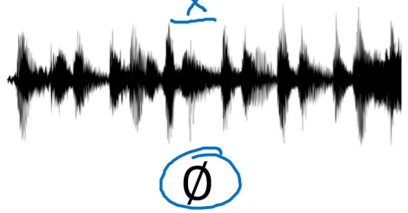
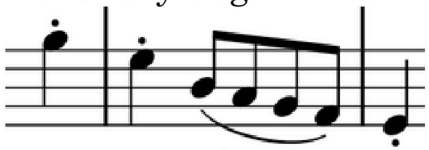
deeplearning.ai

Recurrent Neural Networks

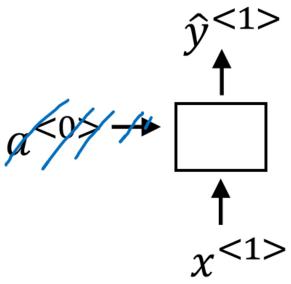
Why sequence models?

Examples of sequence data

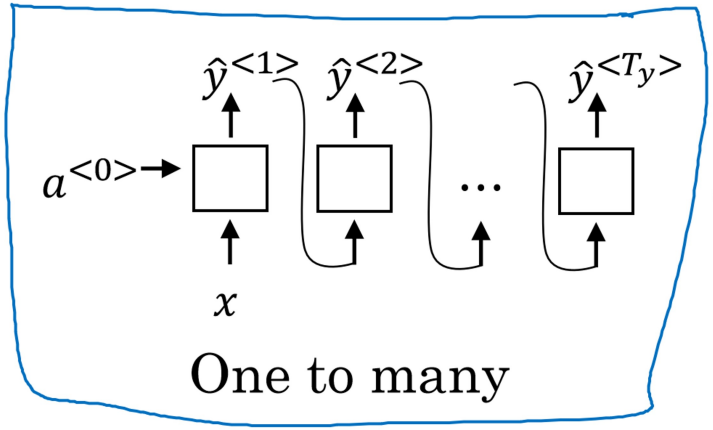
Speech recognition		"The quick brown fox jumped over the lazy dog."
Music generation		
Sentiment classification	"There is nothing to like in this movie."	★☆☆☆☆
DNA sequence analysis	AGCCCCTGTGAGGAACTAG	AGCCCCTGTGAGGAACTAG
Machine translation	Voulez-vous chanter avec moi?	Do you want to sing with me?
Video activity recognition		Running
Name entity recognition	Yesterday, Harry Potter met Hermione Granger.	Yesterday, Harry Potter met Hermione Granger.

