Recurrent Neural Networks

deep learning 3

Raoul Grouls, 13 Mei 2025

Neural networks

Neural networks

$$\sigma(wx + b)$$

Neural networks

$$\sigma(wx + b)$$

Input

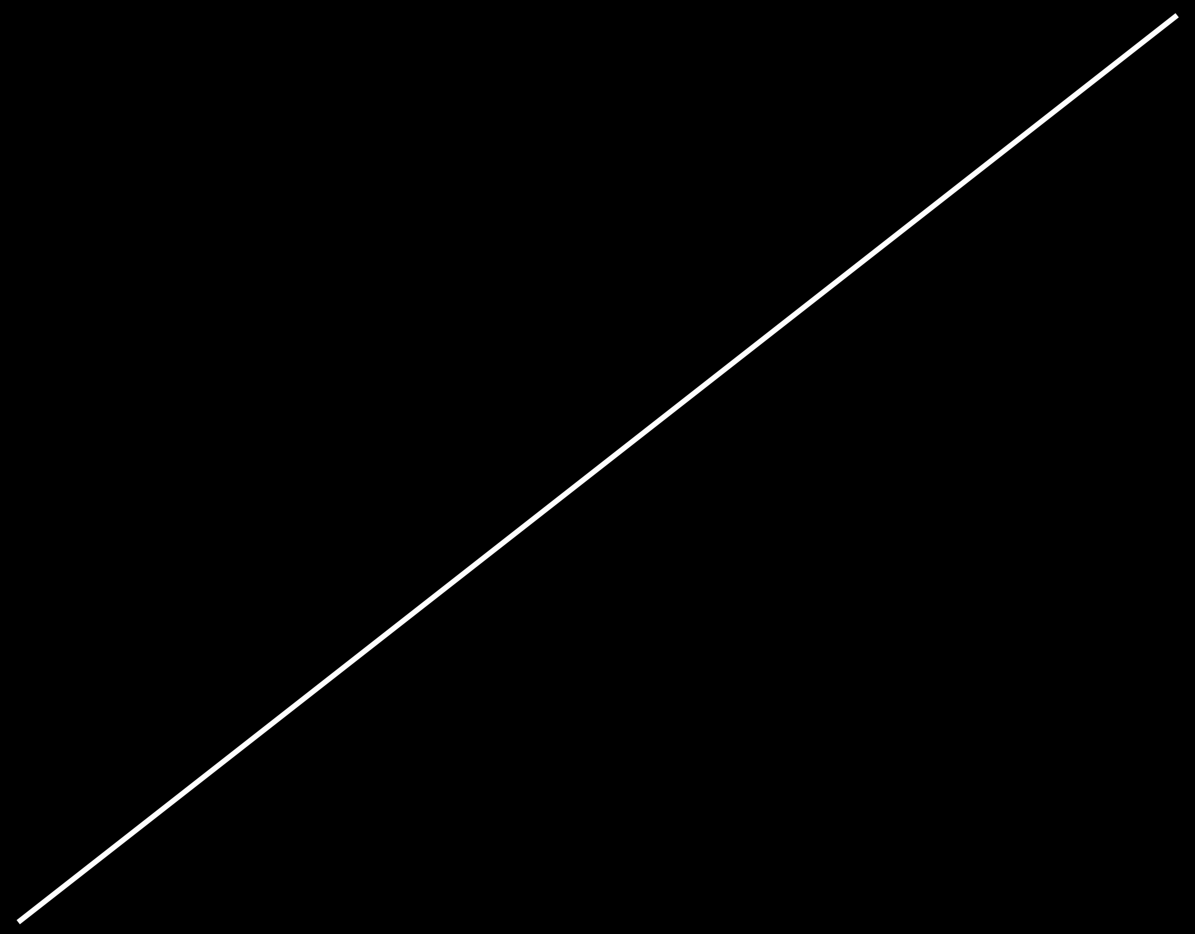
Neural networks

$$\sigma(wx + b)$$

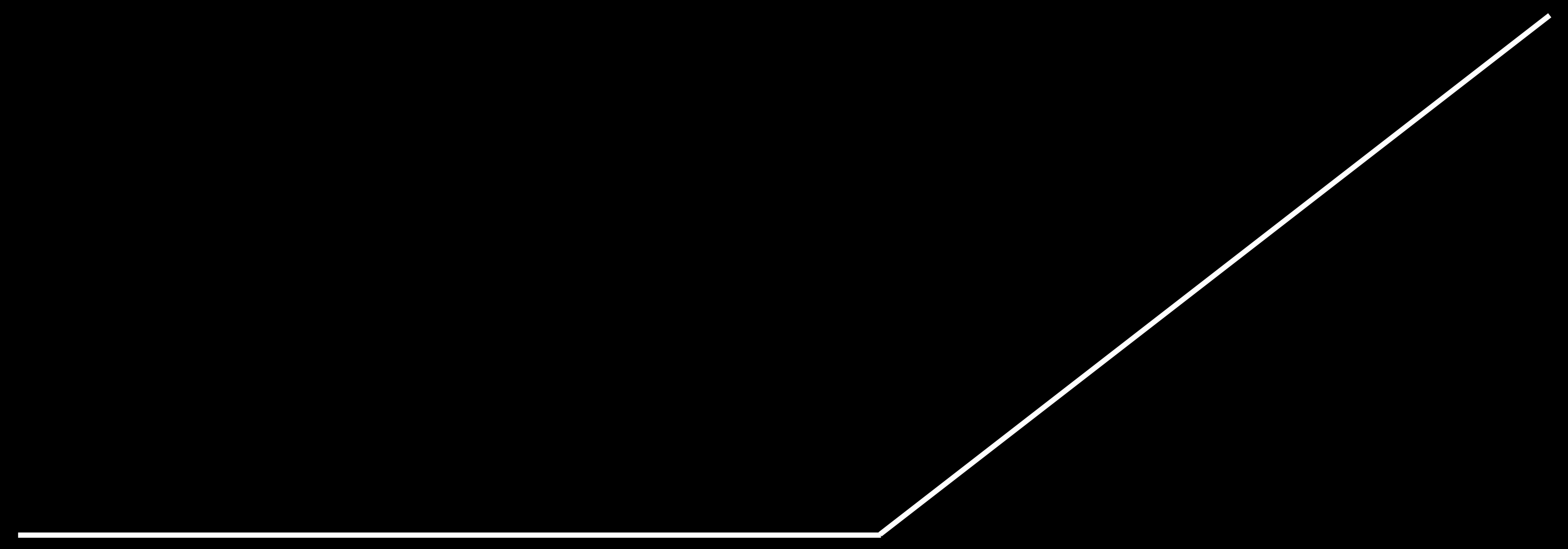
Non-linear transformation

Linear transformation

Neural networks



Linear



Nonlinear

Neural networks

$$\sigma(wx + b)$$

Learnable

Neural networks

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

Prediction

Neural networks

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

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

Change w and b such that z is minimal

Neural networks

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

$$\left.\begin{matrix} \dfrac{\partial z}{\partial w} & \dfrac{\partial z}{\partial b} \end{matrix}\right\} \text{Gradient}$$

