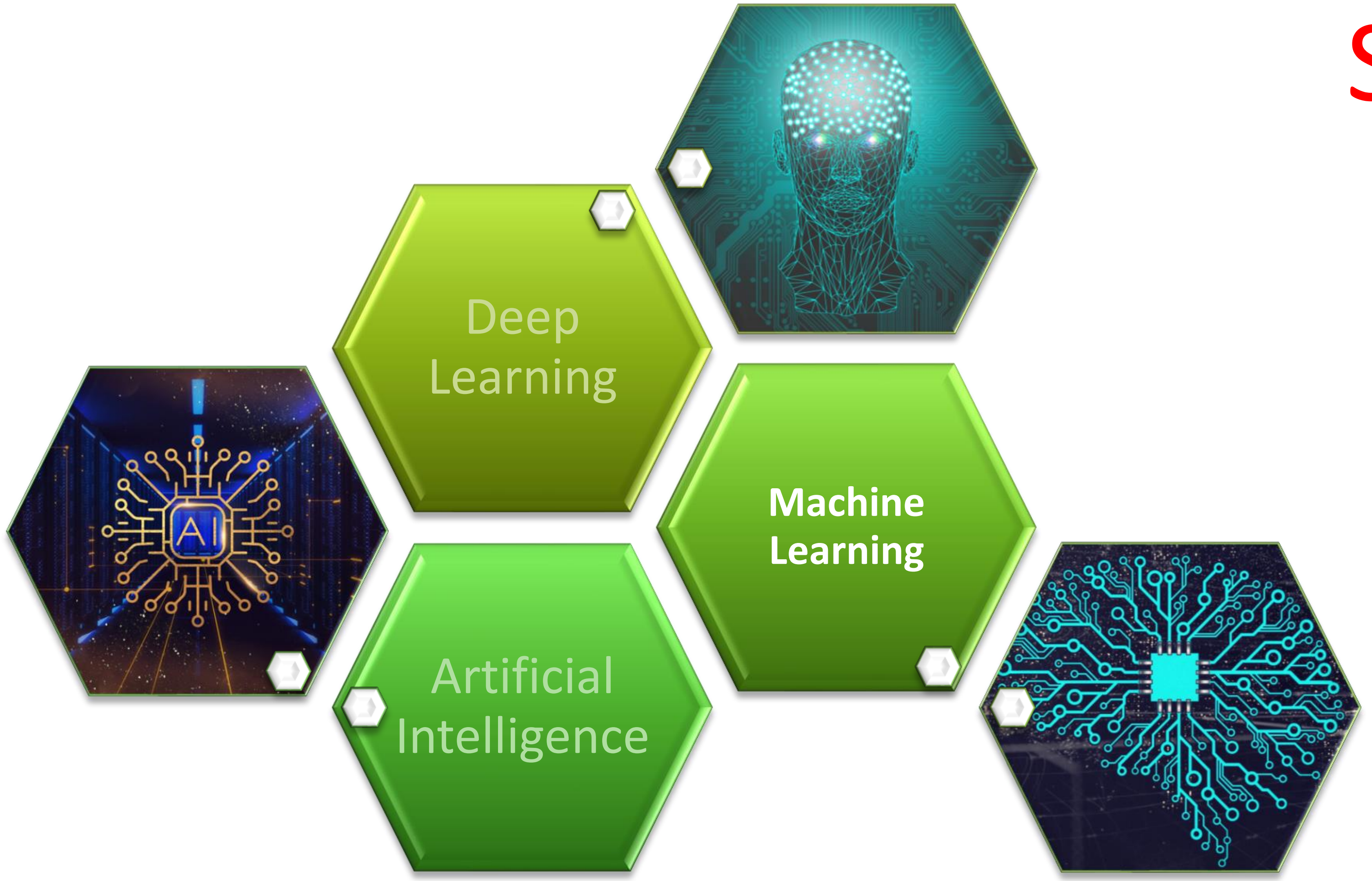
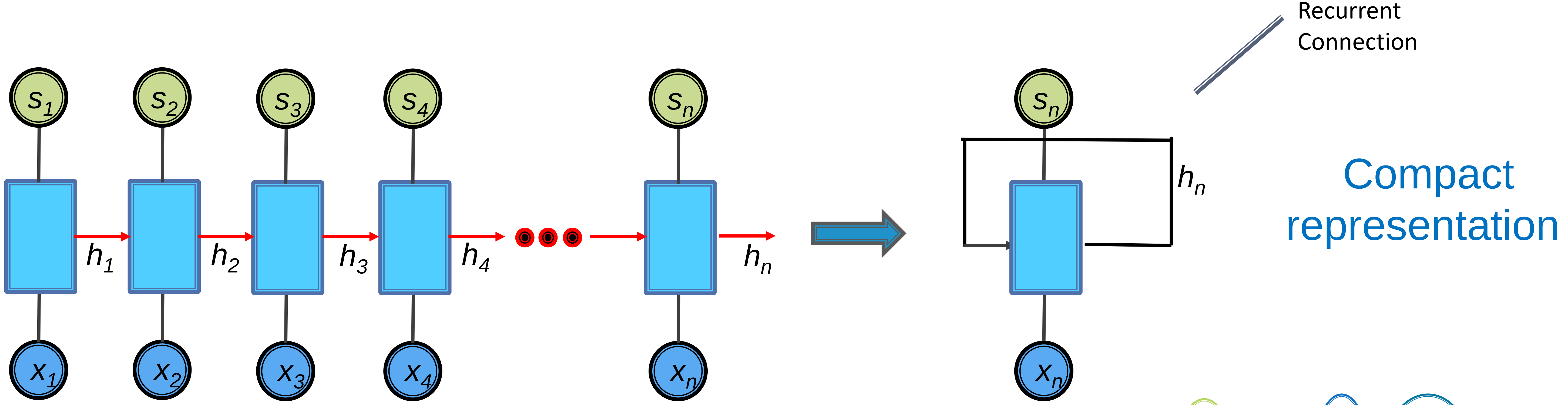


Artificial Intelligence Summer Internship/USRF

- Day 7: **LSTM models and GRU**

Recurrent Neural Network



- New input x_n at each time step
- Output s_n is generated by applying **same function** *f* at each time step
- At each time step, h_n is updated as a sequence of input is processed

$$s_n = f(x_n, h_{n-1})$$

output input past memory

Limitations of RNN

- Gradient calculation involves many factors of weights and contribution of activation function
- This may lead to:

Weight Updation

$$W \leftarrow W - \eta \frac{\partial L}{\partial W}$$

Exploding Gradient

$$\frac{\partial L}{\partial W} \gg 1 \rightarrow resulting\ in\ NAN\ values$$

- makes learning unstable

Vanishing Gradient

$$\frac{\partial L}{\partial W} << 1 \rightarrow resulting\ in\ 0\ values$$

- Short term dependencies

"the stars shine in the ?" → sky ***(RNN works good here)***

- Long term dependencies

"I grew up in Spain…................................ I speak fluent Spanish".
(Difficult for RNN to remember as gap increases)

Possible Solutions

Exploding Gradient

- ❑ Gradient clipping

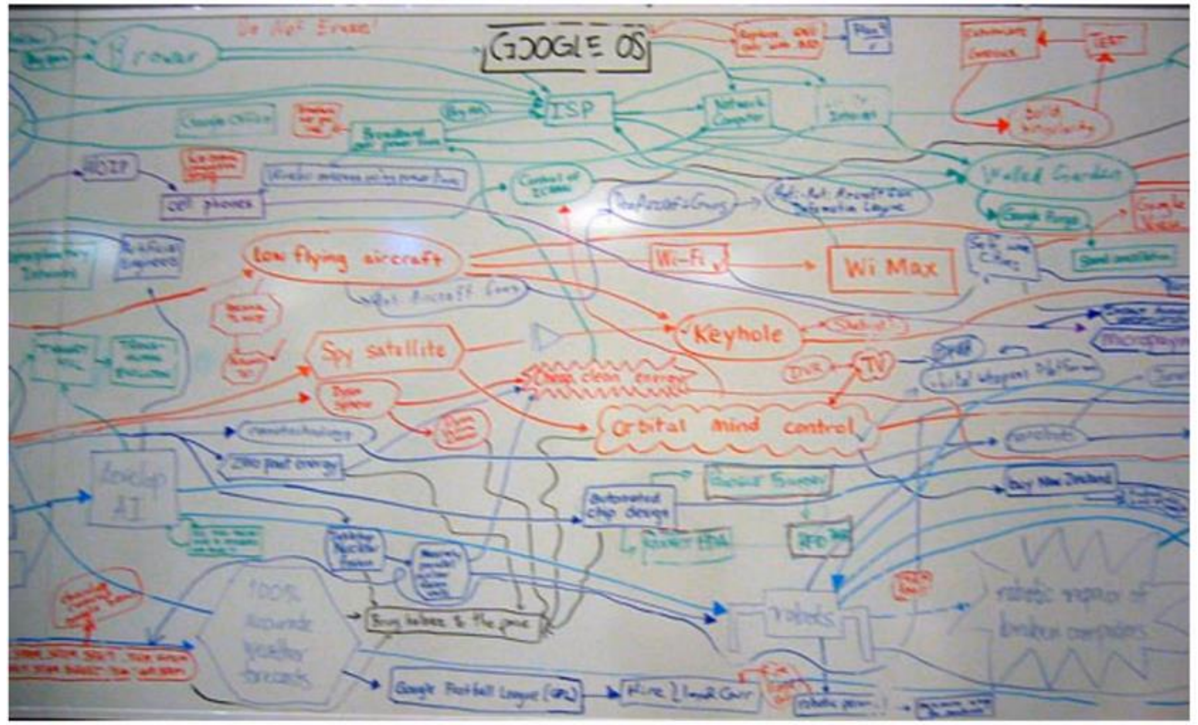
```
# inside the optimizer we are doing clipping
optimizer=tf.keras.optimizers.SGD(clipvalue=0.5)
```

Vanishing Gradient

- ❑ Activation function (ReLu)
- ❑ Weight initialization (identity matrix)
- ❑ Gated cells (LSTM,GRU,etc)

