Sign-In: https://goo.gl/YDJxqx

Code: https://goo.gl/x9d6z4

Shakespeare.txt: https://goo.gl/EbfH1p

Download and upload shakespeare.txt into your Colab folder in your google drive. Then follow instructions in colab folder.

Sequential Data

BOSTON UNIVERSITY
MACHINE INTELLIGENCE
COMMUNITY

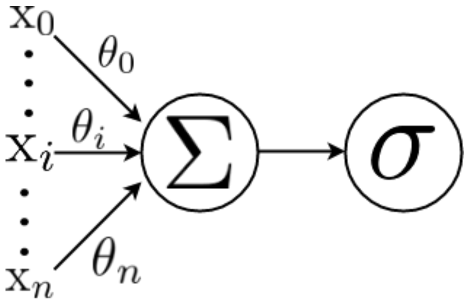
Devin de Hueck, Tyrone Hou
November, 2018

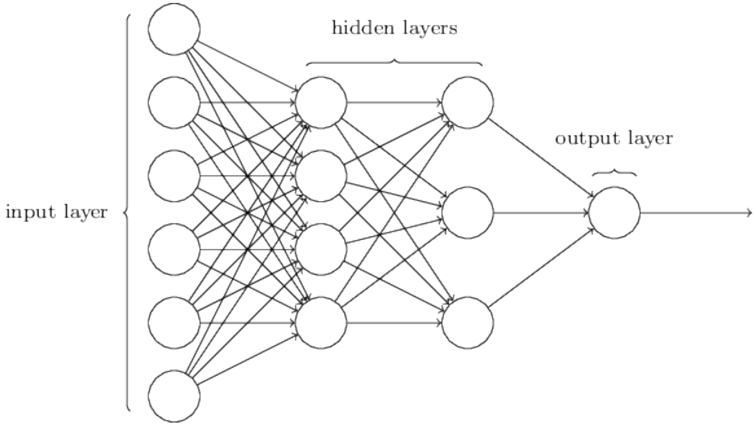
What we will cover

- Recurrent States
- Vanilla Recurrent Neural Networks
- LSTMs - Long Short Term Memory

Feedforward Neural Networks - Recap

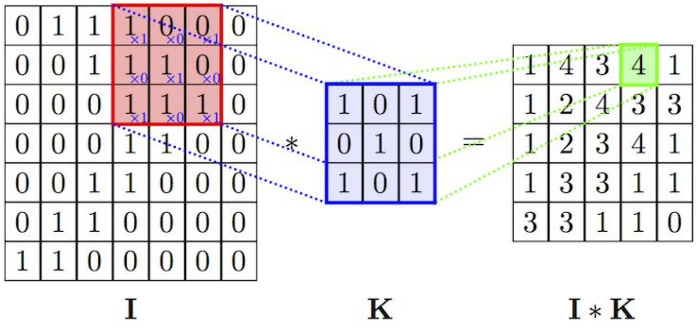
- Each neuron in a neural network takes a linear combination of its inputs and weights.
- Output of each neuron in each layer is the input to the following layer.
- Learns complex functions for a variety of tasks.

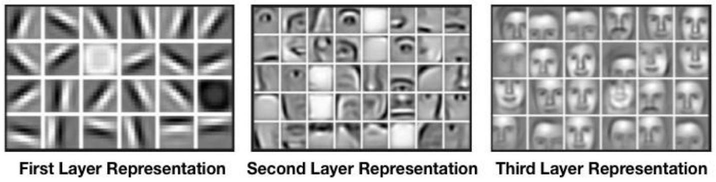




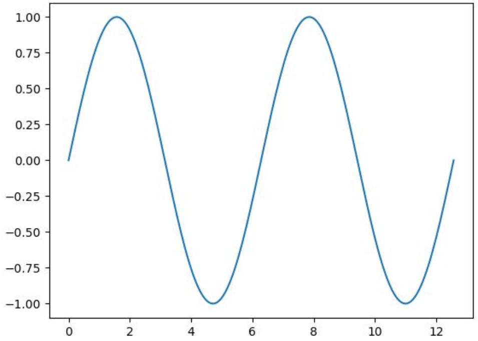
Convolutional Neural Networks - Recap

- Greatly reduces number of parameters needed when analyzing data like images compared to a feed forward neural network.
- Extracts features from images using convolutions.
    - learns lower level features like edges to task specific features.

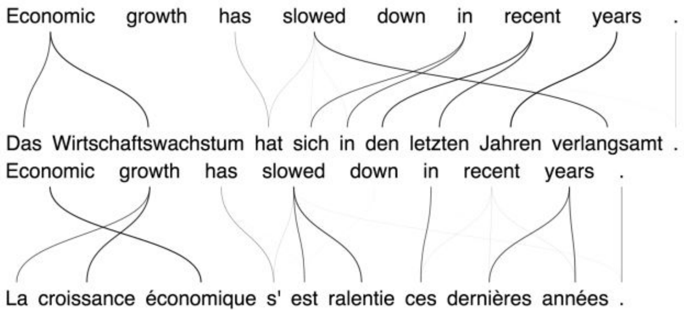




Examples of Sequential Data



"The man who wore a wig on his head went inside"



Assumptions We Made Before

Two key assumptions for feedforward networks:

Data is Independent

Fixed Input Length

Conditional Probability

- Say we want to predict the next word in a sentence.

The color of the bus is ______.

- What is the probability of a word?
  - P(yellow)
- What is the probability of a word given the previous words?
  - P(yellow | the color of the bus is)

output

Previous state

Recurrent States

Think of recurrent state as a type of a memory

$$\boldsymbol{s}^{(t)} = f(\boldsymbol{s}^{(t-1)}; \boldsymbol{\theta})$$

f - State update function

$\boldsymbol{\theta}$ - The parameters of our model

$\boldsymbol{s}^{(t)}$ - The state of our model at time t

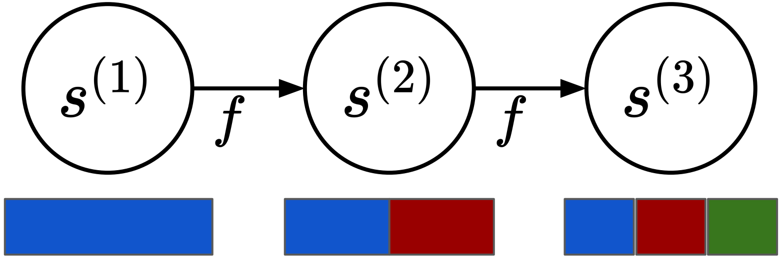
$\boldsymbol{s}^{(t-1)}$ - The state of our model at the previous timestep

Recurrent States Continued

Recurrent Example:

$$s^{(3)} = f(s^{(2)}; \theta) = f(f(s^{(1)}; \theta); \theta)$$

Unrolled Example:

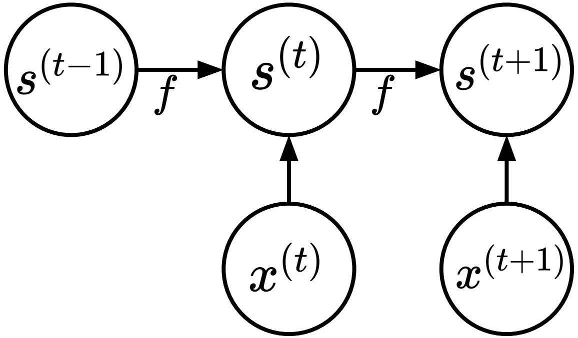


Recurrent States With External Inputs

Recurrent Example:

$$s^{(t)} = f(s^{(t-1)}, x^{(t)}; \theta)$$

Unrolled Example:



The color of the *bus* *is*

The color of the bus is _____.

Why Use Recurrent States?

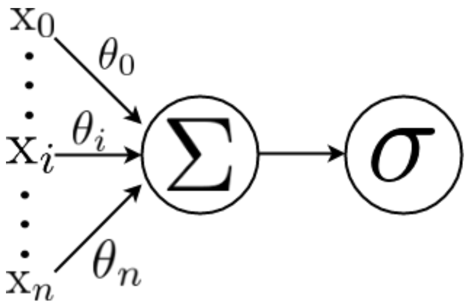
Without recurrent states we would need some function *g* for each timestep:

$$\boldsymbol{s}^{(t)} = f(\boldsymbol{s}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}) \longrightarrow \boldsymbol{s}^{(t)} = g^{(t)}(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(1)})$$

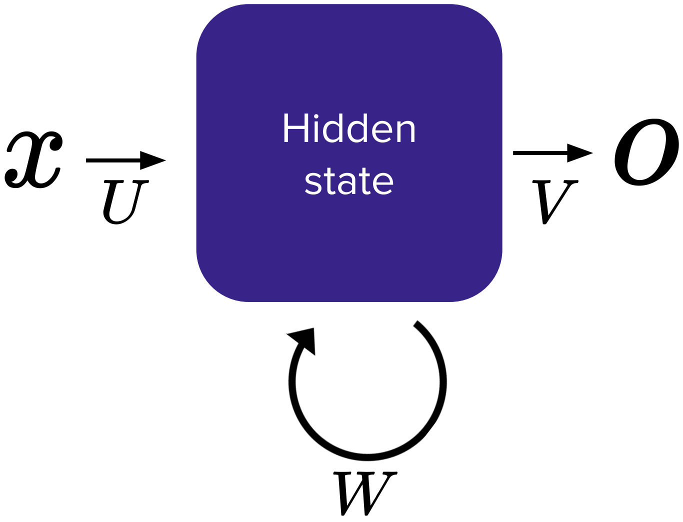
A recurrent state allows us to handle variable sequence input lengths, and use the same function with the same parameters across all timesteps.

