Recurrent Neural network

Sequence model

- Sequence models are the machine learning models that input or(and) output sequences of data.
- Examples: Speech recognition

  image captioning

  sentiment classification

  Language translation

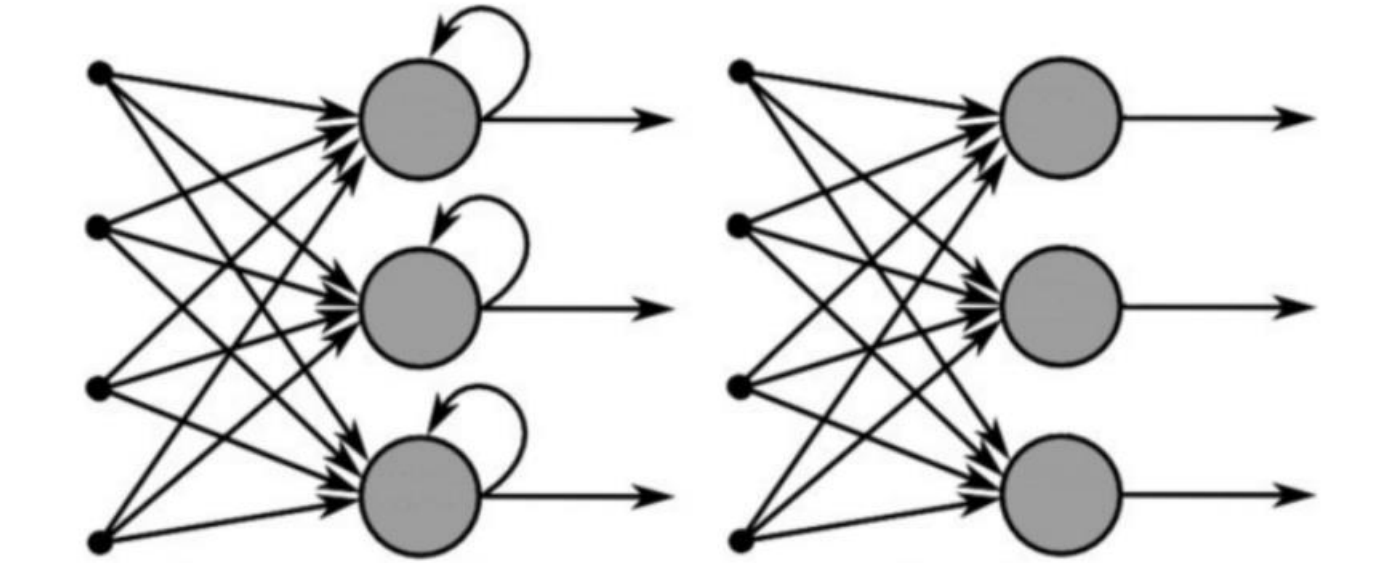
  stock market prediction

  music generation

Why we need RNN ?

- Traditional ANN can not process sequence data because of different sizes in input/output neurons.
- Sequence data processing involves too much computation which ANN can not handle.
- **Parameter sharing** is not possible in ANN.
- ANN does not have internal memory to keep track of previous inputs.

RNN vs ANN



Recurrent Neural Network

Feed-Forward Neural Network

RNN

A recurrent neural network (RNN) is a type of artificial neural network which uses sequential data or time series data.
Extensively used in Language translation, natural language processing (nlp), speech recognition, and image captioning ,etc

In a RNN output of the current layer is the input of the next layer.
In traditional neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words(data).

In a RNN same weights and bias are used at each time step. Because it performs the same task on all the inputs of hidden layer to produce the output.
The main and most important feature of RNN is Hidden state, which remembers some information which gets used in the next layer.

Building block of RNN

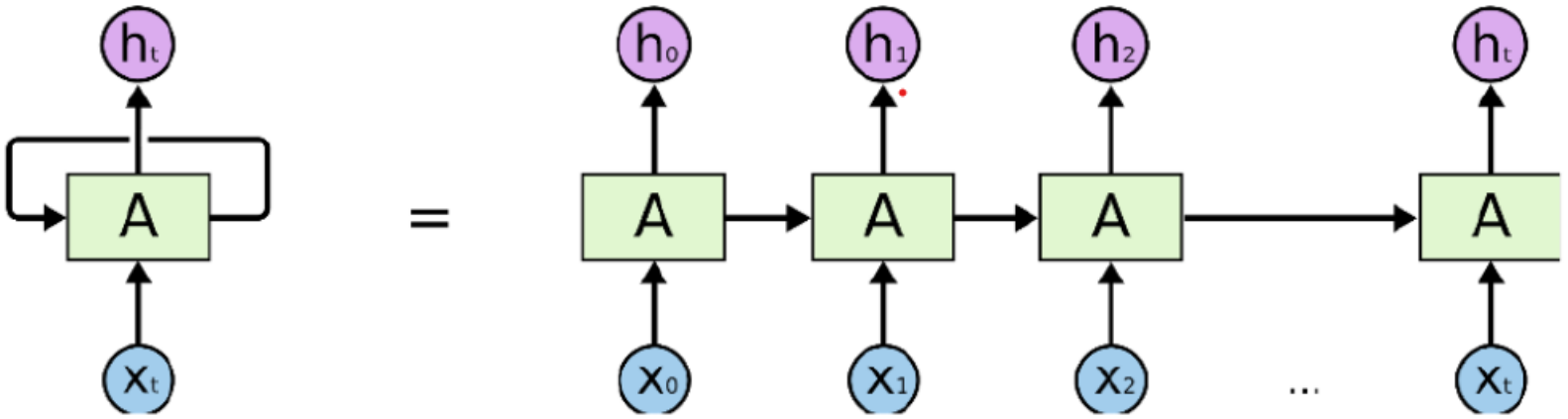
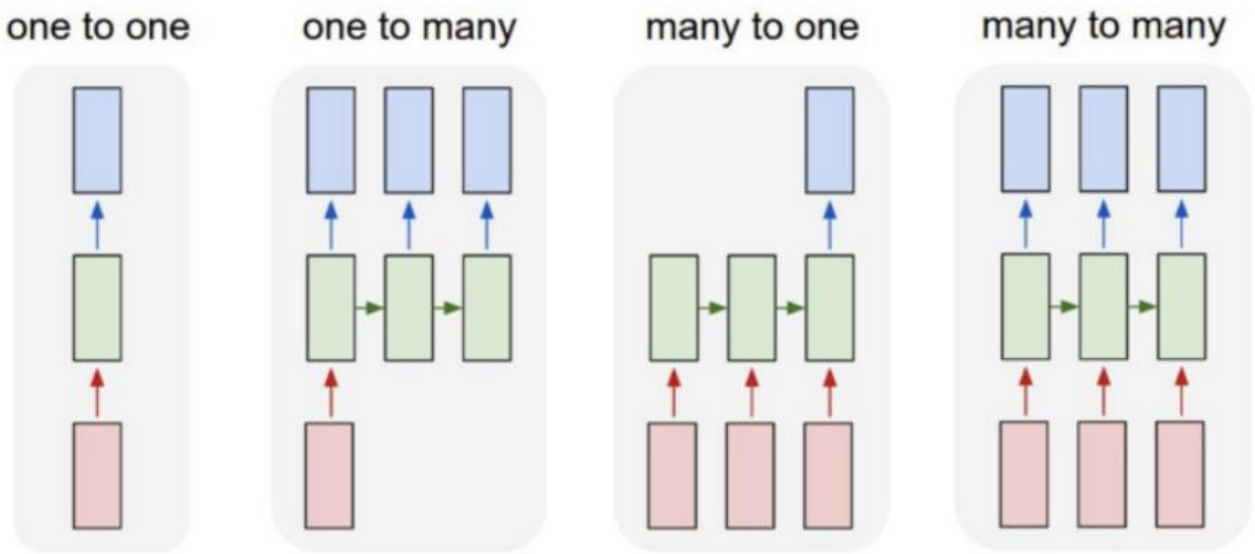
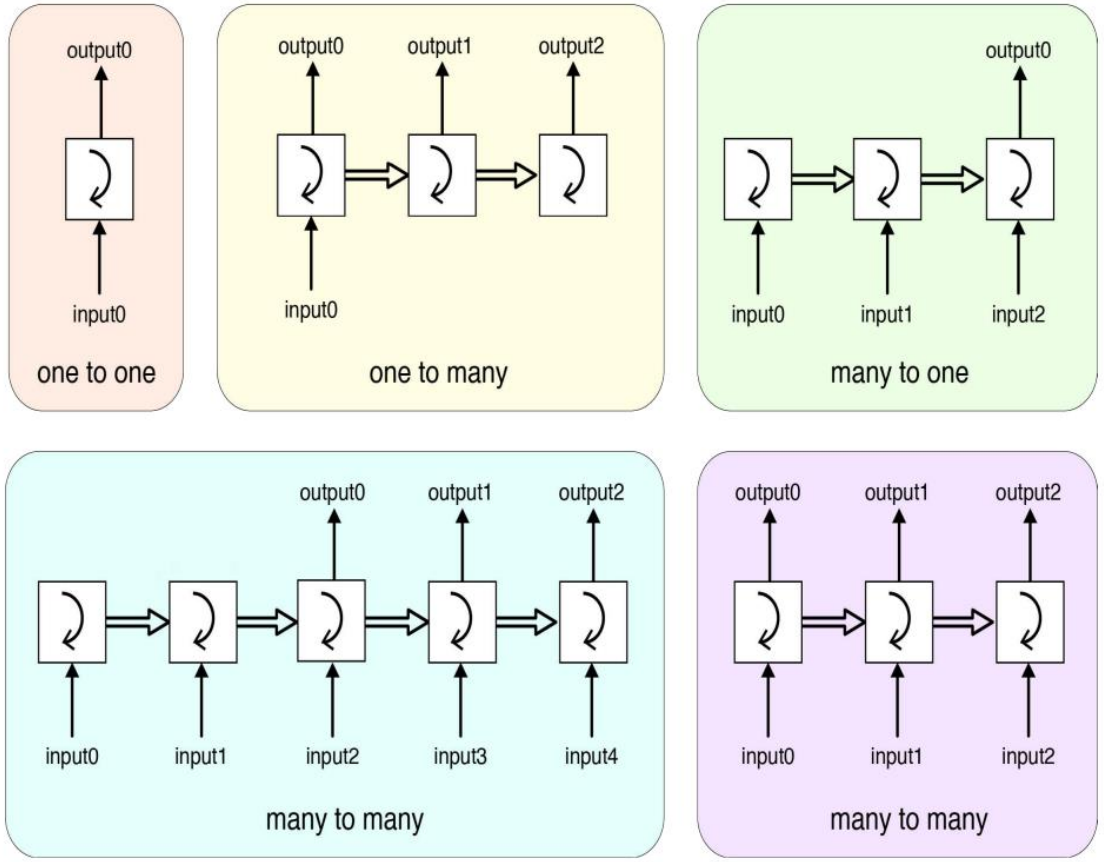


