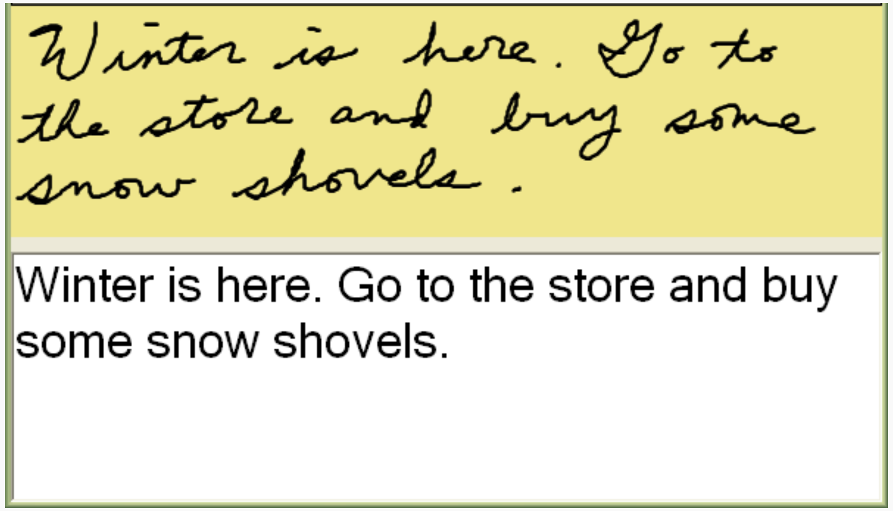
Lecture 10: Recurrent Neural Networks

CS109B Data Science 2

Pavlos Protopapas and Mark Glickman

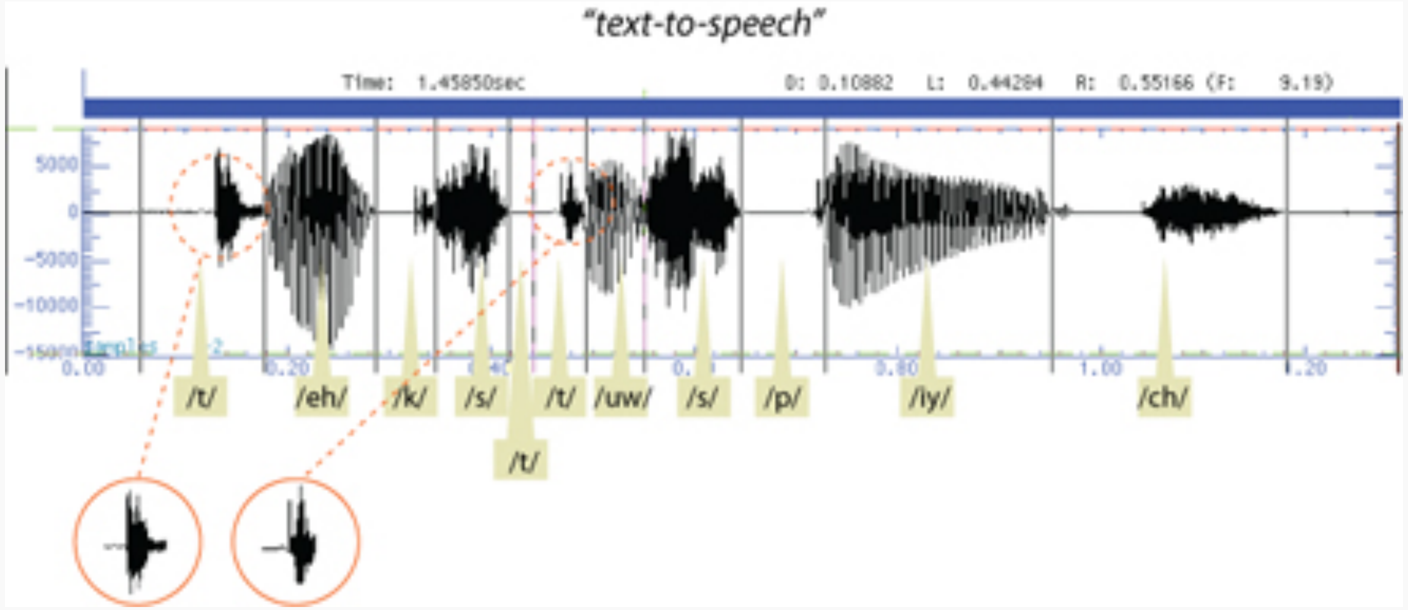
Sequence Modeling: Handwritten Text Translation



- Input : Image
- Output: Text

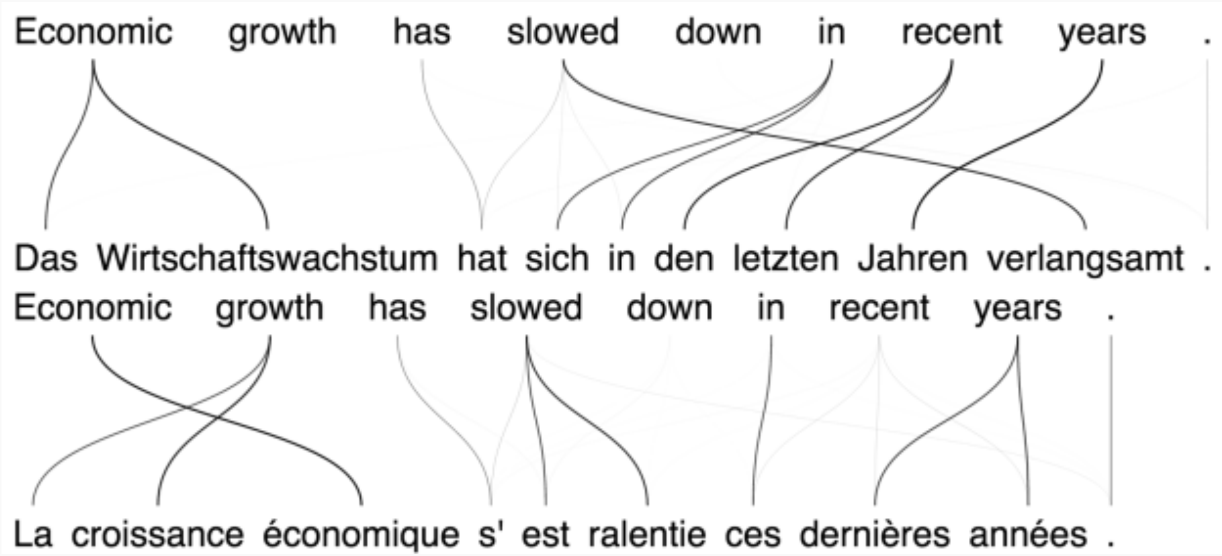
https://towardsdatascience.com/build-a-handwritten-text-recognition-system-using-tensorflow-2326a3487cd5

Sequence Modeling: Text-to-Speech



- Input : Audio
- Output: Text

Sequence Modeling: Machine Translation



- Input : Text
- Output: Translated Text

Rapping-neural-network

https://github.com/robbiebarrat/rapping-neural-network

Outline

Why RNNs

Main Concept of RNNs

More Details of RNNs

RNN training

Gated RNN

Outline

Why RNNs

Main Concept of RNNs

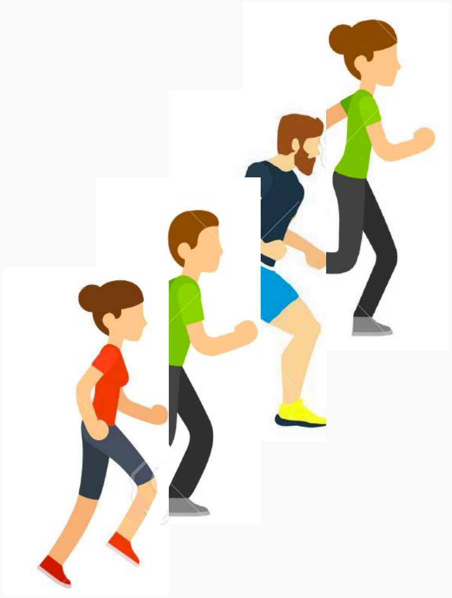
More Details of RNNs

RNN training

Gated RNN

What can my NN do?

Training: Present to the NN examples and learn from them.

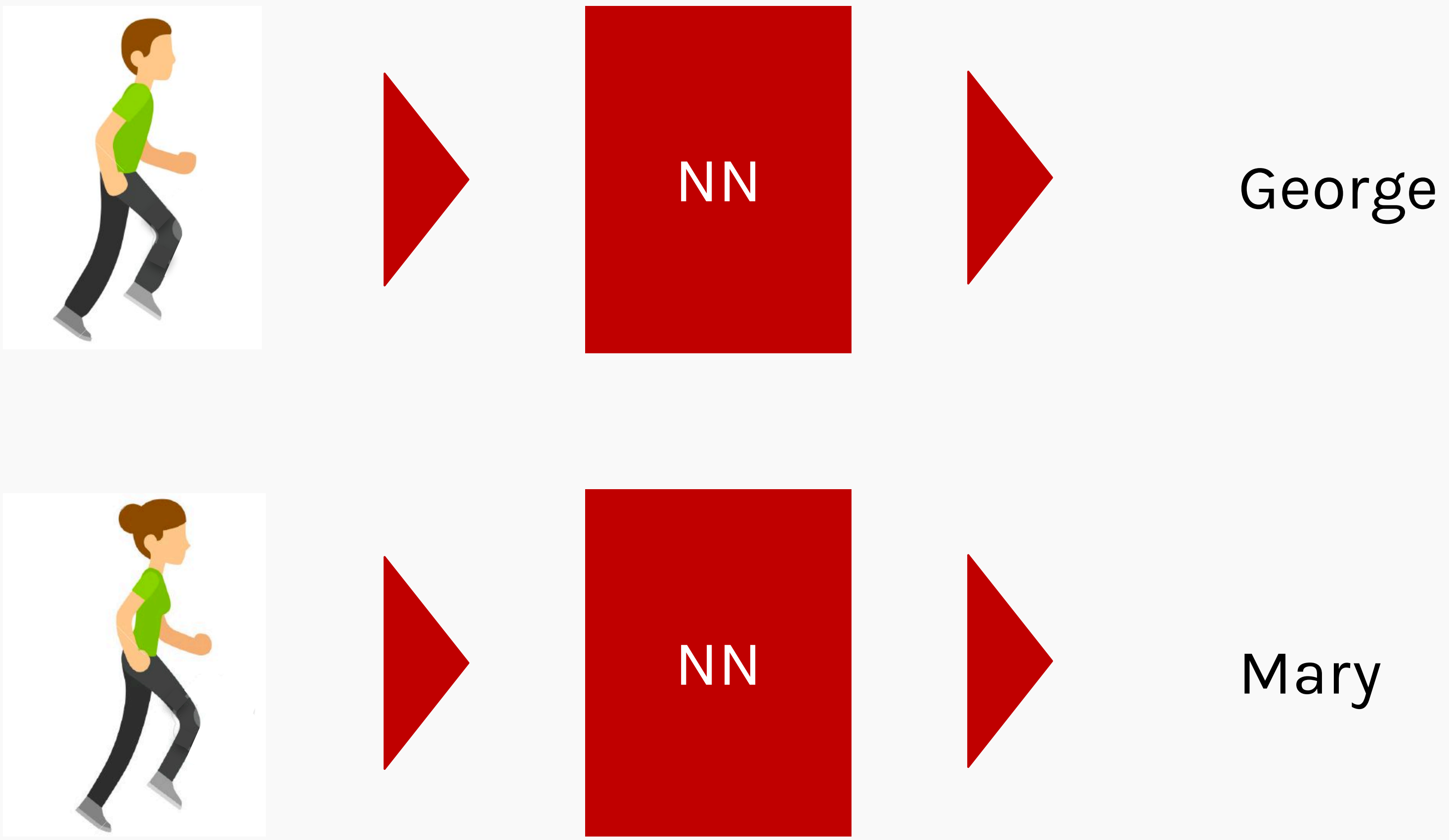


NN

[George, Mary, Tom, Suzie]

What can my NN do?

Prediction: Given an example

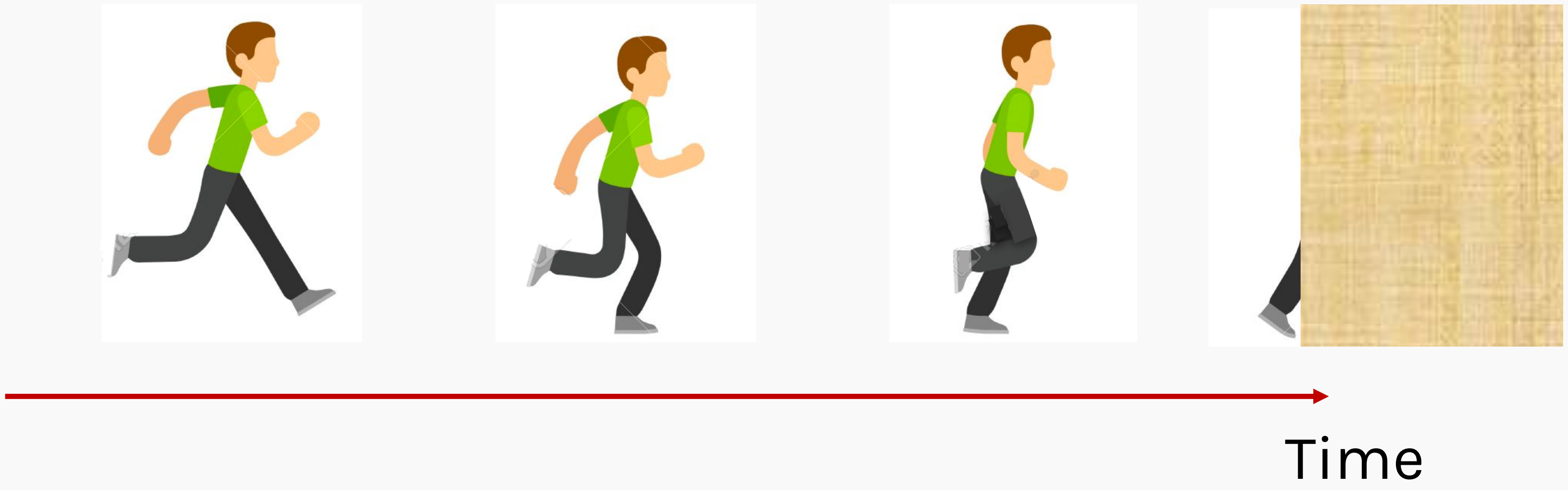


What my NN can NOT do?

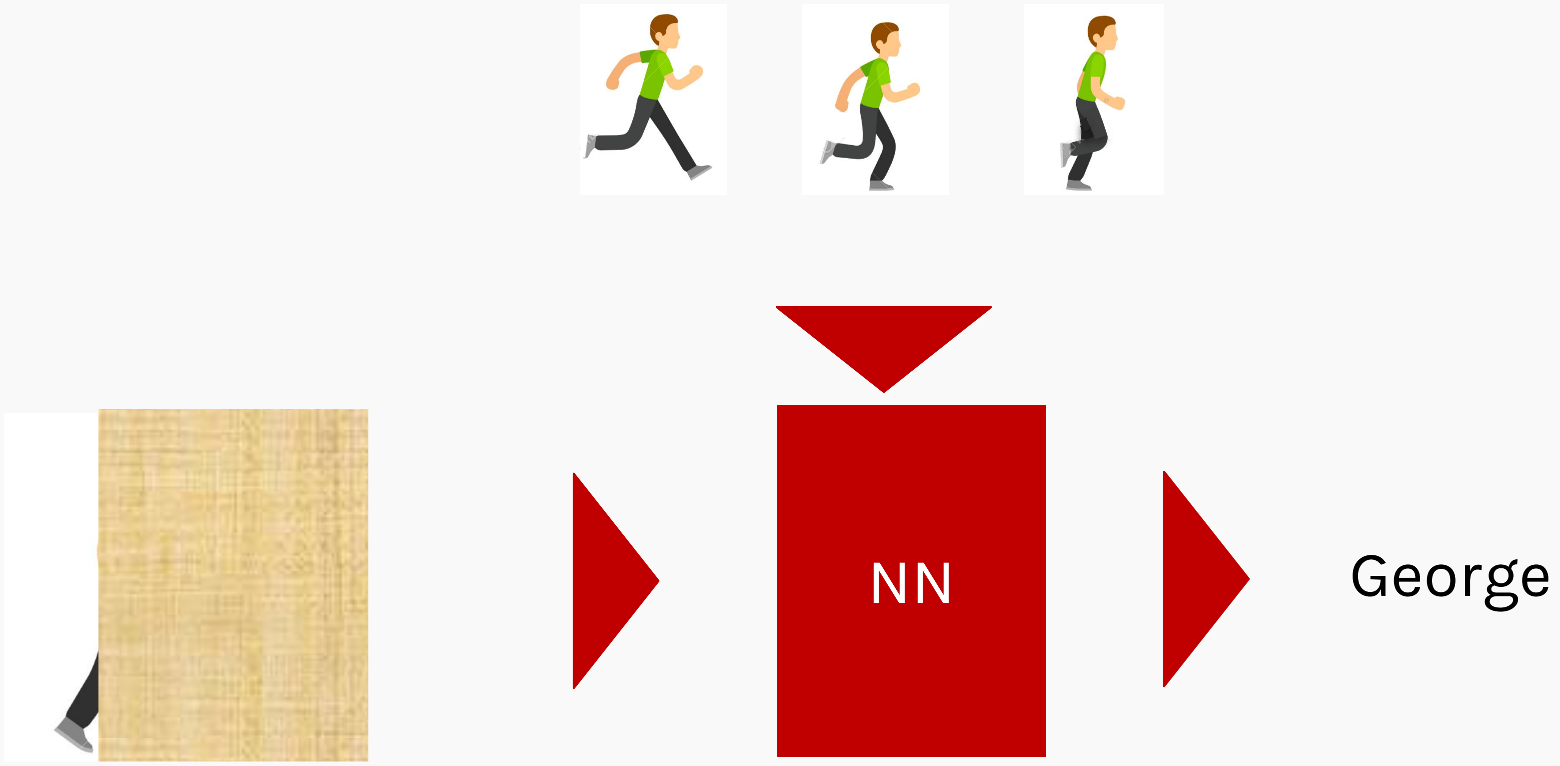
WHO IS IT?

?

Learn from previous examples

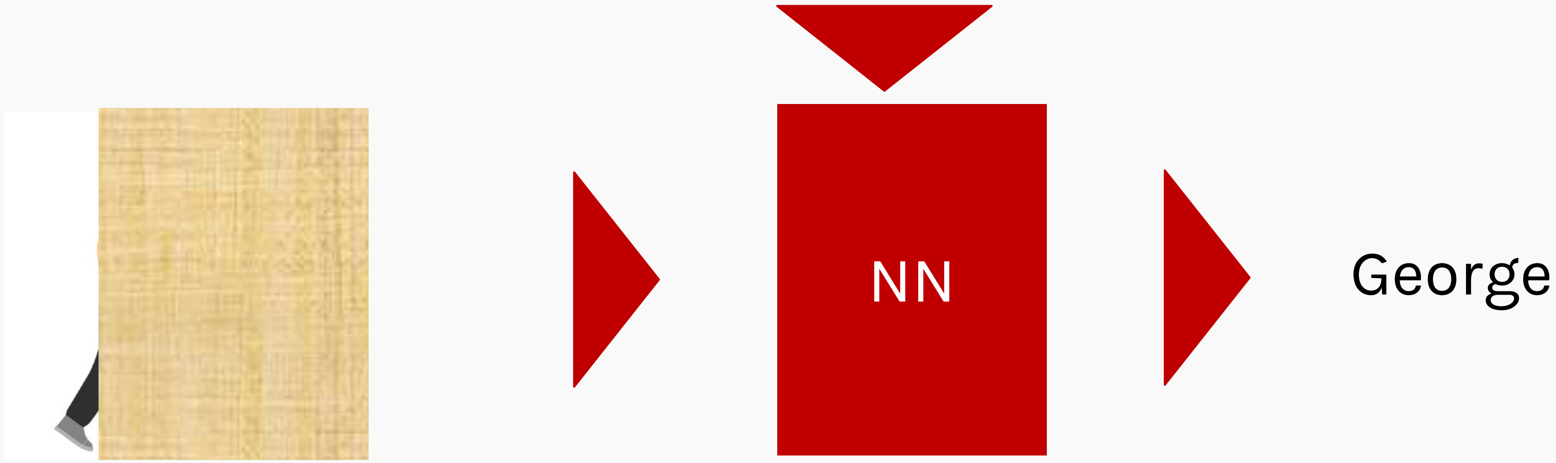


Recurrent Neural Network (RNN)



Recurrent Neural Network (RNN)

I have seen George moving in this way before.



RNNs recognize the data's sequential characteristics and use patterns to predict the next likely scenario.

Recurrent Neural Network (RNN)

He told me I could have it

WHO IS HE?

I do not know. I need to know who said that and what he said before. Can you tell me more?

Our model requires context - or contextual information - to understand the subject (he) and the direct object (it) in the sentence.

RNN – Another Example with Text

- Hellen: Nice sweater Joe.

- Joe: Thanks, Hellen. It used to belong to my brother and ***he told me I could have it****.*

WHO IS HE?

I see what you mean now! The noun “he” stands for Joe’s brother while ”it” for the sweater.

After providing sequential information, the model understood the subject (Joe’s brother) and the direct object (sweater) in the sentence .

