Neuronale Netze
Recurrent Neural Networks (RNNs)

Prof. Dr.-Ing. Sebastian Stober

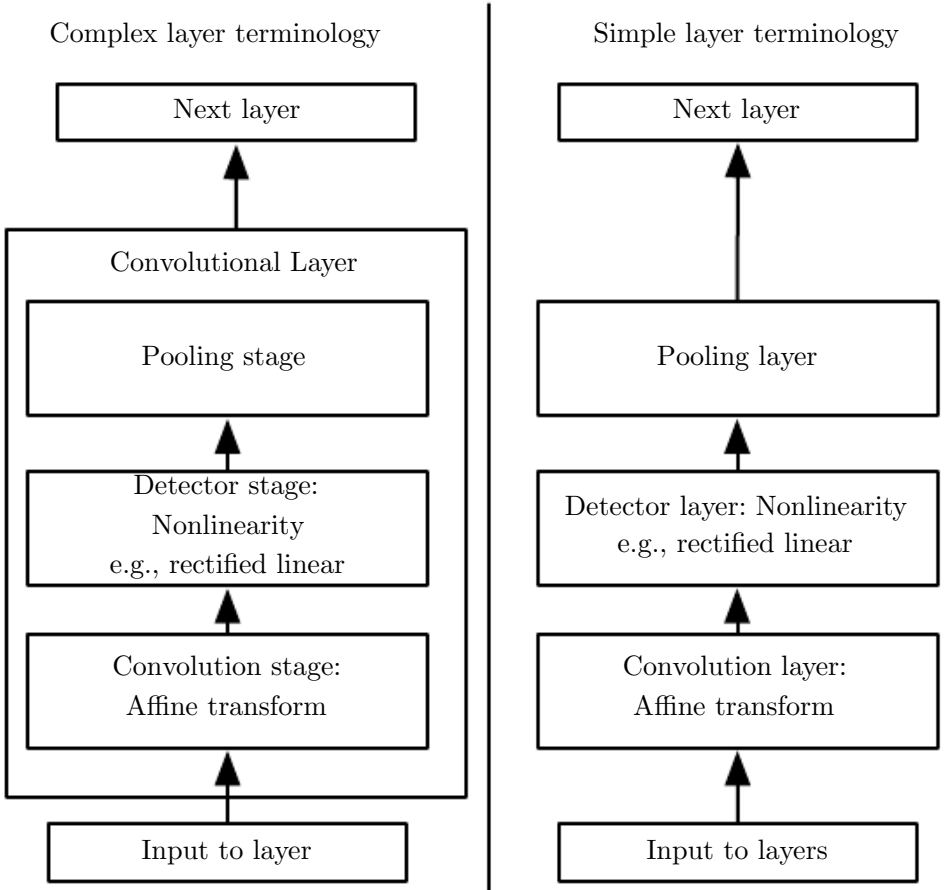
Artificial Intelligence Lab
Institut für Intelligente Kooperierende Systeme
Fakultät für Informatik
stober@ovgu.de

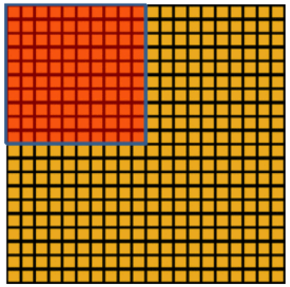
FACULTY OF
COMPUTER SCIENCE

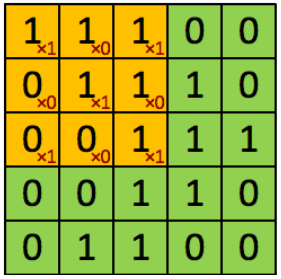
AiLAB

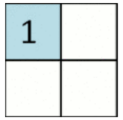
Recap

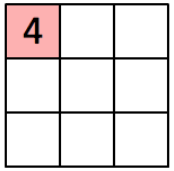
Recap: Convolutional Layers







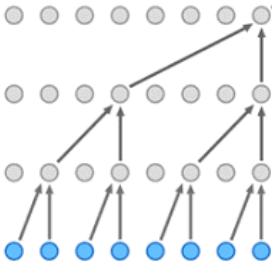




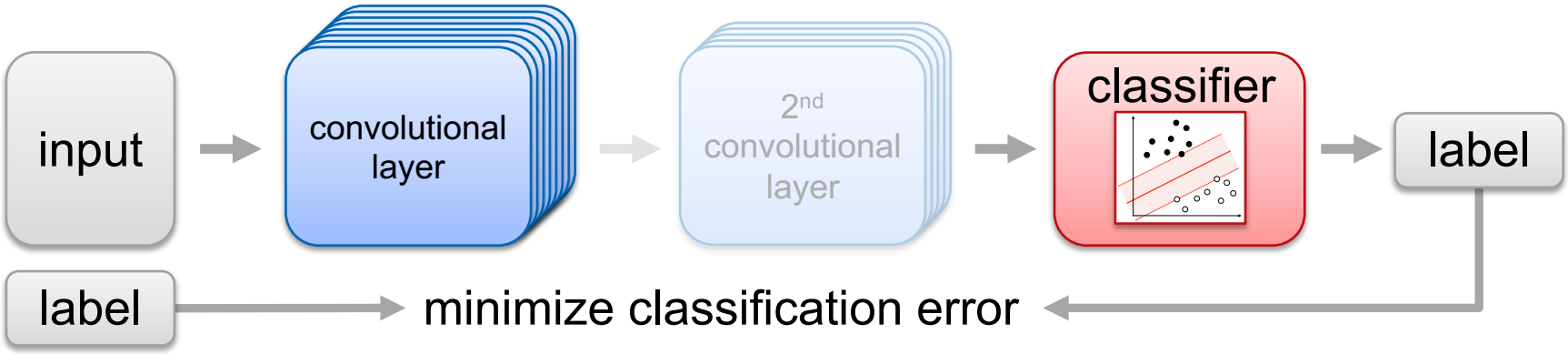
design choices
(hyper params)

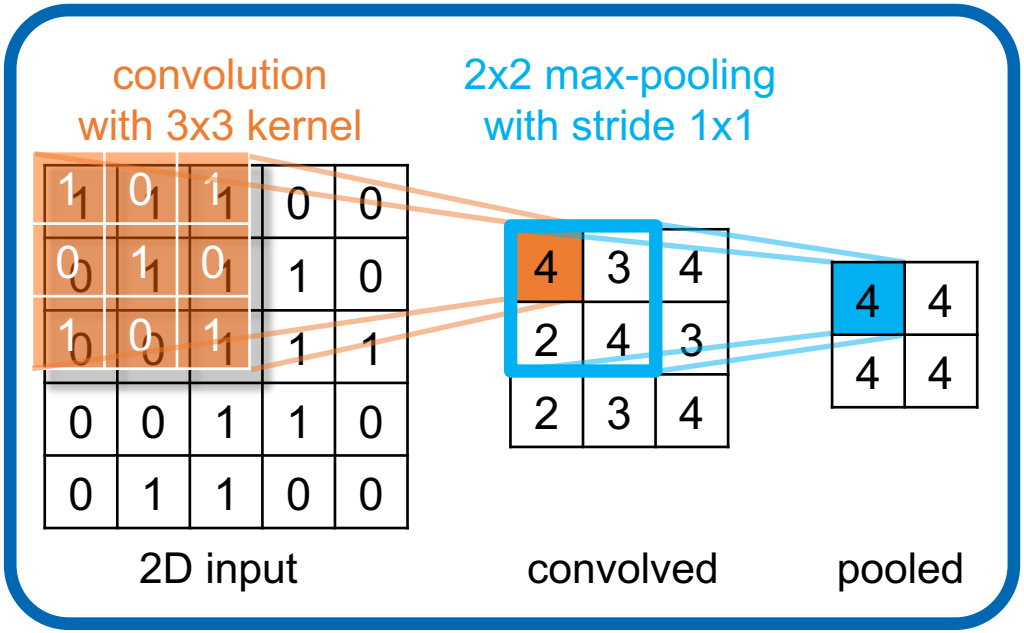
- pool shape
- pool op
- pool stride
- non-linearity
- #kernels
- kernel shape
- kernel stride
- biases?
- padding mode

variant:
dilated
convolution



Convolutional Neural Nets (CNNs)





- **local connectivity**
- **parameter sharing**
- translation **equivariance**
- involves non-linear transform (activation function) after conv.
- pool size controls amount of invariance to input translations
- stride (step size) controls non-linear sub-sampling

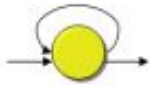
RNNs

(with figures from Deep Learning book)

Formale Definition

Ein **rekurrentes neuronales Netz** ist ein neuronales Netz
mit einem Graph $G = (U, C)$, das Kanten mit einer Rückkopplung enthält:

direkte Rückkopplung: $\exists u \in U \mid u \times u \in C$



in r-schichteten Netzen:

$U_{\text{hidden}} = U^{(1)}_{\text{hidden}} \cup \cdots \cup U^{(r-2)}_{\text{hidden}},$

$\forall 1 \leq i < j \leq r-2: \quad U^{(i)}_{\text{hidden}} \cap U^{(j)}_{\text{hidden}} = \emptyset,$

$C \supseteq \left(U_{\text{in}} \times U^{(1)}_{\text{hidden}}\right) \cup \left(\bigcup_{i=1}^{r-3} U^{(i)}_{\text{hidden}} \times U^{(i+1)}_{\text{hidden}}\right) \cup \left(U^{(r-2)}_{\text{hidden}} \times U_{\text{out}}\right)$

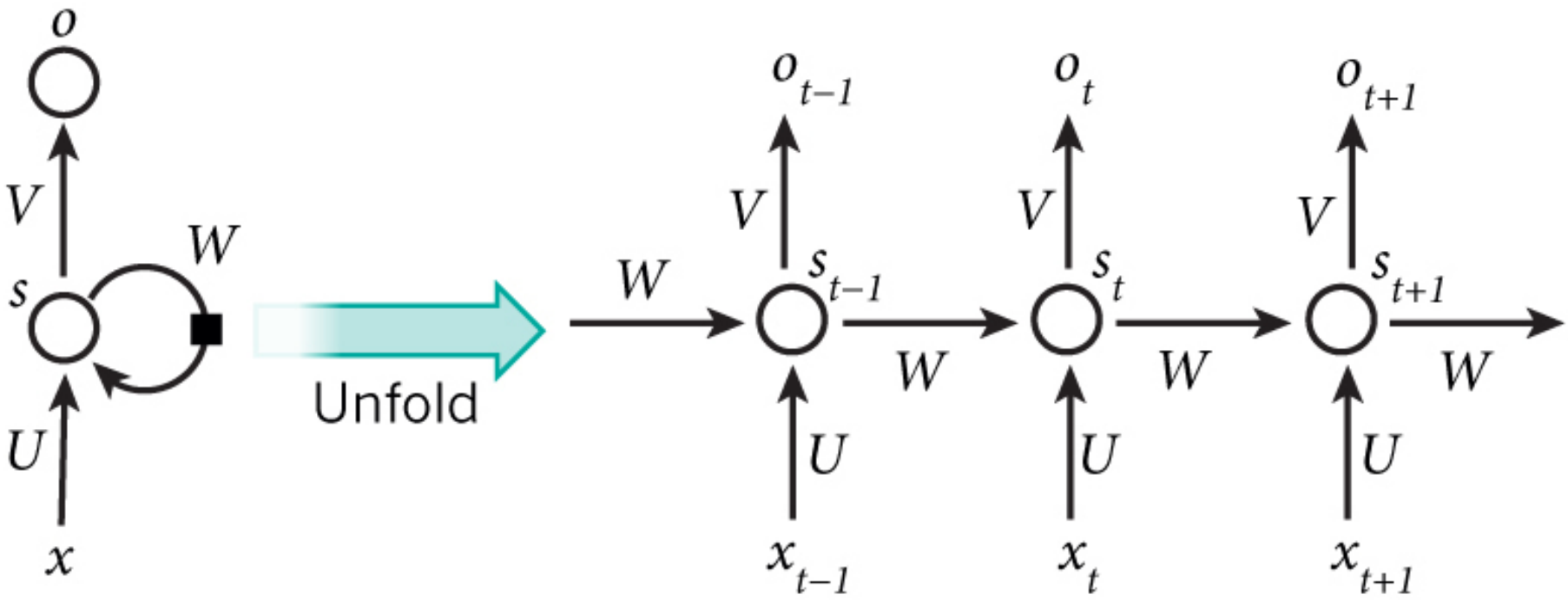
seitliche Rückkopplung:

