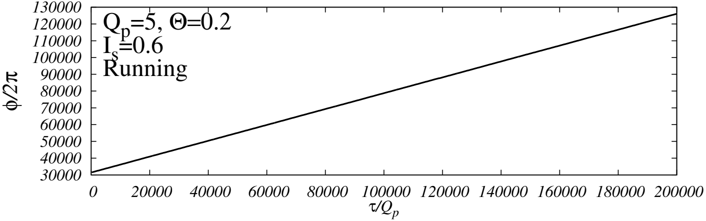
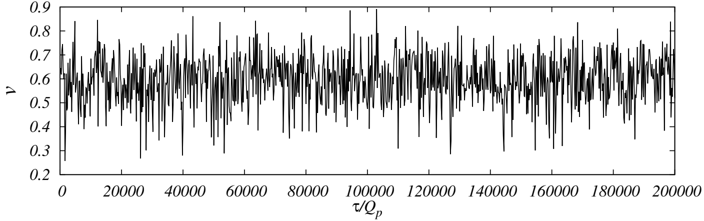
Machine Learning with Time Series
Forecasting, Recurrent neural networks

Martin Žonda and Pavel Baláž

November 2022

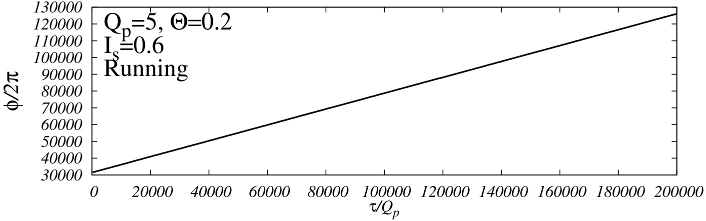
The world is changing in time

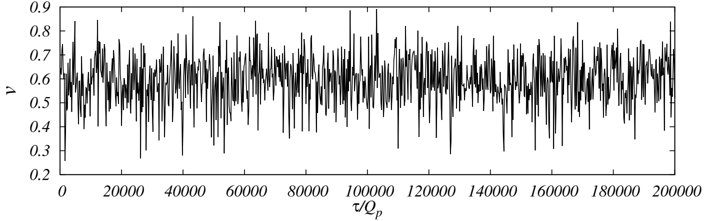




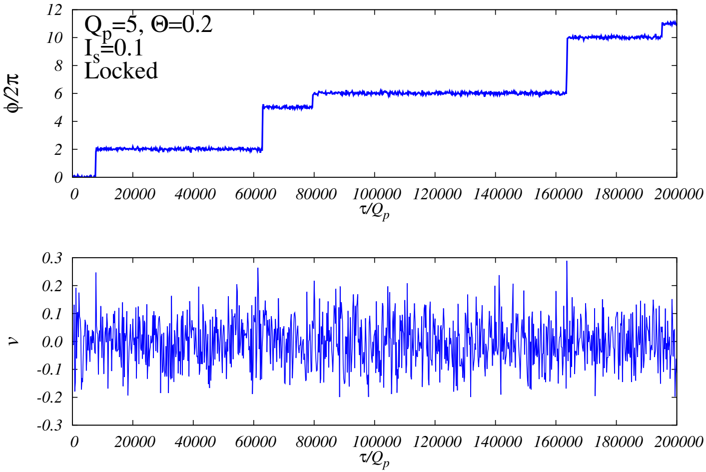
- Time series are ubiquitous in science and industry
- Time dependent data aren't statistically independent
- Forecasting is important, but it is not the only interesting task for ML

The world is changing in time

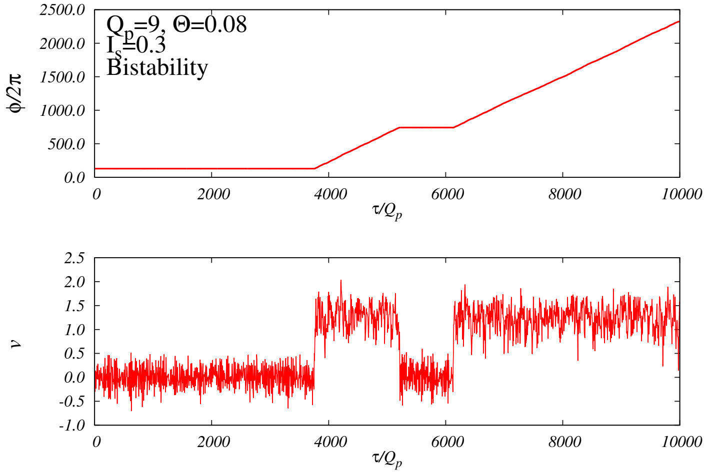




The world is changing in time



The world is changing in time



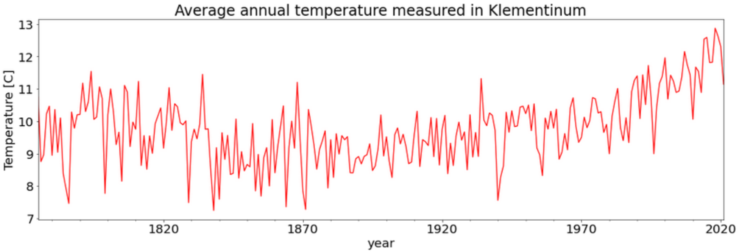
Outline

- **About Data**
- **Common predictor based techniques**
- **Recurrent Neurons and Layers**
  - Memory cells
  - Input and output
- **Simple RNN**
- **Deep RNN**

Time dependent data

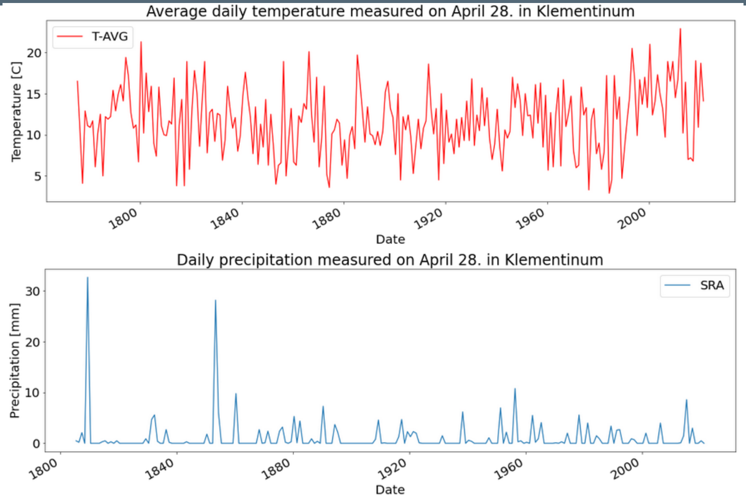
Data types

- Single series, single variable



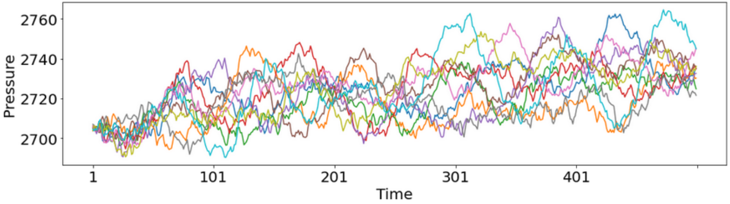
Data types

- Single series, single variable
- Single series, multivariate



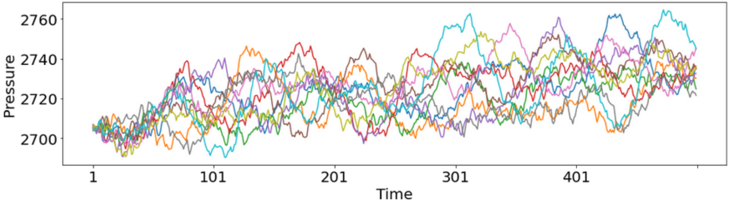
Data types

- Single series, single variable
- Single series, multivariate
- Panel data



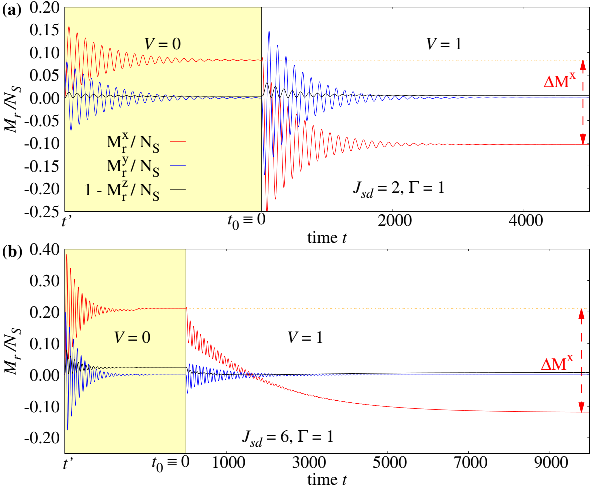
Data types

- Single series, single variable
- Single series, multivariate
- Panel data
    - Panel data may have misaligned time points



