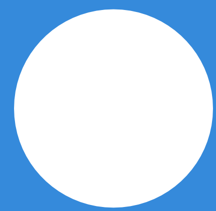
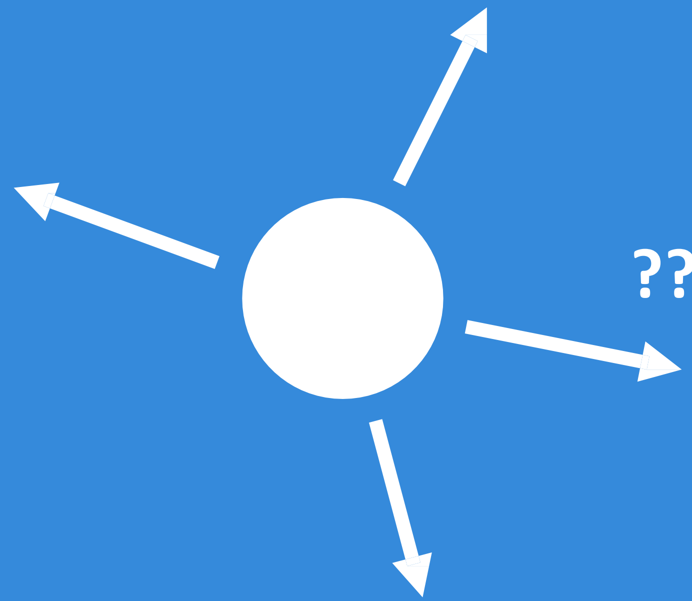
Deep Sequence Modeling

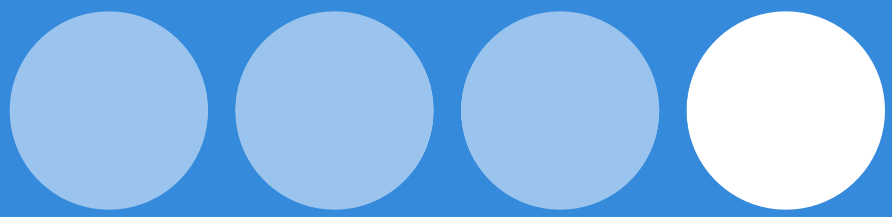
Given an image of a ball,
can you predict where it will go next?



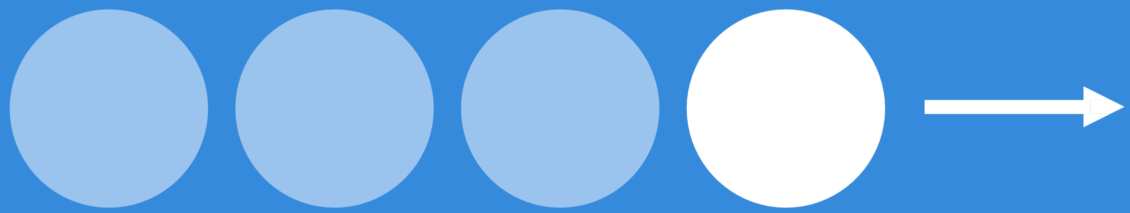
Given an image of a ball,
can you predict where it will go next?



Given an image of a ball,
can you predict where it will go next?



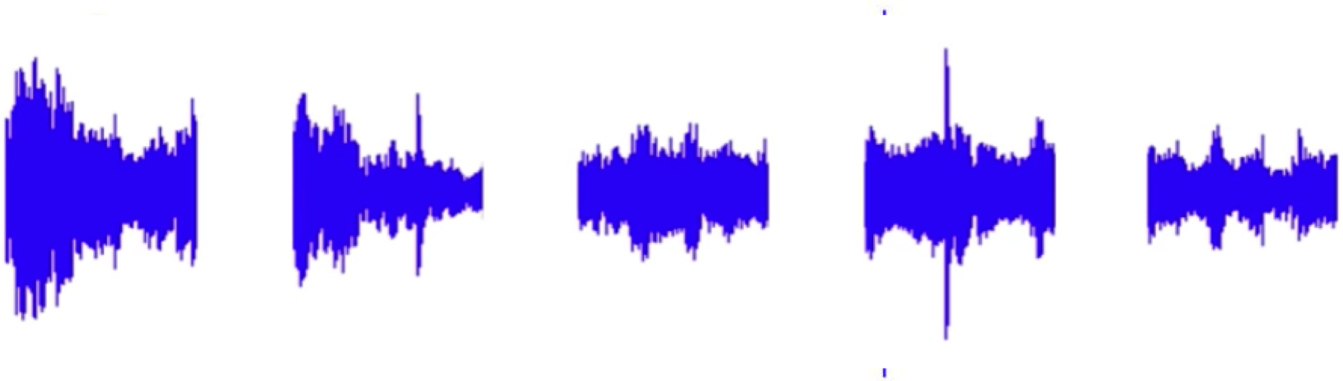
Given an image of a ball,
can you predict where it will go next?



Sequences in the wild

Audio

Sequences in the wild



Audio

Sequences in the wild

character:

6.S191 Introduction to Deep Learning

word:

Text

Sequences in the wild

character: 6 . S 1 9 1

word: Introduction to Deep Learning

Text

A Sequence Modeling Problem: Predict the Next Word

A sequence modeling problem: predict the next word

"This morning I took my cat for a walk."

given these words

predict the next word

Adapted from H. Suresh, 6.S191 2018

Idea #1: use a fixed window

"This morning I took my cat for a walk."

given these two words

predict the next word

One-hot feature encoding: tells us what each word is

[1 0 0 0 0 0 1 0 0 0]

for a

prediction

Adapted from H. Suresh, 6.S191 2018

Problem #1: can't model long-term dependencies

"**France** is where I grew up, but now I live in Boston. I speak a fluent ____"

We need information from the distant past
to accurately predict the current word

Idea #2: use entire sequence as set of counts

"This morning I took my cat for a"

"bag of words"

[0 1 0 0 1 0 0 ... 0 0 1 1 0 0 0 1]

prediction

Adapted from H. Suresh, 6.S191 2018

Problem #2: counts don't preserve order

The food was good, not bad at all.

vs.

The food was bad, not good at all.

Adapted from H. Suresh, 6.S191 2018

Idea #3: use a really big fixed window

"This morning I took my cat for a walk."

given these words

predict the next word

[1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 ...]

morning I took this cat

prediction

Adapted from H. Suresh, 6.S191 2018

Problem #3: no parameter sharing

[1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 ...]

this morning took the cat

Each of these inputs has a **separate parameter:**

[0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 ...]

this morning

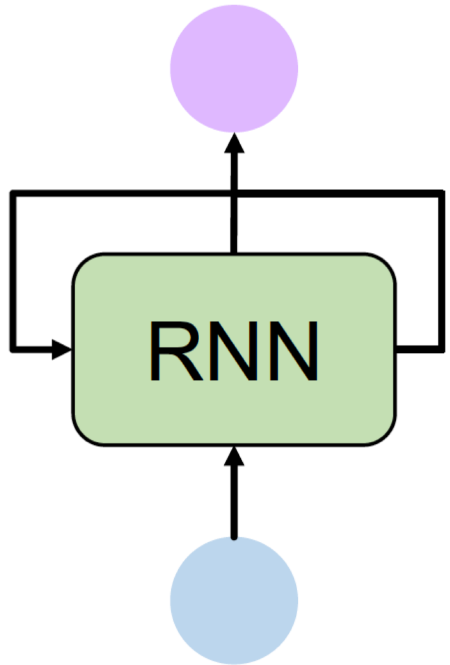
Things we learn about the sequence **won't transfer** if they appear **elsewhere** in the sequence.

Adapted from H. Suresh, 6.S191 2018

Sequence modeling: design criteria

To model sequences, we need to:

1. Handle **variable-length** sequences
2. Track **long-term** dependencies
3. Maintain information about **order**
4. **Share parameters** across the sequence

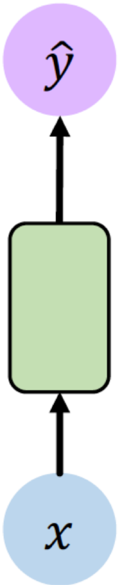


Today: **Recurrent Neural Networks (RNNs)** as an approach to sequence modeling problems

Adapted from H. Suresh, 6.S191 2018

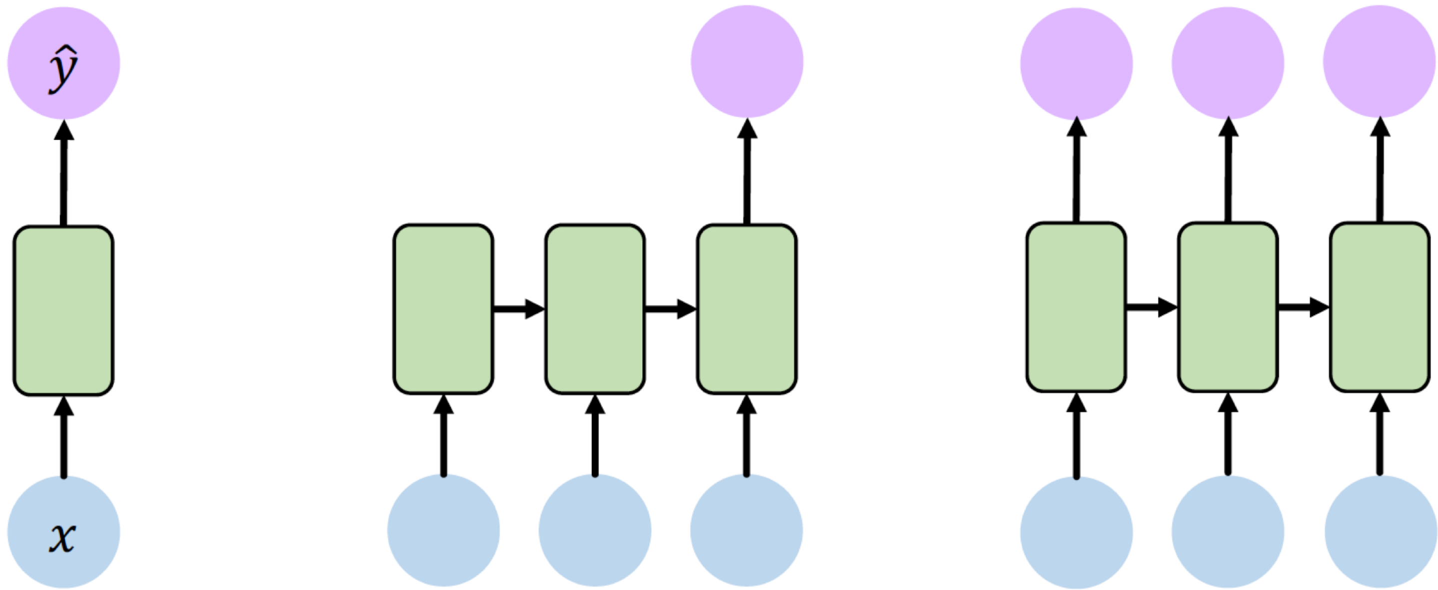
Recurrent Neural Networks (RNNs)