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

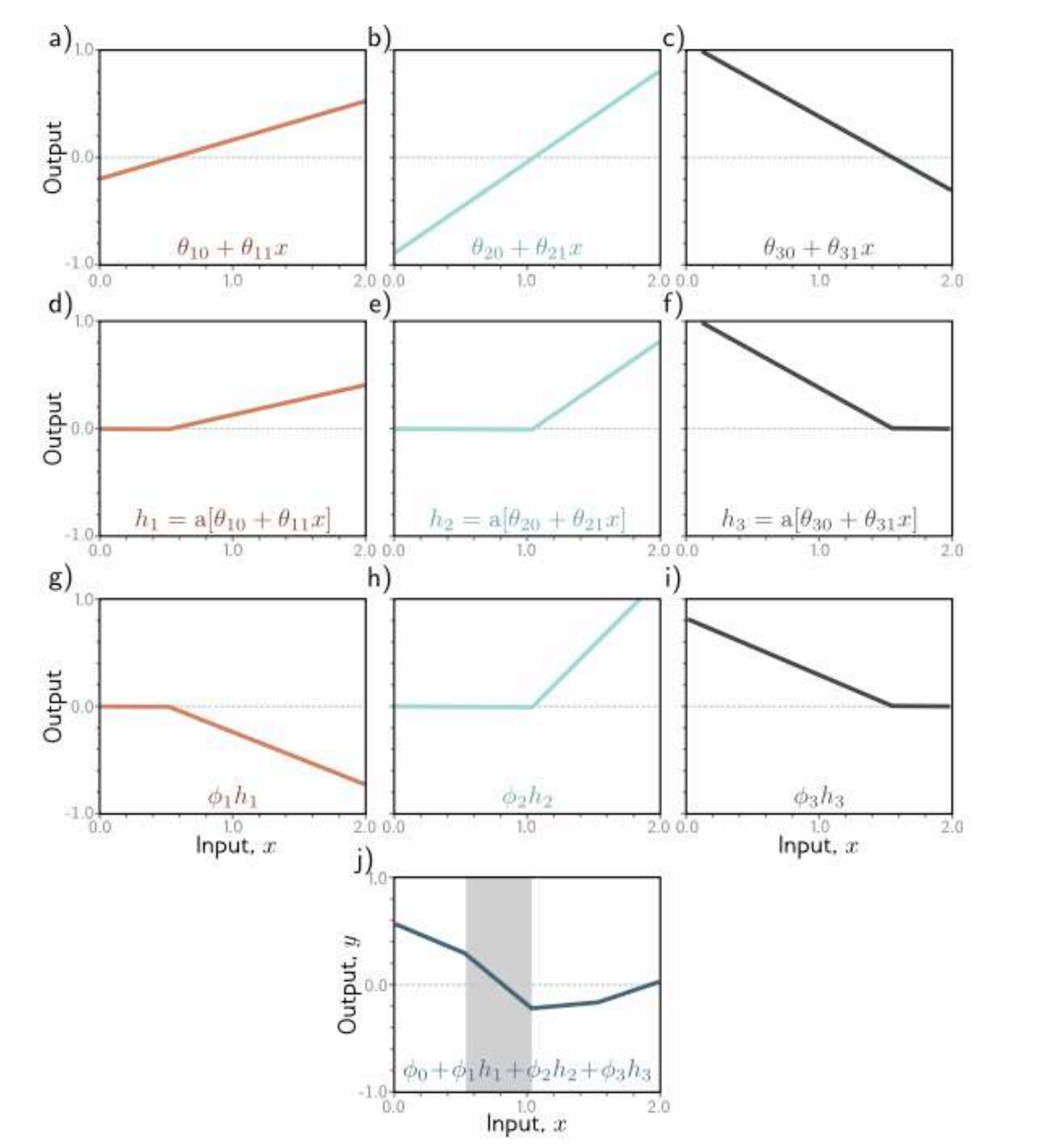
$$b \leftarrow b + \eta \frac{\partial z}{\partial b}$$

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

Identity kernel

-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

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

$$\sigma(w * x)$$

Non-linear transformation

Linear transformation

Deep neural networks

Deep neural networks

$$\downarrow$$

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

$$\downarrow$$

Deep neural networks

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

$$+$$

Timeseries

Motivation

A lot of data is sequential, varying over time:

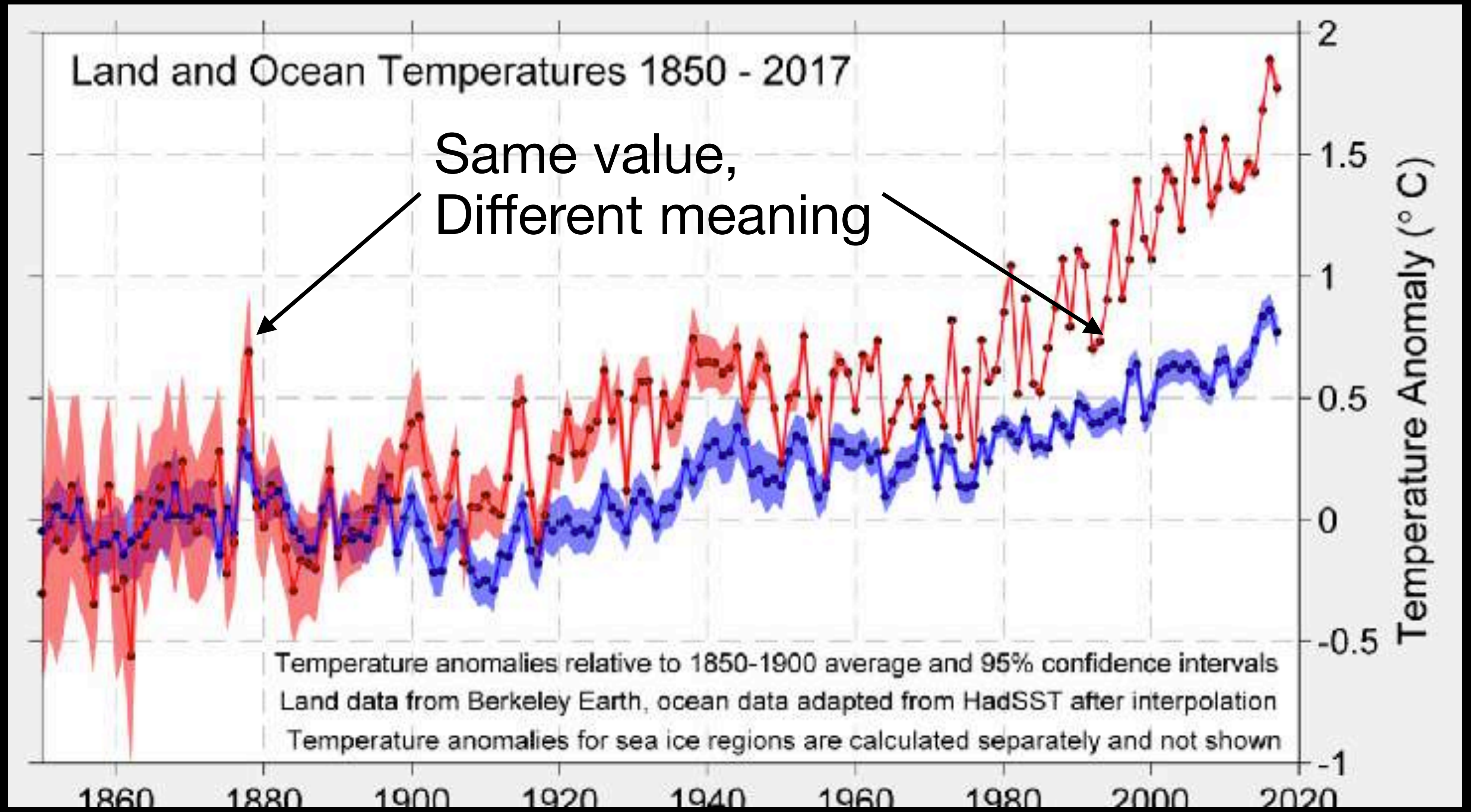
- Sentences
- Music
- EEG
- Movement
- Markets

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:

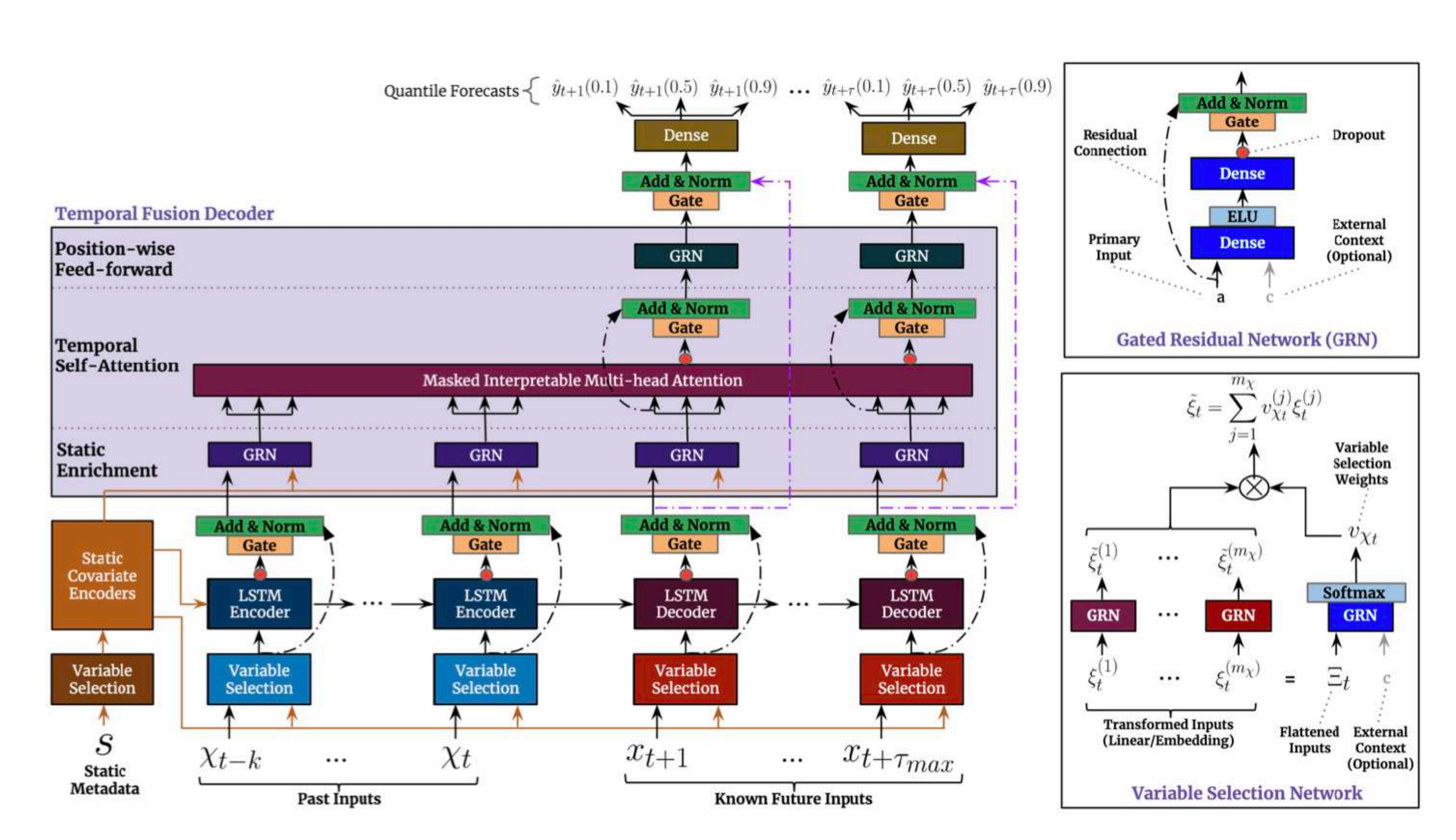
- How much of the past will we need (window)
- How much of the future do we want to predict (horizon)
- How to prepare the data without leaking data

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)

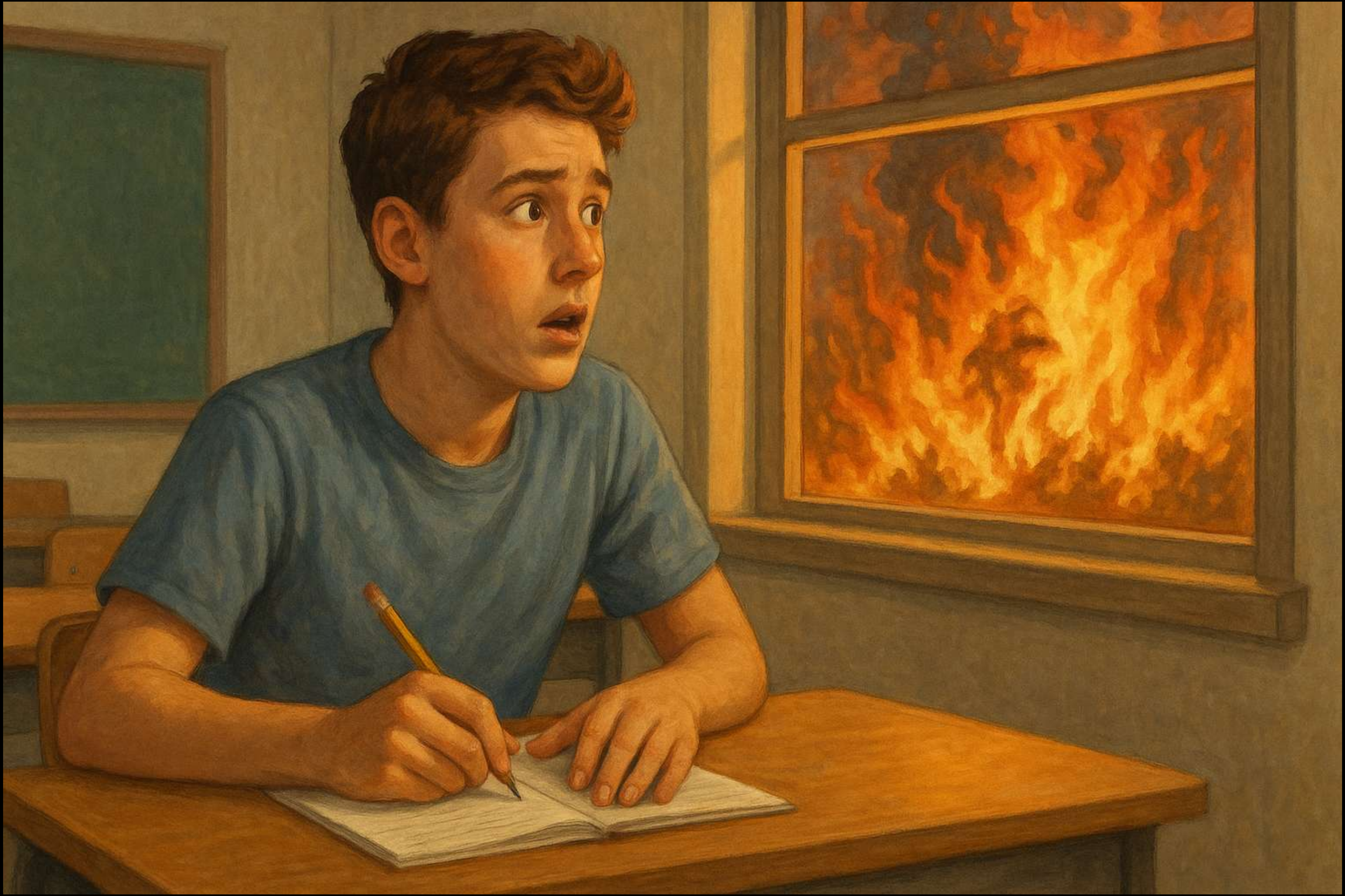
Temporal Fusion Transformer, Lim et al. (2021)