Andrew Ng

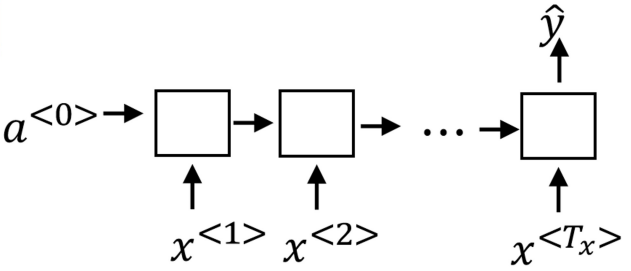
Summary of RNN types



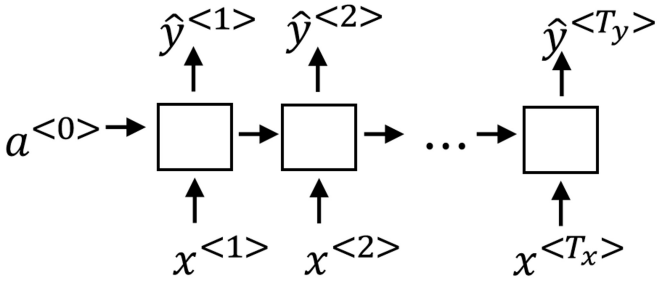
One to one



One to many

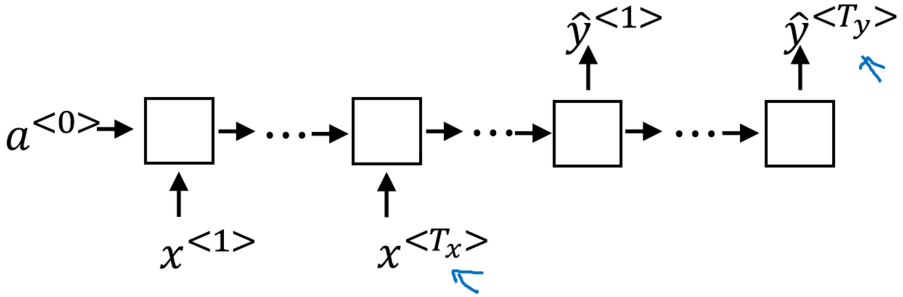


Many to one



Many to many

$T_x = T_y$



Many to many

Andrew Ng

deeplearning.ai

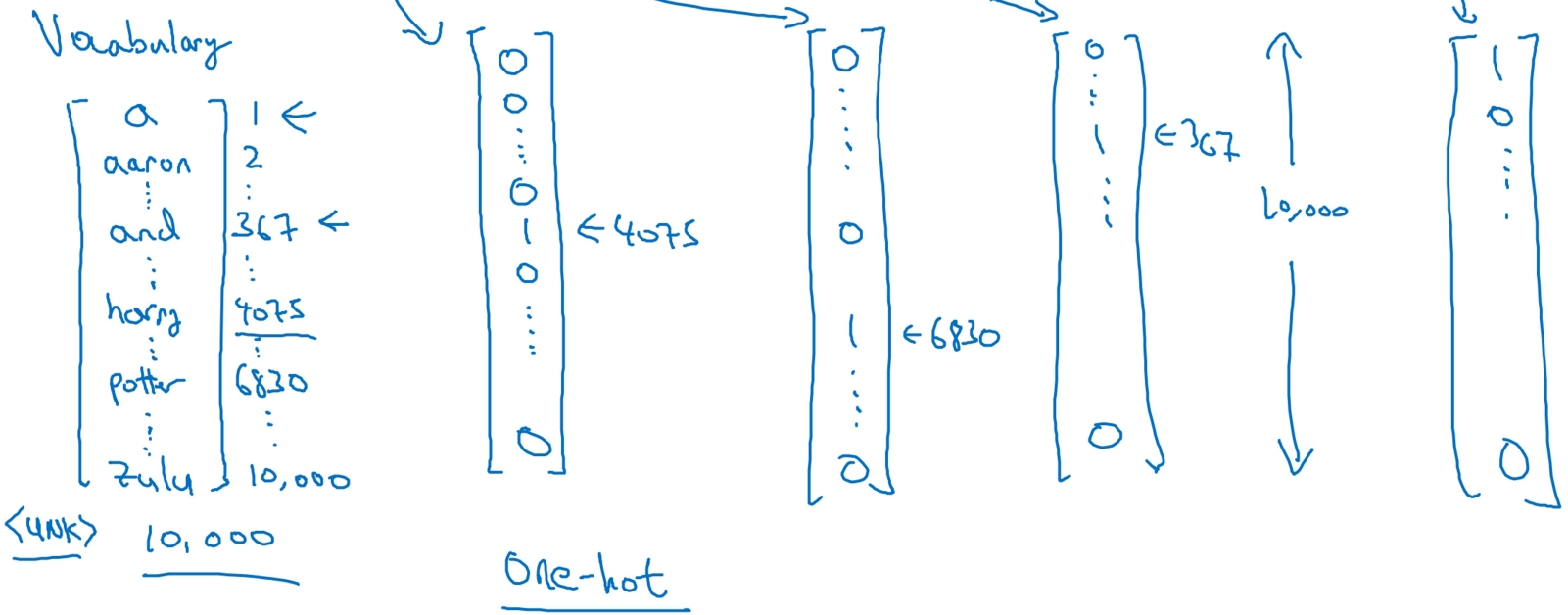
Recurrent Neural Networks

Notation

Representing words

x: Harry Potter and Hermione Granger invented a new spell.

$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ … $x^{<7>}$ $x^{<9>}$



Andrew Ng

Representing words

x: Harry Potter and Hermione Granger invented a new spell.

$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ … $x^{<9>}$

And = 367
Invented = 4700
A = 1
New = 5976
Spell = 8376
Harry = 4075
Potter = 6830
Hermione = 4200
Gran… = 4000

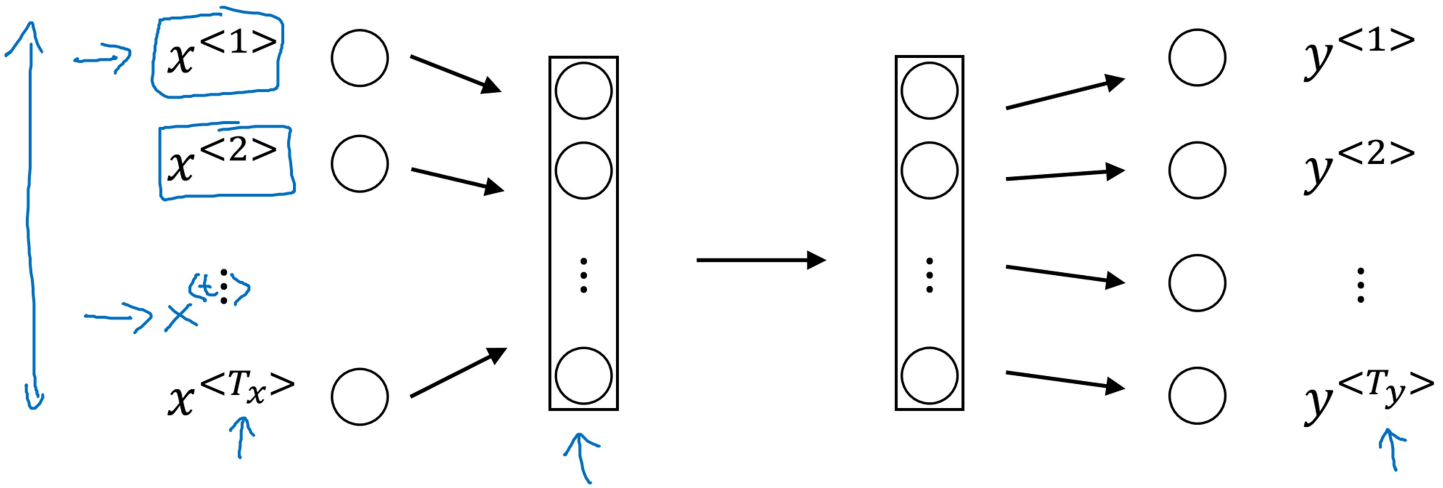
Andrew Ng

deeplearning.ai

Recurrent Neural Networks

Recurrent Neural Network Model

Why not a standard network?

