Deep Learning

4DV661

Convolutional Networks (CNN)
Sequence Modeling: Recurrent Networks (RNN)

Welf Löwe

Course structure

1. Introduction (week 14)
2. Applied Math Basics (week 15)
3. Deep Feedforward Networks (week 16)
4. Regularization for Deep Learning (week 17)
5. Optimization for Training Deep Models (week 18)
6. Convolutional Networks (CNN), Sequence Modeling: Recurrent Networks (RNN) (week 19)
7. Probabilistic DL Models (week 20)
8. Deep Generative Models (week 21)
9. Modelling with LLMs, Attention! (week 22, Wed 27th May 13-15)
10. Exam (week 23, 3rd June 15-17)
11. Deep Reinforcement Learning, teaser (week 24)

Agenda for CNNs

- The Convolution Operation
- Motivation
- Pooling
- Convolution and Pooling as an Infinitely Strong Prior
- Variants of the Basic Convolution Function
- Structured Outputs
- Data Types
- Efficient Convolution Algorithms
- Random or Unsupervised Features
- The Neuroscientific Basis for Convolutional Networks

The Neuroscientific Basis for CNNs

- CNNs are (arguably) the greatest success story of biologically inspired artificial intelligence.
    - Influenced by other fields, some of their key design principles were drawn from neuroscience.
- The path of visual information in the mammalian brain goes from the eye, through the optic nerve and lateral geniculate nucleus, to **V1 (primary visual cortex)**, located at the back of the head
- A convolutional network layer is designed to capture three properties observed in V1

CNN capture three properties observed in V1

Spatial map: V1 is organized in a two-dimensional spatial map that mirrors the structure of the image on the retina.

- CNNs capture this property by having their features defined in terms of two-dimensional maps.

Simple cells: V1 contains many simple cells. A simple cell's activity can be characterized to some extent by a linear function of the image within a small, spatially localized receptive field.

- The detector units in a CNN are designed to emulate these properties of simple cells. Simple cells in the first layer of an artificial neural network respond as a linear function of pixel values.

The weights of most V1 cells are described by Gabor functions.

- Kernal functions in the first layer(s) of CNNs learn features like Gabor functions when applied to natural images.

Complex cells: V1 also contains many complex cells. These cells respond to features like those detected by simple cells, but complex cells are invariant to small shifts in the position of the feature.

- This property inspires the pooling units found in convolutional networks.

Complex cells also show invariance to some changes in lighting

- This has inspired certain cross-channel pooling strategies like maxpooling

Influence on the fundamental operations

Convolution can be thought of as implementing an infinitely strong prior that the function learned by a layer contains only local interactions and is equivariant to translation.

Pooling is seen as an infinitely strong prior that each unit should be invariant to small translations

(Not limited to CNNs)

Rectified Linear Units (ReLU) are partly motivated by biological considerations, intending to capture properties of biological neurons such as inactivity for some inputs and proportionality for others, and encouraging sparse activations.

Pointers for CNN

- Slides of the book discussion on YouTube (partially shown today): https://docs.google.com/presentation/d/1yzNb7e32o1zgwHQ5AIPpwFvpocc28_L70pDXrCeoUyg/edit#slide=id.g23e418f4b4_0_485
- CNN with Tensorflow: https://www.tensorflow.org/tutorials/images/cnn
- More on ResNet: https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035
- Demo of CNN on MNIST classification in the notebook

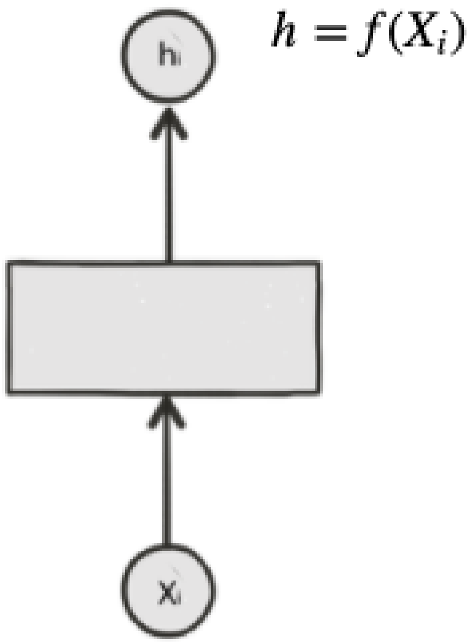
Agenda for RNNs

- Unfolding Computational Graphs
- Recurrent Neural Networks (RNNs)
- Bidirectional RNNs
- Encoder-Decoder Sequence-to-Sequence Architecture
- Deep Recurrent Networks
- Recursive Neural Networks
- The Challenge of Long-Term Dependencies
- Echo State Networks
- Leaky Units and Other Strategies for Multiple Time Scales
- The Long Short-Term Memory and Other Gated RNNs
- Optimization for Long-Term Dependencies
- Explicit Memory

