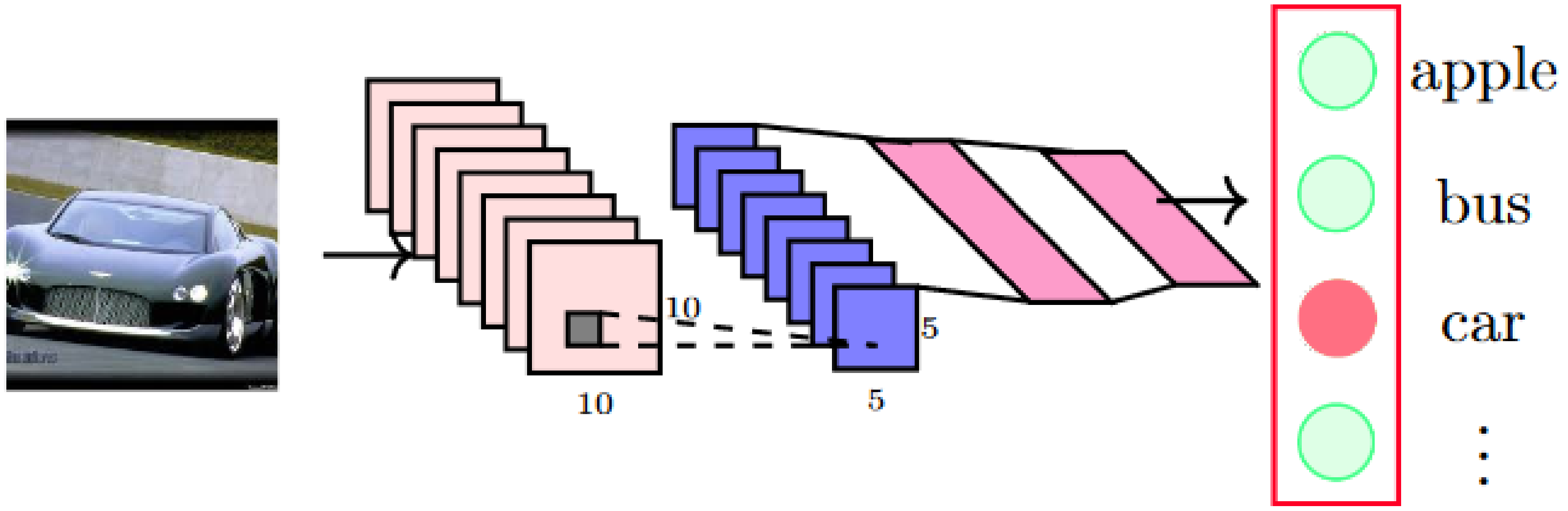
Sequence modeling – recurrent networks

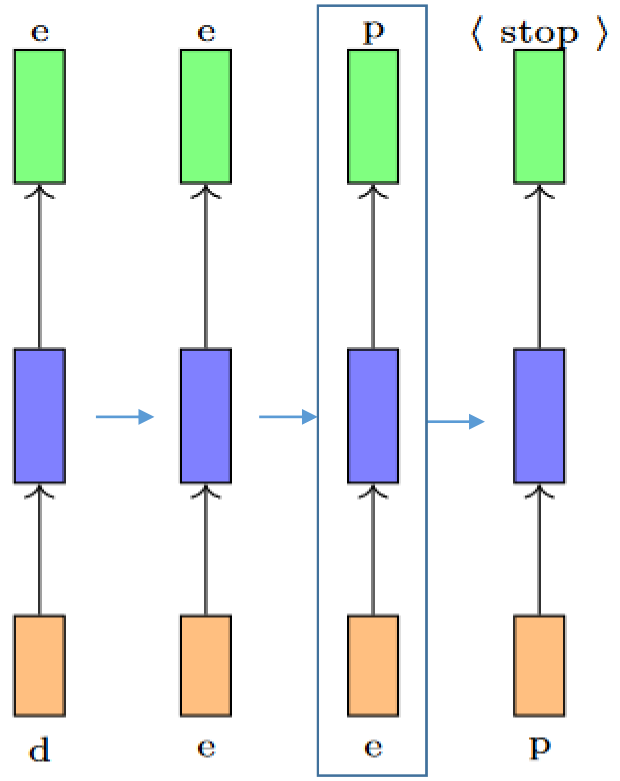
Biplab Banerjee

Sequential learning problem



- ✓ Each input is of fixed size
- ✓ Inputs are independent

Sequential learning problem

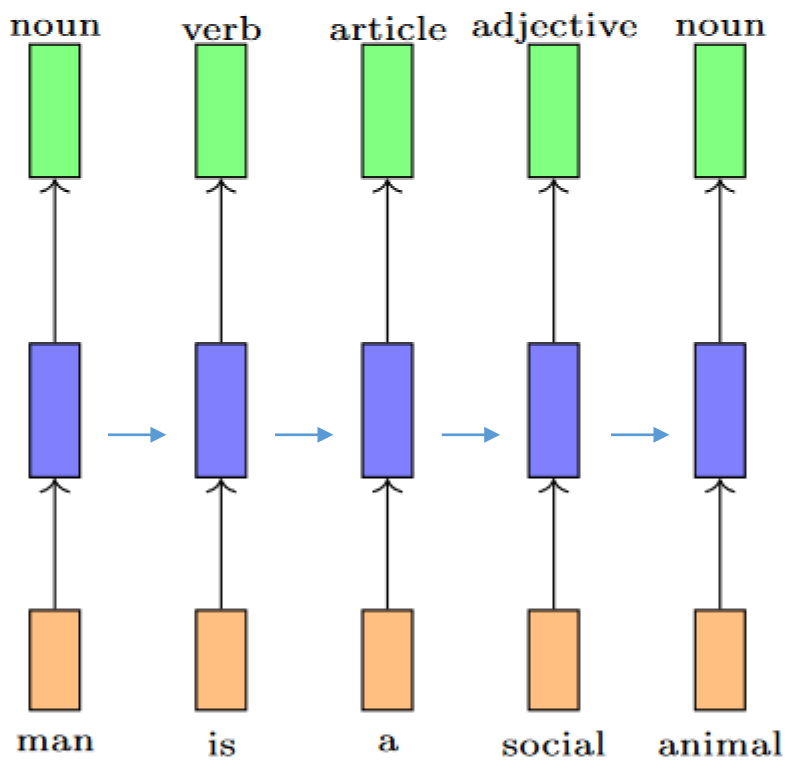


Auto-correct application in e-mail

- ✓ Input lengths can vary
- ✓ Inputs are no longer independent
- ✓ At each time stamp, the same operation is Carried out

Given previous **information** And current input, **predict** Next input

Sequential learning problem



<u>Parts of speech tagging</u>

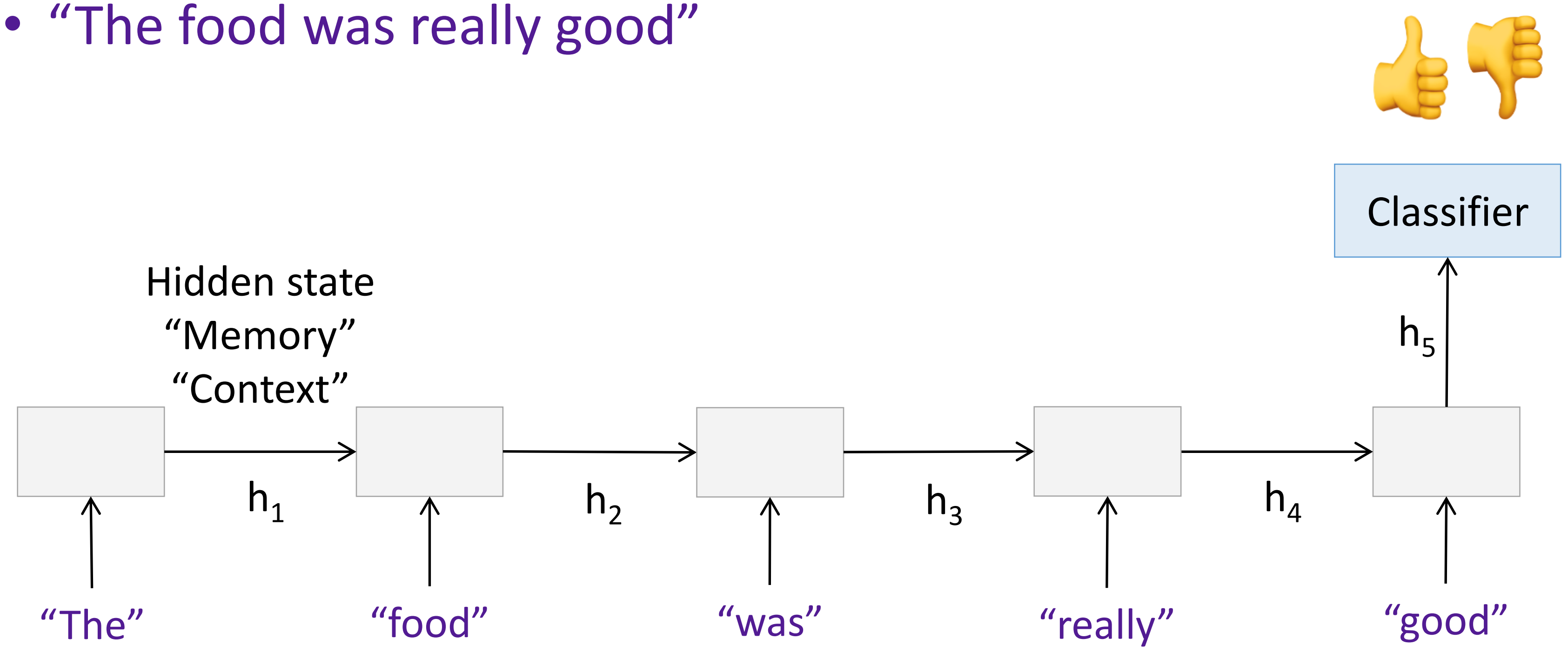
A set of previous time-stamps provide useful information
About the next time-stamp

Text classification

- **Sentiment classification:** classify a restaurant or movie or product review as positive or negative
  - “The food was really good”
  - “The vacuum cleaner broke within two weeks”
  - “The movie had slow parts, but overall was worth watching”
- What feature representation or predictor structure can we use for this problem?

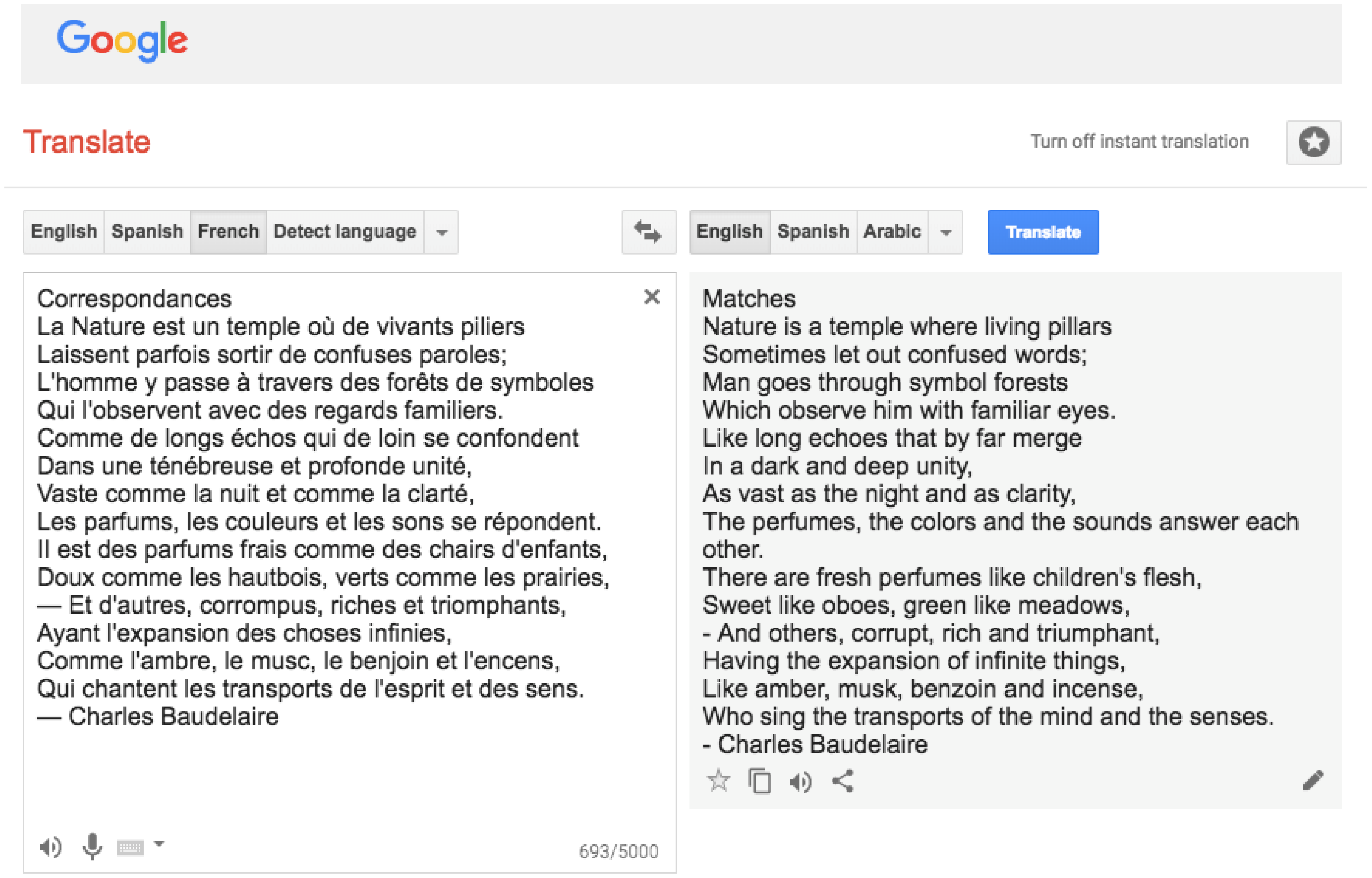
Sentiment classification

- "The food was really good"