Sequences

- We want a machine learning model to understand sequences, not isolated samples.
- Can MLP do this?
- Assume we have a sequence of temperature measurements and we want to take 3 sequential measurements and predict the next one

samples

	features
1	35
2	32
3	45
4	48
5	41
6	39
7	36
…	…

Sequences

- We want a machine learning model to understand sequences, not isolated samples.
- Can MLP do this?
- Assume we have a sequence of temperature measurements and we want to take 3 sequential measurements and predict the next one

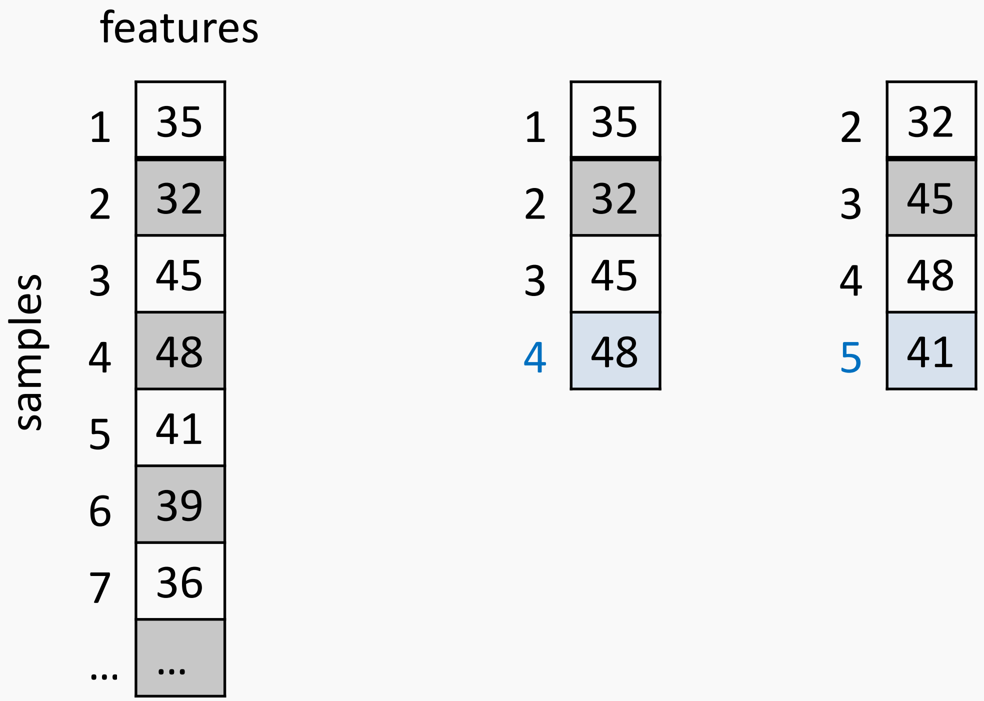
features

samples	
1	35
2	32
3	45
4	48
5	41
6	39
7	36
…	…

1	35
2	32
3	45
4	48

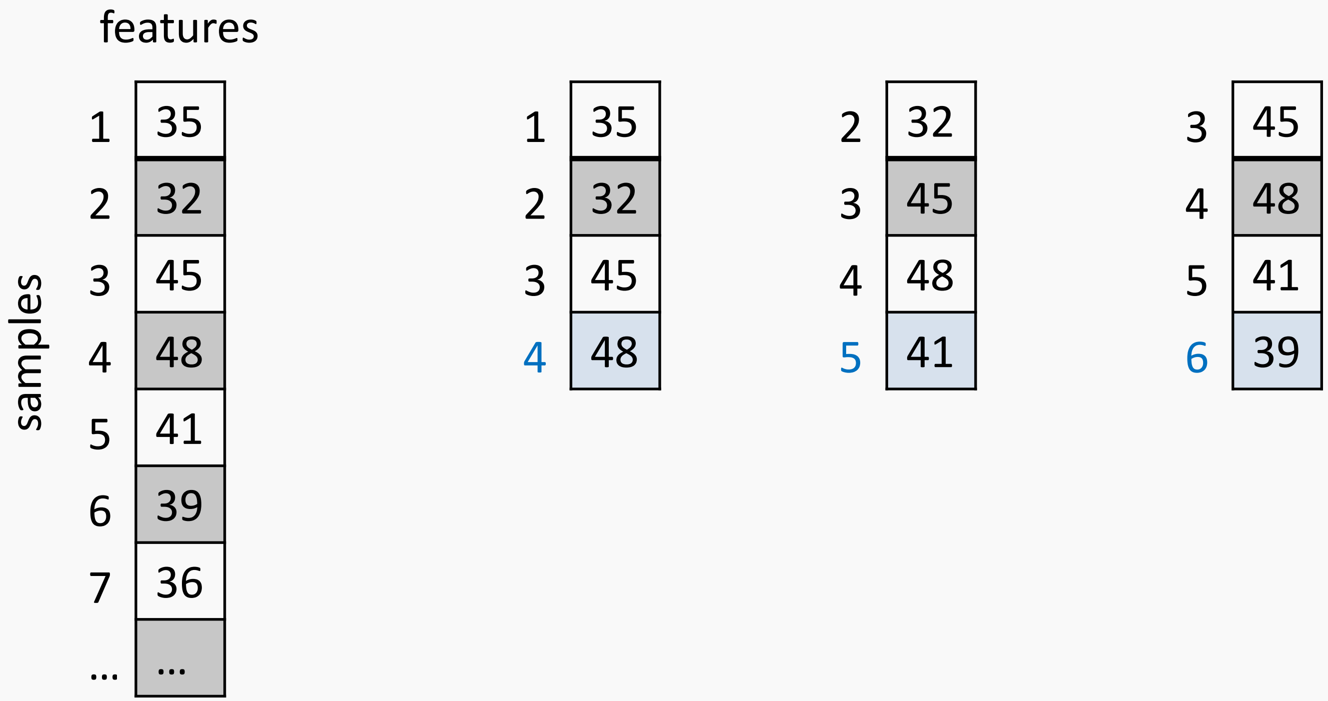
Sequences

- We want a machine learning model to understand sequences, not isolated samples.
- Can MLP do this?
- Assume we have a sequence of temperature measurements and we want to take 3 sequential measurements and predict the next one



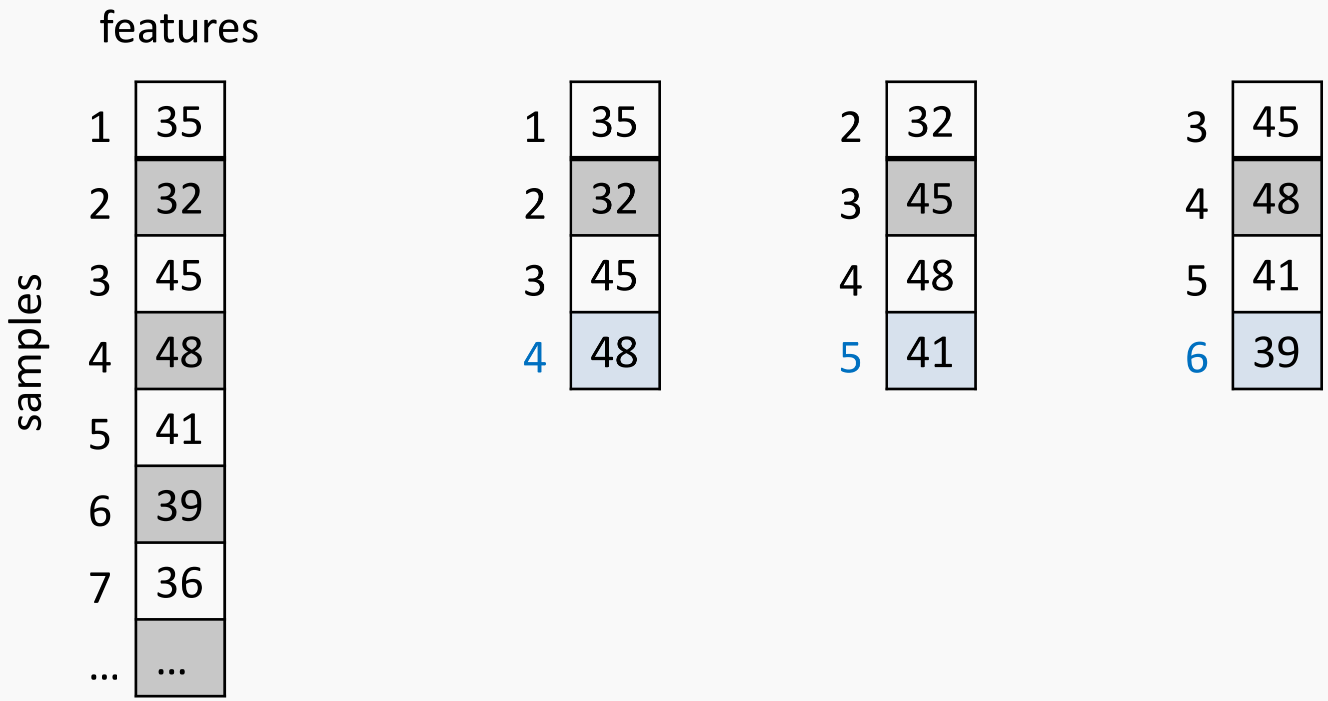
Sequences

- We want a machine learning model to understand sequences, not isolated samples.
- Can MLP do this?
- Assume we have a sequence of temperature measurements and we want to take 3 sequential measurements and predict the next one



Sequences

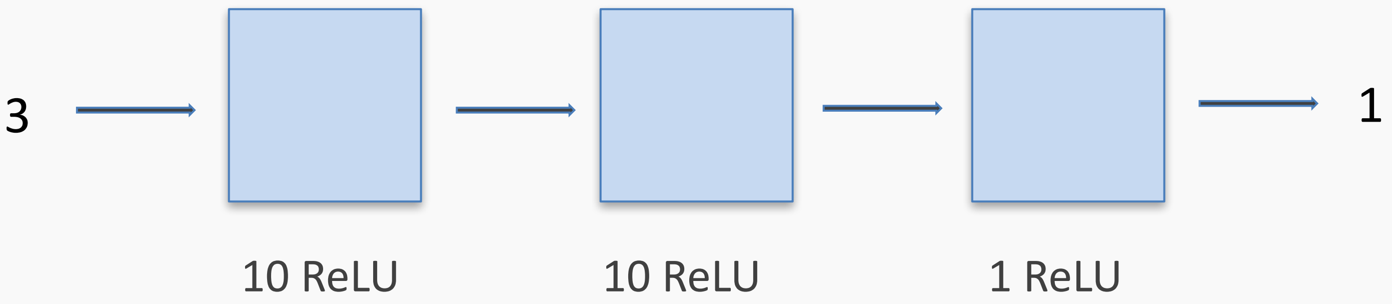
- We want a machine learning model to understand sequences, not isolated samples.
- Can MLP do this?
- Assume we have a sequence of temperature measurements and we want to take 3 sequential measurements and predict the next one



Windowed dataset

This is called **overlapping windowed** dataset, since we're windowing observations to create new.

We can easily do using a MLS:



But re-arranging the order of the inputs like:

3	45	2	32	5	41
1	35	4	48	4	48
2	32	3	45	3	45
4		5		6	

will produce the same results

Why not CNNs or MLPs?

1. MLPs/CNNs require fixed input and output size

2. MLPs/CNNs can't classify inputs in multiple places

Windowed dataset

What follows after: *‘I got in the car and’ ?*

drove away

What follows after: *‘In car the and I’ ?*

Not obvious it should be *‘drove away’*

The order of words matters. This is true for most sequential data.

A fully connected network will not distinguish the order and therefore missing some information.

Outline

Why RNNs

Main Concept of RNNs

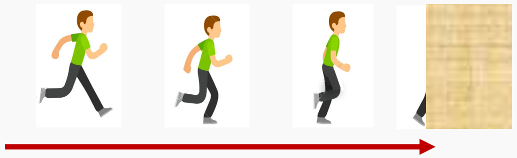
More Details of RNNs

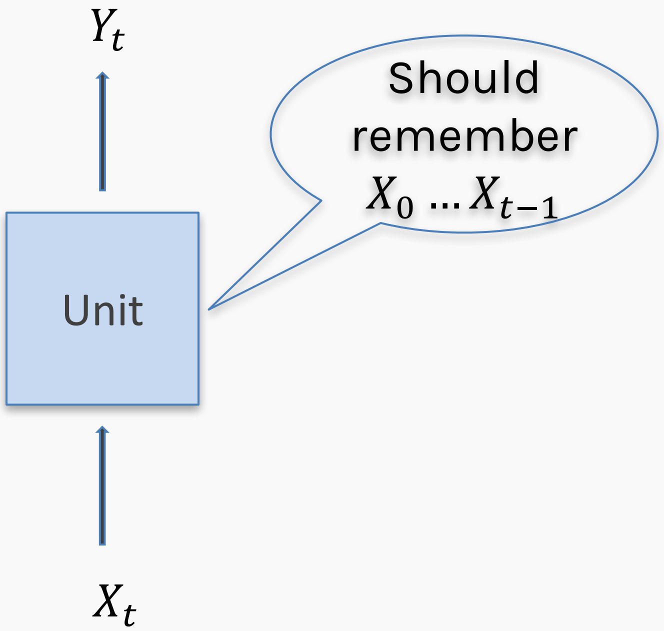
RNN training

Gated RNN

Memory

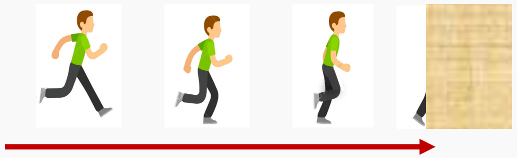
Somehow the computational unit should remember what it has seen before.

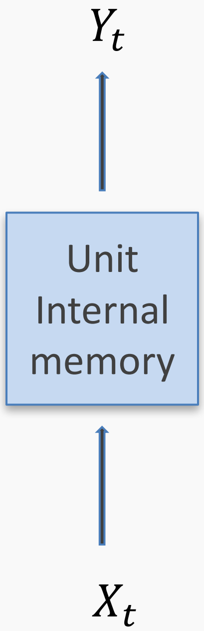




Memory

Somehow the computational unit should remember what it has seen before.

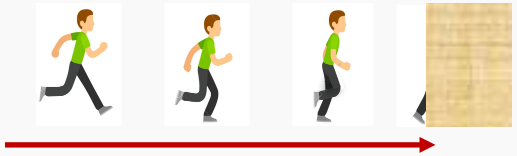


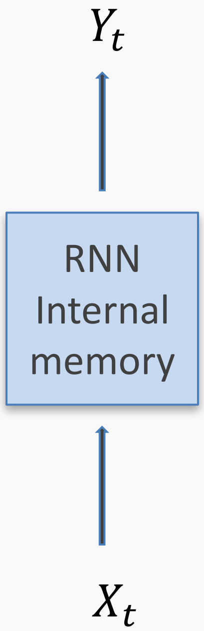


Memory

Somehow the computational unit should remember what it has seen before.

We'll call the information the unit's **state**.





Memory

In neural networks, once training is over, the weights do not change. This means that the network is done learning and done changing.

Then, we feed in values, and it simply applies the operations that make up the network, using the values it has learned.

But the RNN units are able to remember new information after training has completed.

That is, they’re able to keep changing after training is over.

Memory

Question: How can we do this? How can build a unit that remembers the past?

The memory or **state** can be written to a file but in RNNs, we keep it inside the recurrent unit.

In an array or in a vector!

Work with an example:

Anna Sofia said her shoes are too ugly. Her here means Anna Sofia.

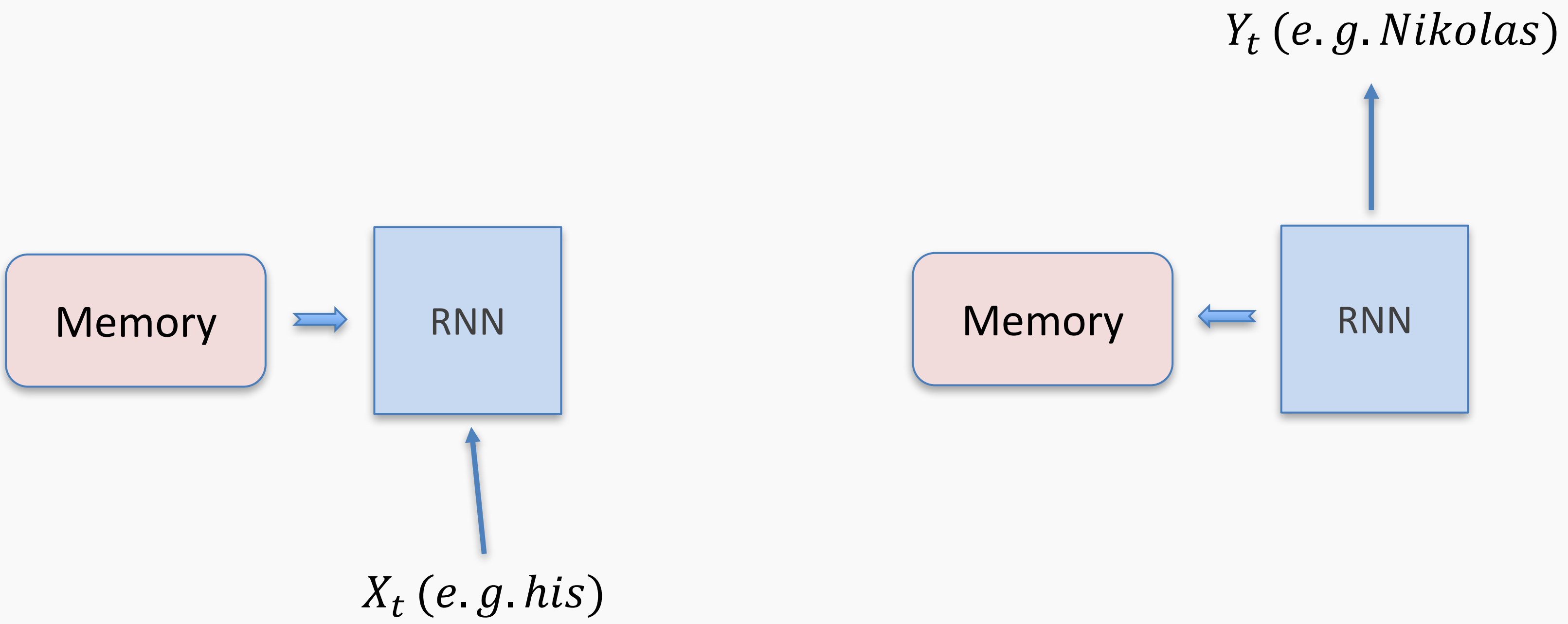
Nikolas put his keys on the table. His here means Nikolas

Memory

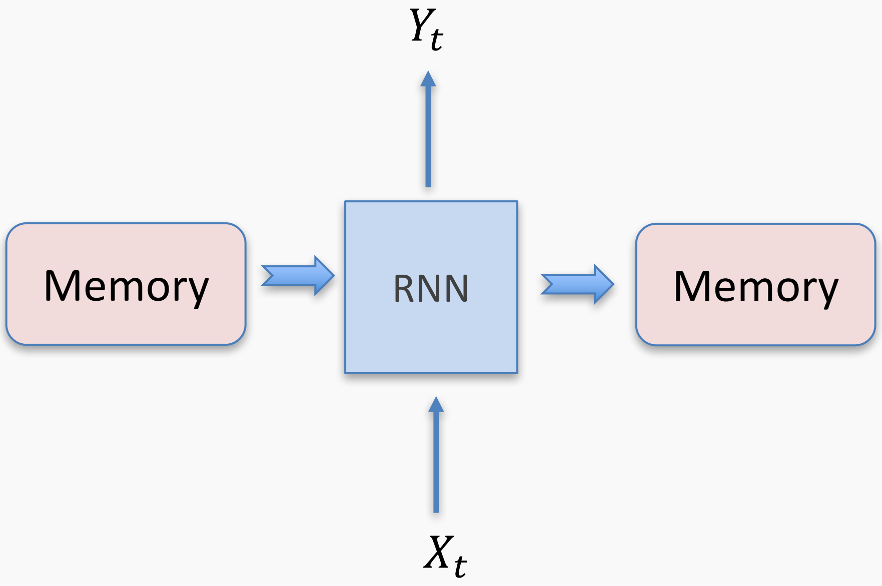
Question: How can we do this? How can build a unit that remembers the past?

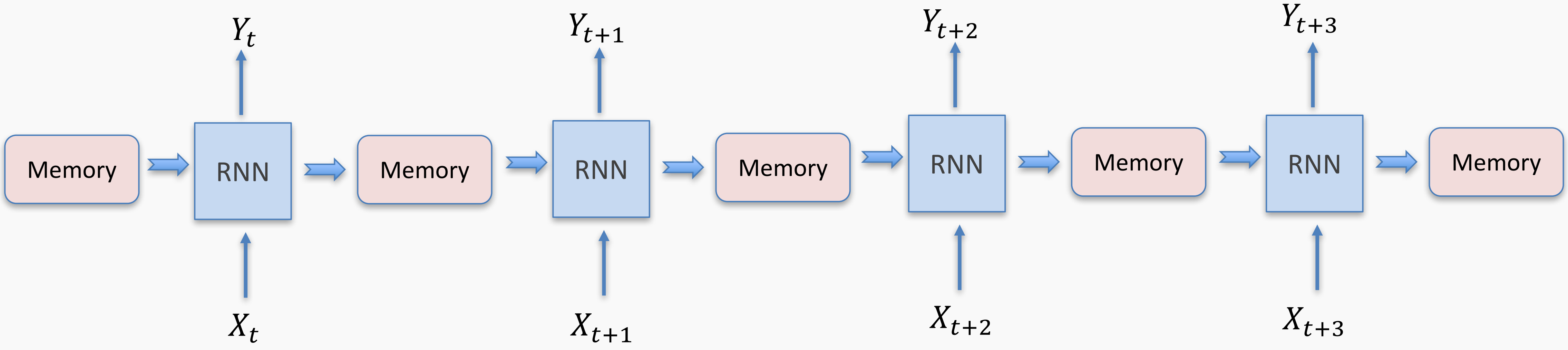
The memory or **state** can be written to a file but in RNNs, we keep it inside the recurrent unit.

In an array or in a vector!