Standard feed-forward neural network



One to One
"Vanilla" neural network

Recurrent neural networks: sequence modeling



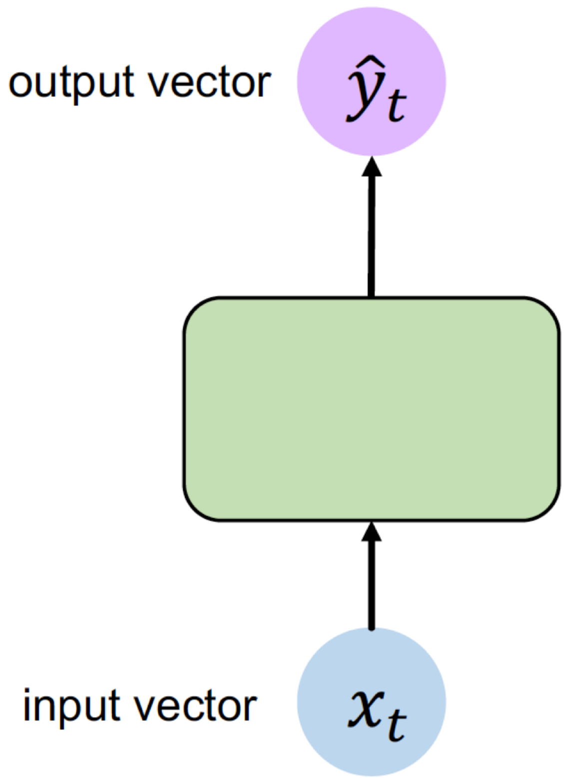
One to One
"Vanilla" neural network

Many to One
Sentiment Classification

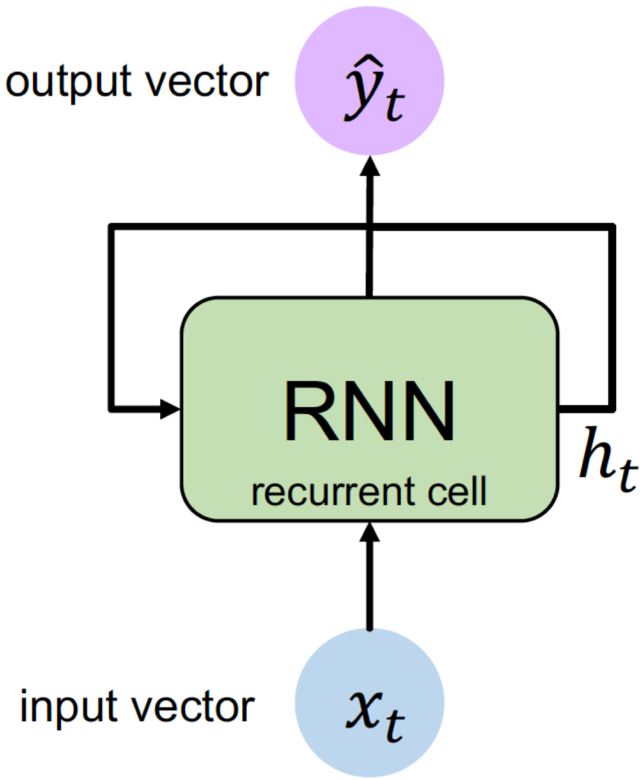
Many to Many
Music Generation

… and many other architectures and applications

A standard "vanilla" neural network



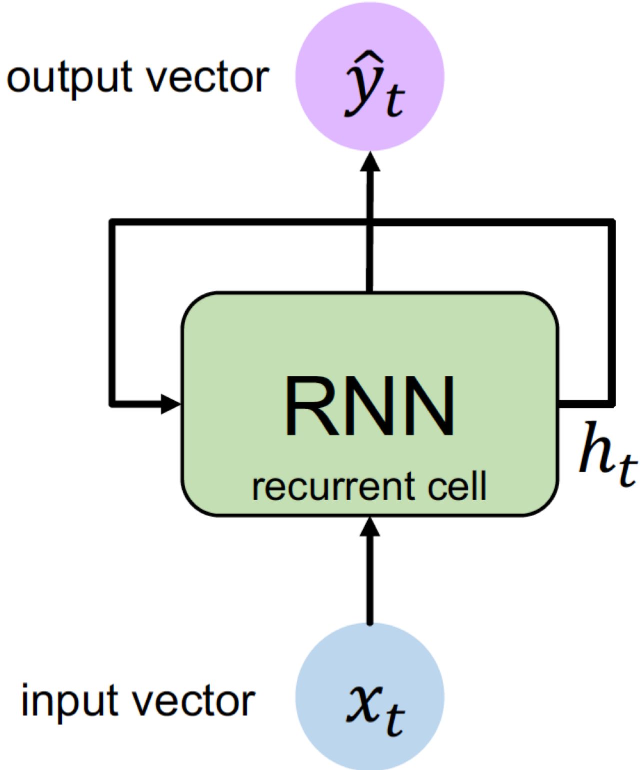
A recurrent neural network (RNN)



Recurrent:

information is being passed internally from one time step to the next

A recurrent neural network (RNN)



Apply a **recurrence relation** at every time step to process a sequence:

$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

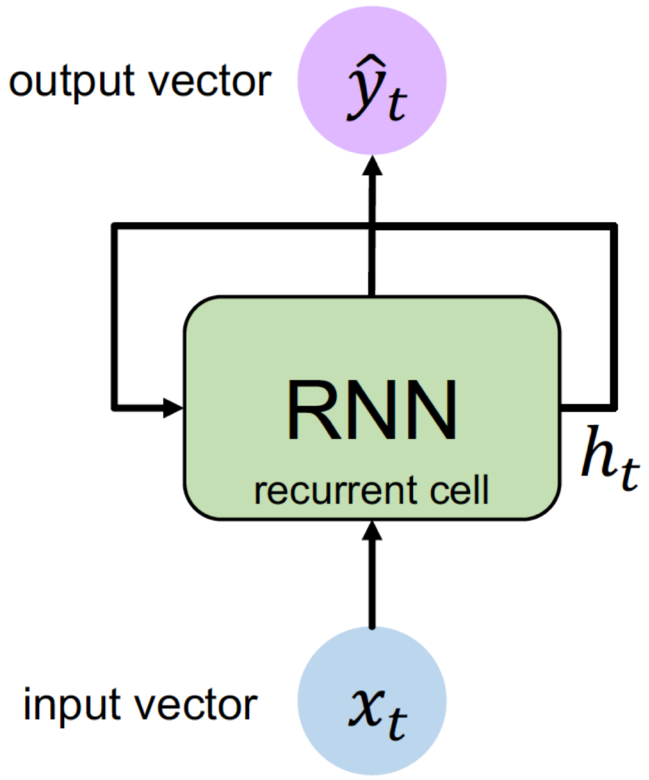
function parameterized by W

old state

input vector at time step *t*

Note: the same function and set of parameters are used at every time step

RNN state update and output



Output Vector

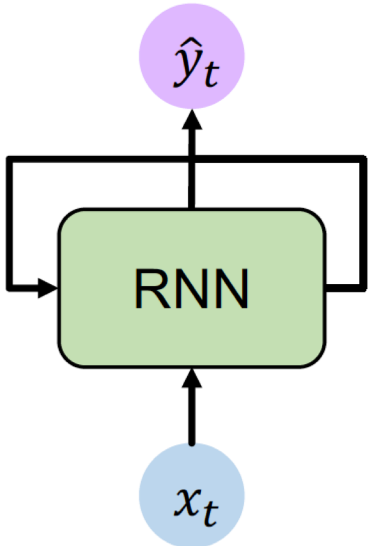
$$\hat{y}_t = \boldsymbol{W}_{\boldsymbol{hy}} h_t$$

Hidden State

$$h_t = \tanh(\boldsymbol{W}_{\boldsymbol{hh}} h_{t-1} + \boldsymbol{W}_{\boldsymbol{xh}} x_t)$$

