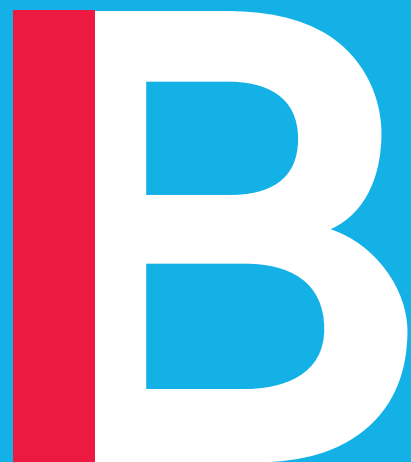


# Machine Learning in Finance

## Lecture 6 Introduction to Sequence Models



Arnaud de Servigny & Hachem Madmoun

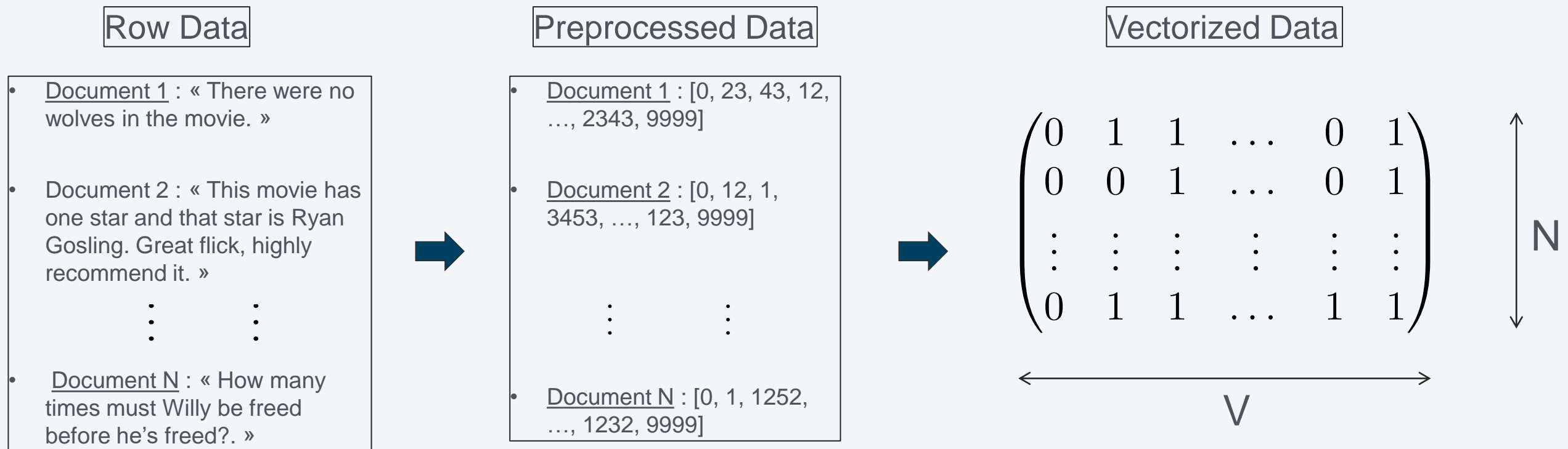
# Outline:

- Introducing the concept of Memory
- The Embedding Layer
- Recurrent Neural Networks
- Programming Session

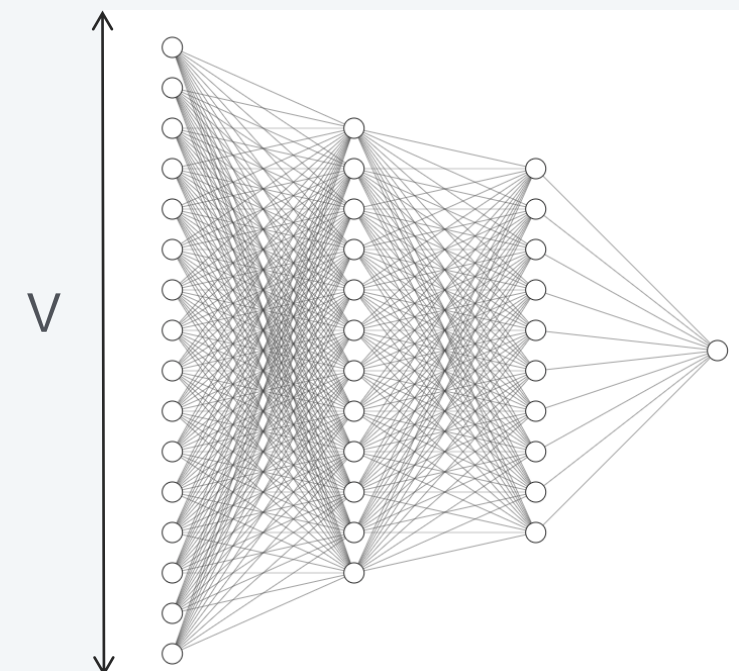
# Part 1 : Introducing the concept of Memory

# Review of the Sentiment Analysis Pipeline

- The Preprocessing steps:



- The Tensor data is of shape (N, V) after the preprocessing steps.
- Then, we feed the tensor data into a traditional Machine Learning Algorithm or a Neural Networks composed of a stack of dense layers.

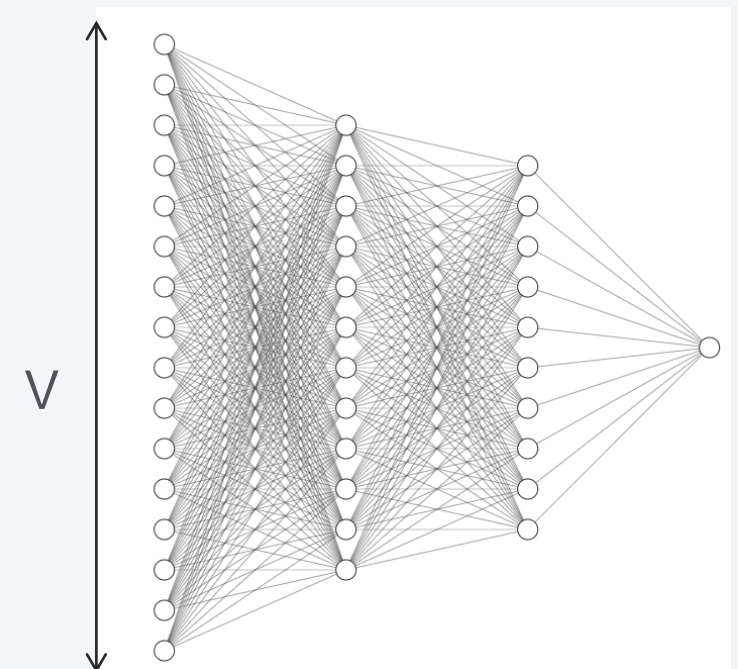
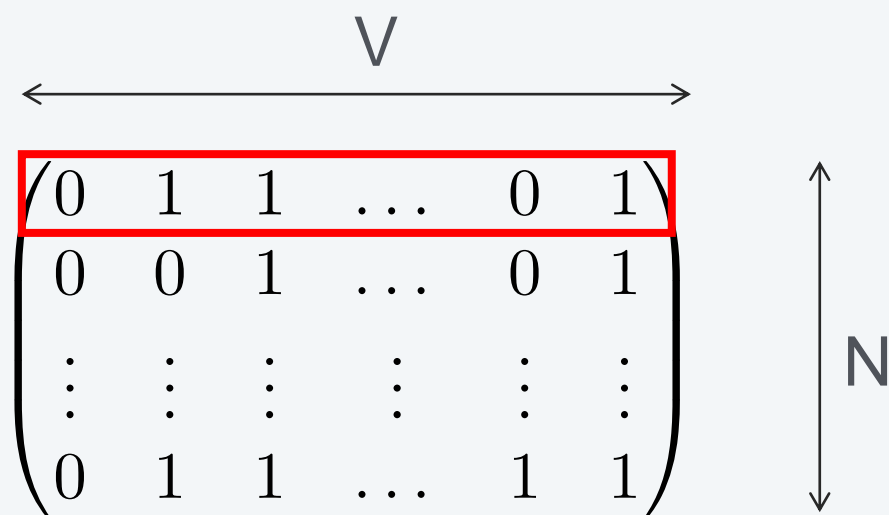


# Limitations of the approach

1. The first limitation comes from the way we encode the sentence into a V-dimensional vector:

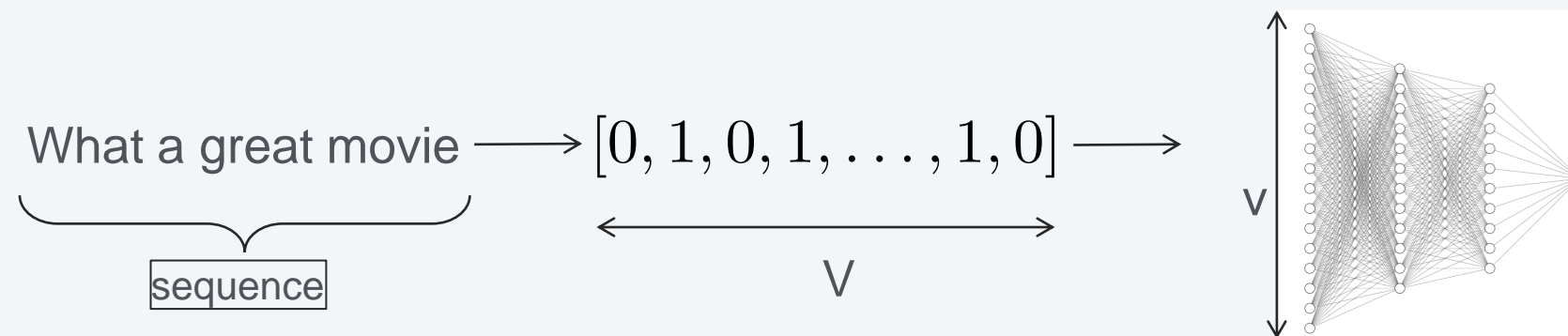
- The encoding is performed regardless of the order in which the words come.
- For instance, these two sentences will be encoded into the same vector:
  - « Never quit. Do your best. »
  - « Never do your best. Quit. »

1. The other limitation comes from the model itself. We feed the entire sequence (encoded into a V-dimensional vector) all at once.