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



Concatenation

1	1	-1	0	1

x_t ← 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

h_{t-1}

← Previous state

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

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

Equivalent output

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

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

RNN

$\hat{y}$

$\sigma\left(wx + b\right)$

x

Neural network

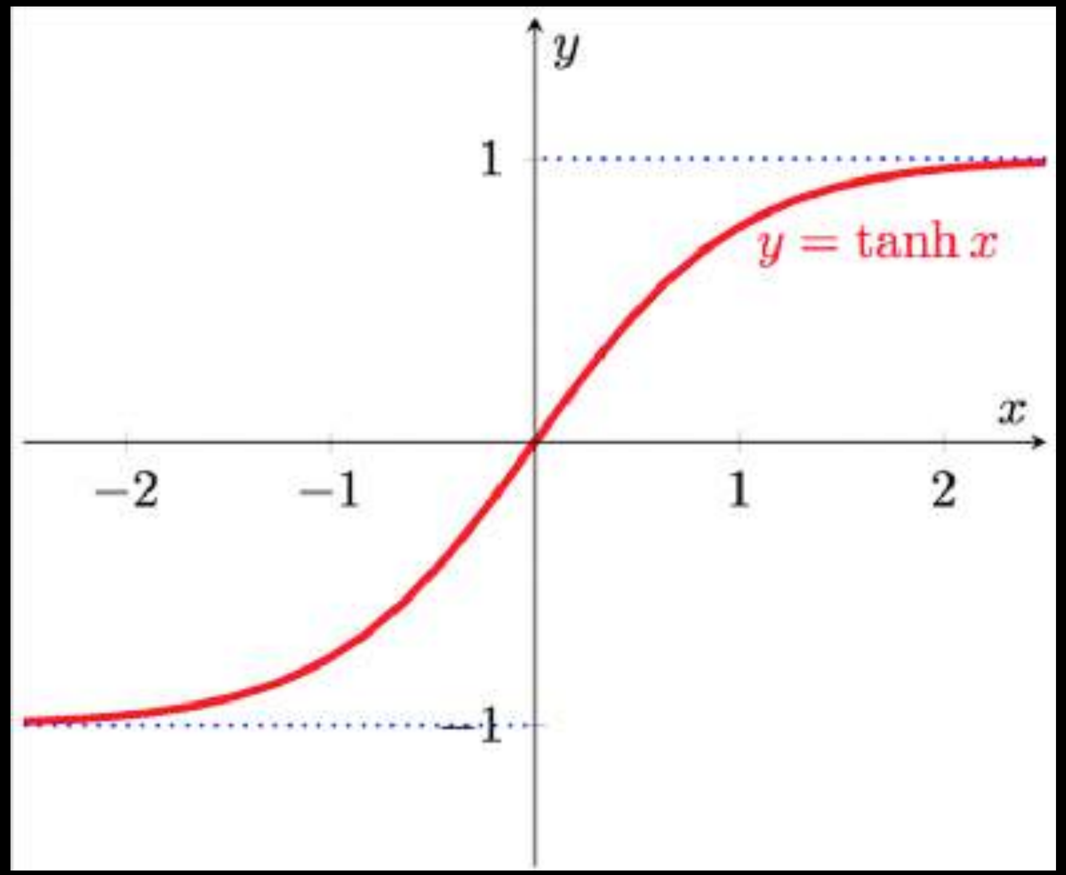
RNN

$\hat{y}_t$

$$\sigma\left(wx_t + b\right)$$

Tanh

x_t

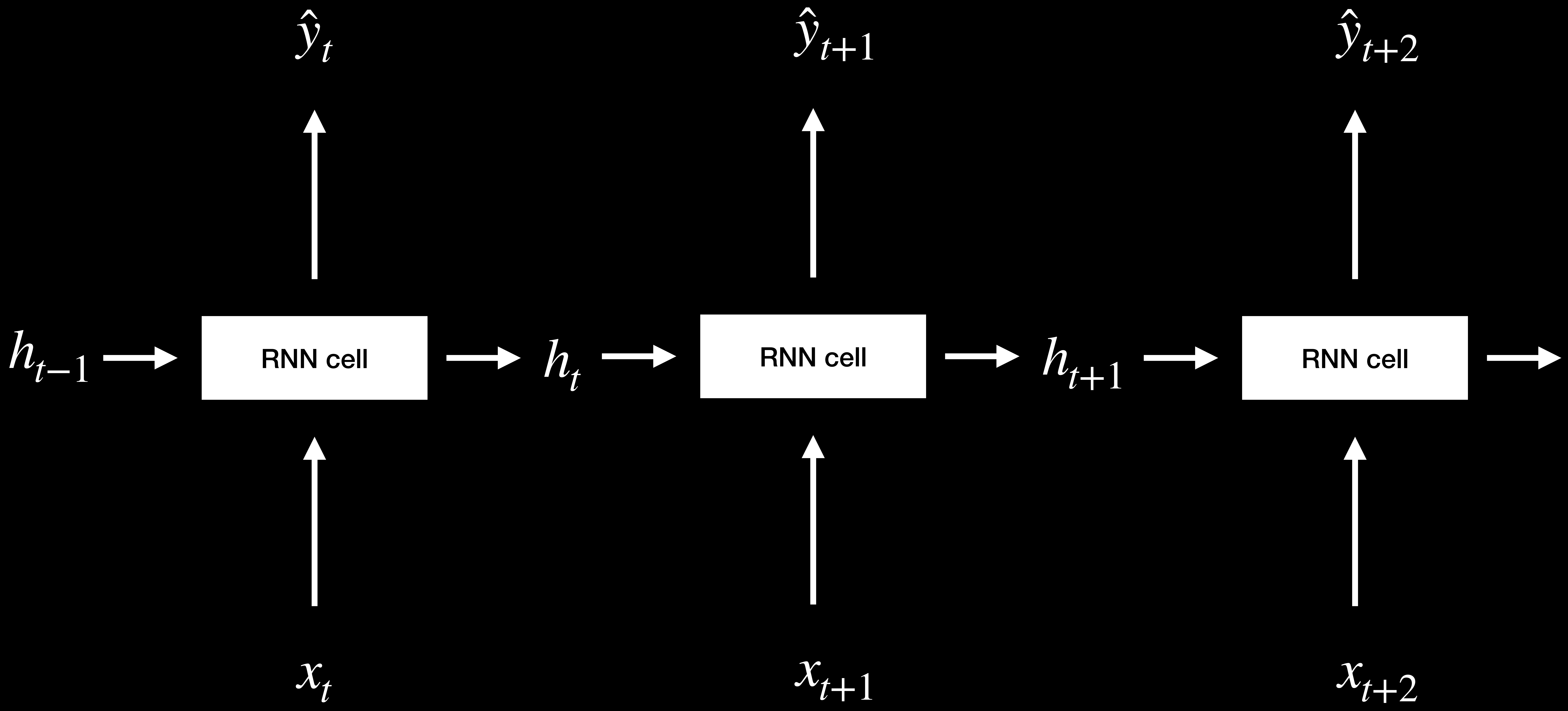


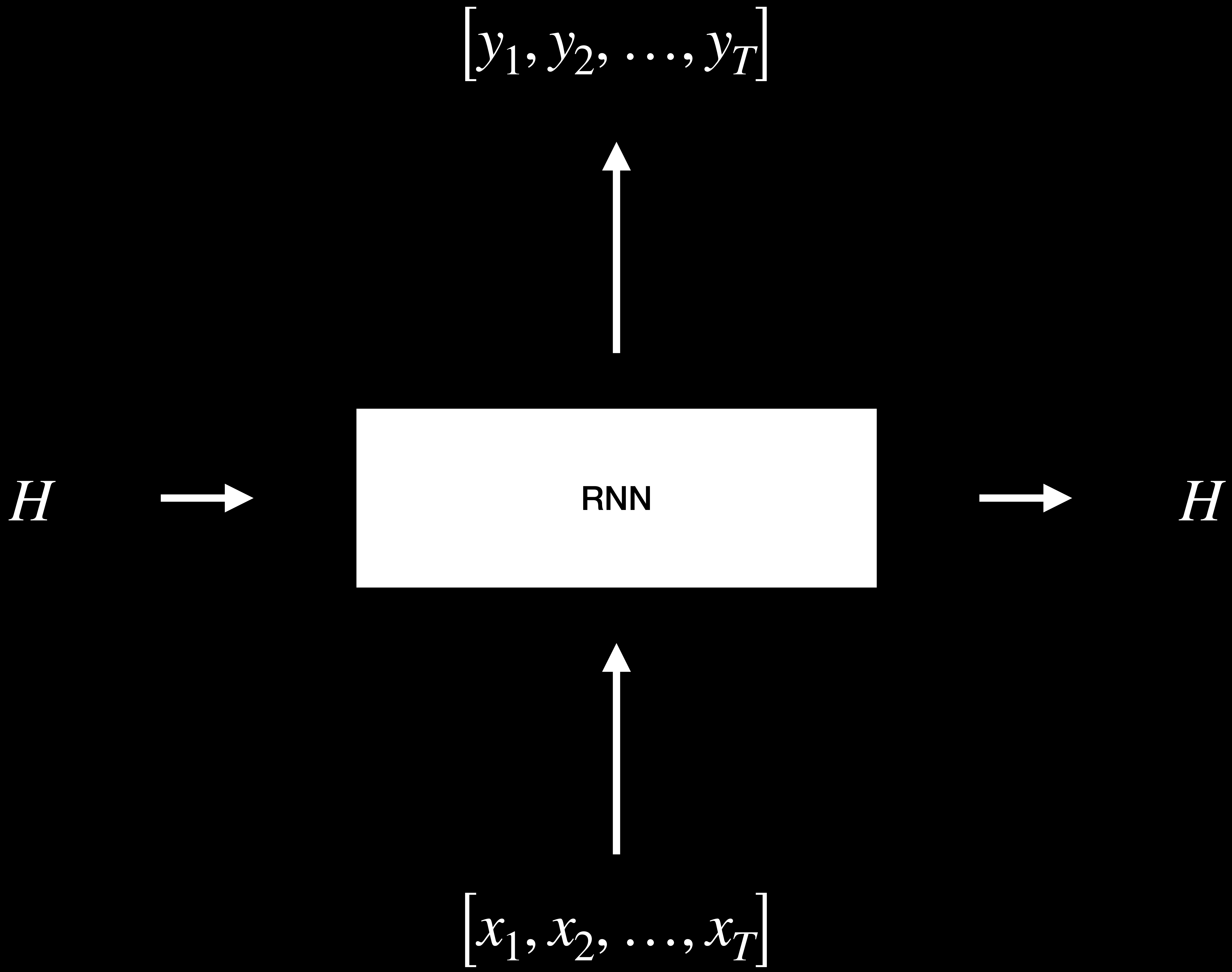
RNN

$$
\begin{tikzcd}
 & \hat{y}_t & \\
h_{t-1} \arrow[r] & \sigma\left(w\left[x_t, h_{t-1}\right]+b\right) \arrow[u] \arrow[r] & h_t \\
 & x_t \arrow[u] &
\end{tikzcd}
$$

