Recurrent Neural Networks

deep learning 3

Raoul Grouls, 21 Mei 2024

Motivation

A lot of data is sequential, varying over time:

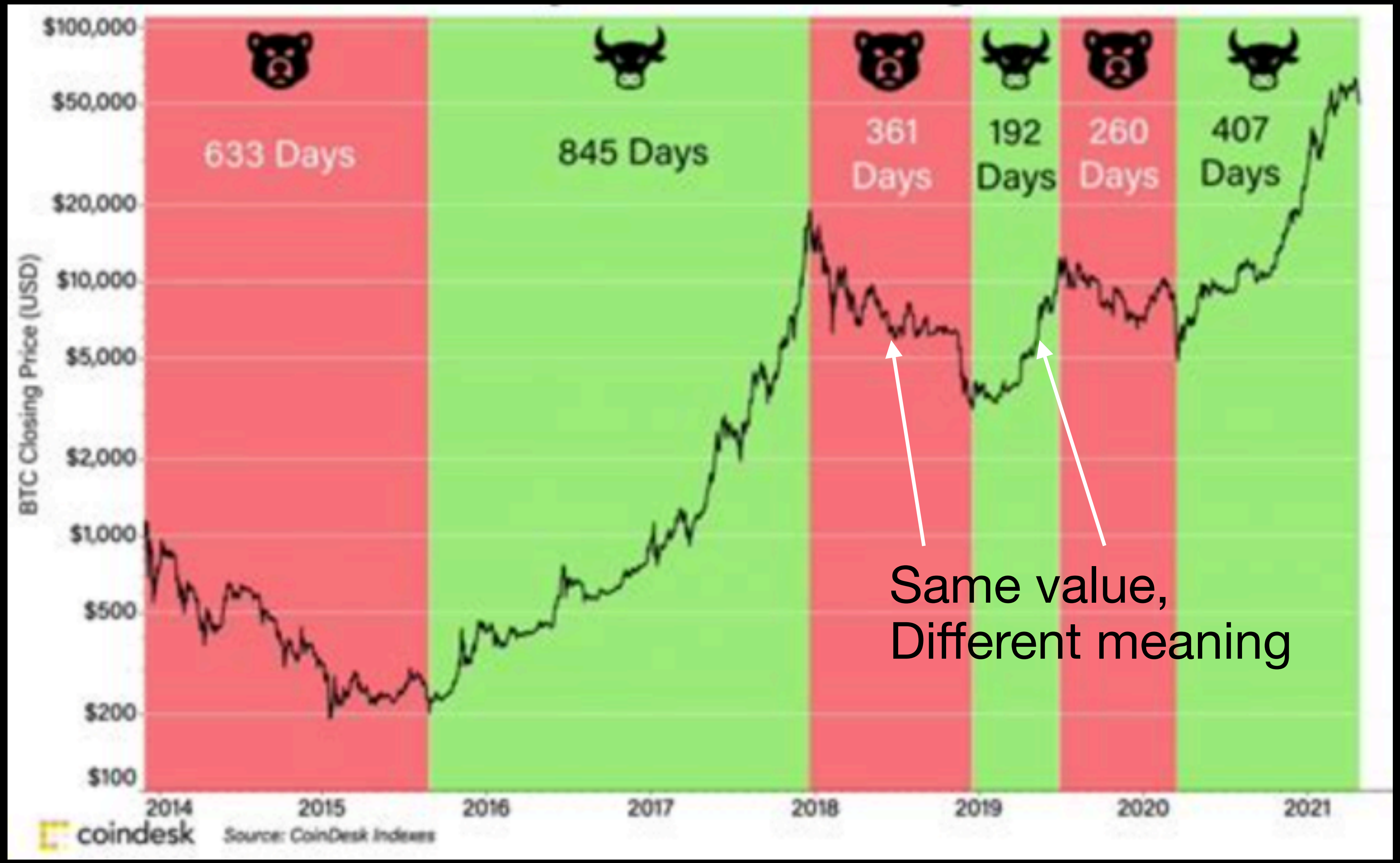
- Sentences
- Music
- EEG
- Movement
- Markets