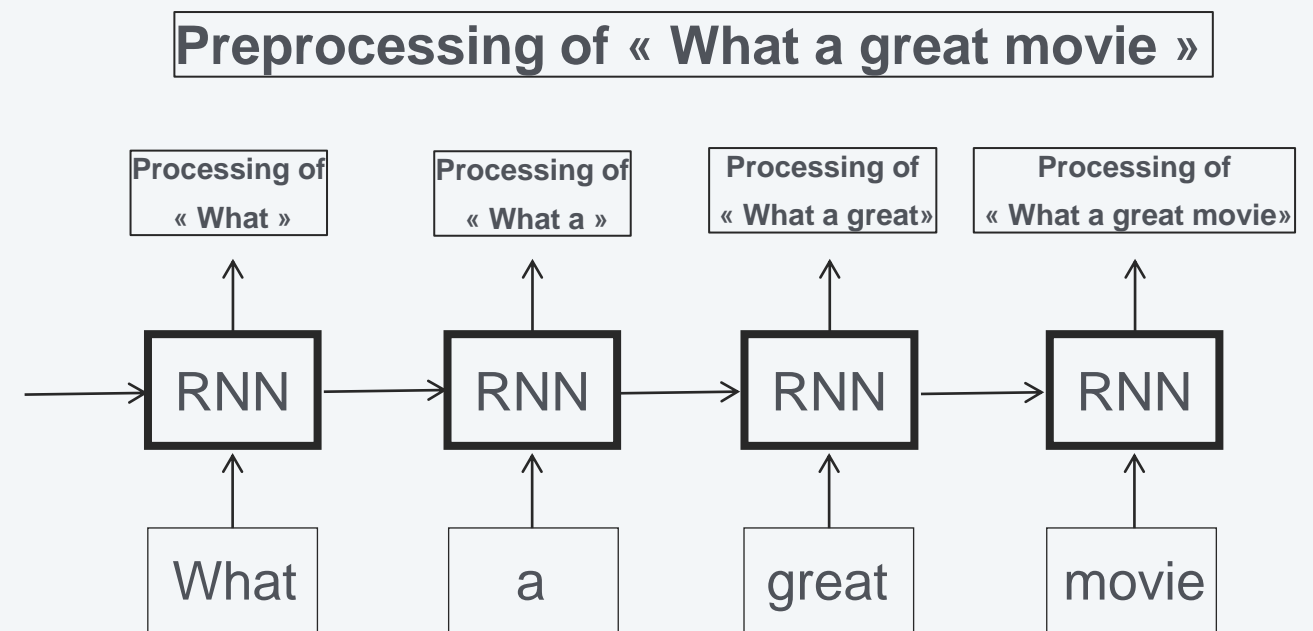
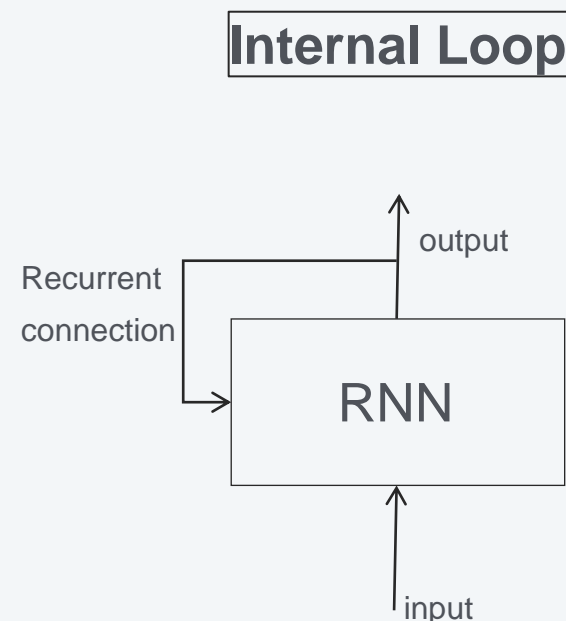


# The concept of Memory in Neural Networks

- When we are dealing with sequential data, the main limitation of the neural networks we have seen so far is that they look at each sequence as a single large vector. We call this kind of networks : **feedforward networks**.

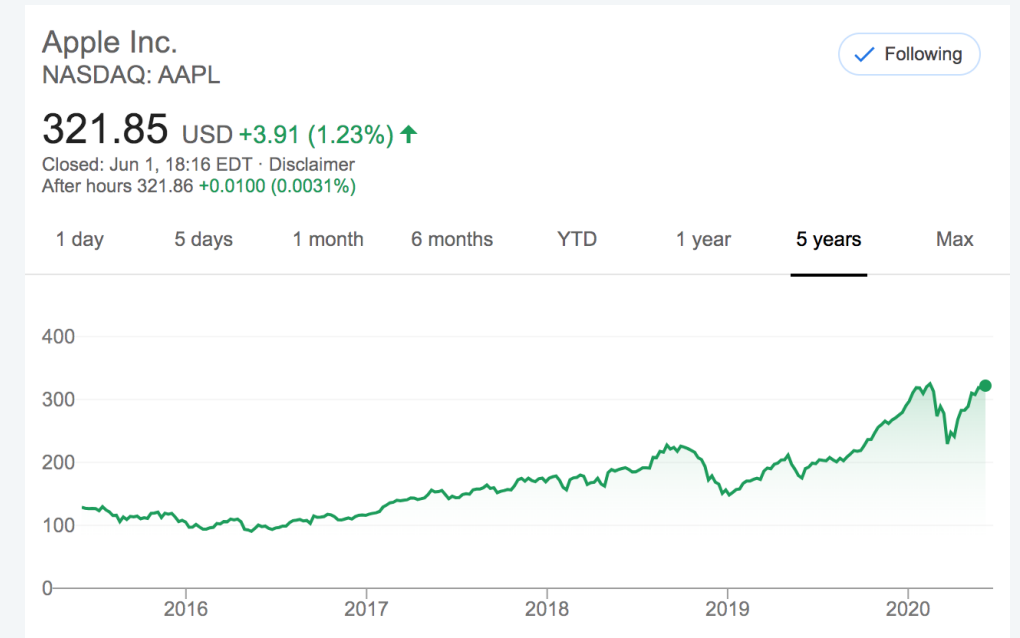


- Conversely, a **Recurrent Neural Network** processes sequences by looking at them element by element and keeping track of a **state** which contains the information processed so far.

