Recurrent Neural Networks: LSTMs and GRUs

Oliver Zeigermann / @DJCordhose

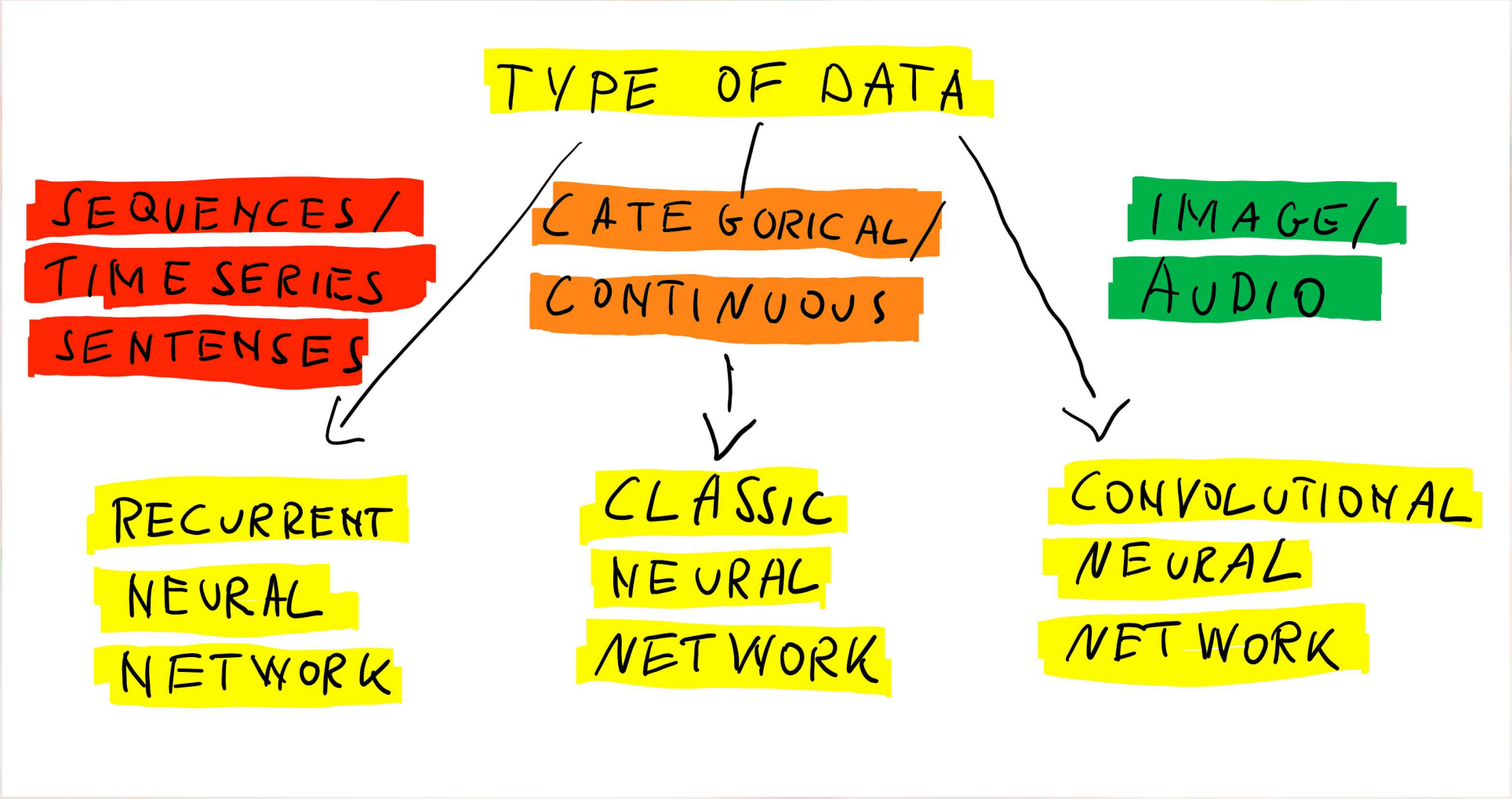
https://djcordhose.github.io/ai/2018_nlp_lstm_gru.html

Material from Stanford

Event	Date	Description	Course Materials
Lecture	Jan 9	Introduction to NLP and Deep Learning [slides]	Suggested Readings: 1. [Linear Algebra Review] 2. [Probability Review] 3. [Convex Optimization Review] 4. [More Optimization (SGD) Review]
Lecture	Jan 11	Word Vectors 1 [slides]	Suggested Readings: 1. [Word2Vec Tutorial - The Skip-Gram Model] 2. [Distributed Representations of Words and Phrases and their Compositionality] 3. [Efficient Estimation of Word Representations in Vector Space]
A1 released	Jan 11	**Assignment #1 released**	[Assignment #1][Written Solutions]
Lecture	Jan 16	Word Vectors 2 [slides]	Suggested Readings: 1. [GloVe: Global Vectors for Word Representation] 2. [Improving Distributional Similarity with Lessons Learned fromWord Embeddings] 3. [Evaluation methods for unsupervised word embeddings]
Lecture	Jan 18	Neural Networks [slides]	Suggested Readings: 1. cs231n notes on [backprop] and [network architectures] 2. [Review of differential calculus] 3. [Natural Language Processing (almost) from Scratch] 4. [Learning Representations by Backpropagating Errors]

http://web.stanford.edu/class/cs224n/

Text and sequences are special



Main issues with RNNs

Vanishing or exploding gradient problem:

- Each step in training applies the same weights to the output, also in back-propagation
- The further we move backwards, the bigger (explodes) or smaller (vanishes) our signal becomes

https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7

Intution of effect

Effectively long term memory does not work:

- RNNs experiences difficulty in memorising words from far away in the sequence
- Predictions based on most recent words only

