\gamma_w \gamma_\sigma = 1 \cdot 1 = 1$

   RNNs can *accumulate* but it might be useful to ***forget***.

2. Creating **paths** through time where **derivatives can flow**.

3. **Learn** when to forget!

Gates allow learning how to read, write and forget!

Two different gated units will be presented:

Long Short-Term Memory (**LSTM**)
Gated Recurrent Unit (**GRU**)

LSTM – block diagram 1

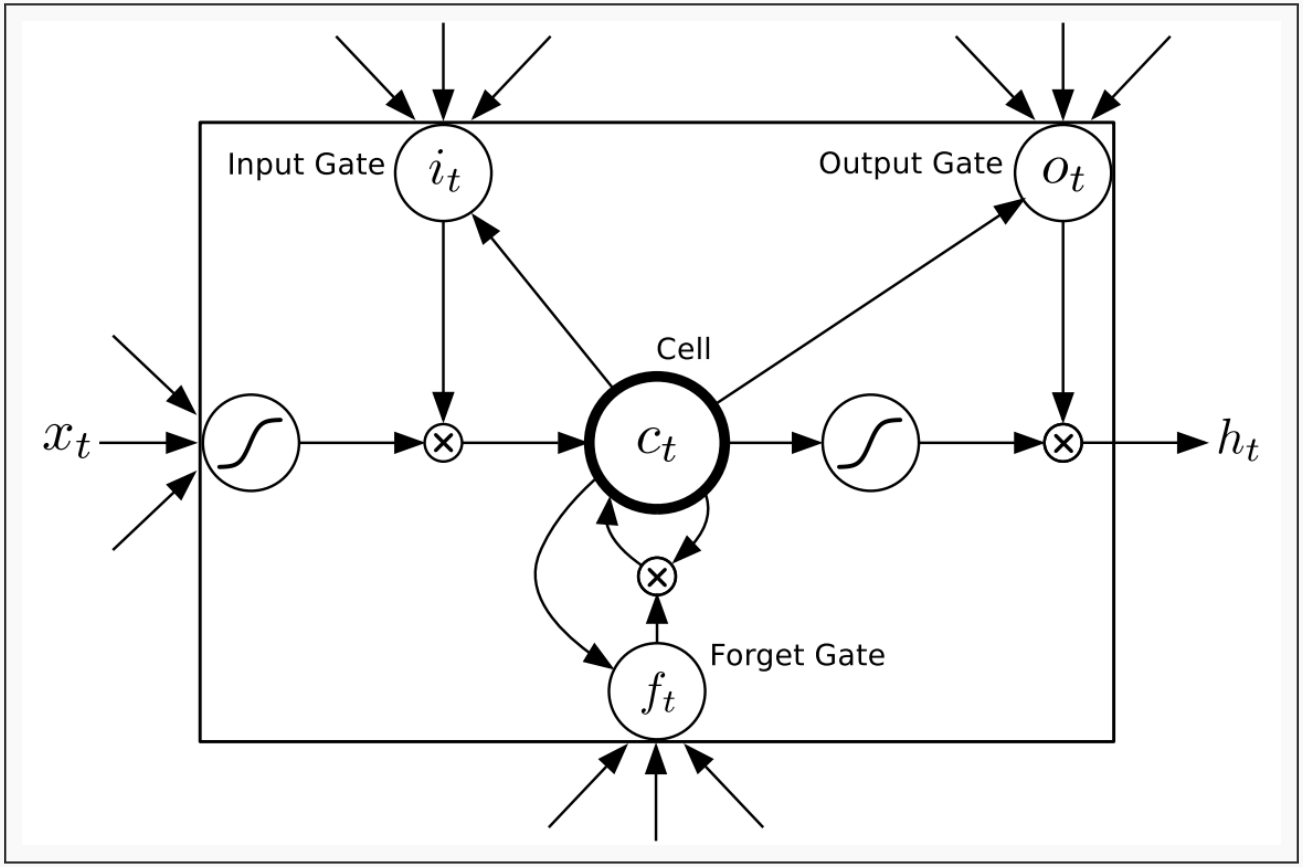


Figure: Traditional LSTM diagram.