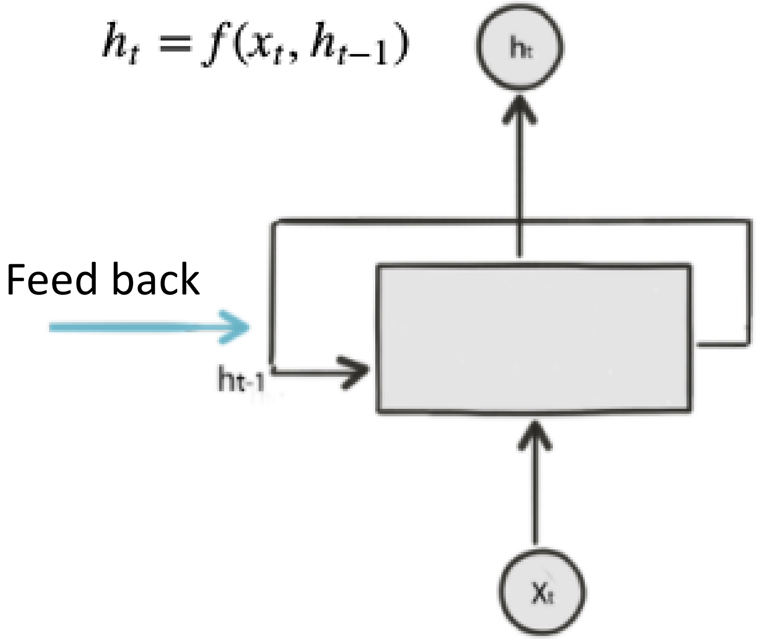
Dense vs Simple RNN

https://jhui.github.io/2017/03/15/RNN-LSTM-GRU/

Dense Neural Network

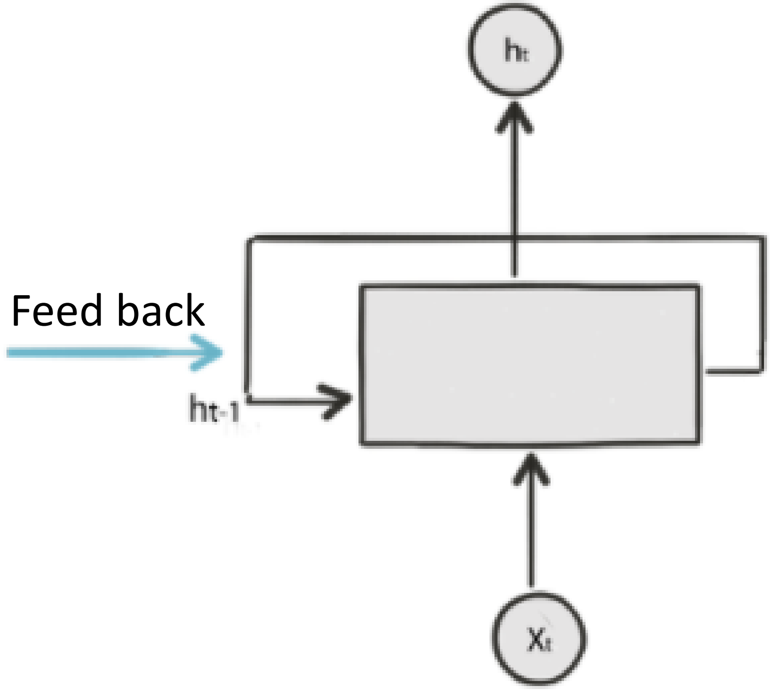


Recurrent Neural Network


Feed back

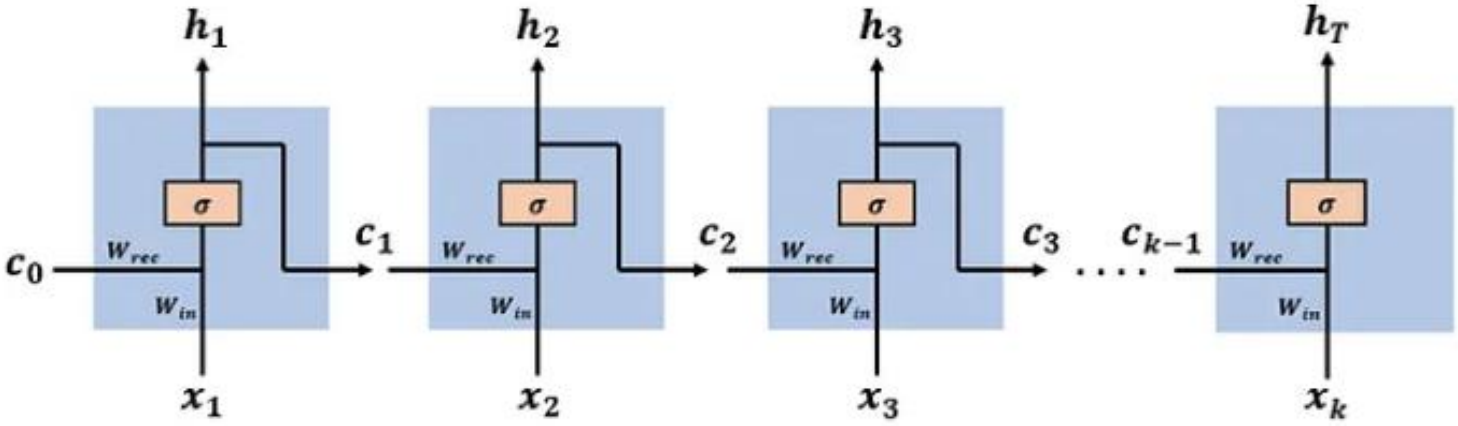
Unfolded Perspective

RNN



Unfolded RNN

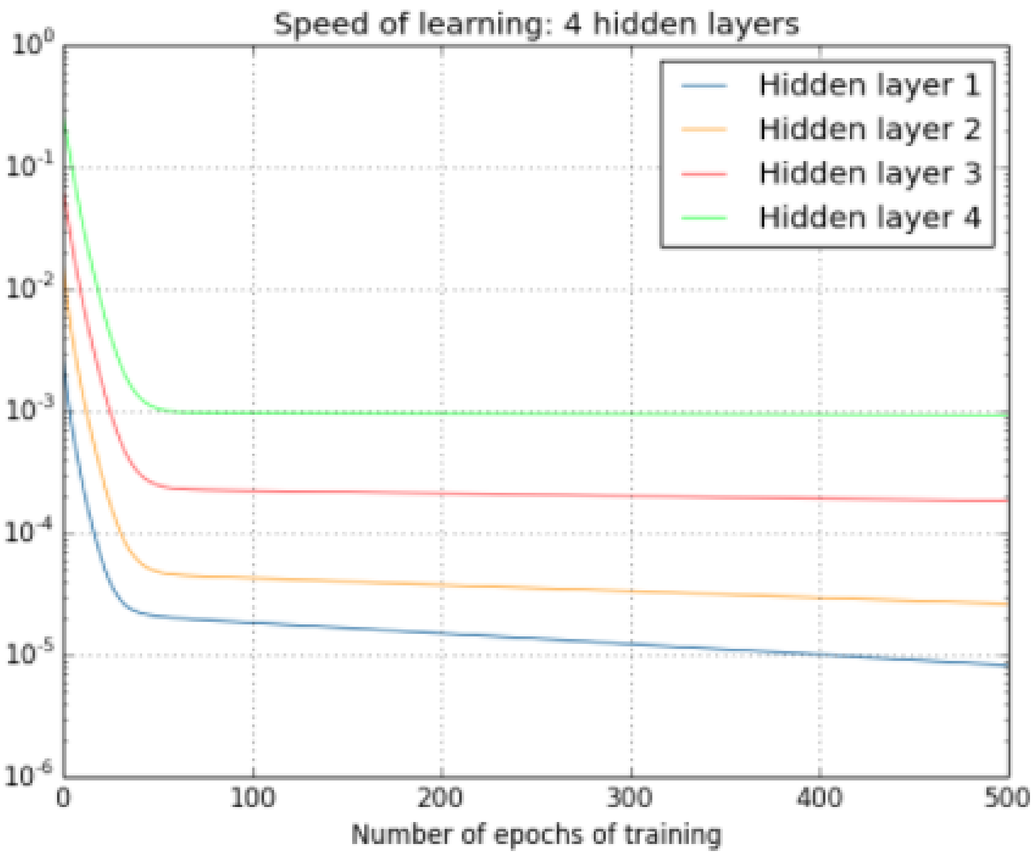
$h_t = \sigma(Wx_t + Uh_{t-1} + b)$ hidden layer



RNNs and vanishing gradients

http://neuralnetworksanddeeplearning.com/chap5.html

- Due to unfolding, many implicit layers are introduced in RNNs
  - depth corresponds to the sequence length
- We observe differences in the layers' learning rate: later layers learn faster than earlier layers
- Gradient as a vector
  - entries determine how quickly weights change, i.e., a layer learns
  - e.g., the L2 norm of gradients is a measure of learning speed

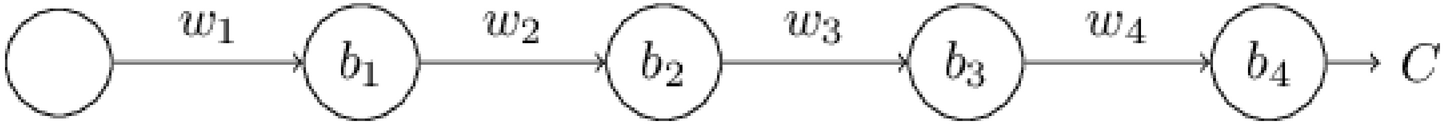


Small gradients a problem?