Recurrent Neural Network (RNN)

Machine translation



Google
Translate
Turn off instant translation
English Spanish French Detect language
English Spanish Arabic
Translate
693/5000

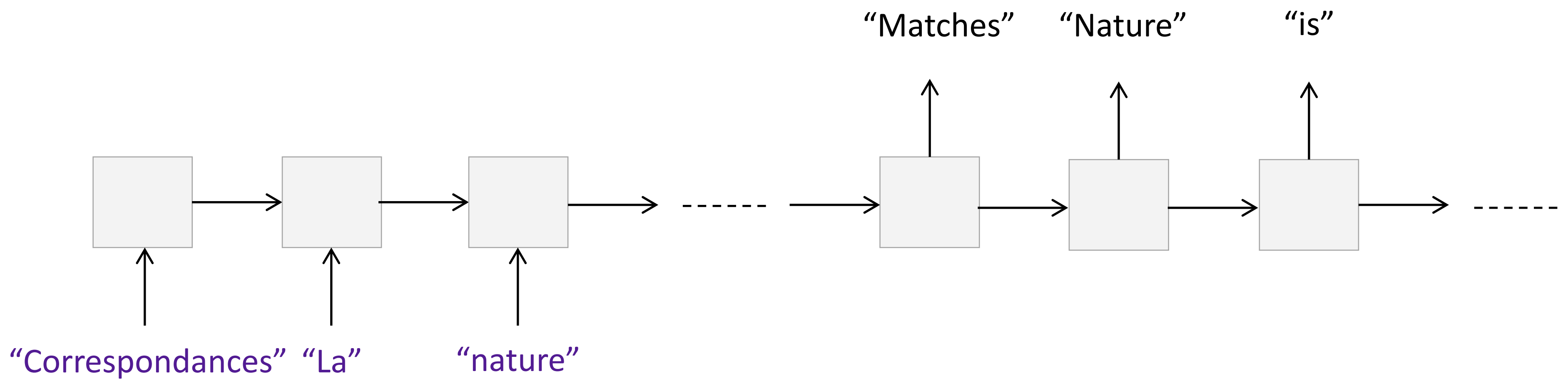
Correspondances
La Nature est un temple où de vivants piliers
Laissent parfois sortir de confuses paroles;
L'homme y passe à travers des forêts de symboles
Qui l'observent avec des regards familiers.
Comme de longs échos qui de loin se confondent
Dans une ténébreuse et profonde unité,
Vaste comme la nuit et comme la clarté,
Les parfums, les couleurs et les sons se répondent.
Il est des parfums frais comme des chairs d'enfants,
Doux comme les hautbois, verts comme les prairies,
— Et d'autres, corrompus, riches et triomphants,
Ayant l'expansion des choses infinies,
Comme l'ambre, le musc, le benjoin et l'encens,
Qui chantent les transports de l'esprit et des sens.
— Charles Baudelaire

Matches
Nature is a temple where living pillars
Sometimes let out confused words;
Man goes through symbol forests
Which observe him with familiar eyes.
Like long echoes that by far merge
In a dark and deep unity,
As vast as the night and as clarity,
The perfumes, the colors and the sounds answer each other.
There are fresh perfumes like children's flesh,
Sweet like oboes, green like meadows,
- And others, corrupt, rich and triumphant,
Having the expansion of infinite things,
Like amber, musk, benzoin and incense,
Who sing the transports of the mind and the senses.
- Charles Baudelaire

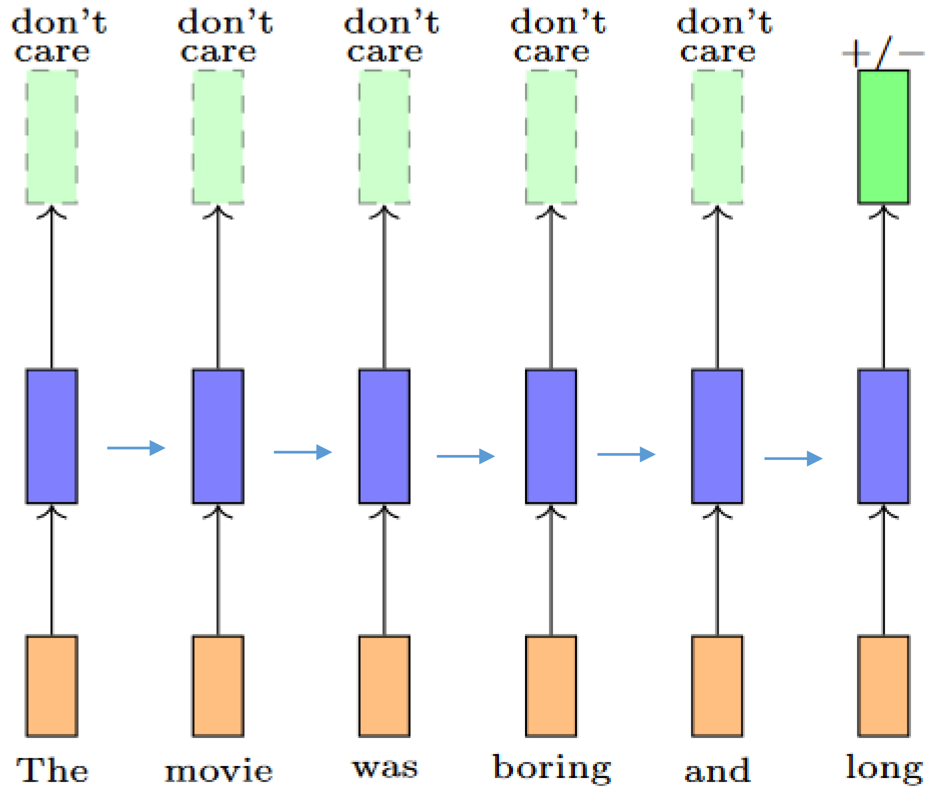
https://translate.google.com/

Machine translation

- Multiple input – multiple output (or sequence to sequence)

