

Recurrent Neural Networks

deep learning 3

Raoul Grouls, 2 December 2025

Neural networks

Neural networks

$$\sigma(wx + b)$$

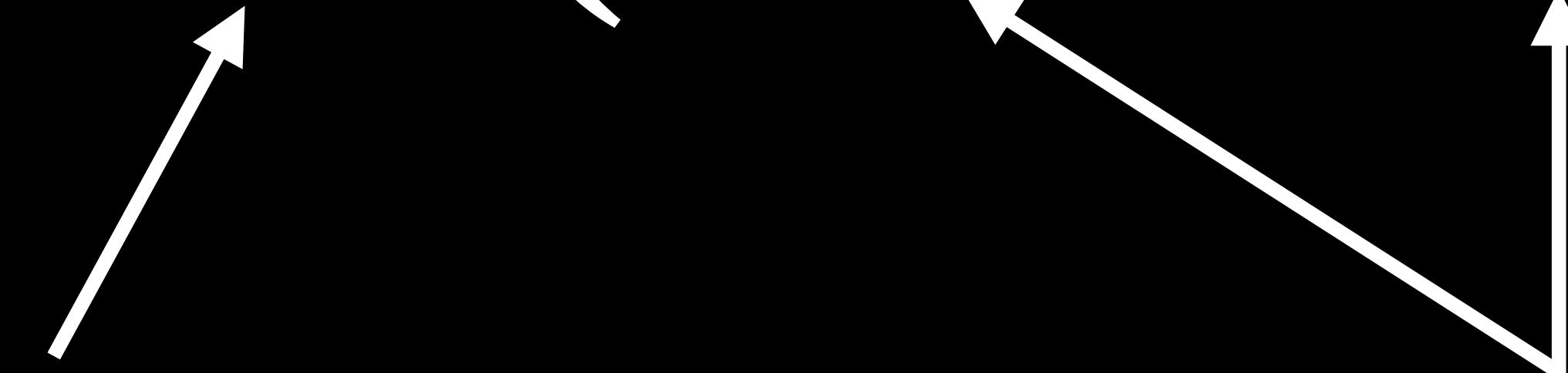
Neural networks

$$\sigma(wx + b)$$

A diagram illustrating a single neuron's activation function. At the bottom, the word "Input" is written next to a vertical white arrow pointing upwards. Above the arrow, the mathematical expression $\sigma(wx + b)$ is displayed in large white font.

Neural networks

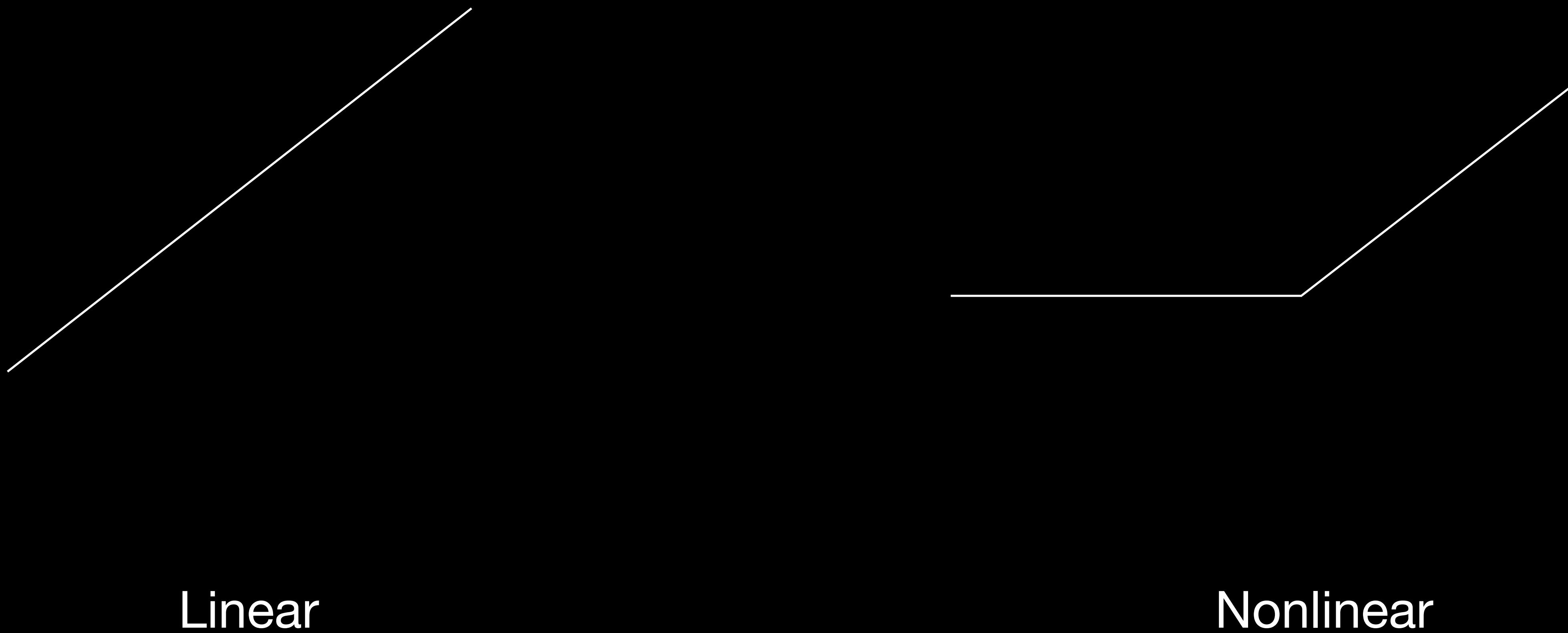
$$\sigma(wx + b)$$



Non-linear transformation

Linear transformation

Neural networks



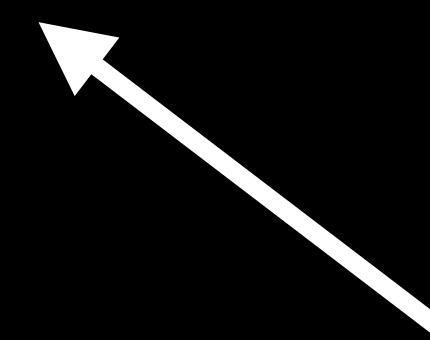
Neural networks

$$\sigma(wx + b)$$


Learnable

Neural networks

$$\hat{y} = \sigma(wx + b)$$



Prediction

Neural networks

$$\hat{y} = \sigma(wx + b)$$

$$z = (y - \hat{y})^2$$

Change w and b such that z is minimal

Neural networks

$$\hat{y} = \sigma(wx + b)$$

$$\left. \begin{array}{l} \frac{\partial z}{\partial w} \\ \frac{\partial z}{\partial b} \end{array} \right\} \text{Gradient}$$

How much do we need to change w and b

Neural networks

$$w \leftarrow w + \eta \frac{\partial z}{\partial w}$$

$$b \leftarrow b + \eta \frac{\partial z}{\partial b}$$

Update the weights

Neural networks

$$\left. \begin{aligned} w &\leftarrow w + \eta \frac{\partial z}{\partial w} \\ b &\leftarrow b + \eta \frac{\partial z}{\partial b} \end{aligned} \right\} \text{Optimizer}$$

Neural networks

$$w \leftarrow w + \eta \frac{\partial z}{\partial w}$$

Learning rate

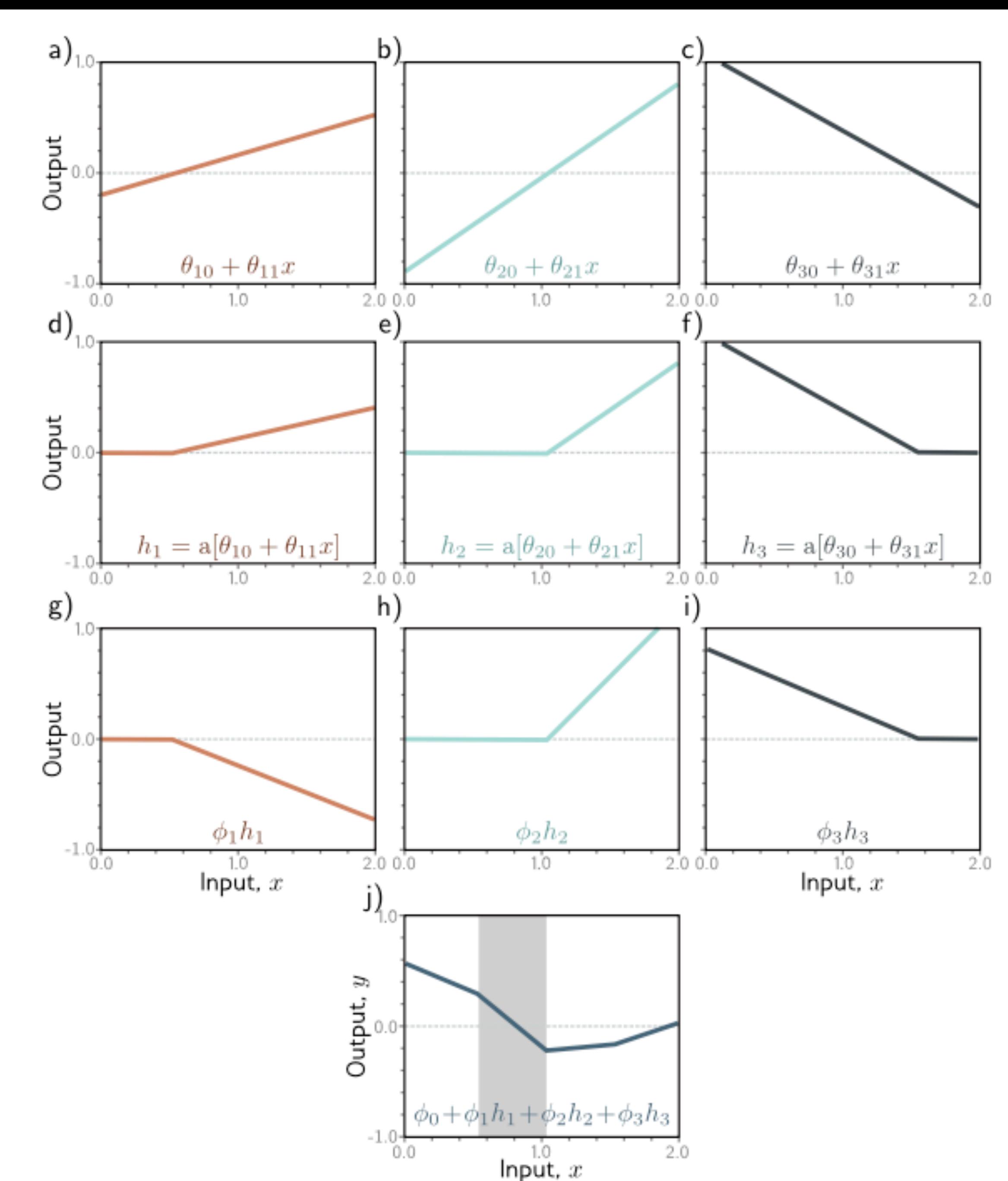
$$b \leftarrow b + \eta \frac{\partial z}{\partial b}$$

Universal approximation theorem

Any function can be approximated to arbitrary precision

Universal approximation theorem

- Any continuous function on a finite interval $[a, b]$
- Can be approximated to arbitrary precision
- By a shallow neural network $f_2 \circ \sigma \circ f_1$ where f are linear transformations and σ is a nonlinear transformation



