Lecture 12:

RNNs & LSTMs

Markus Hohle

University California, Berkeley

Bayesian Data Analysis and Machine Learning for Physical Sciences

Course Map

https://www.analyticsvidhya.com

Outline

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

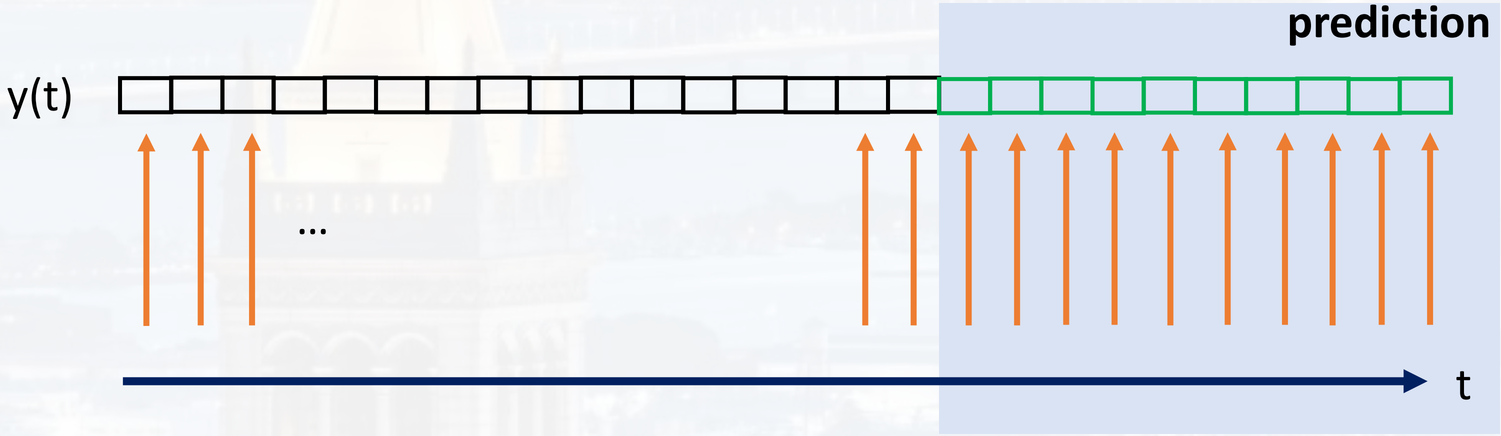
https://www.analyticsvidhya.com

Outline

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

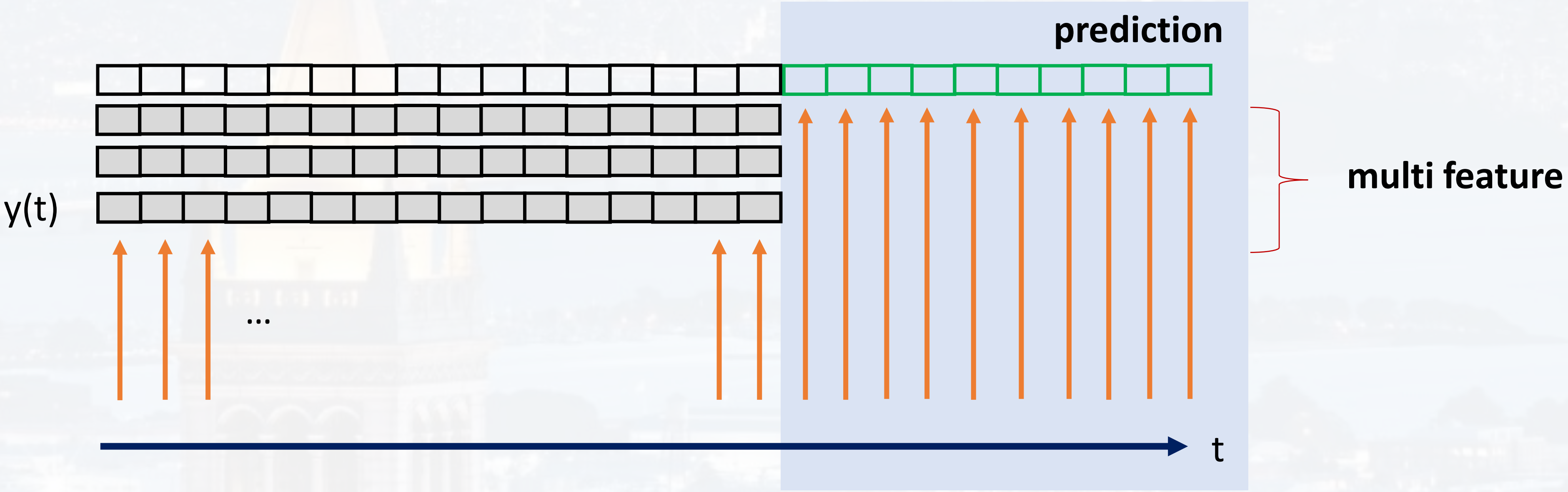
- **R**ecurrent **N**eural **N**etwork

- 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



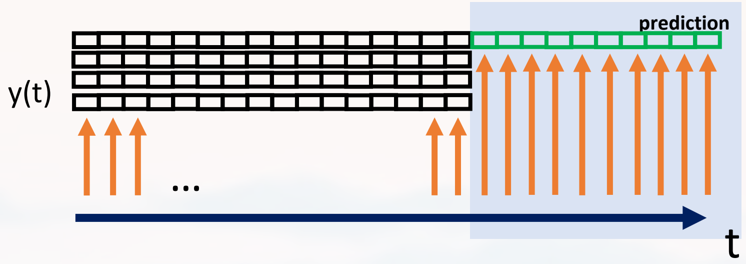
- **R**ecurrent **N**eural **N**etwork

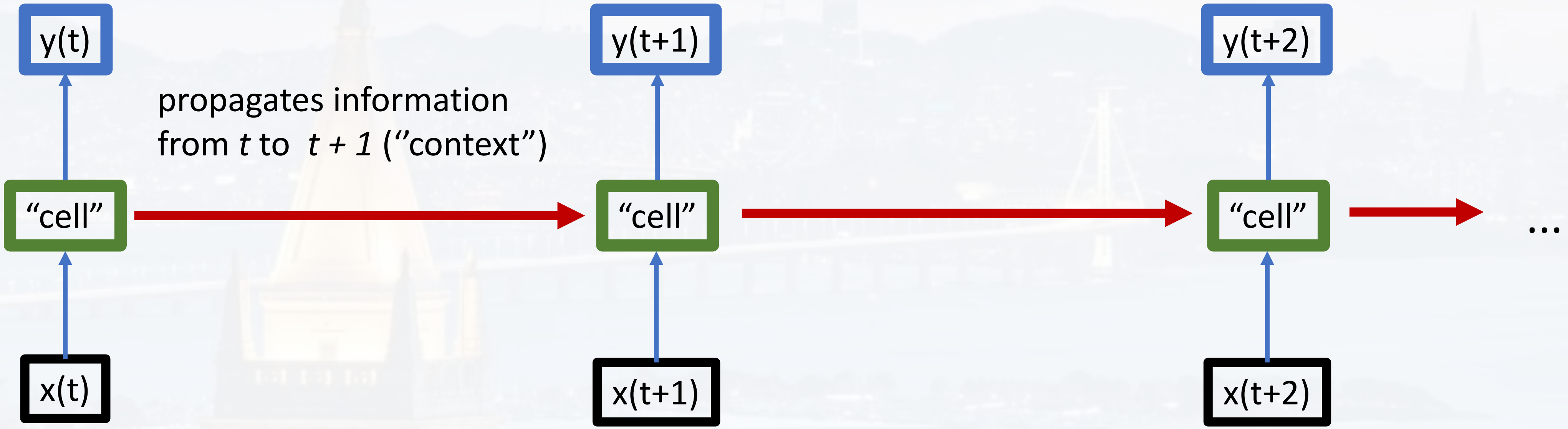
- time series analysis regression (**prediction** and **forecasting**)
- first step towards GenAI
- time series analysis classification
- early speech recognition
- handwriting
- “precursor” of LSTMs
- invented by **Shun'ichi Amari** in 1972



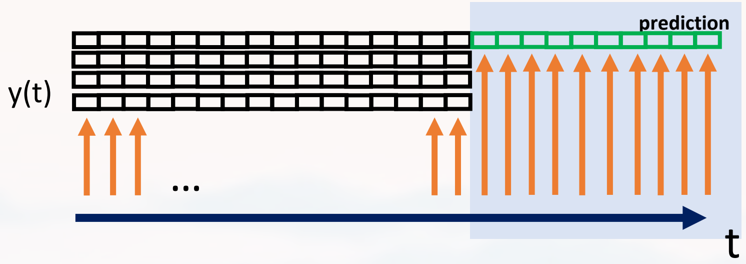
RNNs & LSTMs:

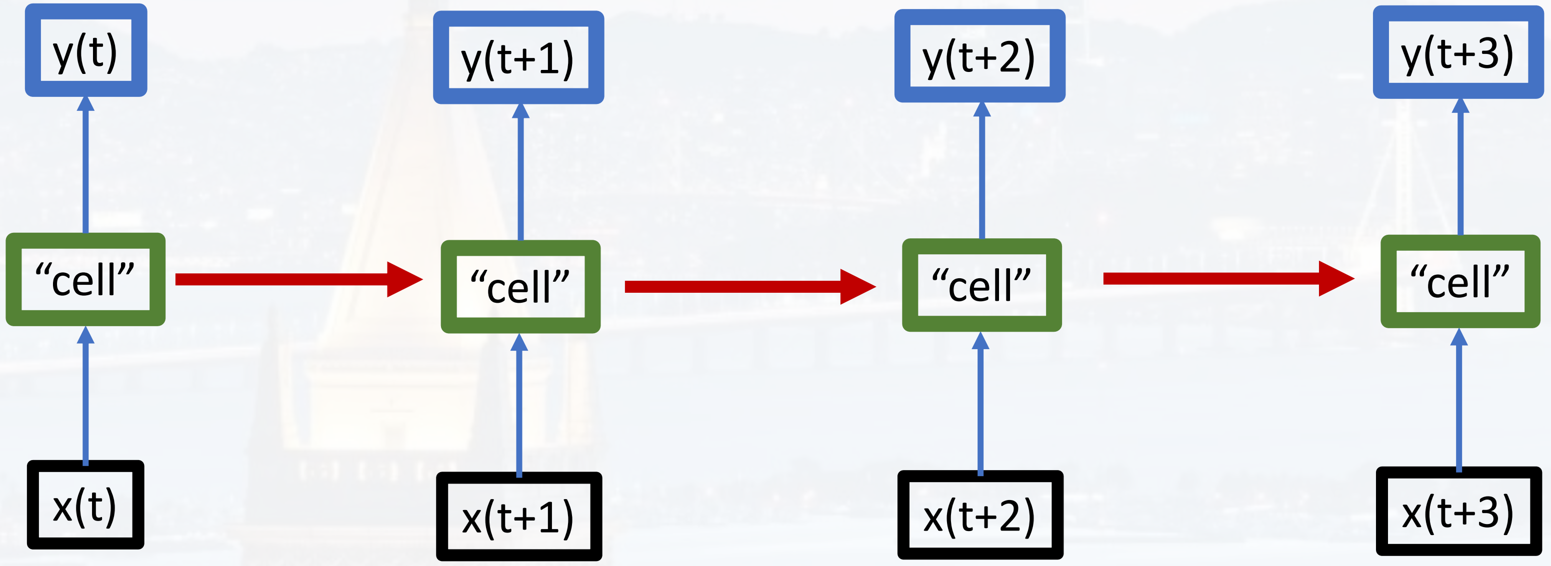
Idea and classic RNNs



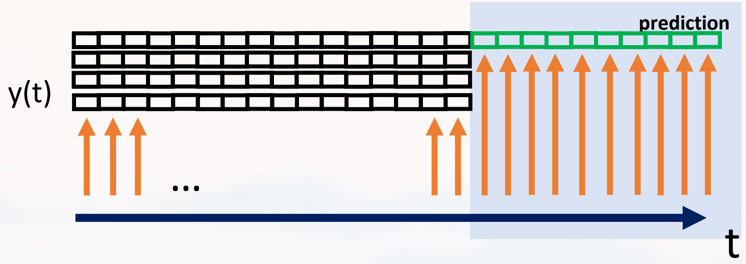


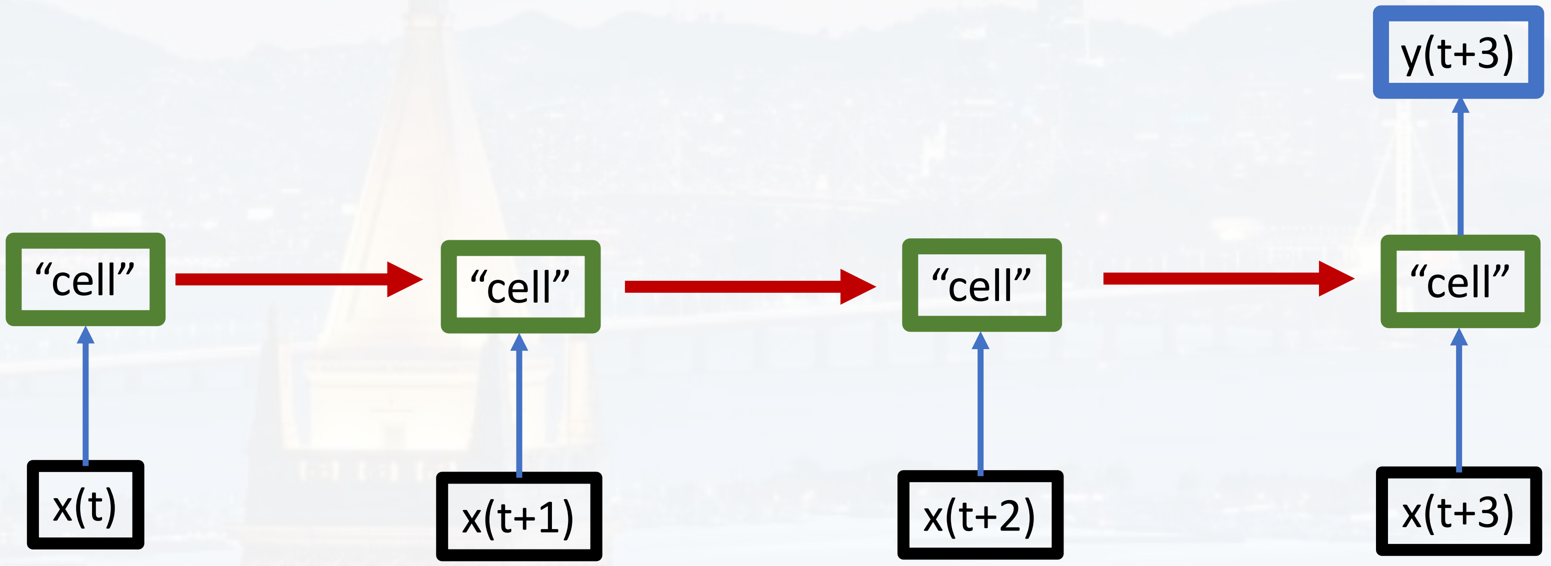
“many to many”


prediction
y(t)
...
t

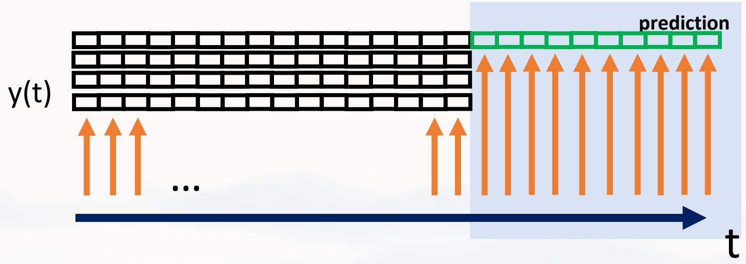

y(t)
y(t+1)
y(t+2)
y(t+3)
“cell”
“cell”
“cell”
“cell”
x(t)
x(t+1)
x(t+2)
x(t+3)

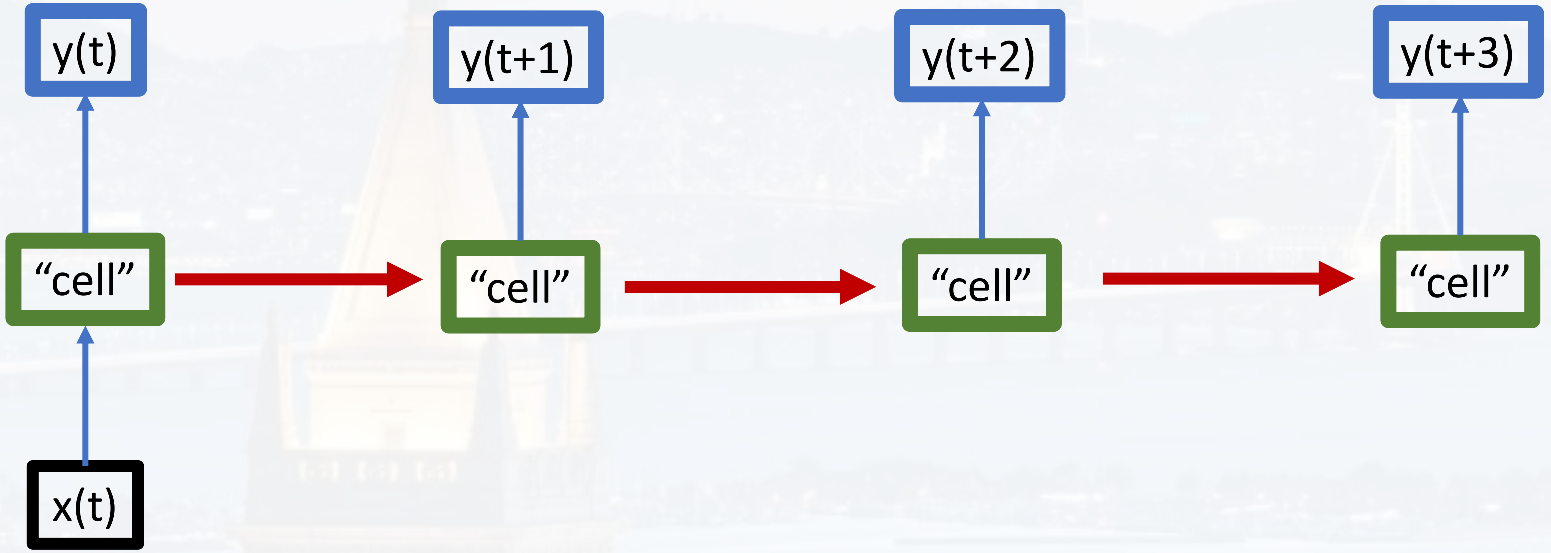
"many to one"
(classification)



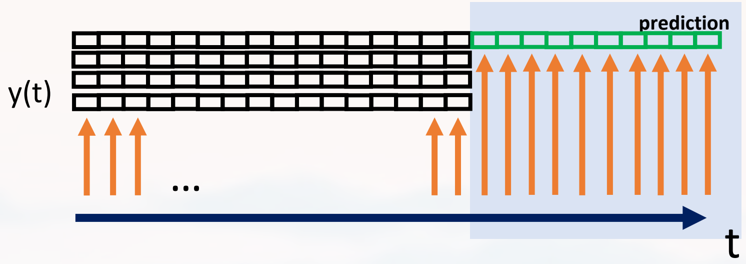


"one to many"
(image capturing)



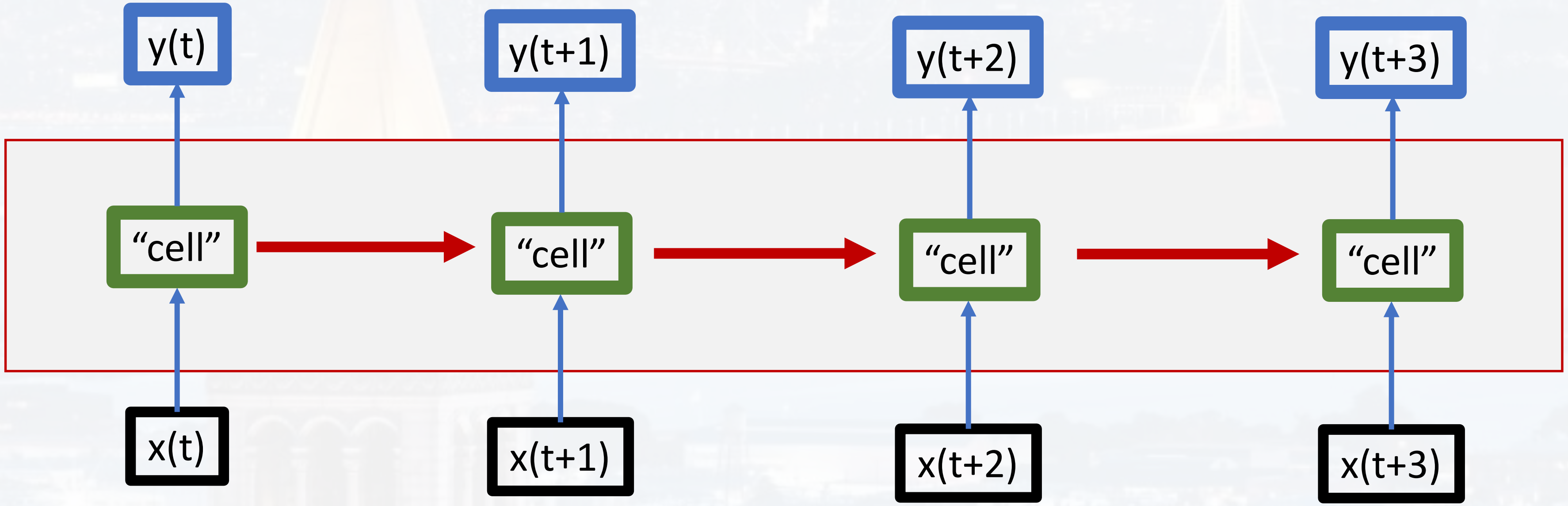


Applying the **identical** cell **recursively**!



