LSTM and its variants

PUNEET KUMAR JAIN, PhD

National Institute of Technology Rourkela

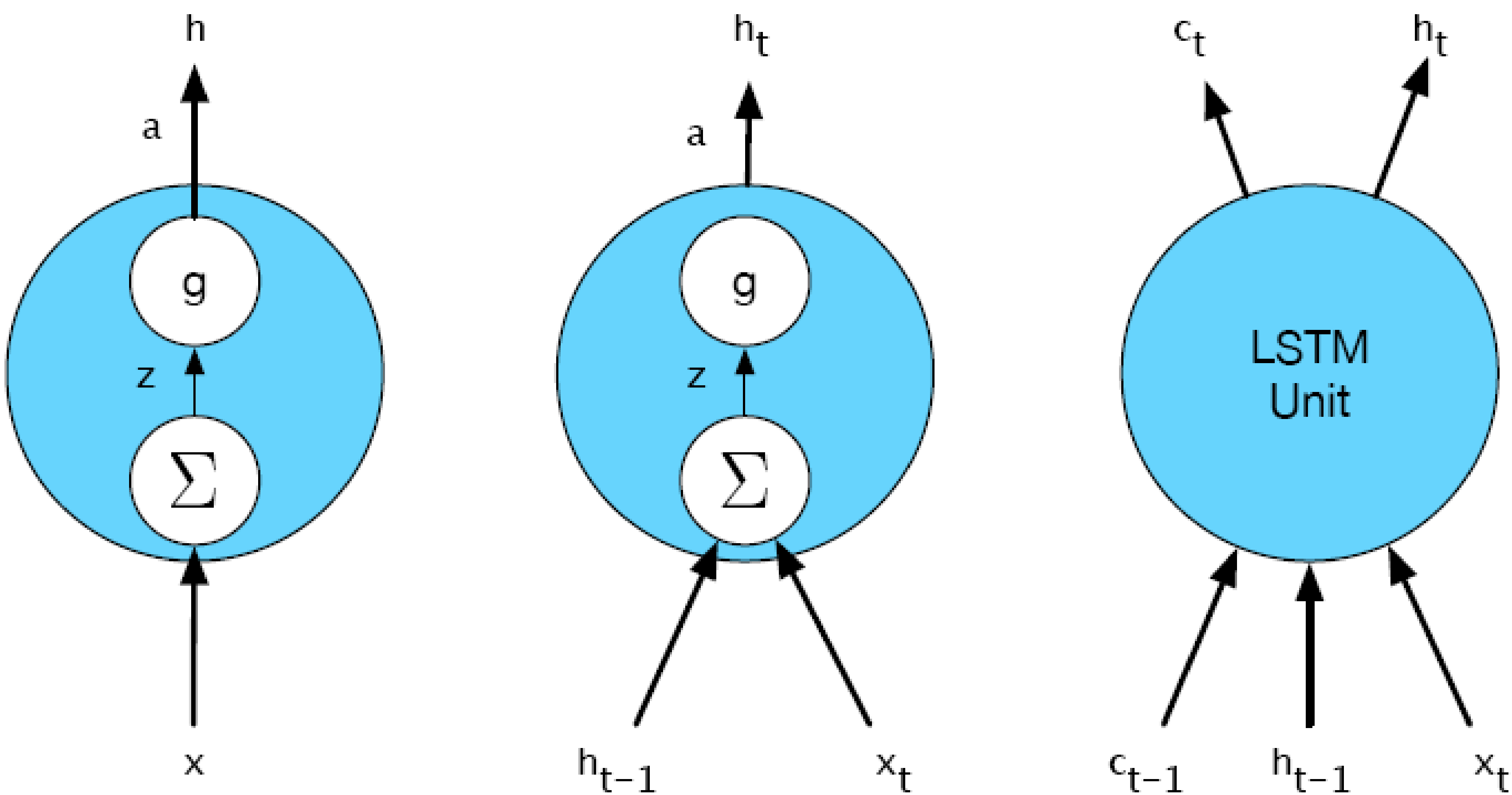
References

- Daniel Jurafsky, James H. Martin, “Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition”, 2nd Ed., PEARSON, 2013

- Several slides adapted from: Chetan Arora, IIT Delhi; Vineeth Balasubramanian, IIT Hyderabad; and others

- Basics of Data Science: https://learncloudbits.com/post/5-core-steps-to-understand-machine-learning-workflow

Feed-Forward NN vs LSTM



Feed-forward NN

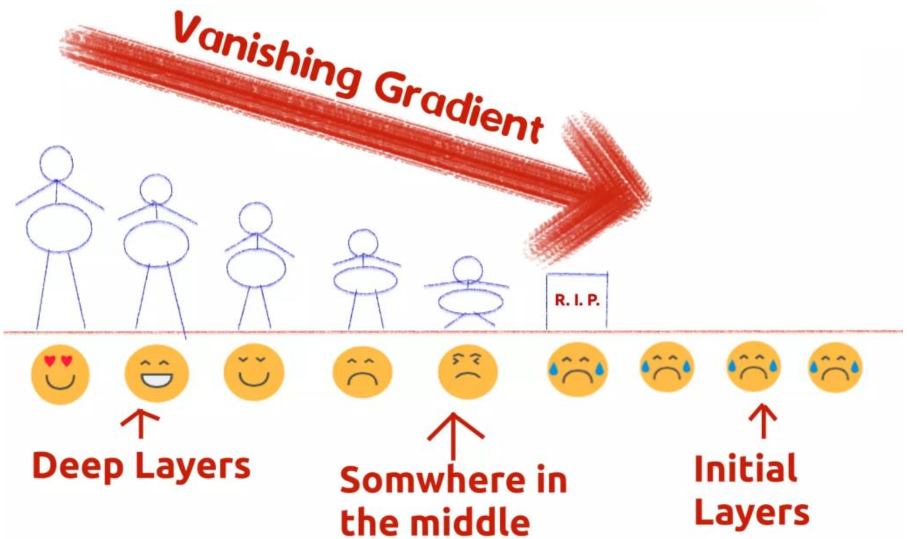
Simple recurrent networks

long short-term memory NN

Challenges with RNN: Long-Term Dependencies

- The hidden layer in SRNs are being asked to perform two tasks simultaneously:
  - Provide information useful to the decision being made in the current context, and
  - Updating and carrying forward information useful for future decisions.