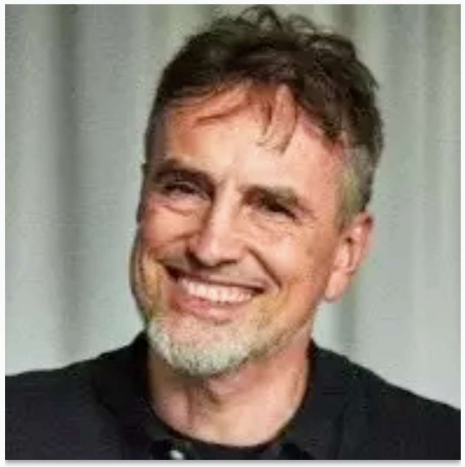
https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7

Creating our very own sentiment analysis

Part III (Preview): Using embeddings to train recurrent networks

Notebook:
https://colab.research.google.com/github/djcordhose/haw/blob/master/notebooks/nlp/2-rnn.ipynb

LSTMs



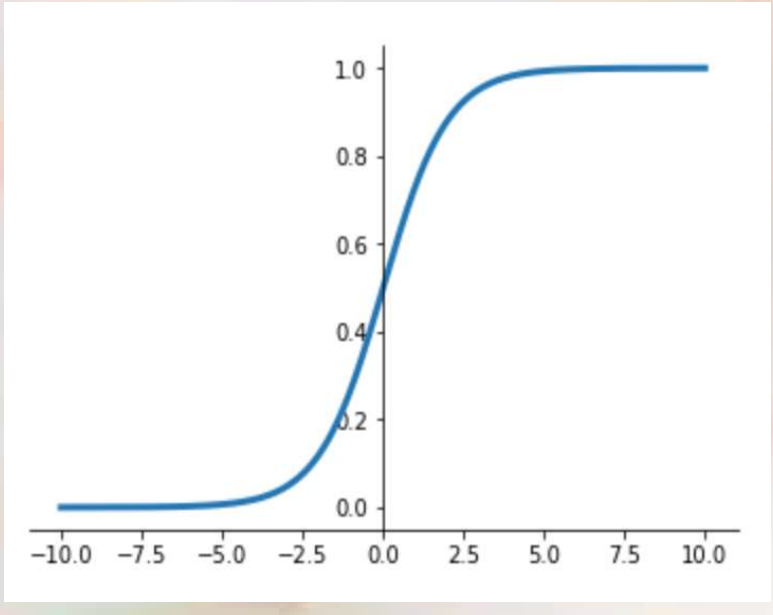
Jürgen Schmidhuber

Father Of AI

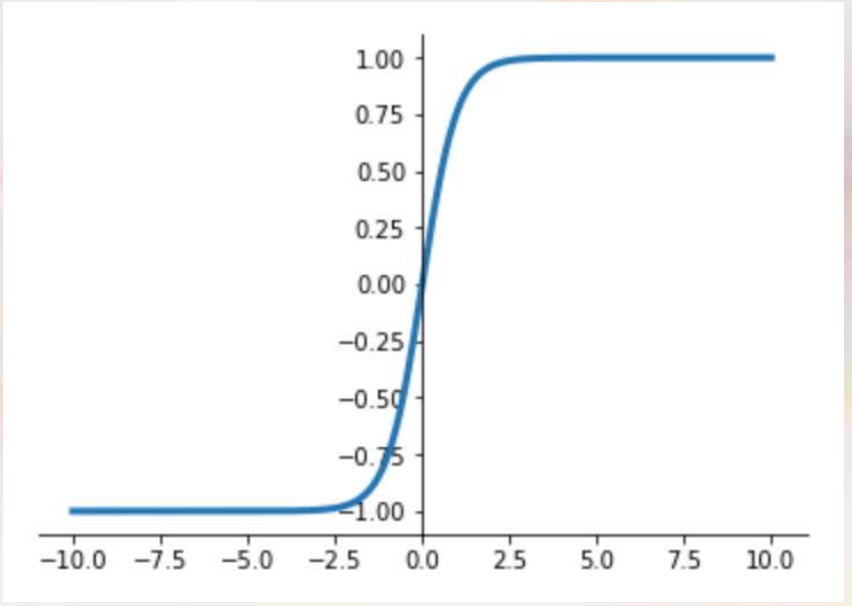
Director & Professor at The Swiss AI Lab IDSIA - USI & SUPSI

