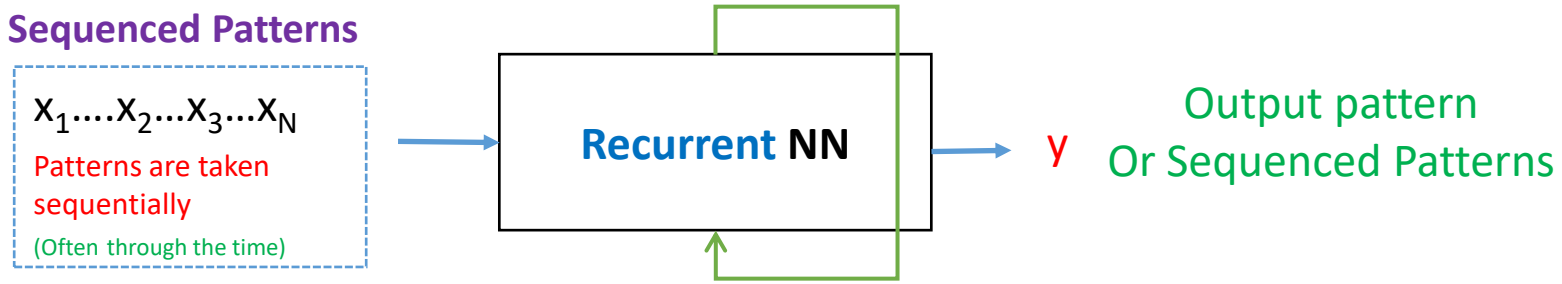
Chapter 5

Recurrent Neural Networks

Memory Neural Networks

NNs which associate an input pattern (or sequenced input patterns) to an output pattern(or sequenced output patterns) (Supervised Learning)

* such NNs make memories for pattern association.



1. RNN (Recurrent Neural Network)
2. LSTM (Long Short Term Memory)
3. GRU (Gated Recurrent Unit)

Some Applications of RNNs

1. Language Modeling and Generating Text

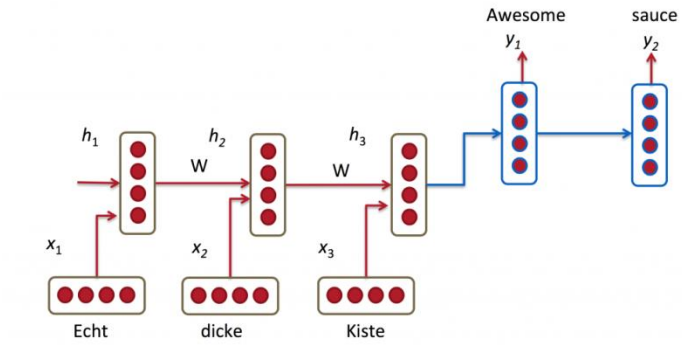
Given a sequence of words we want to predict the probability of each word given the previous words. Language Models allow us to measure how likely a sentence is, which is an important input for Machine Translation (since high-probability sentences are typically correct).

- Recurrent neural network based language model
- Extensions of Recurrent neural network based language model
- Generating Text with Recurrent Neural Networks

2. Machine Translation

- A Recursive Recurrent Neural Network for Statistical Machine Translation
- Sequence to Sequence Learning with Neural Networks
- Joint Language and Translation Modeling with Recurrent Neural Networks

Machine Translation is similar to language modeling in that our input is a sequence of words in our source language (e.g. German). We want to output a sequence of words in our target language (e.g. English).



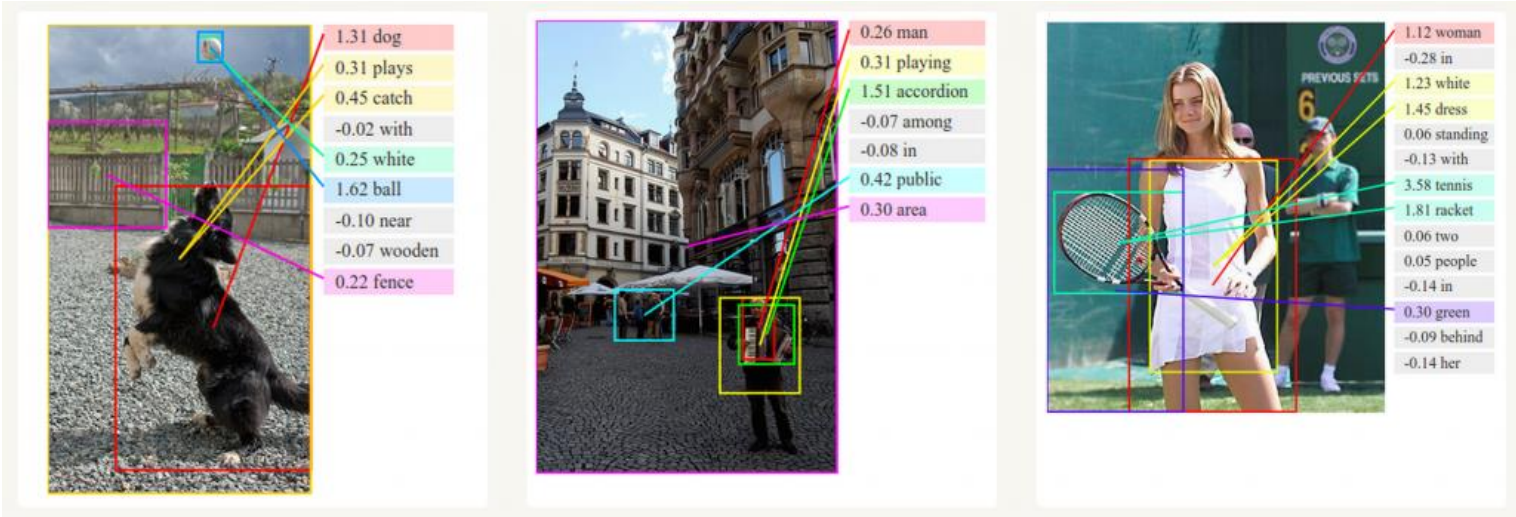
3. Speech Recognition

Given an input sequence of acoustic signals from a sound wave, we can predict a sequence of phonetic segments together with their probabilities.

1. Towards End-to-End Speech Recognition with Recurrent Neural Networks

4. Generating Image Descriptions

- Together with convolutional Neural Networks, RNNs have been used as part of a model to generate descriptions for unlabeled images



Challenges in learning of RNNs

MNNs with a sequenced input patterns

1. The sequenced input patterns may be combined with disturbance and non-related patterns.
2. The required features (to associate true output(outputs)) are hidden through sequenced inputs.
3. Among the sequenced input patterns, it may be short and long dependencies.
4. The sequenced input patterns may be presented with different modalities.