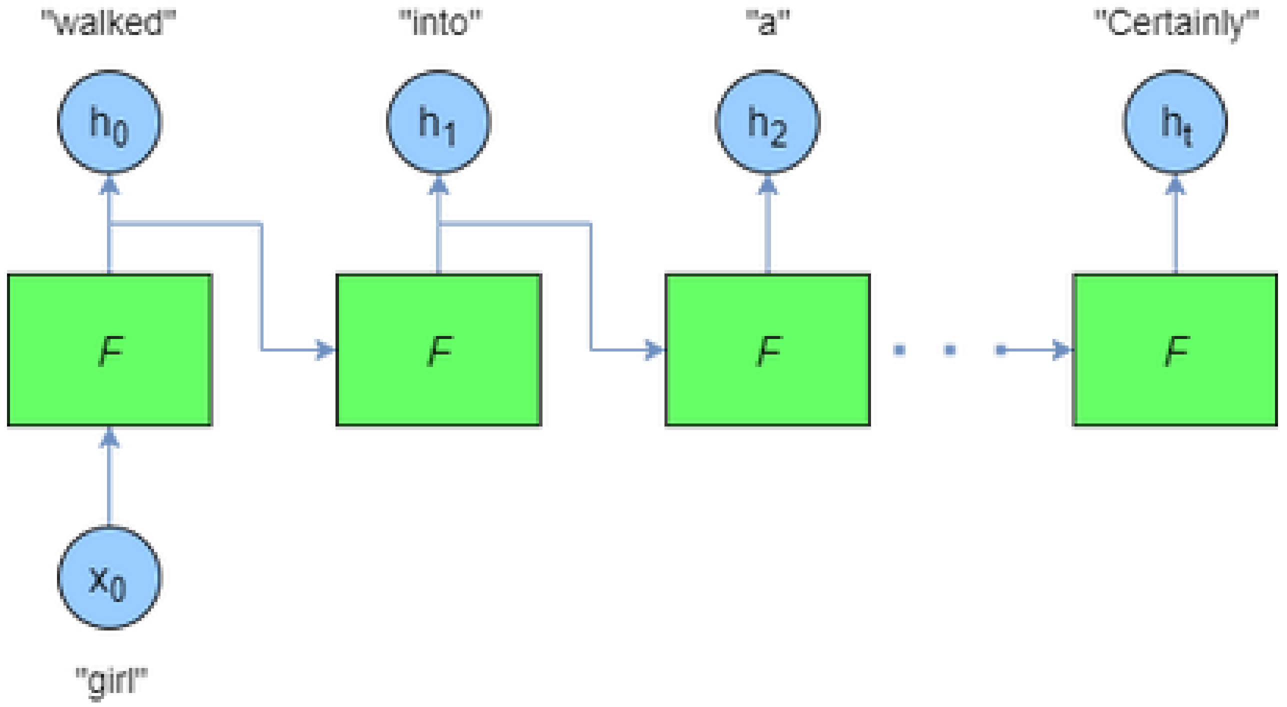


Different input output combinations



Movie review – many to one

Different input output combinations



Many to Many -

Machine translation

Dialogue

Different input output combinations

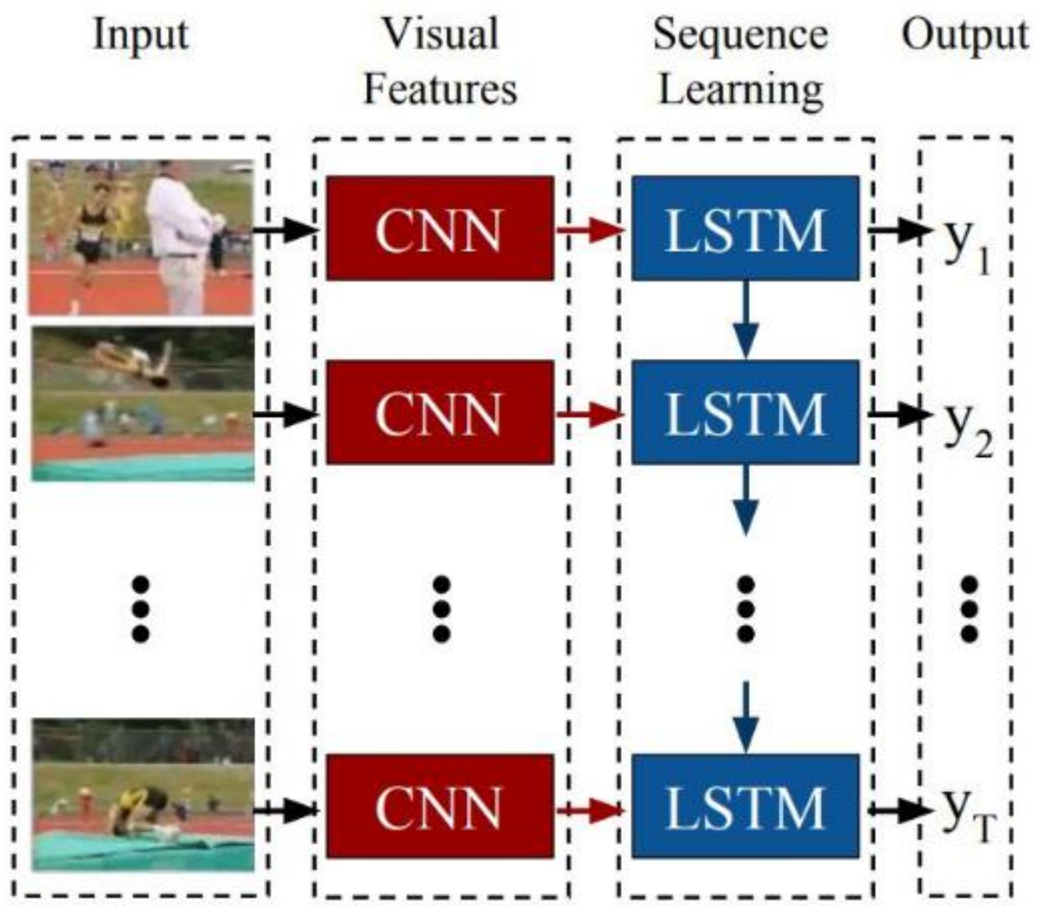
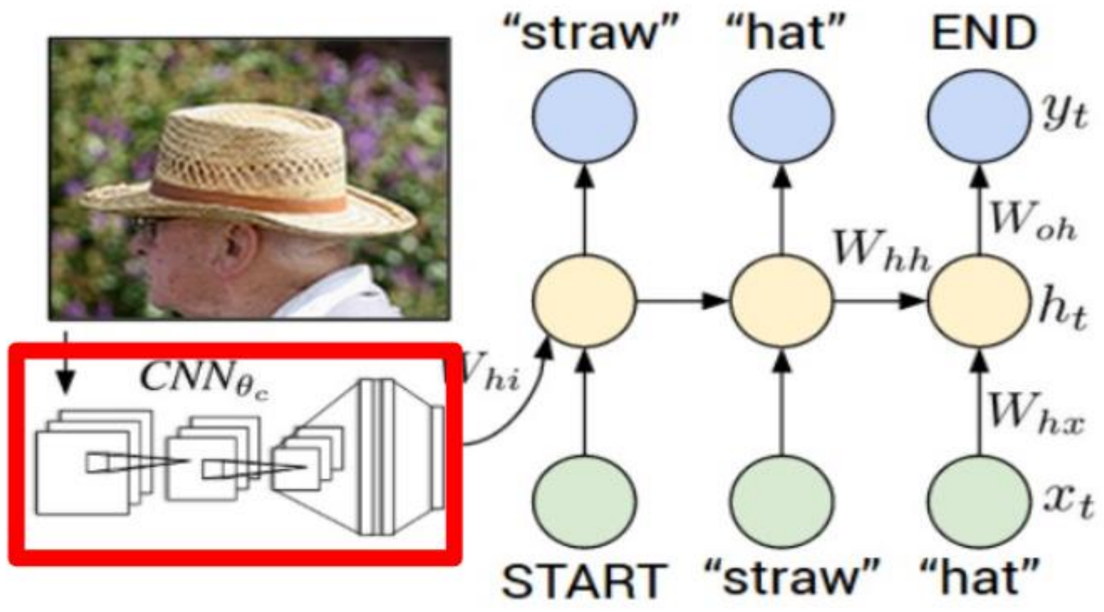
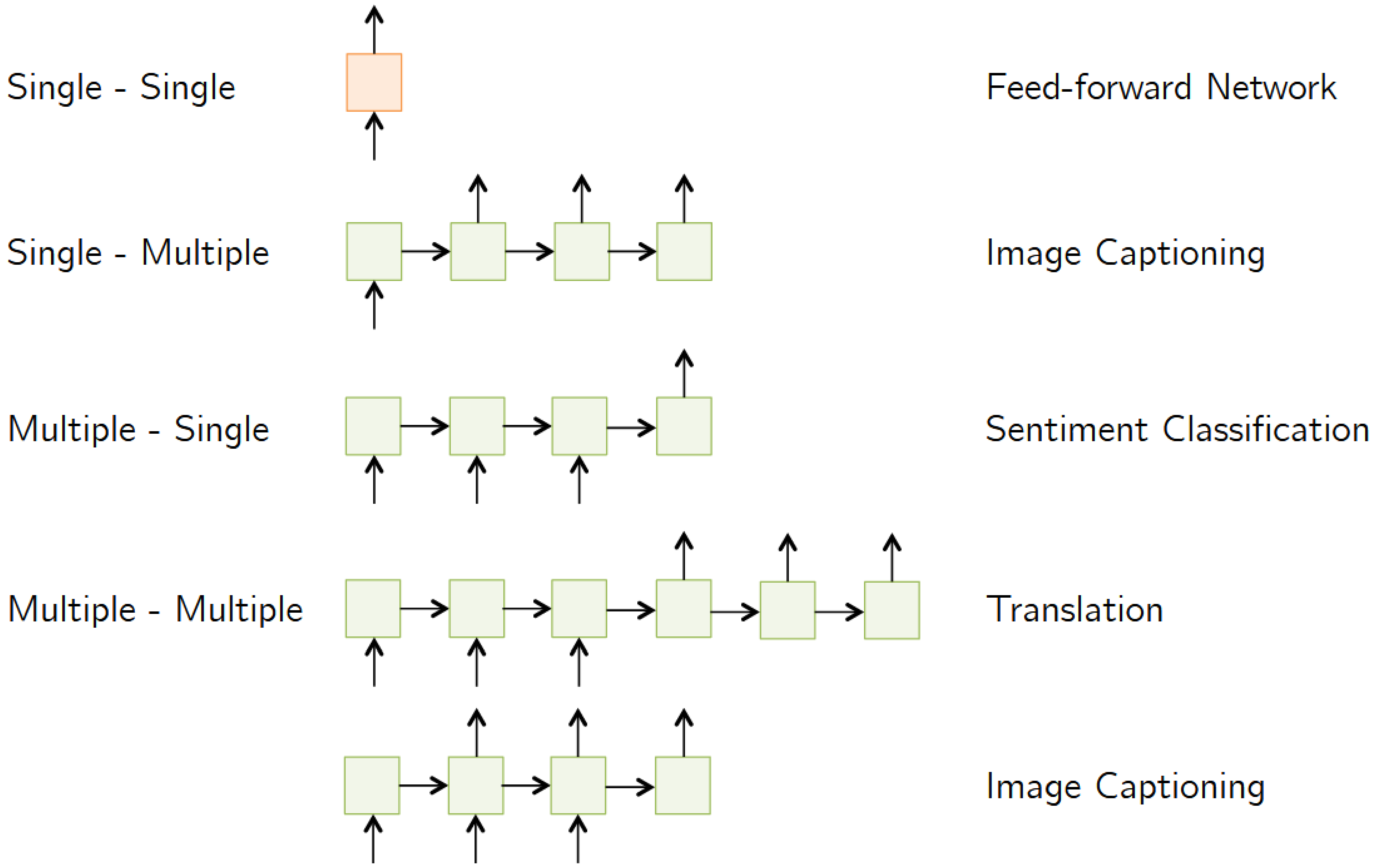


Image and video captioning



Many to many and one to many


Single - Single
Feed-forward Network
Single - Multiple
Image Captioning
Multiple - Single
Sentiment Classification
Multiple - Multiple
Translation
Image Captioning

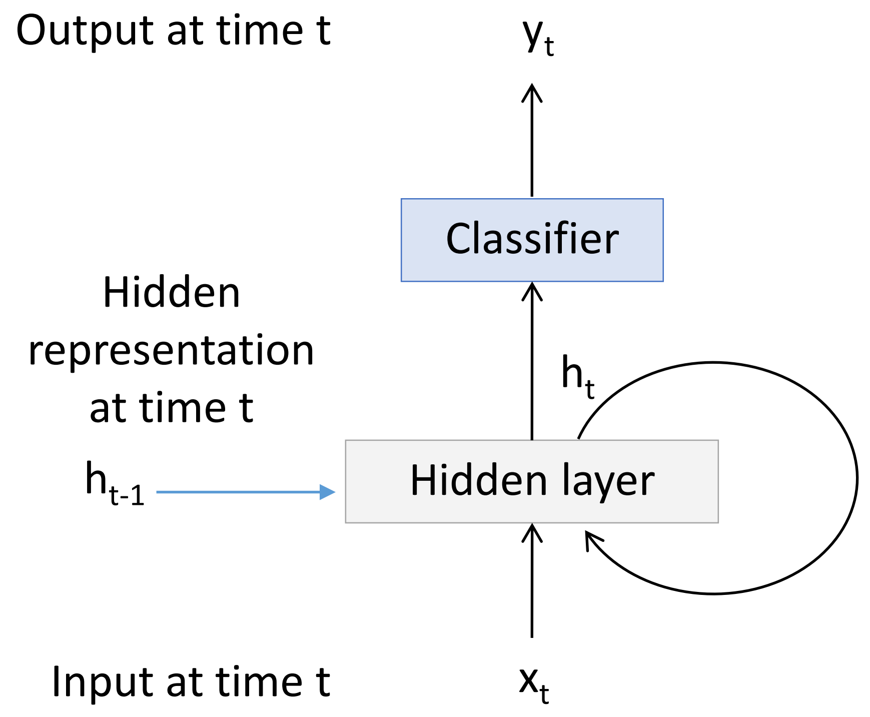
Sequential learning problem

$$P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i \mid w_1, \ldots, w_{i-1}) \approx \prod_{i=1}^{m} P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1})$$

The predicted word at a certain point can be inferred given a few previous words

Markovian Property of languages

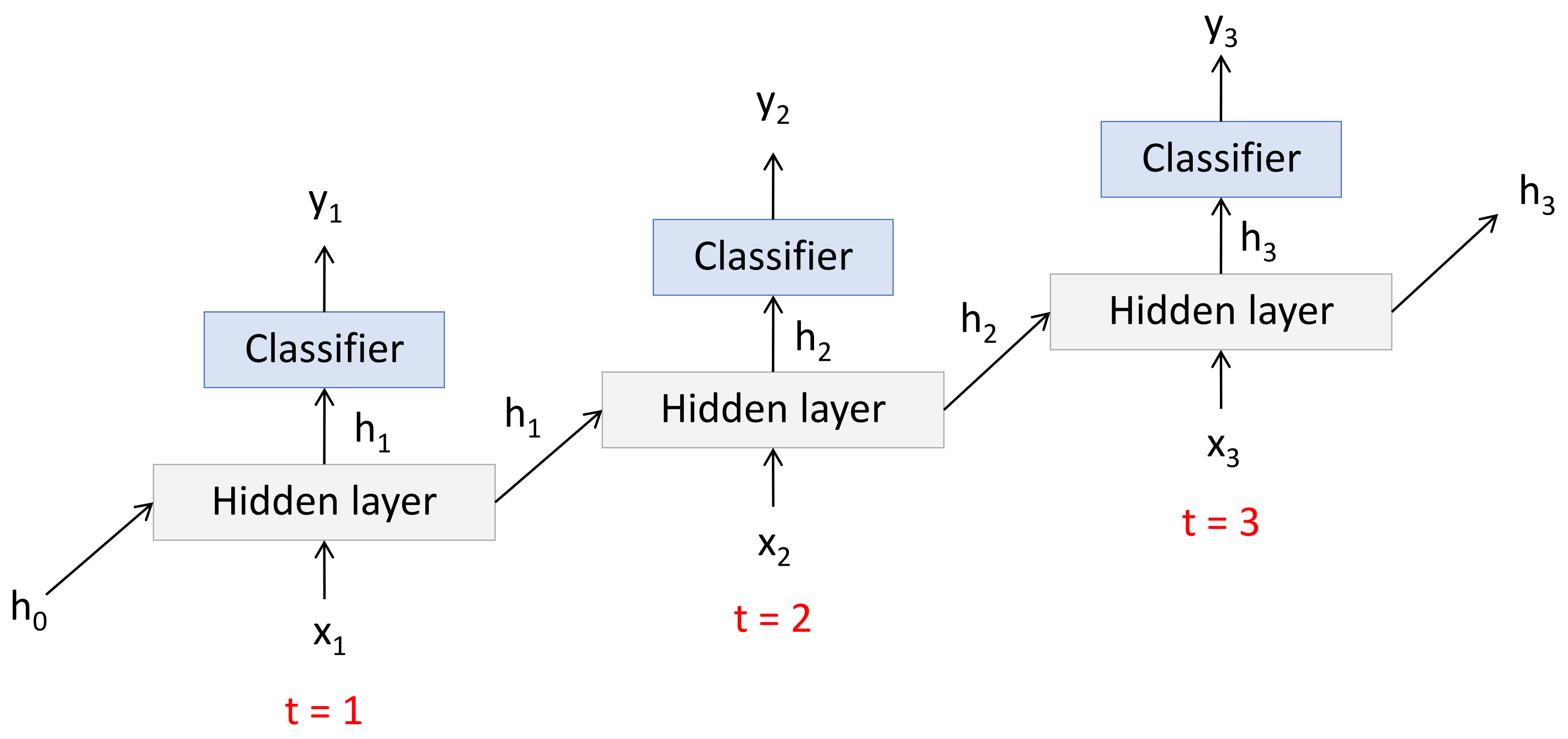
Recurrent Neural Network (RNN)

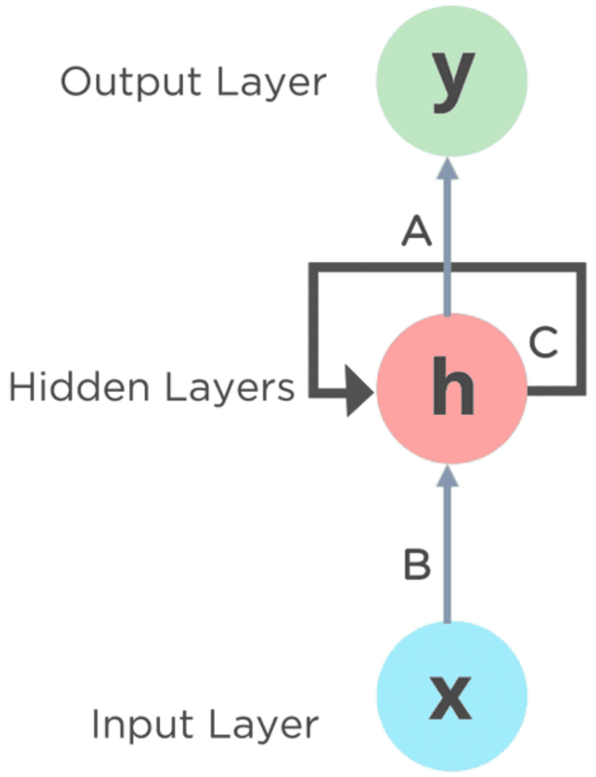


Recurrence:

$$\underbrace{h_t}_{\text{new state}} = \underbrace{f_W}_{\text{function of W}}(\underbrace{x_t}_{\text{input at time t}}, \underbrace{h_{t-1}}_{\text{old state}})$$

