

AA 2022/2023

Machine Learning for Modelling: *Supervised Learning*

Simone Bianco

1

AA 2022/2023

Recurrent and Recursive Neural Networks (RNNs)

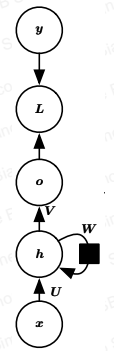
2

RNNs

- Recurrent neural networks or RNNs are a family of neural networks for processing **sequential data**.
- Much as a convolutional network is a neural network that is specialized for processing a grid of values X such as an image, a recurrent neural network is a neural network that is specialized for processing a sequence of values $x(1), \dots, x(\tau)$.
- Just as convolutional networks can readily scale to images with large width and height, and some convolutional networks can process images of variable size, recurrent networks can scale to much longer sequences than would be practical for networks without sequence-based specialization.
- Most recurrent networks can also process sequences of variable length.

3

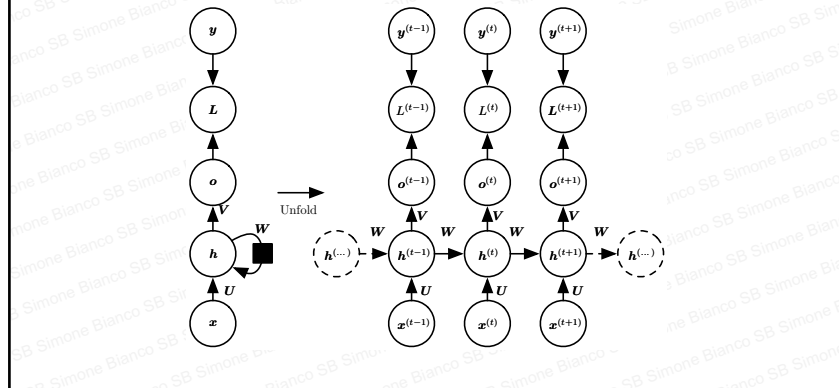
RNNs



- Computational graph to compute the training loss of a recurrent network that maps an input sequence of x values to a corresponding sequence of output o values.
- A loss L measures how far each o is from the corresponding training target y .
- When using softmax outputs, we assume o is the unnormalized log probabilities.
- The loss L internally computes $\hat{y} = \text{softmax}(o)$ and compares this to the target y .
- The RNN has input-to-hidden connections parametrized by a weight matrix U , hidden-to-hidden recurrent connections parametrized by a weight matrix W , and hidden-to-output connections parametrized by a weight matrix V .

