Sequential Data

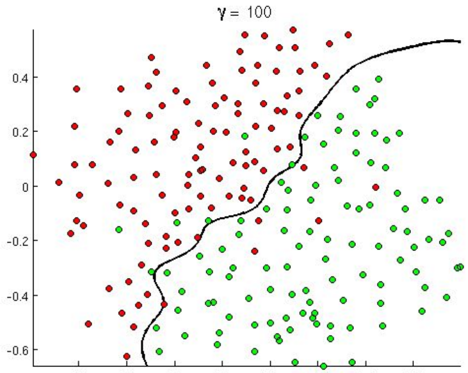
BOSTON UNIVERSITY
MACHINE INTELLIGENCE COMMUNITY

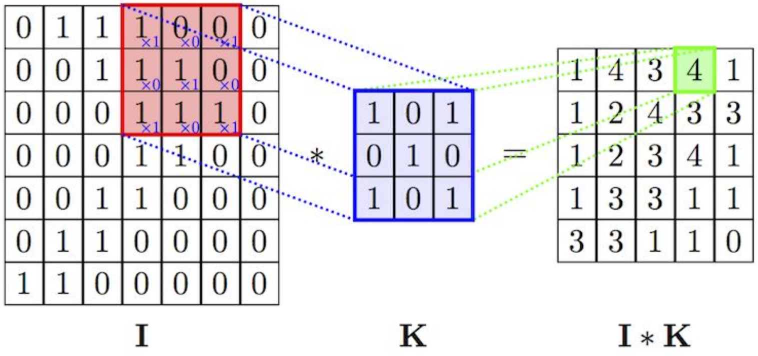
Tyrone Hou
Nov. 7, 2017

https://deepjazz.io/

Brief Recap: Neural Network Architectures

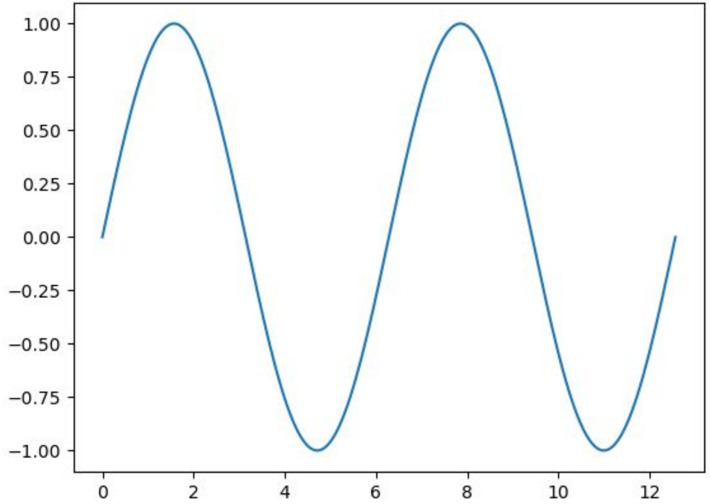
- Feedforward Networks take the weighted sum of inputs passed through a nonlinear activation function to learn patterns within the data.
- Convolutional Neural Networks perform local convolutions of the input data with a kernel to learn spatial patterns within a data sample.





Sequential Data Patterns

- Convolutional and Feedforward networks are agnostic to input order
  - Only localized/relative patterns within a single input are learned
  - To learn relationships across a sequence of inputs, we need to keep track of state



M I C

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 State Update

$$\vdots$$

network state

$$s_{t-1} = f(s_{t-2}, \overbrace{x_1, x_2, \ldots, x_{t-1}}^{\text{input sequence}}, \phi)$$

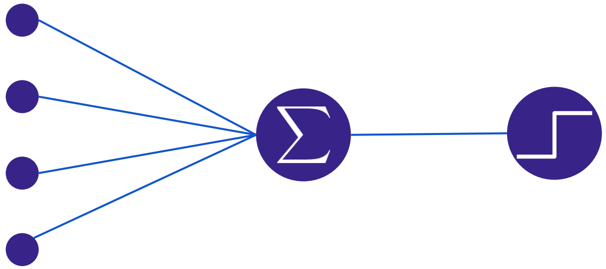
$$s_t = f(s_{t-1}, x_1, x_2, \ldots, x_t, \phi)$$

