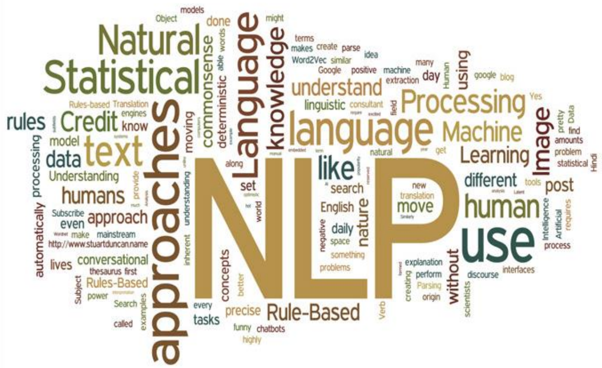


به نام خدا

بخش دوازدهم

شبکه های عصبی بازگشتی برای مدلسازی زبانی

(RNNs for Language Modeling)

گروه هوش مصنوعی، دانشکده مهندسی کامپیوتر

حمیدرضا برادران کاشانی

Limitations of traditional LMs

- ✓ Using language models in different applications, such as:
  - ✓ Speech recognition
  - ✓ Machine translation
  - ✓ Abstractive summarization
  - ✓ …

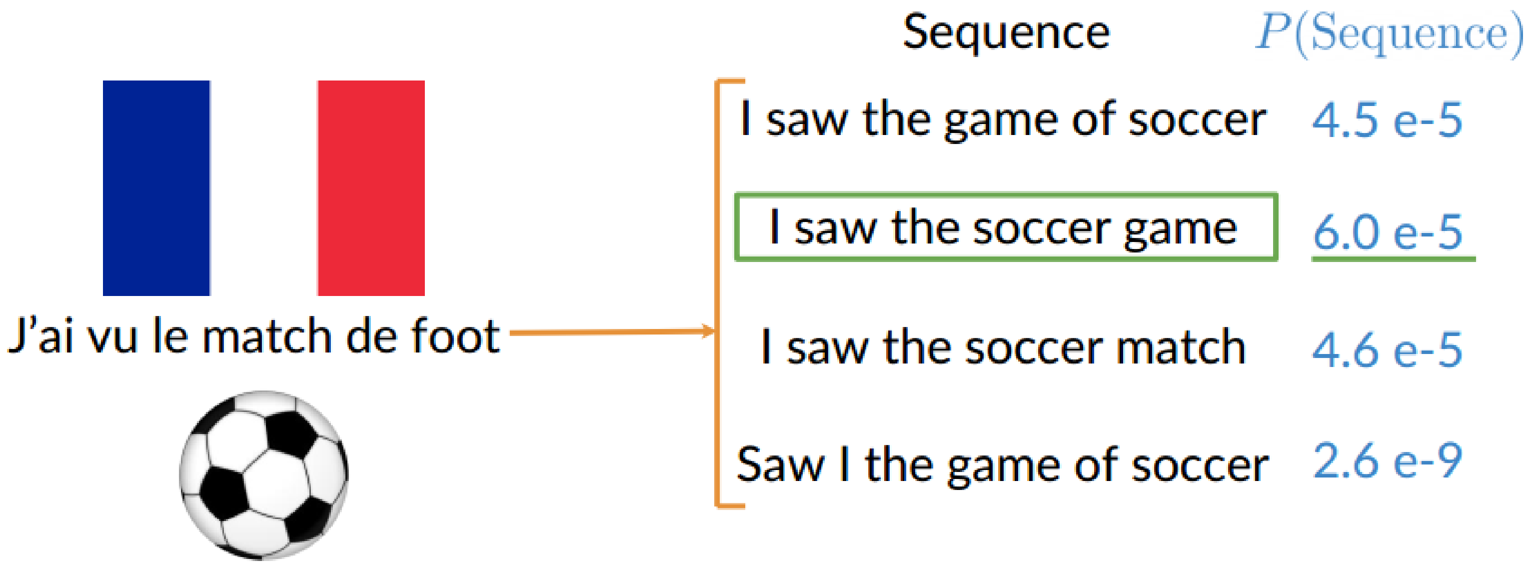


Figure from deeplearning.ai

Limitations of traditional LMs

- To build an N-gram language model with traditional algorithms, such as Markov-based language models, e.g. bigram LM:
  - ✓ We have to compute conditional probabilities for bigrams

$$P(w_2|w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)} \longrightarrow \text{Bigrams}$$

$$P(w_3|w_1, w_2) = \frac{\text{count}(w_1, w_2, w_3)}{\text{count}(w_1, w_2)} \longrightarrow \text{Trigrams}$$

$$\boxed{P(w_1, w_2, w_3) = P(w_1) \times P(w_2|w_1) \times P(w_3|w_2)}$$

- **A main problem with Markov-based LMs**
  - ✓ Large N-grams to capture dependencies between distant words (need a large corpora to estimate conditional probabilities)
  - ✓ Need a lot of space and RAM to store the probabilities of all possible combinations

Advantages of RNNs

- Using traditional (trigram) LM to complete this sentence
  - Ali was supposed to study with me. I called him, but he did not --------- .

want
respond
choose
want
have
ask
attempt
answer
know

→ Similar probabilities with trigram

But "have" does not make any sense

Advantages of RNNs

- A better alternative to fill the blank: **RNNs**

  - Ali was supposed to study with me. I called him, but he did not --------- .
    Answer

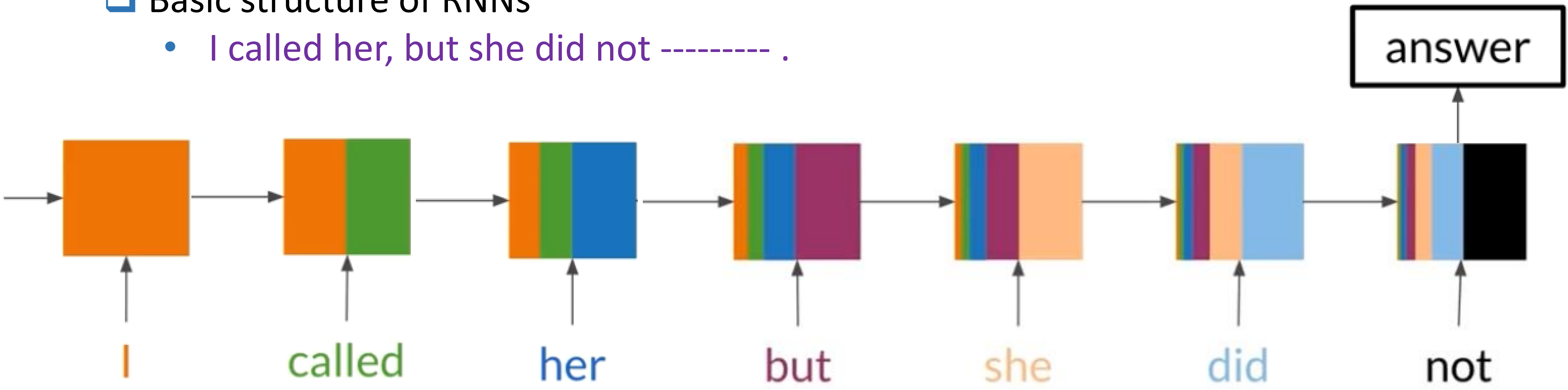
✓ RNNs looks at every previous word (are not limited to look at just the previous N-1 words).

✓ RNNs propagate information from the beginning of the sentence to the end.

✓ Gets a better prediction for the blank using RNNs: **Answer**

The Basic Structure of RNN

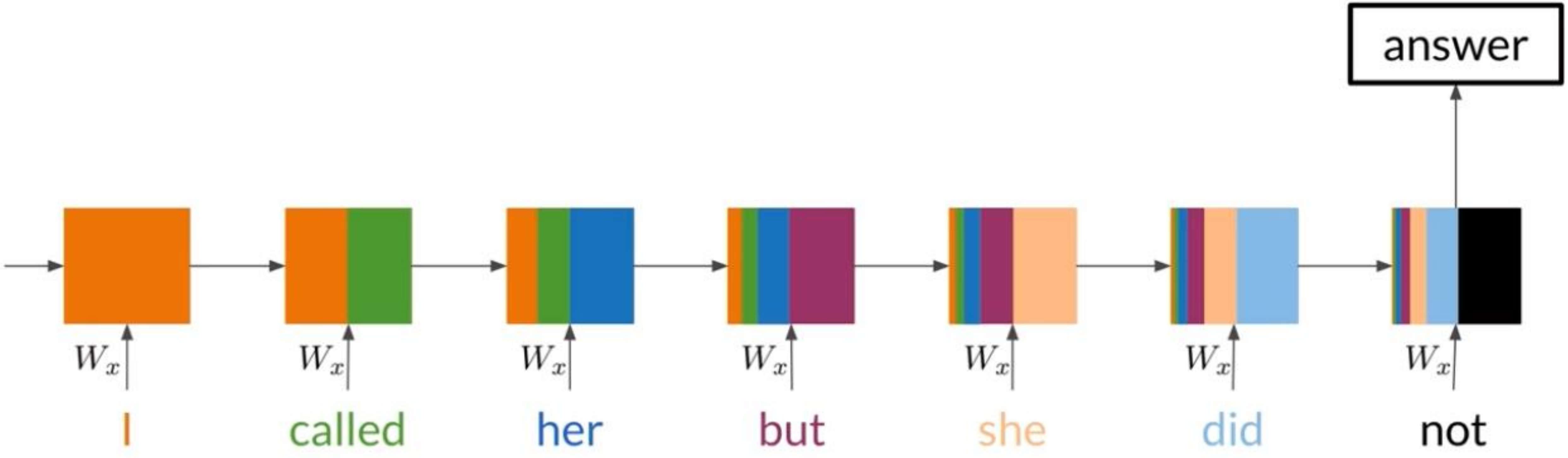
- Basic structure of RNNs
  - I called her, but she did not --------- .



✓ Each of the boxes in this diagram represents the computations made at each step.
✓ The colors represent the information that is used for every computation.
✓ The computations made at the last step have information from all the words in this sentence.

Figure from deeplearning.ai

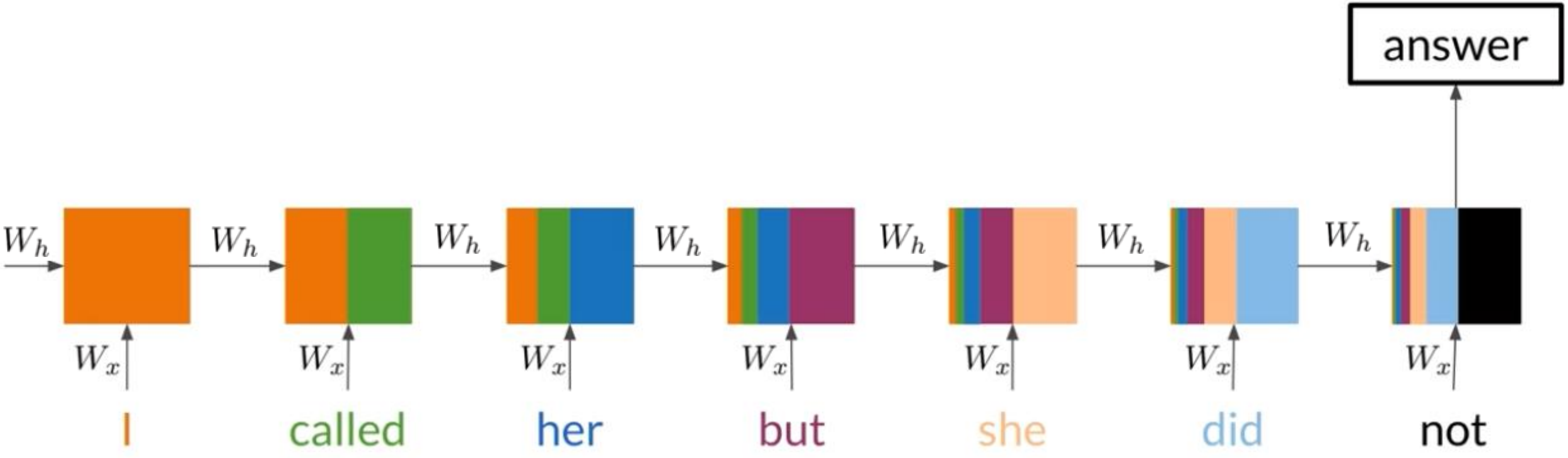
The Basic Structure of RNN



✓ The information from every word in the sequence is multiplied by the **same weight**, **Wx**.

Figure from deeplearning.ai

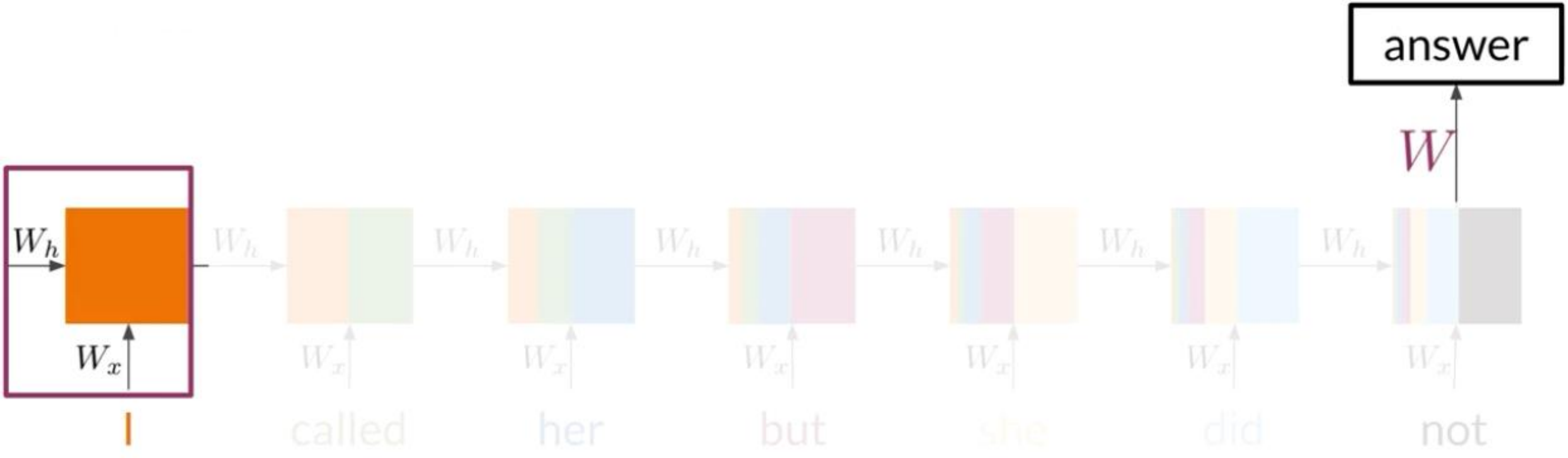
The Basic Structure of RNN



- ✓ The information from every word in the sequence is multiplied by the **same weights**, **Wx**.
- ✓ The information propagated from the beginning to the end is multiplied by **Wh**.

