

# How does a DL model make predictions?

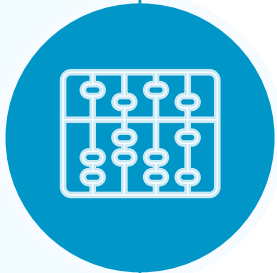
Deep learning models use information from the past to make accurate predictions for the future.

- ▶ **E.g., Financial markets** using information from the past to predict the future stock value. Brokers can use the insight to see the market trends before making crucial decisions.



# To make use of the past information: Recurrent Neural Network (RNN)

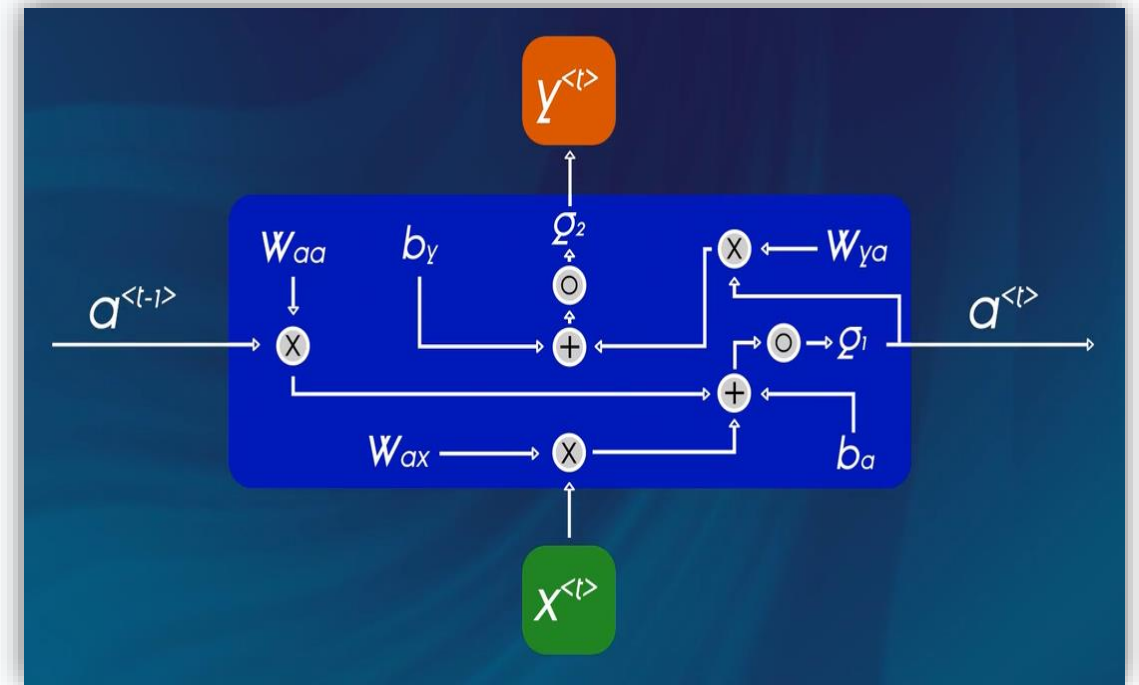
## What do RNNs do?



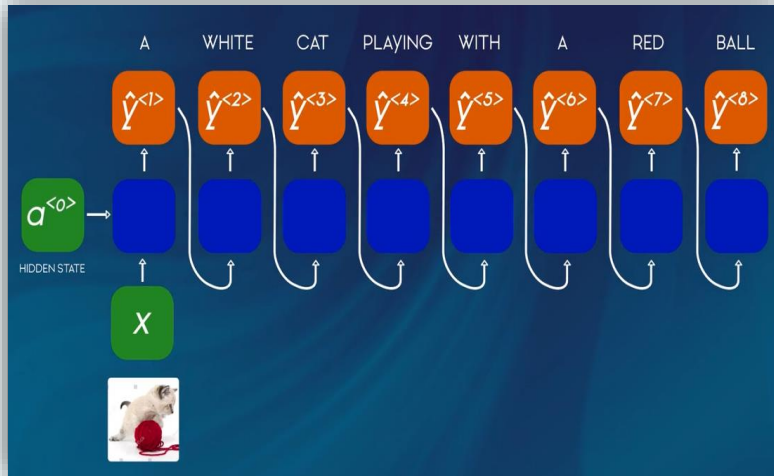
- **RNNs** are designed to process the sequential data, in which each data point is connected to the previous and or the future datapoint.
- A traditional **RNN** has an input layer, a hidden state and an output layer. The hidden state represents information from previous time steps and is responsible for summarizing and storing information from the past.