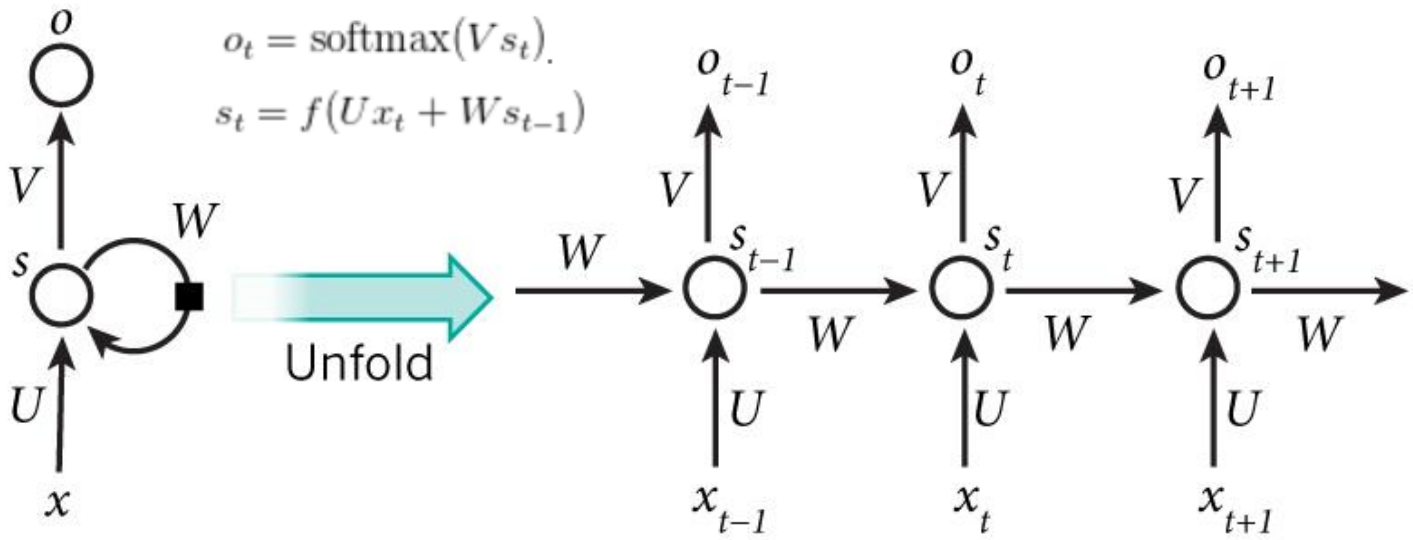
* RNNs make robust memories against distortion, disturbances and dimension variations of input patterns or different sampling rates.

- Some known Recurrent NNs
  - Recurrent Neural Networks(RNNs)
  - Long-Short Term Memory (LSTM)
  - Gated Recurrent Unit (GRU)

- Such Networks make nonlinear non-homogenous difference Equations. By solving such equations, the nonlinear functions, which generate the output patterns based on the implicit sequenced input patterns, are revealed.
- Recurrent structures will make a plenty of memories utilized in pattern associations between implicit inputs and target outputs.

- Pattern Association (Memories) types.
  - **One to one:** many classification problems: fingerprint/biometric signals/signature
  - **Many to one:** voice/handwriting/…
  - **One to Many-:** image captioning
  - **Many to Many** -- Translation machines….text to picture/Film

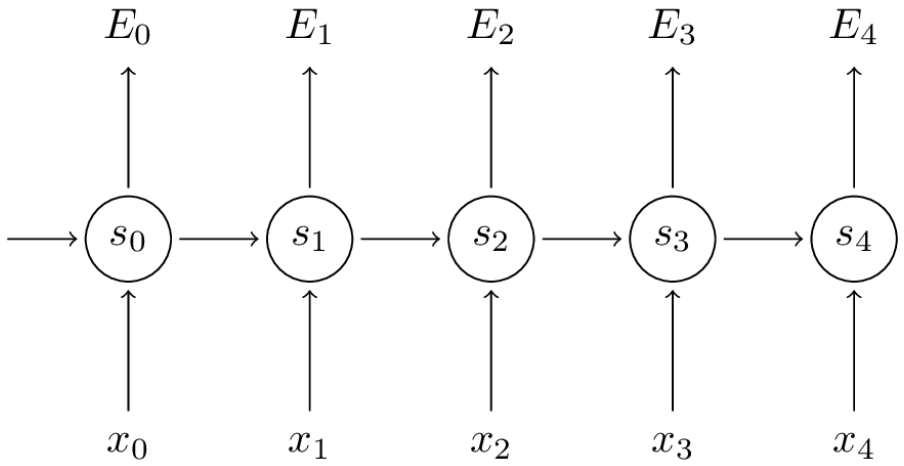
A Typical simple RNN



- x_t is the input at time step t. For example, x_1 could be a one-hot vector corresponding to the second word of a sentence.
- s_t is the hidden state at time step t. It's the "memory" of the network. s_t is calculated based on the previous hidden state and the input at the current step: $s_t = f(Ux_t + Ws_{t-1})$. The function f usually is a nonlinearity such as tanh or ReLU. s_{-1}, which is required to calculate the first hidden state, is typically initialized to all zeroes.
- o_t is the output at step t. For example, if we wanted to predict the next word in a sentence it would be a vector of probabilities across our vocabulary. $o_t = \text{softmax}(Vs_t)$.

Training RNNs

Back Propagation Through Time (BPTT)



$$s_t = \tanh(Ux_t + Ws_{t-1})$$
$$\hat{y}_t = \mathrm{softmax}(Vs_t)$$

Loss function

$$E_t(y_t, \hat{y}_t) = \underbrace{0.5(y_t - \hat{y}_t)^2}_{Squared\ Error}$$

or

$$E_t(y_t, \hat{y}_t) = \underbrace{-y_t \log(\hat{y}_t)}_{Cross\ entropy}$$

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

Updating Rules

$$W^+ = W^- - \eta \frac{\partial E_3}{\partial W}\Big|_{U^-, V^-, W^-}$$

$$U^+ = U^- - \eta \frac{\partial E_3}{\partial U}\Big|_{U^-, V^-, W^-}$$

$$V^+ = V^- - \eta \frac{\partial E_3}{\partial V}\Big|_{U^-, V^-, W^-}$$

Back Propagation Through Time (BPTT)

$$s_t = \tanh(Ux_t + Ws_{t-1})$$
$$\hat{y}_t = \text{softmax}(Vs_t)$$

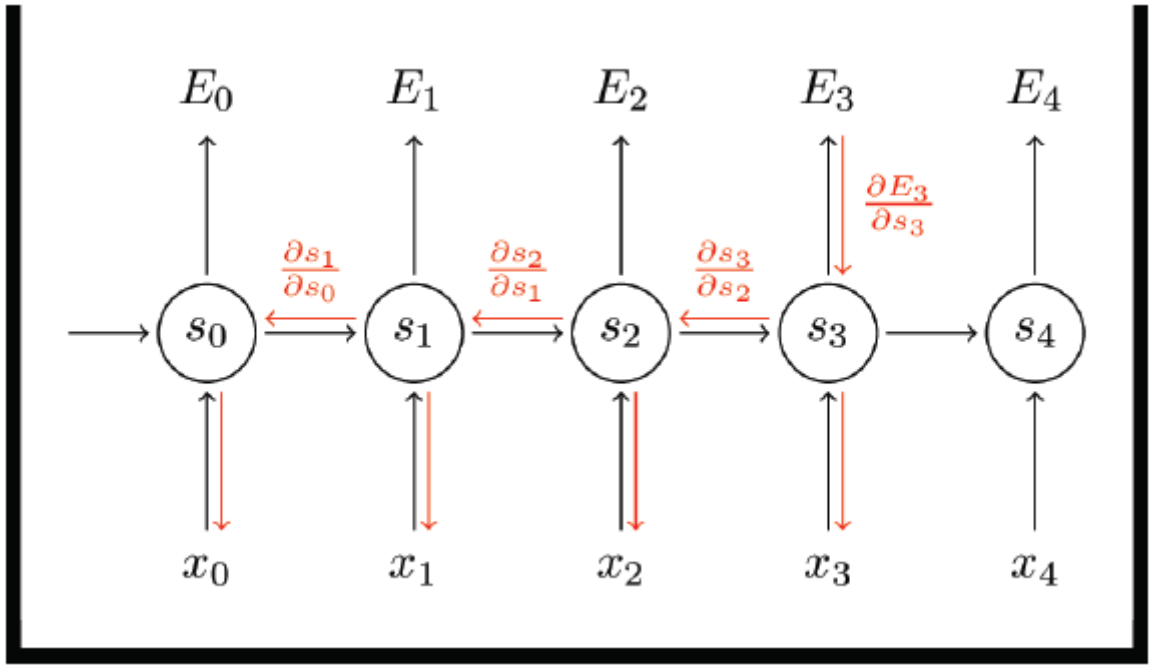
$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial W}$$

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

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

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

$$\frac{\partial E_3}{\partial V} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial V} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial z_3}\frac{\partial z_3}{\partial V} = (\hat{y}_3 - y_3) \otimes s_3$$



Main limitation in RNN

derivative of sigmoid functions is less than one.

$$\frac{\partial \tanh(x)}{\partial x} < 1 \ , \frac{\partial \text{sigm}(x)}{\partial x} < 1$$

In "BPTT" when the RNN is unfolded for many times the back-propagated gradient coefficient is vanished for the inputs taken at older times.

$$\text{In computing } \left(\frac{\partial E_t}{\partial W}\right) \text{ or } \left(\frac{\partial E_t}{\partial U}\right) \text{ for } d \gg 1:$$

$$\left\| \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial s_t} \frac{\partial s_t}{\partial s_{t-1}} \dots \frac{\partial s_{t-d+1}}{\partial s_{t-d}} \right\| \to 0$$

The learning for long dependencies is not effective any more.