Recurrent Neural Networks (RNNs)

Feed Forward Cell:

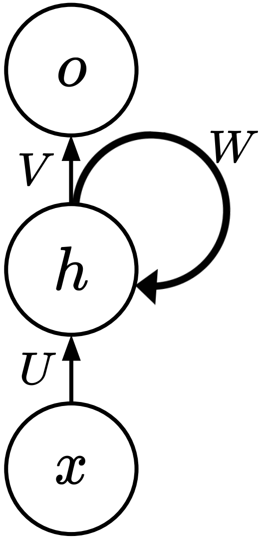


RNN Cell:



RNN Weights

RNNS:



Weights are constant throughout time.

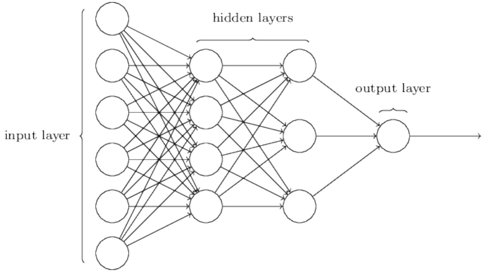
W - Hidden to Hidden Weights - **mxn**

U - Input to Hidden Weights

V - Hidden to Output Weights

*** W and U update hidden state**

Feed Forward NNs:



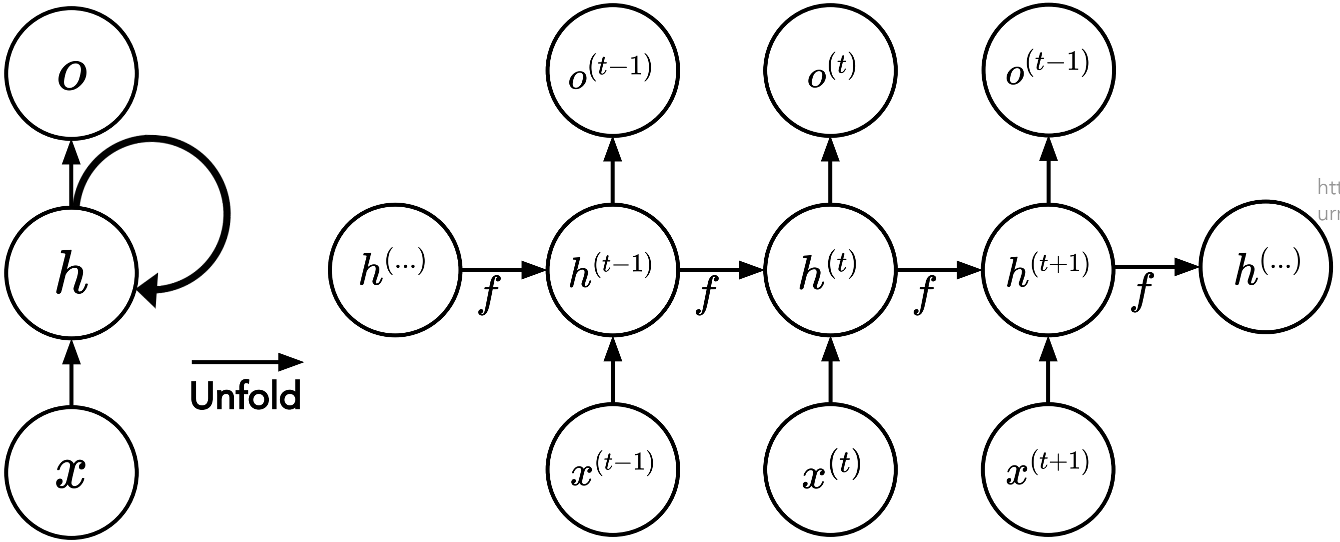
$$\sigma(Wx) = o$$

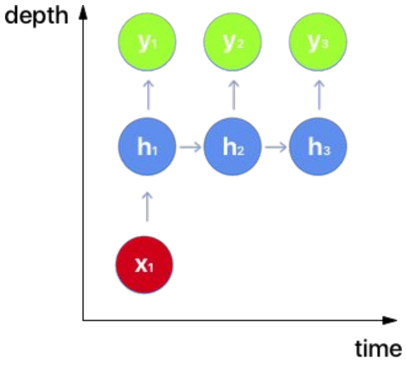
$$\sigma(\begin{bmatrix} w_{11} & w_{12} & \dots & w_{1n} \\ w_{21} & w_{22} & \dots & w_{2n} \\ \dots & \dots & \dots & \dots \\ w_{m1} & w_{m2} & \dots & w_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix})$$

Just one weight matrice per layer for feed forward NNs

Recurrent Neural Networks (RNNs)

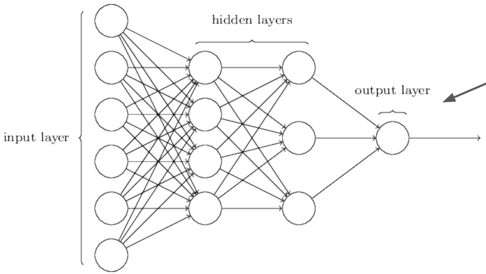
$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta)$$



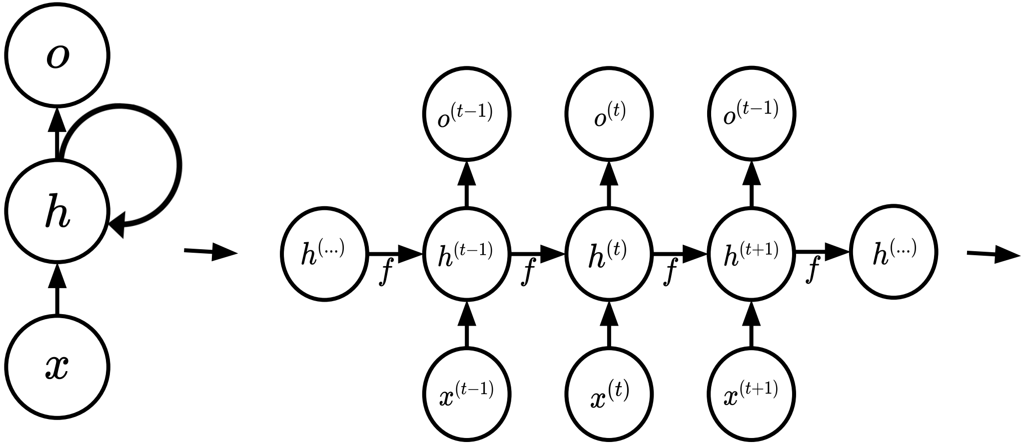


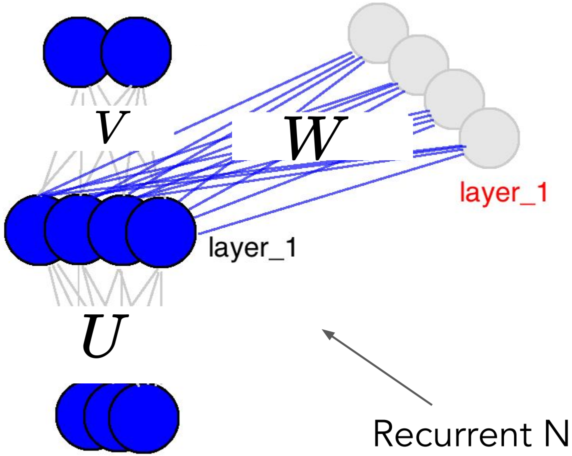
https://ayearofai.com/rohan-lenny-3-recurrent-neural-networks-10300100899b

