Recurrent Neural Networks(RNN)

Why Recurrent Neural Networks

- So far we have seen networks such as densely connected networks and convolutional neural networks
- Each input shown to them is processed independently
- language translation, natural language processing (nlp), speech recognition, and image captioning
- Here Data is sequence
- Traditional neural networks can’t process such kind of sequential data, it is a major shortcoming.

Limitations of Feed Forward Neural Networks

- **1. Will consider only present input**
  - In Feed farward networks Previously processed image(cat) will not effect on next image(dog)
- **2. Fixed-sized vector as input**
  - (Fixed-sized vector input e.g. an image) and produce a fixed-sized vector as output (e.g. probabilities of different classes).

Limitations of Feed Forward Neural Networks

- **3. Can’t process variable length data**
  - in CNN all images are re scaled to fixed size before processing like 64x64

- In sequential data input can’t be fixed length
- So we need new type of models to process this kind of problems

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

Why Recurrent Neural Networks

- Recurrent neural networks address this issue.
- Humans don’t start their thinking from scratch every second.
- As we read essay, you understand each word based on your understanding of previous words.
- We don’t throw everything away and start thinking from scratch again.
- Our thoughts have persistence.
- Means we keeping memories of what came before

Recurrent Neural Networks

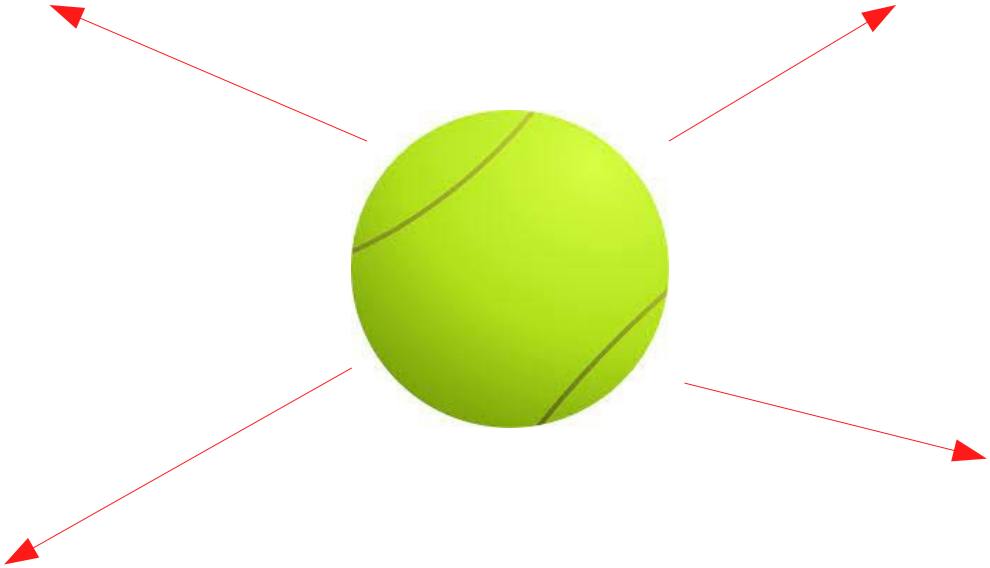
- Biological intelligence processes information incrementally
- while maintaining an internal model of what it's processing, built from past information
- and constantly updated as new information comes in.

Recurrent Neural Networks

- Can you predict where this ball go next?

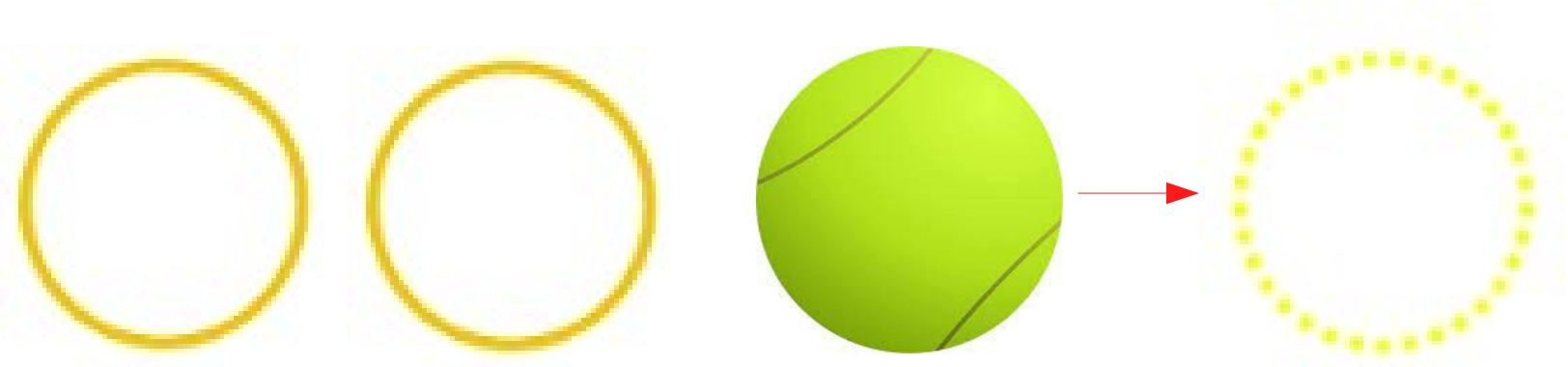
Recurrent Neural Networks

- Can you predict where this ball go next?



Recurrent Neural Networks

- Can you predict where this ball go next?



Recurrent Neural Networks

Text

RNN is a class of artificial neural networks

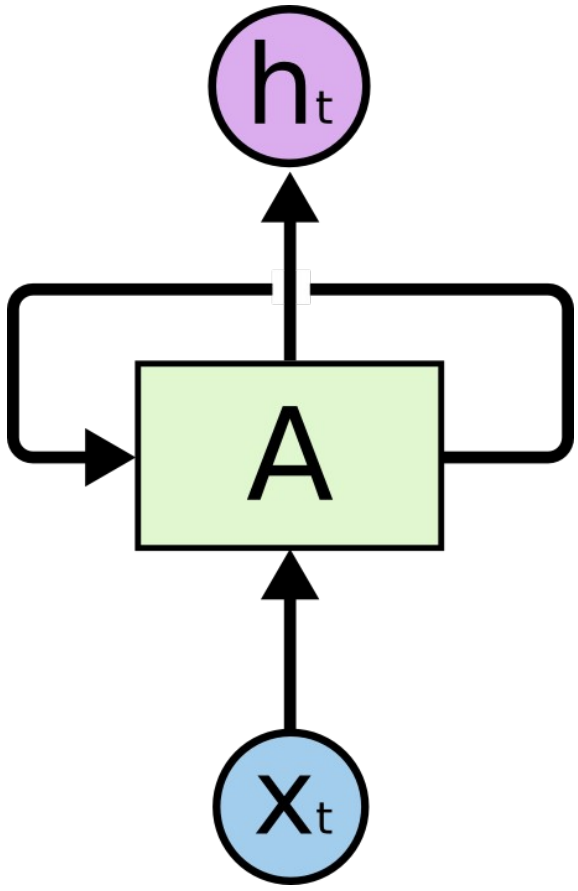
RNN is a class of artificial neural networks

R N N i s.....

Recurrent Neural Networks

- A chunk of neural network, A, looks at some input x_t and outputs a value h_t.
- A loop allows information to be passed from one step of the network to the next.

$$h_t = A(h_{t-1}, x_t)$$

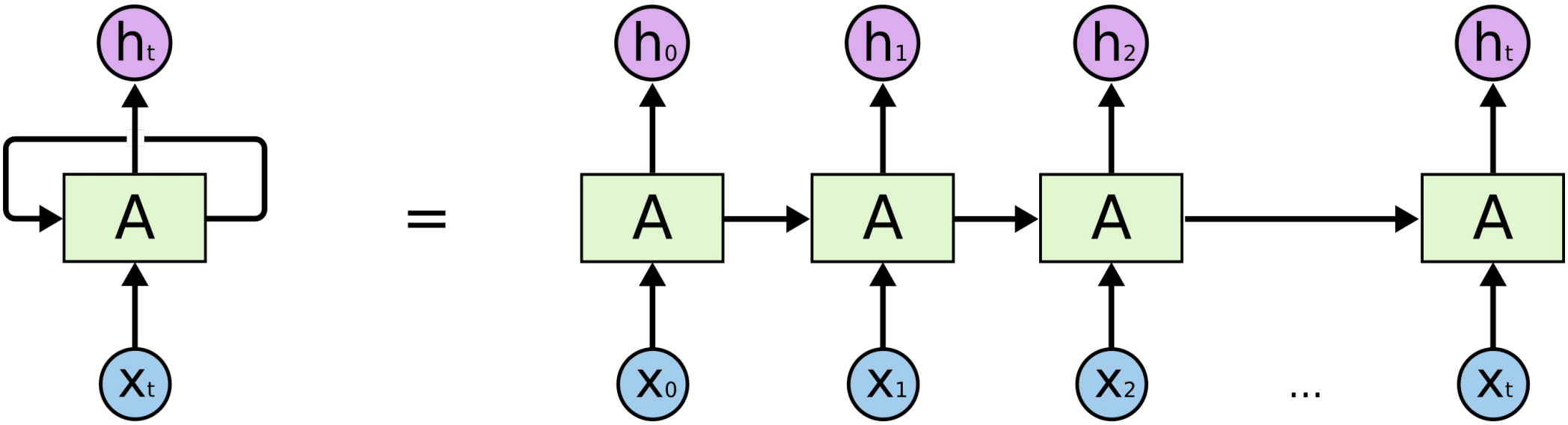


Christopher Olah https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent Neural Networks

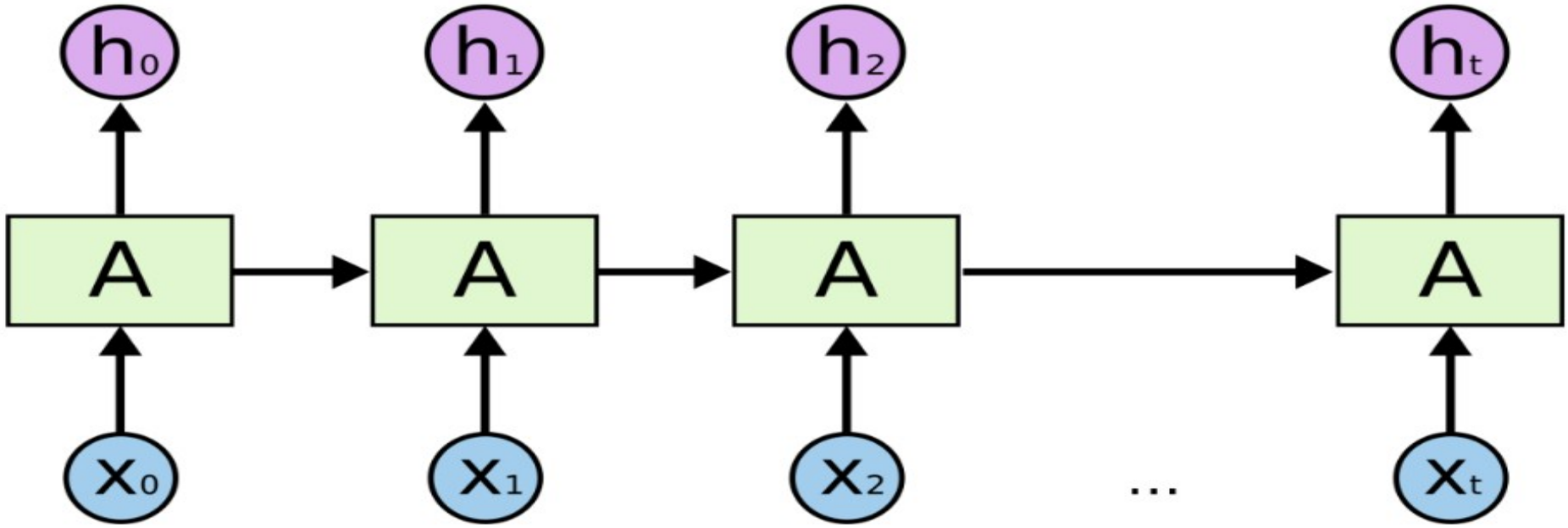
- A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.

An unrolled recurrent neural network.



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

Recurrent Neural Networks



$$h_t = g_t\left(x_t, x_{t-1}, x_{t-2}, \ldots x_2, x_1\right)$$

$$h_3 = A\left(A\left(A\left(h_{(-1)}, x_0\right), x_1\right), x_2\right)$$

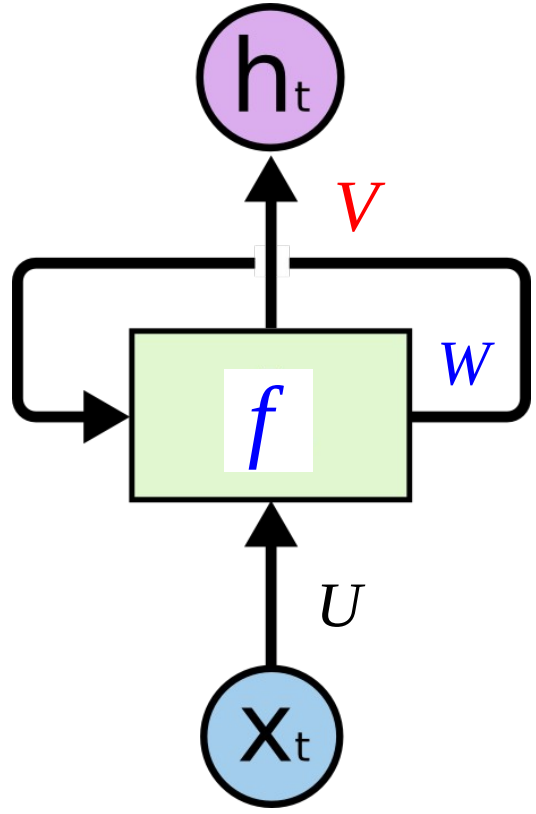
Reference: Deep Learning (Ian J. Goodfellow, Yoshua Bengio and Aaron Courville), MIT Press, 2016.

Recurrent Neural Networks

- Let x_t be the value of input at time t
- and let h_t be the network output at time t
- Many recurrent neural networks use

$$h_t = f\left(h_{t-1}, x_t, \theta\right)$$

- To define the values of their hidden units. To indicate that the state is the hidden units of the network



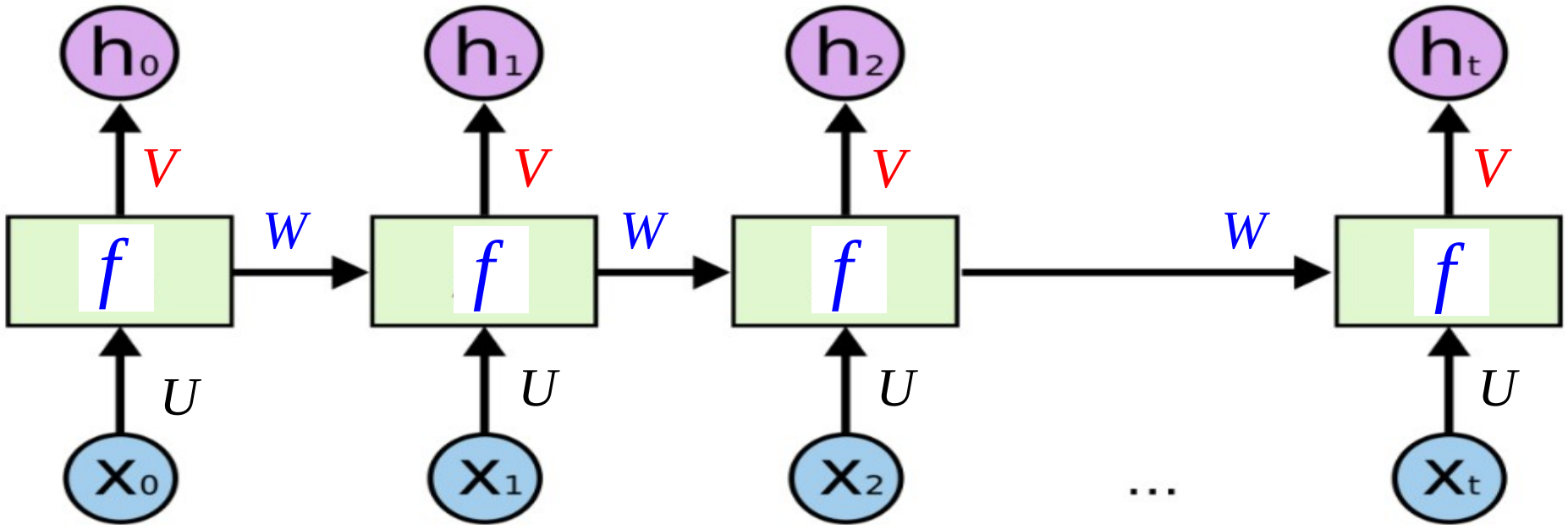
Reference: Deep Learning (Ian J. Goodfellow, Yoshua Bengio and Aaron Courville), MIT Press, 2016.

