intel Nervana AI Academy

TEXT / WORD VECTORS

MOTIVATION

- Have shown how to use Neural Networks with structured numerical data
- Images can be upsampled / downsampled to be a certain size
- Image values are numbers (greyscale, RGB)
- But how do we work with text?
- Issue 1: How to deal with pieces of text (sequences of words that vary in length)?
- Issue 2: How to convert words into something numerical?

ISSUE: VARIABLE LENGTH SEQUENCES OF WORDS

- With images, we forced them into a specific input dimension
- Not obvious how to do this with text
- We will use a new structure of network called a “Recurrent Neural Network” which will be discussed next lecture

TOKENIZATION

- Need to convert word into something numerical
- First approach: Tokenization
- Treat as a categorical variable with huge number of categories (one hot encoding)
- Deal with some details around casing, punctuation, etc.

“The cat in the hat.”

↓

[‘the’,’cat’,’in’,’the’,’hat’,’<EOS>’]

TOKENIZATION

- Use tokens to build a vocabulary
- Vocabulary is a one-to-one mapping from index # to a token
- Usually represented by a list and a dictionary

index → word

```
[
  '<EOS>',
  'the',
  'cat',
  'in',
  'hat',
  '.'
]
```

index → word

```
{
  '<EOS>': 0,
  'the':   1,
  'cat':   2,
  'in':    3,
  'hat':   4,
  '.':  5
}
```

ISSUES WITH TOKENIZATION

- Tokenization loses a lot of information about words:
  - Part of speech
  - Synonymy (distinct words with same or similar meaning)
  - Polysemy (single word with multiple meanings)
  - General context in which word is likely to appear
    (e.g. “unemployment” and “inflation”) are both about economics
- Increasing vocabulary size is difficult (would require re-training the model)
- Vector length is huge -> large number of weights
- Yet information in vector is very sparse

WORD VECTORS

- Goal: represent a word by an m-dimensional vector (for medium-sized m, say, m=300)
- Have “similar” words be represented by “nearby” vectors in this m-dimensional space
- Words in a particular domain (economics, science, sports) could be closer to one another than words in other domains.
- Could help with synonymy
  - e.g. “big” and ”large” have nearby vectors
- Could help with polysemy
  - “Java” and ”Indonesia” could be close in some dimensions
  - “Java” and “Python” are close in other dimensions

WORD VECTORS

- Vectors would be shorter length and information-dense, rather than very long and information-sparse
- Would require fewer weights and parameters
- Fortunately, there are existing mappings which can be downloaded and used
- These were trained on big corpora for a long time
- Let’s understand how they were developed and trained

WHAT MAKES TWO WORDS SIMILAR?

- Idea: similar words occur in similar contexts
- For a given word, look at the words in a “window” around it
- Consider trying to predict a word given the context
- This is exactly the CBOW (continuous bag of words) model

“We hold these truths to be self-evident, that all men are created equal”

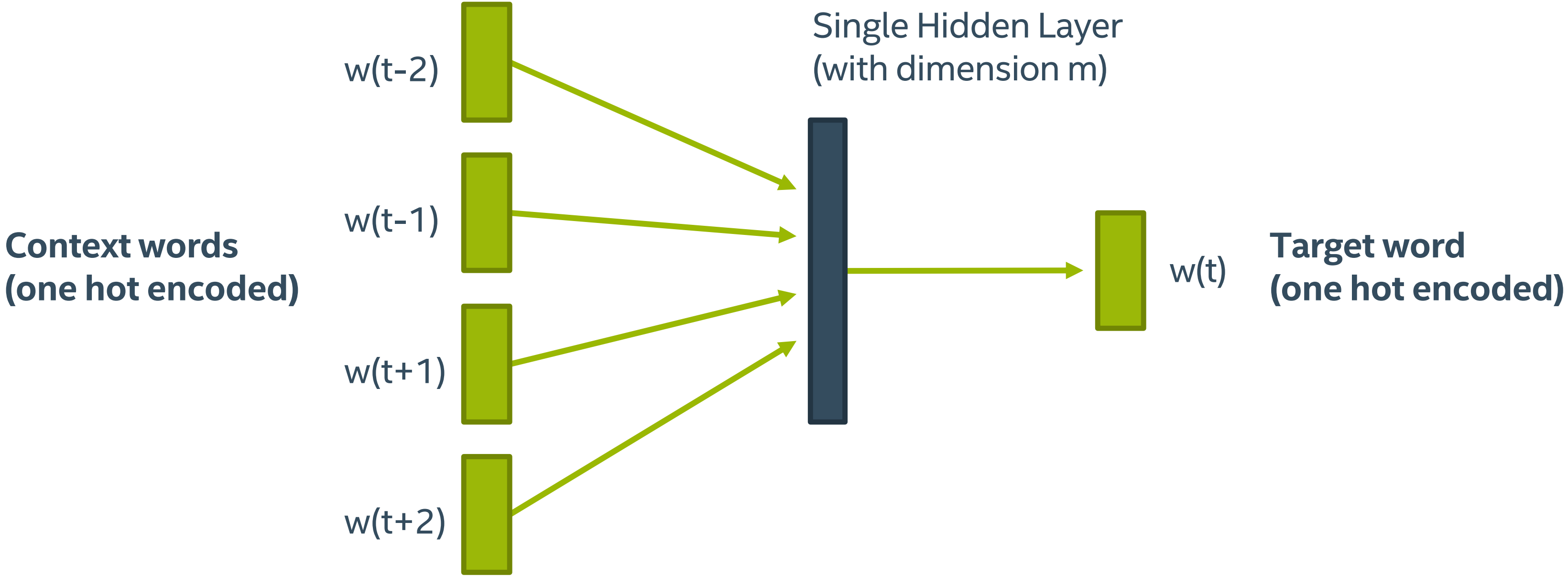
```
([‘truths’, ‘to’, ‘be’, ‘that’, ’all’, ‘men’], ‘self-evident’)
```

context

target word

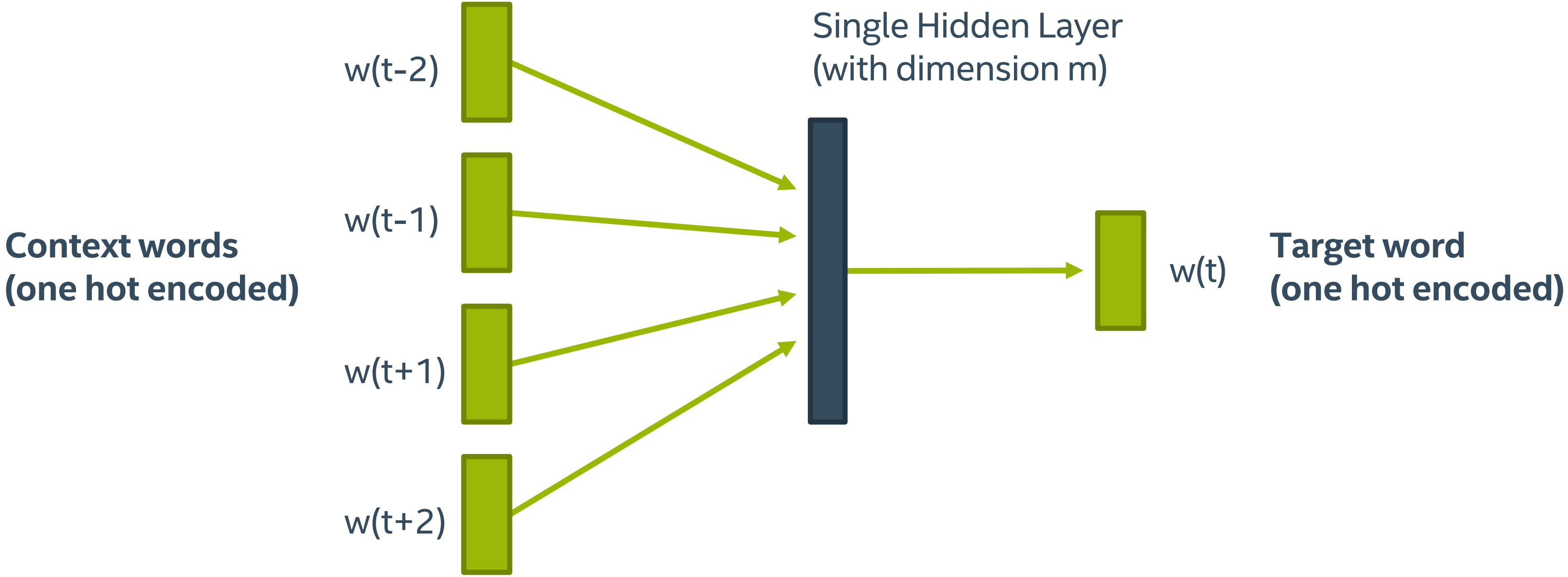
CBOW MODEL

Train a neural network on a large corpus of data.



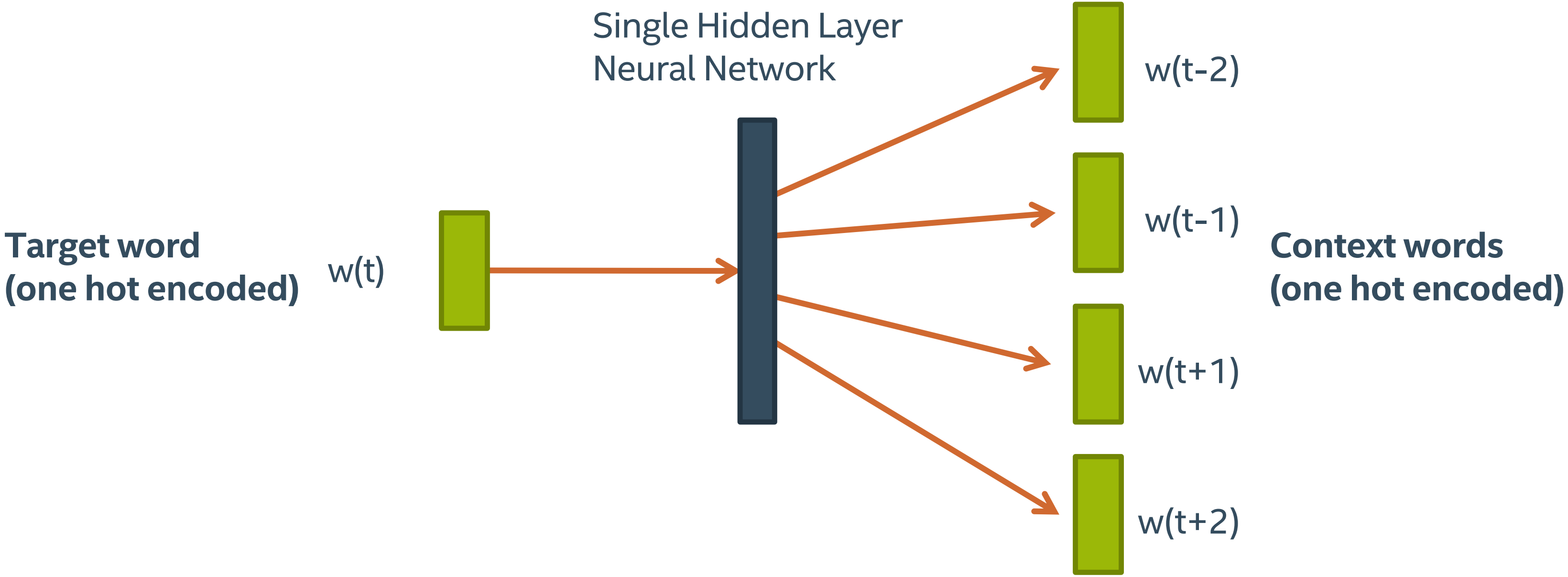
CBOW MODEL

Once the network is trained, weights -> word vectors.



