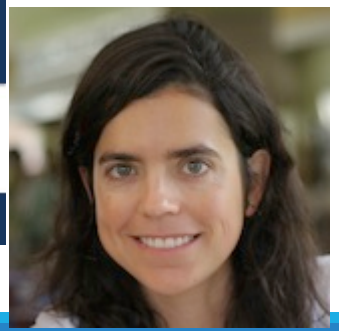
#DLUPC

Day 3 Lecture 2

Recurrent neural networks



Marta R. Costa-jussà
marta.ruiz@upc.edu

Ramón y Cajal Researcher
Universitat Politecnica de Catalunya
Technical University of Catalonia

telecom BCN

UPC

[course site]

Outline

1. The importance of context. Sequence modeling
2. Feed-forward networks review
3. Vanilla RNN
4. Vanishing gradient

1.The importance of context

The importance of context. Sequence modeling

- Recall the 5th digit of your phone number
- Sing your favourite song beginning at third sentence
- Recall 10th character of the alphabet

Probably you went straight from the beginning of the stream in each case… because in sequences order matters!

Idea: retain the information preserving the importance of order

Language Applications

- Language Modeling (probability)
- Machine Translation
- Speech Recognition

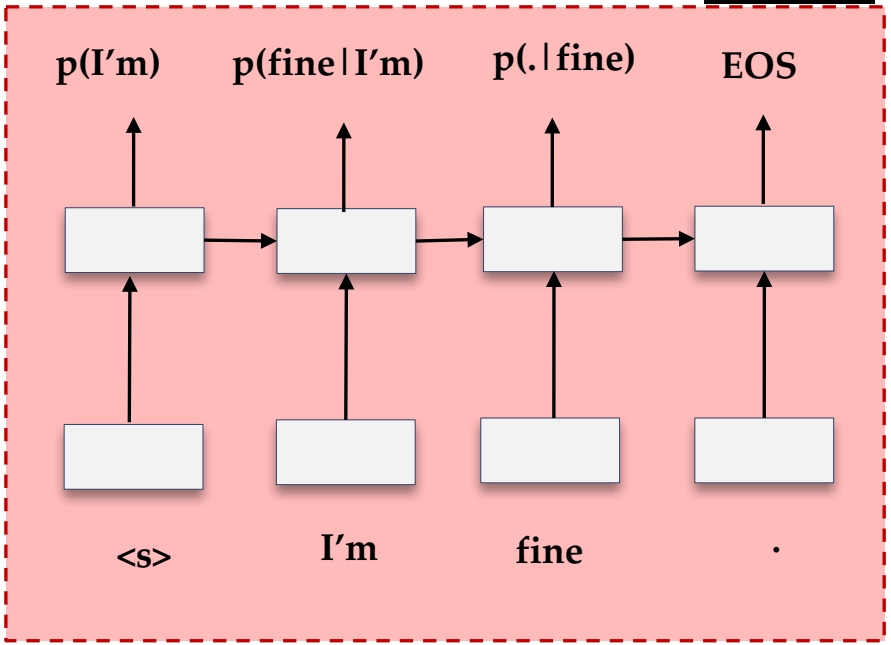


Image Applications

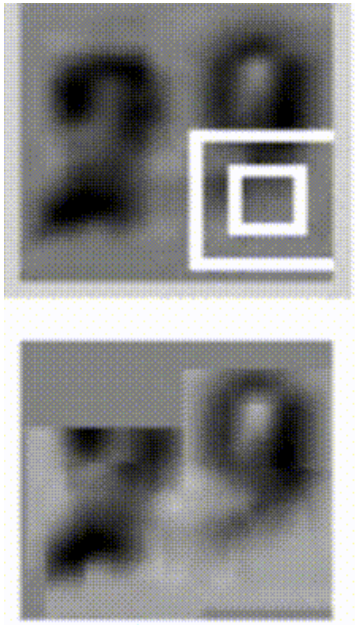


Image credit: The Unreasonable Effectiveness of RNNs, André Karpathy

2. Feedforward Network Review

Feedforward Networks

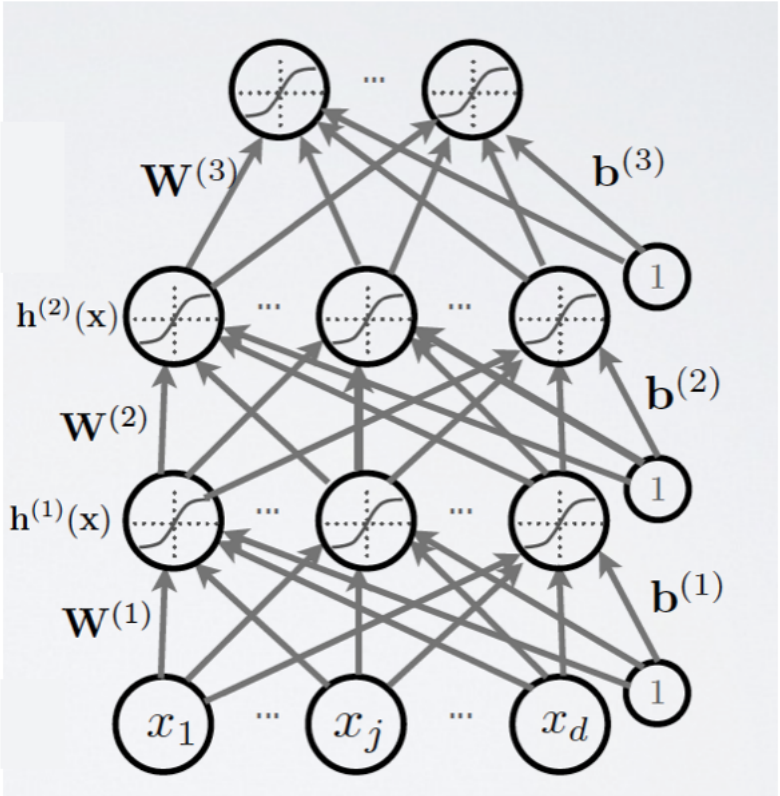
Every **y/hi** is computed from the sequence of forward activations out of input **x**.

$$y = f(W_3 \cdot h_2 + b_3)$$

$$h_2 = f(W_2 \cdot h_1 + b_2)$$

$$h_1 = f(W_1 \cdot x + b_1)$$

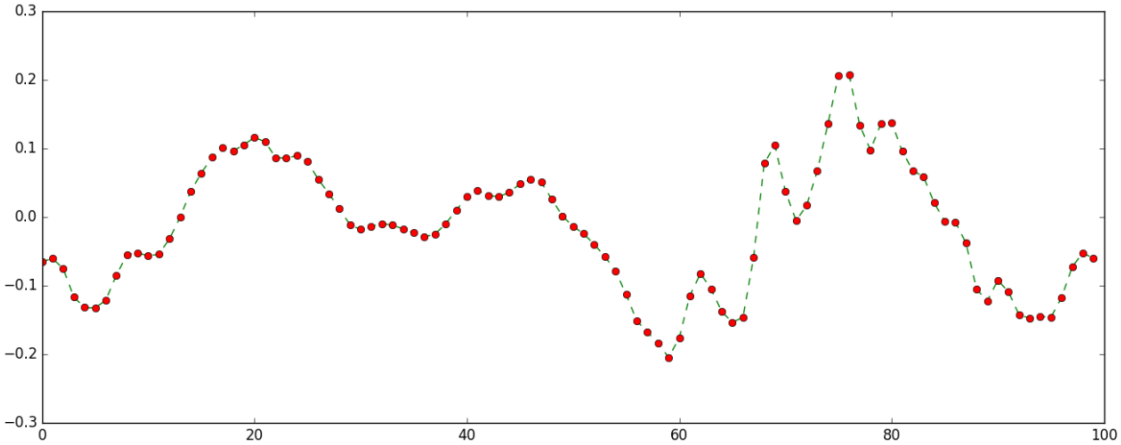
Feed FORWARD



Slide Credit: Hugo Laroche NN course

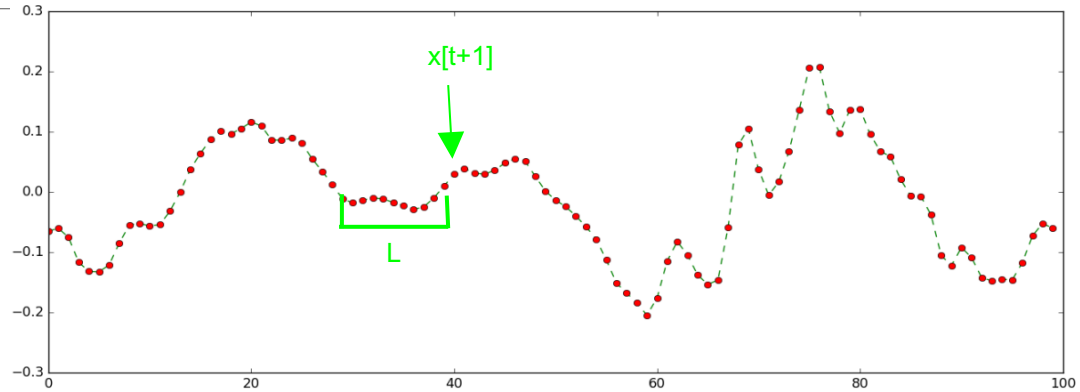
Where is the Memory I?

