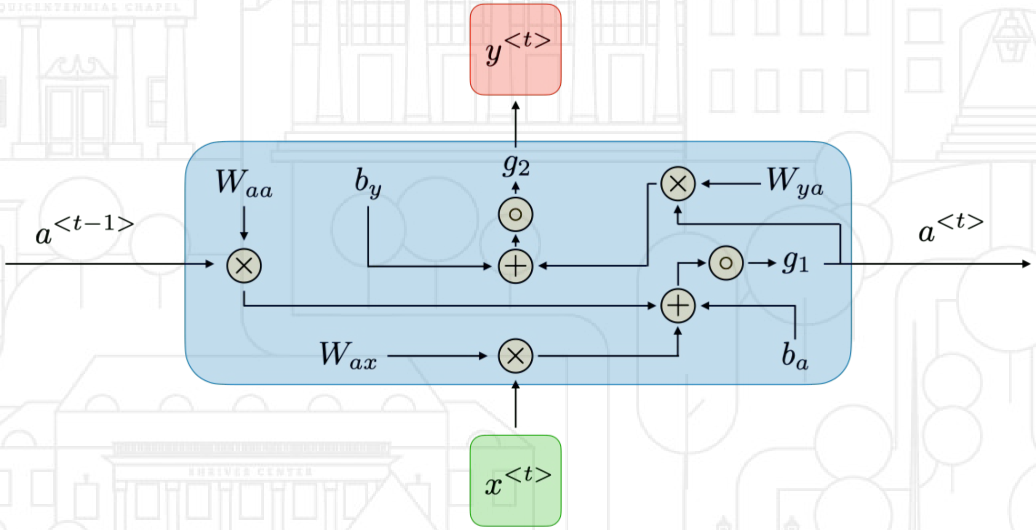
Module 6

Recurrent Neural Networks

DL: RNNs



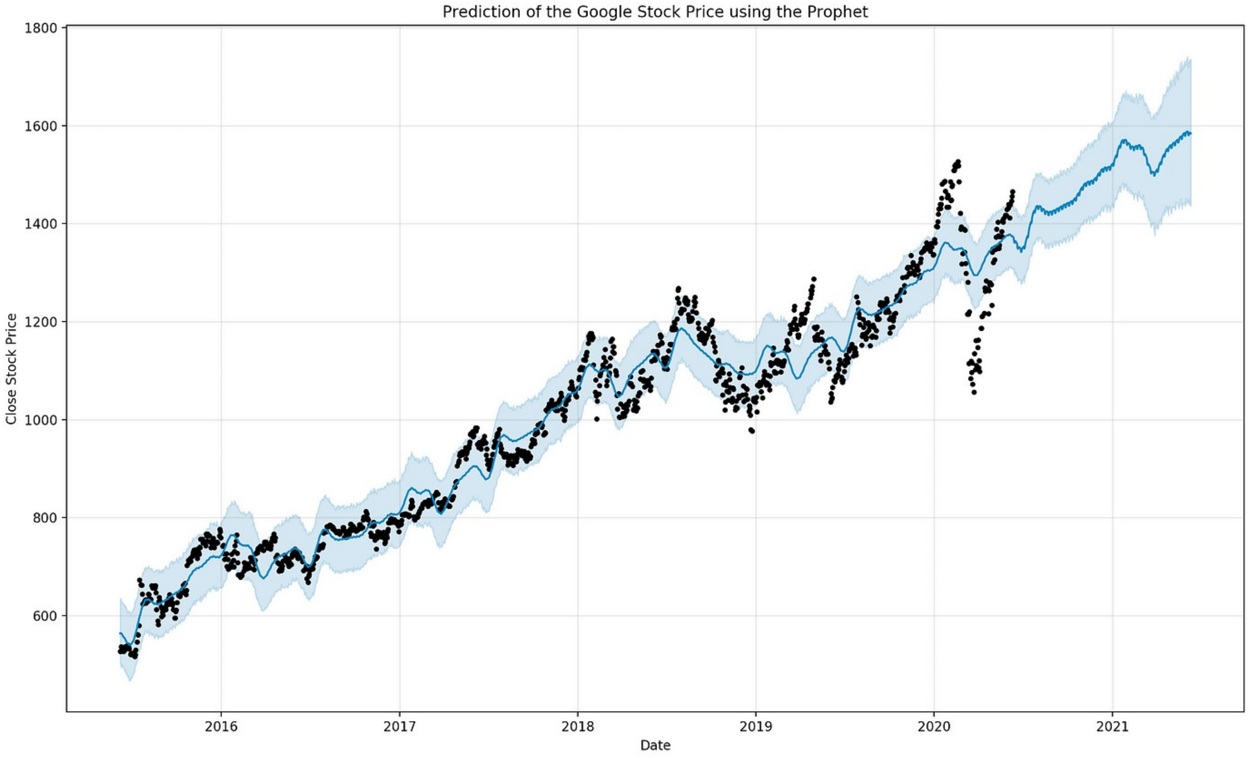
Sequences

- Enumerated collection of objects
- Repetitions in sequences are allowed and order matters.
- Time series, speech rely on changes over a time interval

