Programming of Neural Networks

Today: Recurrent Neural Networks

Why do we need it?

- So far we have investigated **static** problems, that is we the shapes of the tensors in advance!
- But what if for example, we deal with text (author prediction)?

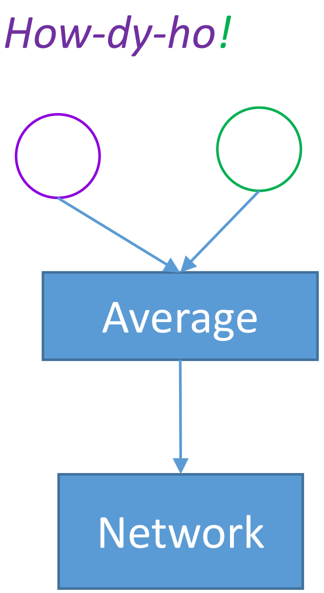
How-dy-ho! *(Mr. Hankey)*

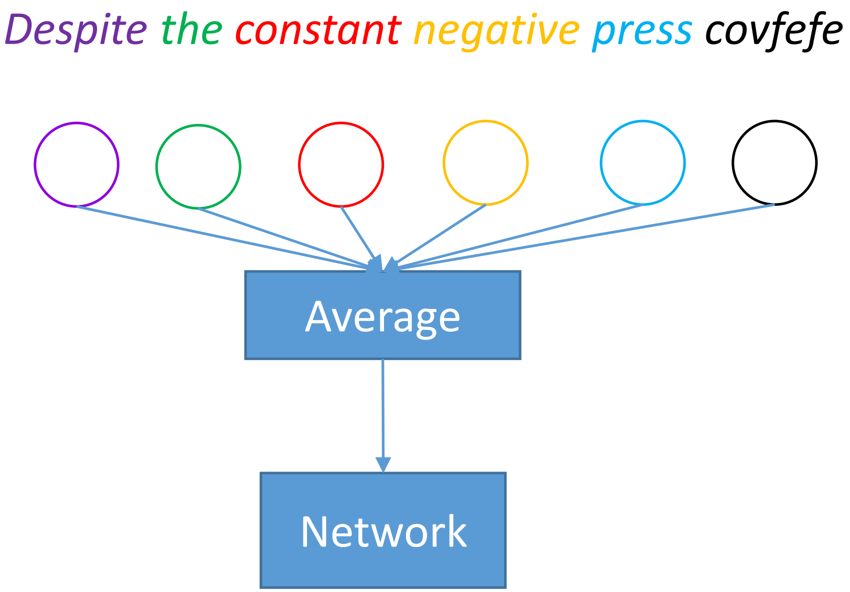
Vs

Despite the constant negative press covfefe

A simple (but bad) solution

- Map every token to a vector of 100 numbers, and average it

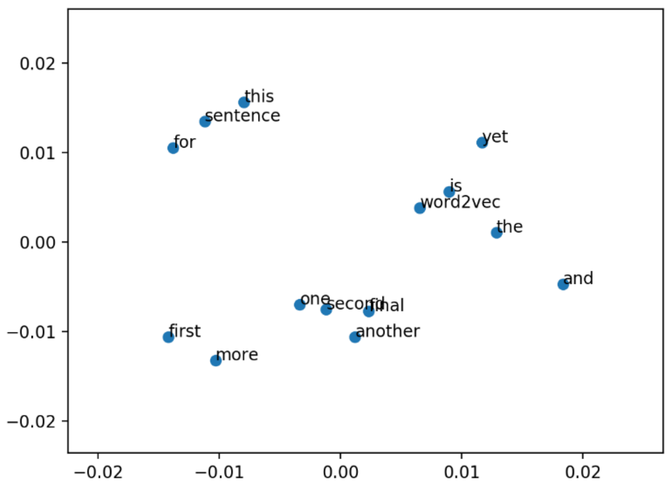




Why is this a bad solution?

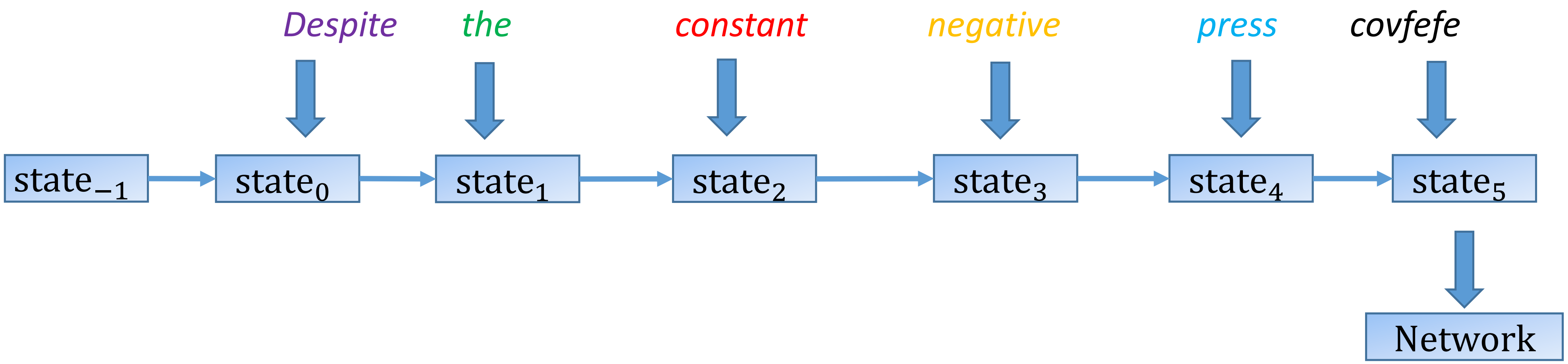
- The uniqueness of a token gets completely destroyed
- The order of the tokens is not taken into accout („Bag of words“)
- The amount of tokens is ignored

➔ What would be a better solution?



Motivation: Recurrent Networks

- Instead of simply averaging the tokens, we could iteratively update a **state** and use the final state into our network



➔ We call this structure a **Recurrent Structure**

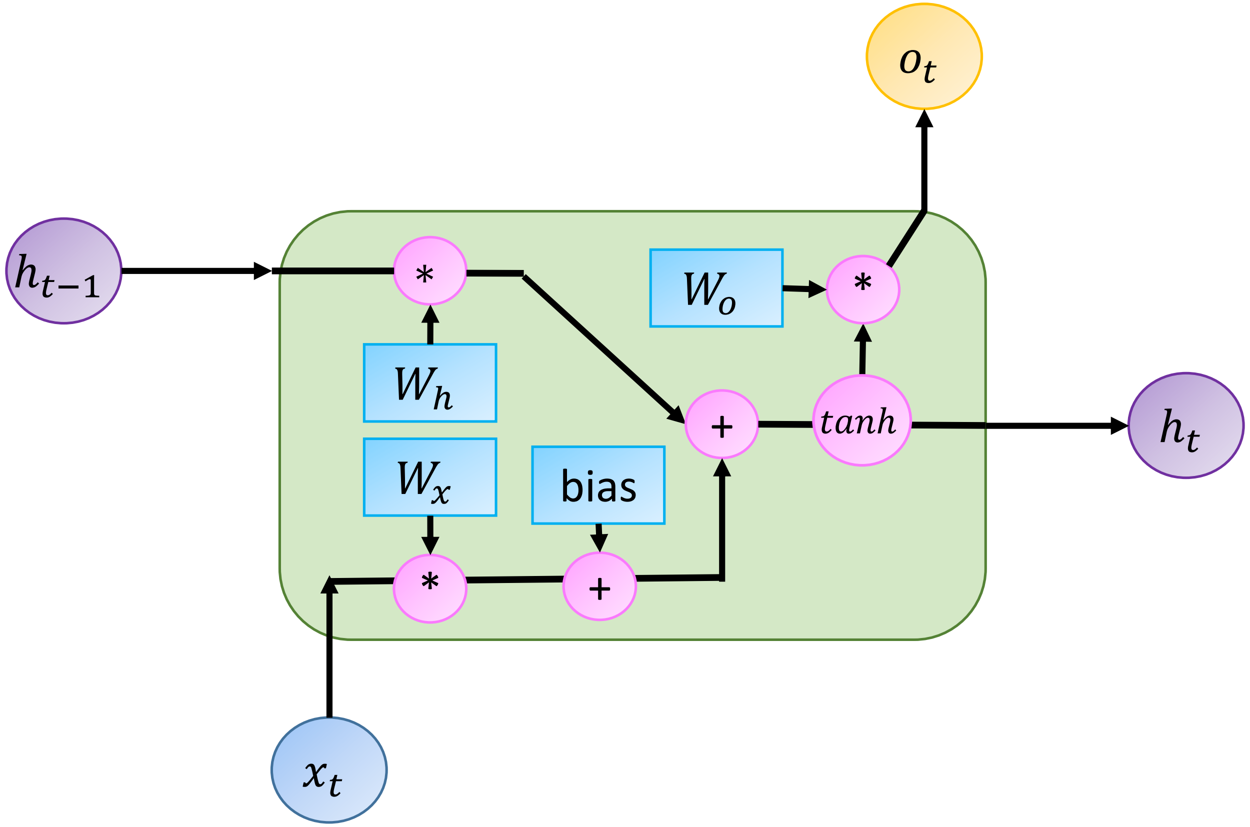
Modelling the state update

- While it is rather clear that the state can be modelled as a vector, how to update the internal state remains unclear

➔ How can we force a machine to compress (and store) the important information for a task in a single vector?

