La Serena School for Data Science

Time Series Analysis 2024

Federica Bianco
University of Delaware
Rubin Observatory

this slide deck:

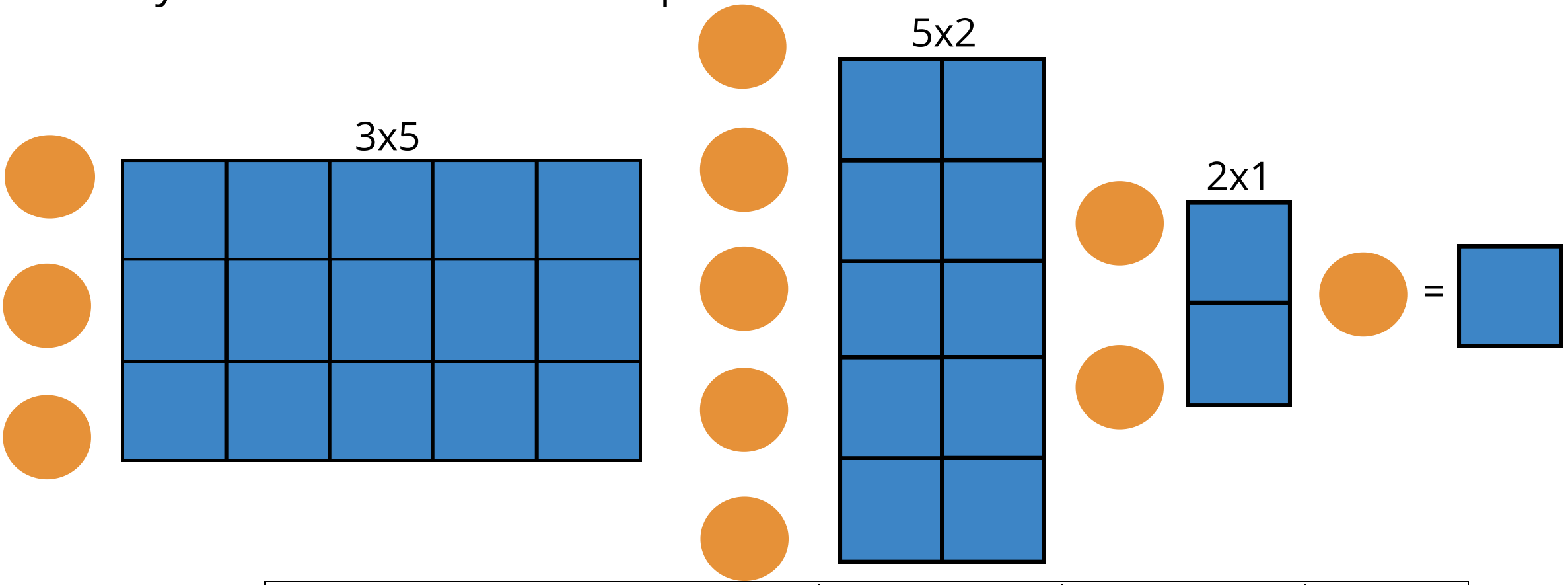
https://slides.com/federicabianco/LSDSS24_ts

1 Deep Learning

DeepNeuralNetwork

what we are doing, except for the activation function

is exactly a series of matrix multiplictions.



$$f^{(3)}(f^{(2)}(f^{(1)}(\vec{x} \cdot W_1 + \vec{b_1}) \cdot W_2 + \vec{b_2}) \cdot W_3 + \vec{b_3}) = y$$

DeepNeuralNetwork

what we are doing, except for the activation function
is exactly a series of matrix multiplictions.

The purpose is to
approximate a function φ

y = *φ(x)*

which (in general) is not linear
with linear operations

$$\phi(\vec{x}) \sim f^{(3)}(f^{(2)}(f^{(1)}(\vec{x} \cdot W_1 + \vec{b_1}) \cdot W_2 + \vec{b_2}) \cdot W_3 + \vec{b_3}) = y$$

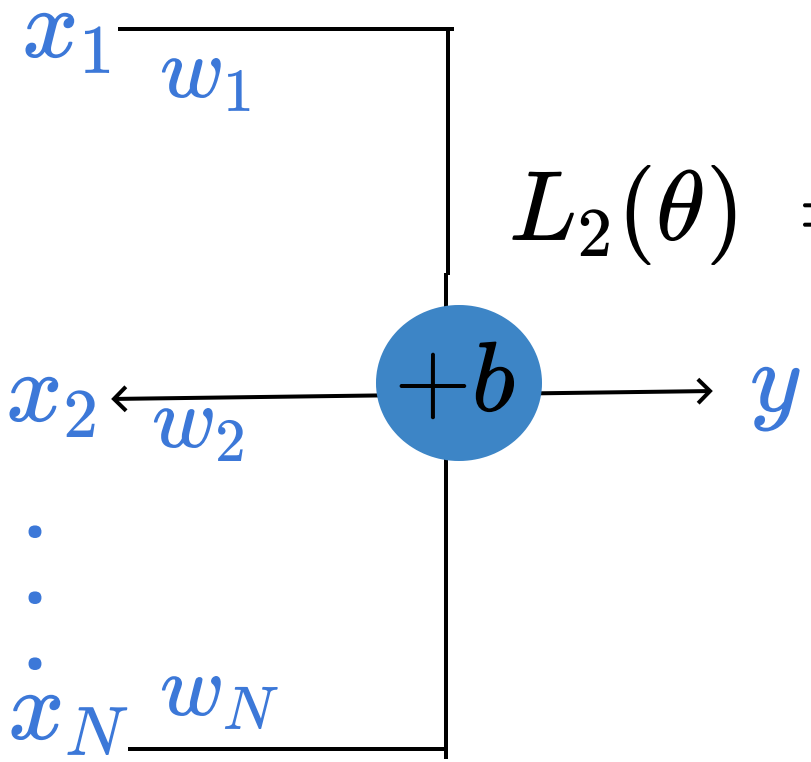
Gradient Descent

Find the best parameters by finding the minimum of the L2 hyperplane

y : prediction

y_{true} : target

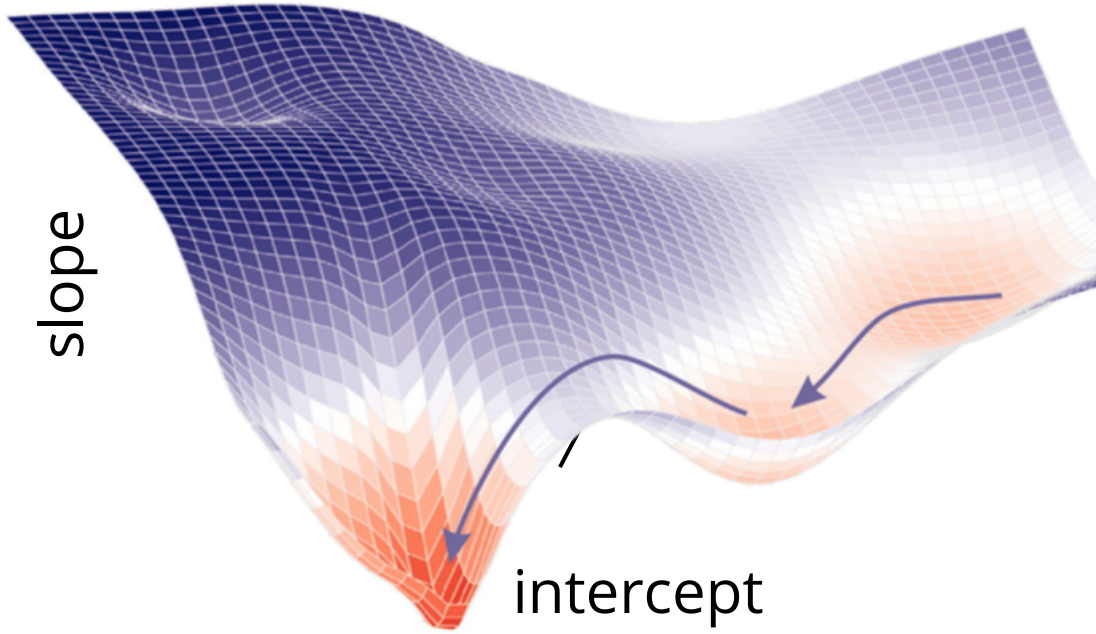
Any linear model:



Error: e.g.

$$L_2(\theta) = |y(\theta) - y_{\text{model}}|^2$$

at every step look around and choose the best direction



$$\vec{y} = \vec{x}W + b$$

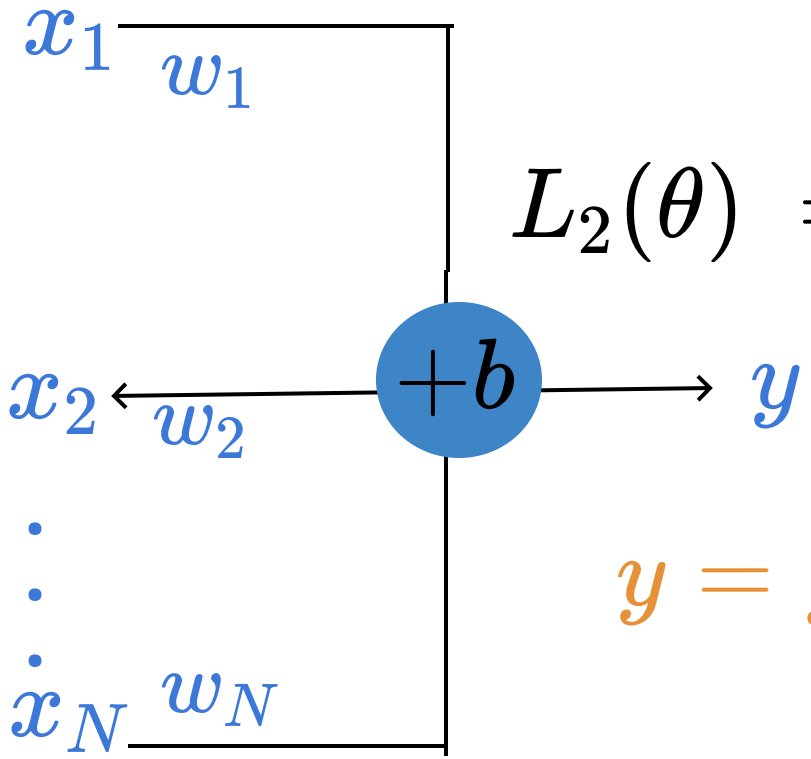
Gradient Descent

Find the best parameters by finding the minimum of the L2 hyperplane

Any linear model:

y : prediction

y$_{true}$: target



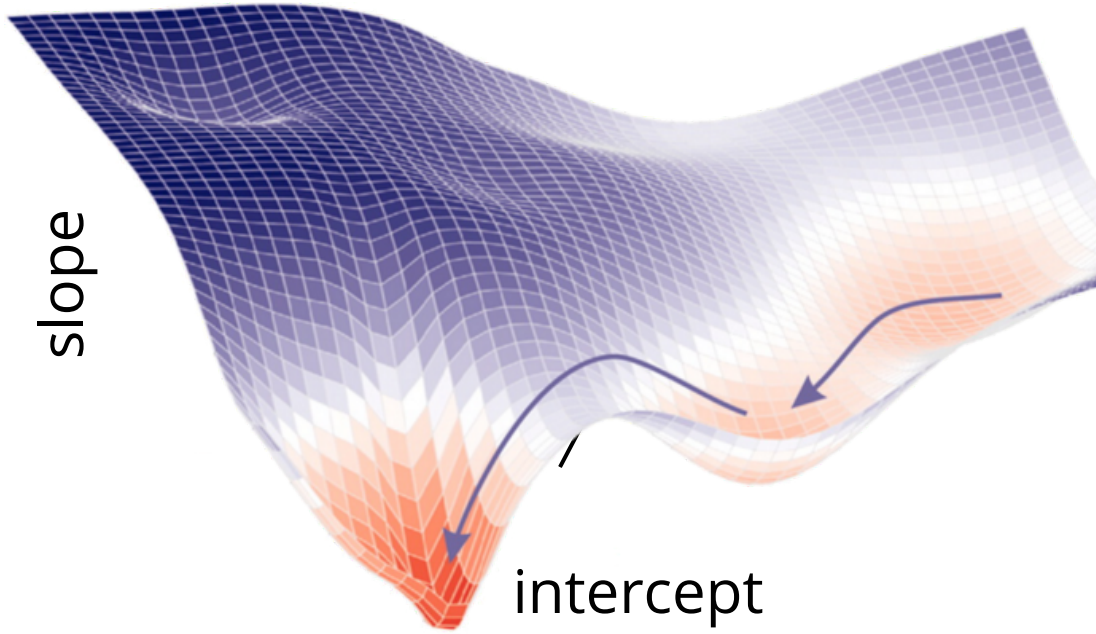
Error: e.g.

$$L_2(\theta) = |y(\theta) - y_{\text{model}}|^2$$

$$y = f(\sum \vec{w}\vec{x} + b)$$

$$p_2 = p_1 - e\,\delta f(p_1)$$

at every step look around and choose the best direction



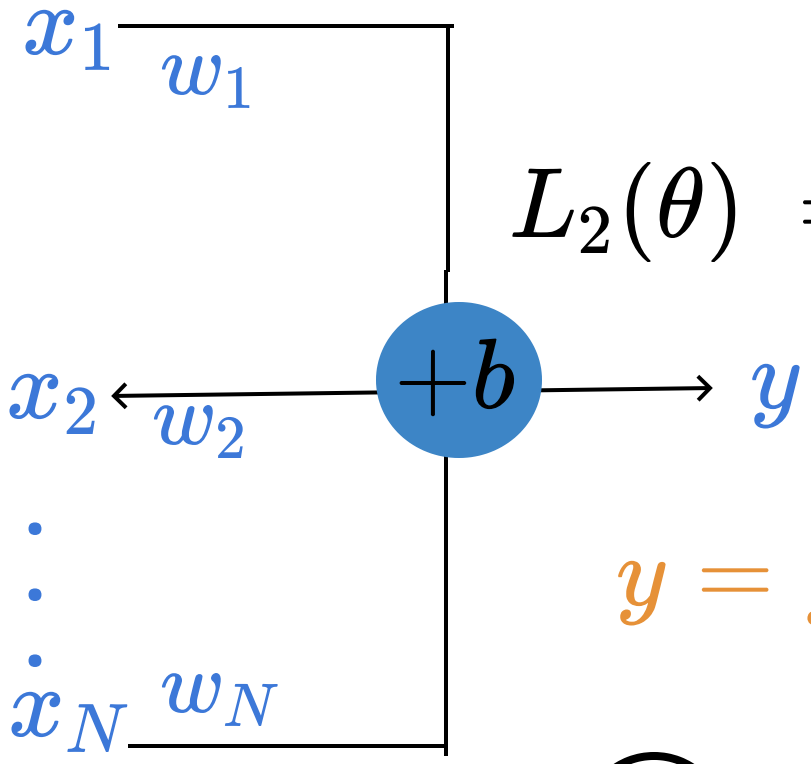
Gradient Descent

Find the best parameters by finding the minimum of the L2 hyperplane

Any linear model:

y : prediction

y$_{true}$: target



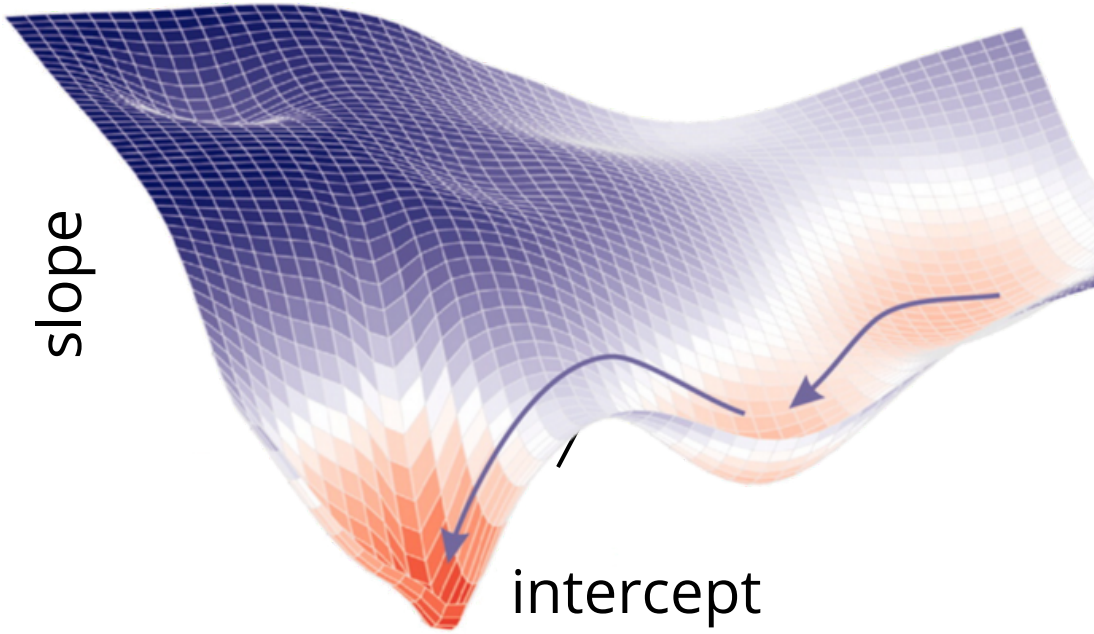
Error: e.g.

$$L_2(\theta) = |y(\theta) - y_{\text{model}}|^2$$

$$y = f(\sum \vec{w}\vec{x} + b)$$

new position $p_2 = p_1 - e\,\delta f(p_1)$

at every step look around and choose the best direction



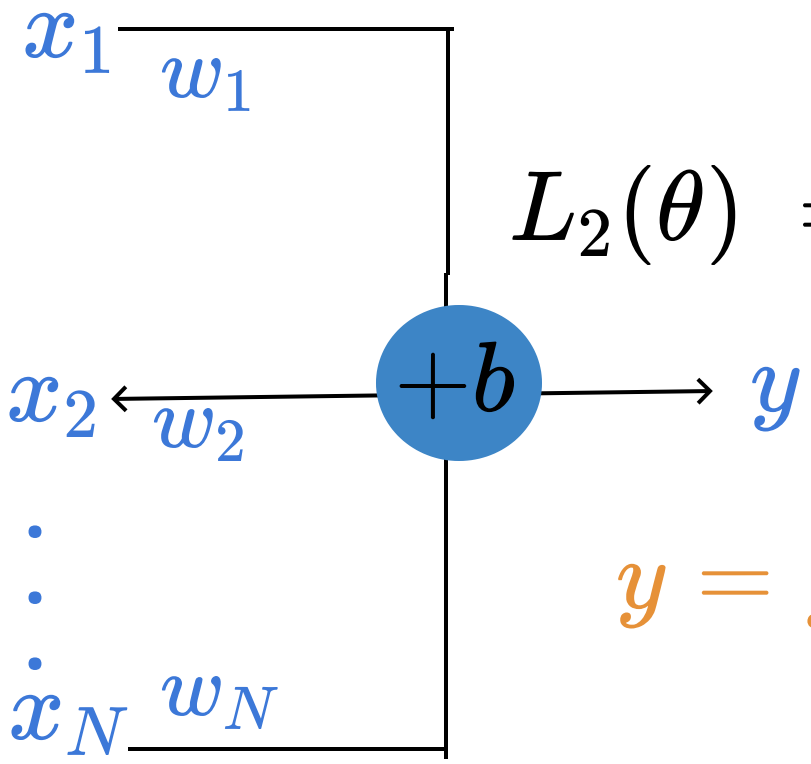
Gradient Descent

Find the best parameters by finding the minimum of the L2 hyperplane

Any linear model:

y : prediction

y_{true} : target



Error: e.g.

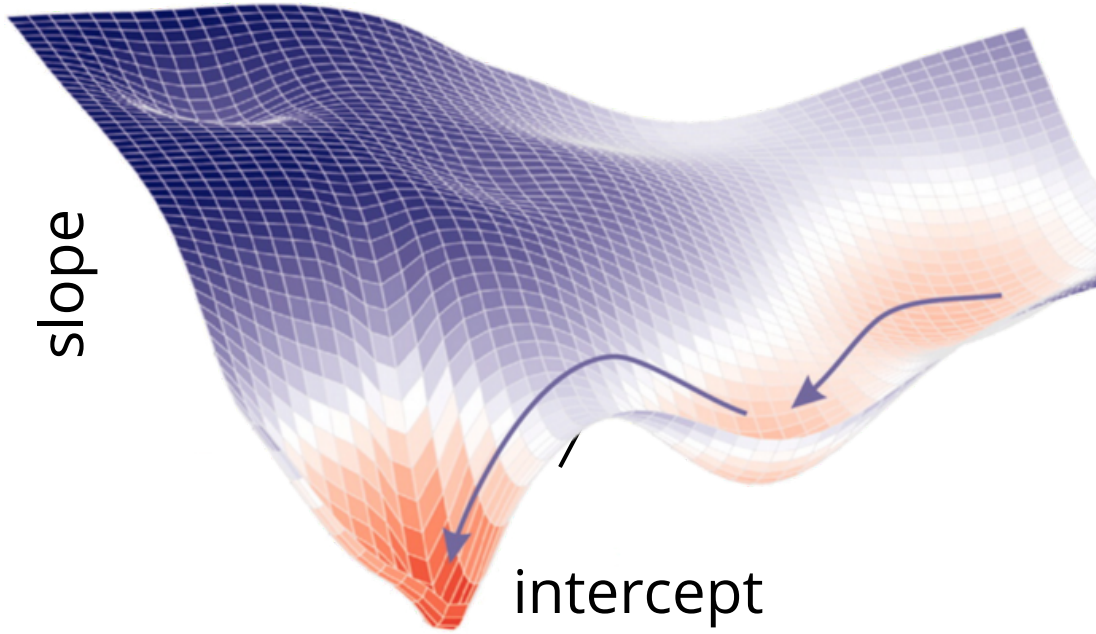
$$L_2(\theta) = |y(\theta) - y_{\text{model}}|^2$$

at every step look around and choose the best direction

$$y = f(\sum \vec{w}\vec{x} + b)$$

$$p_2 = p_1 - e\,\delta f(p_1)$$

old position



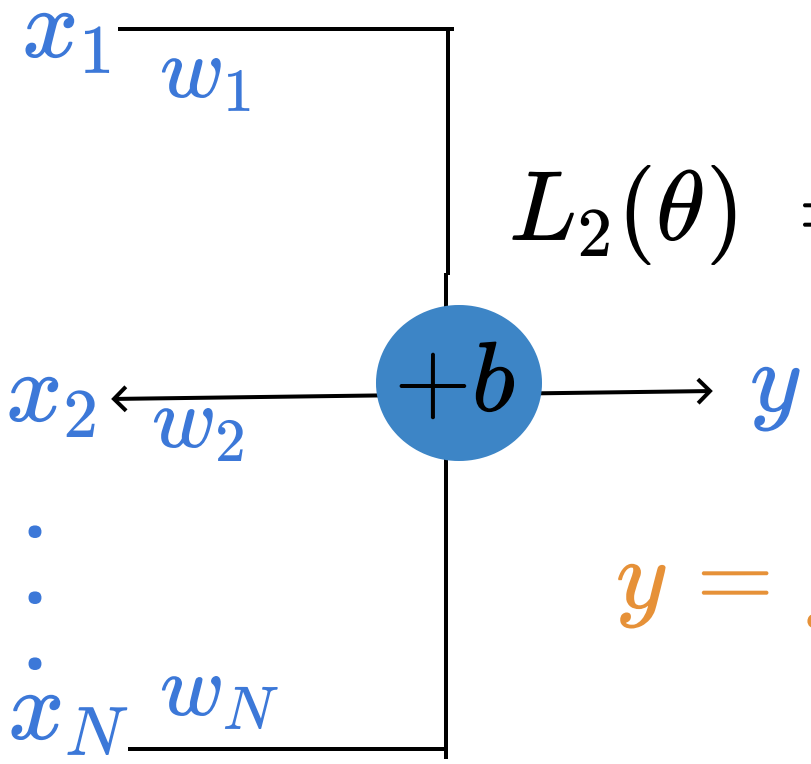
Gradient Descent

Find the best parameters by finding the minimum of the L2 hyperplane

Any linear model:

y : prediction

y$_{true}$: target



Error: e.g.

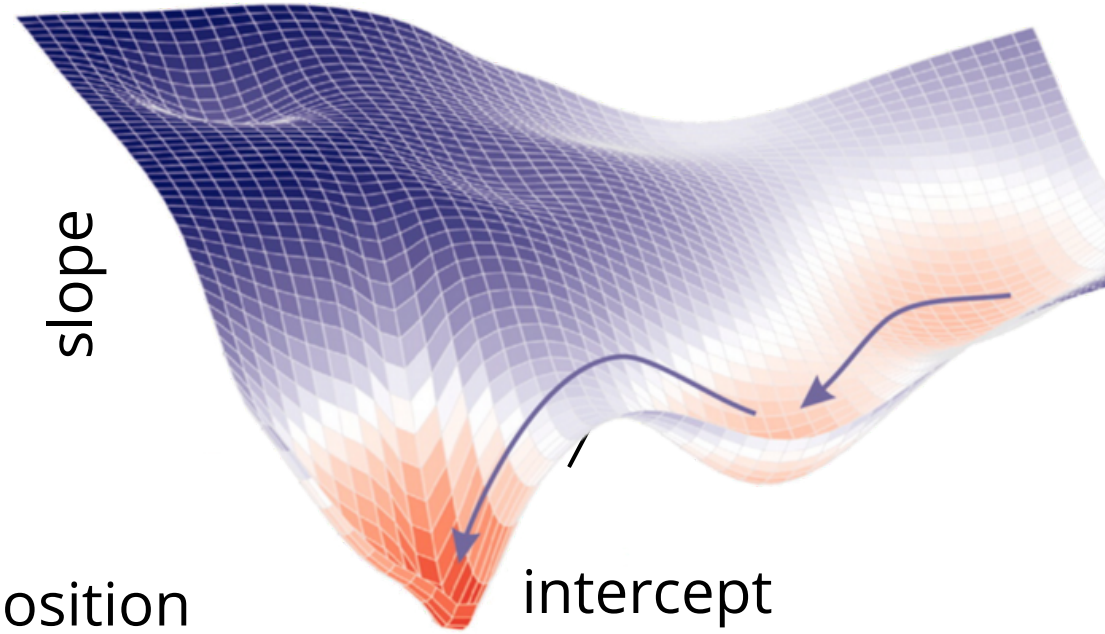
$$L_2(\theta) = |y(\theta) - y_{\text{model}}|^2$$

at every step look around and choose the best direction

$$y = f(\sum \vec{w}\vec{x} + b)$$

$$p_2 = p_1 - e\,\delta f(p_1)$$

gradient at the old position



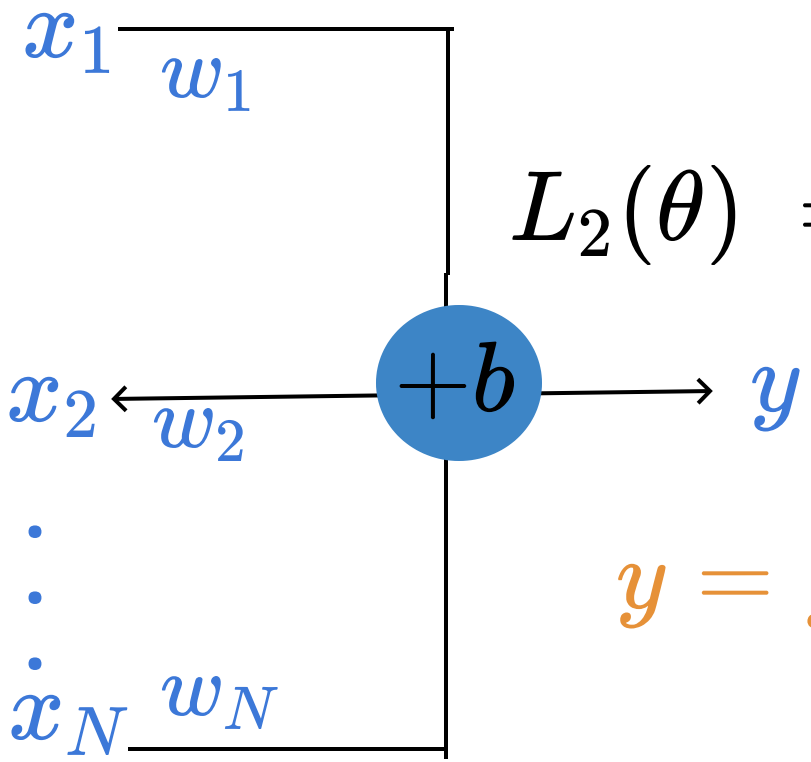
Gradient Descent

Find the best parameters by finding the minimum of the L2 hyperplane

Any linear model:

y : prediction

y$_{true}$: target



Error: e.g.