Input Vector

RNNs: computational graph across time



=

Represent as computational graph unrolled across time

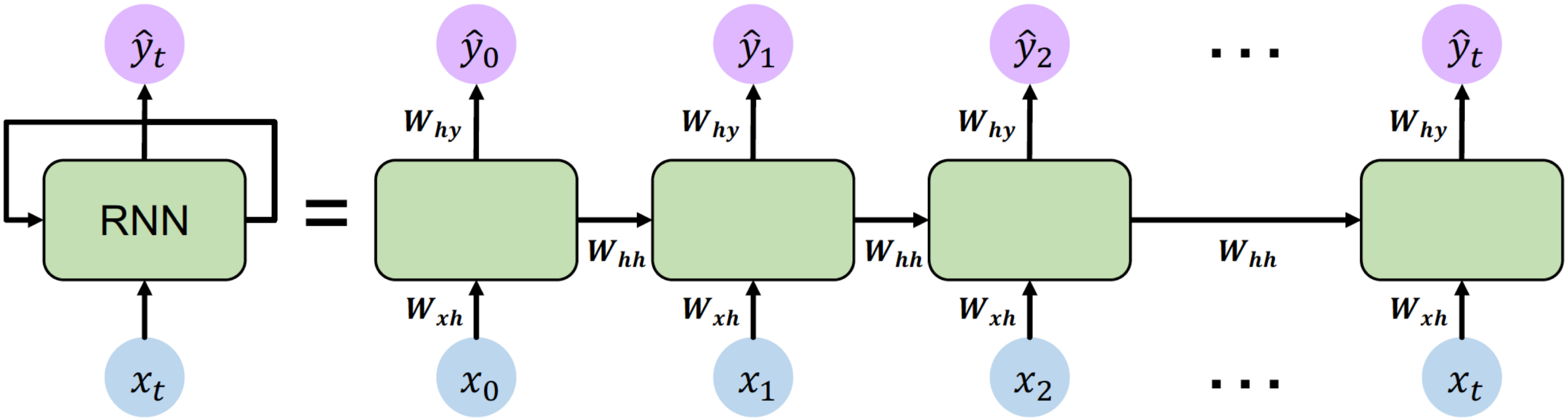
RNNs: computational graph across time

Re-use the **same weight** matrices at every time step

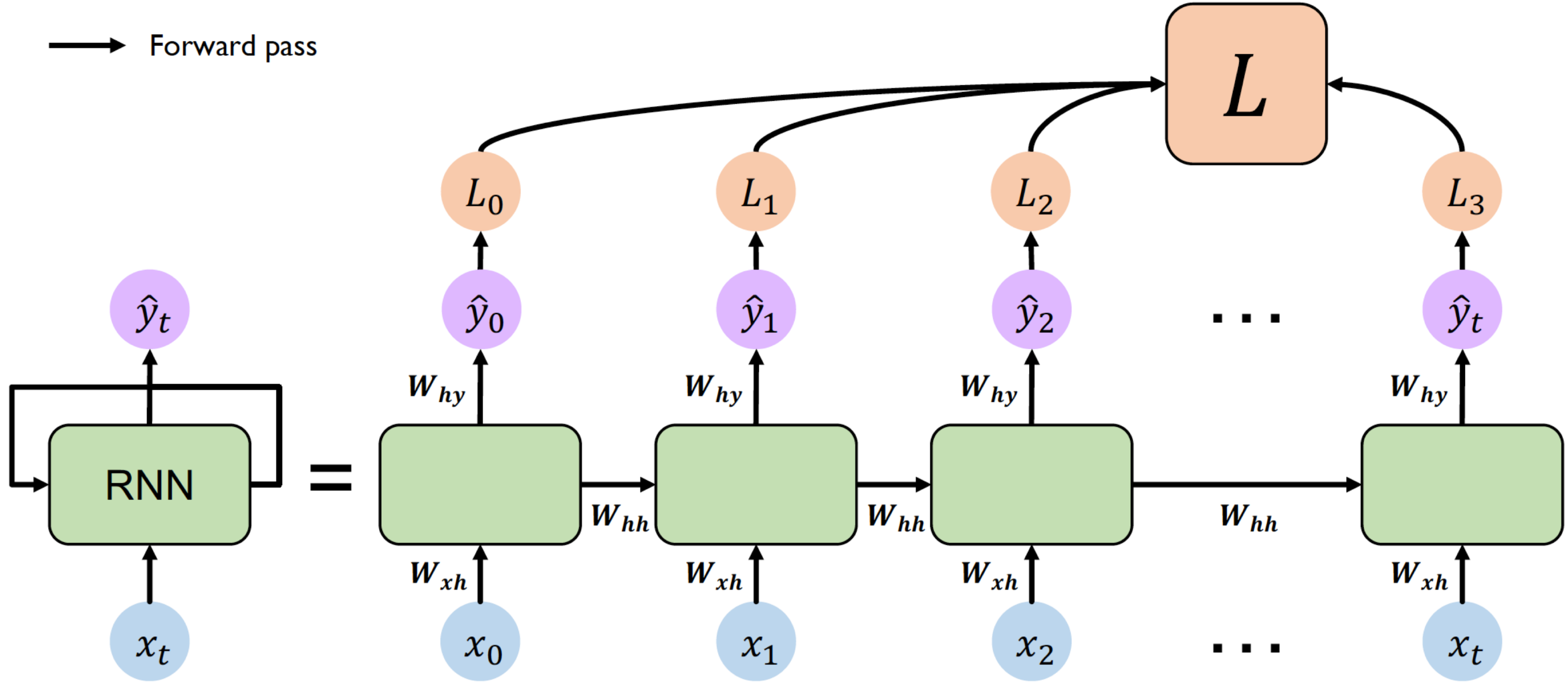
$W_{xh}: x \rightarrow h$

$W_{hh}: h \rightarrow h$

$W_{hy}: h \rightarrow y$

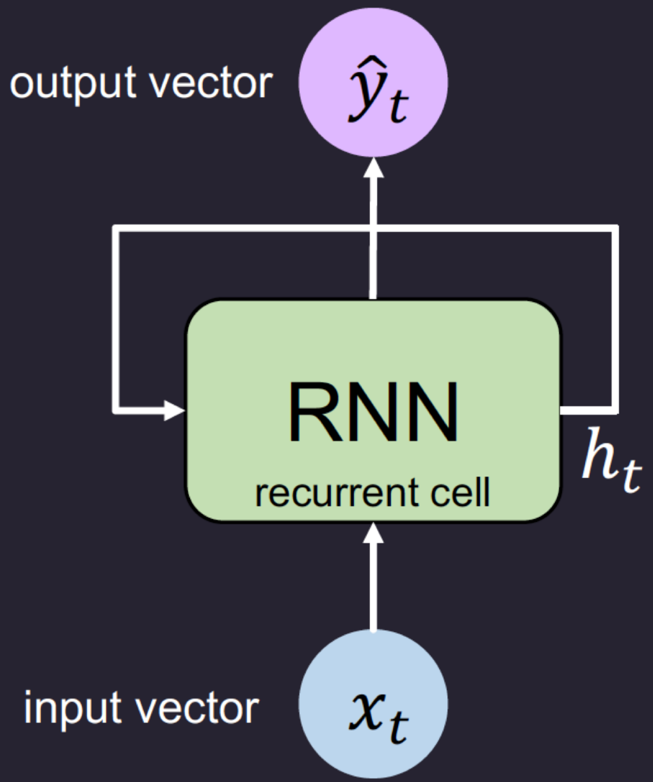


RNNs: computational graph across time



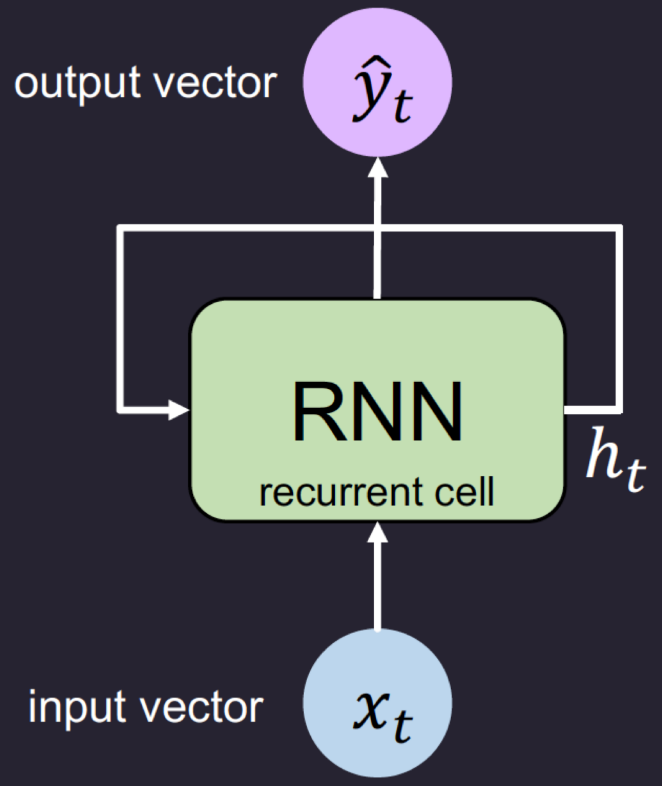
RNNs from Scratch

```
class MyRNNCell(tf.keras.layers.Layer):
  def __init__(self, rnn_units, input_dim, output_dim):
    super(MyRNNCell, self).__init__()

    # Initialize weight matrices
    self.W_xh = self.add_weight([rnn_units, input_dim])
    self.W_hh = self.add_weight([rnn_units, rnn_units])
    self.W_hy = self.add_weight([output_dim, rnn_units])

    # Initialize hidden state to zeros
    self.h = tf.zeros([rnn_units, 1])

  def call(self, x):
    # Update the hidden state
    self.h = tf.math.tanh( self.W_hh * self.h + self.W_xh * x )

    # Compute the output
    output = self.W_hy * self.h

    # Return the current output and hidden state
    return output, self.h
```



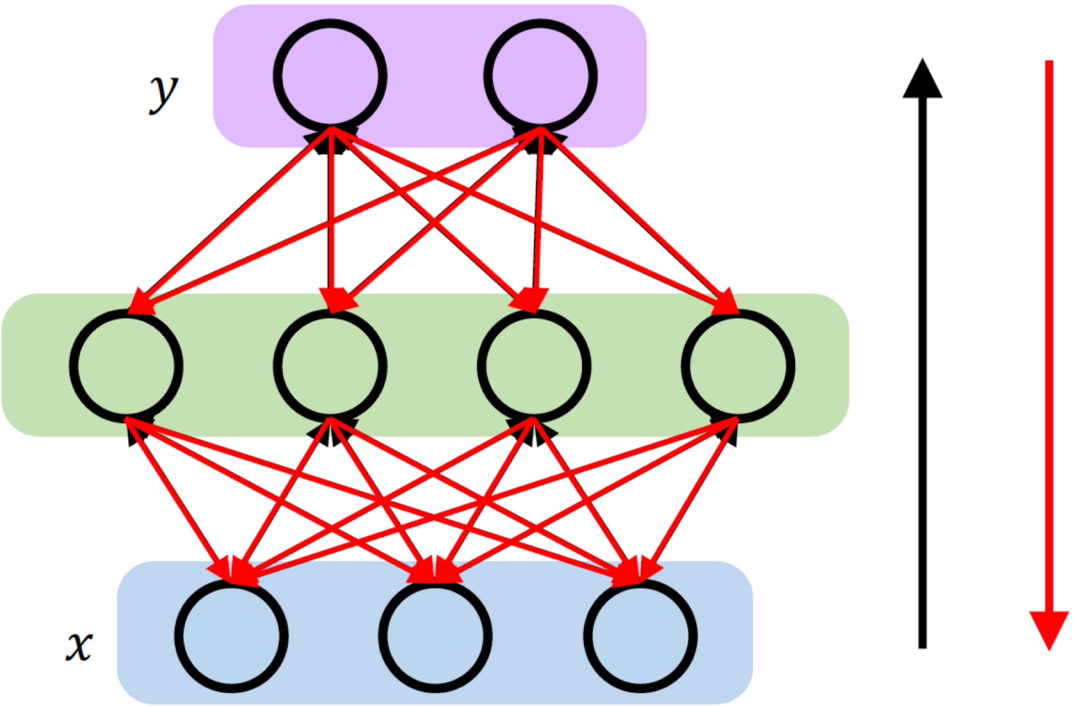
RNN Implementation in TensorFlow

```
tf.keras.layers.SimpleRNN(rnn_units)
```



Backpropagation Through Time(BPTT)

Recall: backpropagation in feed forward models

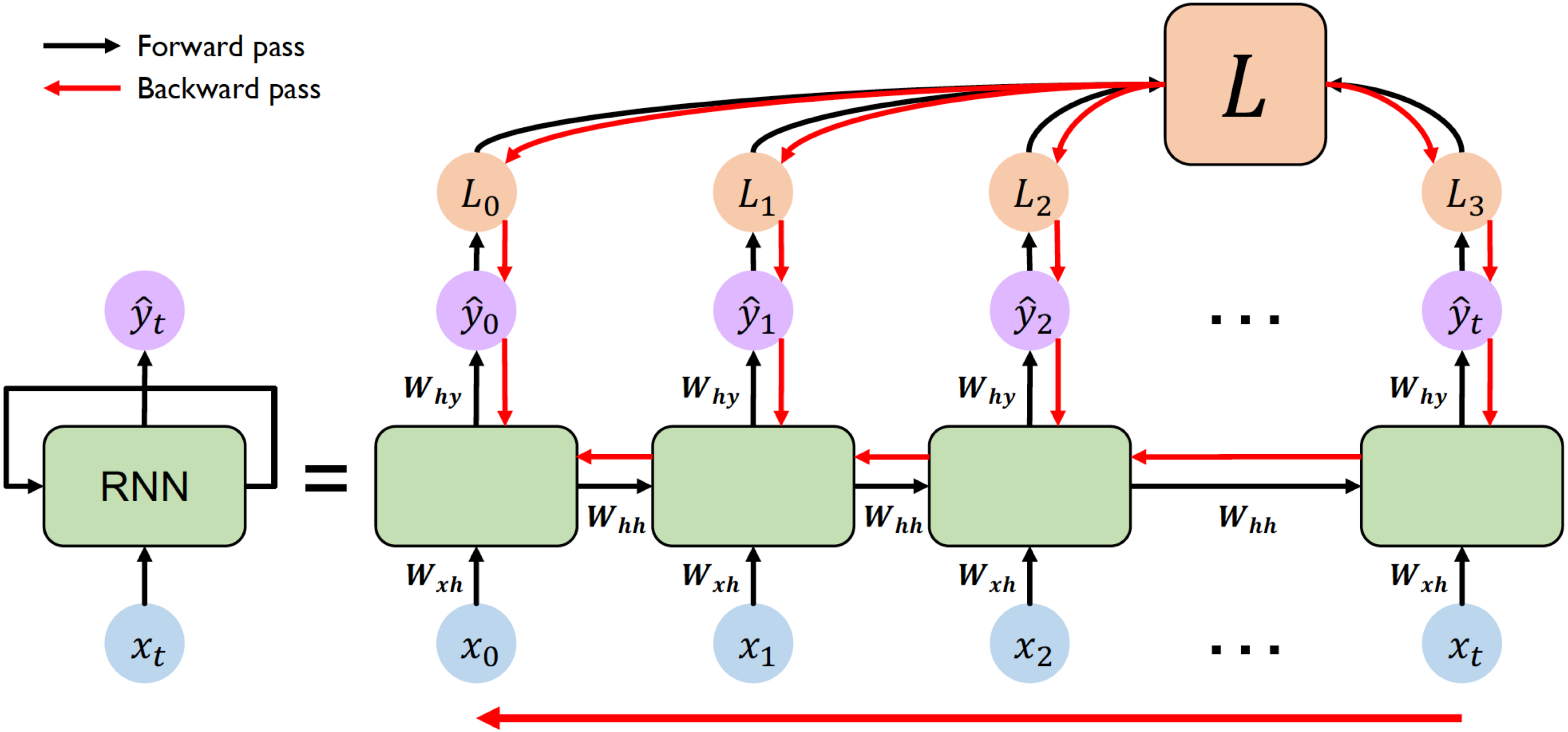


Backpropagation algorithm:

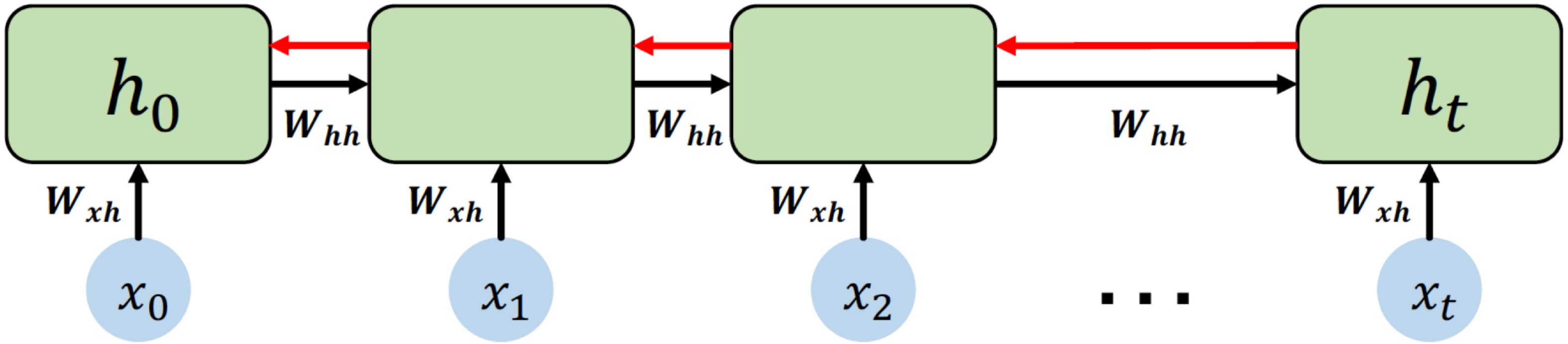
1. Take the derivative (gradient) of the loss with respect to each parameter
2. Shift parameters in order to minimize loss

[3]

RNNs: Backpropagation Through Time



Standard RNN gradient flow: exploding gradients



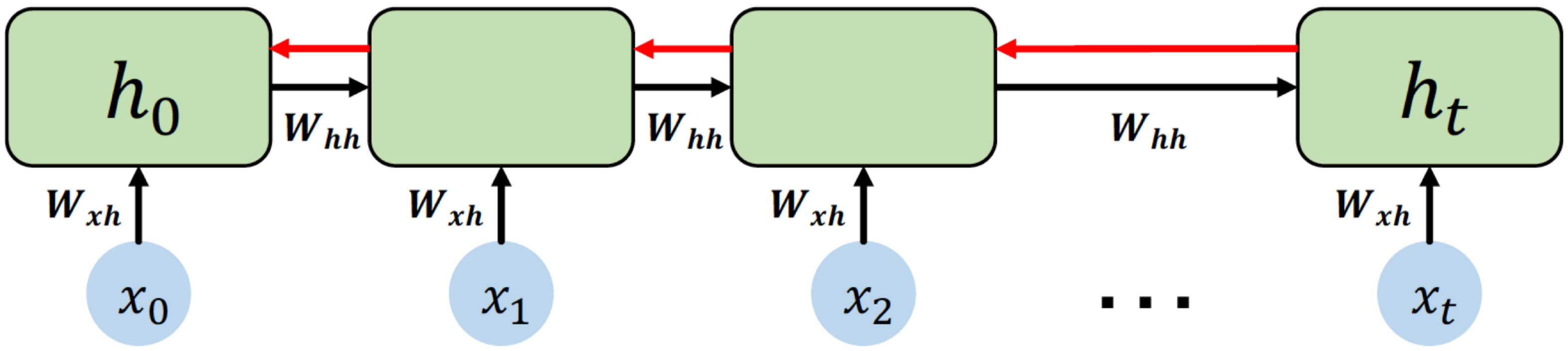
Computing the gradient wrt h_0 involves **many factors of W_{hh}** (and repeated f'!)

Many values > 1:
exploding gradients

Gradient clipping to scale big gradients

[1]

Standard RNN gradient flow: vanishing gradients



Computing the gradient wrt h_0 involves **many factors of W_{hh}** (and repeated f'!)

Largest singular value > 1:
exploding gradients

Gradient clipping to scale big gradients

Largest singular value < 1:
vanishing gradients

1. Activation function
2. Weight initialization
3. Network architecture

[1]

The problem of long-term dependencies

Why are vanishing gradients a problem?