# How do RNNs work?

- 1 The model takes the input ( $x^{<t>}$ ) and multiplies it by its associated weights ( $w_{ax}$ )
  - 2 The model takes the hidden state ( $a^{<t-1>}$ ) and multiplies it by its associated weights ( $w_{aa}$ )
  - 3 Then, it adds these two products & biases and applies the activation function ( $g_1$ )
- **The result** of these steps will give the input to the output layer ( $y^{<t>}$ ) and the hidden state for the next time step ( $a^{<t>}$ ).

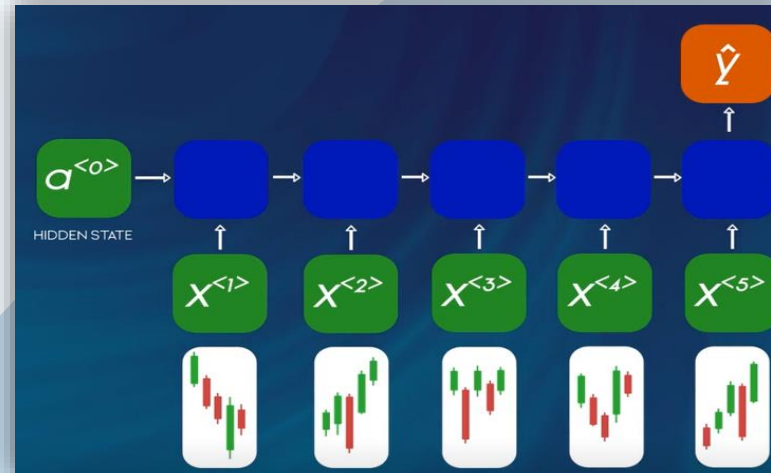


# Different types of RNNs and their applications



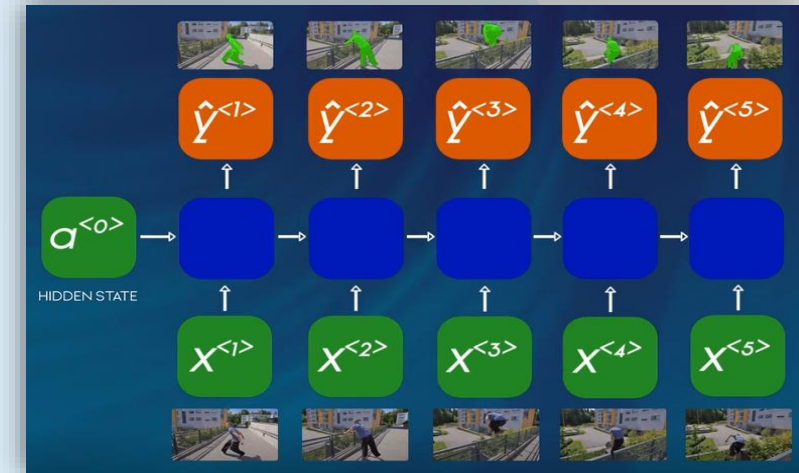
## (I) One-to-many

The model generates a sequence of words using images as input



## (II) Many-to-one

The model makes predictions using past information as input



## (III) Many-to-many

The model tracks objects in videos using video frames as input

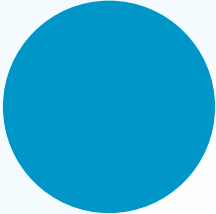
# Training RNNs

## Where do we start?

- The first step of the training is to **calculate the loss**, which is the sum of the losses at each time step.
- Backpropagation is performed at each time step to **calculate the gradient** from the first time step to the last time step.
- Then all these parameters are updated accordingly.

$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}(\hat{y}^{<t>}, y^{<t>})$$

