# CSE440: Natural Language Processing II

Dr. Farig Sadeque
Associate Professor
Department of Computer Science and Engineering
BRAC University

# Lecture 6: Neural Nets and RNN

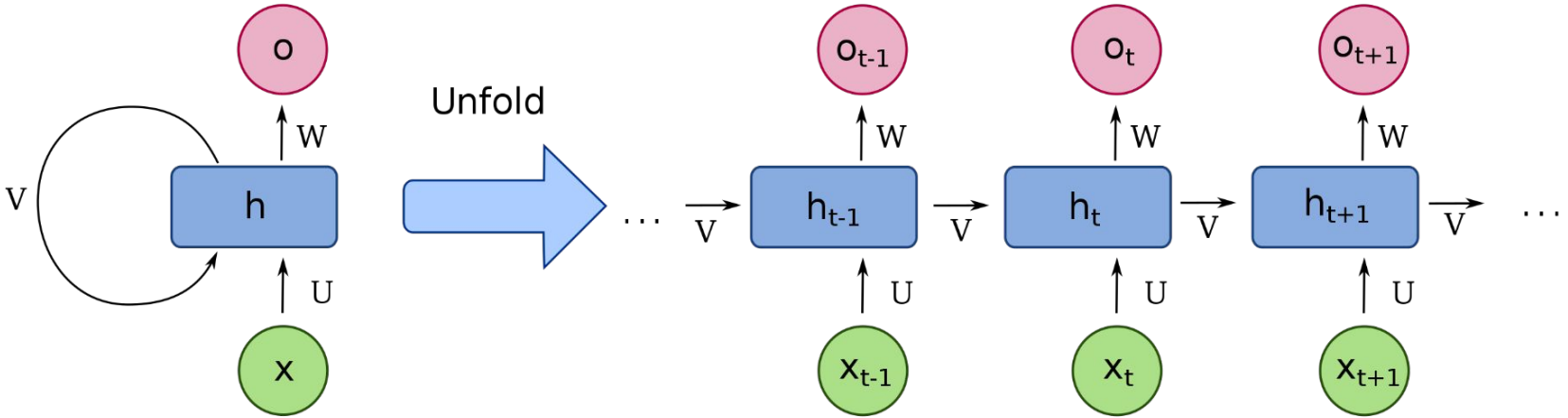
# Recurrent Neural Networks (RNN)

- Simple recurrent networks
- Bidirectional and gated recurrent networks
- Recurrent architectures
- Seq2Seq models (next session)
- Attention (next session)

