HERIOT WATT UNIVERSITY

Applied Text Analytics (F20/21AA)

Heriot Watt University , Dubai Campus

Dr. Radu Mihailescu

RNN (Recurrent Neural Network)

Brief Recap

RNN Important Take Away

- Take a sequential input of any length
- Apply the same weights on each step (network parameters are shared among time stamps, i.e. activation layers)

(Model size not affected by longer input)

- Should be able to able to use information from previous inputs (i.e. previous words in the sentence)

I have a cat, its name is Kitty, and it likes to drink ……

However..

- Vanishing gradient problem prevents RNNs from learning long term dependencies

Vanishing/exploding gradients

A general problem in deep NN

- Derivative decreases/grows exponentially as a function of the layer

- Vanishing gradient: (more difficult)
  - Gradient effect from to earlier states are lost (effects of earlier inputs get diminished).
  - Basic RNNs are not good at capturing long-term dependencies.
  - *RNN cannot learn to preserve information over many timesteps (*hidden state is constantly overwritten)
  - Each step's prediction depends on the previous predictions, so it's hard to parallelize RNN operations.

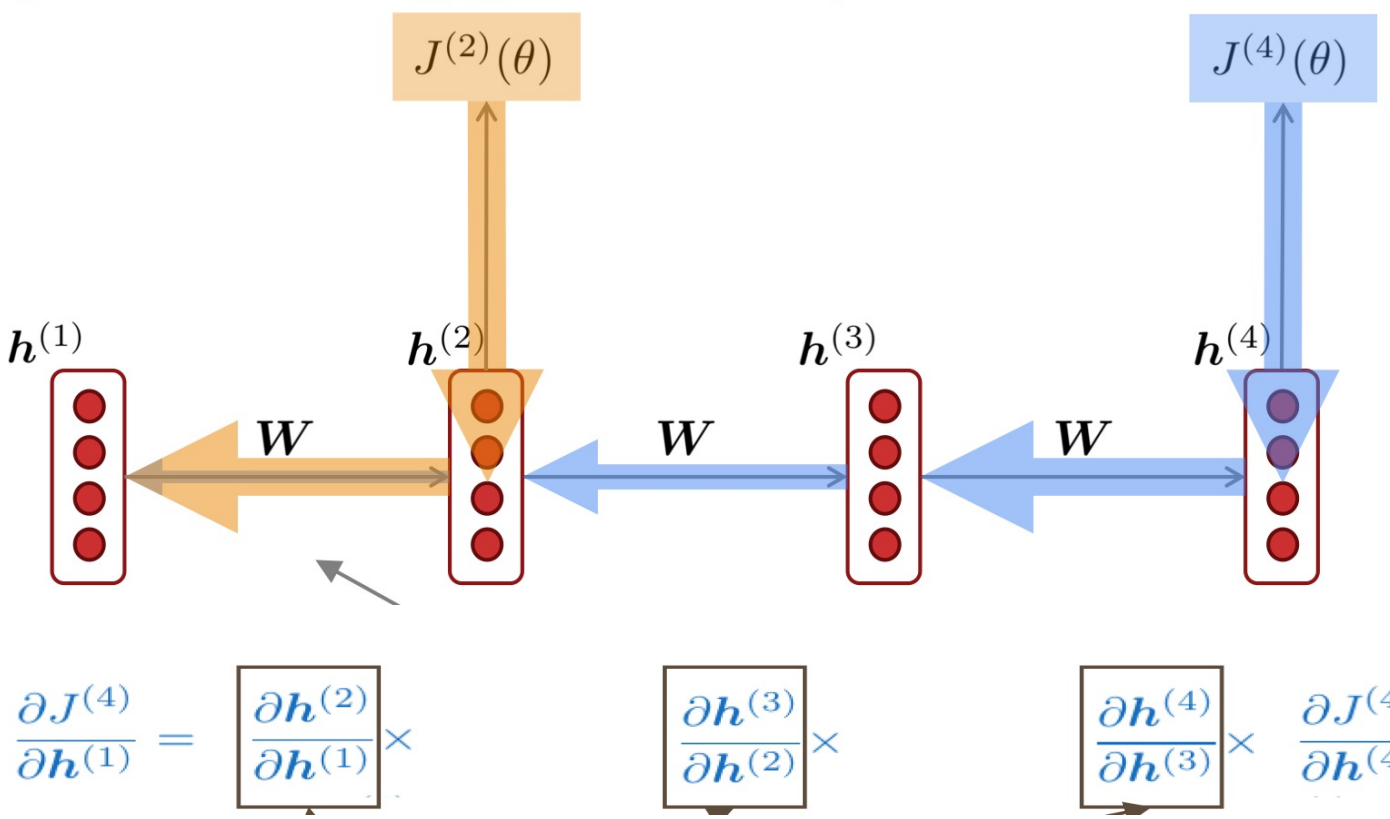
- Exploding gradients:
  - Can be solved by "gradient clipping"

Recall: Backward propagation (through time)

The output at any step will be influenced by the values in the sequence close to it. Don't work well for long dependencies. Training through updating weights to minimize loss function.

Gradient signal from faraway is lost

So model weights are updated only with respect to near effects, not long-term effects.



The main problem is that *it's too difficult for the RNN to learn to preserve information over many timesteps.*

In a vanilla RNN, the hidden state is constantly being rewritten

Gradient descent using chain rule!

Example (From Manning Lecture notes)

Sentence 1

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

Sentence 2

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

- RNNs are more likely to correctly predict the blank spot in Sentence ? than in Sentence ?.

