Intel Nervana™ AI Academy

RECURRENT NEURAL NETWORKS (RNNS)

ISSUE: VARIABLE LENGTH SEQUENCES OF WORDS

- With images, we forced them into a specific input dimension
- Not obvious how to do this with text
- For example, classify tweets as positive, negative, or neutral
- Tweets can have a variable number of words
- What to do?

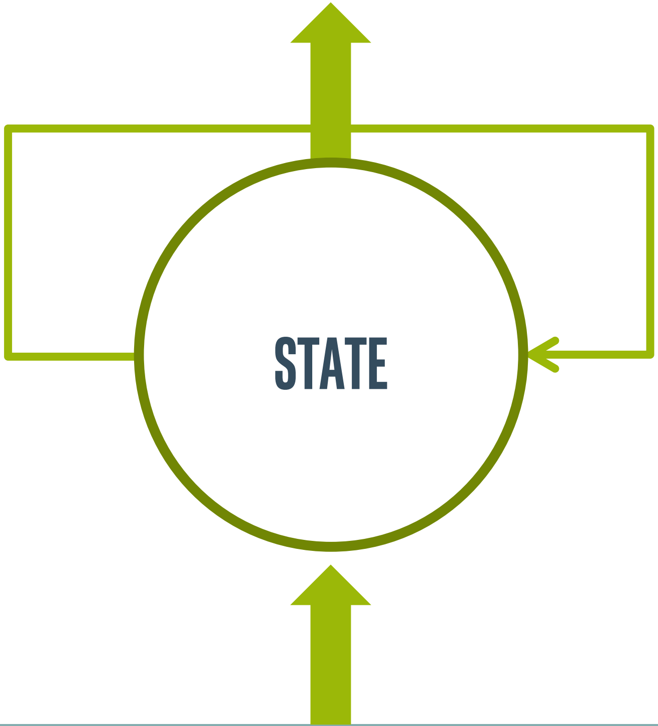
ISSUE: ORDERING OF WORDS IS IMPORTANT

- Want to do better than “bag of words” implementations
- Ideally, each word is processed or understood in the appropriate context
- Need to have some notion of “context”
- Words should be handled differently depending on “context”
- Also, each word should update the context

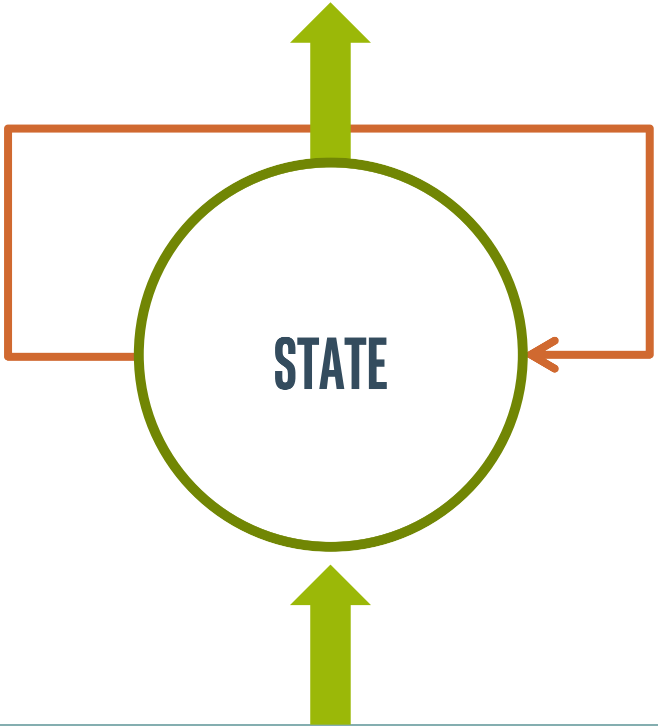
IDEA: USE THE NOTION OF "RECURRENCE"

- Input words one by one
- Network outputs two things:
  - Prediction: What would be the prediction if the sequence ended with that word
  - State: Summary of everything that happened in the past
- This way, can handle variable lengths of text
- The response to a word depends on the words that preceded it

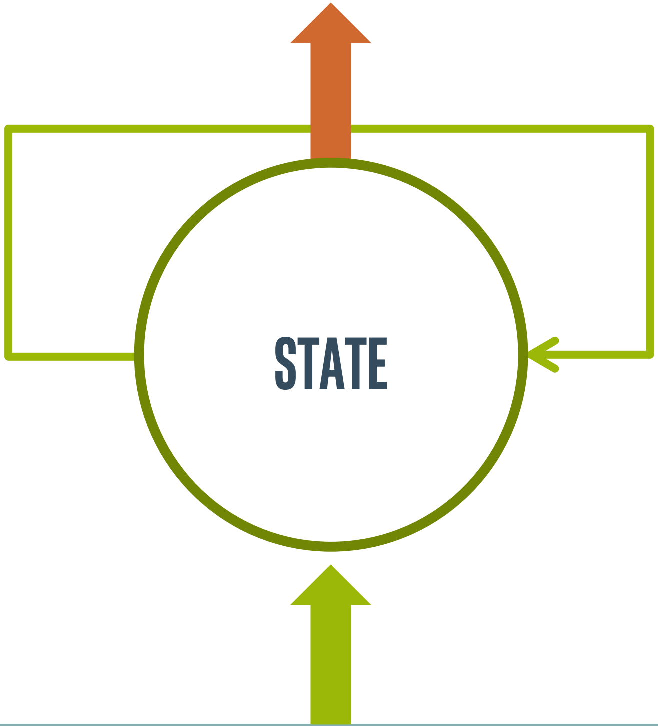
IDEA: USE THE NOTION OF "RECURRENCE"



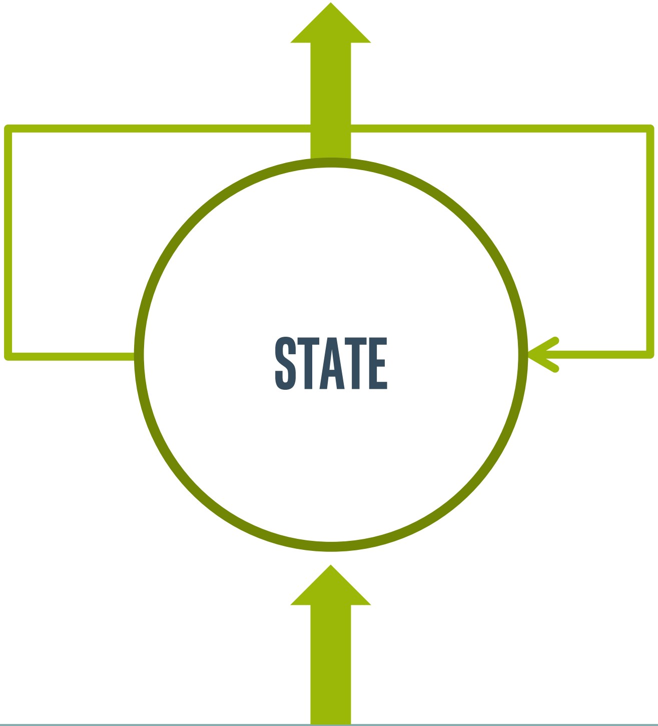
IDEA: USE THE NOTION OF "RECURRENCE"



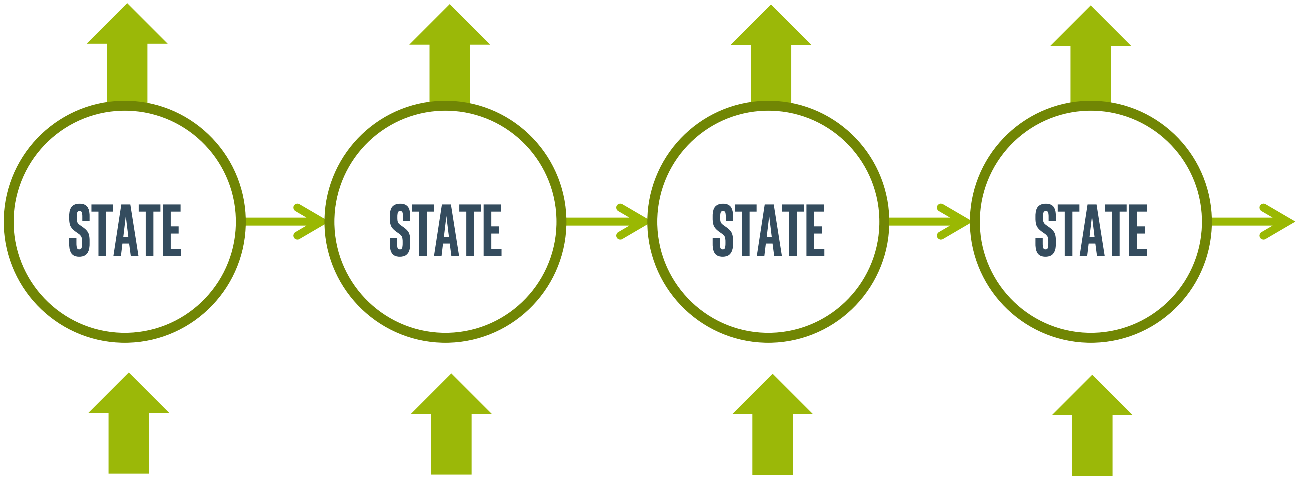
IDEA: USE THE NOTION OF "RECURRENCE"