SKIP-GRAM MODEL

Same idea, except we predict the context from the target.



WORD2VEC

- *Distributed Representations of Words and Phrases and Their Compositionality*—Mikolov et al.
- Uses a Skip-gram model to train on a large corpus
- Lots of details to make it work better
  - Aggregation of multi-word phrases (e.g. Boston Globe)
  - Subsampling (i.e. oversample less common words)
  - Negative Sampling (give network examples of wrong words)

GLOVE

- Global Vectors for Word Representation (GloVe)
- Use co-occurrence matrix with neighboring words to determine similarity

$$J = \frac{1}{2} \sum_{i,j=1}^{W} f(P_{ij})\left(u_i^T v_j - \log(P_{ij})\right)^2$$

f → **frequency of a word, with a maximum cap**

P_ij → **probability words i and j occur together**

GLOVE

- GloVe is publicly available
- Developed at Stanford: https://nlp.stanford.edu/projects/glove/
- Trained on huge corpora

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?

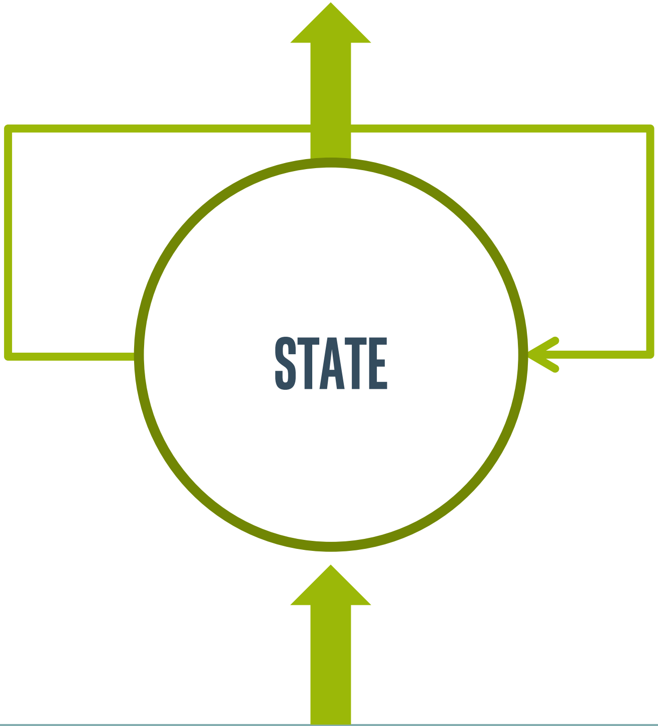
ISSUE: ORDERING OF WORDS IS IMPORTANT

- 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

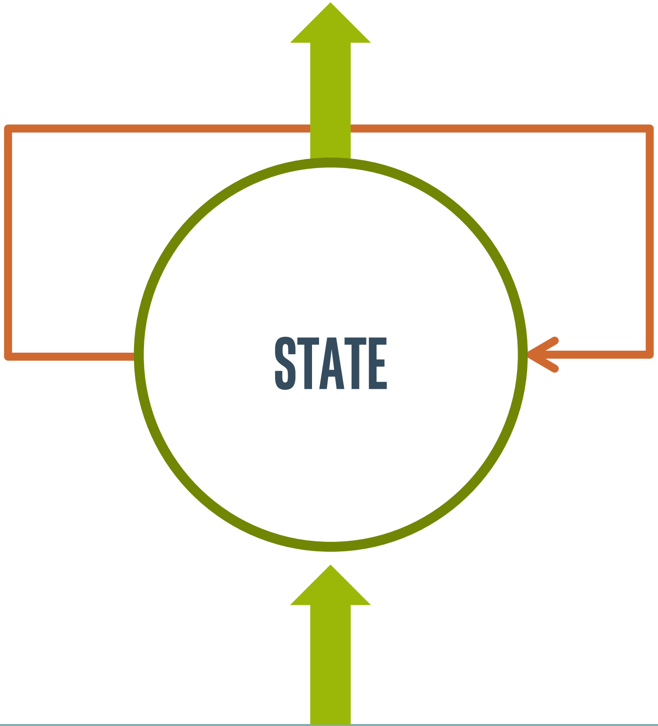
IDEA: USE THE NOTION OF "RECURRENCE"

- 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

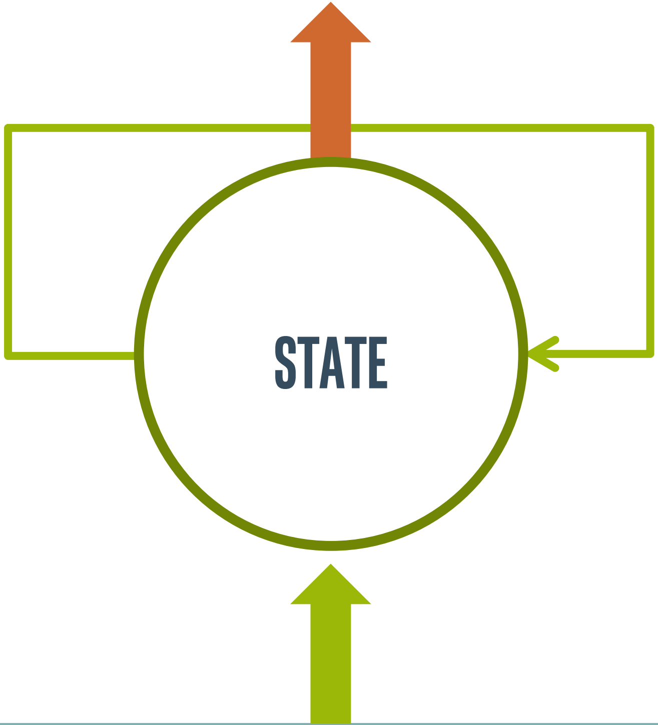
IDEA: USE THE NOTION OF "RECURRENCE"



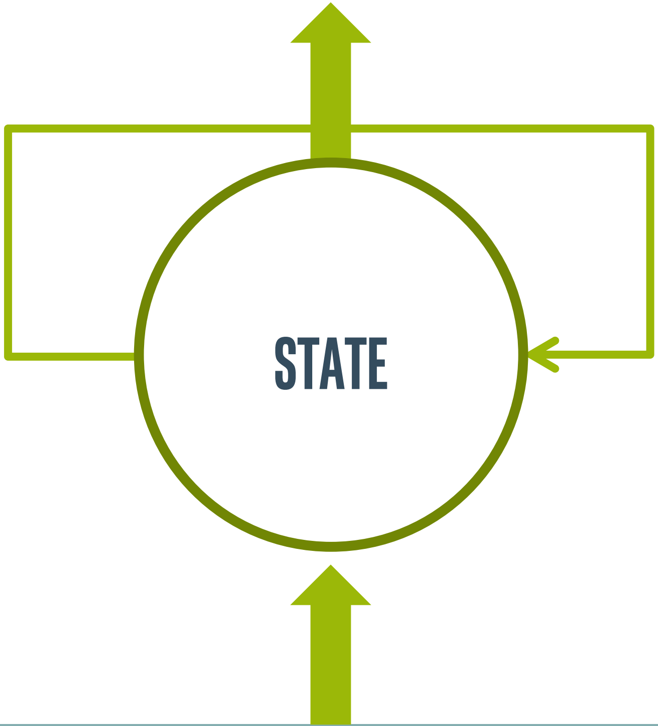
IDEA: USE THE NOTION OF “RECURRENCE”



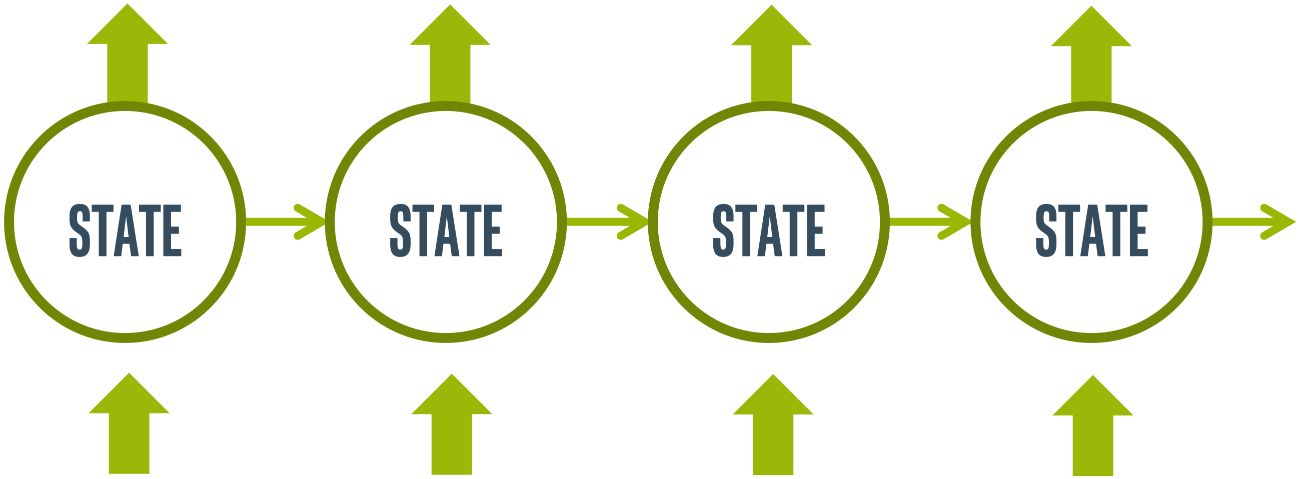
IDEA: USE THE NOTION OF "RECURRENCE"



