15 - Recurrent Networks

Ludwig Krippahl

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Summary

- Recurrent neural networks
  - Unfolding
- Backpropagation through time
- Long term dependencies
- Structured RNNs
  - Gated recurrent units (GRU)
  - Long short term memory (LSTM)
- Brief introduction:
  - RNN are being replaced by CNN and Transformers in many applications

Recurrent neural networks

Recurrent Neural Network

- In a recurrent network outputs are fed into the network with a delay
- Two important concepts:
  - Stacking nonlinear transformations (the usual in deep networks)
  - Parameter sharing (like we saw in CNN)
- Motivation:
  - Recurrent networks use the same parameters through a sequence
  - But the state can be a function of the history of the inputs
- Especially suited for problems with sequential data

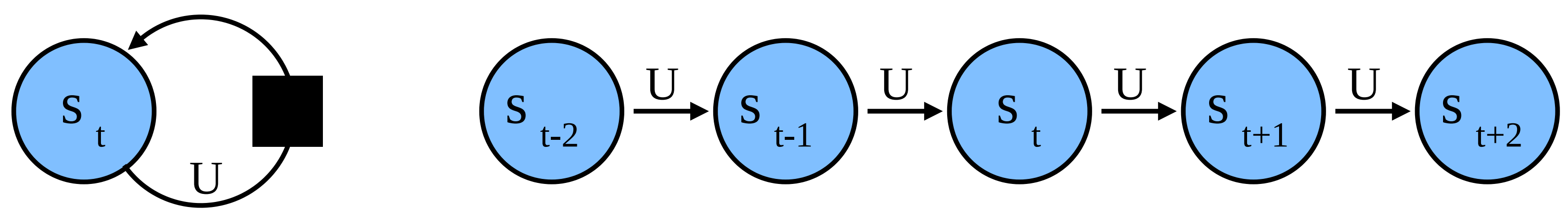
Unfolding the network

- Let us consider the following recursion:

$$s_t = f(U, s_{t-1})$$

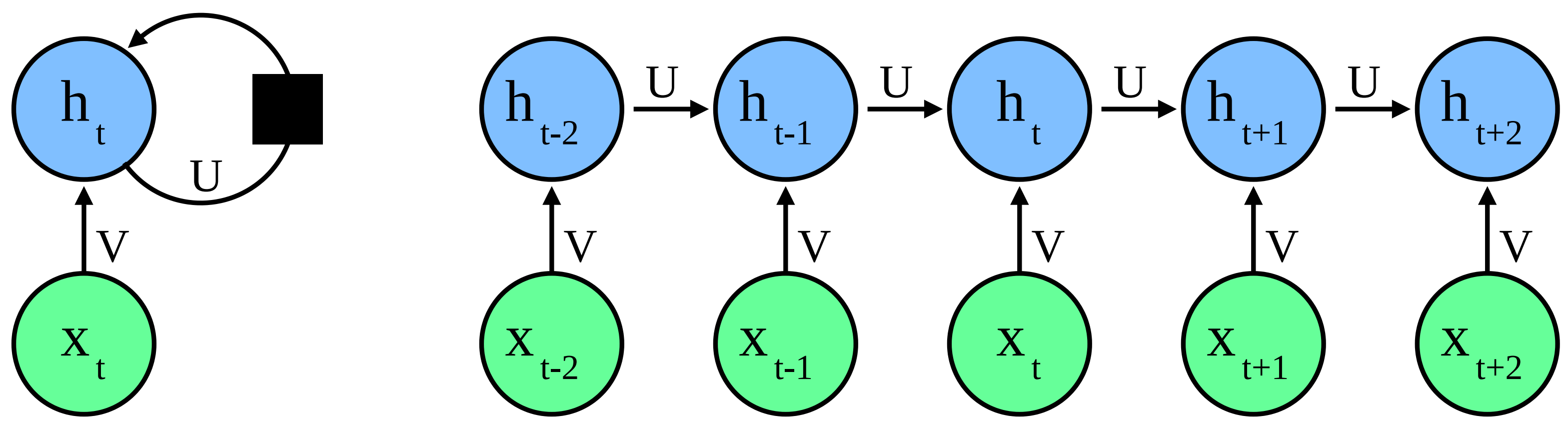
- We can unfold it over 2 steps with:

$$s_t = f(U, f(U, f(U, V, x_{t-2})))$$



RNN

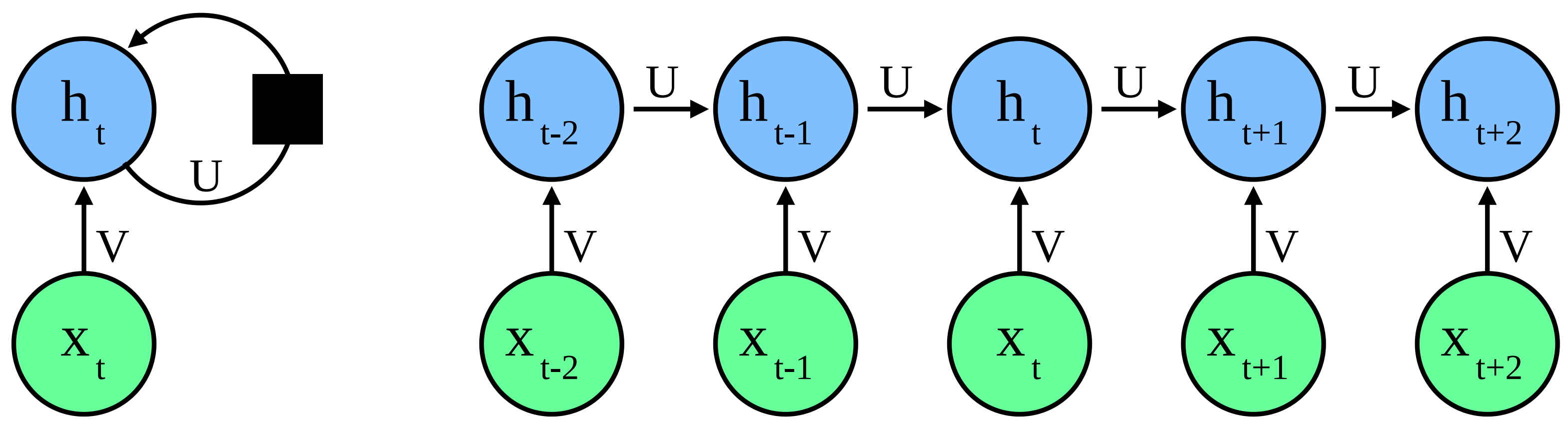
- To deal with sequential data, we feed inputs in sequence



- The state keeps a historical record of the inputs
- The shared parameters make it easier to recognize patterns that do not depend on position

RNN

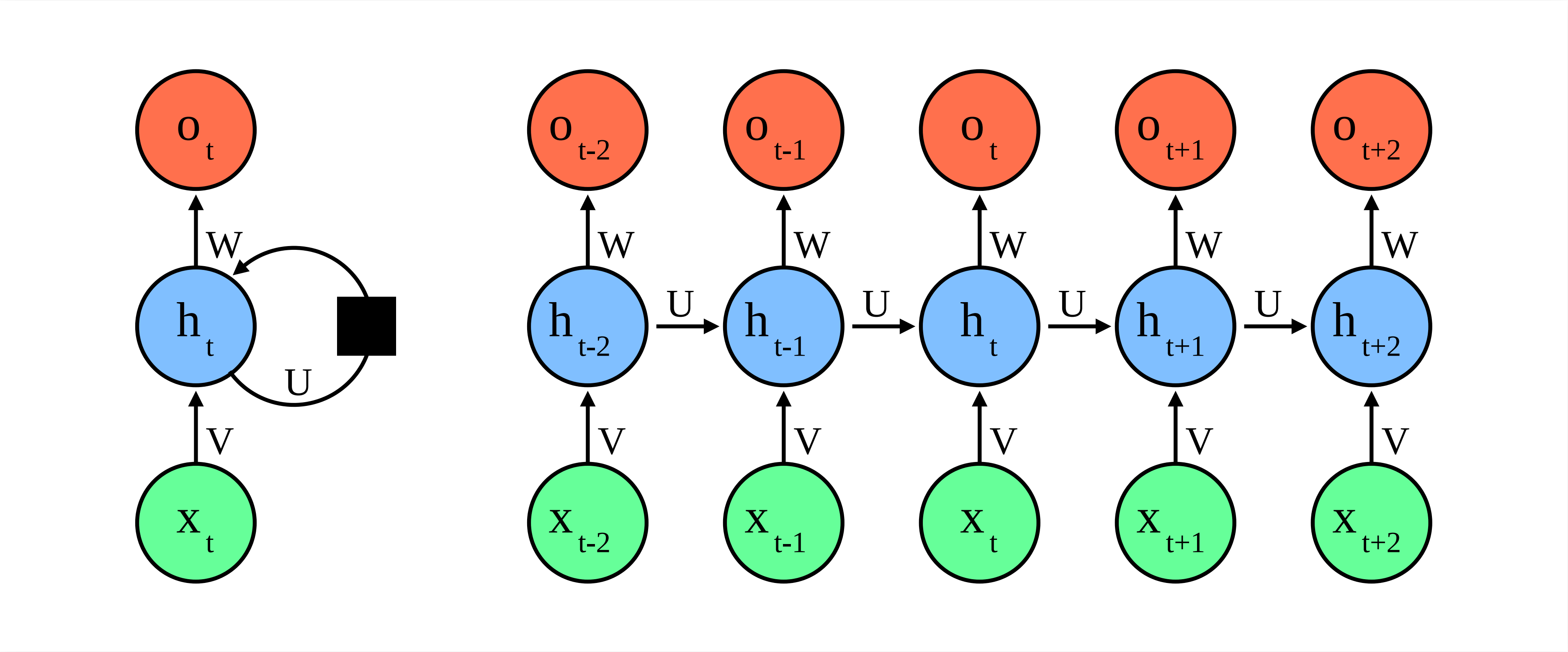
- To deal with sequential data, we feed inputs in sequence



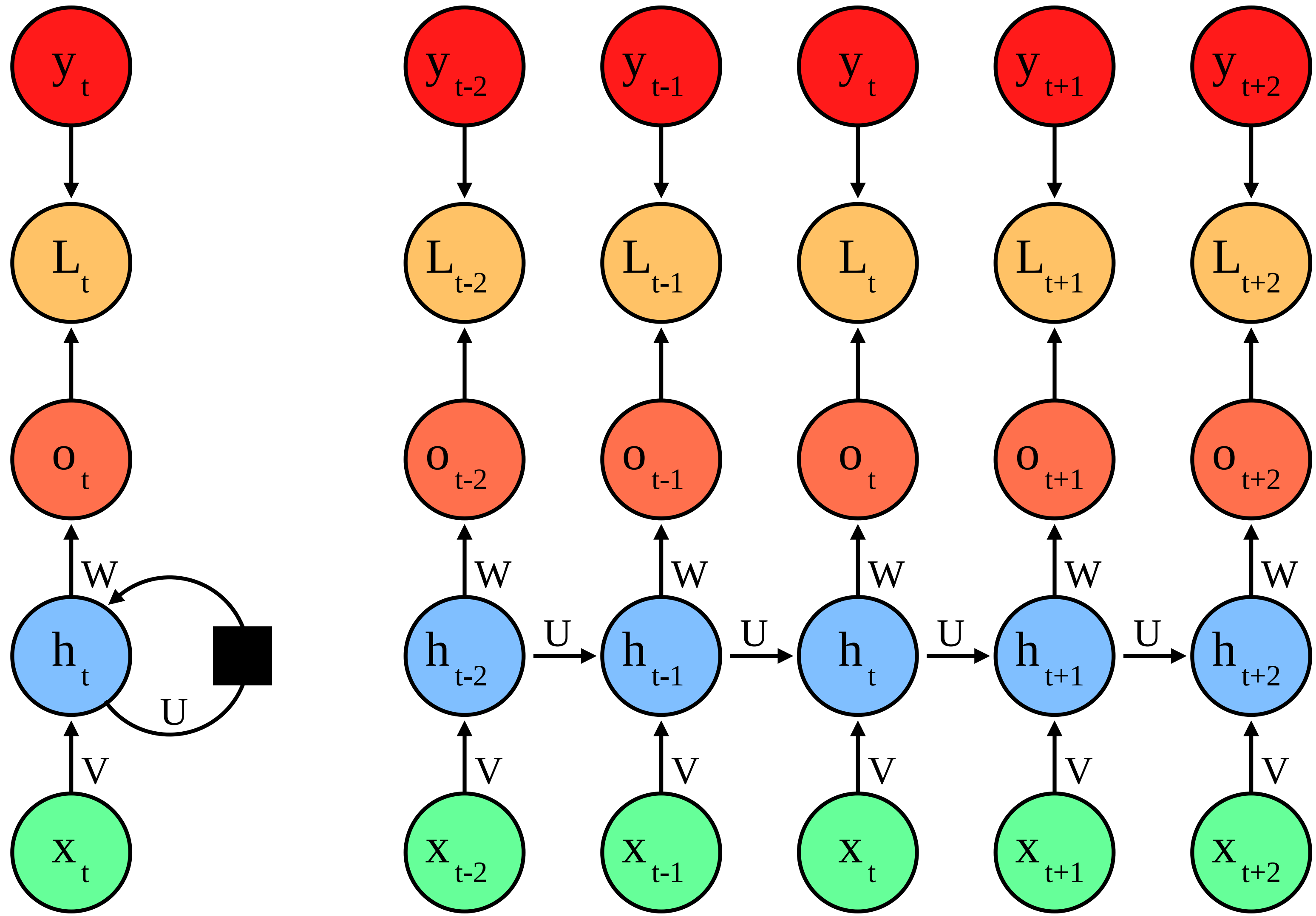
- This makes RNN good for text, audio, time series (weather, markets, etc)
- Can deal with input sequences of variable length
  - But not in the same batch during training in Keras