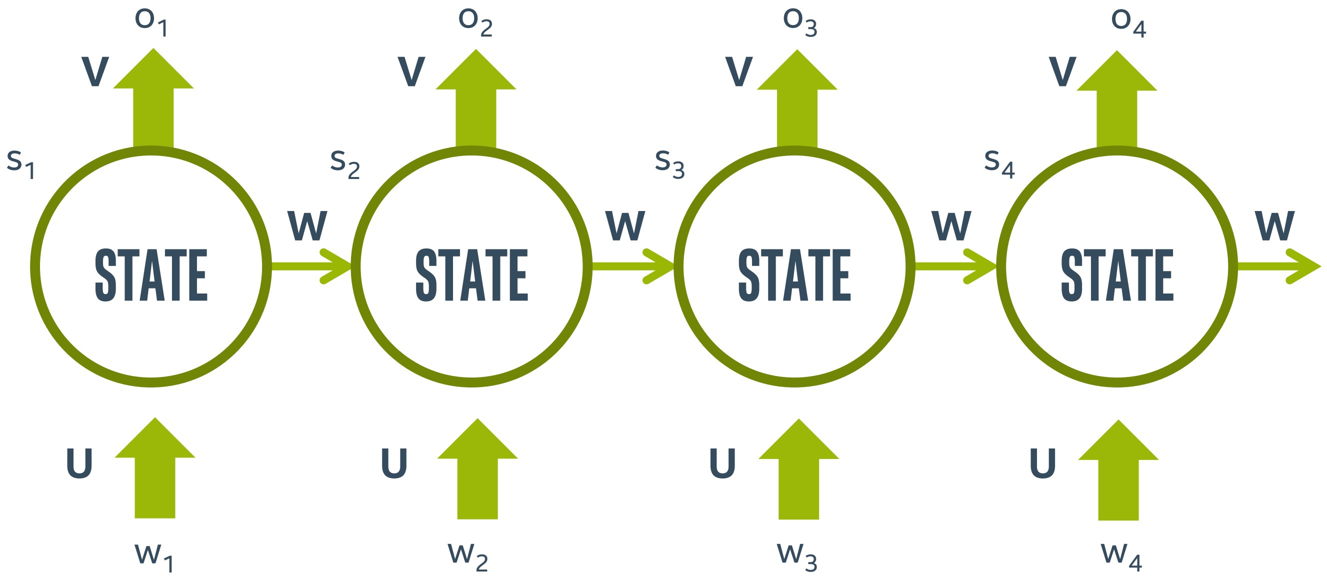
IDEA: USE THE NOTION OF “RECURRENCE”



