Advanced NLP - Session 2: RNNs & LSTMs

Prof. Dr. Richard Sieg
TH Köln IWS - WS 25/26

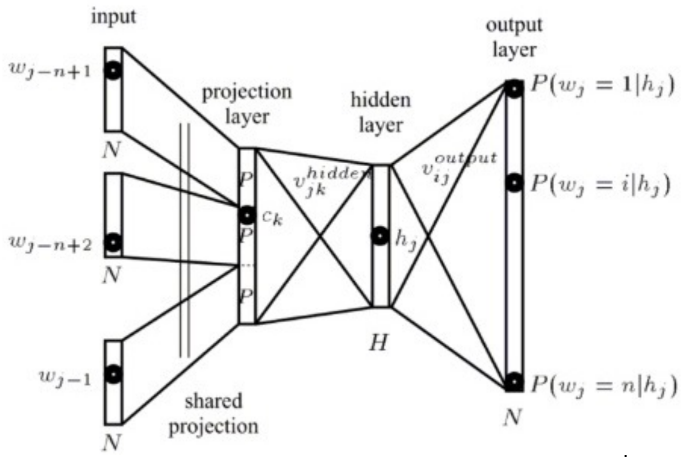
Agenda

Recap: Language Modeling with Neural Networks

- The conditional probability can be expressed as a *softmax* layer inside a neural network

$$p(w_t|w_{t-1}, \dots, w_{t-n-1}) = \frac{\exp(h^T v'_{w_t})}{\sum_{w_i} \exp(h^T v'_{w_i})}$$

- h: *context vector* (output of the hidden layer on $w_{t-1}, \dots, w_{t-n-1}$)
- v'_{w_t}: the embedding to be learned for the token/word w_t
- Objective function: Maximize probability for the correct word



Issues

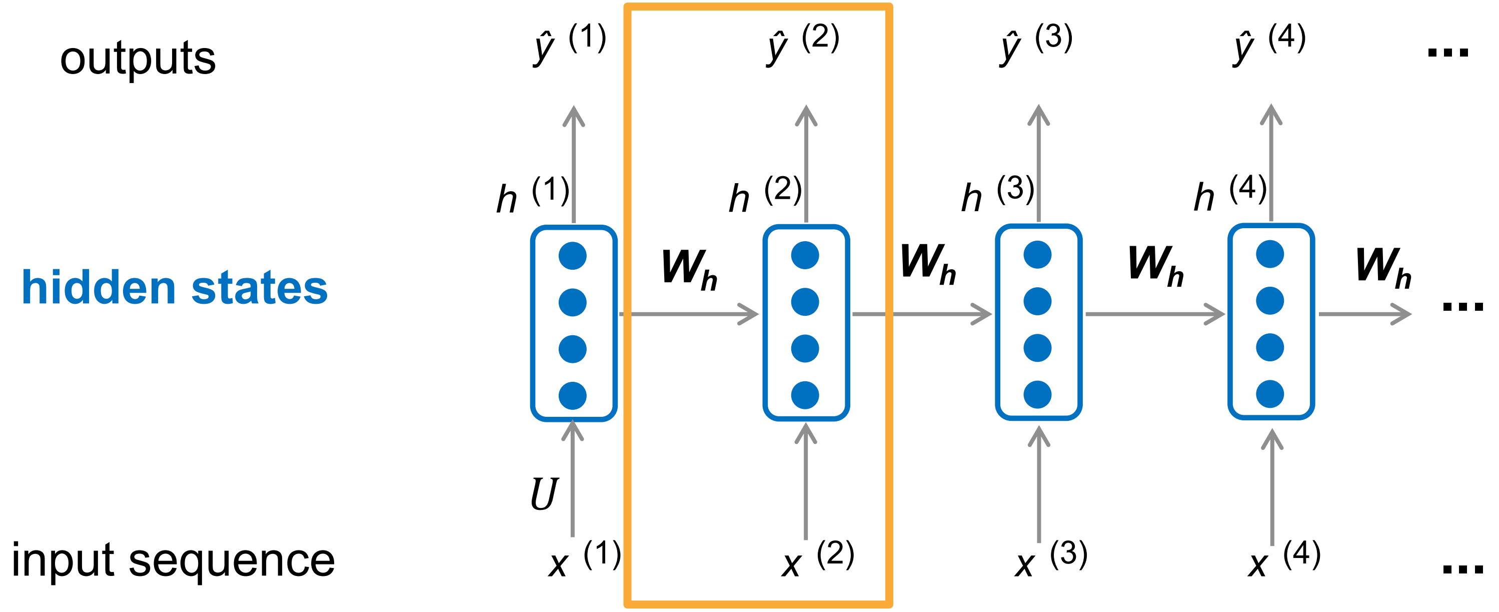
- Learned embedding of a word is independent of context
  - e.g. *mouse* in “the computer mouse” or “the mouse likes cheese”
  - Model has now sense of a *sequence of words*
- Only works with a fixed word window
- Enlarging the window also enlarges the model
  - Becomes impractical fast

We need a model that can process sequences of any length

RNNs

Recurrent Neural Networks

$$h^{(t)} = \sigma\left(W_h h^{(t-1)} + U x^{(t)} + b\right)$$



The same weights *W* are applied on every input.

