HSLU Lucerne University of Applied Sciences and Arts

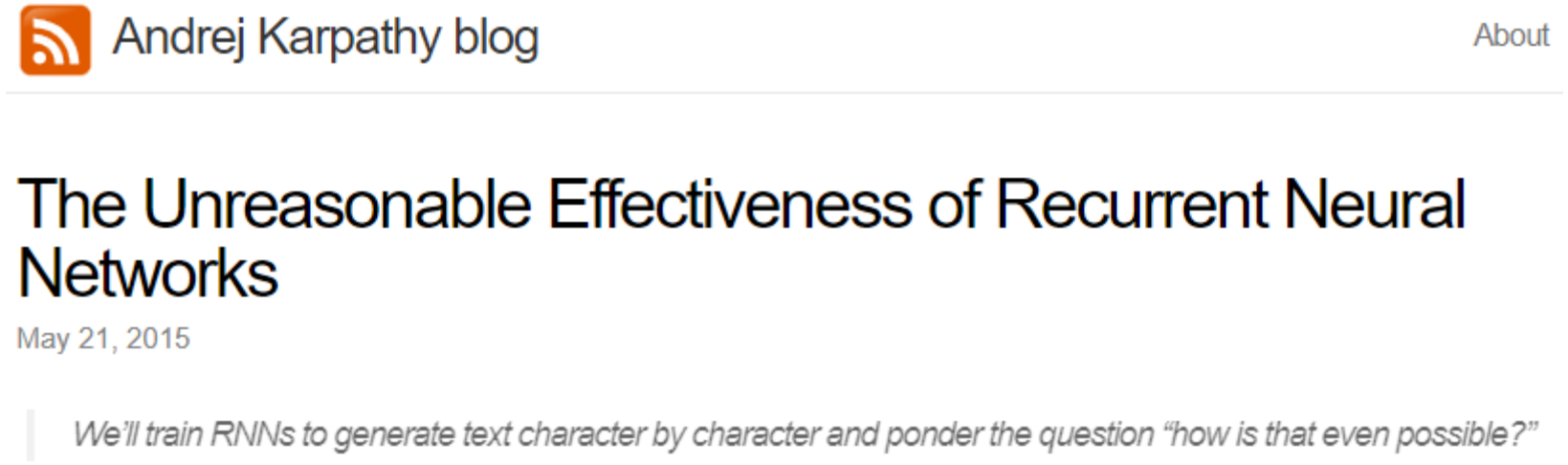
Recurrent Neural Networks

NLP
Andreas Marfurt

Information Technology
06.03.2025

FH Zentralschweiz

Motivation



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

Motivation

Shakespeare

It looks like we can learn to spell English words. But how about if there is more structure and style in the data? To examine this I downloaded all the works of Shakespeare and concatenated them into a single (4.4MB) file. We can now afford to train a larger network, in this case lets try a 3-layer RNN with 512 hidden nodes on each layer. After we train the network for a few hours we obtain samples such as:

```
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.
```


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

Motivation

Algebraic Geometry (Latex)

Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*

Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

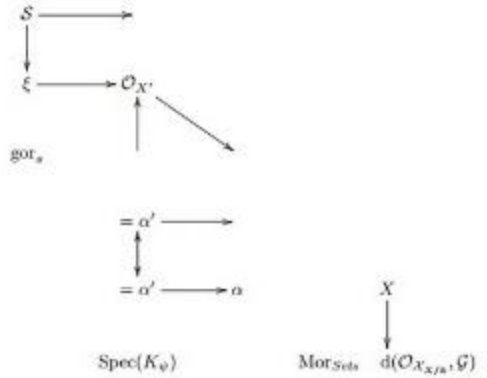
be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram



is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type f_*. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

□

Proof. We have see that $X = \mathrm{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. □

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \quad \text{-1}(\mathcal{O}_{X_{étale}}) \longrightarrow \mathcal{O}_{X_i}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\zeta}^{\overline{w}})$$

is an isomorphism of covering of $\mathcal{O}_{X_i}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over S.
If $\mathcal{F}$ is a scheme theoretic image points. □

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma ??. This is a sequence of $\mathcal{F}$ is a similar morphism.

More hallucinated algebraic geometry. Nice try on the diagram (right).

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

Motivation

Linux Source Code

I wanted to push structured data to its limit, so for the final challenge I decided to use code. In particular, I took all the source and header files found in the Linux repo on Github, concatenated all of them in a single giant file (474MB of C code) (I was originally going to train only on the kernel but that by itself is only ~16MB). Then I trained several as-large-as-fits-on-my-GPU 3-layer LSTMs over a period of a few days. These models have about 10 million parameters, which is still on the lower end for RNN models. The results are superfun:

```
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
```

Motivation

Generating Baby Names

Lets try one more for fun. Lets feed the RNN a large text file that contains 8000 baby names listed out, one per line (names obtained from here). We can feed this to the RNN and then generate new names! Here are some example names, only showing the ones that do not occur in the training data (90% don't):

Rudi Levette Berice Lussa Hany Mareanne Chrestina Carissy Marylen Hammine Janye Marlise Jacacrie Hendred Romand Charienna Nenotto Ette Dorane Wallen Marly Darine Salina Elvyn Ersia Maralena Minoria Ellia Charmin Antley Nerille Chelon Walmor Evena Jeryly Stachon Charisa Allisa Anatha Cathanie Geetra Alexie Jerin Cassen Herbett Cossie Velen Daurenge Robester Shermond Terisa Licia Roselen Ferine Jayn Lusine Charyanne Sales Sanny Resa Wallon Martine Merus Jelen Candica Wallin Tel Rachene Tarine Ozila Ketia Shanne Arnande Karella Roselina Alessia Chasty Deland Berther Geamar Jackein Mellisand Sagdy Nenc Lessie Rasemy Guen Gavi Milea Anneda Margoris Janin Rodelin Zeanna Elyne Janah Ferzina Susta Pey Castina

You can see many more here. Some of my favorites include "Baby" (haha), "Killie", "Char", "R", "More", "Mars", "Hi", "Saddie", "With" and "Ahbort". Well that was fun. Of course, you can imagine this being quite useful inspiration when writing a novel, or naming a new startup :)

Still relevant in the Transformers era?

RWKV Language Model

RWKV (pronounced as RwaKuv) is an RNN with GPT-level LLM performance, which can also be directly trained like a GPT transformer (parallelizable).

So it's combining the best of RNN and transformer - great performance, fast inference, fast training, saves VRAM, "infinite" ctxlen, and free sentence embedding. Moreover it's 100% attention-free.

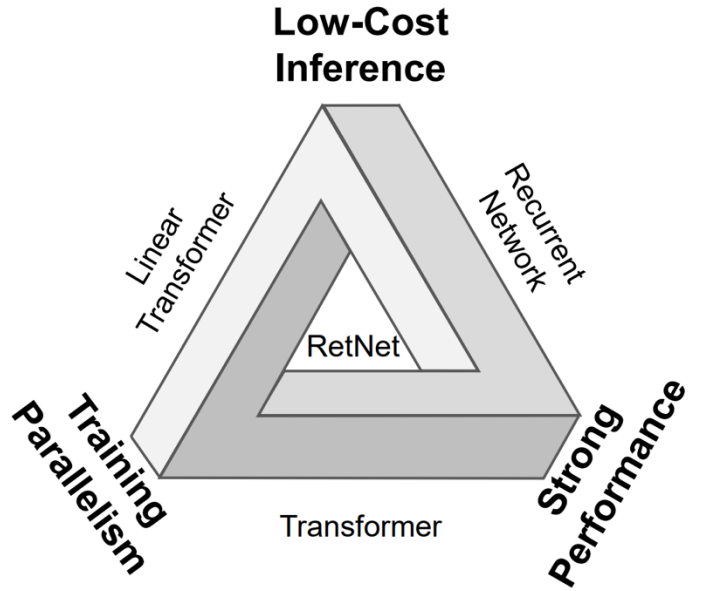


Figure 2: RetNet makes the "impossible triangle" possible, which achieves training parallelism, good performance, and low inference cost simultaneously.

https://wiki.rwkv.com/ & RetNet paper

Overview

- Language Modeling
- N-gram Language Models
- Recurrent Neural Networks
- Long Short-Term Memory (LSTM)
- Gated Recurrent Unit (GRU)

Language Modeling & N-gram Language Models

Language Modeling

- Task: Predict which word comes next.
- The students opened their _____
  - books
  - laptops
  - exams
  - minds

- A system that does this is called a *language model.*

Why Language Modeling?

- Language Modeling is a benchmark task that helps us measure our progress on understanding language
- Language Modeling is a subcomponent of many NLP tasks, especially those involving generating text or estimating the probability of text:
  - Spelling/grammar correction
  - Machine translation
  - Dialogue
  - Handwriting recognition
  - Speech recognition
  - etc.

Language Modeling

- Predict the next word w_i from the past words $w_{<i}$

$$w_{<i} = (w_1, \ldots, w_{i-1})$$

- Determine the probability of a sequence of words w

$$w = (w_1, \ldots, w_n)$$

- n is the total number of words

Language modeling

$$p(w) = p(w_1) * p(w_2|w_1) * p(w_3|w_1, w_2) * \cdots * p(w_n|w_{<n})$$

Language model

Language Modeling

$$p(w) = \prod_{i=1}^{n} p(w_i | w_{<i})$$

Question: What happens if we don’t look at previous words?

Question: What happens if we don’t look at previous words?