1997 by Jürgen Schmidhuber
https://en.wikipedia.org/wiki/Long_short-term_memory

Repetition Activation Functions



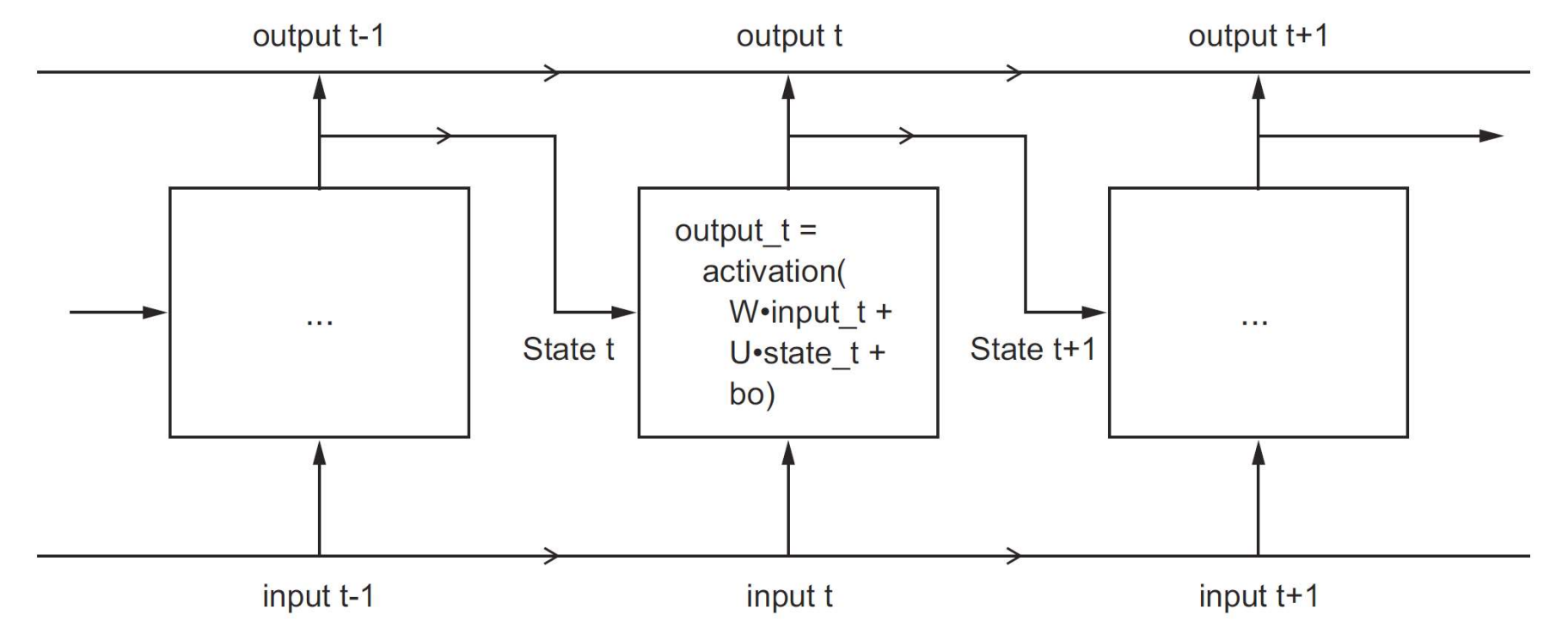
Sigmoid, floating from 0 to 1



Tangens Hyperbolicus, floating from -1 to 1

https://notebooks.azure.com/djcordhose/libraries/buch/html/kap7-iris.ipynb

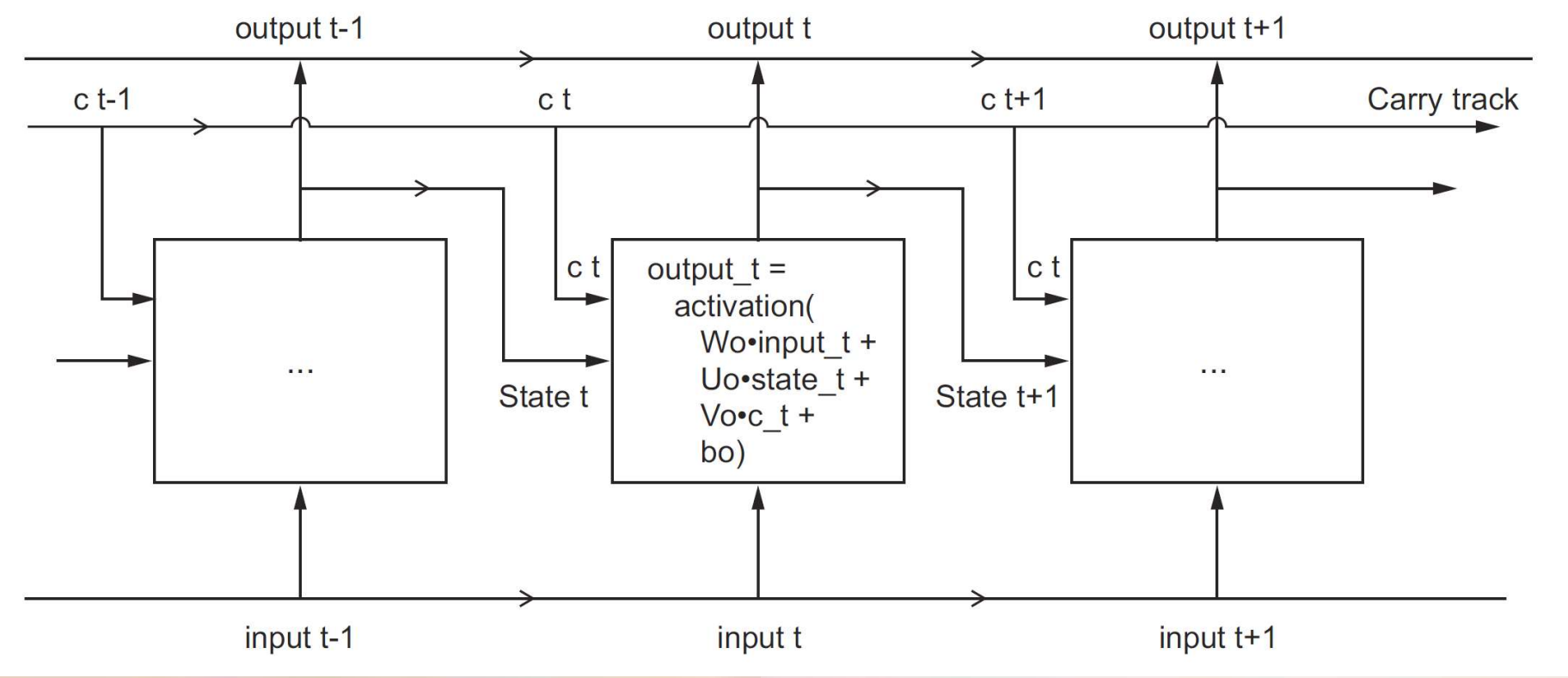
Starting from a Simple RNN



Deep Learning with Python, Chapter 6, François Chollet, Manning

Adding Carry

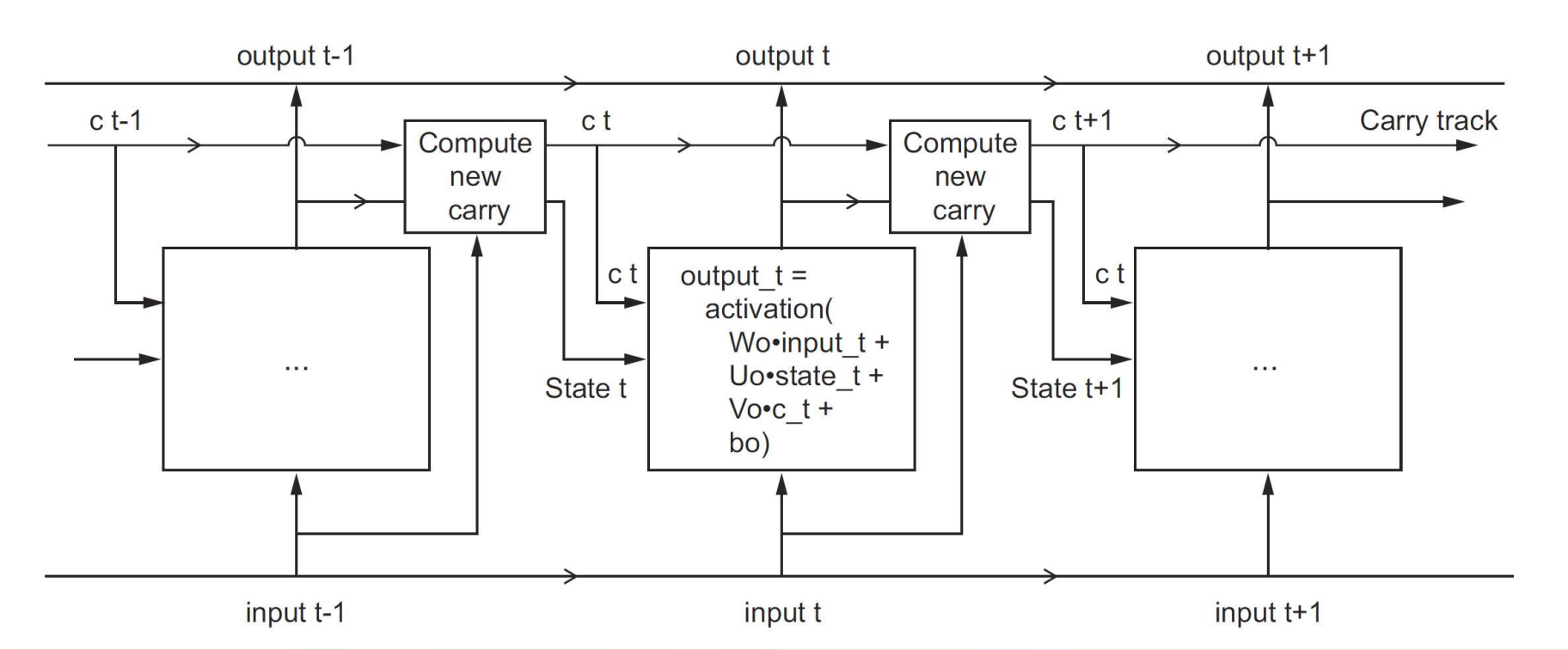
Modulate the next output and the next state via a carry across timesteps



Deep Learning with Python, Chapter 6, François Chollet, Manning

Complete LSTM

New carry is computed from input, output and previous carry



Deep Learning with Python, Chapter 6, François Chollet, Manning