Images

The curse of dimensionality



$O(n^2)$

Width x Height



28x28



100x100



200x200



400x400

Width x Height

Features



28x28

784



100x100

10.000



200x200

40.000



400x400

160.000

Width x Height

Features

Weights



28x28

784

614.656



100x100

10.000

100.000.000



200x200

40.000

1.600.000.000



400x400

160.000

25.600.000.000



Convolutions

Convolutions

	1			
1	-1	1		
	-1			
1	-1			

0	0	0
0	1	0
0	0	0

Convolutions

0	1	0	0	0
1	-1	1	0	0
0	0	0	0	0
0	-1	0	0	0
0	1	-1	0	0

0	0	0
0	1	0
0	0	0

Convolutions

0	1	0	0	0
1	-1	1	0	0
0	0	0	0	0
0	-1	0	0	0
0	1	-1	0	0

0	0	0
0	1	0
0	0	0

Convolutions

0	1	0	0	0
1	-1	1	0	0
0	0	0	0	0
0	-1	0	0	0
0	1	-1	0	0

0	0	0
0	1	0
0	0	0

-1		

Convolutions

0	1	0	0	0
1	-1	1	0	0
0	0	0	0	0
0	-1	0	0	0
0	1	-1	0	0

0	0	0
0	1	0
0	0	0

-1	1	

Convolutions

0	1	0	0	0
1	-1	1	0	0
0	0	0	0	0
0	-1	0	0	0
0	1	-1	0	0

0	0	0
0	1	0
0	0	0

-1	1	0

Convolutions

0	1	0	0	0
1	-1	1	0	0
0	0	0	0	0
0	-1	0	0	0
0	1	-1	0	0

0	0	0
0	1	0
0	0	0

-1	1	0
0		

Convolutions

0	1	0	0	0
1	-1	1	0	0
0	0	0	0	0
0	-1	0	0	0
0	1	-1	0	0

0	0	0
0	1	0
0	0	0

-1	1	0
0	0	

Convolutions

0	1	0	0	0
1	-1	1	0	0
0	0	0	0	0
0	-1	0	0	0
0	1	-1	0	0

0	0	0
0	1	0
0	0	0

-1	1	0
0	0	0

Convolutions

0	1	0	0	0
1	-1	1	0	0
0	0	0	0	0
0	-1	0	0	0
0	1	-1	0	0

0	0	0
0	1	0
0	0	0

-1	1	0
0	0	0
-1		

Convolutions

0	1	0	0	0
1	-1	1	0	0
0	0	0	0	0
0	-1	0	0	0
0	1	-1	0	0

0	0	0
0	1	0
0	0	0

-1	1	0
0	0	0
-1	0	

Convolutions

0	1	0	0	0
1	-1	1	0	0
0	0	0	0	0
0	-1	0	0	0
0	1	-1	0	0

0	0	0
0	1	0
0	0	0

-1	1	0
0	0	0
-1	0	0

Convolutions

0	1	0	0	0
1	-1	1	0	0
0	0	0	0	0
0	-1	0	0	0
0	1	-1	0	0

0	0	0
0	1	0
0	0	0

-1	1	0
0	0	0
-1	0	0

Identity kernel

Convolutions

0	1	0	0	0
1	-1	1	0	0
0	0	0	0	0
0	-1	0	0	0
0	1	-1	0	0

w	w	w
w	w	w
w	w	w

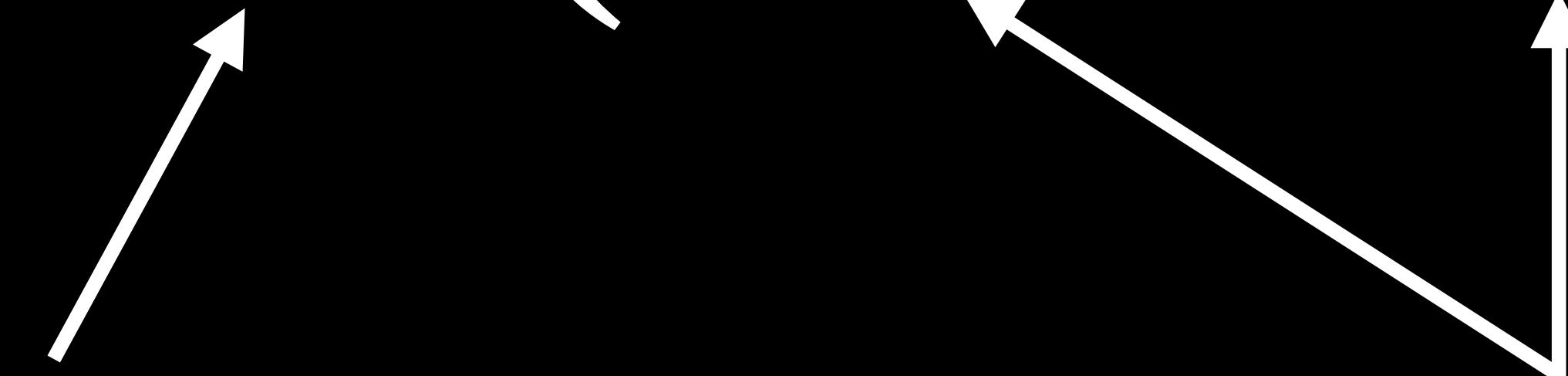
Learnable

\hat{y}	\hat{y}	\hat{y}
\hat{y}	\hat{y}	\hat{y}
\hat{y}	\hat{y}	\hat{y}

$O(k)$

Neural networks

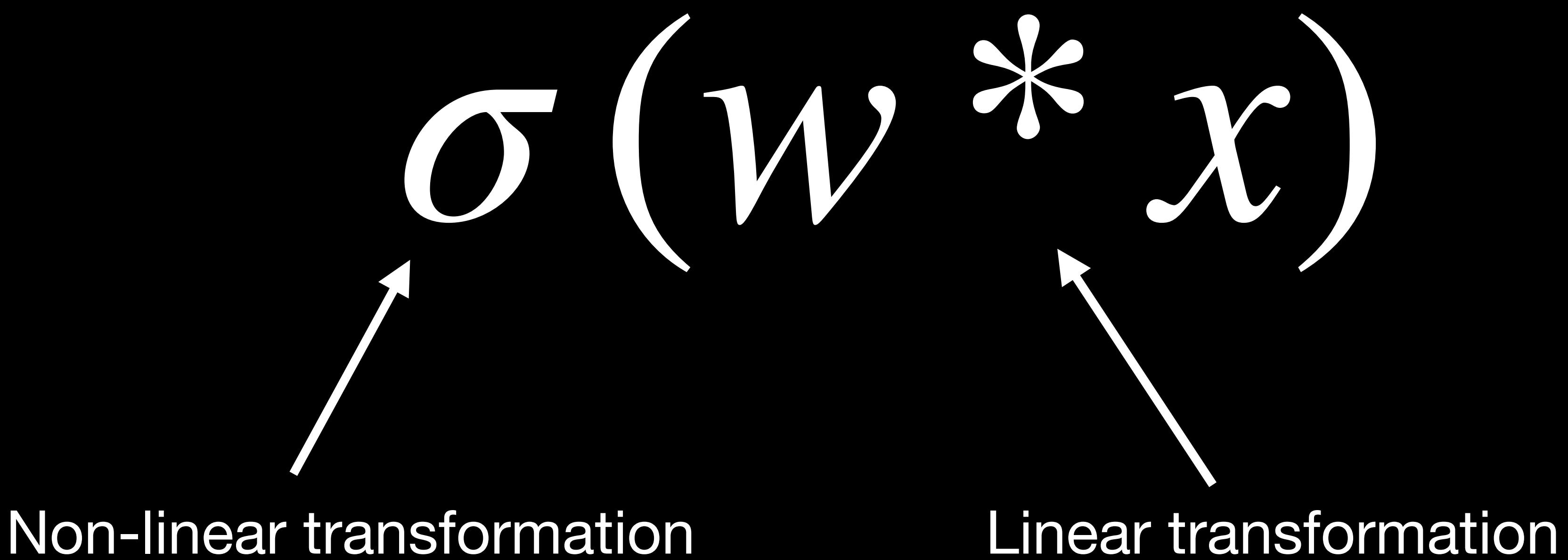
$$\sigma(wx + b)$$



Non-linear transformation

Linear transformation

Convolutions

$$\sigma(w * x)$$


The diagram illustrates the components of a convolution operation. At the top center is the mathematical expression $\sigma(w * x)$. A white arrow points from the left towards the w in the expression, labeled "Non-linear transformation". Another white arrow points from the right towards the x , labeled "Linear transformation".

Deep neural networks

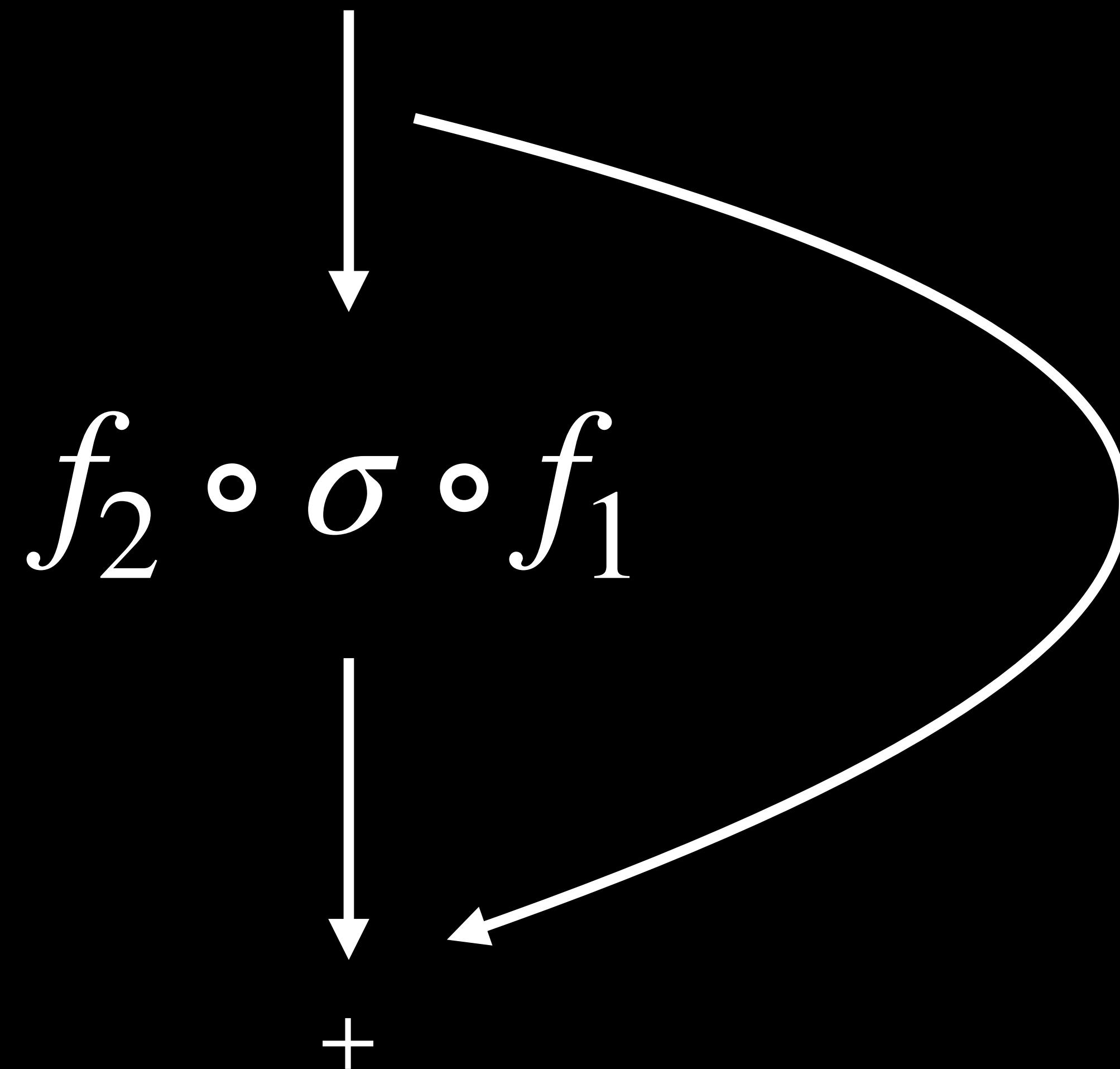
Deep neural networks



$$f_2 \circ \sigma \circ f_1$$



Deep neural networks



Timeseries

Motivation

A lot of data is sequential, varying over time:

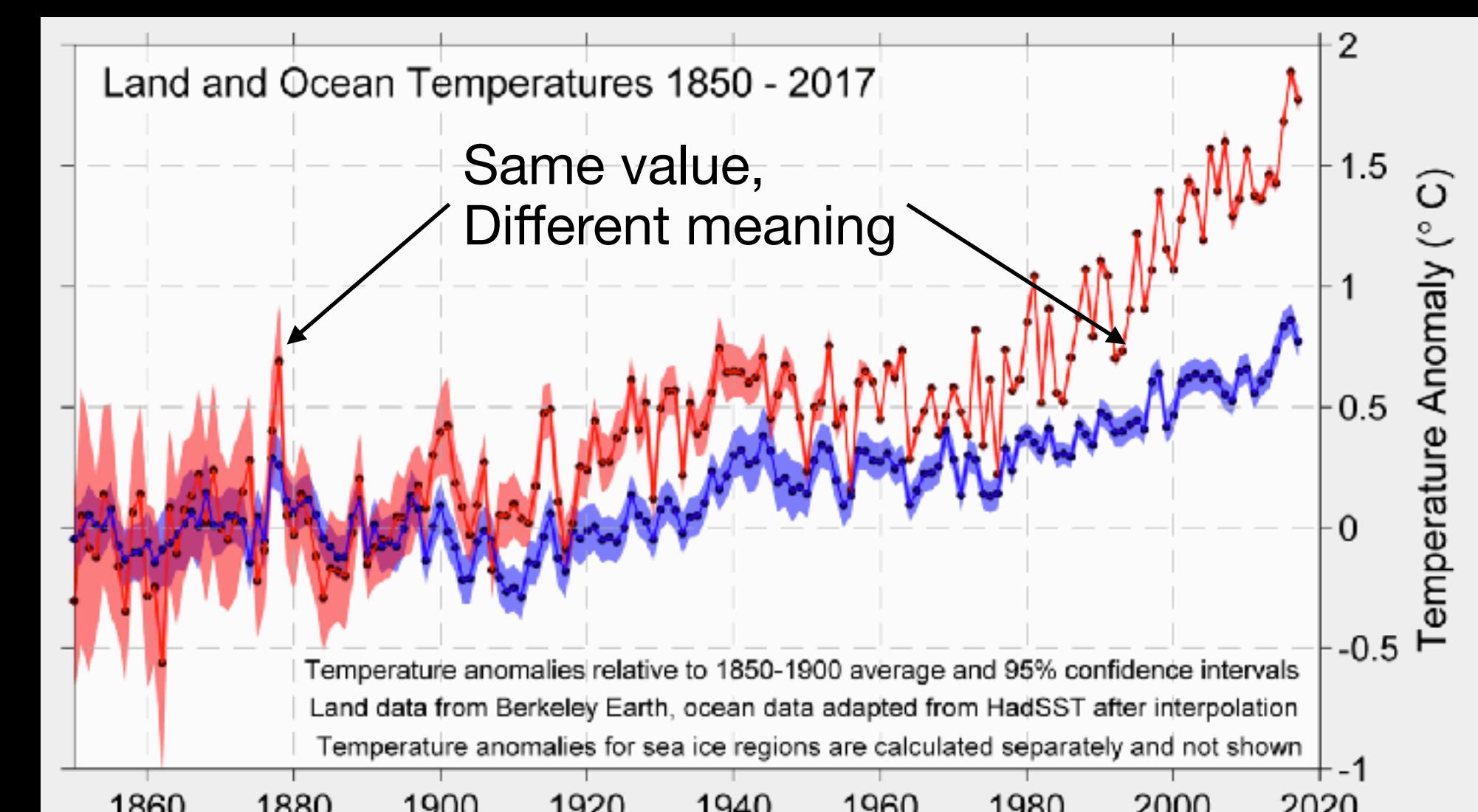
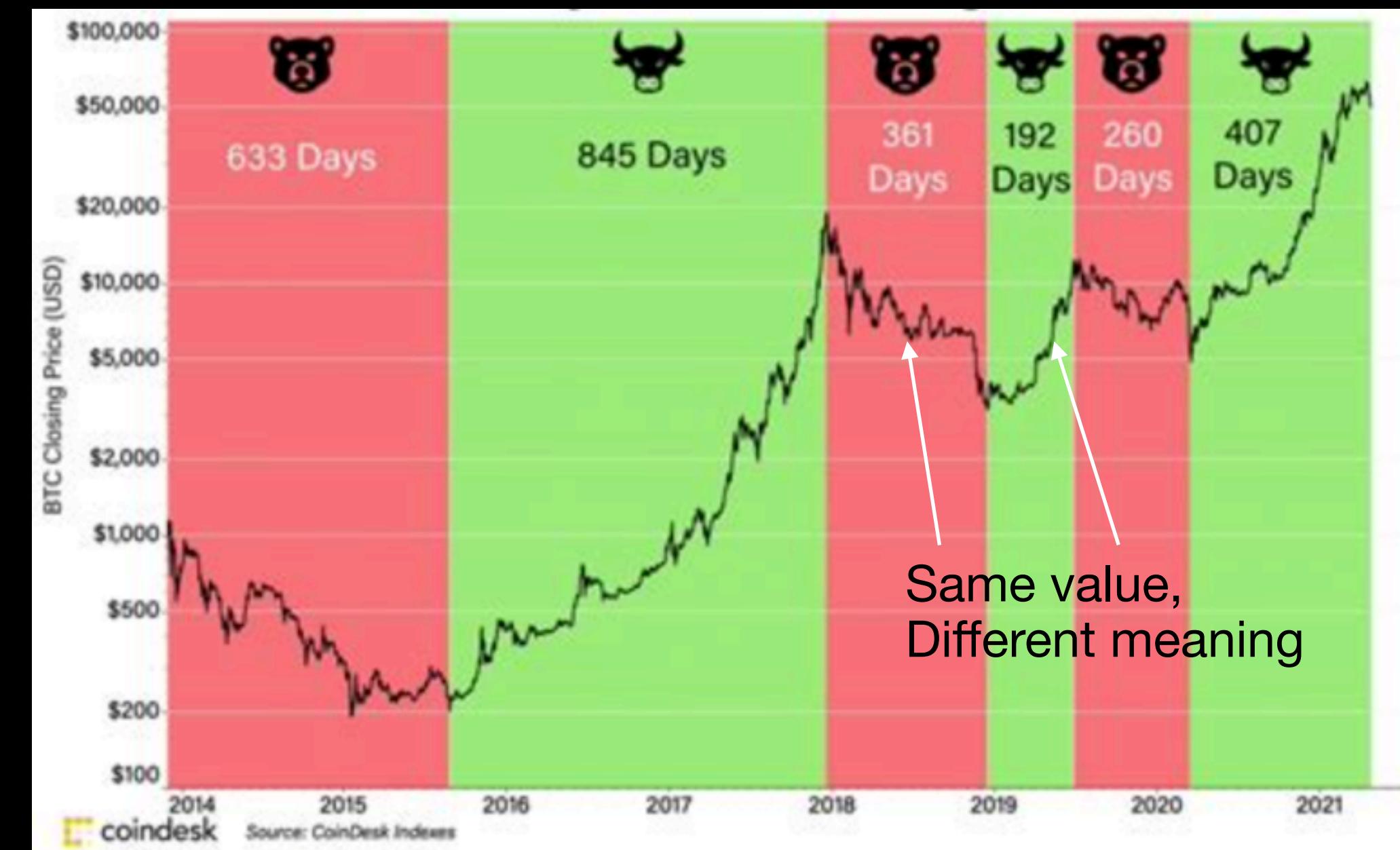
- Sentences
- Music
- EEG
- Movement
- Markets