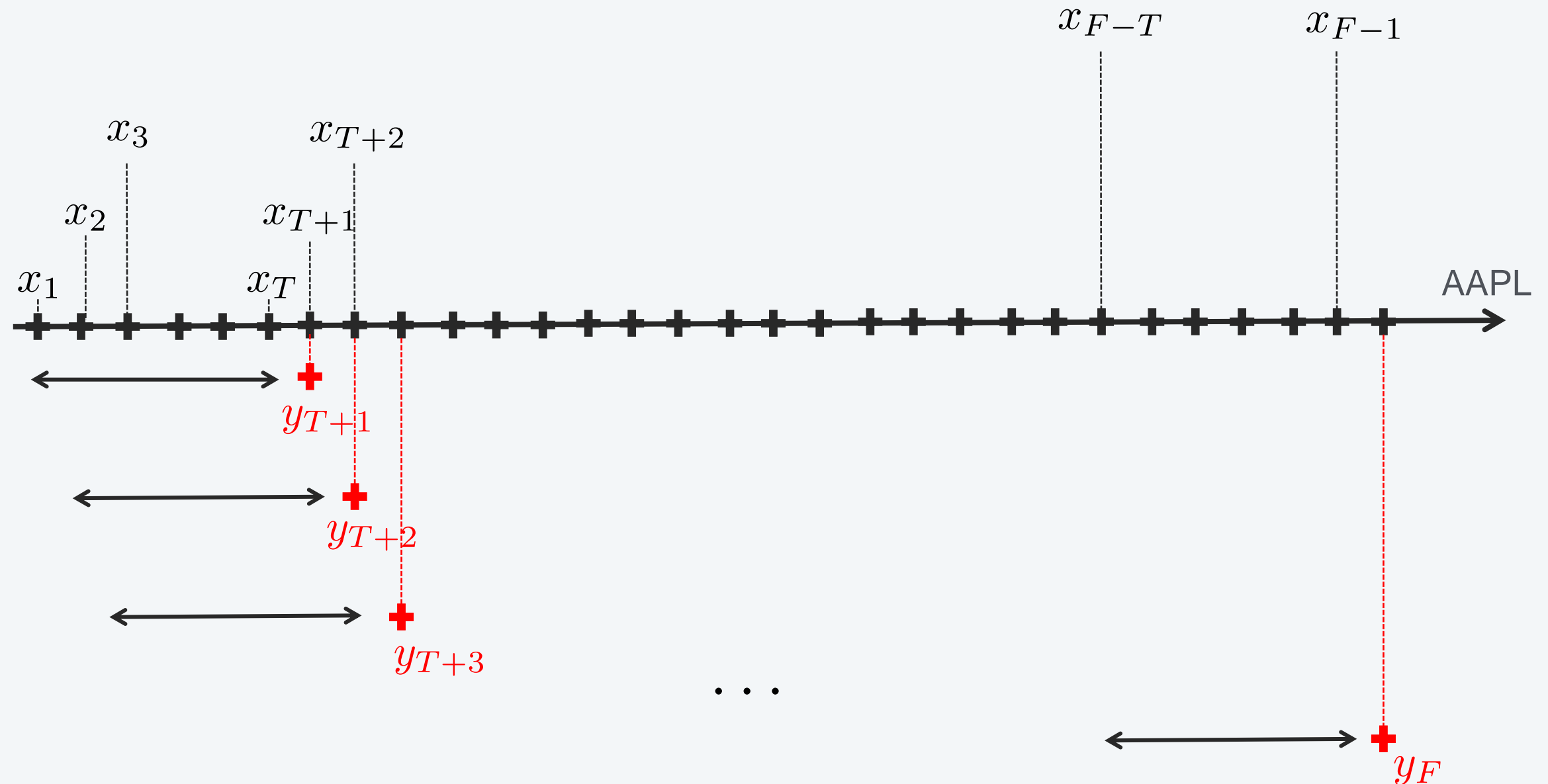


# A simple example – Part 1 –

- We want to predict whether the stock AAPL is going up or down for the next day, based on  $D = 4$  characteristics of the stock during the last  $T$  days.
- At each time step  $t$ , let  $x_t = [c_t^1, c_t^2, c_t^3, c_t^4]$  denote the four characteristics of the stock AAPL at time  $t$ .
- Let  $y_t$  denote the stock movement at time  $t$ :
  - $y_t = 1$  when the stock goes up between  $t-1$  and  $t$ .
  - $y_t = 0$  when the stock goes down between  $t-1$  and  $t$ .
- Let  $x_1, \dots, x_F$  be the whole sequence of characteristics from 2015 to 2020
- We want to predict the stock movement as follows:
  - From the sequence  $x_1, \dots, x_T$ , we want to predict  $y_{T+1}$
  - From the sequence  $x_2, \dots, x_{T+1}$ , we want to predict  $y_{T+2}$
  - $\vdots$
  - From the sequence  $x_{F-T}, \dots, x_{F-1}$ , we want to predict  $y_F$



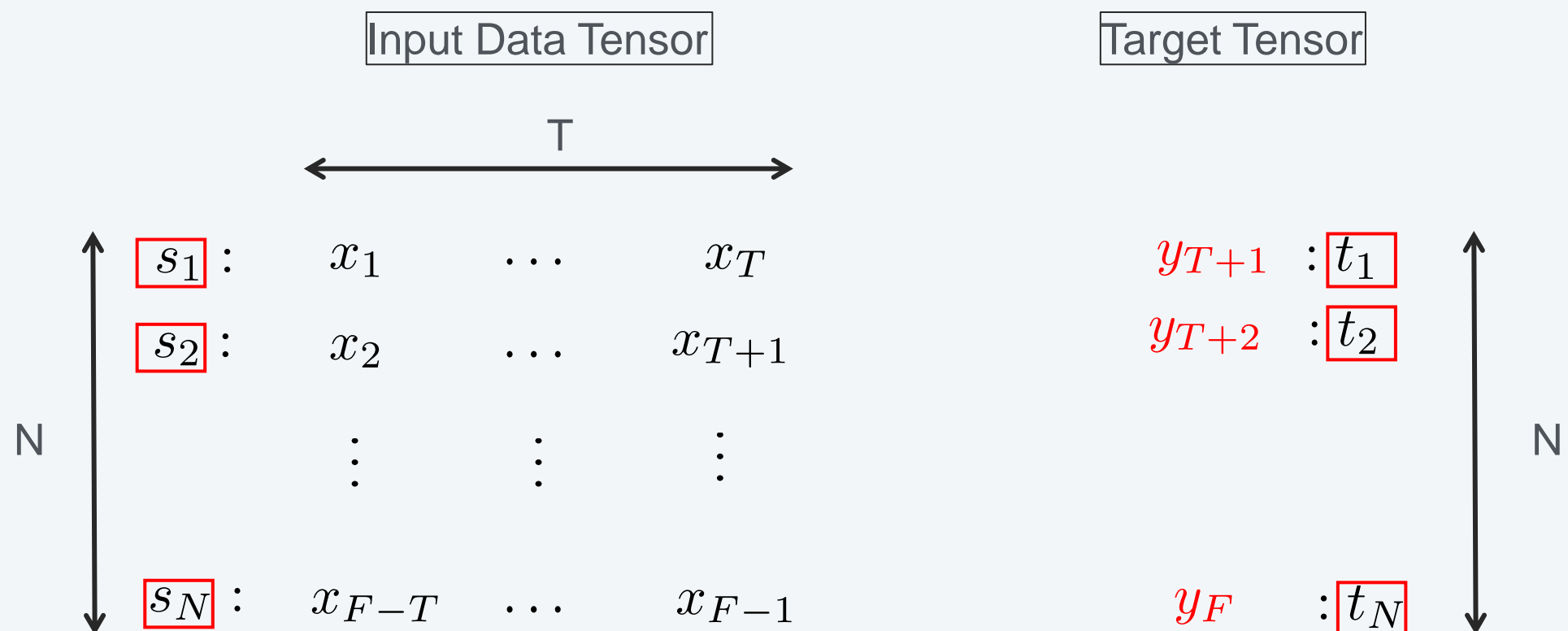
# A simple example – Part 2 –



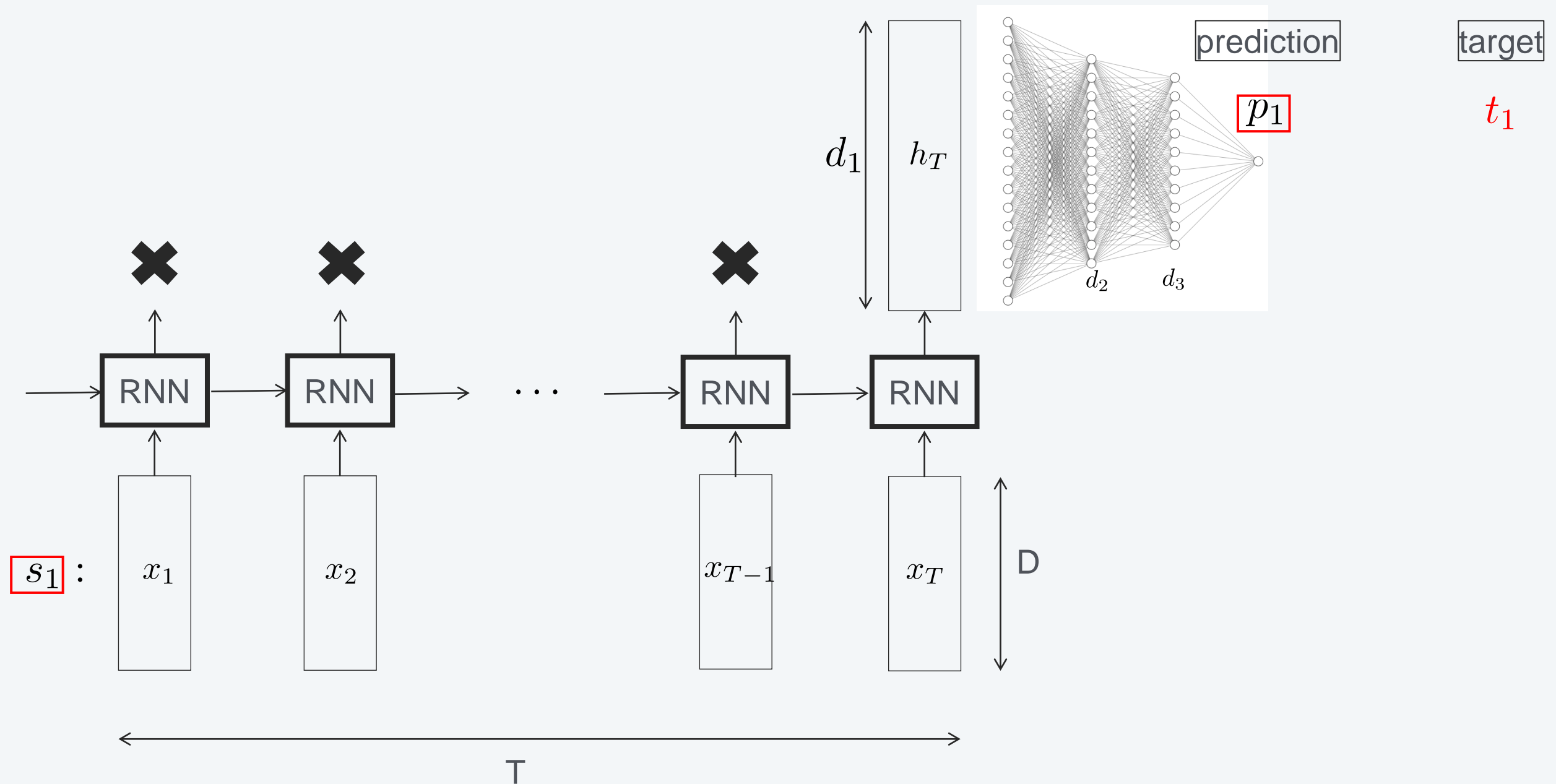


# A simple example – Part 3 –

- The input data tensor is a tensor of shape (N, T, D):
  - Each sample  $i$  among the N samples is a sequence  $s_i$  of length T with elements of dimension D.
  - We have  $N = F - T$  sequences.
- The target tensor data is of shape (N,):
  - Each sample  $i$  is associated with a target  $t_i$
  - Each target is a binary output : 1 for « up » and 0 for « down ».