- Many different approaches, e.g.
  - Vanilla Recurrent Cell
  - Gated Recurrent Unit (GRU)
  - Long Short Term Memory (LSTM)

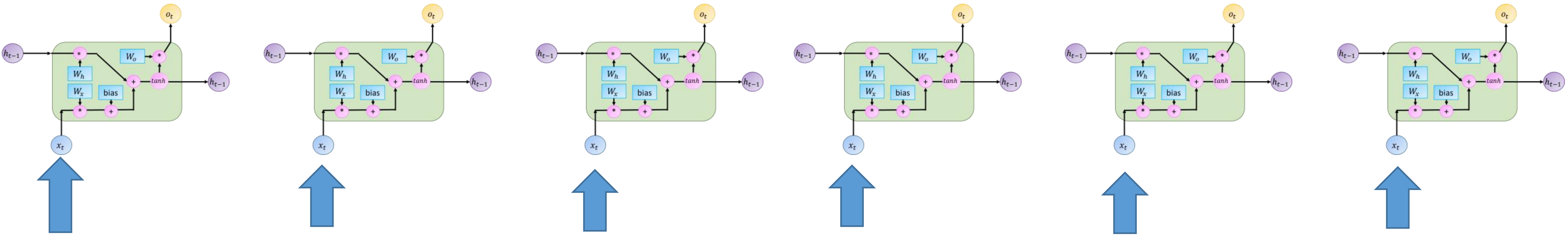
The Vanilla Recurrent Cell (one possibility)



$$h_t = \tanh(h_{t-1} \cdot W_h + x_t \cdot W_x + bias)$$
$$o_t = h_t \cdot W_o$$

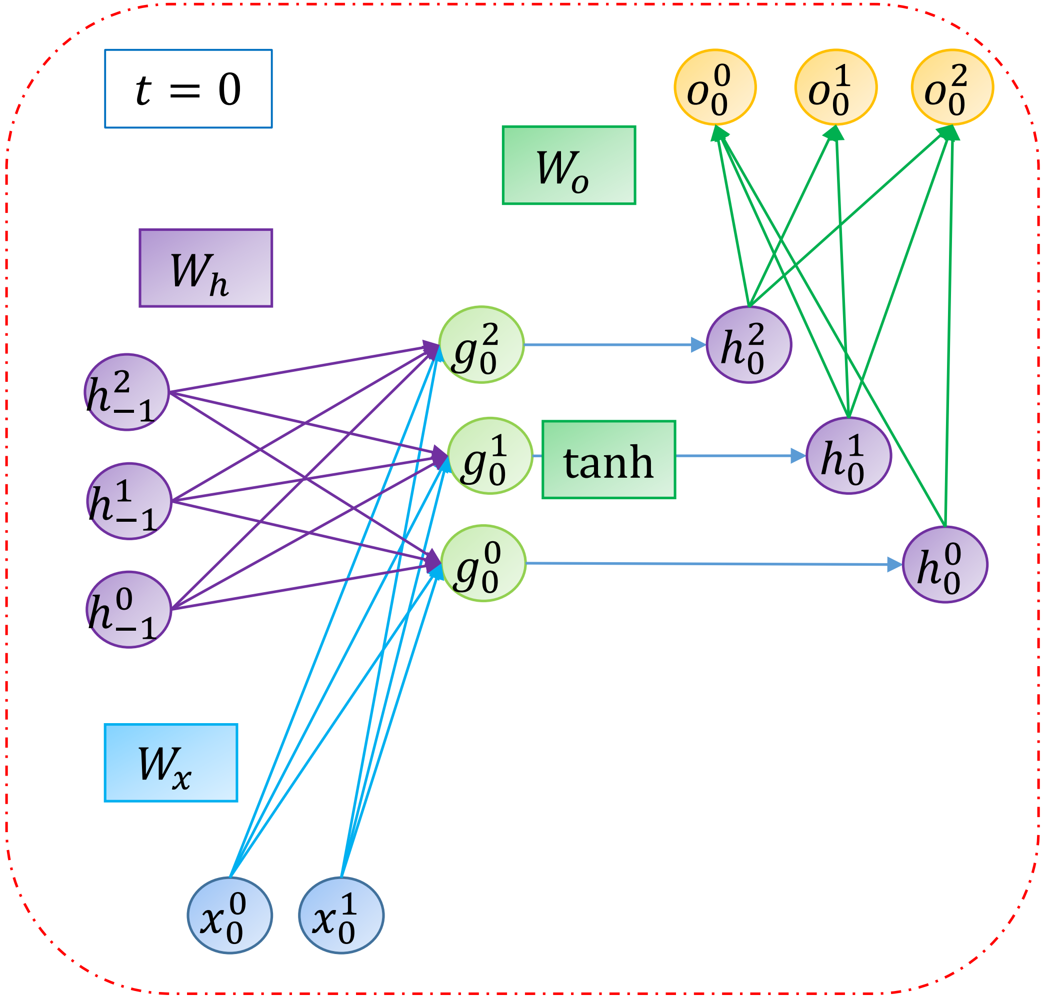
The Vanilla Recurrent Cell in action

We can apply the Recurrent Cell by **unrolling** (copy + pasting) it as many times as required