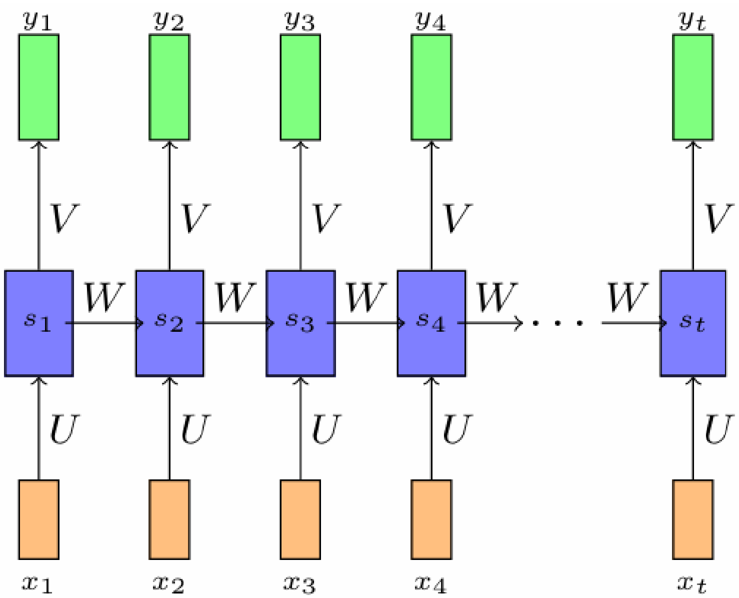
- A second difficulty (**vanishing gradients**) to successfully training simple recurrent networks arises during the backward pass of training



• LSTM

- Selective Read
- Selective wright
- Selective Forget

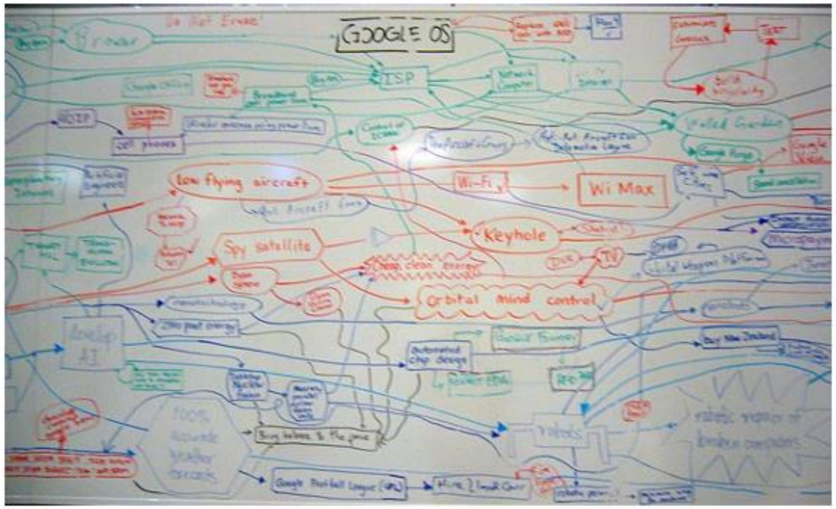
Selective read, write, and forget



- The state (s_i) of an RNN records information from all previous time steps
- At each new timestep the old information gets morphed by the current input
- One could imagine that after t steps the information stored at time step $t-k$ (for some $k < t$) gets completely morphed

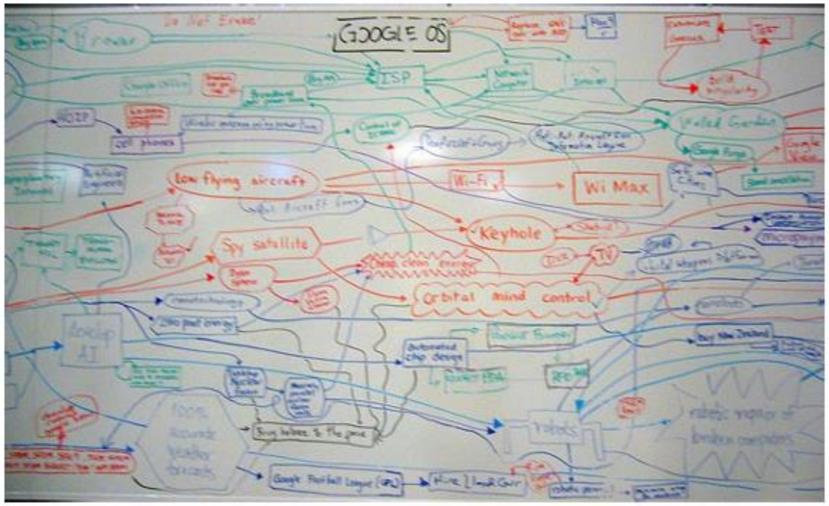
  so much that it would be impossible to extract the original information stored at time step $t-k$