If we have a sequence of samples...



predict sample x[t+1] knowing previous values {x[t], x[t-1], x[t-2], …, x[t-τ]}

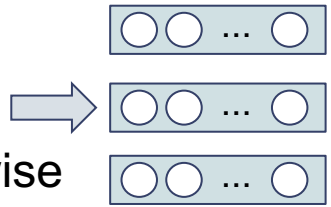
Where is the Memory II?



x[t+1]

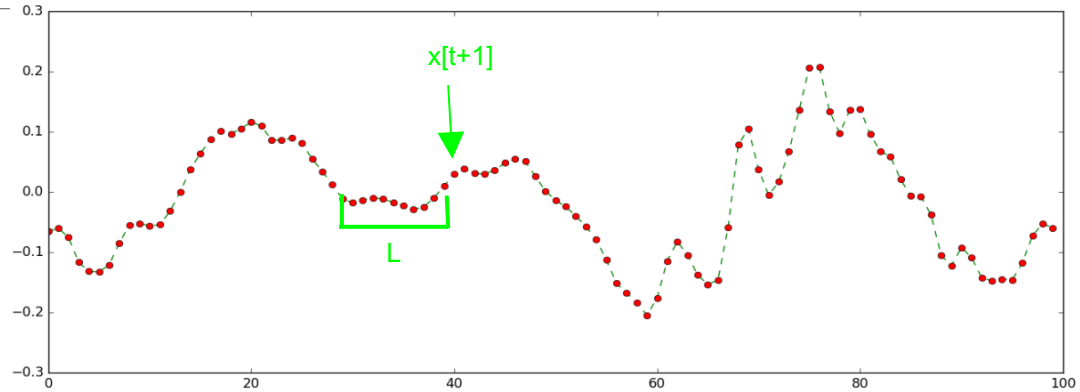
Feed Forward approach:

- static window of size L
- slide the window time-step wise



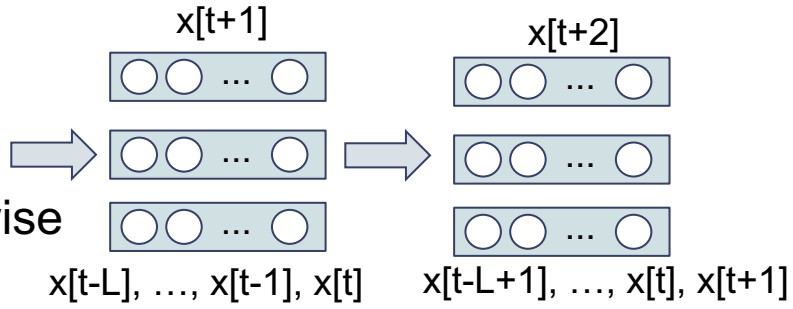
x[t-L], …, x[t-1], x[t]

Where is the Memory III?

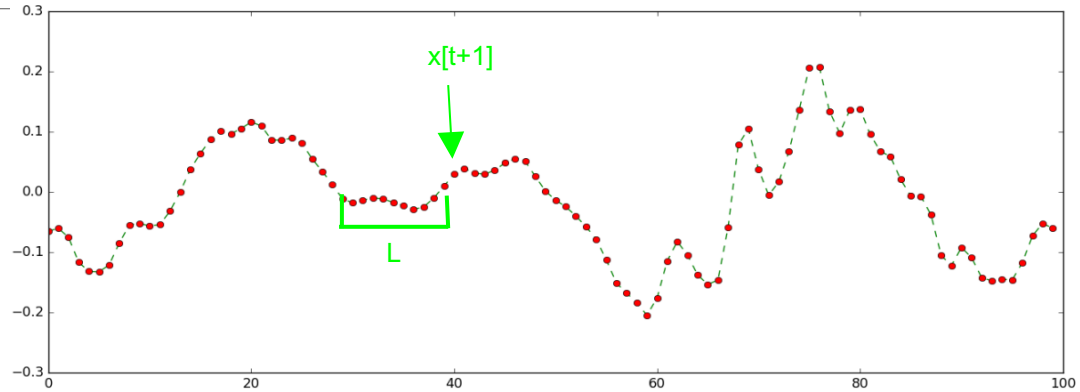


Feed Forward approach:

- static window of size L
- slide the window time-step wise

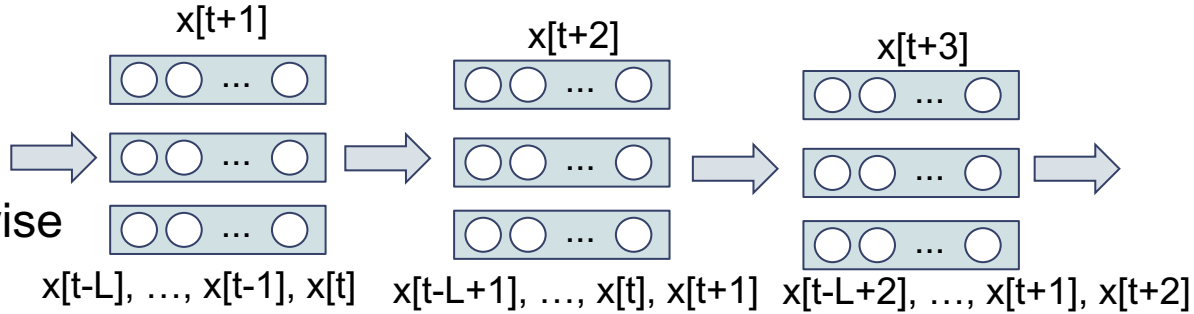


Where is the Memory IV?