LSTM – block diagram 2

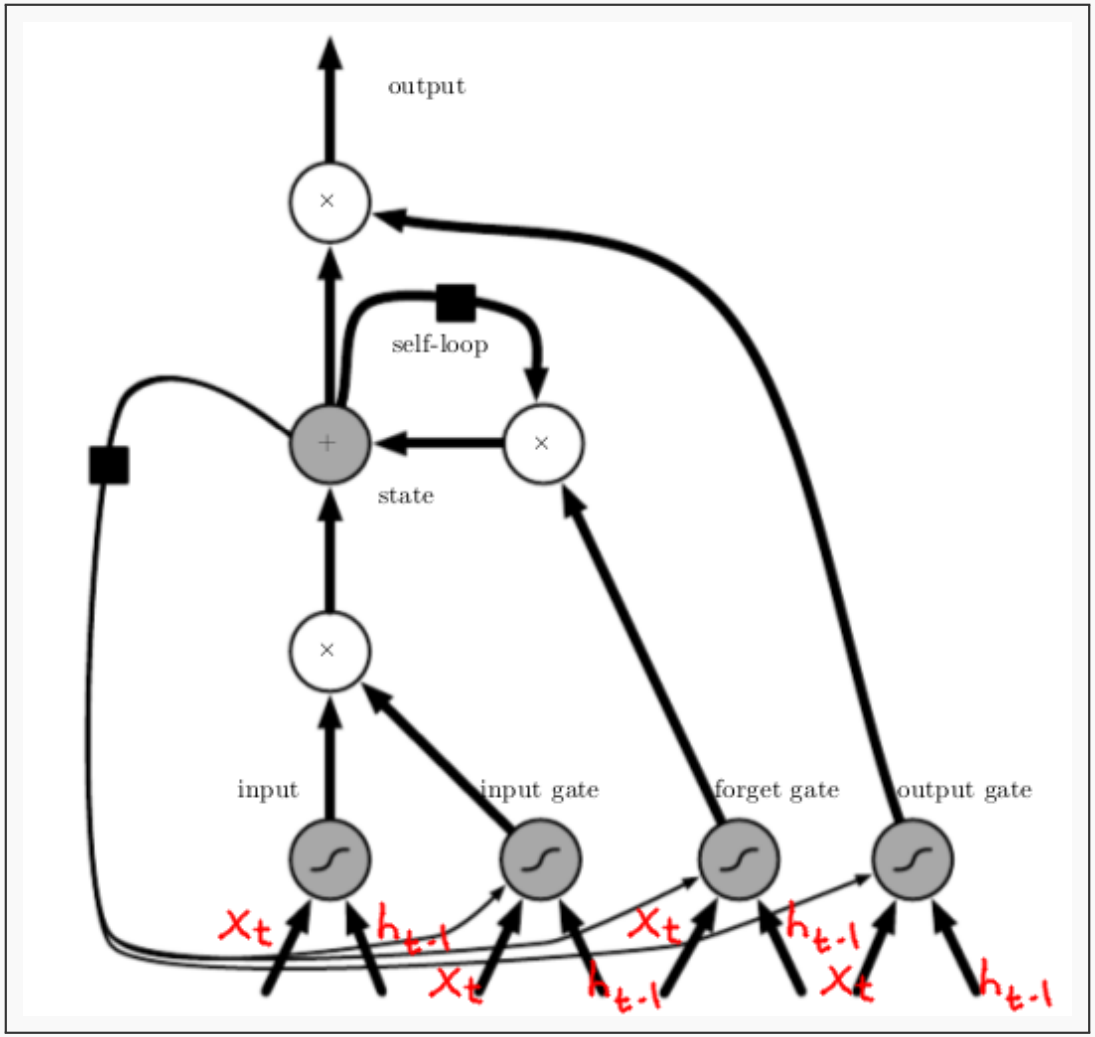


Figure: LSTM diagram as in the Deep Learning Book.