Selective Read, Selective Write, Selective Forget – The Whiteboard Analogy

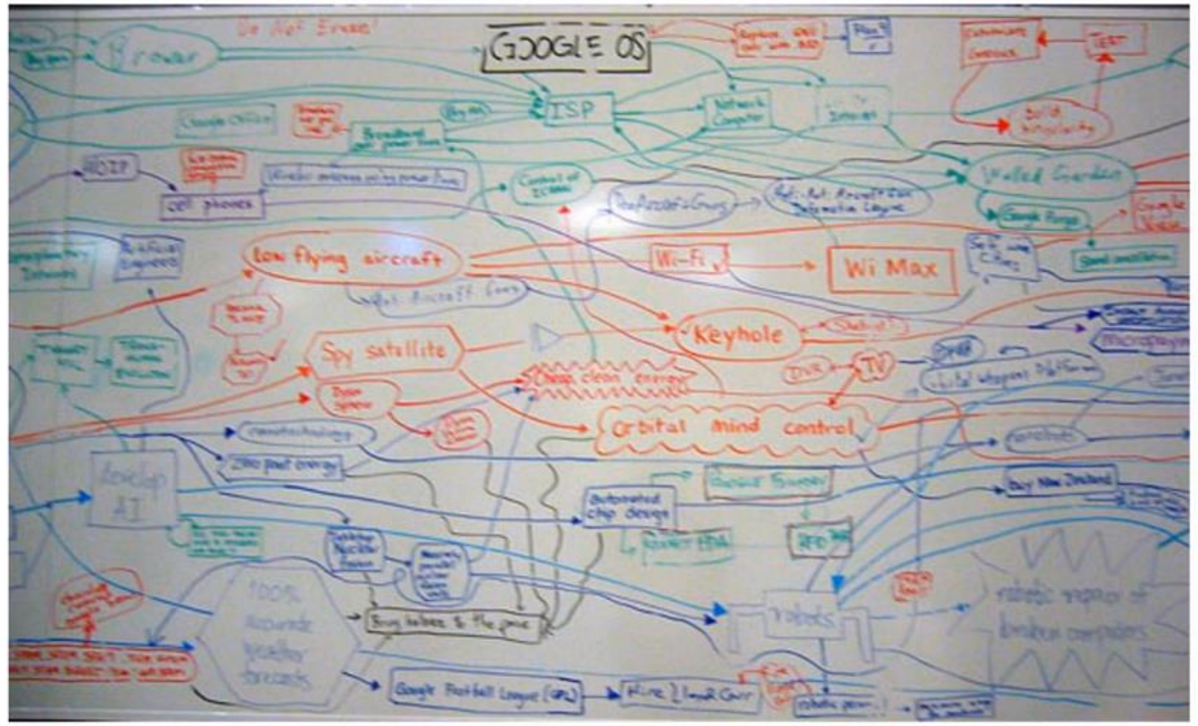
Whiteboard Analogy



Let us see an analogy for this

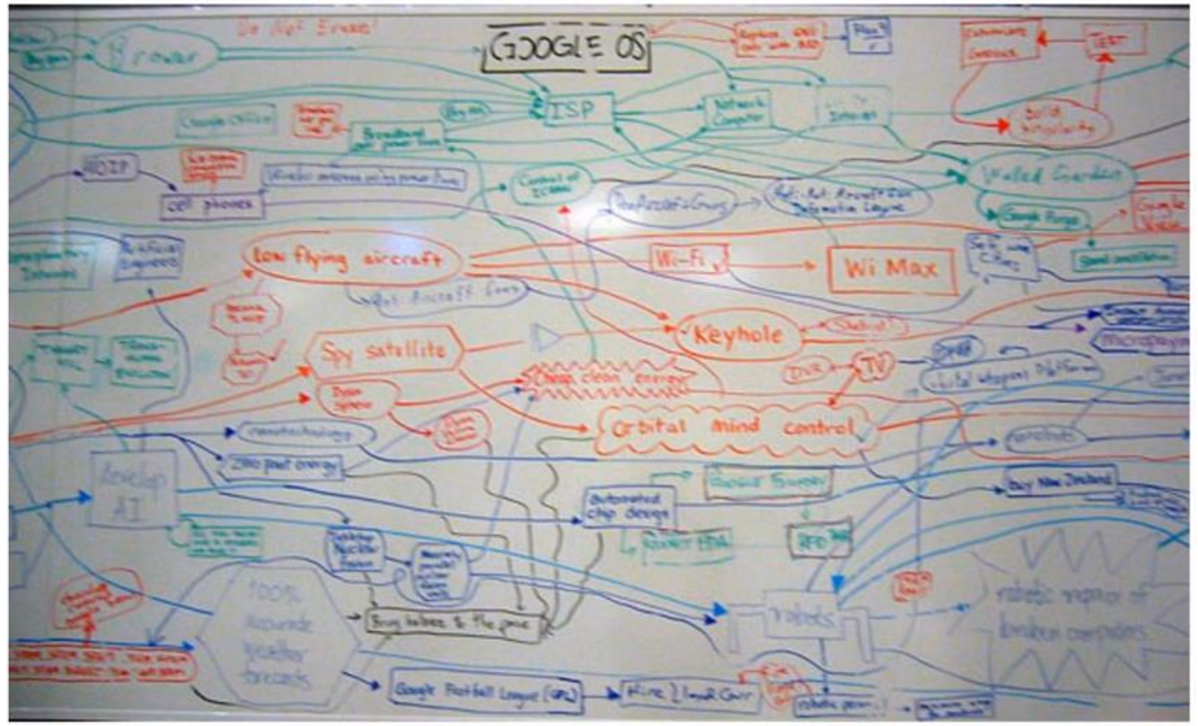
Image source:https://prvnk10.medium.com/the-whiteboard-analogy-to-deal-vanishing-and-exploding-gradients-1c0d47bfd6e1

Whiteboard Analogy



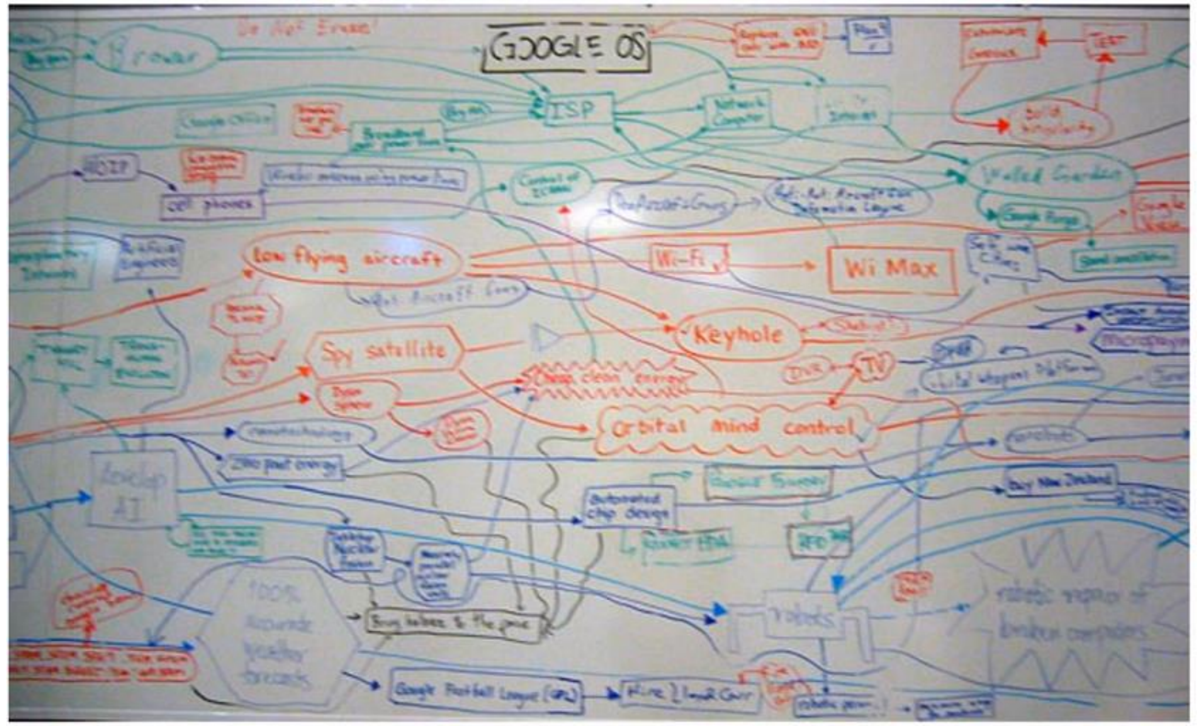
- ❑ Selectively write

Whiteboard Analogy



- Selectively write
- Selectively read

Whiteboard Analogy



- ❑ Selectively write
- ❑ Selectively read
- ❑ Selectively forget

Derive an Expression on Whiteboard

$a = 2, b = 5, c = 7, d = 10$

Compute $ac(bd + a) + ad$

Say "board" can have only 3 statements at a time.

- Selectively write

Derive an Expression on Whiteboard

$a = 2, b = 5, c = 7, d = 10$

Compute $ac(bd + a) + ad$
Say "board" can have only 3 statements at a time.

- Selectively write

1. ac
2. bd
3. $bd + a$
4. $ac(bd + a)$
5. ad
6. $ac(bd + a) + ad$

$ac = 14$

Derive an Expression on Whiteboard

$a = 2, b = 5, c = 7, d = 10$

Compute $ac(bd + a) + ad$
Say "board" can have only 3 statements at a time.

1. ac
2. bd
3. $bd + a$
4. $ac(bd + a)$
5. ad
6. $ac(bd + a) + ad$

$ac = 14$ $bd = 50$

- Selectively write

Derive an Expression on Whiteboard

$a = 2, b = 5, c = 7, d = 10$

Compute $ac(bd + a) + ad$
Say "board" can have only 3 statements at a time.

❑ Selectively read

1. ac
2. bd
3. $bd + a$
4. $ac(bd + a)$
5. ad
6. $ac(bd + a) + ad$

$ac = 14$
$bd = 50$

Derive an Expression on Whiteboard

$a = 2, b = 5, c = 7, d = 10$

Compute $ac(bd + a) + ad$
Say "board" can have only 3 statements at a time.

1. ac
2. bd
3. $bd + a$
4. $ac(bd + a)$
5. ad
6. $ac(bd + a) + ad$

$$ac = 14$$
$$bd = 50$$
$$bd + a = 52$$

- ❑ Selectively read

Derive an Expression on Whiteboard

$a = 2, b = 5, c = 7, d = 10$

Compute $ac(bd + a) + ad$
Say "board" can have only 3 statements at a time.

1. ac
2. bd
3. $bd + a$
4. $ac(bd + a)$
5. ad
6. $ac(bd + a) + ad$

$$\boxed{\begin{gathered} ac = 14 \\ bd = 50 \\ bd + a = 52 \end{gathered}}$$

- Selectively forget

Derive an Expression on Whiteboard

$a = 2, b = 5, c = 7, d = 10$

Compute $ac(bd + a) + ad$
Say "board" can have only 3 statements at a time.

1. ac
2. bd
3. $bd + a$
4. $ac(bd + a)$
5. ad
6. $ac(bd + a) + ad$

$ac = 14$
$bd + a = 52$

- Selectively forget

Derive an Expression on Whiteboard

$a = 2, b = 5, c = 7, d = 10$

Compute $ac(bd + a) + ad$
Say "board" can have only 3 statements at a time.

1. ac
2. bd
3. $bd + a$
4. $ac(bd + a)$
5. ad
6. $ac(bd + a) + ad$

$$\boxed{\begin{array}{c} ac = 14 \\ ac(bd + a) = 884 \\ bd + a = 52 \end{array}}$$

- ❑ Selectively forget

Derive an Expression on Whiteboard

$a = 2, b = 5, c = 7, d = 10$

Compute $ac(bd + a) + ad$
Say "board" can have only 3 statements at a time.

1. ac
2. bd
3. $bd + a$
4. $ac(bd + a)$
5. ad
6. $ac(bd + a) + ad$

$ac(bd + a) = 884$
$bd + a = 52$

- Selectively forget

Derive an Expression on Whiteboard

$a = 2, b = 5, c = 7, d = 10$

Compute $ac(bd + a) + ad$
Say "board" can have only 3 statements at a time.

- Selectively forget

1. ac
2. bd
3. $bd + a$
4. $ac(bd + a)$
5. ad
6. $ac(bd + a) + ad$

$$ad + ac(bd + a) = 748$$
$$ac(bd + a) = 728$$
$$ad = 20$$

Derive an Expression on Whiteboard

$a = 2, b = 5, c = 7, d = 10$

Compute $ac(bd + a) + ad$
Say "board" can have only 3 statements at a time.

1. ac
2. bd
3. $bd + a$
4. $ac(bd + a)$
5. ad
6. $ac(bd + a) + ad$

$$\boxed{\begin{gathered} ad + ac(bd + a) = 748 \\ ac(bd + a) = 728 \\ ad = 20 \end{gathered}}$$

- Selectively forget

Derive an Expression on Whiteboard

$a = 2, b = 5, c = 7, d = 10$

Compute $ac(bd + a) + ad$
Say "board" can have only 3 statements at a time.

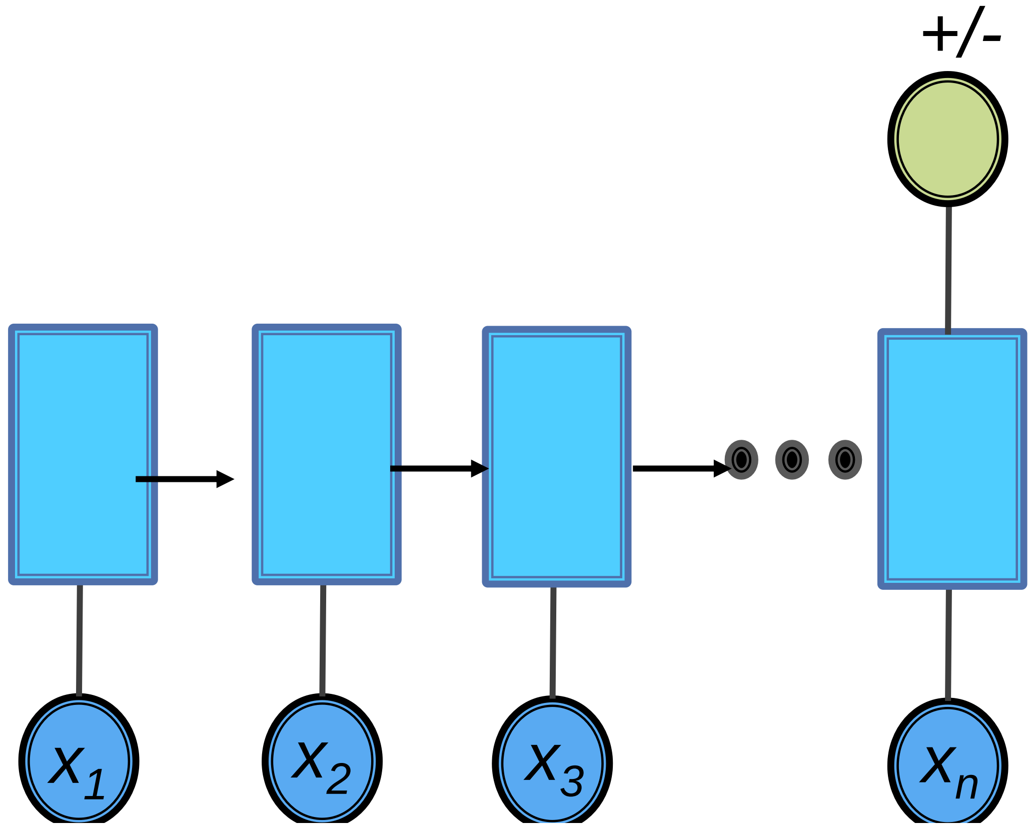
1. ac
2. bd
3. $bd + a$
4. $ac(bd + a)$
5. ad
6. $ac(bd + a) + ad$

$$ad + ac(bd + a) = 748$$
$$ac(bd + a) = 728$$
$$ad = 20$$

- Selectively forget

RNN has finite state size.
Thus, we need selective read, selective write and selective forget!!

Understand the concept using real-time example



Customers Review

Amazing! This box of cereal gave me a perfectly balanced breakfast, as all things should be. I only ate half of it but will definitely be buying again!