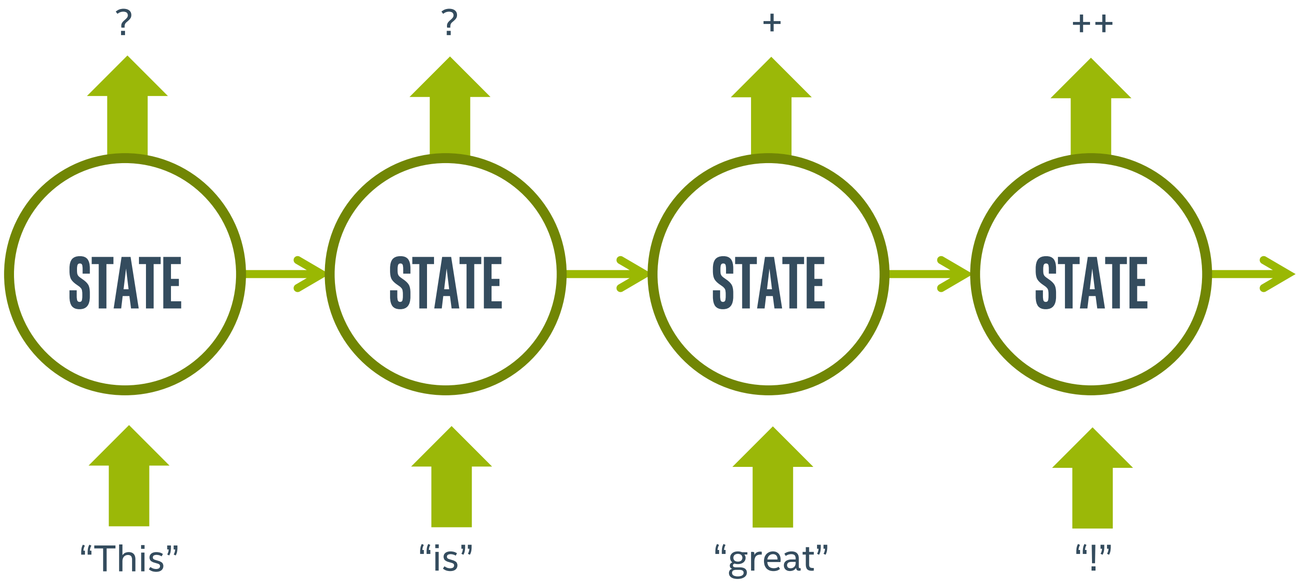
“UNROLLING” THE RNN



"UNROLLING" THE RNN

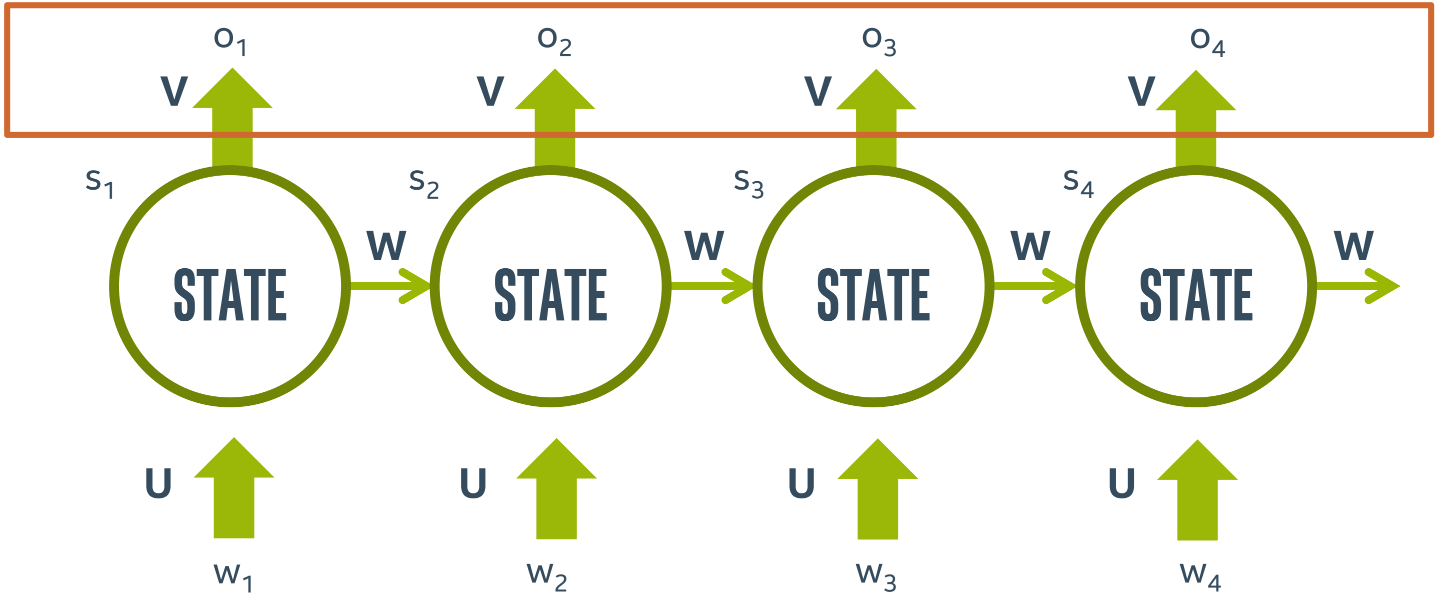


"UNROLLING" THE RNN



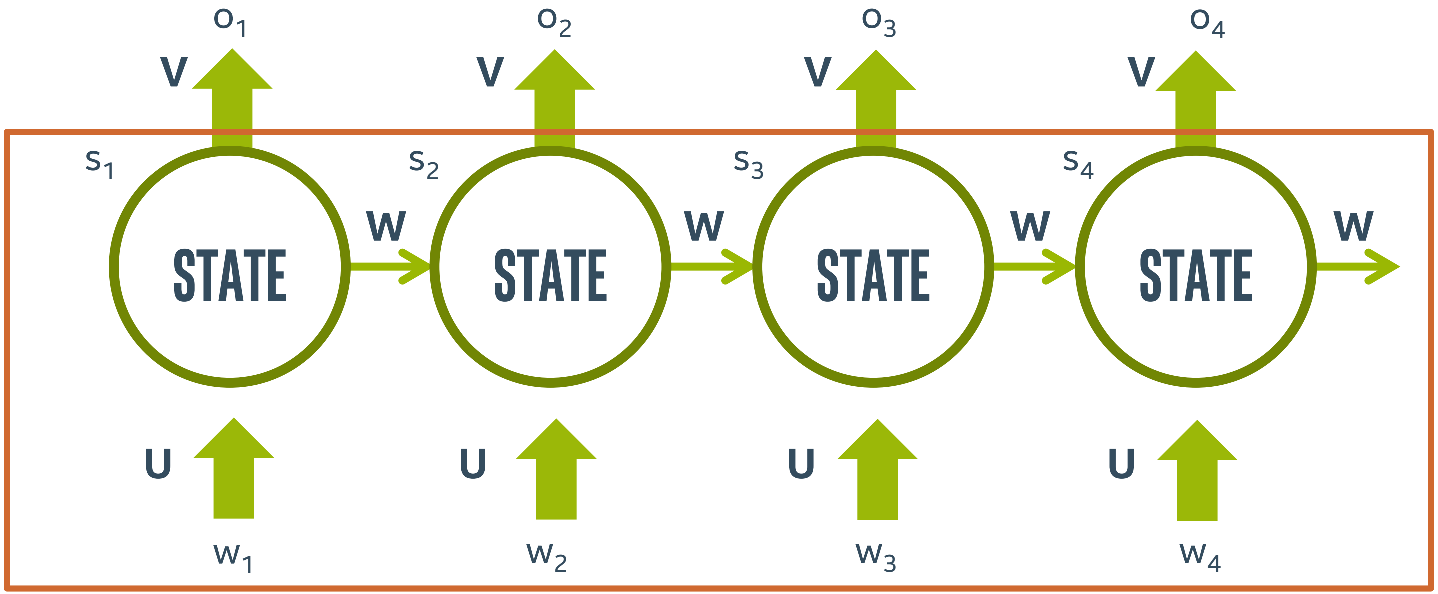
"UNROLLING" THE RNN

In Keras, this part is accomplished by a subsequent Dense layer.



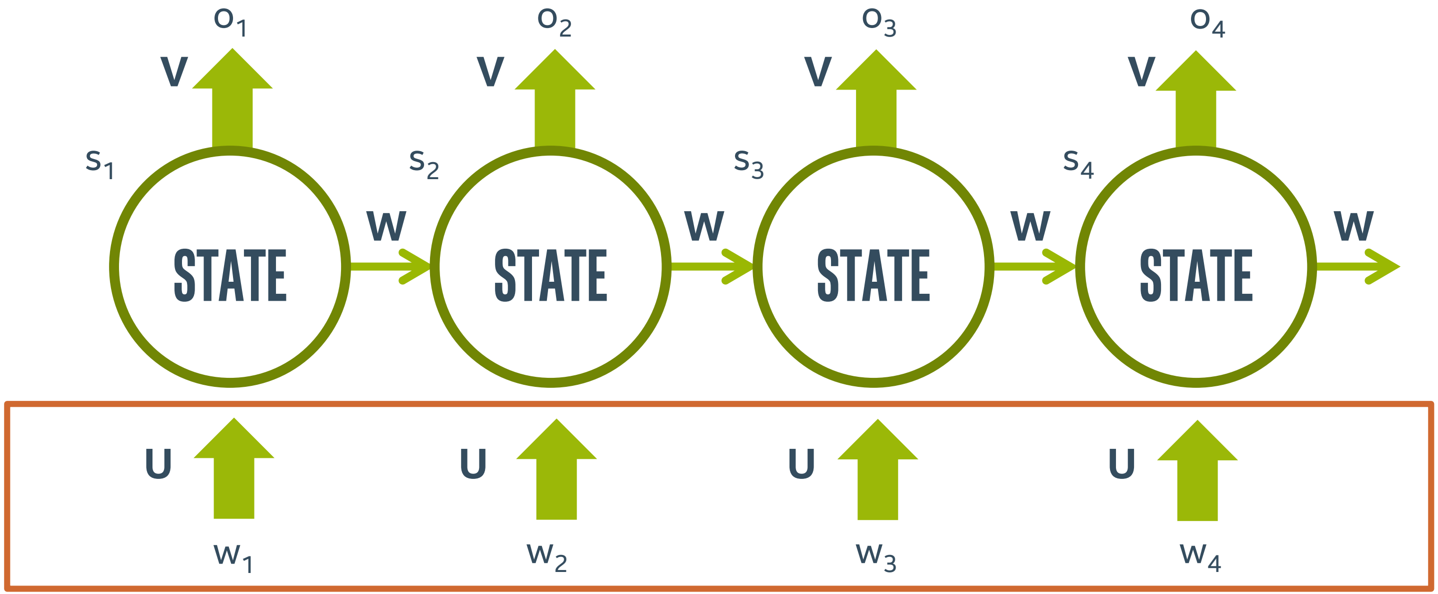
“UNROLLING” THE RNN

This part is the core RNN.



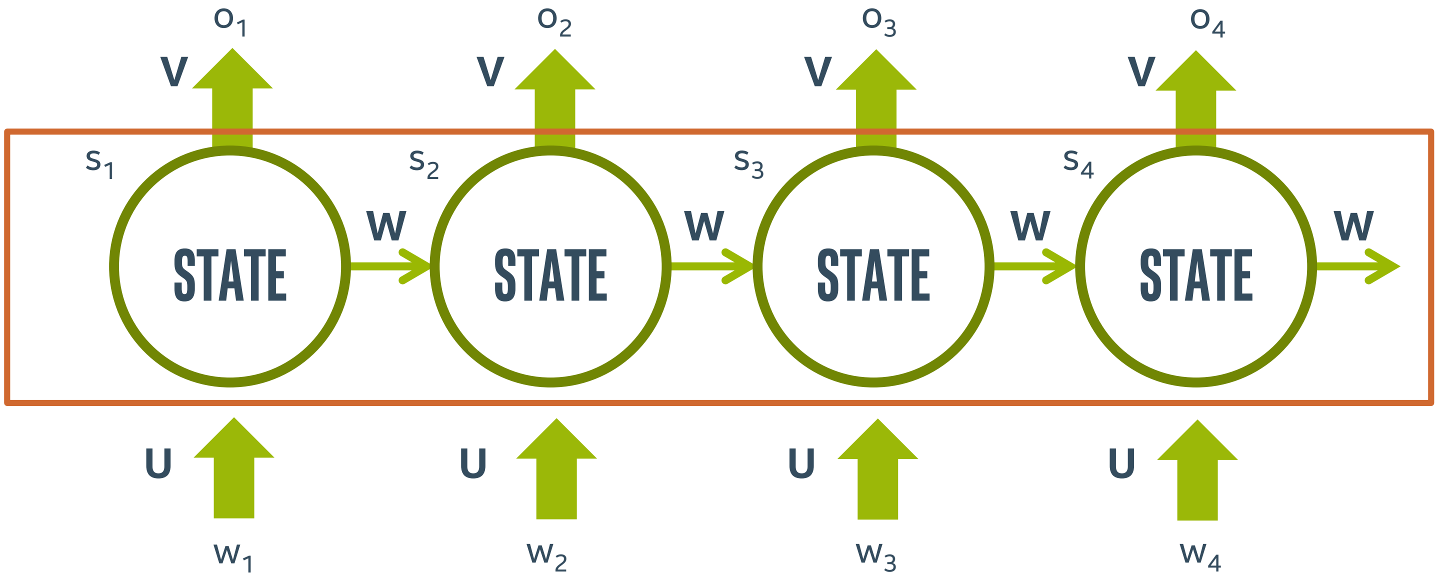
“UNROLLING” THE RNN

Keras calls this part the “kernel” (e.g. `kernel_initializer`,...).



"UNROLLING" THE RNN

Keras calls this part "recurrent" (`recurrent_initializer,...`).



PRACTICAL DETAILS

- Often, we train on just the ”final” output and ignore the intermediate outputs
- Slight variation called Backpropagation Through Time (BPTT) is used to train RNNs
- Sensitive to length of sequence (due to “vanishing/exploding gradient” problem)
- In practice, we still set a maximum length to our sequences
  - If input is shorter than maximum, we “pad” it
  - If input is longer than maximum, we truncate

OTHER USES OF RNNS

- We have focused on text/words as application
- But, RNNs can be used for other sequential data
  - Time-Series Data
  - Speech Recognition
  - Sensor Data
  - Genome Sequences

WEAKNESSES OF RNNS

- Nature of state transition means it is hard to keep information from distant past in current memory without reinforcement
- In the next lecture, we will introduce LSTMs, which have a more complex mechanism for updated the state