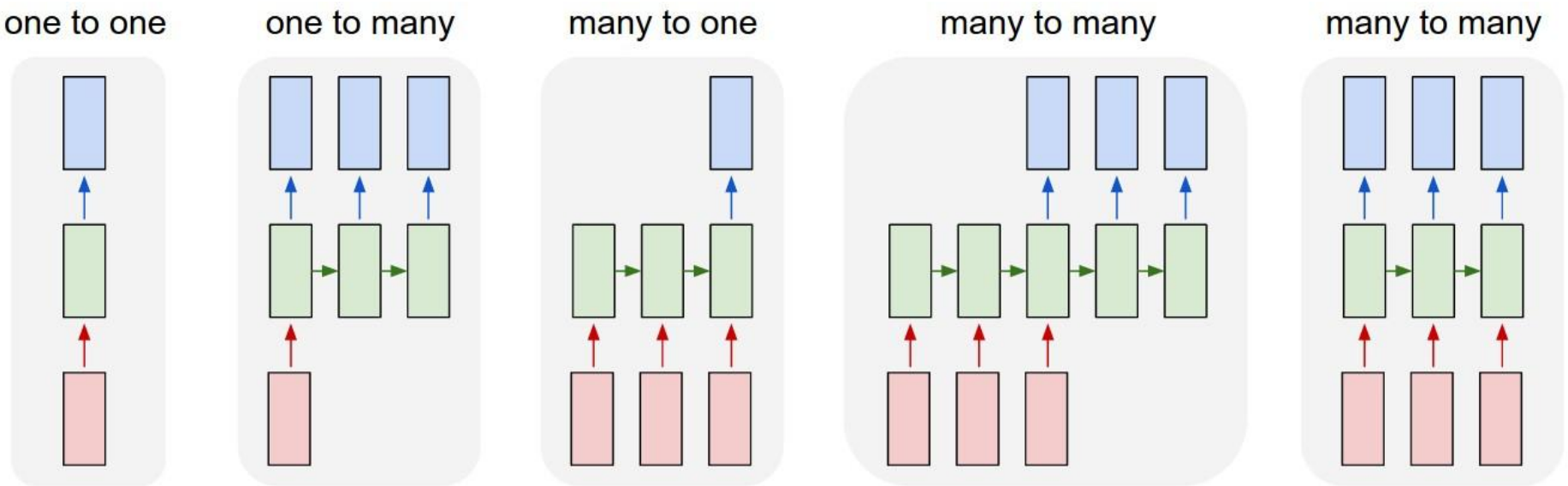
# Short history of RNN

- 1986: RNNs are Introduced by David Rumelhart
- 1995: LSTMs are introduced by Sepp Hochreiter and Jürgen Schmidhuber based on Hochreiter's 1991 research on vanishing gradient problem
- 2001: Gers and Schmidhuber trained LSTMs to learn language models (unlearnable by HMMs)
- 2009: Graves et al. won ICDAR handwriting recognition competition using LSTMs
- 2013: Hinton and his team destroyed previous record for speech recognition using LSTM
- 2014: GRU is introduced by Cho et al.
- 2015: Widespread use in both academia and industry due to Google's adaptation of LSTM in their Google Voice speech recognition system

# Structure of an RNN

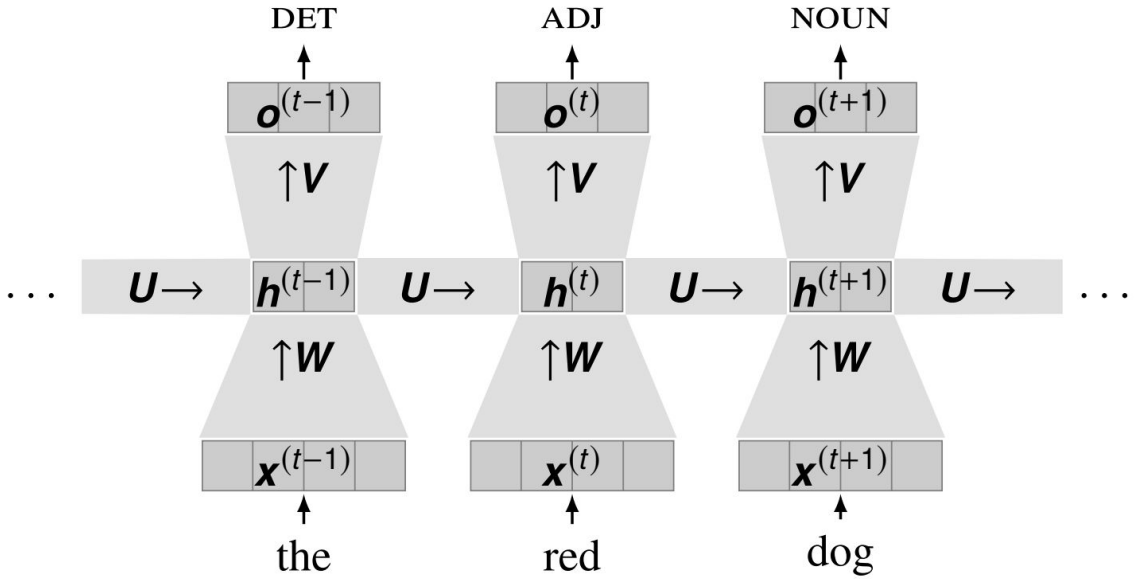


# Types of RNNs




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

# A completely unrolled Simple RNN




$$o^{(t)} = \text{softmax}(Vh^{(t)})$$

$$h^{(t)} = \tanh(Uh^{(t-1)} + Wx^{(t)})$$

# Simple RNN

Intuitions:

- Each step combines the current input with the history
- Each prediction is made based on this combination

Observations:

- The input and hidden state change at each time step
- The parameters W, U, V are the same at each step