$C \supseteq (U_{\text{in}} \times U_{\text{in}}) \cup \left(\bigcup_{i=1}^{r-3} U^{(i)}_{\text{hidden}} \times U^{(i)}_{\text{hidden}}\right) \cup (U_{\text{out}} \times U_{\text{out}})$

indirekte Rückkopplung von einer hinteren Schicht in eine vordere:

$C \supseteq \left(U^{(1)}_{\text{hidden}} \times U_{\text{in}}\right) \cup \left(\bigcup U^{(j)}_{\text{hidden}} \times U^{(i)}_{\text{hidden}}\right) \ldots \cup \left(U_{\text{out}} \times U^{(r-2)}_{\text{hidden}}\right)$

Recurrent Neural Nets



LeCun, Bengio & Hinton. ”Deep Learning.” *nature* 521.7553 (2015)

Motivation

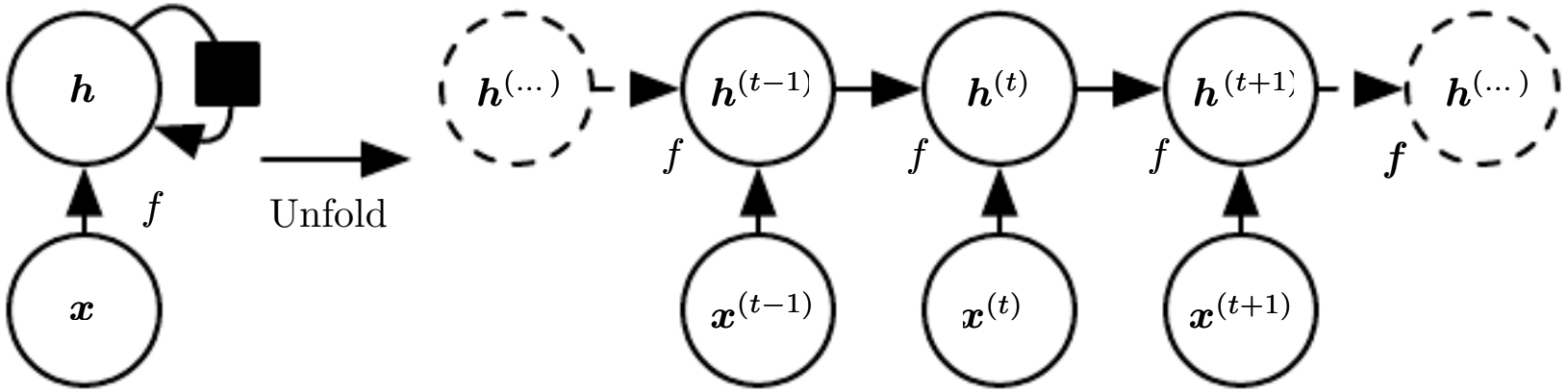
- process sequential data
- capture history of inputs/states
- share parameters through a very deep computational graph
  - output is a function of the previous output
  - produced using the same update rule applied to the previous outputs.
- different from convolution across time steps

Cyclic Connections

- computation includes cycles (recursion)
- represent influence of the present value of a variable on its own value at future time steps

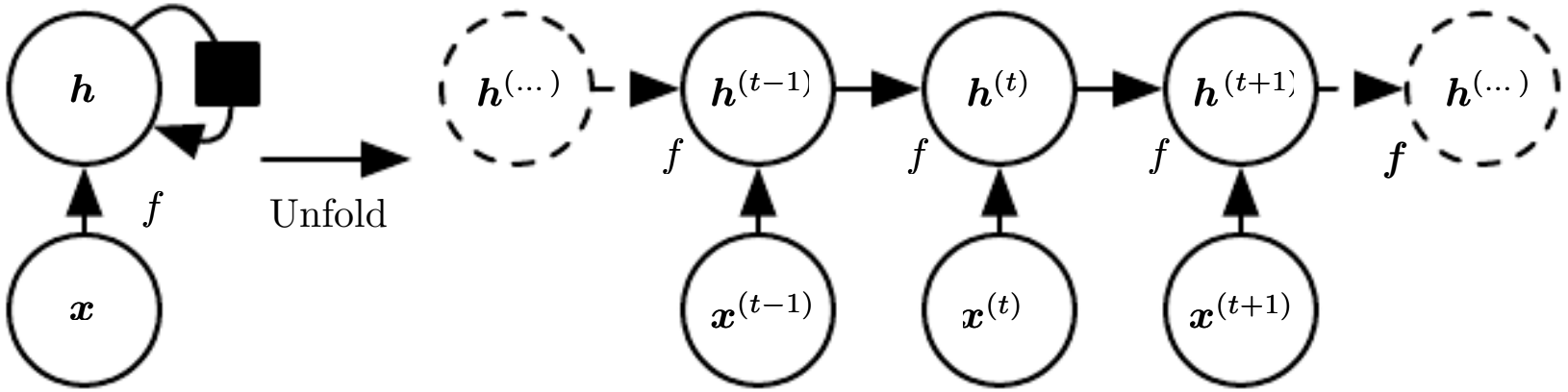
- unfolding to yield a graph that does not involve recurrence
  => gets very deep very quickly

Unfolding



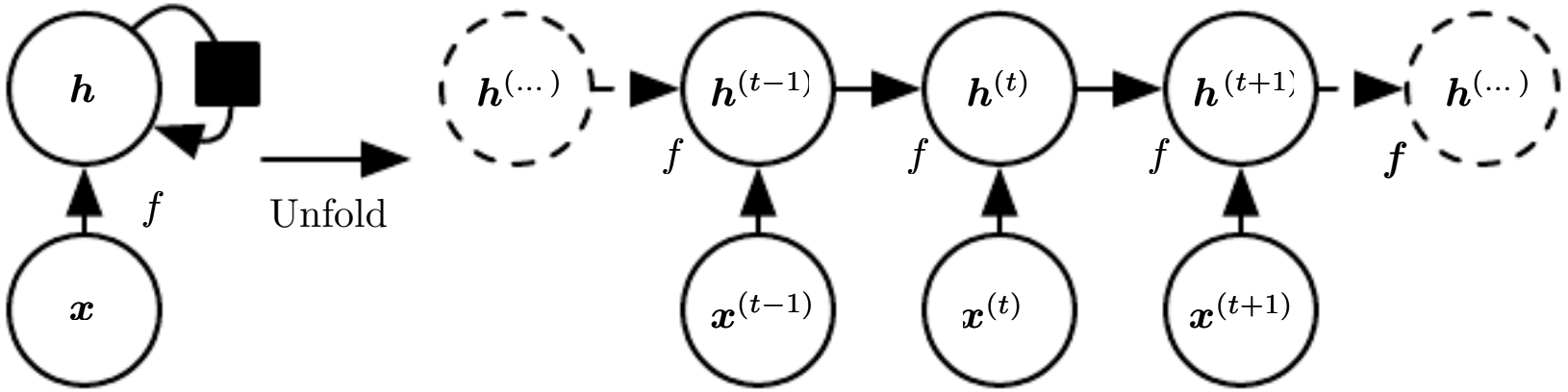
1. regardless of the sequence length, the learned model always has the same input dimensionality
2. can use the same transition function f with the same parameters at every time step

Back-Propagation Through Time (BPTT)



- gradient computation for unfolded loss function w.r.t. parameters very expensive
- O(T) where T is history length
- no parallelization (sequential dependence)

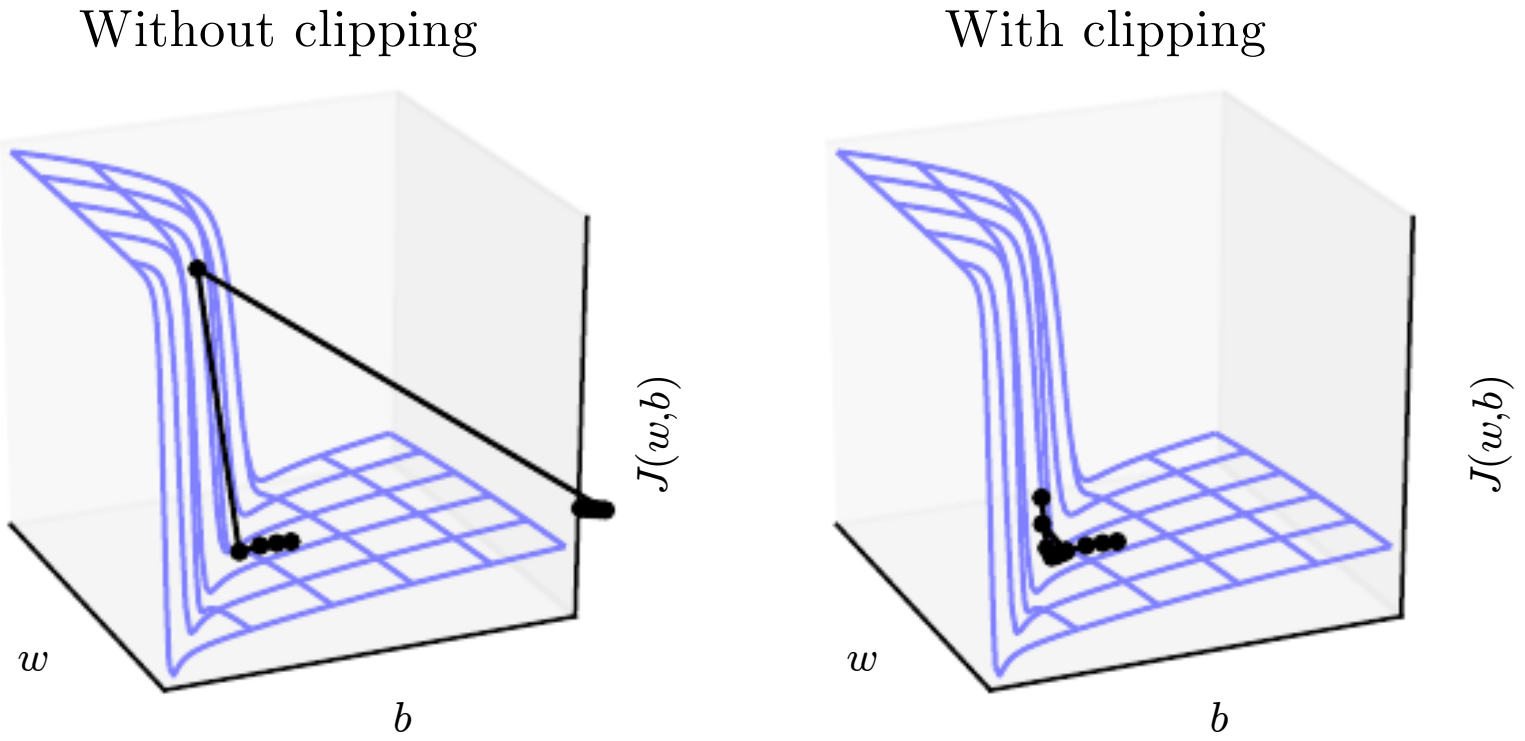
RNN Challenges



- repeated application of the same operation f
  - exploding gradients
  - vanishing gradients
- long-term dependencies

Gradient Clipping

to counteract exploding gradients



Dynamic System

- network now contains information about the whole past sequence:
  - inputs
  - states
  - outputs

Hidden State

- h(t) as a kind of “lossy summary” of the task-relevant aspects of the history up to t
  - lossy compression necessary
  - selectivity based on training criterion (cost)