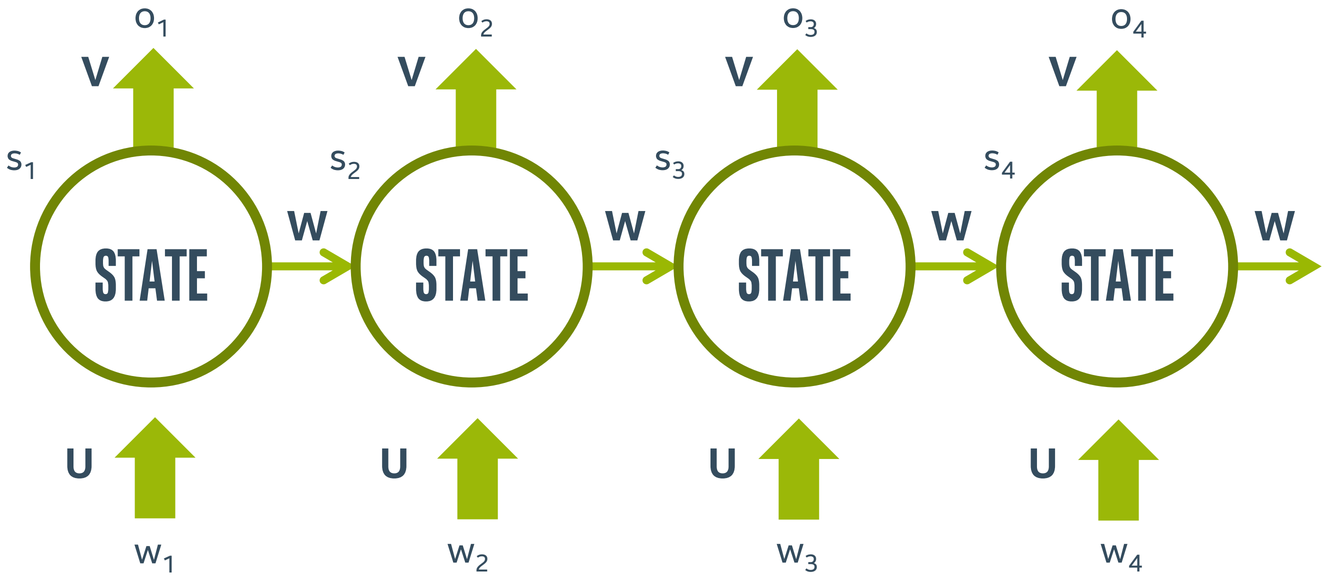
IDEA: USE THE NOTION OF "RECURRENCE"