Figure from deeplearning.ai

The Basic Structure of RNN



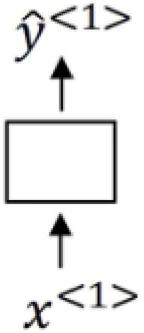
- ✓ **This purple block** is repeated for every word in the sequence.
- ✓ The only learnable parameters are the ones in Wx, Wh, and W.

Figure from deeplearning.ai

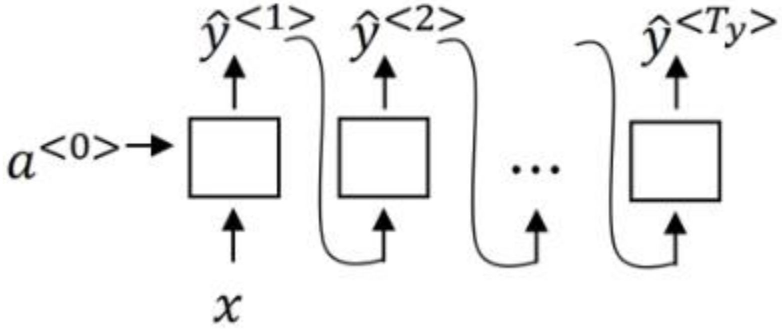
❑ Summary

- ✓ RNNs model relationships among distant words.
- ✓ RNNs are capable of capturing dependencies and remembers a previous word although it is at the beginning of a sentence or paragraph.
- ✓ In RNNs, a lot of computations share parameters.

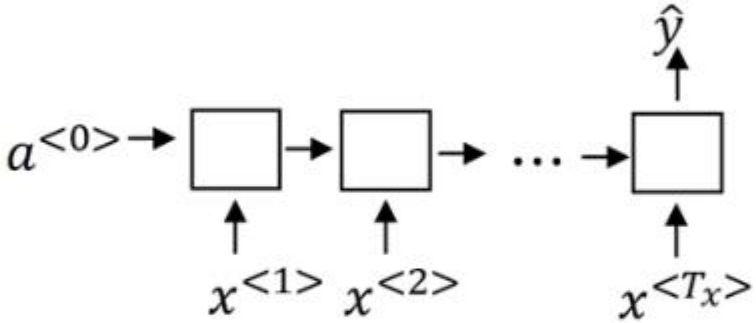
Different Types of RNNs



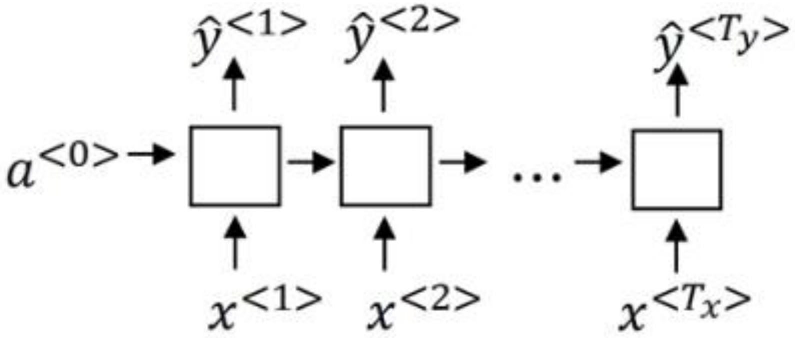
One to one



One to many

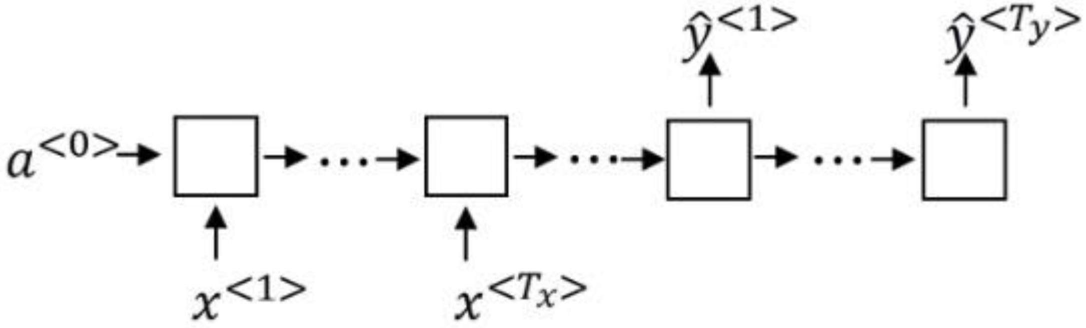


Many to one



Many to many

$T_x = T_y$



Many to many

$T_x \neq T_y$

Different Types of RNNs

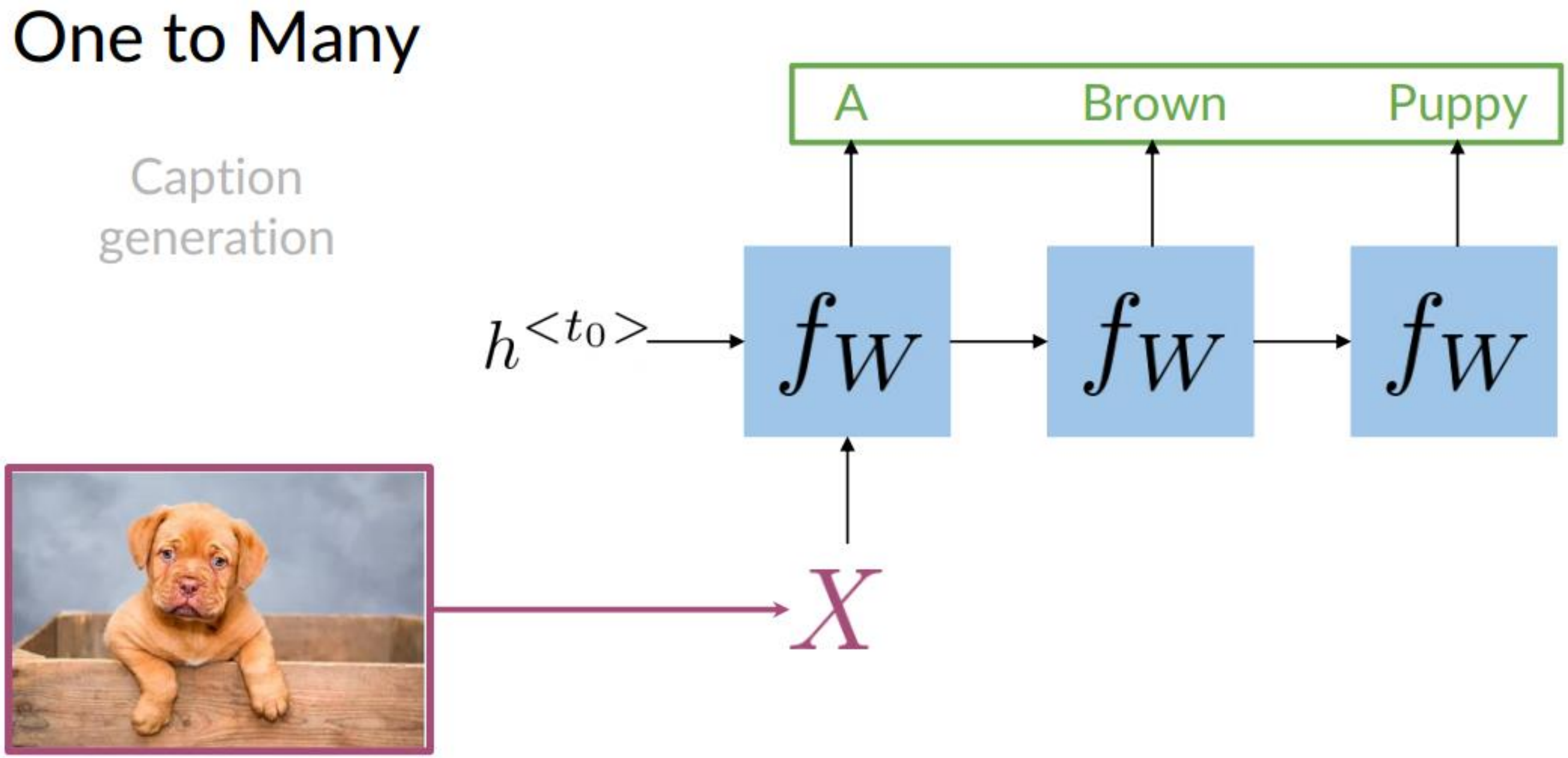


Figure from deeplearning.ai

Different Types of RNNs

Many to One

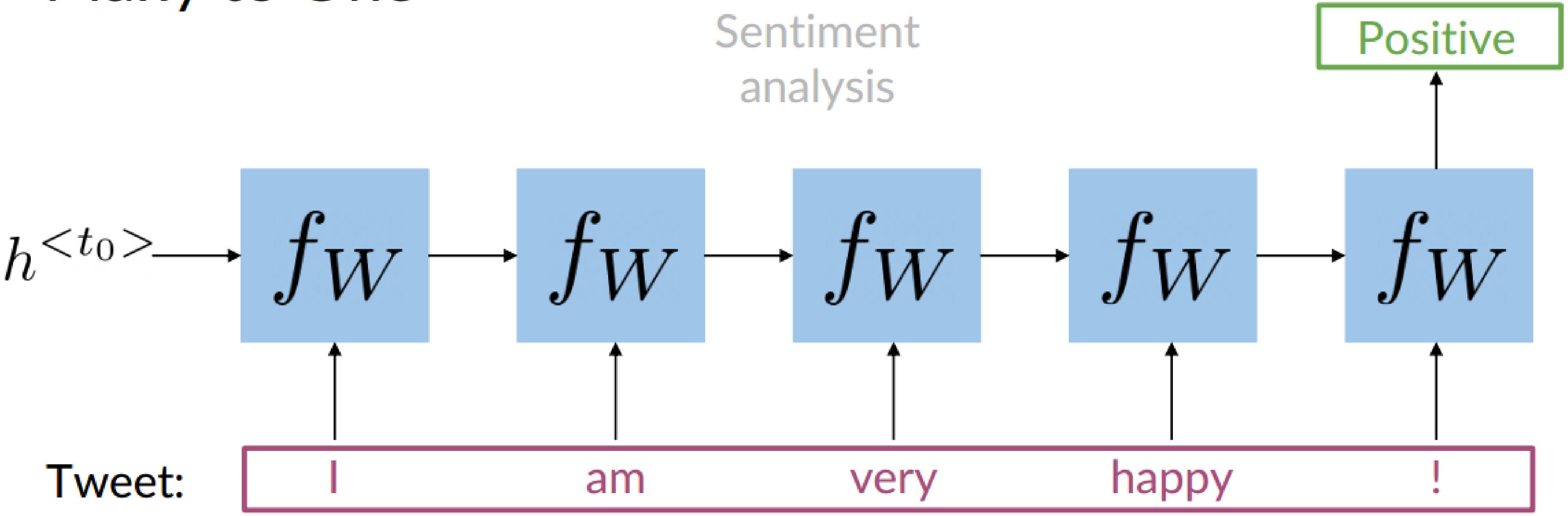


Figure from deeplearning.ai

Different Types of RNNs

Many to Many

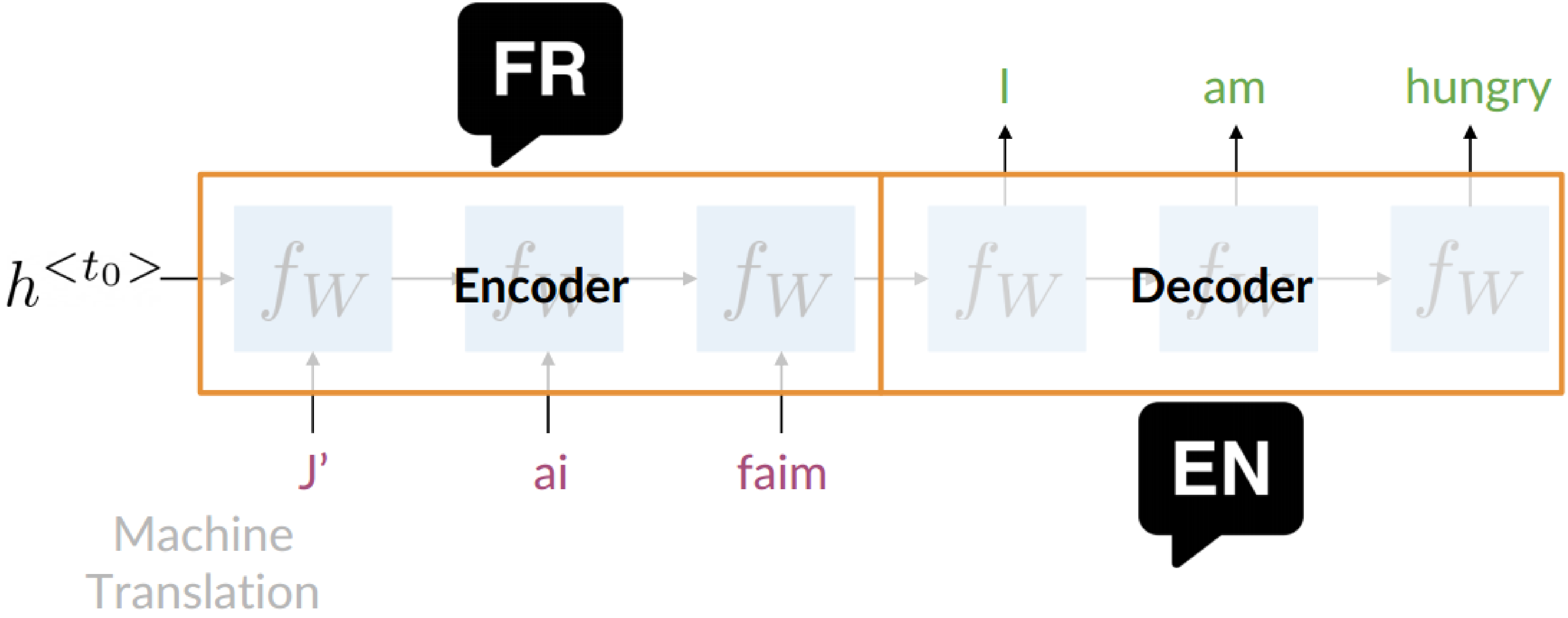
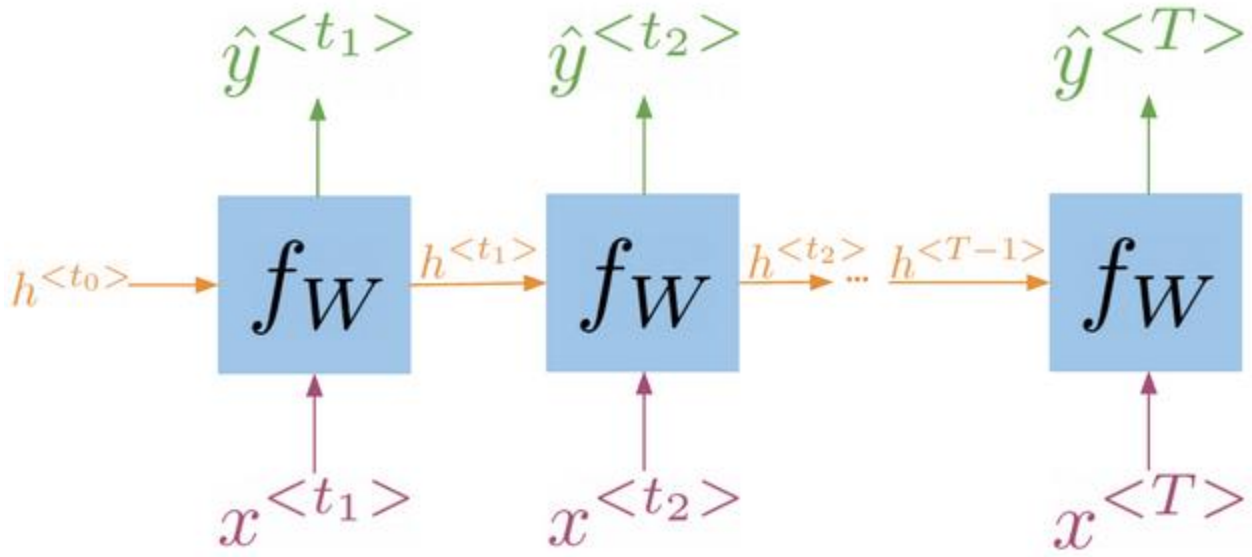


