An Introduction to Recurrent Neural Networks

Sangeeta Oswal

VESIT

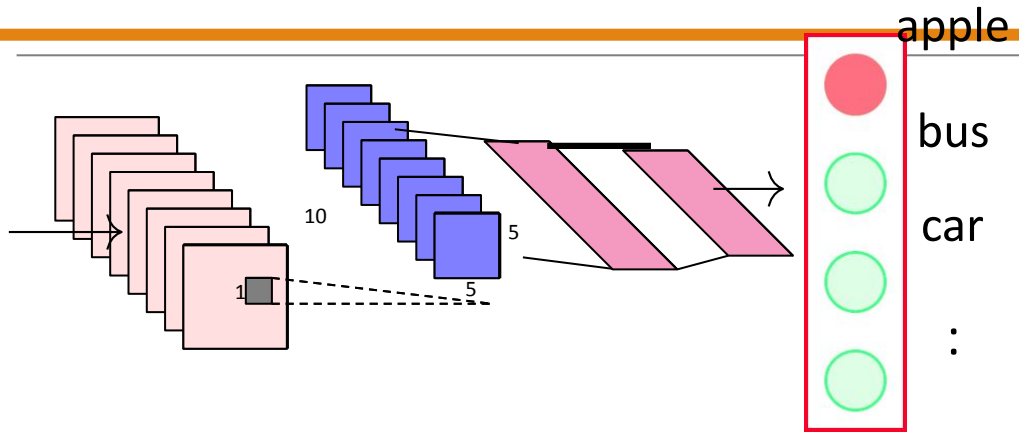
Outline

Recurrent Neural Networks
- Vanilla RNN
- Exploding and Vanishing Gradients

Networks with Memory
- Long Short-Term Memory (LSTM)
- Gated Recurrent Unit (GRU)

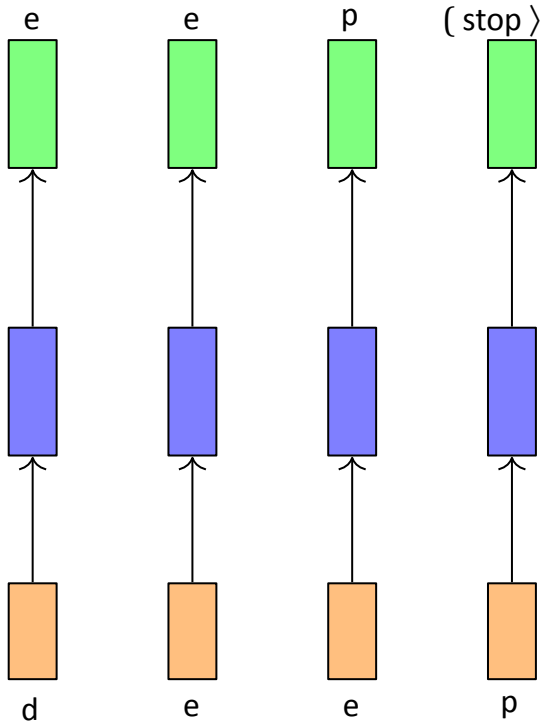
Sequence Learning Architectures
- Sequence Learning with one RNN Layer
- Sequence Learning with multiple RNN Layers

Sequence Learning



- In feedforward and convolutional neural networks the size of the input was always fixed
- For example, we fed fixed size (32 × 32) images to convolutional neural networks for image classification
- Similarly in word2vec, we fed a fixed window (*k*) of words to the network
- Further, each input to the network was independent of the previous or future inputs
- For example, the computatations, outputs and decisions for two success- ive images are completely independ- ent of each other

Sequence Learning



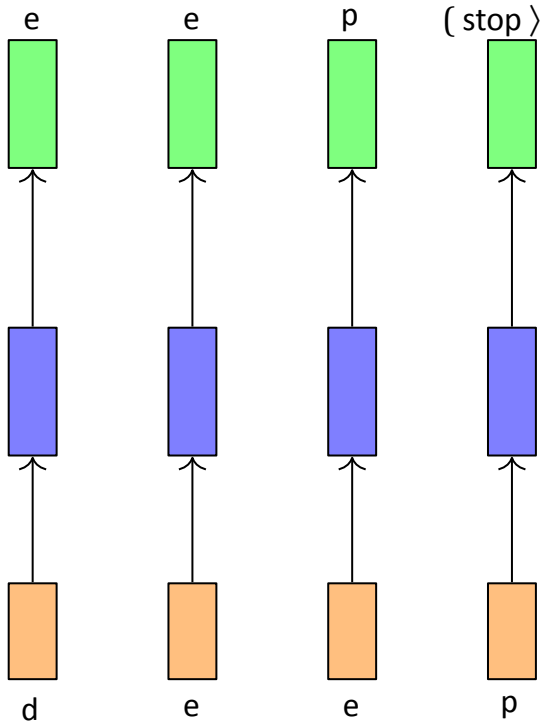
In many applications the input is not of a fixed size

Further successive inputs may not be independent of each other

For example, consider the task of auto completion

Given the first character ‘d’ you want to predict the next character ‘e’ and so on

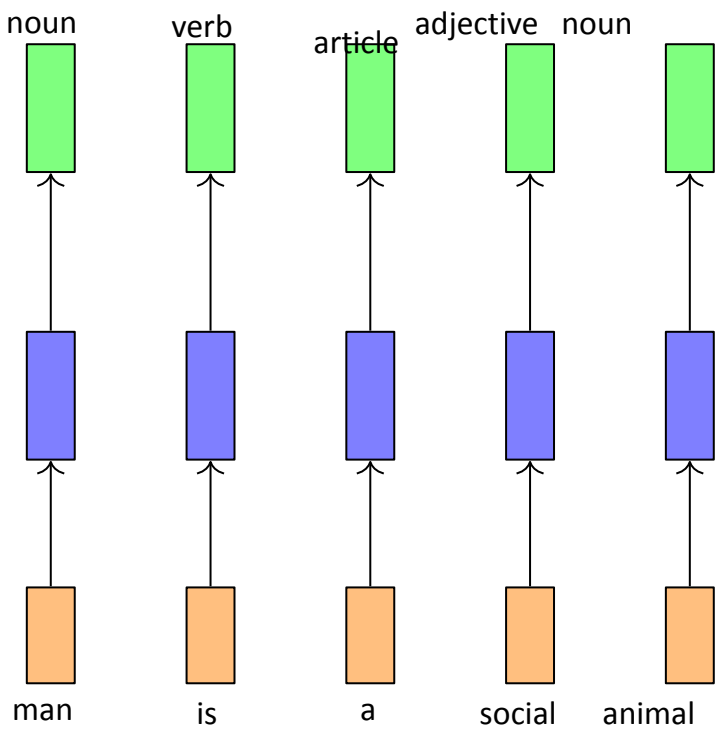
Sequence Learning



First, successive inputs are no longer independent (while predicting ‘e’ you would want to know what the previous input was in addition to the cur- rent input)

Second, the length of the inputs and the number of predictions you need to make is not fixed (for example, “learn”, “deep”, “machine” have different number of characters)

Third, each network (orange-blue- green structure) is performing the same task (**input** : character **output**: character)

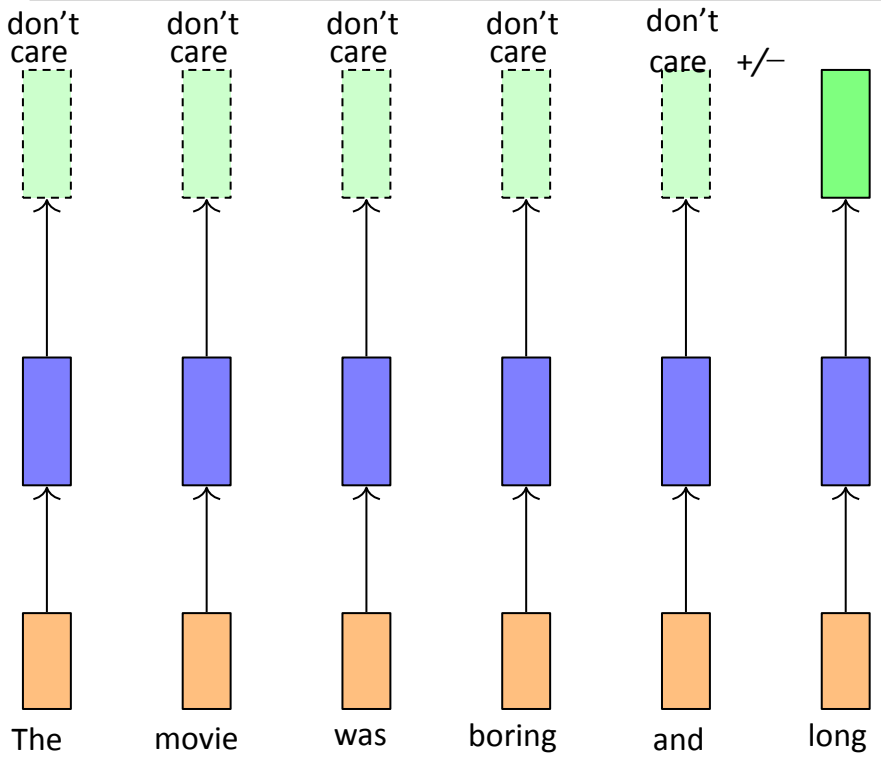


Consider the task of predicting the part of speech tag (noun, adverb, adjective verb) of each word in a sentence

Once we see an adjective (social) we are <u>almost</u> sure that the next word should be a noun (man)

Thus the current output (noun) depends on the current input as well as the previous input

Further the size of the input is not fixed (sentences could have arbitrary number of words)