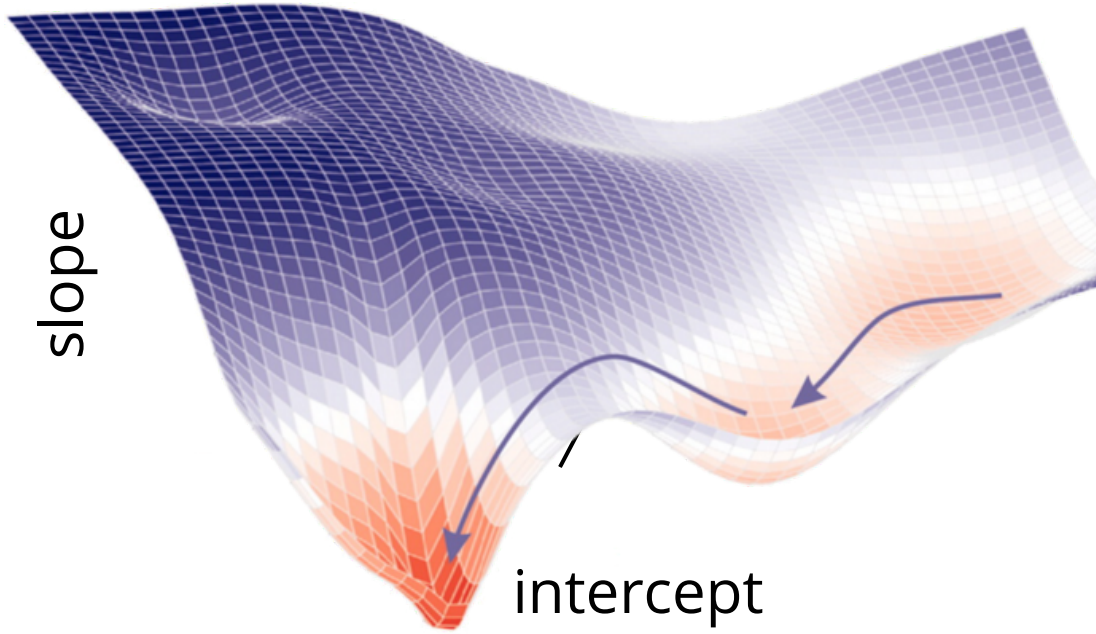
$$L_2(\theta) = |y(\theta) - y_{\text{model}}|^2$$

at every step look around and choose the best direction

$$y = f(\sum \vec{w}\vec{x} + b)$$

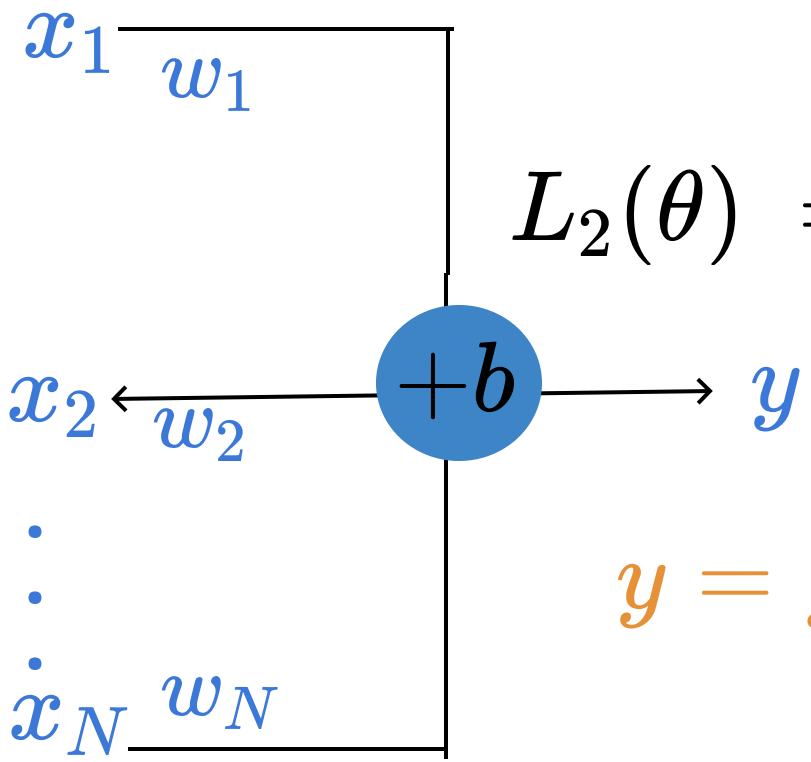
$$p_2 = p_1 - e\,\delta f(p_1)$$

learning rate



Gradient Descent

Any linear model:



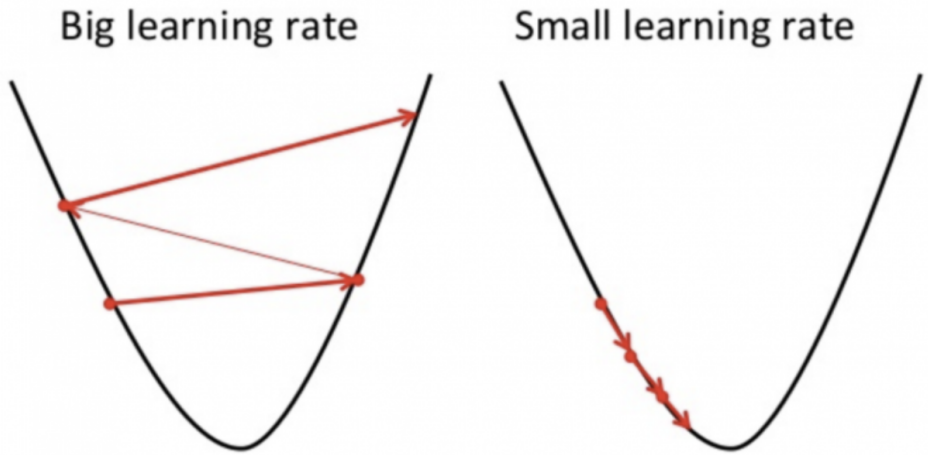
Error: e.g.

$$L_2(\theta) = |y(\theta) - y_{\text{model}}|^2$$

$$y = f(\sum \vec{w}\vec{x} + b)$$

$$p_2 = p_1 - e\,\delta f(p_1)$$

learning rate



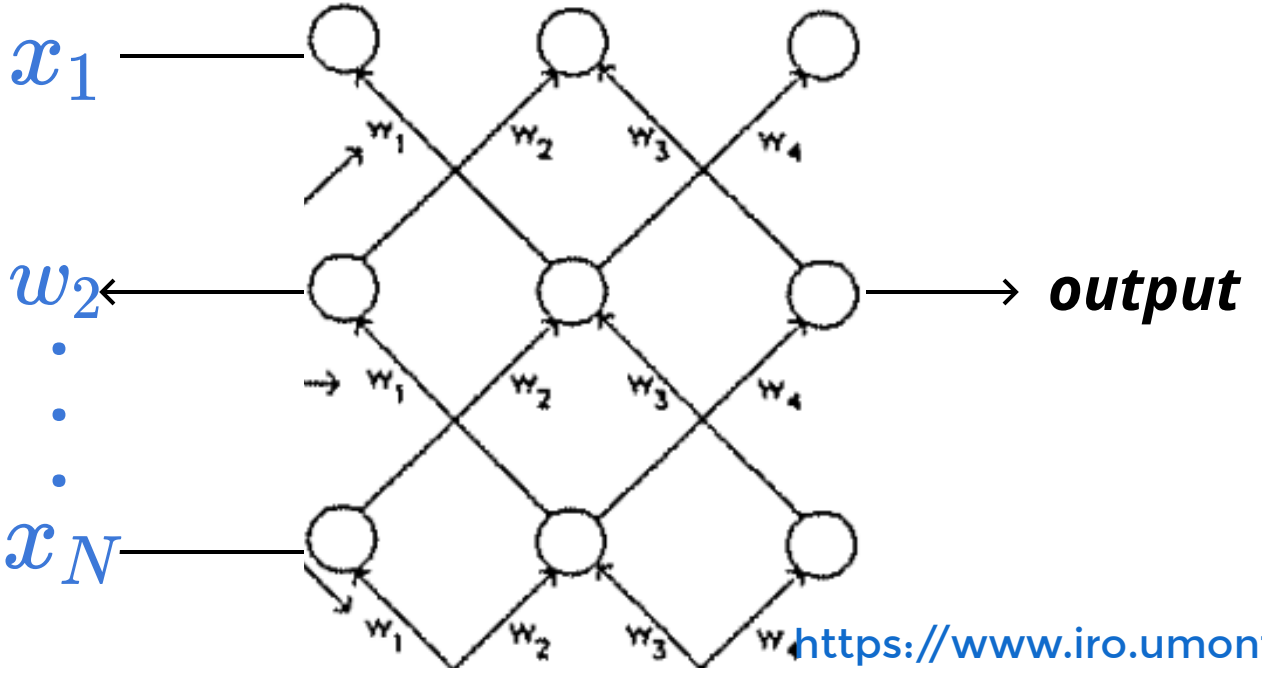
adaptive lr

$$e = e(L_2(\theta))$$

back-propagation

how does linear descent look when you have a whole network structure with hundreds of weights and biases to optimize??

$$x_j = \sum_i y_i w_{ji} \qquad y_j = \frac{1}{1+e^{-x_j}}$$



nature

Learning representations by back-propagating errors

David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams

Nature **323**, 533–536(1986) | Cite this article

22k Accesses | **7872** Citations | **167** Altmetric | Metrics

Abstract

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

https://www.iro.umontreal.ca/~vincentp/ift3395/lectures/backprop_old.pdf

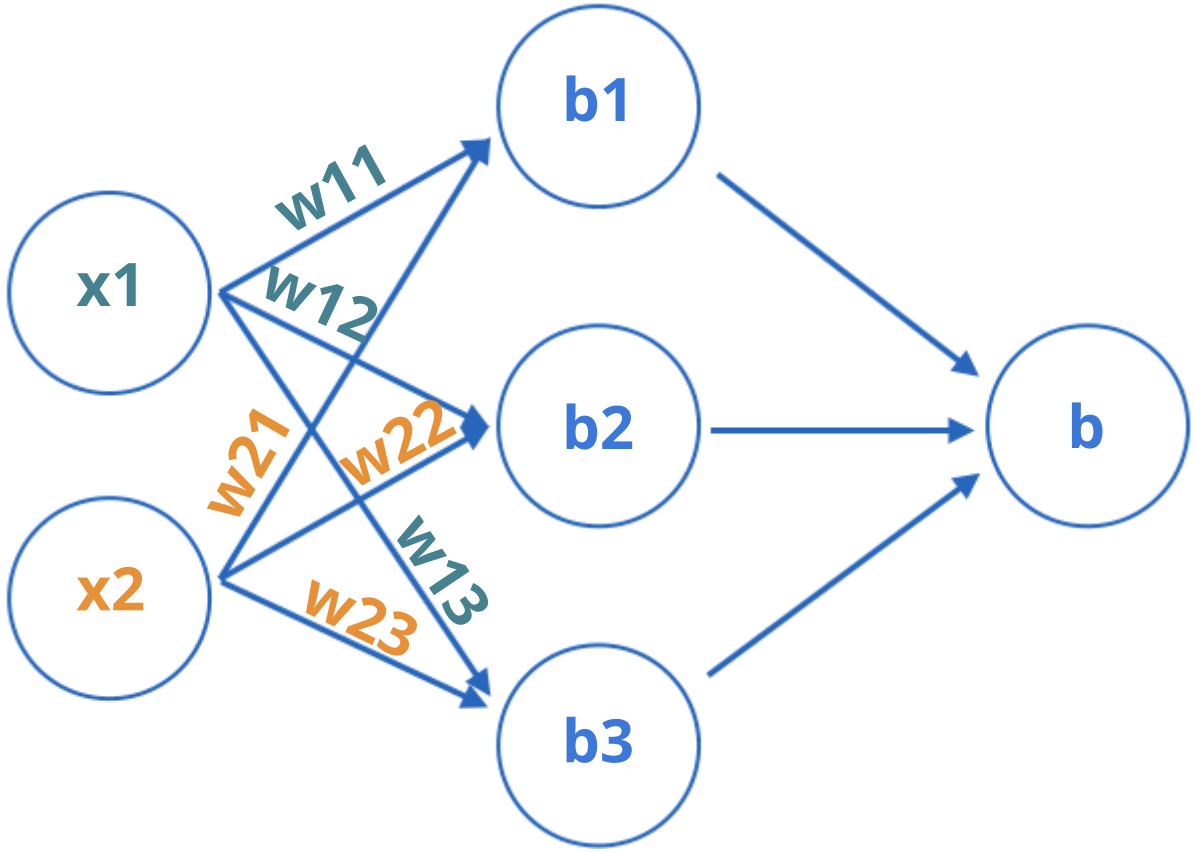
Training models with this many parameters requires a lot of care:

. defining the metric

. optimization schemes

. training/validation/testing sets

But just like our simple linear regression case, **small changes in the parameters lead to small changes in the output** for the right activation functions.



define a cost function, e.g.

$$C = \frac{1}{2}|y - a^L|^2 = \frac{1}{2}\sum_j (y_j - a_j^L)^2$$

$$\text{output} = \frac{1}{1+e^{-\frac{w_7}{1+e^{-w_1x_1-w_4x_2-b_1}}-\frac{w_8}{1+e^{-w_2x_1-w_5x_2-b_2}}-\frac{w_9}{1+e^{-w_3x_1-w_6x_2-b_3}}-b_4}}$$

$$\vec{y} = f_N(....(f_1(\vec{x}W_i + b_1 ... W_N + b_N)))$$

Training models with this many parameters requires a lot of care:

. defining the metric

. optimization schemes

. training/validation/testing sets

But just like our simple linear regression case, **small changes in the parameters lead to small changes in the output** for the right activation functions.

$$z = z(y)$$

$$y = y(x)$$

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx},$$

define a cost function, e.g.

$$C = \tfrac{1}{2}|y - a^L|^2 \; = \; \tfrac{1}{2}\sum_j (y_j - a_j^L)^2$$

$$\text{output} = \frac{1}{1+e^{-\frac{w_7}{1+e^{-w_1x_1-w_4x_2-b_1}}-\frac{w_8}{1+e^{-w_2x_1-w_5x_2-b_2}}-\frac{w_9}{1+e^{-w_3x_1-w_6x_2-b_3}}-b_4}}$$

$$\vec{y} = f_N(....(f_1(\vec{x}W_i + b_1 ... W_N + b_N)))$$

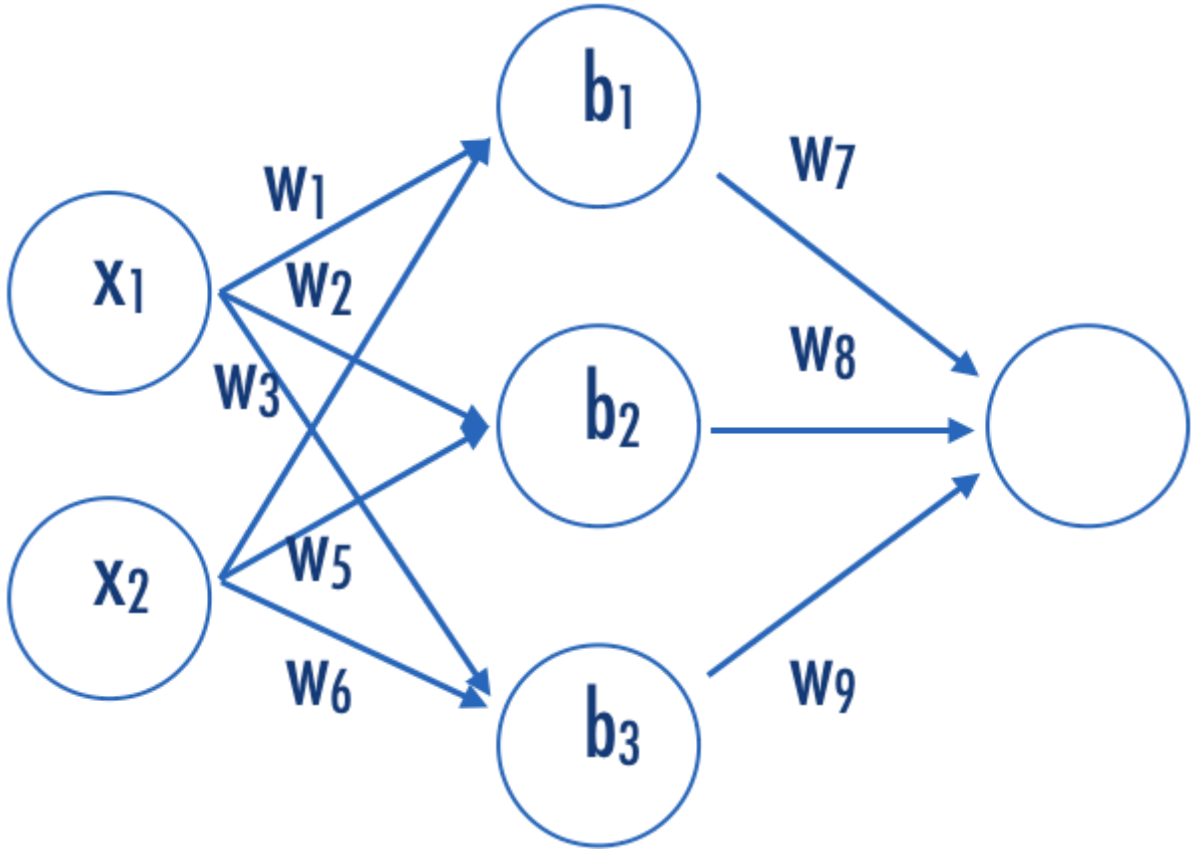
Training models with this many parameters requires a lot of care:

. defining the metric

. optimization schemes

. training/validation/testing sets

But just like our simple linear regression case, the fact that small changes in the parameters leads to small changes in the output for the right activation functions.



define a cost function, e.g.

$$\text{output} = \frac{1}{1+e^{-\frac{w_7}{1+e^{-w_1x_1-w_4x_2-b_1}}-\frac{w_8}{1+e^{-w_2x_1-w_5x_2-b_2}}-\frac{w_9}{1+e^{-w_3x_1-w_6x_2-b_3}}-b_4}}$$

$$C = \frac{1}{2}|y - a^L|^2 = \frac{1}{2}\sum_j (y_j - a_j^L)^2$$

$$\vec{y} = f_N(....(f_1(\vec{x}W_i + b_1 ... W_N + b_N)))$$

Training a DNN

Training models with this many parameters requires a lot of care:

. defining the metric

. optimization schemes

. training/validation/testing sets

But just like our simple linear regression case, the fact that small changes in the parameters leads to small changes in the output for the right activation functions.

define a cost function, e.g.

$$C = \frac{1}{2}|y - a^L|^2 = \frac{1}{2}\sum_j (y_j - a_j^L)^2$$

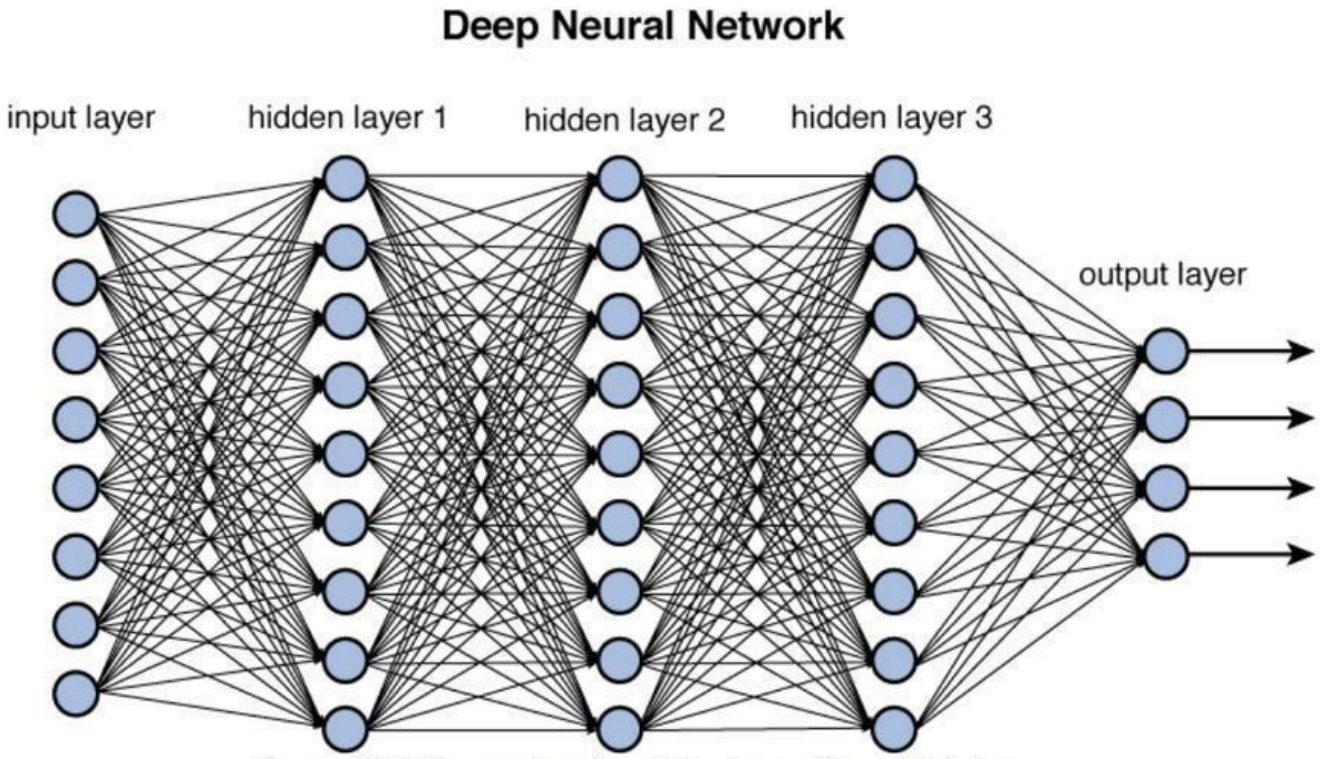


Figure 12.2 Deep network architecture with multiple layers.

feed data forward through network and calculate cost metric

for each layer, calculate effect of small changes on next layer

$$\vec{y} = f_N(....(f_1(\vec{x}W_i + b_1 ... W_N + b_N)))$$

Training a DNN

back-propagation

how does linear descent look when you have a whole network structure with hundreds of weights and biases to optimize??

think of applying just gradient to a function of a function... use:

1) partial derivatives, 2) chain rule

define a cost function, e.g. $C = \frac{1}{2}|y - a^L|^2 = \frac{1}{2}\sum_j (y_j - a_j^L)^2$

An equation for the error in the output layer, δ^L: The components of δ^L are given by

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L). \tag{BP1}$$

matrix-based form, as

$$\delta^L = \nabla_a C \odot \sigma'(z^L). \tag{BP1a}$$

Here, $\nabla_a C$ is defined to be a vector whose components are the partial derivatives $\partial C/\partial a_j^L$. You can think of $\nabla_a C$ as expressing the rate of change of C with respect to the output activations

The backpropagation equations provide us with a way of computing the gradient of the cost function. Let's explicitly write this out in the form of an algorithm:

1. **Input x:** Set the corresponding activation a^1 for the input layer.
2. **Feedforward:** For each $l = 2, 3, \ldots, L$ compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$.
3. **Output error δ^L:** Compute the vector $\delta^L = \nabla_a C \odot \sigma'(z^L)$.
4. **Backpropagate the error:** For each $l = L-1, L-2, \ldots, 2$ compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$.
5. **Output:** The gradient of the cost function is given by $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l$ and $\frac{\partial C}{\partial b_j^l} = \delta_j^l$.

Examining the algorithm you can see why it's called *back*propagation. We compute the error vectors δ^l backward, starting from the final layer. It may seem peculiar that we're going

http://neuralnetworksanddeeplearning.com/chap2.html

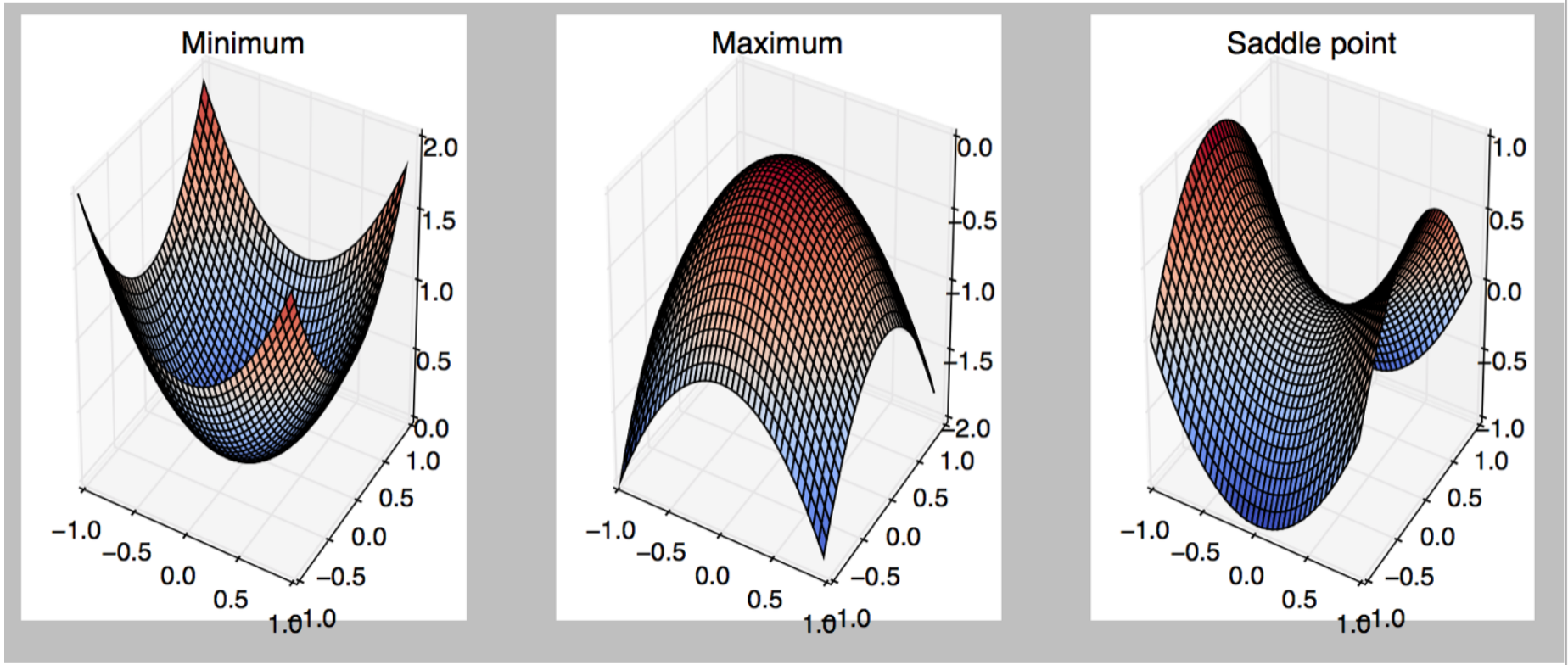
$$\vec{y} = f_N(....(f_1(\vec{x}W_i + b_1 ... W_N + b_N)))$$

Gradient Descent

at every step look around and choose the best direction

Critical points

Zero gradient, and Hessian with…



All positive eigenvalues All negative eigenvalues

Some positive and some negative

why do we not worry about local minima?

the course of dimensionality actually is a blessing here!

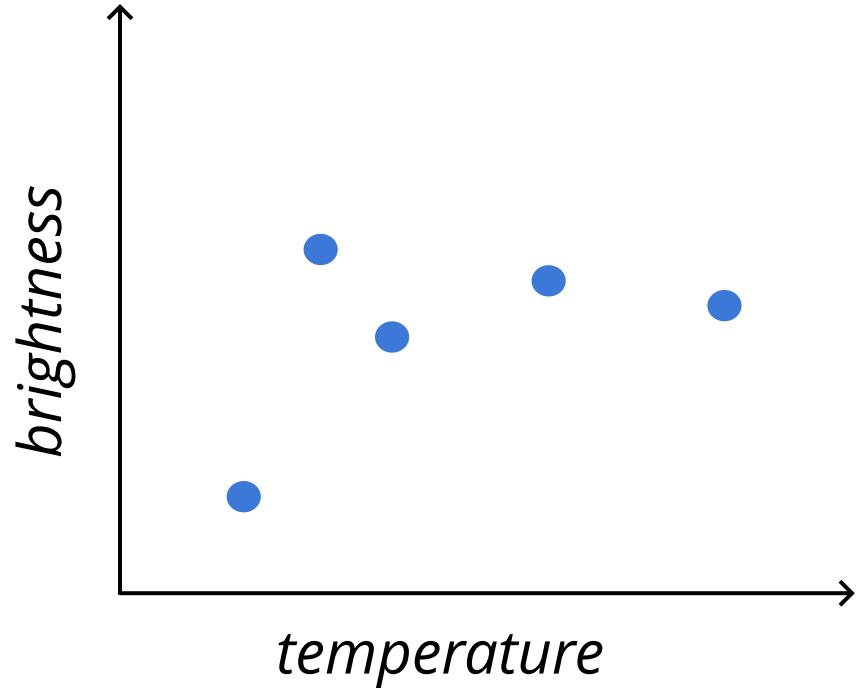
2 Time series analysis

what is a time series?