LSTM Intution

gates manage what memory to take over

1. Forget Gate: forget irrelevant information in the carry dataflow (one weights matrix)
2. Keep Gate: provide information about the present, updating the carry track with new information (two weights matrices)

https://arxiv.org/ftp/arxiv/papers/1701/1701.05923.pdf

... but don't overdo it

In summary: you don’t need to understand anything about the specific architecture of an LSTM cell; as a human, it shouldn’t be your job to understand it. Just keep in mind what the LSTM cell is meant to do: allow past information to be reinjected at a later time, thus fighting the vanishing-gradient problem.

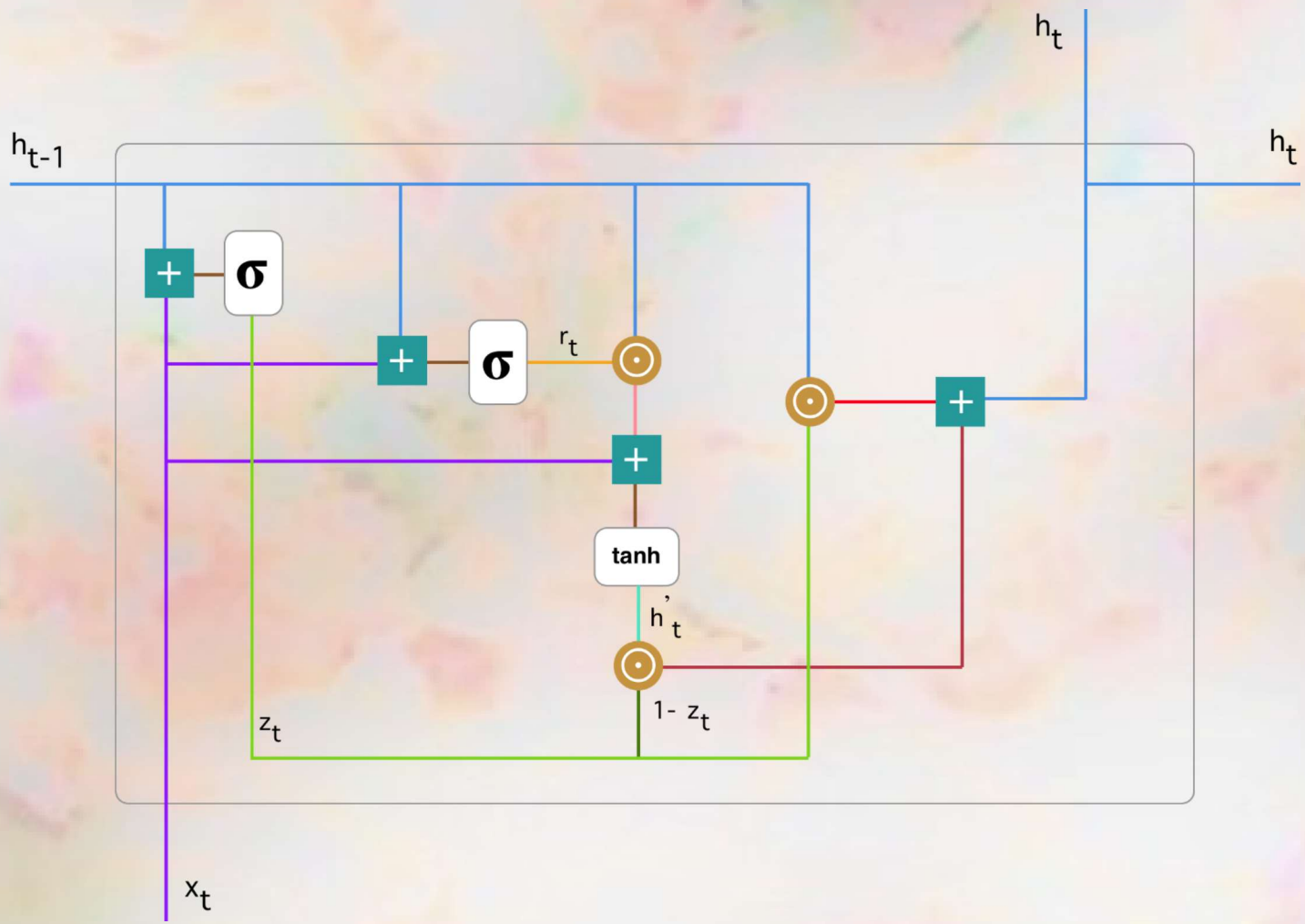
Deep Learning with Python, Chapter 6.2.2, François Chollet, Manning

