Lecture 11:

Long Short-Term Memory Networks (LSTMs) – Part I

Markus Hohle

University California, Berkeley

Machine Learning Algorithms

MSSE 277B, 3 Units

Fall 2024

https://www.analyticsvidhya.com

Outline

- **Idea and classic RNNs**
- **LSTMs**
- ***B*ack*P*ropagation *T*hrough *T*ime (BPTT)**
- **Syntax and some examples**

https://www.analyticsvidhya.com

Outline

- Idea and classic RNNs

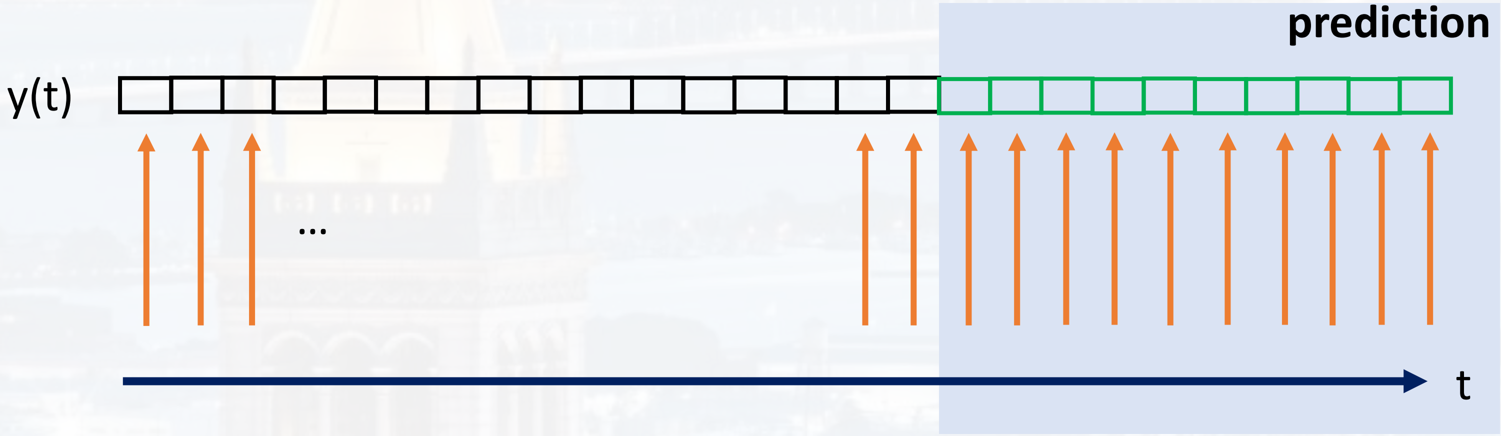
- LSTMs

- *B*ack*P*ropagation *T*hrough *T*ime (BPTT)

- Syntax and some examples

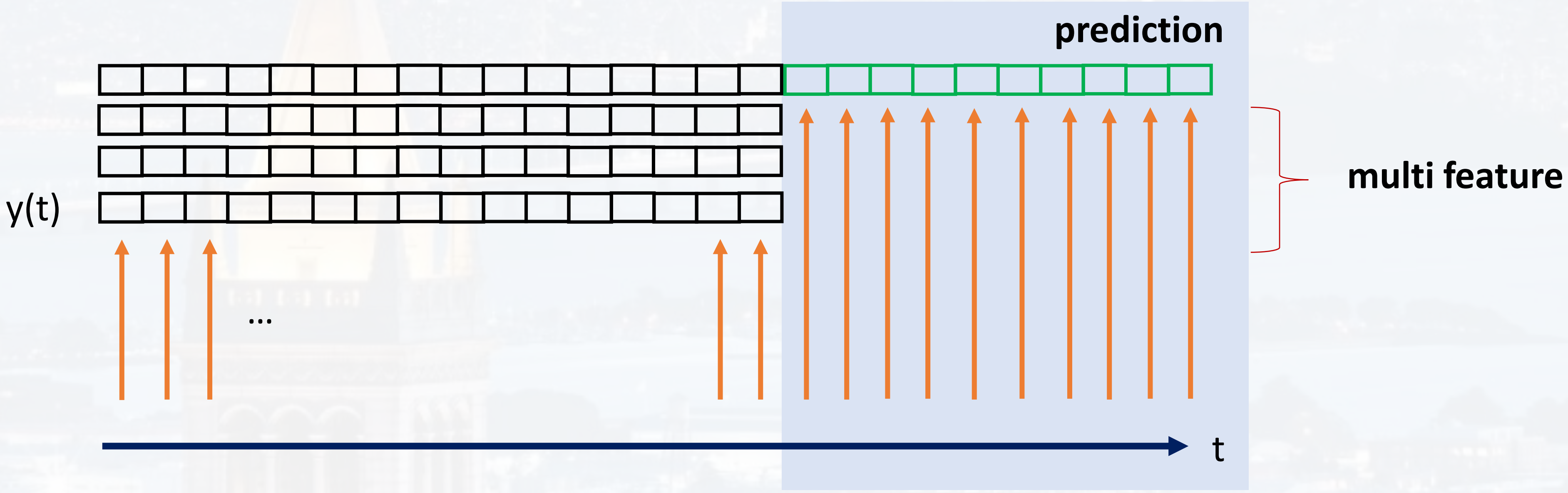
- **R**ecurrent **N**eural **N**etwork

- time series analysis regression (**prediction** and **forecasting**)
- first step towards GenAI
- time series analysis classification
- early speech recognition
- handwriting
- “precursor” of LSTMs
- invented by **Shun'ichi Amari** in 1972

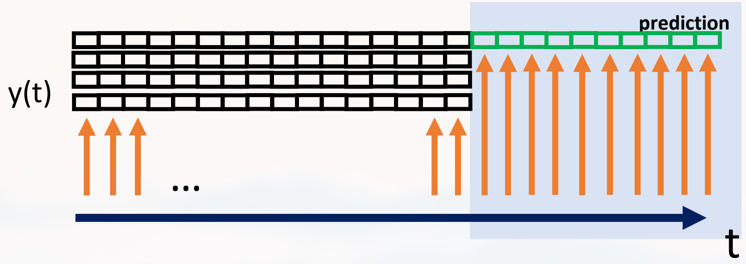


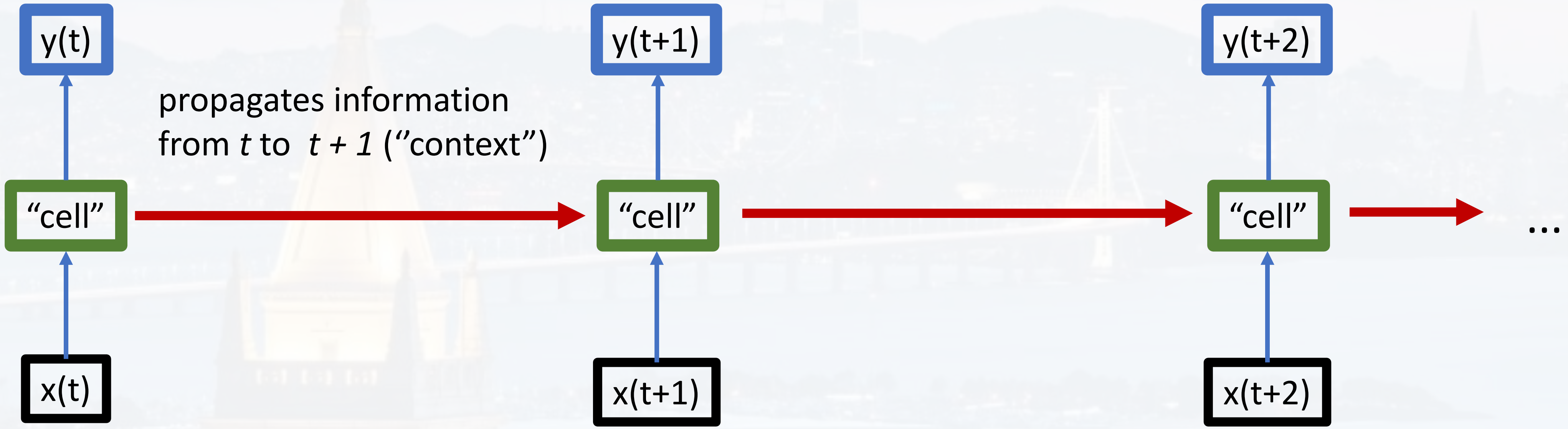
- **R**ecurrent **N**eural **N**etwork

- time series analysis regression (**prediction** and **forecasting**)
- first step towards GenAI
- time series analysis classification
- early speech recognition
- handwriting
- “precursor” of LSTMs
- invented by **Shun'ichi Amari** in 1972



LSTMs: Idea and classic RNNs

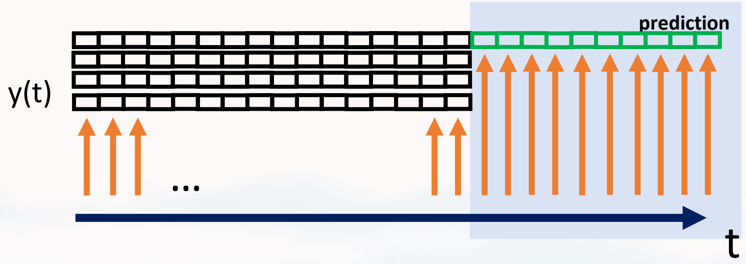


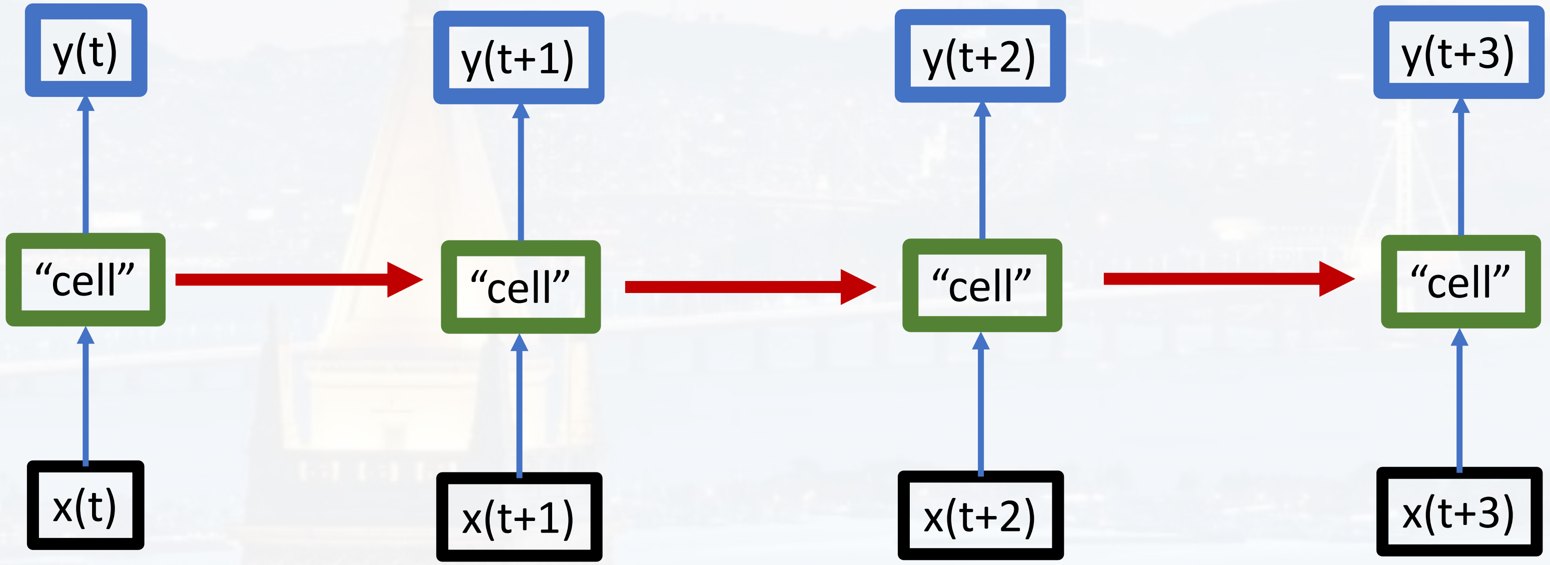


LSTMs:

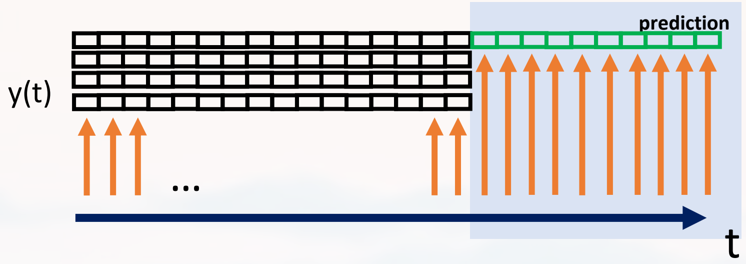
Idea and classic RNNs

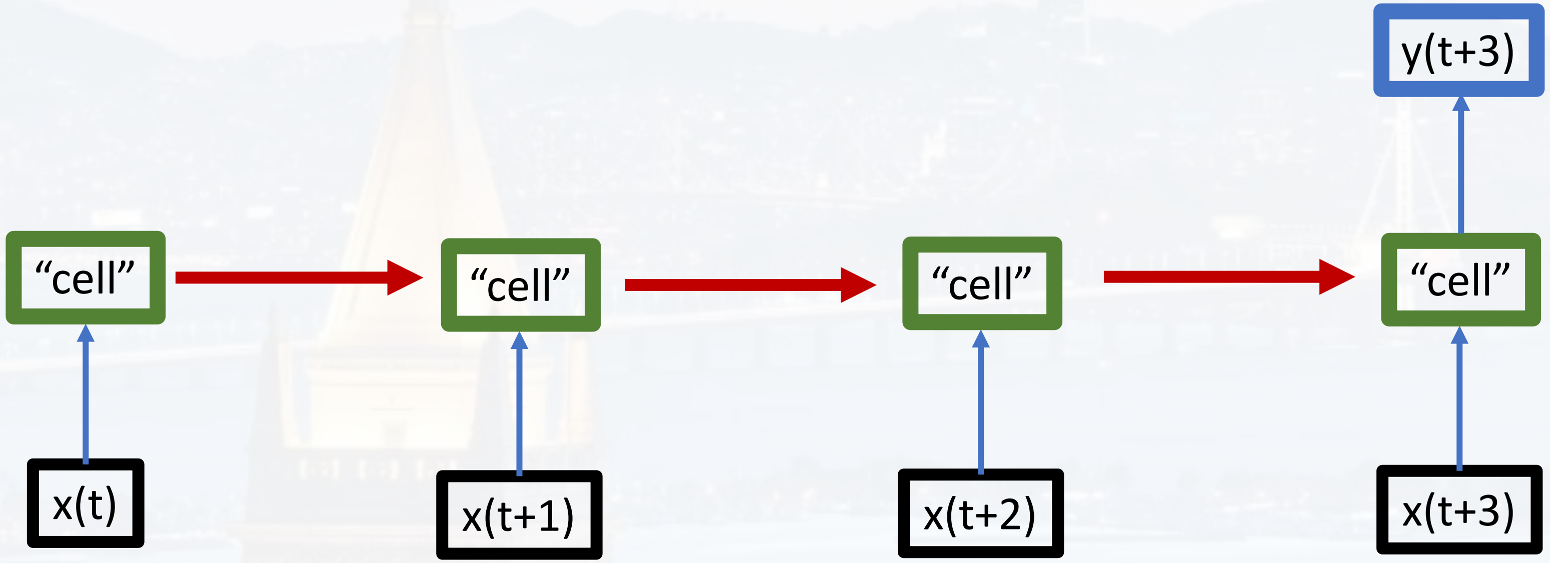
“many to many”





