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

- Russell & Norvig: Section 21.6, 24.2

Today

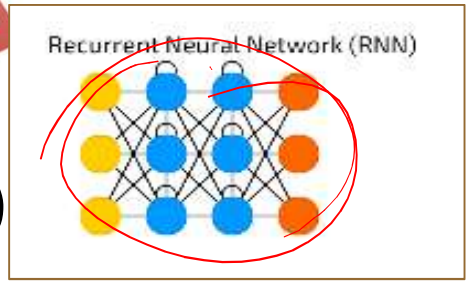
1. Introduction
2. Bag of word model
3. n-gram models
4. Deep Learning for NLP
    1. Word Embeddings
    2. Recurrent Neural Networks

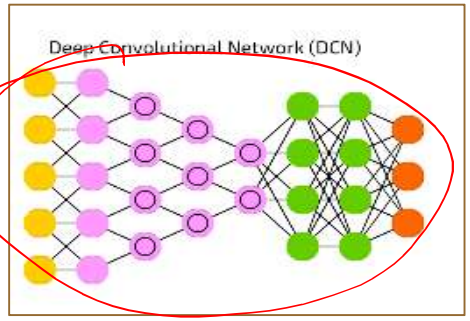
Deep Learning for NLP

Deep learning models for NLP use:

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

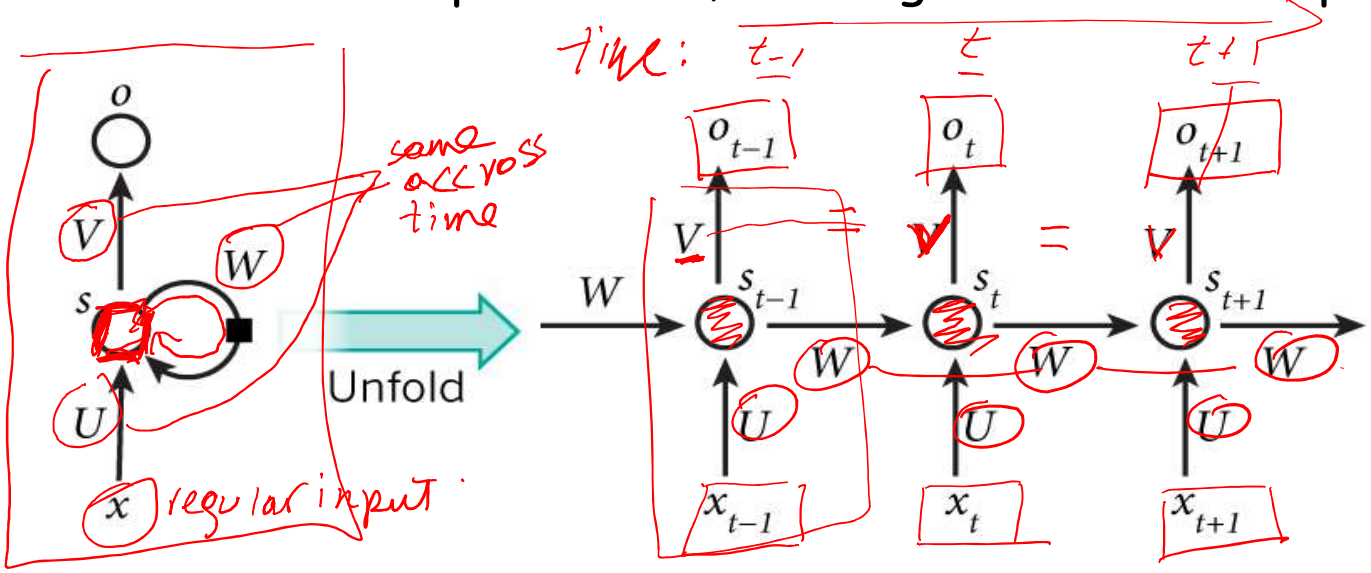
- Neural network structures
  - Recurrent Neural Networks (RNNs)
  - Recursive Neural Networks
  - Convolutional Networks (CNNs)
  - …





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.

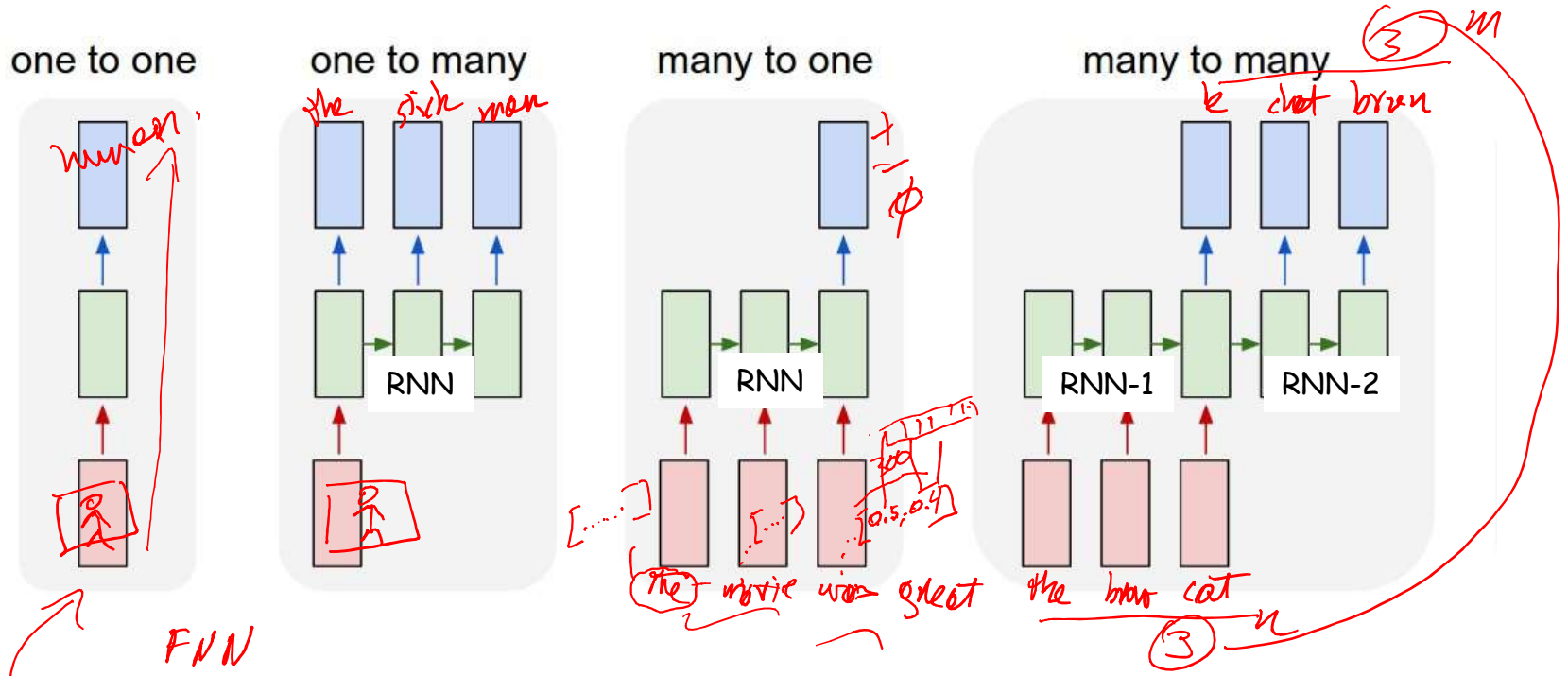


http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

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



(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)

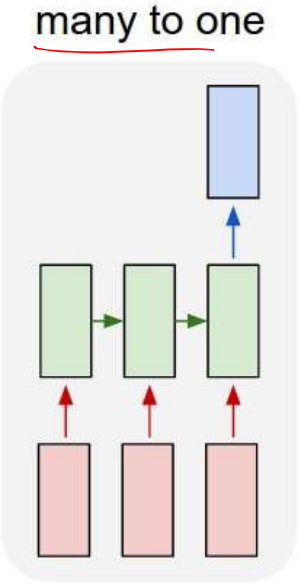
(4) RNN – input: sequence, output: sequence (e.g. Machine Translation)

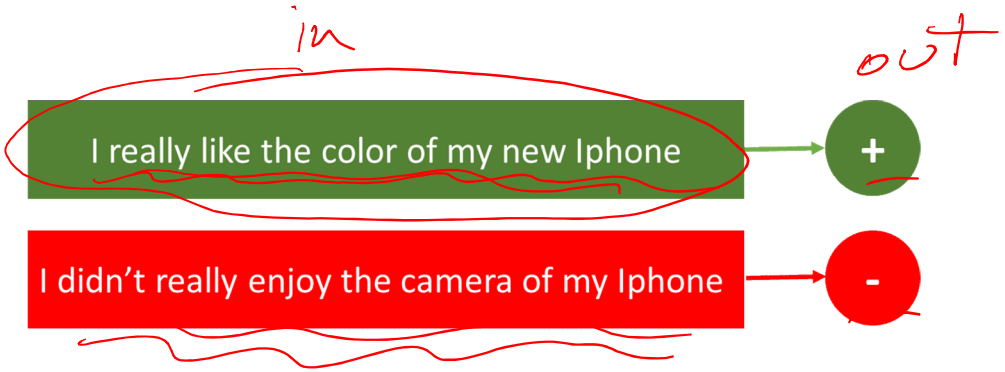
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Applications of RNNs

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

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





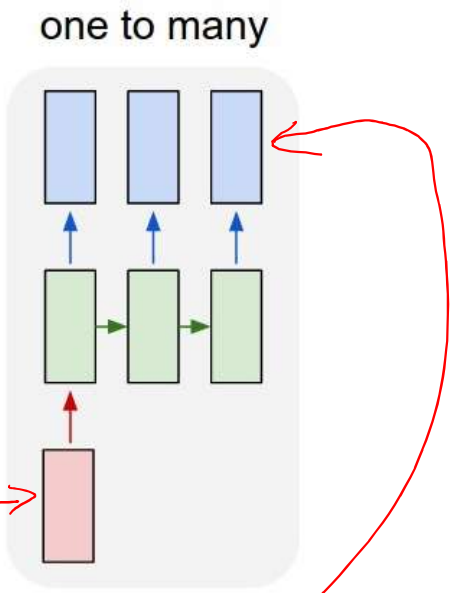
https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

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.