- The best you can do is just to always predict the most frequent word
- You can do a little better by predicting the most frequent word based on the position in the sentence…

Markov Chains/Processes

- Markov chain/process: probability of each state depends only on the previous state
- Markov chain of order *m*: … depends on the previous *m* states

- Rephrased for language modeling: The probability of each word depends only on the previous *m* words.

N-Gram Language Model

- Predict the next word from the previous *(m-1)* words

$$p(w) = \prod_{i=1}^{n} p\left(w_i \middle| w_{i-(m-1),\ldots,i-1}\right)$$

Bigram Language Model

- $m = 2$ (“bigram”)

$$p(w_2|w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)}$$

- Example:

  p(apple|the) = count(“the apple”) / count(“the”)
  p(York|New) = count(“New York”) / count(“New”)

Trigram Language Model

- m = 3 (“trigram”)

$$p(w_3|w_1, w_2) = \frac{\text{count}(w_1, w_2, w_3)}{\text{count}(w_1, w_2)}$$

- Example:

  p(is|“the apple”) = count(“the apple is”) / count(“the apple”)

Any-Gram Language Model?

- We can go back an arbitrary number of *m* words
- But: We need to store all *m*-grams and (*m-1*)-grams we have seen in the training corpus
  - Combinatorial explosion of memory requirement

- If you're interested: Liu et al., 2024 (∞-gram LM)

Sparsity Problems with N-Gram LMs

$$p(w_3|w_1, w_2) = \frac{\text{count}(w_1, w_2, w_3)}{\text{count}(w_1, w_2)}$$

- Unseen n-grams: If w_1 and w_2 never appear together, both the numerator and the denominator are 0

  → the probability of w_3 cannot be computed (or is 0, which is also not helpful)

Sparsity Problems with N-Gram LMs

- Solution 1: Smoothing
  - Additive/Laplace/"add-one" smoothing

$$p(w_3|w_1, w_2) = \frac{\text{count}(w_1, w_2, w_3) + \delta}{\text{count}(w_1, w_2) + \delta N}$$

  - δ can be chosen ($\delta = 1$ for "add-one" smoothing)
  - *N* is the number of all words in the vocabulary, but can also be chosen differently

https://en.wikipedia.org/wiki/Additive_smoothing

Sparsity Problems with N-Gram LMs

- Solution 2: Backoff
  - If $\text{count}(w_1, w_2)$ is 0, back off to a (n-1)-gram language model

$$p(w_3|w_1, w_2) = p(w_3|w_2) = \frac{\text{count}(w_2, w_3)}{\text{count}(w_2)}$$

https://en.wikipedia.org/wiki/Additive_smoothing

Exercise: N-Gram LM

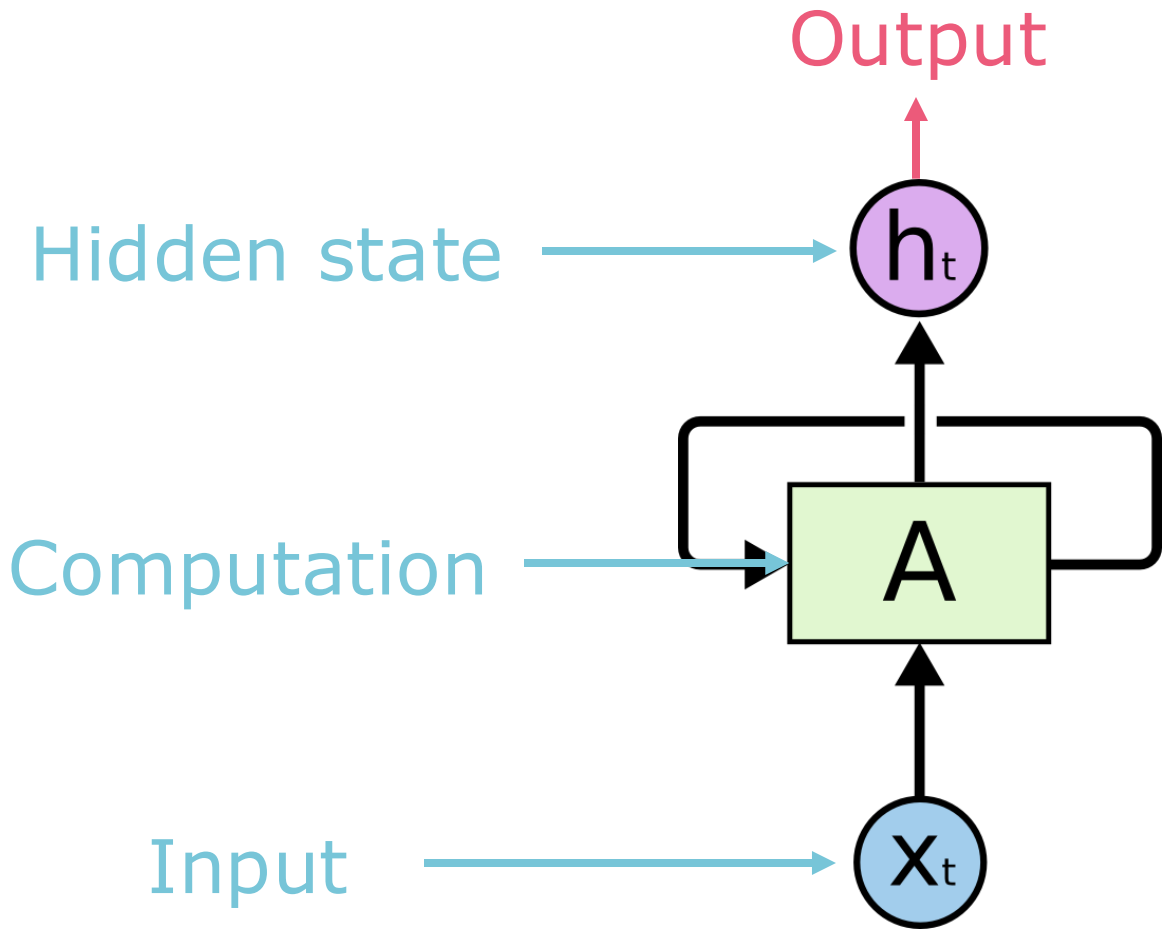
Recurrent Neural Network (RNN)

Recurrent Neural Network

- Based on Rumelhart et al., 1986
  - Learning representations by back-propagating errors
  - David Rumelhart was a psychologist working on sequential thought process

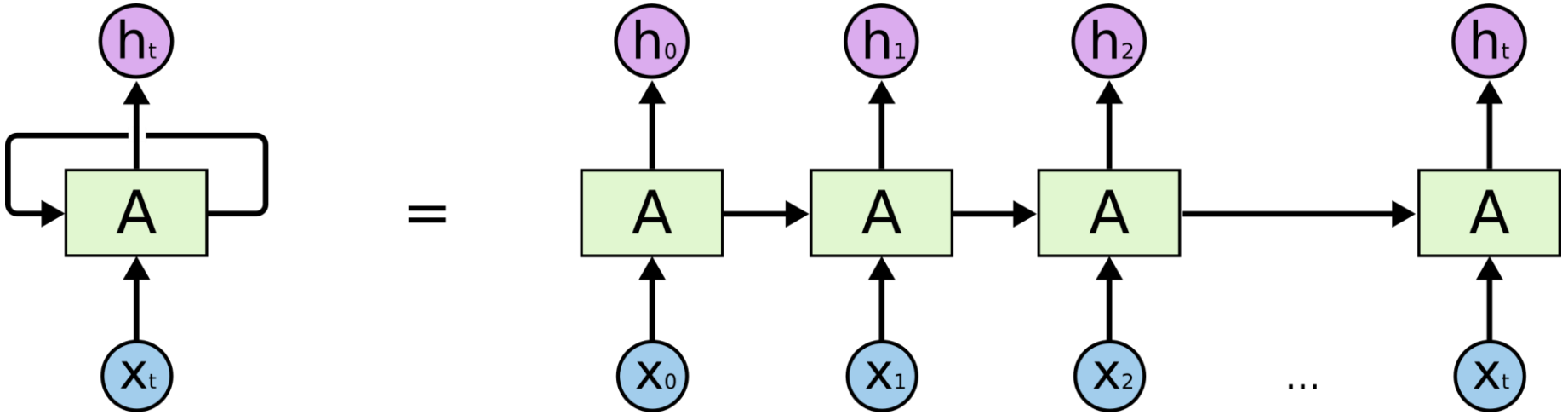
- Think in time steps *t*
  - At each time step, a new input is received, and an output is produced

Recurrent Neural Network



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

Loop Unrolling/Unfolding



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

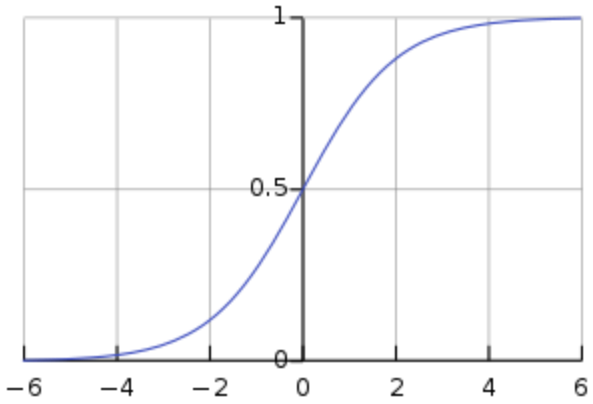
Recurrent Neural Network

Hidden state update:

weight matrices

$$h_t = \sigma\left(W^{(hh)} h_{t-1} + W^{(hx)} x_t\right)$$

Sigmoid function:



Stanford CS224n lecture notes, https://en.wikipedia.org/wiki/Sigmoid_function

Recurrent Neural Network

Output $\hat{y}_t$: A probability distribution over the words in our vocabulary

$$\hat{y}_t = \text{softmax}\left(W^{(s)} h_t\right)$$

Softmax function:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

sum over all j

Cross-entropy

$$H(p, q) = -\sum_{x \in \mathcal{X}} p(x) \log\big(q(x)\big)$$

true distribution

estimated distribution

https://en.wikipedia.org/wiki/Cross_entropy

RNN Loss: Cross-entropy

- Total loss is average over time steps:

$$\mathcal{L} = \frac{1}{T}\sum_{t=1}^{T}\mathcal{L}_t$$

- Loss at time step t:

$$\mathcal{L}_t = -\sum_{i=1}^{|V|} y_{t,i} \times \log\left(\hat{y}_{t,i}\right)$$

0 except for our actual word w_i

Language Modeling Evaluation: Perplexity

- Perplexity is the common metric to evaluate language modeling performance

$$ppl(w) = p(w)^{-\frac{1}{n}}$$

- It measures how surprised the model is when it sees a sample
- Lower is better
- To measure perplexity on a dataset, average the model's perplexity on each data point

Language Modeling Evaluation: Perplexity

- Perplexity is also the exponential of the cross-entropy

$$ppl(w) = 2^{H(p,q)} = 2^{-\sum_i p(w_i) \log_2 q(w_i)}$$

- Can measure perplexity in bits (recommended) or nats
  - Bits: as above
  - Nats: exponentiate with *e* and take the natural logarithm
- Article about the connection between perplexity and cross-entropy

RNN: Computing Gradients

- We differentiate the loss with respect to the weights W

$$\frac{\partial \mathcal{L}}{\partial W} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}_t}{\partial W}$$

$$\frac{\partial \mathcal{L}_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial \mathcal{L}_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

wrt. *all* previous time steps k

Backpropagation through Time (BPTT)

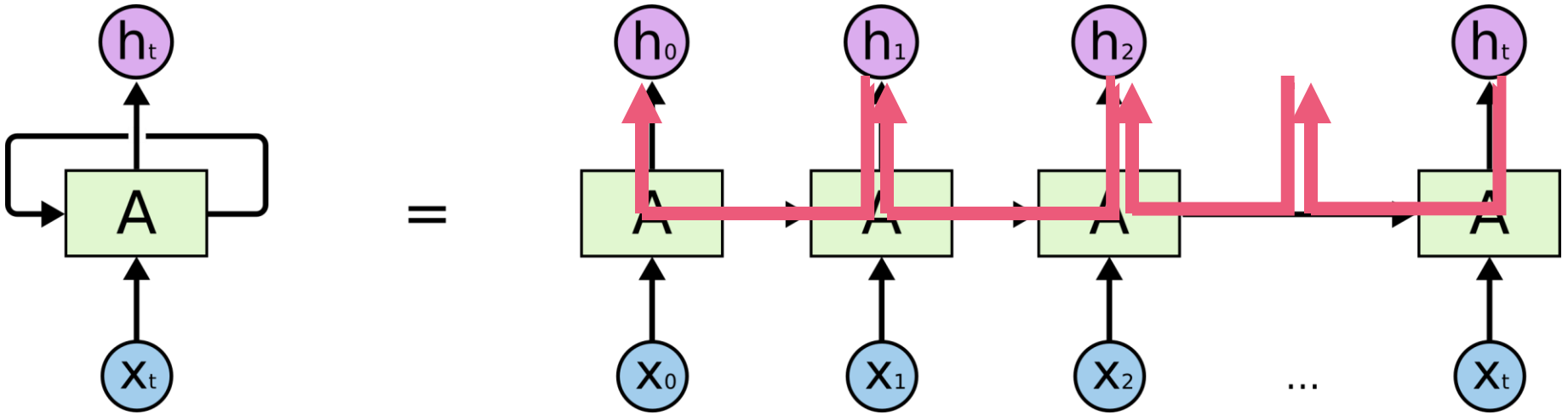
We have to compute the derivative for all previous time steps up to *k*

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}$$

Backpropagation through Time (BPTT)

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}$$

In the unrolled view:

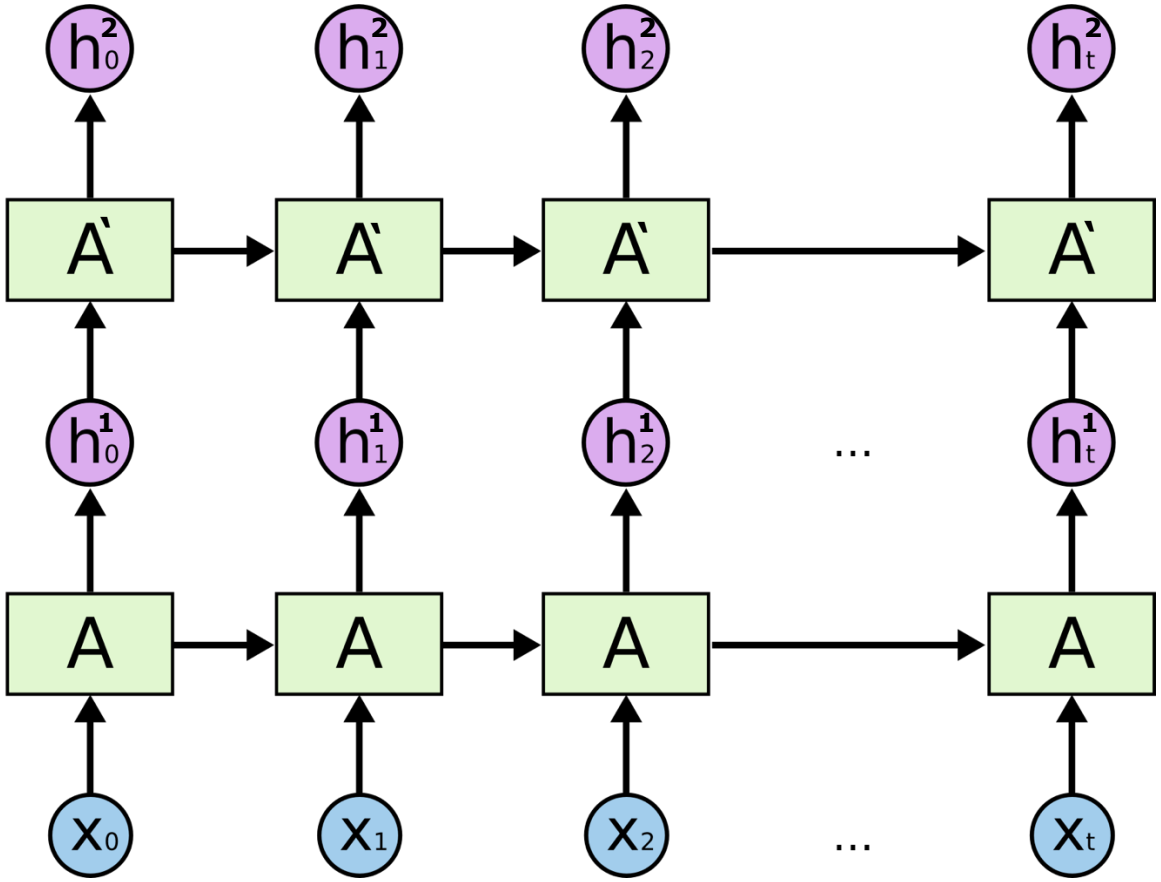


Inputs: Sequences and Time Series

Inputs can be any sequence or time series:

- Characters
- BPE tokens
- Words
- Stock market
- Weather
- ...

Multiple Layers



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

RNN Pros & Cons

Pros

- Can process input sequences of any length
- Model size is fixed: It does not increase with longer inputs
- Can (in theory) use information from many steps back

Cons

- Computation is sequential, can’t be parallelized
  - Need the previous hidden state to compute the next
- Information from many steps back is not usually available in practice

Exercise: RNN

Long Short-Term Memory (LSTM)

Long-range Dependencies

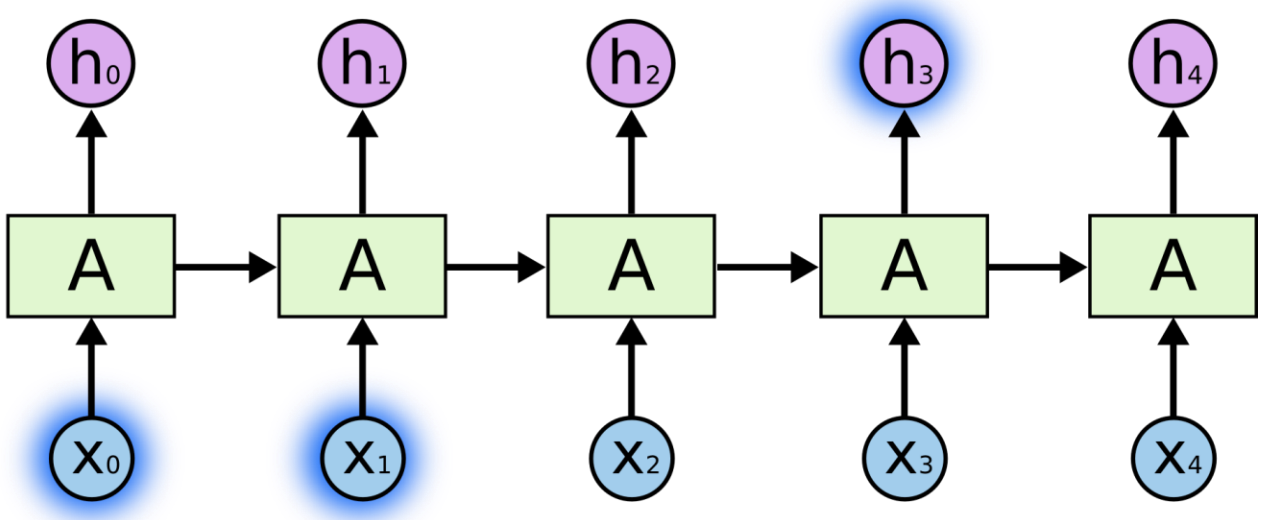
- Compare these two sentences:

Jane walked into the room. John walked in too. Jane said hi to ___

Jane walked into the room. John walked in too. It was late in the day, and everyone was going home after a long day at work. Jane said hi to ___

Long-range Dependencies

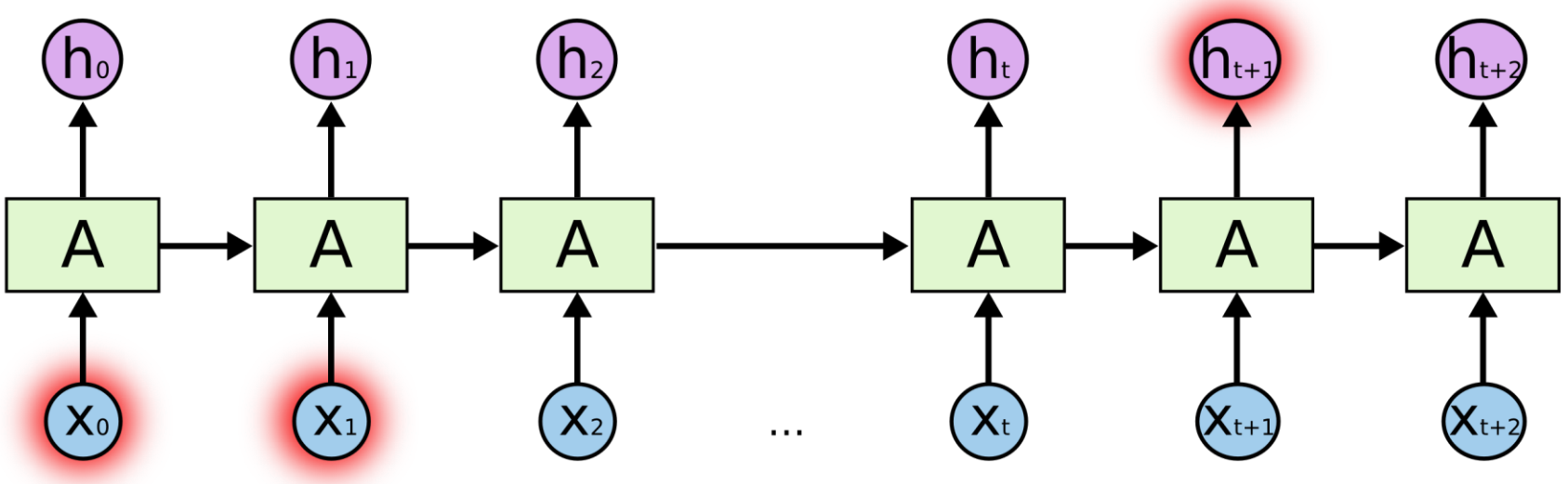
Jane walked into the room. John walked in too. Jane said hi to ___



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

Long-range Dependencies

Jane walked into the room. John walked in too. It was late in the day, and everyone was going home after a long day at work. Jane said hi to ___



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

Long-range Dependencies

- Long-range dependencies are hard to learn for RNNs in practice
- Loss needs to propagate along a path from the time step where the next word prediction was wrong, to the time step where the solution (“John”) was input

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}$$

Question: What happens if we chain a lot of multiplications?

- Multiply x with itself n times

$$x^n$$

- If x > 1: $\lim_{n \to \infty} x = \infty$
- If x < 1: $\lim_{n \to \infty} x = 0$

Vanishing or Exploding Gradients

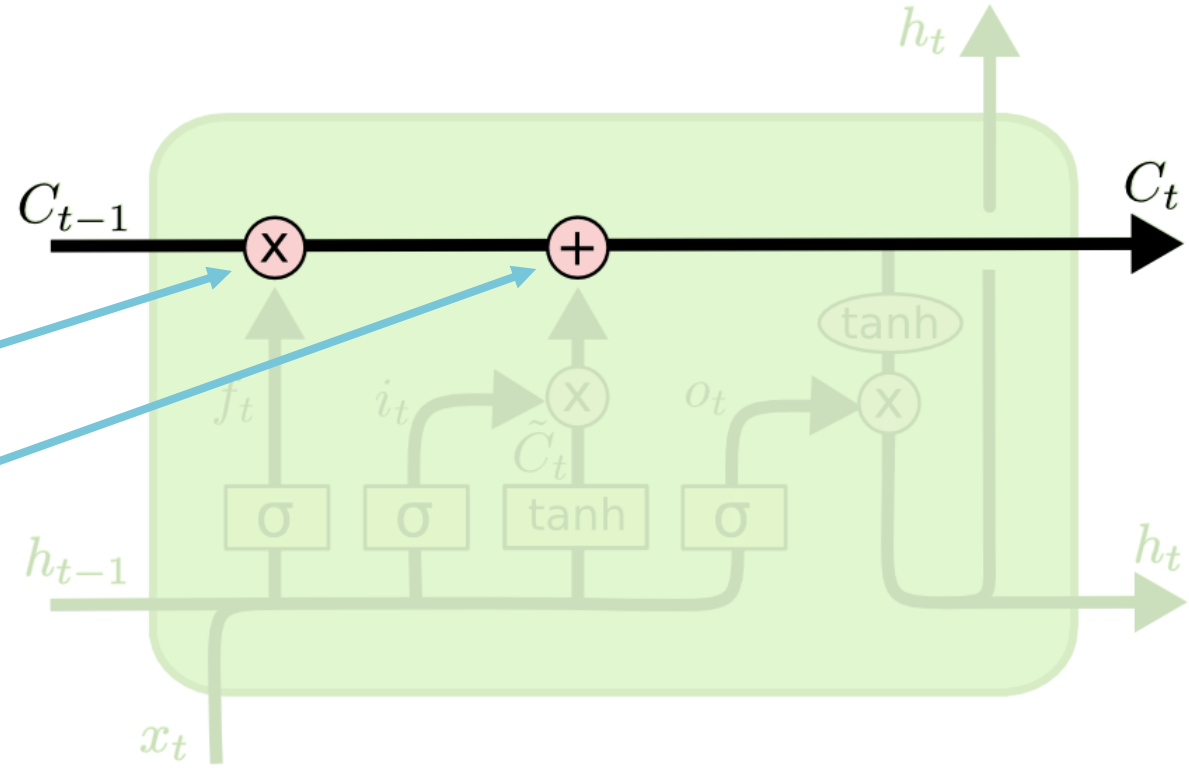
- The same can happen when we multiply partial derivatives
- Vanishing gradients: Parameters aren't updated anymore → No learning
- Exploding gradients: Parameters become infinity → You'll get NaNs in your computation → Have to stop training and restart from an earlier checkpoint
  - Hack to prevent gradients from exploding: *Gradient clipping*
    Clip gradients if they become larger than some predefined value

Long Short-Term Memory

- Hochreiter and Schmidhuber, 1997
- Special version of an RNN designed to
  1) Keep long-range dependencies in memory
  2) Avoid the vanishing/exploding gradients problem

LSTM Cell State

- Cell state *c* stores information across all time steps
  - At each time step, some information is removed…
  - … and some new information is added



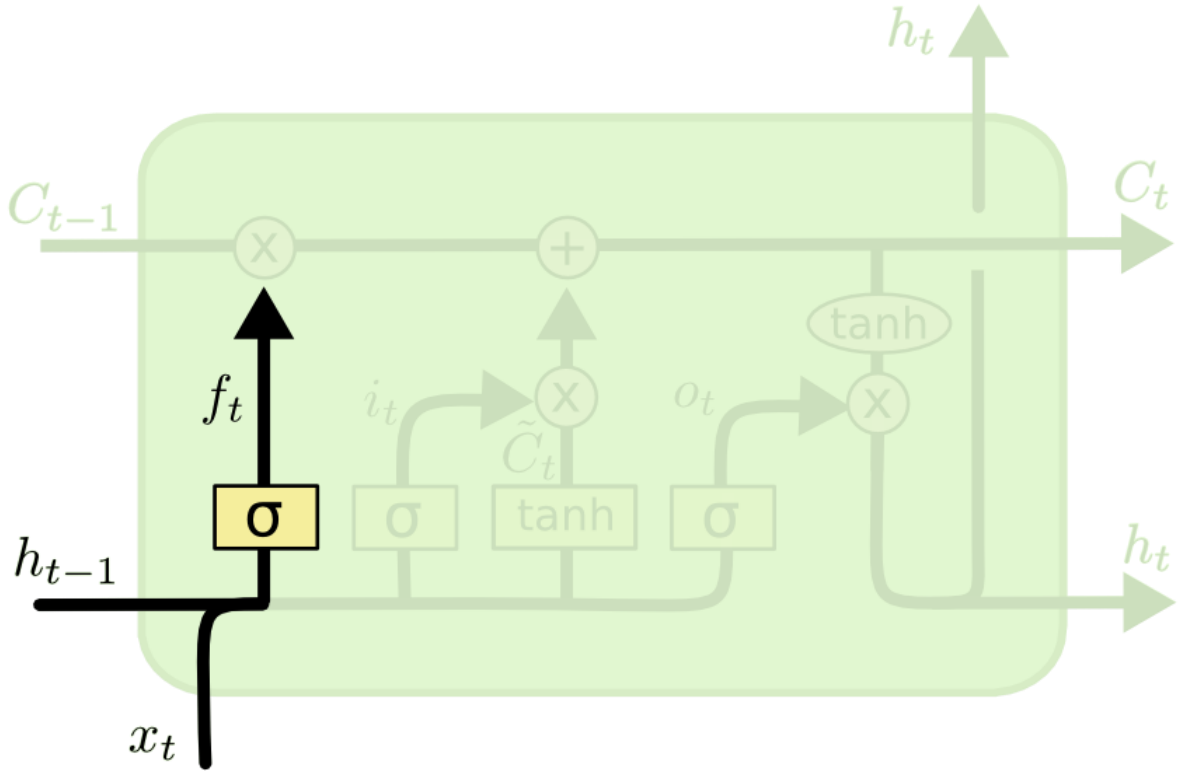
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Architecture