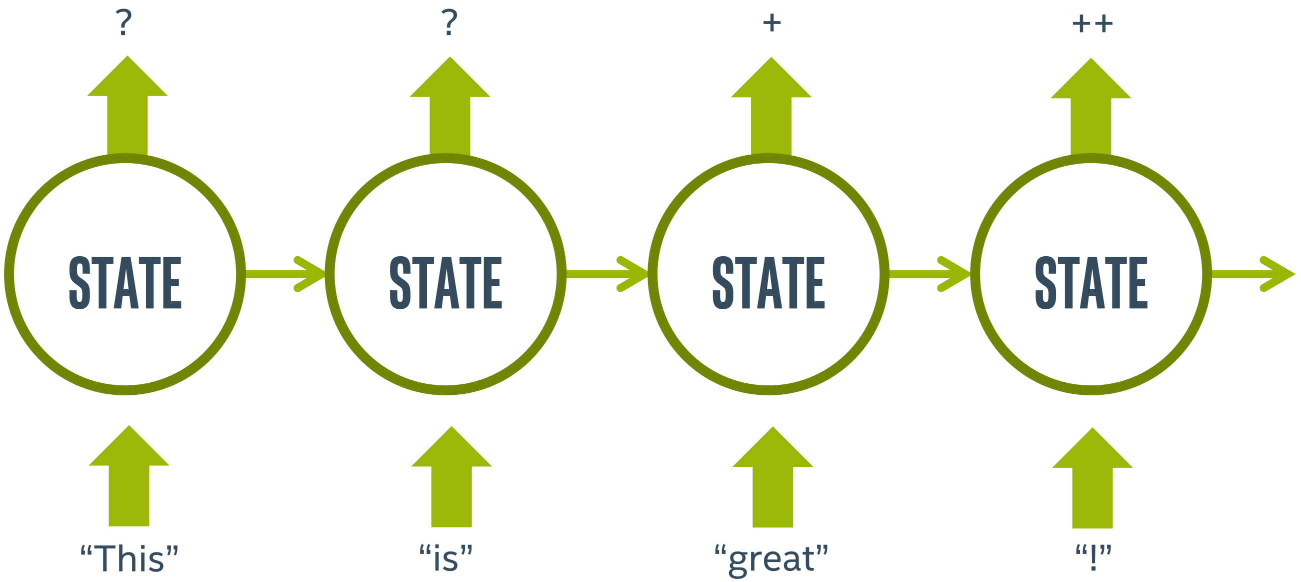
"UNROLLING" THE RNN



"UNROLLING" THE RNN

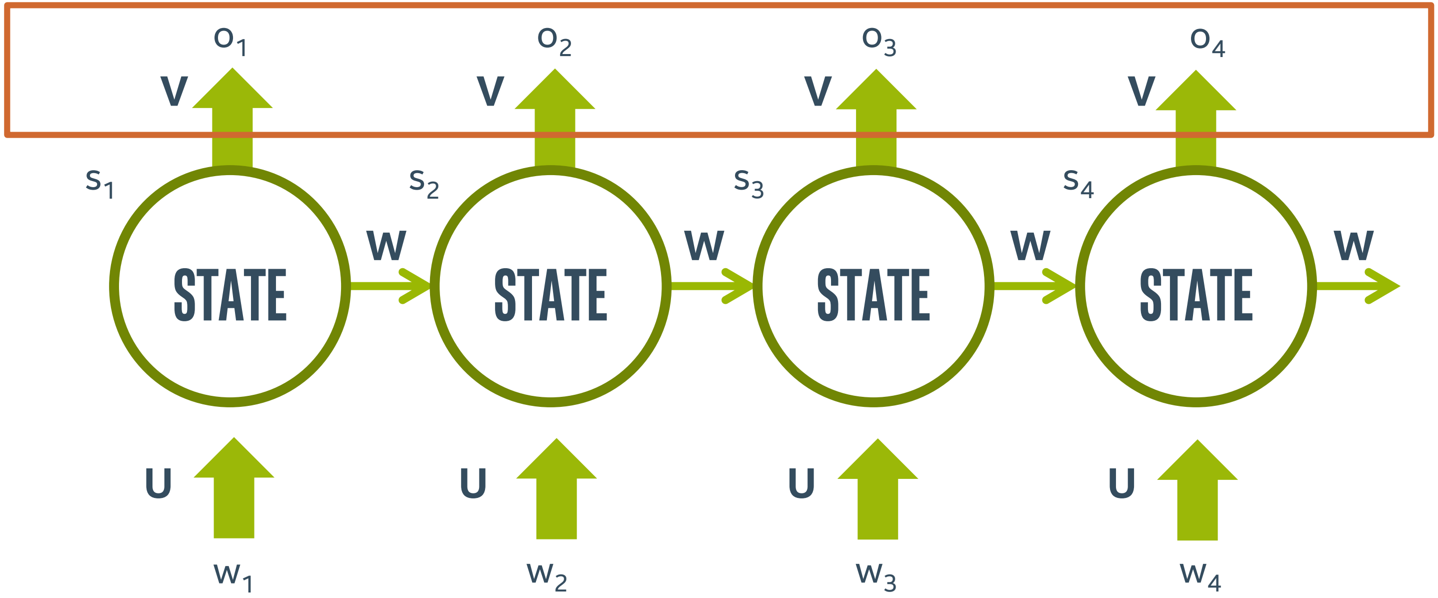


"UNROLLING" THE RNN



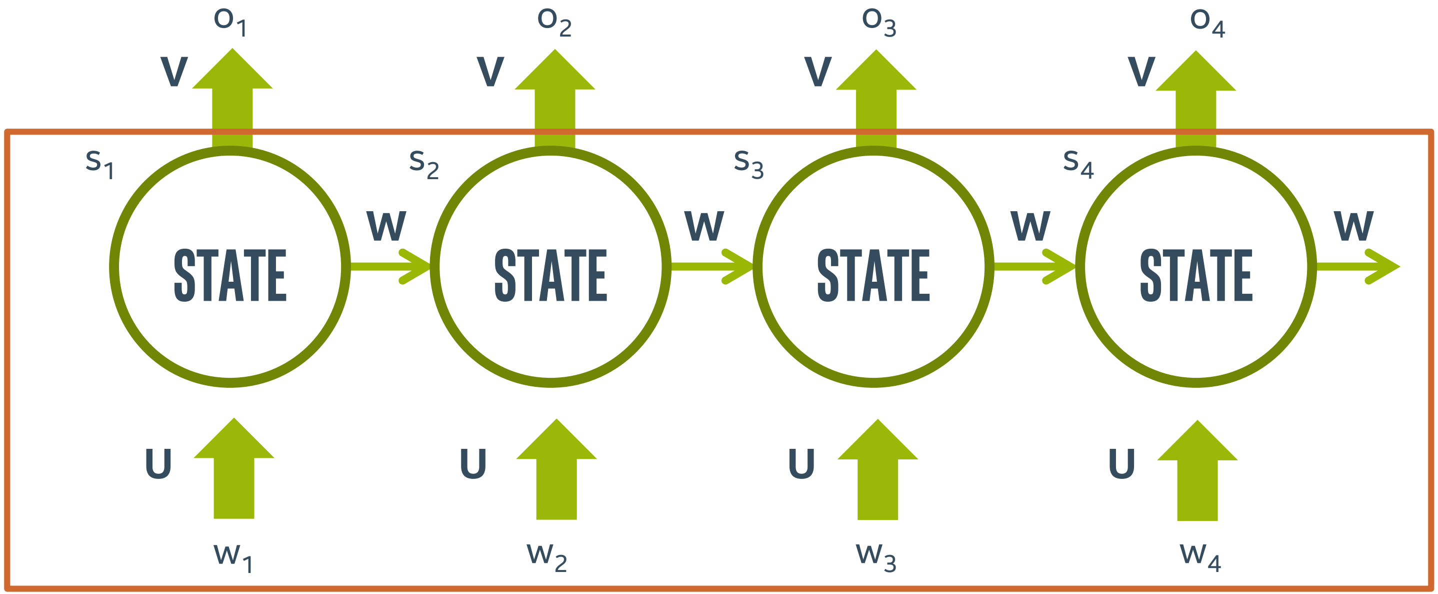
"UNROLLING" THE RNN

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



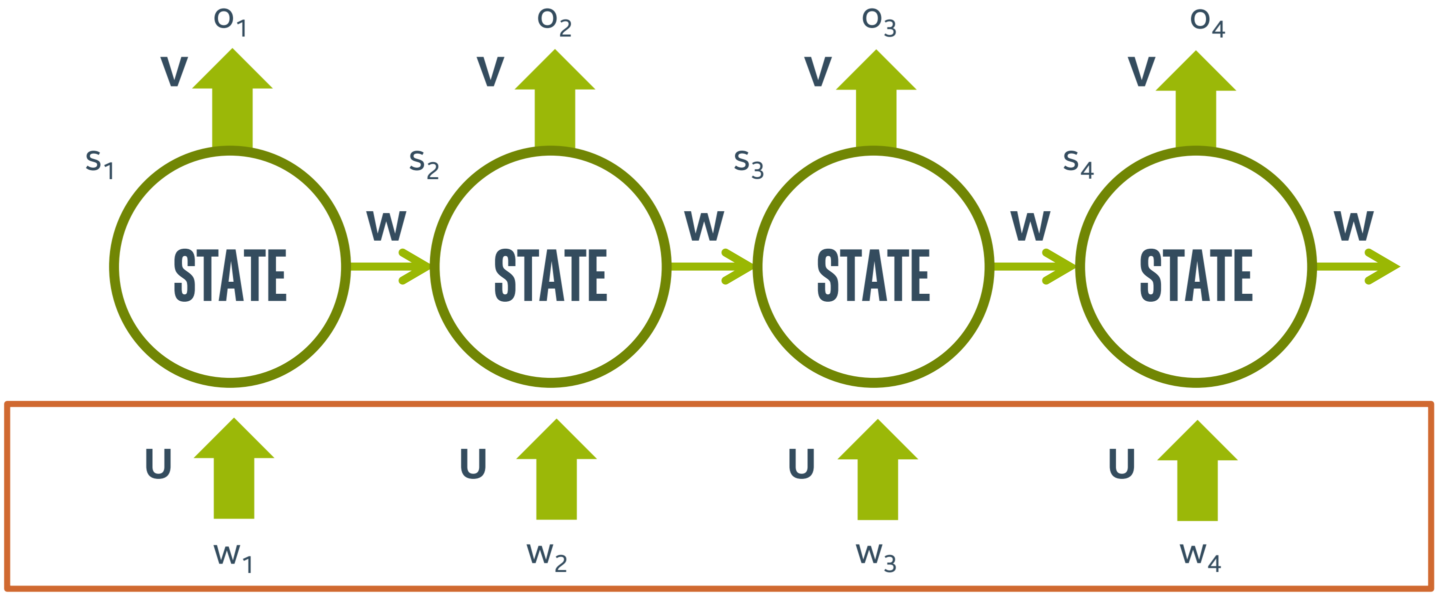
"UNROLLING" THE RNN

This part is the core RNN.



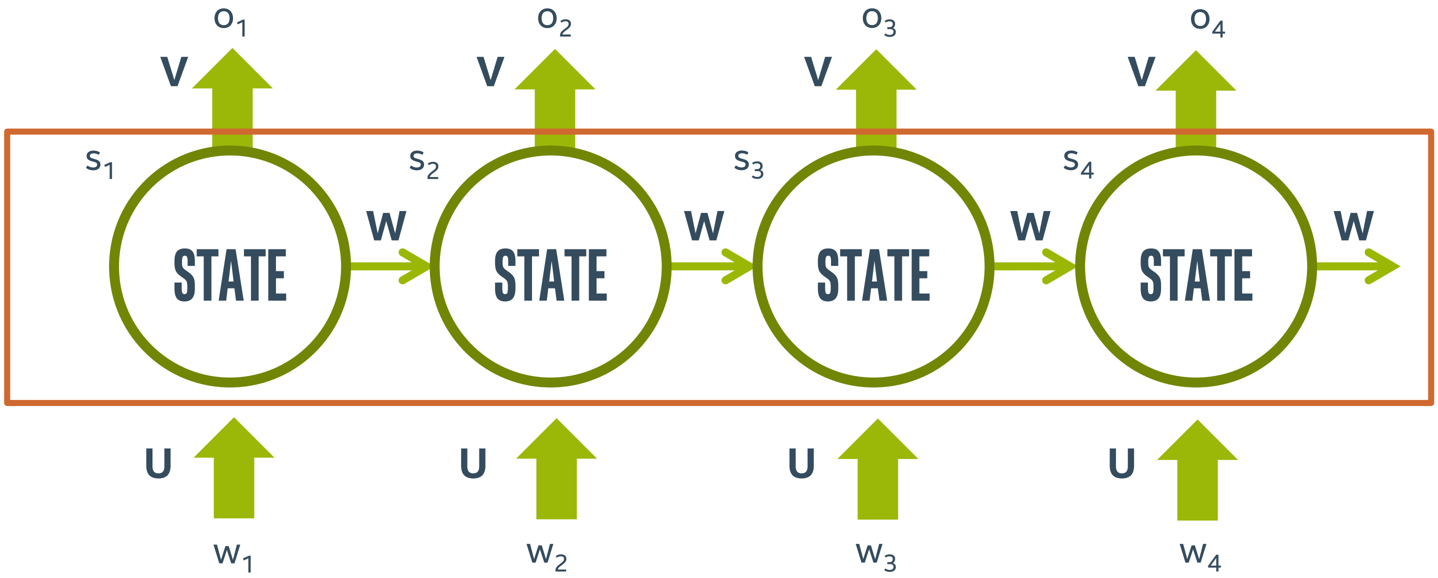
“UNROLLING” THE RNN

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



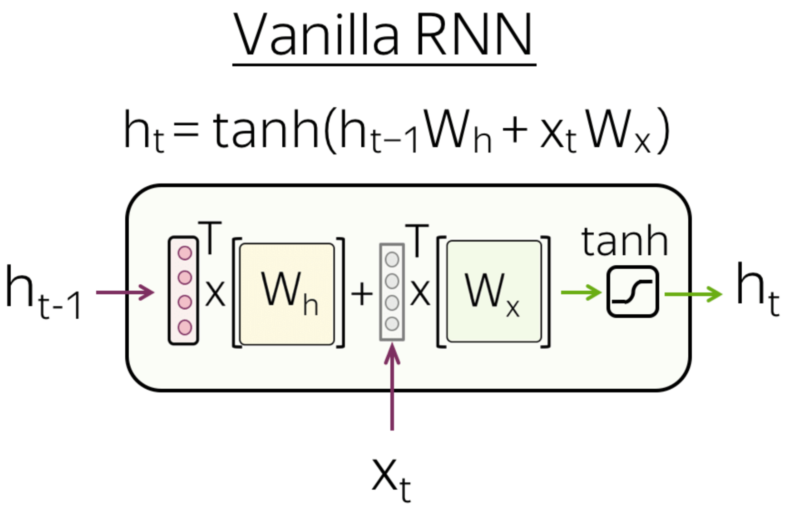
"UNROLLING" THE RNN

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



MATHEMATICAL DETAILS

- w_i is the word at position i
- s_i is the state at position i
- o_i is the output at position i
- $s_i = f(Uw_i + Ws_{i-1})$ (Core RNN)
- $o_i = softmax(Vs_i)$ (subsequent dense layer)



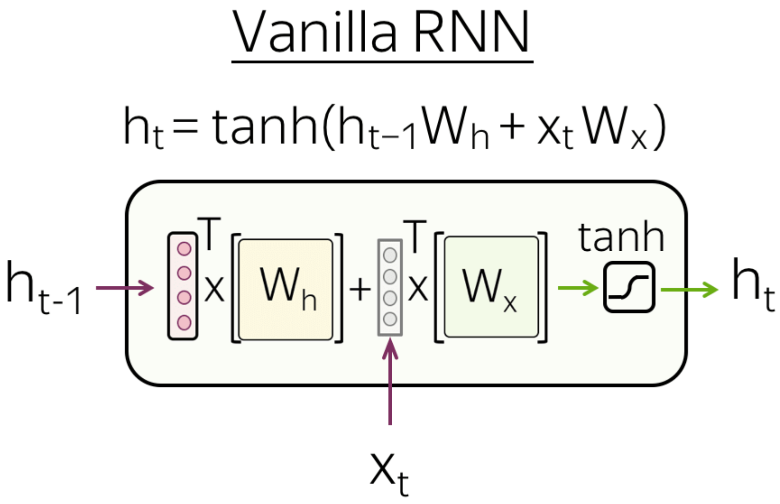
Note: the notations in the illustration are different from the text

MATHEMATICAL DETAILS

- w_i is the word at position i
- s_i is the state at position i
- o_i is the output at position i
- $s_i = f(Uw_i + Ws_{i-1})$ (Core RNN)
- $o_i = softmax(Vs_i)$ (subsequent dense layer)

In other words:

- current state = function1(old state, current input)
- current output = function2(current state)
- We learn function1 and function2 by training our network!



MORE MATHEMATICAL DETAILS

- r = dimension of input vector
- s = dimension of hidden state
- t = dimension of output vector (after dense layer)
- U is a s x r matrix
- W is a s x s matrix
- V is a t x s matrix

Note: The weight matrices U,V,W are the same across all positions.

EXAMPLE

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense

model = Sequential(name='RNN')
model.add(SimpleRNN(4, input_shape=(3, 2), name='RNN_Layer'))
model.add(Dense(1))
```

RNN layer number of parameters = recurrent_weights + input_weights + biases

recurrent_weights = num_units * num_units (4*4)

input_weights = num_feats * num_units (2*4)

biases = num_units (4)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
RNN_Layer (SimpleRNN)        (None, 4)                 28

dense_5 (Dense)              (None, 1)                 5

=================================================================
Total params: 33
_________________________________________________________________
```