Consider a dataset that is a time series

1D: exogenous-endogenous variable

y depend on *x*

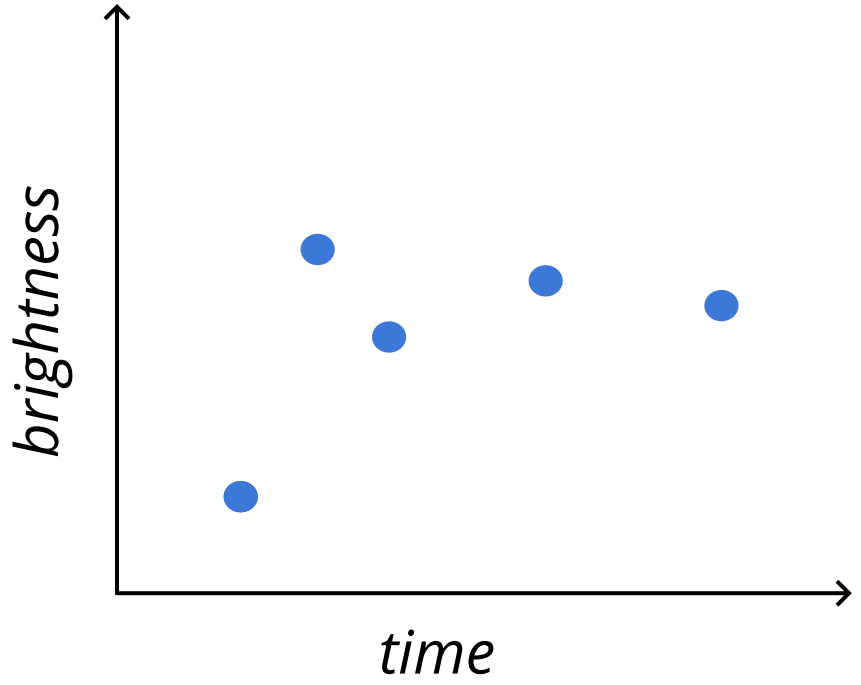


what is a time series?

Consider a dataset that is a time series

1D: exogenous-endogenous variable

y depend on *x*



what is a time series?

Consider a dataset that is a time series

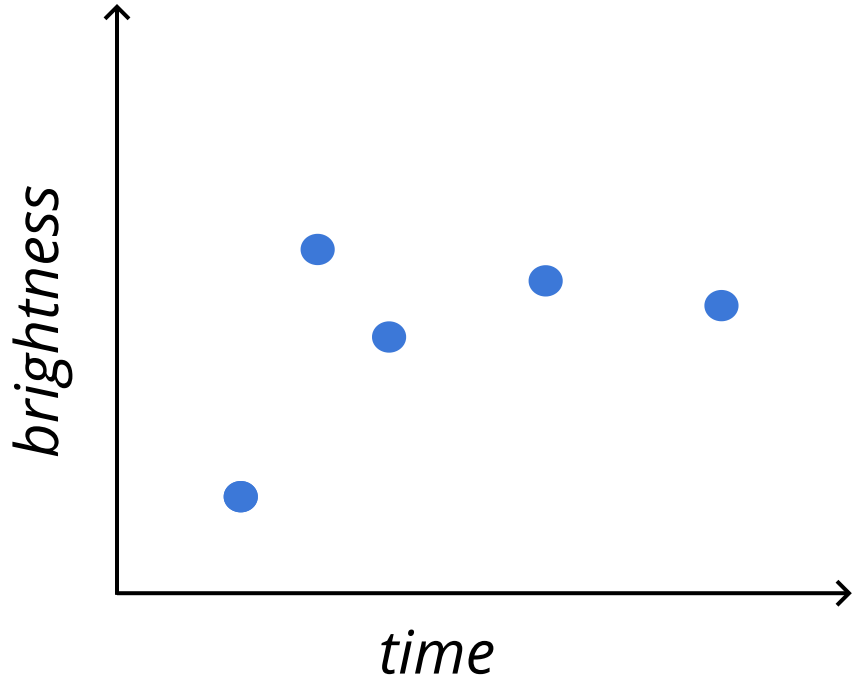
1D: exogenous-endogenous variable

y depend on *x*

exogenous variable is sequencial

time has an directionality:

y(t+1) depends on y(t)



what is a time series?

A time series is any measurable quantity sampled at multiple points in time.

Time series are series of exogenous-endogenous variable pairs where the exogenous variable is time, and therefore it is a sequential quantity with a specific direction of evolution.

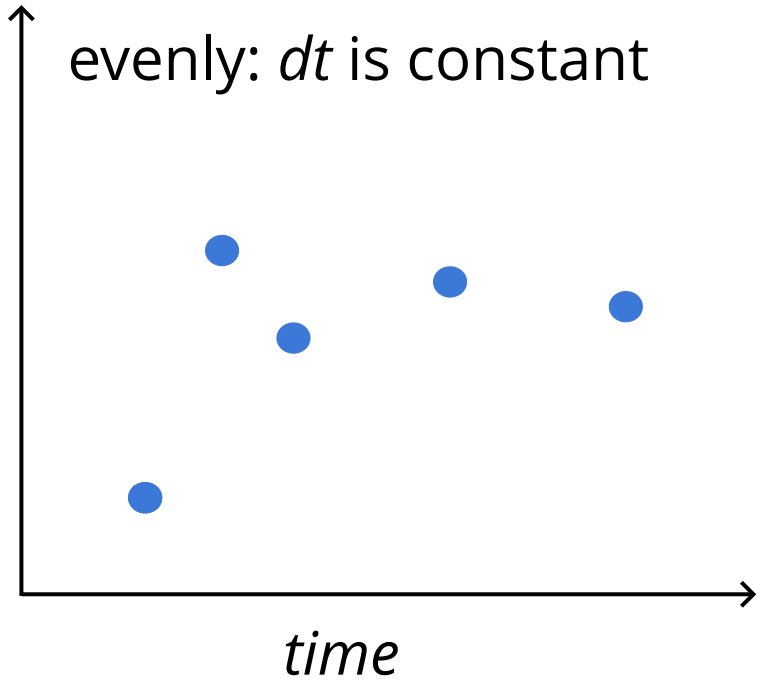
Key Concept

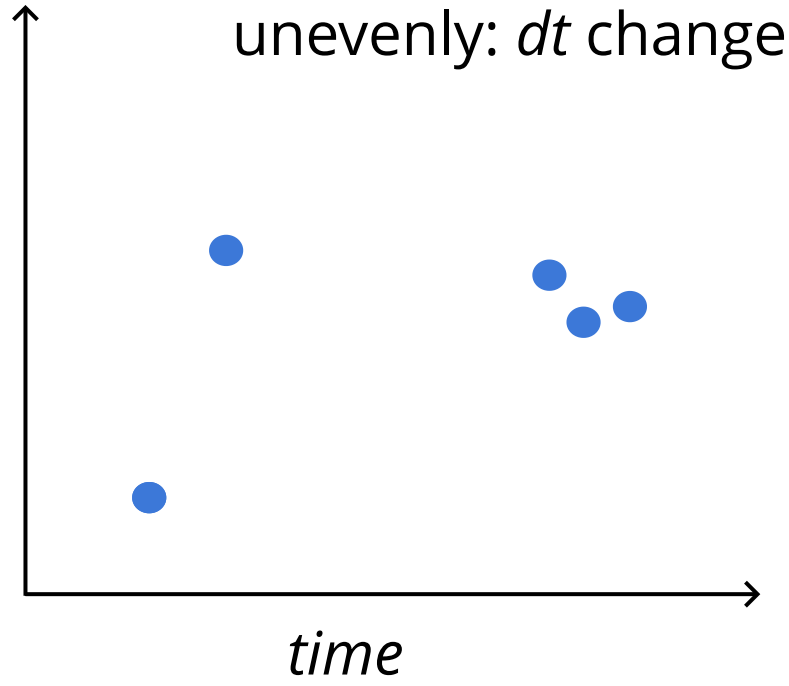
what is a time series?

Evenly vs Unevenly sampled time series.

Most statistical methods are developed for evenly sampled TS.

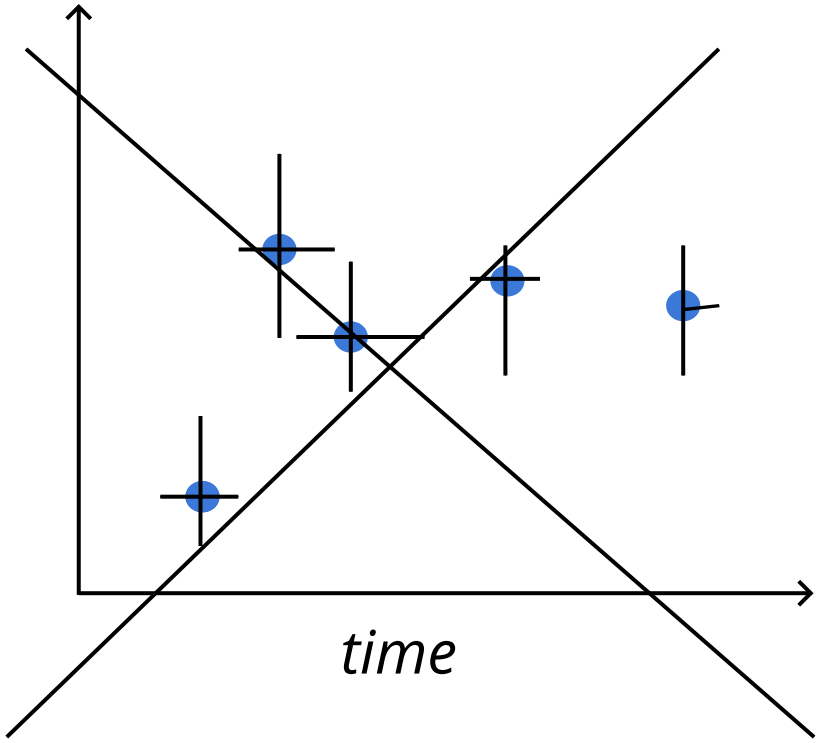
Most physical TS are unevenly sampled

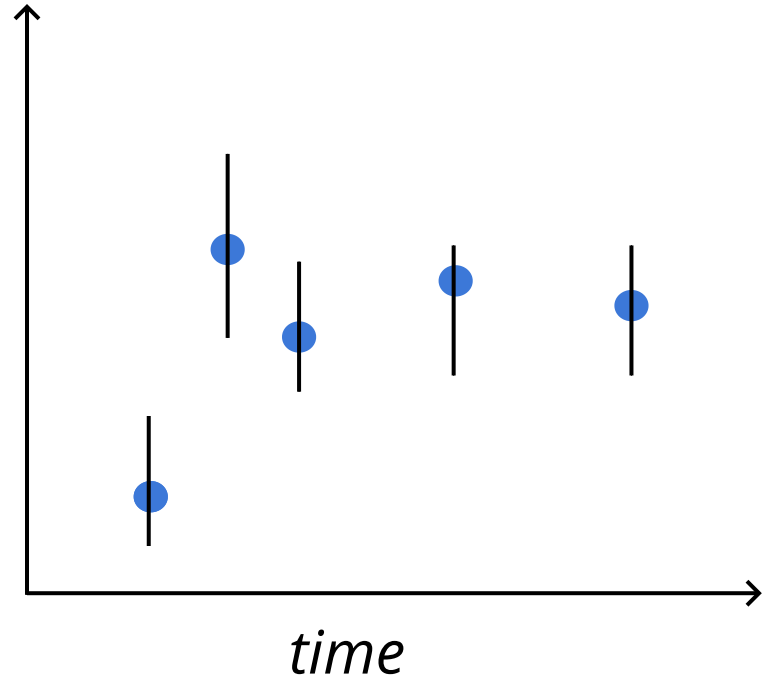




what is a time series?

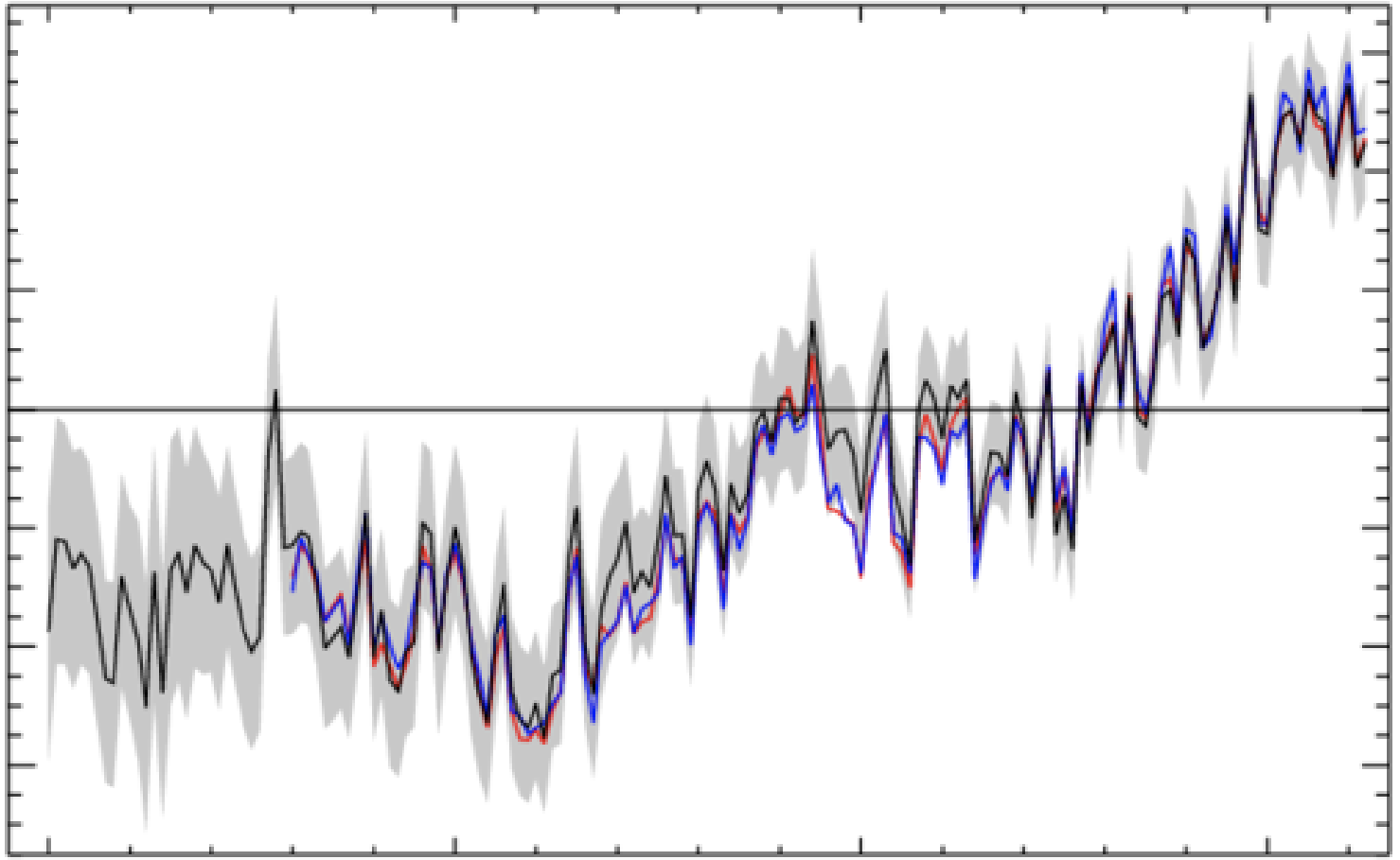
Unusally time of sampling is known





Time Series

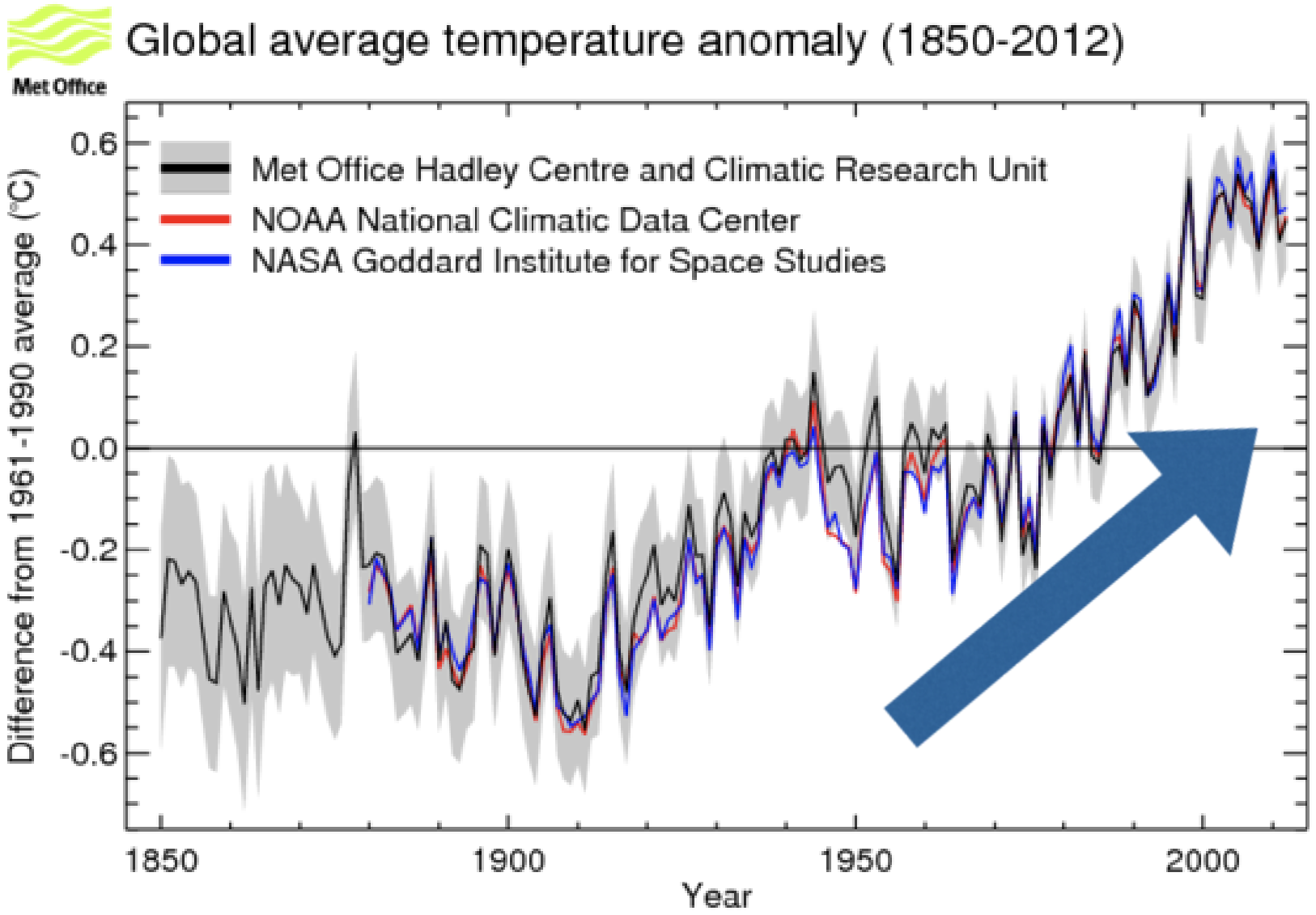
what is interesting in this time series?



Time Series

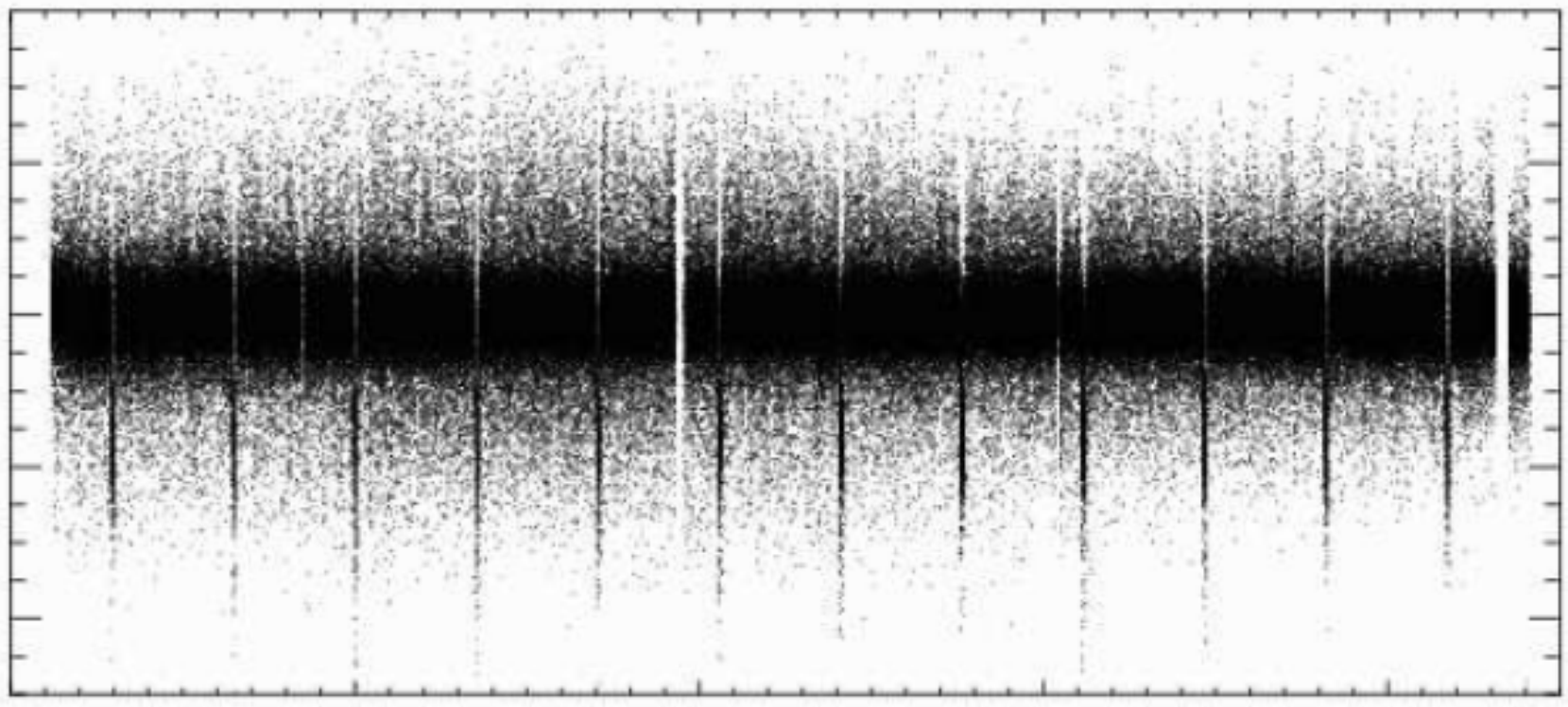
what is interesting in this time series?

trend



Time Series

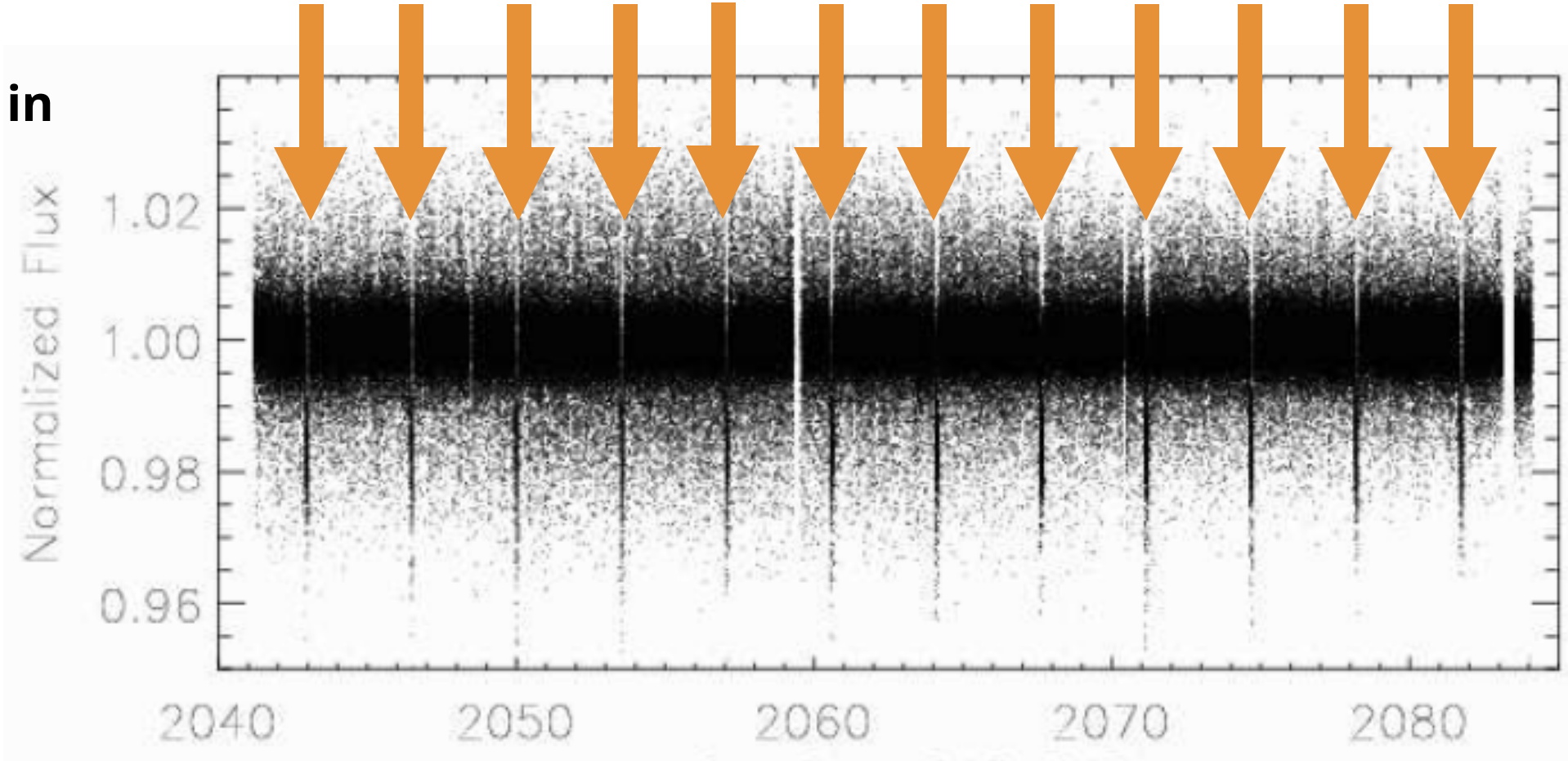
what is interesting in this time series?



Time Series

what is interesting in this time series?

HD 209458, the first transiting planet to be discovered.

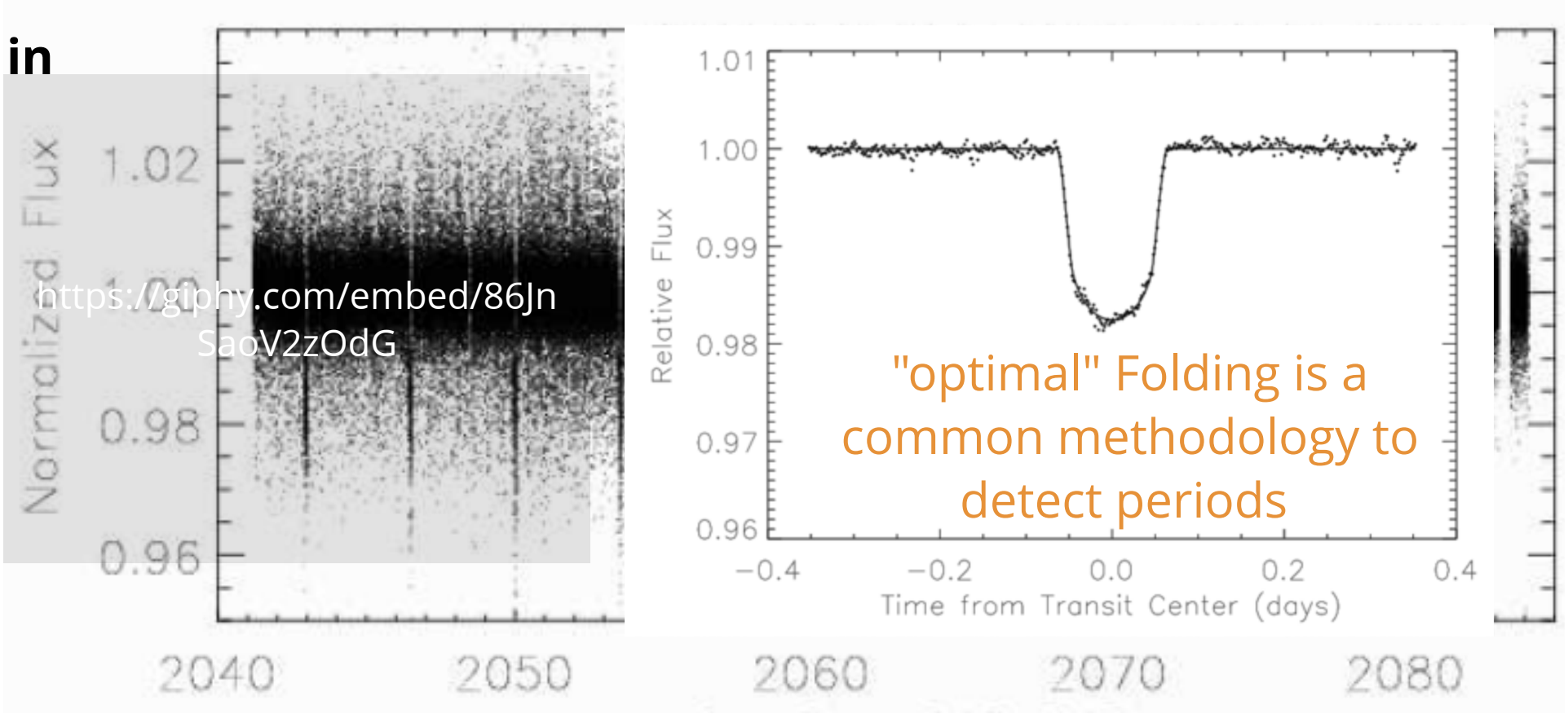


periodicity (repetitive patterns)

Time Series

HD 209458, the first transiting planet to be discovered.