$$\{a_n\} = \{ a_1, a_2, a_3, \dots , a_n\}$$

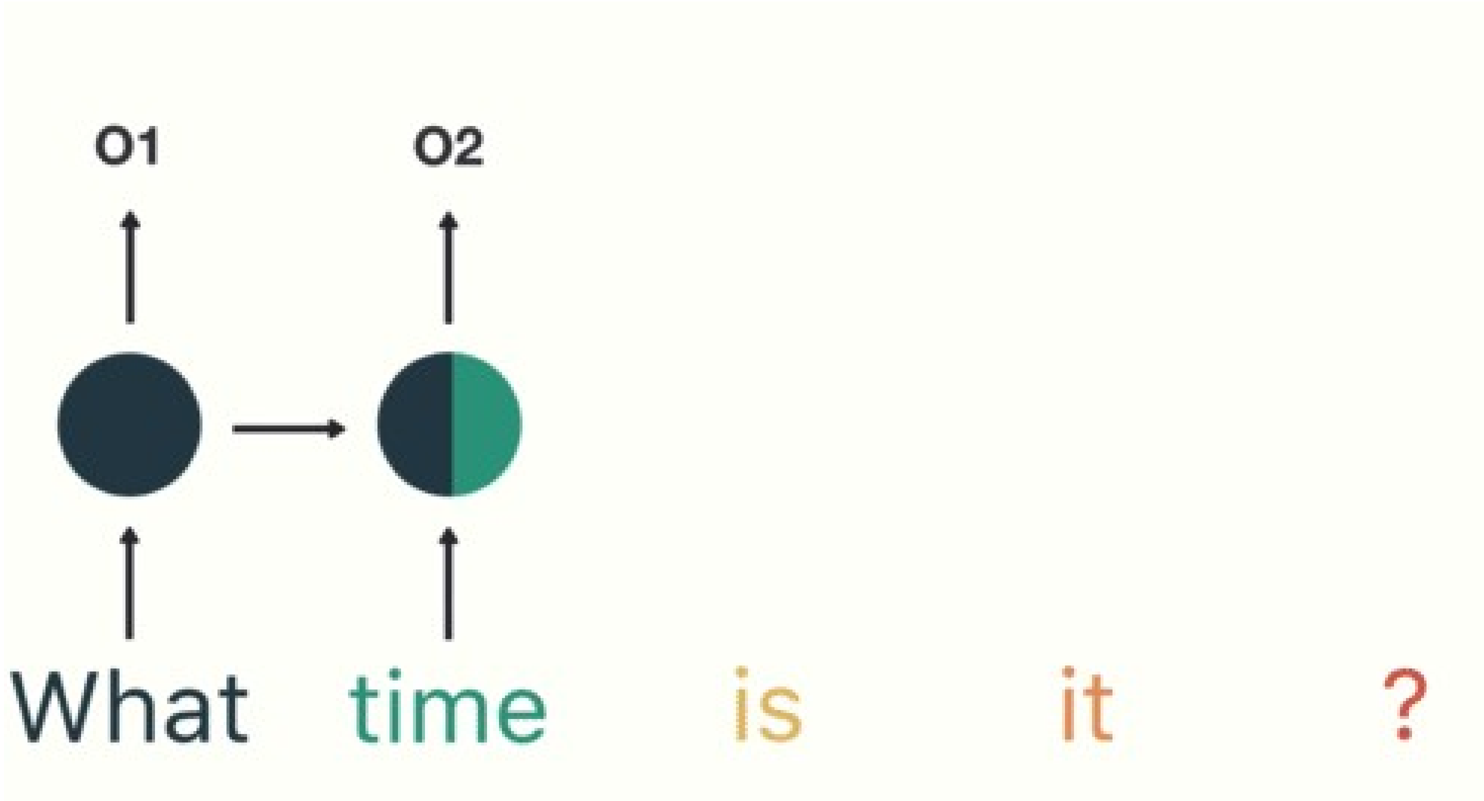
Sequences Examples

- Stock price, financial instruments, inventory predictions



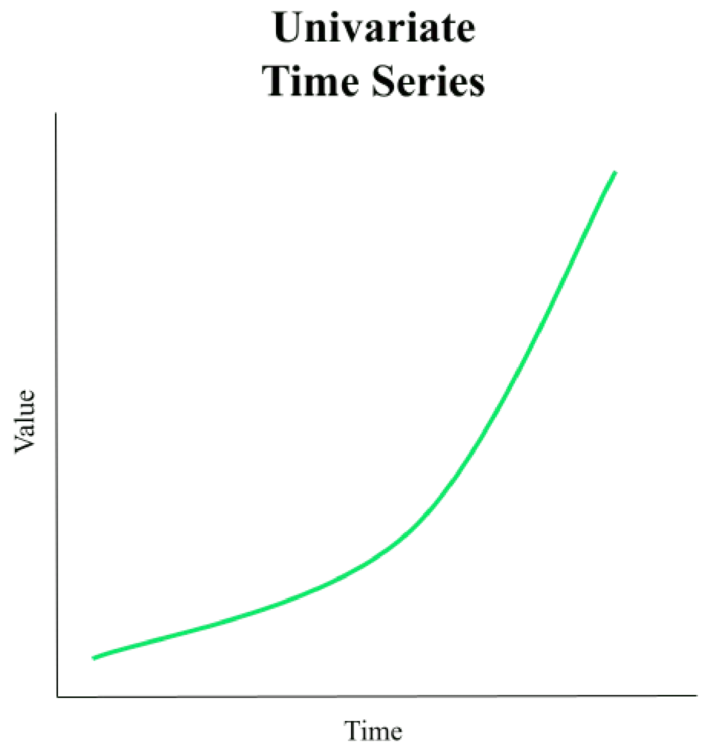
Sequences Examples

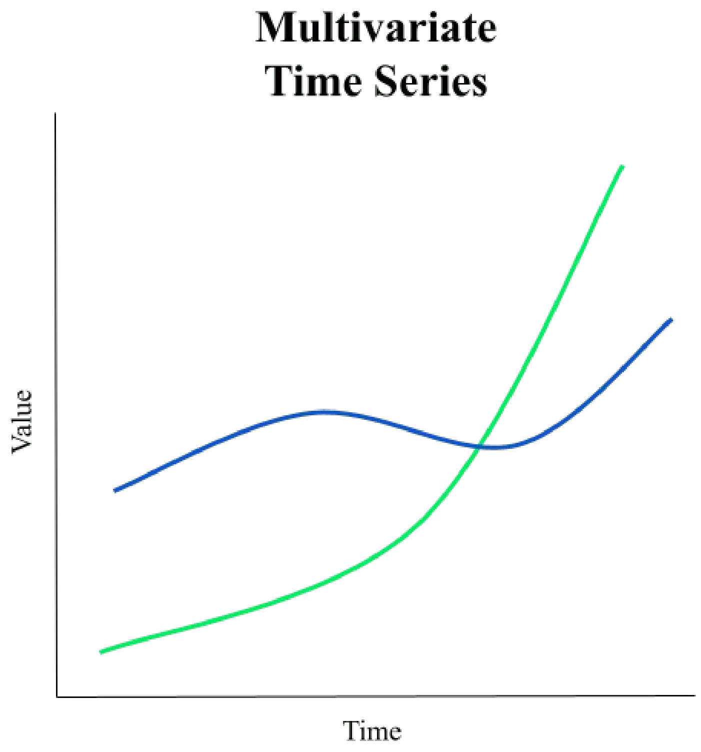
- Natural language processing, LLMs



Univariate vs Multivariate Sequences

- We may need to understand more than one sequence





Feature Creation

- We can use a window size (width, length, step) and horizon
- This method is called the “sliding window”

```
# Window for one week with the target of predicting the next day (Bitcoin prices)
[123.654, 125.455, 108.584, 118.674, 121.338, 120.655, 121.795] -> [123.033]
[125.455, 108.584, 118.674, 121.338, 120.655, 121.795, 123.033] -> [124.049]
[108.584, 118.674, 121.338, 120.655, 121.795, 123.033, 124.049] -> [125.961]
```

