Recurrent Neural Networks

Intro to Deep Learning

Sequential data

So far, our focus has been on handling unordered data, where we simply fed a single matrix or image into the model.

However, when dealing with sequences like time-series data, disregarding the order of the data would lead to improper training.

Inputting such sequential data as-is without considering its order can adversely affect the learning process.

Sequential Data: Daily stock price, weather, document (text), sound, video, and so on.

Recurrent Neural Network

Working with time series data can be surprisingly straightforward.

The key is to input the data in chronological order.

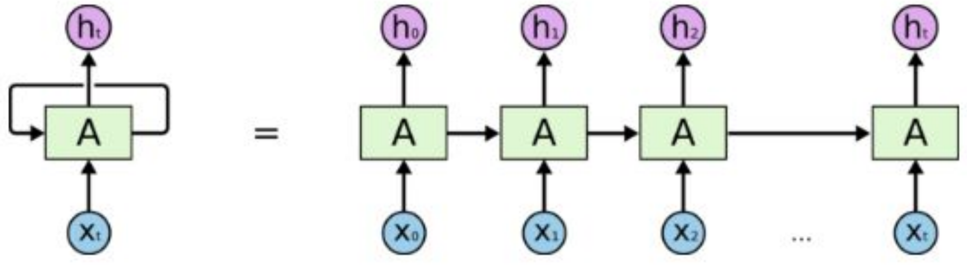
To build an effective model, you must incorporate both historical and current information.

This is achieved by feeding the model's previous output back into the input, a technique known as recursion.

In the realm of deep learning, this is referred to as a **Recurrent Neural Network** (RNN).

Why recurrent?

RNN is a method of storing previously input data for a while when multiple data are input in sequence. And it judges how important the memorized data is and gives it a separate weight and moves on to the next data. It does this for every input value in sequence, so it looks like it's hovering around the same layer before moving on to the next layer.



An unrolled recurrent neural network.

RNN - memory of previous node

In a recurrent neural network (RNN), the output of the previous node, known as the **memory** or hidden state, is utilized as input for the current node.

This characteristic holds significant importance as RNNs are specifically designed to handle sequential data, where each element's information in the sequence relies on the preceding elements.

By incorporating the previous memory as an input for the current node, the RNN **can effectively consider the contextual information from earlier steps**, enabling it to make informed predictions and decisions based on the historical context.

Memory of history

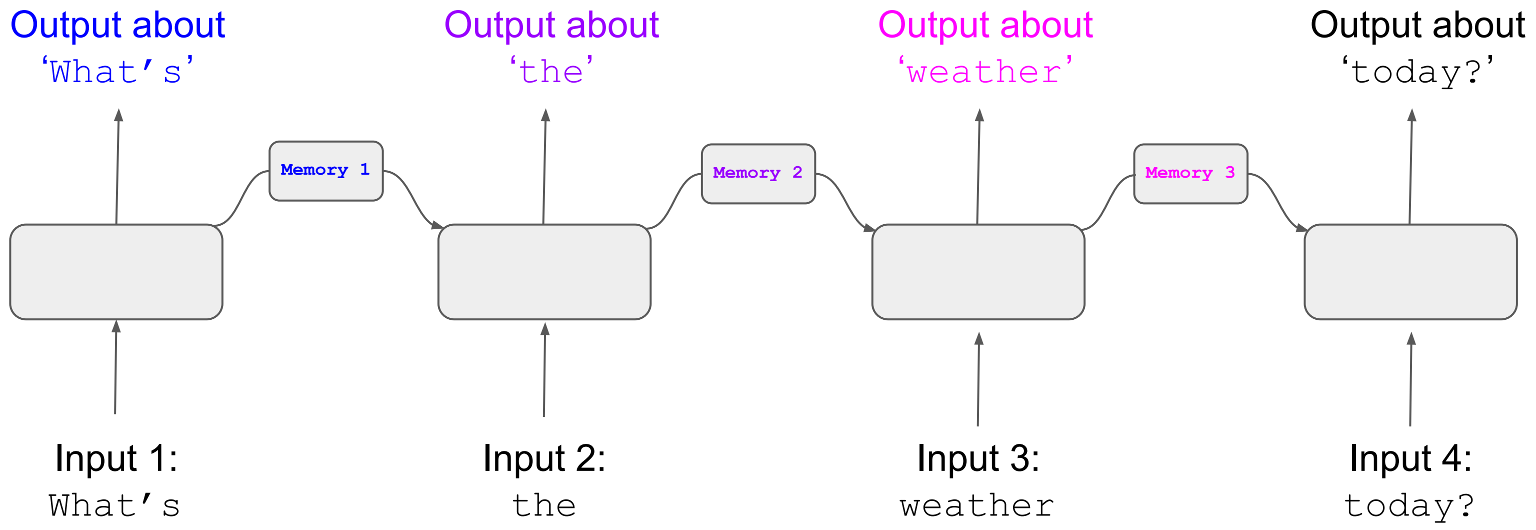
The incorporation of the previous memory serves a vital purpose, enabling the model to comprehend temporal dependencies effectively.

This feature proves highly beneficial across various applications like language modeling, speech recognition, and time series prediction.

By preserving a memory of past inputs, the RNN becomes adept at capturing long-term dependencies and employing that knowledge to anticipate future outputs.