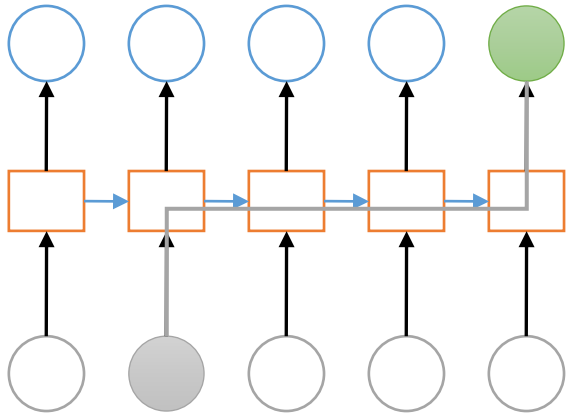
The Vanishing Gradient Problem

Error gradients vanish exponentially quickly with the size of the time lag between important events

RNNs are good to make Short Term Memory

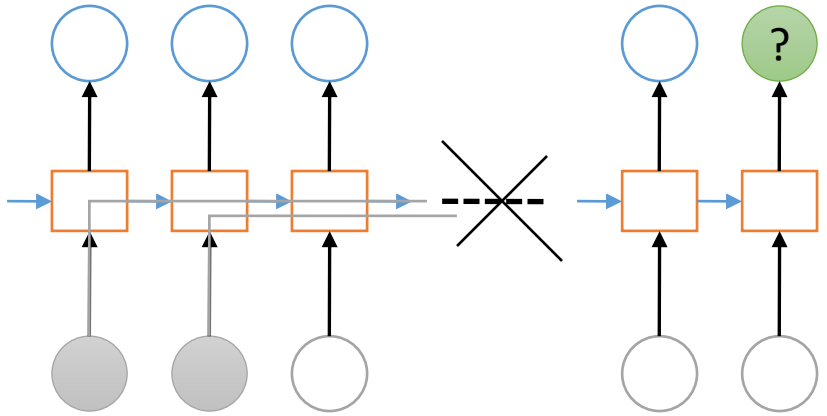
The clouds are in the SKY

implicit input → target



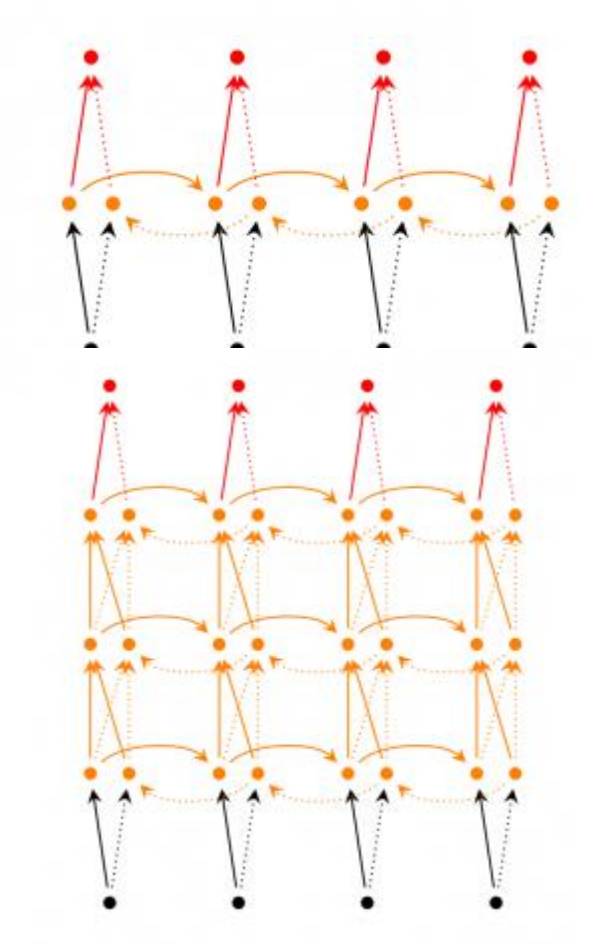
RNNs are not good to make Long Term Memory

I grew up in France……..I speak fluent French



RNN Extensions

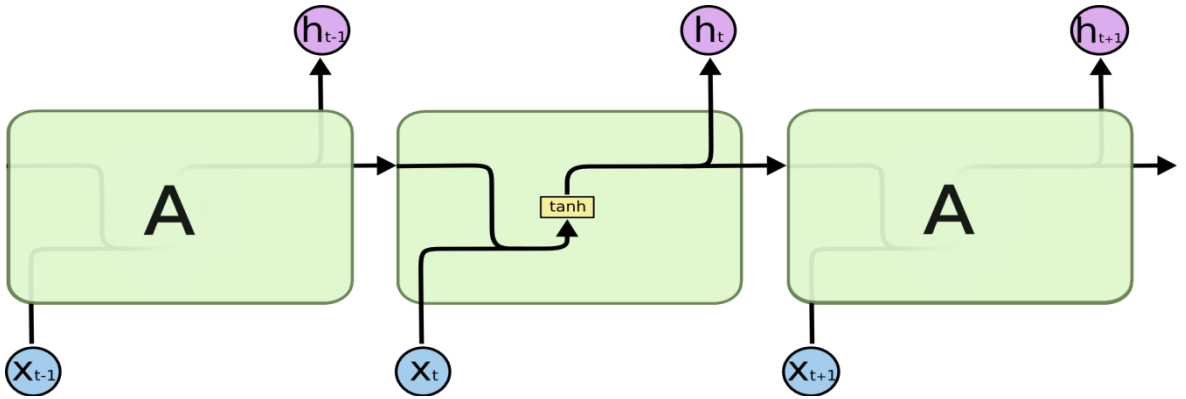
- **Bidirectional RNNs** are based on the idea that the output at time may not only depend on the previous elements in the sequence, but also future elements. For example, to predict a missing word in a sequence you want to look at both the left and the right context. Bidirectional RNNs are quite simple. They are just two RNNs stacked on top of each other. The output is then computed based on the hidden state of both RNNs.
- **Deep (Bidirectional) RNNs** are similar to Bidirectional RNNs, only that we now have multiple layers per time step. In practice this gives us a higher learning capacity (but we also need a lot of training data)



LSTM (long short term memory)