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

A Visual Guide to Recurrent Layers in Keras

Intel Nervana™ AI Academy

LSTM (LONG-SHORT TERM MEMORY) RNNS

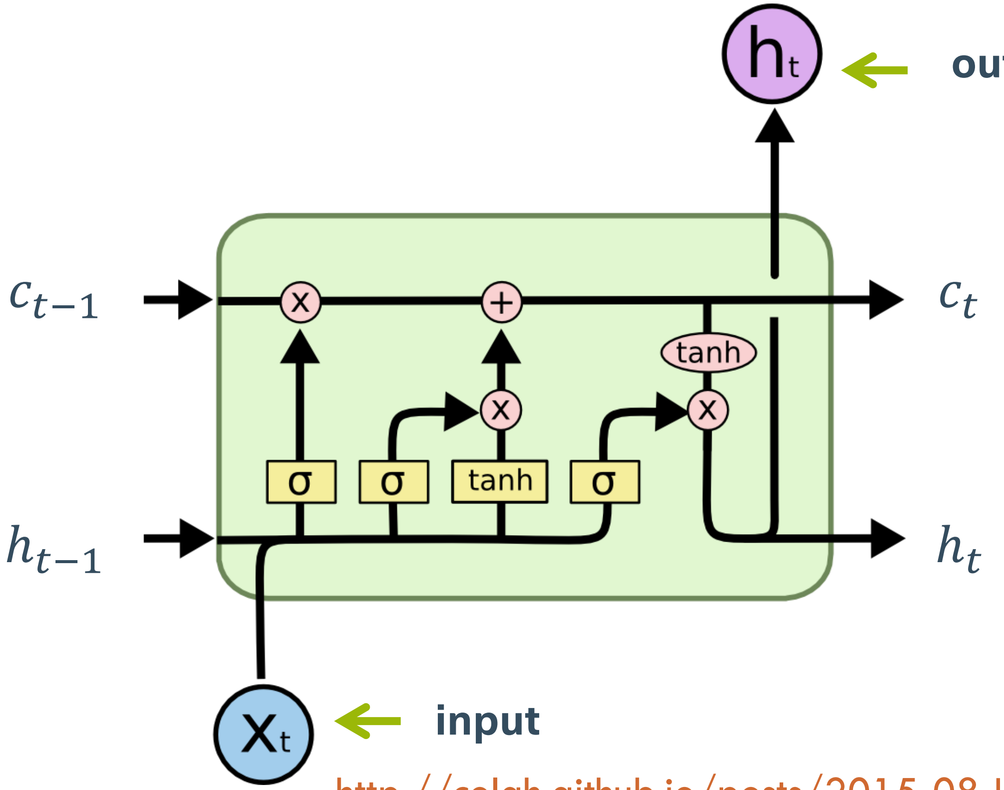
ISSUE: STANDARD RNNS HAVE POOR MEMORY

- 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

LSTM has an extra "CELL" state besides the hidden RNN state

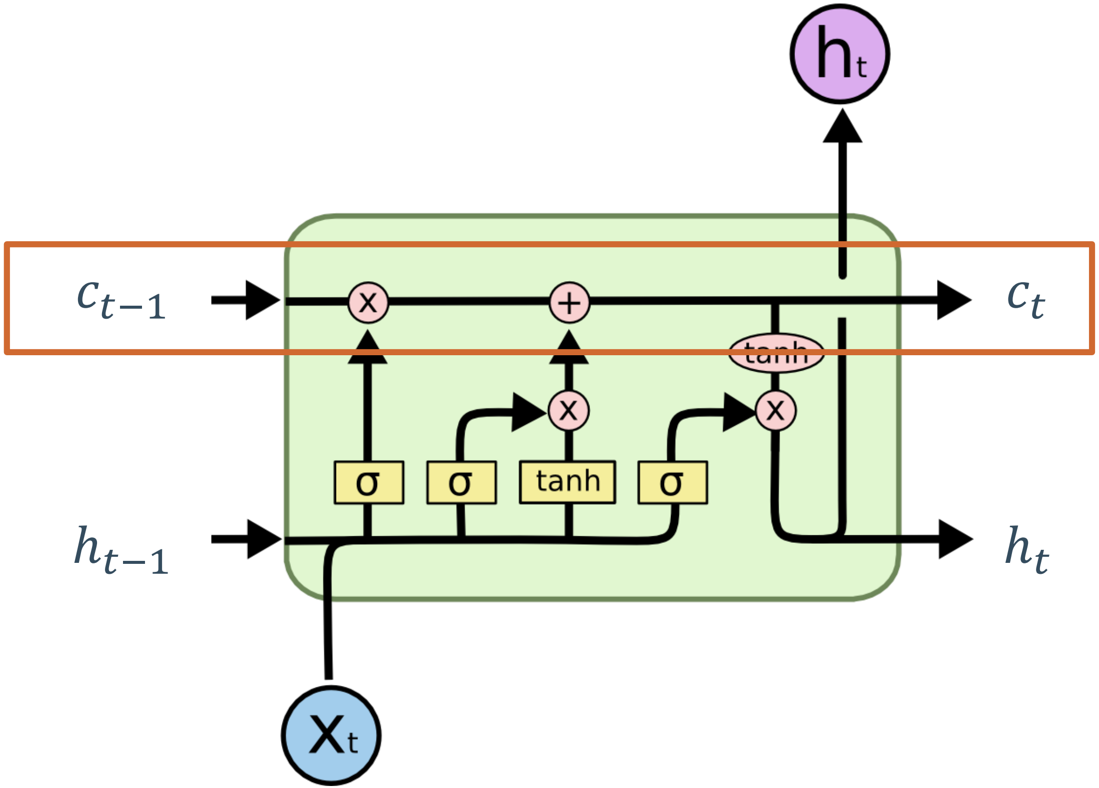
cell state

output (which also feeds into next unit)

input

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

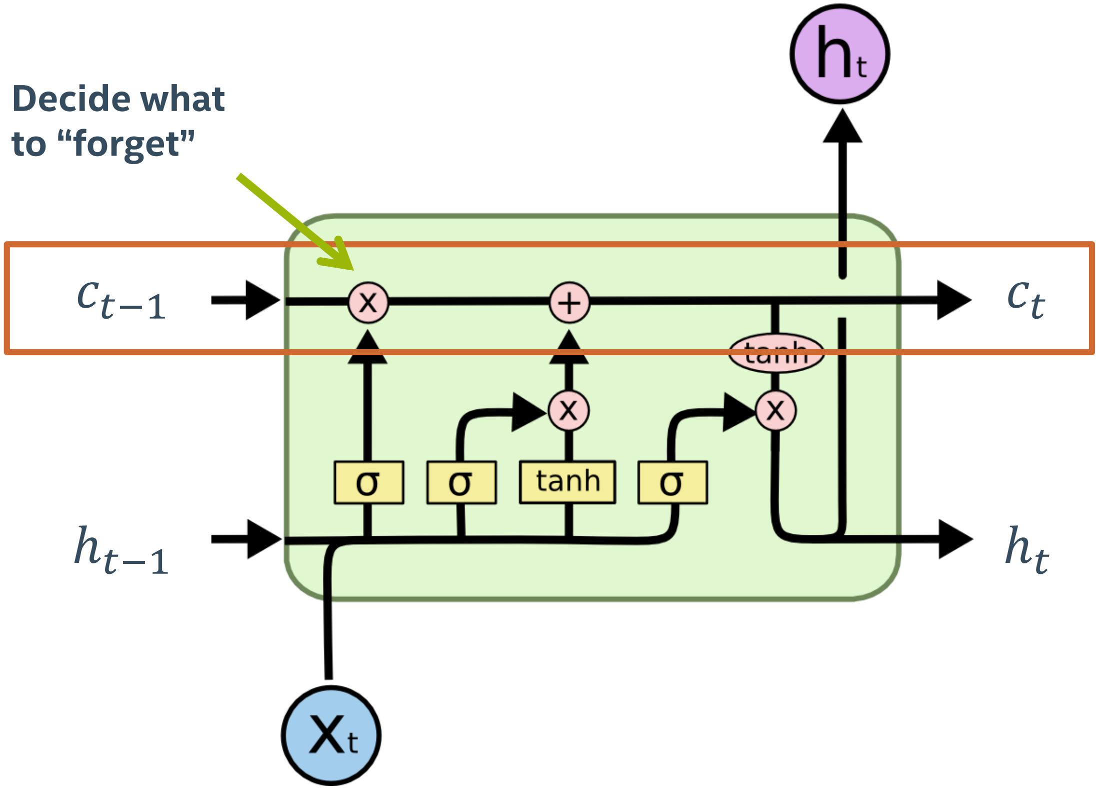
LSTM DIAGRAM



cell state gets updated in two stages

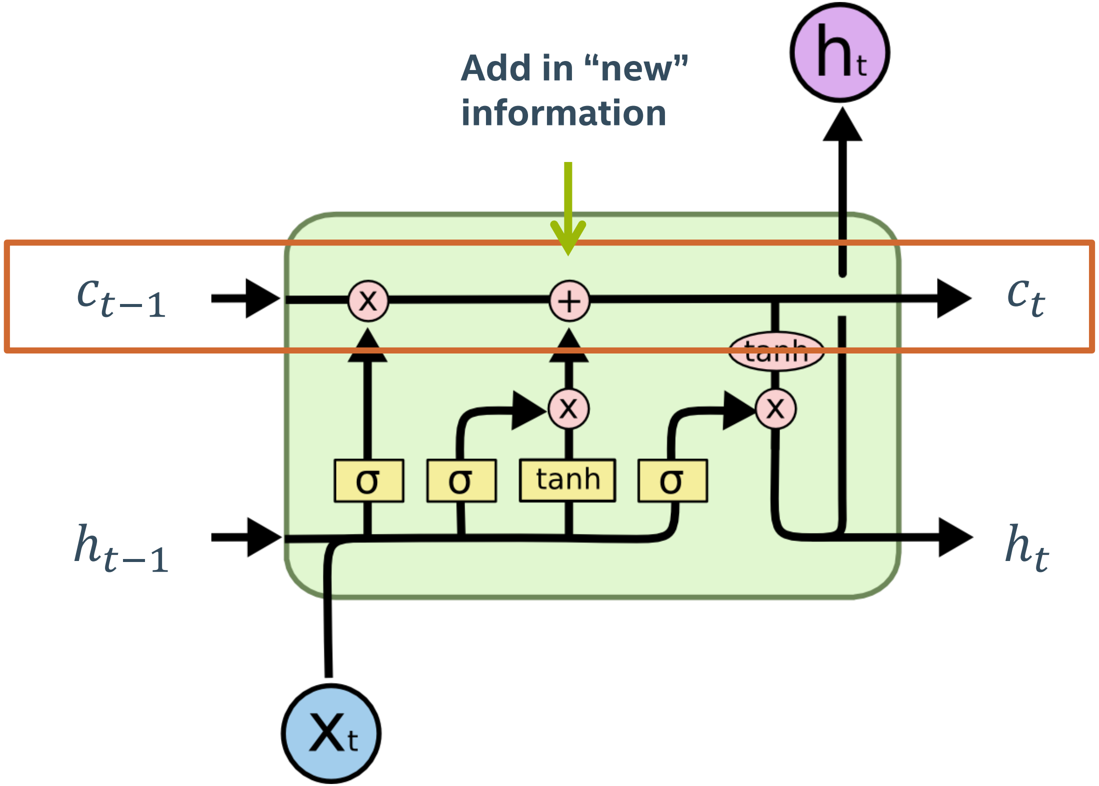
LSTM DIAGRAM

Decide what to "forget"