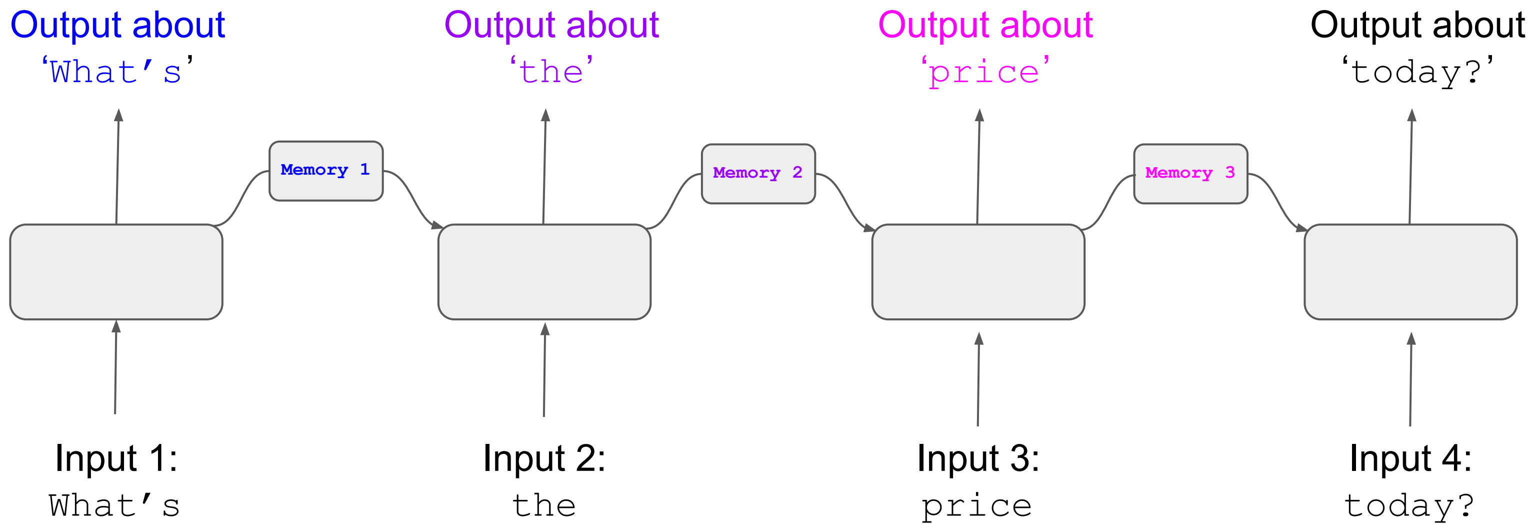
Essentially, the previous memory plays a pivotal role in facilitating the RNN's ability to learn from historical information, ultimately enhancing its predictive capabilities and decision-making prowess.