Feed Forward approach:

- static window of size L
- slide the window time-step wise



Any problems with this approach?

Problems for the FNN + static window approach I

- If increasing L, fast growth of num of parameters!

Problems for the FNN + static window approach II

- If increasing L, fast growth of num of parameters!
- Decisions are independent between **time-steps**!
  - The network doesn't care about what happened at previous time-step, only present window matters → doesn't look good

TIME-STEPS: the memory do you want to include in your network.
If you want your network to have memory of 60 characters, this number should be 60.

Problems for the FNN + static window approach III

- If increasing L, fast growth of num of parameters!
- Decisions are independent between time-steps!
  - The network doesn't care about what happened at previous time-step, only present window matters → doesn't look good
- Cumbersome padding when there are not enough samples to fill L size
  - Can't work with variable sequence lengths

3. Vanilla RNN

Recurrent Neural Network (RNN) adding the “temporal” evolution

Allow to build specific connections capturing ”history”

$$h_t = f(W \cdot x_t + U \cdot h_{t-1} + b)$$

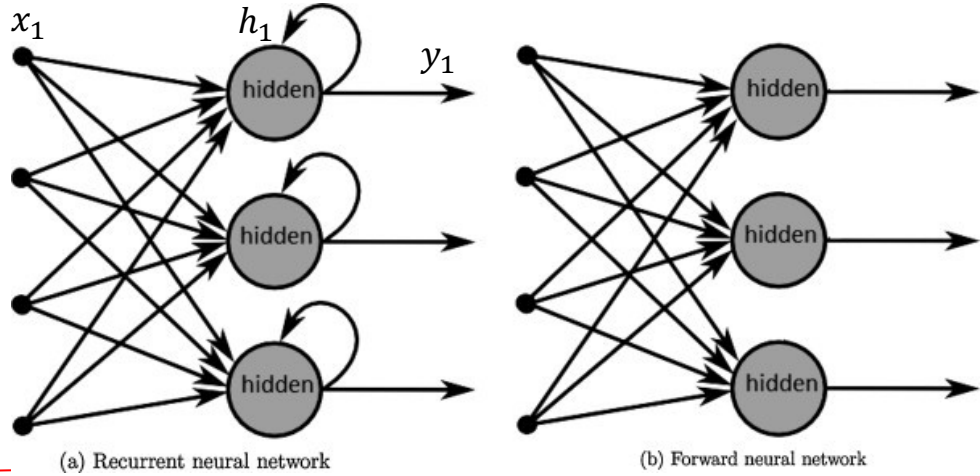
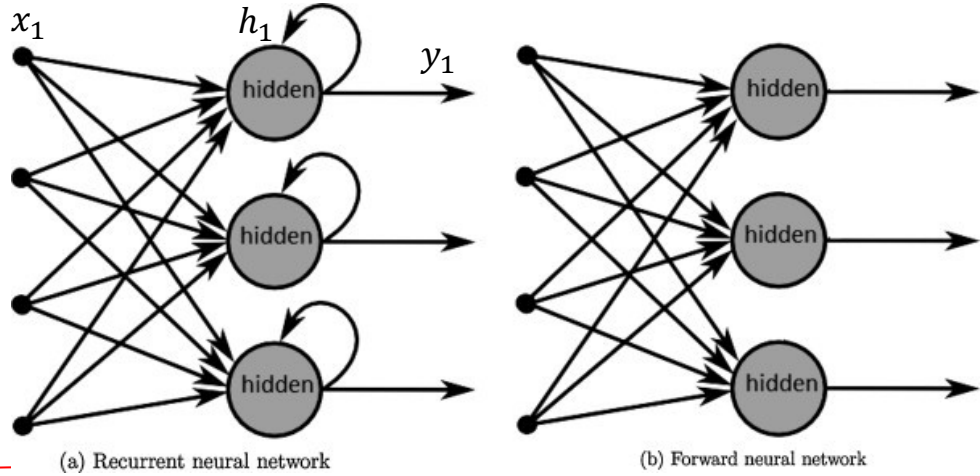


Image src: https://medium.com/towards-data-science/introduction-to-recurrent-neural-network-27202c3945f3