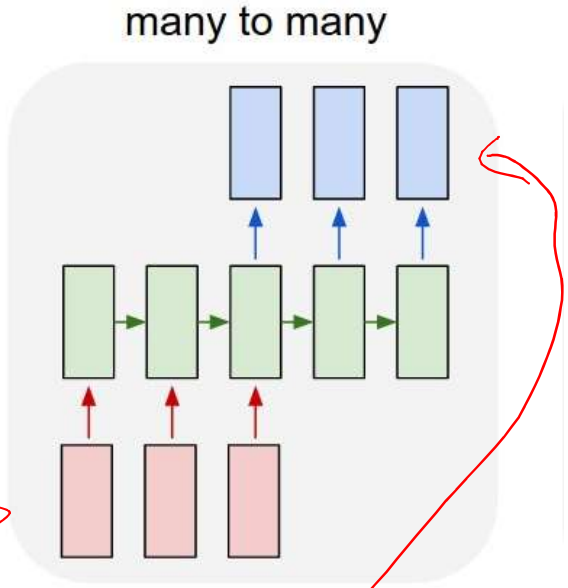


https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

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



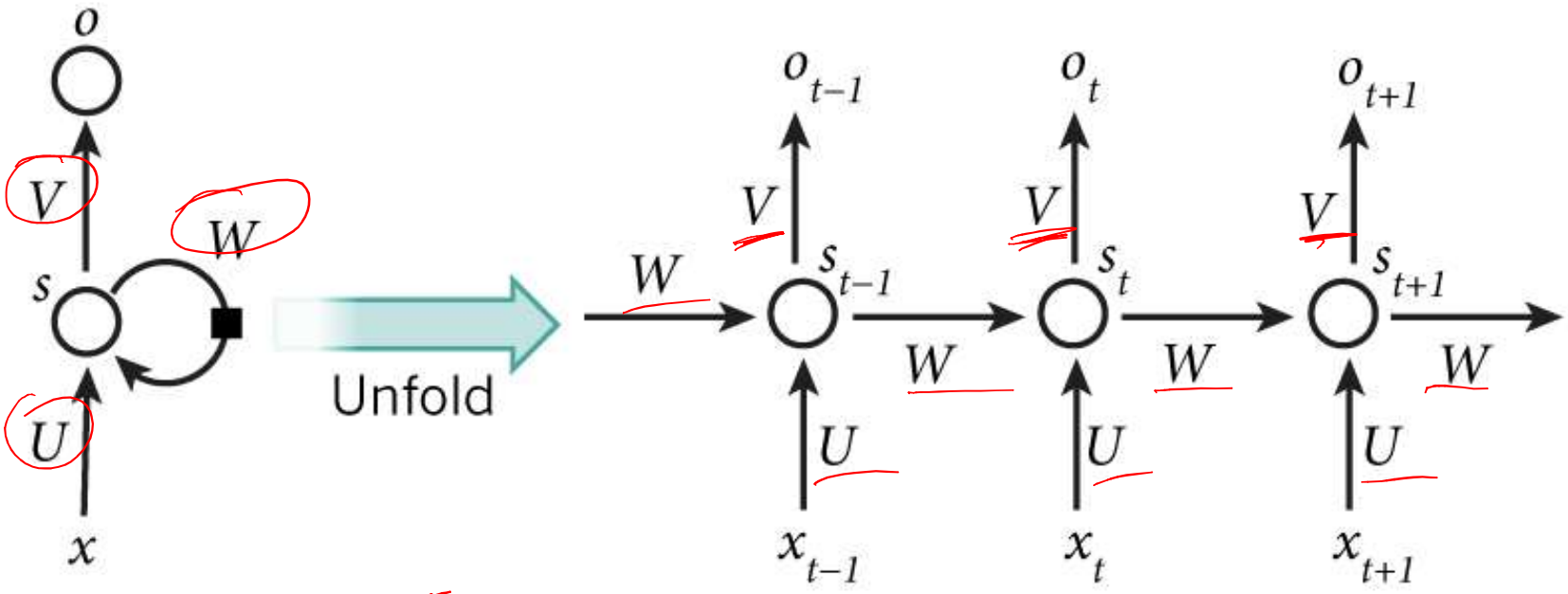
French was the official language of the colony of French Indochina, comprising modern-day Vietnam, Laos, and Cambodia. It continues to be an administrative language in Laos and Cambodia, although its influence has waned in recent years.

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.

https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Vanilla RNN



weight sharing

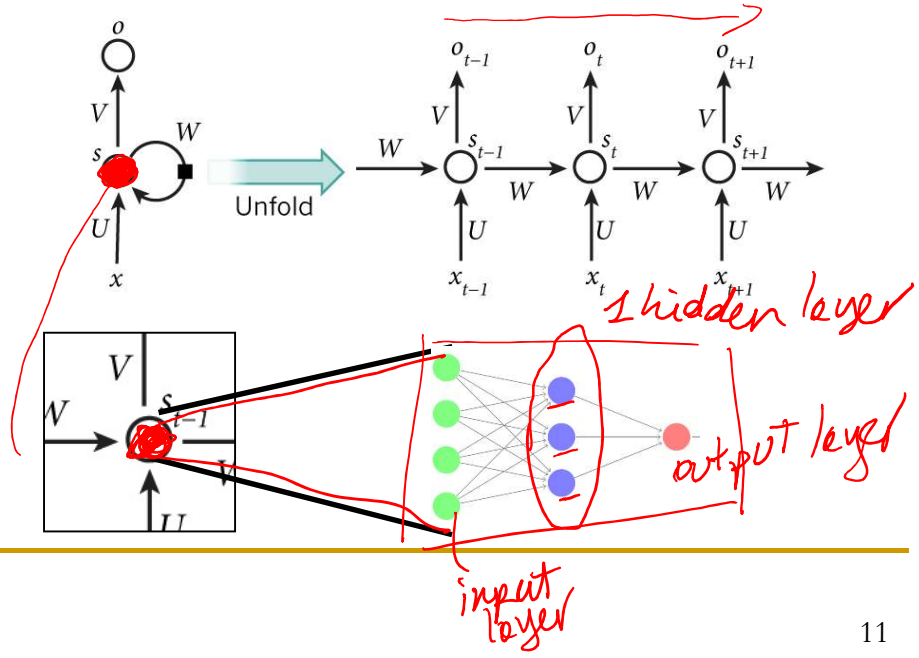
http://www.omegaxy.com/Pictures/NatureDL/rnn.png

RNN Walk through Example

- Task : Character-based Language Modeling

hello

  - 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}
    - nb of hidden layers = 1
    - nb of hidden nodes = 3

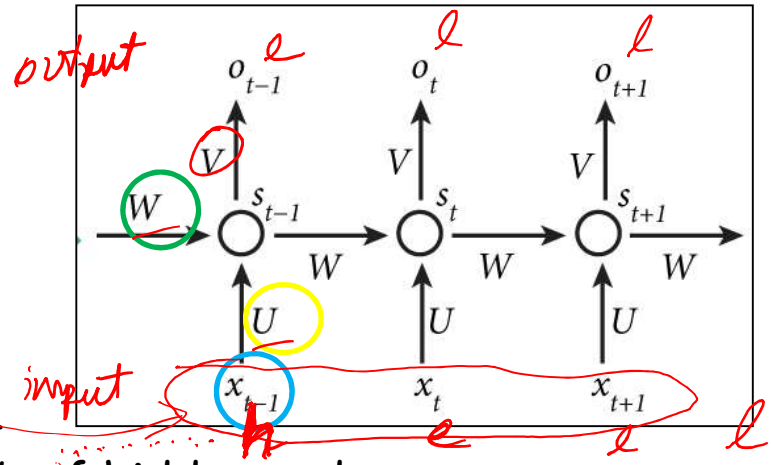


RNN Walk through: Set-up

- Assume:
  - We feed the network with 1-hot vectors

h	e	l	l
1	0	0	0
0	1	0	0
0	0	1	1
0	0	0	0

x_1 x_2 x_3 x_4



  - Matrix U is initialised to:

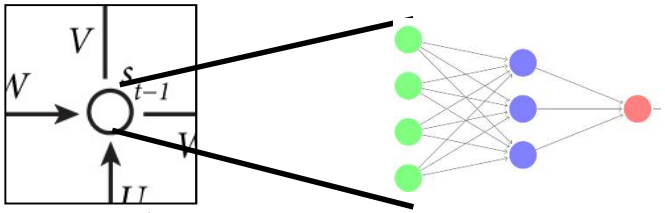
0.3738	0.7460	0.9057	0.8303
0.2885	0.9384	0.7007	0.0873
0.7306	0.0016	0.5606	0.6094

3 x 4
3 = nb of hidden nodes
4 = |V|

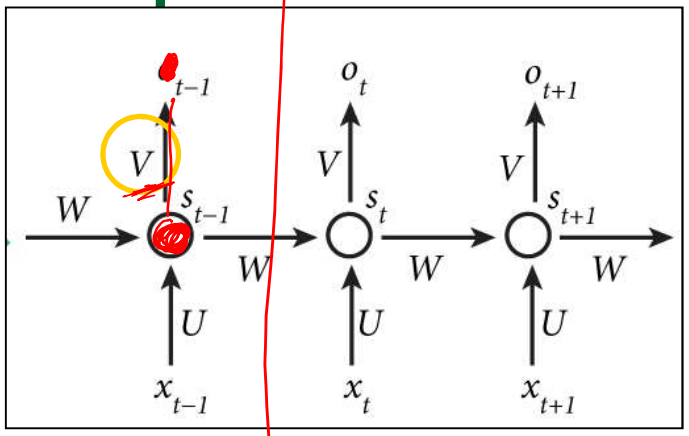
  - Matrix W is initialised to:

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

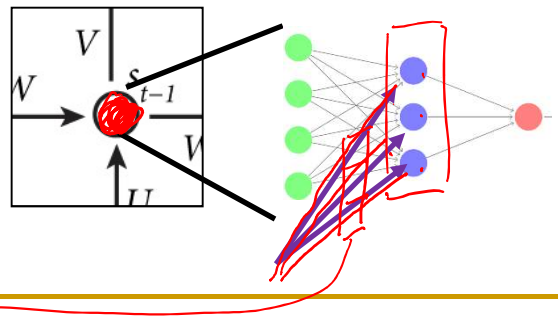
0.7170	0.0616	0.7425
0.5761	0.5914	0.7324
0.4216	0.3439	0.5849
0.9257	0.8114	0.0483

4 x 3
4 = |V|
3 = nb of hidden nodes

  - Bias weights into hidden nodes are initialised to:

random

0.6917
0.1565
0.1411



  - Bias weights into output nodes are initialised to:

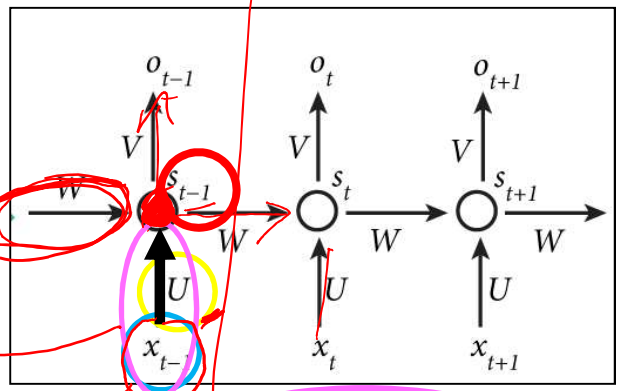
random

0.4567
0.7654
0.3456
0.0013

RNN walk through: Feedforward

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

s_t-part-a = U*x_t



part-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

✖

x_1
1
0
0
0

=

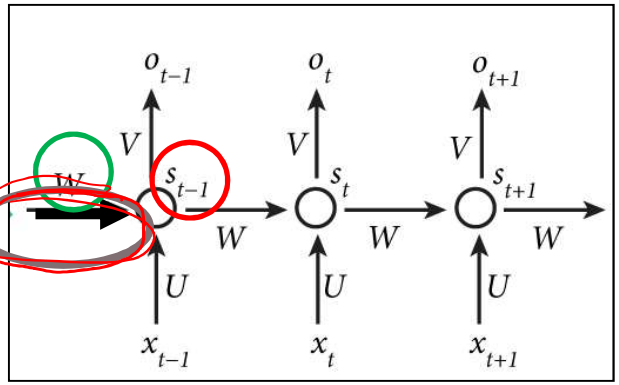
S_1-part-a

0.3738
0.2885
0.7306

RNN walk through: Feedforward

Step 2: compute s_1-part-b

s_t -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.