RNN

- Multiple outputs:

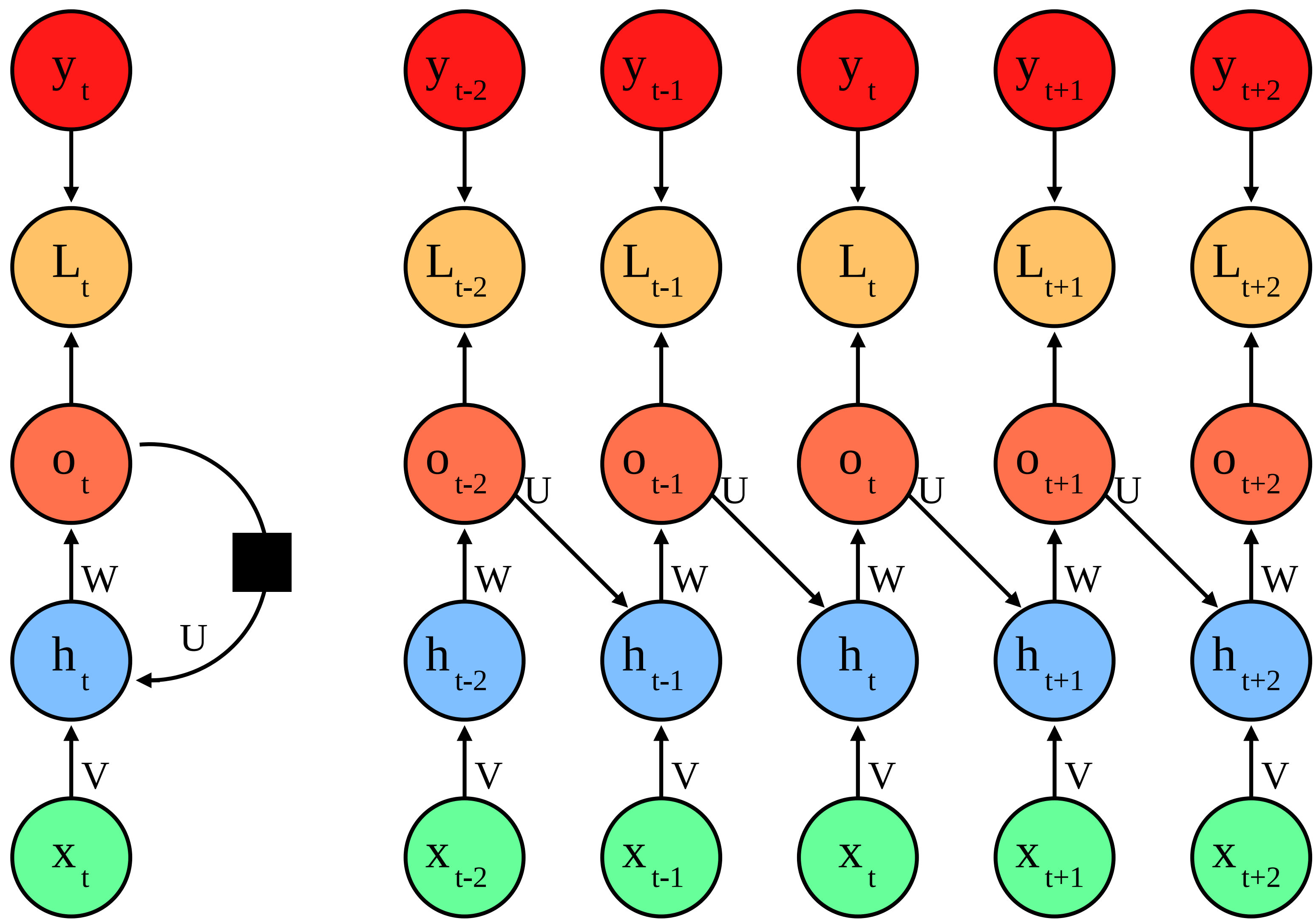


RNN



FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

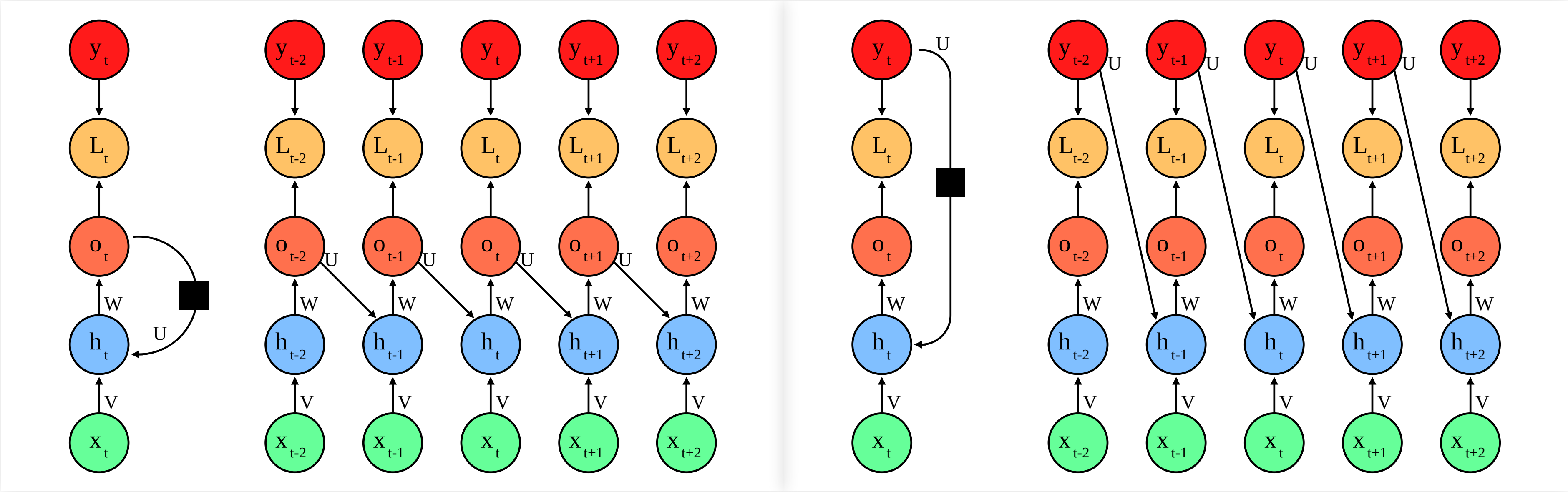
RNN



y t
L t
o t
h t
x t
W
U
V
y t-2
y t-1
y t
y t+1
y t+2
L t-2
L t-1
L t
L t+1
L t+2
o t-2
o t-1
o t
o t+1
o t+2
h t-2
h t-1
h t
h t+1
h t+2
x t-2
x t-1
x t
x t+1
x t+2
U
U
U
U
W
W
W
W
W
V
V
V
V
V

Teacher Forcing

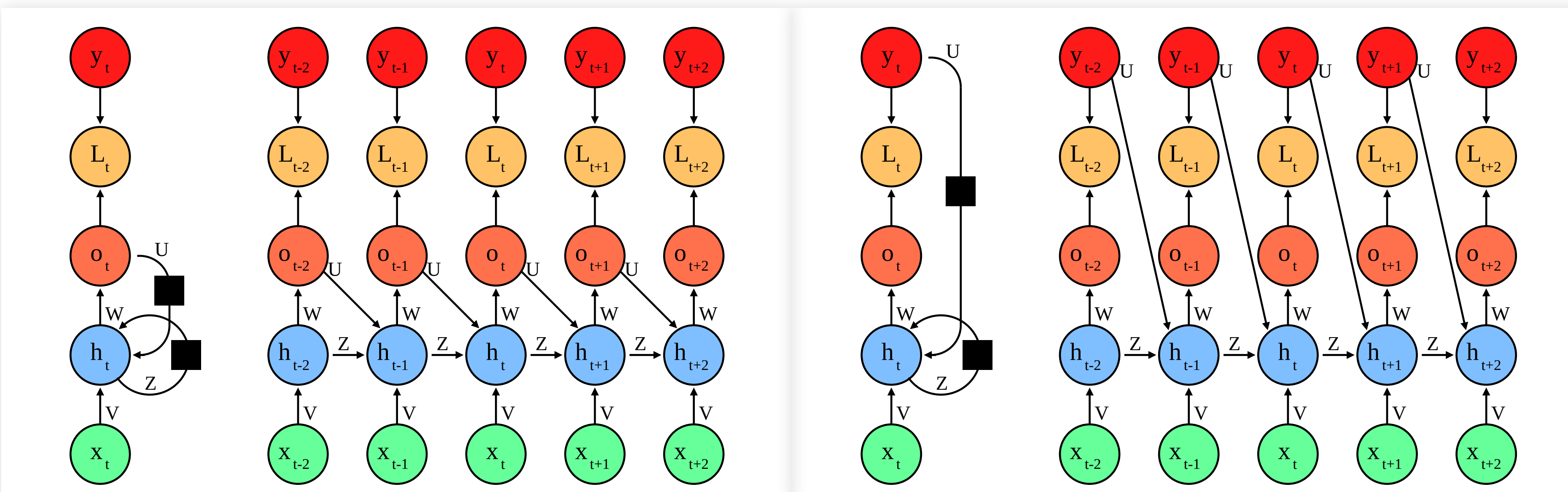
- If the state depends on the output of the previous iteration we can train with the ground truth instead of the predicted values
- This not only improves training but makes it easy to parallelize if state s_t only depends on y_{t-1}



Teacher Forcing

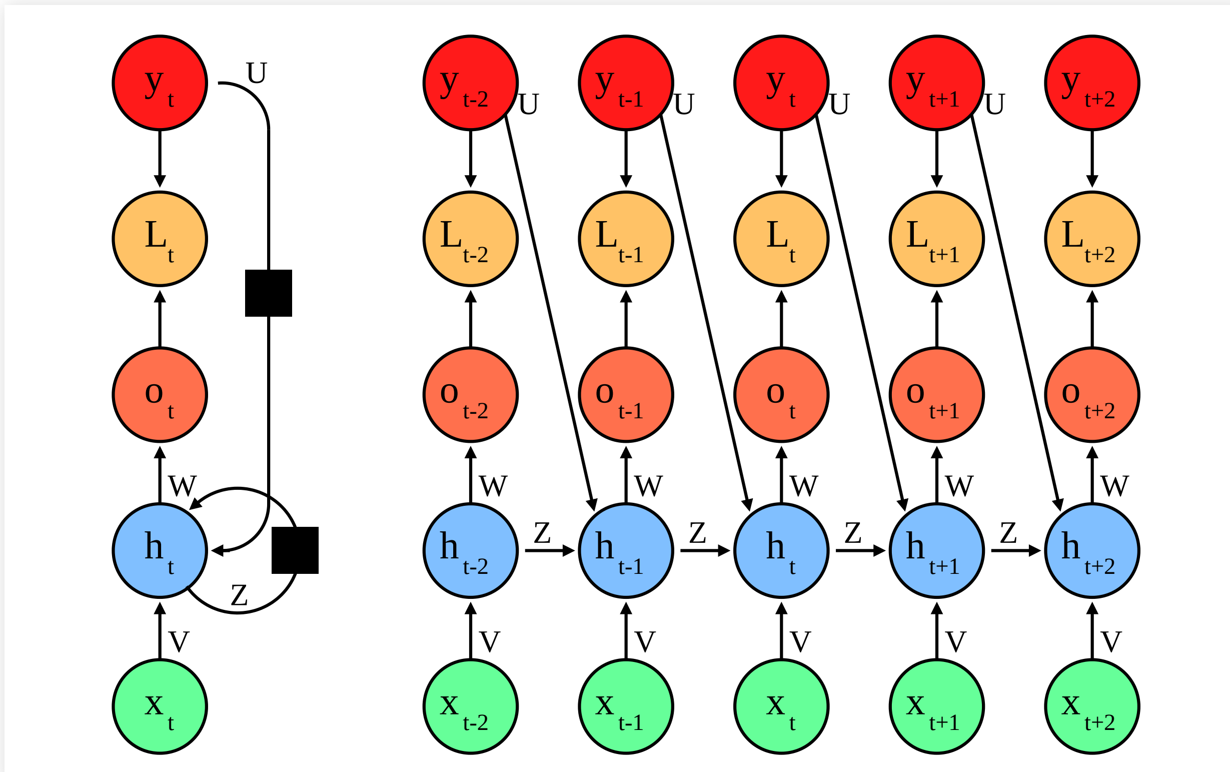
- Even if we cannot parallelize training (if the state depends on hidden layers) using y_{t-1} instead of o_{t-1} is the best approach since it is the ML solution.

- But during inference we use o_{t-1}, since y_{t-1} is only available for training.