- Forget gate f_t

$$f_t = \sigma\left(W_f h_{t-1} + U_f x_t + b_f\right)$$

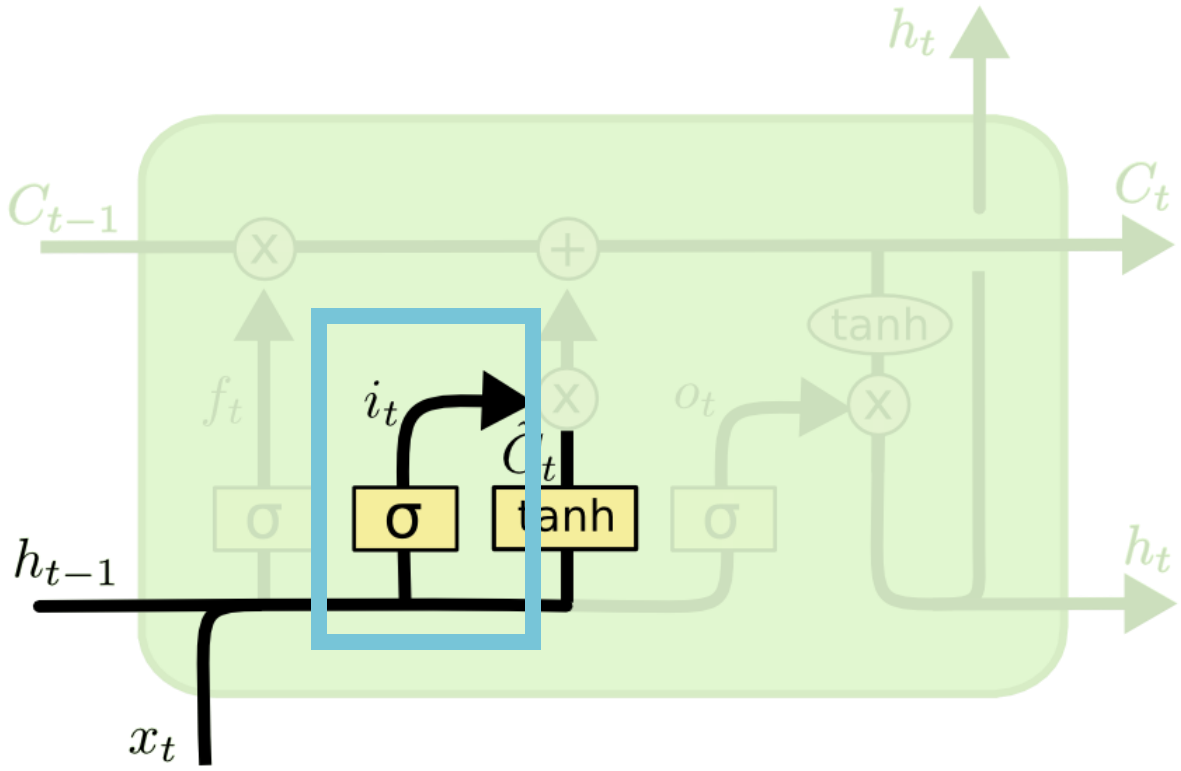
weights bias



LSTM Architecture

- Input gate i_t

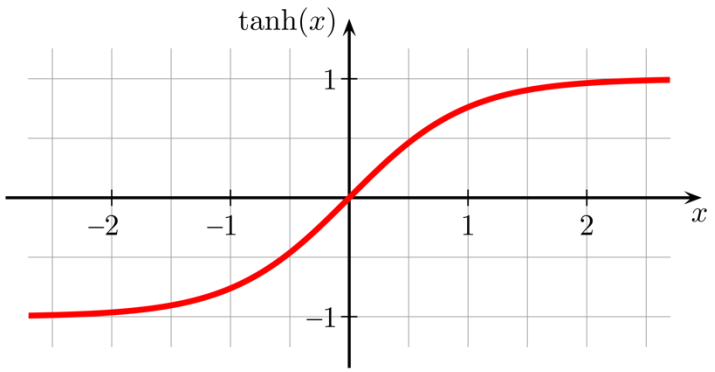
$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$$