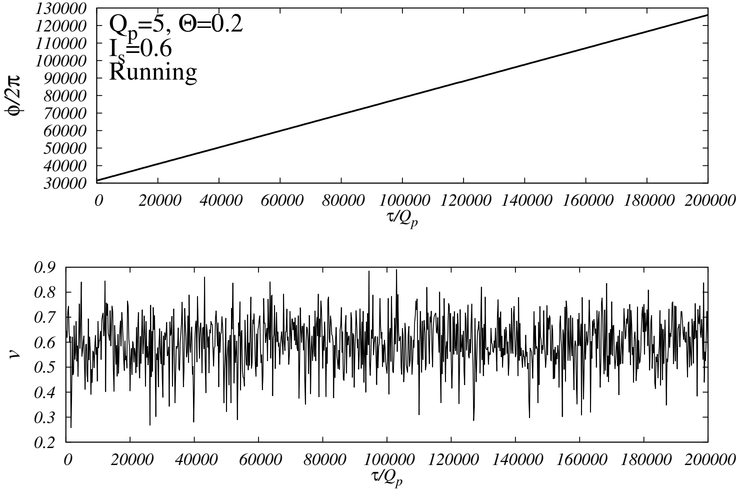
Time series characteristics

- Trend analysis



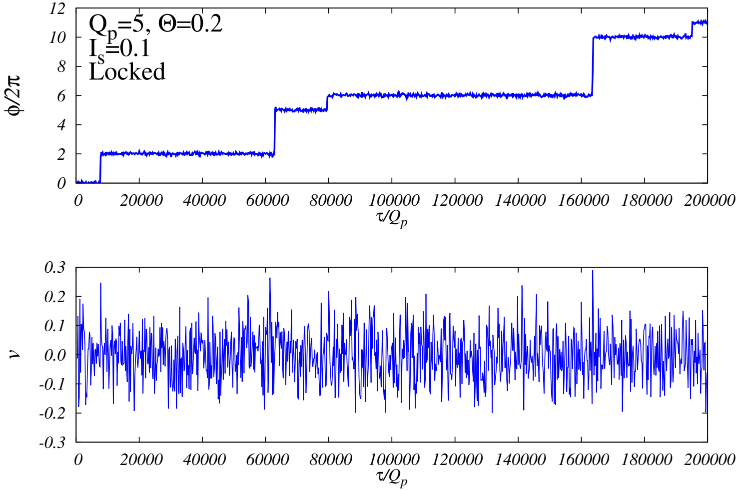
Time series characteristics

- Trend analysis



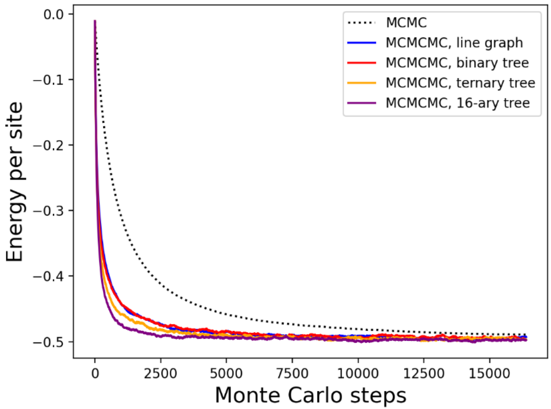
Time series characteristics

- Trend analysis
- Outliers



Time series characteristics

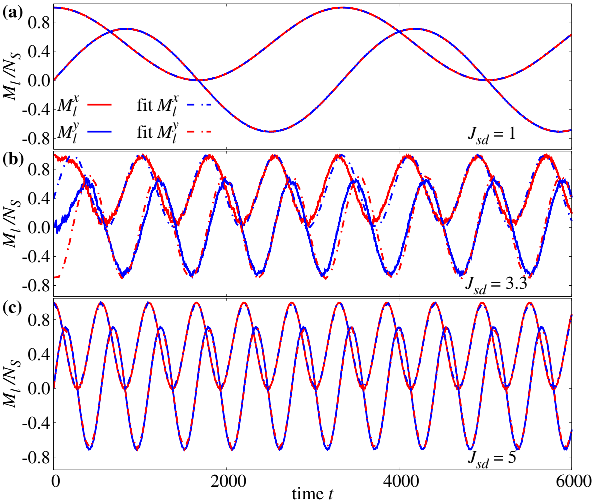
- Trend analysis
- Outliers
- Stationarity



arXiv:2102.05613

Time series characteristics

- Trend analysis
- Outliers
- Stationarity
- Periodicity



Forecasting

Using standard ML techniques

Strategy for forecasting with common ML techniques:

d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	$\dots$	d_N	?	?	?

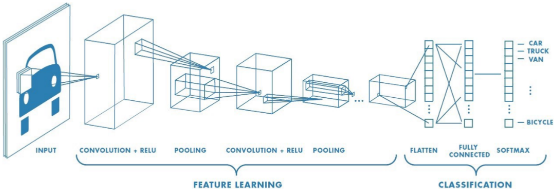
We can make our own training data:

d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	$\dots$	d_N	?	?	?
d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	$\dots$	d_N	?	?	?
d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	$\dots$	d_N	?	?	?
							$\vdots$							
d_1	d_2	d_3	d_4	d_5	d_6	d_7	$\dots$	d_{N-3}	d_{N-2}	d_{N-1}	d_N	?	?	?

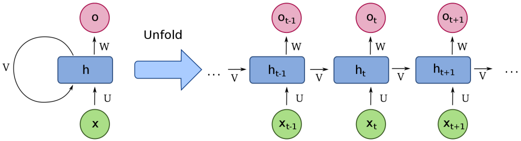
Training data:

d_1	d_2	d_3	d_4	d_5
d_2	d_3	d_4	d_5	d_6
d_3	d_4	d_5	d_6	d_7
		$\vdots$		

RNN and CNN



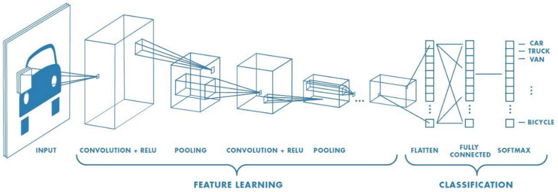
source: towardsdatascience.com



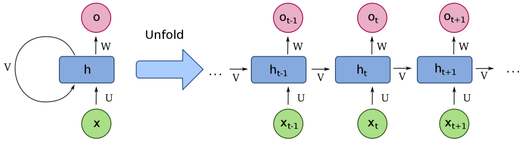
source: wikipedia.com

- CNN and RNN are both part of Deep Learning

RNN and CNN



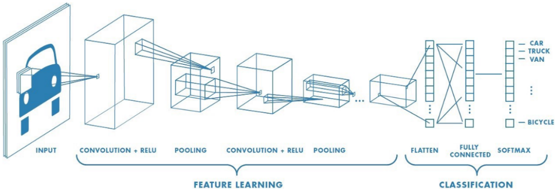
source: towardsdatascience.com



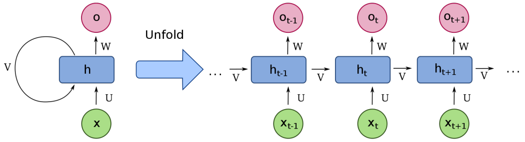
source: wikipedia.com

- CNN and RNN are both part of Deep Learning
- CNN and RNN follow different architectures

