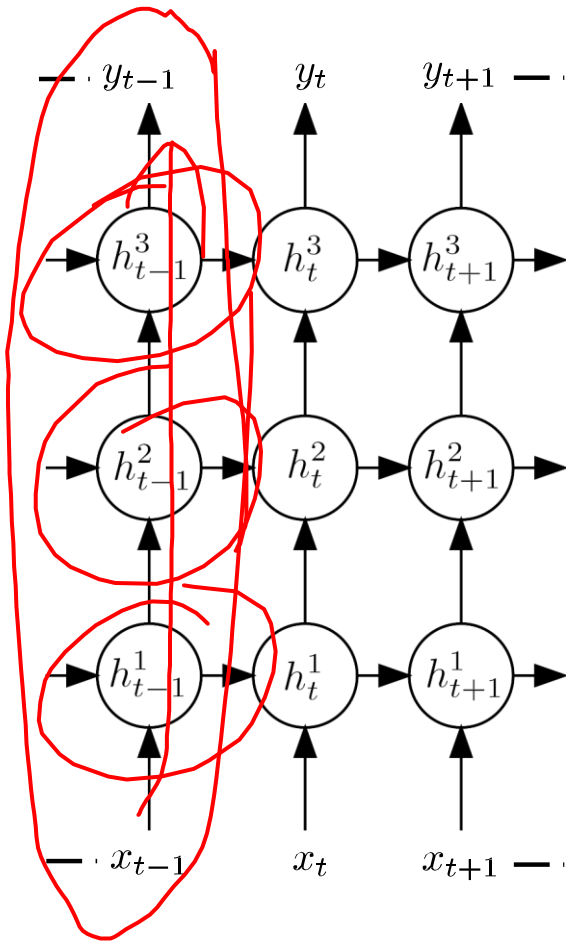
Learning With Memory – II
RNN variants and
Long Short-Term Memory

Major shortcomings of RNNs

- Handling of complex non-linear interactions
- Difficulties using Backpropagation Through Time to capture long-term dependencies - exploding gradients
- Vanishing gradients

Handling non-linear interactions

- Have depth not only in temporal dimension
- But also in space (at each time step)
- Empirically shown to provide significant improvement in tasks like ASR, un-supervised training using videos

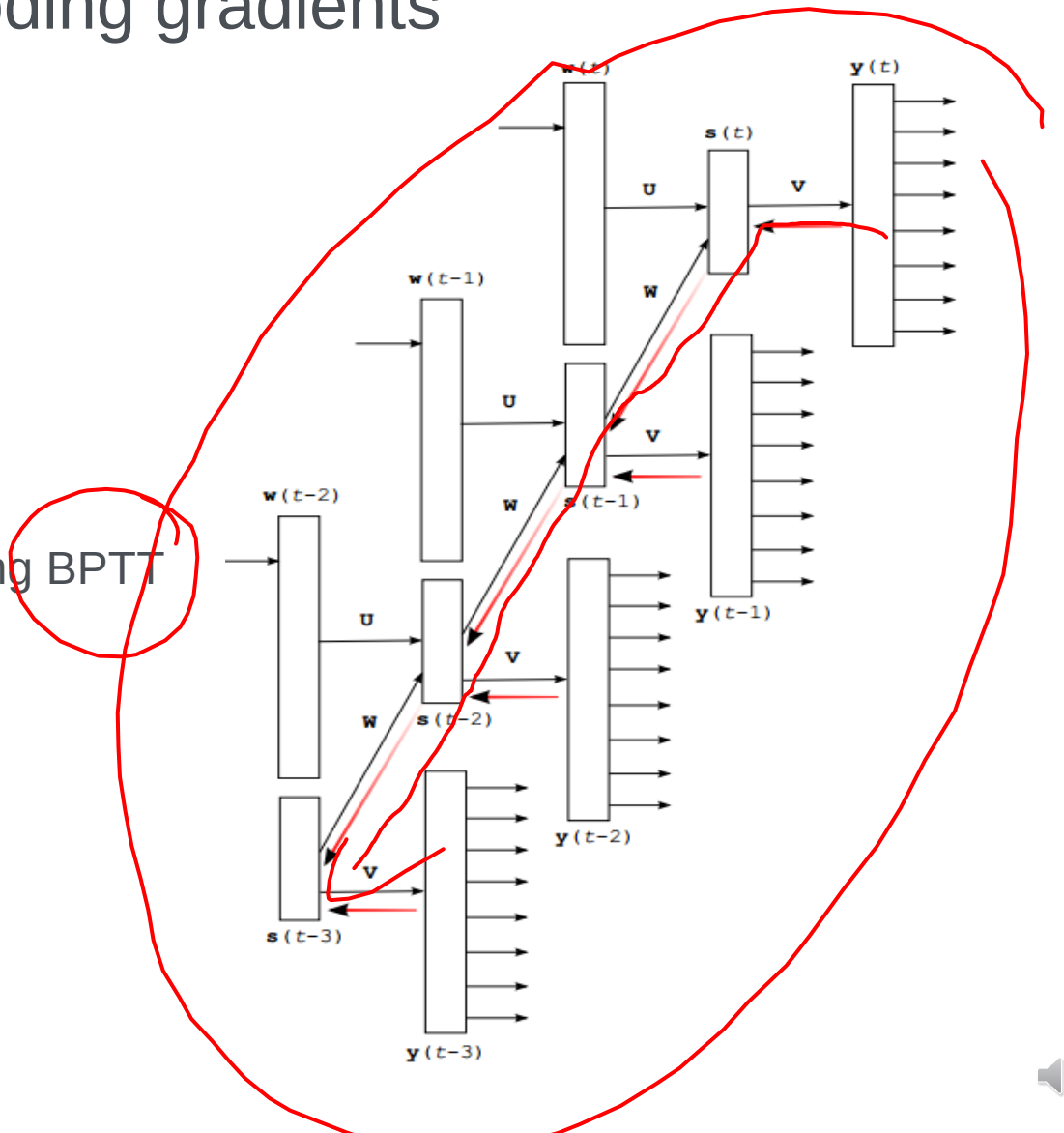


Exploding gradients

- Sometimes, the gradients start to increase exponentially during backpropagation through the recurrent weights
- Happens rarely, but the effect can be catastrophic:
  - huge gradients will lead to big change of weights,
  - This will destroy what has been learned so far
- One of the main reasons why RNNs become unstable
- Simple solution (first published in RNNLM toolkit in 2010): clip or normalize values of the gradients to avoid huge changes of weights

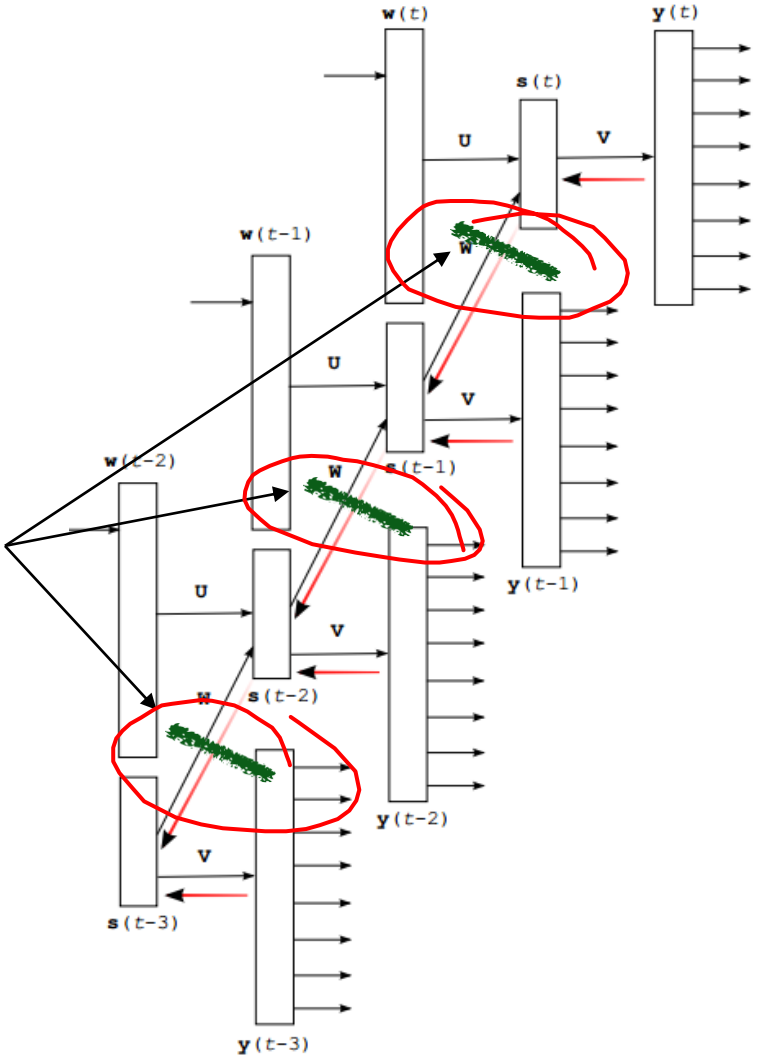
Training: Exploding gradients

Gradient clipping during BPTT



Training: Exploding gradients

Gradient clipping during BPTT

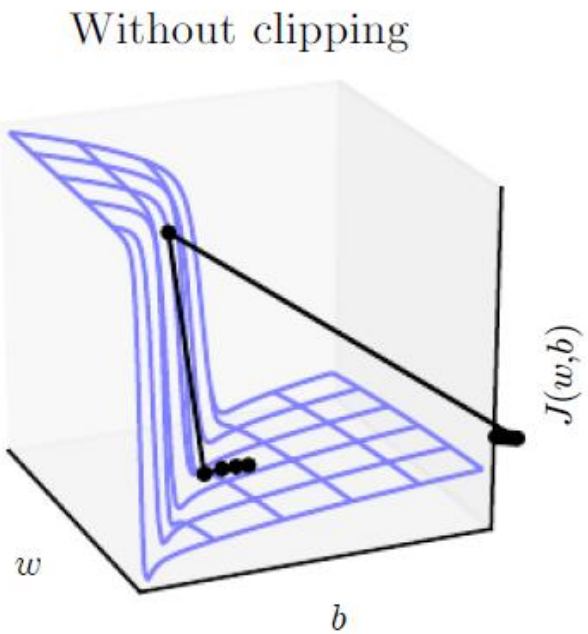


Exploding gradients – Gradient clipping

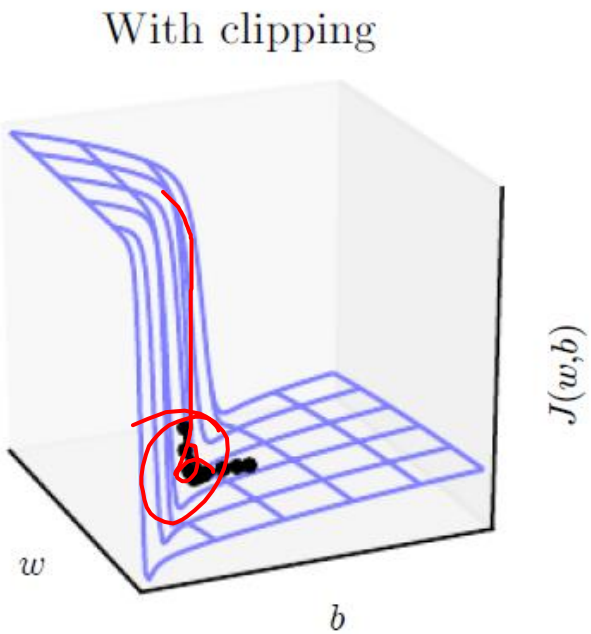
- Gradient clipping can make gradient descent perform more reasonably in the vicinity of extremely steep cliffs. These steep cliffs commonly occur in recurrent networks near where a recurrent network behaves approximately linearly.

- The cliff is exponentially steep in the number of time steps because the weight matrix is multiplied by itself once for each time step.

Exploding gradients – Gradient clipping



Gradient descent without gradient clipping overshoots the bottom of this small ravine, then receives a very large gradient from the cliff face. The large gradient catastrophically propels the parameters outside the axes of the plot.



Gradient clipping has a more moderate reaction to the cliff. While it does ascend the cliff face, the step size is restricted so that it cannot be propelled away from steep region near the solution.