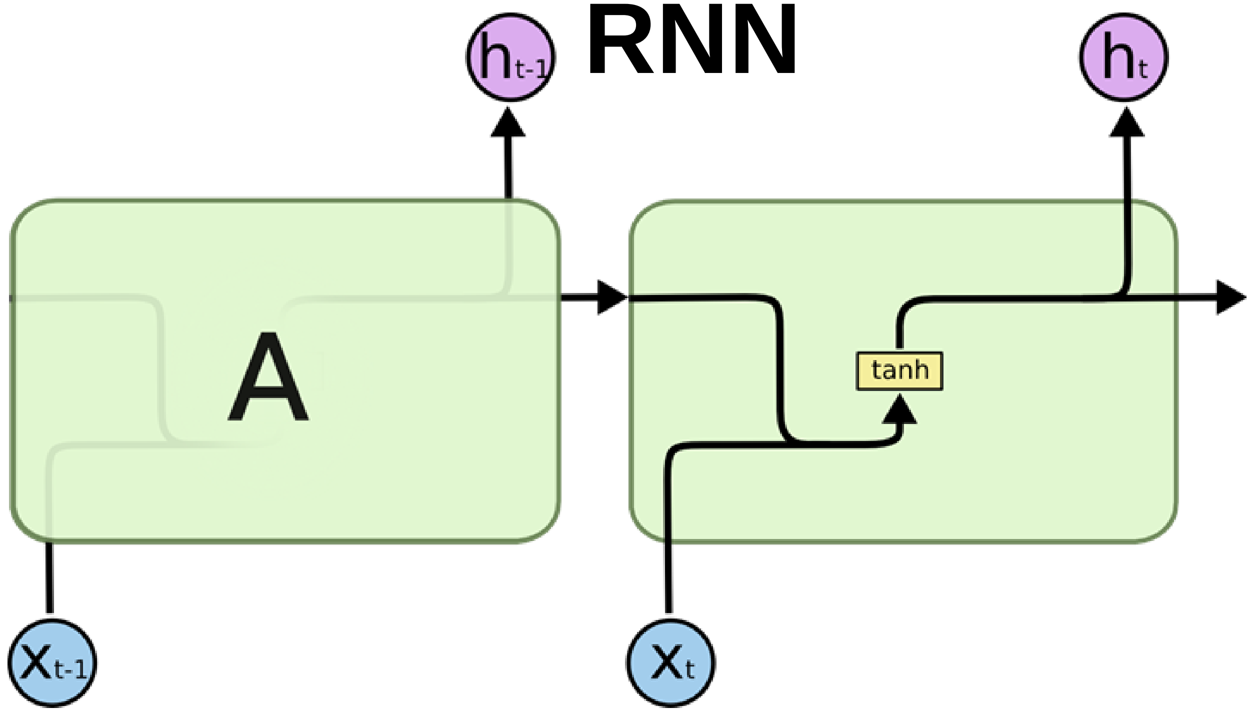
LSTMs can learn to keep only relevant information to make predictions, and forget non relevant data

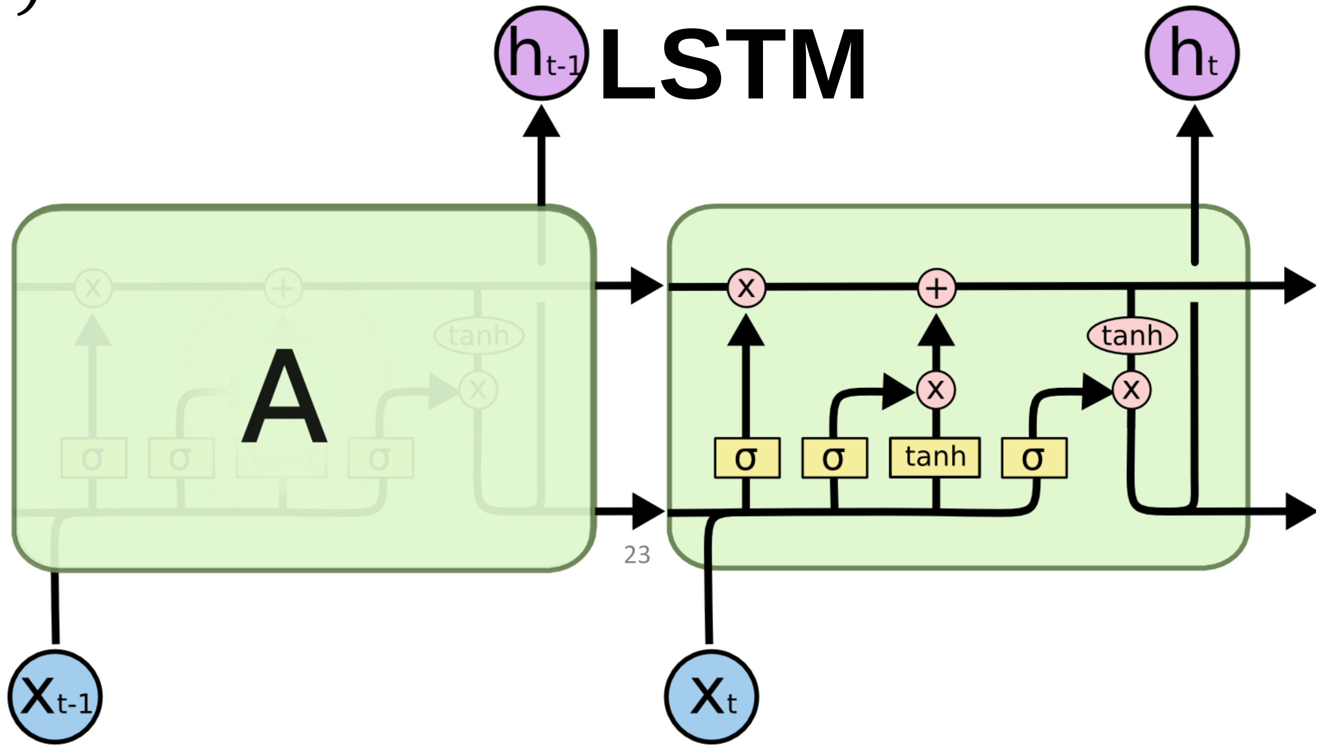
Amazing! This box of cereal gave me a **perfectly balanced breakfast**, as all things should be. I only ate half of it but **will definitely be buying again!**

Long Short Term Memory networks (LSTM)

- Introduced by Hochreiter & Schmidhuber (1997)
- LSTMs learn long-term dependencies using Cell State and Gates

$$h_t = f(h_{t-1} \cdot W, x_t \cdot U)$$



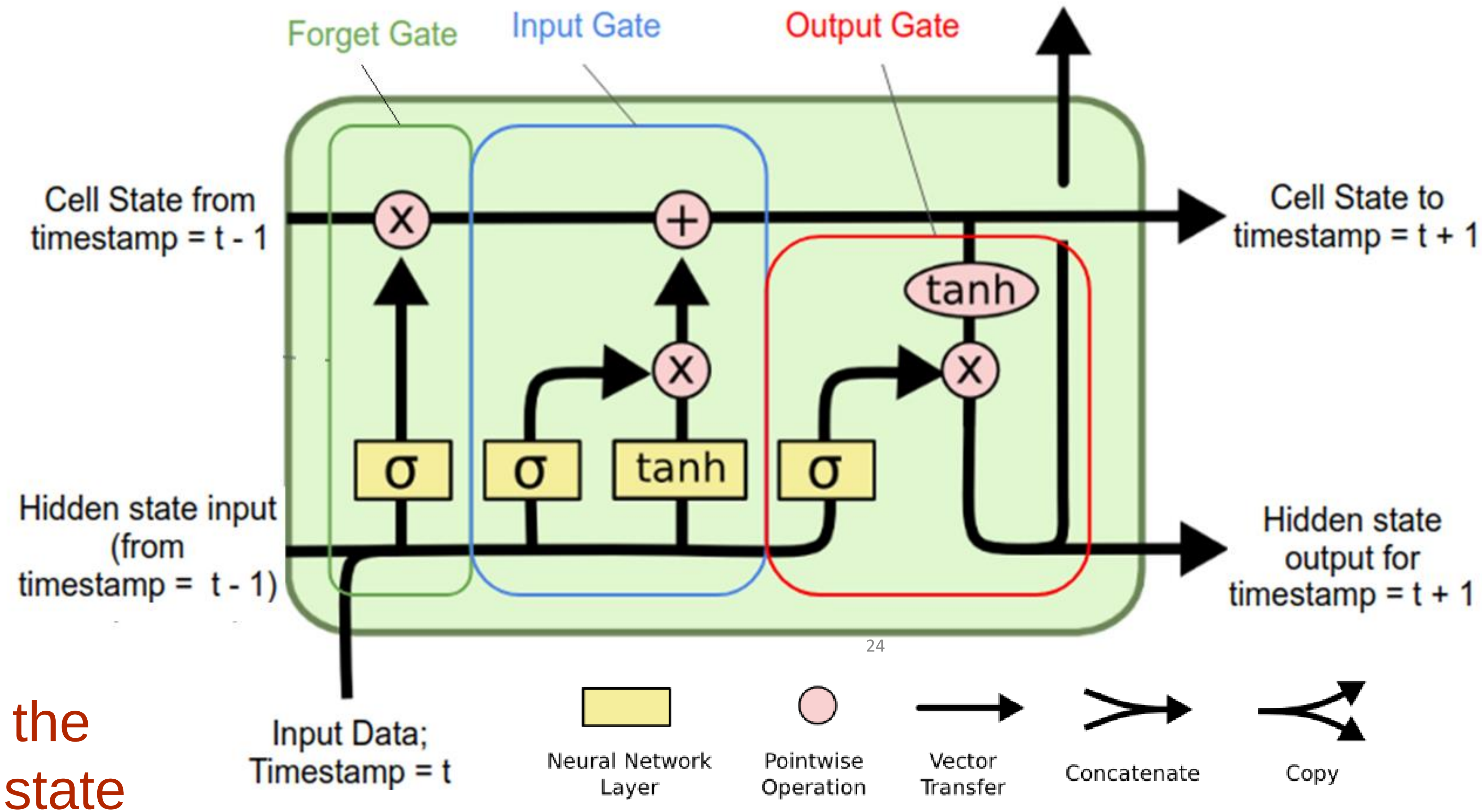


Reference: https://builtin.com/artificial-intelligence/transformer-neural-network

Long Short-Term Memory (LSTM)

- Cell State: encode information from *earlier* time steps, reducing the effect of memory loss
- Hidden State: working memory is usually called the hidden state
- Input at time step t

Regular RNNs have just the hidden state and no cell state



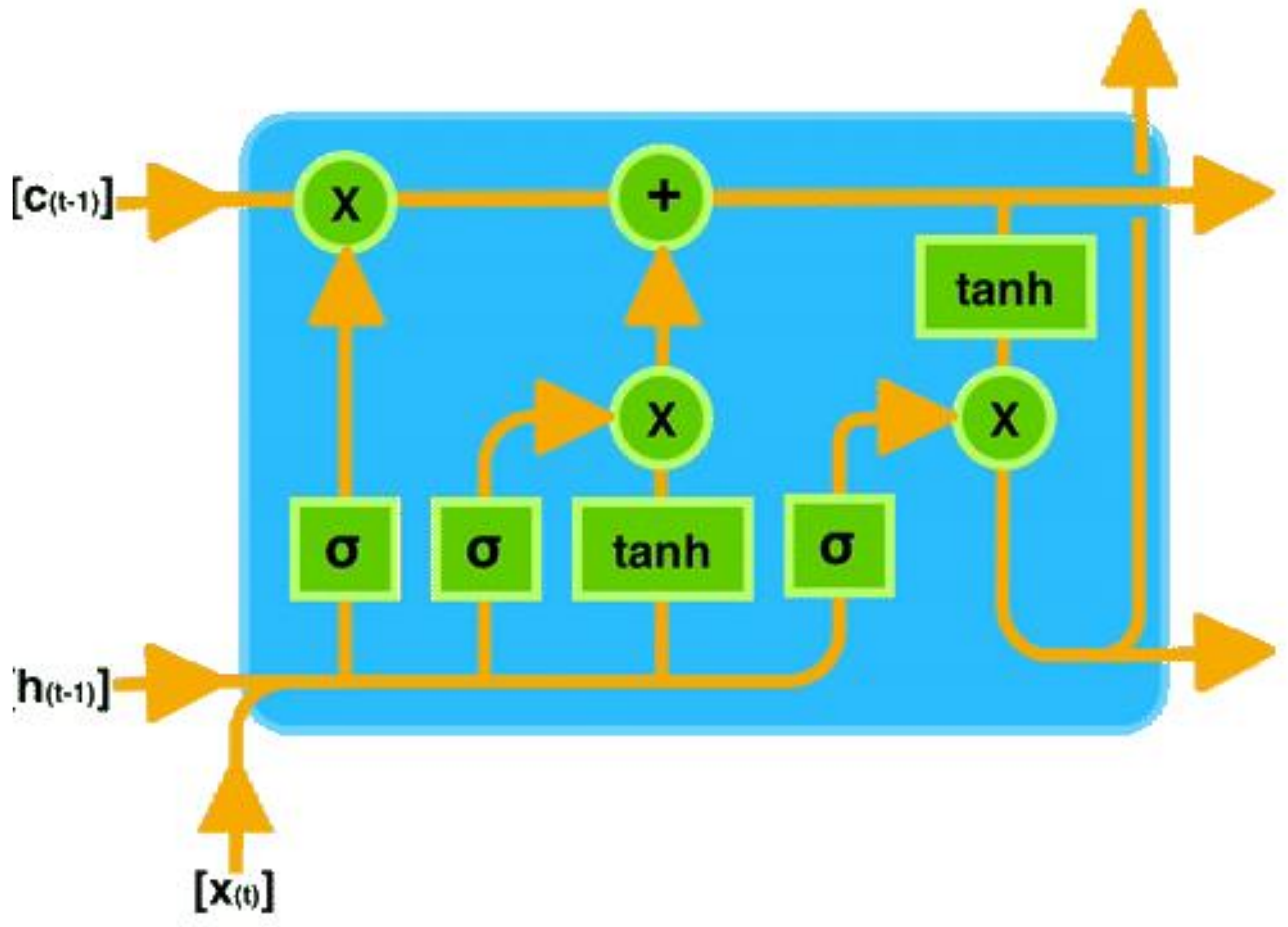
Long Short-Term Memory (LSTM)

How do LSTM work?