# A simple example – Part 4 – The Forward Propagation:

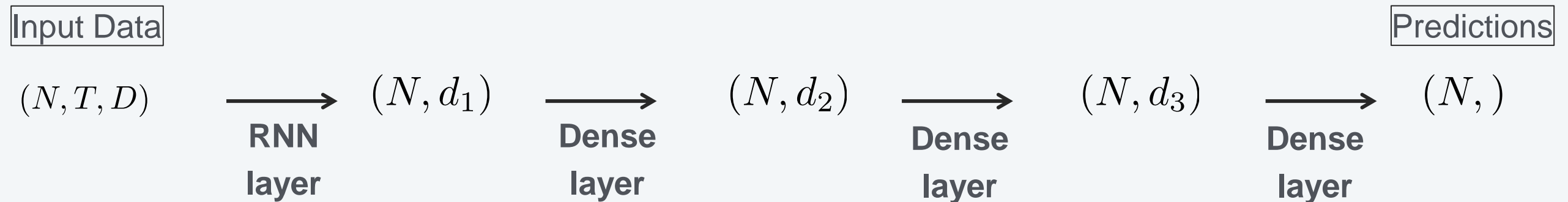


- The loss function associated with the N samples is the following binary cross entropy loss function:

$$J = -\frac{1}{N} \sum_{i=1}^N (t_i \log(p_i) + (1 - t_i) \log(1 - p_i))$$

# A simple example – Part 4 – The Forward Propagation:

- Let's focus on the evolution of the tensor shape at each layer transformation for the stock movement prediction problem:



- In the previous example, the input data is a 3D tensor representing  $N$  sequences of length  $T$ . Each sequence is composed of  $D$ -dimensional continuous vectors.
- In the Sentiment Analysis problem, the input data comes as sequences of integers.
- In order to use RNN models for the Sentiment Analysis problem, we will need an intermediate layer called the **Embedding layer**.

# Interactive Session

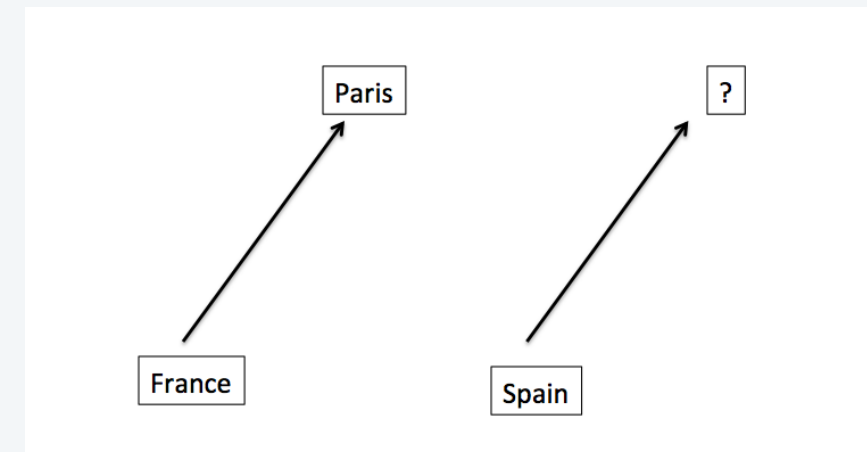
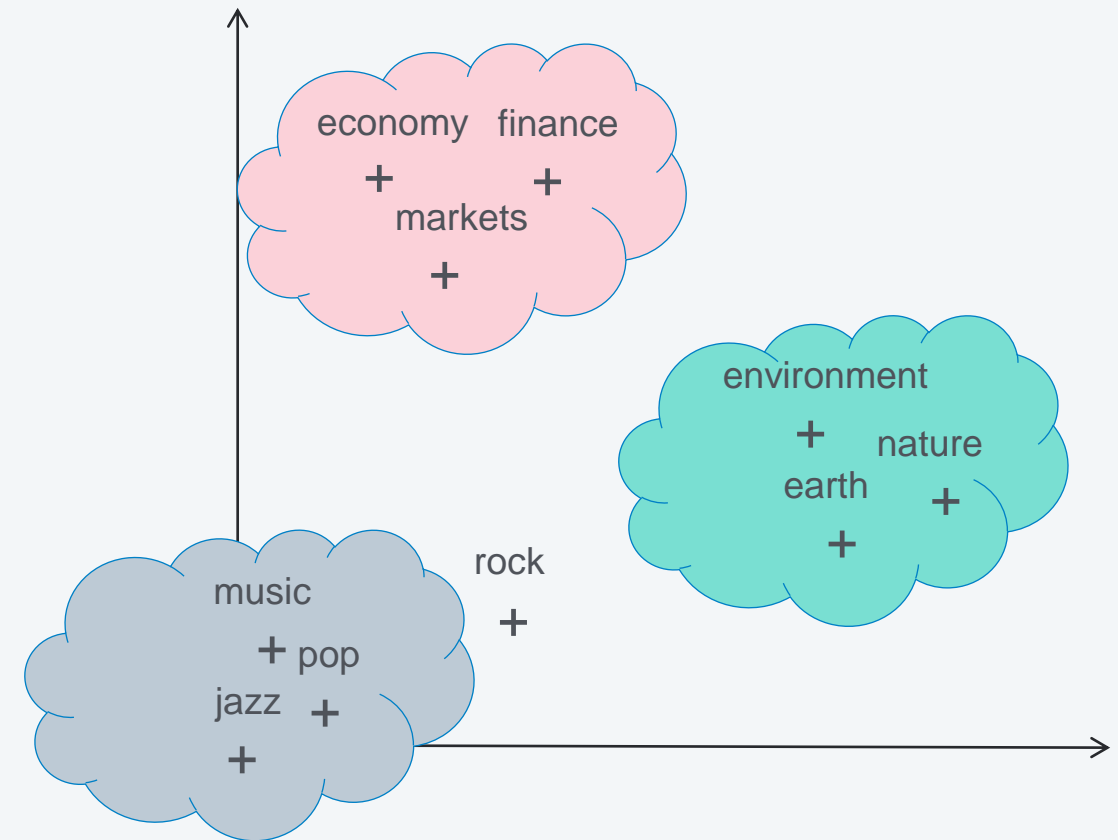


## Part 2 : The Embedding Layer

# The Embedding Space

- The embedding layer aims at mapping each word into a geometric space, called **the embedding space**.
- Each word is encoded into a D-dimensional vector (D is around 50 or 100).
- In the embedding space, words with similar meaning are encoded into similar word vectors.
- Since we use unsupervised learning to obtain these word vectors, they don't necessarily have to make sense to us. They only have to make sense geometrically.
- Common example of meaningful geometric transformations are « gender ».

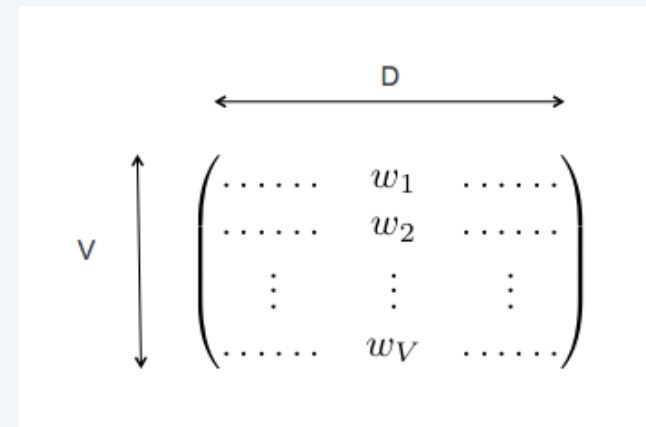
$$w_{\text{King}} - w_{\text{Man}} \approx w_{\text{Queen}} - w_{\text{Woman}}$$



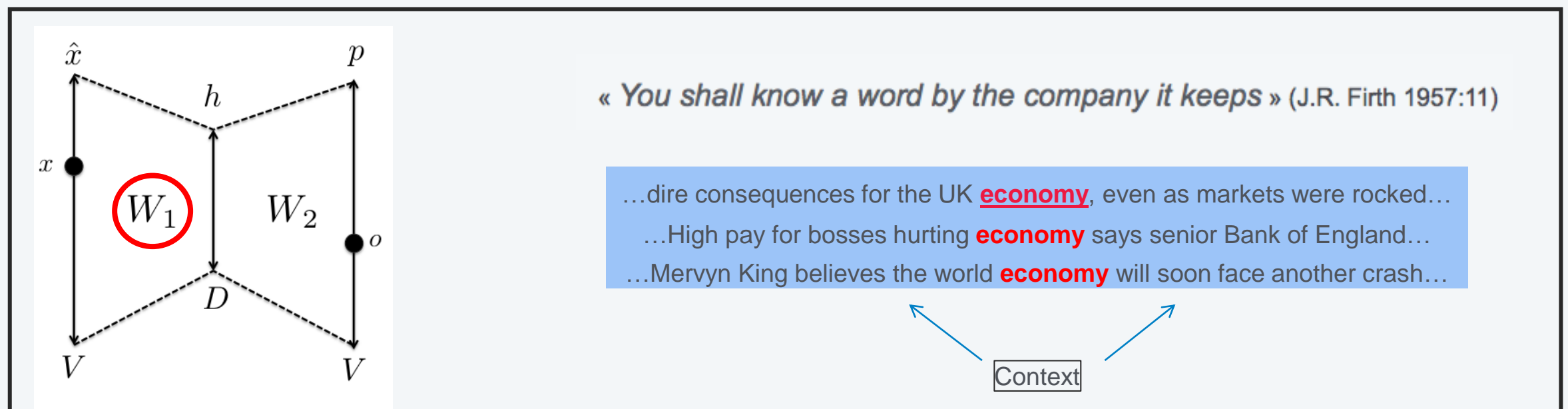
$$w_{\text{France}} - w_{\text{Paris}} \approx w_{\text{Spain}} - w_{\text{Madrid}}$$

# Different Word Embeddings

- We can store all the word vectors into a matrix called : **The Embedding Matrix**.