Hochreitor & Shmidhuber 1997

Module in RNN

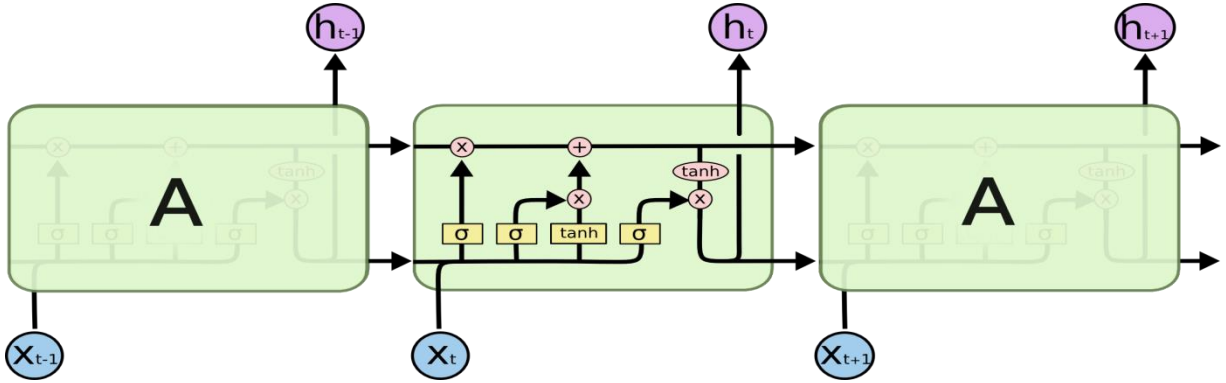


($\mathbf{h}_t$ is the same $\mathbf{s}_t$)

$\mathbf{h}_t = \tanh(Ux_t + W\,\mathbf{h}_{t-1})$

The repeating module in a standard RNN contains a single layer.

Module in LSTM

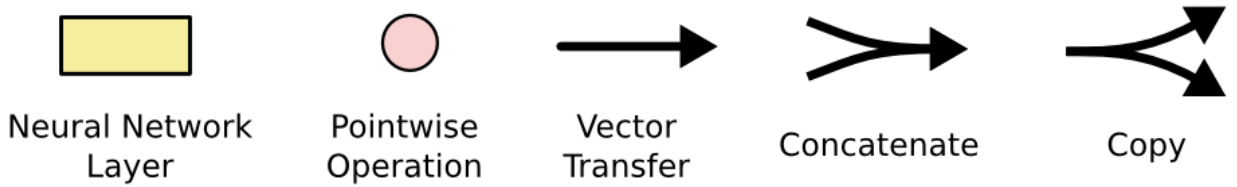


The repeating module in an LSTM contains four interacting layers.

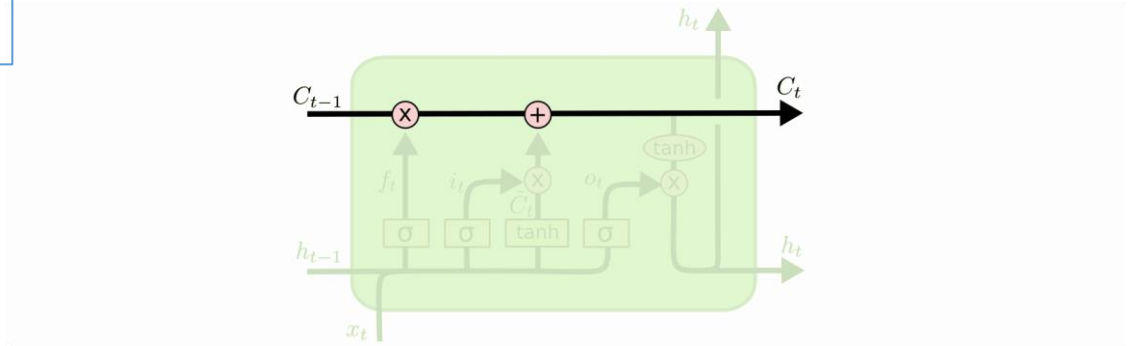
the LSTM can read, write and delete information from its memory.

Some Concepts

The notation

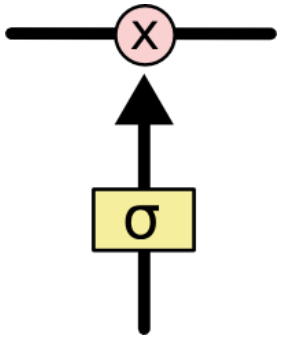


Two key concepts in LSTMs



1. Cell state

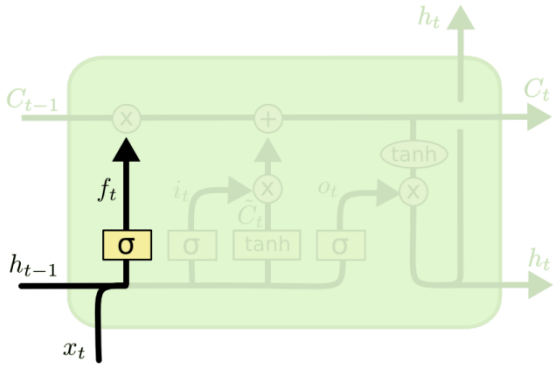
2. Gate



Gates are a way to optionally let information through.
They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.

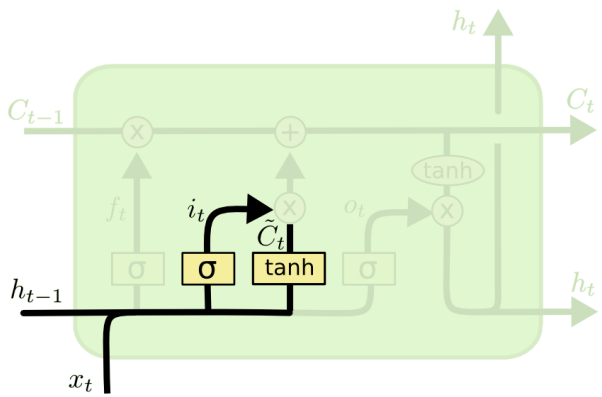
Step-by-Step LSTM Walk Through

Step1: to decide what information we're going to throw away from the cell state.



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

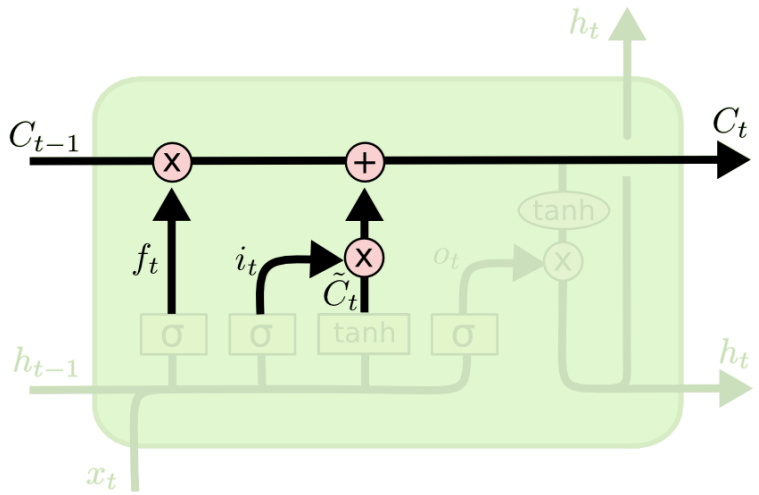
Step2: to decide what new information we're going to store in the cell state.



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

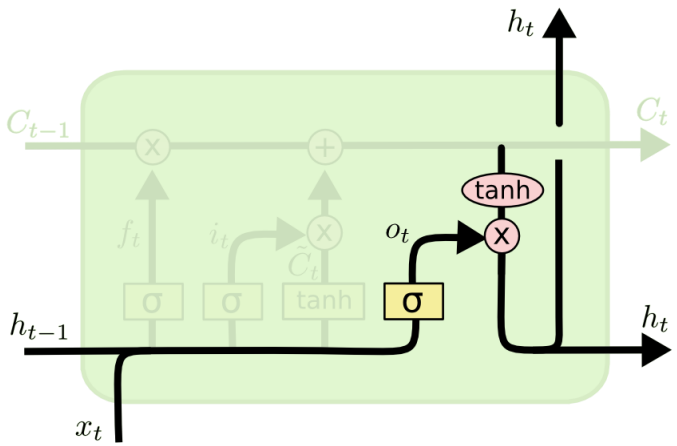
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Step3: drop the information about the old subject's gender and add the new information, as we decided in the previous steps.