Recurrent Neural Networks

- The unfolding process thus introduces two major advantages:
  - 1. Regardless of the sequence length, the learned model always has the same input size, because it is specified in terms of transition from one state to another state, rather than specified in terms of a variable-length history of states.
  - 2. It is possible to use the same transition function f with the same parameters at every time step.

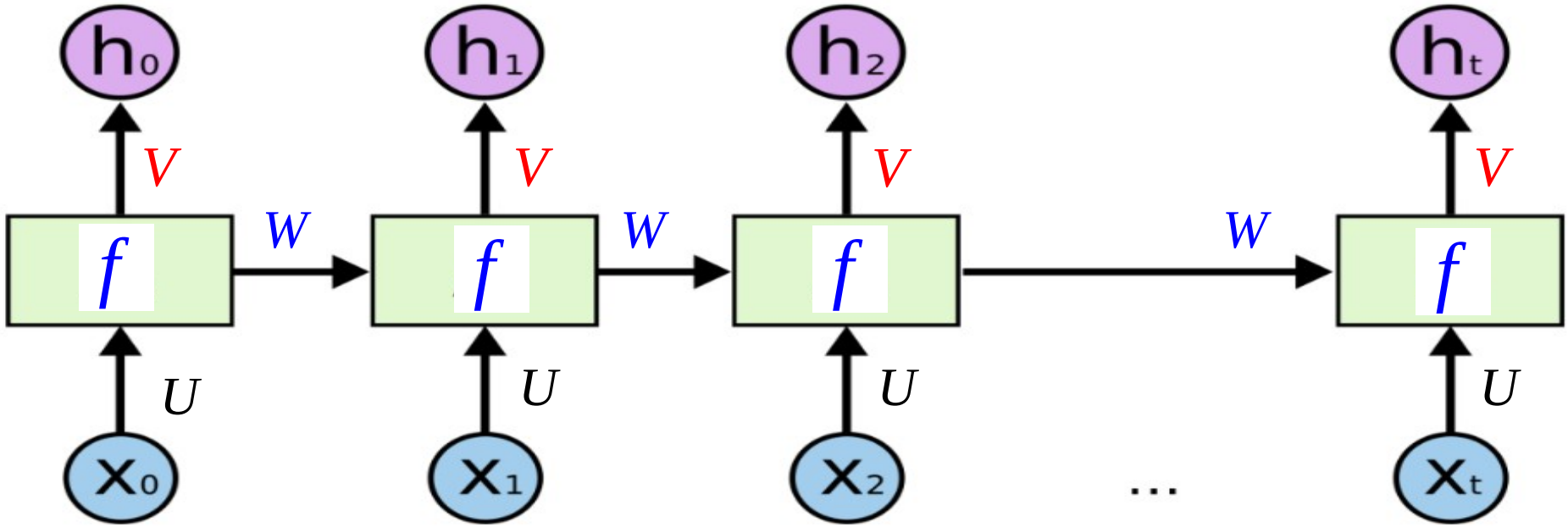
Reference: Deep Learning (Ian J. Goodfellow, Yoshua Bengio and Aaron Courville), MIT Press, 2016.

Recurrent Neural Networks



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

Recurrent Neural Networks



1. Variable length sequences

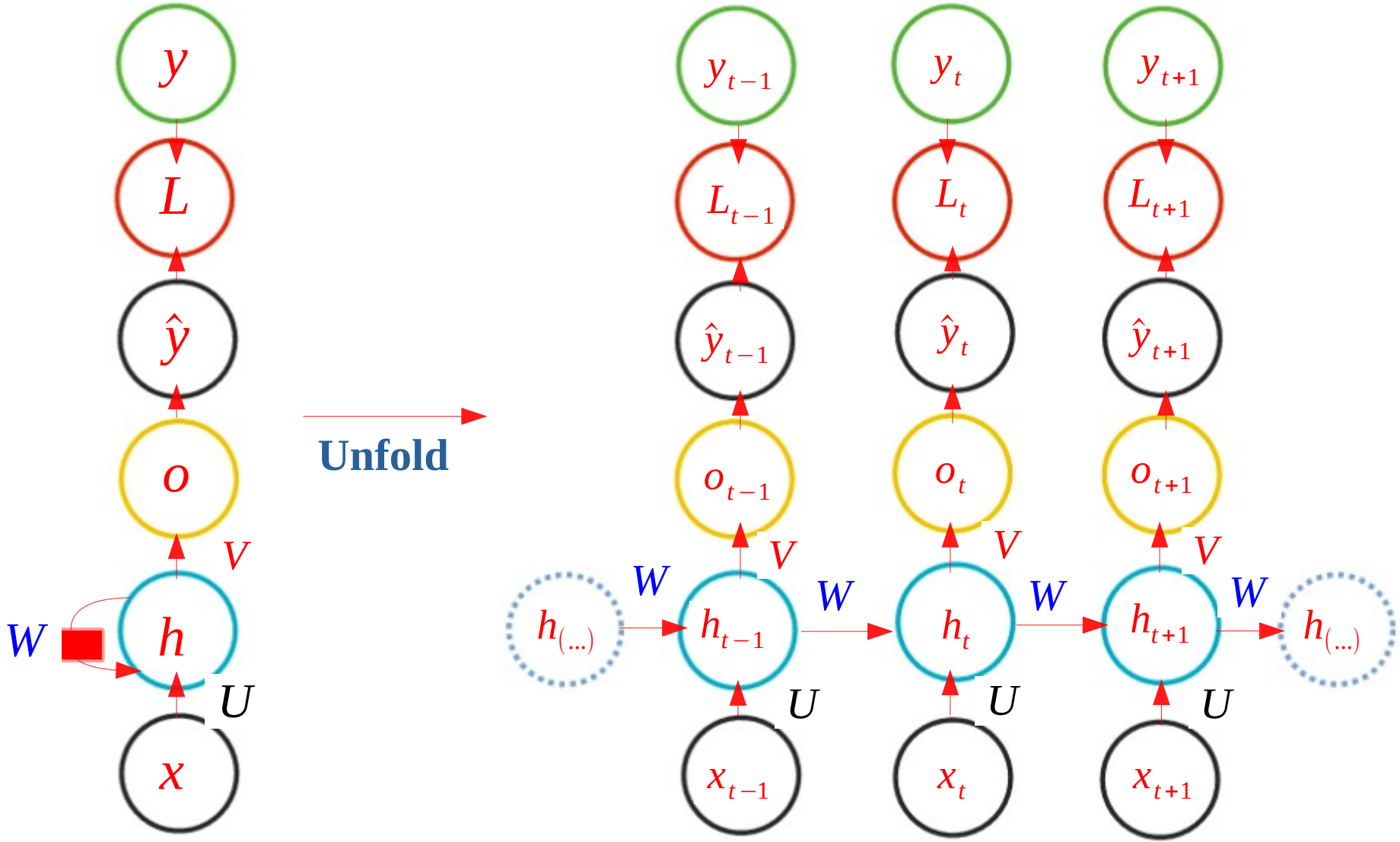
2. Parameter sharing across the sequence

3. Maintain information about order

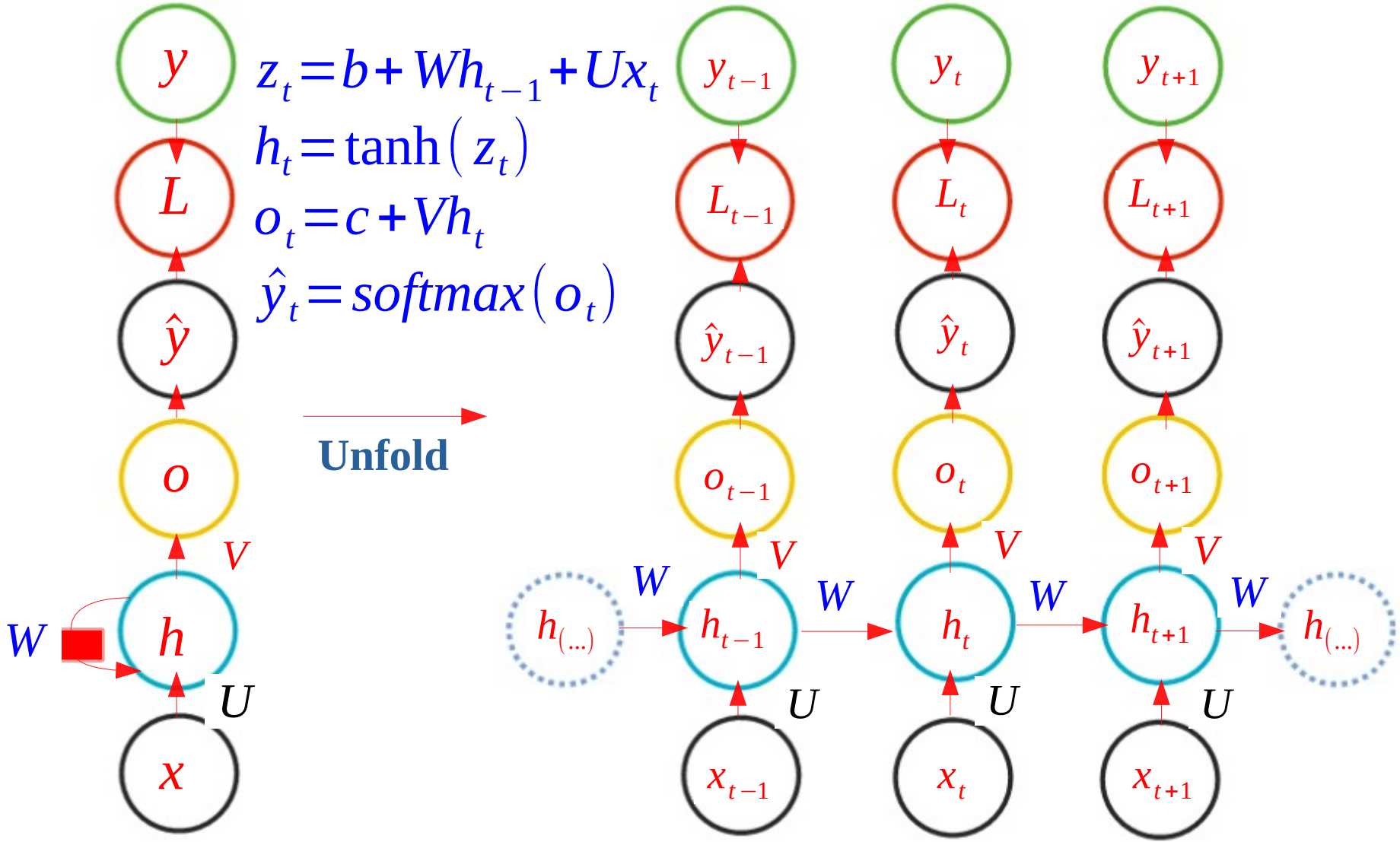
4. track long term dependencies but not too long

RNN - Forward Pass

- The forward pass of an RNN is the same as that of a multilayer perceptron with a single hidden layer
- Except that activations arrive at the hidden layer from both the current external input and the hidden layer activations from the previous timestep.



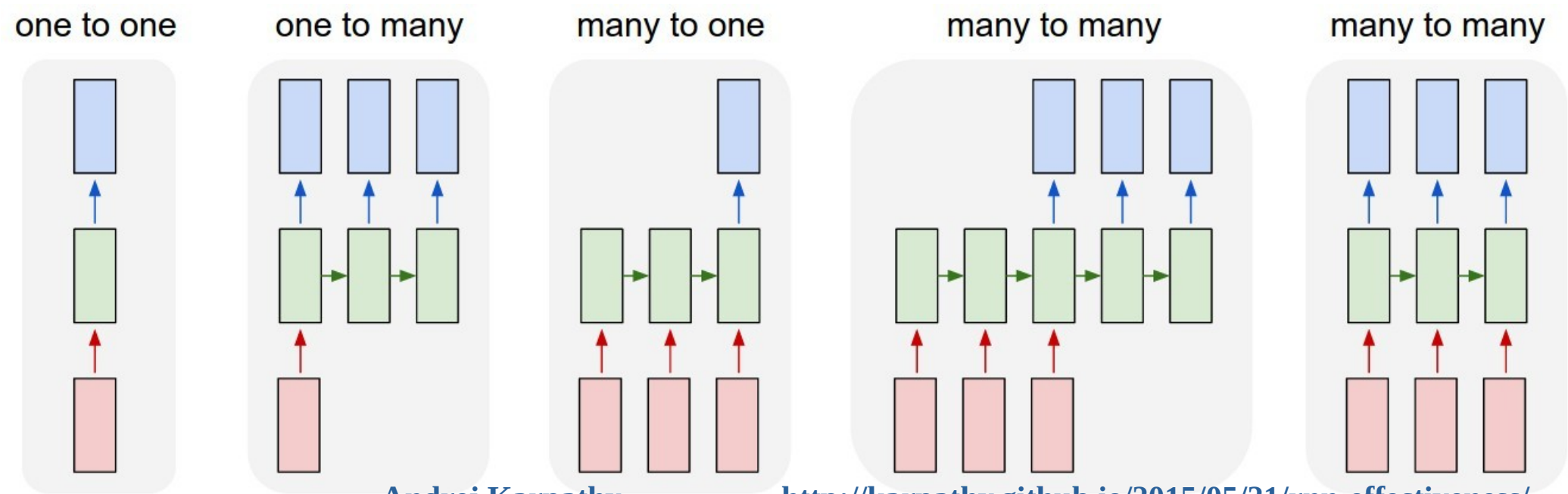
Reference: Deep Learning (Ian J. Goodfellow, Yoshua Bengio and Aaron Courville), MIT Press, 2016.



Reference: Deep Learning (Ian J. Goodfellow, Yoshua Bengio and Aaron Courville), MIT Press, 2016.

Types of Recurrent Neural Networks

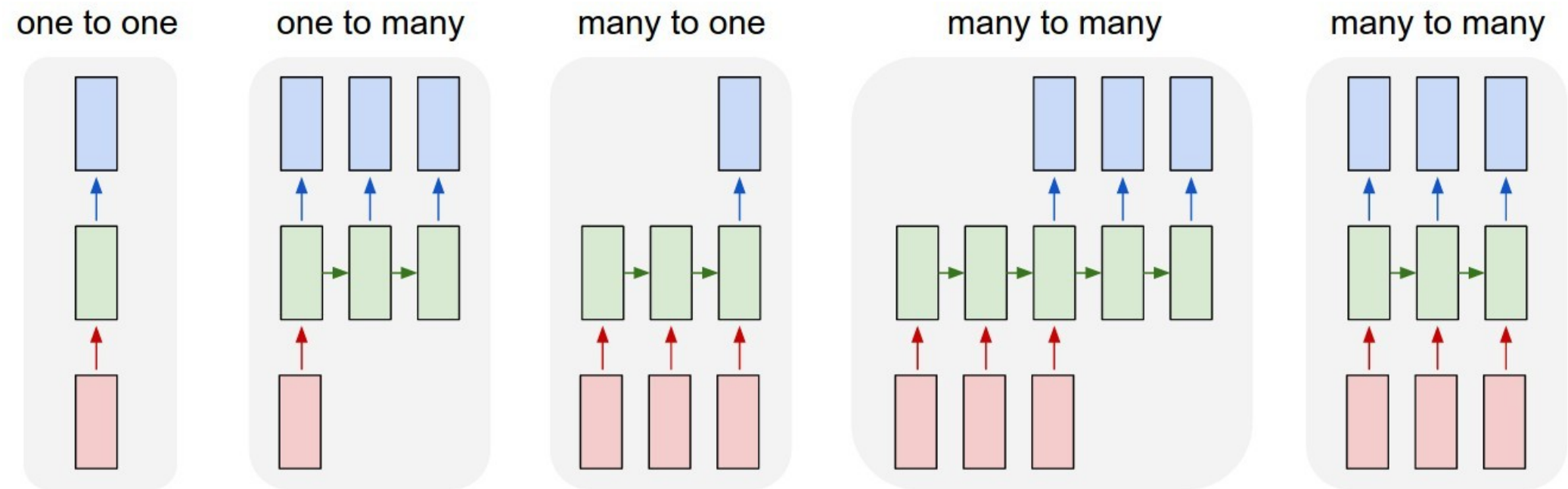
- The Recurrent Neural Nets allow us to operate over sequences of vectors: Sequences in the input, the output, or in the most general case both.