Building an RNN





Outline

Why RNNs

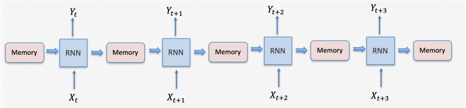
Main Concept of RNNs

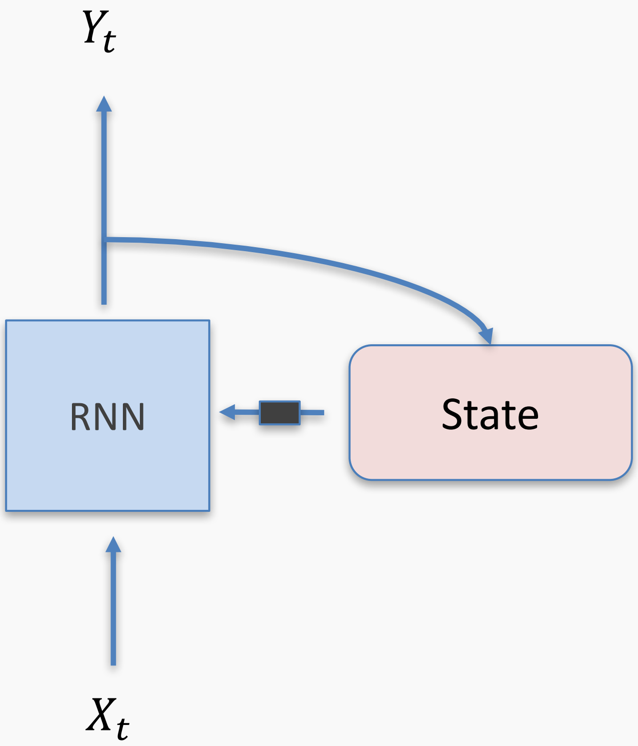
More Details of RNNs

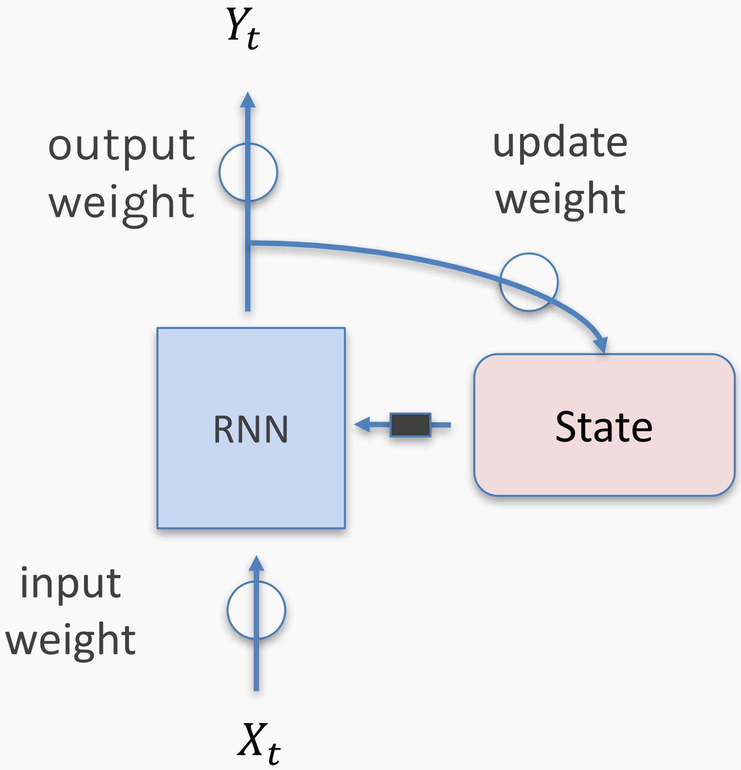
RNN training

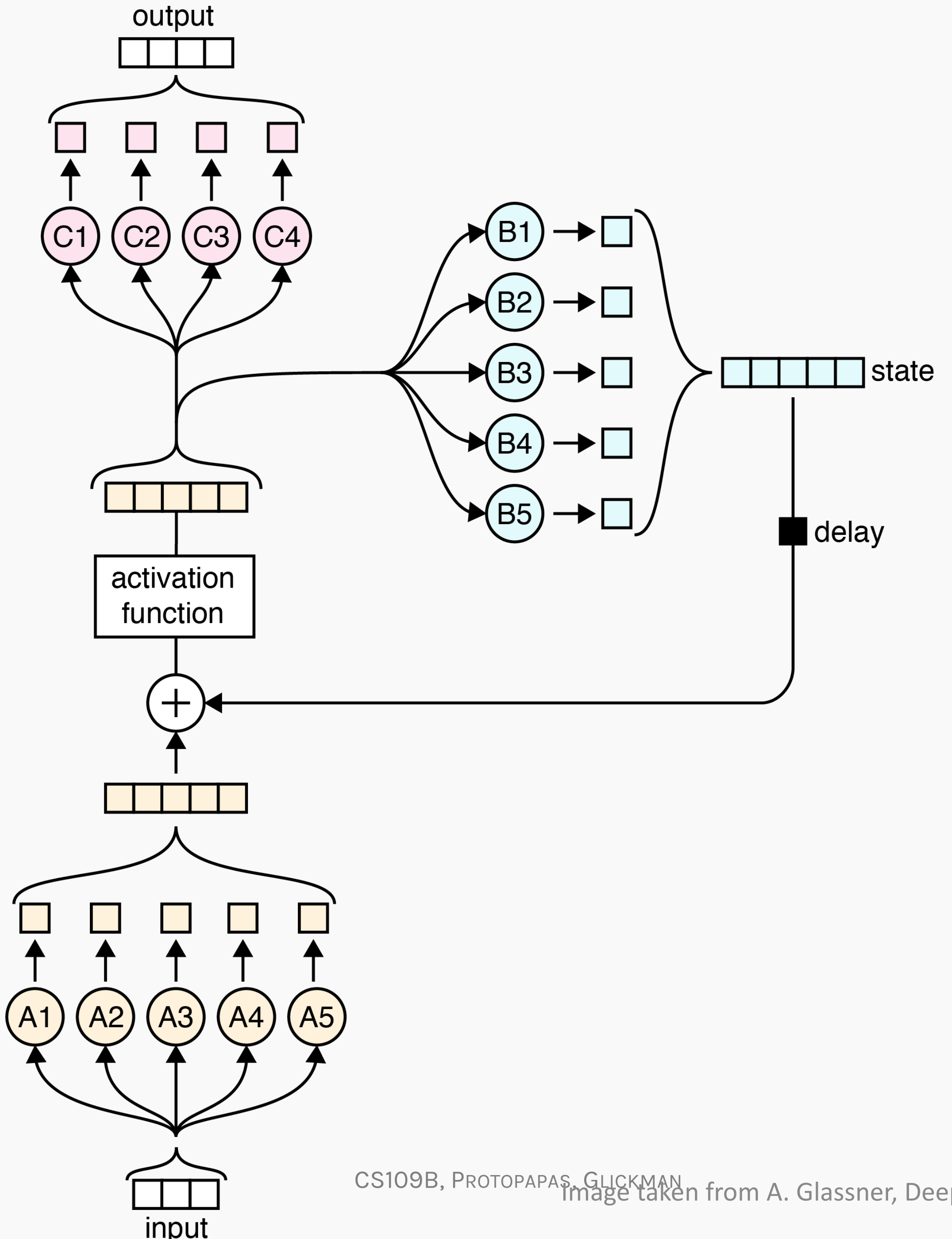
Gated RNN

Structure of an RNN cell









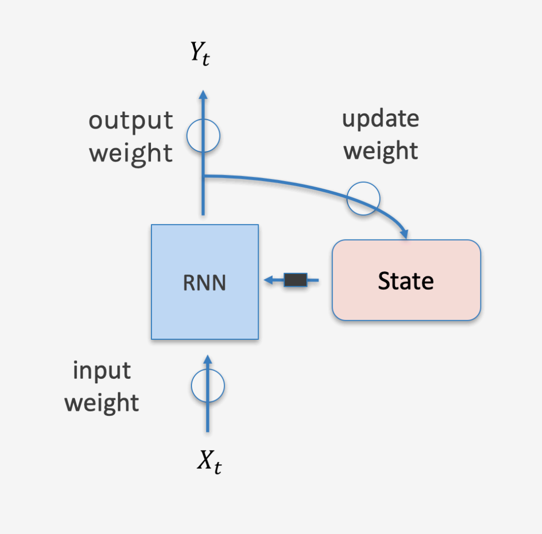


Image taken from A. Glassner, Deep Learning, Vol. 2: From Basics to Practice

Outline

Why RNNs

Main Concept of RNNs

More Details of RNNs

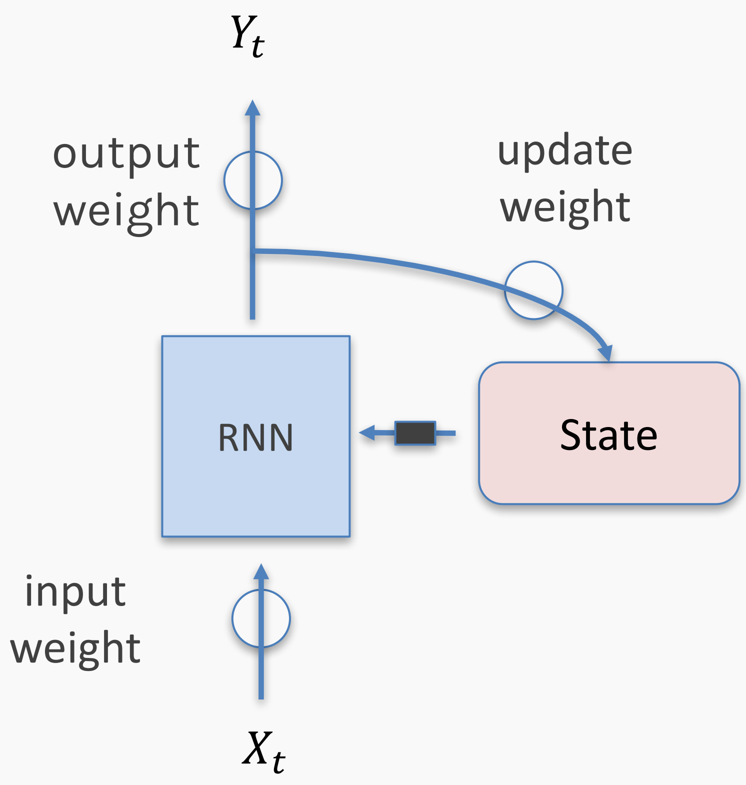
RNN training

Gated RNN

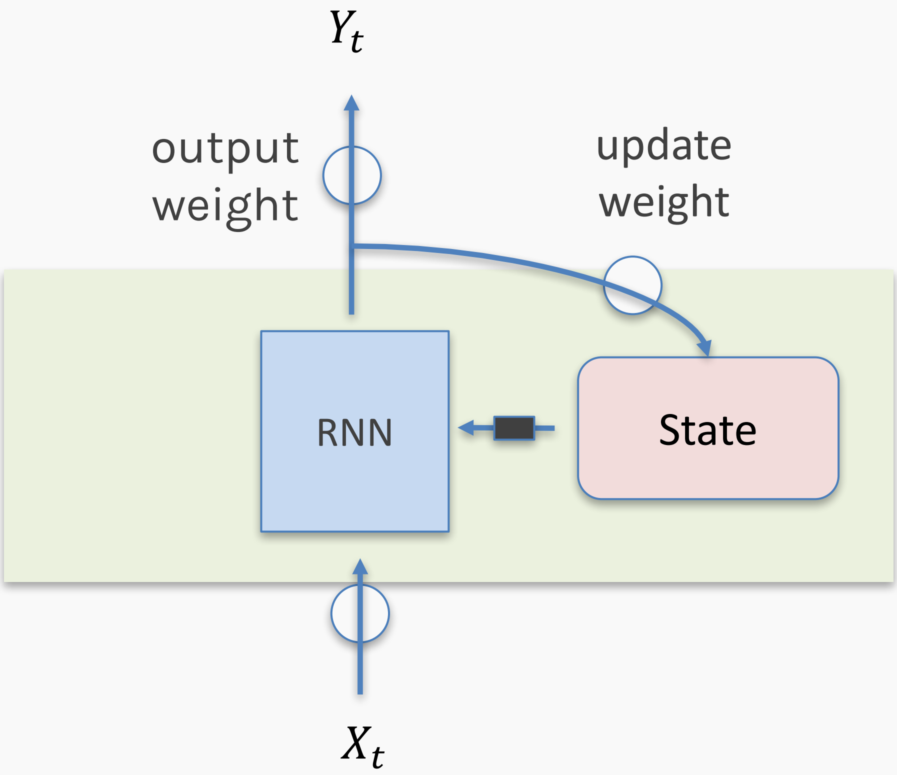
Backprop Through Time

- For each input, unfold network for the sequence length T
- Back-propagation: apply forward and backward pass on unfolded network
- Memory cost: $O(T)$

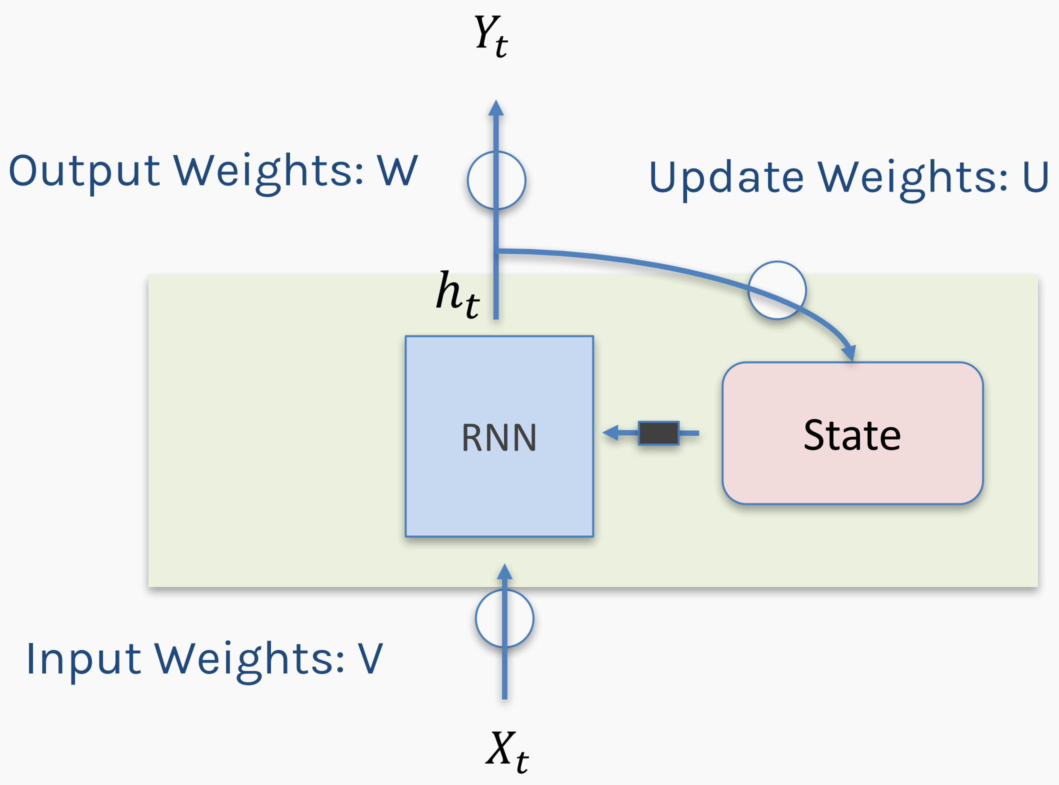
Backprop Through Time



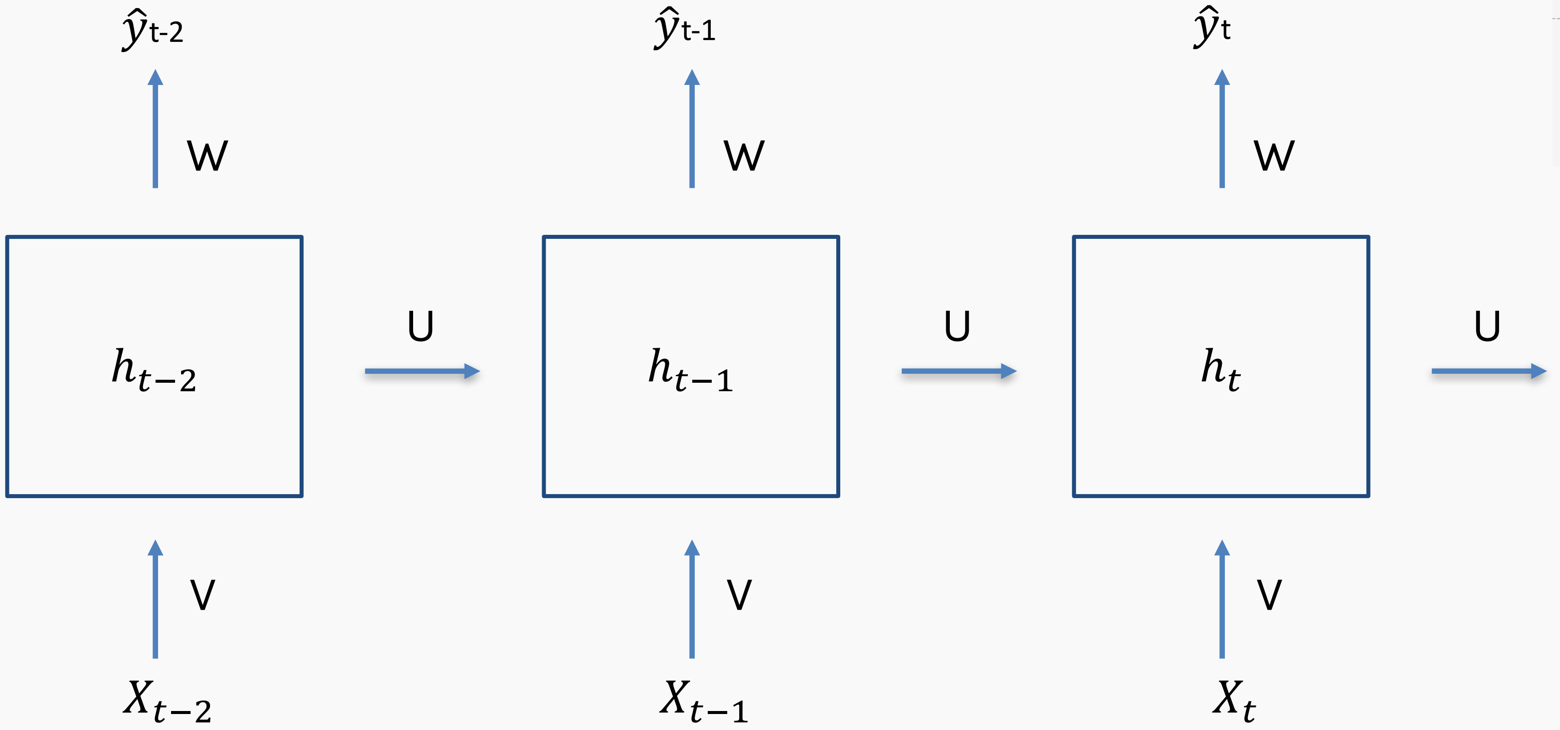
Backprop Through Time

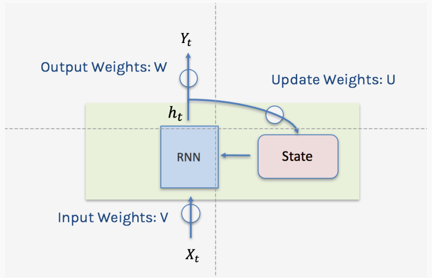


Backprop Through Time



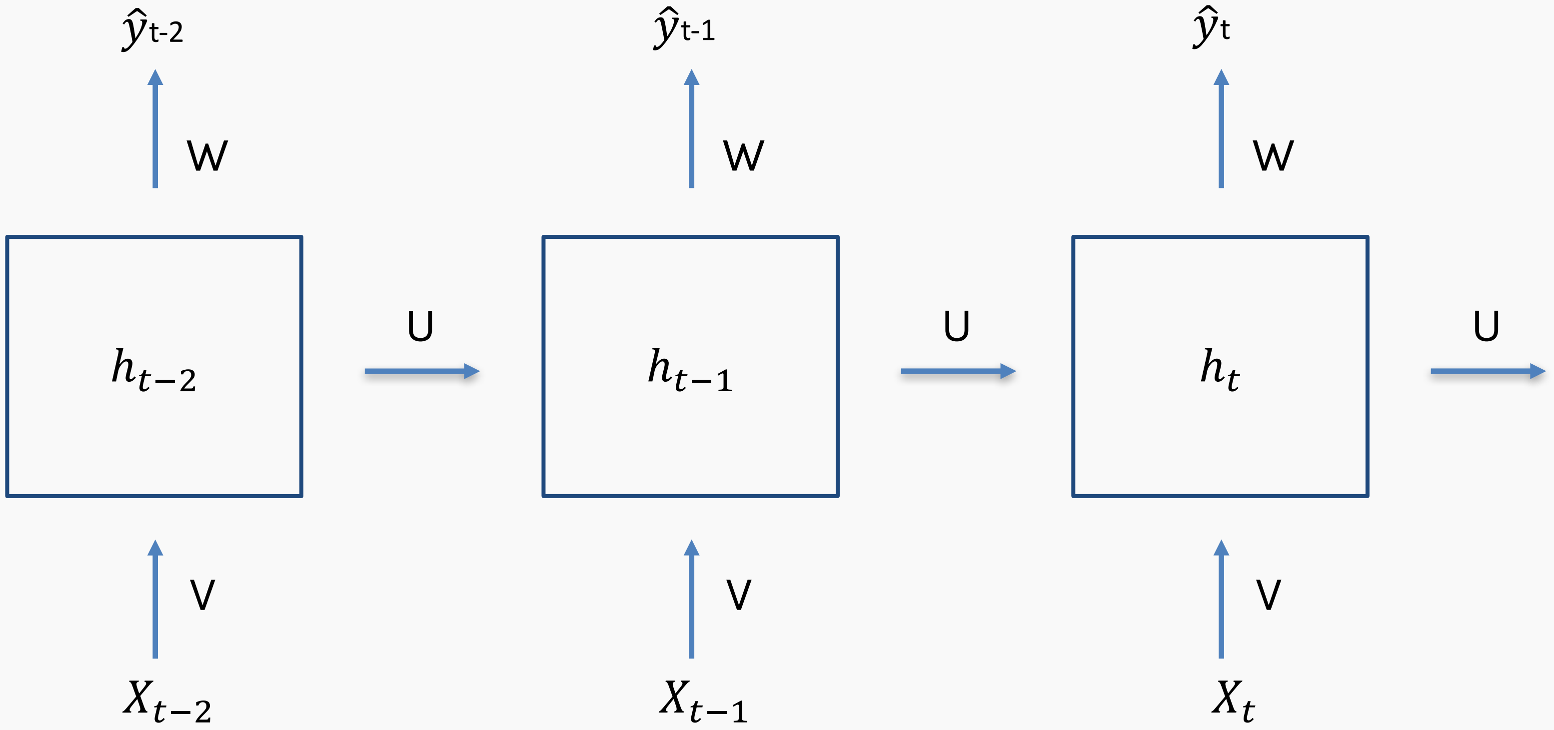
Backprop Through Time





You have two activation functions g_h which serves as the activation for the hidden state and g_y which is the activation of the output. In the example shown before g_y was the identity.