Nervana™ AI Academy

Intel Nervana™ AI Academy

LSTM (LONG-SHORT TERM MEMORY) RNNS

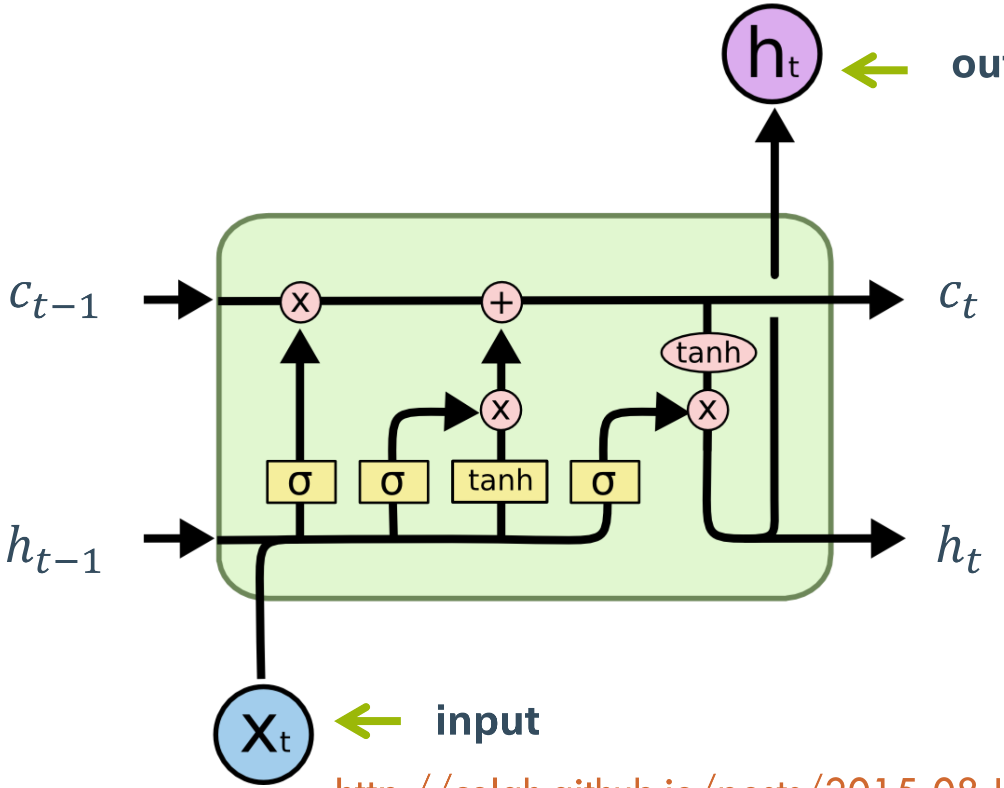
ISSUE: STANDARD RNNS HAVE POOR MEMORY

- Transition Matrix necessarily weakens signal
- Need a structure that can leave some dimensions unchanged over many steps
- This is the problem addressed by so-called Long-Short Term Memory RNNs (LSTM)

IDEA: MAKE "REMEMBERING" EASY

- Define a more complicated update mechanism for the changing of the internal state
- By default, LSTMs remember the information from the last step
- Items are overwritten as an active choice

LSTM DIAGRAM



output

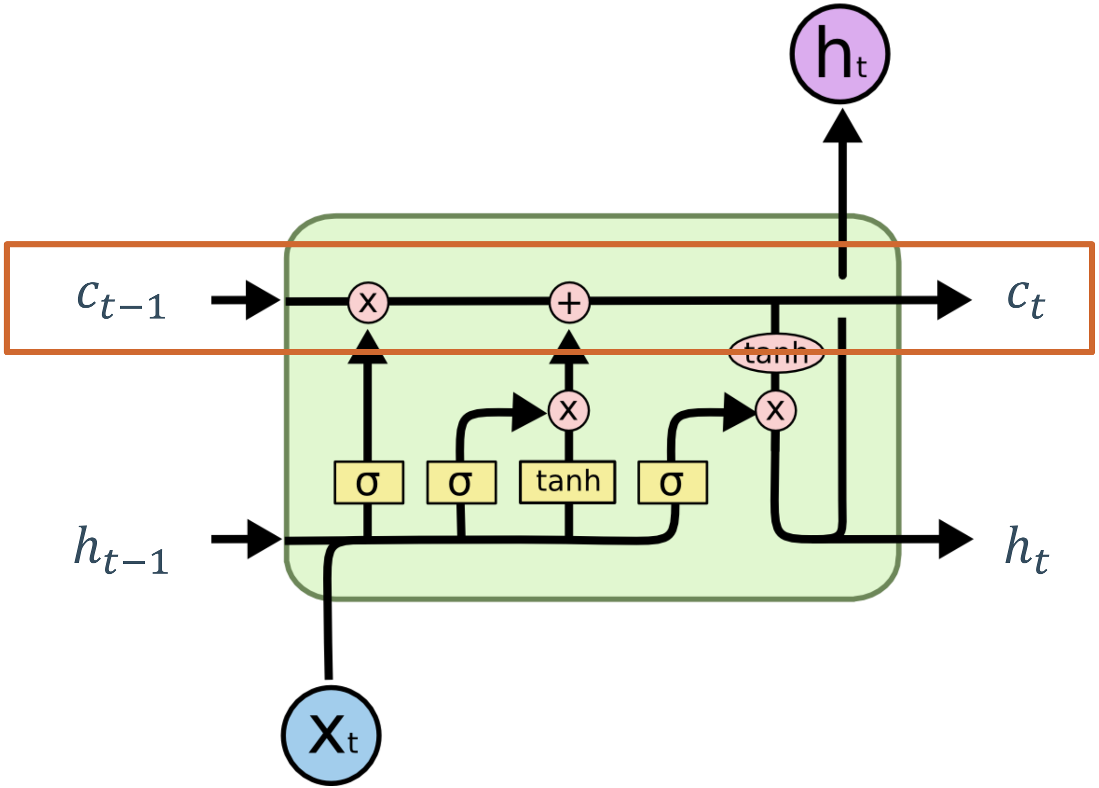
cell state

output (which also feeds into next unit)