Equations:

$$\boldsymbol{h}^{(t)} = \tanh(\boldsymbol{U}\boldsymbol{h}^{(t-1)} + \boldsymbol{W}\boldsymbol{x}^{(t)})$$

$$\boldsymbol{o}^{(t)} = \mathrm{softmax}(\boldsymbol{V}\boldsymbol{h}^{(t)})$$

# Classwork

Consider an RNN that predicts as:

$$\boldsymbol{o}^{(t)} = \text{softmax}(\boldsymbol{V}\boldsymbol{h}^{(t)})$$

$$\boldsymbol{h}^{(t)} = \boldsymbol{U}\boldsymbol{h}^{(t-1)} + \boldsymbol{W}\boldsymbol{x}^{(t)}$$

whose parameters have been set to:

$$\boldsymbol{U} = \begin{bmatrix} 2 & 0 \\ 1 & 2 \end{bmatrix} \quad \boldsymbol{W} = \begin{bmatrix} 2 & 0 & -1 \\ 1 & -1 & 1 \end{bmatrix} \quad \boldsymbol{V} = \begin{bmatrix} 1 & 0 \\ -1 & 1 \\ 0 & 3 \end{bmatrix}$$

If you are given the following input:

$$h^{(0)} = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \quad x^{(1)} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad x^{(2)} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Which label will be predicted for each word if in the final softmax, index 0=ADJ, index 1=DET, and index 2=NOUN?

# Drawbacks of simple RNN

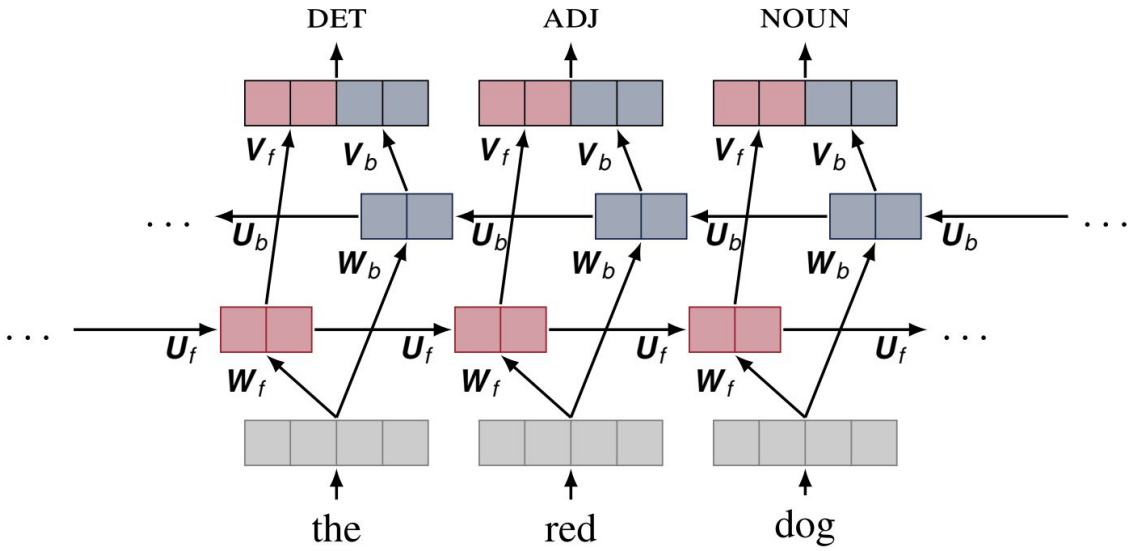
- They can only see the past, not the future
- They must forget the same amount of history at each time step
  - Theoretically, they don’t have to
  - Maintaining long term memory is difficult

# Bidirectional RNNs

Intuition:

- Run one forward RNN
- Run one backward RNN
- Combine their outputs

# Bidirectional RNNs




$$\boldsymbol{o}^{(t)} = \text{softmax}(\boldsymbol{V}_f \boldsymbol{h}_f^{(t)} + \boldsymbol{V}_b \boldsymbol{h}_b^{(t)})$$

$$\boldsymbol{h}_b^{(t)} = \tanh(\boldsymbol{U}_b \boldsymbol{h}^{(t+1)} + \boldsymbol{W}_b \boldsymbol{x}^{(t)})$$