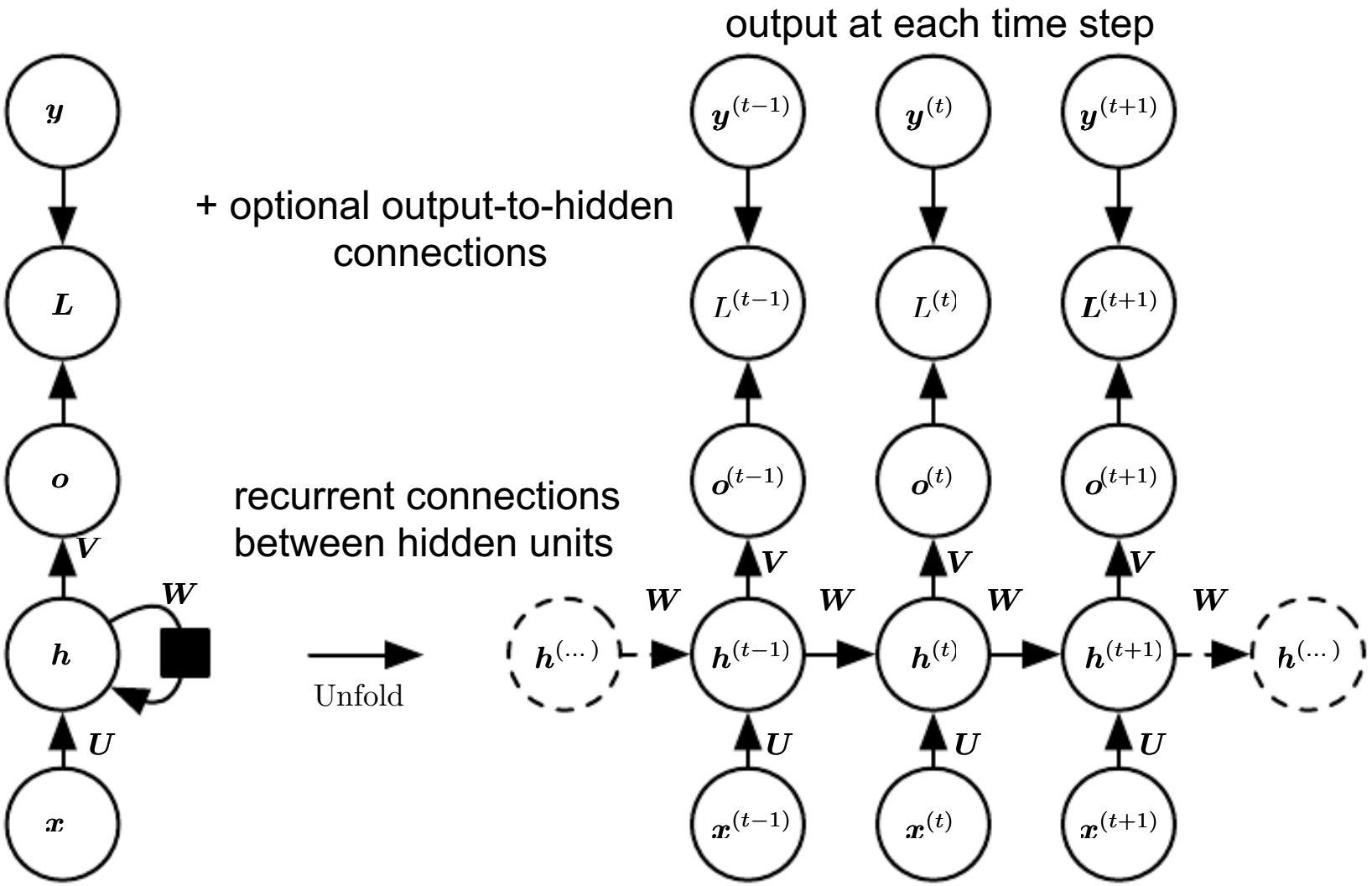
- most demanding situation: rich enough representation h(t) to allow approximate recovery of input sequences (autoencoder)

Design Choices

1. input: vector or sequence
2. output: vector or sequence
3. sync vs. async
4. forward vs. bi-directional
5. output-to-hidden connections?
6. cell types
7. attention
8. …

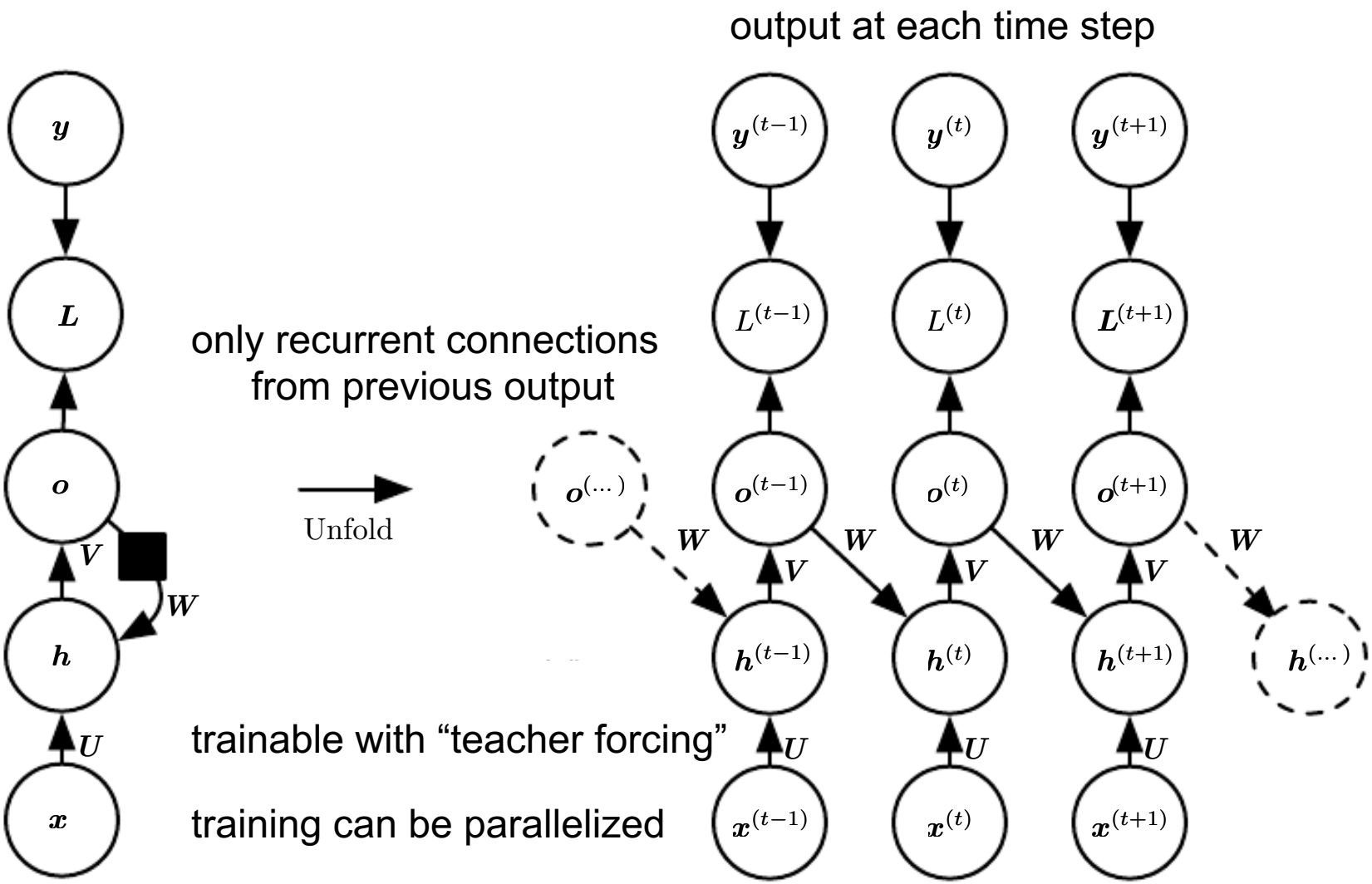
Basic RNN Architectures

sequence to sequence (same length)



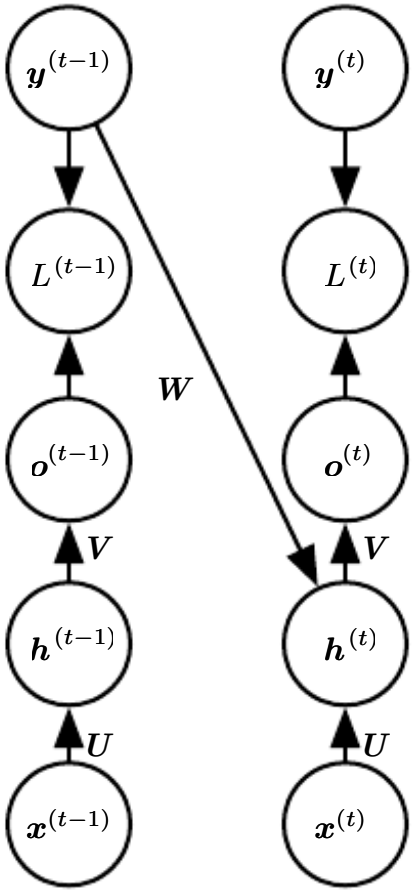
can compute any function computable by a Turing machine (universal function approximator)

sequence to sequence (same length)