S_1-part-b

W		
0.2464	0.1306	0.4511
0.7445	0.7055	0.6592
0.5034	0.4701	0.9174

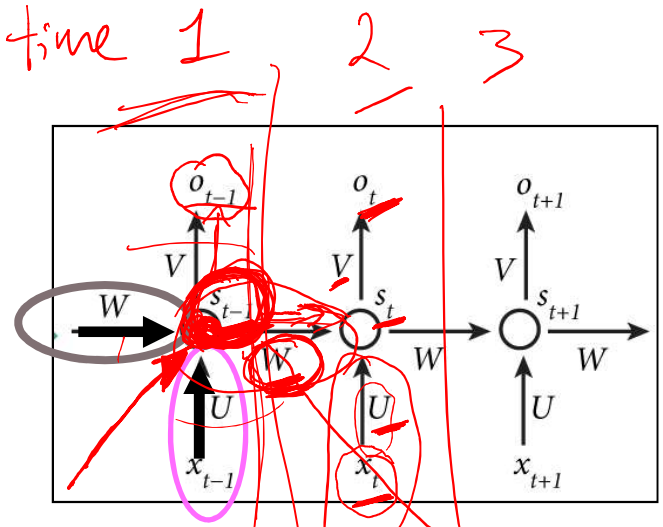
$\times$ S_0 = 0, 0, 0 $=$ 0, 0, 0

RNN walk through: Feedforward

Step 3: compute s_1

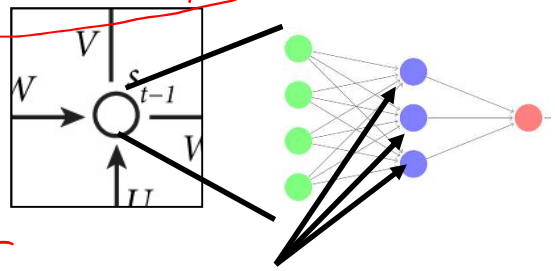
- Combine the 2 parts to get the value of hidden state s_1

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



s_1-part-a	s_1-part-b	bias		
0.3738	0	0.6917	=	1.0654
0.2885	0	0.1565	=	0.4450
0.7306	0	0.1411	=	0.8717

$$s_1 = \tanh\begin{pmatrix}1.0654\\0.4450\\0.8717\end{pmatrix} = \begin{pmatrix}0.7877\\0.4178\\0.7022\end{pmatrix}$$

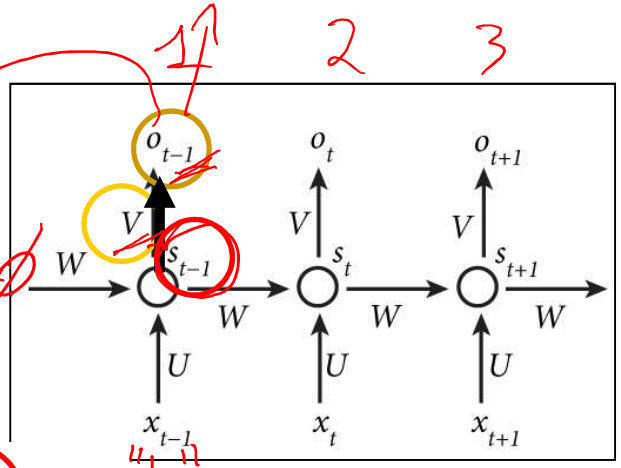


RNN walk through: Feedforward

Step 4: compute O_1

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

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



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

s_1

0.7877
0.4178
0.7022

0.4567
0.7654
0.3456
0.0013

$$\text{softmax}(\quad \times \quad + \quad)$$

$$= \text{softmax}(\quad) = \quad O_1$$

1.5687
1.9806
1.2321
1.2034

0.255241
0.385333
0.182292
0.177134

RNN walk through: Feedforward

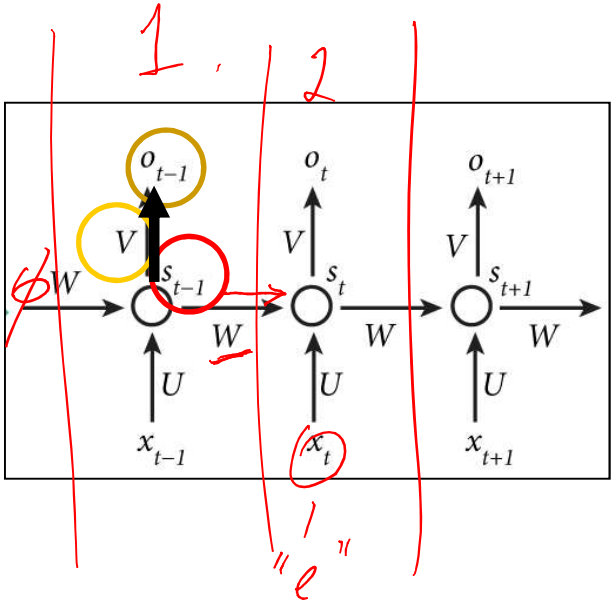
Output at time t=1 is:

O_1

0.255241
0.385333
0.182292
0.177134

Recall that V was:

h	e	l	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"

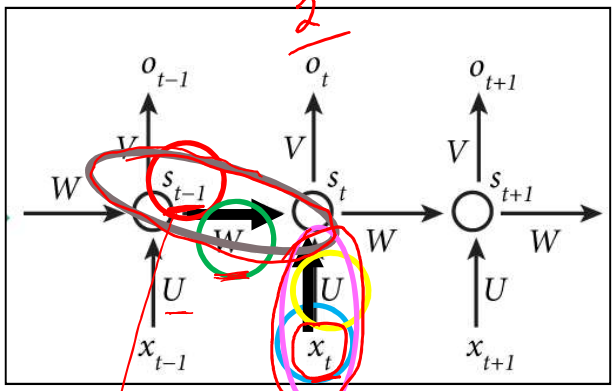
Error = 0

RNN walk through: Feedforward

0.6917
0.1565
0.1411

and we continue feedforwarding:

Let's go on to the next state with x_2 = "e"



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

s_2-part-a

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

x_2

0
1
0
0

=

0.7460
0.9384
0.0016

s_2-part-b

W

0.2464	0.1306	0.4511
0.7445	0.7055	0.6592
0.5034	0.4701	0.9174

×

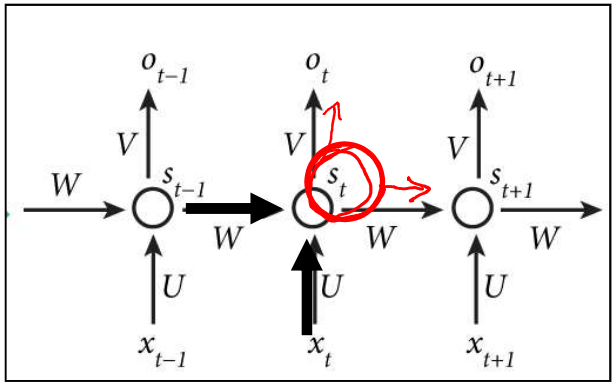
0.7877
0.4178
0.7022

=

0.5654
1.3441
1.2371

RNN walk through: Feedforward

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

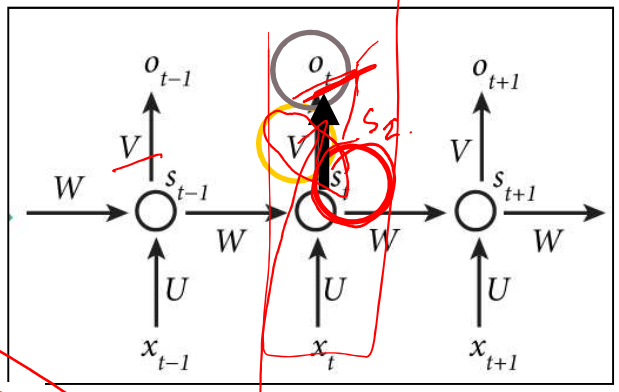


$S_2 = \tanh($

s_1-part-b	+	s_1-part-a	+	bias	) =
0.7460		0.5654		0.6917	0.9642
0.9384		1.3441		0.1565	0.9849
0.0016		1.2371		0.1411	0.8809

RNN walk through: Feedforward

O_2 = softmax(V * s_2 + bias)



O_2 = softmax(V | s_2 | bias)

V			s_2	bias
0.7170	0.0616	0.7425	0.9642	0.4567
0.5761	0.5914	0.7324	0.9849	0.7654
0.4216	0.3439	0.5849	0.8809	0.3456
0.9257	0.8114	0.0483		0.0013

softmax(1.8628, 2.5485, 1.6061, 1.7356) = 0.2155, 0.4279, 0.1667, 0.1898

RNN walk through: Feedforward

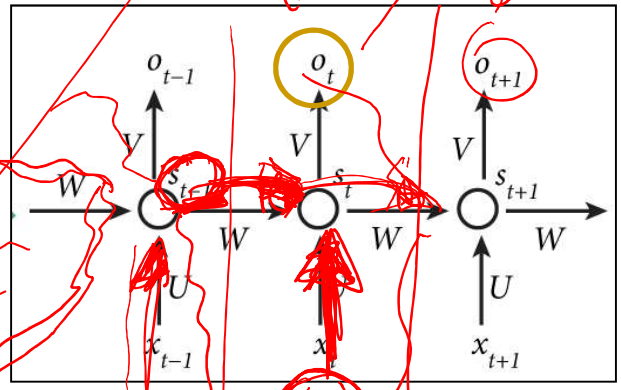
Output at time t=2 is:

O_2

0.2155
0.4279
0.1667
0.1898

Recall that V was:

h	e	l	o
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1



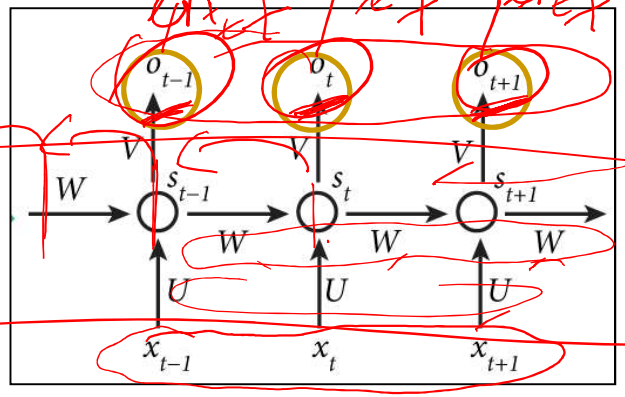
So, at time t=2:

- 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

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

$$Error(T, O) = -\sum_{t=1}^{n}(T_t \times \log(O_t))$$ // 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

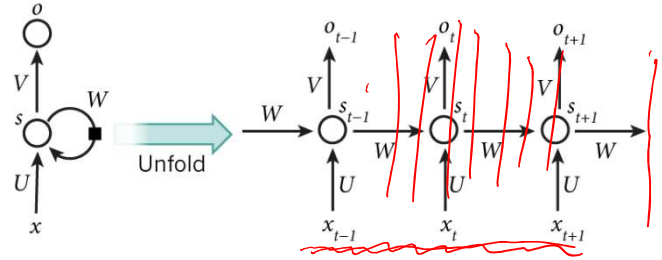
3. Update weights

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"