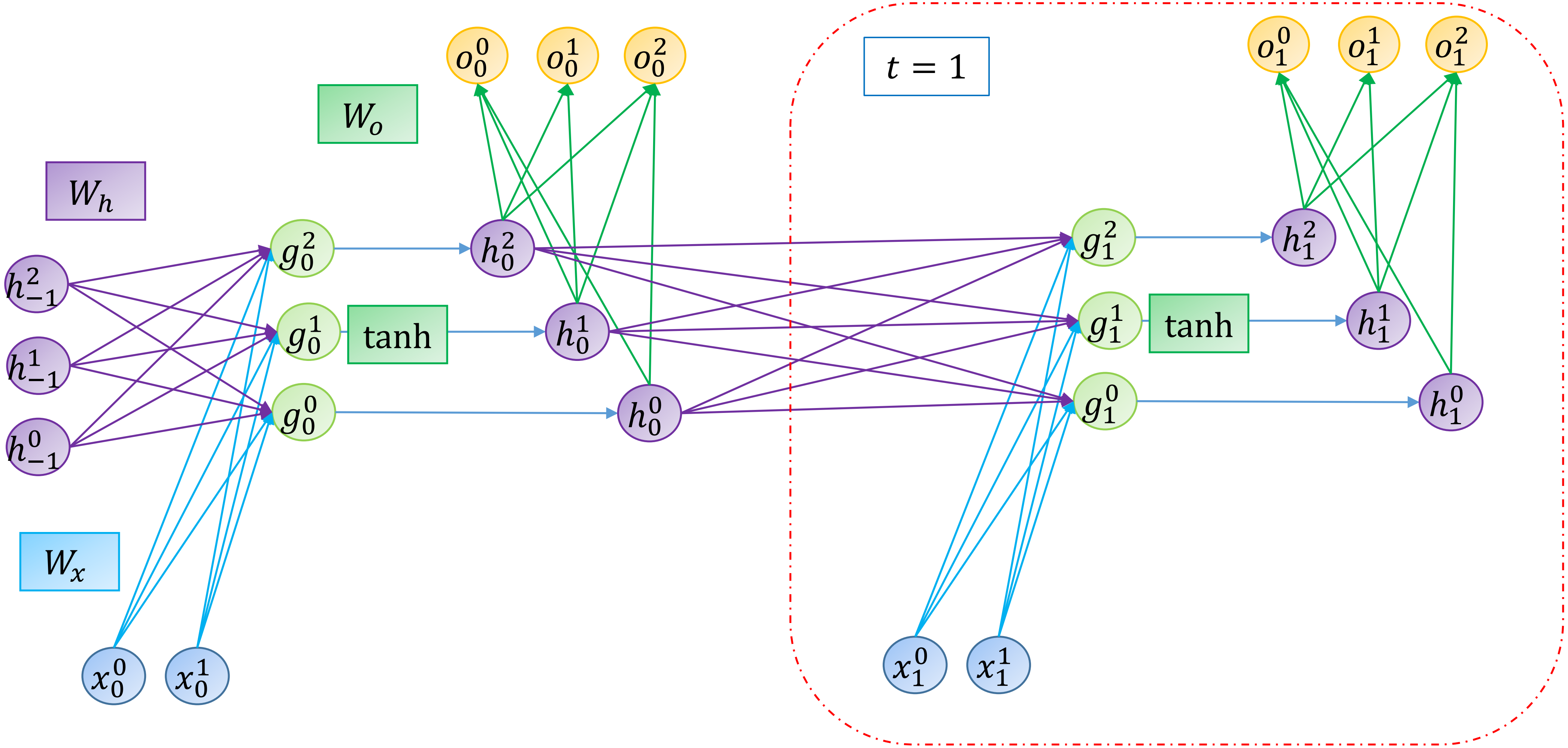


Despite *the* *constant* *negative* *press* *covfefe*

Vanilla Recurrent Under the hood



Vanilla Recurrent Under the hood



Vanilla Recurrent: The framework

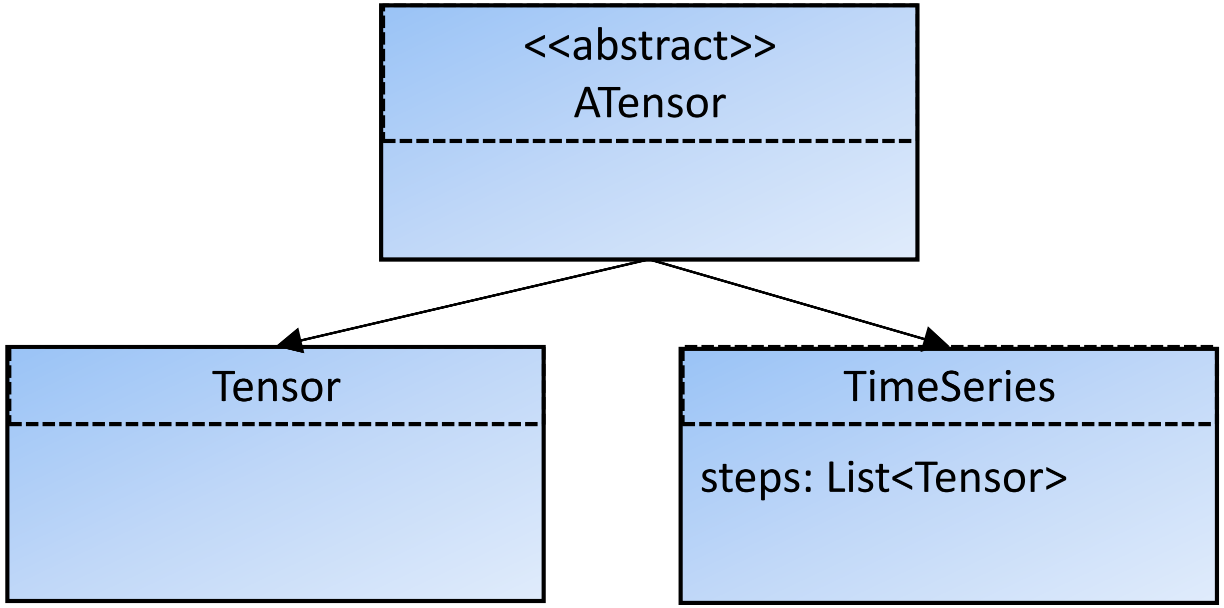
- As its input it requires a „Timeseries“, which is a list of tensors that is naturally ordered by time

- Its output is either:
  - a single Tensor (just the last output)
  - A Timeseries of Tensors (a list of all outputs)

Extending the Tensor class

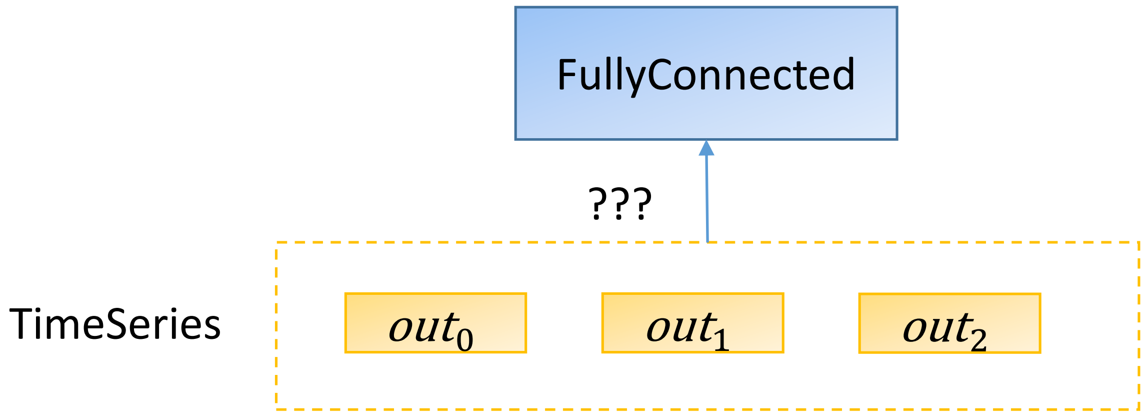
- A tensor is now either:
  - A regular Tensor
  - or a TimeSeries

- No further changes to the layer interfaces



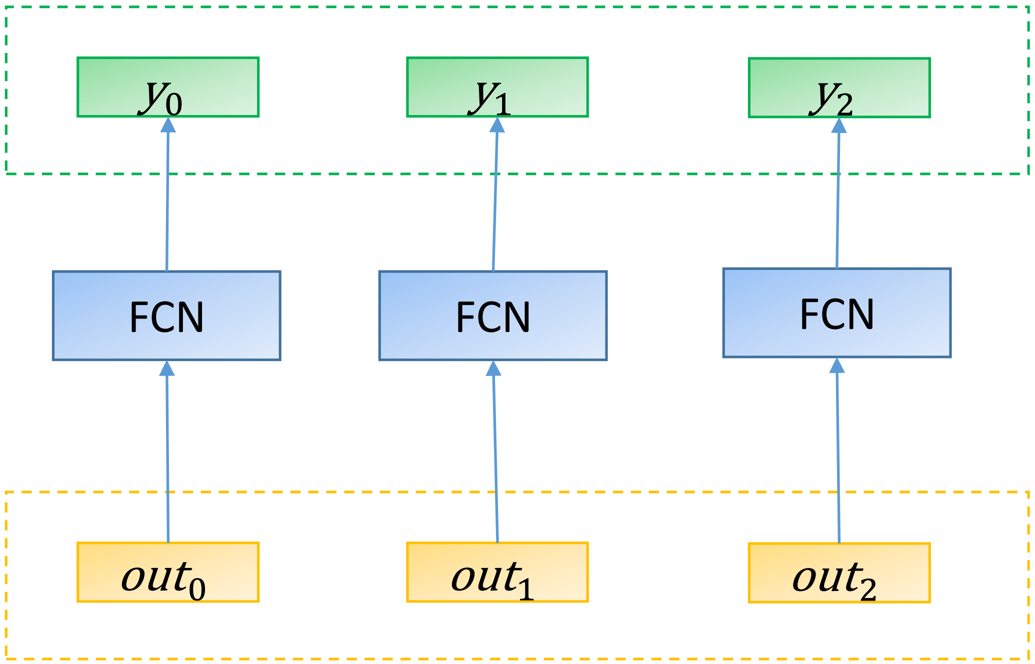
Implications to the framework

- What should (for example) a Fully Connected Layer do with a TimeSeries?