window size = 7, horizon = 1

Feature Creation

- Using a window size (input width) and horizon (label width)
- This method is called the sliding window

time,	measure
1,	100
2,	110
3,	108
4,	115
5,	120

LAG 1 feature (window size = 1, horizon = 1)

X,	y
?,	100
100,	110
110,	108
108,	115
115,	120
120,	?

Feature Creation

- Using a window size (input width) and horizon (label width)
- This method is called the sliding window

```
time,   measure
1,      100
2,      110
3,      108
4,      115
5,      120
```

LAG 1 feature (window size = 1, horizon = 2)

```
X1,   y1,   y2
?     100,  110
100,  110,  108
110,  108,  115
108,  115,  120
115,  120,  ?
120,  ?,    ?
```

Python

Creating Lags on Multiple Features

- Using the sliding window method on multiple predictors can help
- The windows used can be different

time,	measure
1,	100
2,	110
3,	108
4,	115
5,	120

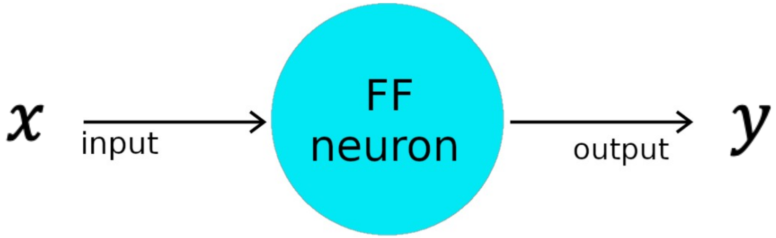
LAG 1 feature (window size = 1, horizon = 2)

X1,	y1,	y2
?	100,	110
100,	110,	108
110,	108,	115
108,	115,	120
115,	120,	?
120,	?,	?

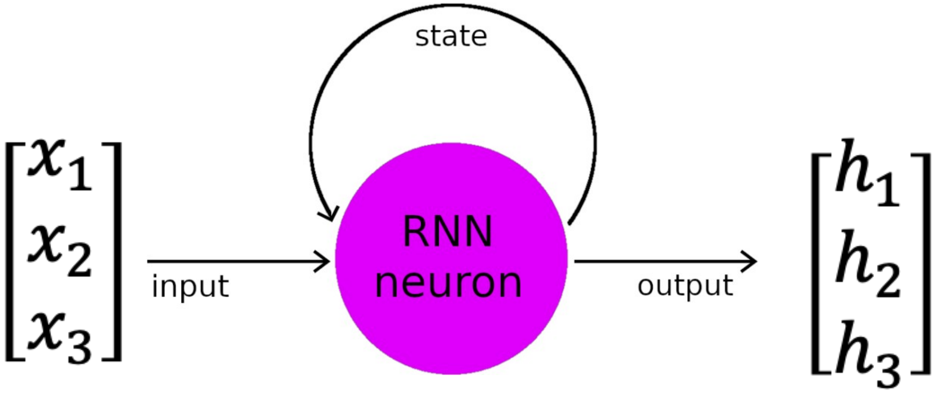
Python

Recurrent Neural Networks

- RNN neurons contain a feedback loop to receive sequence inputs
- The neurons (units) can be rolled to account for a sequence.