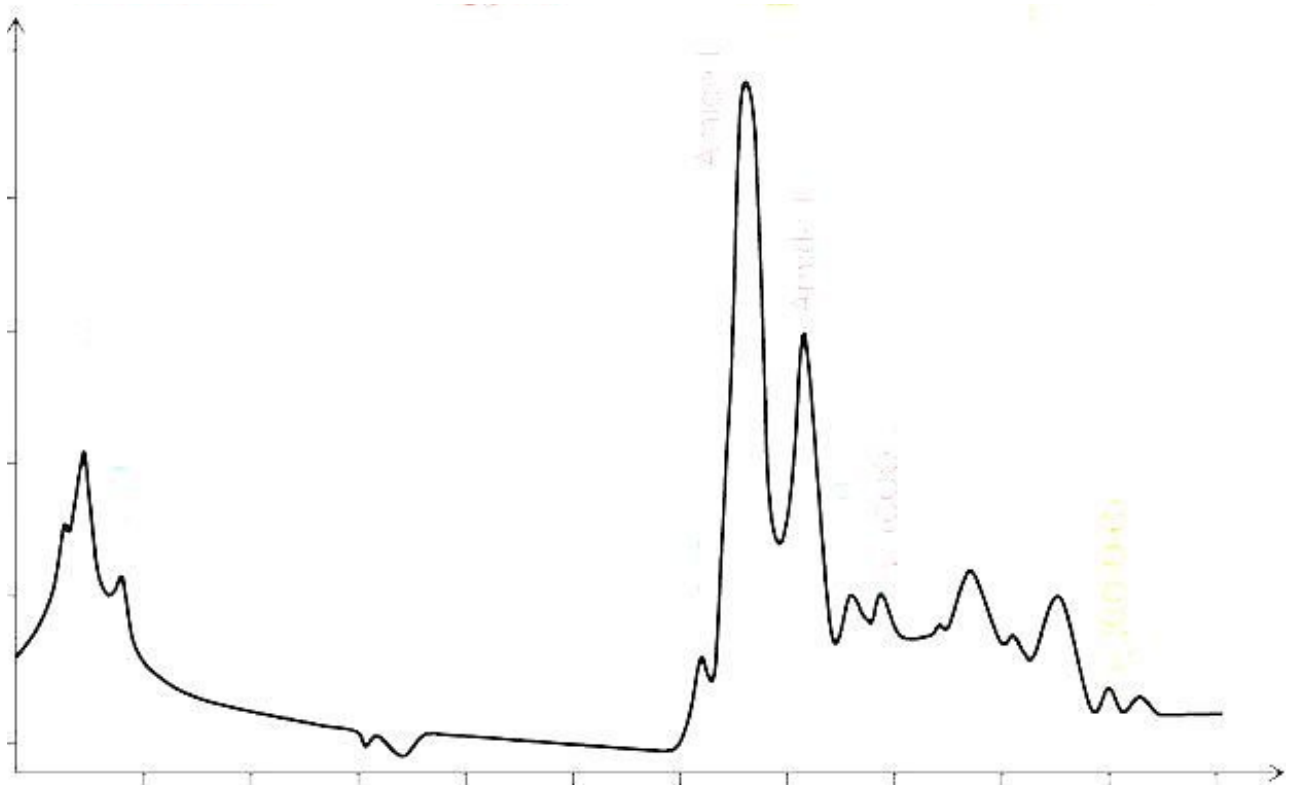
what is interesting in this time series?



periodicity (repetitive patterns)

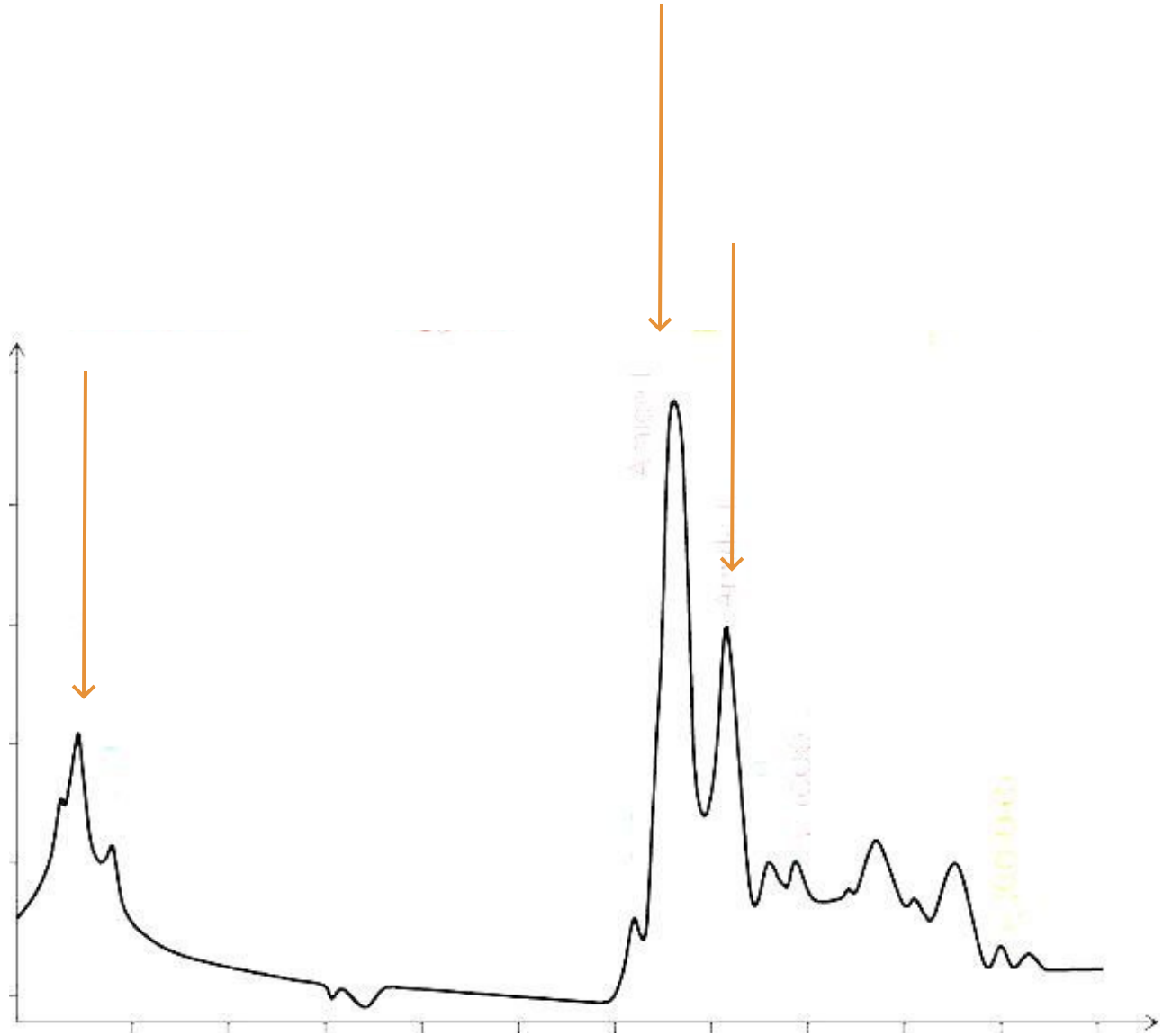
Time Series

what is interesting in this time series?



Time Series

what is interesting in this time series?



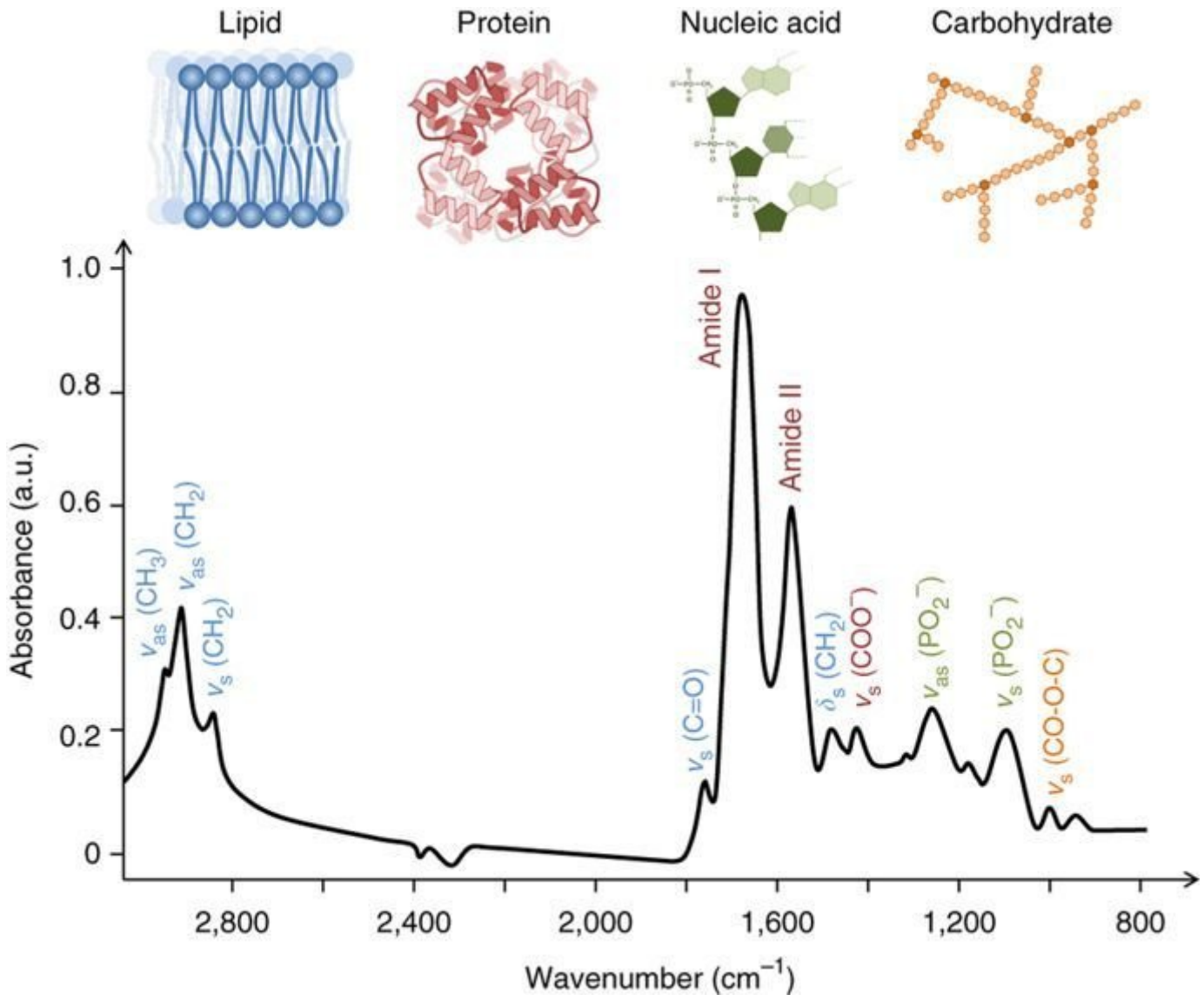
events

Time Series

what is interesting in this time series?

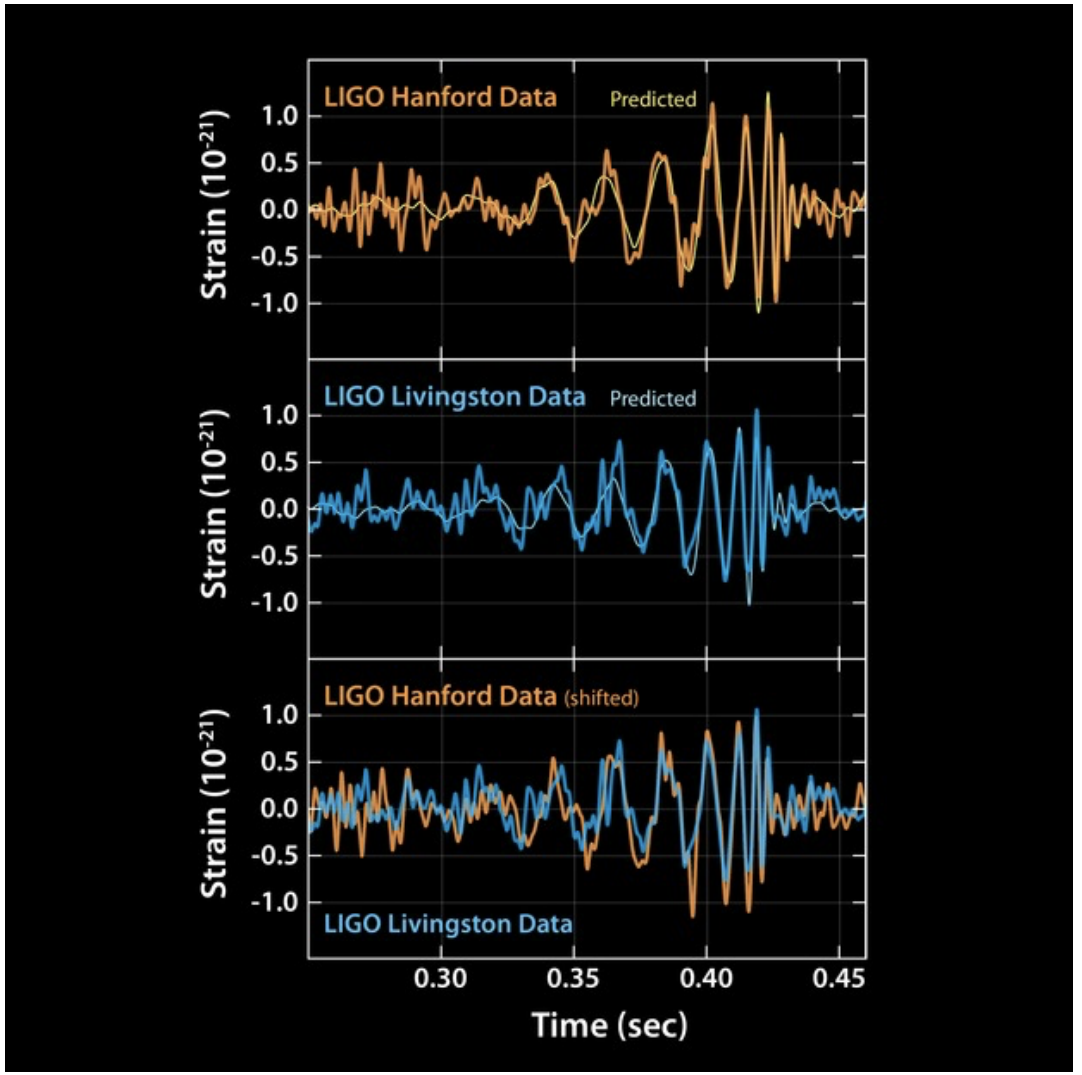
this is not a "time" series, but a spectral series, *but it is still a 1-D dataset with directional exogenous variable* so the same methodologies can be applied

events



Time Series

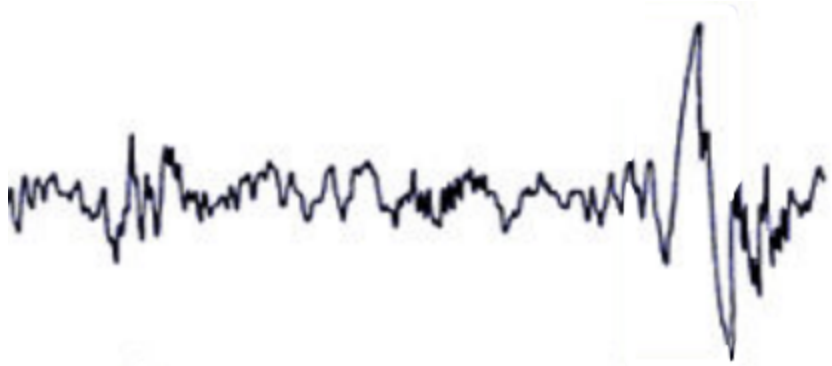
what is interesting in this time series?



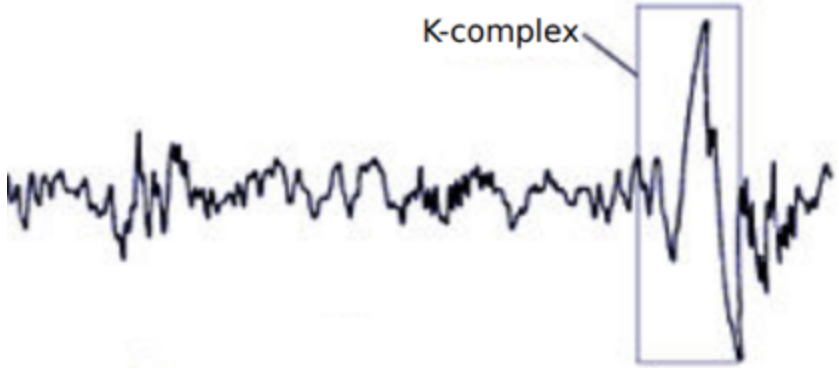
LIGO gravitational wave detection
Abbott et al. Physical Review Letters 116, 061102 (2016)

event detection/template matching

what is interesting here?

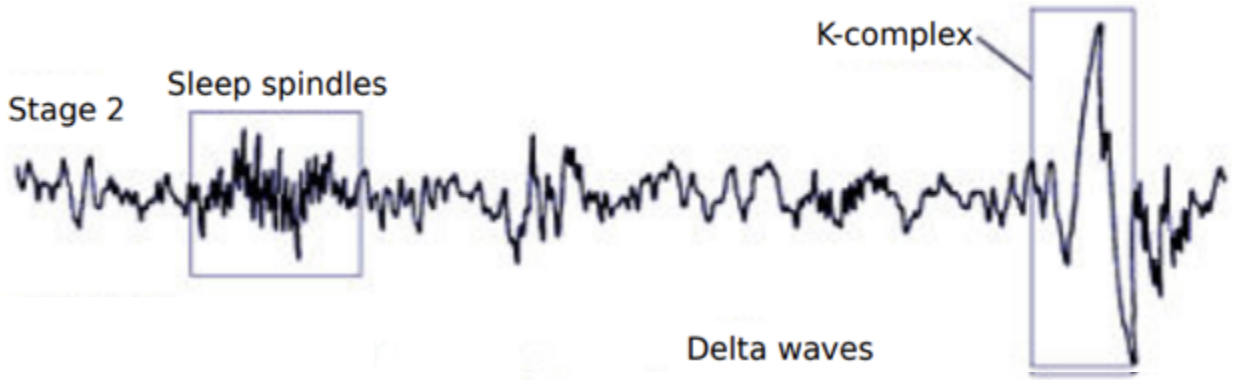


what is interesting here?



event detection

what is interesting here?



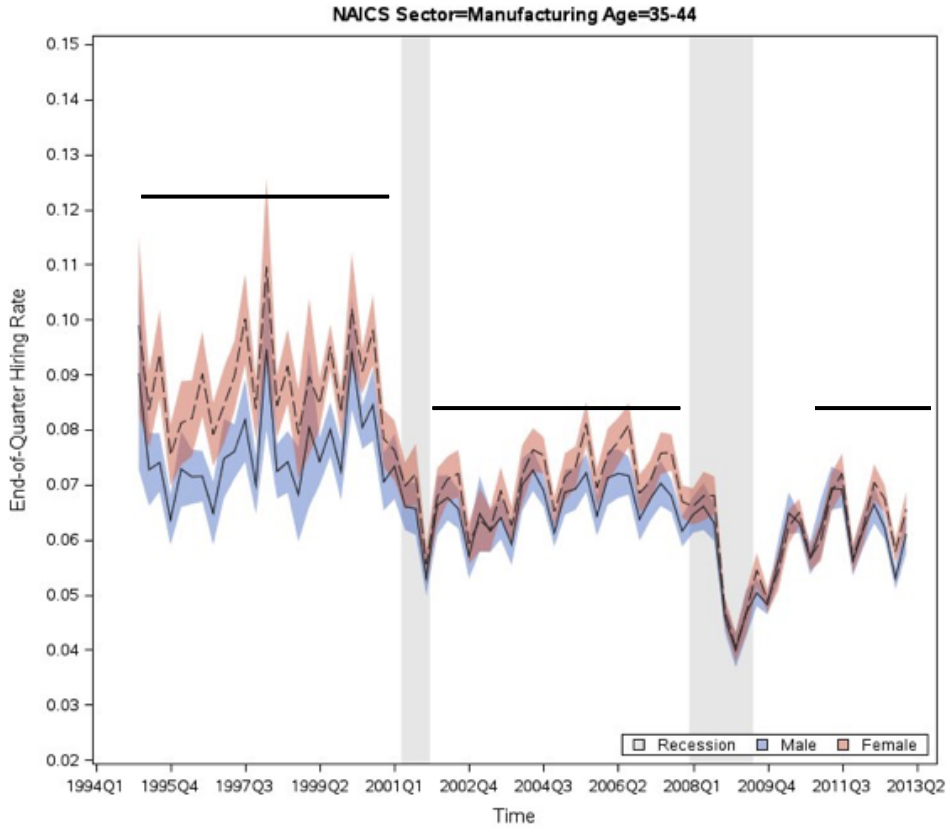
https://www.neurologic.theclinics.com/article/S0733-8619(05)00041-1/fulltext

The 3 behavioral states of wakefulness, rapid eye movement (REM) sleep, and non-REM (NREM)

sleep are characterized by specific changes in electroencephalography,

event detection

what is interesting here?



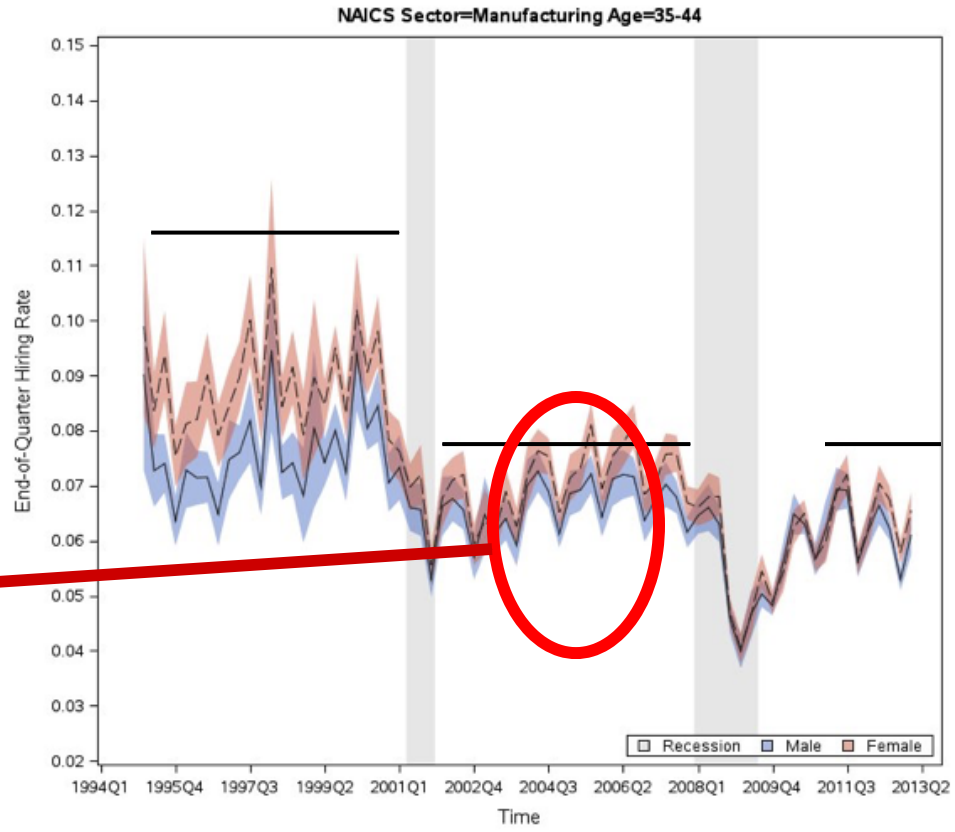
Longitudinal Employer-Household Dynamics

https://lehd.ces.census.gov/data/

Point of change detection

what is interesting here?

seasonal variations,
cyclic variation,
periodicity



Longitudinal Employer-Household Dynamics

https://lehd.ces.census.gov/data/

TSA topics

1. Trend detection
2. Periodicity/seasonality detection
3. Event detection / Anomaly detection
4. Point of change detection
5. Forecasting/prediction
6. Classification

forecasting

space state model

unobserved state

$$Y_t = x_t\beta + \epsilon_t; \quad \epsilon_t \sim N(0, \sigma_\epsilon^2)$$

$$x_{t+1} = x_t + \nu_t; \quad \nu_t \sim N(0, \sigma_\nu^2)$$

Underlying state x is a time varying Markovian process (the position of the pace craft)

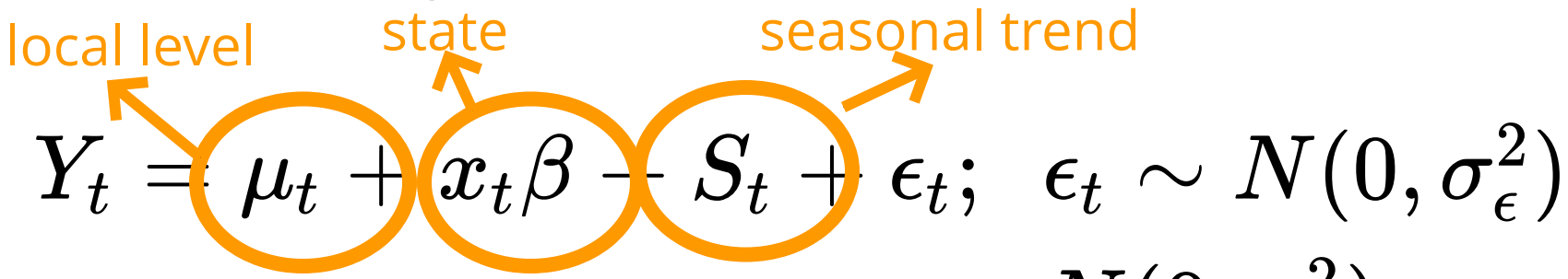
The observed variable depends at least on the state and on noise.

Other elements (e.g. seasonality) can be included in the model too

forecasting

space state model

we can write a Bayesian structural model like this:



$$Y_t = \mu_t + x_t\beta + S_t + \epsilon_t; \quad \epsilon_t \sim N(0, \sigma_\epsilon^2)$$

$$\mu_t = \mu_{t-1} + \nu_t; \quad \nu \sim N(0, \sigma_\nu^2)$$

forecasting

space state model

we can write a Bayesian structural model like this:

unobserved trend

seasonal variations

$$Y_t = \mu_t + x_t\beta + S_t + \epsilon_t; \quad \epsilon_t \sim N(0, \sigma_\epsilon^2)$$

$$\mu_{t+1} = \mu_t + \nu_t; \quad \nu_t \sim N(0, \sigma_\nu^2)$$

Its a *Markovian Process*:
stochastic process with
1-step memory

there is a hidden or latent process x_t called the state process (the position of the space craft)

Learning Long-Term Dependencies with Gradient Descent is Difficult

Yoshua Bengio†, Patrice Simard†, and Paolo Frasconi‡

†AT&T Bell Laboratories

‡Dip. di Sistemi e Informatica, Universitá di Firenze

1994

http://www.comp.hkbu.edu.hk/~markus/teaching/comp7650/tnn-94-gradient.pdf

An time-domain enabled AI system should:

1. the system be able to store information for an arbitrary duration,

2. the system be resistant to noise (i.e., fluctuations of the inputs that are random or irrelevant to predicting a correct output).

3. the system parameters be trainable (in reasonable time).

Learning Long-Term Dependencies with Gradient Descent is Difficult

Yoshua Bengio†, Patrice Simard†, and Paolo Frasconi‡

†AT&T Bell Laboratories

‡Dip. di Sistemi e Informatica, Universitá di Firenze

1994

t

1. the system be able to store information for an arbitrary duration,

you need to pick

2. the system be resistant to noise (i.e., fluctuations of the inputs that are random or irrelevant to predicting a correct output).

3. the system parameters be trainable (in reasonable time).

Learning Long-Term Dependencies with Gradient Descent is Difficult

Yoshua Bengio†, Patrice Simard†, and Paolo Frasconi‡

†AT&T Bell Laboratories

‡Dip. di Sistemi e Informatica, Universitá di Firenze

1994

http://www.comp.hkbu.edu.hk/~markus/t
aching/comp7650/tnn-94-gradient.pdf

1. the system be able to store information for an arbitrary duration,

you need to pick

2. the system be resistant to noise (i.e., fluctuations of the inputs that are random or irrelevant to predicting a correct output).

3. the system parameters be trainable (in reasonable time).

We show why gradient based learning algorithms face an increasingly dicult problem as the duration of the dependencies to be captured increases

Learning Long-Term Dependencies with Gradient Descent is Difficult

Yoshua Bengio†, Patrice Simard†, and Paolo Frasconi‡

†AT&T Bell Laboratories

‡Dip. di Sistemi e Informatica, Universitá di Firenze

1994

http://www.comp.hkbu.edu.hk/~markus/teaching/comp7650/tnn-94-gradient.pdf

1. the system be able to store information for an arbitrary duration,

2. the system be resistant to noise (i.e., fluctuations of the inputs that are random or irrelevant to predicting a correct output).

3. the system parameters be trainable (in reasonable time).

you need to pick

We show why gradient based learning algorithms face an increasingly dicult problem as the duration of the dependencies to be captured increases

the magnitude of the derivative of the state of a dynamical system at time t with respect to the state at time 0 decreases exponentially as t increases.