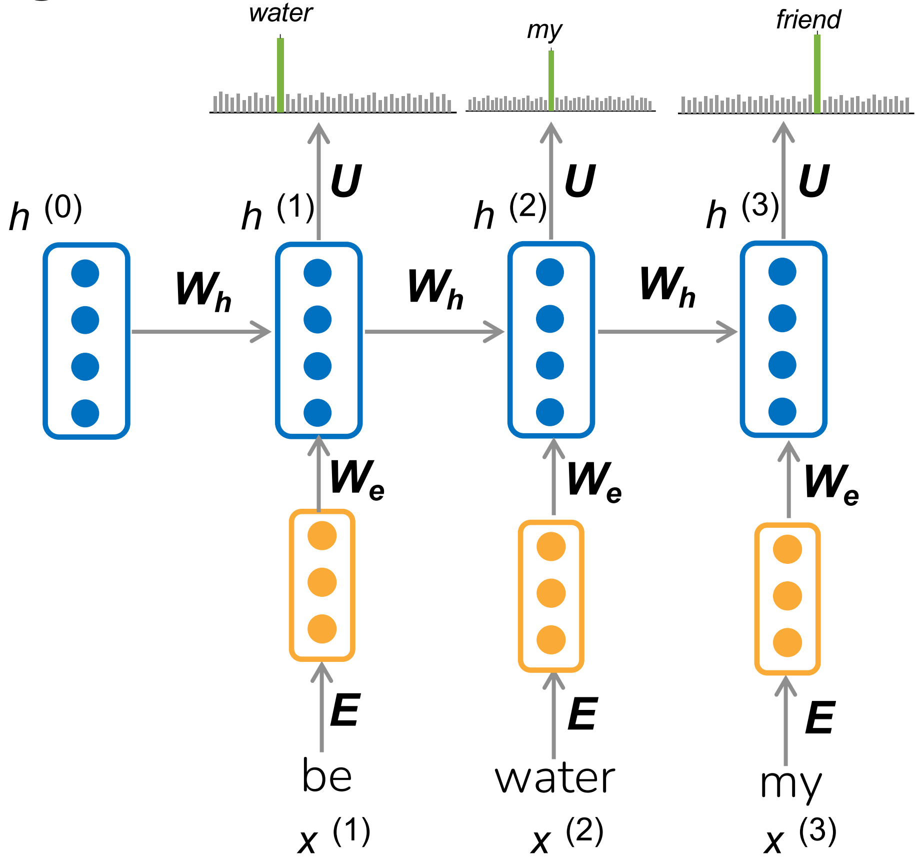
RNN for language modeling

output distribution

hidden states

word embeddings

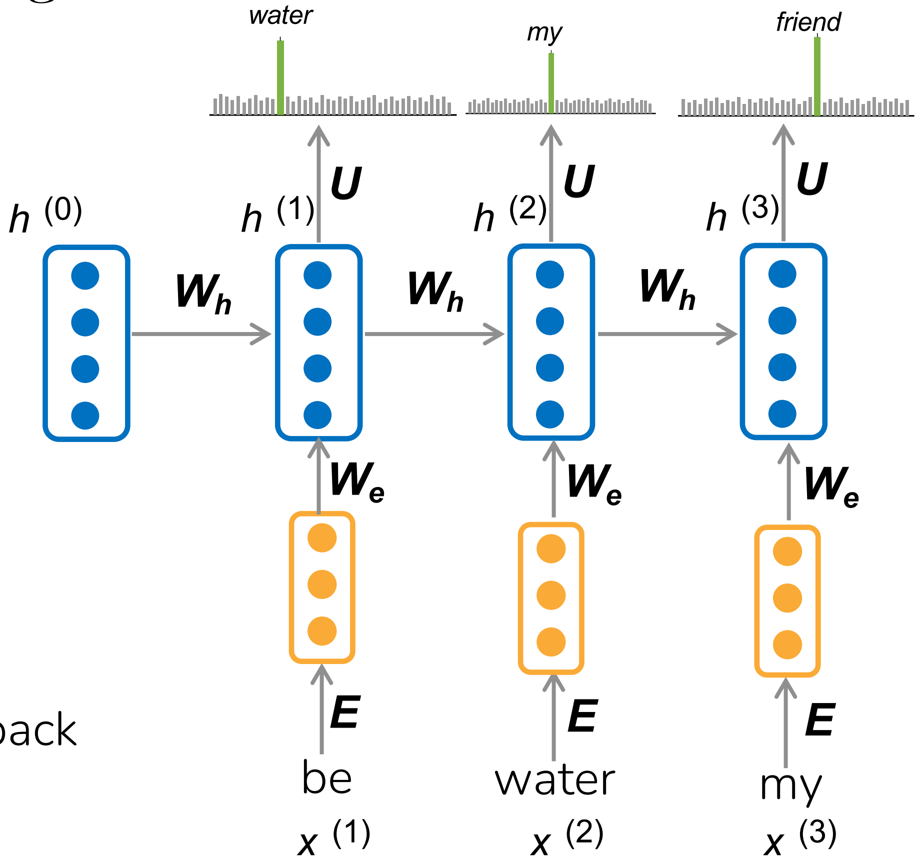
input words as one-hot vectors



RNN for language modeling

+ We can now process sequences of any length
+ Size of the model does not depend on input size
+ Next word prediction can theoretically use information from the whole sequence

- Computation is very slow