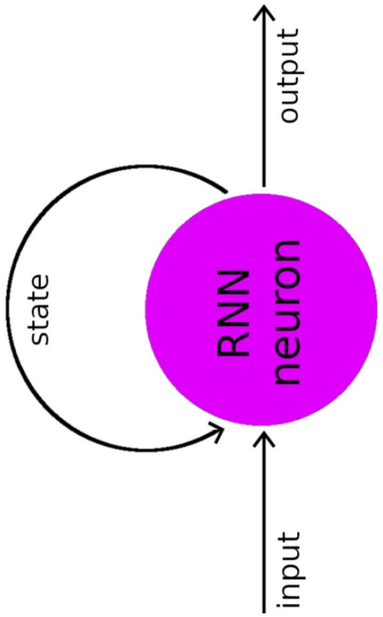
$$y = f(w_{ih}x + b)$$



$$h_t = f(w_{ih}x + w_{hh}h_{t-1} + b)$$

Recurrent Neural Networks

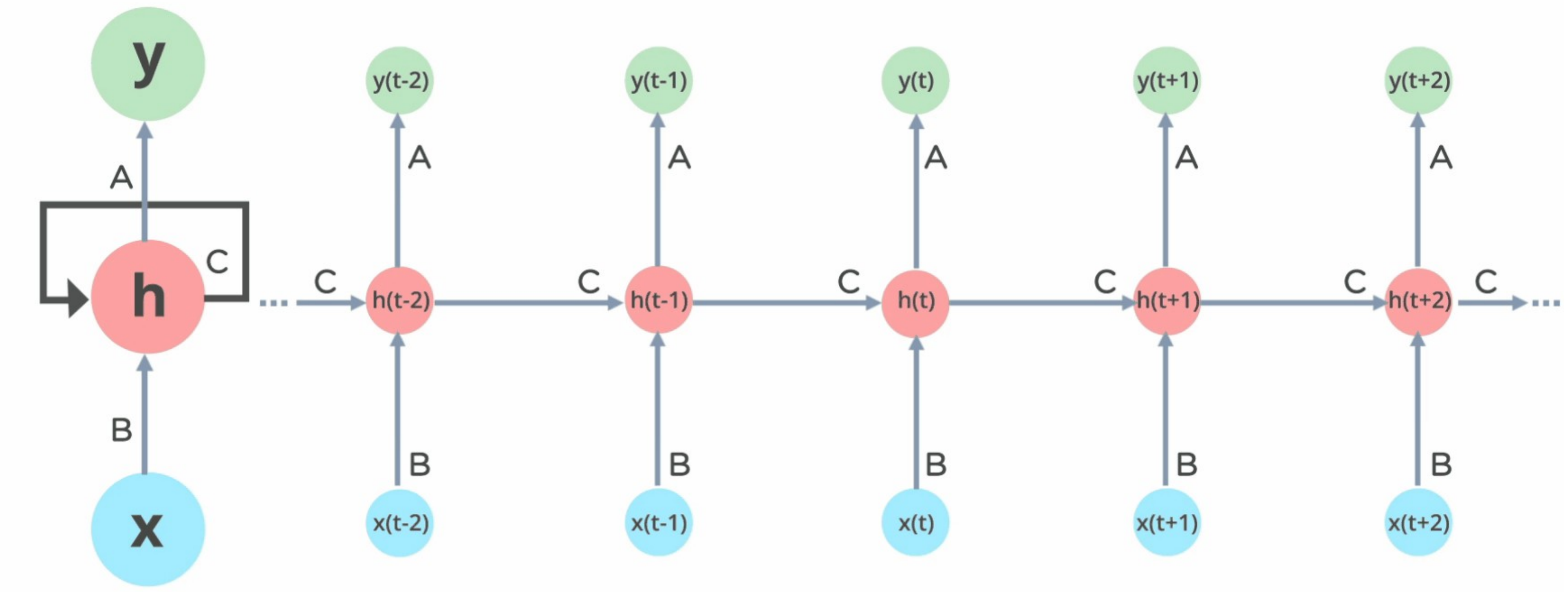
- ❑ RNN neurons contain a feedback loop to receive sequence inputs
- ❑ The neurons (units) can be rolled to account for a sequence.



$$h_t = f(w_{ih}x + w_{hh}h_{t-1} + b)$$

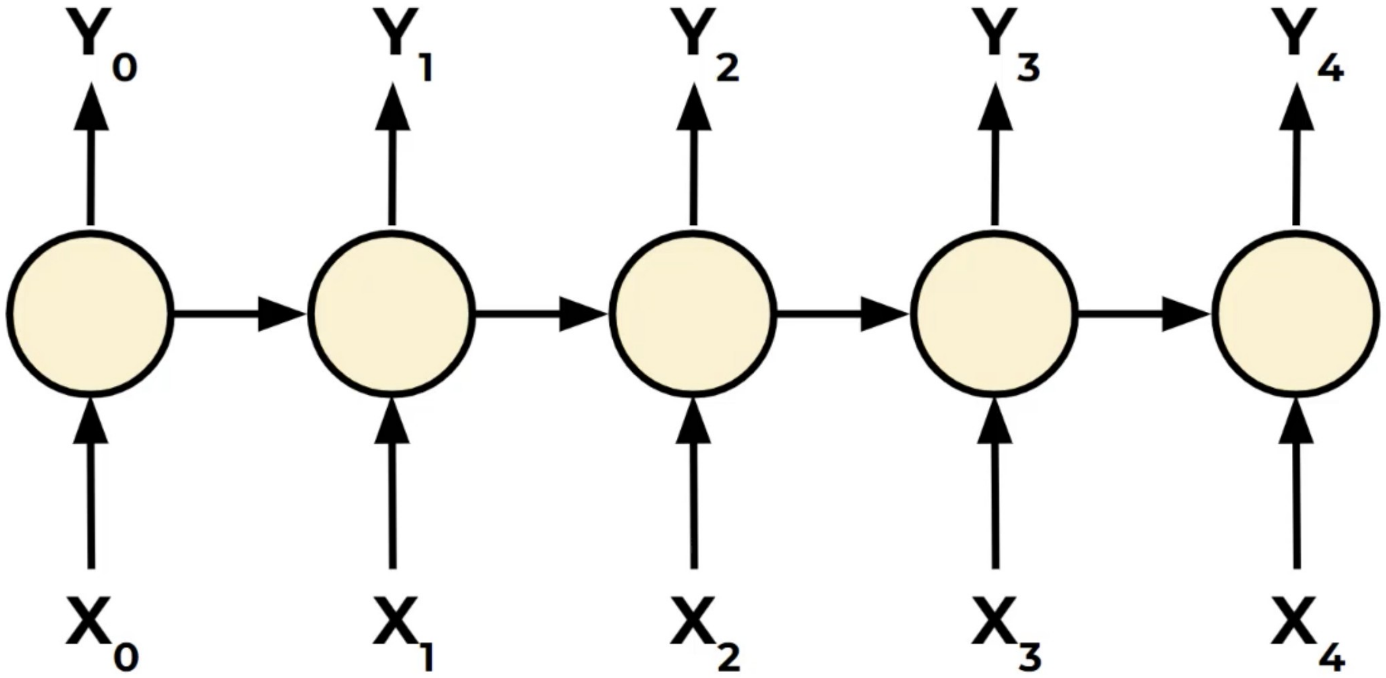
Recurrent Neural Networks

- RNN neurons contain a feedback loop to receive sequence inputs
- The neurons (units) can be rolled to account for a sequence.