RNN and CNN



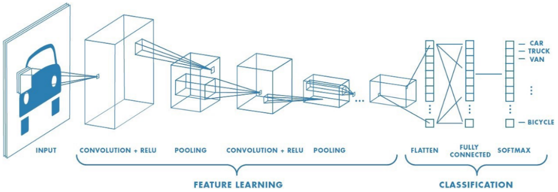
source: towardsdatascience.com



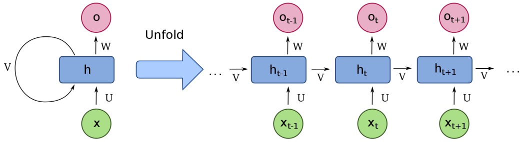
source: wikipedia.com

- CNN and RNN are both part of Deep Learning
- CNN and RNN follow different architectures
- RNN are specifically build for forecasting

RNN and CNN



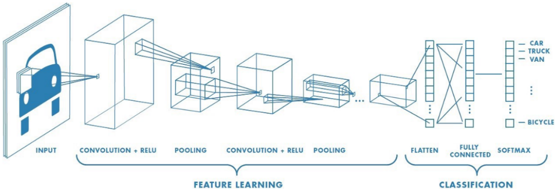
source: towardsdatascience.com



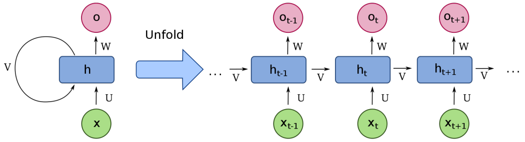
source: wikipedia.com

- CNN and RNN are both part of Deep Learning
- CNN and RNN follow different architectures
- RNN are specifically build for forecasting
- Yet, there are CNN capable to deal with large time series

RNN and CNN



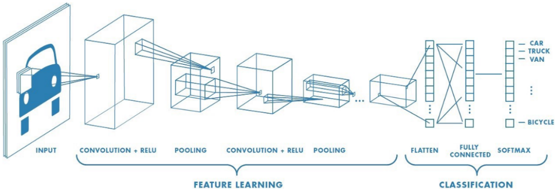
source: towardsdatascience.com



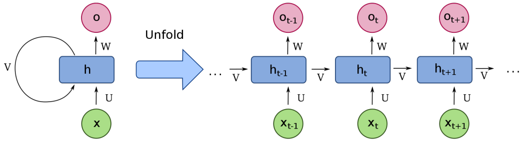
source: wikipedia.com

- CNN and RNN are both part of Deep Learning
- CNN and RNN follow different architectures
- RNN are specifically build for forecasting
- Yet, there are CNN capable to deal with large time series
- e.g: WaveNet

RNN and CNN



source: towardsdatascience.com



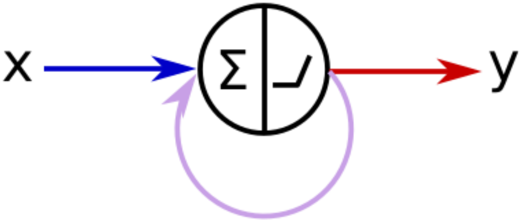
source: wikipedia.com

- CNN and RNN are both part of Deep Learning
- CNN and RNN follow different architectures
- RNN are specifically build for forecasting
- Yet, there are CNN capable to deal with large time series
- e.g: WaveNet
- Nevertheless, today we will focus on RNN

Recurrent Neurons and Layers

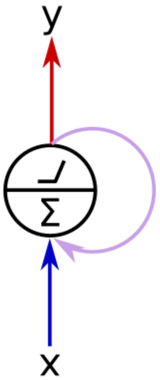
Recurrent Neuron

- RNN looks like a standard feedforward NN



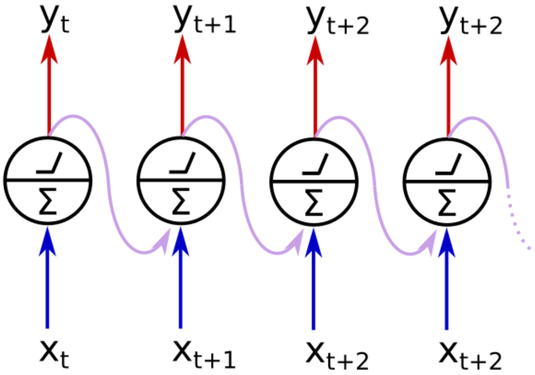
Recurrent Neuron

- RNN looks like a standard feedforward NN
- Except it has connections from output to input



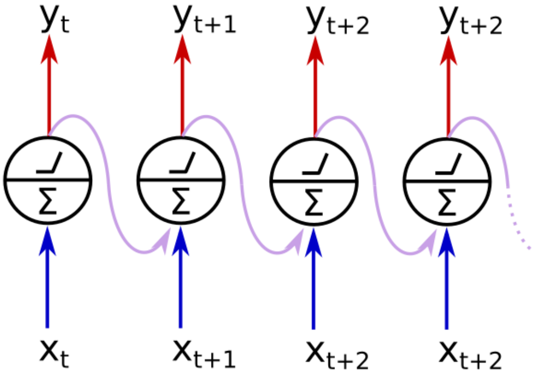
Recurrent Neuron

- RNN looks like a standard feedforward NN
- Except it has connections from output to input
- Neuron in time $t + 1$ receives input x_{t+1} and output of y_t



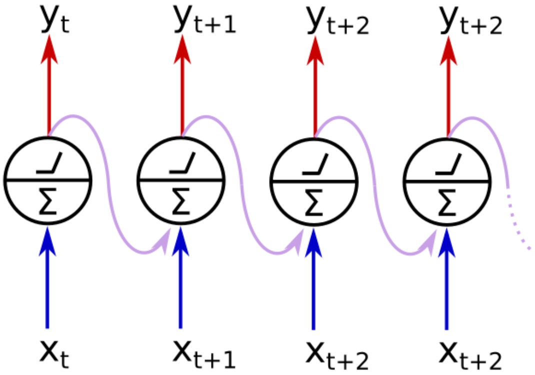
Recurrent Neuron

- RNN looks like a standard feedforward NN
- Except it has connections from output to input
- Neuron in time $t + 1$ receives input x_{t+1} and output of y_t
- This is still the same (one) neuron plotted in different times



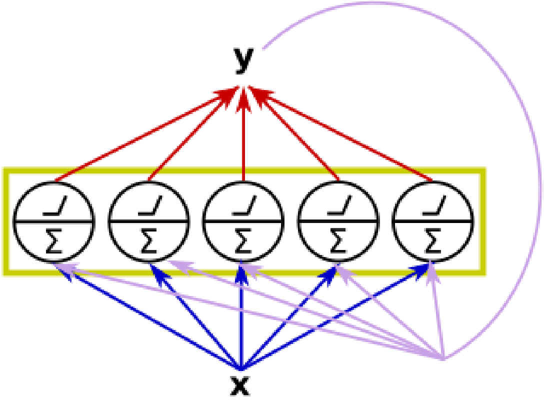
Recurrent Neuron

- RNN looks like a standard feedforward NN
- Except it has connections from output to input
- Neuron in time $t+1$ receives input x_{t+1} and output of y_t
- This is still the same (one) neuron plotted in different times
- We can build a layer