# What could go wrong with training?



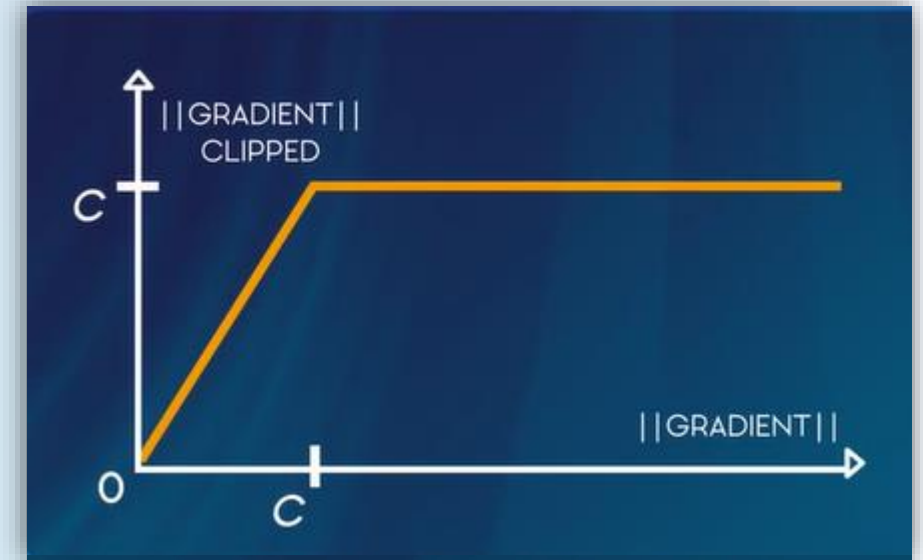
If the sequences are long, the hidden state must represent the information from all previous time steps, which means that the computation from the first time step could affect the computation of the output at time step "x".

- This could cause the gradients to explode, **i.e., the gradient goes to infinity**. As a result, the RNN might not be able to learn because the gradient descent is not meaningful, and **the training becomes very unstable**.

# To cope with the exploding gradient problem: gradient clipping

## How does gradient clipping work?

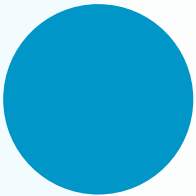
- During backpropagation, the maximum value of the gradient can be capped.
- When the gradient exceeds a certain maximum value, we assign the capping value to it. In this way, no number reaches infinity.





# Another problem with training: vanishing gradients

## When this happens?



This is the case where the gradients become **zero** or **very close to zero**. It usually happens when RNNs are not able to capture long term dependencies in the input sequence.

For example, in a text prediction model, a vanishing gradient can cause predictions to be based on the last few words and not the entire context of the text.



To solve this problem, we can use two modified versions of RNNs:  
**GRUs & LSTMs**



# Gated Recurrent Units and Long Short-Term Memory Units: GRUs and LSTMs

There are two most common types of RNNs to deal with vanishing gradients problem

Gated Recurrent Unit  
(GRU)

Has two gates\*:  
(1) reset gate  
(2) update gate

Long Short-Term  
Memory Unit  
(LSTM)

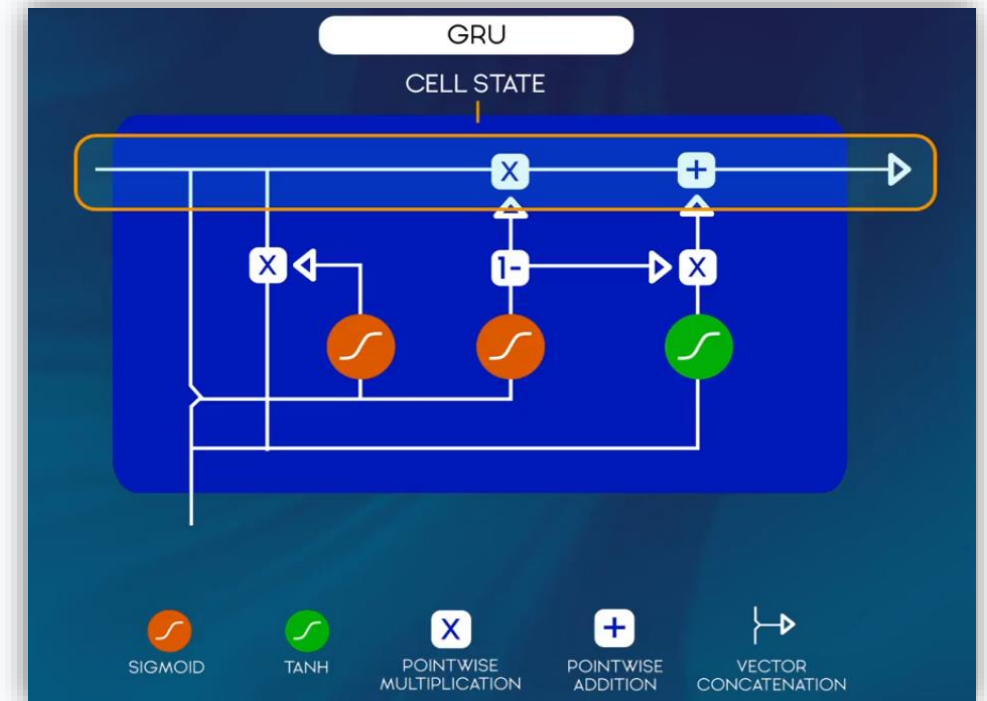
Has three gates:  
(1) input gate  
(2) forget gate  
(3) output gate