A Quick Note on Notation



Feed Forward NN

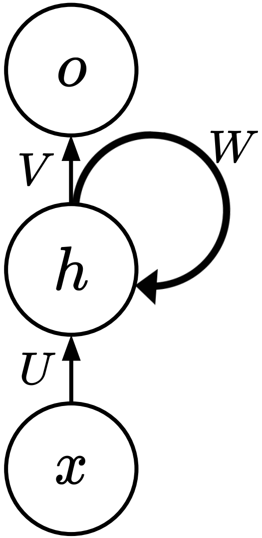




Recurrent NN

https://iamtrask.github.io/2015/11/15/anyone-can-code-lstm/

RNN Forward Propagation



bias vector

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$$

$$\downarrow$$

$$h^{(t)} = \sigma(a^{(t)})$$

$$\downarrow$$

$$o^{(t)} = c + Vh^{(t)}$$

bias vector

A Common Input

Vocabulary: Total words or characters used

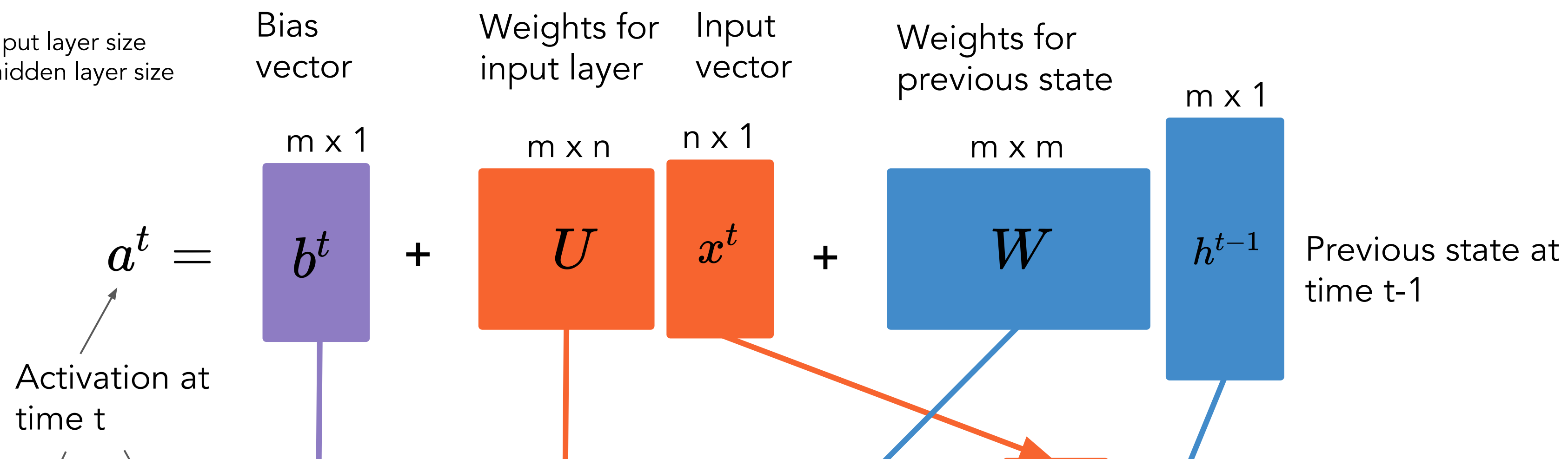
$$\begin{bmatrix} hello \\ john \\ \dots \\ sky \\ red \end{bmatrix}$$

One Hot Vector:

$$\begin{bmatrix} 0 \\ 0 \\ \dots \\ 1 \\ 0 \end{bmatrix} = \boldsymbol{sky}$$

Forward Propagation - Matrix Representation

n: input layer size
m: hidden layer size



$$a^t = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} + \begin{bmatrix} u_{11} & u_{12} & w_{11} & w_{12} & w_{13} \\ u_{21} & u_{22} & w_{21} & w_{22} & w_{23} \\ u_{31} & u_{32} & w_{31} & w_{32} & w_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix}$$

Forward Propagation - Matrix Representation

m: hidden layer size
k: output layer size

m x 1

$$h^{(t)} = \sigma \odot a^{(t)} \quad \rightarrow \quad \underbrace{V}_{k \times m}\ \underbrace{h^{(t)}}_{m \times 1} = \hat{y}^{(t)}$$

$$h^{(t)} = \sigma \odot \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \\ v_{31} & v_{32} & v_{33} \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} = \hat{y}^{(t)}$$

Backpropagation Through Time - BPTT

Target matrix

$$y = \begin{bmatrix} y_1 & y_2 & \dots & y_t \end{bmatrix}$$

Loss function for single time step

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

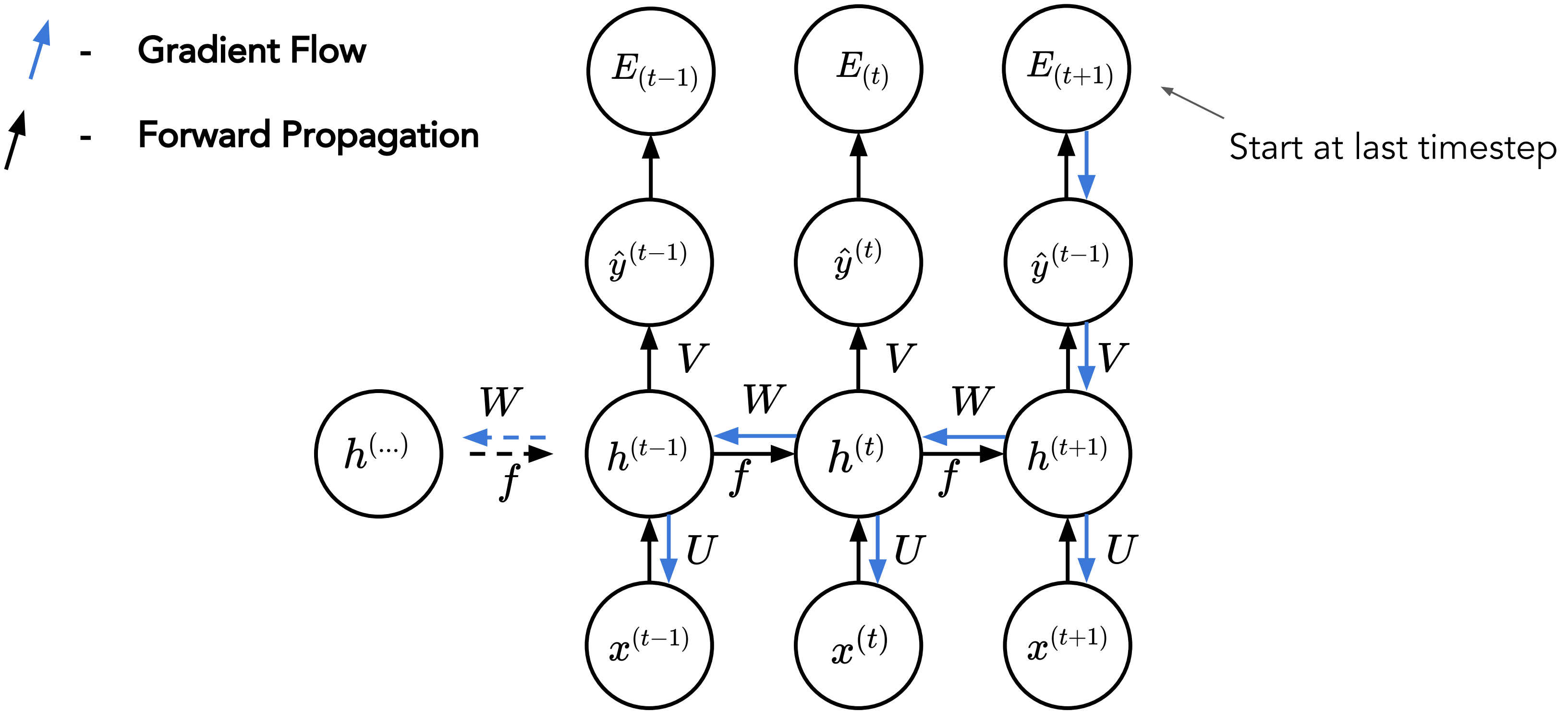
Cross entropy loss

Loss function across all time steps

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

Sum over every time step

Backpropagation Through Time - BPTT



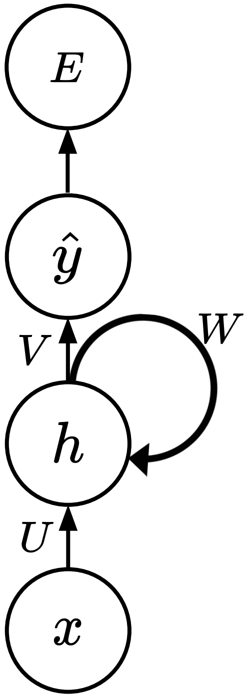
Backpropagation Through Time - BPTT

$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t \qquad h_t = tanh(Ux + Wh_{t-1}) \qquad \hat{y} = softmax(Vh_t)$$