a) Input

b) Forget

c) Output



Source: https://medium.com/analytics-vidhya/tagged/lstm

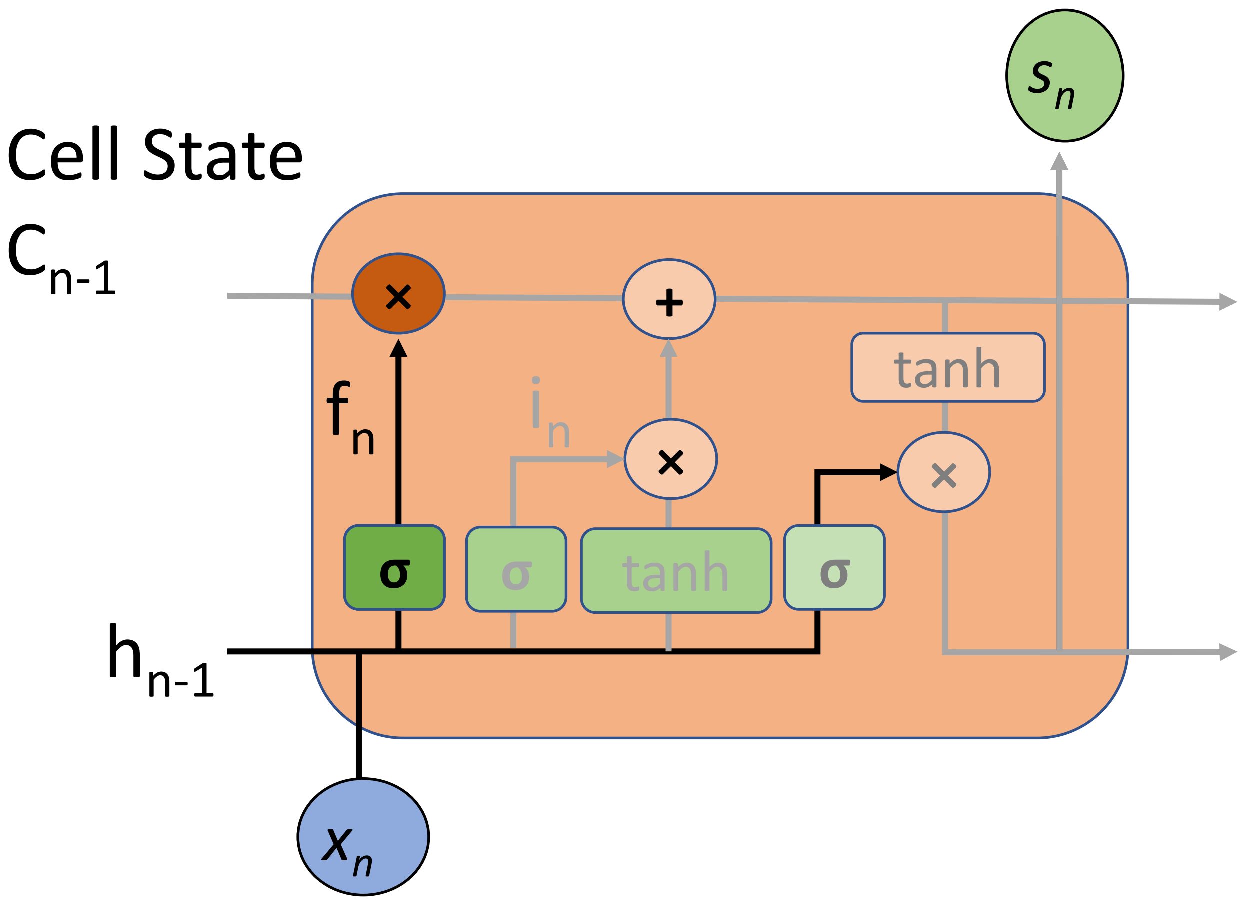
Long Short-Term Memory (LSTM)

How do LSTMs work?

a) Forget

b) Input

c) Output



Forget gate gets rid of irrelevant information

Output of Sigmoid Activation (σ):
0 => Forget,
1 => Keep

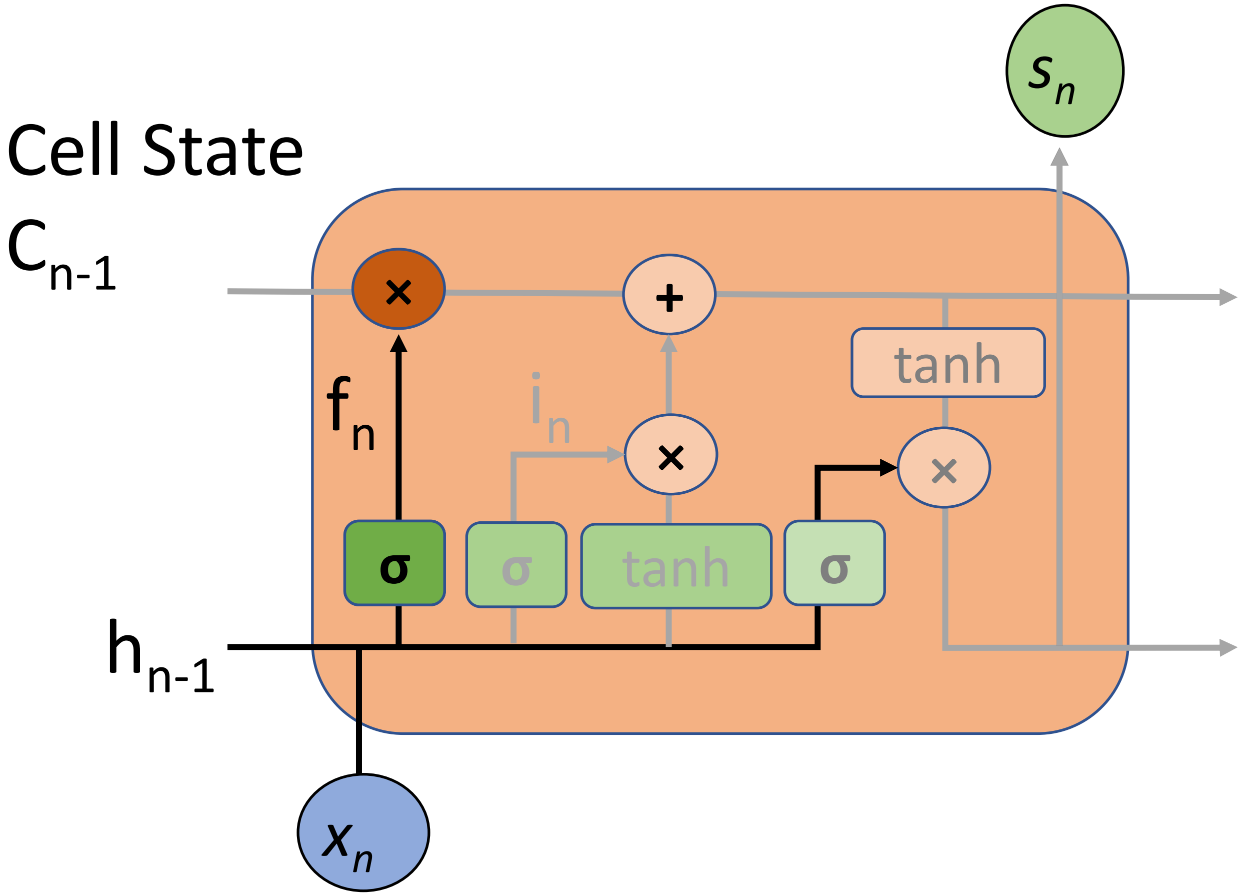
Long Short-Term Memory (LSTM)

How do LSTMs work?

a) Forget

b) Input

c) Output



Forget gate gets rid of irrelevant information

$$f_n = \sigma(W_{hf} h_{n-1} + W_{xf} x_n + b_f)$$

Output of Sigmoid Activation (σ):
0 => Forget,
1 => Keep

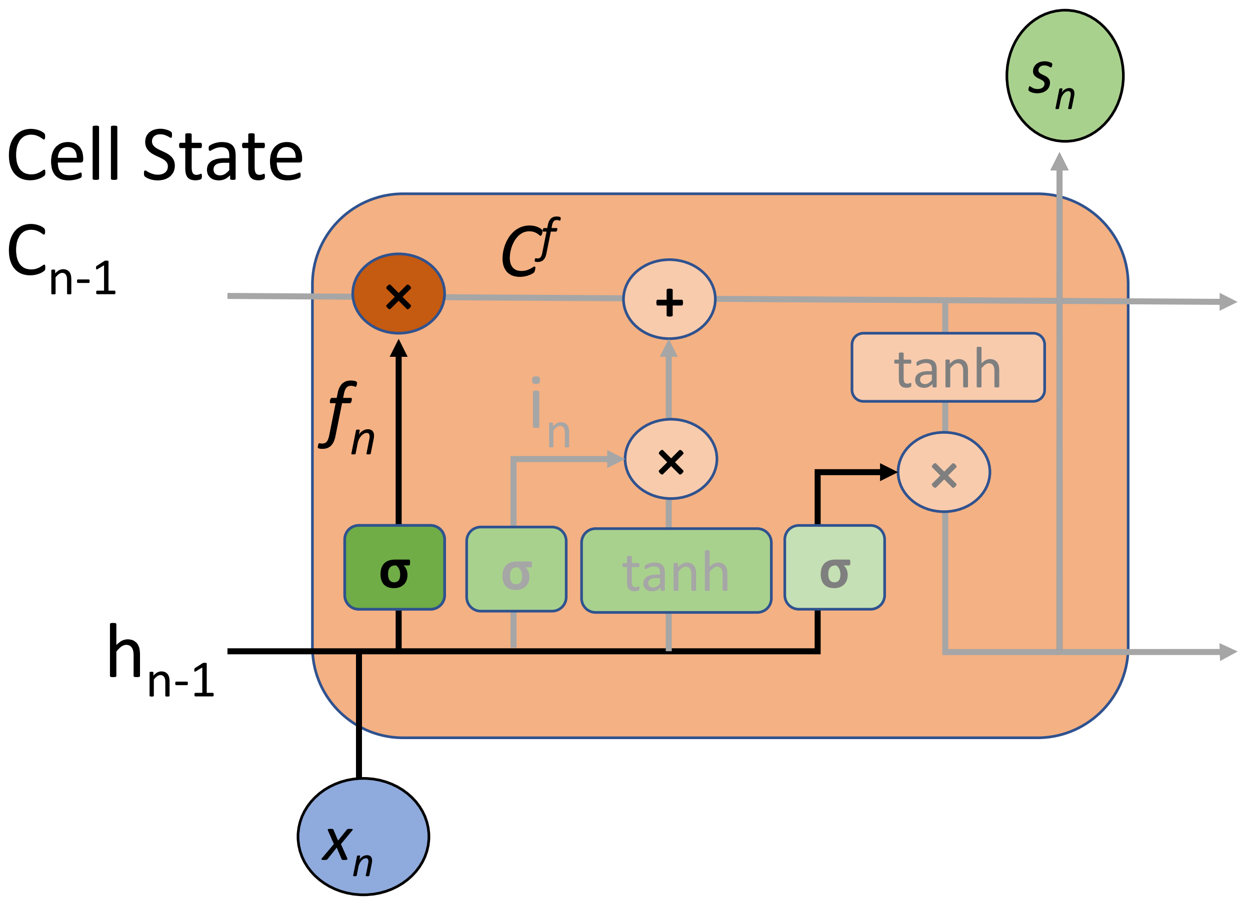
Long Short-Term Memory (LSTM)

How do LSTMs work?

a) Forget

b) Input

c) Output



Forget gate gets rid of irrelevant information

$$f_n = \sigma(W_{hf}h_{n-1} + W_{xf}x_n + b_f)$$
$$C^f = f_n * C_{n-1}$$

Output of Sigmoid Activation (σ):
0 => Forget,
1 => Keep

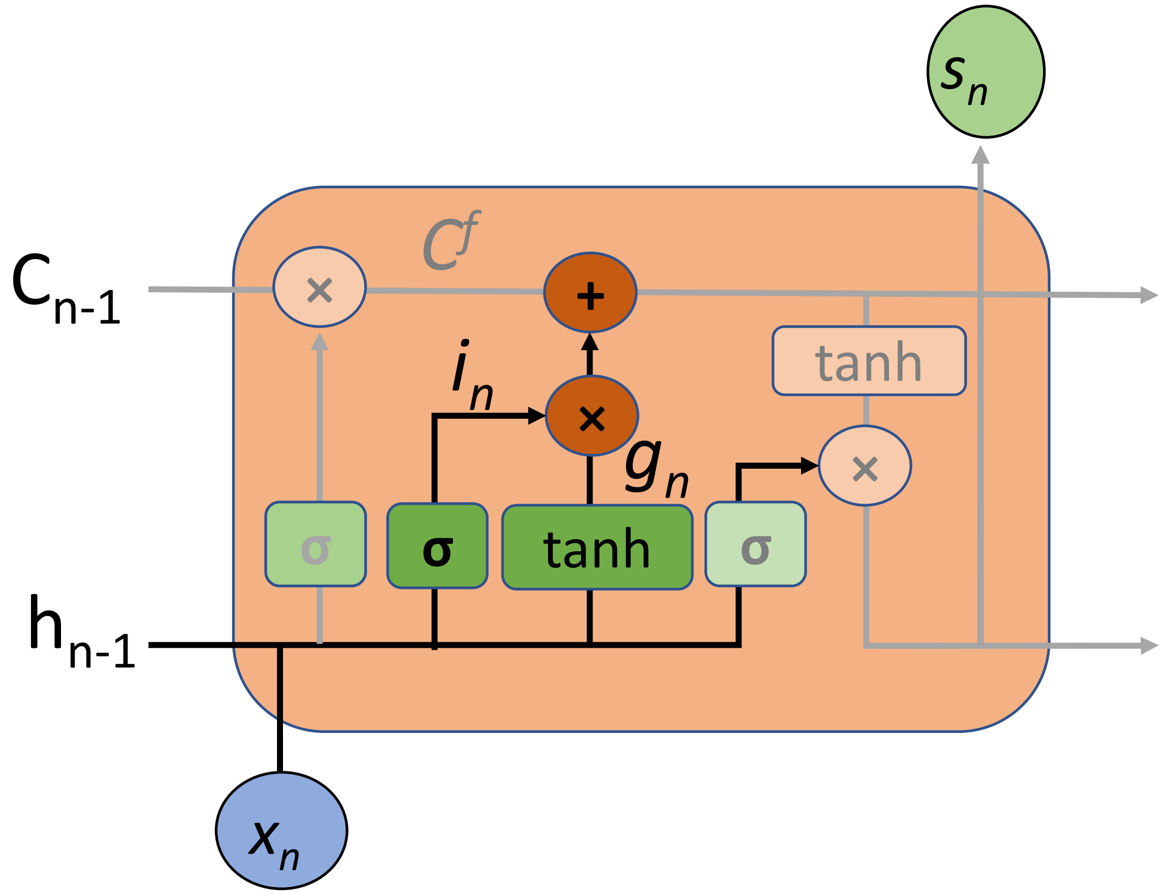
Long Short-Term Memory (LSTM)