Figure from deeplearning.ai

Math in Simple RNNs

- **Outline**
  - ✓ **How RNNs propagate information (Through time!)**
  - ✓ **How RNNs make predictions**

Math in Simple RNNs



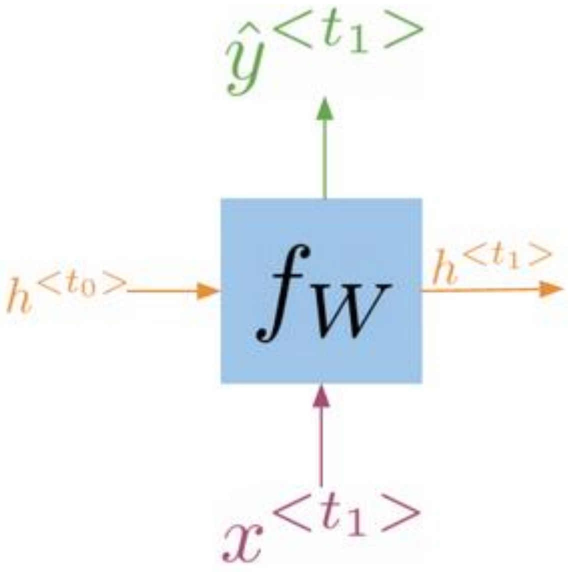
$$h^{<t>} = g(\boxed{W_h[h^{<t-1>}, x^{<t>}]} \boxed{+ b_h})$$

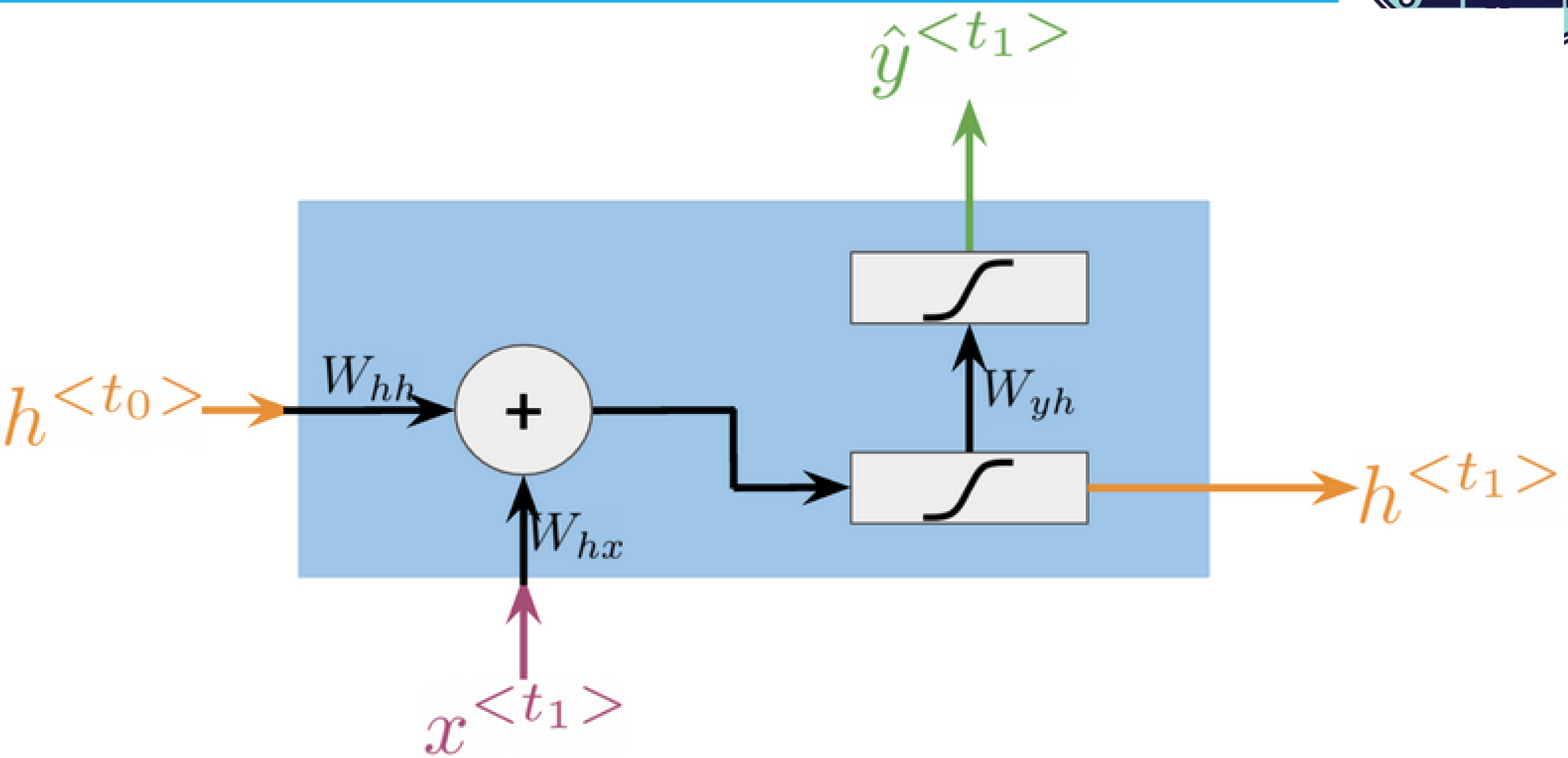
$$\boxed{h^{<t>} = g(W_{hh}h^{<t-1>} \oplus W_{hx}x^{<t>} + b_h)}$$

$$\hat{y}^{<t>} = g(W_{yh}h^{<t>} + b_y)$$

Figure from deeplearning.ai

Math in Simple RNNs





$$\boxed{h^{<t>} = g(W_{hh}h^{<t-1>} \oplus W_{hx}x^{<t>} + b_h)}$$

$$\hat{y}^{<t>} = g(W_{yh}h^{<t>} + b_y)$$

Figure from deeplearning.ai

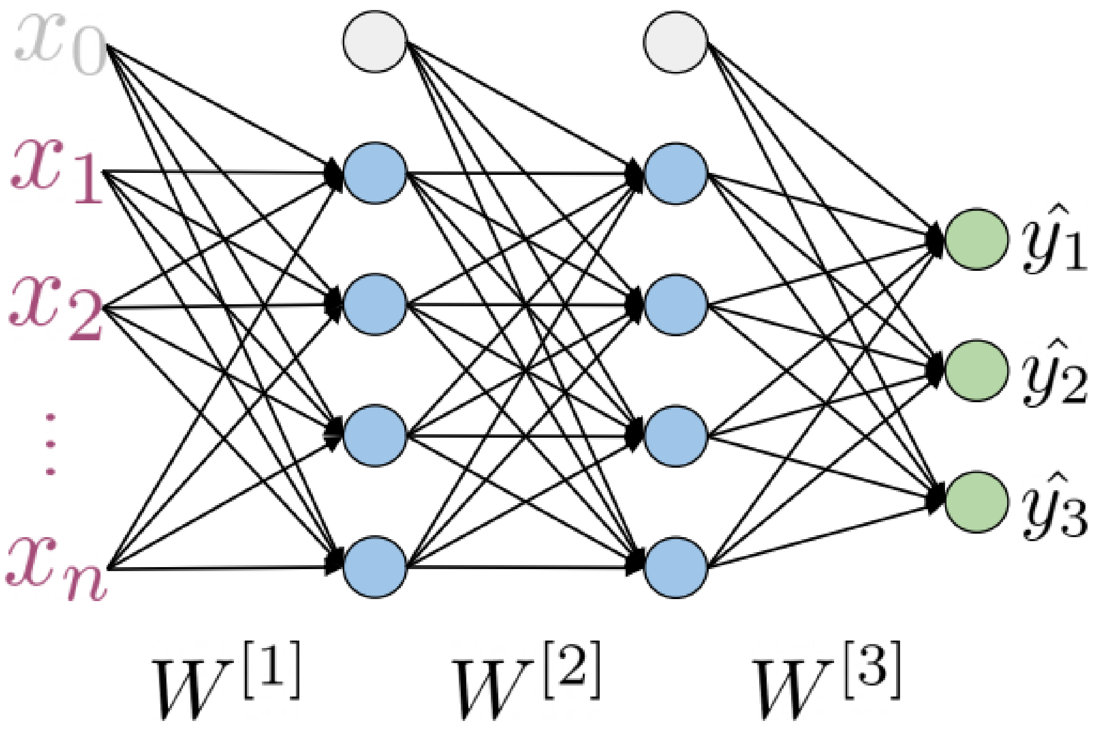
Math in Simple RNNs

❑ Summary

- ✓ Hidden states propagate information through time.
- ✓ Basic recurrent units have two inputs at each time: $h^{<t-1>}$ $x,^{<t>}$

Cost Function for RNNs

❑ Cross Entropy Loss



K - classes or possibilities

Either 0 or 1

$$J = -\sum_{j=1}^{\boxed{K}} \boxed{y_j} \log \hat{y}_j$$

Looking at a single example (x, y)

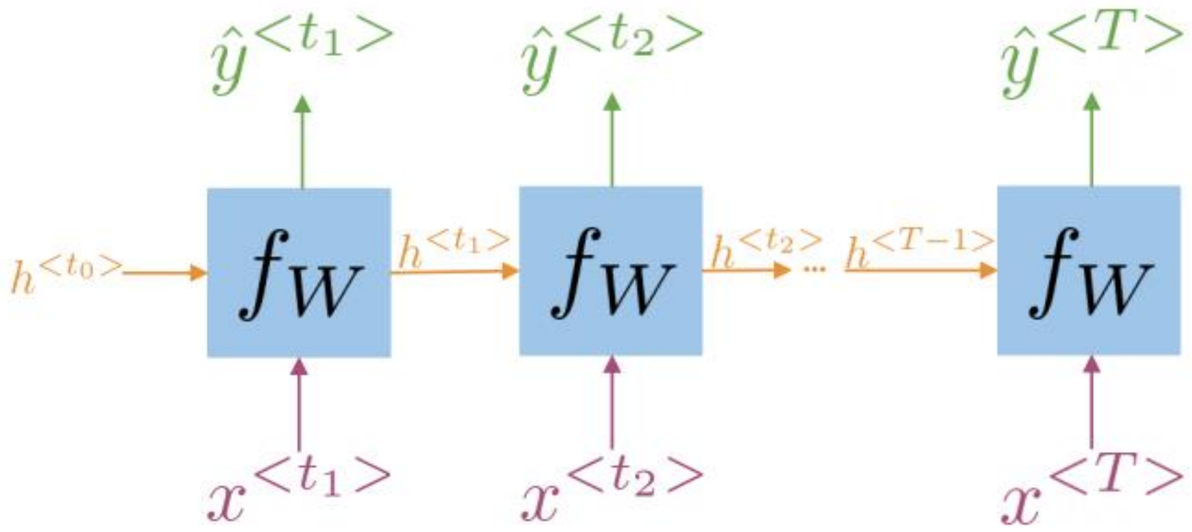
Figure from deeplearning.ai

Cost Function for RNNs

- **Cross Entropy Loss**

$$h^{<t>} = g(W_h[h^{<t-1>}, x^{<t>}] + b_h)$$

$$\hat{y}^{<t>} = g(W_{yh} h^{<t>} + b_y)$$



$$J = -\frac{1}{T} \sum_{t=1}^{T} \sum_{j=1}^{K} y_j^{<t>} \log \hat{y}_j^{<t>}$$

Average with respect to time

For RNNs the loss function is just an average through time!

Figure from deeplearning.ai

Gated Recurrent Units (GRUs)

❑ Outline

- ✓ Gated recurrent unit (GRU) structure
- ✓ Comparison between GRUs and vanilla RNNs

Gated Recurrent Units (GRUs)

"Ants are really interesting. __They___ are everywhere."

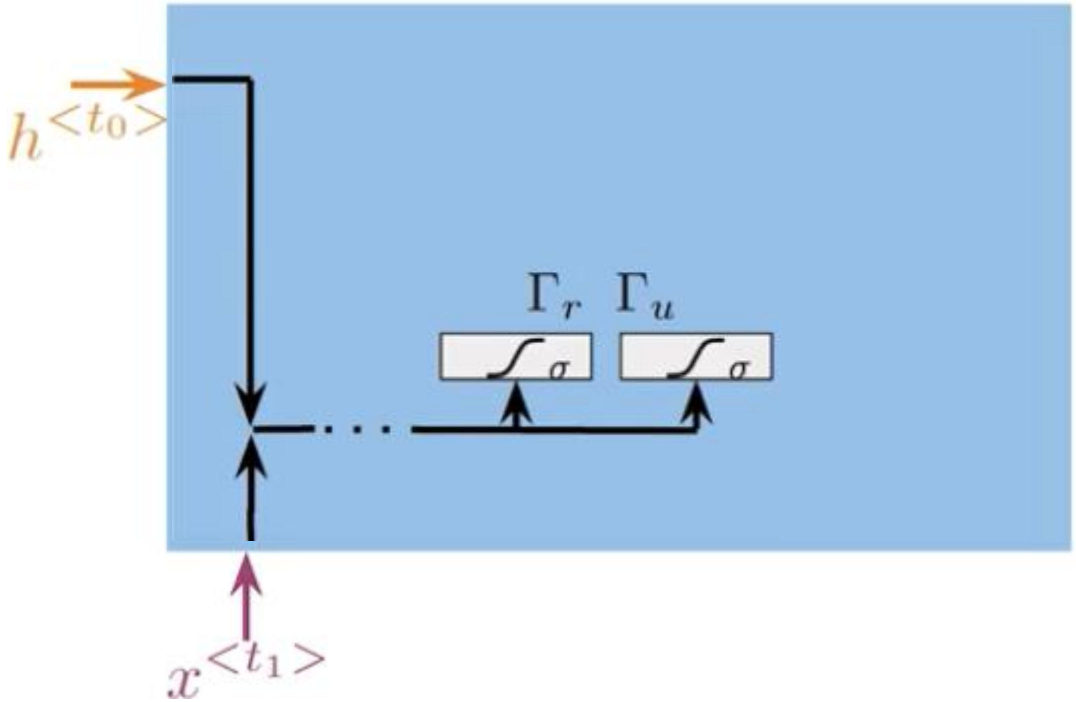
Plural

✓ GRU learns to keep the information about the subject, in this case, whether it is plural or singular, in the hidden states.