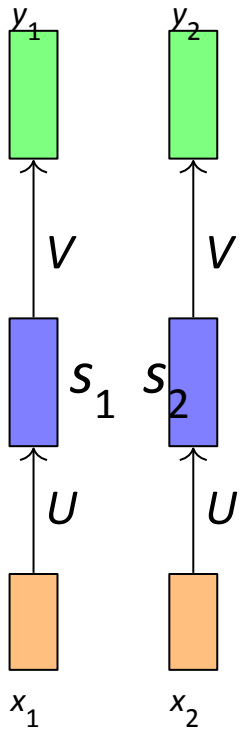
Instead we would look at the full sequence and then produce an output

For example, consider the task of predicting the polarity of a movie review

The prediction clearly does not depend only on the last word but also on some words which appear before

Here again we could think that the network is performing the same task at each step (input : word, output : +/−) but it's just that we don't care about intermediate outputs

Sequence Learning : RNN

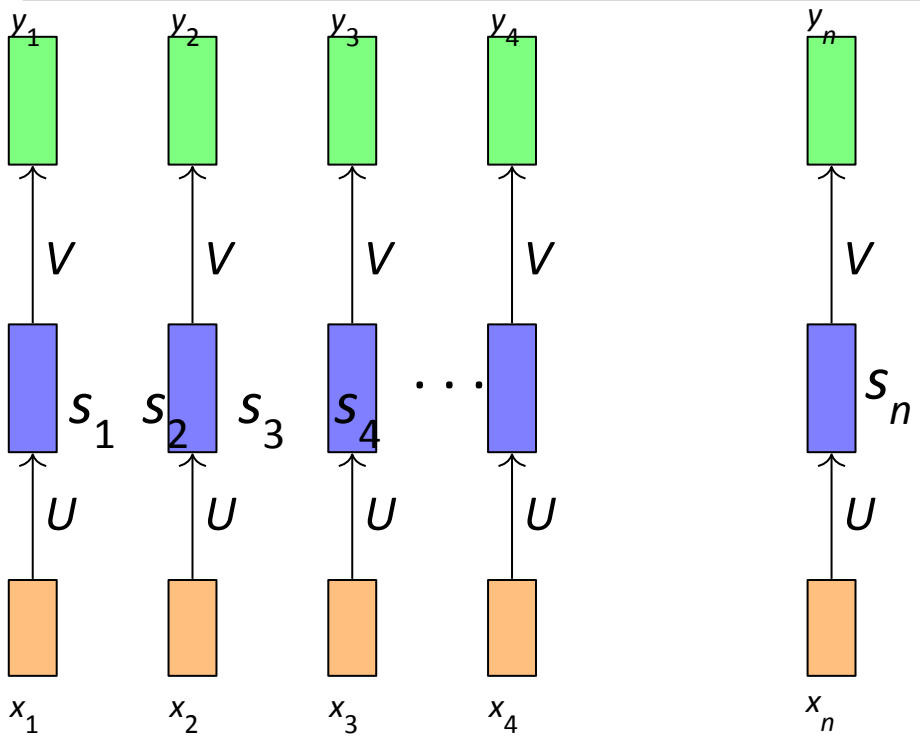


What is the function being executed at each time step ?

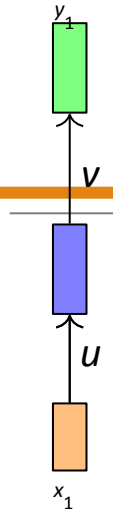
$$s_i = \sigma(Ux_i + b) \quad y_i = O(V s_i + c) \quad i = \text{timestep}$$

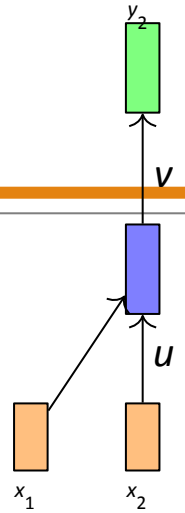
- Since we want the same function to be executed at each timestep we should share the same network (i.e., same parameters at each timestep)

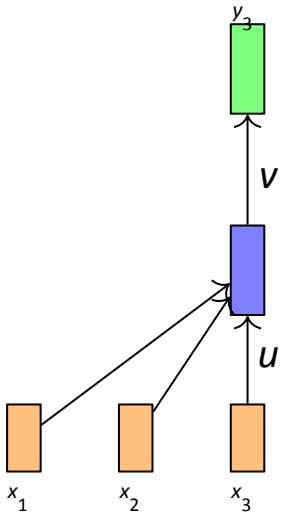
RNN

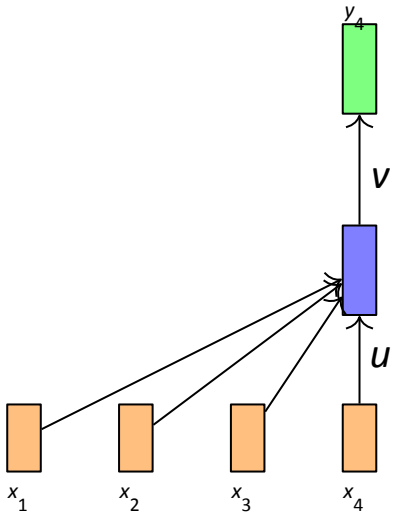


- This parameter sharing also ensures that the network becomes agnostic to the length (size) of the input.
- Since we are simply going to compute the same function (with same parameters) at each timestep, the number of timesteps doesn't matter.
- We just create multiple copies of the network and execute them at each timestamp.









How do we account for dependence between inputs ?

Let us first see an infeasible way of doing this

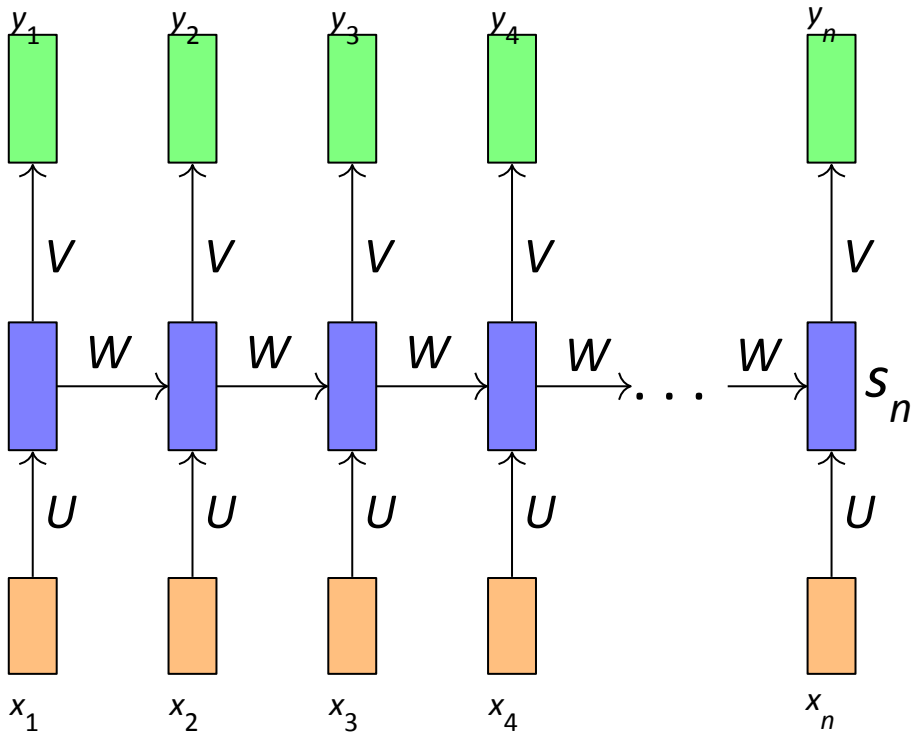
At each timestep we will feed all the previous inputs to the network

Is this okay ?

No, it violates the function

$$y_1 = f_1(x_1)$$
$$y_2 = f_2(x_1, x_2)$$
$$y_3 = f_3(x_1, x_2, x_3)$$

RNN



add a recurrent connection in the network

$$s_i = \sigma(Ux_i + Ws_{i-1} + b) \quad y_i = O(V s_i + c)$$

or

$$y_i = f(x_i, s_{i-1}, W, U, V, b, c)$$

- s_i is the state of the network at timestep i
- The parameters are W, U, V, c, b which are shared across timesteps
- The same network (and parameters) can be used to compute $y_1, y_2, \ldots, y_{10}$ or y_{100}

Recurrent Neural Networks

Human brain deals with information streams. Most data is obtained, processed, and generated sequentially.

- E.g., listening: soundwaves vocabularies/sentences
- E.g., action: brain signals/instructions sequential muscle movements

Human thoughts have persistence; humans don't start their thinking from scratch every second.

- As you read this sentence, you understand each word based on your prior knowledge.

The applications of standard Artificial Neural Networks (and also Convolutional Networks) are limited due to:

- They only accepted a fixed-size vector as input (e.g., an image) and produce a fixed-size vector as output (e.g., probabilities of different classes).
- These models use a fixed amount of computational steps (e.g. the number of layers in the model).

Recurrent Neural Networks (RNNs) are a family of neural networks introduced to **learn sequential data**.

- Inspired by the temporal-dependent and persistent human thoughts

Real-life Sequence Learning Applications

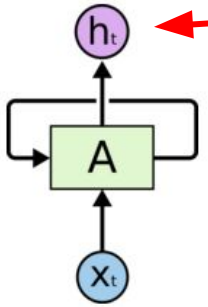
RNNs can be applied to various type of sequential data to learn the temporal patterns.

- Time-series data (e.g., stock price) □ Prediction, regression
- Raw sensor data (e.g., signal, voice, handwriting) □ Labels or text sequences
- Text □ Label (e.g., sentiment) or text sequence (e.g., translation, summary, answer)
- Image and video □ Text description (e.g., captions, scene interpretation)

Task	**Input**	**Output**
Activity Recognition (Zhu et al. 2018)	Sensor Signals	Activity Labels
Machine translation (Sutskever et al. 2014)	English text	French text
Question answering (Bordes et al. 2014)	Question	Answer
Speech recognition (Graves et al. 2013)	Voice	Text
Handwriting prediction (Graves 2013)	Handwriting	Text
Opinion mining (Irsoy et al. 2014)	Text	Opinion expression

Recurrent Neural Networks

Recurrent Neural Networks are networks with loops, allowing information to persist.