- Why?

Example (From Manning Lecture notes)

Sentence 1
"Jane walked into the room. John walked in too. Jane said hi to ___"

Sentence 2
"Jane walked into the room. John walked in too. It was late in the day, and everyone was walking home after a long day at work. Jane said hi to ___"

- RNNs are more likely to correctly predict the blank spot in **Sentence 1** than in **Sentence 2**.
- Contribution of ***gradient values gradually vanishes as they propagate to earlier timesteps***

To summarize: Recurrent Neural Networks

RNN family of neural networks for temporal/sequential data

- RNN popular model that has been used in NLP.

RNN can perceive the text as a sequence of characters & words.

- problem they face vanishing/exploding gradients

Solutions widely used: LSTM and GRU cells.

- Today will show how to implement RNNs using TensorFlow (lab) & for text classification

- Next: RNN more fancy models & applications

Beyond Vanilla RNNs

- Though vanishing/exploding gradients are a general problem, RNNs are particularly unstable due to the repeated multiplication by the same weight matrix [Bengio et al, 1994]

- Solutions widely used: LSTM and GRU cells & some variations

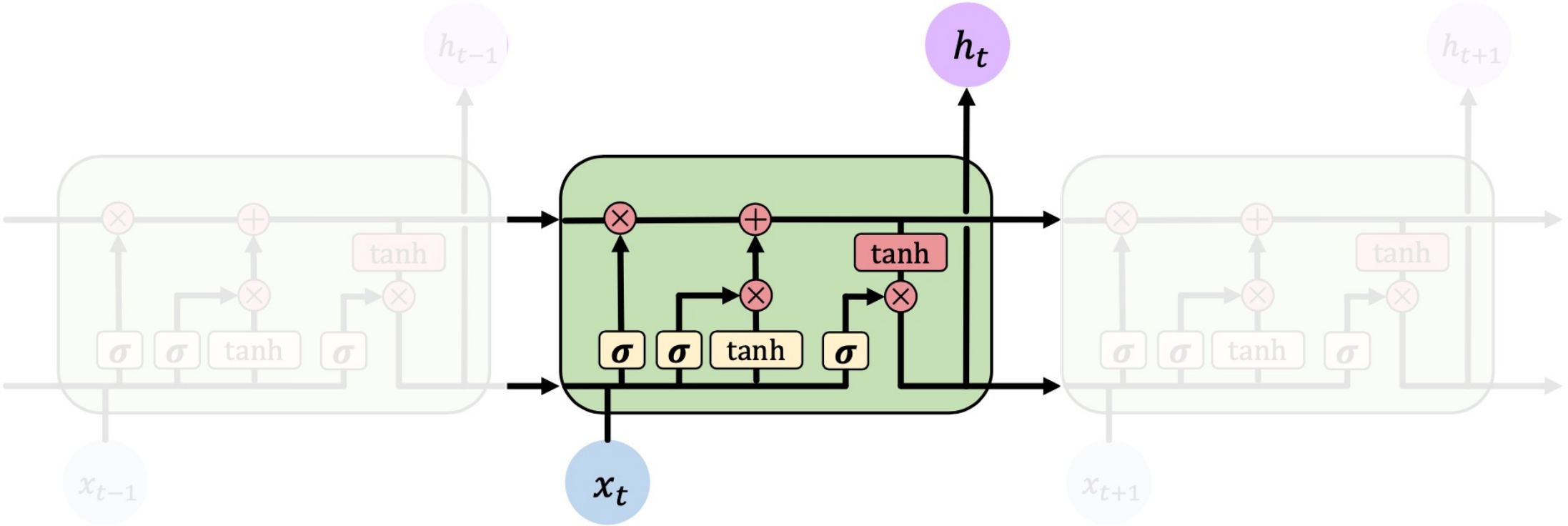
(more flavours than vanilla)

- More complicated RNNs that use **gates to control information** flow; they are more resilient to vanishing gradients (add notion of **memory**)

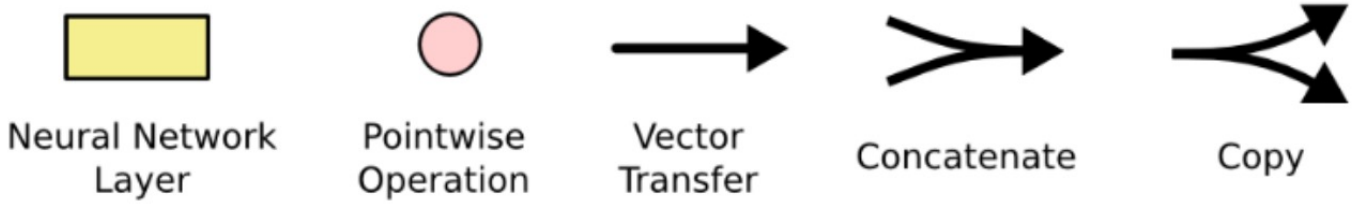
Long Short Term Memory (LSTM)

Long Short Term Memory (LSTMs)

