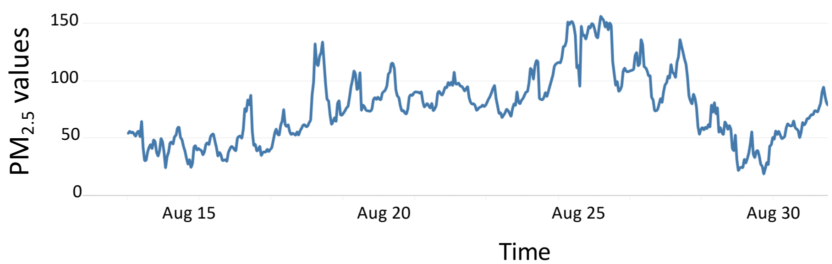
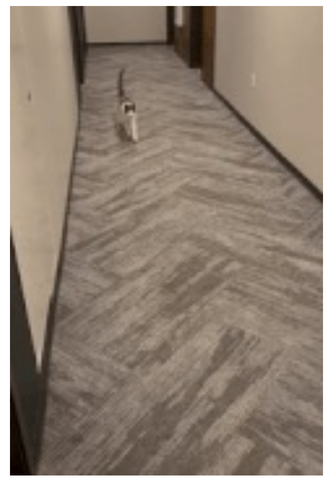
# Recurrent Neural Networks I

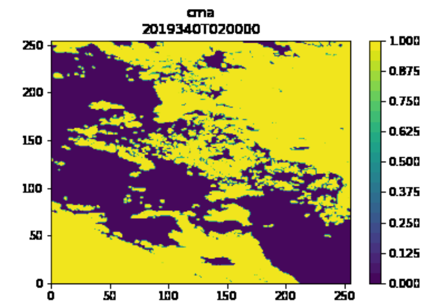
Yao-Yi Chiang
Computer Science and Engineering
University of Minnesota
yaoyi@umn.edu

# Sequences

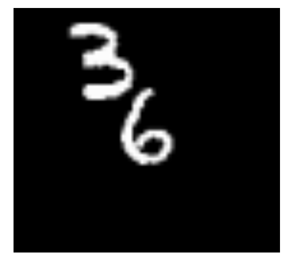
- Sequences are everywhere
  - Natural language
    - "This morning I took my cat for a walk"
  - Audio and video
  - Sensor observations (e.g., air quality, traffic, noise)






Cloud Mask from Weather4cast



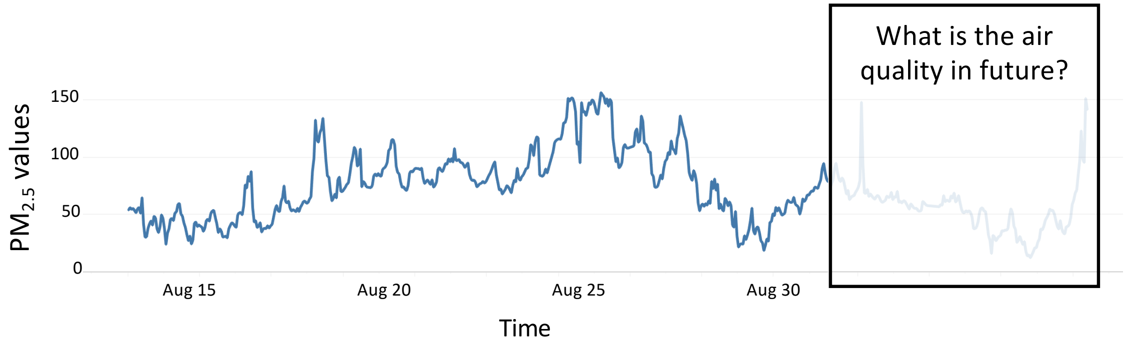
Moving MNIST

# Sequence Modeling

- Sequence modeling is the task of predicting what comes next
  - E.g., “This morning I took my cat for a walk ”

    given previous words    predict the next word

  - E.g., given historical air quality, forecast air quality in next couple of hours



# A Sequence Modeling Example
# Predict the Next Word

- Idea #1: Use a fixed window

"This morning I took my cat for a walk"

given previous two words — predict the next word

# A Sequence Modeling Example
# Predict the Next Word

- Idea #1: Use a fixed window

"This morning I took my cat for a walk."

given previous two words — predict the next word

- Limitation: Cannot model long-term dependencies
  - E.g., "France is where I grew up, but I now live in Boston. I speak fluent ___."

# A Sequence Modeling Example
# Predict the Next Word

- Idea #1: Use a fixed window

“This morning I took my cat for a walk ”

given previous two words — predict the next word

- Limitation: Cannot model long-term dependencies
  - E.g., “France is where I grew up, but I now live in Boston. I speak fluent ___.”

- We need information from **the distant past** to accurately predict the correct word

# A Sequence Modeling Example
# Predict the Next Word

- Idea #2: Use entire sequence as set of counts

"This morning I took my cat for a walk"

predict the
next word

# A Sequence Modeling Example
# Predict the Next Word

- Idea #2: Use entire sequence as set of counts

"This morning I took my cat for a walk"

predict the next word

- Bag-of-words model
  - Define a vocabulary and initialize a zero vector where each element represents for each word
  - Compute word frequency and update the correspond position in the vector

    [0 1 0 0 1 0 1 … … 0 0 1 1 0 0 0 1 0 ]

  - Use the vector for prediction

Here 1 is the count for the work "a"

# A Sequence Modeling Example
# Predict the Next Word

- Idea #2: Use entire sequence as set of counts

"This morning I took my cat for a walk"

predict the next word

- Limitation: Counts don't preserve order
  - "The food was good, not bad at all." VS. "The food was bad, not good at all."

- We need to preserve the information about **order**

# A Sequence Modeling Example
# Predict the Next Word

- Idea #3: Use a big fixed window

"This morning I took my cat for a walk"

given these words

predict the next word

- One-hot encoding

[1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 … …]

this morning I

- Use the one-hot encoding vector for prediction

# A Sequence Modeling Example
# Predict the Next Word

- Idea #3: Use a big fixed window

“This morning I took my cat for a walk ”

given these words

predict the next word

- Limitation: Each of these inputs has a separate parameter
  - “I took my cat this morning for a walk”

[0 0 0 1 0 0 0 … … 1 0 0 0 0 0 0 0 0 0 0 0 1 … …]

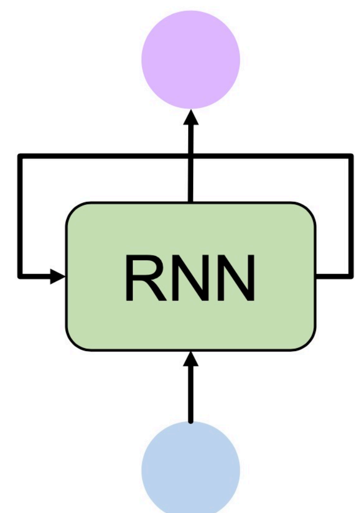
I this morning

- Things we learn about the sequence should be applicable when they appear elsewhere in the sequence

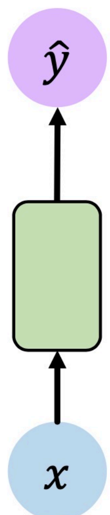
# Sequence Modeling

- To model sequences, we need to:
  - Handle **variable-length** sequences
  - Track **long-term** dependencies
  - Maintain information about **order**
  - **Share parameters** across the sequence

- Solution:
  - Recurrent Neural Networks (RNNs)

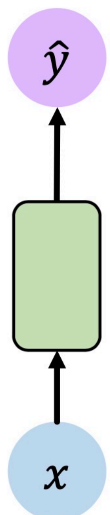


# Standard Feed-Forward Neural Network

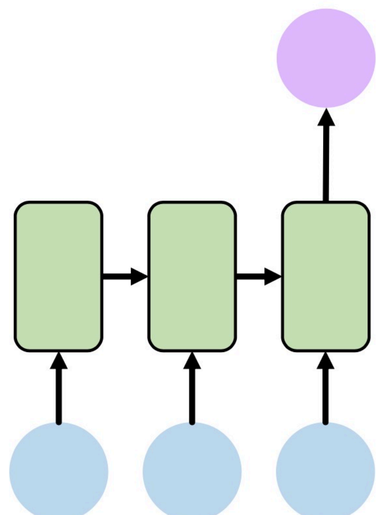



One to One
"Vanilla" neural network

# Recurrent Neural Networks

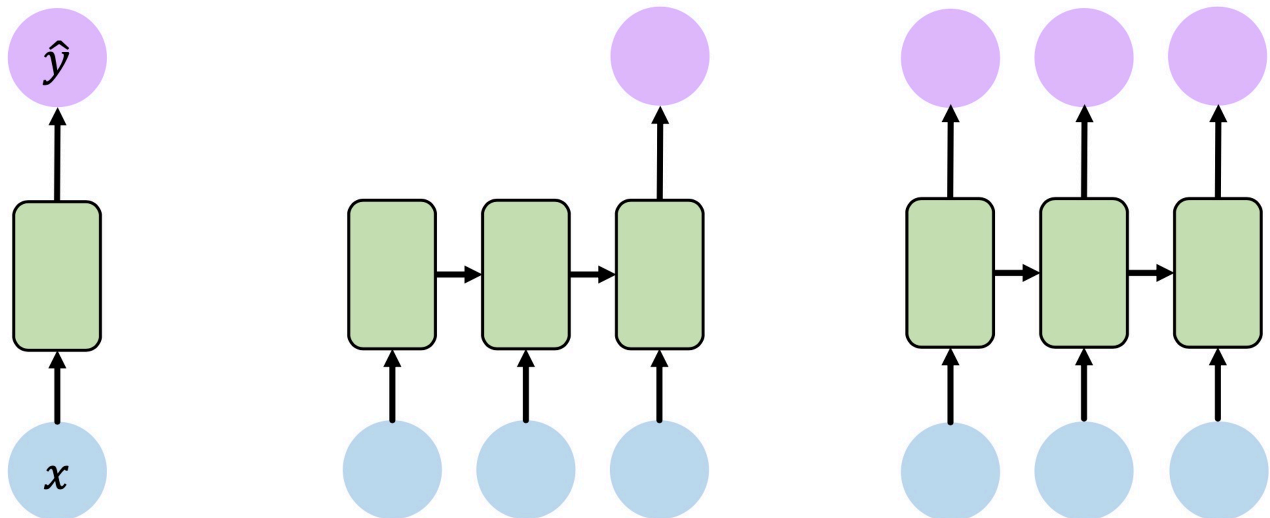



One to One
“Vanilla” neural network



Many to One
Sentiment Classification

# Recurrent Neural Networks



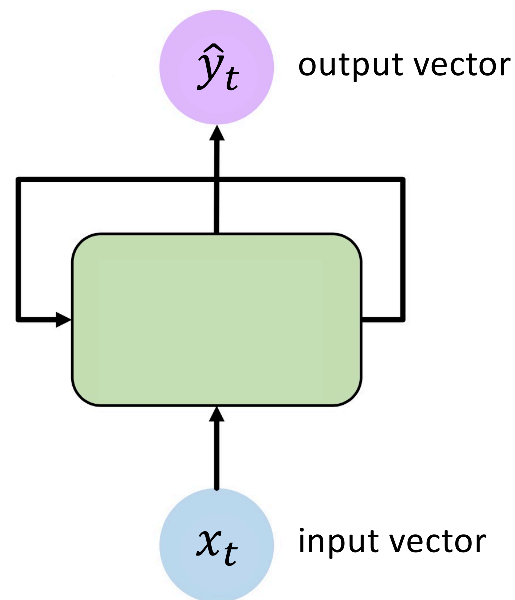

One to One
“Vanilla” neural network

Many to One
Sentiment Classification

Many to Many
Music Generation

... and many other architectures and applications

# A Recurrent Neural Network (RNN)



# A Recurrent Neural Network (RNN)

- Apply a recurrence relation at every time step to process a sequence:

cell state

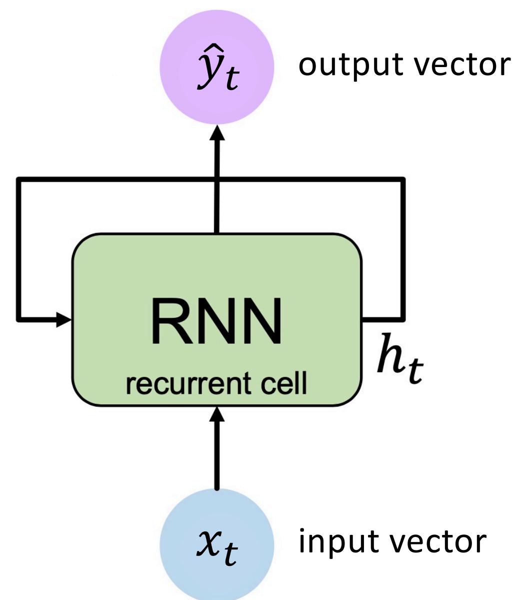
old state

$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

a function parameterized by $\boldsymbol{W}$

current input

Note: the same function and set of parameters are used at every time step




https://r2rt.com/non-zero-initial-states-for-recurrent-neural-networks.html

# RNN: State Update and Output

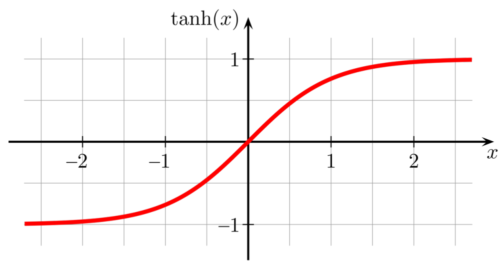
$$\overbrace{\boxed{h_t}}^{\text{cell state}} = \underbrace{\boxed{f_{\boldsymbol{W}}}}_{\text{a function parameterized by } \boldsymbol{W}}(\overbrace{\boxed{h_{t-1}}}^{\text{old state}}, \underbrace{\boxed{x_t}}_{\text{current input}})$$