network parameters

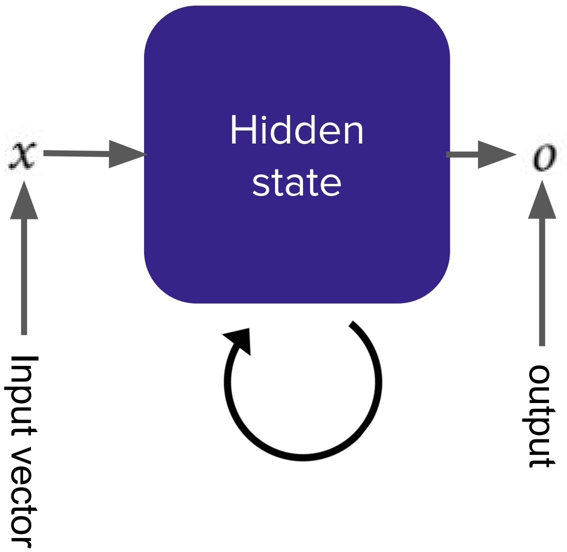
$$\vdots$$

RNN Cell

Perceptron

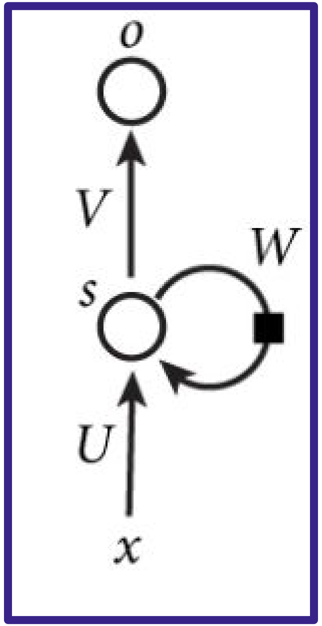


RNN Cell



RNN Cell

RNN Cell



$$s_t = f(s_{t-1}, x_1, x_2, \ldots, x_t, \phi)$$

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

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

Matrix Representation

Weights for input layer

Input vector

Weights for previous state

$$s_t = U \; x_t + W \; s_{t-1}$$

Previous state at time t-1

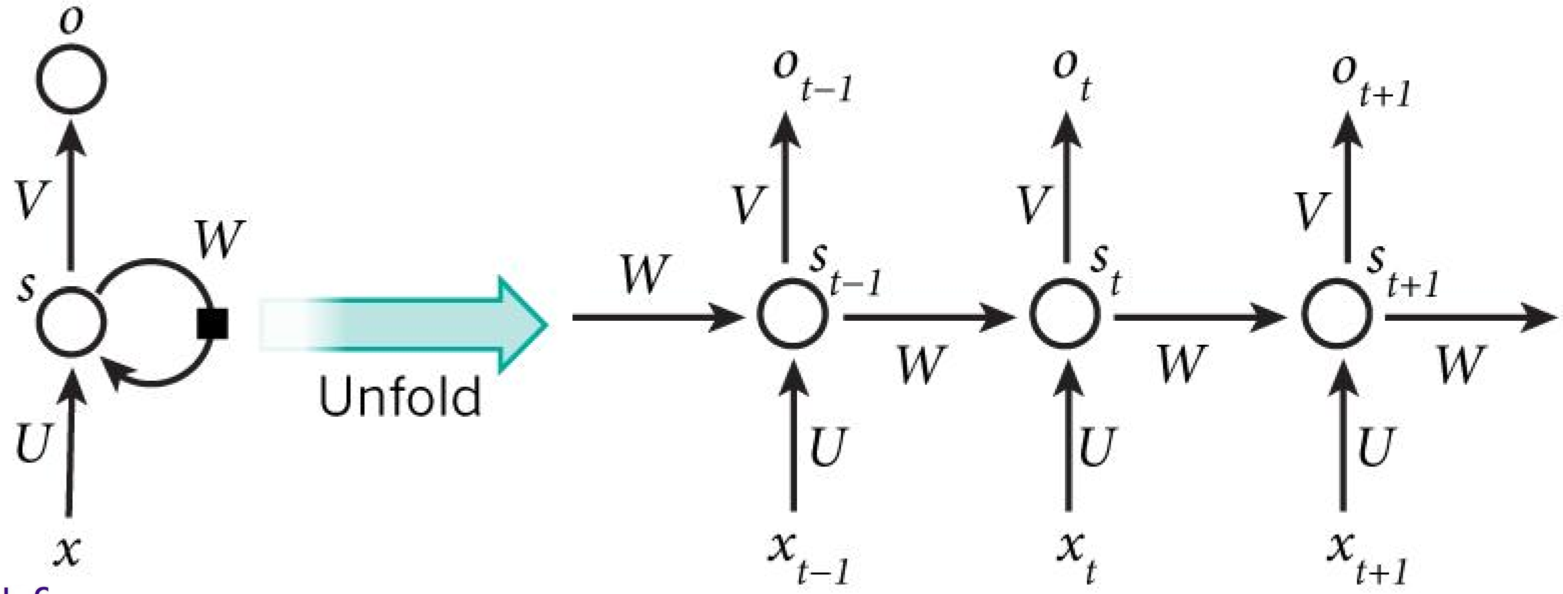
Current state at time t

$$s_t = \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \\ u_{31} & u_{32} \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix}$$

M I C

Forward Pass

- Similar to forward pass for feedforward network
    - **Weight sharing/tying**: weights are shared across all time steps



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

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

M I C

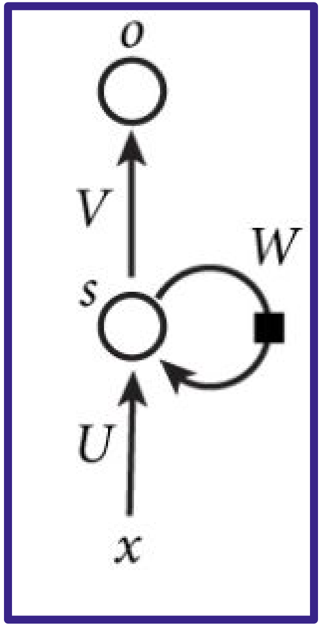
Backpropagation Through Time cont.

$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$ $\quad s_t = tanh(Ux + Ws_{t-1})$ $\quad \hat{y} = softmax(Vs_t)$

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

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

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

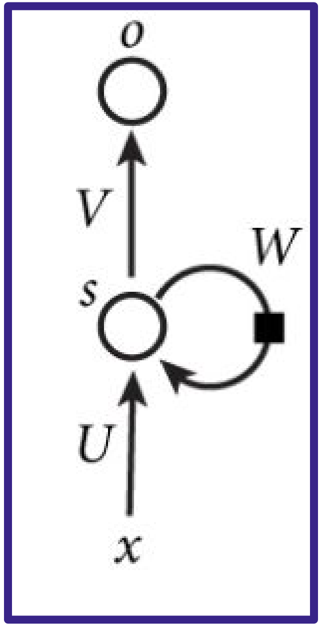


M I C

Backpropagation Through Time cont.

$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$ $\quad s_t = tanh(Ux + Ws_{t-1})$ $\quad \hat{y} = softmax(Vs_t)$

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

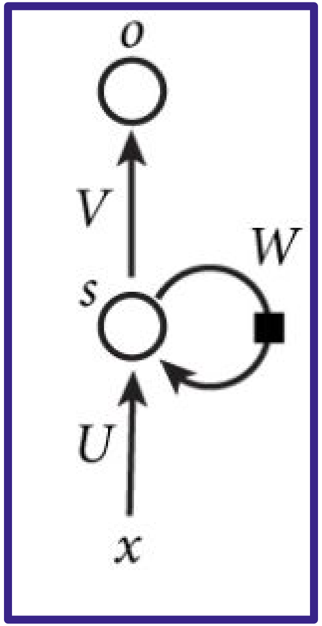


M I C

Backpropagation Through Time cont.

$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t \qquad s_t = tanh(Ux + Ws_{t-1}) \qquad \hat{y} = softmax(Vs_t)$

$$\begin{aligned}\frac{\partial E_t}{\partial V} &= \frac{\partial E_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial V s_t}\frac{\partial V s_t}{\partial V} \\ &= (\hat{y}_t - y_t) \otimes s_t\end{aligned}$$



M I C

Backpropagation Through Time cont.

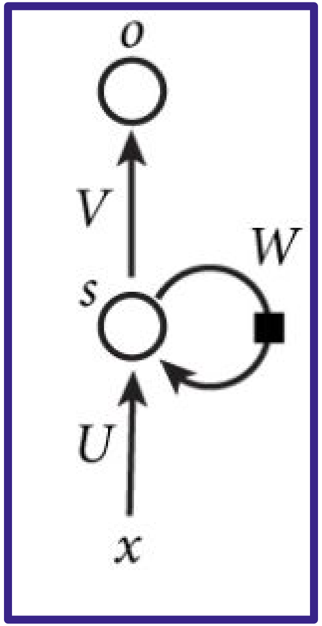
$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$

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

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

$$s_{t-1} = tanh(Ux + Ws_{t-2})$$

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



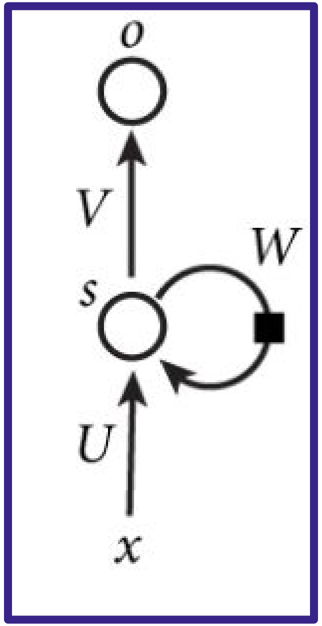
M I C

Backpropagation Through Time cont.