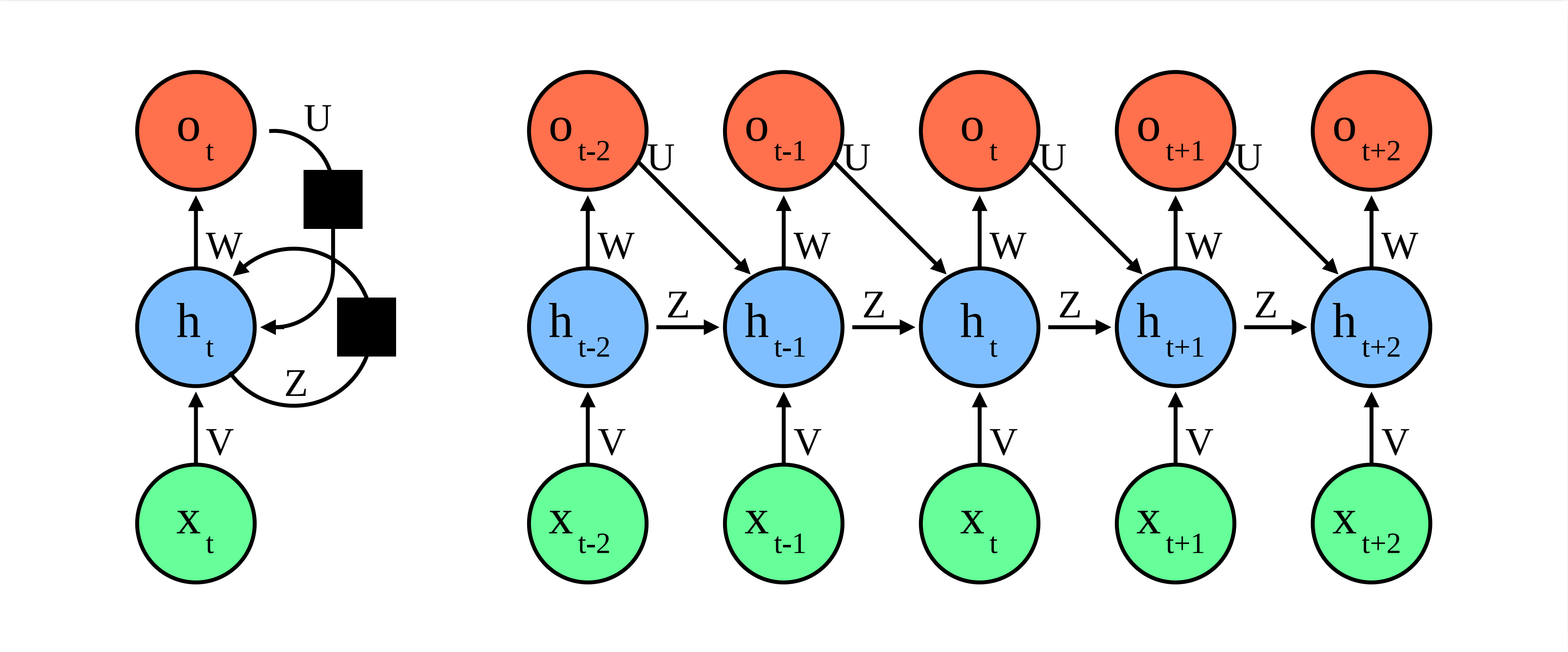
Teacher Forcing

- During training:



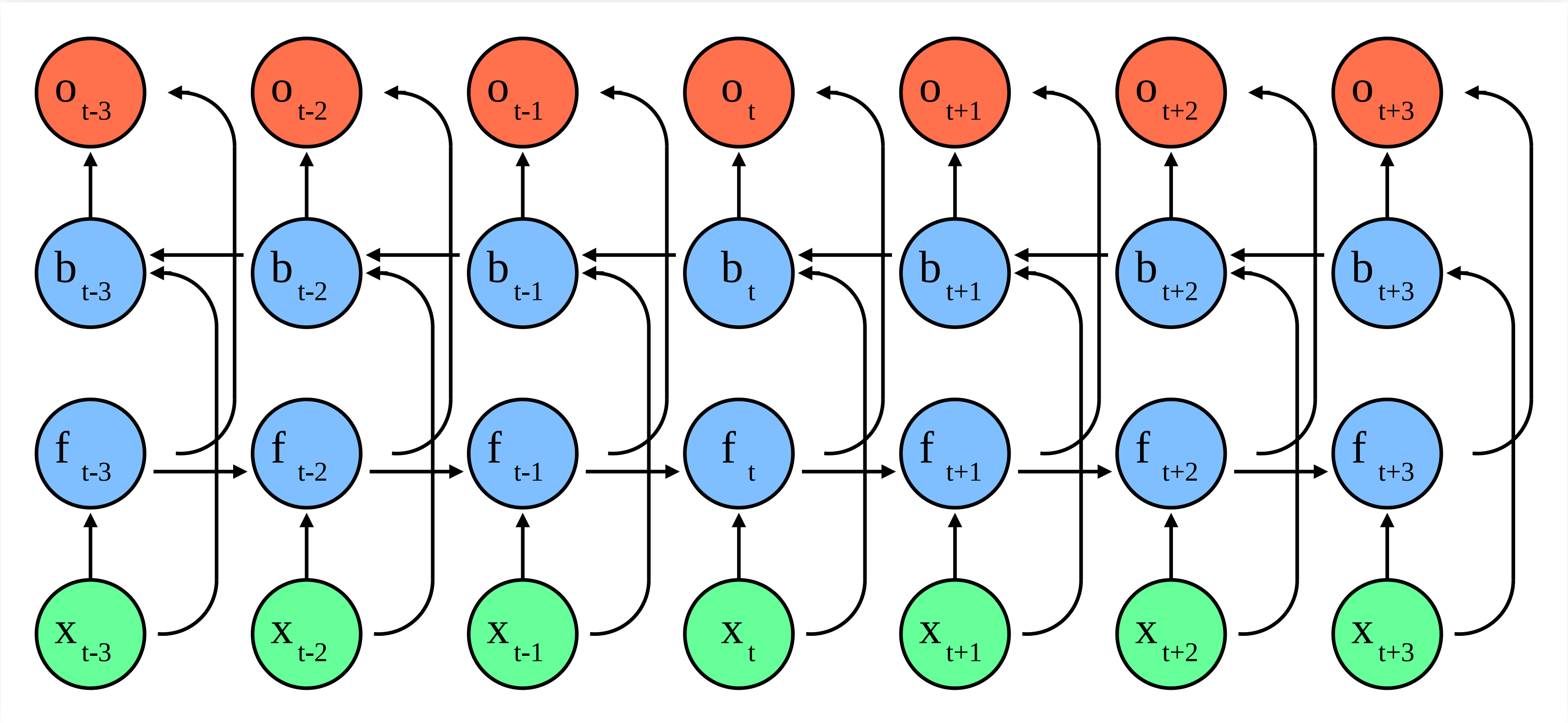
Teacher Forcing

- After training, for prediction (inference):



RNN

- RNN can be bidirectional (e.g. processing text)



- With Keras: `Bidirectional` layer

RNN

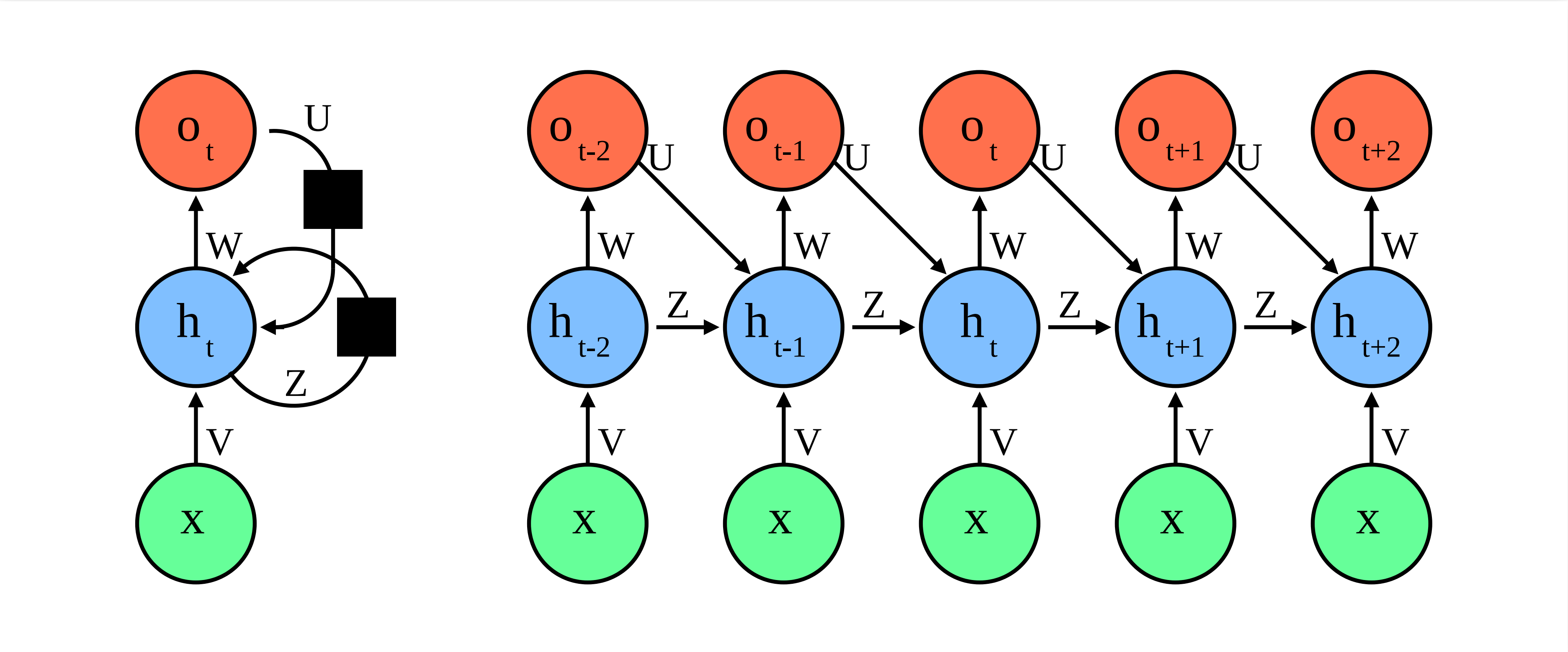
- RNN are not only for sequential data
- E.g Image captioning:

The black cat is walking on grass

The white cat is walking on road

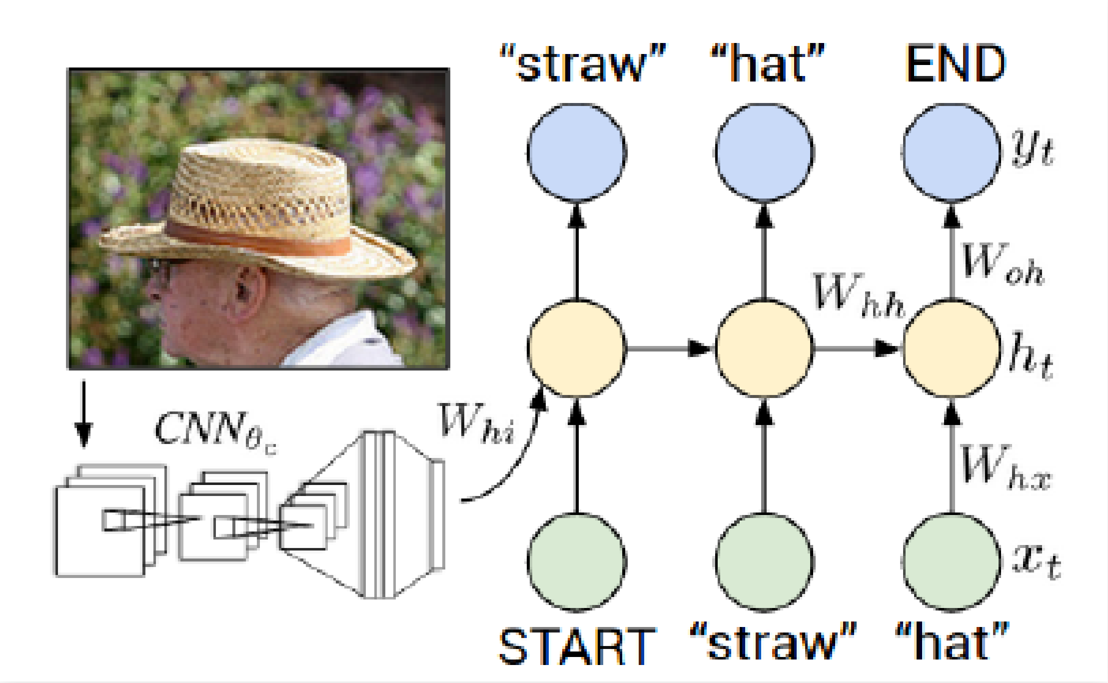
RNN

- Image captioning:
  - Conceptually, generate words giving image as context



RNN

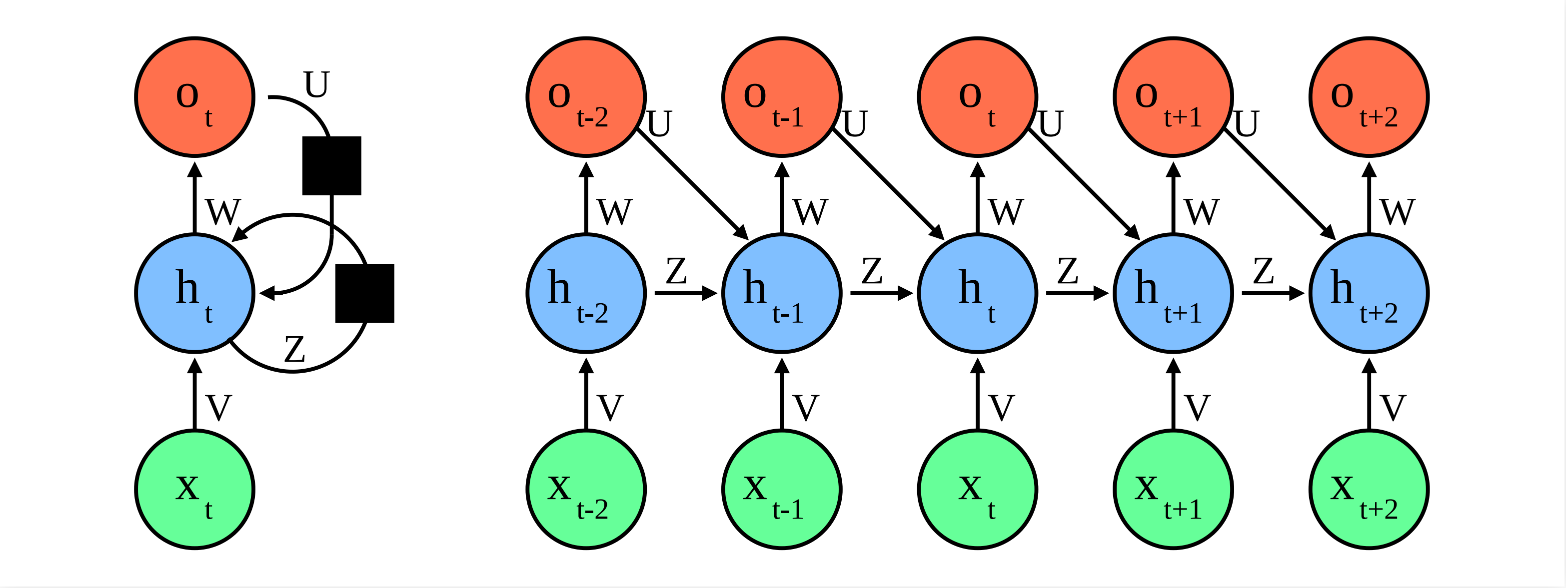
- In practice:
  - This is not easy to do because of having to join fixed and time dependent inputs
  - A better way is to condition the starting state of the RNN using the fixed data