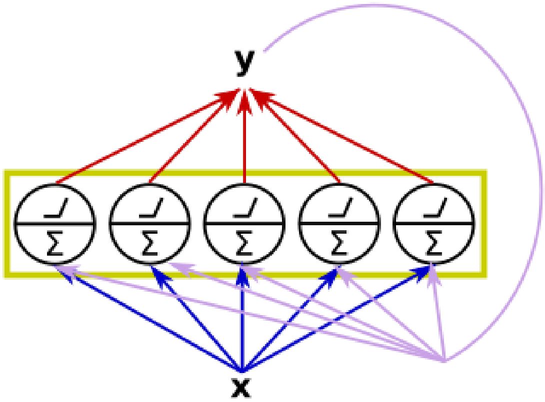
Recurrent Layer

- Each recurrent neuron has two sets of weights $\mathbf{w}_x$ and $\mathbf{w}_y$



- Each recurrent neuron has two sets of weights $\mathbf{w}_x$ and $\mathbf{w}_y$
- Output of RNN at time t:

$$\mathbf{y}_t = \Phi(\mathbf{W}_x \mathbf{x}_t^T + \mathbf{W}_y \mathbf{y}_{t-1}^T + \mathbf{b}) \quad (1)$$

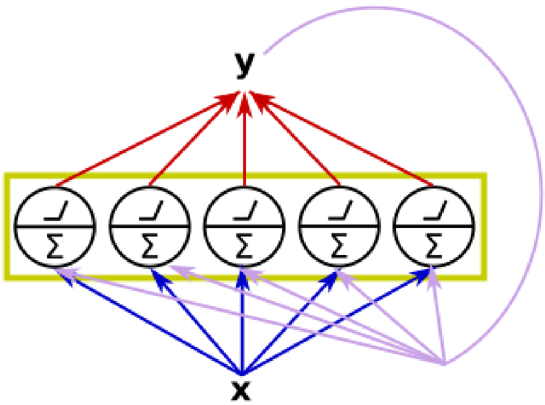


Recurrent Layer

- Each recurrent neuron has two sets of weights $\mathbf{w}_x$ and $\mathbf{w}_y$
- Output of RNN at time t:

$$\mathbf{y}_t = \Phi(\mathbf{W}_x \mathbf{x}_t^T + \mathbf{W}_y \mathbf{y}_{t-1}^T + \mathbf{b}) \quad (1)$$

- Here $\Phi()$ is the activation function

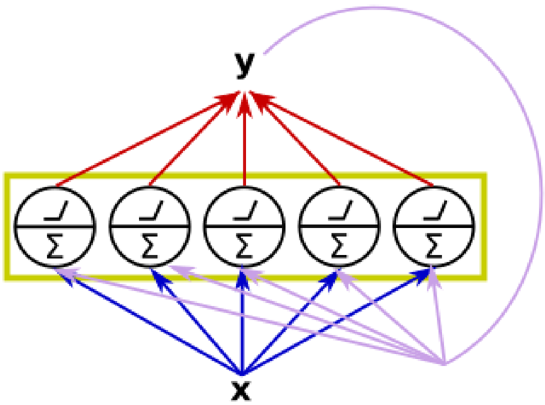


Recurrent Layer

- Each recurrent neuron has two sets of weights $\mathbf{w}_x$ and $\mathbf{w}_y$
- Output of RNN at time t:

$$\mathbf{y}_t = \Phi(\mathbf{W}_x \mathbf{x}_t^T + \mathbf{W}_y \mathbf{y}_{t-1}^T + \mathbf{b}) \quad (1)$$

- Here $\Phi()$ is the activation function
- $\mathbf{y}_t$ is a function of $\mathbf{x}_t$ and $\mathbf{y}_{t-1} \rightarrow$ recurrent behavior

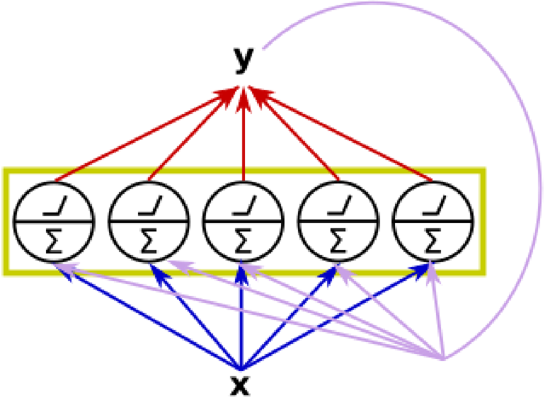


Recurrent Layer

- Each recurrent neuron has two sets of weights $\mathbf{w}_x$ and $\mathbf{w}_y$
- Output of RNN at time t:

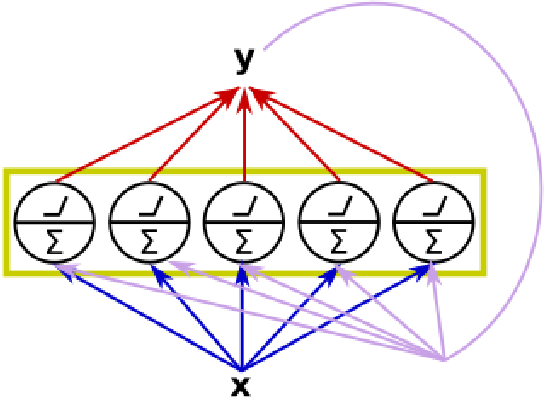
$$\mathbf{y}_t = \Phi(\mathbf{W}_x \mathbf{x}_t^T + \mathbf{W}_y \mathbf{y}_{t-1}^T + \mathbf{b}) \quad (1)$$

- Here $\Phi()$ is the activation function
- $\mathbf{y}_t$ is a function of $\mathbf{x}_t$ and $\mathbf{y}_{t-1} \rightarrow$ recurrent behavior
- At $t = 0$ we set $\mathbf{y}_{-1} = 0$



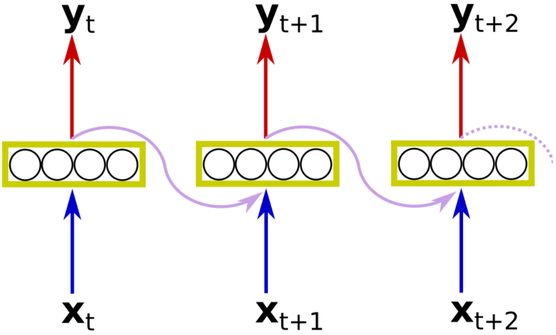
Memory cells

- It is a trivial example of a memory cell



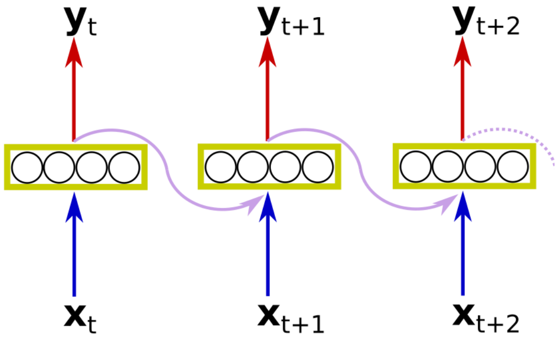
Memory cells

- It is a trivial example of a memory cell
- Output depends on the inputs from previous time steps



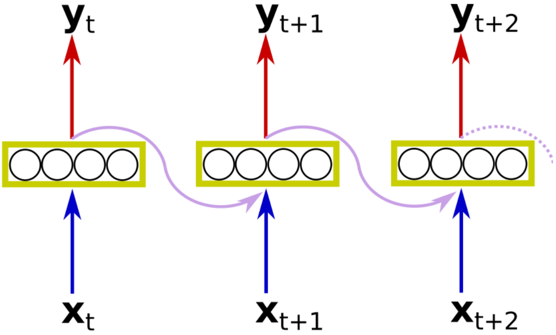
Memory cells

- It is a trivial example of a memory cell
- Output depends on the inputs from previous time steps
- Neuron computes a weighted sum of the inputs