RNN

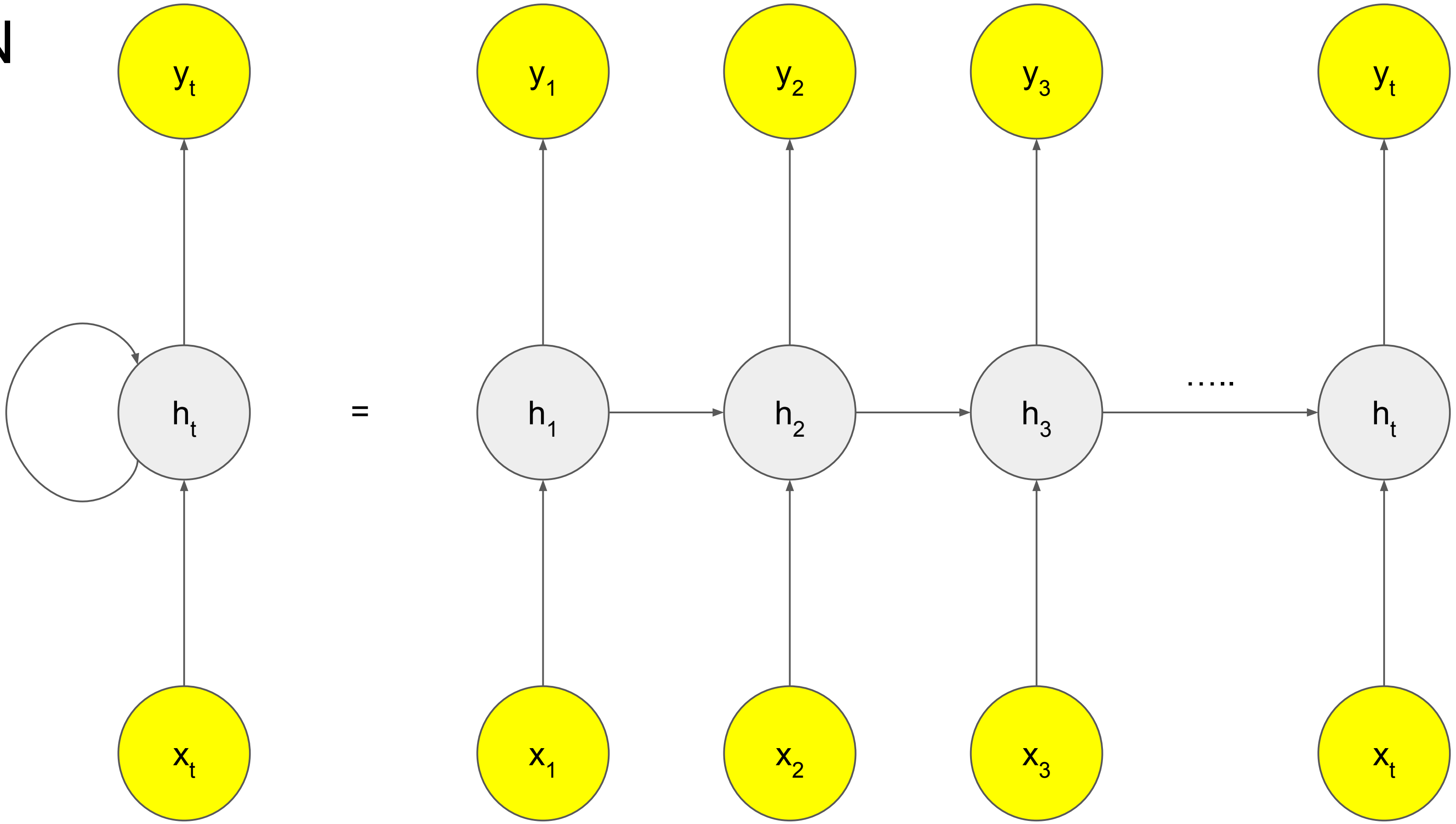


Output about 'What's'
Output about 'the'
Output about 'weather'
Output about 'today?'
Memory 1
Memory 2
Memory 3
Input 1: What's
Input 2: the
Input 3: weather
Input 4: today?

RNN


Output about
'What's'
Output about
'the'
Output about
'price'
Output about
'today?'
Memory 1
Memory 2
Memory 3
Input 1:
What's
Input 2:
the
Input 3:
price
Input 4:
today?

RNN



RNN

When the first input (x_1) comes in, the first memory (h_1) is created. When the second input (x_2) comes in, the existing memory (h_1) is referenced along with the new input to create a new memory (h_2). This process can be repeated for any length of inputs.

In short, the input is (x) from the data and memory (h), and the output becomes y along with the memory (h).

RNN type

One-to-One: It is difficult to call it an RNN because there is no recurrence. It is a basic neural network structure where each input produces one output, without any recurrent connections.

One-to-Many: It is a structure where one input produces multiple outputs. A typical example is **image captioning**, where an image is given as input, and a sentence describing the image is generated as the output.

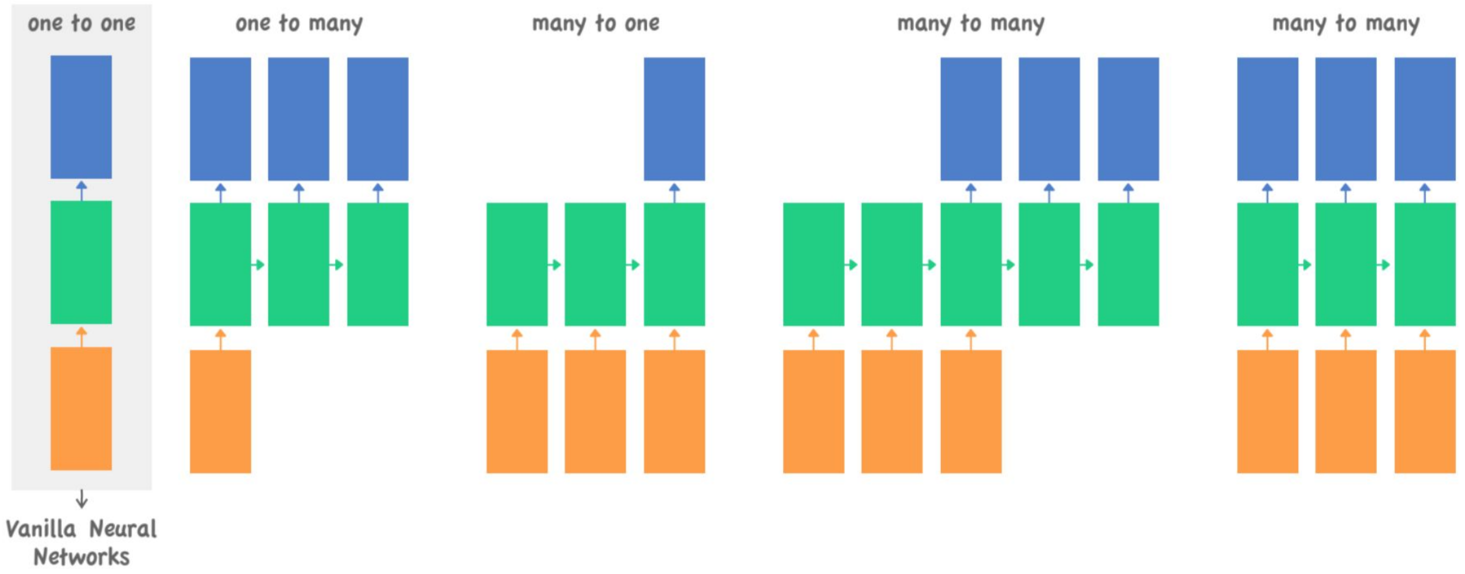
Many-to-One: It is a structure where multiple inputs produce one output. A **sentiment analyzer** is a representative example of this. It takes a sentence as input and outputs the sentiment (positive/negative) of that sentence.

Many-to-Many: It is a structure where multiple inputs produce multiple outputs. The lengths of the input and output sequences can be different. **Machine translation** or speech recognition tasks can be examples of this structure.

