UCL

Advanced Machine Learning in Finance

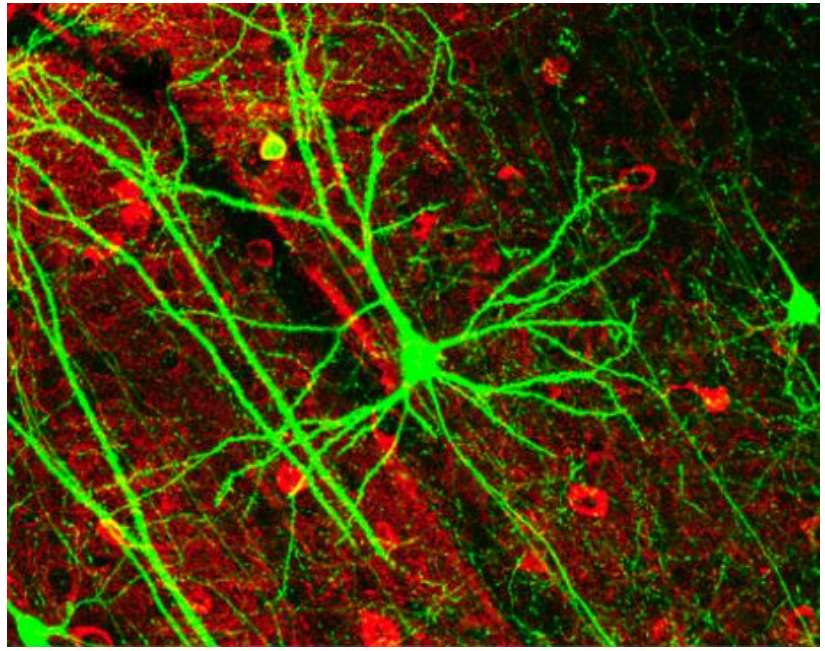
Paolo Barucca
Tomaso Aste

Lecture 4

- Autoregressive Models
- Recurrent Neural Networks
- Recurrent Units
- Case study: Time Series Prediction

The Great Revolution

The great revolution brought about by neural networks in the field of artificial intelligence started from **models inspired by the brain** and, in particular, by neurons considered to be its fundamental computational units.



Real neurons are connected by intricate synaptic connections regulated by chemical reactions.
In comparison, the **artificial neurons** in neural networks have simpler and more well-defined rules.

Examples of Sequence Modelling

Examples of Sequence data	Input Data	Output
Speech Recognition		*This is RNN*
Machine Translation	Hello, I am Pankaj.	Hallo, ich bin Pankaj. हैलो, मैं पंकज हूं।
Language Modeling	Recurrent neural ? based ? model	network language
Named Entity Recognition	Pankaj lives in Munich	Pankaj lives in Munich *person* *location*
Sentiment Classification	There is nothing to like in this movie.	★☆☆☆☆
Video Activity Analysis		Punching

Types of Sequence Models

Short-memory models: Autoregressive methods model linear dependencies between historical time series and predicted values

Feedforward neural networks: model nonlinear relationships between input and output

Recurrent neural networks: contains a dynamic hidden state to store long-term information

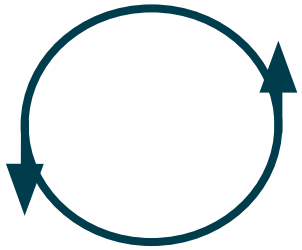
Recurrence

Artificial neurons are computational units that receive numerical values from other neurons or an external data source, and compute a numerical output which they in turn send to other artificial neurons.

Artificial neural networks have **well-defined layers** to be able to control in detail the structure of the connections and their basic properties, like in the RBM.

Recurrence

One of these fundamental properties is **recurrence**, that is the presence of closed neuronal circuits (loops).



In a recurring network, a neuron can end up influencing itself thanks to a loop of connections, its value influences neurons which directly or indirectly return their output values to it.

Recurrence

Artificial neural networks can therefore be classified into two macro-classes:

- **feedforward neural networks** that have no loops and the flow of information is unidirectional

- **recurrent neural networks** (RNN), which instead present cycles of neuronal connections, in which a neuron can receive feedback from its output

In practice, RNNs tend to have a specific recurrent structure, rather than being a generic network of neurons with cycles.

Recurrence

In feedforward neural networks learning is simplified by the absence of cycles.

The **absence of cycles** makes it easier to compute the **direct effect** of each parameter of the neural network, with algorithms like **backpropagation**.

Conversely, in recurrent neural networks, the effect of the choice of parameters of a neuron affects the values that will return to the input of that neuron in subsequent iterations, making the optimization of the parameters more complicated.

Properties of Recurrent Neural Networks

RNNs possess a dynamic hidden state that can store long-term information

Non-linear dynamics: can update their hidden state in complicated ways

Temporal and accumulative: can build semantics e.g. word-by-word in sequence over time

Goals of Recurrent Neural Networks

To model long term dependencies in sequences such as time series data

Connect previous information to the present task

Model sequence of events with loops, allowing information to persist

Recurrent Neural Networks

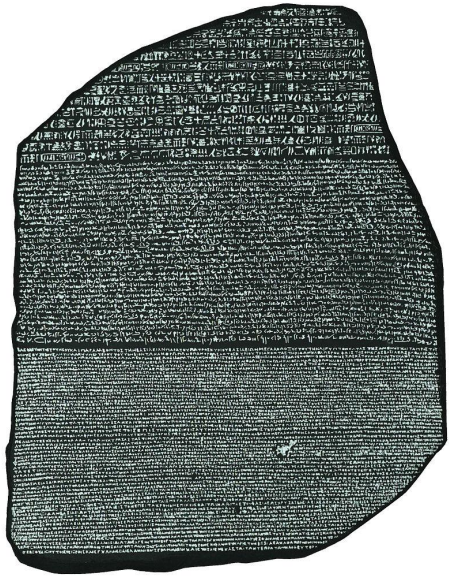
Recurrent Neural Networks (RNNs) introduce recurrence to exploit the sequential nature of their input.