Backprop Through Time



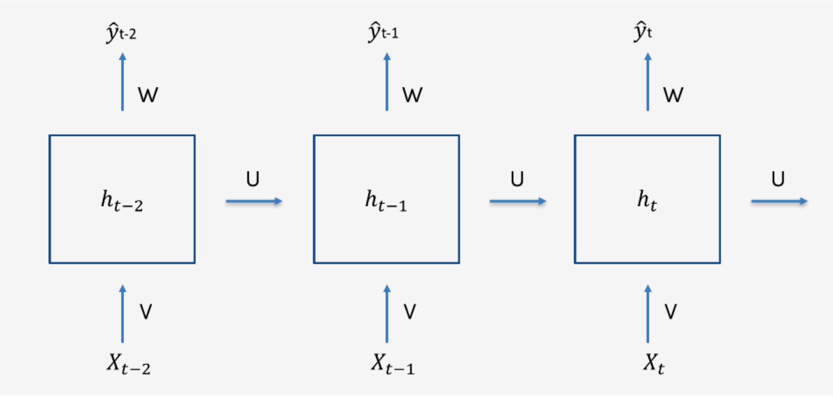
Backprop Through Time

$$\hat{y}_t = g_y(Wh_t + b)$$

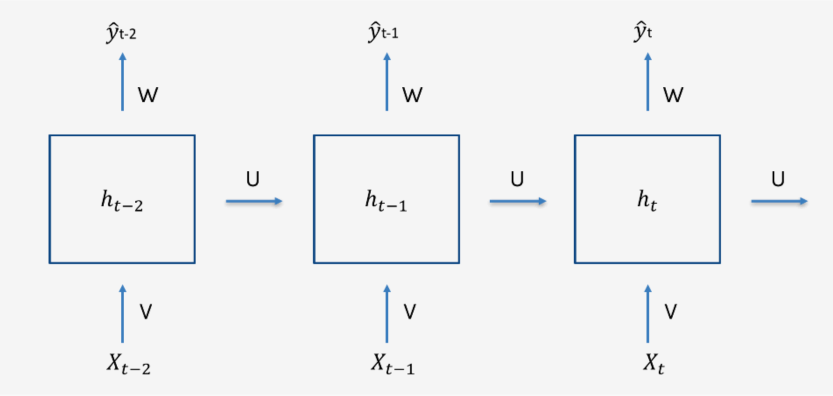
$$L = \sum_t L_t \qquad\qquad L_t = L_t(\hat{y}_t)$$

$$\frac{dL}{dW} = \sum_t \frac{dL_t}{dW} = \sum_t \frac{\partial L_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial W}$$

$$\frac{\partial \hat{y}_t}{\partial W} = g_y{}' h_t$$



Backprop Through Time



$$\hat{y}_t = g_y(Wh_t + b)$$

$$h_t = g_h(Vx_t + Uh_{t-1} + b')$$

$$\hat{y}_t = g_y(Wg_h(Vx_t + Uh_{t-1} + b') + b)$$

$$L = \sum_t L_t \qquad L_t = L_t(\hat{y}_t)$$

$$\frac{dL}{dU} = \sum_t \frac{\partial L_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\frac{\partial h_t}{\partial U}$$

$$\frac{\partial h_t}{\partial U} = \sum_{k=1}^{t} \frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial U}$$

$$\frac{\partial h_j}{\partial h_{j-1}} = g_h' U$$

$$\frac{\partial h_t}{\partial h_k} = \frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots \frac{\partial h_{k+1}}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}$$

$$\frac{\partial L_t}{\partial U} = \frac{\partial L_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\left(\frac{dh_t}{dU} + \frac{dh_t}{dh_{t-1}}\frac{dh_{t-1}}{dU} + \frac{dh_t}{dh_{t-1}}\frac{dh_{t-1}}{dh_{t-2}}\frac{dh_{t-2}}{dU} + \cdots\right)$$

Gradient Clipping

Prevents exploding gradients

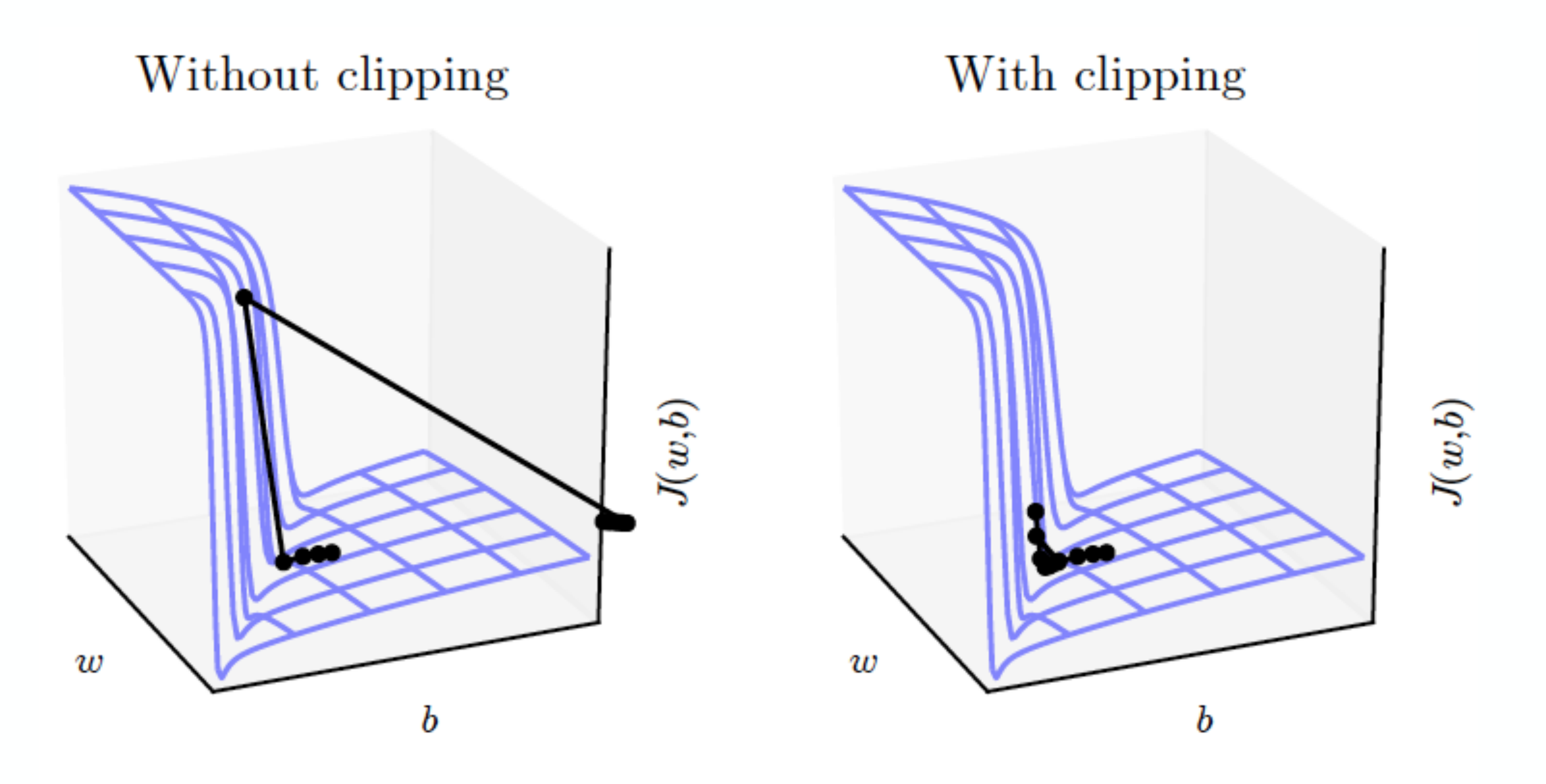
Clip the norm of gradient before update.

For some derivative g, and some threshold u

$$\text{if } ||g|| > u$$

$$g \leftarrow \frac{gu}{||g||}$$

Gradient Clipping



Outline

Why RNNs

Main Concept of RNNs

More Details of RNNs

RNN training

Gated RNN

Long-term Dependencies

Unfolded networks can be very deep

Long-term interactions are given exponentially smaller weights than small-term interactions

Gradients tend to either *vanish* or *explode*

Long Short-Term Memory

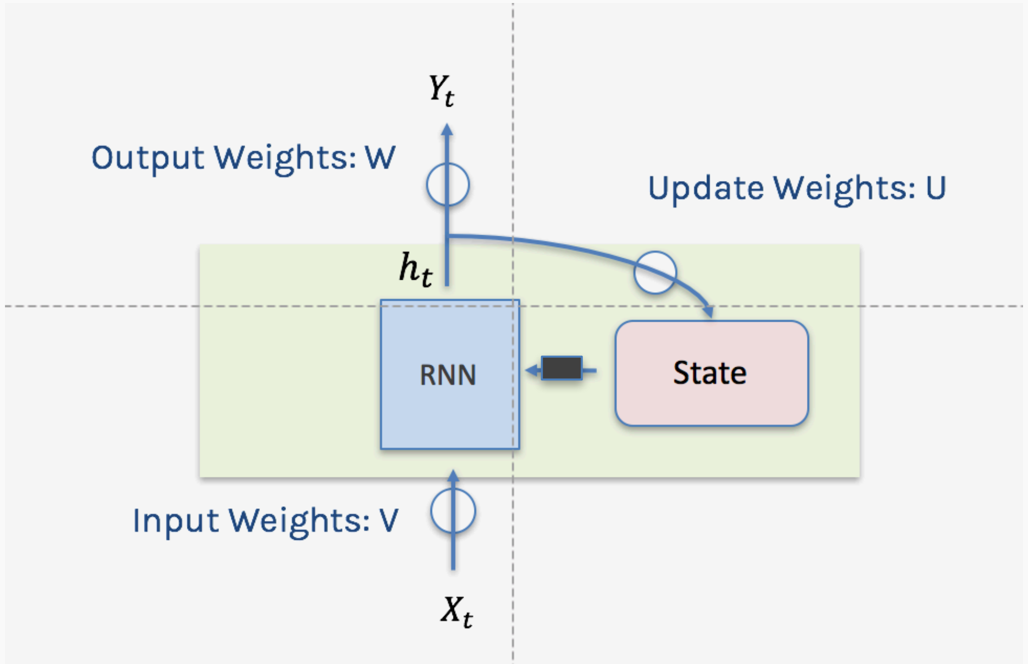
Handles long-term dependencies

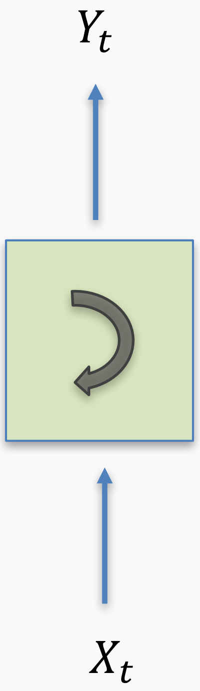
Leaky units where weight on self-loop α is context-dependent

Allow network to decide whether to accumulate or forget past info

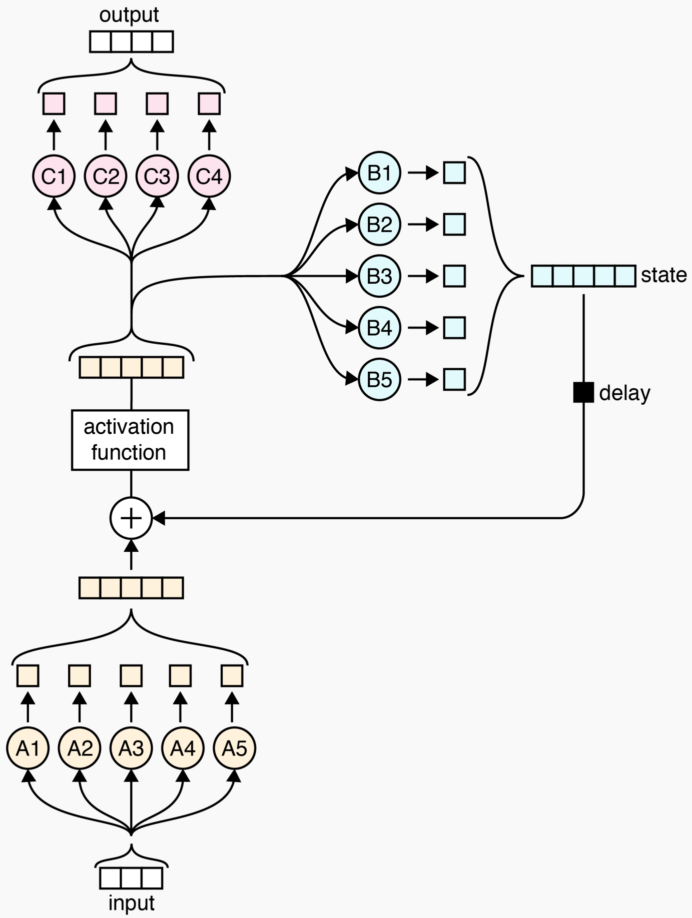
Notation

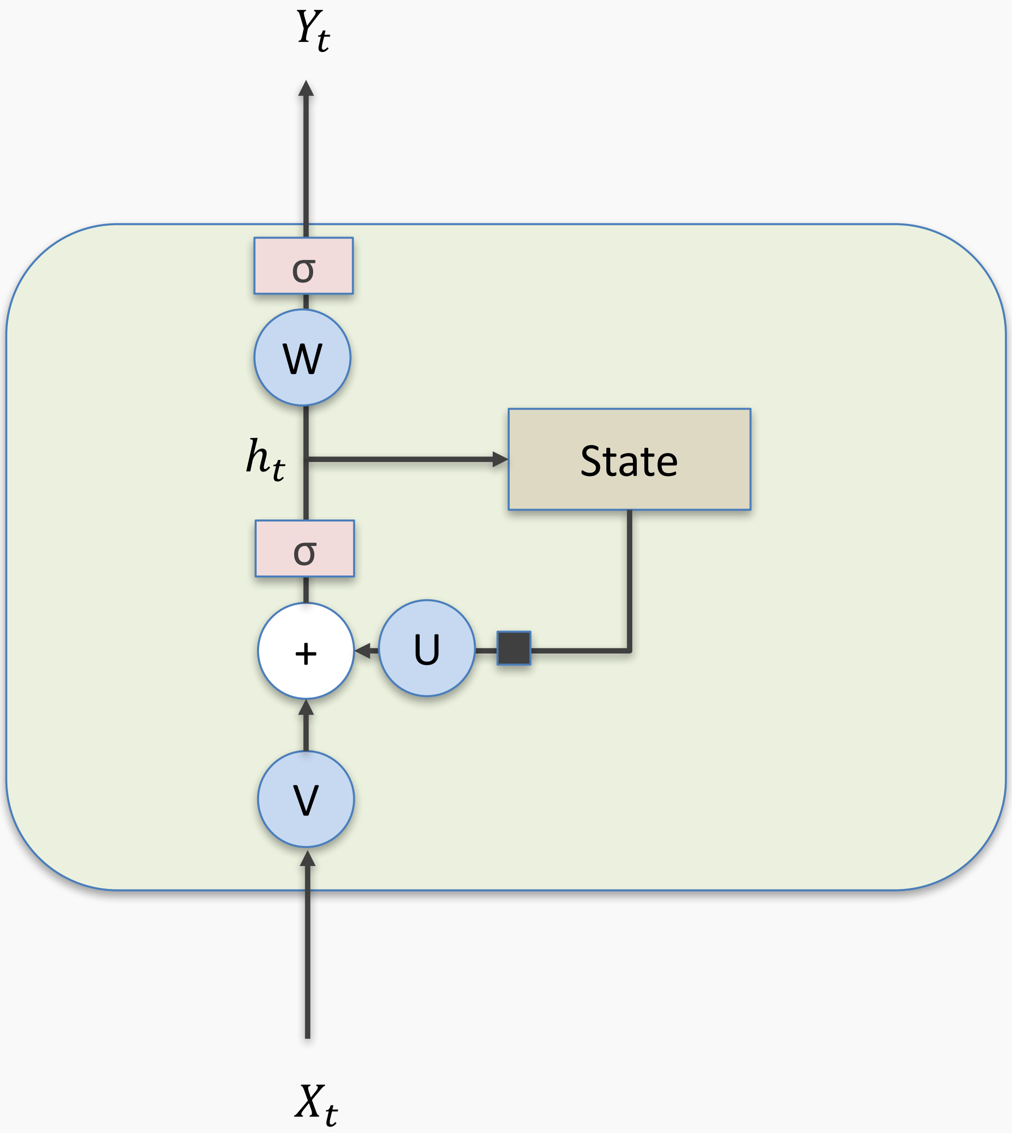
Using conventional and convenient notation