\*A gate = a neural network that regulates the information flowing through the sequence chain

# Gated Recurrent Units: GRU cells

In GRUs the hidden state is referred to as a candidate cell (also serves as the output)

- ➔ The **update gate** decides if the cell state\* should be updated with the candidate state, the current activation value, or not.
- ➔ The **reset gate** decides whether the previous cell state is important or not.
- ➔ The **cell state** for the next time step is dependent on the update gate. It may update the cell state by adding some useful information and by removing some information that is not required.

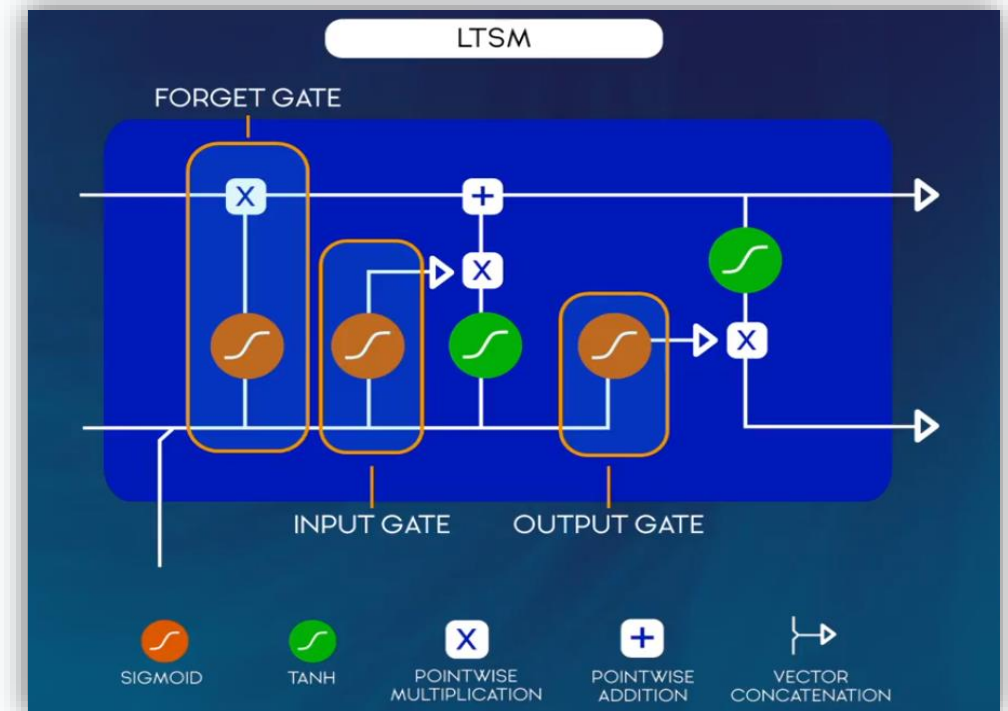


\*Cell state = transfers relative information down the sequence chain

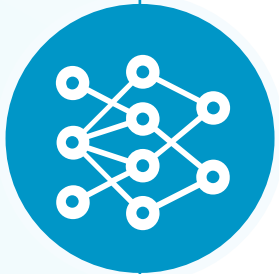
# Long Short-Term Memory Units: LSTMs

The hidden state = candidate cell (also serves as the output)

- ➡ The **input gate** decides what relevant information can be added from the current step.
- ➡ The **forget gate** decides how much information from the previous state should be kept and forget the rest.
- ➡ The **output gate** controls which parts of the cell are output to the hidden state and determine what the next hidden state will be.



# What is the purpose of GRUs & LSTMs?



Useless or redundant information is dropped, and only useful information is carried to the next time step by the candidate cell. This **does not let the candidate cell to be overwhelmed with information** and thus it can solve the vanishing gradient problem simply.

# What are the differences between GRUs and LSTMs?

## GRUs

- Two gates
- Less complex structure
- Preferred for smaller datasets

## LSTMs

- Three gates
- More complex structure
- Preferred for larger datasets