✓ **GRUs accomplish this by computing two gates:**

- Relevance gate
- Update gate

Gated Recurrent Units (GRUs)

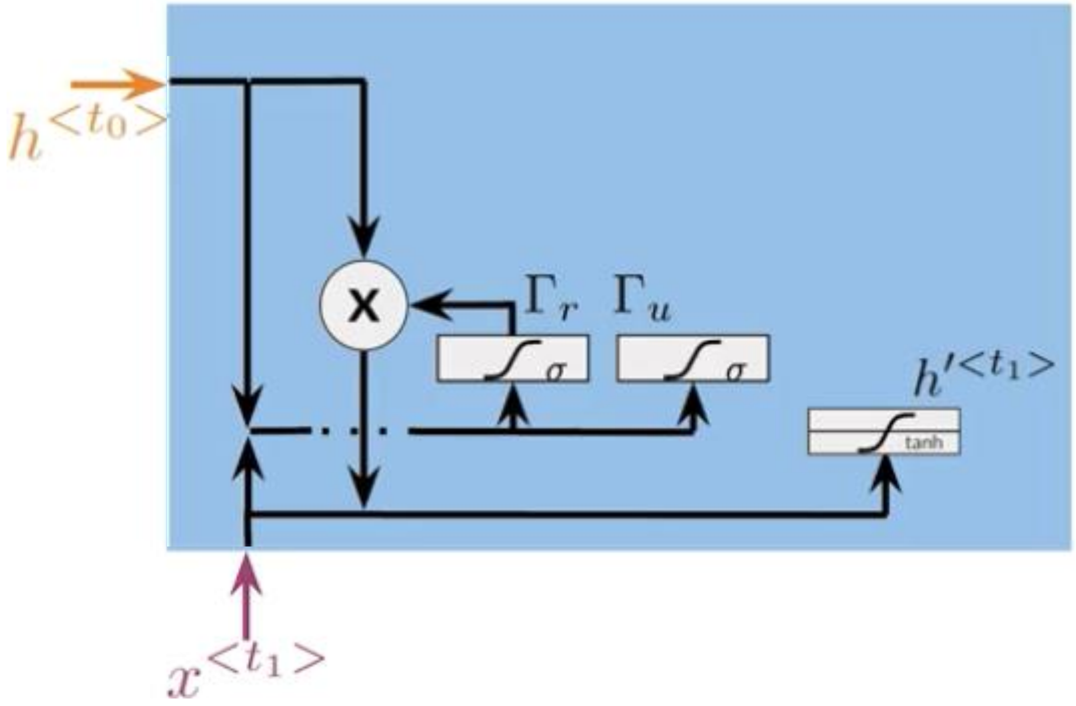


Gates to keep/update relevant information in the hidden state

$$\Gamma_r = \sigma(W_r[h^{<t_0>}, x^{<t_1>}] + b_r)$$
$$\Gamma_u = \sigma(W_u[h^{<t_0>}, x^{<t_1>}] + b_u)$$

- ✓ The update and relevance gates in GRUs are the most important computations.
- ✓ Relevance gate determines which information from the previous hidden states is relevant.
- ✓ Update gate determines which value should be updated with current information.

Gated Recurrent Units (GRUs)



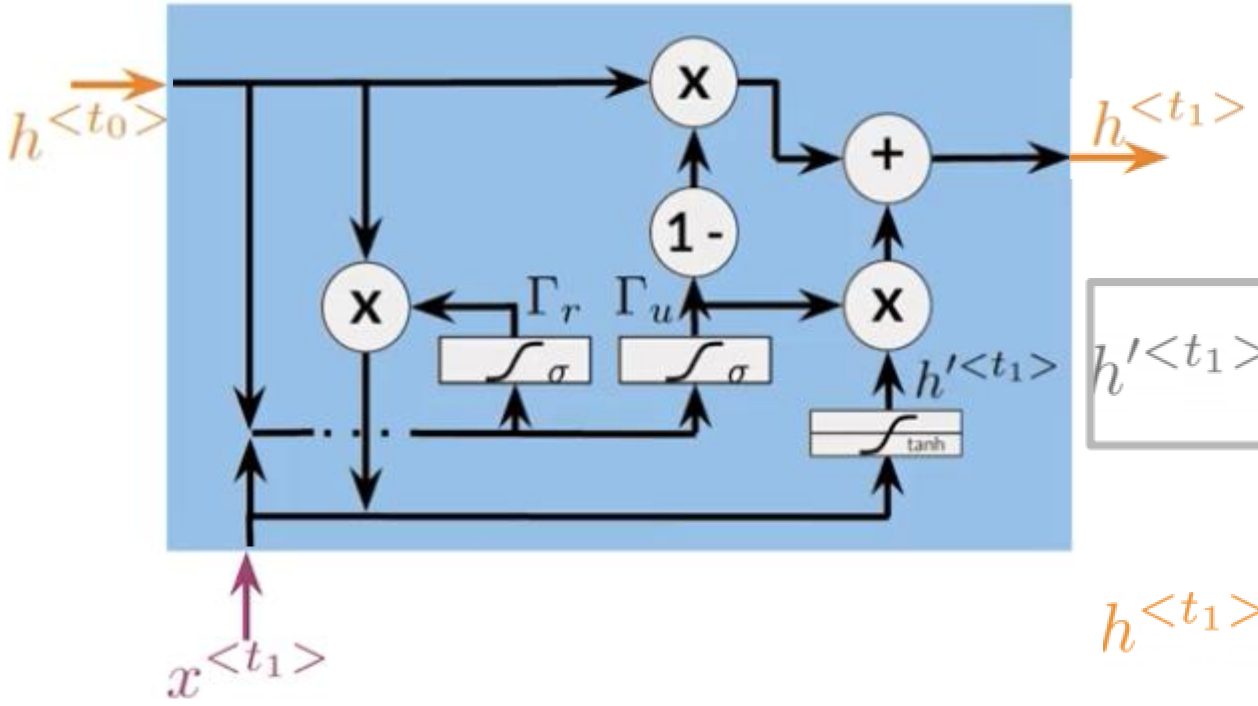
$$\Gamma_r = \sigma(W_r[h^{<t_0>}, x^{<t_1>}] + b_r)$$
$$\Gamma_u = \sigma(W_u[h^{<t_0>}, x^{<t_1>}] + b_u)$$

$$h'^{<t_1>} = \tanh\left(W_h[\Gamma_r * h^{<t_0>}, x^{<t_1>}] + b_h\right)$$

Hidden state candidate

✓ The "hidden state candidate" stores all the candidates for information thus could override the one contained in the previous hidden states.

Gated Recurrent Units (GRUs)



$$\Gamma_r = \sigma(W_r[h^{<t_0>}, x^{<t_1>}] + b_r)$$

$$\Gamma_u = \sigma(W_u[h^{<t_0>}, x^{<t_1>}] + b_u)$$

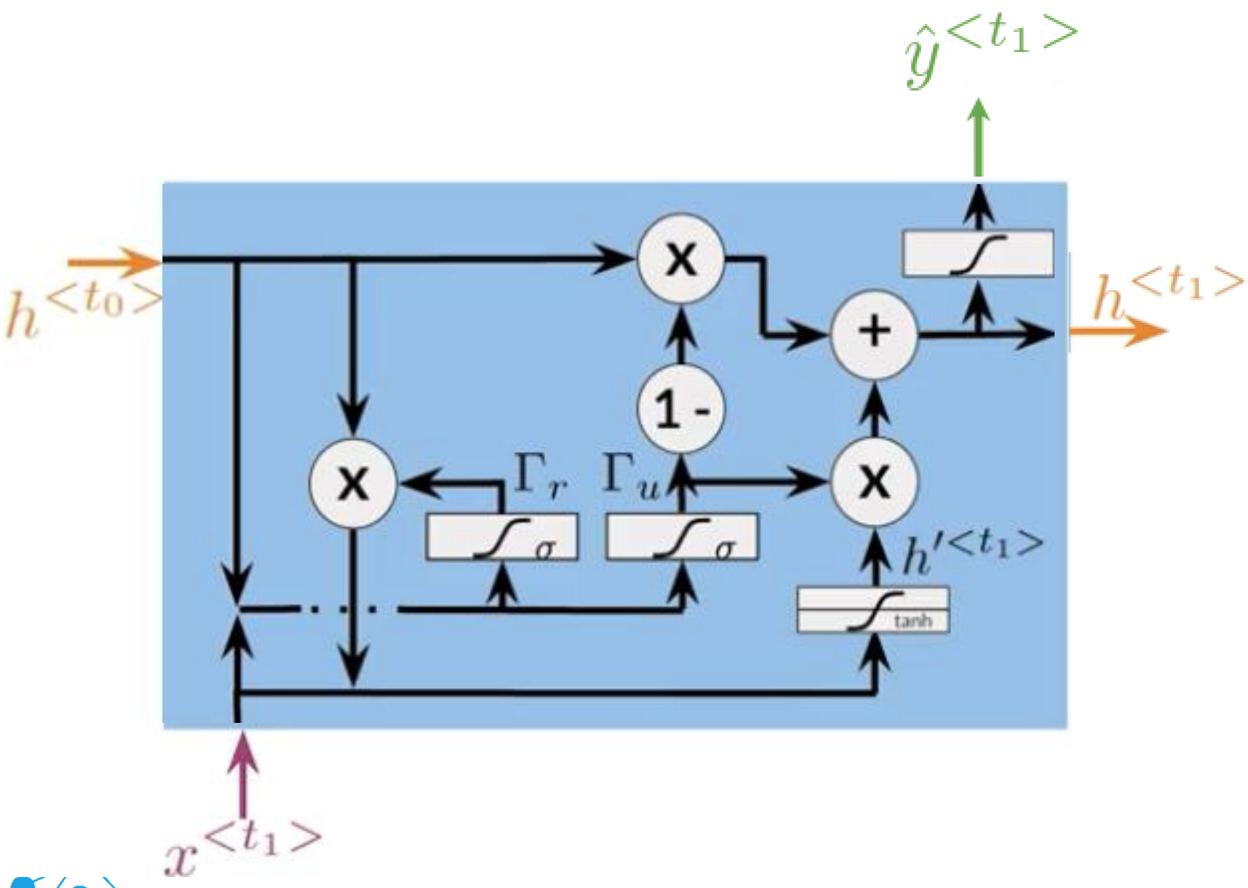
$$h'^{<t_1>} = \tanh\left(W_h[\Gamma_r * h^{<t_0>}, x^{<t_1>}] + b_h\right)$$

Hidden state candidate

$$h^{<t_1>} = (1 - \Gamma_u) * h^{<t_0>} + \Gamma_u * h'^{<t_1>}$$

✓ The updates gate determines how much of information from the previous hidden state will be overwritten.

Gated Recurrent Units (GRUs)



$$\Gamma_r = \sigma(W_r[h^{<t_0>}, x^{<t_1>}] + b_r)$$
$$\Gamma_u = \sigma(W_u[h^{<t_0>}, x^{<t_1>}] + b_u)$$

$$h'^{<t_1>} = \tanh\left(W_h[\Gamma_r * h^{<t_0>}, x^{<t_1>}] + b_h\right)$$

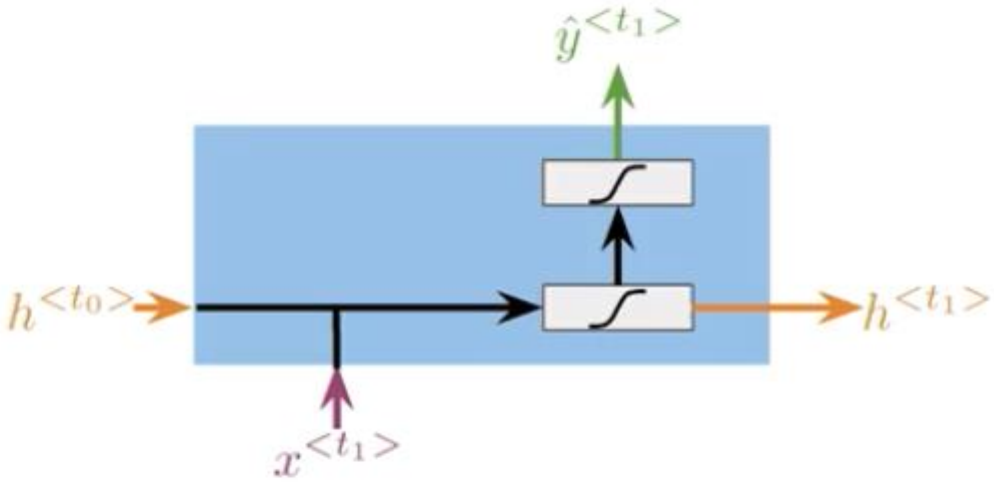
Hidden state candidate

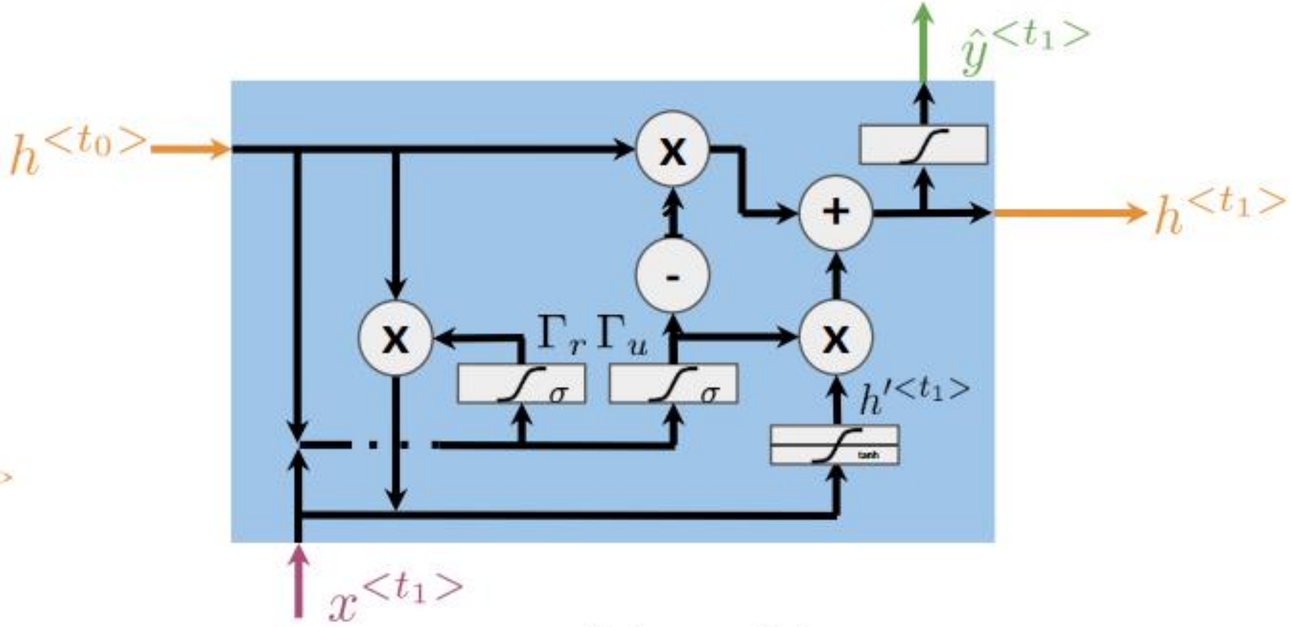
$$h^{<t_1>} = (1 - \Gamma_u) * h^{<t_0>} + \Gamma_u * h'^{<t_1>}$$

$$\hat{y}^{<t_1>} = g(W_y h^{<t_1>} + b_y)$$

Figure from deeplearning.ai

Vanilla RNNs vs. GRUs





$$h^{<t>} = g(W_h[h^{<t-1>}, x^{<t>}] + b_h)$$

$$\hat{y}^{<t>} = g(W_{yh} h^{<t>} + b_y)$$

$$\Gamma_u = \sigma(W_u[h^{<t_0>}, x^{<t_1>}] + b_u)$$

$$\Gamma_r = \sigma(W_r[h^{<t_0>}, x^{<t_1>}] + b_r)$$

$$h'^{<t_1>} = \tanh\left(W_h[\Gamma_r * h^{<t_0>}, x^{<t_1>}] + b_h\right)$$

$$h^{<t_1>} = (1 - \Gamma_u) * h^{<t_0>} + \Gamma_u * h'^{<t_1>}$$

$$\hat{y}^{<t_1>} = g(W_y h^{<t_1>} + b_y)$$

- ✓ RNN updates the hidden state at every time step. So for long sequences, the information tends to vanish (Vanishing Gradient problem).