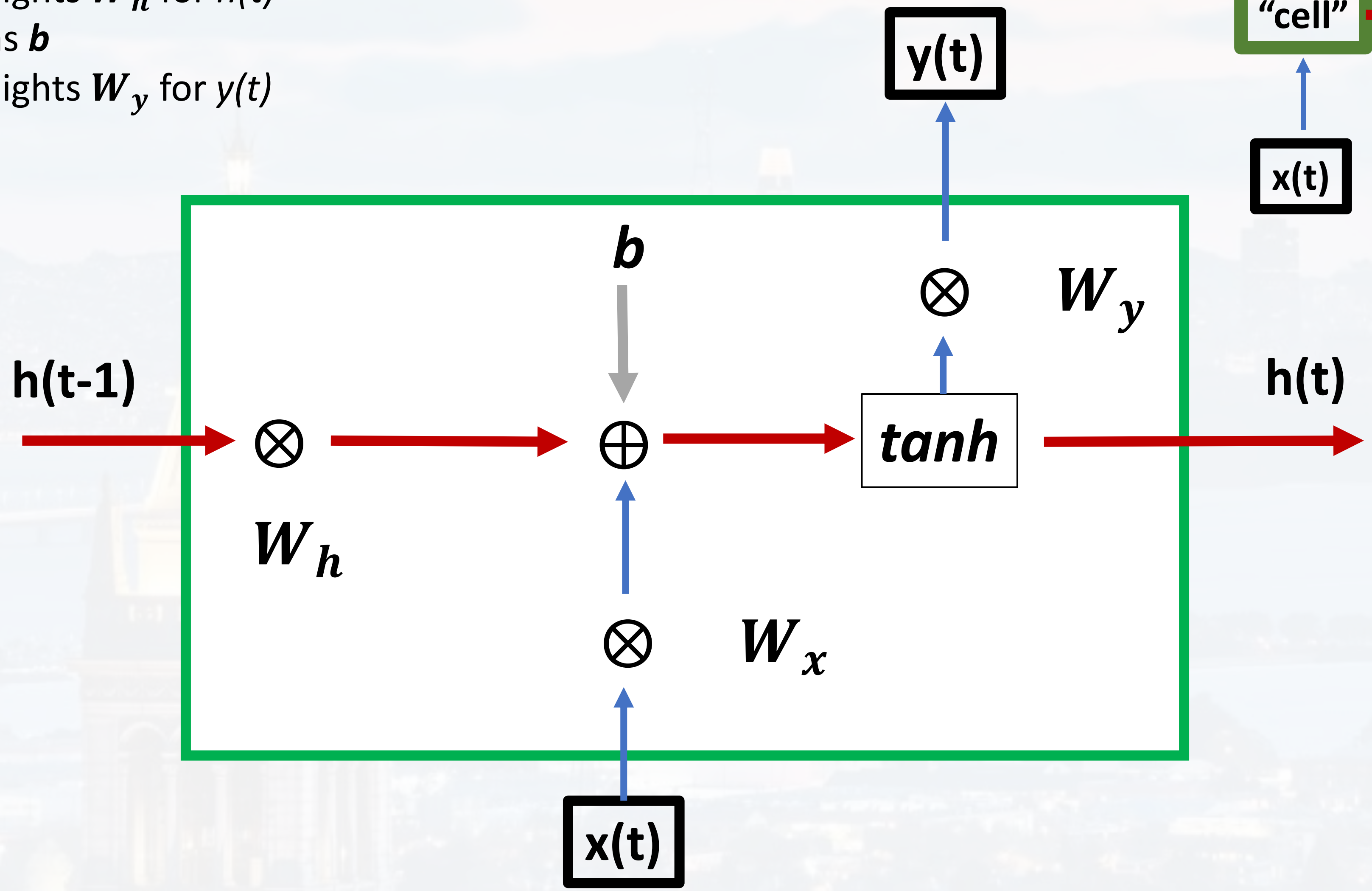
→ easy to implement
→ direction (arrow of time, see later)

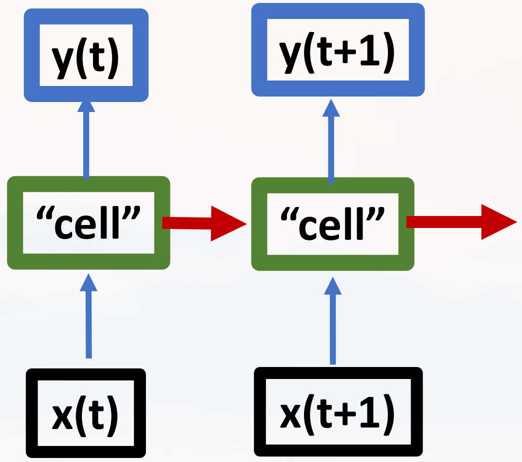
→ exploding/vanishing gradients
→ classic RNN has a "short memory"



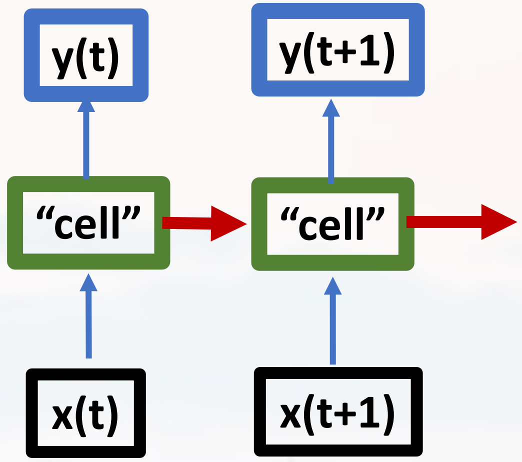
RNNs & LSTMs:

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

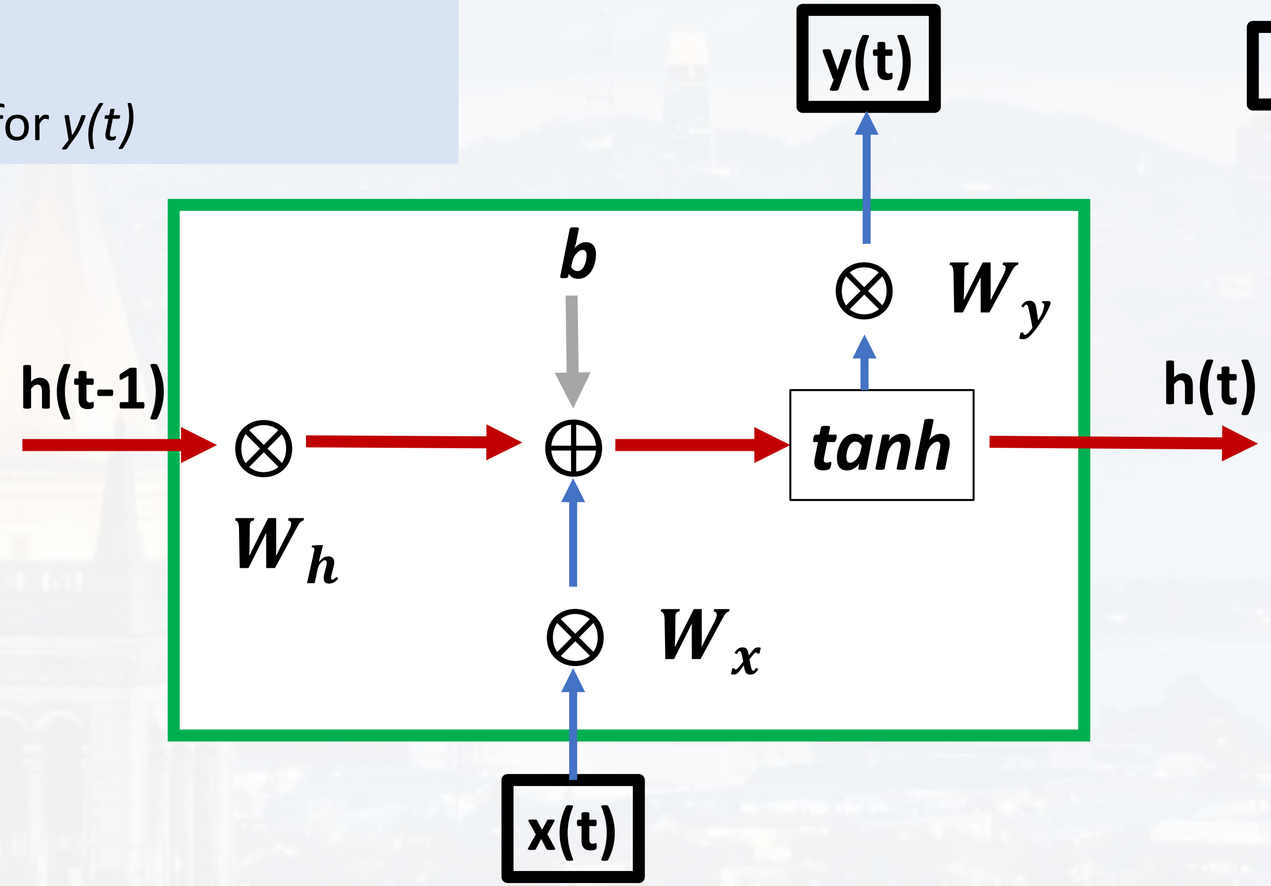




- **state vector** *h(t)*, i. e. the *"memory"* the system has along time



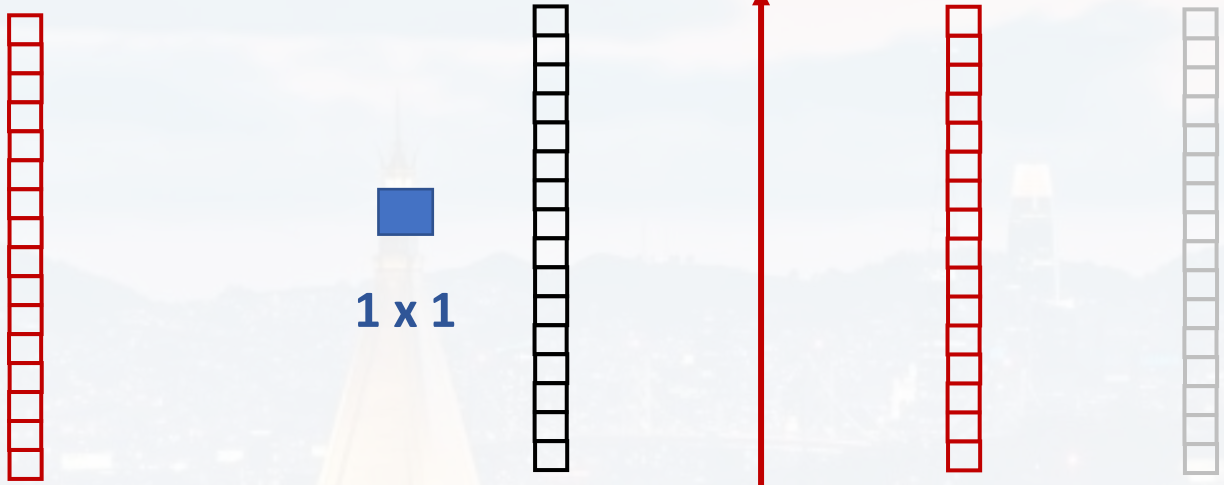
- weights $\boldsymbol{W_x}$ for *x(t)* **learnable**
- weights $\boldsymbol{W_h}$ for *h(t)*
- bias $\boldsymbol{b}$
- weights $\boldsymbol{W_y}$ for *y(t)*



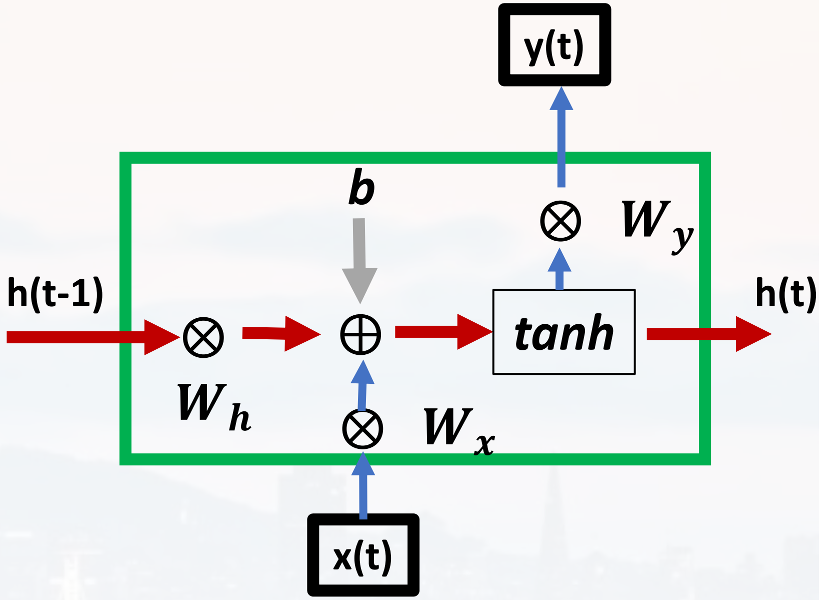
RNNs & LSTMs: Idea and classic RNNs

for **single timepoint, single feature**: shape of $x(t) = 1\ x\ 1$

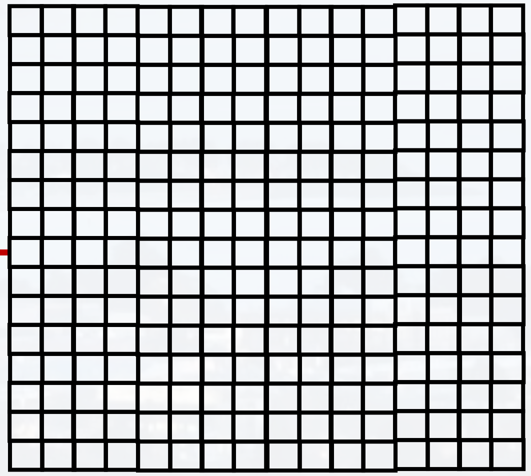
$$h(t) = \tanh[\, x(t) * W_x + \boxed{W_h} * h(t-1) + b\,]$$



n x 1 n x 1 n x 1 n x 1



n: number of states/ neurons

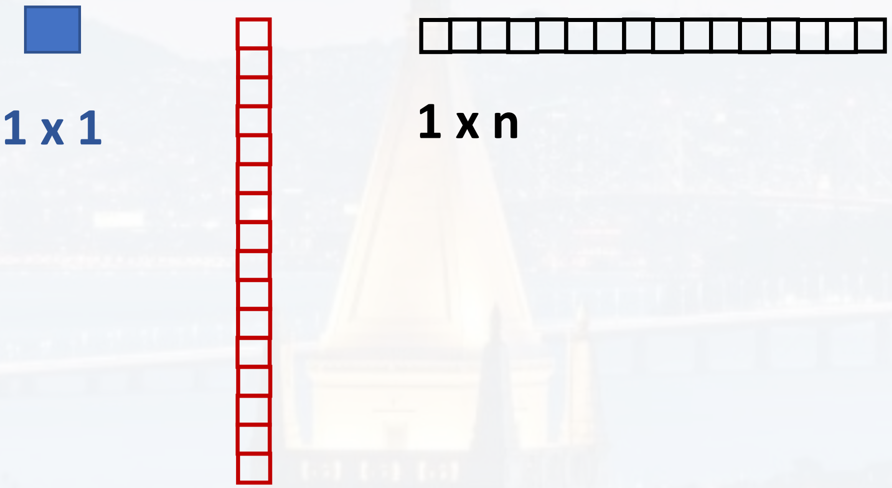


$$y(t) = h(t) * W_y$$

for **single timepoint, single feature**: shape of $x(t) = 1\ x\ 1$

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

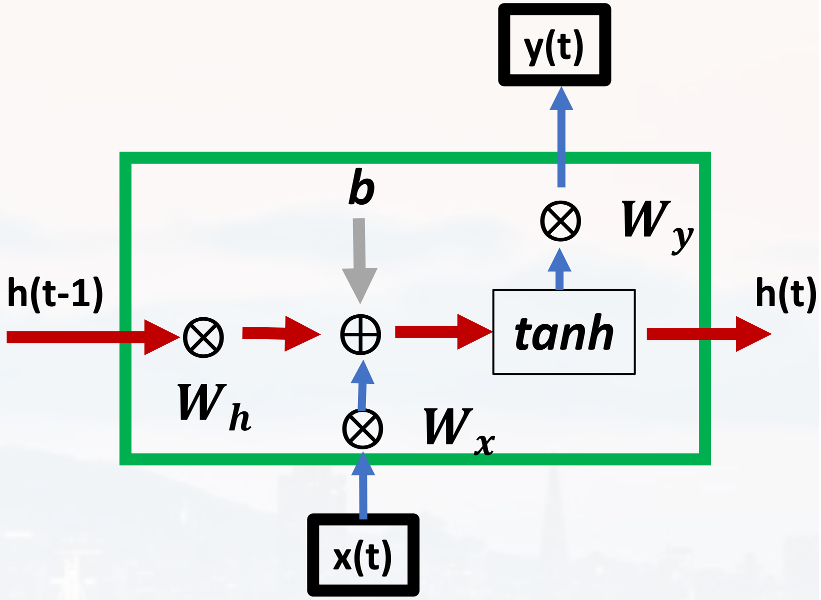
$$y(t) = h(t) * W_y$$



1 x 1

n x 1

1 x n

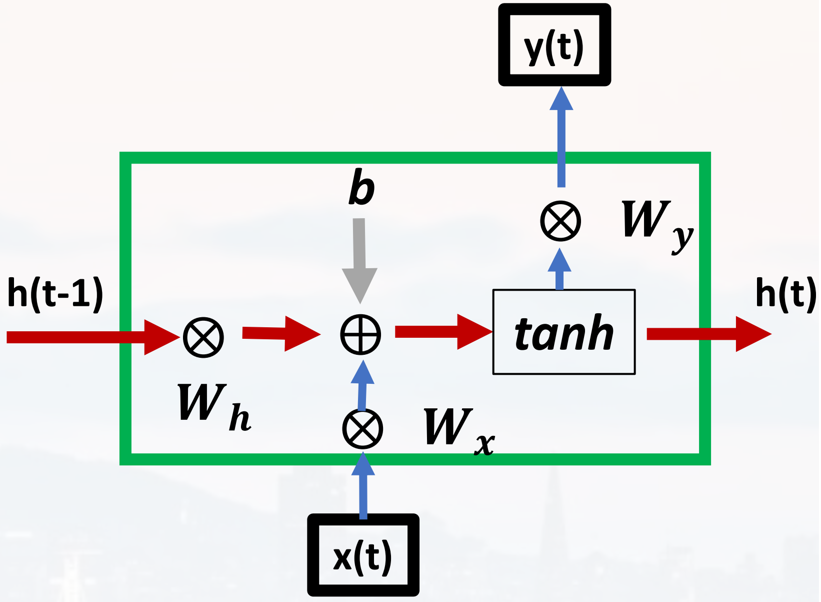


n: number of states/ neurons

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

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

usually, $x(t)$ comes in **batches of size B and of length T** and has F **features** (see also later)



for **each time point** t:

$$x(t) * W_x^T + h(t-1) * W_h + b$$

n: number of states/ neurons

shape: $\underbrace{(B \times 1 \times F) * (F \times n)^T}_{B \times 1 \times n} + \underbrace{(B \times 1 \times n) * (n \times n)}_{B \times 1 \times n} + B \times 1 \times n$

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

shape: $(B \times 1 \times F) \quad (B \times 1 \times n) * (F \times n)^T$

https://www.analyticsvidhya.com

Outline

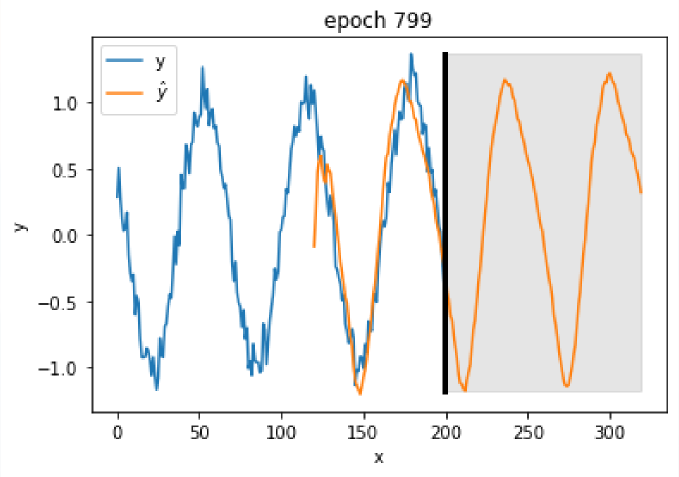
- Idea and classic RNNs
- **LSTMs**
- *B*ack*P*ropagation *T*hrough *T*ime (BPTT)
- Syntax and some examples

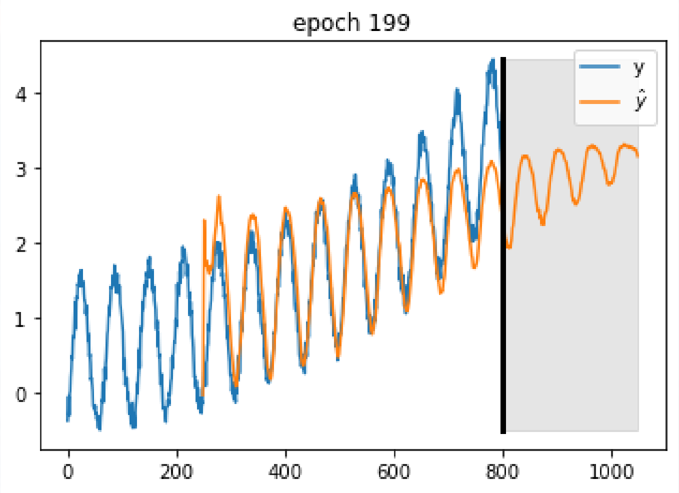
- **L**ong- **S**hort **T**erm **M**emory

new:

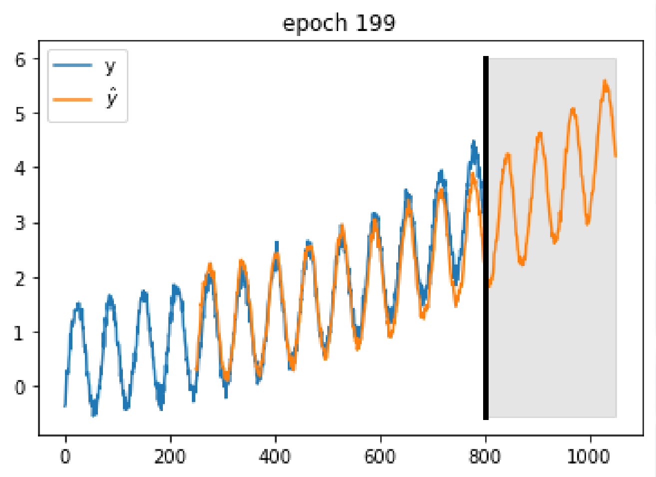
- **long-term** and **short-term** memory
- dealing with vanishing/exploding gradient
- invented 1997 by Sepp Hochreiter und Jürgen Schmidhuber