Implications to the framework

➔ It is applied (with the same parameters) to **every tensor** in the time series separately, and hence produces another **TimeSeries**



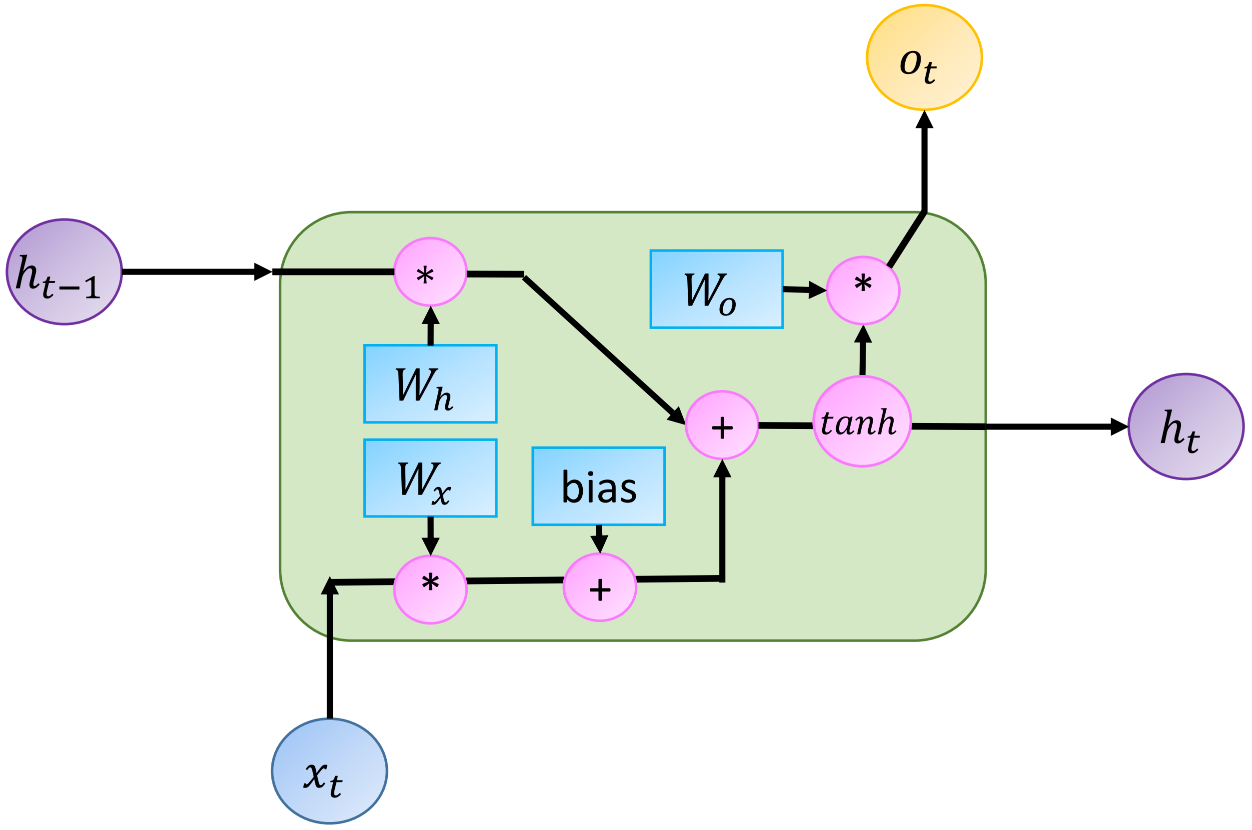
Implications to the framework

- Integrating a TimeSeries into the framework, forces you to rework quite a bit of code, since every layer needs to decide what it is going to do with it!

Vanilla Recurrent: Parameters

W_h: recurrent weights, shape [h,h]
W_x: input weights, shape [x,h]
W_o: output weights, shape [h,out]
bias:, shape [h]

➔ Hidden states have to be stored somewhere clever, so that you got access to all of them during the backward pass!



Vanilla Recurrent: The math

- Forward pass is straight forward (pun intended)

$$g_t = h_{t-1} \cdot W_h + x_t \cdot W_x + bias$$
$$h_t = \tanh(g_t)$$
$$o_t = h_t \cdot W_o$$

- h_{-1} is initialized as a zero Tensor of according size

Vanilla Recurrent: Backward

- We have to derive 4 equations:

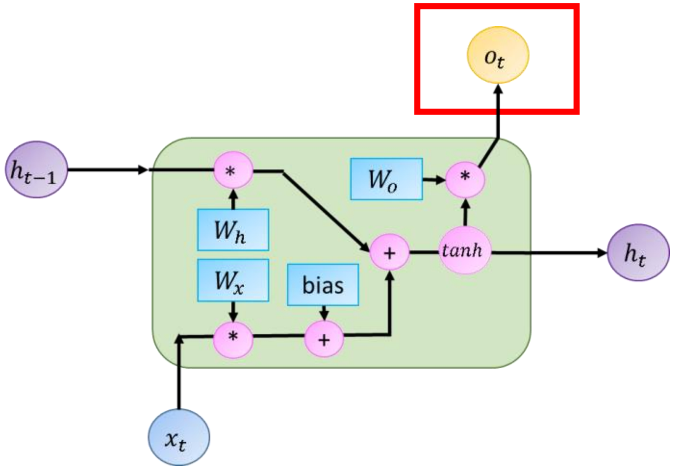
$$\delta o = ???$$
$$\delta h = ???$$
$$\delta g = ???$$
$$\delta x = ???$$

Why a delta for the input x ???

Vanilla Recurrent: Backward

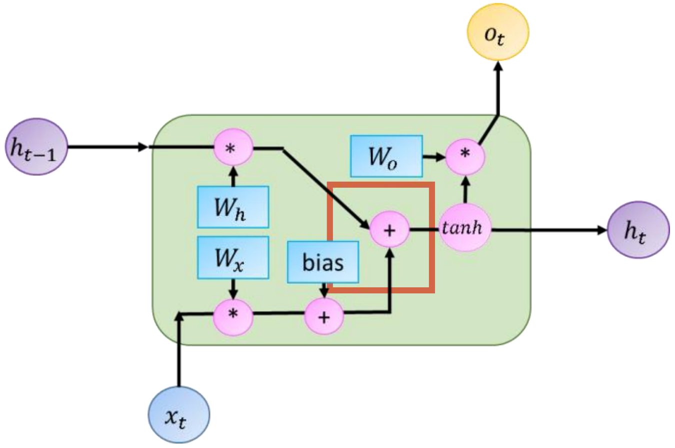
- We have to derive 4 equations δo_t:

- δo_t: These are the deltas that are passed from above



Vanilla Recurrent: Backward

- We have to derive 4 equations δg_t:



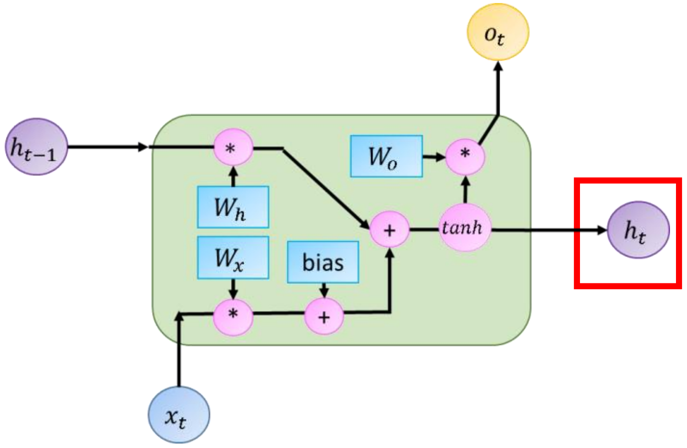
$$h_t = \tanh(g_t)$$

$$\frac{\partial L}{\partial g_t} = \frac{\partial L}{\partial h_t} \cdot \frac{\partial h_t}{\partial g_t}$$

$$\delta g_t = \delta h_t \cdot (1 - (\tanh g_t)^2)$$

$$= \delta h_t \cdot (1 - h_t^2)$$

Vanilla Recurrent: Backward



- We have to derive 4 equations δh_t:
- h is involved in 2 equations:

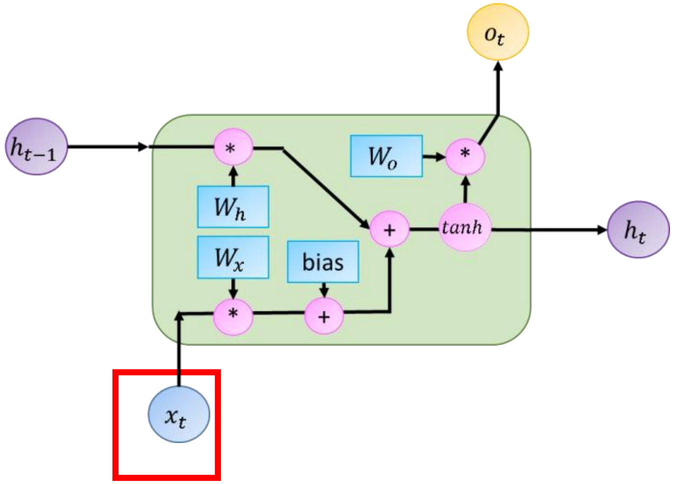
$$g_t = h_{t-1} \cdot W_h + x_t \cdot W_x + bias$$
$$o_t = h_t \cdot W_o$$