GRUs

❑ Summary

- ✓ GRUs “decide” how to update the hidden state.
- ✓ GRUs help preserve important information.

Deep and Bidirectional RNNs

❑ Outline

- ✓ How bidirectional RNNs propagate information
- ✓ Forward propagation in deep RNNs

Bidirectional RNNs

I was trying really hard to get a hold of ___________. **Louise,** finally answered when I was about to give up.

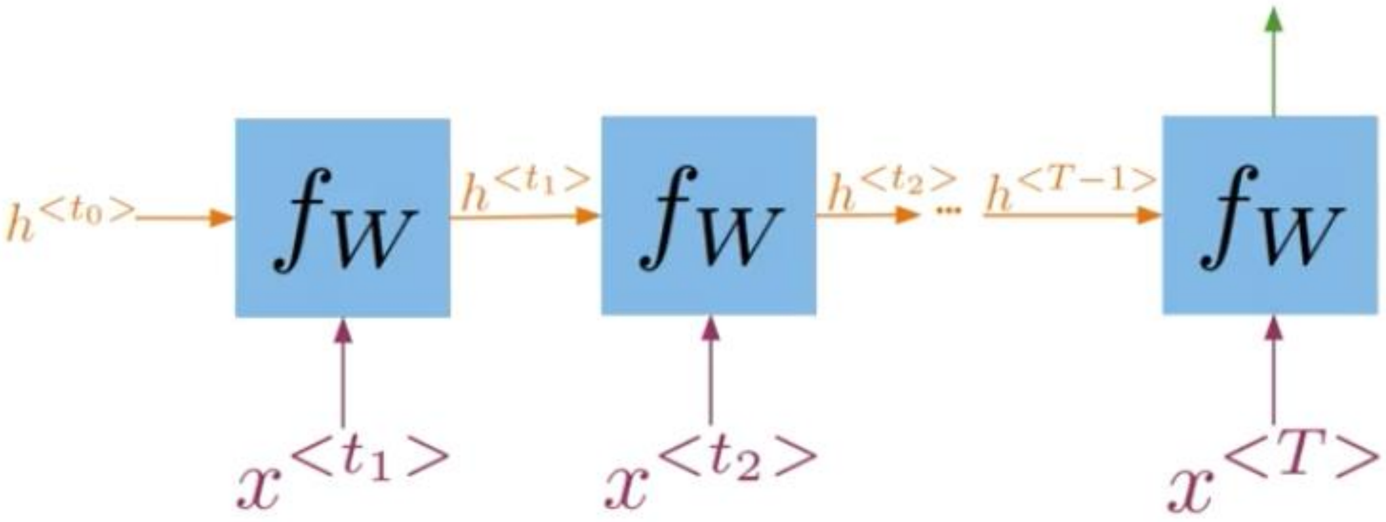


Figure from deeplearning.ai

Bidirectional RNNs

I was trying really hard to get a hold of ___________. **Louise**, finally answered when I was about to give up.

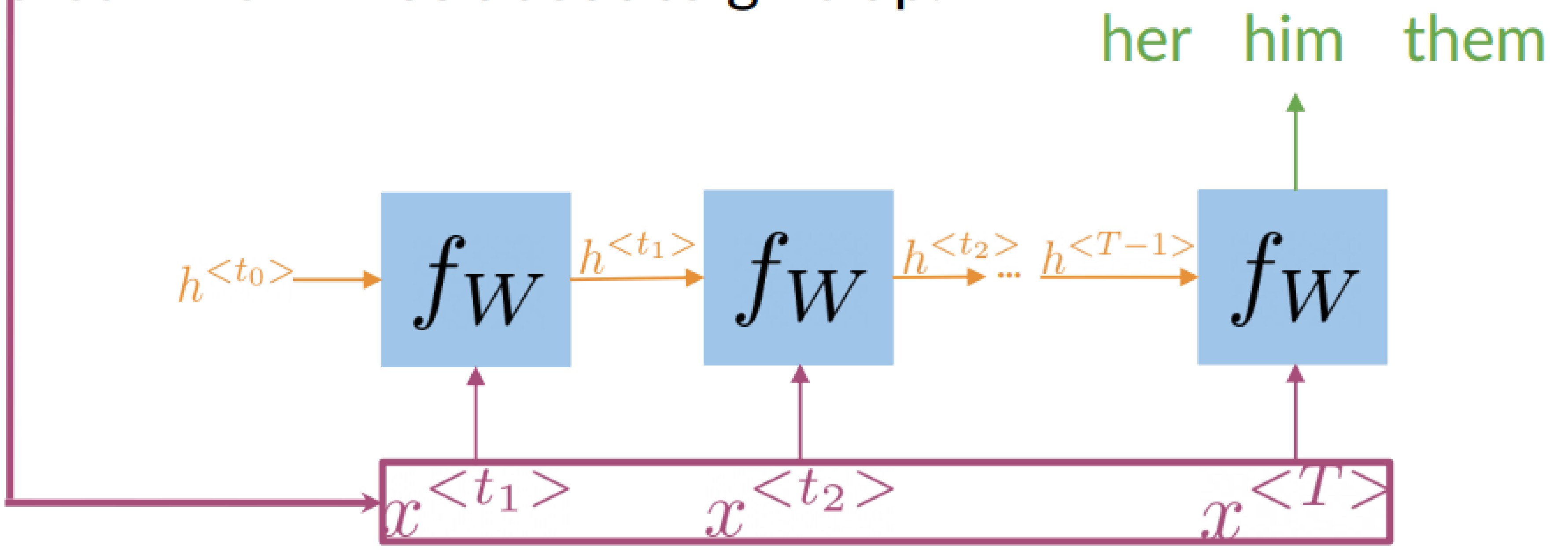
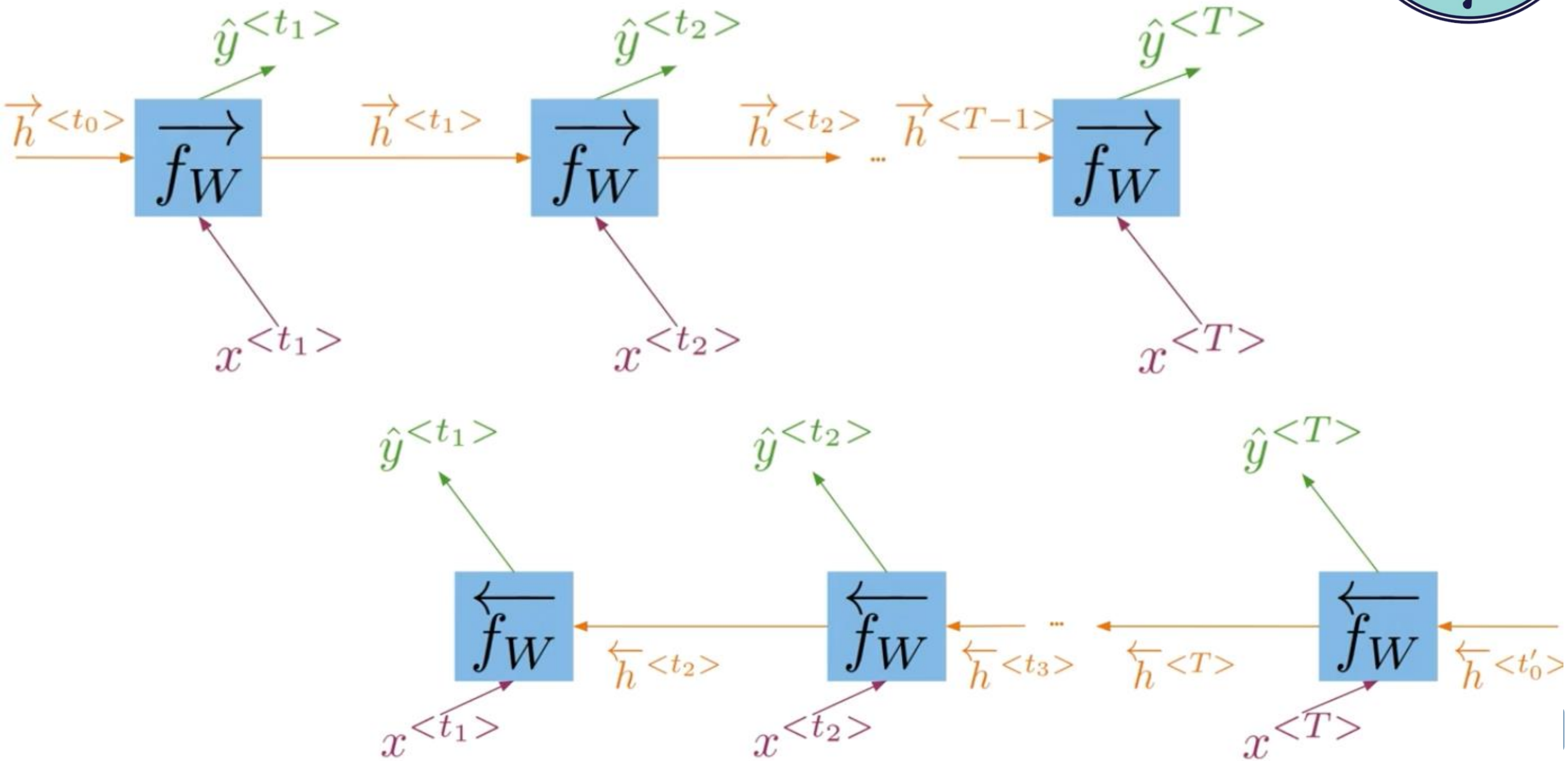
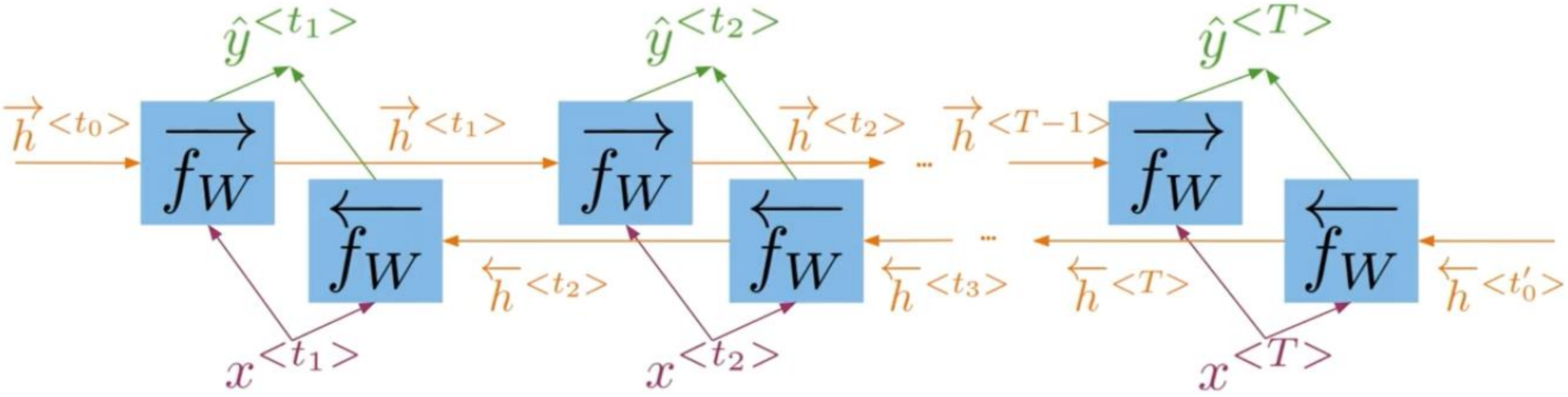