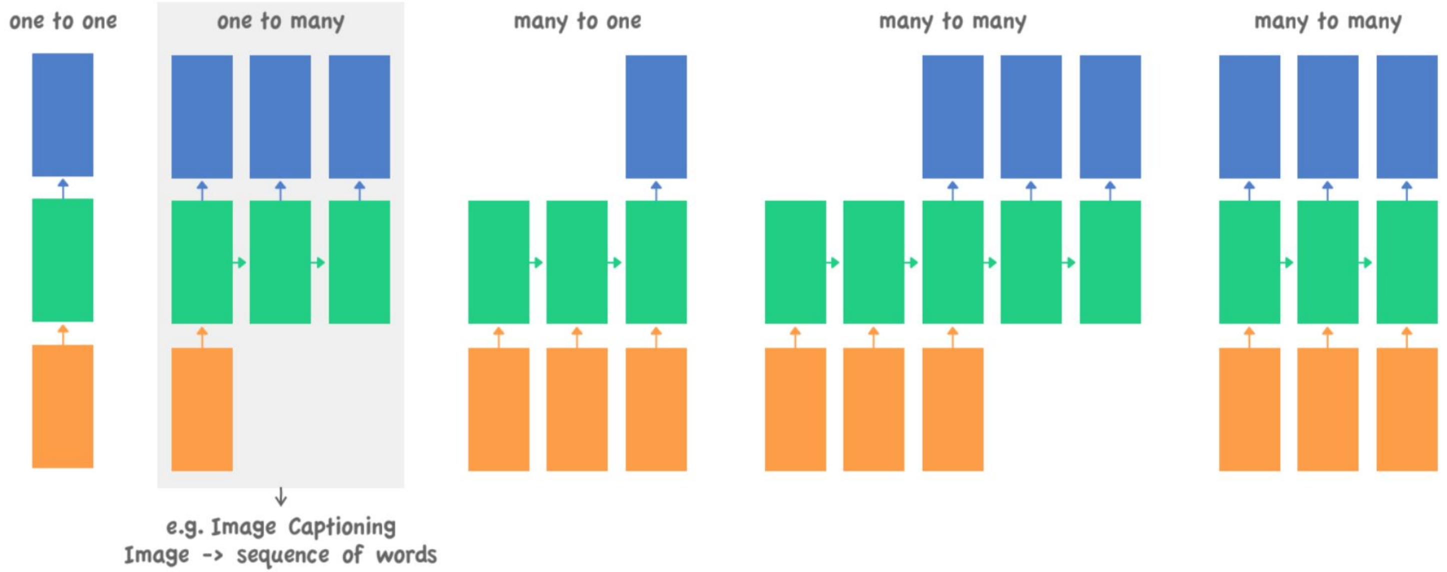
Applications

- Language Modeling
- Speech Recognition
- Machine Translation
- Conversation Modeling / Question Answering
- Image / Video Captioning
- Image / Music / Dance Generation

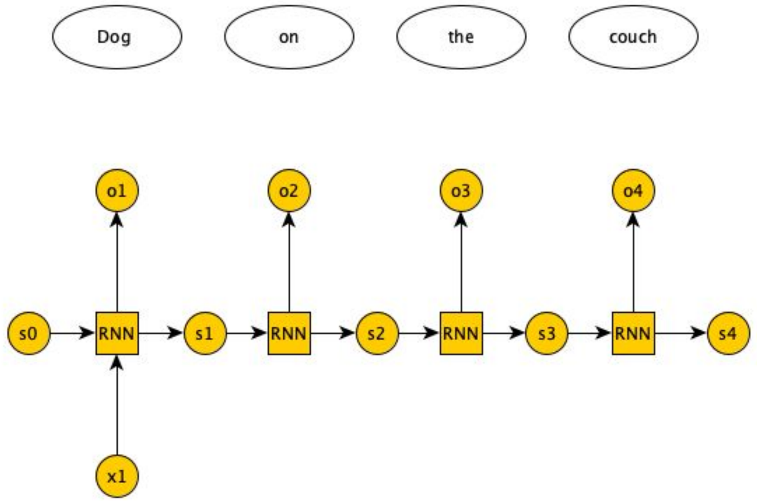
Recurrent Networks Offer a Lot of Flexibility