Karpathy, Fei-Fei, Deep Visual-Semantic Alignments for Generating Image Description, 2014

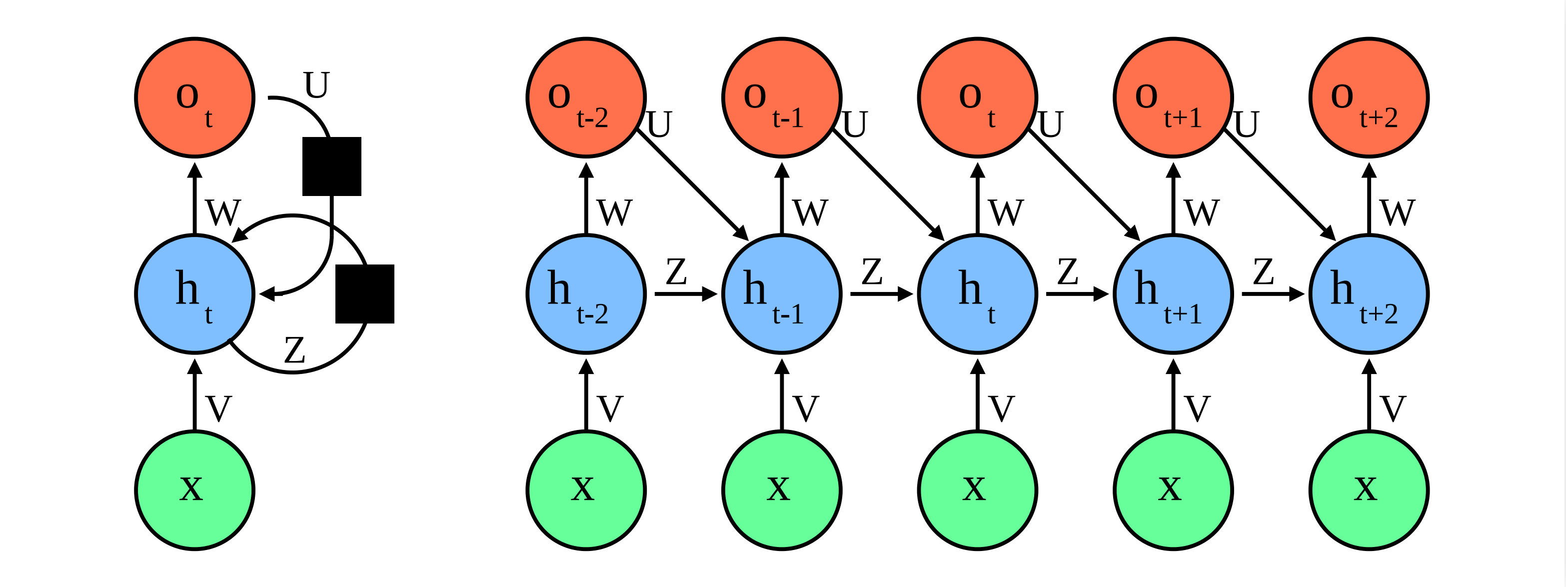
Encoder-Decoder RNN

- One RNN can map from a sequence to a fixed-length vector
  - E.g. the final output



Encoder-Decoder RNN

- Another RNN can map from a fixed-length vector to a sequence



Encoder-Decoder RNN

- One RNN can map from a sequence to a fixed-length vector
- Another RNN can map from a fixed-length vector to a sequence
- This allows mapping from one sequence to a different sequence
  - Speech to text, translation, dialog, etc

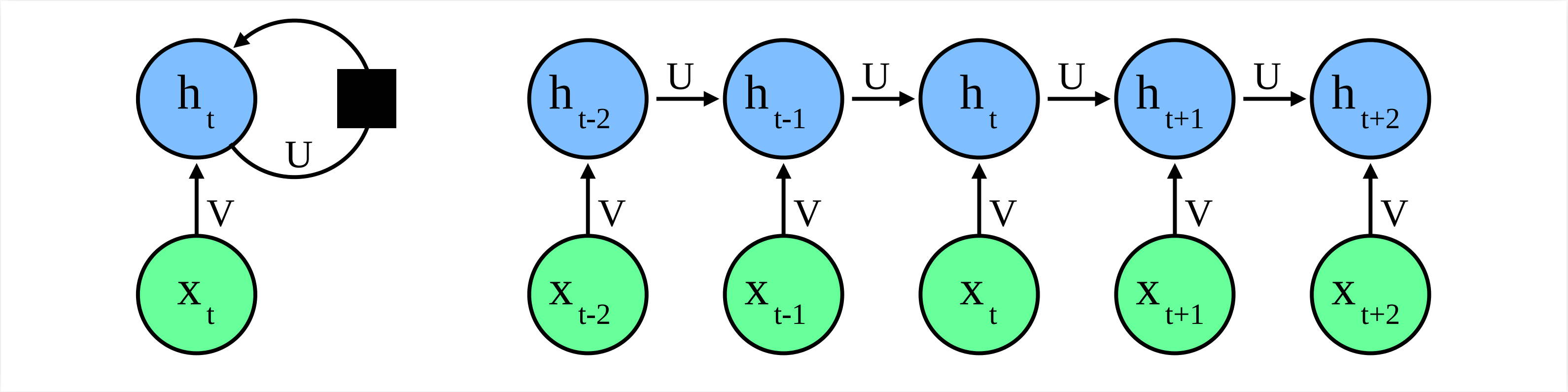
Recurrent Autoencoders

- This can be applied to obtain as output the same as the input
- Unsupervised learning of fixed-length representations of sequences

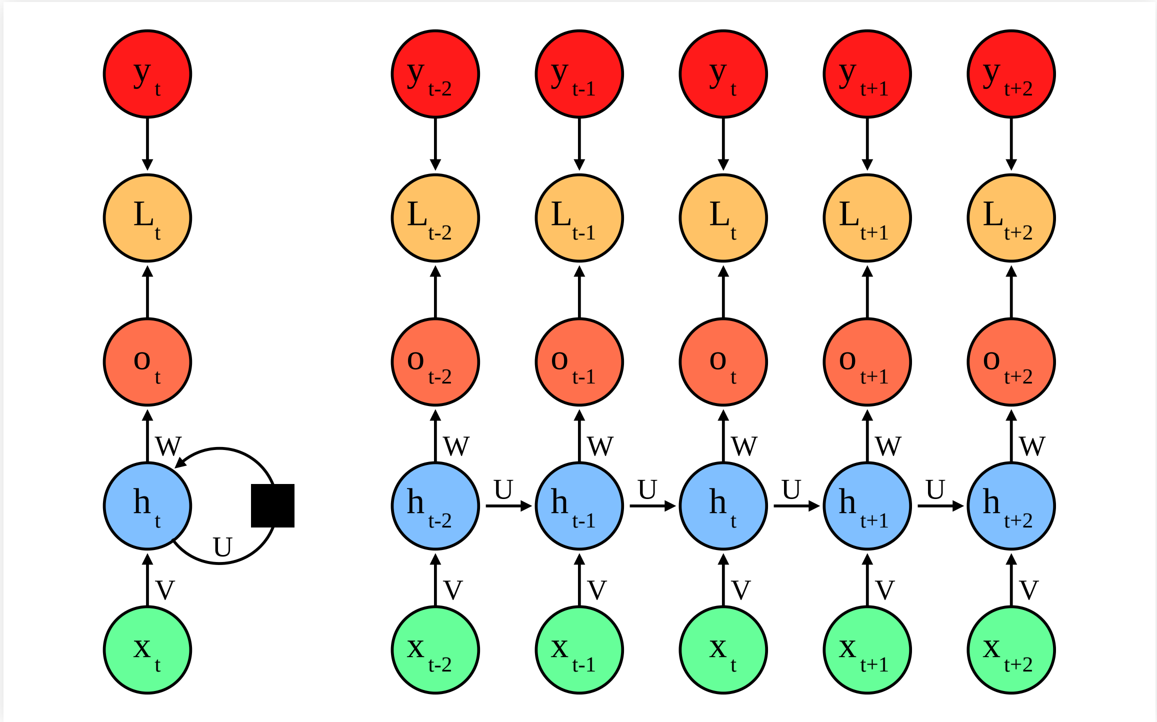
Backpropagation Through Time

Training a RNN

- To train a RNN we need to backpropagate the error computing the gradients
- But we need to do this over the multiple time steps:
  - The state of the network is changing
  - Different inputs
  - Shared weights
- This requires backpropagation through time (BPTT)



BPTT, conceptually



- If we unfold the network this is just normal backpropagation with
  - Aside from the shared weight matrices U, V and W

BPTT, in practice

- The `tf.while_loop`

- By default Keras uses a symbolic loop to activate and backpropagate gradients
- Slower, but needs less memory

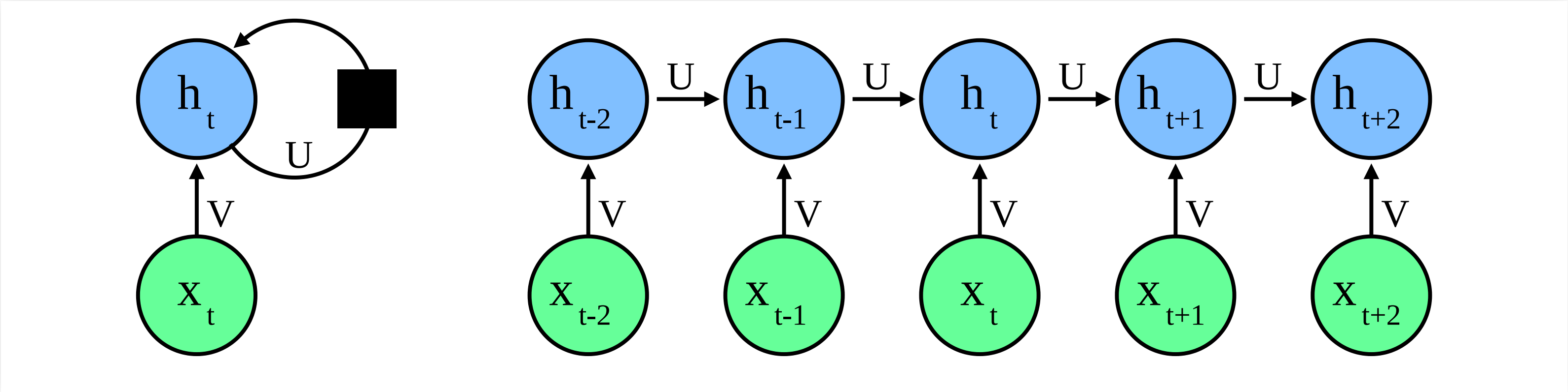
- Option `unroll = True`

- Unfolds the network into a feed-forward graph
- Faster but better for short sequences (otherwise lots of memory)

BPTT

Truncated BPTT

- If our sequence has length of k we would process k time steps forward and then backpropagate for k time steps
- This takes longer, increases risk of vanishing or exploding gradients and just updates the same weights many times



FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE NOVA DE LISBOA

BPTT

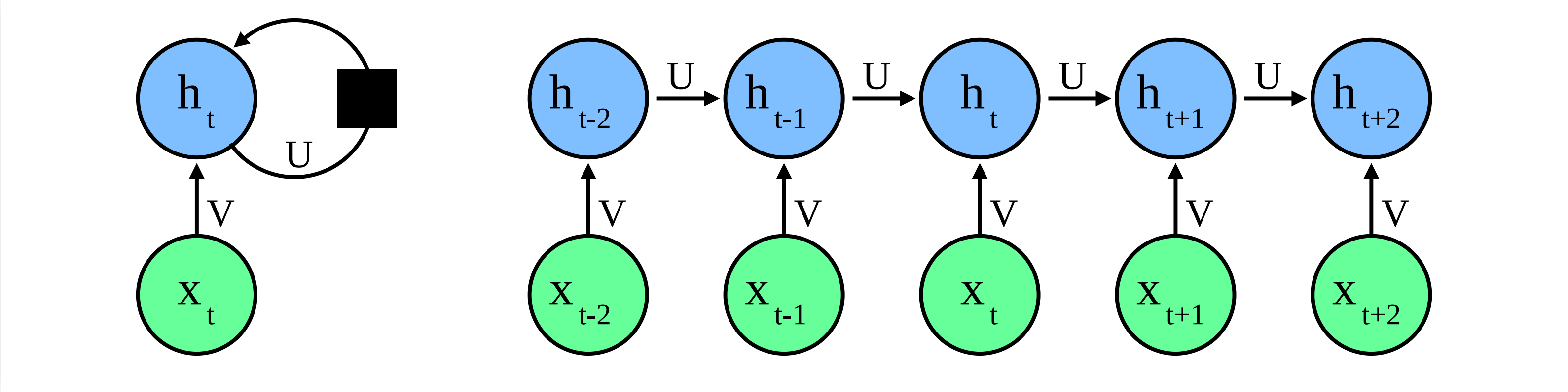
Truncated BPTT

- If our sequence has length of k we would process k time steps forward and then backpropagate for k time steps
- This takes longer, increases risk of vanishing or exploding gradients and just updates the same weights many times
- It is more efficient to truncate this process:
  - Forward pass with k_1 steps, the length of the relevant sequence
  - Backpropagate for $k_2 < k_1$ steps
- With Keras we can do this by:
  - Splitting the sequences to length k_2
  - Use `stateful=True` to keep state between batches, and `reset_states()`
  - (Hack: keep the right order within batches, as the state resets after a batch)