Motivation

With sequences, the past offers context:

- Ik krijg geld van de **bank**
- Ik wil een nieuwe **bank** aanschaffen

We need the past to make sense of the future.



Data considerations

We need to worry about:

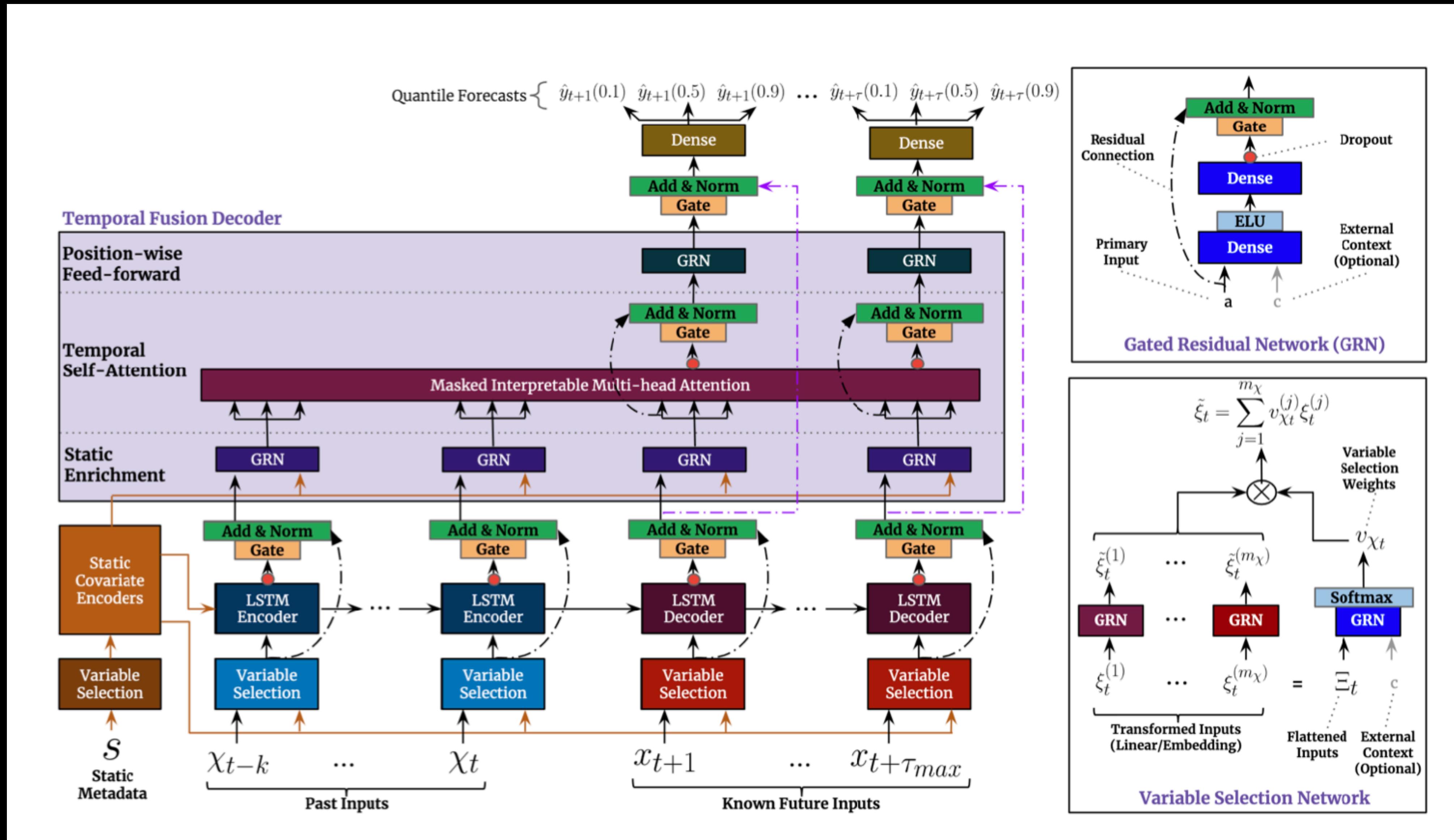
- How much of the past will we need (window)
- How much of the future do we want to predict (horizon)
- How to prepare the data without leaking data

For the last point, we need to be very careful not to “leak” the future back into the present.

History of RNNs

- 1982 RNN are discovered by John Hopfield
- 1995 The LSTM architecture was proposed with input and output gates
- 1999 Forget gates were added
- 2009 LSTM won the handwriting recognition competition
- 2013 LSTM outperformed other models at natural speech recognition
- 2014 GRU architecture was introduced
- 2017 probabilistic forecasting (DeepAR, MQRNN, TFT)

Temporal Fusion Transformer, Lim et al. (2021)



Hidden states

State

- A state gives context to new input
- Influences which elements of the input requires attention, and which elements can be ignored
- New input can change the state, such that attention is shifted to other elements of the input

X



X in context 1



X in context 2





Gate open to state

Gate blocks new input



Gate blocks new input

Gate open to state

Concatenation

1	1	-1	0	1
---	---	----	---	---



x_t

New Input

Concatenation

0	1	0	0	0
1	-1	1	0	0
0	0	0	0	0
0	-1	0	0	0
0	1	-1	0	0



Previous state

h_{t-1}

1	1	-1	0	1
---	---	----	---	---

x_t

Concatenation

0	1	0	0	0
1	-1	1	0	0
0	0	0	0	0
0	-1	0	0	0
0	1	-1	0	0

h_{t-1}

1	1	-1	0	1
---	---	----	---	---

x_t

0	1	0	0	0
1	-1	1	0	0
0	0	0	0	0
0	-1	0	0	0
0	1	-1	0	0
1	1	-1	0	1

$[x_t, h_{t-1}]$

New state

$$y = \sigma(WX + b)$$

Equivalent output

$$\left\{ \begin{array}{l} h_t = \sigma(W_x X_t + \textcolor{red}{W_h h_{t-1}} + b) \quad \leftarrow \text{Slower} \\ h_t = \sigma(W[X_t, \textcolor{red}{h_{t-1}}] + b) \quad \leftarrow \text{Faster} \end{array} \right.$$

RNN

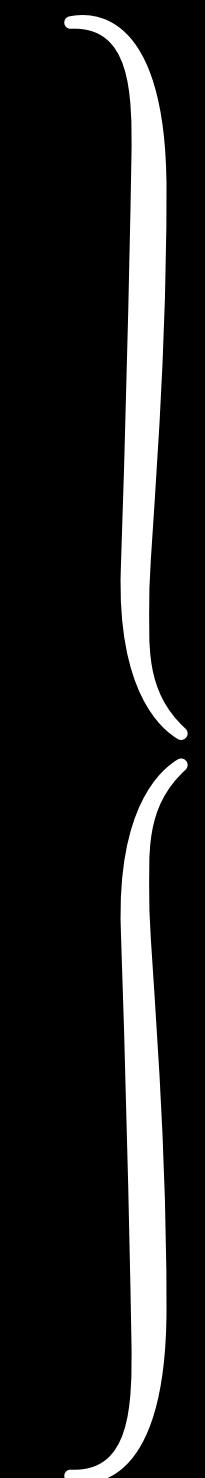
\hat{y}



$$\sigma(wx + b)$$

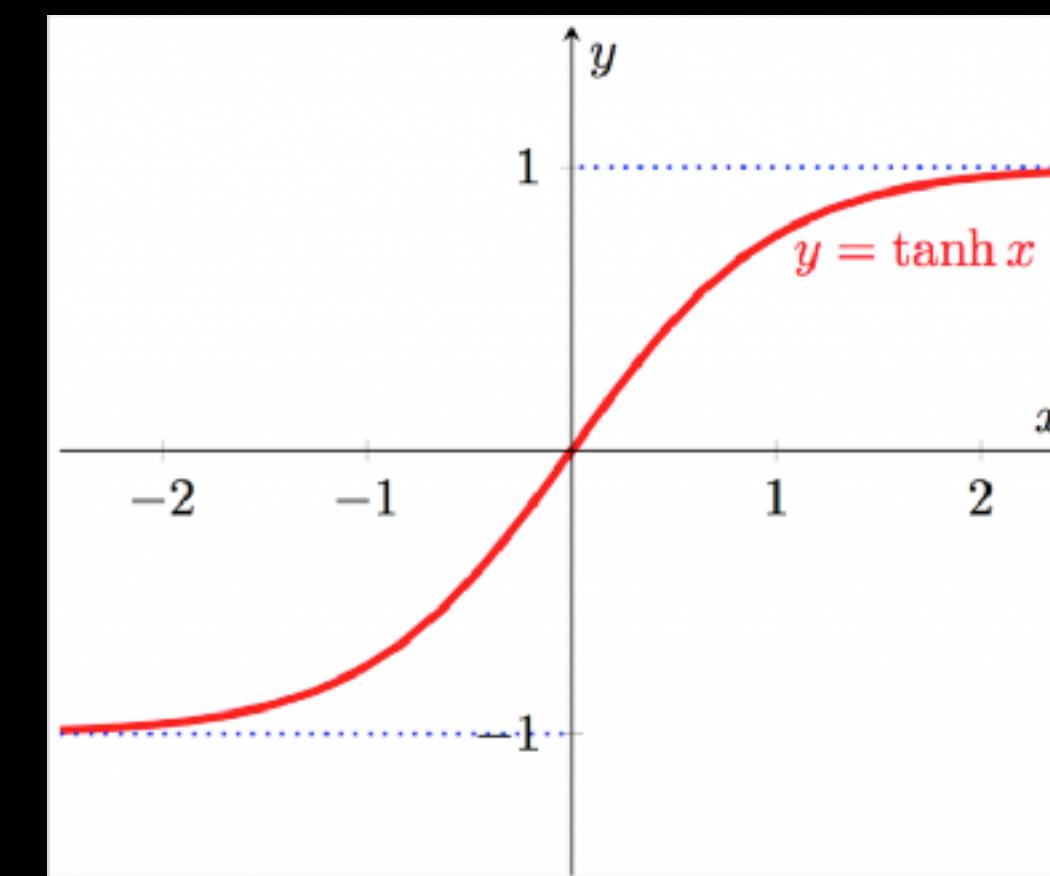
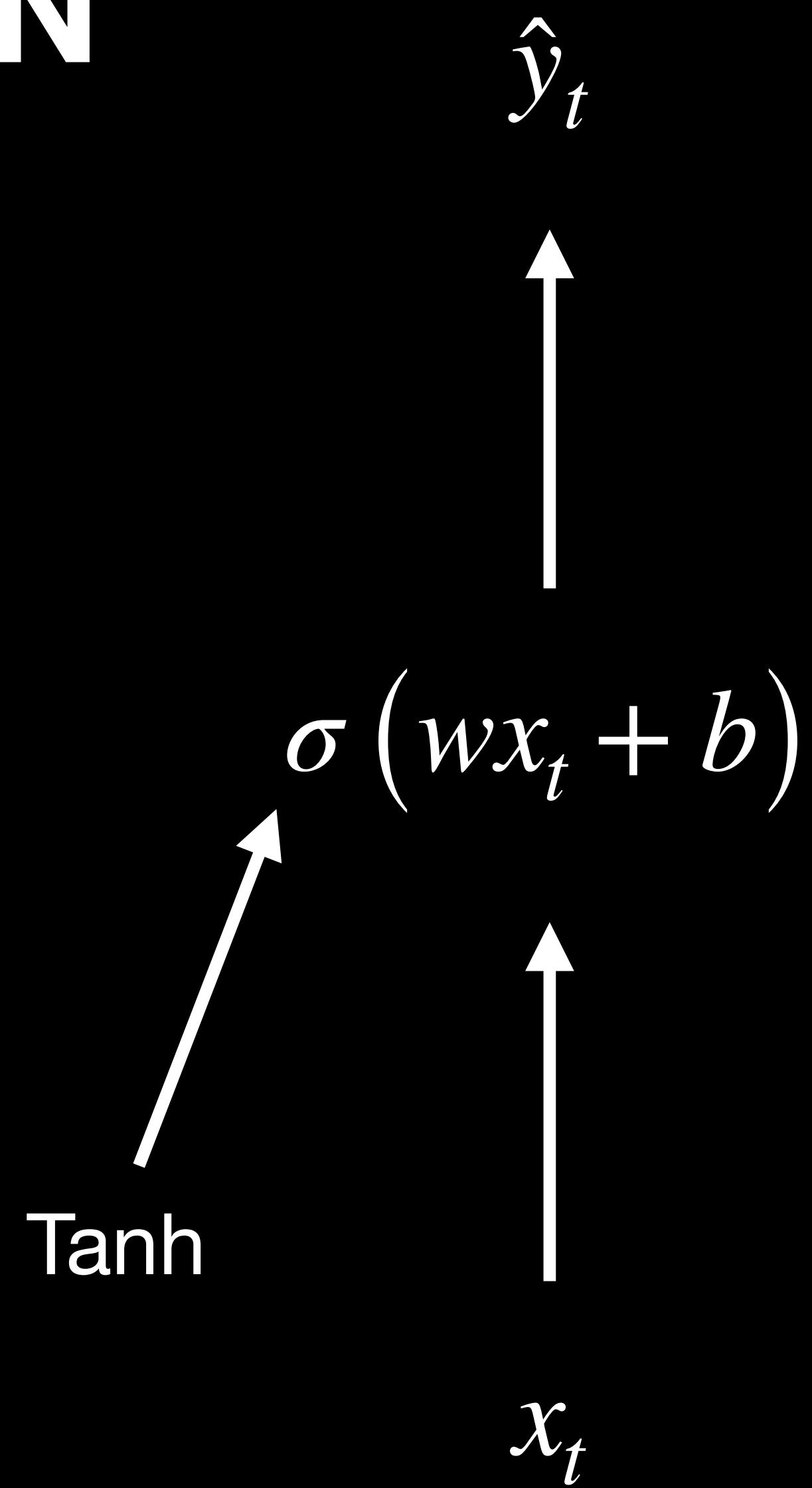