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

Step4: to decide what we're going to output



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

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

Compact Form Equations

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

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

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

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

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

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

$$\hat{y}_t = f(V.h_t + b_v)$$

$\boldsymbol{x_t \in \mathbf{R}^n}$

$\boldsymbol{W_{f,i,c,o} \in \mathbf{R}^{h \times (h+n)}}$

$\boldsymbol{b_{f,i,c,o} \in \mathbf{R}^{h \times 1}}$

$\boldsymbol{f_t, i_t, o_t, h_t. c_t \in \mathbf{R}^h}$

$\boldsymbol{V \in \mathbf{R}^{m \times h}}$ $\boldsymbol{\hat{y}_t, y_t \in \mathbf{R}^m}$

Parameters: $\{(W_i, b_i), (W_f, b_f), (W_C, b_c), (W_o, b_o), (V, b_V)\}$

$$\text{Loss Function: } E_t = \sum_t E(y_t - \hat{y}_t)$$

$$\text{Parameters}^{+} = \text{Parameters}^{-} - \gamma \frac{\partial \text{E}_\text{t}}{\partial \text{Parameters}}$$

- Parameters are updated by “BPTT”
  All computed gradients of parameters are added together through the time

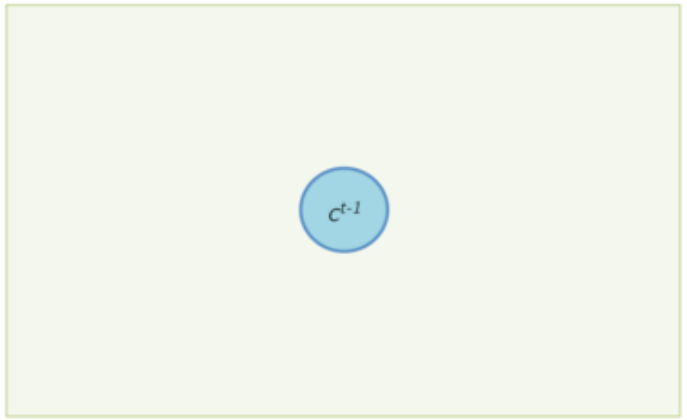
LSTM Learning methods

1. **Like RNN, "BPTT" can be performed to learn the weights and biases of a LSTM module. Unlike standard RNNs, the error remains in the unit's memory.**
2. LSTM can also be trained by a combination of artificial evolution for weights to the hidden units, and pseudo-inverse or support vector machines for weights to the output units.
3. In reinforcement learning applications LSTM can be trained by policy gradient methods,evolution strategies or genetic algorithms.

LSTM Forward and Backward Pass, Arun Mallya

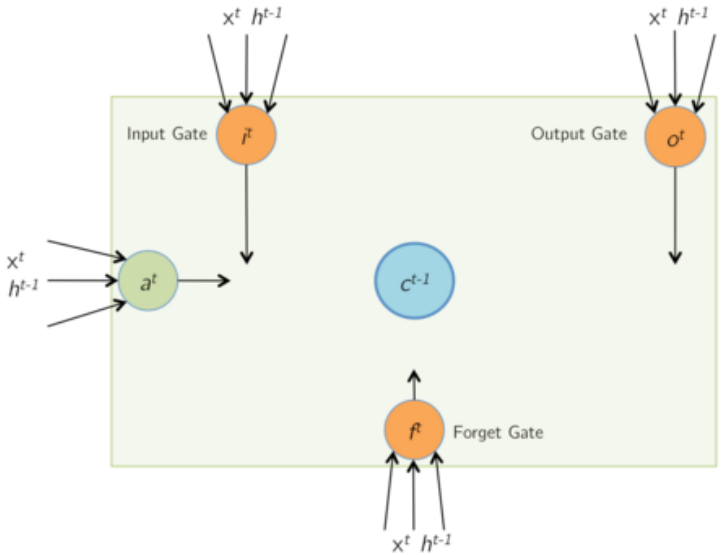
Forward Pass: Initial State

Initially, at time t, the memory cells of the LSTM contain values from the previous iteration at time (t − 1).



Forward Pass: Input and Gate Computation

At time t, The LSTM receives a new input vector x^t (including the bias term), as well as a vector of its output at the previous timestep, h^{t-1}.



$$
\begin{aligned}
a^t &= \tanh(W_c x^t + U_c h^{t-1}) = \tanh(\hat{a}^t) \\
i^t &= \sigma(W_i x^t + U_i h^{t-1}) = \sigma(\hat{i}^t) \\
f^t &= \sigma(W_f x^t + U_f h^{t-1}) = \sigma(\hat{f}^t) \\
o^t &= \sigma(W_o x^t + U_o h^{t-1}) = \sigma(\hat{o}^t)
\end{aligned}
$$

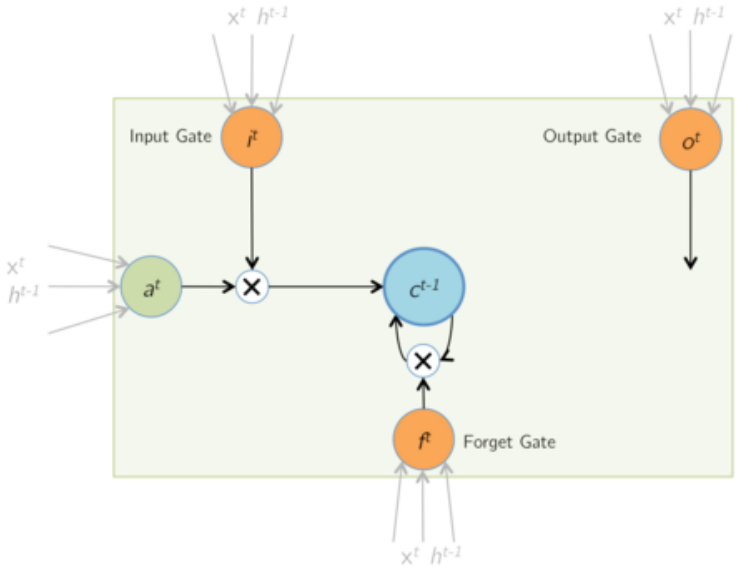
Ignoring the non-linearities,

$$
z^t = \begin{bmatrix} \hat{a}^t \\ \hat{i}^t \\ \hat{f}^t \\ \hat{o}^t \end{bmatrix} = \begin{bmatrix} W^c & U^c \\ W^i & U^i \\ W^f & U^f \\ W^o & U^o \end{bmatrix} \times \begin{bmatrix} x^t \\ h^{t-1} \end{bmatrix} = W \times I^t
$$

If the input x^t is of size $n \times 1$, and we have d memory cells, then the size of each of W_* and U_* is $d \times n$, and and $d \times d$ resp. The size of W will then be $4d \times (n + d)$. Note that each one of the d memory cells has its own weights W_* and U_*, and that the only time memory cell values are shared with other LSTM units is during the product with U_*.