input

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

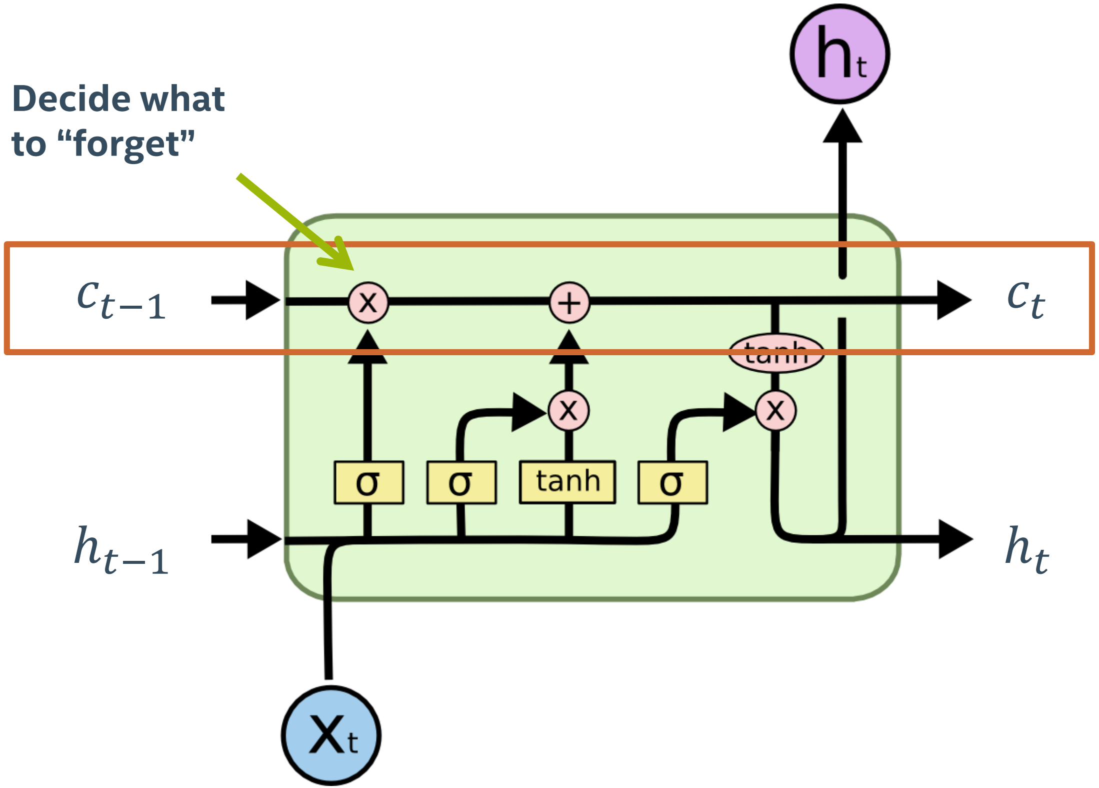
LSTM DIAGRAM



cell state gets updated in two stages

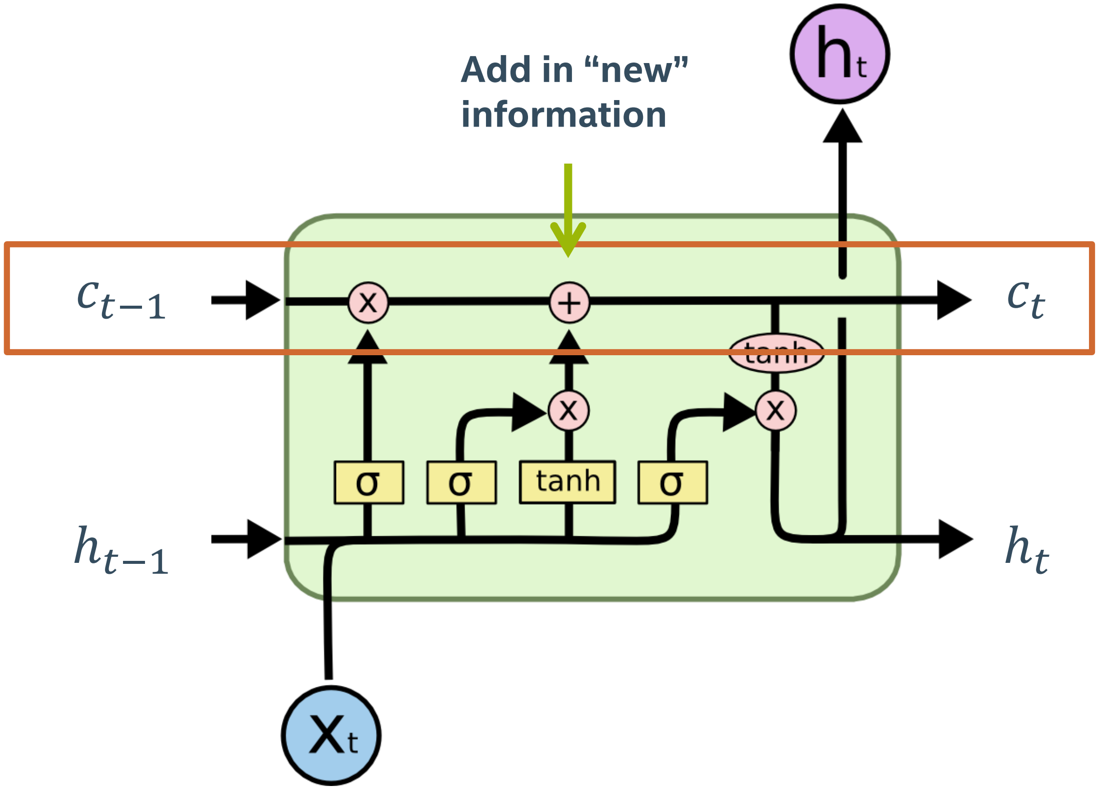
LSTM DIAGRAM

Decide what
to "forget"



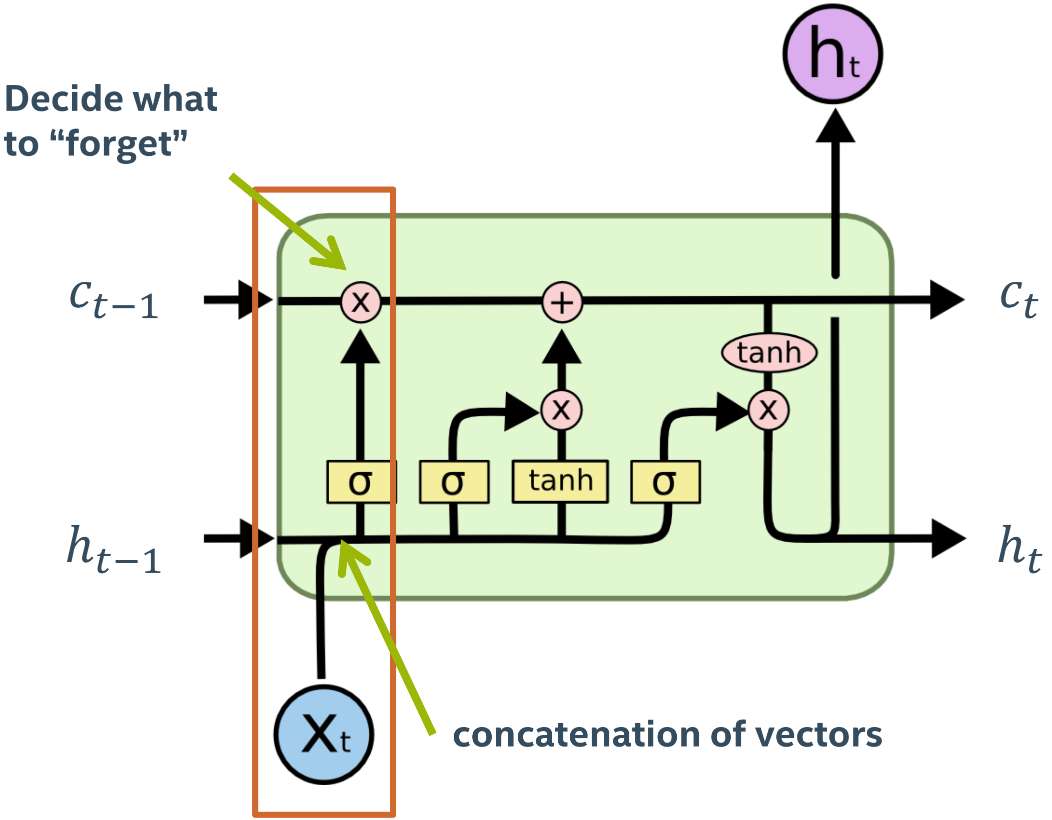
cell state gets updated
in two stages

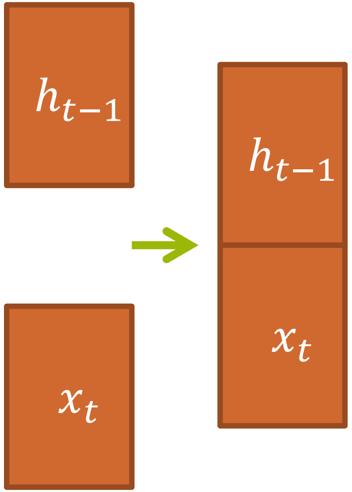
LSTM DIAGRAM



cell state gets updated in two stages

LSTM DIAGRAM

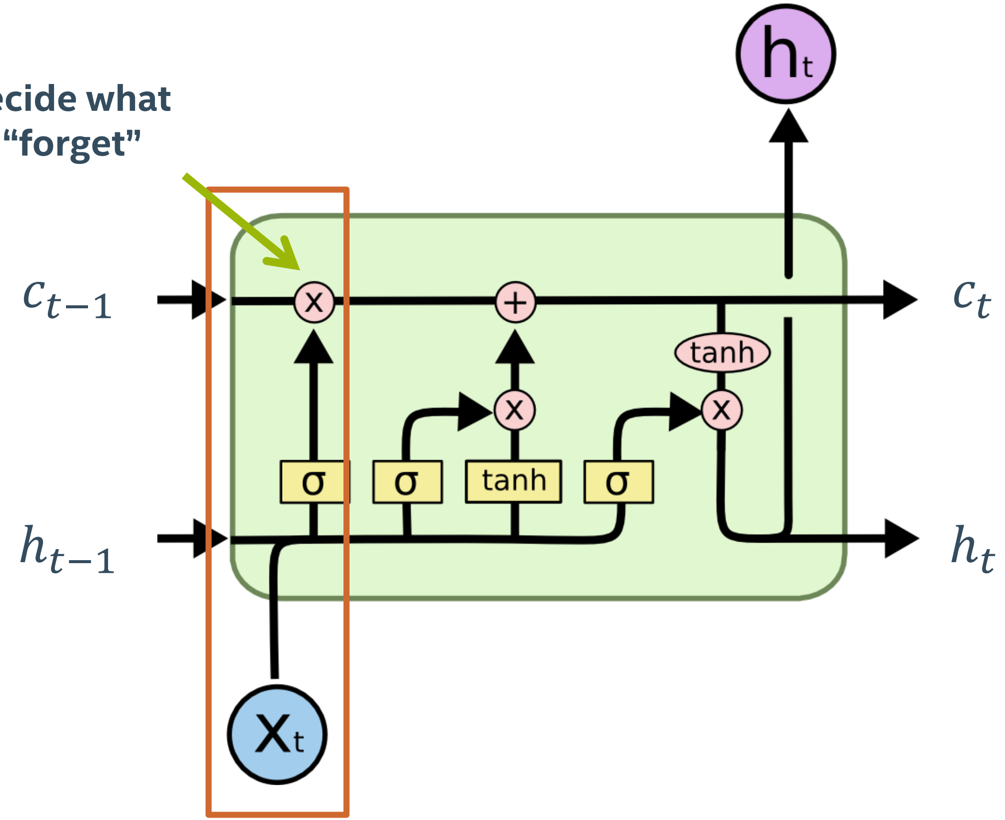




$[h_{t-1}, x_t]$

LSTM DIAGRAM

Decide what to “forget”