Fig-1:Building Blocks of RNN

- RNN uses a loop to remember the relation among all the previous outputs to predict the next output.
- So first X(0) is the input and h(0) is the output. And also h(0) is input for the next step where the already existing input is x(1). Similarly output of this layer which is h(1), and the x(2) will be the input for the next step. This way it keeps remembering the context in training time
- Here A is hidden layer activation function. Activation function takes decision for the output to be qualified or not for our result .Activation function is used based on my desired output pattern or type

Types of RNN


one to one
one to many
many to one
many to many


output0
input0
one to one
output0
output1
output2
input0
one to many
output0
input0
input1
input2
many to one
output0
output1
output2
input0
input1
input2
input3
input4
many to many
output0
output1
output2
input0
input1
input2
many to many

- **One to one structure**: We give it a single input (that is, a feature with a single time step) and it produces a single output.
  - This is a waste of RNN. Why? because with a sequence length of 1, the RNN cell is making no use of its unique ability to remember things about its input sequence.
- **One to many structure:** Takes in a single piece of data and produces a sequence.
  - We do this by giving the RNN the information we have to get it started, and then we let it run for a while, producing multiple pieces of output.
  - **Example**: Give it the starting note for a song, and the network produces the rest of the melody.
- **Many to one structure:** Reads in a sequence and gives us back a single value.
  - This organization is used frequently in the field of **sentiment analysis**, where the network is given a piece of text and then reports on some quality inherent in the writing.
  - **Example:** A common example is to look at a movie review and determine if it was positive or negative.

- **Many to many structures**: Most interesting. Two cases
  - Bottom left of Fig.: We "prime" the RNN with several pieces of input before we ask it to start producing outputs. So, output is delayed.
  - **Example**: Machine translation. In some languages words don't come in the same order, so we can't start translating right away.
  - The English sentence "The black dog slept in the hot sun" can be expressed in French as "Le chien noir dormait dans le soleil chaud."
  - In French, the adjective "noir" (black) follows the noun "chien" (dog), so we need to have some kind of buffer so we can produce the words in their proper English order.
  - Bottom right of Fig. : We start producing outputs right away. So, each new input produces a corresponding new output.
  - **Example**: A description for every frame of a video.
  - Suppose we have a piece of video of a ball flying through the air.
  - We'd like to assign a label to every frame.

Classical Dynamical Systems

$$s^t = f(s^{(t-1)}; \theta) \quad \textit{A system is recurrent : } s^t \textit{ is a function of } s^{t-1}$$

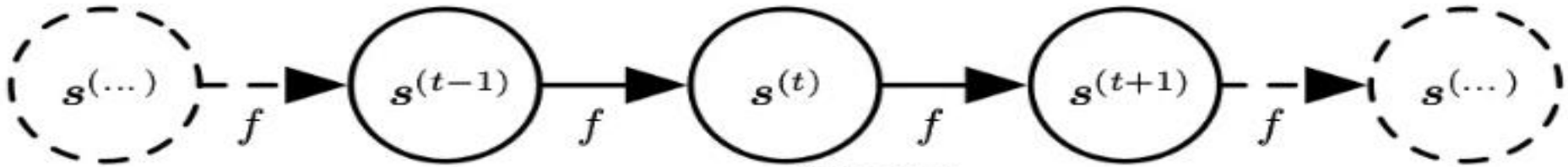


Figure 10.1

$$h^t = f(h^{(t-1)}; x^t; \theta)$$

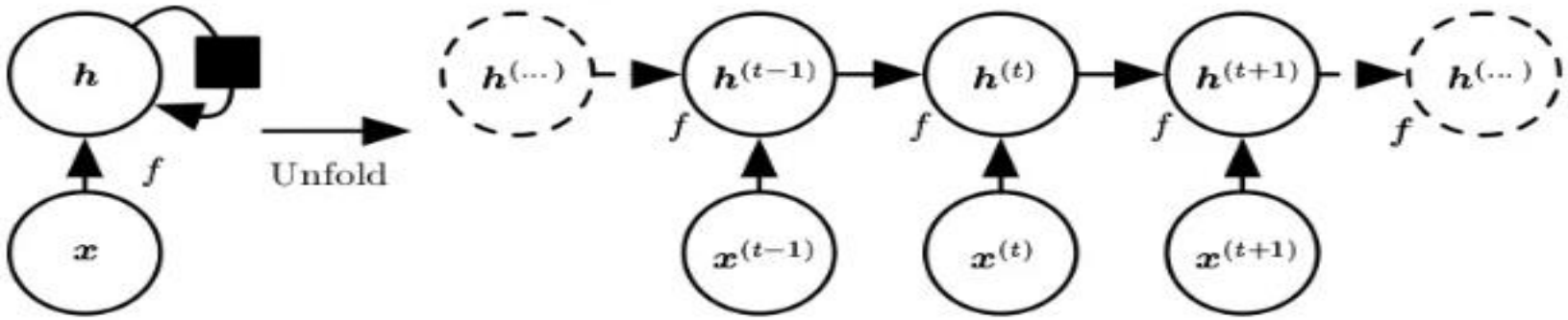
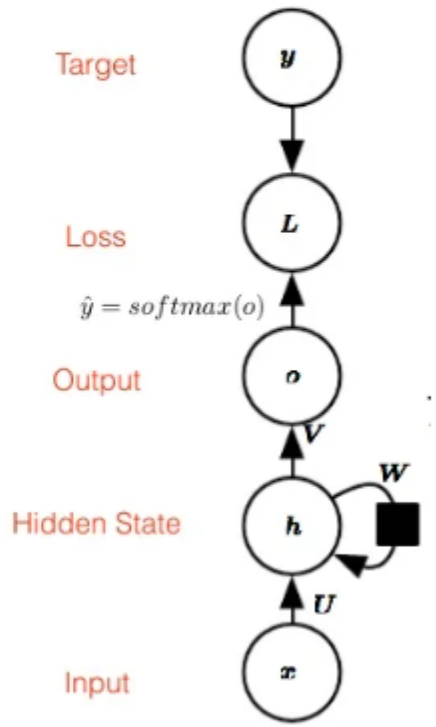


Figure 10.2

Recurrent Hidden Units

Basic Form



For Every Time Step!!

Recurrent Hidden Units

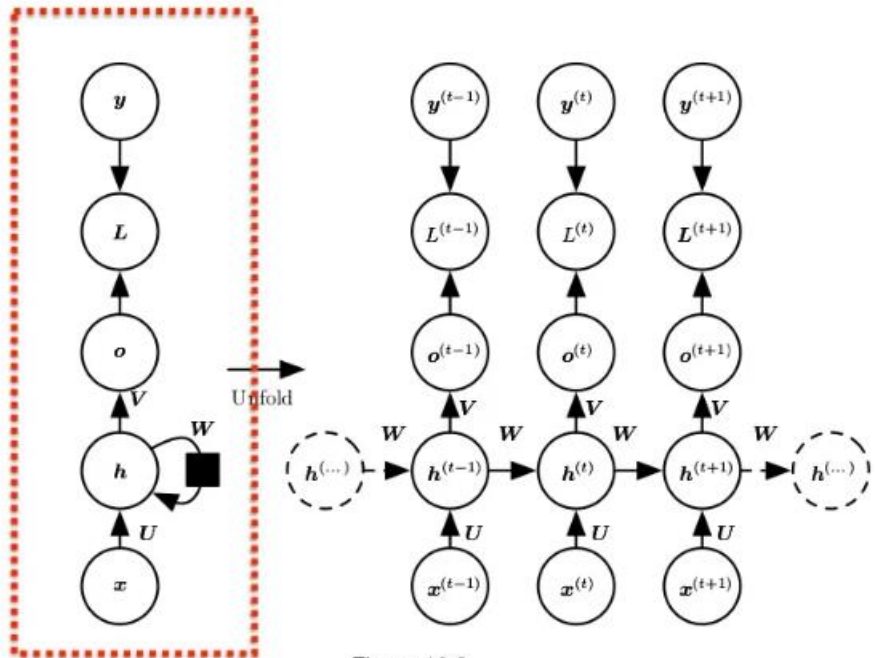


Figure 10.3

Backpropagation Through time(BPTT)

- Regular (feedforward) backprop applied to RNN unfolded in time
- Truncated BPTT approximation