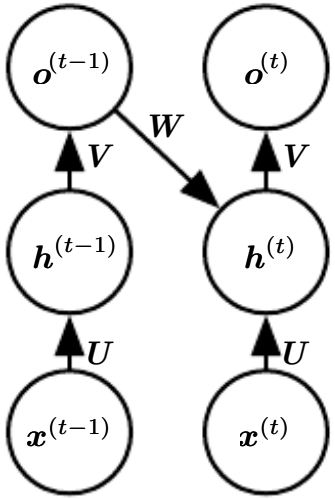
Teacher Forcing

- use targets as prior outputs
- time steps decoupled
- training parallelizable



Train time

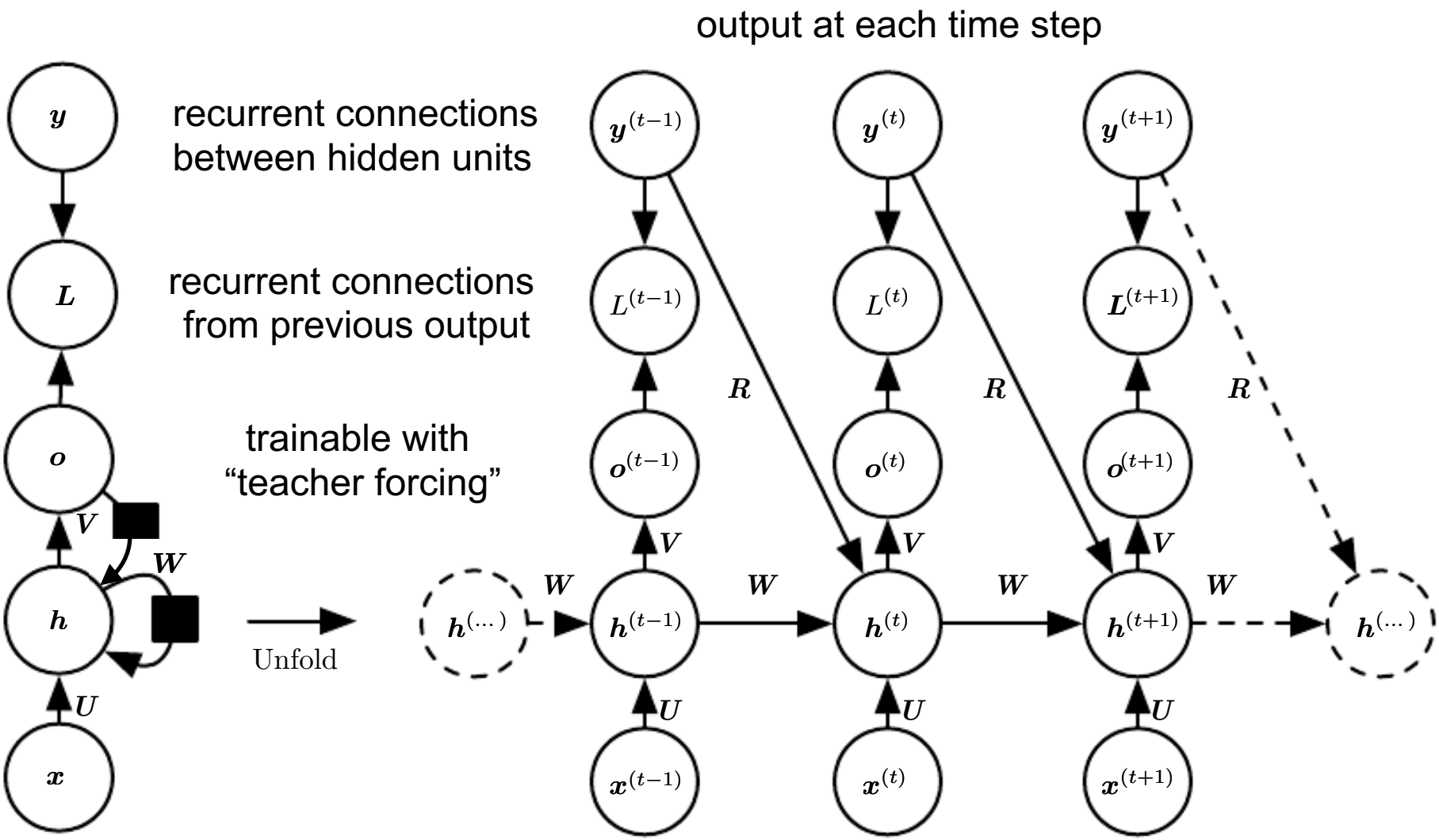
approximate correct output



Test time

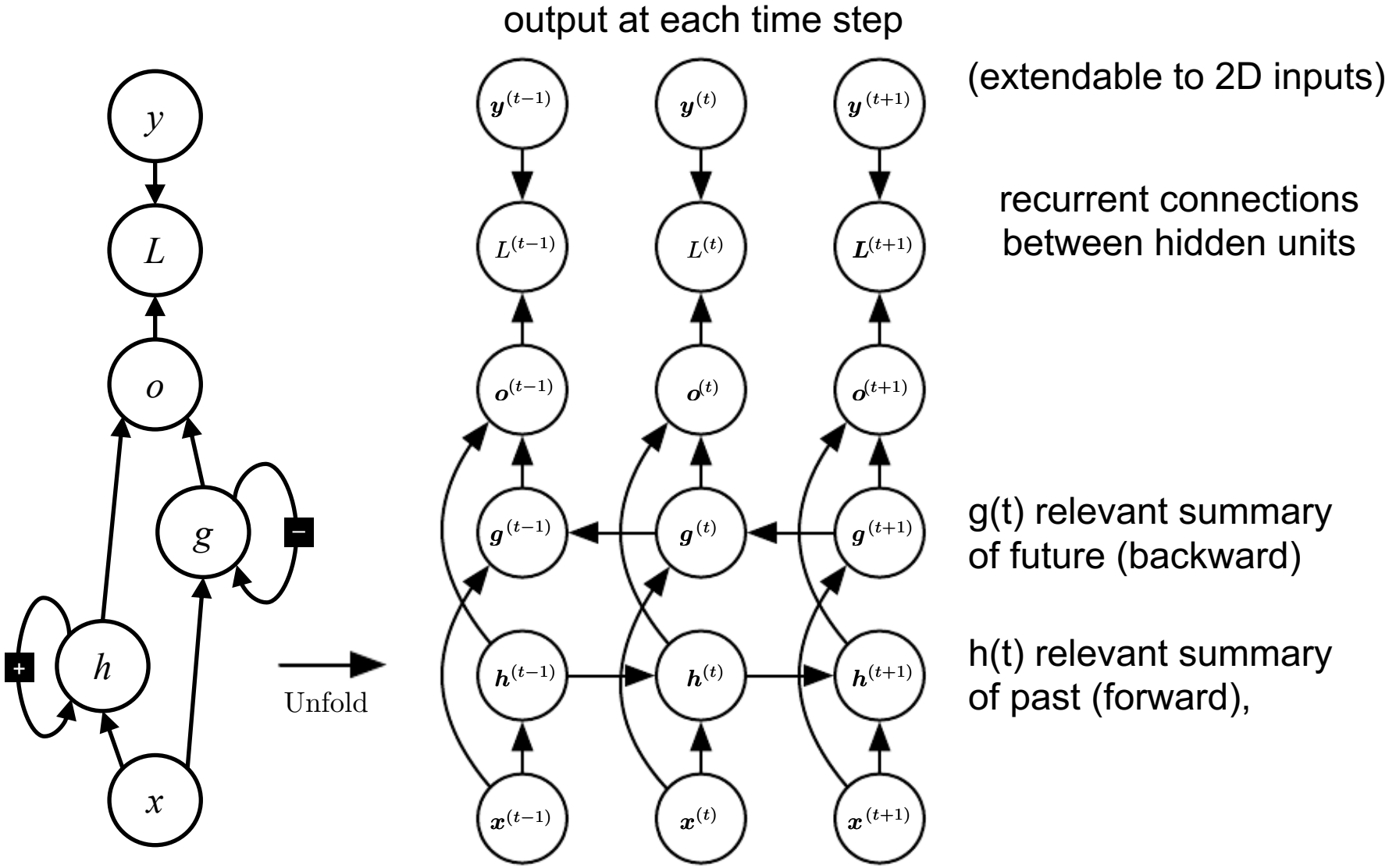
(may also be applied to RNNs with additional hidden-to-hidden connections)

sequence to sequence (same length)



can model arbitrary distribution over sequences of y given sequences of x

bi-directional sequence to sequence (same length)



(extendable to 2D inputs)

recurrent connections between hidden units

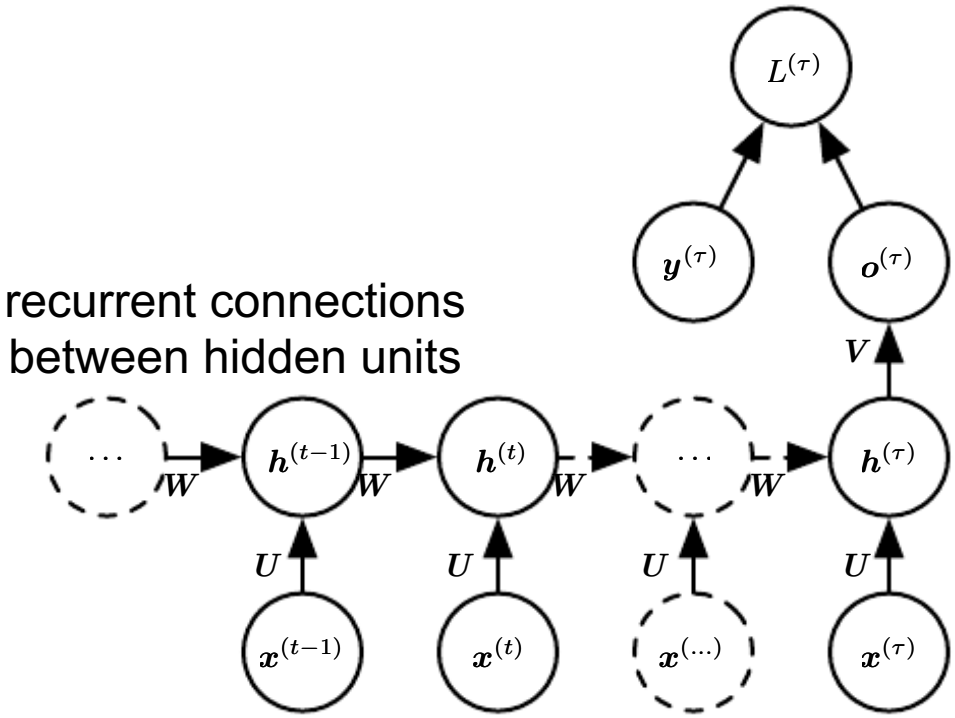
g(t) relevant summary of future (backward)

h(t) relevant summary of past (forward),

can model dependencies on both the past and the future

sequence to fixed-size vector

output after full input sequence has been read



encoder (reader): read input sequence, generate hidden state
(= encoder part of encoder-decoder architecture)

fixed-size (“context”) vector to sequence

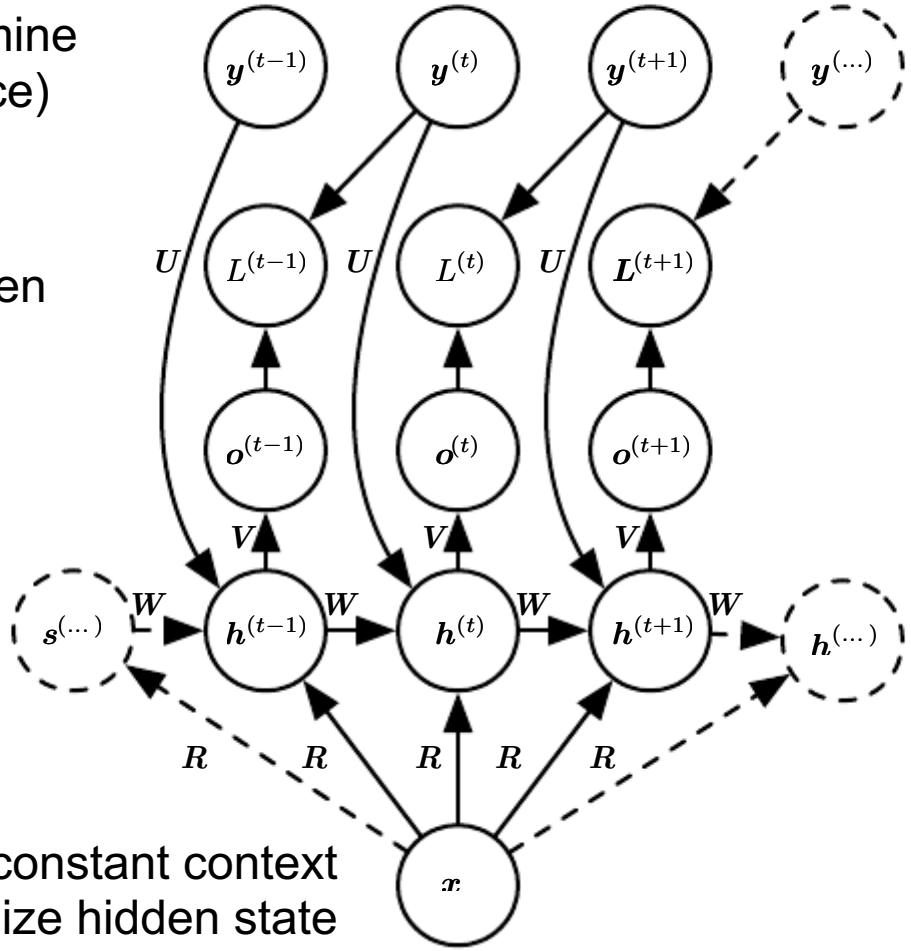
strange indexing (stressing prediction of next output)

(needs to determine end of sequence)

recurrent connections from [previous] output (usually with output-to-hidden connections)

trainable with “teacher forcing”

recurrent connections between hidden units



input x serves as constant context or / and to initialize hidden state

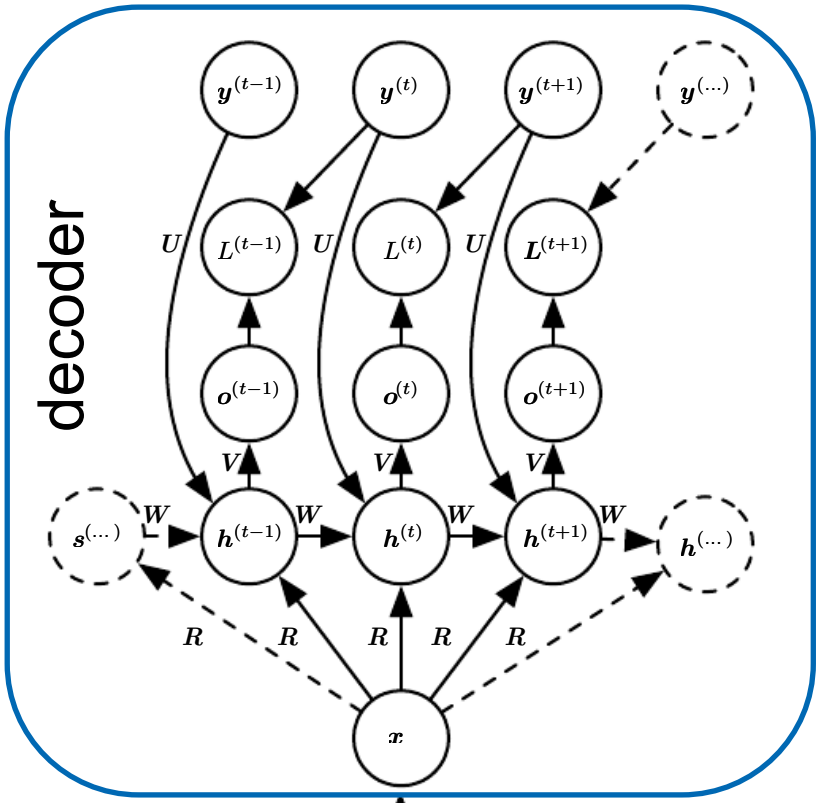
decoder (writer): generate output sequence from hidden state
(= decoder part of encoder-decoder architecture)

sequence to sequence (variable length)

encoder-decoder architecture

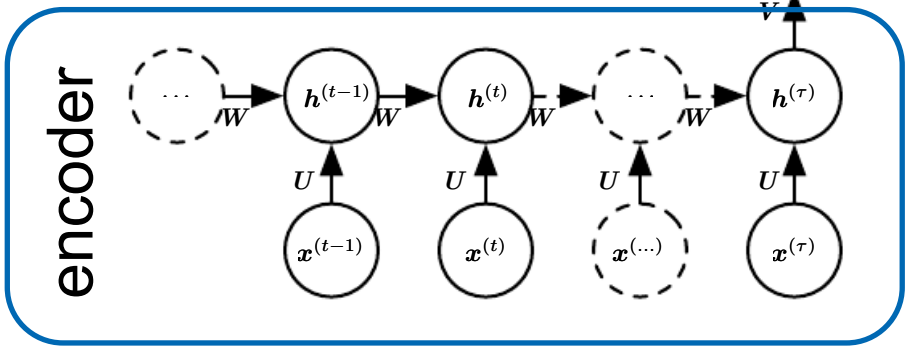
decoder (writer):
generate output sequence
from hidden state