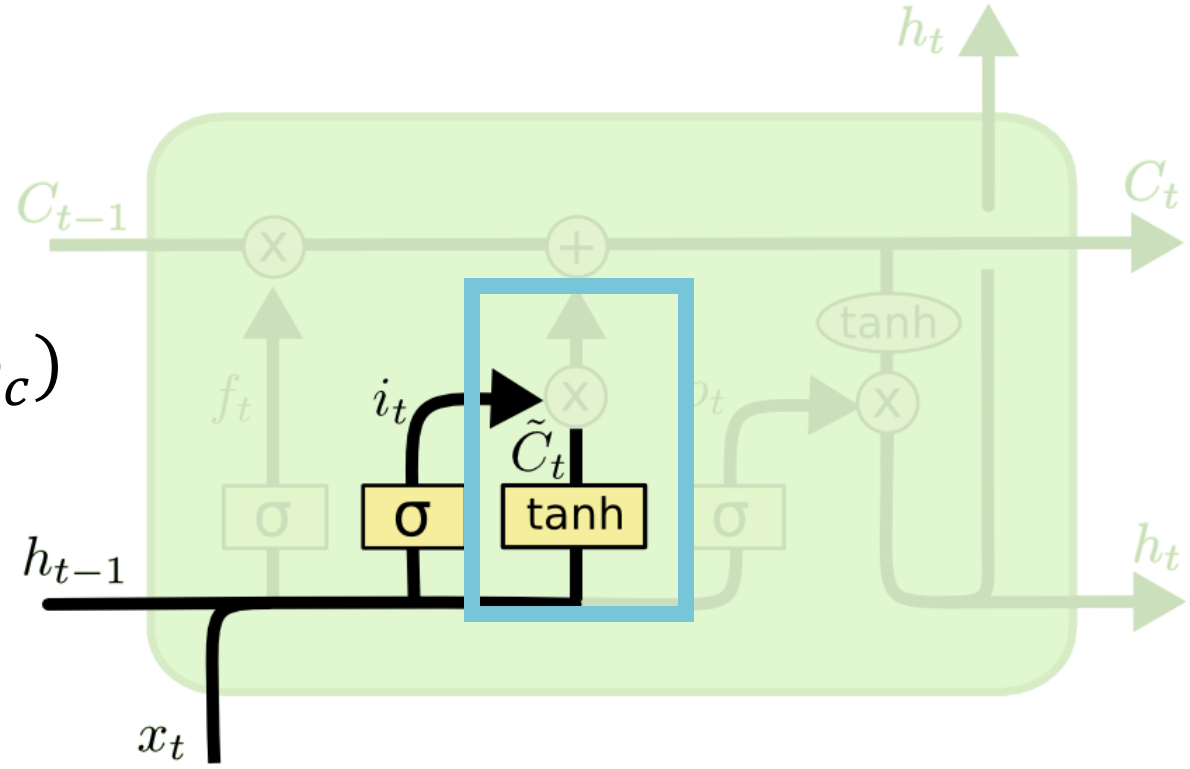


LSTM Architecture

- New cell memory $\tilde{c}_t$

$$\tilde{c}_t = \tanh(W_c h_{t-1} + U_c x_t + b_c)$$





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

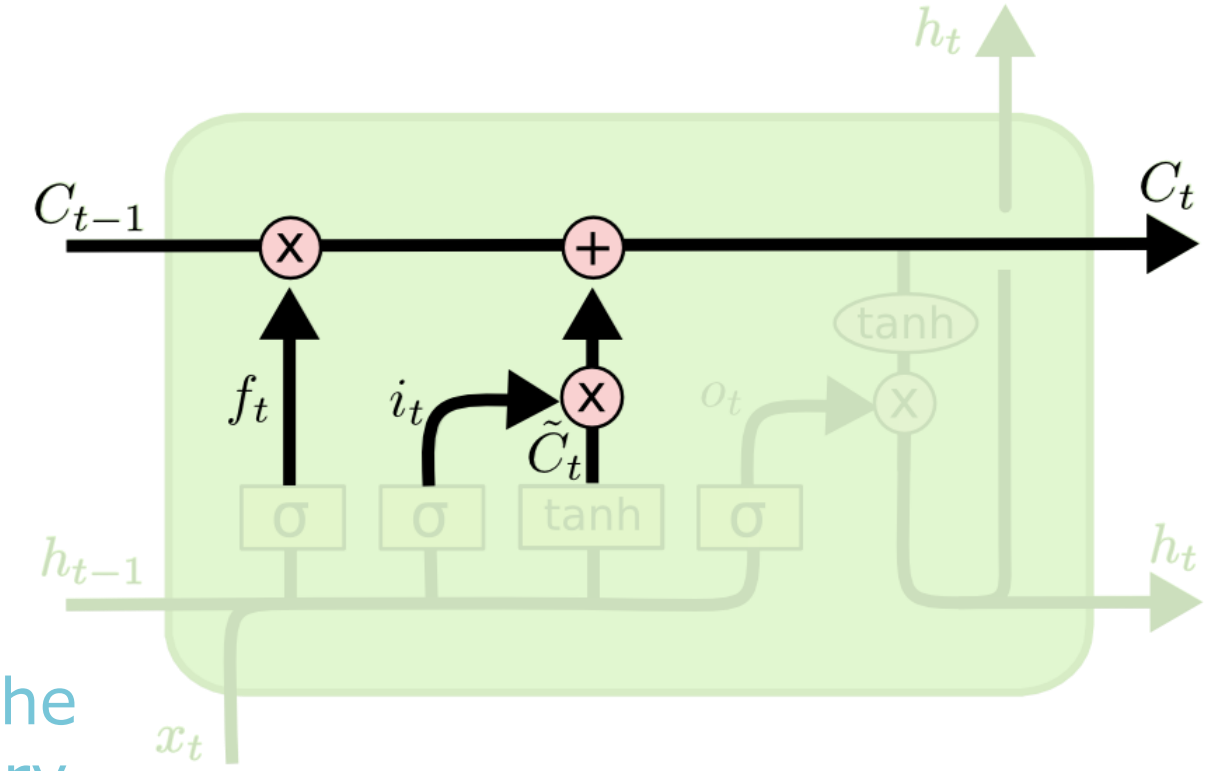
LSTM Architecture

- New cell state c_t

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t$$

What parts of the old cell state do we forget?

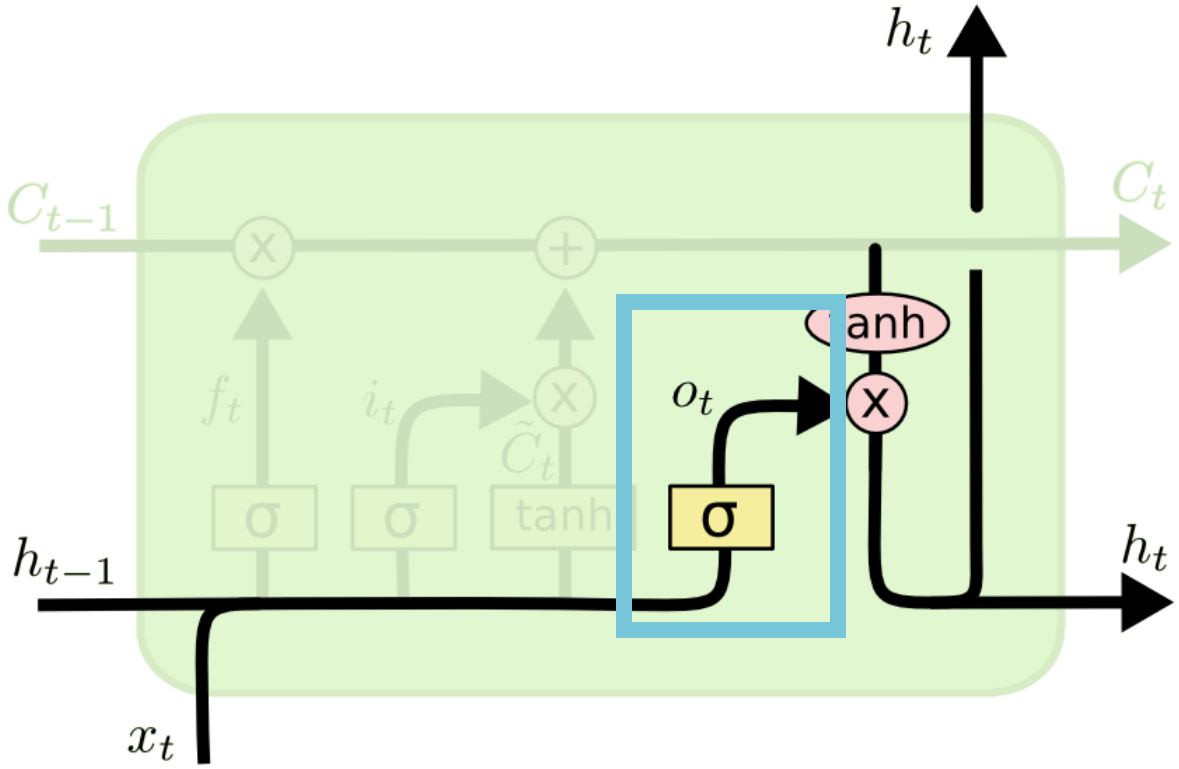
What parts of the new cell memory do we add?



LSTM Architecture

- Output gate o_t

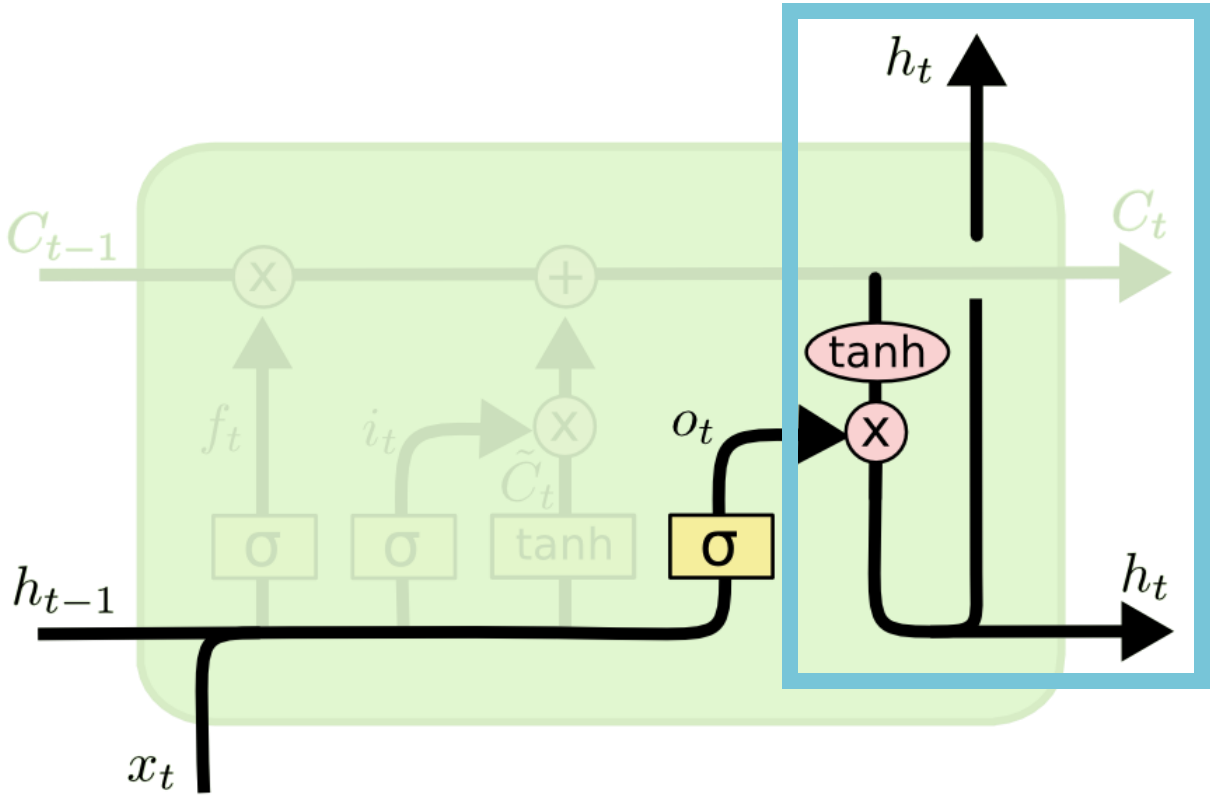
$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$$