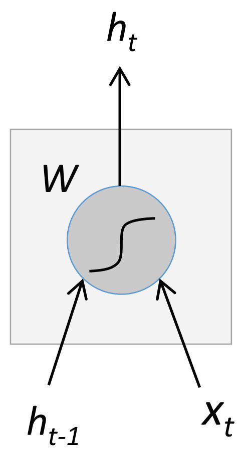
Unrolling the RNN





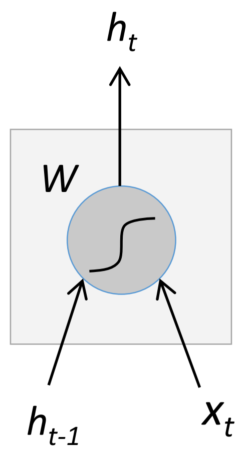
A, B and C are the parameters

Vanilla RNN Cell

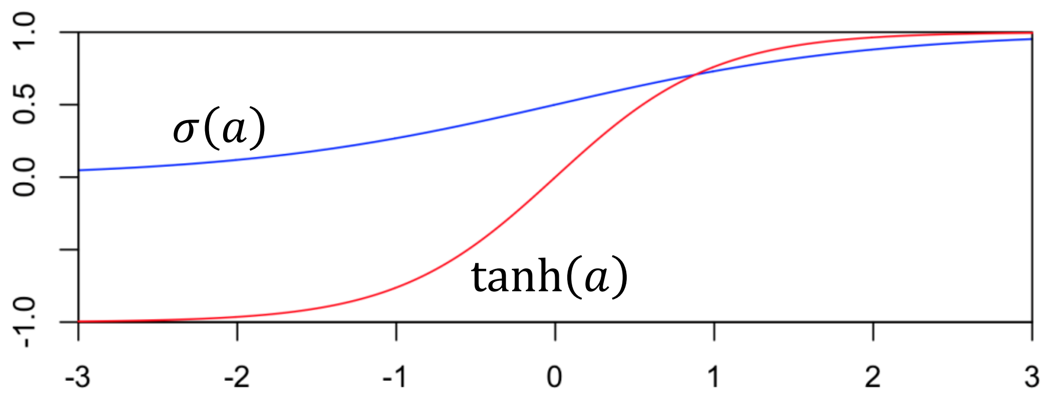


$$h_t = y_W(x_t, h_{t-1})$$
$$= \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

Vanilla RNN Cell



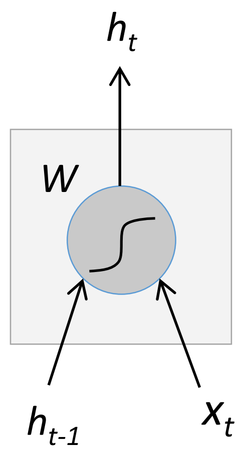
$$h_t = f_W(x_t, h_{t-1})$$
$$= \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$



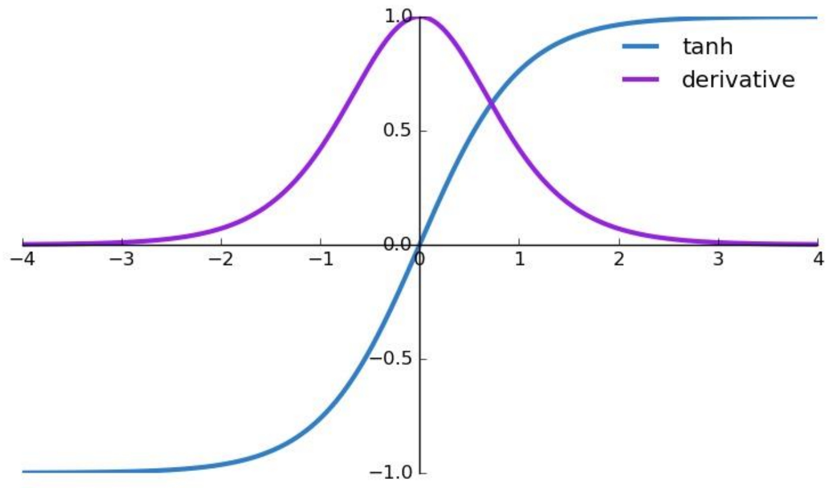
$$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$
$$= 2\sigma(2a) - 1$$

Image source

Vanilla RNN Cell



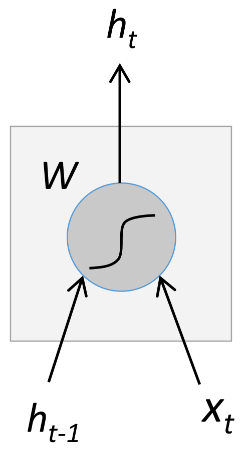
$$h_t = f_W(x_t, h_{t-1})$$
$$= \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$



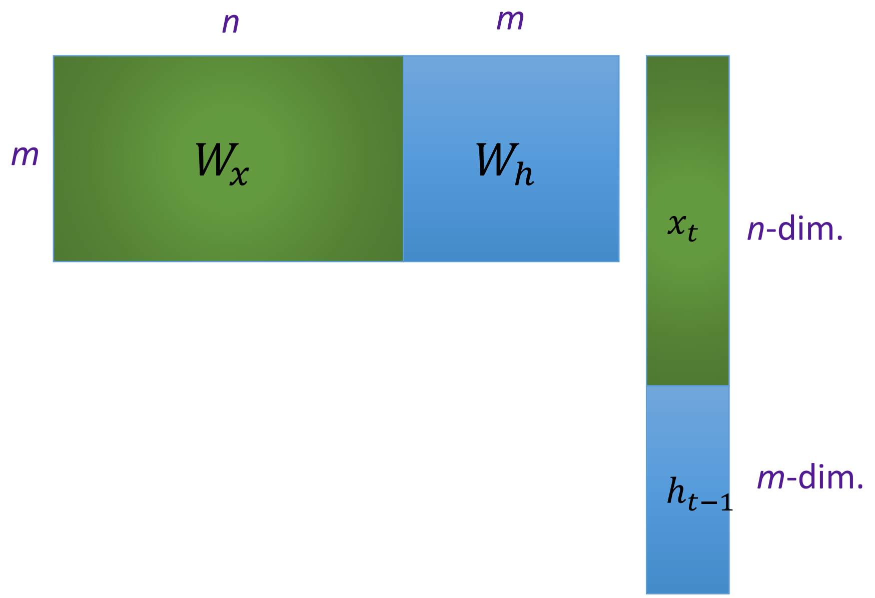
$$\frac{d}{da}\tanh(a) = 1 - \tanh^2(a)$$

Image source

Vanilla RNN Cell



$$
\begin{aligned}
h_t &= f_W(x_t, h_{t-1}) \\
&= \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} \\
&= \tanh(W_x x_t + W_h h_{t-1})
\end{aligned}
$$



Formulation

h(t) = σ (W h(t-1) + U x(t))

hat(y) (t) = softmax(V h(t))

$$\hat{P}(x_{t+1} = v_j \mid x_t, \ldots, x_1) \quad = \quad \hat{y}_{t,j}$$

Encodes the long-term dependency

$$J^{(t)}(\theta) = -\sum_{j=1}^{|V|} y_{t,j} \log \hat{y}_{t,j}$$

The loss function to be minimized

- ✓ Model for Many to Many prediction
- ✓ One output per time stamp
- ✓ |V| denotes the number of words
- ✓ Training with softmax cross-entropy
- ✓ **The weights are shared for all the time-stamps**

Training is **Back-Propagation Through Time**

How to define the classifier

- There could be multiple possibilities for defining the classifier:
  - Ensemble of classifiers for all the states?
  - Single classifier at the final state?

- It is application dependent. Let us see two examples for sentiment classification and captioning

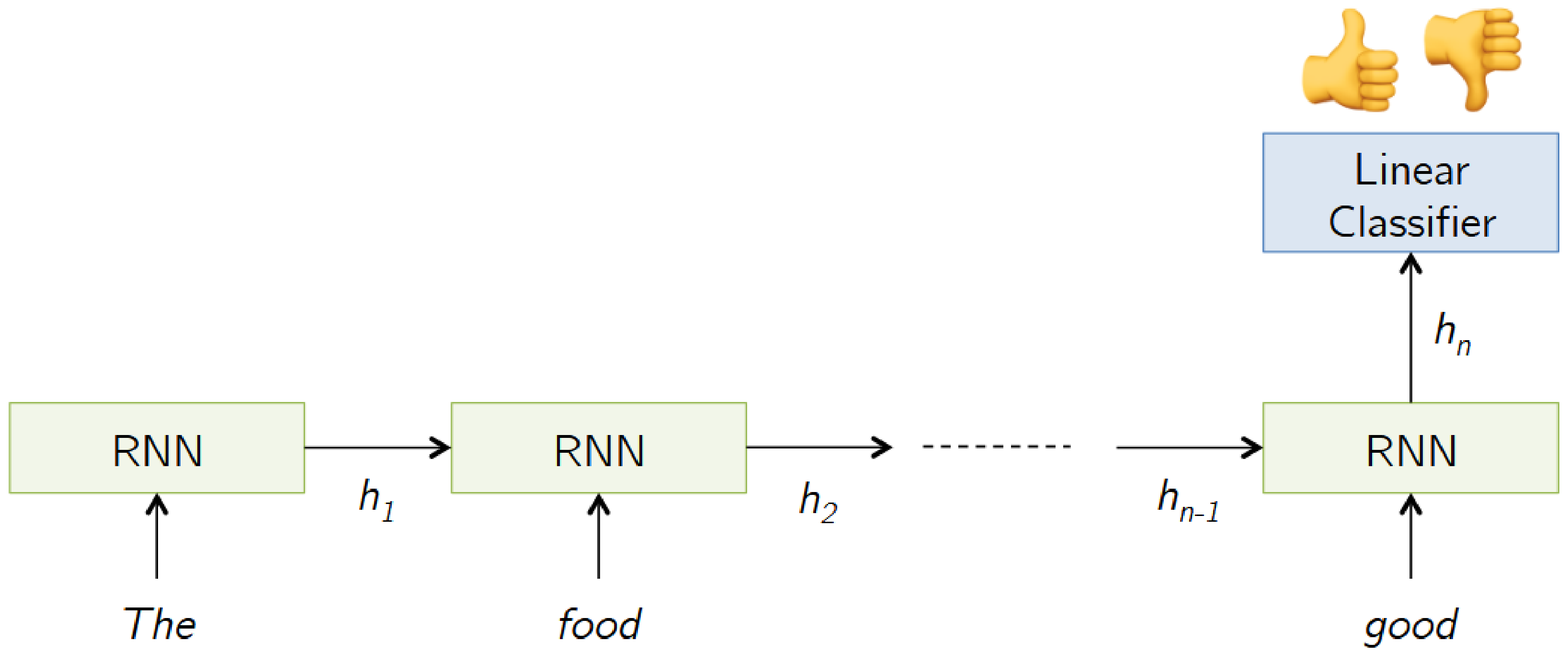
Example – sentiment classification

- Classify a
  restaurant review from Yelp! OR
  movie review from IMDB OR
  ...
  as positive or negative

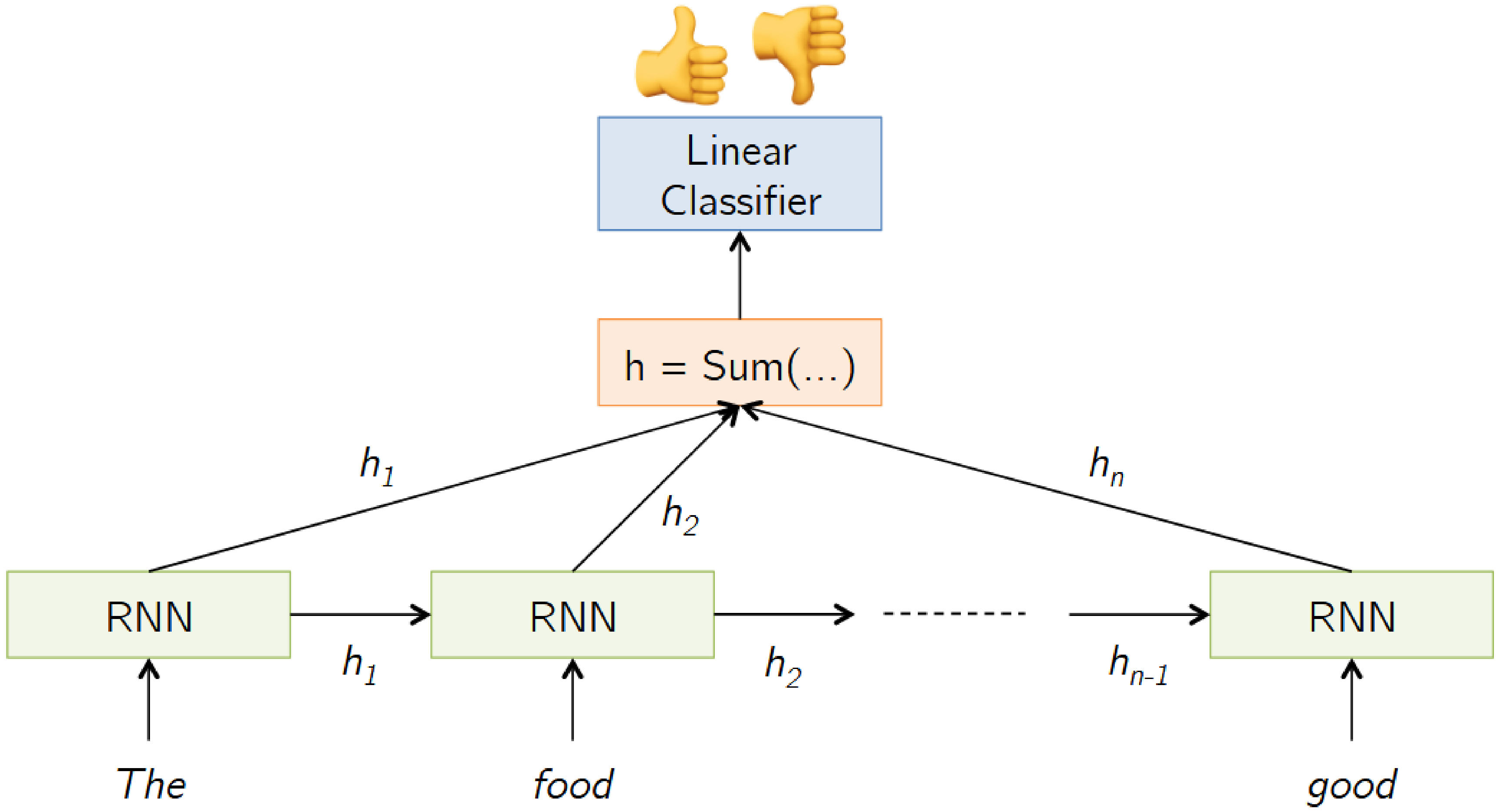
- **Inputs:** Multiple words, one or more sentences
- **Outputs:** Positive / Negative classification

