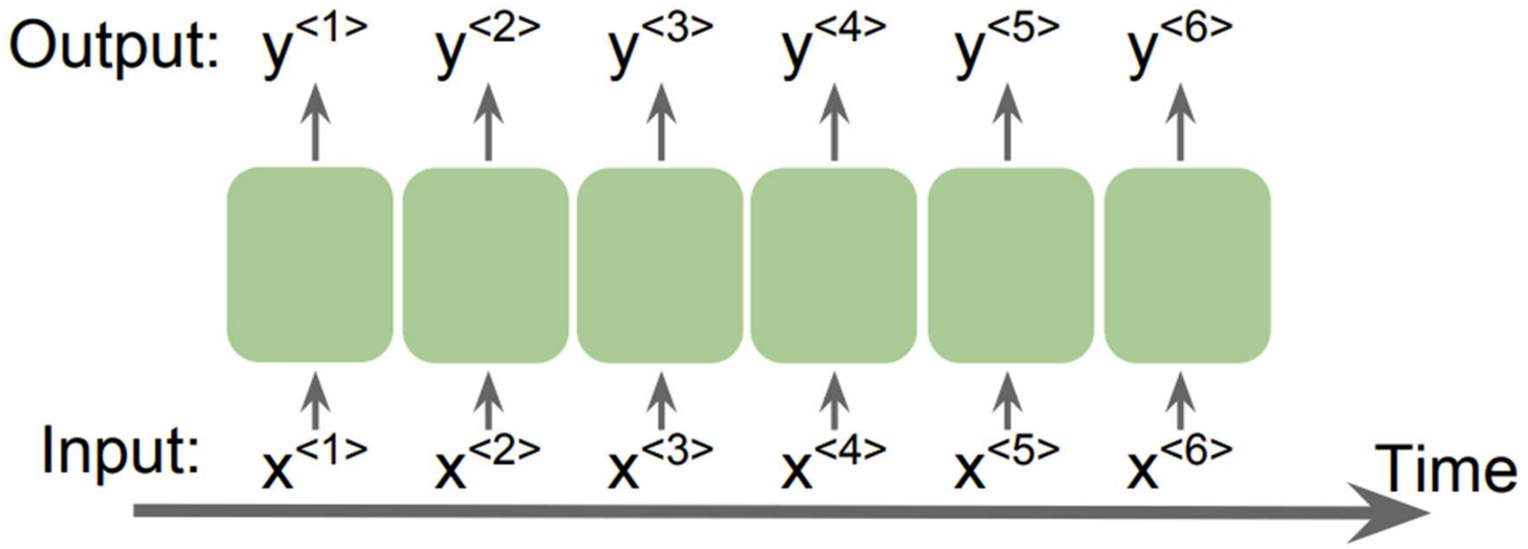
RNN : Recurrent Neural Network

Sourav Karmakar

souravkarmakar29@gmail.com

Sequence Data: order matters

The movie my friend has **not** seen is good
The movie my friend has seen is **not** good



Output: $y^{<1>}$ $y^{<2>}$ $y^{<3>}$ $y^{<4>}$ $y^{<5>}$ $y^{<6>}$
Input: $x^{<1>}$ $x^{<2>}$ $x^{<3>}$ $x^{<4>}$ $x^{<5>}$ $x^{<6>}$
Time

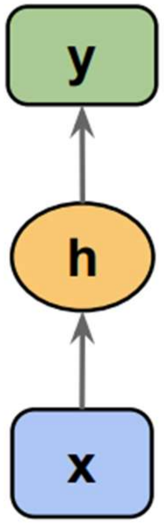
Applications: Working with Sequential data

- Text classification
- Speech recognition (acoustic modelling)
- Language translation

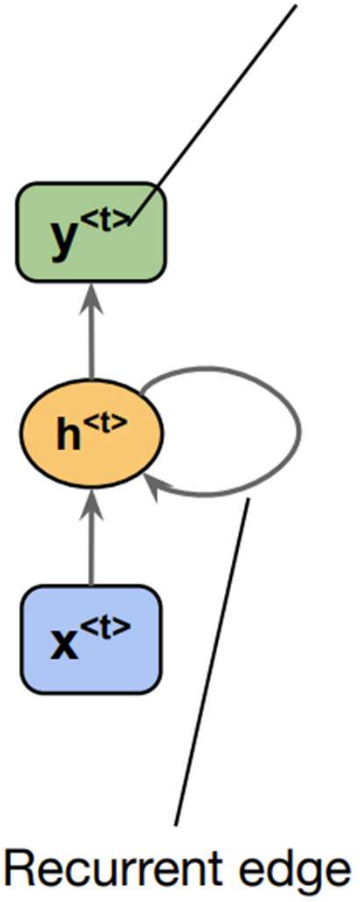
. . . and many more

Recurrent Neural Network: Overview

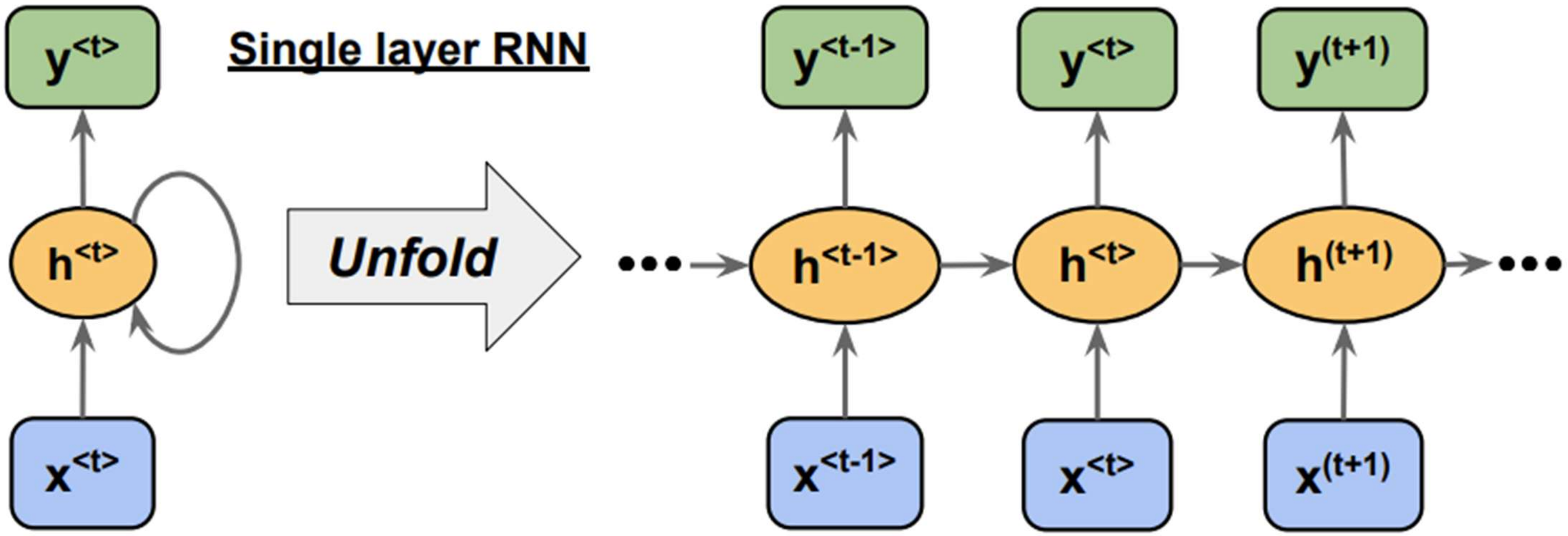
Networks we used previously: also called feedforward neural networks



Recurrent Neural Network (RNN)