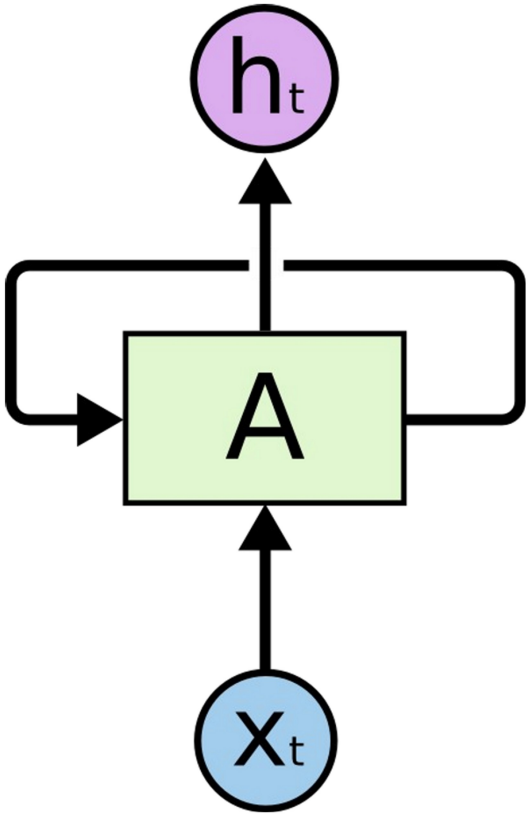


Problems:

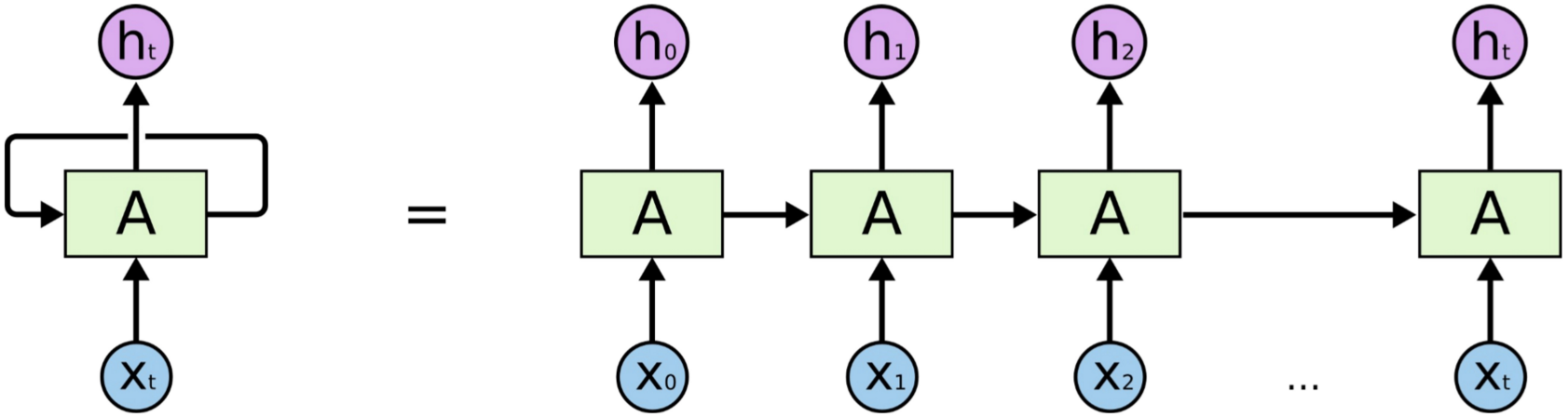
- Inputs, outputs can be different lengths in different examples.
- Doesn't share features learned across different positions of text.