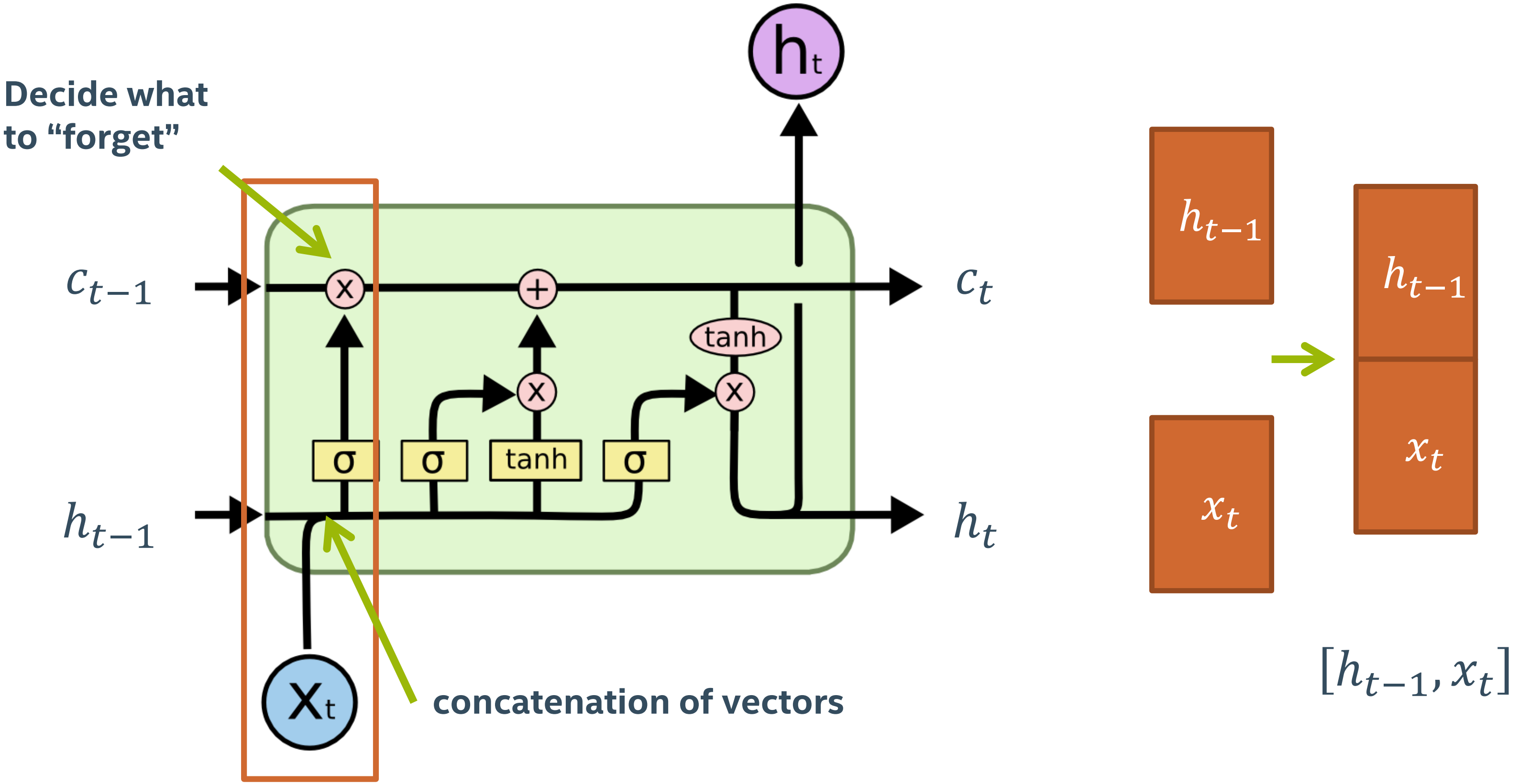
cell state gets updated in two stages

LSTM DIAGRAM



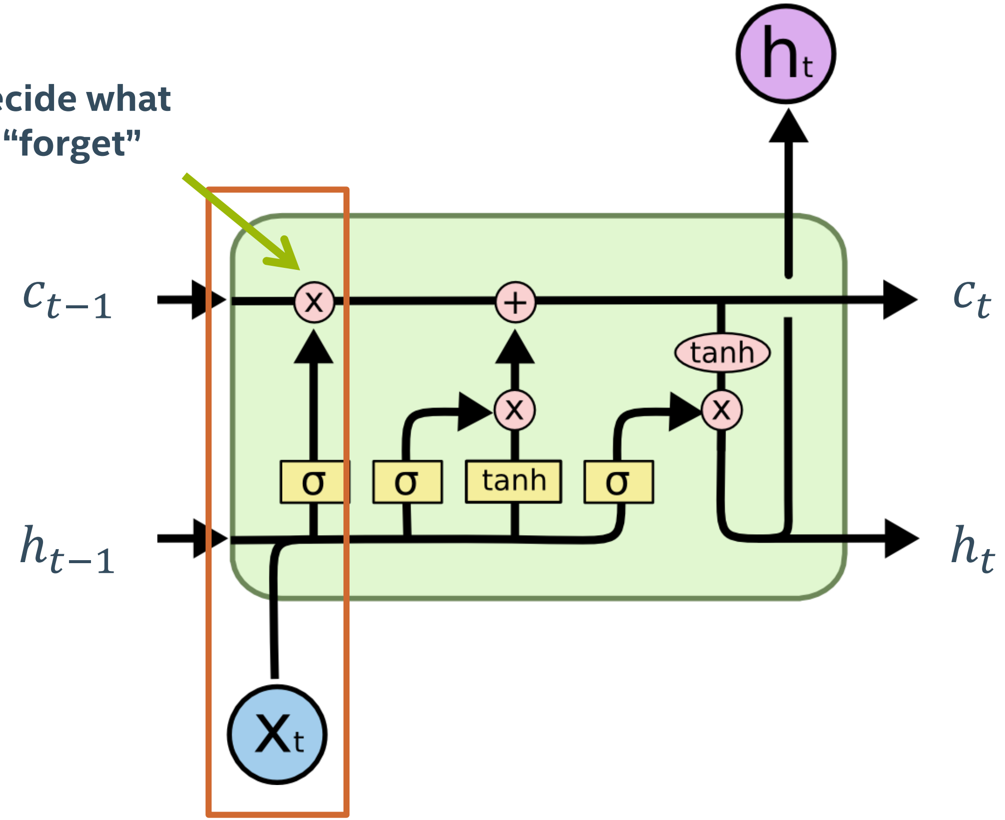
cell state gets updated in two stages

LSTM DIAGRAM



LSTM DIAGRAM

Decide what to "forget"

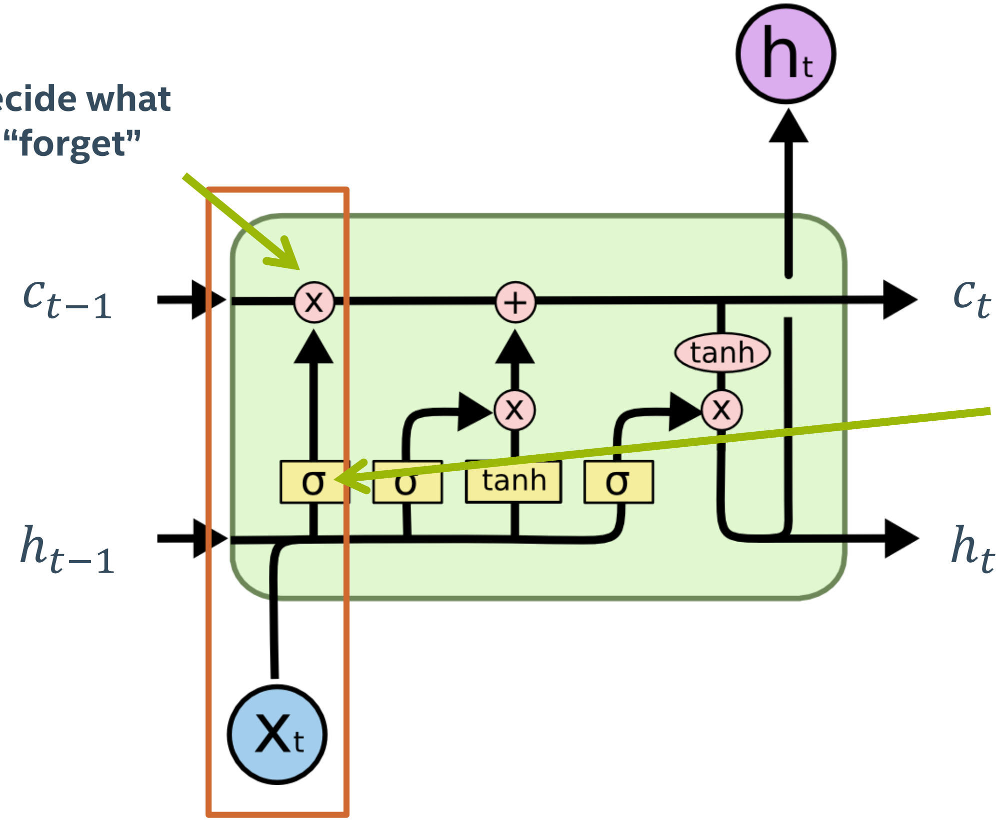


$$f_t = \sigma\left(W_f[h_{t-1}, x_t] + b_f\right)$$

based on previous output and current input

LSTM DIAGRAM

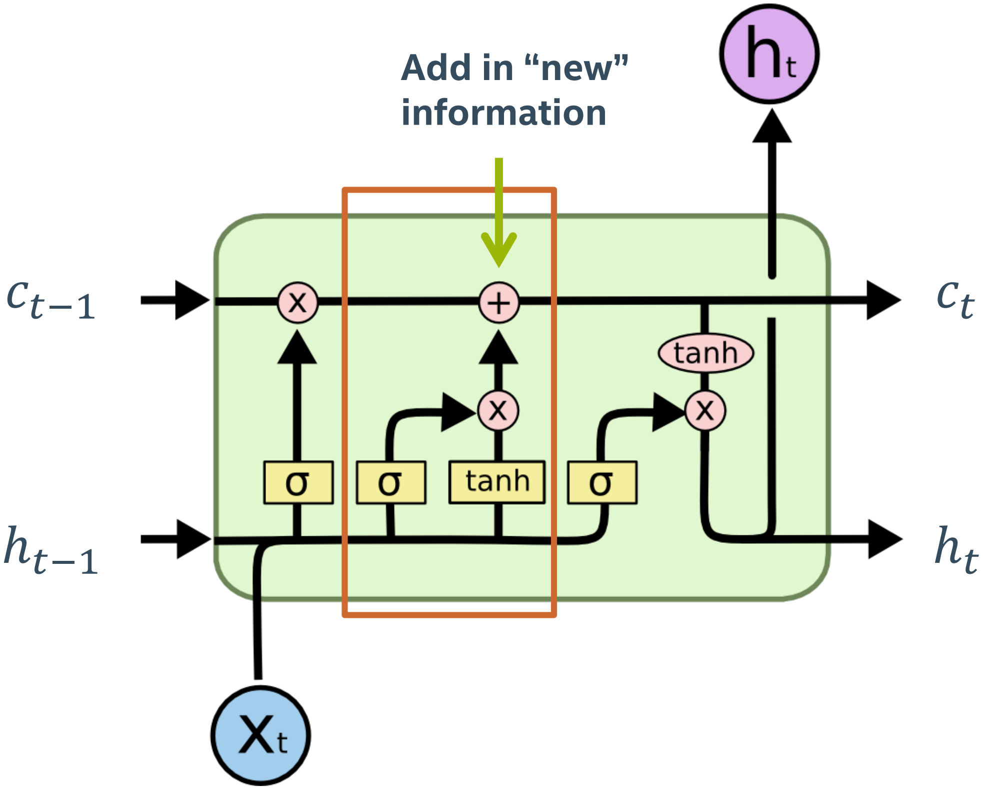
Decide what to "forget"