Recurrent Neural Networks have loops.

Output is to predict a vector h_t, where $output\ y_t = \varphi(h_t)$ at some time steps (t)

new state

old state

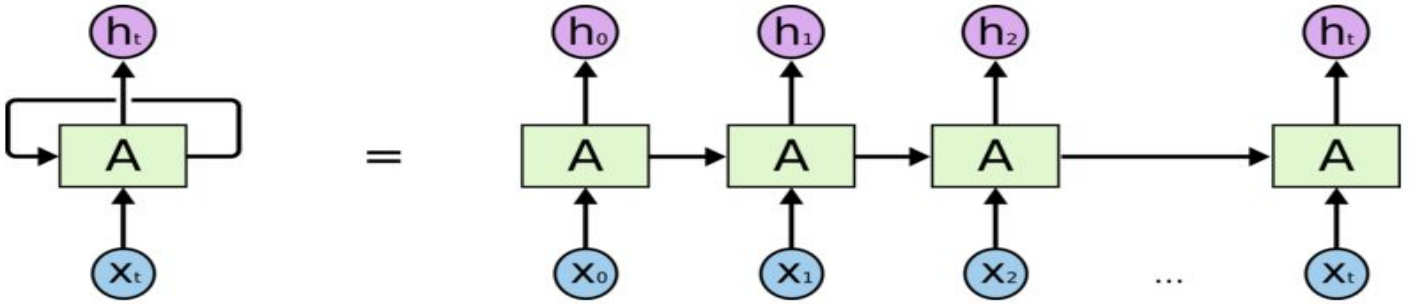
$$h_t = f_W(h_{t-1}, x_t)$$

function with parameter W

Input vector at some time step

Recurrent Neural Networks

Unrolling RNN



An unrolled recurrent neural network.

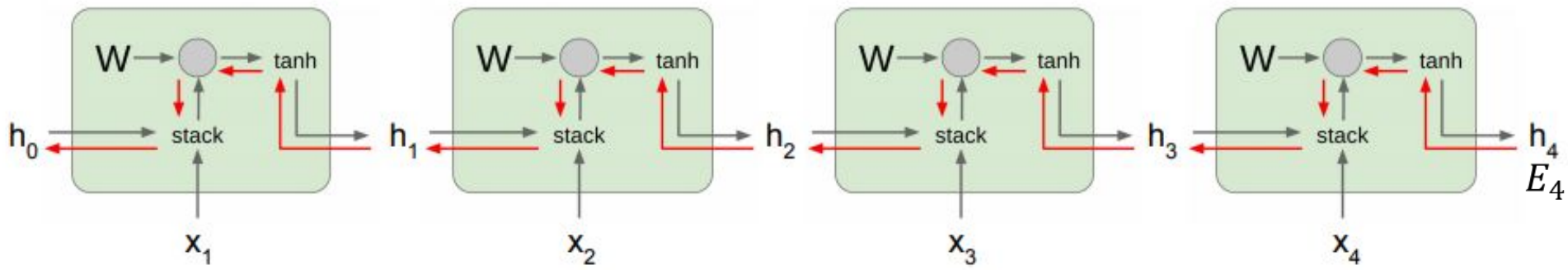
A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. The diagram above shows what happens if we **unroll the loop**.

Recurrent Neural Networks

The recurrent structure of RNNs enables the following characteristics:

- Specialized for processing a sequence of values $x^{(1)}, \dots, x^{(\tau)}$
  - Each value $x^{(i)}$ is processed with the **same network *A* that preserves past information**
- Can scale to much **longer sequences** than would be practical for networks without a recurrent structure
  - Reusing network ***A*** reduces the required amount of parameters in the network
- Can process **variable-length sequences**
  - The network complexity does not vary when the input length change
- However, vanilla RNNs suffer from the training difficulty due to **exploding and vanishing gradients**.

Exploding and Vanishing Gradients



$E_4 = J(y_4, \widehat{y_4})$

In vanilla RNNs, computing this gradient involves many factors of W_{hh} (and repeated tanh). If we decompose the singular values of the gradient multiplication matrix,

- Largest singular value > 1 → **Exploding gradients**
  - Slight error in the late time steps causes drastic updates in the early time steps → Unstable learning
- Largest singular value < 1 → **Vanishing gradients**
  - Gradients passed to the early time steps is close to 0. → Uninformed correction

Networks with Memory

Vanilla RNN operates in a “multiplicative” way (repeated tanh).

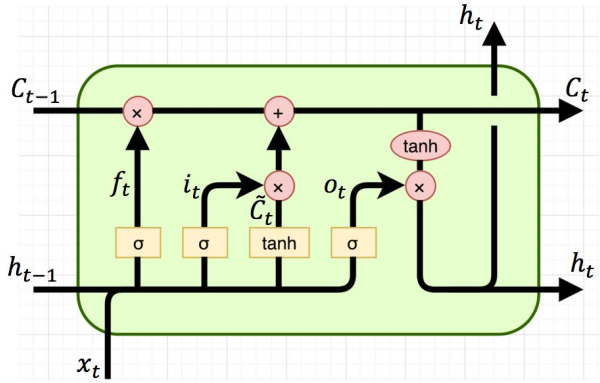
Two recurrent cell designs were proposed and widely adopted:

- **Long Short-Term Memory (LSTM)** (Hochreiter and Schmidhuber, 1997)
- Gated Recurrent Unit (GRU) (Cho et al. 2014)

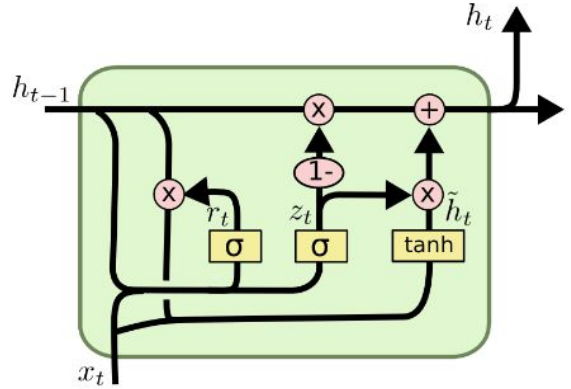
Both designs process information in an “additive” way with gates to control information flow.

- **Sigmoid gate outputs numbers between 0 and 1, describing how much of each component should be let through.**

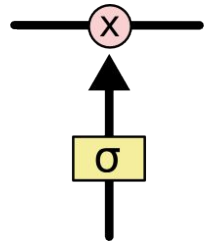
E.g. $f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) = Sigmoid\left(W_f x_t + U_t h_{t-1} + b_f\right)$



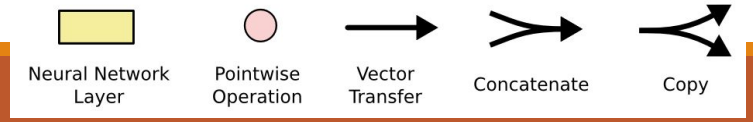
Standard LSTM Cell



GRU Cell



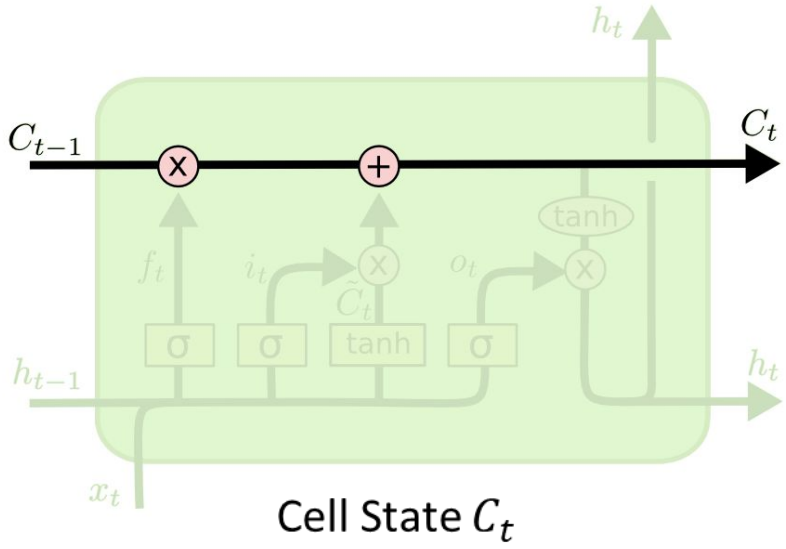
A Sigmoid Gate



Long Short-Term Memory (LSTM)

The key to LSTMs is the **cell state**.