Vanishing gradients

- As we propagate the gradients back in time, usually their magnitude quickly decreases: “vanishing gradient problem”

- In practice this means that learning long term dependencies in data is difficult for simple RNN architecture

- One approach: better initialization of the recurrent matrix and using momentum during training
  - *On the Importance of Initialization and Momentum in Deep Learning (Sutskever et al. 2013)*

- Special RNN architectures address this problem
  - *Exponential trace memory* (Jordan 1987, Mozer 1989)
  - *Long Short-term Memory* (Hochreiter & Schmidhuber, 1997))

GRU

Long Short-Term Memory (LSTM)

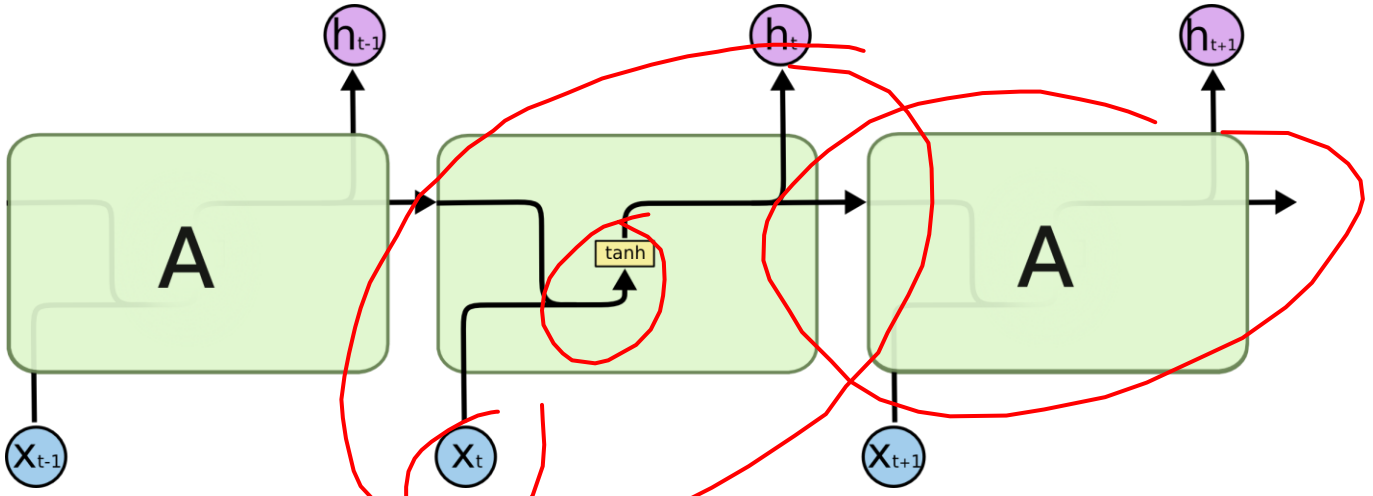
- Recently gained a lot of popularity
- Main idea: Introduce self-loops to produce paths where the gradient can flow for long durations
  - And make the weight of this self-loop gated (controlled by another hidden unit) -> time scale of integration can change dynamically
- Have explicit memory “cells” to store short-term activations the presence of additional gates partly alleviates the vanishing gradient problem
- Multi-layer versions shown to work quite well on tasks which have “medium term” dependencies

Hochreiter et.al. 1997: Long Short-Term Memory

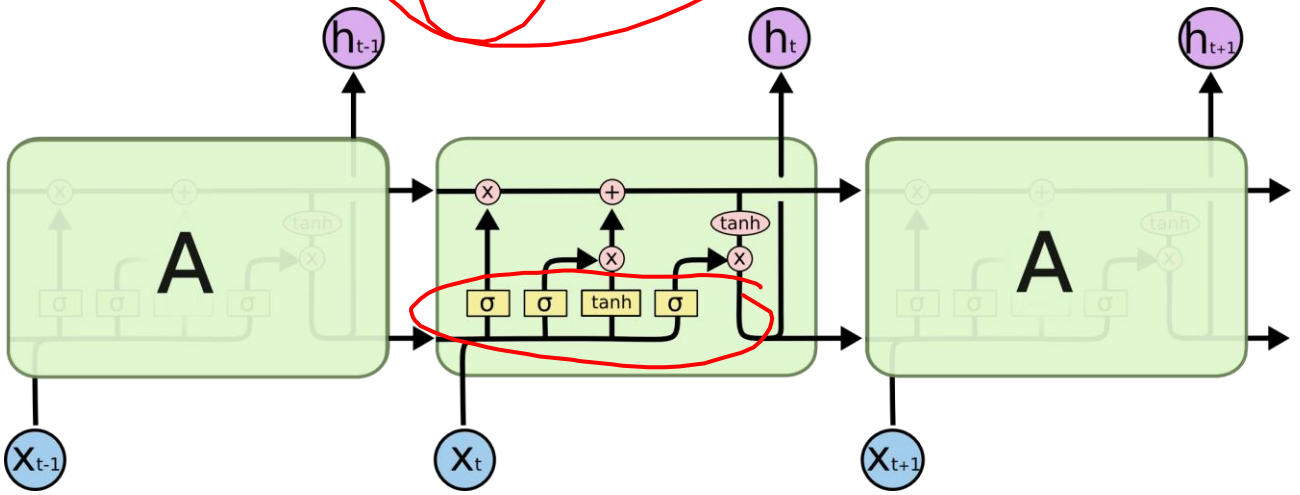
Long Short-Term Memory (LSTM)

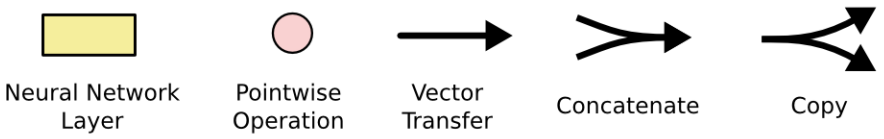
- LSTM has been found to be successful in many applications:
  - Handwriting recognition (Graves et al., 2009)
  - Speech recognition (Graves et al., 2013; Graves and Jaitly, 2014),
  - Handwriting generation (Graves, 2013)
  - Machine translation (Sutskever et al., 2014),
  - Image captioning (Kiros et al., 2014b; Vinyals et al., 2014b; Xu et al., 2015)
  - Parsing (Vinyals et al., 2014a).
  - Human Action Recognition
  - Music Composition

RNN

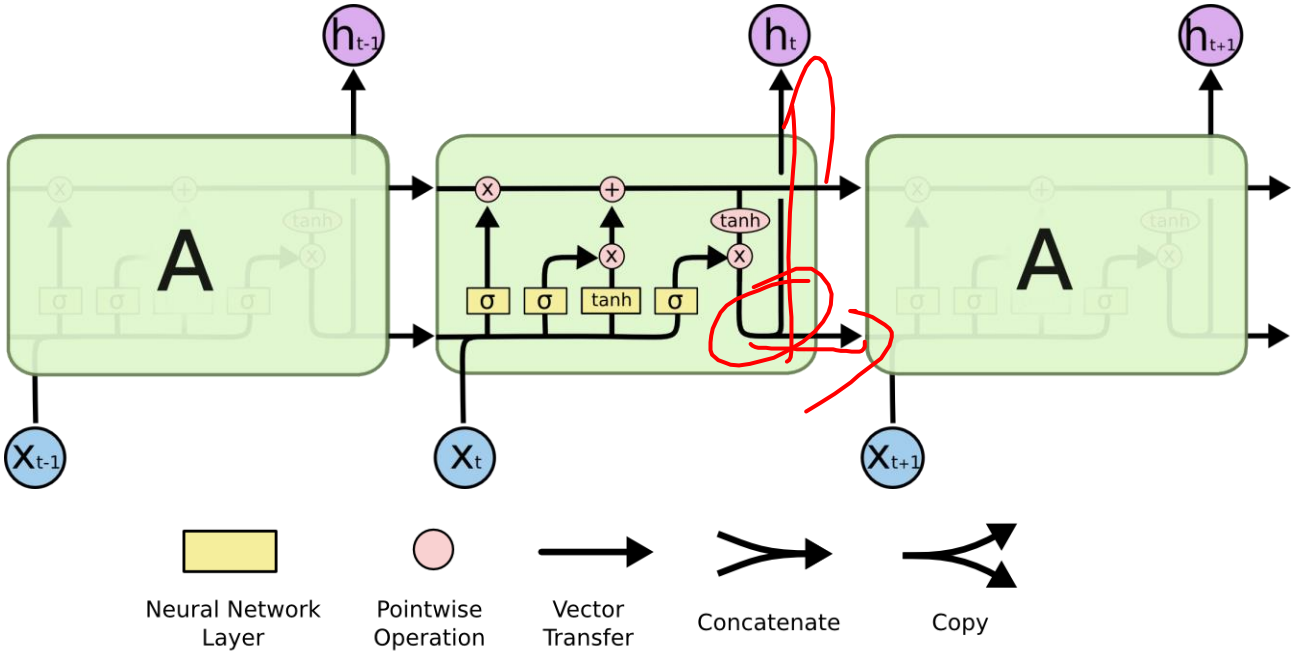


LSTM



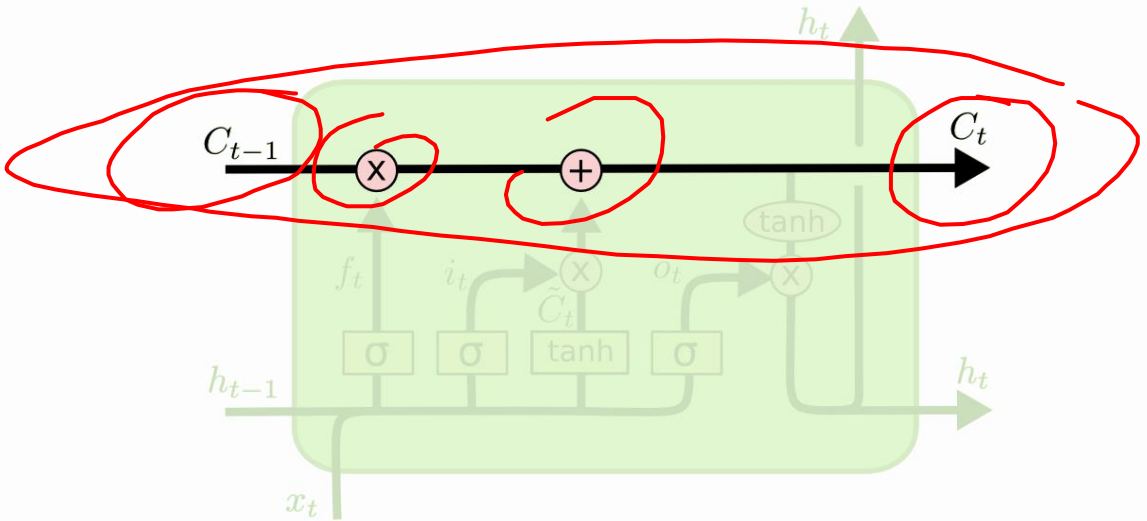


https://colah.github.io/posts/2015-08-Understanding-LSTMs/



- Each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition.
- Yellow boxes are learned neural network layers.
- Lines merging denote concatenation.
- Line forking denote its content being copied and the copies going to different locations.

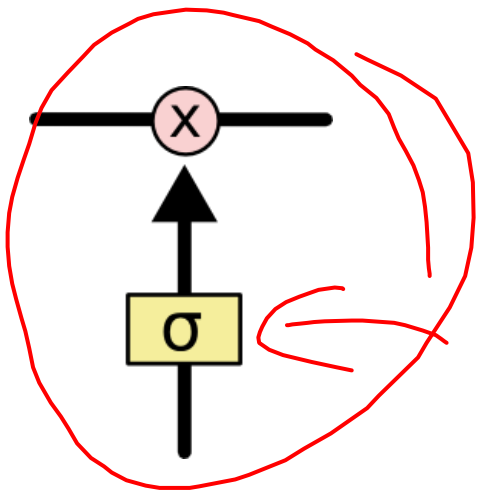
https://colah.github.io/posts/2015-08-Understanding-LSTMs/



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

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

- The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.
- Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.

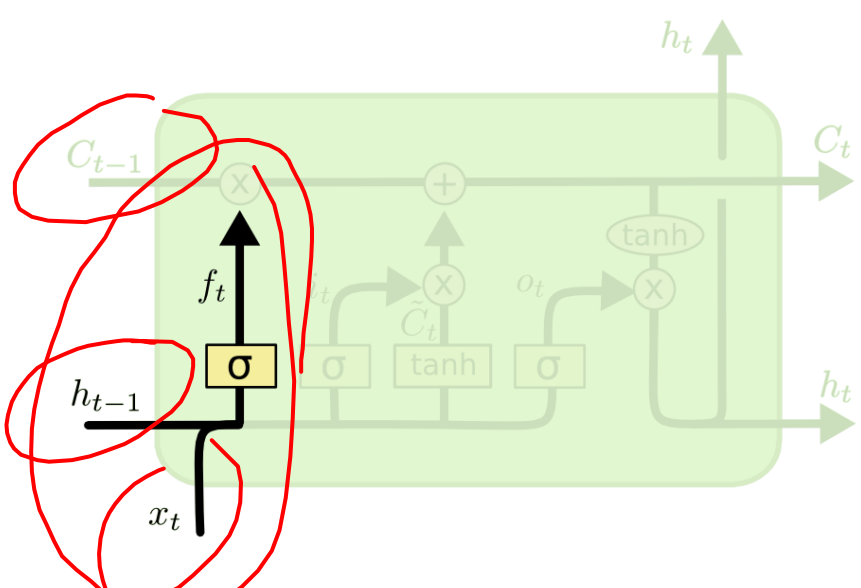


Remember: The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means “let nothing through,” while a value of one means “let everything through!”

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Forget Gate

- The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer."
- It looks at h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}.
- 1 mean "completely keep this" while a 0 represents "completely get rid of this."



