

COMP 472: Artificial Intelligence Natural Language Processing per + 5 Recurrent Neural Networks video 6

- Russell & Norvig: Section 21.6, 24.2

Today

1. Introduction
2. Bag of word model
3. n-gram models
4. Deep Learning for NLP
 1. Word Embeddings
 2. Recurrent Neural Networks



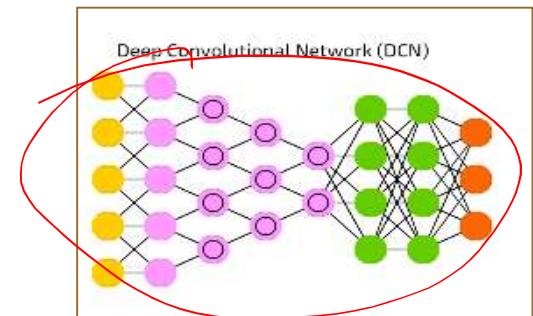
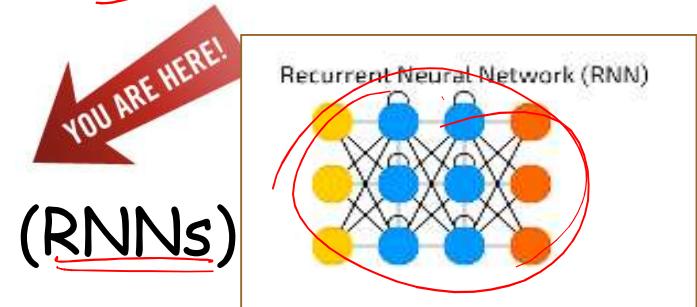
Deep Learning for NLP

Deep learning models for NLP use:

- Vector representation of words
 - i.e., word embeddings

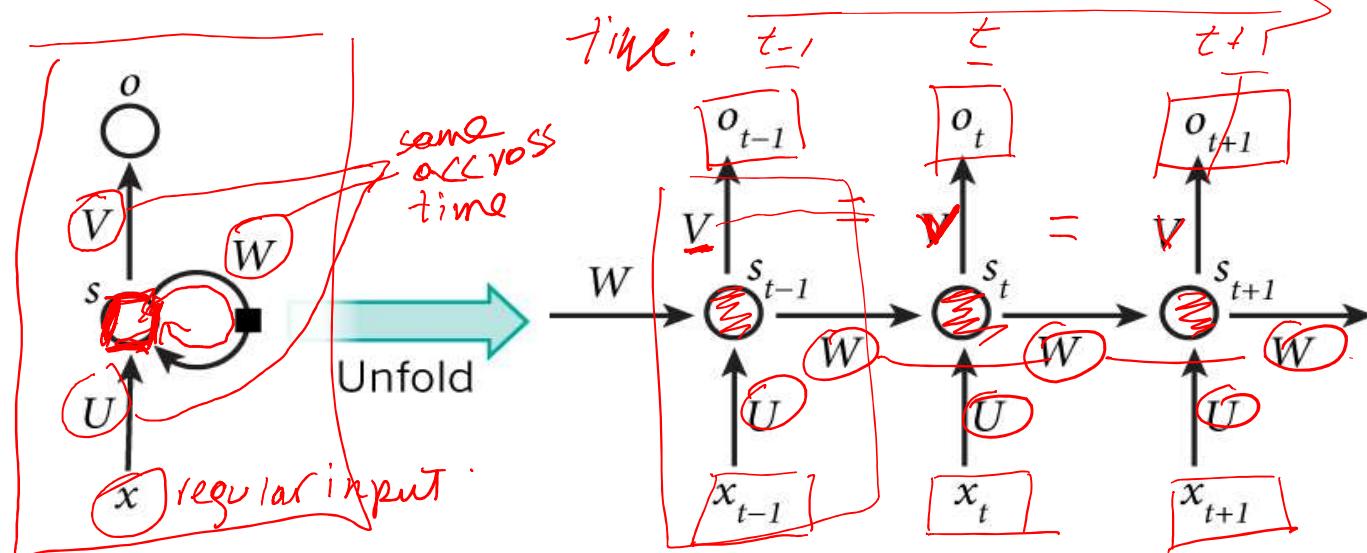
- Neural network structures

- Recurrent Neural Networks (RNNs)
 - Recursive Neural Networks
 - Convolutional Networks (CNNs)
 - ...



Recurrent Neural Networks

- To model sequences of decisions, such as machine translation, language modelling,
 - eg. A word at position n can influence a word/decision at position $n+t$
- decision/output from the past can influence current decision/output
- Networks with loops in them, allowing information to persist.

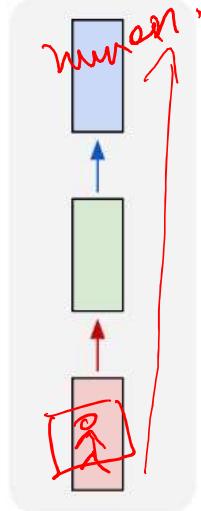


Recurrent Neural Networks

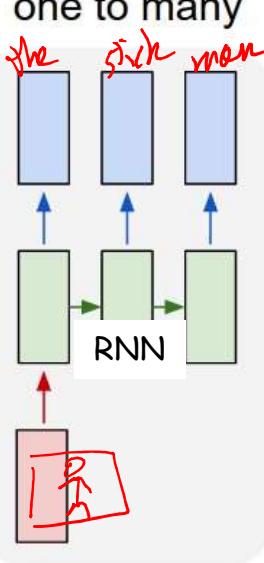
- To model data:
 - with temporal or sequential structures
 - eg. A word at position n can influence a word/decision at position $n+t$
 - and, varying length of inputs and outputs

Applications of RNNs

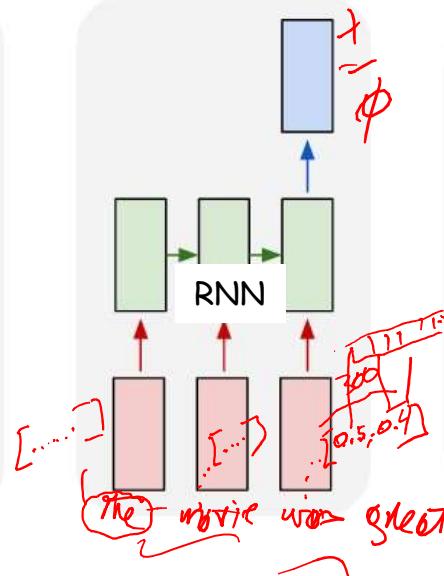
one to one



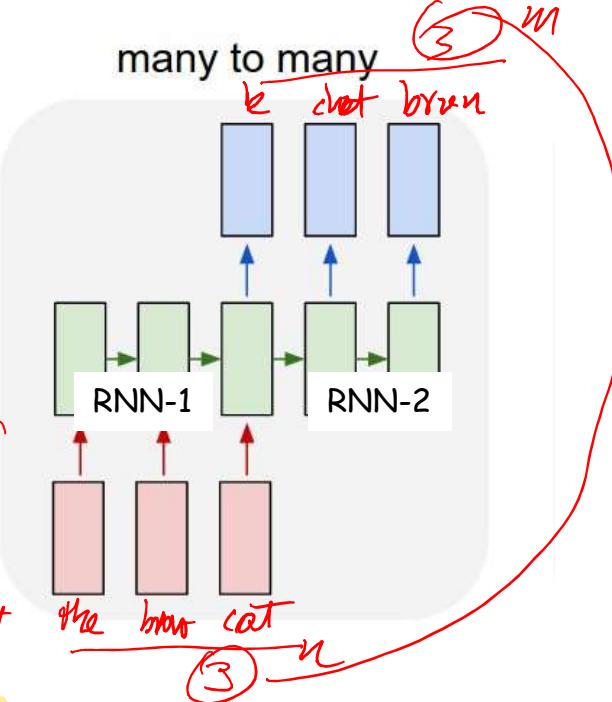
one to many



many to one



many to many



FNN

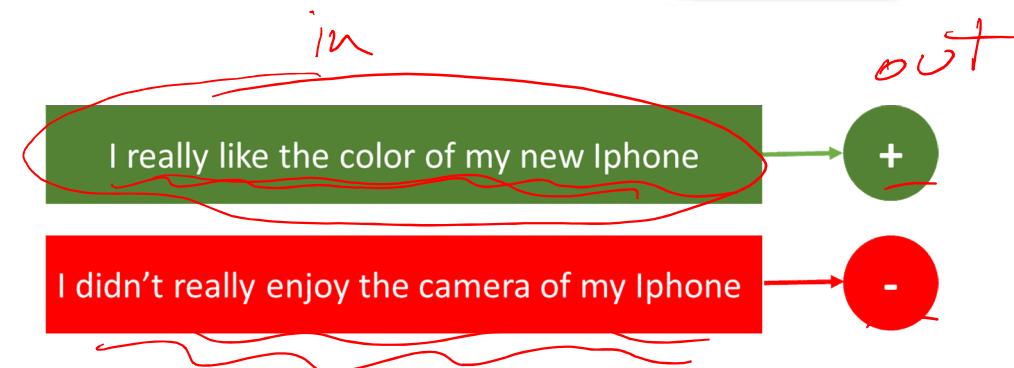
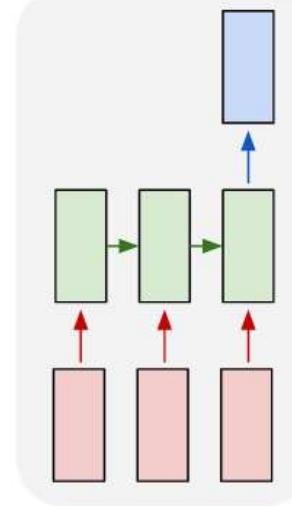
- (1) Vanilla ANN -- input: 1 unit, output: 1 unit (e.g. image classification)
- (2) RNN - input: 1 unit, output: sequence (e.g. image captioning)
- (3) RNN - input: sequence, output: 1 unit (e.g. sentiment analysis)
- (4) RNN - input: sequence, output: sequence (e.g. Machine Translation)

Applications of RNNs

1. input: sequence of n items
output: 1 item

- eg. language modelling / classification:
- input: sequence of n words
- output: 1 class

many to one



Applications of RNNs

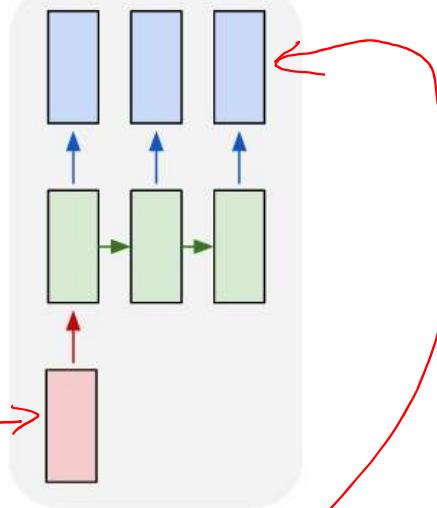
2. input: 1 item
output: sequence of n items

- eg. image captioning
- input: 1 image
- output: sequence of n words



A group of young people
playing a game of frisbee.

one to many



Applications of RNNs

3. input: sequence of n items
output: sequence of m items

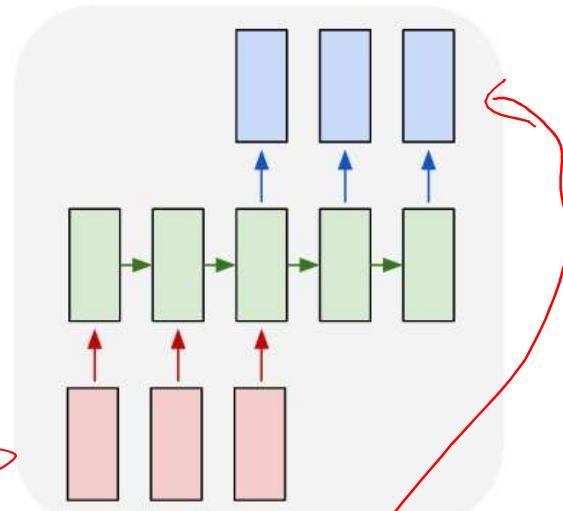
- eg. machine translation:
- input: sequence of n words
- output: sequence of m words

French was the official language of the colony of French Indochina, comprising modern-day Vietnam, Laos, and Cambodia. It continues to be an administrative language in Laos and Cambodia, although its influence has waned in recent years.