$$\frac{\partial E_t}{\partial V}$$

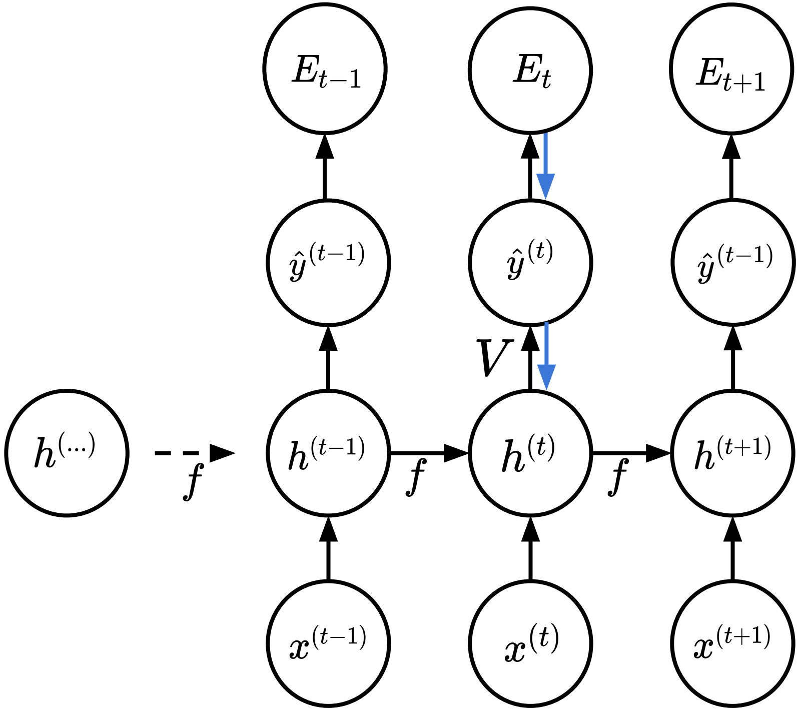
$$\frac{\partial E_t}{\partial U}$$

$$\frac{\partial E_t}{\partial W}$$



Backpropagation Through Time - BPTT

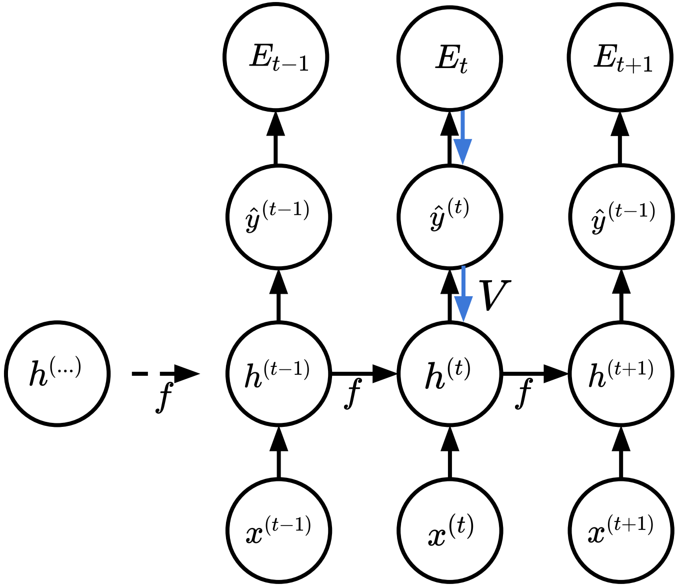
$$\frac{\partial E_t}{\partial V}$$



Backpropagation Through Time - BPTT

$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t \qquad h_t = tanh(Ux + Wh_{t-1}) \qquad \hat{y} = softmax(Vh_t)$

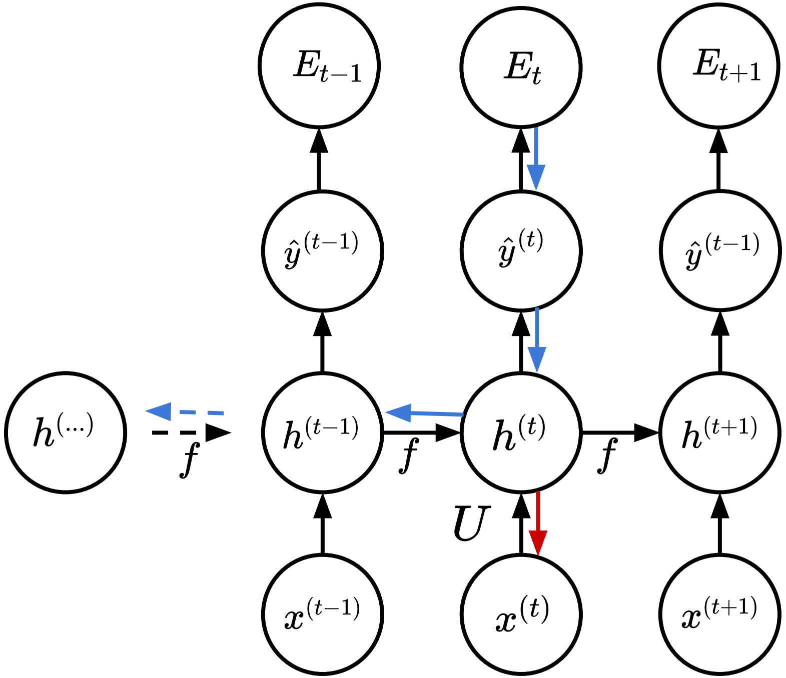
$$\frac{\partial E_t}{\partial V} = \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial V h_t} \frac{\partial V h_t}{\partial V}$$
$$= (\hat{y} - y) \otimes h_t$$



Backpropagation Through Time - BPTT

$$s^{(3)} = f(s^{(2)}; \theta) = f(f(s^{(1)}; \theta); \theta)$$

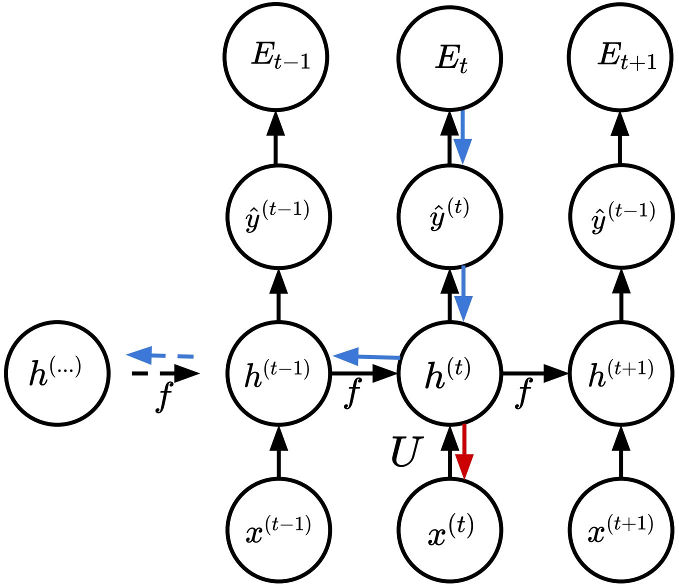
$$\frac{\partial E_t}{\partial U}$$



Backpropagation Through Time - BPTT

$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$ $\qquad h_t = tanh(Ux + W\underline{h_{t-1}})$ $\qquad \hat{y} = softmax(Vh_t)$

$$\frac{\partial E_t}{\partial U} = \sum_{k=0}^{t} \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \boxed{\frac{\partial h_t}{\partial h_k}} \frac{\partial h_k}{\partial U}$$



Backpropagation Through Time - BPTT

$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t \qquad h_t = tanh(Ux + Wh_{t-1}) \qquad \hat{y} = softmax(Vh_t)$$

Multiplying all previous state gradients together

$$\frac{\partial E_t}{\partial U} = \sum_{k=0}^{t} \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \boxed{\frac{\partial h_t}{\partial h_k}} \frac{\partial h_k}{\partial U}$$

$$\frac{\partial E_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \boxed{\frac{\partial h_t}{\partial h_k}} \frac{\partial h_k}{\partial W}$$

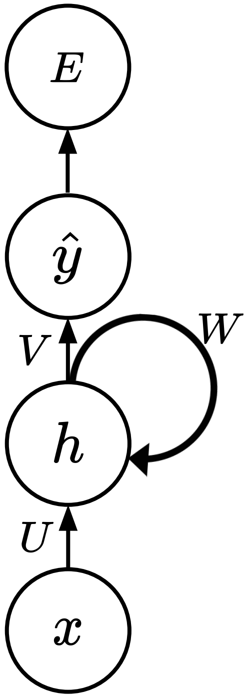
$$\begin{aligned} \frac{\partial h_t}{\partial h_k} = \frac{\partial h_4}{\partial h_1} &= \frac{\partial h_4}{\partial h_3} \frac{\partial h_3}{\partial h_1} \\ &= \frac{\partial h_4}{\partial h_3} \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1} \end{aligned}$$

Backpropagation Through Time - BPTT

$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$ $\quad h_t = tanh(Ux + Wh_{t-1})$ $\quad \hat{y} = softmax(Vh_t)$

$$\frac{\partial E_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

$$= \sum_{k=0}^{t} \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \left(\prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial W}$$