How do LSTMs work?

a) Forget

b) Input

c) Output



Input gate stores relevant information from current input

Output of Sigmoid Activation (σ):
0 => Forget,
1 => Keep

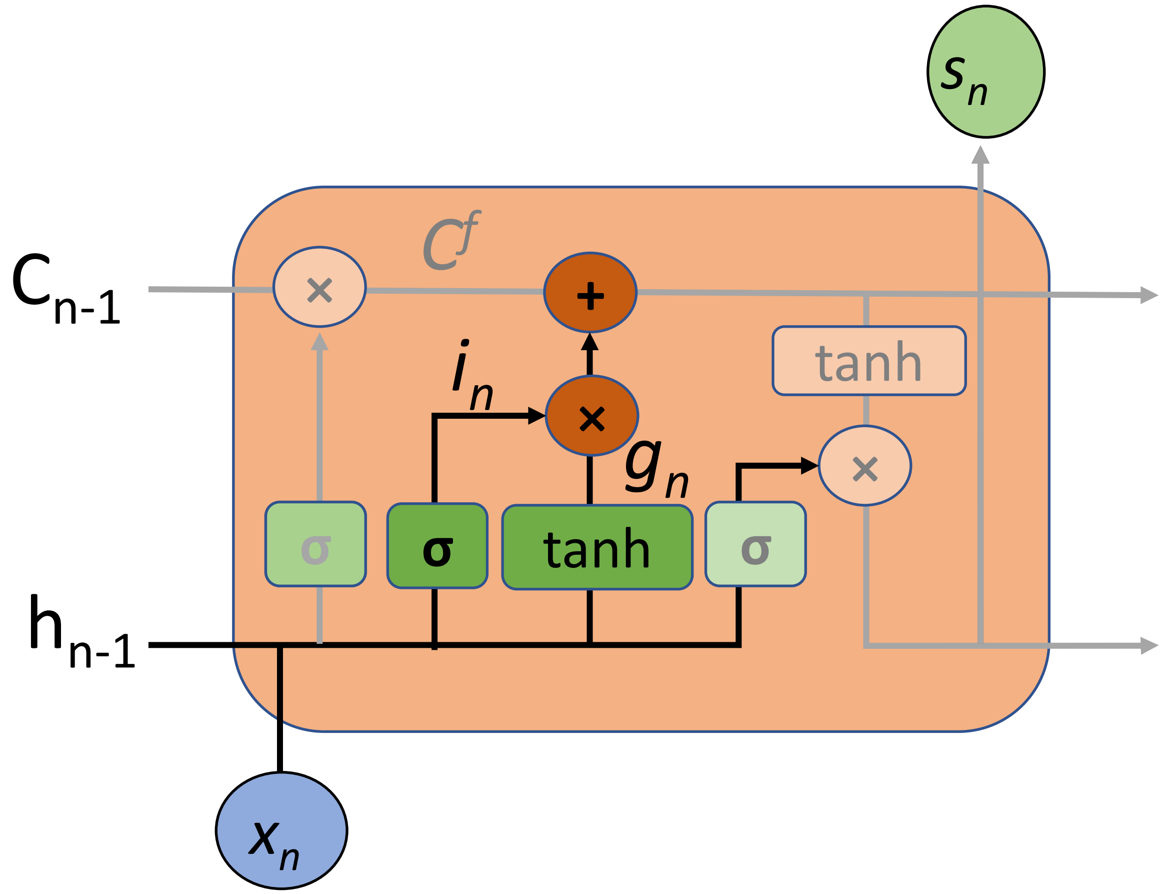
Long Short-Term Memory (LSTM)

How do LSTMs work?

a) Forget

b) Input

c) Output



Output of Sigmoid Activation (σ):
0 => Forget,
1 => Keep

Input gate stores relevant information from current input

$$i_n = \sigma(W_{hi}h_{n-1} + W_{xi}x_n + b_i)$$
$$g_n = tanh(W_{hg}h_{n-1} + W_{xg}x_n + b_g)$$

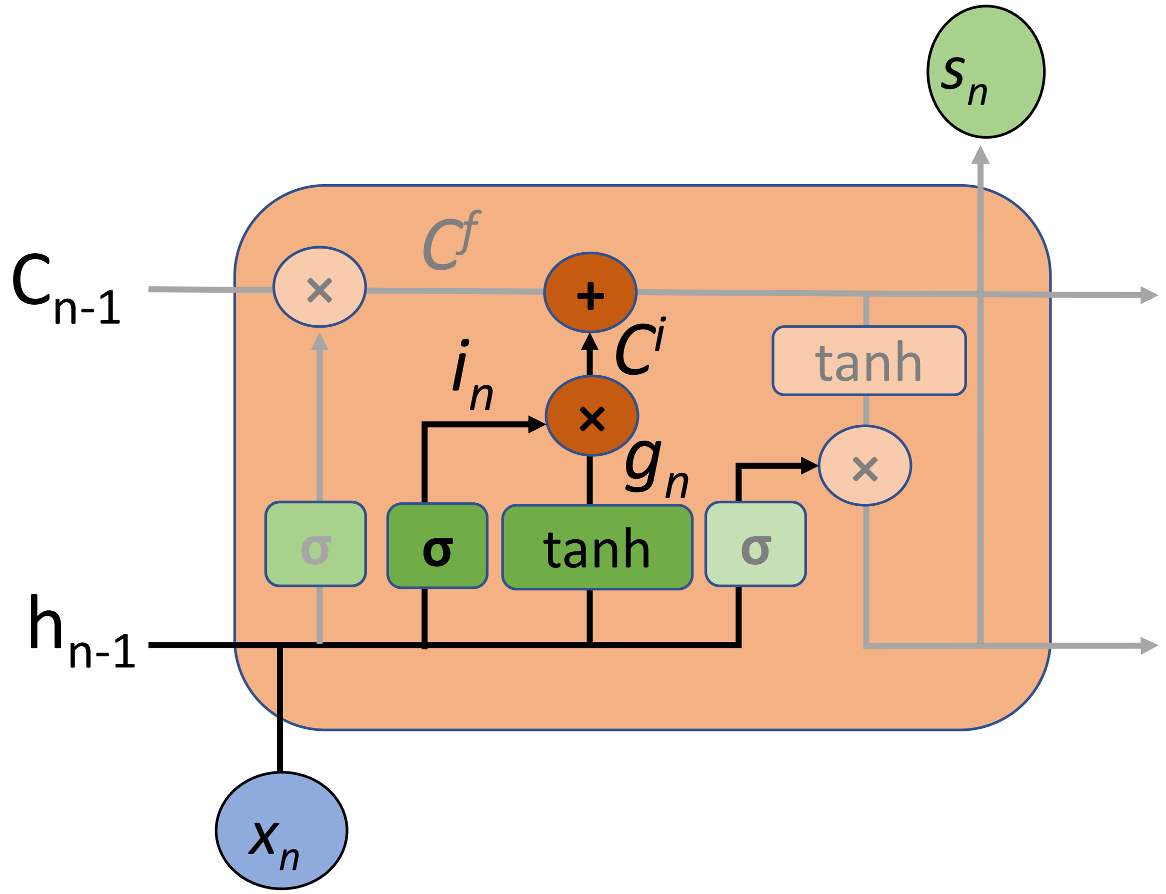
Long Short-Term Memory (LSTM)

How do LSTMs work?

a) Forget

b) Input

c) Output



Input gate stores relevant information from current input

$$C^i = (i_n * g_n)$$

Output of Sigmoid Activation (σ):
0 => Forget,
1 => Keep

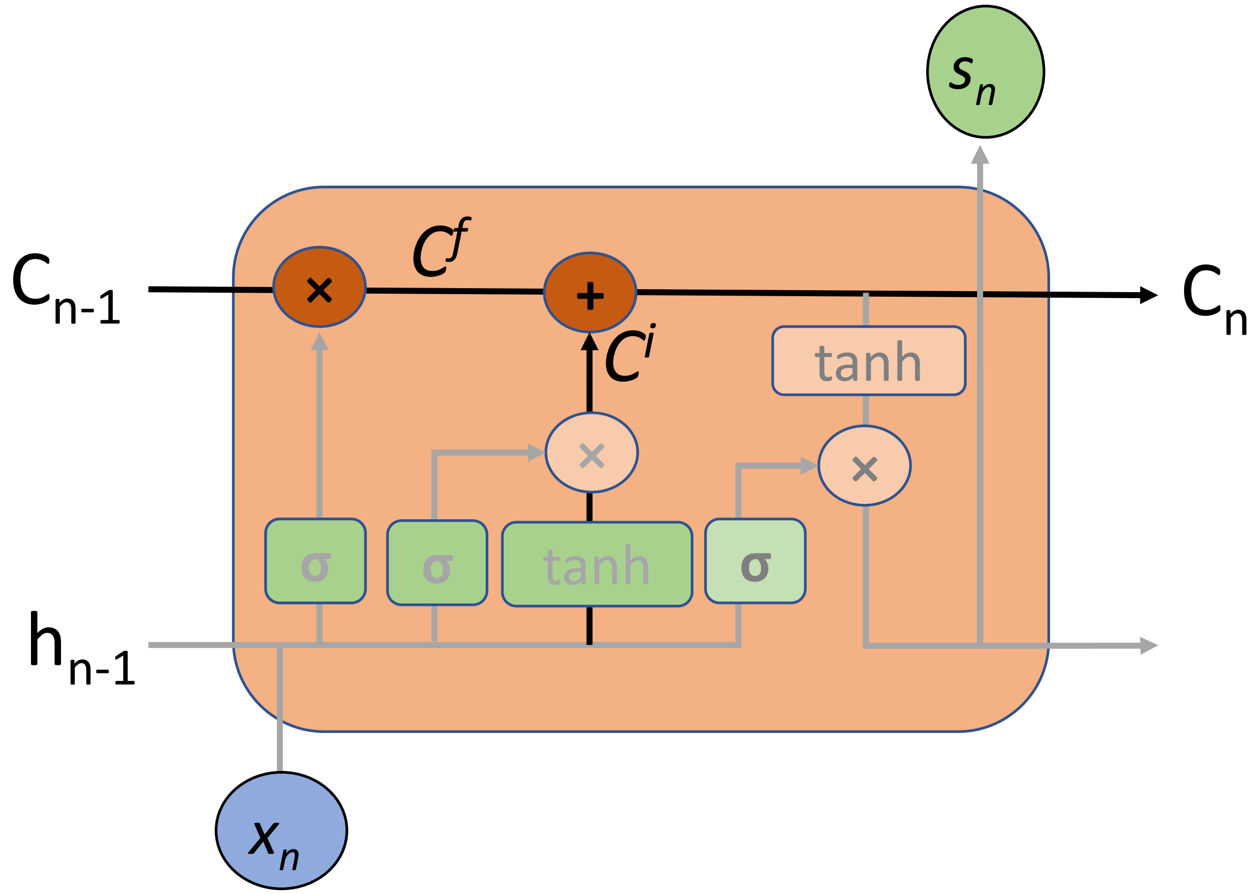
Long Short-Term Memory (LSTM)

How do LSTMs work?