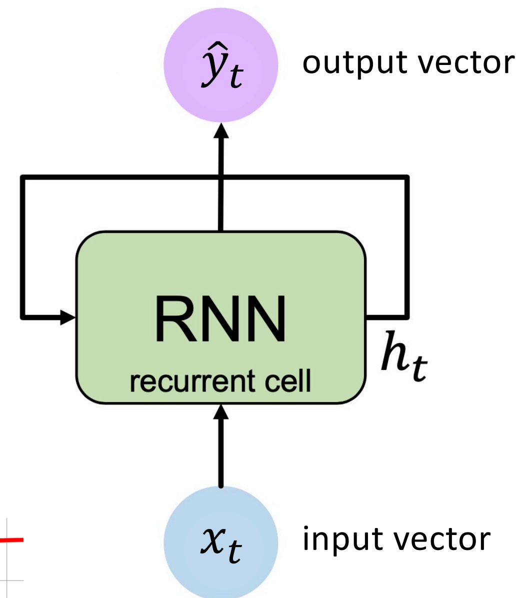
- Update hidden state, $f_W$

$$h_t = \tanh(\boldsymbol{W_{hh}} h_{t-1} + \boldsymbol{W_{xh}} x_t)$$

- Compute output vector

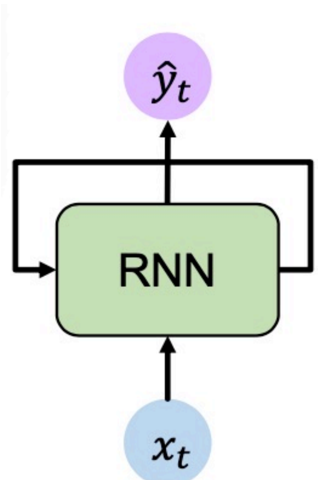
$$\hat{y}_t = \boldsymbol{W_{hy}} h_t$$



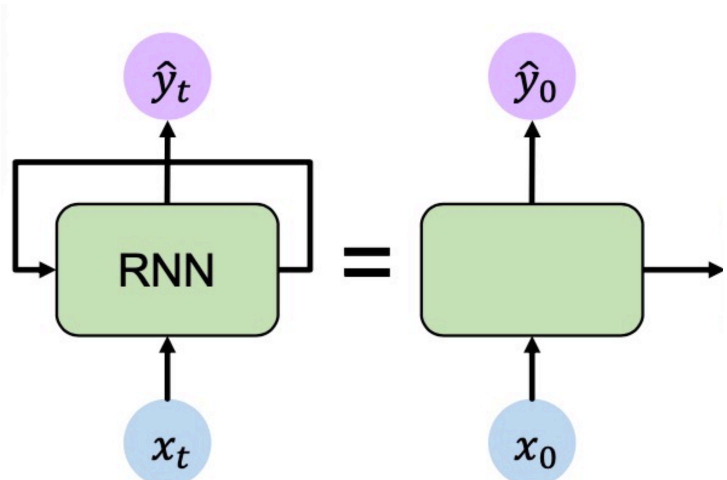
# RNN: Computational Graph Across Time

• Represent as computational graph unrolled across time



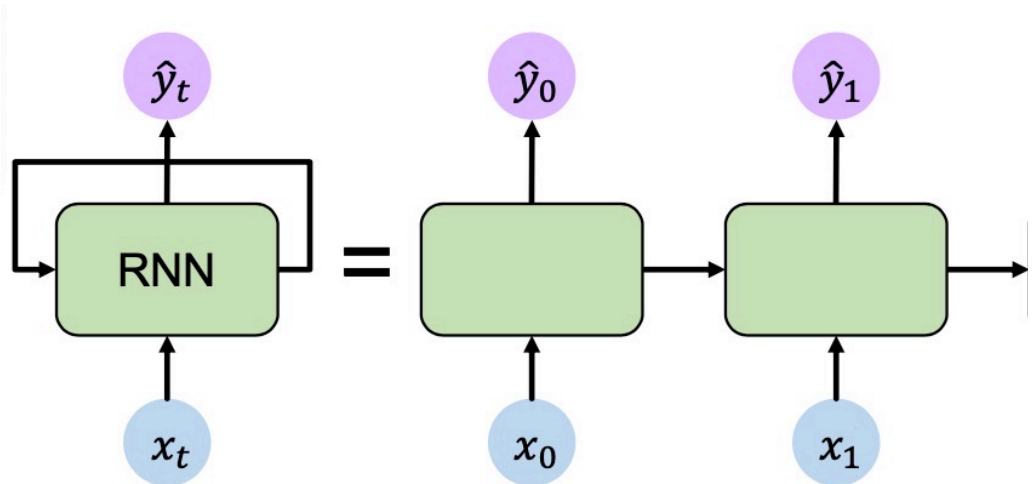
# RNN: Computational Graph Across Time

- Represent as computational graph unrolled across time



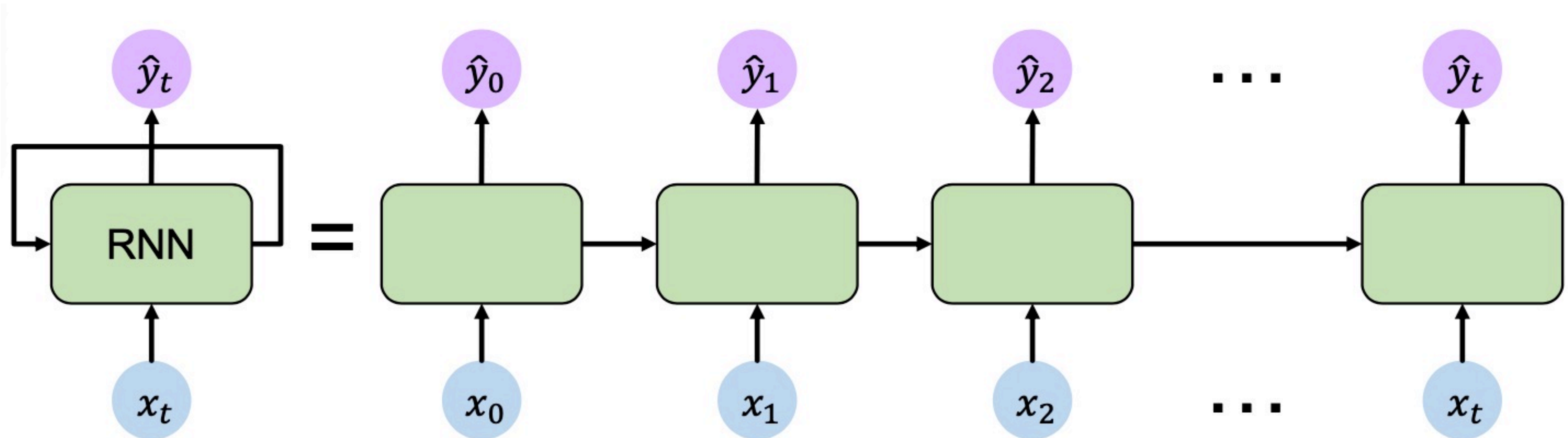
# RNN: Computational Graph Across Time

- Represent as computational graph unrolled across time



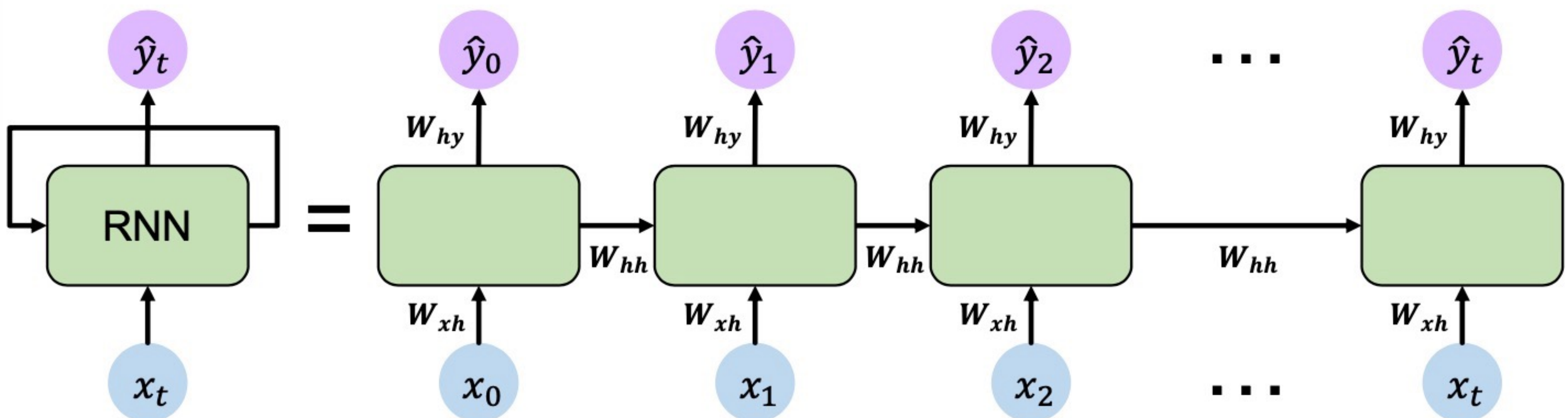
# RNN: Computational Graph Across Time

- Represent as computational graph unrolled across time



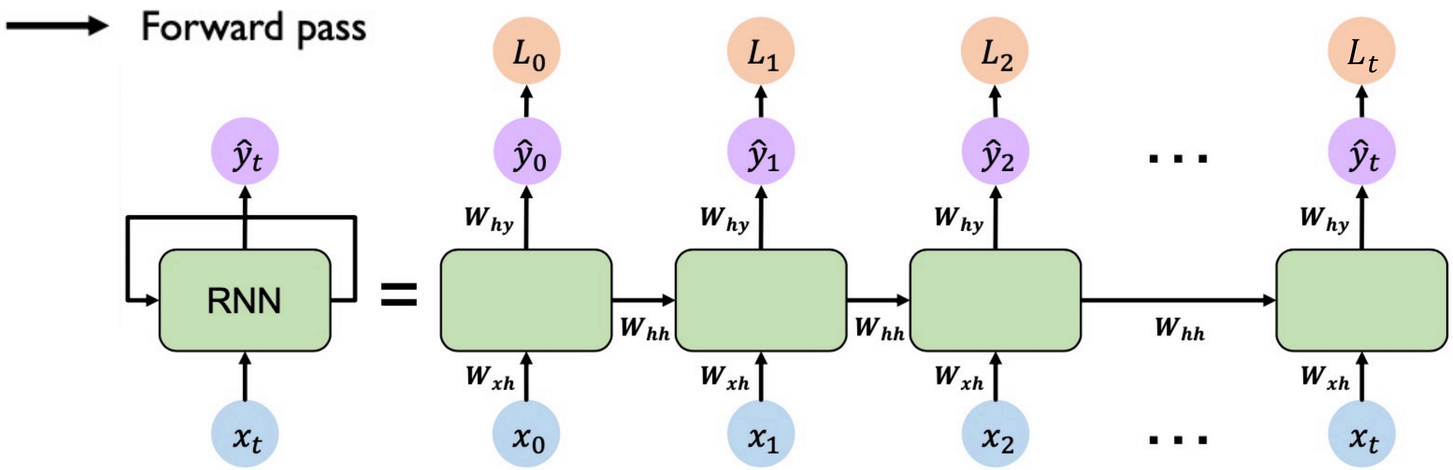
# RNN: Computational Graph Across Time

- Re-use the same weight matrices at every time step



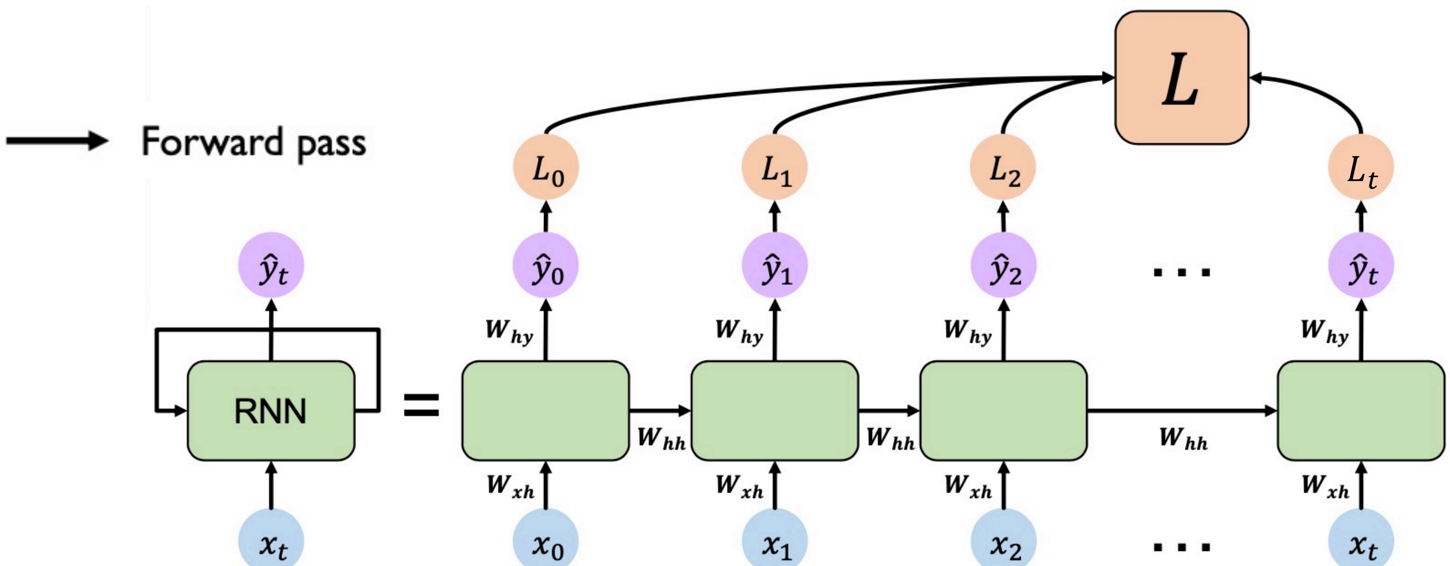
# RNN: Computational Graph Across Time

- Compute the loss $L_t$ by comparing $\hat{y}_t$ and $y_t$ ($y_t$ is ground truth)
  - E.g., $L_t = (\hat{y}_t - y_t)^2$



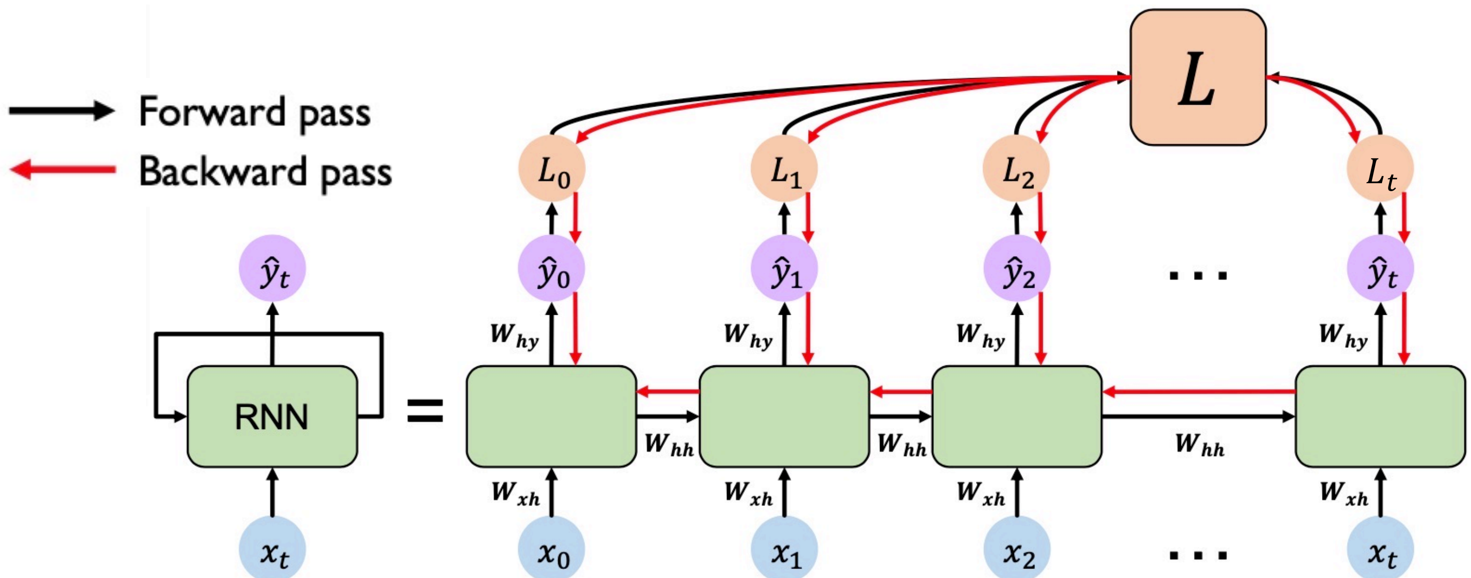
# RNN: Computational Graph Across Time

- Total loss $L = \sum_{t=1}^{T} L_t$

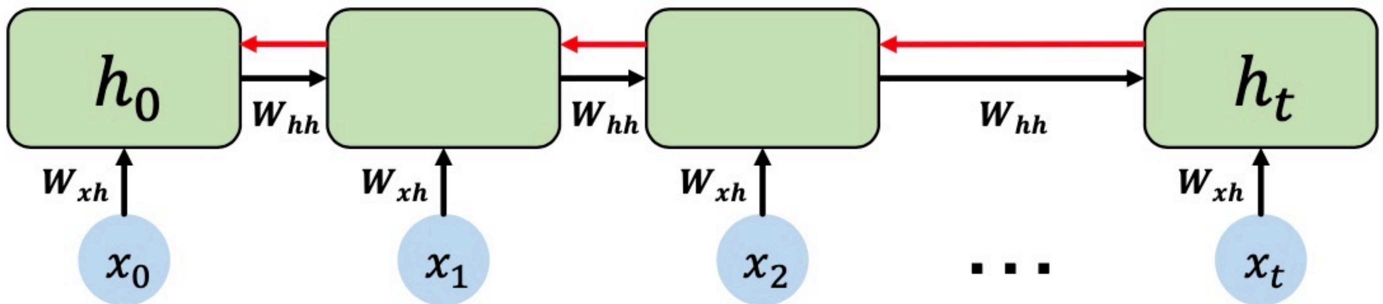


# RNN: Backpropagation Through Time

- For backpropagation, we need to compute the gradients w.r.t. $W_{hy}$, $W_{hh}$, $W_{xh}$



# RNN: Backpropagation Through Time




Computing the gradient involves **many multiplications** (and repeated $f'$)

- When $w_h$ changes (in a small amount), how much (and direction) would $L$ change?