3

RNN

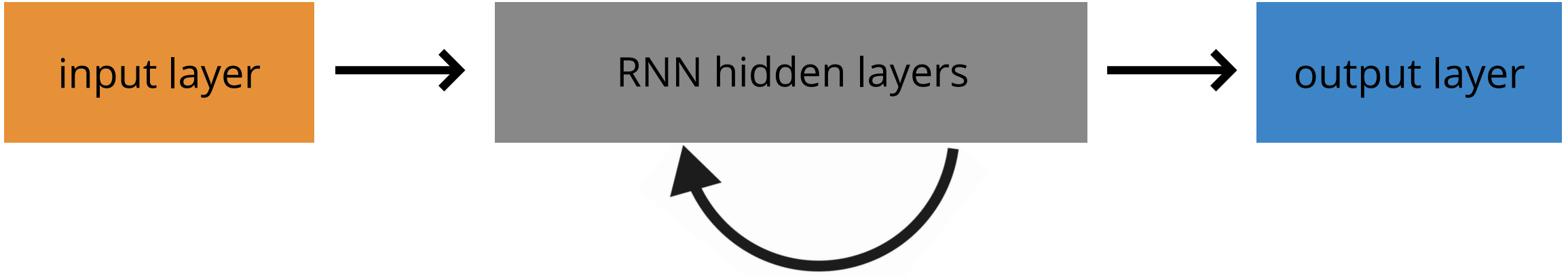
RNN architecture

Feed-forward NN architecture

RNN architecture

Feed-forward NN architecture

Recurrent NN architecture



RNN architecture

In TSA this is a State Space Probem

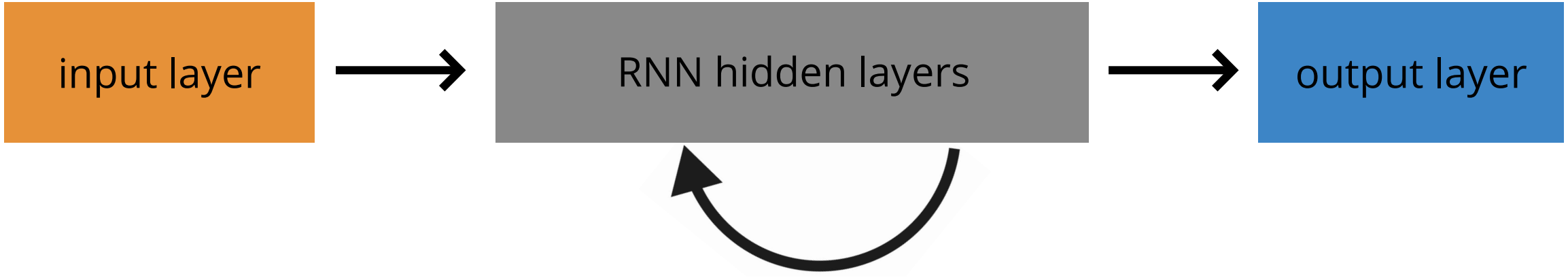
we want process a sequence of vectors *x* applying a recurrence formula at every time step:

$$h_t = f_q(h_{t-1}, x_t)$$

current state

previous state

current input



RNN architecture

In TSA this is a State Space Probem

we want process a sequence of vectors *x* applying a recurrence formula at every time step:

function with parameters *q*

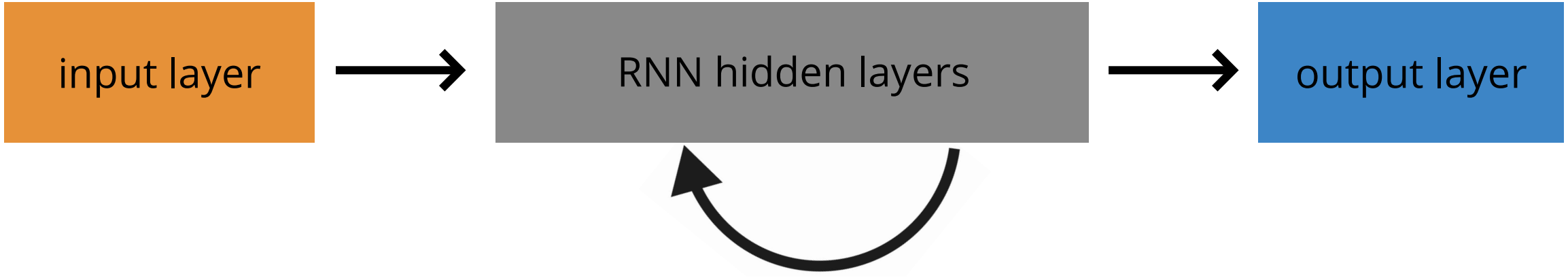
$$h_t = f_q(h_{t-1}, x_t)$$

current state

previous state

features

(can be time dependent



state space model

$$y_t = Hx_t + \epsilon_t; \quad \epsilon_t \sim N(0, \Sigma_\epsilon^2)$$

$$x_t = \Phi x_{t-1} + \nu_t; \quad \nu_t \sim N(0, \Sigma_\nu^2)$$

A State-space model is a model to derive the value of a time-dependent variable *x(t)*, the state, generated by a noisy Markovian process, from observations of a variable *y(t),* also subject to noise, linearly related to the target variable

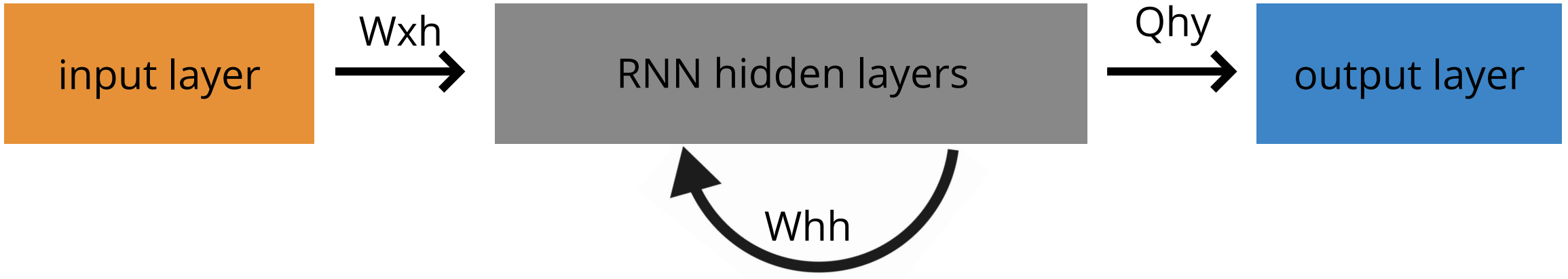
Definition

RNN architecture

Simplest possible RNN

$$h_t = f_q(h_{t-1}, x_t)$$

$$y_t = Q_{hy} \cdot h_t$$

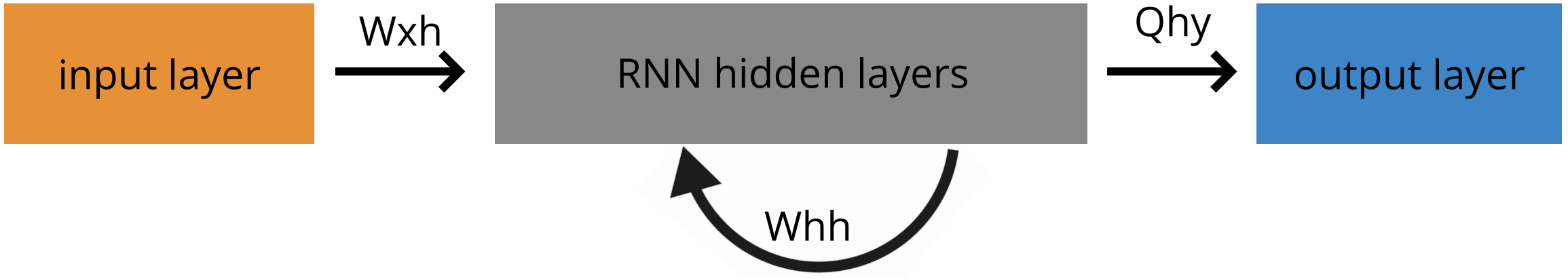


RNN architecture

Simplest possible RNN

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

$$y_t = Q_{hy} \cdot h_t$$



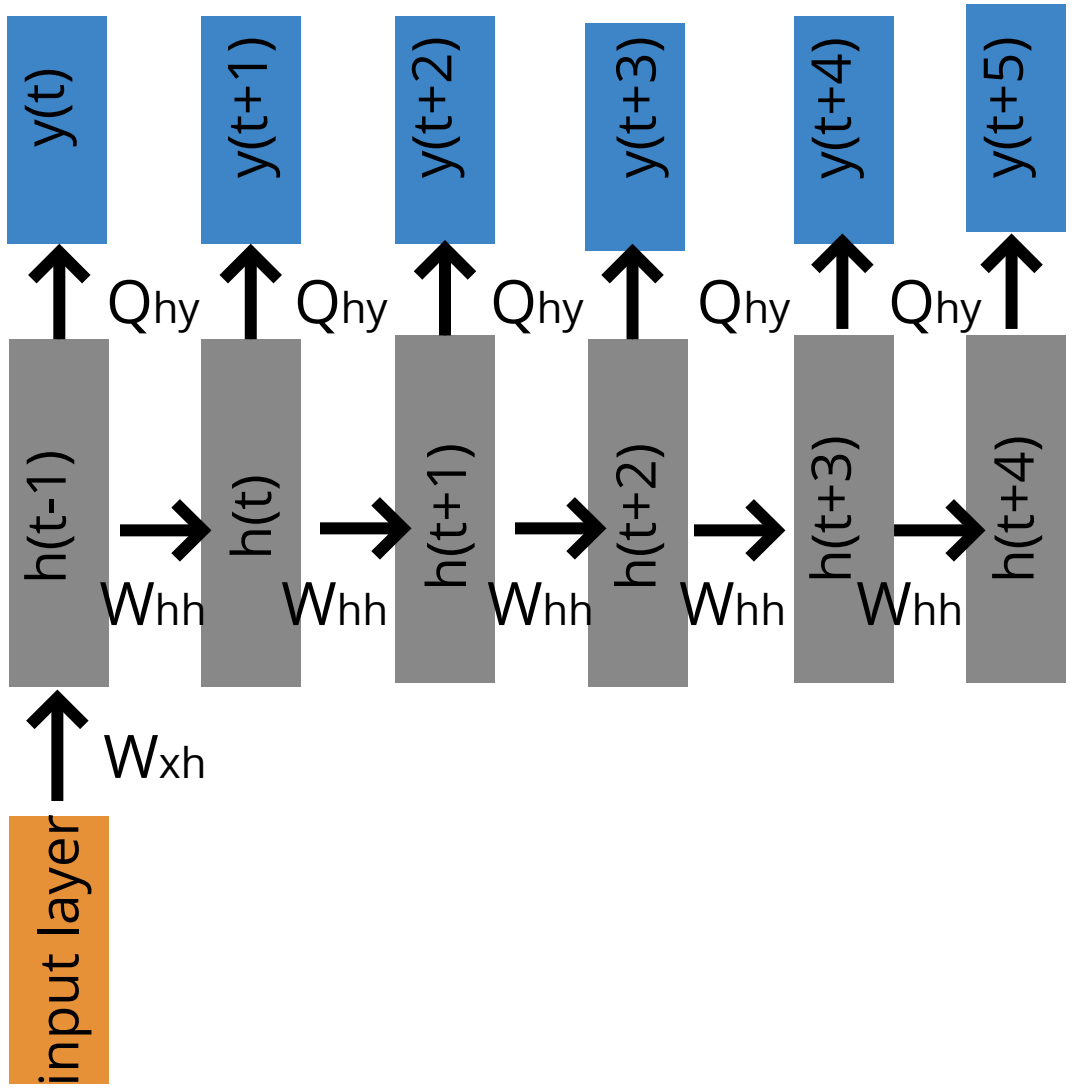
RNN architecture

Alternative graphical representation of RNN

$$h_t = f_q(h_{t-1}, x_t)$$

$$y_t = Q_{hy} \cdot h_t$$

the weights are the same! always the same Whh and Qhy



RNN architecture

applications

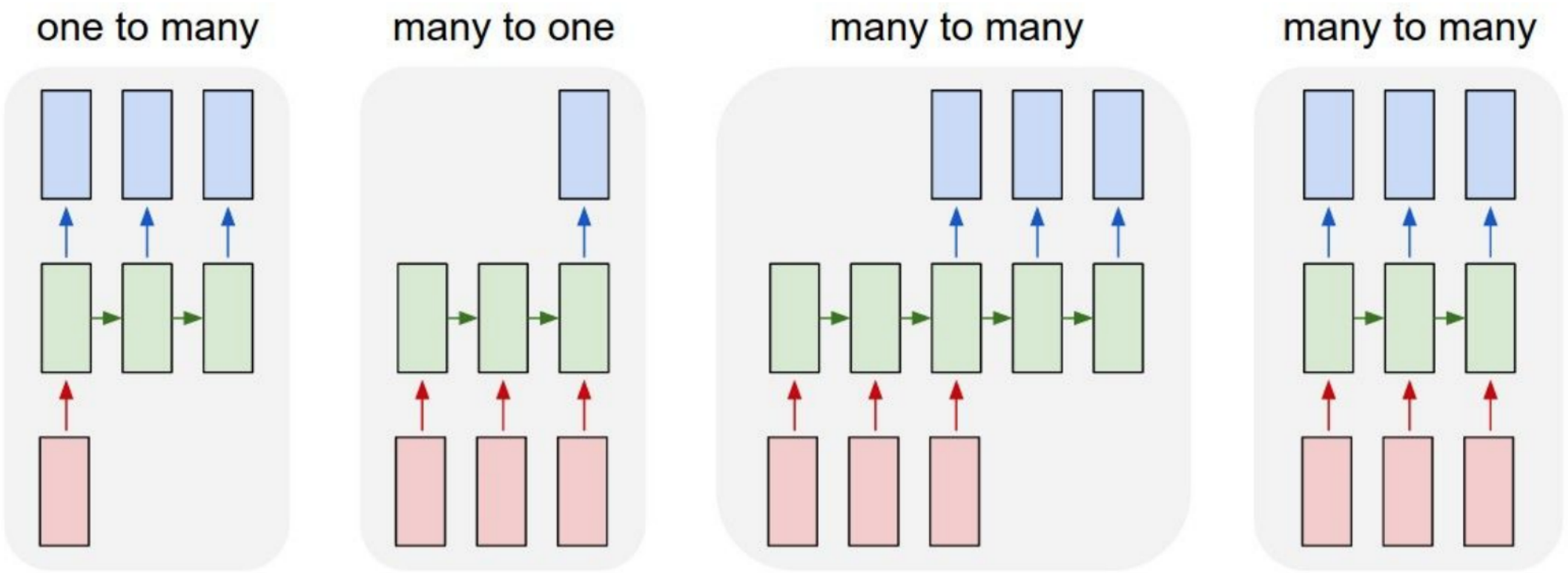


image captioning:
one image to a
sequence of words

RNN architecture

applications

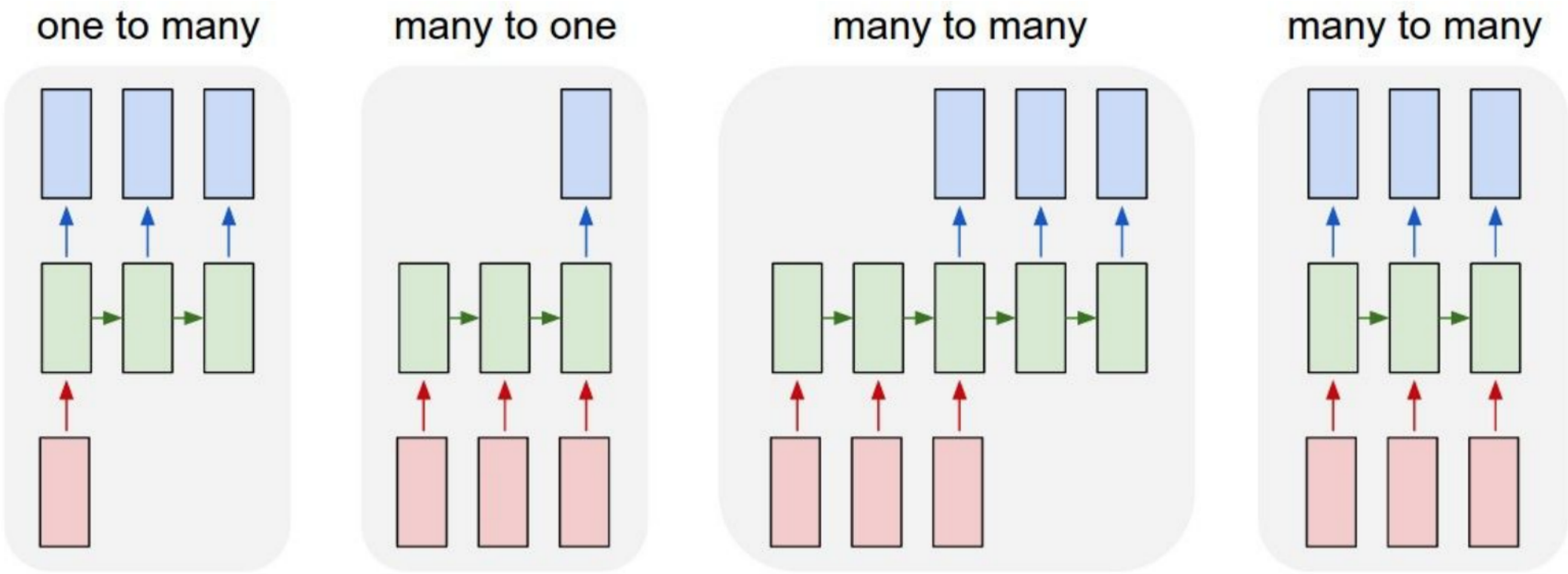


image captioning:
one image to a
sequence of words

sentiment analysis
sequence of words
to one sentiment

RNN architecture

applications

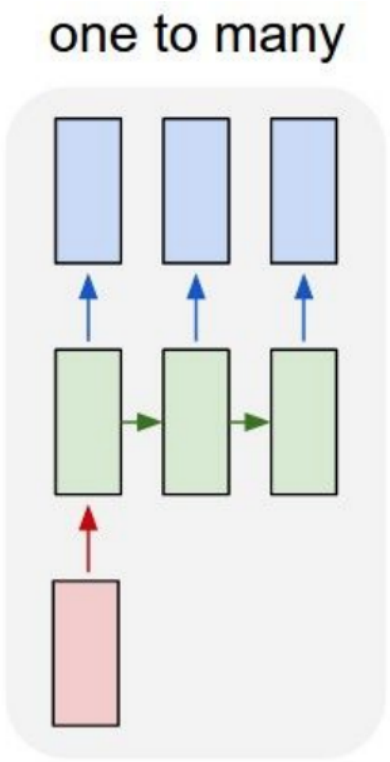
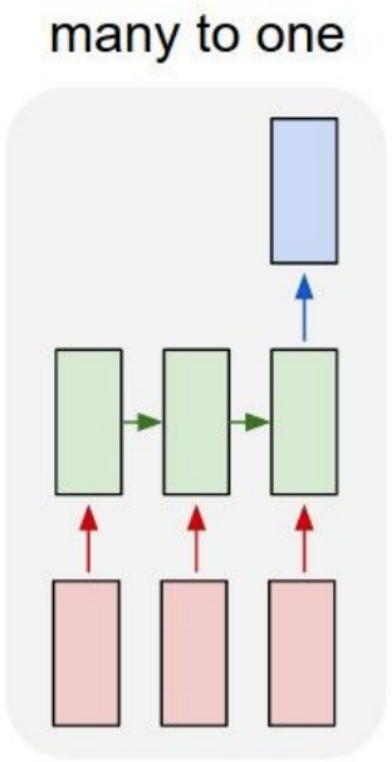
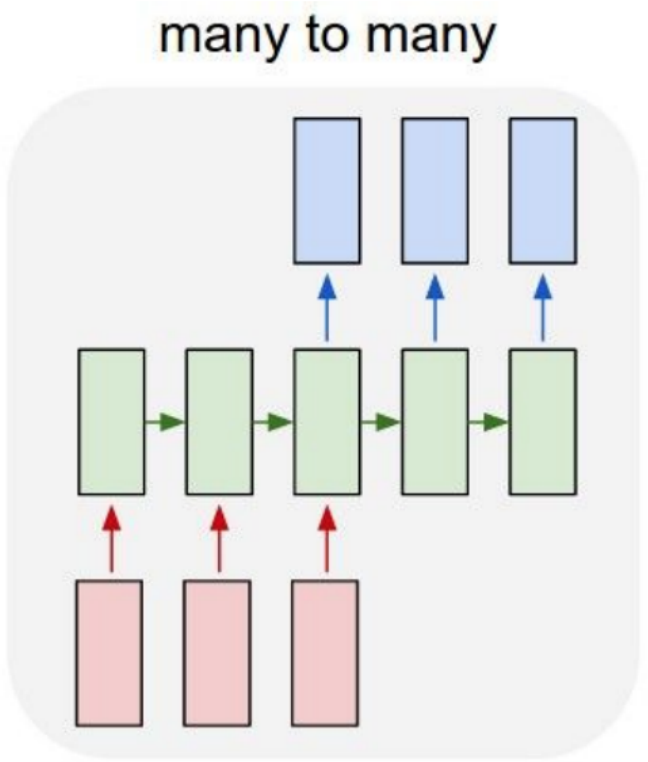


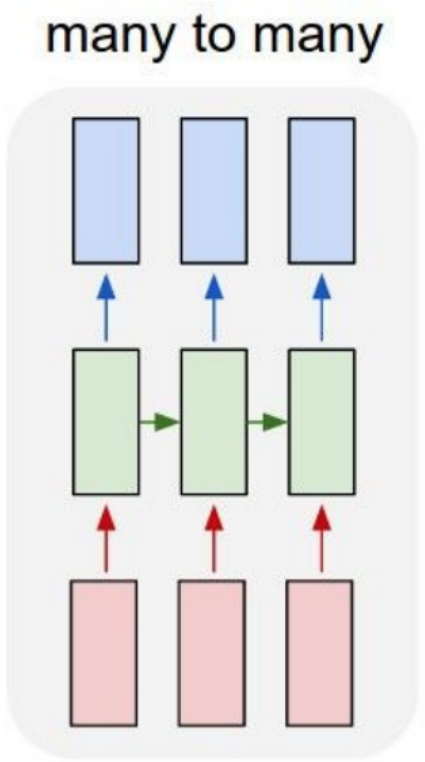
image captioning:
one image to a
sequence of words



sentiment analysis
sequence of words
to one sentiment



language translator
sequence of words to
sequence of words



RNN architecture

applications

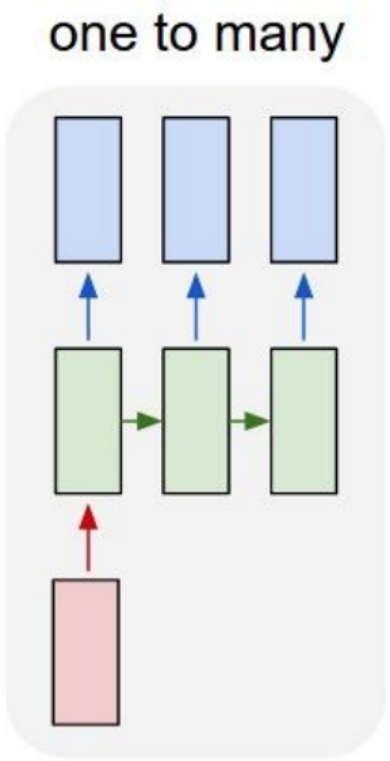
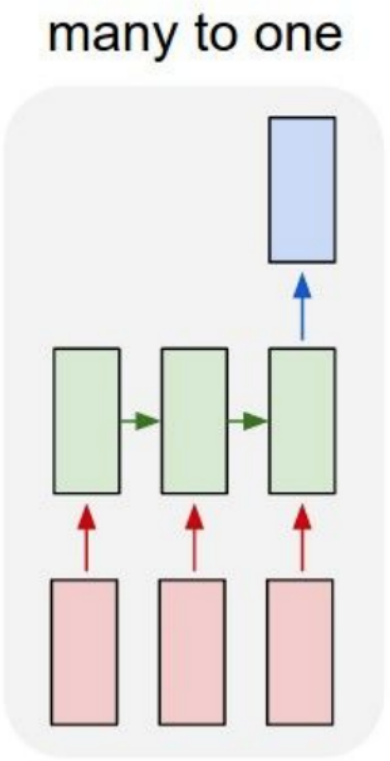
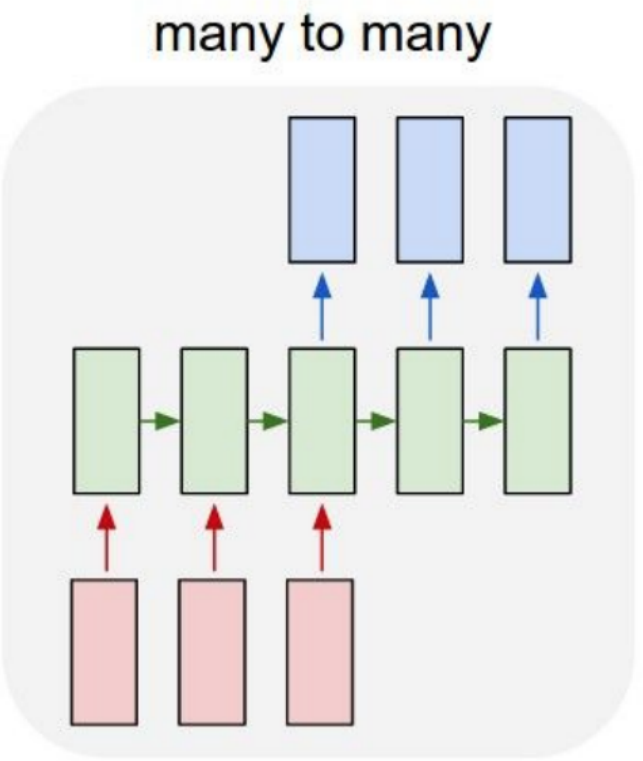