- Wouldn't it be good news if the derivative $f'(x)$ was small?
  - Wouldn't that mean we were already near an optimum?
  - The small gradient in early layers could mean that no much adjustment of the weights and biases needed?
- Unfortunately, it is a problem
  - Recall the random initialization of the weights and biases
  - It is extremely unlikely that the initial weights and biases were correct
  - Small gradients just mean slow learning

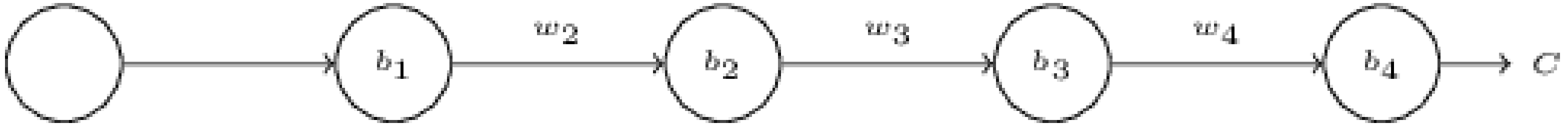
Cause of the vanishing gradients

- Long multiplicative chains for computing the gradients
- Simplified NN model with $a_j = \sigma(z_j)$ and $z_j = w_j a_{j\text{-}1} + b_j$ and sigmoid activation σ input to a cost C



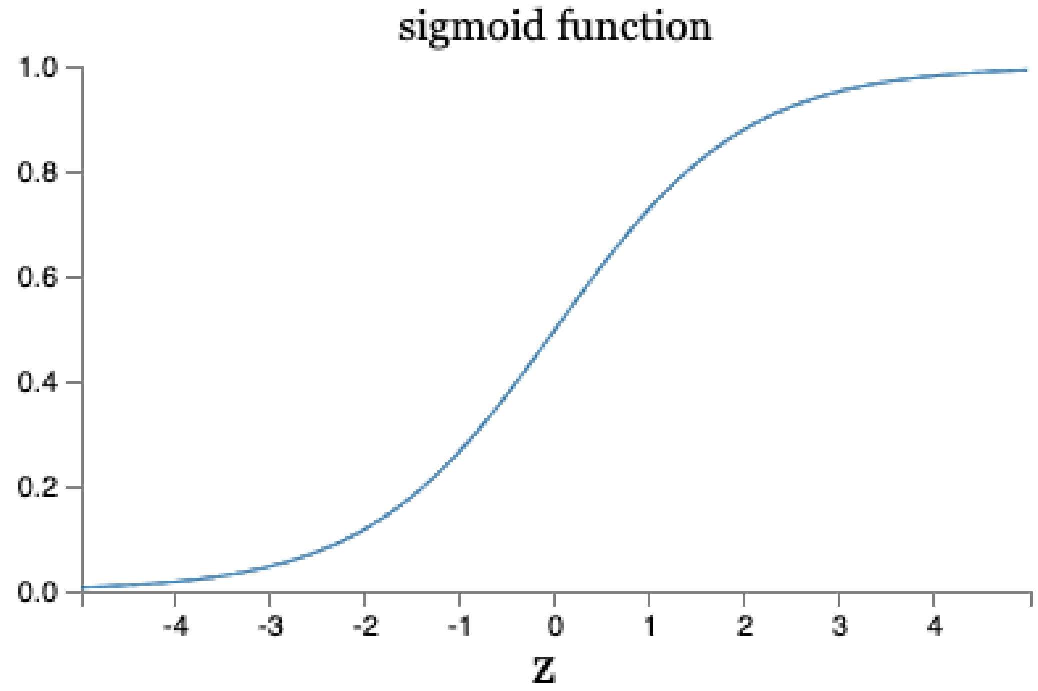
- Take, e.g., the gradient of the cost C wrt. the first bias b_1

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times \boxed{w_2 \times \sigma'(z_2)} \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$