Forward Pass: Memory Cell Update

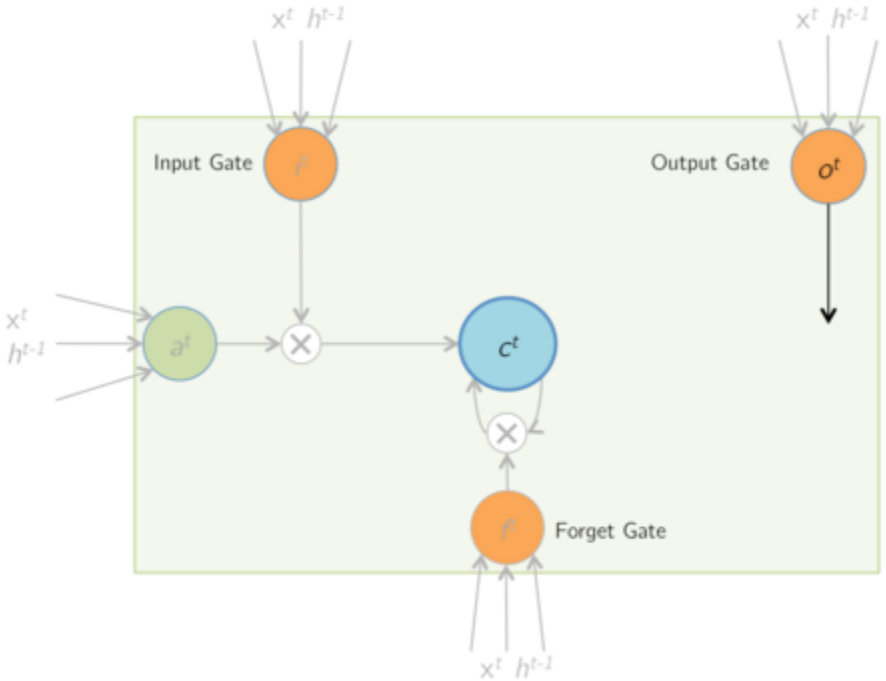
During this step, the values of the memory cells are updated with a combination of a^t, and the previous cell contents c^{t-1}. The combination is based on the magnitudes of the input gate i^t and the forget gate f^t. $\odot$ denotes elementwise product (Hadamard product).



$$c^t = i^t \odot a^t + f^t \odot c^{t-1}$$

Forward Pass: Updated Memory Cells

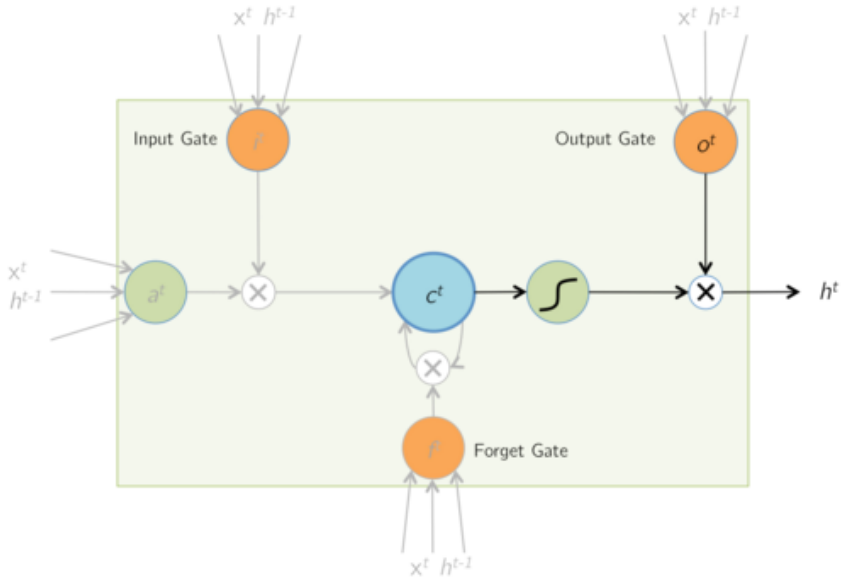
The contents of the memory cells are updated to the latest values.



$$c^{t-1} \rightarrow c^t$$

Forward Pass: Output

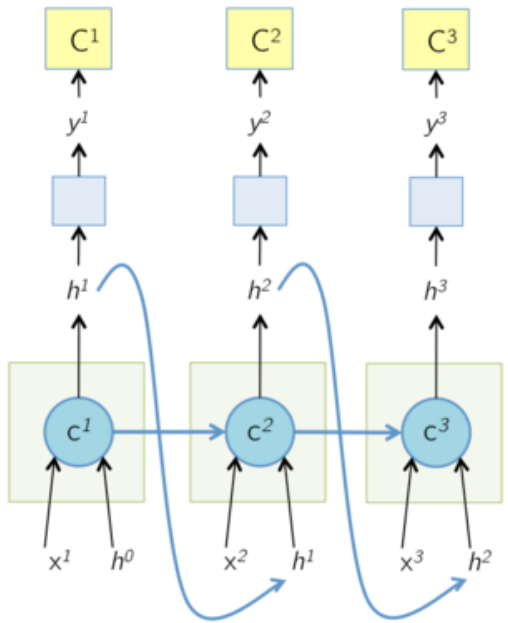
Finally, the LSTM cell computes an output value by passing the updated (and current) cell value through a non-linearity. The output gate determines how much of this computed output is actually passed out of the cell as the final output h^t.



$$h^t = o^t \odot \tanh(c^t)$$

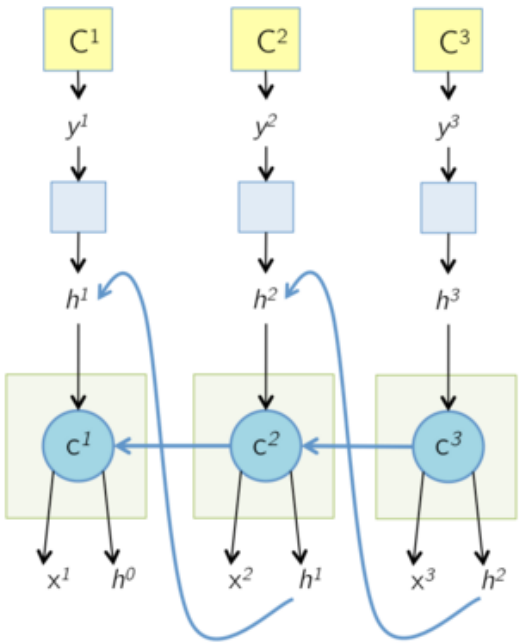
Forward Pass: Unrolled Network

The unrolled network during the forward pass is shown below. Note that the gates have not been shown for brevity. An interesting point to note here is that in the computational graph below, the cell state at time T, c^T is responsible for computing h^T as well as the next cell state c^{T+1}. At each time step, the cell output h^T is shown to be passed to some more layers on which a cost function C^T is computed, as the way an LSTM would be used in a typical application like captioning or language modeling.

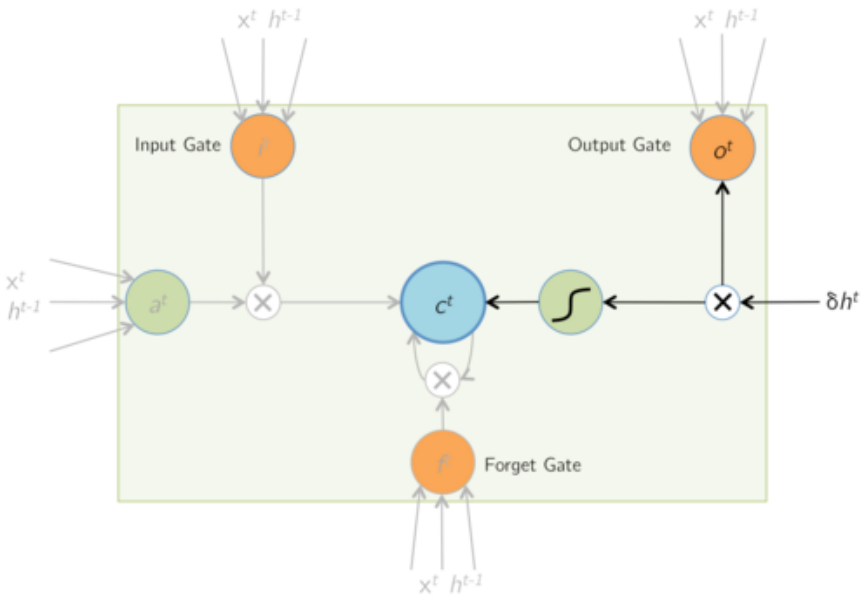


Backward Pass: Unrolled Network

The unrolled network during the backward pass is shown below. All the arrows in the previous slide have now changed their direction. The cell state at time T, c^T receives gradients from h^T as well as the next cell state c^{T+1} . The next few slides focus on computing these two gradients. At any time step T, these two gradients are accumulated before being backpropagated to the layers below the cell and the previous time steps.