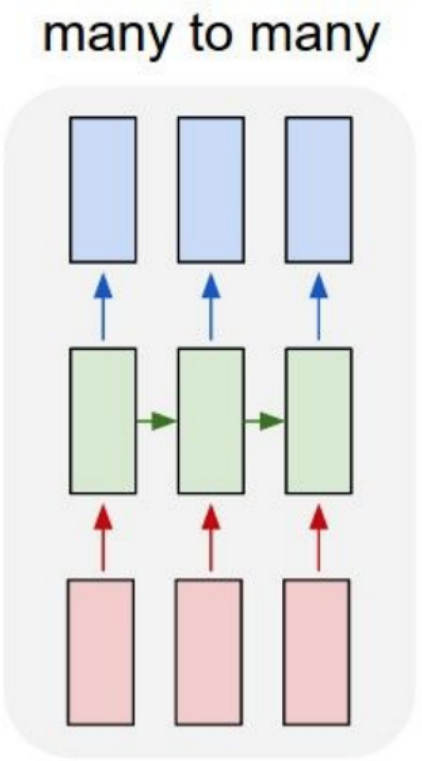
image captioning:
one image to a sequence of words



sentiment analysis
sequence of words to one sentiment



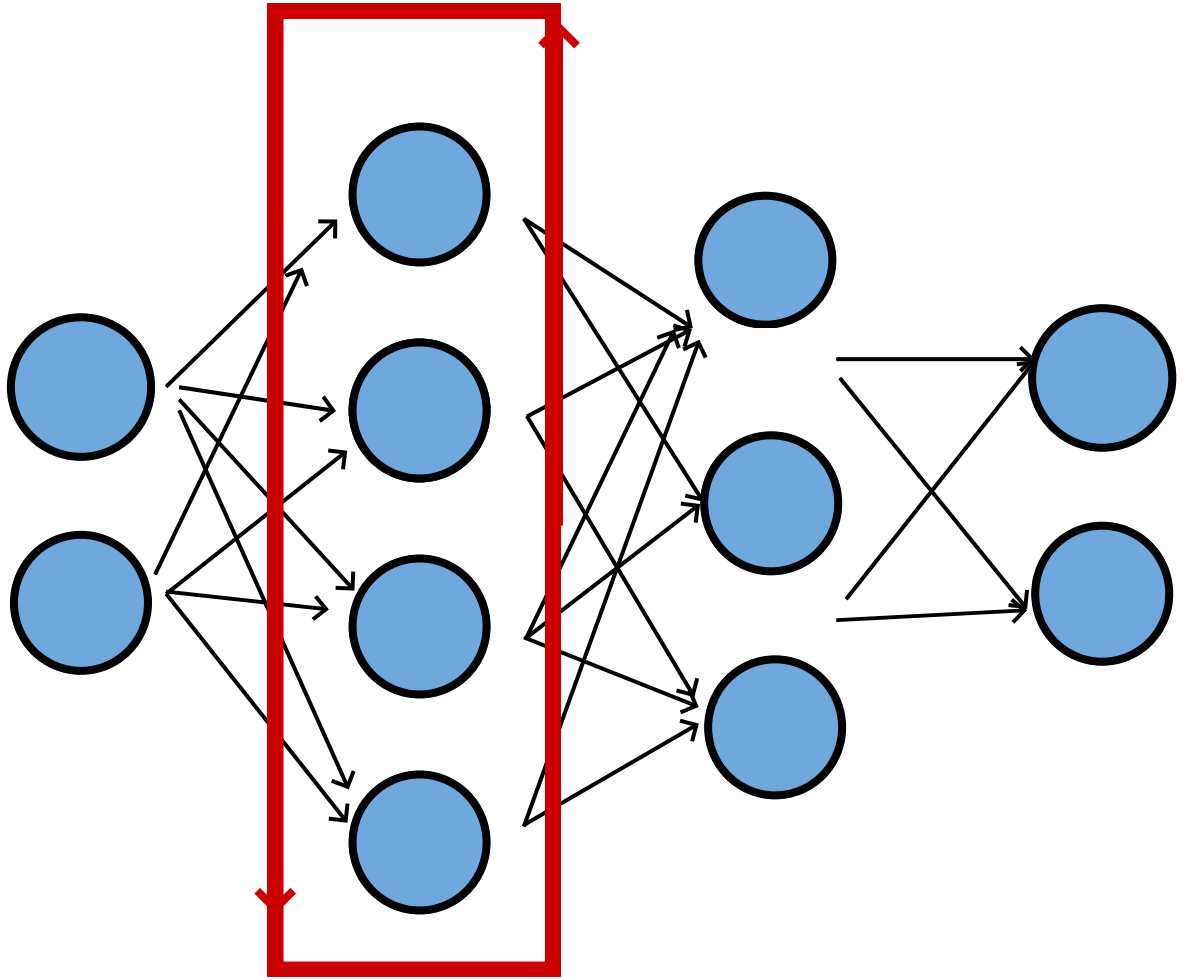
language translator
sequence of words to sequence of words



online: video classification frame by frame

RNN architecture

more complicated RNNs



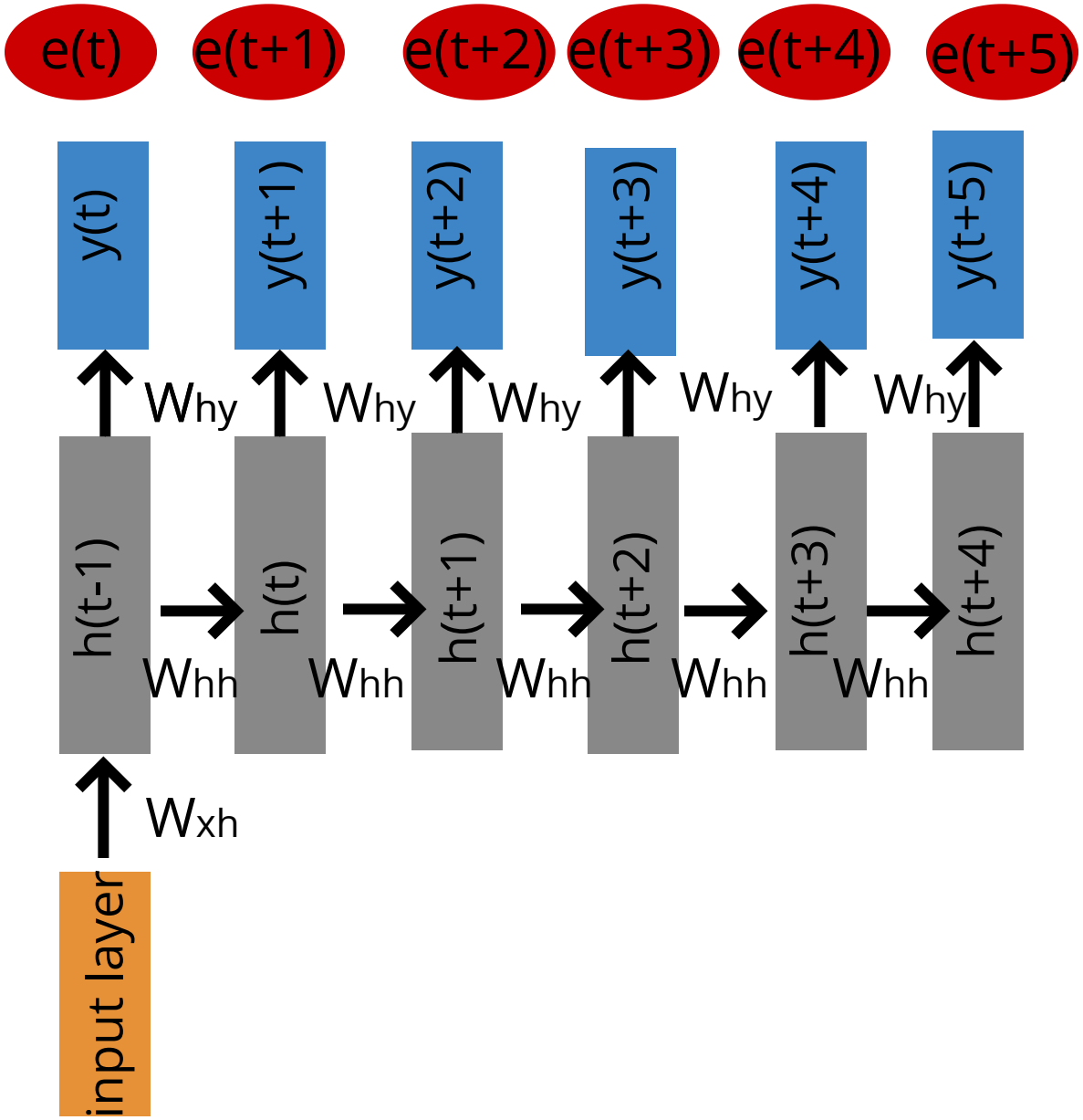
Some layers will be recurrent, others will not. Does not need to be fully connected

RNN architecture

$$h_t = W_h \phi(h_{t-1}) + W_x x(t)$$

$$y_t = W_y \phi(h_t)$$

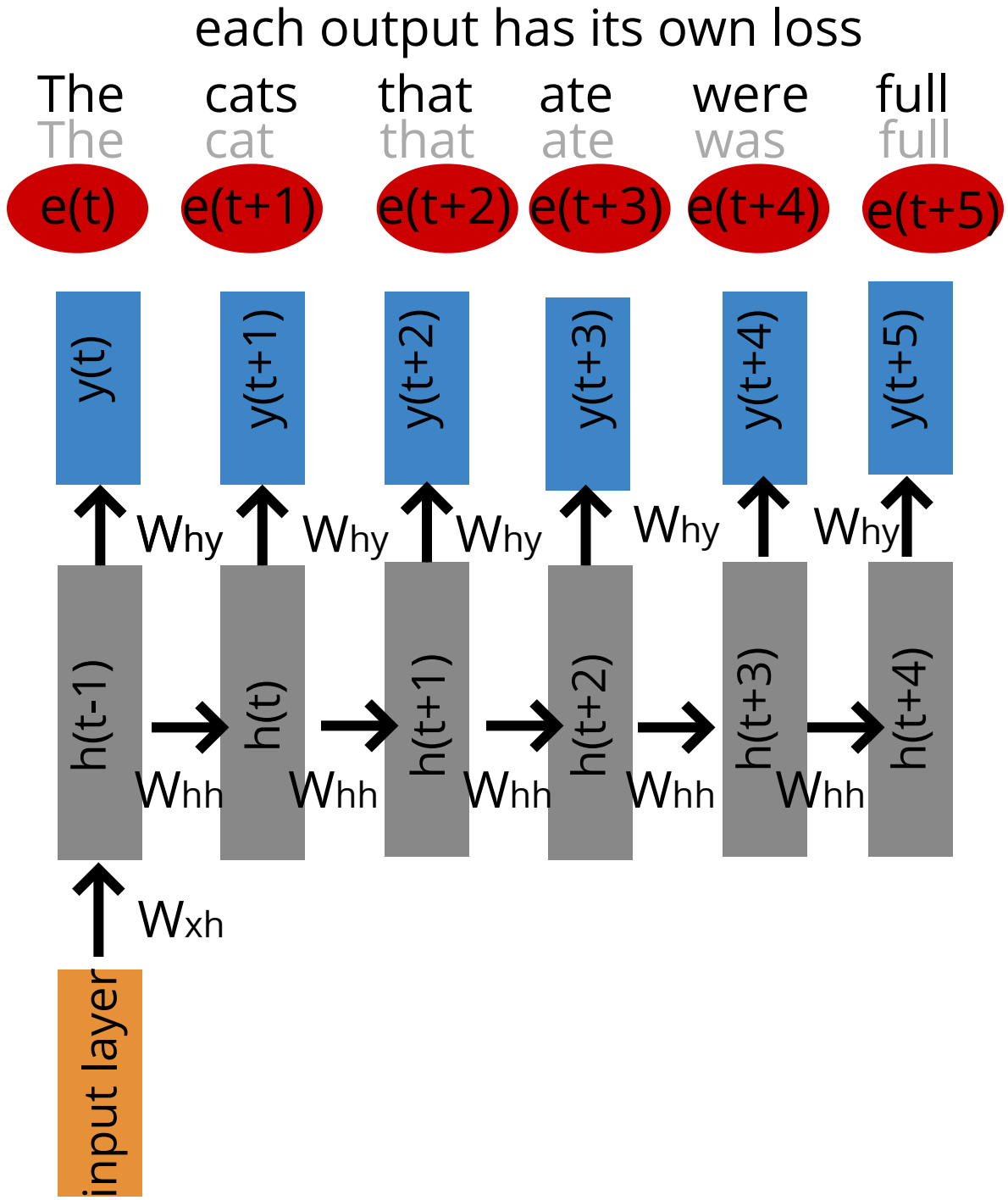
each output has its own loss



RNN architecture

$$h_t = W_h \phi(h_{t-1}) + W_x x(t)$$

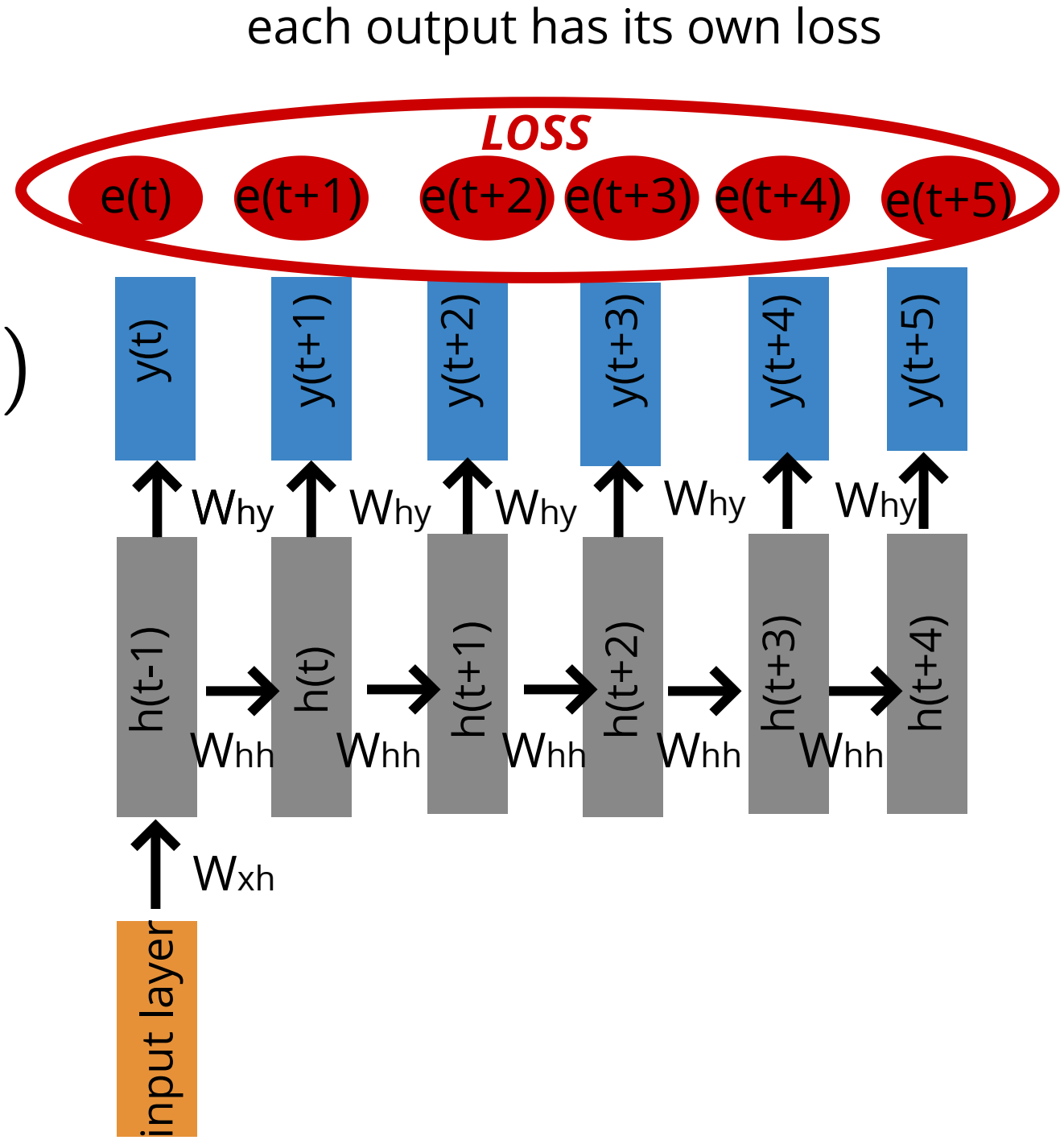
$$y_t = W_y \phi(h_t)$$



RNN architecture

$$h_t = W_h \phi(h_{t-1}) + W_x x(t)$$

$$y_t = W_y \phi(h_t)$$



RNN architecture

each output has its own loss

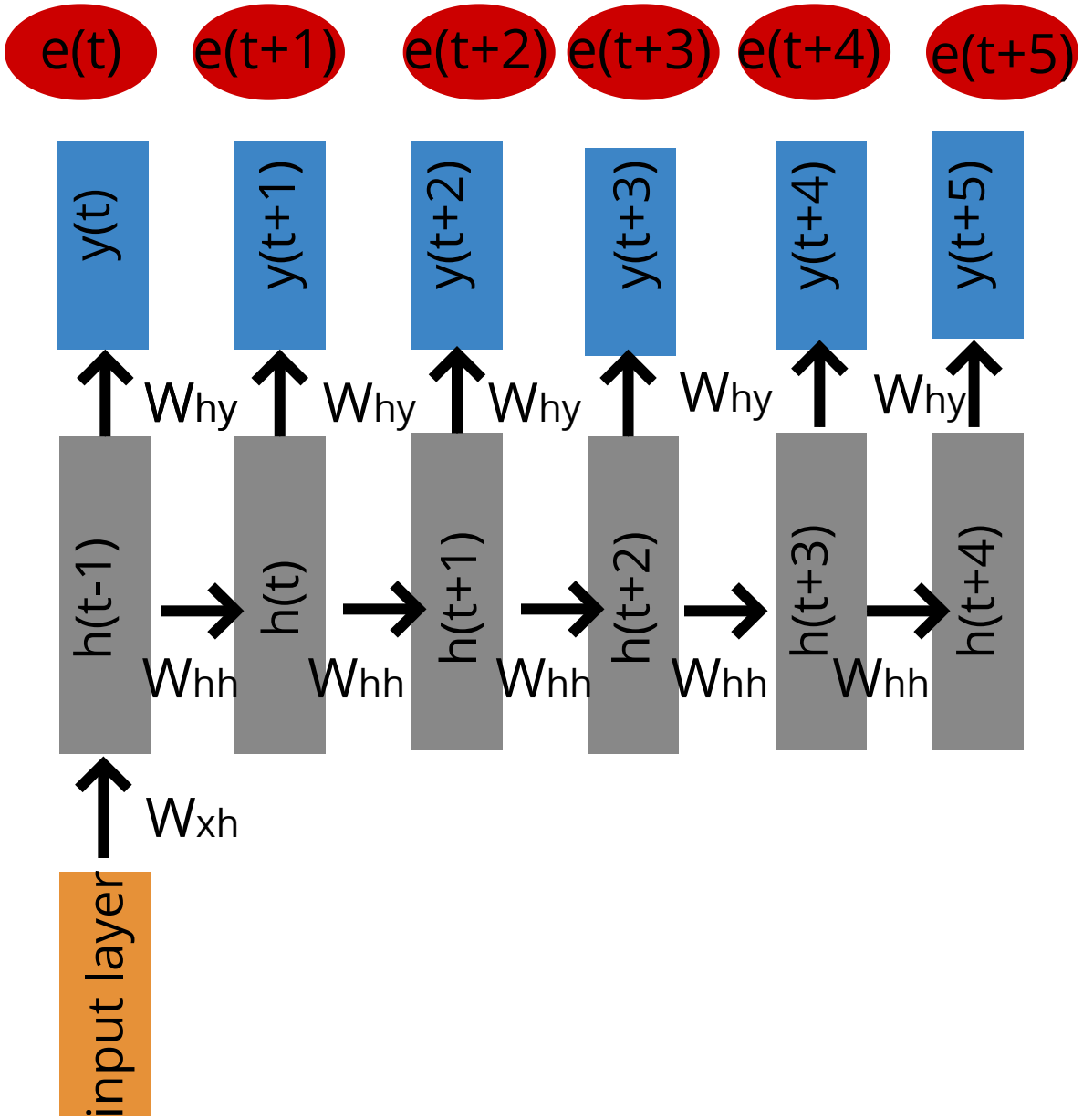
$$h_t = W_h \phi(h_{t-1}) + W_x x(t)$$

$$y_t = W_y \phi(h_t)$$

Total loss:

$$\frac{\partial E}{\partial \theta} = \sum_{t=1}^{N} \frac{\partial e_t}{\partial \theta}$$

$$\frac{\partial e_t}{\partial \theta} = \sum_{k=1}^{t} \frac{\partial e_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_k}{\partial W} \frac{\partial h_t}{\partial h_k}$$



RNN architecture

each output has its own loss

$$h_t = W_h \phi(h_{t-1}) + W_x x(t)$$

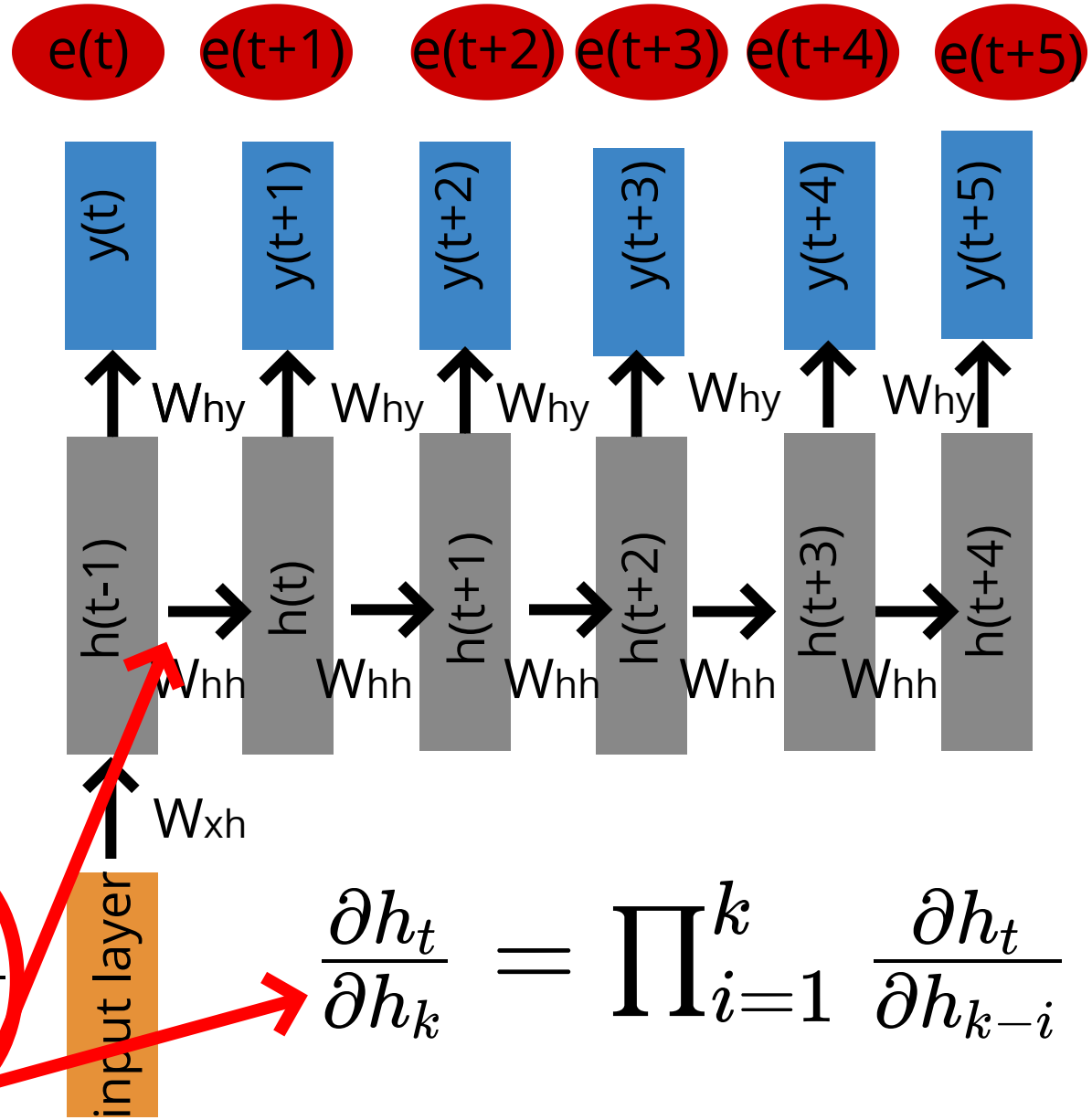
$$y_t = W_y \phi(h_t)$$

Total loss:

$$\frac{\partial E}{\partial \theta} = \sum_{t=1}^{N} \frac{\partial e_t}{\partial \theta}$$

$$\frac{\partial e_t}{\partial \theta} = \sum_{k=1}^{t} \frac{\partial e_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_k}{\partial W} \frac{\partial h_t}{\partial h_k}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=1}^{k} \frac{\partial h_t}{\partial h_{k-i}}$$



RNN architecture

each output has its own loss

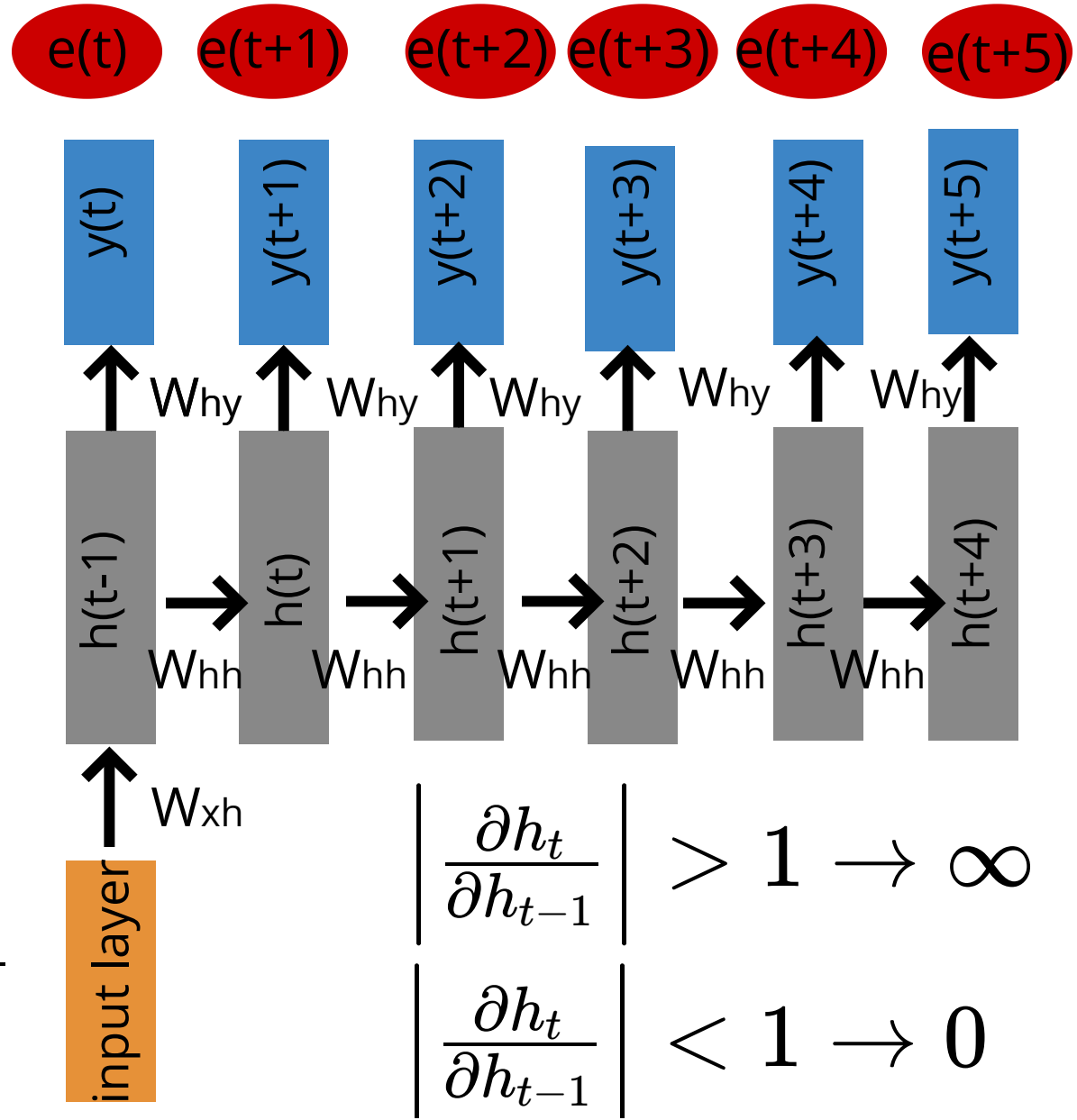
$$h_t = W_h \phi(h_{t-1}) + W_x x(t)$$

$$y_t = W_y \phi(h_t)$$

Total loss:

$$\frac{\partial E}{\partial \theta} = \sum_{t=1}^{N} \frac{\partial e_t}{\partial \theta}$$

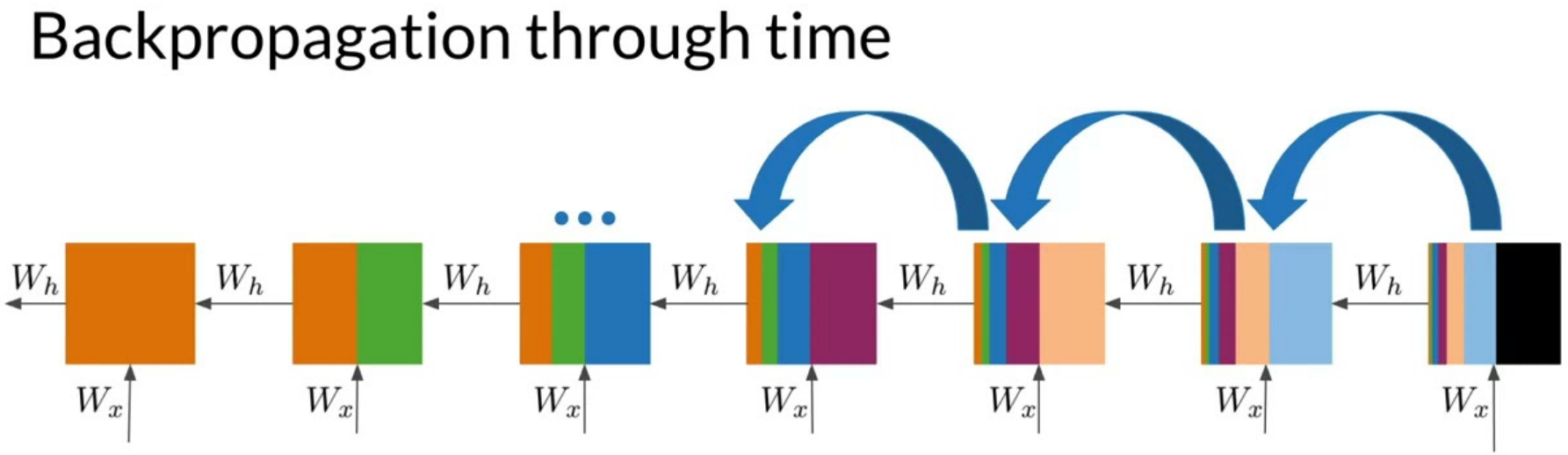
$$\frac{\partial e_t}{\partial \theta} = \sum_{k=1}^{t} \frac{\partial e_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_k}{\partial W} \frac{\partial h_t}{\partial h_k}$$



$$\left|\frac{\partial h_t}{\partial h_{t-1}}\right| > 1 \rightarrow \infty$$

$$\left|\frac{\partial h_t}{\partial h_{t-1}}\right| < 1 \rightarrow 0$$

RNN architecture



vanishing gradient problem!