Long-Term dependencies

Long-Term dependencies

- RNN compose the same function many times



- This makes it unstable
  - Feed-forward networks can compensate using different parameters in different layers
  - With RNN it is easy to explode or vanish gradients and values
- Dependencies get exponentially weaker as time interval increases

Long-Term dependencies

Some solutions

- Reservoir methods:
  - Hidden states are computed by a recurrent network with non-trainable weights
  - Weights are initialized at random in ranges that optimize stability
  - Only the weights from the hidden state to the output are trained
- Skip connections
  - "Jump" through time intervals
- Leaky units
  - Keep linear self-connections, retaining part of previous activations
- Removing Connections
  - Keep connections over larger time intervals, remove length one connections
- Gated units

Gated recurrent unit

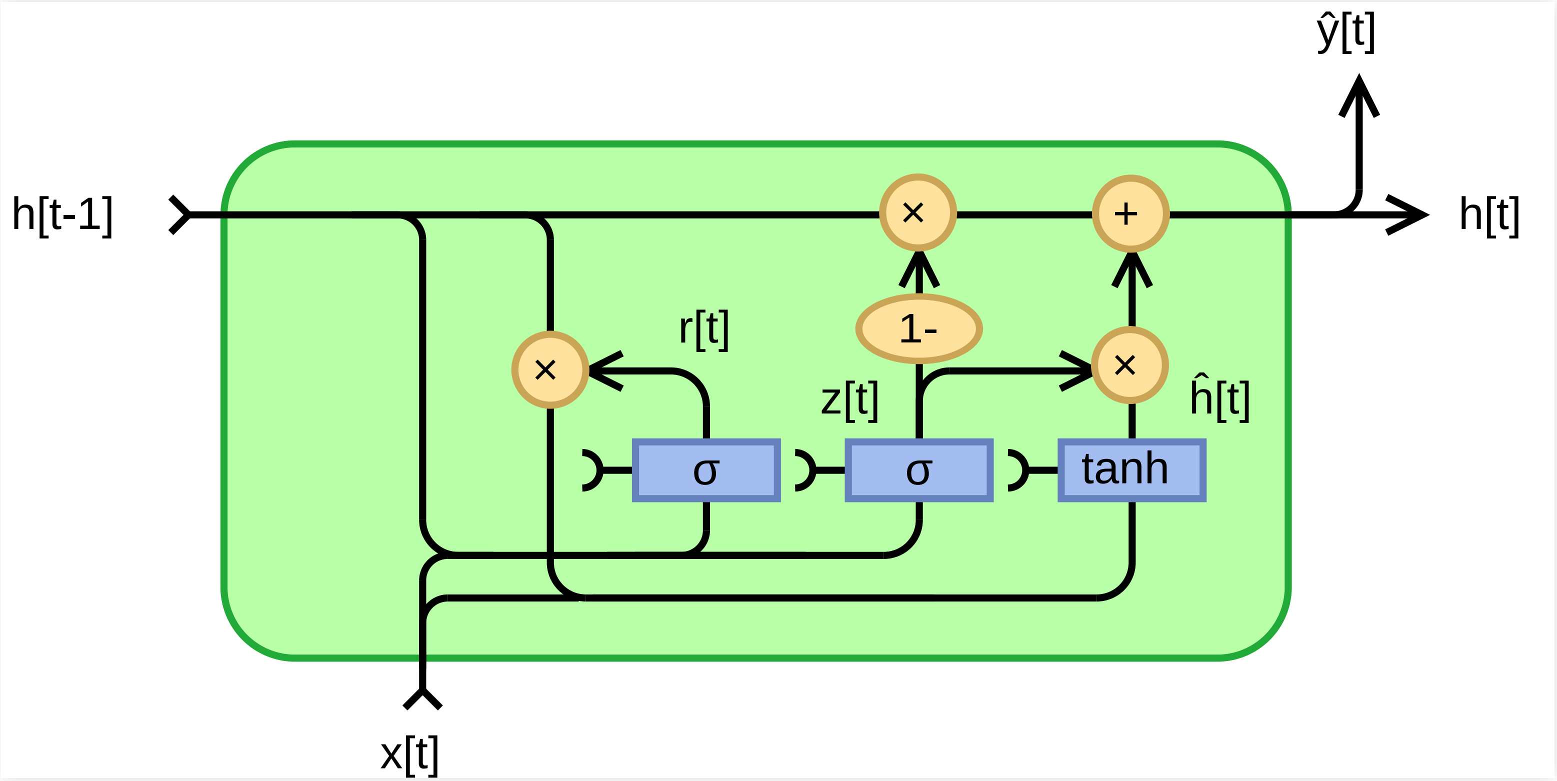


Image by Jeblad, CC BY-SA 4.0

Gated units, GRU

- Update gate: how much to pass from previous to future state

$$z_t = \sigma\left(W_z x_t + U_z h_{t-1} + b_z\right)$$

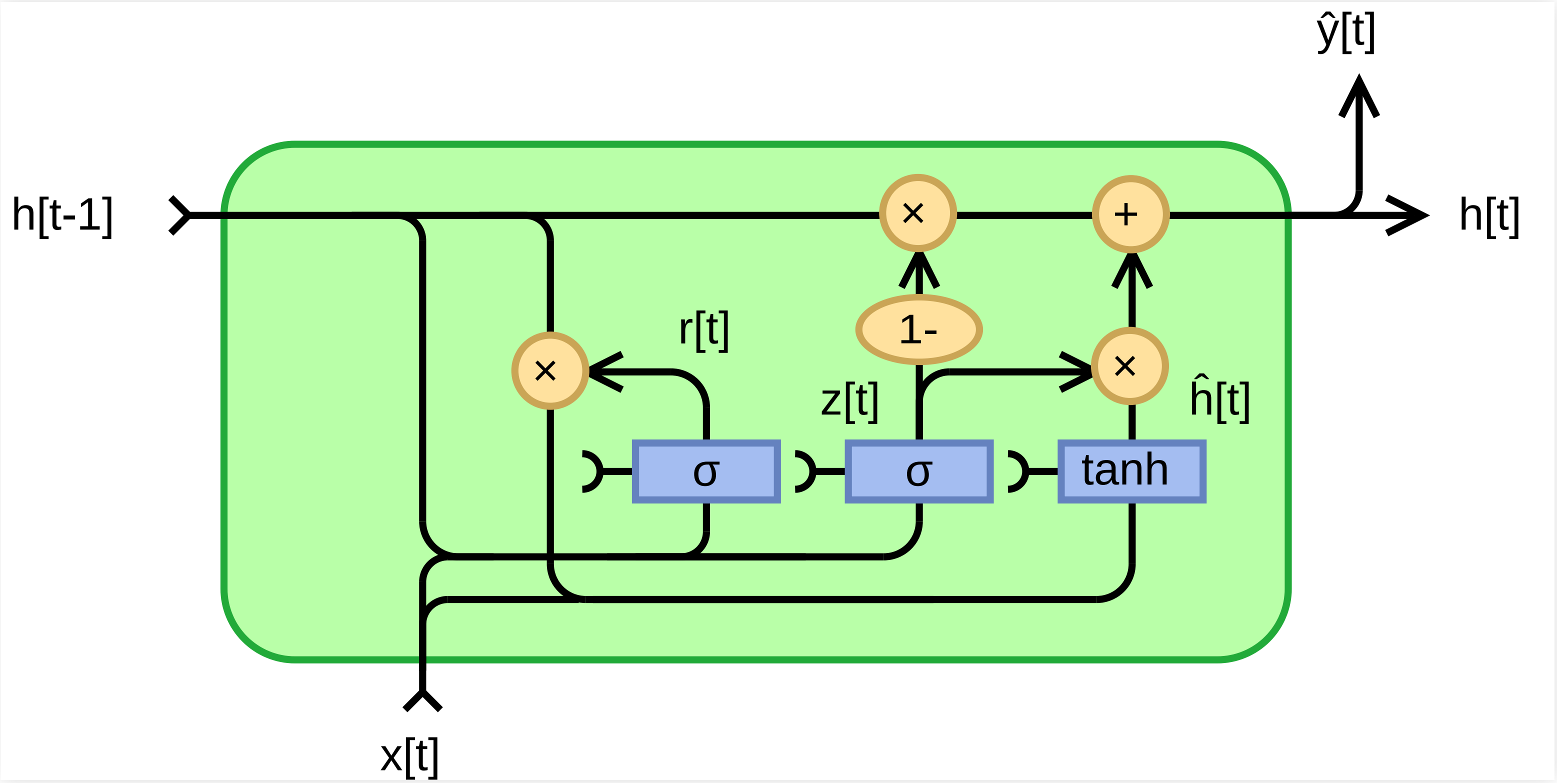


Image by Jeblad, CC BY-SA 4.0

Gated units, GRU

- Reset gate: how much to "forget" from previous state

$$r_t = \sigma\left(W_r x_t + U_r h_{t-1} + b_r\right)$$

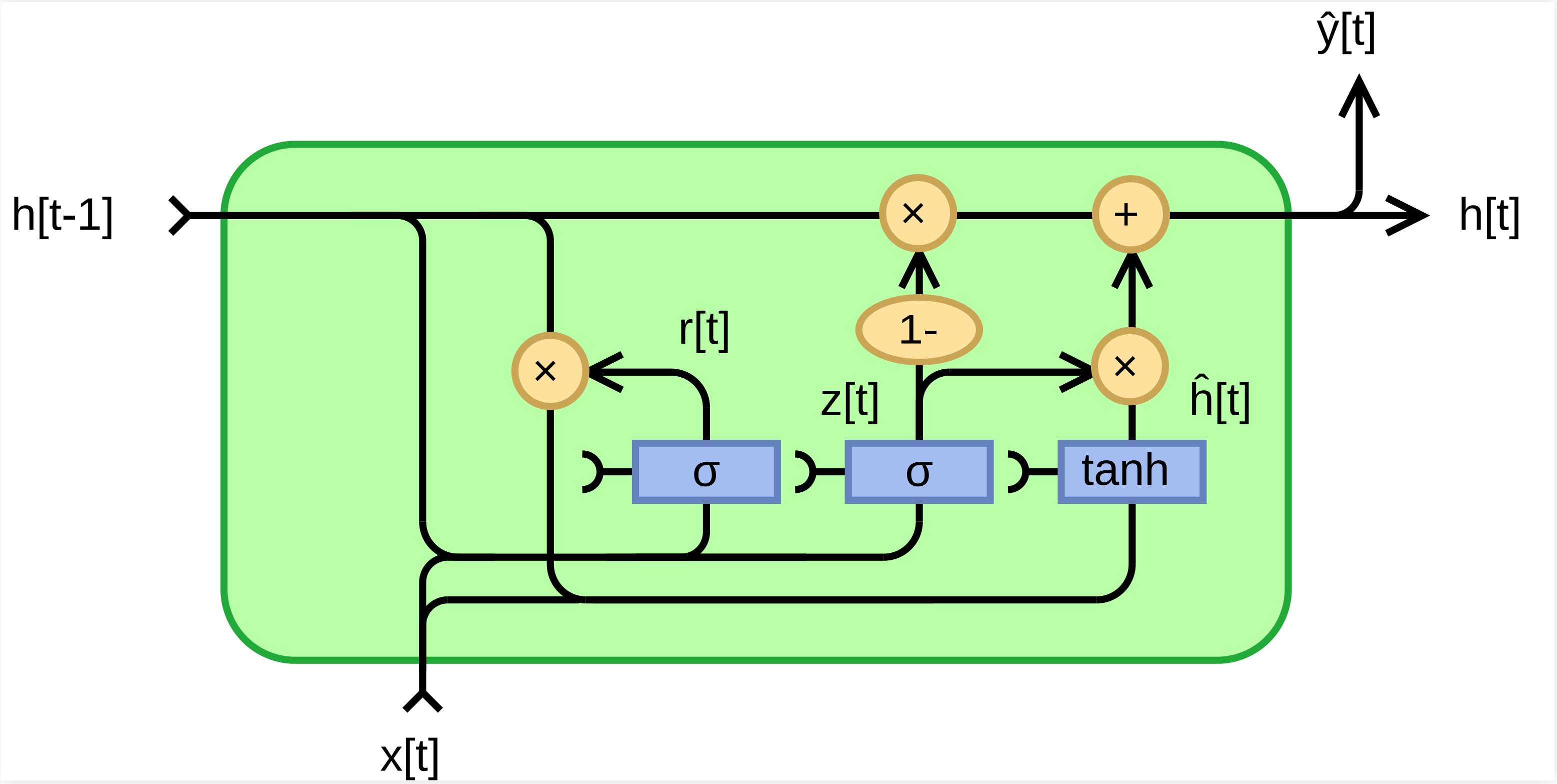
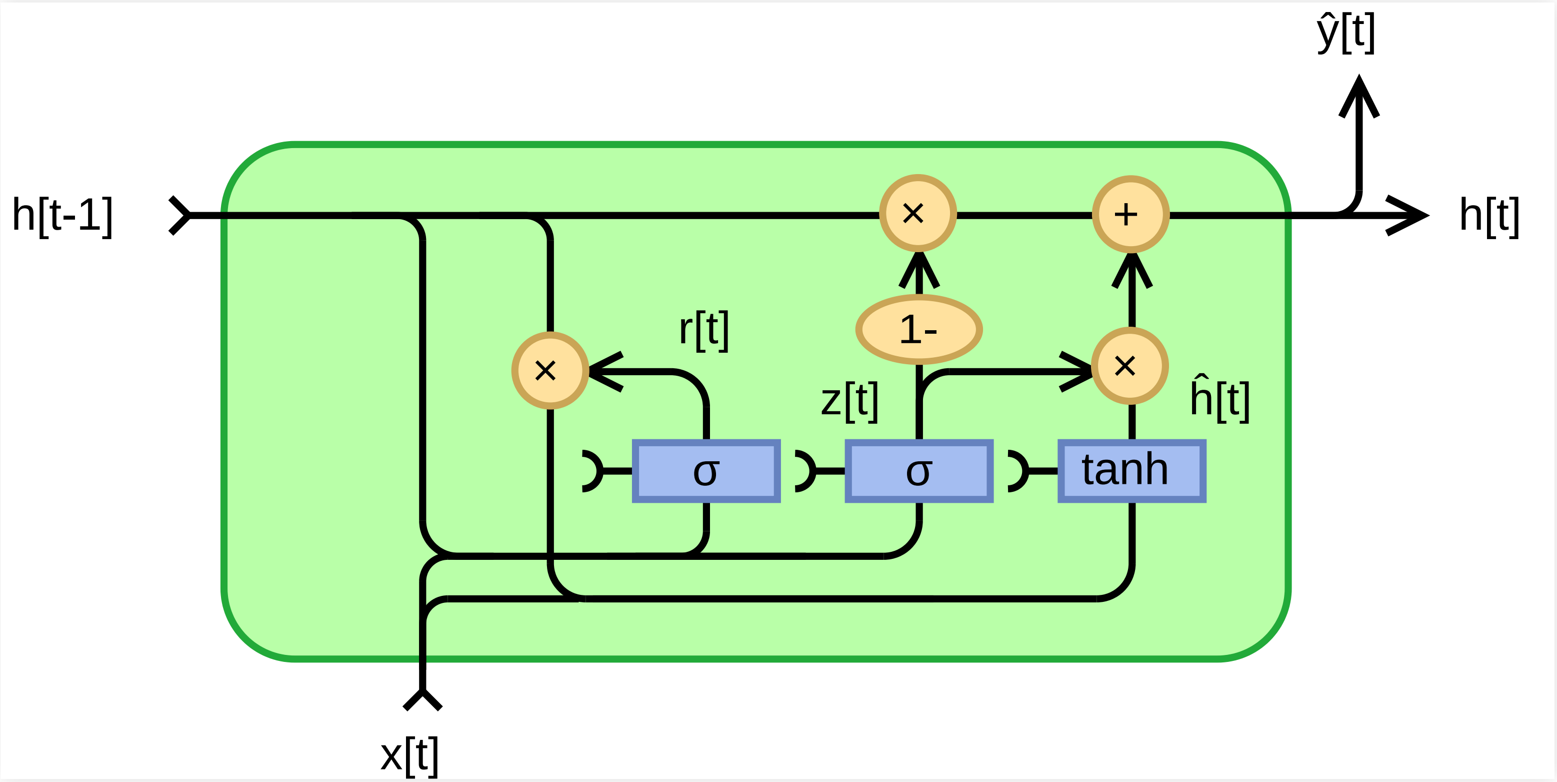