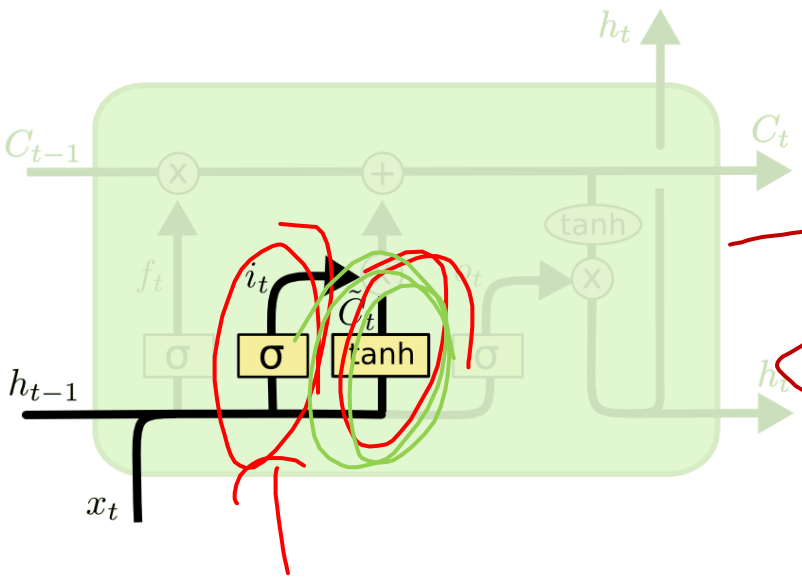
she her

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Input Gate

- The next step is to decide what new information we're going to store in the cell state. This has two parts:
  - "input gate" decides which values we'll update.
  - A tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state. In the next step, we'll combine these two to create an update to the state.



She

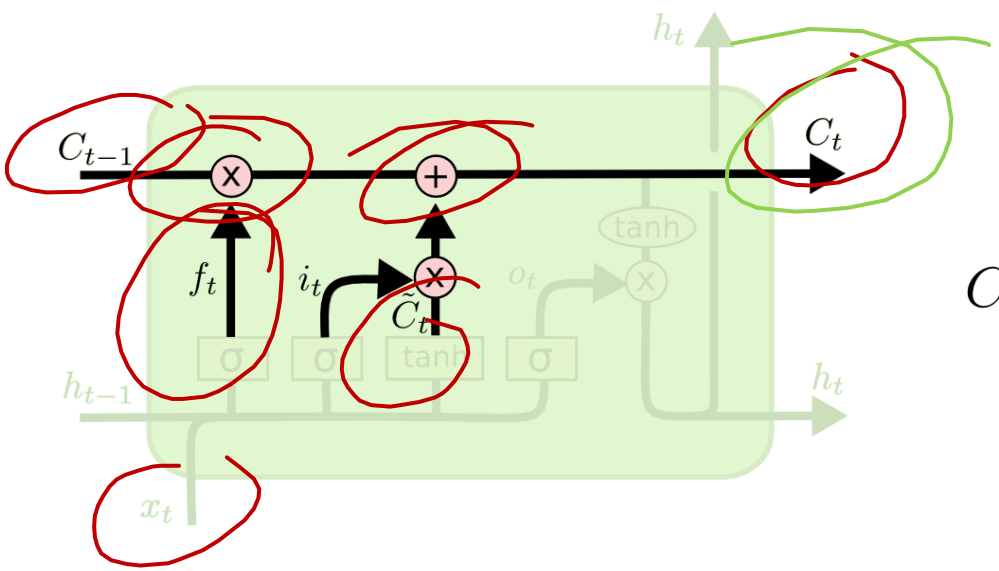
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

New Cell State

– It's now time to update the old cell state, C_{t-1}, into the new cell state C_t. The previous steps already decided what to do, we just need to actually do it.

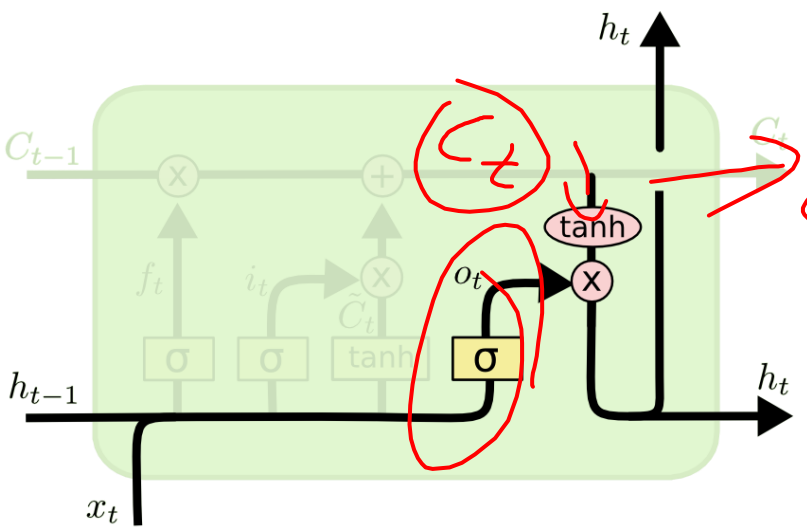
– We multiply the old state by f_t, forgetting the things we decided to forget earlier. Then we add $i_t * \tilde{C}_t$ -this is the new candidate values, scaled by how much we decided to update each state value-



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Output Gate

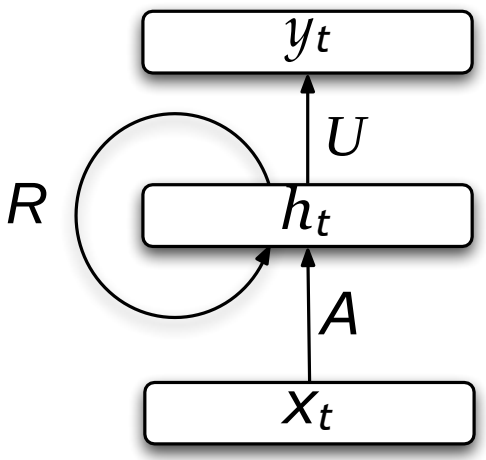
- Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version.
- First, a sigmoid layer decides what parts of the cell state we're going to output.
- Then, we put the cell state through tanh (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.



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

$$h_t = o_t * \tanh\left(C_t\right)$$

Long Short-Term Memory (LSTM)



Multilayer RNNs

$$h_t^l = \tanh W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$h \in \mathbb{R}^n$ $\quad$ $W^l \ [n \times 2n]$

LSTM:

$W^l \ [4n \times 2n]$

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

$$h_t^l = o \odot \tanh(c_t^l)$$

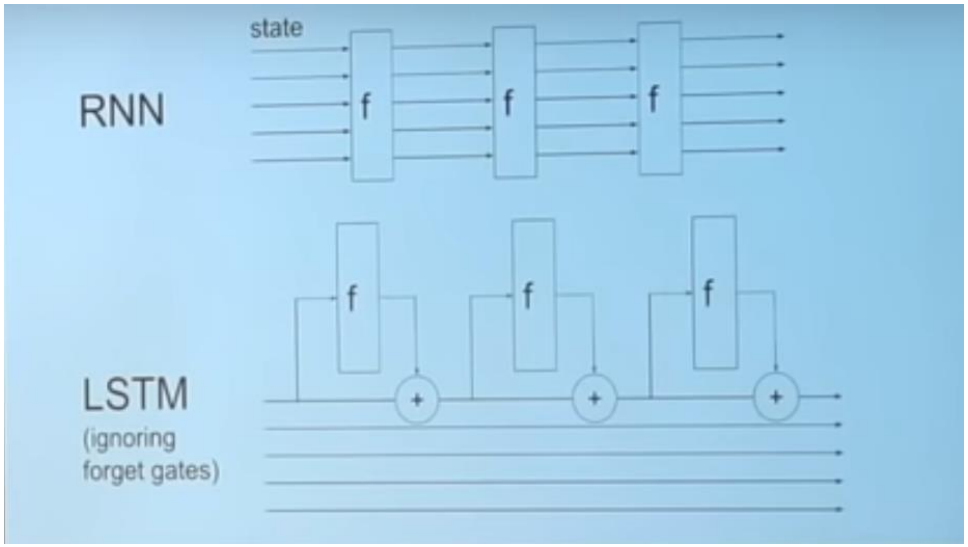
RNN has a single vector, LSTM also has the vector c_t : cell state vector

Hochreiter et.al.,1997: Long Short-Term Memory

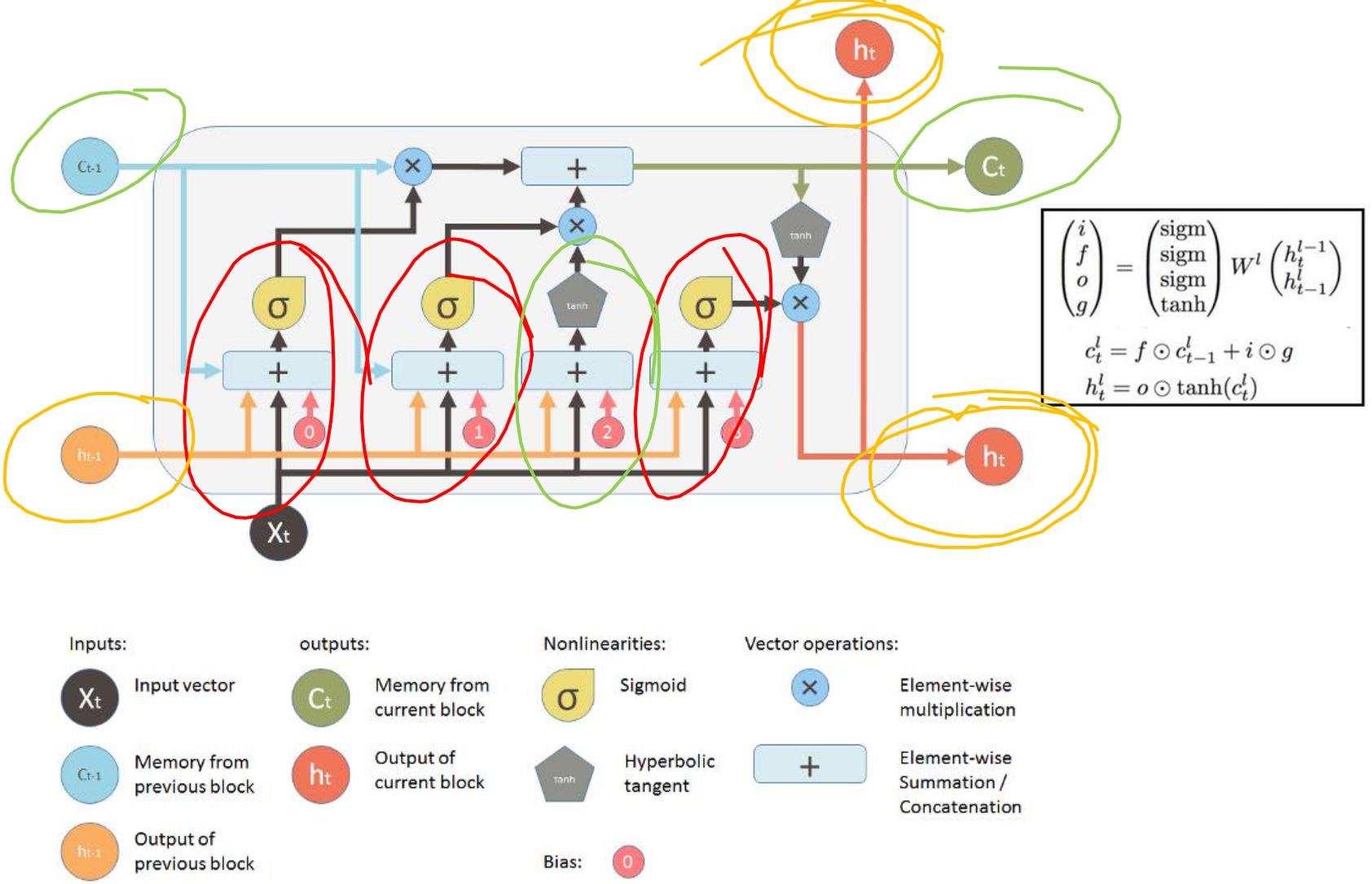
Training: Vanishing gradients

– LSTM is very good with the vanishing gradient problem because of these highways of cells that are only changed with additive interactions where the gradients just flow they never die down if you're multiplying by the same matrix again and again

– While LSTMs are better dynamically, we still do gradient clipping because the gradients in LSTMs can still potentially explode (but they don't usually vanish)



Note: Forget gates can turn on and kill the gradient -> these highways are true when there is no forget gates



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

$$h_t^l = o \odot \tanh(c_t^l)$$

Inputs:

- X_t Input vector
- C_{t-1} Memory from previous block
- h_{t-1} Output of previous block

outputs:

- C_t Memory from current block
- h_t Output of current block

Nonlinearities:

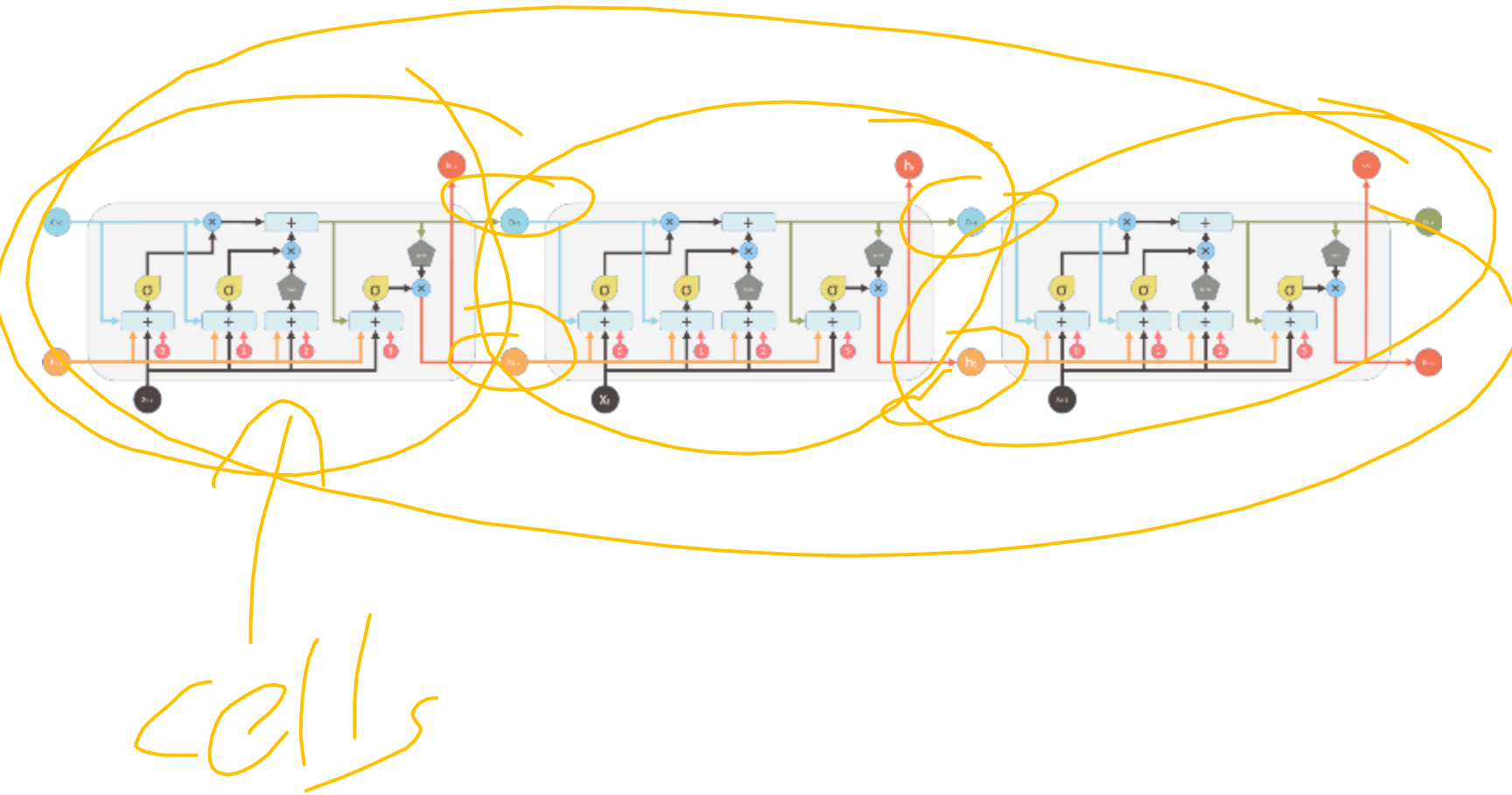
- σ Sigmoid
- tanh Hyperbolic tangent

Bias: 0

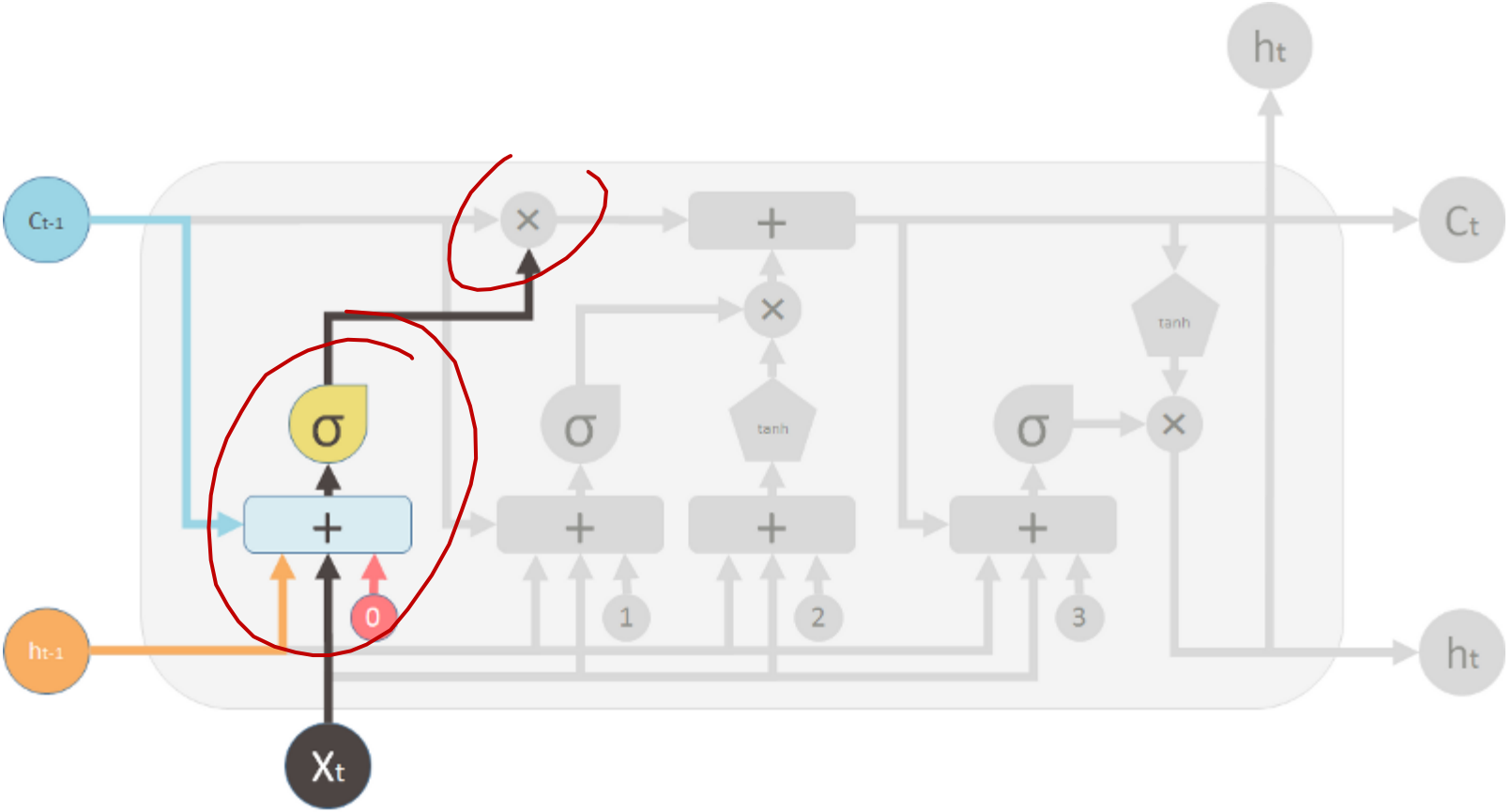
Vector operations:

- × Element-wise multiplication
- \+ Element-wise Summation / Concatenation

https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714



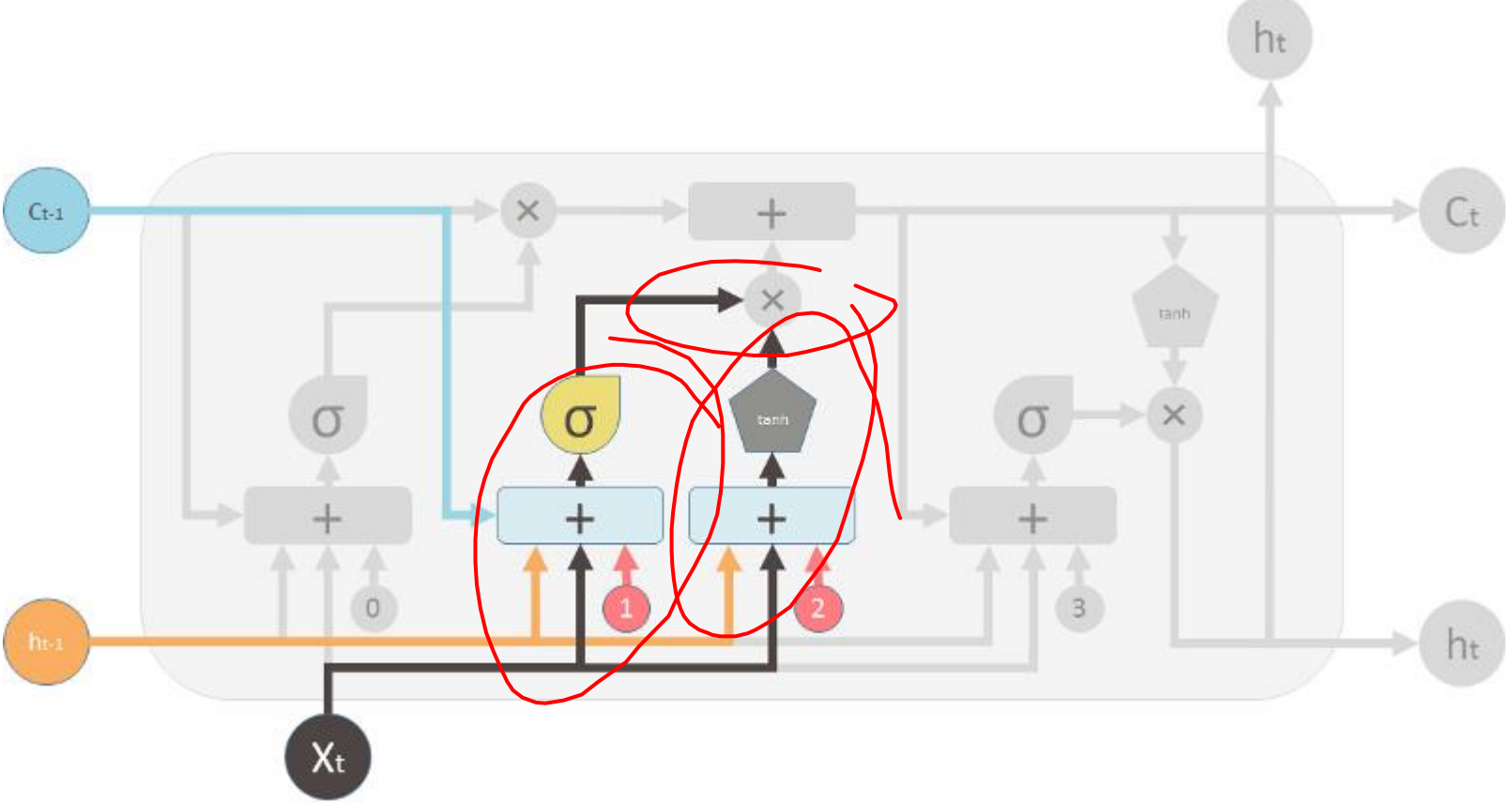
https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714



Forget gate:
If it is shut, no old memory will be kept.
If it is fully open, all old memory will pass through.

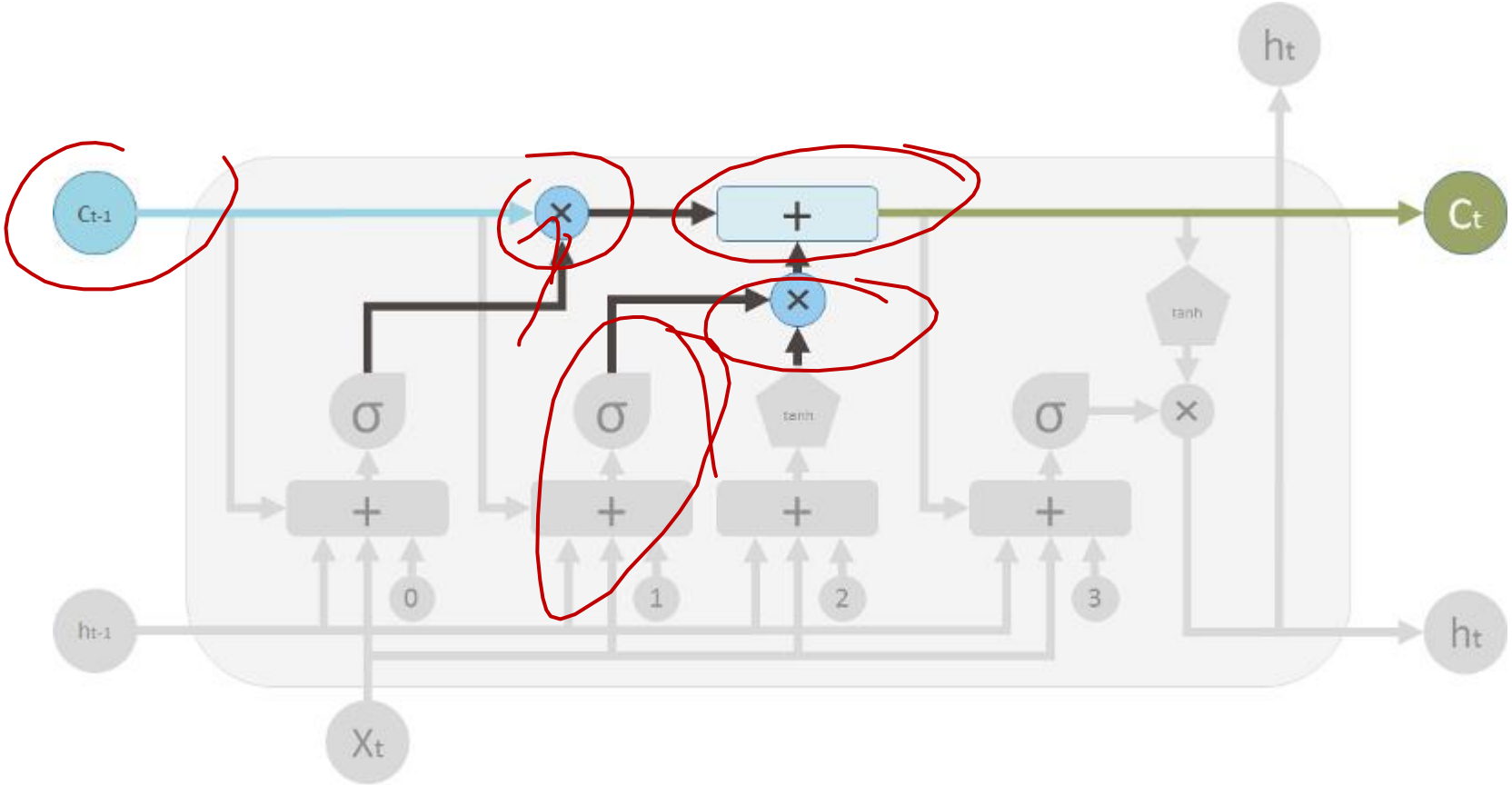
It is actually an element-wise multiplication operation:
If old memory C_{t-1} is multiplied with a vector that is close to 0: forget most of the old memory.
If multiplied with a vector that is close to 1: old memory is let to go through.