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

Types of Recurrent Neural Networks

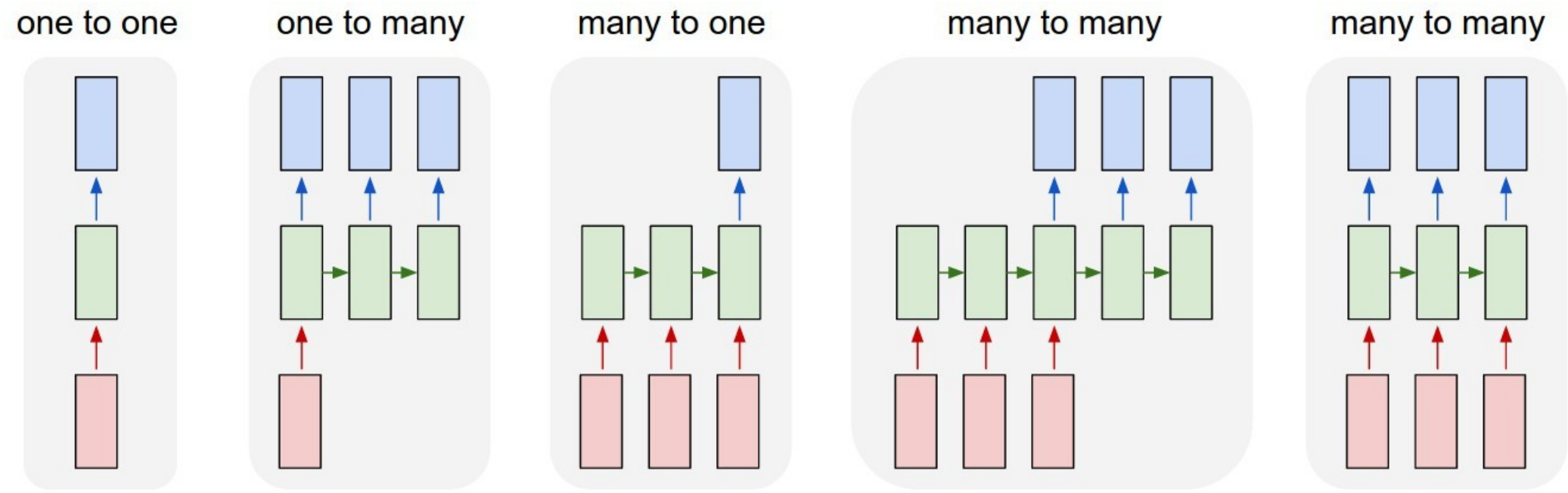
Feed farward Network



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

Types of Recurrent Neural Networks

Image caption generation

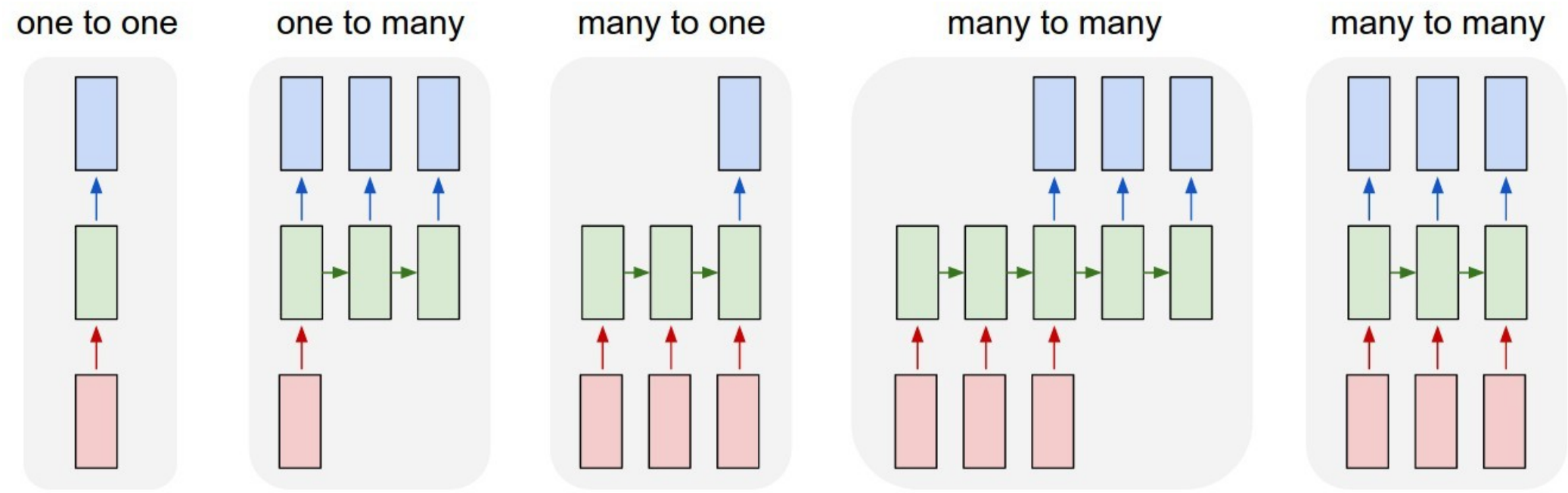


Andrej Karpathy **http://karpathy.github.io/2015/05/21/rnn-effectiveness/**

Types of Recurrent Neural Networks

Image caption Generation

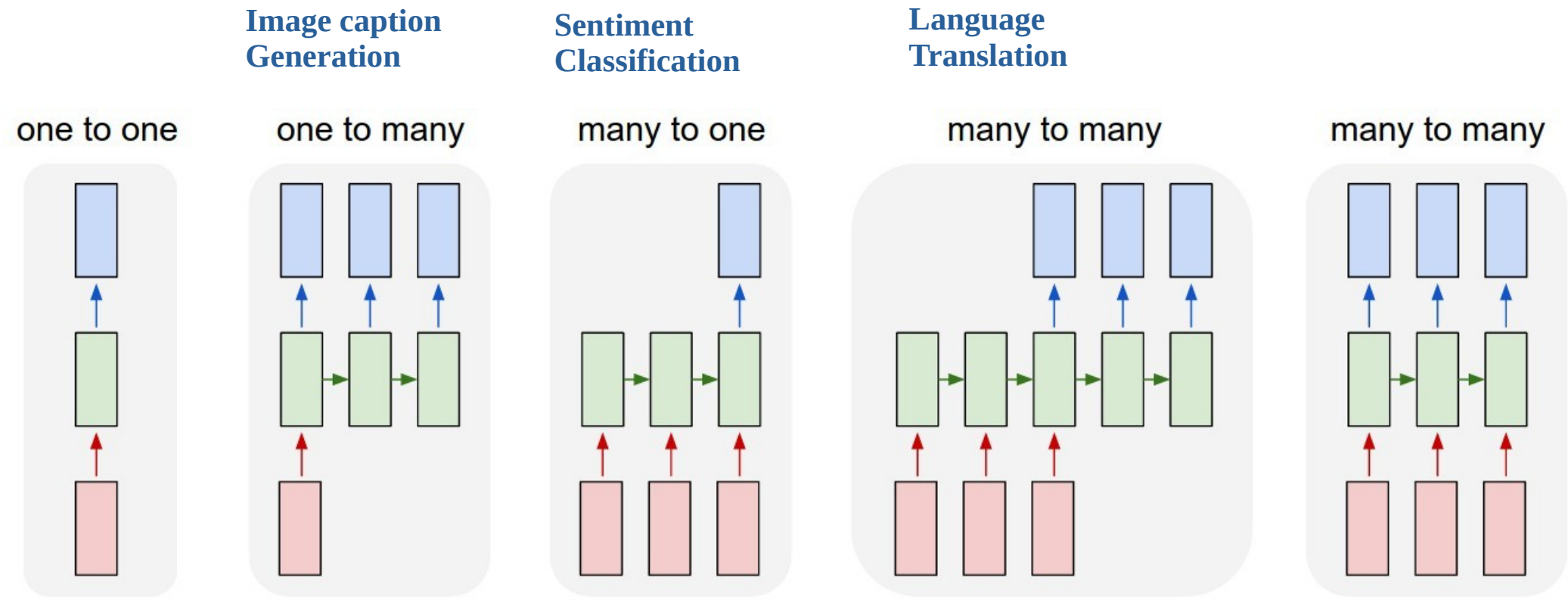
Sentiment Classification



Andrej Karpathy

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

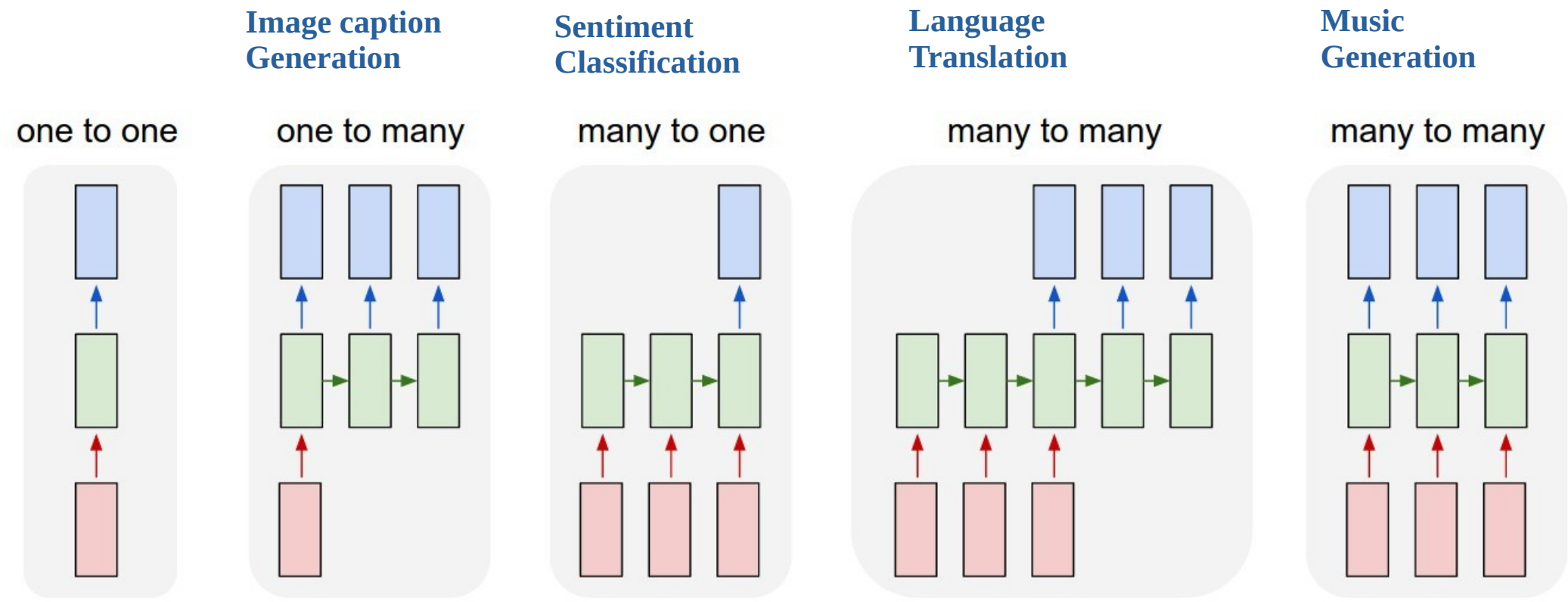
Types of Recurrent Neural Networks



Andrej Karpathy

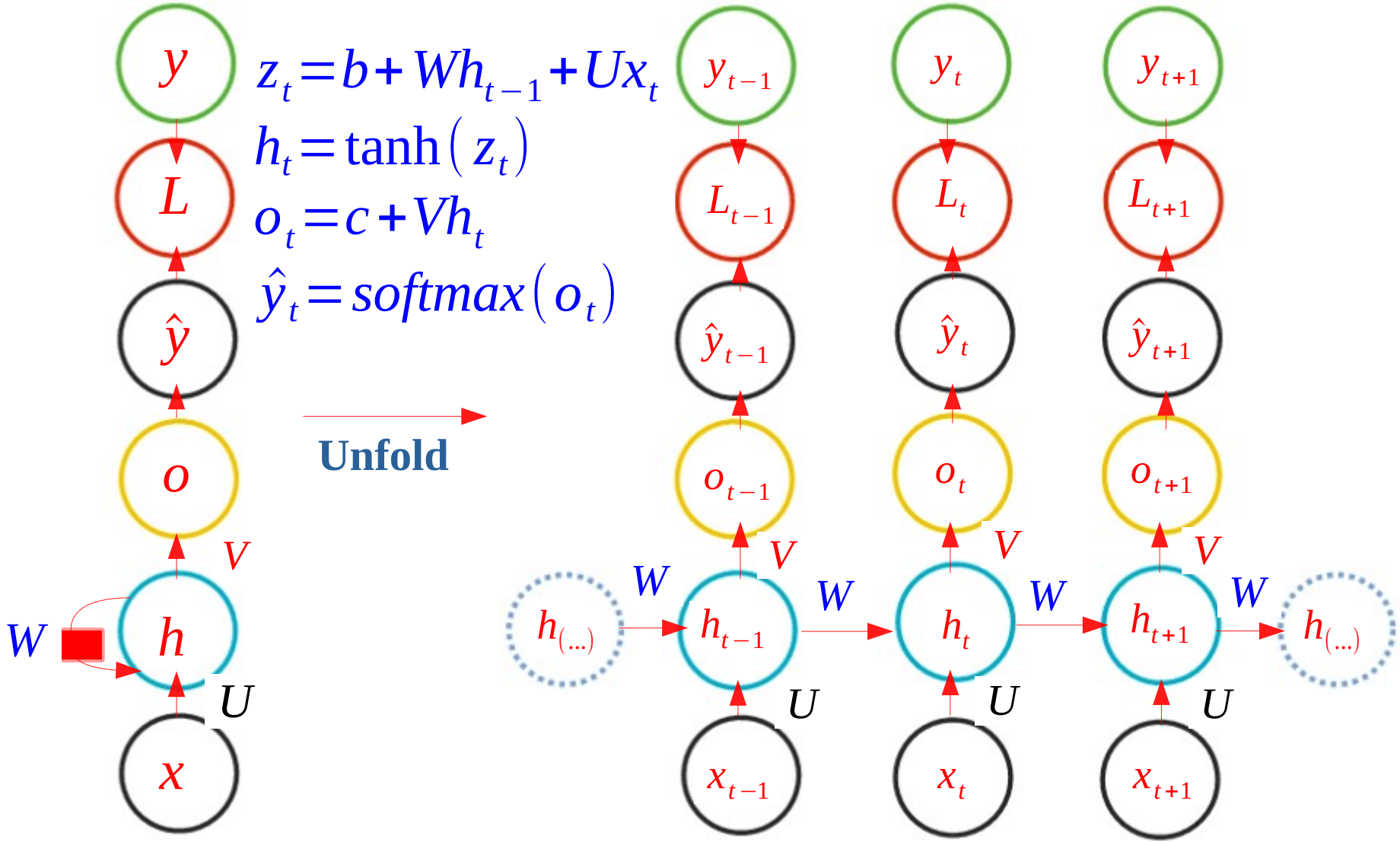
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Types of Recurrent Neural Networks



Andrej Karpathy

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



Reference: Deep Learning (Ian J. Goodfellow, Yoshua Bengio and Aaron Courville), MIT Press, 2016.