x

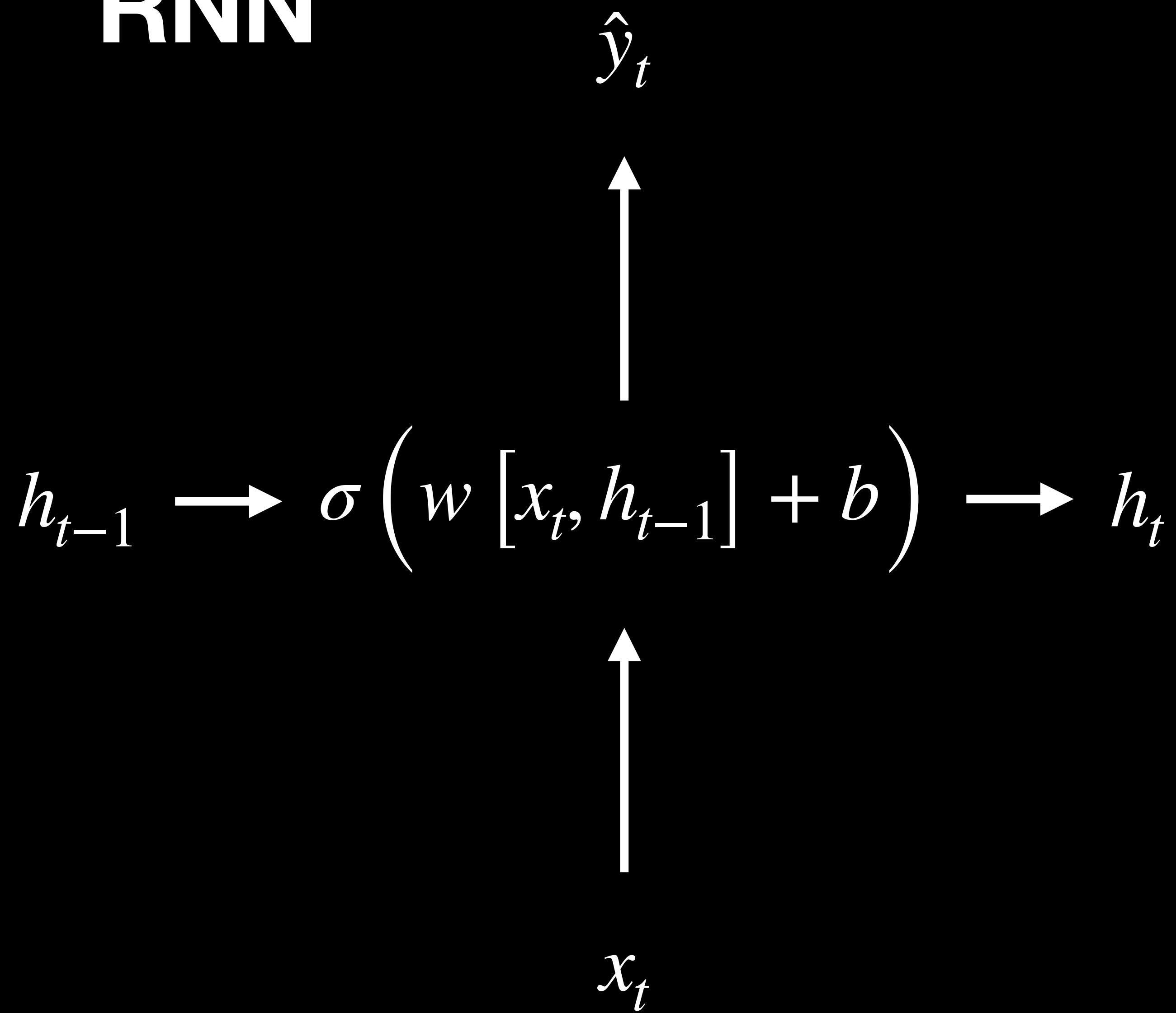


Neural network

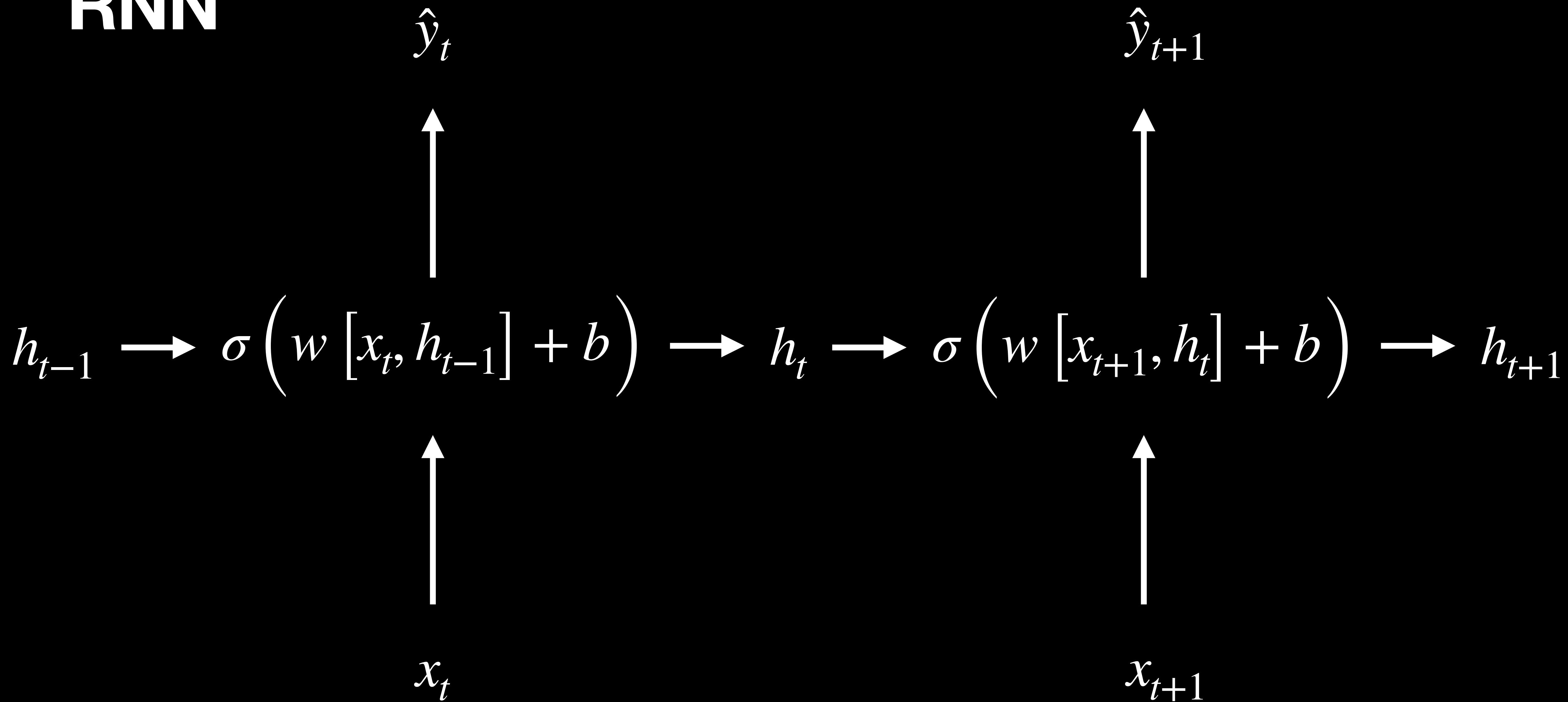
RNN

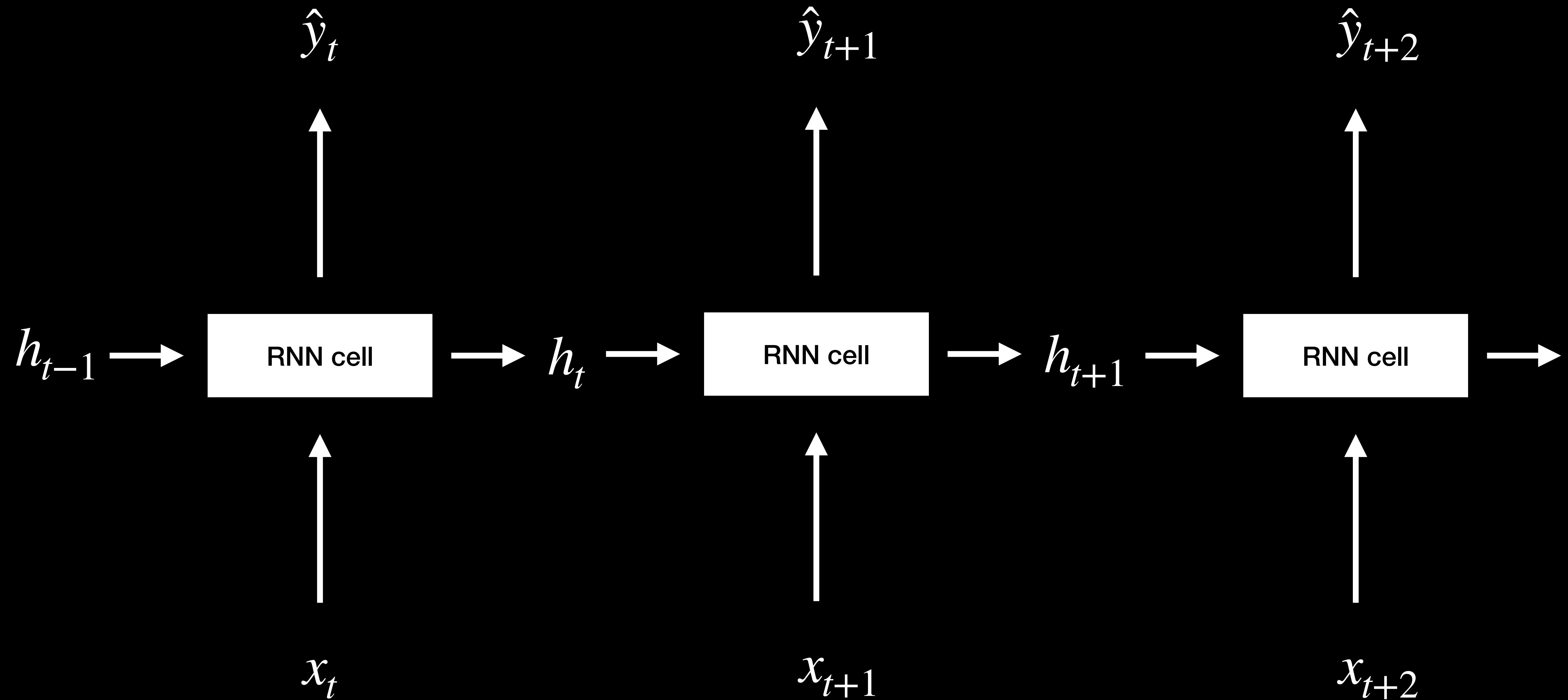


RNN



RNN





$[y_1, y_2, \dots, y_T]$  $[x_1, x_2, \dots, x_T]$

The art of forgetting

RNNs have not explicit way to forget or retain memory.