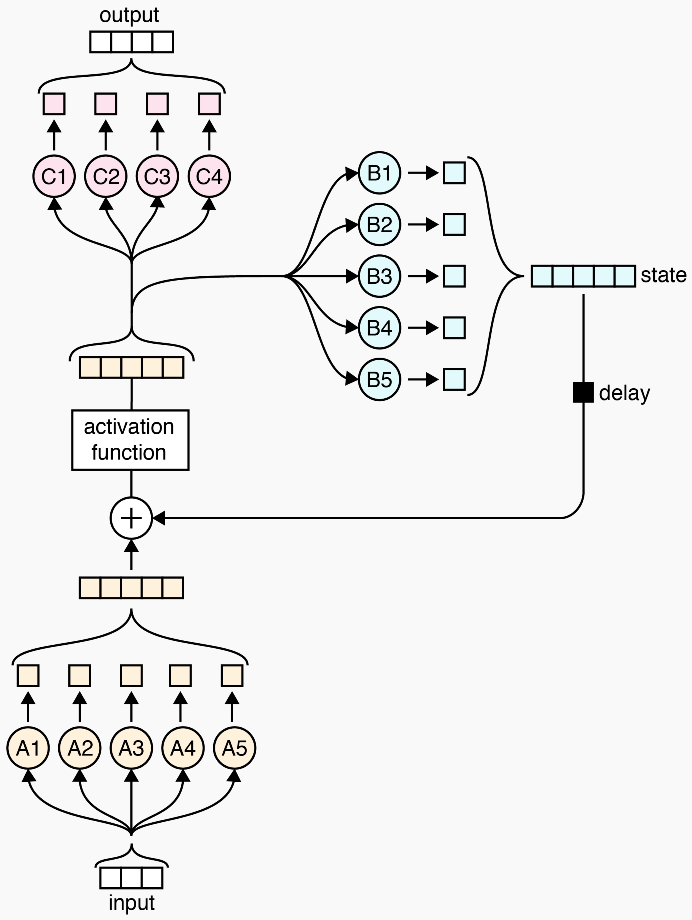


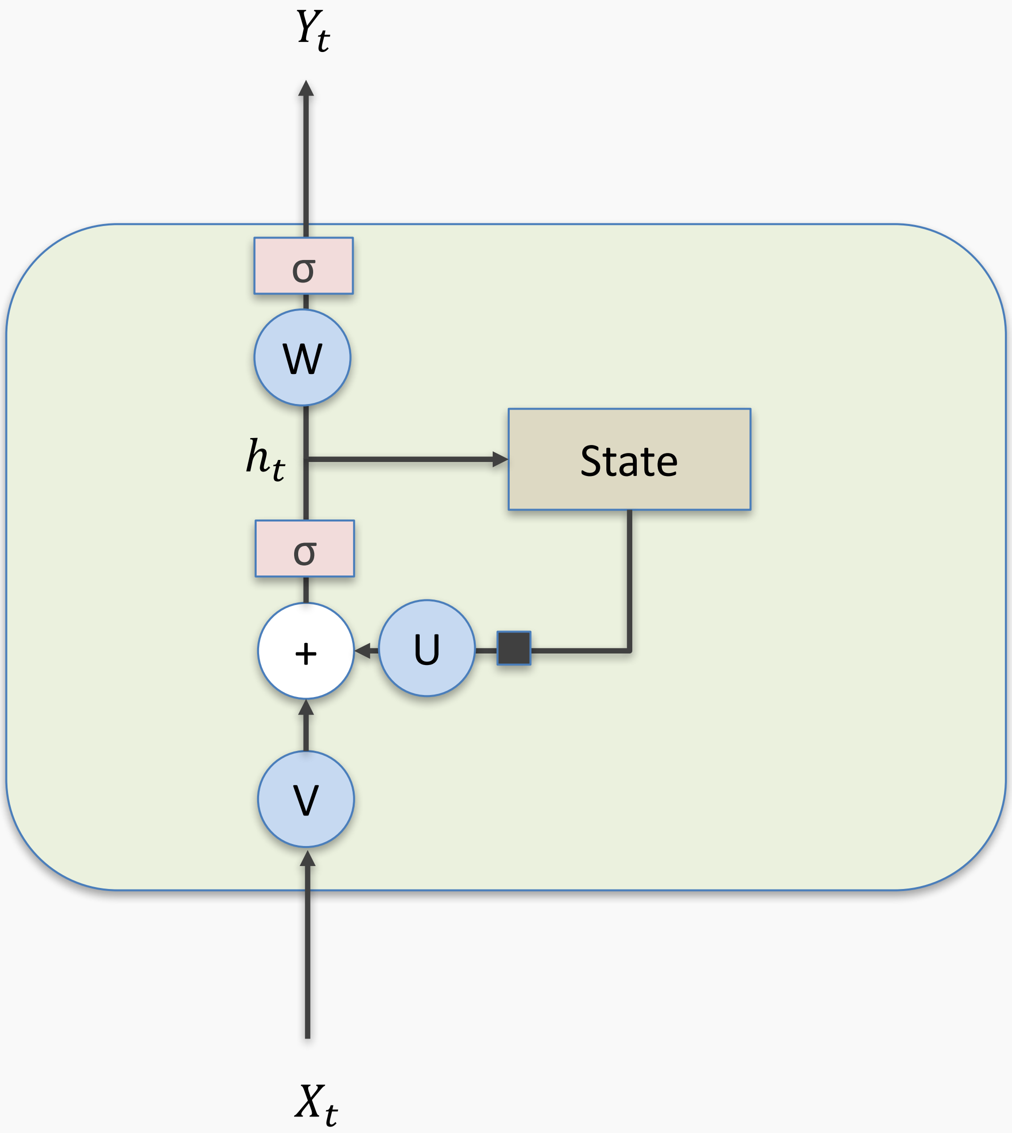
Simple RNN again



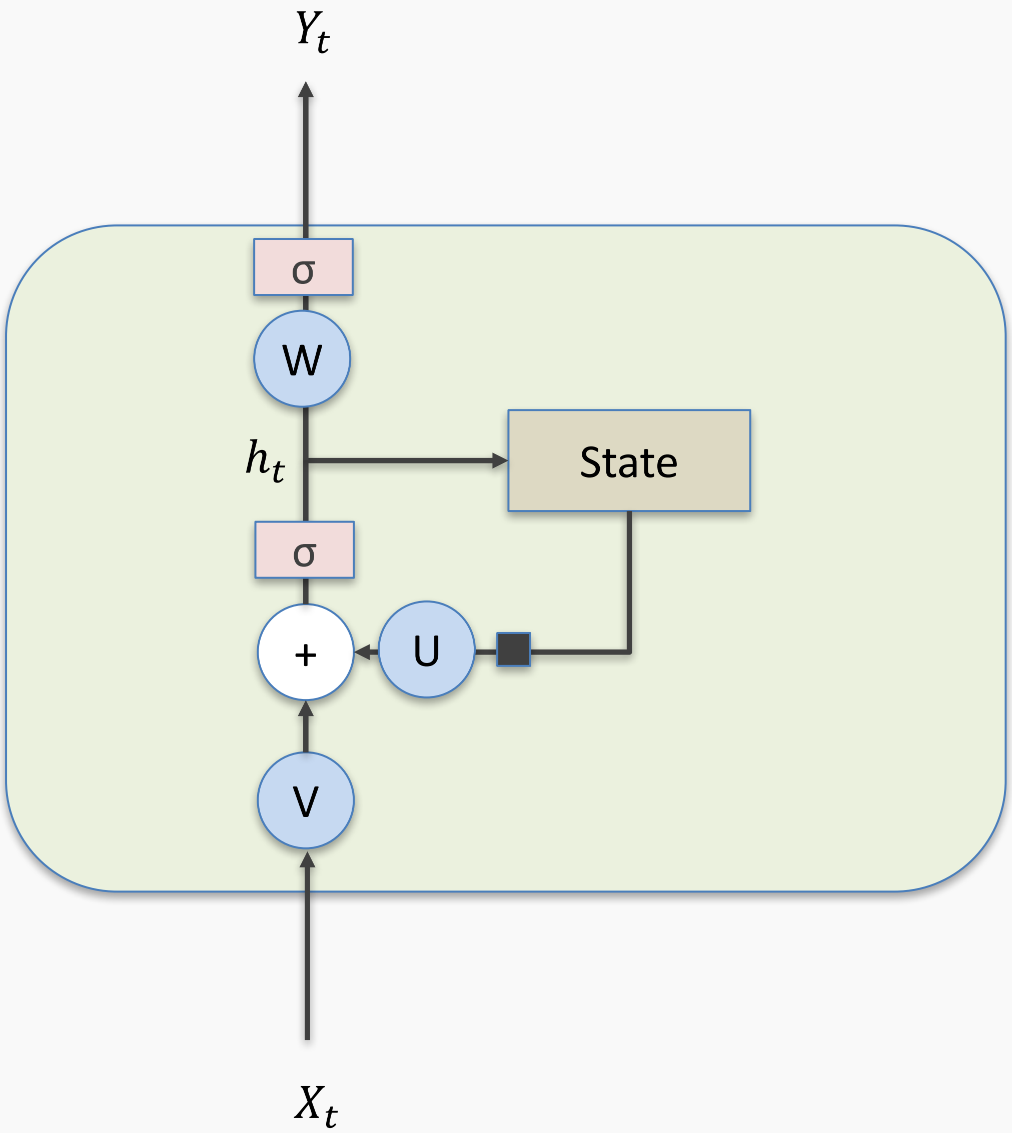


Simple RNN again

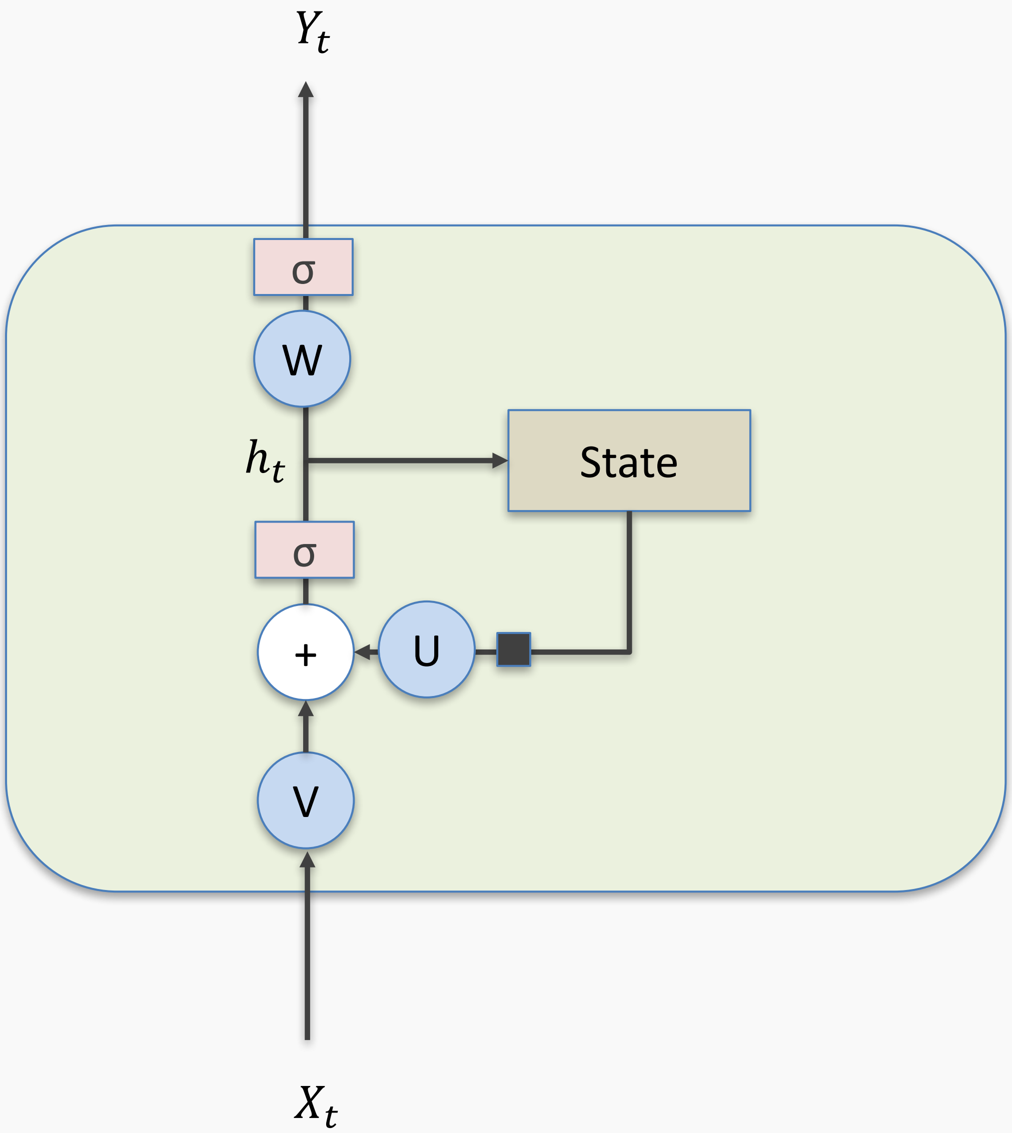




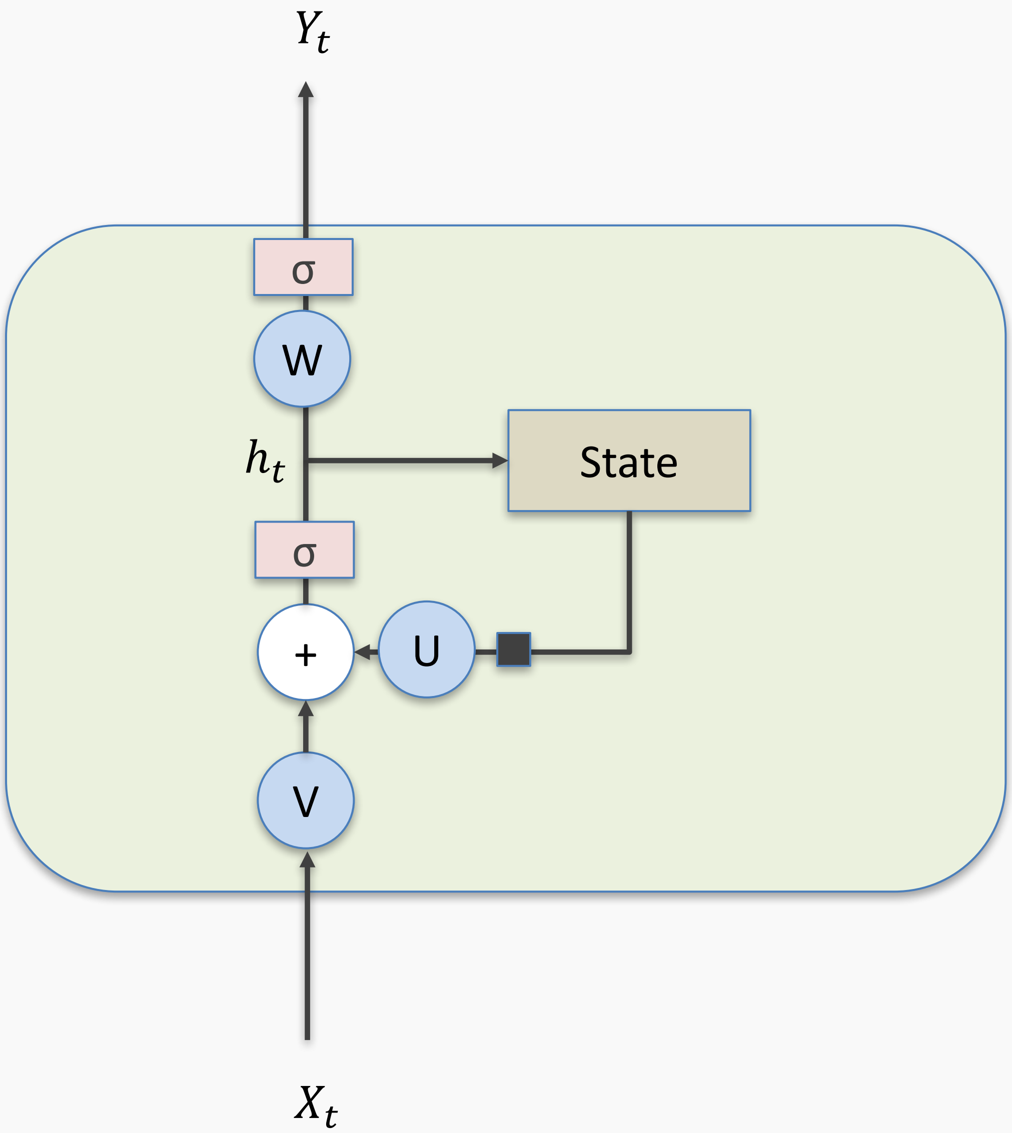
Simple RNN again: **Memories**

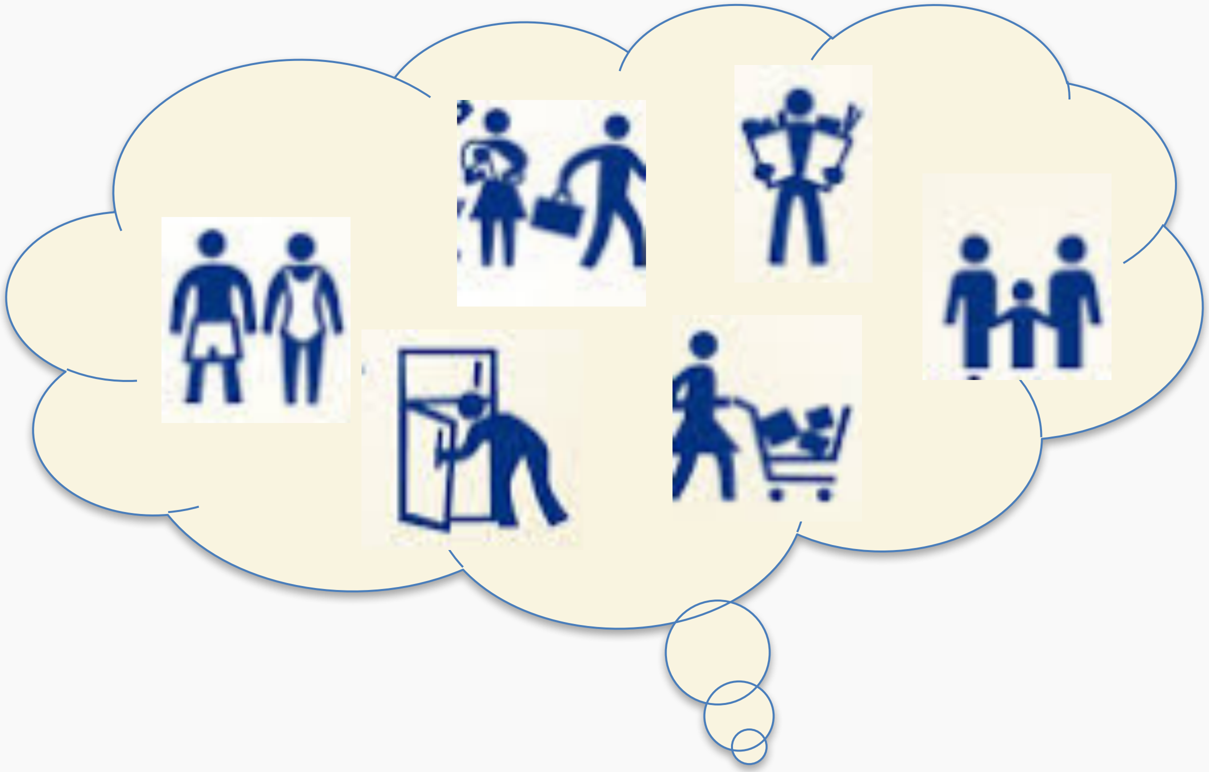


Simple RNN again: **Memories - Forgetting**

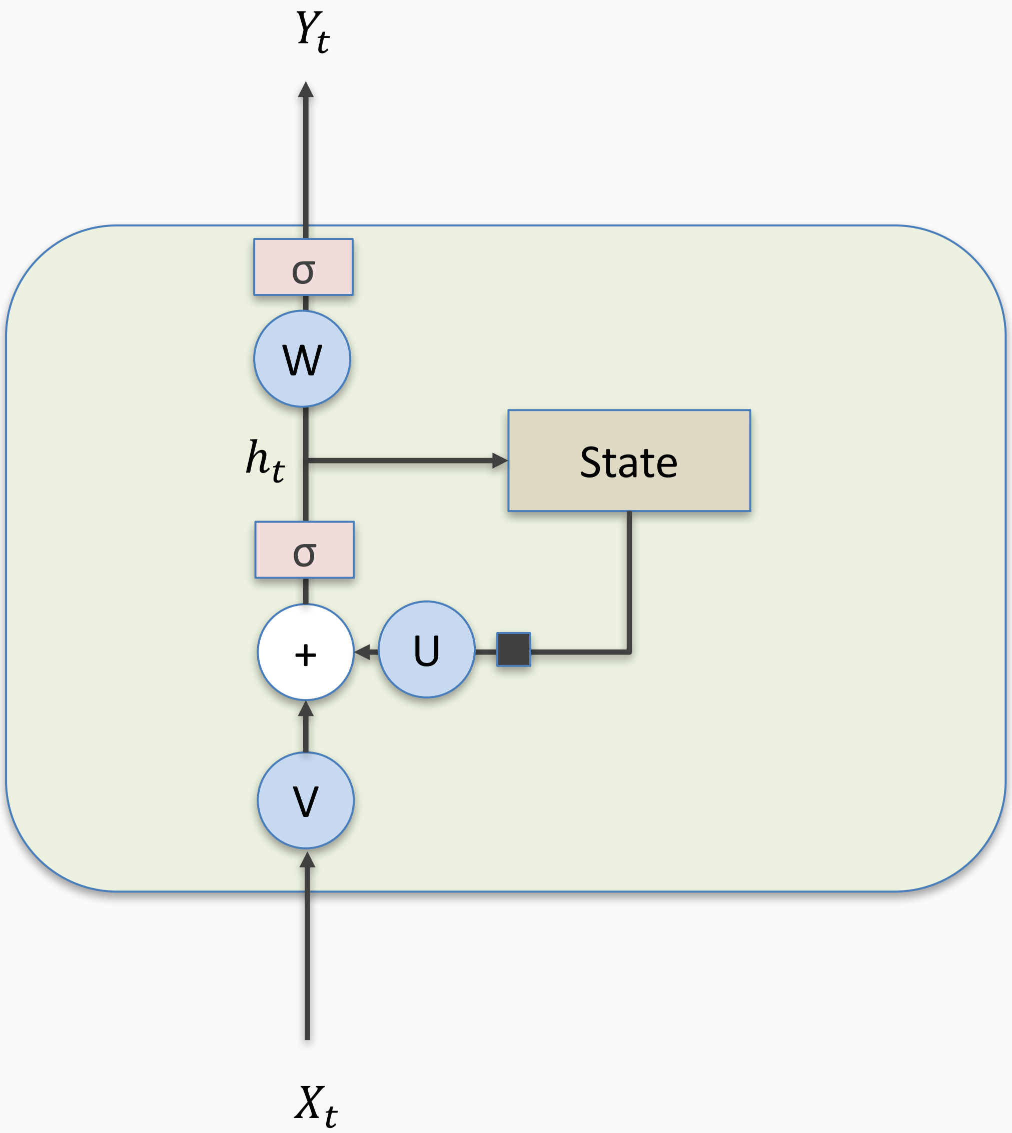


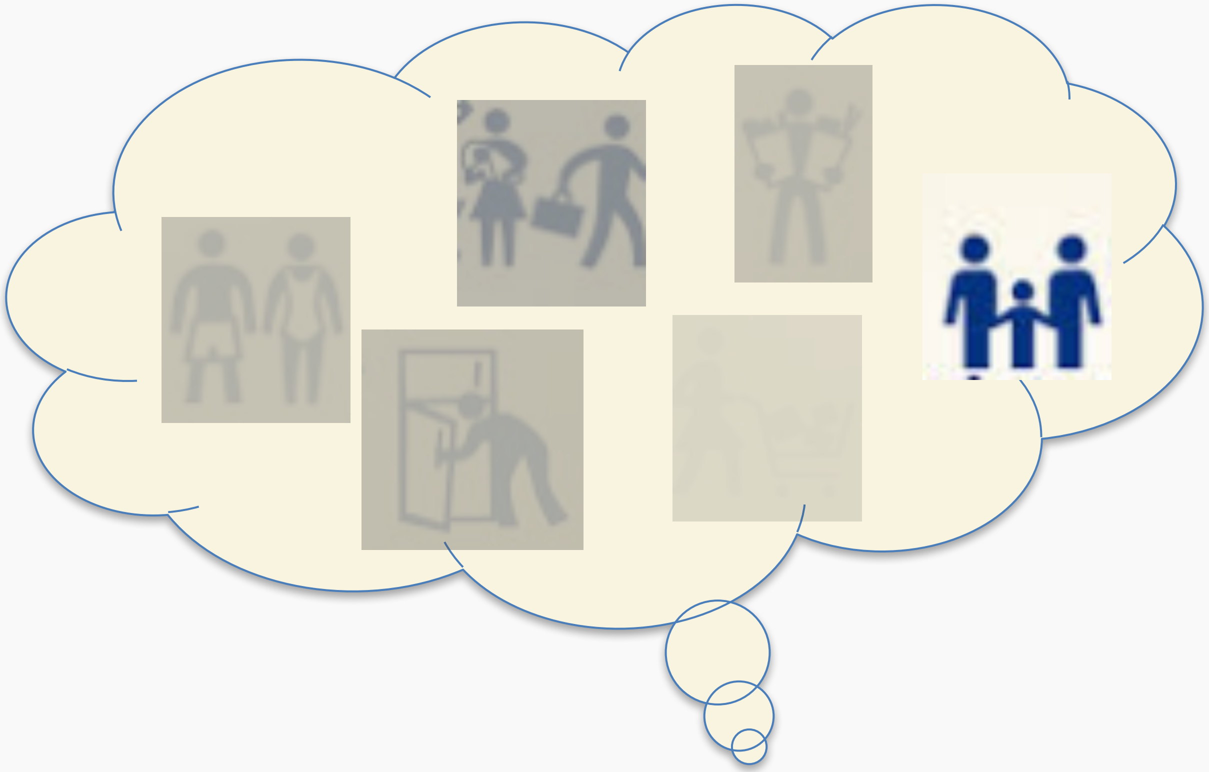
Simple RNN again: **New Events**



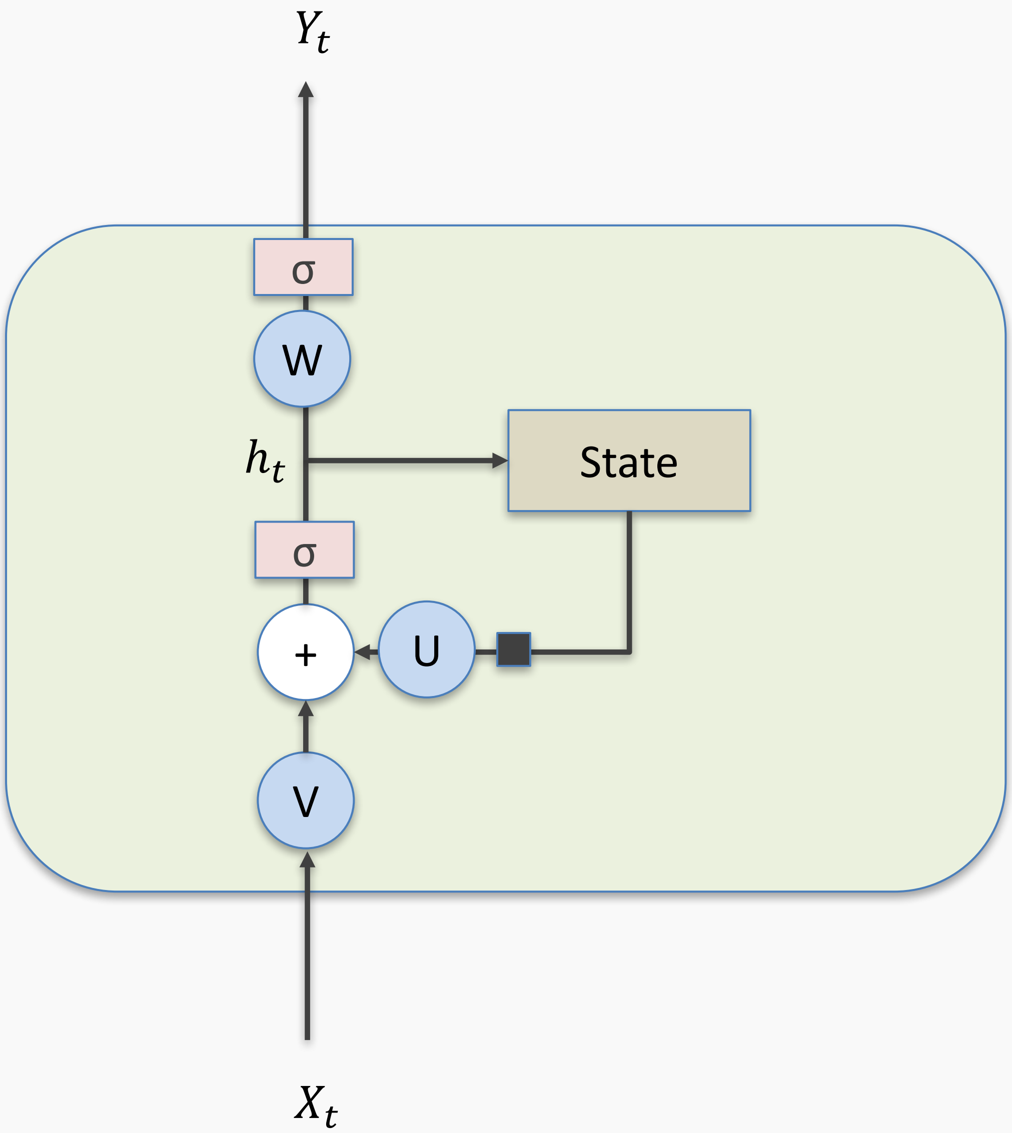


Simple RNN again: **New Events Weighted**





Simple RNN again: **Updated memories**



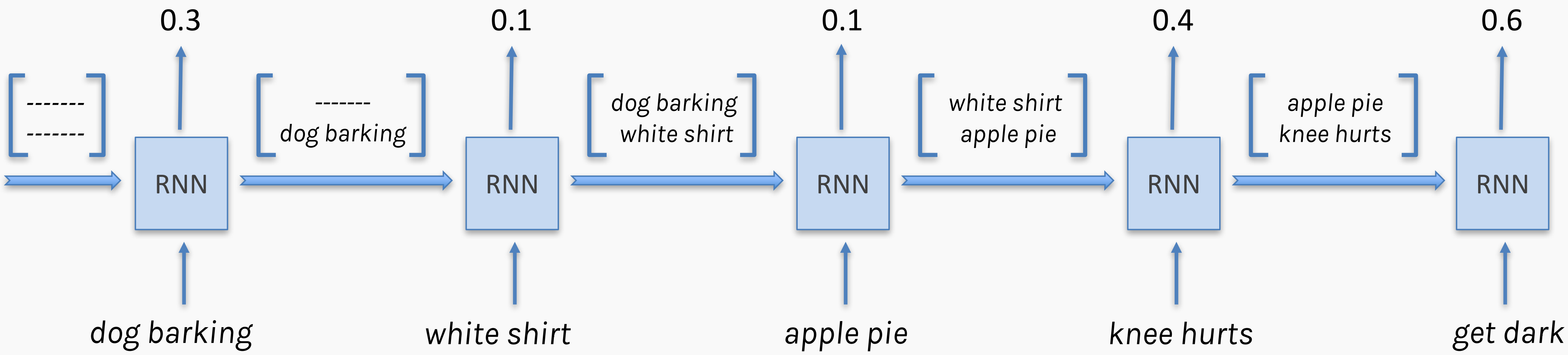
Ref

- [Chen17b] Qiming Chen, Ren Wu, “CNN Is All You Need”, arXiv 1712.09662, 2017. https://arxiv.org/abs/1712.09662
- [Chu17] Hang Chu, Raquel Urtasun, Sanja Fidler, “Song From PI: A Musically Plausible Network for Pop Music Generation”, arXiv preprint, 2017. https://arxiv.org/abs/1611.03477
- [Johnson17] Daniel Johnson, “Composing Music with Recurrent Neural Networks”, Heahedria, 2017. http://www.hexahedria.com/2015/08/03/ composing-music-with-recurrent-neural-networks/
- [Deutsch16b] Max Deutsch, “Silicon Valley: A New Episode Written by AI”, Deep Writing blog post, 2017. https://medium.com/deep-writing/ silicon-valley-a-new-episode-written-by-ai-a8f832645bc2
- [Fan16] Bo Fan, Lijuan Wang, Frank K. Soong, Lei Xie “Photo-Real Talking Head with Deep Bidirectional LSTM”, Multimedia Tools and Applications, 75(9), 2016. https://www.microsoft.com/en-us/research/wp-content/uploads/2015/04/icassp2015_fanbo_1009.pdf