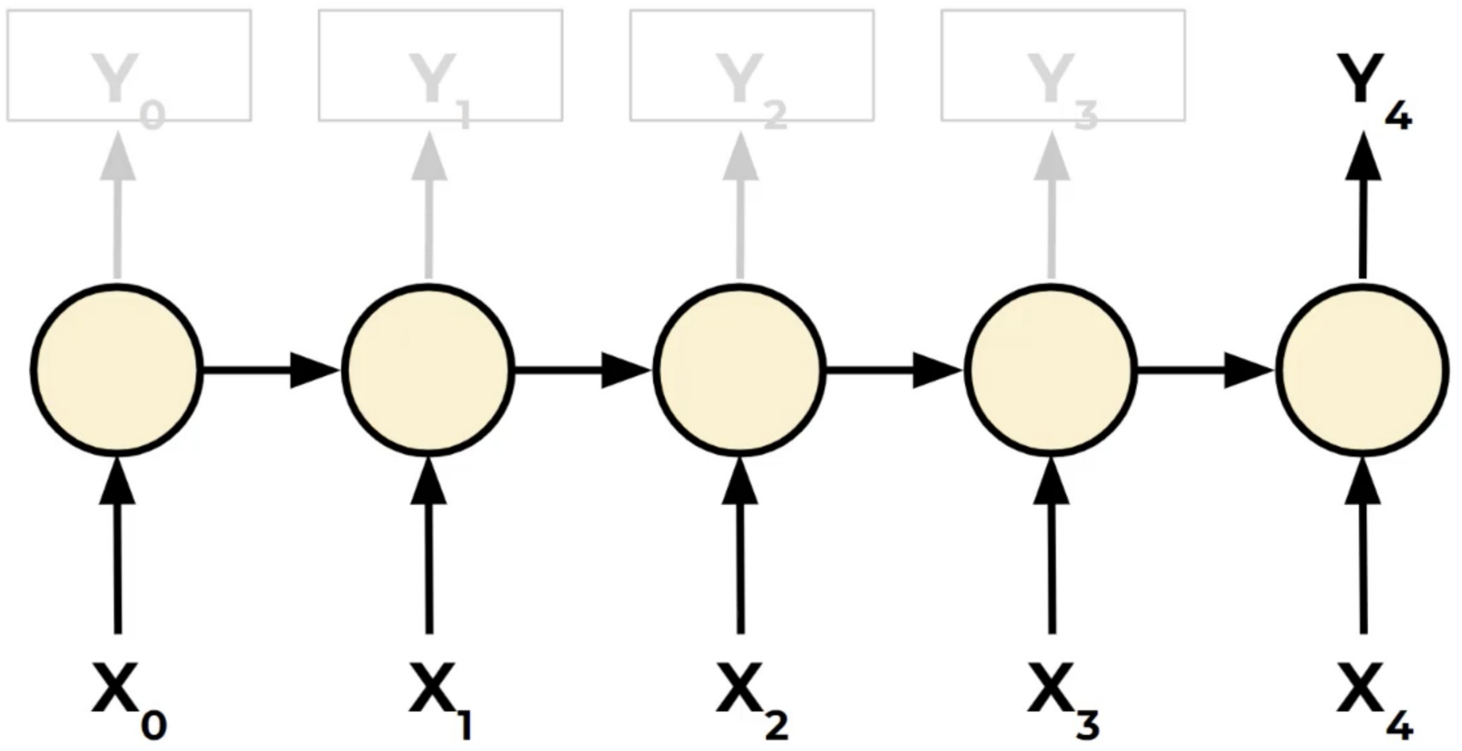
Recurrent Neural Networks

- Sequence to Sequence (seq2seq)



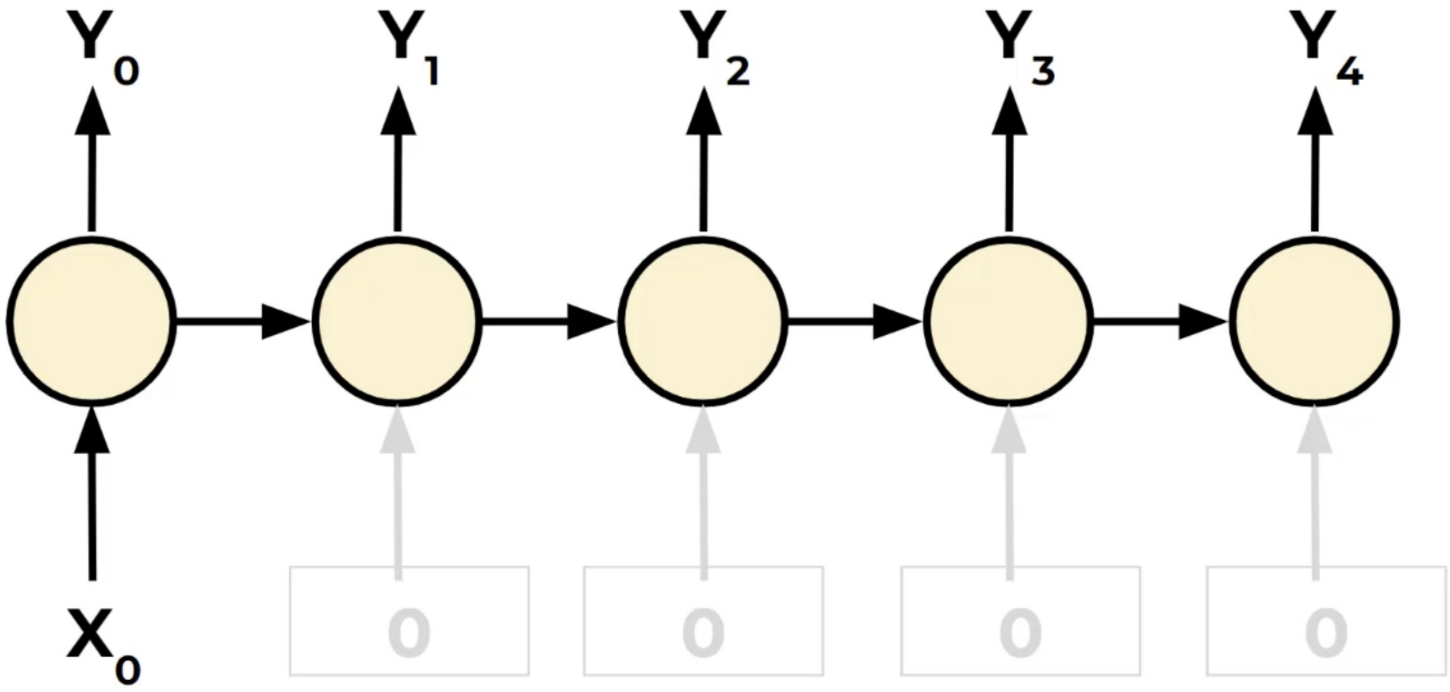
Recurrent Neural Networks

- Sequence to Vector (seq2vec)