RNN

$$
\begin{tikzcd}
 & \hat{y}_t & & \hat{y}_{t+1} & \\
h_{t-1} \arrow[r] & \sigma\left(w\left[x_t, h_{t-1}\right] + b\right) \arrow[u] \arrow[r] & h_t \arrow[r] & \sigma\left(w\left[x_{t+1}, h_t\right] + b\right) \arrow[u] \arrow[r] & h_{t+1} \\
 & x_t \arrow[u] & & x_{t+1} \arrow[u] &
\end{tikzcd}
$$


$\hat{y}_t$
$\hat{y}_{t+1}$
$\hat{y}_{t+2}$
h_{t-1}
RNN cell
h_t
RNN cell
h_{t+1}
RNN cell
x_t
x_{t+1}
x_{t+2}


$[y_1, y_2, \ldots, y_T]$
H
RNN
H
$[x_1, x_2, \ldots, x_T]$

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

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

$$X \otimes \Gamma$$

Hadamard product

Gates

Same shape as X

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

$X \quad \otimes \quad \Gamma$

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

$$X \otimes \Gamma$$

Gate

Gates

$$\underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}}_{X} \otimes \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{\Gamma} = \underbrace{\begin{bmatrix} & & & \\ & & & \\ & & & \\ & & & \end{bmatrix}}_{Y}$$

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

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

$$X \otimes \Gamma = Y$$

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

$$X \otimes \Gamma = Y$$

Gates

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}$$

$$X \otimes \Gamma = Y$$

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

$$X \otimes \Gamma = Y$$

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

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

$$X \otimes \Gamma = Y$$

Gates

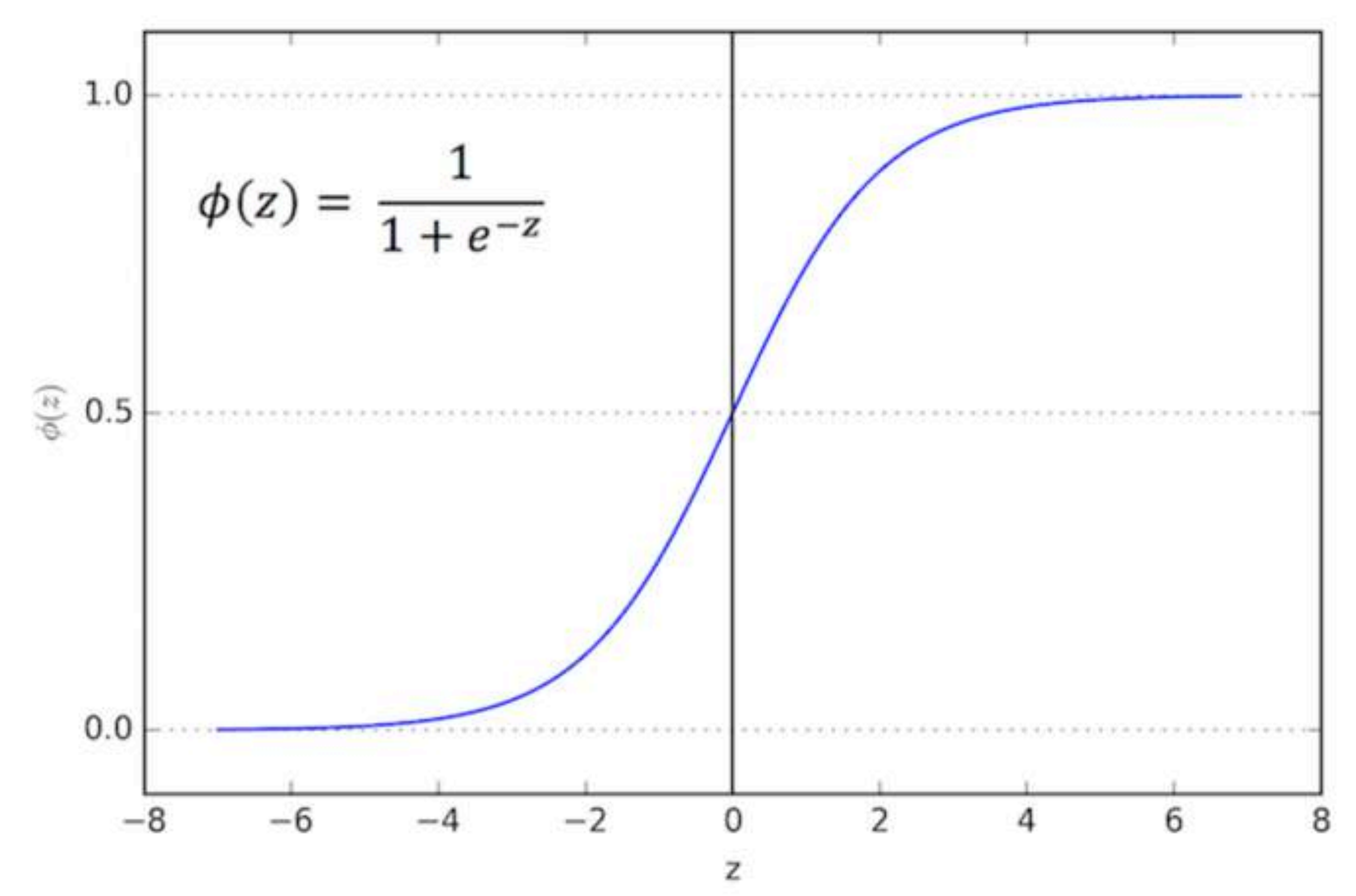
$$
\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}
\otimes
\begin{bmatrix} w & w & w & w \\ w & w & w & w \\ w & w & w & w \\ w & w & w & w \end{bmatrix}
=
\begin{bmatrix} y & y & y & y \\ y & y & y & y \\ y & y & y & y \\ y & y & y & y \end{bmatrix}
$$