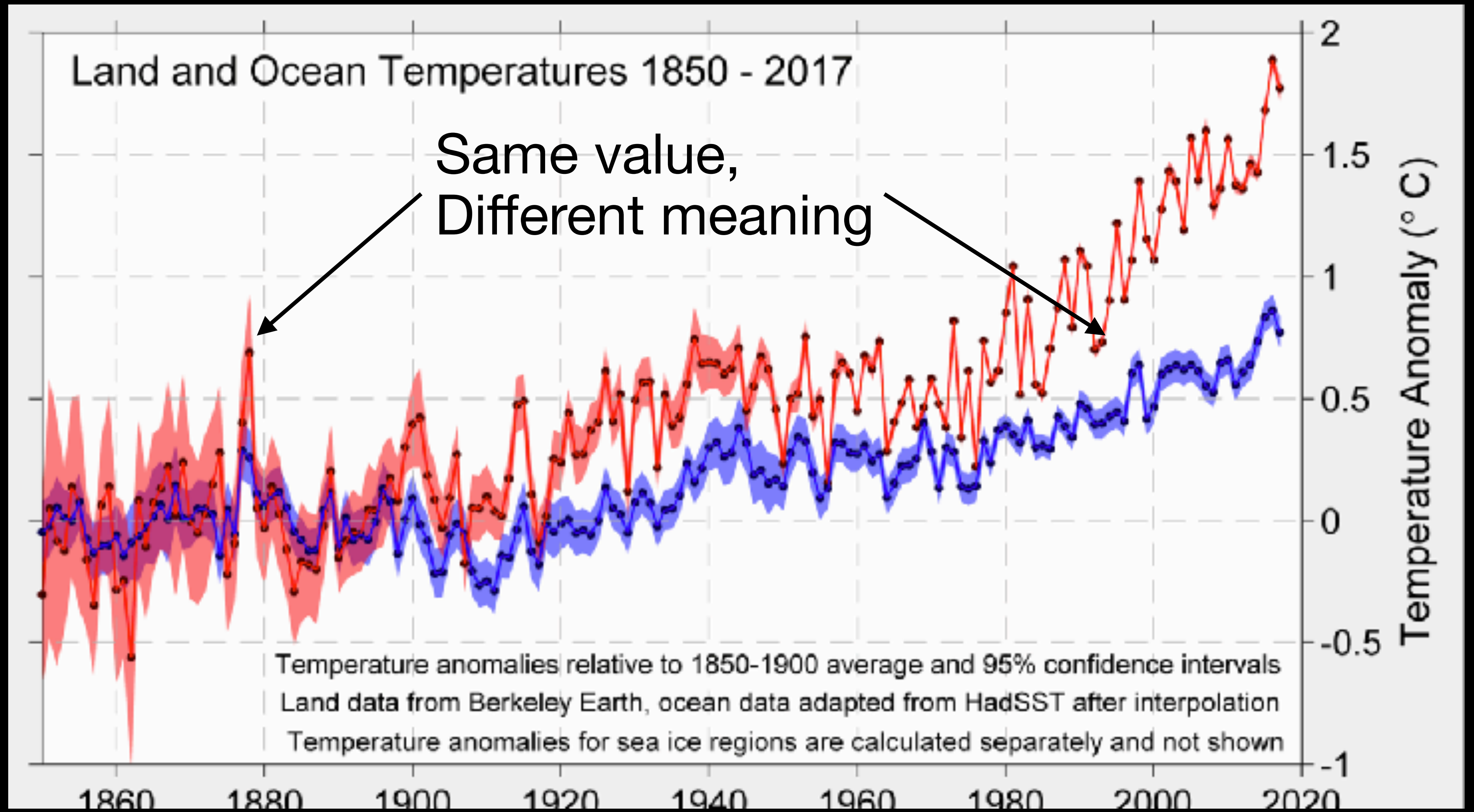
Motivation

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

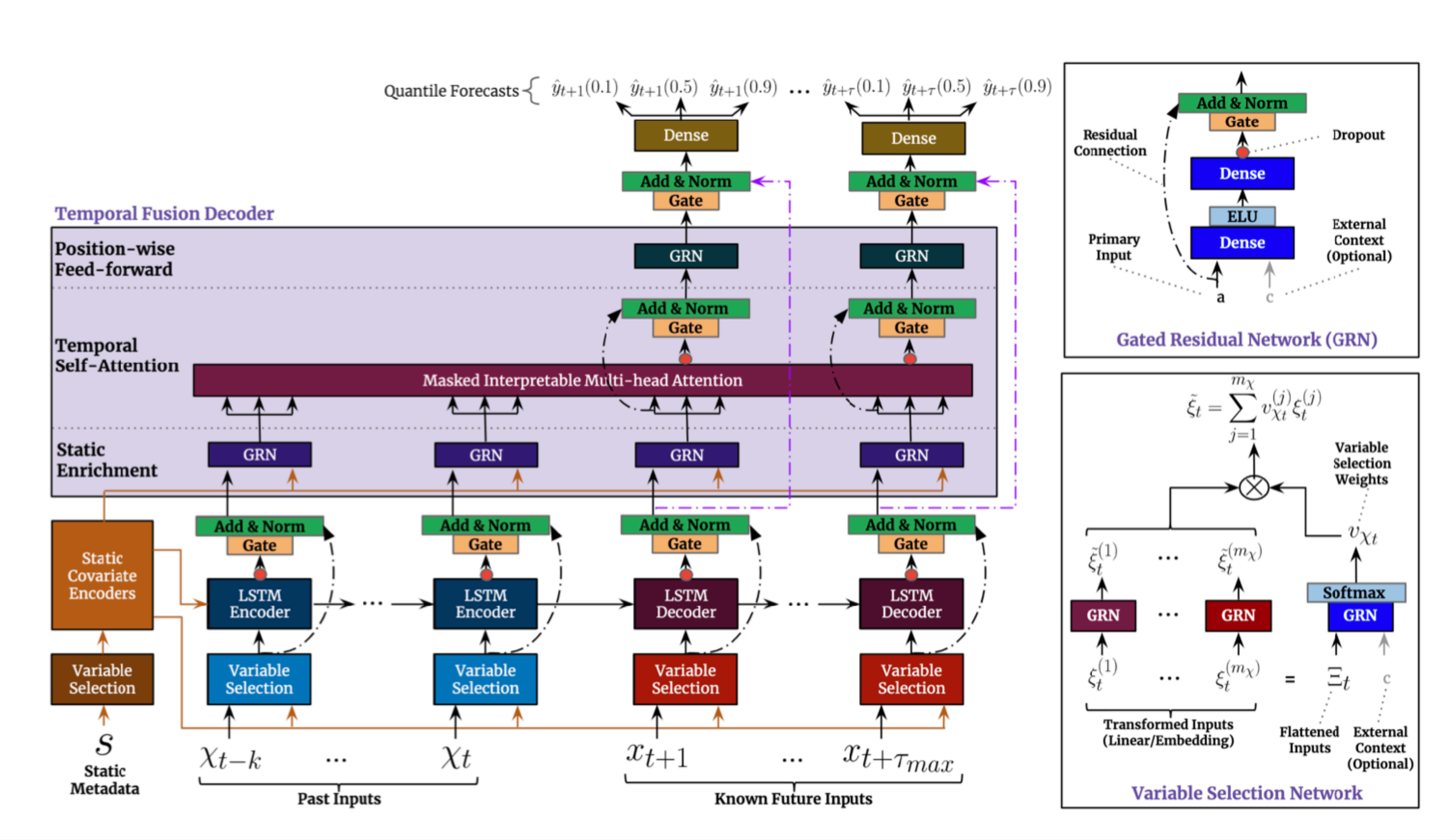
2009 LSTM won the handwriting recognition competition

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)