- Information from many steps back gets lost easily

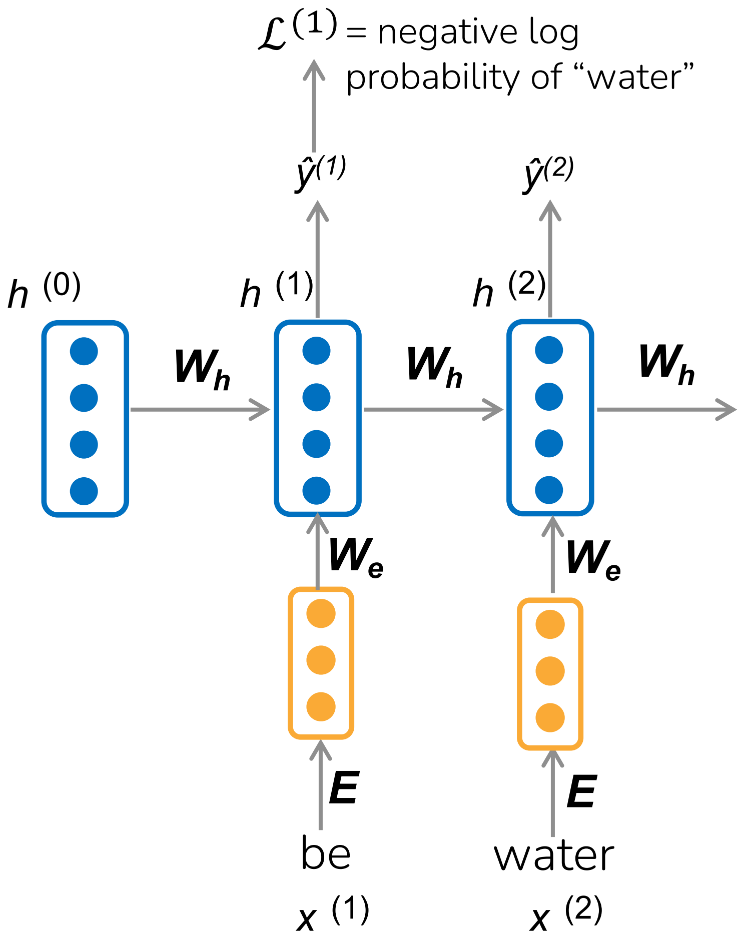


Training RNNs

- Get some training data and chunk it into batches (usually sentences)
- Feed into RNN and for every step t compute the output distribution $\widehat{y^{(t)}}$
- Compute the loss at step t: *Cross Entropy* between output distribution and the actual next word, which is the one-hot vector for $x^{(t+1)}$