$$X \otimes \Gamma = Y$$

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

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

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

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

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

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

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

GRU

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

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

h_{t+1}
Gate
X
X
X
X
X
X
X
X
X
X
X
X
X
X

GRU - full

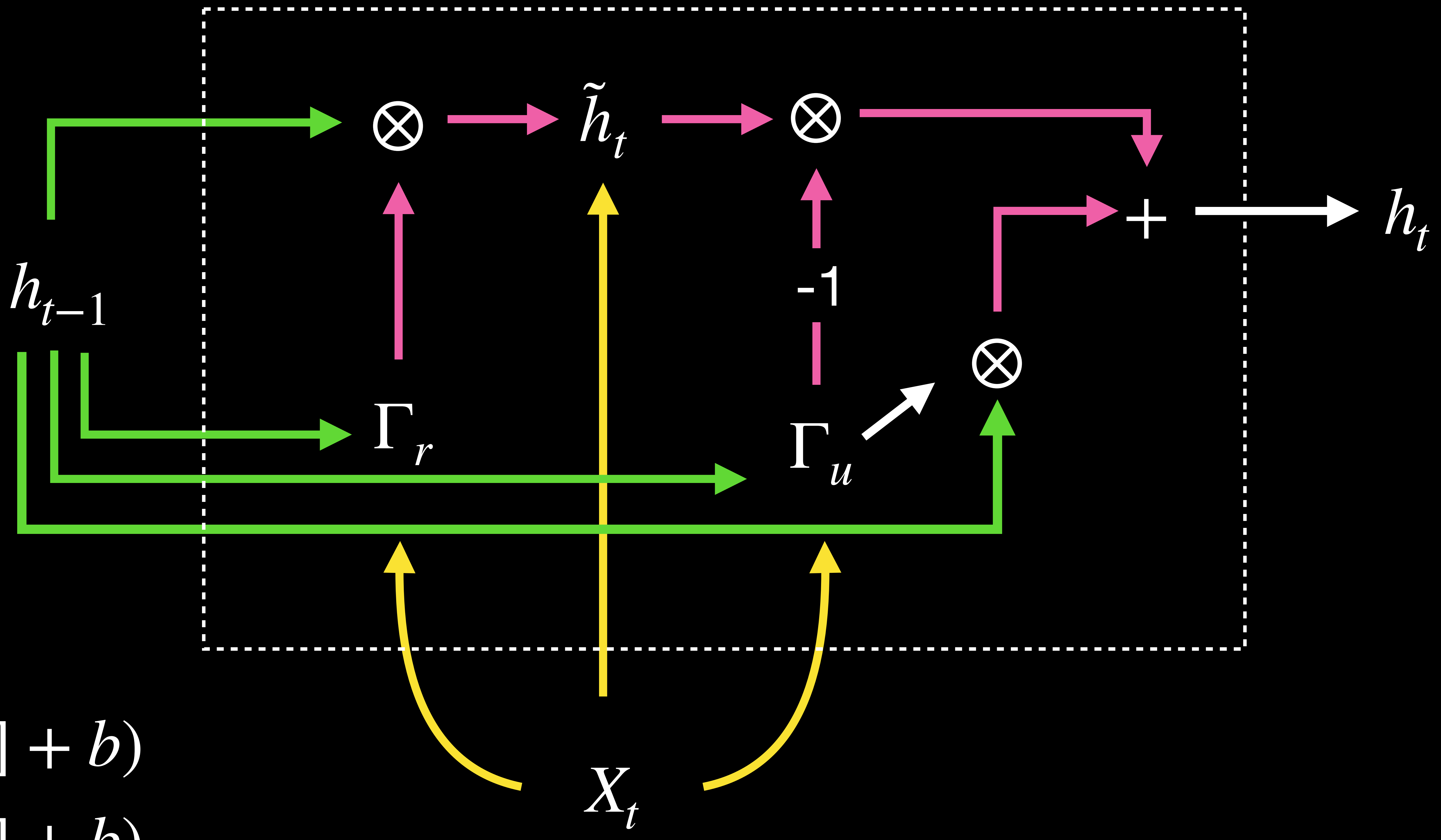
The full GRU has two gates, but the principle is the same

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

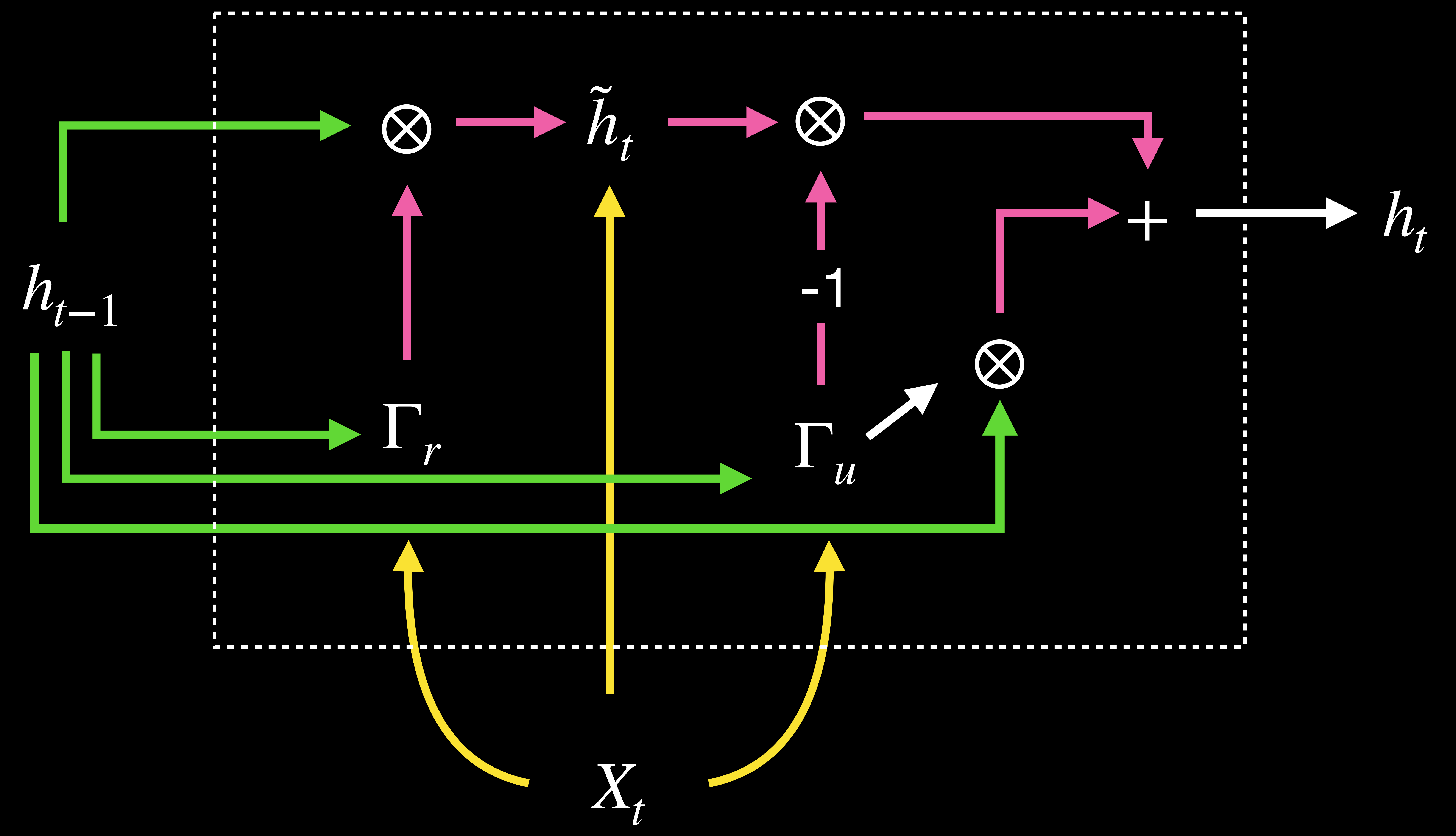
$$\Gamma_r = \sigma(W[X_t, h_{t-1}] + b)$$