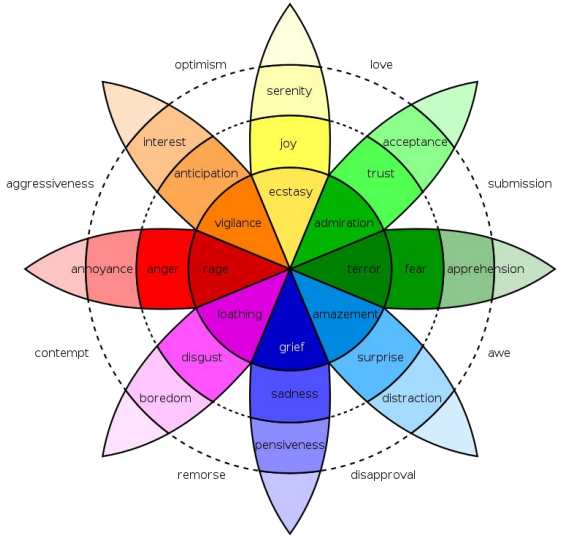
Sequential inputs could be text, speech, time series, and any other dataset where the occurrence of an element in a sequence depends on the elements that precede it, in one equation:

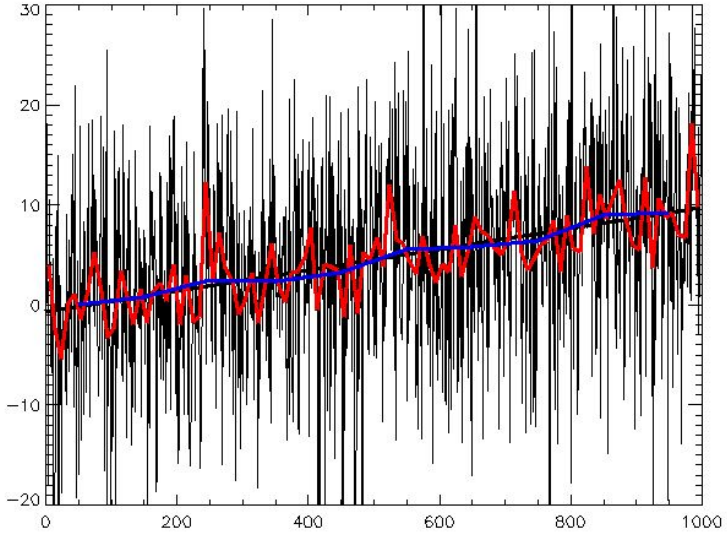
$$h_t = f(X_t, h_{t-1})$$

Recurrent Neural Networks

RNNs are used to solve **traditionally human tasks** such as speech recognition, language modeling, machine translation, sentiment analysis, image captioning, or time series prediction.



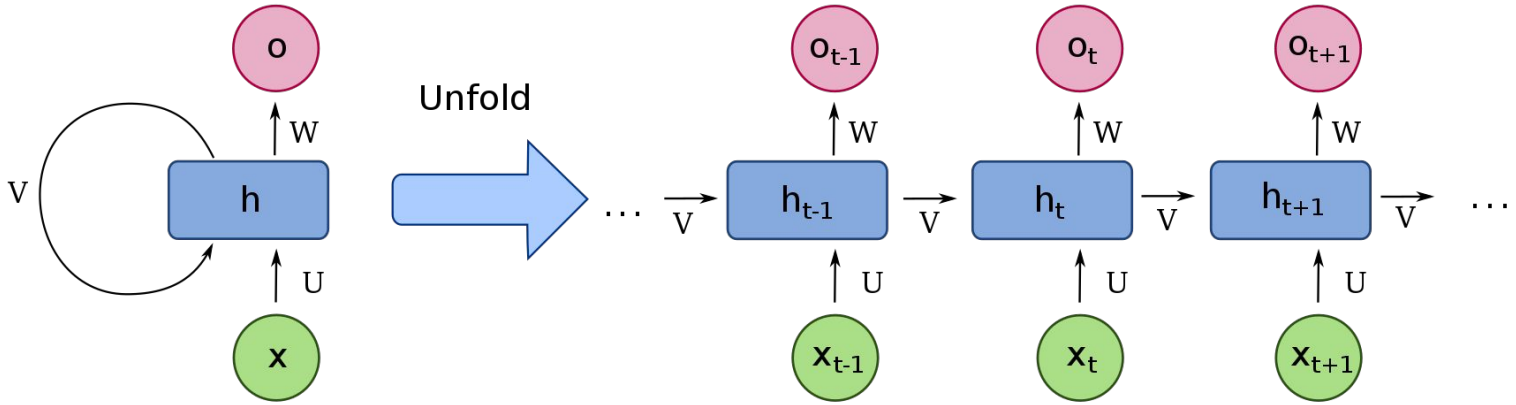




Recurrent Neural Networks

The basic structure of the algorithms of recurrent neural networks can be represented with the following graph where:

- x, are the inputs: text, speech, numerical series, image, video
- h, are the memory variables that are learned
- o, are any outputs used to compute an objective function
- V, W, U are numeric matrices that transform the data entering and leaving the unit