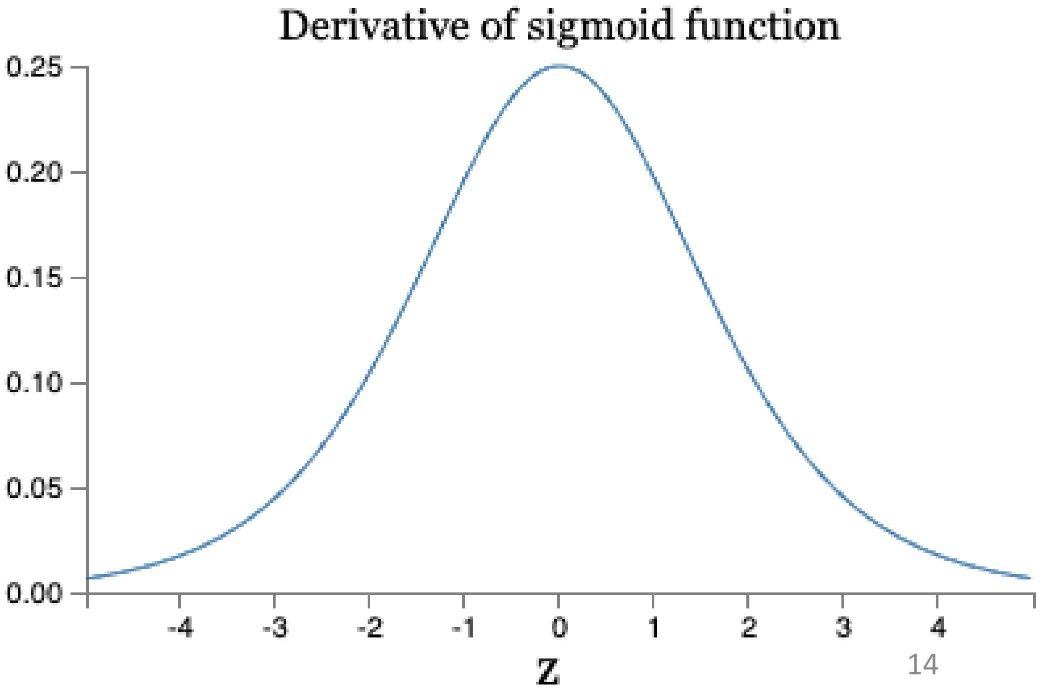


Cause of the vanishing gradients (cont’d)

- The Glorot uniform weight initializer (default in TensorFlow) draws samples from a uniform distribution in [-*limit*, *limit*], *limit*=$\sqrt{6/(\mathrm{in}+\mathrm{out})}$
- $|w_j| < 4 \Longrightarrow w_j \times \sigma'(z_j) < 1$ is likely to happen
- Might lead to vanishing gradients



sigmoid function



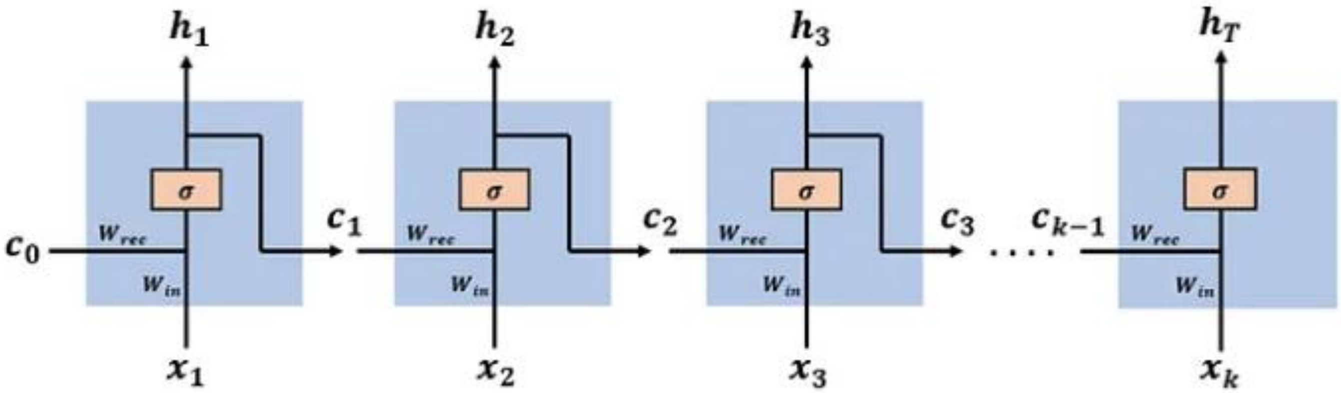
Derivative of sigmoid function

The exploding gradient problem

- Less likely than vanishing gradients but still possible
- Assume all the weights in the network to be large,
  - For instance, $w_1 = w_2 = w_3 = w_4 = 100$
- Assume the biases so that the weighted input z_j to each activation is close to 0 (and, hence, $\sigma'(z_j) \approx 1/4$).
  - For instance, to achieve $z_1 = w_\mathrm{j} a_0 + b_1 = 0$, set $b_1 = -100 \times a_0$
  - Select the other biases similarly
- Then $w_j \times \sigma'(z_j) = 100/4 = 25$
- Might lead to exploding gradients

In the Context of RNNs

https://medium.datadriveninvestor.com/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577



$$c_t = \quad h_t = \sigma(Wx_t + Uh_{t-1} + b)$$ hidden

$$E = \sum_{t=1}^{T} E_t$$ total loss

$$\frac{\partial E}{\partial \theta} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial \theta}$$ total loss gradient for $\theta \in \{W, U, b\}$

$$\frac{\partial E_k}{\partial W} = \frac{\partial E_k}{\partial h_k} \frac{\partial h_k}{\partial c_k} \cdots \frac{\partial c_2}{\partial c_1} \frac{\partial c_1}{\partial W}$$ contained terms

$$= \frac{\partial E_k}{\partial h_k} \frac{\partial h_k}{\partial c_k} \left(\prod_{t=2}^{k} \frac{\partial c_t}{\partial c_{t-1}} \right) \frac{\partial c_1}{\partial W}$$