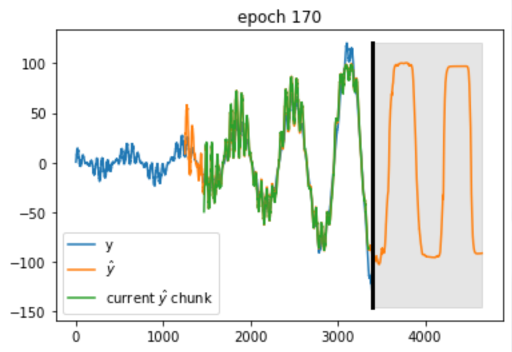
classical RNN



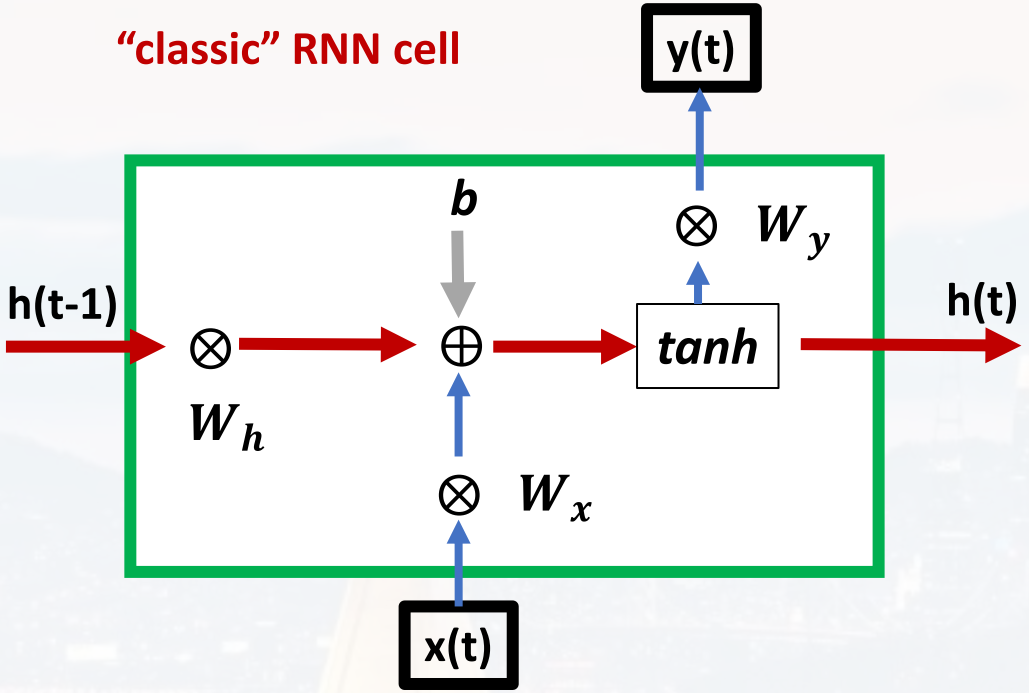


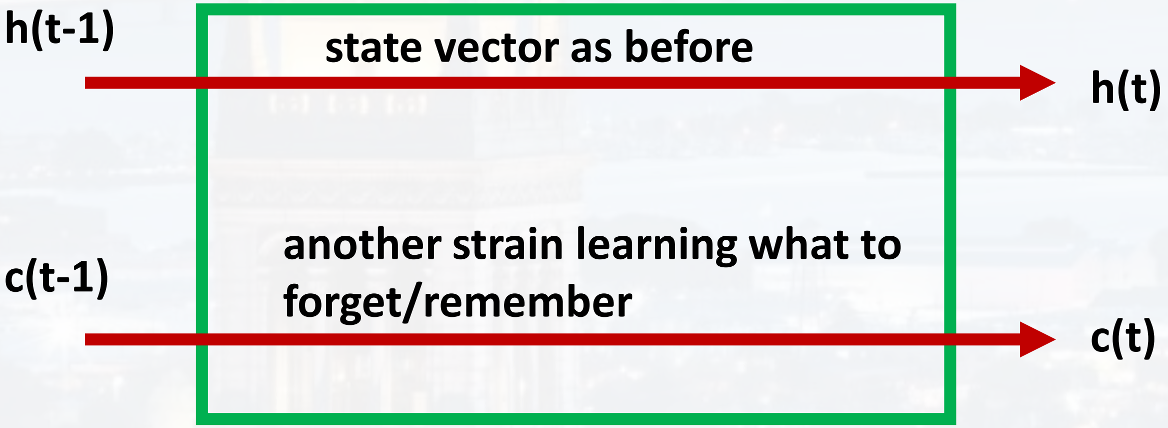
LSTM



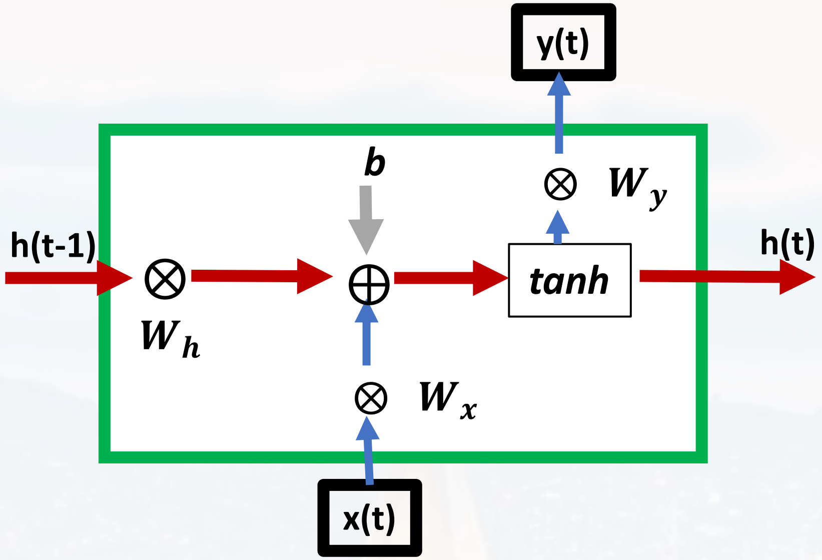


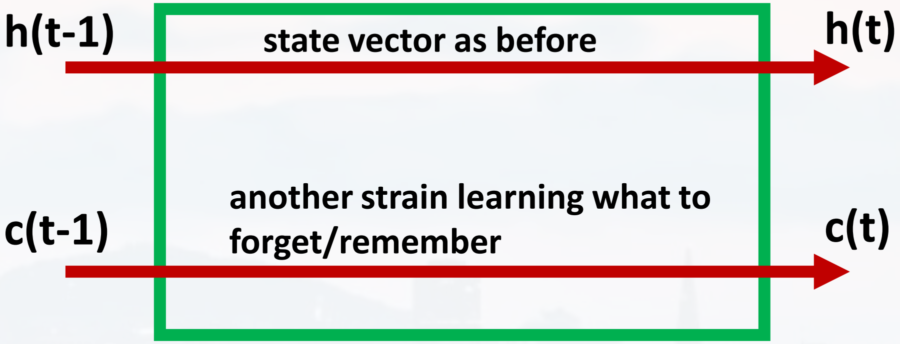
(adding more noise)

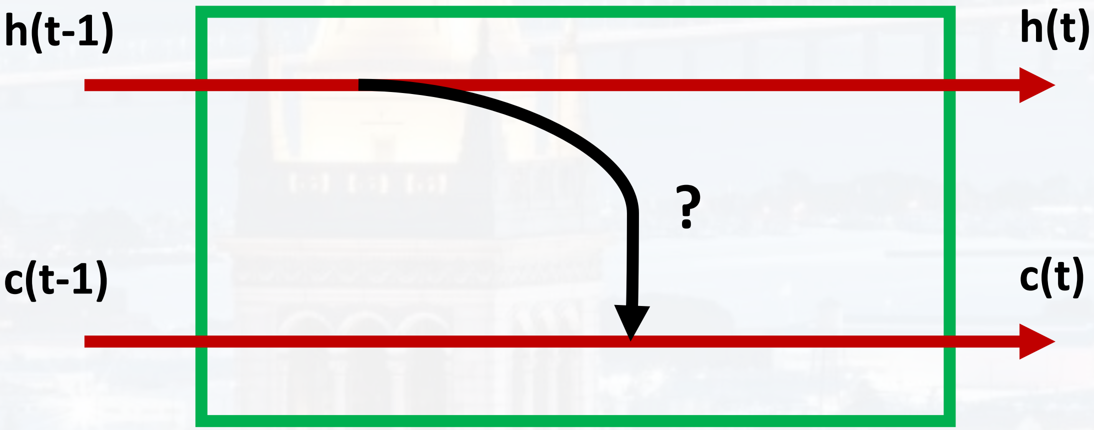

"classic" RNN cell
y(t)
b
W_y
h(t-1)
h(t)
tanh
W_h
W_x
x(t)


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

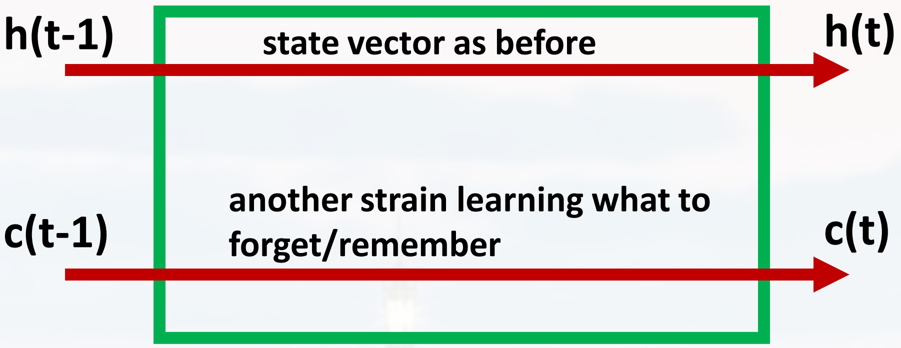
"classic" RNN cell

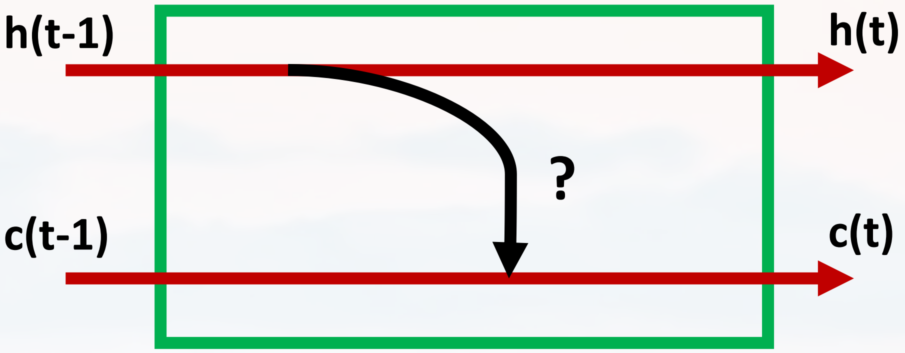




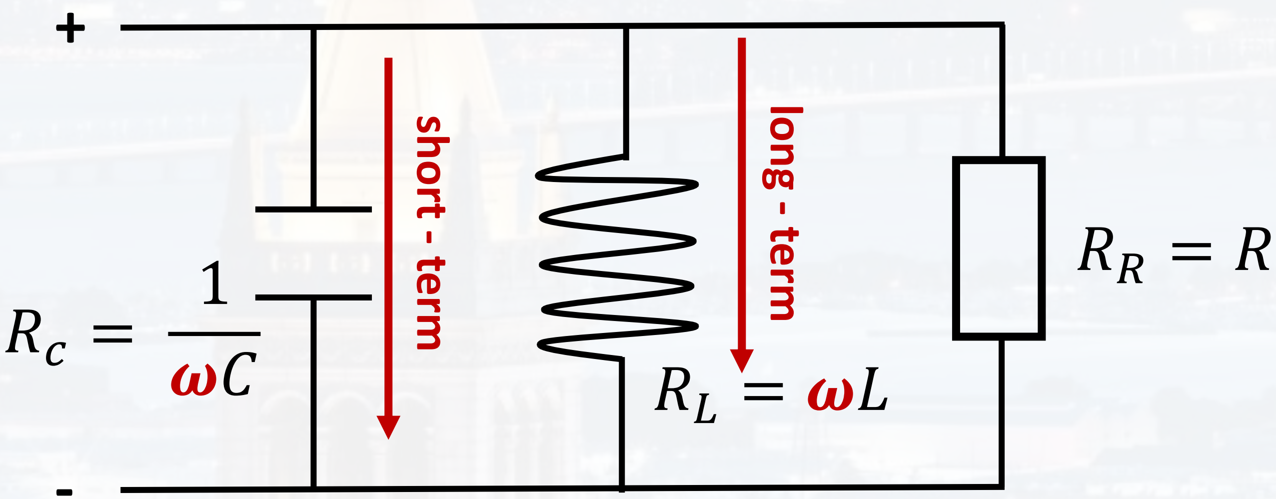


RNNs & LSTMs:





electrical circuits:



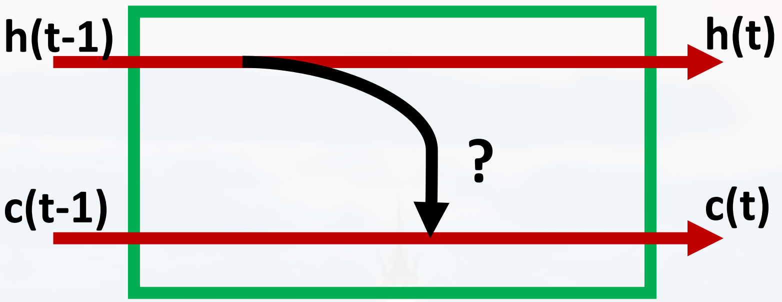
AC: $I(t) = I_0\, e^{i(\omega t + \varphi)}$

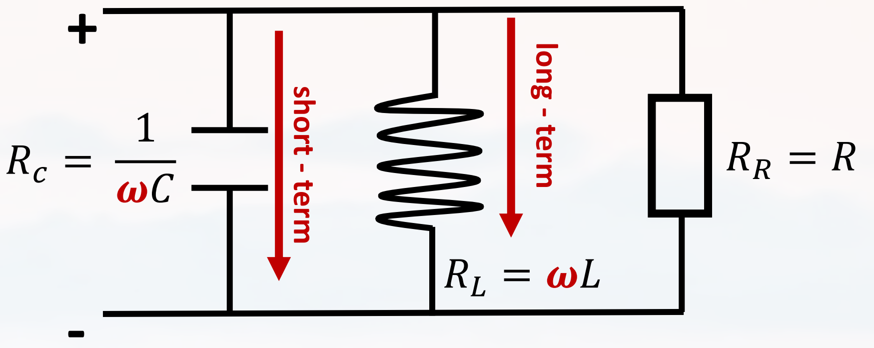
R_c: passes **short** -term changes

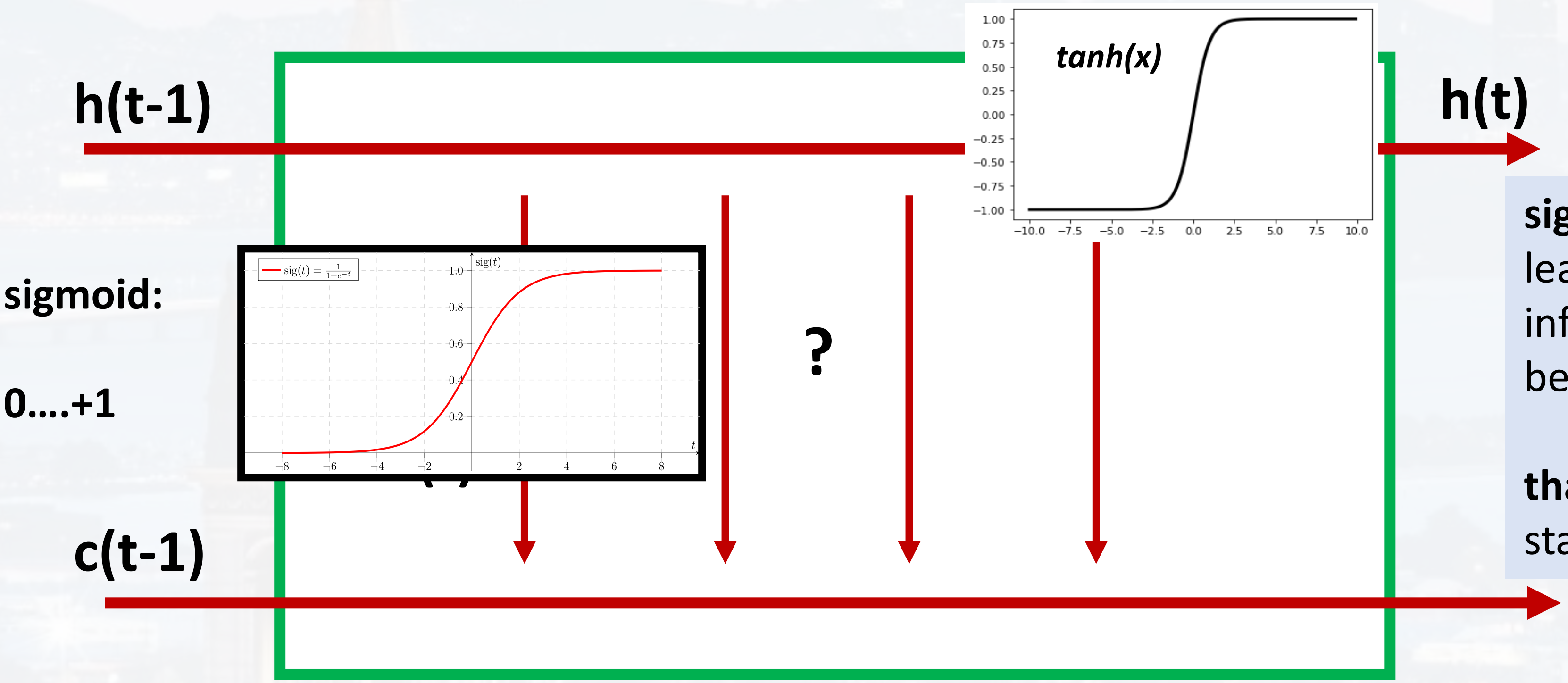
R_L: passes **long** -term changes

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

RNNs & LSTMs:





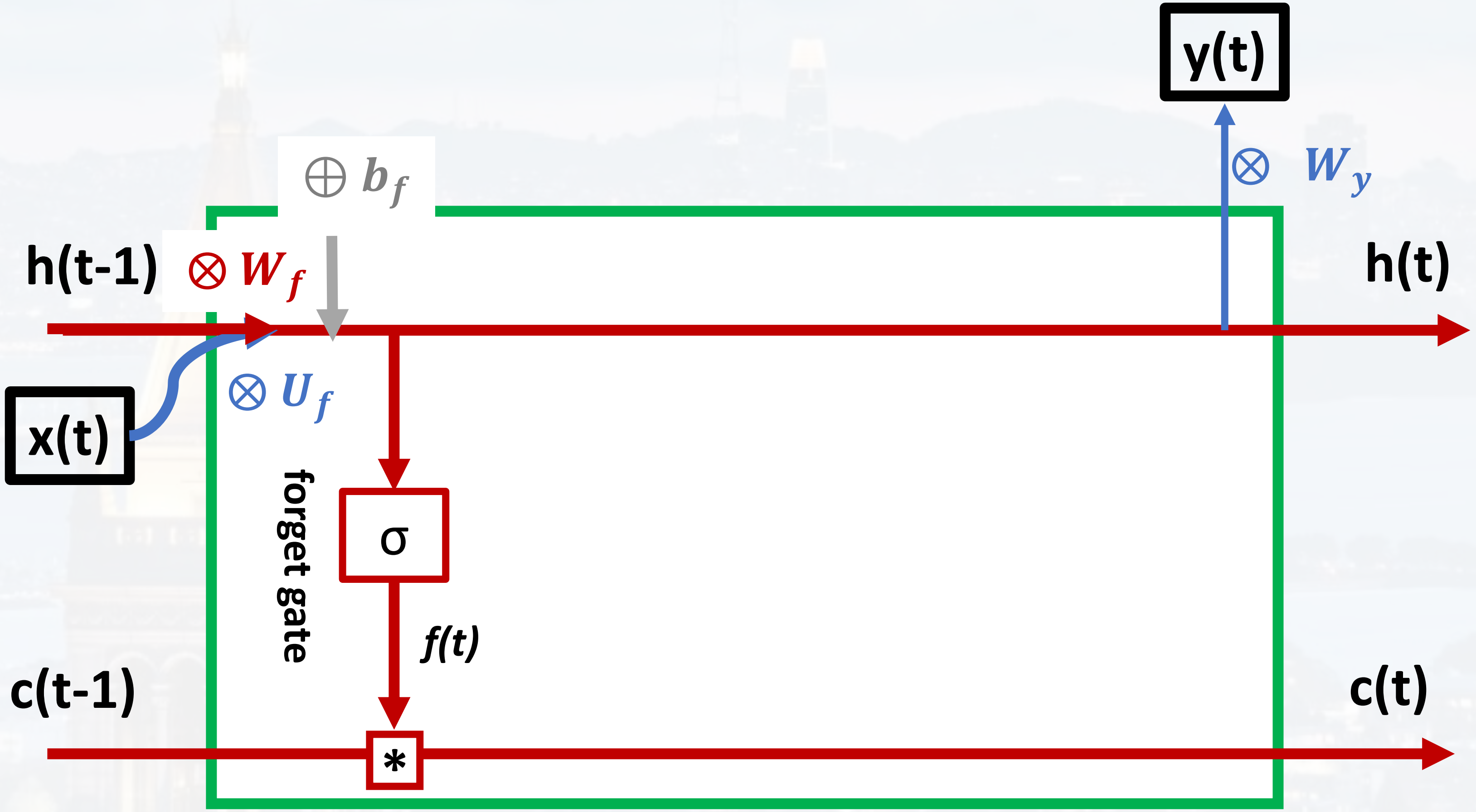


sigmoid:
learning whether information has to be passed on

thanh:
state vector as for RNN

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

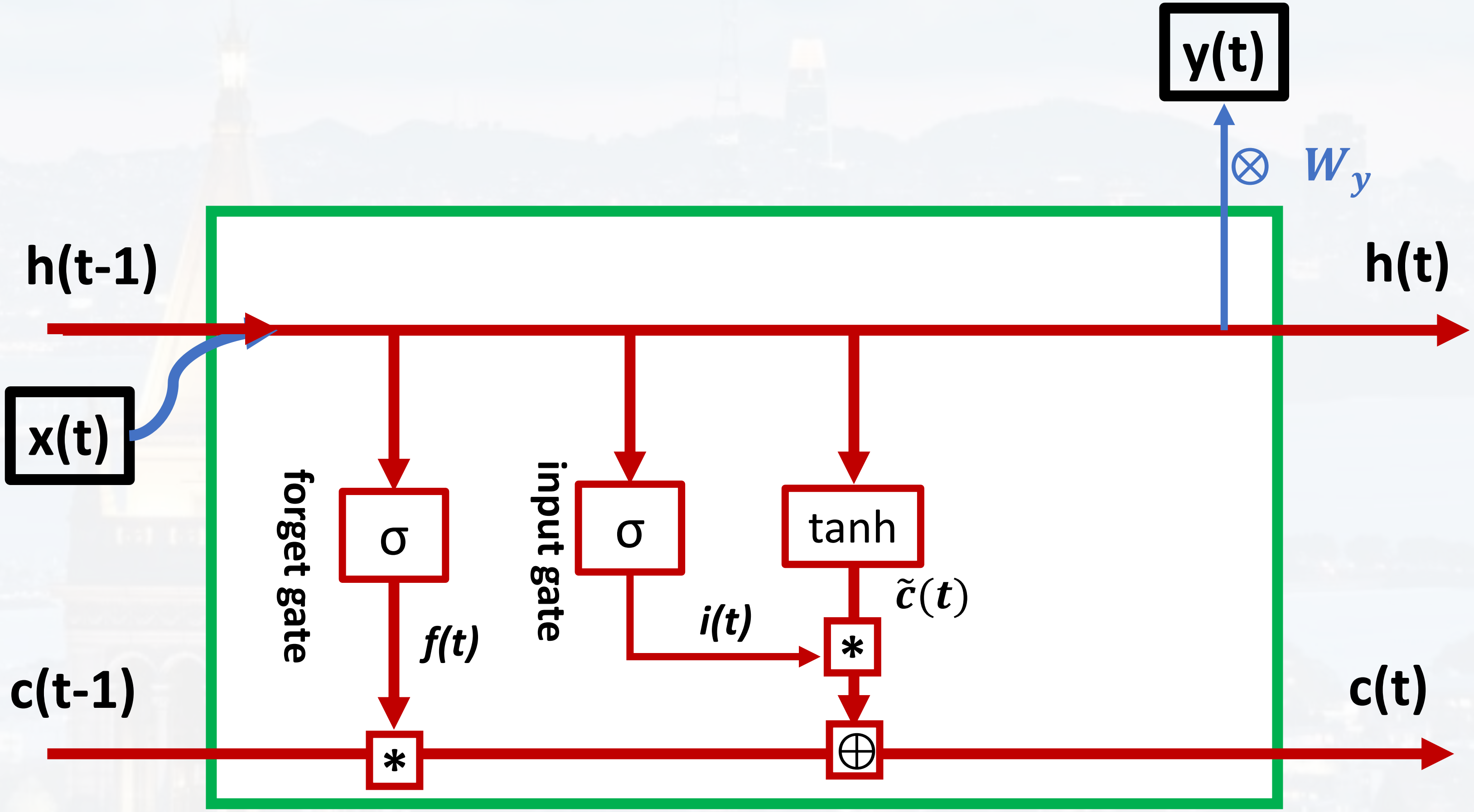
* element – wise multiplication



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

* element – wise multiplication

$$i(t) = \sigma\left(U_i \otimes x(t) + W_i \otimes h(t-1) + b_i\right) \qquad \tilde{c}(t) = tanh\left(U_g \otimes x(t) + W_g \otimes h(t-1) + b_g\right)$$