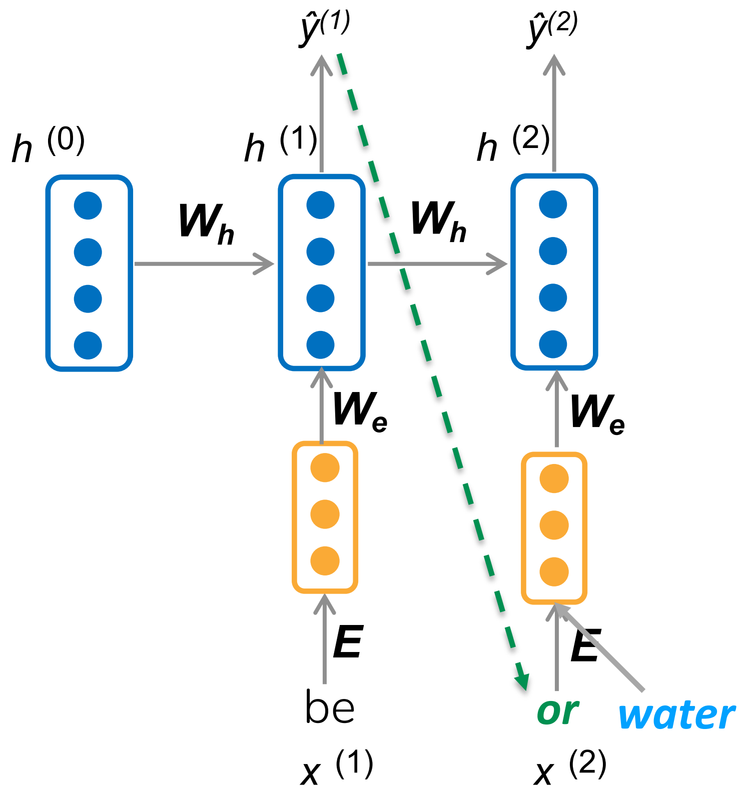
$$\mathcal{L}^{(\mathrm{t})}\,(\theta) = -\sum_{\mathrm{w}\in\mathrm{V}} y_w^{(t)} \log \widehat{y_w^{(t)}} = -\log \widehat{y_{x_{t+1}}^{(t)}}$$

- Overall loss is the average over all steps



Inference & Teacher Forcing

- Each output distribution can generate a word by taking the most probable word in the distribution or at random based on distribution
- During test time / inference, the generated word in the output sequence is used as the new input for generating the next word (***autoregressive generation***)
- During training we have a choice:
  - (A) Use the generated word
  - (B) Use the actual ground truth word (***teacher forcing***)



Issue: Vanishing & Exploding Gradients

- Backpropagation during training is done over timesteps and summing the gradients (***backprogation through time***) – details omitted
- Since the previous hidden states influence the new hidden state all previous hidden states appear in the gradient (***chain rule***)
- Simplified: A word that is k times away from the current word has a product of k matrices contributing to the gradient of the current word

→ Small (< 1) or large (>1) values propagate through in the gradient calculation

→ Gradient gets too large: adjustment steps are too large, bad learning and parameters (easy to solve with ***gradient clipping***)

→ Gradient gets too small: Model is unable to predict **long-distance dependencies** between words

Vanishing Gradient Example

- “When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her ________
- The word tickets is very far away from the target word in the end
- If the gradient is too small, the model cannot learn this dependency

Can we fix this?

- RNN vanilla architecture makes it too difficult to preserve information over many steps since the hidden state is constantly being rewritten

→ We need some kind of separate ***memory*** (LSTMs)

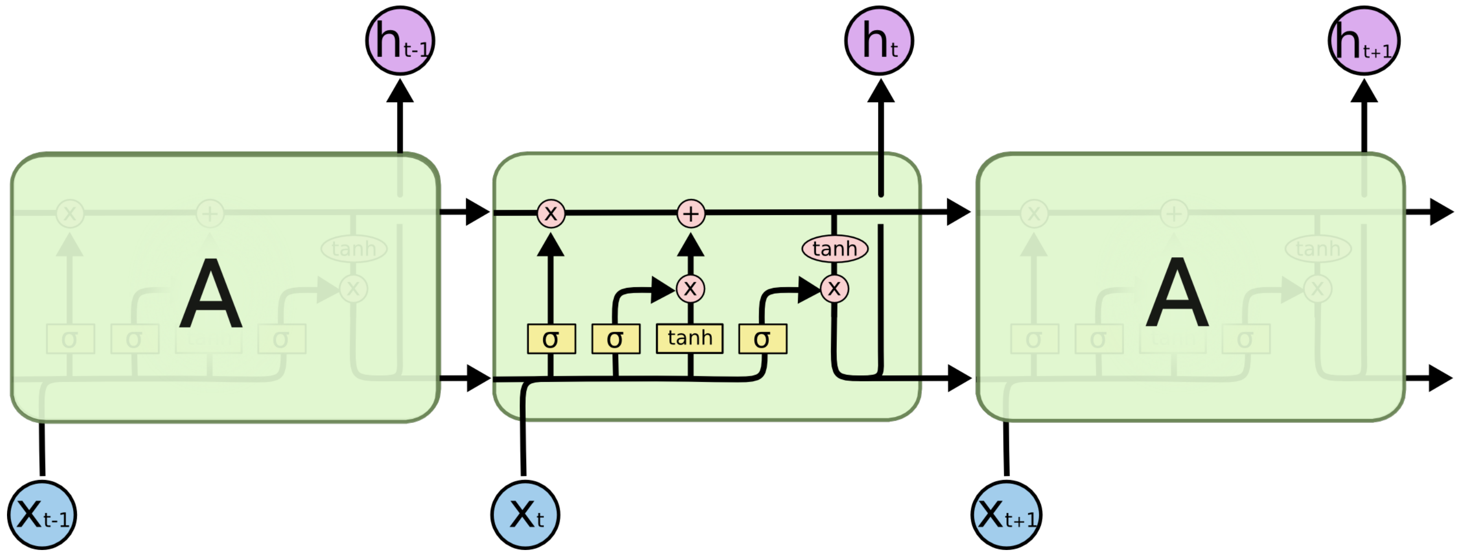
→ Or a direct passthrough where all words refer to all other words (👀 ***attention***)