Product of Jacobians

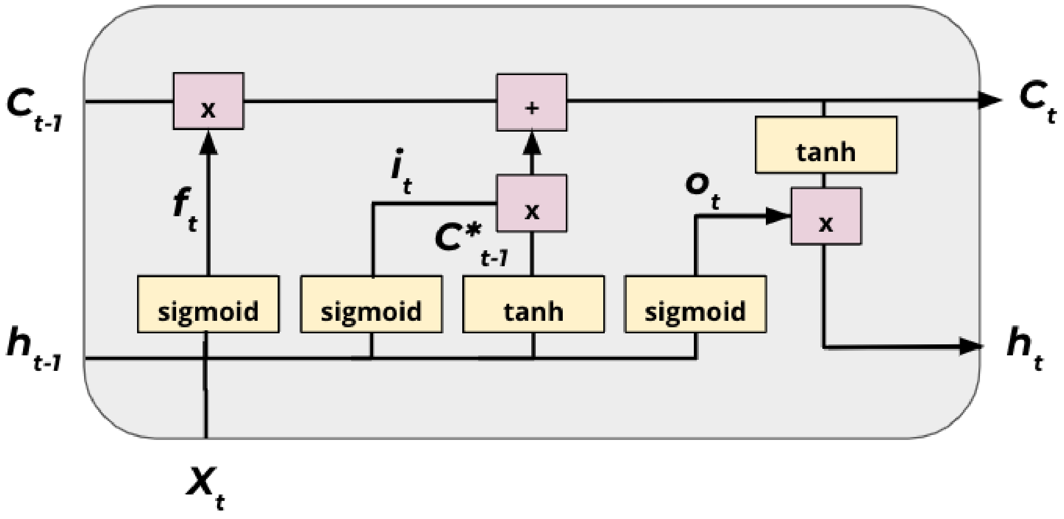
Vanishing and Exploding Gradients

- The gradient is calculated by **recursively accumulating** gradients backwards through time (BPTT).
- Jacobian terms depends on the activation function derivatives and the weight matrix W
- If they are:
  - **less than 1**, they **shrink** → **vanishing gradients**.
  - **greater than 1**, they **grow exponentially** → **exploding gradients**.

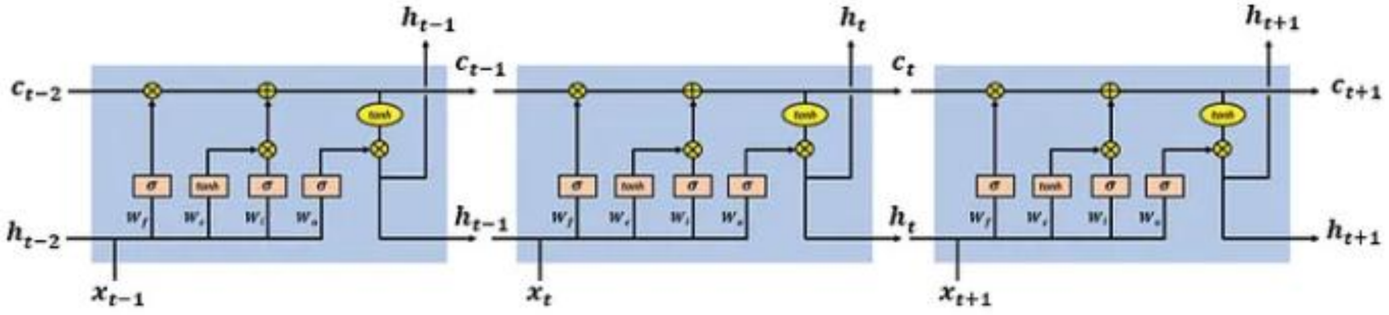
What to do then?

- General techniques for NN avoiding exploding gradients
    - Conjugated gradients methods (only gradient direction is used)
    - Gradient clipping (threshold avoids exploding gradients)
        - pre-determined gradient threshold be introduced,
        - gradients norms that exceed this threshold are scaled down
- For RNN
    - Changing the network so that the gradient does not contain the long multiplicative chains
        - Long Short-Term Memory network (LSTM)
        - Gated Recurrent Unit networks (GRU)
    - How does this work?
        - Gates control data flow through the network to ensure only relevant information stays.
        - Gates control also the backflow of gradients through the network to ensure that they don’t vanish if there was a corresponding forward flow (otherwise OK).