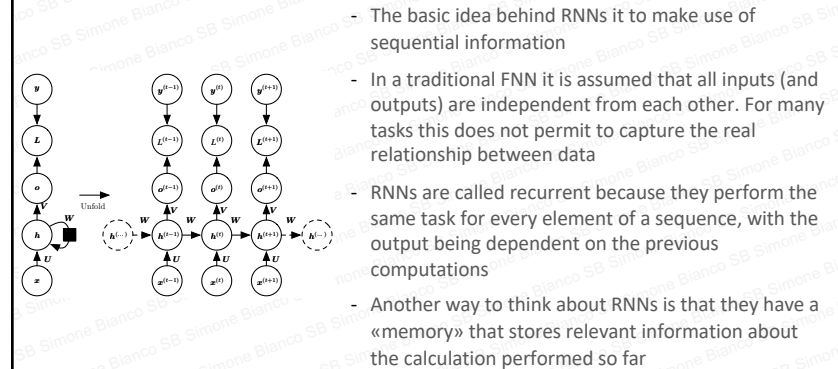
4

Unfolding the computational graph of RNNs



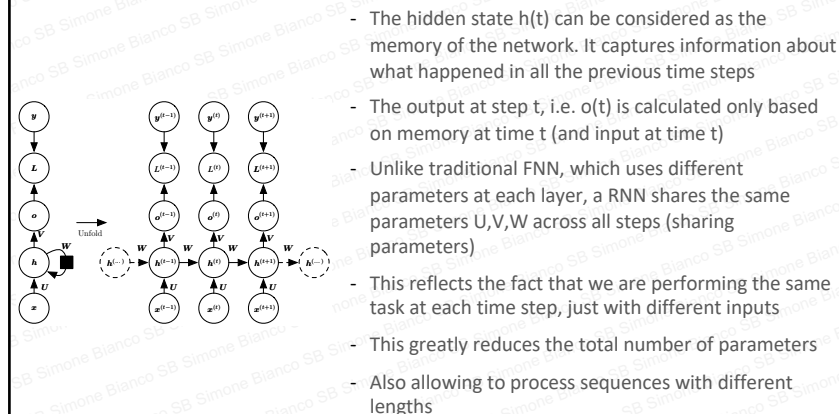
5

Unfolding the computational graph of RNNs



6

Unfolding the computational graph of RNNs



7

RNNs

- The previous scheme considers outputs at each time step, but depending on the task we are facing this may be not necessary (e.g., we want to analyze a video and classify its content)

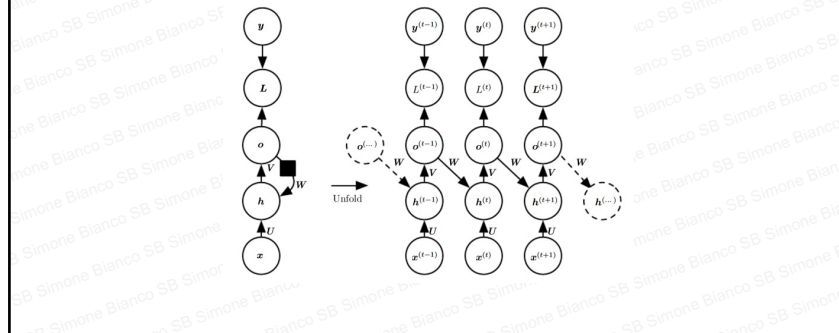
Some of the most relevant RNN architectures are:

- RNNs that produce an output at each time step and have recurrent connections between hidden units (i.e., the one seen in the previous slide)
- RNNs that produce an output at each time step and have recurrent connections only from the output at one time step to the hidden units at the next time step
- RNNs with recurrent connections between hidden units, that analyze an entire sequence and then produce a single output

8

RNNs

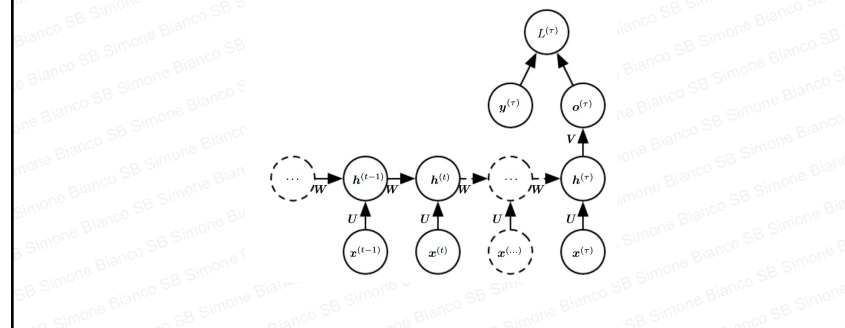
RNNs that produce an output at each time step and have recurrent connections only from the output at one time step to the hidden units at the next time step.



9

RNNs

RNNs with recurrent connections between hidden units, that analyze an entire sequence and then produce a single output



10

Deep RNNs

The computation in most RNNs can be decomposed into 3 blocks of parameters and associated transformations:

1. From the input to the hidden state
2. From the previous hidden state to the next hidden state
3. From the hidden state to the output

For the RNNs architectures seen so far each of the 3 blocks is associated with a single weight matrix

Would it be advantageous to introduce depth in each of these operations?