Simple RNN

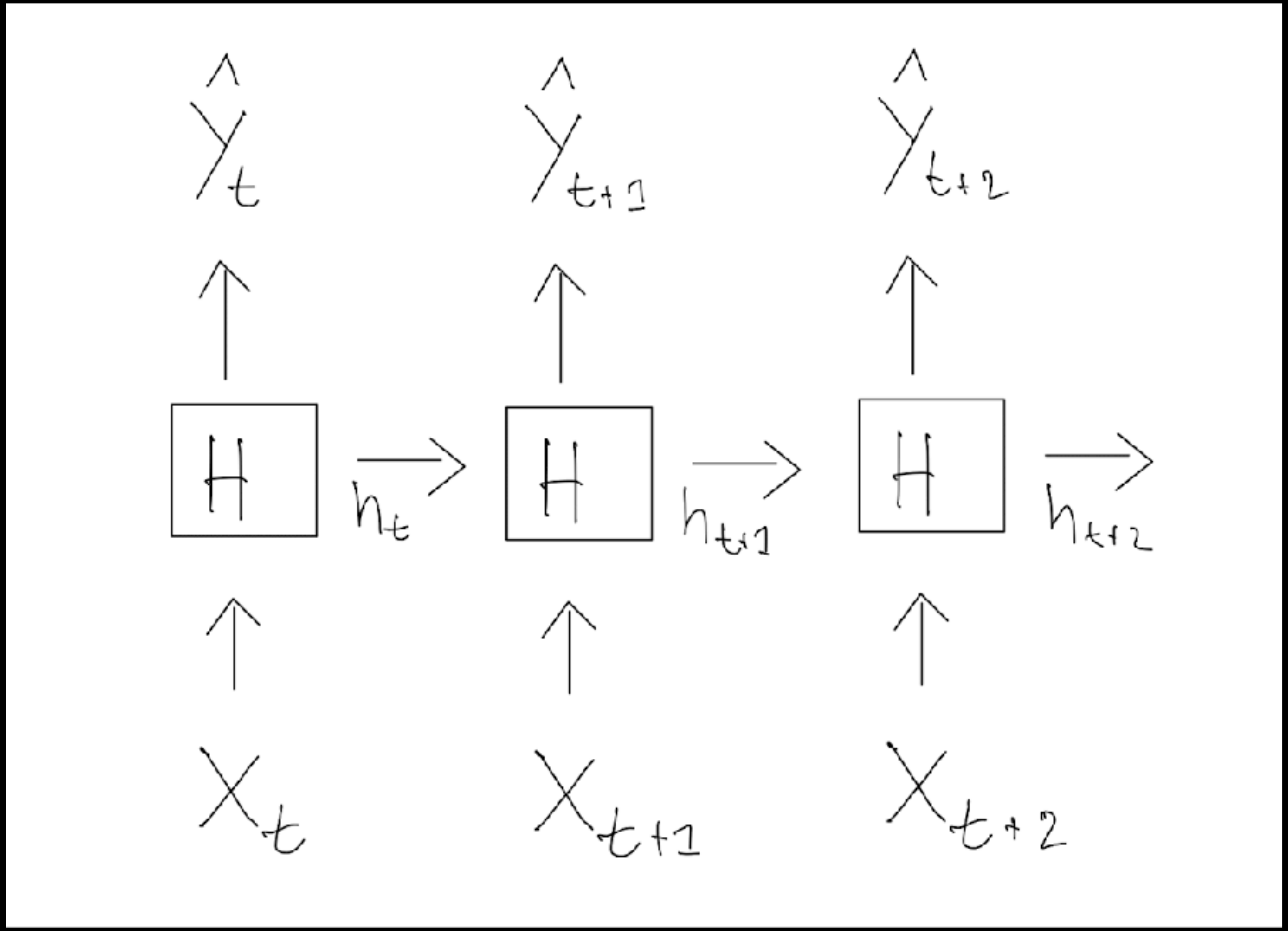
We start with a simple neural network H

To add time, we introduce the concept of a hidden state h_t that we pass on.

While this might look confusing at first, there is just a small difference with the