Andrew Ng

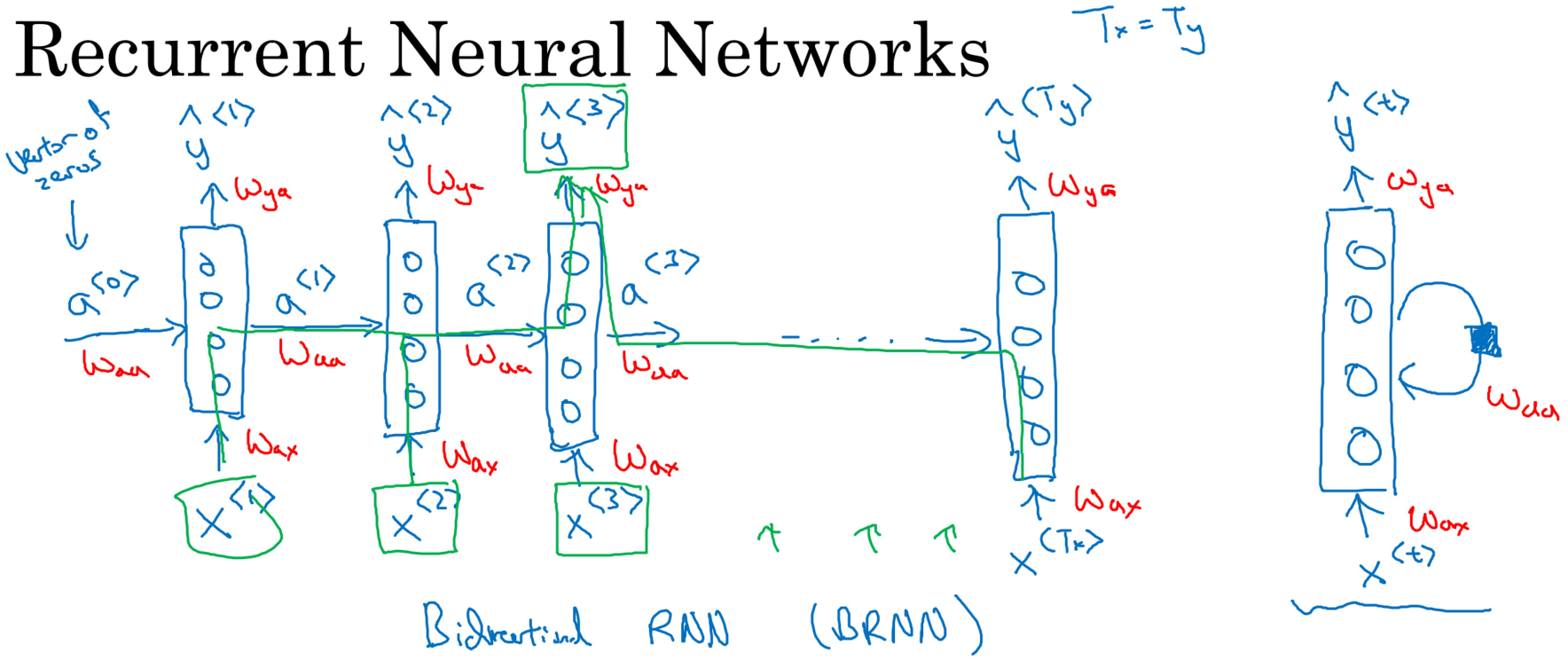
Recurrent Neural Networks



An unrolled recurrent neural network



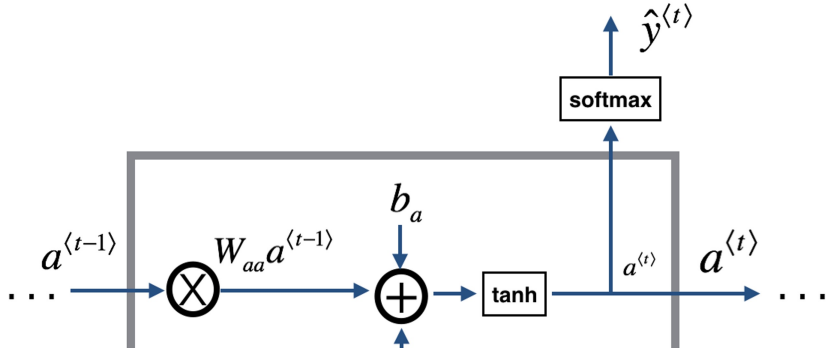
Recurrent Neural Networks

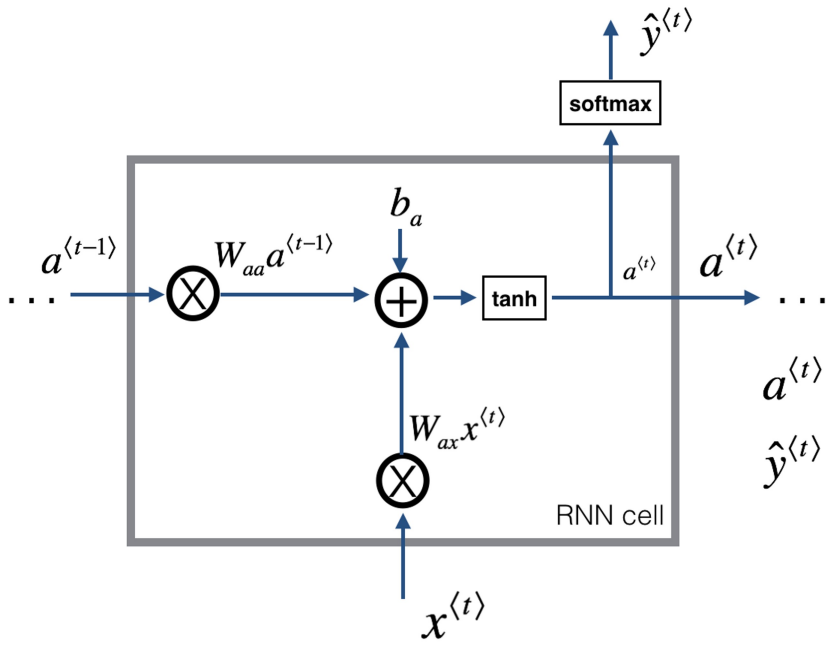


He said, "Teddy Roosevelt was a great President."

He said, "Teddy bears are on sale!"

Andrew Ng





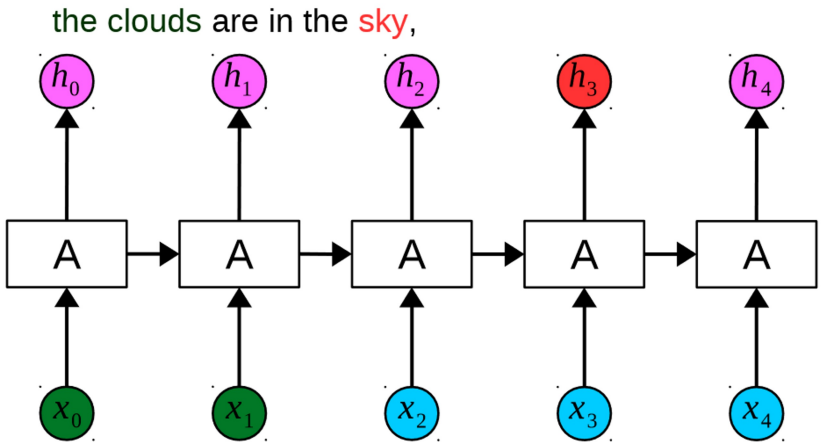
$$a^{\langle t \rangle} = \tanh(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b_a)$$