BPTT Gradient Update

learning rate

cost/error function

parameter matrices

$$W := W + \alpha \frac{\partial}{\partial W} E(y, \hat{y})$$

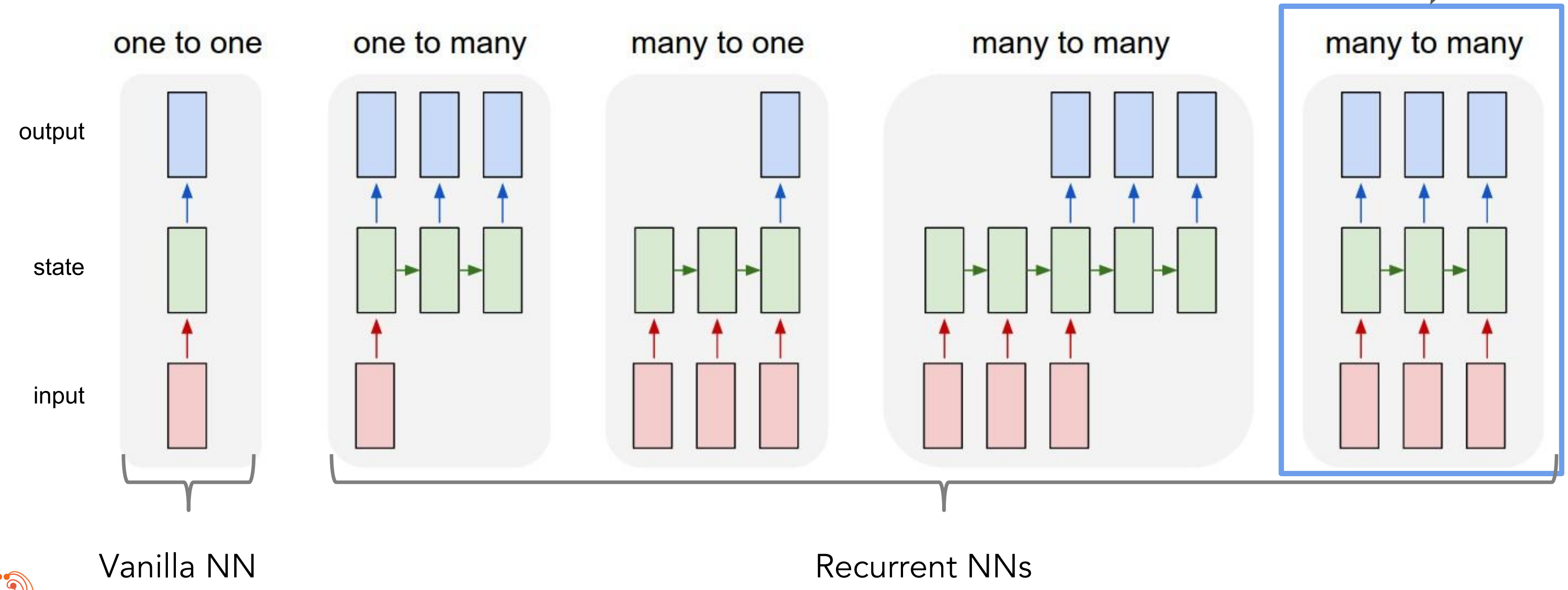
$$V := V + \alpha \frac{\partial}{\partial V} E(y, \hat{y})$$

$$U := U + \alpha \frac{\partial}{\partial U} E(y, \hat{y})$$

Architectures

- A single RNN Cell can function as a complete network

What we just did

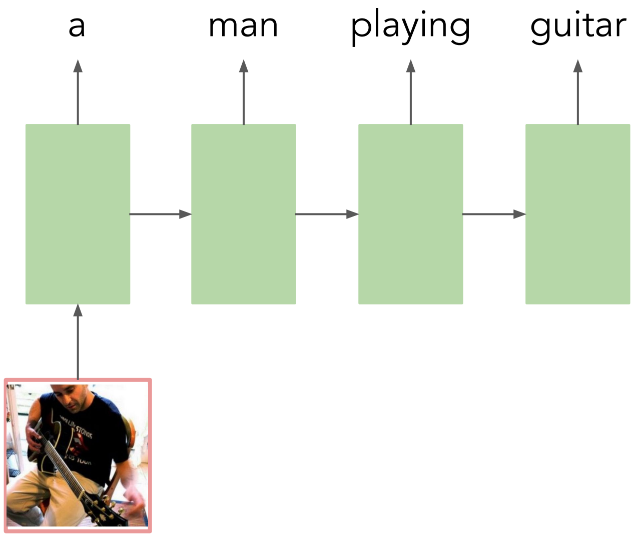


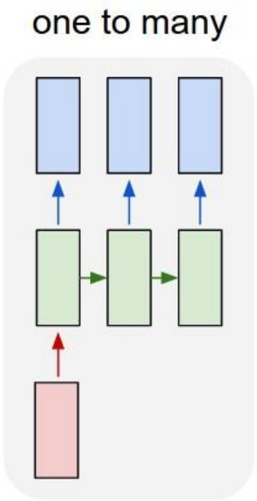
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Architectures - One to Many