LSTM cells are designed to track information throughout many timesteps

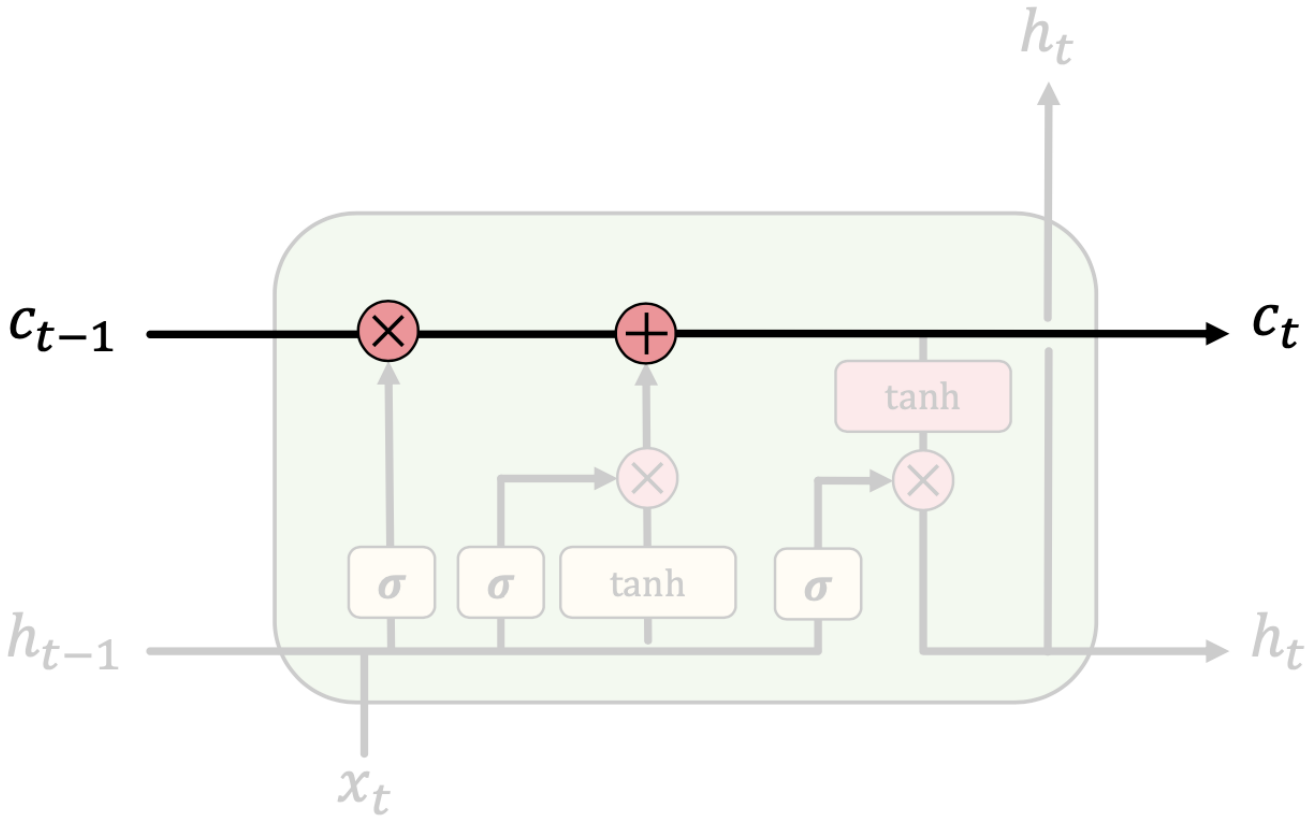


Long Short-term memory, Hochreiter et al., 1997



Long Short Term Memory (LSTMs)

LSTMs maintain a **cell state c_t** where it's easy for information to flow



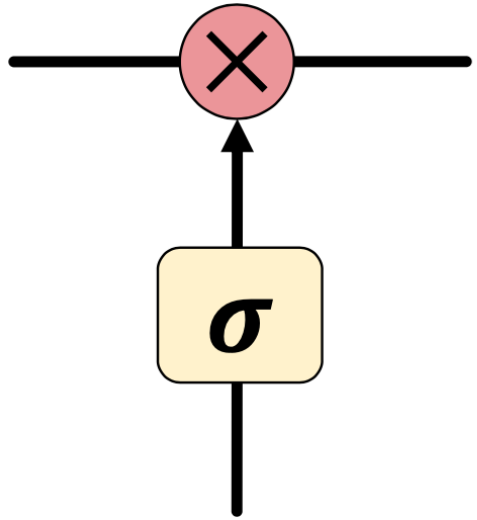
Long Short Term Memory (LSTMs)

Conveyer belt for previous state and new data:

- Decide what to **forget** (state)
- Decide what to **remember** (state)
- Decide what to **output** (if anything)

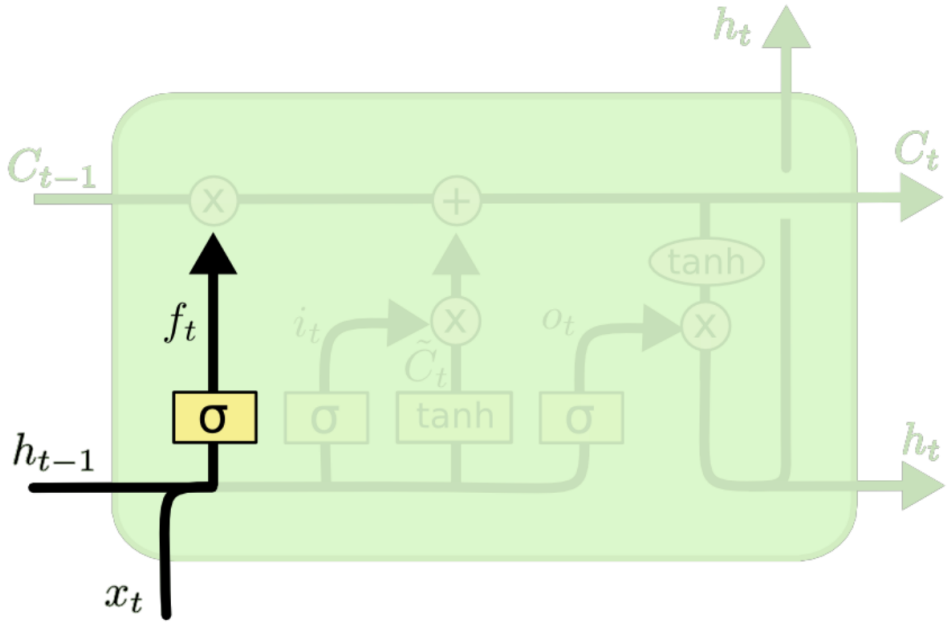
Long Short Term Memory (LSTMs)