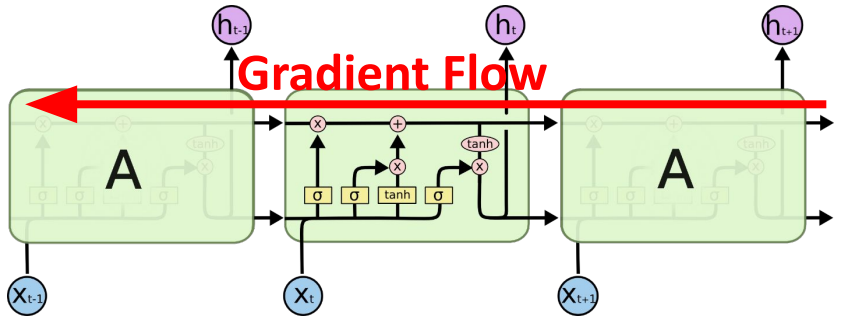
- **Stores information of the past □ long-term memory**
- **Passes along time steps with minor linear interactions □ "additive"**
- Results in an **uninterrupted gradient flow** □ errors in the past pertain and impact learning in the future



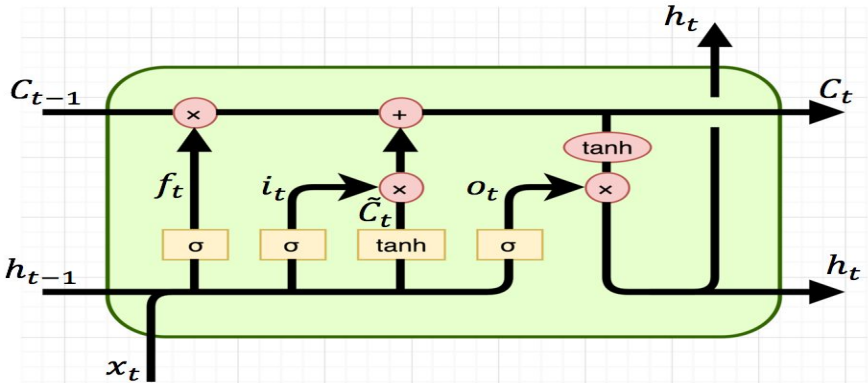
Cell State C_t

The LSTM cell manipulates input information with three gates.

- **Input gate** □ controls the intake of new information
- **Forget gate** □ determines what part of the cell state to be updated
- **Output gate** □ determines what part of the cell state to output



LSTM: Components & Flow



LSM unit output

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

Output gate units

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

Transformed memory cell contents

$$\tanh(C_t)$$

Gated update to memory cell units

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

Forget gate units

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

Input gate units

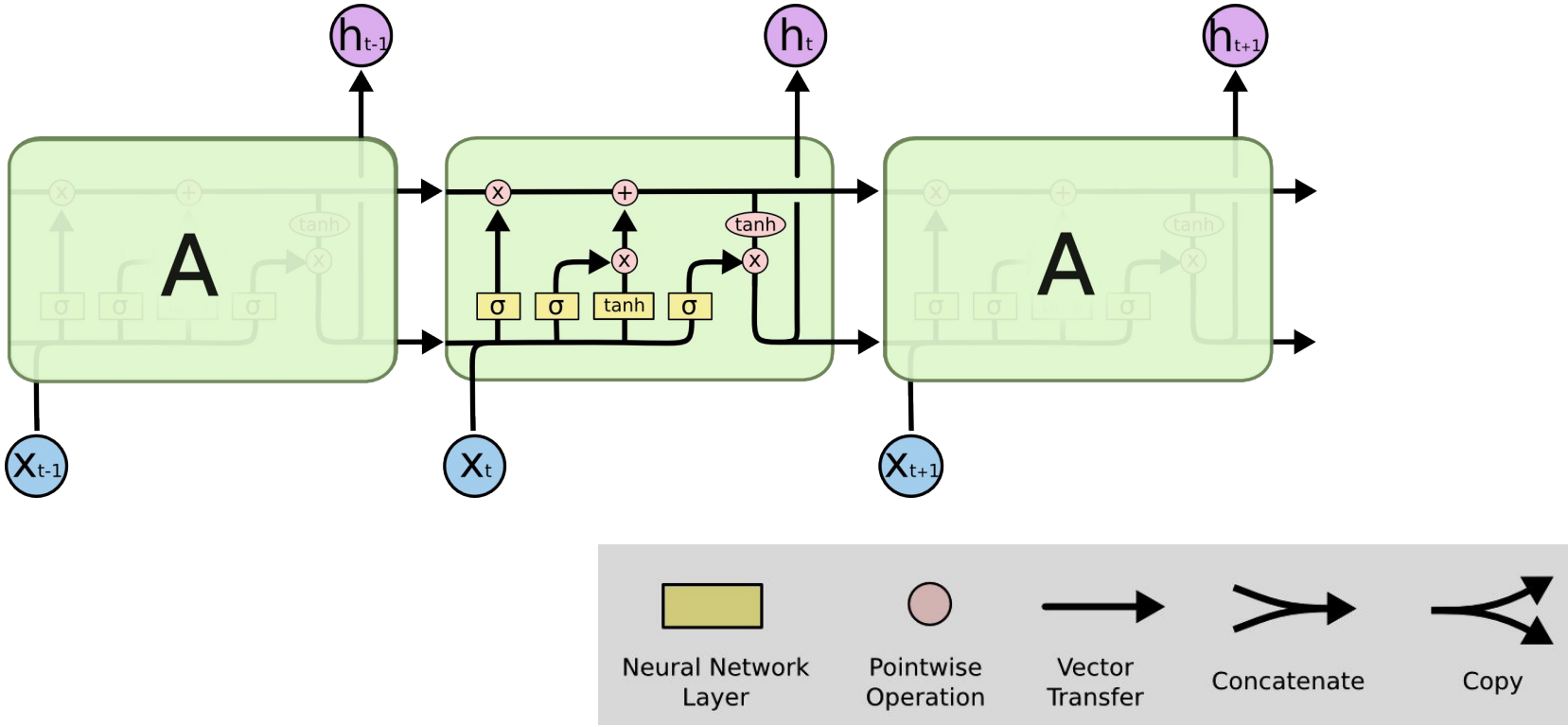
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$

Potential *input* to memory cell

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

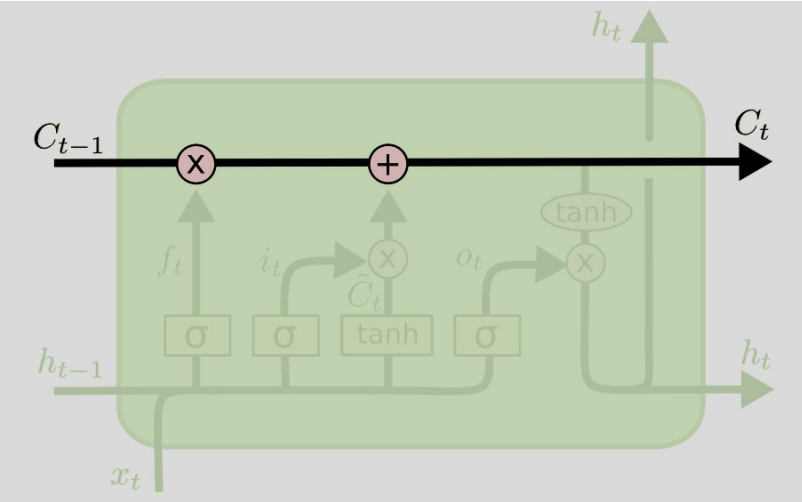
The long-short term memory (LSTM) module

LSTMs are explicitly designed to avoid the long-term dependency problem.

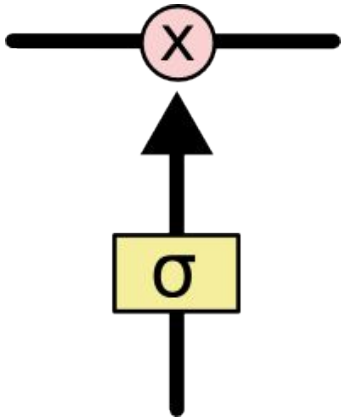


Two keys of LSTM

“Cell state” which works like a conveyor belt runs straight down the entire chain, easy for information to flow along without changes.
“Gates” which control or decide what kind of information could go or throw away from the cell state.



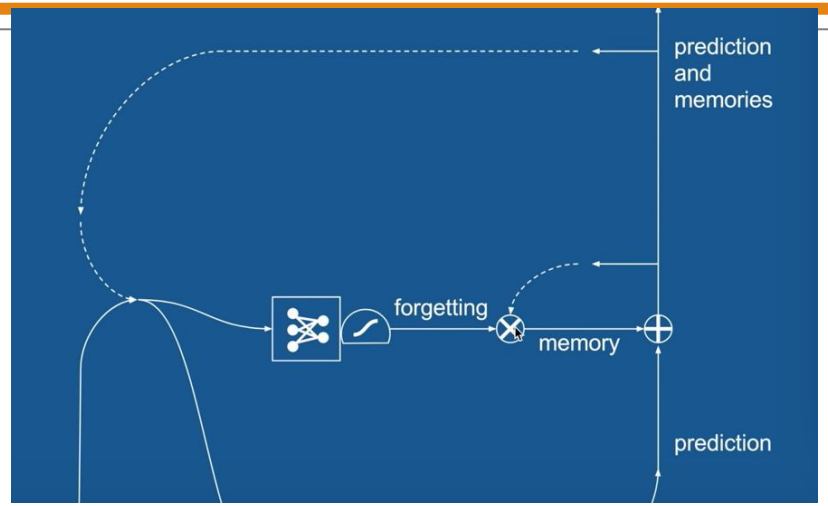
Cell state

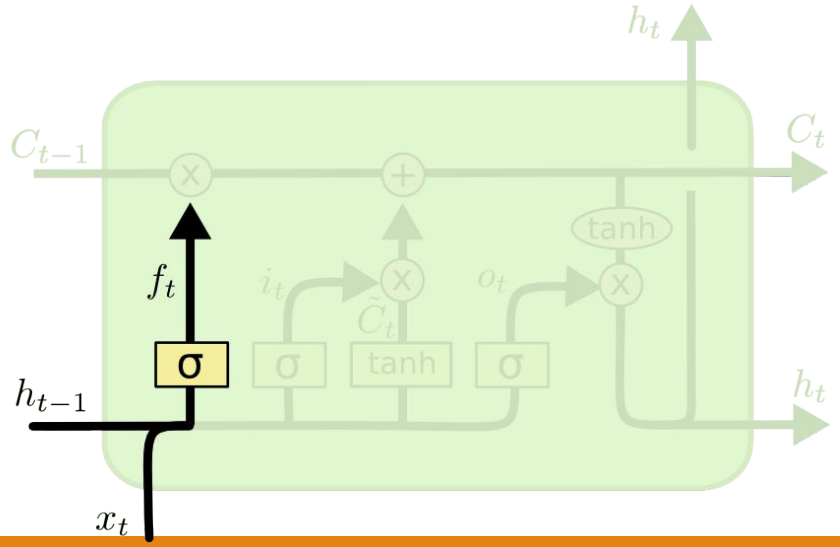


Gate

LSTM – forget gate

Take the example of a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.