a) Forget

b) Input

c) Output



Update gate selectively update cell state value

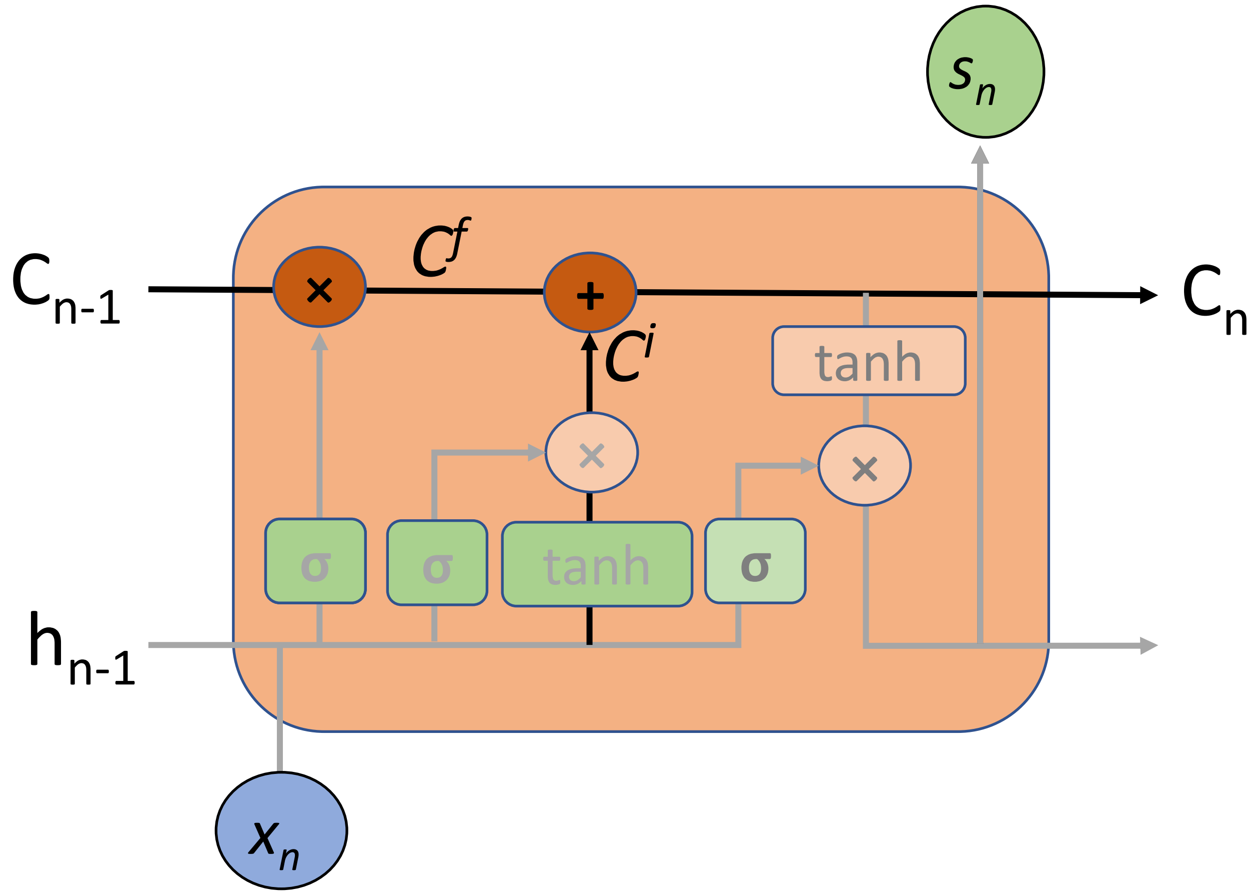
Long Short-Term Memory (LSTM)

How do LSTMs work?

a) Forget

b) Input

c) Output



Update gate selectively update cell state value

$$C_n = C^i + C^f$$

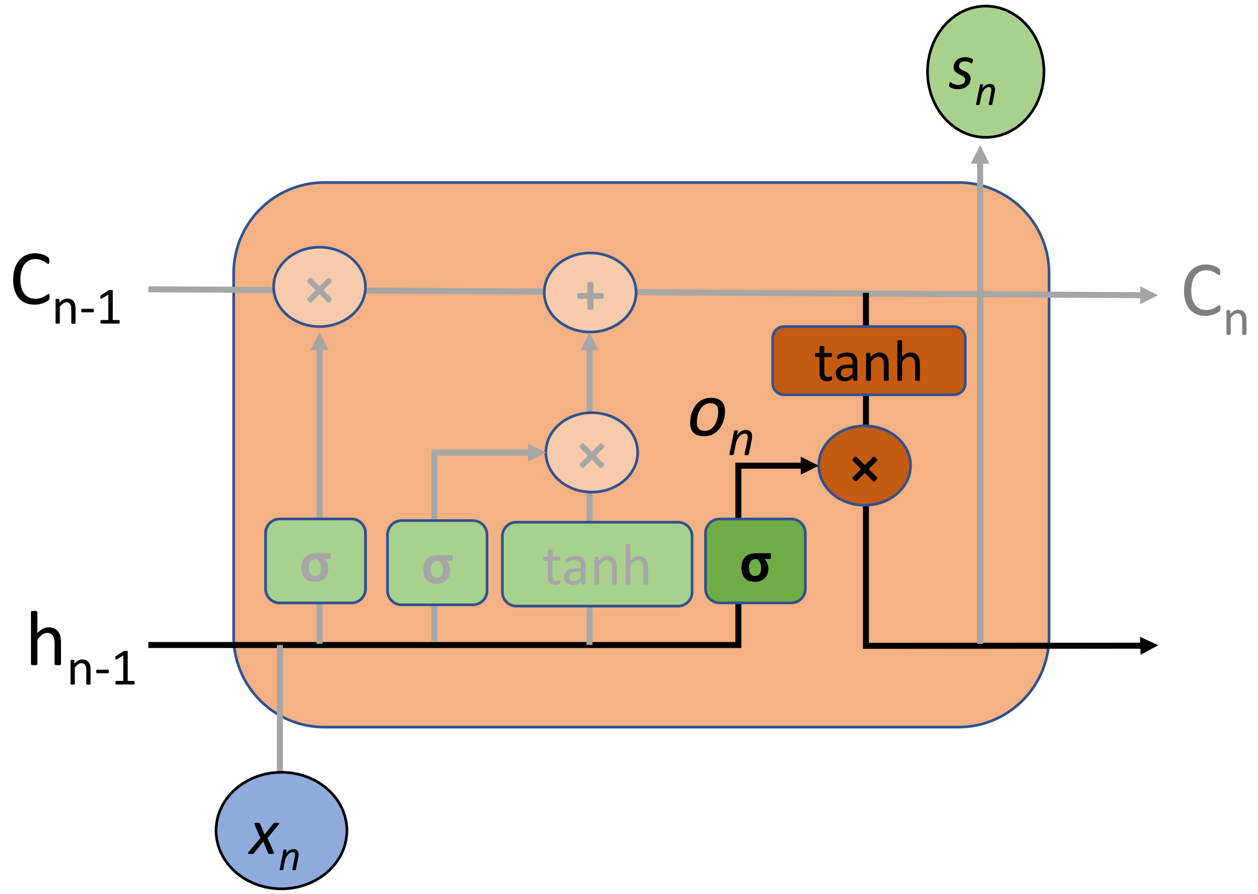
Long Short-Term Memory (LSTM)

How do LSTMs work?

a) Forget

b) Input

c) Output



Output gate returns a filtered version of the cell state

Long Short-Term Memory (LSTM)

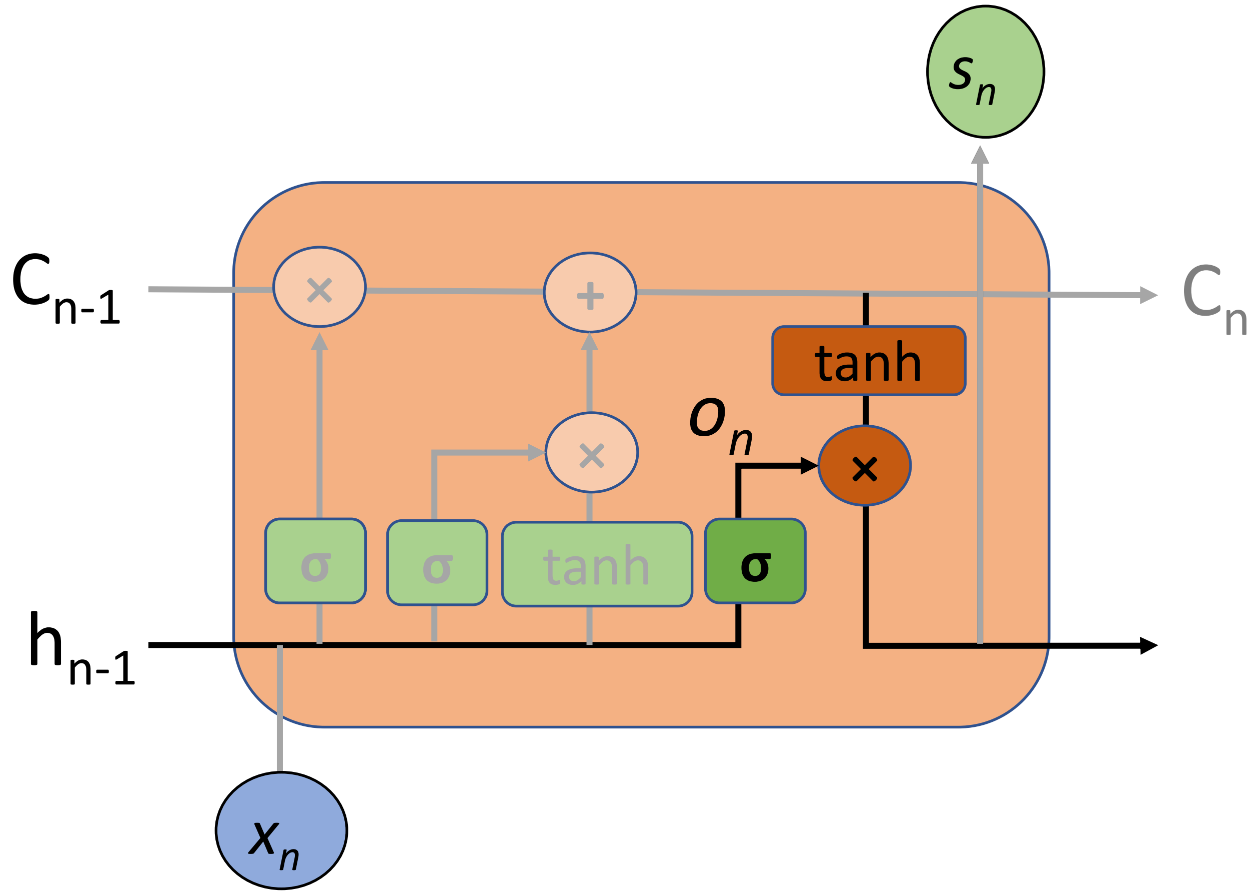
How do LSTMs work?

a) Forget

b) Input

c) Output

Output gate returns a filtered version of the cell state



$$o_n = \sigma(W_{ho}h_{n-1} + W_{xo}x_n + b_o)$$

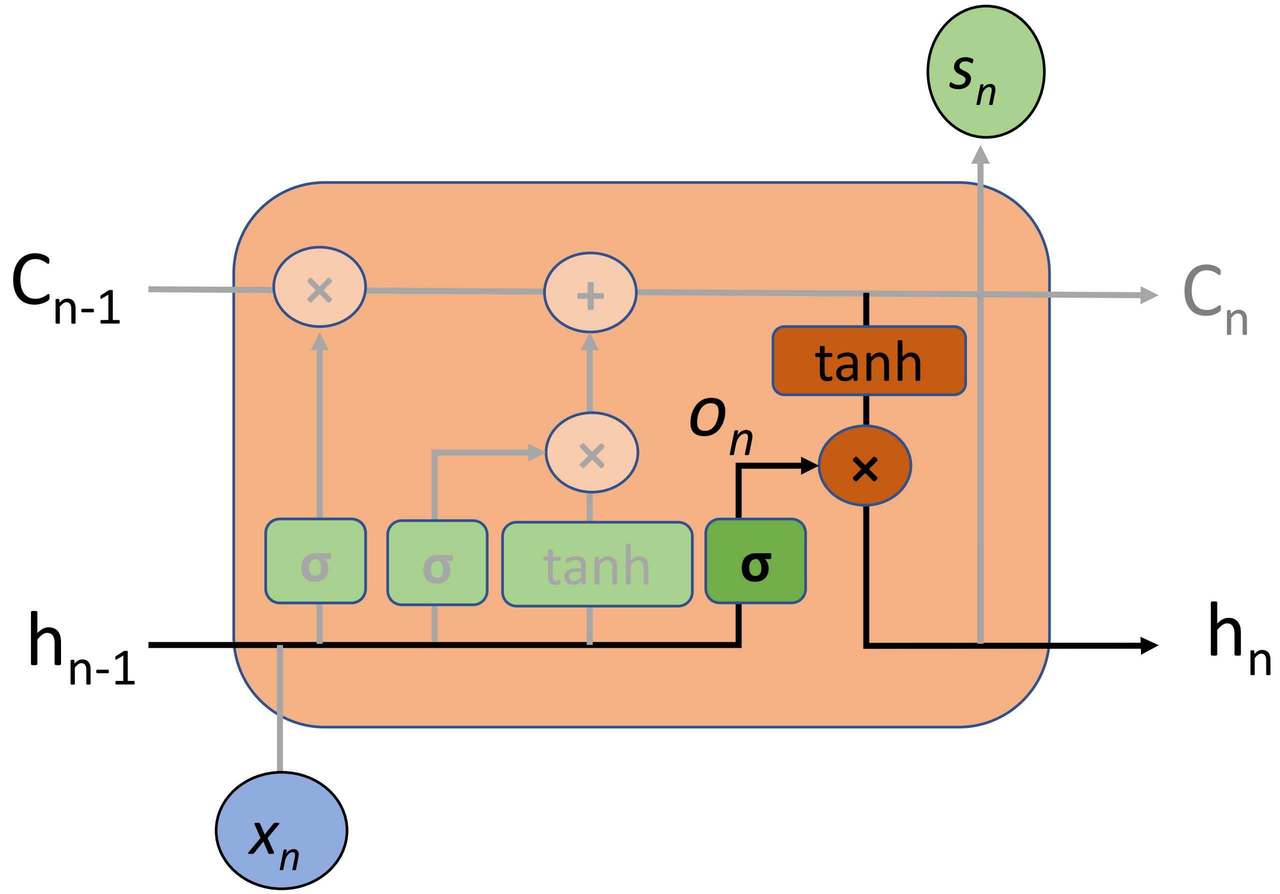
Long Short-Term Memory (LSTM)