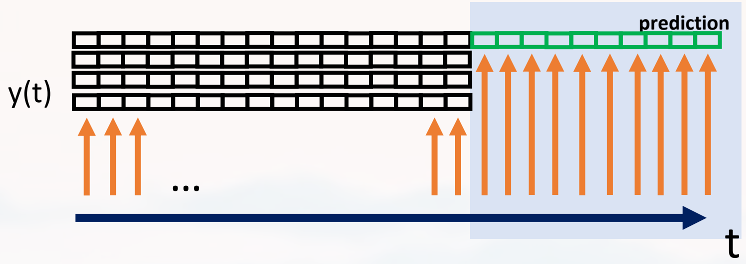
“many to one”
(classification)

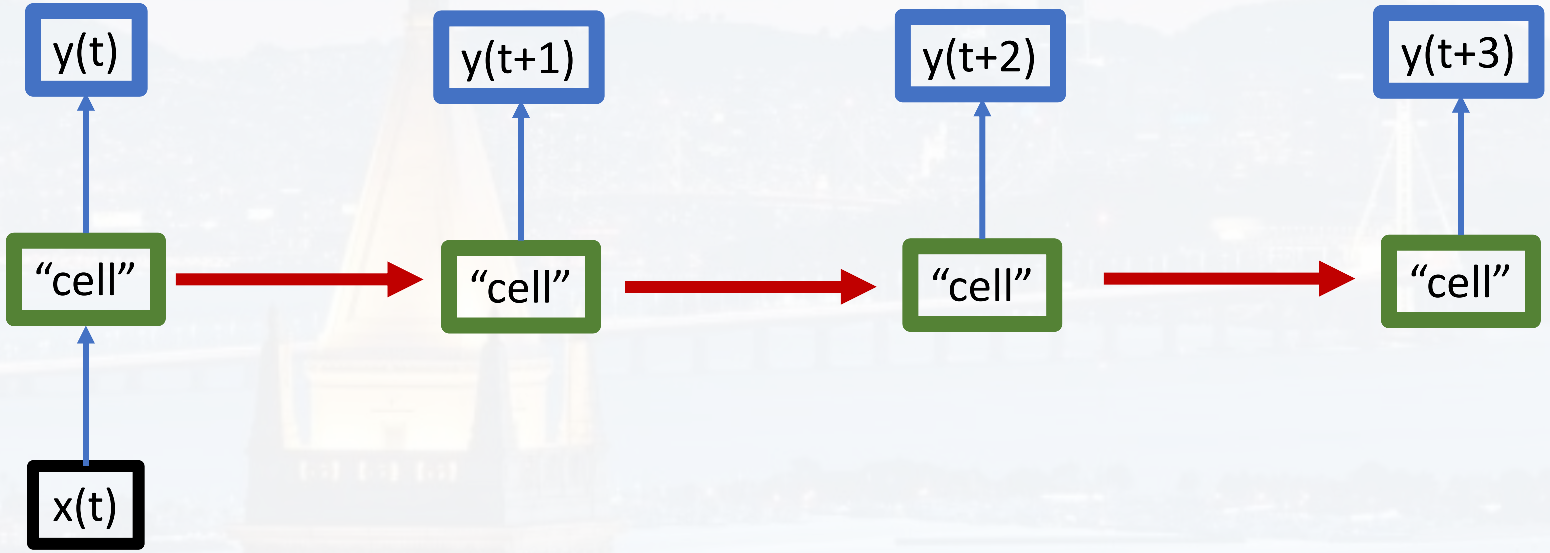




LSTMs:

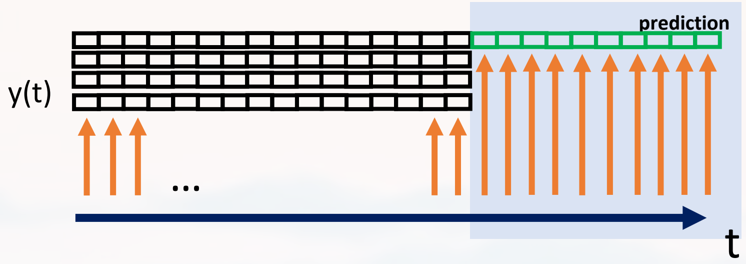
“one to many”
(image capturing)





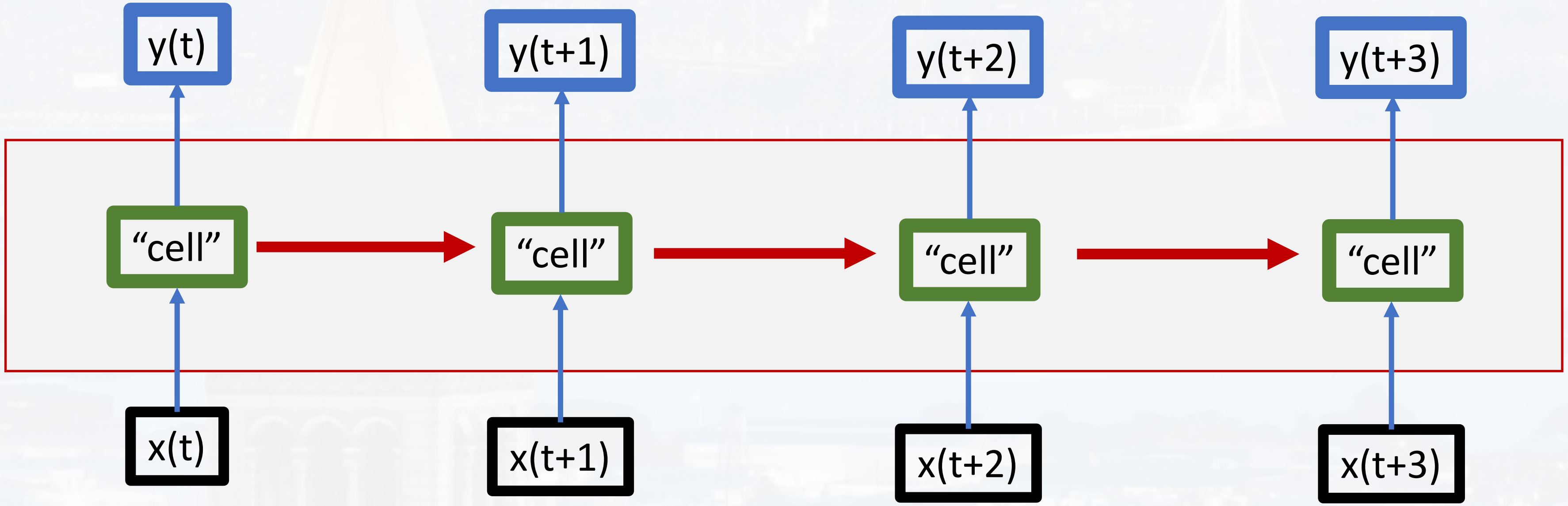
LSTMs: Idea and classic RNNs

Applying the **identical** cell **recursively**!

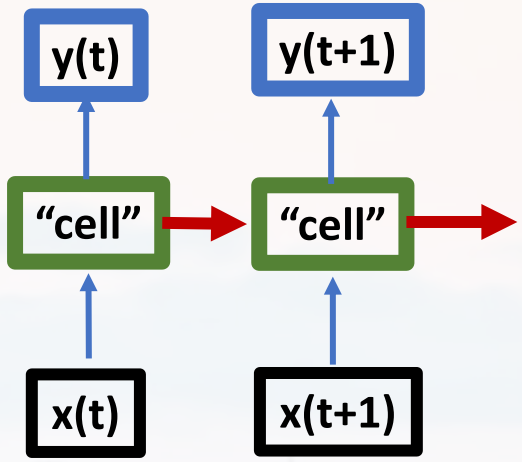


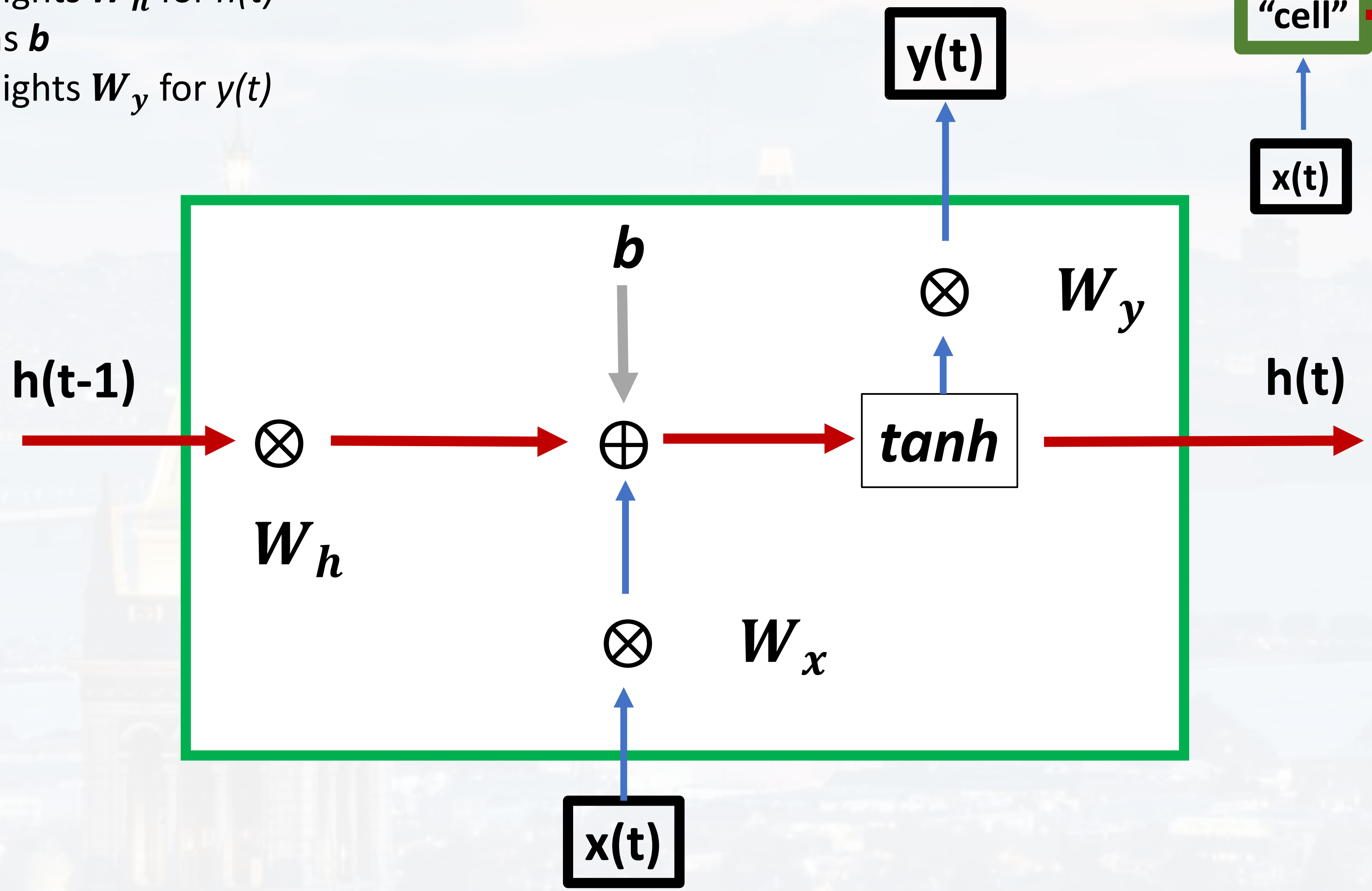
→ easy to implement
→ direction (arrow of time, see later)

→ exploding/vanishing gradients
→ classic RNN has a “short memory”

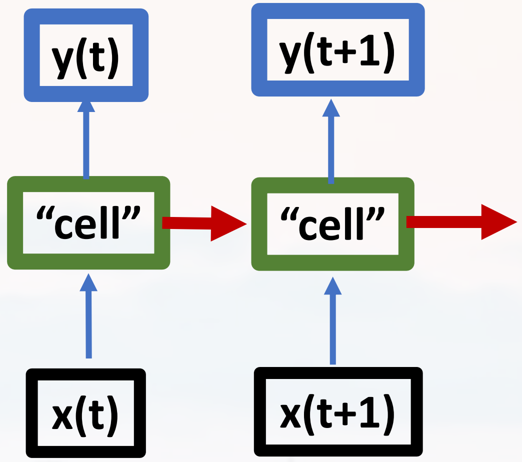


- **state vector** *h(t)*, i. e. the *"memory"* the system has along time
- weights W_x for *x(t)*
- weights W_h for *h(t)*
- bias b
- weights W_y for *y(t)*