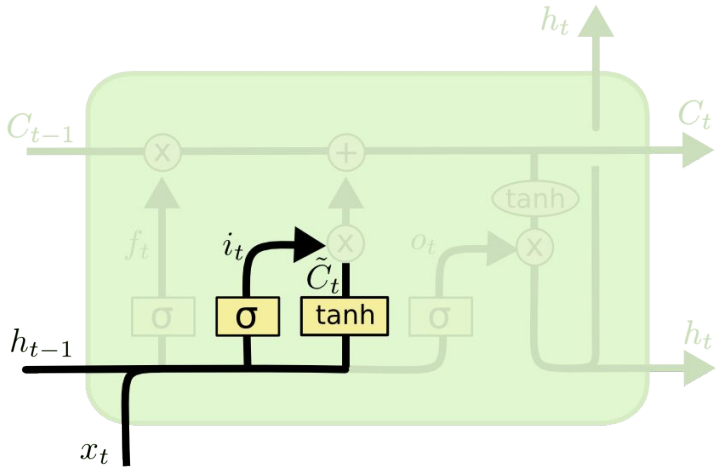




$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

LSTM – input gate

It decides what new information we're going to store in the cell state. It has two parts. First, a sigmoid layer called the "input gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state. In the next step, we'll combine these two to create an update to the state.

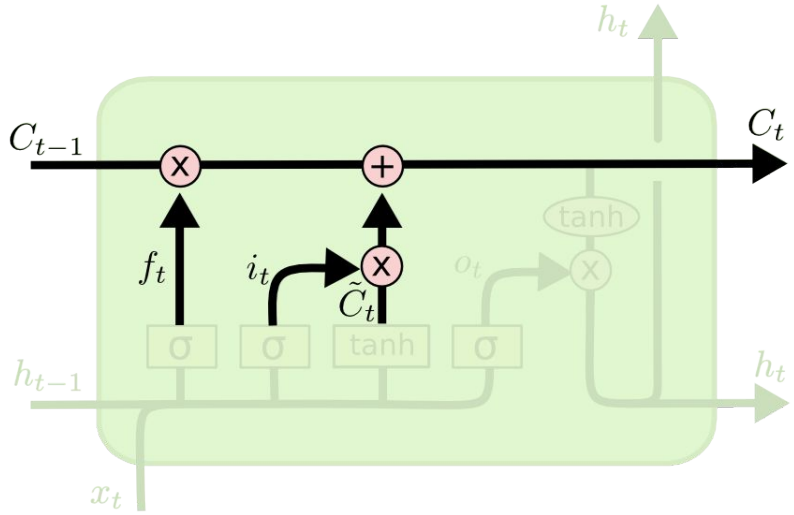


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

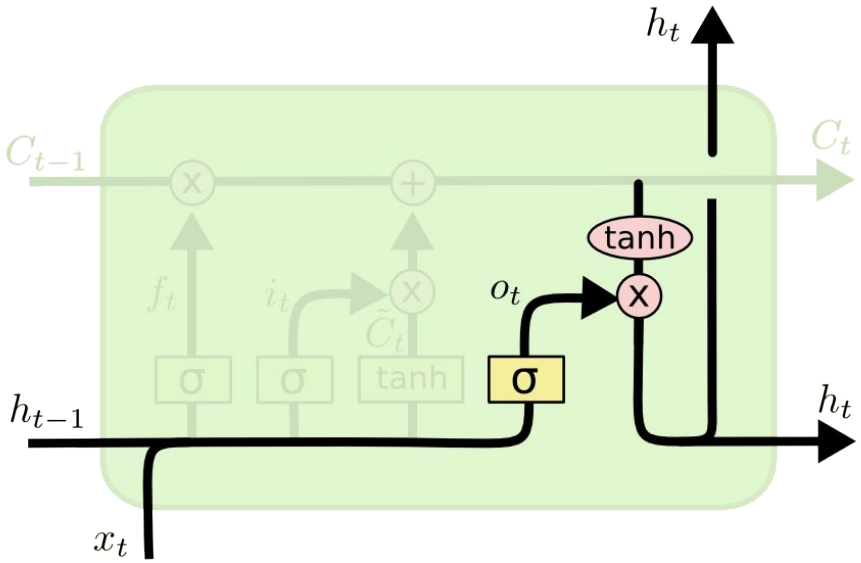
LSTM – cell state update

It actually drops the information about the old subject's gender and add the new information, as we decided in the previous steps.



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

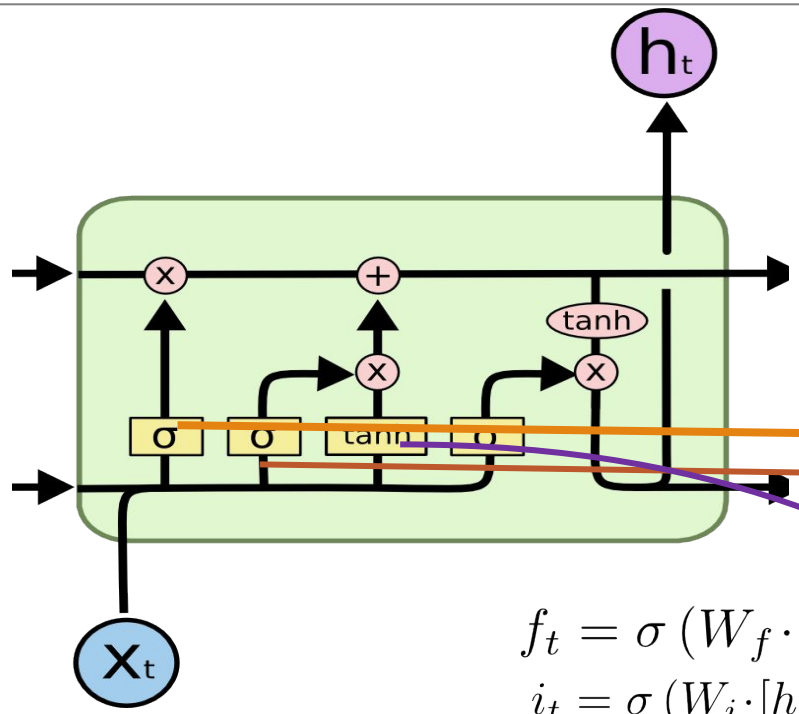
LSTM – output gate



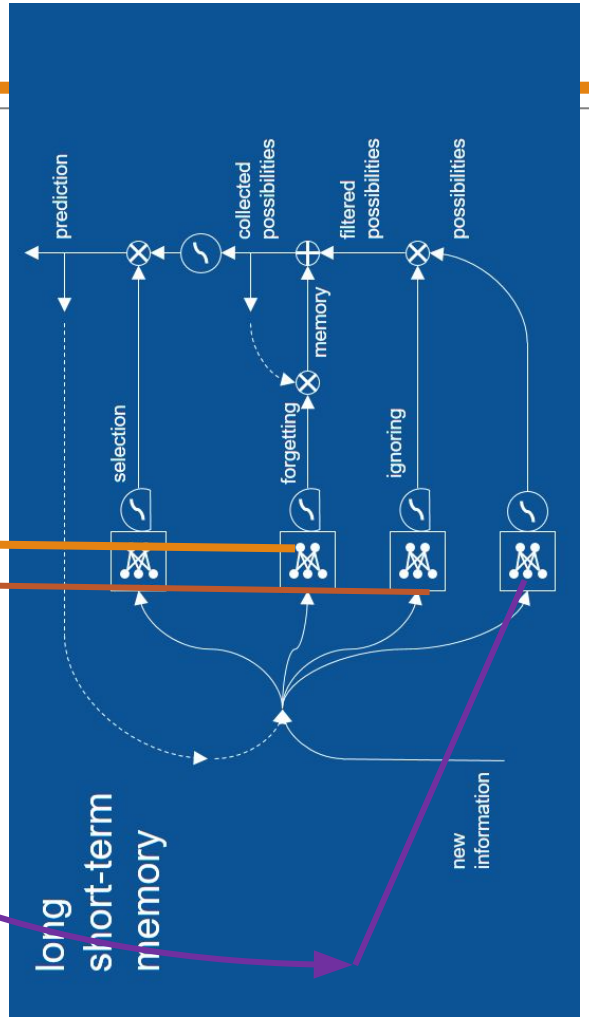
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

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

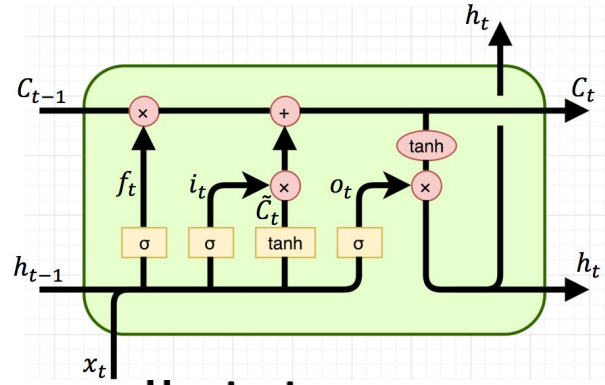
LSTM - revisit



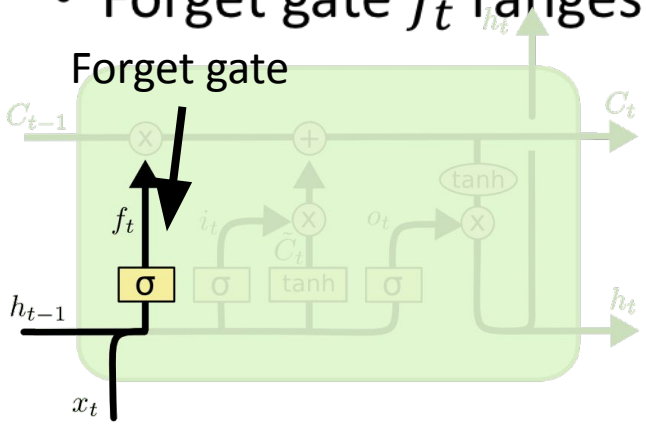
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$