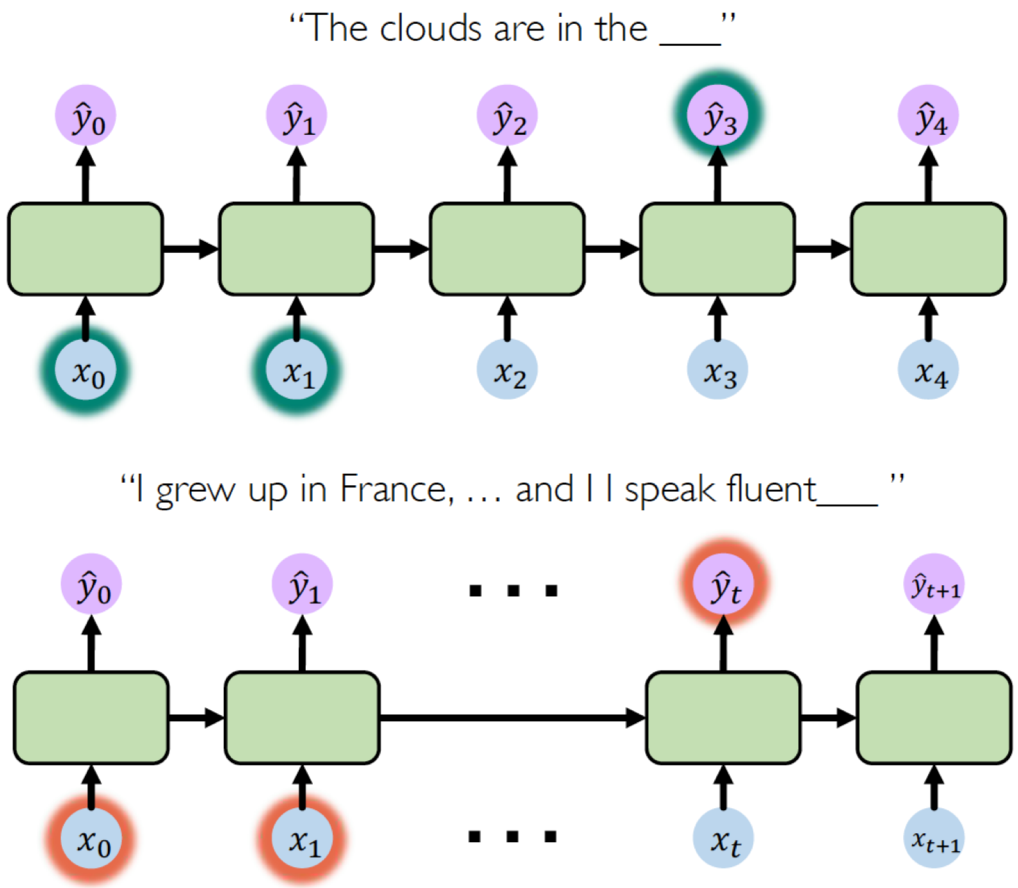
Multiply many **small numbers** together

↓

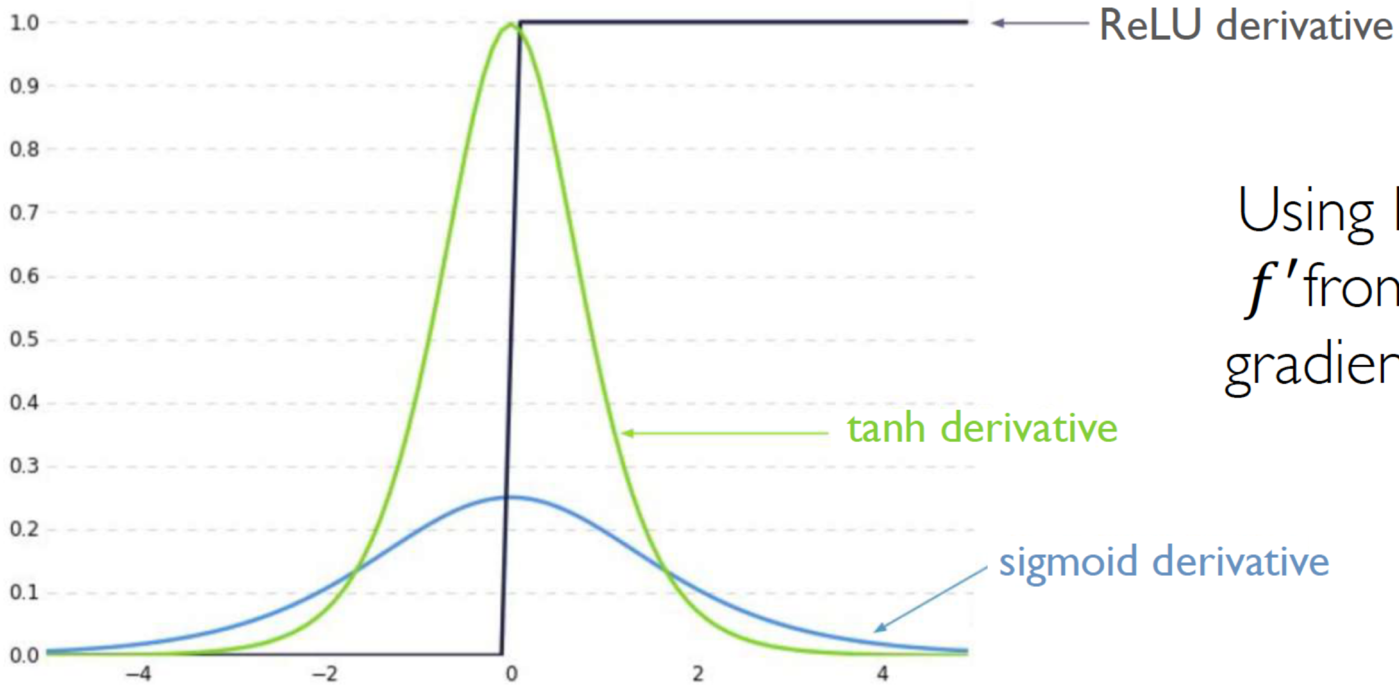
Errors due to further back time steps have smaller and smaller gradients

↓

Bias parameters to capture short-term dependencies



Trick #1: activation functions



Using ReLU prevents f' from shrinking the gradients when $x > 0$

Adapted from H. Suresh, 6.S191 2018

Trick #2: parameter initialization

Initialize **weights** to identity matrix

Initialize **biases** to zero

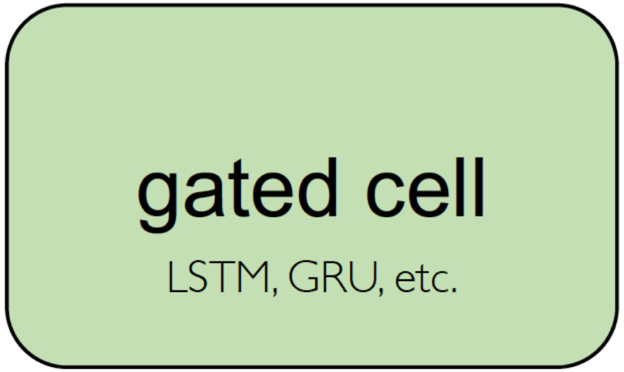
$$I_n = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

This helps prevent the weights from shrinking to zero.

Adapted from H. Suresh, 6.S191 2018

Solution #3: gated cells

Idea: use a more **complex recurrent unit with gates** to control what information is passed through



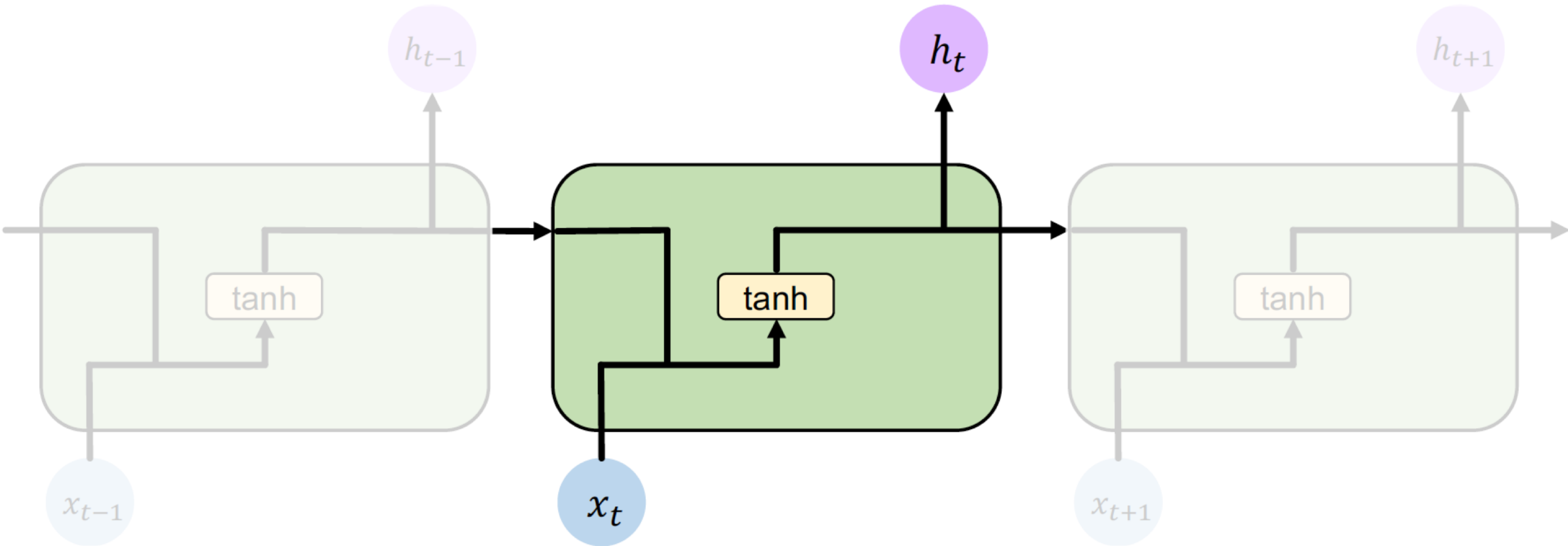
Long Short Term Memory (LSTMs) networks rely on a gated cell to track information throughout many time steps.

Adapted from H. Suresh, 6.S191 2018

Long Short Term Memory (LSTM) Networks

Standard RNN

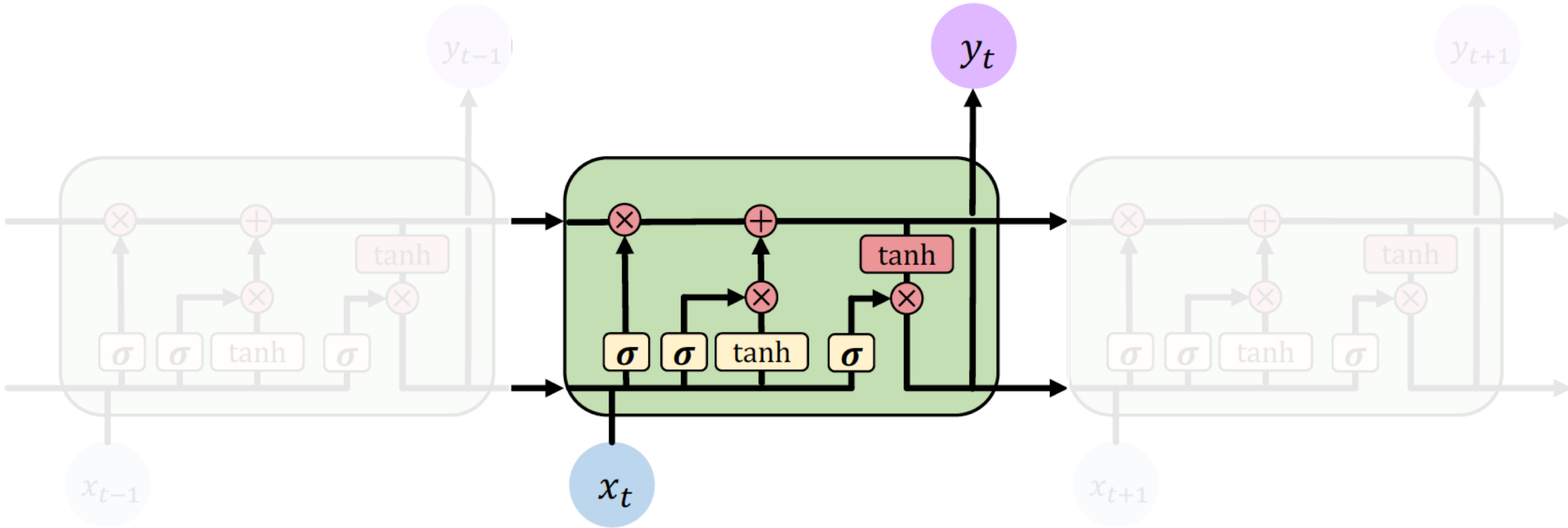
In a standard RNN, repeating modules contain a **simple computation node**



[2]

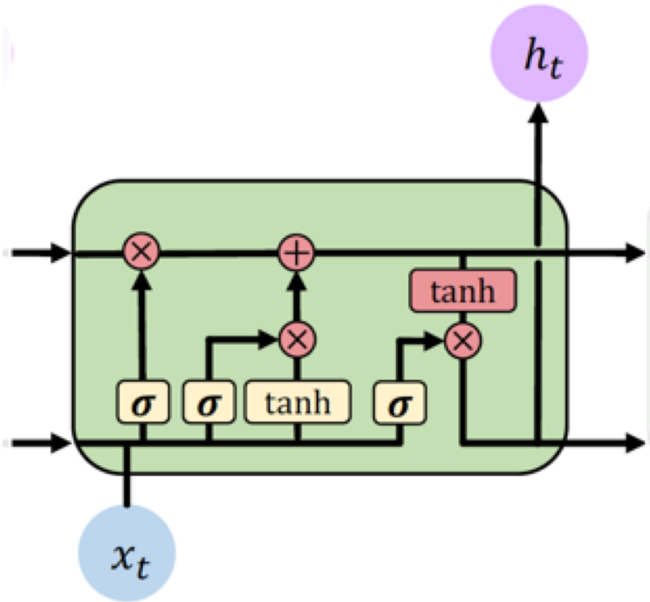
Long Short Term Memory (LSTMs)