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

formula we have been using so far.



Simple RNN

To incorporate the hidden state, we simply add it:

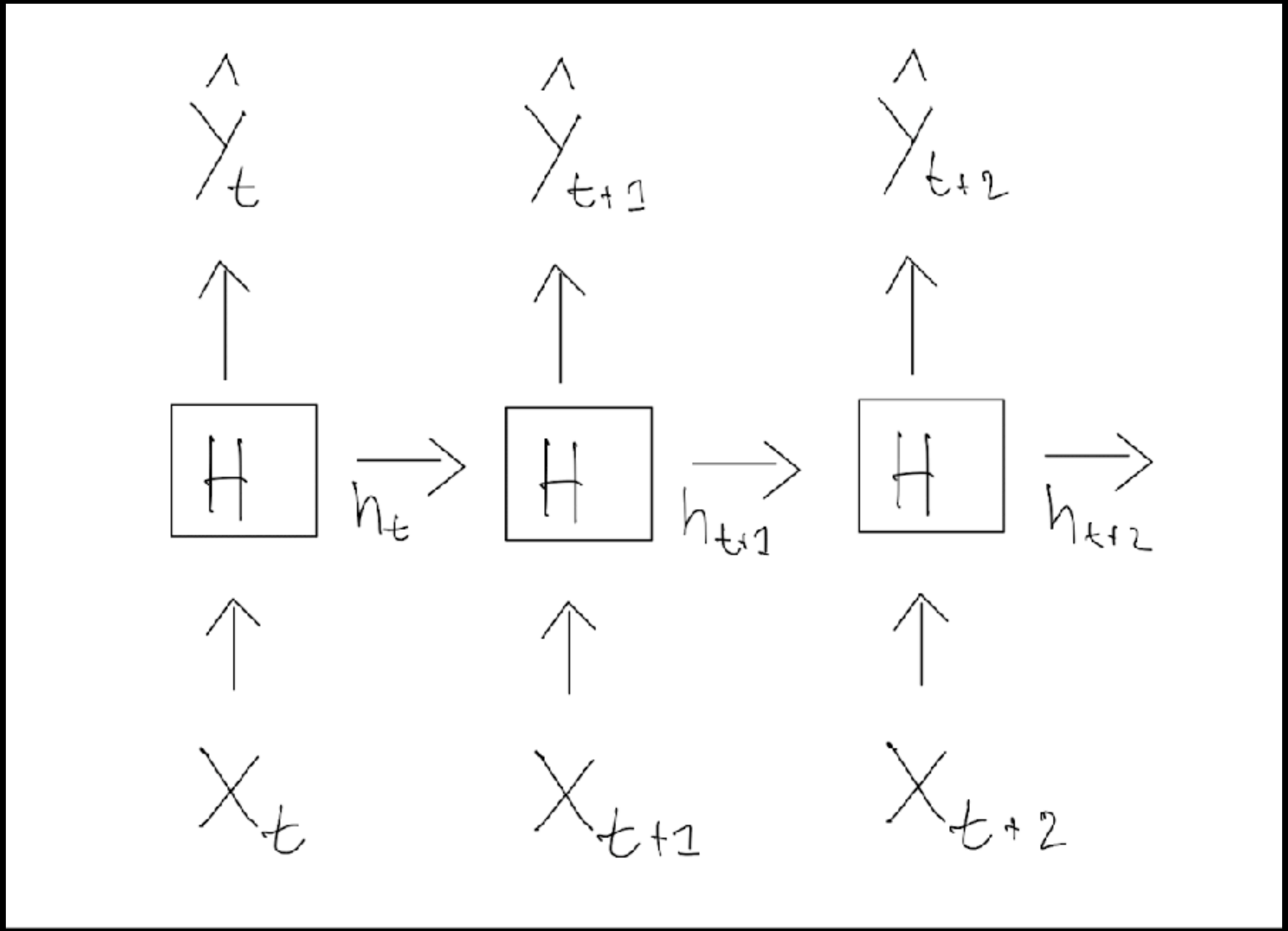
$$h_t = \sigma(W_x X_t + W_h h_{t-1} + b)$$