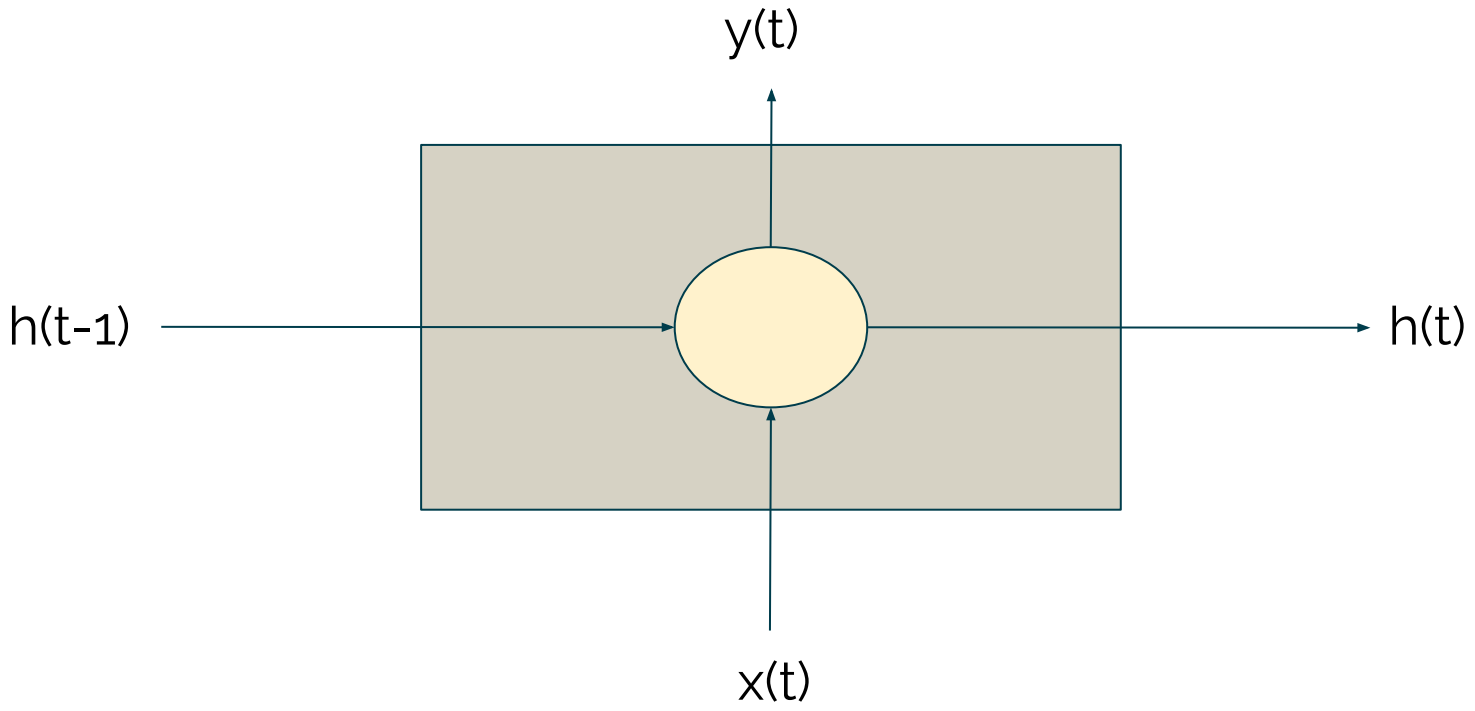
Recurrent Neural Networks

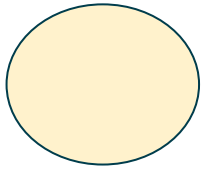
What exactly happens in the single unit of a RNN?
The signal $\boldsymbol{x}$ is processed together with the variables $\boldsymbol{h}$ to calculate the update of the variables $\boldsymbol{h}$ and output $\boldsymbol{o}$.

There are several units that differ in the mathematical functions used and the number of parameters and intermediate variables introduced, in particular we consider:

- Simple RNN "Vanilla" unit (1986)
- Gated Recurrent unit (2014)
- LSTM (1997)

Vanilla Unit





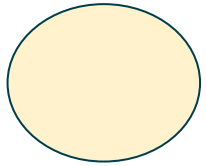
$$h(t) = \tanh\left(W_{hx}x(t) + W_{hh}h(t-1) + b_h\right)$$

Vanilla Unit

We can simply calculate the **total number of parameters** in a vanilla unit:

- If input x is of size D and the hidden variables are L
- Then the matrix W_{hx} has size L x D
- And the matrix W_{hh} has size L x L
- And biases have size L

So that in total we get L x D + L x L + L parameters.



$$h(t) = \tanh\left(W_{hx}x(t) + W_{hh}h(t-1) + b_h\right)$$

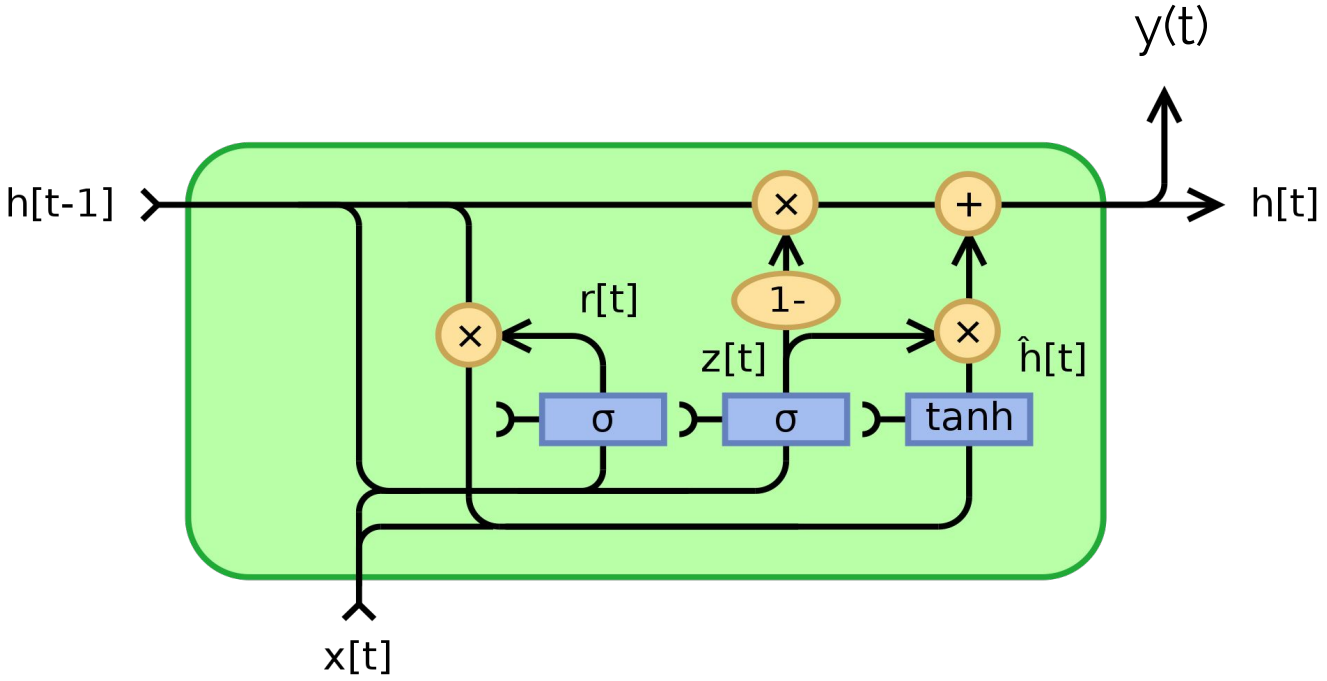
LSTM vs GRU

The Long Short-Term Memory (LSTM) unit introduced in 1987 features an intricate structure that serves to maintain both short and long-term memory of the sequence.

The simpler GRU was introduced in 2014 precisely to maintain the properties of the LSTM unit but with a simpler structure and is proving very competitive in all the applications where LSTM was state-of-the-art.

Gated Recurrent Unit

In GRUs intermediate variables are introduced:
the update variable z(t), and the reset variable r(t)