Selective read, write, and forget



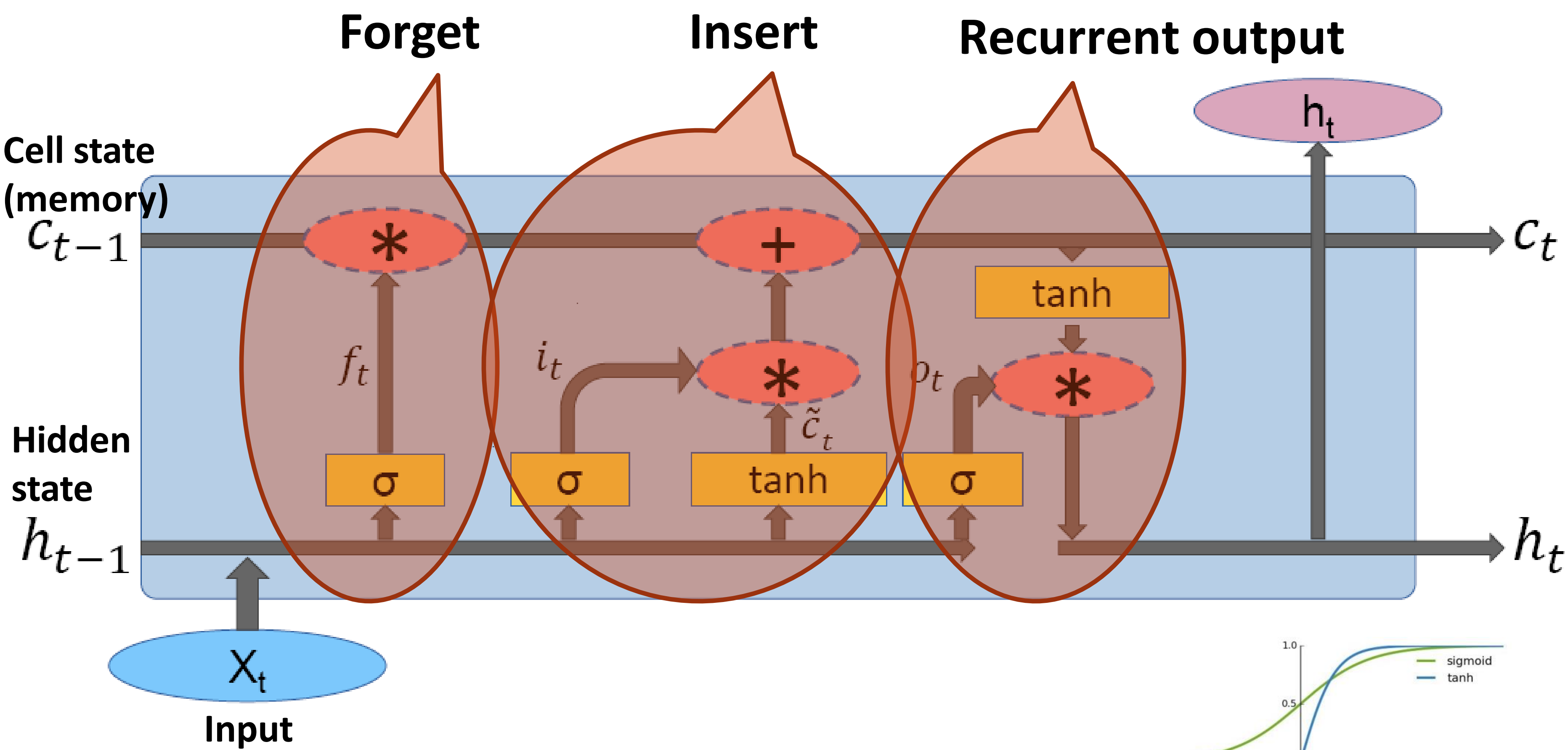
- Let us see an analogy for this
- We can think of the state as a fixed size memory
- Compare this to a fixed size white board that you use to record information
- At each time step (periodic intervals) we keep writing something to the board
- Effectively at each time step we morph the information recorded till that time point
- After many timesteps it would be impossible to see how the information at time step $t - k$ contributed to the state at timestep t

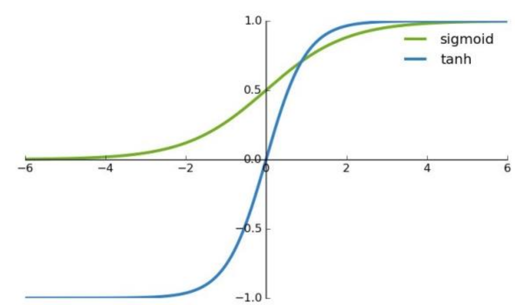
Selective read, write, and forget



- Continuing our whiteboard analogy, suppose we are interested in deriving an expression on the whiteboard
- We follow the following strategy at each time step
- Selectively write on the board
- Selectively read the already written content
- Selectively forget (erase) some content
- Let us look at each of these in detail

LSTM Networks

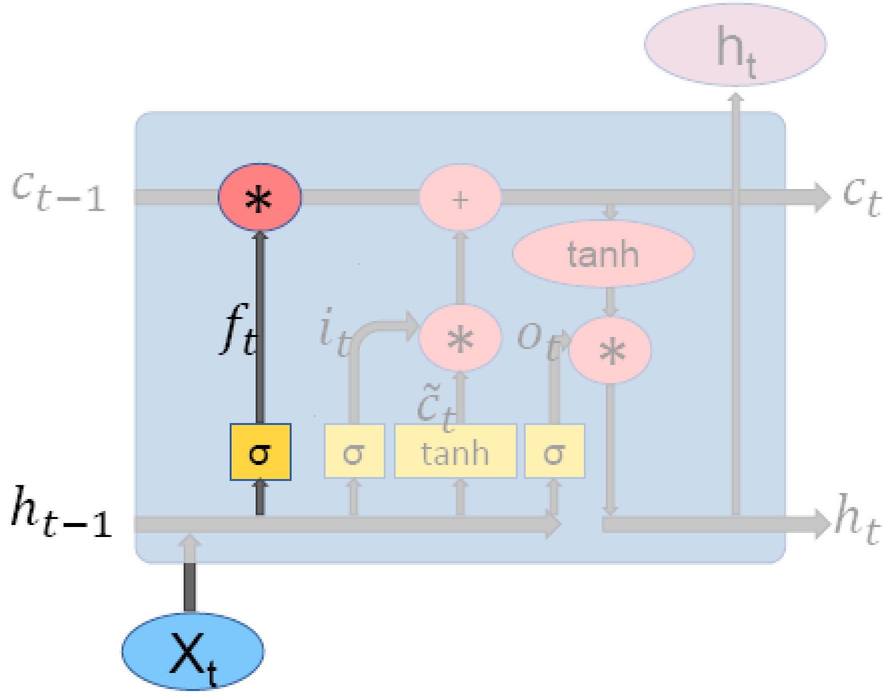




LSTM Operations: Forget

- First step is to decide what information to throw away from the cell state.
- A sigmoid layer names "forget gate layer" makes this decision.
- It looks at past state output, $\boldsymbol{h_{t-1}}$ and current input, $\boldsymbol{x_t}$ and outputs a number between 0(Forget) and 1(Keep) telling how much to keep.