Information is **added** or **removed** to cell state through structures called **gates**



Gates optionally let information through, via a sigmoid neural net layer and pointwise multiplication

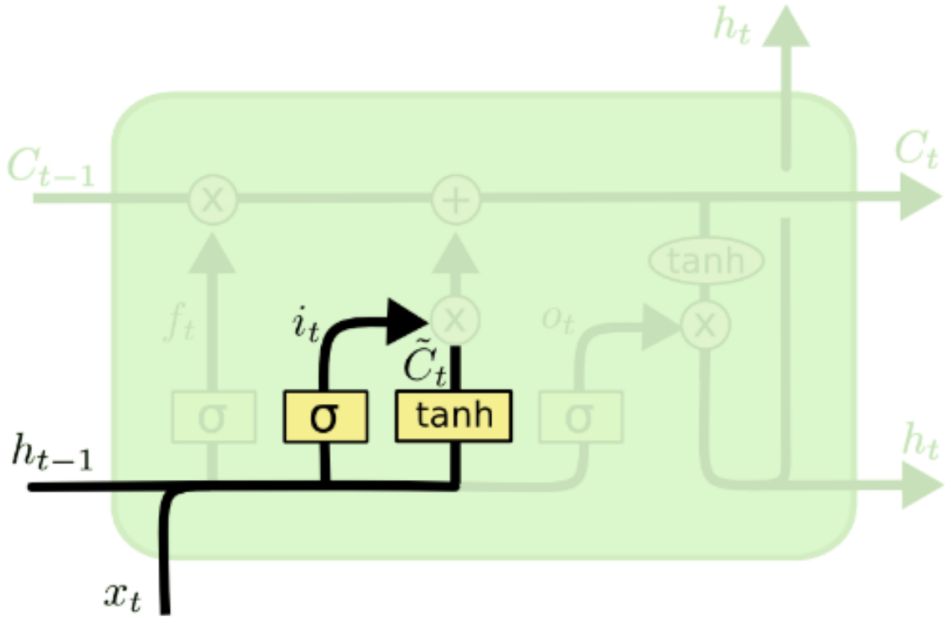
Long Short Term Memory (LSTMs)



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

Step 1: Decide what to forget / ignore

Long Short Term Memory (LSTMs)

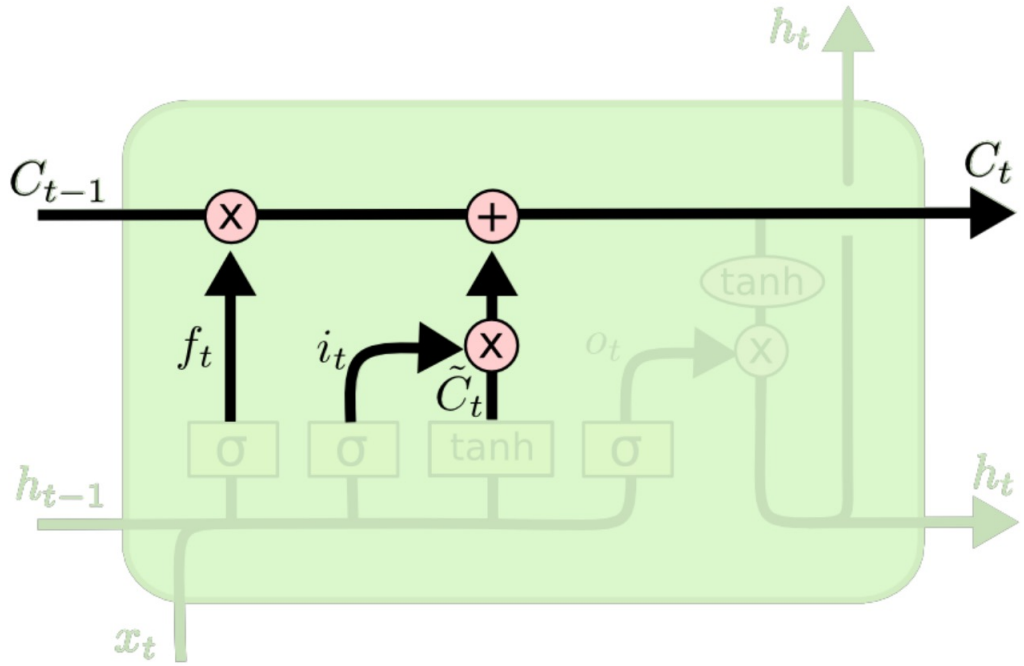


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

Step 2: Decide which state values to update (w/ sigmoid) and what values to update with (w/ tanh)