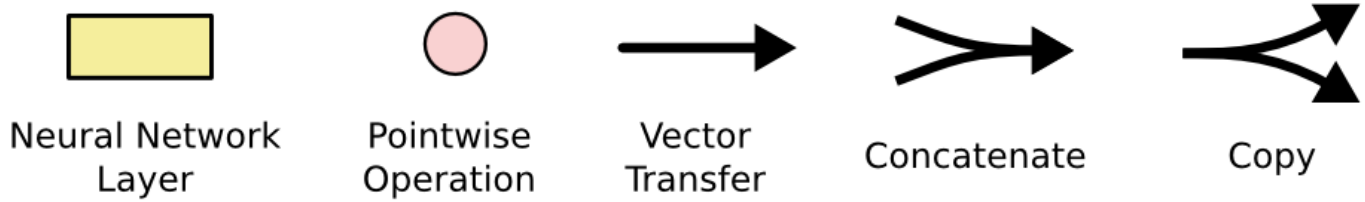
LSTM modules contain **computational blocks** that **control information flow**



LSTM cells are able to track information throughout many timesteps

LSTM components

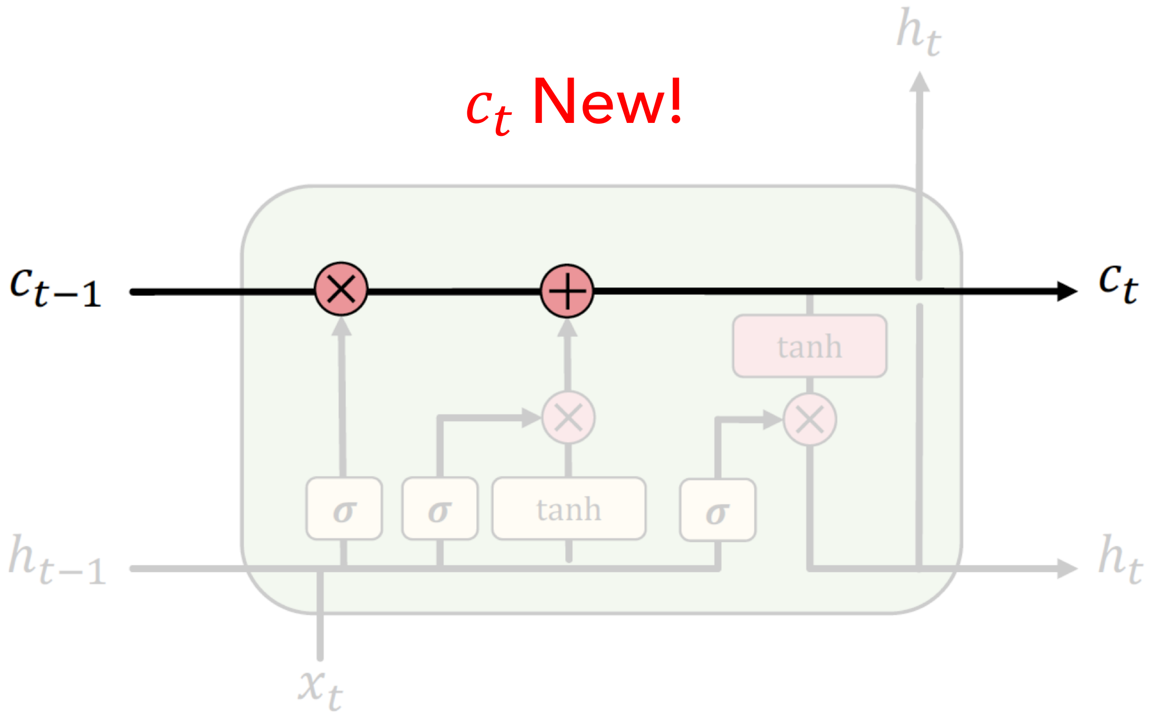




- <u>Yellow boxes</u>: learned neural network layers.
- <u>Pink circles</u>: pointwise operations (ex vector addition)
- <u>Lines merging</u>: concatenation
- <u>Line forking</u>: copies go to different locations

Long Short Term Memory (LSTMs)

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

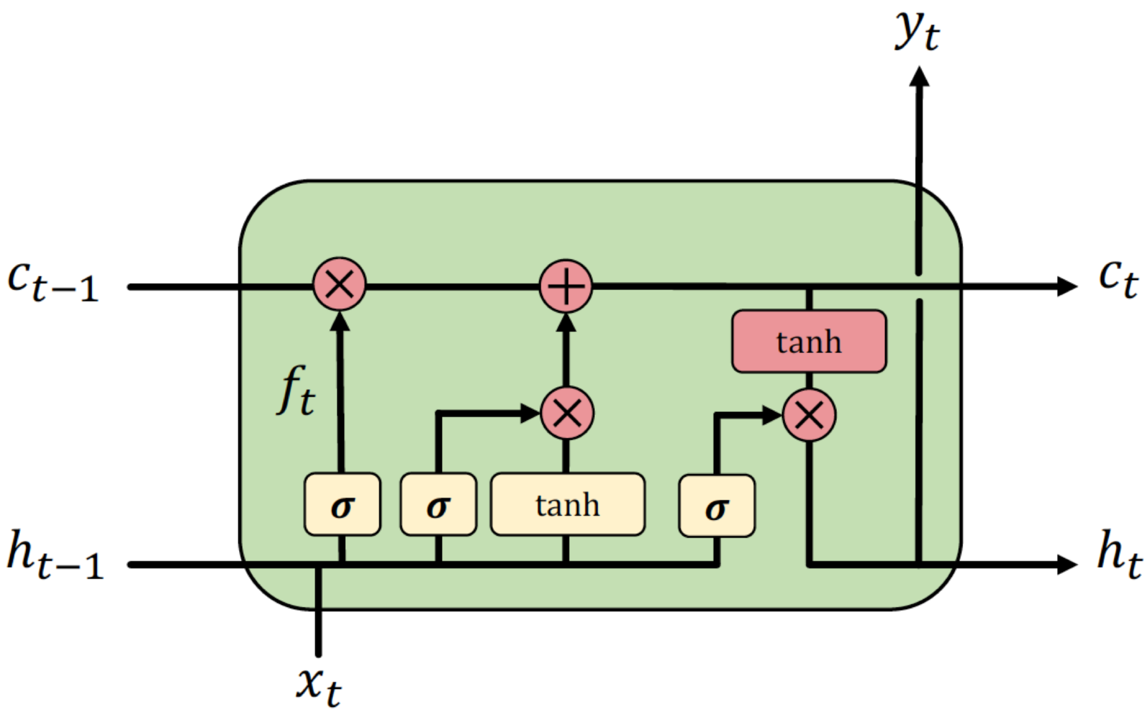


[2, 5]

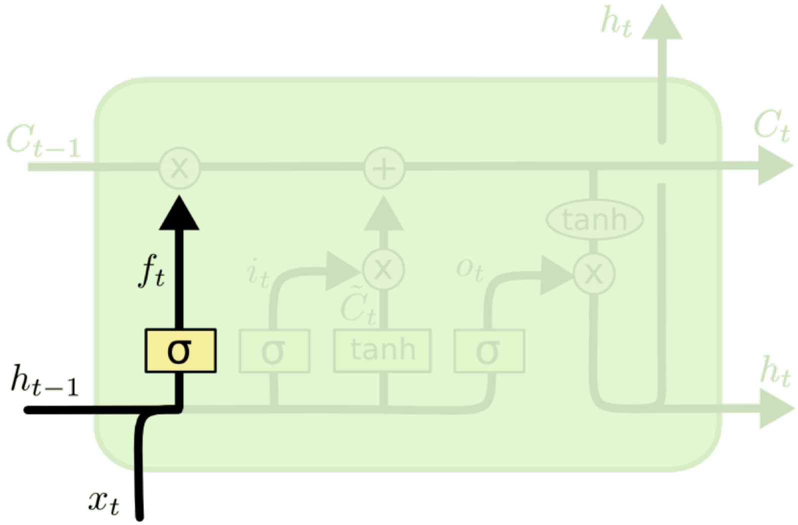
Long Short Term Memory (LSTMs)

How do LSTMs work?

1) Forget 2) Store 3) Update 4) Output



Forget gate layer



Forget gate:

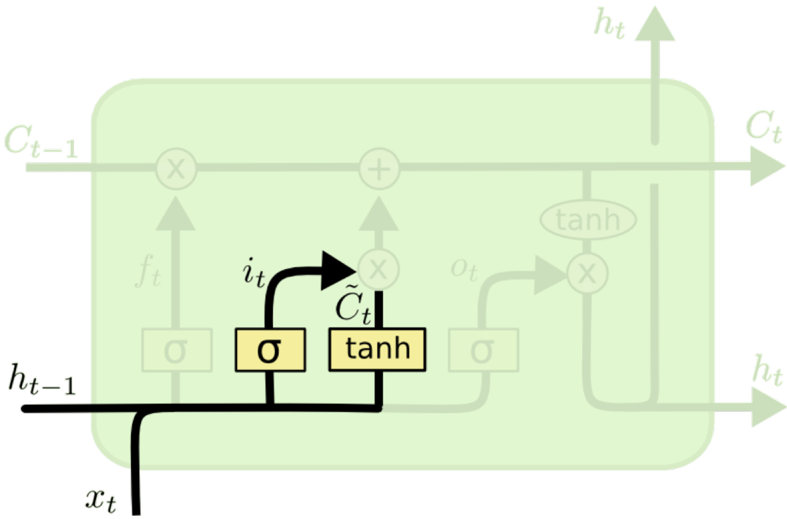
- it controls which information to remember and which to forget
- it can also reset the cell state

Mathematically:

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

- a <u>Sigmoid</u> σ
- <u>Input</u>: h_{t-1} and x_t
- <u>Output</u>: nb. between 0 and 1:
  - 0: forget
  - 1: remember

Input gate layer



Input gate:

- decide what new information to store in the cell state
- 2 parts

Mathematically:

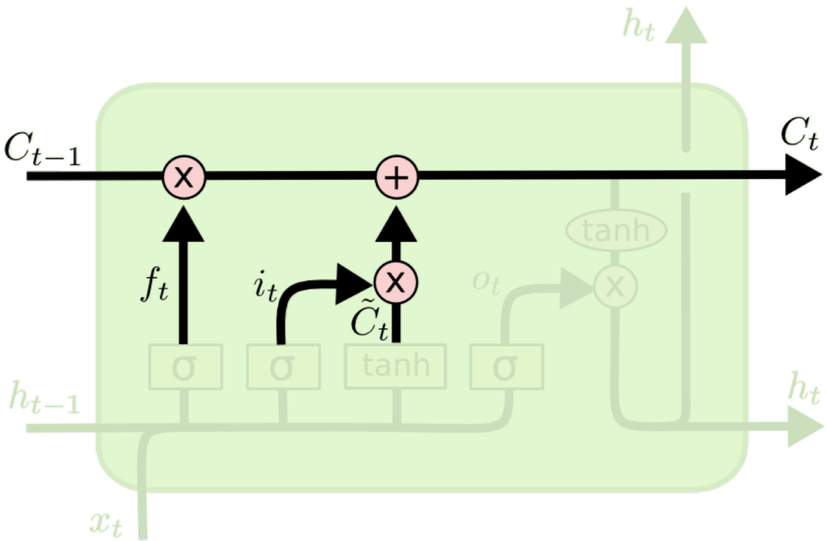
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

- A *tanh*
  (create a new candidate to be possibly added to the state)

- a Sigmoid σ
  (to decide which values to update)

Cell State update



Cell state update:

- Update C_{t-1} to C_t
- Apply the decision taken in the previous step

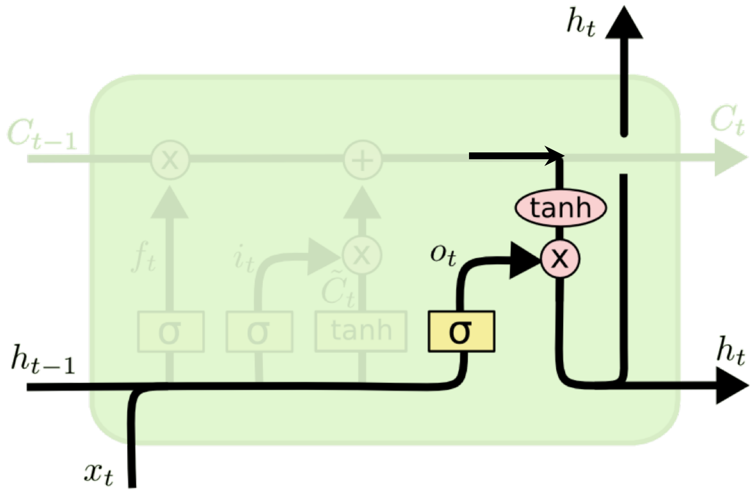
Mathematically:

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

Forget old irrelevant information

Add the weighted new candidate

Output gate layer



Mathematically:

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

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

- a <u>Sigmoid</u> σ
  (to decide which part of the cell state to output)

- A <u>*tanh*</u>
  (cell state pushed between -1 1)

Output gate:

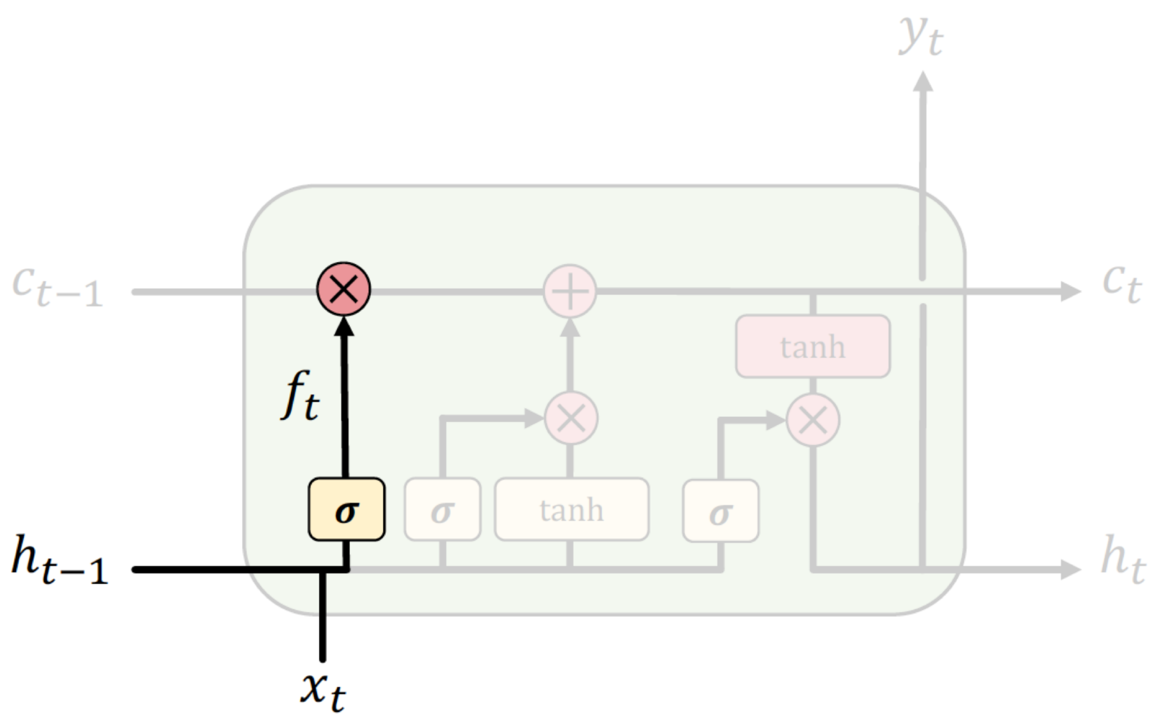
- Output: filtered version of the cell state

- 2 parts

Long Short Term Memory (LSTMs)

1) Forget 2) Store 3) Update 4) Output

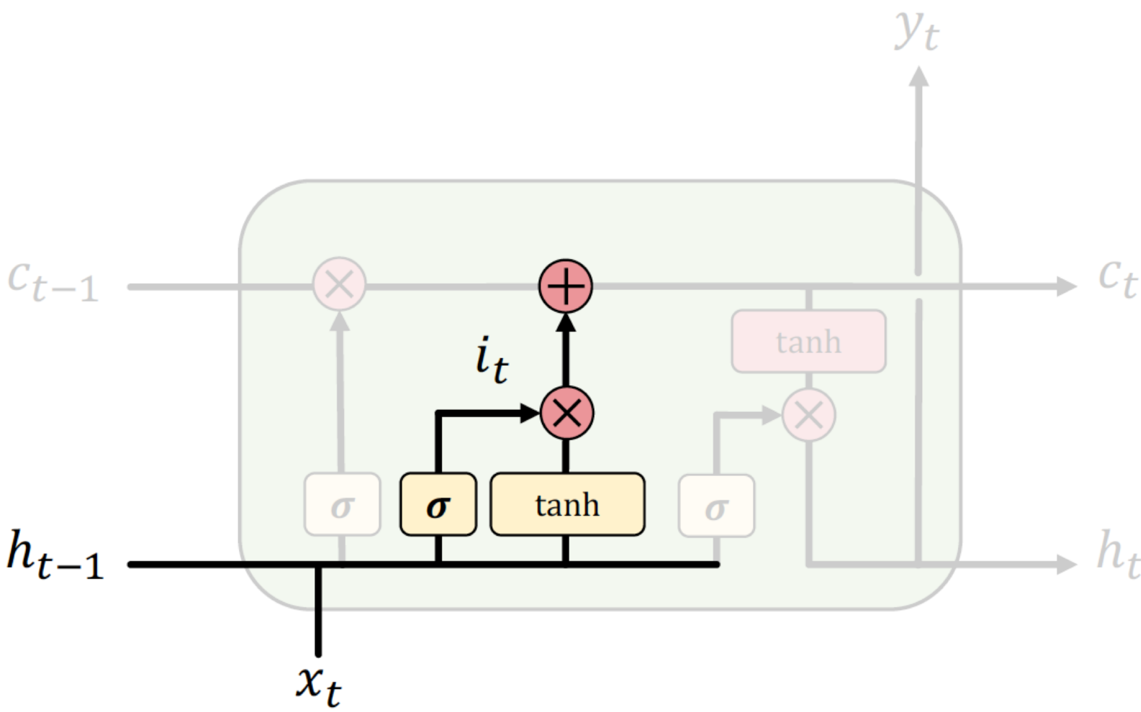
LSTMs **forget irrelevant** parts of the previous state



Long Short Term Memory (LSTMs)

1) Forget **2) Store** 3) Update 4) Output

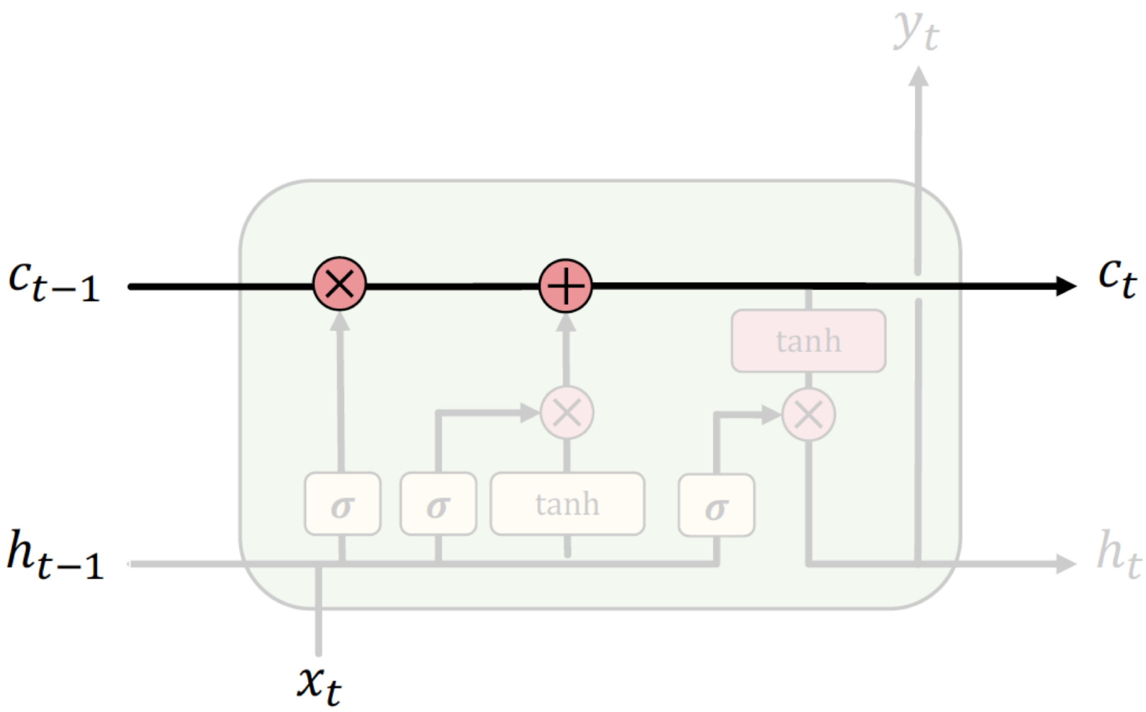
LSTMs **store relevant** new information into the cell state



Long Short Term Memory (LSTMs)

1) Forget 2) Store **3) Update** 4) Output

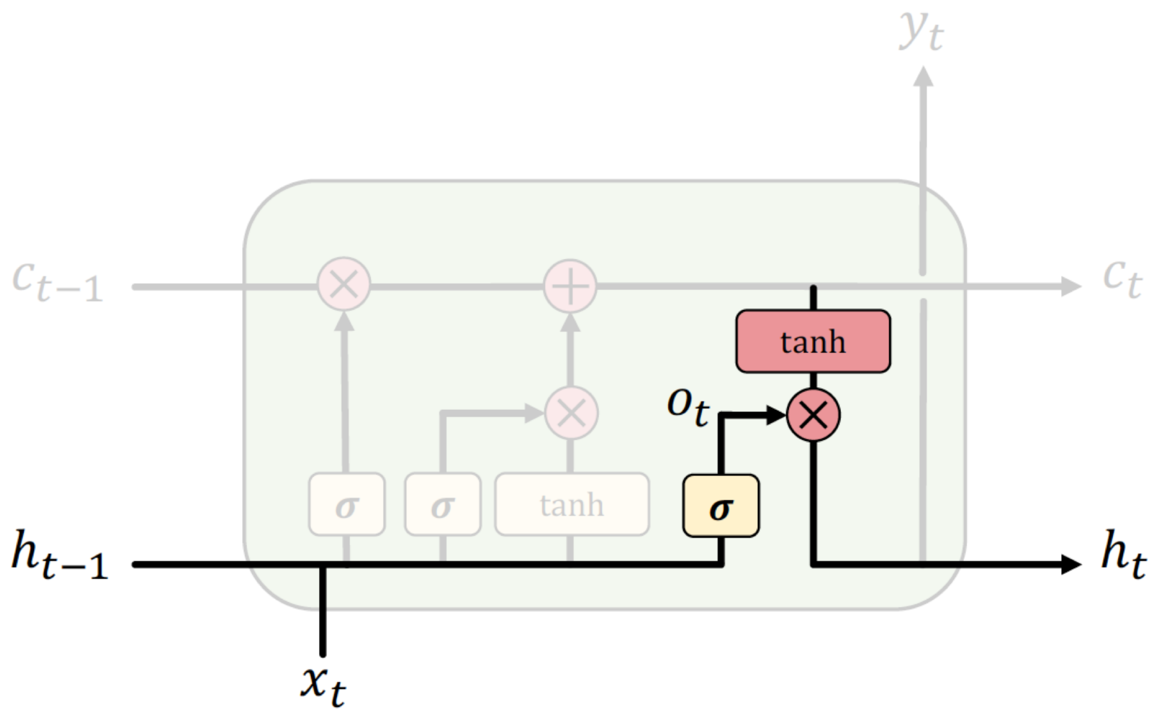
LSTMs **selectively update** cell state values



Long Short Term Memory (LSTMs)

1) Forget 2) Store 3) Update **4) Output**

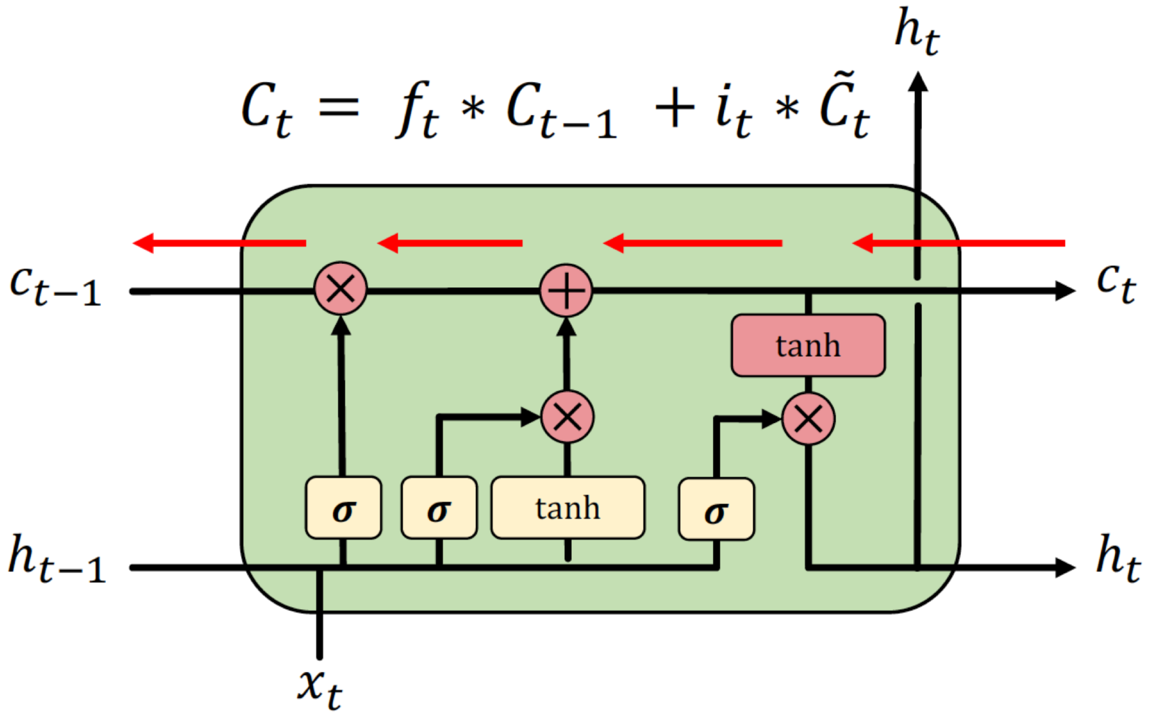
The **output gate** controls what information is sent to the next time step