<u>Example:</u>

Image captioning

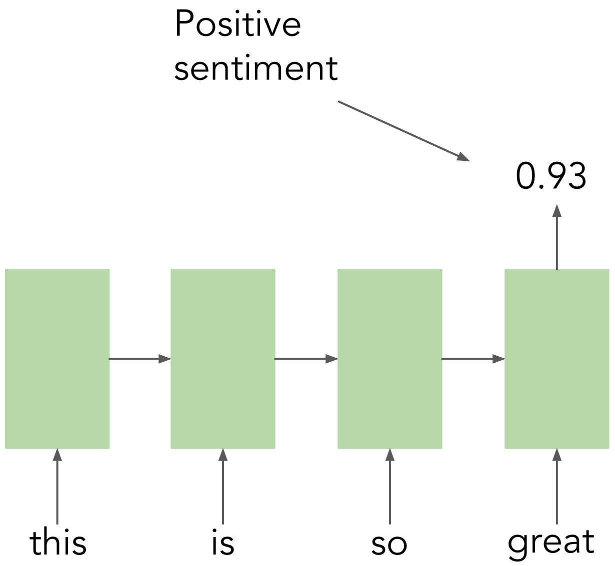


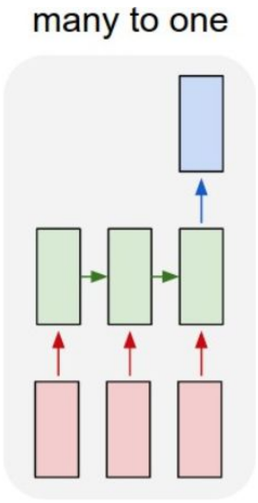


Architectures - Many to One

<u>Example:</u>

Sentiment analysis
/Any regression task

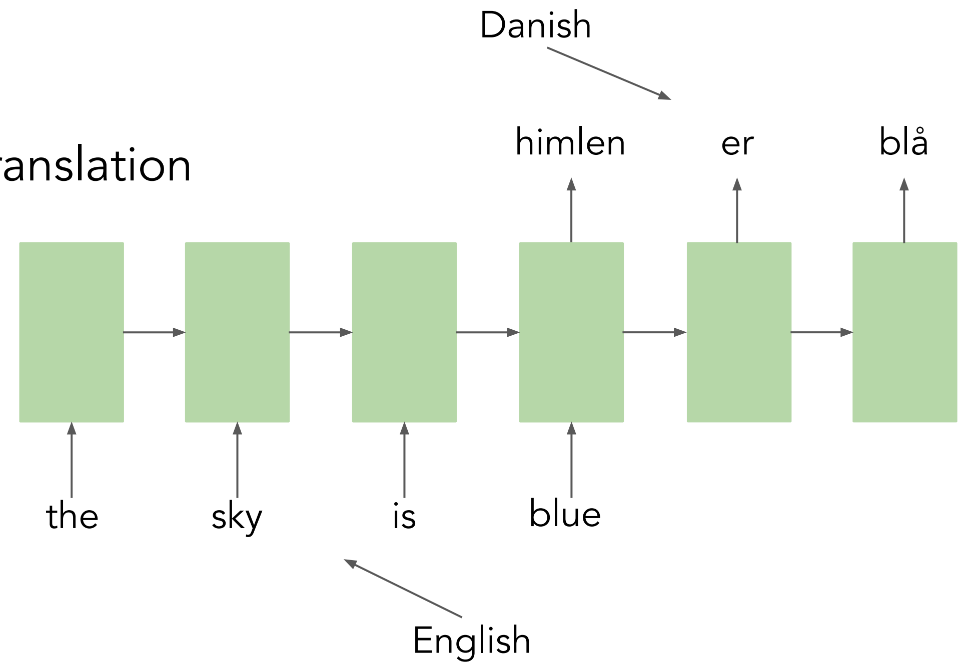


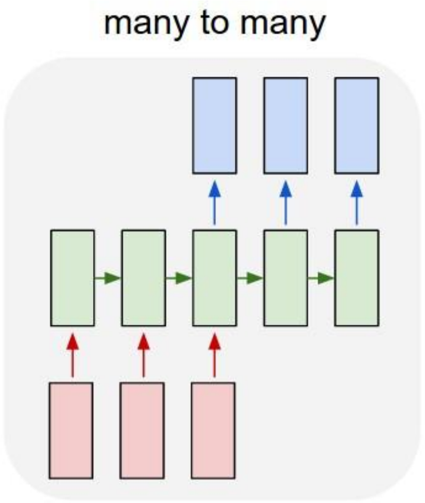


Architectures - Many to Many

<u>Example:</u>

Machine Translation

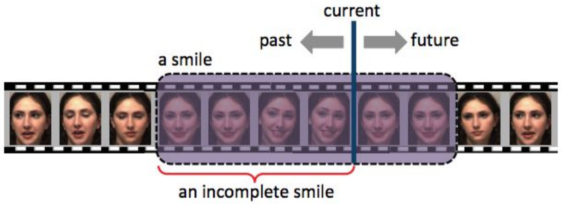


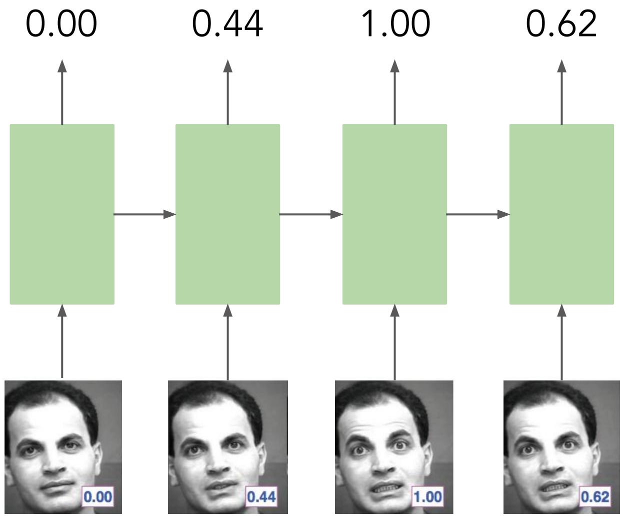


Architectures - Many to Many Synced

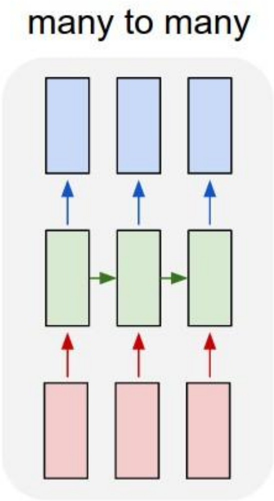
<u>Example:</u>

Early event detection



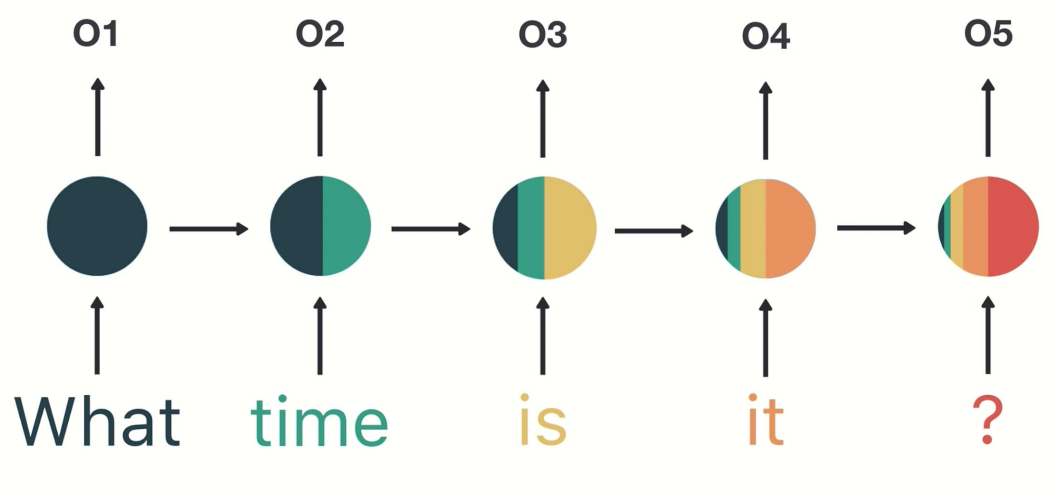


Hoai, M., De la Torre, F.: Max-margin early event detectors. In: CVPR. (2012)



The Problem of Long-Term Dependencies

"The clouds are in the ***sky***." VS "I grew up in France… I speak fluent ***French***."

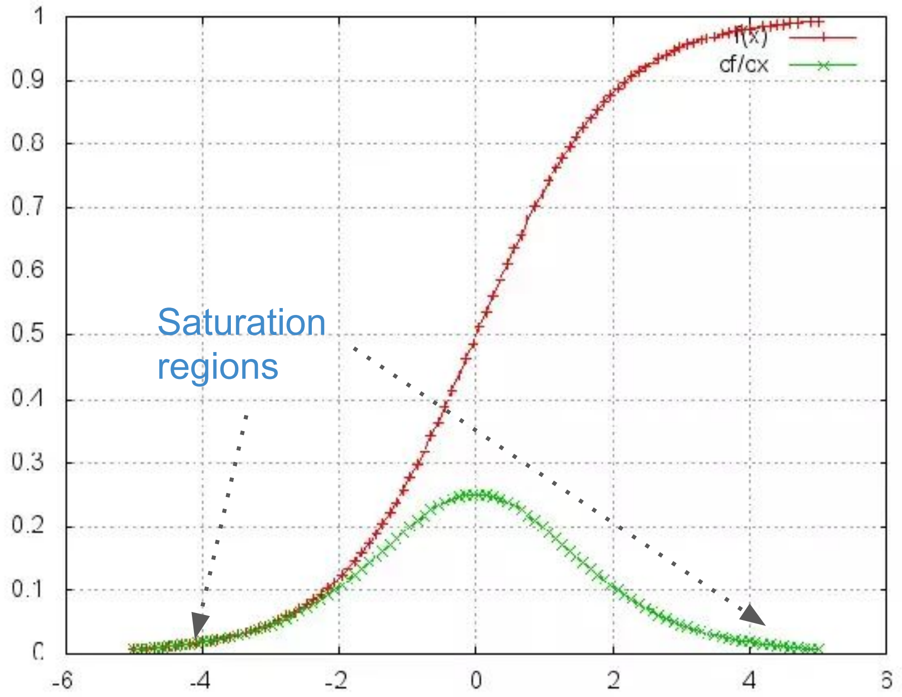


https://www.youtube.com/watch?v=LHXXI4-IEns

The Problem of Long-Term Dependencies - Sigmoid/Tanh Gradient

- As the value of **sigmoid** and **tanh** function is close to 0 or 1, the derivative approaches 0.

- *What happens when the gradient is backpropagated?*



Vanishing & Exploding Gradients

Vanishing Gradient

$$\lim_{x \to \infty} 0.5^x = 0$$

As the product of partial derivatives approaches zero, the gradient vanishes.

Exploding Gradient

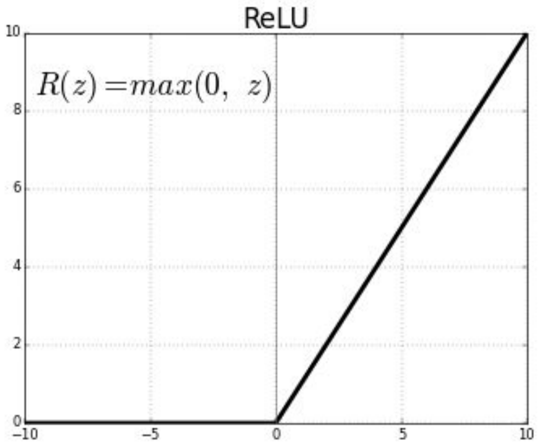
$$\lim_{x \to \infty} 1.5^x = \infty$$

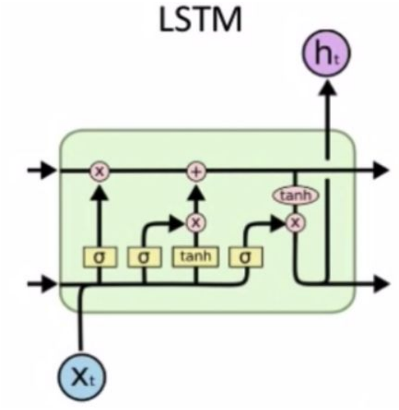
As the product of partial derivatives approaches infinity, the gradient explodes.

$$\frac{\partial E_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \left(\boxed{\prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}} \right) \frac{\partial h_k}{\partial W}$$

Solutions to Vanishing and Exploding Gradient Problem

- Use the ReLu activation function
- Gradient Capping
- LSTMs (a more complex architecture)