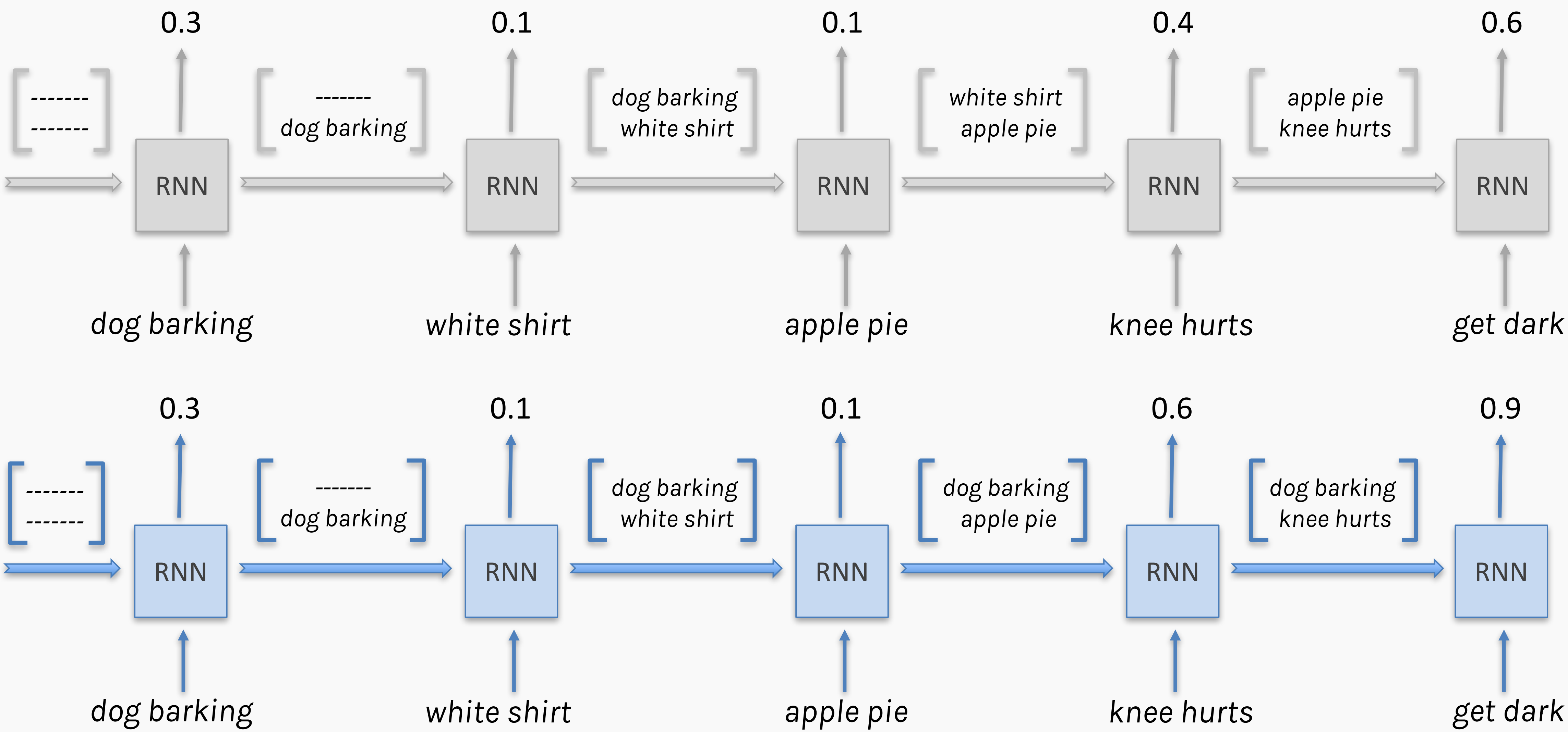
Continue on Wednesday

RNN

Is it raining? We build an RNN to the probability if it is raining:



RNN + Memory

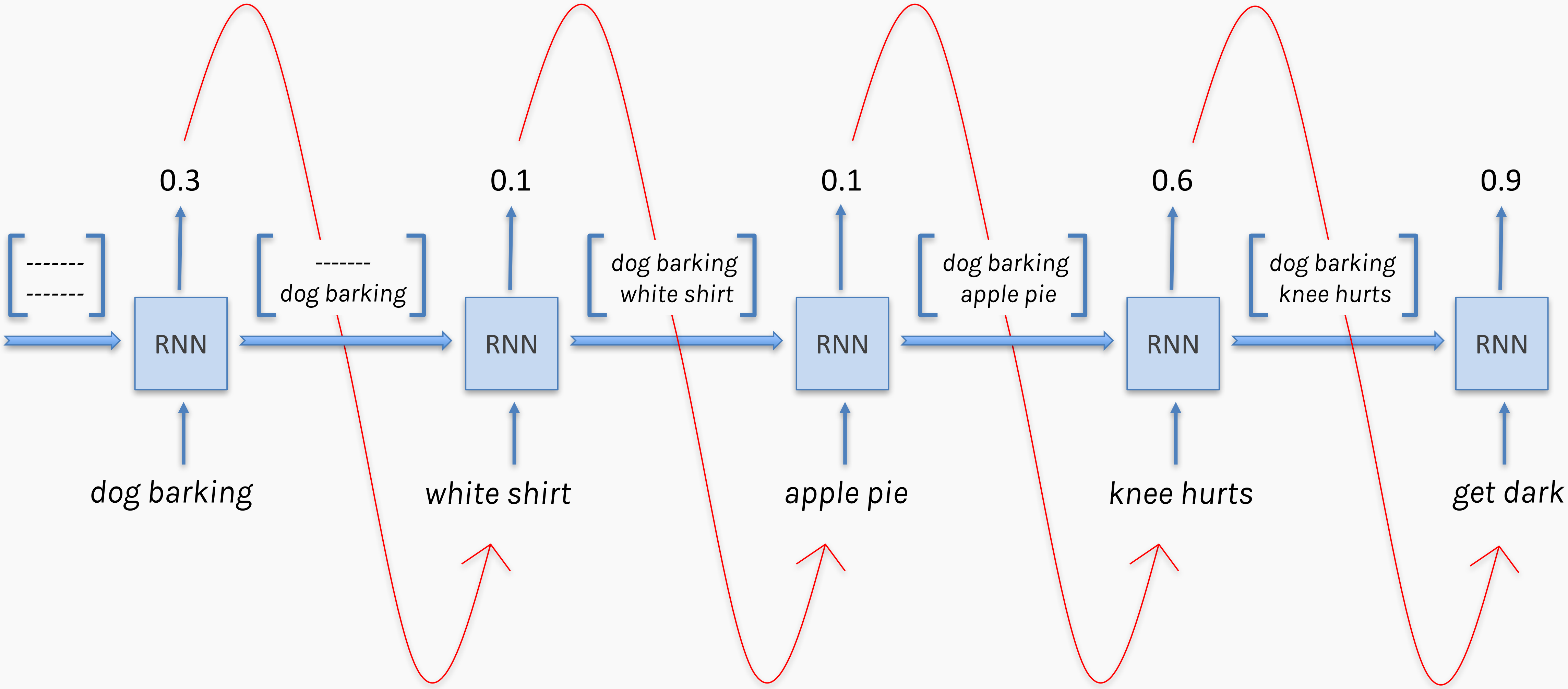


0.3
0.1
0.1
0.4
0.6

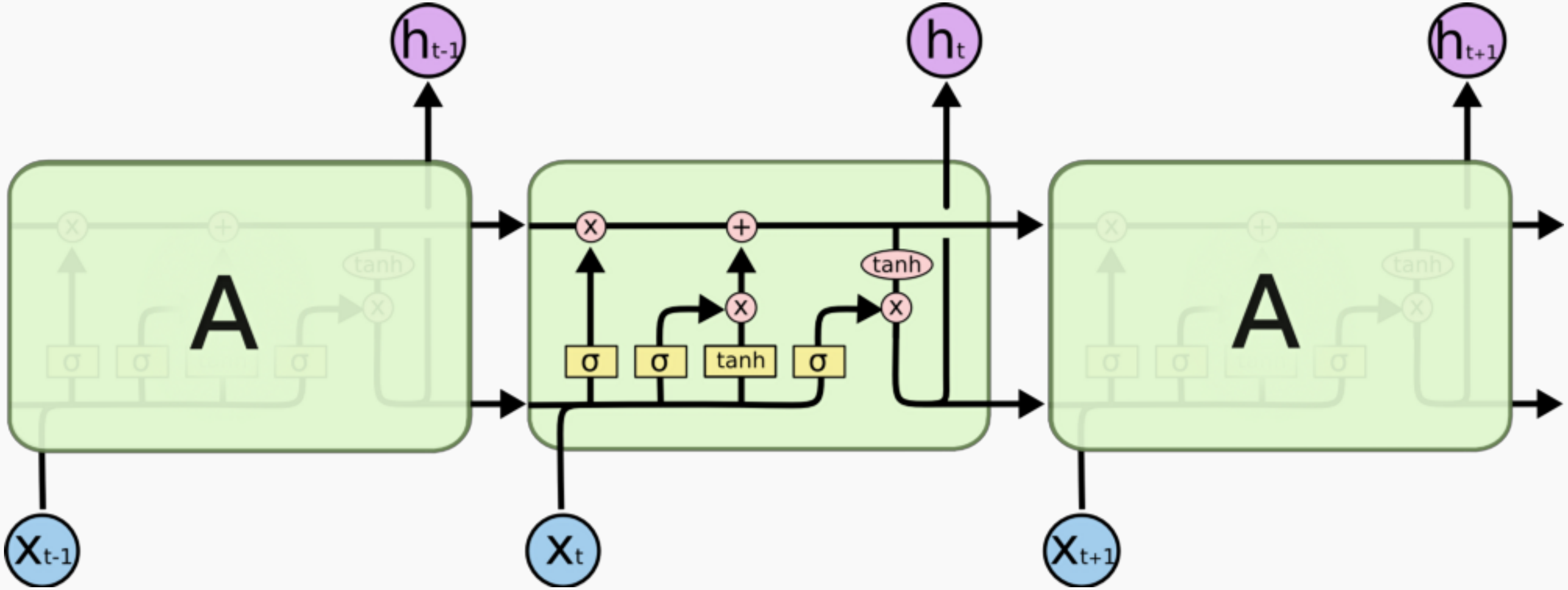
----
dog barking
dog barking
white shirt
white shirt
apple pie
apple pie
knee hurts
RNN
RNN
RNN
RNN
RNN
dog barking
white shirt
apple pie
knee hurts
get dark
0.3
0.1
0.1
0.6
0.9

----
dog barking
dog barking
white shirt
dog barking
apple pie
dog barking
knee hurts
RNN
RNN
RNN
RNN
RNN
dog barking
white shirt
apple pie
knee hurts
get dark

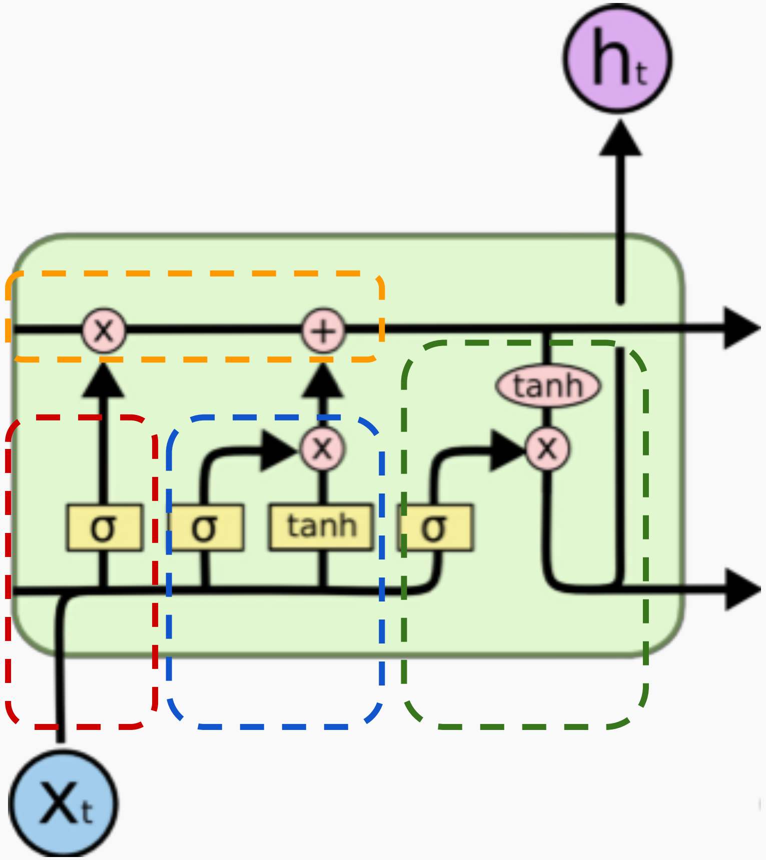
RNN + Memory + Output



LSTM: Long short term memory



Before to really understand LSTM lets see the big picture …



Cell State

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

Output Gate

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

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

Forget Gate

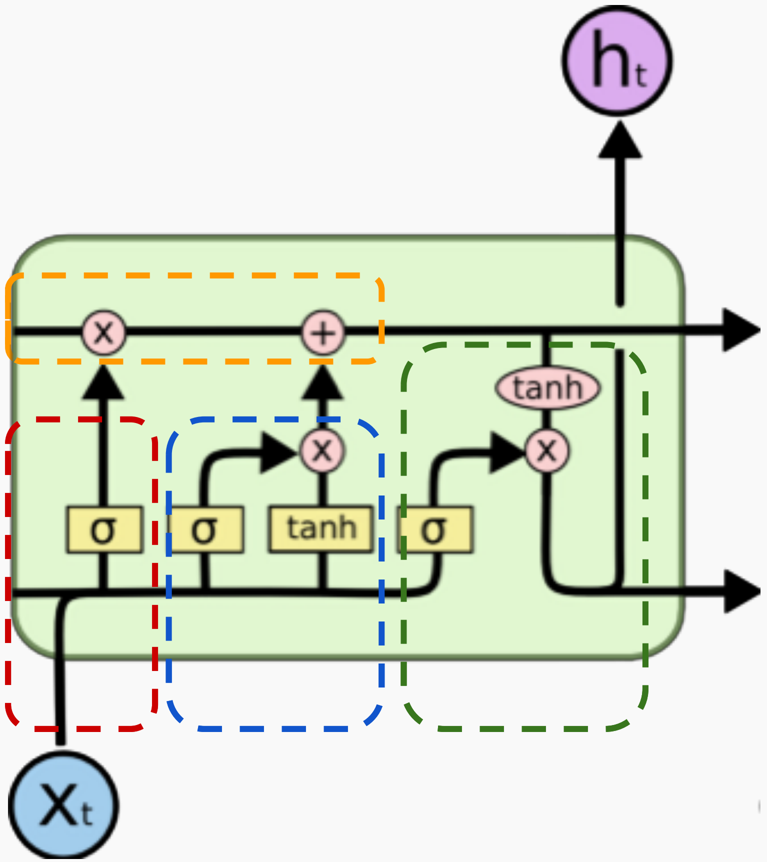
$$f_t = \sigma(W_f \cdot [h_{t-1}, x_{t-1}] + b_f)$$

Input Gate

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

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

Before to really understand LSTM lets see the big picture …



1. LSTM are recurrent neural network with a cell and a hidden state, boths of these are updated in each step and can be thought as memories.