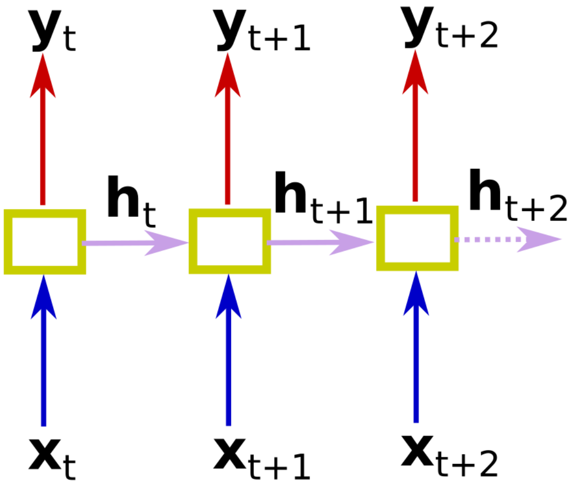
Memory cells

- It is a trivial example of a memory cell
- Output depends on the inputs from previous time steps
- Neuron computes a weighted sum of the inputs
- The memory is a bit short ($\approx$ 10 time steps)



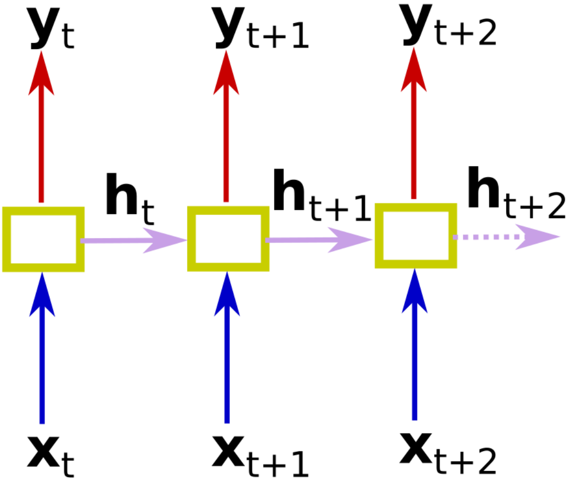
Memory cells

- It is a trivial example of a memory cell
- Output depends on the inputs from previous time steps
- Neuron computes a weighted sum of the inputs
- The memory is a bit short ($\approx$ 10 time steps)
- A more complex cells are needed for longer memories



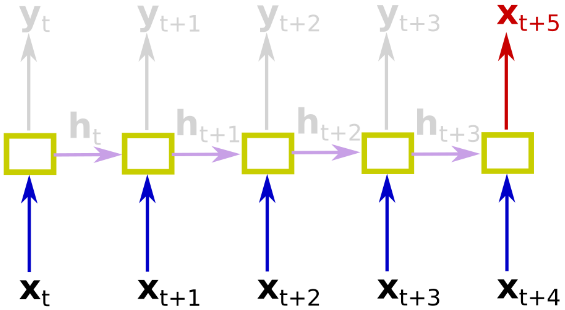
Memory cells

- It is a trivial example of a memory cell
- Output depends on the inputs from previous time steps
- Neuron computes a weighted sum of the inputs
- The memory is a bit short (≈ 10 time steps)
- A more complex cells are needed for longer memories
- Memory function can be also generalized $\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$



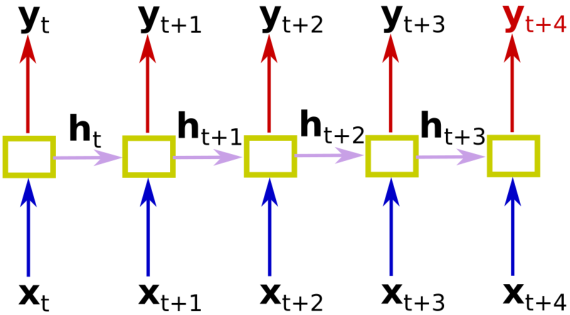
What is this good for?

- RNN are build for forecasting



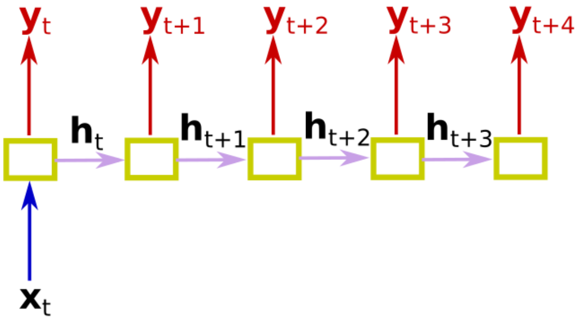
What is this good for?

- RNN are build for forecasting
- But they can do more (e.g., classification)



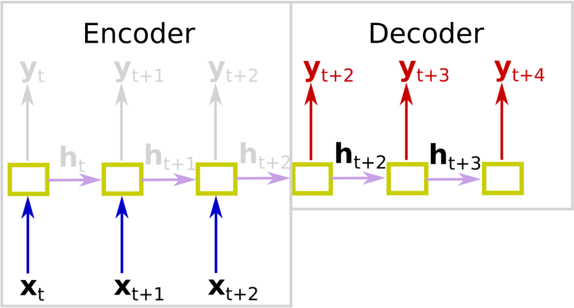
What is this good for?

- RNN are build for forecasting
- But they can do more (e.g., classification)
- Sequence generation (similar architecture as CNN)



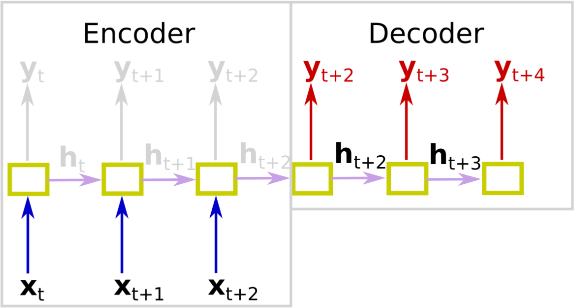
What is this good for?

- RNN are build for forecasting
- But they can do more (e.g., classification)
- Sequence generation (similar architecture as CNN)
- And then there are combinations, e.g., Encoder/Decoder



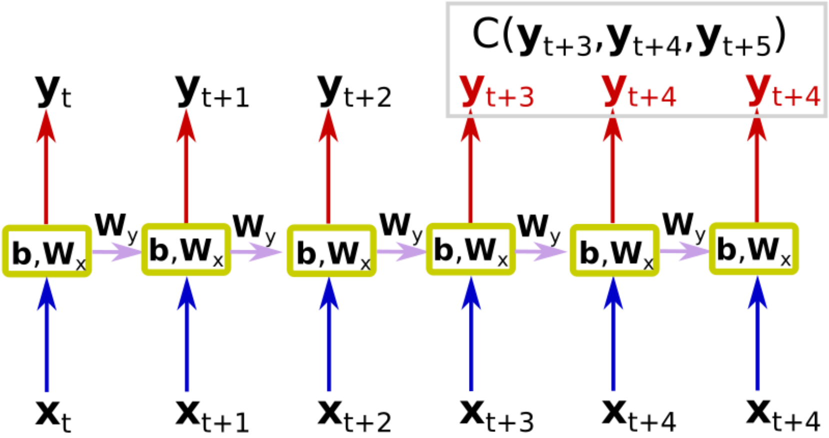
What is this good for?

- RNN are build for forecasting
- But they can do more (e.g., classification)
- Sequence generation (similar architecture as CNN)
- And then there are combinations, e.g., Encoder/Decoder
- Transformation of the data to different basis?