For example, $\frac{\partial L}{\partial w_h} = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l(y_t, \hat{y}_t)}{\partial w_{hh}}$

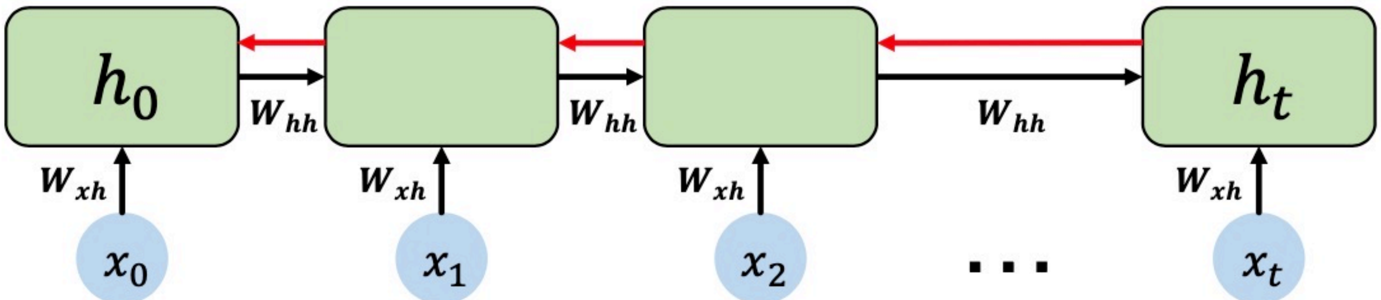
$$h_t = \tanh(\boldsymbol{W}_{\boldsymbol{xh}} x_t + \boldsymbol{W}_{\boldsymbol{hh}} h_{t-1})$$

$$= \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l(y_t, \hat{y}_t)}{\partial \hat{y}_t} \frac{\partial g(h_t, w_{hy})}{\partial h_t} \boxed{\frac{\partial h_t}{\partial w_{hh}}}.$$

$$\frac{\partial h_t}{\partial w_{hh}} = \frac{\partial f(x_t, h_{t-1}, w_{hh})}{\partial w_{hh}} = \frac{\partial f(x_t, h_{t-1}, w_{hh})}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial w_{hh}}.$$

https://d2l.ai/chapter_recurrent-neural-networks/bptt.html

# Gradient Flow: Exploding Gradients




Case 1: Many values are > 1

**Exploding gradients**

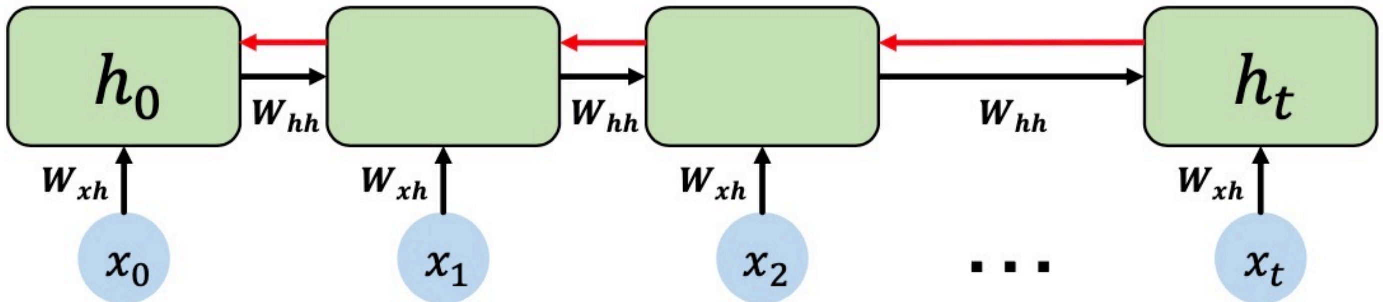
Trick : Gradient clipping to scale big gradients

Case 2: Many values are < 1

**Vanishing gradients**

Trick 1: Activation functions
Trick 2: Weight initialization
Trick 3: Network architecture

# Gradient Flow: Vanishing Gradients




Case 1: Many values are > 1

**Exploding gradients**

Trick 1: Gradient clipping to scale big gradients

Case 2: Many values are < 1

**Vanishing gradients**

Trick 1: Activation functions

Trick 2: Network architecture

# Vanishing Gradients

## What causes vanishing gradients?

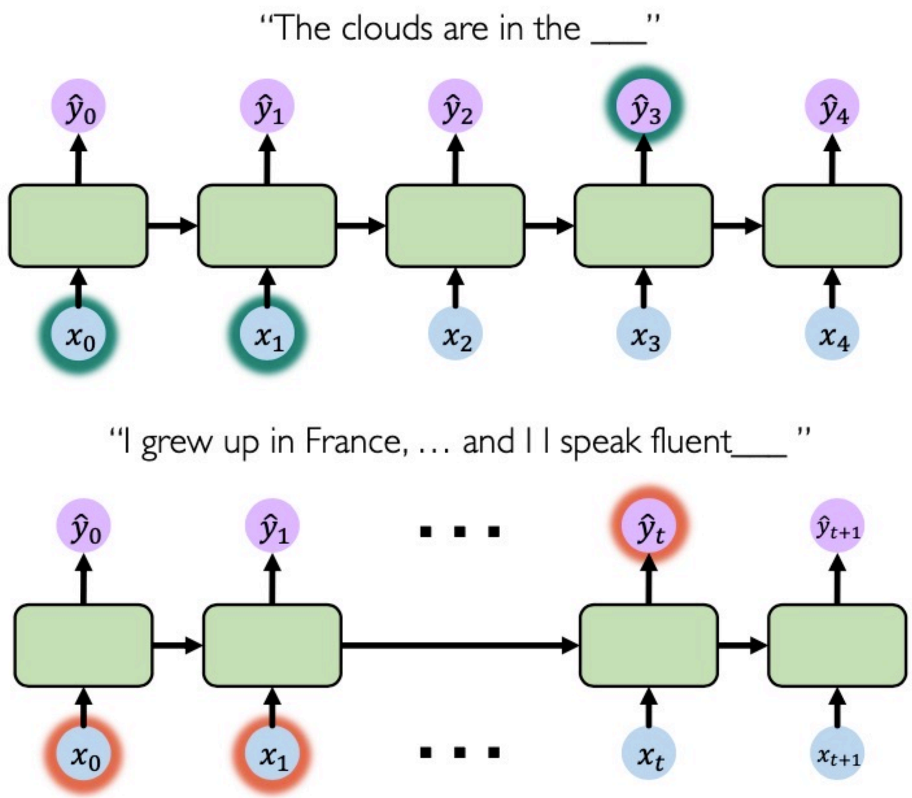
Multiply many small numbers together

↓

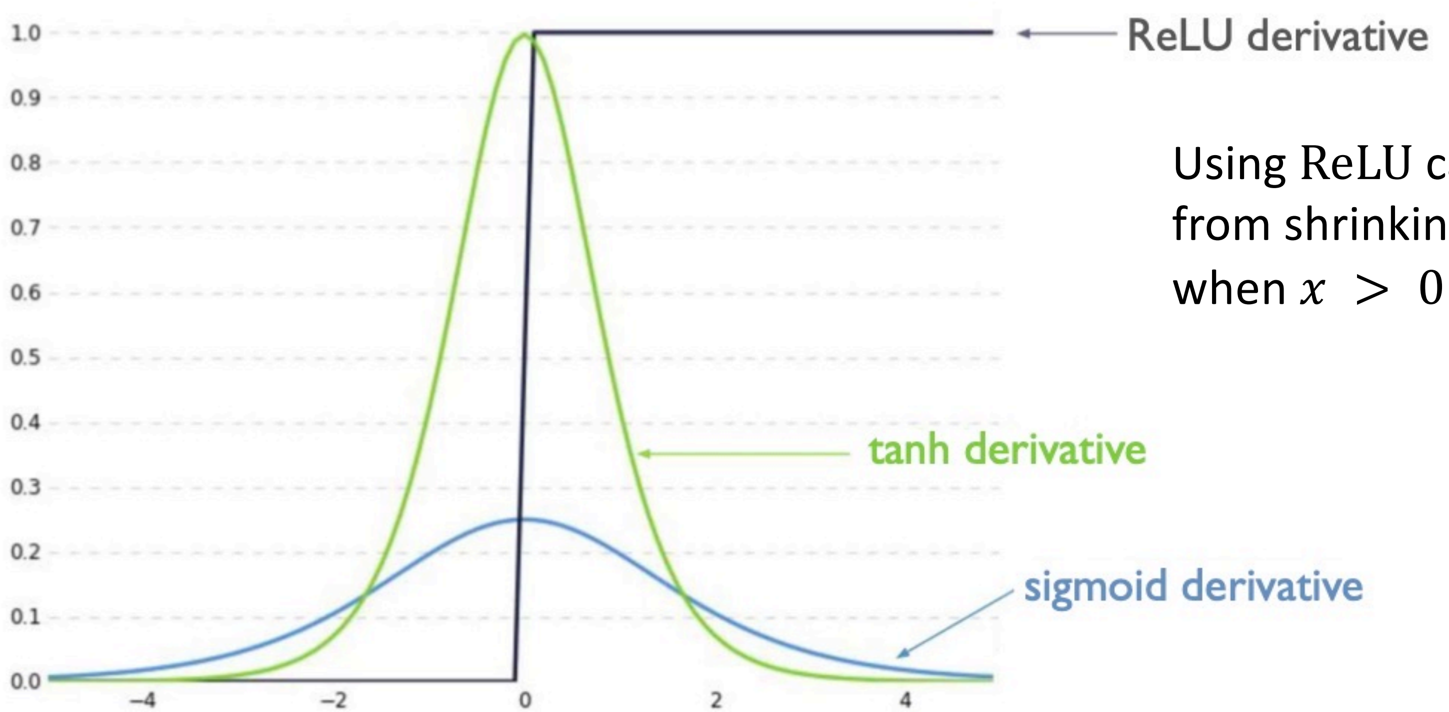
Further back time steps would have smaller and smaller gradients

↓

Fail to capture long-term dependencies

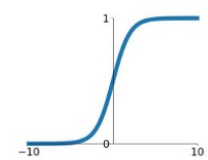


# Trick 1: Activation Functions

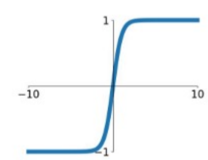



Using $\mathrm{ReLU}$ can prevent $f'$ from shrinking the gradients when $x > 0$

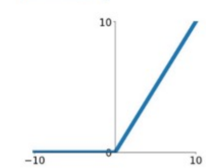
**Sigmoid**
$\sigma(x) = \frac{1}{1+e^{-x}}$