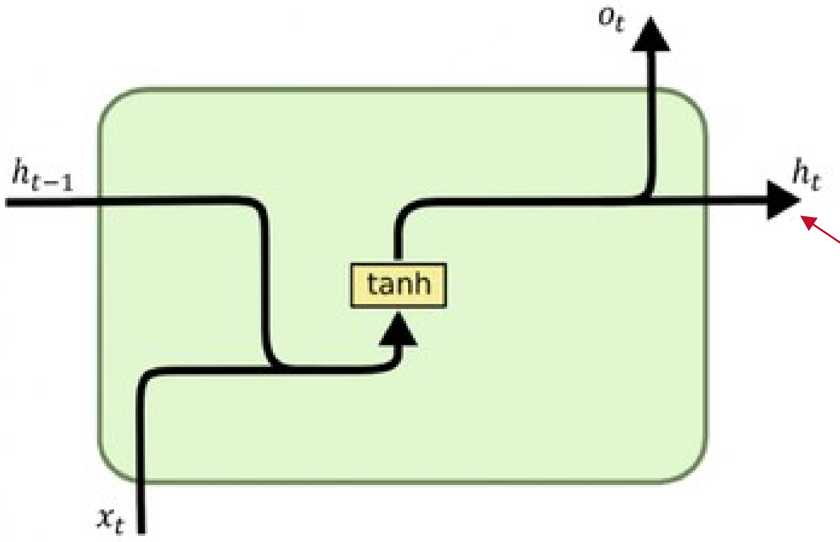
Recurrent Neural Networks

- Vector to Sequence (vec2seq)



Recurrent Neural Networks

- Elman network
- Tanh activation

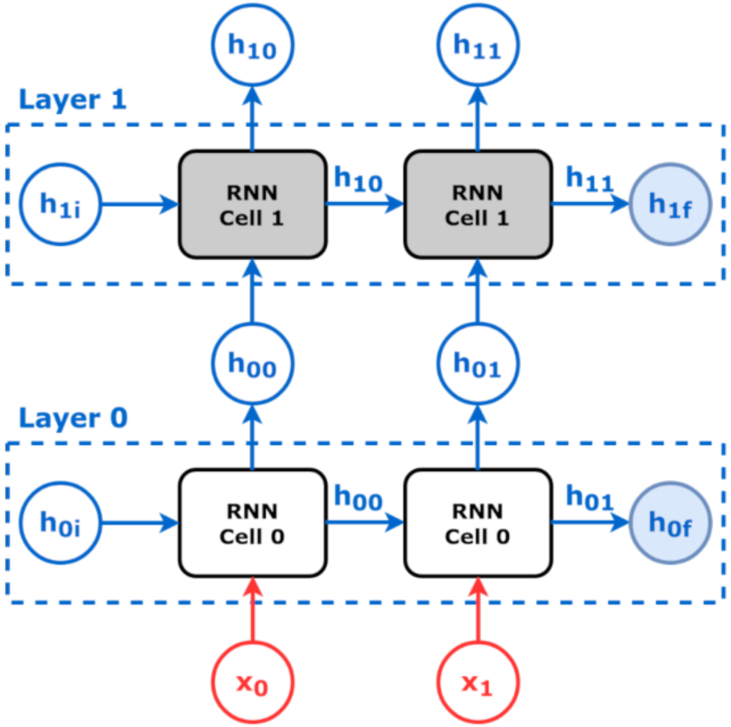


(a) Simple RNN

HIDDEN STATE

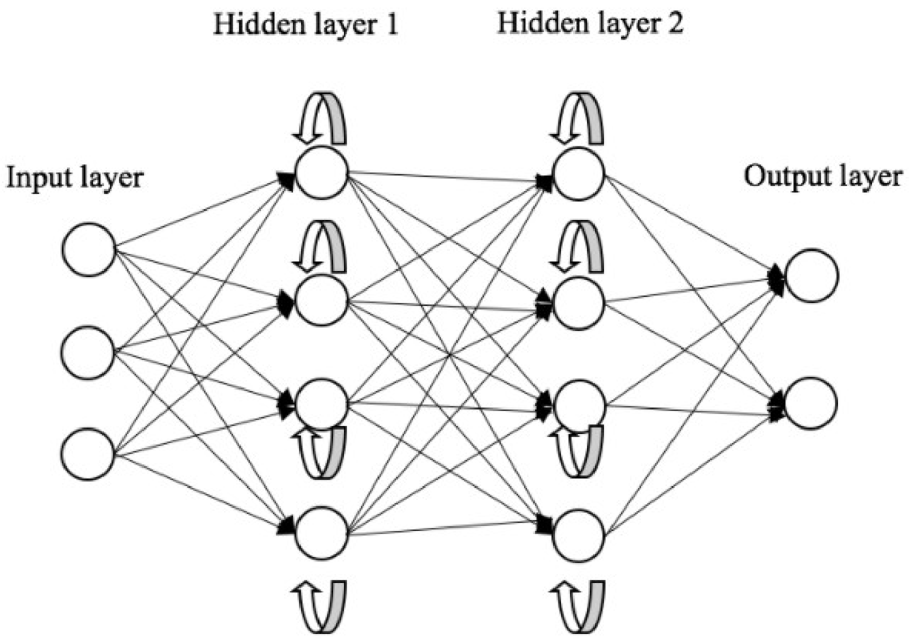
Stacked RNN Architecture

- Stacked (deep) RNNs are composed of multiple RNNs stacked one above the other.