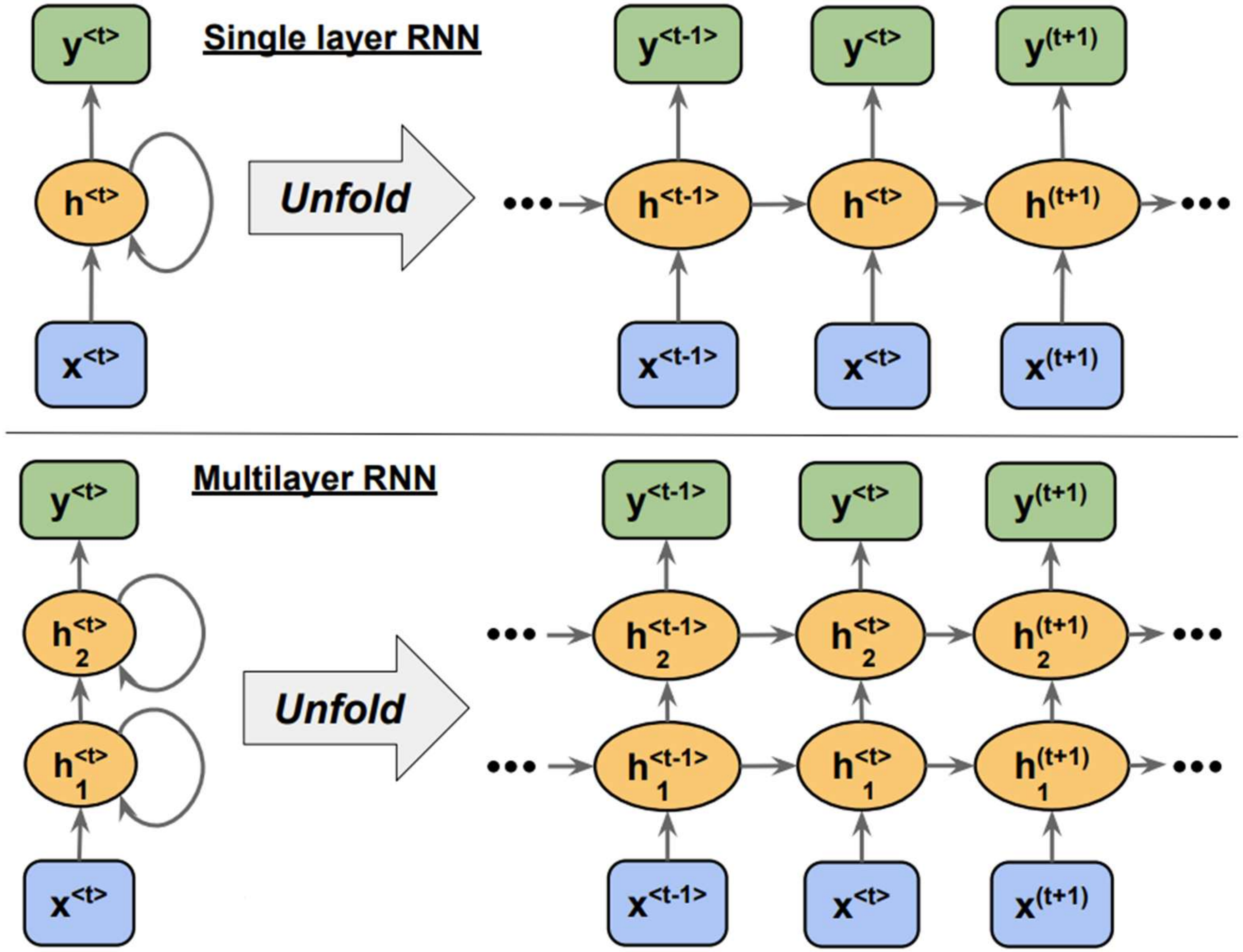


Recurrent Neural Network: Overview



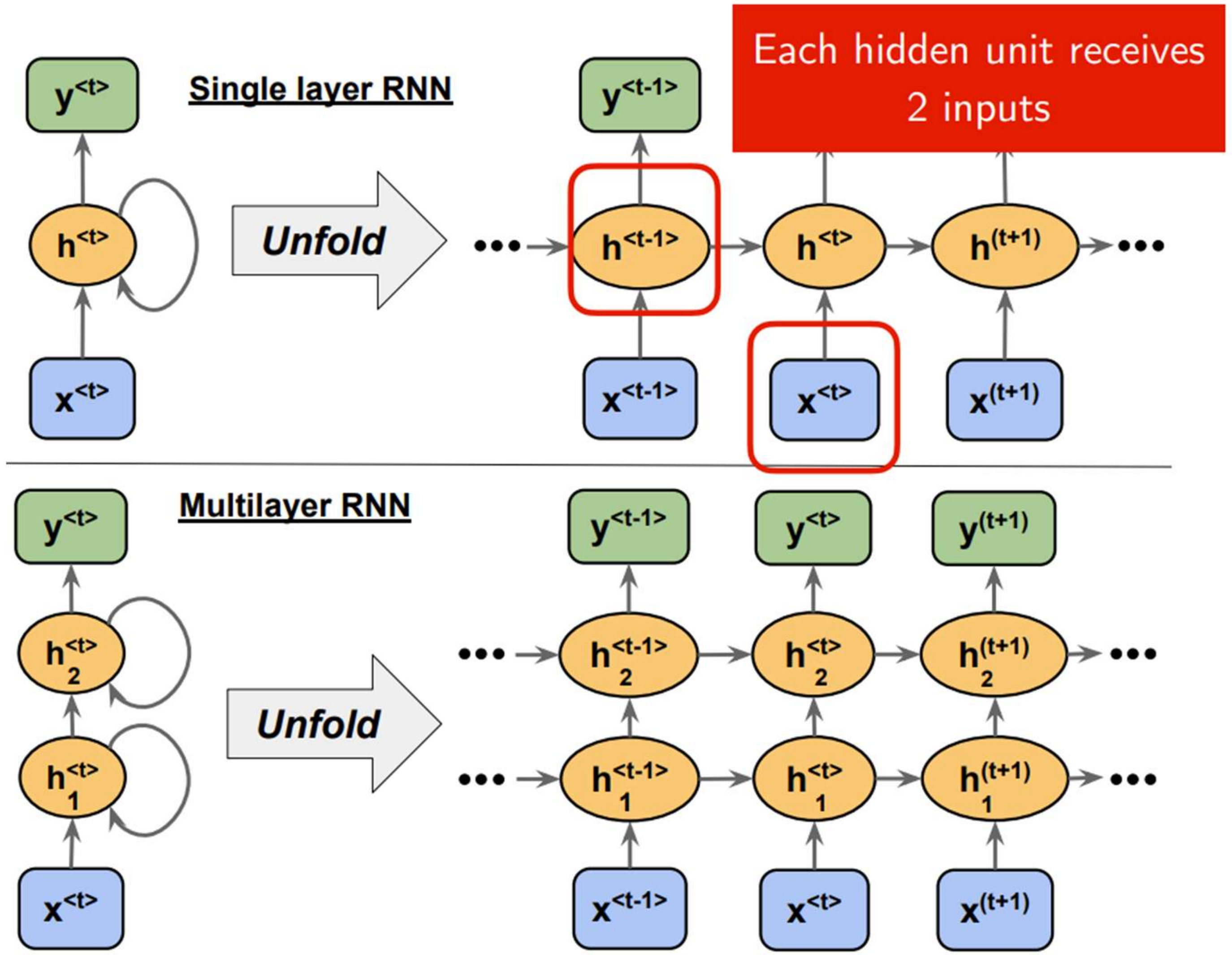
Time – unrolled version of RNN

Recurrent Neural Network: Overview



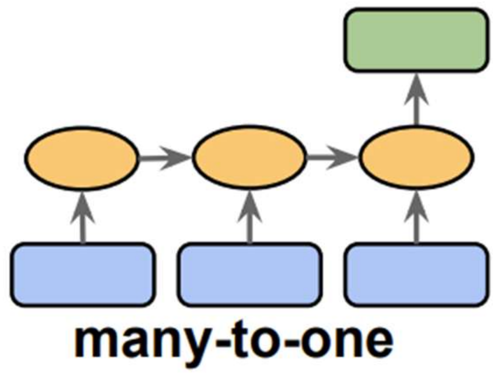
Single layer RNN
$y^{<t>}$
$h^{<t>}$
$x^{<t>}$
Unfold
$y^{<t-1>}$ $y^{<t>}$ $y^{(t+1)}$
$h^{<t-1>}$ $h^{<t>}$ $h^{(t+1)}$
$x^{<t-1>}$ $x^{<t>}$ $x^{(t+1)}$
Multilayer RNN
$y^{<t>}$
$h_2^{<t>}$
$h_1^{<t>}$
$x^{<t>}$
Unfold
$y^{<t-1>}$ $y^{<t>}$ $y^{(t+1)}$
$h_2^{<t-1>}$ $h_2^{<t>}$ $h_2^{(t+1)}$
$h_1^{<t-1>}$ $h_1^{<t>}$ $h_1^{(t+1)}$
$x^{<t-1>}$ $x^{<t>}$ $x^{(t+1)}$

Recurrent Neural Network: Overview



Different types of sequence modelling tasks

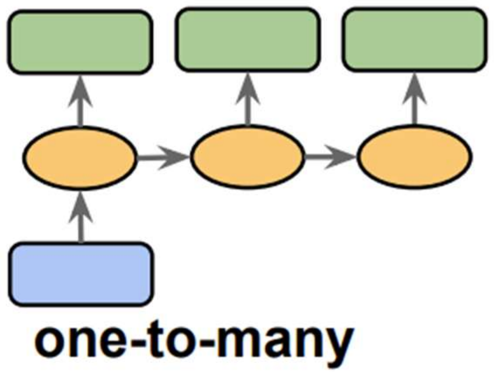
Many-to-one



The input data is a sequence, but the output is a fixed size vector (or scalar), not a sequence.

Example: sentiment analysis, the input is some text, and output is a class label.

One-to-many

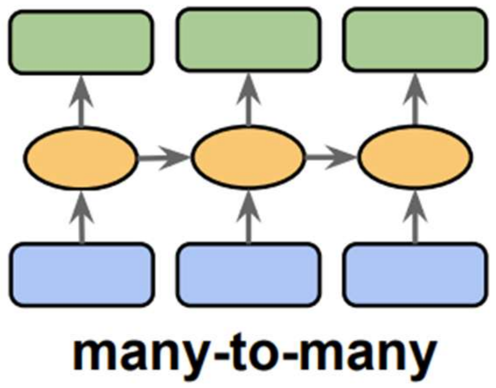


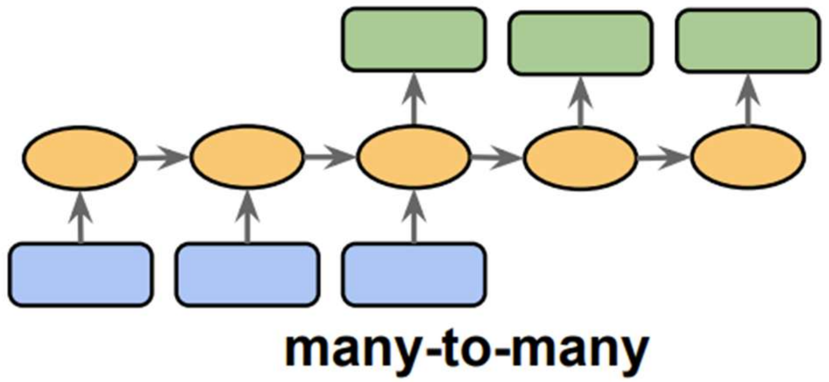
The input data is in a standard format (not a sequence), the output is a sequence.

Example: Image captioning, where the input us an image, the output is a text description of that image.