- “The food was really good”
- “The chicken crossed the road because it was uncooked”

Model-1



Model-2



Linear Classifier
h = Sum(...)
h_1
h_2
h_n
RNN
RNN
RNN
h_1
h_2
h_{n-1}
The
food
good

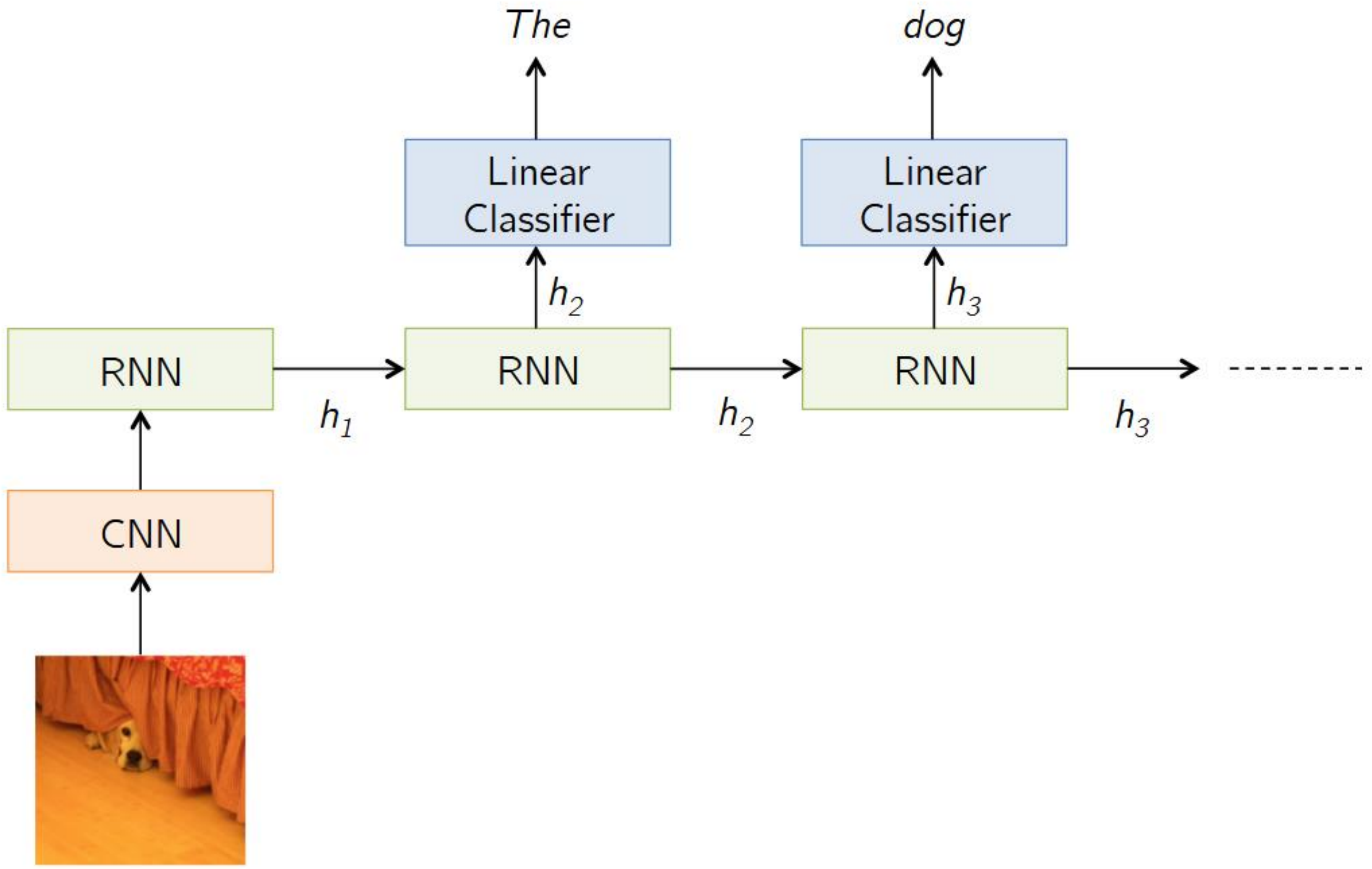
Image captioning

- Given an image, produce a sentence describing its contents

- **Inputs:** Image feature (from a CNN)
- **Outputs:** Multiple words (let's consider one sentence)

: The dog is hiding

Model


The
dog
Linear Classifier
Linear Classifier
h_2
h_3
RNN
RNN
RNN
h_1
h_2
h_3
CNN

A person riding a motorcycle on a dirt road.

Two dogs play in the grass.

A herd of elephants walking across a dry grass field.

A group of young people playing a game of frisbee.

Two hockey players are fighting over the puck.

A close up of a cat laying on a couch.

Show and Tell: A Neural Image Caption Generator, CVPR 15

Some highlights

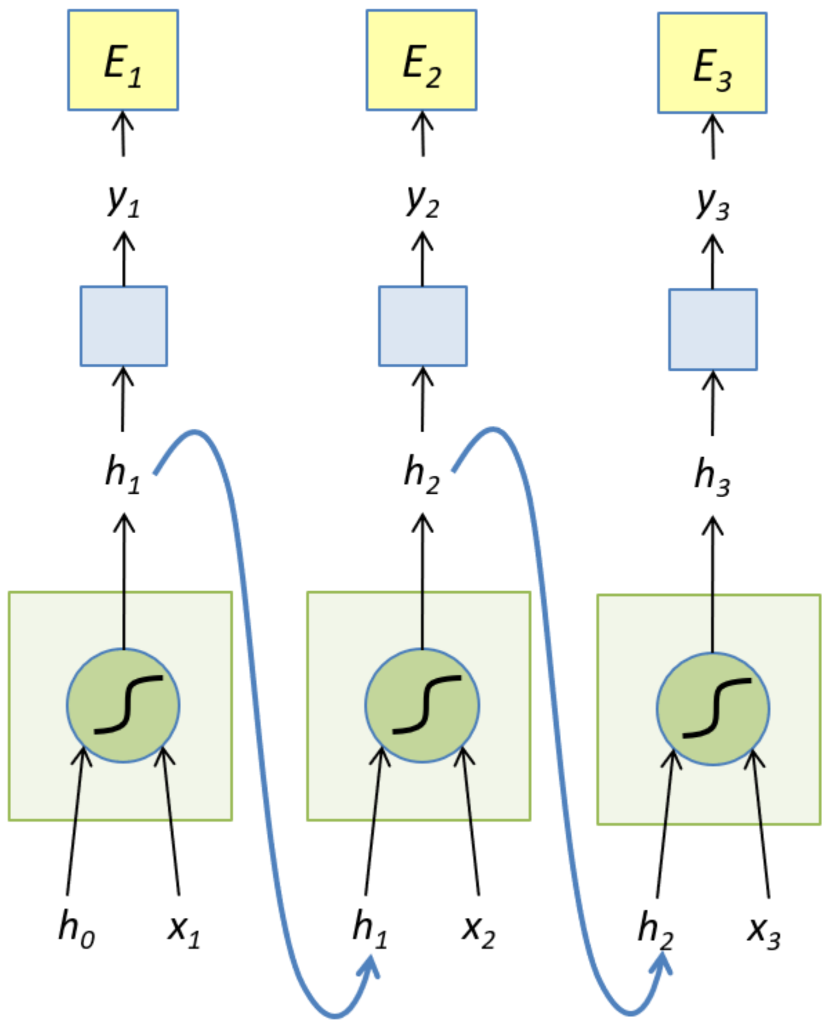
- RNN takes the previous output or hidden state as inputs at every time together with the usual input. This means that the composite input at time t has some historical information about the happenings at time T < t.

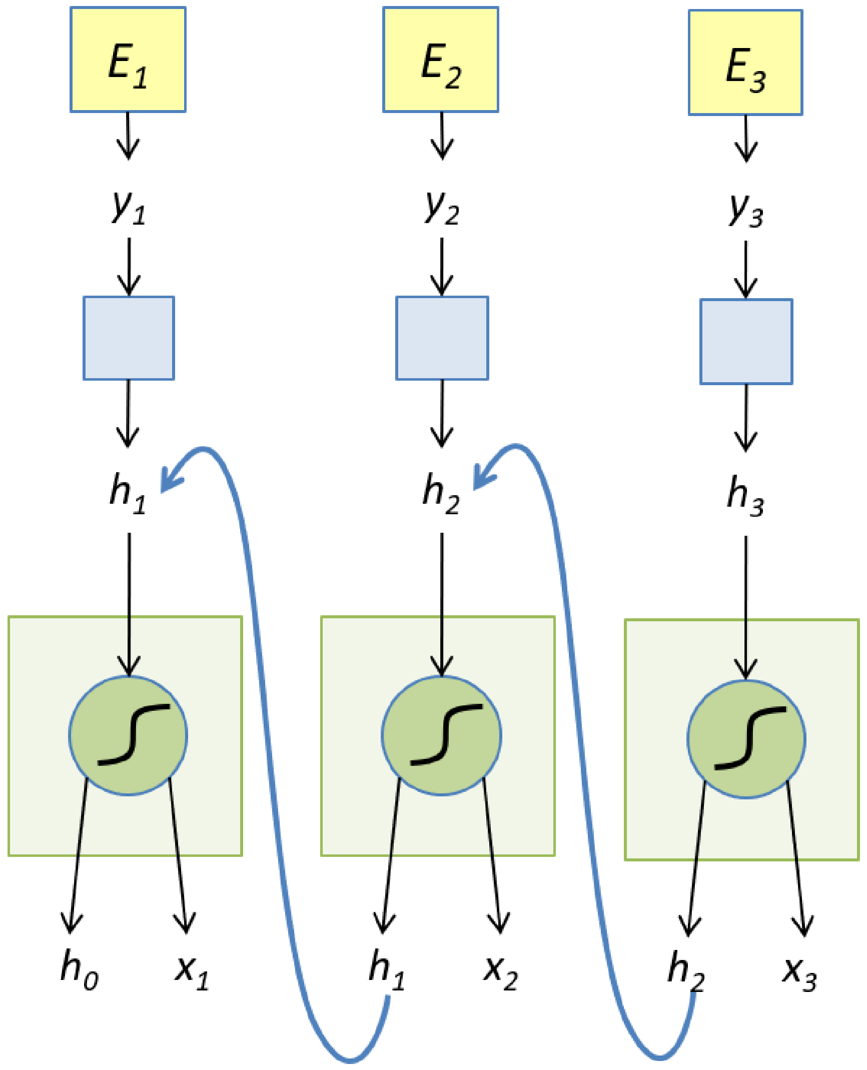
- RNNs gained popularity in sequence modeling without forgetting important past information as their intermediate values (state) can store information about the past inputs for a time that is not fixed a priori.

Back propagation through time

- Considered to be the standard method used to train RNNs.
- The unfolded network (used during forward pass) is treated as one feed-forward network that accepts the whole time series as input.
- The weight updates are computed for each copy in the unfolded network, then accumulated and applied to the model weights and biases.