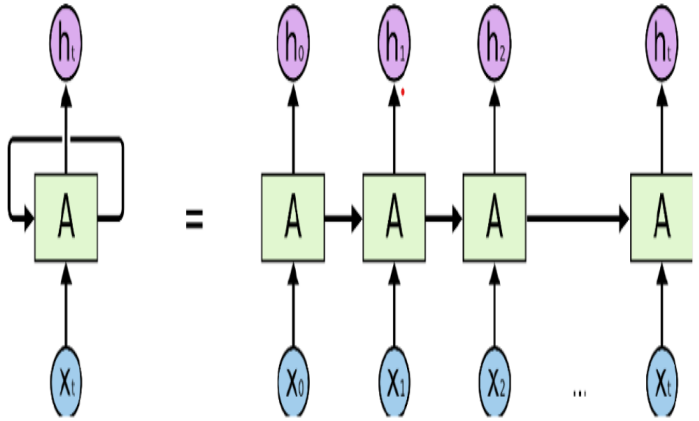


Fig-1:Building Blocks of RNN

Recurrent Hidden Units

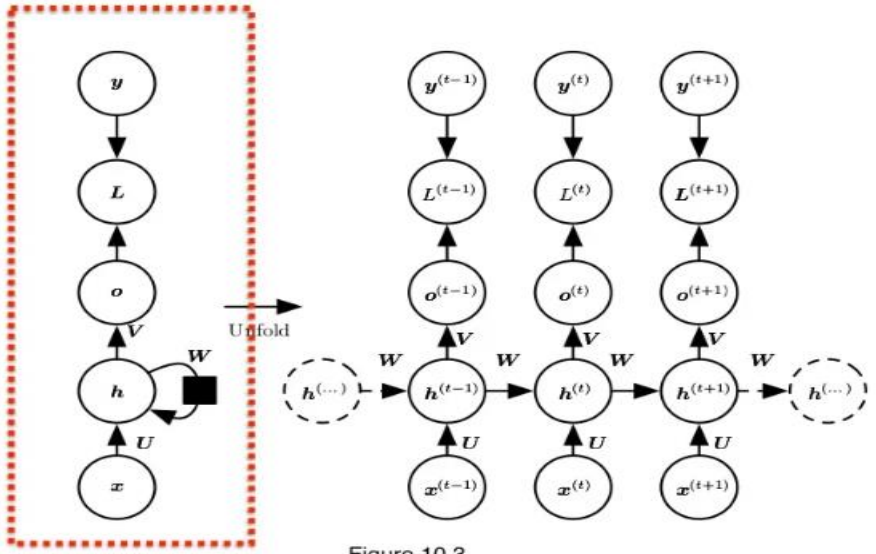


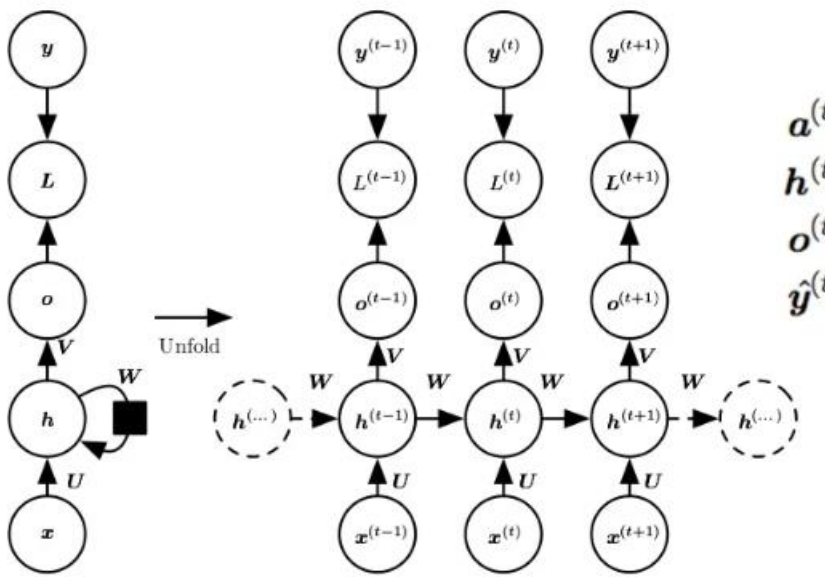
Figure 10.3

(Goodfellow 2016)

Loss calculation:

- At individual time step
- After final computation

Recurrent Hidden Units



$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$

Data sample representation

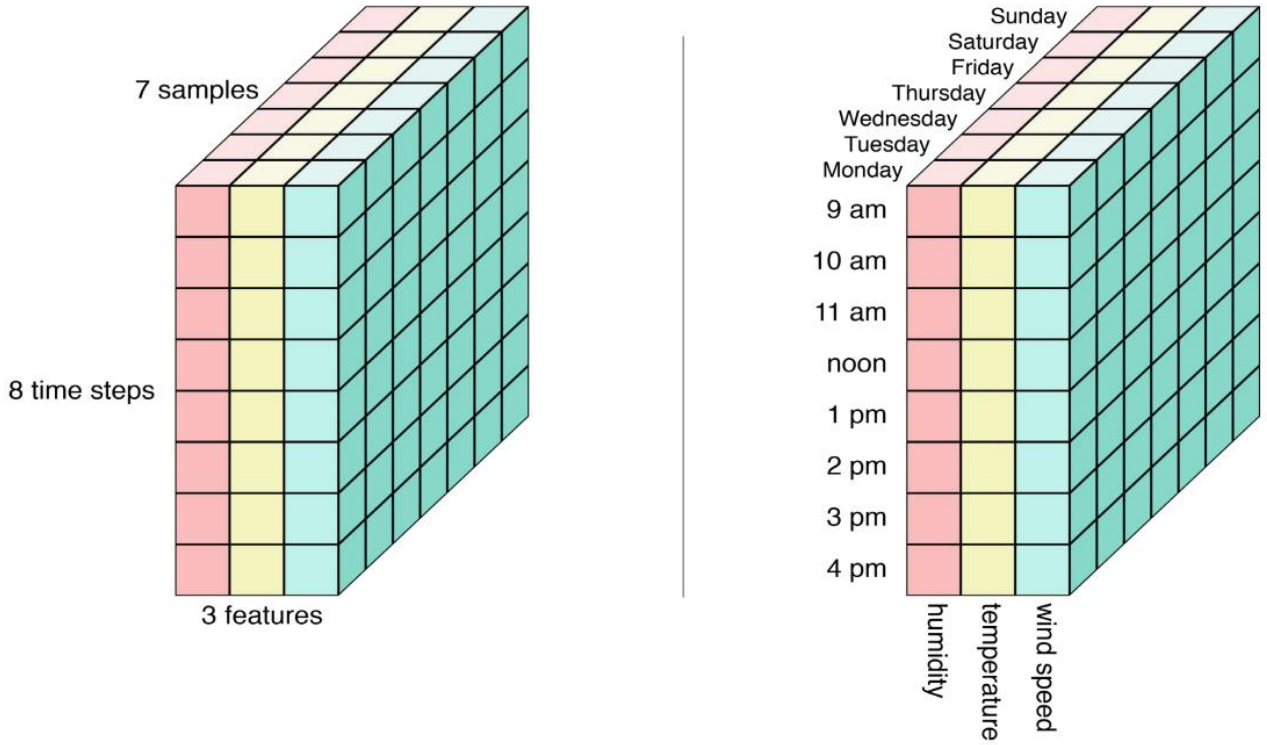
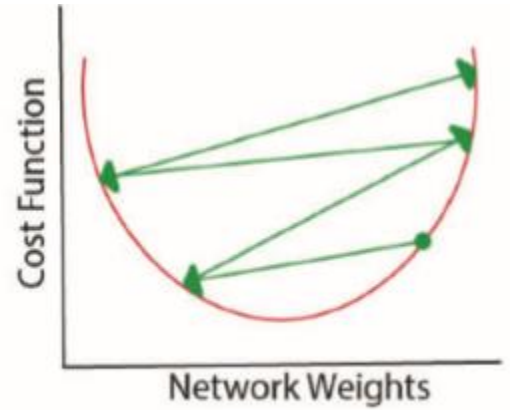


Figure: Suppose we collect our 8 measurements of 3 types of data on 7 consecutive days.

We can arrange our data as 7 samples, each composed of 3 features, with 8 time steps per feature.

Issues with standard RNN

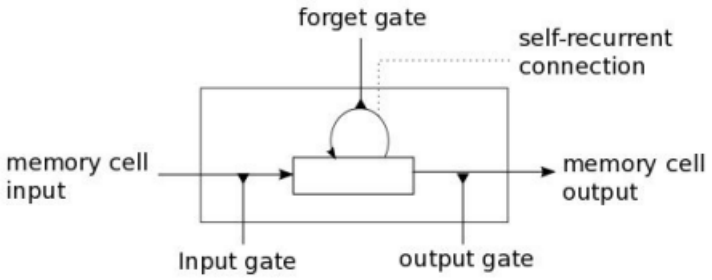


- Two main issues :
  - Vanishing Gradient problem
  - Exploding Gradient problem

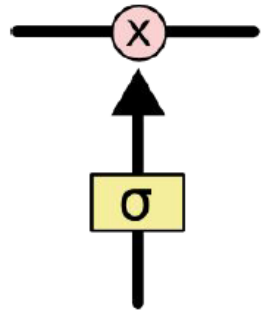
- Standard RNN can’t remember information if it has very large inputs.
- So LSTM comes into Action.

LSTM(Long-short term memory)

- A type of RNN architecture that addresses the vanishing/exploding gradient problem and allows learning of long-term dependencies
- Recently risen to prominence with state-of-the-art performance in speech recognition, language modeling, translation, image captioning



Gates



- Gate: sigmoid neural net layer followed by pointwise multiplication operator
- Gates control the flow of information to/from the memory
- Gates are controlled by a concatenation of the output from the previous time step and the current input and optionally the cell state vector.

An LSTM uses gates for three purposes.

1. Forgetting
2. Remembering - (in input gate)
3. Selecting – (in output gate)

The remember and select gates are also called the **input** and **output** gates.

Gates

- A key mechanism in LSTM is a **gate**. First, lets understand it.
- Fig shows a physical gate controlling flow of water out of a pipe

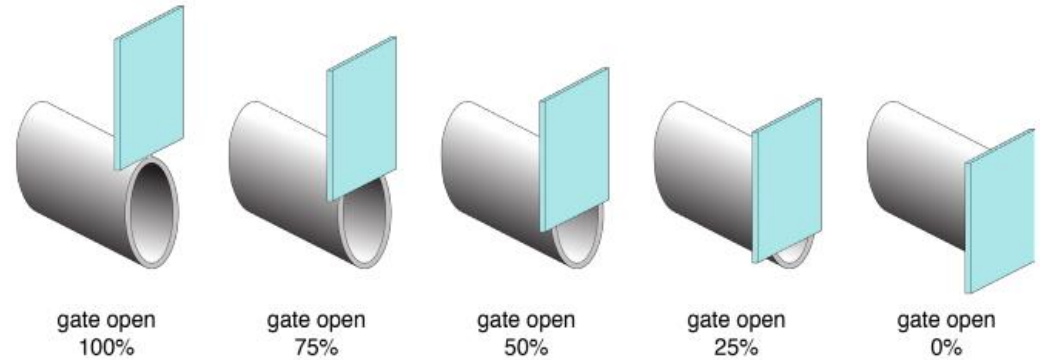


Figure : Picturing a gate over a water pipe.

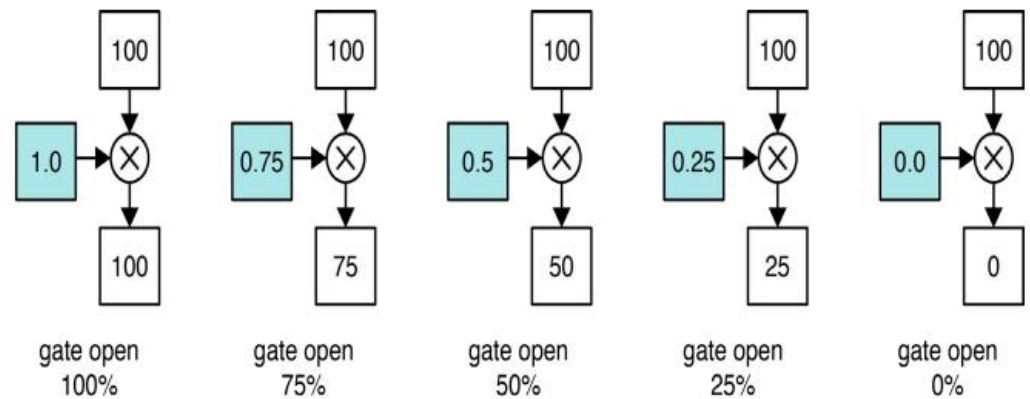


Figure: Implementing a gate with numbers.