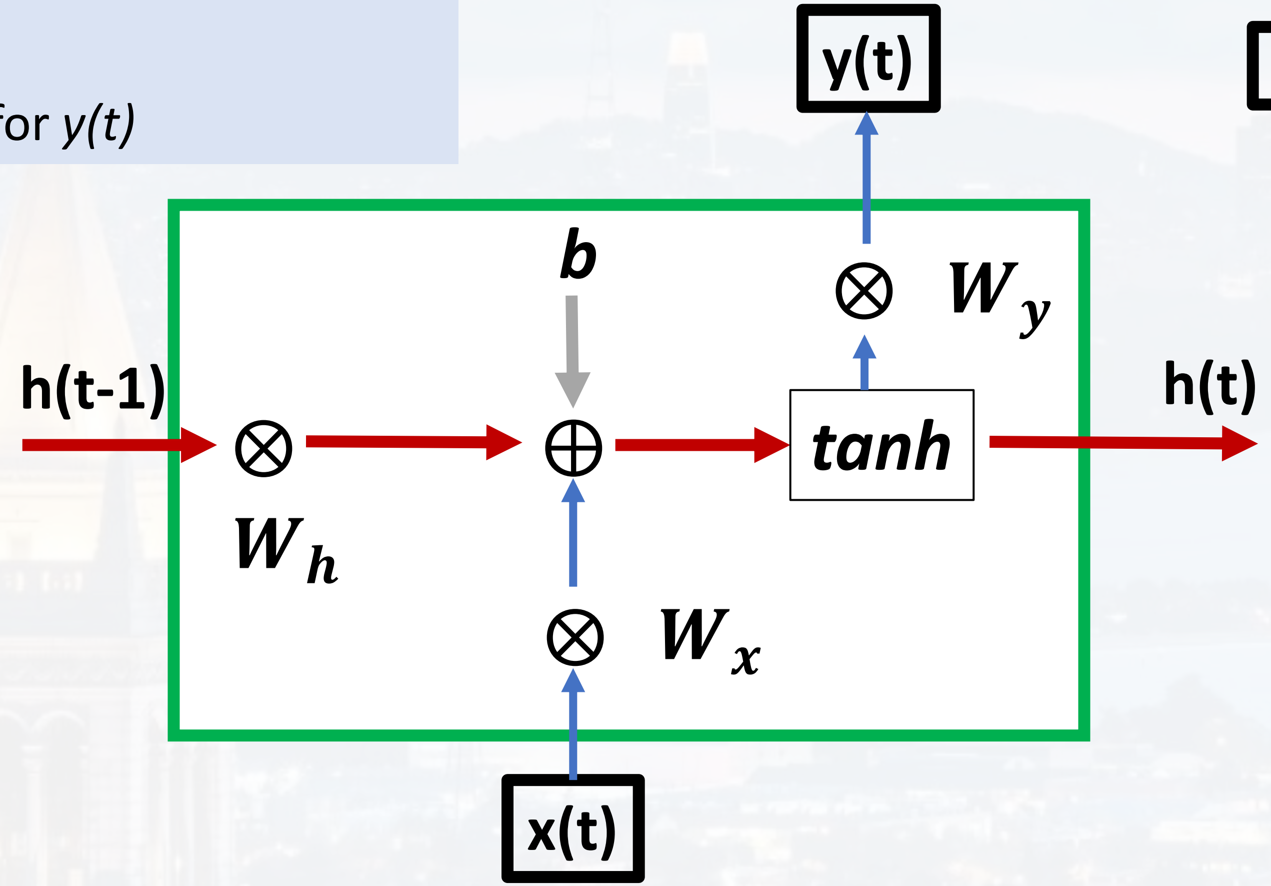




- **state vector** *h(t)*, i. e. the *"memory"* the system has along time



- weights $\boldsymbol{W_x}$ for *x(t)* **learnable**
- weights $\boldsymbol{W_h}$ for *h(t)*
- bias ***b***
- weights $\boldsymbol{W_y}$ for *y(t)*



$$h(t) = \tanh[\, x(t) * W_x + \boxed{W_h} * h(t-1) + b\,]$$

1 x 1

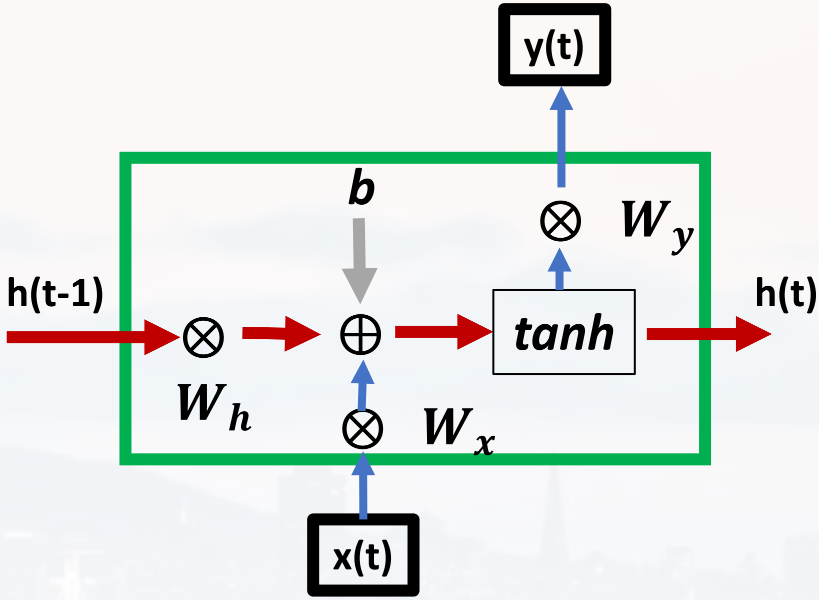
n x 1

n x 1

n x 1

n x 1

n x n

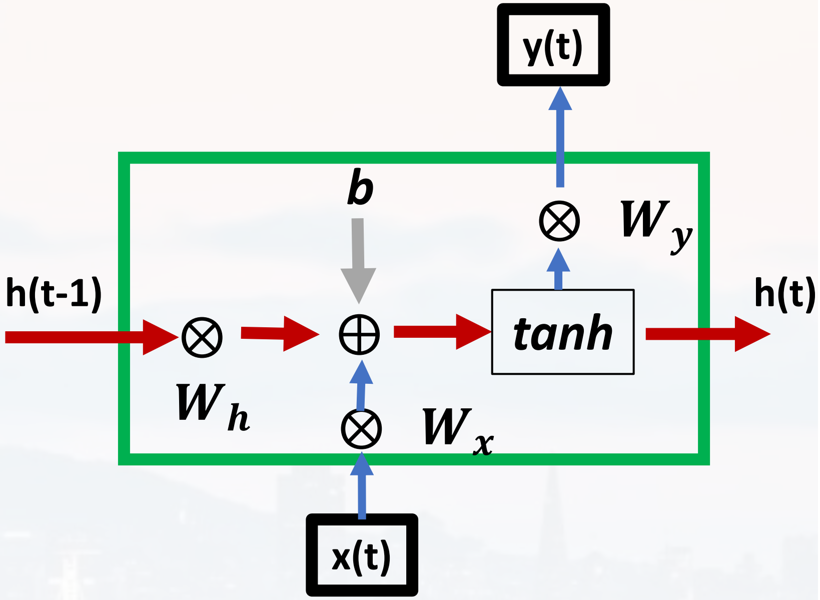


n: number of states/ neurons

$$y(t) = h(t) * W_y$$

LSTMs:

$$h(t) = \tanh[\, x(t) * W_x + W_h * h(t-1) + b \,]$$



$$y(t) = h(t) * W_y$$

1 x 1

1 x n

n x 1

n: number of states/ neurons

note, for **M** features: $x(t)$

M x 1

https://www.analyticsvidhya.com

Outline

- Idea and classic RNNs
- **LSTMs**
- *B*ack*P*ropagation *T*hrough *T*ime (BPTT)
- Syntax and some examples

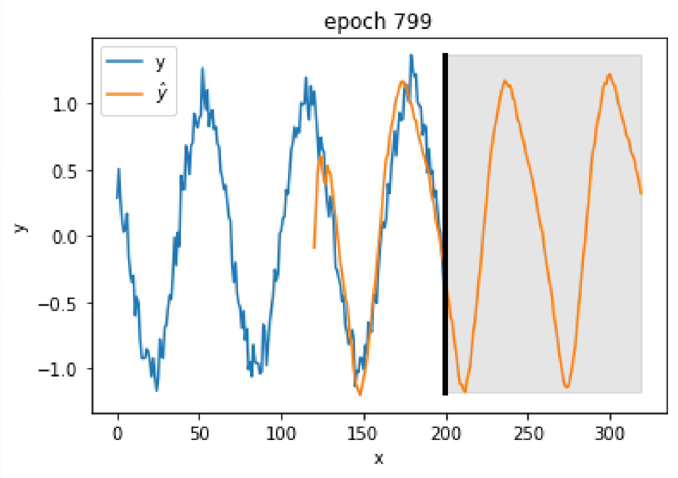
LSTMs:

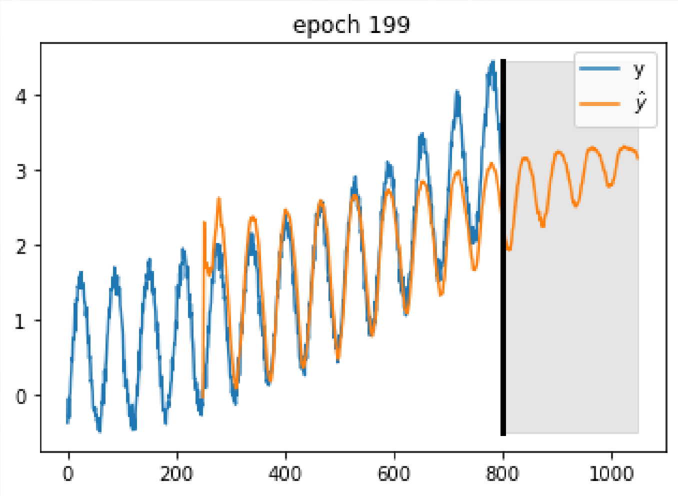
- **L**ong- **S**hort **T**erm **M**emory

<u>new:</u>

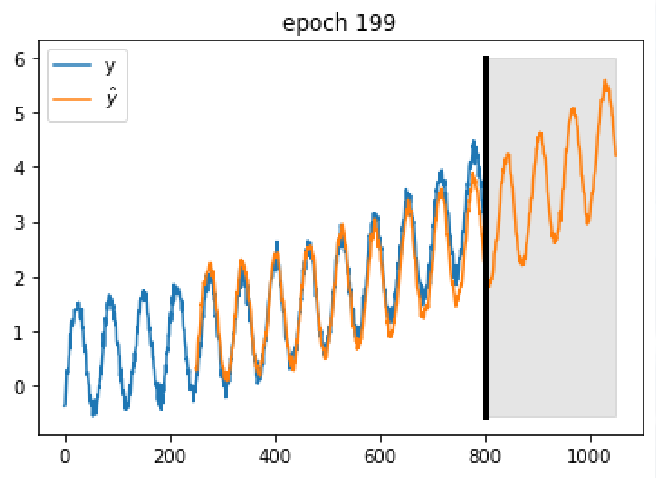
- **long-term** and **short-term** memory
- dealing with vanishing/exploding gradient
- invented 1997 by Sepp Hochreiter und Jürgen Schmidhuber

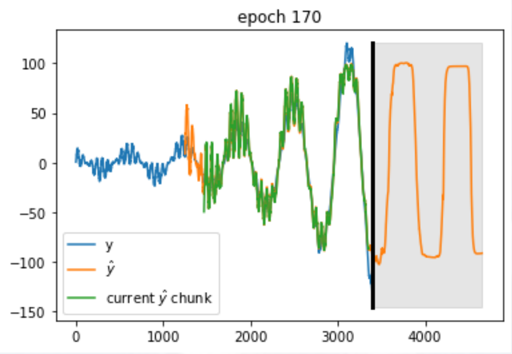
classical RNN





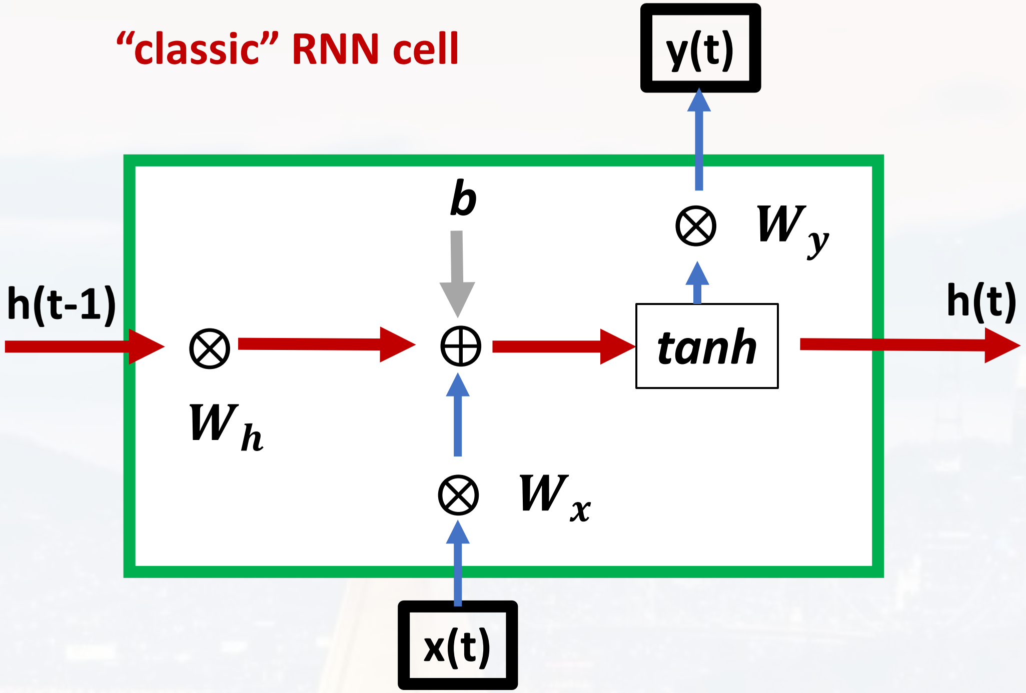
LSTM

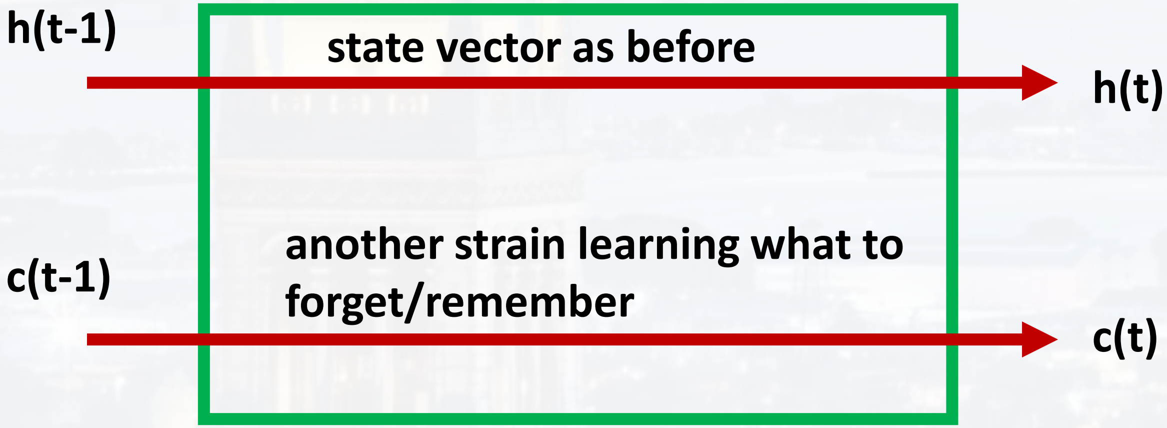




(adding more noise)