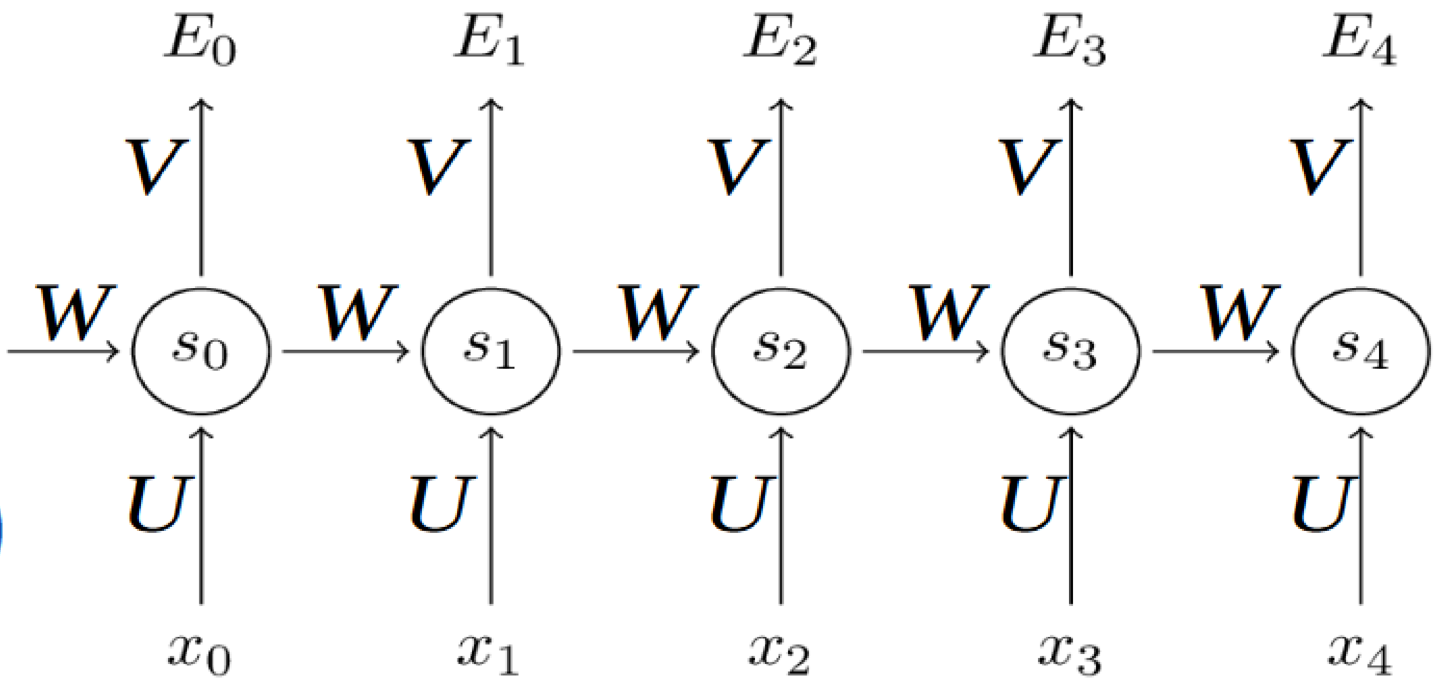
Forward and backward pass





$$s_t = \tanh(Ux_t + Ws_{t-1})$$
$$\hat{y}_t = softmax(Vs_t)$$

$$E(y, \hat{y}) = -\sum_t E_t(y_t, \hat{y}_t)$$



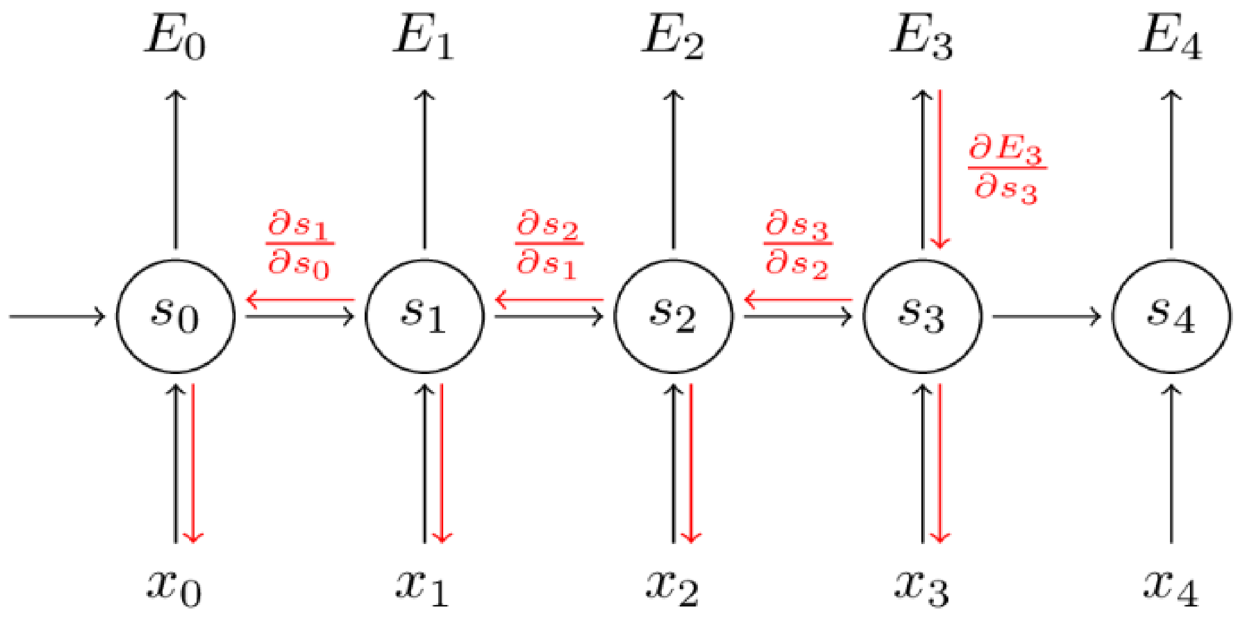
$$\frac{\partial E}{\partial \boldsymbol{W}} = \sum_t \frac{\partial E_t}{\partial \boldsymbol{W}}$$

$$\frac{\partial E_3}{\partial \boldsymbol{W}} = \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial \boldsymbol{W}}$$

But $s_3 = \tanh\left(Ux_t + Ws_2\right)$

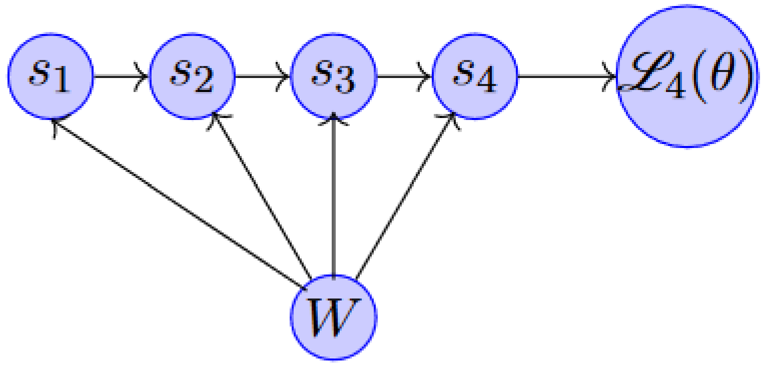
S_3 depends on s_2, which depends on W and s_1, and so on.

$$\frac{\partial E_3}{\partial \boldsymbol{W}} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial \boldsymbol{W}}$$


E_0
E_1
E_2
E_3
E_4
$\frac{\partial E_3}{\partial s_3}$
$\frac{\partial s_1}{\partial s_0}$
$\frac{\partial s_2}{\partial s_1}$
$\frac{\partial s_3}{\partial s_2}$
s_0
s_1
s_2
s_3
s_4
x_0
x_1
x_2
x_3
x_4

$$\frac{\partial E_3}{\partial \boldsymbol{W}} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial \boldsymbol{W}}$$

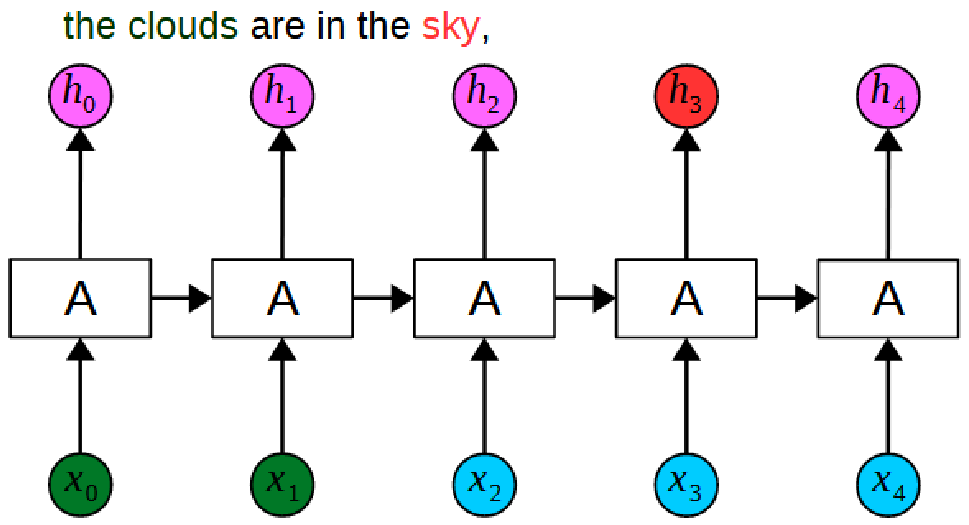
$$\frac{\partial E_3}{\partial \boldsymbol{W}} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \left(\prod_{j=k+1}^{3} \frac{\partial s_j}{\partial s_{j-1}} \right) \frac{\partial s_k}{\partial \boldsymbol{W}}$$



- Derivative of a vector w.r.t a vector is a matrix called jacobian
- 2-norm of the above Jacobian matrix has an upper bound of 1
- **tanh** maps all values into a range between -1 and 1, and the **derivative is bounded by 1**
- With multiple matrix multiplications, gradient values **shrink exponentially**
- Gradient contributions from “far away” steps become zero
- Depending on activation functions and network parameters, gradients could **explode** instead of vanishing

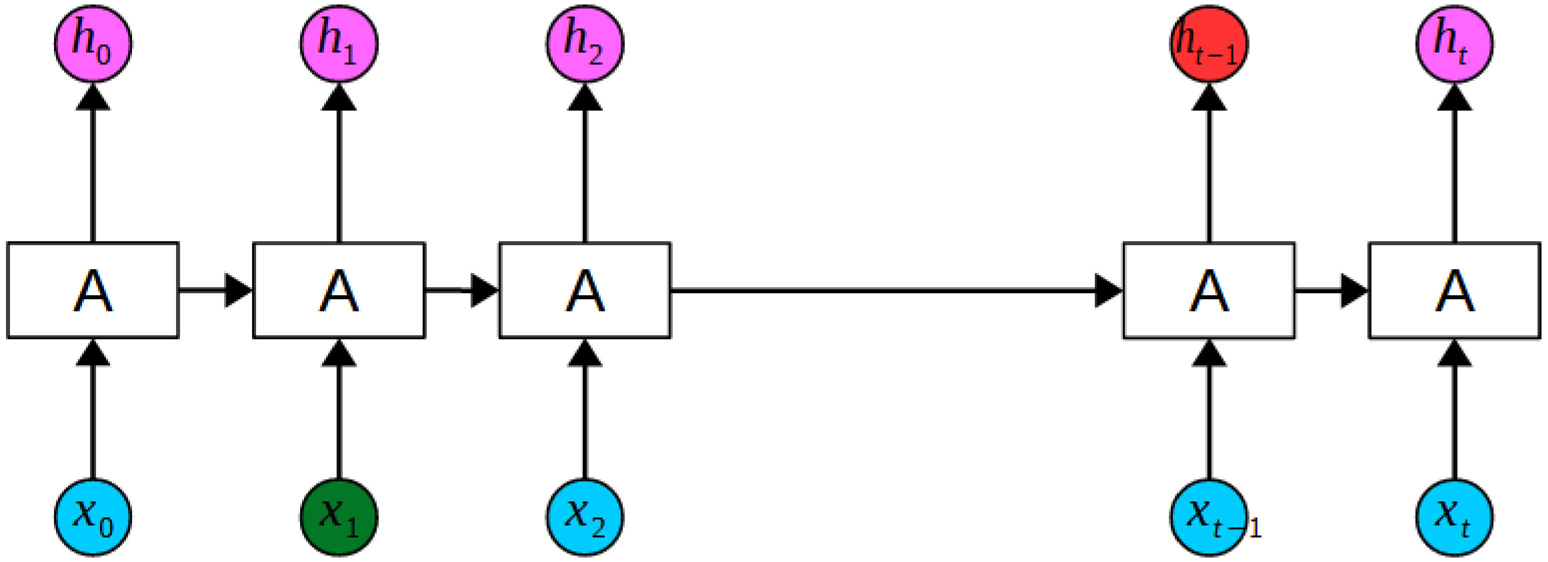
Short term dependency

Language model trying to predict the next word based on the previous ones



Long term dependency

I grew up in India… I speak fluent Hindi.



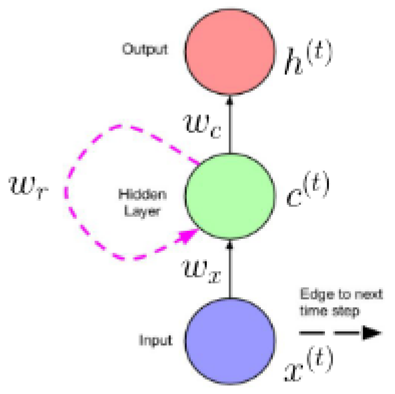
RNN fails to preserve the long term dependency due to the vanishing gradient problem

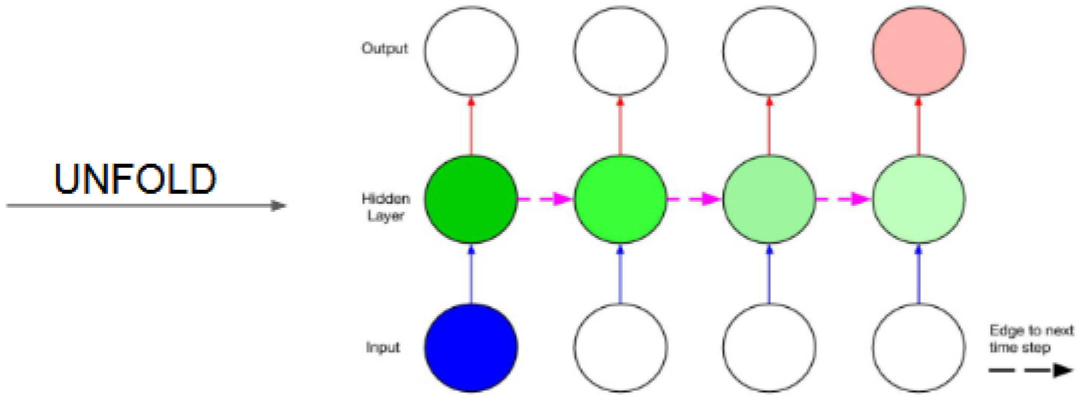
We should distinguish between what to remember and what to forget more wisely