Figure from deeplearning.ai

Bidirectional RNNs



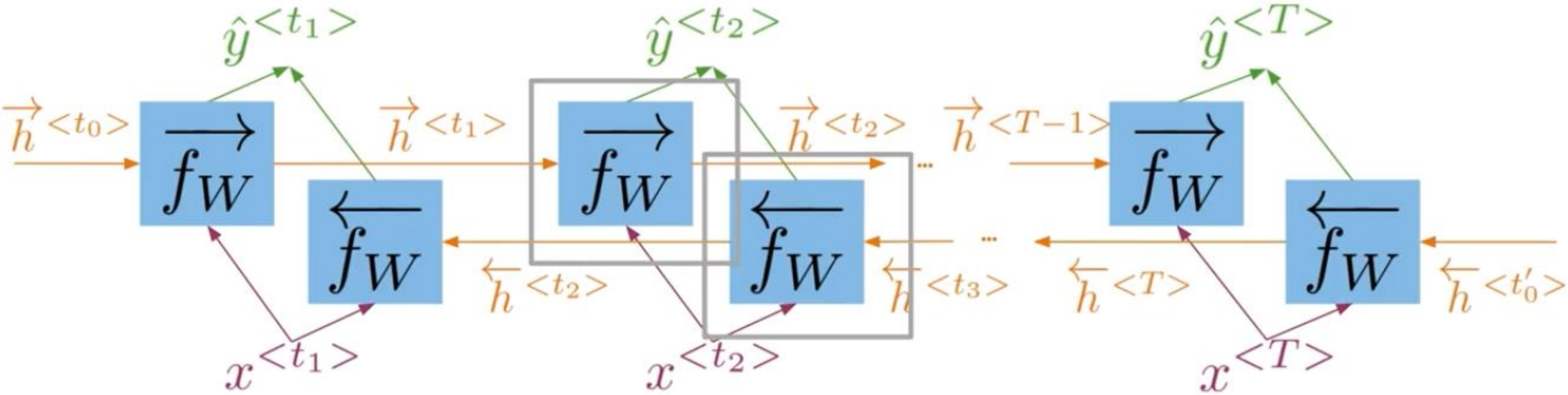
Bidirectional RNNs



Information flows from the past and from the future **independently**

Figure from deeplearning.ai

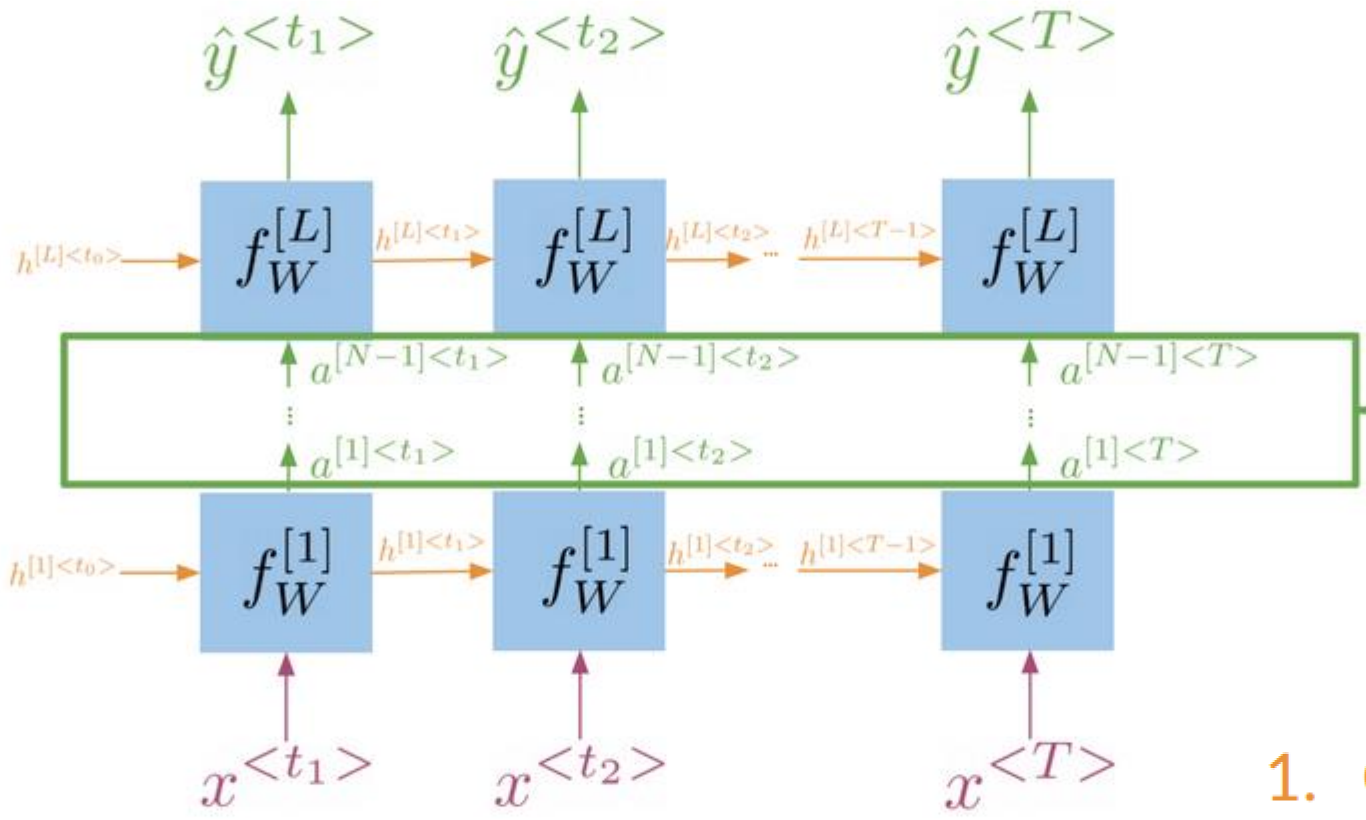
Bidirectional RNNs



$$\hat{y}^{<t>} = g(W_y \boxed{[\overrightarrow{h}^{<t>}, \overleftarrow{h}^{<t>}]} + b_y)$$

Figure from deeplearning.ai

Deep RNNs



$$h^{[l]<t>} = f^{[l]}(W_h^{[l]}[h^{[l]<t-1>}, a^{[l-1]<t>}] + b_h^{[l]})$$

$$a^{[l]<t>} = f^{[l]}(W_a^{[l]} h^{[l]<t>} + b_a^{[l]})$$

Intermediate layers and activations

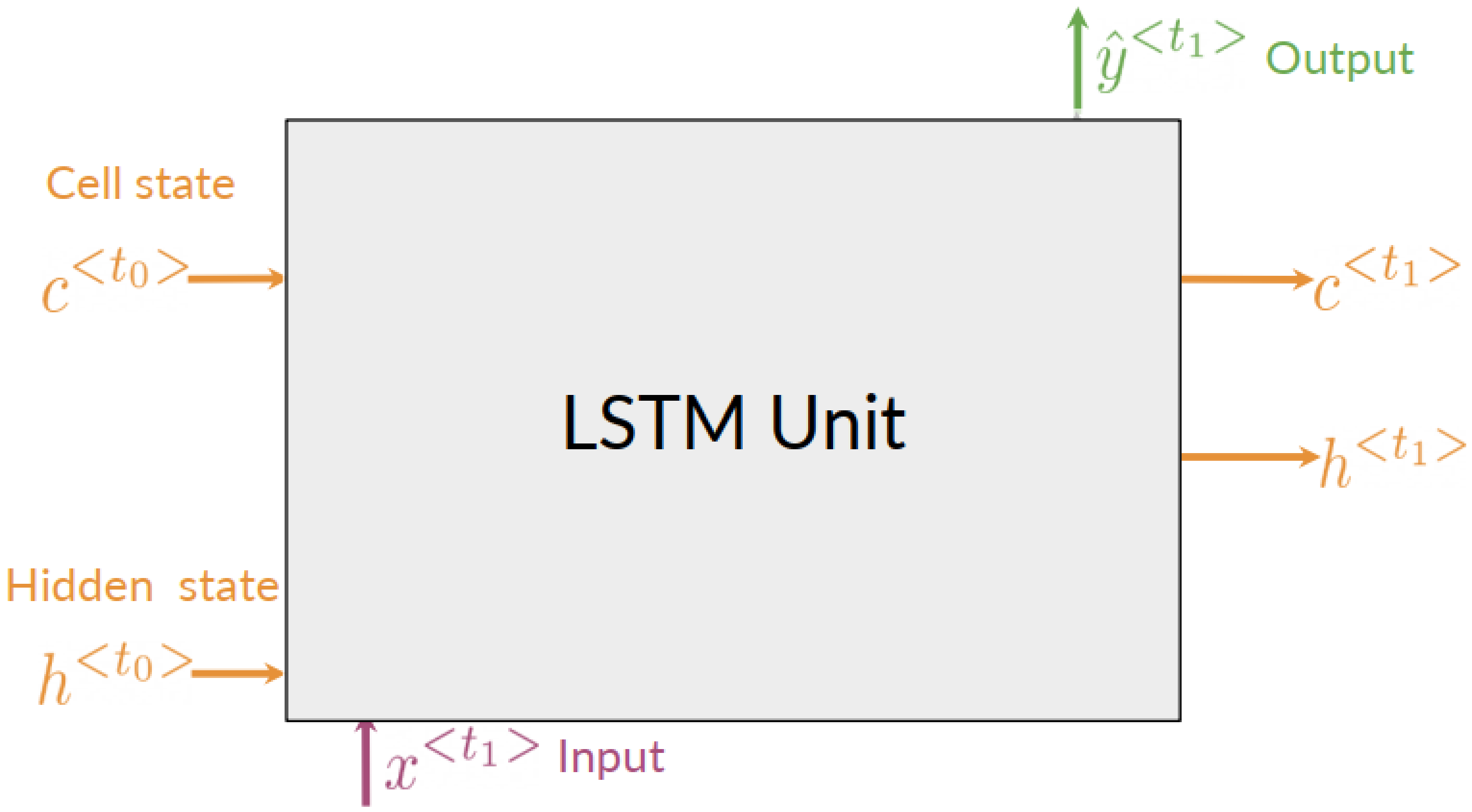
1. Get hidden states for current layer
2. Pass the activations to the next layer

Figure from deeplearning.ai

Deep and Bidirectional RNNs

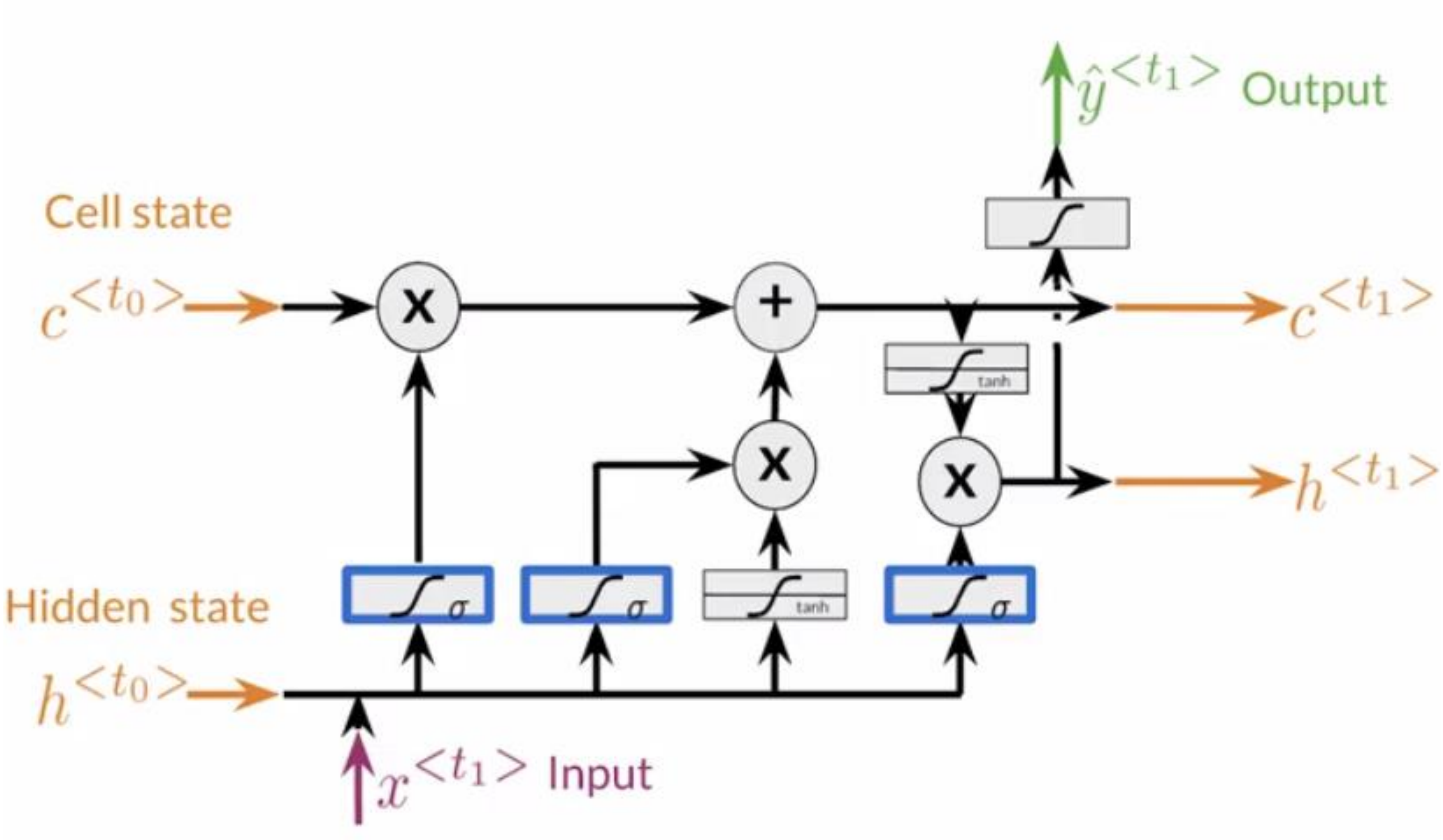
- **Summary**
  - ✓ In bidirectional RNNs, the outputs take information from the past and the future.
  - ✓ Deep RNNs have more than one layer, which helps in complex tasks.

LSTMs



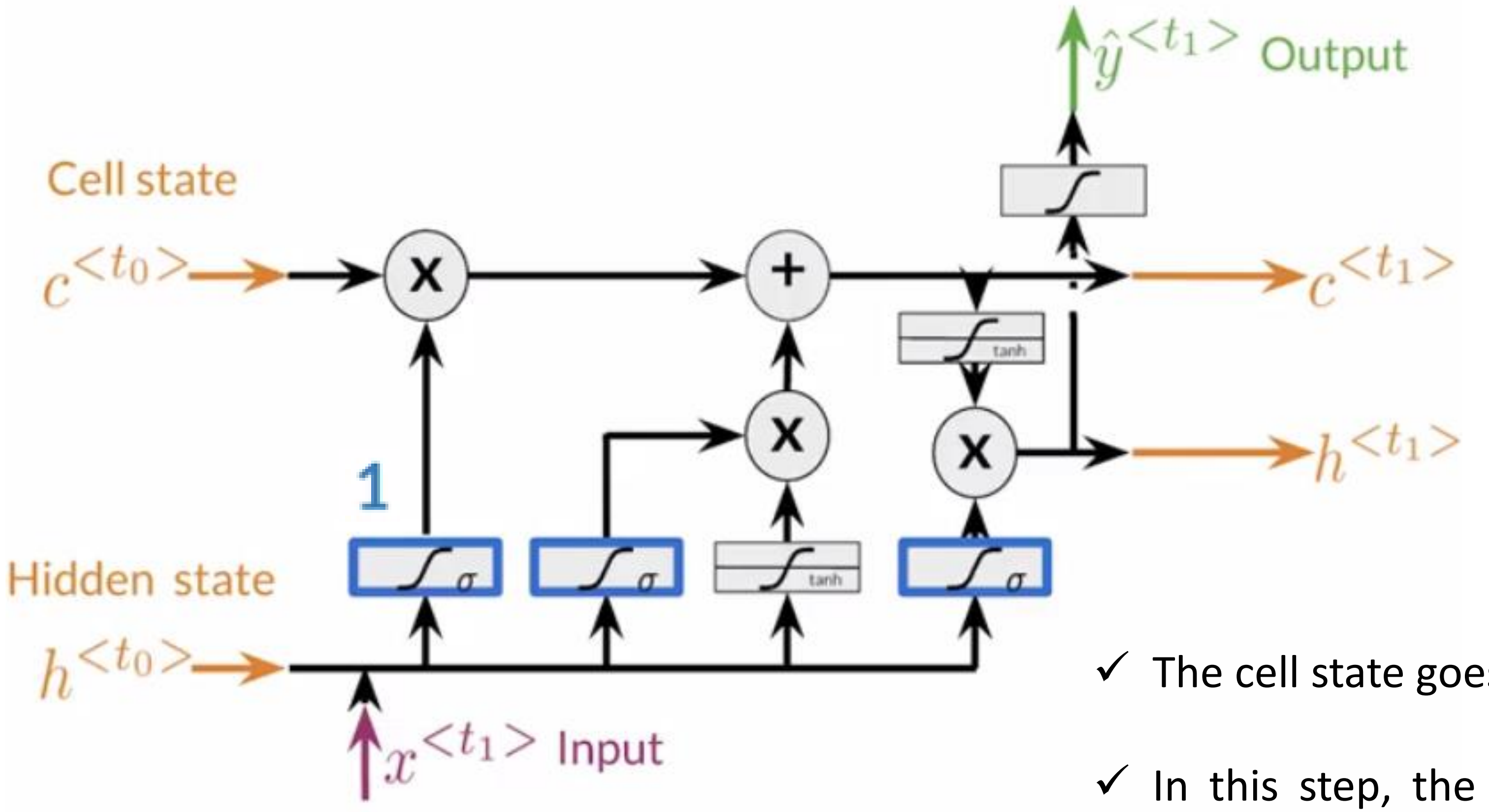
$\hat{y}^{<t_1>}$ Output
Cell state
$c^{<t_0>}$
LSTM Unit
$c^{<t_1>}$
$h^{<t_1>}$
Hidden state
$h^{<t_0>}$
$x^{<t_1>}$ Input