$$\delta_n = O_n\,(1 - O_n) \times (O_n - Tn)$$

…

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

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

$$\delta_3 = O_3\,(1 - O_3) \times \sum(w\,\delta_4)$$

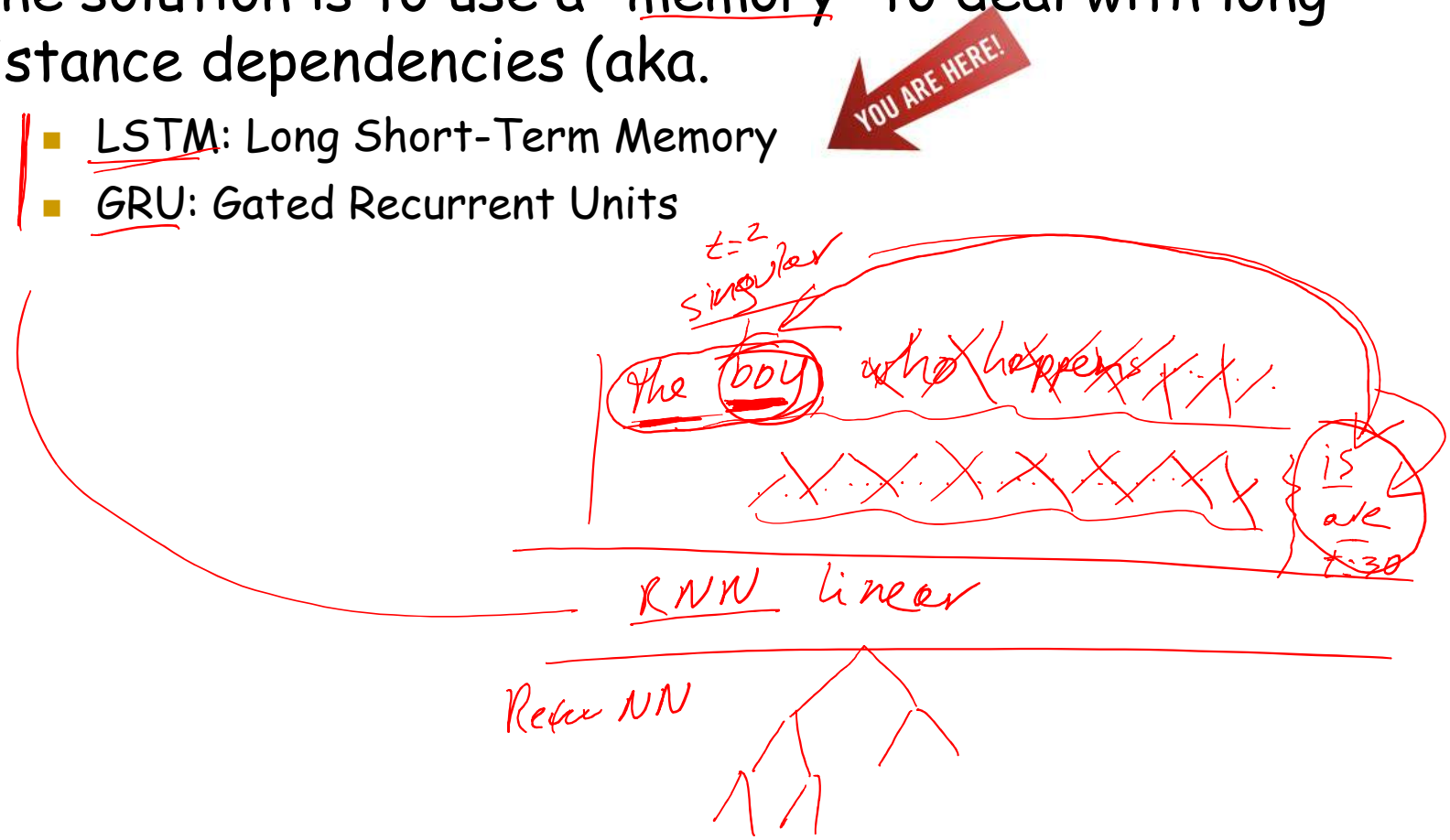
…

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

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

Problems with vanilla RNN

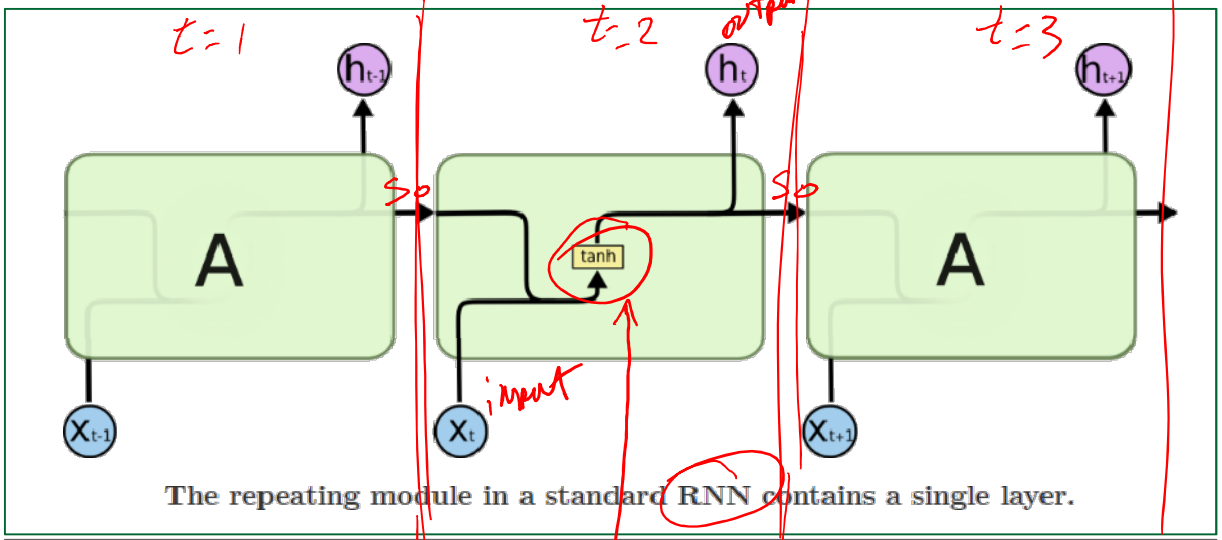
- One solution is to use a "memory" to deal with long distance dependencies (aka.
  - LSTM: Long Short-Term Memory
  - GRU: Gated Recurrent Units



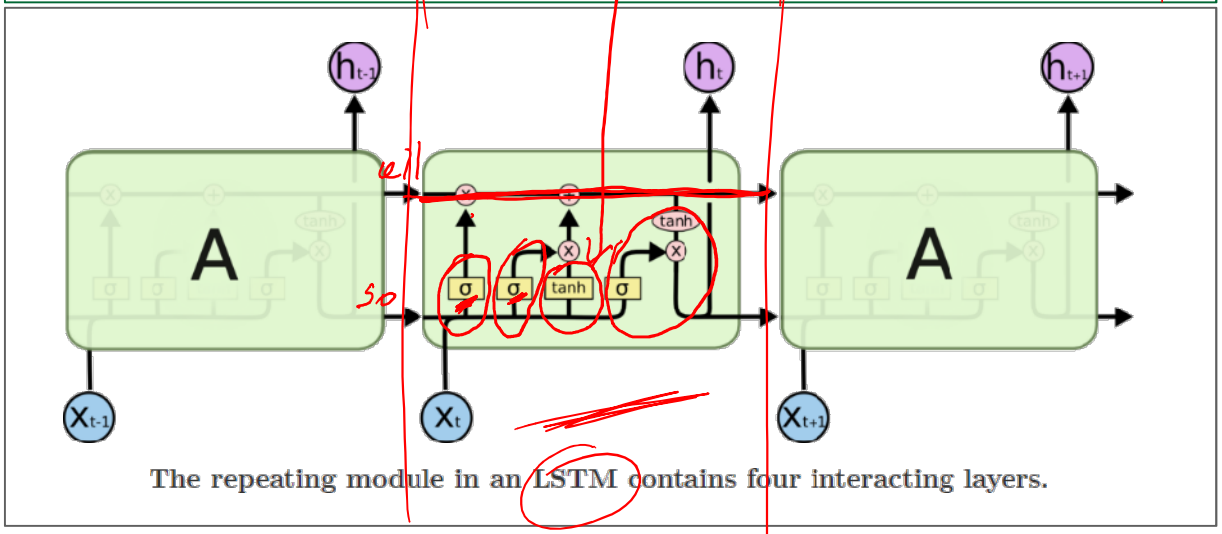
LSTMs

- 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



The repeating module in a standard RNN contains a single layer.



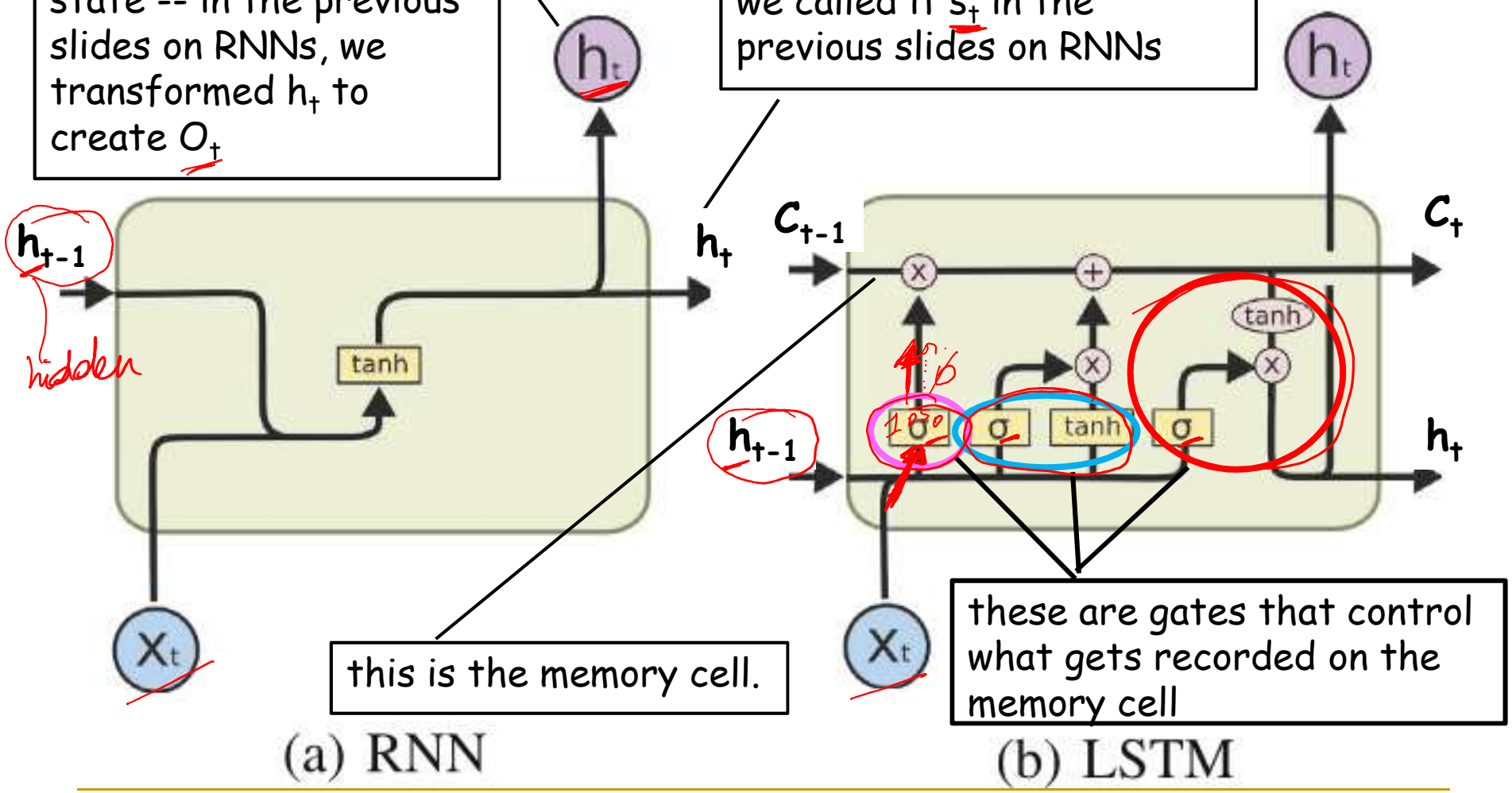
The repeating module in an LSTM contains four interacting layers.

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

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



this is the memory cell.