Training

We can use a regular back-propagation, however, it is rolled through time



More than one time-step

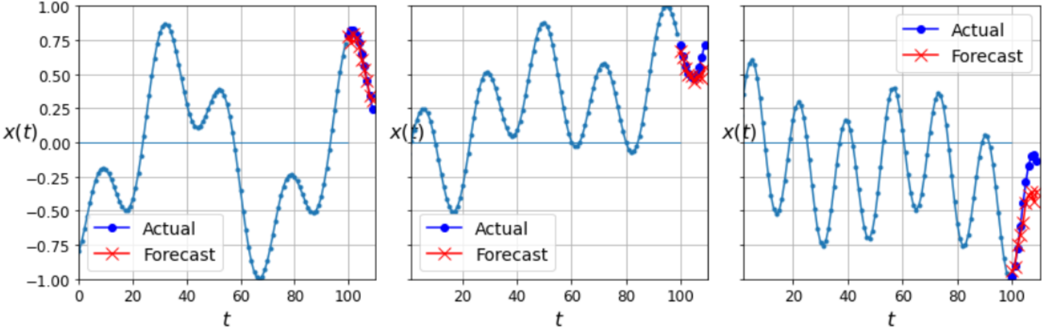
- We can simply add the prediction of the last point to the series and predict again
- It is not a good strategy
- Errors accumulate (Dense model is often superior to RNN)
- It is better to train RNN to predict all next values at once

More than one time-step

- We can simply add the prediction of the last point to the series and predict again
- It is not a good strategy
- Errors accumulate (Dense model is often superior to RNN)
- It is better to train RNN to predict all next values at once

```
model = keras.models.Sequential([
keras.layers.SimpleRNN(20,return_sequences=True,input_shape=[None,1]),
keras.layers.SimpleRNN(20),
keras.layers.Dense(10) ])
```

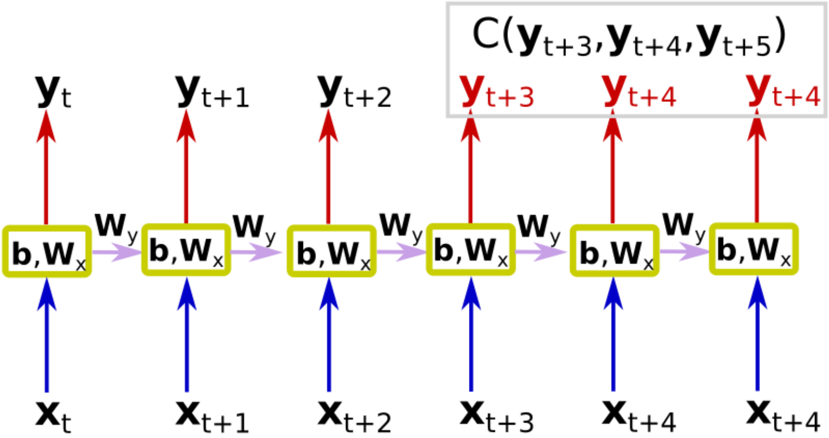
Deep RNN predicting 10 time steps at once:



MSE for the last point was 0.07.

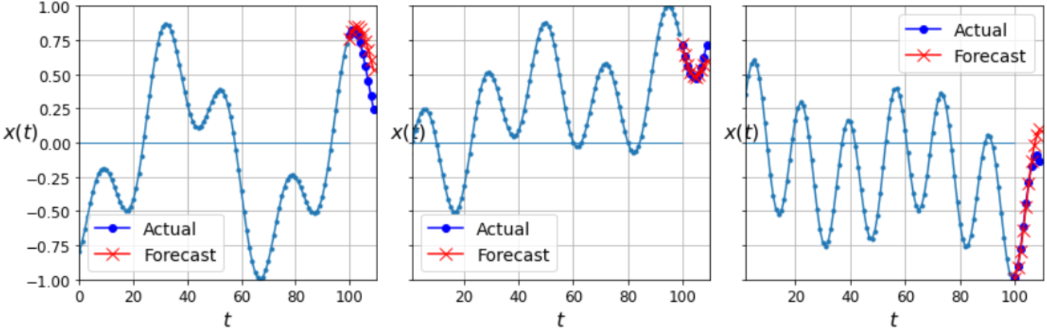
We can still do better

Instead of predicting N time steps at the last time, we can predict and evaluate (calculate the loss function) N time steps at each time step!



We can still do better

Deep RNN sequence to sequence model:

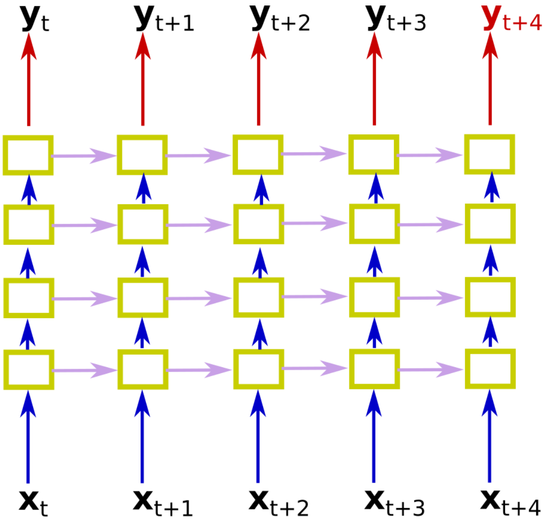


MSE for the last point was 0.015

Common problems and their solutions

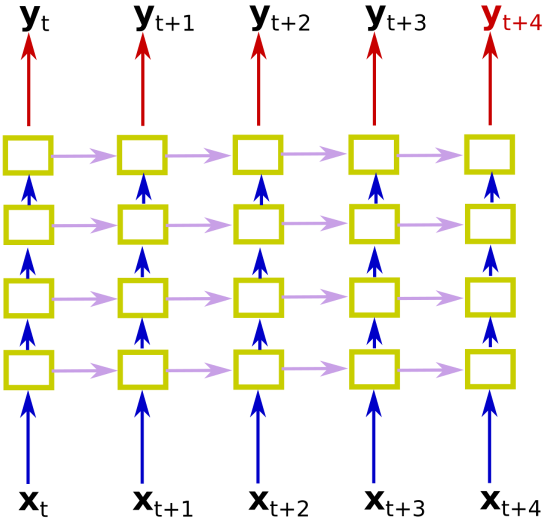
Long Sequences

- RNN for long sequence is basically a very deep network



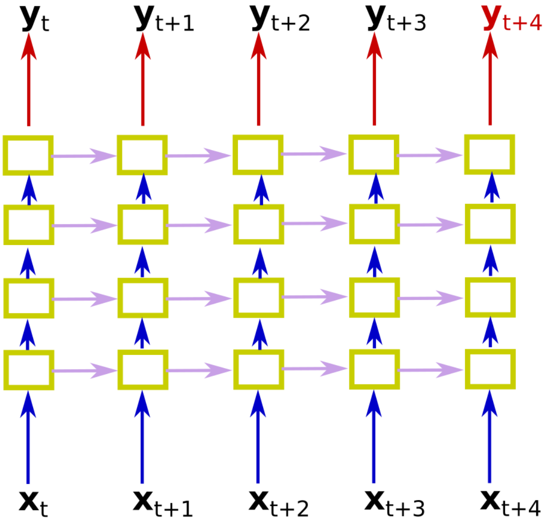
Long Sequences

- RNN for long sequence is basically a very deep network
- It suffers from the problem of **unstable gradient**



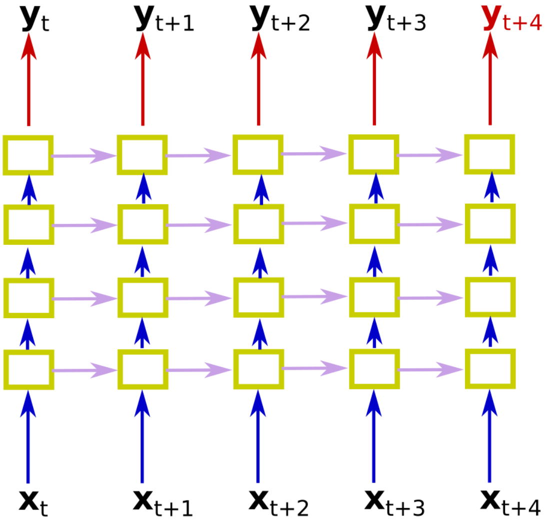
Long Sequences

- RNN for long sequence is basically a very deep network
- It suffers from the problem of **unstable gradient**
- RNN has typically **short memory**