https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714



The new memory itself, however is generated by another neural network. It is also a one layer network, but uses tanh as the activation function.
The output of this network will element-wise multiple the new memory valve, and add to the old memory to form the new memory

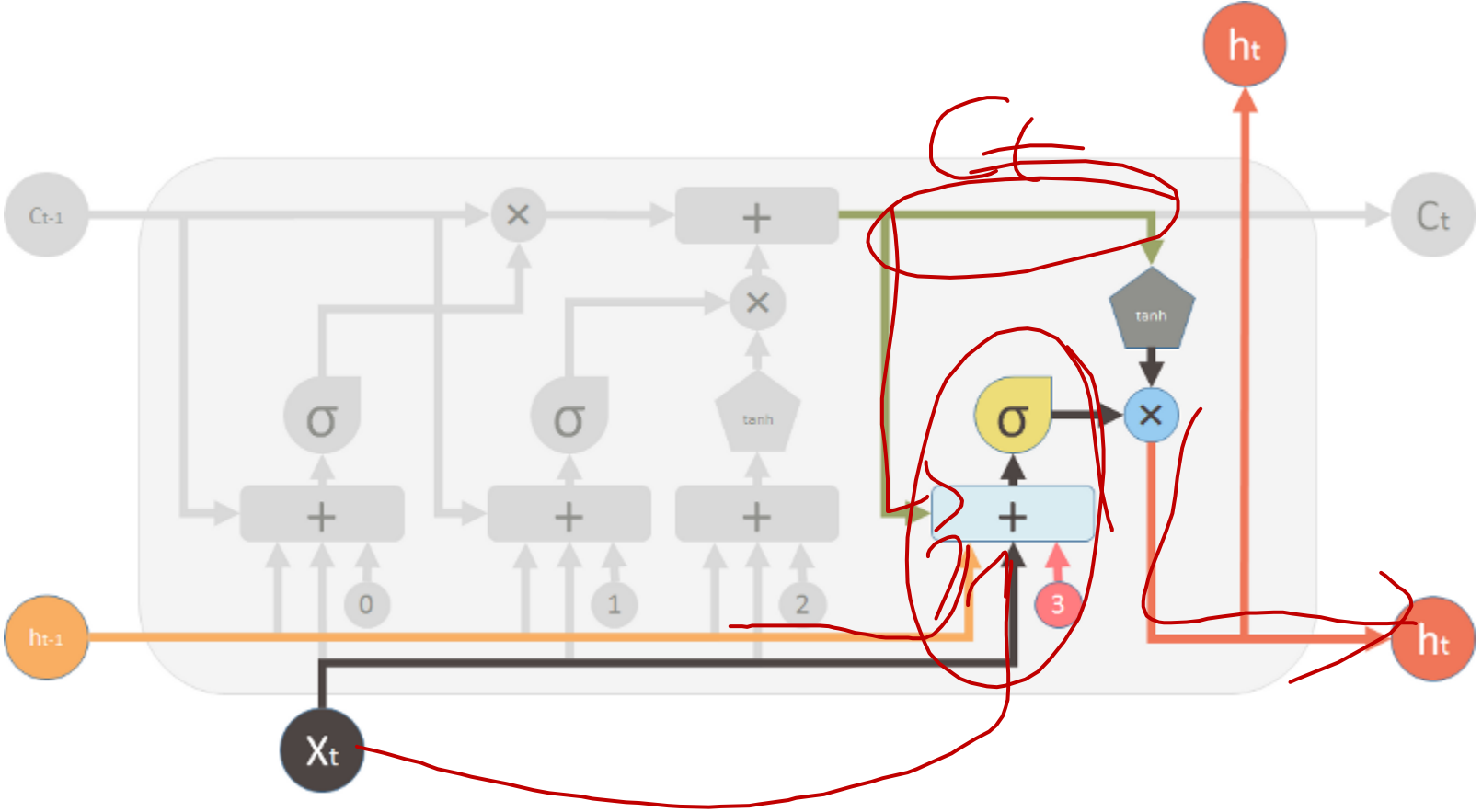
https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714



Piece-wise summation: New memory and the old memory will merge by this operation.

How much new memory should be added to the old memory is controlled by the ✖ below the + sign.

https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714



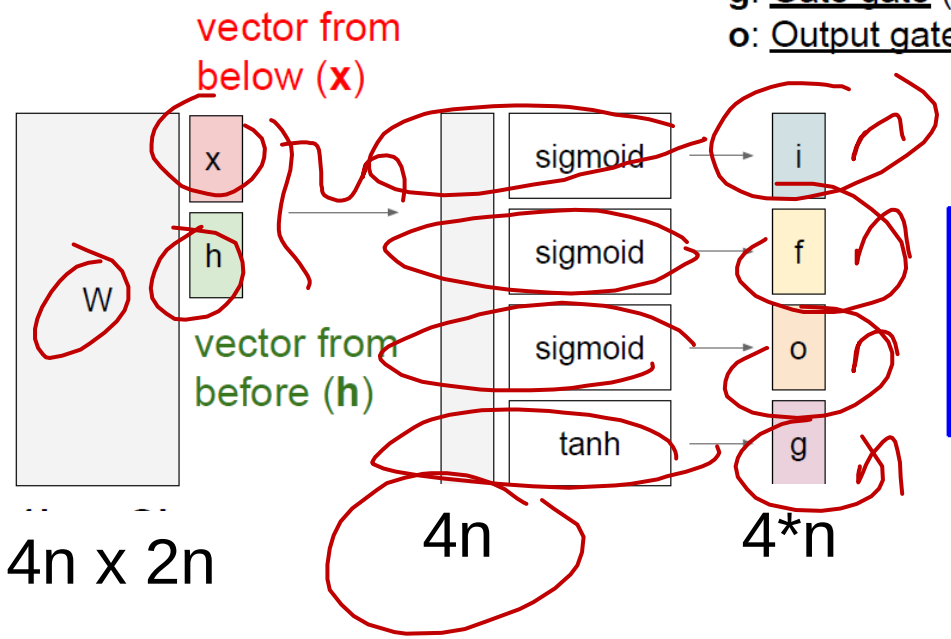
Output gate:
Finally, we need to generate the output for this LSTM unit.
This step has an output gate that is controlled by the new memory, the previous output h_{t-1}, the input X_t and a bias vector.
This gate controls how much new memory should output to the next LSTM unit.

https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

f: Forget gate, Whether to erase cell
i: Input gate, whether to write to cell
g: Gate gate (?), How much to write to cell
o: Output gate, How much to reveal cell



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

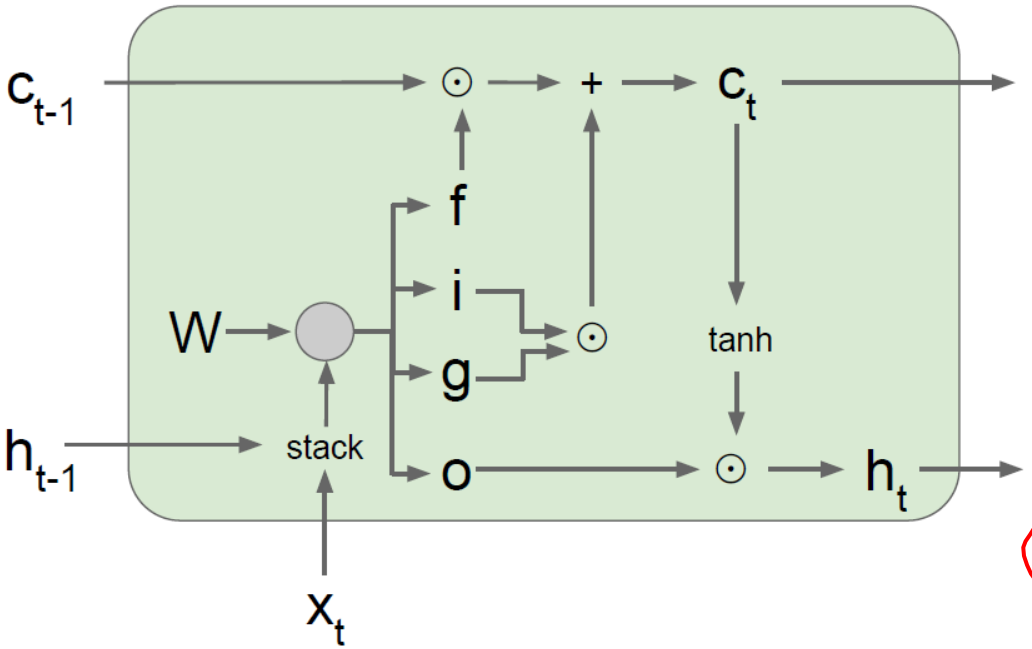
$$h_t = o \odot \tanh(c_t)$$

- We take two vector: from below and from before, these are size n and map them through transformation W (4n x 2n) -> result is size 4n
- We use these to operate over the cell state c_t (depending on what's before and below -> context)
- $\odot$ is element-wise multiplication
- i, f, o are binary (like a gate) but we make them sigmoids because we want this to be differentiable so that we can back propagate through everything

Source: Stanford CS231n: Convolutional Neural Networks for Visual Recognition.

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

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

The first interaction here is f ⊙ c :f is an output of a sigmoid and basically gating your cells with a multiplicative interaction so if f is 0 you will shut off the cell and reset the counter

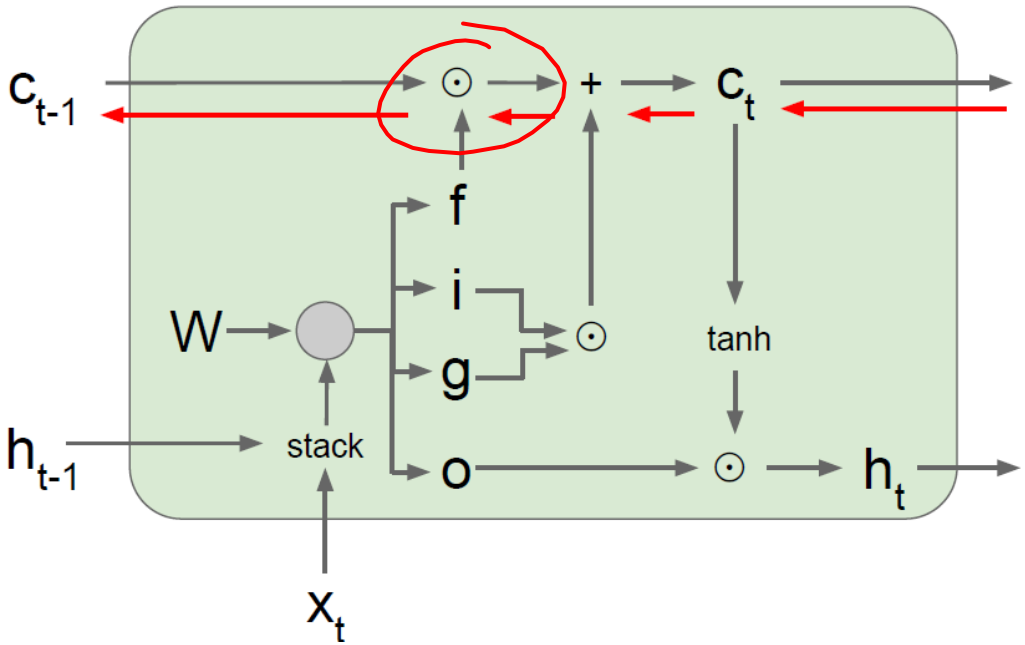
i ⊙ g part is basically adding to the cell state then the cell state leaks into the hidden state but it only leaks through tanh. It is gated by the *o* vector that decides which parts of the cell state to actually reveal into the hidden cell. This hidden state not only goes to the next iteration of the LSTM but it also actually would flow up to higher layers because this is the hidden state vector that we actually end up plugging into further LSTMs above or goes into a prediction.

Source: Stanford CS231n: Convolutional Neural Networks for Visual Recognition.