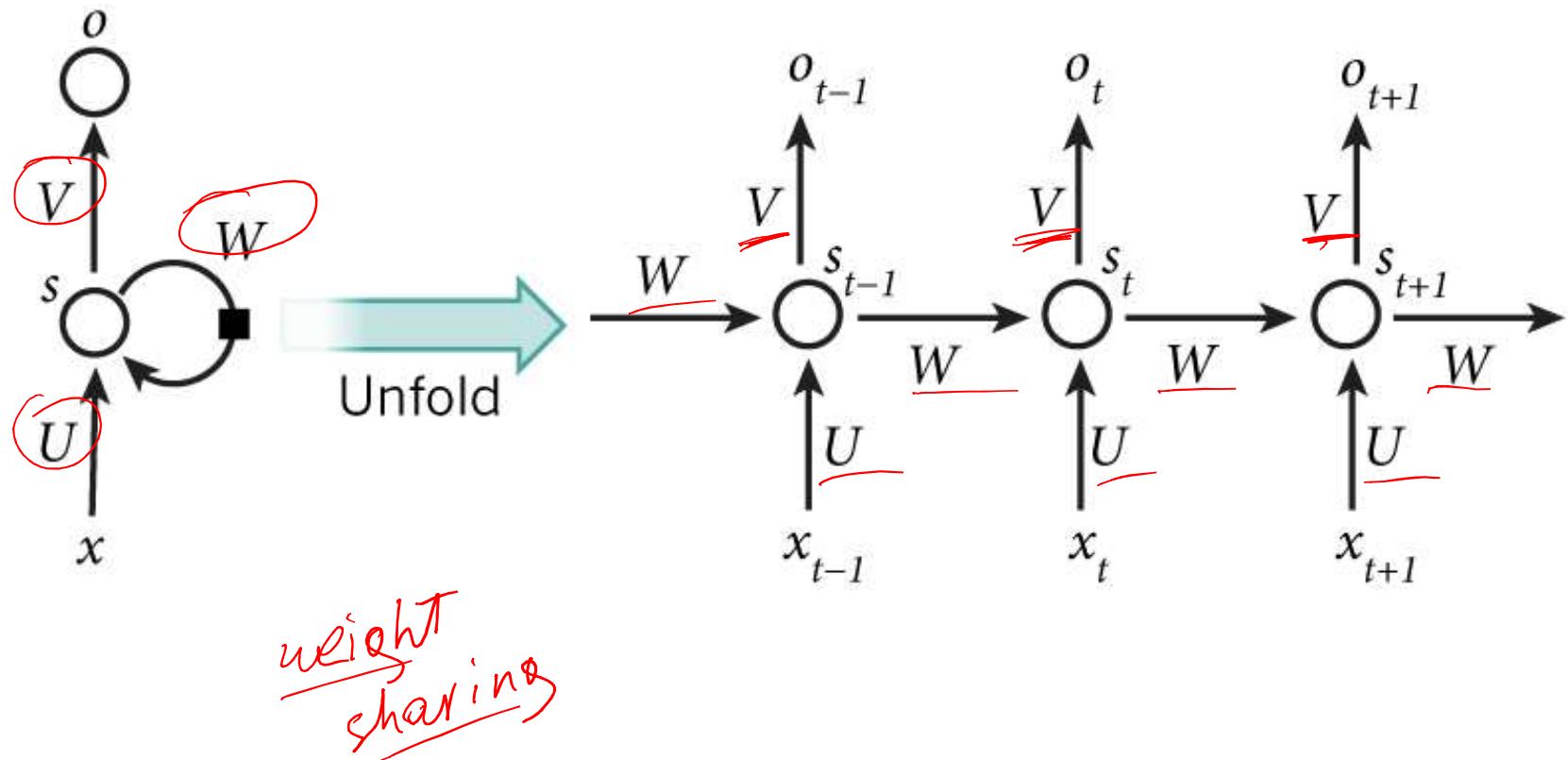
Translate into : French

Le français était la langue officielle de la colonie de l'Indochine française, comprenant le Vietnam d'aujourd'hui, le Laos et le Cambodge. Il continue d'être une langue administrative au Laos et au Cambodge, bien que son influence a décliné au cours des dernières années.

many to many



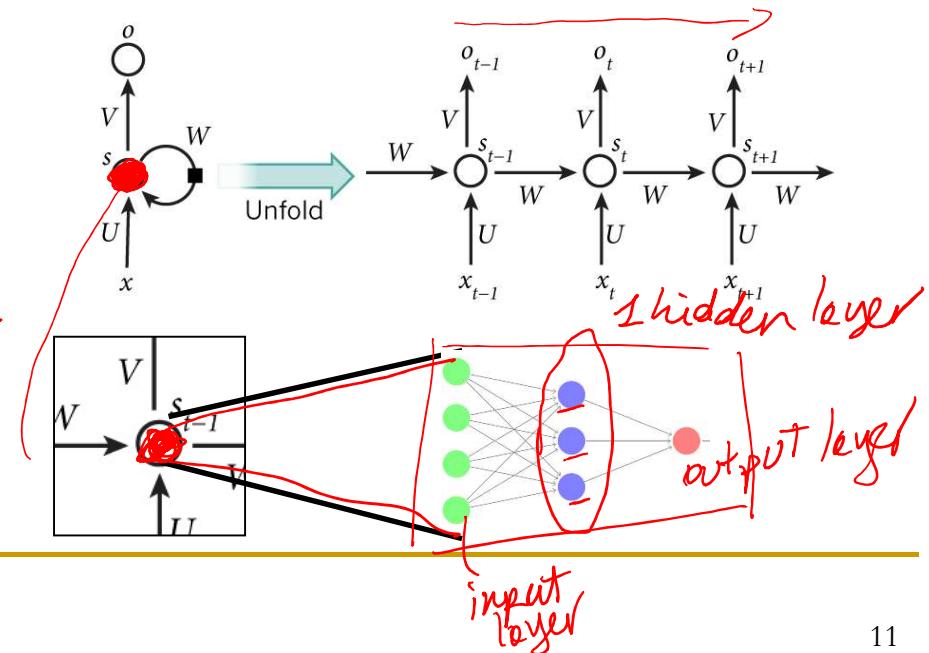
Vanilla RNN



RNN Walk through Example

Task : Character-based Language Modeling

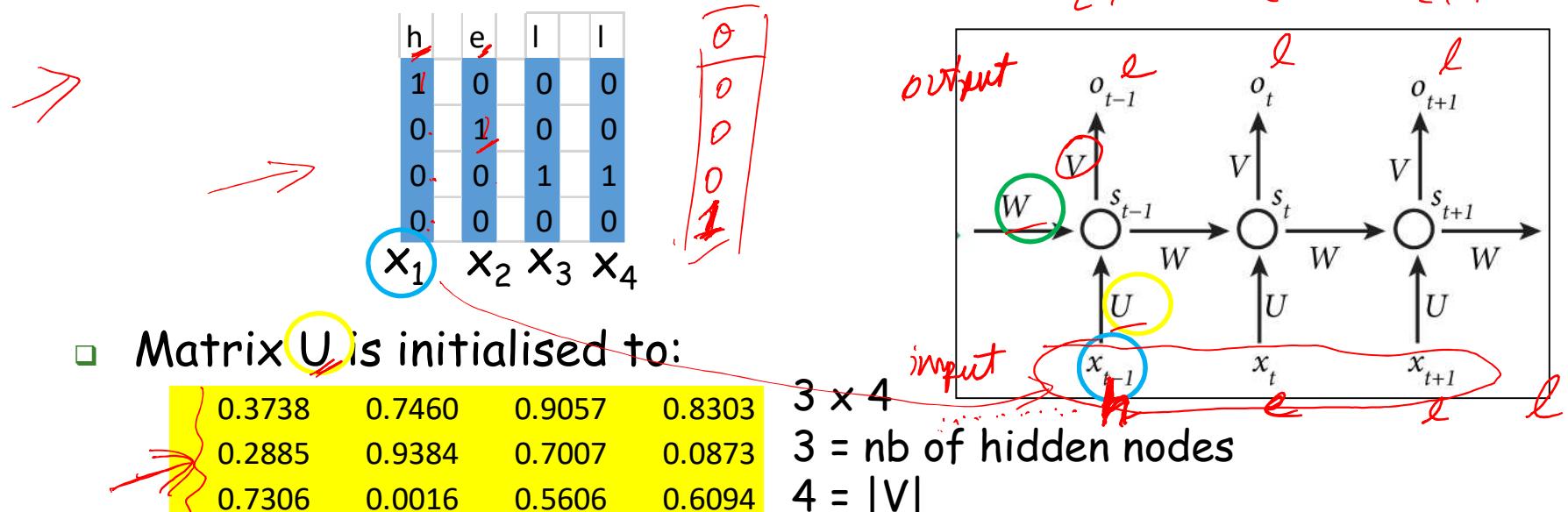
- input: We provide the first 4 letters: h,e,l,l
- output: The network has to predict the 'o'.
- assume:
 - $V = \{h, e, l, o\}$ *4 possible letters only*
 - nb of hidden layers = 1
 - nb of hidden nodes = 3



RNN Walk through: Set-up

- Assume:

- We feed the network with 1-hot vectors



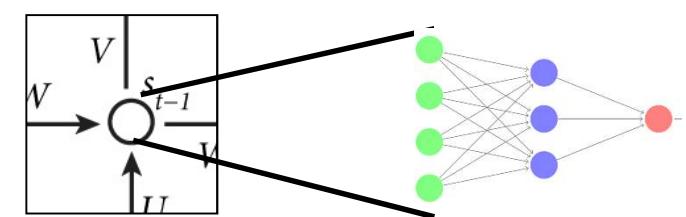
- Matrix U is initialised to:

0.3738	0.7460	0.9057	0.8303
0.2885	0.9384	0.7007	0.0873
0.7306	0.0016	0.5606	0.6094

- Matrix W is initialised to:

0.2464	0.1306	0.4511
0.7445	0.7055	0.6592
0.5034	0.4701	0.9174

3 x 3 3 = nb of hidden nodes



RNN Walk through: Set-Up

- Assume:

- Matrix V is initialised to:

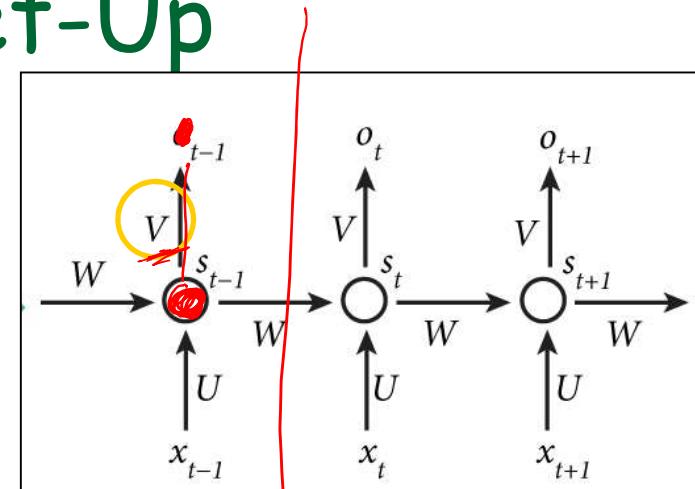
random

0.7170	0.0616	0.7425
0.5761	0.5914	0.7324
0.4216	0.3439	0.5849
0.9257	0.8114	0.0483

4×3

$4 = |V|$

$3 = \text{nb of hidden nodes}$

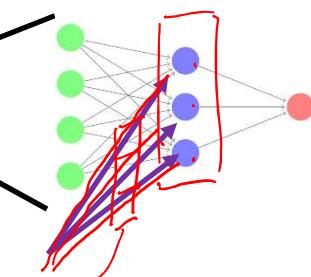
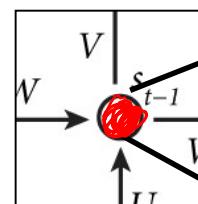


- Bias weights into hidden nodes are initialised to:

- Bias weights into output nodes are initialised to:

random

0.6917
0.1565
0.1411



random

0.4567
0.7654
0.3456
0.0013

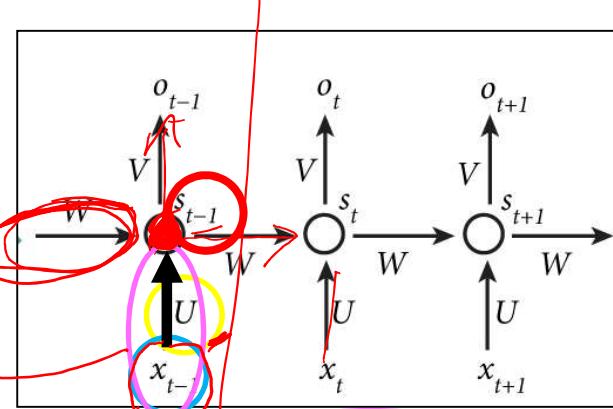
RNN walk through: Feedforward

he // o

Step 1: compute s_1 -part-a for input $x_1 = "h"$,

$$s_t\text{-part-a} = U^*x_t$$

port-b.



U	0.3738	0.7460	0.9057	0.8303
0.2885	0.9384	0.7007	0.0873	
0.7306	0.0016	0.5606	0.6094	

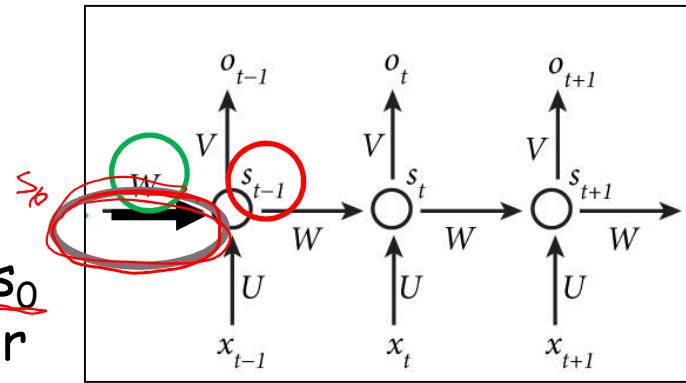
$$\begin{matrix} \times & \begin{matrix} x_1 \\ 1 \\ 0 \\ 0 \\ 0 \end{matrix} & = & \begin{matrix} S_1\text{-part-a} \\ 0.3738 \\ 0.2885 \\ 0.7306 \end{matrix} \end{matrix}$$

RNN walk through: Feedforward

Step 2: compute $s_1\text{-part-b}$

$$s_t\text{-part-b} = W^* s_{t-1}$$

- For the 1st input, the previous state s_0 = [0,0,0] since there is no letter prior to it.



$$\begin{matrix} W \\ \hline 0.2464 & 0.1306 & 0.4511 \\ 0.7445 & 0.7055 & 0.6592 \\ 0.5034 & 0.4701 & 0.9174 \end{matrix} \times \begin{matrix} s_0 \\ \hline 0 \\ 0 \\ 0 \end{matrix} = \begin{matrix} s_1\text{-part-b} \\ \hline 0 \\ 0 \\ 0 \end{matrix}$$