- The idea of a low dimensional embedding space for words, computed using unsupervised learning was initially explored by Bengio et al. in the early 2000s.
- It started to take off in the industry after the release of the **Word2vec** algorithm, explored by Tomas Mikolov at Google in 2013.
- In the coursework, we will explore the **Word2vec** algorithm.



- In addition to the Word2vec vectors, there are various pretrained word embeddings that can be downloaded and used, like **GloVe** and **FastText**.

# The Embedding Layer

- The **Embedding Layer** takes as input the sequences of integers. But all the sequences should be of the same length  $T$ , so that we can pack them into the same tensor :
  - Sequences that are shorter than  $T$  are padded with zeros.
  - Sequences that are longer than  $T$  are truncated.

## Row Data

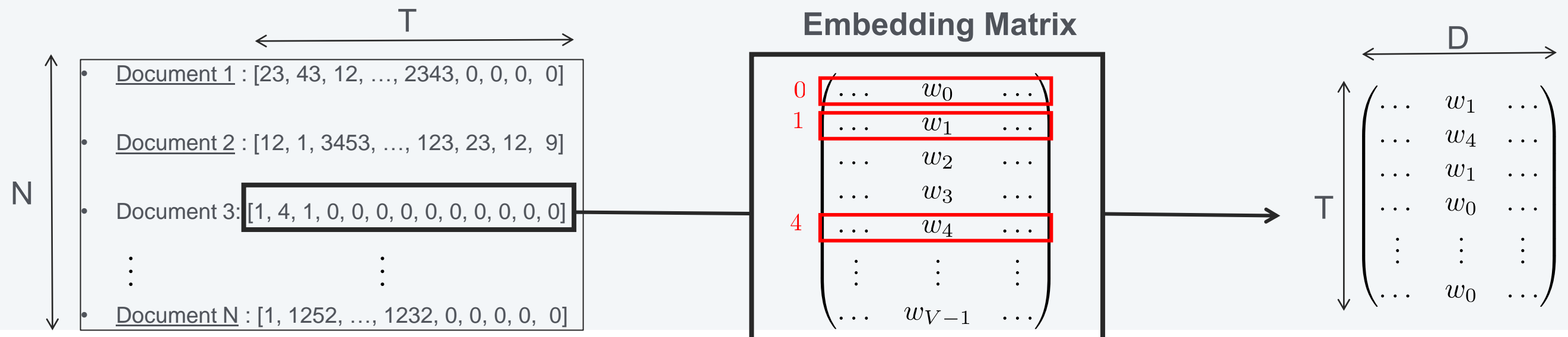
- Document 1 : « There were no wolves in the movie. »
- Document 2 : « This movie has one star and that star is Ryan Gosling. Great flick, highly recommend it. »
- $\vdots$
- Document N : « How many times must Willy be freed before he's freed?. »

## Preprocessed Data

- Document 1 : [23, 43, 12, ..., 2343, 0, 0, 0, 0]
- Document 2 : [12, 1, 3453, ..., 123, 23, 12, 9]
- $\vdots$
- Document N : [1, 1252, ..., 1232, 0, 0, 0, 0, 0]

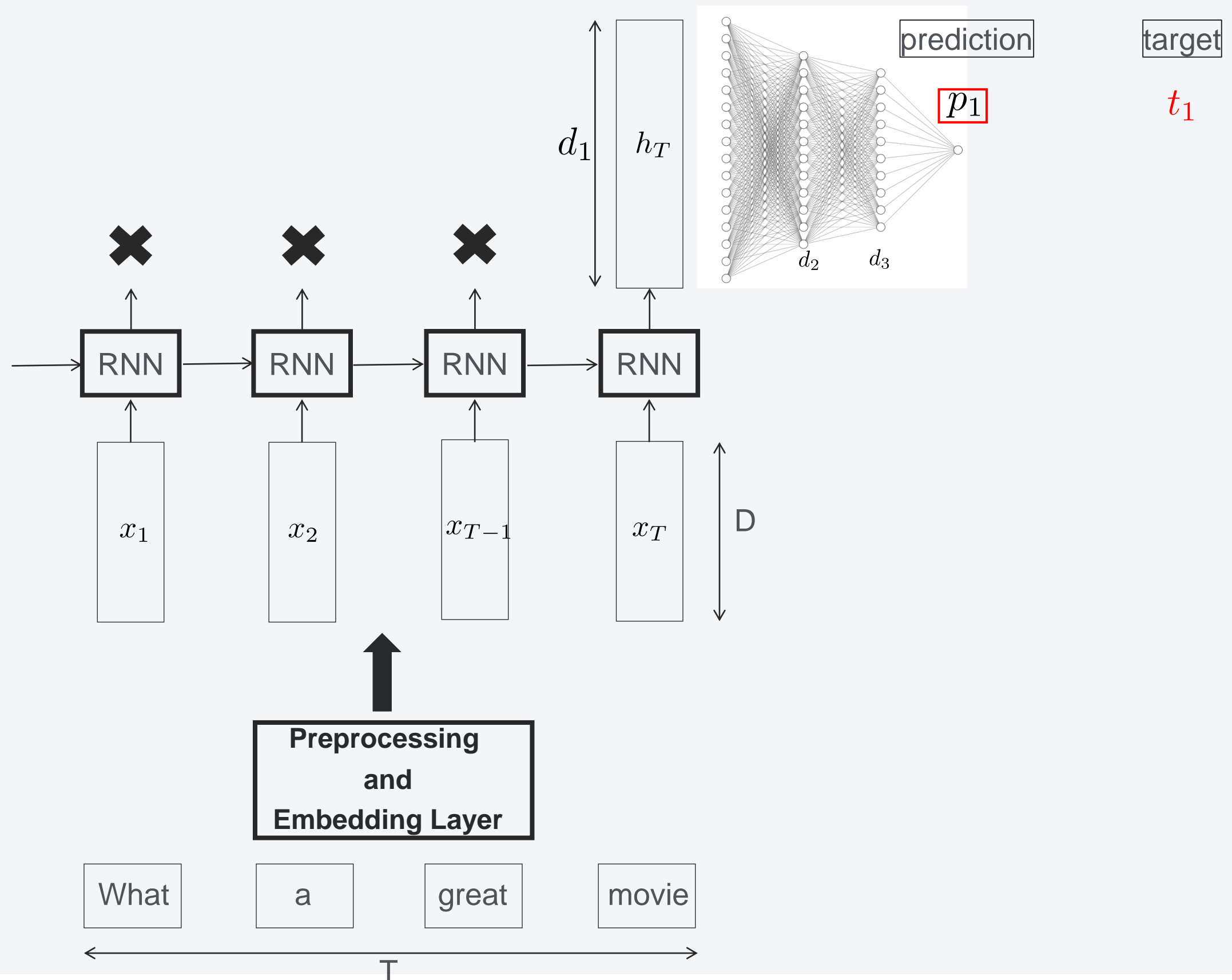
2D tensor of integers, of shape  $(N, T)$

- The Embedding Layer transforms the 2d input tensor of shape  $(N, T)$  into a tensor of shape  $(N, T, D)$ .



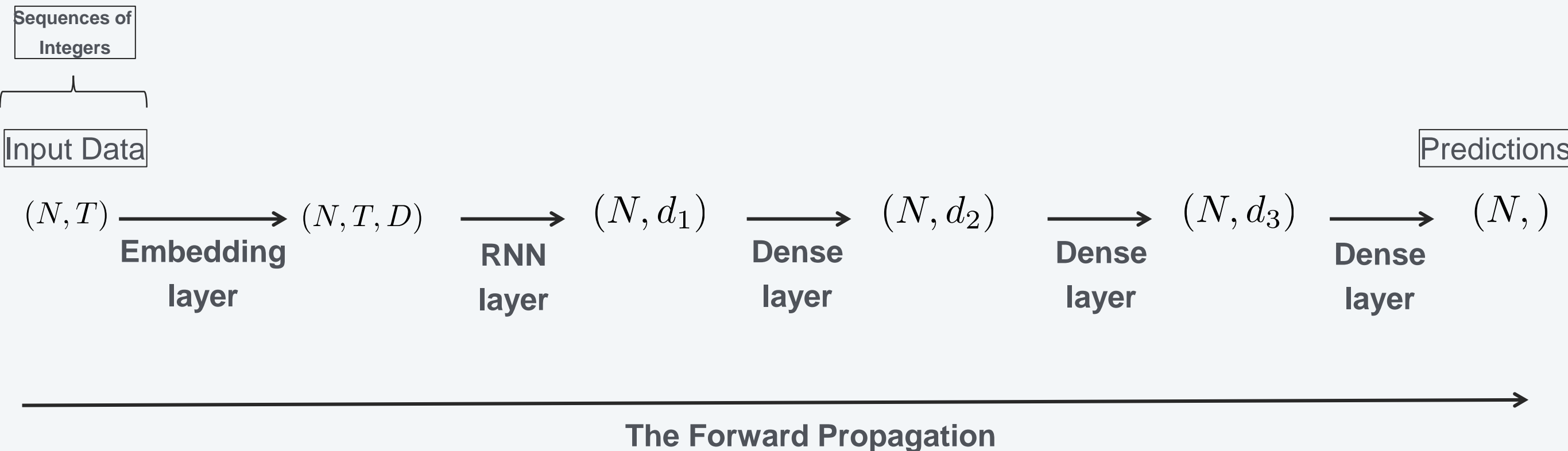


# The Sentiment Analysis new Pipeline – Part 1 –



# The Sentiment Analysis new Pipeline – Part 2 –

- Let's keep track of the evolution of the tensor shape at each layer transformation:



- The next sections will detail the RNN transformation
  - We will first start with a simple RNN model.
  - But the simple RNN model suffers from the vanishing gradient problem
  - We will then explain how to solve the vanishing gradient problem by using a better transformation called the Long Short Term Memory model.