Different types of sequence modelling tasks

Many-to-many





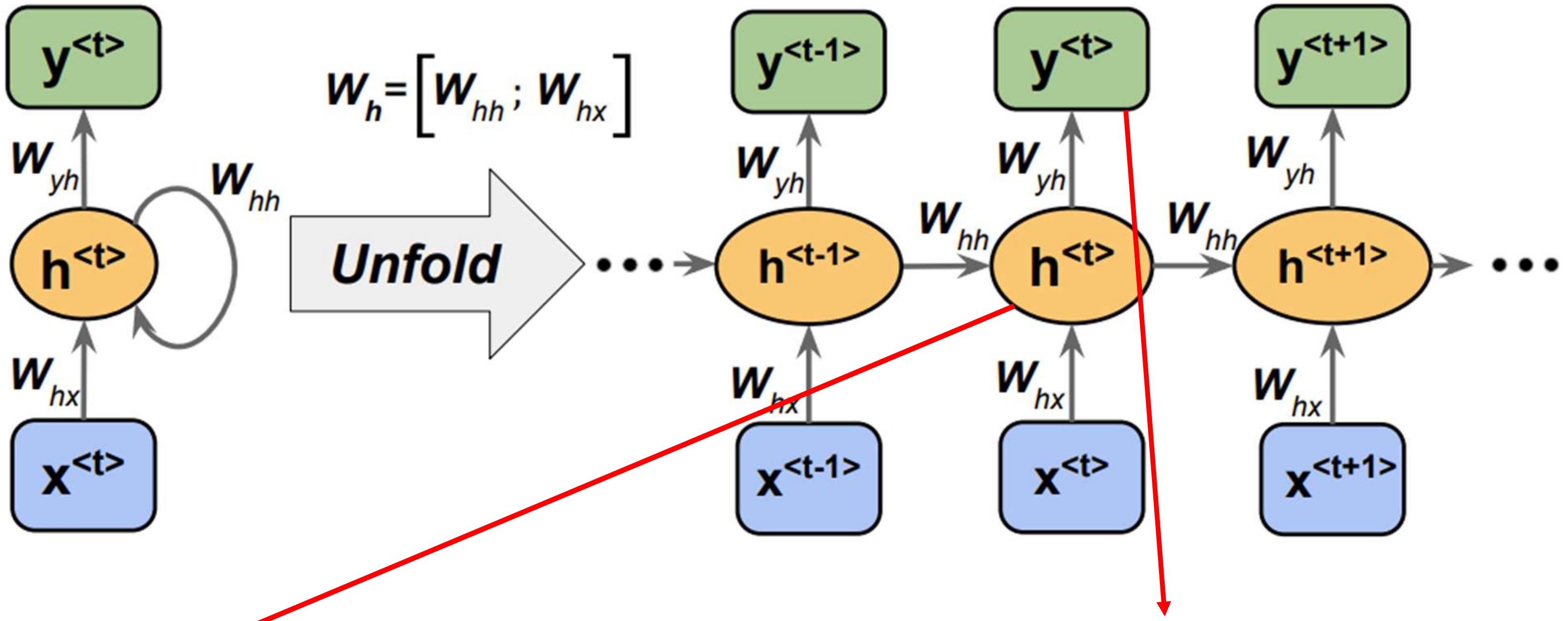
Both inputs and outputs are sequence. Can be direct or delayed. This is also called sequence to sequence (seq2seq) modelling.

Example:

1. Video-captioning: i.e. describing a sequence of images via text (direct)
2. Translating one language into another (delayed)

Recurrent Neural Network: Equations

Weight matrices in a single-hidden layer RNN



Net input:

$$\boldsymbol{z}_h^{(t)} = \boldsymbol{W}_{hx}\boldsymbol{x}^{(t)} + \boldsymbol{W}_{hh}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_h$$

Activation:

$$\boldsymbol{h}^{(t)} = \sigma_h\left(\boldsymbol{z}_h^{(t)}\right)$$

Net input:

$$\boldsymbol{z}_y^{(t)} = \boldsymbol{W}_{yh}\boldsymbol{h}^{(t)} + \boldsymbol{b}_y$$

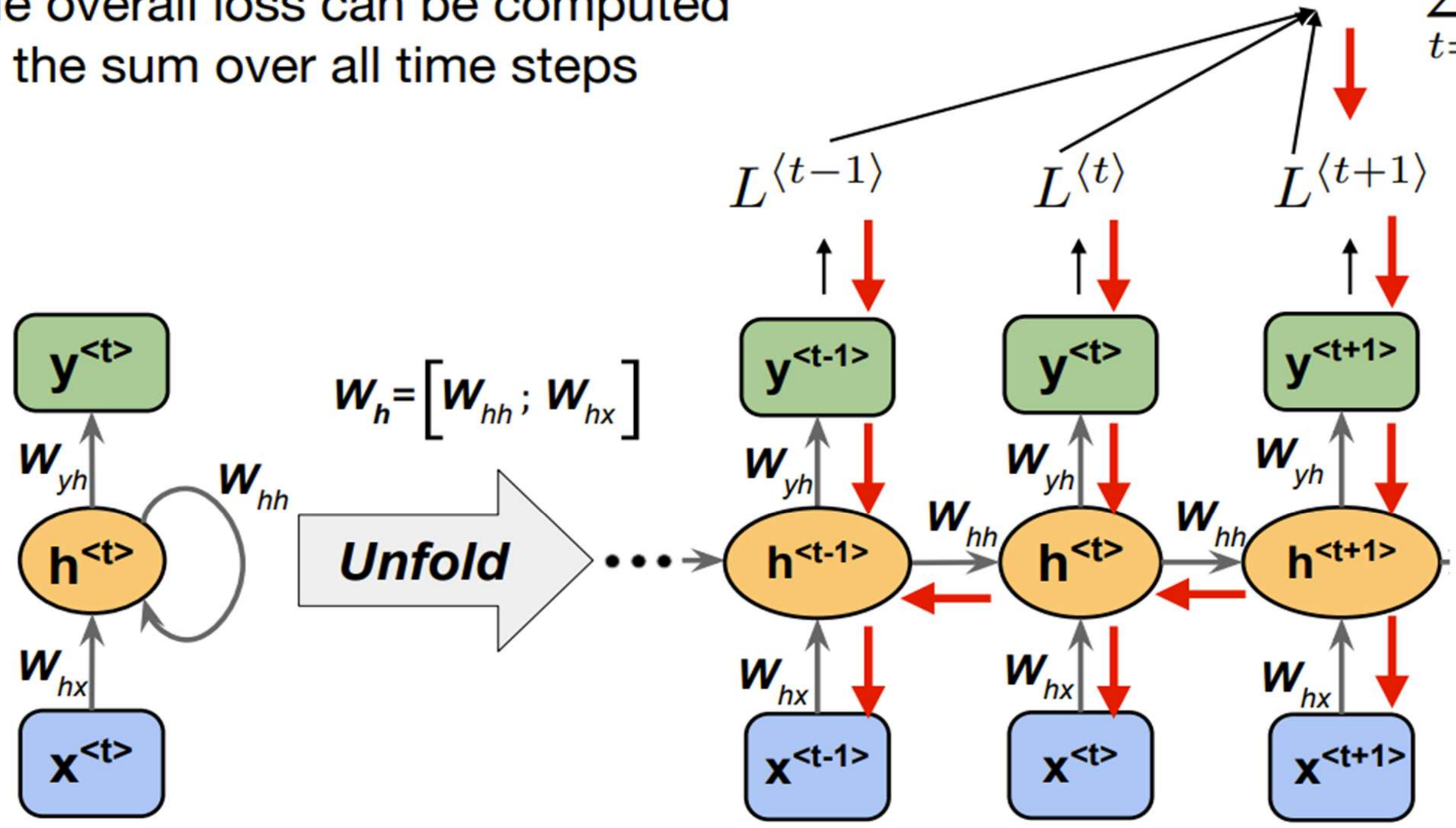
Activation:

$$\boldsymbol{y}^{(t)} = \sigma_y\left(\boldsymbol{z}_y^{(t)}\right)$$

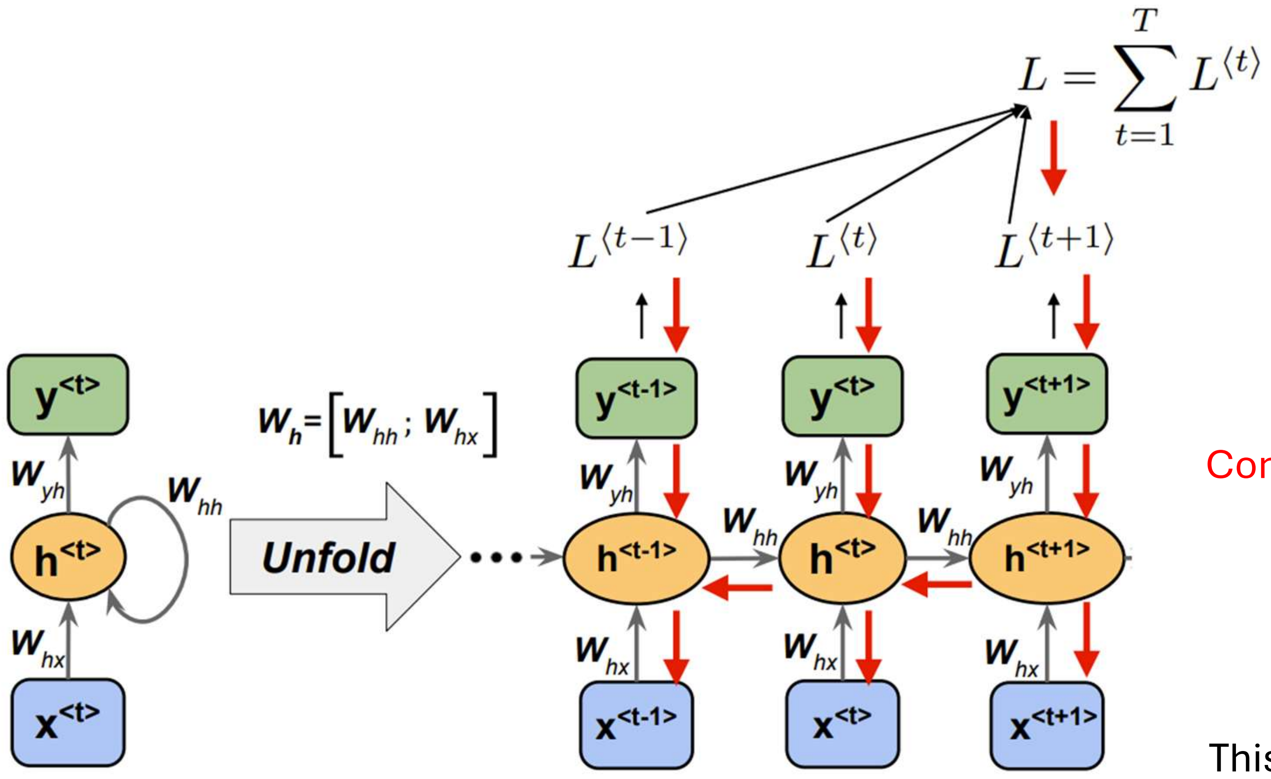
Back-propagation Through Time (BPTT)