$$\hat{y}^{\langle t \rangle} = soft\max(W_{ya}a^{\langle t \rangle} + b_y)$$

The Problem of Long-Term Dependencies

RNN short-term dependencies

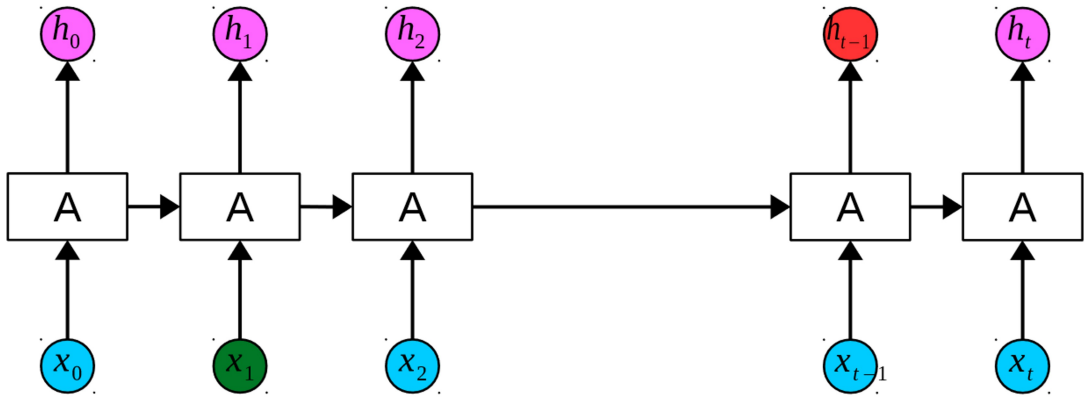
Language model trying to predict the next word based on the previous ones



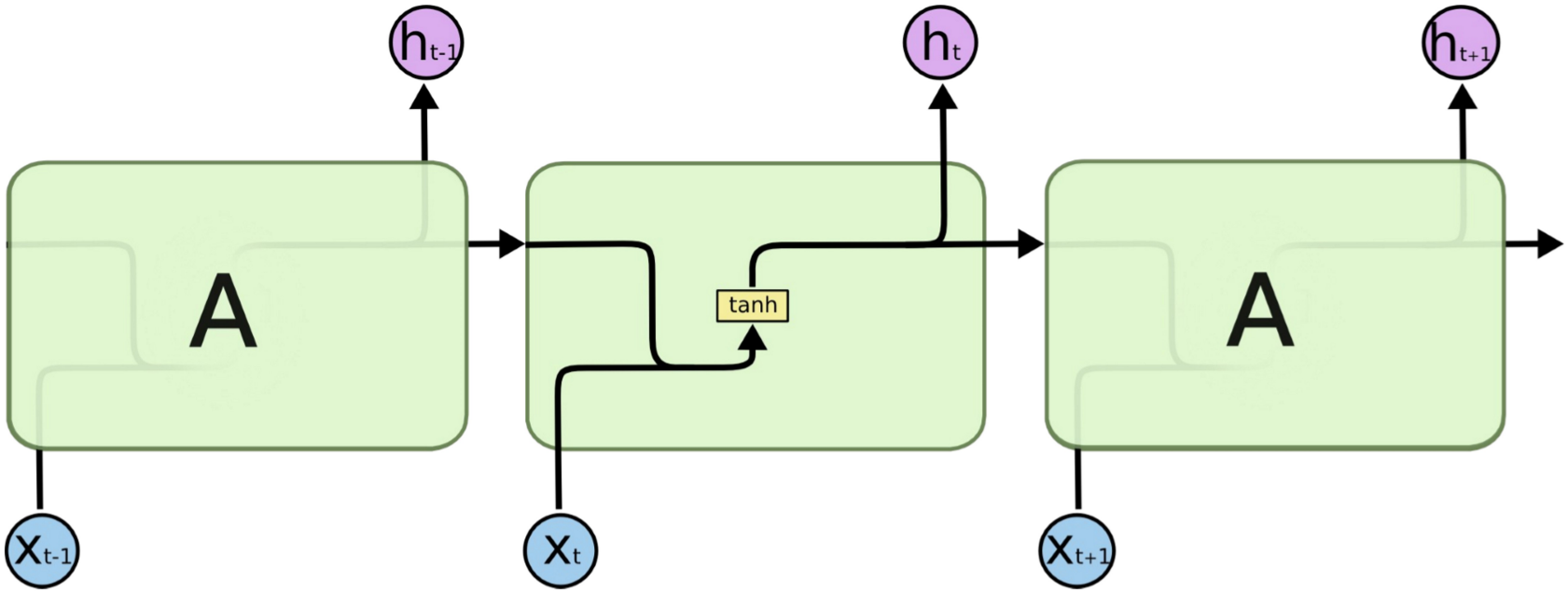
RNN long-term dependencies

Language model trying to predict the next word based on the previous ones

I grew up in India… I speak fluent Hindi.



Standard RNN



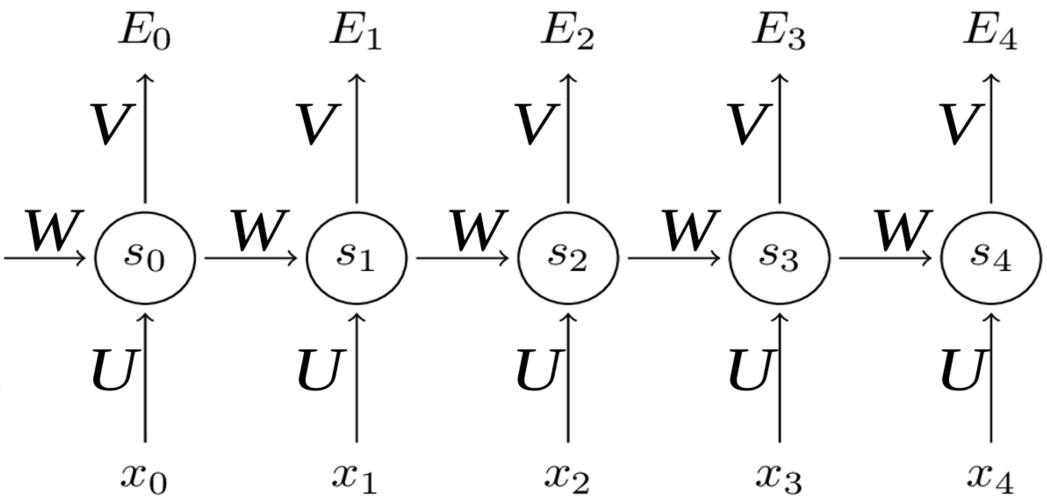
Backpropagation Through Time (BPTT)