In each diagram, the input value of 100 is shown at the top, the gate value is shown in blue, and the result is in the box at the bottom.

We just multiply the input value by the gate value to get back the gated value.

Forgetting using Gates

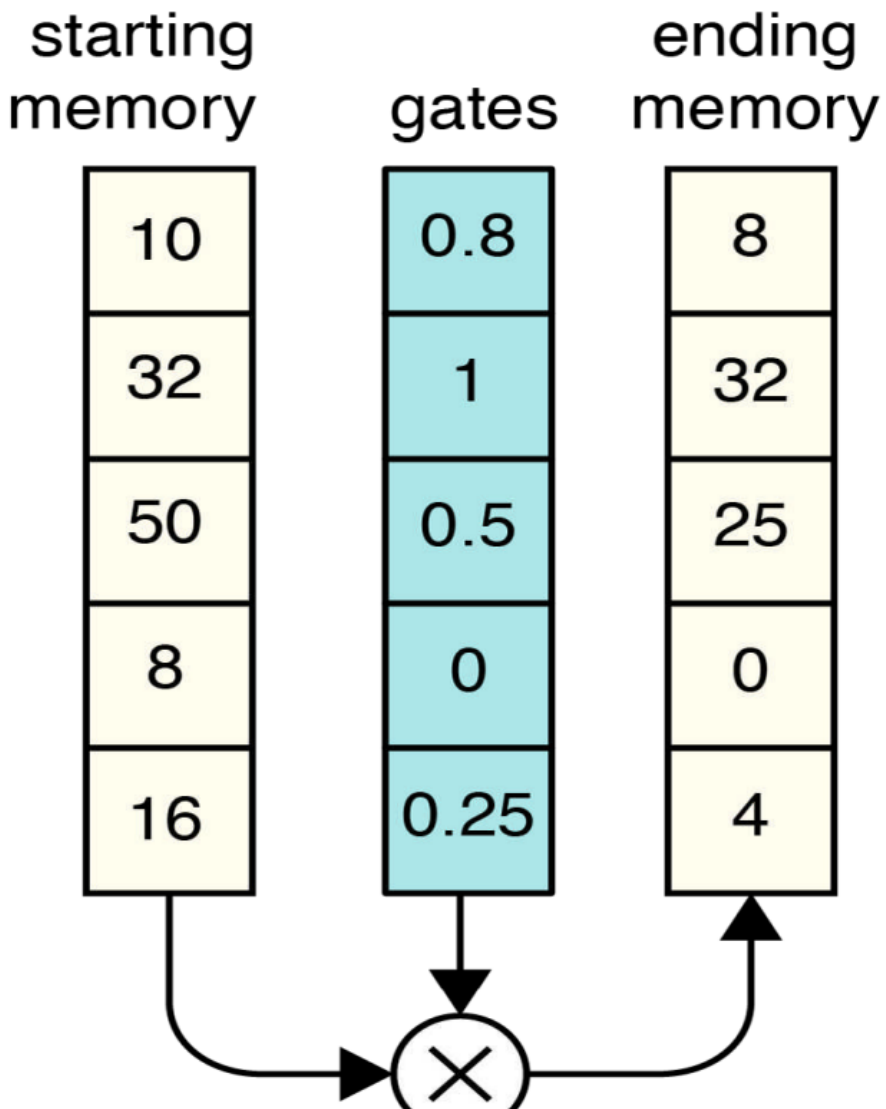
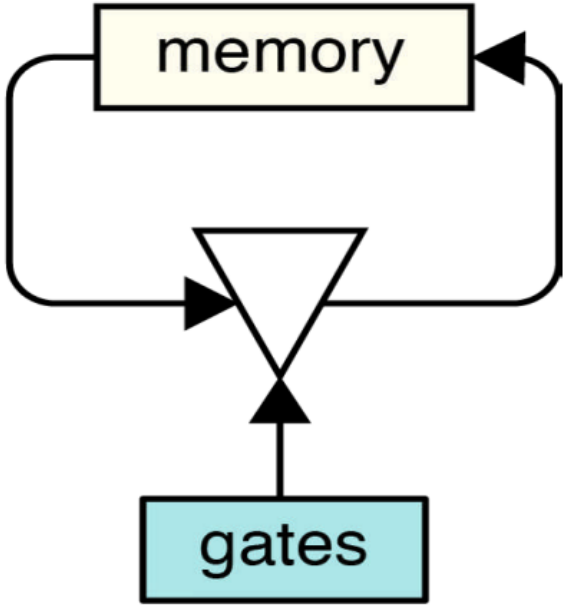


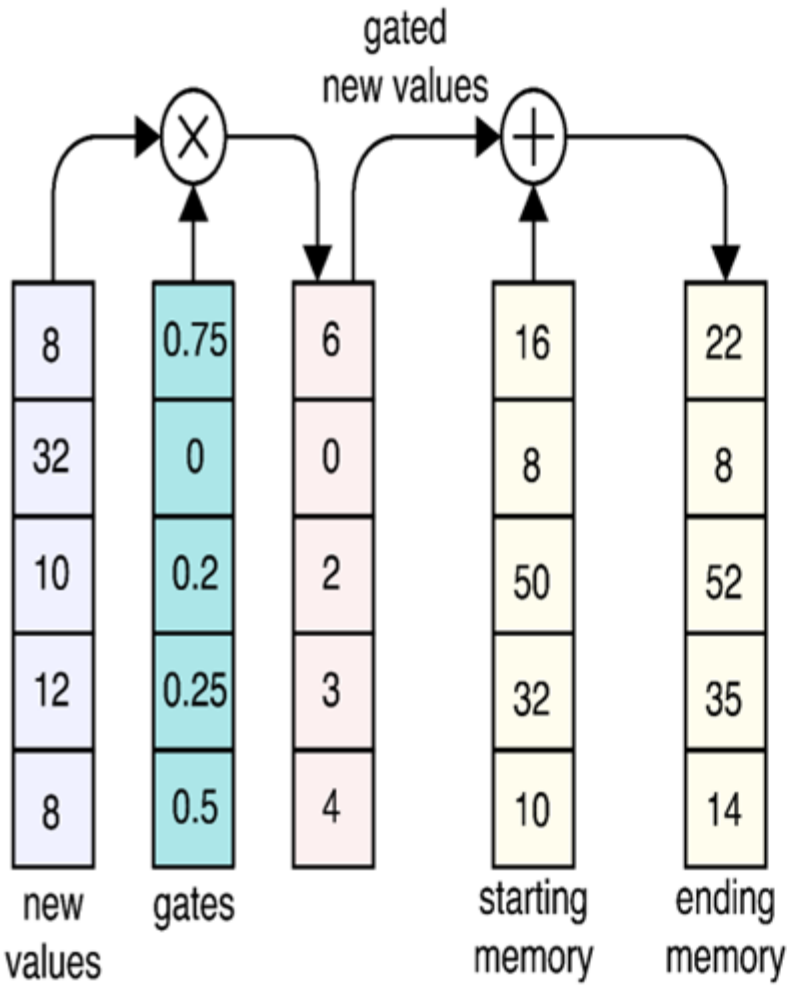
Figure: Forgetting values in memory.

(a) Each value is multiplied by a gate, and the result is stored back into the memory. Gate values of 1 don't forget anything about their corresponding memory cell, while gate values of 0 cause that value to be completely forgotten. Intermediate values of the gate forget the cell contents by a corresponding amount.

(b) The operation in schematic form.



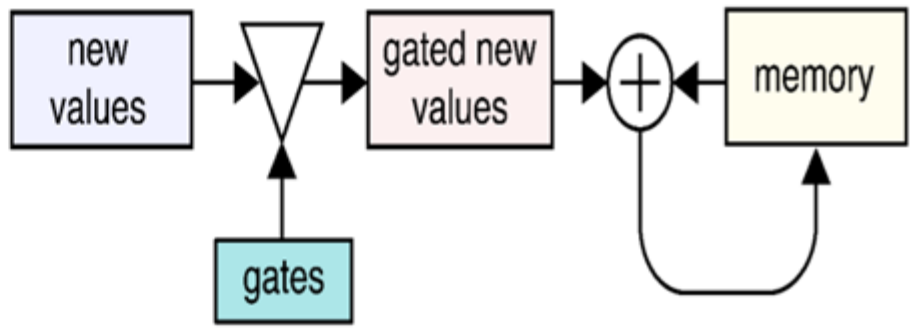
Remembering using Gates



(a)

Figure: The process of remembering. (a) We start with new values to remember, shown on the left. We may not want to remember these at full strength, so we first apply gates.

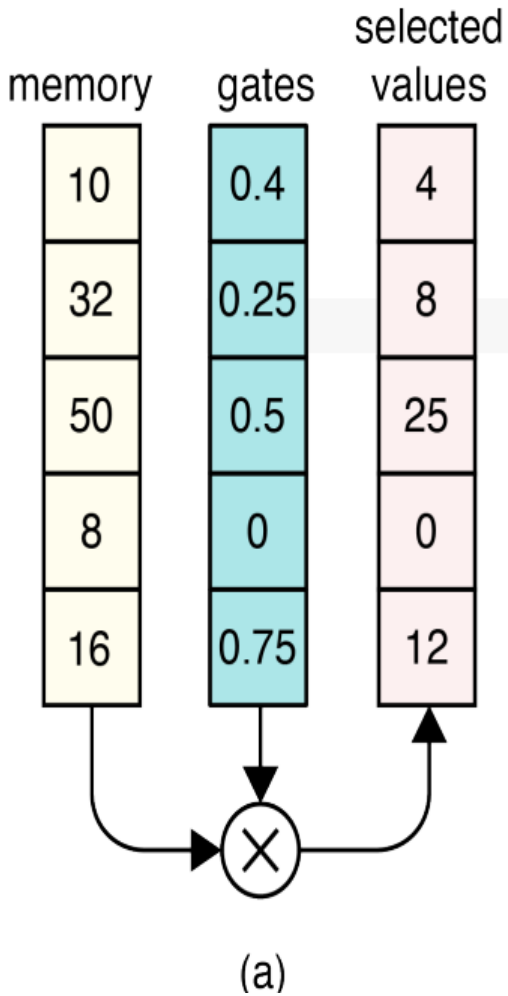
Then we add the gated values to the existing memory, and save that back into the memory.