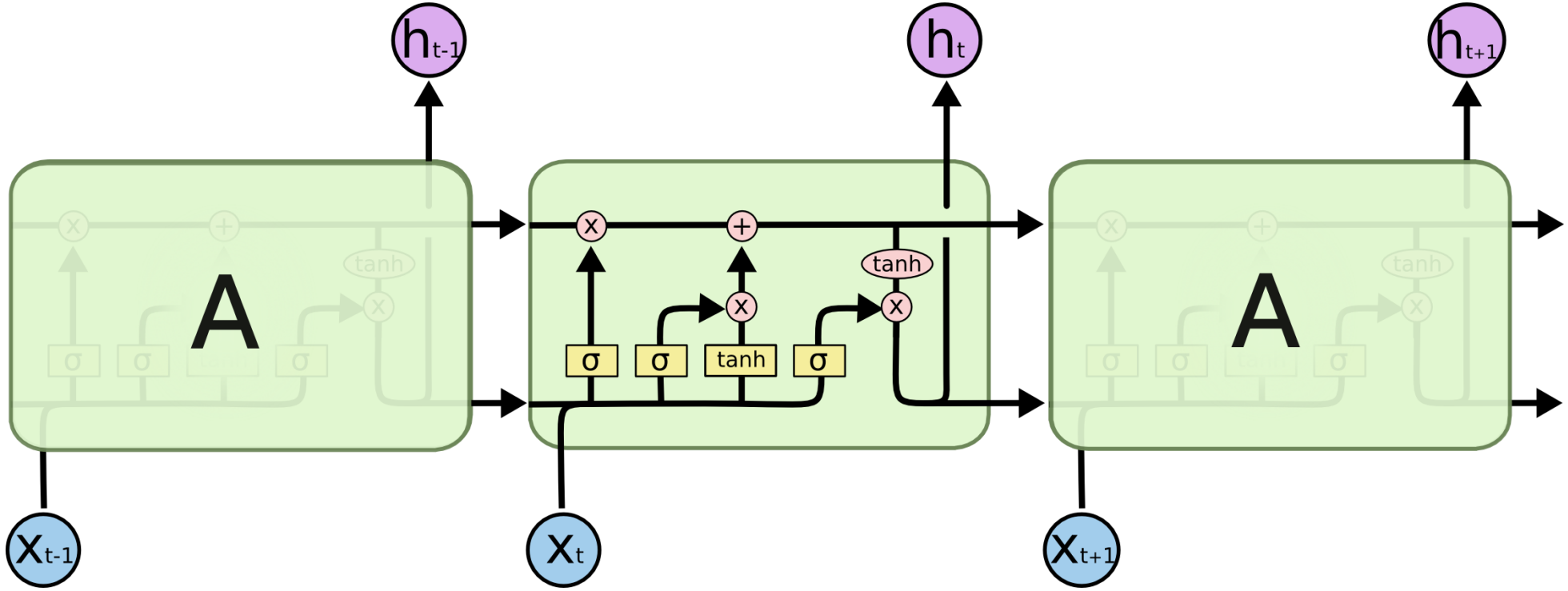
LSTM Architecture

- New hidden state h_t

$$h_t = o_t * \tanh(c_t)$$



LSTM Architecture



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

LSTM Definition

$$f_t = \sigma\left(W_f h_{t-1} + U_f x_t + b_f\right)$$
$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$$
$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$$
$$\tilde{c}_t = \tanh(W_c h_{t-1} + U_c x_t + b_c)$$
$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t$$
$$h_t = o_t * \tanh(c_t)$$

Gates

New cell memory

Updated cell/hidden state

LSTM Review

- Architecture seems extremely involved…
  - How did they come up with this exact architecture?
  - Maybe it is too complicated?
- Many researchers tried to improve upon LSTM's architecture
  - None were successful to consistently improve across tasks and datasets
- LSTMs only became popular with the advent of deep learning (~2014)
  - Larger datasets
  - More compute (GPUs excel at matrix multiplications)

Gated Recurrent Unit (GRU)

GRU Motivation

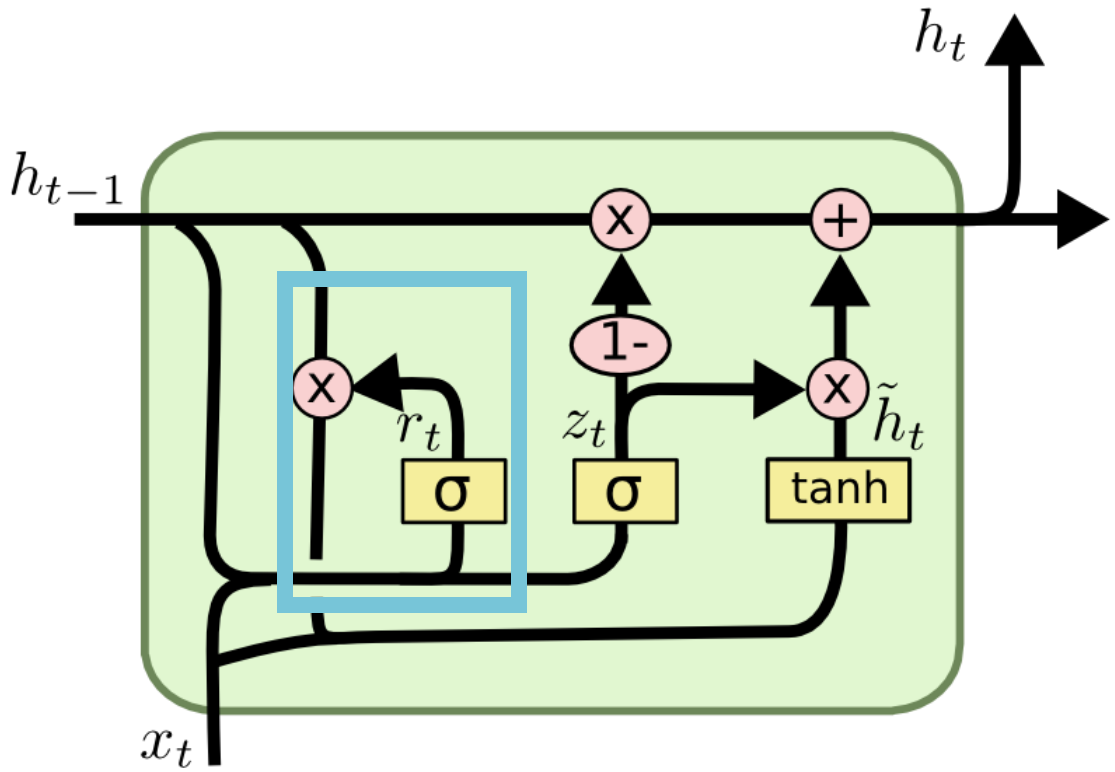
- Can we simplify the LSTM architecture?
  - ... without losing performance
  - ... maybe even increasing performance due to simpler structure/easier gradient flow?

- Gated Recurrent Unit (GRU) by Cho et al., 2014

GRU Architecture

- Reset gate r_t

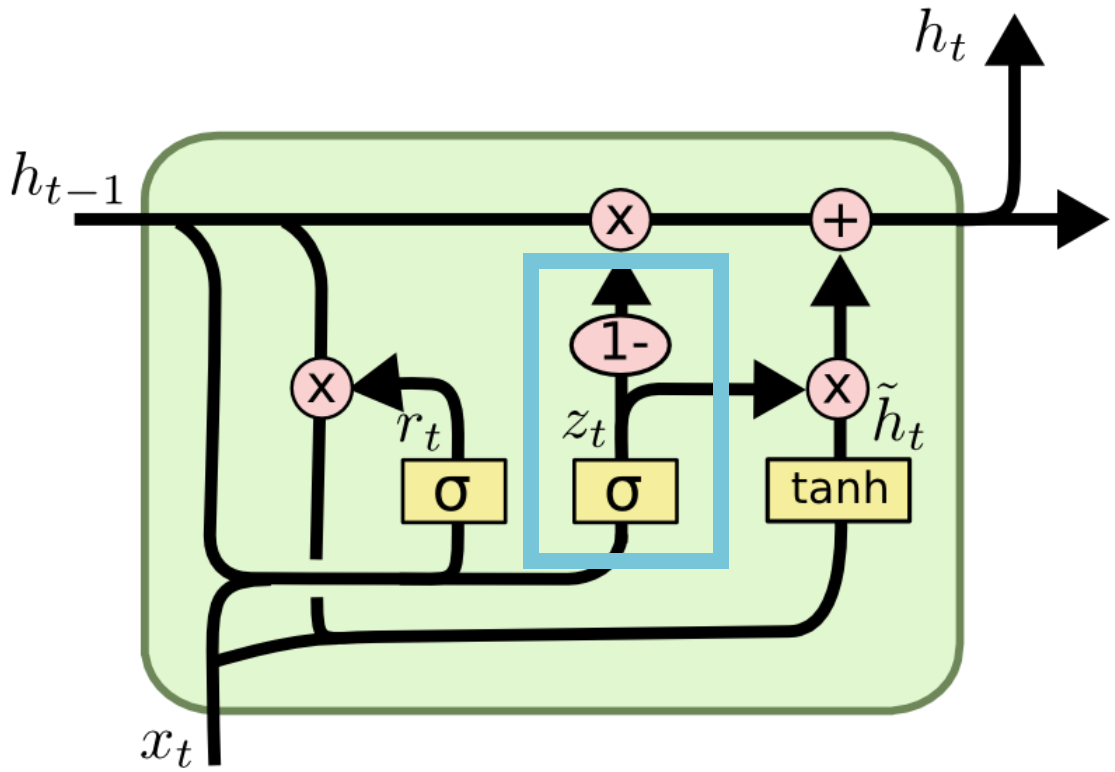
$$r_t = \sigma(W_r h_{t-1} + U_r x_t + b_r)$$



GRU Architecture

- Update gate z_t

$$r_t = \sigma(W_r h_{t-1} + U_r x_t + b_r)$$
$$z_t = \sigma(W_z h_{t-1} + U_z x_t + b_z)$$