Recurrent Neural Network (RNN) adding the “temporal” evolution

Allow to build specific connections capturing ”history”

$$h_t = f(W \cdot x_t + U \cdot h_{t-1} + b)$$

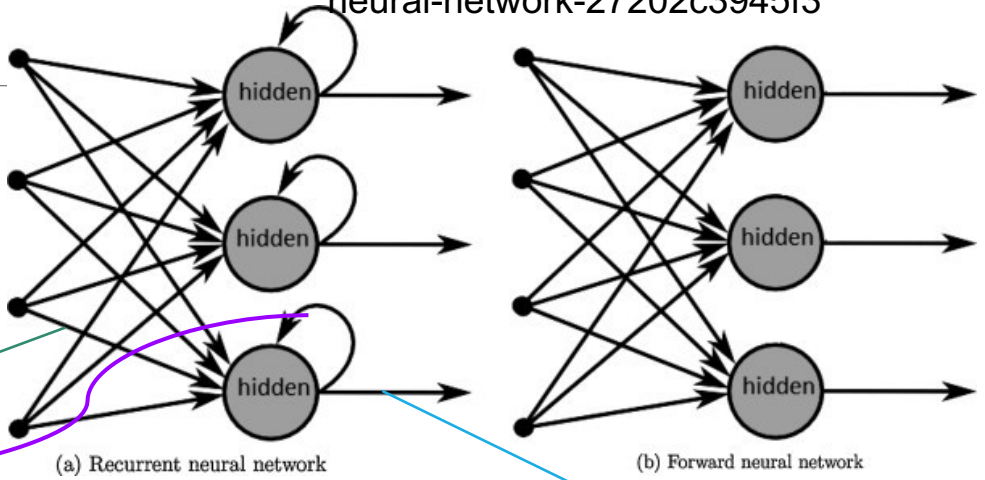


$$y_t = softmax(Vh_t)$$

Image src: https://medium.com/towards-data-science/introduction-to-recurrent-neural-network-27202c3945f3

RNN: parameters

Image src: https://medium.com/towards-data-science/introduction-to-recurrent-neural-network-27202c3945f3



(a) Recurrent neural network

(b) Forward neural network

$$h_t = f(W \cdot x_t + U \cdot h_{t-1} + b)$$

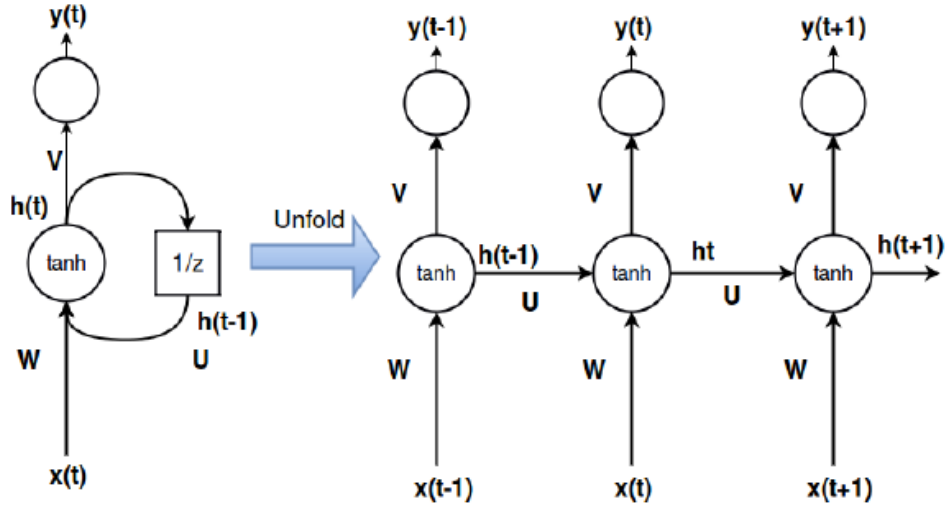
$$y_t = softmax(Vh_t)$$

$$N^i_{params} = N^i_{inputs} \times N^i_{units} + N^i_{units} \times N^i_{units} + N^i_{units}$$

$$N^i_{units} \times N^i_{outputs}$$

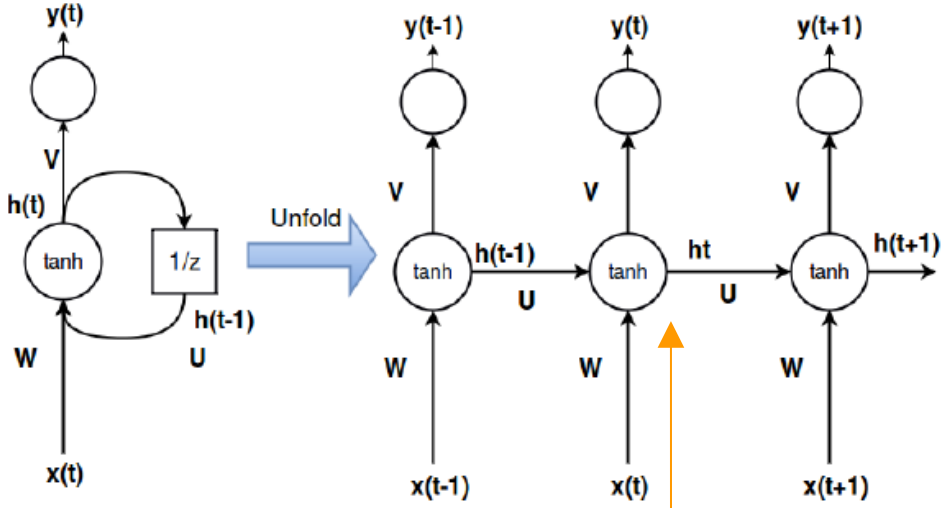
RNN: unfolding

BEWARE: We have extra depth now! Every time-step is an extra level of depth (as a deeper stack of layers in a feed-forward fashion!)