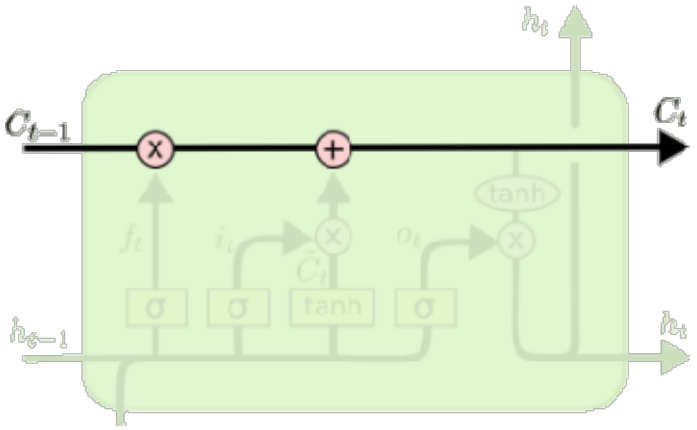
these are gates that control what gets recorded on the memory cell

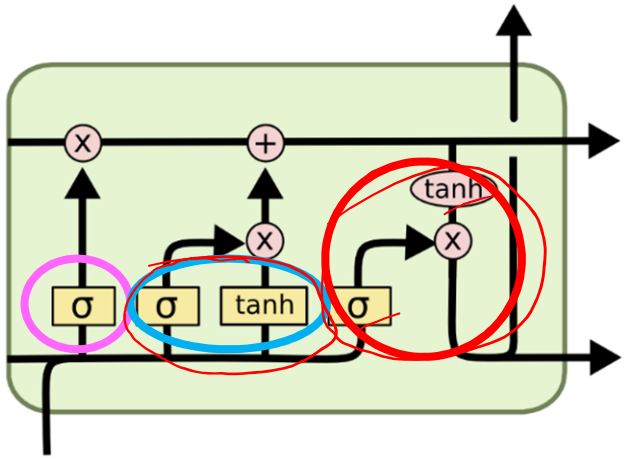
(a) RNN (b) LSTM

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

Intuition Behind LSTMs

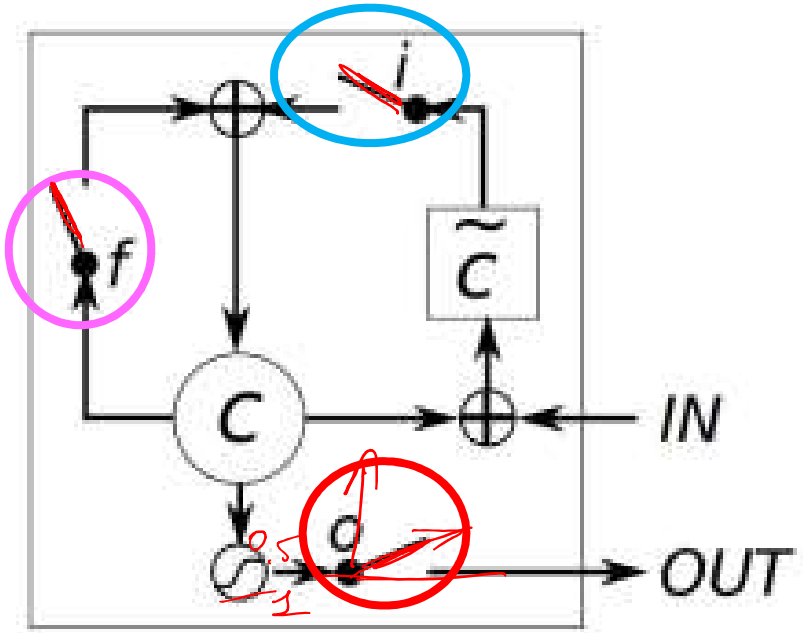
- The key to LSTMs is the **cell state C_t**
  - The horizontal line running through the top of the diagram
- Information can be stored in the cell or removed from the cell through 3 gates:
  1. Forget gate – decides/learns what info to remove from the cell
  2. Input gate – decides/learns what new info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

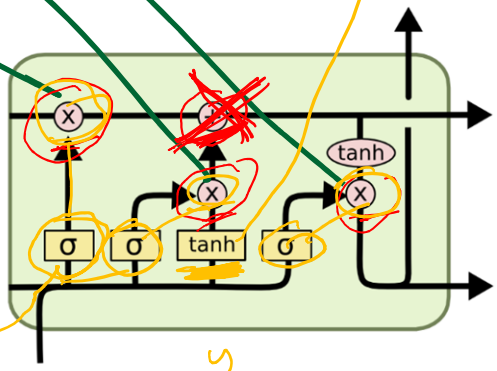


(a) Long Short-Term Memory

(Chung et al., 2014)

Intuition behind the gates

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



- *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

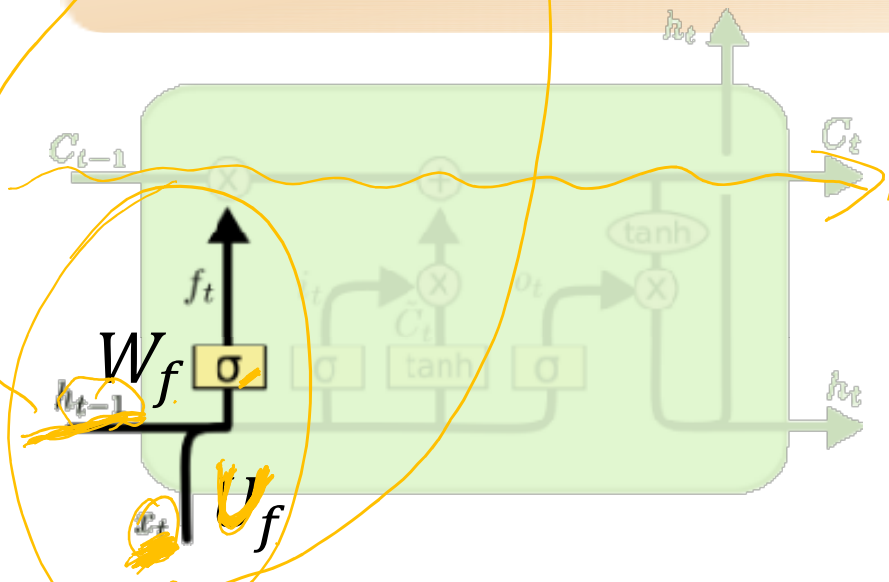
LSTM Walk Through: Forget Gate

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 sons are sick, but my daughter is not.

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



$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$$

sigmoid

forget gate

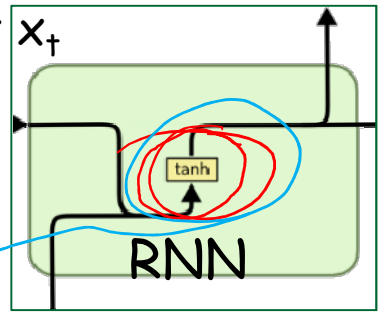
bias

eg: $f_t = [0 \ 0.5 \ 1]$

LSTM Walk Through: Input Gate

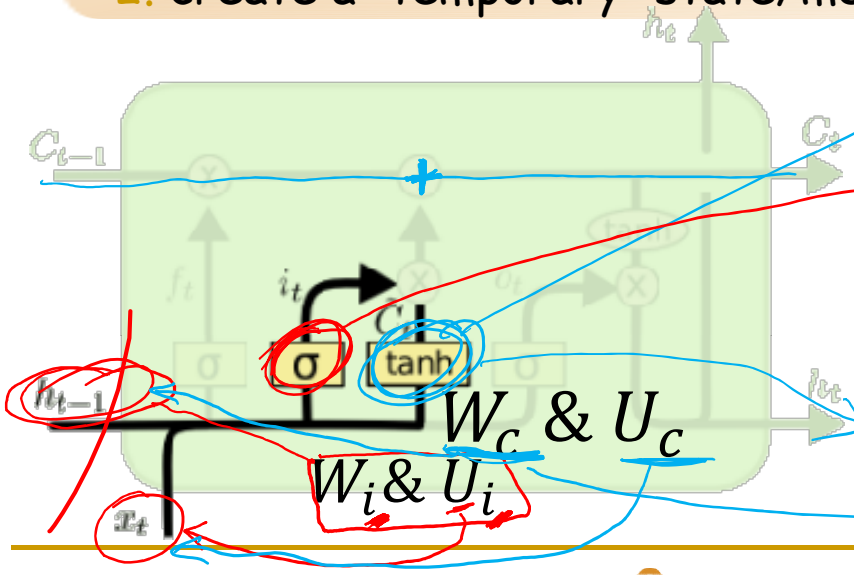
Input gate learns what new information will be stored in the cell

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 $\tilde{C}_t$



1. $i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$

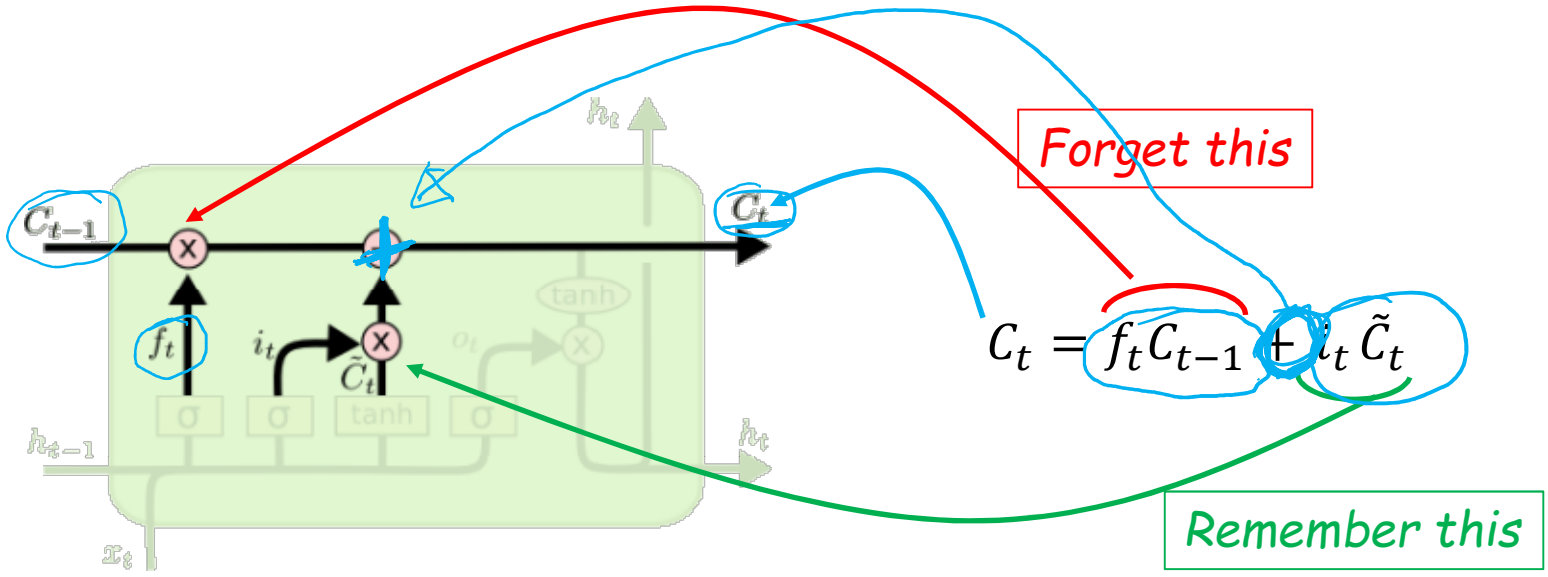
eg: $i_t = [0 \quad 0.5 \quad 1]$

2. $\tilde{C}_t = tanh(W_c h_{t-1} + U_c x_t + b_c)$

LSTM Walk Through: Update the cell

Finally, we create the current cell state C_t as:

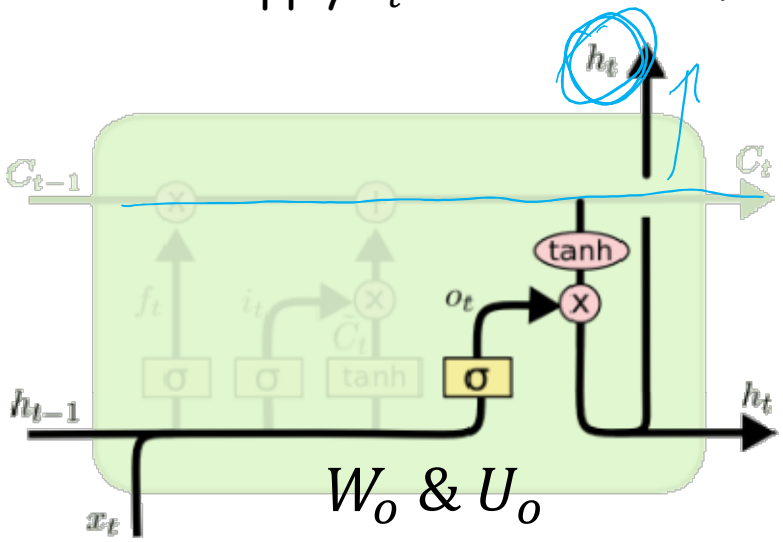
- The previous cell state $C_{t-1} \times f_t$ (the forget vector)
- Then we add (+) the new info to remember $i_t \times \widetilde{C}_t$



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



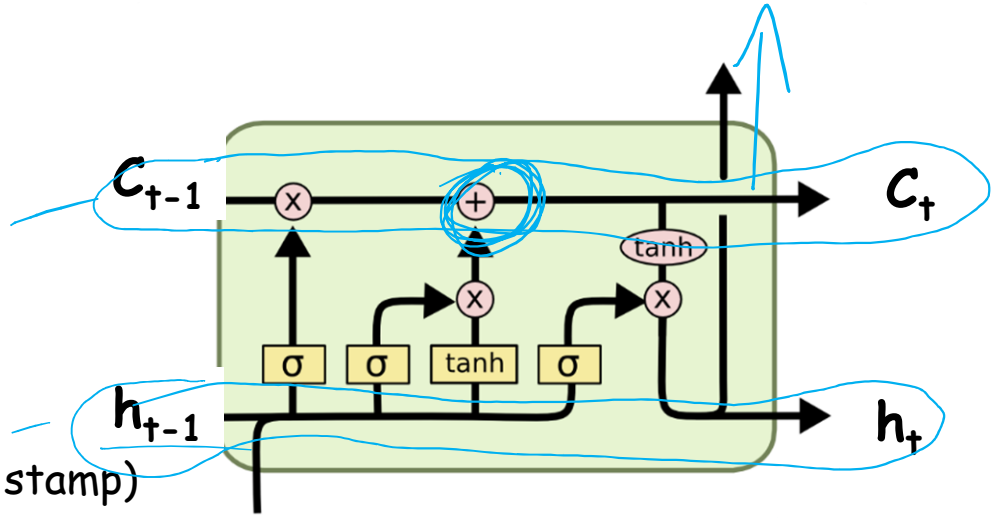
1. $o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$

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

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

LSTM - Recap

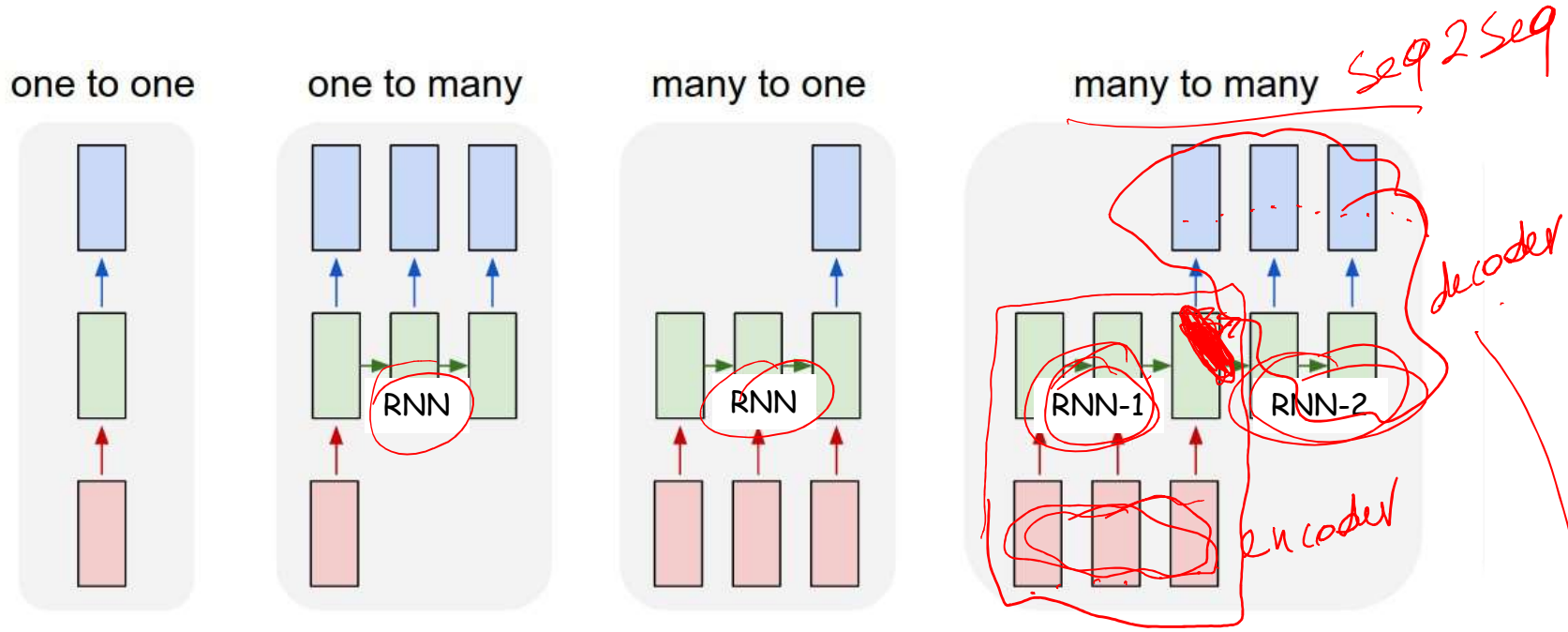
- The keys to LSTMs is that:
  - the model learns what to
    1. Forget (from the past)
    2. Input (from the present)
    3. Output (at the current time stamp)
  - stores this info in the **cell state Ct**
  - and updates to the cell state is based on addition, not multiplication



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



(1) Vanilla ANN -- input: 1 unit, output: 1 unit (e.g. image classification)

(2) RNN – input: 1 unit, output: sequence (e.g. image captioning)

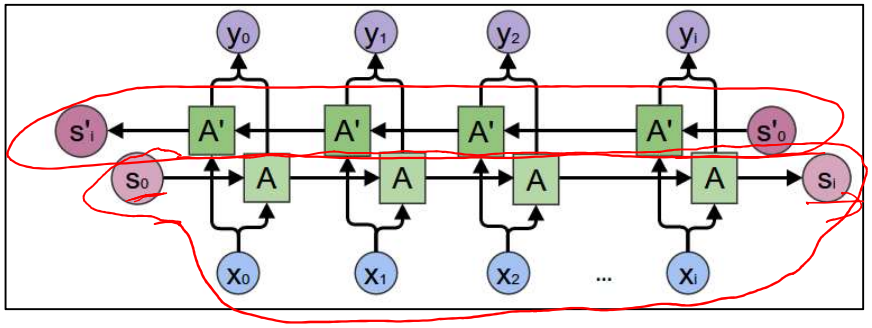
(3) RNN – input: sequence, output: 1 unit (e.g. sentiment analysis)

(4) RNN – input: sequence, output: sequence (e.g. Machine Translation)

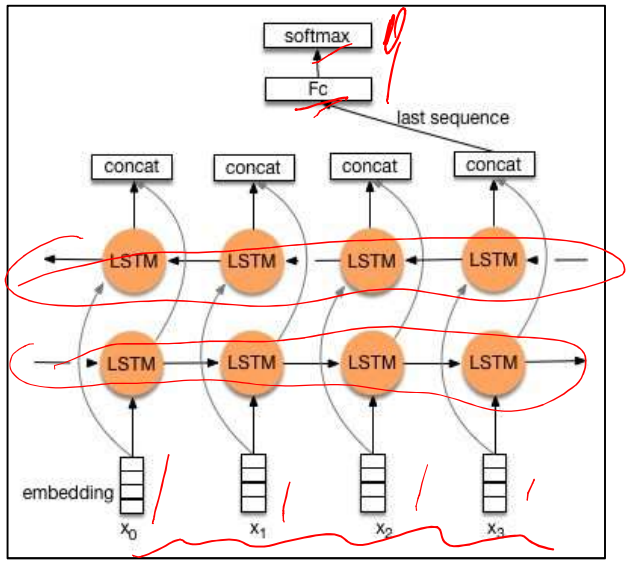
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Bidirectional LSTM

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



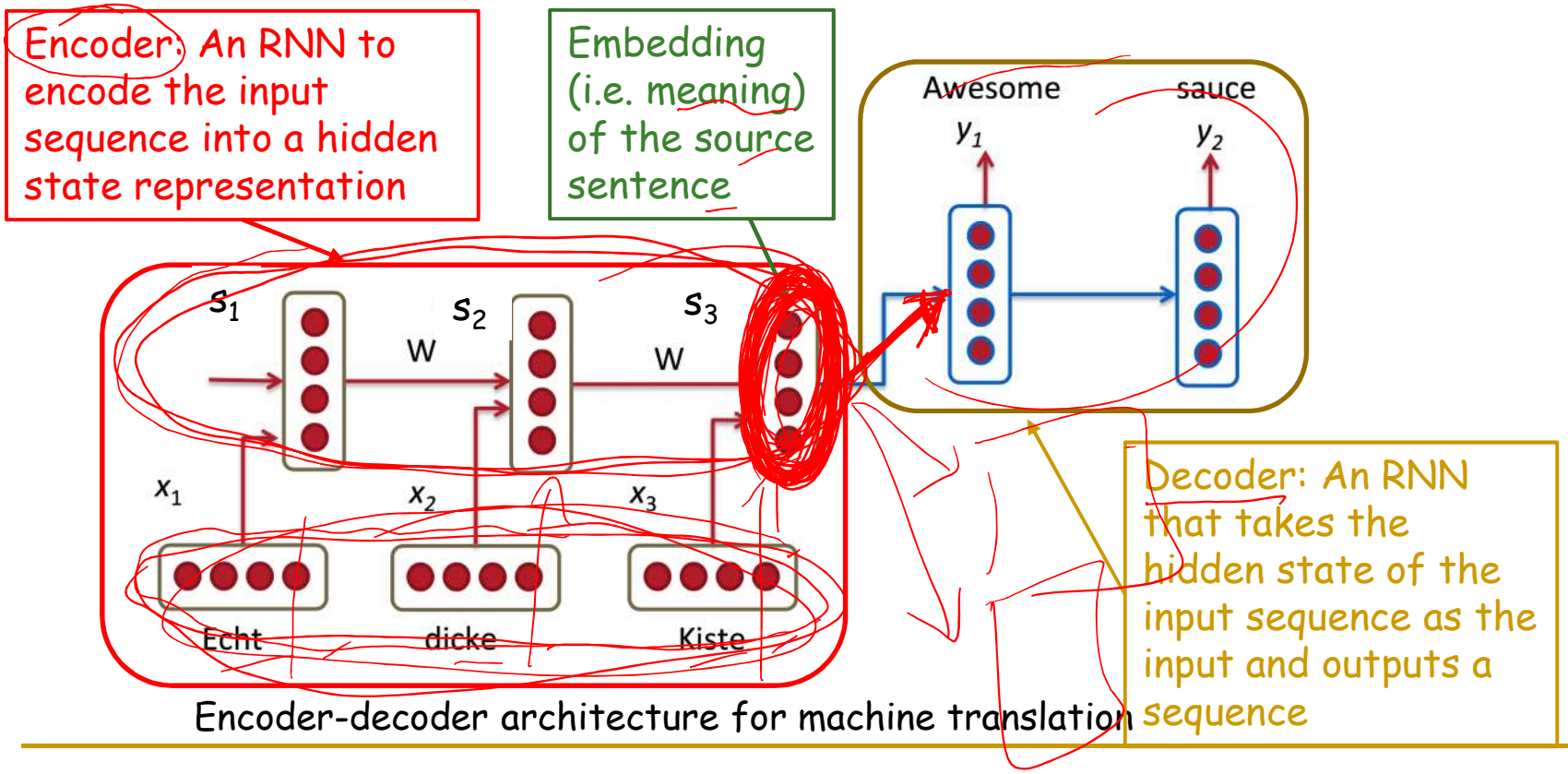
Ex: A classifier that feeds the sentence in both directions



https://towardsdatascience.com/underastanding-bidirectional-rnn-in-pytorch-5bd25a5dd66