recurrent connections
from [previous] output

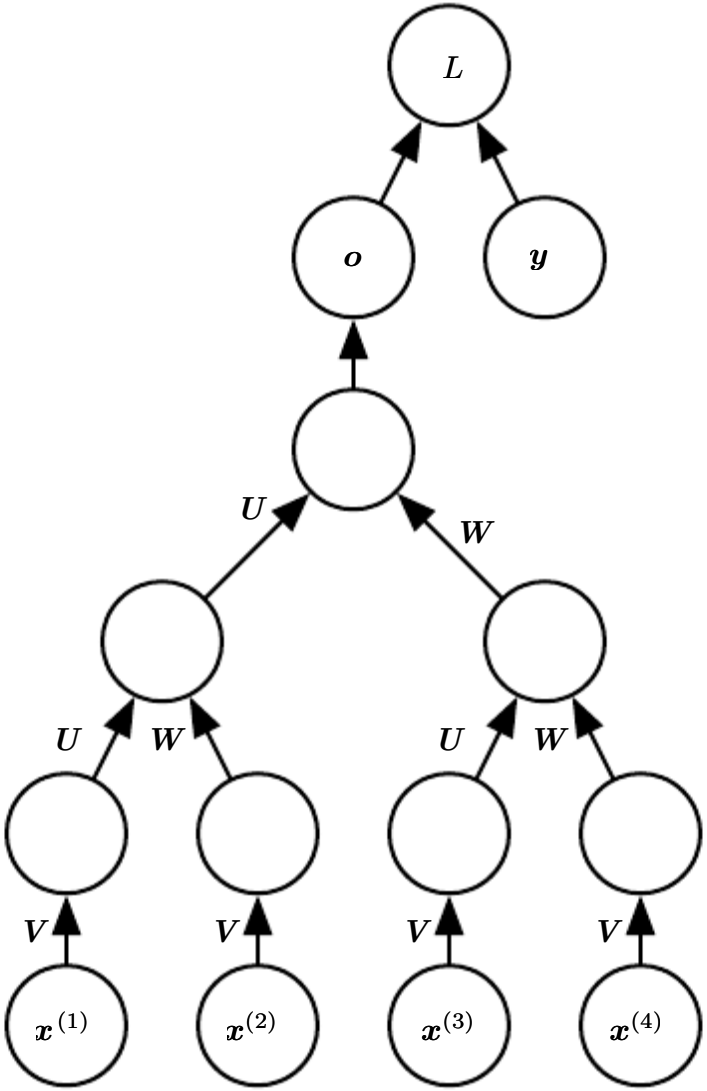


encoder (reader):
read input sequence,
generate hidden state

recurrent connections
between hidden units



complex structure to fixed-size vector



recursive neural network

generalization of RNNs

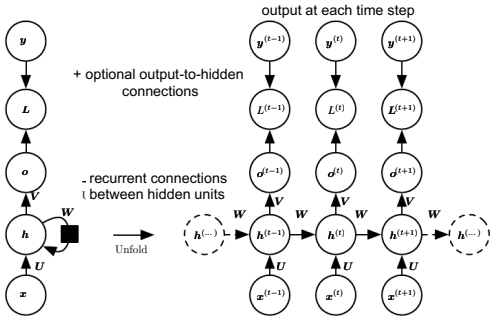
computational graph (given from external tool such as parser) structured as deep tree

Recursive NNs

- generalization of RNNs
- computational graph structured as deep tree
  - reduces to sequence in RNNs
- can process complex data structures
  - e.g. parse trees

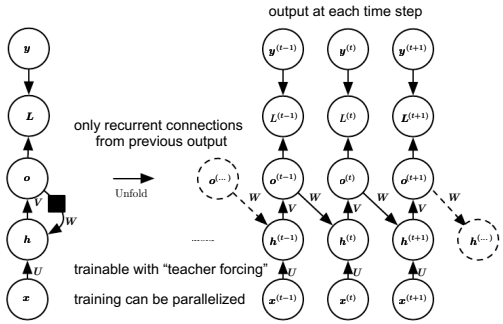
recursive networks

sequence to sequence (same length)



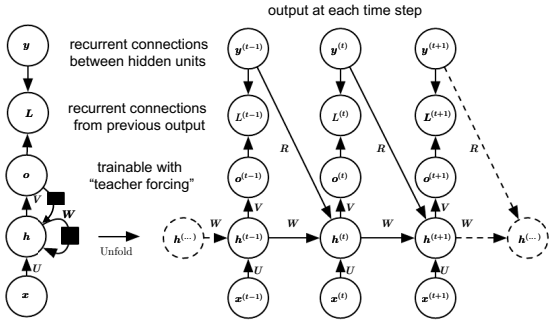
can compute any function computable by a Turing machine (universal function approximator)

sequence to sequence (same length)



lacks important information from past unless o is very high-dimensional & rich

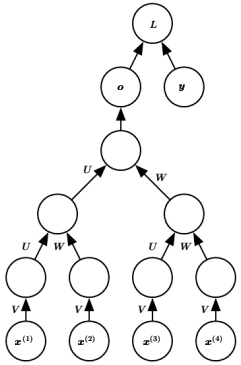
sequence to sequence (same length)



can model arbitrary distribution over sequences of y given sequences of x

sequence-to-sequence

complex structure to fixed-size vector



recursive neural network

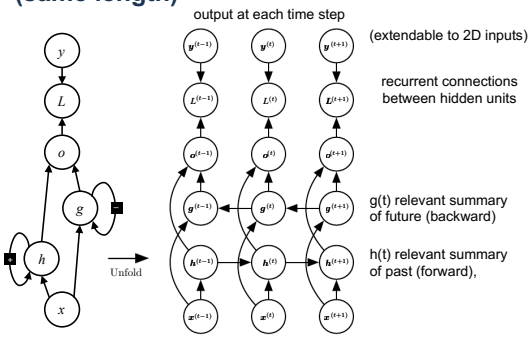
generalization of RNNs

computational graph (given from external tool such as parser) structured as deep tree

generalization

RNNs

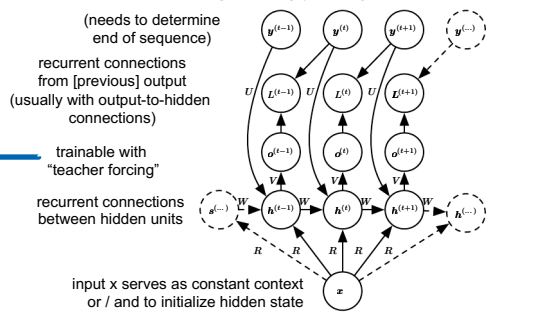
bi-directional sequence to sequence (same length)



can model dependencies on both the past and the future

bi-directional

fixed-size ("context") vector to sequence

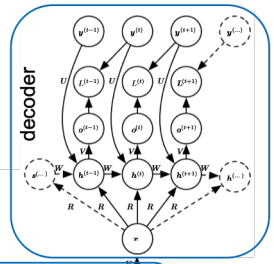


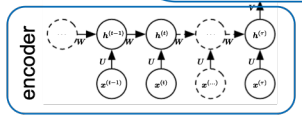
decoder (writer): generate output sequence from hidden state (= decoder part of encoder-decoder architecture)

decoder

sequence to sequence (variable length)

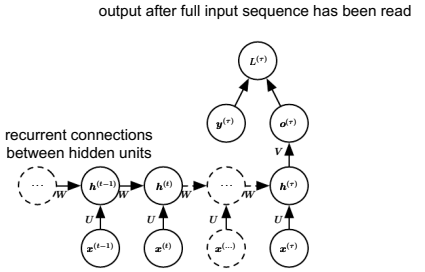
encoder-decoder architecture





encoder-decoder

sequence to fixed-size vector



encoder (reader): read input sequence, generate hidden state (= encoder part of encoder-decoder architecture)

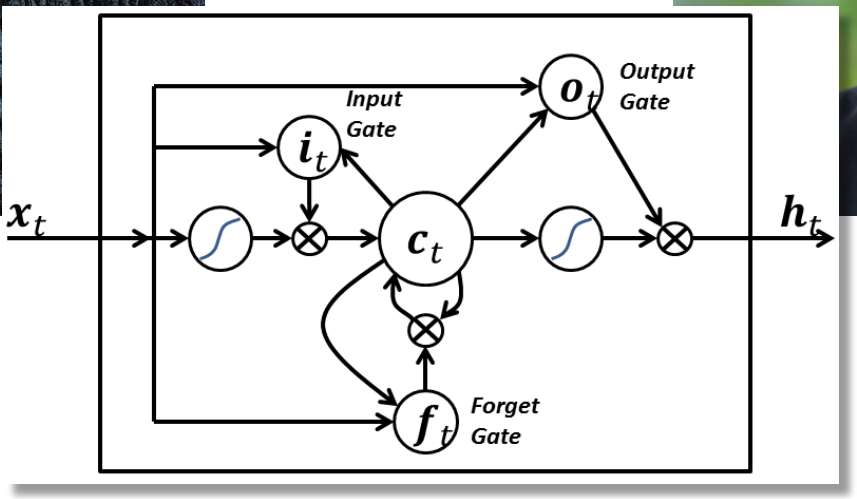
encoder

Complex Recurrent Units

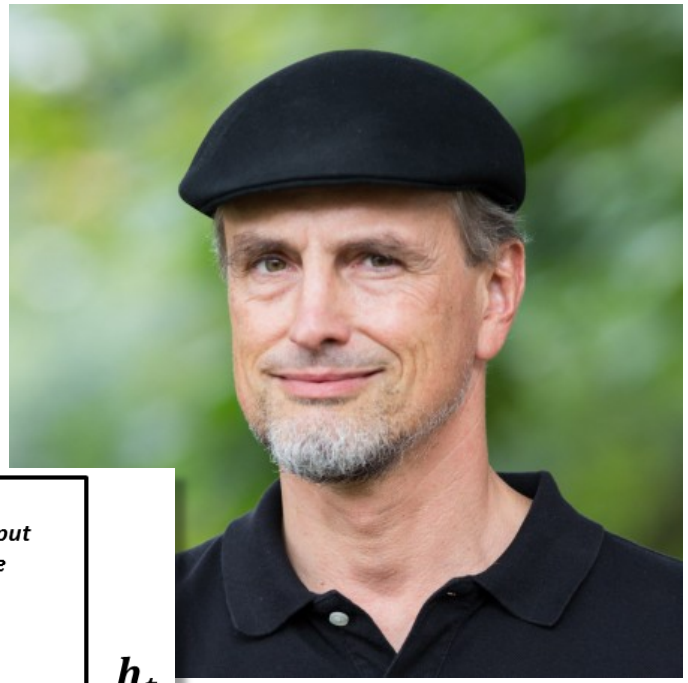
LSTM & GRU

Long Short-Term Memory Units (LSTMs, 1991)