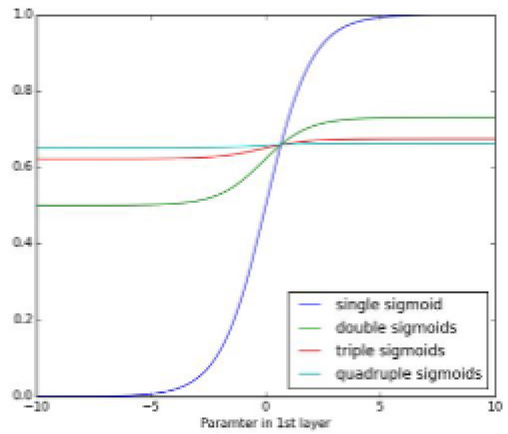
LSTM (Long short term memory) to the rescue

The RNN problem revisited

- Problem: can’t capture long-term dependencies due to vanishing/exploding gradients during backpropagation





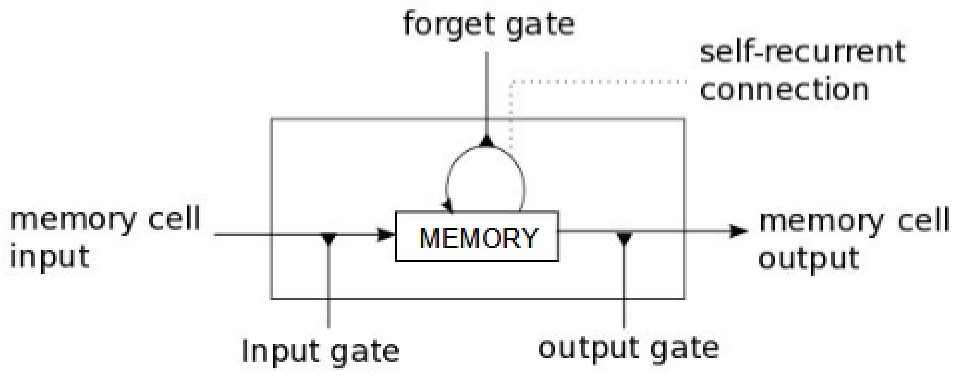


$$h^{(t)} = \sigma(w_c \cdot c^{(t)})$$
$$c^{(t)} = \sigma(w_r \cdot c^{(t-1)} + w_x \cdot x^{(t)})$$

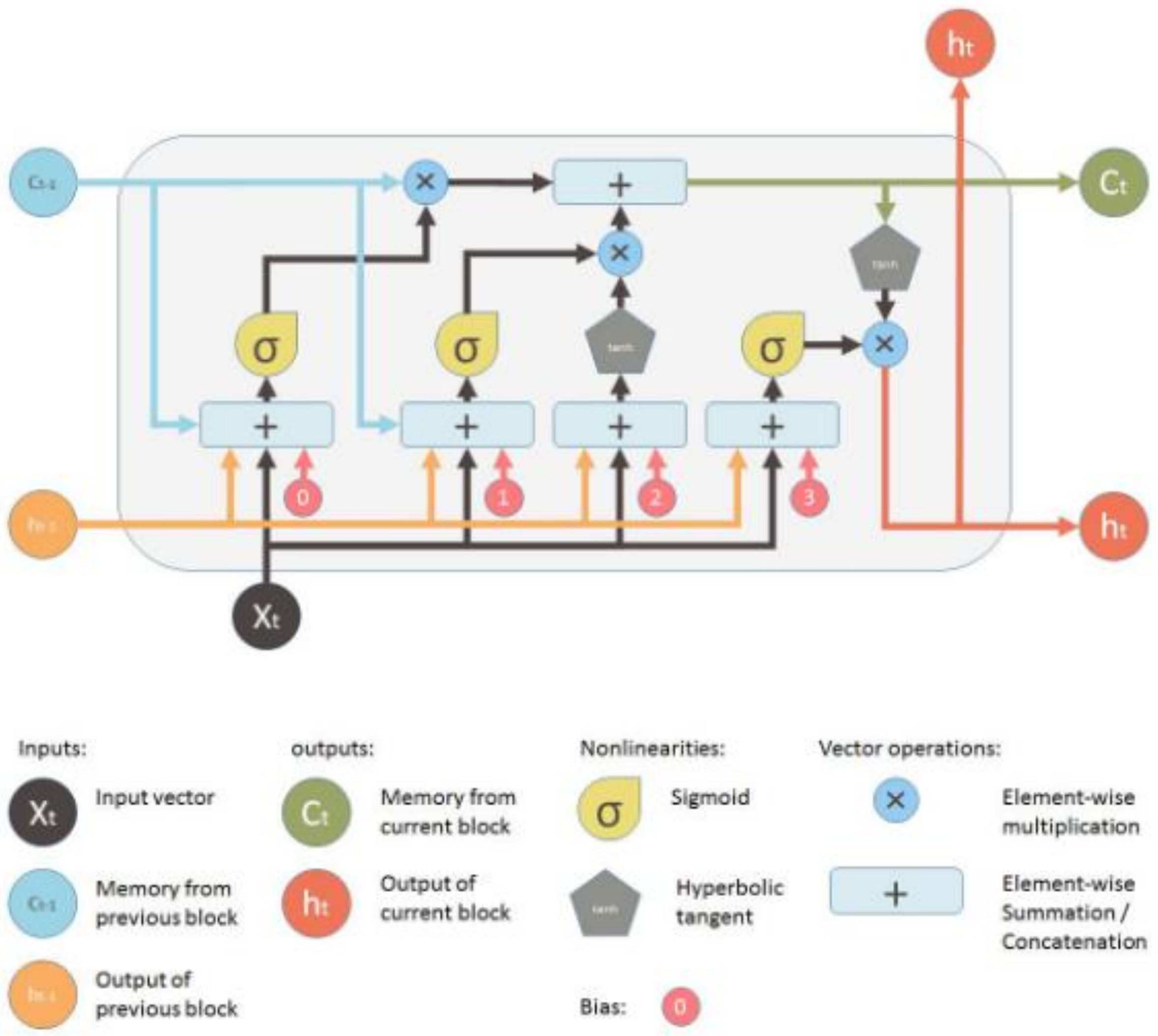
$$\begin{aligned} h^{(3)} &= \sigma(w_c \cdot c^{(3)}) \\ &= \sigma(w_c \cdot \sigma(w_x \cdot x^{(3)} + w_r \cdot c^{(2)})) \\ &= \sigma(w_c \cdot \sigma(w_x \cdot x^{(3)} + w_r \cdot \sigma(w_x \cdot x^{(2)} + w_r \cdot c^{(1)})))))) \\ &= \sigma(w_c \cdot \sigma(w_x \cdot x^{(3)} + w_r \cdot \sigma(w_x \cdot x^{(2)} + w_r \cdot \sigma(w_x \cdot x^{(1)} + w_r \cdot c^{(0)}))))))) \end{aligned}$$

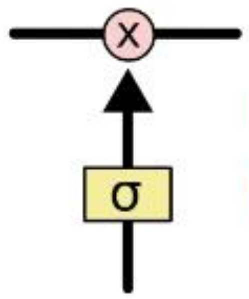
LSTM

Central Idea: A **memory cell** (interchangeably **block**) which can maintain its state over time, consisting of an explicit memory (aka the **cell state vector**) and **gating units** which regulate the information flow into and out of the memory.

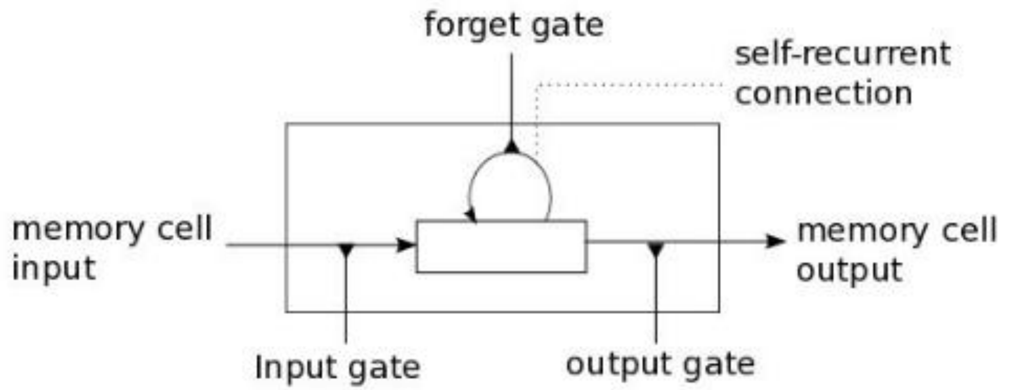


LSTM Memory Cell





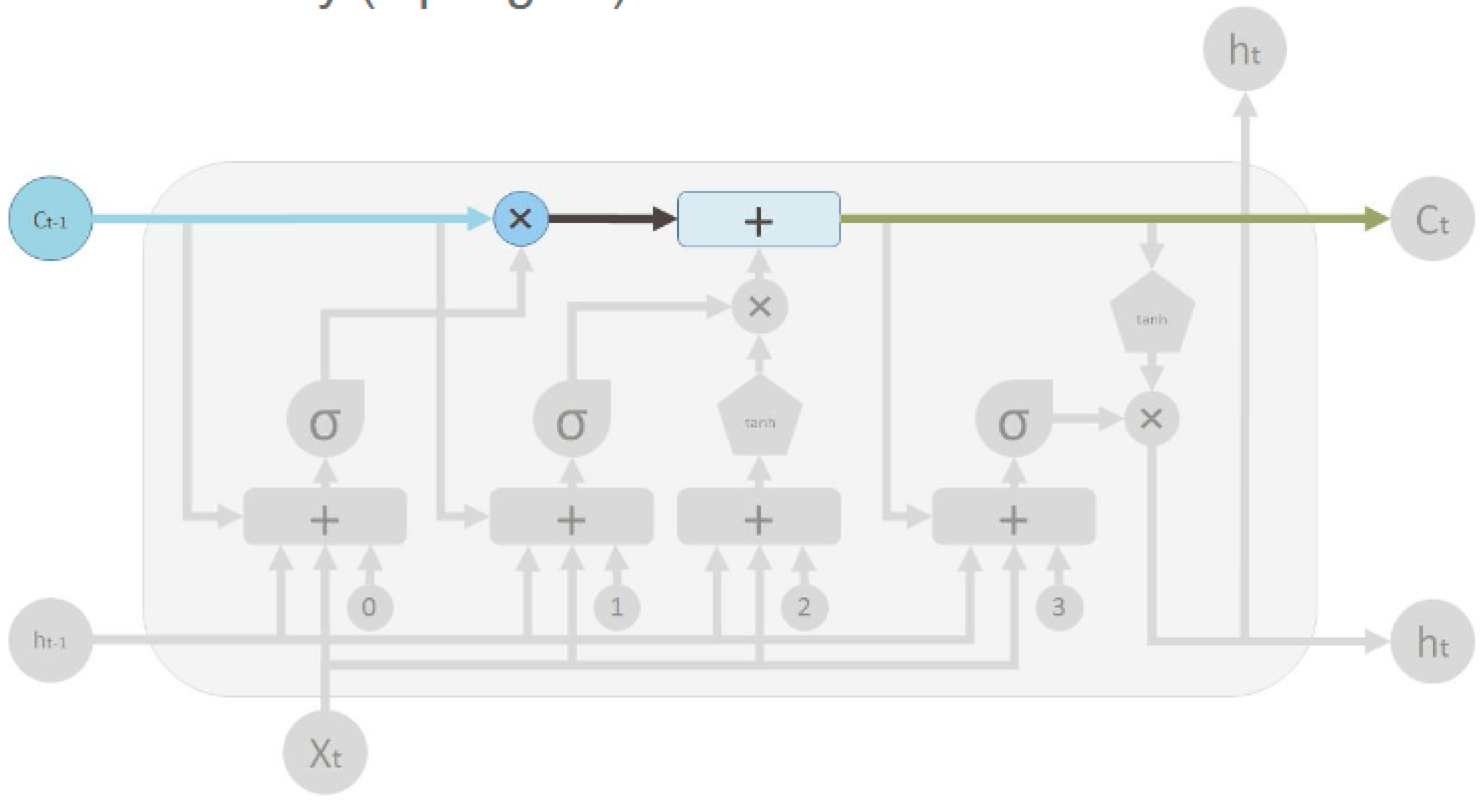
Gate (sigmoid layer followed by pointwise multiplication)