RNN forward pass

$$s_t = \tanh(Ux_t + Ws_{t-1})$$

$$\hat{y}_t = softmax(Vs_t)$$

$$E(y, \hat{y}) = -\sum_t E_t(y_t, \hat{y}_t)$$



Backpropagation Through Time

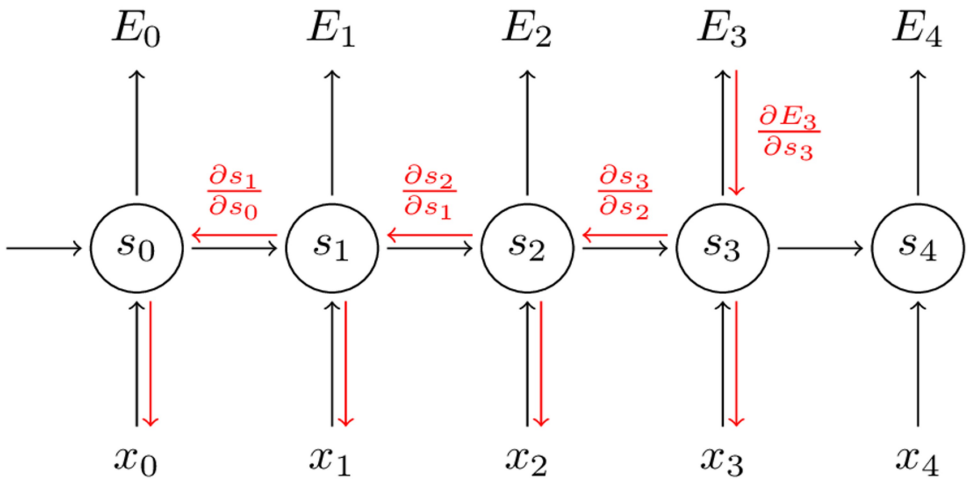
$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial W}$$

But $s_3 = \tanh(Ux_t + Ws_2)$

S_3 depends on s_2, which depends on W and s_1, and so on.

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial s_k}\frac{\partial s_k}{\partial W}$$



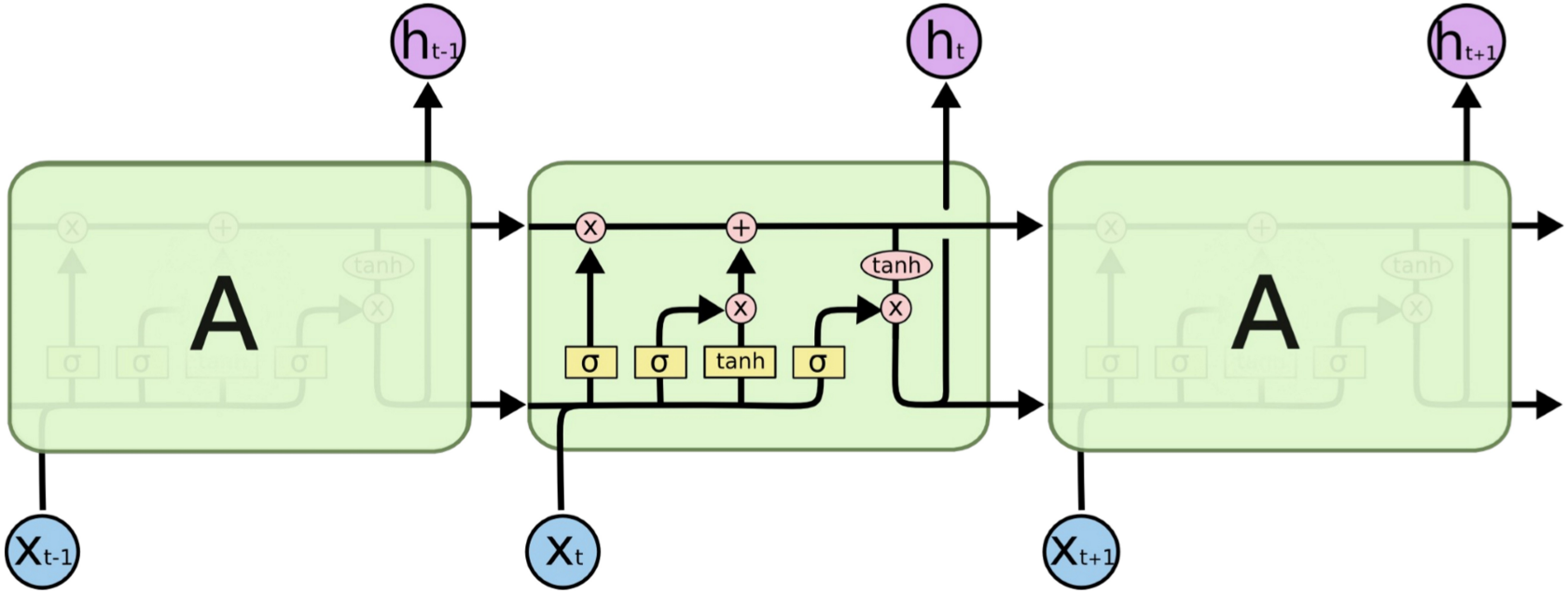
The Vanishing Gradient Problem

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W}$$

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \left(\prod_{j=k+1}^{3} \frac{\partial s_j}{\partial s_{j-1}} \right) \frac{\partial s_k}{\partial W}$$