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

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

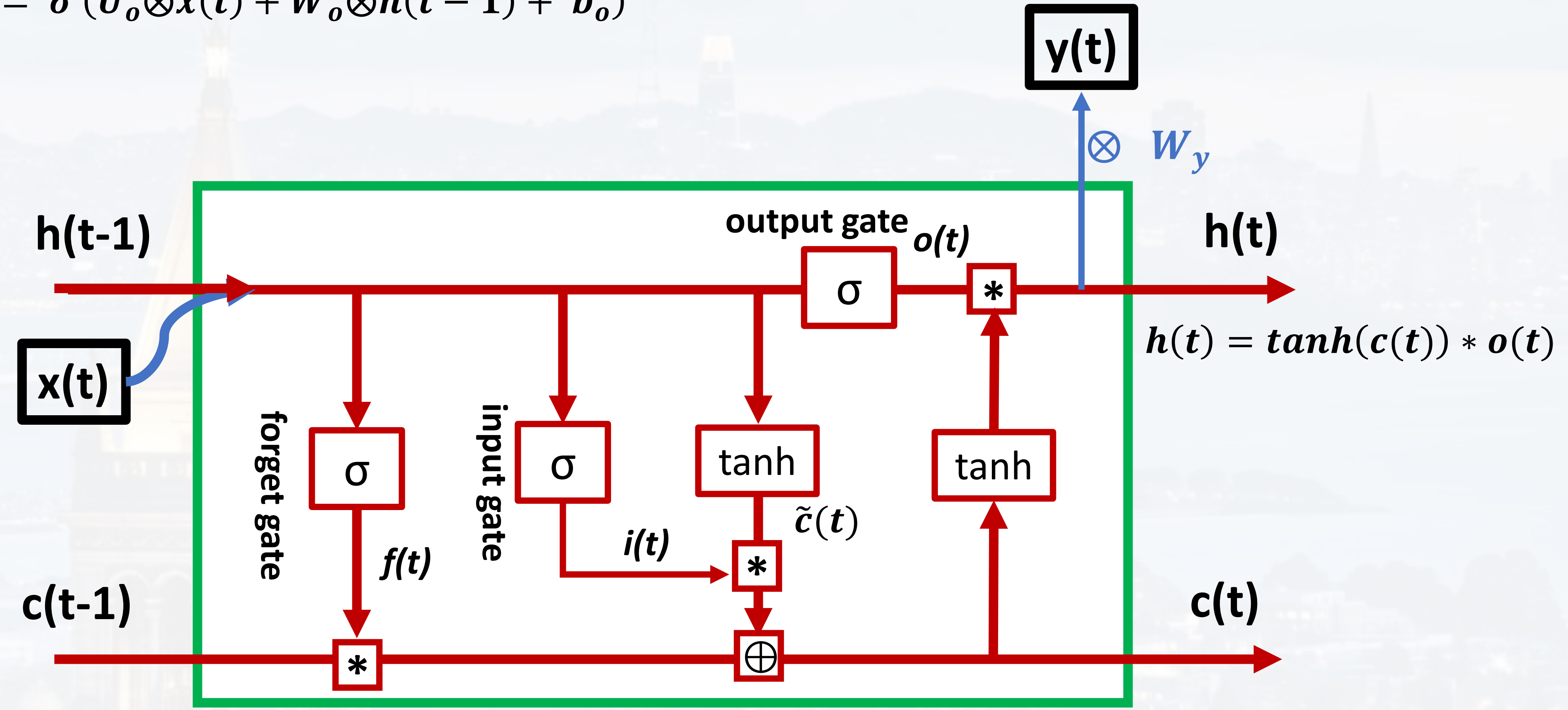
$*$ element – wise multiplication

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

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

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

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



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

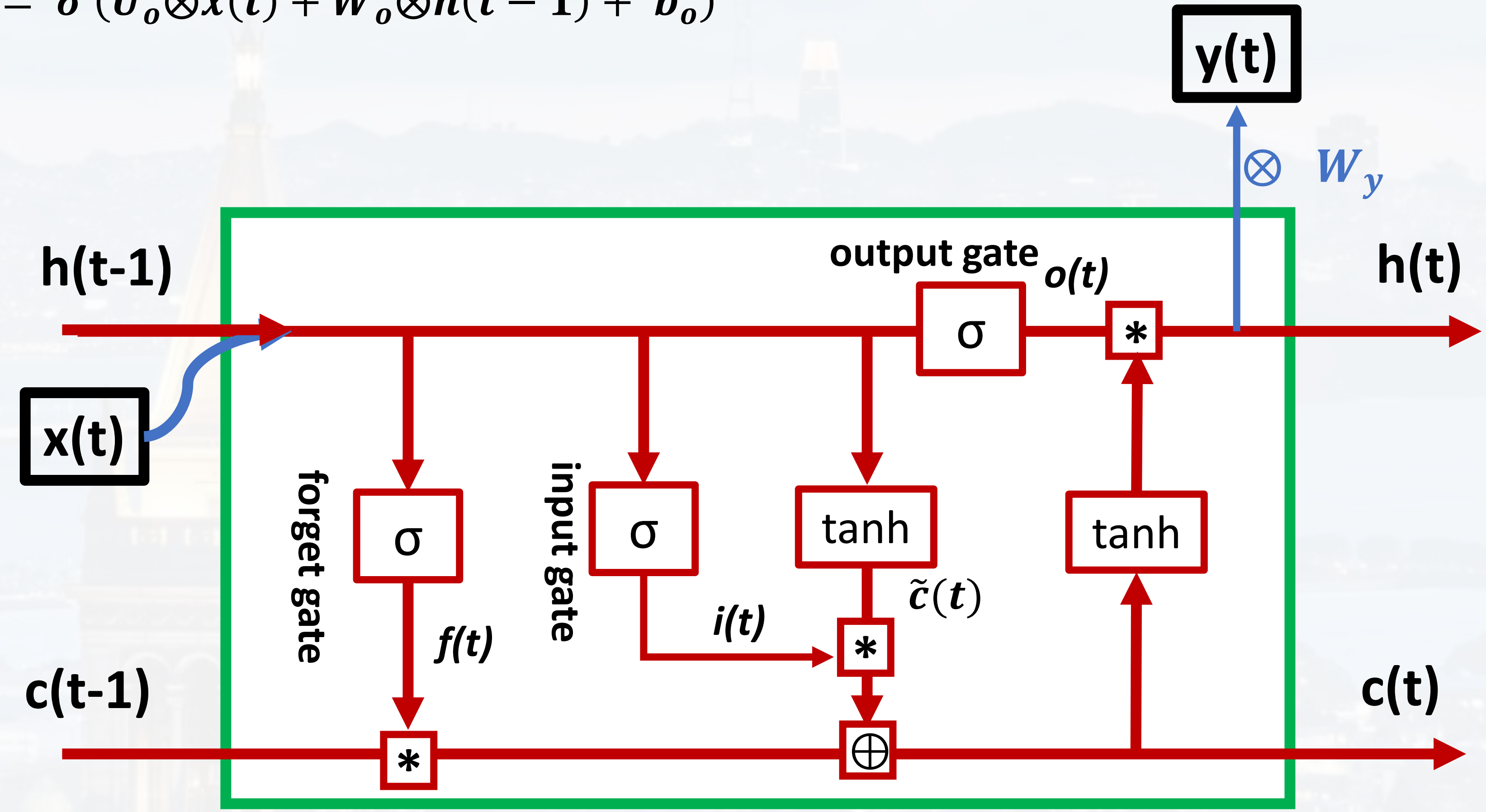
$*$ element – wise multiplication

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

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

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

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



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

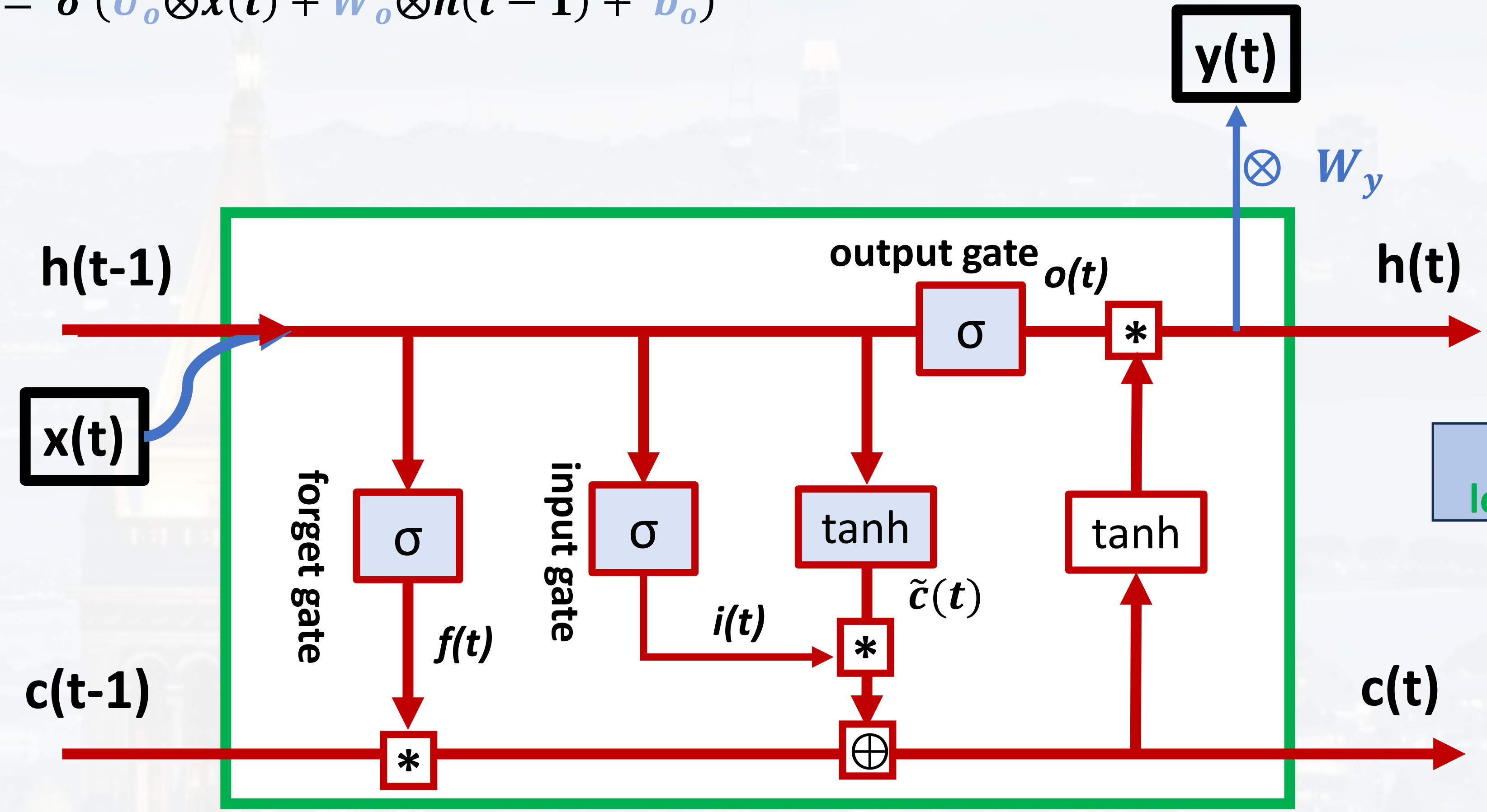
$$i(t) = \sigma\left(U_i \otimes x(t) + W_i \otimes h(t-1) + b_i\right)$$

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

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

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

* element – wise multiplication

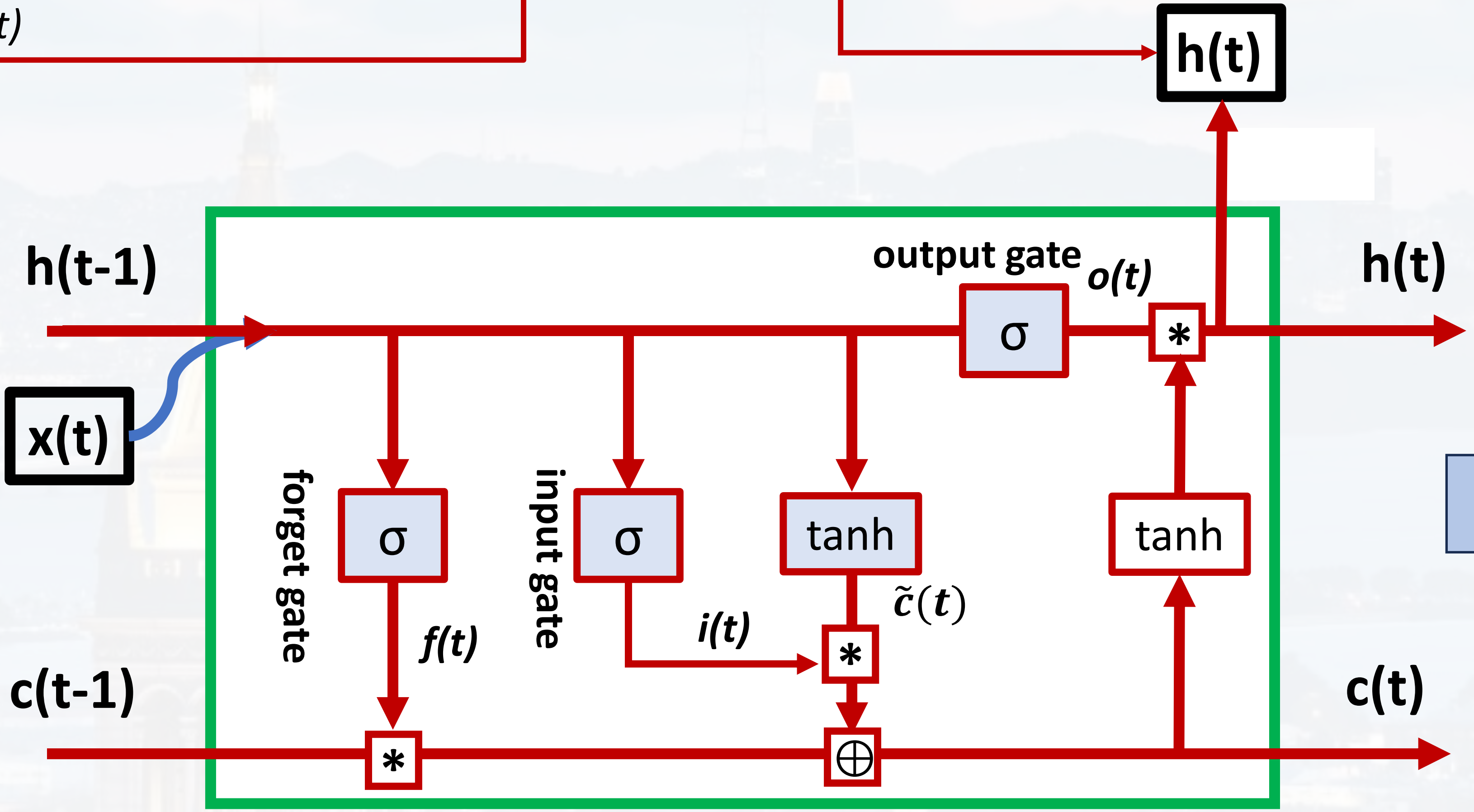


contain learnables!

There is one more thing:

* element – wise multiplication

we will add a **dense layer** instead of $\boldsymbol{W}_y$ at the end to convert *h(t)* to *y(t)*



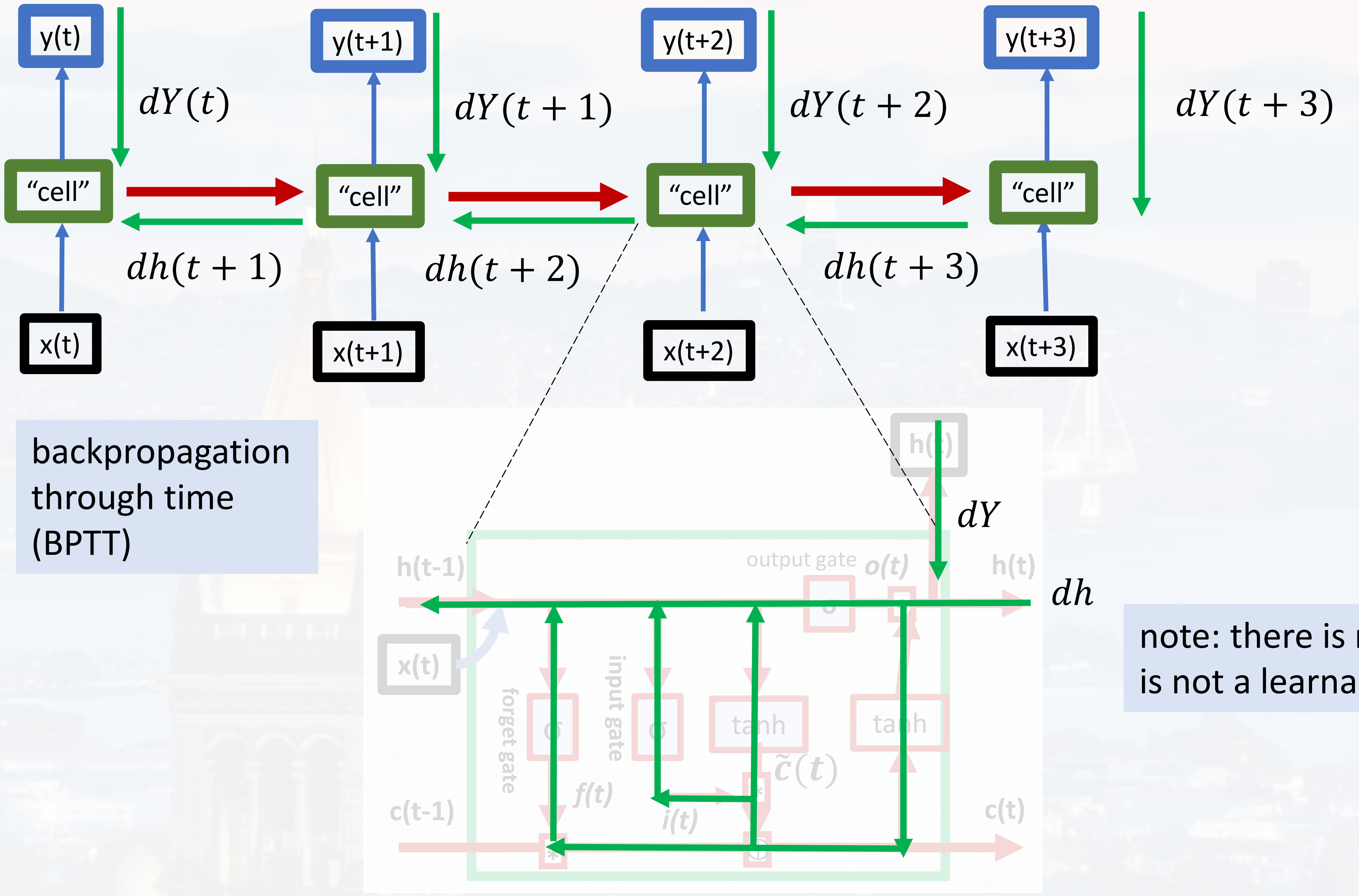
contain learnables!

https://www.analyticsvidhya.com

<u>Outline</u>

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

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

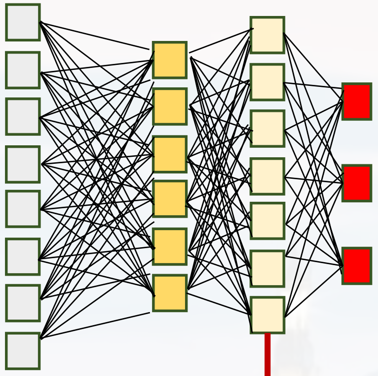


backpropagation through time (BPTT)

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

RNNs & LSTMs:

dense

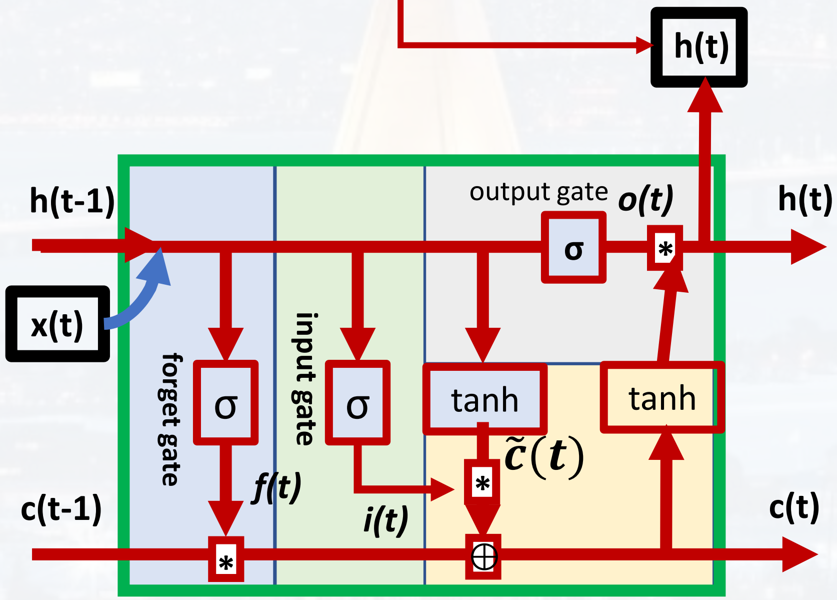


see module 10

$$L = \frac{1}{2}\sum_{t=1}^{T} \boxed{[\,\hat{y}(t) - y(t)\,]^2}$$

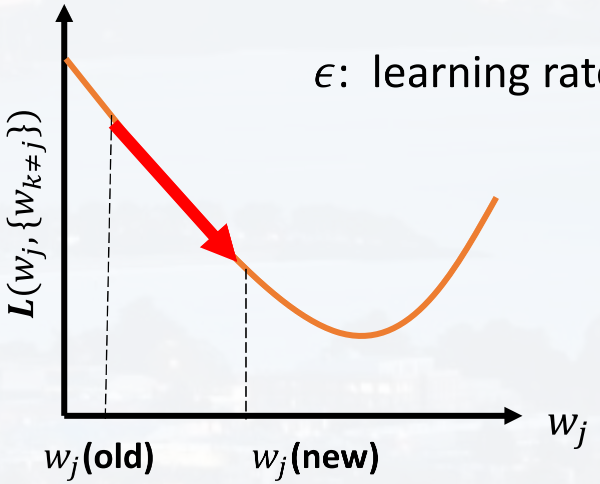
$$dY = dY(\boldsymbol{U_f}, \boldsymbol{W_f}, \boldsymbol{b_f}, \boldsymbol{U_i}, \boldsymbol{W_i}, \boldsymbol{b_i}, \boldsymbol{U_o}, \boldsymbol{W_o}, \boldsymbol{b_o}, \boldsymbol{U_g}, \boldsymbol{W_g}, \boldsymbol{b_g})$$

dense.dinputs



$$\Delta w_j = -\,\epsilon\,\frac{\partial L(w_j, \{w_{k\neq j}\})}{\partial w_j}$$

ϵ: learning rate



RNNs & LSTMs:

$$h(t) = \boxed{tanh(c(t)) * o(t)}$$

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

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

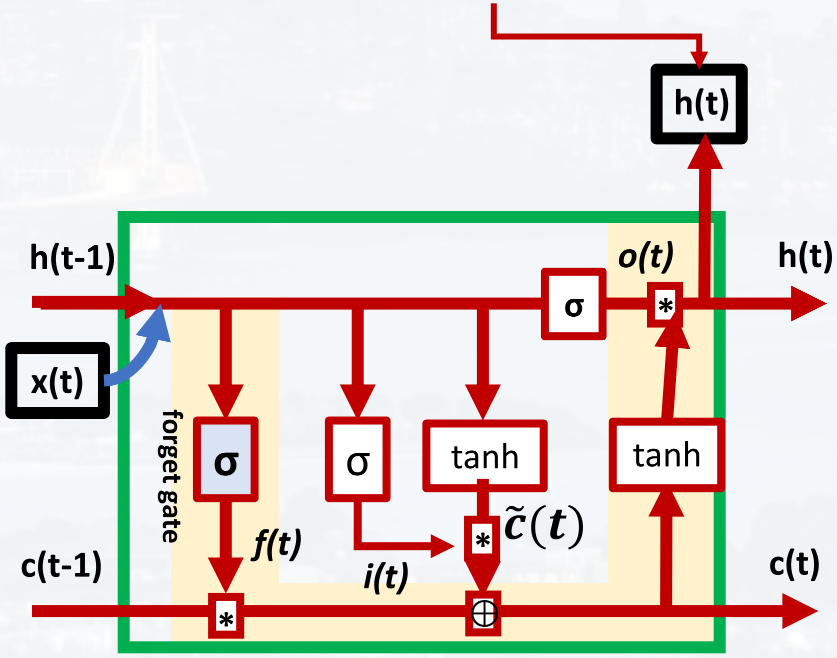
$$\Delta = \Delta\,(U_f, W_f, b_f, U_i, W_i, b_i, U_o, W_o, b_o, U_g, W_g, b_g)$$

$$\frac{\partial h(t)}{\partial U_f} = \frac{\partial h(t)}{\partial tanh}\frac{\partial tanh}{\partial c(t)} * o(t) \quad \frac{\partial c(t)}{\partial f(t)} \quad \frac{\partial f(t)}{\partial \sigma} \quad \frac{\partial \sigma}{\partial U_f}$$

$$\frac{\partial c(t)}{\partial f(t)} = c(t-1) \qquad \frac{\partial \sigma}{\partial U_f} = x(t)$$

```
dht      = dvalues[-1].reshape(self.n_neurons,1)

Tanh2[t].backward(dht)
dtanh2 = Tanh2[t].dinputs

dhtdtanh = np.multiply(O[t], dtanh2)

dctdft   = np.multiply(dhtdtanh,C[t-1])

Sigmf[t].backward(dctdft)
dsigmf = Sigmf[t].dinputs

dsigmfdUf = np.dot(dsigmf,xt)
dUf += dsigmfdUf
```