The overall loss can be computed as the sum over all time steps

$$L = \sum_{t=1}^{T} L^{\langle t \rangle}$$



Back-propagation Through Time (BPTT)



$$\frac{\partial L^{(t)}}{\partial \boldsymbol{W}_{hh}} = \frac{\partial L^{(t)}}{\partial y^{(t)}} \cdot \frac{\partial y^{(t)}}{\partial \boldsymbol{h}^{(t)}} \cdot \left(\sum_{k=1}^{t} \boxed{\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(k)}}} \cdot \frac{\partial \boldsymbol{h}^{(k)}}{\partial \boldsymbol{W}_{hh}} \right)$$

Computed as a multiplication of adjacent time-steps

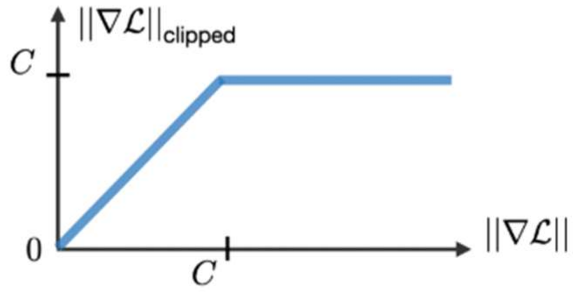
$$\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(k)}} = \prod_{i=k+1}^{t} \frac{\partial \boldsymbol{h}^{(i)}}{\partial \boldsymbol{h}^{(i-1)}}$$

This is very problematic as this can lead to vanishing or exploding gradient problem!

For long sequences, RNNs fail to capture the long-term dependency due to vanishing / exploding gradient problems.

Solutions to the vanishing/exploding gradient problem

1. **Gradient Clipping:**
   set a max value for gradients if they grow to large (solves only exploding gradient problem)

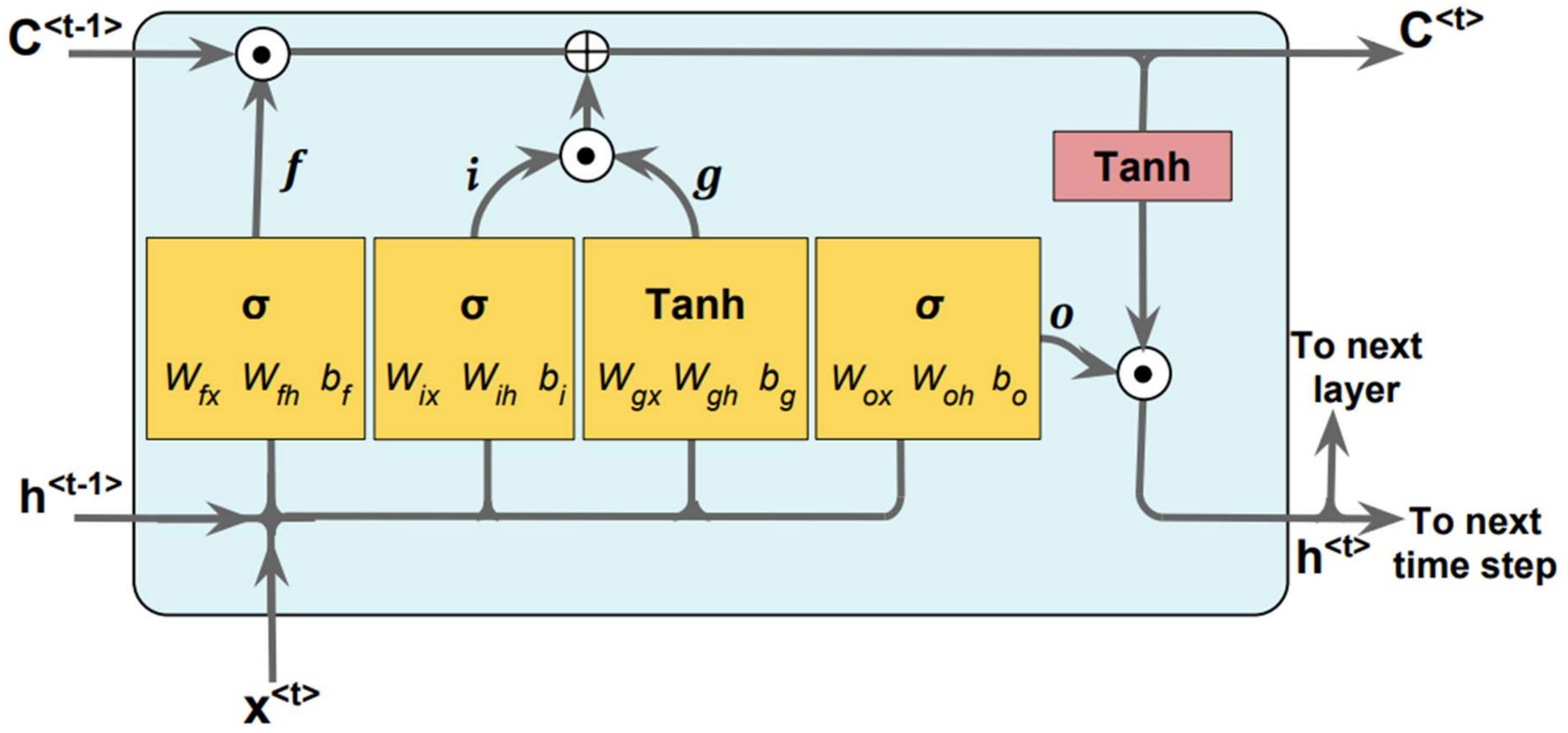


2. Truncated back-propagation through time (TBPTT):
simply limits the number of time steps the signal can backpropagate after each forward pass. E.g., even if the sequence has 100 elements/steps, we may only backpropagate through 20 or so.

3. Long Short-Term Memory (LSTM):
uses a memory cell for modeling long-range dependencies and avoid vanishing gradient problems.

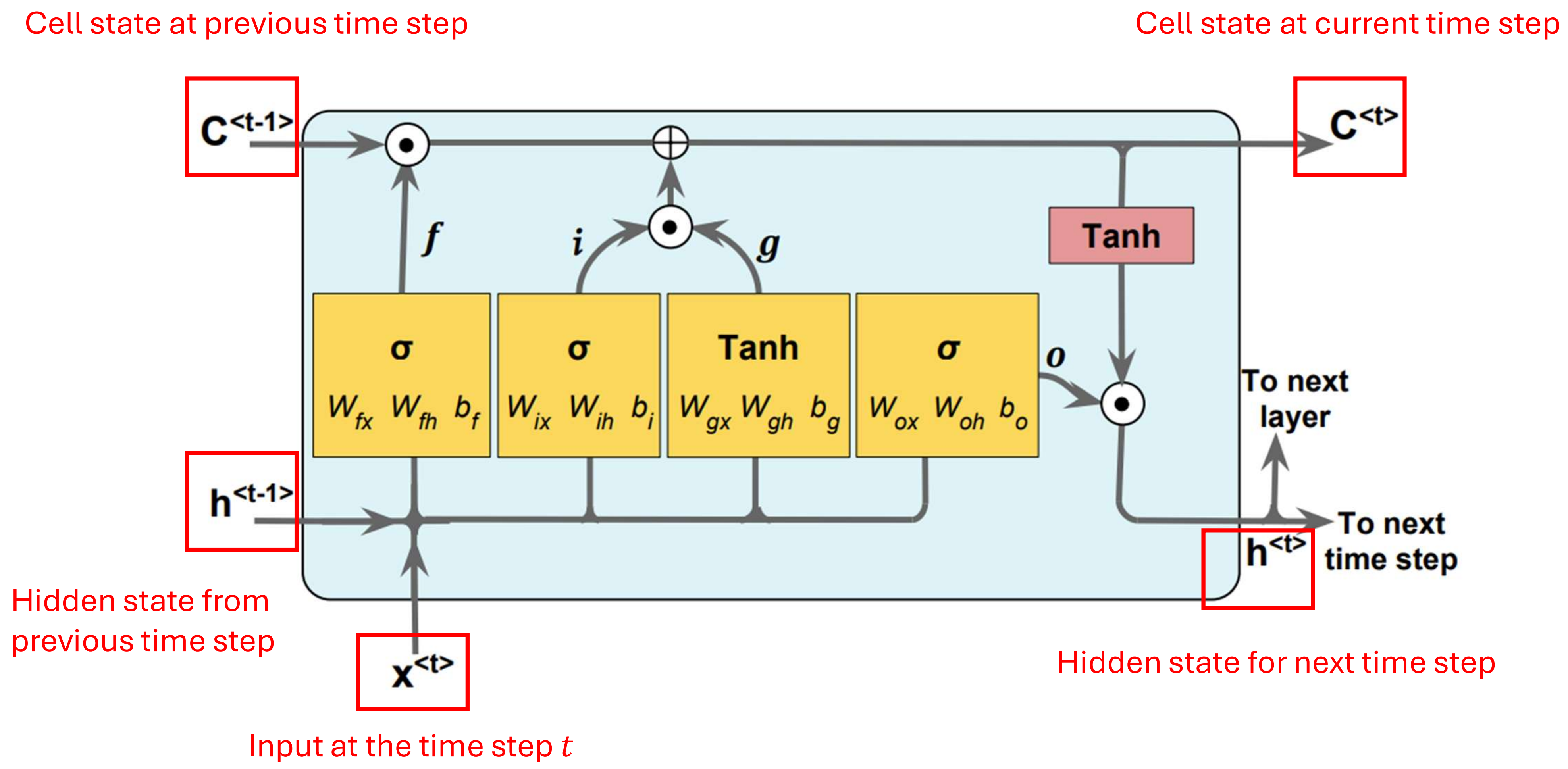
Long Short Term Memory (LSTM)