2. Cell states work as a long term memory and the updates depends on the relation between the hidden state in t -1 and the input.

3. The hidden state of the next step is a transformation of the cell state and the output (which is the section that is in general used to calculate our loss, ie information that we want in a short memory).

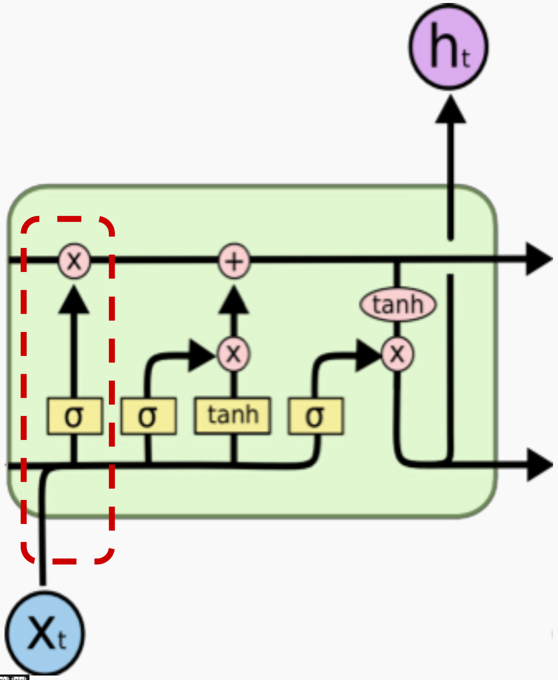
Let's think about my cell state

$$x_t = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} \quad h_{t-1} = \begin{array}{c} 😁 \\ ⏱ \end{array} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad C_{t-1} = \begin{array}{c} 😁 \\ ⏱ \end{array} \begin{bmatrix} 0.7 \\ 0.3 \end{bmatrix}$$

Let's predict if i will help you in the homework in time t

Forget Gate

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



The forget gate tries to estimate what features of the cell state should be forgotten.

$$f_t = \sigma\left(\underbrace{\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} \text{😁} & 1 \\ \text{⏱} & 1 \end{bmatrix}} + \underbrace{\begin{bmatrix} 1 & 1 & -100 \\ 0.1 & 0.1 & -100 \end{bmatrix} \begin{bmatrix} \text{👪} & 1 \\ \text{🕷} & -1 \\ \text{6AM} & 1 \end{bmatrix}} \right)$$

$$\underbrace{f_t = \sigma\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -100 \\ -100 \end{bmatrix} \right)}$$

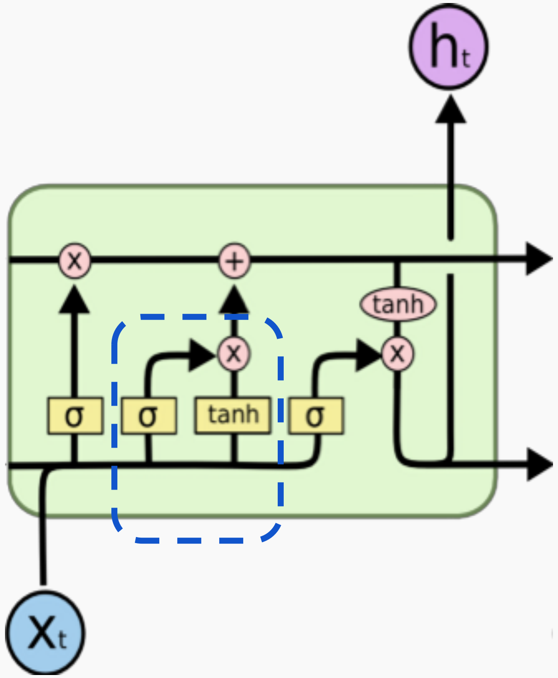
$$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Erase everything!

Input Gate

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

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



The input gate layer works in a similar way that the forget layer, the input gate layer estimate the degree of confidence of $\tilde{C}_t$ and $\tilde{C}_t$ is a new estimation of the cell state.

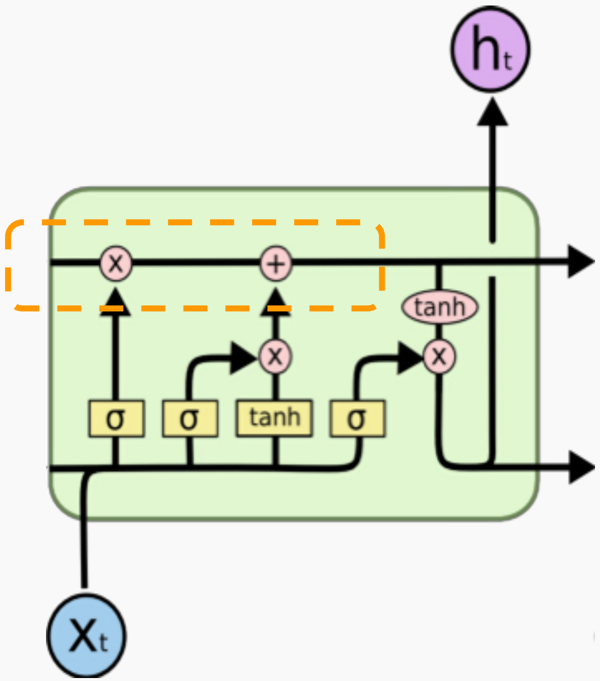
Let's say that my input gate estimation is: $i_t = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

$$\tilde{C}_t = \tanh\left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 10 & 1 & -1 \\ -1 & 1 & 10 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} \right)$$

$$\tilde{C}_t = \tanh \underbrace{\left(\begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 10 \\ 10 \end{bmatrix} \right)}_{\begin{bmatrix} 1 \\ 1 \end{bmatrix}}$$

Cell state

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



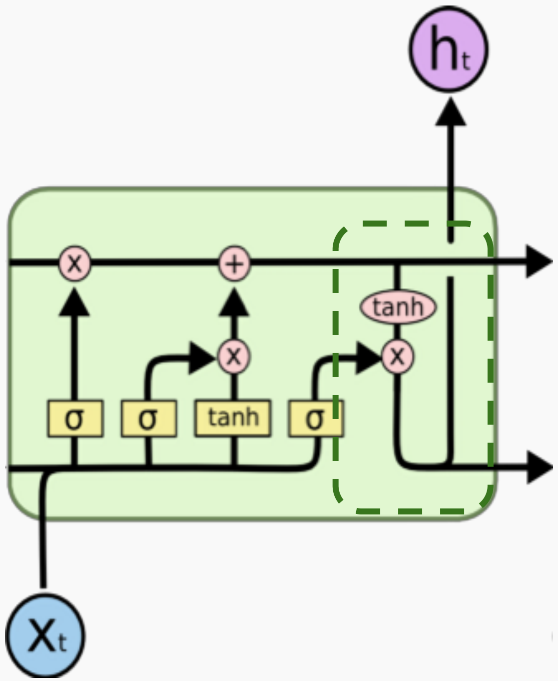
After the calculation of forget gate and input gate we can update our cell state.

$$C_t = \underbrace{\begin{bmatrix}0\\0\end{bmatrix} * \begin{bmatrix}\text{😁} & 0.7\\ \text{⏱} & 0.3\end{bmatrix} + \begin{bmatrix}1\\1\end{bmatrix} * \begin{bmatrix}\text{😁} & 1\\ \text{⏱} & 1\end{bmatrix}}$$

$$C_t = \begin{bmatrix}\text{😁} & 1\\ \text{⏱} & 1\end{bmatrix}$$

Output gate

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



- The output gate layer is calculated using the information of the input x in time t and hidden state of the last step.
- It is important to notice that hidden state used in the next step is obtained using the output gate layer which is usually the function that we optimize.

$$\underbrace{o_t = \sigma\left(\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} \text{😁} & 1 \\ \text{⏱} & 1 \end{bmatrix} + \begin{bmatrix} 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} & 1 \\ & -1 \\ \text{6AM} & 1 \end{bmatrix} \right)}_{o_t \approx 0.9}$$

$$h_t \approx 0.9 * \begin{bmatrix} \text{😁} & 1 \\ \text{⏱} & 1 \end{bmatrix} = \begin{bmatrix} \text{😁} & 0.9 \\ \text{⏱} & 0.9 \end{bmatrix}$$

wcct! = we can calculate this!

To optimize my parameters i basically need to do: $W = W - \eta \frac{\partial \mathcal{L}}{\partial W}$

Let's calculate all the derivatives in some time t!

$$\frac{\partial \mathcal{L}}{\partial W_f} = \frac{\partial \mathcal{L}}{\partial C^t} \underbrace{\frac{\partial C^t}{\partial f^t}}_{C^{t-1}} \overbrace{\frac{\partial f^t}{\partial W_f}}^{\text{wcct!}}$$

$$\frac{\partial \mathcal{L}}{\partial W_c} = \frac{\partial \mathcal{L}}{\partial C^t} \underbrace{\frac{\partial C^t}{\partial (i^t \odot \hat{C}^t)}}_{1} \overbrace{\frac{\partial (i^t \odot \hat{C}^t)}{\partial W_c}}^{\text{wcct!}}$$

$$\frac{\partial \mathcal{L}}{\partial W_i} = \frac{\partial \mathcal{L}}{\partial C^t} \underbrace{\frac{\partial C^t}{\partial (i^t \odot \hat{C}^t)}}_{1} \overbrace{\frac{\partial (i^t \odot \hat{C}^t)}{\partial W_i}}^{\text{wcct!}}$$

$$\frac{\partial \mathcal{L}}{\partial W_o} = \frac{\partial \mathcal{L}}{\partial h^t} \underbrace{\frac{\partial h^t}{\partial o^t}}_{\tanh C^t} \overbrace{\frac{\partial o^t}{\partial W_o}}^{\text{wcct!}}$$

So… every derivative is wrt the cell state or the hidden state

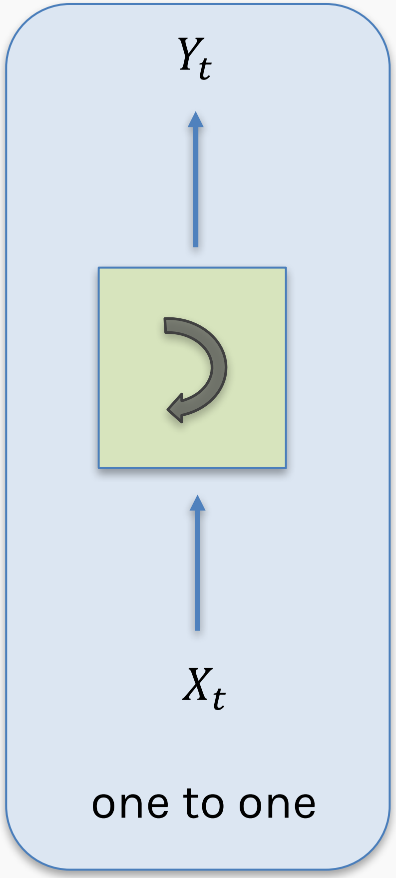
Let’s calculate the cell state and the hidden state

$$\frac{\partial \mathcal{L}}{\partial h^{t-1}} = \frac{\partial \mathcal{L}}{\partial C^t} \left(\frac{\partial C^t}{\partial f^t} \frac{\partial f^t}{\partial h^t} + \frac{\partial C^t}{\partial (i^t \odot \hat{C}^t)} \frac{\partial (i^t \odot \hat{C}^t)}{\partial h^t} \right) + \frac{\partial \mathcal{L}}{\partial h^t} \frac{\partial h^t}{\partial o^t} \frac{\partial o^t}{\partial h^{t-1}}$$

$$\frac{\partial \mathcal{L}}{\partial C^t} = \underbrace{\frac{\partial \mathcal{L}}{\partial (f^{t+1} \odot C^t + i^{t+1} \odot \hat{C}^t)}} \underbrace{\frac{\partial (f^{t+1} \odot C^t + i^{t+1} \odot \hat{C}^t)}{\partial C^t}}$$

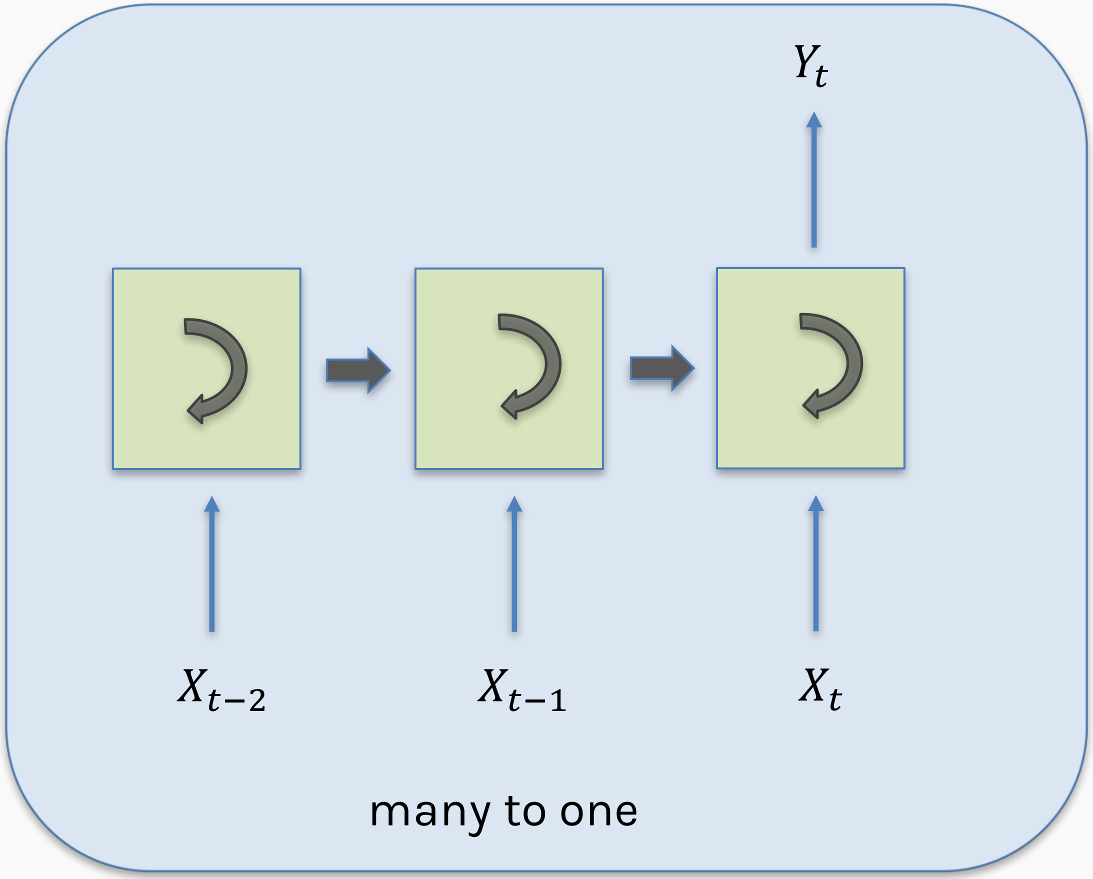
$$\left(\frac{\partial \mathcal{L}}{\partial C^{t+1}} + \frac{\partial \mathcal{L}}{\partial h^{t+1}} \frac{\partial h^{t+1}}{\partial C^{t+1}} \right) \odot f^{t+1}$$

RNN Structures



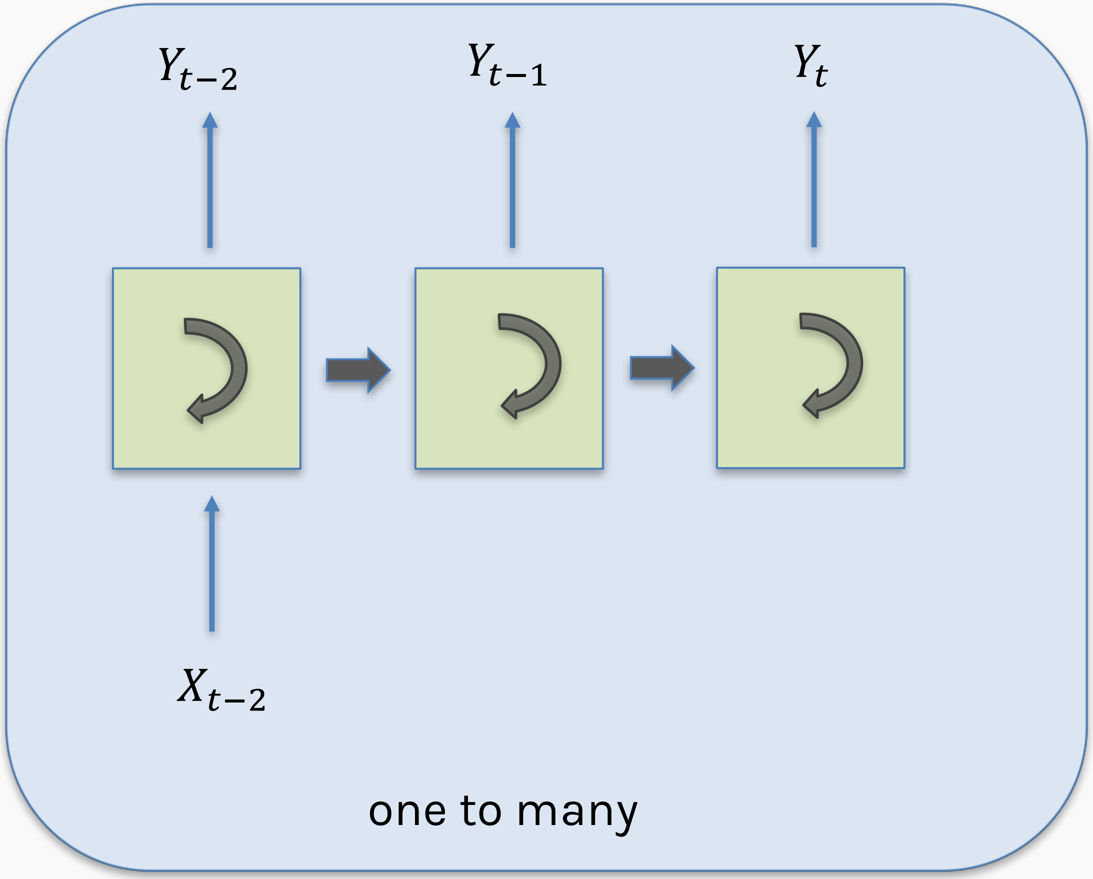
- The **one to one** structure is useless.
- It takes a single input and it produces a single output.
- Not useful because the RNN cell is making little use of its unique ability to remember things about its input sequence

RNN Structures (cont)



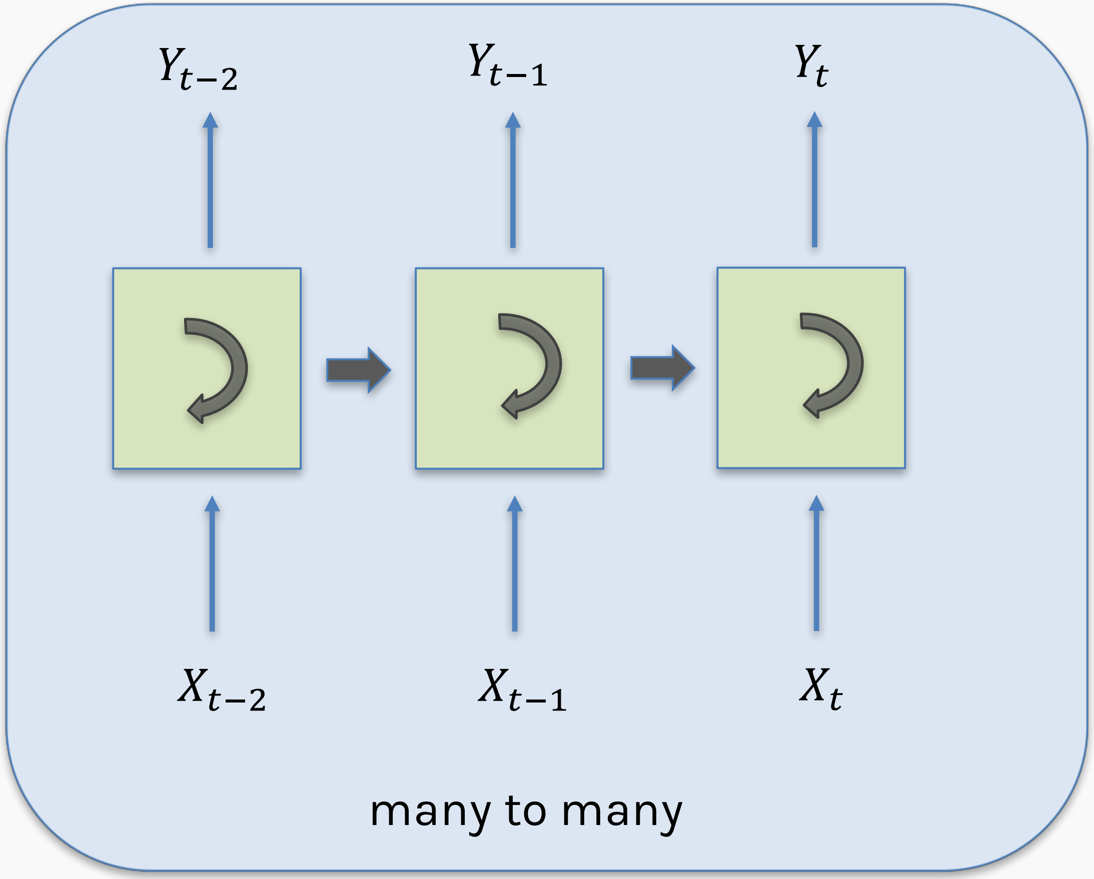
The **many to one** structure reads in a sequence and gives us back a single value. Example: Sentiment analysis, where the network is given a piece of text and then reports on some quality inherent in the writing. A common example is to look at a movie review and determine if it was positive or negative.