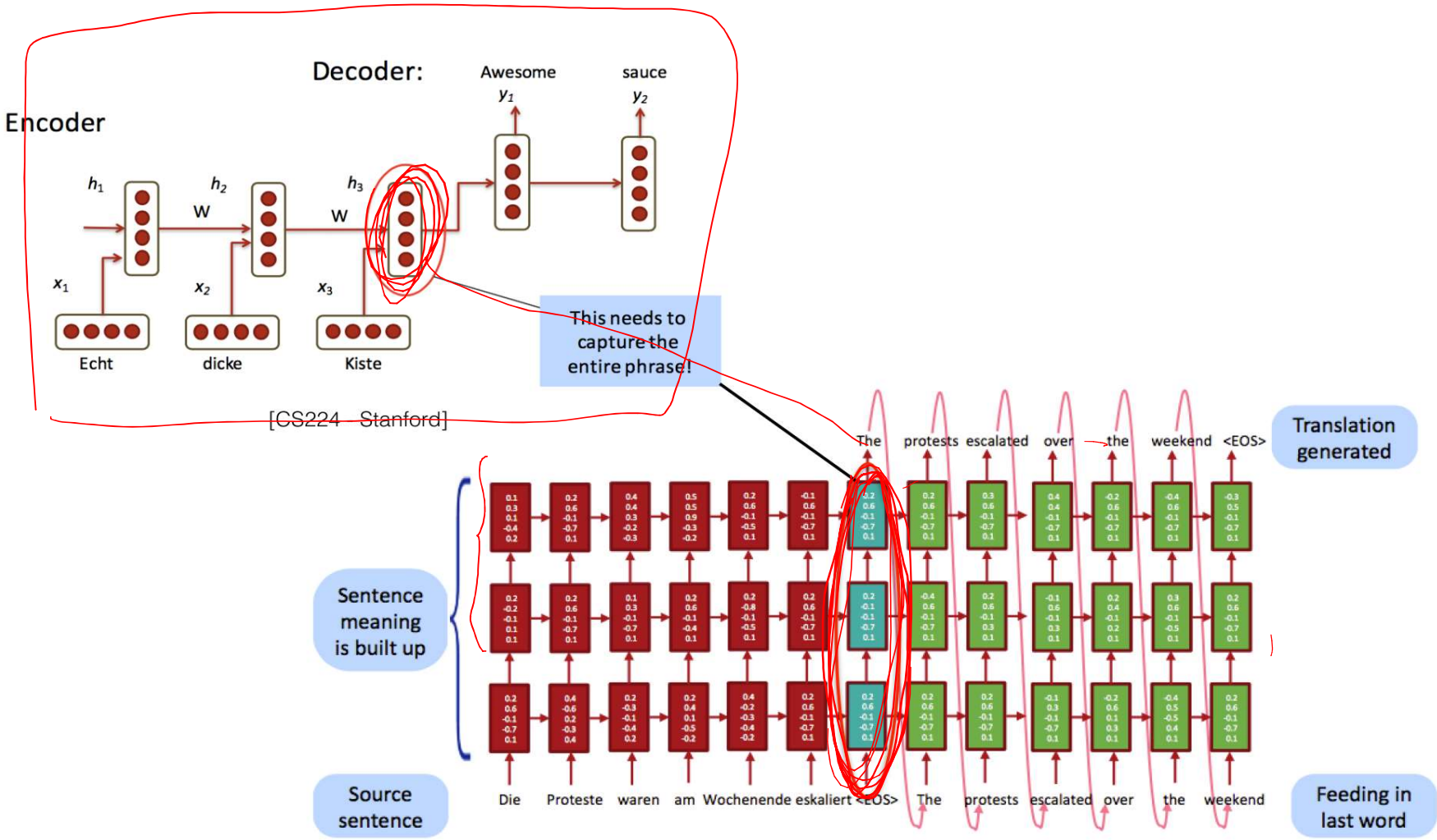
Seq2seq for Machine Translation

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

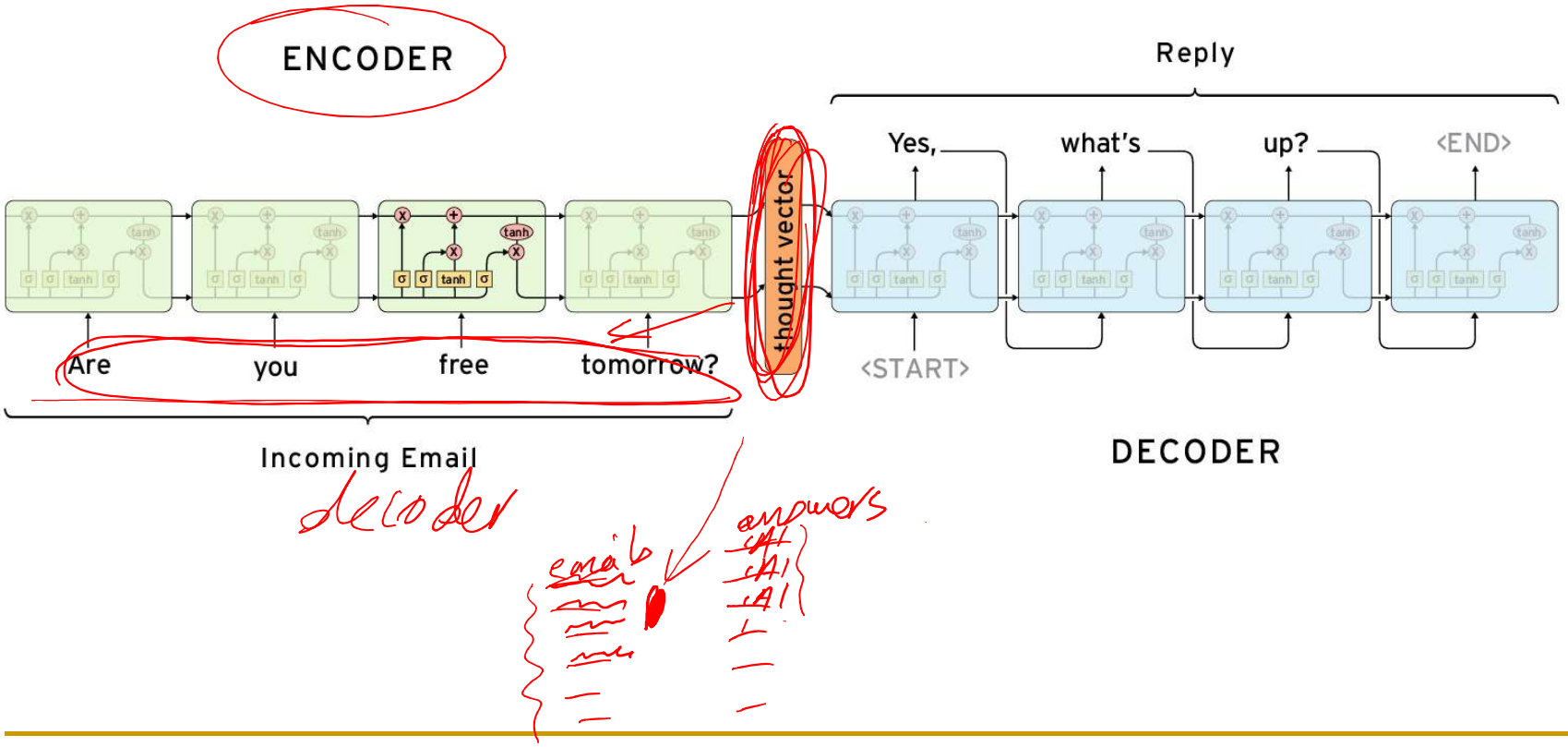


Encoder-decoder architecture for machine translation

Seq2seq for Machine Translation

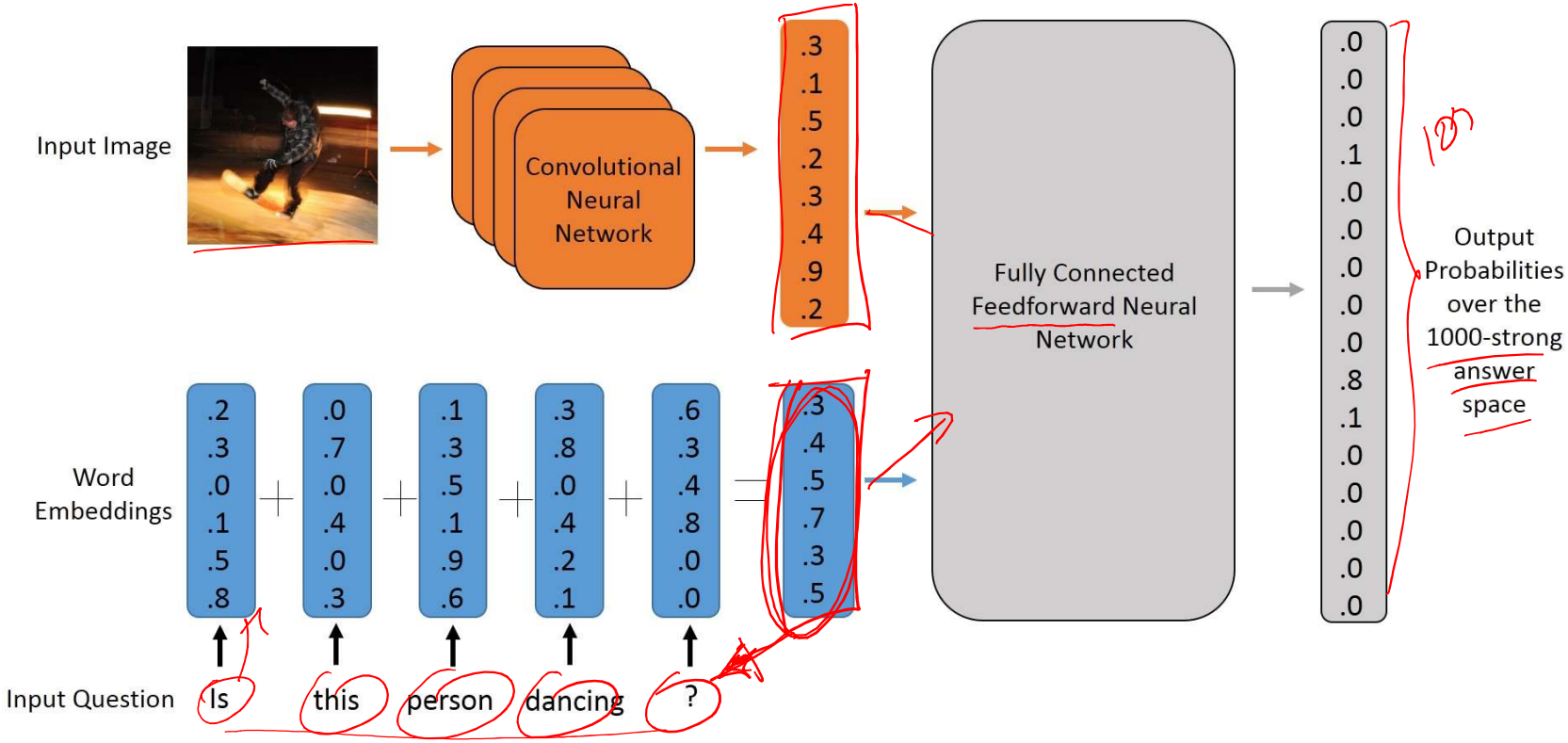


Seq2seq for Conversational Agents



http://www.wildml.com/2016/04/deep-learning-for-chatbots-part-1-introduction/

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

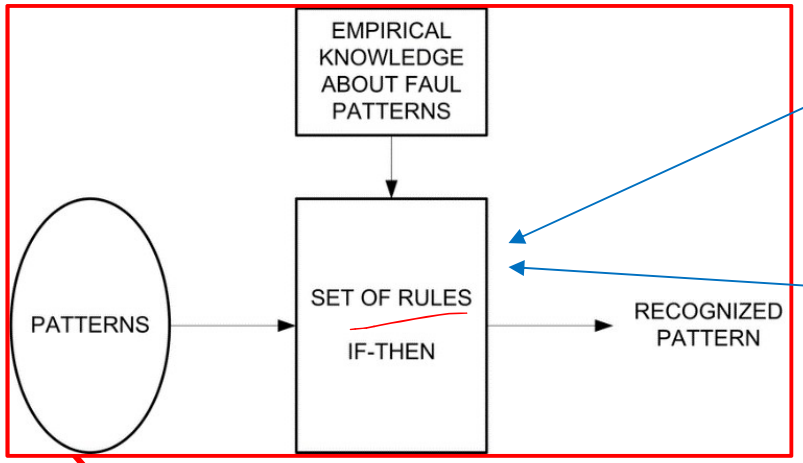
1. Introduction ✓
2. Bag of word model ✓
3. n-gram models ✓
4. Deep Learning for NLP ✓
   1. Word Embeddings ✓
   2. Recurrent Neural Networks ✓

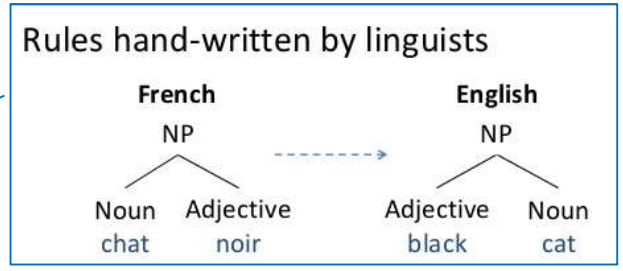
you made it!

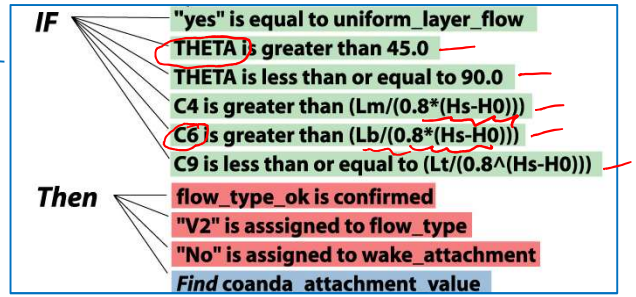
here are some final words...