Backward Pass: Output



Forward Pass: $h^t = o^t \odot \tanh(c^t)$

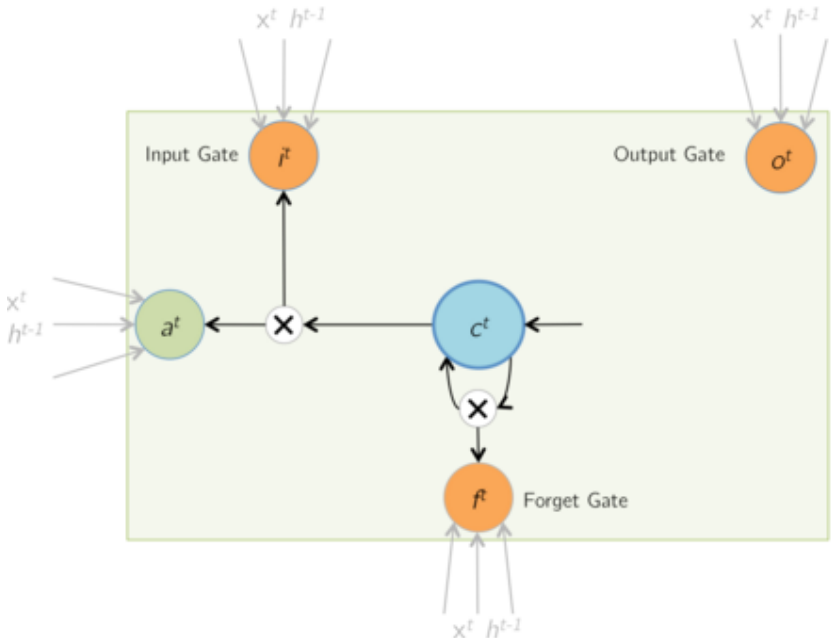
Given $\delta h^t = \dfrac{\partial E}{\partial h^t}$, find $\delta o^t, \delta c^t$

$$\begin{aligned}\frac{\partial E}{\partial o_i^t} &= \frac{\partial E}{\partial h_i^t} \cdot \frac{\partial h_i^t}{\partial o_i^t} \\ &= \delta h_i^t \cdot \tanh(c_i^t) \\ \therefore \delta o^t &= \delta h^t \odot \tanh(c^t)\end{aligned}$$

$$\begin{aligned}\frac{\partial E}{\partial c_i^t} &= \frac{\partial E}{\partial h_i^t} \cdot \frac{\partial h_i^t}{\partial c_i^t} \\ &= \delta h_i^t \cdot o_i^t \cdot (1 - \tanh^2(c_i^t)) \\ \therefore \delta c^t &+= \delta h^t \odot o^t \odot (1 - \tanh^2(c^t))\end{aligned}$$

Note that the $+=$ above is so that this gradient is added to gradient from time step $(t+1)$ (calculated on next slide, refer to the gradient accumulation mentioned in the previous slide)

Backward Pass: LSTM Memory Cell Update



Forward Pass: $c^t = i^t \odot a^t + f^t \odot c^{t-1}$

Given $\delta c^t = \dfrac{\partial E}{\partial c^t}$, find $\delta i^t, \delta a^t, \delta f^t, \delta c^{t-1}$

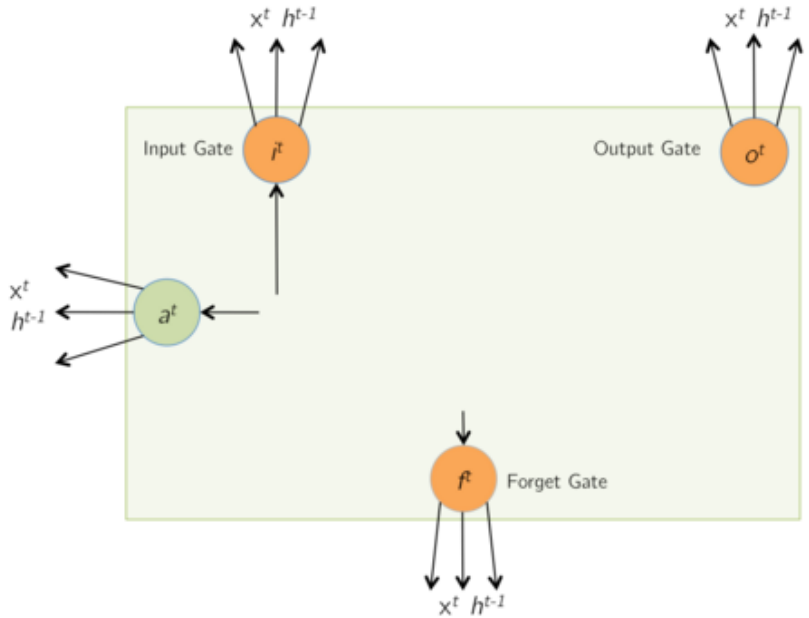
$$\frac{\partial E}{\partial i_i^t} = \frac{\partial E}{\partial c_i^t} \cdot \frac{\partial c_i^t}{\partial i_i^t} = \delta c_i^t \cdot a_i^t \quad \therefore \delta i^t = \delta c^t \odot a^t$$

$$\frac{\partial E}{\partial f_i^t} = \frac{\partial E}{\partial c_i^t} \cdot \frac{\partial c_i^t}{\partial f_i^t} = \delta c_i^t \cdot c_i^{t-1} \quad \therefore \delta f^t = \delta c^t \odot c^{t-1}$$

$$\frac{\partial E}{\partial a_i^t} = \frac{\partial E}{\partial c_i^t} \cdot \frac{\partial c_i^t}{\partial a_i^t} = \delta c_i^t \cdot i_i^t \quad \therefore \delta a^t = \delta c^t \odot i^t$$

$$\frac{\partial E}{\partial c_i^{t-1}} = \frac{\partial E}{\partial c_i^t} \cdot \frac{\partial c_i^t}{\partial c_i^{t-1}} = \delta c_i^t \cdot f_i^t \quad \therefore \delta c^{t-1} = \delta c^t \odot f^t$$

Backward Pass: Input and Gate Computation - I

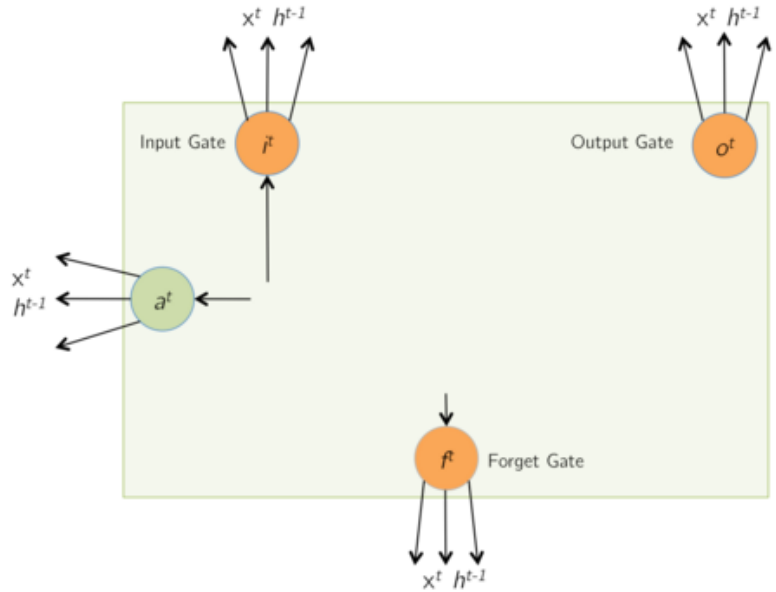


$$\text{Forward Pass: } z^t = \begin{bmatrix} \hat{a}^t \\ \hat{i}^t \\ \hat{f}^t \\ \hat{o}^t \end{bmatrix} = W \times I^t$$

$$\text{Given } \delta a^t, \delta i^t, \delta f^t, \delta o^t, \text{ find } \delta z^t$$

$$\delta \hat{a}^t = \delta a^t \odot (1 - \tanh^2(\hat{a}^t))$$
$$\delta \hat{i}^t = \delta i^t \odot i^t \odot (1 - i^t)$$
$$\delta \hat{f}^t = \delta f^t \odot f^t \odot (1 - f^t)$$
$$\delta \hat{o}^t = \delta o^t \odot o^t \odot (1 - o^t)$$
$$\delta z^t = \left[\delta \hat{a}^t, \delta \hat{i}^t, \delta \hat{f}^t, \delta \hat{o}^t\right]^T$$