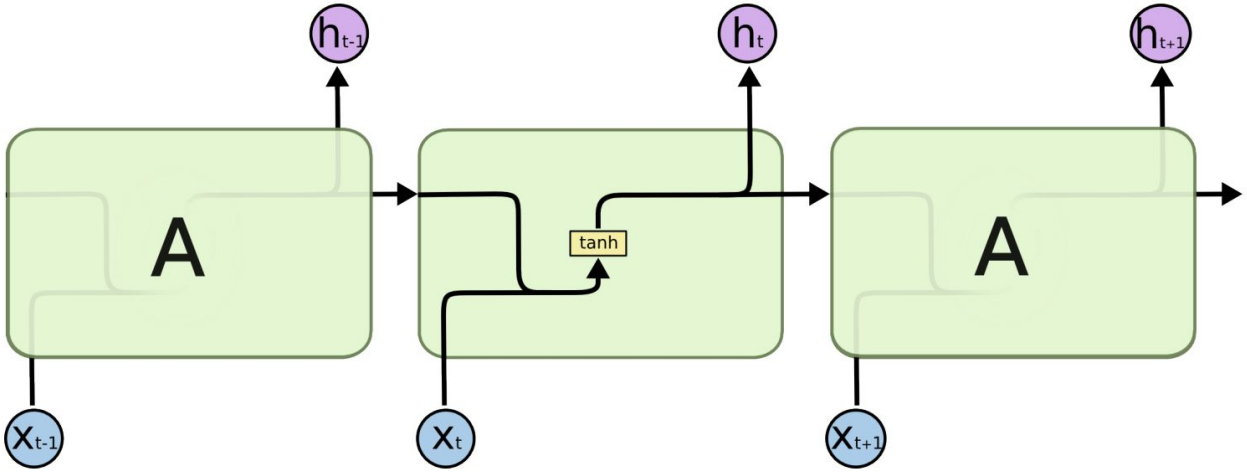
$$\boldsymbol{h}_f^{(t)} = \tanh(\boldsymbol{U}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_f \boldsymbol{x}^{(t)})$$

# Gated RNNs

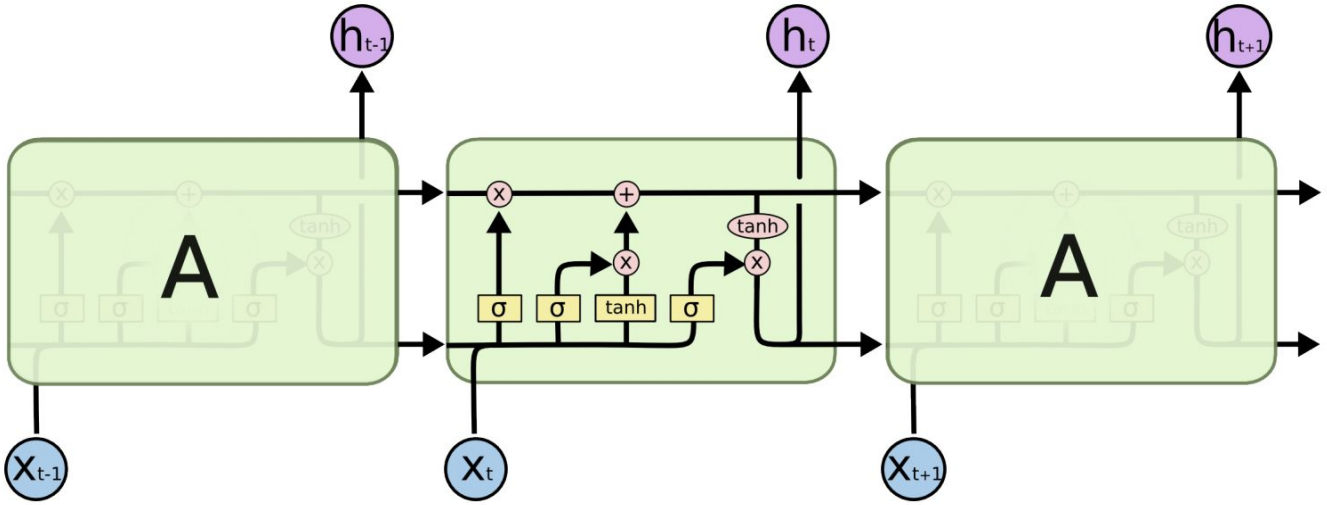
Intuition:

- Simple RNNs forget the same amount at each time step
- Look at the previous hidden state and the current input
- Decide how much to forget based on those
- Two gated RNNs
  - Long Short-Term Memory (LSTM)
  - Gated Recurrent Units (GRU)

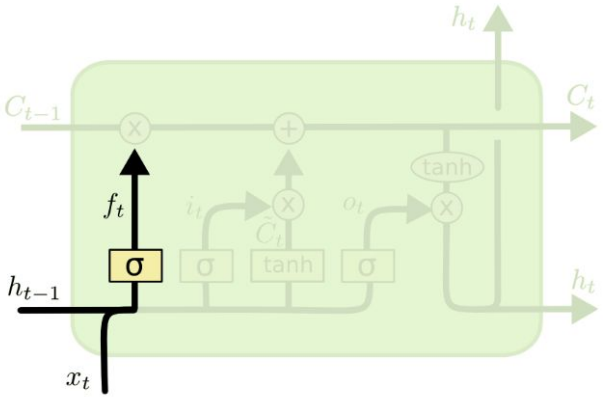
# Simple RNN



# LSTM

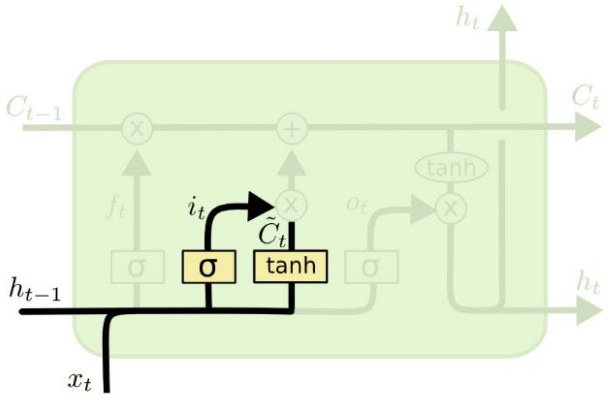


# Forget gate




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

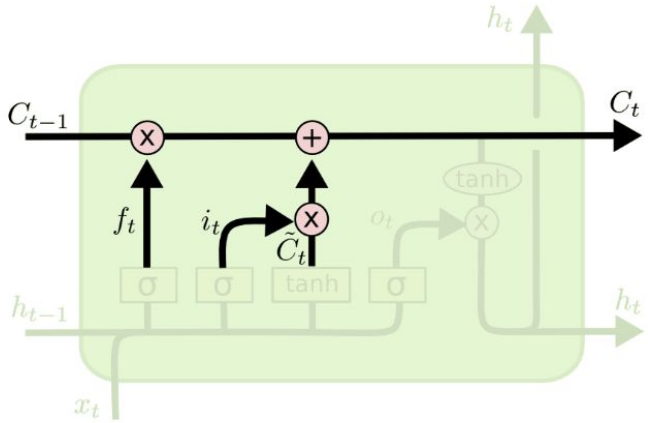
# Input gate




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

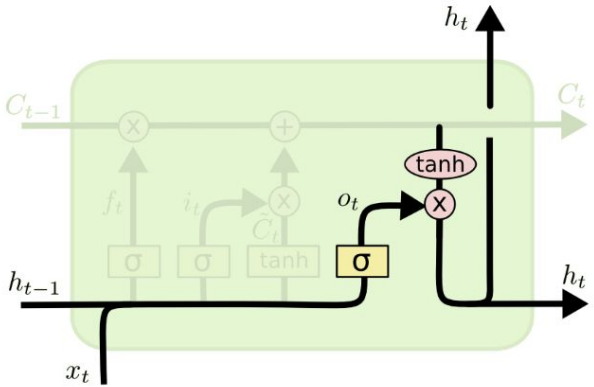
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