Step-by-step LSTM Walk Throug



- **Step 1**: Decide what information to throw away from the cell state (memory) ➔ $f_t * C_{t-1}$
  - The output of the previous state h_{t-1} and the new information x_t jointly determine what to forget
    - h_{t-1} contains selected features from the memory C_{t-1}
  - Forget gate f_t ranges between $[0, 1]$

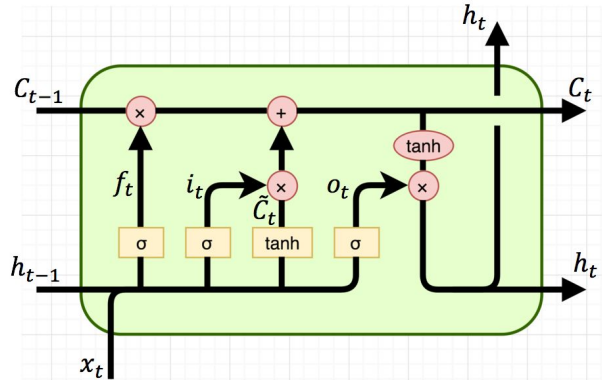


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

Neural Network Layer | Pointwise Operation | Vector Transfer | Concatenate | Copy

Text processing example:
Cell state may include the gender of the current subject (h_{t-1}). When the model observes a new subject (x_t), it may want to forget ($f_t \rightarrow 0$) the old subject in the memory (C_{t-1}).

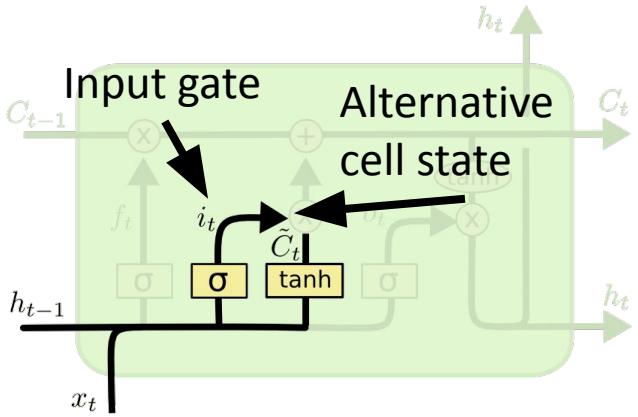
Step-by-step LSTM Walk Throug



- **Step 2**: Prepare the updates for the cell state

  from input ➔ $i_t * \tilde{C}_t$

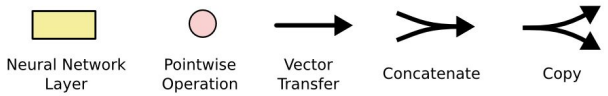
  - An alternative cell state $\tilde{C}_t$ is created from the new information x_t with the guidance of h_{t-1}.
  - Input gate i_t ranges between $[0, 1]$



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

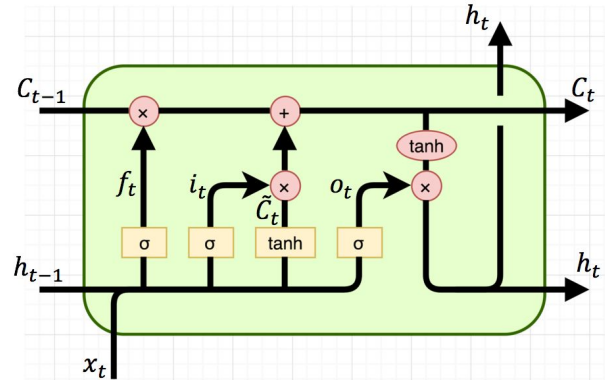
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

$$C_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$



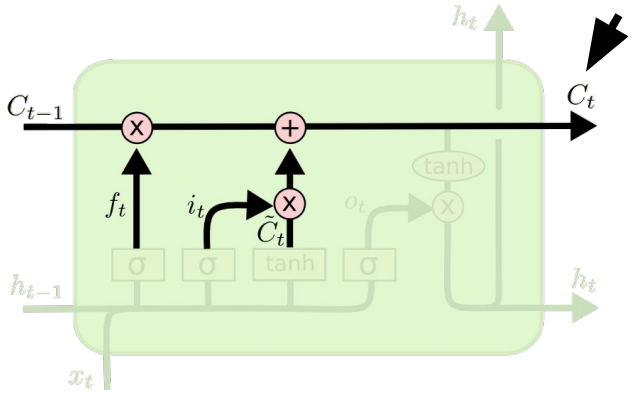
Example:
The model may want to add ($i_t \rightarrow 1$) the gender of new subject ($\tilde{C}_t$) to the cell state to replace the old one it is forgetting.

Step-by-step LSTM Walk Throug

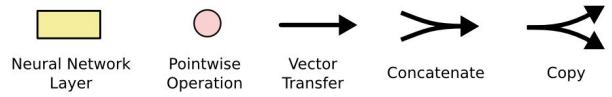


- **Step 3**: Update the cell state ➔ $f_t * C_{t-1} + i_t * \tilde{C}_t$
  - The new cell state C_t is comprised of information from the past $f_t * C_{t-1}$ and valuable new information $i_t * \widetilde{C}_t$
  - $*$ denotes elementwise multiplication

New cell state



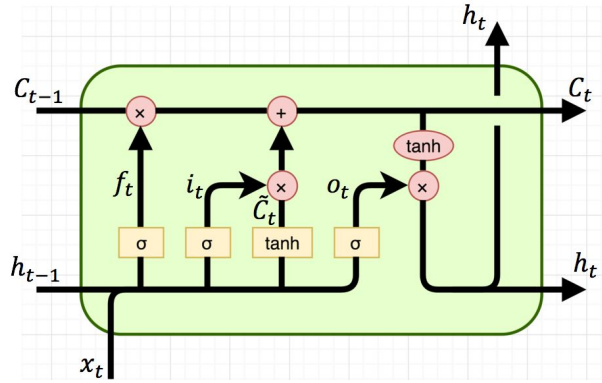
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



Example:

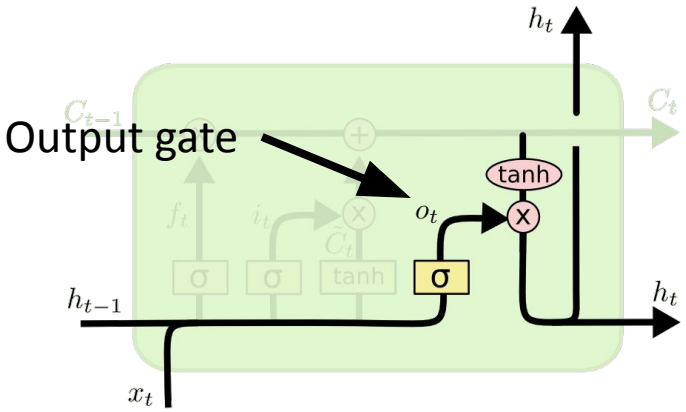
The model drops the old gender information ($f_t * C_{t-1}$) and adds new gender information ($i_t * \widetilde{C}_t$) to form the new cell state (C_t).

Step-by-step LSTM Walk Throug



Step 4: Decide the filtered output from the new cell state ➔ $o_t * \tanh(C_t)$

- tanh function filters the new cell state to characterize stored information
  - Significant information in C_t → ± 1
  - Minor details → 0
- Output gate o_t ranges between $[0, 1]$
- h_t serves as a control signal for the next time step



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

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

- **Step 4**: Decide the filtered output from the new cell state ➔
- tanh function filters the new cell state to characterize stored information
  - Significant information in C_t → ± 1
  - Minor details → 0
- Output gate o_t ranges between $[0, 1]$
- h_t serves as a control signal for the next time step

Example:
Since the model just saw a new subject (x_t), it might want to output ($o_t \rightarrow 1$) information relevant to a verb ($\tanh(C_t)$), e.g., singular/plural, in case a verb comes next.

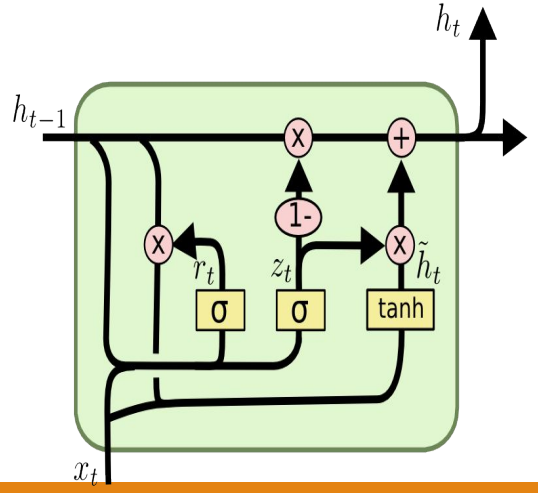
Gated Recurrent Unit (GRU)

GRU is a variation of LSTM that also adopts the gated design.