LSTMs

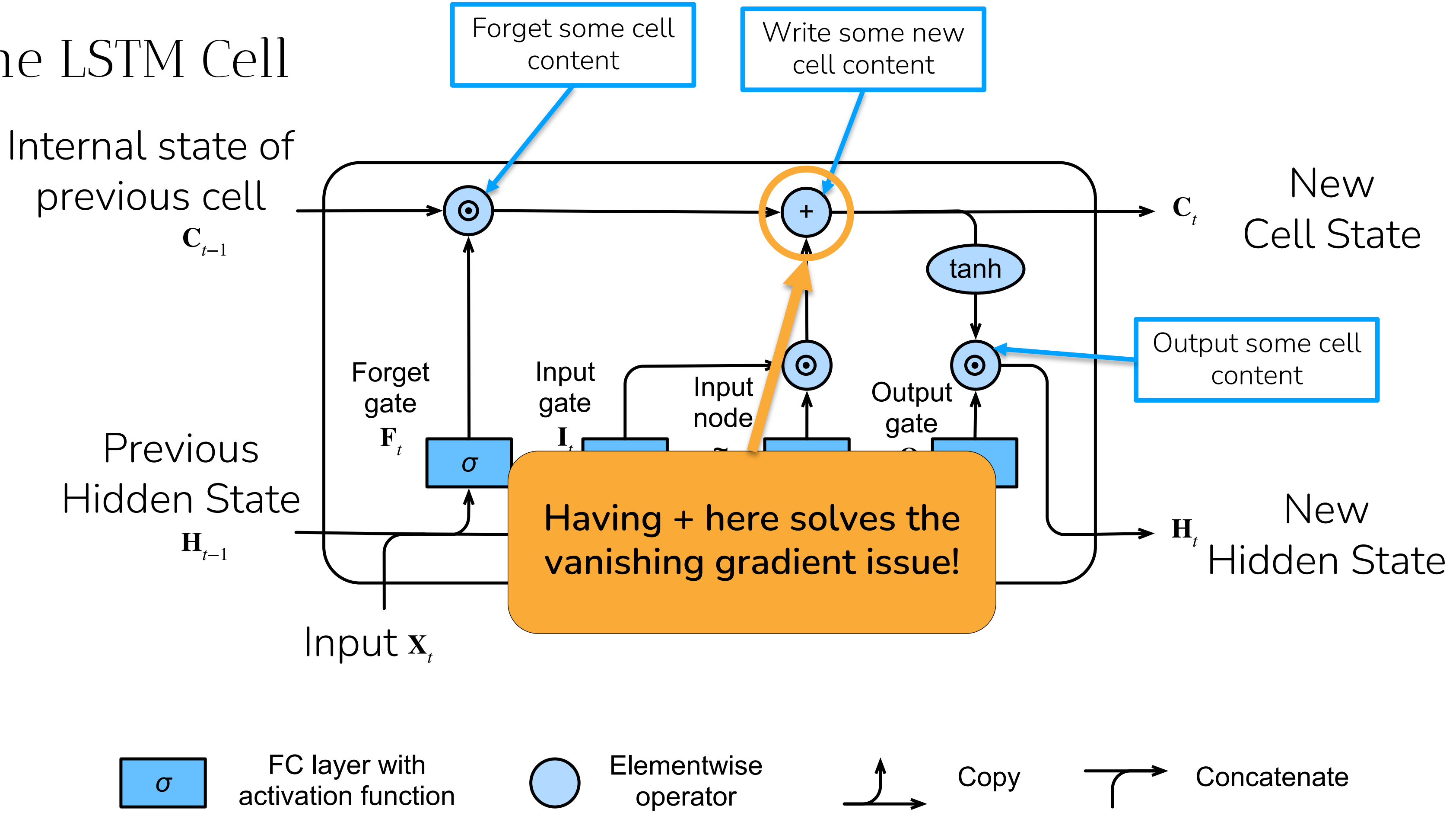
LSTMs

- Introduced by Hochreiter and Schmidhuber, 1997 and Gers, Schmidhuber, and Cummins, 2000
- Became popular in 2013 when Geoffrey Hinton joined Google
- For each step t there is a ***hidden state*** $\mathrm{h}^{(\mathrm{t})}$ and a ***cell state*** $\mathrm{c}^{(\mathrm{t})}$ which are vectors of the same dimension n
- The cell's purpose is to store long-term information
- The LSTM learns to ***read, erase, and write*** information from the cell
- These operations are controlled by ***gates***, vectors of the same dimension n with values between 0 (closed gate) and 1 (open gate)
- These gates are multiplied element-wise with the states (***Hadamard product***)

LSTMs



One LSTM Cell


Forget some cell content
Write some new cell content
Internal state of previous cell $\mathbf{C}_{t-1}$
New Cell State $\mathbf{C}_t$
tanh
Output some cell content
Forget gate $\mathbf{F}_t$
Input gate $\mathbf{I}_t$
Input node
Output gate
σ
Previous Hidden State $\mathbf{H}_{t-1}$
New Hidden State $\mathbf{H}_t$
Input $\mathbf{X}_t$
+
FC layer with activation function
Elementwise operator
Copy
Concatenate