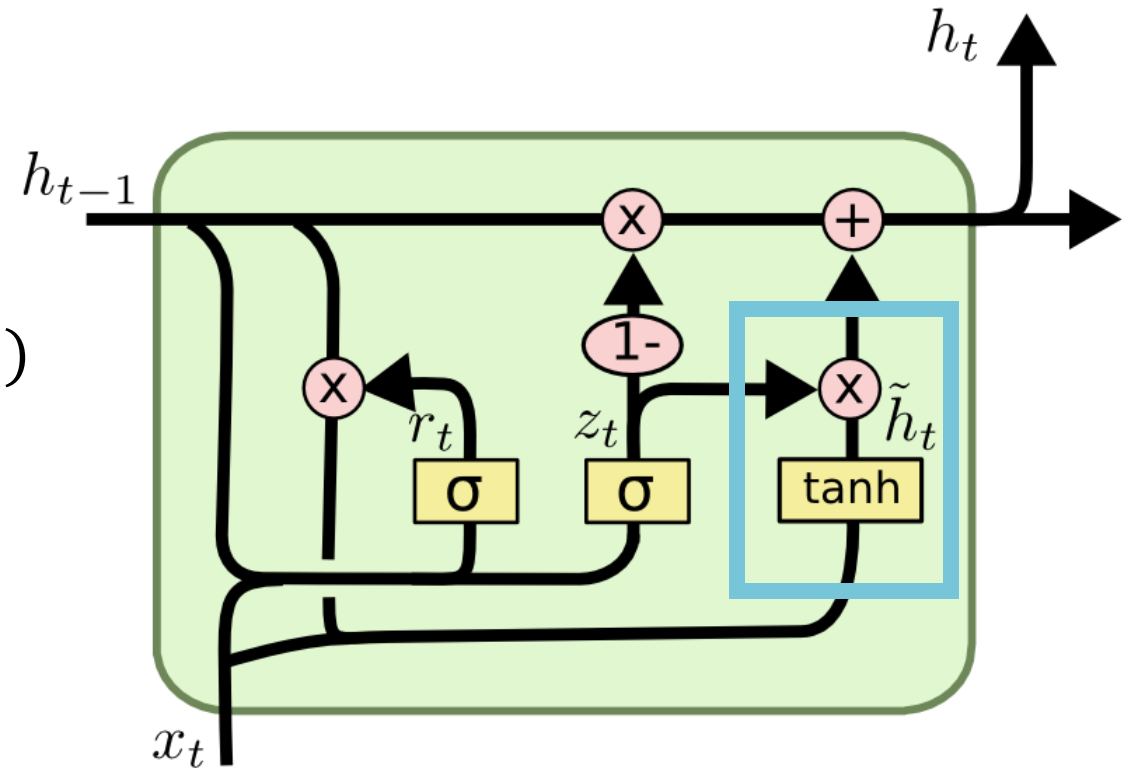


GRU Architecture

- New memory $\tilde{h}_t$

$$r_t = \sigma(W_r h_{t-1} + U_r x_t + b_r)$$
$$z_t = \sigma(W_z h_{t-1} + U_z x_t + b_z)$$
$$\tilde{h}_t = \tanh(W_h(r_t * h_{t-1}) + U_h x_t + b_h)$$

Some parts of the previous hidden state are reset

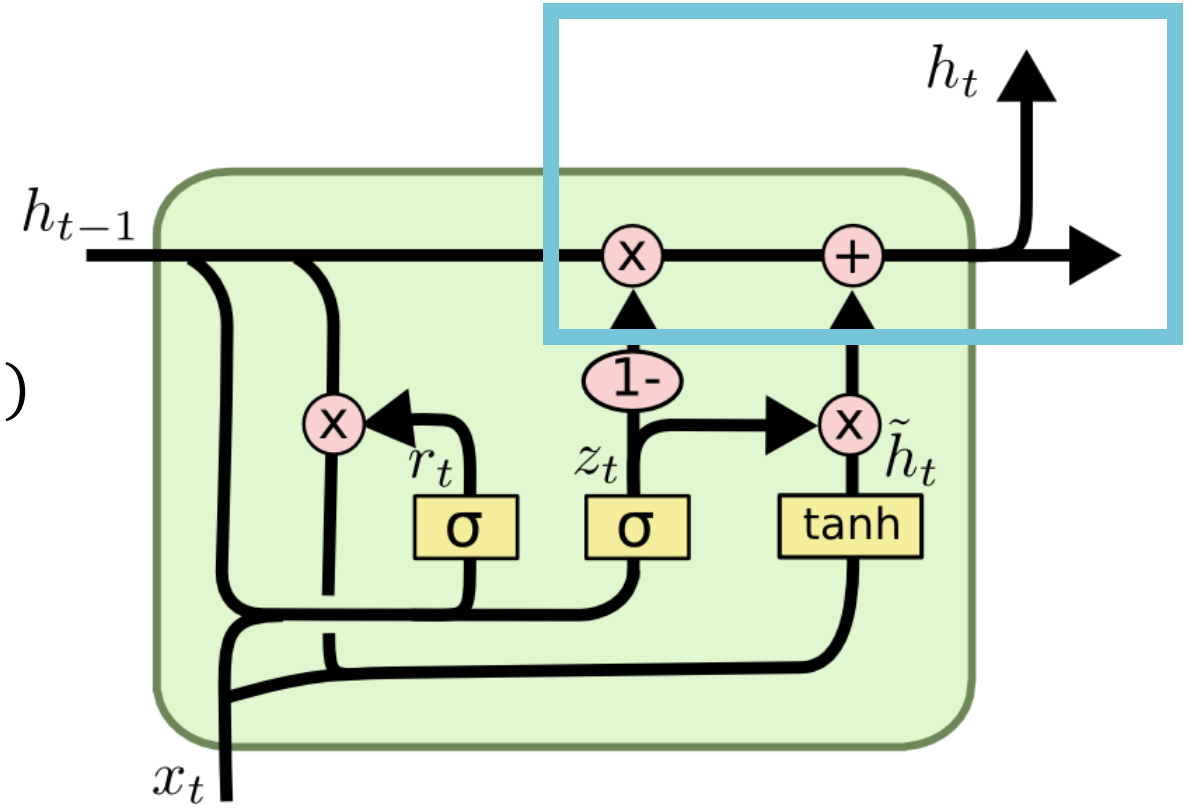


GRU Architecture

- Updated hidden state h_t

$$r_t = \sigma(W_r h_{t-1} + U_r x_t + b_r)$$
$$z_t = \sigma(W_z h_{t-1} + U_z x_t + b_z)$$
$$\tilde{h}_t = \tanh(W_h(r_t * h_{t-1}) + U_h x_t + b_h)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

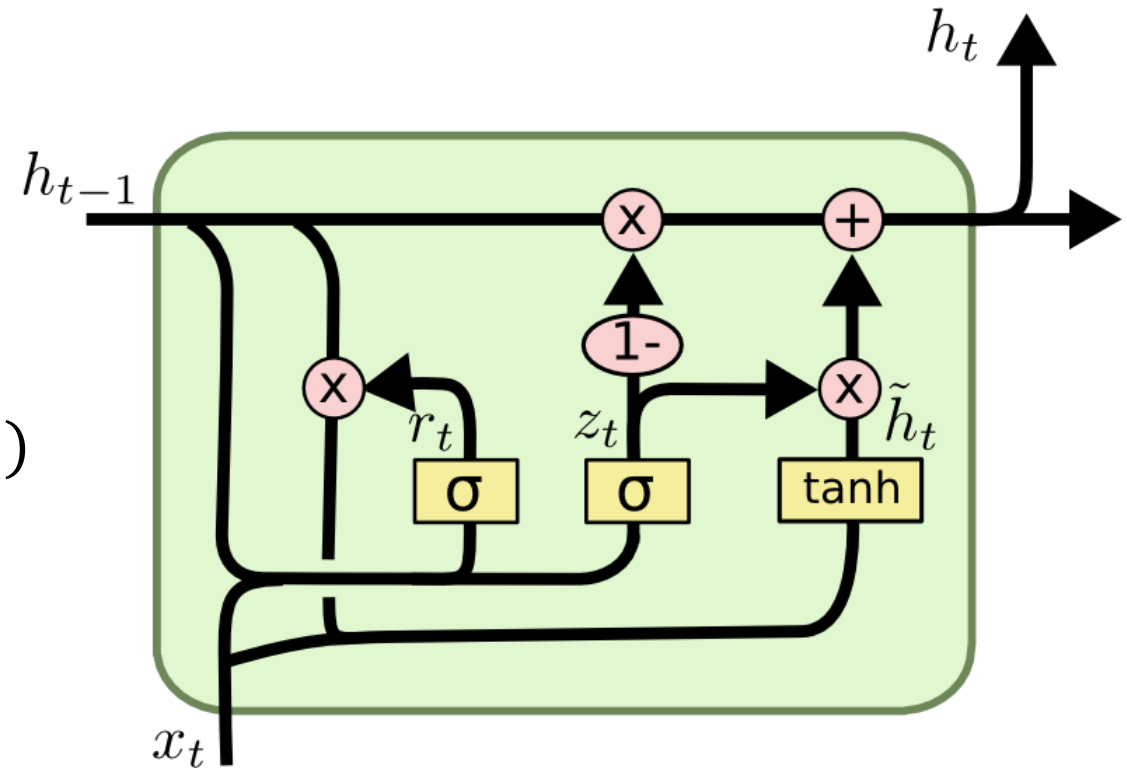
Combination of old and new hidden state (with a single update gate vs. forget and input gates in LSTM)



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

GRU Definition

$$
\begin{aligned}
r_t &= \sigma(W_r h_{t-1} + U_r x_t + b_r) \\
z_t &= \sigma(W_z h_{t-1} + U_z x_t + b_z) \\
\tilde{h}_t &= \tanh(W_h (r_t * h_{t-1}) + U_h x_t + b_h) \\
h_t &= (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t
\end{aligned}
$$



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

LSTM vs. GRU

- GRU a bit faster and easier to train
- Similar overall task performance
  - Which one is better depends on the task and dataset
  - … and probably on how lucky you got with selecting the best hyperparameters

Exercise: LSTM