Differences:

- GRU uses an **update gate z** to substitute the input and forget gates i_t and f_t
- Combined the cell state C_t and hidden state h_t in LSTM as a single cell state h_t

GRU obtains similar performance compared to LSTM with fewer parameters and faster convergence. (Cho et al. 2014)



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

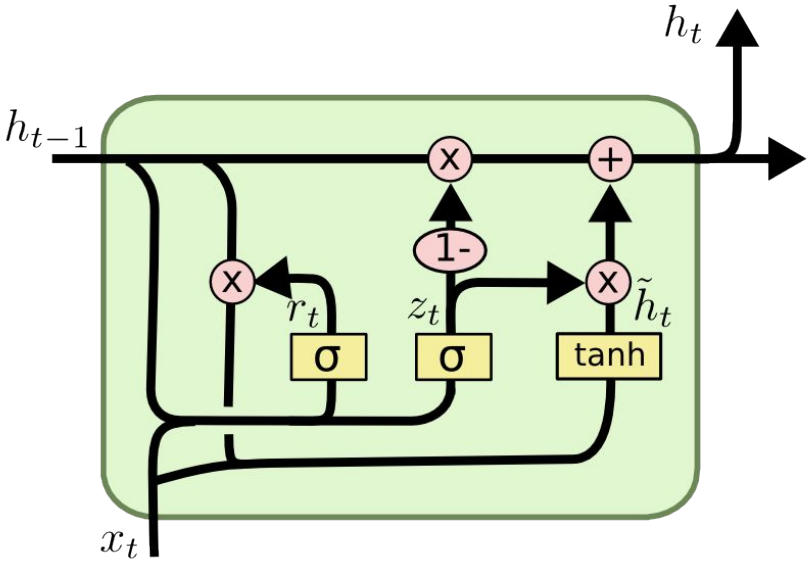
Update gate: controls the composition of the new state

Reset gate: determines how much old information is needed in the alternative state $\tilde{h}_t$

Alternative state: contains new information

New state: replace selected old information with new information in the new state

The gated recurrent units (GRUs) module



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

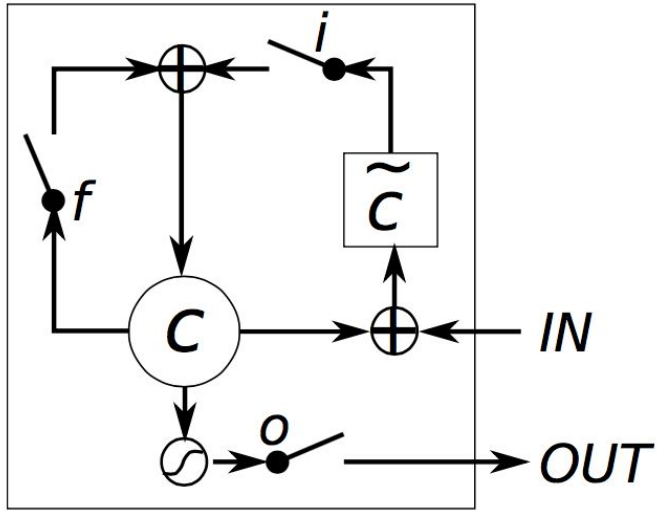
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

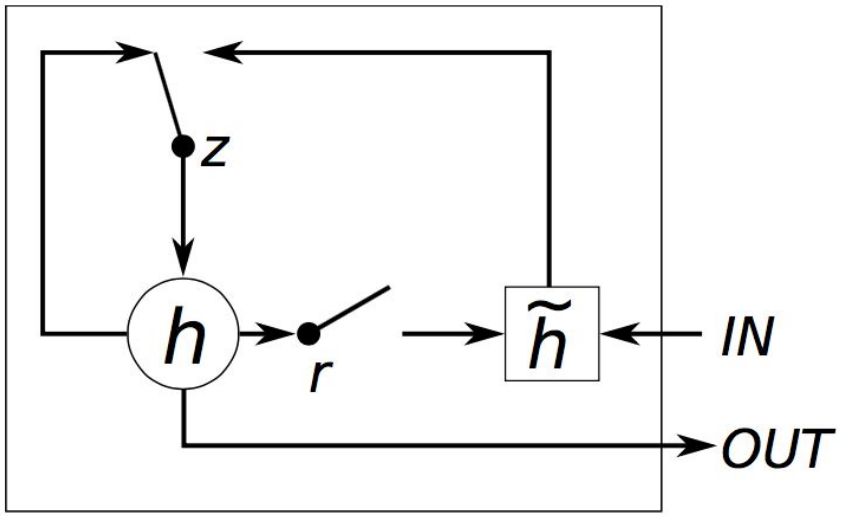
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

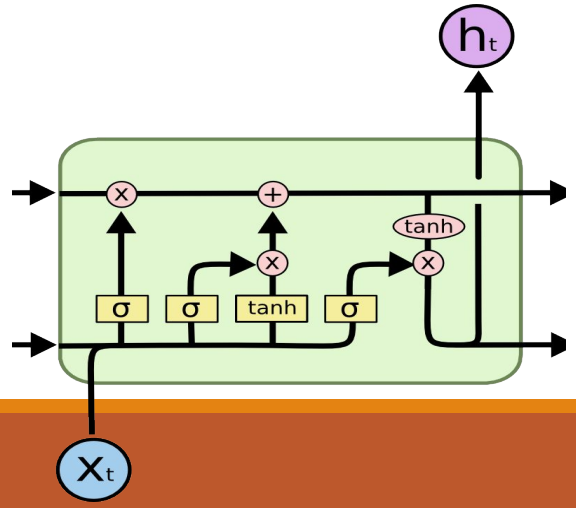
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

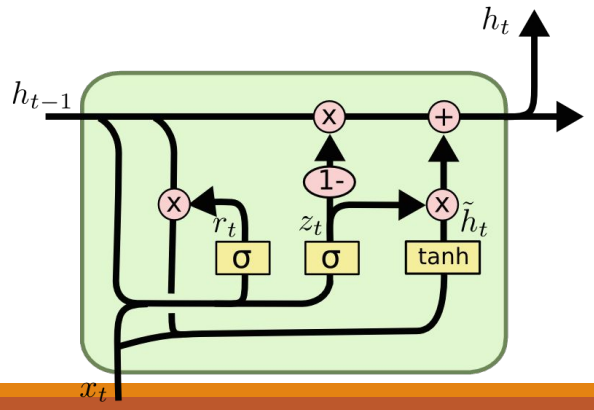
Similar with LSTM but with only two gates and less parameters.
The "update gate" determines how much of previous memory to be kept.
The "reset gate" determines how to combine the new input with the previous memory.

Comparison of the gating mechanism

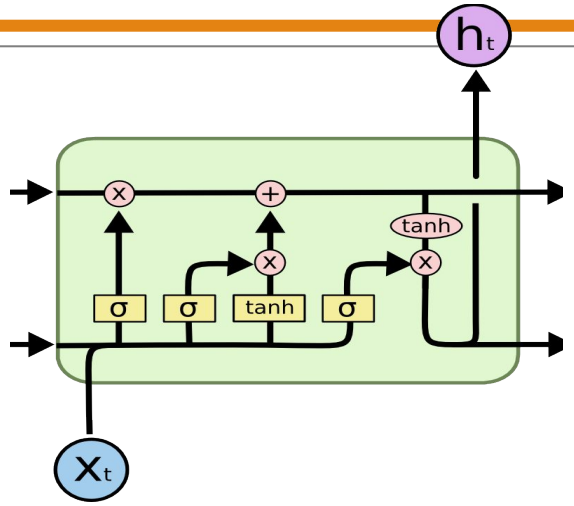


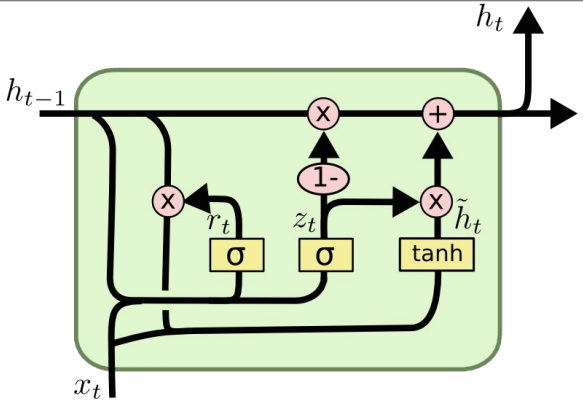






LSTM vs. GRU





$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Sequence Learning Architectures

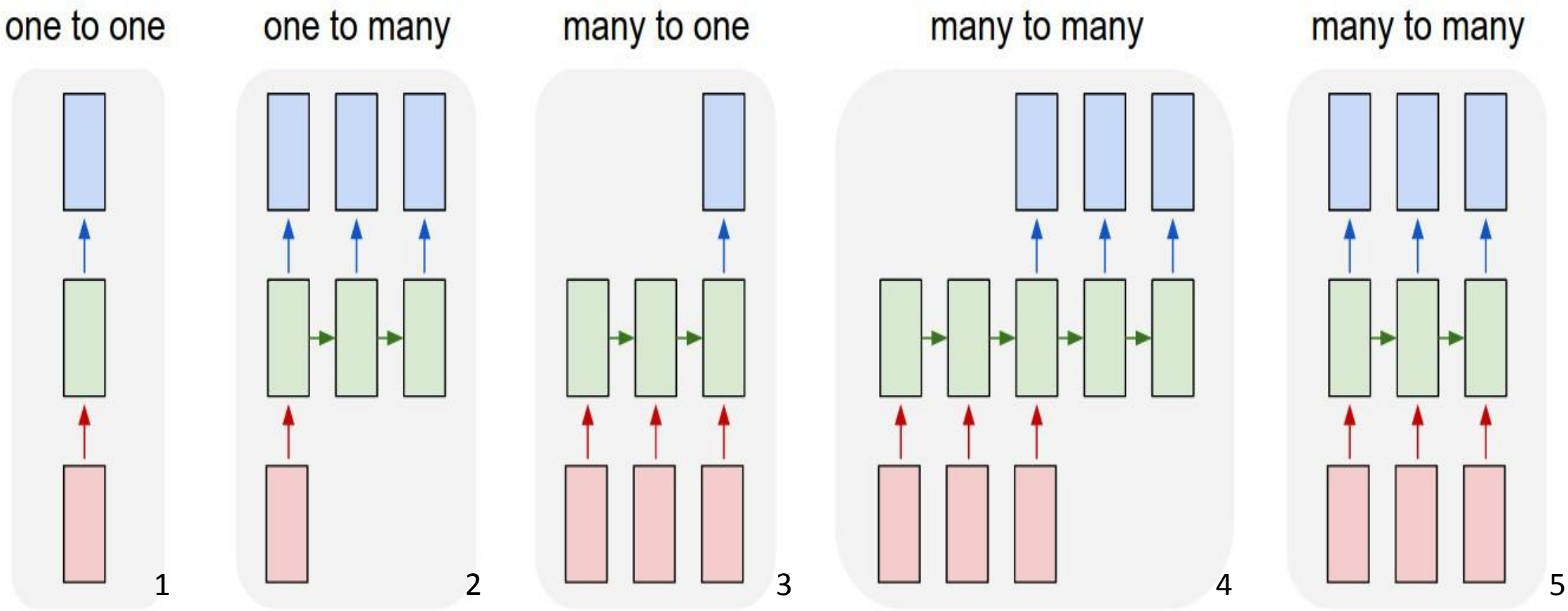
Learning on RNN is more robust when the vanishing/exploding gradient problem is resolved.