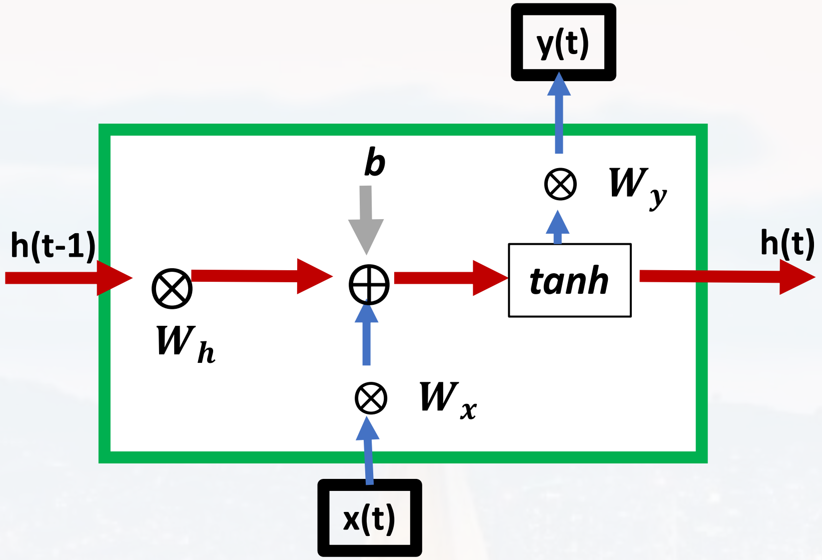
LSTMs:

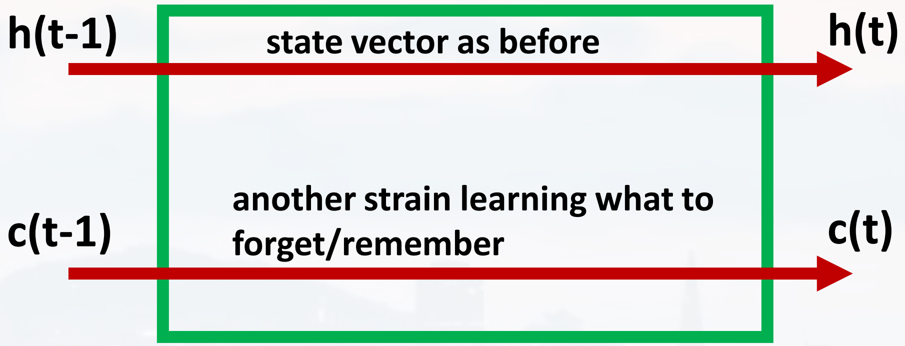

"classic" RNN cell
y(t)
b
W_y
h(t-1)
tanh
h(t)
W_h
W_x
x(t)

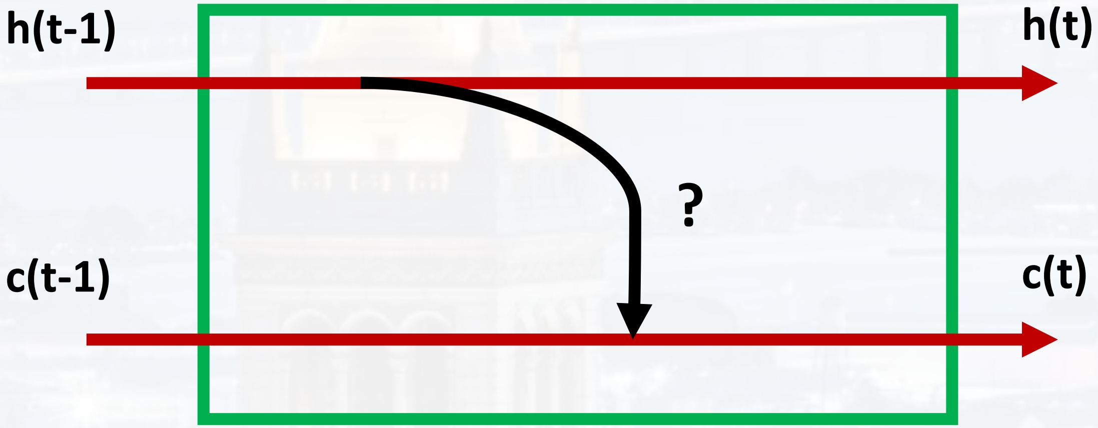

h(t-1)
state vector as before
h(t)
c(t-1)
another strain learning what to forget/remember
c(t)

LSTMs:

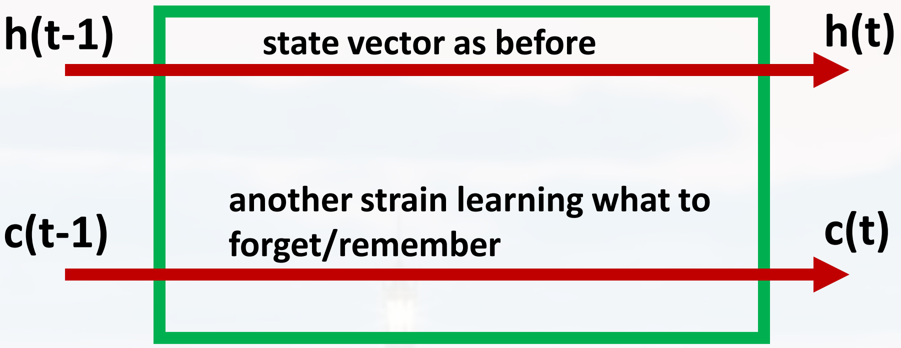
"classic" RNN cell

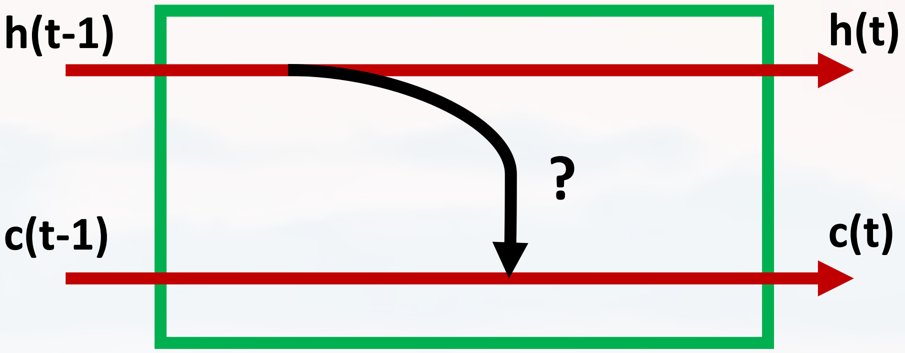




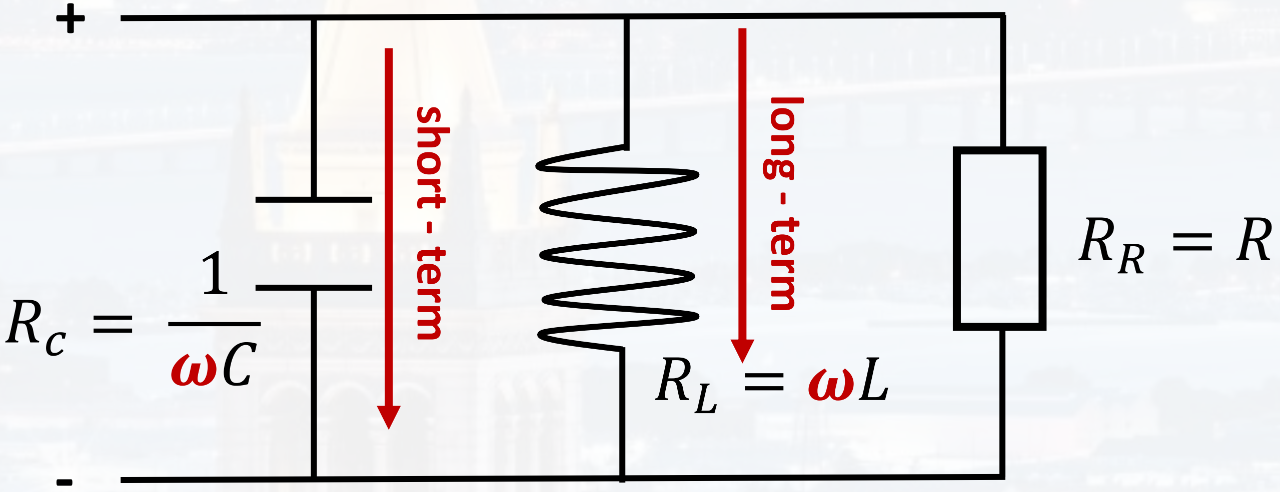


LSTMs:





electrical circuits:



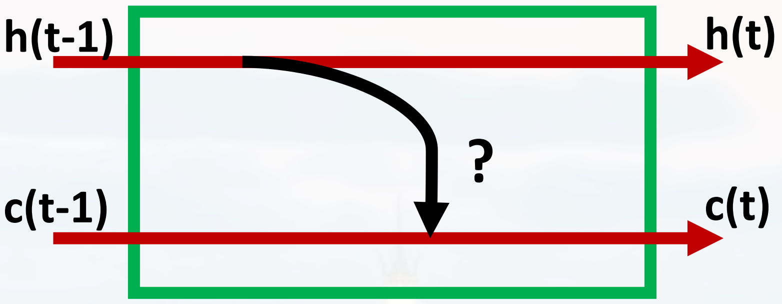
$\boxed{\textbf{AC:} \quad I(t) = I_0 \, e^{i(\omega t + \varphi)}}$

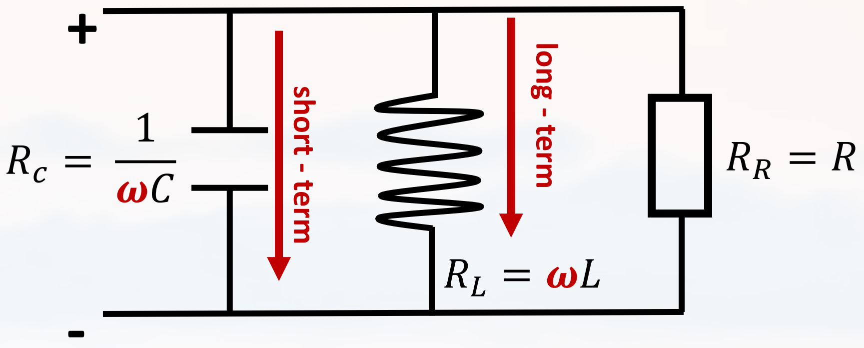
R_c: passes **short** -term changes

R_L: passes **long** -term changes

$$\frac{1}{R_{tot}} = \frac{1}{R_R} + \frac{1}{R_C} + \frac{1}{R_L}$$

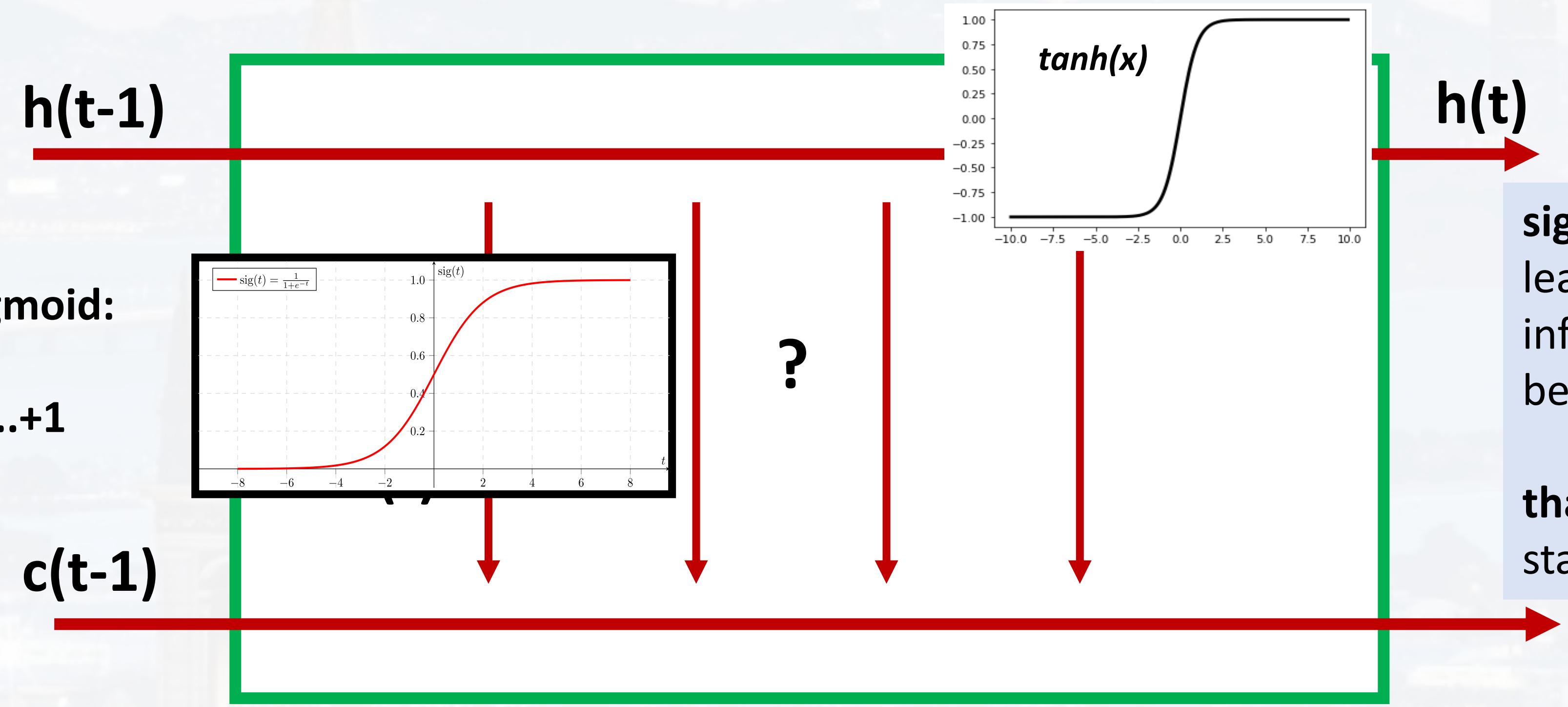
LSTMs:





sigmoid:

0….+1



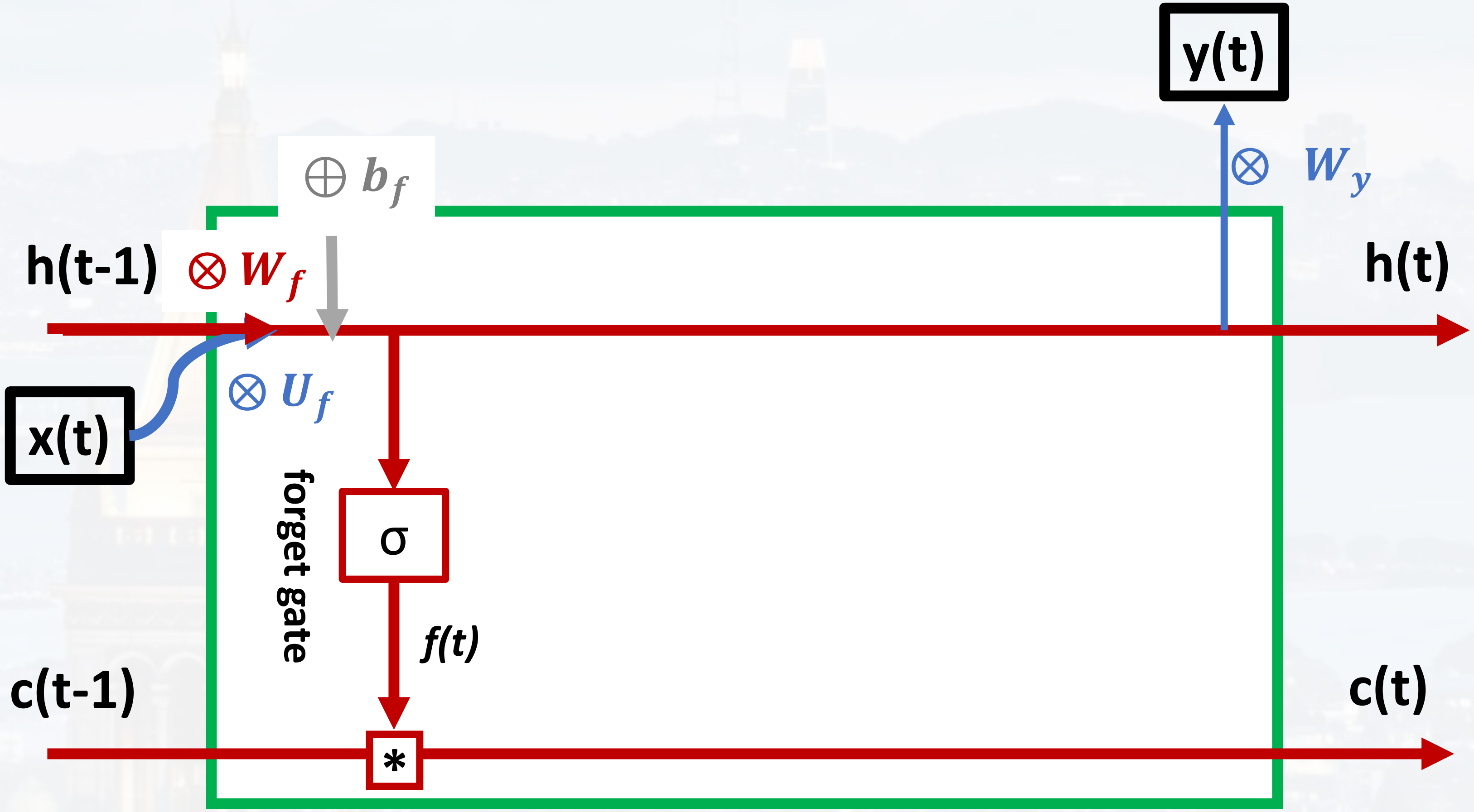
sigmoid:
learning whether information has to be passed on

thanh:
state vector as for RNN

LSTMs:

$$f(t) = \sigma\left(U_f \oplus x(t) + W_f \oplus h(t-1) + b_f\right)$$

$*$ element – wise multiplication



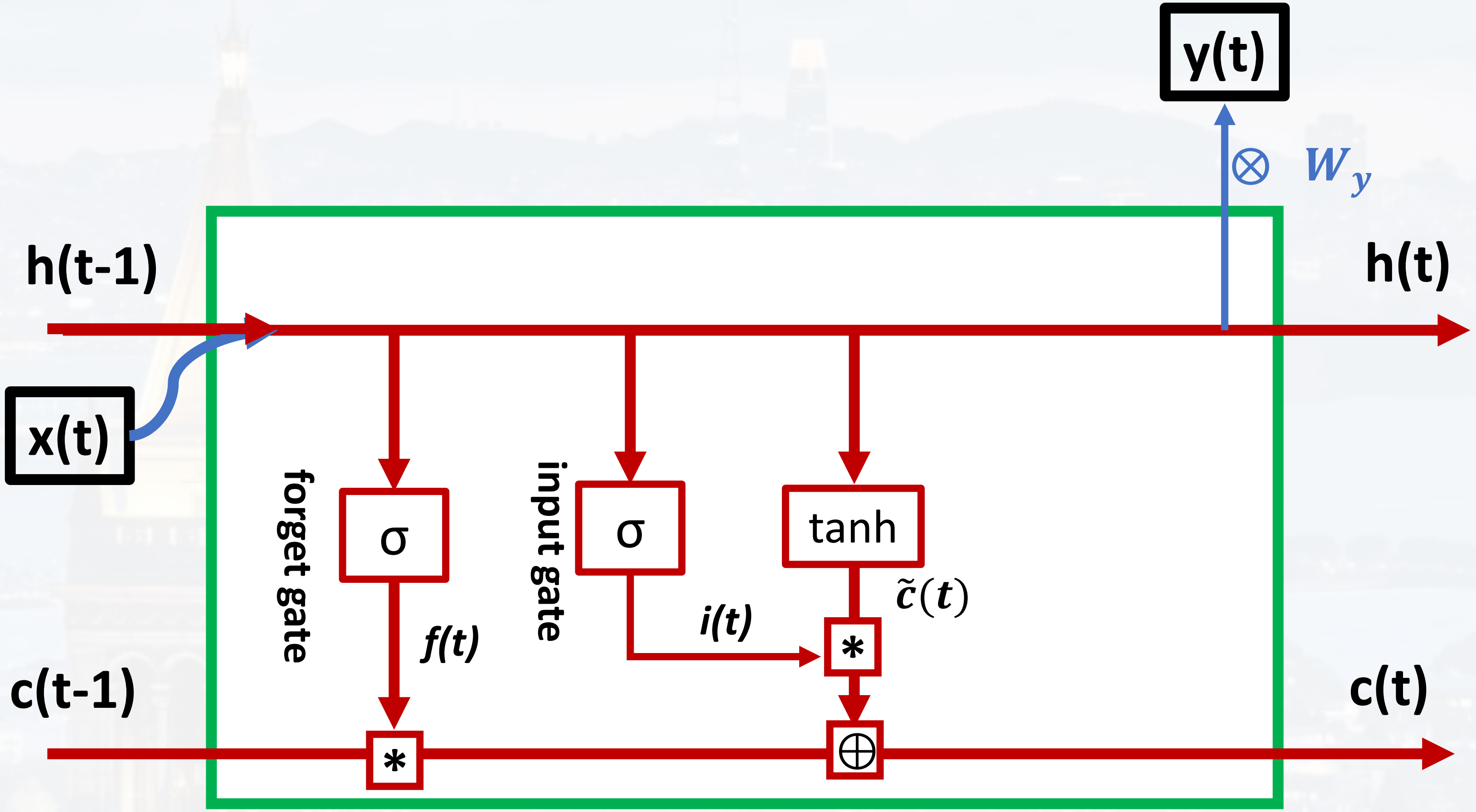
LSTMs:

$$f(t) = \sigma\left(U_f \oplus x(t) + W_f \oplus h(t-1) + b_f\right)$$

$$i(t) = \sigma\left(U_i \oplus x(t) + W_i \oplus h(t-1) + b_i\right)$$

$*$ element – wise multiplication

$$\tilde{c}(t) = tanh(U_g \oplus x(t) + W_g \oplus h(t-1) + b_g)$$



$$c(t) = f(t) * c(t-1) + i(t) * \tilde{c}(t)$$

LSTMs:

$$f(t) = \sigma\left(U_f \oplus x(t) + W_f \oplus h(t-1) + b_f\right)$$

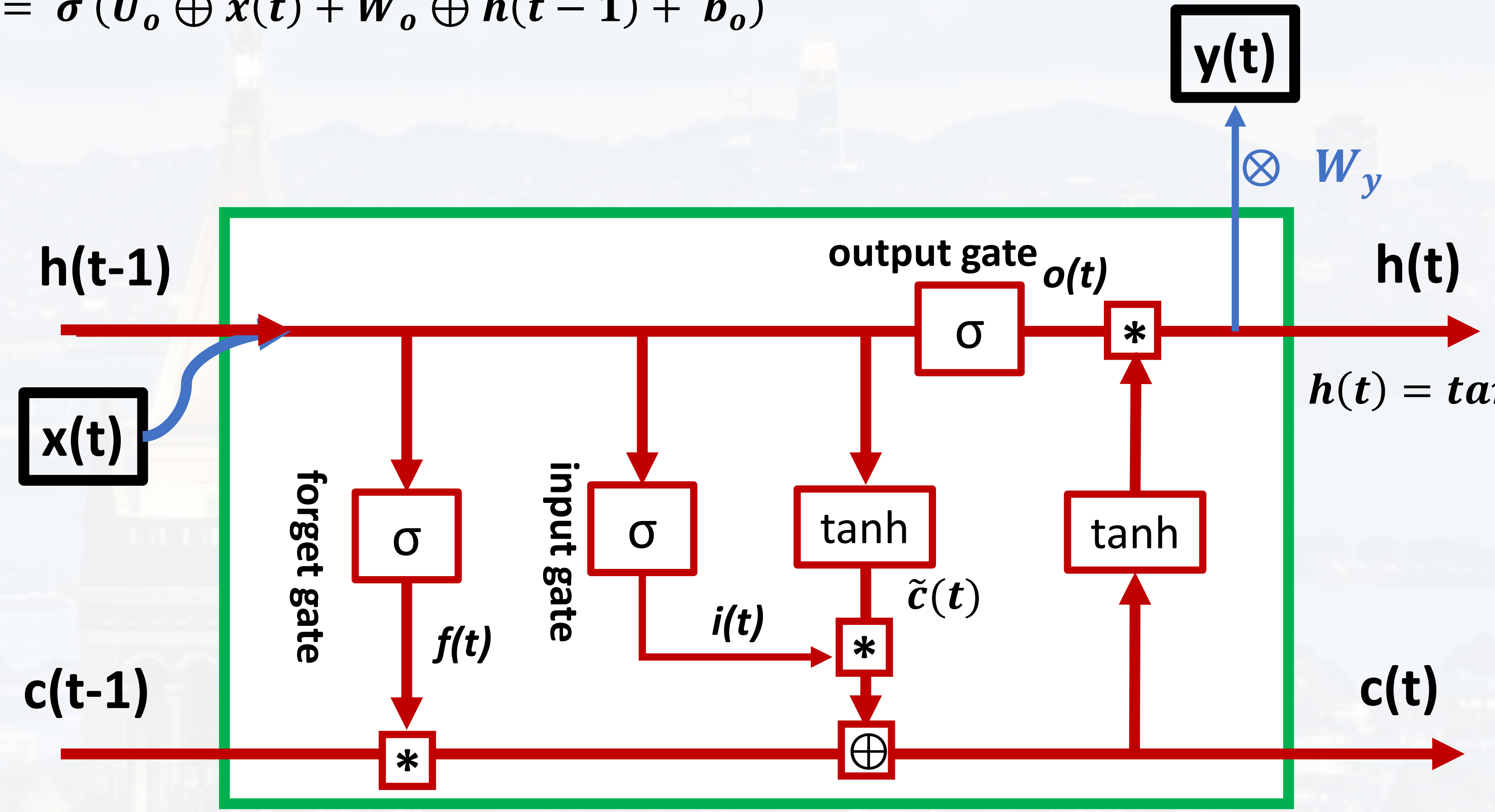
$*$ element – wise multiplication

$$i(t) = \sigma\left(U_i \oplus x(t) + W_i \oplus h(t-1) + b_i\right)$$

$$\tilde{c}(t) = tanh(U_g \oplus x(t) + W_g \oplus h(t-1) + b_g)$$

$$c(t) = f(t) * c(t-1) + i(t) * \tilde{c}(t)$$

$$o(t) = \sigma\left(U_o \oplus x(t) + W_o \oplus h(t-1) + b_o\right)$$



$$h(t) = tanh(c(t)) * o(t)$$

LSTMs:

$$f(t) = \sigma\left(U_f \oplus x(t) + W_f \oplus h(t-1) + b_f\right)$$

$*$ element – wise multiplication

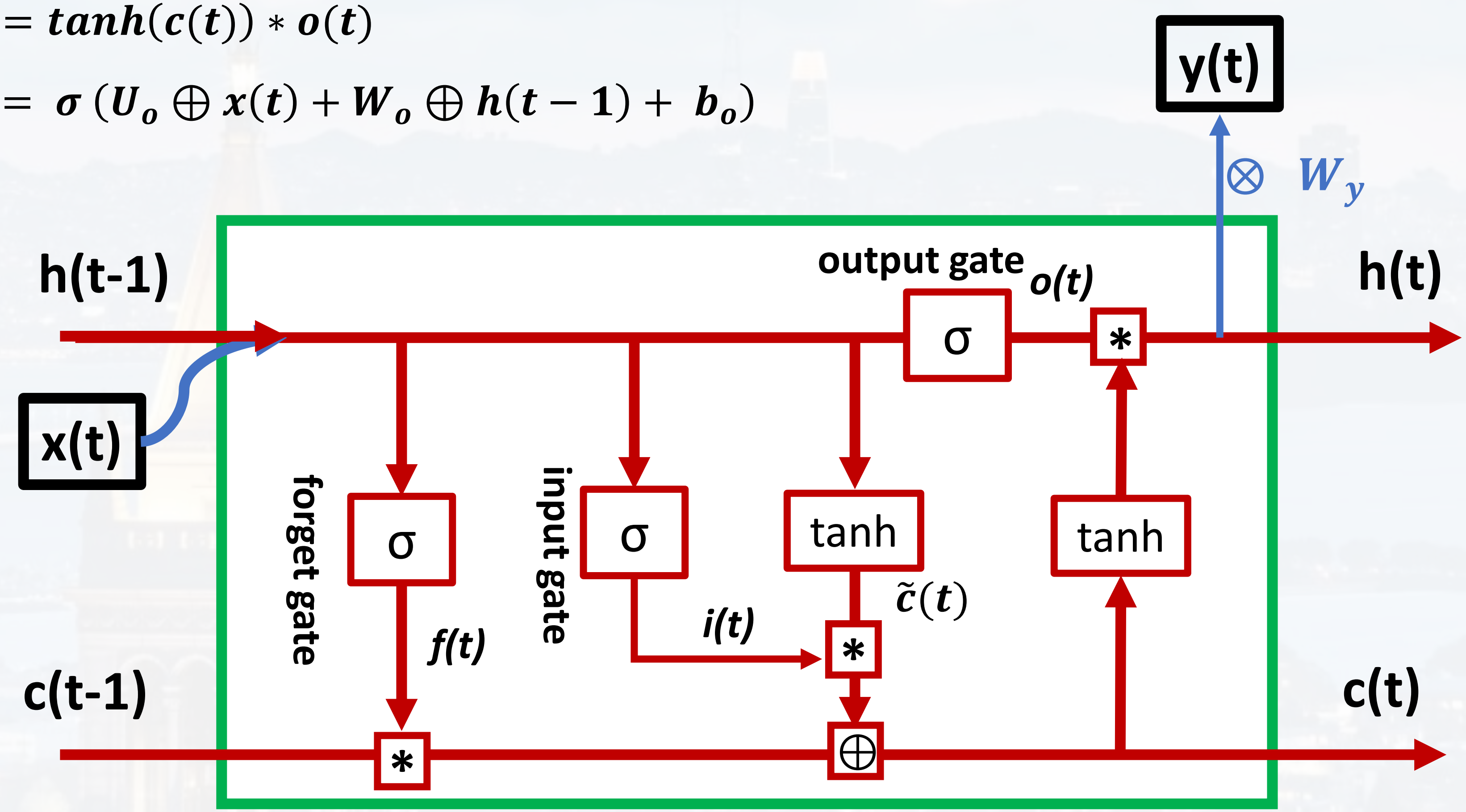
$$i(t) = \sigma\left(U_i \oplus x(t) + W_i \oplus h(t-1) + b_i\right)$$