Back Propagation Through Time(BPTT)

$$V = V - \alpha \frac{\partial L}{\partial V}$$

$$W = W - \alpha \frac{\partial L}{\partial W}$$

$$U = U - \alpha \frac{\partial L}{\partial U}$$

Back Propagation Through Time(BPTT)

$$V = V - \alpha \frac{\partial L}{\partial V}$$

$$\frac{\partial L}{\partial V} = \frac{\partial L_1}{\partial V} + \frac{\partial L_2}{\partial V} + \ldots + \frac{\partial L_n}{\partial V}$$

$$\frac{\partial L}{\partial V} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial V}$$

$$W = W - \alpha \frac{\partial L}{\partial W}$$

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$U = U - \alpha \frac{\partial L}{\partial U}$$

$$\frac{\partial L}{\partial U} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial U}$$

Back Propagation Through Time(BPTT)

- The total loss is simply the sum of the loss over all time steps

$$L(\theta) = \sum_{t=1}^{T} L_t(\theta)$$

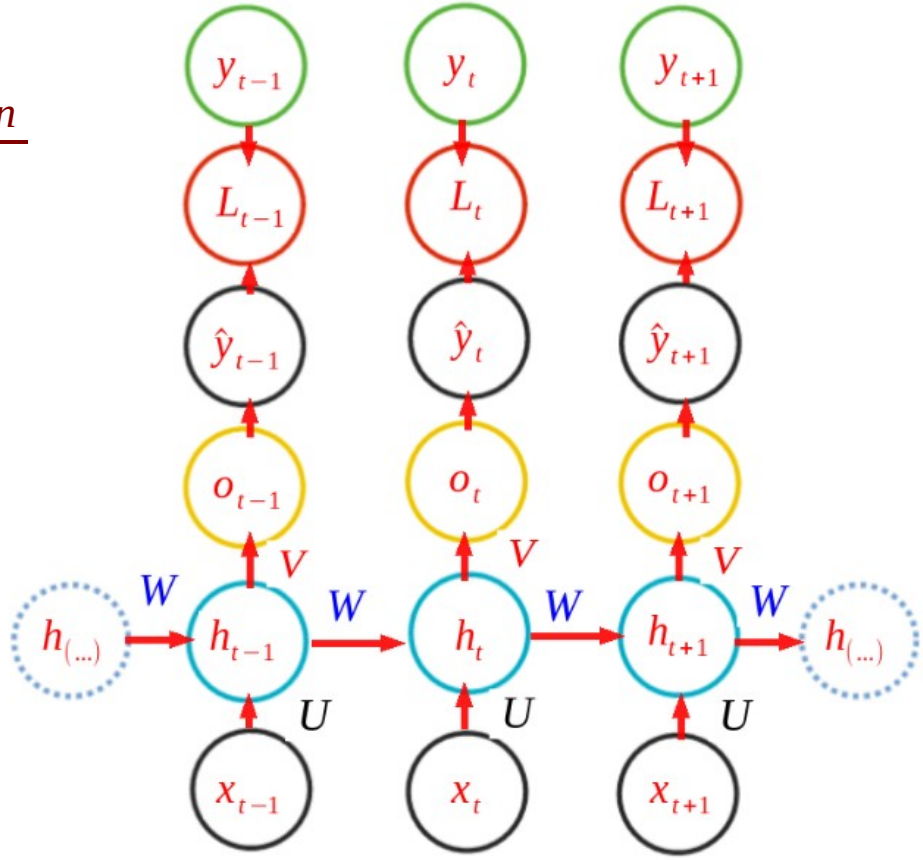
Reference: https://github.com/go2carter/nn-learn/blob/master/grad-deriv-tex/rnn-grad-deriv.pdf

BPTT - Gradient Calculations

$$\frac{\partial L}{\partial V} = \frac{\partial L_1}{\partial V} + \frac{\partial L_2}{\partial V} + \ldots + \frac{\partial L_n}{\partial V}$$

$$\frac{\partial L}{\partial V} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial V}$$

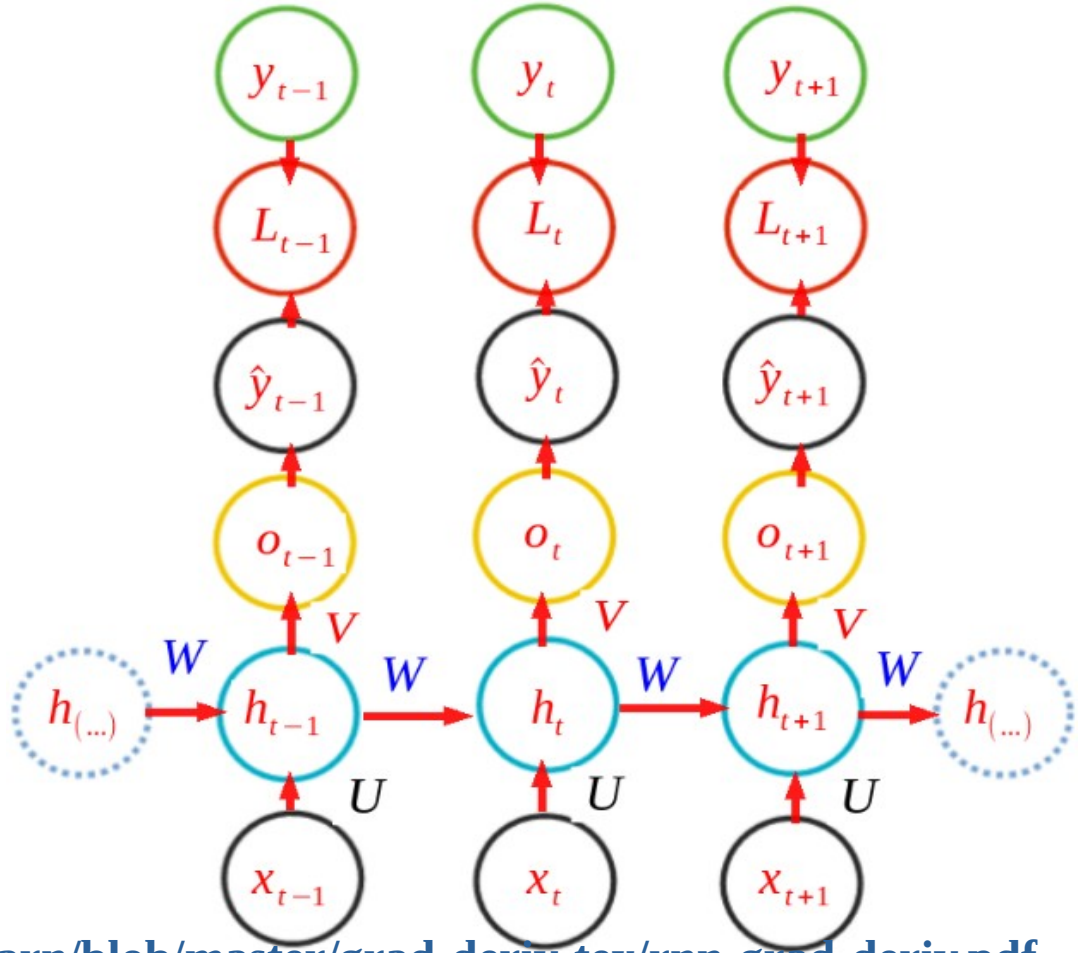
$$\frac{\partial L_t}{\partial V} = \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial o_t} \frac{\partial o_t}{\partial V}$$



Reference: https://github.com/go2carter/nn-learn/blob/master/grad-deriv-tex/rnn-grad-deriv.pdf

BPTT - Gradient Calculations

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

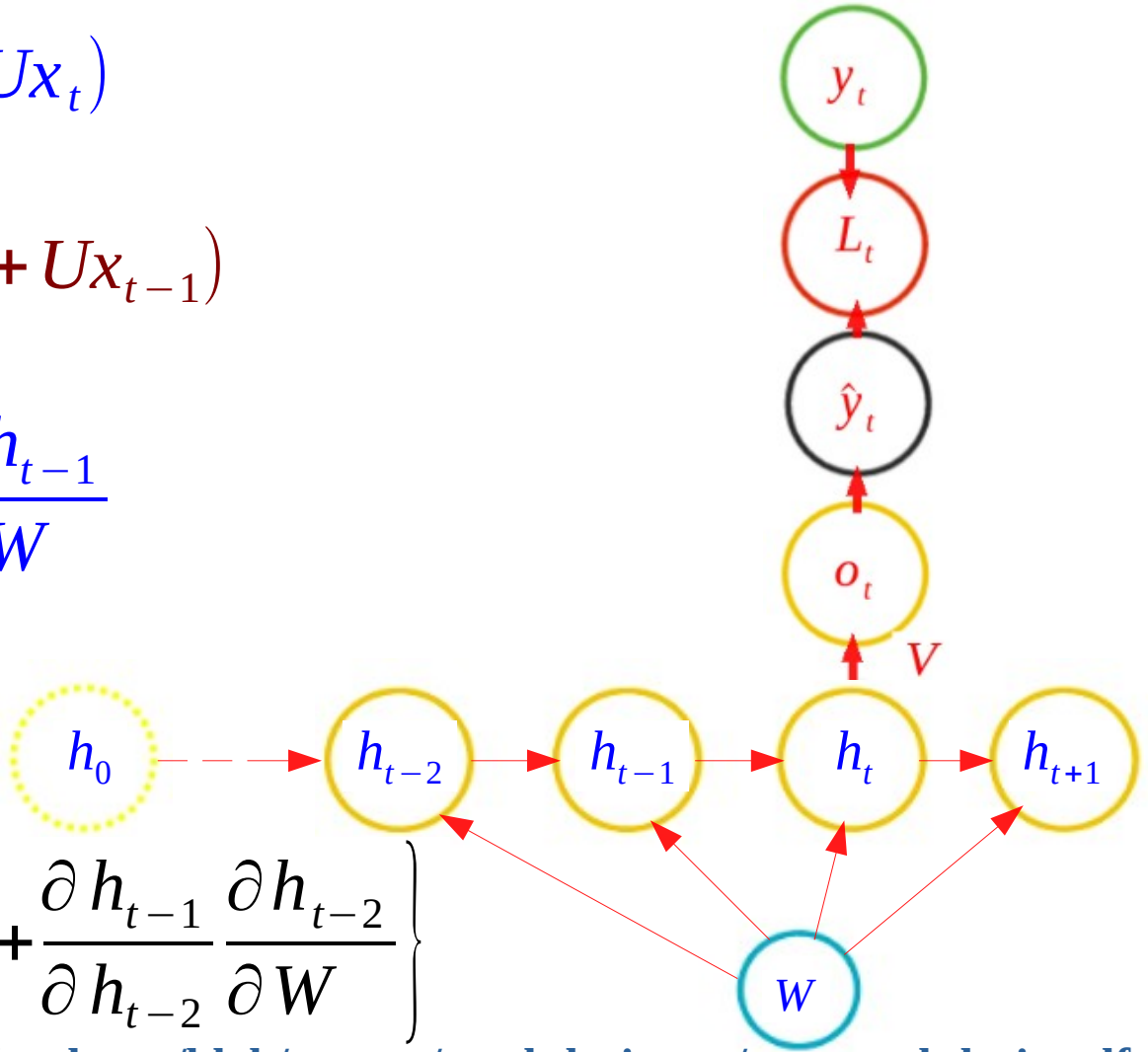


Reference: https://github.com/go2carter/nn-learn/blob/master/grad-deriv-tex/rnn-grad-deriv.pdf

$$h_t = \tanh\left(b + W h_{t-1} + U x_t\right)$$

$$h_{t-1} = \tanh\left(b + W h_{t-2} + U x_{t-1}\right)$$

$$\frac{\partial h_t}{\partial W} = \frac{\partial h_t}{\partial W} + \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W}$$



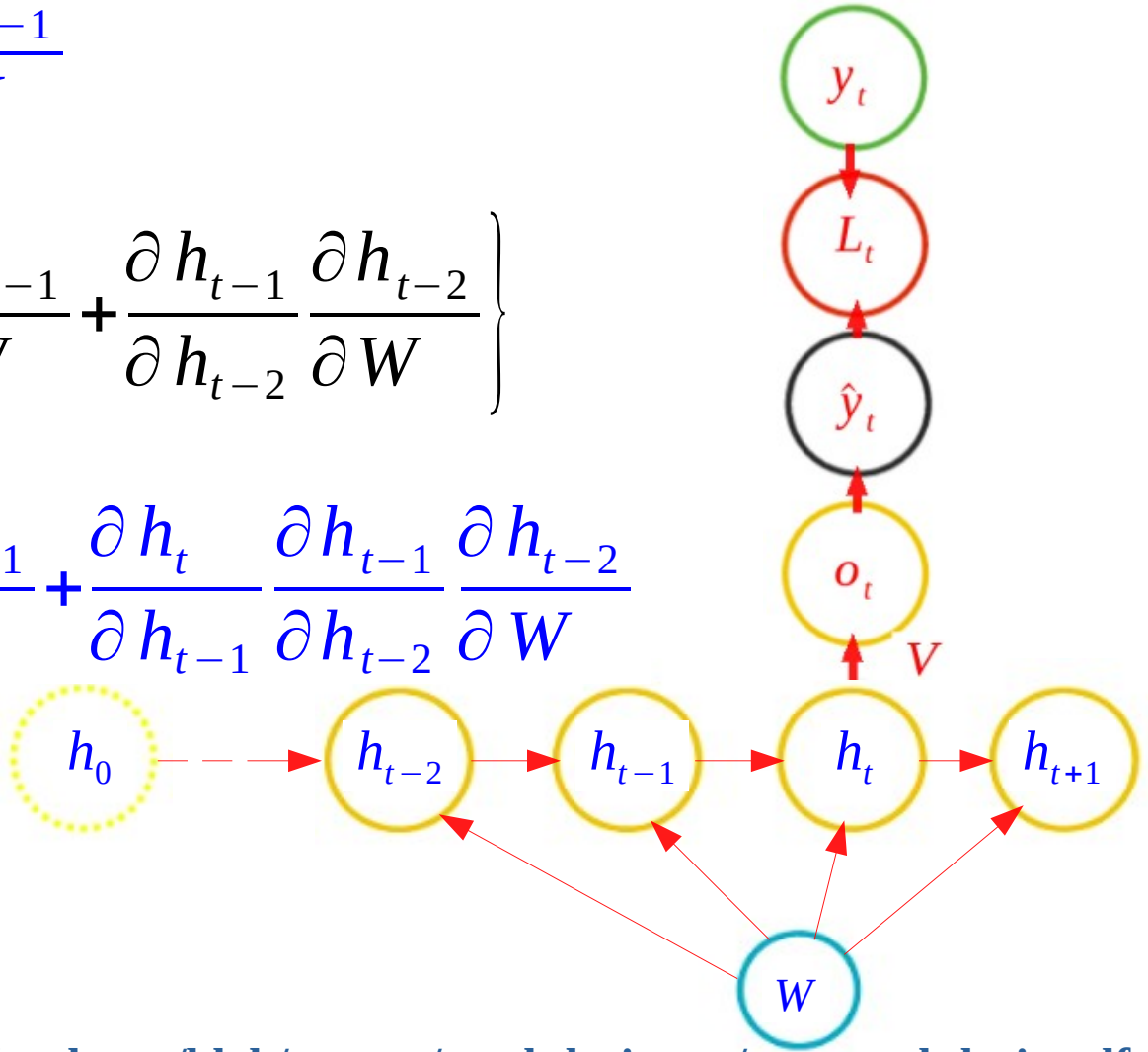
$$\frac{\partial h_t}{\partial W} = \frac{\partial h_t}{\partial W} + \frac{\partial h_t}{\partial h_{t-1}} \left\{ \frac{\partial h_{t-1}}{\partial W} + \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial W} \right\}$$

Reference: https://github.com/go2carter/nn-learn/blob/master/grad-deriv-tex/rnn-grad-deriv.pdf

$$\frac{\partial h_t}{\partial W} = \frac{\partial h_t}{\partial W} + \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W}$$

$$\frac{\partial h_t}{\partial W} = \frac{\partial h_t}{\partial W} + \frac{\partial h_t}{\partial h_{t-1}} \left\{ \frac{\partial h_{t-1}}{\partial W} + \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial W} \right\}$$

$$\frac{\partial h_t}{\partial W} = \frac{\partial h_t}{\partial W} + \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W} + \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial W}$$



Reference: https://github.com/go2carter/nn-learn/blob/master/grad-deriv-tex/rnn-grad-deriv.pdf

BPTT - Gradient Calculations

$$\frac{\partial h_t}{\partial W} = \frac{\partial h_t}{\partial W} + \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W} + \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial W}$$

$$\frac{\partial h_t}{\partial W} = \frac{\partial h_t}{\partial W} + \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W} + \frac{\partial h_t}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial W}$$

$$\frac{\partial h_t}{\partial W} = \frac{\partial h_t}{\partial h_t} \frac{\partial h_t}{\partial W} + \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W} + \frac{\partial h_t}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial W} = \sum_{r=1}^{t} \frac{\partial h_t}{\partial h_r} \frac{\partial h_r}{\partial W}$$

$$\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial o_t} \frac{\partial o_t}{\partial h_t} \sum_{r=1}^{t} \frac{\partial h_t}{\partial h_r} \frac{\partial h_r}{\partial W}$$

Reference: https://github.com/go2carter/nn-learn/blob/master/grad-deriv-tex/rnn-grad-deriv.pdf

BPTT - Gradient Calculations

- Taking the gradient of U is similar to doing it for W since they both require taking sequential derivativs of an h_t vector.