RNN walk through: Feedforward

Step 3: compute s_1

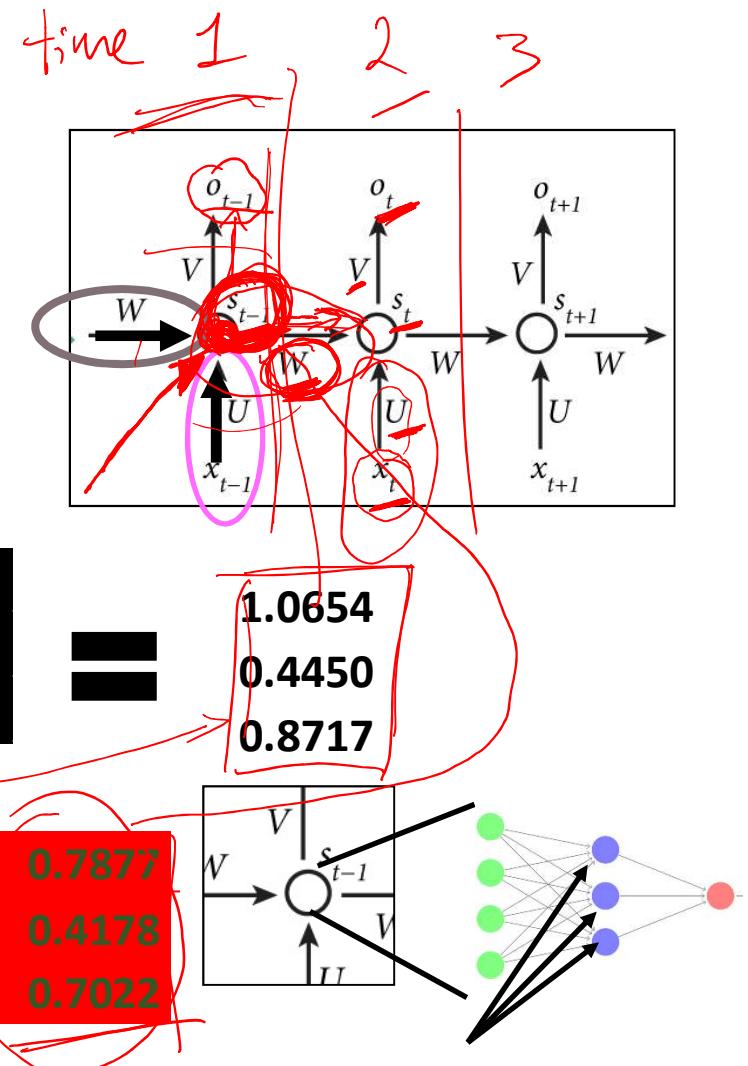
- Combine the 2 parts to get the value of hidden state s_1

$$s_t = \tanh(U^*x_t + W^*s_{t-1} + \text{bias})$$

$s_1\text{-part-a}$ $s_1\text{-part-b}$ bias

0.3738	0	0.6917
0.2885	0	0.1565
0.7306	0	0.1411

$s_1 = \tanh(1.0654, 0.4450, 0.8717)$



RNN walk through: Feedforward

Step 4: compute O_1

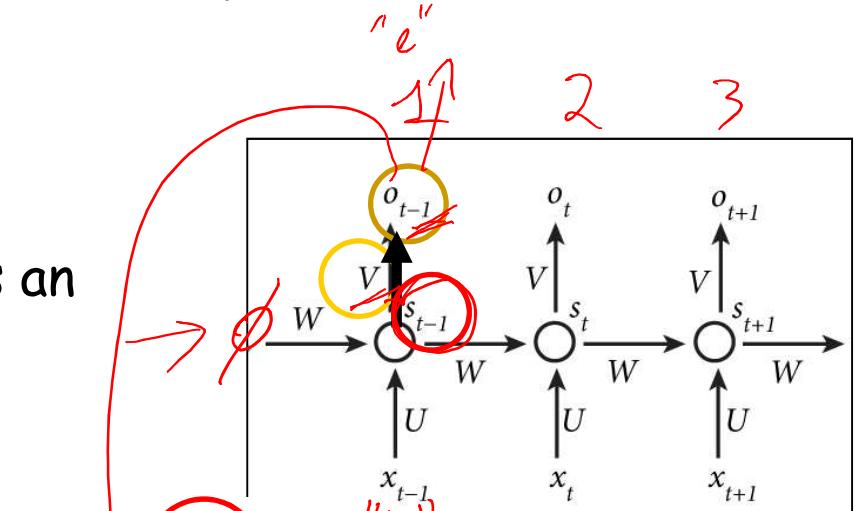
- At each state, the RNN produces an output O_t as well.

$$O_t = \text{softmax}(V * s_t + \text{bias})$$

softmax(

V

0.7170	0.0616	0.7425
0.5761	0.5914	0.7324
0.4216	0.3439	0.5849
0.9257	0.8114	0.0483



$$\text{softmax}(\underbrace{\begin{pmatrix} 0.7877 \\ 0.4178 \\ 0.7022 \end{pmatrix}}_{\times} \underbrace{\begin{pmatrix} 0.4567 \\ 0.7654 \\ 0.3456 \\ 0.0013 \end{pmatrix}}_{+})$$

$$= \text{softmax}(\begin{pmatrix} 1.5687 \\ 1.9806 \\ 1.2321 \\ 1.2034 \end{pmatrix}) =$$

$$\boxed{\begin{pmatrix} 0.255241 \\ 0.385333 \\ 0.182292 \\ 0.177134 \end{pmatrix}}$$

$\Sigma = 1$

RNN walk through: Feedforward

he do

Output at
time $t=1$ is:

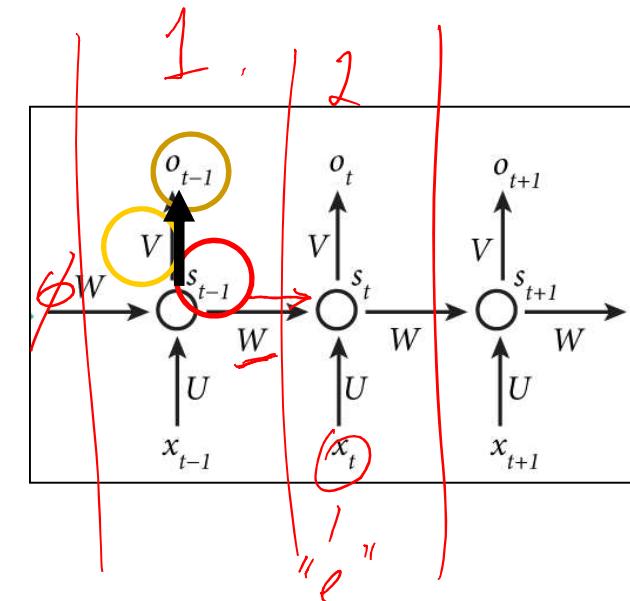
O_1

0.255241
0.385333
0.182292
0.177134

$\Sigma = 1$

Recall that V was:

h	e	i	o
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1



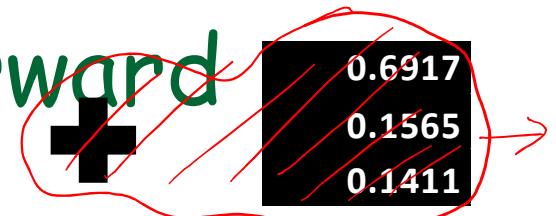
So, at time $t=1$:

- the network outputs "e"
- And the target was "e"

Error = 0



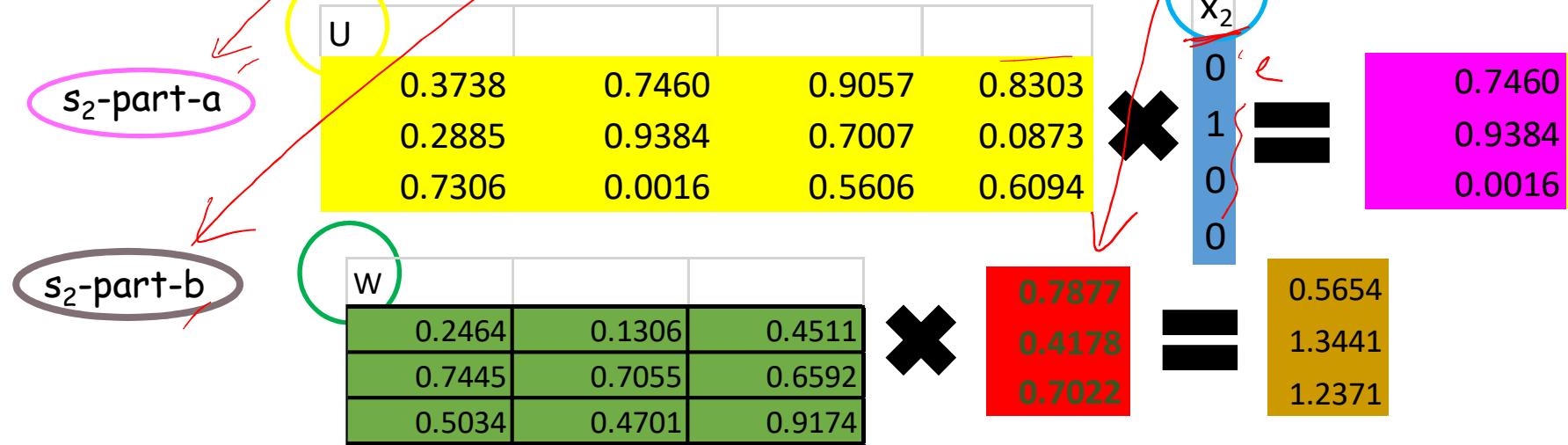
RNN walk through: Feedforward



and we continue feedforwarding:

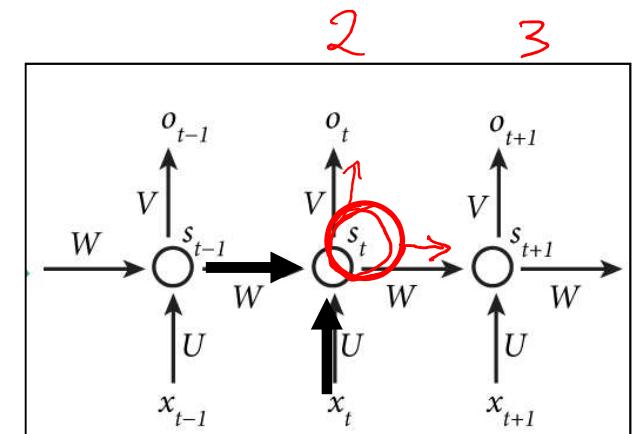
Let's go on to the next state with $x_2 = \text{"e"}$

$$s_2 = \tanh(U^*x_2 + W^*s_1 + \text{bias})$$



RNN walk through: Feedforward

$$s_2 = \tanh(U^*x_2 + W^*s_1 + \text{bias})$$



$$S_2 = \tanh($$

$s_1\text{-part-b}$	
0.7460	+
0.9384	+
0.0016	

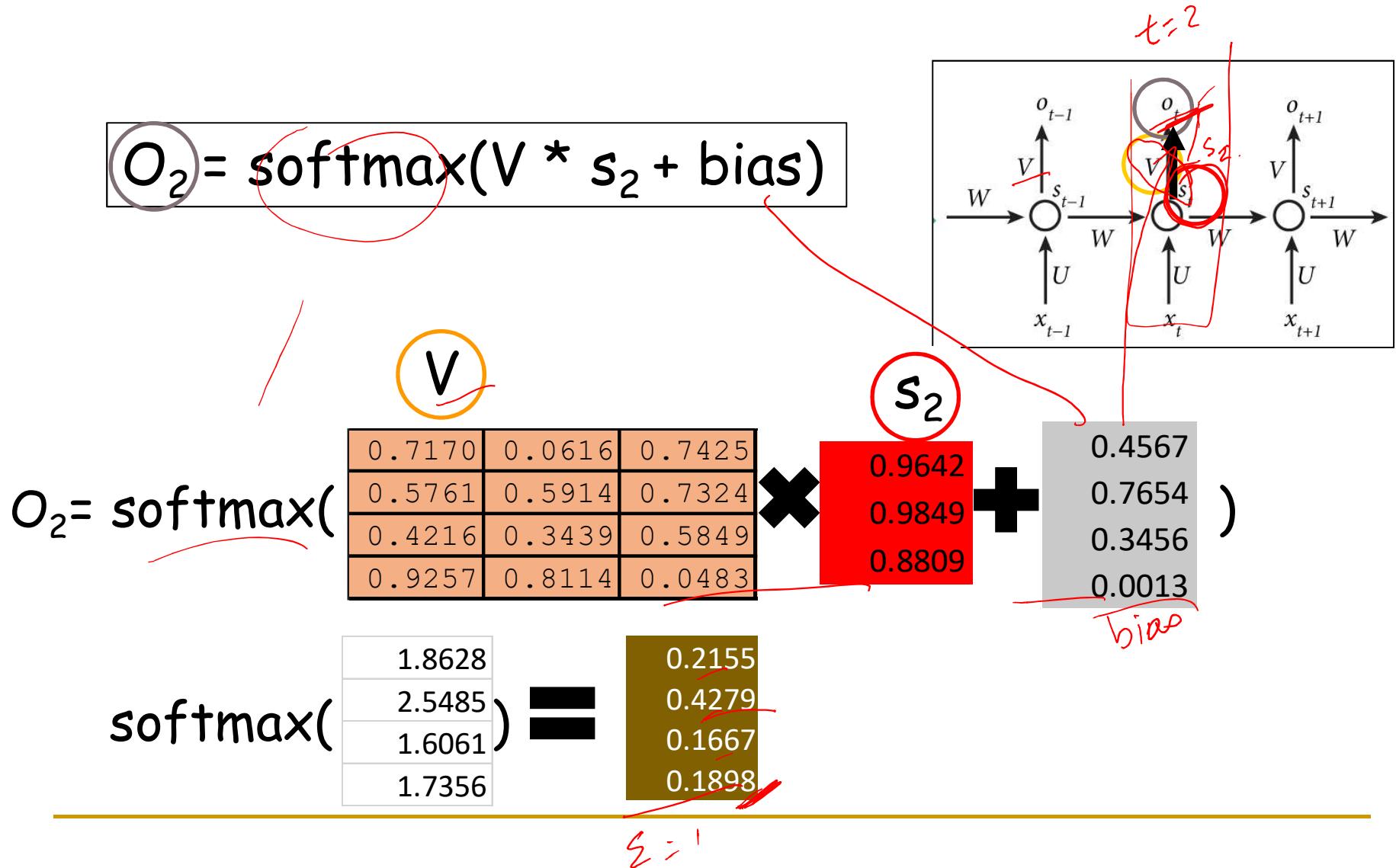
$s_1\text{-part-a}$	
0.5654	+
1.3441	+
1.2371	

bias	
0.6917	=
0.1565	=
0.1411	

 $) =$

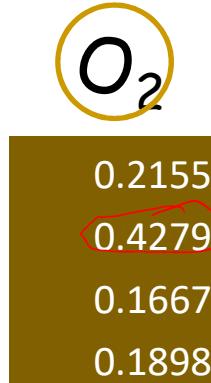
0.9642
0.9849
0.8809

RNN walk through: Feedforward



RNN walk through: Feedforward

Output at time $t=2$ is:



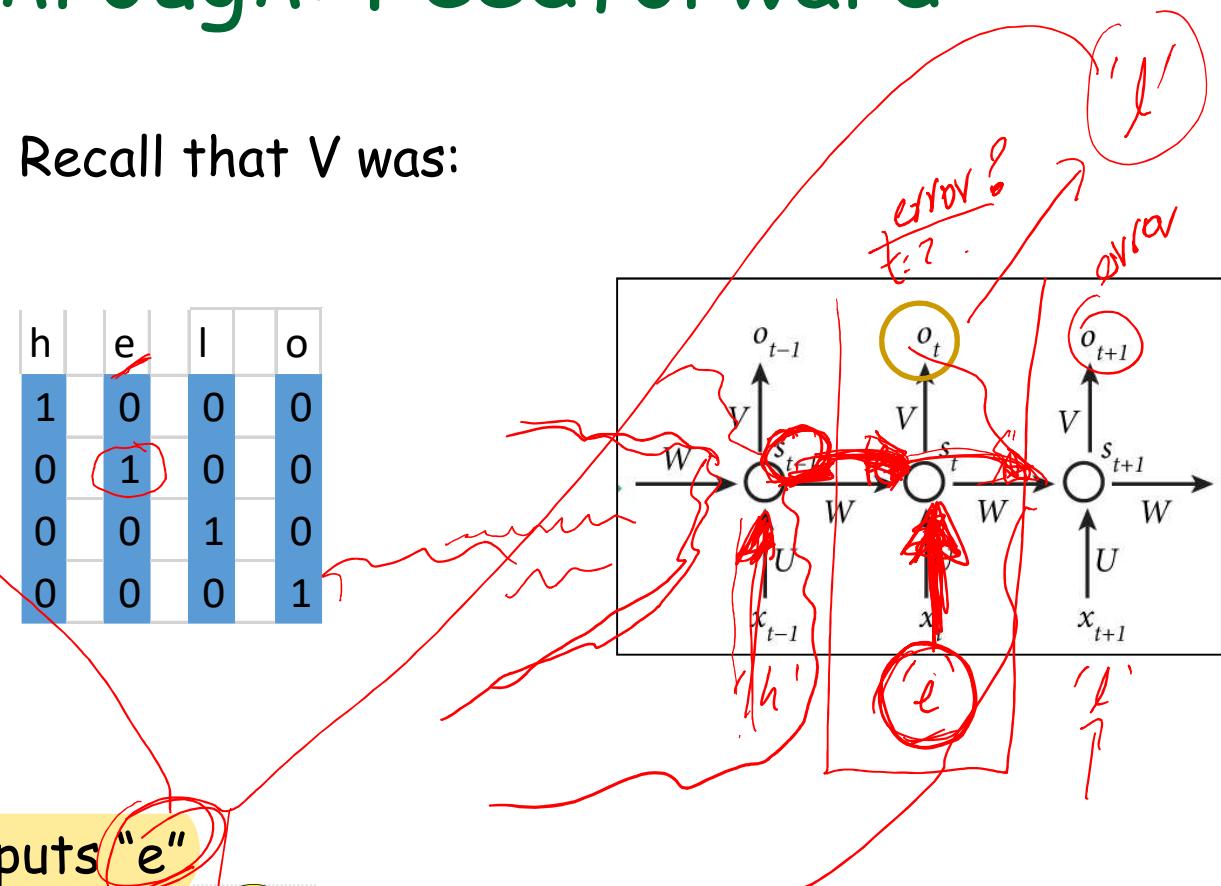
Recall that V was:

h	e	i	o
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

So, at time $t=2$:

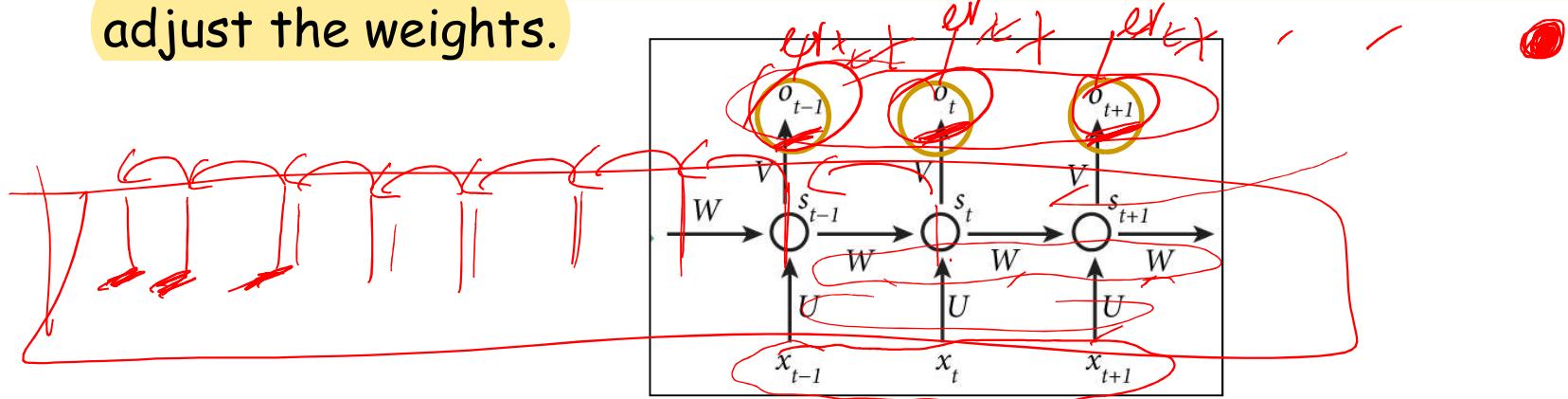
- the network outputs "e"
- but the target was "l"

We will need to backpropagate this error eventually...



RNN walk through: Feedforward

- And so on, until the entire input sequence is fed.
- Once the entire sequence is fed, we backpropagate the error, and adjust the weights.



- The RNN produced a sequence of n outputs O_i , we need to compare them to their targets
- Because RNNs have a feedback loop, standard backprop must be modified a bit --> backpropagation through time (BPTT)

RNN walk through: BPTT

- For LM, we are interested in the sequence of O_i
- so the error of the network is a function of all O_i

$$Error_t(T_t, O_t) = -T_t \times \log(O_t) // \text{cross entropy loss at time t}$$

$$Error(T, O) = -\sum_{t=1}^n (T_t \times \log(O_t)) // \text{error of the entire network}$$

// sum of the errors for all time steps

1. Calculate the gradient for each time step with respect to the weights
2. Combine gradients together for all time steps
3. Update weights

Problems with vanilla RNN

Standard backpropagation does not scale well with multiple layers...

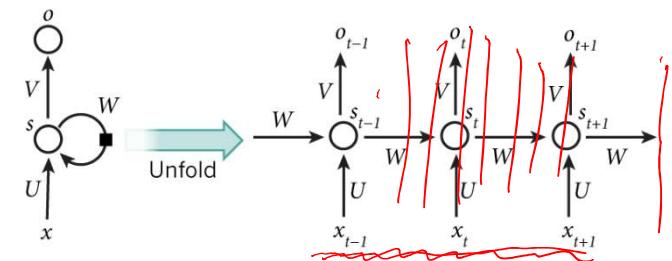
When we multiply the gradients many times
(for each layer), it can lead to ...

1. Vanishing gradient problem:

- gradients shrink exponentially with the number of layers
- So weights of early layers change very very slowly so network learns very very slowly
- Very annoying ...

2. Exploding gradient problem:

- multiplying gradients could also make them grow exponentially.
- this results in large updates to the weights (the weights can become so large as to overflow and result in NaN values)
- Can do "gradient clipping"



$$\delta_n = O_n (1 - O_n) \times (O_n - T_n)$$

$$\dots$$

$$\delta_5 = O_5 (1 - O_5) \times \sum(w \delta_6)$$

$$\dots$$

$$\delta_4 = O_4 (1 - O_4) \times \sum(w \delta_5)$$

$$\dots$$

$$\delta_1 = O_1 (1 - O_1) \times \sum(w \delta_2)$$

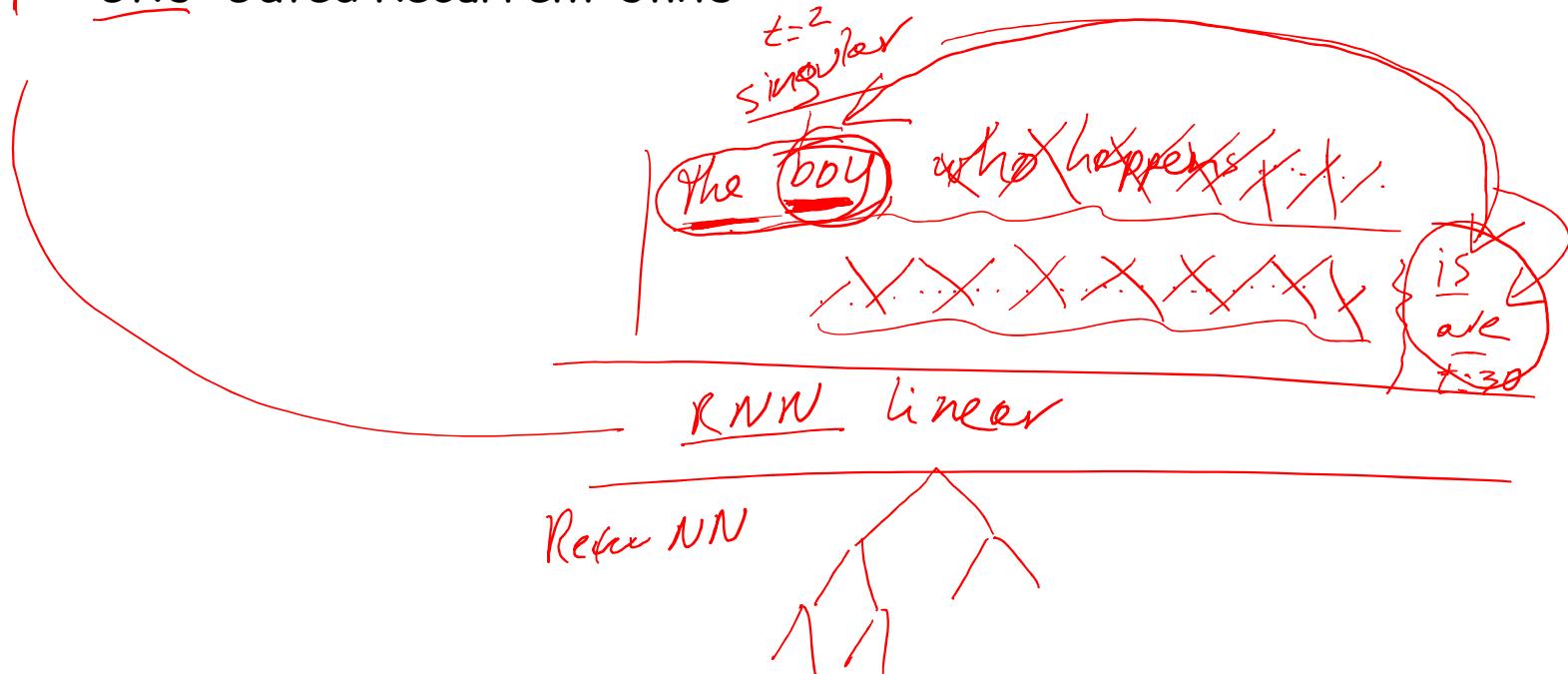
$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij} \quad \text{where } \Delta w_{ij} = -\eta \delta_j O_i$$

Problems with vanilla RNN

- One solution is to use a "memory" to deal with long distance dependencies (aka.

- gated |
- LSTM: Long Short-Term Memory
 - GRU: Gated Recurrent Units

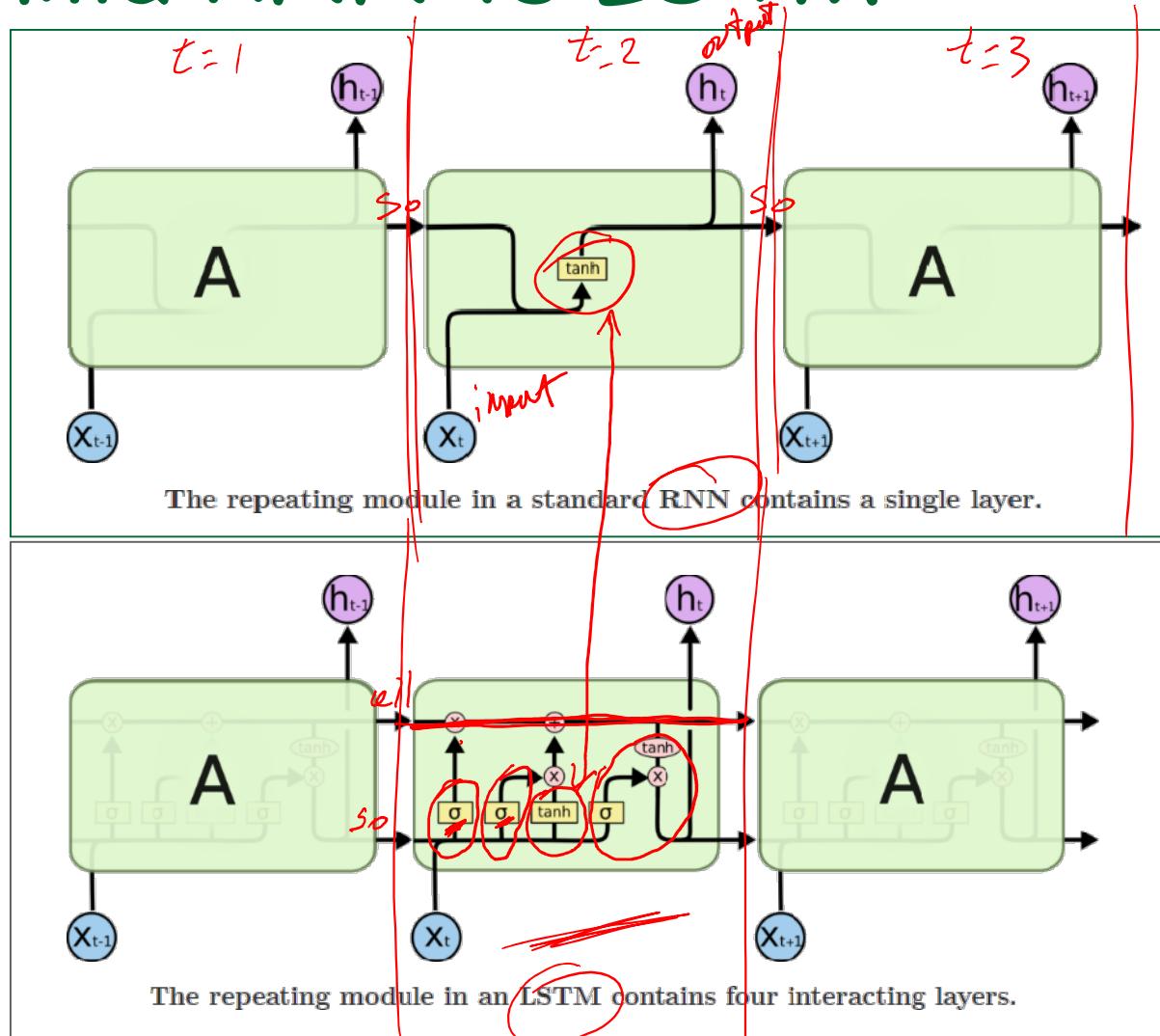
YOU ARE HERE!