$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t \qquad s_t = tanh(Ux + Ws_{t-1}) \qquad \hat{y} = softmax(Vs_t)$$

$$s_{t-1} = tanh(Ux + Ws_{t-2})$$

$$\frac{\partial E_t}{\partial U} = \sum_{k=0}^{t} \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial s_t} \frac{\partial s_t}{\partial s_k} \frac{\partial s_k}{\partial U}$$



Backpropagation Through Time cont.

$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t \qquad s_t = tanh(Ux + Ws_{t-1}) \qquad \hat{y} = softmax(Vs_t)$$

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

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

$$\frac{\partial s_t}{\partial s_k} = \frac{\partial s_4}{\partial s_1} = \frac{\partial s_4}{\partial s_3} \frac{\partial s_3}{\partial s_1}$$

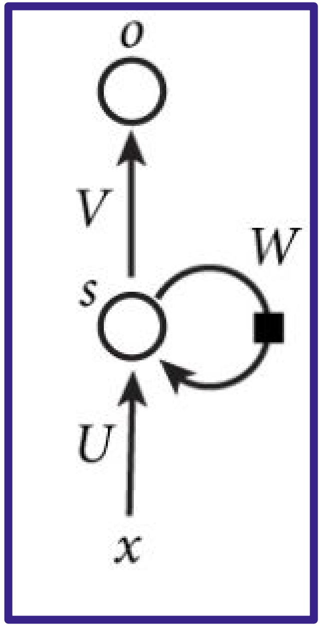
$$= \frac{\partial s_4}{\partial s_3} \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1}$$

Backpropagation Through Time cont.

$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t \qquad s_t = tanh(Ux + Ws_{t-1}) \qquad \hat{y} = softmax(Vs_t)$$

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

$$= \sum_{k=0}^{t} \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial s_t} \left(\prod_{j=k+1}^{t} \frac{\partial s_j}{\partial s_{j-1}} \right) \frac{\partial s_k}{\partial W}$$



M I C

Gradient update

parameter matrix

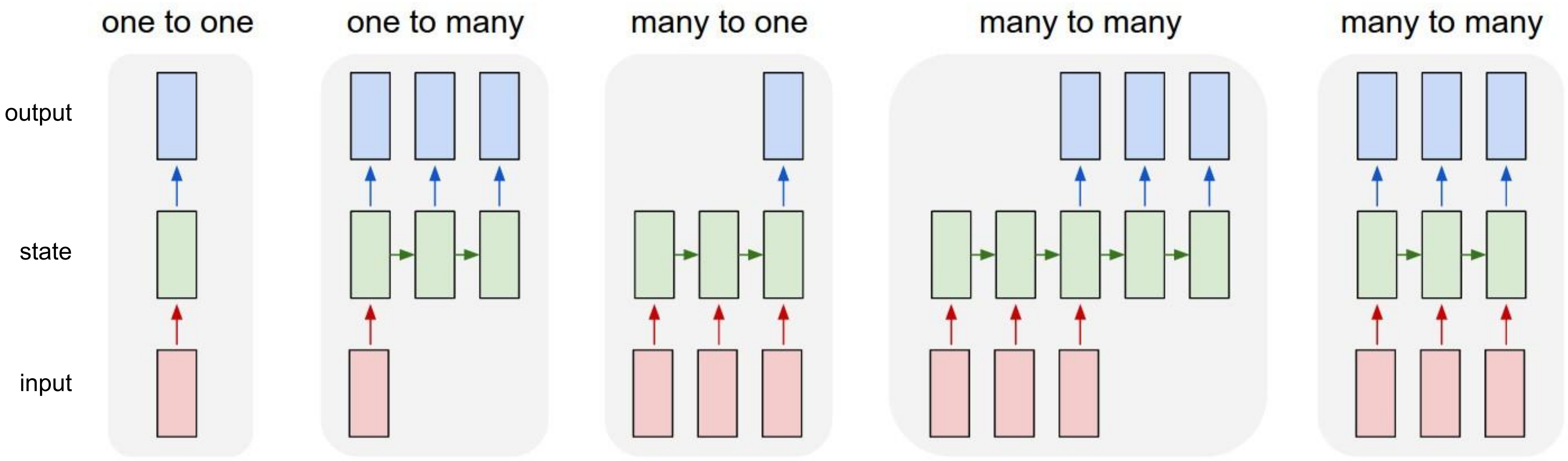
learning rate

cost/error function

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

Architecture: Single Cell

- A single RNN Cell can function as a complete network



Architecture: Single Cell

One to one: Character/sentence generation

One to many: Image captioning

Many to one: Sentiment analysis

Many to many: Sentence translation

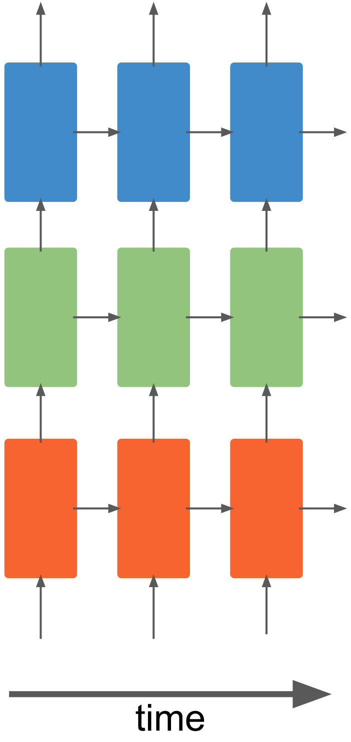
Architecture: Stacked RNNs

- Network learns hierarchical relationships through time
- Example: Word generation

Layer 3
Word Discrimination

Layer 2
Two letter combinations

Layer 1
Vowels vs Consonants



RNN example

```
>> import torch.nn as nn
>> cell = nn.RNNCell(10, 20,
nonlinearity='relu')
>> cell
RNNCell(10, 20, nonlinearity=relu)
```

```
>> import torch.nn as nn
>> cell = nn.LSTMCell(10, 20,
nonlinearity='relu')
>> cell
LSTMCell(10, 20,
nonlinearity=relu)
```

```
>> net = nn.RNN(400, 10, 3)
>> net
RNN(400, 100, 3)
>>
```