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

$$\frac{\partial L_t}{\partial U} = \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial o_t} \frac{\partial o_t}{\partial h_t} \sum_{r=1}^{t} \frac{\partial h_t}{\partial h_r} \frac{\partial h_r}{\partial U}$$

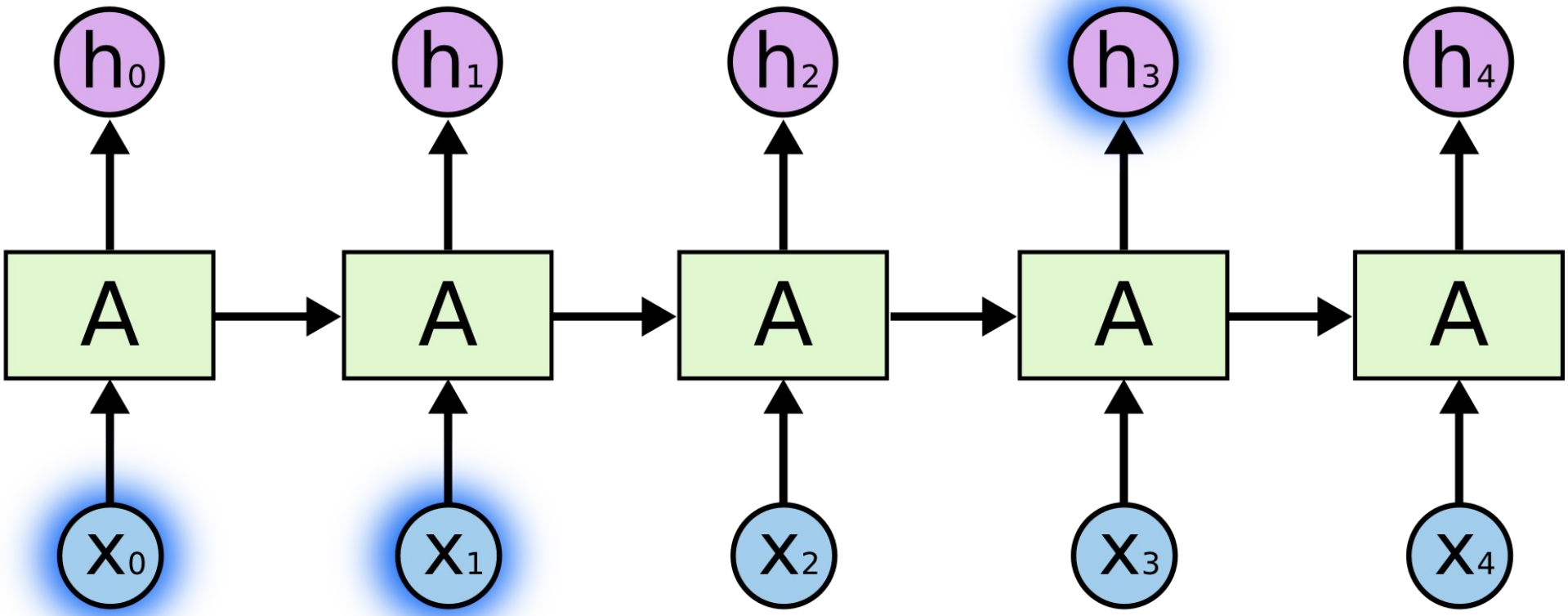
Reference: https://github.com/go2carter/nn-learn/blob/master/grad-deriv-tex/rnn-grad-deriv.pdf

The Problem of Long Term Dependencies

- One of the appeals of RNNs is the idea that they might be able to connect previous information to the present task
- previous video frames might inform the understanding of the present frame.
- But can they?

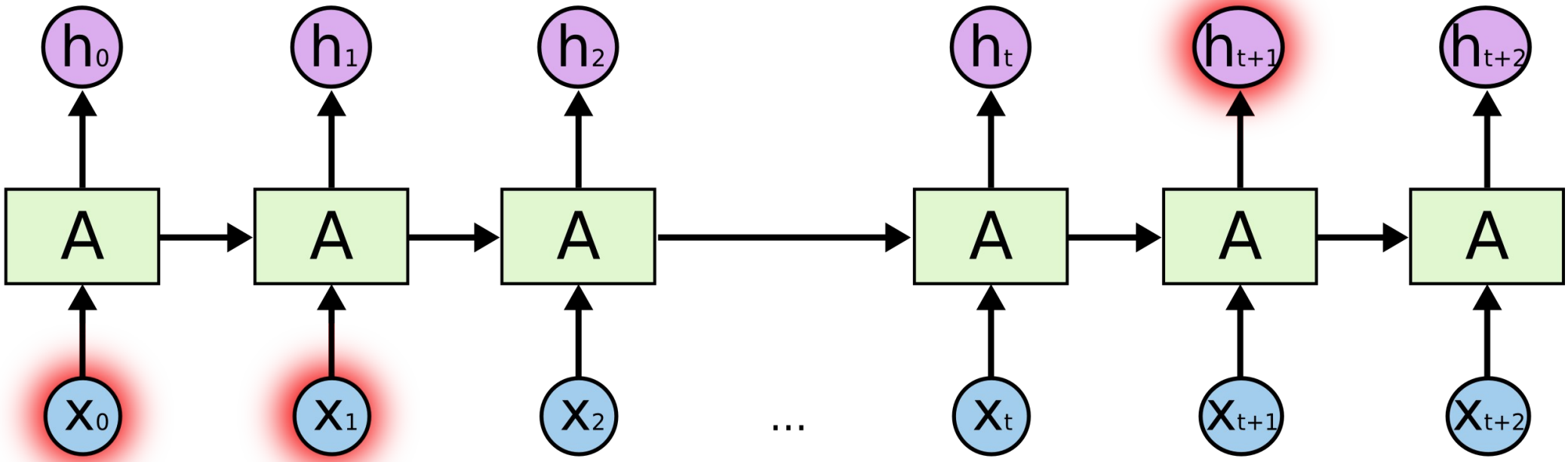
Reference: http://proceedings.mlr.press/v28/pascanu13.pdf

The Problem of Long Term Dependencies



I used to live in France and I learned how to speak.....

Vanishing/Exploding Gradient Problem in RNN



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

Difficulty of training RNNs

- There are two widely known issues with properly training recurrent neural networks with longer sequences

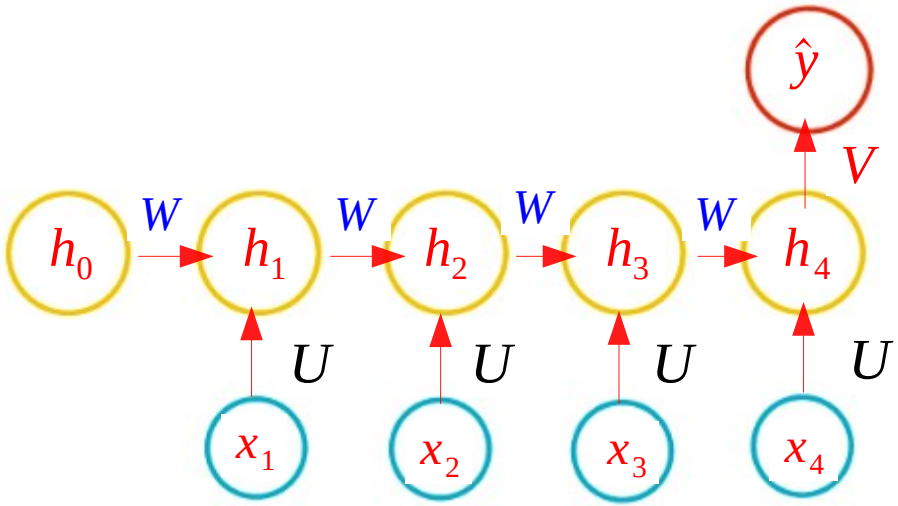
  1. **Vanishing gradient** problem
  2. **Exploding gradient** problem

- In practice many people truncate the backpropagation to a few steps.

Difficulty of training RNNs

- The network computes a single output y
- and the initial state h_0 is initialised with zeros
- The state of the network at time t is given by

$$h_t = \sigma(b + Wh_{t-1} + Ux_t)$$



An SRN "unrolled" for four time steps (t ∈ [1, 4])

- When the network computes a categorical distribution its output is given by $\hat{y} = softmax(c + Vh_t)$

Reference: https://jmlr.org/papers/volume21/18-141/18-141.pdf

Back Propagation Through Time(BPTT)

$$V = V - \alpha \frac{\partial L}{\partial V}$$

$$W = W - \alpha \frac{\partial L}{\partial W}$$

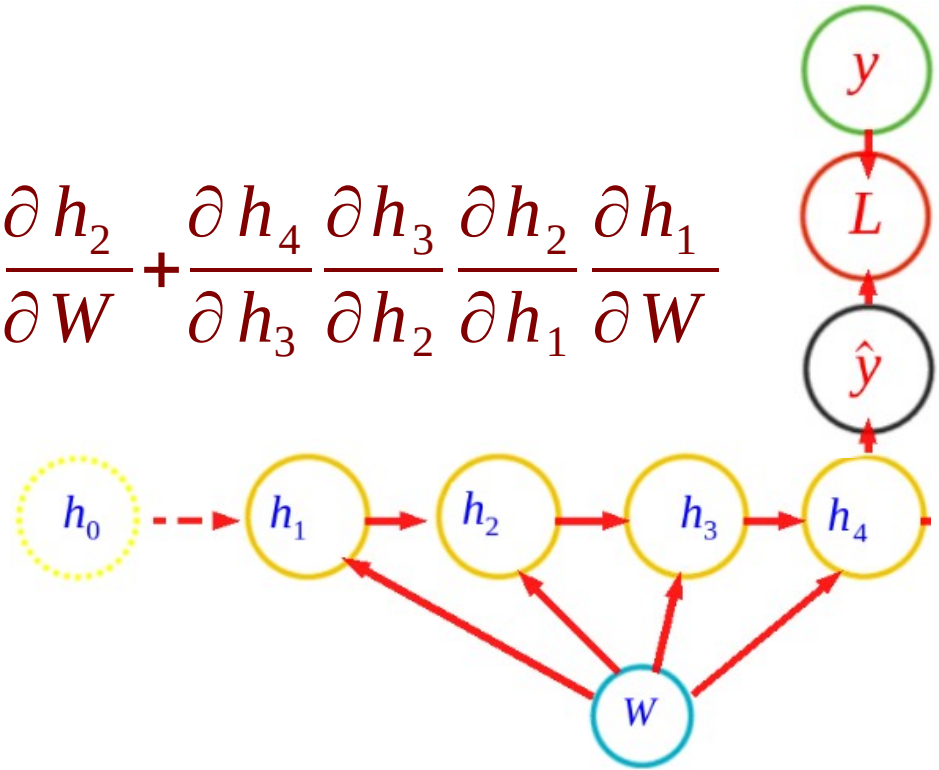
$$U = U - \alpha \frac{\partial L}{\partial U}$$

Difficulty of training RNNs

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_t} \frac{\partial h_t}{\partial W}$$

$$\frac{\partial h_4}{\partial W} = \frac{\partial h_4}{\partial W} + \frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial W} + \frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial W} + \frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial W}$$

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_t} \sum_{k=1}^{t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$



Difficulty of training RNNs

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_t} \sum_{k=1}^{t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial z_i} \frac{\partial z_i}{\partial h_{i-1}}$$

$$\frac{\partial h_i}{\partial h_{i-1}} = diag\left(\sigma'(z_i)\right) W$$

$$\frac{\partial h_4}{\partial h_1} = \frac{\partial h_4}{\partial h_3} \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1}$$

$$h_t = \sigma\left(b + W h_{t-1} + U x_t\right)$$

$$h_i = \sigma(z_i)$$

$$z_i = b + W h_{i-1} + U x_i$$

$$\frac{\partial h_i}{\partial z_i} = diag\left(\sigma'(z_i)\right)$$

Difficulty of training RNNs

$$\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\| = \left\| diag\left(\sigma'(z_i) \right) W \right\|$$

$$\leq \left\| diag\left(\sigma'(z_i) \right) \right\| \left\| W \right\|$$

$$\sigma'(z_i) \leq \frac{1}{4} = \gamma \; [if \;\; \sigma \;\; is \;\; sigmoid]$$

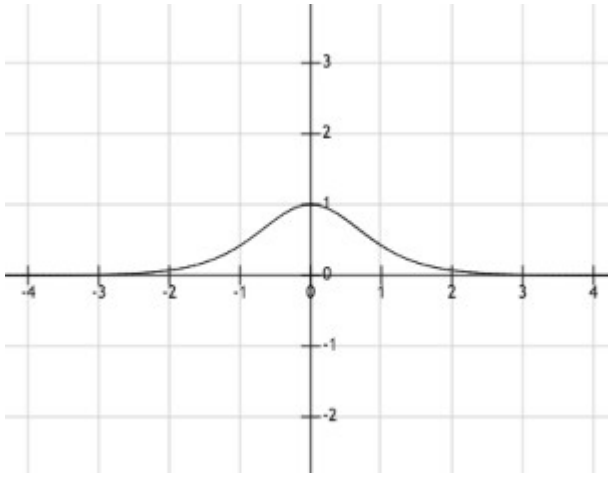
$$\sigma'(z_i) \leq 1 = \gamma \; [if \;\; \sigma \;\; is \;\; \tanh]$$

$$\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\| \leq \gamma \left\| W \right\|$$

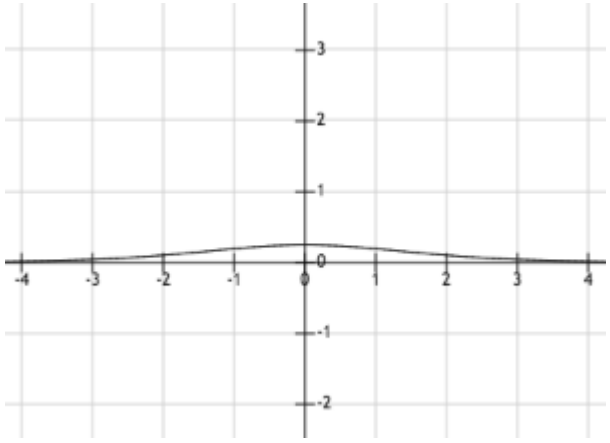
$$\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\| \leq \gamma \lambda$$

Tanh Activation Function

- Since data is centered around 0, the derivatives are higher



For input between [-1,1], we have derivative between [0.42, 1].