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

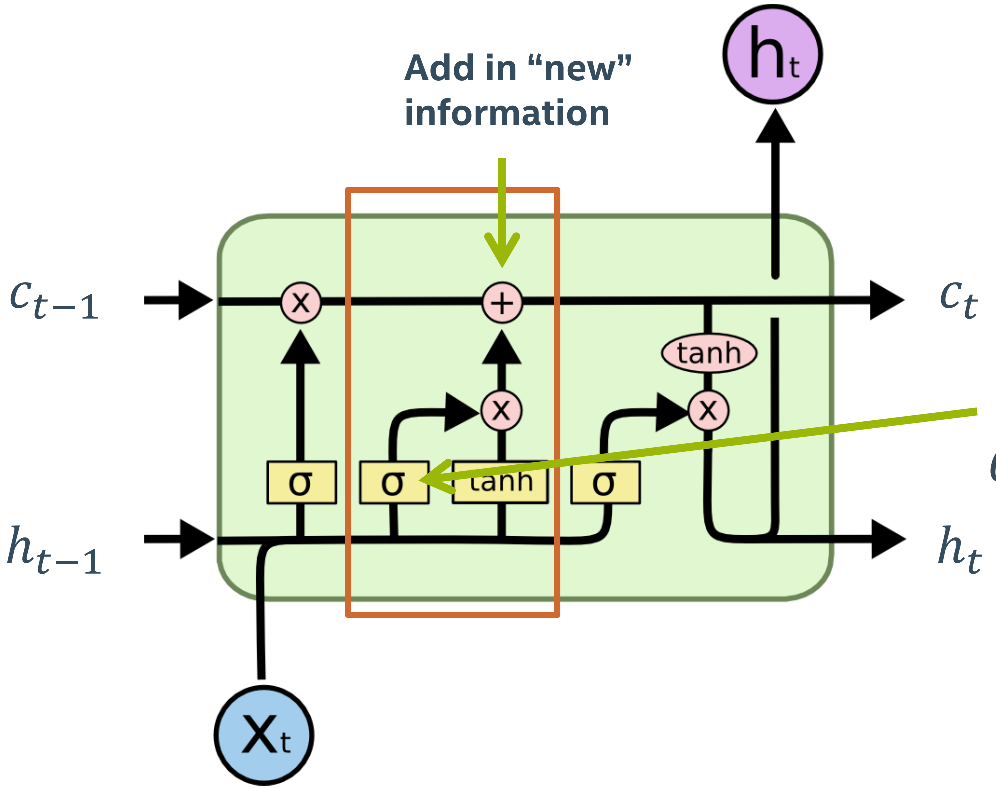
based on previous output and current input

LSTM DIAGRAM



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

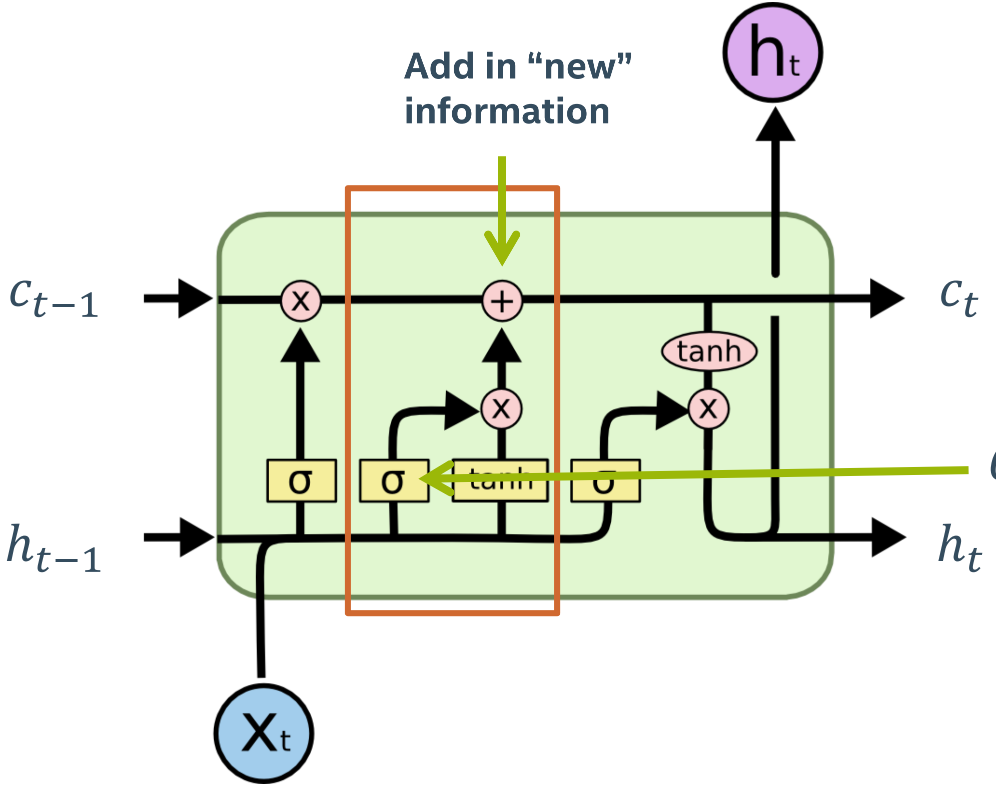
LSTM DIAGRAM



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

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

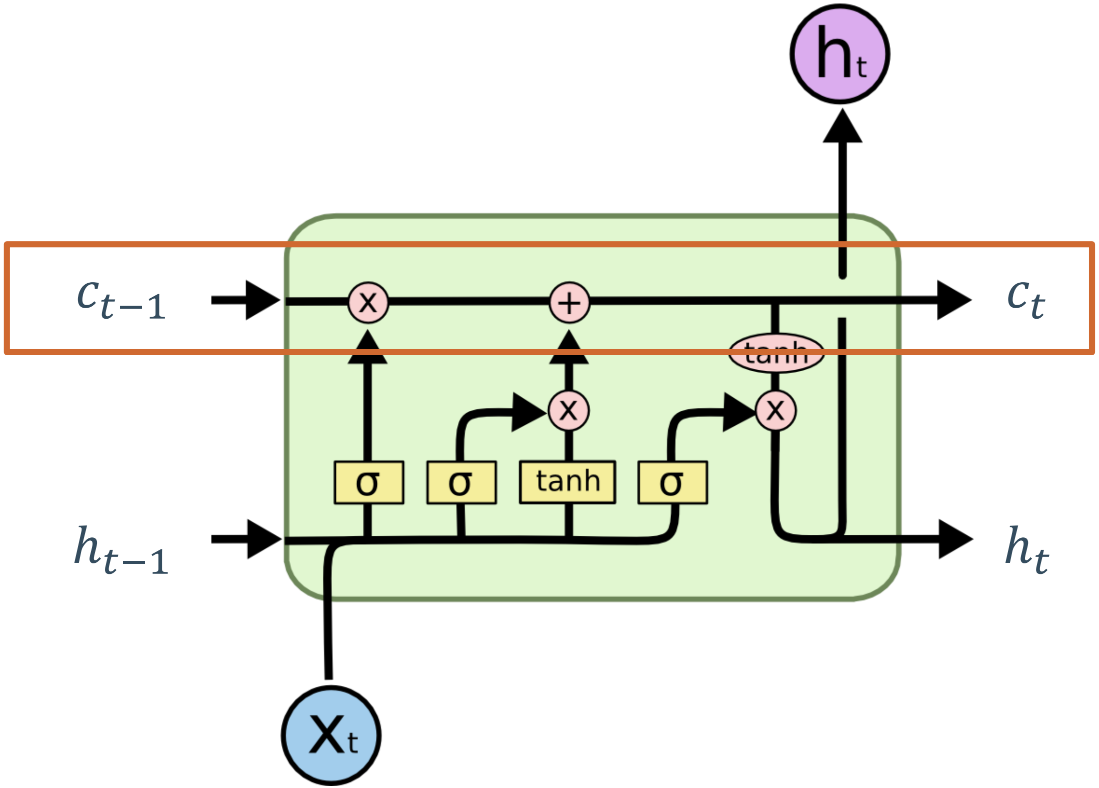
LSTM DIAGRAM



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

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

LSTM DIAGRAM



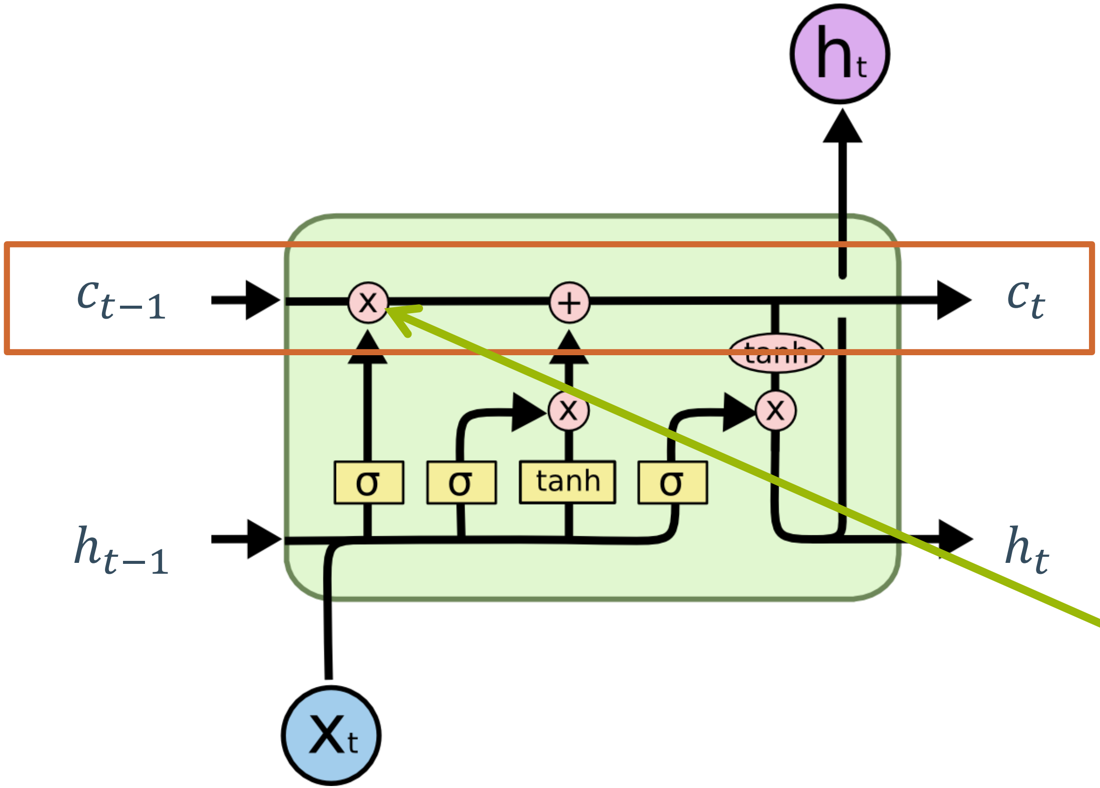
Note: '∗' represents element-wise multiplication

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

forget the old (or not)

add the new (or not)