**tanh**
$\tanh(x)$

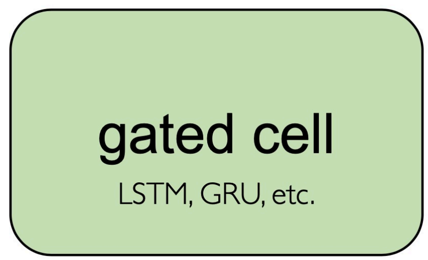


**ReLU**
$\max(0, x)$



# Trick 2: Network Architecture – Gated Cells

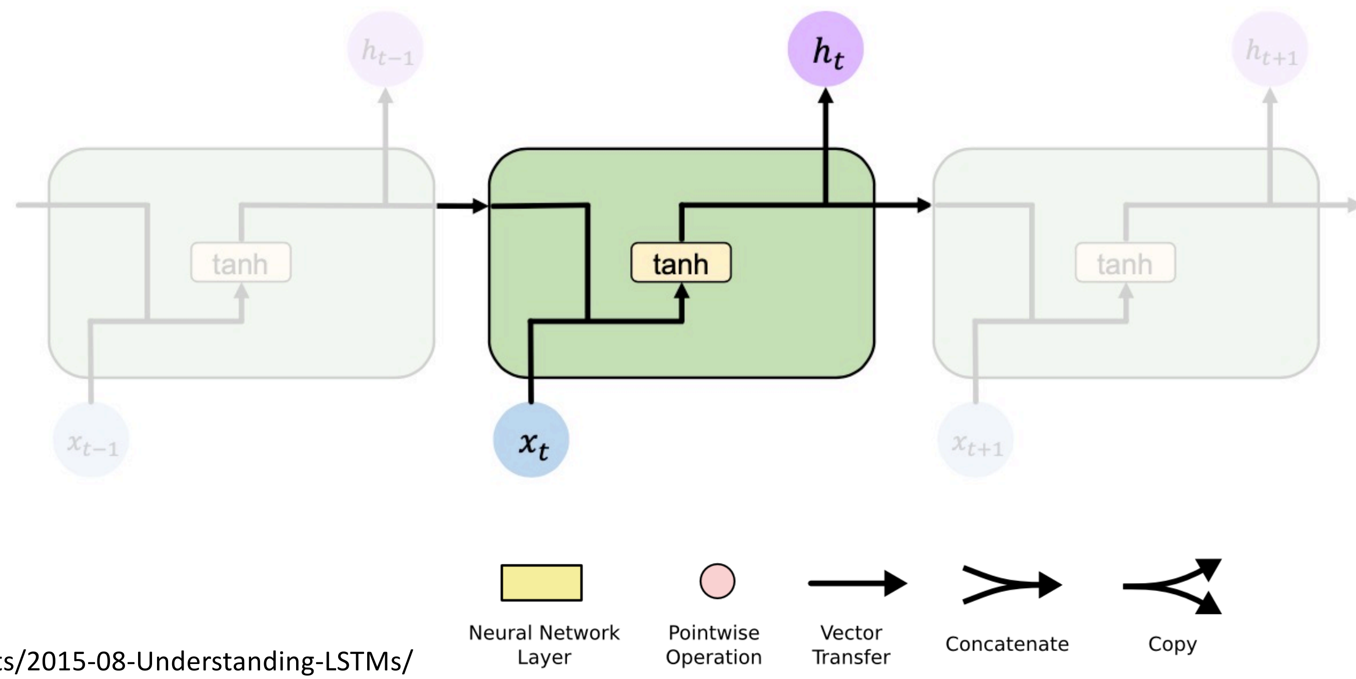
- Use a more **complex recurrent unit with gates** to control what information is passed through




- Long Short-Term Memory (LSTM) networks rely on gated cells to track information throughout many time steps.

# Standard RNNs

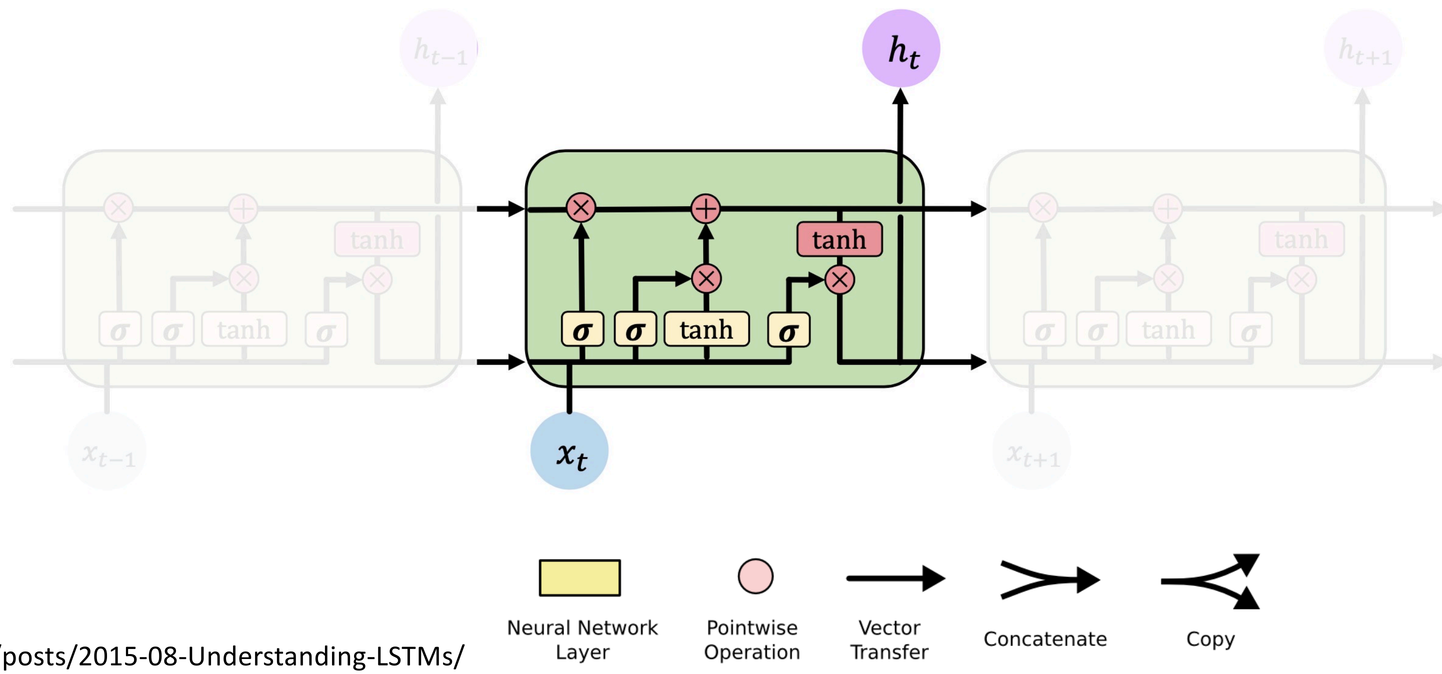
- In a standard RNN, recurrent modules contain **simple computation**




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

# Long Short-Term Memory (LSTM)

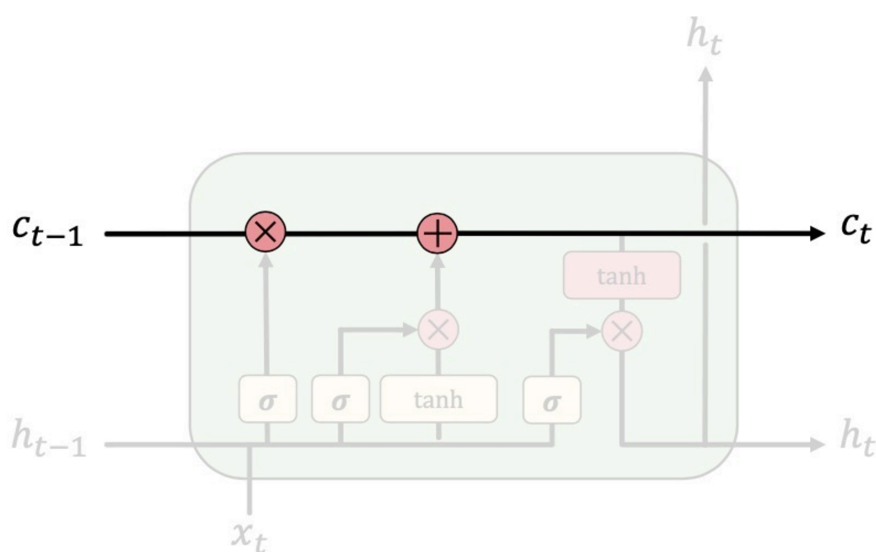
- In an LSTM network, recurrent modules contain **gated cells** that control the information flow [Hochreiter et al., 1997]




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

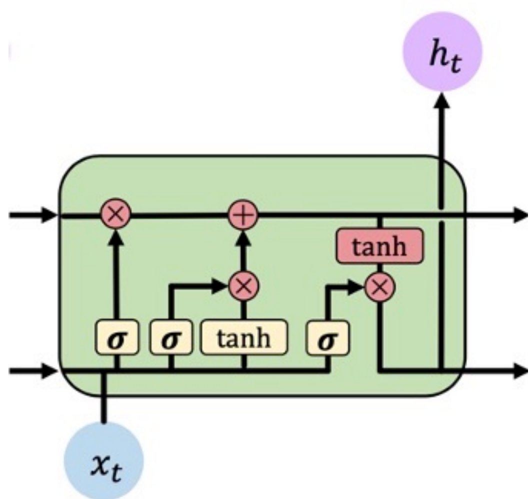
# Long Short-Term Memory (LSTM)

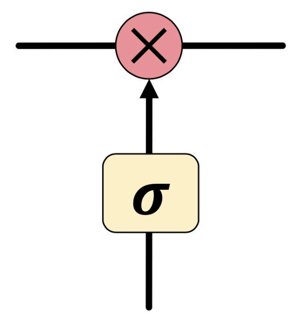
- Besides hidden state $h_t$ (same as RNN), LSTM maintains a **cell state $\boldsymbol{C_t}$** where it's easy for information to flow



# Long Short-Term Memory (LSTM)

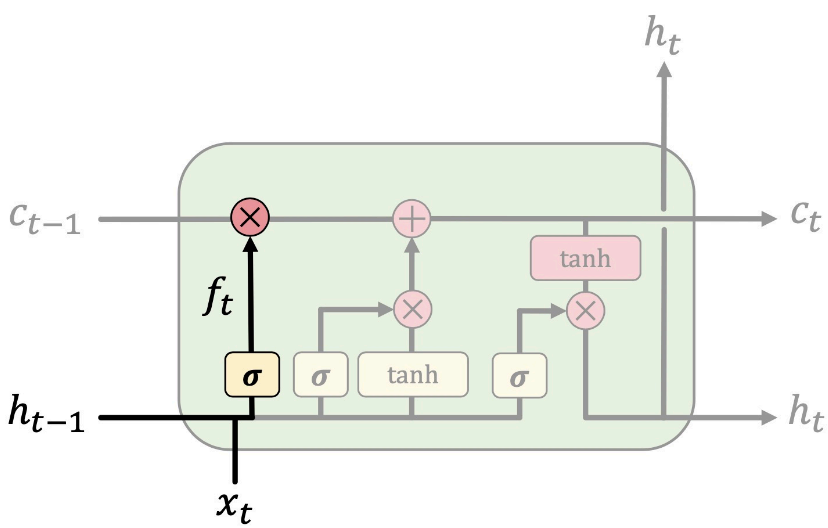
- Information is **added** or **removed** to cell state through structures called **gates**




Gates optionally let information through, via a sigmoid layer and pointwise multiplication