GRU (Gated Recurrent Unit)

As powerful as LSTMs, but simpler

- Invented in 2014
- variation of LSTMs
- LSTMs have 4 times the complexity of RNNs
- GRUs only 3 times as much

https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be
https://datascience.stackexchange.com/questions/14581/when-to-use-gru-over-lstm
https://arxiv.org/ftp/arxiv/papers/1701/1701.05923.pdf



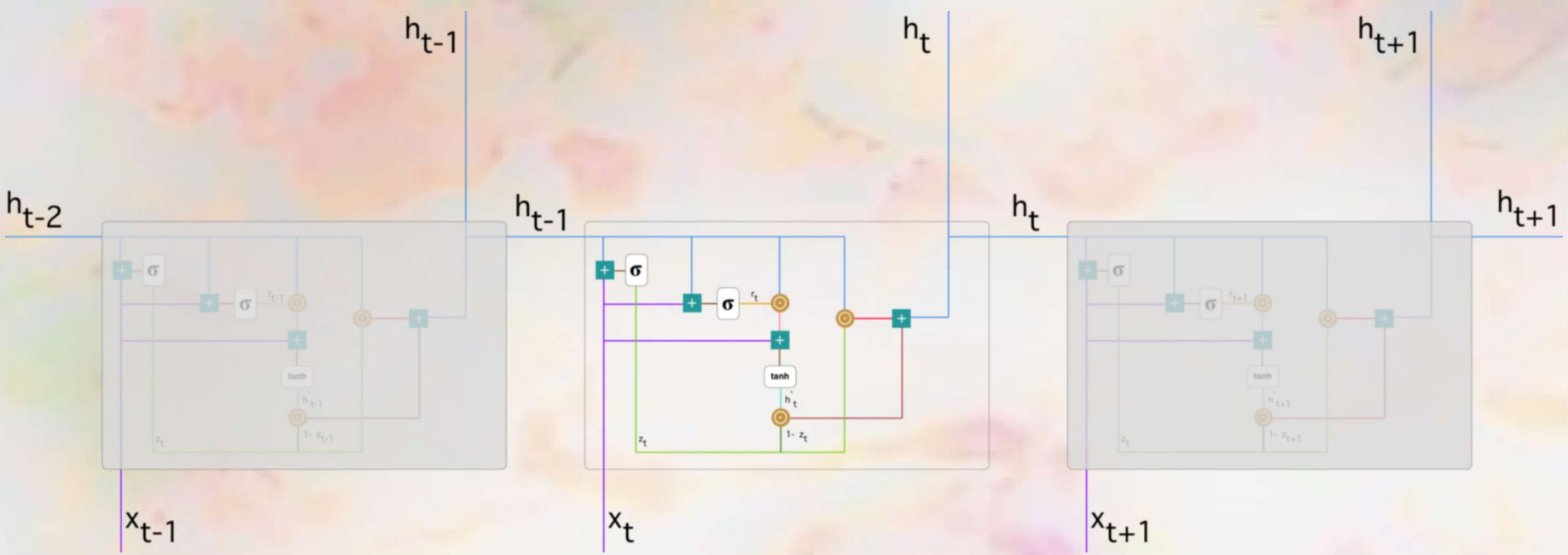
"plus" operation

"sigmoid" function

"Hadamard product" operation

"tanh" function

Feeding input over time into a single unit



Gates

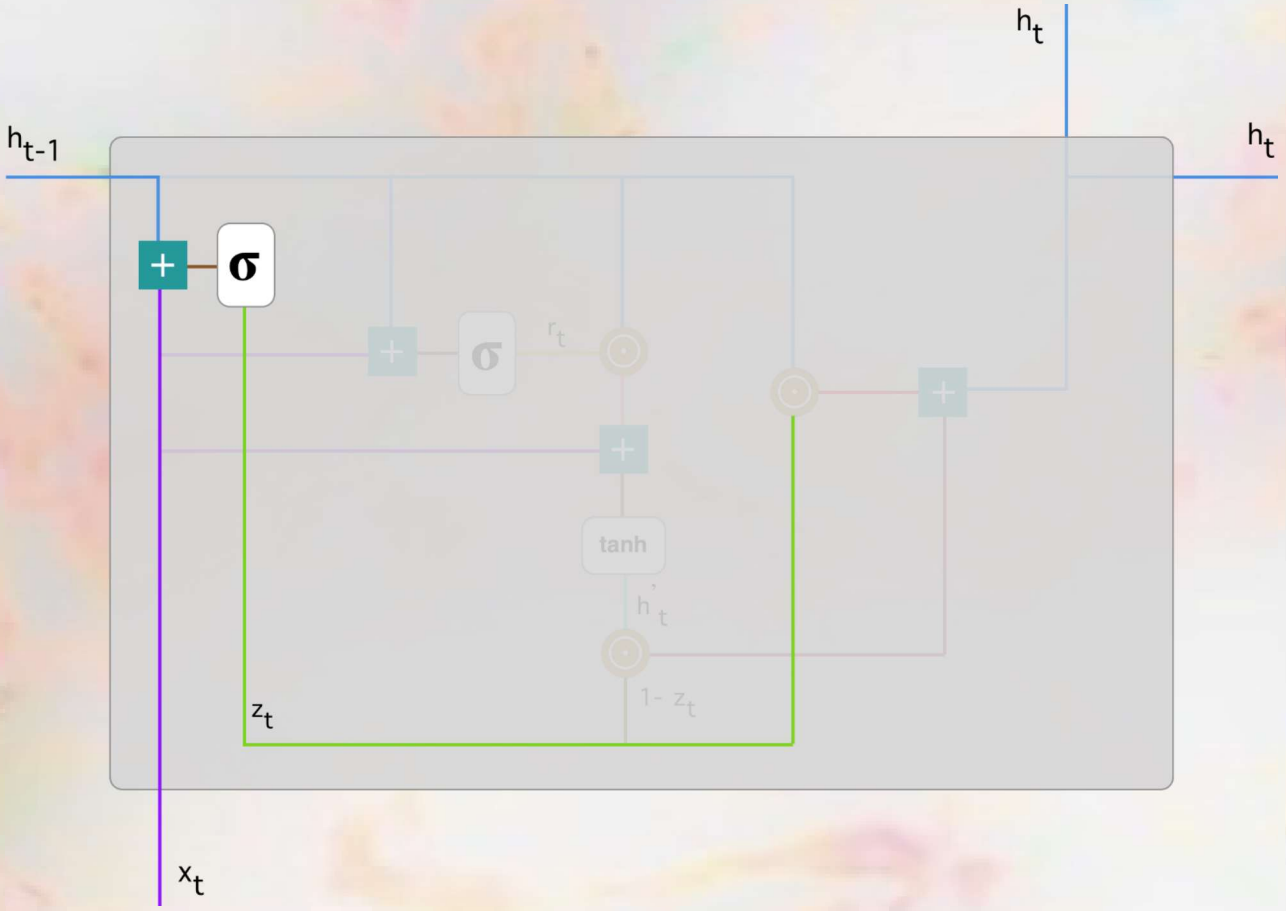
- Decide which information is passed to output