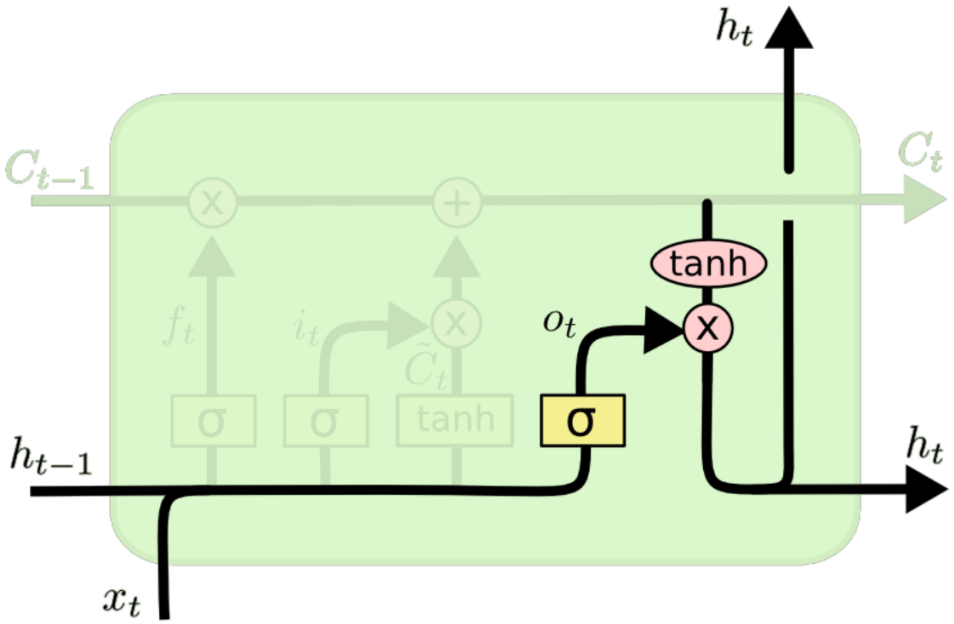
Long Short Term Memory (LSTMs)



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

Step 3: Perform the forgetting and the state update

Long Short Term Memory (LSTMs)



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

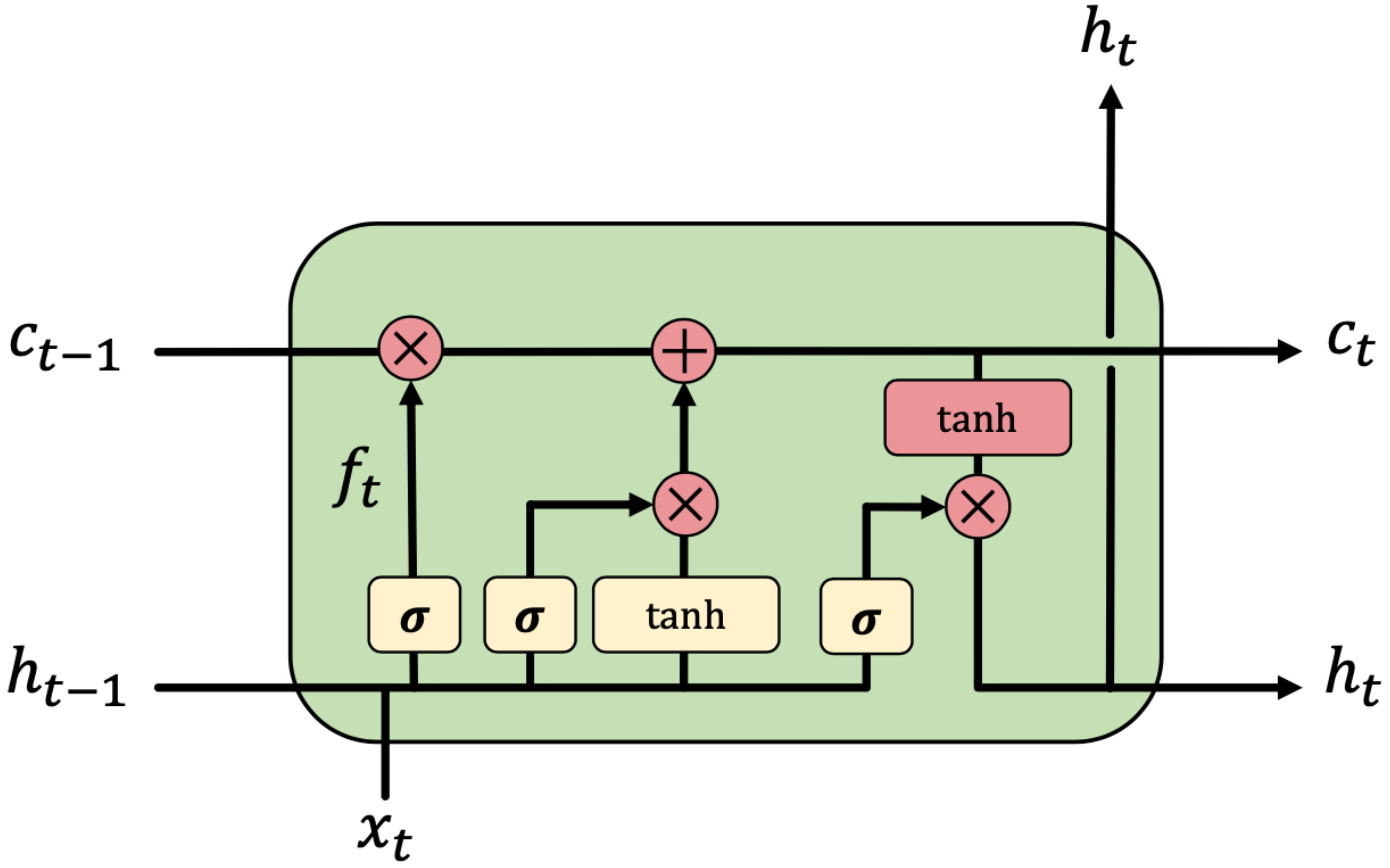
$$h_t = o_t * \tanh\left(C_t\right)$$

Step 4: Produce output with tanh [-1, 1] deciding the values and sigmoid [0, 1] deciding the filtering