Backward Pass: Input and Gate Computation - II



Forward Pass: $z^t = W \times I^t$

Given δz^t, find $\delta W^t, \delta h^{t-1}$

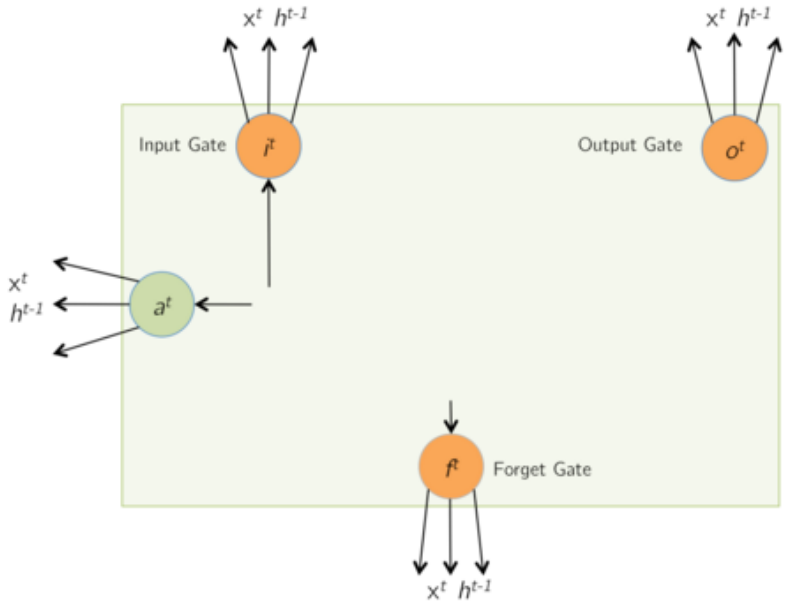
$$\delta I^t = W^T \times \delta z^t$$

$$\text{As } I^t = \begin{bmatrix} x^t \\ h^{t-1} \end{bmatrix},$$

δh^{t-1} can be retrieved from δI^t

$$\delta W^t = \delta z^t \times (I^t)^T$$

Backward Pass: Input and Gate Computation - II



Forward Pass: $z^t = W \times I^t$

Given δz^t, find $\delta W^t, \delta h^{t-1}$

$$\delta I^t = W^T \times \delta z^t$$

$$\text{As } I^t = \begin{bmatrix} x^t \\ h^{t-1} \end{bmatrix},$$

δh^{t-1} can be retrieved from δI^t

$$\delta W^t = \delta z^t \times (I^t)^T$$

Parameter Update

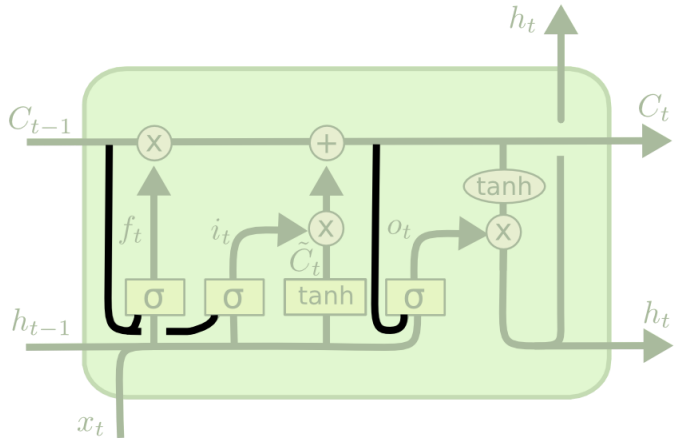
If input x has T time-steps, i.e. $x = [x^1, x^2, \cdots, x^T]$, then

$$\delta W = \sum_{t=1}^{T} \delta W^t$$

W is then updated using an appropriate Stochastic Gradient Descent solver.

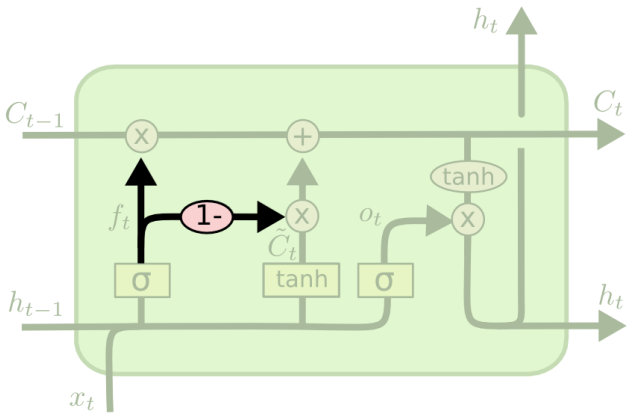
Extended Versions

1. One popular LSTM variant, introduced by Gers & Schmidhuber (2000), is adding “peephole connections.” This means that we let the gate layers look at the cell state.



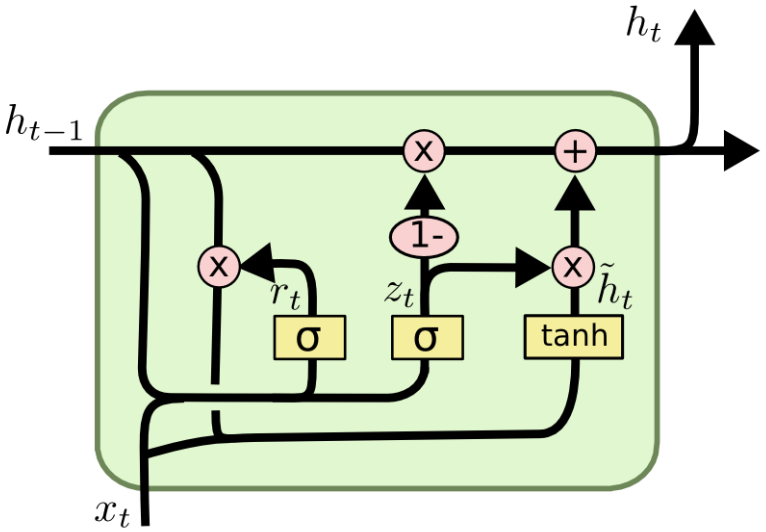
$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

2. Another variation is to use coupled forget and input gates. Instead of separately deciding what to forget and what we should add new information to, we make those decisions together. We only forget when we're going to input something in its place. We only input new values to the state when we forget something older.



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

2. A slightly more dramatic variation on the LSTM is the Gated Recurrent Unit, or GRU, introduced by Cho, et al. (2014). It combines the forget and input gates into a single "update gate." It also merges the cell state and hidden state, and makes some other changes. The resulting model is simpler than standard LSTM models, and has been growing increasingly popular.



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

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

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

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

End of Chapter 5

Thank you