Experimental evidence is in agreement with the idea that we need enough depth in order to perform the required mappings.

11

Deep RNNs

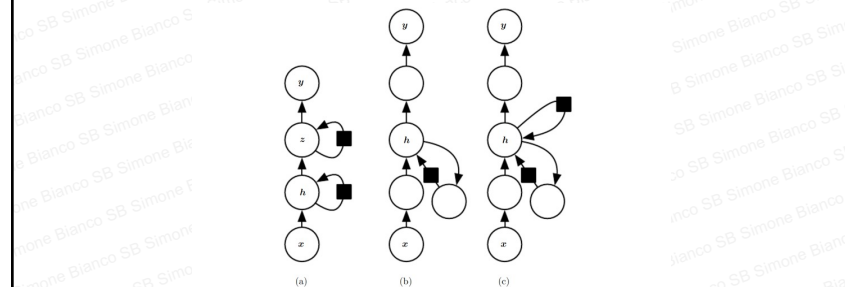


Figure 10.13: A recurrent neural network can be made deep in many ways (Pascanu et al., 2014a). (a) The hidden recurrent state can be broken down into groups organized hierarchically. (b) Deeper computation (e.g., an MLP) can be introduced in the input-to-hidden, hidden-to-hidden and hidden-to-output parts. This may lengthen the shortest path linking different time steps. (c) The path-lengthening effect can be mitigated by introducing skip connections.

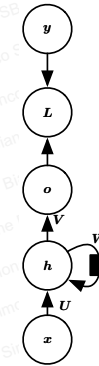
12

Training a RNN

- Training a RNN is similar to training a traditional Neural Network
- Backpropagation algorithm is used, but with a little twist
- Since the parameters are shared by all the time steps in the network, the gradient at each step depends on both:
 - the calculation of the current time step
 - the calculation of the previous time step
- For example: in order to calculate the gradient at the time step $t=4$ we need to backpropagate gradients to the previous 3 time steps and sum all the gradients
- This is why the RNN training algorithm is called Backpropagation Through Time (BPTT)

13

BPTT algorithm (1)



- For notation purposes let's use $o^{(t)} = \hat{y}^{(t)}$
- Then consider:

$$h^{(t)} = \tanh(Ux^{(t)} + Wh^{(t-1)})$$

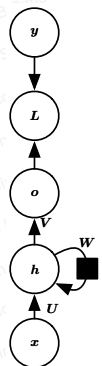
$$\hat{y}^{(t)} = \text{softmax}(Vh^{(t)})$$
- And the cross entropy as the loss (i.e., error) function to minimize:

$$E^{(t)}(y^{(t)}, \hat{y}^{(t)}) = -y^{(t)} \log(\hat{y}^{(t)})$$

$$E(y, \hat{y}) = \sum_t E^{(t)}(y^{(t)}, \hat{y}^{(t)}) = -\sum_t y^{(t)} \log(\hat{y}^{(t)})$$

14

BPTT algorithm (2)



- The goal of BPTT is to calculate the gradients of the error wrt U, V, W and then learn good values of these 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:

$$\frac{\partial E}{\partial U} = \sum_t \frac{\partial E^{(t)}}{\partial U}, \text{ and } \frac{\partial E}{\partial V} = \sum_t \frac{\partial E^{(t)}}{\partial V}, \text{ and } \frac{\partial E}{\partial W} = \sum_t \frac{\partial E^{(t)}}{\partial W}$$
- To calculate these gradients, we use the differentiation chain rule, that is the (traditional) backpropagation algorithm
- While, depending on the RNN architecture, the computation maybe quite simple for $\frac{\partial E}{\partial V}$, the chain rule is much more complicated for $\frac{\partial E}{\partial U}$ and $\frac{\partial E}{\partial W}$
- Since we need to backpropagate the gradients from the output through the network all the way back to $t=0$

15

Vanishing gradients problem

- Computing the gradient with respect to $h^{(0)}$ involves many factors and repeated gradient computations
- It may happen that we have many values > 1 :
 - Exploding gradients

How can we solve them?