LSTM DIAGRAM



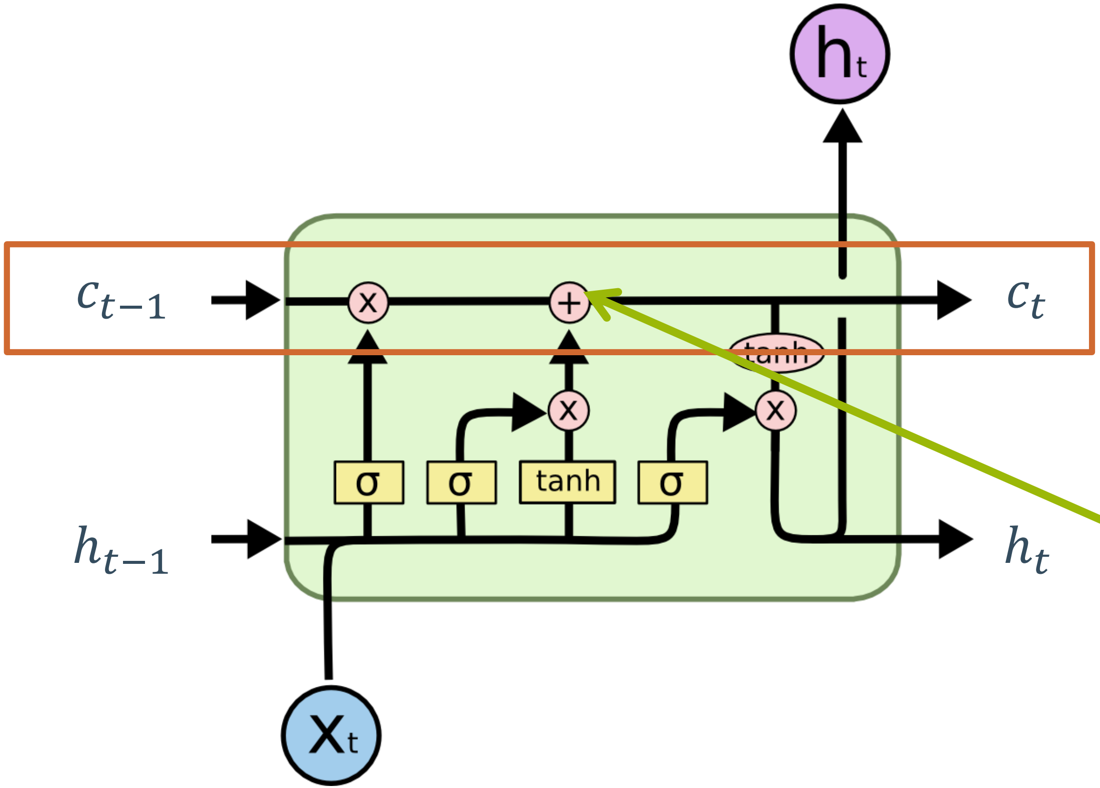
Note: '∗' represents element-wise multiplication

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

forget the old (or not)

add the new (or not)

LSTM DIAGRAM



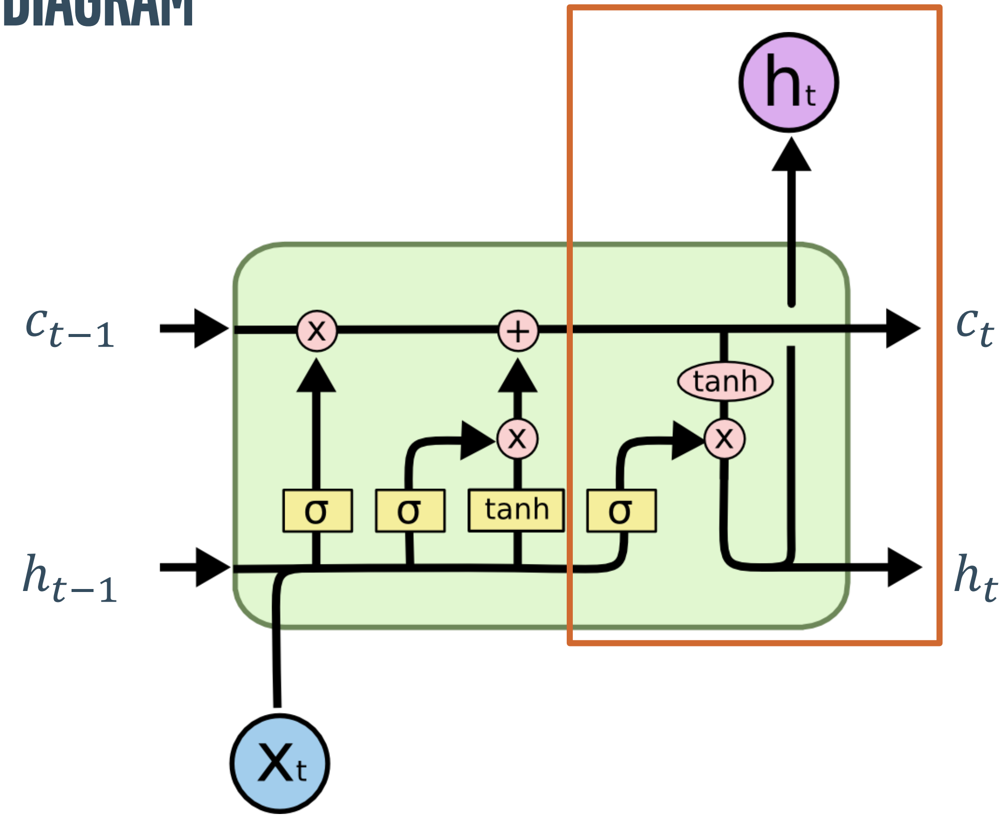
Note: '∗' represents element-wise multiplication

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

forget the old (or not)

add the new (or not)

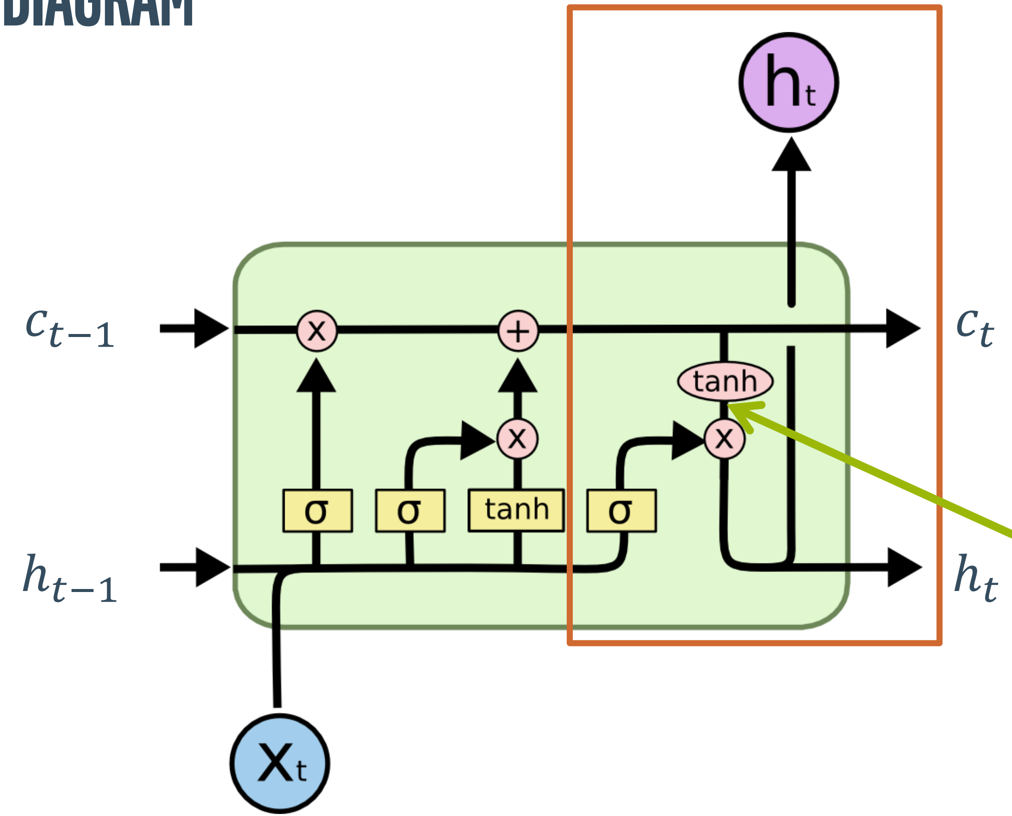
LSTM DIAGRAM



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

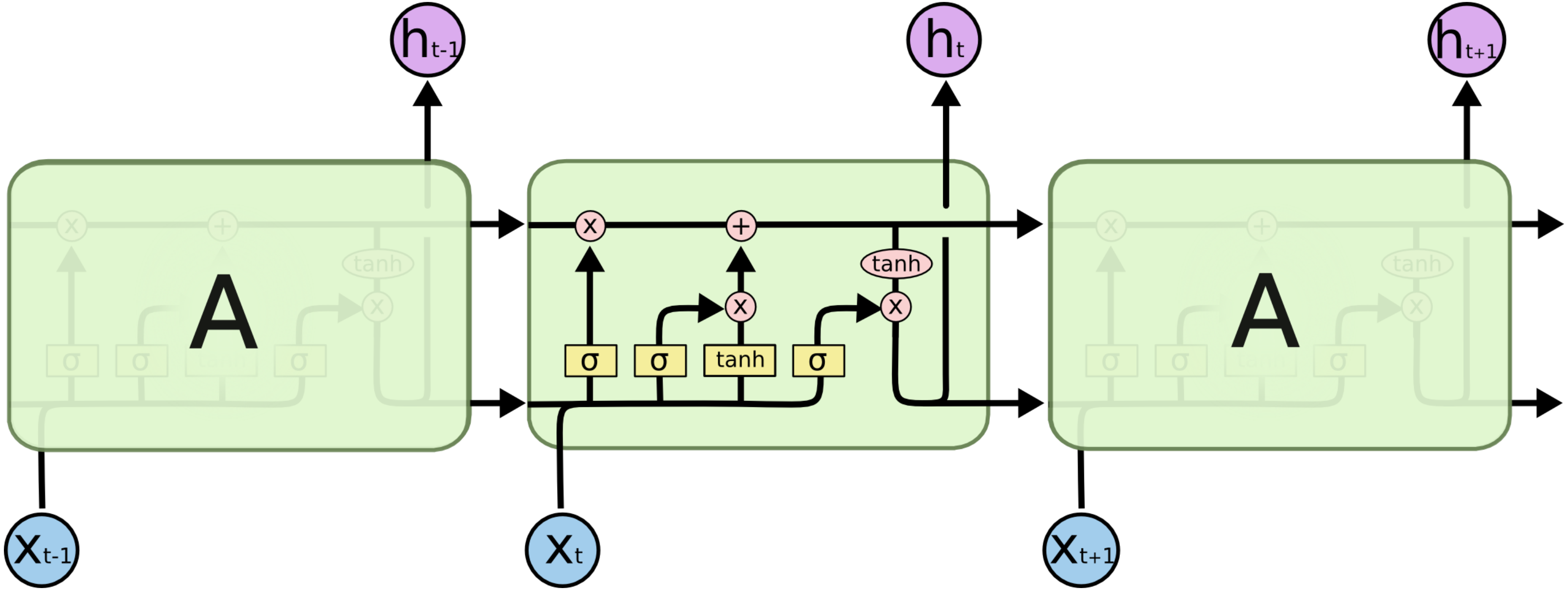


Final stage computes the output

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

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

Nervana™ AI Academy