Long Short-Term Memory (LSTM)

Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]



Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no matrix multiply by W

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

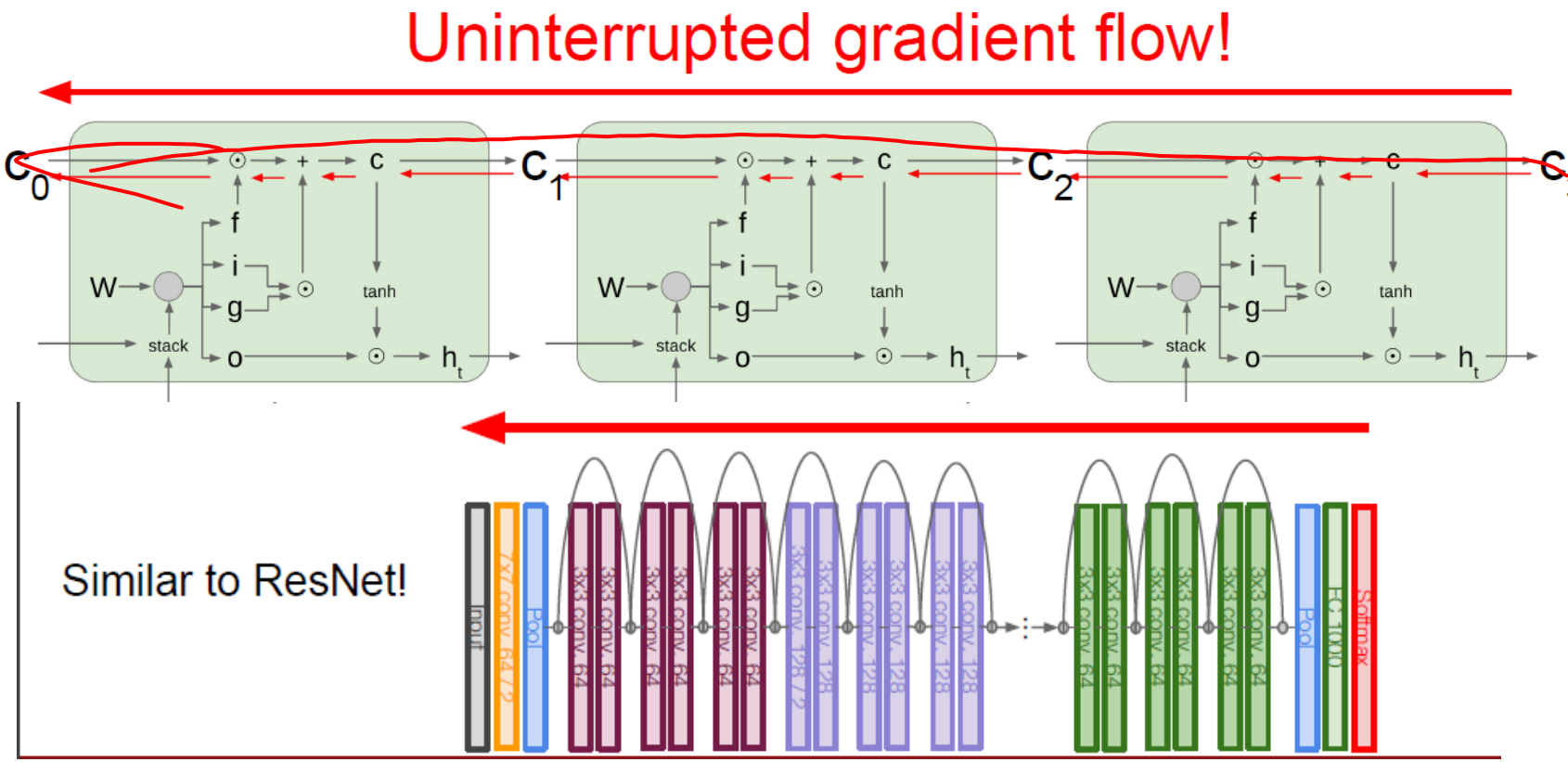
$$c_t = f \odot c_{t-1} + i \odot g$$

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

Source: Stanford CS231n: Convolutional Neural Networks for Visual Recognition.

Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]



Source: Stanford CS231n: Convolutional Neural Networks for Visual Recognition.

Unsupervised training on video

– Auto-encoder model

Multilayer Long Short Term Memory (LSTM) networks to *learn representations of video sequences*.

The representation can be used to perform different tasks, such as

- **reconstructing the input sequence**
- predicting the future sequence
- action recognition (supervised)

Srivastav et. al., 2014: Unsupervised Learning of Video Representation using LSTMs

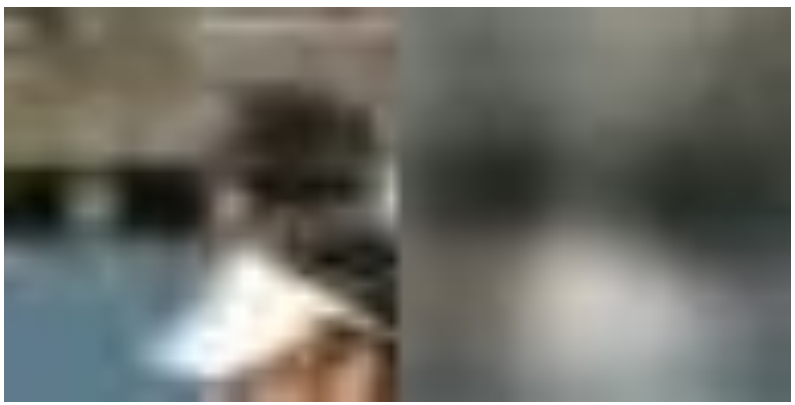
Unsupervised training on video

– Auto-encoder model

We use multilayer Long Short Term Memory (LSTM) networks to learn representations of video sequences.

The representation can be used to perform different tasks, such as

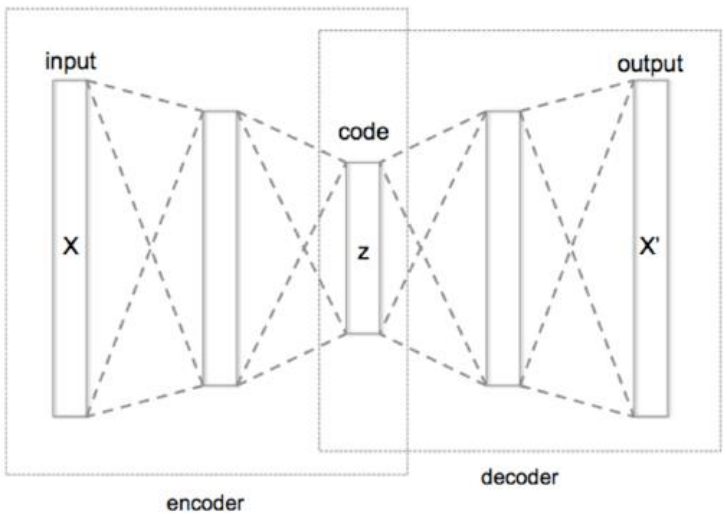
- reconstructing the input sequence
- **predicting the future sequence**
- action recognition (supervised)

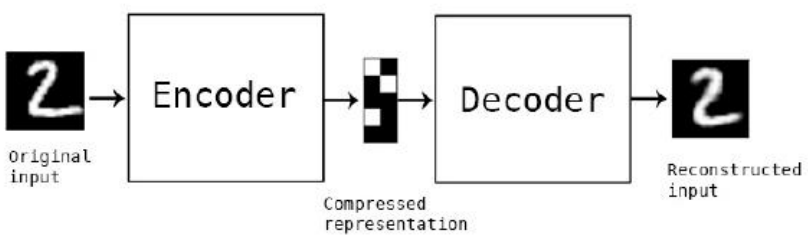


Srivastav et. al., 2014: Unsupervised Learning of Video Representation using LSTMs

Unsupervised training on video

– Auto-encoder model

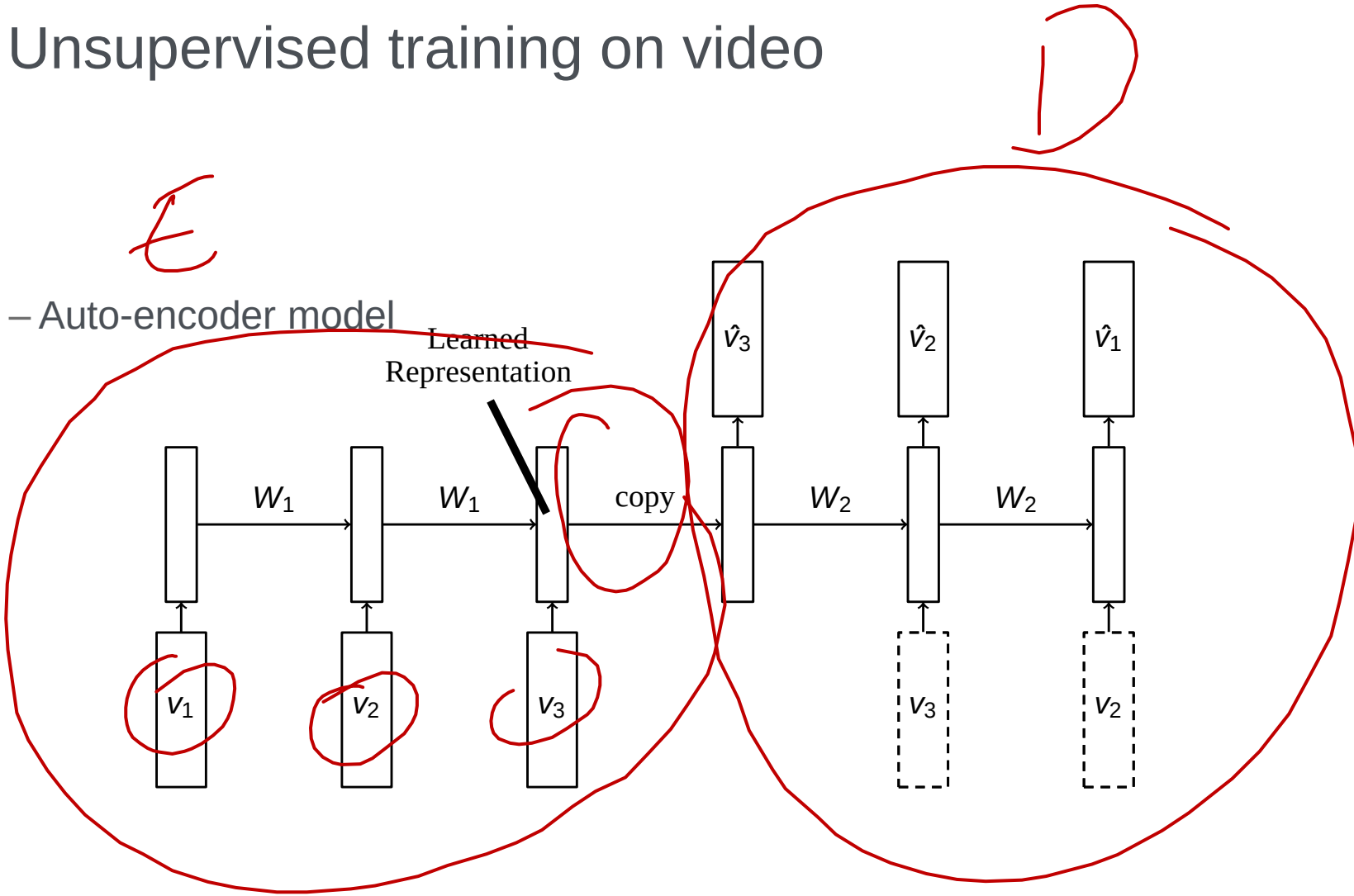




Srivastav et. al., 2014: Unsupervised Learning of Video Representation using LSTMs

Unsupervised training on video

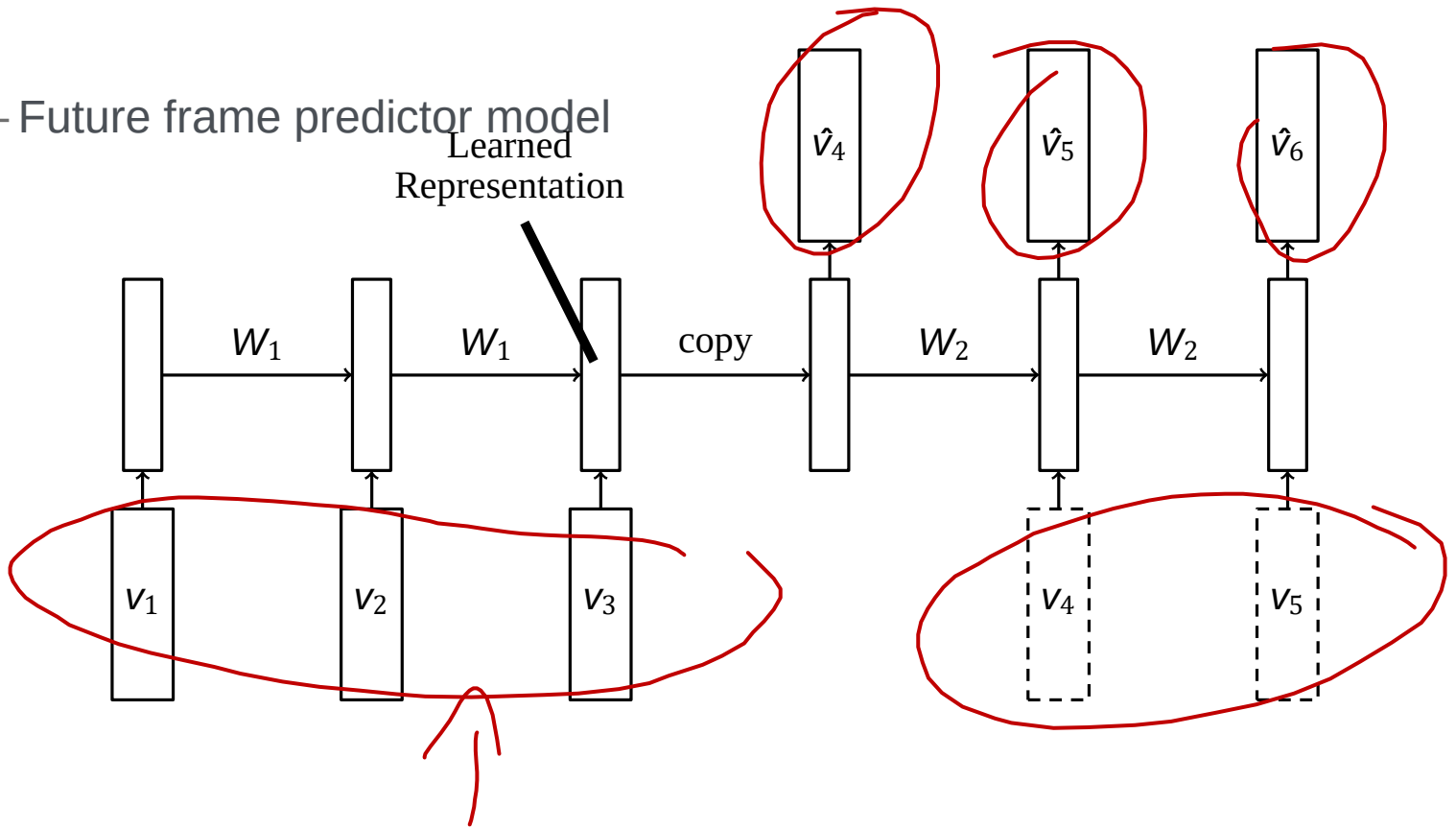
– Auto-encoder model



Srivastav et. al., 2014: Unsupervised Learning of Video Representation using LSTMs