RNN Architecture

- ❑ We can construct layers of RNN neurons (units)
- ❑ We can also stack layers (need to return the sequences)

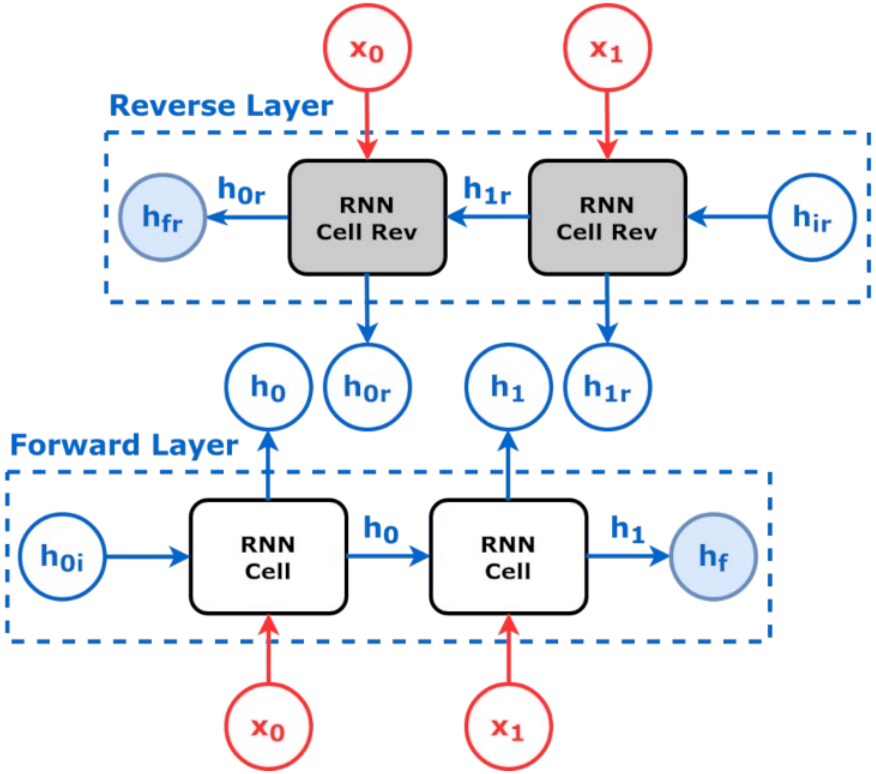


General Form of RNNs

Python

Bi-directional RNN Architecture

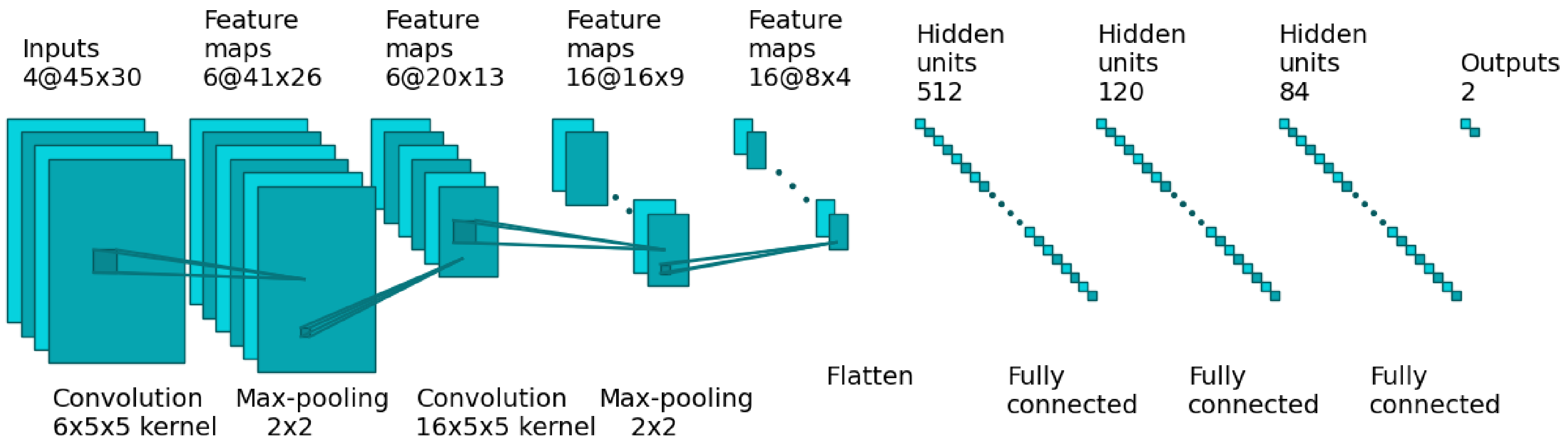
- Composed of two RNNs
- The input sequence (RNN # 1) and opposite direction (RNN # 2)



Python

CNN for Sequences

- CNNs excel at capturing **local dependencies** (e.g., trends, peaks).
- Effective in **repeating patterns** regardless of position in sequence.
- Fewer parameters than RNNs; allows **parallel computation**.



TimeSeriesGenerator

- Converts raw time series into supervised learning **format**
- Efficient for memory use and model training
- Supports **sliding window** logic

python

Copy

```python
TimeseriesGenerator(
    data,                 # Numpy array of time series
    targets,              # Numpy array of target values
    length=10,            # Number of time steps per input sample
    sampling_rate=1,      # Period between samples
    stride=1,             # Step between successive windows
    batch_size=32,        # Number of samples per batch
    shuffle=False         # Whether to shuffle samples
)
```

TimeSeriesGenerator

- Converts raw time series into supervised learning **format**
- Efficient for memory use and model training
- Supports **sliding window** logic

Time	Stock	Feature 1	Feature 2	Feature ...	Feature N			
T+0	1	[some value]	[some value]	[some value]	[some value]			
T+1	1	[some value]	[some value]	[some value]	[some value]			
T+2	1	[some value]	[some value]	[some value]	[some value]			
T+3	1	[some value]	[some value]	[some value]	[some value]			
T+4	1	[some value]	[some value]	[some value]	[some value]	Sequence 1		
T+5	1	[some value]	[some value]	[some value]	[some value]		Sequence 2	
T+0	2	[some value]	[some value]	[some value]	[some value]			Sequence 3
T+1	2	[some value]	[some value]	[some value]	[some value]			
T+2	2	[some value]	[some value]	[some value]	[some value]			
T+3	2	[some value]	[some value]	[some value]	[some value]			
T+4	2	[some value]	[some value]	[some value]	[some value]			
T+5	2	[some value]	[some value]	[some value]	[some value]			
T+0	3	[some value]	[some value]	[some value]	[some value]			
T+1	3	[some value]	[some value]	[some value]	[some value]			
T+2	3	[some value]	[some value]	[some value]	[some value]			

Review: Convolutions

- Suppose we have a sequence $I_{n\times p}$ and a kernel $K_{k\times l}$
- The resulting image $O_{n-k+1\ \times\ p-l+1}$