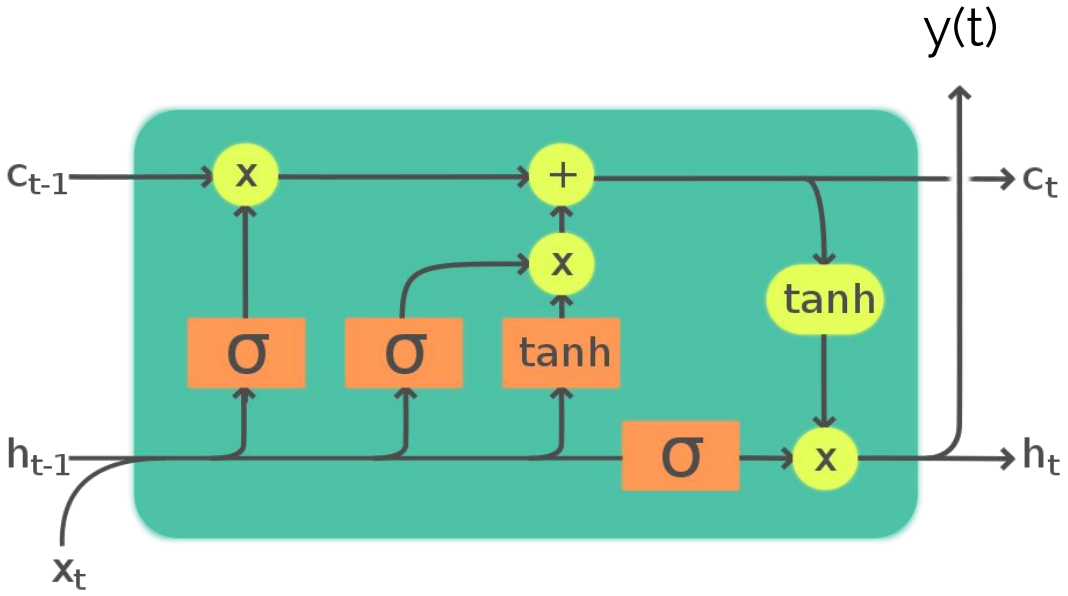


$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$
$$\hat{h}_t = \phi_h(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h)$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t$$

σ_g and ϕ_h are **activation functions**

Long-short term memory unit

LSTM units have update, forget, cell state c(t) variables



$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

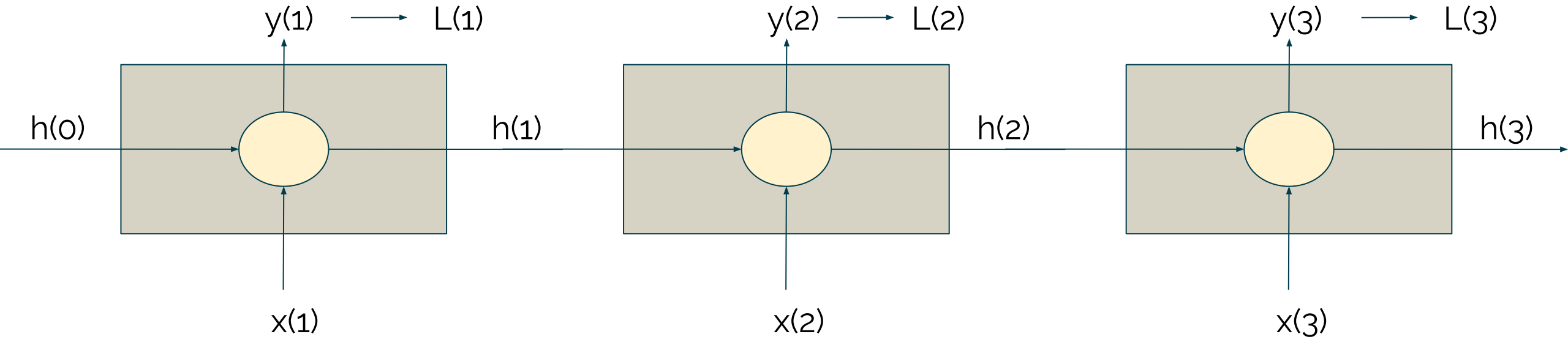
The output

All RNNs must learn their parameters by minimizing an objective function based on the output of the RNN, dependent on the specific task and a set of learning data. For example

- for the prediction of a time series of numerical data, the output consists of **the predicted value** for the next step in the series, which is naturally contained in the series itself,
- in other cases, the output can be: **a text for creating image captions**, a **sentiment class** for sentiment analysis, or **a text in another language** for machine translation.

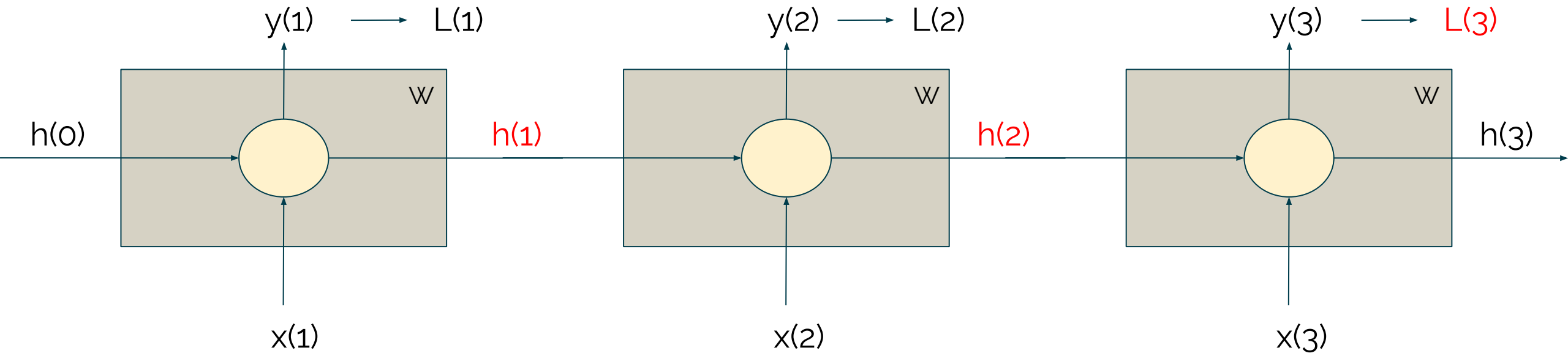
The loss function

In general, we can associate a loss L(t) for the RNN to each step of a sequence. Summing all the losses we can define the total loss L for the RNN.



Training an RNN

During training we want to minimize the total loss by changing the parameters in the RNN unit, keeping in mind that losses at each step are interdependent.



BPTT

This dependence over time leads to the backpropagation through time (BPTT) algorithm, that is the propagation that goes backwards in time.

To calculate the gradient of the total loss function, which is necessary for learning, all the contributions must be reconstructed by going back in time, or back in the input sequence.

Mathematically this implies repeated multiplication. In the previous example it would be:

$$\frac{\partial L(3)}{\partial W} = \sum_{t=1}^{2} \frac{\partial L(3)}{\partial y(3)} \frac{\partial y(3)}{\partial h(2)} \frac{\partial h(2)}{\partial h(t)} \frac{\partial h(t)}{\partial W}$$

Vanishing and exploding gradients

For a long sequence, dependence on the first steps implies a long series of multiplications, which have two possible results:

- if each term is greater than one, multiplication leads to an **exploding gradient**
- if each term is less than one, the multiplication leads to a **vanishing gradient**

This problem hinders learning and forces you to choose between short (**vanishing gradient**) and long memory (**exploding gradient**).

LSTM units and GRUs were introduced precisely to solve this fundamental problem and integrate both long and short memory into RNNs.