LSTMs

- Explicitly designed to deal with long-term dependencies
- LSTMs contain information outside the normal flow of the recurrent network in a gated cell.
- Information can be:
 1. Stored in the cell
 2. Removed from the cell
 3. Read from the cell.
- The cell learns when to allow data to enter or be deleted through feedforward - backpropagation
- Works well in many problems and now widely used in NLP

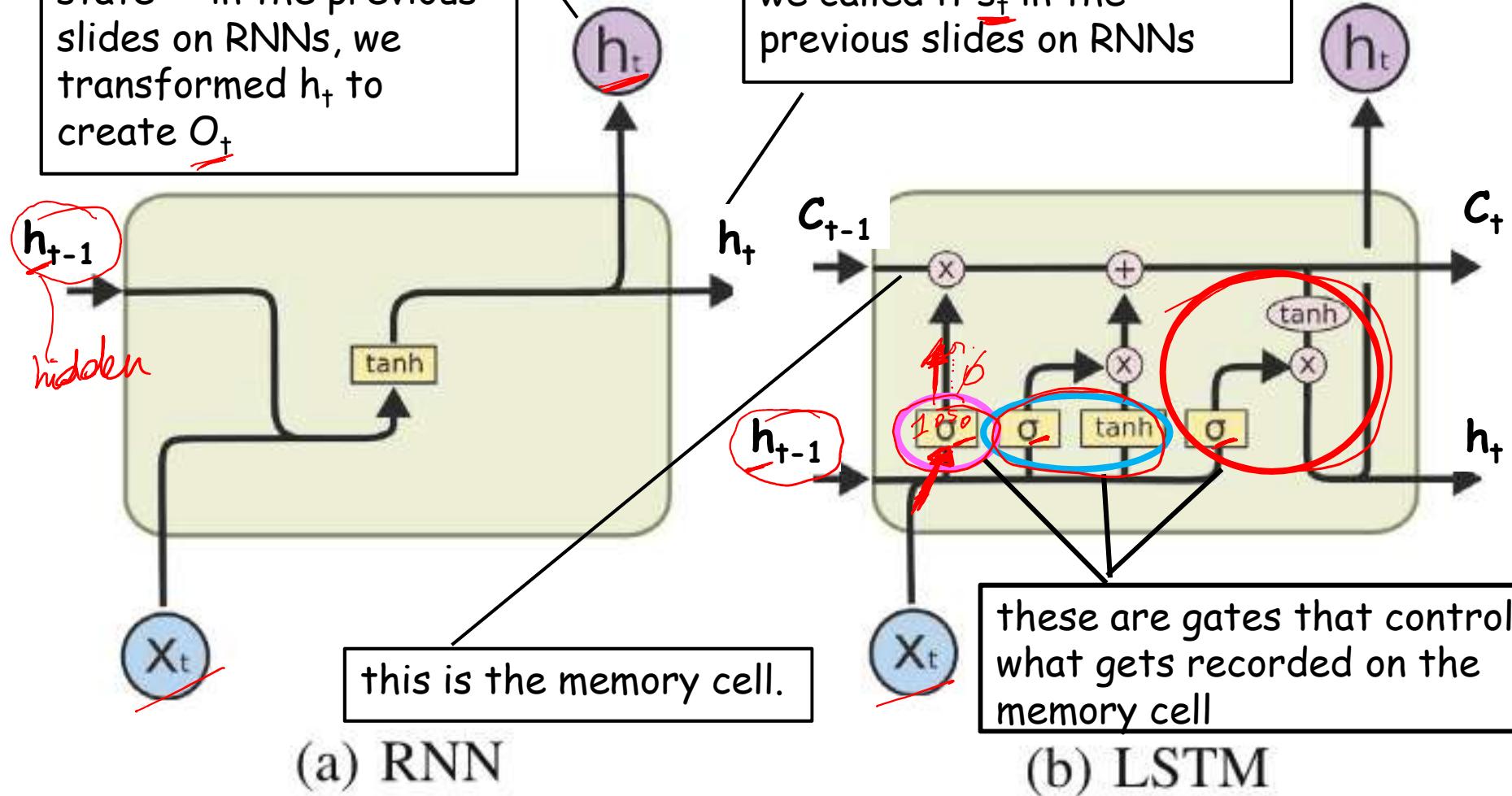
Vanilla RNN vs LSTM



Up Close

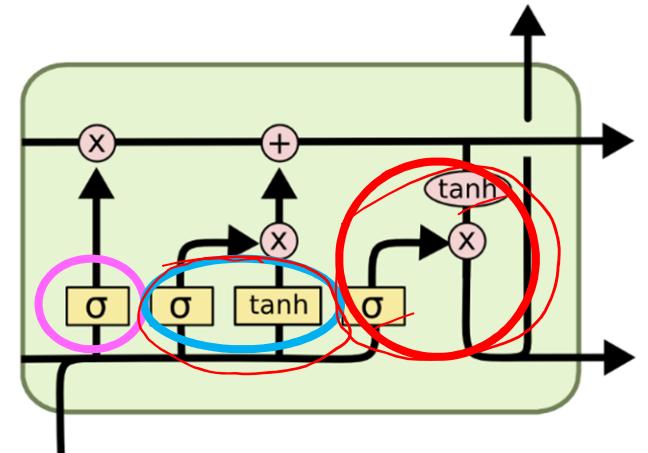
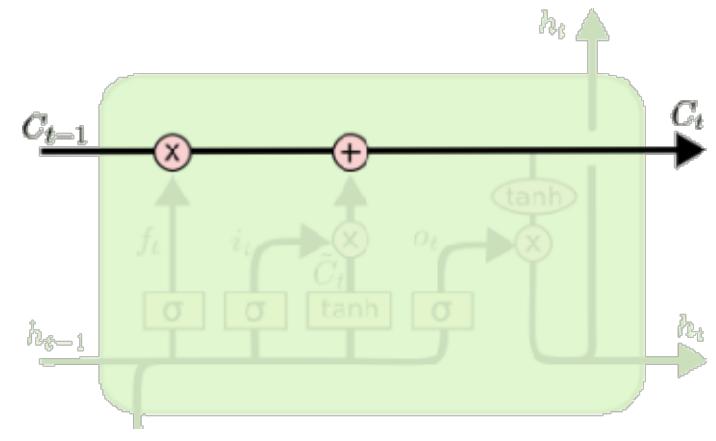
the output is the hidden state -- in the previous slides on RNNs, we transformed h_t to create O_t

this is the hidden state -- we called it s_t in the previous slides on RNNs



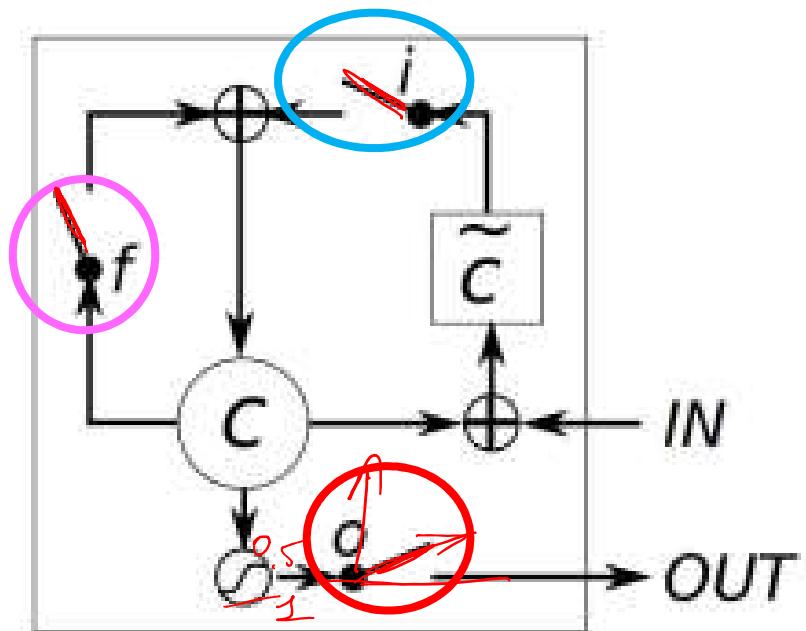
Intuition Behind LSTMs

- The key to LSTMs is the **cell state C_t**
 - The horizontal line running through the top of the diagram
- Information can be stored in the cell or removed from the cell through 3 gates:
 1. **Forget gate** - decides/learns what info to remove from the cell
 2. **Input gate** - decides/learns what new info to store in the cell
 3. **Output gate** - decides/learns what values in the cell are used to compute the output of the LSTM unit



Close-up of the gates

1. Forget gate - learns what info to remove from the cell
2. Input gate - learns what new info to store in the cell
3. Output gate - learns what values in the cell are used to compute the output of the LSTM unit



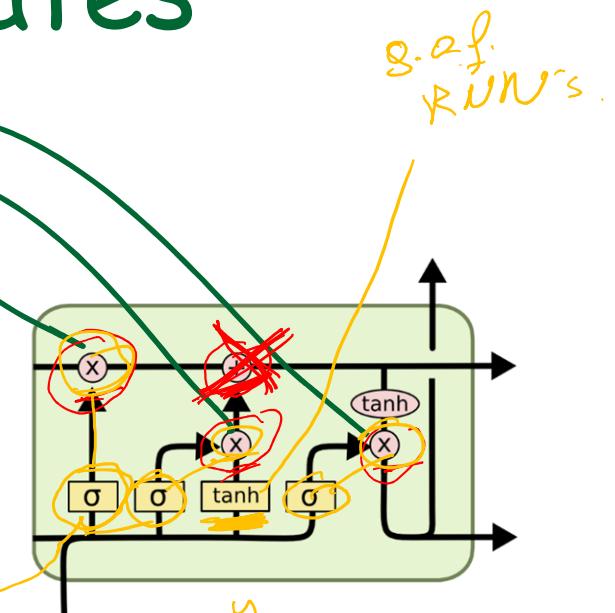
(a) Long Short-Term Memory

Intuition behind the gates

$$a \times b = \begin{bmatrix} 0 & \frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \left(0b_1 + \frac{1}{2}b_2 + 1b_3 \right)$$

(Handwritten annotations: 'gates' written above the first matrix, circled '0', circled '1/2', circled '1', circled 'b1', circled 'b2', circled 'b3', circled '0b1 + 1/2b2 + 1b3')

- tanh returns a value $[-1, 1]$ ☹
- sigmoid returns a value $[0, 1]$ ☺
- so network learns the values that allows sigmoid to return
 - 0 --> ignore completely //closed gate
 - 1 --> let through completely //open gate wide
 - $1/n$ --> let through a bit



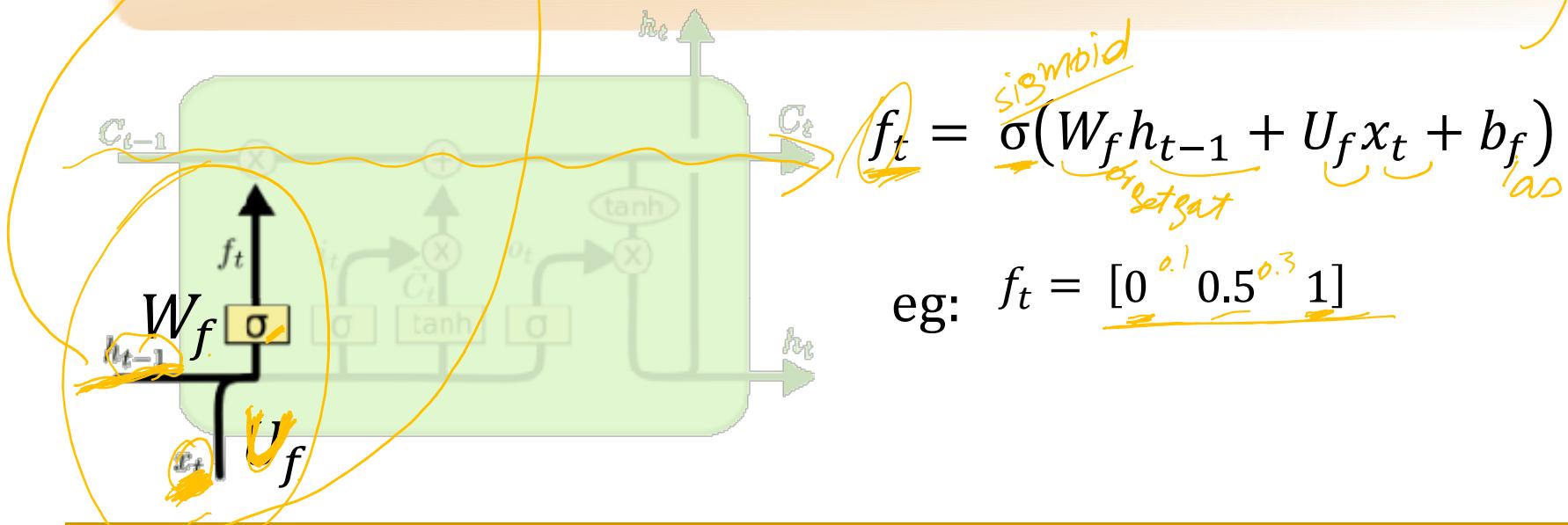
LSTM Walk Through: Forget Gate

Forget gate learns what information will be removed from the cell state

- Looks at h_{t-1} and x_t and outputs a value between 0 and 1
- 0 (completely ignore this) ... 1 (completely keep this)

My ~~sos~~ s are sick, but my daughter is not.

--> forget the number of the old subject, when we see a new subject.