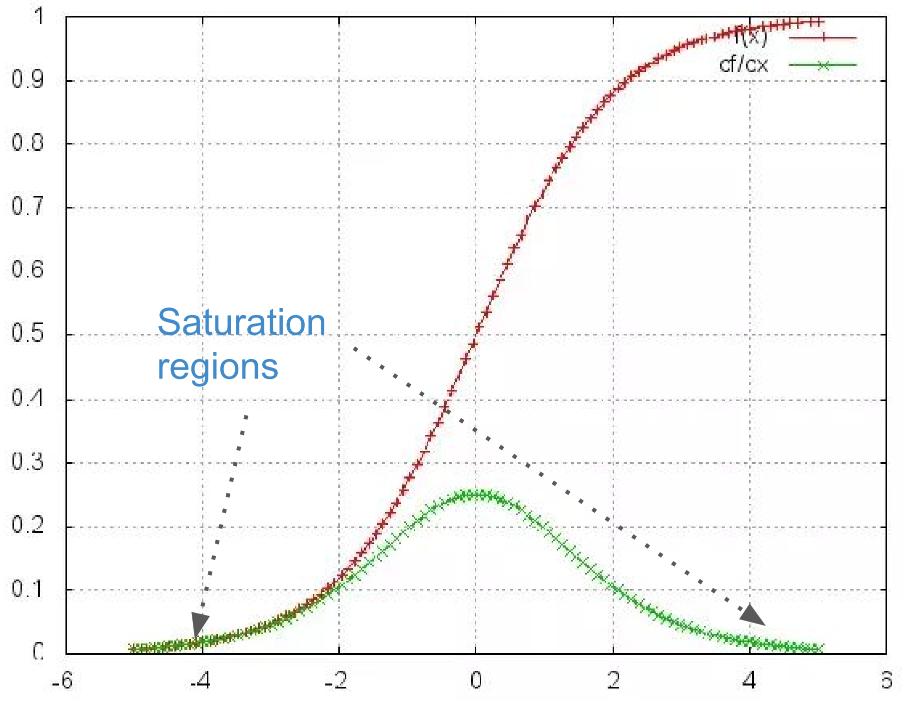
```
>> net = nn.LSTM(20, 20, 1)
>> net
RNN(20, 20, 1)
>>
```

M I C

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.

$$\frac{\partial E_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial E_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial s_t} \boxed{\left(\prod_{j=k+1}^{t} \frac{\partial s_j}{\partial s_{j-1}} \right)} \frac{\partial s_k}{\partial W}$$

M I C

Learning Long-term Dependencies

- Vanishing & Exploding gradients prevent vanilla RNNs from effectively learning patterns between inputs many time steps apart

Solutions:

1. Orthogonal weight initialization
2. Modify the activation function
3. Modify the way gradients are propagated

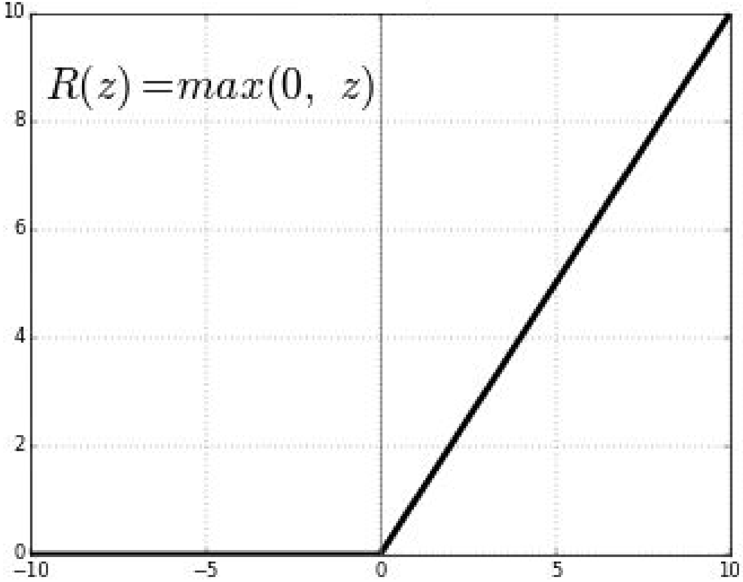
Orthogonal initialization

- In an orthogonal matrix, the rows and columns are orthonormal to each other.
  - Encourages weights to learn different input features
  - They are norm-preserving, i.e. $||Wx|| = ||x||$

Rectified Linear Units

ReLUs can help deal with the vanishing gradient problem

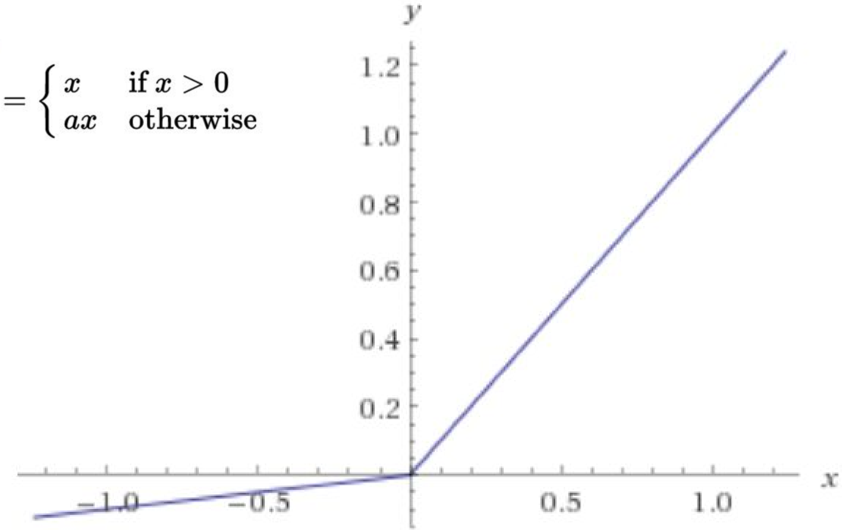
ReLU



$$R(z) = max(0, \; z)$$

Leaky ReLu

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ ax & \text{otherwise} \end{cases}$$



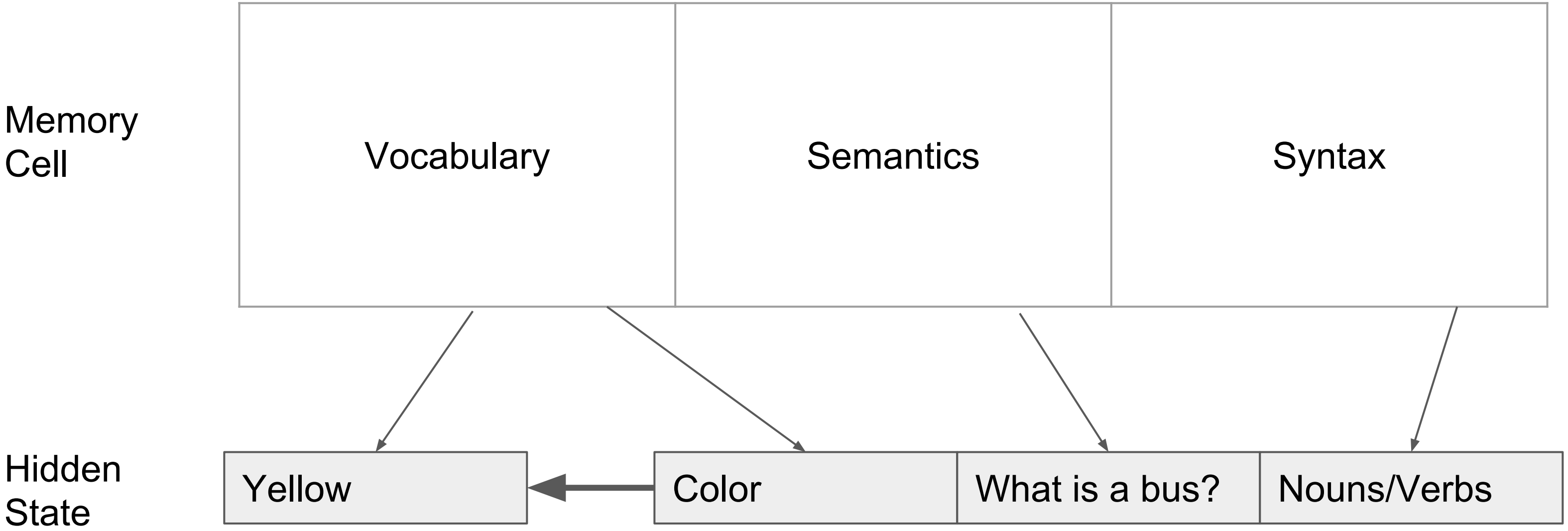
LSTM Network

- Long Short Term Memory
  - Extension of RNN cells to handle vanishing gradients
- Key Concepts
  - **Memory Cell** - 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