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

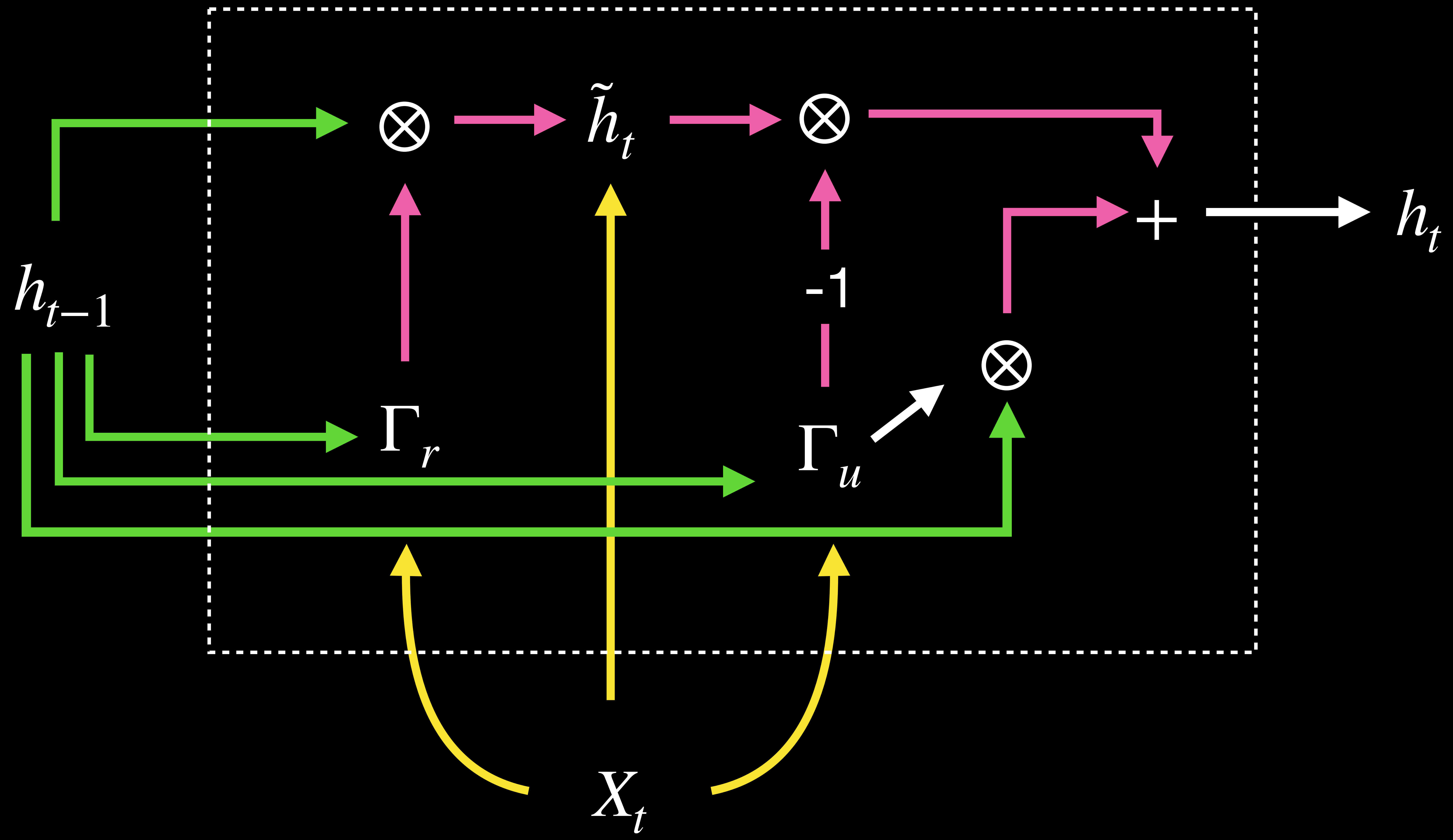
$$h_t = \Gamma_u \otimes h_{t-1} + (1 - \Gamma_u) \otimes \tilde{h}_t$$



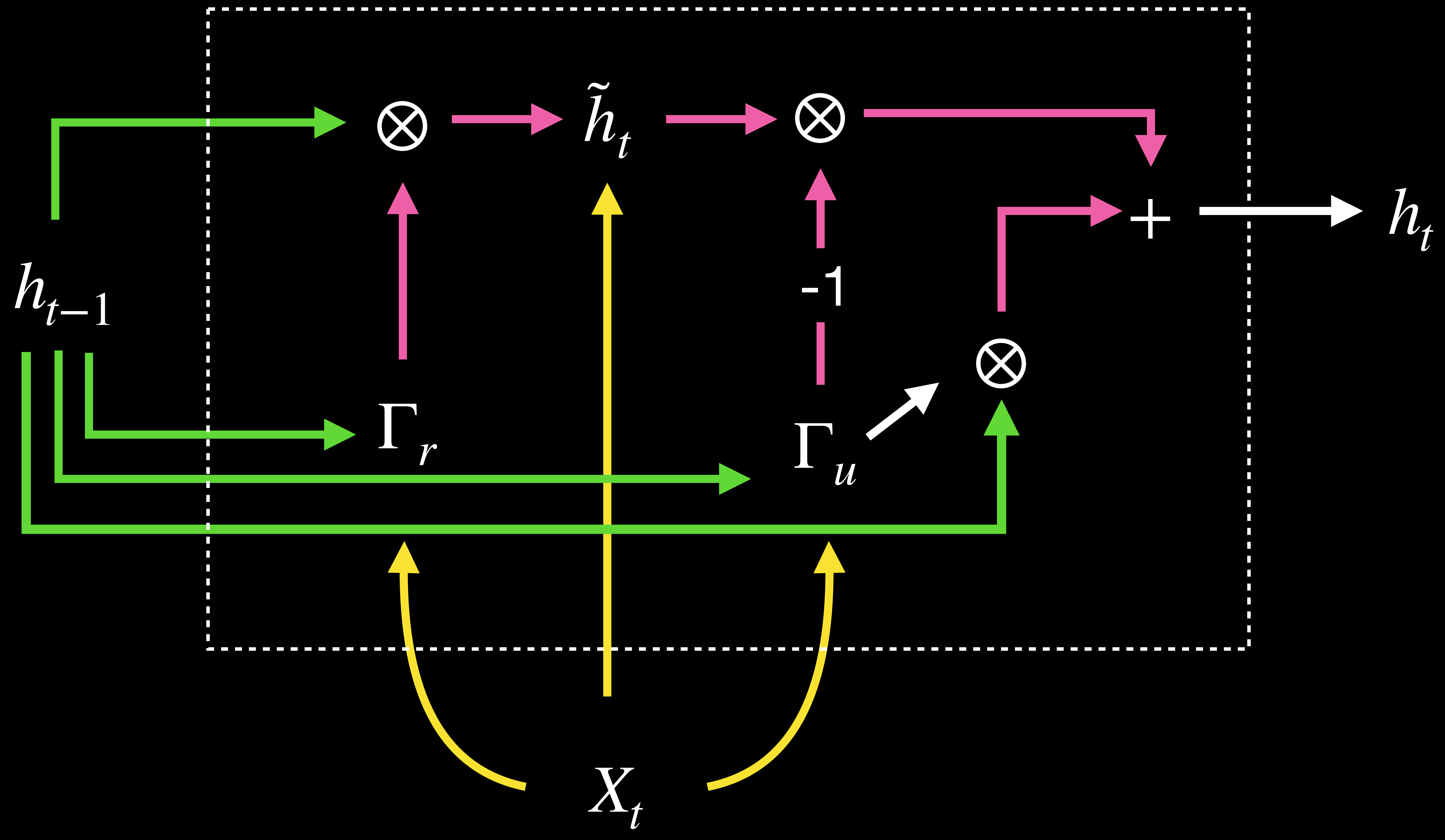
$$\Gamma_u = \sigma(W[X_t, h_{t-1}] + b)$$
$$\Gamma_r = \sigma(W[X_t, h_{t-1}] + b)$$
$$\tilde{h}_t = tanh(W[X_t, \Gamma_r \otimes h_{t-1}] + b)$$
$$h_t = \Gamma_u \otimes h_{t-1} + (1 - \Gamma_u) \otimes \tilde{h}_t$$