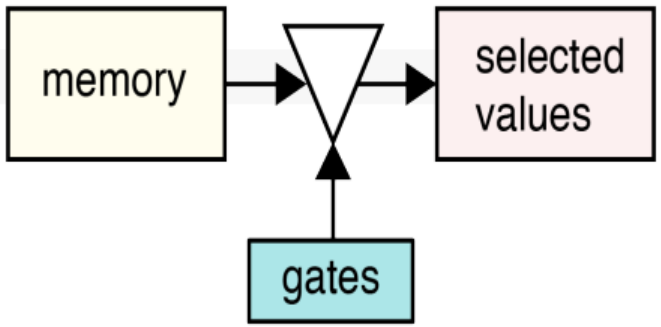
(b)

Selection using Gates

To **select** from memory, just determine how much of each element we want to use.



(a)



(a) To select memories, we gate the values in our memory. The gated results are our selections.

Apply gates to the memory elements, and the results are a list of scaled memories.

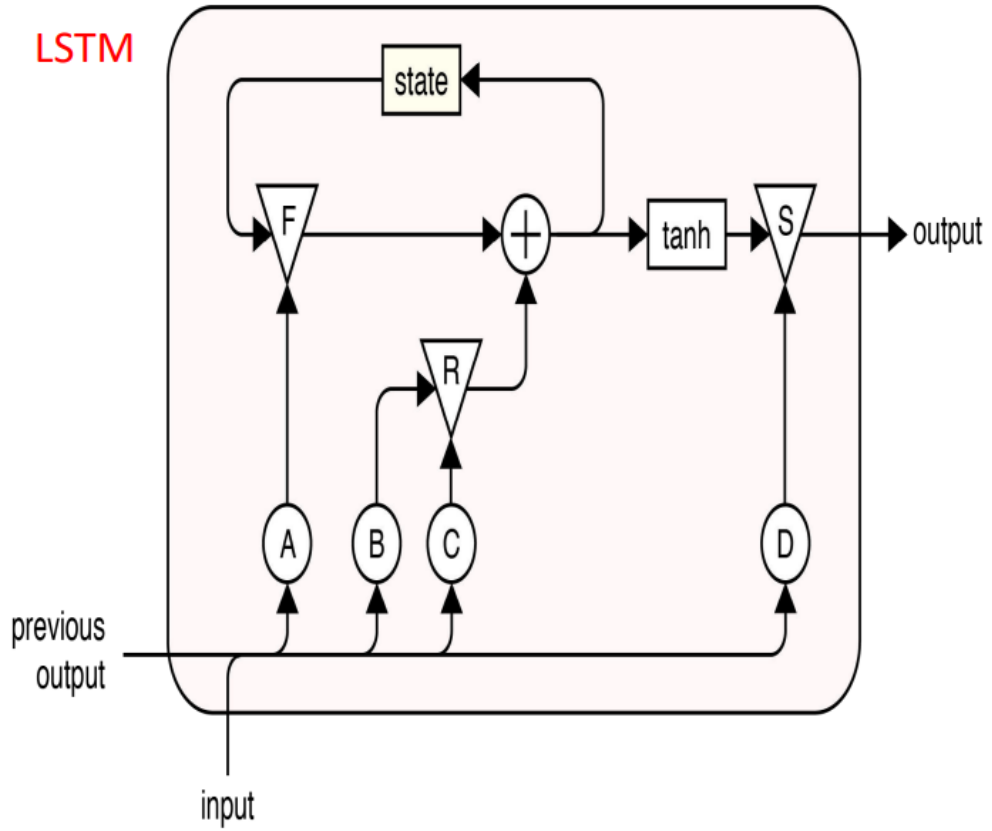


Figure: The architecture of an LSTM unit.

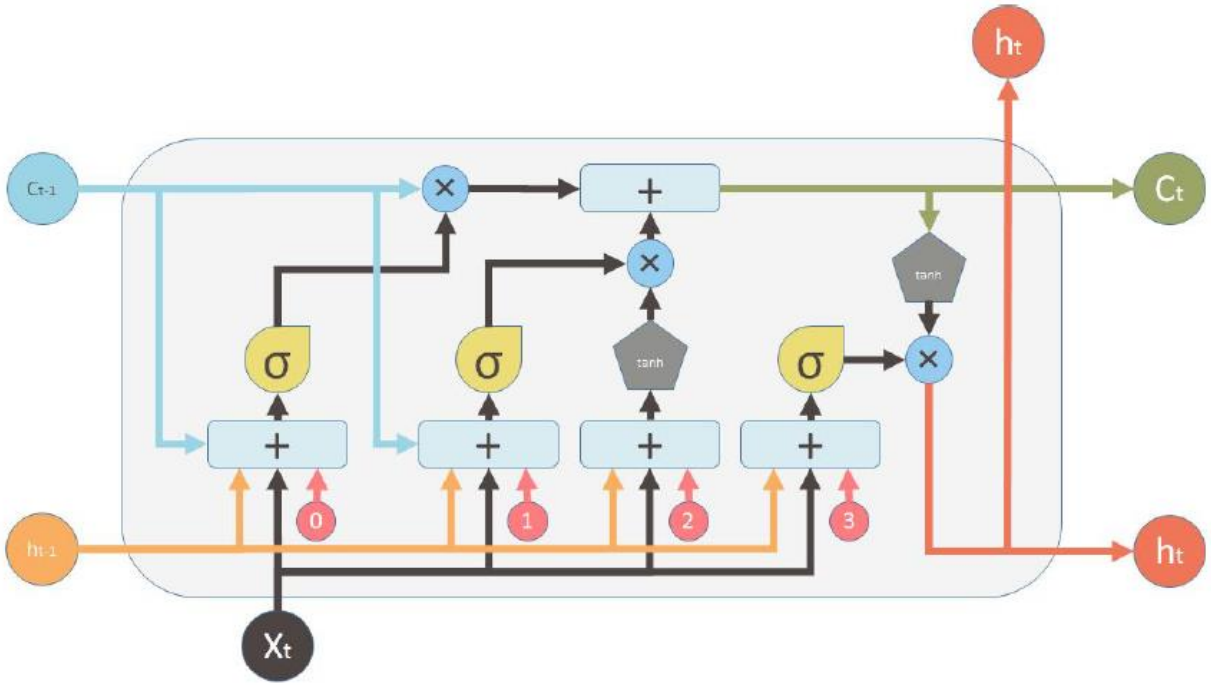
Inputs: Previous output and a new input

Output: New output on right extreme.

State memory: At top of diagram.

- Triangles represent gates: There are three gates, F for forget, R for remember, and S for select.
  - Gates manage internal memory and give a lot of control over what cell remembers and forgets over time, so it can manage its internal cell memory in the most effective way.
- Circles represent sets of neurons, labeled A through D. The input to these neurons is a single list formed by simply placing the previous output and the new input one atop the other.

LSTM Memory Cell



Inputs:

- X_t Input vector
- C_{t-1} Memory from previous block
- h_{t-1} Output of previous block

outputs:

- C_t Memory from current block
- h_t Output of current block

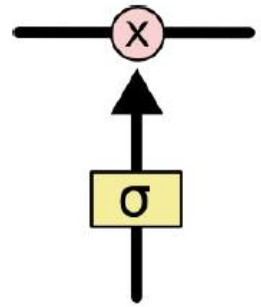
Nonlinearities:

- σ Sigmoid
- tanh Hyperbolic tangent

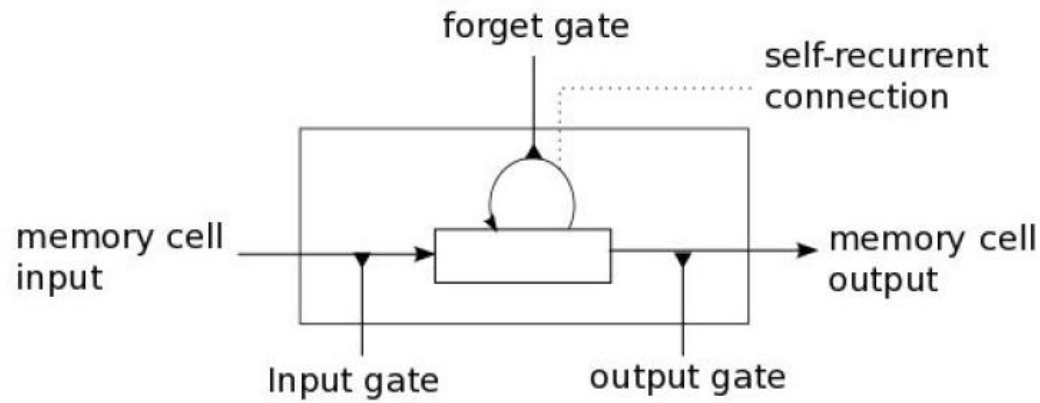
Bias: 0

Vector operations:

- × Element-wise multiplication
- \+ Element-wise Summation / Concatenation



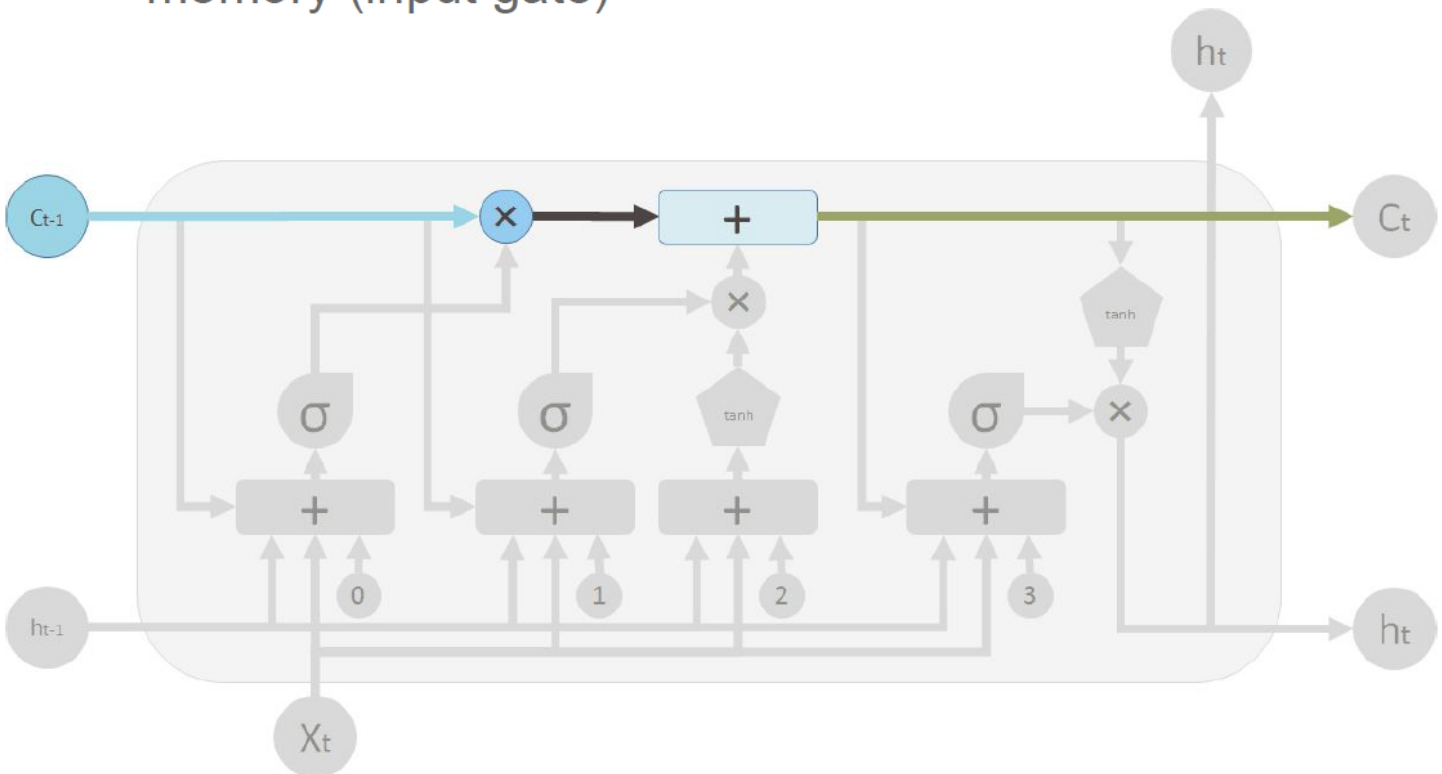
Gate (sigmoid layer followed by pointwise multiplication)