The architecture of LSTM Cell



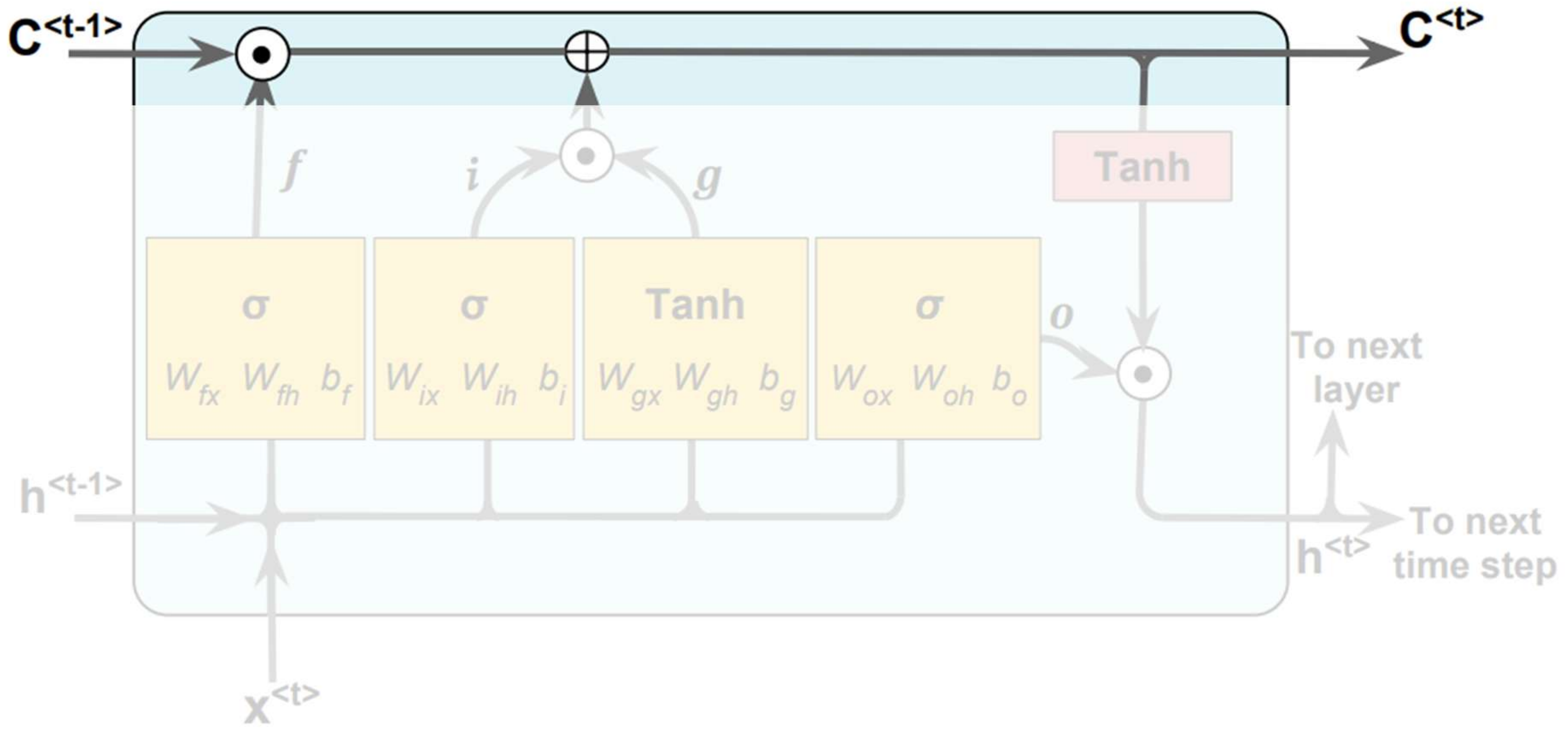
We will describe the architecture and working principle of LSTM in details.

Long Short Term Memory (LSTM)



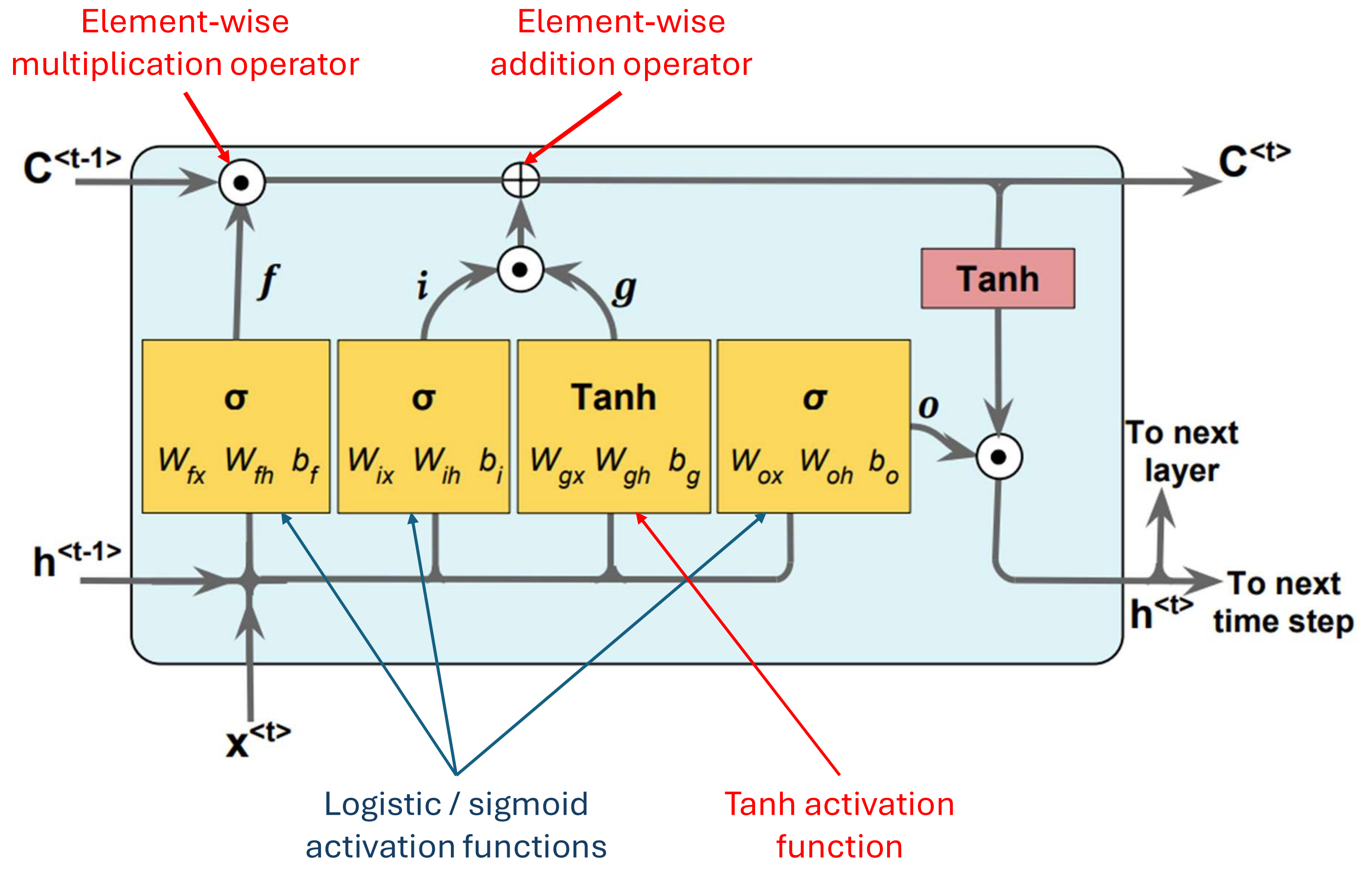
Long Short Term Memory (LSTM)

The key to LSTMs is the **cell state**, the horizontal line running through the top of the diagram. The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.



The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

Long Short Term Memory (LSTM)