RNN: depth 1

Forward in space propagation

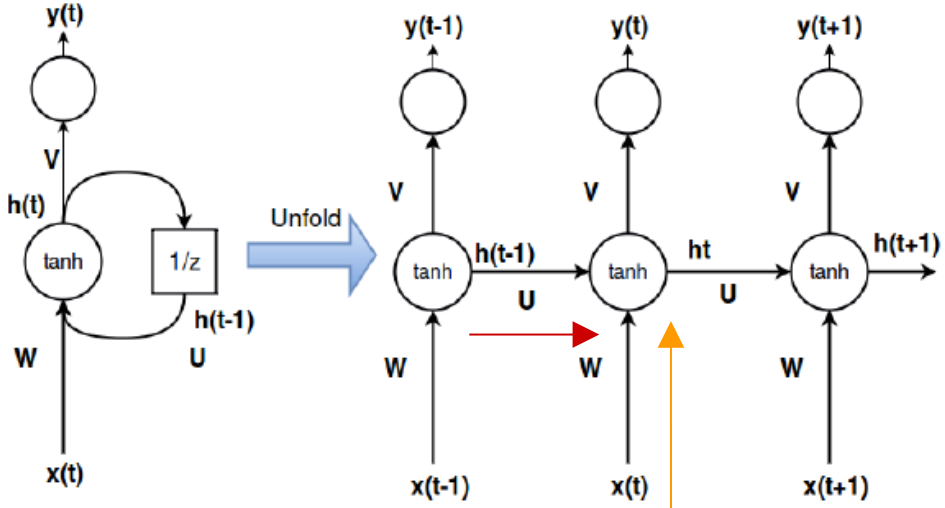


$$\mathbf{h}_t = g(\mathbf{W} \cdot \mathbf{x}_t + \mathbf{U} \cdot \mathbf{h}_{t-1} + \mathbf{b}_h)$$

$\mathbf{x}_t \in \mathbb{R}^n$

RNN: depth 2

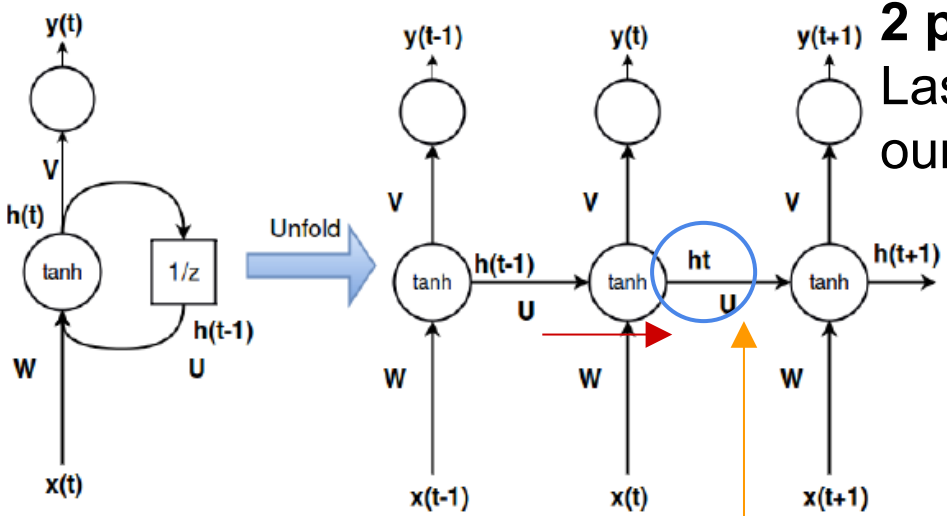
Forward in time propagation



$$\mathbf{h}_t = g(\mathbf{W} \cdot \mathbf{x}_t + \mathbf{U} \cdot \mathbf{h}_{t-1} + \mathbf{b}_h)$$

$\mathbf{x}_t \in \mathbb{R}^n$

RNN: depth in space + time



Hence we have two data flows:
Forward in space + time propagation:
2 projections per layer activation
Last time-step includes the context of our decisions recursively