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

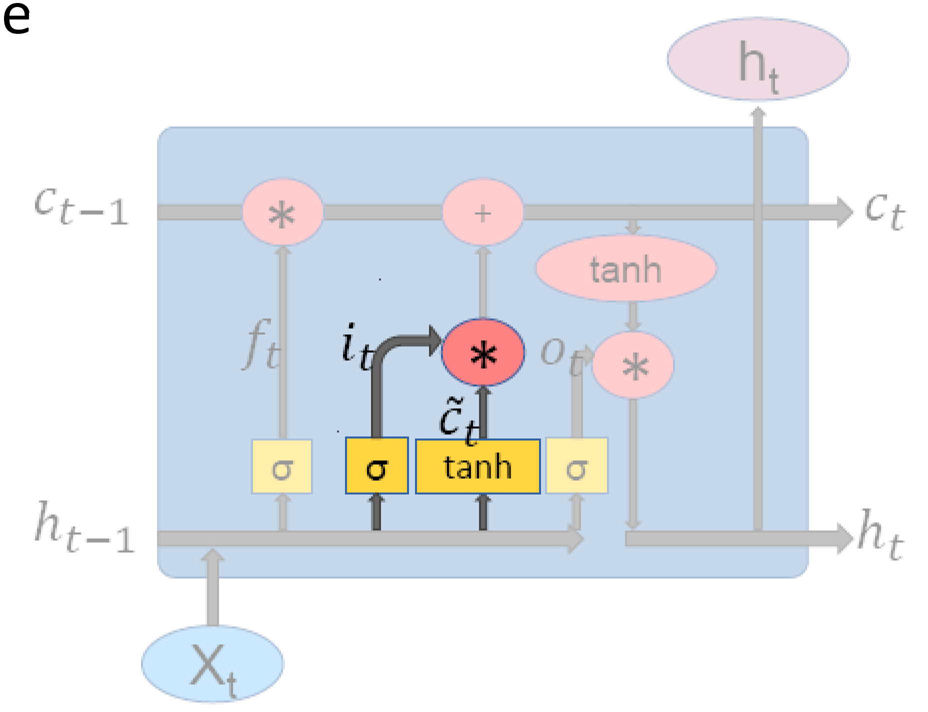


LSTM Operations: Input/Insert

- Next step is to decide what new information should be stored in the cell state.
- First, a sigmoid layer called the "input gate layer" decides which values should be updated, $\boldsymbol{i_t}$. 0(not important) 1(important)
- Next, a ***tanh*** layer creates a vector of new candidate values, $\tilde{\boldsymbol{c}}_t$ that could be added to the state

$$\boldsymbol{i}_t = \boldsymbol{\sigma}(\boldsymbol{W}_i \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_i)$$

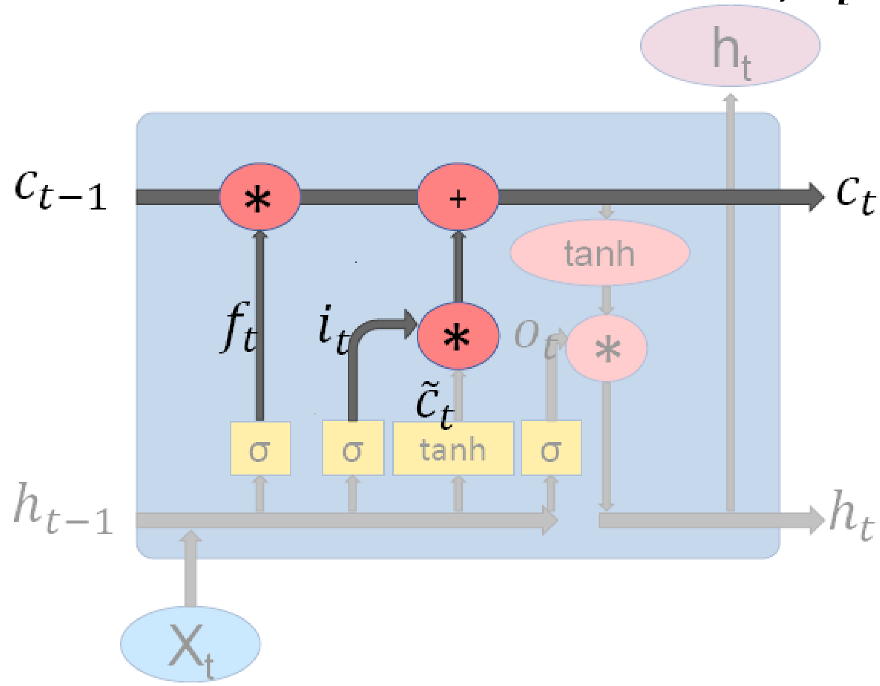
$$\tilde{\boldsymbol{c}}_t = \boldsymbol{tanh}(\boldsymbol{W}_c \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_c)$$



LSTM Operations: Update

- Old cell state, c_{t-1} is updated into the new cell state, c_t.
- Old state is multiplied by forget layer output, f_t.
- Input gate layer output, it is multiplied with candidate values, $\tilde{c}_t$and the result is added to values obtained by above multiplication.
- The output of above computations is the new candidate value, c_t.