Long Short Term Memory (LSTMs)

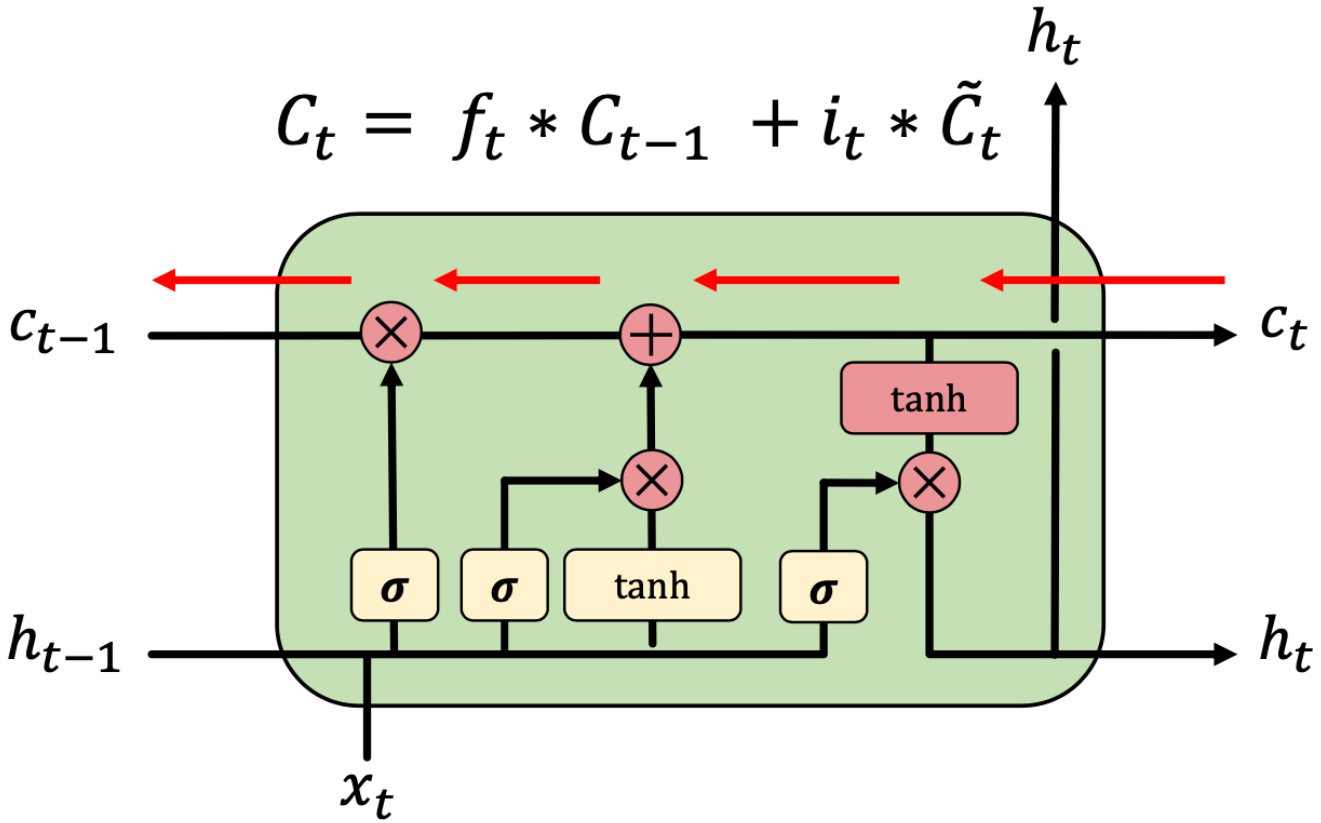
How do LSTMs work?

1) Forget 2) Update 3) Output



Long Short Term Memory (LSTMs)

Backpropagation from C_t to C_{t-1} requires only elementwise multiplication!
No matrix multiplication → avoid vanishing gradient problem.



Long Short Term Memory (LSTMs)