M I C

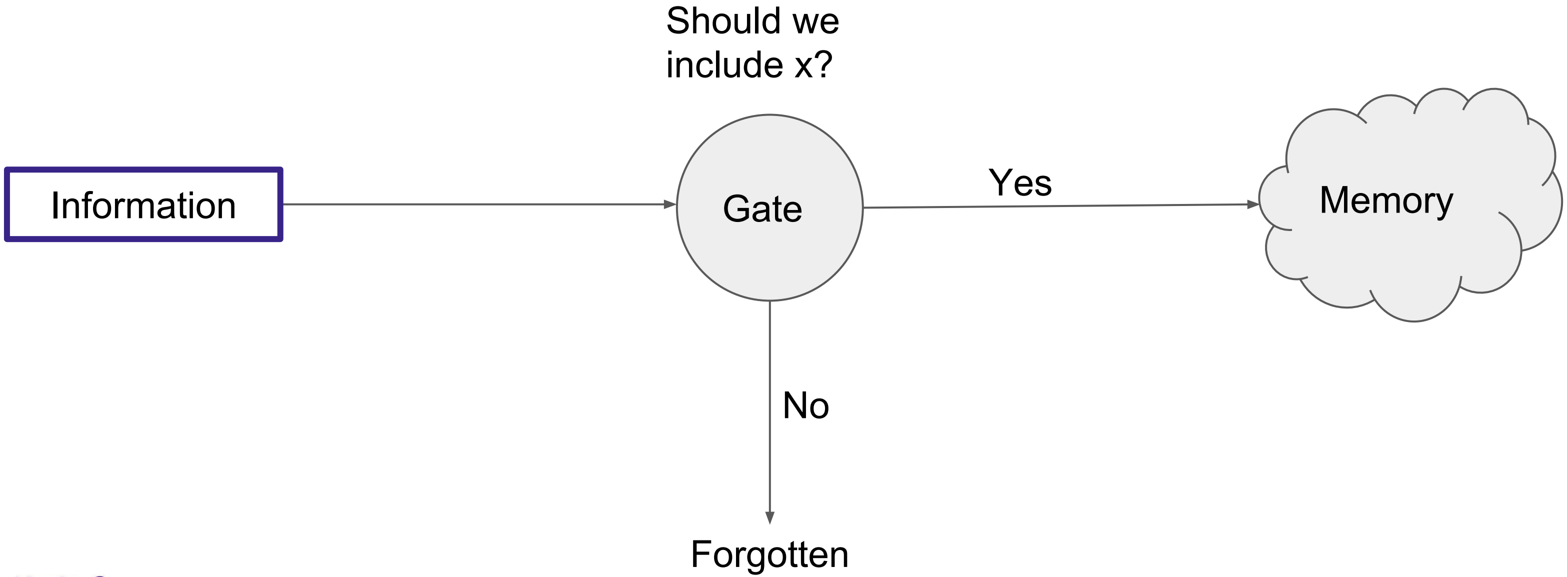
Memory Example

The color of the bus is _______.