- Can be trained to keep information from long ago
- Can decide to copy all the information from the past eliminating the risk of vanishing gradient problem
- Update Gate
- Reset Gate

https://arxiv.org/pdf/1406.1078v3.pdf

Intution

- ***reset gates*** can be trained to ***wash out previous memories*** if they turn out to be irrelevant
- ***update gates*** can be trained to much more ***rely on past memories*** if they are more relevant to the output

Update Gate



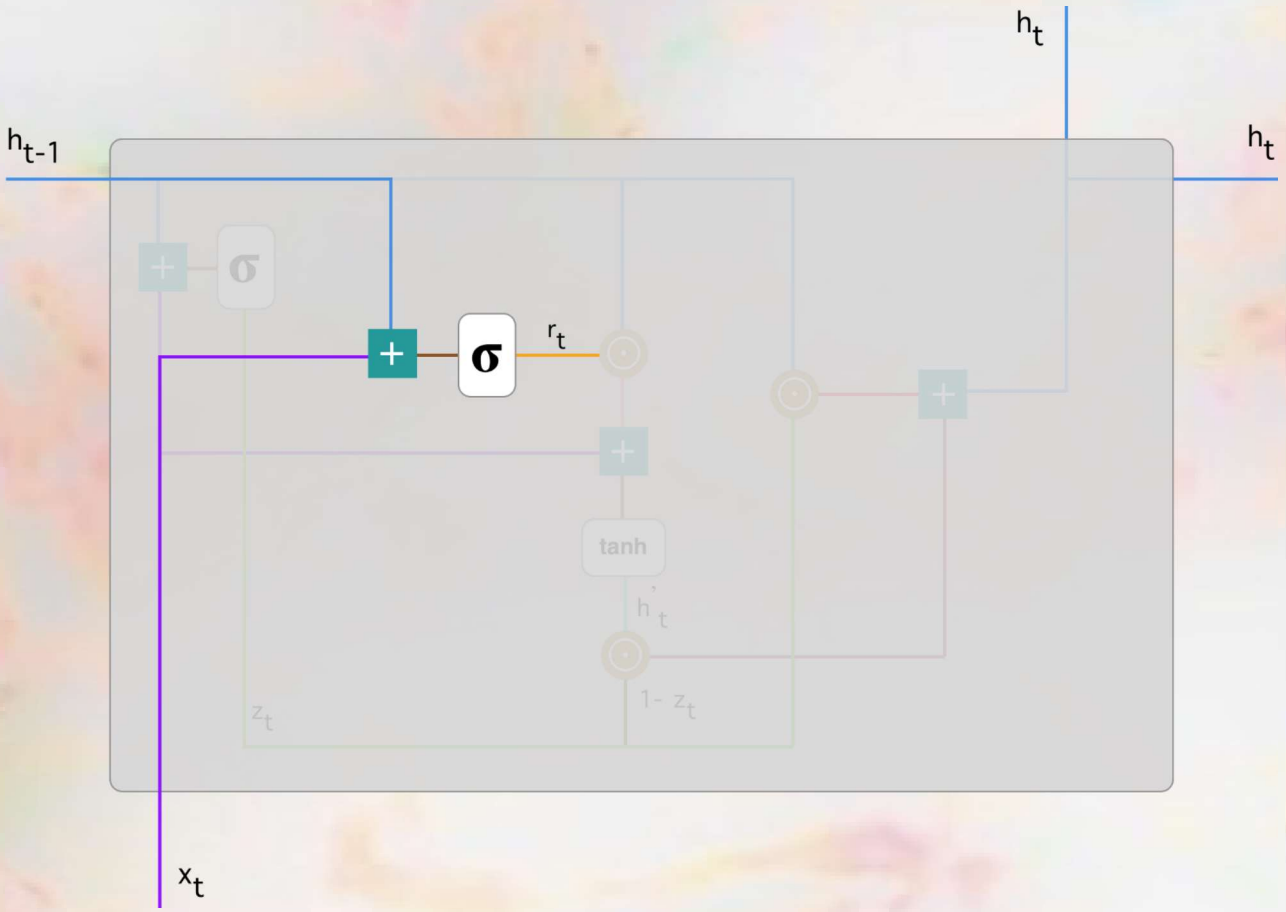
Helps to determine how much of the past information (from previous time steps) needs to be passed along to the future.