Vanishing and exploding gradients continued

There are a few other ways to mitigate vanishing/exploding gradients in addition to using an LSTM/GRU such as:

Gradient clipping

Applying L2 regulariser on neural network weights (gradient vanishing)

Add skip connections (such as in ResNet)

Applying layer normalisation (layer-norm)

Bidirectional RNN

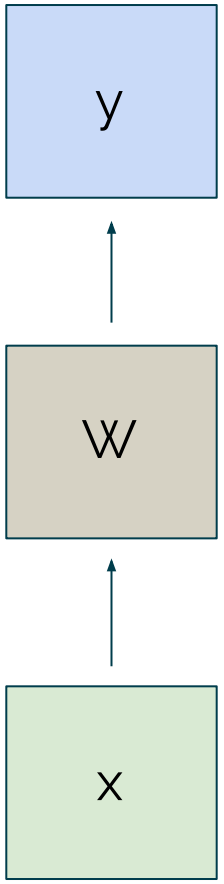
The output of the RNN depends on the output of all previous time steps, but in a sequence it is often the case that the output **also depends on the succeeding output values**.

A bidirectional RNN consists of two RNNs stacked on top of each other, one processing left-to-right input and the other processing right-to-left input.

In natural language processing (NLP), in fact, the predictive factors of the word or phrase that you are trying to predict can depend on the entire sentence that encloses it, not only on the words that precede it. For example in the sentence:

"The child ... (Y).... the apple"

RNN I/O topologies



To understand the structures of recurrent neural networks we start from a network that is NOT recurrent.

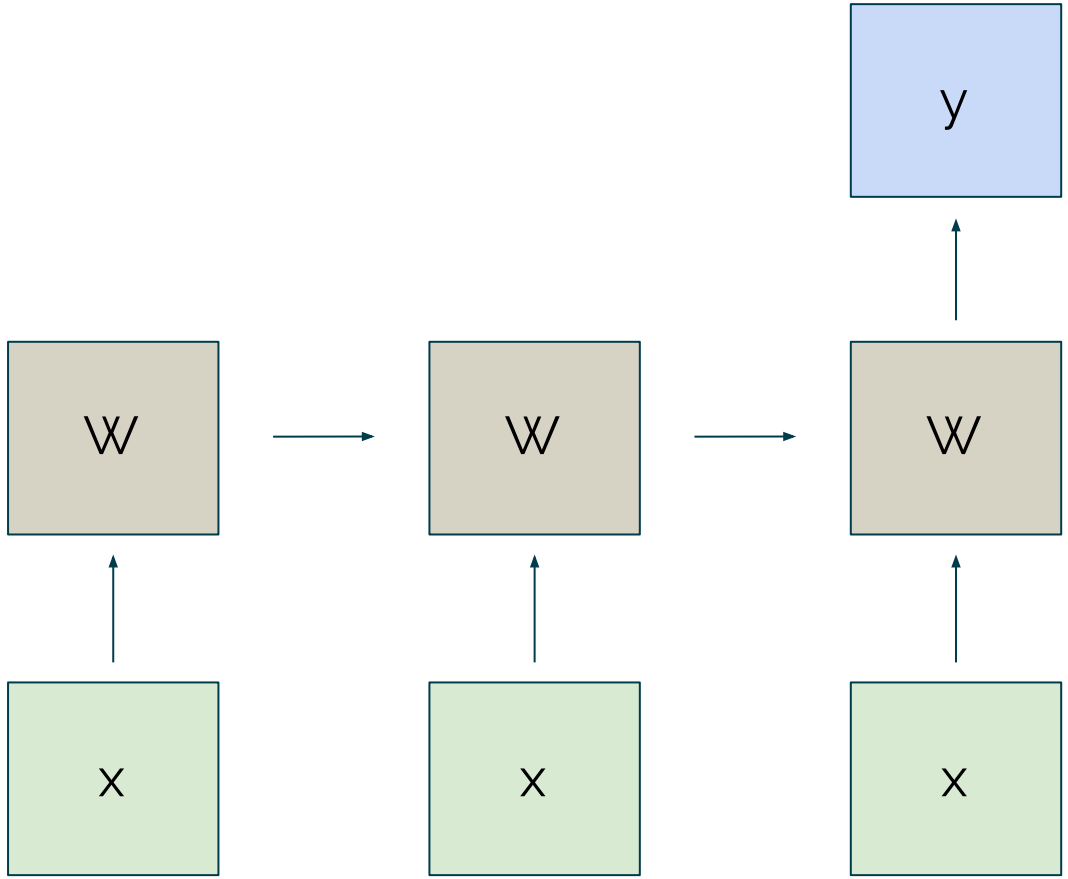
This image represents a feedforward network, we have an input and an output **but no sequential recurrence** relationship, neither for the input nor for the output.

We can have many samples for learning but the output for a given input sample is not received as input in subsequent iterations.

There are no cycles and there is no recurrence.

An example of an application is the classification of images.