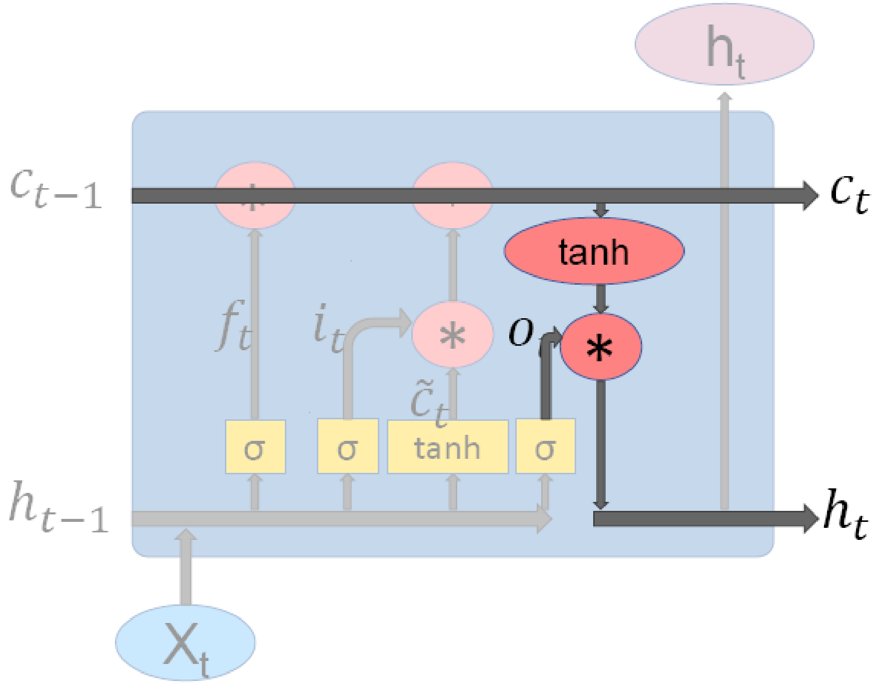
$$C_t = f_t * c_{t-1} + i_t * \tilde{c}_t$$



LSTM Operations: Output/Recurrent

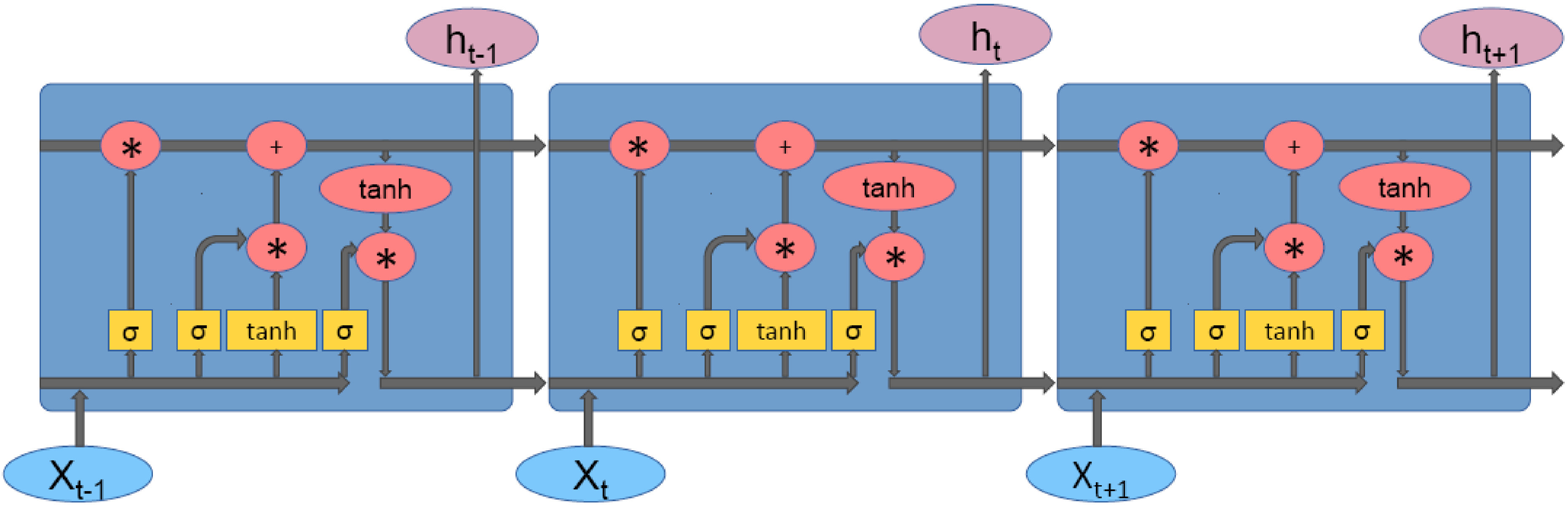
- **Final step is to decide what to output.**
- Output is based on the current cell state, $\boldsymbol{c_t}$.
- First a sigmoid layer decides what parts of hidden state is going to output.
- Then cell state is passed through a ***tanh*** layer.
- The output is then multiplied by the output of the sigmoid gate.