Update Gate

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

- When x(t) is plugged into the network unit, it is multiplied by its own weight W(z).
- The same goes for h(t-1) which holds the information for the previous t-1 units and is multiplied by its own weight U(z).
- Both results are added together and a sigmoid activation function is applied to squash the result between 0 and 1.

Reset Gate



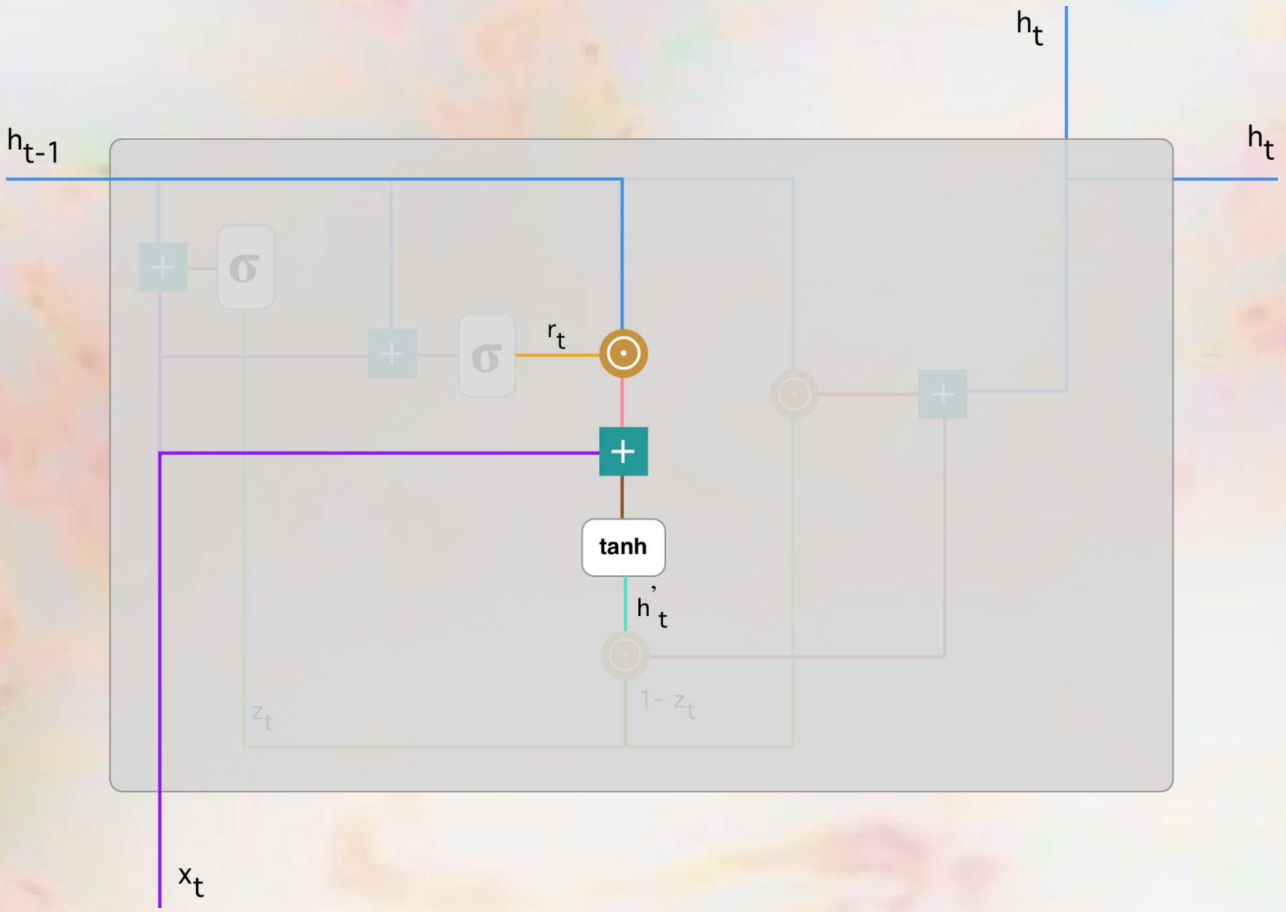
Used to decide how much of the past information to forget

Reset Gate

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

- Same formula as for Update Gate
- But other weights
- Different usage

Current memory content



Uses the reset gate to store relevant information from the past

Hadamard product

- Multiply two matrices of the same dimension
- Multiply entry by entry
- Results in a matrix of the same dimension

Example:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \circ \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} a_{11}\, b_{11} & a_{12}\, b_{12} & a_{13}\, b_{13} \\ a_{21}\, b_{21} & a_{22}\, b_{22} & a_{23}\, b_{23} \\ a_{31}\, b_{31} & a_{32}\, b_{32} & a_{33}\, b_{33} \end{pmatrix}$$