$\Delta =$ dense.dinputs



RNNs & LSTMs:

$$\Delta = \Delta\,(U_f, \boldsymbol{W_f}, b_f, U_i, W_i, b_i, U_o, W_o, b_o, U_g, W_g, b_g)$$

$$\boldsymbol{h(t)} = \boxed{\boldsymbol{tanh(c(t)) * o(t)}}$$

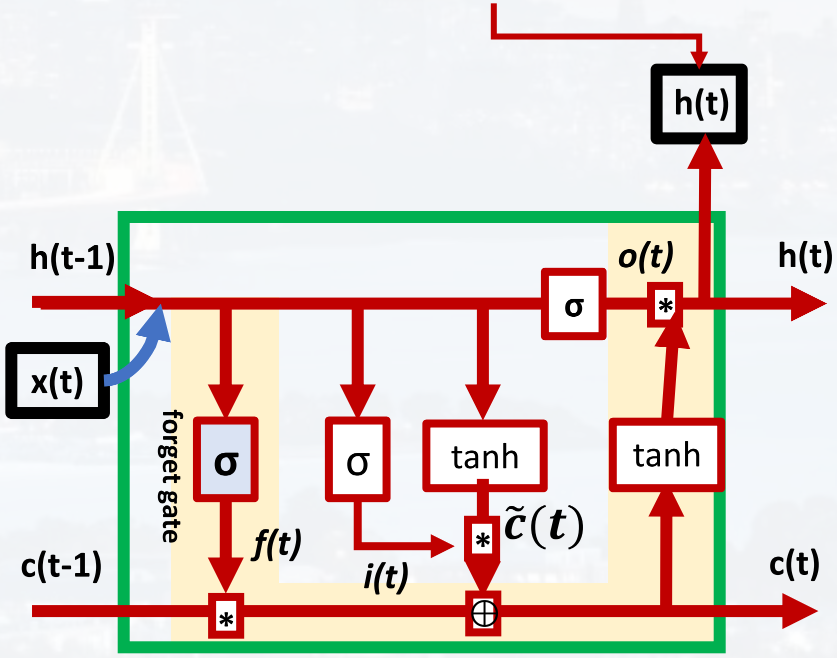
$$\mathbf{c}(\boldsymbol{t}) = \boxed{\boldsymbol{f(t) * c(t-1)}} + \boldsymbol{i(t) * \tilde{c}(t)}$$

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

$$\frac{\partial \boldsymbol{h(t)}}{\partial \boldsymbol{W_f}} = \frac{\partial \boldsymbol{h(t)}}{\partial \boldsymbol{tanh}} \frac{\partial \boldsymbol{tanh}}{\partial \boldsymbol{c(t)}} * \boldsymbol{o(t)} \underbrace{\frac{\partial \boldsymbol{c(t)}}{\partial \boldsymbol{f(t)}}}_{= \boldsymbol{c(t-1)}} \frac{\partial \boldsymbol{f(t)}}{\partial \boldsymbol{\sigma}} \underbrace{\frac{\partial \boldsymbol{\sigma}}{\partial \boldsymbol{W_f}}}_{= \boldsymbol{h(t-1)}}$$

```
dht      = dvalues[-1].reshape(self.n_neurons,1)

Tanh2[t].backward(dht)
dtanh2 = Tanh2[t].dinputs

dhtdtanh = np.multiply(O[t], dtanh2)

dctdft   = np.multiply(dhtdtanh,C[t-1])

Sigmf[t].backward(dctdft)
dsigmf = Sigmf[t].dinputs

dsigmfdWf = np.dot(dsigmf,H[t-1].T)
dWf += dsigmfdWf
```

$\Delta =$ dense.dinputs



RNNs & LSTMs:

$$\Delta = \Delta\,(U_f, W_f, b_f, U_i, W_i, b_i, U_o, W_o, b_o, U_g, W_g, b_g)$$

$$h(t) = \boxed{tanh(c(t)) * o(t)}$$

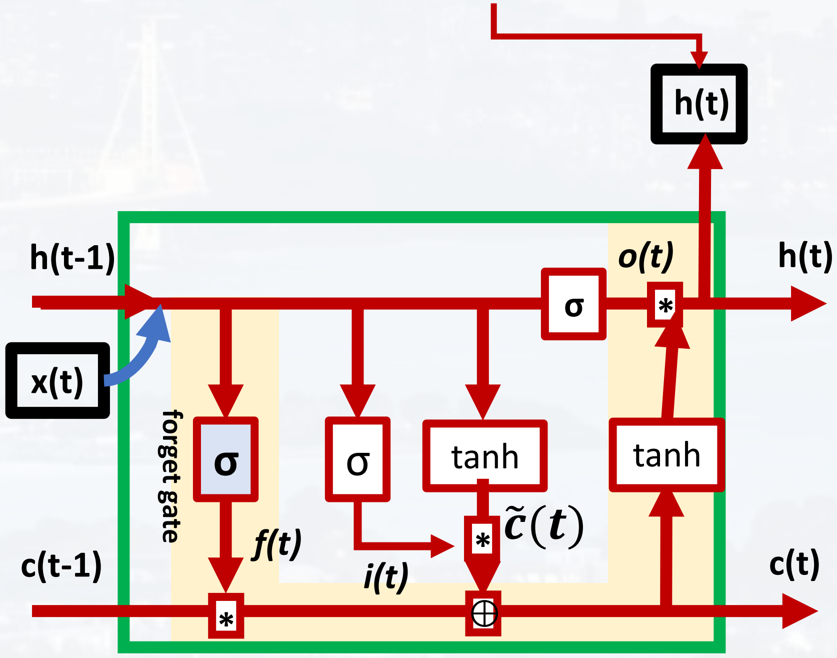
$$c(t) = \boxed{f(t) * c(t-1)} + i(t) * \tilde{c}(t)$$

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

$$\frac{\partial h(t)}{\partial b_f} = \frac{\partial h(t)}{\partial tanh}\frac{\partial tanh}{\partial c(t)} * o(t) \quad \underbrace{\frac{\partial c(t)}{\partial f(t)}}_{= c(t-1)} \quad \frac{\partial f(t)}{\partial \sigma} \underbrace{\frac{\partial \sigma}{\partial b_f}}_{= 1}$$

```
dht      = dvalues[-1].reshape(self.n_neurons,1)

Tanh2[t].backward(dht)
dtanh2 = Tanh2[t].dinputs

dhtdtanh = np.multiply(O[t], dtanh2)

dctdft   = np.multiply(dhtdtanh,C[t-1])

Sigmf[t].backward(dctdft)
dsigmf = Sigmf[t].dinputs

dbf += dsigmf
```

Δ = dense.dinputs



RNNs & LSTMs:

$$\Delta = \Delta\,(U_f, W_f, b_f, \boldsymbol{U_i}, W_i, b_i, U_o, W_o, b_o, U_g, W_g, b_g)$$

$$\boldsymbol{h(t)} = \boxed{\boldsymbol{tanh(c(t)) * o(t)}}$$

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

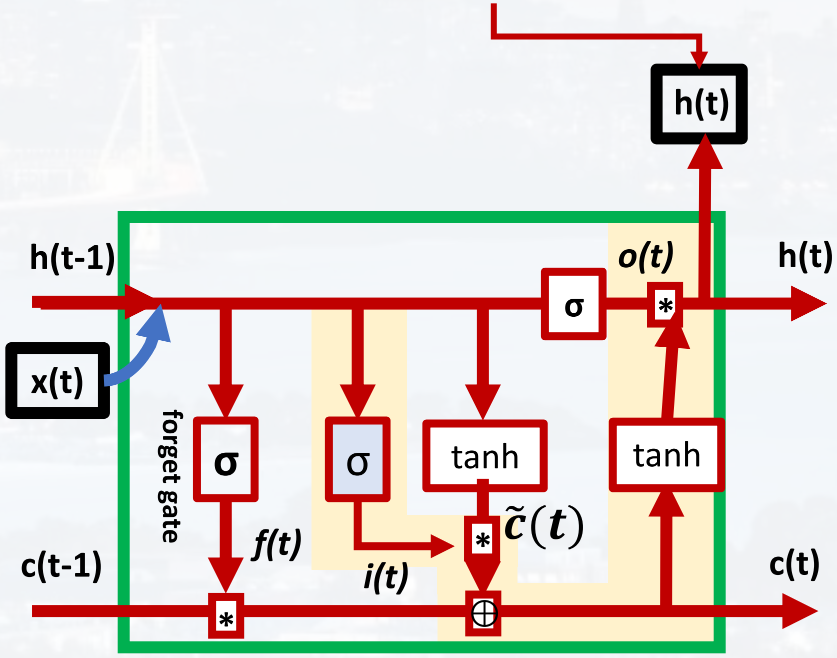
$$\boldsymbol{i(t)} = \boldsymbol{\sigma}\,\boxed{(\boldsymbol{U_i \oplus x(t)})} + \boldsymbol{W_i \oplus h(t-1)} + \boldsymbol{b_i})$$

$$\frac{\partial h(t)}{\partial U_i} = \frac{\partial h(t)}{\partial tanh}\frac{\partial tanh}{\partial c(t)} * o(t)\ \frac{\partial c(t)}{\partial i(t)} * \tilde{c}(t)\ \frac{\partial i(t)}{\partial \sigma}\ \frac{\partial \sigma}{\partial U_i}$$

$$= x(t)$$

```
dht    = dvalues[-1].reshape(self.n_neurons,1)

Tanh2[t].backward(dht)
dtanh2 = Tanh2[t].dinputs

dhtdtanh = np.multiply(O[t], dtanh2)

dctdit   = np.multiply(dhtdtanh,C_tilde[t])

Sigmi[t].backward(dctdit)
dsigmi = Sigmi[t].dinputs

dsigmidUi = np.dot(dsigmi,xt)
dUi += dsigmidUi
```

$\Delta =$ dense.dinputs



RNNs & LSTMs:

$$\Delta = \Delta\,(U_f, W_f, b_f, U_i, \boldsymbol{W_i}, b_i, U_o, W_o, b_o, U_g, W_g, b_g)$$

$$\boldsymbol{h(t)} = \boxed{\boldsymbol{tanh(c(t)) * o(t)}}$$

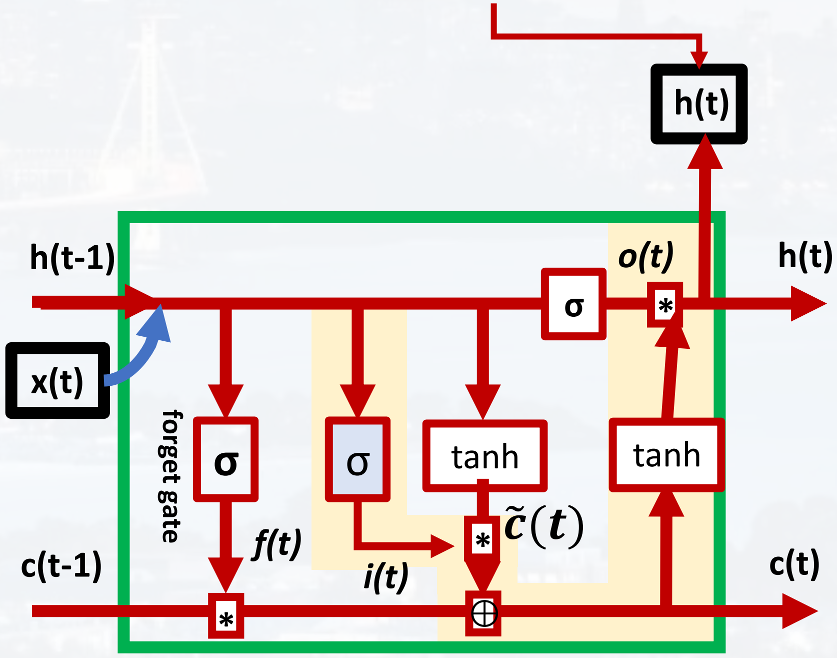
$$\mathbf{c(t)} = \boldsymbol{f(t) * c(t-1)} + \boxed{\boldsymbol{i(t) * \tilde{c}(t)}}$$

$$\boldsymbol{i(t)} = \boldsymbol{\sigma}\,(\boldsymbol{U_i \oplus x(t)} + \boxed{\boldsymbol{W_i \oplus h(t-1)}} + \boldsymbol{b_i})$$

$$\frac{\partial \boldsymbol{h(t)}}{\partial \boldsymbol{W_i}} = \frac{\partial \boldsymbol{h(t)}}{\partial \boldsymbol{tanh}} \frac{\partial \boldsymbol{tanh}}{\partial \boldsymbol{c(t)}} * \boldsymbol{o(t)}\ \frac{\partial \boldsymbol{c(t)}}{\partial \boldsymbol{i(t)}} * \boldsymbol{\tilde{c}(t)}\ \frac{\partial \boldsymbol{i(t)}}{\partial \boldsymbol{\sigma}}\ \underbrace{\frac{\partial \boldsymbol{\sigma}}{\partial \boldsymbol{W_i}}}_{= \boldsymbol{h(t-1)}}$$

```
dht      = dvalues[-1].reshape(self.n_neurons,1)

Tanh2[t].backward(dht)
dtanh2 = Tanh2[t].dinputs

dhtdtanh = np.multiply(O[t], dtanh2)

dctdit   = np.multiply(dhtdtanh,C_tilde[t])

Sigmi[t].backward(dctdit)
dsigmi = Sigmi[t].dinputs

dsigmidWi = np.dot(dsigmi,H[t-1].T)
dWi += dsigmidWi
```

Δ = dense.dinputs



$$\Delta = \Delta\,(U_f, W_f, b_f, U_i, W_i, \boldsymbol{b_i}, U_o, W_o, b_o, U_g, W_g, b_g)$$

$$\boldsymbol{h(t)} = \boxed{\boldsymbol{tanh(c(t)) * o(t)}}$$

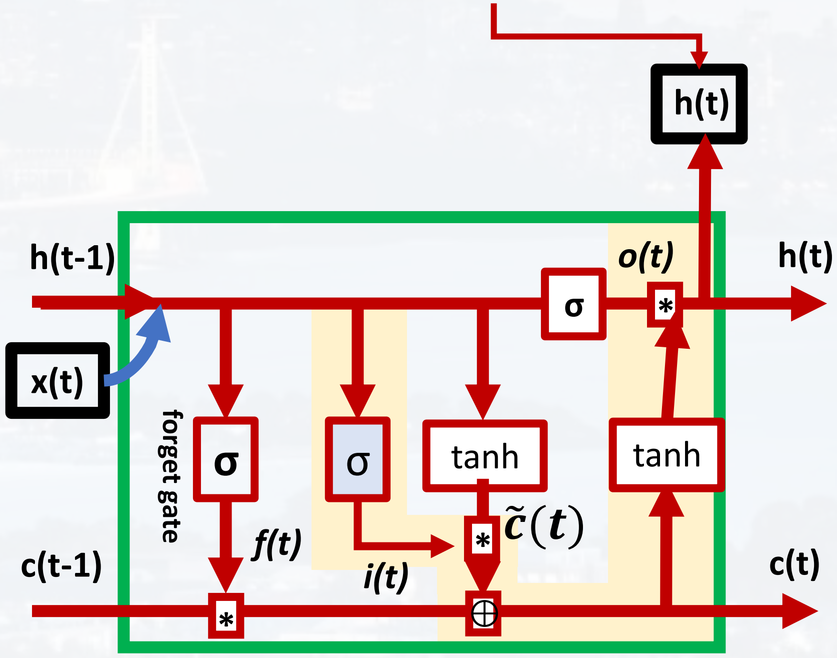
$$\mathbf{c}(\boldsymbol{t}) = \boldsymbol{f(t) * c(t-1)} + \boxed{\boldsymbol{i(t) * \tilde{c}(t)}}$$

$$\boldsymbol{i(t)} = \boldsymbol{\sigma\,(U_i \oplus x(t) + W_i \oplus h(t-1) +} \boxed{\boldsymbol{b_i}}\boldsymbol{)}$$

$$\frac{\partial \boldsymbol{h(t)}}{\partial \boldsymbol{b_i}} = \frac{\partial \boldsymbol{h(t)}}{\partial \boldsymbol{tanh}} \frac{\partial \boldsymbol{tanh}}{\partial \boldsymbol{c(t)}} * \boldsymbol{o(t)} \; \frac{\partial \boldsymbol{c(t)}}{\partial \boldsymbol{i(t)}} * \boldsymbol{\tilde{c}(t)} \; \frac{\partial \boldsymbol{i(t)}}{\partial \boldsymbol{\sigma}} \; \underbrace{\frac{\partial \boldsymbol{\sigma}}{\partial \boldsymbol{b_i}}}_{=\,\mathbf{1}}$$

$\Delta =$ dense.dinputs

```
dht     = dvalues[-1].reshape(self.n_neurons,1)

Tanh2[t].backward(dht)
dtanh2 = Tanh2[t].dinputs

dhtdtanh = np.multiply(O[t], dtanh2)

dctdit   = np.multiply(dhtdtanh,C_tilde[t])

Sigmi[t].backward(dctdit)
dsigmi = Sigmi[t].dinputs

dbi += dsigmi
```



RNNs & LSTMs:

$$\Delta = \Delta\,(U_f, W_f, b_f, U_i, W_i, b_i, \boldsymbol{U_o}, W_o, b_o, U_g, W_g, b_g)$$

$$\boldsymbol{h(t)} = \boxed{\boldsymbol{tanh(c(t)) * o(t)}}$$

$$\boldsymbol{o(t)} = \boldsymbol{\sigma}\,\boxed{\boldsymbol{(U_o \oplus x(t)}} \boldsymbol{+ W_o \oplus h(t-1) + b_o)}$$

$$\frac{\partial \boldsymbol{h(t)}}{\partial \boldsymbol{U_o}} = \frac{\partial \boldsymbol{h(t)}}{\partial \boldsymbol{o(t)}} * \boldsymbol{tanh(c(t))} \; \frac{\partial \boldsymbol{o(t)}}{\partial \boldsymbol{\sigma}} \; \frac{\partial \boldsymbol{\sigma}}{\partial \boldsymbol{U_o}}$$

$$= \boldsymbol{x(t)}$$

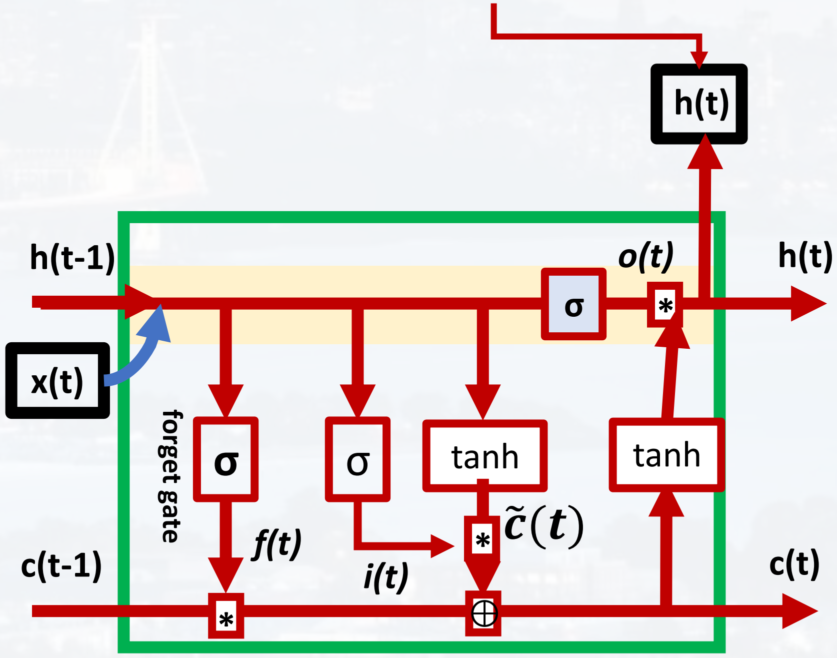
```
dht     = dvalues[-1].reshape(self.n_neurons,1)
```

$\Delta =$ `dense.dinputs`

```
Sigmo[t].backward(np.multiply(dht, Tanh2[t].output))
dsigmo = Sigmo[t].dinputs
```

```
dsigmodUo = np.dot(dsigmo,xt)
```

```
dUo += dsigmodUo
```



$$\Delta = \Delta\,(U_f, W_f, b_f, U_i, W_i, b_i, U_o, \boldsymbol{W_o}, b_o, U_g, W_g, b_g)$$

$$\boldsymbol{h(t)} = \boxed{\boldsymbol{tanh(c(t)) * o(t)}}$$

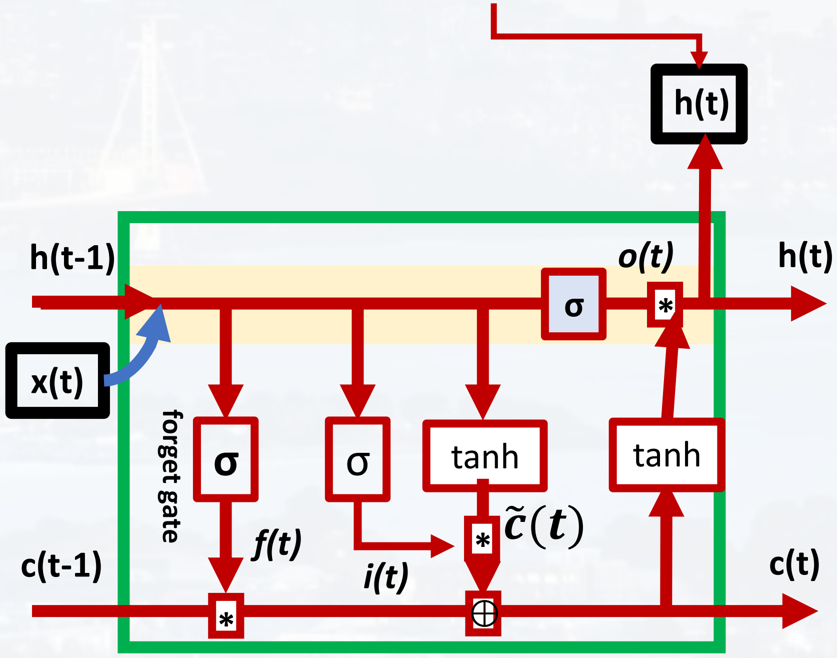
$$\boldsymbol{o(t)} = \boldsymbol{\sigma}\,(\boldsymbol{U_o} \oplus \boldsymbol{x(t)} + \boxed{\boldsymbol{W_o} \oplus \boldsymbol{h(t-1)}} + \boldsymbol{b_o})$$

$$\frac{\partial \boldsymbol{h(t)}}{\partial \boldsymbol{W_o}} = \frac{\partial \boldsymbol{h(t)}}{\partial \boldsymbol{o(t)}} * \boldsymbol{tanh(c(t))}\,\frac{\partial \boldsymbol{o(t)}}{\partial \boldsymbol{\sigma}}\,\frac{\partial \boldsymbol{\sigma}}{\partial \boldsymbol{W_o}}$$

$$= \boldsymbol{h(t-1)}$$

```
dht     = dvalues[-1].reshape(self.n_neurons,1)

Sigmo[t].backward(np.multiply(dht, Tanh2[t].output))
dsigmo = Sigmo[t].dinputs

dsigmodWo = np.dot(dsigmo,H[t-1].T)

dWo += dsigmodWo
```