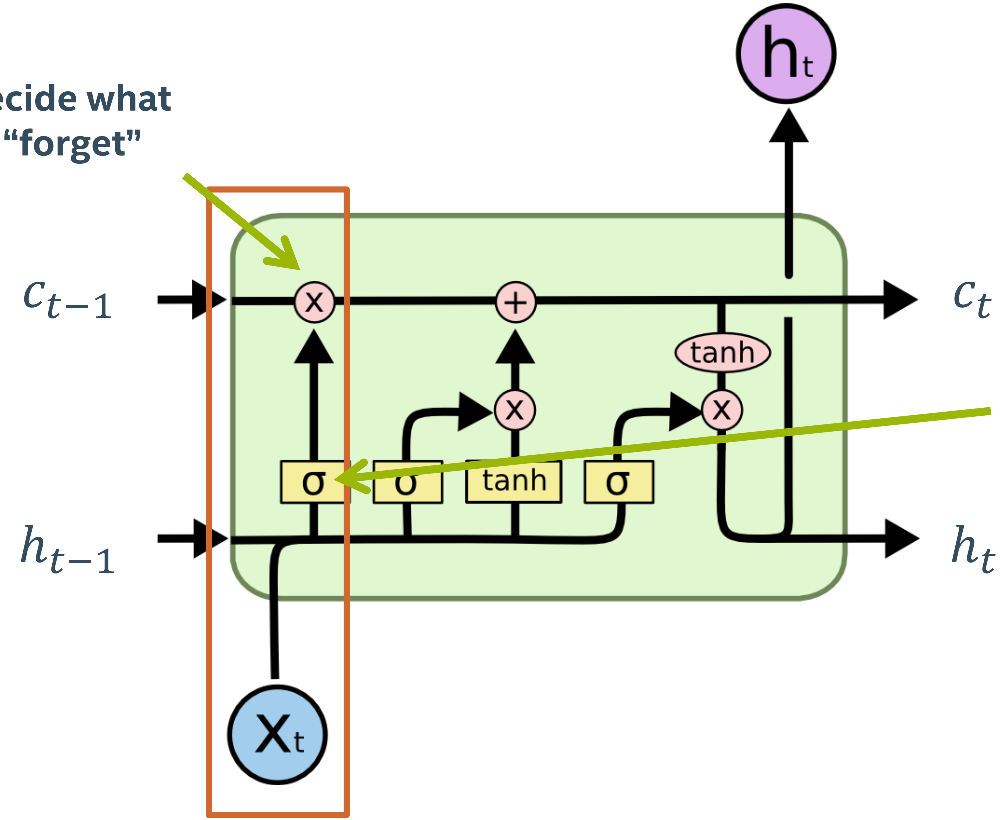


$$f_t = \sigma\left(W_f[h_{t-1}, x_t] + b_f\right)$$

based on previous output and current input

LSTM DIAGRAM

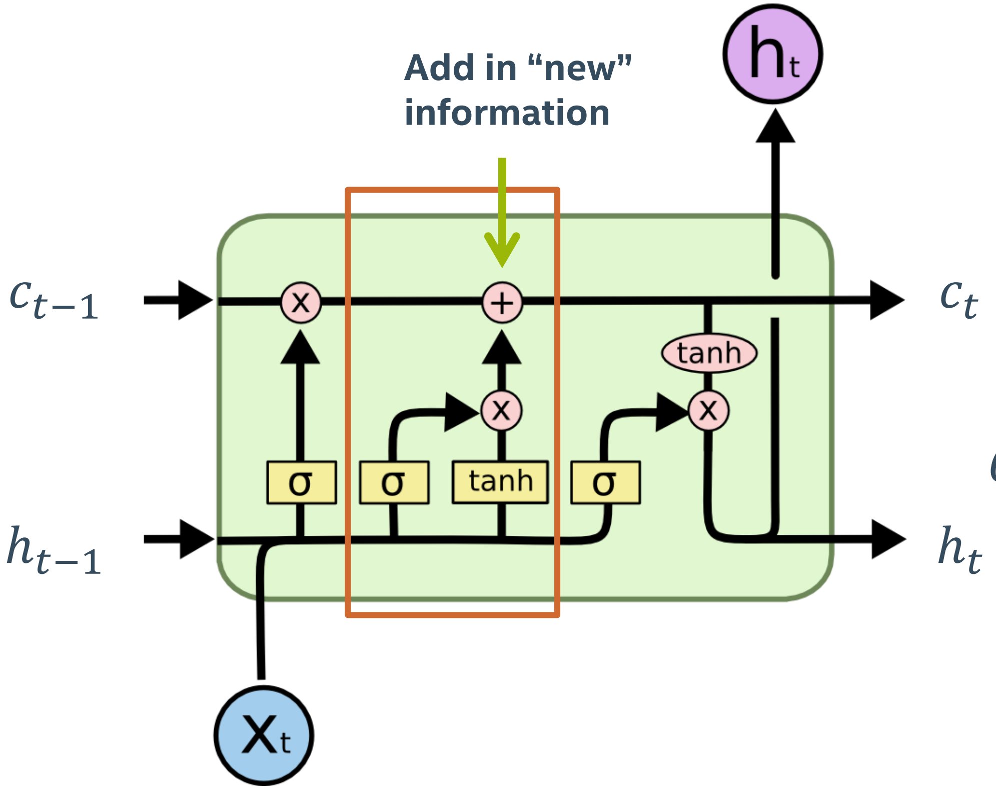
Decide what to "forget"



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

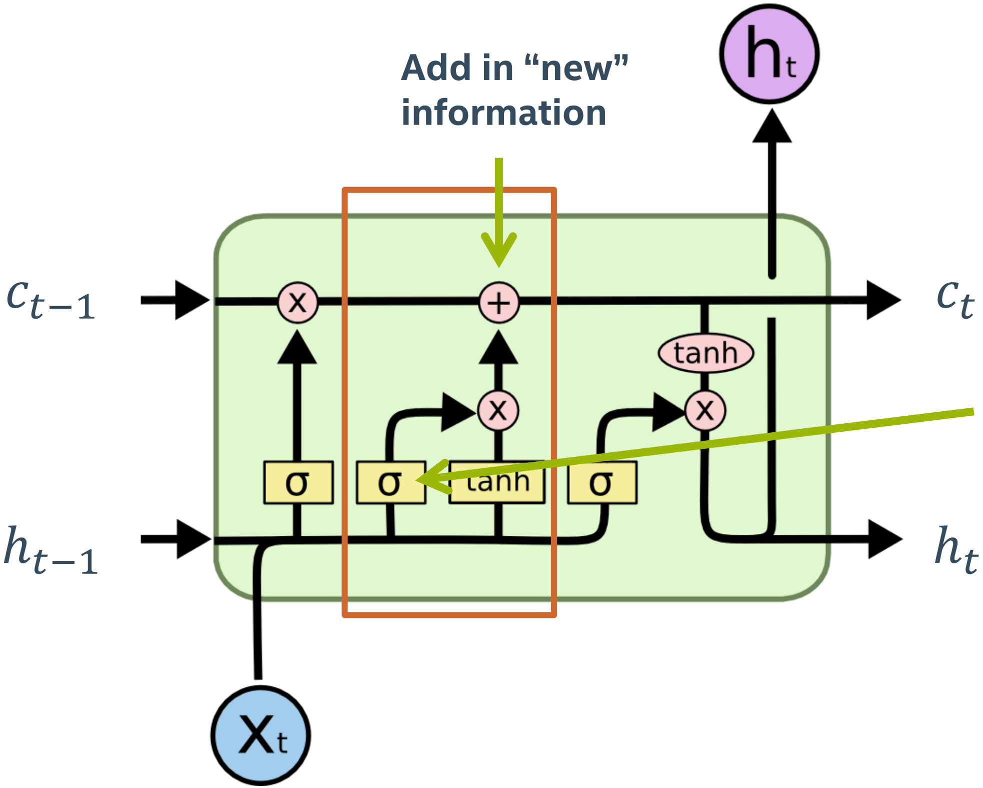
based on previous output and current input

LSTM DIAGRAM



$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$
$$C_t' = tanh(W_C[h_{t-1}, x_t] + b_C)$$