Long Sequences

- RNN for long sequence is basically a very deep network
- It suffers from the problem of **unstable gradient**
- RNN has typically **short memory**
- It gradually forgets old inputs



Short-Term Memory Problem

- Various types of memory cells have been introduced to solve this problem

Short-Term Memory Problem

- Various types of memory cells have been introduced to solve this problem
- W.g., the Long Short-Term Memory (LSTM) cell

Short-Term Memory Problem

- Various types of memory cells have been introduced to solve this problem
- W.g., the Long Short-Term Memory (LSTM) cell
- The key idea is that it allows to learn what is important in a long-term run and store it

Short-Term Memory Problem

- Various types of memory cells have been introduced to solve this problem
- W.g., the Long Short-Term Memory (LSTM) cell
- The key idea is that it allows to learn what is important in a long-term run and store it
- The details are complicated (see A. Géron's Hands on ML), we will instead check simpler GRU cell

Short-Term Memory Problem

- Various types of memory cells have been introduced to solve this problem
- W.g., the Long Short-Term Memory (LSTM) cell
- The key idea is that it allows to learn what is important in a long-term run and store it
- The details are complicated (see A. Géron's Hands on ML), we will instead check simpler GRU cell
- but there is a layer for that:

Short-Term Memory Problem

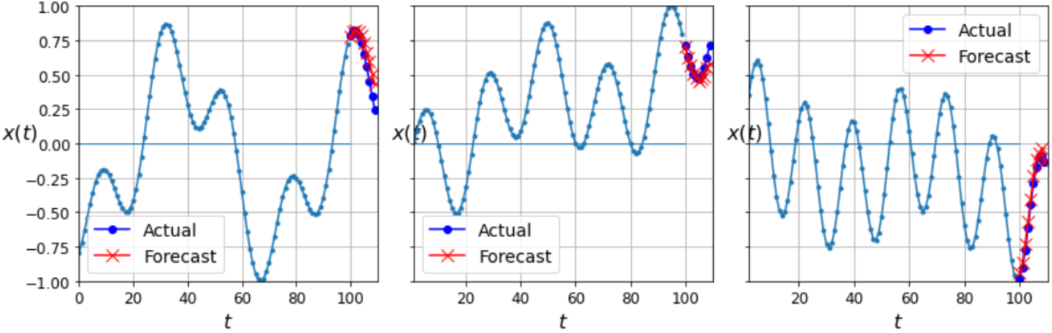
- Various types of memory cells have been introduced to solve this problem
- W.g., the Long Short-Term Memory (LSTM) cell
- The key idea is that it allows to learn what is important in a long-term run and store it
- The details are complicated (see A. Géron's Hands on ML), we will instead check simpler GRU cell
- but there is a layer for that:

Short-Term Memory Problem

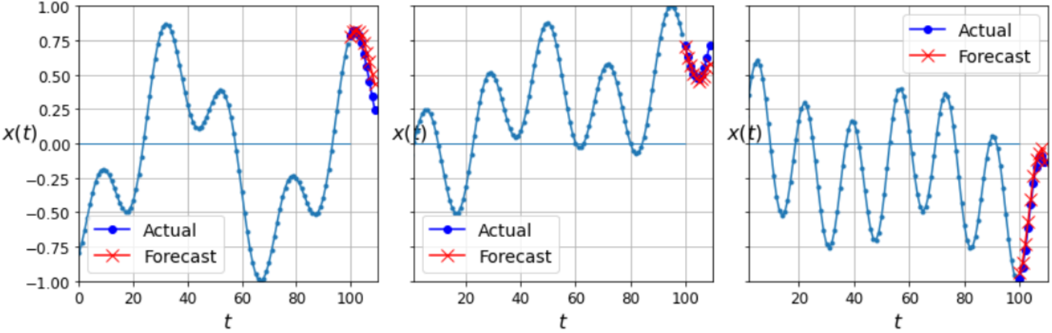
- Various types of memory cells have been introduced to solve this problem
- W.g., the Long Short-Term Memory (LSTM) cell
- The key idea is that it allows to learn what is important in a long-term run and store it
- The details are complicated (see A. Géron's Hands on ML), we will instead check simpler GRU cell
- but there is a layer for that:

```
model = keras.models.Sequential([ keras.layers.LSTM(20,
return_sequences=True, input_shape=[None, 1]),
keras.layers.LSTM(20, return_sequences=True),
keras.layers.TimeDistributed(keras.layers.Dense(10)) ])
```

RNN with LSTM cells

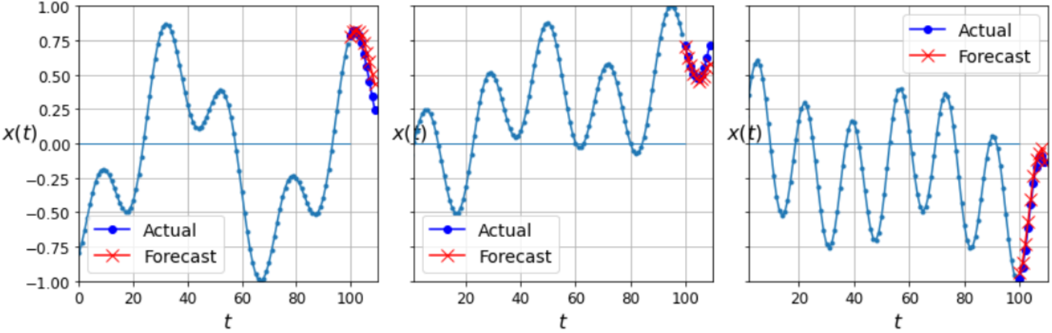


RNN with LSTM cells



MSE for the last point was 0.01. We are getting better.

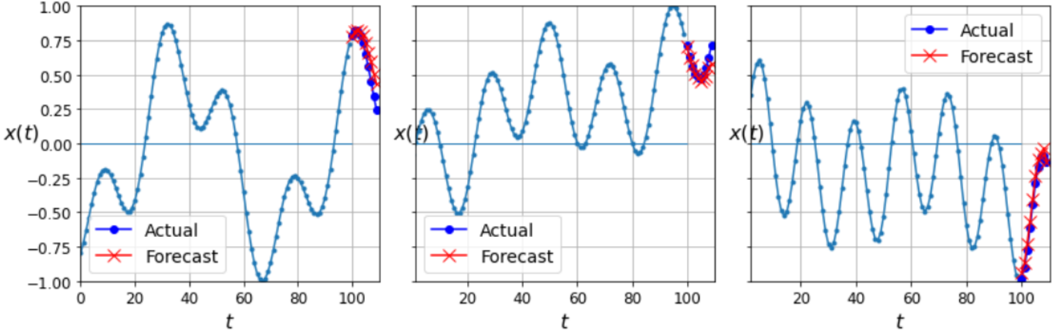
RNN with LSTM cells



MSE for the last point was 0.01. We are getting better.

But the memory is still limited to 100 time steps.

RNN with LSTM cells



MSE for the last point was 0.01. We are getting better.

But the memory is still limited to 100 time steps.

We can do more if we combine memory cells with CNN.