RNN I/O topologies

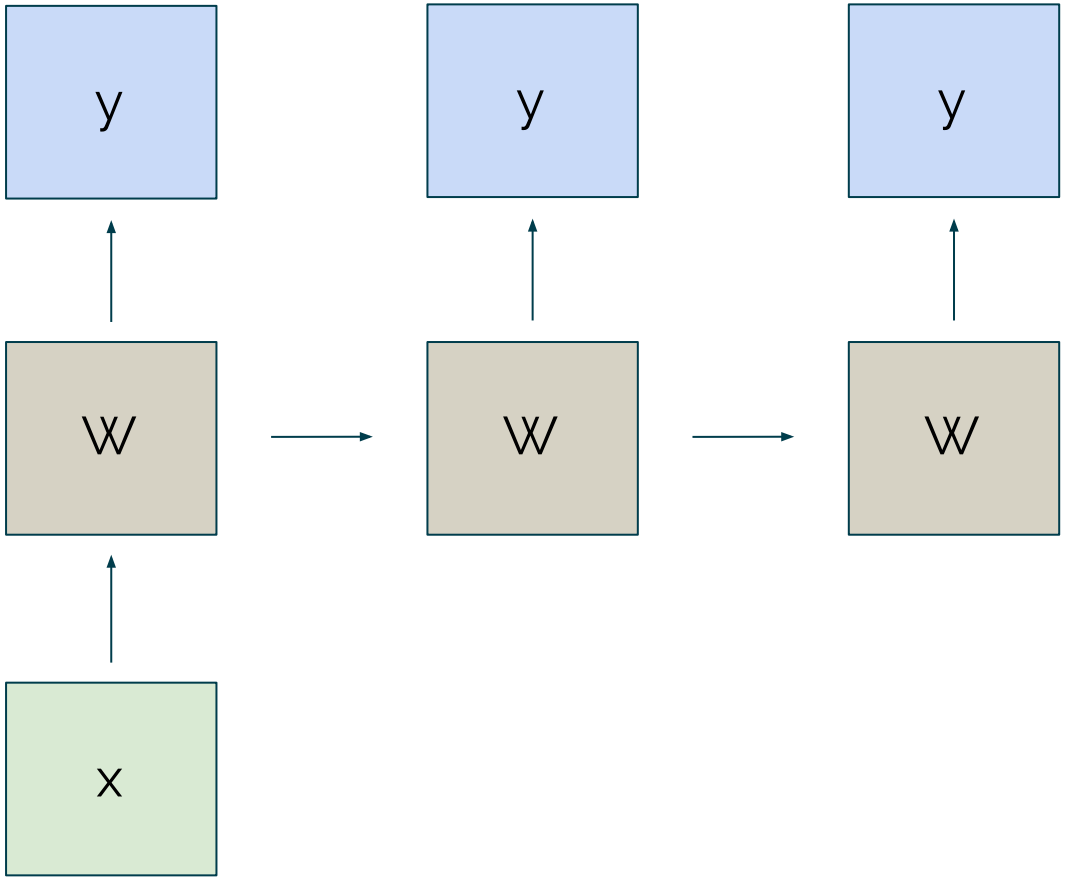


Many-to-one topology

Applications:

- Text and audio classification
- Sentiment analysis
- Time series classification

RNN I/O topologies

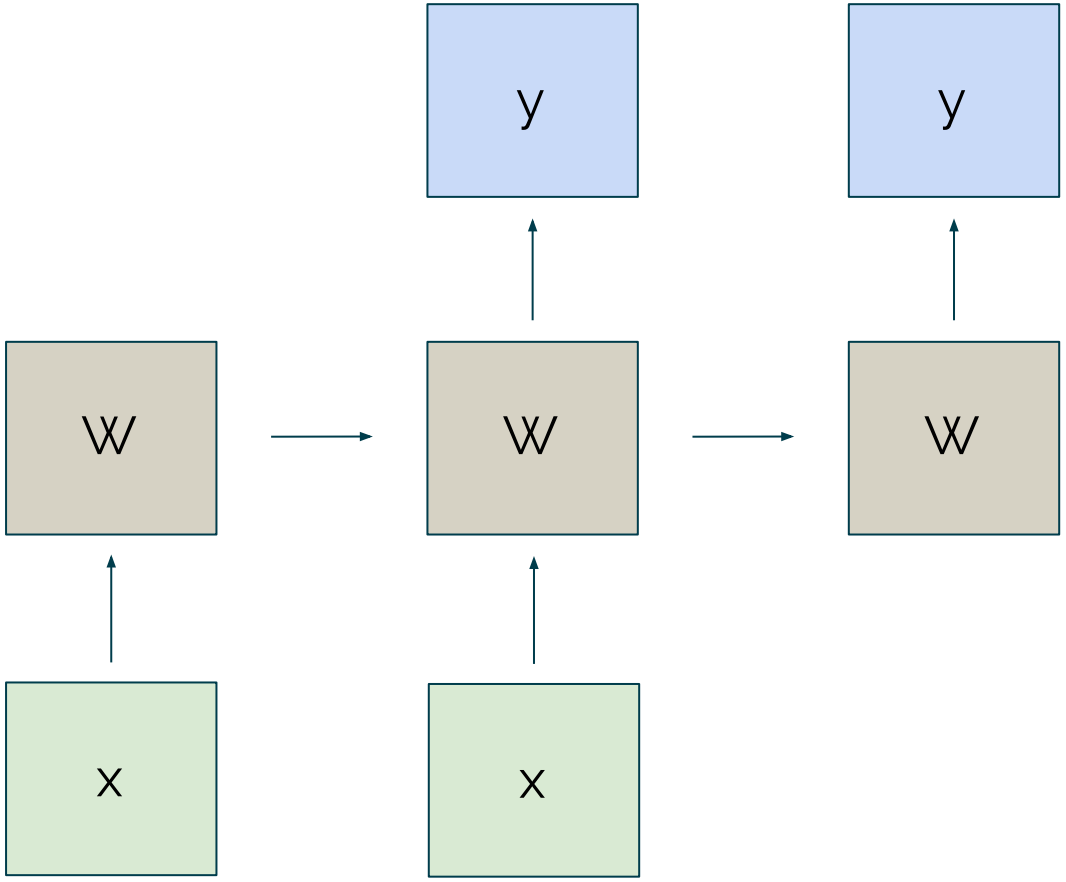


One-to-many topology

Applications:

- Image captioning
- Time series generation

RNN I/O topologies

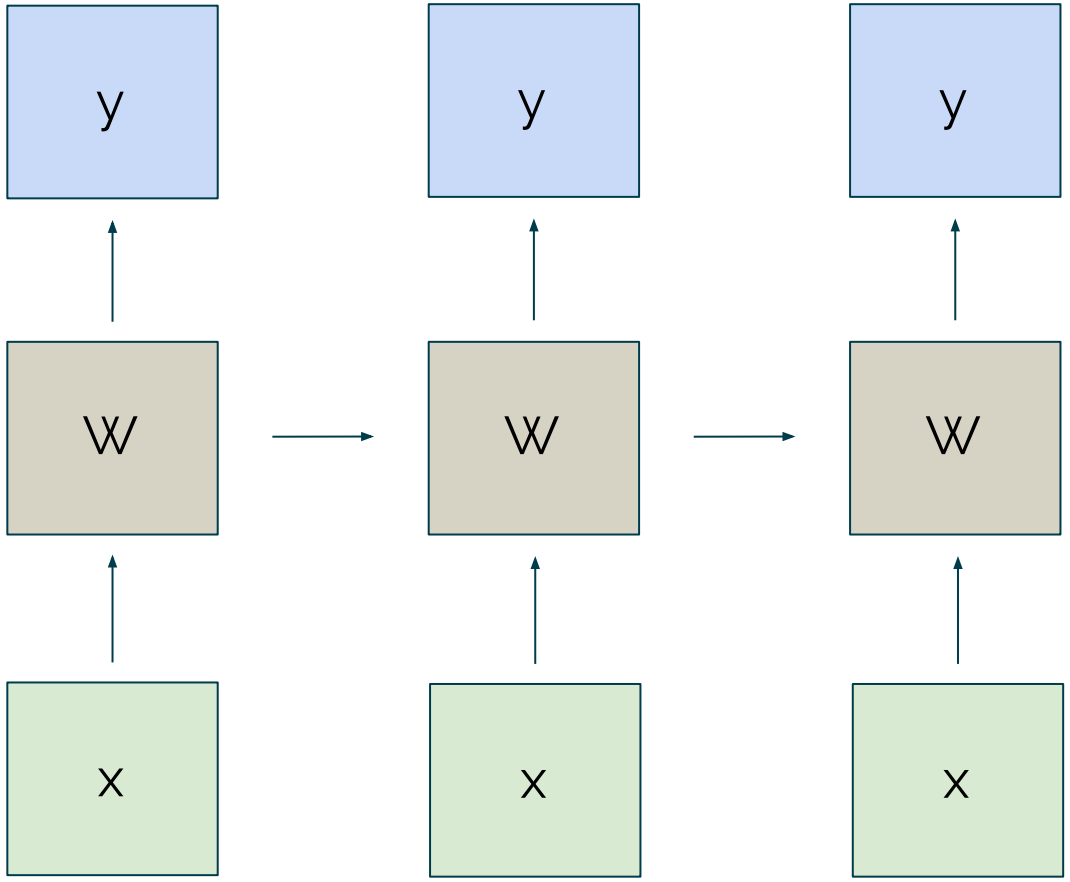


Many-to-many topology, also referred to as seq2seq

Applications:

- Machine translation

RNN I/O topologies



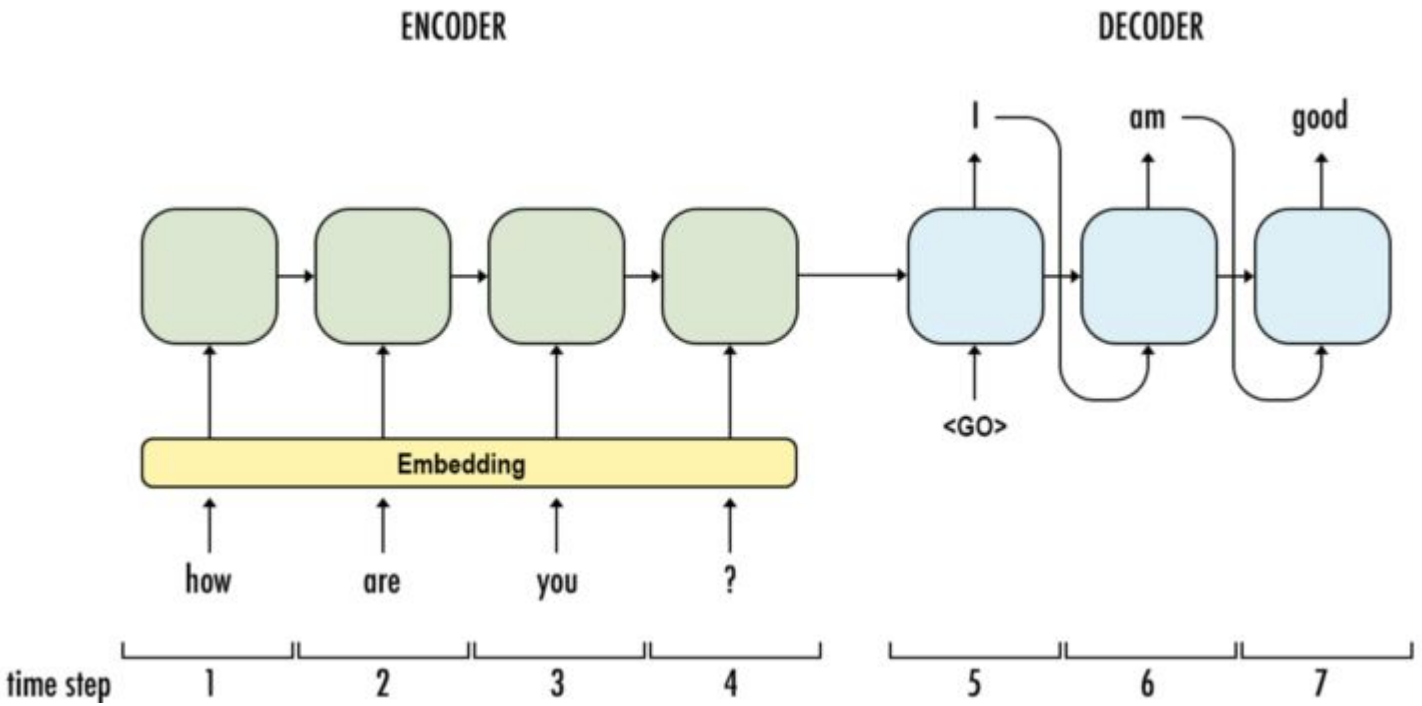
Many-to-many topology (Type II)