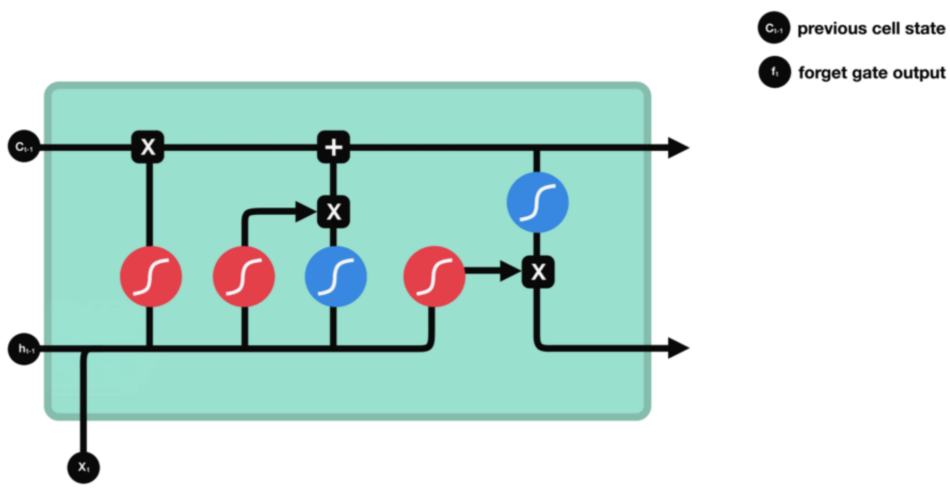
# LSTM: Forget Irrelevant Information




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

- Concatenate previous hidden state and current input
- When $\sigma$ outputs 0, the network will “completely forget” the information from $c_{t-1}$
- When $\sigma$ outputs 1, “completely keep”

# LSTM: Forget Irrelevant Information

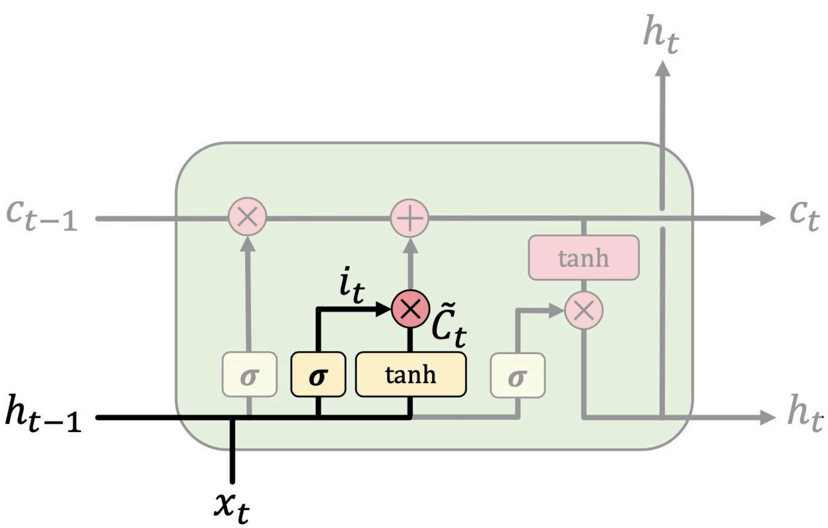



$$f_t = \sigma\left(W_f\left[h_{t-1}, x_t\right] + b_f\right)$$

- Concatenate previous hidden state and current input
- When $\sigma$ outputs 0, the network will "completely forget" the information from $c_{t-1}$
- When $\sigma$ outputs 1, "completely keep"

https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

# LSTM: Add New Information

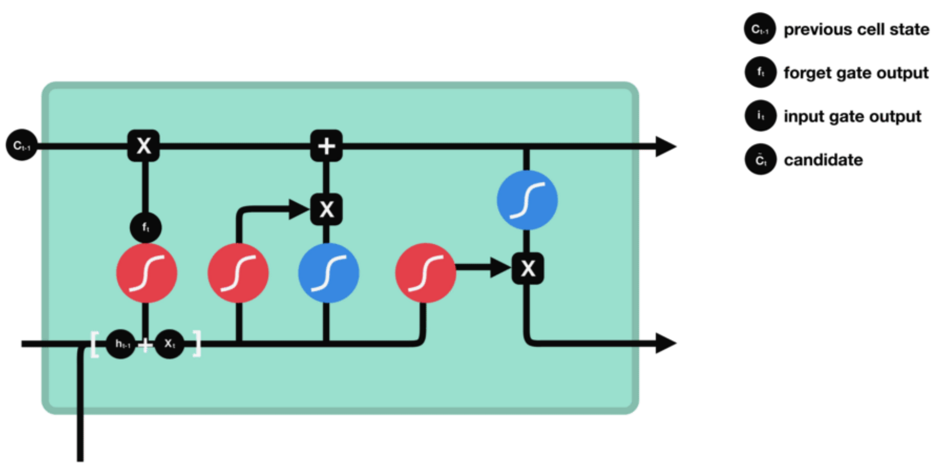



$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

$$\tilde{c}_t = \tanh(W_c[h_{t-1}, x_t] + b_c)$$

- $\sigma$ decides what values to update
- tanh generates "candidate values" that could be added to cell state

# LSTM: Add New Information



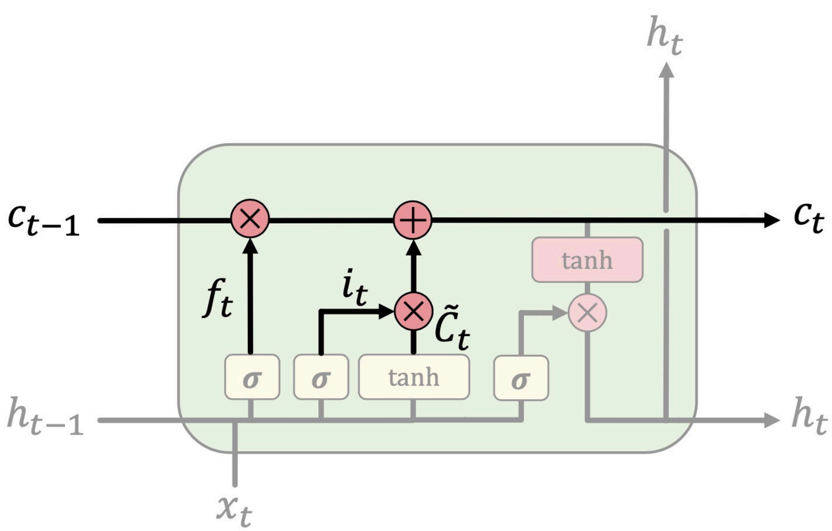

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

$$\tilde{c}_t = \tanh(W_c[h_{t-1}, x_t] + b_c)$$

- $\sigma$ decides what values to update
- tanh generates "candidate values" that could be added to cell state

https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

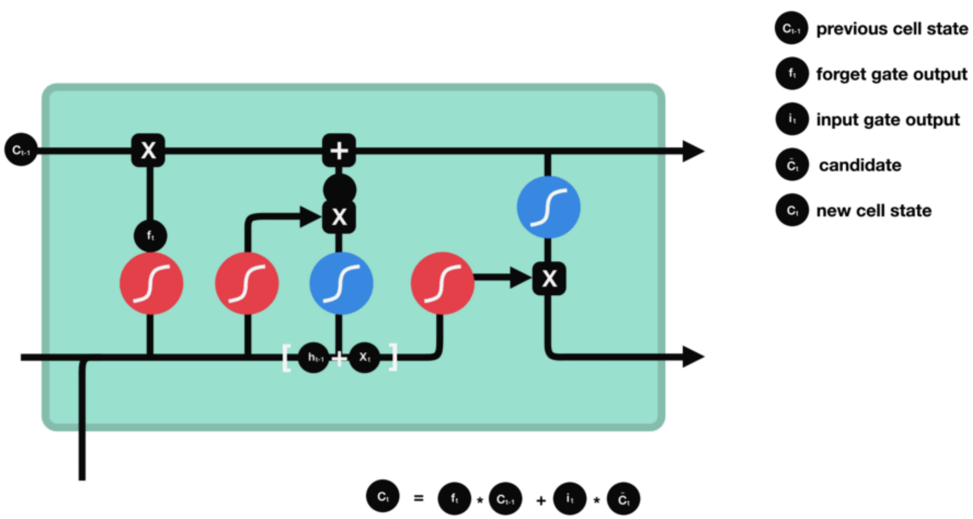
# LSTM: Update Cell State




$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t$$

- $f_t * c_{t-1}$ is to apply forget gate to previous cell state
- $i_t * \tilde{c}_t$ is to apply input gate to add new candidate values to cell state

# LSTM: Update Cell State

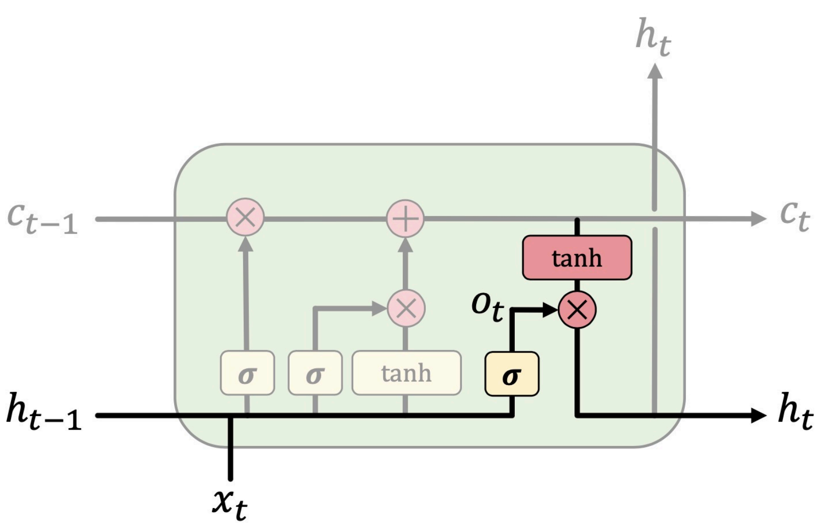



$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t$$

- $f_t * c_{t-1}$ is to apply forget gate to previous cell state
- $i_t * \tilde{c}_t$ is to apply input gate to add new candidate values to cell state

https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

# LSTM: Output Filtered Version of Cell State



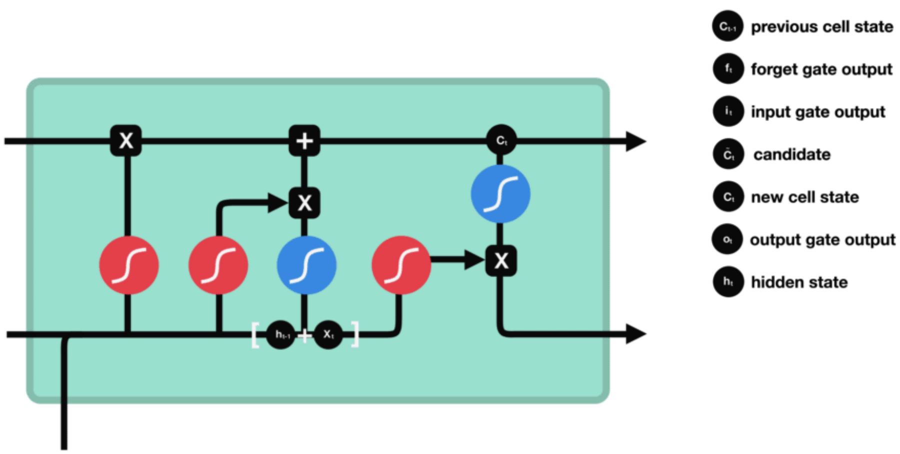

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(c_t)$$

- $\sigma$ decides what parts of the cell state to output as current hidden state
- tanh squashes values between -1 and 1
- $o_t * \tanh(c_t)$ is to output filtered version of cell state
- $h_t$ will be used to compute $\hat{y}_t$

https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

# LSTM: Output Filtered Version of Cell State




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

$$h_t = o_t * \tanh(c_t)$$

- $\sigma$ decides what parts of the current state and input to output as current hidden state
- tanh squashes values between -1 and 1
- $o_t * \tanh(c_t)$ is to output filtered version of cell state
- $h_t$ will be used to compute $\hat{y}_t$

https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

# LSTM: Feed Forward

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

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

$$\tilde{c}_t = \tanh(W_c[h_{t-1}, x_t] + b_c)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