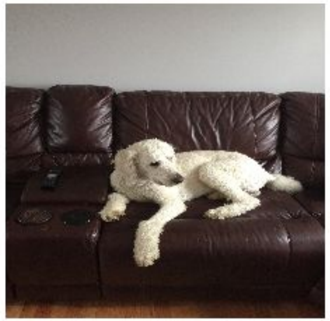


Recurrent Networks Offer a Lot of Flexibility

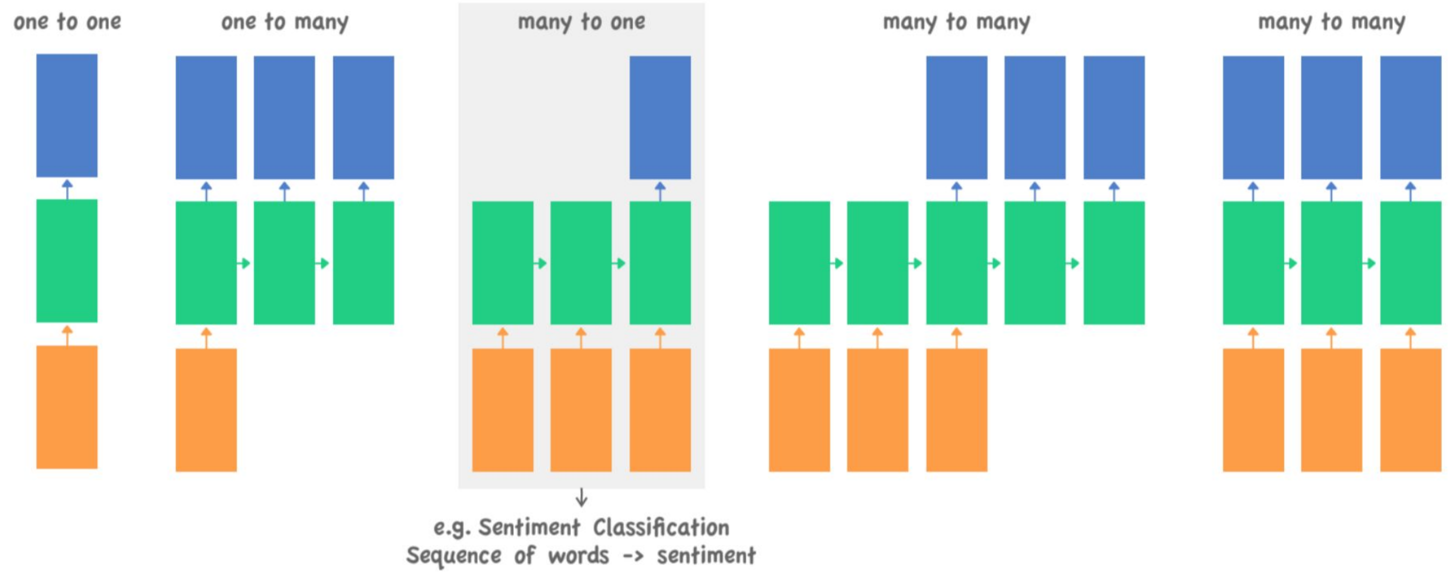


Example

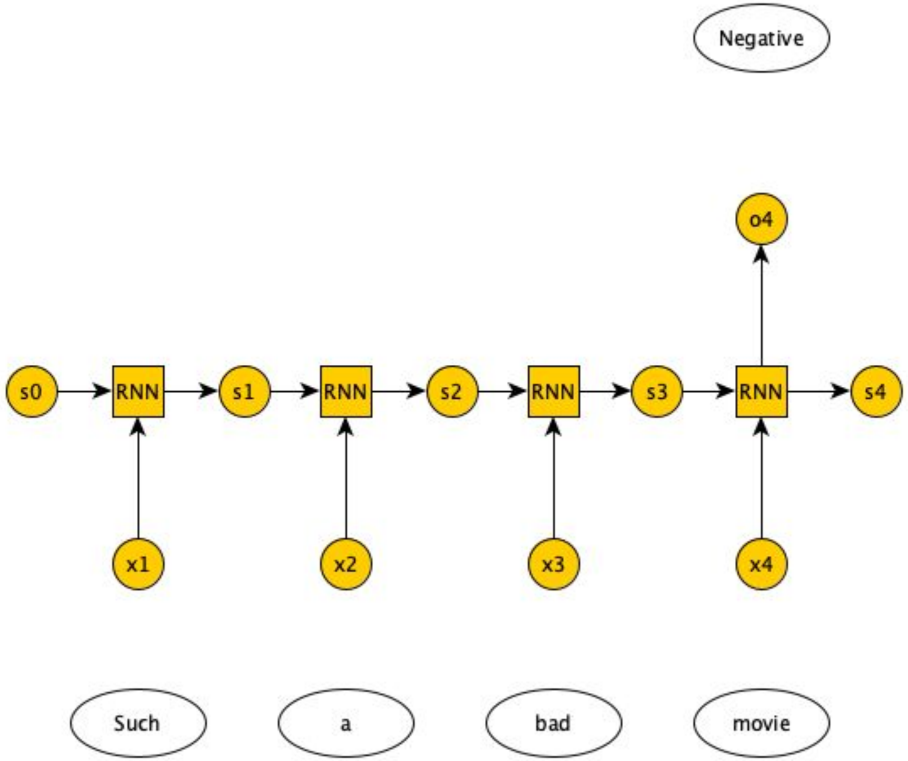




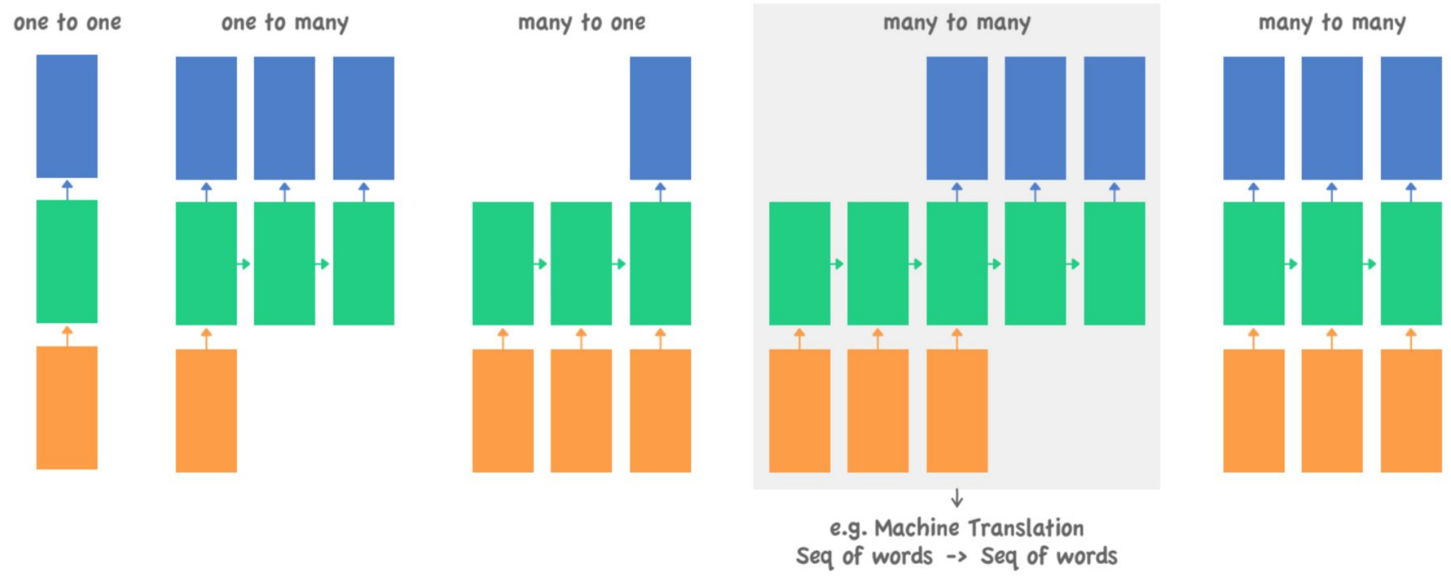
Recurrent Networks Offer a Lot of Flexibility