- Gradient clipping to scale big gradients

- It may happen that we have many values < 1 :
 - Vanishing gradients

How can we solve them?

1. Activation function
2. Weight initialization
3. Network architecture

16

Vanishing gradients problem

- Why are vanishing gradients a problem?

Multiply many small numbers together

Errors due to further back time steps have smaller and smaller gradients

Bias parameters to capture short-term dependencies

17

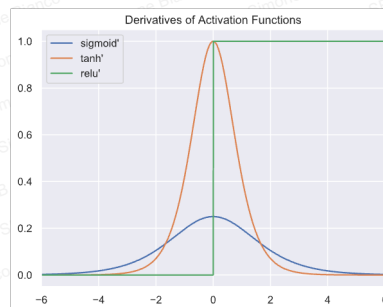
Vanishing gradients problem

- RNNs are not able to learn long-term dependencies: gradients contributions from «far away» steps become zero, and the state at those steps does not contribute to what you are learning!
- Vanishing gradients also happen in deep Feedforward Neural Networks (recall the introduction of ResNets)
- RNNs tend to be very deep (as deep as the sequence length), which makes the problem much more frequent

18

Vanishing gradients problem

Solution #1: Activation function



ReLU prevents f' to shrink gradients when $x > 0$

19

Vanishing gradients problem

Solution #2: Parameter initialization

- Initialize weights to identity matrix
- Initialize biases to zero
- This helps preventing the weights from shrinking to zero

$$I_n = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix}$$

20

Vanishing gradients problem

Solution #3: Network architecture

- Use a more complex recurrent unit with gates to control what information is passed through

21

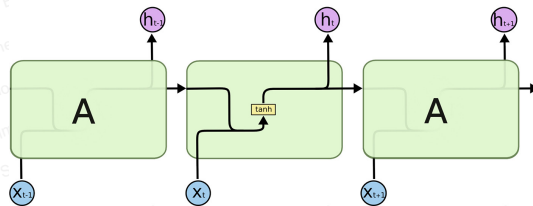
Gated RNNs

- The most effective sequence models used in practical applications are so called gated RNNs.
- They include both Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU)
- Gated RNNs are based on the idea of creating paths through time that have derivatives that neither vanish nor explode
- To achieve this goal, they use connection weights that may change at every time step

22

LSTM Long Short-Term Memory

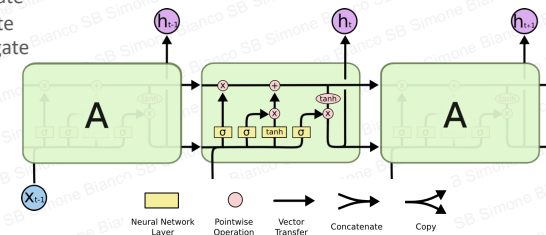
In standard RNNs, repeating modules contain a simple computation node



23

LSTM Long Short-Term Memory

- The LSTM model is organized in cells which include several operations
- LSTM has an internal state variable, which is passed from one cell to another and modified by the following Operation Gates:
 - Forget gate
 - Input gate
 - Output gate

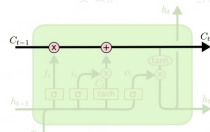


24

LSTM Long Short-Term Memory

Cell State

- Maintains a vector C_t that has the same dimensionality as the hidden state h_t
- Information can be added or deleted from this state vector via the forget gate or the input gate

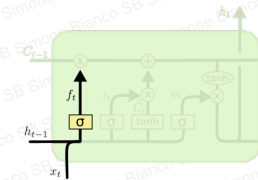


25

LSTM – Forget Gate

- It is a sigmoid layer that takes the hidden state at timestep t-1, i.e. h_{t-1} and the current input at time t, i.e. x_t , concatenates them and applies a linear transformation followed by a sigmoid:

$$f^{(t)} = \sigma(W_f[h^{(t-1)}, x^t] + b_f)$$
- Because of the sigmoid, the output of the Forget Gate is between 0 and 1
- If $f^{(t)} = 0$, then the previous internal state is completely forgotten
- If $f^{(t)} = 1$, then the previous internal state will be passed through unaltered