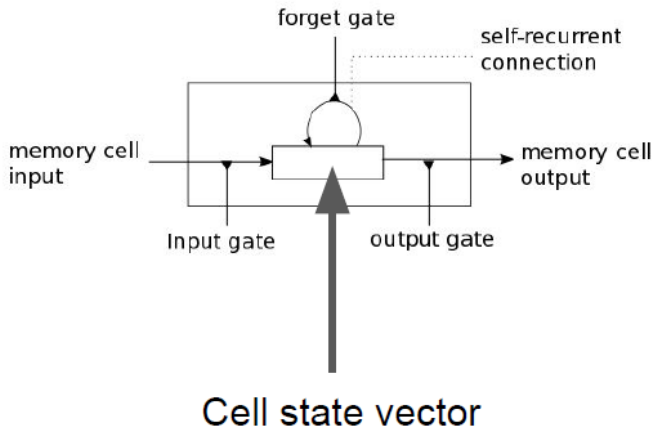


Simplified schematic for reference

Cell state vector

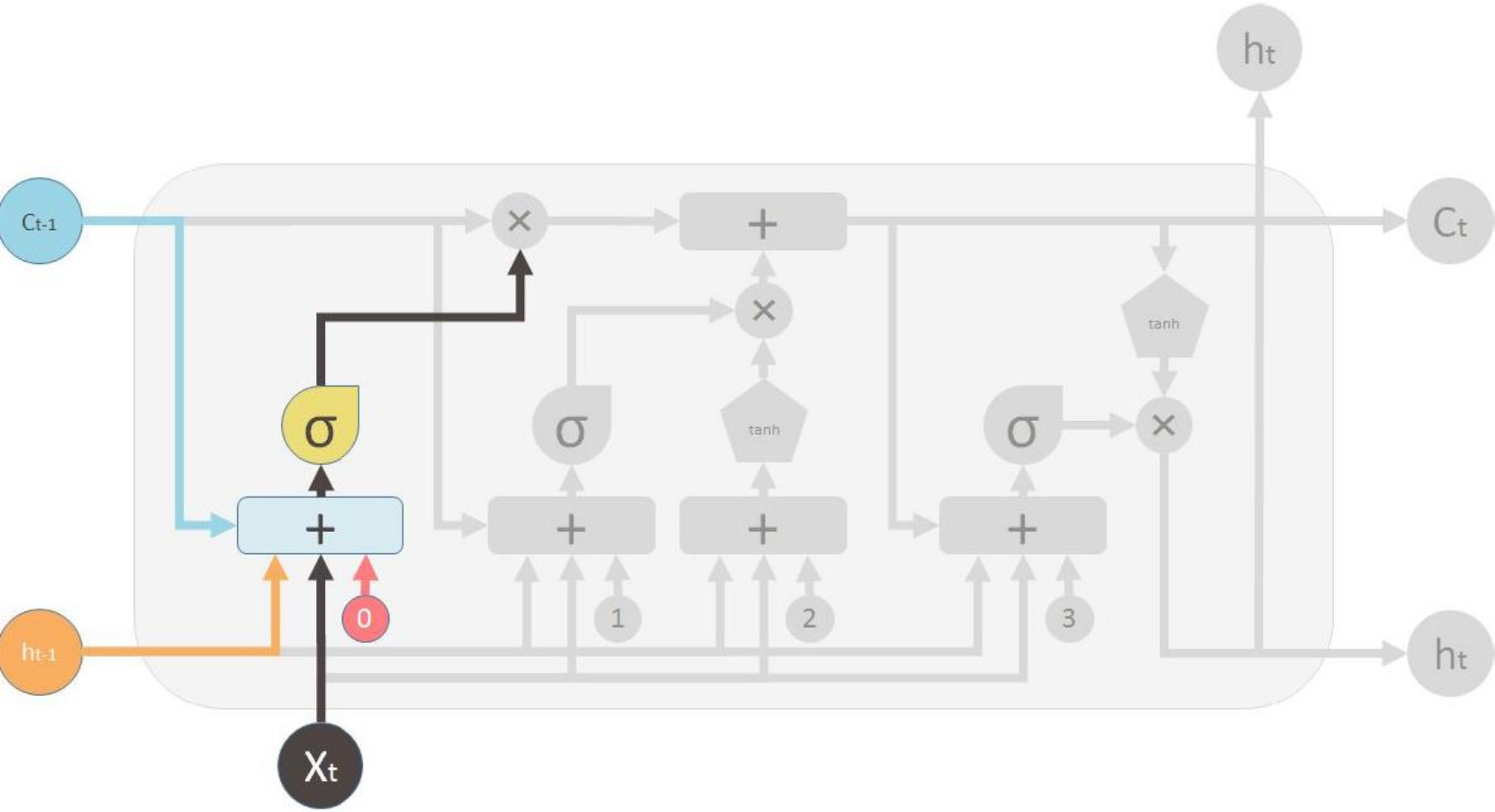
- Represents the memory of the LSTM
- Undergoes changes via forgetting of old memory (forget gate) and addition of new memory (input gate)