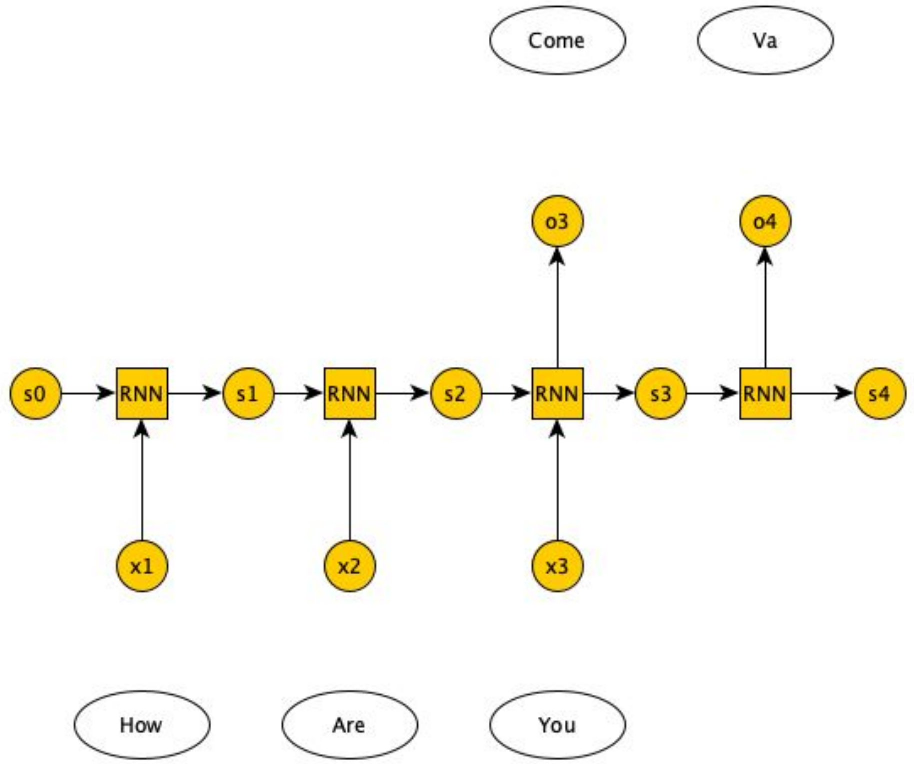
Example


Negative
o4
s0
RNN
s1
RNN
s2
RNN
s3
RNN
s4
x1
x2
x3
x4
Such
a
bad
movie

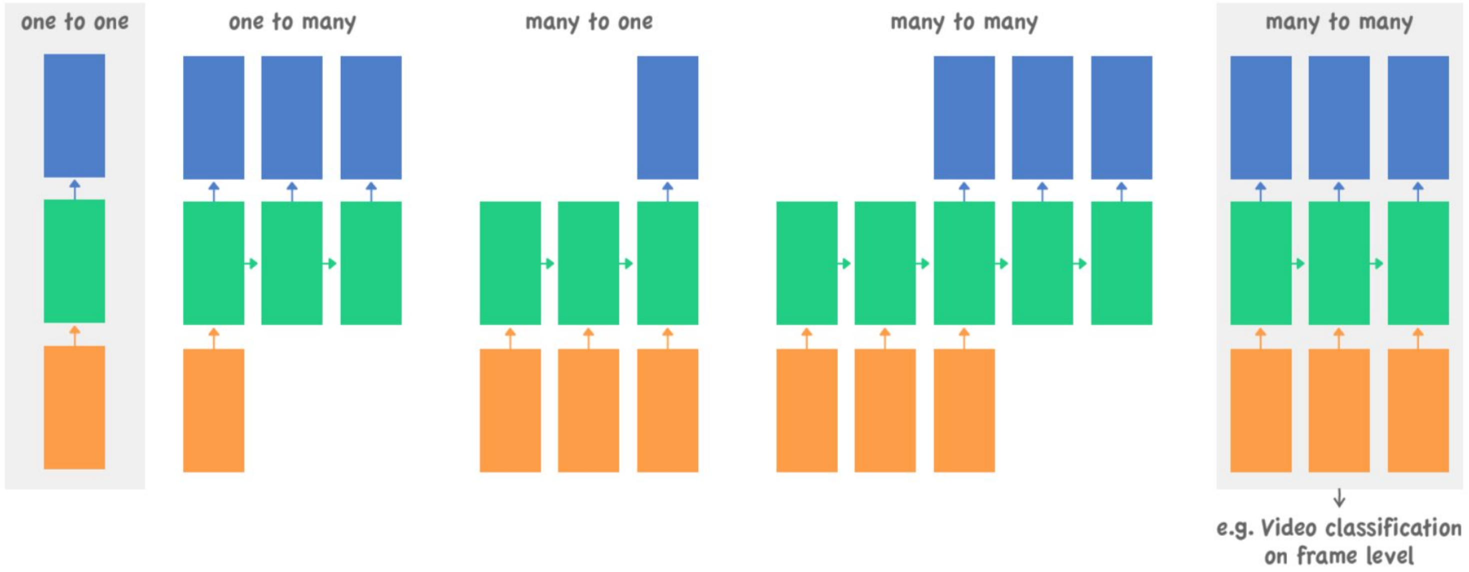
Recurrent Networks Offer a Lot of Flexibility