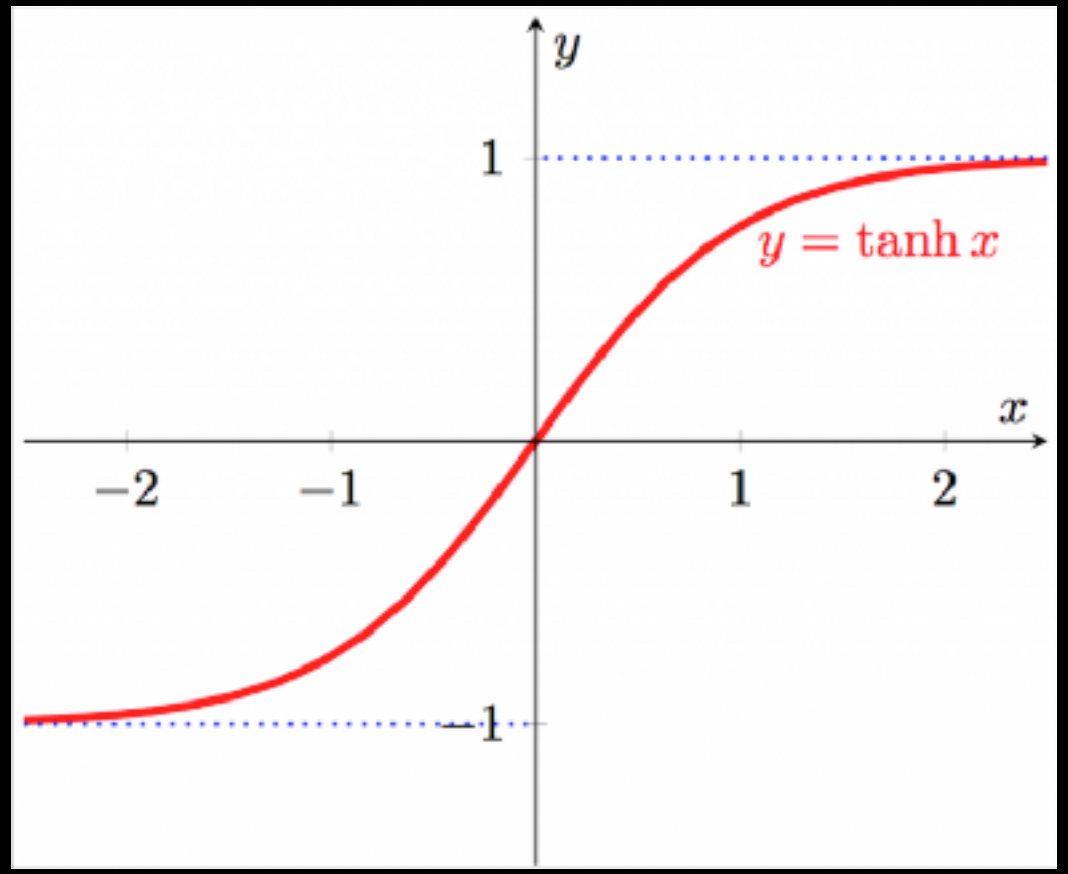
This is equivalent to

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

where $[X, h]$ means concatenate or stack

σ is an activation function, typically $tanh$





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.