$$\tilde{c}(t) = tanh(U_g \oplus x(t) + W_g \oplus h(t-1) + b_g)$$

$$c(t) = f(t) * c(t-1) + i(t) * \tilde{c}(t)$$

$$h(t) = tanh(c(t)) * o(t)$$

$$o(t) = \sigma\left(U_o \oplus x(t) + W_o \oplus h(t-1) + b_o\right)$$

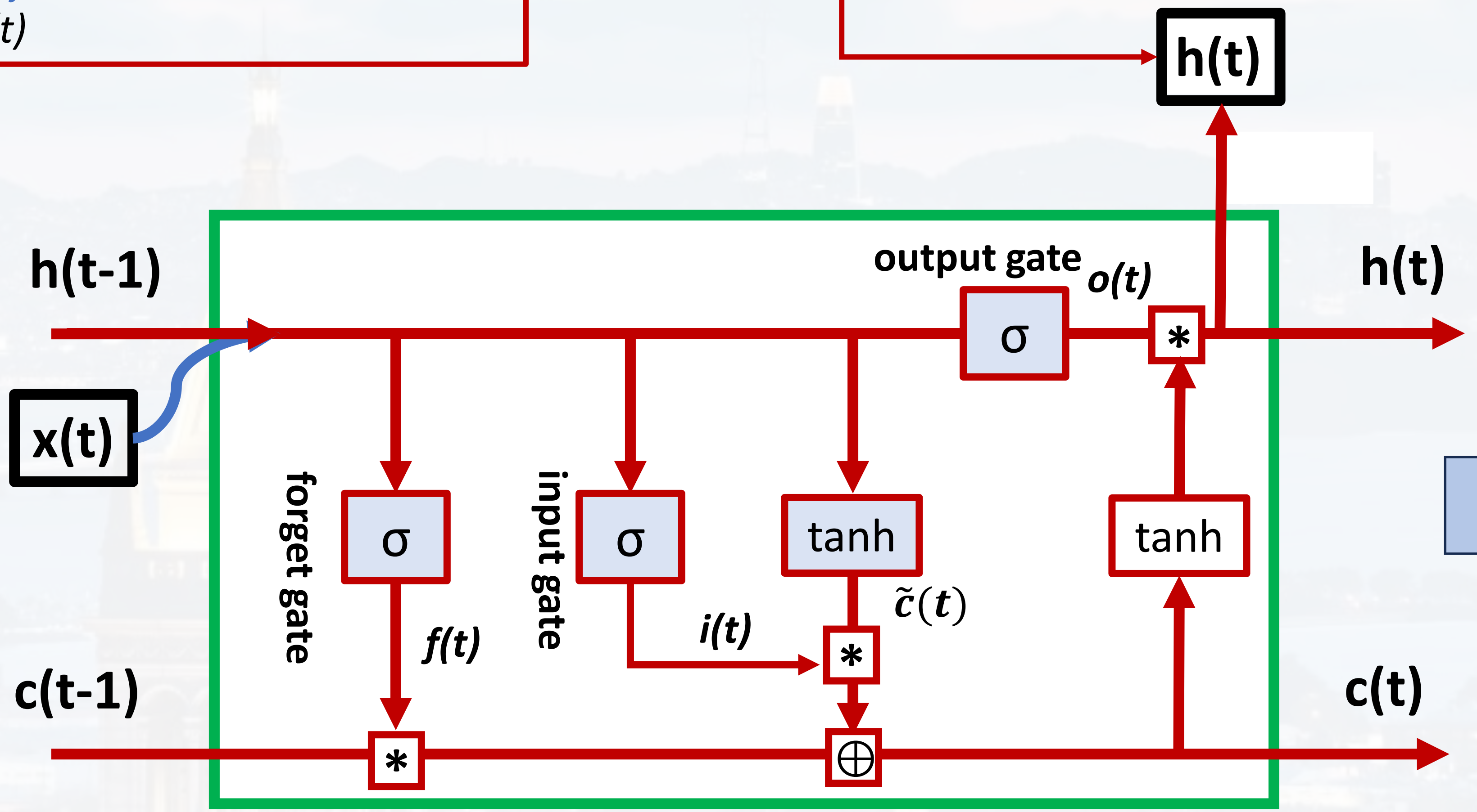


LSTMs:

There is one more thing:

we will add a **dense layer** instead of W_y at the end to convert *h(t)* to *y(t)*

* element – wise multiplication



contain learnables!

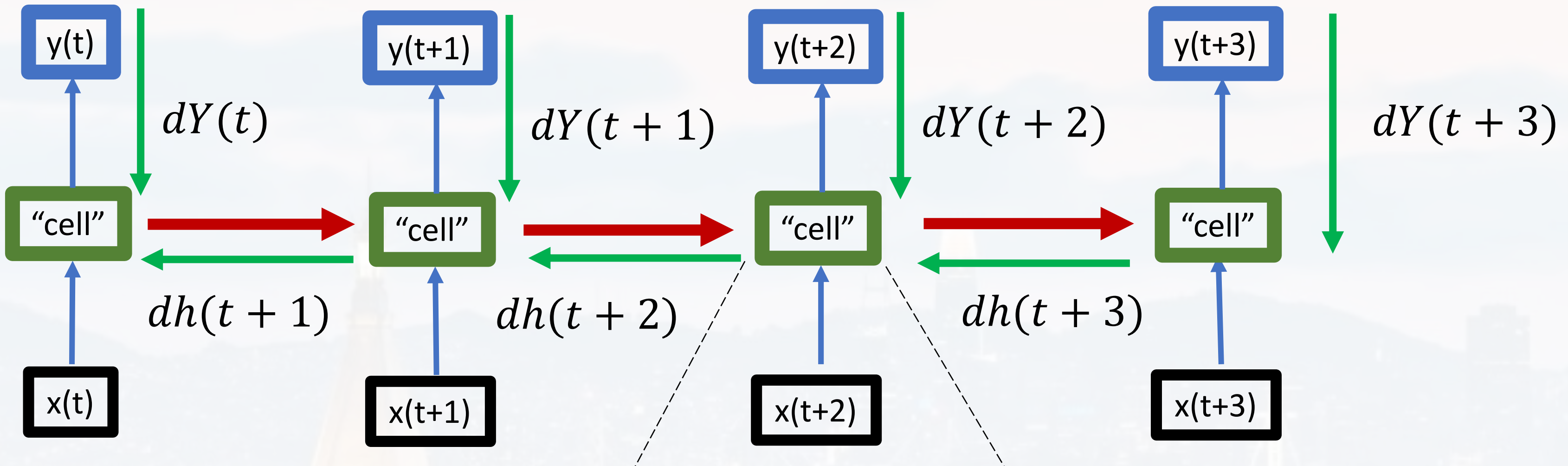
https://www.analyticsvidhya.com

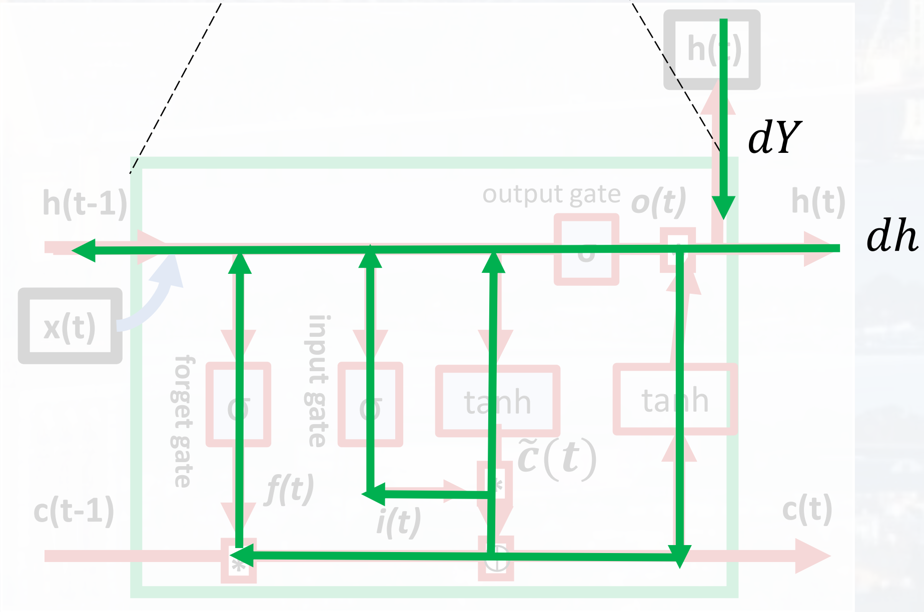
Outline

- Idea and classic RNNs
- LSTMs
- ***B**ack**P**ropagation **T**hrough **T**ime (BPTT)**
- Syntax and some examples

LSTMs:

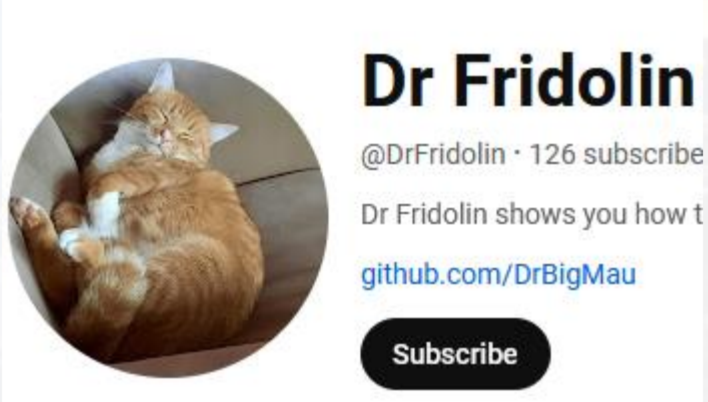
because of the RNN/LSTM architecture, backpropagation works a bit different:





backpropagation through time (BPTT)

more details here:



note: there is no dc since $c(t)$ is not a learnable!

https://www.analyticsvidhya.com

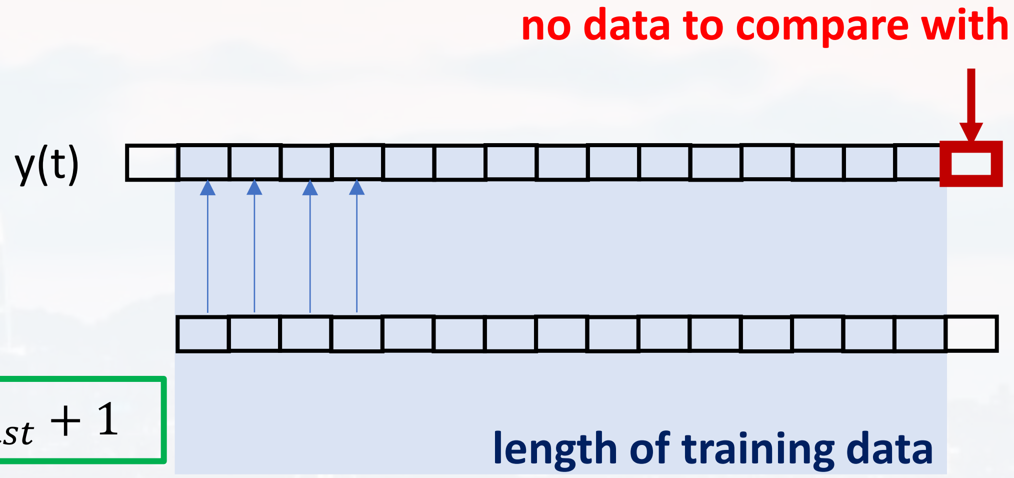
Outline

- Idea and classic RNNs
- LSTMs
- *Back**P**ropagation **T**hrough **T**ime* (BPTT)
- **Syntax and some examples**

Let us first understand the logic:

predicting **one** step in the future $dt_{futu} = 1$

by **one** step from the past $dt_{past} = 1$



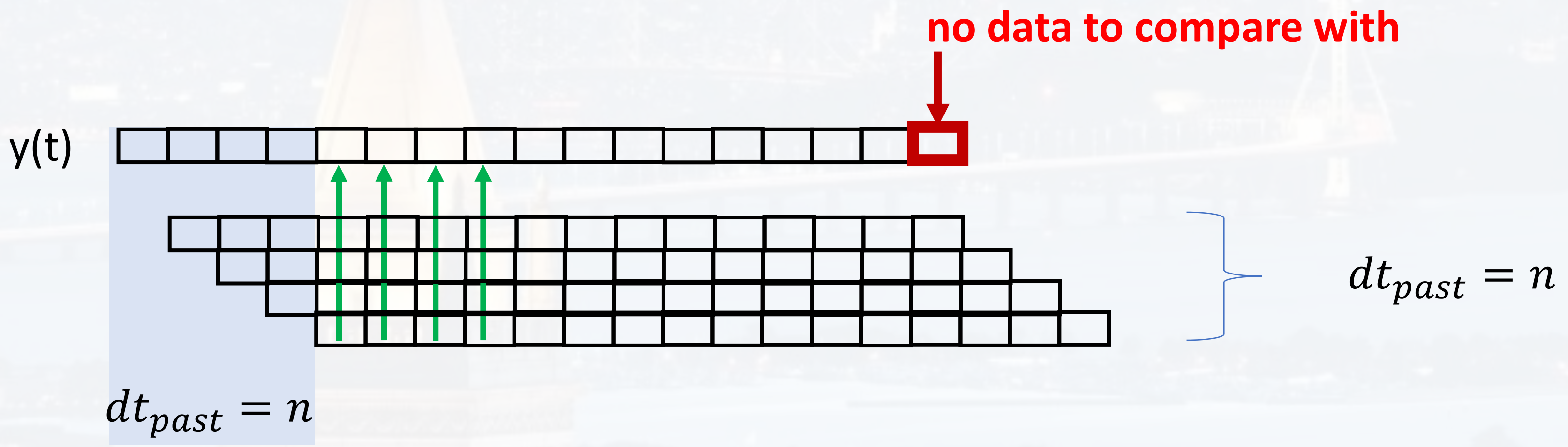
length of training data is: `len[y(t)]` - dt_{futu} - $dt_{past} + 1$

LSTMs:

Let us first understand the logic:

length of training data is: `len[y(t)]` - dt_{futu} - dt_{past} $+ 1$

predicting **m** steps in the future $\quad dt_{futu} = m$
by **n** steps from the past $\quad dt_{past} = n$