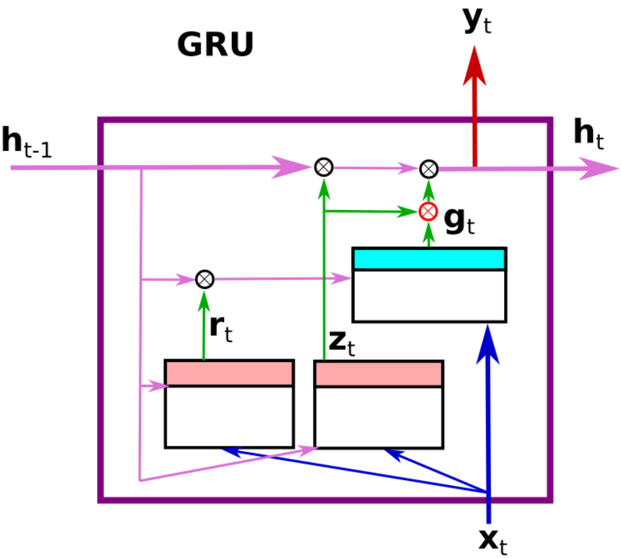
GRU is a simplified version of LSTM

$$\mathbf{z}_t = \sigma(\mathbf{W}_{xz}\mathbf{x}_t + \mathbf{W}_{hz}\mathbf{h}_{t-1} + \mathbf{b}_z)$$

$$\mathbf{r}_t = \sigma(\mathbf{W}_{xr}\mathbf{x}_t + \mathbf{W}_{hr}\mathbf{h}_{t-1} + \mathbf{b}_r)$$

$$\mathbf{g}_t = \tanh\left(\mathbf{W}_{xg}\mathbf{x}_{t-1} + \mathbf{W}_{hg}(\mathbf{r}_t.\mathbf{h}_{t-1}) + \right.$$

$$\mathbf{h}_t = \mathbf{z}_t.\mathbf{h}_{t-1} + (1 - \mathbf{z}_t).\mathbf{g}_t$$



GRU plus 1D CNN

- GRU and LSTM allow to increase the memory to hundreds of time-steps

```
model = keras.models.Sequential([
keras.layers.Conv1D(filters=20,kernel_size=4,strides=2,padding="valid",
input_shape=[None,1]), keras.layers.GRU(20, return_sequences=True),
keras.layers.GRU(20, return_sequences=True),
keras.layers.TimeDistributed(keras.layers.Dense(10)) ])
```

GRU plus 1D CNN

- GRU and LSTM allow to increase the memory to hundreds of time-steps
- But we might need to work with much longer sequences

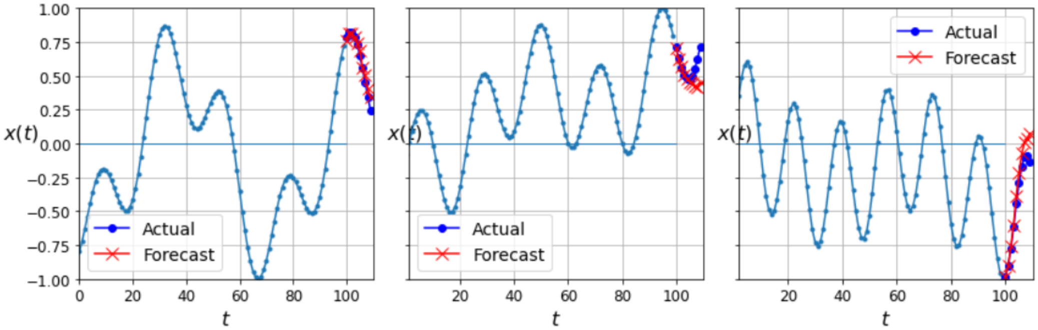
```
model = keras.models.Sequential([
keras.layers.Conv1D(filters=20,kernel_size=4,strides=2,padding="valid",
input_shape=[None,1]), keras.layers.GRU(20, return_sequences=True),
keras.layers.GRU(20, return_sequences=True),
keras.layers.TimeDistributed(keras.layers.Dense(10)) ])
```

GRU plus 1D CNN

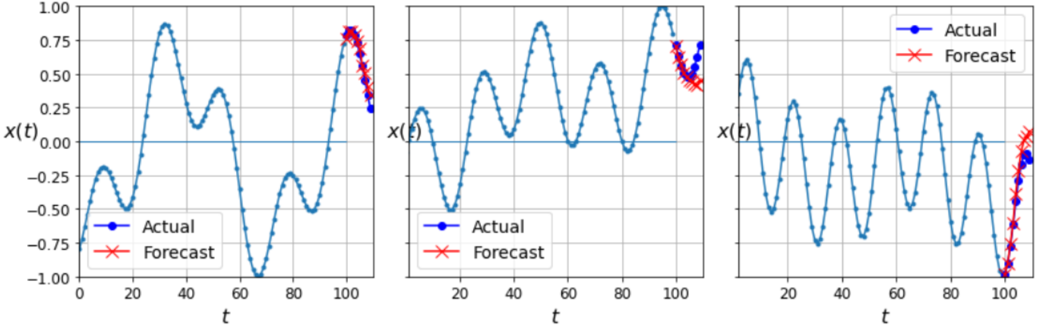
- GRU and LSTM allow to increase the memory to hundreds of time-steps
- But we might need to work with much longer sequences
- The remedy is to use CNN

```
model = keras.models.Sequential([
keras.layers.Conv1D(filters=20,kernel_size=4,strides=2,padding="valid",
input_shape=[None,1]), keras.layers.GRU(20, return_sequences=True),
keras.layers.GRU(20, return_sequences=True),
keras.layers.TimeDistributed(keras.layers.Dense(10)) ])
```

RNN plus 1D CNN

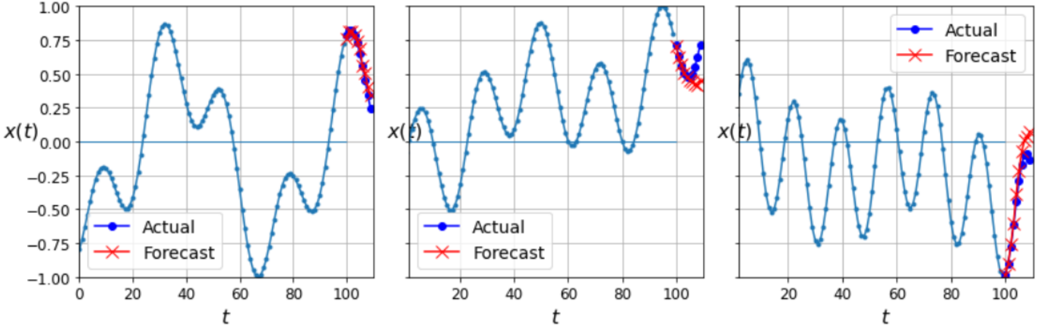


RNN plus 1D CNN



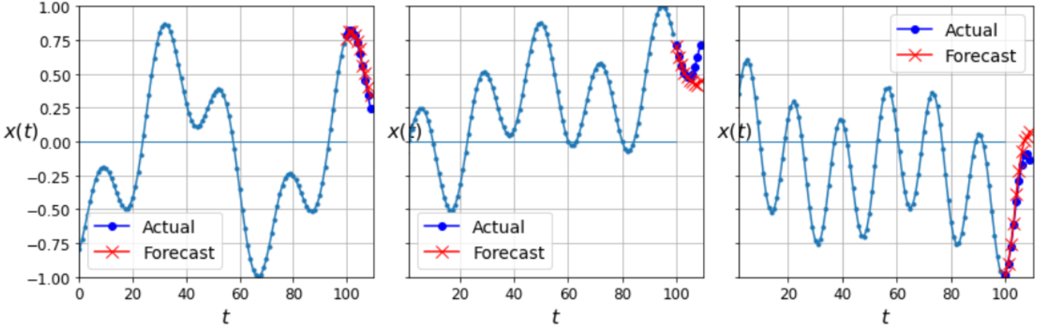
This is so far the best network. Its MSE is for last predicted point is 0.0085

RNN plus 1D CNN



This is so far the best network. Its MSE is for last predicted point is 0.0085
Yet for soma tasks it can be still improved simply by dropping the RNN and using just CNNs E.g., by using WaveNet.

RNN plus 1D CNN



This is so far the best network. Its MSE is for last predicted point is 0.0085

Yet for soma tasks it can be still improved simply by dropping the RNN and using just CNNs E.g., by using WaveNet.

WaveNet