We can make this a bit more advanced by adding gates.

A gate Γ controls

- what part of the past we retain
- what part we forget.

GRU - Remember & forget

Gated Residual Unit

We need to be able to:

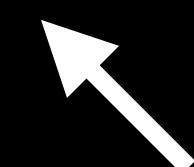
- *Remember* the past, and completely ignore the new state
- *Forget* the past, and focus on the present
- *Something in between* where we find a ratio between forgetting and remembering.

We also want to gate to be influenced by both the new input and the old state.

Gates

0	1	0	0
1	-1	1	0
0	0	0	0
0	-1	0	0

X



Input

Gates

0	1	0	0
1	-1	1	0
0	0	0	0
0	-1	0	0

X



Γ

→ Hadamard product

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Gates

0	1	0	0
1	-1	1	0
0	0	0	0
0	-1	0	0

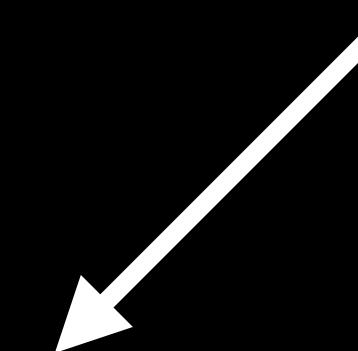
X

\otimes

Γ

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Same shape as X



Gates

0	1	0	0
1	-1	1	0
0	0	0	0
0	-1	0	0

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

X

\otimes

Γ

Gate

Gates

0	1	0	0
1	-1	1	0
0	0	0	0
0	-1	0	0

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

 X \otimes Γ $=$ Y

Gates

0	1	0	0
1	-1	1	0
0	0	0	0
0	-1	0	0

 \otimes

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

 Γ $=$ Y

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Gates

0	1	0	0
1	-1	1	0
0	0	0	0
0	-1	0	0

1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

 X \otimes Γ $=$ Y

Gates

0	1	0	0
1	-1	1	0
0	0	0	0
0	-1	0	0

1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

0	1	0	0
1	-1	1	0
0	0	0	0
0	-1	0	0

 X \otimes Γ $=$ Y

Gates

0	1	0	0
1	-1	1	0
0	0	0	0
0	-1	0	0

0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5

 X \otimes Γ $=$ Y

Gates

0	1	0	0
1	-1	1	0
0	0	0	0
0	-1	0	0

 \otimes

0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5
0.5	0.5	0.5	0.5

 Γ $=$ Y

0	0.5	0	0
0.5	-0.5	0.5	0
0	0	0	0
0	-0.5	0	0

Gates

$$X \in \mathbb{R}^{w \times h}$$

0.4	0.2	2.1	0.01
2.1	0.3	-1.2	0.2
-1.2	0.2	1.3	0.3
1.3	-0.1	0	-1.8

$$\Gamma \in \mathbb{R}^{w \times h}$$

w	w	w	w
w	w	w	w
w	w	w	w
w	w	w	w

$$Y \in \mathbb{R}^{w \times h}$$

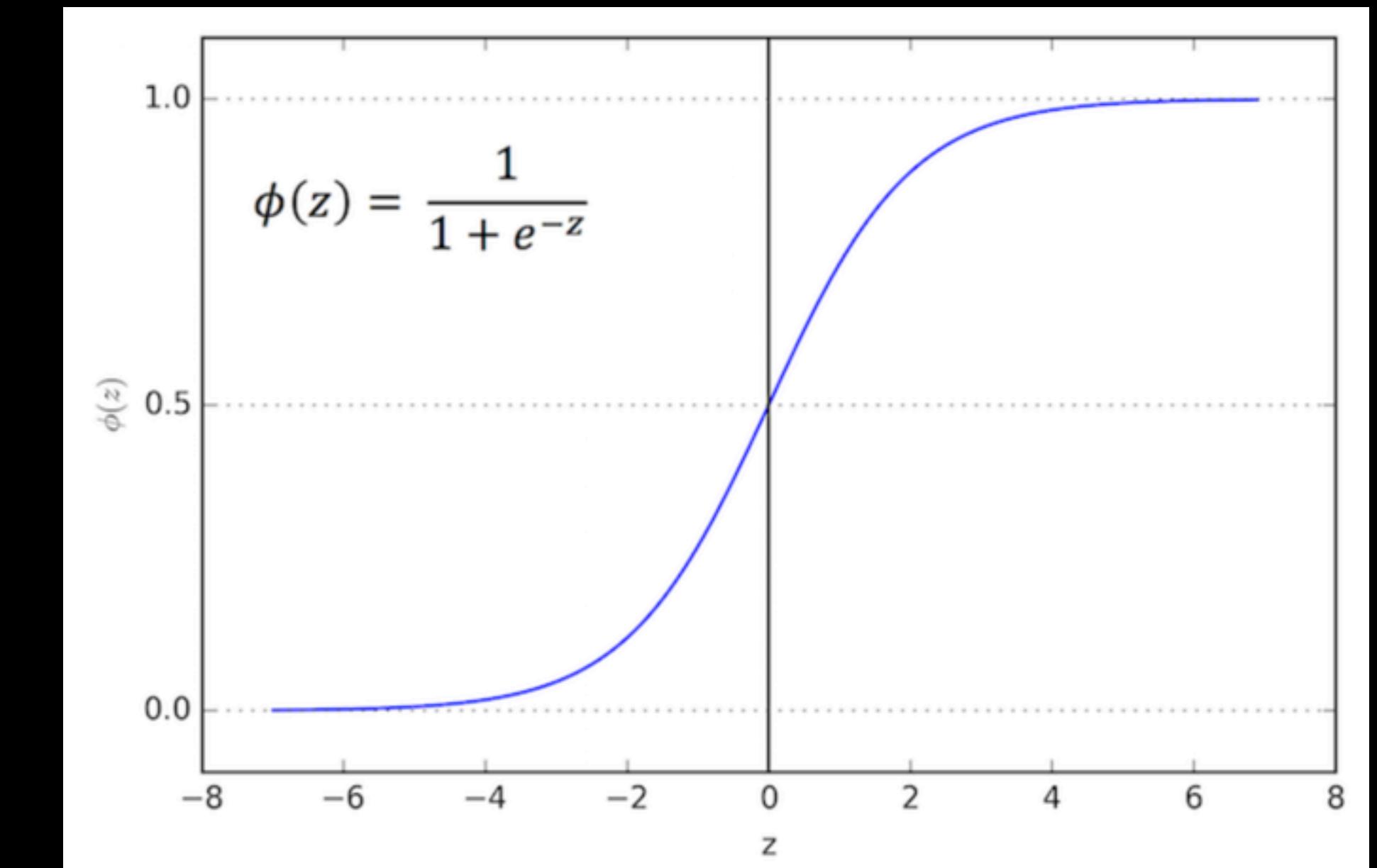
y	y	y	y
y	y	y	y
y	y	y	y
y	y	y	y

$$X \otimes \Gamma = Y$$

Gates

$\Gamma = \sigma(W[X_t, h_{t-1}] + b)$

Sigmoid



GRU

Gate \longrightarrow

$$\Gamma = \sigma(W_\Gamma[X_t, h_{t-1}] + b_\Gamma)$$
$$\tilde{h}_t = \tanh(W_H[X_t, h_{t-1}] + b_H)$$
$$h_t = \Gamma \otimes h_{t-1} + (1 - \Gamma) \otimes \tilde{h}_t$$

GRU

$$\begin{aligned}\Gamma &= \sigma(W_\Gamma[X_t, h_{t-1}] + b_\Gamma) \\ \text{Candidate State} \longrightarrow \tilde{h}_t &= \tanh(W_H[X_t, h_{t-1}] + b_H) \\ h_t &= \Gamma \otimes h_{t-1} + (1 - \Gamma) \otimes \tilde{h}_t\end{aligned}$$

GRU

$$\Gamma = \sigma(W_\Gamma[X_t, h_{t-1}] + b_\Gamma)$$

$$\tilde{h}_t = \tanh(W_H[X_t, h_{t-1}] + b_H)$$

Next State $\longrightarrow h_t = \Gamma \otimes h_{t-1} + (1 - \Gamma) \otimes \tilde{h}_t$

GRU

Same input



$$\Gamma = \sigma(W_\Gamma[X_t, h_{t-1}] + b_\Gamma)$$

$$\tilde{h}_t = \tanh(W_H[X_t, h_{t-1}] + b_H)$$

$$h_t = \Gamma \otimes h_{t-1} + (1 - \Gamma) \otimes \tilde{h}_t$$

GRU

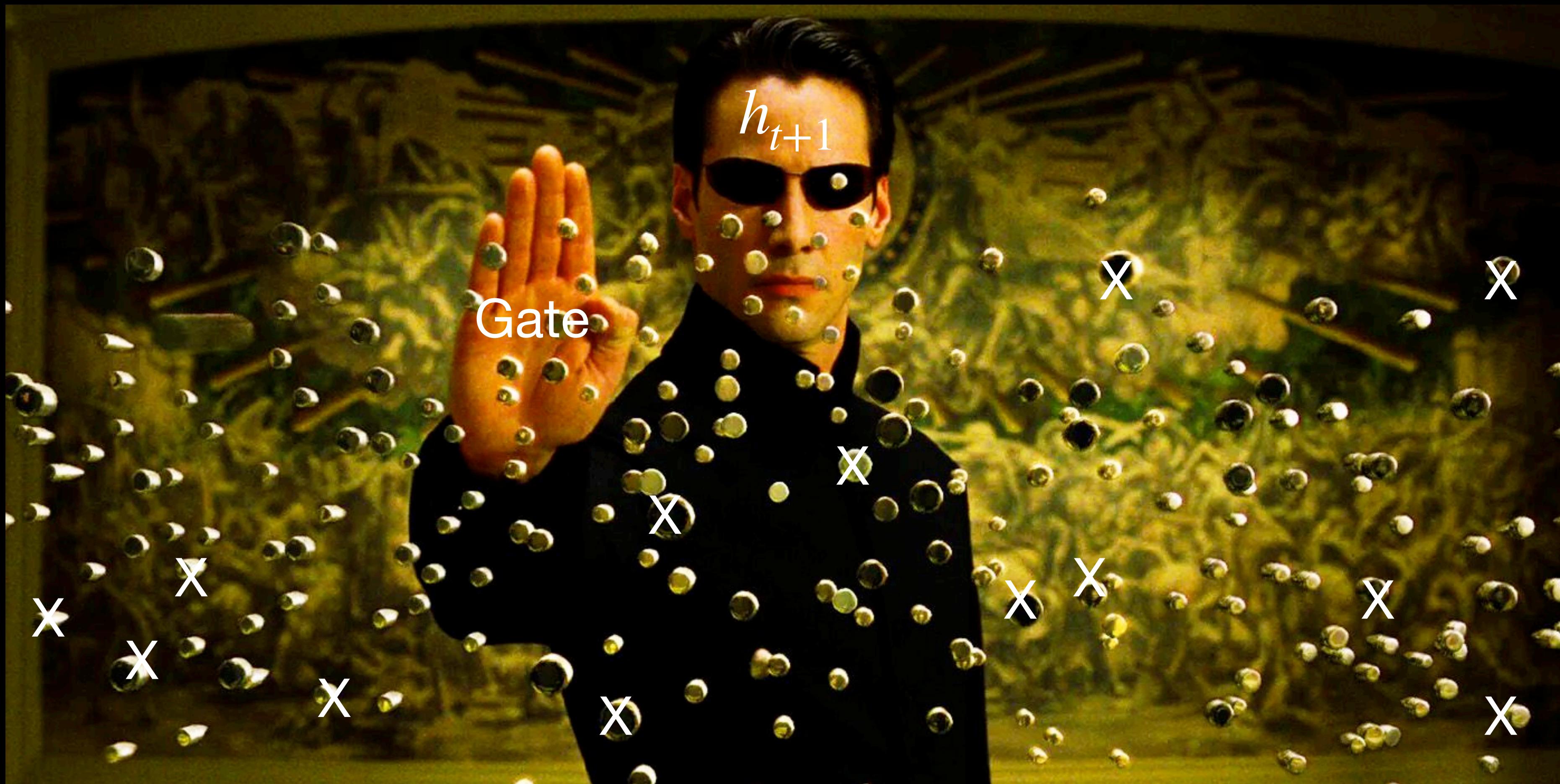
Different weights

$$\left\{ \begin{array}{l} \Gamma = \sigma(\textcolor{red}{W}_\Gamma[X_t, h_{t-1}] + \textcolor{red}{b}_\Gamma) \\ \tilde{h}_t = \tanh(\textcolor{yellow}{W}_H[X_t, h_{t-1}] + \textcolor{yellow}{b}_H) \\ h_t = \Gamma \otimes h_{t-1} + (1 - \Gamma) \otimes \tilde{h}_t \end{array} \right.$$

GRU

$$\begin{aligned}\Gamma &= \sigma(W_\Gamma[X_t, h_{t-1}] + b_\Gamma) \\ \tilde{h}_t &= \tanh(W_H[X_t, h_{t-1}] + b_H) \\ h_t &= \Gamma \otimes h_{t-1} + (1 - \Gamma) \otimes \tilde{h}_t\end{aligned}$$

The gate Γ controls, based on input and context, how much remembered.