LSTM gradient flow

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

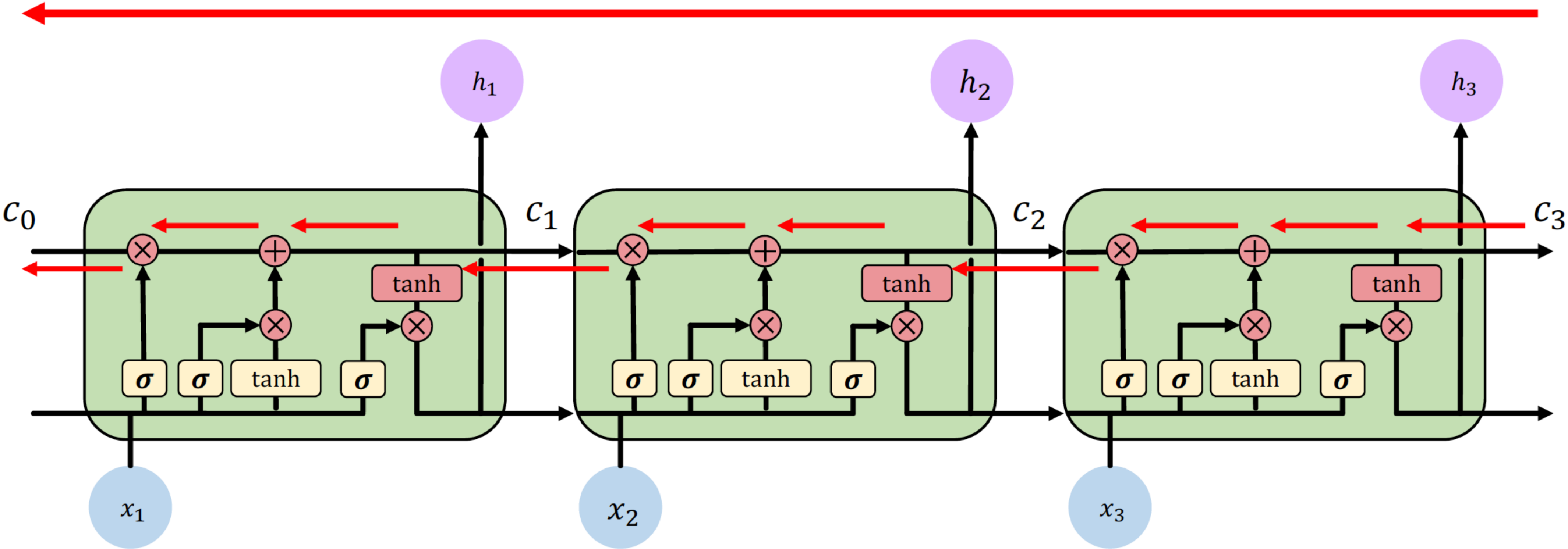
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



[2, 5]

LSTM gradient flow

Uninterrupted gradient flow!



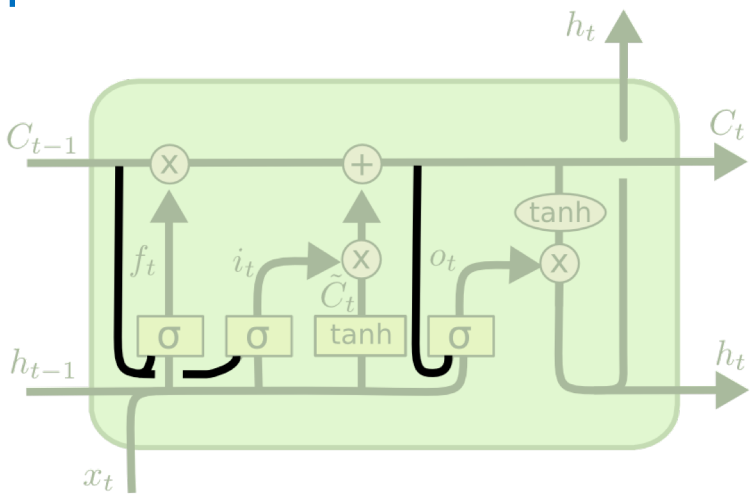
LSTMs: key concepts

1. Maintain a **separate cell state** from what is outputted
2. 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
3. Backpropagation from c_t to c_{t-1} doesn't require matrix multiplication: **uninterrupted gradient flow**

Variants on Long Short Term Memory

Many variants, almost in each paper

- Peephole connections



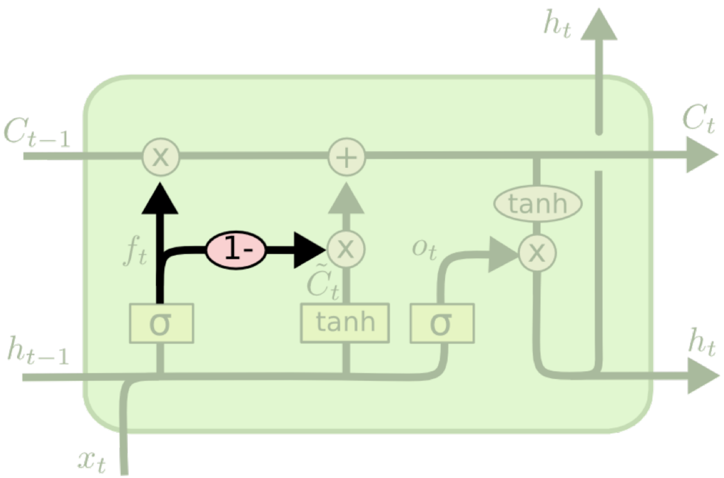
$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] + b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] + b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] + b_o\right)$$

The gate layers look at the cell state

Variants on Long Short Term Memory

Many variants, almost in each paper

- Tie connections



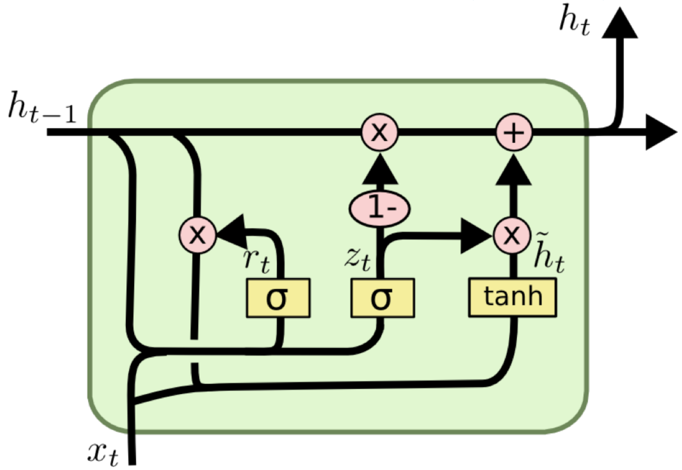
$$C_t = f_t * C_{t-1} + (\mathbf{1} - \boldsymbol{f_t}) * \tilde{C}_t$$

- Forget when we input something in its place.
- Input new values to the state when we forget something older.

Variants on Long Short Term Memory

Many variants, almost in each paper

- Gated Recurrent Unit (GRU)



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- "Update gate": combines forget and input gates
- Merges Cell and hidden states

Variants on Long Short Term Memory

Many variants, almost in each paper

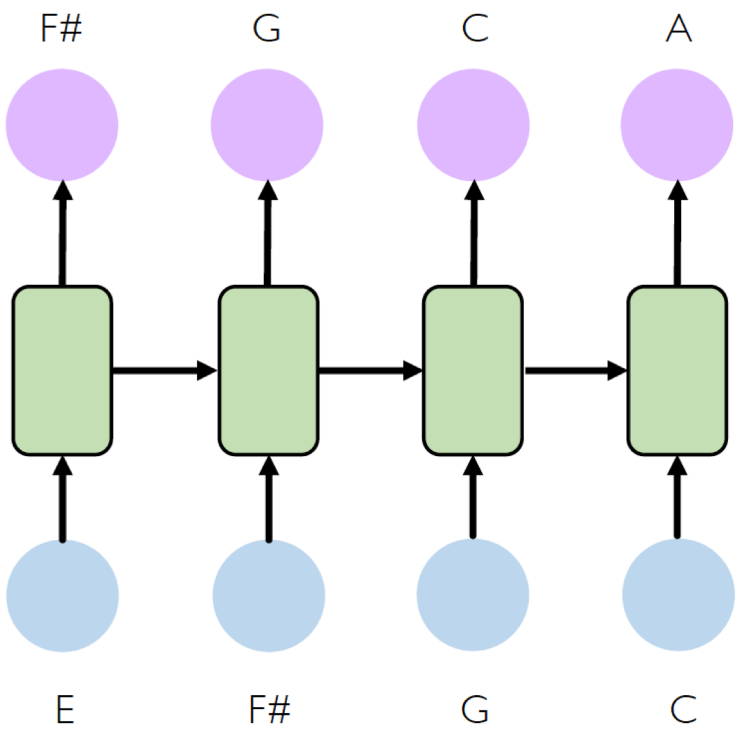
- Depth Gated RNNs
- Clockwork RNNs
- …

Which one is the best?
Comparisons in:

- Greff, Klaus, et al. "LSTM: A search space odyssey." *IEEE TNNLS*
- Jozefowicz, Rafal, et al. "An empirical exploration of recurrent network architectures." In: *Int'l conference on machine learning*. 2015. p. 2342-2350.

RNN Applications

Example task: music generation

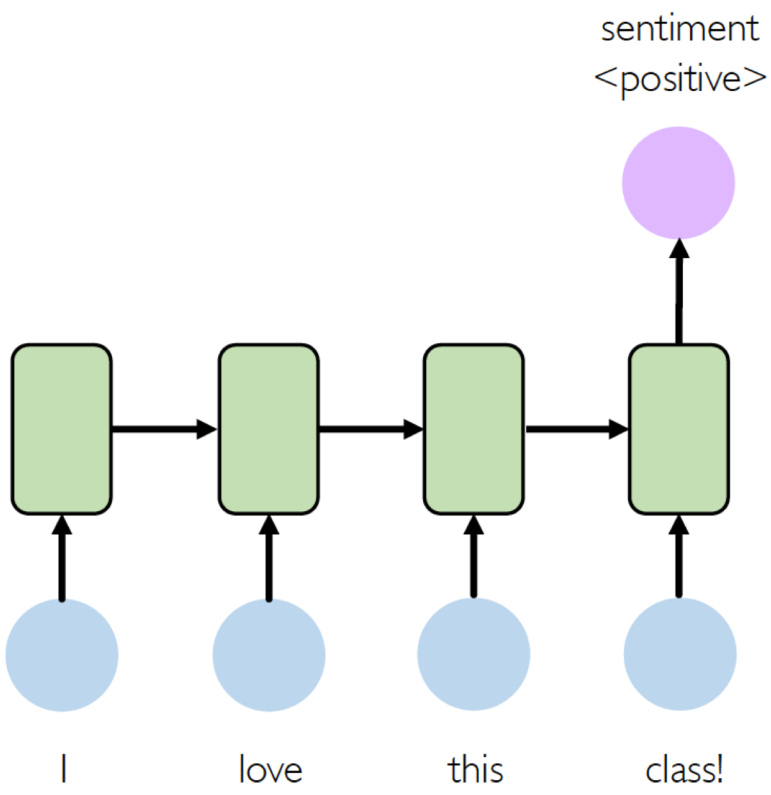


Input: sheet music

Output: next character in sheet music

Adapted from H. Suresh, 6.S191 2018

Example Task: Sentiment Classification

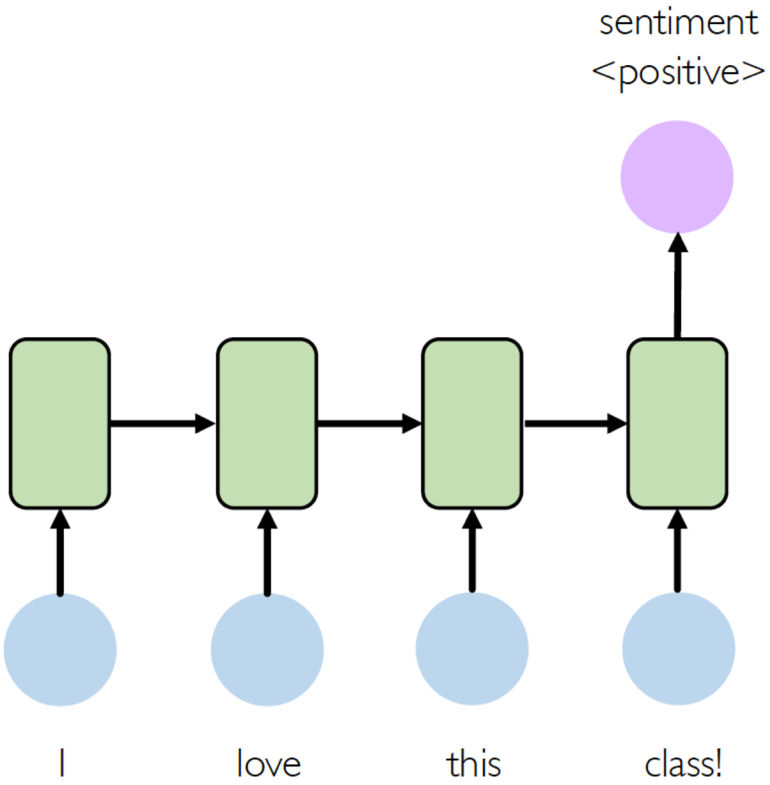


Input: sequence of words

Output: probability of having positive sentiment

```
loss = tf.nn.softmax_cross_entropy_with_logits(y, predicted)
```