For input between [0,1], we have derivative between [0.20, 0.25]

Refrence: https://stats.stackexchange.com/questions/101560/tanh-activation-function-vs-sigmoid-activation-function

Difficulty of training RNNs

$$\left\| \frac{\partial h_t}{\partial h_k} \right\| = \left\| \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} \right\|$$

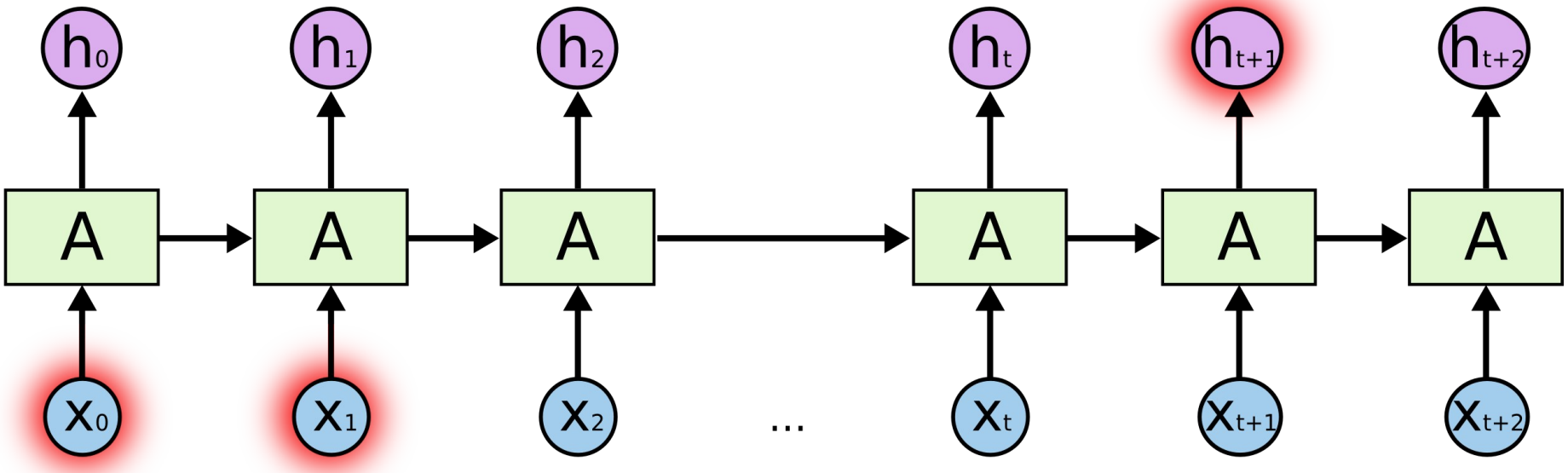
$$\leq \prod_{i=k+1}^{t} \gamma \lambda$$

$$\leq (\gamma \lambda)^{(t-k)}$$

If $(\gamma \lambda) < 1$ *then gradient will vanish*

If $(\gamma \lambda) > 1$ *then gradient will explode*

Truncated BPTT



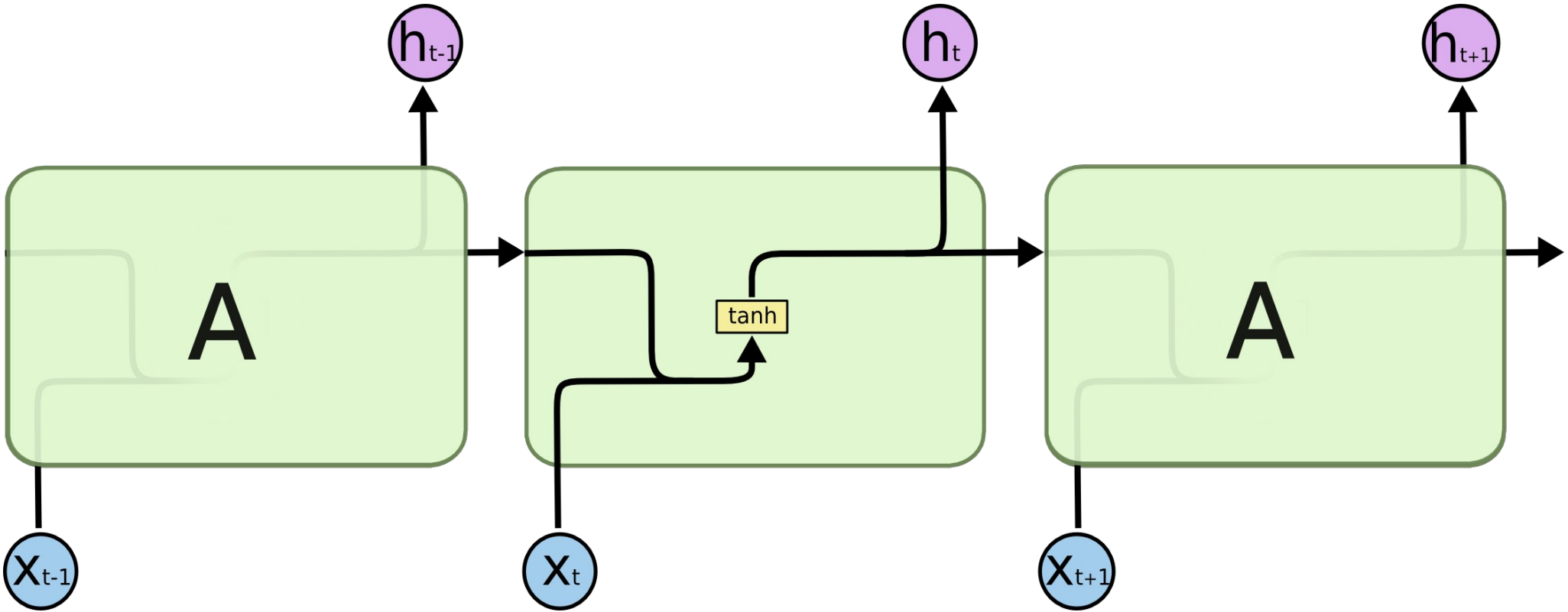
LSTM Networks

SimpleRNN

- All recurrent neural networks have the form of a chain of repeating modules of neural network.
- This lets them maintain information in '**memory**' over time
- But, it can be difficult to train standard RNNs to solve problems that require learning long-term temporal dependencies.
- This is because of **vanishing/exploding** gradient problem

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

Simple RNN

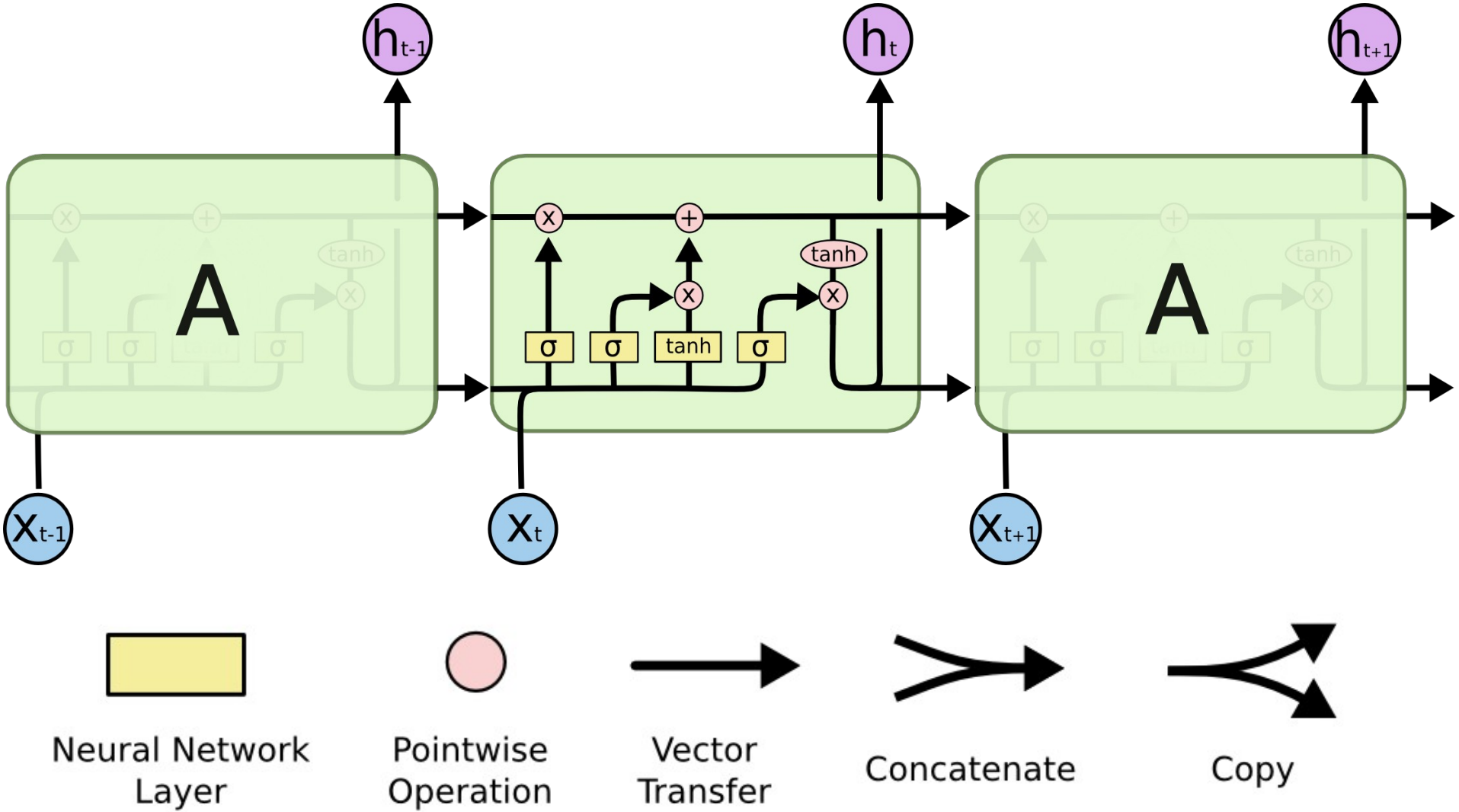


Christopher Olah https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Networks

- Long Short Term Memory networks – usually just called “LSTMs”
- Special kind of RNN
- LSTMs are explicitly designed to avoid the long-term dependency problem.
- Remembering information for long periods of time is practically their default behavior

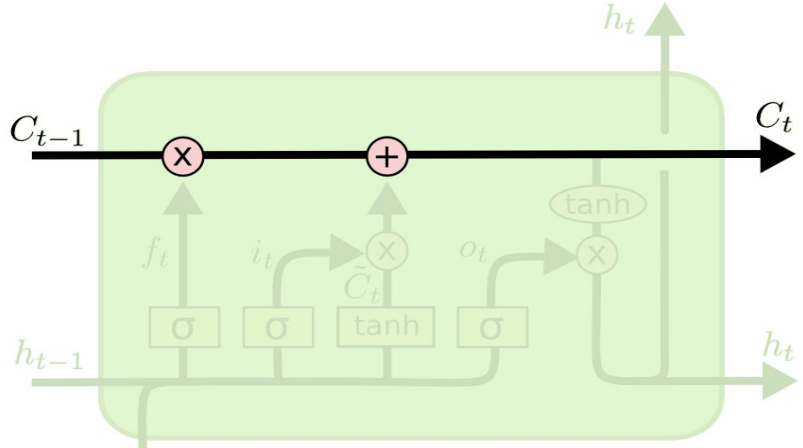
https://arxiv.org/pdf/1412.3555.pdf



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

The Core Idea Behind LSTMs

- The key to LSTMs is the **cell state(C)**, the horizontal line running through the top of the diagram.
- It runs straight down the entire chain, with only some minor linear interactions. It's very easy for **information** to just flow along it **unchanged**.



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

The Core Idea Behind LSTMs

- Unlike to the traditional recurrent unit which **overwrites its content at each time-step**
- LSTM unit is able to decide whether **to keep the existing memory** via the introduced gates.
- Intuitively, if the LSTM unit detects an important feature from an input sequence at early stage
- It easily carries this information (the existence of the feature) over a long distance
- Hence, capturing potential long-distance dependencies.

Step-by-Step LSTM Walk Through

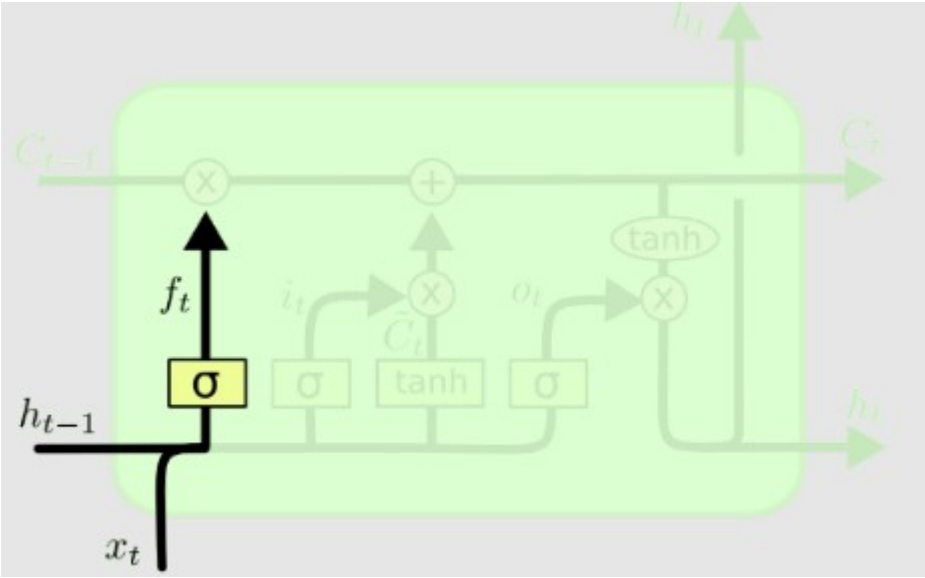
- Language model trying to predict the next word based on all the previous ones.
- The **cell state(C)** might include the **gender of the present subject**, so that the correct pronouns can be used.
- When we see a new subject, we want to forget the gender of the old subject.

Forget Gate

- The first step in our LSTM is to decide what information we're going to throw away from the cell state(C).
- This decision is made by a sigmoid layer called the "forget gate layer."
- It looks at h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}.
- **1** represents "**completely keep this**"
- **0** represents "**completely get rid of this**."

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

Forget Gate



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

$$h_{t-1} = [1,2,3]$$

$$x_t = [4,5,6]$$

$$[h_{t-1}, x_t] = [1,2,3,4,5,6]$$

if we have 3 units

3 cell states w_f = 3*6= 3(3+3)

$$w_f = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -1 \\ 5 & 6 & 7 & 8 & 9 & 10 \\ 3 & 4 & 5 & 6 & 7 & 8 \end{bmatrix}$$

$$b_f = [1,2,3]$$

Reference: https://datascience.stackexchange.com/questions/19196/forget-layer-in-a-recurrent-neural-network-rnn/19269#19269

Forget Gate

C_{t-1}

5	5	5

0	1	1

f_t

σ

-5	177	136

\+ 1 2 3

-6	175	133

h_{t-1}

1	2	3

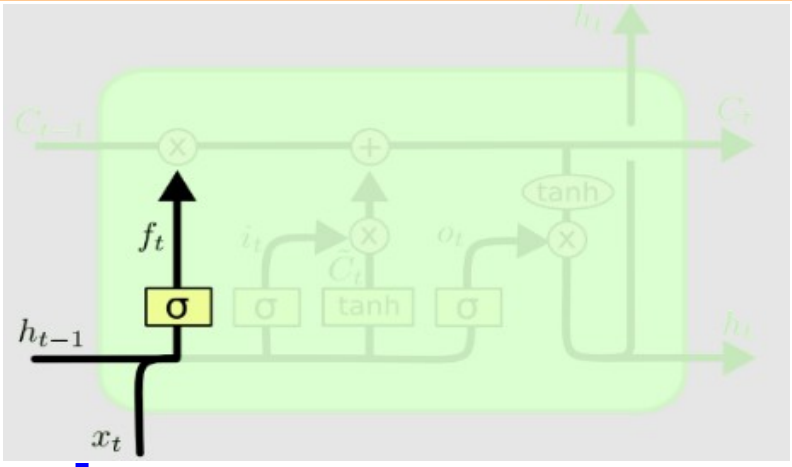
1	2	3	4	5	6

4	5	6

x_t

$b_f = [1,2,3]$

$$w_f = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -1 \\ 5 & 6 & 7 & 8 & 9 & 10 \\ 3 & 4 & 5 & 6 & 7 & 8 \end{bmatrix}$$