LSTM Walk Through: Input Gate

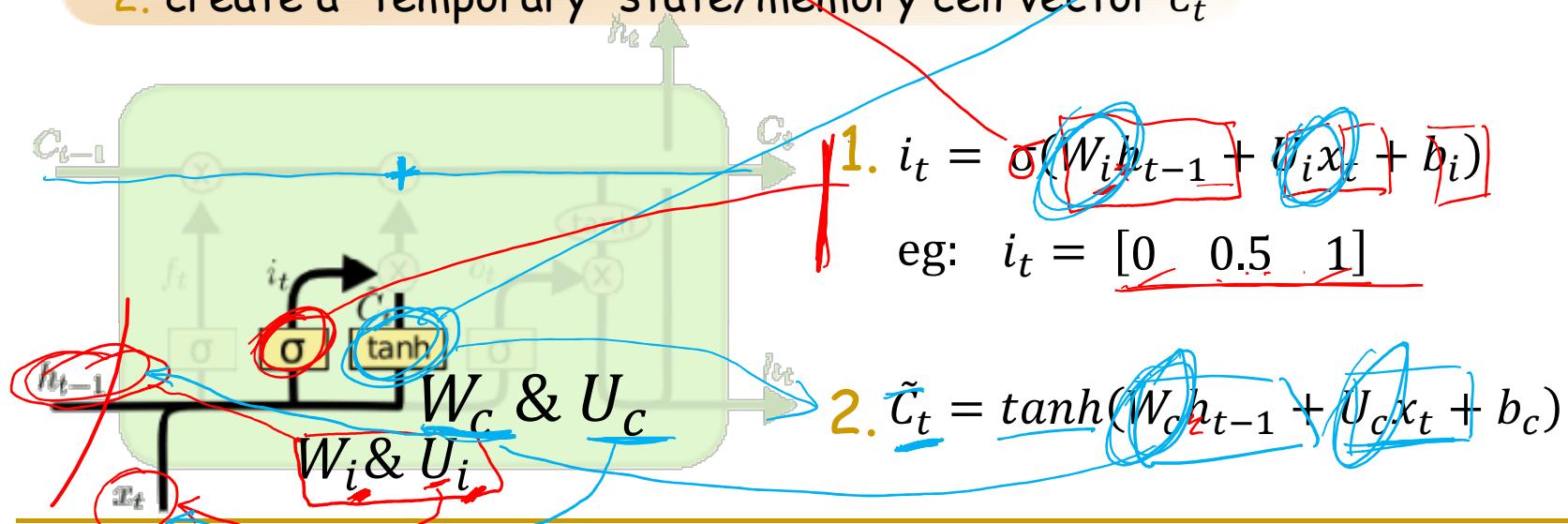
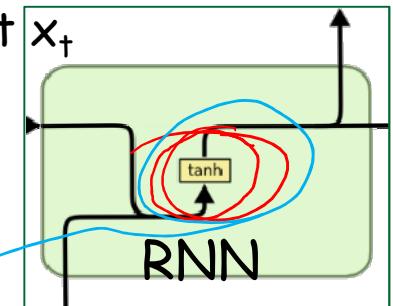
Input gate learns what new information will be stored in the cell

1. sigmoid layer decides what values we take from the input
2. tanh layer does what the RNN's tanh layer did

~~My sons are sick, but my daughter is not.~~

1. decide to keep the number of the new subject

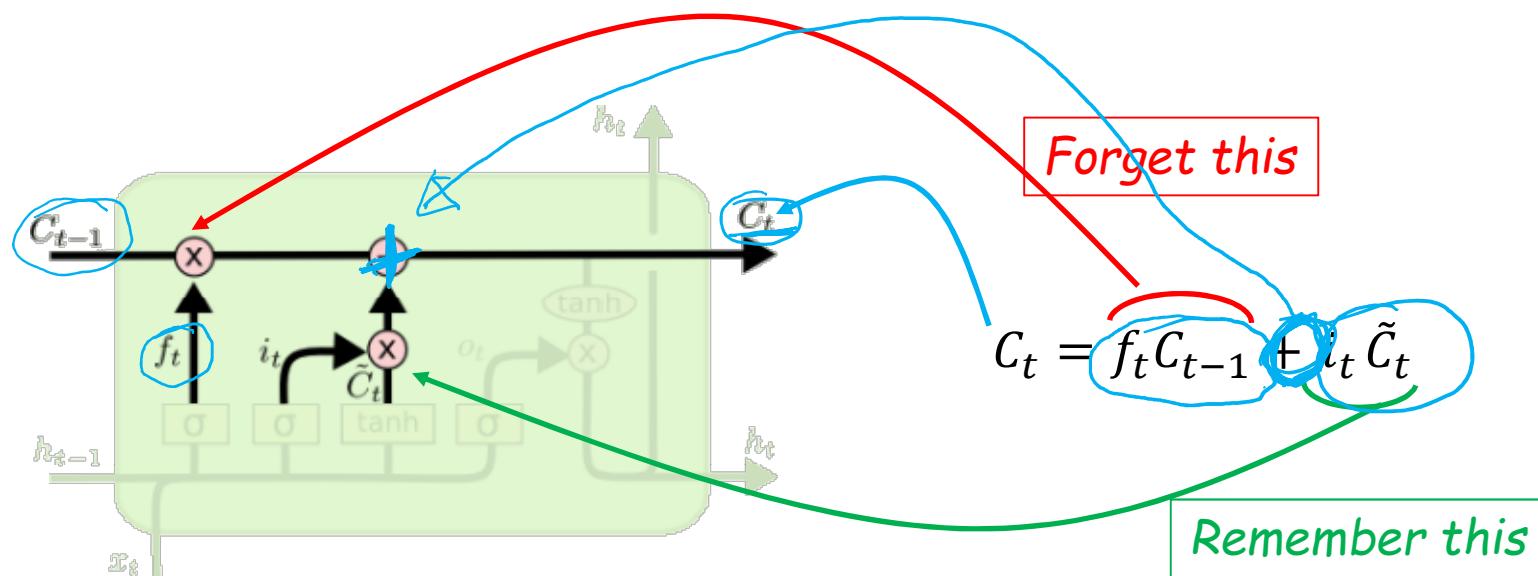
2. create a "temporary" state/memory cell vector \tilde{C}_t



LSTM Walk Through: Update the cell

Finally, we create the current cell state C_t as:

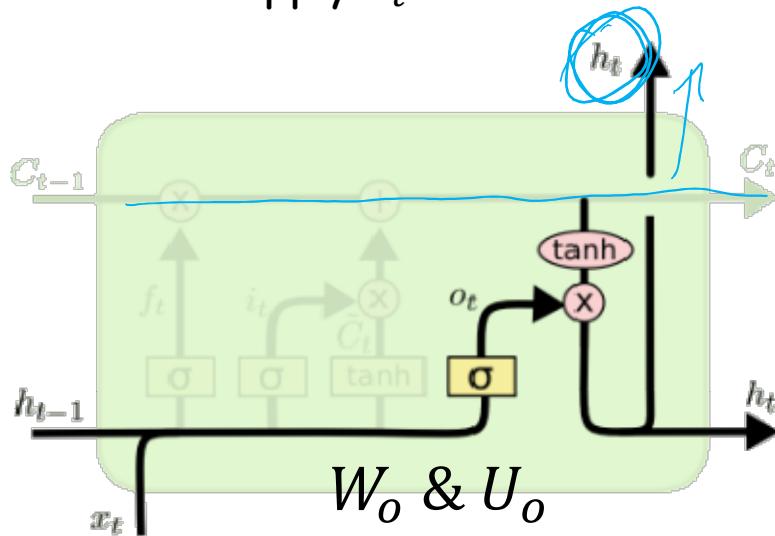
- The previous cell state $C_{t-1} \times f_t$ (the forget vector)
- Then we add (+) the new info to remember $i_t \times \tilde{C}_t$



LSTM Walk Through: Output Gate

Output gate controls how the value in the cell is used to compute the output of the LSTM unit.

1. Apply sigmoid to create o_t (i.e. decide what parts of the hidden state h_{t-1} and the input x_t we will output)
2. Apply o_t to the tanh of the cell C_t



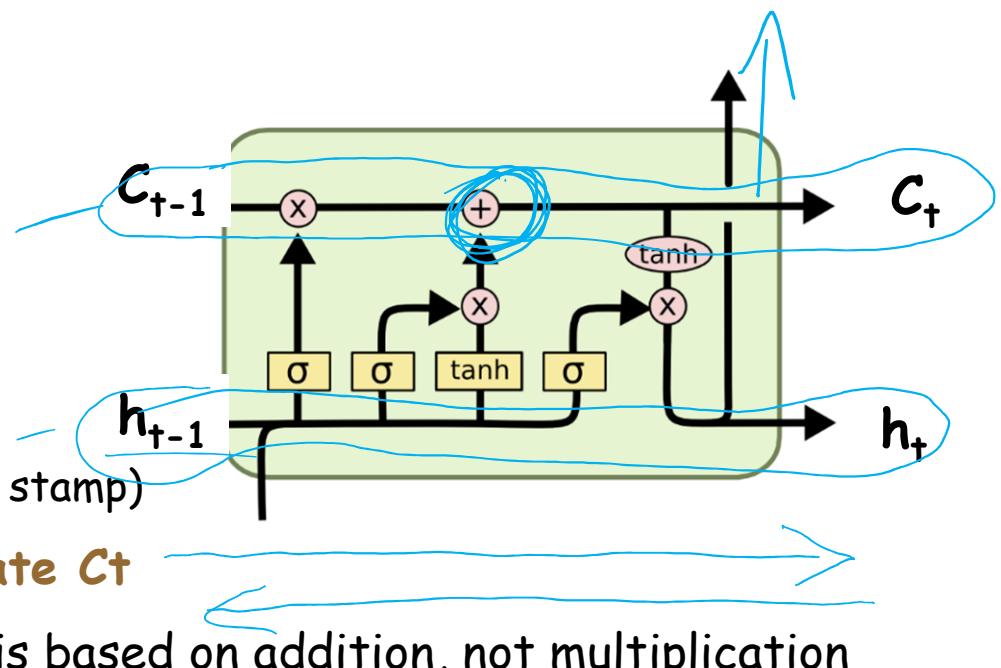
$$1. o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$$

eg: $o_t = [0 \quad 0.5 \quad 1]$

$$2. h_t = o_t \tanh(C_t)$$

LSTM - Recap

- The keys to LSTMs is that:
 - the model learns what to
 1. Forget (from the past)
 2. Input (from the present)
 3. Output (at the current time stamp)
 - stores this info in the **cell state C_t**
 - and updates to the cell state is based on addition, not multiplication

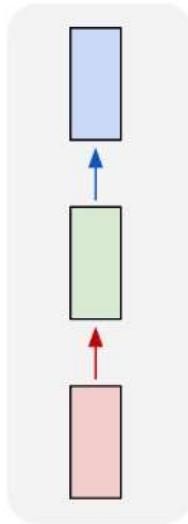


Conclusion

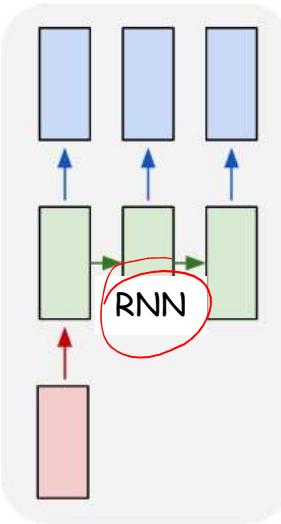
- LSTMs were a significant step in what we can accomplish with RNNs
- Other variants include:
 - Bidirectional LSTM
 - Gated Recurrent Units (GRU)
 - Sequence-to-sequence Models
 - Memory Networks
 - Attention Mechanisms
 - Transformers

Applications of RNNs

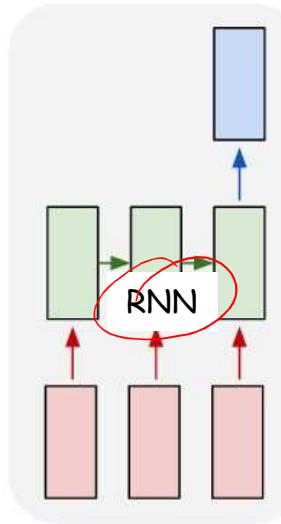
one to one



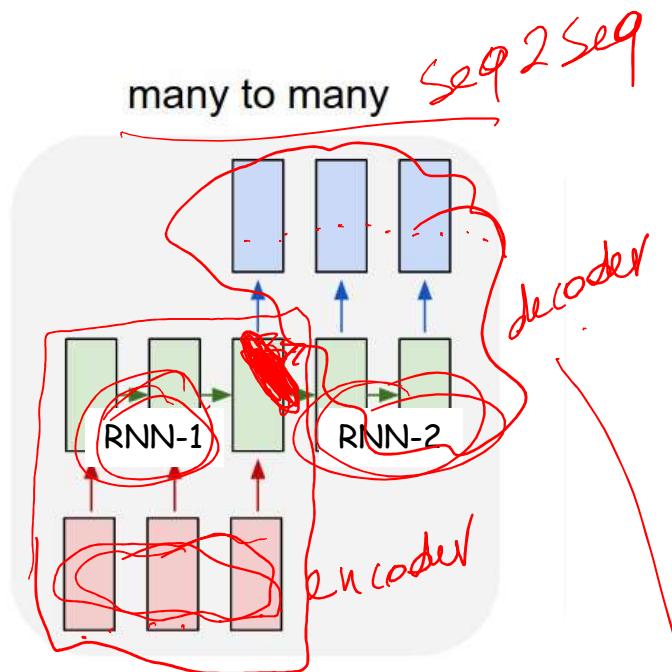
one to many



many to one



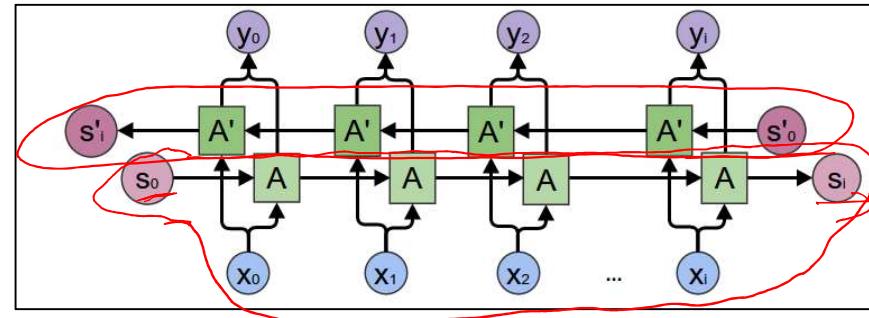
many to many



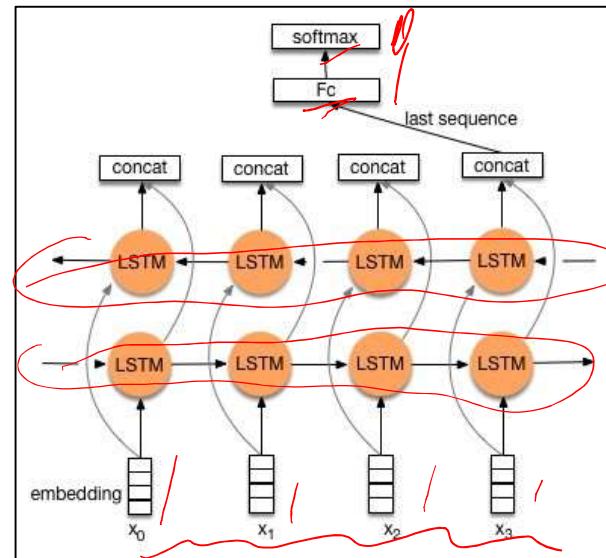
- (1) Vanilla ANN -- input: 1 unit, output: 1 unit (e.g. image classification)
- (2) RNN - input: 1 unit, output: sequence (e.g. image captioning)
- (3) RNN - input: sequence, output: 1 unit (e.g. sentiment analysis)
- (4) RNN - input: sequence, output: sequence (e.g. Machine Translation)