$$\boldsymbol{h_t = o_t * tanh(c_t)}$$

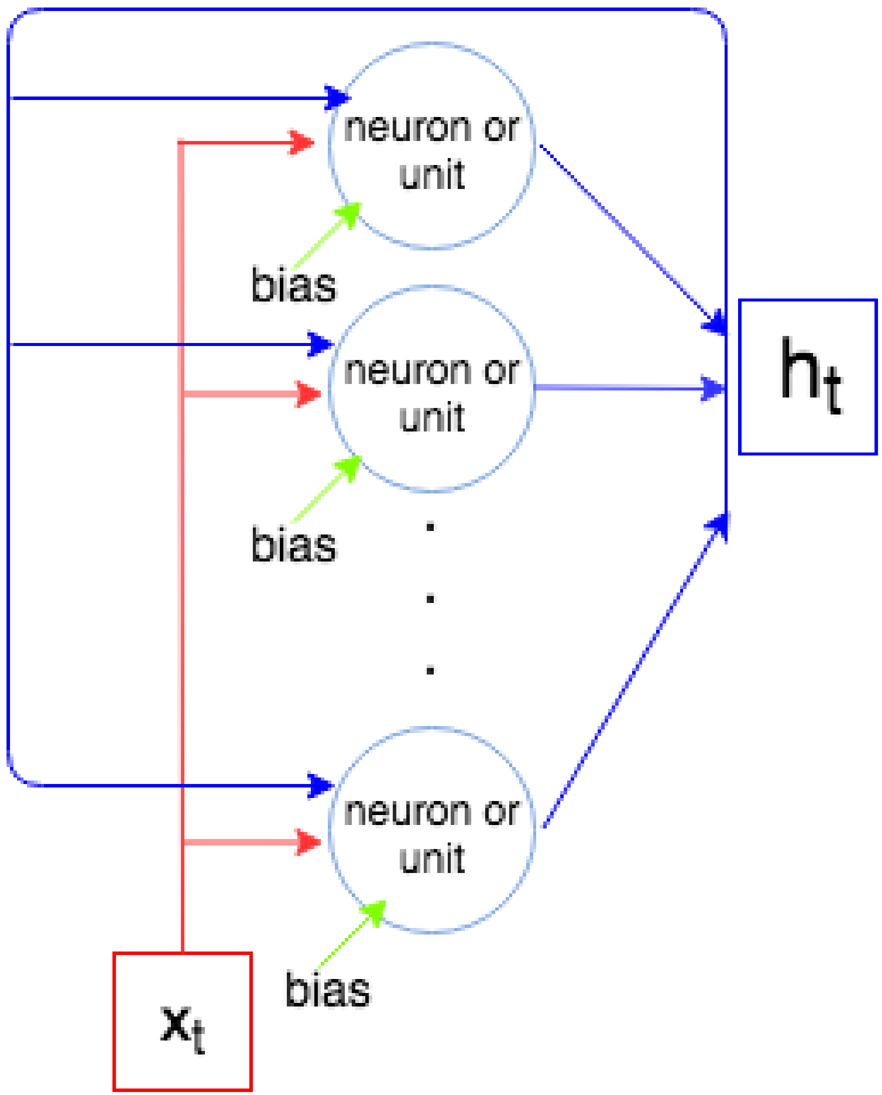


LSTM Architecture

- Each line carries an entire vector.
- Ovals represent pointwise operations, like vector addition.
- Solid rectangles are learned neural network layers.
- Lines merging denote concatenation of vectors.
- Lines splitting denotes vectors being copied and copies going to different locations.