W, U, V shared across all steps

$$\mathbf{h}_t = g(\mathbf{W} \cdot \mathbf{x}_t + \mathbf{U} \cdot \mathbf{h}_{t-1} + \mathbf{b}_h)$$

$$\boldsymbol{y_t = softmax(V h_t)}$$

$\mathbf{x}_t \in \mathbb{R}^n$

Alternatives of recurrence

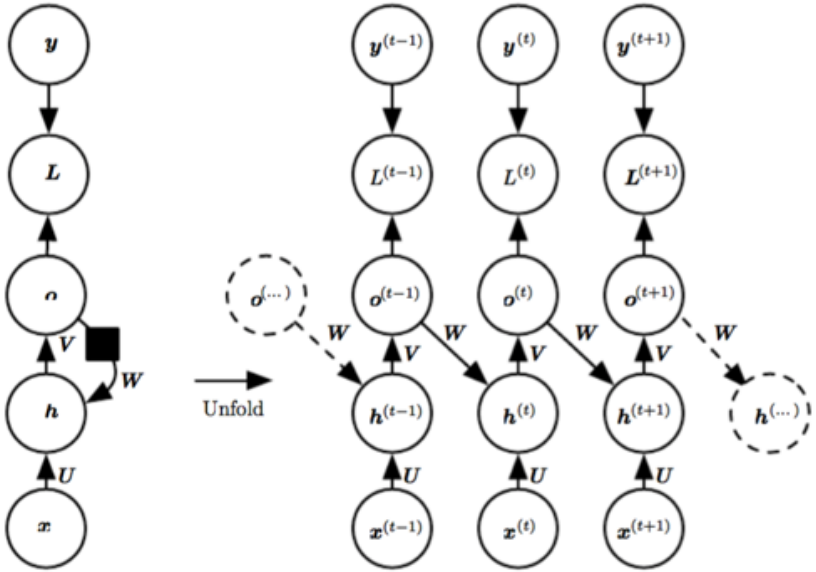


Image src: Goodfellow et al. Deep Learning. The MIT Press

Alternatives of outputs

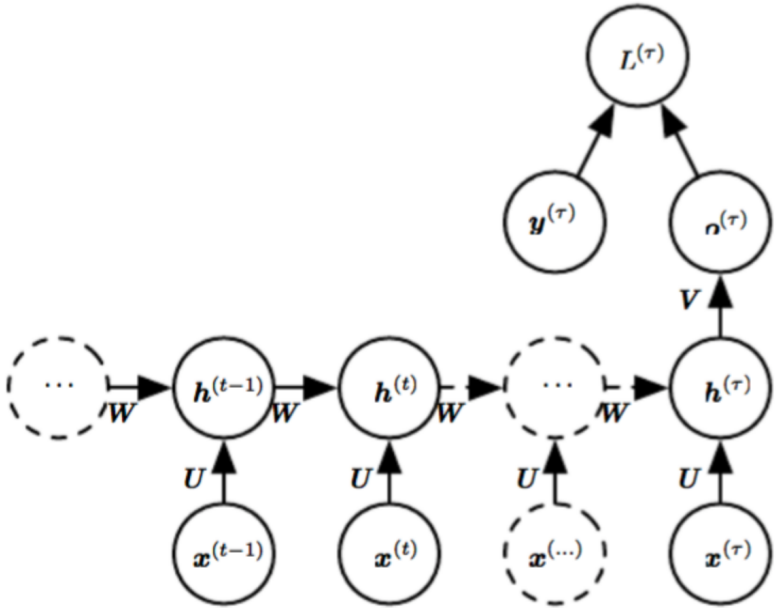


Image src: Goodfellow et al. Deep Learning. The MIT Press

Training a RNN I: BPTT

Backpropagation through time (BPTT): The training algorithm for updating network weights to minimize error including time

Remember BackPropagation

1. Present a training input pattern and propagate it through the network to get an output.
2. Compare the predicted outputs to the expected outputs and calculate the error.
3. Calculate the derivatives of the error with respect to the network weights.
4. Adjust the weights to minimize the error.
5. Repeat.

BPTT I

Cross Entropy Loss

$$\boldsymbol{h_t} = f(\boldsymbol{W} \cdot \boldsymbol{x_t} + \boldsymbol{U} \cdot \boldsymbol{h_{t-1}} + \boldsymbol{b})$$

$$y_t = softmax(Vh_t)$$

$$\begin{aligned} E_t(y_t, \hat{y}_t) &= -y_t \log \hat{y}_t \\ E(y, \hat{y}) &= \sum_t E_t(y_t, \hat{y}_t) \\ &= -\sum_t y_t \log \hat{y}_t \end{aligned}$$

BPTT II

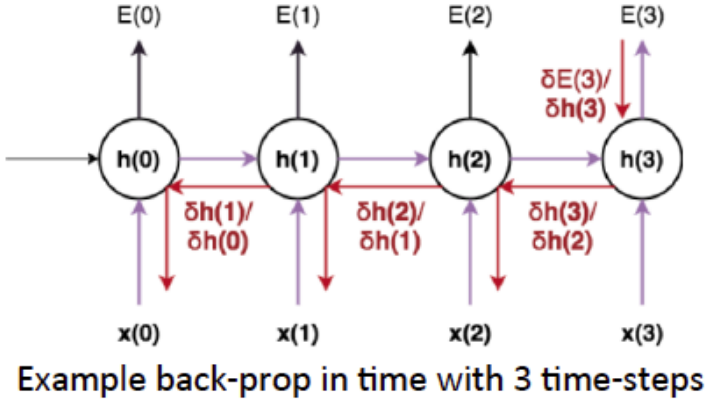
$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}.$$