Example

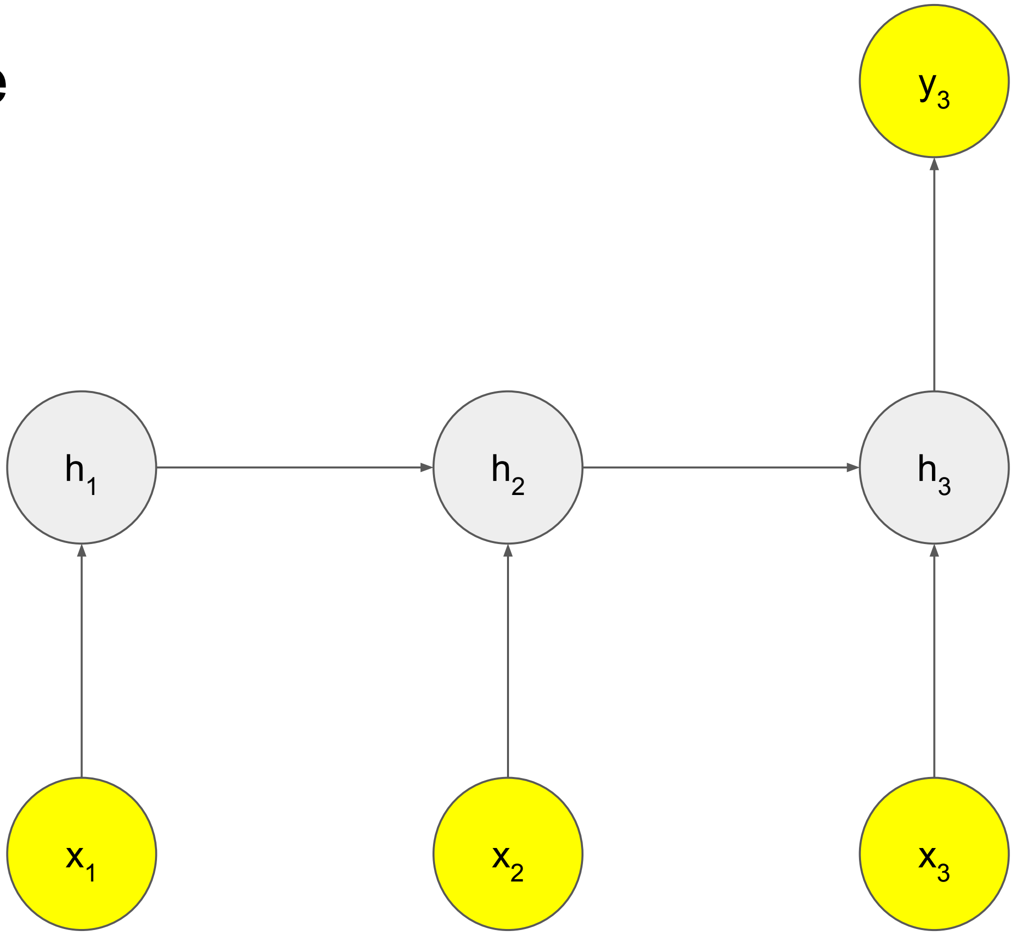


Recurrent Networks Offer a Lot of Flexibility



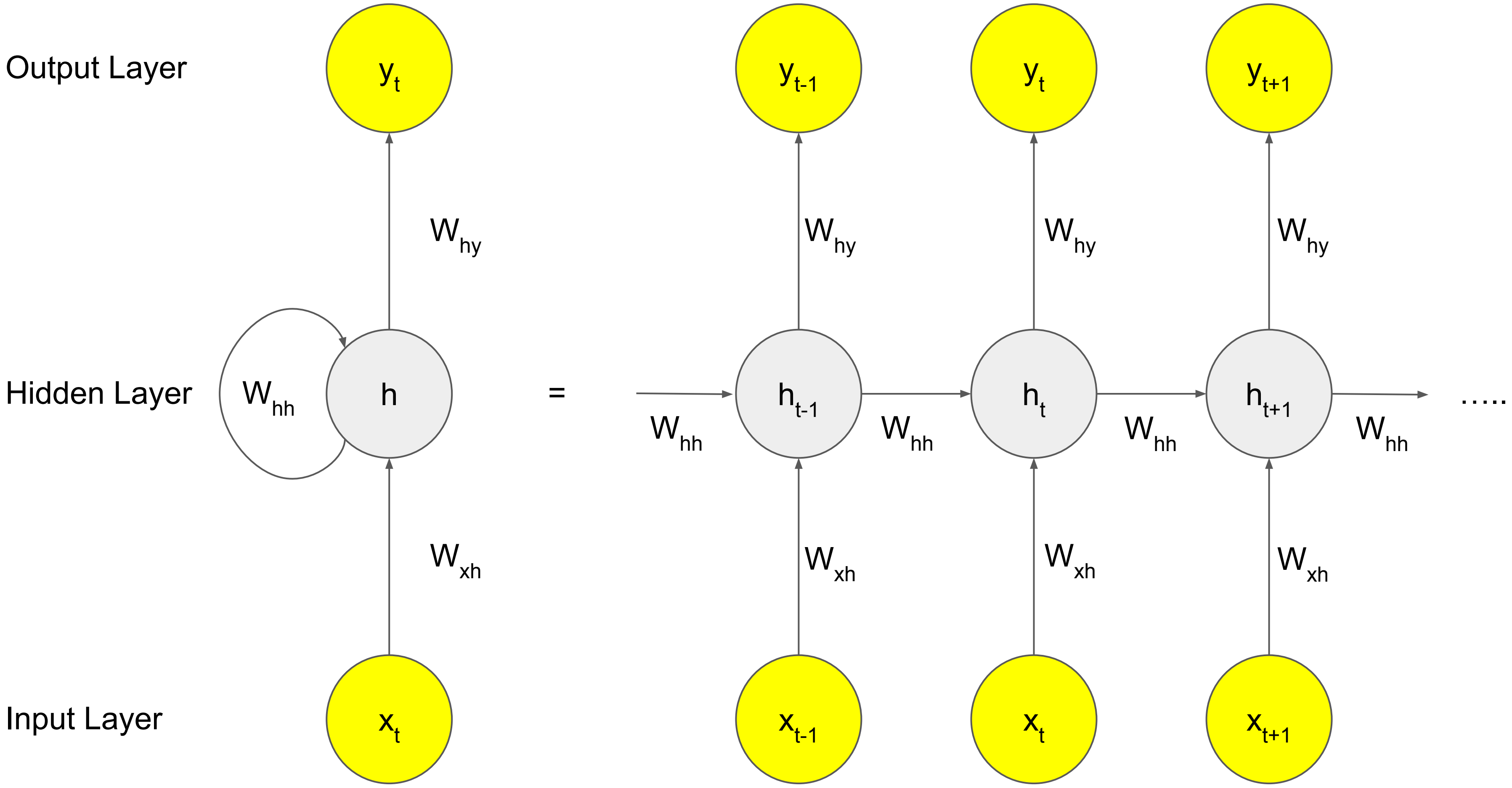
RNN Structure

Many-to-one



Many-to-one

$$\begin{tikzcd}
 & & & y_3 \\
\text{Layer 2} & h_1 \arrow[r] & h_2 \arrow[r] & h_3 \arrow[u] \\
\text{Layer 1} & h_1 \arrow[r] \arrow[u] & h_2 \arrow[r] \arrow[u] & h_3 \arrow[u] \\
 & x_1 \arrow[u] & x_2 \arrow[u] & x_3 \arrow[u]
\end{tikzcd}$$