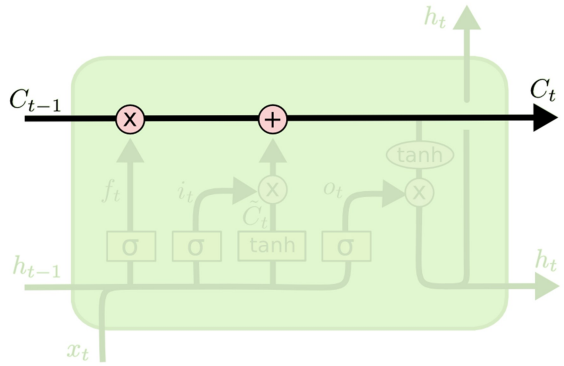
- Derivative of a vector w.r.t a vector is a matrix called jacobian
- 2-norm of the above Jacobian matrix has an upper bound of 1
- **tanh** maps all values into a range between -1 and 1, and the **derivative is bounded by 1**
- With multiple matrix multiplications, gradient values **shrink exponentially**
- Gradient contributions from “far away” steps become zero
- Depending on activation functions and network parameters, gradients could **explode** instead of vanishing

LSTM with four interacting layers



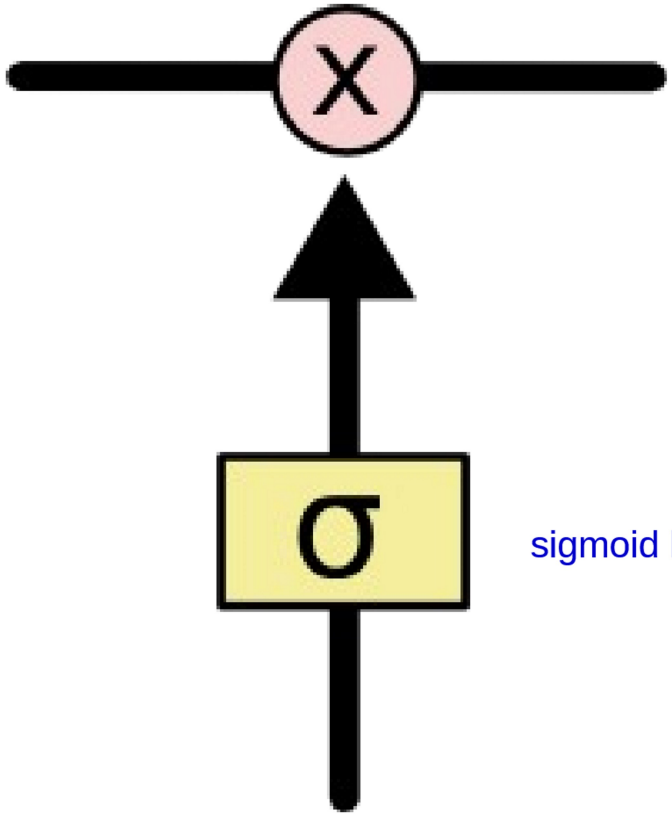
Cell States

- The **key** to LSTMs is the **cell state**, the horizontal line running through the top of the diagram.
- The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some **minor linear interactions**. It's very easy for **information to just flow along it unchanged**



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

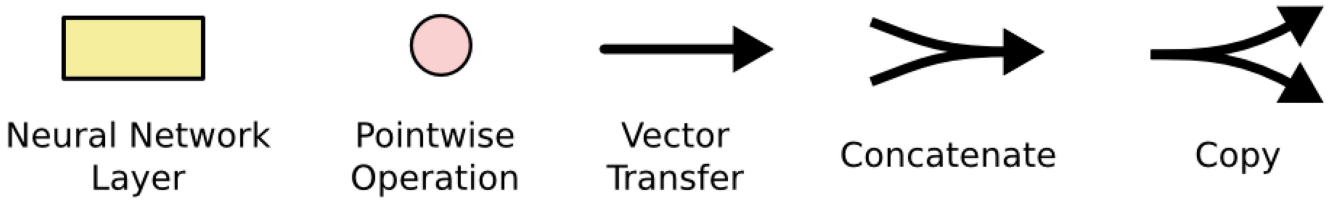
Gates



sigmoid layer

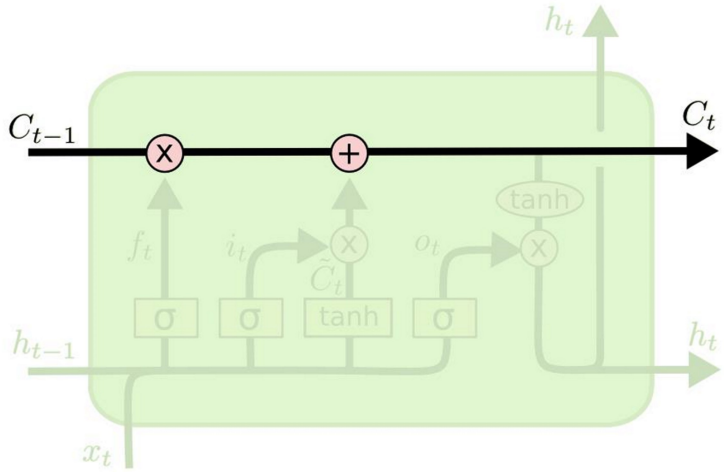
Step-by-Step LSTM Walk Through

For now, let’s just try to get comfortable with the notation we’ll be using.