$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial o_t} \cdot \frac{\partial o_t}{\partial h_t} + \frac{\partial L}{\partial g_{t+1}} \cdot \frac{\partial g_{t+1}}{\partial h_t}$$

$$\delta h_t = \delta o_t \cdot W_o^T + \delta g_{t+1} \cdot W_h^T$$

Vanilla Recurrent: Backward

- We have to derive 4 equations δx_t:



$$g_t = h_{t-1} \cdot W_h + x_t \cdot W_x + bias$$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial g_t} \cdot \frac{\partial g_t}{\partial x_t}$$

$$\delta x_t = \delta g_t \cdot W_x^T$$

Vanilla Recurrent: Backward

- Backward equations:

$$\delta o_t \text{:from above}$$

$$\delta h_t = \delta o_t \cdot W_o^T + \delta g_{t+1} \cdot W_h^T$$

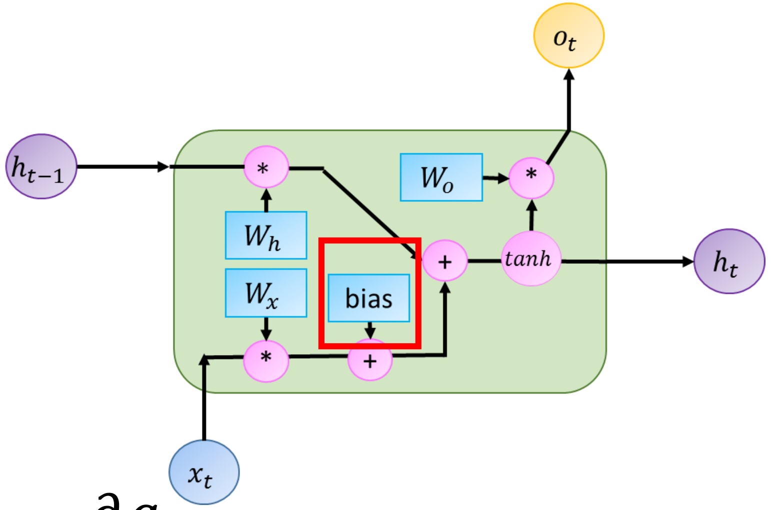
$$\delta g_t = \delta h_t \cdot (1 - h_t^2)$$

$$\delta x_t = \delta g \cdot W_x^T$$

- h_{-1} is initialized as a zero Tensor of according size

Vanilla Recurrent: Weight updates

- We have to derive 4 equations for the weights:



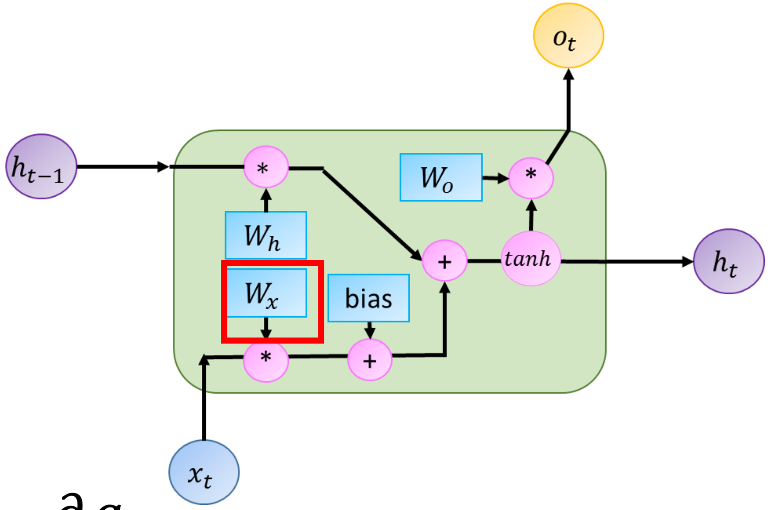
$$g_t = h_{t-1} \cdot W_h + x_t \cdot W_x + bias$$

$$\frac{\partial L}{\partial bias} = \frac{\partial L}{\partial g_t} \cdot \frac{\partial g_t}{\partial bias} \text{ ➔ weight sharing ➔} = \sum_t \frac{\partial L}{\partial g_t} \cdot \frac{\partial g_t}{\partial bias}$$

$$\Delta bias = \sum_t \delta g_t$$

Vanilla Recurrent: Weight updates

- We have to derive 4 equations for the weights:

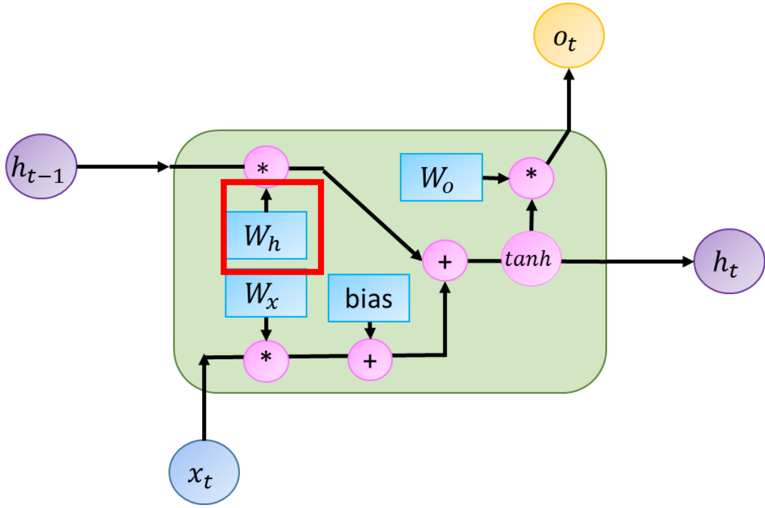


$$g_t = h_{t-1} \cdot W_h + x_t \cdot W_x + bias$$

$$\frac{\partial L}{\partial W_x} = \frac{\partial L}{\partial g_t} \cdot \frac{\partial g_t}{\partial W_x} \rightarrow \text{weight sharing} \rightarrow = \sum_t \frac{\partial L}{\partial g_t} \cdot \frac{\partial g_t}{\partial W_x}$$

$$\Delta W_x = \sum_t x_t^T \cdot \delta g_t$$

Vanilla Recurrent: Weight updates



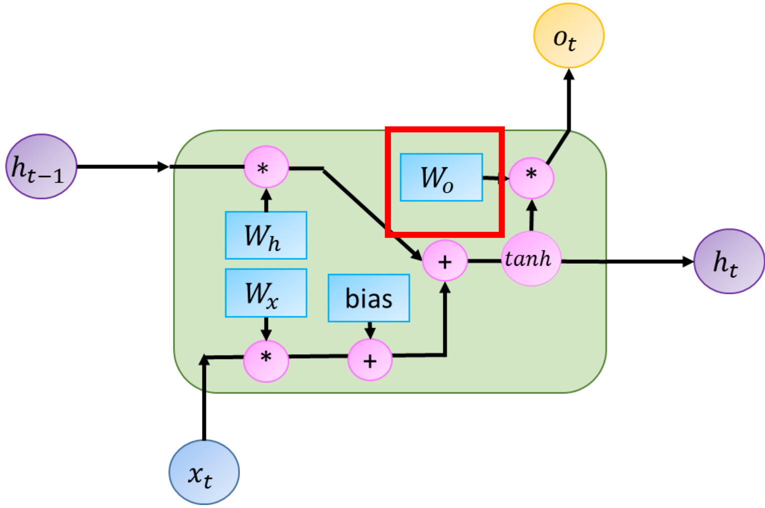
- We have to derive 4 equations for the weights:

$$g_t = h_{t-1} \cdot W_h + x_t \cdot W_x + bias$$

$$\frac{\partial L}{\partial W_h} = \frac{\partial L}{\partial g_t} \cdot \frac{\partial g_t}{\partial W_h} \text{ ➔ weight sharing ➔} = \sum_{t=1}^{|steps|} \frac{\partial L}{\partial g_t} \cdot \frac{\partial g_t}{\partial W_x}$$

$$\Delta W_h = \sum_{t=1} h_{t-1}^T \cdot \delta g_t$$

Vanilla Recurrent: Weight updates



- We have to derive 4 equations for the weights:

$$o_t = h_t \cdot W_o$$

$$\frac{\partial L}{\partial W_o} = \frac{\partial L}{\partial o_t} \cdot \frac{\partial p_t}{\partial W_o}$$

$$\Delta W_o = \sum_{t=0} h_t^T \cdot \delta o_t$$

Vanilla Recurrent: Weight Updates

- Weight Updates:

$$\Delta W_h = \sum_{t=1}^{|steps|} h_{t-1}^T \cdot \delta g_t$$

$$\Delta W_x = \sum_t x_t^T \cdot \delta g_t$$

$$\Delta bias = \sum_t \delta g_t$$

$$\Delta W_o = \sum_{t=0} h_t^T \cdot \delta o_t$$

Problems of the Vanilla Cell

- During forward pass we did not direct the network about how it should update the internal state