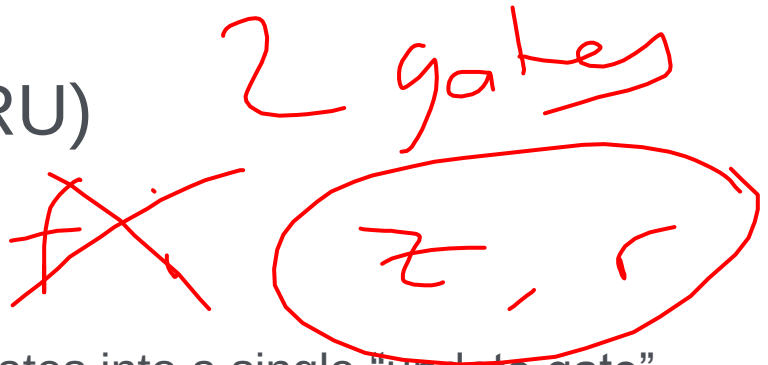
Unsupervised training on video

– Future frame predictor model



Srivastav et. al., 2014: Unsupervised Learning of Video Representation using LSTMs

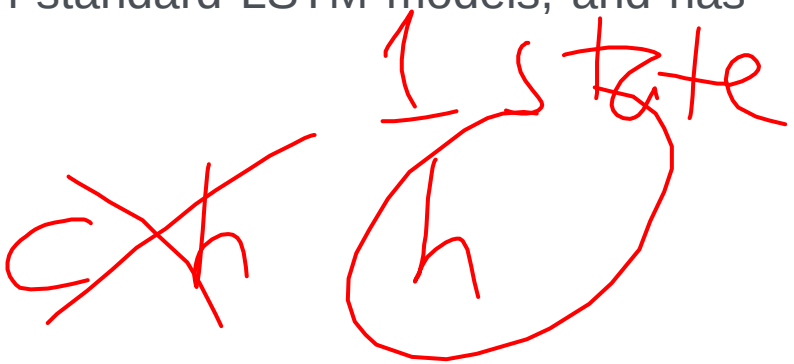
Gated Recurrent Units (GRU)



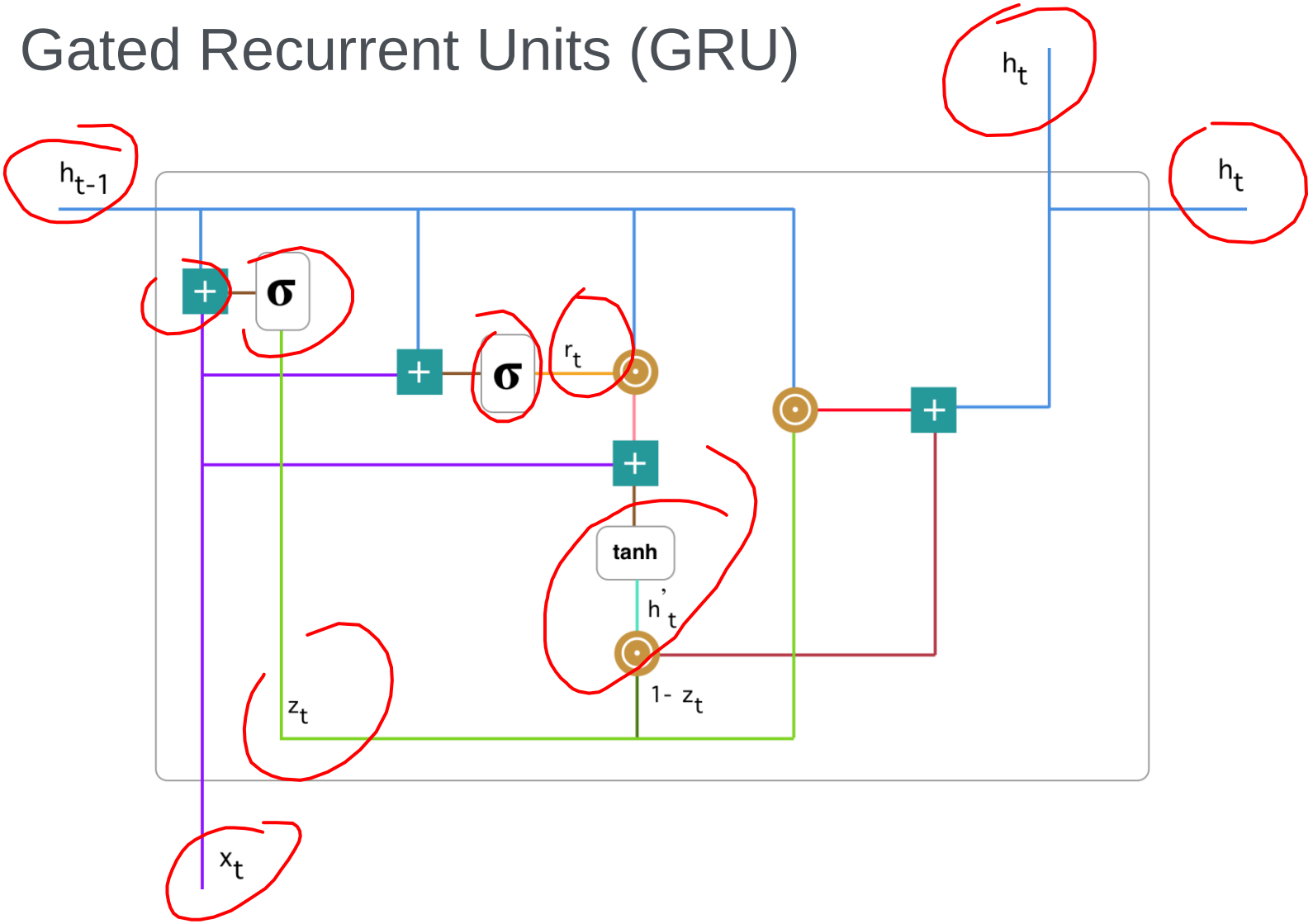
- GRU is a change on the LSTM:
  - It combines the forget and input gates into a single "update gate"
  - It also merges the cell state and hidden state

- Implementation wise there is only a single hidden state vector in forward pass as opposed to two vectors

- The resulting model is simpler than standard LSTM models, and has been growing increasingly popular.

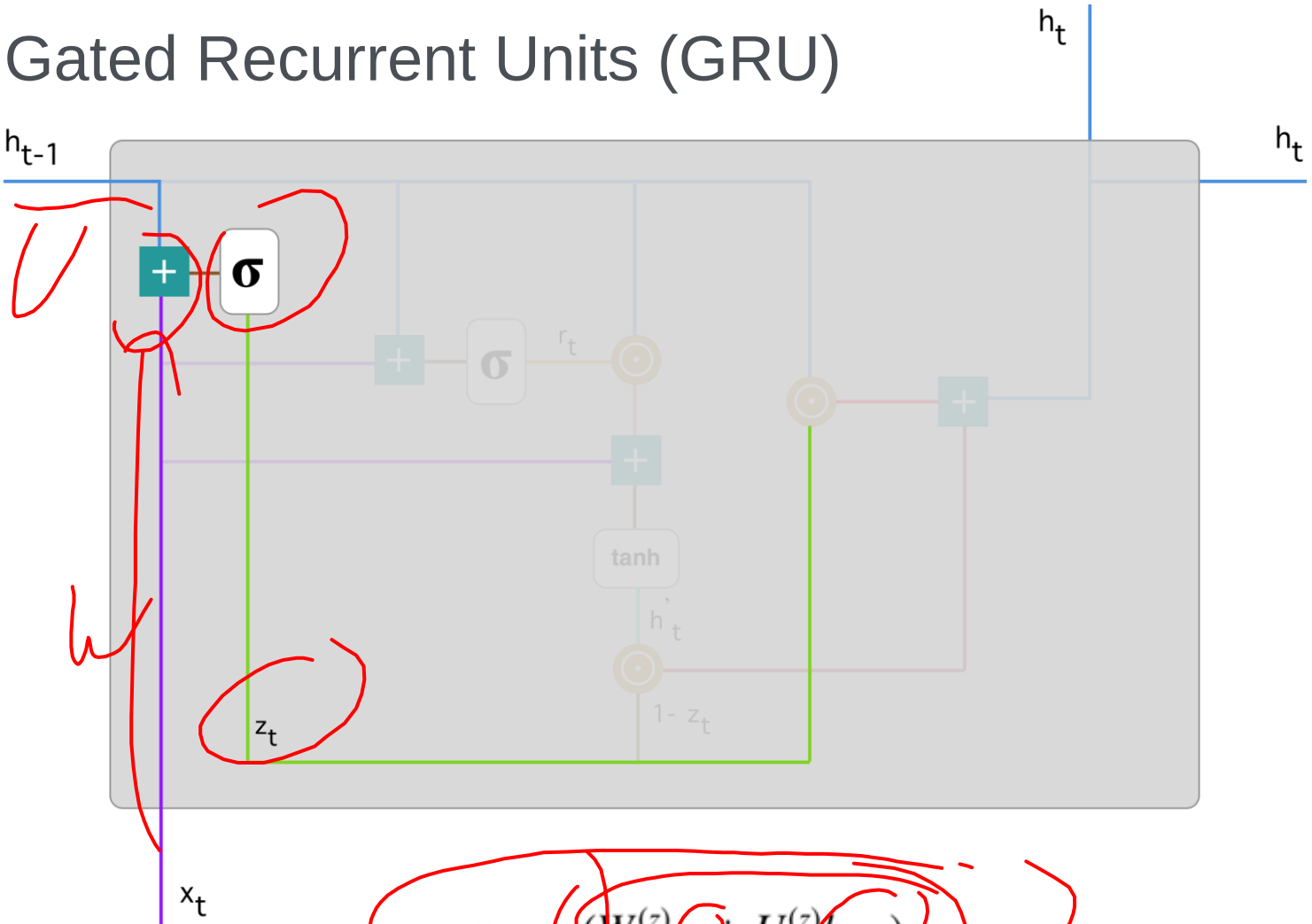


Gated Recurrent Units (GRU)