LSTM – block diagram 3

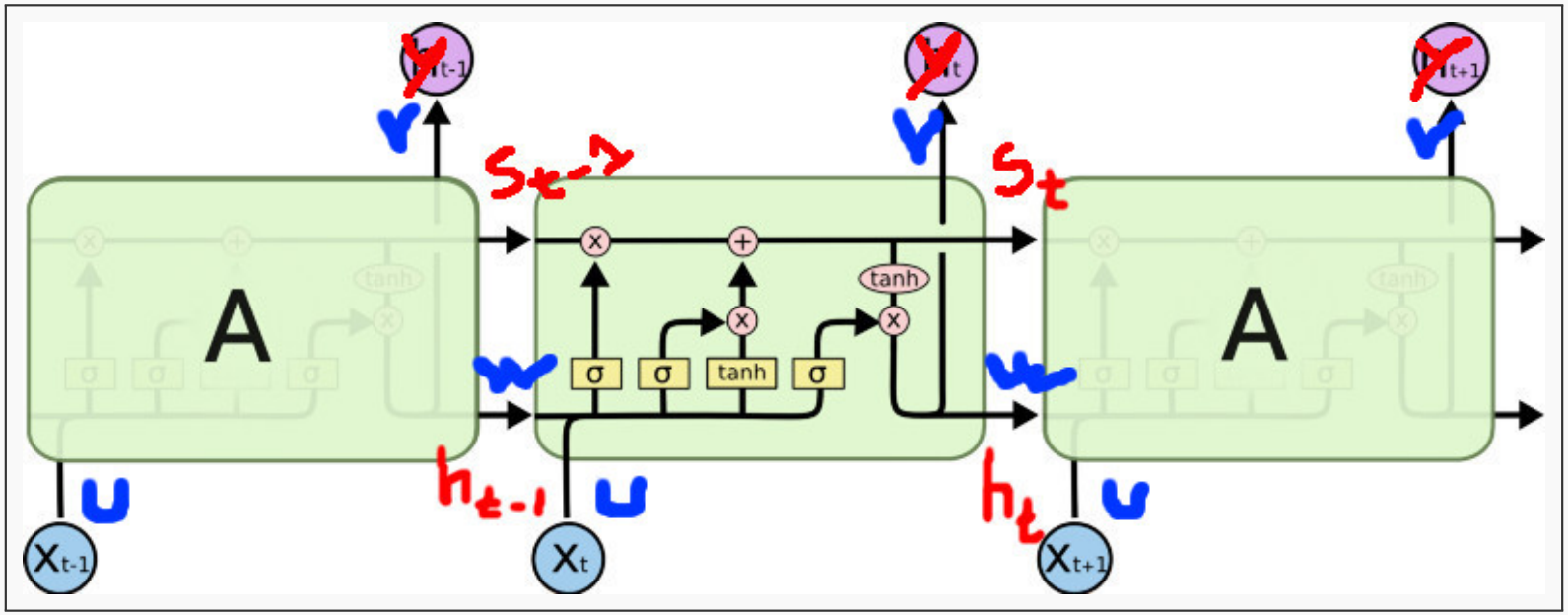


Figure: LSTM diagram as in colah.github.io

- **Two recurrences!**
  $\rightarrow$ Two past informations, through: **W** and direct.

LSTM – formulation

$$
\begin{aligned}
\textit{Input}: \quad & i^{(t)} = \theta(b + Ux^{(t)} + Wh^{(t-1)}) \\
\textit{Gate}: \quad & g_?^{(t)} = \sigma(b_? + U_? x^{(t)} + W_? h^{(t-1)}) \\
\textit{State}: \quad & s^{(t)} = g_f^{(t)} s^{(t-1)} + g_i^{(t)} i^{(t)} \\
\textit{Hidden}: \quad & h^{(t)} = \theta(s^{(t)}) g_o^{(t)} \\
\textit{Output}: \quad & o^{(t)} = c + Vh^{(t)}
\end{aligned}
$$

Where for $g_?^{(t)}$ gate:

? can be **f/i/o** – standing for **forget/input/output**.