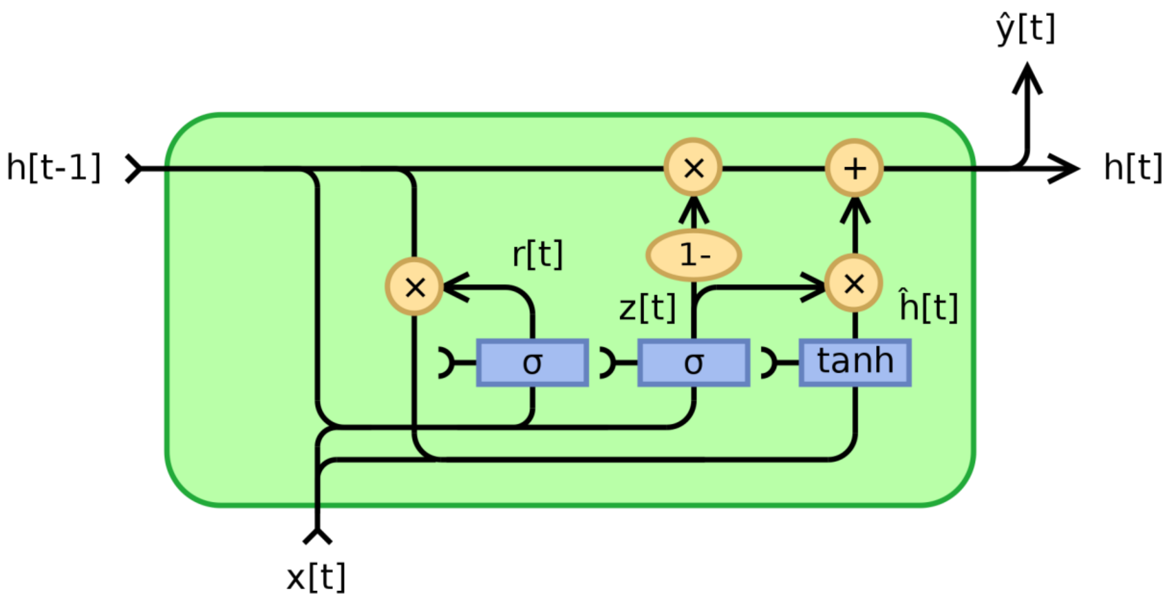
$$g_t = h_{t-1} \cdot W_h + x_t \cdot W_x + bias$$
$$h_t = \tanh(g_t)$$
$$o_t = h_t \cdot W_o$$

➔Can we direct it a bit so that we can integrate a controlled update and reset of the internal state?

➔Also there is an issue called „Exponential Weight Decay“

Gated Recurrent Unit (GRU)

- Instead of a single gate g, it makes use of 2 gates, r and z



Source: Wikipedia: GRU

$$z'_t = h_{t-1} \cdot W_{Rz} + x_t \cdot W_z + b_z$$
$$z_t = \sigma(z'_t)$$

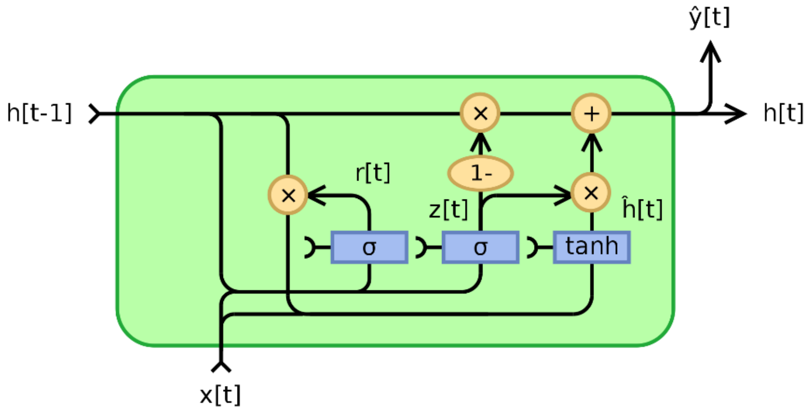
$$r'_t = h_{t-1} \cdot W_{Rr} + x_t \cdot W_r + b_r$$
$$r_t = \sigma(r'_t)$$

$$f_t = r_t \cdot h_{t-1}$$

$$o'_t = f_t \cdot W_{Ro} + x_t \cdot W_o + b_o$$
$$o_t = \tanh(o'_t)$$

$$h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot o_t$$

Gated Recurrent Unit (GRU)



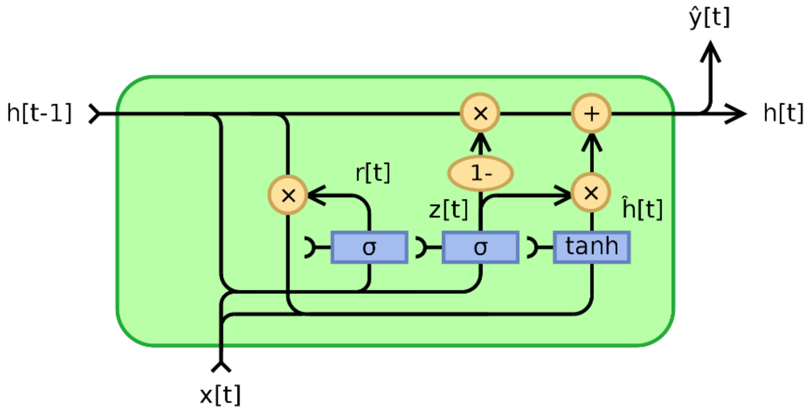
$$z'_t = h_{t-1} \cdot W_{Rz} + x_t \cdot W_z + b_z$$
$$z_t = \sigma(z'_t)$$

Input-gates: Similar to g in the vanilla state, they capture different characteristics of the input and the last state

$$r'_t = h_{t-1} \cdot W_{Rr} + x_t \cdot W_r + b_r$$
$$r_t = \sigma(r'_t)$$

Source: Wikipedia: GRU

Gated Recurrent Unit (GRU)



$$f_t = r_t \cdot h_{t-1}$$

Which entries of the last state should remain and which should be forgotten

$$o'_t = f_t \cdot W_{Ro} + x_t \cdot W_o + b_o$$
$$o_t = \tanh(o'_t)$$

Ratio of „What to keep“ and „What to add“

$$h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot o_t$$

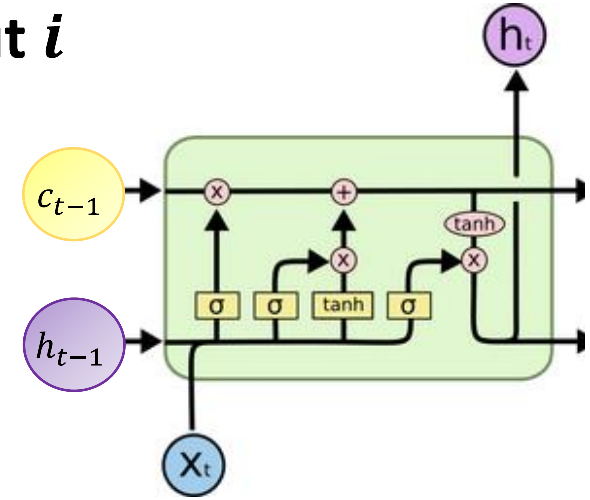
Source: Wikipedia: GRU

Gated Recurrent Unit (GRU): Backward

- Your exercise! Should be fairly easy by now

Long Short Term Memory (LSTM)

- Instead of 2 Gates, involved in the GRU, it uses 3:
  - **Forget *f***
  - **Output *o***
  - **Input *i***



Source: Colahs Blog

$$f_t' = h_{t-1} \cdot W_{Rf} + x_t \cdot W_f + b_\mathrm{f}$$
$$f_t = \sigma(f_t')$$