Applications

- Time series prediction
- Part-of-speech, POS

Here output can be generated before the full processing of the entire sequence

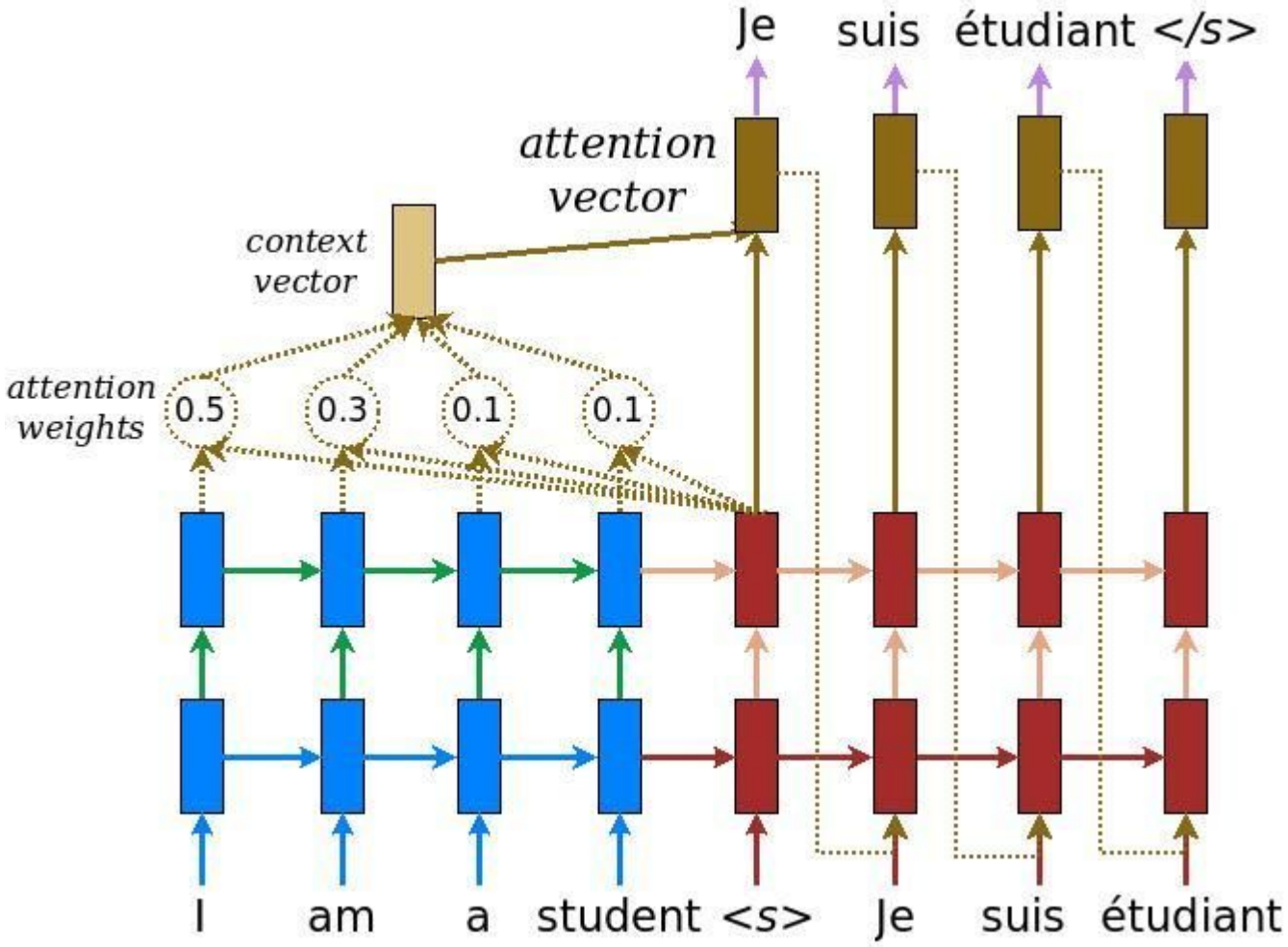
Attention Mechanism



Encoder-Decoder setups with RNNs allow us to map an input sequence to an output sequence

https://medium.com/syncedreview/a-brief-overview-of-attention-mechanism-13c578ba9129

Attention Mechanism



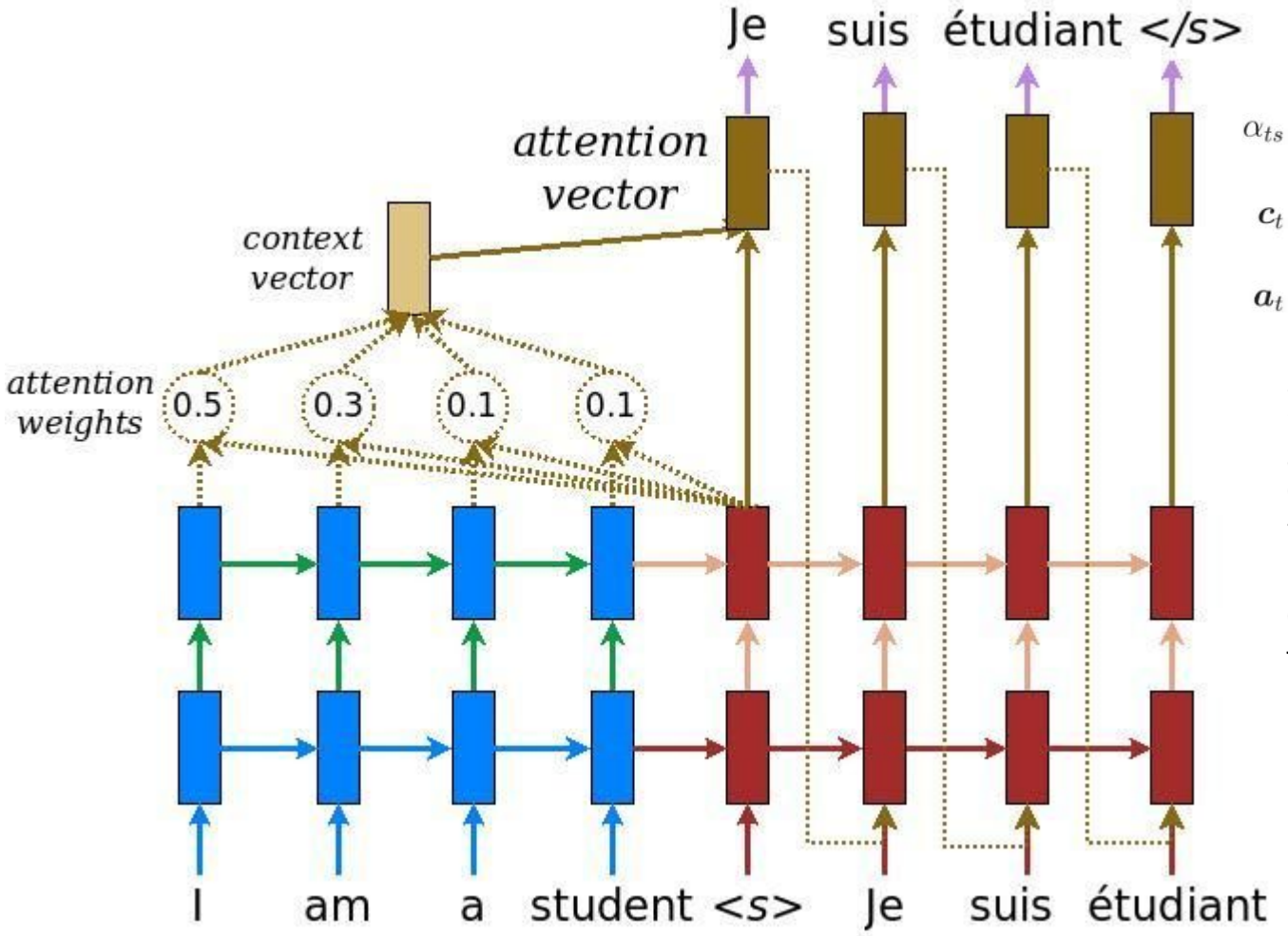
Attention improves the performance of Encoder-Decoder RNN on tasks such as machine translation and multistep time series forecasting

Allows focus on local or global features

The output of the encoder is the context vector which serves as the input of the decoder network

https://medium.com/syncedreview/a-brief-overview-of-attention-mechanism-13c578ba9129

Attention Mechanism



$$\alpha_{ts} = \frac{\exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)\right)}{\sum_{s'=1}^{S} \exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_{s'})\right)} \quad \text{[Attention weights]} \quad (1)$$

$$\boldsymbol{c}_t = \sum_s \alpha_{ts} \bar{\boldsymbol{h}}_s \quad \text{[Context vector]} \quad (2)$$

$$\boldsymbol{a}_t = f(\boldsymbol{c}_t, \boldsymbol{h}_t) = \tanh(\boldsymbol{W}_c[\boldsymbol{c}_t; \boldsymbol{h}_t]) \quad \text{[Attention vector]} \quad (3)$$

For token at time t we compute the attention weights by generating scores between current hidden state and all encoder hidden states. We then apply Softmax to normalise the scores

Given the attention weights we can compute the context vector and the attention vector

https://medium.com/syncedreview/a-brief-overview-of-attention-mechanism-13c578ba9129