Δ = dense.dinputs



RNNs & LSTMs:

$$h(t) = \boxed{tanh(c(t)) * o(t)}$$

$$o(t) = \sigma\,(U_o \oplus x(t) + W_o \oplus h(t-1) + \boxed{b_o})$$

$$\Delta = \Delta\,(U_f, W_f, b_f, U_i, W_i, b_i, U_o, W_o, \mathbf{b_o}, U_g, W_g, b_g)$$

$$\frac{\partial h(t)}{\partial b_o} = \frac{\partial h(t)}{\partial o(t)} * tanh(c(t))\ \frac{\partial o(t)}{\partial \sigma}\ \frac{\partial \sigma}{\partial b_o}$$

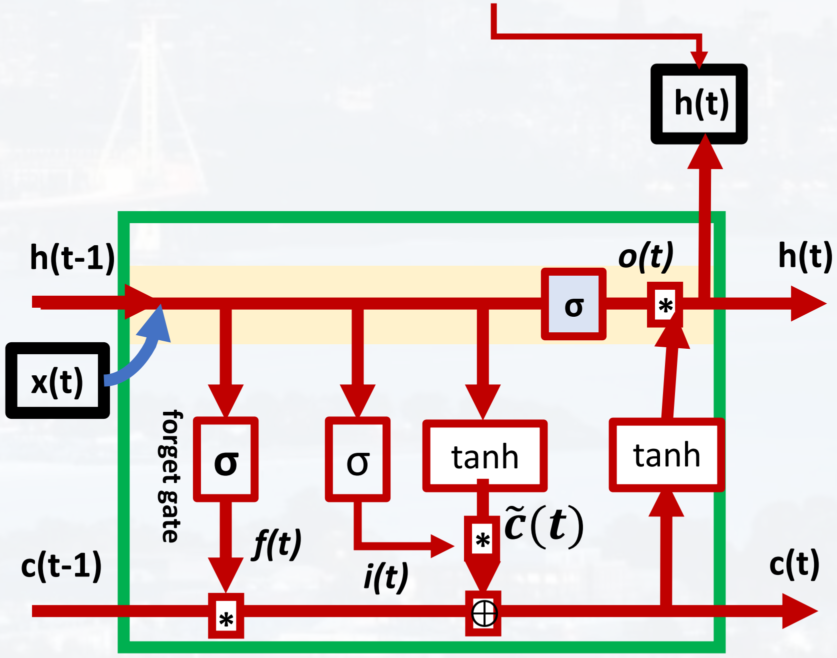
$$= 1$$

```
dht     = dvalues[-1].reshape(self.n_neurons,1)
```

$\Delta =$ `dense.dinputs`

```
Sigmo[t].backward(np.multiply(dht, Tanh2[t].output))
dsigmo = Sigmo[t].dinputs
```

```
dbo += dsigmo
```



RNNs & LSTMs: BPTT

$$h(t) = \boxed{tanh(c(t)) * o(t)}$$

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

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

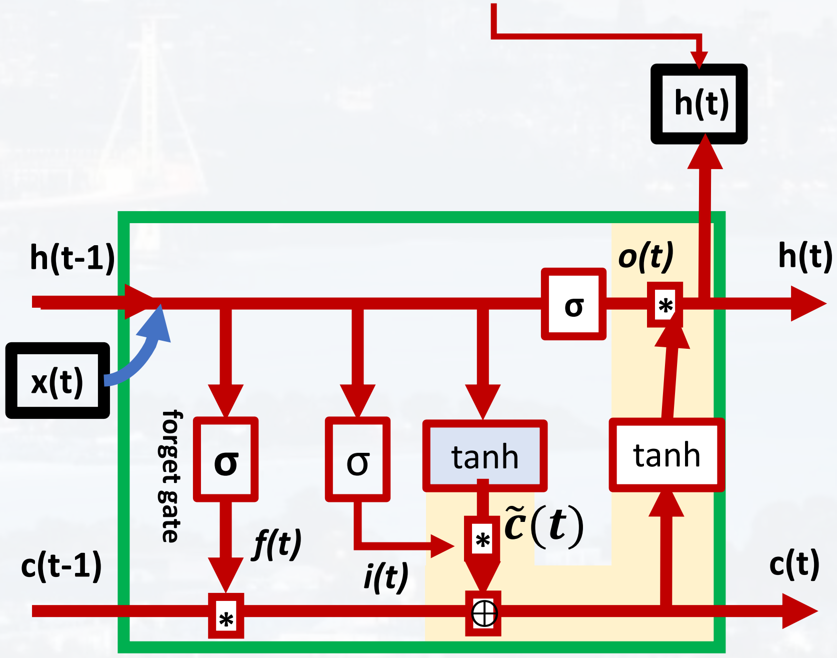
$$\Delta = \Delta\,(U_f, W_f, b_f, U_i, W_i, b_i, U_o, W_o, b_o, U_g, W_g, b_g)$$

$$\frac{\partial h(t)}{\partial U_g} = \frac{\partial h(t)}{\partial tanh}\frac{\partial tanh}{\partial c(t)} * o(t)\ \frac{\partial c(t)}{\partial \tilde{c}(t)} * i(t)\ \frac{\partial \tilde{c}(t)}{\partial tanh}\frac{\partial tanh}{\partial U_g}$$

$$= x(t)$$

```
dht     = dvalues[-1].reshape(self.n_neurons,1)

Tanh2[t].backward(dht)
dtanh2 = Tanh2[t].dinputs

dhtdtanh = np.multiply(O[t], dtanh2)

dctdct_tilde = np.multiply(dhtdtanh,I[t])

Tanh1[t].backward(dctdct_tilde)
dtanh1 = Tanh1[t].dinputs

dtanh1dUg = np.dot(dtanh1,xt)
dUg += dtanh1dUg
```

$\Delta =$ dense.dinputs



RNNs & LSTMs:

$$\Delta = \Delta\,(U_f, W_f, b_f, U_i, W_i, b_i, U_o, W_o, b_o, U_g, \boldsymbol{W_g}, b_g)$$

$$\boldsymbol{h(t)} = \boxed{\boldsymbol{tanh(c(t)) * o(t)}}$$

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

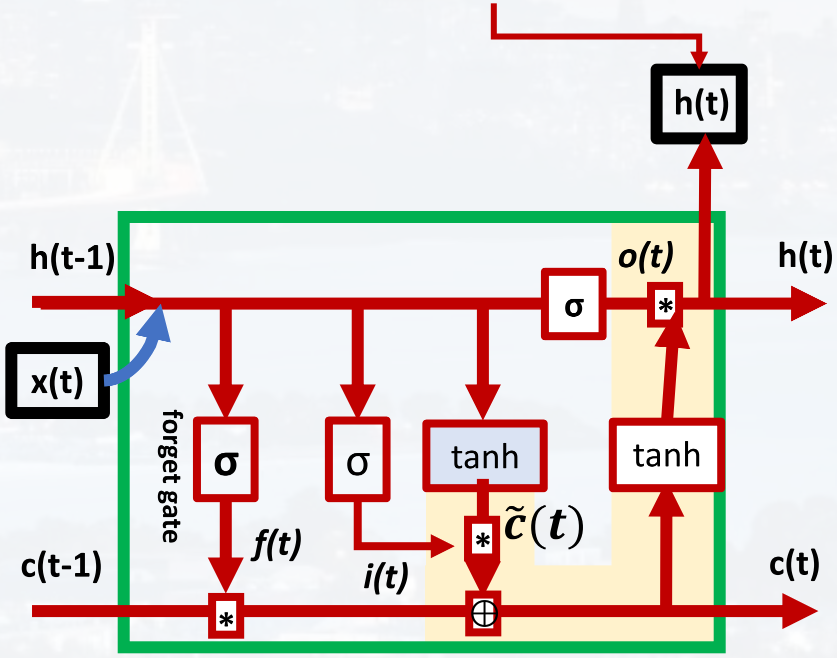
$$\boldsymbol{\tilde{c}(t) = tanh(U_g \oplus x(t)} + \boxed{\boldsymbol{W_g \oplus h(t-1)}} + \boldsymbol{b_g)}$$

$$\frac{\boldsymbol{\partial h(t)}}{\boldsymbol{\partial W_g}} = \frac{\boldsymbol{\partial h(t)}}{\boldsymbol{\partial tanh}} \frac{\boldsymbol{\partial tanh}}{\boldsymbol{\partial c(t)}} * \boldsymbol{o(t)} \frac{\boldsymbol{\partial c(t)}}{\boldsymbol{\partial \tilde{c}(t)}} * \boldsymbol{i(t)} \frac{\boldsymbol{\partial \tilde{c}(t)}}{\boldsymbol{\partial tanh}} \frac{\boldsymbol{\partial tanh}}{\boldsymbol{\partial W_g}}$$

$$= \boldsymbol{h(t-1)}$$

```
dht      = dvalues[-1].reshape(self.n_neurons,1)

Tanh2[t].backward(dht)
dtanh2 = Tanh2[t].dinputs

dhtdtanh = np.multiply(O[t], dtanh2)

dctdct_tilde = np.multiply(dhtdtanh,I[t])

Tanh1[t].backward(dctdct_tilde)
dtanh1 = Tanh1[t].dinputs

dtanh1dWg = np.dot(dtanh1,H[t-1].T)
dWg += dtanh1dWg
```

$\Delta =$ `dense.dinputs`



RNNs & LSTMs:

$$\Delta = \Delta\,(U_f, W_f, b_f, U_i, W_i, b_i, U_o, W_o, b_o, U_g, W_g, \boldsymbol{b_g})$$

$$\boldsymbol{h(t)} = \boxed{\boldsymbol{tanh(c(t)) * o(t)}}$$

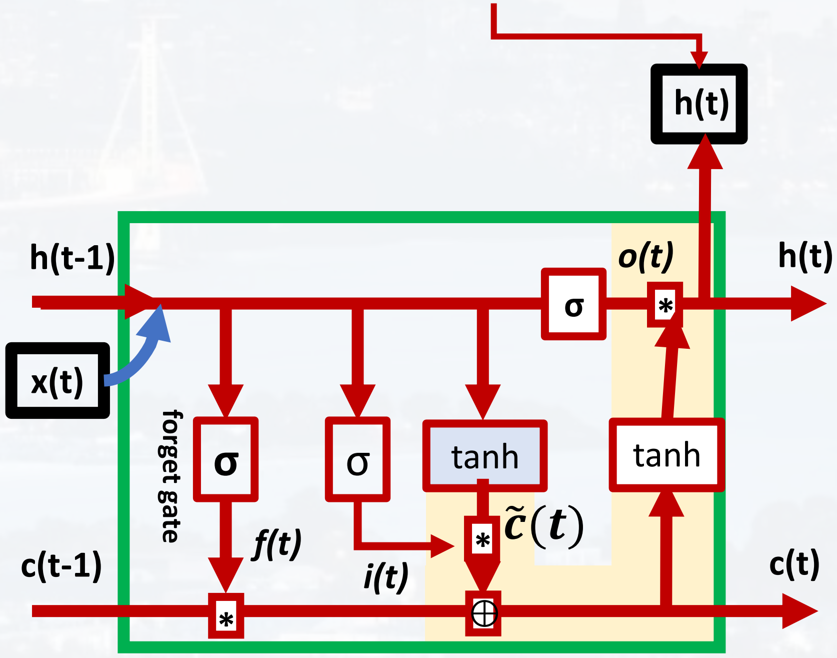
$$\mathbf{c}(\boldsymbol{t}) = \boldsymbol{f}(\boldsymbol{t}) * \boldsymbol{c}(\boldsymbol{t}-\mathbf{1}) + \boxed{\boldsymbol{i}(\boldsymbol{t}) * \tilde{\boldsymbol{c}}(\boldsymbol{t})}$$

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

$$\frac{\partial \boldsymbol{h}(\boldsymbol{t})}{\partial \boldsymbol{b_g}} = \frac{\partial \boldsymbol{h}(\boldsymbol{t})}{\partial \boldsymbol{tanh}} \frac{\partial \boldsymbol{tanh}}{\partial \boldsymbol{c}(\boldsymbol{t})} * \boldsymbol{o}(\boldsymbol{t}) \frac{\partial \boldsymbol{c}(\boldsymbol{t})}{\partial \tilde{\boldsymbol{c}}(\boldsymbol{t})} * \boldsymbol{i}(\boldsymbol{t}) \frac{\partial \tilde{\boldsymbol{c}}(\boldsymbol{t})}{\partial \boldsymbol{tanh}} \underbrace{\frac{\partial \boldsymbol{tanh}}{\partial \boldsymbol{b_g}}}_{=\mathbf{1}}$$

```
dht     = dvalues[-1].reshape(self.n_neurons,1)

Tanh2[t].backward(dht)
dtanh2 = Tanh2[t].dinputs

dhtdtanh = np.multiply(O[t], dtanh2)

dctdct_tilde = np.multiply(dhtdtanh,I[t])

Tanh1[t].backward(dctdct_tilde)
dtanh1 = Tanh1[t].dinputs

dbg += dtanh1
```

$\Delta =$ `dense.dinputs`



We finally need *dh(t)* for the previous cell

$$\frac{\partial h(t)}{\partial h(t-1)} =$$

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

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

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

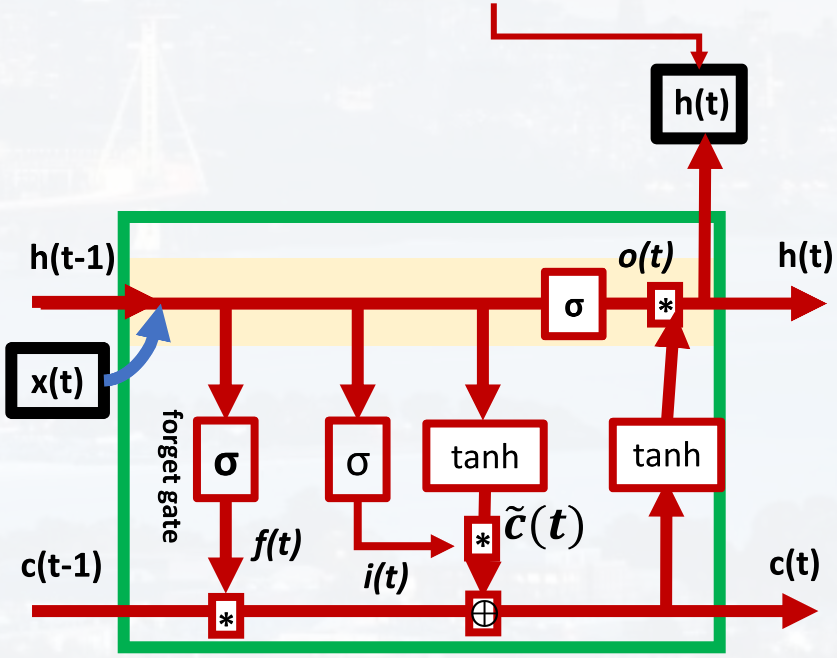
$$\tilde{c}(t) = tanh(U_g \oplus x(t) + W_g \oplus h(t-1) + b_g)$$

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

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

```
dht =   np.dot(Wf, dsigmf) + np.dot(Wi, dsigmi) +\
        np.dot(Wo, dsigmo) + np.dot(Wg, dtanh1) +\
        dvalues[t-1,:].reshape(self.n_neurons, 1)
```

Δ = dense.dinputs



https://www.analyticsvidhya.com

Outline

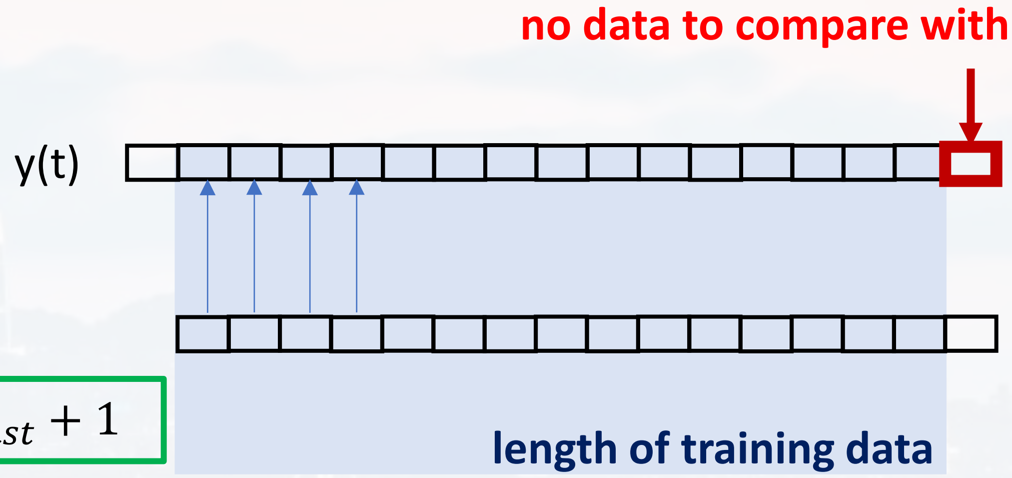
- Idea and classic RNNs
- LSTMs
- *B*ack*P*ropagation *T*hrough *T*ime (BPTT)
- **Syntax and some examples**

RNNs & LSTMs:

Let us first understand the logic:

y(t) 

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



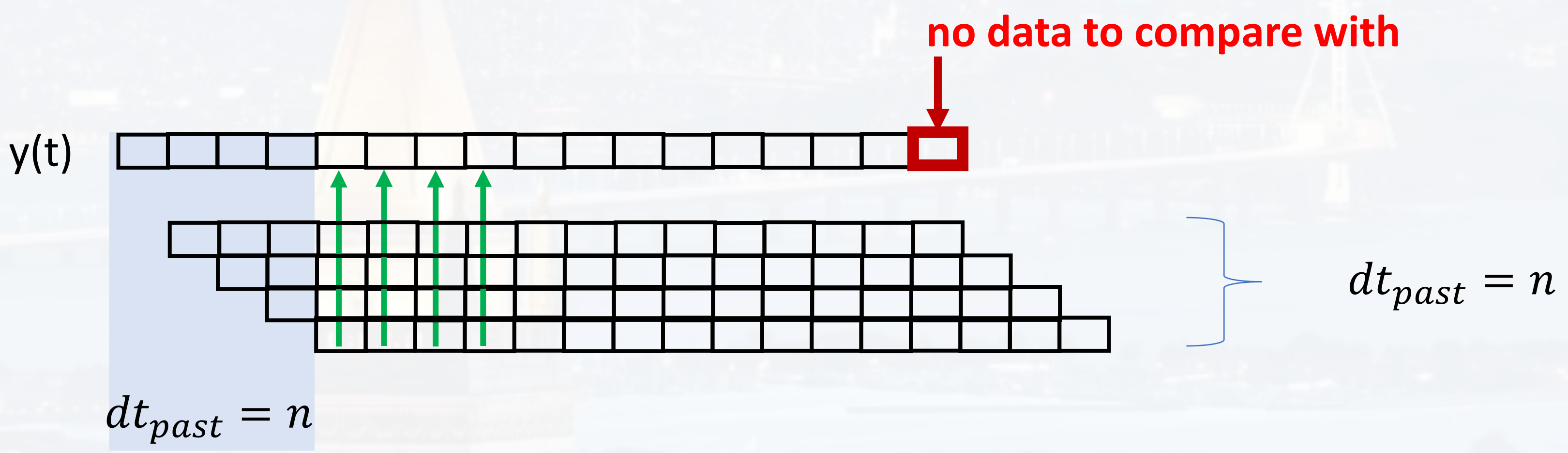
length of training data is: `len[y(t)]` - dt_{futu} - dt_{past} + 1

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

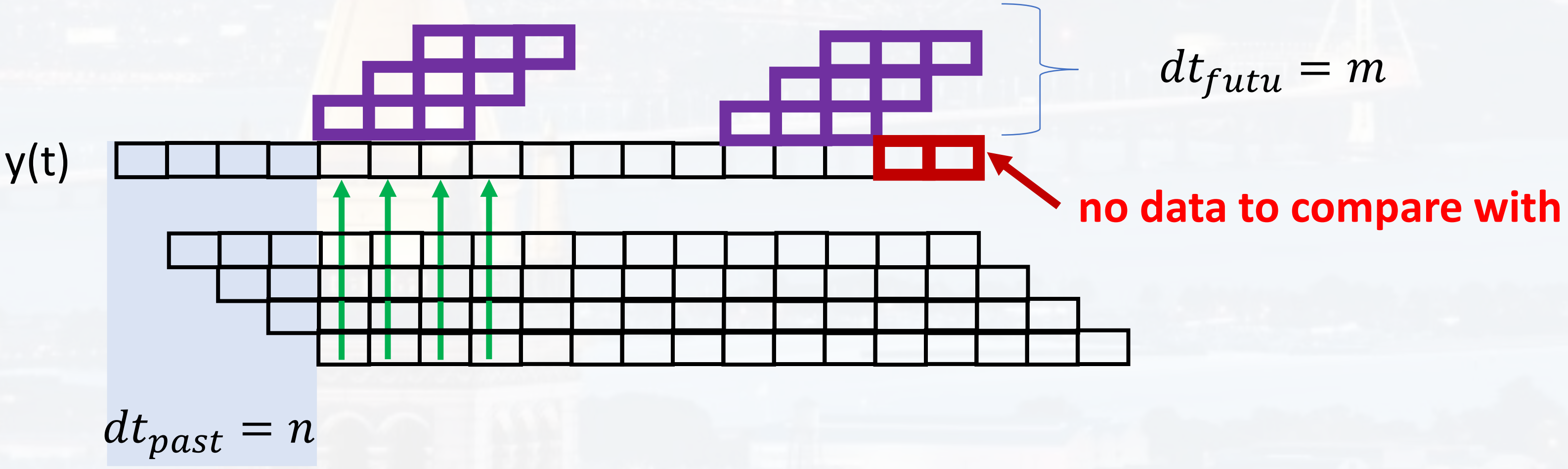
Let us first understand the logic:

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

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

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

predicting m steps of the future



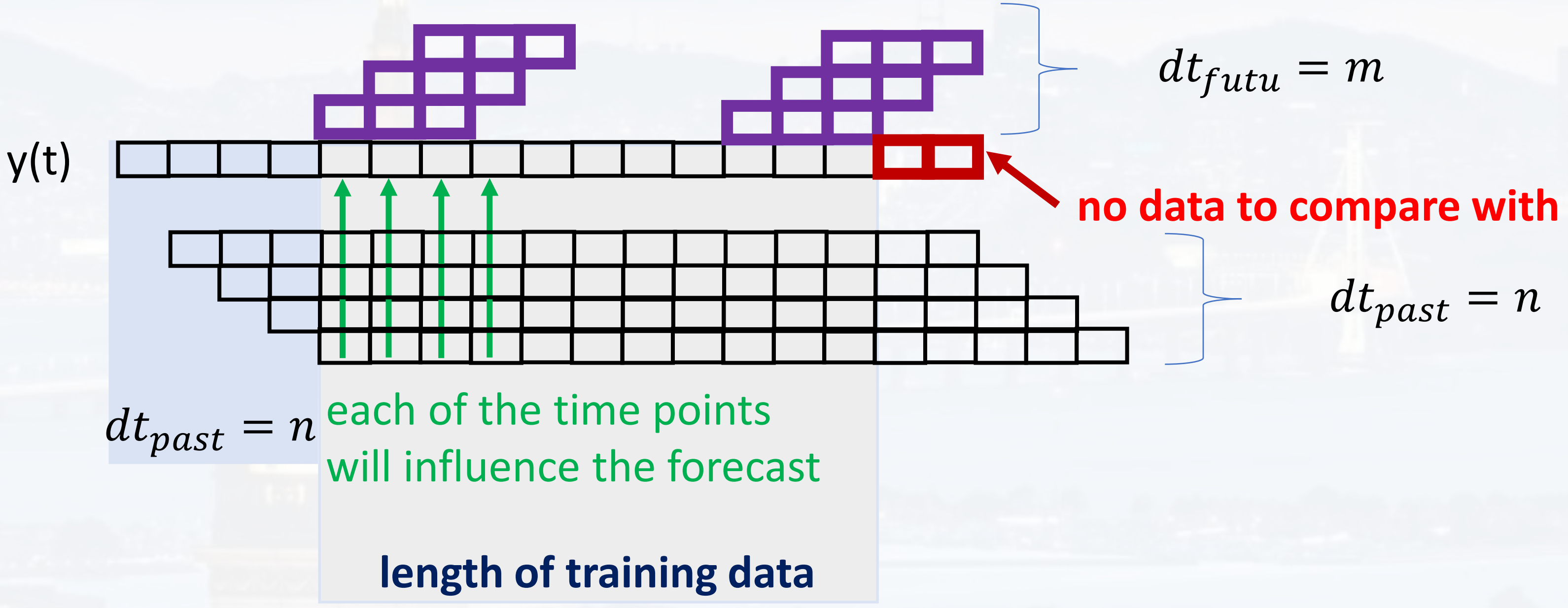
each of the time points
will influence the forecast