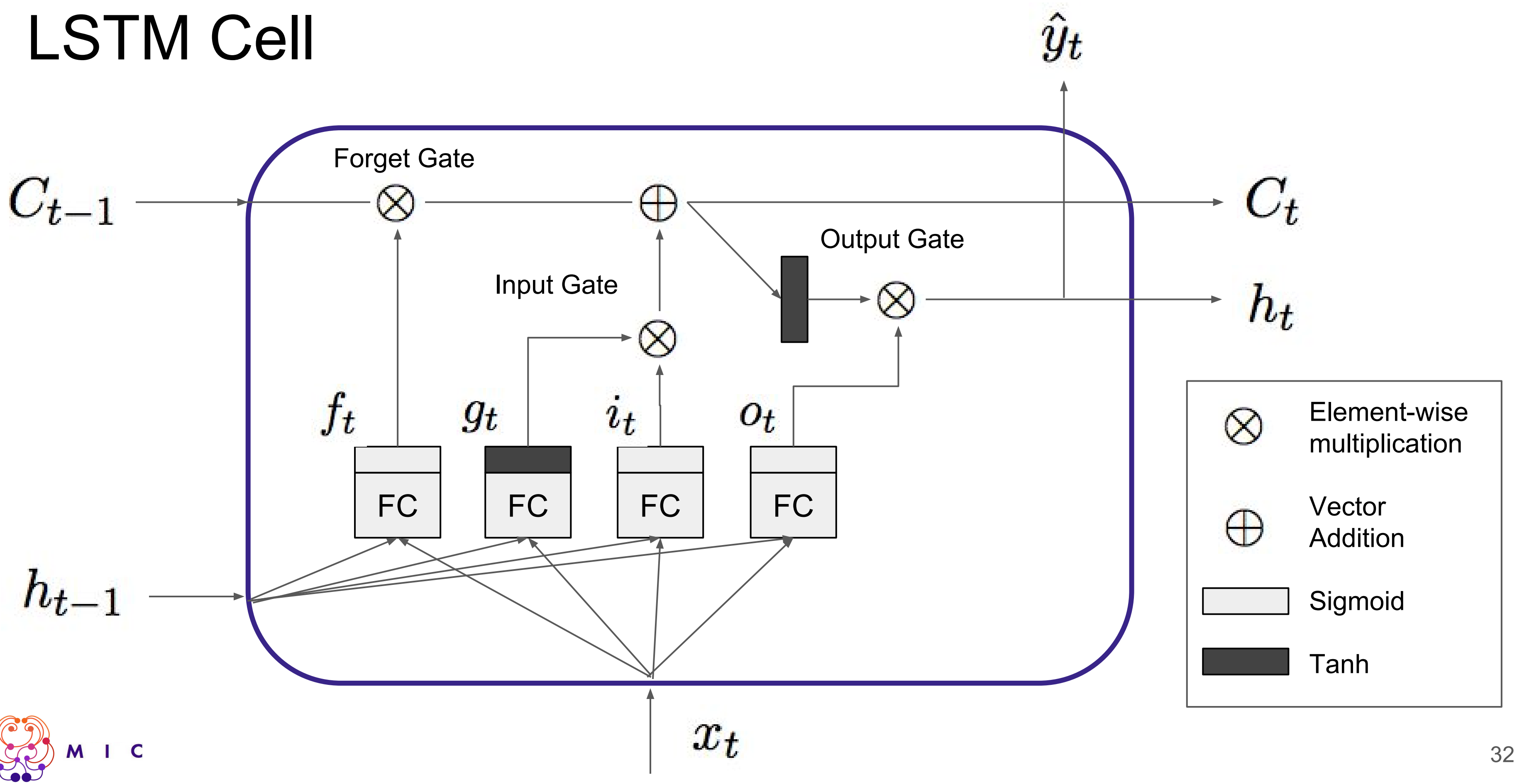


Gated Memory

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

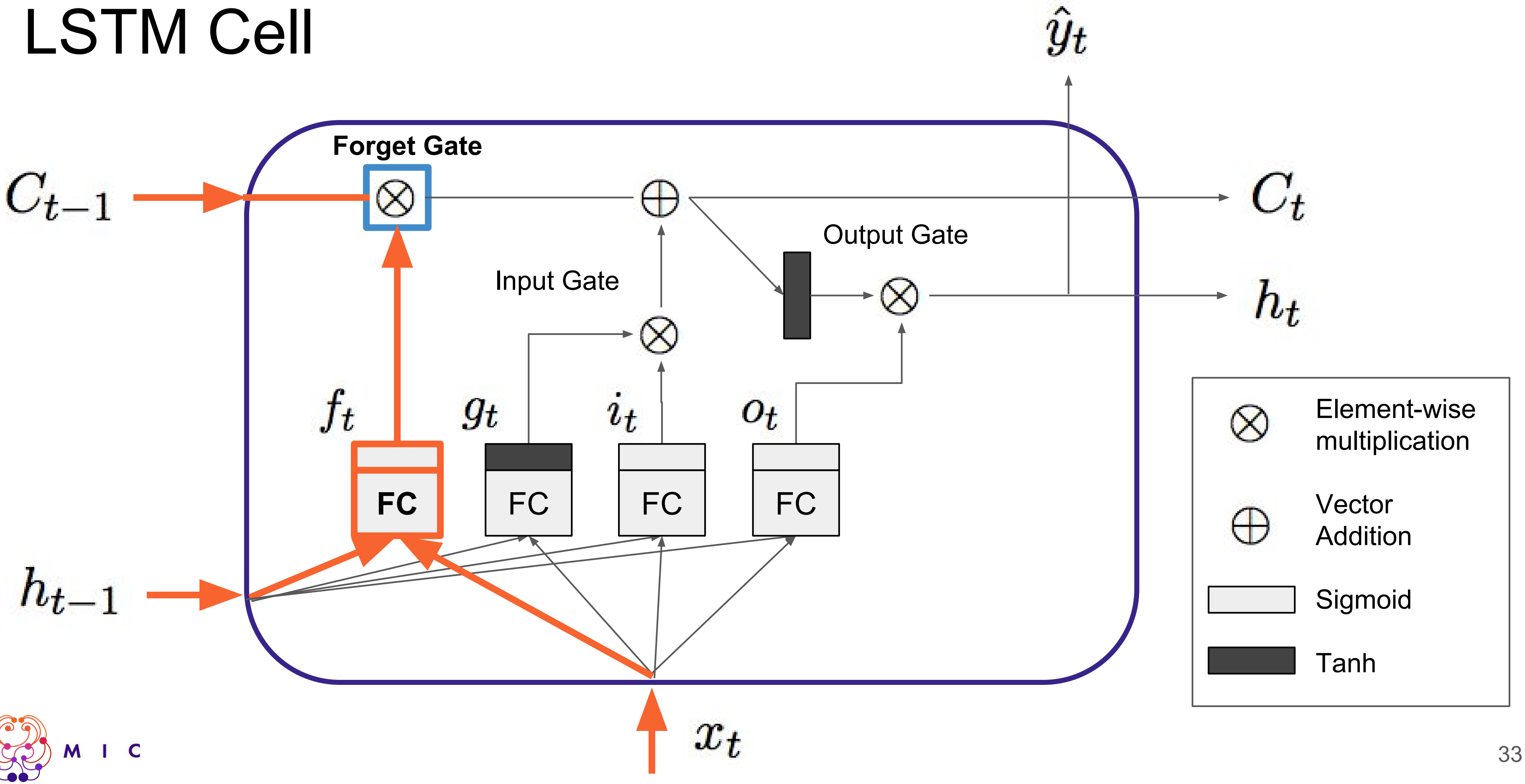


LSTM Cell



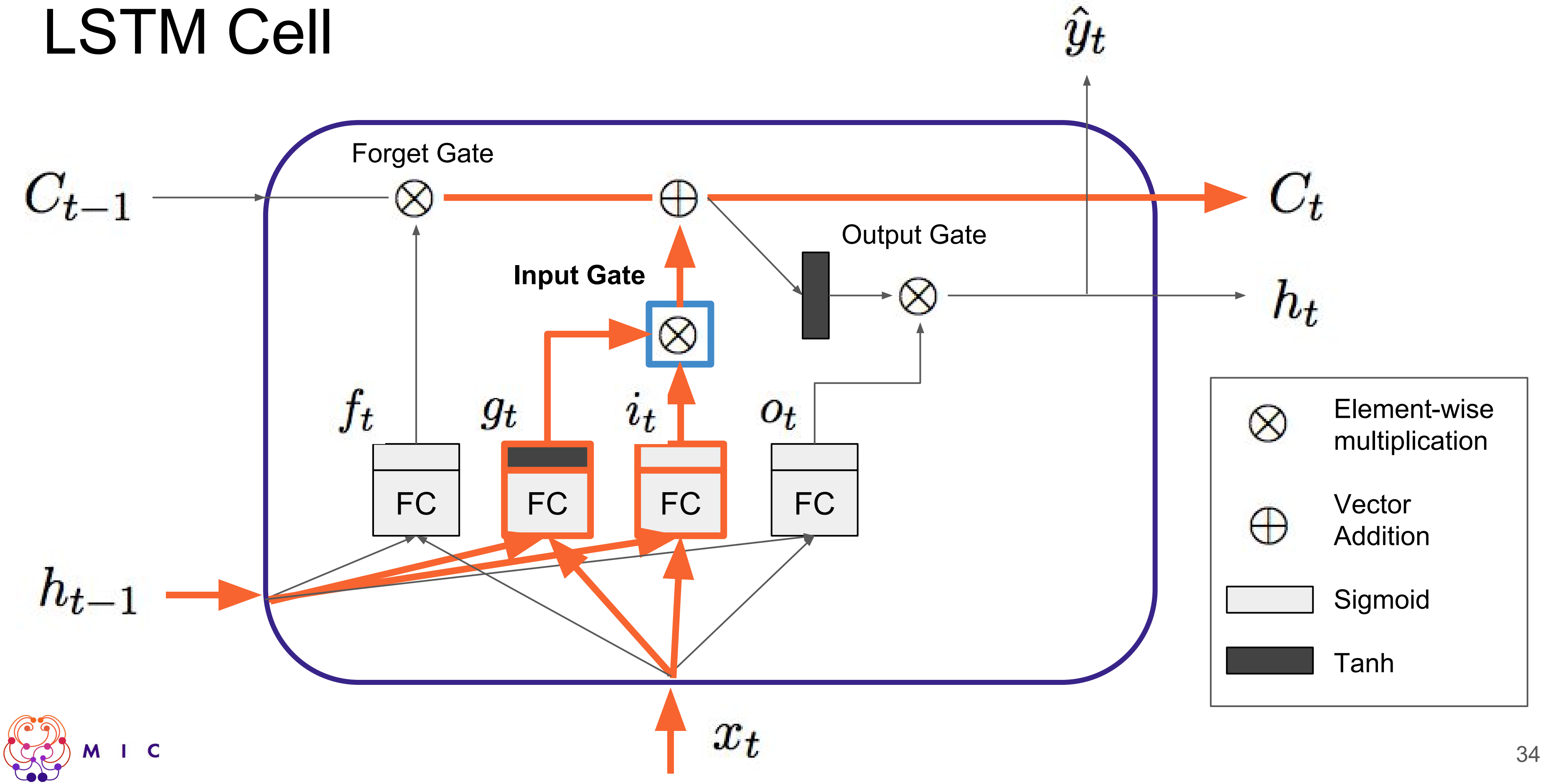
M I C

LSTM Cell



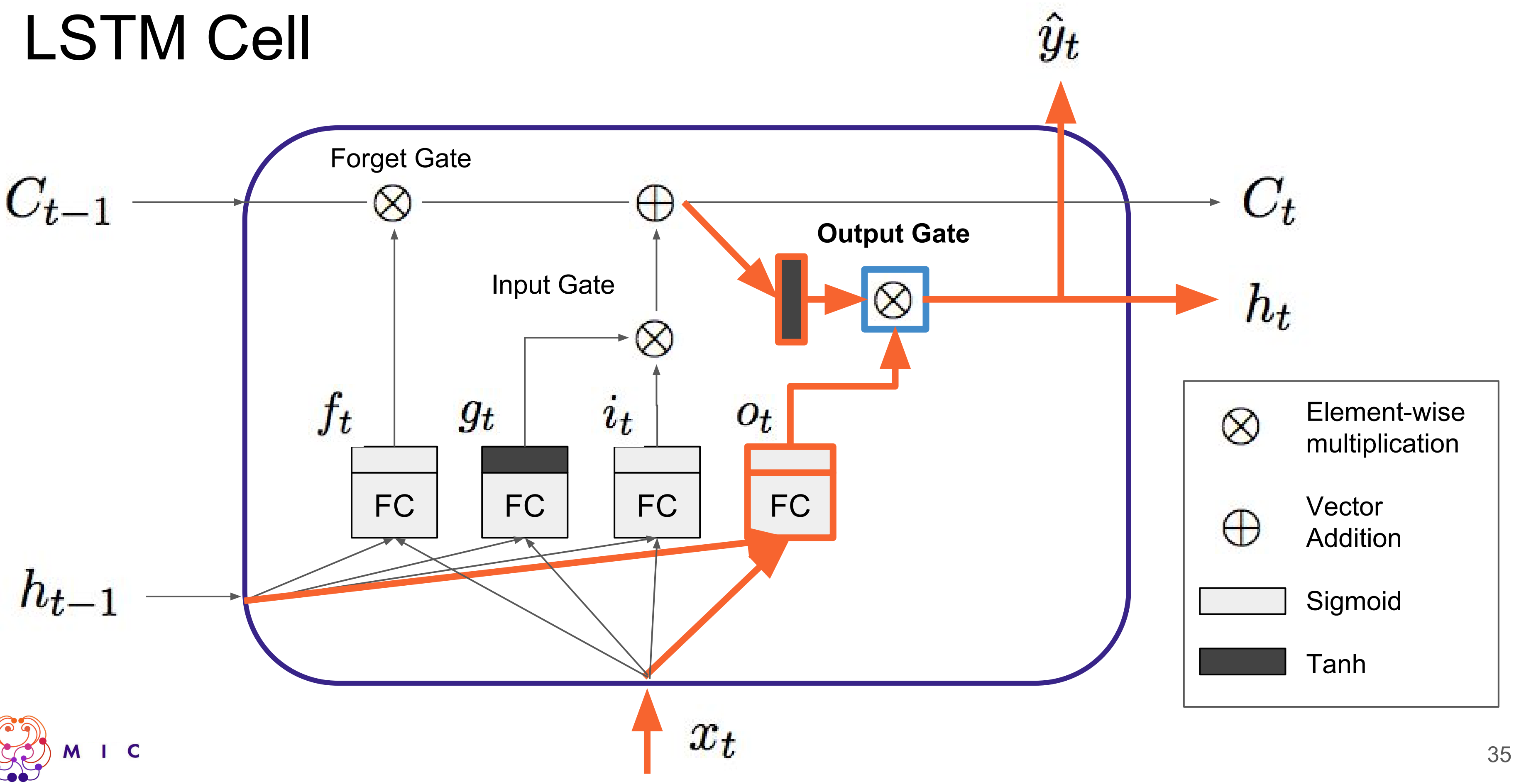
M I C

LSTM Cell



$\hat{y}_t$
Forget Gate
C_{t-1}
C_t
Output Gate
Input Gate
h_t