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

$$h_t = o_t \odot \tanh(c_t)$$

$\odot$ is element-wise multiplication

Rewrite the functions for computing backpropagation

$$a_f = W_{hf} \cdot h_{t-1} + W_{xf} \cdot x_t + b_f$$

$$a_i = W_{hi} \cdot h_{t-1} + W_{xi} \cdot x_t + b_i$$

$$a_g = W_{hg} \cdot h_{t-1} + W_{xg} \cdot x_t + b_g$$

$$a_o = W_{ho} \cdot h_{t-1} + W_{xo} \cdot x_t + b_o$$

$$f_t = \sigma\left(a_f\right)$$

$$i_t = \sigma(a_i)$$

$$\tilde{c}_t = \tanh\left(a_g\right)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

$$o_t = \sigma(a_o)$$

$$h_t = o_t \odot \tanh(c_t)$$

# LSTM: Backpropagation Through Time

$$a_f = W_{hf} \cdot h_{t-1} + W_{xf} \cdot x_t + b_f$$
$$a_i = W_{hi} \cdot h_{t-1} + W_{xi} \cdot x_t + b_i$$
$$a_g = W_{hg} \cdot h_{t-1} + W_{xg} \cdot x_t + b_g$$
$$a_o = W_{ho} \cdot h_{t-1} + W_{xo} \cdot x_t + b_o$$
$$f_t = \sigma(a_f)$$
$$i_t = \sigma(a_i)$$
$$\tilde{c}_t = \tanh(a_g)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$
$$o_t = \sigma(a_o)$$
$$h_t = o_t \odot \tanh(c_t)$$

- Compute gradient w.r.t. hidden state

$$\frac{\partial L_t}{\partial h_t} = \frac{\partial L_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial h_t}$$

This depends on the output function $\hat{y}_t$ = output_function($h_t$), e.g., fully connected layer



https://cs231n.github.io/optimization-2/

# LSTM: Backpropagation Through Time

$$a_f = W_{hf} \cdot h_{t-1} + W_{xf} \cdot x_t + b_f$$
$$a_i = W_{hi} \cdot h_{t-1} + W_{xi} \cdot x_t + b_i$$
$$a_g = W_{hg} \cdot h_{t-1} + W_{xg} \cdot x_t + b_g$$
$$a_o = W_{ho} \cdot h_{t-1} + W_{xo} \cdot x_t + b_o$$
$$f_t = \sigma(a_f)$$
$$i_t = \sigma(a_i)$$
$$\tilde{c}_t = \tanh(a_g)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$
$$o_t = \sigma(a_o)$$
$$h_t = o_t \odot \tanh(c_t)$$

- Compute gradient w.r.t. output gate
  - $\frac{\partial L}{\partial o_t} = \frac{\partial L}{\partial h_t} \cdot \frac{\partial h_t}{\partial o_t} = \frac{\partial L}{\partial h_t} \cdot \tanh(c_t)$
  - $\frac{\partial L}{\partial a_o} = \frac{\partial L}{\partial o_t} \cdot \frac{\partial o_t}{\partial a_o} = \frac{\partial L}{\partial h_t} \cdot \tanh(c_t) \cdot \frac{d(\sigma(a_o))}{d a_o}$

    $= \frac{\partial L}{\partial h_t} \cdot \tanh(c_t) \cdot \sigma(a_o)(1 - \sigma(a_o))$

    $= \frac{\partial L}{\partial h_t} \cdot \tanh(c_t) \cdot o_t(1 - o_t)$
  - $\frac{\partial L}{\partial W_{ho}} = \frac{\partial L}{\partial a_o} \cdot \frac{\partial a_o}{\partial W_{ho}} = \frac{\partial L}{\partial a_o} \cdot h_{t-1}$
  - $\frac{\partial L}{\partial W_{xo}} = \frac{\partial L}{\partial a_o} \cdot \frac{\partial a_o}{\partial W_{xo}} = \frac{\partial L}{\partial a_o} \cdot x_t$
  - $\frac{\partial L}{\partial b_o} = \frac{\partial L}{\partial a_o} \cdot \frac{\partial a_o}{\partial b_o} = \frac{\partial L}{\partial a_o}$

e.g., when $W_{ho}$ changes for a small amount ($\partial W_{ho}$), how much would $L$ change (and the change direction)?

# LSTM: Backpropagation Through Time

$$a_f = W_{hf} \cdot h_{t-1} + W_{xf} \cdot x_t + b_f$$

$$a_i = W_{hi} \cdot h_{t-1} + W_{xi} \cdot x_t + b_i$$

$$a_g = W_{hg} \cdot h_{t-1} + W_{xg} \cdot x_t + b_g$$

$$a_o = W_{ho} \cdot h_{t-1} + W_{xo} \cdot x_t + b_o$$

$$f_t = \sigma(a_f)$$

$$i_t = \sigma(a_i)$$

$$\tilde{c}_t = \tanh(a_g)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

$$o_t = \sigma(a_o)$$

$$h_t = o_t \odot \tanh(c_t)$$

- Compute gradient w.r.t. cell state
  - $\frac{\partial L}{\partial c_t} = \frac{\partial L}{\partial h_t} \cdot \frac{\partial h_t}{\partial c_t} = \frac{\partial L}{\partial h_t} \cdot o_t \cdot (1 - \tanh(c_t)^2)$
  - $\frac{\partial L}{\partial \tilde{c}_t} = \frac{\partial L}{\partial c_t} \cdot \frac{\partial c_t}{\partial \tilde{c}_t} = \frac{\partial L}{\partial c_t} \cdot i_t$
  - $\frac{\partial L}{\partial a_g} = \frac{\partial L}{\partial \tilde{c}_t} \cdot \frac{\partial \tilde{c}_t}{\partial a_g} = \frac{\partial L}{\partial c_t} \cdot i_t \cdot \frac{d(\tanh(a_g))}{d a_g} = \frac{\partial L}{\partial c_t} \cdot i_t \cdot (1 - {\tilde{c}_t}^2)$
  - $\frac{\partial L}{\partial W_{hg}} = \frac{\partial L}{\partial a_g} \cdot \frac{\partial a_g}{\partial W_{hg}} = \frac{\partial L}{\partial a_g} \cdot h_{t-1}$
  - $\frac{\partial L}{\partial W_{xg}} = \frac{\partial L}{\partial a_g} \cdot \frac{\partial a_g}{\partial W_{xg}} = \frac{\partial L}{\partial a_g} \cdot x_t$
  - $\frac{\partial L}{\partial b_g} = \frac{\partial L}{\partial a_g} \cdot \frac{\partial a_g}{\partial b_g} = \frac{\partial L}{\partial a_g}$

# LSTM: Backpropagation Through Time

$$a_f = W_{hf} \cdot h_{t-1} + W_{xf} \cdot x_t + b_f$$
$$a_i = W_{hi} \cdot h_{t-1} + W_{xi} \cdot x_t + b_i$$
$$a_g = W_{hg} \cdot h_{t-1} + W_{xg} \cdot x_t + b_g$$
$$a_o = W_{ho} \cdot h_{t-1} + W_{xo} \cdot x_t + b_o$$
$$f_t = \sigma(a_f)$$
$$i_t = \sigma(a_i)$$
$$\tilde{c}_t = \tanh(a_g)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$
$$o_t = \sigma(a_o)$$
$$h_t = o_t \odot \tanh(c_t)$$

- Compute gradient w.r.t. input gate
  - $\frac{\partial L}{\partial i_t} = \frac{\partial L}{\partial c_t} \cdot \frac{\partial c_t}{\partial i_t} = \frac{\partial L}{\partial c_t} \cdot \tilde{c}_t$
  - $\frac{\partial L}{\partial a_i} = \frac{\partial L}{\partial i_t} \cdot \frac{\partial i_t}{\partial a_i} = \frac{\partial L}{\partial c_t} \cdot \tilde{c}_t \cdot \frac{d(\sigma(a_i))}{d a_i}$

    $= \frac{\partial L}{\partial c_t} \cdot \tilde{c}_t \cdot \sigma(a_i)(1 - \sigma(a_i)) = \frac{\partial L}{\partial c_t} \cdot \tilde{c}_t \cdot i_t(1 - i_t)$
  - $\frac{\partial L}{\partial W_{hi}} = \frac{\partial L}{\partial a_i} \cdot \frac{\partial a_i}{\partial W_{hi}} = \frac{\partial L}{\partial a_i} \cdot h_{t-1}$
  - $\frac{\partial L}{\partial W_{xi}} = \frac{\partial L}{\partial a_i} \cdot \frac{\partial a_i}{\partial W_{xi}} = \frac{\partial L}{\partial a_i} \cdot x_t$
  - $\frac{\partial L}{\partial b_i} = \frac{\partial L}{\partial a_i} \cdot \frac{\partial a_i}{\partial b_i} = \frac{\partial L}{\partial a_i}$

# LSTM: Backpropagation Through Time

$$a_f = W_{hf} \cdot h_{t-1} + W_{xf} \cdot x_t + b_f$$
$$a_i = W_{hi} \cdot h_{t-1} + W_{xi} \cdot x_t + b_i$$
$$a_g = W_{hg} \cdot h_{t-1} + W_{xg} \cdot x_t + b_g$$
$$a_o = W_{ho} \cdot h_{t-1} + W_{xo} \cdot x_t + b_o$$
$$f_t = \sigma(a_f)$$
$$i_t = \sigma(a_i)$$
$$\tilde{c}_t = \tanh(a_g)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$
$$o_t = \sigma(a_o)$$
$$h_t = o_t \odot \tanh(c_t)$$

- Compute gradient w.r.t. forget gate
  - $\frac{\partial L}{\partial f_t} = \frac{\partial L}{\partial c_t} \cdot \frac{\partial c_t}{\partial f_t} = \frac{\partial L}{\partial c_t} \cdot c_{t-1}$
  - $\frac{\partial L}{\partial a_f} = \frac{\partial L}{\partial f_t} \cdot \frac{\partial f_t}{\partial a_f} = \frac{\partial L}{\partial c_t} \cdot c_{t-1} \cdot \frac{d\left(\sigma(a_f)\right)}{d a_f}$

    $= \frac{\partial L}{\partial c_t} \cdot c_{t-1} \cdot \sigma(a_f)\left(1 - \sigma(a_f)\right) = \frac{\partial L}{\partial c_t} \cdot c_{t-1} \cdot f_t(1 - f_t)$
  - $\frac{\partial L}{\partial W_{hf}} = \frac{\partial L}{\partial a_f} \cdot \frac{\partial a_f}{\partial W_{hf}} = \frac{\partial L}{\partial a_f} \cdot h_{t-1}$
  - $\frac{\partial L}{\partial W_{xf}} = \frac{\partial L}{\partial a_f} \cdot \frac{\partial a_f}{\partial W_{xf}} = \frac{\partial L}{\partial a_f} \cdot x_t$
  - $\frac{\partial L}{\partial b_f} = \frac{\partial L}{\partial a_f} \cdot \frac{\partial a_f}{\partial b_f} = \frac{\partial L}{\partial a_f}$