Maintain a **separate cell state** from what is outputted

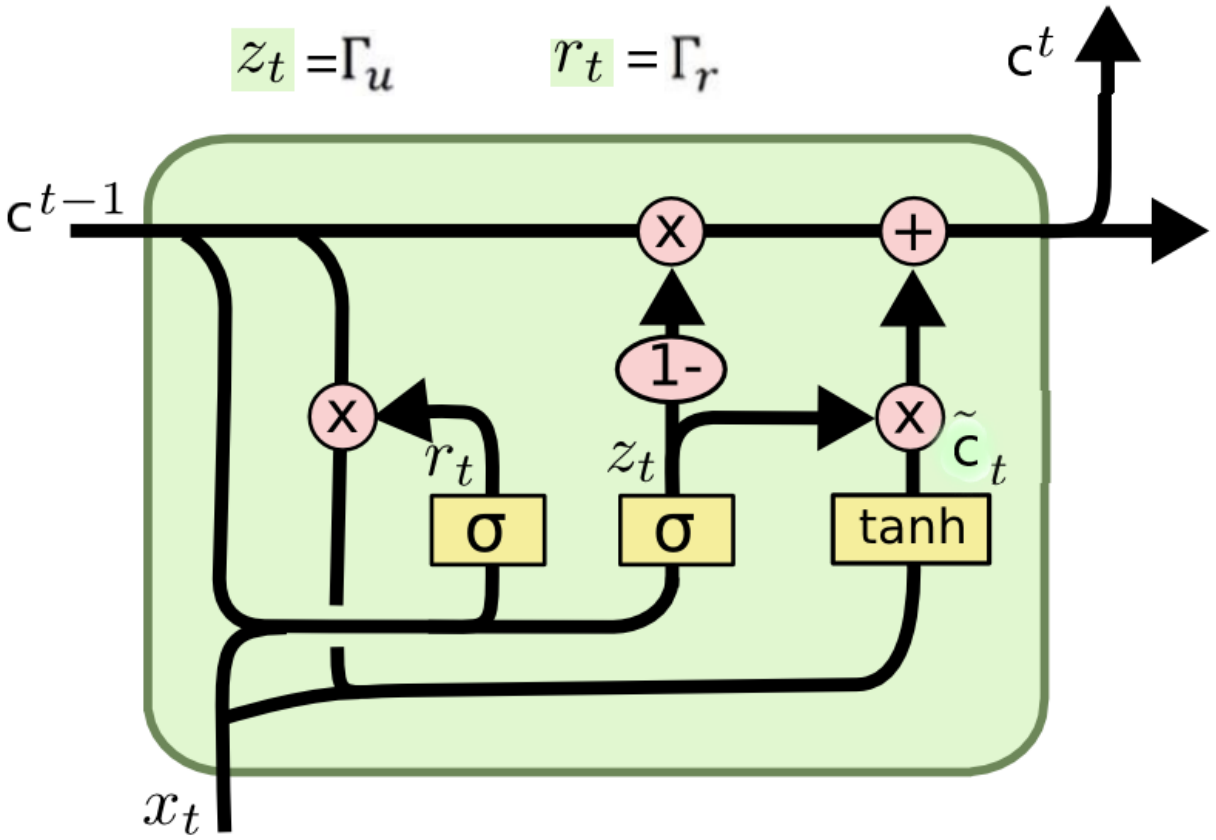
Use **gates** to control the **flow of information**

- Forget gate gets rid of irrelevant information
- Selectively update cell state
- Output gate returns a filtered version of the cell state

Backpropagation from c_t to c_{t-1} doesn't require matrix multiplication: **uninterrupted gradient flow**

Gated Recurrent Unit (GRU)

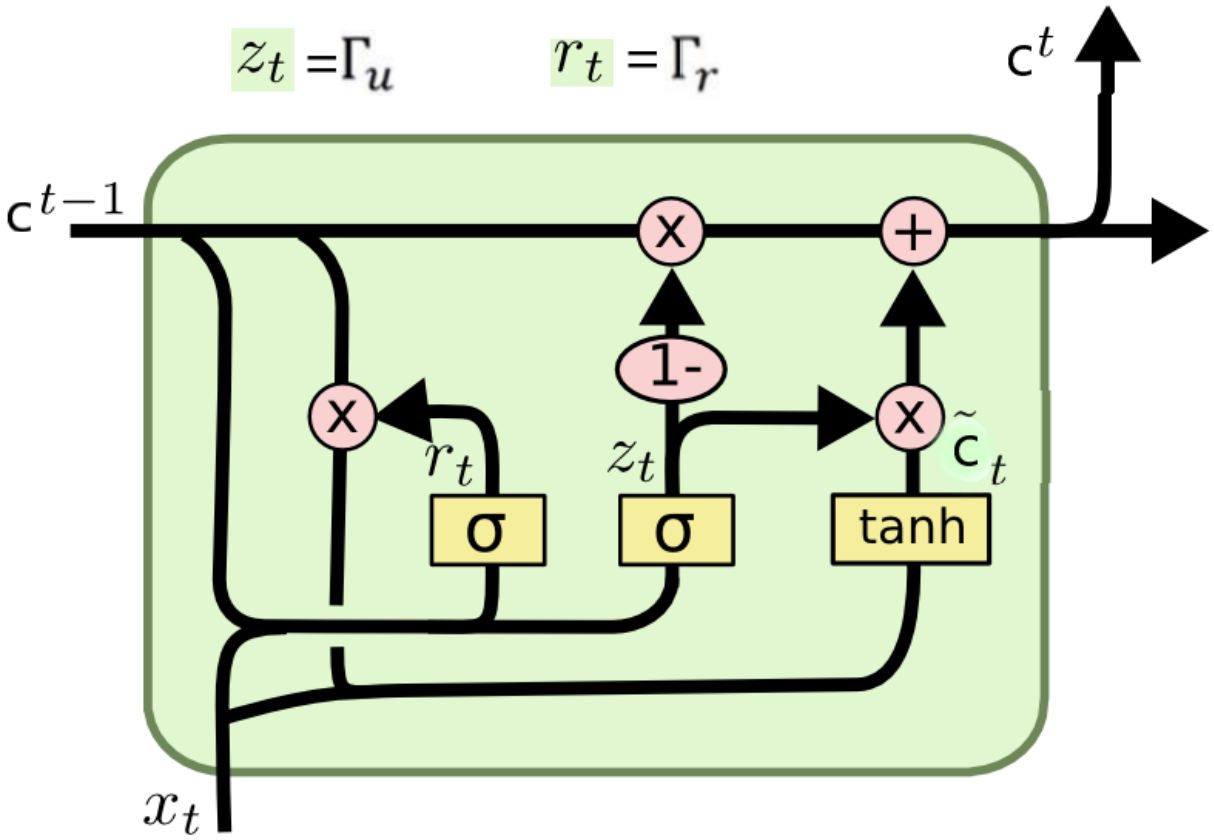
Gated Recurrent Units (GRUs)



- The GRU cell is a simplified version of the LSTM with similar performance.
- Both state vectors are merged into a single vector.