f_t
g_t
i_t
o_t
FC
FC
FC
FC
h_{t-1}
x_t
Element-wise multiplication
Vector Addition
Sigmoid
Tanh

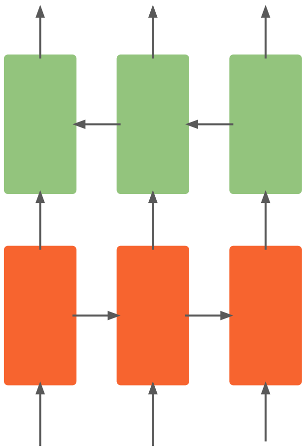
LSTM Cell



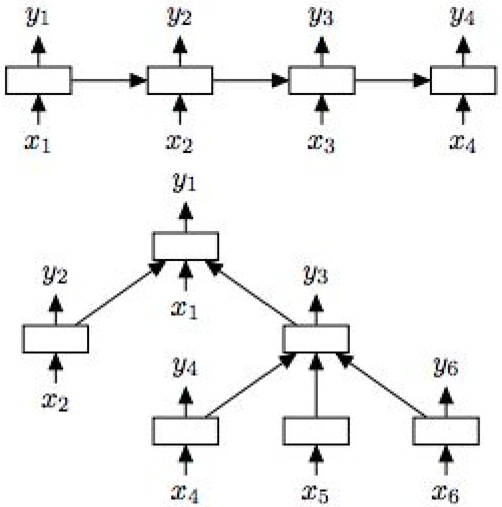
$\hat{y}_t$
Forget Gate
C_{t-1}
C_t
Output Gate
Input Gate
h_t
f_t
g_t
i_t
o_t
FC
FC
FC
FC
h_{t-1}
x_t
Element-wise multiplication
Vector Addition
Sigmoid
Tanh