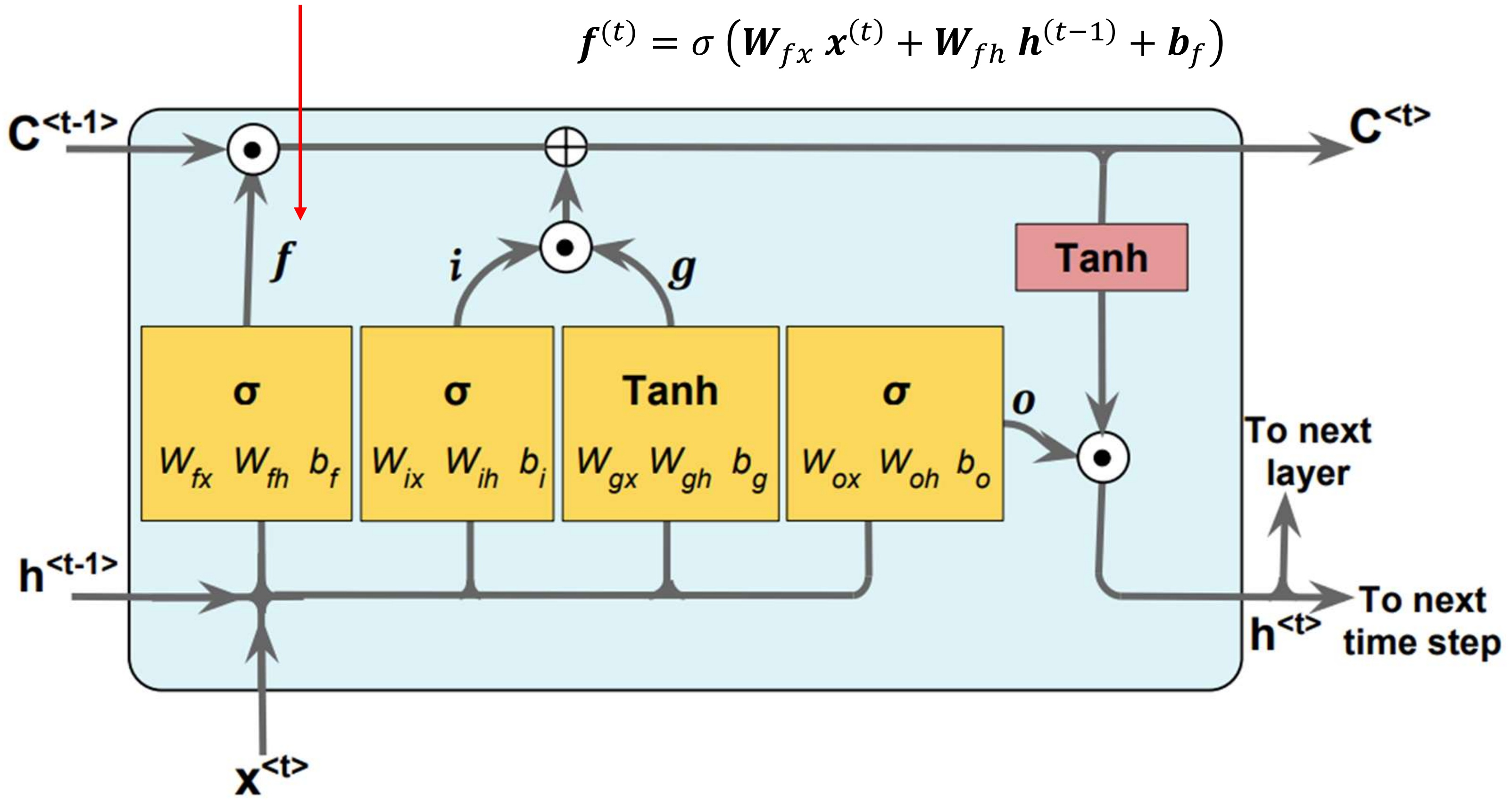


Element-wise multiplication operator
Element-wise addition operator
$C^{<t-1>}$
$C^{<t>}$
f
i
g
o
Tanh
σ W_{fx} W_{fh} b_f
σ W_{ix} W_{ih} b_i
Tanh W_{gx} W_{gh} b_g
σ W_{ox} W_{oh} b_o
$h^{<t-1>}$
$x^{<t>}$
To next layer
$h^{<t>}$
To next time step
Logistic / sigmoid activation functions
Tanh activation function

Long Short Term Memory (LSTM)

Forget Gate: Controls which information is remembered, and which is forgotten; can reset the cell state.

$$\boldsymbol{f}^{(t)} = \sigma\left(\boldsymbol{W}_{fx}\,\boldsymbol{x}^{(t)} + \boldsymbol{W}_{fh}\,\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_f\right)$$

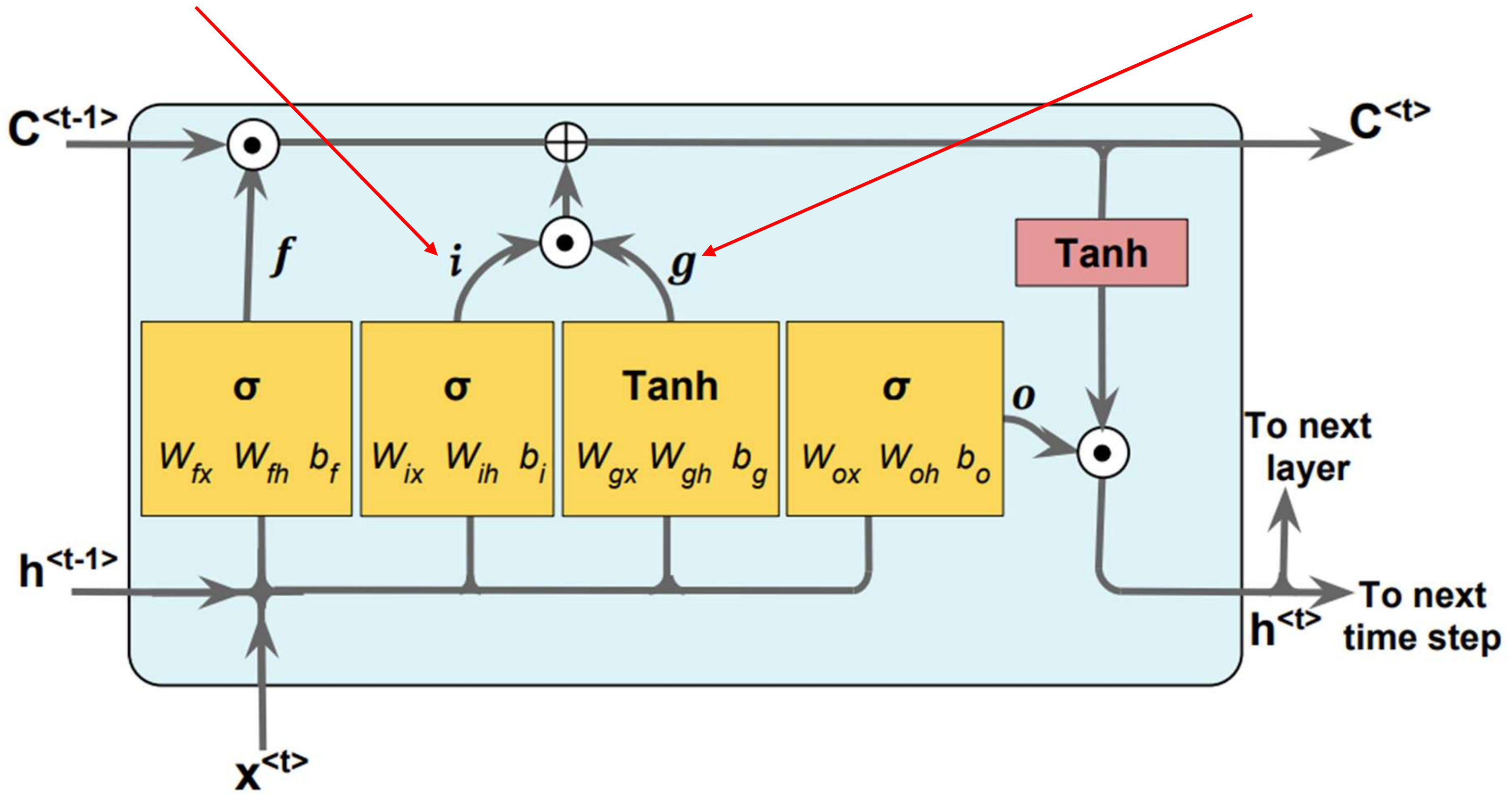


The sigmoid layer outputs numbers between zero and one. A one represents “completely keep this” while a zero represents “completely get rid of this”. For example: the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.

Long Short Term Memory (LSTM)

Input Gate: $\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_{ix}\,\boldsymbol{x}^{(t)} + \boldsymbol{W}_{ih}\,\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_i\right)$

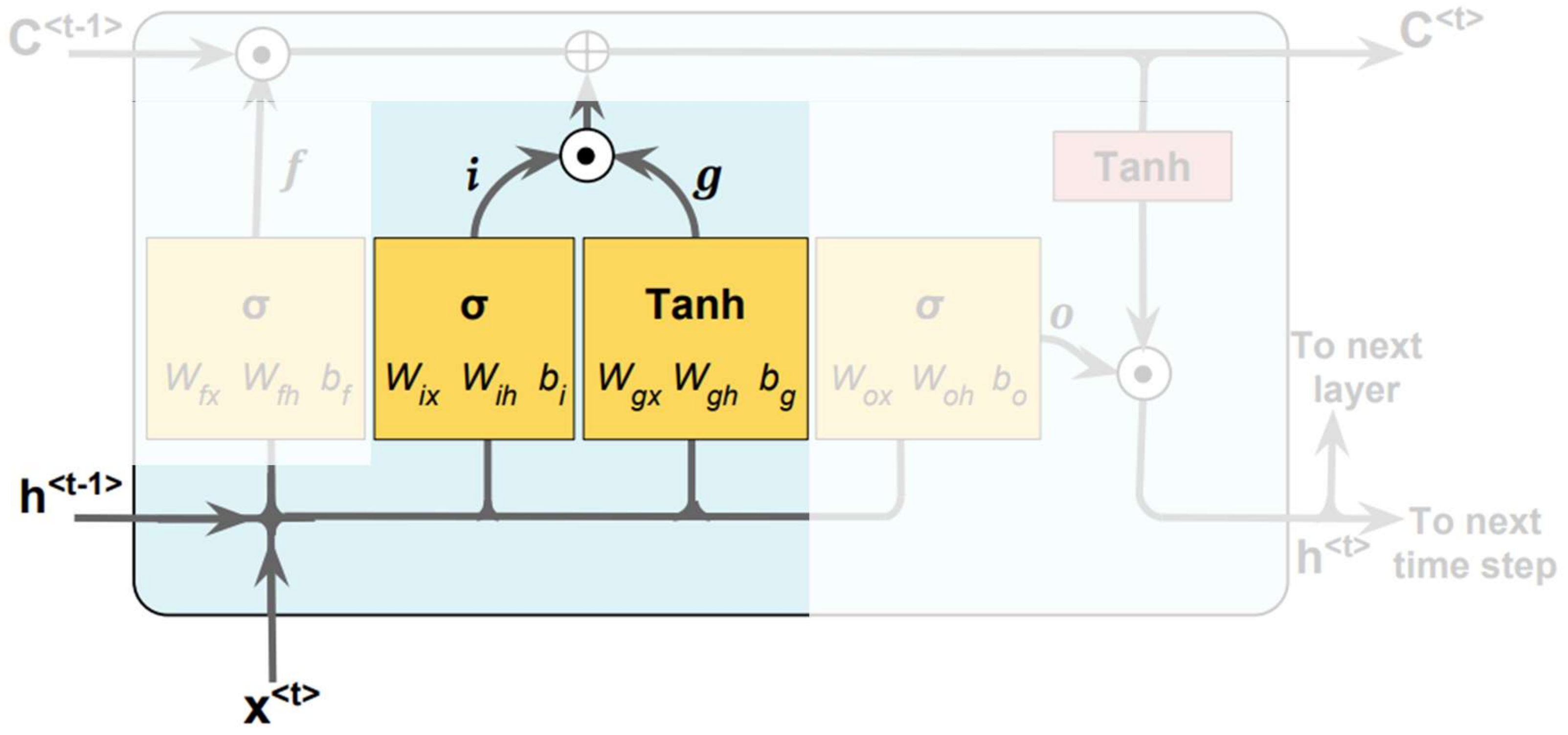
Input Node: $\boldsymbol{g}^{(t)} = tanh\left(\boldsymbol{W}_{gx}\,\boldsymbol{x}^{(t)} + \boldsymbol{W}_{gh}\,\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_i\right)$



Updating the cell state: $\boldsymbol{C}^{(t)} = \left(\boldsymbol{C}^{(t-1)} \odot \boldsymbol{f}_t\right) \oplus \left(\boldsymbol{i}_t \odot \boldsymbol{g}_t\right)$

Long Short Term Memory (LSTM)

This step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer ($\boldsymbol{i}^{(t)}$)" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values $\boldsymbol{g}^{(t)}$, that could be added to the state.