# Cell update




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

# Output gate



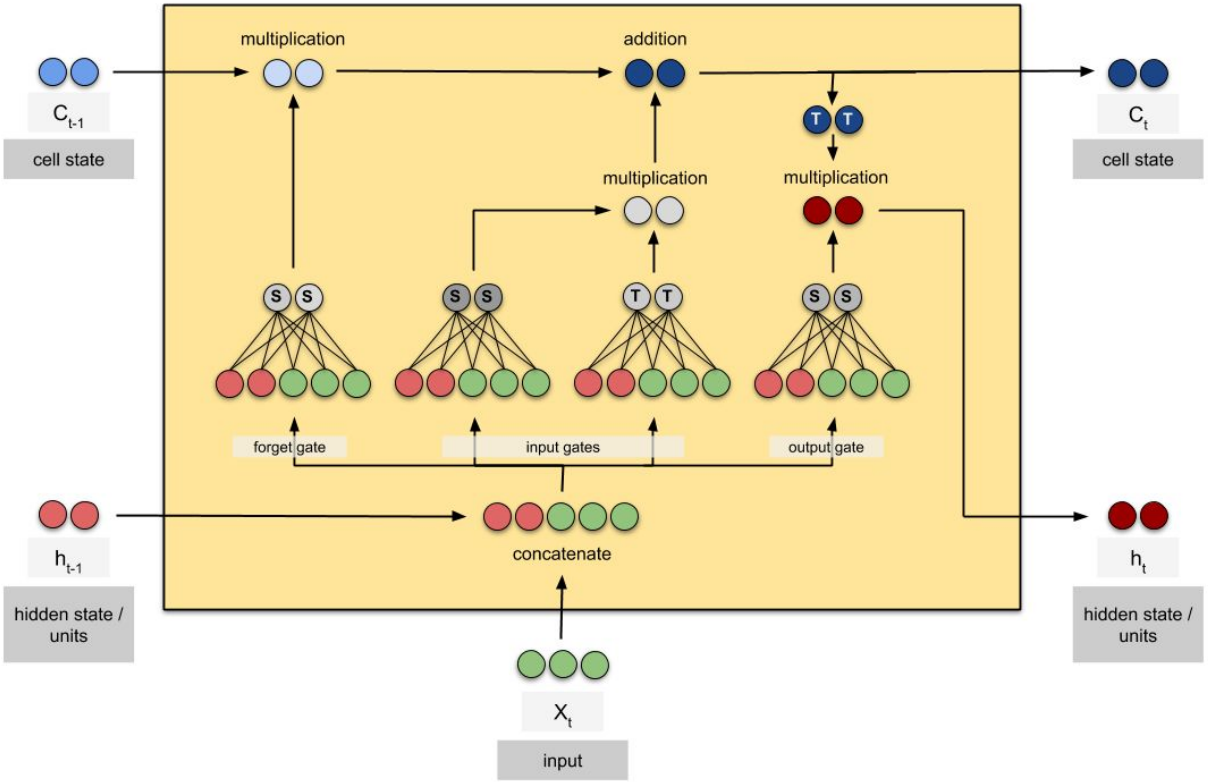

$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

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

# Useful blog from Colah

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

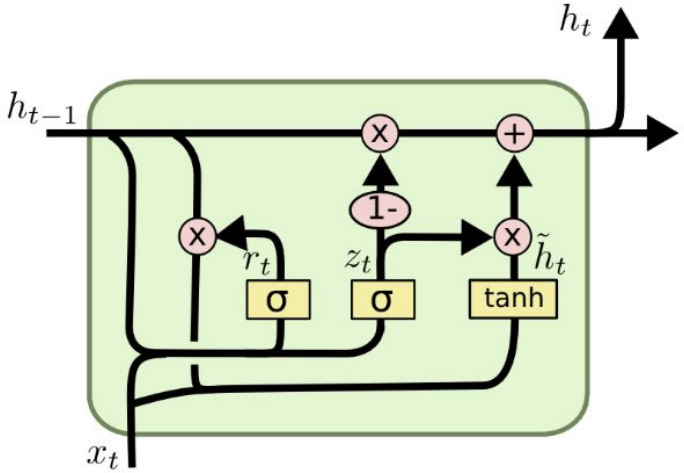
# One LSTM cell



# LSTM

- Step 1: Calculate forget ($f_t$), input ($i_t$) and output gates ($o_t$)
- Step 2: Calculate cell state update
- Step 3: Update cell state ($C_t$)
- Step 4: Calculate $h_t$