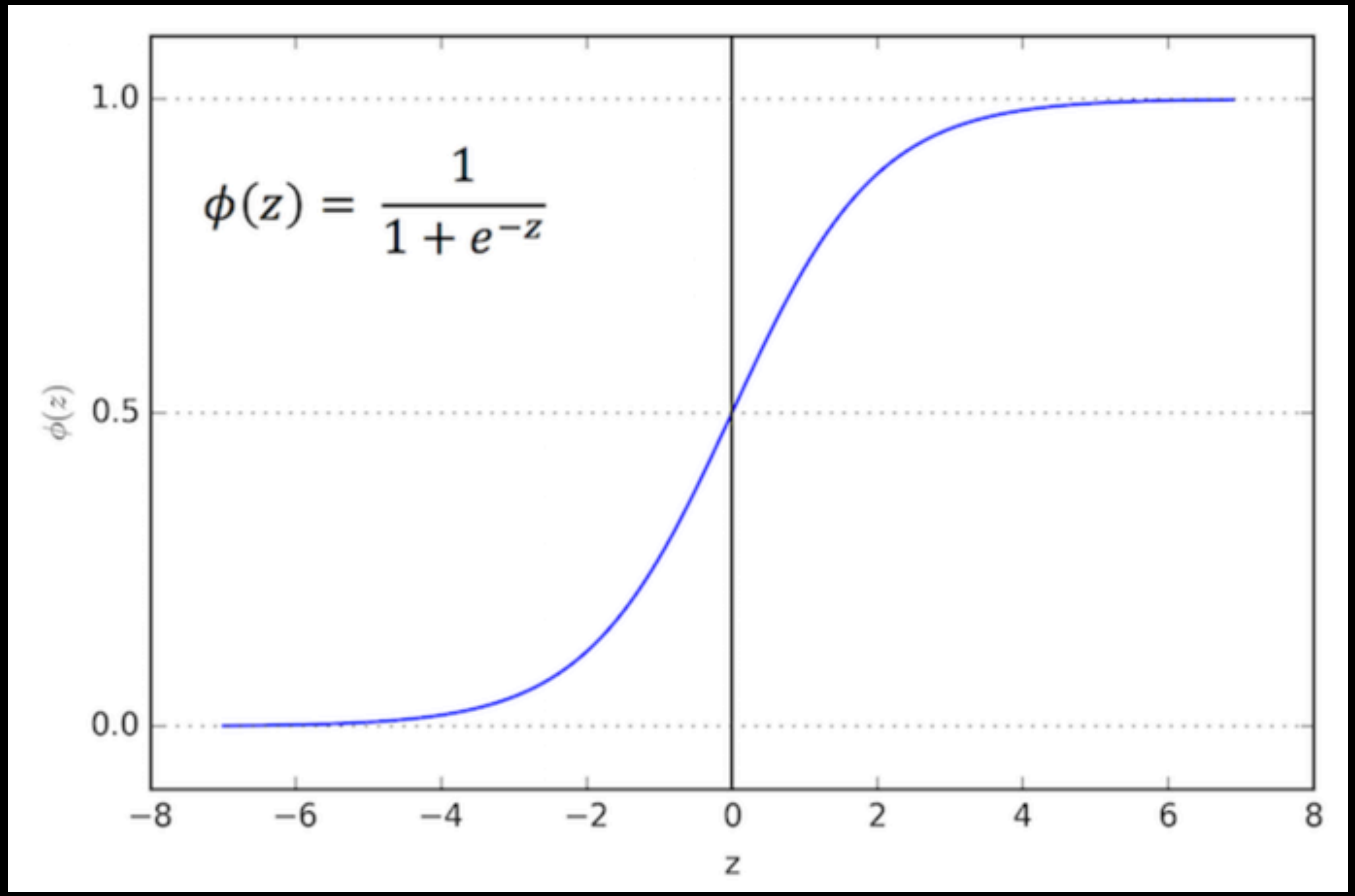
GRU - Gates

To create a gate, we will use a sigmoid activation and pick a W such that Γ has the same dimensions as X:

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

This gives us numbers of the same shape as the input, between $[0,1]$

To apply the gate, we will use what is called a Hadamard product $\otimes$



$$\underset{X}{\begin{bmatrix} 1.0 & 2.0 \\ 0.5 & -2.4 \end{bmatrix}} \otimes \underset{\Gamma}{\begin{bmatrix} 0.9 & 0.02 \\ 0.5 & 0.2 \end{bmatrix}} = \underset{\text{output}}{\begin{bmatrix} 0.9 & 0.04 \\ 0.25 & -0.48 \end{bmatrix}}$$

GRU - simplified version

Concatenate state, create gate, hadamard

The GRU creates

- a *candidate* state $\tilde{h}$
- a gate Γ

where the gate Γ decides, based on context, how much of the past is remembered.

The W and b in the formulas are different weights, but I left out the subscripts (eg W_1) to simplify the formula.