https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be

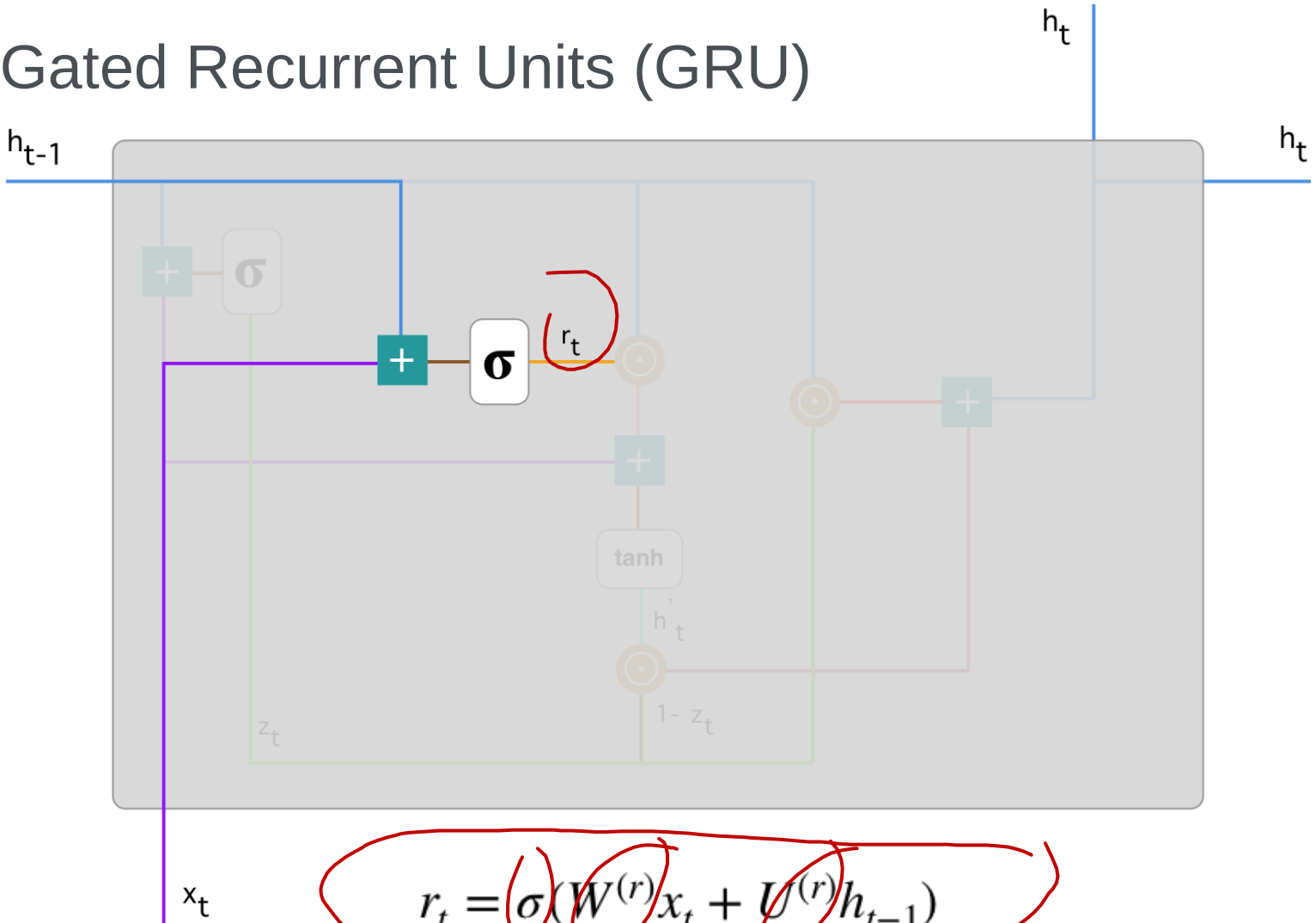
Gated Recurrent Units (GRU)



$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1})$$

https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be

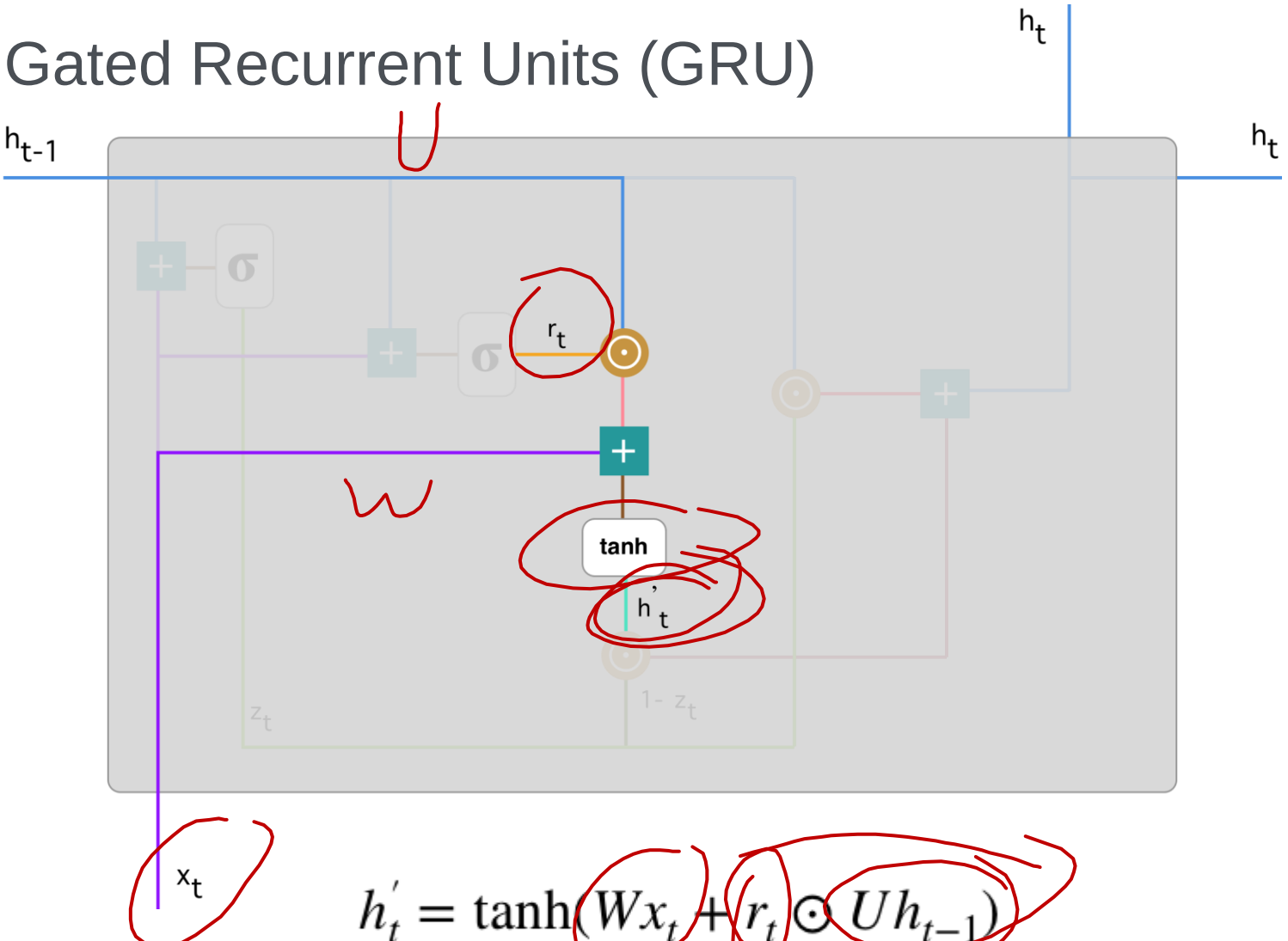
Gated Recurrent Units (GRU)



$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1})$$

https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be

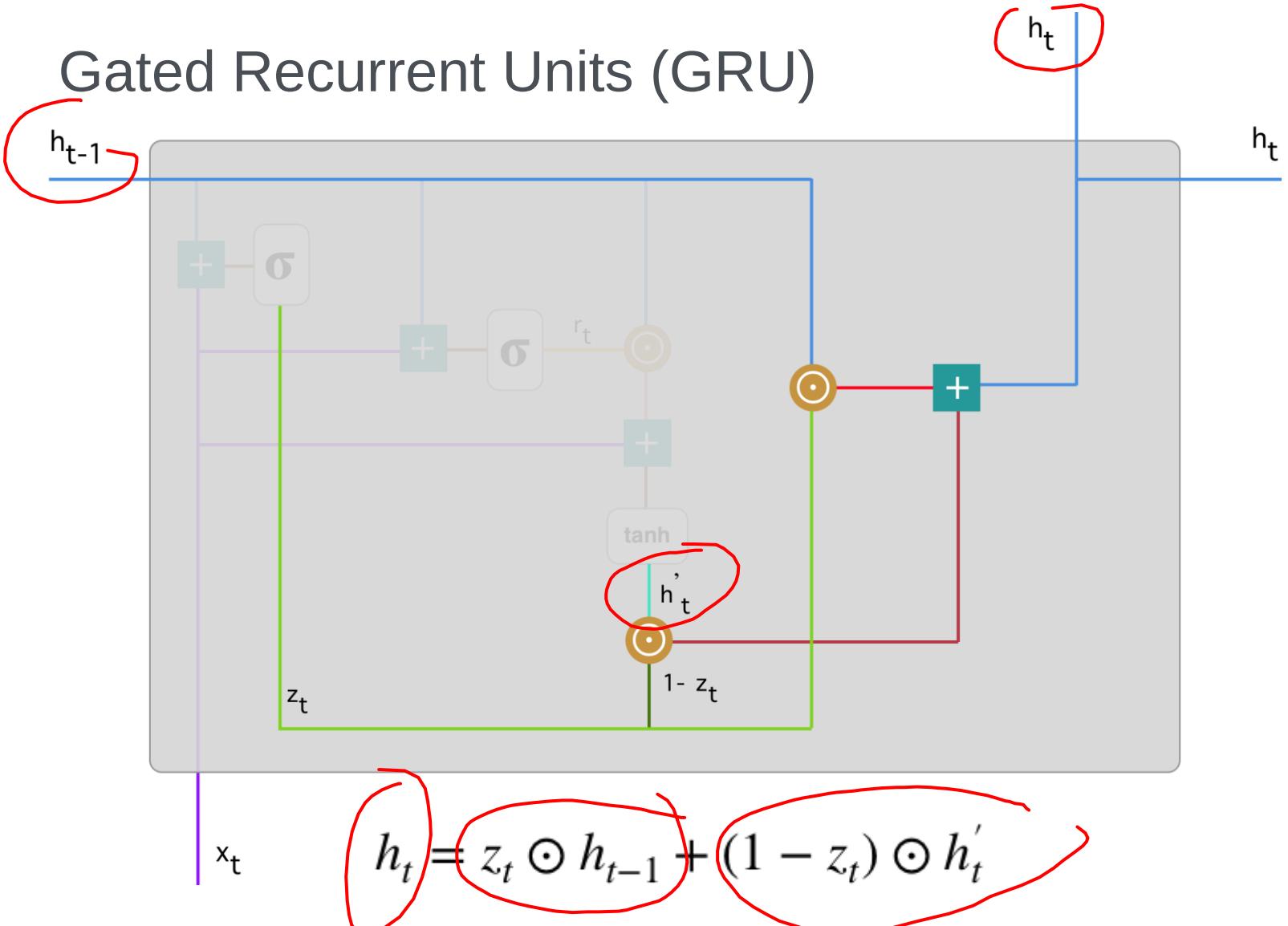
Gated Recurrent Units (GRU)



$$h_t' = \tanh(Wx_t + r_t \odot Uh_{t-1})$$

https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be

Gated Recurrent Units (GRU)



$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h'_t$$

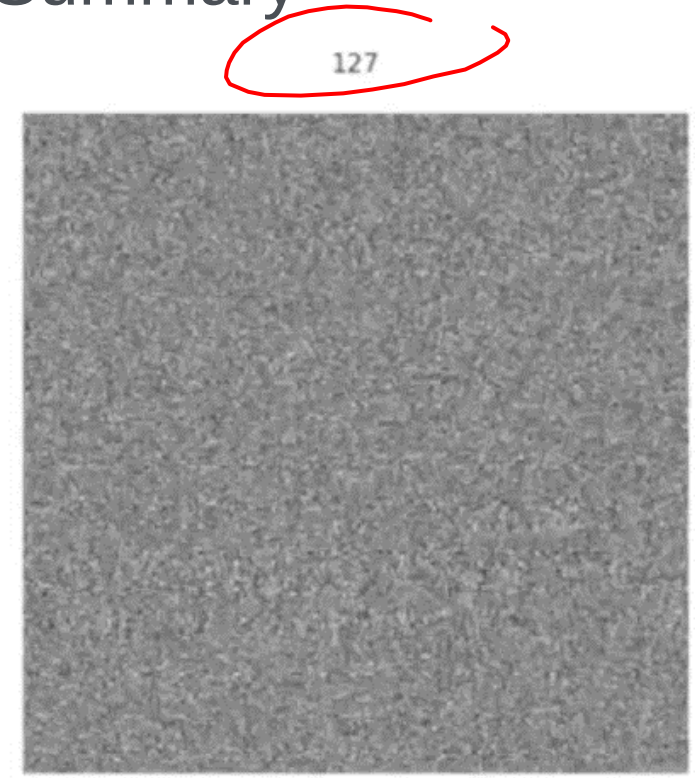
https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be

LSTM: A search space odyssey

- The first large-scale analysis of eight LSTM variants on three representative tasks: speech recognition, handwriting recognition, and polyphonic music modeling.
- The hyperparameters of all LSTM variants for each task were optimized separately using random search, and their importance was assessed using the powerful fANOVA framework.
- In total, the results of 5400 experimental runs (≈15 years of CPU time), is summarized.
- Results show that:
  - none of the variants can improve upon the standard LSTM architecture significantly,
  - the forget gate and the output activation function are its most critical components.
  - the studied hyperparameters are virtually independent

Greff, K., Srivastava, R.K., Koutník, J., Steunebrink, B.R. and Schmidhuber, J., 2017. LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems*, *28*(10), pp.2222-2232.

Summary



RNN vs LSTM gradients on the input weight matrix

Error is generated at 128th step and propagated back. No error from other steps.
At the beginning of training weights sampled from Normal Distribution in (-0.1, 0.1).

Summary

- Raw RNN does not actually work very well —> use LSTMs or GRUs instead
  - having additive interactions allow gradients to flow much better and prevents vanishing gradient problem
  - we still have to worry a bit about the exploding gradient problem so it's common to clip these gradients
- Better understanding of LSTMs is needed
  - We're not fully understanding yet: why that works so well and which parts of it work well
  - We need a much better understanding both theoretical and empirical way
- Better and simpler architectures are desirable- a hot research topic-