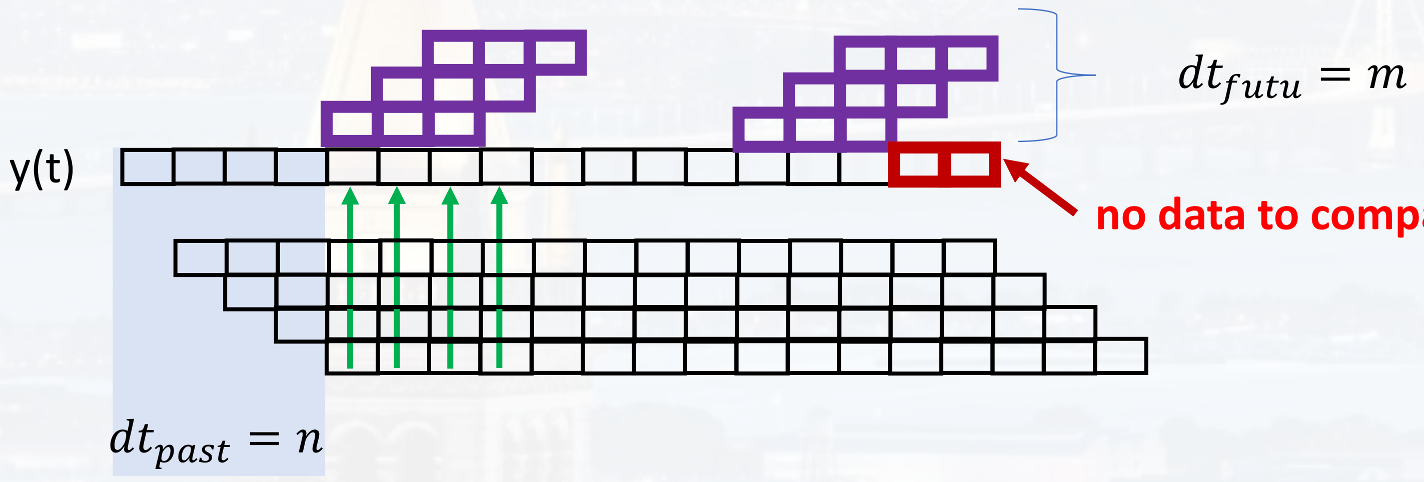
each of the time points
will influence the forecast

Let us first understand the logic:

length of training data is: `len[y(t)]` - dt_{futu} - dt_{past} $+ 1$

predicting **m** steps in the future $dt_{futu} = m$
by **n** steps from the past $dt_{past} = n$

predicting m steps of the future



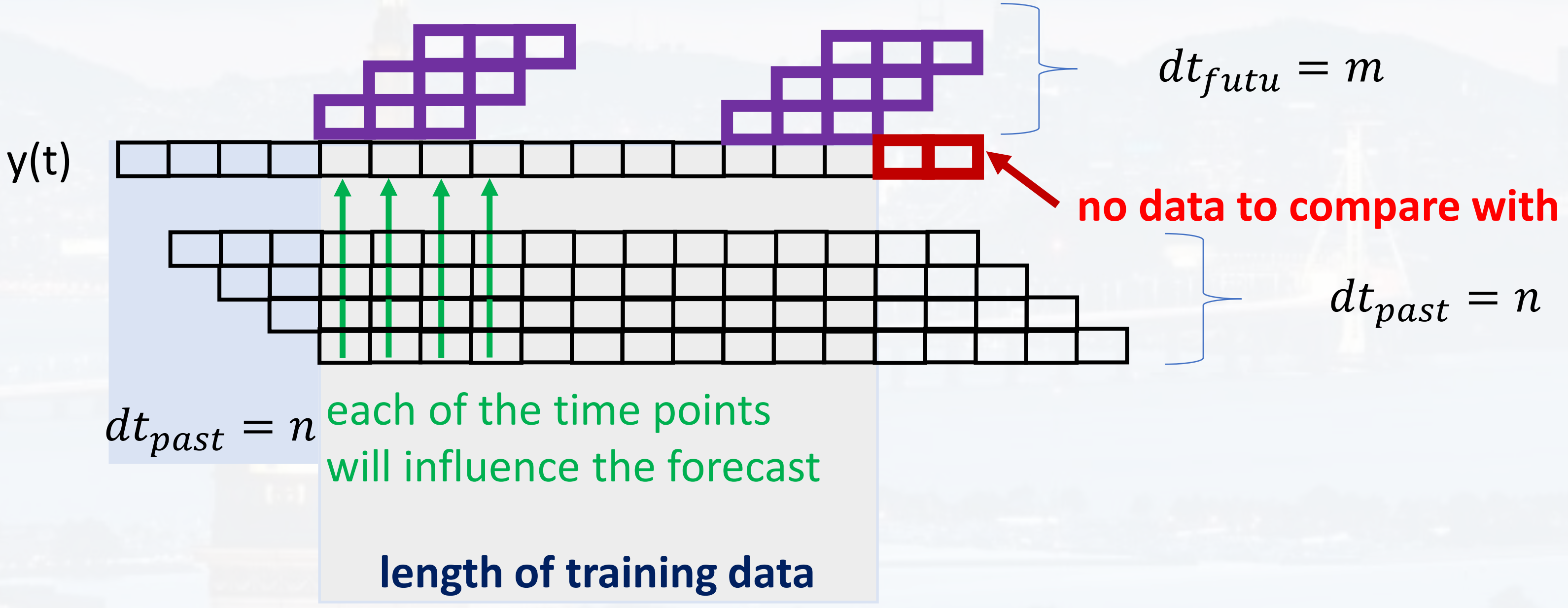
each of the time points
will influence the forecast

Let us first understand the logic:

predicting **m** steps in the future $dt_{futu} = m$

by **n** steps from the past $dt_{past} = n$

predicting m steps of the future



X.shape = (len[y(t)] - dt_{futu} - dt_{past} + 1) x dt_{past} x $n_{feature}(X)$

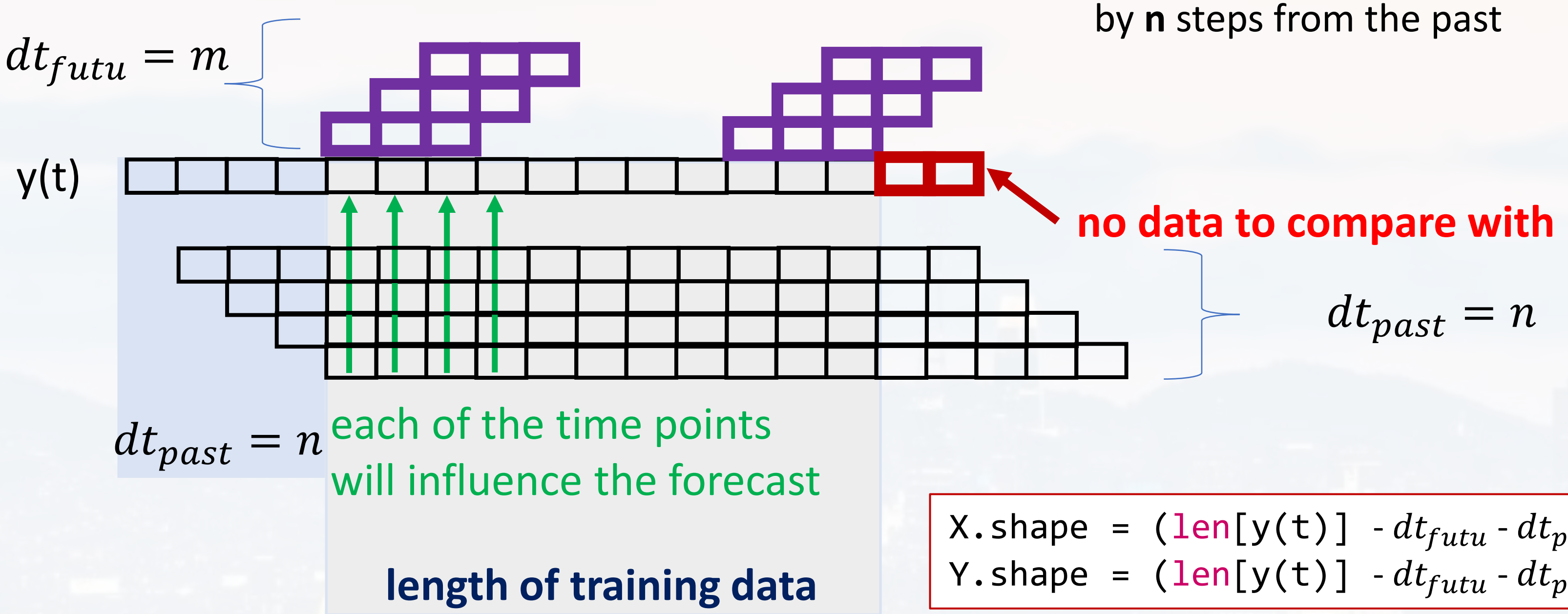
Y.shape = (len[y(t)] - dt_{futu} - dt_{past} + 1) x dt_{futu} x $n_{feature}(Y)$

LSTMs: Syntax and some examples

predicting m steps of the future

predicting **m** steps in the future $dt_{futu} = m$

by **n** steps from the past $dt_{past} = n$



X.shape = (len[y(t)] - dt_{futu} - dt_{past} +1) x dt_{past} x $n_{feature}(X)$

Y.shape = (len[y(t)] - dt_{futu} - dt_{past} + 1) x dt_{futu} x $n_{feature}(Y)$

X dt_{past}

len[y(t)] - dt_{futu} - dt_{past} + 1

```
[0.23364871, 0.25531086, 0.29226308, 0.30477917, 0.34526381]
[0.25531086, 0.29226308, 0.30477917, 0.34526381, 0.32876229]
[0.29226308, 0.30477917, 0.34526381, 0.32876229, 0.34967038]
[0.30477917, 0.34526381, 0.32876229, 0.34967038, 0.32374534]
[0.34526381, 0.32876229, 0.34967038, 0.32374534, 0.34168462]
[0.32876229, 0.34967038, 0.32374534, 0.34168462, 0.27602807]
[0.34967038, 0.32374534, 0.34168462, 0.27602807, 0.2313527 ]
[0.32374534, 0.34168462, 0.27602807, 0.2313527 , 0.20877584]
[0.34168462, 0.27602807, 0.2313527 , 0.20877584, 0.16455034]
[0.27602807, 0.2313527 , 0.20877584, 0.16455034, 0.11714726]
```

Y dt_{futu}

```
[0.05263142, 0.10779498, 0.12263184],
[0.10779498, 0.12263184, 0.12821065],
[0.12263184, 0.12821065, 0.20806335],
[0.12821065, 0.20806335, 0.2518744 ],
[0.20806335, 0.2518744 , 0.28025766],
[0.2518744 , 0.28025766, 0.27699119],
[0.28025766, 0.27699119, 0.30965494],
[0.27699119, 0.30965494, 0.37666627],
[0.30965494, 0.37666627, 0.37879347],
[0.37666627, 0.37879347, 0.36811853]]
```

LSTMs: Syntax and some examples

Let us first understand the logic:

Once, we have fitted the model: how do we apply the prediction?

```
PredY = model.predict(TestX)

(TestX.shape[0], dt_futu) = PredY.shape
```

X dt_{past}

len[y(t)] - dt_{futu} - dt_{past} + 1

```
[0.23364871, 0.25531086, 0.29226308, 0.30477917, 0.34526381]
[0.25531086, 0.29226308, 0.30477917, 0.34526381, 0.32876229]
[0.29226308, 0.30477917, 0.34526381, 0.32876229, 0.34967038]
[0.30477917, 0.34526381, 0.32876229, 0.34967038, 0.32374534]
[0.34526381, 0.32876229, 0.34967038, 0.32374534, 0.34168462]
[0.32876229, 0.34967038, 0.32374534, 0.34168462, 0.27602807]
[0.34967038, 0.32374534, 0.34168462, 0.27602807, 0.2313527 ]
[0.32374534, 0.34168462, 0.27602807, 0.2313527 , 0.20877584]
[0.34168462, 0.27602807, 0.2313527 , 0.20877584, 0.16455034]
[0.27602807, 0.2313527 , 0.20877584, 0.16455034, 0.11714726]
```

Y dt_{futu}

```
[0.05263142, 0.10779498, 0.12263184],
[0.10779498, 0.12263184, 0.12821065],
[0.12263184, 0.12821065, 0.20806335],
[0.12821065, 0.20806335, 0.2518744 ],
[0.20806335, 0.2518744 , 0.28025766],
[0.2518744 , 0.28025766, 0.27699119],
[0.28025766, 0.27699119, 0.30965494],
[0.27699119, 0.30965494, 0.37666627],
[0.30965494, 0.37666627, 0.37879347],
[0.37666627, 0.37879347, 0.36811853]]
```

`TestX[0,:,0]` should predict `TestY[0,:,0]`
`TestX[1,:,0]` should predict `TestY[1,:,0]` etc

LSTMs:

Syntax and some examples

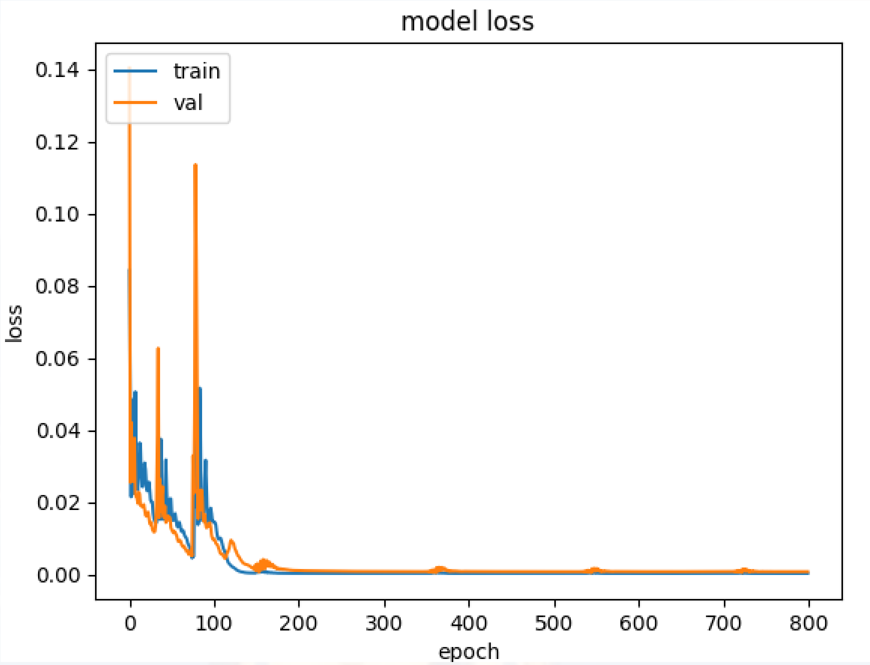
Let us explore LSTMI.ipynb

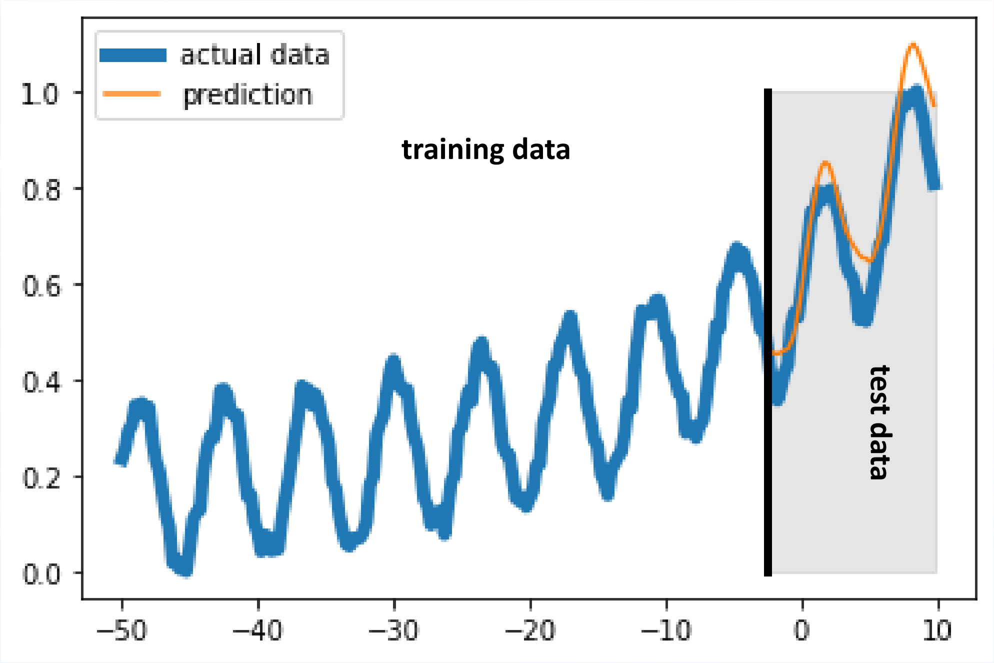
```
n_neurons  = 400
batch_size = 128

model = Sequential()
model.add(LSTM(n_neurons, input_shape = (dt_past, n_features),\
               activation = 'tanh'))
model.add(Dense(dt_futu))

opt = optimizers.Adam()
model.compile(loss = 'mean_squared_error', optimizer = opt)

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 200)               161600

 dense (Dense)               (None, 8)                 1608

=================================================================
Total params: 163208 (637.53 KB)
Trainable params: 163208 (637.53 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

LSTMs: Syntax and some examples

Let us explore `LSTMI.ipynb`

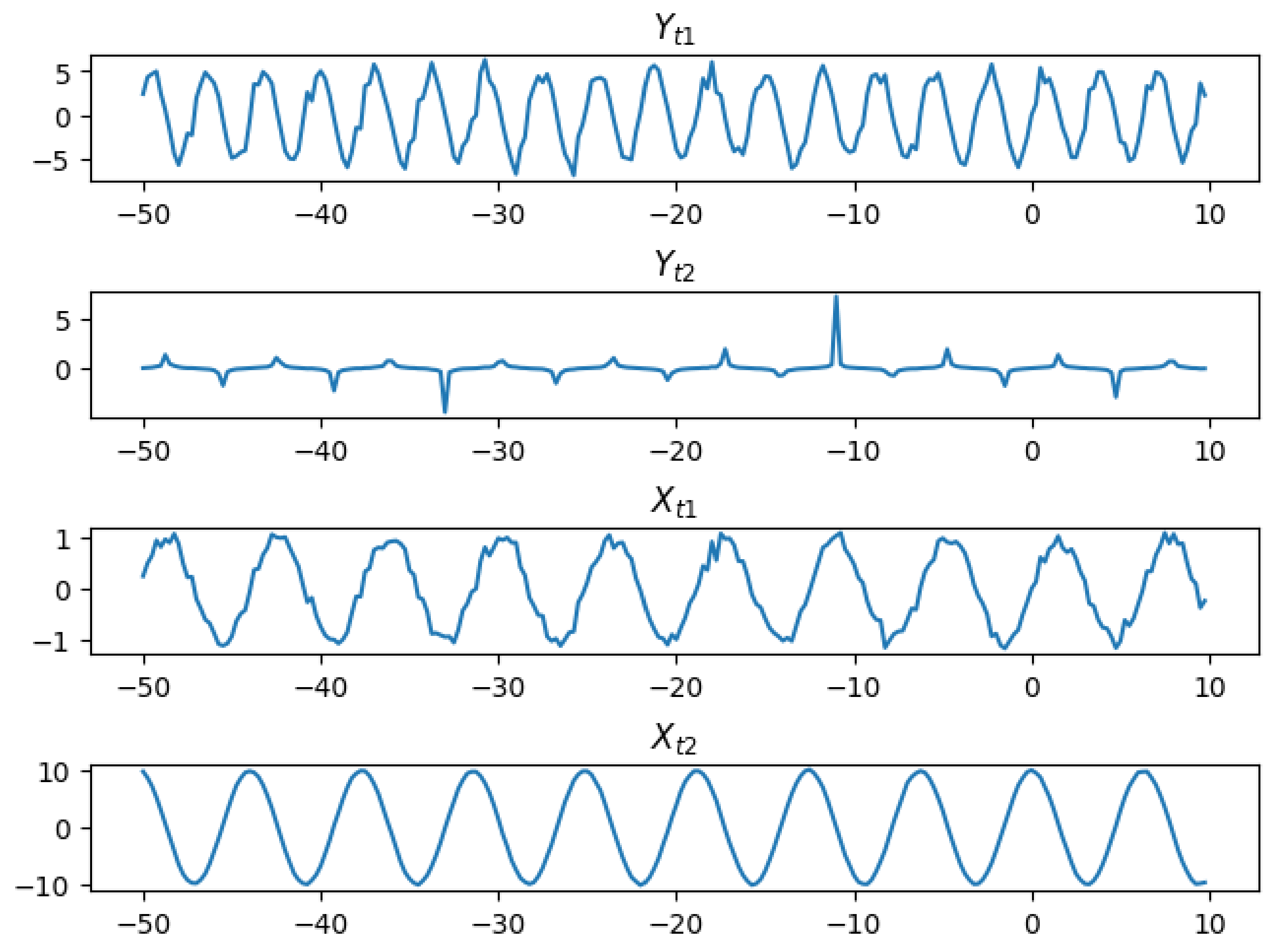




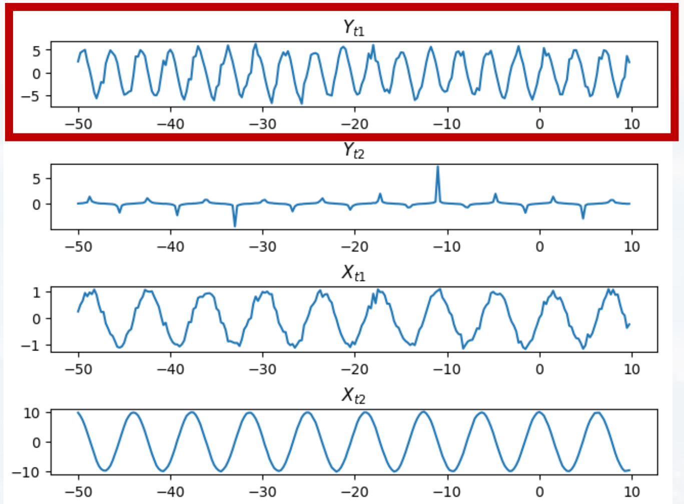
LSTMs:

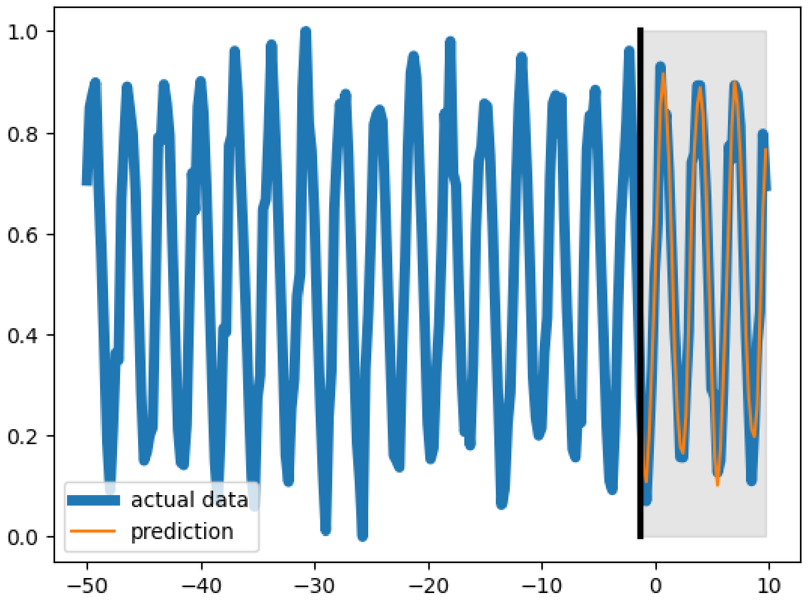
Syntax and some examples

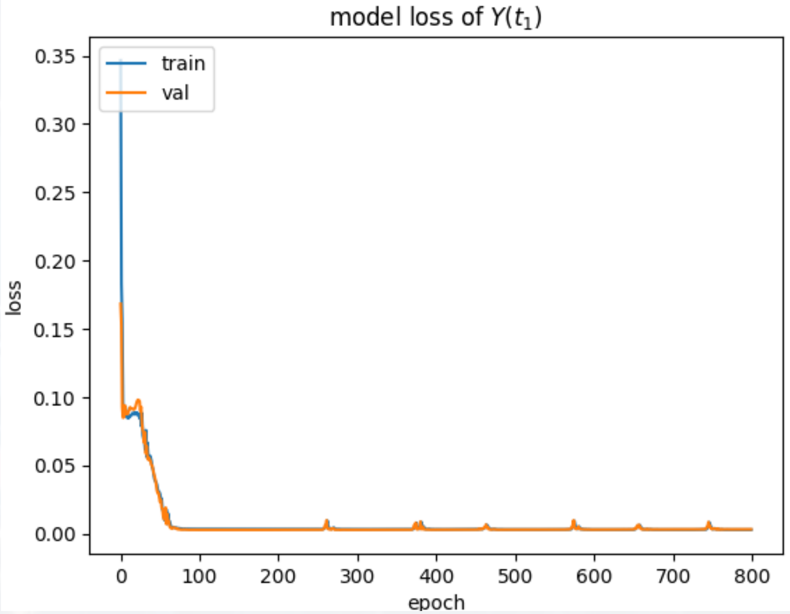
Explore `LSTMII.ipynb` for a multivariate, multi feature time series:



Explore `LSTMII.ipynb` for a multivariate, multi feature time series:

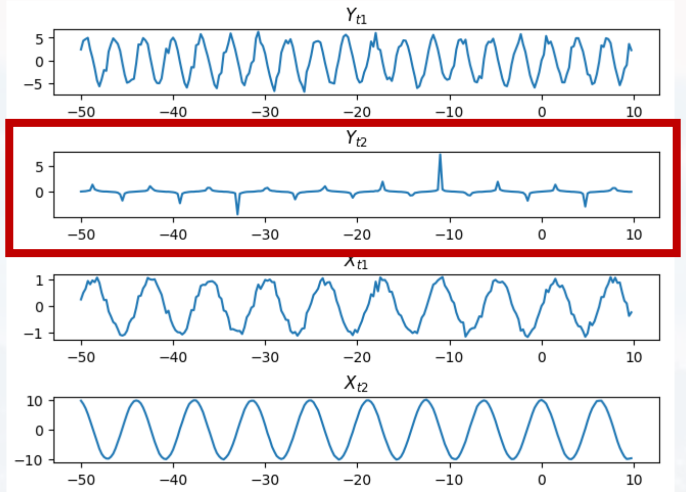


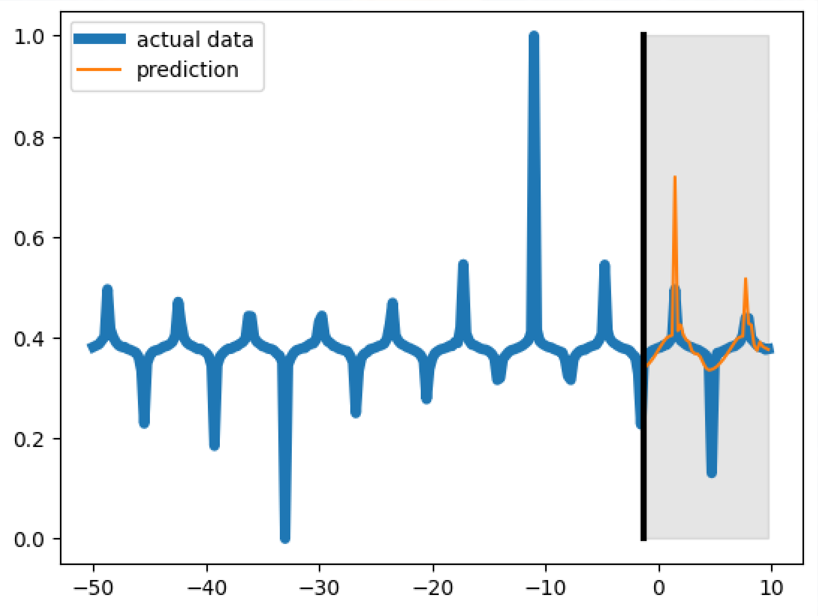


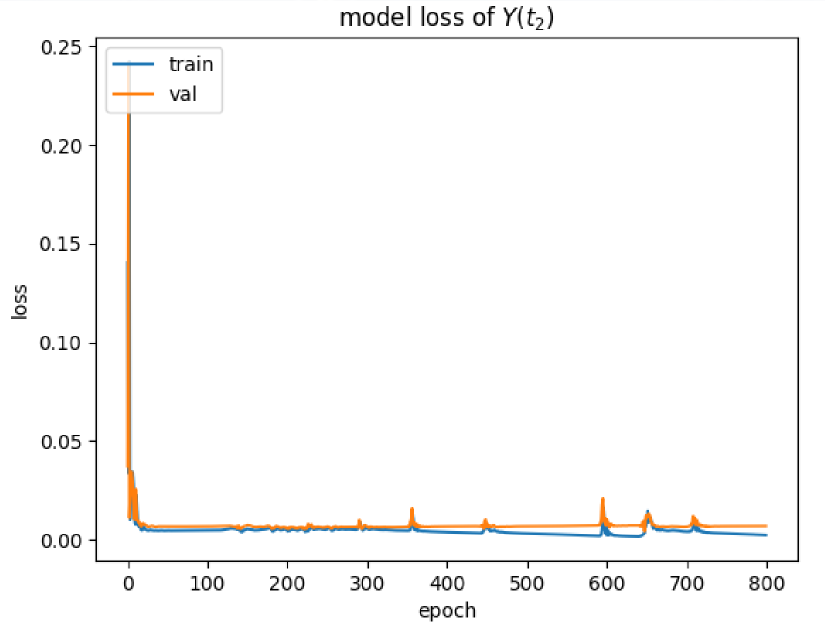


LSTMs: Syntax and some examples

Explore `LSTMII.ipynb` for a multivariate, multi feature time series:







Thank you very much for your attention!

https://www.analyticsvidhya.com