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

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

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

GRU - full

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

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

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

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

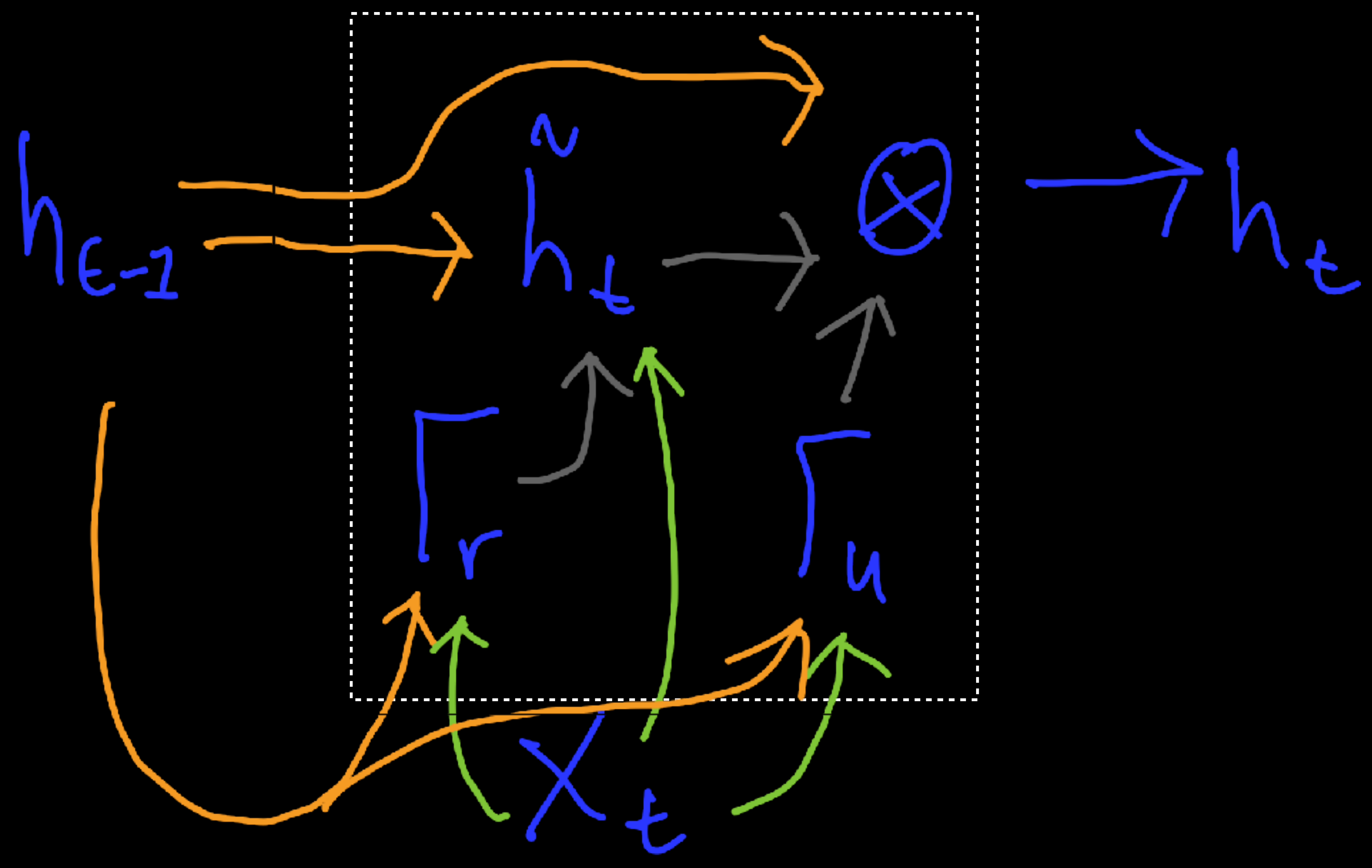
$$h_t = \Gamma_u \otimes h_{t-1} + (1 - \Gamma_u) \otimes \tilde{h}_t$$