Output Layer
Hidden Layer
Input Layer
y_t
W_{hy}
W_{hh}
h
W_{xh}
x_t
=
y_{t-1}
y_t
y_{t+1}
W_{hy}
W_{hy}
W_{hy}
h_{t-1}
h_t
h_{t+1}
.....
W_{hh}
W_{hh}
W_{hh}
W_{hh}
W_{xh}
W_{xh}
W_{xh}
x_{t-1}
x_t
x_{t+1}

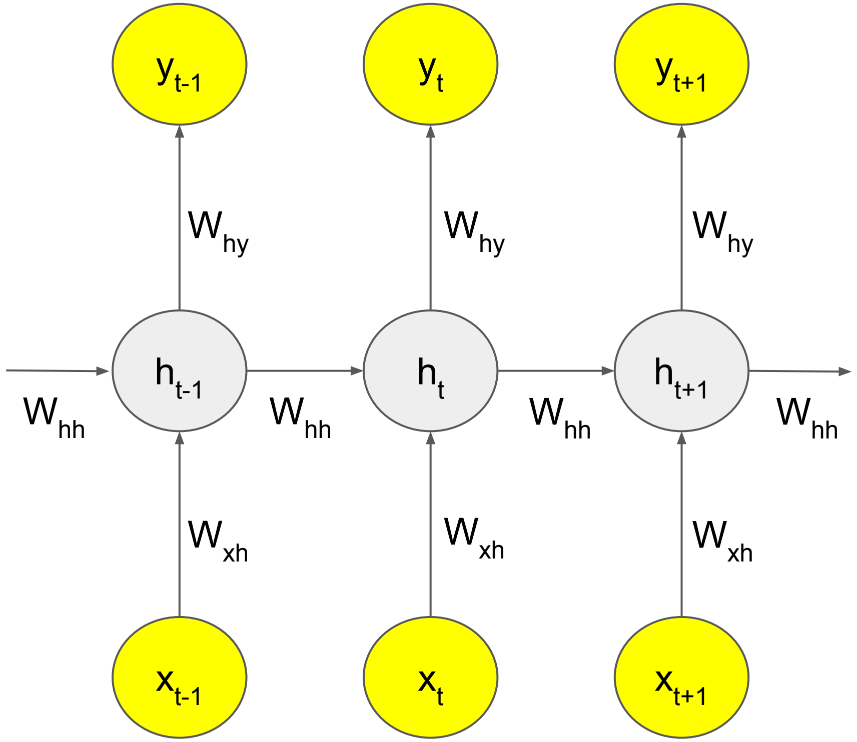
Hidden Layer and Output Layer

- Hidden Layer

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

- Output Layer

$$y_t = softmax(W_{hy}\, h_1)$$



Let’s implement a simple RNN structure!