How do LSTMs work?

a) Forget

b) Input

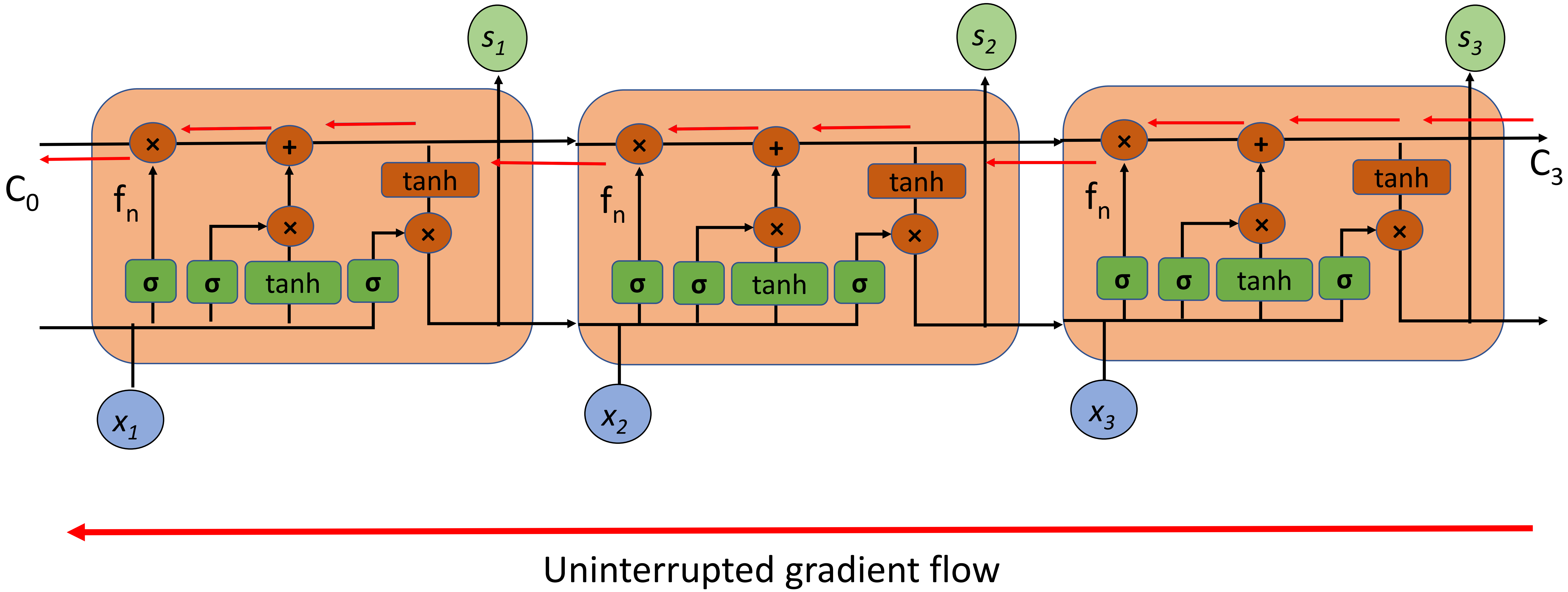
c) Output



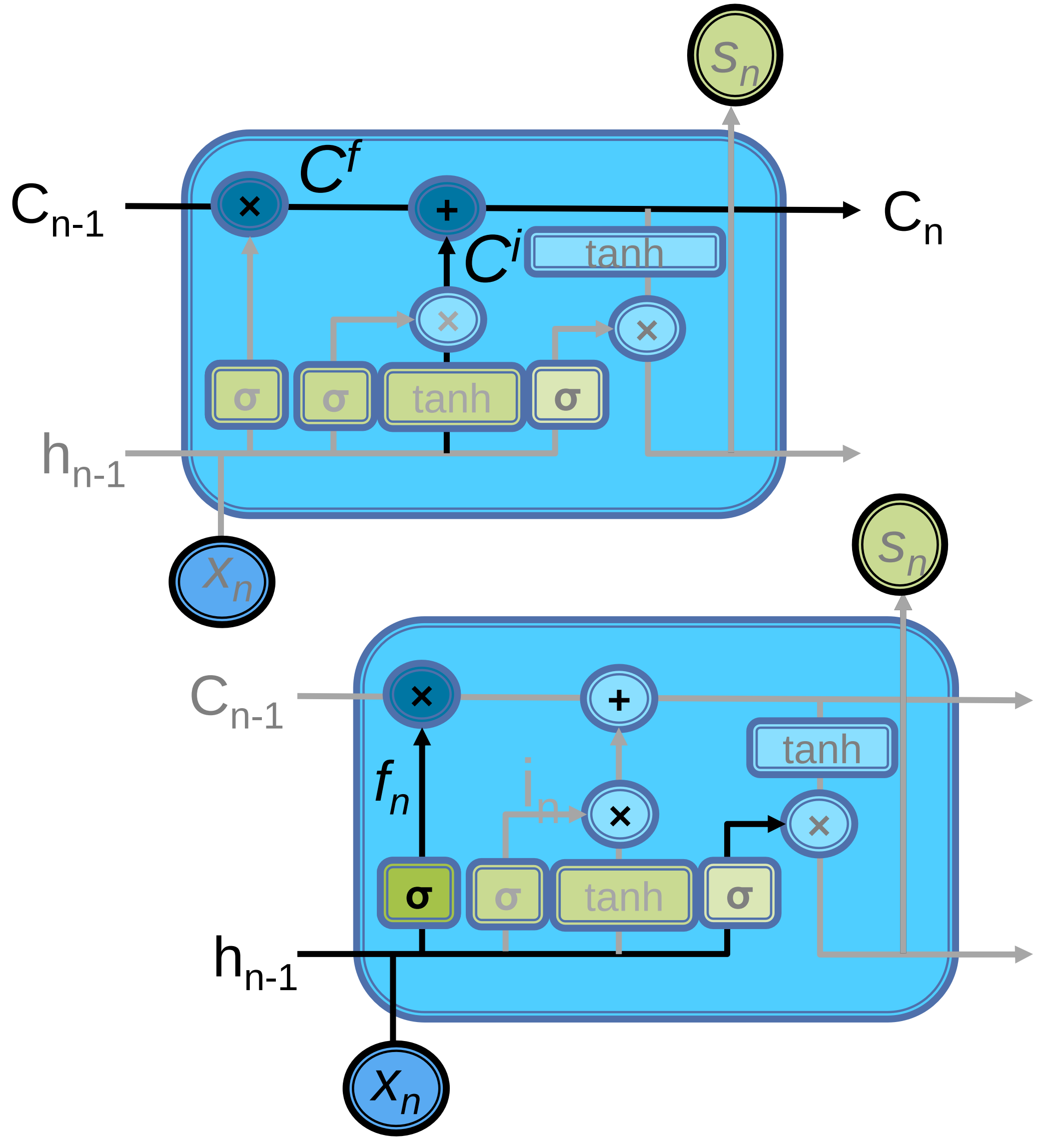
Output gate returns a filtered version of the cell state

$$h_n = o_n * \tanh(C_n)$$

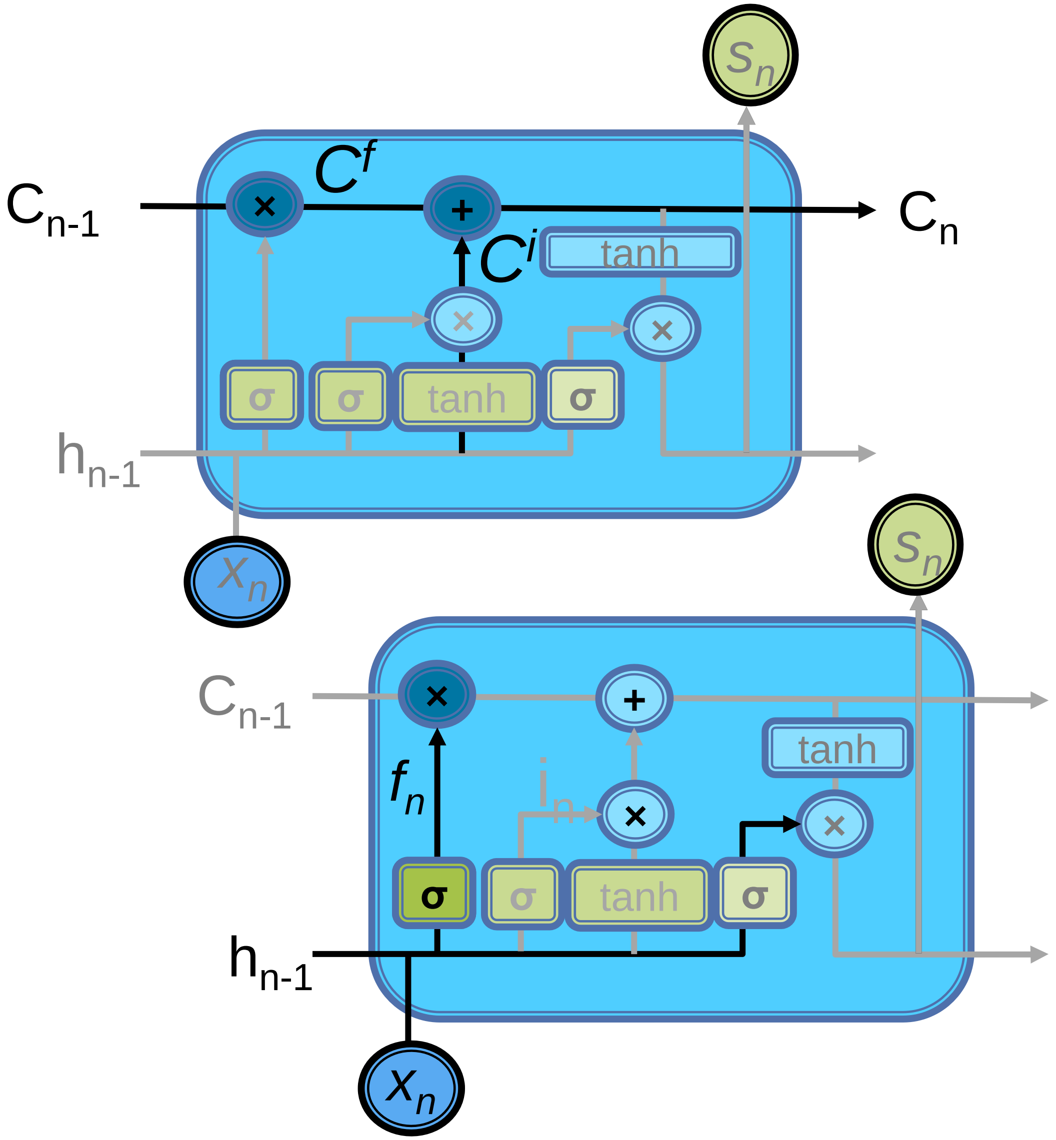
LSTM Gradient Flow



LSTM: How it solves the vanishing gradient problem?



LSTM: How it solves the vanishing gradient problem?



Ans: Using Forget gate

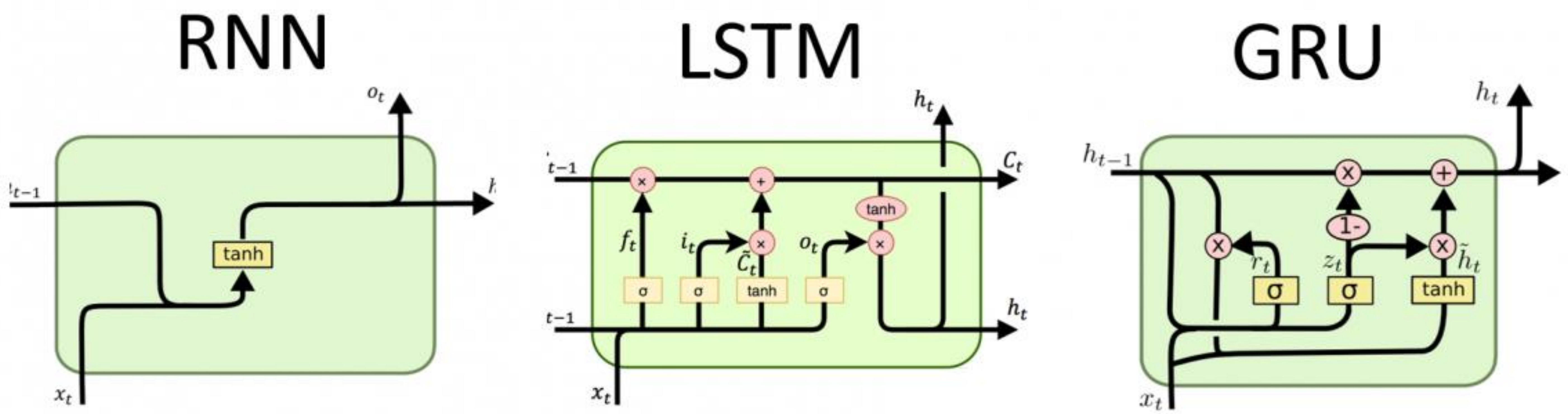
Gradient at C_n passed on to C_{n-1} is unaffected by any other operations, but the forget gate.