# LSTM: Backpropagation Through Time

$$a_f = W_{hf} \cdot h_{t-1} + W_{xf} \cdot x_t + b_f$$
$$a_i = W_{hi} \cdot h_{t-1} + W_{xi} \cdot x_t + b_i$$
$$a_g = W_{hg} \cdot h_{t-1} + W_{xg} \cdot x_t + b_g$$
$$a_o = W_{ho} \cdot h_{t-1} + W_{xo} \cdot x_t + b_o$$
$$f_t = \sigma(a_f)$$
$$i_t = \sigma(a_i)$$
$$\tilde{c}_t = \tanh(a_g)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$
$$o_t = \sigma(a_o)$$
$$h_t = o_t \odot \tanh(c_t)$$

- These computation for backpropagation will be **calculated $\boldsymbol{T}$ times** (the number of time steps)
- The weights will be updated using the accumulated gradient w.r.t. each weight for all time steps
- For example, $\frac{\partial L}{\partial W_{hf}} = \sum_{t=1}^{T} \frac{\partial L}{\partial W_{hf}^{t}}$

$$W_{hf} \mathrel{+}= \alpha * \frac{\partial L}{\partial W_{hf}}$$

# LSTM: Mitigate Vanishing Gradient

- Vanilla RNNs $$\frac{\partial \mathcal{E}}{\partial \theta} = \frac{\partial \mathcal{E}}{\partial \boldsymbol{h}_n} \sum_{k=1}^{n} \left( \left( \prod_{i=k+1}^{n} \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{h}_{i-1}} \right) \frac{\partial \boldsymbol{h}_k}{\partial \theta} \right)$$

$$\left\| \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{h}_{i-1}} \right\| < 1 \quad \rightarrow \quad \prod_{i=2}^{n} \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{h}_{i-1}} \quad \dots \quad \text{Vanish!}$$

$$\left\| \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{h}_{i-1}} \right\| > 1 \quad \rightarrow \quad \prod_{i=2}^{n} \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{h}_{i-1}} \quad \dots \quad \text{Explode!}$$



https://naokishibuya.medium.com/long-short-term-memory-394aa8461a35

# LSTM: Mitigate Vanishing Gradient

- Vanilla RNNs $$\frac{\partial \mathcal{E}}{\partial \theta} = \frac{\partial \mathcal{E}}{\partial \boldsymbol{h}_n} \sum_{k=1}^{n} \left( \left( \prod_{i=k+1}^{n} \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{h}_{i-1}} \right) \frac{\partial \boldsymbol{h}_k}{\partial \theta} \right)$$
- LSTM $$\frac{\partial \mathcal{E}}{\partial \theta} = \frac{\partial \mathcal{E}}{\partial \boldsymbol{c}_n} \sum_{k=1}^{n} \left( \left( \prod_{i=k+1}^{n} \frac{\partial \boldsymbol{c}_i}{\partial \boldsymbol{c}_{i-1}} \right) \frac{\partial \boldsymbol{c}_k}{\partial \theta} \right)$$
- Recall that in
  - Vanilla RNNS, $h_t = \tanh(\boldsymbol{W_{hh}} h_{t-1} + \boldsymbol{W_{xh}} x_t)$
  - LSTM, $h_t = o_t \odot \tanh(c_t)$ and $c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$



https://naokishibuya.medium.com/long-short-term-memory-394aa8461a35

# LSTM: Mitigate Vanishing Gradient

- LSTM

$$\frac{\partial \mathcal{E}}{\partial \theta} = \frac{\partial \mathcal{E}}{\partial \boldsymbol{c}_n} \sum_{k=1}^{n} \left( \left( \prod_{i=k+1}^{n} \frac{\partial \boldsymbol{c}_i}{\partial \boldsymbol{c}_{i-1}} \right) \frac{\partial \boldsymbol{c}_k}{\partial \theta} \right)$$

$$c_t = c_{t-1} \otimes \sigma(W_f \cdot [h_{t-1}, x_t]) \oplus$$

$$tanh\,(W_c \cdot [h_{t-1}, x_t]) \otimes \sigma(W_i \cdot [h_{t-1}, x_t])$$

$$\frac{\partial c_t}{\partial c_{t-1}} = \frac{\partial}{\partial c_{t-1}} [c_{t-1} \otimes f_t \oplus \tilde{c}_t \otimes i_t]$$

$$= \frac{\partial}{\partial c_{t-1}} [c_{t-1} \otimes f_t] + \frac{\partial}{\partial c_{t-1}} [\tilde{c}_t \otimes i_t]$$

$$= \frac{\partial f_t}{\partial c_{t-1}} \cdot c_{t-1} + \frac{\partial c_{t-1}}{\partial c_{t-1}} \cdot f_t + \frac{\partial i_t}{\partial c_{t-1}} \cdot \tilde{c}_t + \frac{\partial \tilde{c}_t}{\partial c_{t-1}} \cdot i_t$$

Note that the notation is different from previous slides



https://medium.datadriveninvestor.com/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577

# LSTM: Mitigate Vanishing Gradient

- LSTM

$$\frac{\partial \mathcal{E}}{\partial \theta} = \frac{\partial \mathcal{E}}{\partial \boldsymbol{c}_n} \sum_{k=1}^{n} \left( \left( \prod_{i=k+1}^{n} \frac{\partial \boldsymbol{c}_i}{\partial \boldsymbol{c}_{i-1}} \right) \frac{\partial \boldsymbol{c}_k}{\partial \theta} \right)$$

$$a_f = W_{hf} \cdot h_{t-1} + W_{xf} \cdot x_t + b_f$$
$$a_i = W_{hi} \cdot h_{t-1} + W_{xi} \cdot x_t + b_i$$
$$a_g = W_{hg} \cdot h_{t-1} + W_{xg} \cdot x_t + b_g$$
$$a_o = W_{ho} \cdot h_{t-1} + W_{xo} \cdot x_t + b_o$$
$$f_t = \sigma(a_f)$$
$$i_t = \sigma(a_i)$$
$$\tilde{c}_t = \tanh(a_g)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$
$$o_t = \sigma(a_o)$$
$$h_t = o_t \odot \tanh(c_t)$$

$$c_t = c_{t-1} \otimes \sigma(W_f \cdot [h_{t-1}, x_t]) \oplus tanh\,(W_c \cdot [h_{t-1}, x_t]) \otimes \sigma(W_i \cdot [h_{t-1}, x_t])$$

$$\mathrm{sigm}'(x) = \mathrm{sigm}(x)(1 - \mathrm{sigm}(x))$$

$$\frac{\partial c_t}{\partial c_{t-1}} = \frac{\partial}{\partial c_{t-1}} [c_{t-1} \otimes f_t \oplus \tilde{c}_t \otimes i_t]$$
$$= \frac{\partial}{\partial c_{t-1}} [c_{t-1} \otimes f_t] + \frac{\partial}{\partial c_{t-1}} [\tilde{c}_t \otimes i_t]$$
$$= \frac{\partial f_t}{\partial c_{t-1}} \cdot c_{t-1} + \frac{\partial c_{t-1}}{\partial c_{t-1}} \cdot f_t + \frac{\partial i_t}{\partial c_{t-1}} \cdot \tilde{c}_t + \frac{\partial \tilde{c}_t}{\partial c_{t-1}} \cdot i_t$$

$$\frac{\partial c_t}{\partial c_{t-1}} = \sigma'(W_f \cdot [h_{t-1}, x_t]) \cdot W_f \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot c_{t-1}$$
$$+ f_t$$
$$+ \sigma'(W_i \cdot [h_{t-1}, x_t]) \cdot W_i \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot \tilde{c}_t$$
$$+ \sigma'(W_c \cdot [h_{t-1}, x_t]) \cdot W_c \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot i_t$$



https://medium.datadriveninvestor.com/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577

# LSTM: Mitigate Vanishing Gradient

- LSTM

$$\frac{\partial \mathcal{E}}{\partial \theta} = \frac{\partial \mathcal{E}}{\partial \boldsymbol{c}_n} \sum_{k=1}^{n} \left( \left( \prod_{i=k+1}^{n} \frac{\partial \boldsymbol{c}_i}{\partial \boldsymbol{c}_{i-1}} \right) \frac{\partial \boldsymbol{c}_k}{\partial \theta} \right)$$

$$\frac{\partial c_t}{\partial c_{t-1}} = \sigma'\left(W_f \cdot [h_{t-1}, x_t]\right) \cdot W_f \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot c_{t-1}$$

$$+ f_t$$

$$+ \sigma'(W_i \cdot [h_{t-1}, x_t]) \cdot W_i \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot \tilde{c}_t$$

$$+ \sigma'(W_c \cdot [h_{t-1}, x_t]) \cdot W_c \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot i_t$$

$$A_t = \sigma'\left(W_f \cdot [h_{t-1}, x_t]\right) \cdot W_f \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot c_{t-1}$$

$$B_t = f_t$$

$$C_t = \sigma'(W_i \cdot [h_{t-1}, x_t]) \cdot W_i \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot \tilde{c}_t$$

$$D_t = \sigma'(W_c \cdot [h_{t-1}, x_t]) \cdot W_c \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot i_t$$

$$\frac{\partial c_t}{\partial c_{t-1}} = A_t + B_t + C_t + D_t \qquad (6)$$



https://medium.datadriveninvestor.com/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577

# LSTM: Mitigate Vanishing Gradient

- LSTM

$$\frac{\partial \mathcal{E}}{\partial \theta} = \frac{\partial \mathcal{E}}{\partial \boldsymbol{c}_n} \sum_{k=1}^{n} \left( \left( \prod_{i=k+1}^{n} \frac{\partial \boldsymbol{c}_i}{\partial \boldsymbol{c}_{i-1}} \right) \frac{\partial \boldsymbol{c}_k}{\partial \theta} \right)$$

$$\frac{\partial c_t}{\partial c_{t-1}} = A_t + B_t + C_t + D_t \qquad (6)$$

- Addictive function (rather than multiplying) and $B_t$ (forget gate vector) help mitigate the gradient vanishing problem



https://medium.datadriveninvestor.com/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577

# LSTM: Key Concepts

- Maintain a separate **cell state** from what is outputted
- Use **gates** to control the flow of information
  - Forget gate gets rid of irrelevant information
  - Selectively updates cell state
  - Output gate returns a filtered version of the cell state
- LSTM can mitigate vanishing gradient problem

# Acknowledgements


- Deep learning slides adapted from https://m2dsupsdlclass.github.io/lectures-labs/ by Olivier Grisel and Charles Ollion (CC-By 4.0 license)
- Gil, Yolanda (Ed.) Introduction to Computational Thinking and Data Science. Available from http://www.datascience4all.org
- https://lilianweng.github.io/posts/2018-08-12-vae/