Image by Jeblad, CC BY-SA 4.0

Gated units, GRU

- Candidate output: current memory

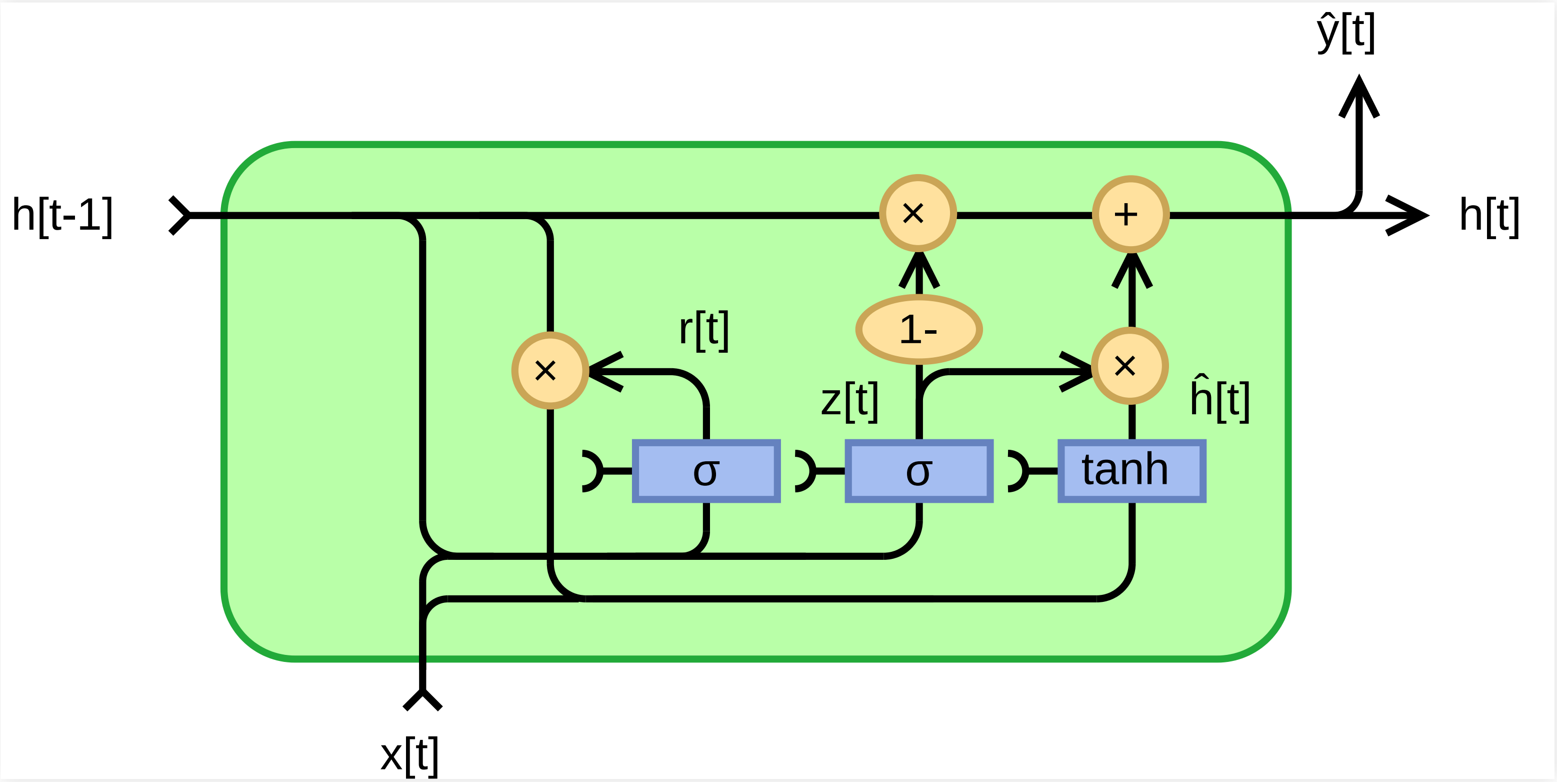
$$\hat{h}_t = \tanh(Wx_t + r_t \odot Uh_{t-1} + b_h)$$



Gated units, GRU

- State and output: what comes out of the unit

$$h_t = z_t \odot \hat{h}_t + (1 - z_t) \odot h_{t-1}$$



Gated units, GRU

- Controls how much "memory" is preserved
- Avoids vanishing gradients using linear transformations