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
```

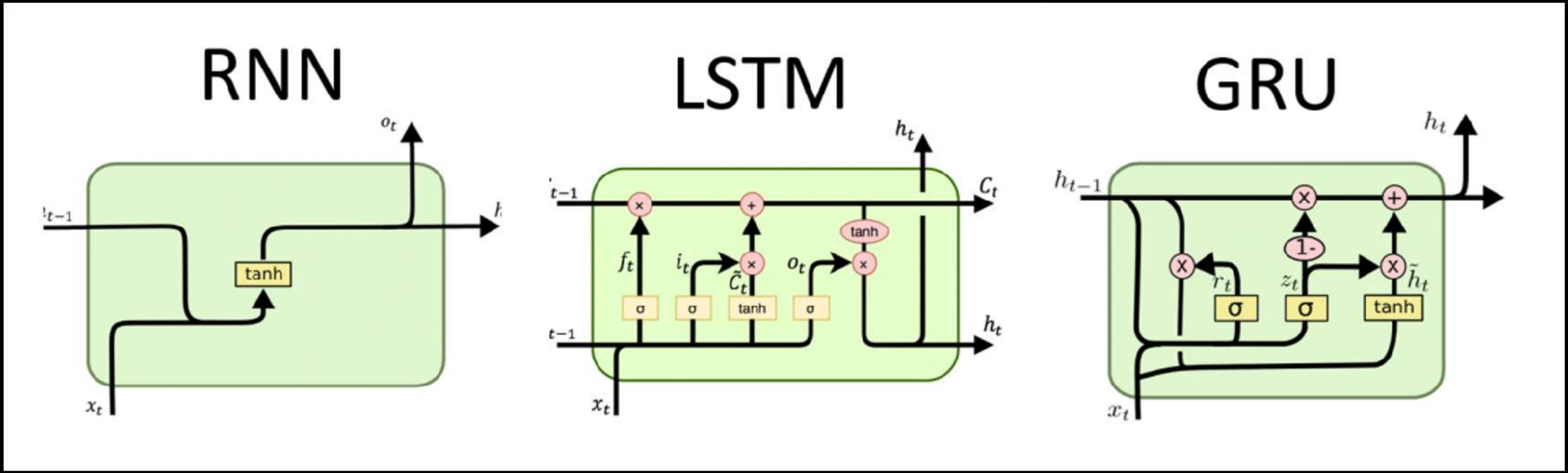
LSTM

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

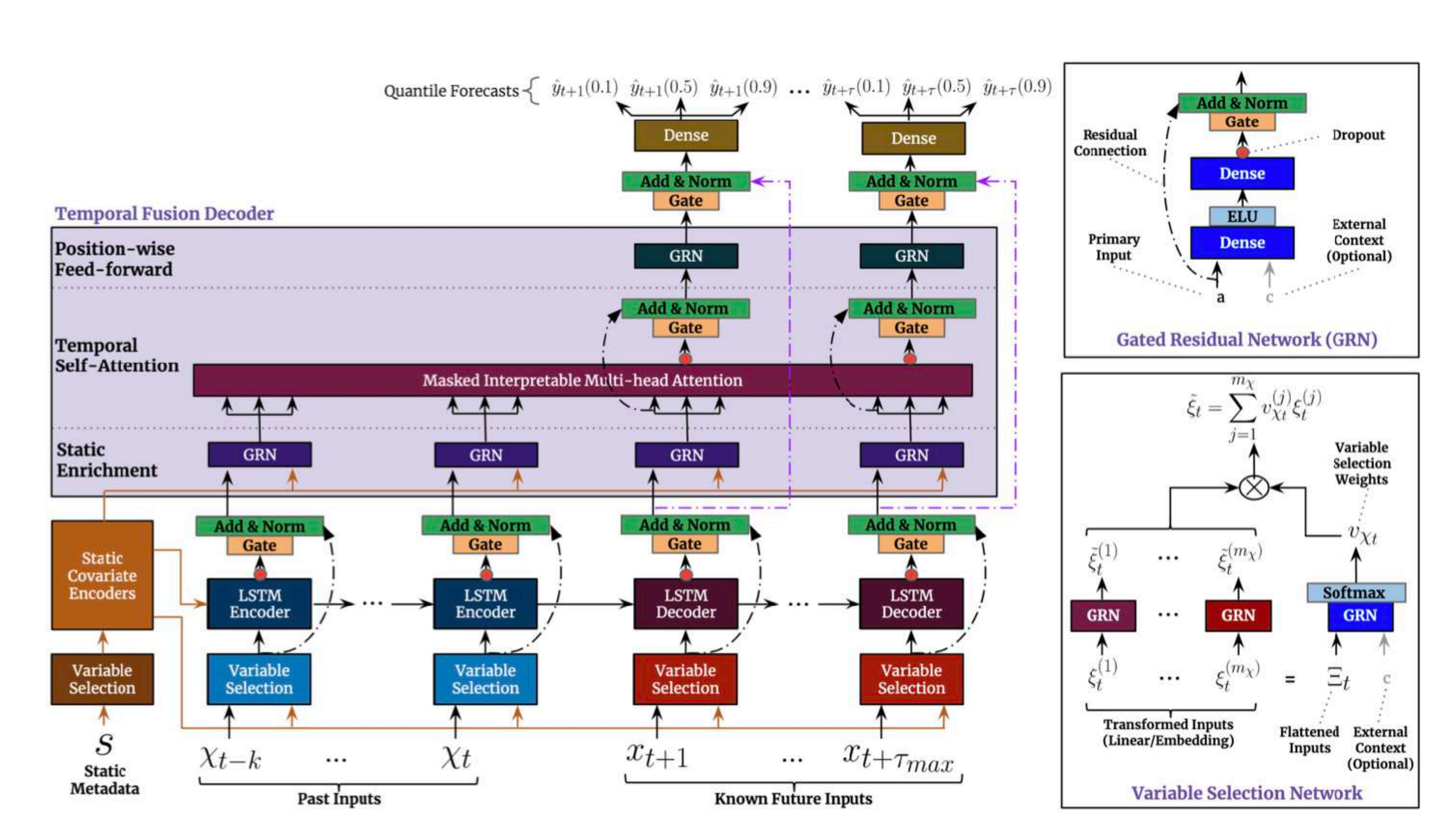
$$\Gamma_u = \sigma(W[X_t, h_{t-1}] + b)$$
$$\Gamma_i = \sigma(W[X_t, h_{t-1}] + b)$$
$$\Gamma_f = \sigma(W[X_t, h_{t-1}] + b)$$
$$\tilde{h} = \Gamma_i \otimes tanh(W[X_t, h_{t-1}] + b)$$
$$\tilde{C} = tanh(\Gamma_f \otimes C + \tilde{h})$$
$$h_t = \Gamma_u \otimes \tilde{C}$$

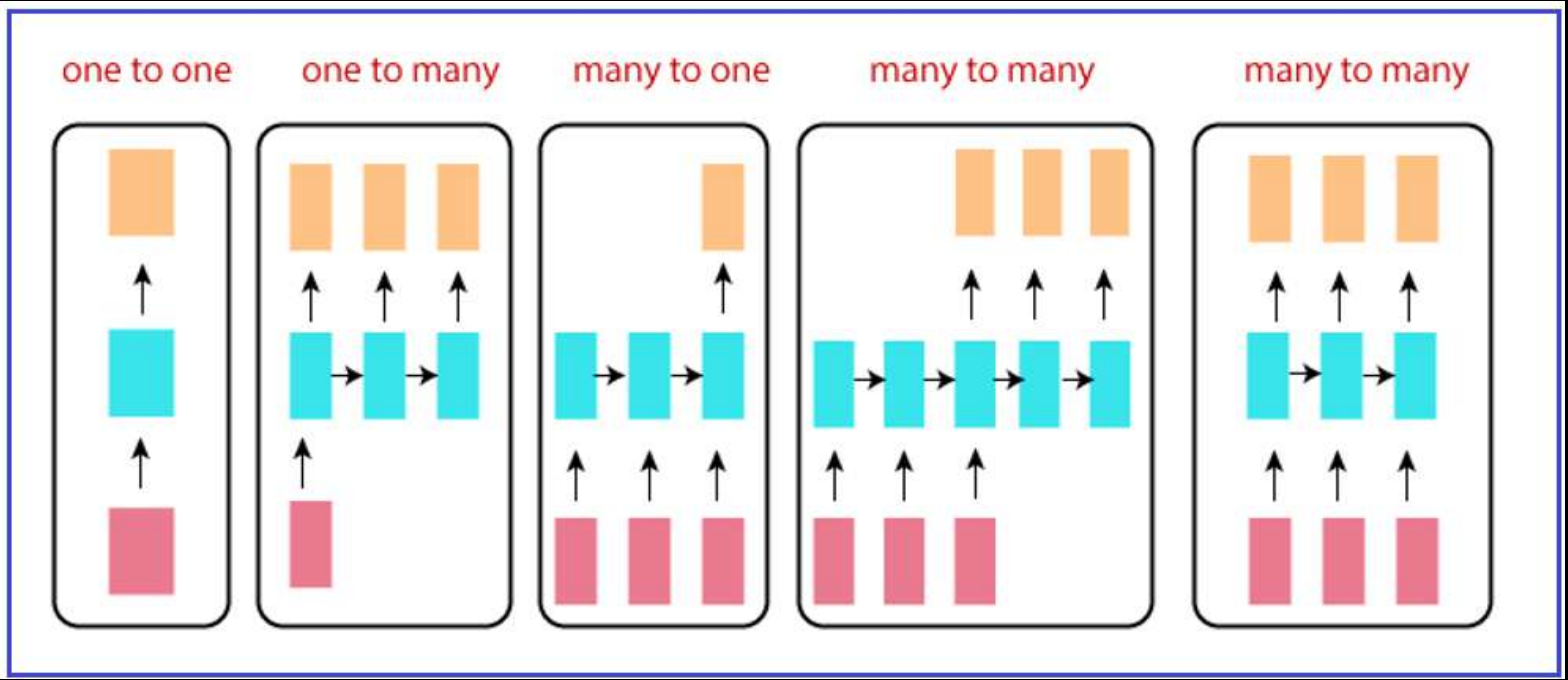
Overview



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

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 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.