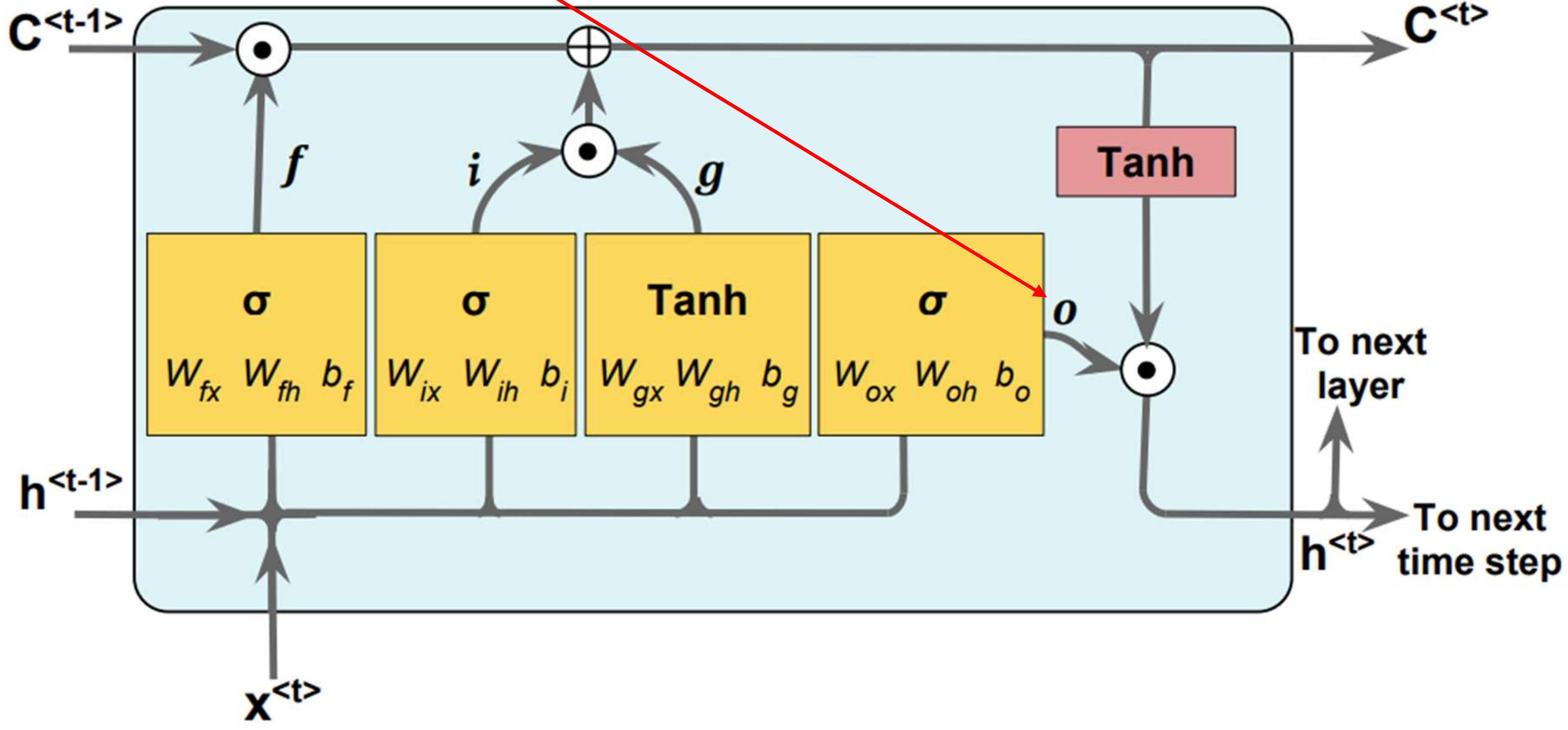


In the case of the language model, this is where we'd actually drop the information about the old subject's gender and add the new information, as we decided in the previous steps.

Long Short Term Memory (LSTM)

Output Gate: For updating the values of the hidden units (hidden state)

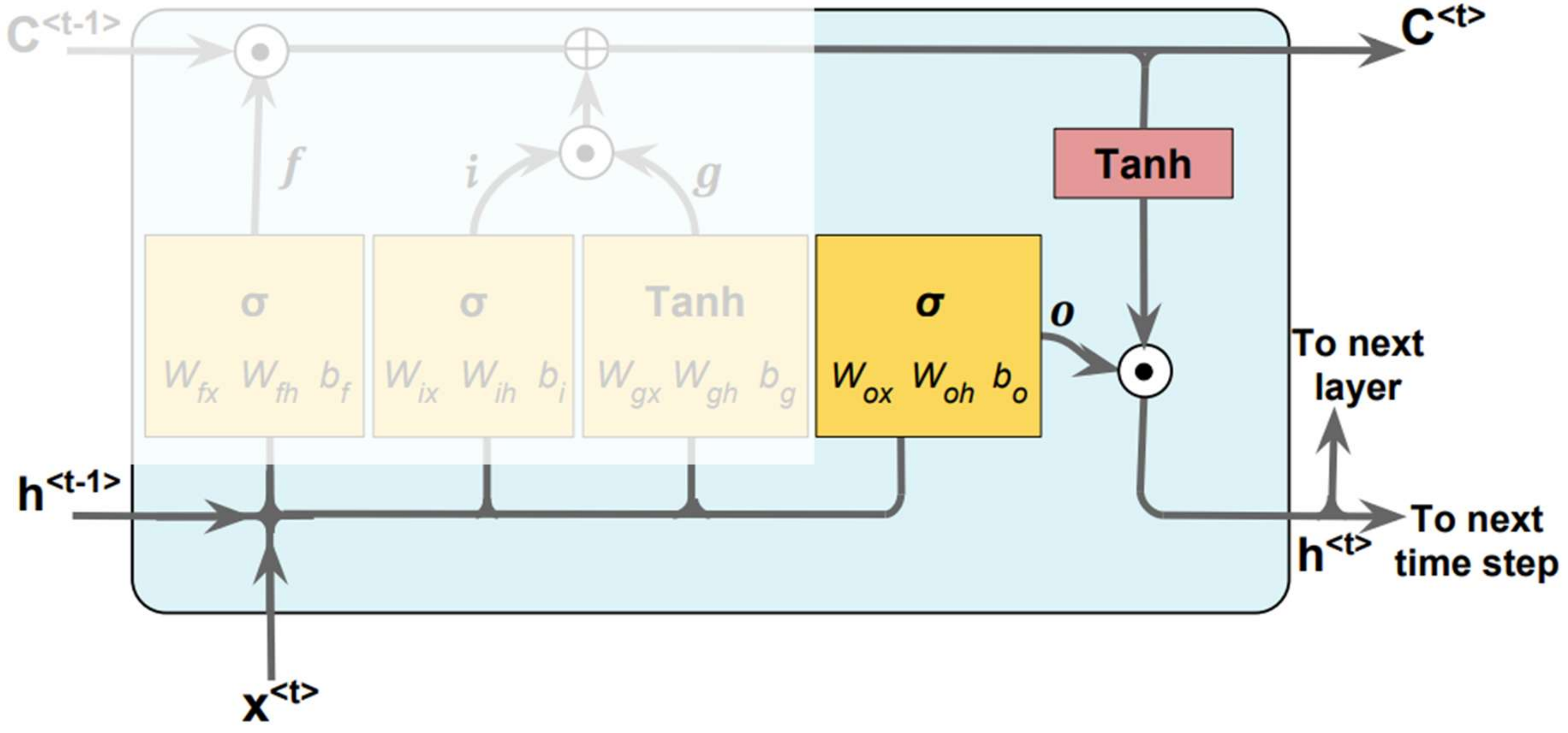
$$o^{(t)} = \sigma\left(\boldsymbol{W}_{ox}\,\boldsymbol{x}^{(t)} + \boldsymbol{W}_{oh}\,\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_o\right)$$



Updating the hidden state: $\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \odot \tanh\left(\boldsymbol{c}^{(t)}\right)$

Long Short Term Memory (LSTM)

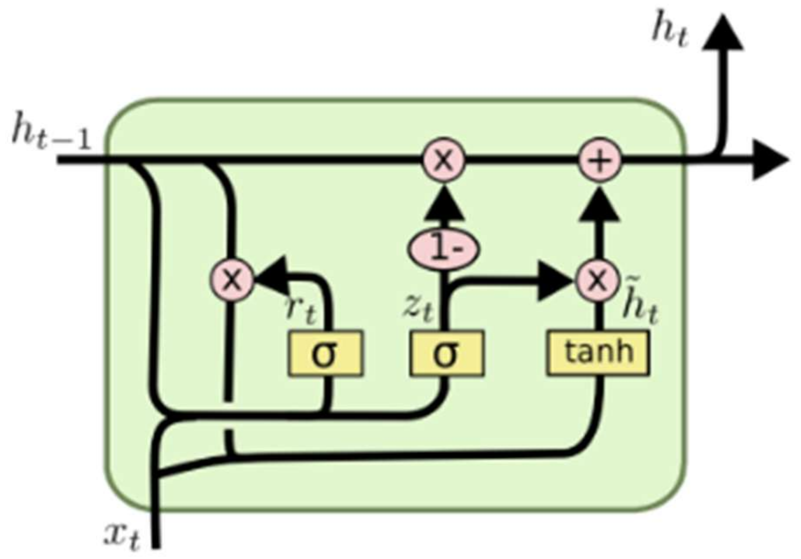
This output will be based on our cell state, but will be a filtered version. First, we put the cell state through **tanh** (to push the values to be between -1 to +1) and multiply it by the output of the sigmoid gate ($\boldsymbol{o}^{(t)}$) , so that we only output the parts of the cell state.