Long Sort Term Memory (LSTM)



$$c_t = c_{t-1} \otimes f_t \oplus c_t^* \otimes i_t$$



…

	Gates
$f_t = \sigma(W_f \cdot [h_{t-1}, X_t] + b_f)$	forget
$i_t = \sigma(W_i \cdot [h_{t-1}, X_t] + b_i)$	input
$C_t^* = \tanh(W_c \cdot [h_{t-1}, X_t] + b_c)$	candidate
$C_t = f_t \times C_{t-1} + i_t \times C_t^*$	cell state
$o_t = \sigma(W_o \cdot [h_{t-1}, X_t] + b_o)$	output
$h_t = o_t \times \tanh(C_t)$	

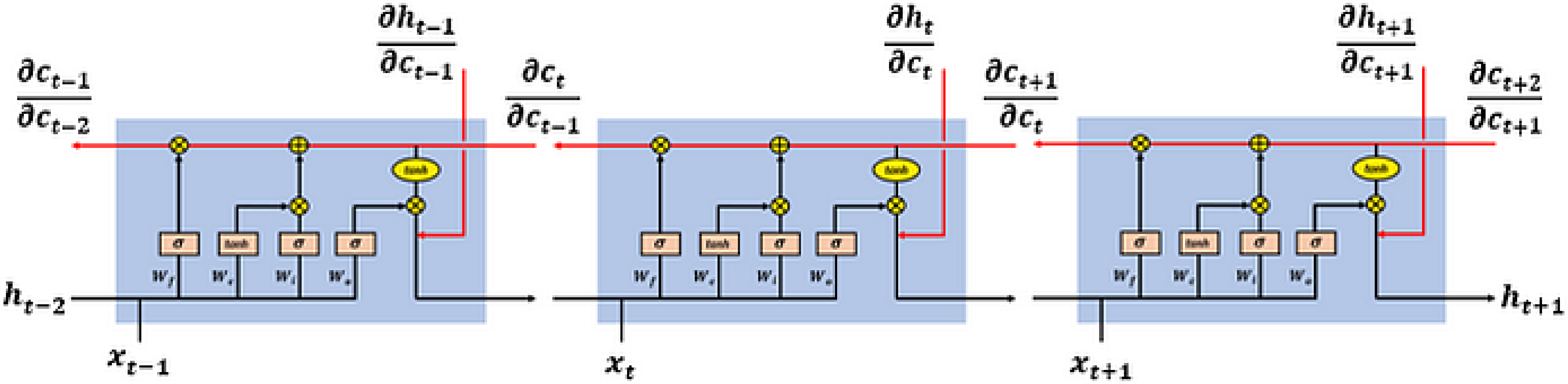
Backpropagation over cell states

https://medium.datadriveninvestor.com/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577

$$\frac{\partial E_k}{\partial W} = \frac{\partial E_k}{\partial h_k}\frac{\partial h_k}{\partial c_k}\dots\frac{\partial c_2}{\partial c_1}\frac{\partial c_1}{\partial W}$$

$$= \frac{\partial E_k}{\partial h_k}\frac{\partial h_k}{\partial c_k}\left(\prod_{t=2}^{k}\frac{\partial c_t}{\partial c_{t-1}}\right)\frac{\partial c_1}{\partial W}$$

gradient again contains terms with products of Jacobians (like vanilla RNNs)



$\frac{\partial h_{t-1}}{\partial c_{t-1}}$
$\frac{\partial h_t}{\partial c_t}$
$\frac{\partial h_{t+1}}{\partial c_{t+1}}$
$\frac{\partial c_{t-1}}{\partial c_{t-2}}$
$\frac{\partial c_t}{\partial c_{t-1}}$
$\frac{\partial c_{t+1}}{\partial c_t}$
$\frac{\partial c_{t+2}}{\partial c_{t+1}}$
tanh
σ
W_f
W_c
W_i
W_o
h_{t-2}
h_{t+1}
x_{t-1}
x_t
x_{t+1}

Avoiding the gradient to vanish, i.e., $\prod_{t=2}^{k} \frac{\partial c_t}{\partial c_{t-1}} \to 0$

$$\frac{\partial c_t}{\partial c_{t-1}} = \sigma'(W_f \cdot [h_{t-1}, x_t]) \cdot W_f \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot c_{t-1}$$

$$+ f_t$$

$$+ \sigma'(W_i \cdot [h_{t-1}, x_t]) \cdot W_i \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot c_t^*$$

$$+ \sigma'(W_c \cdot [h_{t-1}, x_t]) \cdot W_c \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot i_t$$

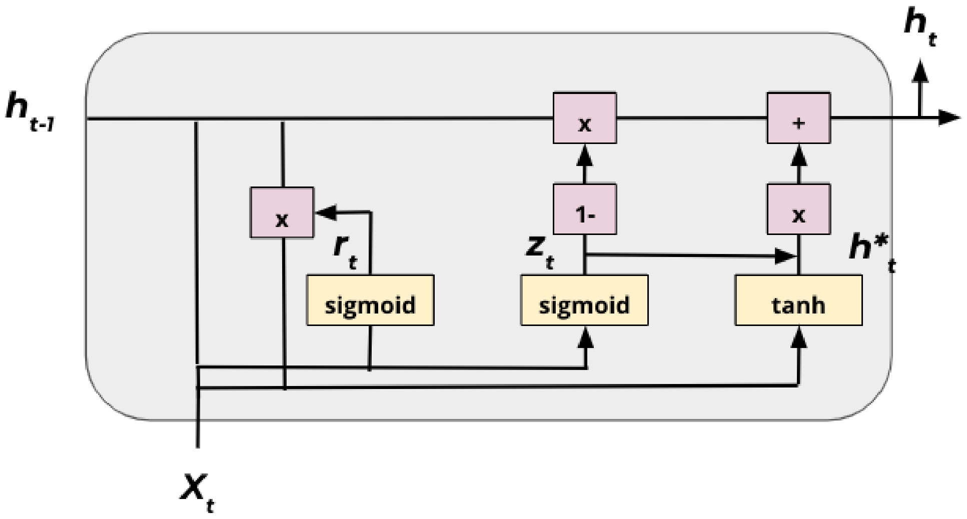
In forward propagation the gates control the information flow