Other LSTM Architectures

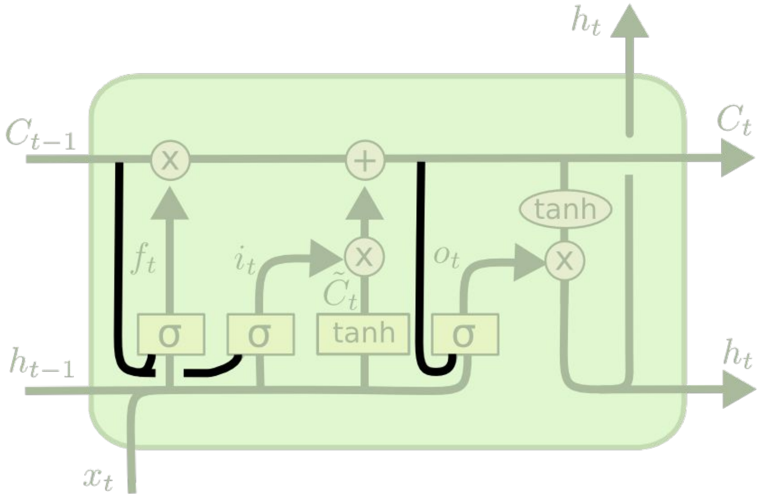
Birdirectional LSTM



Tree LSTM



Peephole Connections

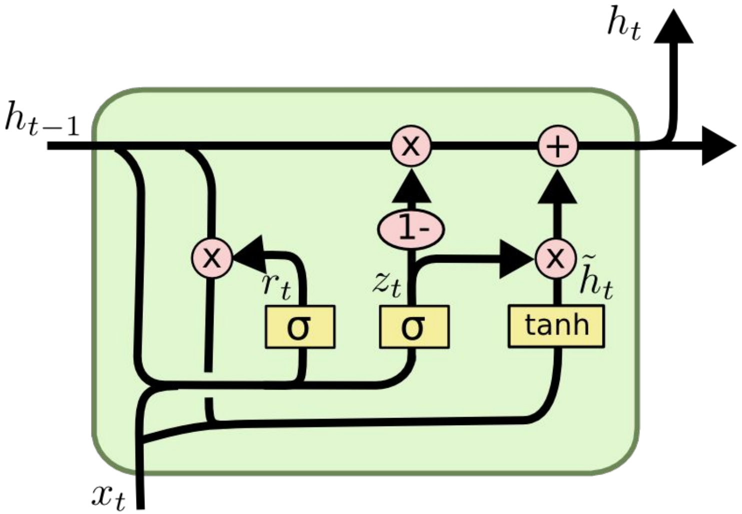


$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_f\right)$$

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

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

Gated Recurrent Units (GRUs)



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

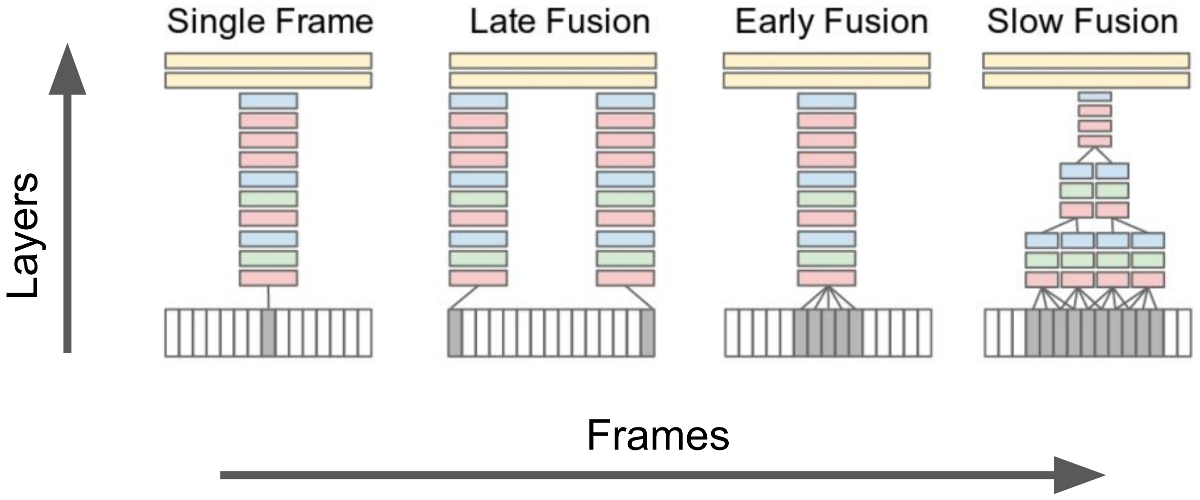
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

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

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

CNNs for Video Classification

- Large-scale Video Classification with Convolutional Neural Networks
  - Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, Li Fei-Fei
  - Modified CNN architecture for capturing temporal relationships



CNNs for Video Classification cont.

- Beyond Short Snippets: Deep Networks for Video Classification
    - Joe Yue-Hei Ng et al.
    - Comparison of performance of various deep networks on UCF-101 dataset.
        - 13,320 videos
        - 101 types of different actions

Method	3-fold Accuracy (%)
Improved Dense Trajectories (IDTF)s [23]	87.9
Slow Fusion CNN [14]	65.4
Single Frame CNN Model (Images) [19]	73.0
Single Frame CNN Model (Optical Flow) [19]	73.9
Two-Stream CNN (Optical Flow + Image Frames, Averaging) [19]	86.9
Two-Stream CNN (Optical Flow + Image Frames, SVM Fusion) [19]	88.0
Our Single Frame Model	73.3
Conv Pooling of Image Frames + Optical Flow (30 Frames)	87.6
Conv Pooling of Image Frames + Optical Flow (120 Frames)	**88.2**
LSTM with 30 Frame Unroll (Optical Flow + Image Frames)	**88.6**

M I C

Additional Resources

Christopher Olah (Colah) blog:

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

WildML newsletter/informational site:

http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

Sepp Hochreiter's original LSTM paper:

http://www.mitpressjournals.org/doi/pdfplus/10.1162/neco.1997.9.8.1735

Shoutout to our Sponsors

Boston University Computer Science

Boston University SPARK!

Boston University Software Application and Innovation Lab

M I C

Upcoming Events

MIC Paper signup: https://goo.gl/iAm6TL
BUMIC Projects signup: https://goo.gl/GmP9oK

BU MIC DRL Series:

Next semester meeting:
Deep Reinforcement Learning Series
Meeting this Thursday 11.9.17 in Hariri

BU MIC reading group:

Paper: Large Scale Distributed Deep Networks
Location: Fishbowl Conference Room
Date: 11.13.17 Time: 5 PM

Next workshop:

Topic: Deep Reinforcement Learning
Location: BU Hariri Seminar Room
Date: 11.14.17 Time: 7 PM

M I C