Having + here solves the vanishing gradient issue!

LSTMs Success

- LSTMs mitigate the vanishing gradient issue of RNNs, memory can be preserved
- Allows us to process much longer sequences
- From 2013 up until the ***transformer architecture*** in 2017, LSTMs were the dominant NLP approach and achieved state-of-the-art results (e.g. machine translation, image captioning, language models)
- 2016 Google switched to LSTMs for machine translation

RNNs & LSTMs Issues

- **Issue 1:** The meaning of the entire sequence – which we need for seq2seq in a bit - is only capsulated in the ***last hidden state vector***

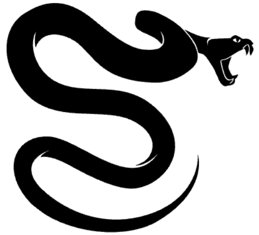
→ Information might get lost and early information fades

- **Issue 2:** Computation is still sequential and cannot be parallelized

→ Slow and inefficient training

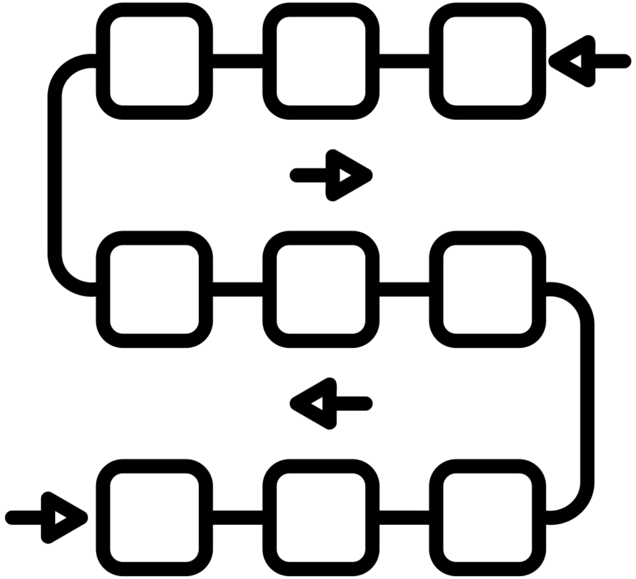
- **Issue 3: Pretraining is inefficient and less effective**
  By design, LSTMs can only be trained for next or previous token prediction (ELMo, 2018), ***masked language modelling*** (next lecture) is more efficient for building language models

MAMBA (2023)
[Anyone up for a master thesis?]