# GRU




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

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

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

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

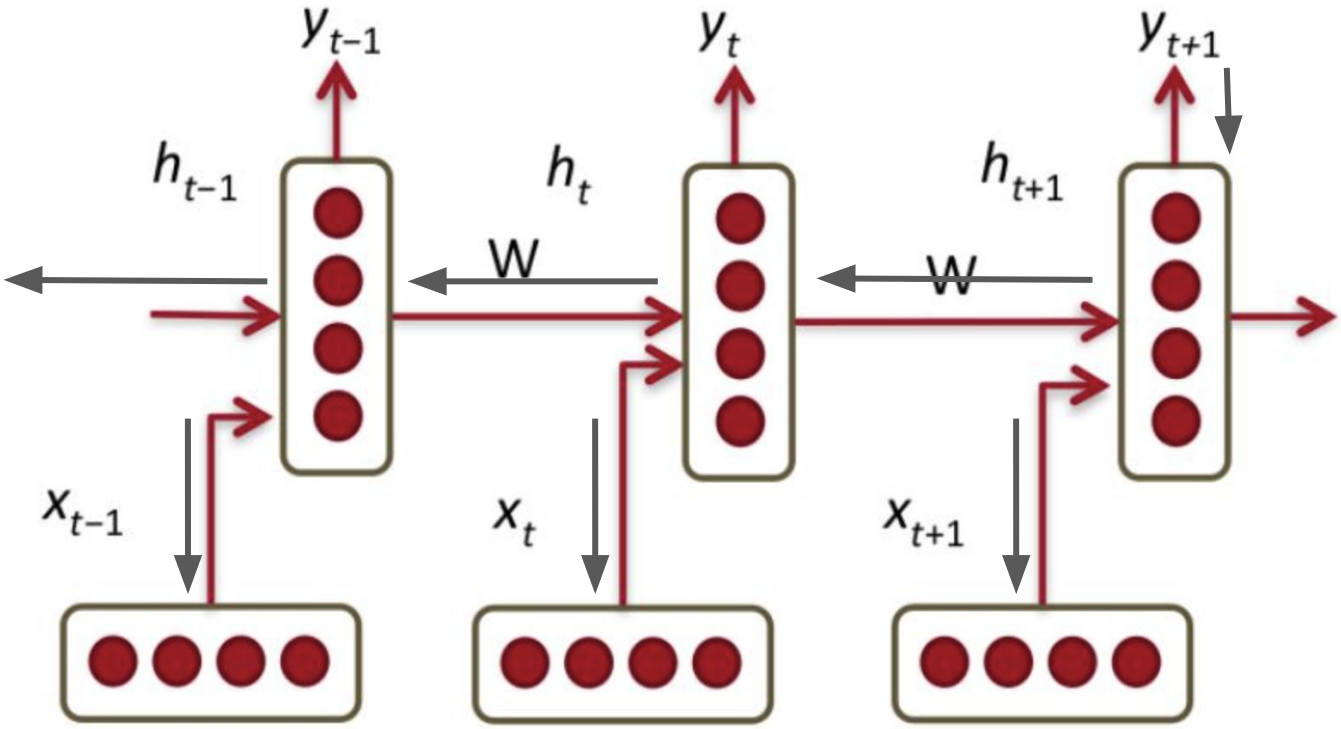
# Gated RNNs

Properties:

- Can forget different amounts at each time step
- Much better at using long distance information