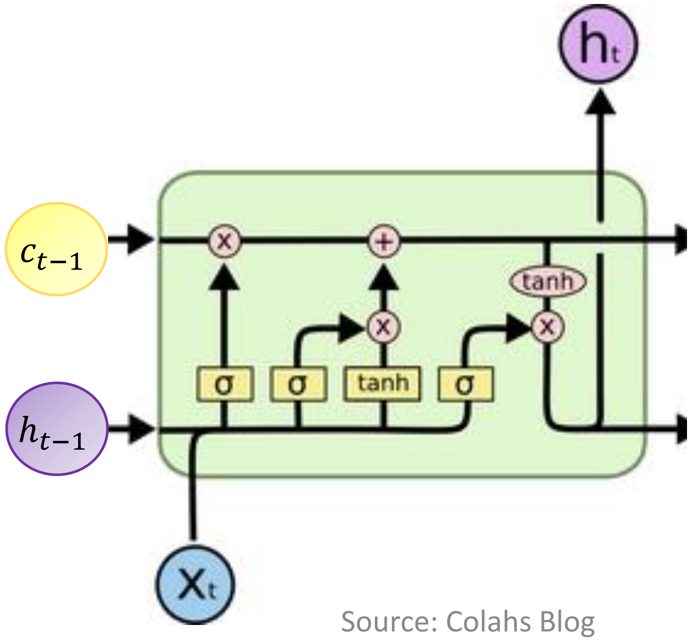
$$o'_t = h_{t-1} \cdot W_{Ro} + x_t \cdot W_o + b_\mathrm{o}$$
$$o_t = \sigma(o_t')$$

$$i'_t = h_{t-1} \cdot W_{Ri} + x_t \cdot W_i + b_\mathrm{i}$$
$$i_t = \sigma(i_t')$$

$$a'_t = h_{t-1} \cdot W_{Ra} + x_t \cdot W_a + b_\mathrm{a}$$
$$a = \tanh(a_t')$$

Long Short Term Memory (LSTM)

- It also introduces an additional internal state c_t which is also updated in every time step



Source: Colahs Blog

$$c_t = a_t \cdot i_t + f_t \cdot c_{t-1}$$

$$h_t = \tanh(c_t) \cdot o_t$$
$$out_t = h_t$$

LSTM: Forward

$$a'_t = h_{t-1} \cdot W_{Ra} + x_t \cdot W_a + b_\mathrm{a}$$
$$a = \tanh(a'_t)$$

$$f'_t = h_{t-1} \cdot W_{Rf} + x_t \cdot W_f + b_\mathrm{f}$$
$$f_t = \sigma(f'_t)$$

$$o'_t = h_{t-1} \cdot W_{Ro} + x_t \cdot W_o + b_\mathrm{o}$$
$$o_t = \sigma(o'_t)$$

$$i'_t = h_{t-1} \cdot W_{Ri} + x_t \cdot W_i + b_\mathrm{i}$$
$$i_t = \sigma(i'_t)$$

$$c_t = a_t \cdot i_t + f_t \cdot c_{t-1}$$

$$h_t = \tanh(c_t) \cdot o_t$$
$$out_t = h_t$$

LSTM: Backward (Derivation on the board)

$$\delta h_t = \delta a'_{t+1} \cdot W_{Ra}^T + \delta i'_{t+1} \cdot W_{Ri}^T + \delta o'_{t+1} \cdot W_{Ro}^T + \delta f'_{t+1} \cdot W_{Rf}^T + \delta out_t$$
$$\delta c_t = \delta h_t \cdot o_t \cdot (1 - \tanh(c_t)) + \delta c_{t+1} \cdot f_{t+1}$$
$$\delta a_t = \delta c_t \cdot i_t$$
$$\delta i_t = \delta c_t \cdot a_t$$
$$\delta f_t = \delta c_t \cdot c_{t-1}$$
$$\delta o_t = \delta h_t \cdot \tanh(c_t)$$

$$\delta a'_t = \delta a_t \cdot (1 - {a_t}^2)$$
$$\delta i'_{\mathrm{i}} = \delta i_t \cdot i_t(1 - i_t)$$
$$\delta f'_t = \delta f_t \cdot f_t(1 - f_t)$$
$$\delta o'_t = \delta o_t \cdot o_t(1 - o_t)$$
$$\delta x_t = \delta a'_t \cdot W_a^T + \delta i'_t \cdot W_i^T + \delta o'_t \cdot W_o^T + \delta f'_t \cdot W_f^T$$

LSTM: Weight Updates

Bias	Input-Weights	Recurrent-Weights
$\Delta b_a = \sum_t \delta a_t'$	$\Delta W_a = \sum_t x_t^T \cdot \delta a_t'$	$\Delta W_{Ra} = \sum_{t=1} h_{t-1}^T \cdot \delta a_t'$
$\Delta b_i = \sum_t \delta i_t'$	$\Delta W_i = \sum_t x_t^T \cdot \delta i_t'$	$\Delta W_{Ri} = \sum_{t=1} h_{t-1}^T \cdot \delta i_t'$
$\Delta b_f = \sum_t \delta f_t'$	$\Delta W_f = \sum_t x_t^T \cdot \delta f_t'$	$\Delta W_{Rf} = \sum_{t=1} h_{t-1}^T \cdot \delta f_t'$
$\Delta b_o = \sum_t \delta o_t'$	$\Delta W_o = \sum_t x_t^T \cdot \delta o_t'$	$\Delta W_{Ro} = \sum_{t=1} h_{t-1}^T \cdot \delta o_t'$

Beyond LSTMs

- In recent years, there was critique towards LSTMs , that they (in theory) are capable of capturing information for a long period, but in reality they dont
- The sequential structure can further be exploited by:

➔Introduction of the „Attention Mechanism“

➔Introduction of **Highway Networks**, especially **Recurrent Highway Networks (RHN)**

➔**Conditional Random Fields** to make use of the structure of the classification label as well

➔**Mogrifier LSTM**

Highway Networks

- Highway Networks were introduced in order to ease the training of very deep networks (>100 layer)

- They are very simple in nature and can be considered a replacement for Fully Connected Layers, when training Deep Networks

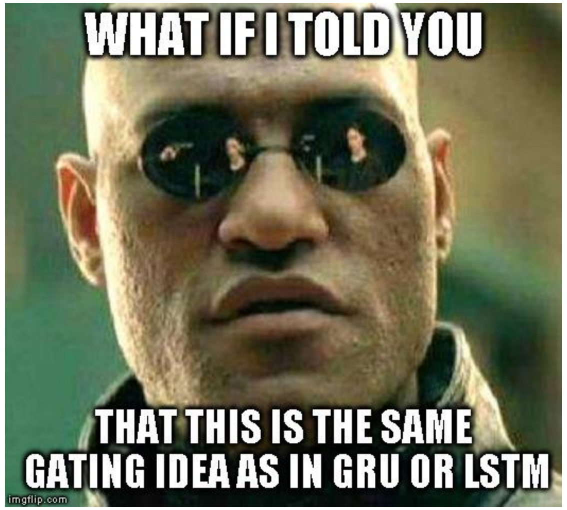
Highway Networks

- Instead of just forwarding $y = x \cdot W + b$ as in Fully Connected layers
- The forward equation is changed to:

$$y = (x \cdot W + b) \cdot T(x) + x \cdot (1 - T(x))$$

With:

$$T(x) = \sigma(x \cdot W_T + b_T)$$



Highway Networks

- Let us introduce $\hat{y}$ to be the forward variable we get from a standard Fully Connected Layer

$$y = \hat{y} \cdot T(x) + x \cdot (1 - T(x))$$

- Since T uses a sigmoid its output is in **[0,1]**

- If $T(x)[i]$ is 0 ➔ $y = x$
- If $T(x)[i]$ is 1 ➔ $y = \hat{y}$
- Else it is a mixture between both!