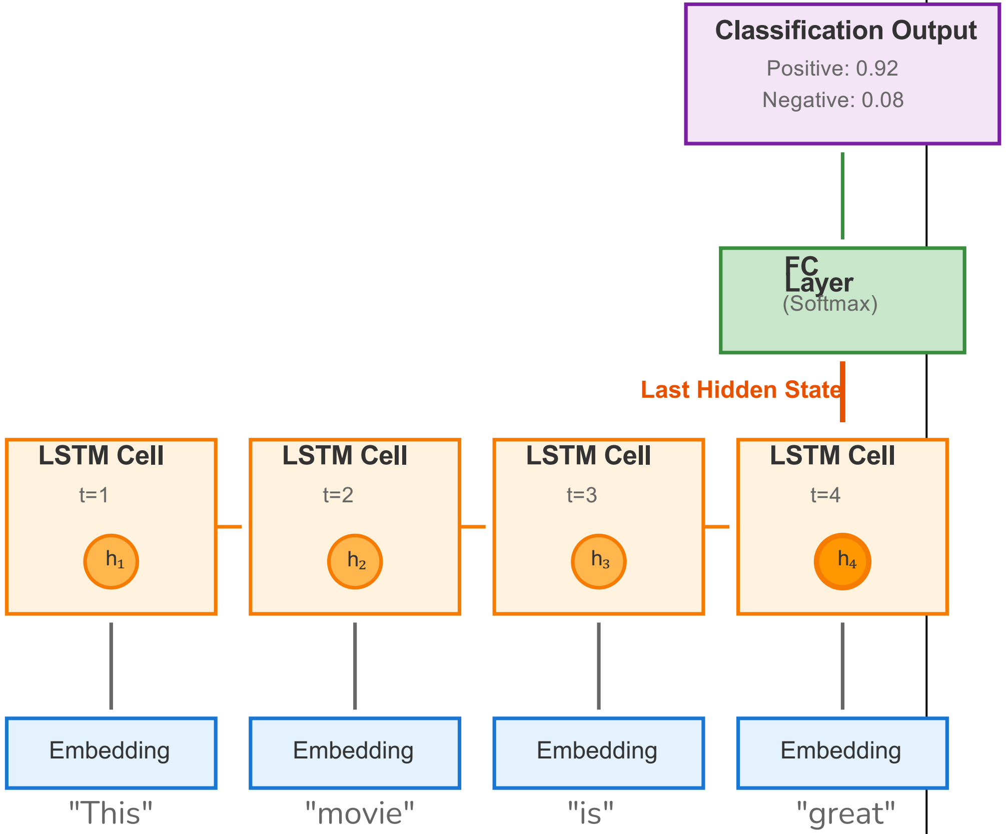
Transformers for the win!

seq2seq



Solving Tasks with RNNs/LSTMs

- So, we can now process input sequences of any length n 🎉
- To ***classify*** sequences: Add a fully connected layer to the last hidden state
- To ***classify each token*** (e.g. POS tagging) add a fully connected layer to each hidden state
- But how can we model an n to m mapping, i.e. generate sequences?