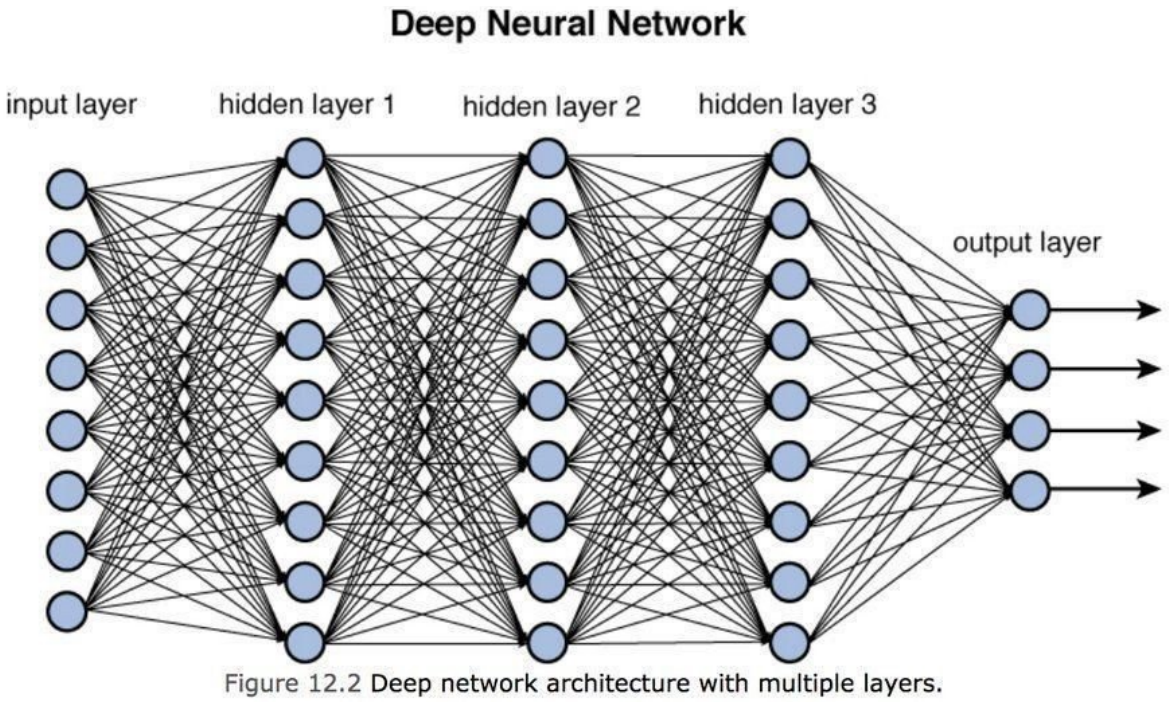
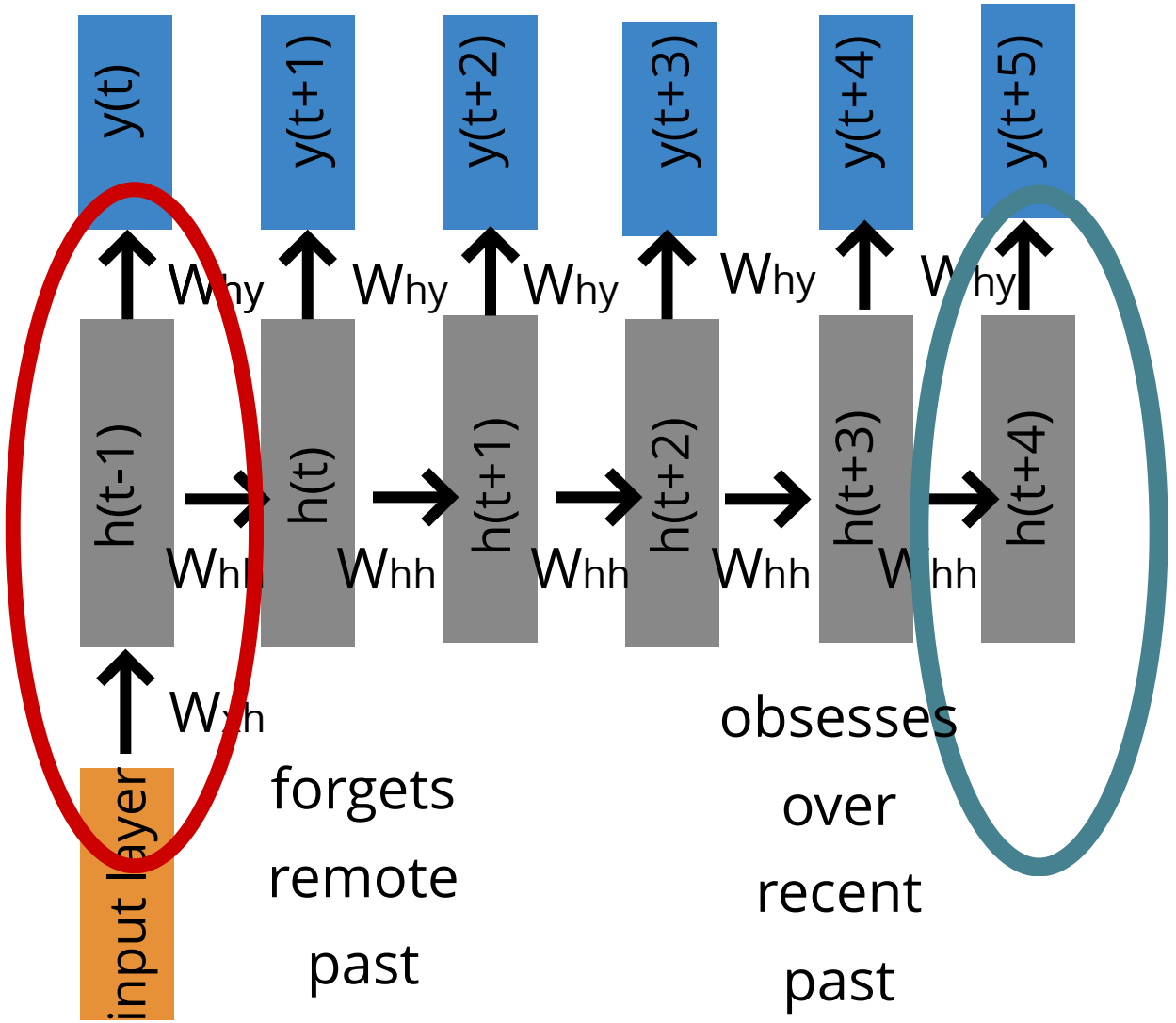


Figure 12.2 Deep network architecture with multiple layers.



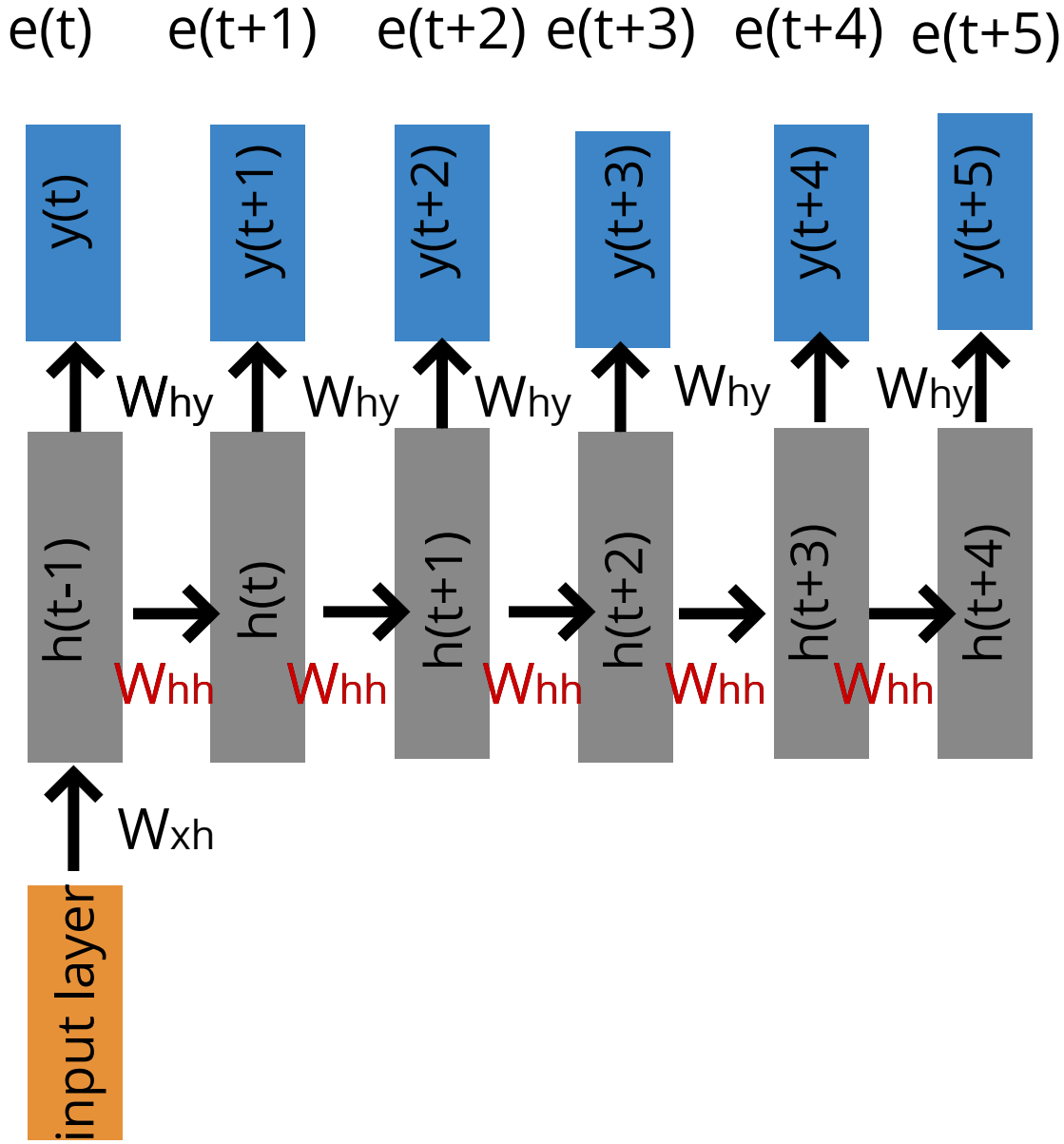
vanishing gradient problem!

vanishing gradient problem is exacerbated by having the same set of weights.

The vanishing gradient problem causes early layer to not to learn as effectively

The earlier layers learn from the remote past

As a result: vanilla RNN would only have short term memory (only learn from recent states)



4

LSTM

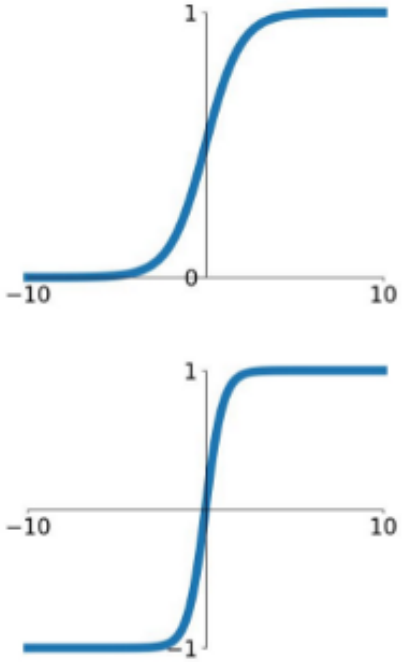
https://www.pluralsight.com/guides/introduction-to-lstm-units-in-rnn

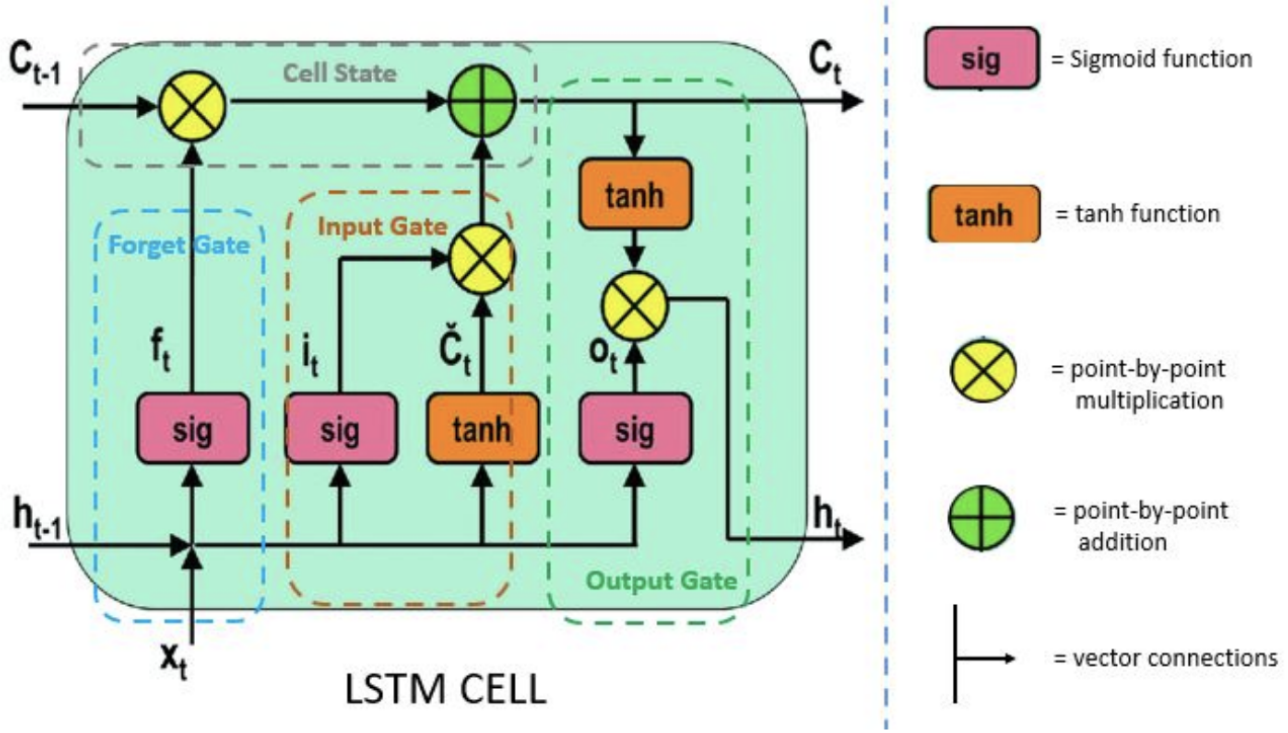
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

tanh

$$\tanh(x)$$





C_t: output

h: hidden states

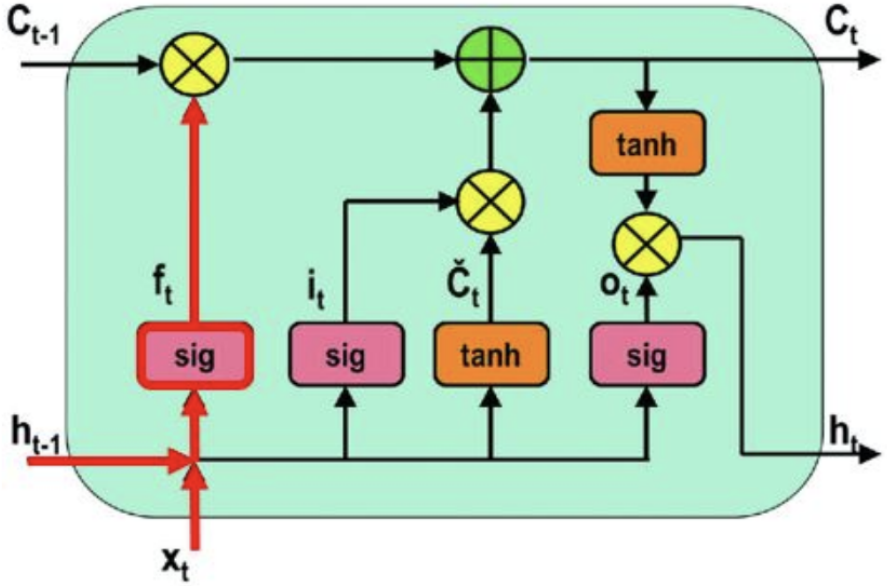
X: input

C_{t-1} : previous cell state (previous output)

h_{t-1} : previous hidden state

x_t : current state (input)

LSTM: long short term memory
solution to the vanishing gradient problem



Forget Gate Operation

forget gate:

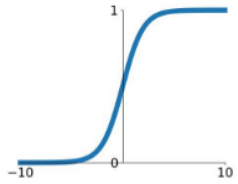
do i keep memory of this past step

$$f^{(t)} = \sigma(W^f [h_{t-1}, x_t] + b^f)$$

LSTM: long short term memory solution to the vanishing gradient problem

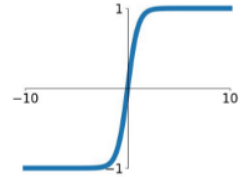
Sigmoid

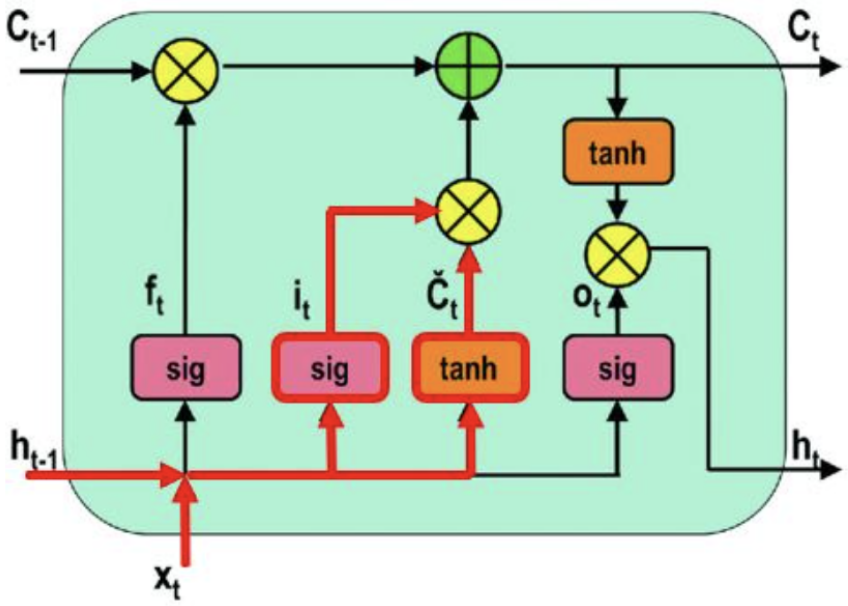
$\sigma(x) = \frac{1}{1+e^{-x}}$



tanh

$\tanh(x)$





Input Gate Operation

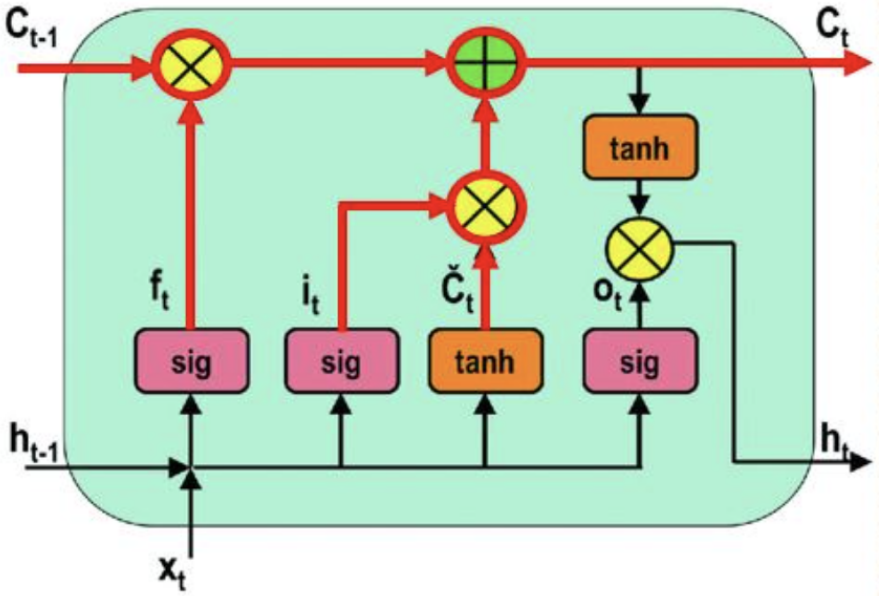
input gate:

do I update the current cell?

$$i^{(t)} = \sigma(W^i[h_{t-1}, x_t] + b^i)$$

$$\hat{C}^{(t)} = tanh(W^C[h_{t-1}, x_t] + b^C)$$

LSTM: long short term memory
solution to the vanishing gradient problem



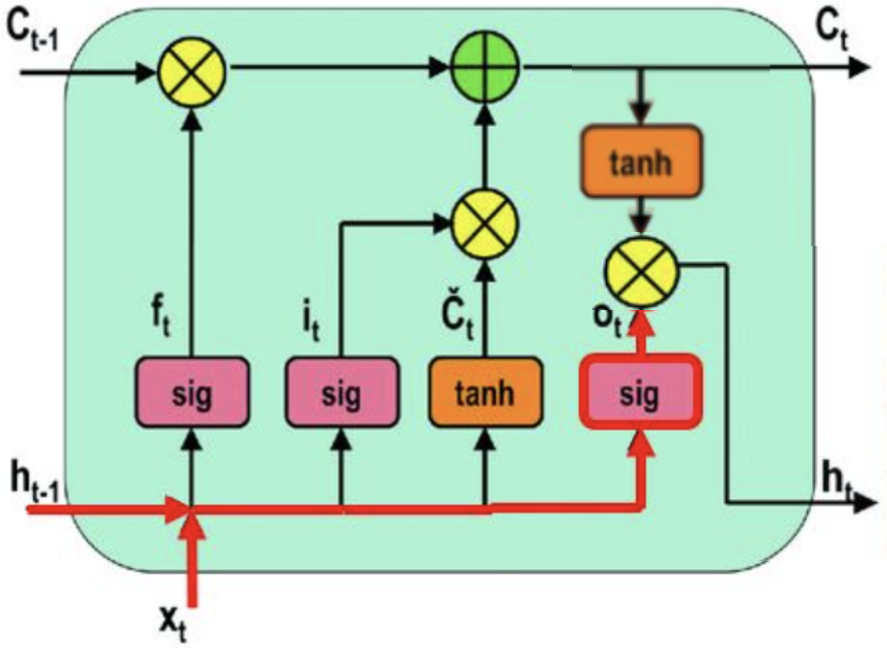
Cell State Operation

cell state:

procuces the prediction

$$C^{(t)} = C^{(t-1)} \times f^{(t)} + i^{(t)} \times \hat{C}^{(t)}$$

LSTM: long short term memory solution to the vanishing gradient problem



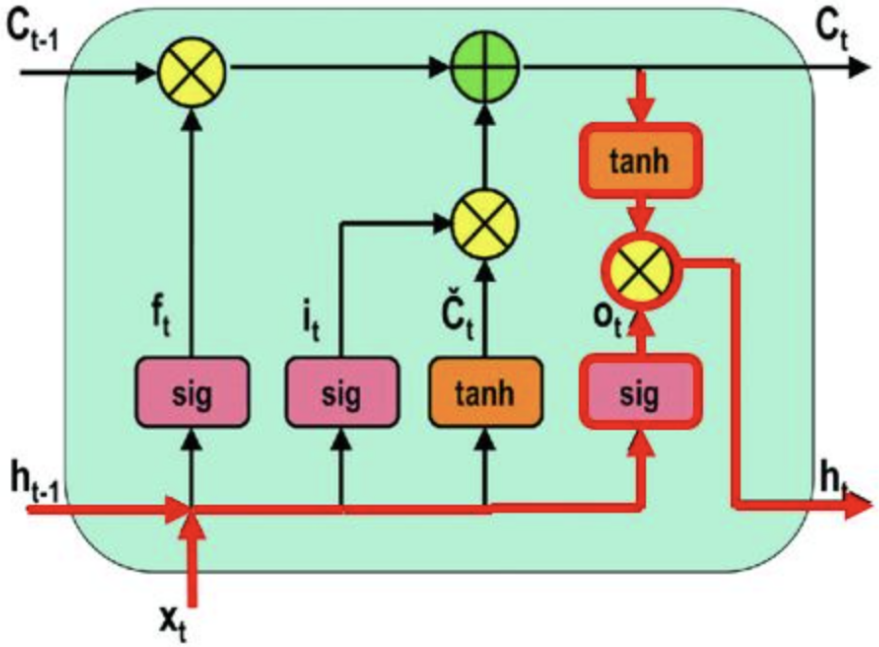
Output Gate Operation

output gate

previous input that goes into the hidden state

$$o^{(t)} = \sigma(W^o[h_{t-1}, x_t] + b^o)$$

LSTM: long short term memory
solution to the vanishing gradient problem



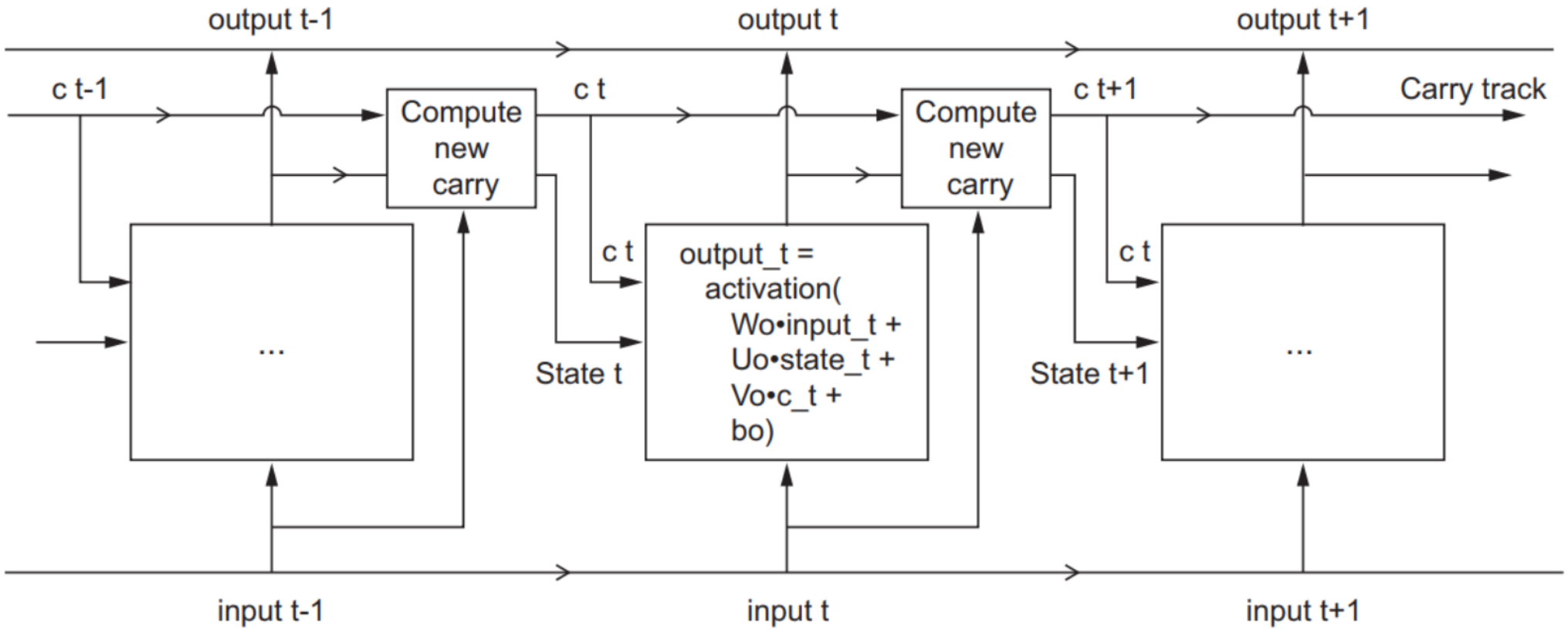
Output Gate Operation

hidden state

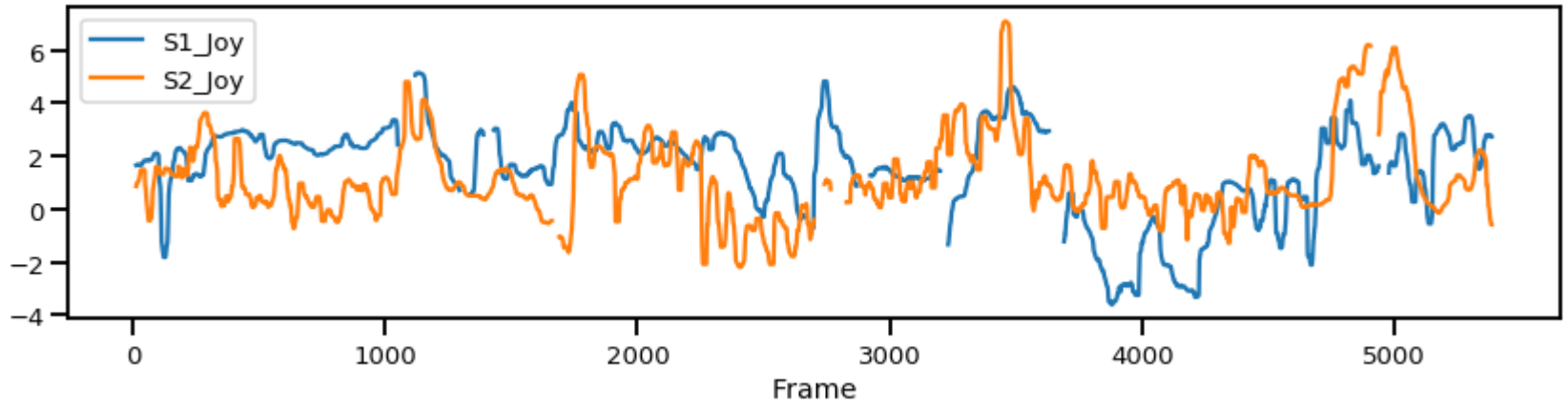
produces the new hidden states

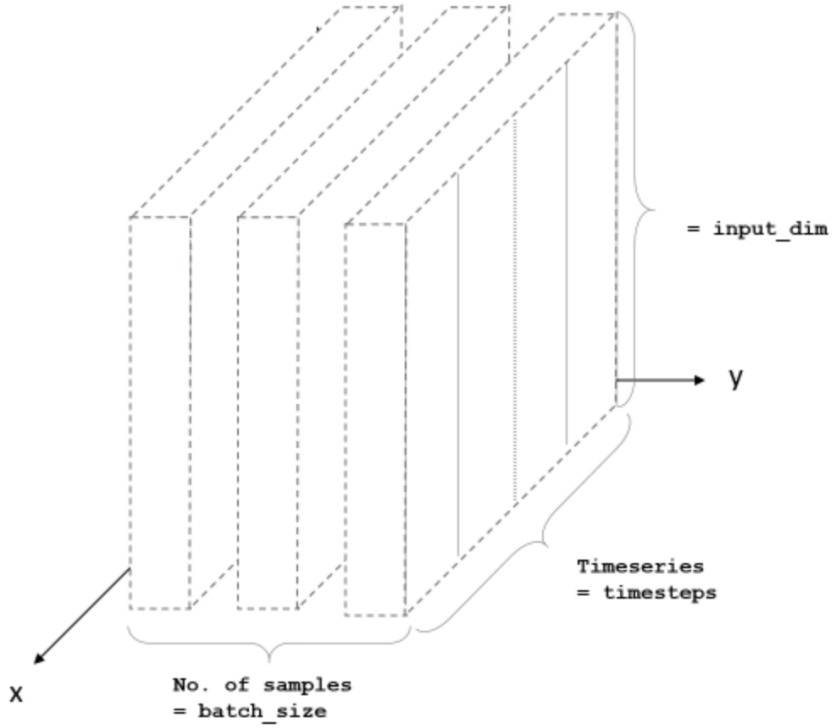
$$h^{(t)} = o^{(t)} * \tanh\left(C^{(t)}\right)$$