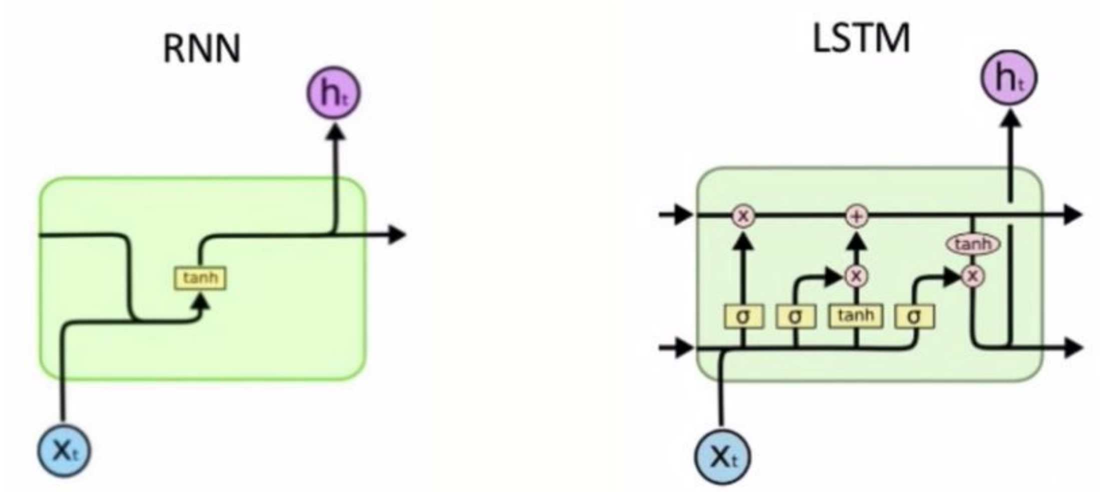




LSTM Network

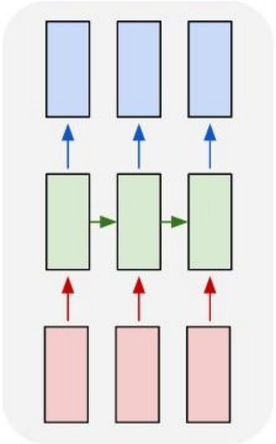
- Long Short Term Memory
  - Extension of Vanilla RNN cells to handle the problem of long-term dependencies
- Key Concepts
  - **Cell Memory** - A representation of all the LSTM's knowledge
  - **Hidden State** - A specific part of memory that the LSTM outputs
  - **Gates** - Interface that controls what is added and deleted from memory

Vanilla RNNs - LSTMs Compared



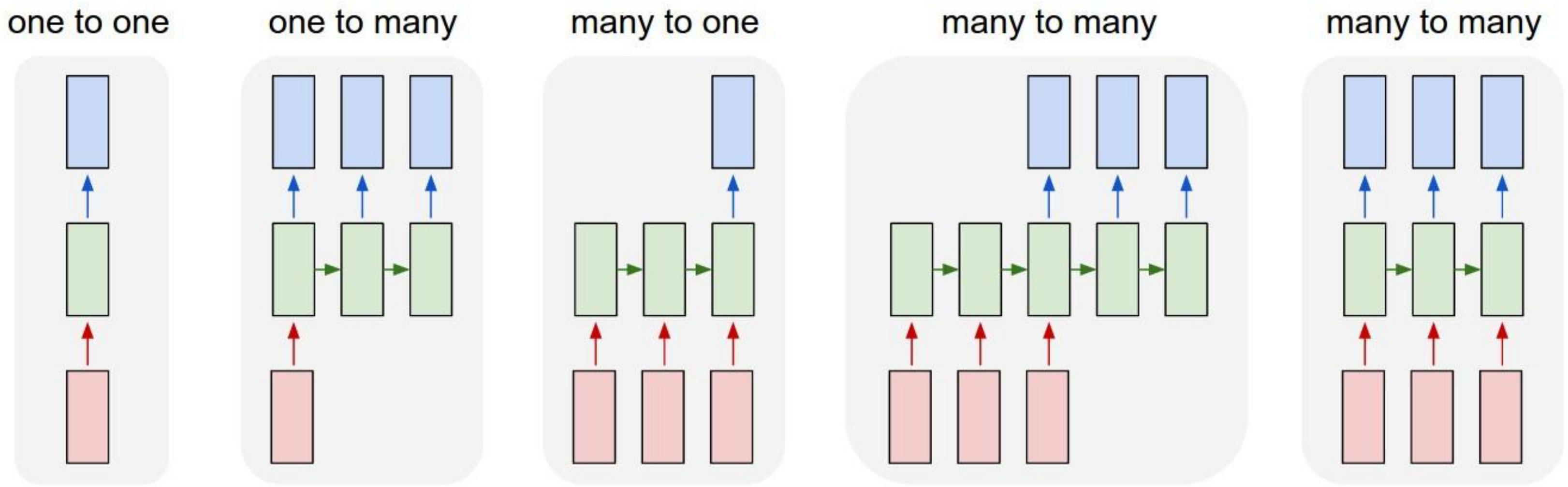
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Can be used as the same cell in the architectures just covered



LSTM Architectures

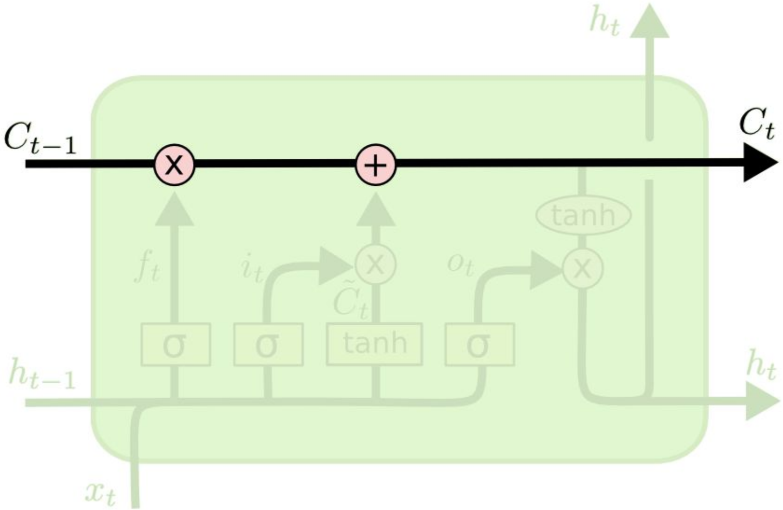
We can use every architecture we have mentioned with LSTMs as the base cell.



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

Cell State

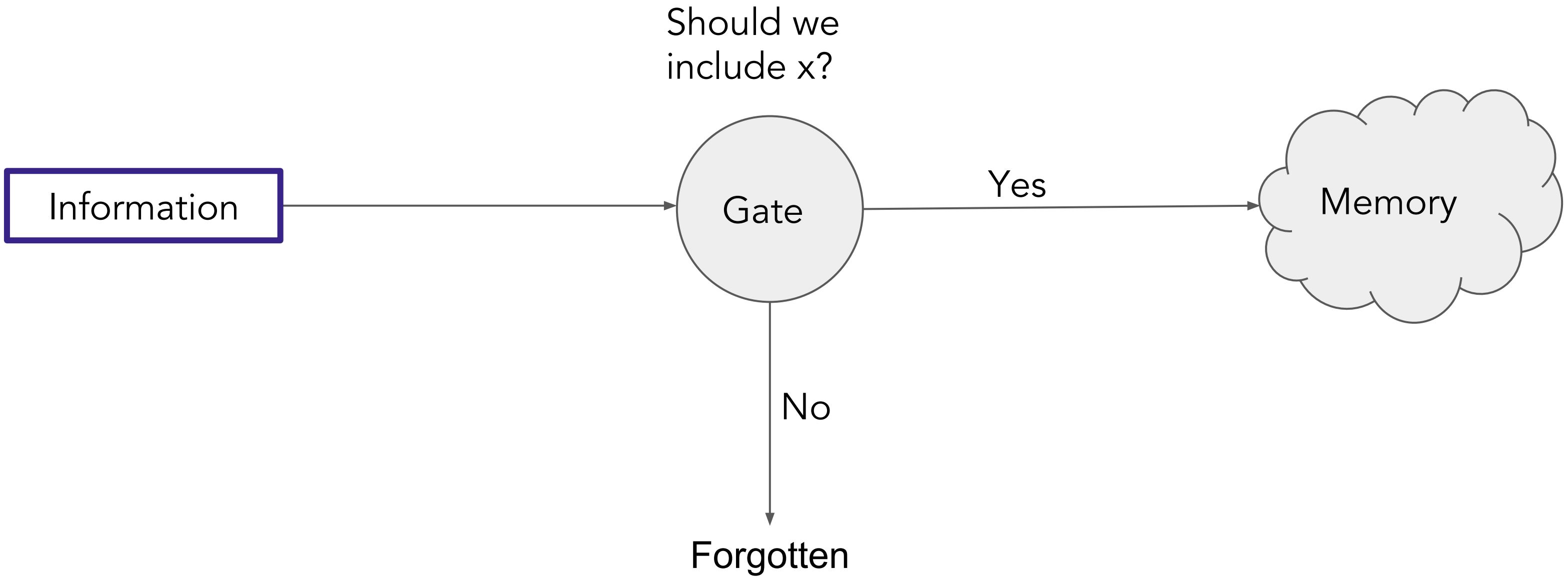
- Easy for information to flow unchanged
  - Only has linear interactions
- LSTM has the ability to add or remove information from cell state
- Flows right through



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

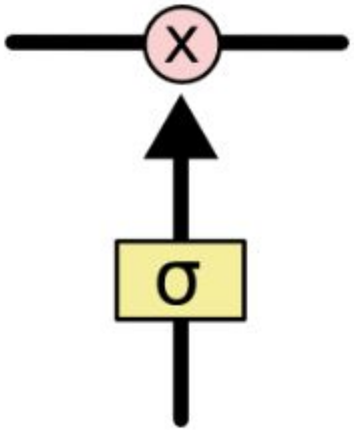
Gated Memory

How do we control what goes in and out of memory?