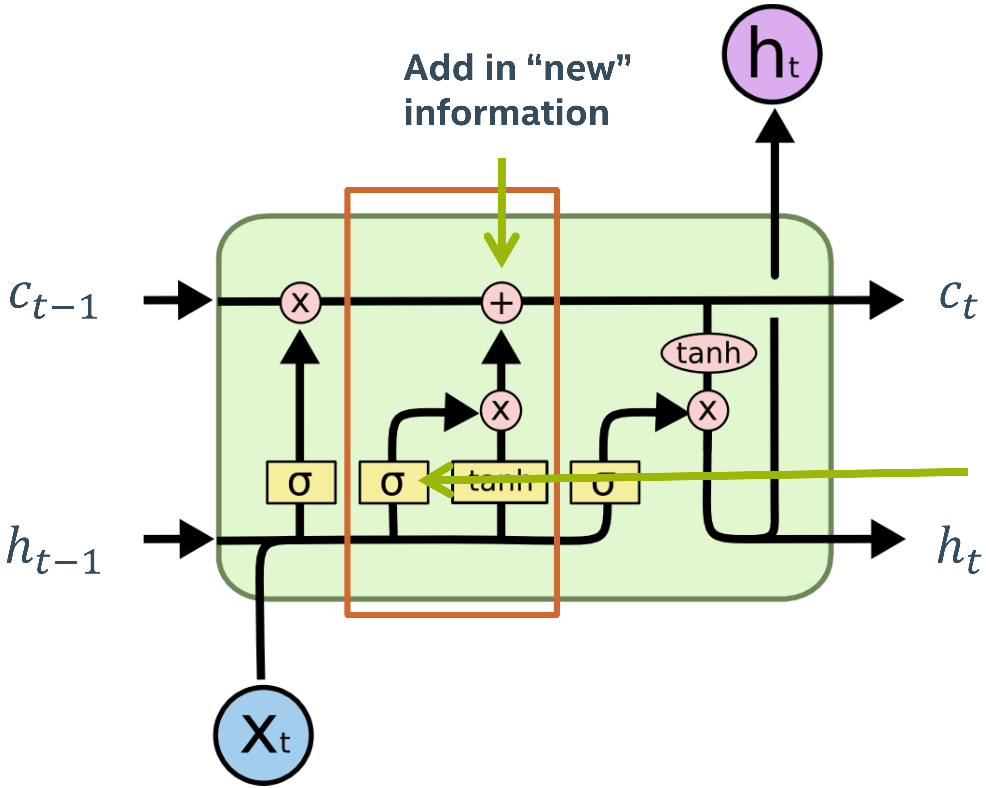
LSTM DIAGRAM



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

$$C_t' = tanh(W_C[h_{t-1}, x_t] + b_C)$$

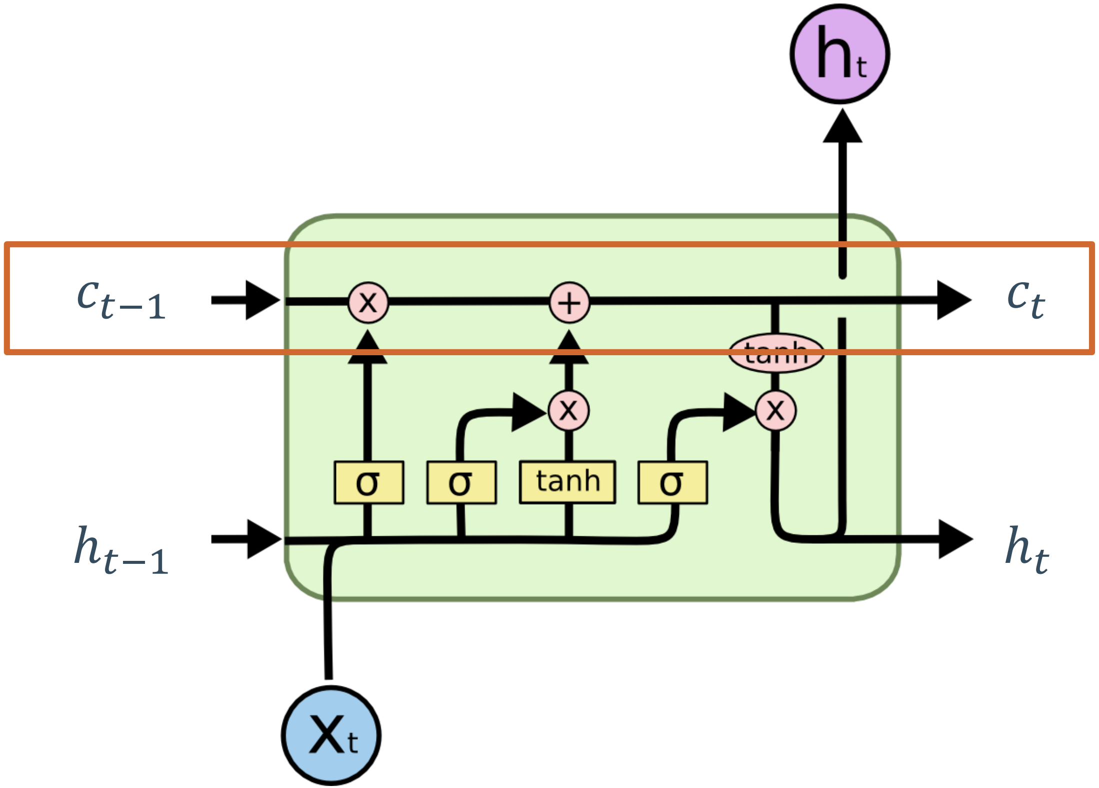
LSTM DIAGRAM



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

$$C_t' = tanh(W_C[h_{t-1}, x_t] + b_C)$$

LSTM DIAGRAM



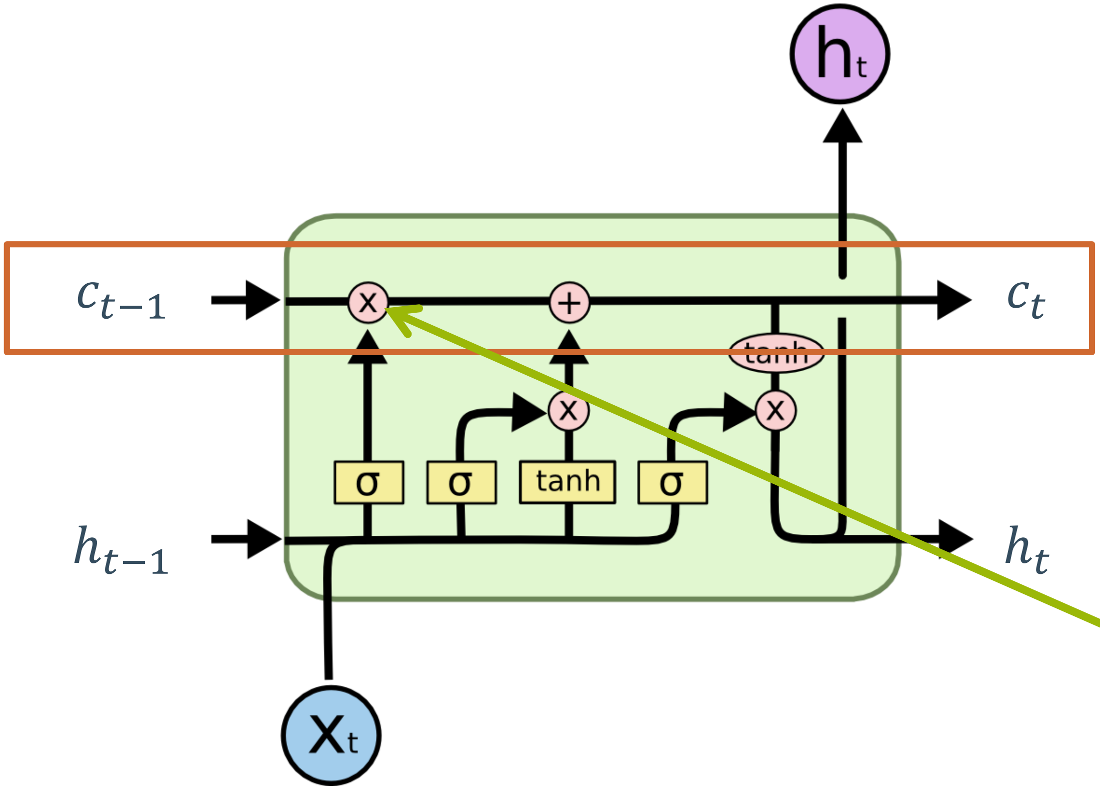
Note: '∗' represents element-wise multiplication

$$C_t = f_i * C_{t-1} + i_t * C_t'$$

forget the old (or not)

add the new (or not)

LSTM DIAGRAM



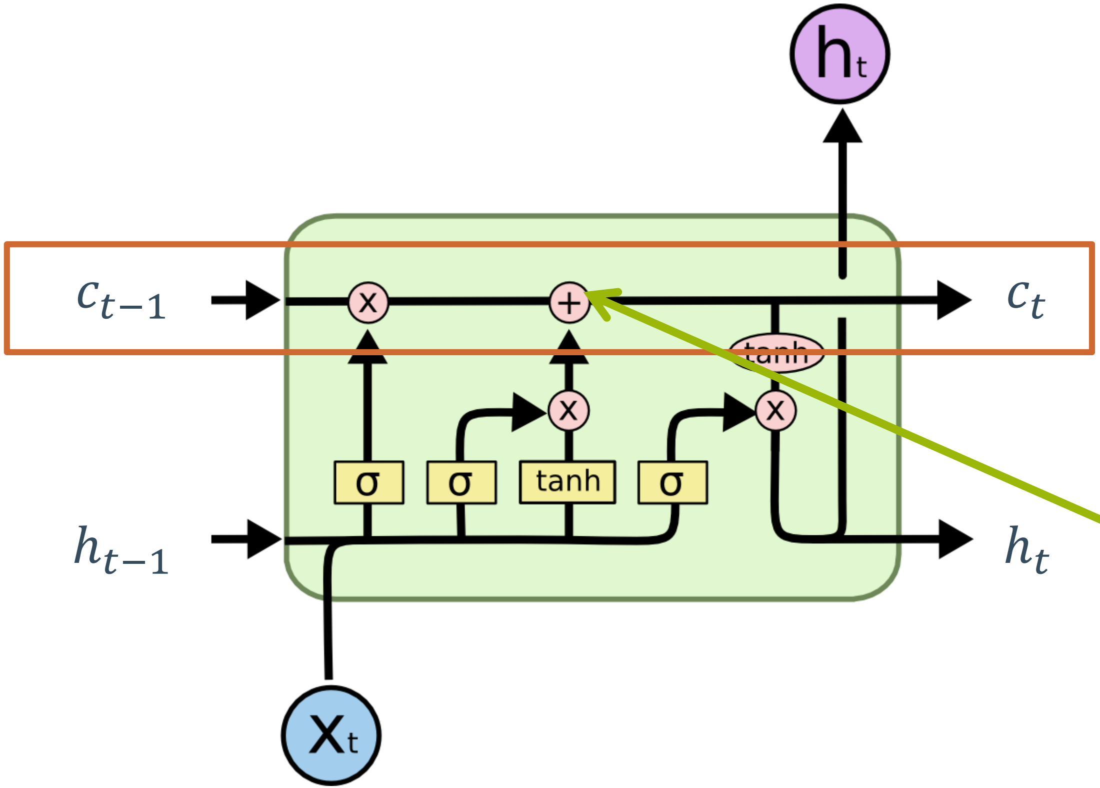
Note: '∗' represents element-wise multiplication

$$C_t = f_i * C_{t-1} + i_t * C'_t$$

forget the old (or not)

add the new (or not)