- They prevent any irrelevant info to be written in the cell state
- E.g., $f_i = 0.5, i \in \{k, \dots, t\}$ there is no contribution of c_k to c_t as it was deluded by a factor of $(0.5)^{t\text{-}k}$

In backwards propagation they control the flow of the gradients

- If c_k does not forward contribute to c_t (e.g. $f_i \to 0$) then gradient flow into c_k might vanish, but it is OK: no need to correct c_k for the error in c_t
- Otherwise, the gradients do not vanish completely because there always exists at least one backward path from c_t to c_k to where they don't vanish

Gated Recurrent Unit (GRU)



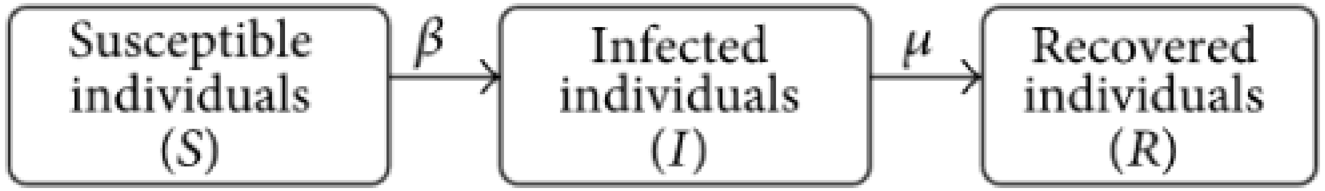
	Gates
$z_t = \sigma(W_z \cdot [h_{t-1}, X_t] + b_z)$	update
$r_t = \sigma(W_r \cdot [h_{t-1}, X_t] + b_r)$	reset
$h^*_t = \tanh(W \cdot [r_t \odot h_{t-1}, X_t] + b)$	candidate
$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot h^*_t$	output

LSTM vs GRU

- LSTM and GRU follow similar ideas to avoid vanishing/exploding gradients
- GRUs have advantages over LSTMs
  - Fewer gates, less complex structure, hence, lower computational effort.
  - Fewer parameters, hence, easier to train.
- GRUs performance can be worse than LSTMs as they struggle to retain long-term dependencies.

Demo RNN Notebook

- Recall the notebook on “Probability and Information Theory”
- We introduced a simplified model for simulating and understanding the spreading of diseases.



- A fraction of β of the susceptible individuals S gets infected each unit of time, say each day. A fraction μ of the infected recover. A simple generalization is a third parameter ν and the assumption that only $\nu\mu$ individuals recover, while $1-\nu\mu$ individuals die.
- Now we try to predict the spreading process as a time series using RNN

Assignment 6 – Reading

- Basics (recap)
  - DL: Chapter 3 (you did this already)
  - alternative Pattern Recognition and Machine Learning: chapters 1+2, pp. 1-90
- Sampling
  - DL: Chapter 17, pp. 581-596
  - alternative Pattern Recognition and Machine Learning: chapter 11, pp. 526-545
- Graphical Models
  - DL: Kap. 16, pp. 549-575
  - alternative Pattern Recognition and Machine Learning: chapter 8, pp. 359-405

“Pattern Recognition and Machine Learning” is available as a free eBook:
https://www.microsoft.com/en-us/research/wp-content/uploads/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf

Assignment 6 – Programming

- Option 1: Modify (extend) the Jupyter notebook “Convolutional Neural Networks for digital image classification”
  - Reduce the number of parameters and/or increase accuracy by a selection of
    - augmentation – but mind the translation invariance
    - modifying the model and learning parameters
    - adding CNN/Pooling layer
    - removing or shrinking the dense layer …
- Option 2: Modify (extend) the Jupyter notebook “Time series forecasting”
  - Reduce the number of parameters and/or increase accuracy by a selection of
    - Substitution of the LSTM layers with GRU layers
    - adding (dense) layers
    - Modifying the history length …
  - Show diagrams with predictions around the peak of infection
- Report the results in a notebook. Discuss your results in the style of
  - Let’s try … because we might assume that …
  - The effect on accuracy was …; on learning convergence was …; on model complexity was …
- Deadline: 2025-05-13