Sepp Hochreiter
JKU Linz

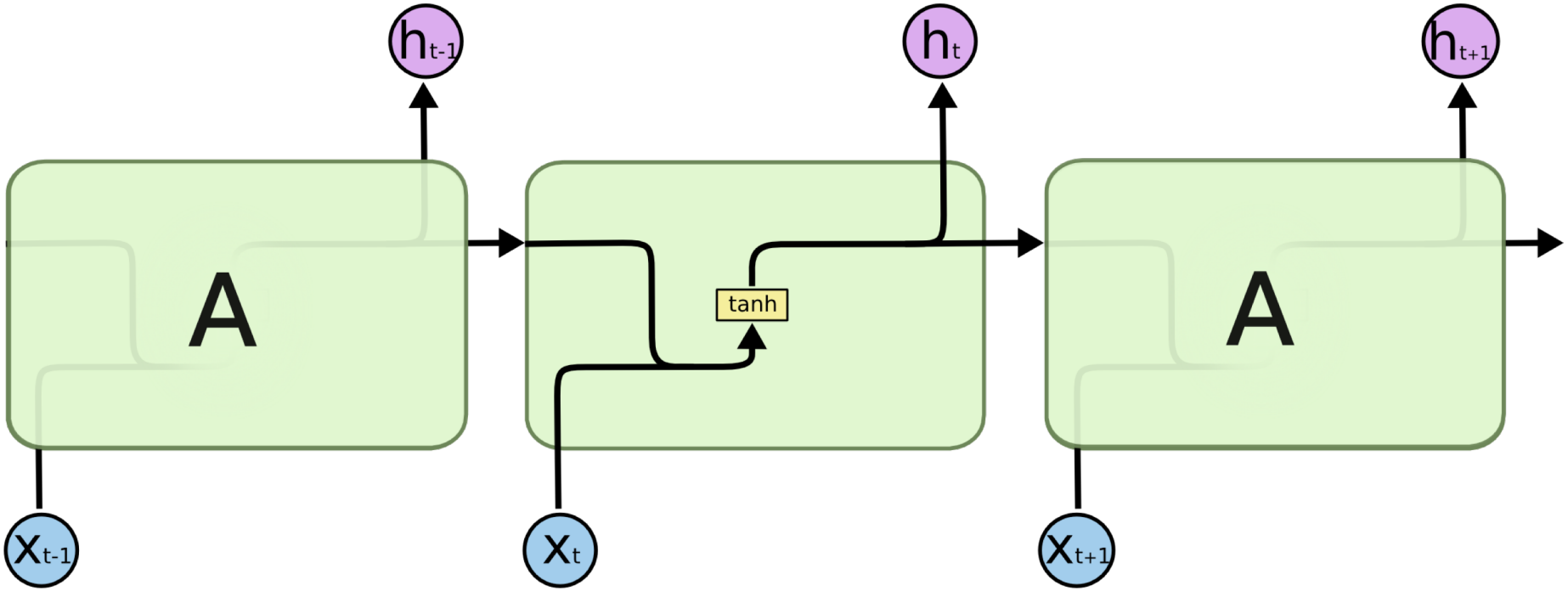


neuron with explicit memory

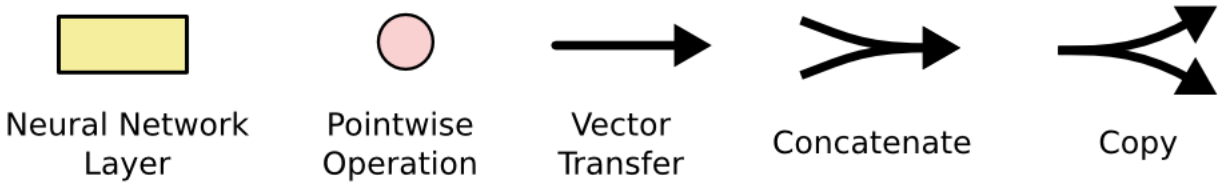


Jürgen Schmidhuber
IDSIA, Lugano

Simple RNN

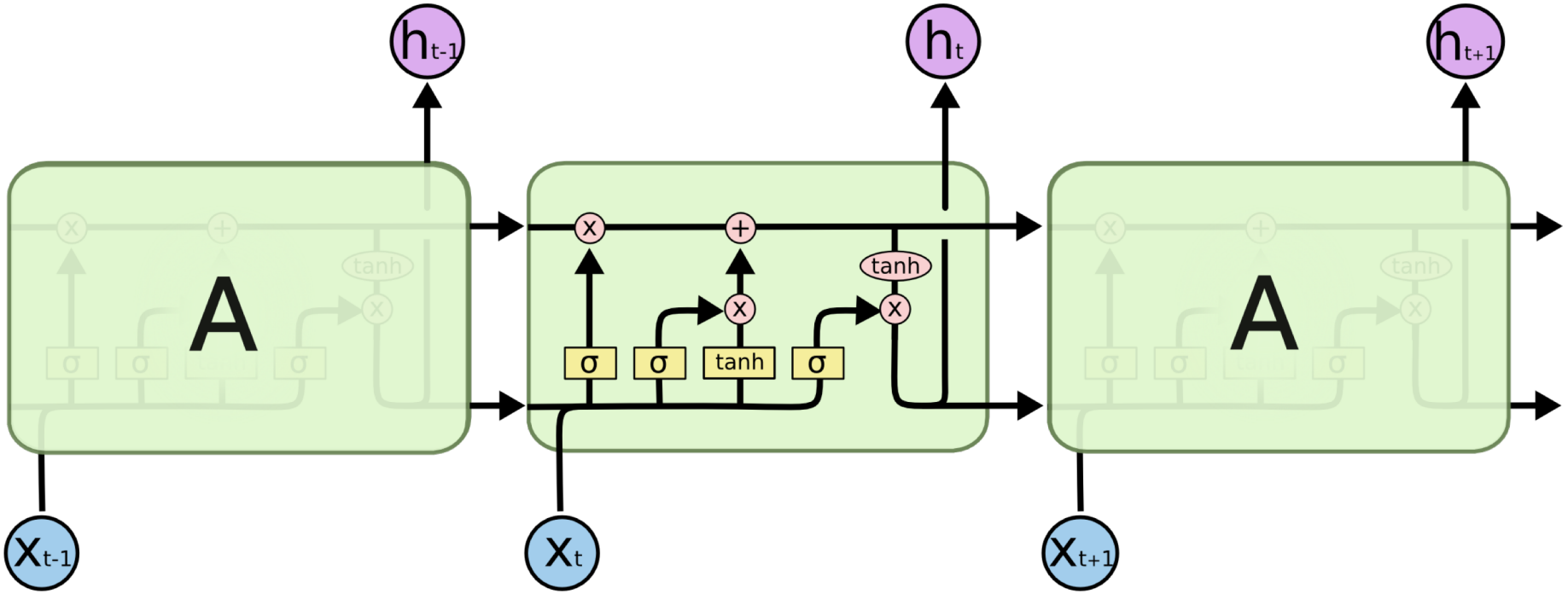


The repeating module in a standard RNN contains a single layer.

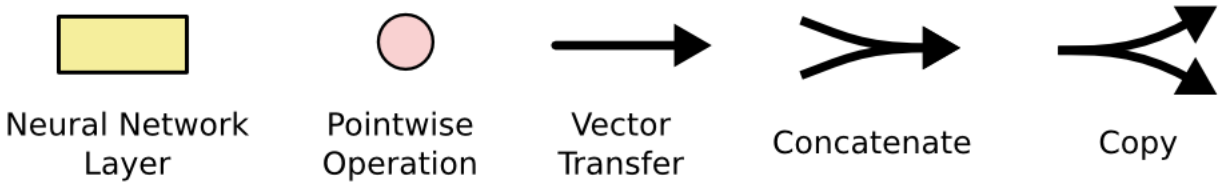


https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM

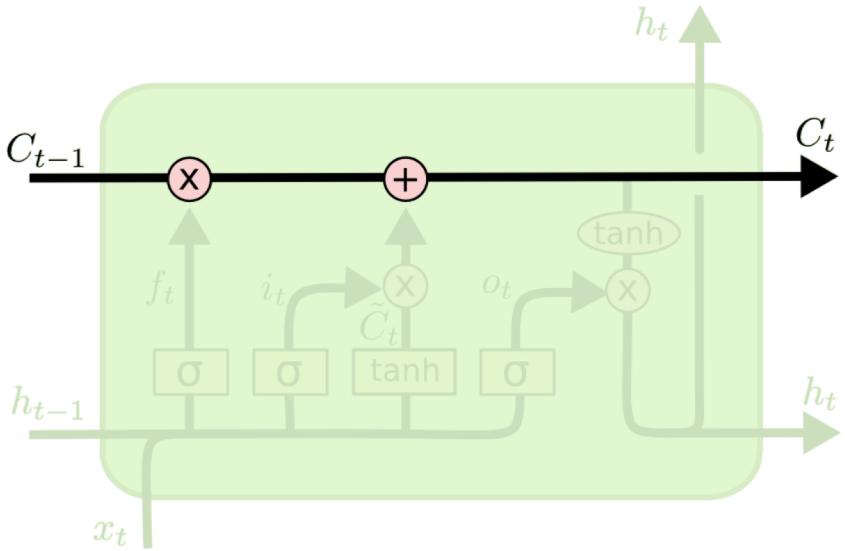


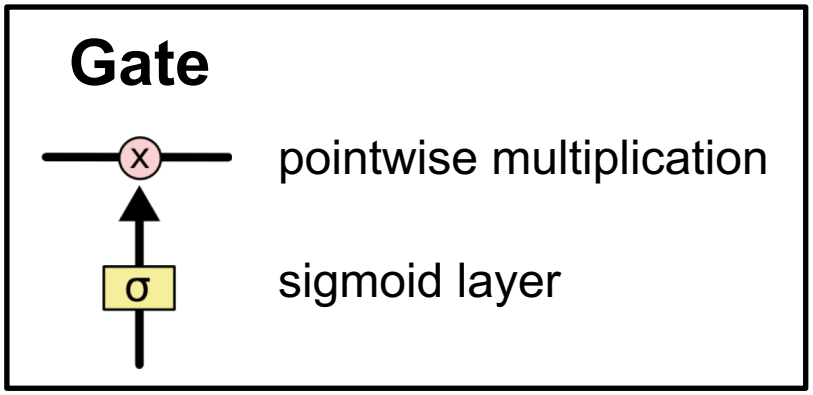
The repeating module in an LSTM contains four interacting layers.



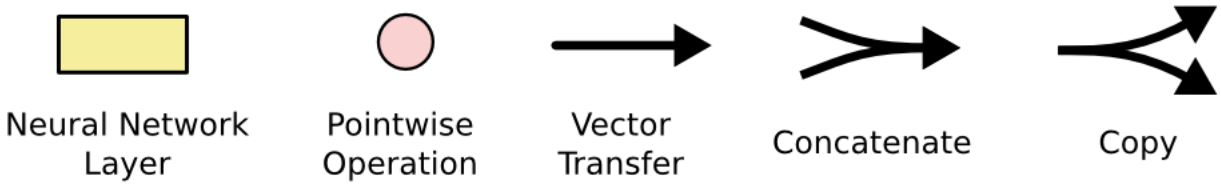
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Cell State



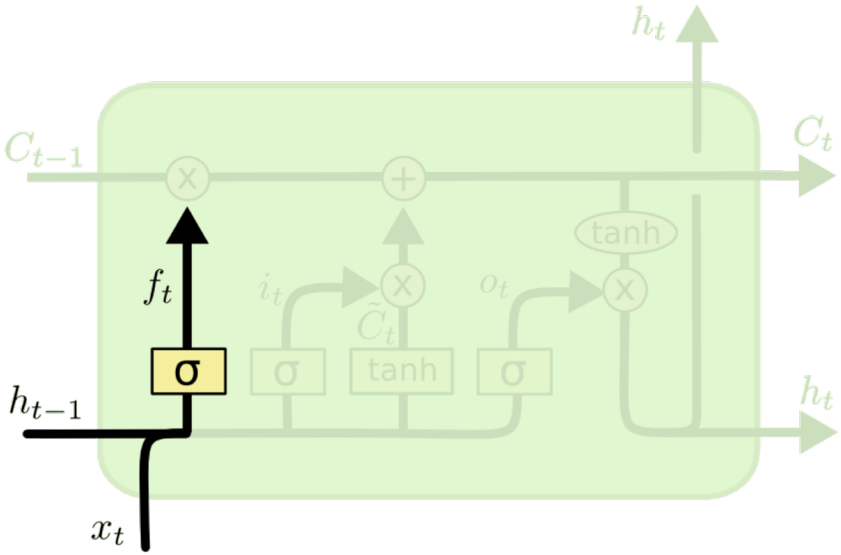


Removing or adding information to the cell state is controlled by gates.



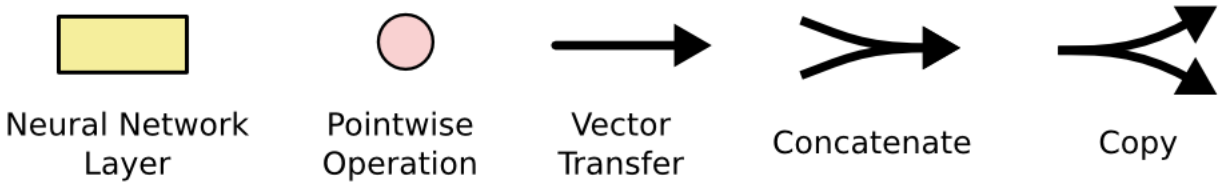
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Forget Gate



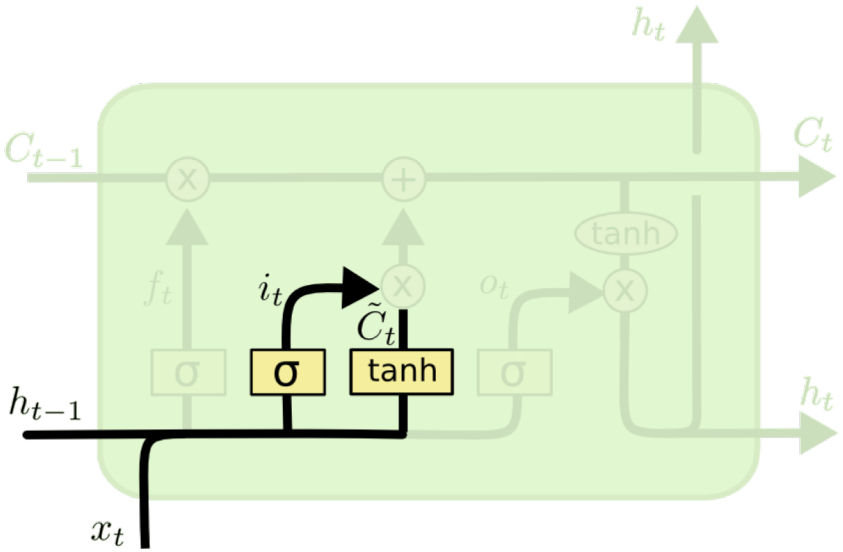
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Decide what information from cell state is deleted (0) or kept (1).