GRU - full

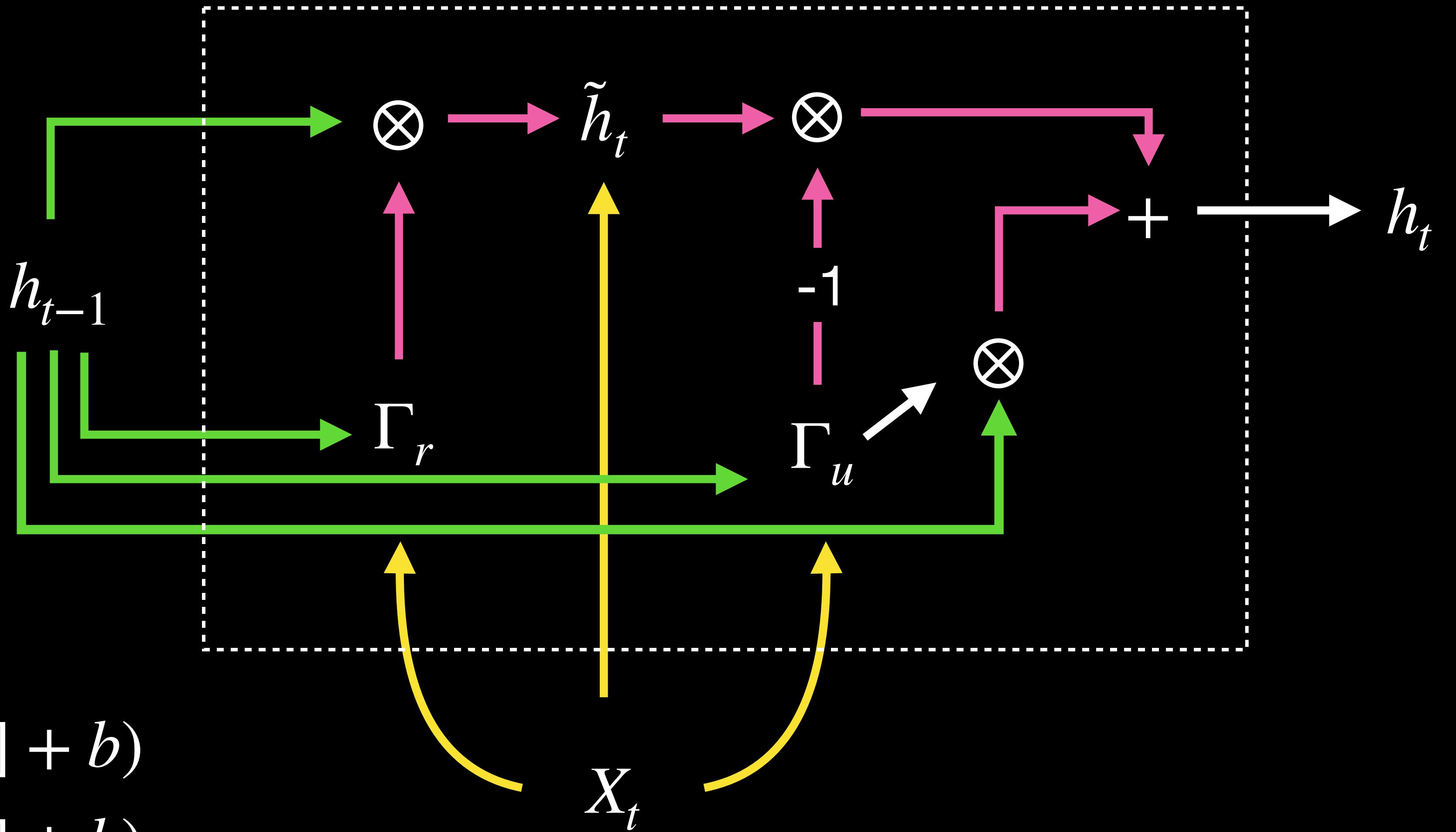
The full GRU has two gates, but the principle is the same

$$\Gamma_u = \sigma(W[X_t, h_{t-1}] + b)$$

$$\Gamma_r = \sigma(W[X_t, h_{t-1}] + b)$$

$$\tilde{h}_t = \tanh(W[X_t, \Gamma_r \otimes h_{t-1}] + b)$$

$$h_t = \Gamma_u \otimes h_{t-1} + (1 - \Gamma_u) \otimes \tilde{h}_t$$

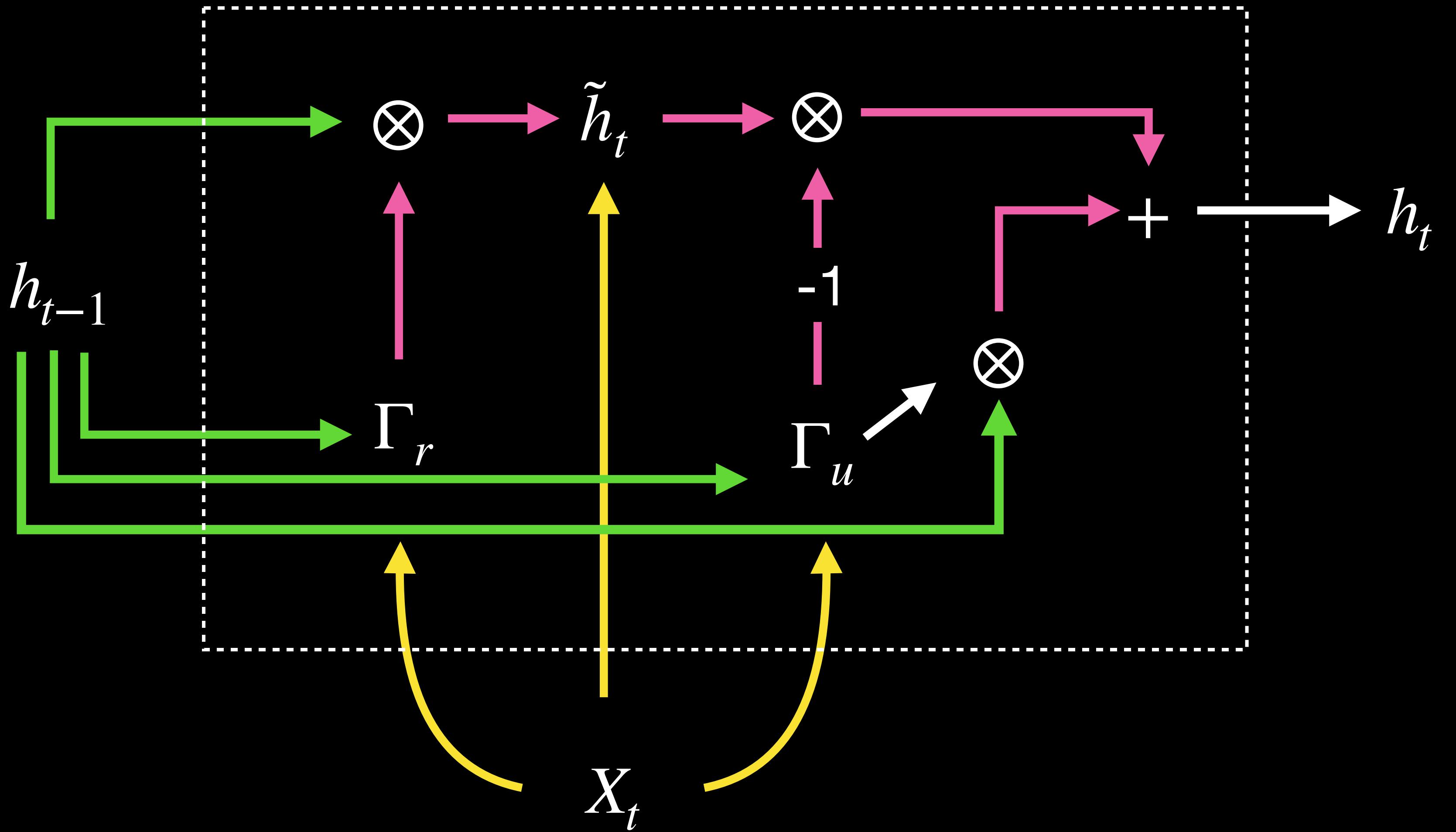


$$\Gamma_u = \sigma(W[X_t, h_{t-1}] + b)$$

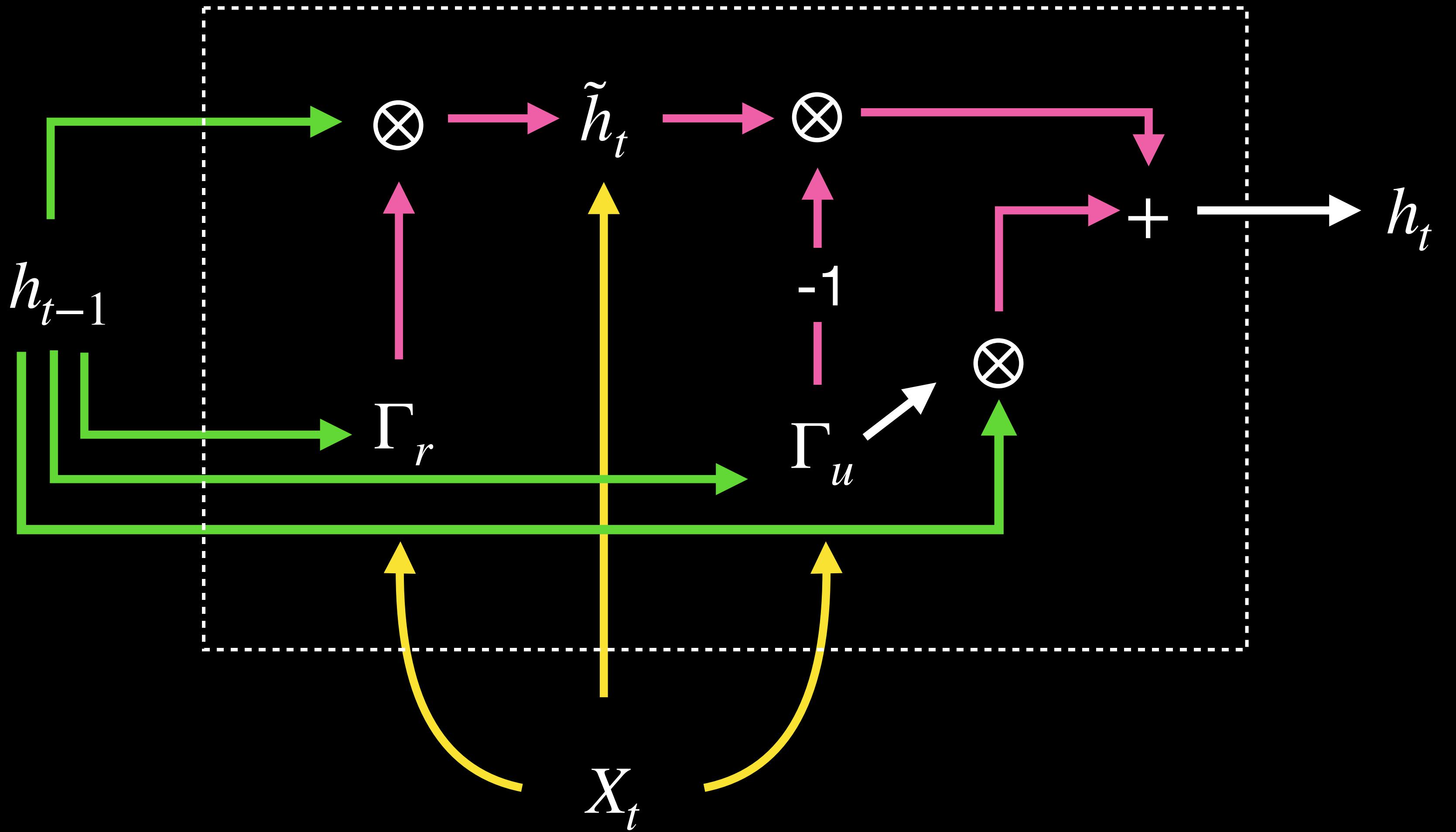
$$\Gamma_r = \sigma(W[X_t, h_{t-1}] + b)$$

$$\tilde{h}_t = \tanh(W[X_t, \Gamma_r \otimes h_{t-1}] + b)$$

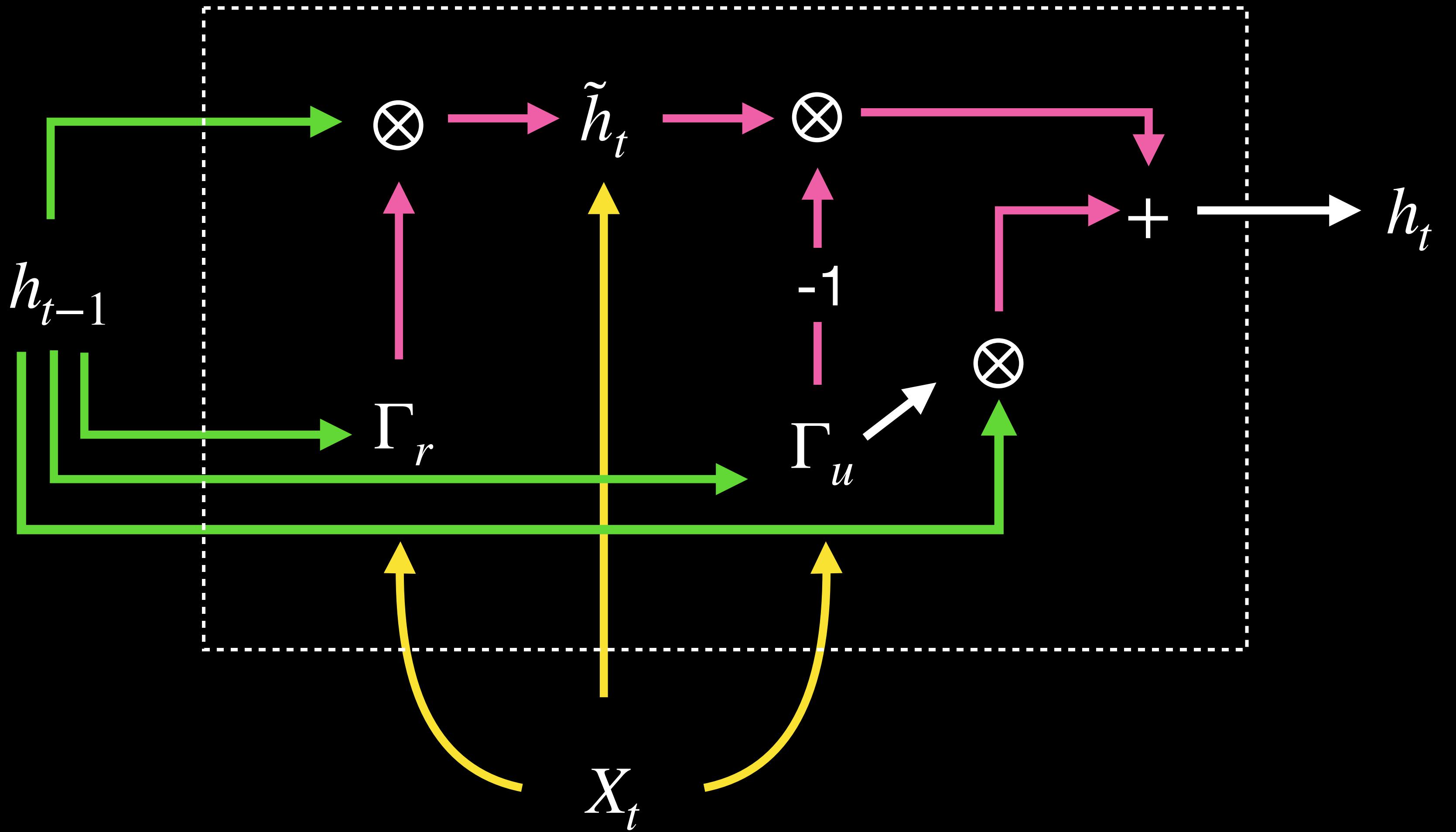
$$h_t = \Gamma_u \otimes h_{t-1} + (1 - \Gamma_u) \otimes \tilde{h}_t$$



We use the hidden state h_{t-1} and X_t to create two gates.