GRU

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

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

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

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

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

```python
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

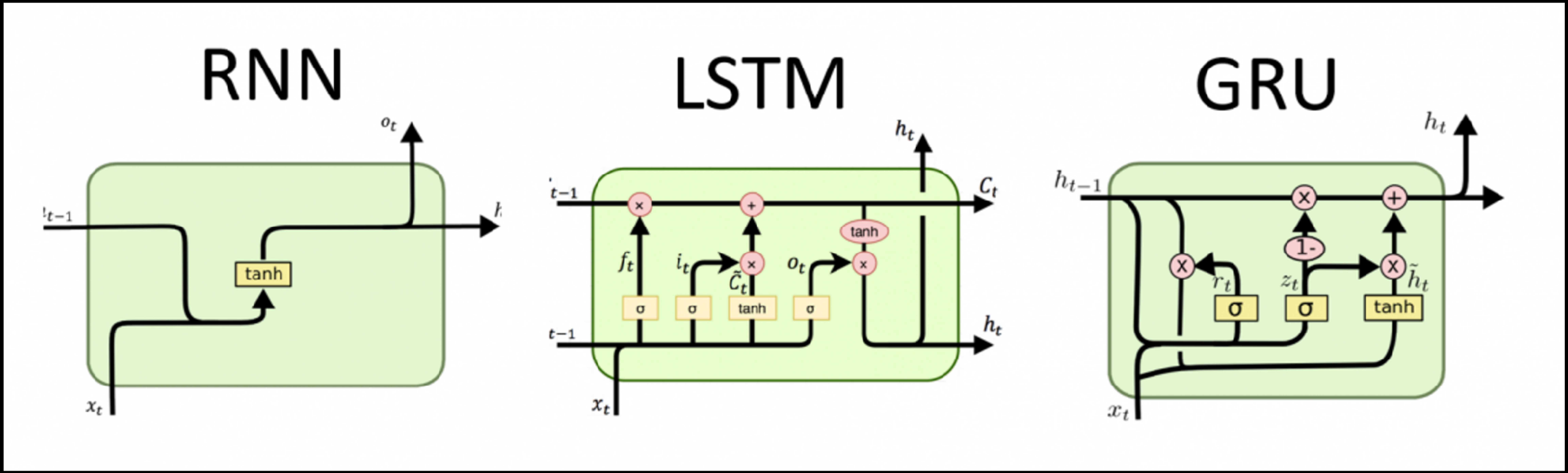
The LSTM has

- <u>three</u> gates (update, input and forget) instead of two (update and reset)

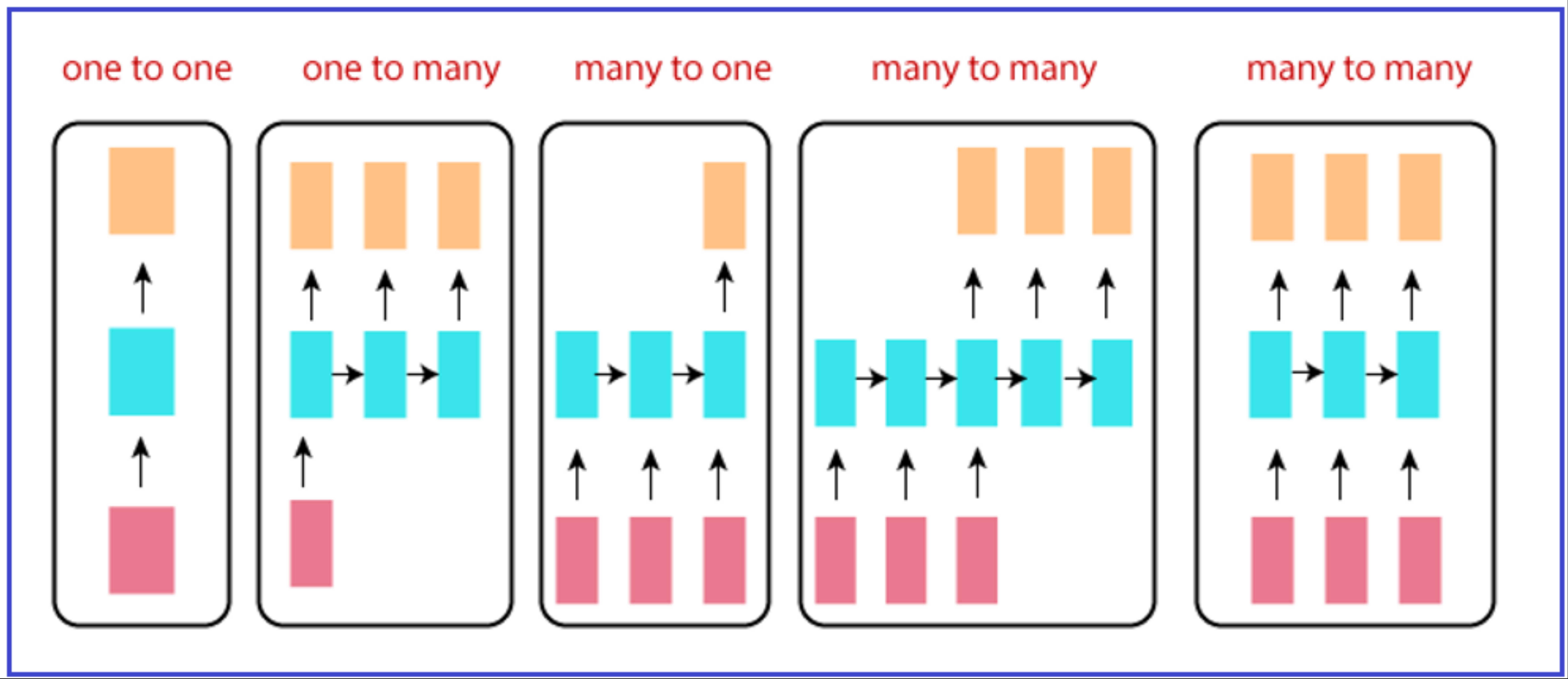
- Has both a context C and a hidden state h

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

Overview



RNN
o_t
h_{t-1}
tanh
x_t
LSTM
h_t
C_{t-1}
C_t
f_t
i_t
$\tilde{C}_t$
o_t
tanh
σ
σ
tanh
σ
h_{t-1}
h_t
x_t
GRU
h_t
h_{t-1}
r_t
z_t
1-
$\tilde{h}_t$
σ
σ
tanh
x_t


one to one
one to many
many to one
many to many
many to many

Overview

- The Simple RNN is the most basic, but does not has 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.