Gates use sigmoid nonlinearities: $\sigma(\cdot) \in [0, 1]$
Input/output nonlinearities are typically a $tanh(\cdot)$

LSTM – block diagram 4

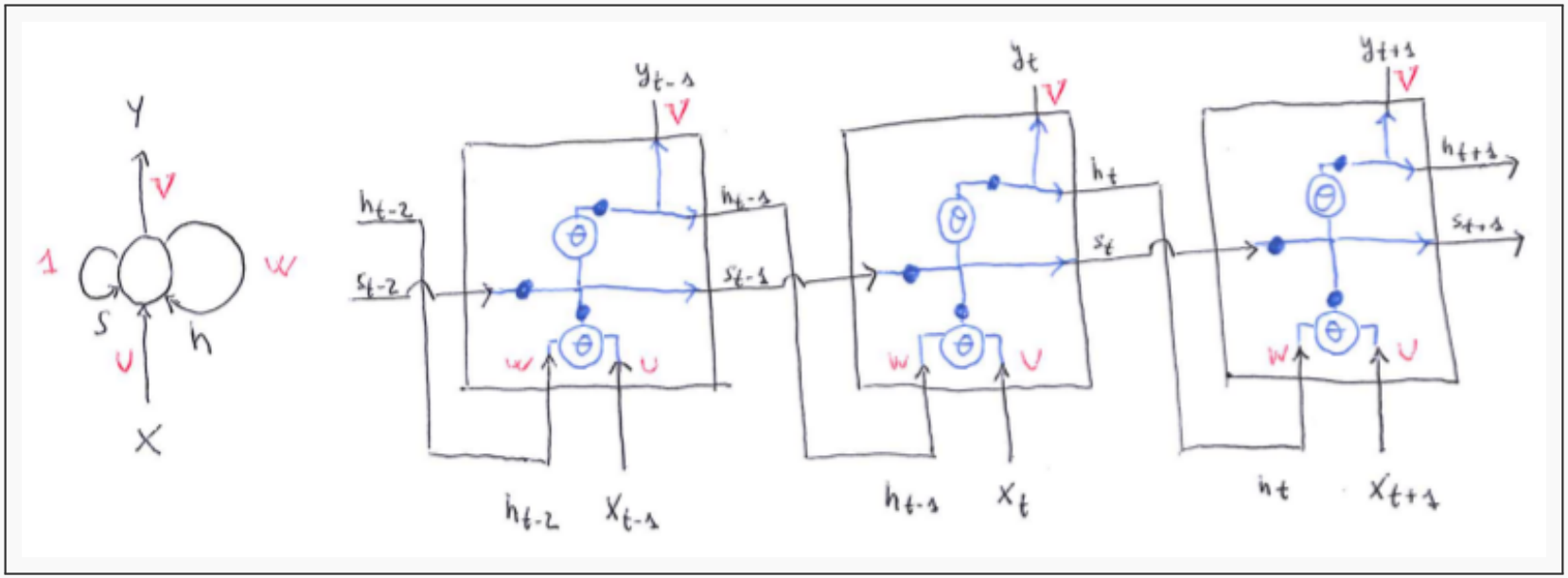


Figure: Simplified diagram – as in my mind.

Blue dots represent gates!

GRU: Gated Recurrent Units

Which pieces of the LSTM architecture are actually necessary?

$$\textit{Vanilla RNN} \rightarrow h^{(t)} = \theta(b + \mathbf{U}\mathbf{x}^{(t)} + \mathbf{W}\mathbf{h}^{(t-1)})$$

$$\textit{LSTM} \rightarrow \textit{complex thing with } s^{(t)} = g_f^{(t)} s^{(t-1)} + g_i^{(t)} i^{(t)}$$

$$\textit{GRU} \rightarrow h^{(t)} = g_u^{(t-1)} h^{(t-1)} + (1 - g_u^{(t-1)})\, \theta(b + \mathbf{U}\mathbf{x}^{(t)} + \mathbf{W} g_r^{(t-1)} \mathbf{h}^{(t-1)})$$

Where $g_u^{(t)}/g_r^{(t)}$ are **update/reset** gates.

Less computation and less number of parameters!

...via removing "intermediate state", and sharing gates!

...while keeping the essence (and performance) of LSTMs!