Cell States

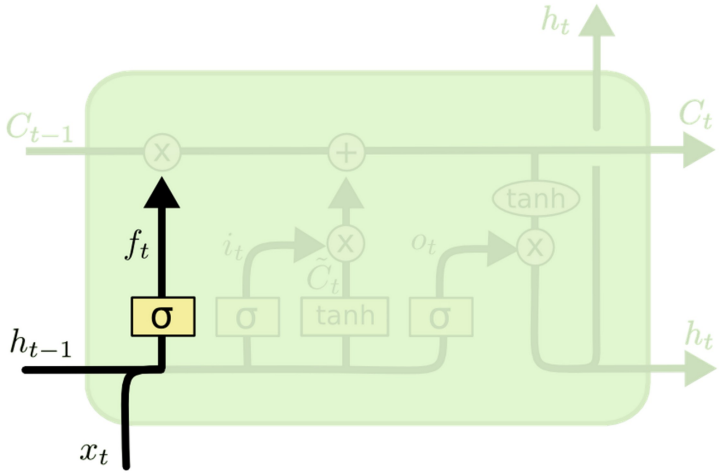
- The **key** to LSTMs is the **cell state**, the horizontal line running through the top of the diagram.
- The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some **minor linear interactions**. It's very easy for **information to just flow along it unchanged**



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

Forget Gate

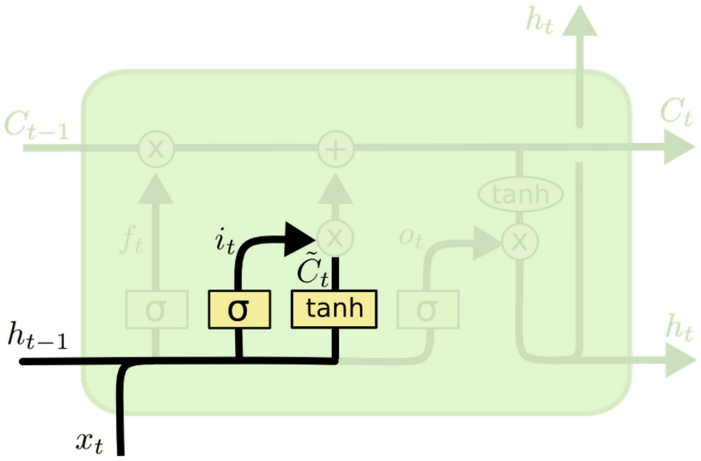
The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer." It looks at h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}. A 1 represents "completely keep this" while a 0 represents "completely get rid of this."



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

Input Gate

The next step is to decide what new information we're going to store in the cell state. This has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state. In the next step, we'll combine these two to create an update to the state.



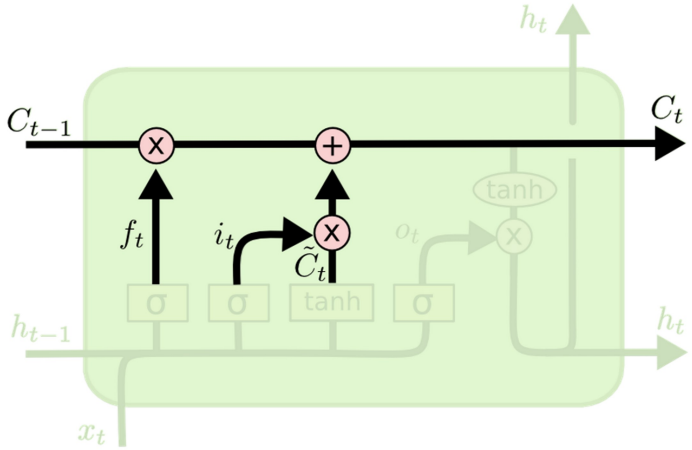
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

Update The Old Cell State

It's now time to update the old cell state, C_{t-1}, into the new cell state C_t. The previous steps already decided what to do, we just need to actually do it.

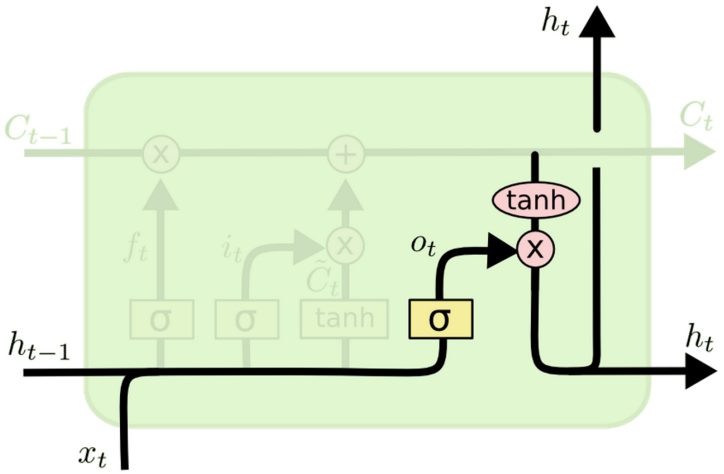
We multiply the old state by f_t, forgetting the things we decided to forget earlier. Then we add $i_t * \tilde{C}_t$. This is the new candidate values, scaled by how much we decided to update each state value.



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

Output Gate

Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through **tanh** (to push the values to be between -1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.



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

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