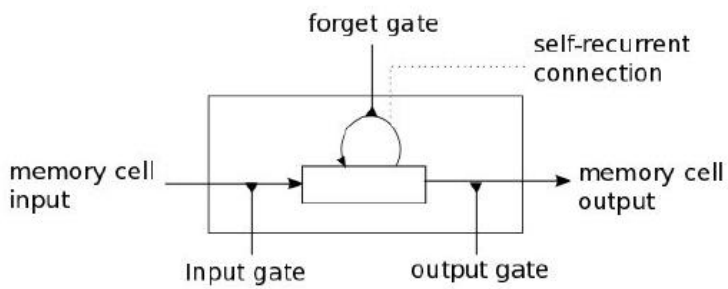


Forget Gate

- Controls what information to throw away from memory

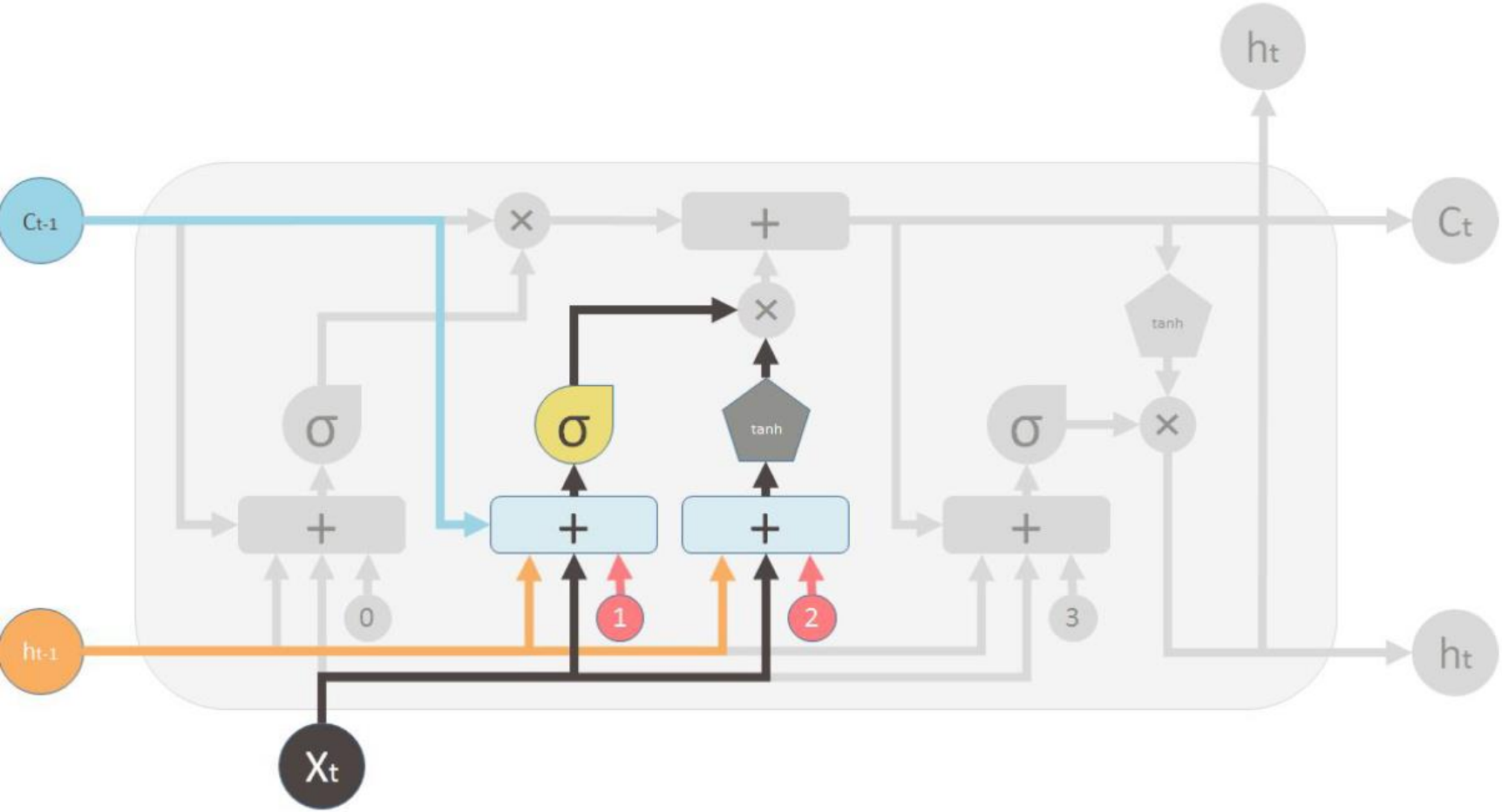


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$



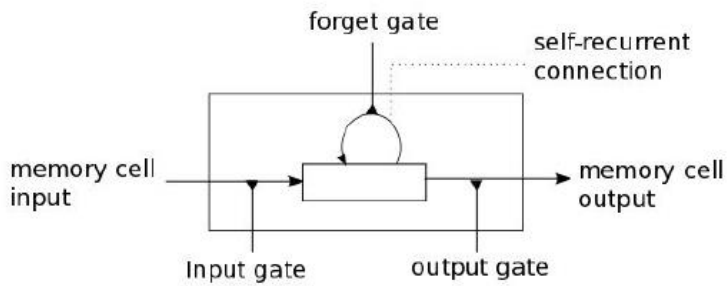
Input Gate

- Controls what new information is added to cell state from current input



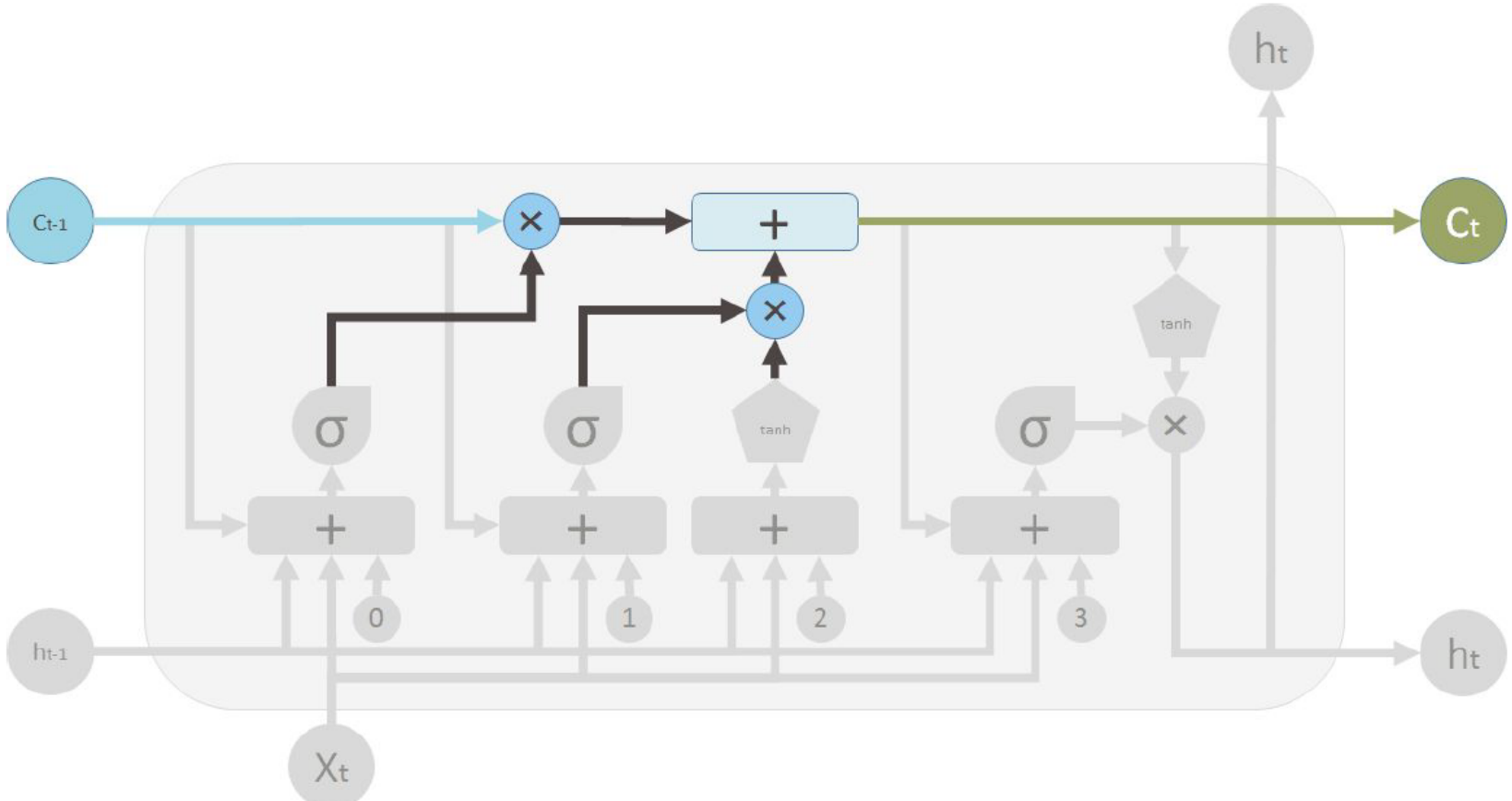
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

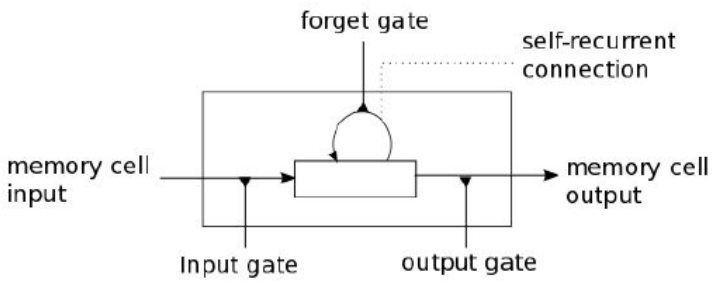


Memory Update

- The cell state vector aggregates the two components (old memory via the forget gate and new memory via the input gate)