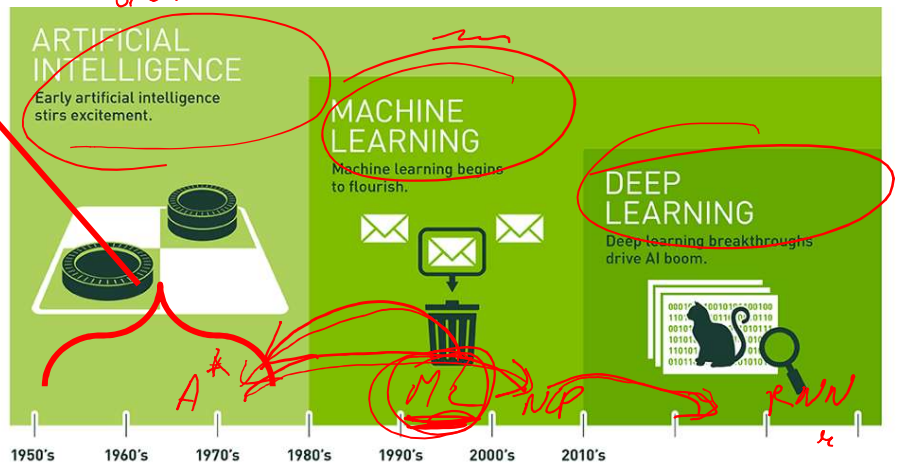
History of AI

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







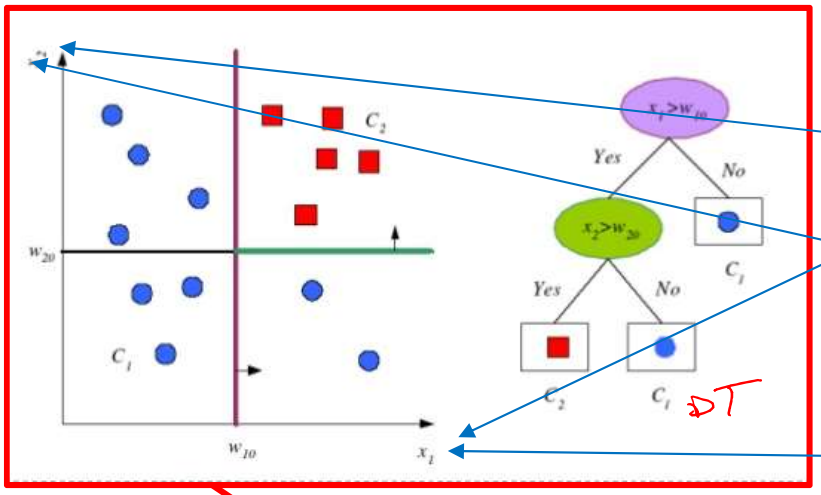


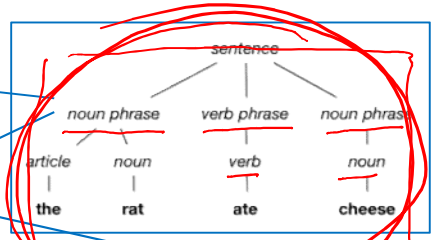
https://www.researchgate.net/profile/Dubravko_Miljkovic/publication/268239364/figure/fig30/AS:394719407427587@147111

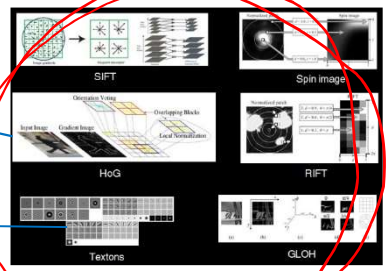
http://www.cormix.info/images/RuleTreeExample.jpg

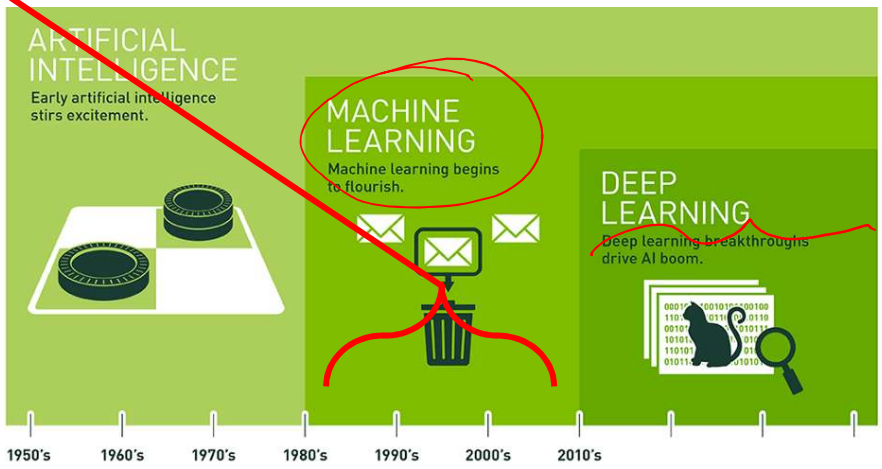
History of AI

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



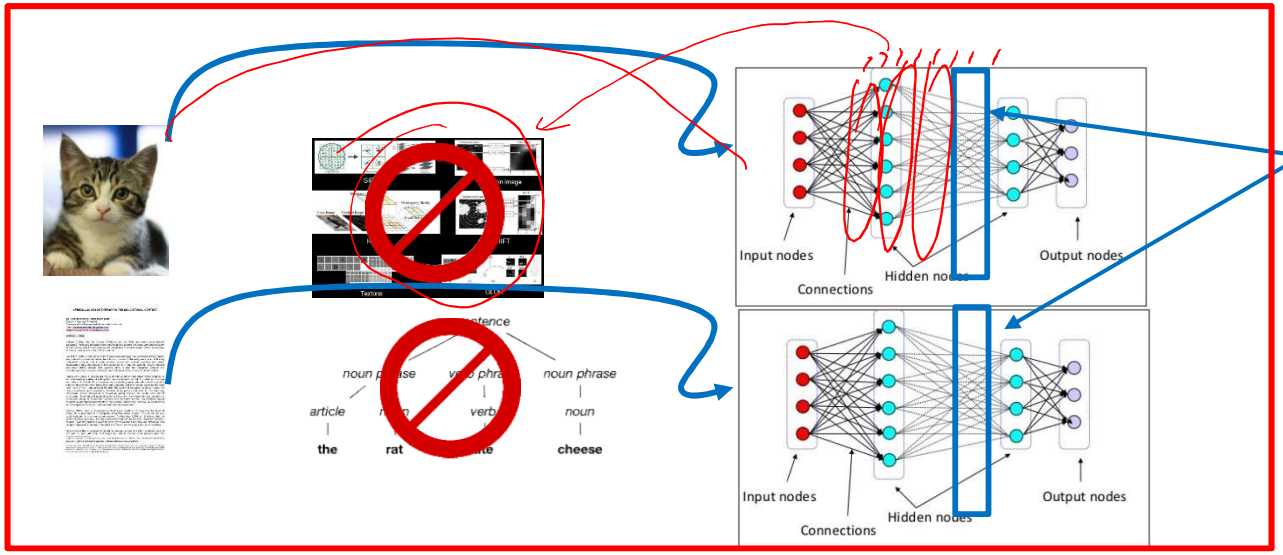




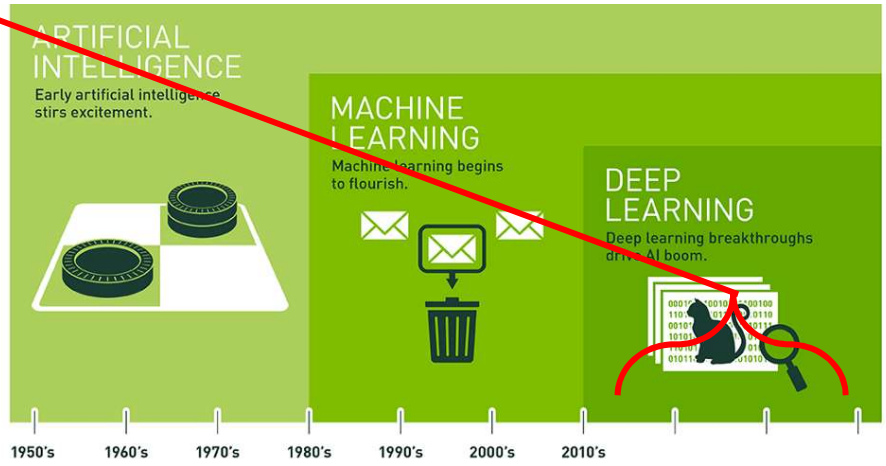


https://www.linkedin.com/pulse/goedels-incompleteness-theorem-emergence-ai-eberhard-schoeneburg/

History of AI



Rules AND features learned from the data



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