RNN Structures (cont)



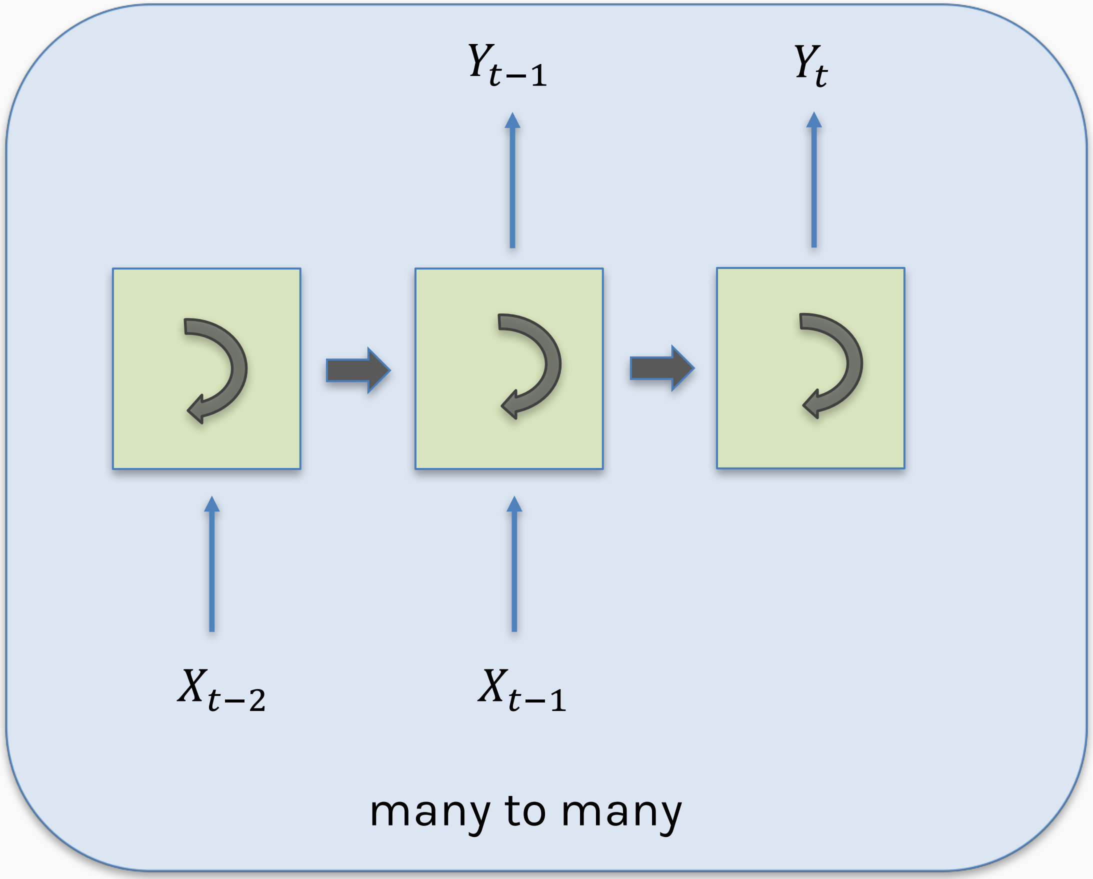
The **one to many** takes in a single piece of data and produces a sequence.
For example we give it the starting note for a song, and the network produces the rest of the melody for us.

RNN Structures (cont)



The **many to many** structures are in some ways the most interesting. used for machine translation.
Example: Predict if it will rain given some inputs.

RNN Structures (cont)



This form of **many to many** can be used for machine translation.

For example, the English sentence:
"The black dog jumped over the cat"
In Italian as:
"Il cane nero saltò sopra il gatto"
In the Italia, the adjective "nero" (black) follows the noun "cane" (dog), so we need to have some kind of buffer so we can produce the words in their proper English.

Bidirectional

LSTM and RNN are designed to analyze sequence of values.

For example: *Patrick said he needs a vacation.*

he here means *Patrick* and we know this because *Patrick* was before the word *he.*

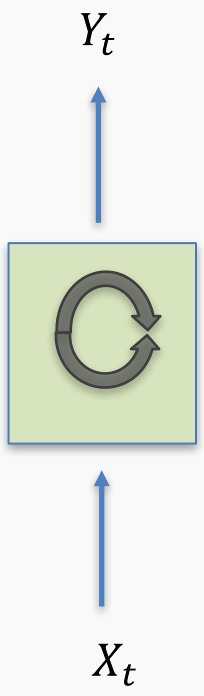
However consider the following sentence:

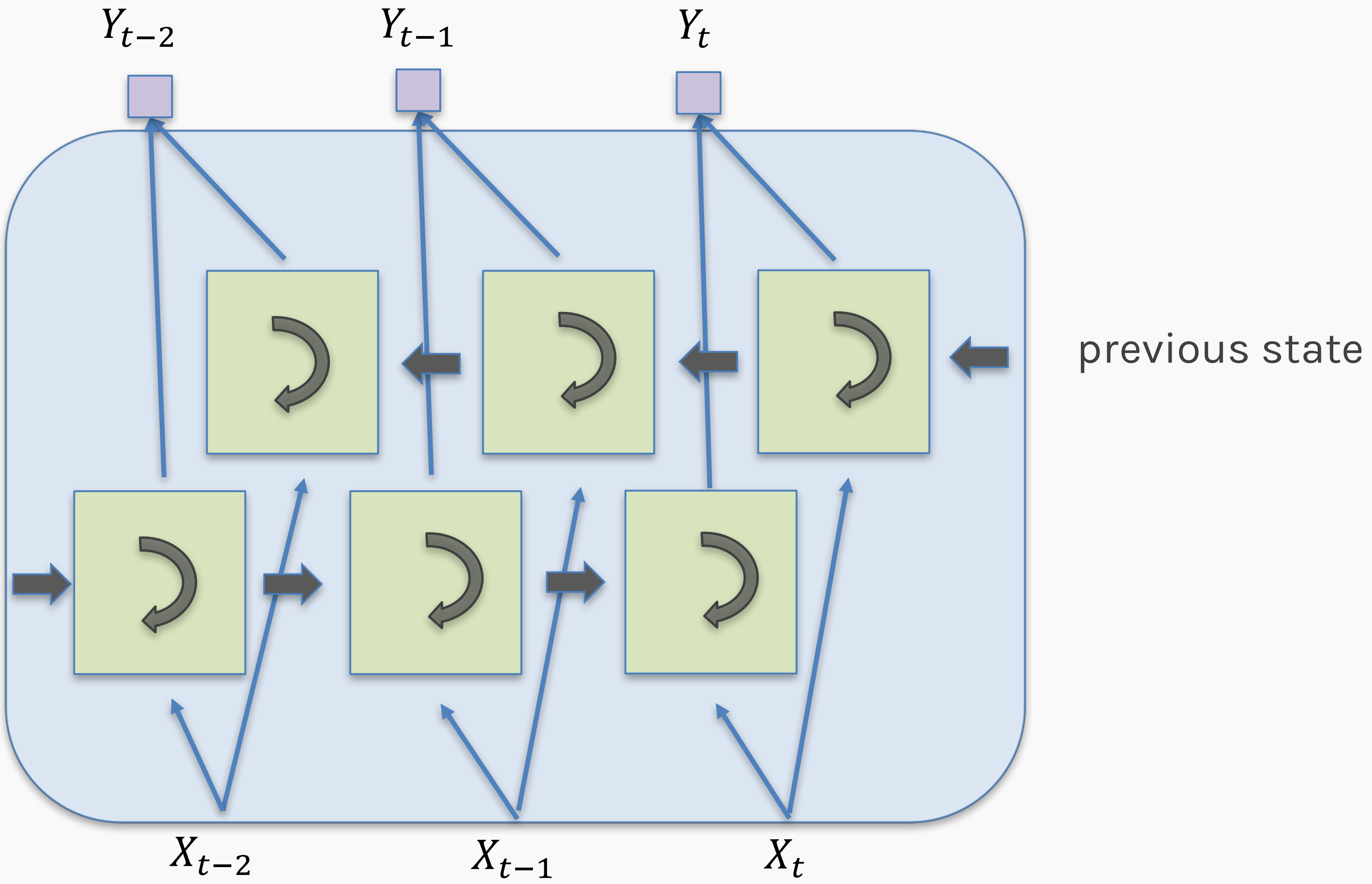
He needs to work more, Pavlos said about Patrick.

Bidirectional RNN or BRNN or bidirectional LSTM or BLSTM when using LSTM units.

Bidirectional (cond)

symbol for a BRNN



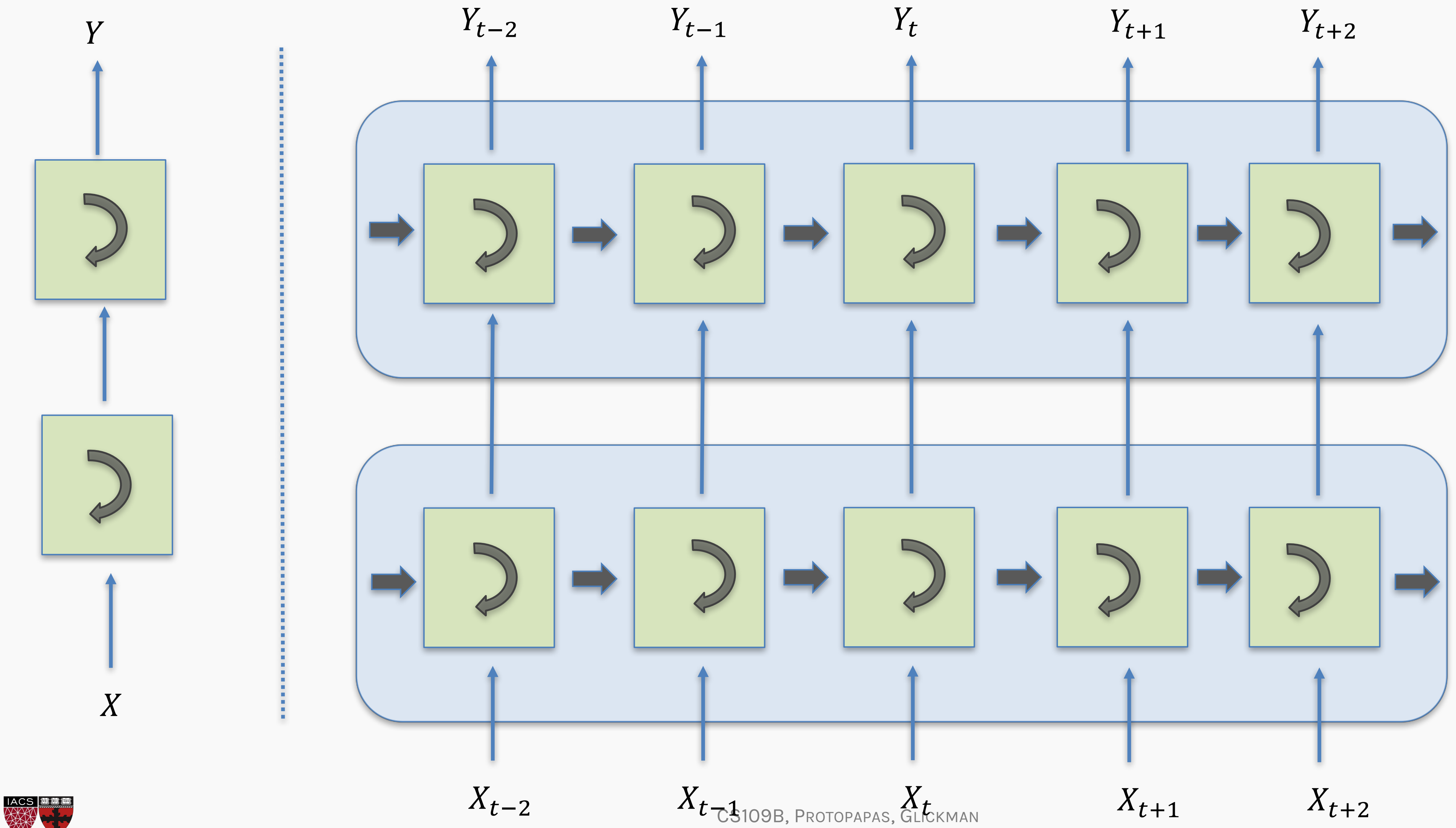


Deep RNN

LSTM units can be arranged in layers, so that each the output of each unit is the input to the other units. This is called **a deep RNN**, where the adjective “deep” refers to these multiple layers.

- Each layer feeds the LSTM on the next layer
- First time step of a feature is fed to the first LSTM, which processes that data and produces an output (and a new state for itself).
- That output is fed to the next LSTM, which does the same thing, and the next, and so on.
- Then the second time step arrives at the first LSTM, and the process repeats.

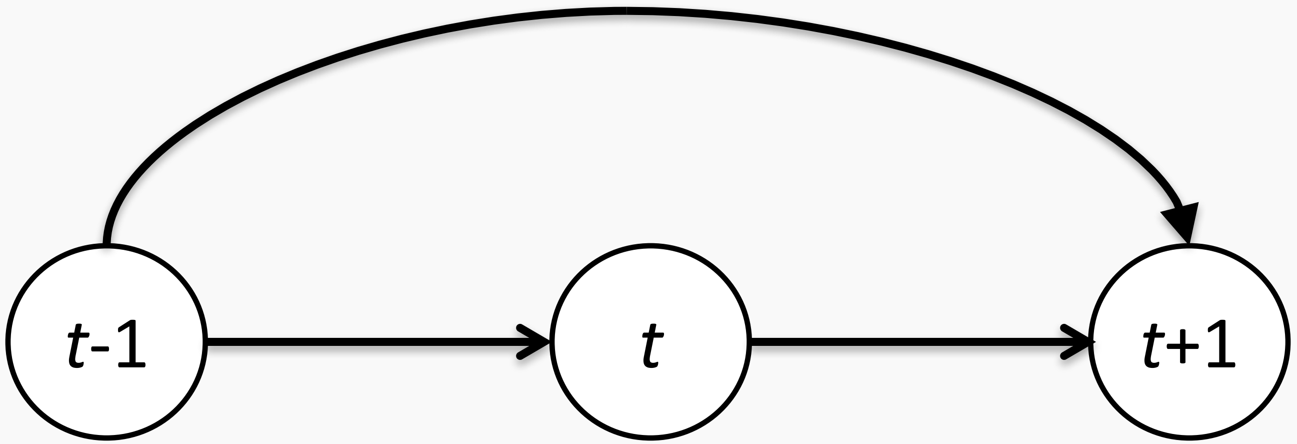
Deep RNN



Skip Connections

Add additional connections between units *d* time steps apart

Creating paths through time where gradients neither vanish or explode

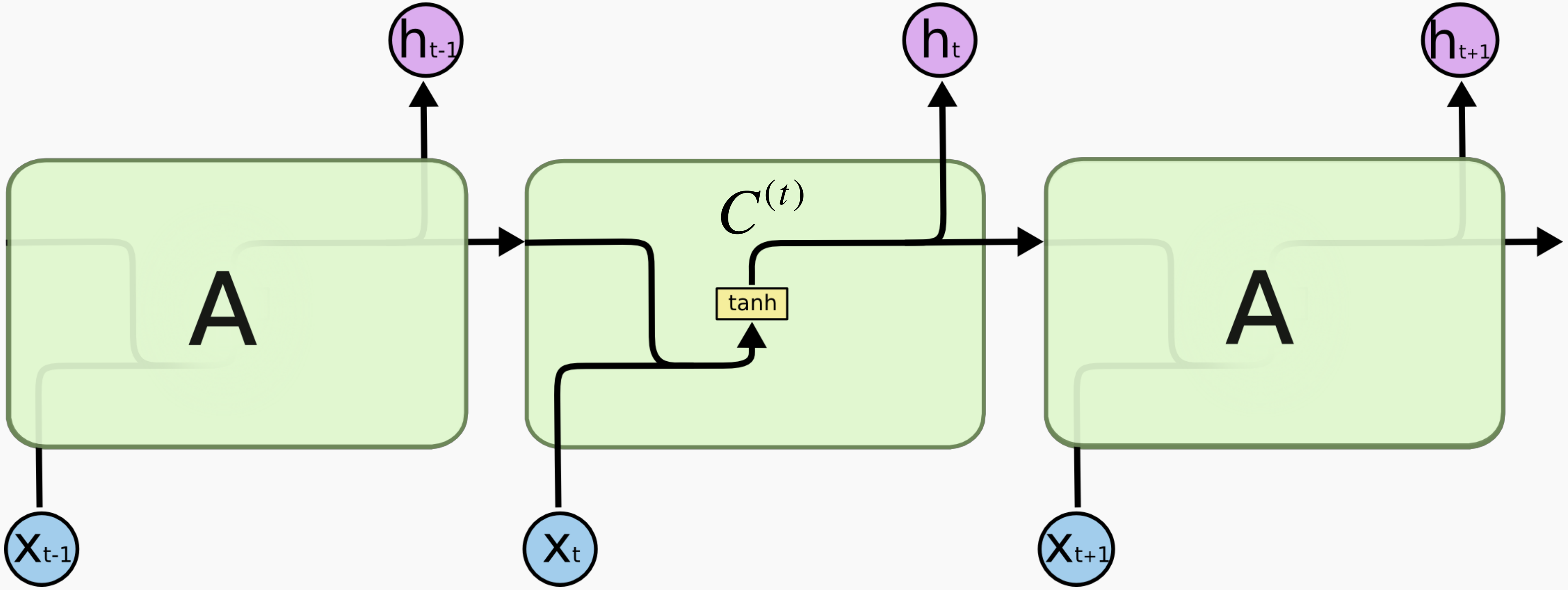


Leaky Units

Linear self-connections

Maintain cell state: running average of past hidden activations

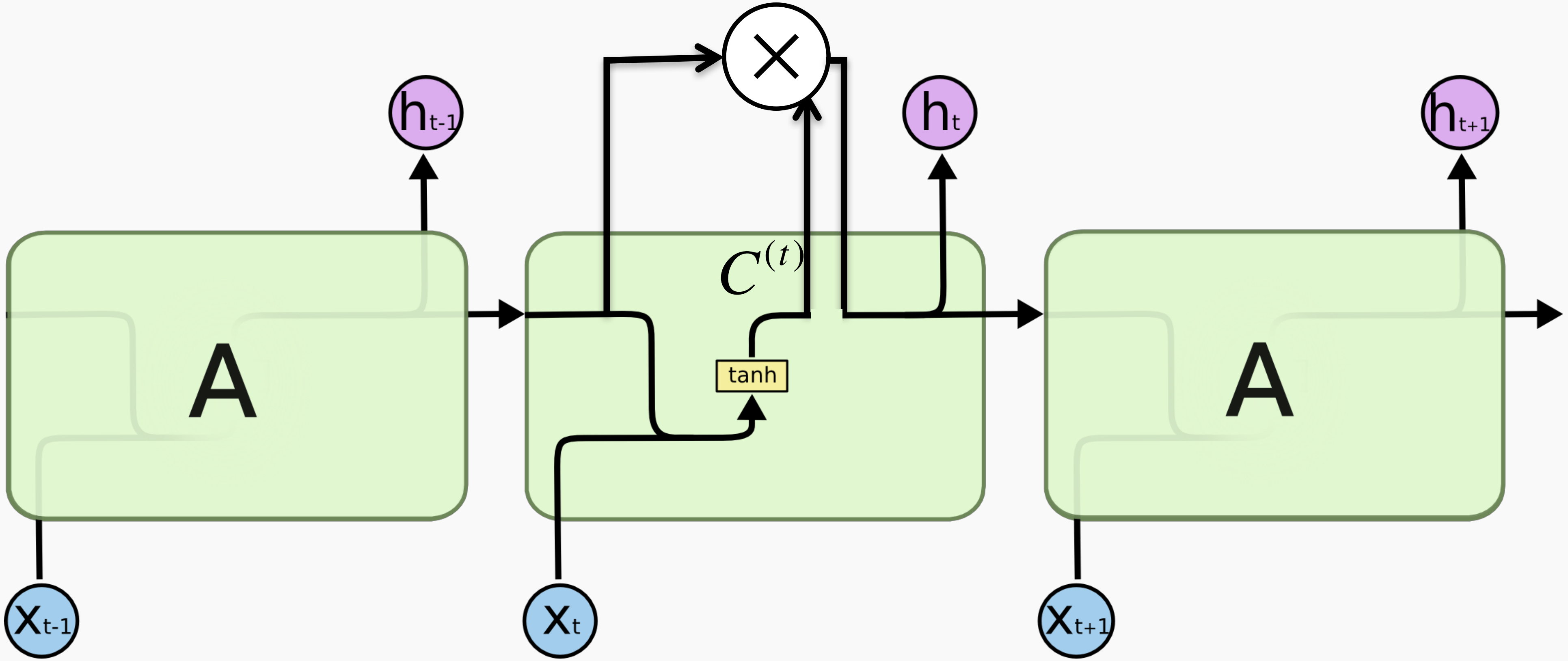
Standard RNN



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

$$h^{(t)} = C^{(t)}$$

Leaky Unit



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

$$h^{(t)} = \alpha h^{(t-1)} + (1\text{-}\alpha)C^{(t)}$$

colah.github.io