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

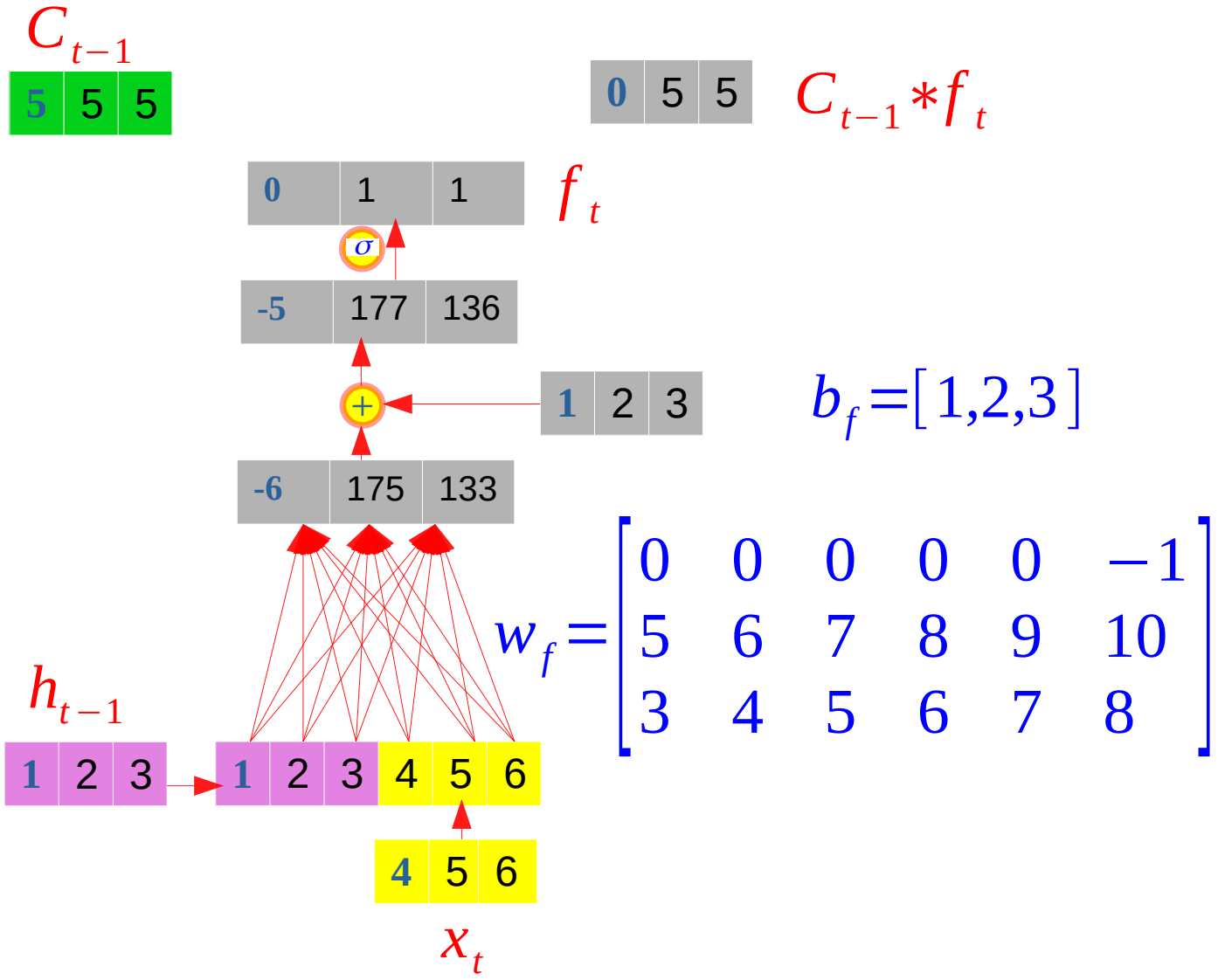
Forget Gate

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

$$w_f.[h_{t-1}, x_t] = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -1 \\ 5 & 6 & 7 & 8 & 9 & 10 \\ 3 & 4 & 5 & 6 & 7 & 8 \end{bmatrix} . \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix} = \begin{bmatrix} -6 \\ 175 \\ 133 \end{bmatrix}$$

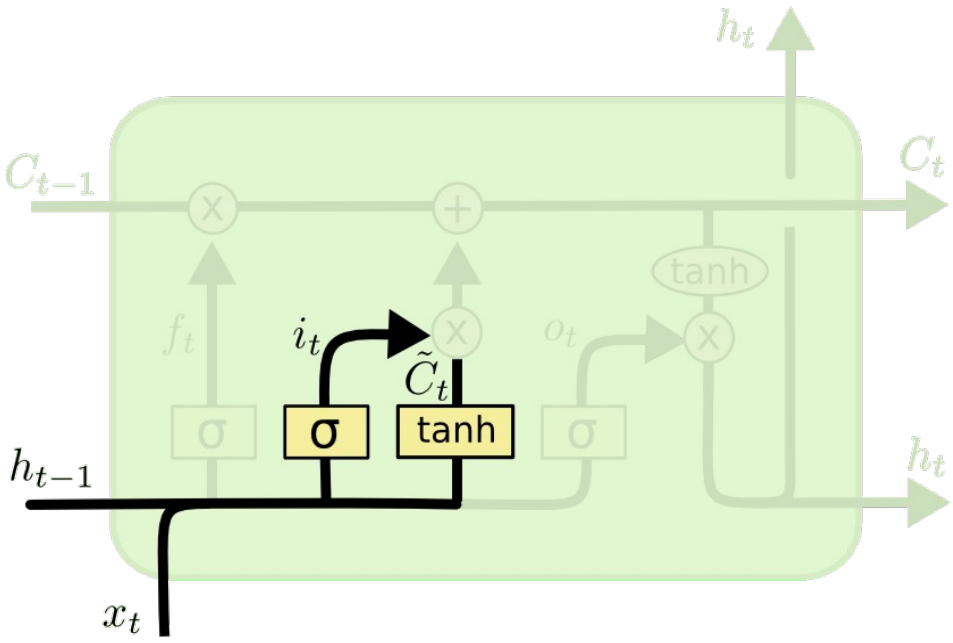
$$w_f.[h_{t-1}, x_t] + b_f = \begin{bmatrix} -6 \\ 175 \\ 133 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} -5 \\ 177 \\ 136 \end{bmatrix} \quad f_t = \sigma\left(w_f.[h_{t-1}, x_t] + b_f\right) = \sigma\left(\begin{bmatrix} -5 \\ 177 \\ 136 \end{bmatrix}\right) = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

Reference: https://datascience.stackexchange.com/questions/19196/forget-layer-in-a-recurrent-neural-network-rnn/19269#19269



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

Input Gate

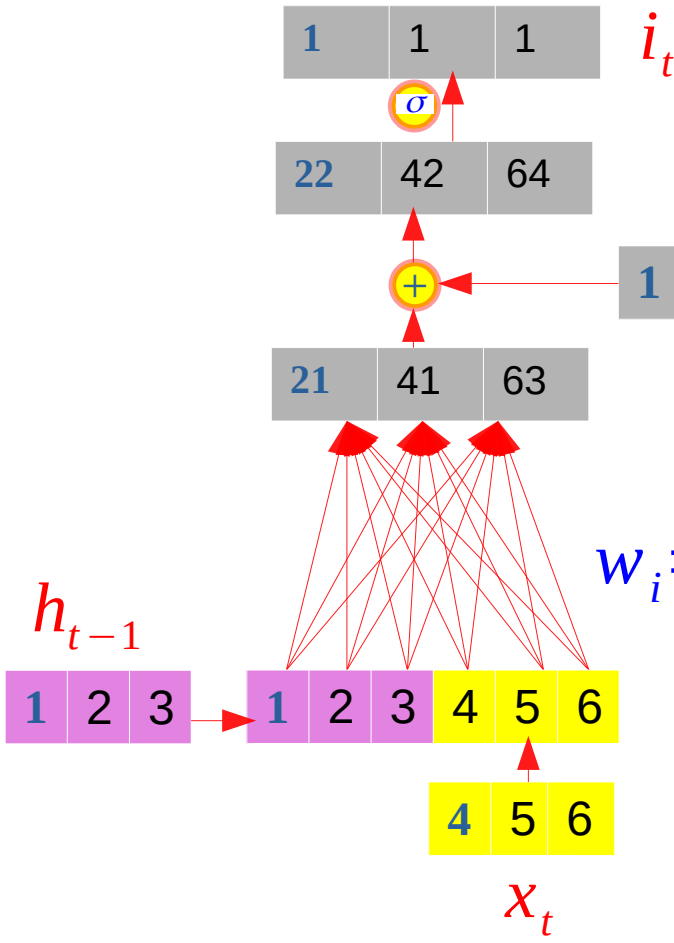


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

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

Input Gate – Sigmoid Layer



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

$$b_i = [1,1,1]$$

$$w_i = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 \end{bmatrix}$$

Input Gate – Sigmoid Layer

$$w_i = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 \end{bmatrix} \qquad b_i = [1,1,1]$$

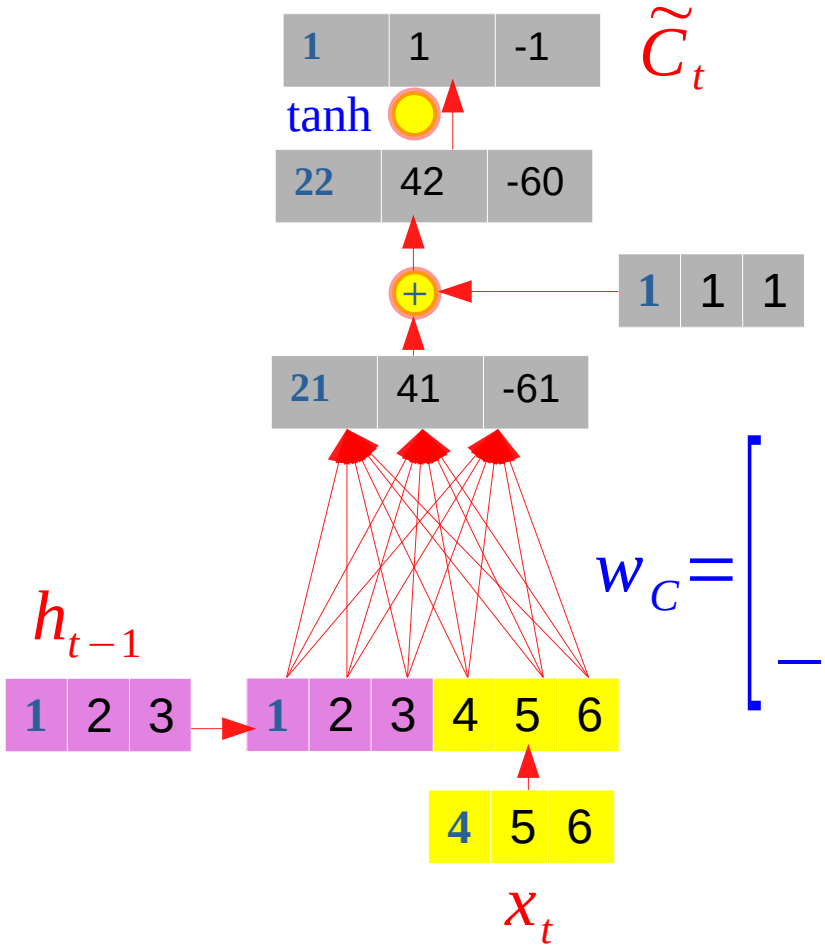
$$w_i . [h_{t-1}, x_t] + b_i = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 \end{bmatrix} . \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 22 \\ 42 \\ 64 \end{bmatrix}$$

$$i_t = \sigma(w_i . [h_{t-1}, x_t] + b_i) = \sigma\left(\begin{bmatrix} 22 \\ 42 \\ 64 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

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

Reference: https://statisticalinterference.wordpress.com/2017/05/27/lstms-in-excruciating-detail/

Input Gate – Tanh Layer



$$\tilde{C}_t = \tanh\left(w_C . [h_{t-1}, x_t] + b_C\right)$$

$$b_C = [1,1,1]$$

$$w_C = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ -3 & -3 & -3 & -3 & -3 & -3 \end{bmatrix}$$

Input Gate – Tanh Layer

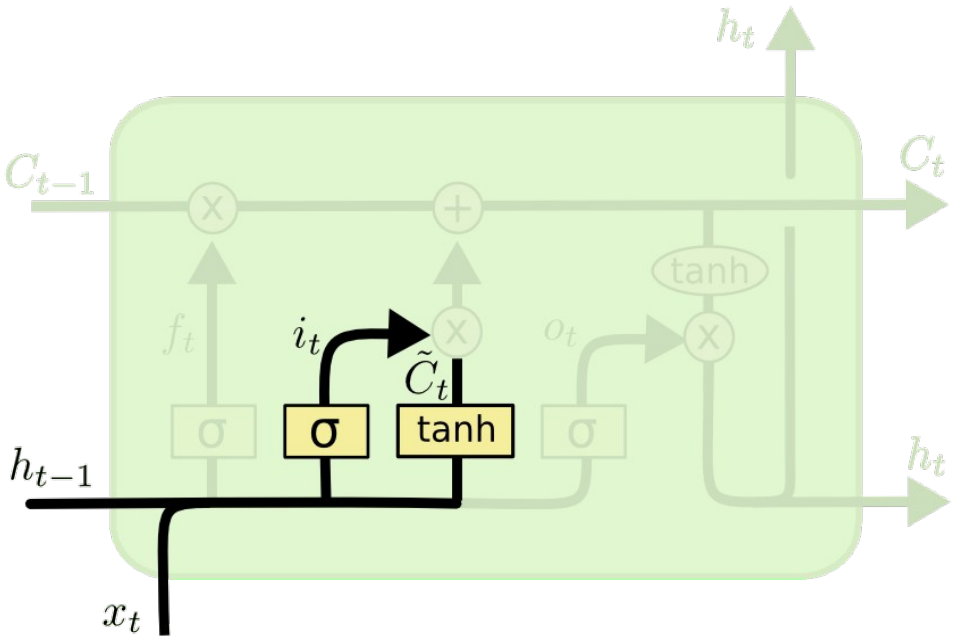
$$w_C = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 \end{bmatrix} \quad b_C = [1,1,1] \quad \widetilde{C}_t = \tanh\left(w_C . [h_{t-1}, x_t] + b_C\right)$$

$$w_C . [h_{t-1}, x_t] + b_C = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ -3 & -3 & -3 & -3 & -3 & -3 \end{bmatrix} . \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 22 \\ 42 \\ -62 \end{bmatrix}$$

$$\widetilde{C}_t = \tanh\left(w_C . [h_{t-1}, x_t] + b_C\right) = \tanh\left(\begin{bmatrix} 22 \\ 42 \\ -62 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$$

Reference: https://statisticalinterference.wordpress.com/2017/05/27/lstms-in-excruciating-detail/