LSTM DIAGRAM



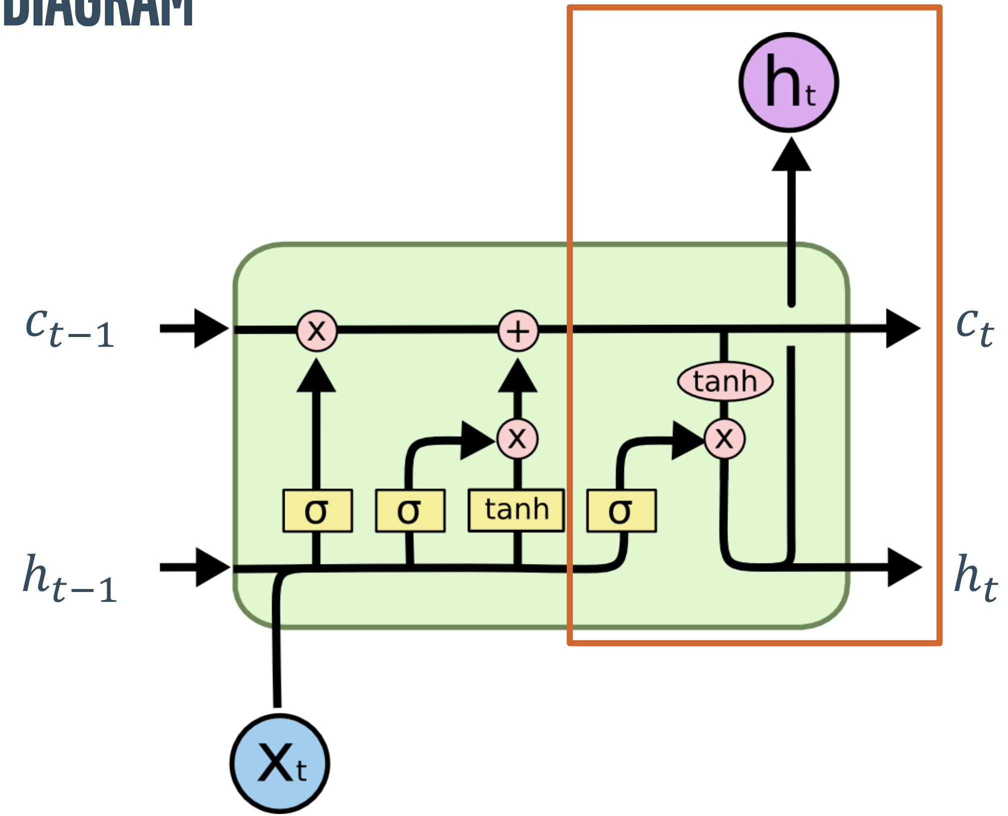
Note: '∗' represents element-wise multiplication

$$C_t = f_i * C_{t-1} + i_t * C_t'$$

forget the old (or not)

add the new (or not)

LSTM DIAGRAM

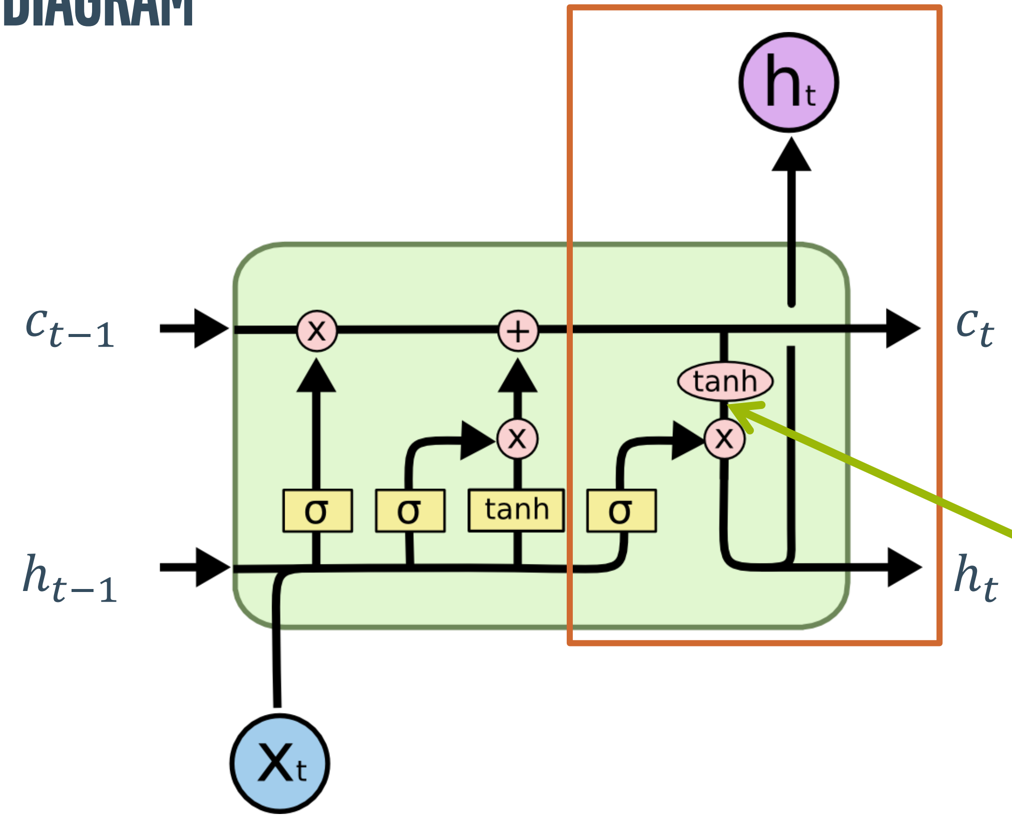


Final stage computes the output

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

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

LSTM DIAGRAM

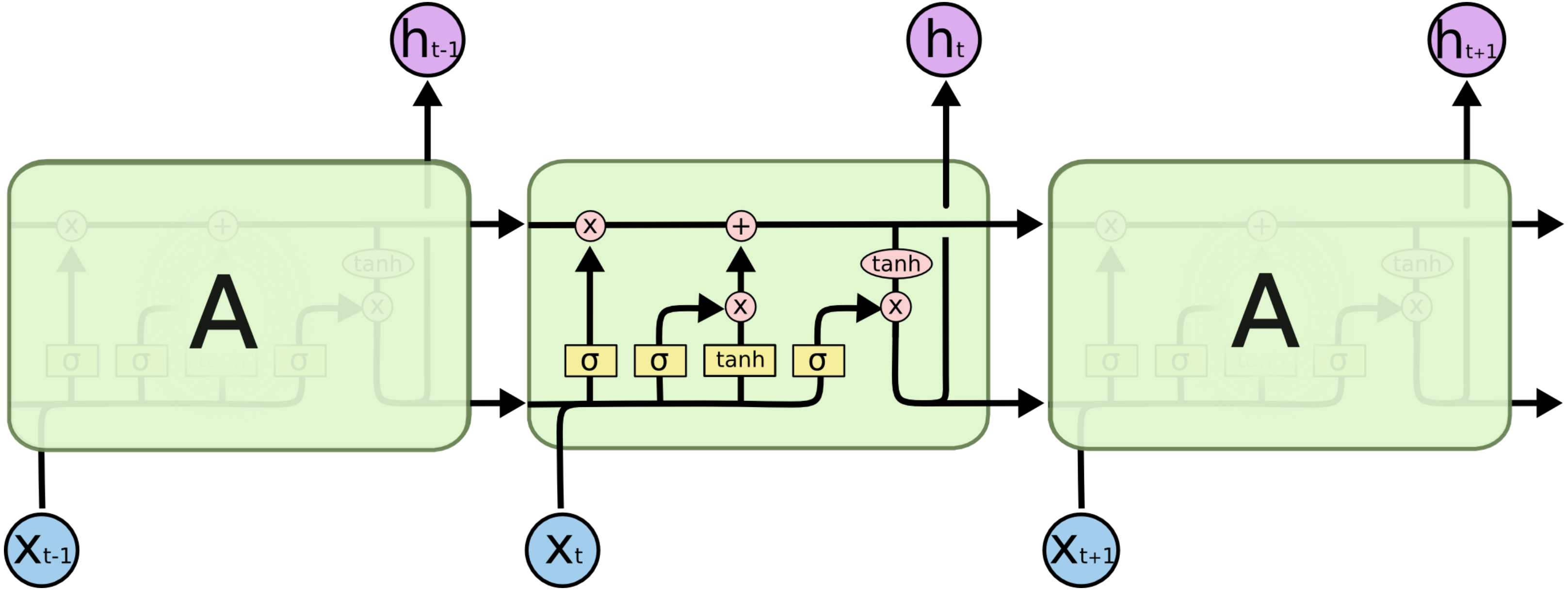


Final stage computes the output

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * \tanh(C_t)$$

Note: No weights here

LSTM UNROLLED



FINAL POINTS

- This is the most common version of LSTM, but there are many different “flavors”
  - Gated Recurrent Unit (GRU)
  - Depth-Gated RNN
- LSTMs have considerably more parameters than plain RNNs
- Most of the big performance improvements in NLP have come from LSTMs, not plain RNN

intel® Nervana™ AI Academy