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

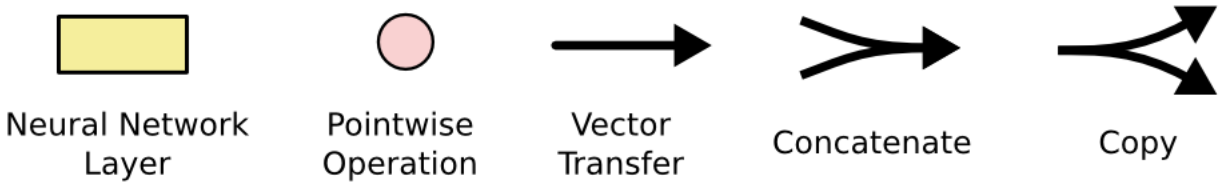
LSTM Input Gate



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

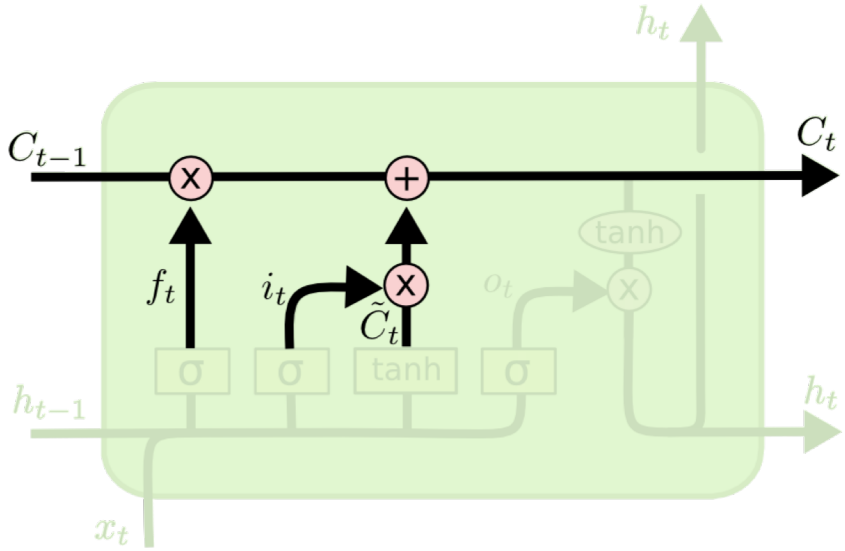
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Decide what new information to store.



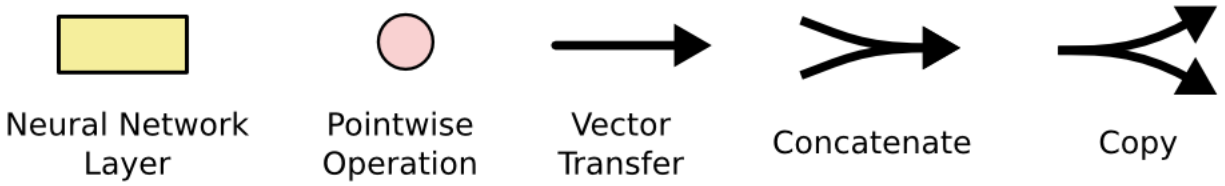
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Cell State Update



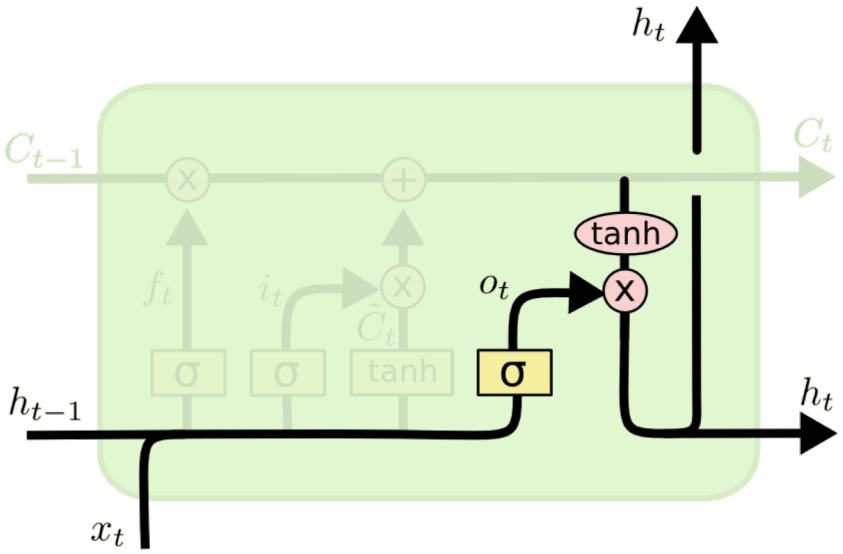
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Delete information and add new one.



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

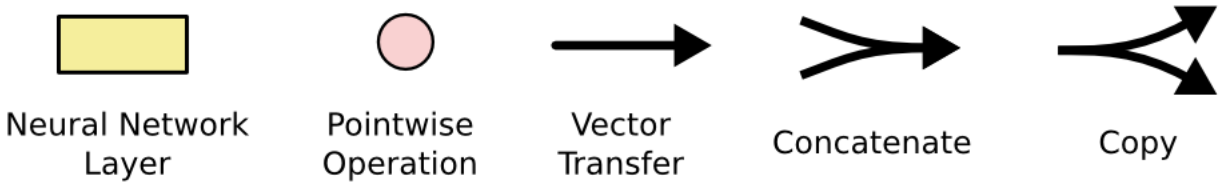
LSTM Output Gate



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

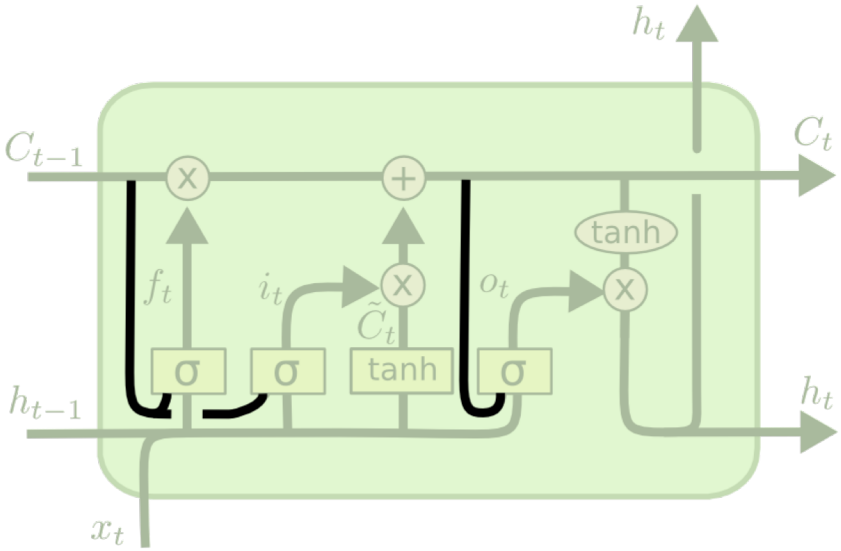
$$h_t = o_t * \tanh\left(C_t\right)$$

Transform state and decide what to output.



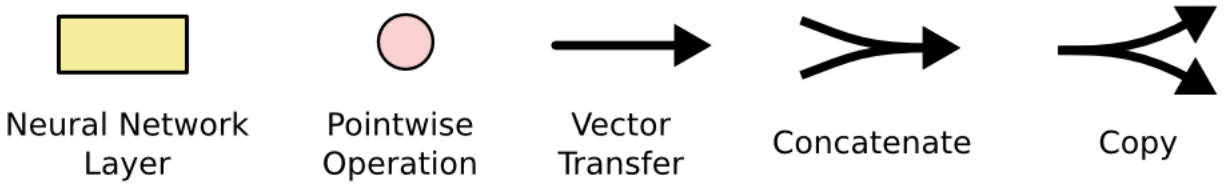
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Option: Peep Hole Connections



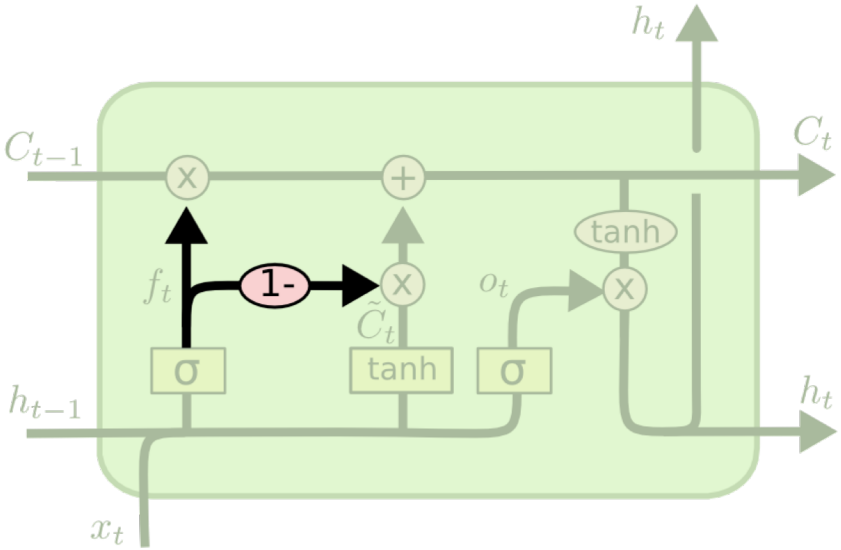
$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] + b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] + b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] + b_o\right)$$

Allow gates to look at cell states.



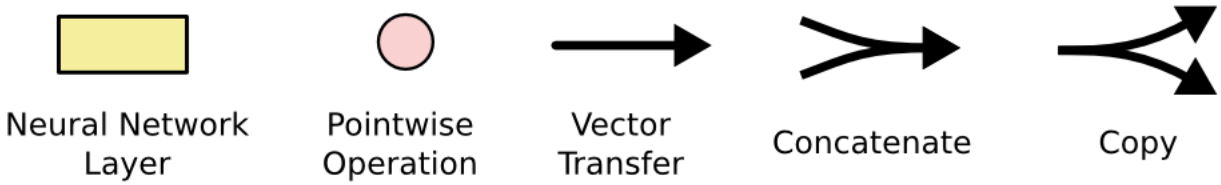
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Coupled Input/Forget Gates



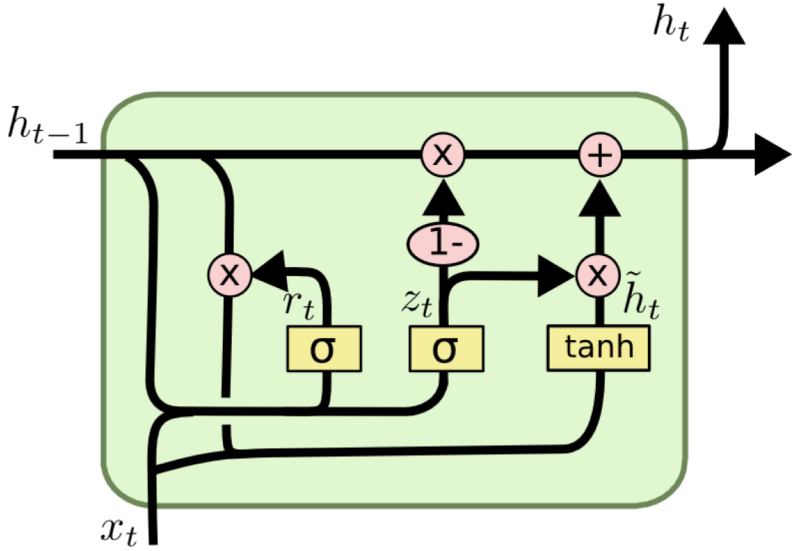
$$C_t = f_t * C_{t-1} + (1 - \boldsymbol{f_t}) * \tilde{C}_t$$

Only input new values to the state when something older gets forgotten.