Input Gate

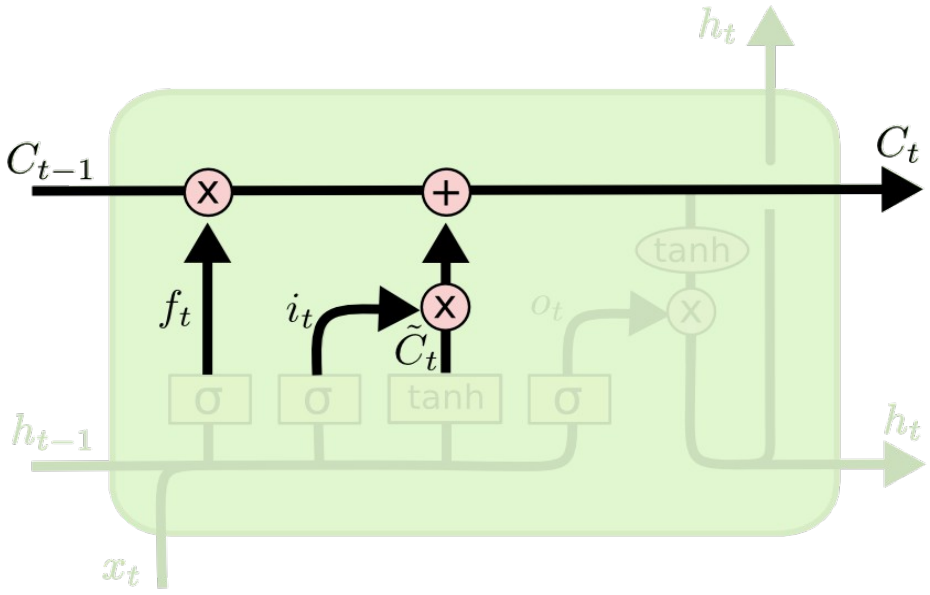


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

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

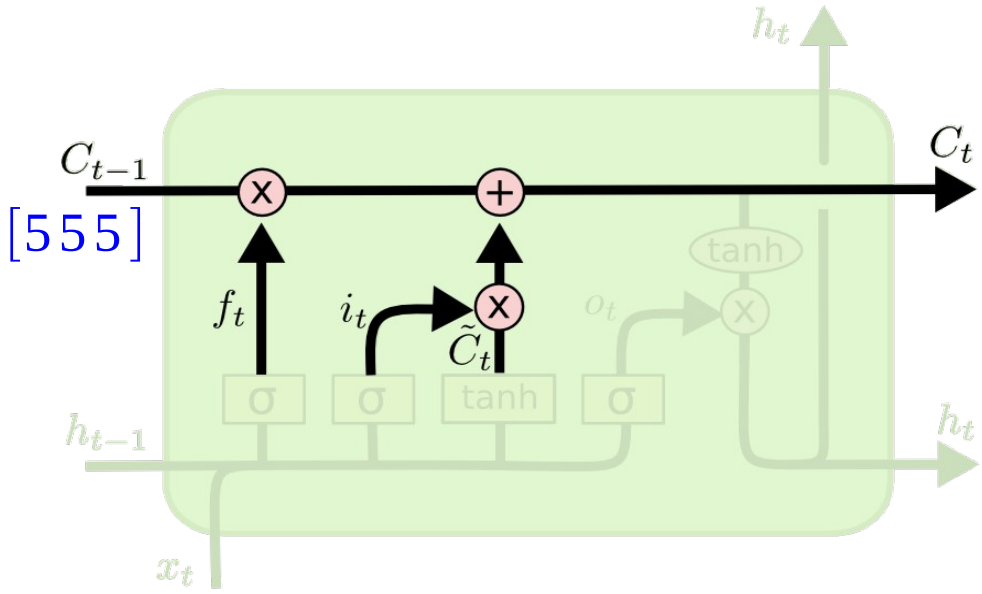
Input Gate – Update Cell Sate



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

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

Input Gate – Update Cell Sate



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

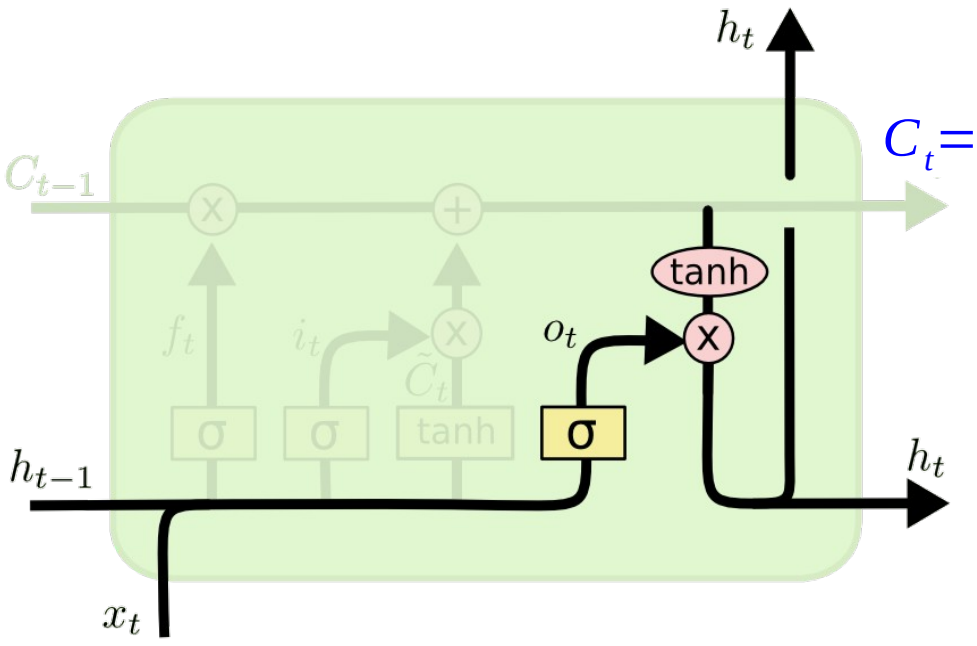
$$C_t = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} * \begin{bmatrix} 5 \\ 5 \\ 5 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 6 \\ 4 \end{bmatrix}$$

Reference: https://statisticalinterference.wordpress.com/2017/05/27/lstms-in-excruciating-detail/

Output Gate

- Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version.
- First, we run a sigmoid layer which decides what parts of the cell state we're going to output.
- Then, we put the cell state through tanh (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

Output Gate



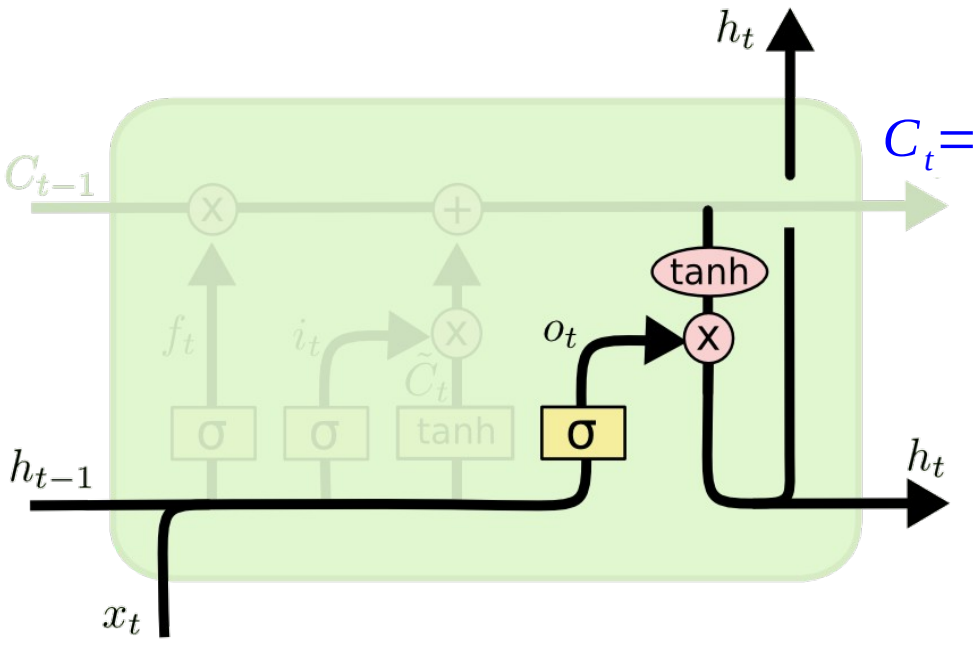
C_t=[1 6 4]

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

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

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

Output Gate



C_t=[1 6 4]

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

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

result of tanh(C_t) is [0.76,0.9999,0.9993]

Suppose O_t = [0,0.5,1]

h_t = [(0*0.76), (0.5*0.99), (1*0.99)]= [0, 0.495,0.99]

Reference: https://statisticalinterference.wordpress.com/2017/05/27/lstms-in-excruciating-detail/

Output Gate

- For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case that’s what is coming next.
- For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that’s what follows next.

Step-by-Step LSTM Walk Through

- **Hidden state(h)** is overall state of **what we have seen so far.**
- **Cell state(C) is selective memory of the past.**
- Both these states are trainable with data.

LSTM Summary

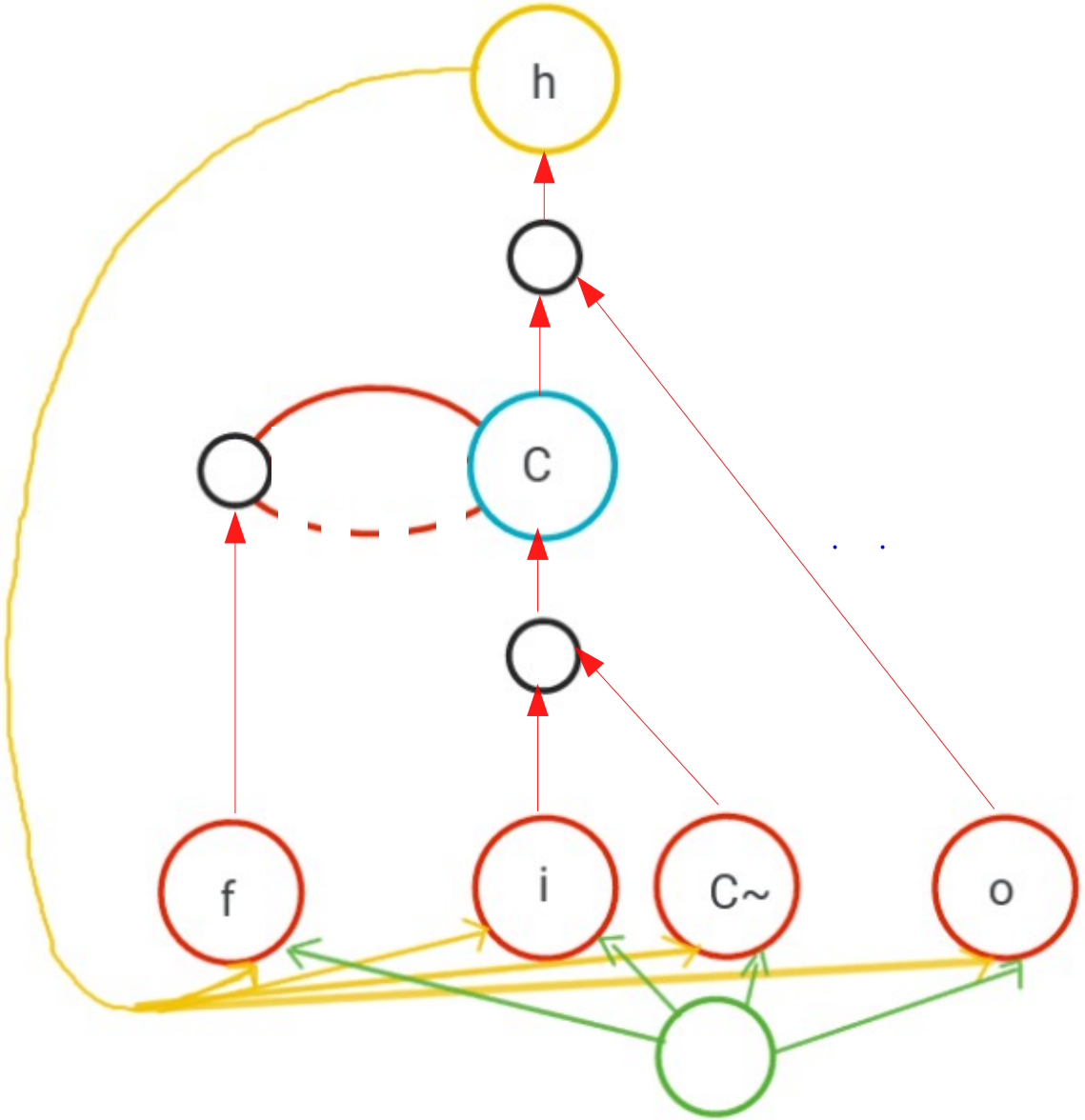
- The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.
- The most prominent feature LSTM is the additive component of its update from t-1 to t, which is lacking in the traditional recurrent unit.
- The traditional recurrent unit always replaces the activation, or the content of a unit with a new value computed from the current input and the previous hidden state.
- On the other hand LSTM unit keep the existing content and add the new content on top of it

LSTM Summary

- This additive nature has two advantages.
- **First**, it is easy for each unit to remember the existence of a specific feature in the input stream for a long series of steps.
- Any important feature will not be overwritten but be maintained as it is.
- **Second**, and perhaps more importantly, this addition effectively creates shortcut paths that bypass multiple temporal steps.
- Because of these shortcuts vanishing gradients problem will not occur

LSTM Summary

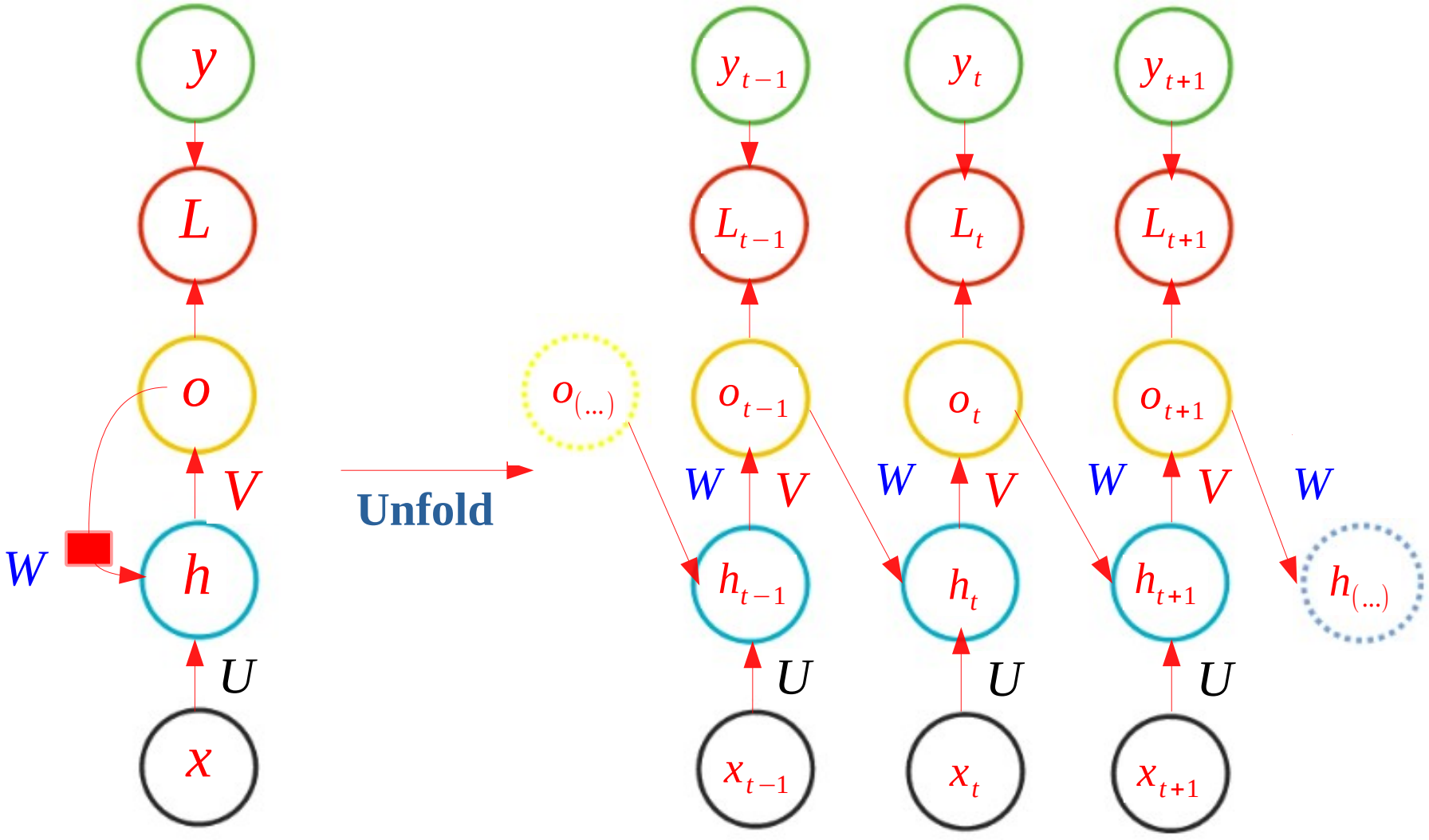
- This additive nature has two advantages.
- **First**, it is easy for each unit to remember the existence of a specific feature in the input stream for a long series of steps.
- Any important feature will not be overwritten but be maintained as it is.
- **Second**, and perhaps more importantly, this addition effectively creates shortcut paths that bypass multiple temporal steps.
- These shortcuts allow the error to be back-propagated easily without too quickly vanishing (if the gating unit is nearly saturated at 1) as a result of passing through multiple, bounded nonlinearities, thus reducing the difficulty due to vanishing gradients



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

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

$$f_t = \sigma(w_f . [h_{t-1}, x_t] + b_f)$$


y
L
o
V
W
h
U
x
Unfold
y_{t-1}
y_t
y_{t+1}
L_{t-1}
L_t
L_{t+1}
$o_{(...)}$
o_{t-1}
o_t
o_{t+1}
W
V
W
V
W
V
W
h_{t-1}
h_t
h_{t+1}
$h_{(...)}$
U
U
U
x_{t-1}
x_t
x_{t+1}