The reset gate Γ_r controls how much of the past h_{t-1} is mixed into X_t to create a new candidate context \tilde{h}



The other gate is the update gate Γ_u
and this balances the old h_{t-1} and
the new \tilde{h}_t

GRU

Compare the [Trax implementation](#) with the formulas

$$\Gamma_u = \sigma(W[X_t, h_{t-1}] + b)$$

$$\Gamma_r = \sigma(W[X_t, h_{t-1}] + b)$$

$$\tilde{h}_t = \tanh(W[X_t, \Gamma_r \otimes h_{t-1}] + b)$$

$$h_t = \Gamma_u \otimes h_{t-1} + (1 - \Gamma_u) \otimes \tilde{h}_t$$

```
def forward(self, inputs):
    x, gru_state = inputs

    # Dense layer on the concatenation of x and h.
    w1, b1, w2, b2 = self.weights
    y = jnp.dot(jnp.concatenate([x, gru_state], axis=-1), w1) + b1

    # Update and reset gates.
    u, r = jnp.split(fastmath.sigmoid(y), 2, axis=-1)

    # Candidate.
    c = jnp.dot(jnp.concatenate([x, r * gru_state], axis=-1), w2) + b2

    new_gru_state = u * gru_state + (1 - u) * jnp.tanh(c)
    return new_gru_state, new_gru_state
```

LSTM

The LSTM has

- three gates (update, input and forget) instead of two (update and reset)
- Has both a context C and a hidden state h

$$\Gamma_u = \sigma(W[X_t, h_{t-1}] + b)$$

$$\Gamma_i = \sigma(W[X_t, h_{t-1}] + b)$$

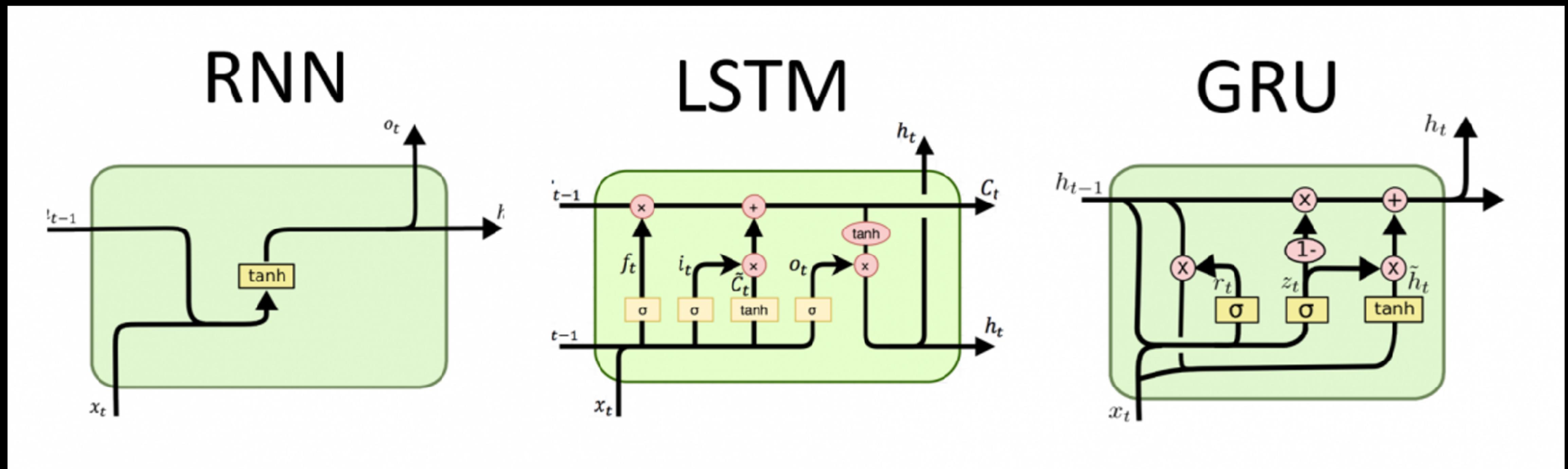
$$\Gamma_f = \sigma(W[X_t, h_{t-1}] + b)$$

$$\tilde{h} = \Gamma_i \otimes \tanh(W[X_t, h_{t-1}] + b)$$

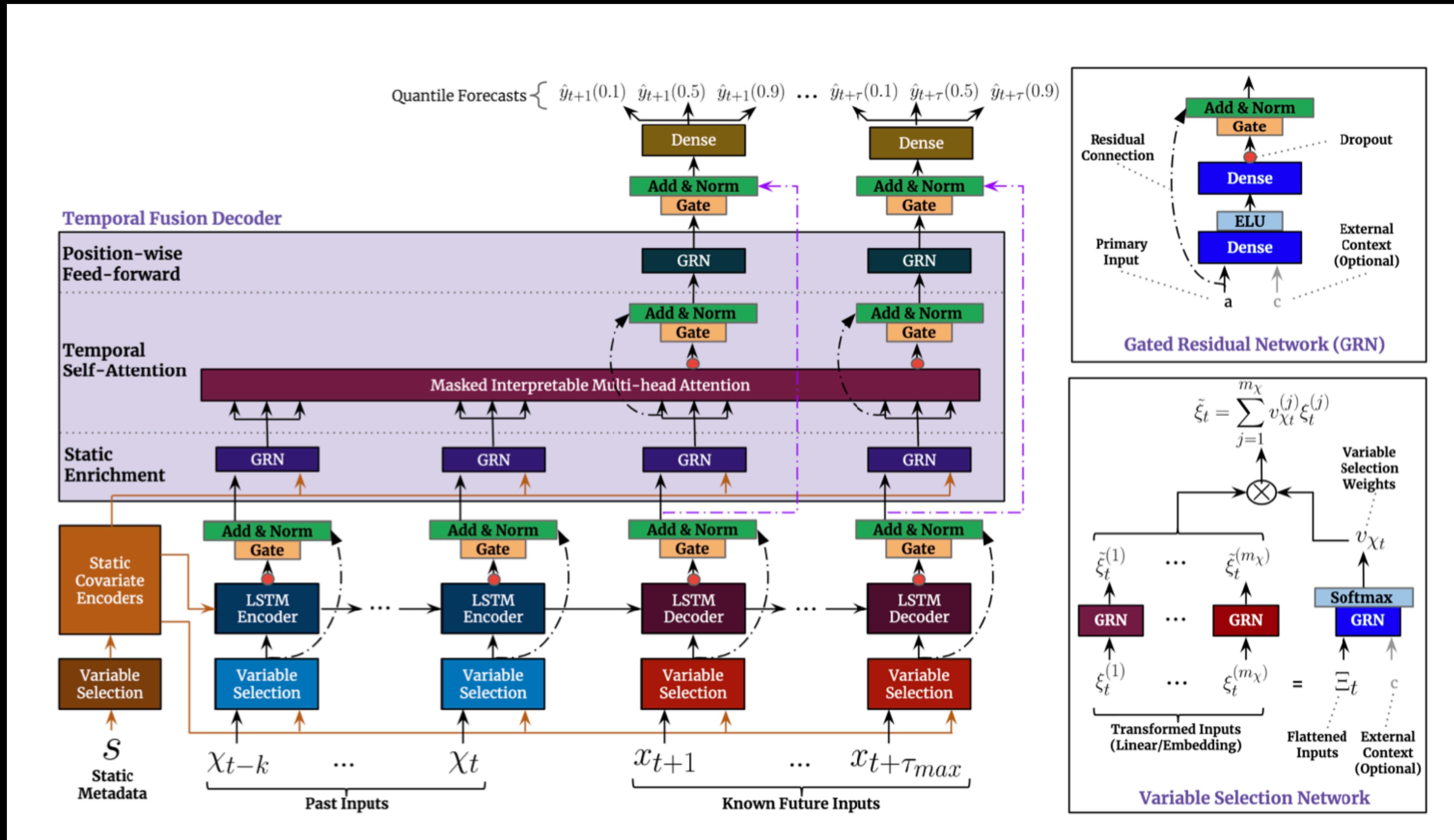
$$\tilde{C} = \tanh(\Gamma_f \otimes C + \tilde{h})$$