LSTM: long short term memory solution to the vanishing gradient problem



LSTM: how to actually run it





batch size: how many sequencies you pass at once

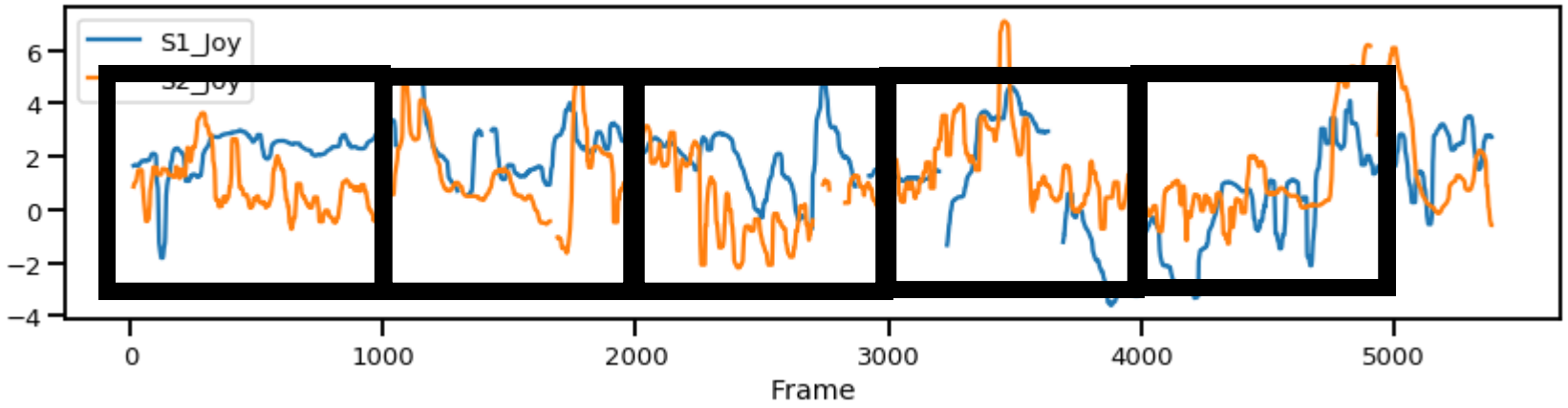
timeseries: how many time stamps in a sequence

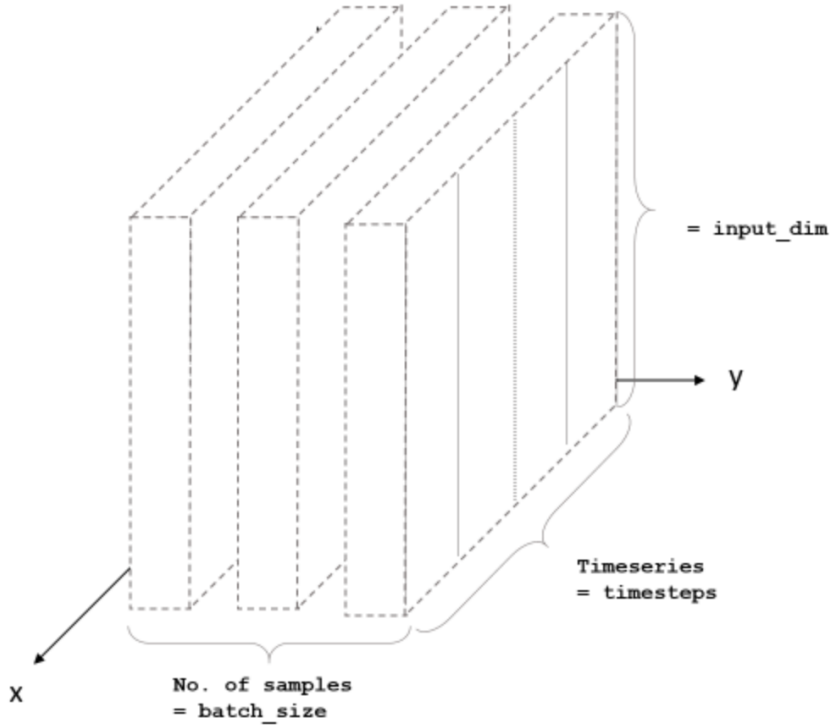
features: how many measurements in the time seris

even if you want to predict a single time series, you need many example

split the time series into chunks

LSTM: how to actually run it





batch size: N

timeseries: 1000

features: 2

```
model = Sequential()
model.add(LSTM(32, input_shape=(50, 2)))
model.add(Dense(2))
```

even if you want to predict a single time series, you need many example

split the time series into chunks

LSTM: how to actually run it

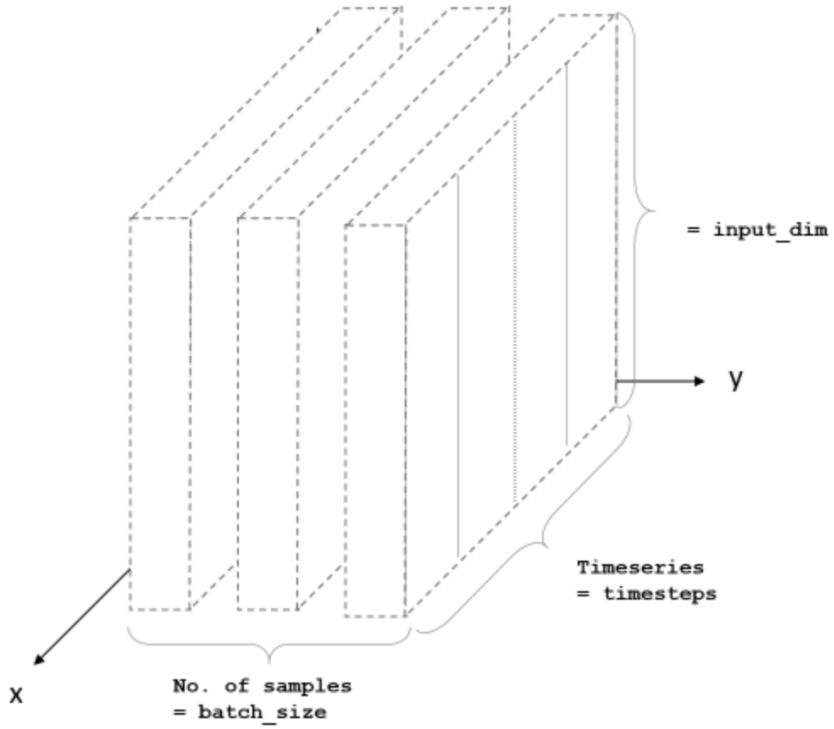
To be or not	to be? this	is the quest	ion. Whether	'tis nobler	in the mind

sequencies of 12 letters

batch size: N

timeseries: 12

features: 1



even if you want to predict a single time series, you need many example

split the time series into chunks

LSTM: how to actually run it

A fun notebook: create new paper titles (takes a lot to train tho so we cannot do it in class)

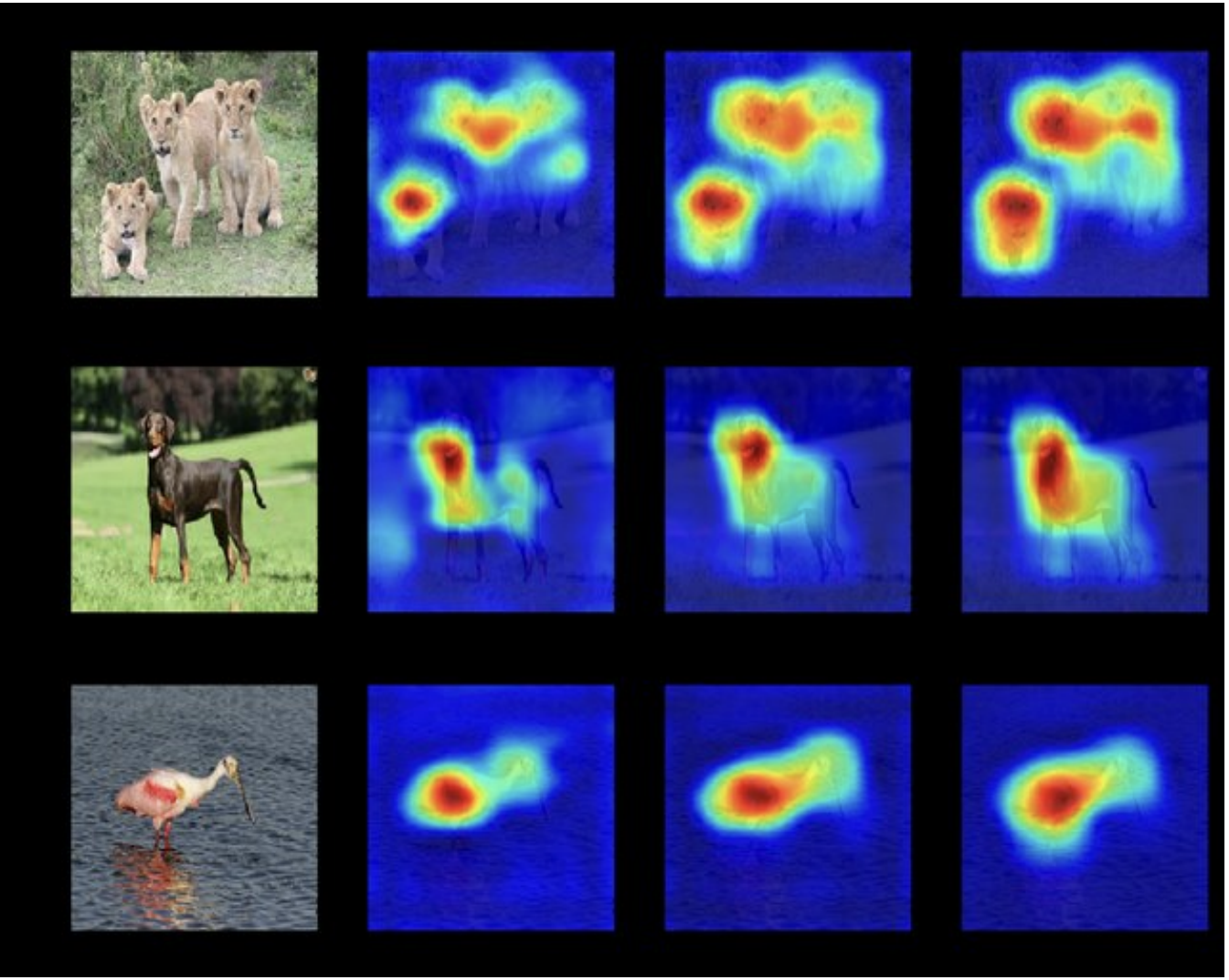
http://davidsd.org/2010/09/the-arxiv-according-to-arxiv-vs-snarxiv/

The arXiv According to arXiv vs. snarXiv — *Sep 17, 2010*

After more than 3/4 of a million guesses, in over 50,000 games played in 67 countries, the results are clear: Science sounds like gobbledygook.

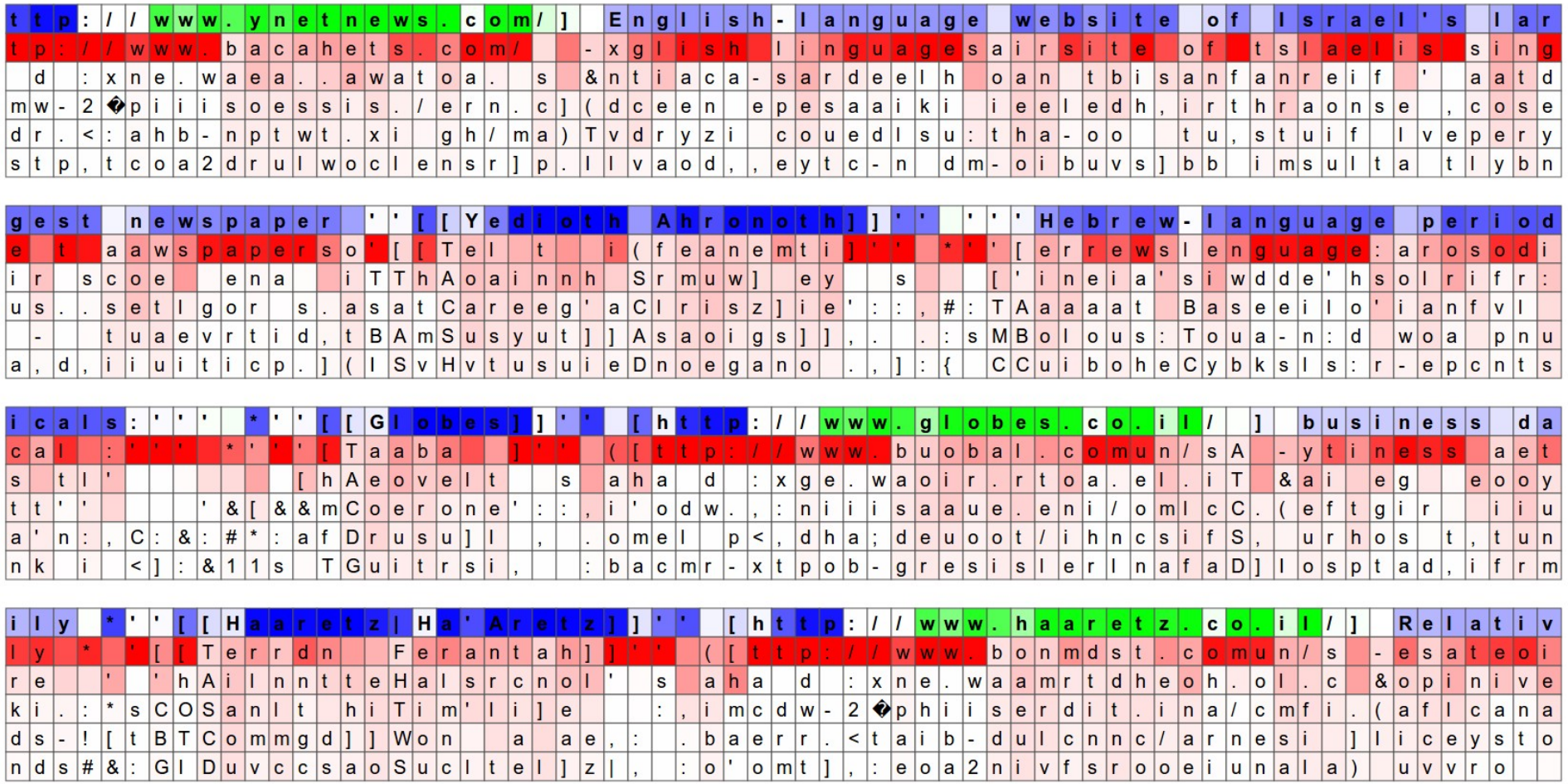
arXiv vs. snarXiv has been live for 6 months now, and it's time to take a look at the results. Here's how the game works. The user sees two titles: one is the title of an actual theoretical high energy physics paper on the arXiv, and the other is a completely fake title randomly generated by the snarXiv. The user guesses which one is real, finds out if they're right or wrong, and then starts over with a new pair of titles.

5 visualizing NNs



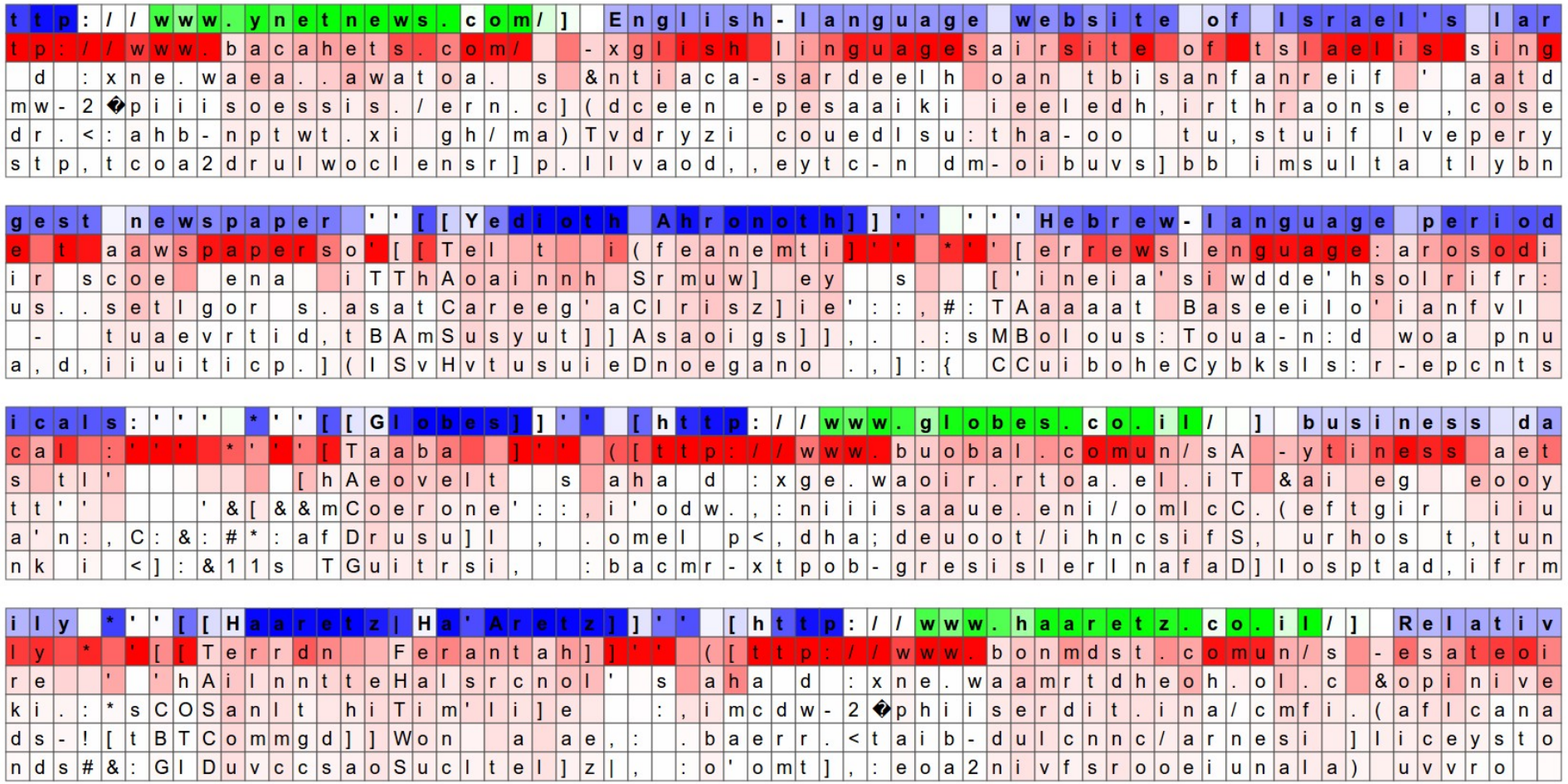
Saliency Maps

Visualizing the predictions and the “neuron” firings in the RNN https://sungsoo.github.io/2017/01/08/recurrent-neural-networks.html



Visualizing the predictions and the “neuron” firings in the RNN https://sungsoo.github.io/2017/01/08/recurrent-neural-networks.html

learning markdown syntax: URL's



"The guesses are colored by their probability (so dark red = judged as very likely, white = not very likely).

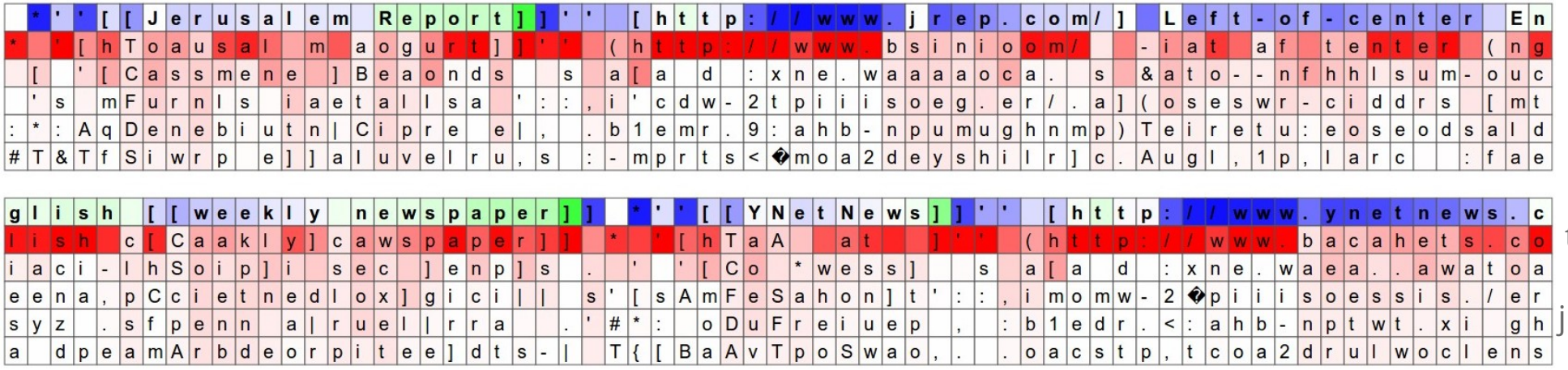
...

The input character sequence (blue/green) is colored based on the *firing* of a randomly chosen neuron in the hidden representation of the RNN. Think about it as green = very excited and blue = not very excited (... these are values between [-1,1] in the hidden state vector, which is just the gated and *tanh*'d LSTM cell state).

Intuitively, this is visualizing the firing rate of some neuron in the “brain” of the RNN while it reads the input sequence. Different neurons might be looking for different patterns.

Visualizing the predictions and the “neuron” firings in the RNN https://sungsoo.github.io/2017/01/08/recurrent-neural-networks.html

learning markdown syntax: [[]]



"The guesses are colored by their probability (so dark red = judged as very likely, white = not very likely).

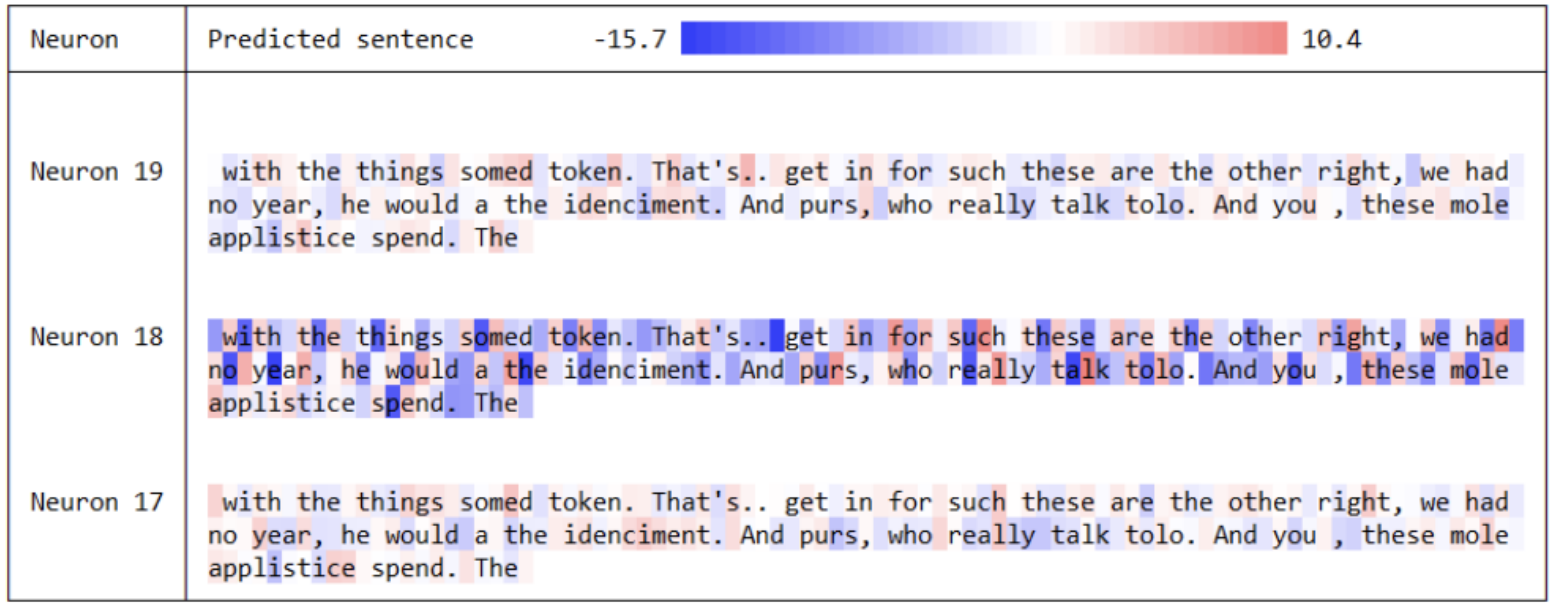
...

The input character sequence (blue/green) is colored based on the *firing* of a randomly chosen neuron in the hidden representation of the RNN. Think about it as green = very excited and blue = not very excited (... these are values between [-1,1] in the hidden state vector, which is just the gated and *tanh*'d LSTM cell state).

Intuitively, this is visualizing the firing rate of some neuron in the “brain” of the RNN while it reads the input sequence. Different neurons might be looking for different patterns.

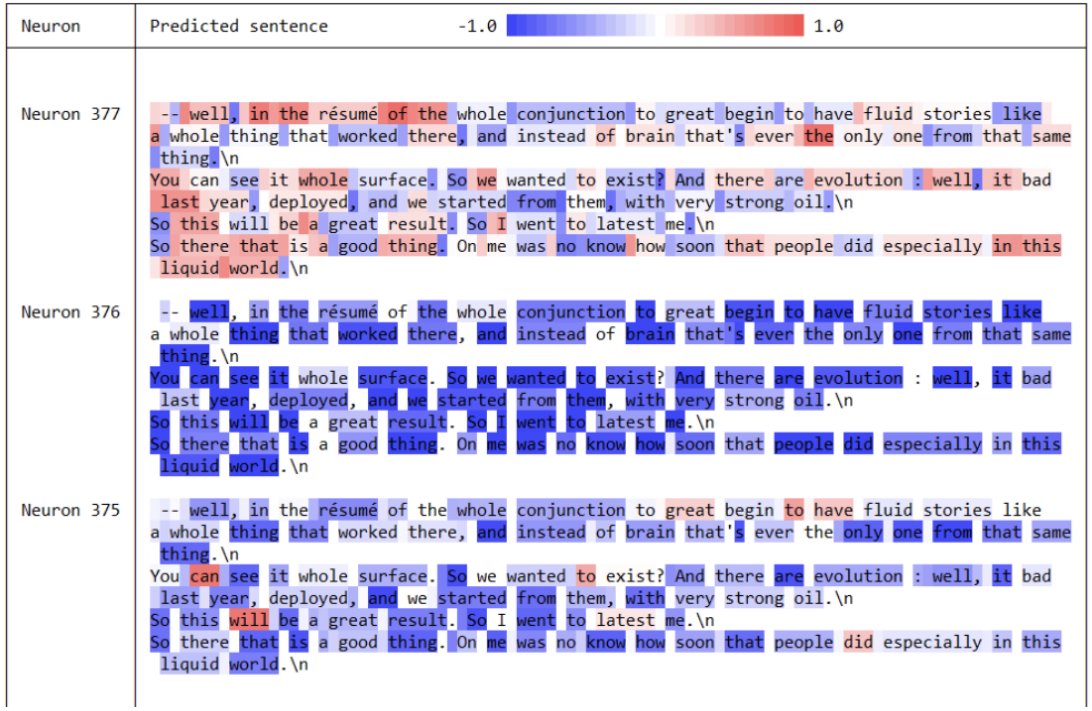
Visualizing the predictions and the “neuron” firings in the RNN

Vanilla RNN



Visualizing the predictions and the “neuron” firings in the RNN

LSTM



The Unreasonable Effectiveness of Recurrent Neural Networks

andrej karpathy

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Neural Network and Deep Learning

an excellent and free book on NN and DL

http://neuralnetworksanddeeplearning.com/index.html

Deep Learning An MIT Press book in preparation

Ian Goodfellow, Yoshua Bengio and Aaron Courville

https://www.deeplearningbook.org/lecture_slides.html

History of NN

https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history2.html

Gradient Descent

https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html