The problem: gradient vanish/explode

One possible solution: gated units (LSTM & GRU)

Other solutions

Hidden units with linear self-connections

$$Goal: \left\| \frac{\partial \mathbf{h}^{(t+1)}}{\partial \mathbf{h}^{(t+k)}} \right\|_2 \leq (\gamma_w \gamma_\sigma)^{|k-1|} \approx 1$$

Proposal: linear units ($\gamma_\sigma = 1$) and weights near one ($\gamma_w \approx 1$).

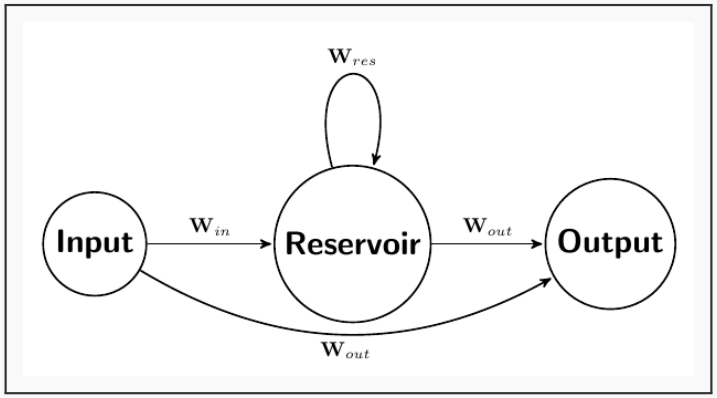
Examples of possible implementations:

- $h^{(t)} \leftarrow \alpha h^{(t-1)} + (1-\alpha)x^{(t)}$ (for a running average?)
- $h^{(t)} \leftarrow w h^{(t-1)} + u x^{(t)}$

$\rightarrow$ When w / α are ≈ 1, it remembers information from the past.
$\rightarrow$ w / α can be fixed or learned.

Echo State Networks

Only learn W_{out}. Keep W_{in} and W_{rec} random!



- **Easy**: train can be a convex optimization problem.
- **Difficult**: set W_{in} and W_{rec}.

$$\left\| \frac{\partial \mathbf{h}^{(t+1)}}{\partial \mathbf{h}^{(t+k)}} \right\|_2 \leq (\gamma_w \gamma_\sigma)^{|k-1|} \qquad \textit{with, tipically} \ : \ \gamma_\sigma \in [0, 1]$$

$$\textit{then, set} : \gamma_w \approx 3$$

Models to operate at multiple time scales

Adding skip connections through time

- Connections with a time-delay try to mitigate the gradient vanish or explode problem.
- Gradients may still vanish/explode. For some intuition, think about the *problematic* forward path: W^t.
- Learn influenced by the past.

Removing connections

- Forcing to operate at longer (coarse) time dependencies.
- For some intuition, think about the *problematic* forward path: W^t where the t horizon is kept, without paying the cost of doing all the multiplications.