Updating the hidden state: $\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \odot \tanh\left(\boldsymbol{c}^{(t)}\right)$

Gated Recurrent Unit (GRU)

A slightly more dramatic variation on the LSTM is the **Gated Recurrent Unit**, or GRU, introduced by Cho, et al. (2014). It combines the forget and input gates into a single "update gate." It also merges the cell state and hidden state, and makes some other changes. The resulting model is simpler than standard LSTM models, and has been very popular.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

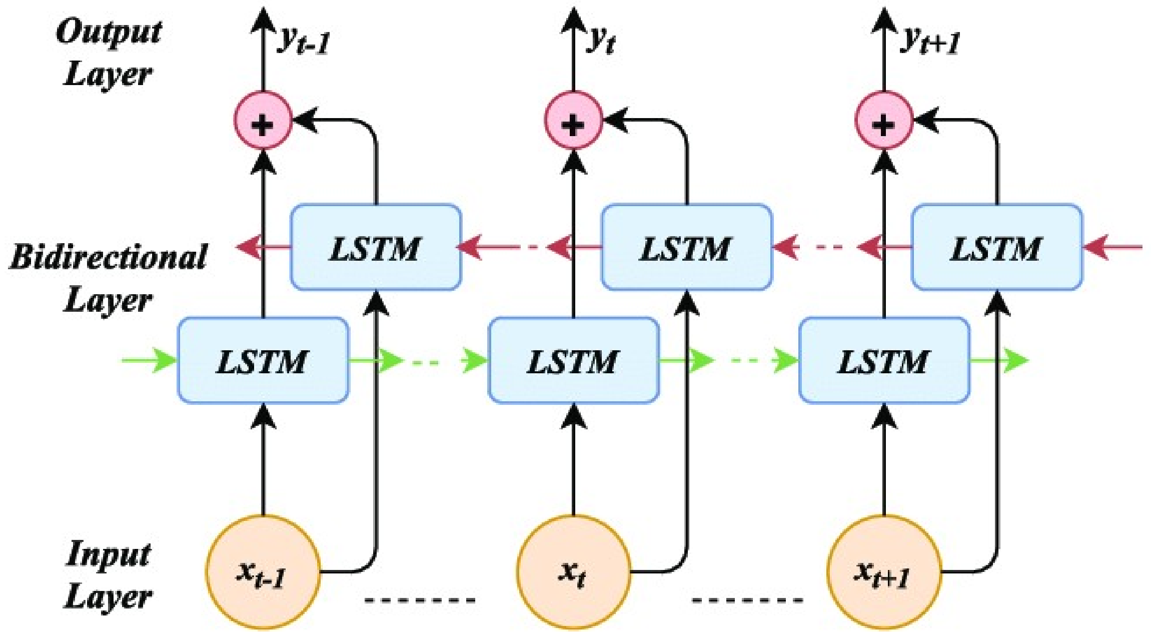
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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

Bidirectional LSTM (Bi-LSTM)

Bi-LSTM (Bidirectional Long Short-Term Memory) is a type of recurrent neural network (RNN) that processes sequential data in both forward and backward directions. It combines the power of LSTM with bidirectional processing, allowing the model to capture both past and future context of the input sequence.



The output is the combination of the forward and backward LSTM layers:

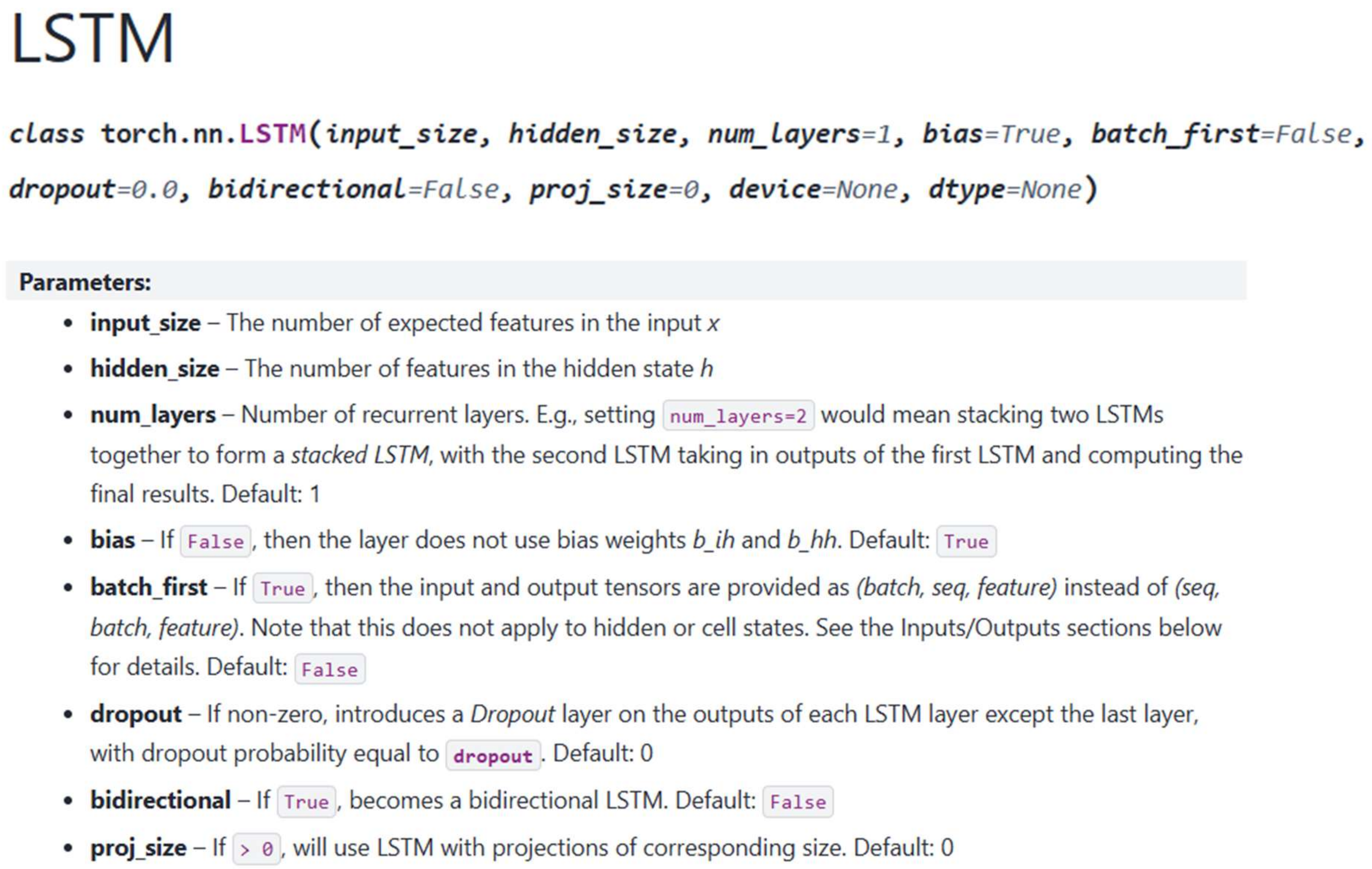
$$\boldsymbol{y}^{(t)} = \boldsymbol{y}_f^{(t)} + \boldsymbol{y}_b^{(t)}$$

Where,

- $\boldsymbol{y}_f^{(t)}$: Output from forward LSTM layer
- $\boldsymbol{y}_b^{(t)}$: Output from backward LSTM layers

During the forward pass, the input sequence is fed into the forward LSTM layer from the first-time step to the last. Simultaneously, the input sequence is also fed into the backward LSTM layer in reverse order, from the last time step to the first.

LSTM in PyTorch



LSTM

```
class torch.nn.LSTM(input_size, hidden_size, num_layers=1, bias=True, batch_first=False,
dropout=0.0, bidirectional=False, proj_size=0, device=None, dtype=None)
```

Parameters:

- **input_size** – The number of expected features in the input *x*
- **hidden_size** – The number of features in the hidden state *h*
- **num_layers** – Number of recurrent layers. E.g., setting `num_layers=2` would mean stacking two LSTMs together to form a *stacked LSTM*, with the second LSTM taking in outputs of the first LSTM and computing the final results. Default: 1
- **bias** – If `False`, then the layer does not use bias weights *b_ih* and *b_hh*. Default: `True`
- **batch_first** – If `True`, then the input and output tensors are provided as *(batch, seq, feature)* instead of *(seq, batch, feature)*. Note that this does not apply to hidden or cell states. See the Inputs/Outputs sections below for details. Default: `False`
- **dropout** – If non-zero, introduces a *Dropout* layer on the outputs of each LSTM layer except the last layer, with dropout probability equal to `dropout`. Default: 0
- **bidirectional** – If `True`, becomes a bidirectional LSTM. Default: `False`
- **proj_size** – If `> 0`, will use LSTM with projections of corresponding size. Default: 0

Thank You