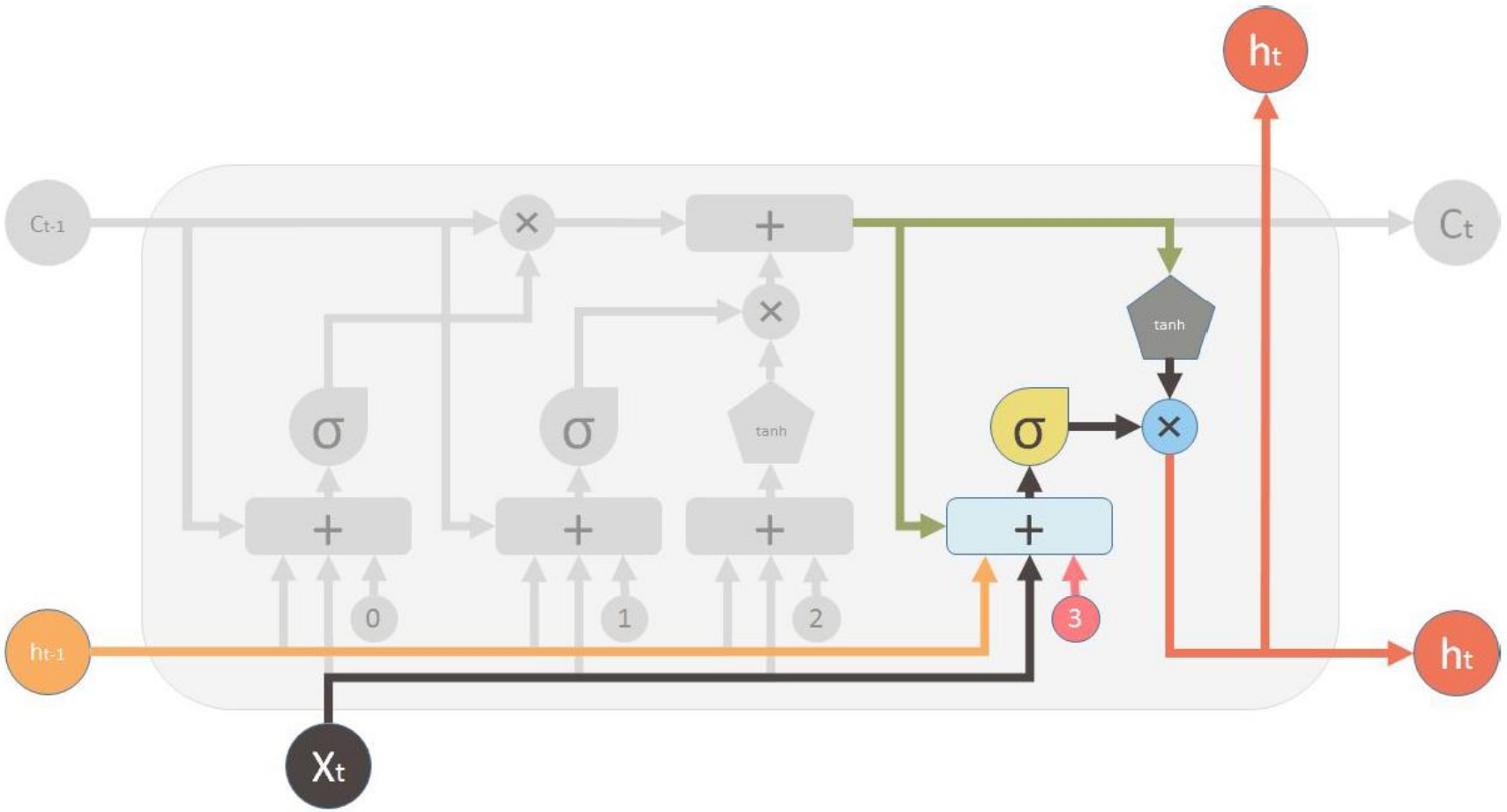


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



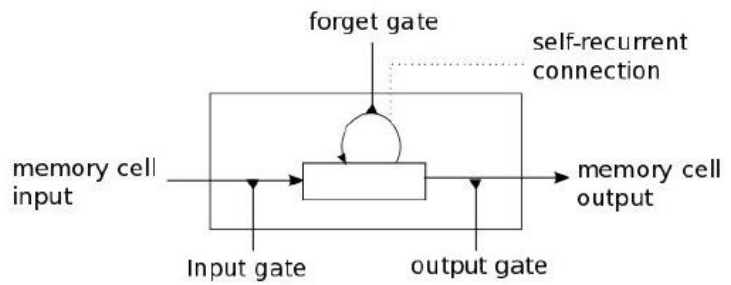
Output Gate

- Conditionally decides what to output from the memory

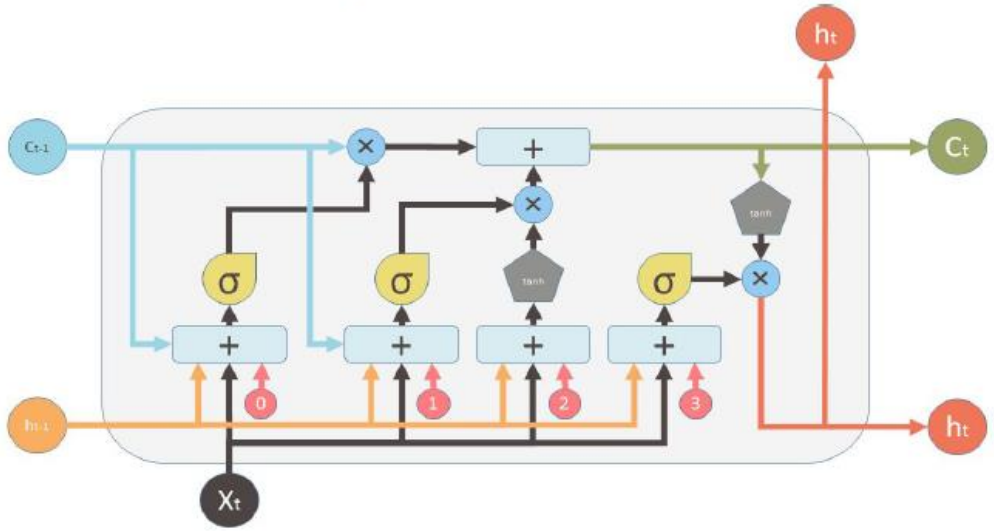


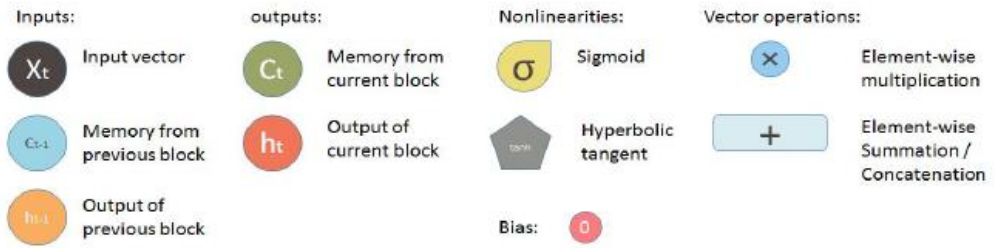
$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

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



LSTM Memory Cell Summary





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

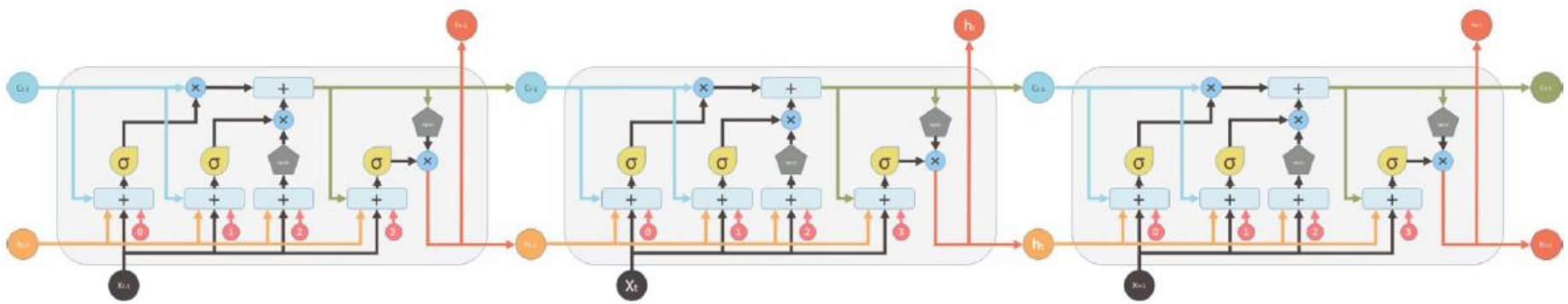
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

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

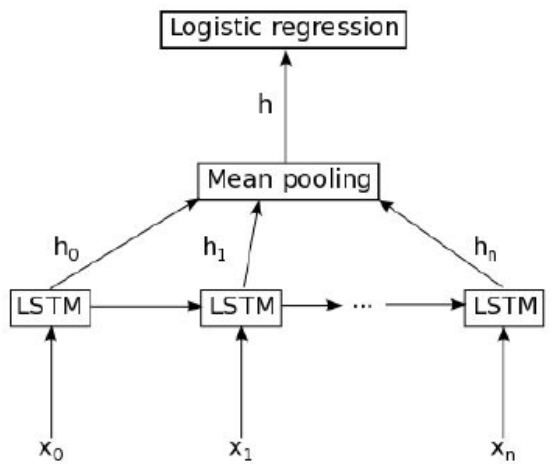
$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$

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



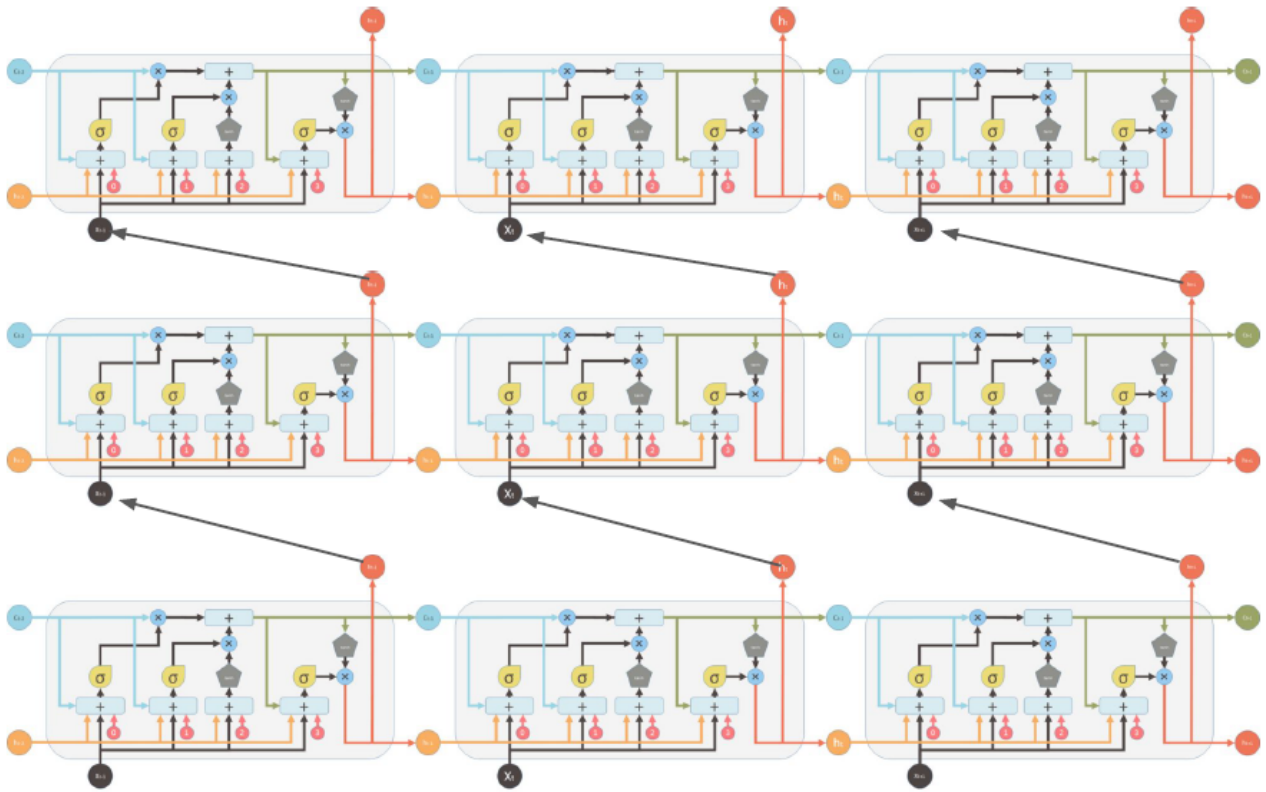
LSTM Training

- Backpropagation Through Time (BPTT) most common
- What weights are learned?
  - Gates (input/output/forget)
  - Input tanh layer
- Outputs depend on the task:
  - Single output prediction for the whole sequence (e.g. below)
  - One output at each time step (sequence labeling)



Deep LSTMs

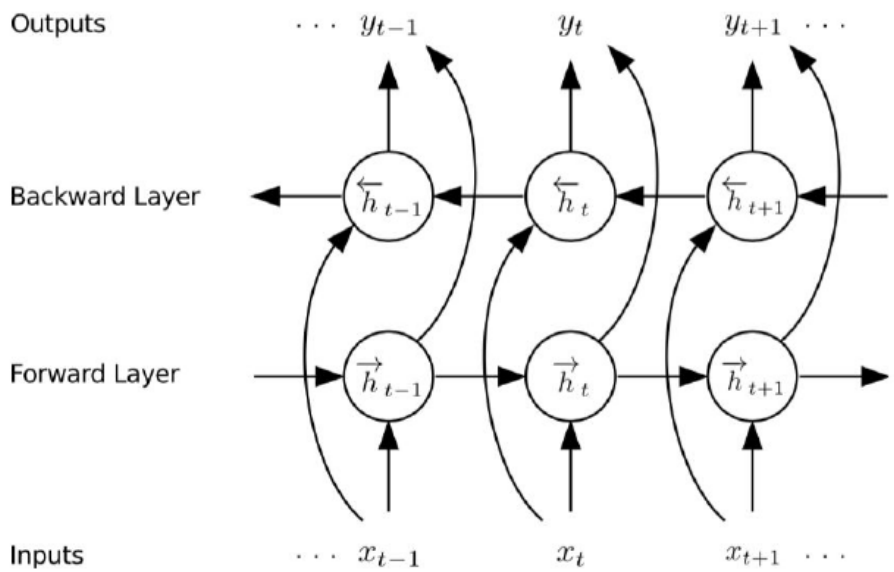
- Deep LSTMs can be created by stacking multiple LSTM layers vertically, with the output sequence of one layer forming the input sequence of the next (in addition to recurrent connections within the same layer)
- Increases the number of parameters - but given sufficient data, performs significantly better than single-layer LSTMs (Graves et al. 2013)
- Dropout usually applied only to non-recurrent edges, including between layers



Bidirectional RNNs

- Data processed in both directions processed with two separate hidden layers, which are then fed forward into the same output layer

- Bidirectional RNNs can better exploit context in both directions, for e.g. bidirectional LSTMs perform better than unidirectional ones in speech recognition (Graves et al. 2013)



LSTMs for Machine Translation

- Encoder and decoder LSTMs