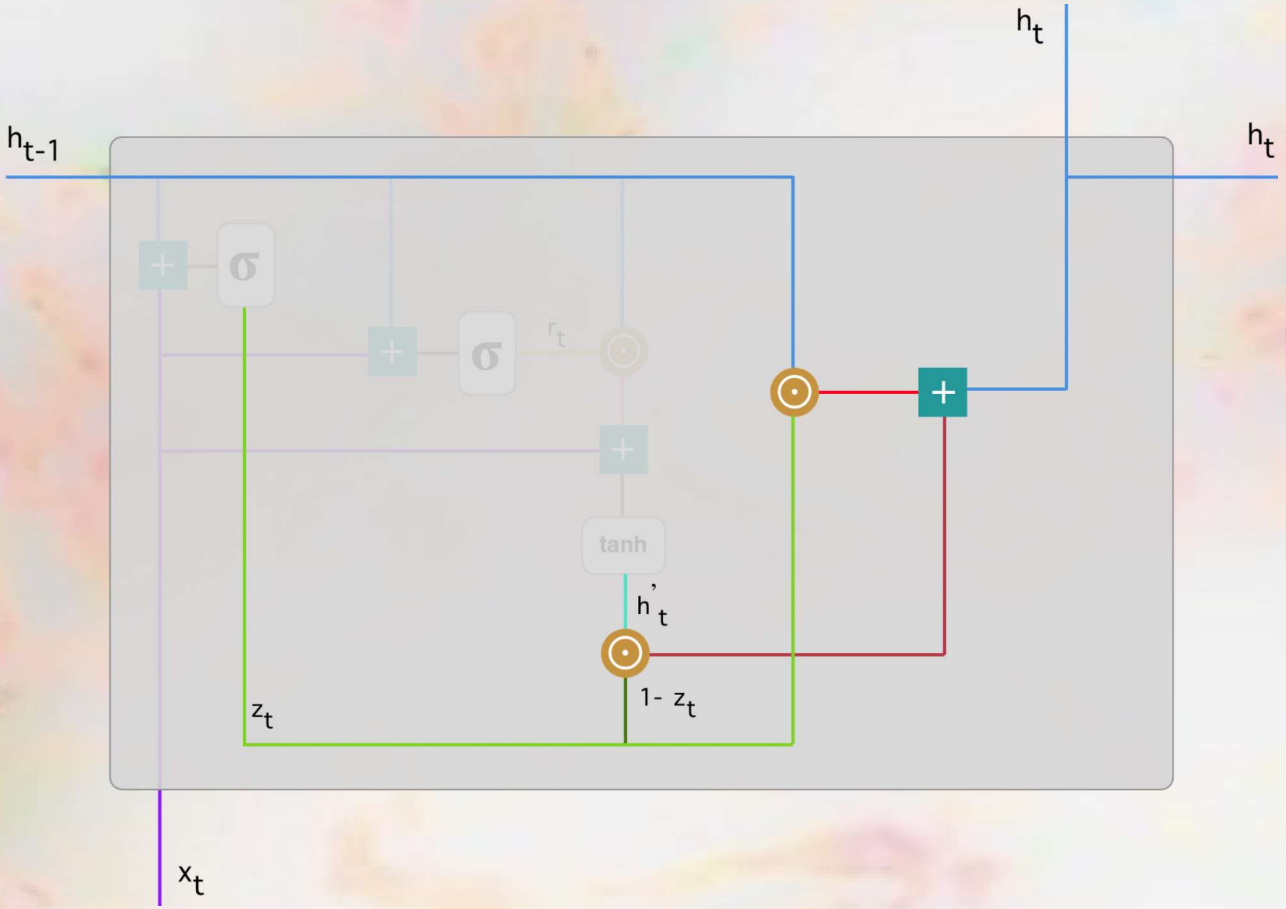
https://en.wikipedia.org/wiki/Hadamard_product_(matrices)

Current memory content

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

- Hadamard (element-wise) product will determine what to remove from the previous time step
- elements of r(t) will be between 0 and 1
- 0 will mean wash out past memories in this unit
- 1 means keep all past memories in this unit
- tanh makes overall output between -1 and 1

Final memory at current time step



Uses update gate to determines what to collect from the current memory content what from the past

Final memory at current time step

Final output is a weighted sum of past and current memory

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

- h(t) is the final output of this unit
- elements of update gate z(t) will be between 0 and 1
- h(t-1) as the input from the previous unit
- 1 - z(t) will be the inverse
- h'(t) is the current memory content

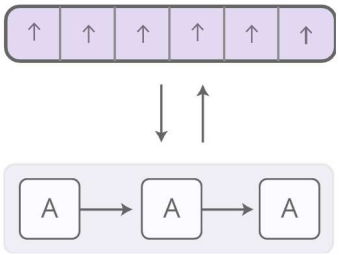
Creating our very own sentiment analysis

Part III: Using embeddings to train recurrent networks

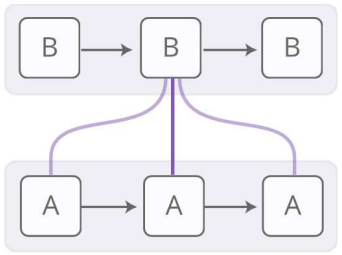
Notebooks:

- https://colab.research.google.com/github/djcordhose/haw/blob/master/notebooks/nlp/2-rnn.ipynb
- https://colab.research.google.com/github/djcordhose/haw/blob/master/notebooks/nlp/2-lstm.ipynb
- https://colab.research.google.com/github/djcordhose/haw/blob/master/notebooks/nlp/3-gru-dropout.ipynb (final version avoiding overfitting)

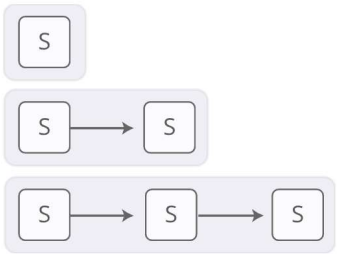
What's next?



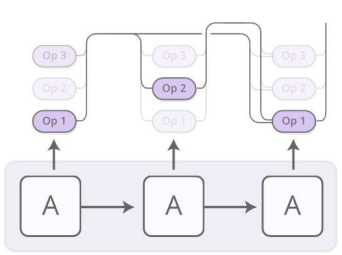
Neural Turing Machines
have external memory that they can read and write to.



Attentional Interfaces
allow RNNs to focus on parts of their input.



Adaptive Computation Time
allows for varying amounts of computation per step.



Neural Programmers
can call functions, building programs as they run.

https://distill.pub/2016/augmented-rnns/
Attention is all you need: https://arxiv.org/pdf/1706.03762.pdf