RNNs & LSTMs: Syntax and some examples

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 $n_{feature}(Y)$

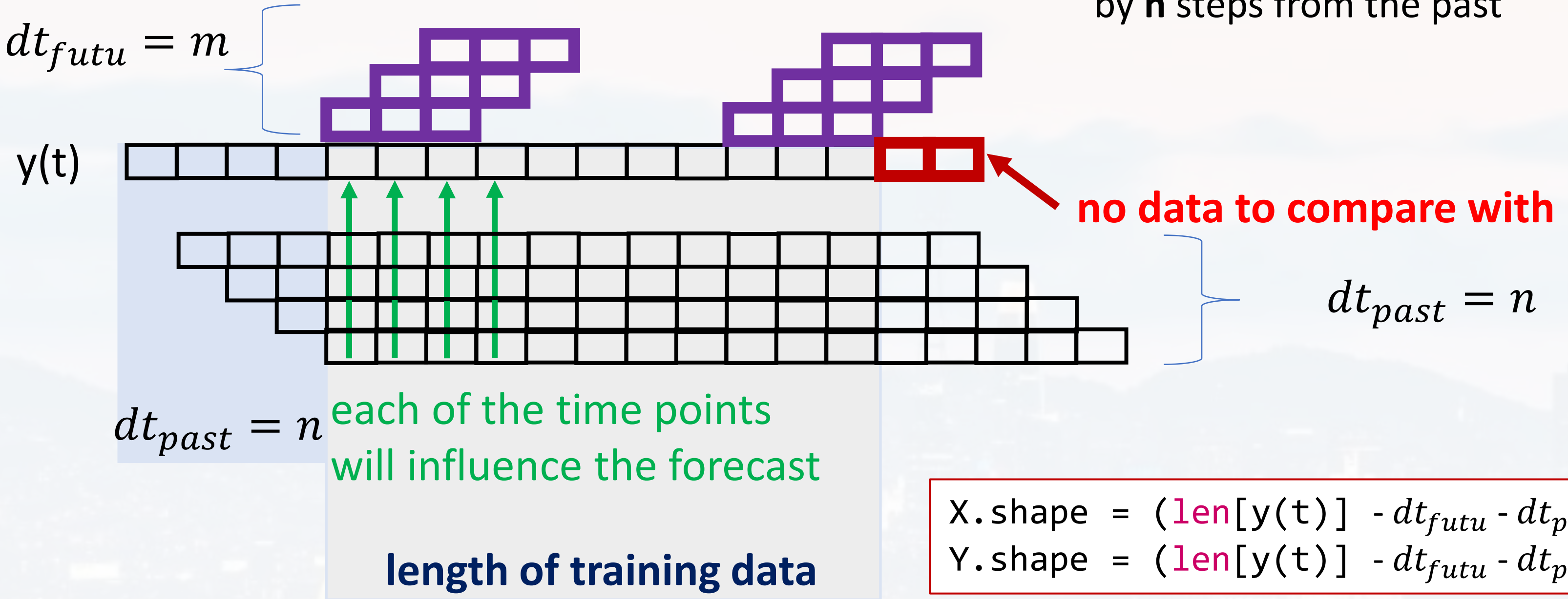
RNNs & LSTMs:

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 $\quad dt_{past} \rightarrow$

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

```
[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 $\quad dt_{futu} \rightarrow$

```
[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]]
```

RNNs & 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

RNNs & LSTMs:

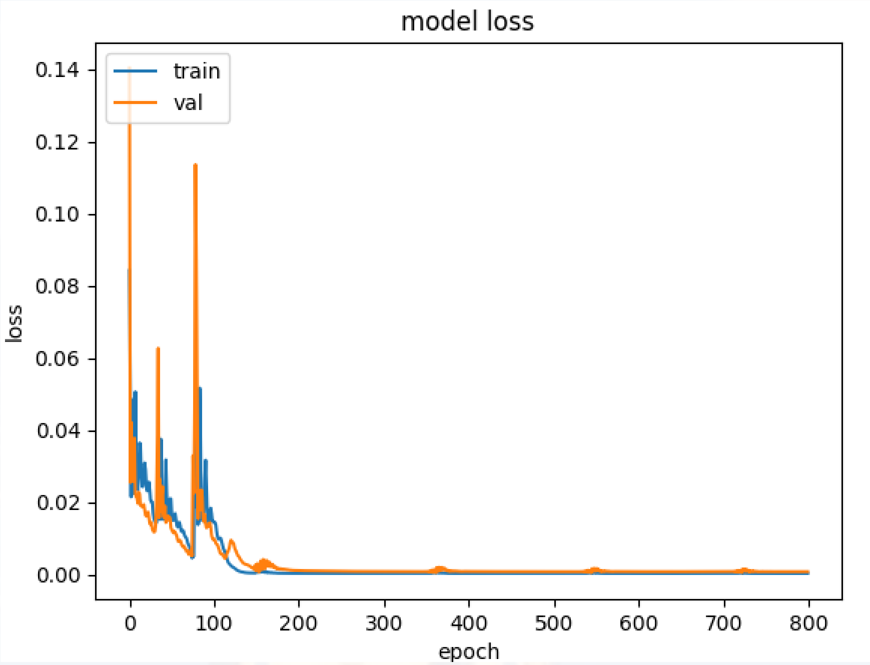
Syntax and some examples

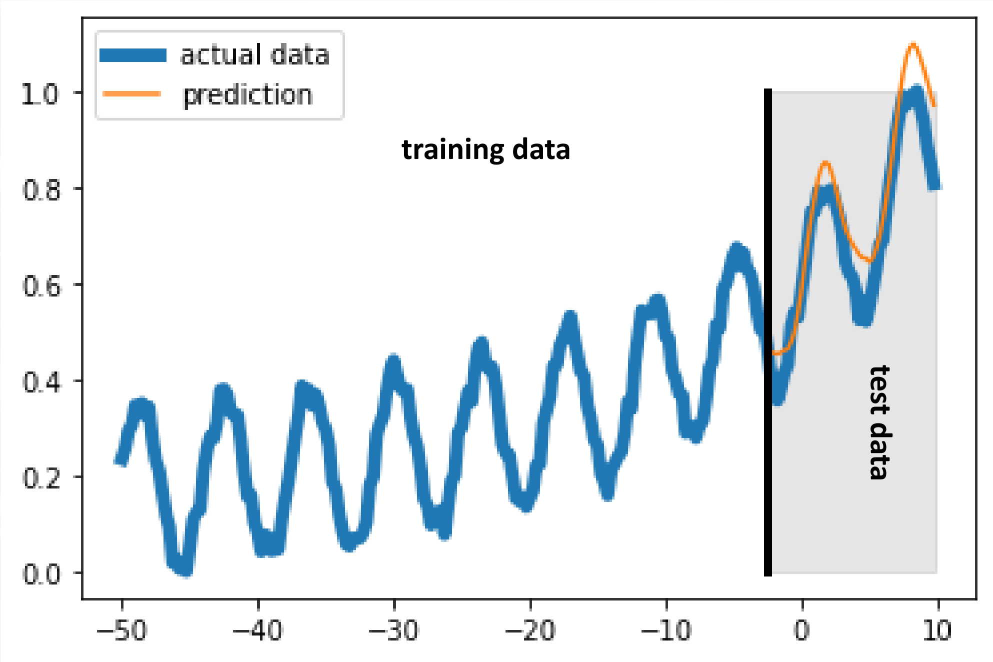
Let us explore LSTMI.ipynb