$$C^f = f_n * C_{n-1}$$

$$C_n = C^i + C^f$$

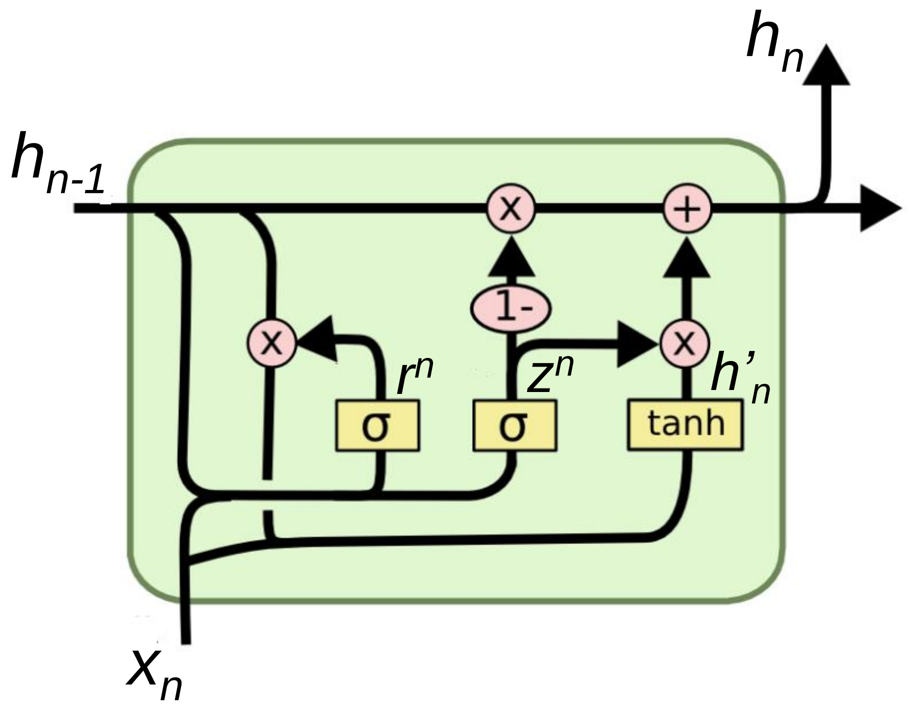
$$h_n = o_n * \tanh(C_n)$$

RNN, LSTM, GRU



Refer: http://dprogrammer.org/rnn-lstm-gru

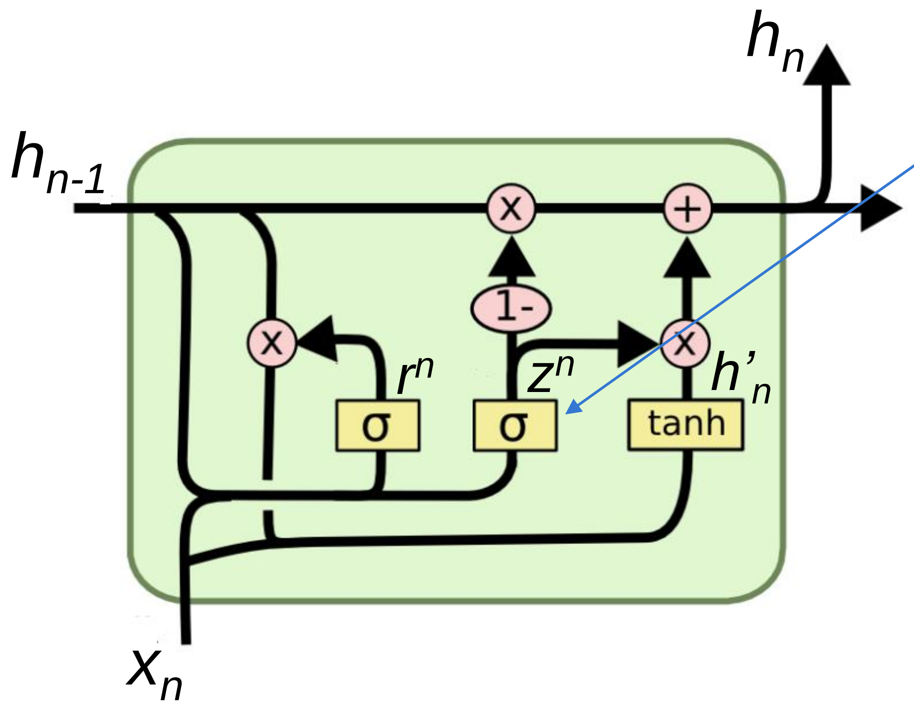
Gated Recurrent Unit (GRU)



- Proposed in 2014 as simpler alternative to LSTM
- Combines **forget** and **input** gates into a single **update** gate
- Merges cell state $\boldsymbol{C_n}$ and hidden state $\boldsymbol{h_n}$

Refer: http://dprogrammer.org/rnn-lstm-gru

Gated Recurrent Unit (GRU)



Update gate: controls what parts of hidden state are updated vs preserved

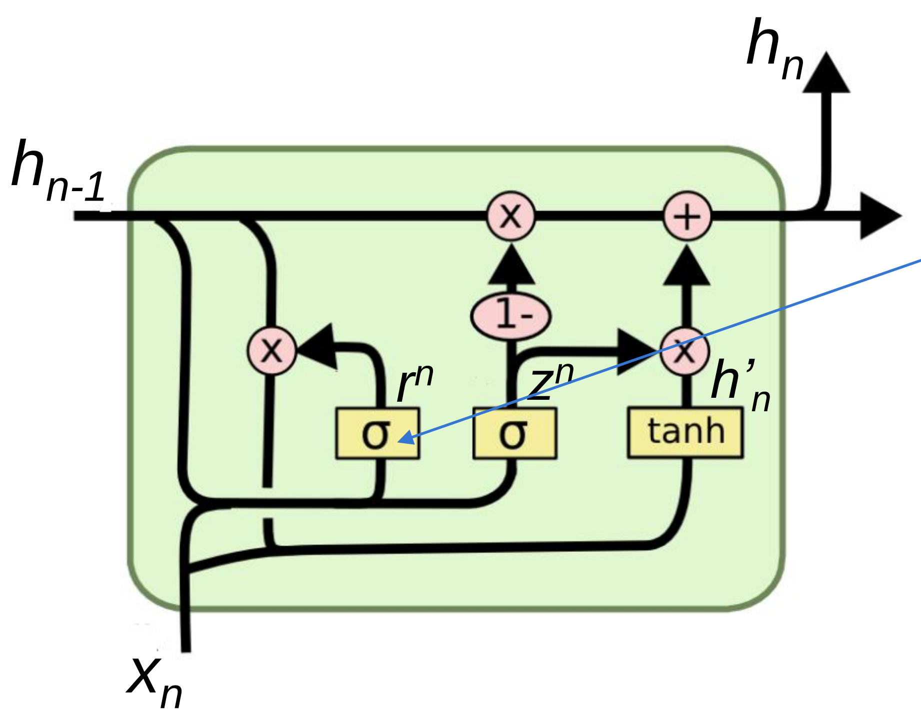
$$\boxed{z^n = \sigma(W_z * [h_{n-1}, x_n])}$$

Reset gate: controls what parts of previous hidden state are used to compute new content

$$\boxed{r^n = \sigma(W_r * [h_{n-1}, x_n])}$$

Refer: http://dprogrammer.org/rnn-lstm-gru

Gated Recurrent Unit (GRU)



Update gate: controls what parts of hidden state are updated vs preserved

$$\boxed{z^n = \sigma(W_z * [h_{n-1}, x_n])}$$

Reset gate: controls what parts of previous hidden state are used to compute new content

$$\boxed{r^n = \sigma(W_r * [h_{n-1}, x_n])}$$

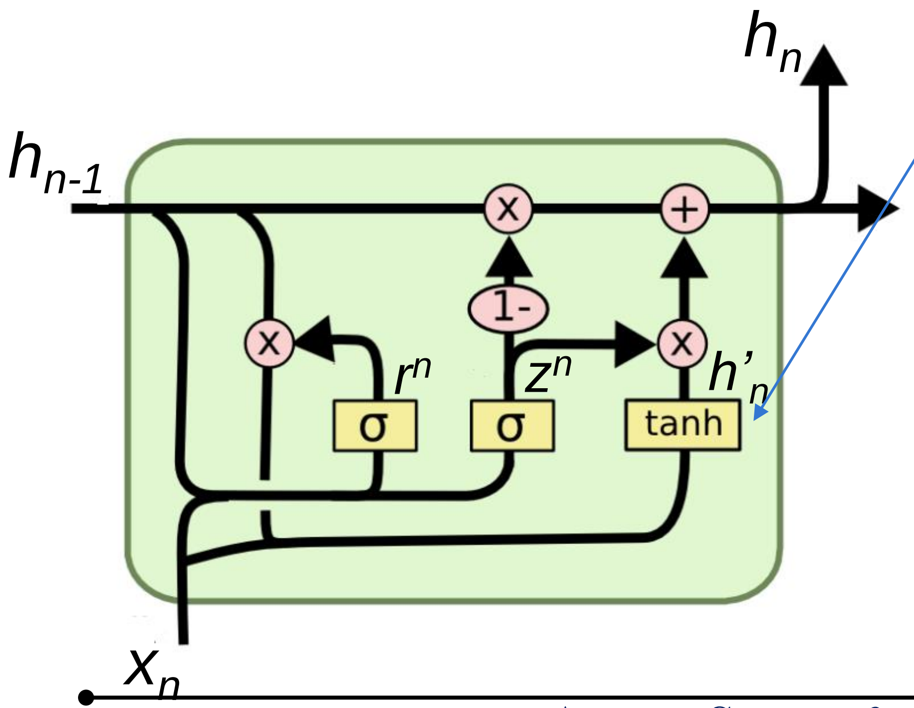
Refer: http://dprogrammer.org/rnn-lstm-gru

Gated Recurrent Unit (GRU)

$$\boxed{z^n = \sigma(W_z * [h_{n-1}, x_n])}$$

$$\boxed{r^n = \sigma(W_r * [h_{n-1}, x_n])}$$

New Hidden state content: selects useful parts of previous hidden state. Use this and current input to compute new hidden content $\boxed{h'_n = \tanh(W * [r^n * h_{n-1}, x_n])}$



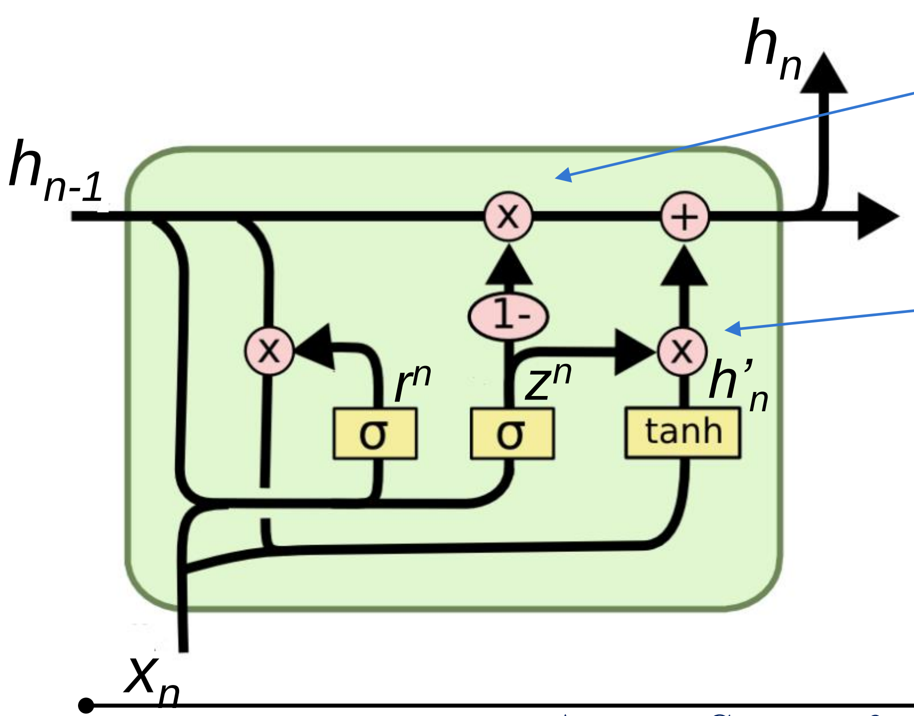
Refer: http://dprogrammer.org/rnn-lstm-gru

Gated Recurrent Unit (GRU)

$$z^n = \sigma(W_z * [h_{n-1}, x_n])$$

$$r^n = \sigma(W_r * [h_{n-1}, x_n])$$

New Hidden state content: selects useful parts of previous hidden state. Use this and current input to compute new hidden content $h'_n = \tanh(W * [r^n * h_{n-1}, x_n])$



$$h_n = (1 - z^n) * h_{n-1} + z^n * h'_n$$

Hidden state: simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

Refer: http://dprogrammer.org/rnn-lstm-gru

LSTM vs GRU

- Input and forget gates of LSTMs are coupled by an update gate in GRUs; reset gate in GRUs is applied directly to previous hidden state
- GRU has two gates, an LSTM has three gates. Lesser parameters to learn!
- In GRUs:
  - No internal memory (c_n) different from exposed hidden state
  - No output gate as in LSTMs
- LSTM is a good default choice (especially if data has long-range dependencies, or if training data is large)
- Switch to GRUs for speech and fewer parameters

References

1. Machine Learning: A Probabilistic Perspective" by Kevin Murphy, published by MIT Press, 2012.
2. "Pattern Recognition and Machine Learning" by Christopher M. Bishop, published by Springer, 2006.
3. "Python Machine Learning" by Sebastian Raschka and Vahid Mirjalili, published by Packt Publishing, 2015.
4. "Machine Learning Yearning" by Andrew Ng, published by Goodfellow Publishers, 2018.
5. "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow" by Aurélien Géron, published by O'Reilly Media, 2019.
6. "Applied Predictive Modeling" by Max Kuhn and Kjell Johnson, published by Springer, 2013.
7. "Reinforcement Learning: An Introduction" by Richard S. Sutton and Andrew G. Barto, published by MIT Press in 2018.

Thank You !

Any Questions