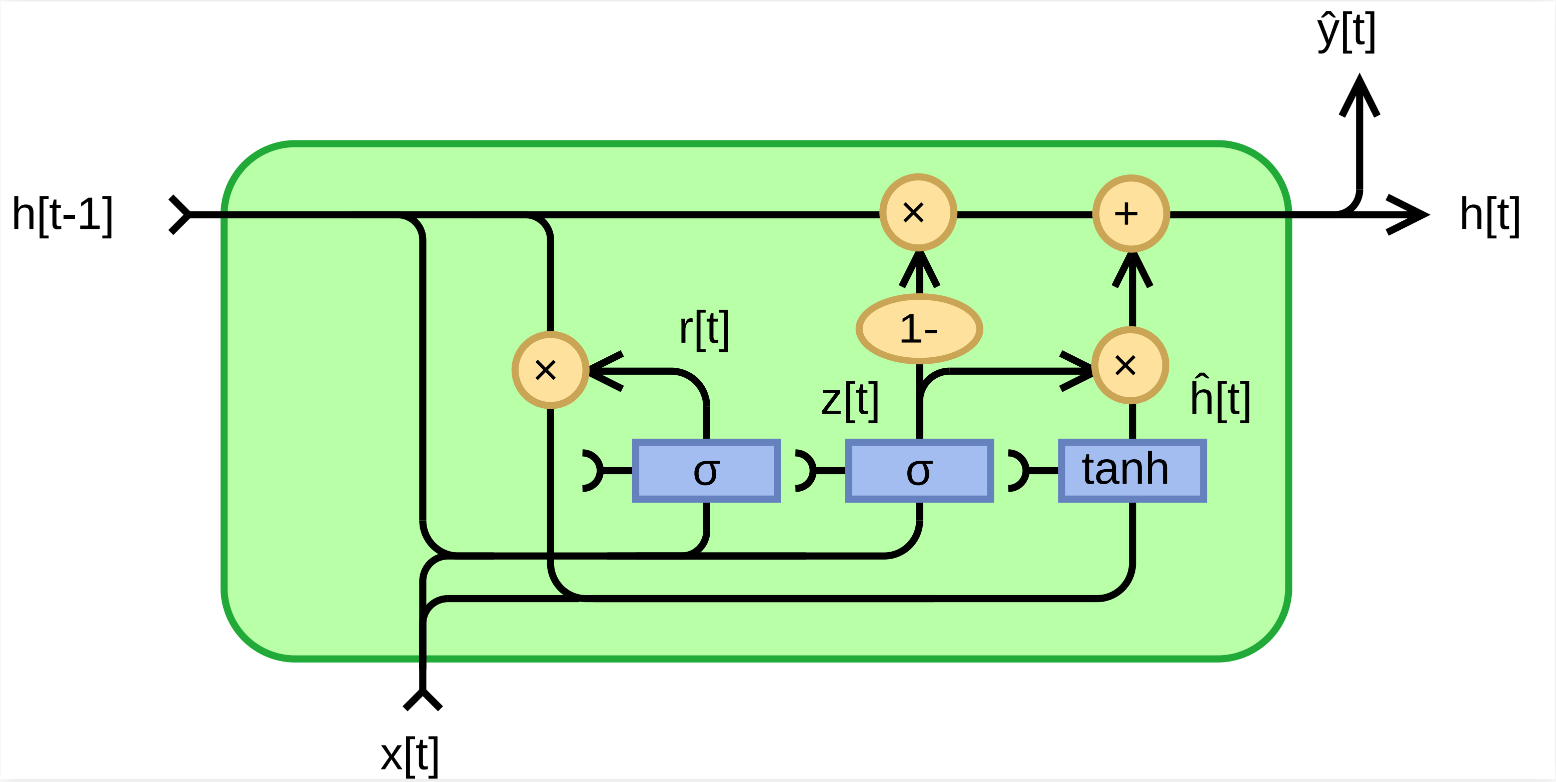


Image by Jeblad, CC BY-SA 4.0

Long Short Term Memory cell

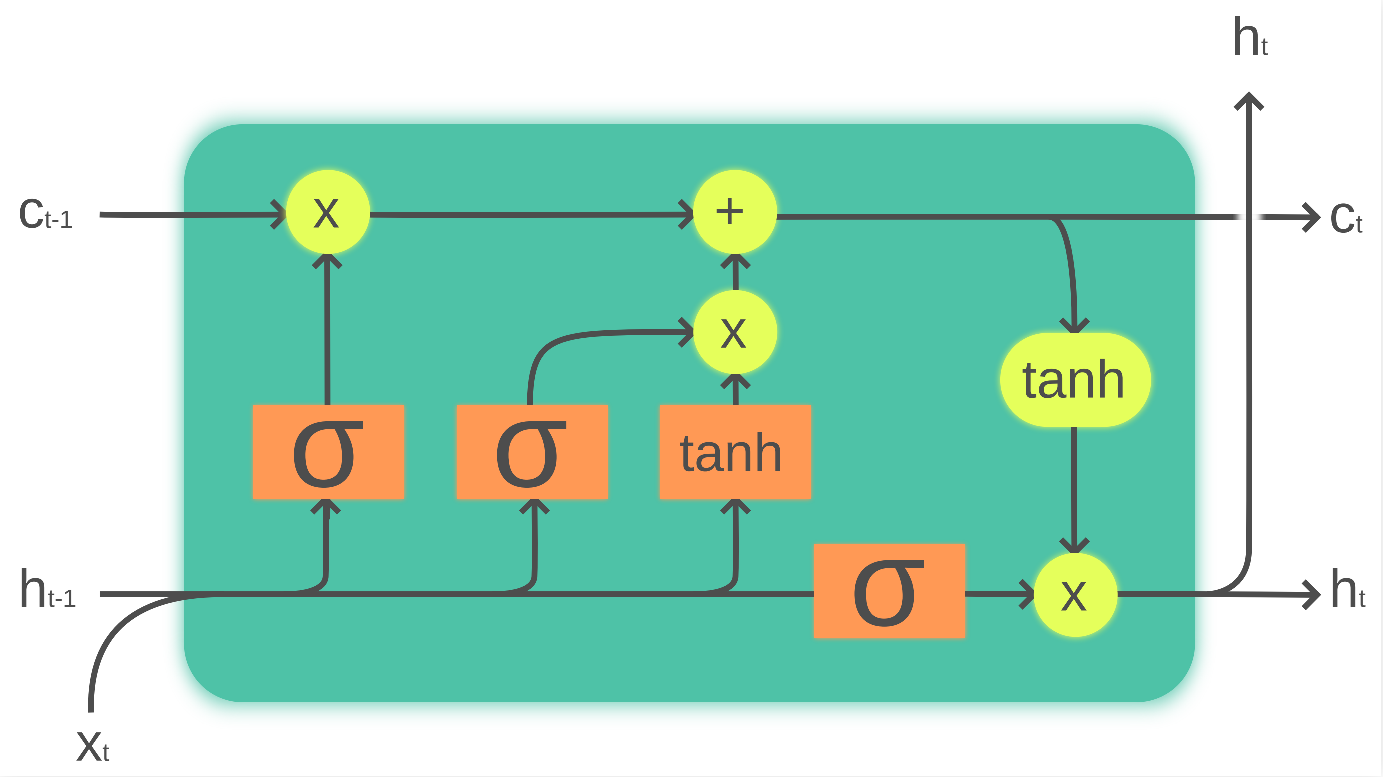


Image by Guillaume Chevalier, CC BY-SA 4.0

Gated units, LSTM

- Like GRU, cell state has only linear transformations

$$c_t = i_t \odot \hat{c}_t + f_t \odot c_{t-1}$$

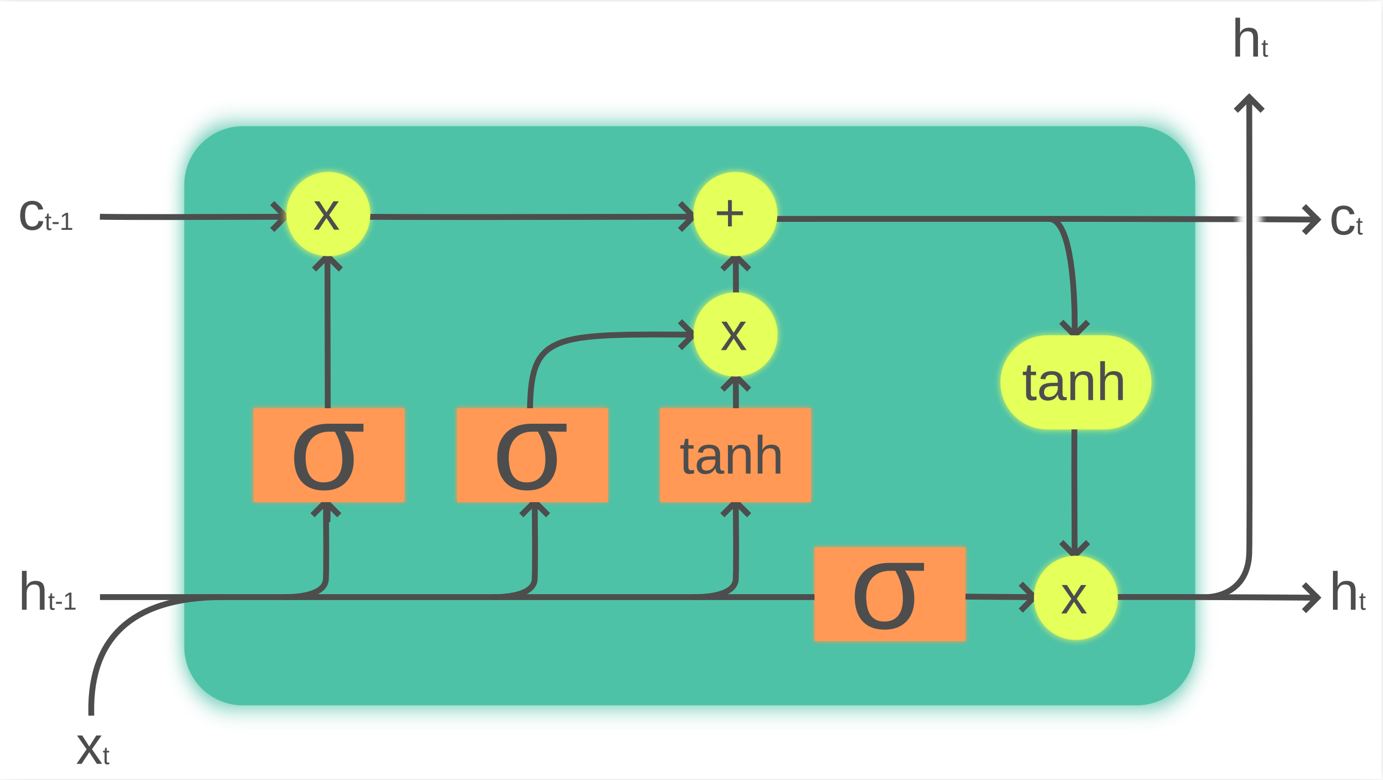
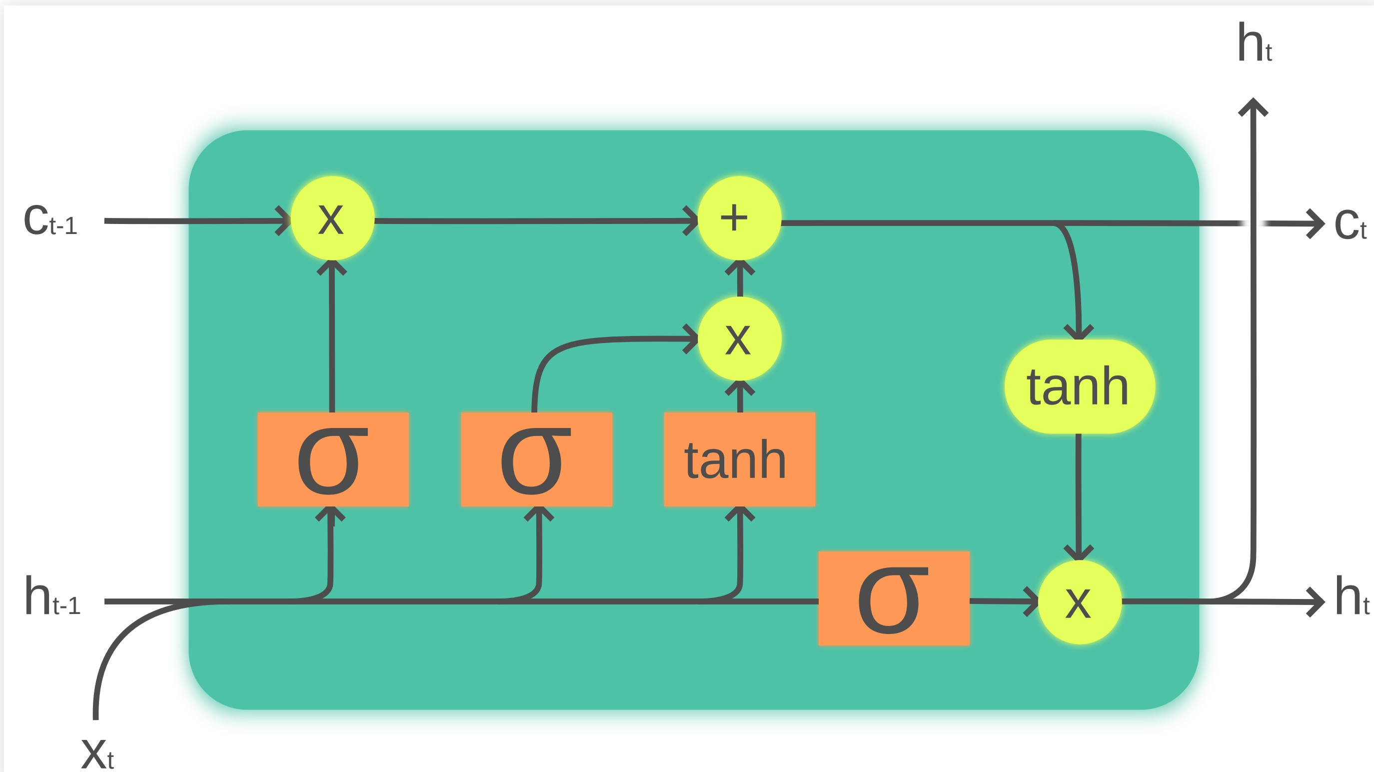