Bidirectional LSTM

Ex: A language model that uses both the past and future words

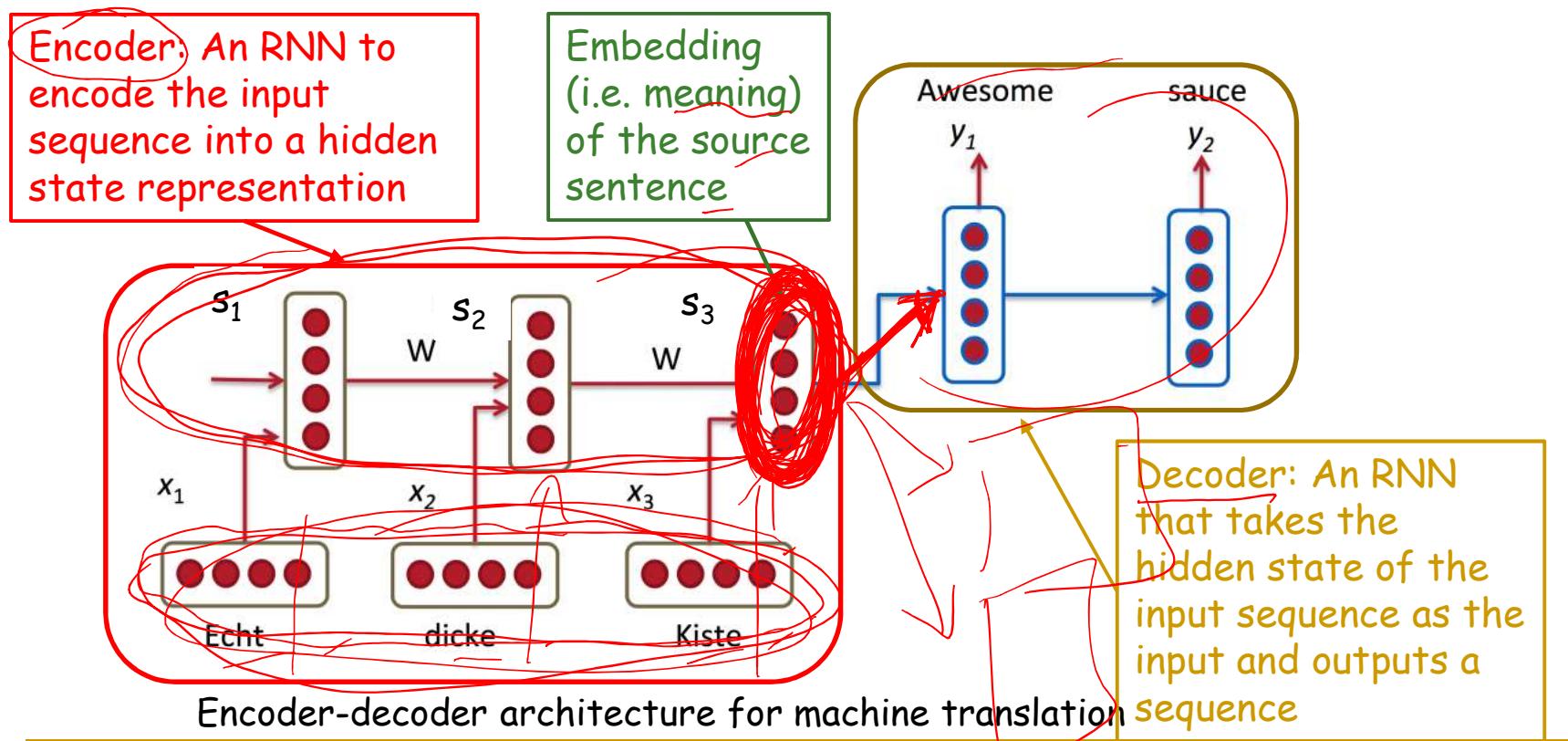


Ex: A classifier that feeds the sentence in both directions

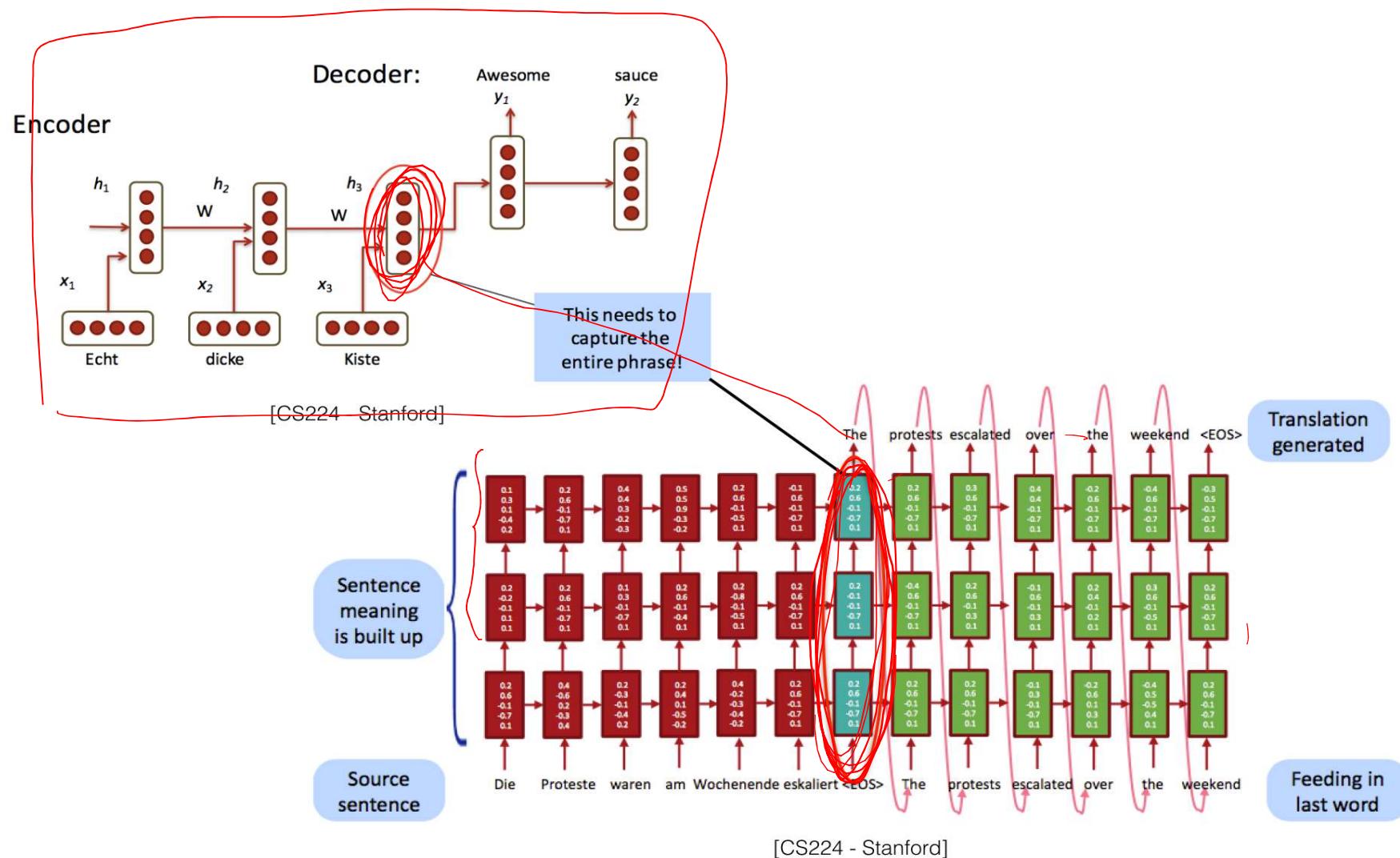


Seq2seq for Machine Translation

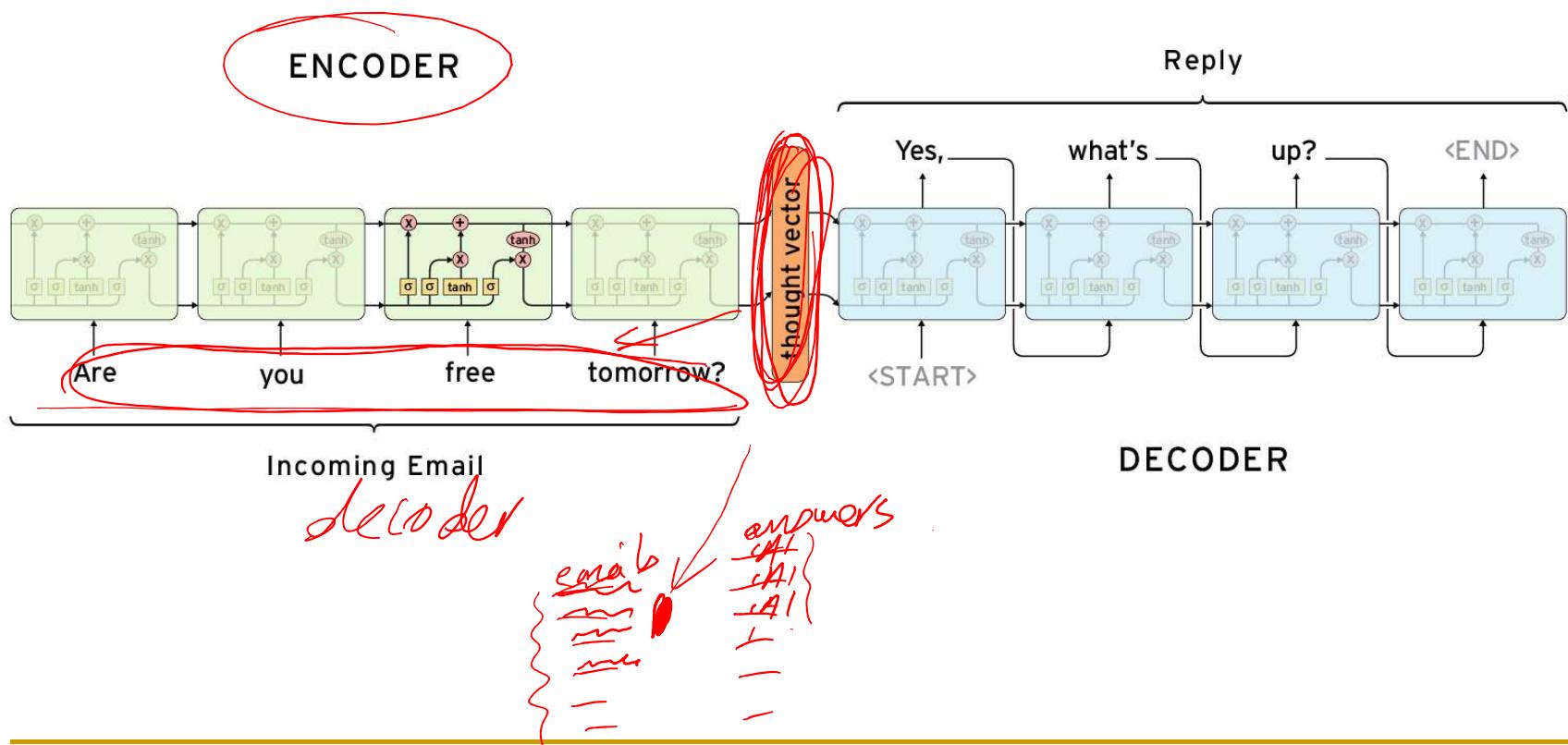
- The input is a sequence of length n
- The output is a sequence of length m



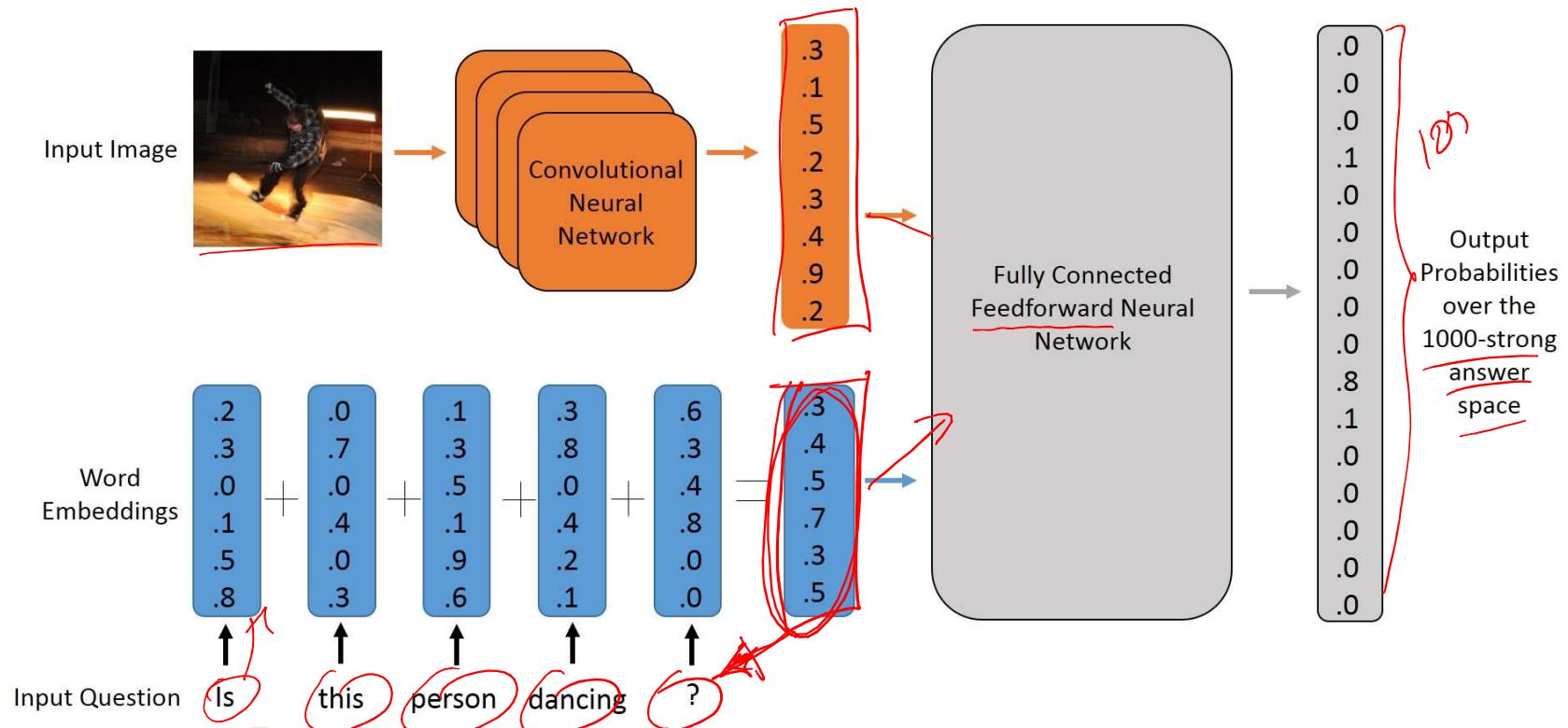
Seq2seq for Machine Translation



Seq2seq for Conversational Agents



Visual Question Answering



- The output is conditioned on both image and textual inputs.
 - A CNN is used to encode the image.
 - A RNN is used to encode the sentence.