$$h_t = \Gamma_u \otimes \tilde{C}$$

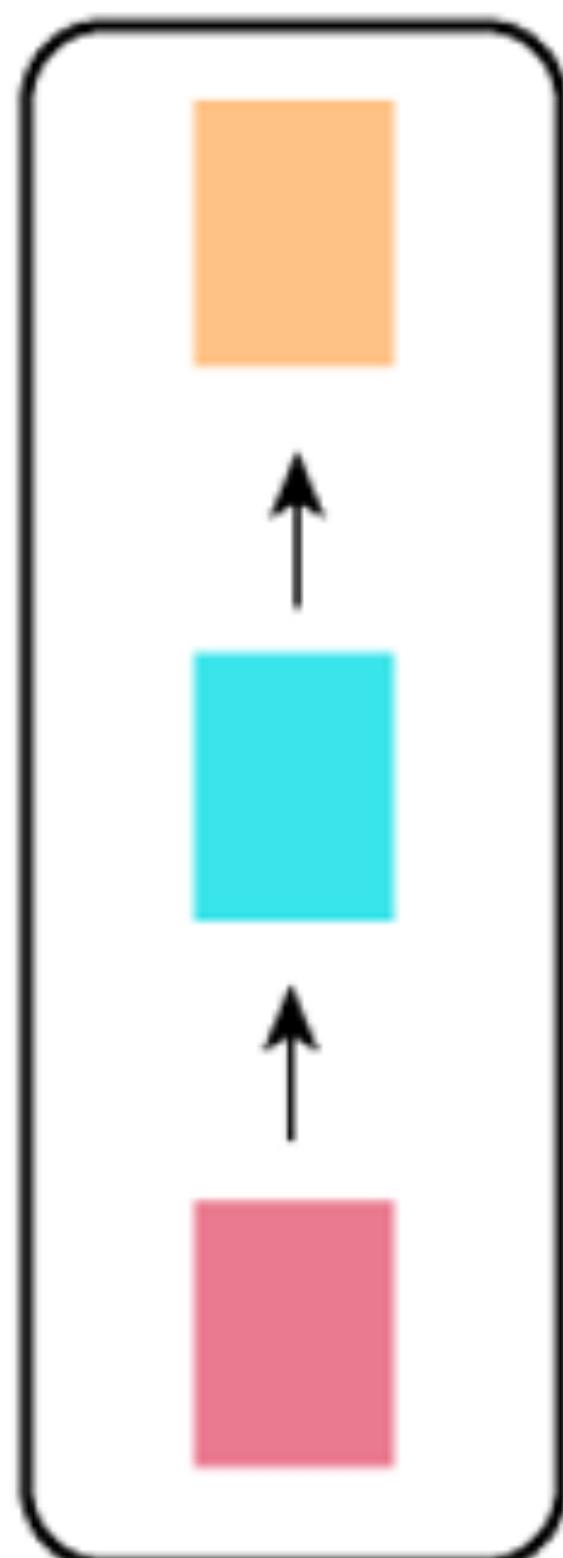
Overview



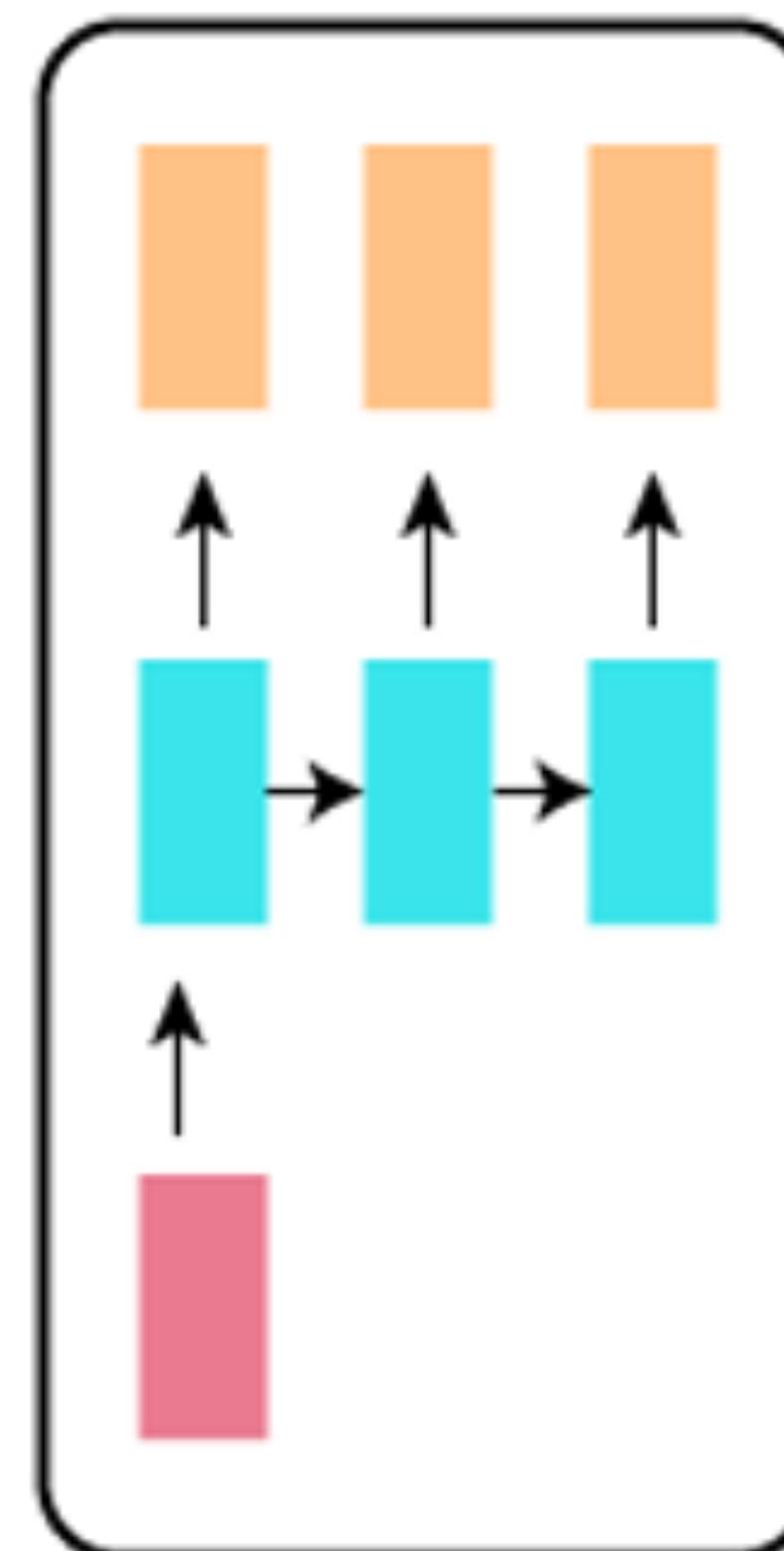
Temporal Fusion Transformer, Lim et al. (2021)



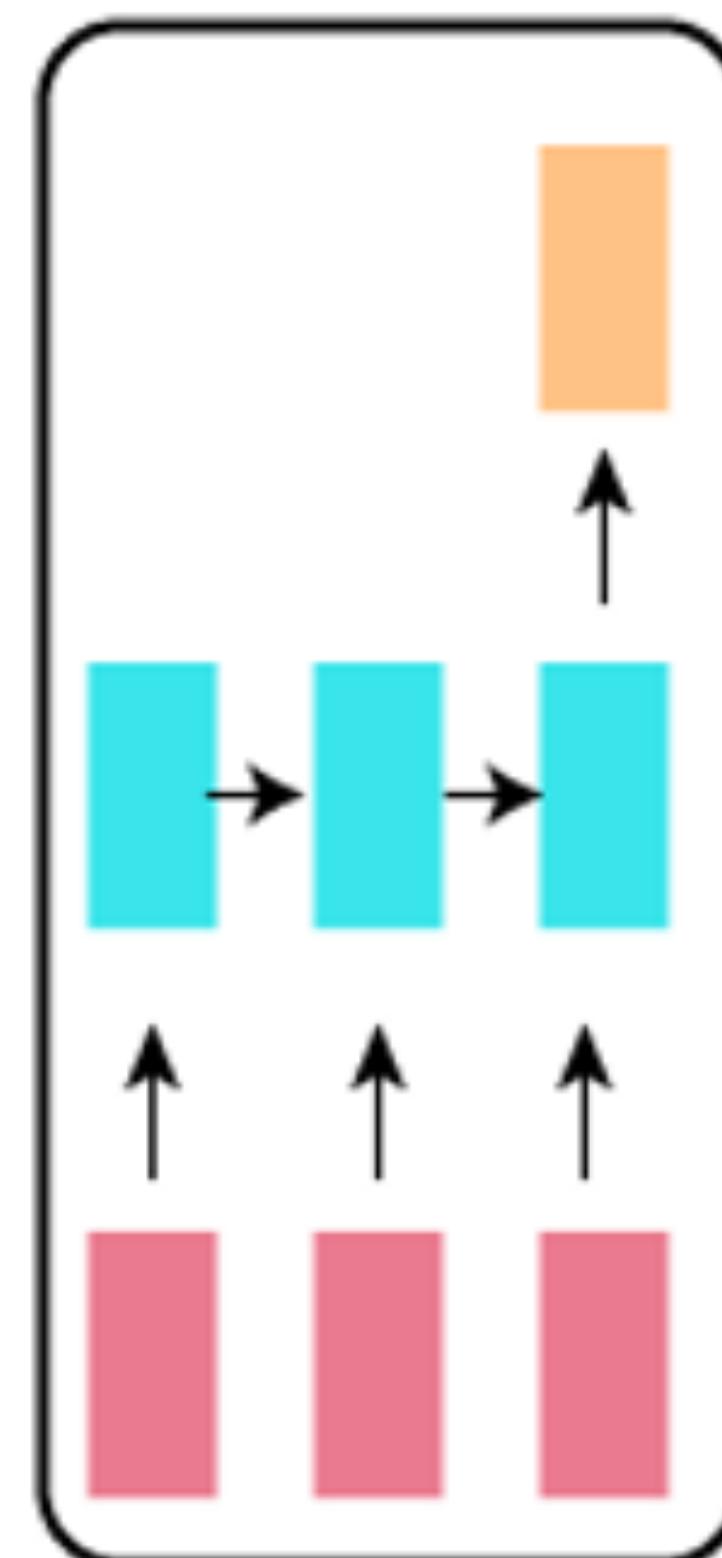
one to one



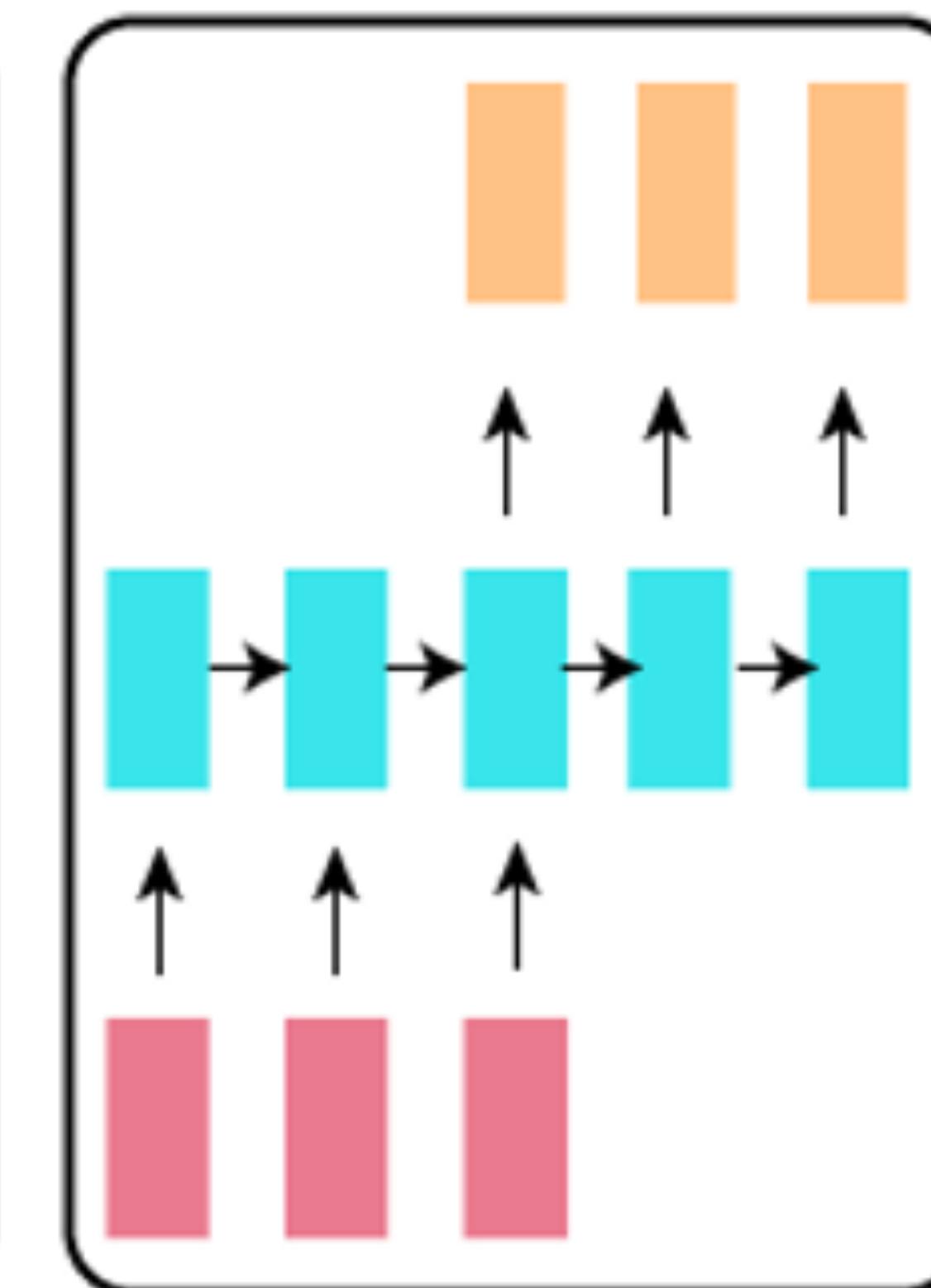
one to many



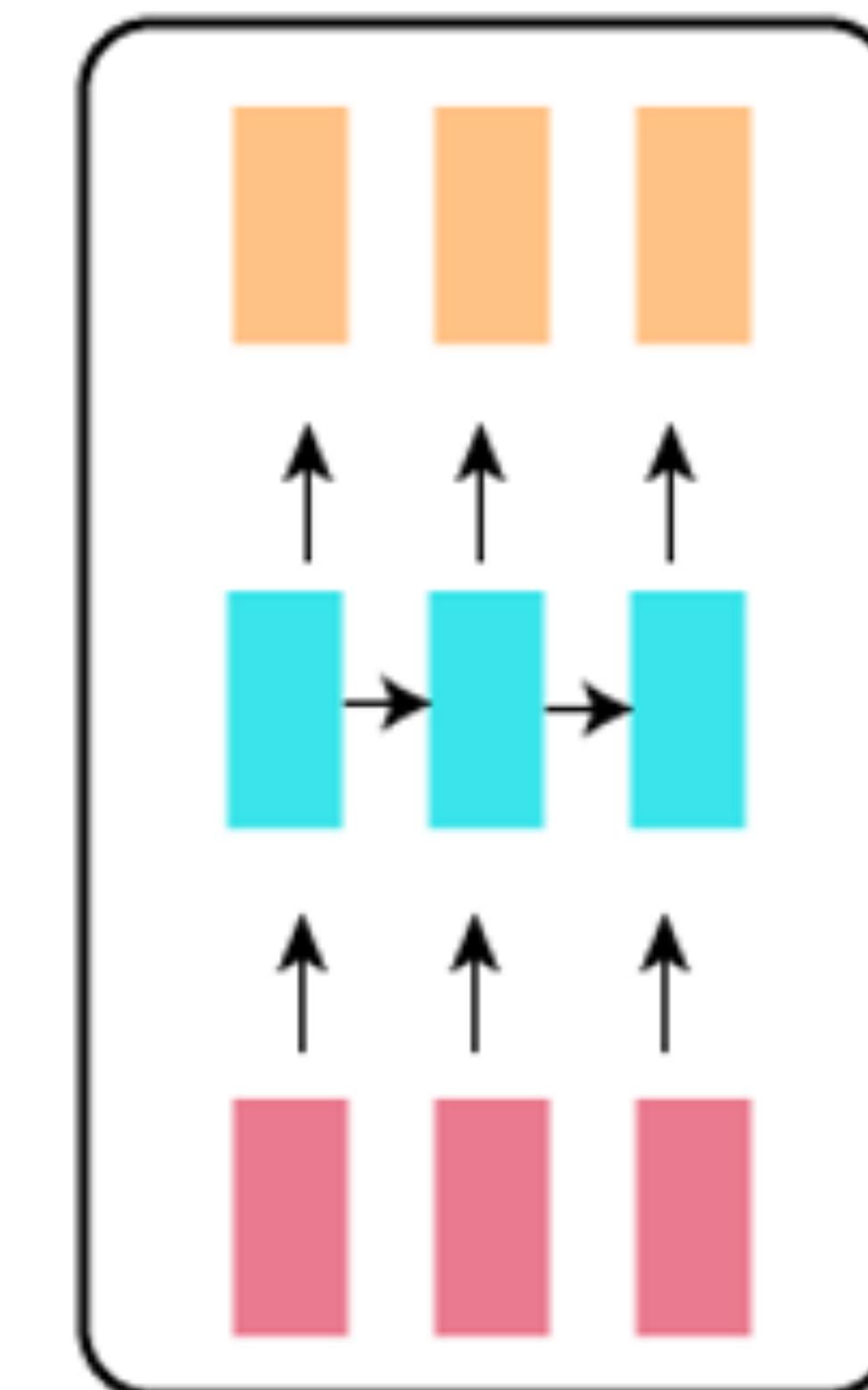
many to one



many to many



many to many



RNN architectures

- One to one: Stock price prediction (current price → next price)
- One to many: Music generation (single note/seed → melody sequence)
- Many to one: Sentiment analysis (sentence → positive/negative)
- Many to many (different lengths): Machine translation (English sentence → French sentence)
- Many to many (same length): Named entity recognition (word sequence → entity tag sequence)

Summary

- The Simple RNN is the most basic, but does not have good ways to control memory
- LSTM has more parameters with three gates and two hidden states, and thus more complexity
- GRU is a simplified version of the LSTM with two gates and one hidden state.

There is no “best” Recurrent Neural Network, this depends on your usecase.