Removing connections to operate at longer time-scales

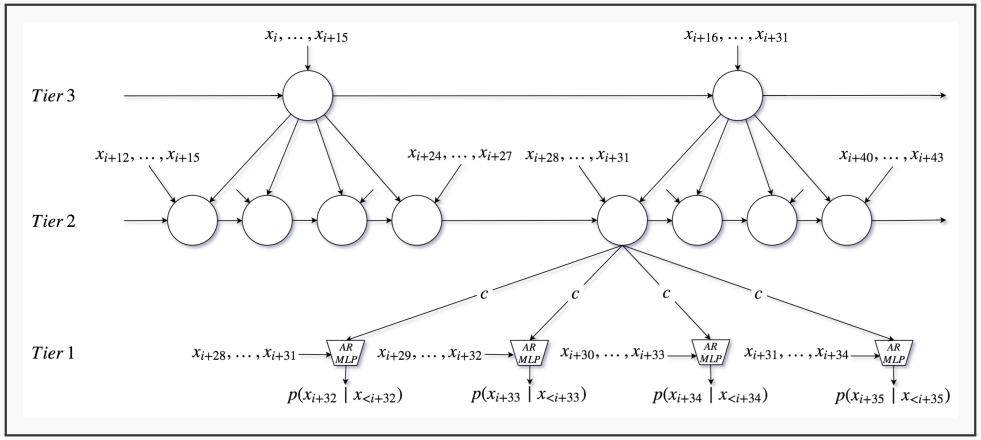


Figure: SampleRNN

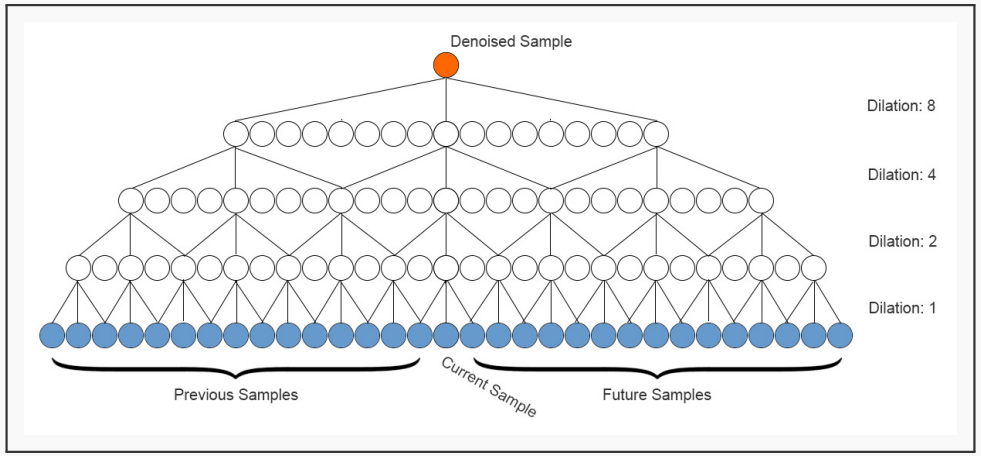


Figure: Non-causal Wavenet

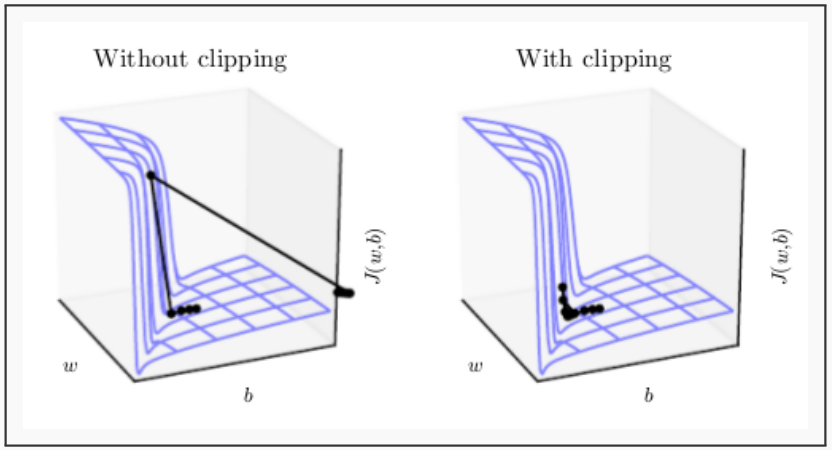
Optimization strategies for RNNs: second order methods

Why don't we improve the optimization by using second order methods?

- These have high computational cost.
- Require a large mini-batch.
- Tendency to be attracted by saddle points.
- **Simpler methods with careful initialization can achieve similar results!**

Research done during 2011-2013.

Optimization strategies for RNNs: clipping gradients



Basic idea:

- To **bound** the gradient per minibatch.
- Avoids doing a detrimental step when the **gradient explodes**.

Introduces an heuristic bias that is known to be useful.

Wrapping up..

The problem: gradient vanish/explode

One possible solution: gated units (LSTM & GRU)

Other solutions

..thanks! :)

Credit: most figures are from the ***Deep Learning Book***.
It is also useful to see this ***video*** by Nando de Freitas.

Tutorial on recurrent neural networks

Jordi Pons

Researcher at Dolby Laboratories
@jordiponsdotme – www.jordipons.me

Master in Sound and Music Computing
Universitat Pompeu Fabra, Winter 2019