Typical Seq2Seq Tasks

Summarization

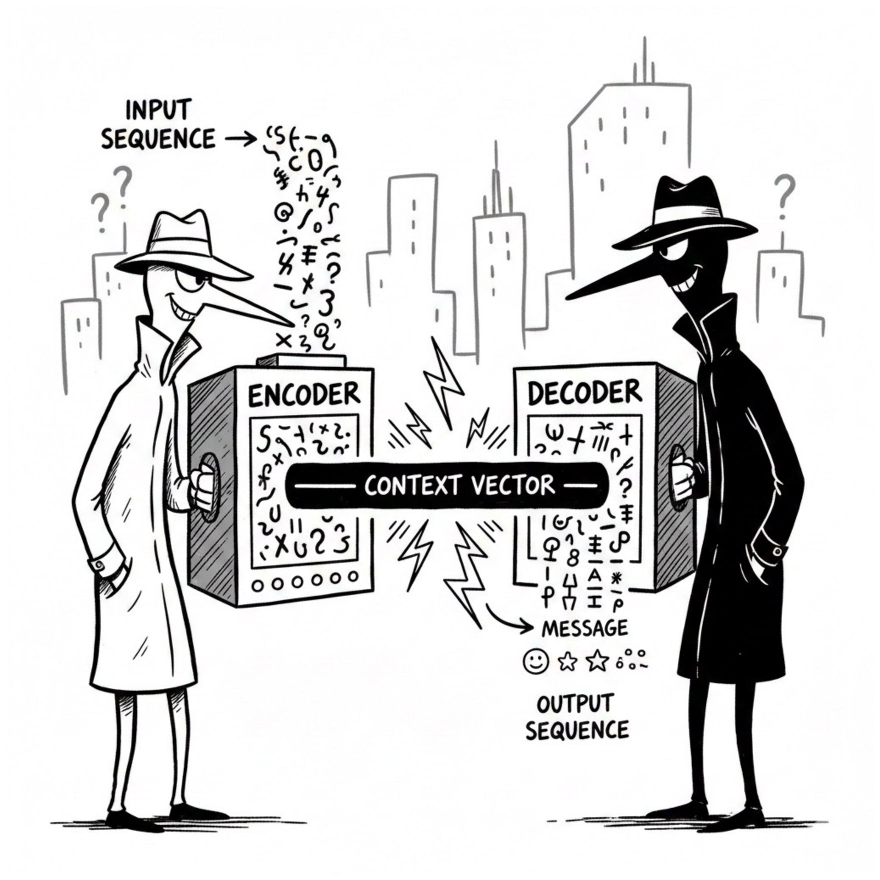
Machine Translation

Text Generation

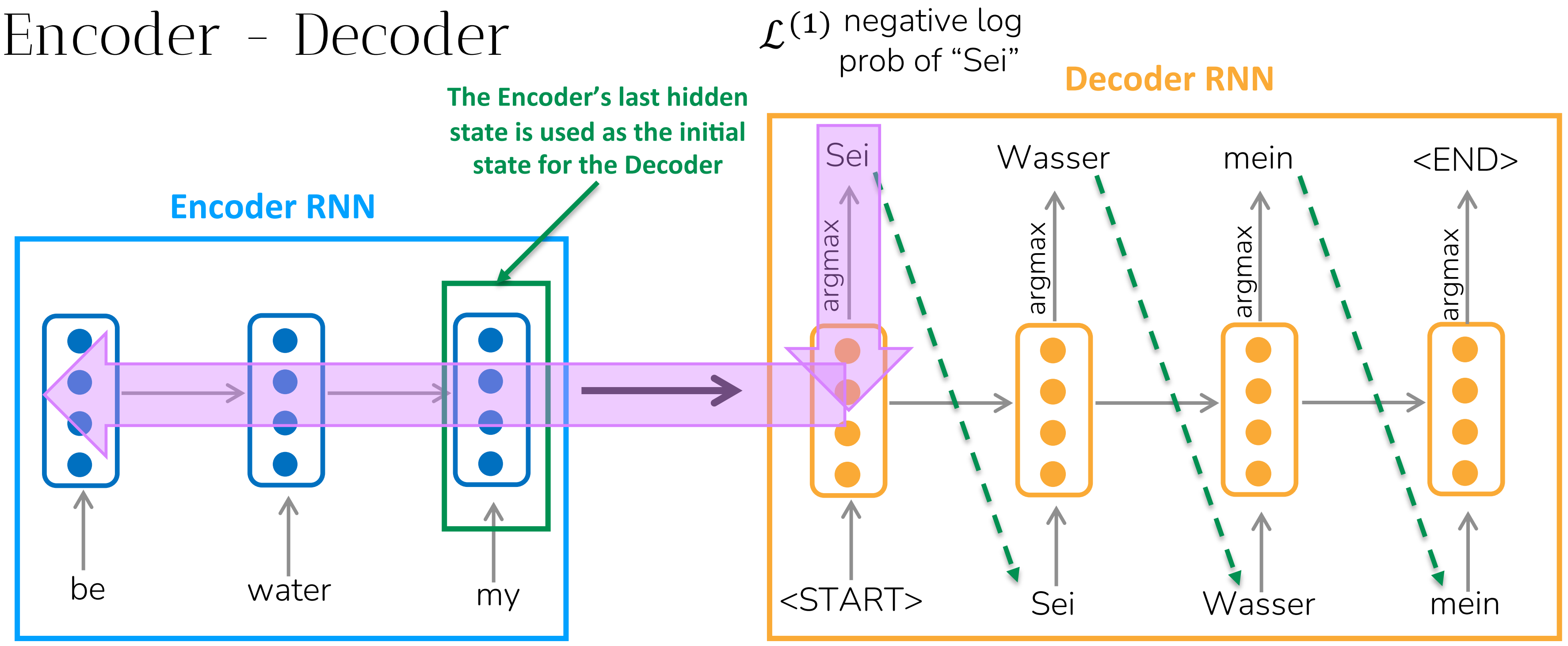
Image Captioning

Simplification

Encoder - Decoder



Encoder - Decoder


$\mathcal{L}^{(1)}$ negative log prob of "Sei"
Decoder RNN
The Encoder's last hidden state is used as the initial state for the Decoder
Encoder RNN
Sei
Wasser
mein
<END>
argmax
argmax
argmax
argmax
be
water
my
<START>
Sei
Wasser
mein

Generates encoding of the source sentence

Based on encoding, generates target sentence.

Training is done as a single system

During inference, the output is used as the next step's input

RNNs & LSTMs Issues - Reprise

- The entire meaning of the source sentence is encapsulated in the last hidden state
- The decoder only uses this last hidden state but has not other interaction with the encoder
- Words are processed in sequential order and thus it is difficult to learn long-distance dependencies
- We have to compute the hidden states sequentially so we cannot parallize and use GPUs for training and inference

Attention Is All You Need

Tutorial