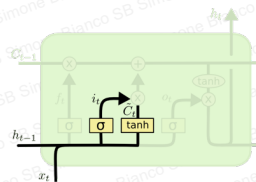
26

LSTM – Input Gate

- First, determine which entries in the cell state to update by computing 0-1 sigmoid output:

$$i^{(t)} = \sigma(W_i[h^{(t-1)}, x^t] + b_i)$$
- Then determine what amount to add/subtract from these entries by computing a tanh output (valued -1 to 1) function of the input and hidden state:

$$\tilde{C}^{(t)} = \tanh(W_C[h^{(t-1)}, x^t] + b_C)$$

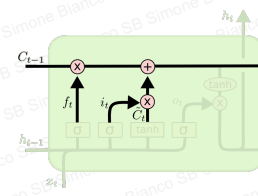


27

LSTM – Update the Cell State

- The previous state is multiplied by the forget gate and then added to the fraction of the new candidate allowed by the input gate:

$$C^{(t)} = f^{(t)}C^{(t-1)} + i^{(t)}\tilde{C}^{(t)}$$
- In other words, cell state is updated by using component-wise vector multiplication to «forget» and vector addition to «include» new information



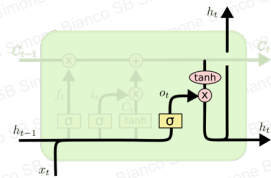
28

LSTM – Output Gate

- Hidden state is updated based on a «filtered» version of the cell state, scaled to -1 to 1 using tanh.
- Output gate computes a sigmoid function of the input and previous hidden state to determine which elements of the cell state to «output»:

$$o^{(t)} = \sigma(W_o[h^{(t-1)}, x^t] + b_o)$$

$$h^{(t)} = o^{(t)} \tanh C^{(t)}$$



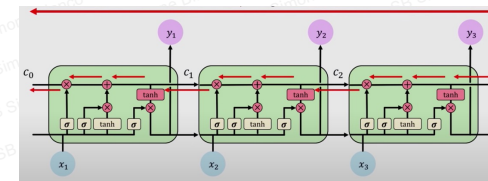
29

LSTM – Key concepts

- Use gates to control the flow of information:

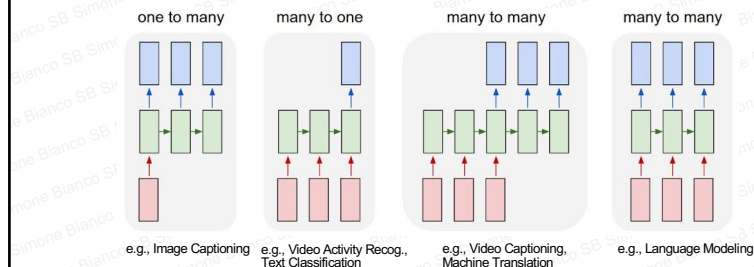
1. Forget gate gets rid of irrelevant information
2. Store relevant information from current input
3. Selectively update cell state
4. Output gate returns a filtered version of the cell state

- Backpropagation through time with uninterrupted gradient flow



30

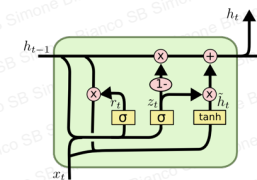
LSTM – Summary of application architectures



31

GRU – Gated Recurrent Unit

- Alternative RNN to LSTM that uses fewer gates
- Combines forget and input gates into «update» gate
- Eliminates cell state vector
- GRU has significant less parameters than LSTM and trains faster
- But...each has problems on which it performs better



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

32