NOTE: our goal is to calculate the gradients of the error with respect to our parameters U, W and V and and then learn good parameters using Stochastic Gradient Descent.
Just like we sum up the errors, we also sum up the gradients at each time step for one training example:

BPTT III (E3 computation)

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial h_3}\frac{\partial h_3}{\partial W}$$

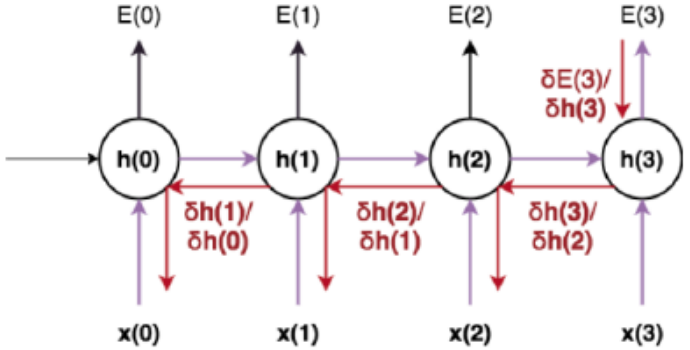
$$h_3 = f(Ux_t + Wh_2)$$



Example back-prop in time with 3 time-steps

BPTT IV

$$\frac{\partial E_3}{\partial W} = \sum_{h=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial h_3} \frac{\partial h_3}{\partial h_k} \frac{\partial h_k}{\partial W}$$



Example back-prop in time with 3 time-steps

4. Vanishing gradient

Example

Maria has studied law and she is a lawyer.

Vanishing gradient I

- **During training gradients explode/vanish easily because of depth-in-time** → Exploding/Vanishing gradients!

Vanishing gradient II

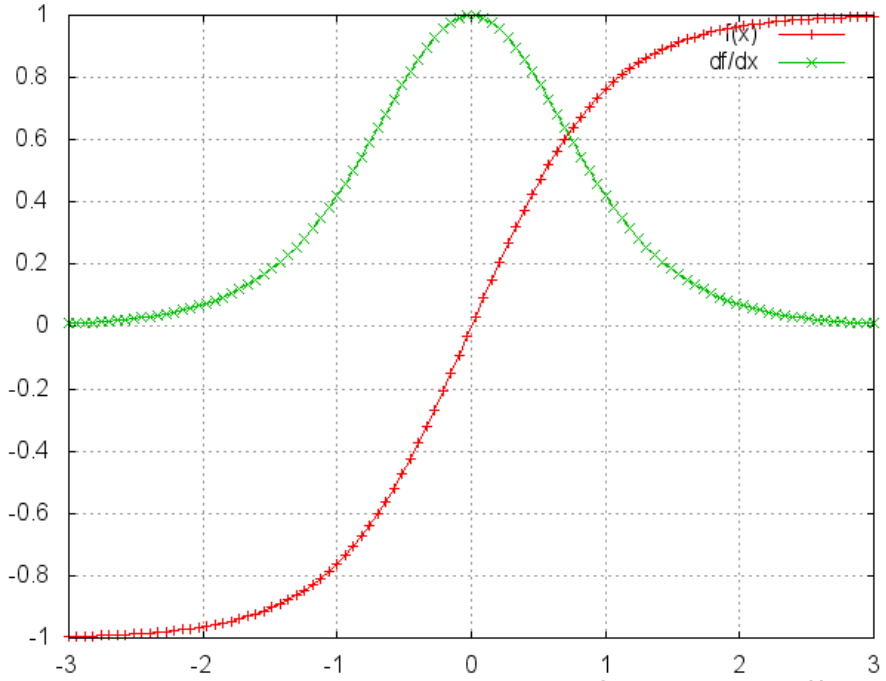
For example,

$$\frac{\partial E_3}{\partial W} = \sum_{h=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial h_3} \frac{\partial h_3}{\partial h_k} \frac{\partial h_k}{\partial W}$$

$$\frac{\partial h_3}{\partial h_1} = \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1}$$

$$\frac{\partial E_3}{\partial W} = \sum_{h=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial h_3} \left(\prod_{j=k+1}^{3} \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial W}$$

Vanishing gradient III



tanh and derivative. Source: http://nn.readthedocs.org/en/rtd/transfer/

$$\frac{\partial E_3}{\partial W} = \sum_{h=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial h_3} \left(\prod_{j=k+1}^{3} \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial W}$$

Standard Solutions

Proper initialization of Weight Matrix

Regularization of outputs or Dropout

Use of ReLU Activations as it’s derivative is either 0 or 1

Thanks ! Q&A ?