```
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()
```

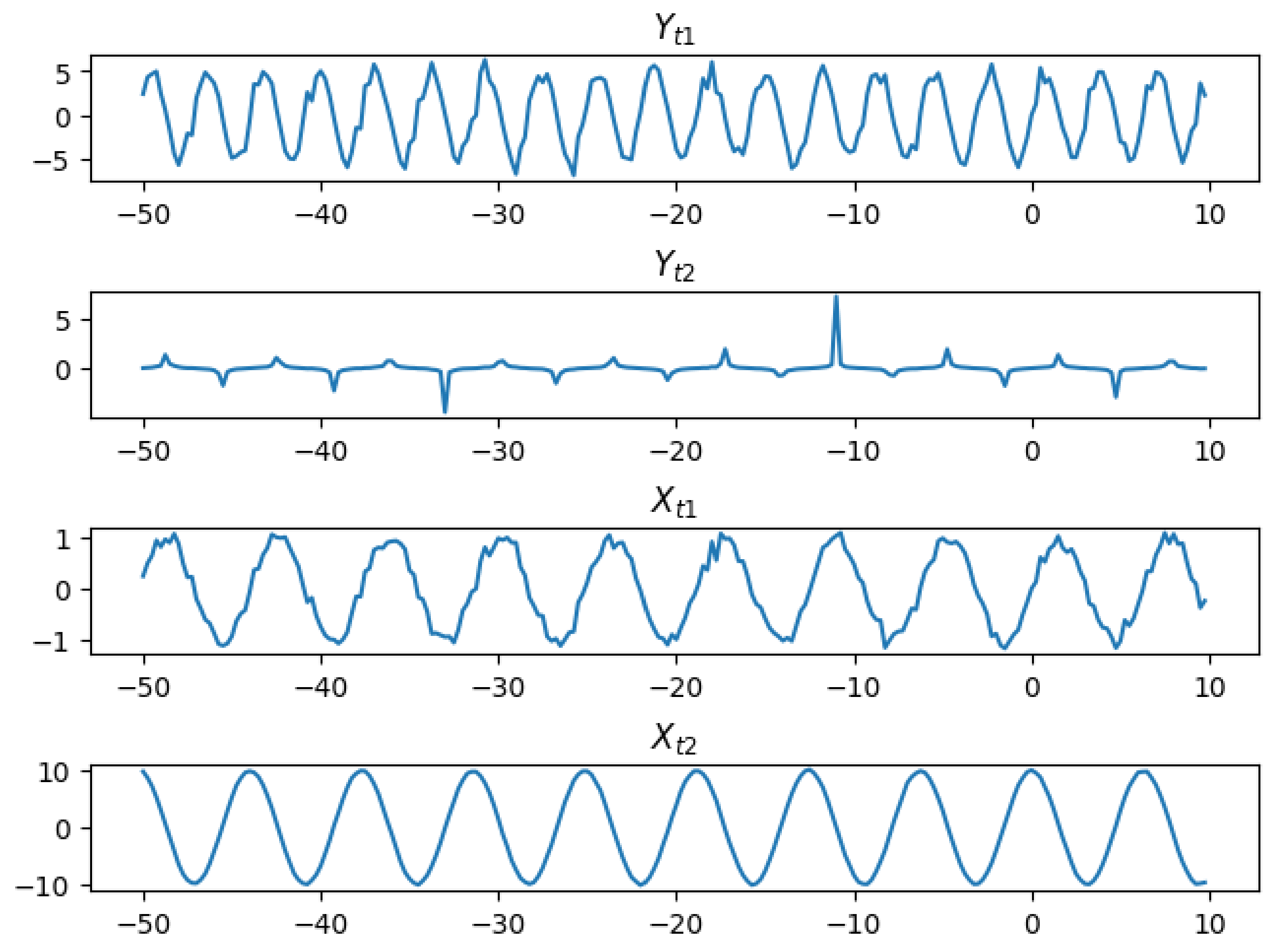
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 400)               643200

 dense (Dense)               (None, 8)                 3208

=================================================================
Total params: 646408 (2.47 MB)
Trainable params: 646408 (2.47 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Let us explore `LSTMI.ipynb`

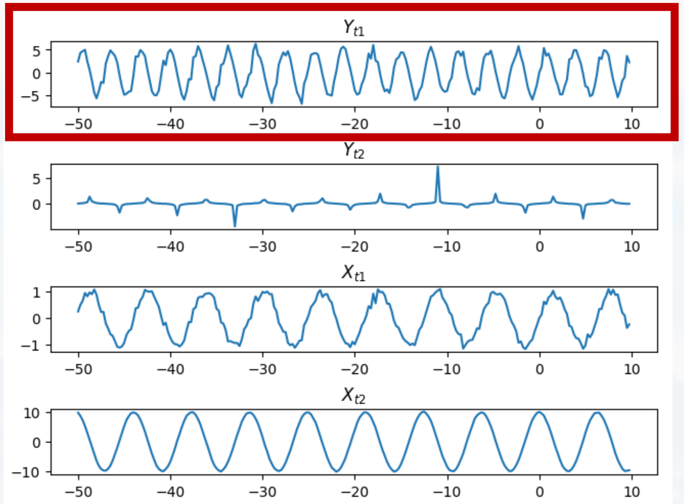


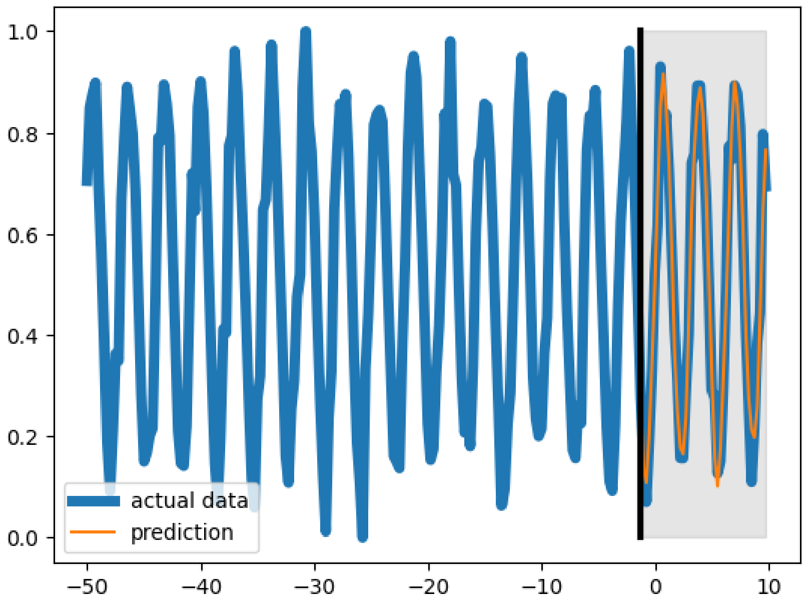


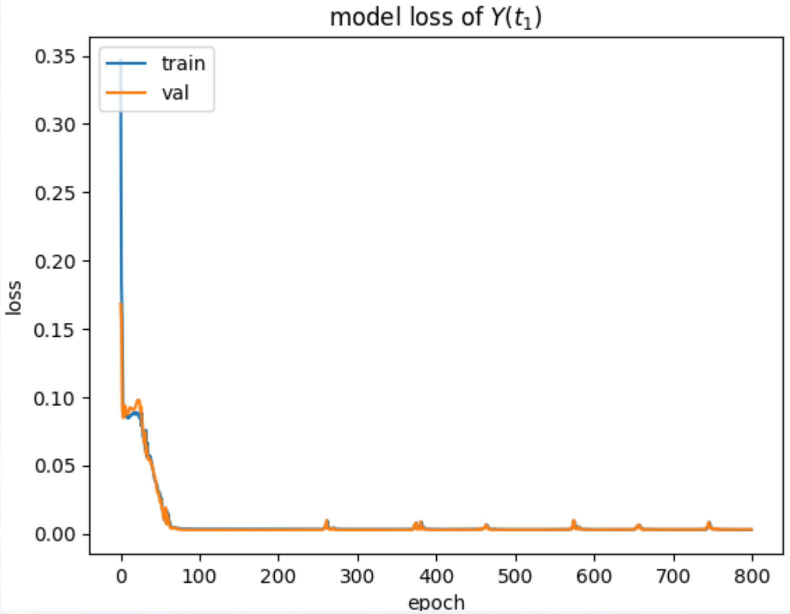
Explore `LSTMII.ipynb` for a multivariate, multi feature time series:



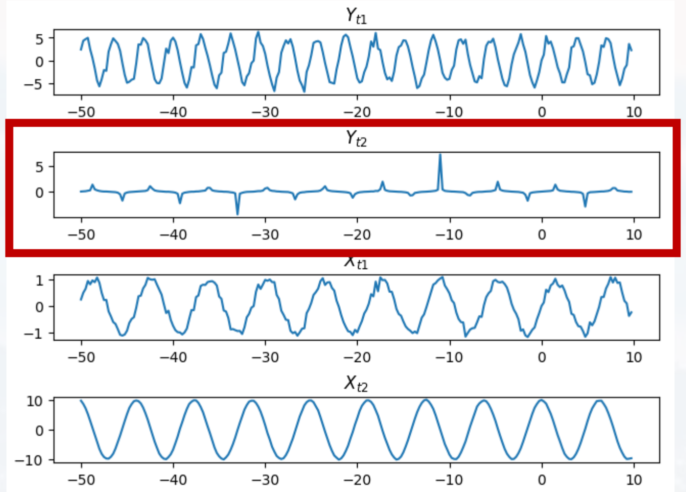
Explore `LSTMII.ipynb` for a multivariate, multi feature time series:

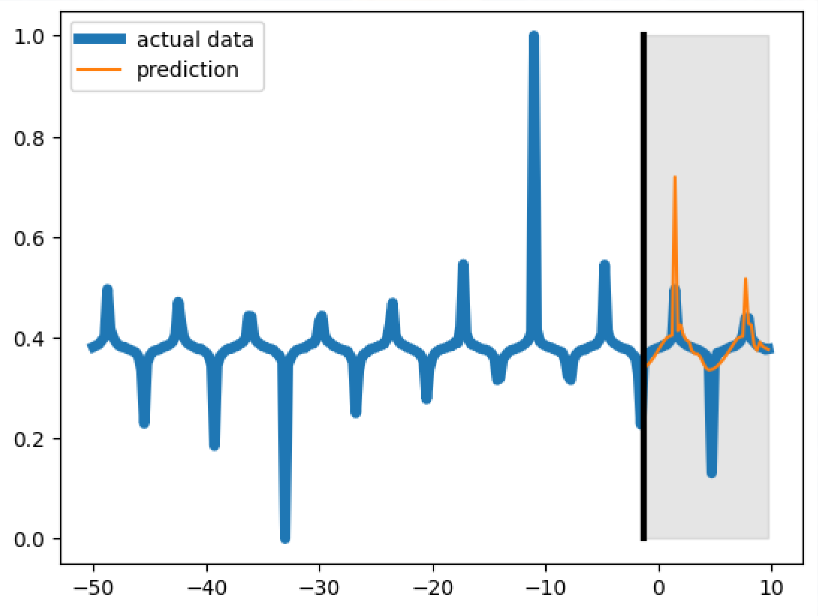


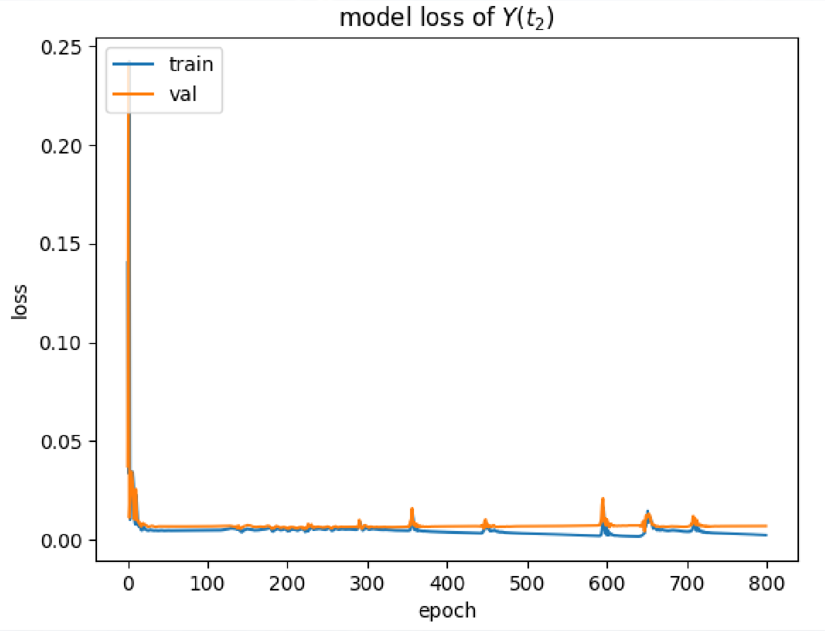




Explore `LSTMII.ipynb` for a multivariate, multi feature time series:

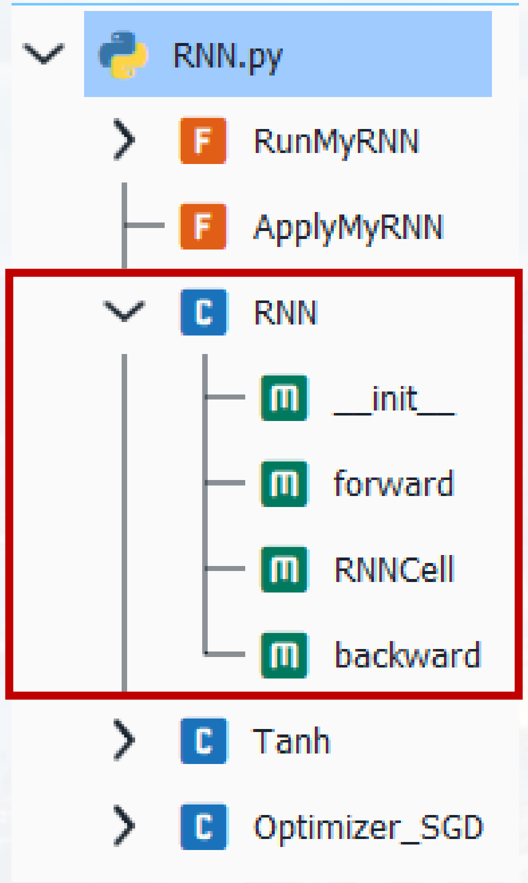


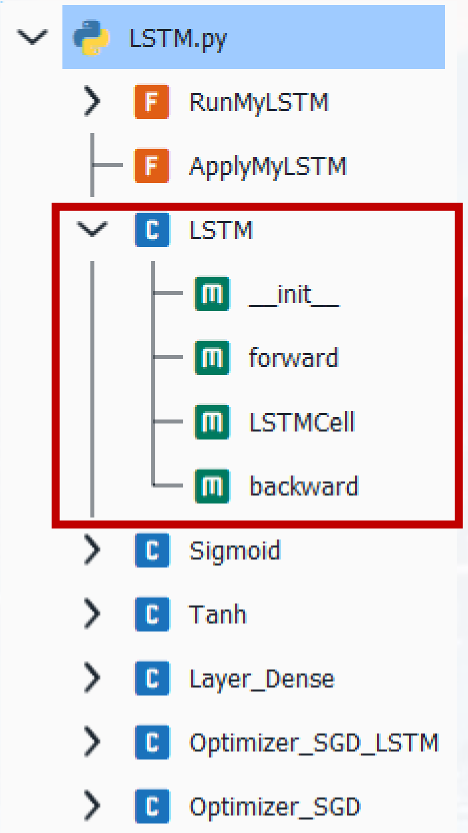




RNNs & LSTMs: Syntax and some examples

RNN & LSTM spelled out: explore **`LSTM.py`** and **`RNN.py`**

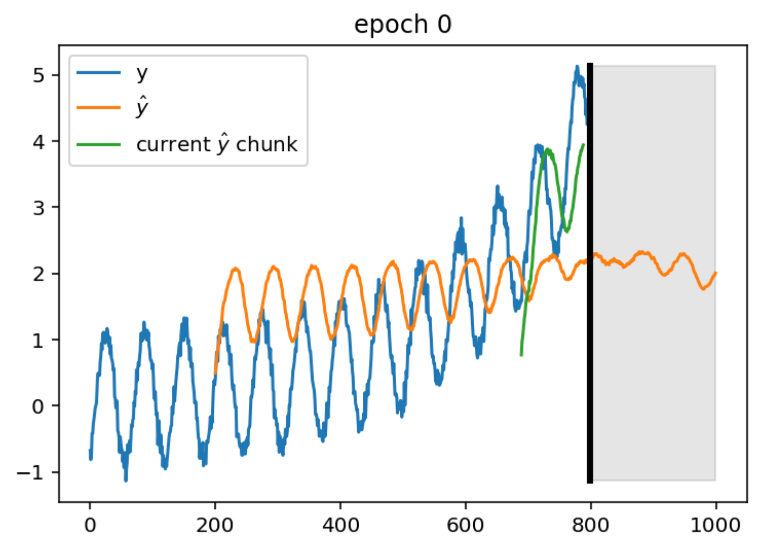


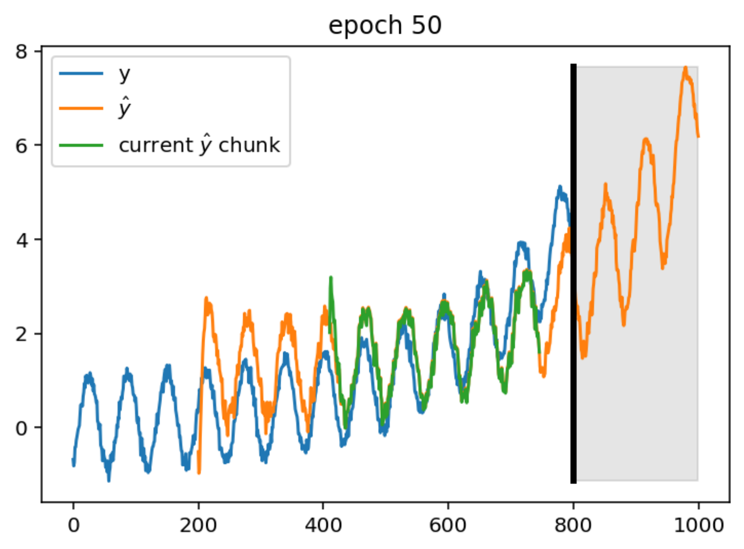


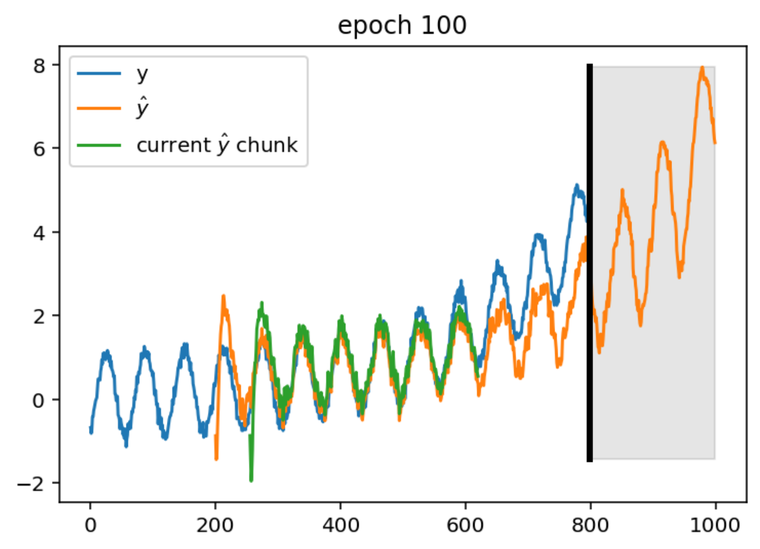
run **`testCell.py`**

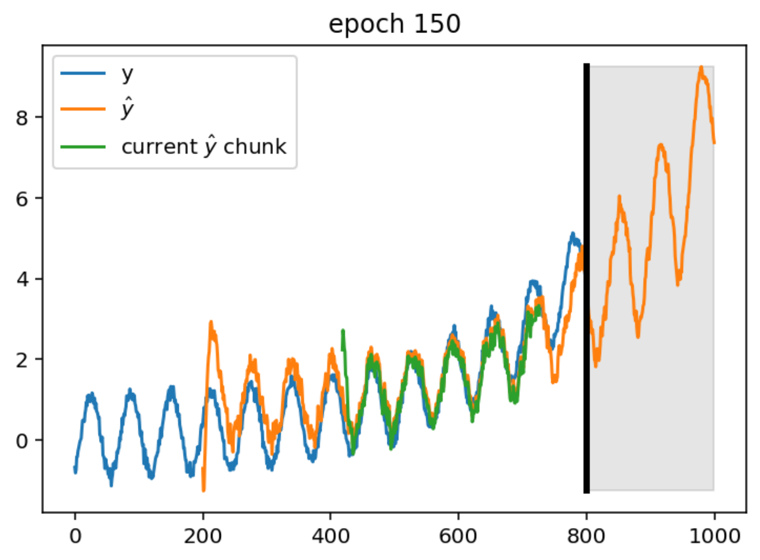
```
LSTM is running...
current MSSE = 0.888
current MSSE = 1.127
current MSSE = 0.619
current MSSE = 0.474
current MSSE = 0.389
current MSSE = 0.364
current MSSE = 0.440
current MSSE = 0.303
current MSSE = 0.366
current MSSE = 0.344
current MSSE = 0.295
current MSSE = 0.269
current MSSE = 0.238
current MSSE = 0.299
current MSSE = 0.191
current MSSE = 0.196
current MSSE = 0.264
current MSSE = 0.203
current MSSE = 0.182
current MSSE = 0.214
Done! MSSE = 0.140
```

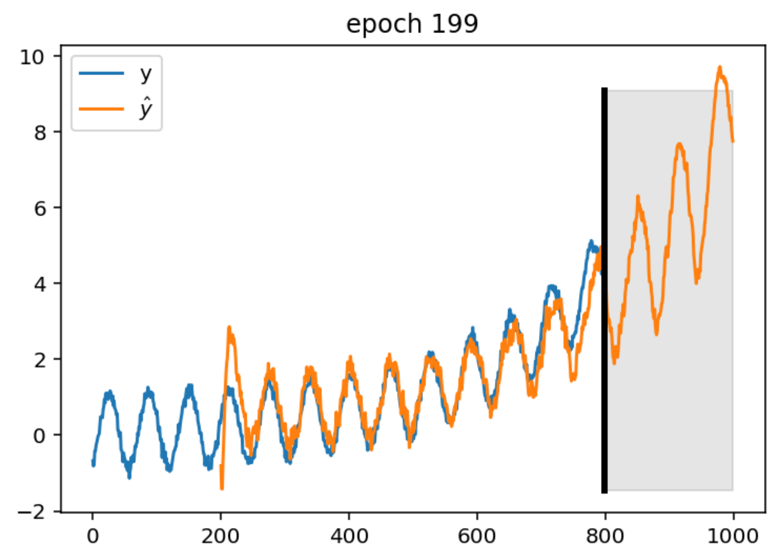
LSTM.py


epoch 0
y
ŷ
current ŷ chunk


epoch 50
y
ŷ
current ŷ chunk


epoch 100
y
ŷ
current ŷ chunk

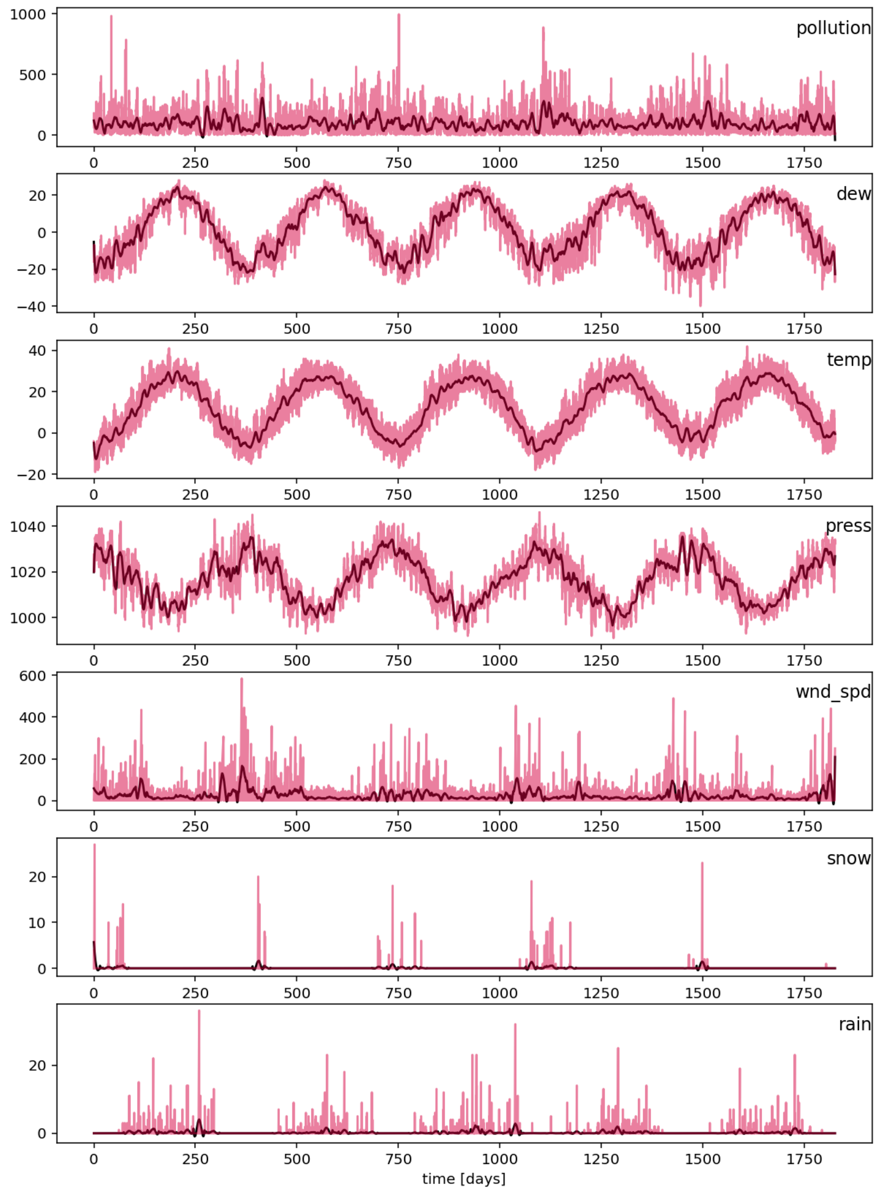

epoch 150
y
ŷ
current ŷ chunk

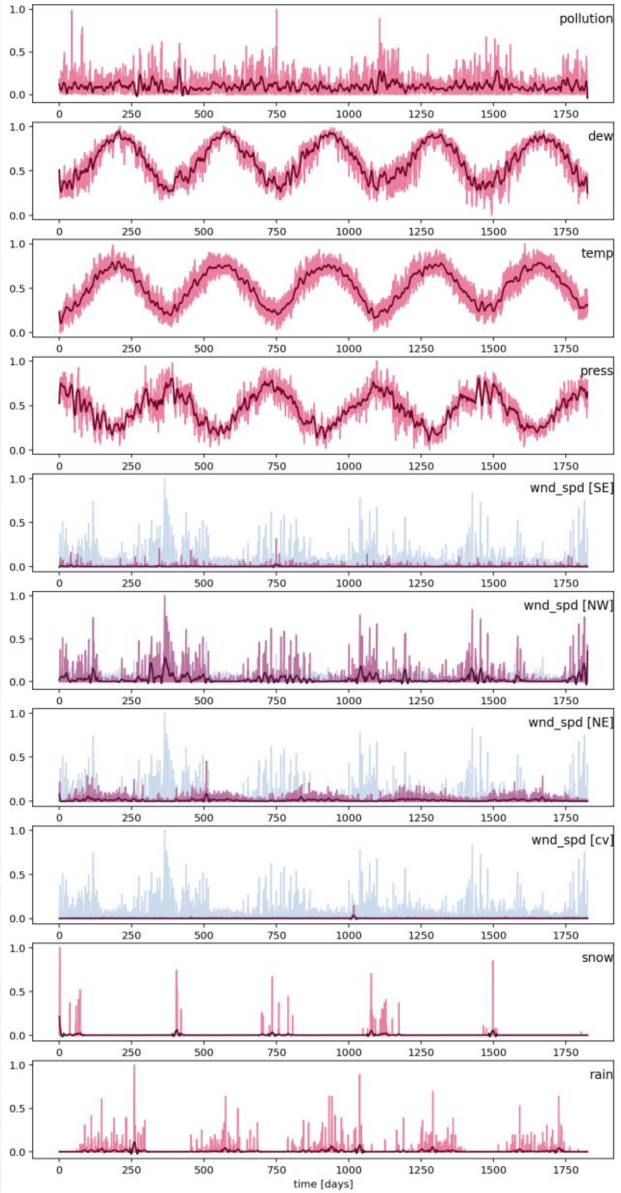

epoch 199
y
ŷ

RNNs & LSTMs: Syntax and some examples

`LSTM_keras.py` multi variate example taken from [here](): forecasting pollution from weather conditions

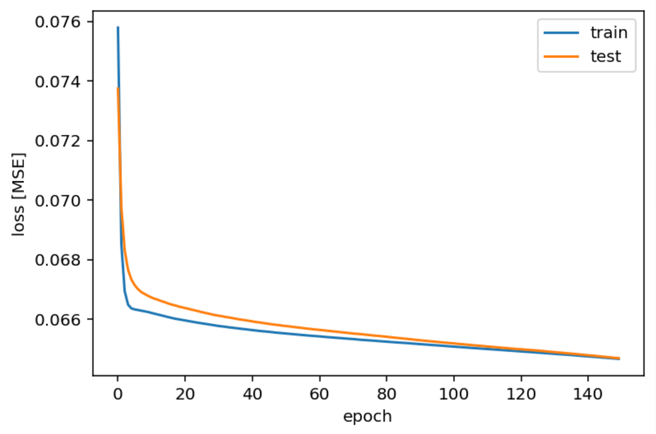
actual data
monthly moving average

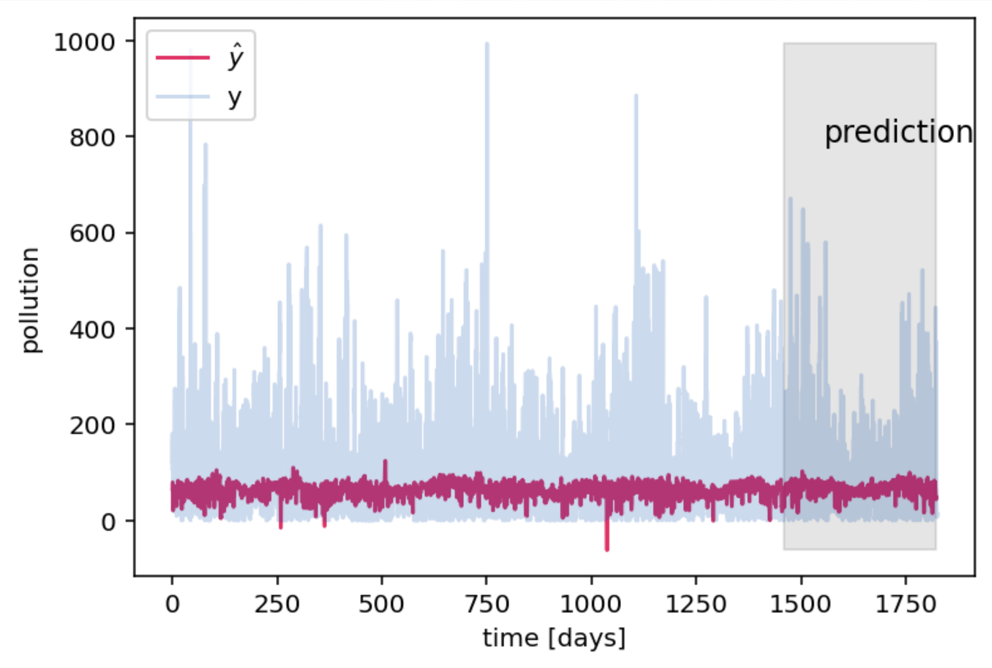


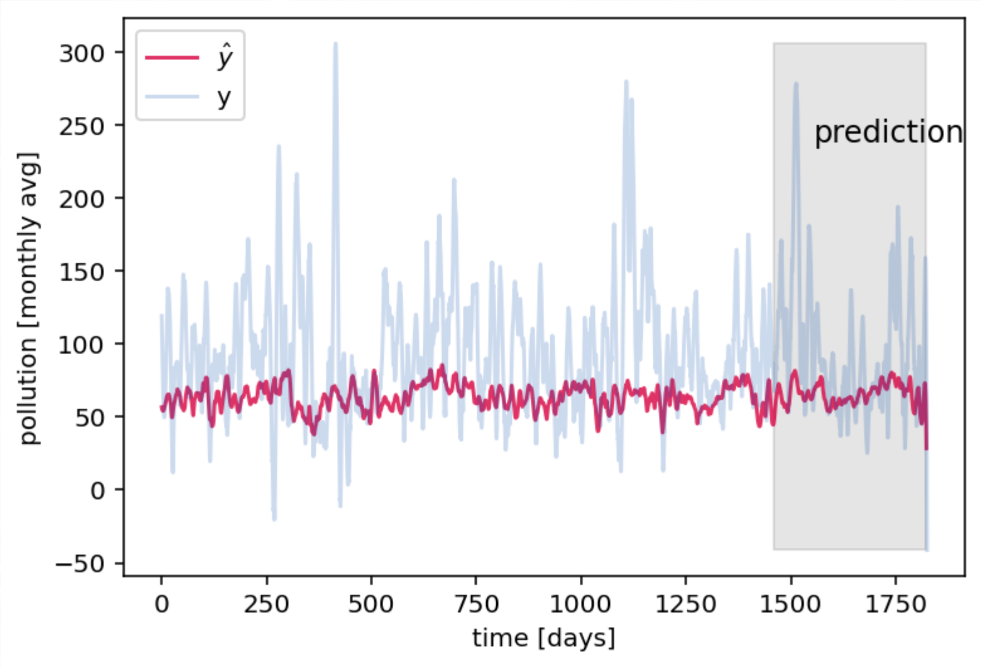


LSTM_keras.py multi variate example taken from here: forecasting pollution from weather conditions

150 neurons



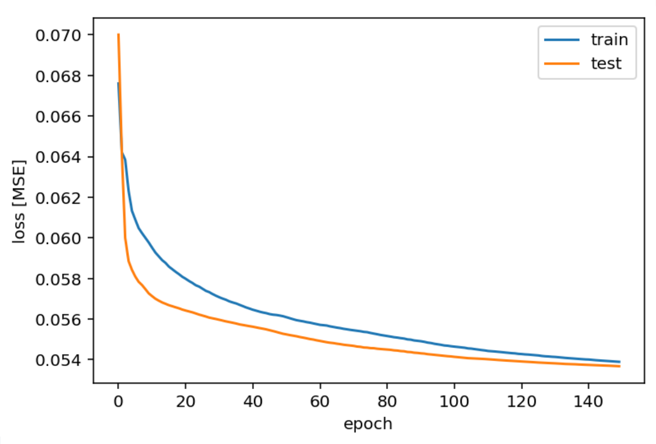


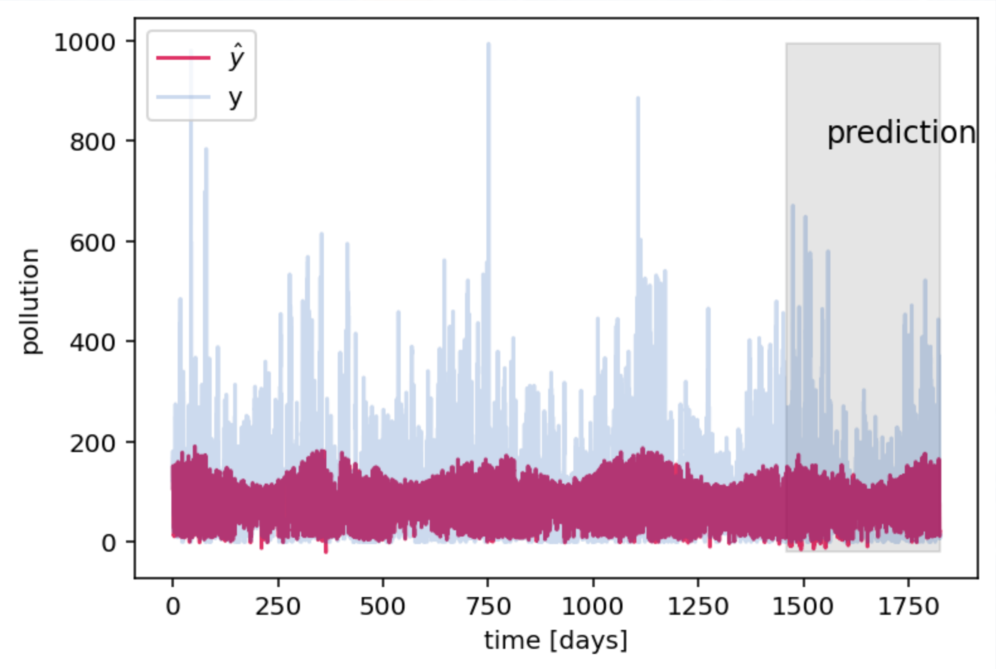


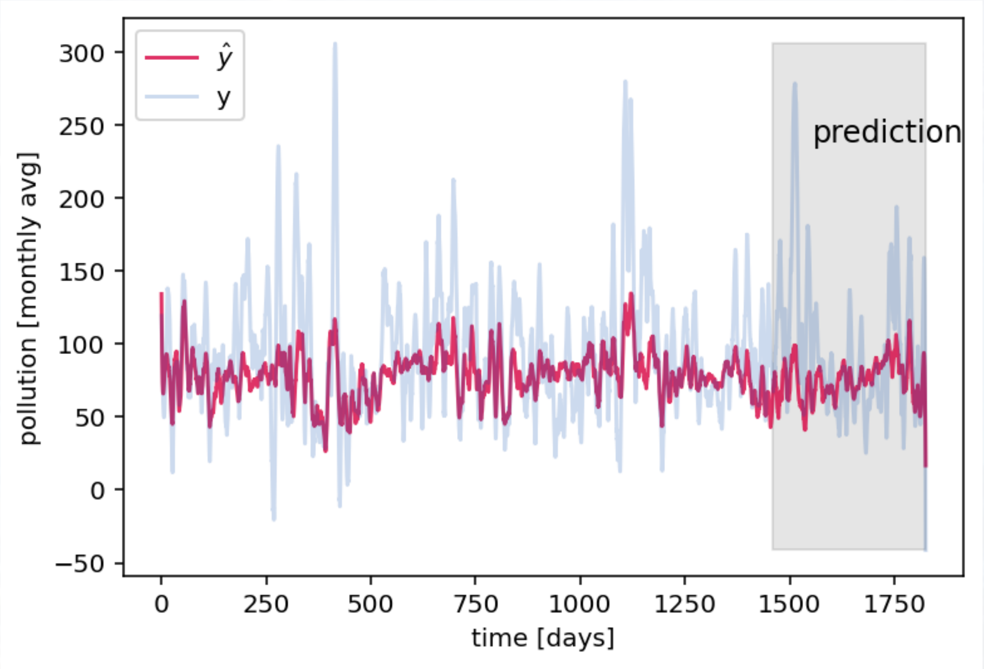
RNNs & LSTMs: Syntax and some examples

`LSTM_keras.py` multi variate example taken from here: forecasting pollution from weather conditions

500 neurons







Thank you very much for your attention!

https://www.analyticsvidhya.com