The signal can simply pass forward in a very quick (High)way

Highway Networks

- You have to pay attention to the shapes! Otherwise the addition does not work !

$$y = \hat{y} \cdot T(x) + x \cdot (1 - T(x))$$

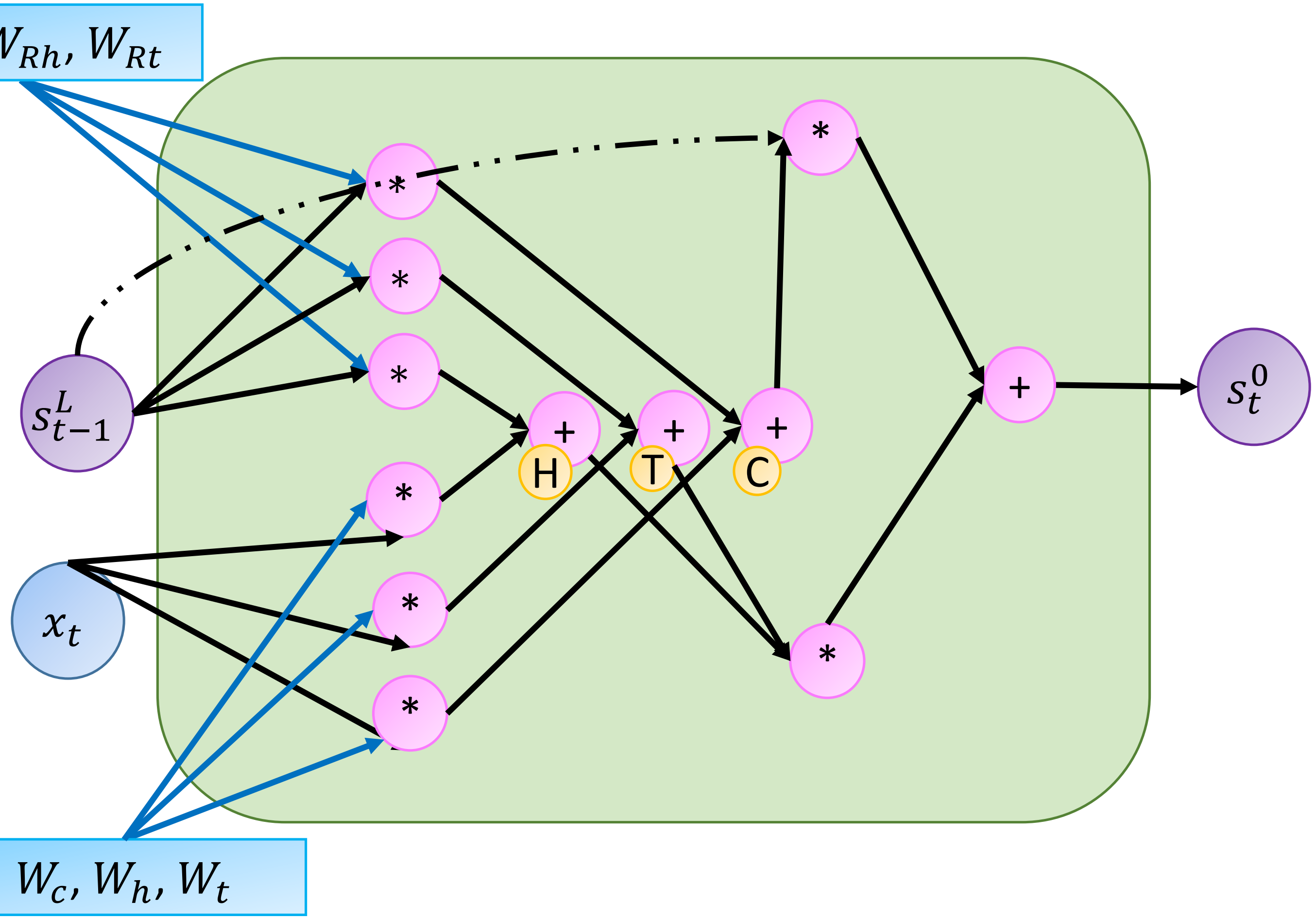
Recurrent Highway Networks (RHN)

- Are the natural extension of a Highway Layer into a Recurrent Form
  - The Highway Layers come with their built-in gating mechanisms anyway!
  - Use **Microcells** and stack these before forwarding to next timestep

- (Note: Bias and activation ignored in the diagram!)

Recurrent Highway Networks (RHN)-Microcell

$$s_t^l = h_t^l \cdot t_t^l + s_t^{l-1} \cdot c_l^t$$



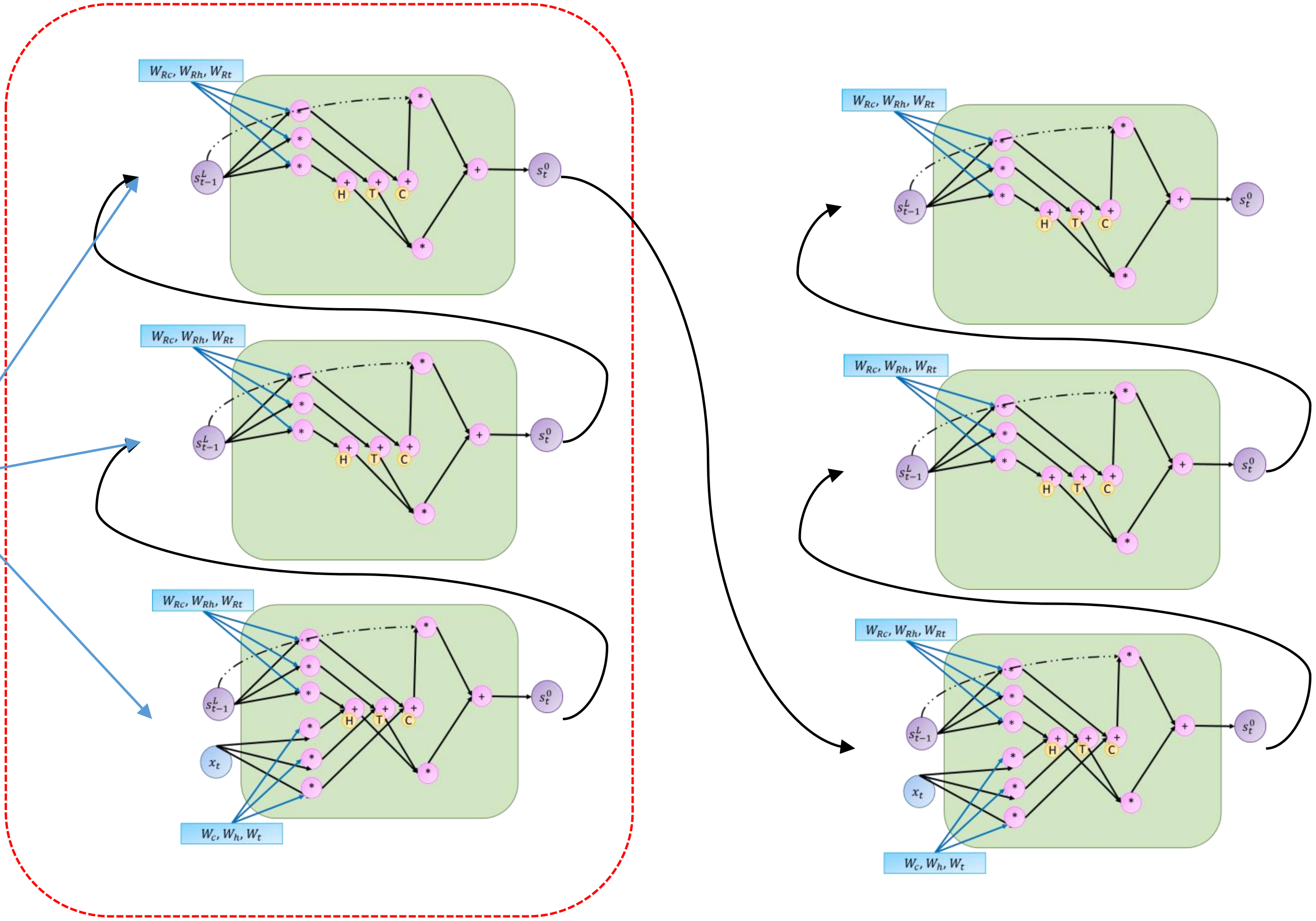
With:

$$h_t^l = \tanh\left(x_t \cdot W_H \cdot \mathbf{1}_{l=0} + s_{l-1} \cdot W_{R_{Hl}} + b_{H_l}\right)$$
$$t_t^l = \sigma\left(x_t \cdot W_T \cdot \mathbf{1}_{l=0} + s_{l-1} \cdot W_{R_{Tl}} + b_{T_l}\right)$$
$$c_t^l = \sigma\left(x_t \cdot W_C \cdot \mathbf{1}_{l=0} + s_{l-1} \cdot W_{R_{Cl}} + b_{C_l}\right)$$

Authors recommend: $c_l^t = (1 - t_t^l)$

Recurrent Highway Layer – Recurrent Cell

Every depth l has different weight matrices!



Recurrent Networks -Summary

- Enables a Neural Network to capture and compress important information of a sequence into a state.

- No clear evidence which Recurrent Cells are the best, but LSTMs are the de facto standard.

- Also there are many variations of the presented cells (e.g. peephole LSTMs, etc…)
- It becomes evident, that manually deriving the backward equations is cumbersome! ➔ Next lecture: **Reverse Mode Automatic Differentiation**