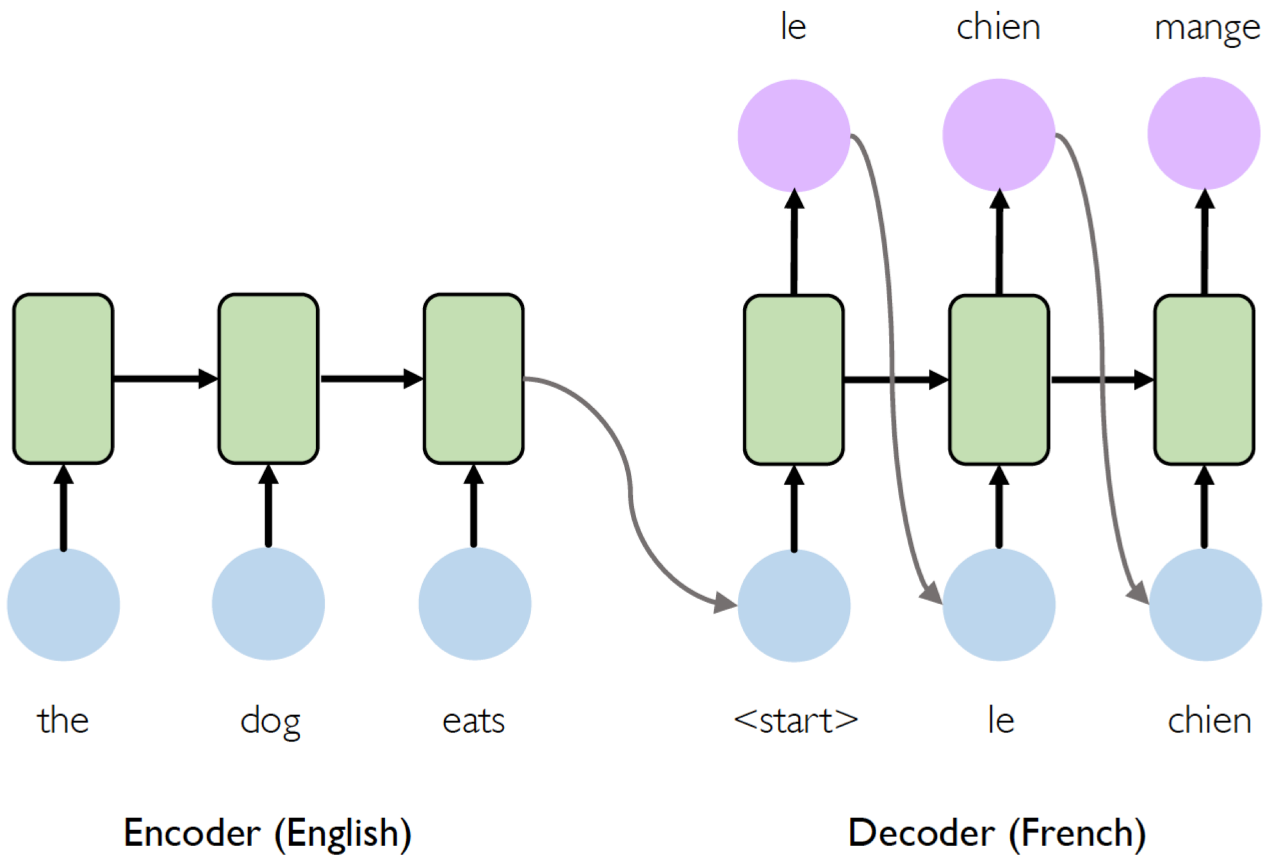
Example task: sentiment classification



Tweet sentiment classification

Adapted from H. Suresh, 6.S191 2018

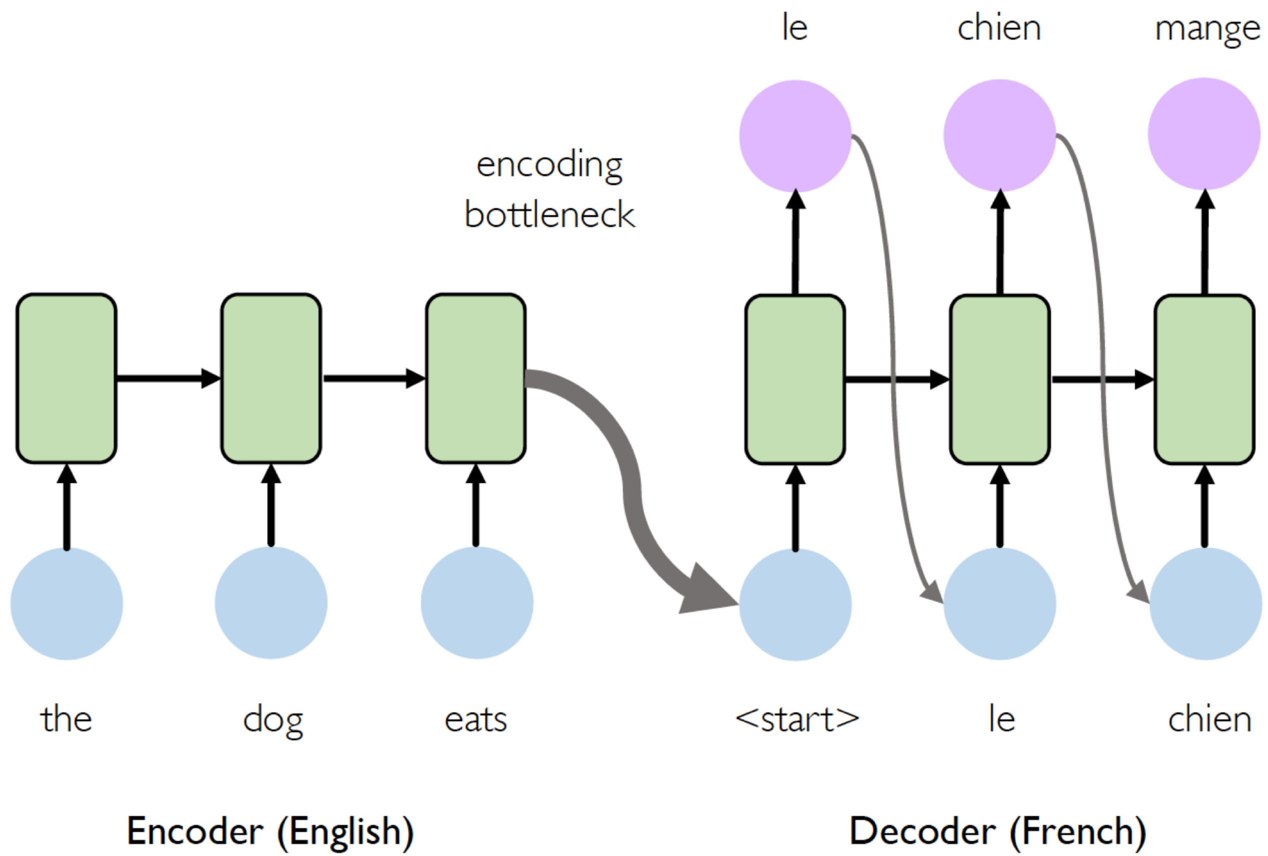
Example task: machine translation



Adapted from H. Suresh, 6.S191 2018

[8,9]

Example task: machine translation

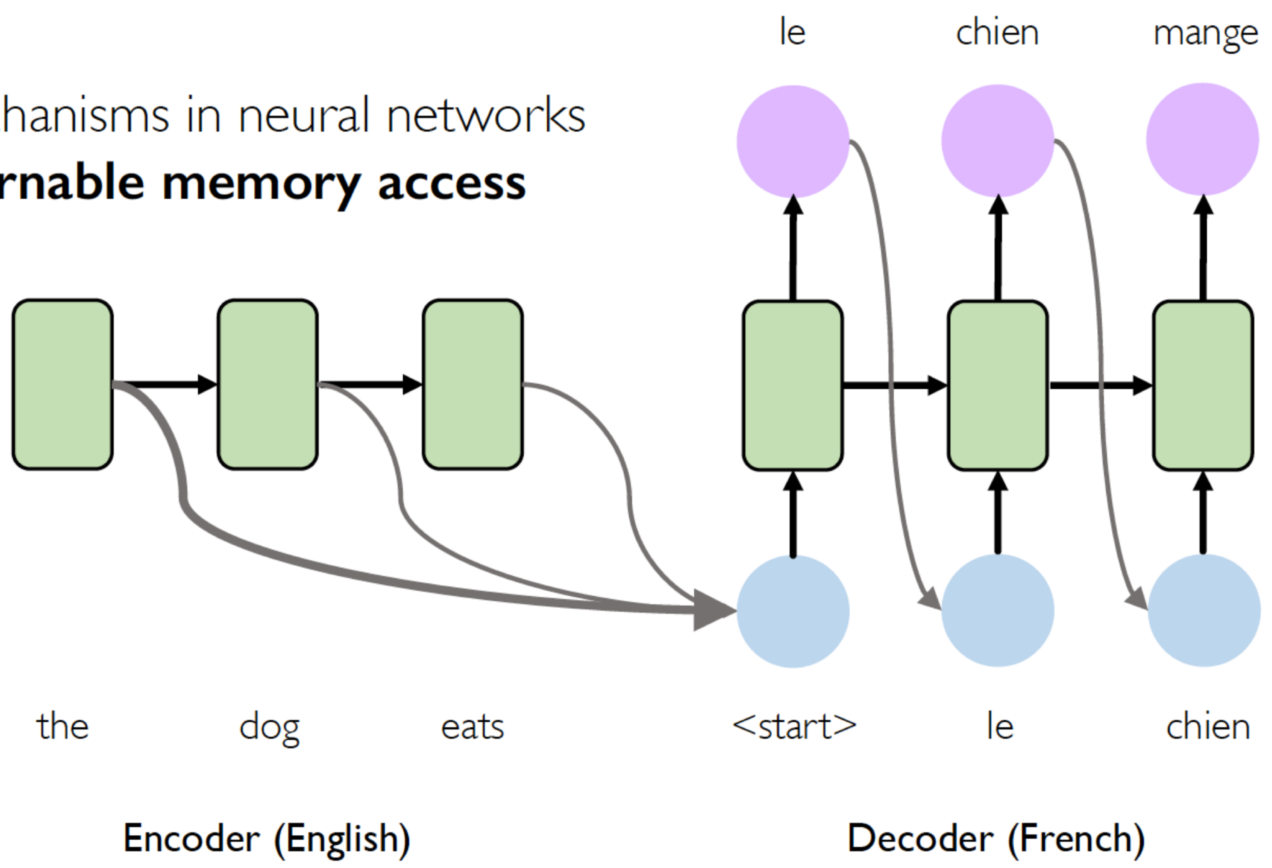


Adapted from H. Suresh, 6.S191 2018

[8,9]

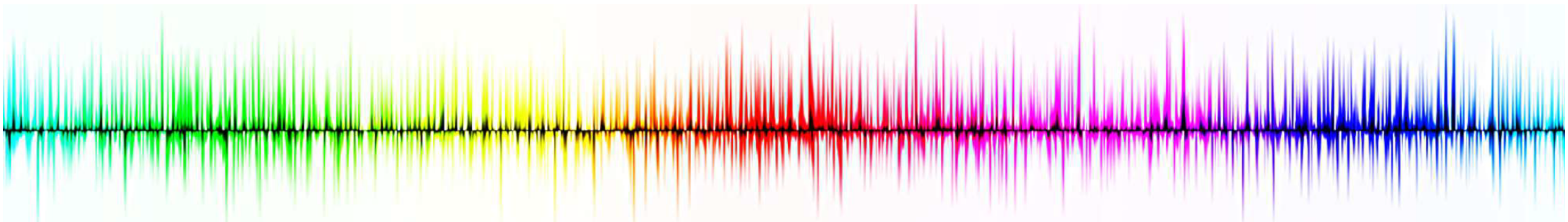
Attention Mechanisms

Attention mechanisms in neural networks provide **learnable memory access**



Recurrent neural networks (RNNs)

1. RNNs are well suited for **sequence modeling** tasks
2. Model sequences via a **recurrence relation**
3. Training RNNs with **backpropagation through time**
4. Gated cells like **LSTMs** let us model **long-term dependencies**
5. Models for **music generation**, classification, machine translation



References

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

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