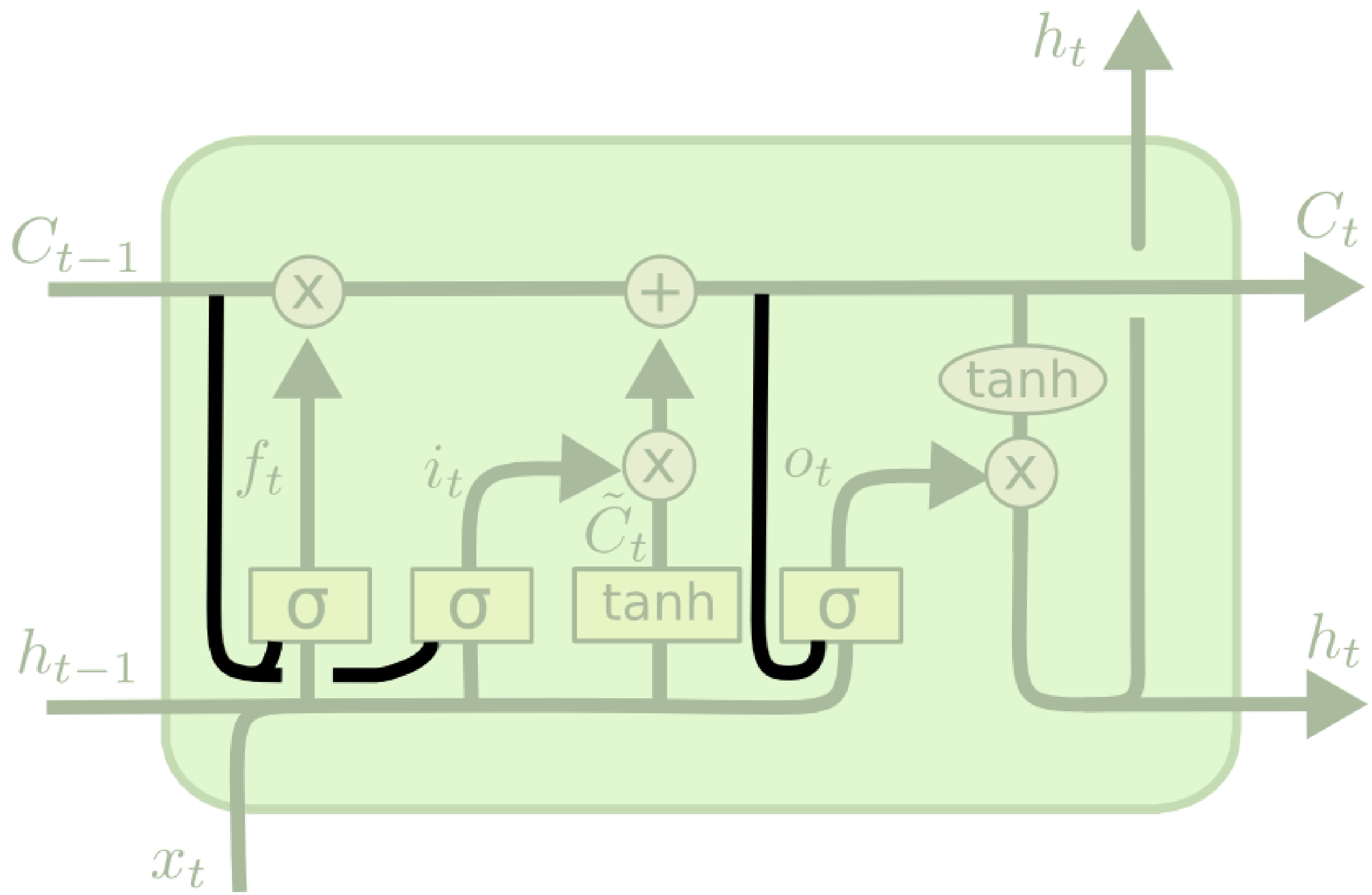


Number of units in hidden layer



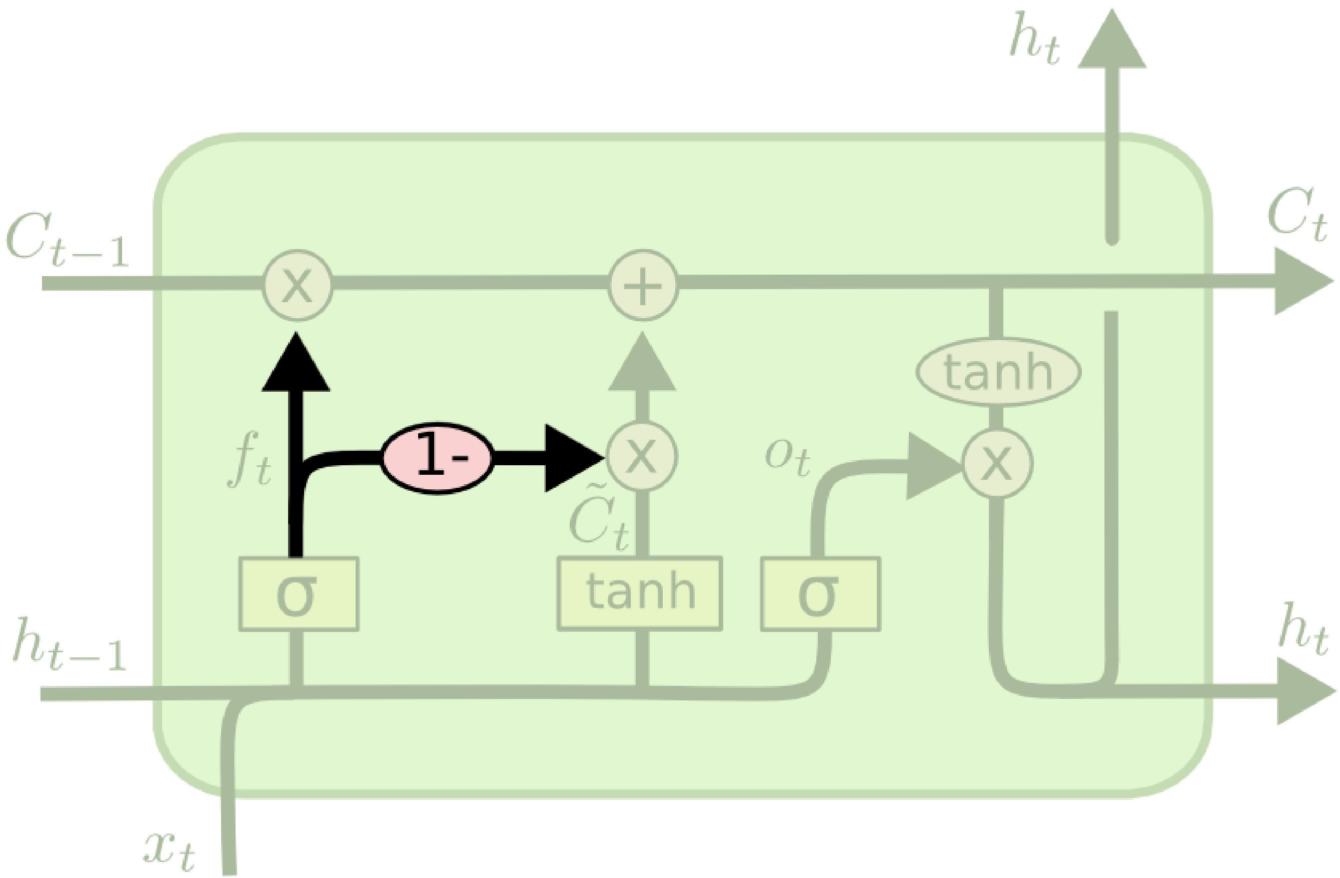
Variations of LSTM Architectures

- **The Peephole Variation**
  - allows the gate layers to read data from the cell state.
  - you could also add peepholes to some gates and not other gates.