# Part 3 : Recurrent Neural Networks

# A Simple RNN layer – Part 1–

- **The input:** A sequence of length  $T$ , composed of  $D$ -dimensional vectors.

$$x_1, \dots, x_T$$

- **The output:** A sequence of length  $T$ , composed of  $d$ -dimensional vectors.

$$h_1, \dots, h_T$$

- **The weights:**

$$W_{xh} \in \mathbb{R}^{D \times d}, \quad W_{hh} \in \mathbb{R}^{d \times d} \quad \text{and} \quad b_h \in \mathbb{R}^d$$

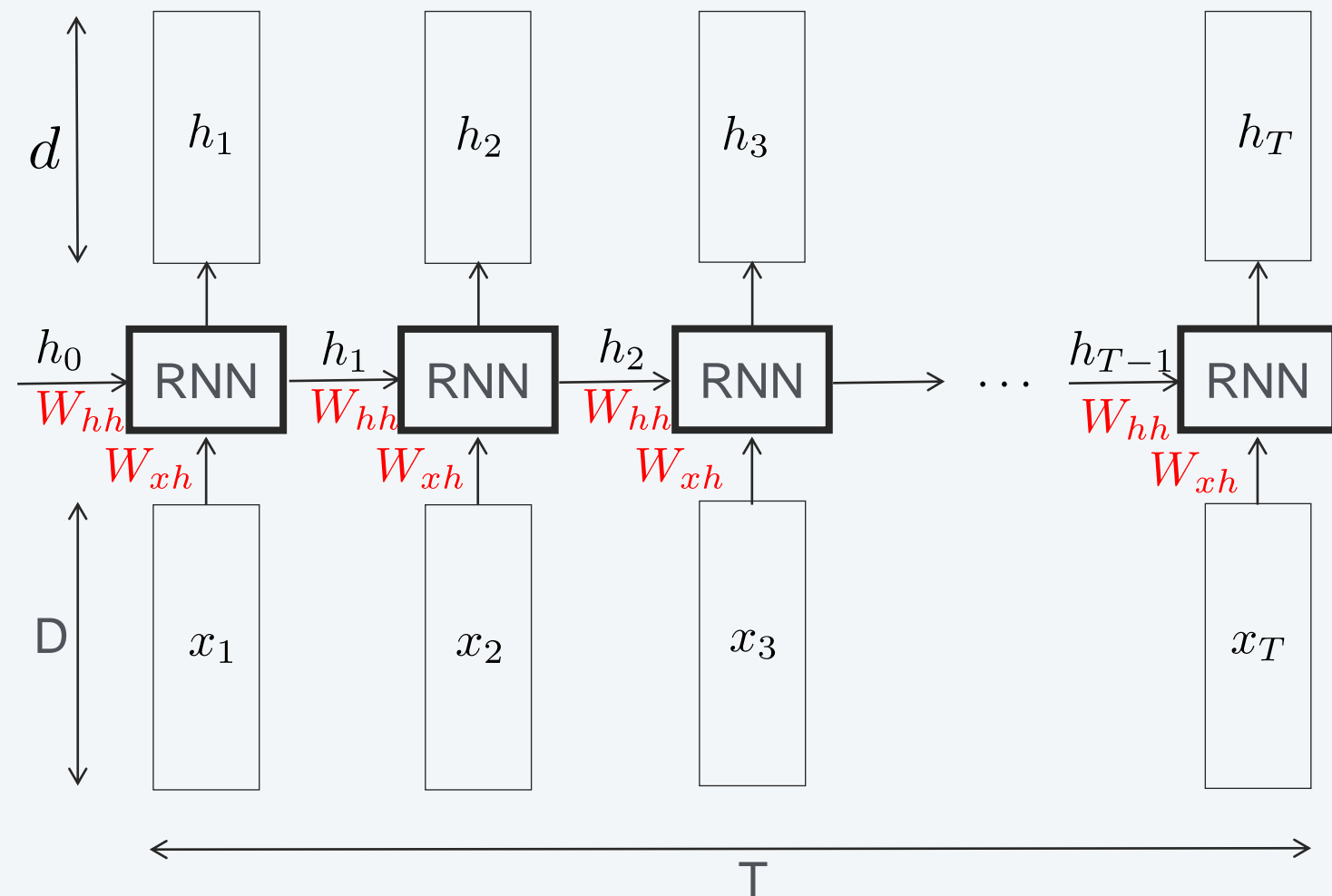
- **The Transformation:**

$$h_1 = \tanh(W_{hh}^T h_0 + W_{xh}^T x_1 + b_h)$$

$$h_2 = \tanh(W_{hh}^T h_1 + W_{xh}^T x_2 + b_h)$$

$$\vdots$$

$$h_T = \tanh(W_{hh}^T h_{T-1} + W_{xh}^T x_T + b_h)$$



$$\forall t \in \{1, \dots, T\}$$

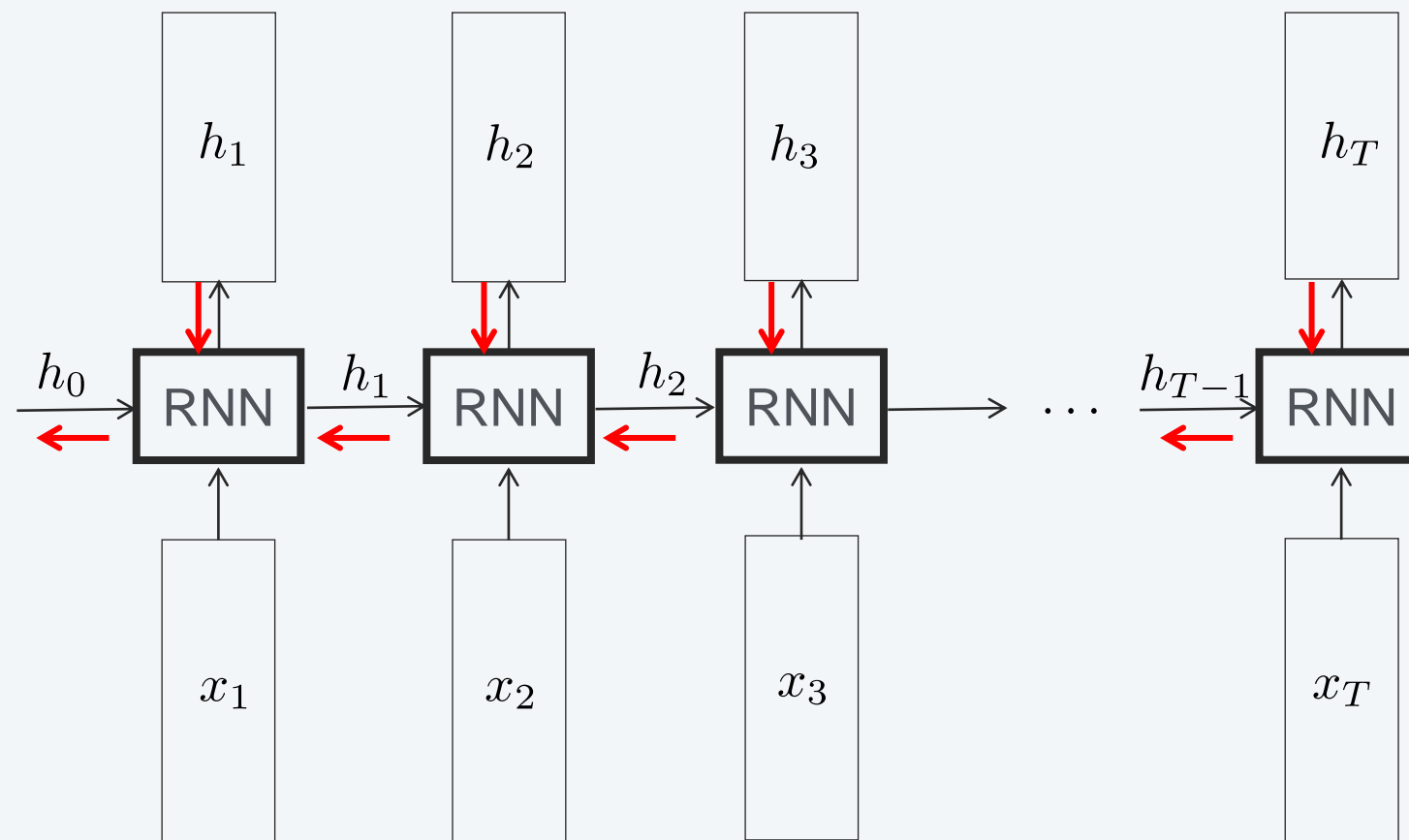
$$h_t = \tanh(W_{hh}^T h_{t-1} + W_{xh}^T x_t + b_h)$$