- RNNs can now be applied to different Sequence Learning tasks.

Recurrent NN architecture is flexible to operate over various sequences of vectors.

- Sequence in the input, the output, or in the most general case both
- Architecture with one or more RNN layers

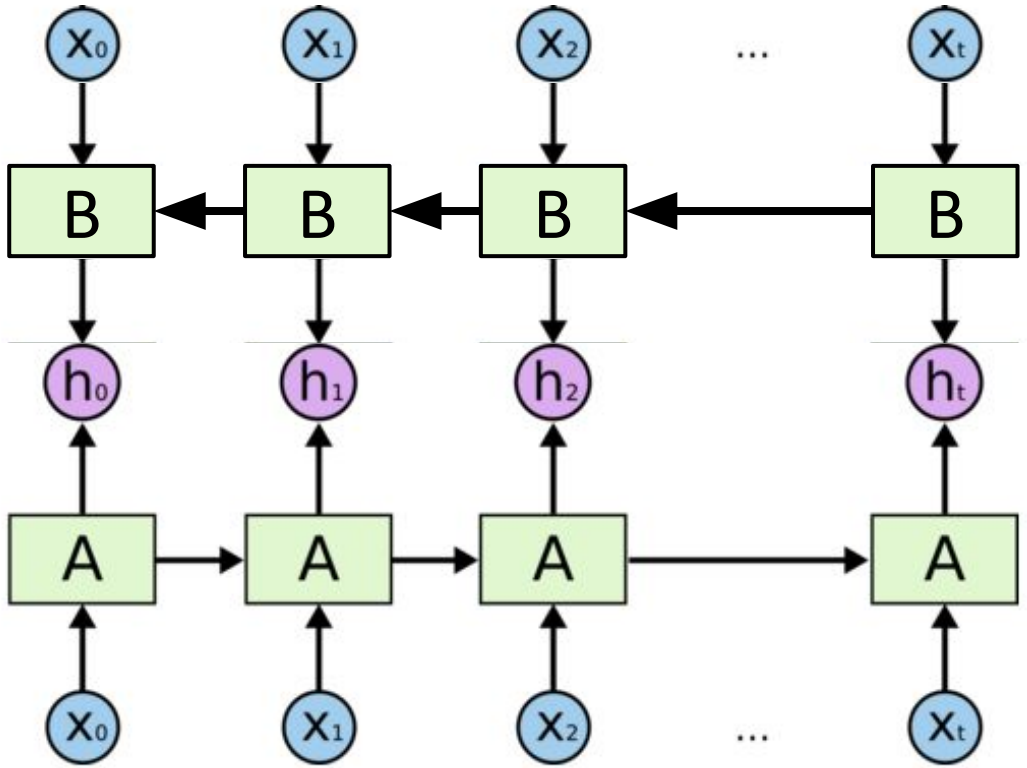
Sequence Learning with One RNN Layer



Sequence Learning with Multiple RNN Layers

Bidirectional RNN

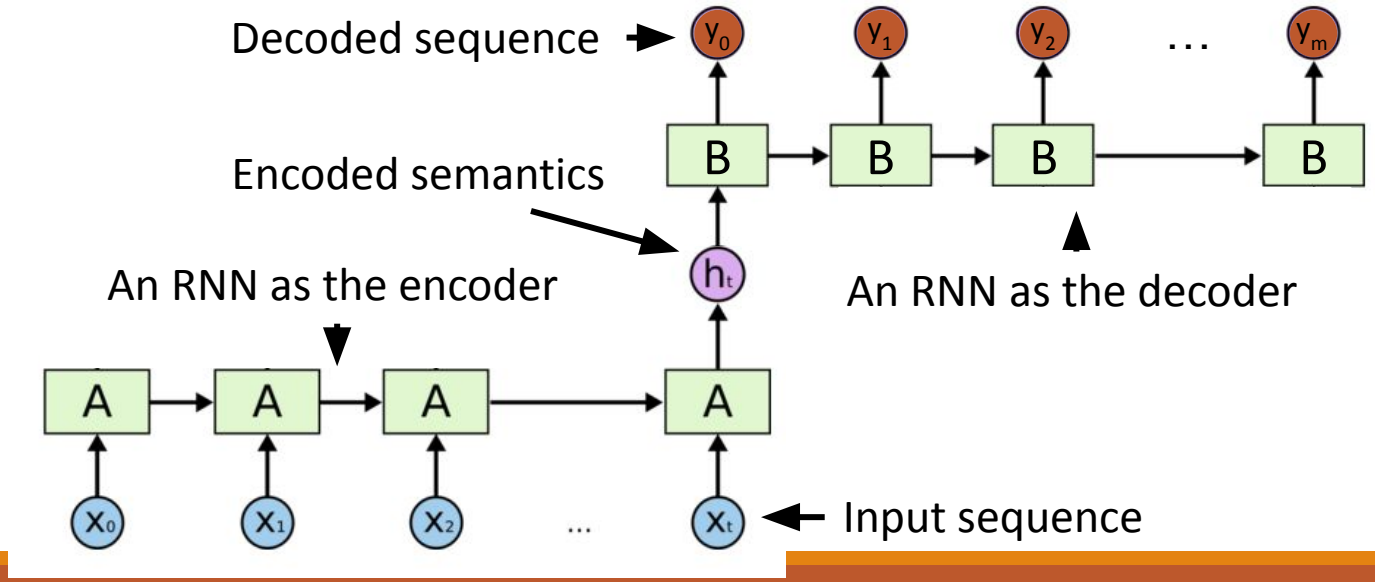
- Connects two recurrent units (synced many-to-many model) of opposite directions to the same output.
- Captures forward and backward information from the input sequence
- Apply to data whose current state (e.g., h_0) can be better determined when given future information (e.g., $x_1, x_2, \ldots, x_t$)
  - E.g., in the sentence "the bank is robbed," the semantics of "bank" can be determined given the verb "robbed."



Sequence Learning with Multiple RNN Layers

Sequence-to-Sequence (Seq2Seq) model

- Developed by Google in 2018 for use in machine translation.
- Seq2seq turns one sequence into another sequence. It does so by use of a recurrent neural network (RNN) or more often LSTM or GRU to avoid the problem of vanishing gradient.
- The primary components are one Encoder and one Decoder network. The encoder turns each item into a corresponding hidden vector containing the item and its context. The decoder reverses the process, turning the vector into an output item, using the previous output as the input context.
- **Encoder RNN**: extract and compress the semantics from the input sequence
- **Decoder RNN**: generate a sequence based on the input semantics
- Apply to tasks such as machine translation
  - Similar underlying semantics
  - E.g., “I love you.” to “Je t’aime.”



Few Studies and Publications

The Author(s) *BMC Medical Informatics and Decision Making* 2017, **17**(Suppl 2):67

BMC Medical Informatics and Making

W&C Track Volume 56

Journal of Medical Internet Research ISSN 1438-8871

JMIR MEDICAL INFORMATICS

Alfattni et al

J Med Internet Res. 2020 Se

Published online 2020 Sep 2

Marrying Medical Electronic Health

Monitoring Editor: Gunther E

Reviewed by Fenglong Ma a

School of Biomedical Informatics, The U

Corresponding Author: Stephen Wu, P
ter at Houston, 7000 Fannin St, Suite 6

Received 3 July 2019; Revised 15 October 2019

Original Paper

Extracting Drug Names and Associated Attributes From Discharge Summaries: Text Mining Study

Ghada Alfattni[1,2], BSc (Hons), MSc; Maksim Belousov[1], BSc (Hons), PhD; Niels Peek[3,4,5], PhD; Goran Nenadic[1,5], BSc (Hons), MSc, PhD

[1]Department of Computer Science, University of Manchester, Manchester, United Kingdom

[2]Department of Computer Science, Jamoum University College, Umm Al-Qura University, Makkah, Saudi Arabia

[3]Centre for Health Informatics, Division of Informatics, Imaging and Data Sciences, University of Manchester, Manchester, United Kingdom

[4]National Institute of Health Research Manchester Biomedical Research Centre, Manchester Academic Health Science Centre, University of Manchester, Manchester, United Kingdom

[5]The Alan Turing Institute, Manchester, United Kingdom

Summary

LSTM and GRU are RNNs that retain past information and update with a gated design.

- The “additive” structure avoids vanishing gradient problem.

RNNs allow flexible architecture designs to adapt to different sequence learning requirements.

RNNs have broad real-life applications.

- Text processing, machine translation, signal extraction/recognition, image captioning
- Mobile health analytics, activity of daily living, senior care

Thank you

QUESTION??

Sequential data