LSTMs



$\hat{y}^{<t_1>}$ Output
Cell state
$c^{<t_0>}$
x
+
tanh
x
x
$c^{<t_1>}$
$h^{<t_1>}$
Hidden state
σ
σ
tanh
σ
$h^{<t_0>}$
$x^{<t_1>}$ Input

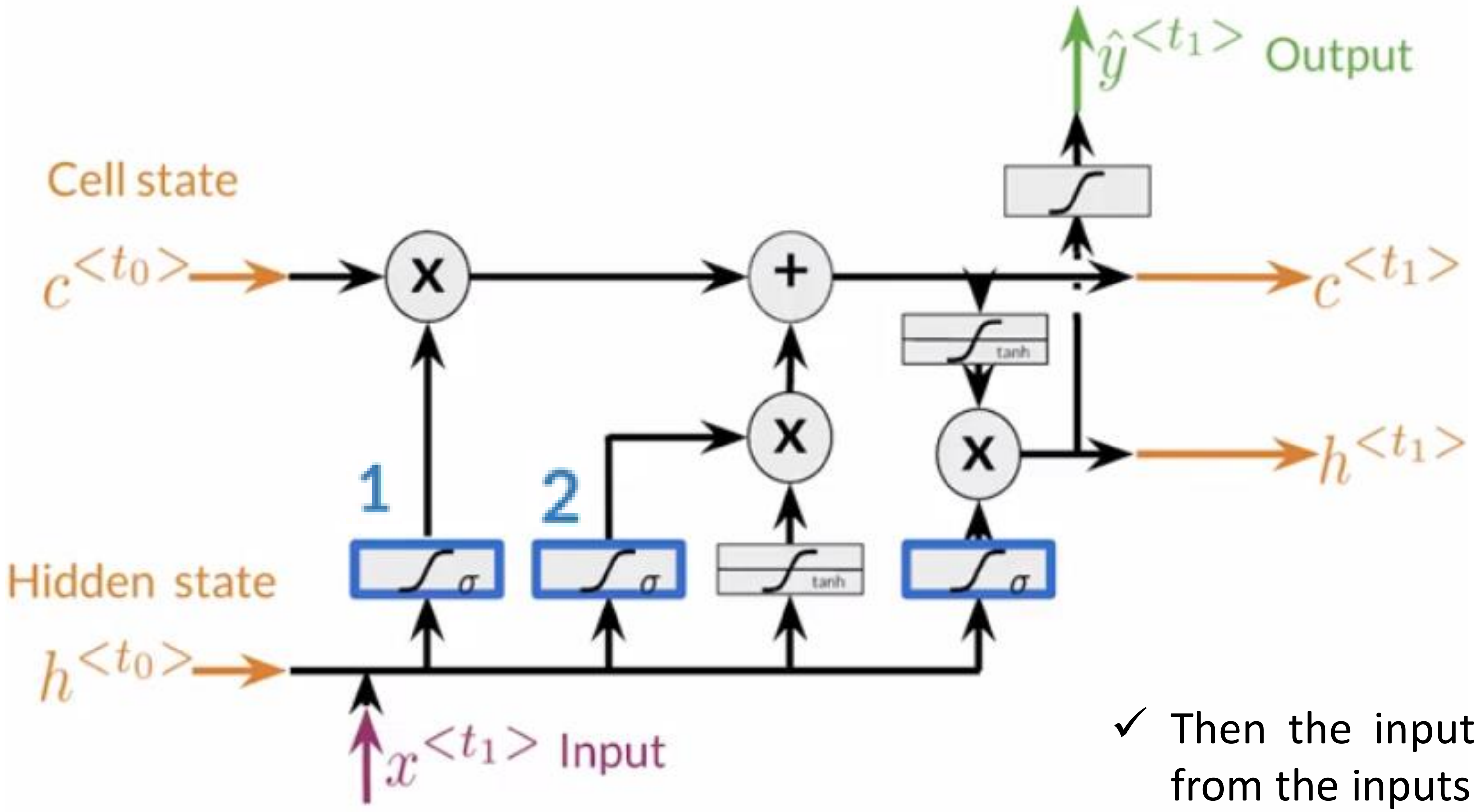
LSTMs



1. Forget Gate:
information that is no longer important

- ✓ The cell state goes through a forget gate.
- ✓ In this step, the inputs and the previous hidden states are used to decide which information from the cell state is no longer important and throws it away.

LSTMs

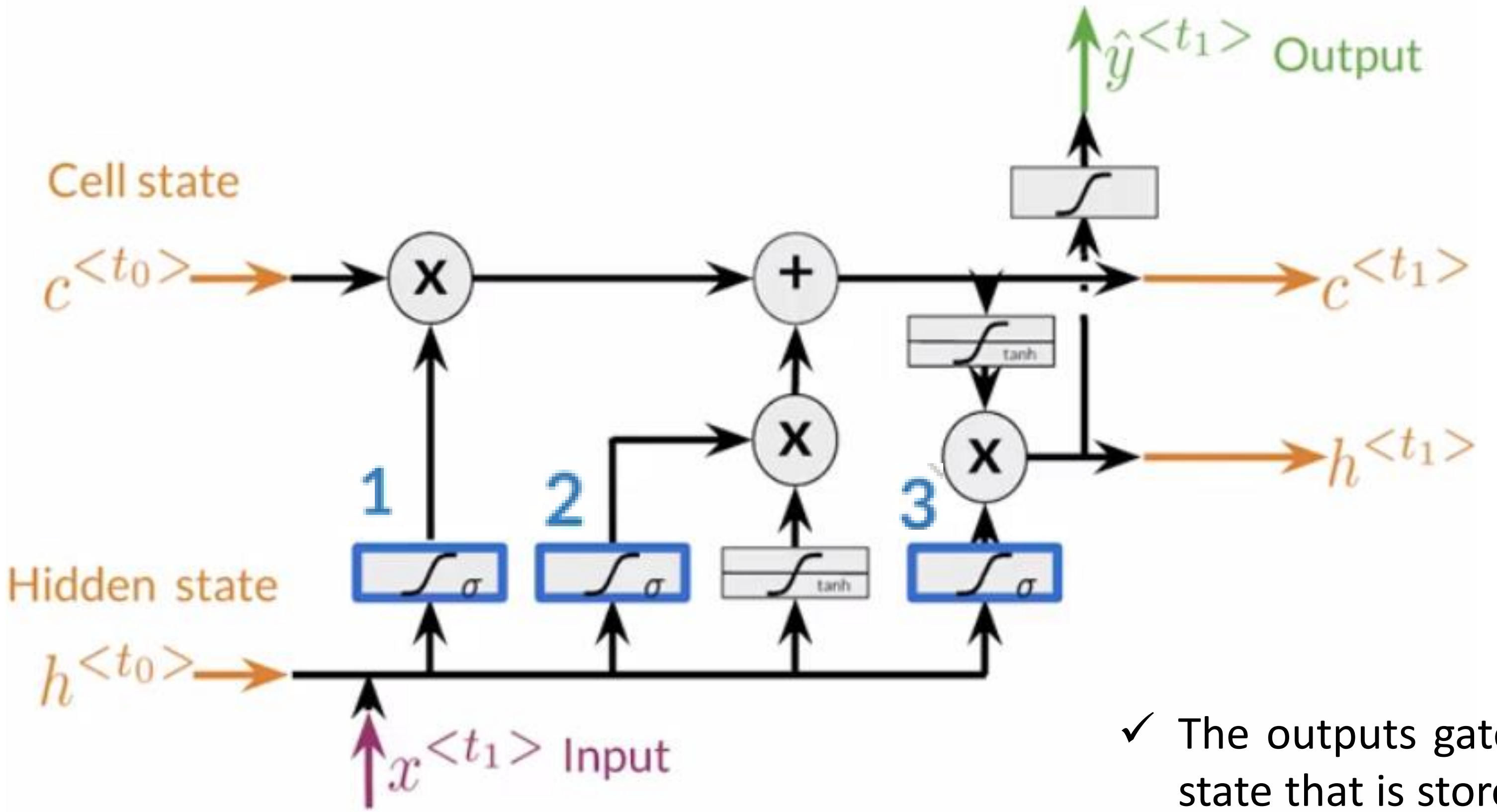


1. Forget Gate: information that is no longer important

2. Input Gate: information to be stored

✓ Then the input gate is used to decide which information from the inputs and the previous hidden state is relevant, so it is added to the cell state.

LSTMs



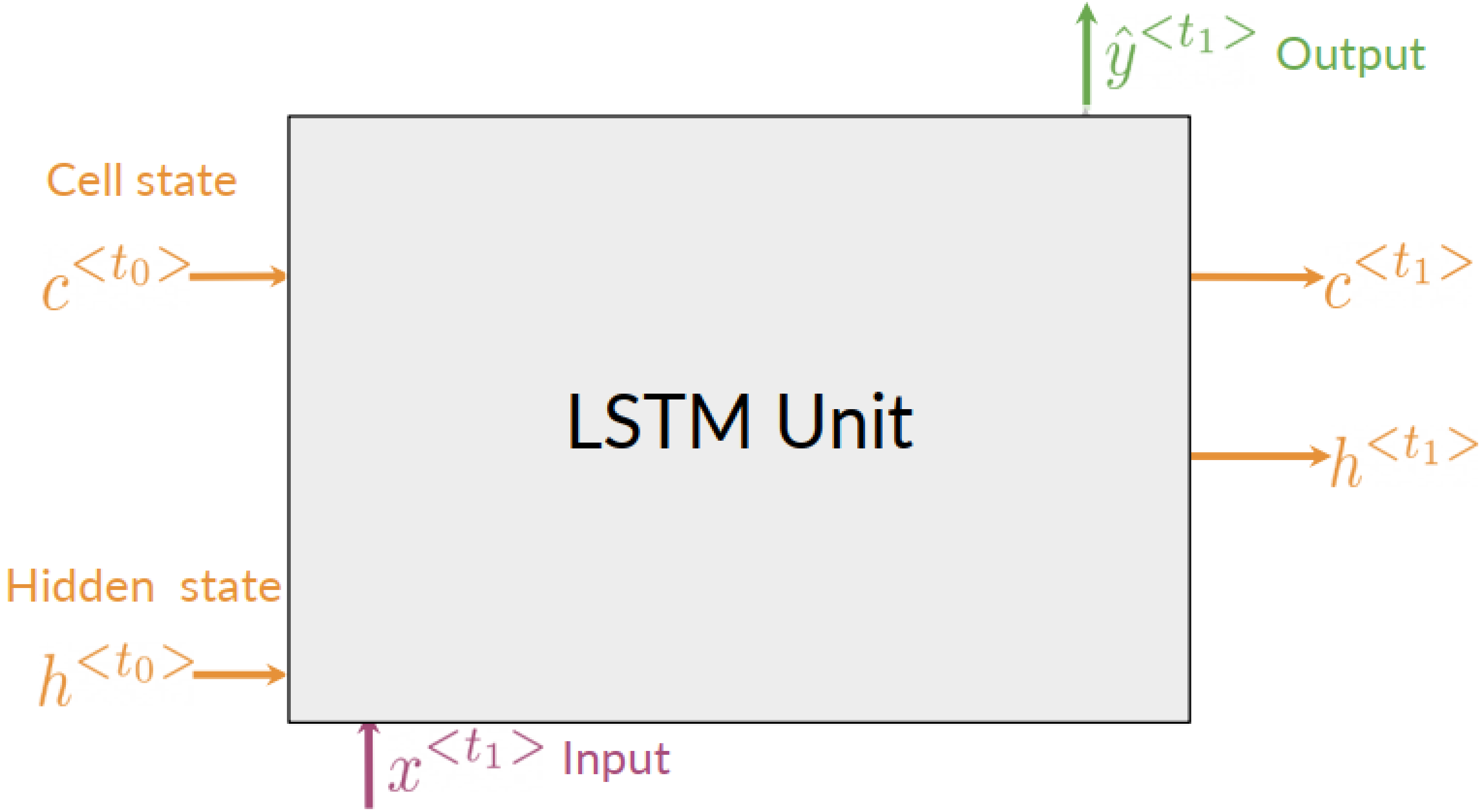
1. Forget Gate: information that is no longer important

2. Input Gate: information to be stored

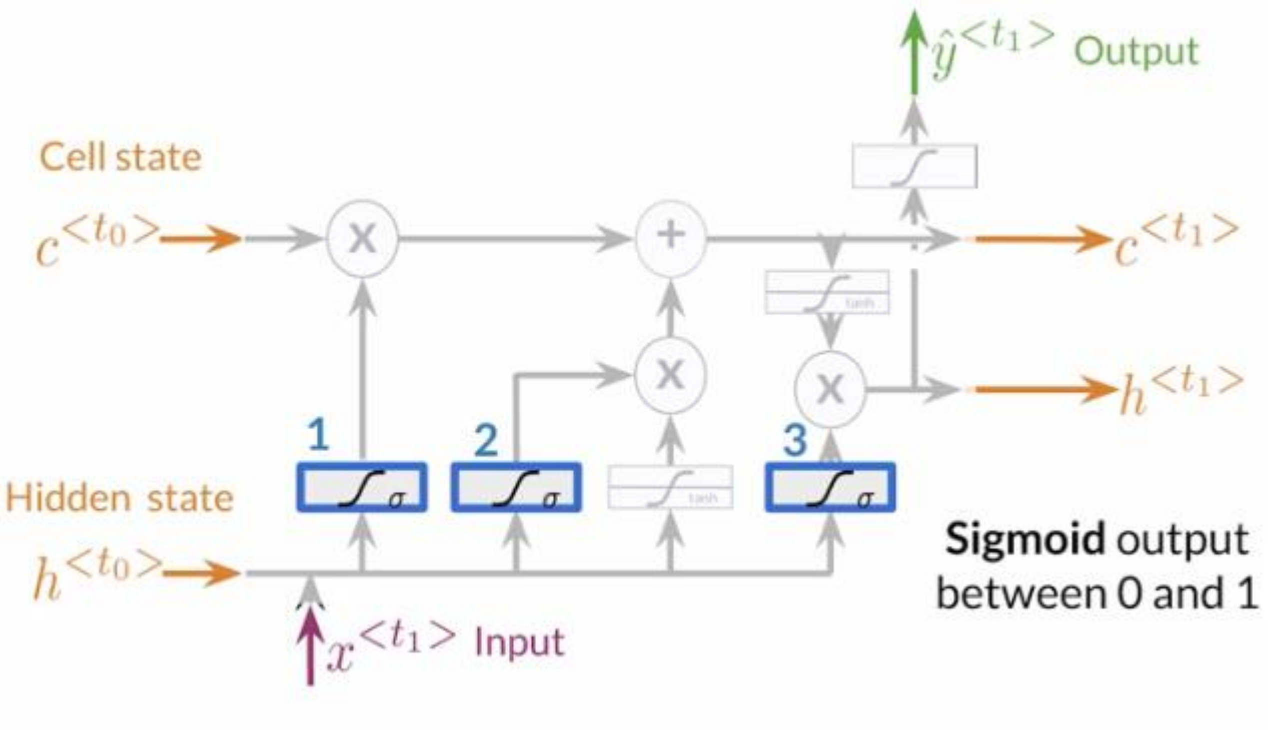
3. Output Gate: information to use at current step

✓ The outputs gate determines the information from the cell state that is stored in the hidden state and used to construct an output at the given time step.