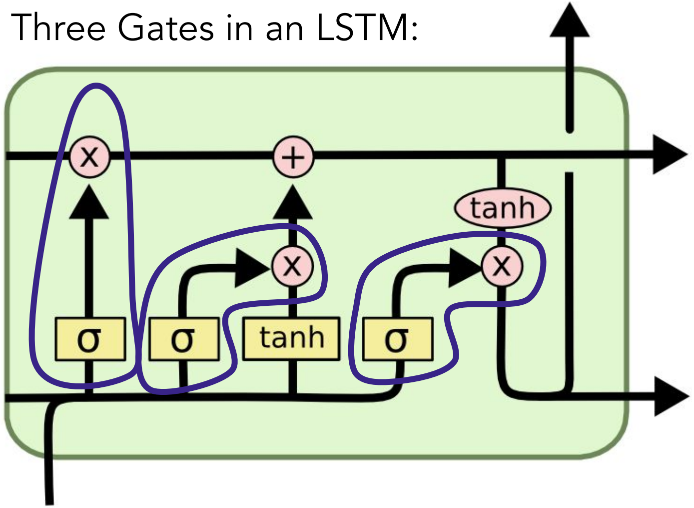


Gated Memory

How do we control what goes in and out of memory?



Three Gates in an LSTM:

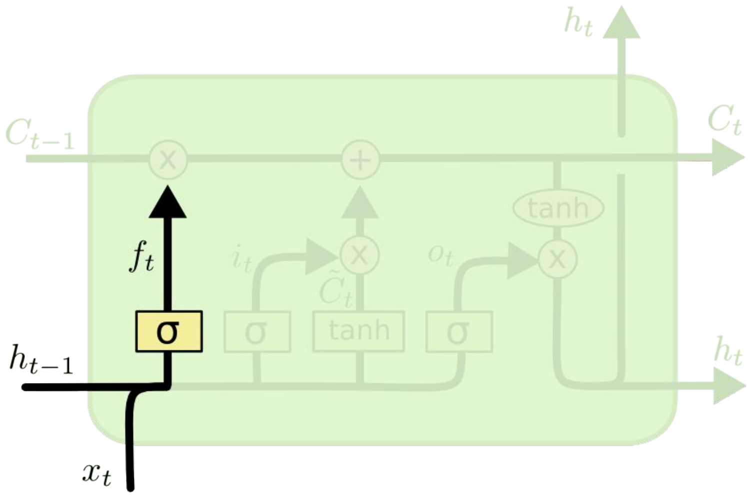


A zero means "completely forget this". A one means "completely keep this".

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

LSTM Cell - Forget Gate

What do we want to forget?



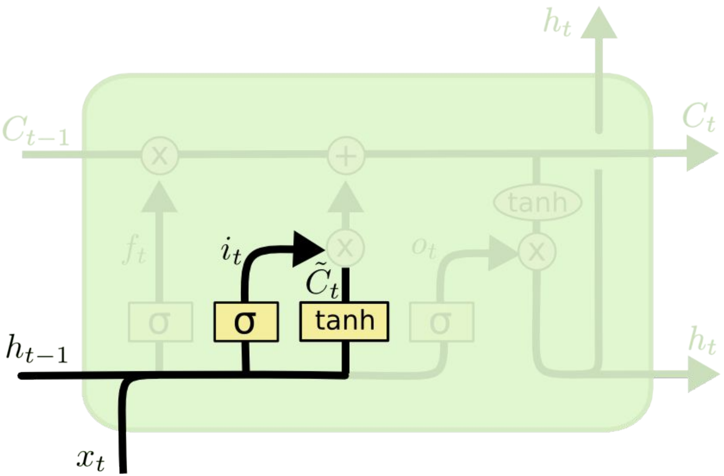
$$f_t = \sigma\left(W_f \cdot \underline{[h_{t-1}, x_t]} + b_f\right)$$

Append previous hidden state to input value

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

LSTM Cell - Input Gate

What do we want to add to the cell state?



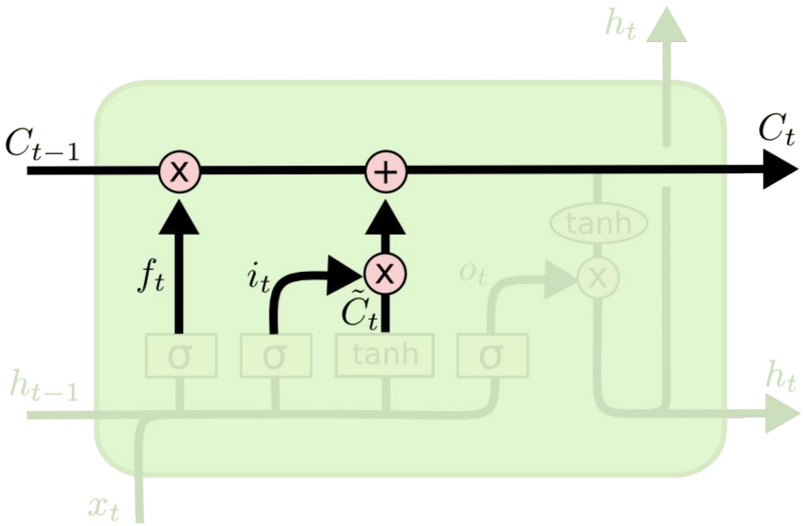
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

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

LSTM Cell - Add to Cell State

Add what we decided to add to the cell state.

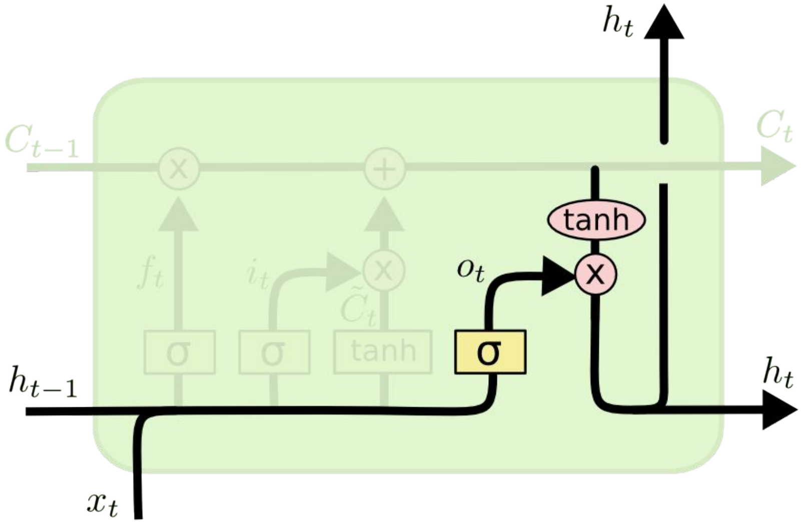


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

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

LSTM Cell - Output Gate

What do we to output?



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

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

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

Summary

- We use a recurrent state to model sequential data so that we can share parameters and handle variable length inputs
- Vanilla RNNs face the problem of long-term dependencies
- LSTMs are one architecture to mitigate the problem of long-term dependencies
- Recurrent Cells whether they are LSTMs or Vanilla RNNs can be used in different overall network architectures (Many-to-Many, One-to-Many, etc.)

References & Further Reading

1. Goodfellow, Ian, et al. "Deep learning." Vol. 1. Cambridge: MIT press, (2016).
2. https://iamtrask.github.io/2015/11/15/anyone-can-code-lstm/
3. http://colah.github.io/posts/2015-08-Understanding-LSTMs/
4. http://karpathy.github.io/2015/05/21/rnn-effectiveness/
5. https://github.com/go2carter/nn-learn/blob/master/grad-deriv-tex/rnn-grad-deriv.pdf

Upcoming Events

- First Paper Discussion tomorrow (Thu. 25th) - ResNet
- Hacknight Sunday
- Deep Reinforcement Learning workshop next Wednesday
- Looking for volunteers for the Machine Intelligence Conference on Nov. 3rd