Image by Guillaume Chevalier, CC BY-SA 4.0

Gated units, LSTM

- Forget gate: how much to "forget" from previous state

$$f_t = \sigma\left(W_f x_t + U_f h_{t-1} + b_f\right)$$



Gated units, LSTM

- Input gate: how much of the input to store in current state

$$i_t = \sigma\left(W_i x_t + U_i h_{t-1} + b_i\right)$$

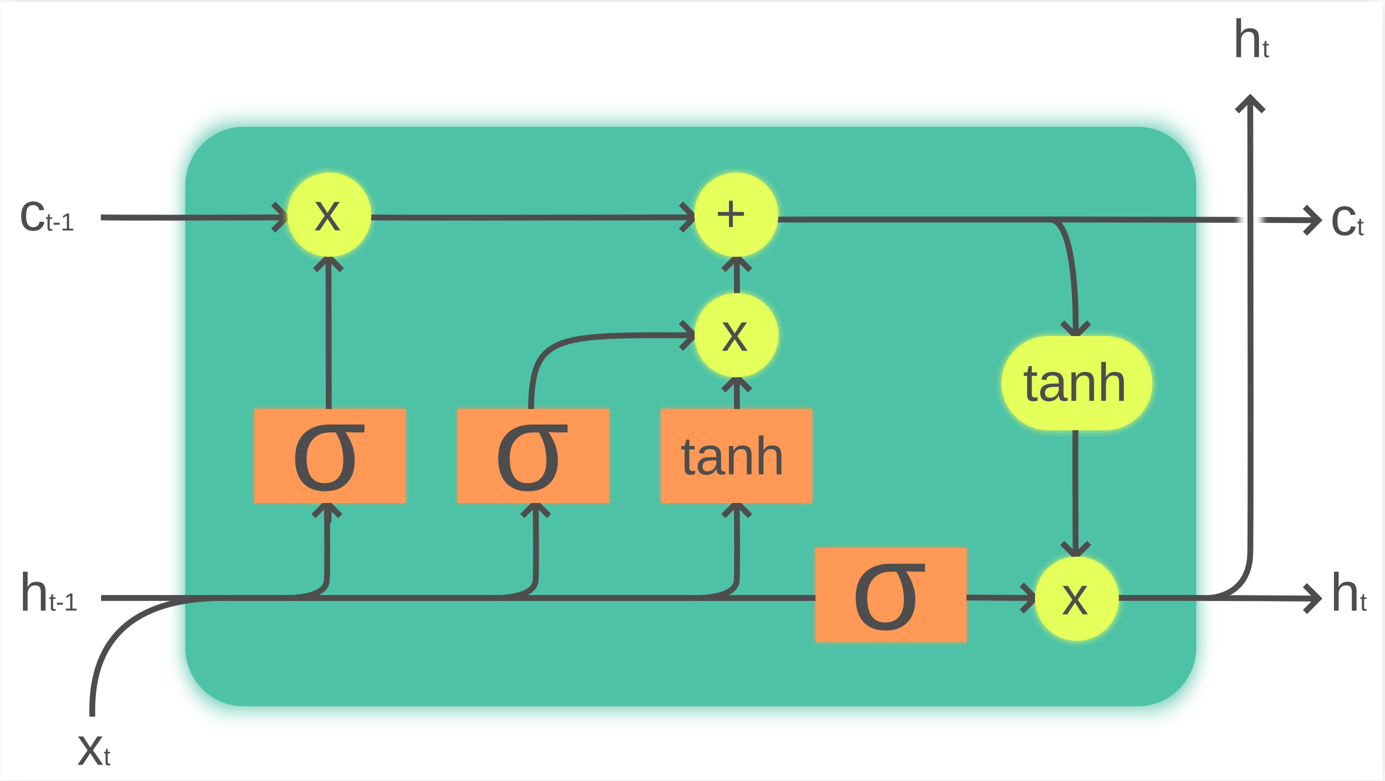


Image by Guillaume Chevalier, CC BY-SA 4.0

Gated units, LSTM

- Candidate state: intermediate step for updating state

$$\hat{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c)$$

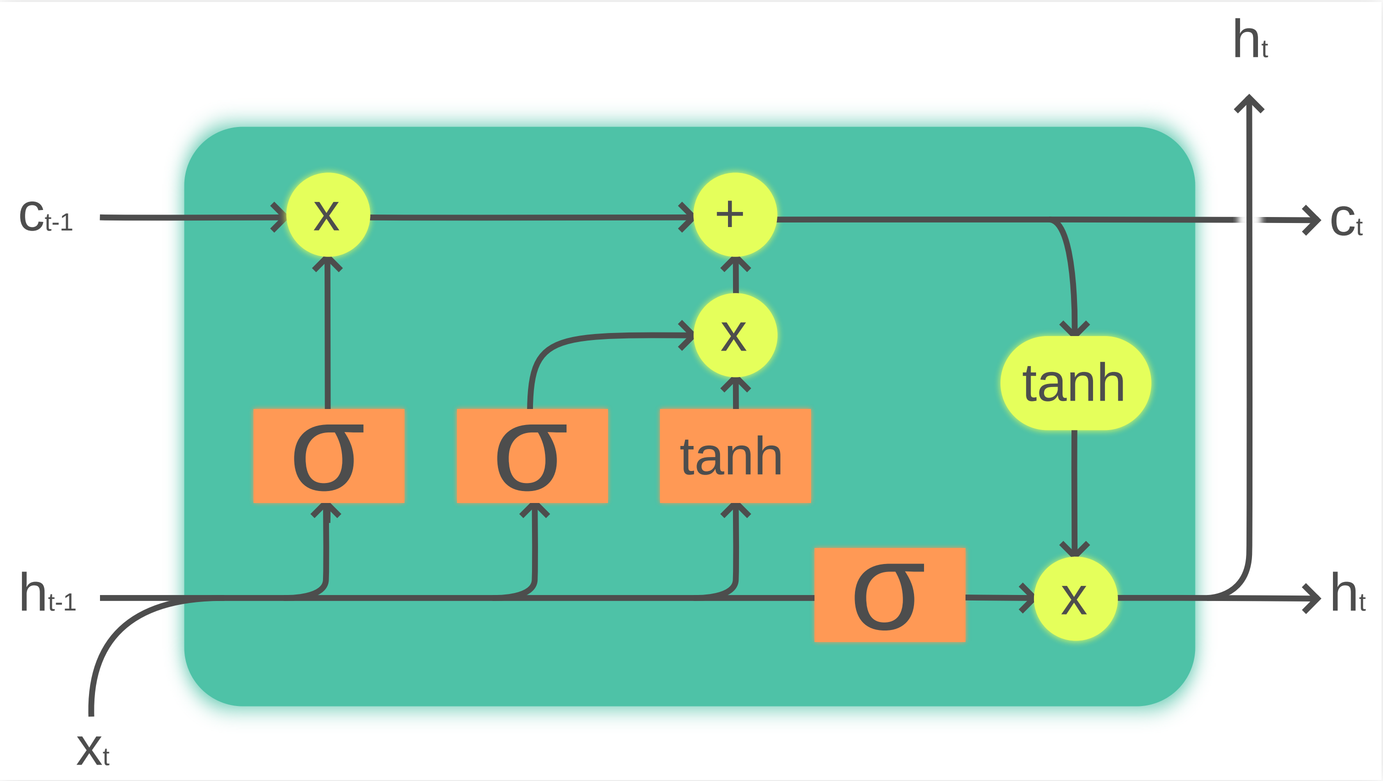


Image by Guillaume Chevalier, CC BY-SA 4.0

FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE NOVA DE LISBOA

Gated units, LSTM

- Cell state: linear memory

$$c_t = f_t \odot c_{t-1} + i_t \odot \hat{c}_t$$

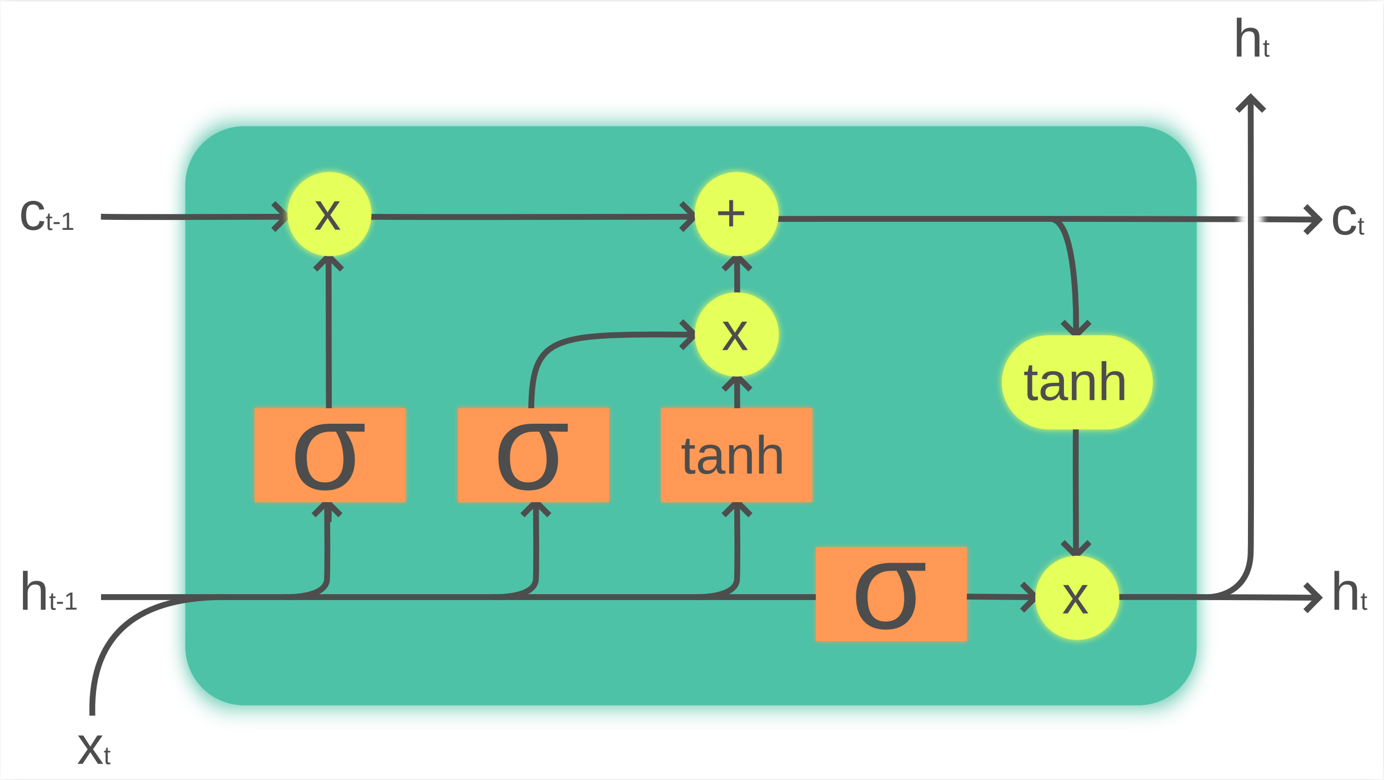


Image by Guillaume Chevalier, CC BY-SA 4.0

Gated units, LSTM

- Output gate: how much of the cell state will output

$$o_t = \sigma\left(W_o x_t + U_o h_{t-1} + b_o\right)$$

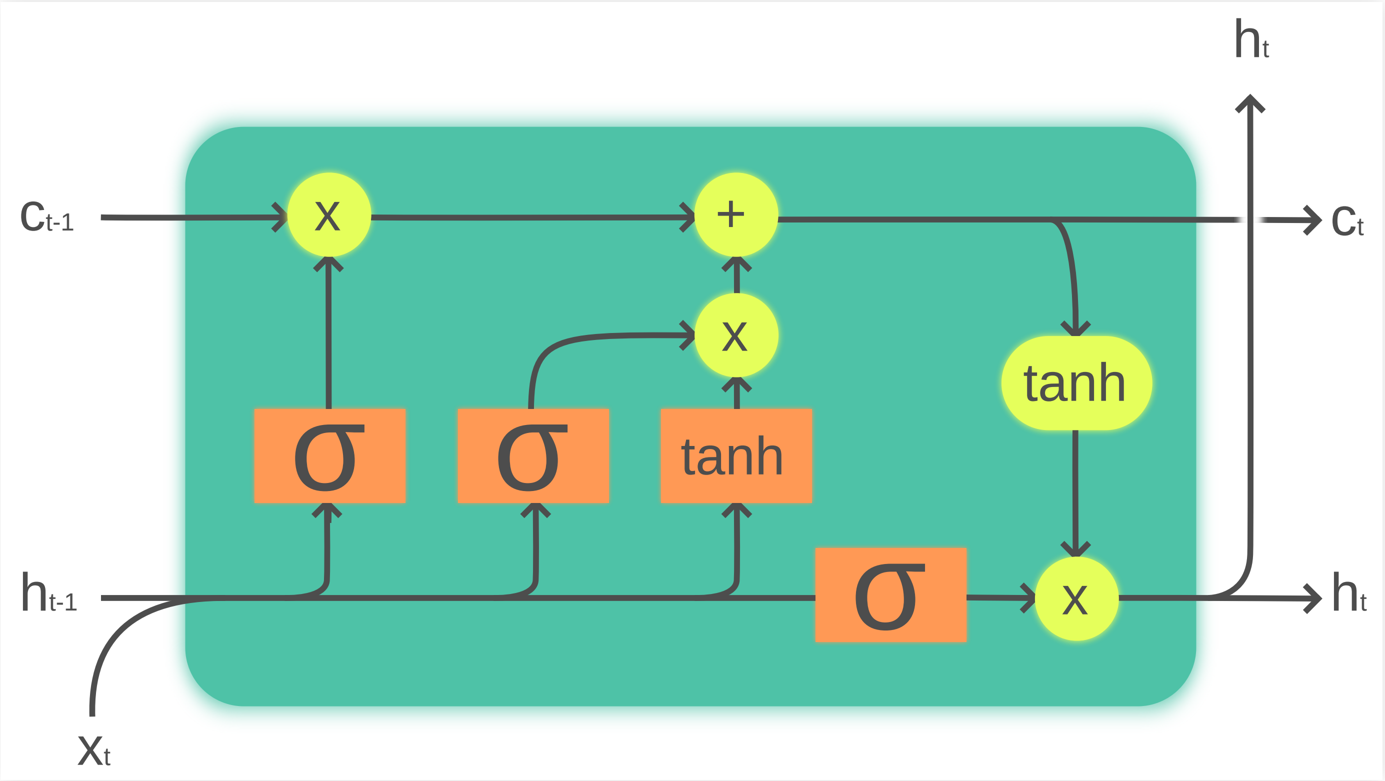


Image by Guillaume Chevalier, CC BY-SA 4.0

Gated units, LSTM

- Output:

$$h_t = o_t \odot \tanh(c_t)$$

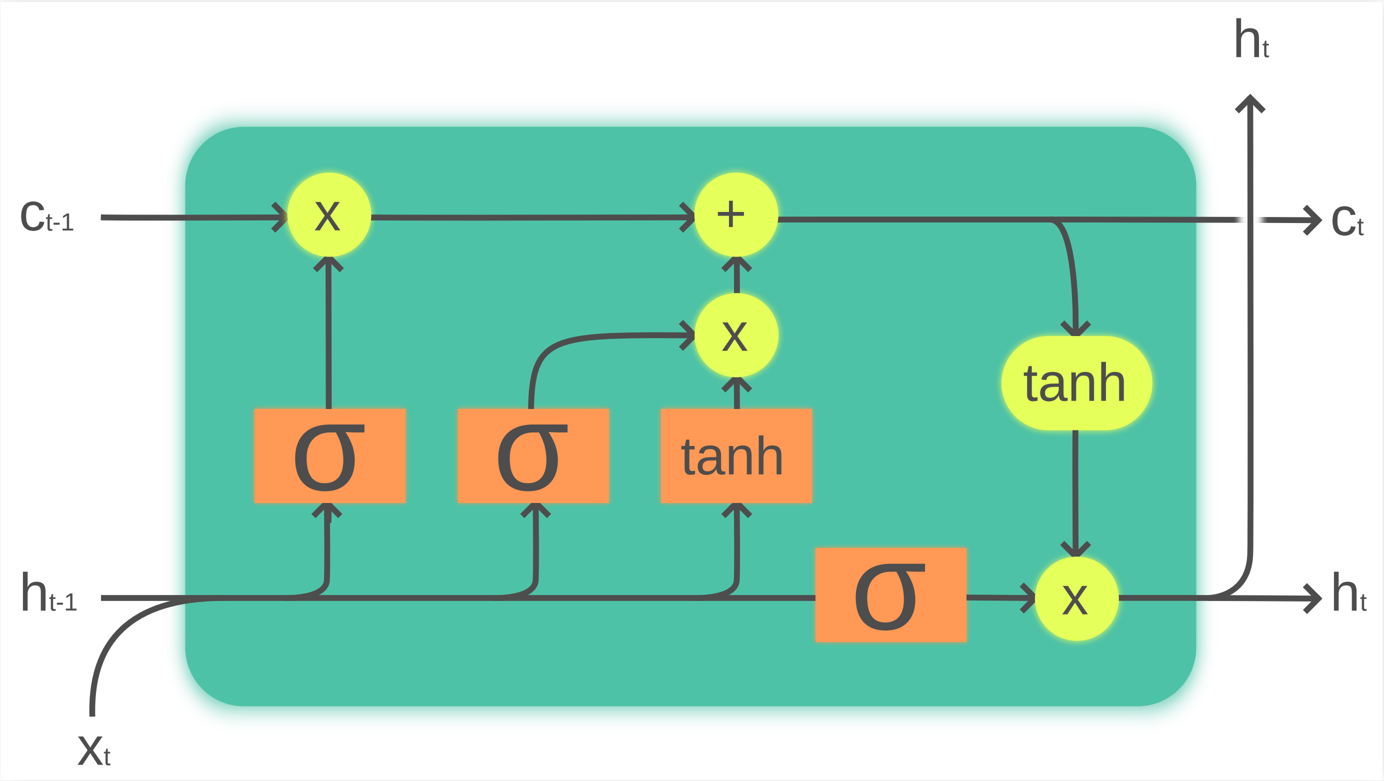


Image by Guillaume Chevalier, CC BY-SA 4.0

Gated units:

- Use gates (sigmoid) to "cut" flows
- Linearly transformed "memory" to avoid vanishing gradients
- GRU:
  - Output/hidden state, linearly transformed
  - Reset and update gate
- LSTM:
  - Forget, input and output gates
  - Cell state, linearly transformed
  - Output, from cell state, input and previous output

Summary

Summary

- Unfolding BPTT
- Different architectures and applications
  - Mapping from and to sequences
- Problem of learning long term dependencies
- Gated RNN:
  - GRU and LSTM

Further reading

- Goodfellow et.al, Deep learning, Chapter 10