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

Gated Recurrent Units (GRUs)



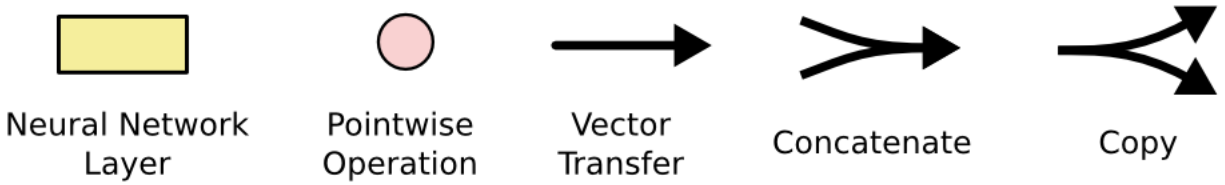
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

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

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

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

Combines forget and input gates into a single update gate.
Merges cell state and hidden state.



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

Example Applications

Optimierung von Windparks

Quelle: Landesregierung Schleswig-Holstein

Eine Windkraftanlage:

Input aus Sensoren:

- Windgeschwindigkeit
- Vibration

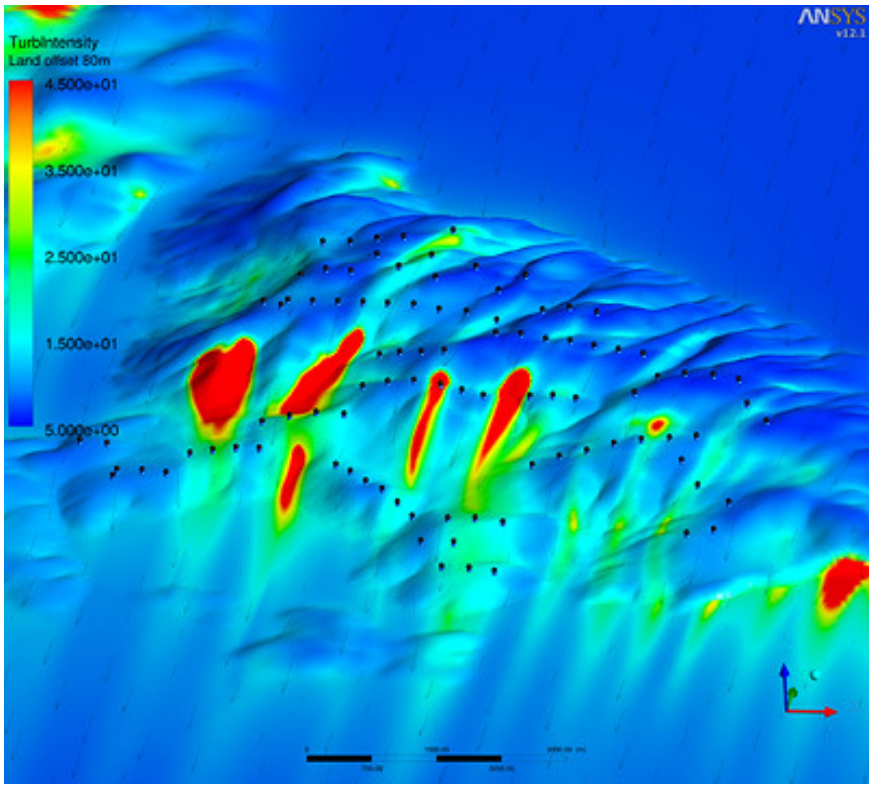
Variablen:

- Stellwinkel der Rotorblätter
- Einstellung des Generators

Ziele:

- Maximierung der Stromgenerierung
- Minimierung von Verschleiß

Optimierung von Windparks



Quelle: Siemens

Viele Windkraftanlagen:

Problem: Windräder verursachen Turbulenzen, die dahinter stehende Windräder beeinträchtigen

Input: Sensordaten aller Windräder

Variablen: Einstellungen aller Windräder

Ziele:

- Maximierung der Summe des erzeugten elektrischen Stroms
- Minimierung von Verschleiß bei allen Windkraftanlagen

Lösung: Verwende ein rekurrentes neuronales Netz

Imitating Shakespeare

PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

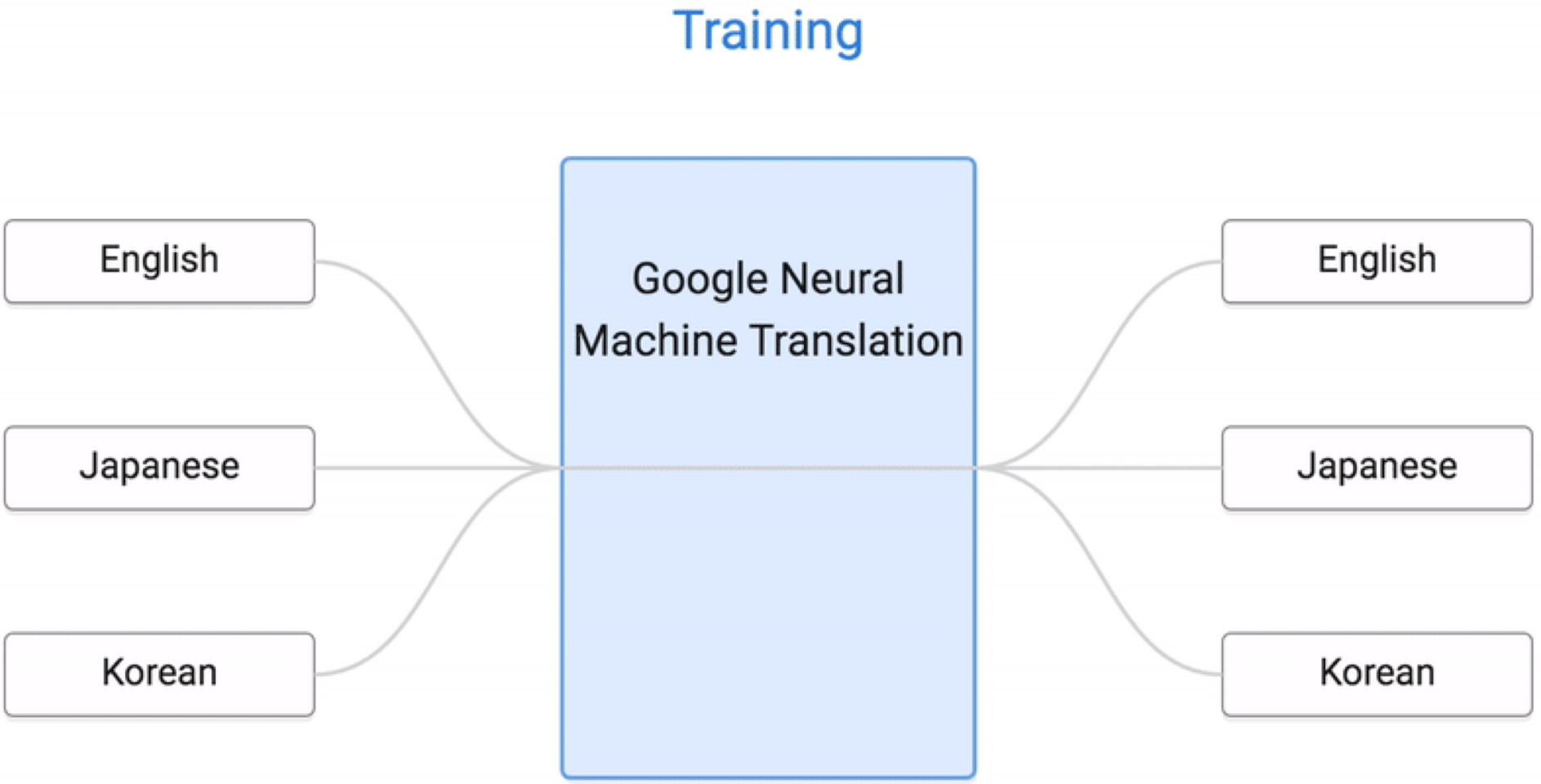
VIOLA:
I'll drink it.

Training:

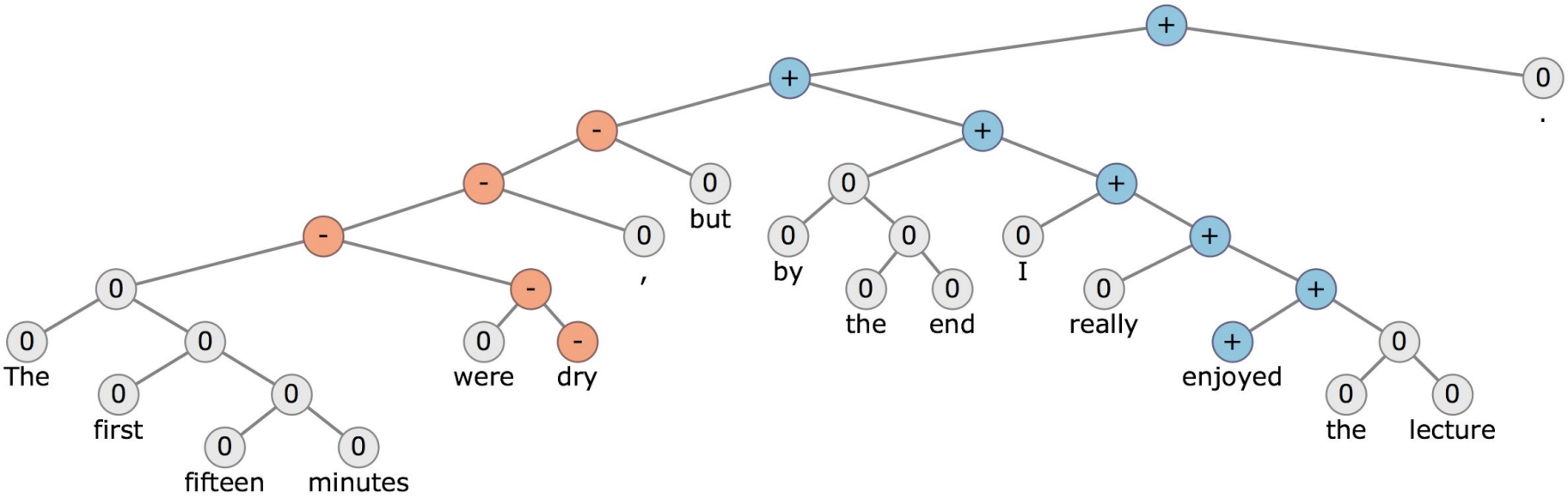
- read Shakespeare character by character and try to **predict the next character**
- **no knowledge of words or grammar!**

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Google Translate



Sentiment Detection



http://nlp.stanford.edu/sentiment

very hot topic, e.g.

- politics (opinion polls, voter analysis, …)
- marketing (product reviews, market analysis, tests, …)
- stock market prediction

Generate Image Captions

Describes without errors	Describes with minor errors	Somewhat related to the image	Unrelated to the image
 A person riding a motorcycle on a dirt road.	 Two dogs play in the grass.	 A skateboarder does a trick on a ramp.	 A dog is jumping to catch a frisbee.
 A group of young people playing a game of frisbee.	 Two hockey players are fighting over the puck.	 A little girl in a pink hat is blowing bubbles.	 A refrigerator filled with lots of food and drinks.
 A herd of elephants walking across a dry grass field.	 A close up of a cat laying on a couch.	 A red motorcycle parked on the side of the road.	 A yellow school bus parked in a parking lot.

Answer Visual Questions

What vegetable is on the plate?
Neural Net: broccoli
Ground Truth: broccoli

What color are the shoes on the person's feet ?
Neural Net: brown
Ground Truth: brown

How many school busses are there?
Neural Net: 2
Ground Truth: 2

What sport is this?
Neural Net: baseball
Ground Truth: baseball

What is on top of the refrigerator?
Neural Net: magnets
Ground Truth: cereal

What uniform is she wearing?
Neural Net: shorts
Ground Truth: girl scout

What is the table number?
Neural Net: 4
Ground Truth:40

What are people sitting under in the back?
Neural Net: bench
Ground Truth: tent

https://avisingh599.github.io/deeplearning/visual-qa/

RNNs

- work well for sequential data
  - time series (with low sampling rate)
  - texts (translation, discourse, sentiment, ...)
- support variable-length input
  - including long-term dependencies
- are hard to parallelize