LSTMs


$\hat{y}^{<t_1>}$ Output
Cell state
$c^{<t_0>}$
$c^{<t_1>}$
LSTM Unit
$h^{<t_1>}$
Hidden state
$h^{<t_0>}$
$x^{<t_1>}$ Input

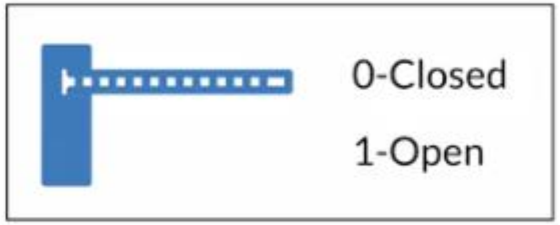
Gates in LSTMs



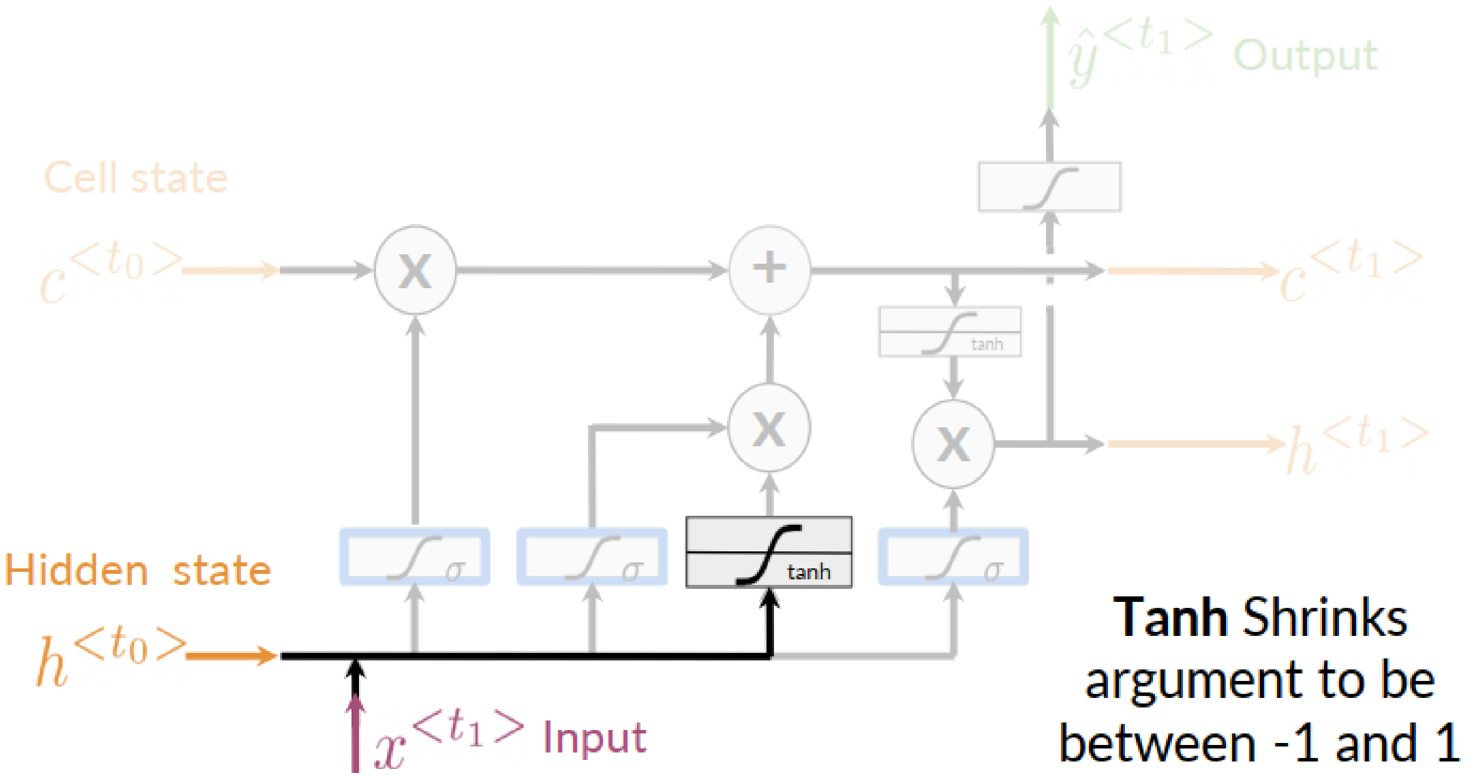
1. **Forget Gate:** information that is no longer important

2. **Input Gate:** information to be stored

3. **Output Gate:** information to use at current step



Candidate Cell State

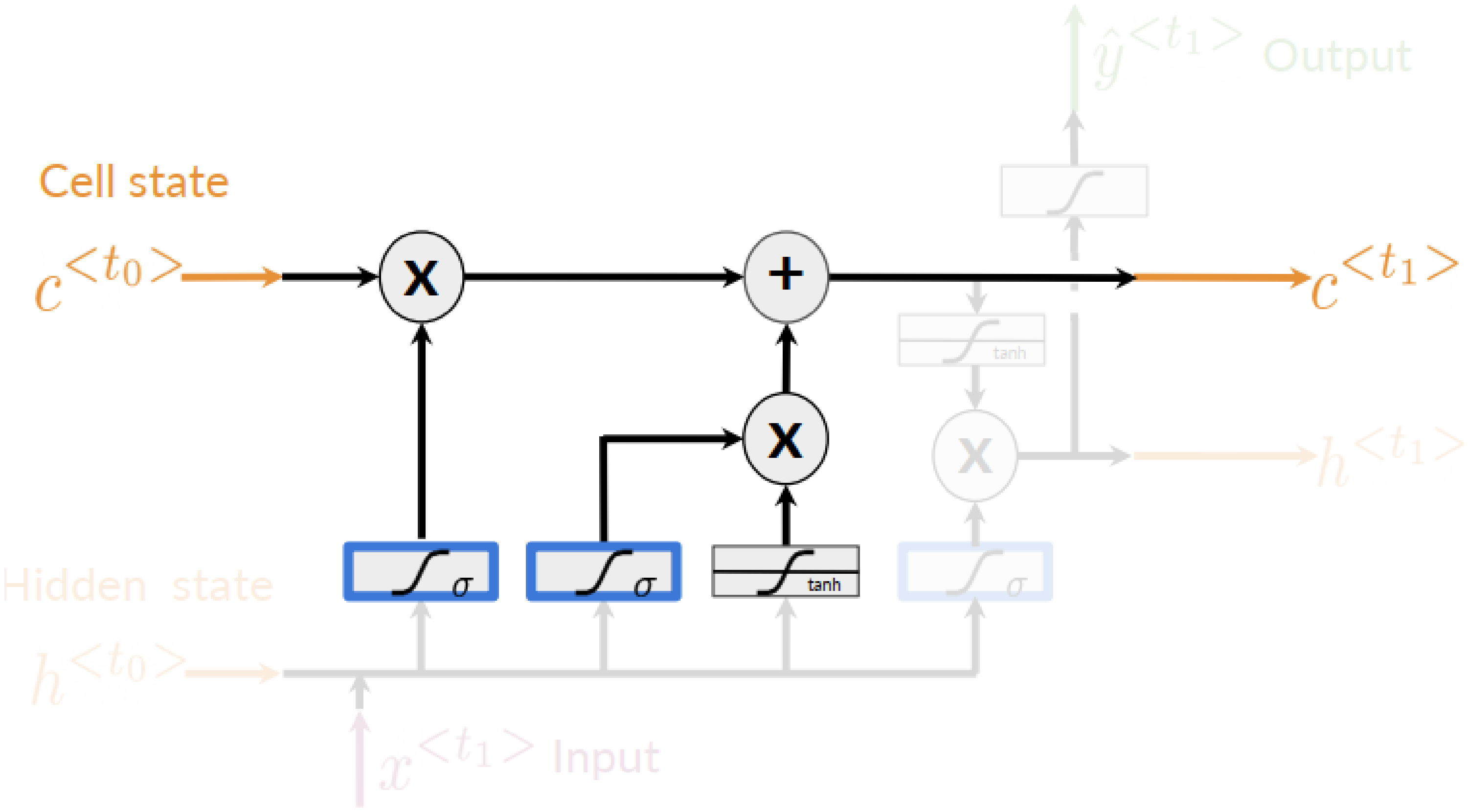


Candidate cell state

Information from the previous hidden state and current input

Tanh Shrinks argument to be between -1 and 1 → **Other activations could be used**

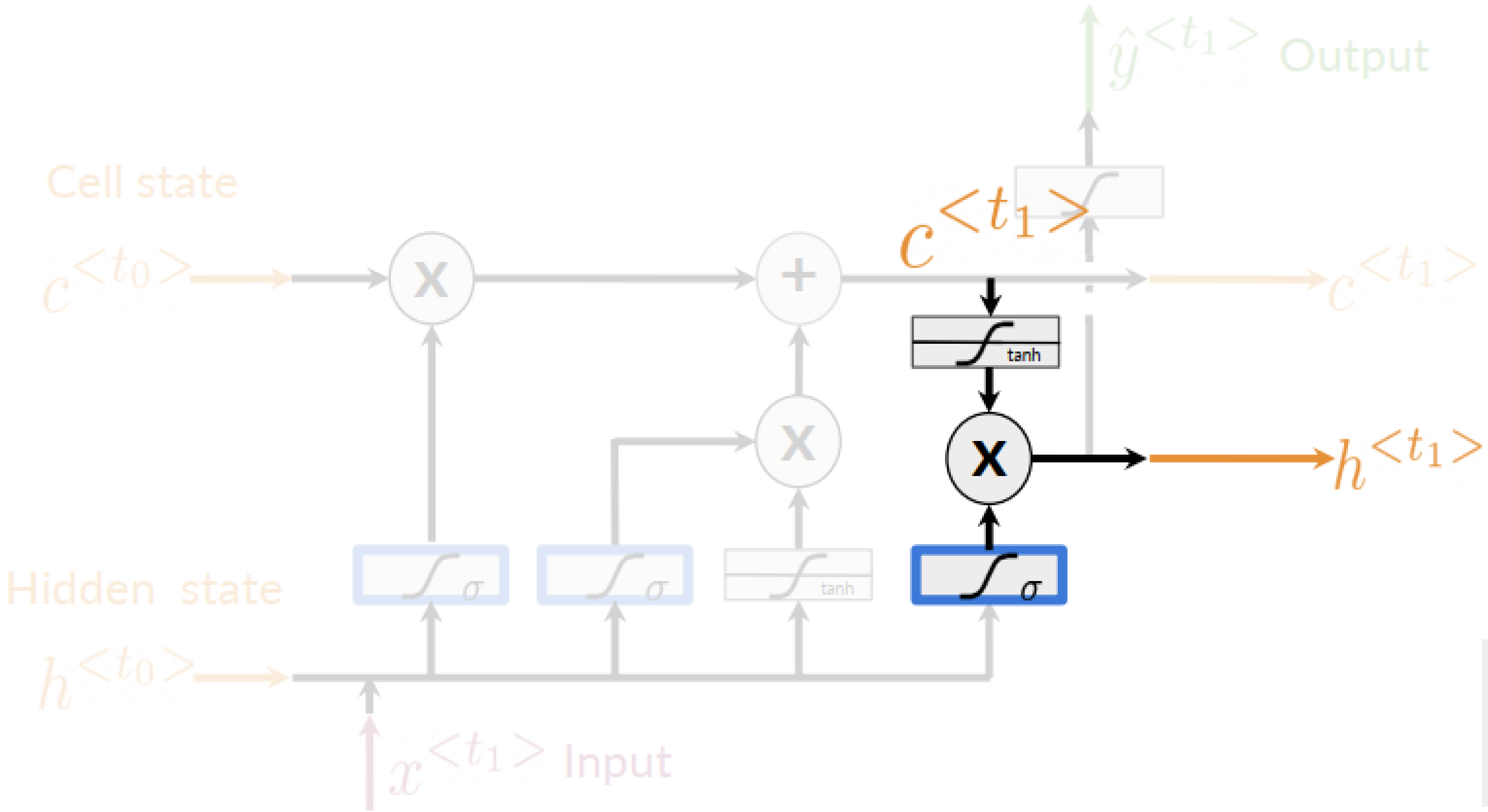
New Cell State



New Cell state

Add information from the **candidate cell state** using the **forget** and **input gates**

New Hidden State



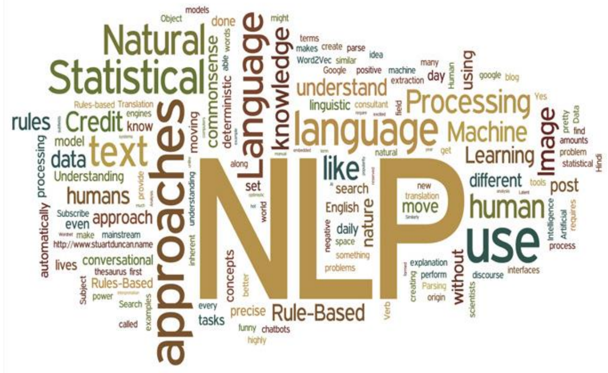
New Hidden State

Select information from the new cell state using the output gate

The **Tanh** activation could be omitted

Summary

- LSTMs use a series of gates to decide which information to keep:
  - Forget gate decides what to keep
  - Input gate decides what to add
  - Output gate decides what the next hidden state will be



با تشکر از توجه شما