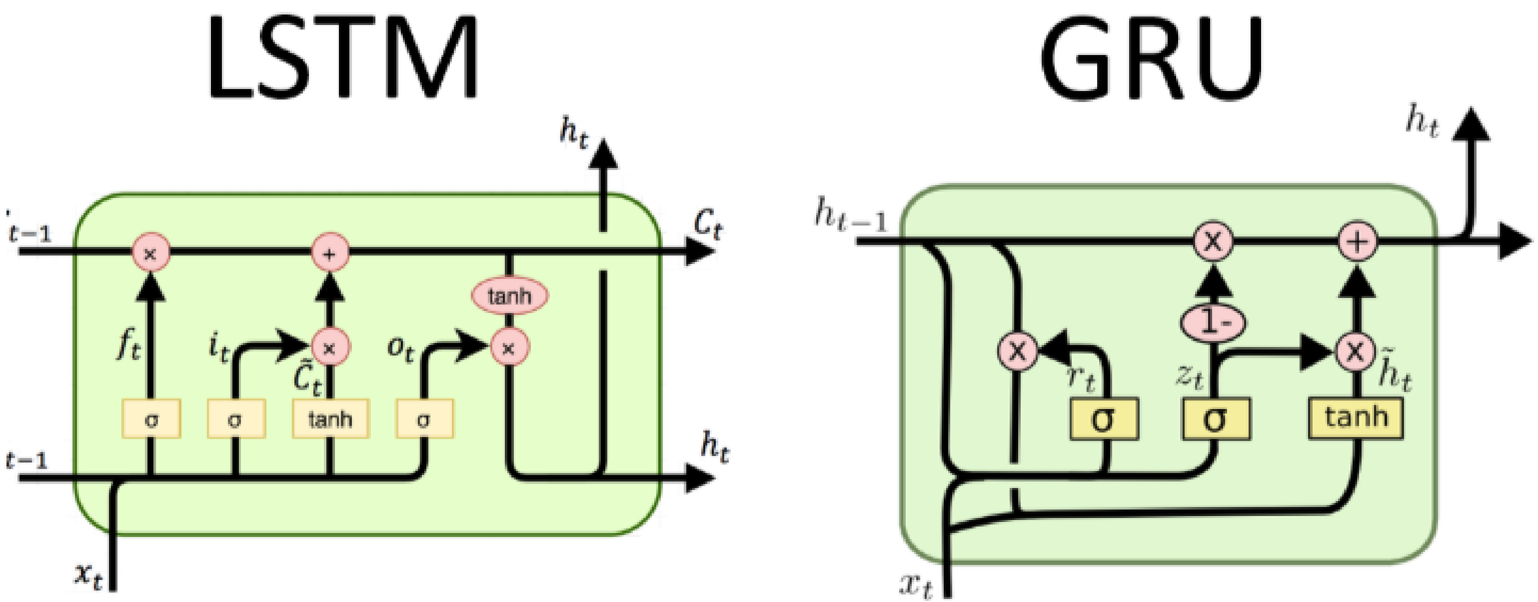
Variations of LSTM Architectures

- **The Coupled Gate Variation**
- the model makes the decision of what to forget and what to add new information to together



Variations of LSTM Architectures

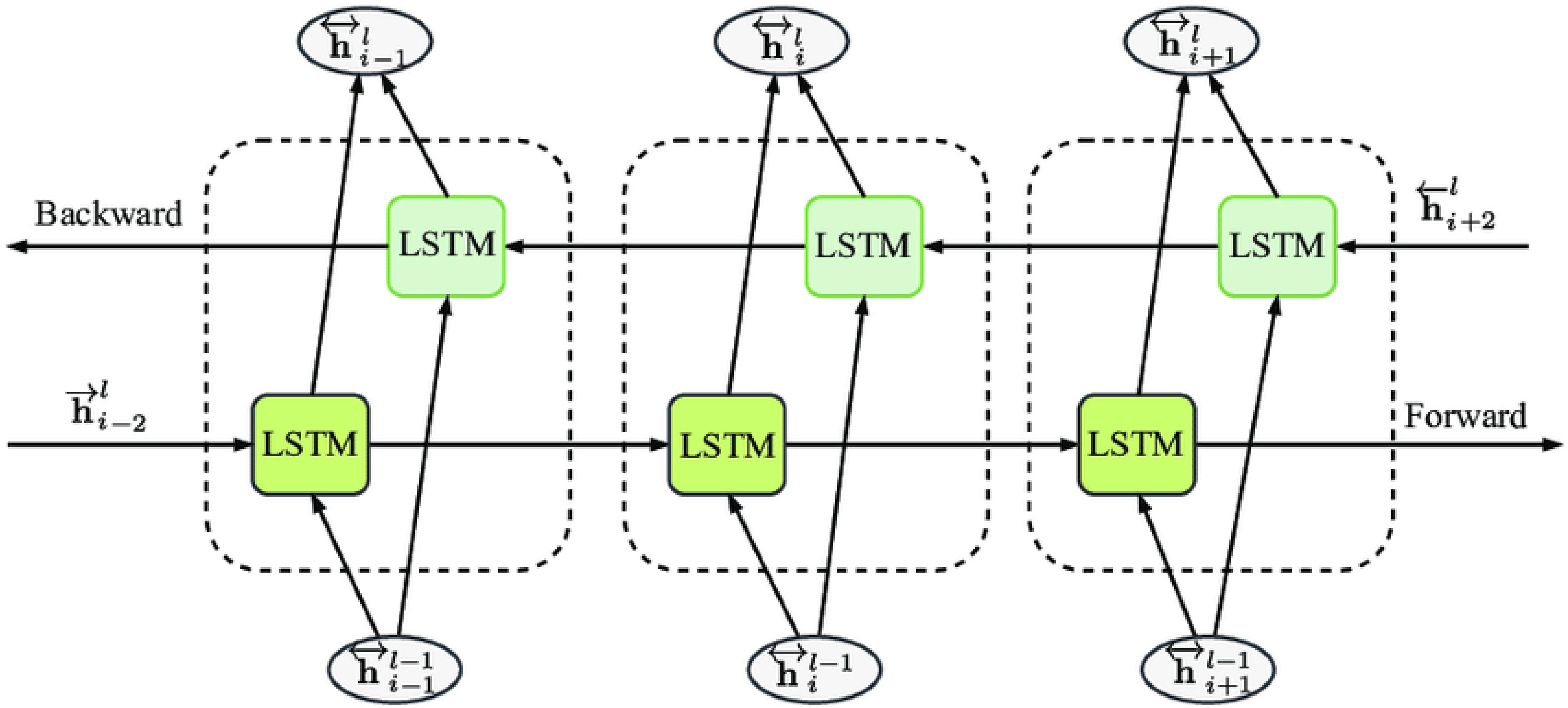
- Gated Recurrent Unit (GRU)
  - The GRU is like a long short-term memory (LSTM) with a forget gate, but has fewer parameters than LSTM, as it lacks an output gate.
  - GRU's performance on certain tasks of polyphonic music modeling, speech signal modeling and natural language processing was found to be similar to that of LSTM.
  - GRUs have been shown to exhibit better performance on certain smaller and less frequent datasets.



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

Variations of LSTM Architectures

- Bidirectional long short term memory (bi-lstm)
  - processes the data in both forward and backward direction.



Conclusions

- Machine Learning for Data Science plays vital role in daily life

- LSTM are essential NN architecture for sequential data
- LSTM and GRU and derivatives are able to learn a lot of longer term information!
- RNN are not hardware friendly
- LSTMs are **prone to overfitting** and it is difficult to apply the dropout algorithm to curb this issue.

Queries

• Thanks

- Puneet Kumar Jain
  jainp@nitrkl.ac.in