Cool Applications

■ RNNs + word embeddings

- Google Translate
 - As of July 2017, uses an RNN + word embeddings (called Neural Machine Translation (NMT))
 - Input sequence: words in a source sentence
 - Output sequence: words in the target language
- Dialogue Systems

■ CNNs + RNNs + word embeddings

- Image Captioning
- Video to Text Descriptions
- Visual Question Answering

- ...

Today

1. Introduction ✓
2. Bag of word model ✓
3. n-gram models ✓
4. Deep Learning for NLP
 1. Word Embeddings ✓
 2. Recurrent Neural Networks ✓

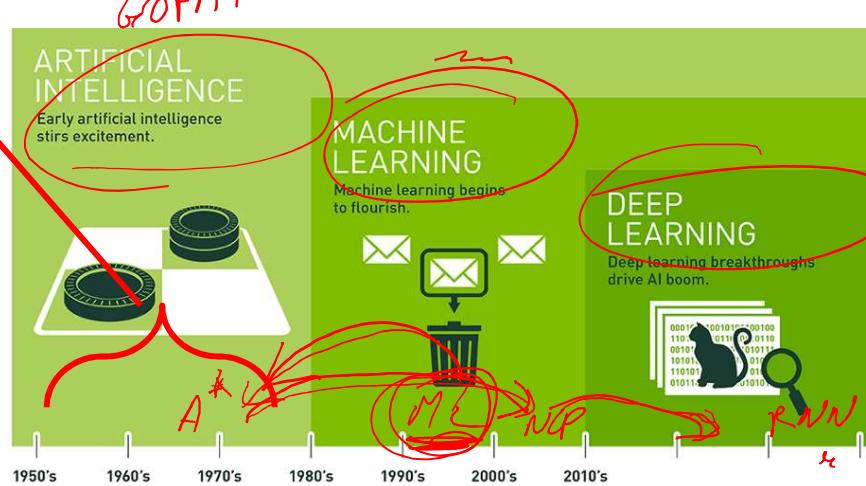
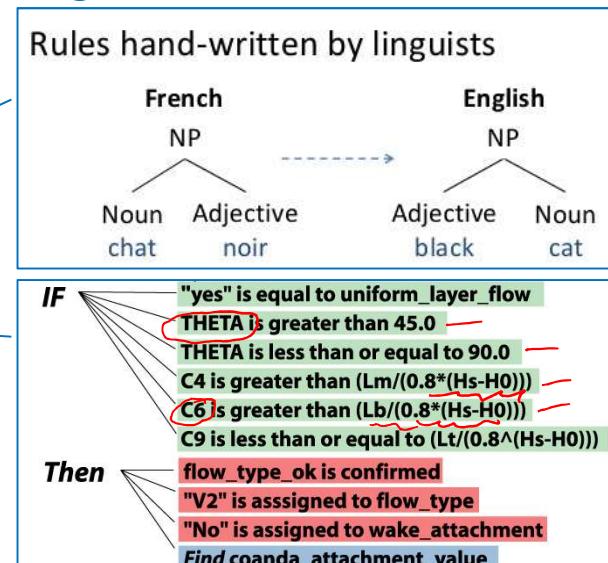
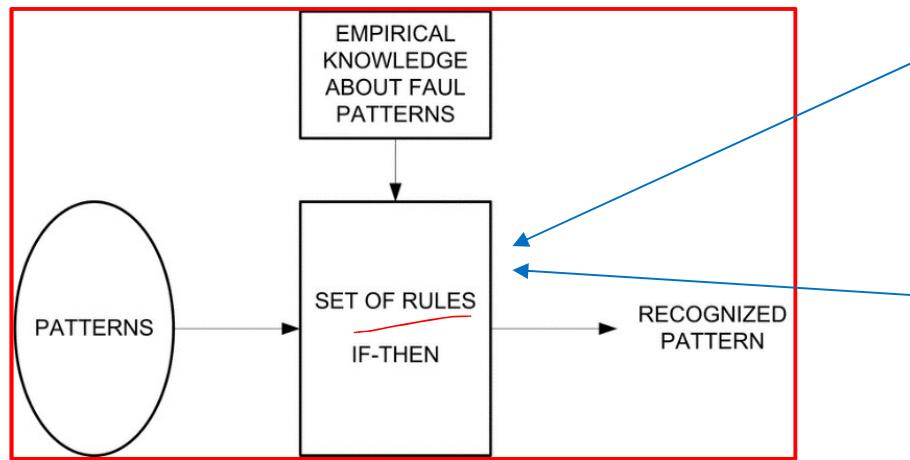


you made it!

here are some final words...

History of AI

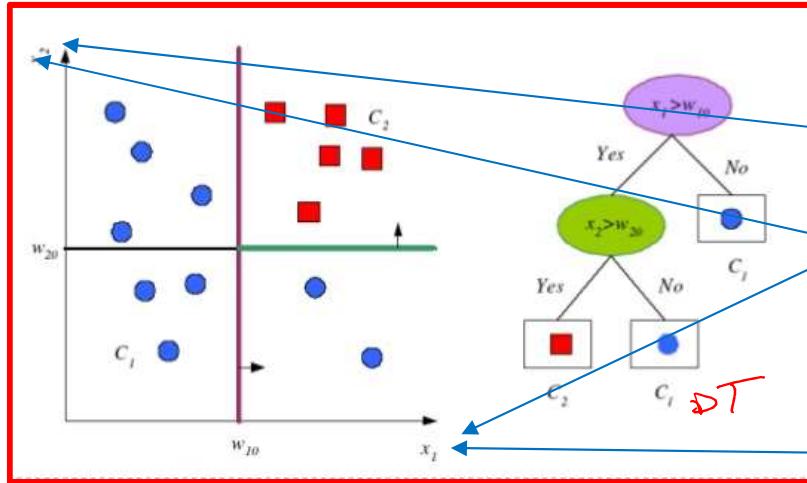
Rules written by experts (eg. linguistics, medical doctors,...) 😞



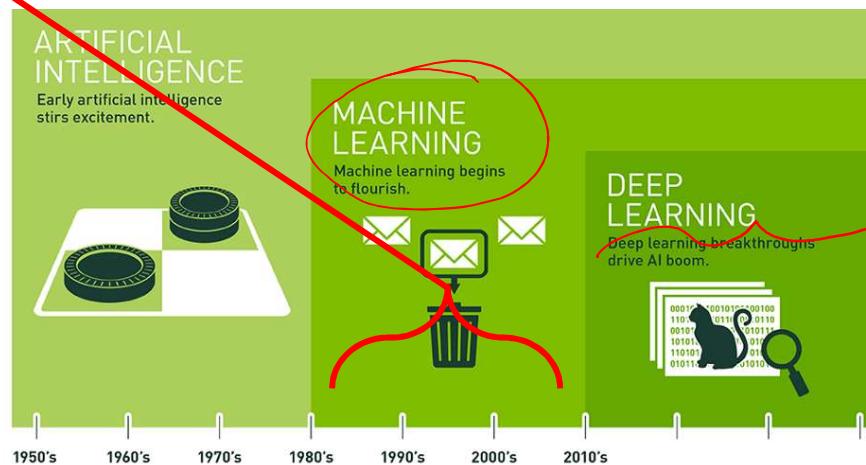
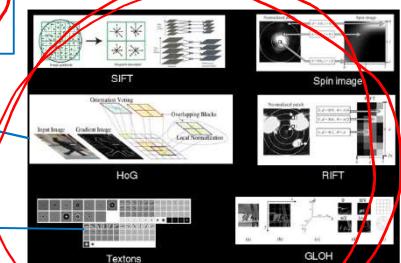
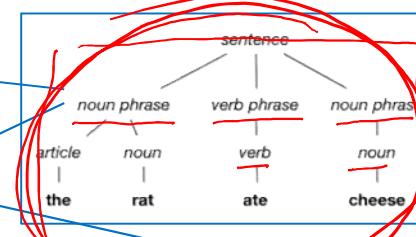
https://www.researchgate.net/profile/Dubravko_Miljkovic/publication/268239364/figure/fig30/AS:394719407427587@147111

<http://www.cormix.info/images/RuleTreeExample.jpg>

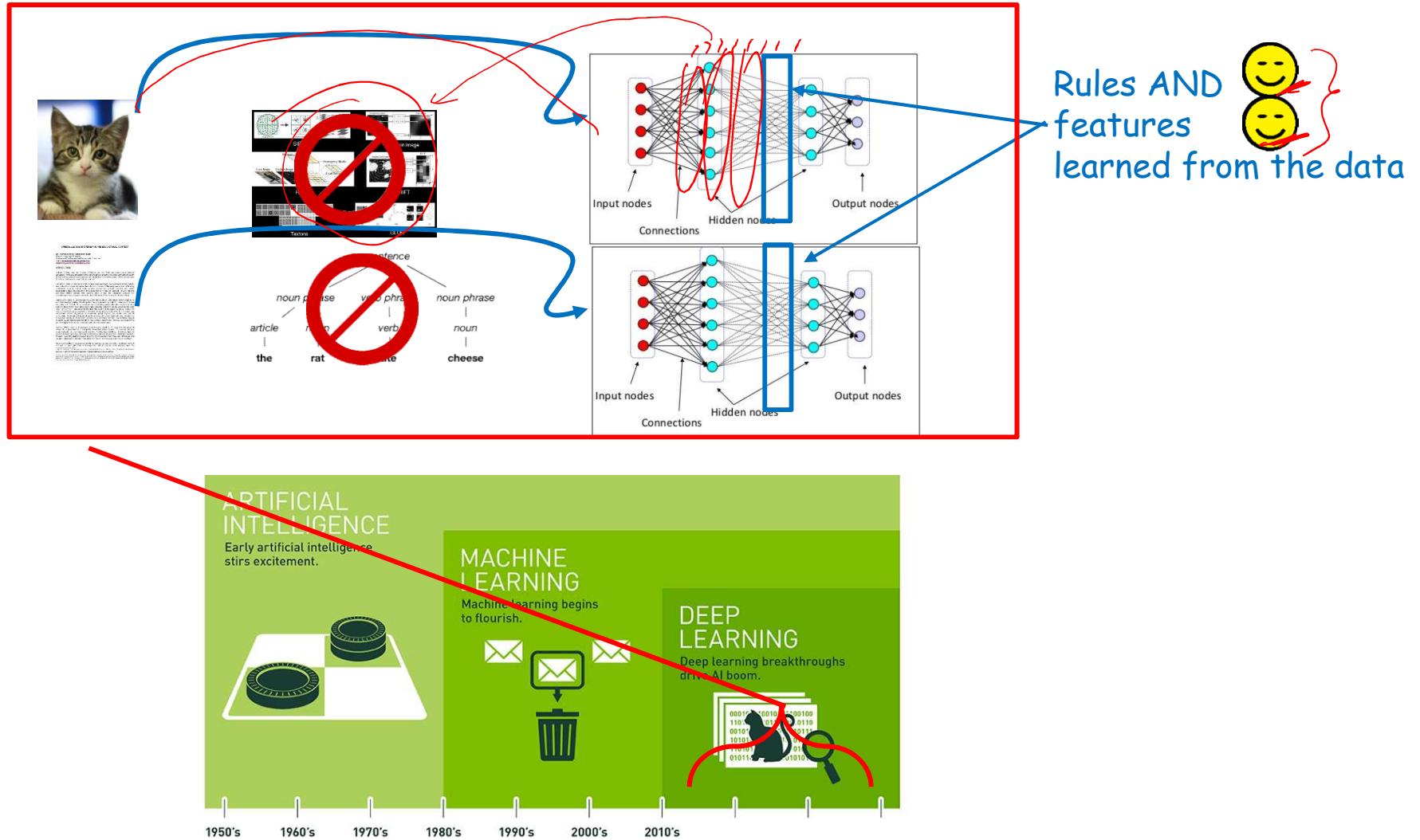
History of AI



Rules learns via the data ;)
But: features identified by the experts
(eg. linguistics, medical doctors,...)



History of AI



<https://www.linkedin.com/pulse/goedels-incompleteness-theorem-emergence-ai-eberhard-schoeneburg/>

Your country needs you!

■ Deep Learning is thriving !

- *vision*
- *image processing*
- *speech recognition*
- *natural language processing*
- ...



■ Canada is a world leader in Deep Learning

1. Montreal: (Bengio et al.) MILA
2. Toronto: (Hinton et al.) Vector Institute
3. Edmonton: (Sutton et al.) AMII



<https://www.canadaneedsyou.ca/canada-needs-you>