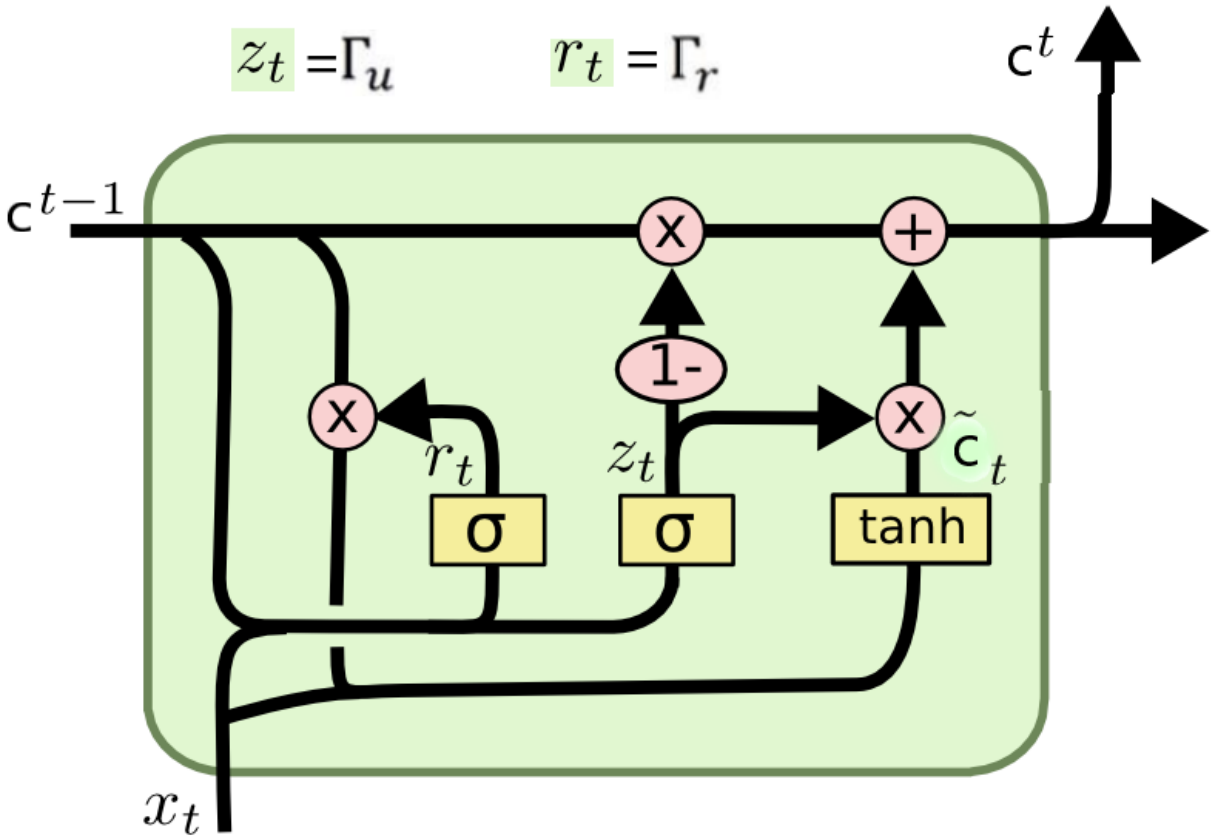
Learning phrase representations using RNN encoder-decoder for statistical machine translation, Cho et al. 2014

Gated Recurrent Units (GRUs)



- **Update gate** decides if the cell state should be updated with the candidate state(current activation value) or not.
- **Reset gate** is used to decide whether the previous cell state is important or not. Sometimes the reset gate is not used in simple GRU.

Gated Recurrent Units (GRUs)



- If reset close to 0, ignore previous hidden state (allows the model to drop information that is irrelevant in the future).
- If update gate close to 1, then we can copy information in that unit through many steps
- Gamma Controls how much of past state should matter now

Recap RNNs

- RNNs are more powerful because they capture a context of potentially “infinite” size.
- The hidden state of a RNN can be interpreted as a way to represent the history of what has been seen so far.
- RNNs can be useful to represent variable length sentences.
- There are lots of RNN variants. The best working ones have gating (units that multiply other units): e.g.: LSTM and GRU.

What works better GRU or LSTM?

- No clear winner, both perform well on different problems
- GRUs is quicker to compute and has fewer parameters and therefore scales better
- LSTMs have been used as **first thing** to try because they are powerful and have been there for a while. They have been proven to be powerful and flexible

(could work really well if you have lots of data & long dependencies)

- The state of the ART for NLP now are **Transformer models** (Machine translation) – Gentle introduction to follow! How Transformers work

Resources

Textbooks:

Speech and Language Processing (3rd ed. draft) -Chapter 9
Dan Jurafsky and James H. Martin

https://web.stanford.edu/~jurafsky/slp3/

Video Lectures Stanford NLP & DL lecture 7 (Fancy RNN)

Lecture 8(Seq2Seq &translation) -

Interesting Reads:

Sequence Models (Based on Andrew NG DL course)

Understanding LSTMs

Recurrent Neural Networks cheat sheet

Thank you!
Questions?