A bidirectional GRU is a good starting point for many sequence tagging tasks

# Backpropagation through time



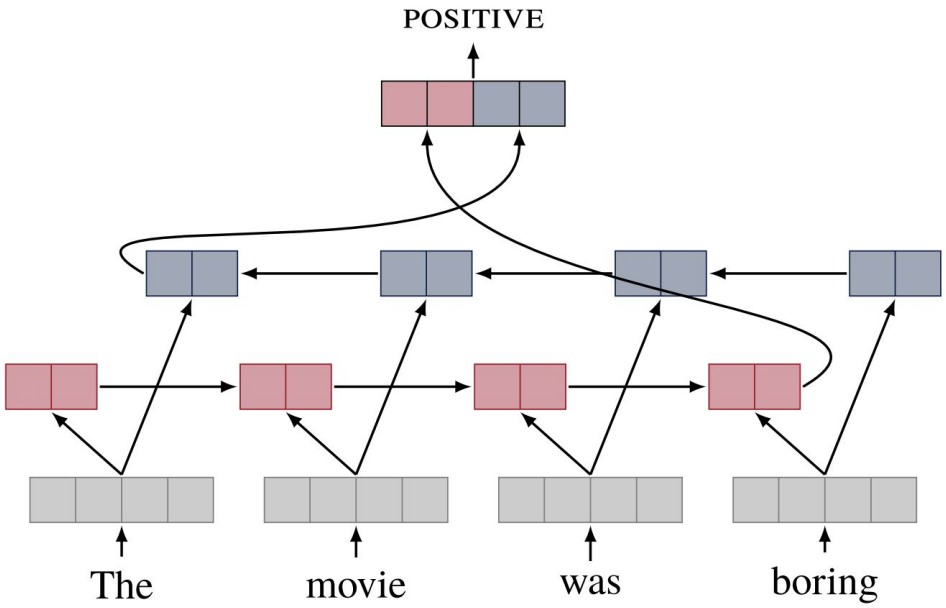
# The vanishing gradient problem

- BPTT calculates gradients backward through time, it involves taking derivatives of the loss with respect to the model's parameters at each time step
- These derivatives are multiplied together as they are propagated backward
- Since gradients are multiplied together, if the gradients at each time step are less than 1, this multiplication leads to a compounding effect
  - As you go further back in time, the gradients become increasingly smaller
- The compounding effect causes the gradients for early time steps to become vanishingly small, approaching zero
  - When the gradients are too close to zero, they don't provide meaningful information for parameter updates
  - This makes it challenging for the RNN to learn long-term dependencies in the data

# Recurrent architectures for related tasks

RNNs for text classification

- Last hidden state of the RNN represents the entire sentence.



# Recurrent architectures for related tasks

What other tasks can RNNs handle?

# Next to come

- Seq2seq models
  - Encoders and decoders
- Attention

# Practice

Equations for LSTM

$f_t = \text{sigmoid}(W_f x_t + U_f h_{t-1} + b_f)$

$i_t = \text{sigmoid}(W_i x_t + U_i h_{t-1} + b_i)$

$o_t = \text{sigmoid}(W_o x_t + U_o h_{t-1} + b_o)$

- All the weights (Ws and Us and bs) will be given. You need to calculate f, i and o.
- How to calculate sigmoid/tanh for a vector (because the argument for sigmoid in this case will be a vector)
  - sigmoid([1, 2, 0]) = [sigmoid(1), sigmoid(2), sigmoid(0)]
  - tanh([1, 2, 0]) = [tanh(1), tanh(2), tanh(0)]

# Practice

How to calculate $f_t$ for input $x_t = [1\ 1]^T$

- Given: $W_f = [1\ 1, 0\ 1]$, $U_f = [0\ 0\ , 2\ 3]$, $h_{t-1} = [4\ 5]^T$, $b_f = [0\ 0]^T$
- Using this equation: $f_t = \text{sigmoid}(W_f x_t + U_f h_{t-1} + b_f)$
  - Multiply $W_f$ with $x_t$ ($W_f$ has a shape of 2x2, xt has a shape of 2x1), output will be a 2x1 vector
  - Multiply $U_f$ with $h_{t-1}$ ($U_f$ has a shape of 2x2, ht-1 has a shape of 2x1), output will be a 2x1 vector
  - $b_f$ already is a 2x1 vector
  - So, $f_t$ will also be a 2x1 vector

Then calculate $i_t$ and $o_t$ and use these values to calculate the $C_t$ and $h_t$

$h_t = o_t * \tanh(C_t)$

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

$\hat{C}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c)$

P.S. * means element-wise multiplication

# Practice

$W_f X_t = [2\ 1]^T$, $U_f h_{t-1} = [0\ 23]^T$ $bf = [0\ 0]^T$, so $W_f x_t + U_f h_{t-1} + b = [2\ 24]^T$

$f_t = \text{sigmoid}([2\ 24]^T) = [\text{sigmoid}(2)\ \text{sigmoid}(24)]^T$

$Ct = f_t * C_{t-1} + i_t * \hat{C}_t$

If $C_{t-1} = [1\ 5]^T$

$f_t * C_{t-1} = [\text{sigmoid}(2)\ \text{sigmoid}(6)]^T * [1\ 5]^T$

$= [\text{sigmoid}(2)*1\ \text{sigmoid}(24)*5]^T$ which is another 2x1 matrix