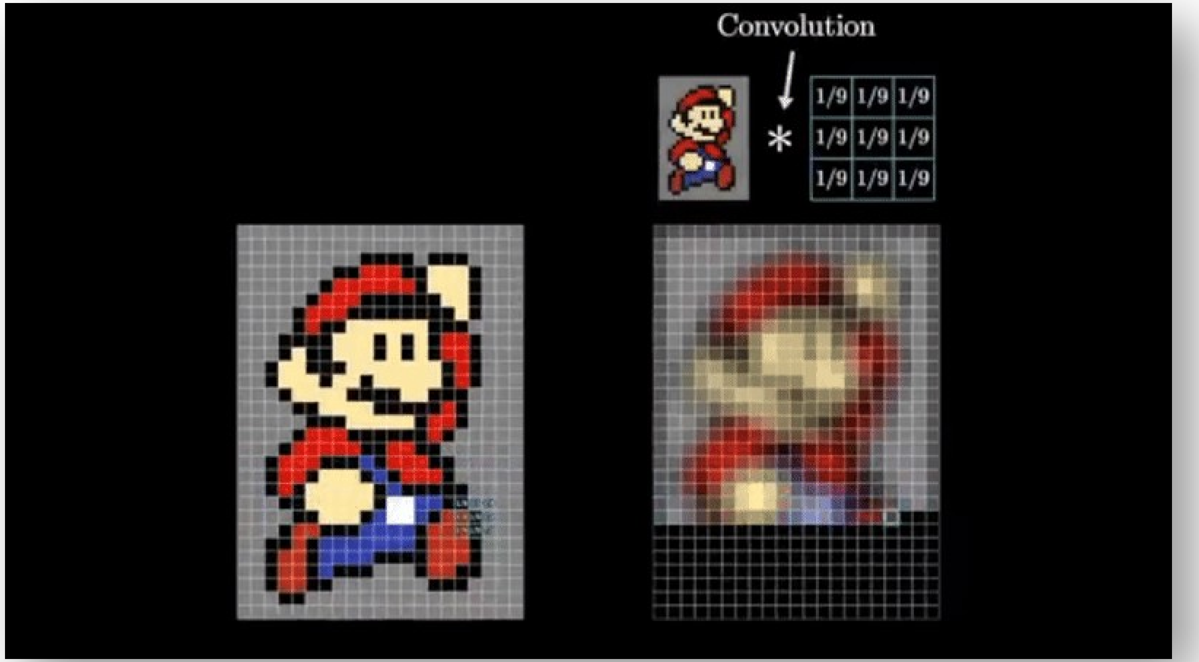
$$
\begin{bmatrix} 7 & 2 & 3 & 3 & 8 \\ 4 & 5 & 3 & 8 & 4 \\ 3 & 3 & 2 & 8 & 4 \\ 2 & 8 & 7 & 2 & 7 \\ 5 & 4 & 4 & 5 & 4 \end{bmatrix}
*
\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}
=
\begin{bmatrix} 6 & & \\ & & \\ & & \end{bmatrix}
$$

7x1+4x1+3x1+
2x0+5x0+3x0+
3x-1+3x-1+2x-1
= 6

$I_{5\times 5}$ $K_{3\times}$ $O_{5-3+1\ \times 5-3+1} = O_{3\times 3}$

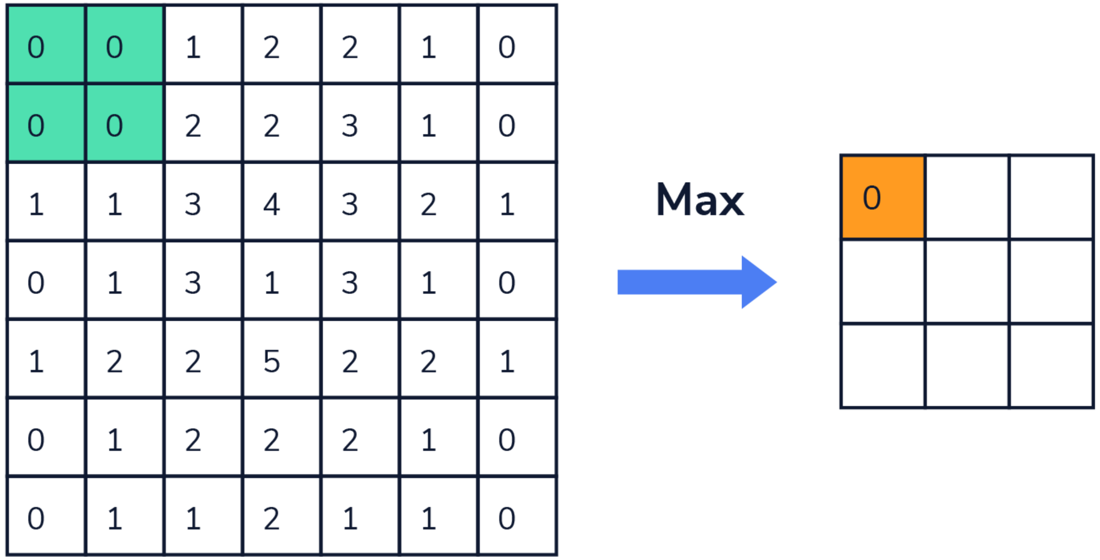
Review: Convolutions

- Suppose we have a sequence $I_{n \times p}$ and a kernel $K_{k \times l}$
- The resulting image $O_{n-k+1 \quad \times \quad p-l+1}$



Review: Pooling

- Pooling is also used to downsample the sequences.
- Pooling filters keep the important parts of the sequence.
- Max, Min and Average Pooling Filters are the most common



Python

Limitations of RNNs

- Struggle with long-term dependencies due to vanishing gradients.
  - Attention (weights or scores for each hidden state)
- Training can be slow and unstable.
- Can be replaced by more advanced RNN-type architectures such as LSTMs and GRUs

Limitations of CNNs on Sequences

- Struggle to capture **long-range temporal dependencies**
- CNNs are **stateless** and don't carry information across time steps
  - Dilations (stride [t, t+2, t+4]) can help
- Unlike RNNs/LSTMs, they **don't "remember" past context**