# Interactive Session



# A Simple RNN layer – Part 2–

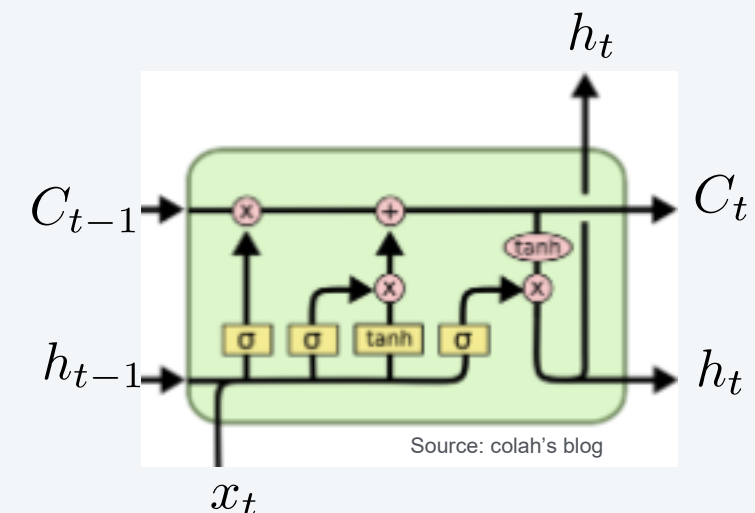
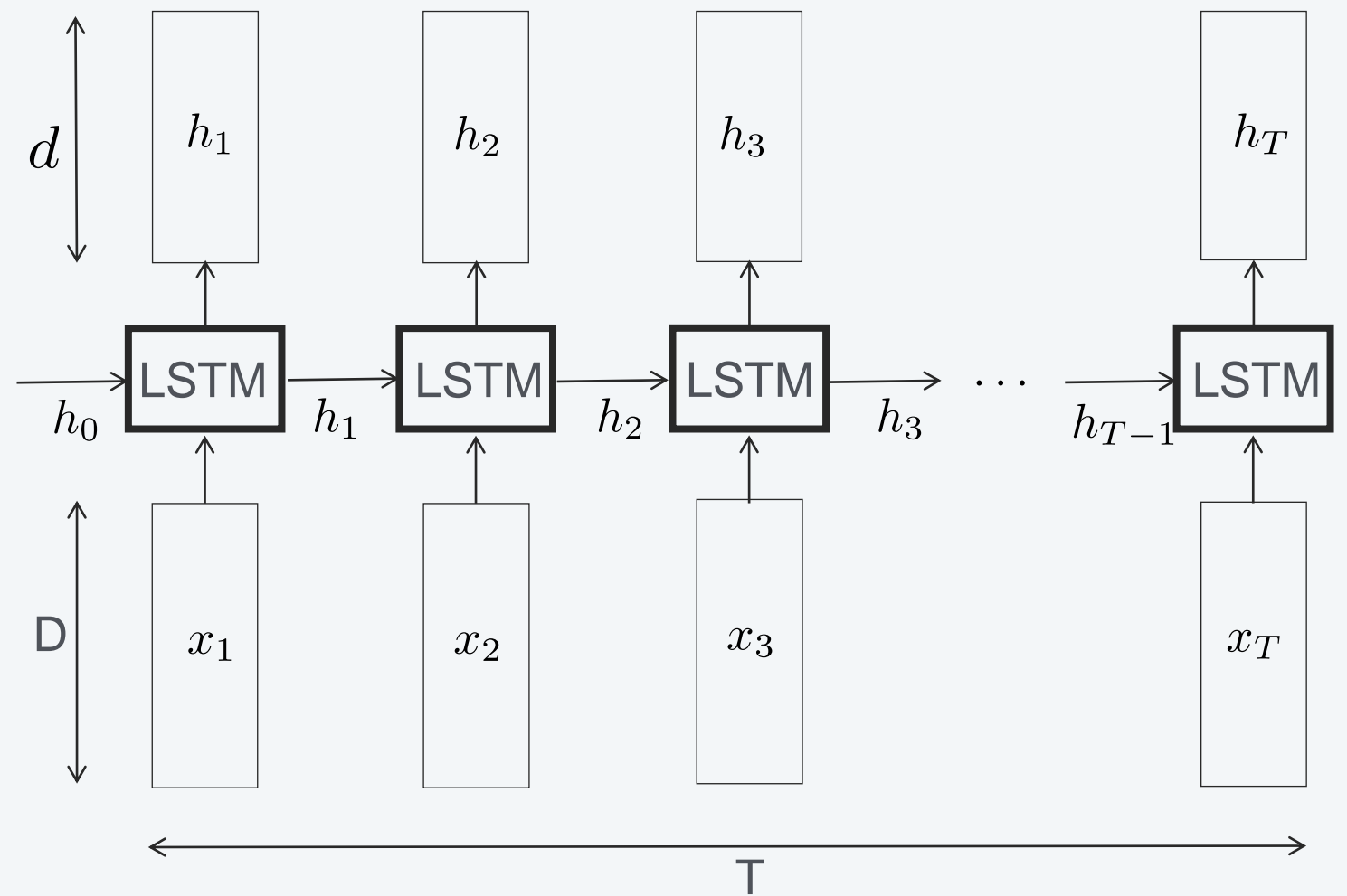
- As usual, we use the Gradient Descent Algorithm to update the weights.



- Unfortunately, simple RNNs aren't capable of learning « long term dependencies » and it's hard to make  $h_T$  influenced by  $x_1, x_2, x_3$  due to the vanishing gradient problem, as explained in:
  - [Hochreiter 1991]: *Untersuchungen zu dynamischen neuronalen Netzen*
  - [Bengio, et al. 1994]: *Learning Long-Term Dependencies with Gradient Descent is Difficult*
  - [Bengio et Mikolov 2013]: *On the difficulty of training recurrent neural networks*

# LSTM networks – Part 1 –

- Long Short Term Memory networks (LSTMs) are a special type of RNN explicitly designed to avoid long term dependency problem
- Like the standard RNNs, LSTMs also have the form of a chain of repeating modules of neural networks.
- Unlike the standard RNNs, the repeating module has four single neural network layers, interacting in a special way.

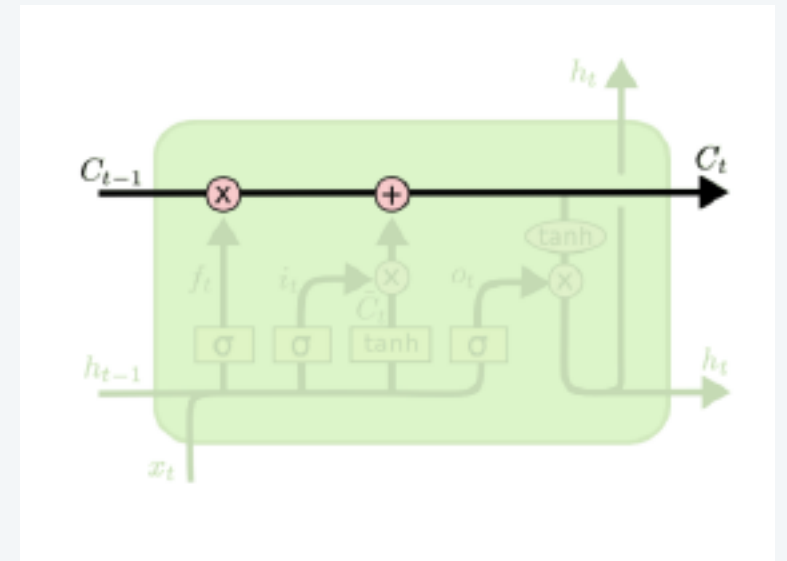


# LSTM networks – Part 2 –

- There are two main concepts in LSTMs:

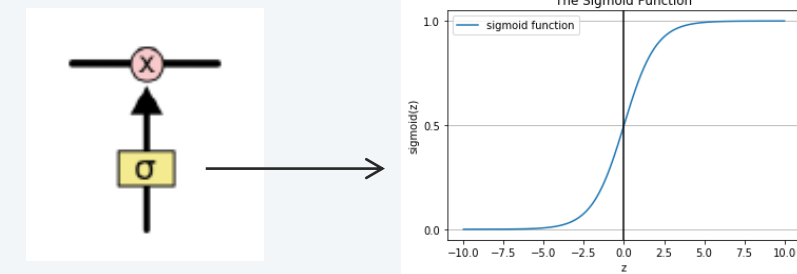
## 1. The cell state :

- It's represented by the sequence  $C_t$  for  $t \in \{1, \dots, T\}$
- The **cell state** represents the memory.
- At each step  $t - 1 \rightarrow t$ , the LSTM will remove some information from the cell state and add some other information using the concept of **gates**.
- The LSTM has 3 gates to protect the cell state.



## 2. The gates:

- The gate is a way to control the amount of information we want to keep or change.
- It's composed of a sigmoid neural network and an element wise multiplication.
- For each dimension, the sigmoid layer outputs a value between zero and 1, describing how much we want to let through: « close to zero » means « let nothing through ! » and « close to one » means « let everything through! ».



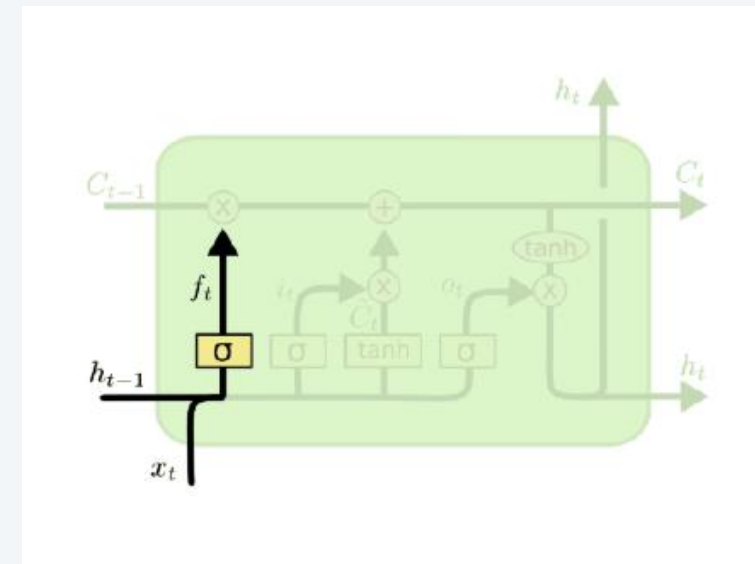


# LSTM networks – Part 3 –

- To transition from  $t-1$  to  $t$  using LSTMs, there are 4 steps:

- STEP 1: The forget gate layer:**

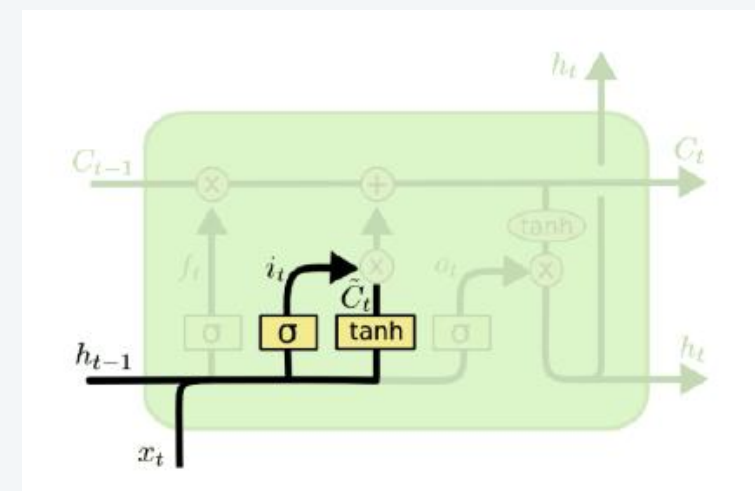
- This step concerns the information we want to throw away from the cell state.
- For that, we use the forget gate layer: A sigmoid applied to the concatenation of  $h_{t-1}$  and  $x_t$  outputs the forget vector  $f_t$
- The parameters are:  $(W_f, b_f)$



$$f_t = \sigma(W_f^T [h_{t-1}, x_t] + b_f)$$

- STEP 2: The input gate layer:**

- Conversely, this step concerns the information we want to store in the cell state.
- For that, we use the input gate layer: Again, a sigmoid applied to the concatenation of  $h_{t-1}$  and  $x_t$  outputs the input vector  $i_t$
- Then, we create a new candidate for the cell state  $\tilde{C}_t$  thanks to a  $\tanh$  layer.
- The parameters are:  $(W_i, b_i)$  and  $(W_C, b_C)$



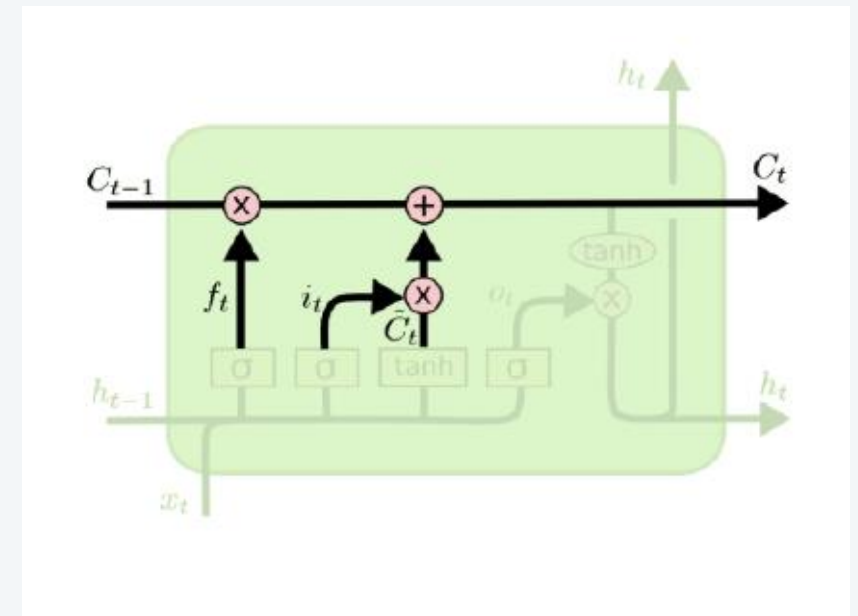
$$i_t = \sigma(W_i^T [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C^T [h_{t-1}, x_t] + b_C)$$

# LSTM networks – Part 4 –

- **STEP 3: Updating the cell state:**

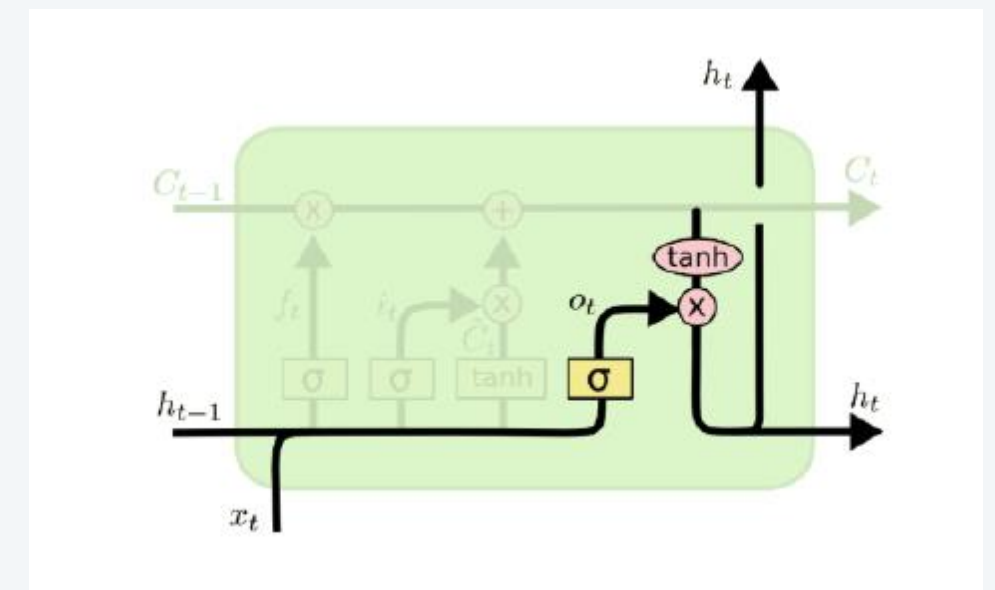
- We know from the previous steps that by multiplying (element wise) the forget vector ( $f_t$ ) by the old cell state ( $C_{t-1}$ ), we obtain the first part of the updated cell state:  $f_t * C_{t-1}$  corresponding to what we want to forget.
- We also know that  $i_t * \tilde{C}_t$  represents the new candidate vector, scaled by how much we want to update each dimension of the cell state.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- **STEP 4: The output gate layer:**

- We want to output a filtered version of the updated cell state.
- For that, we use the output gate layer to decide what dimensions of the cell state we want to output: A simple sigmoid function applied to the concatenation of  $h_{t-1}$  and  $x_t$  gives the output vector  $o_t$
- The output at time t is just the element wise multiplication of the output vector and the updated cell state (after a tanh transformation).
- The parameters are:  $(W_o, b_o)$

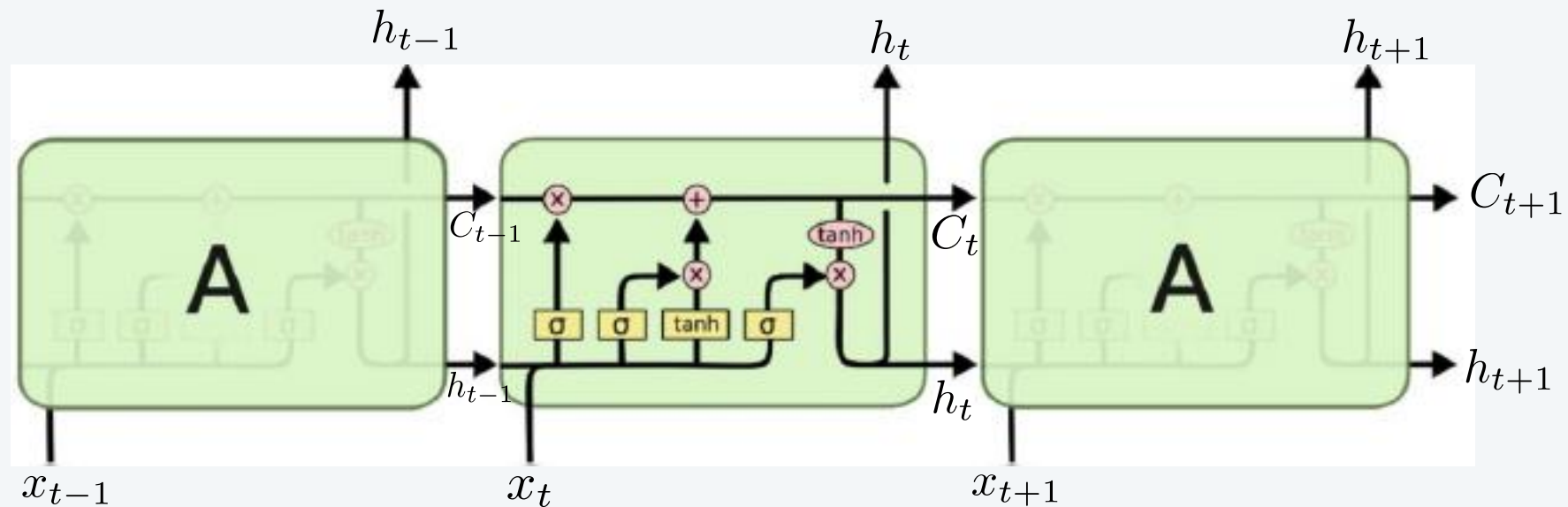


$$o_t = \sigma(W_o^T [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

# LSTM networks – Part 5 –

- Summary:



## Equations of the LSTM

The gates:

$$f_t = \sigma(W_f^T [h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i^T [h_{t-1}, x_t] + b_i)$$

$$o_t = \sigma(W_o^T [h_{t-1}, x_t] + b_o)$$

The updates:

$$\tilde{C}_t = \tanh(W_C^T [h_{t-1}, x_t] + b_C)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

$$h_t = o_t * \tanh(C_t)$$

## Part 4 : Programming Session

# Programming Session





**Go to the following link and take Quiz 6 :**

**<https://mlfbg.github.io/MachineLearningInFinance/>**