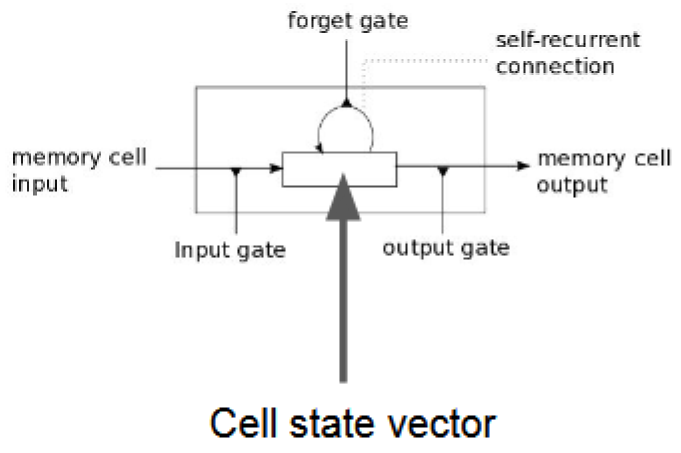


Simplified schematic for reference

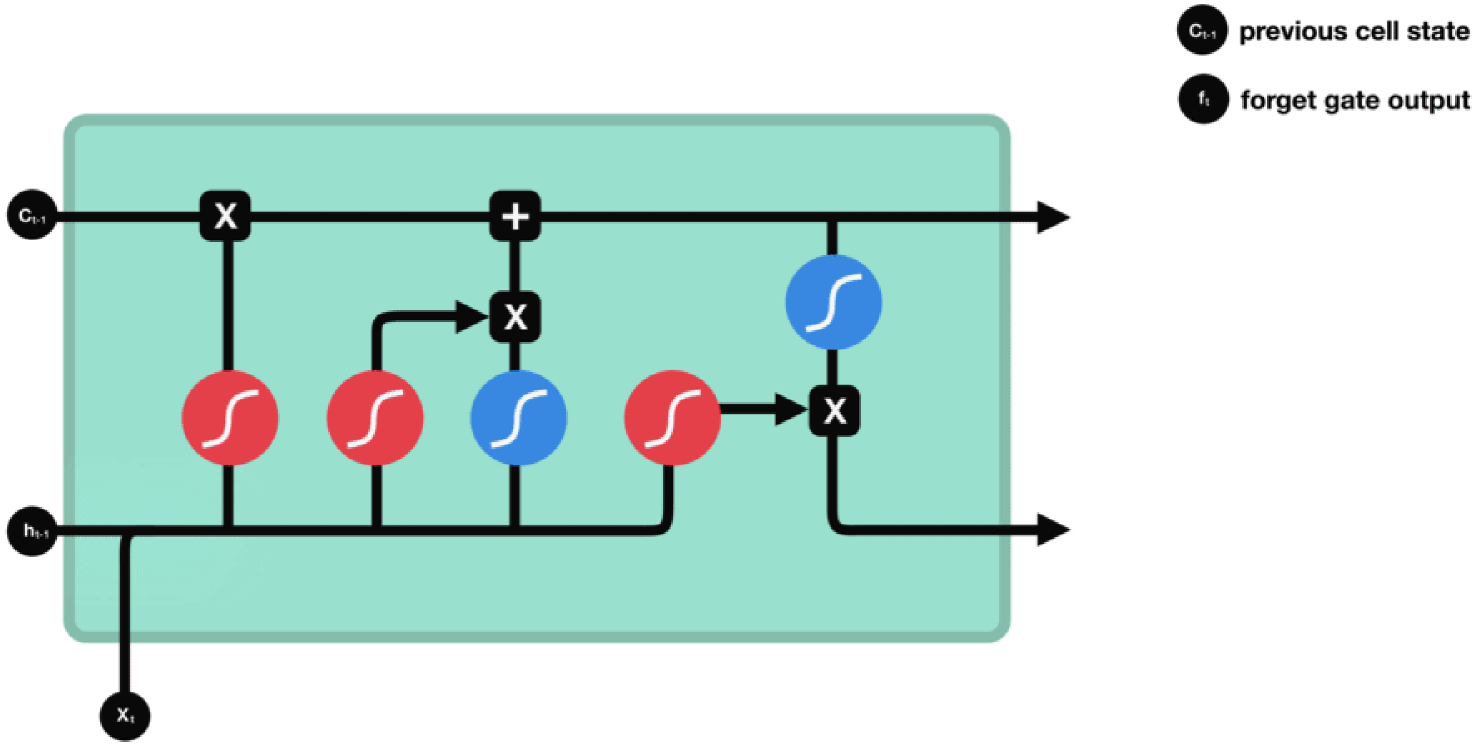
Cell state vector

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



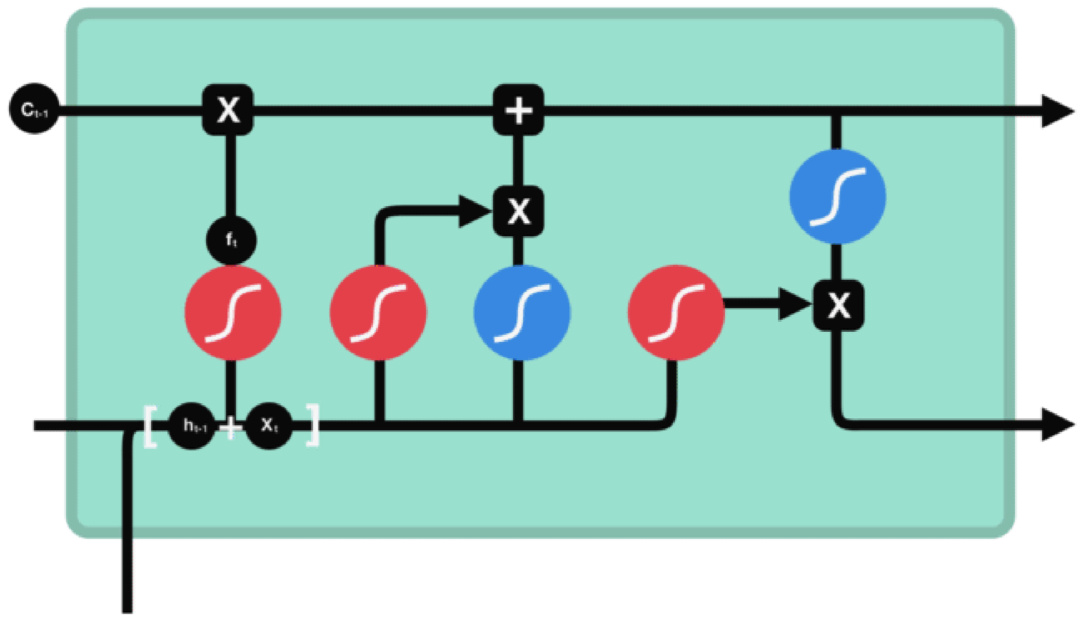


Forget gate



This gate decides what information should be thrown away or kept. Information from the previous hidden state and information from the current input is passed through the sigmoid function. Values come out between 0 and 1. The closer to 0 means to forget, and the closer to 1 means to keep.

Input gate



C_{t-1} previous cell state

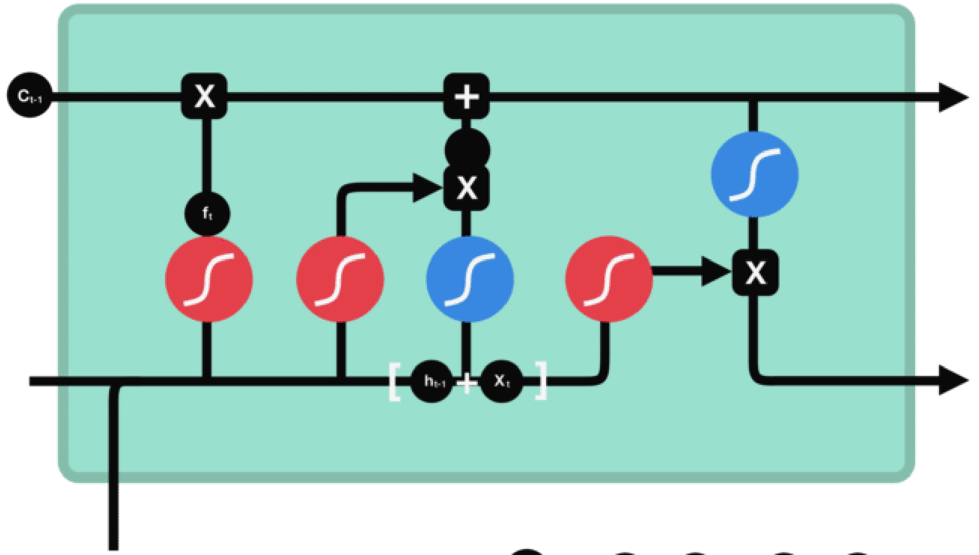
f_t forget gate output

i_t input gate output

$\tilde{C}_t$ candidate

First, we pass the previous hidden state and current input into a sigmoid function. That decides which values will be updated by transforming the values to be between 0 and 1. 0 means not important, and 1 means important. We also pass the hidden state and current input into the tanh function to squish values between -1 and 1 to help regulate the network. Then we multiply the tanh output with the sigmoid output. The sigmoid output will decide which information is important to keep from the tanh output.

Updating the cell state

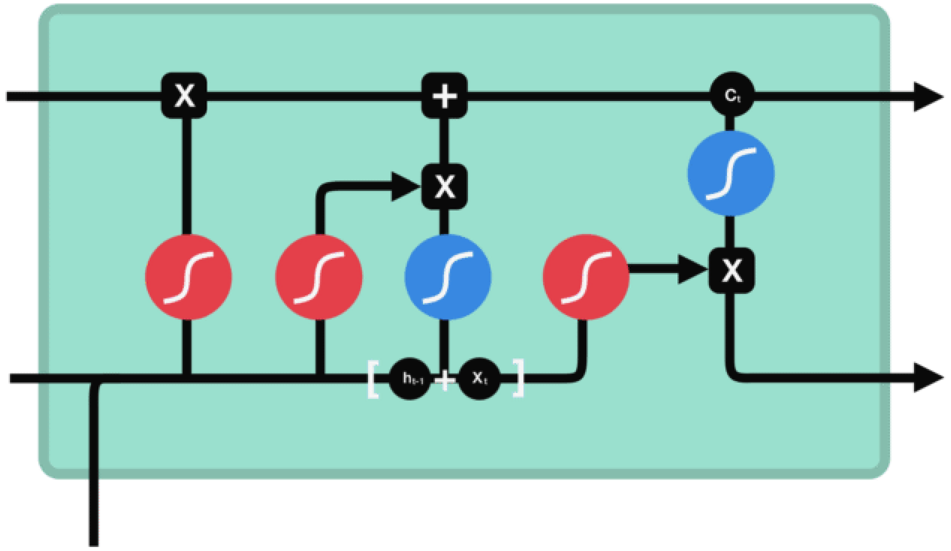


- c_{t-1} previous cell state
- f_t forget gate output
- i_t input gate output
- $\tilde{c}_t$ candidate
- c_t new cell state

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t$$

First, the cell state gets pointwise multiplied by the forget vector. This has a possibility of dropping values in the cell state if it gets multiplied by values near 0. Then we take the output from the input gate and do a pointwise addition which updates the cell state to new values that the neural network finds relevant. That gives us our new cell state.

Output gate



c_{t-1}	previous cell state
f_t	forget gate output
i_t	input gate output
$\tilde{c}_t$	candidate
c_t	new cell state
o_t	output gate output
h_t	hidden state

First, we pass the previous hidden state and the current input into a sigmoid function. Then we pass the newly modified cell state to the tanh function. We multiply the tanh output with the sigmoid output to decide what information the hidden state should carry. The output is the hidden state.

Summarizing the equations

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

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

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

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

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

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

Training LSTM

- Backpropagation through time

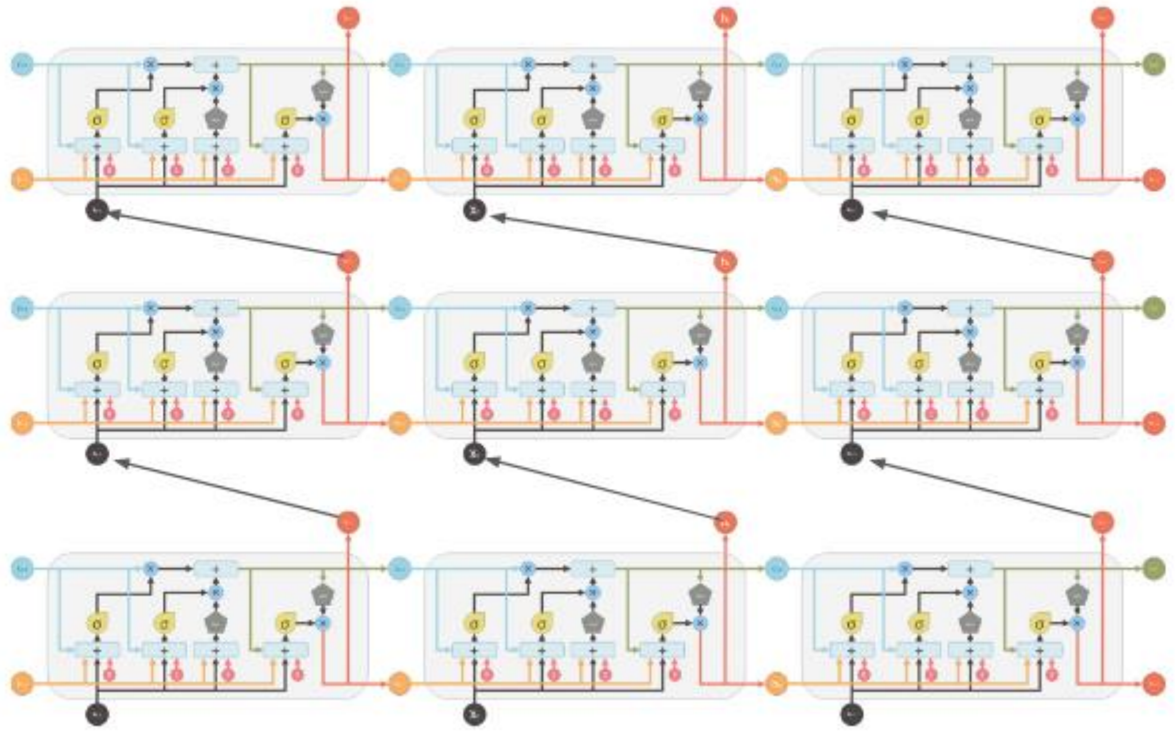
- Each cell has many parameters (W_f, W_i, W_C, W_o)
  - Generally requires lots of training data.
  - Requires lots of compute time that exploits GPU clusters.

Deep LSTM

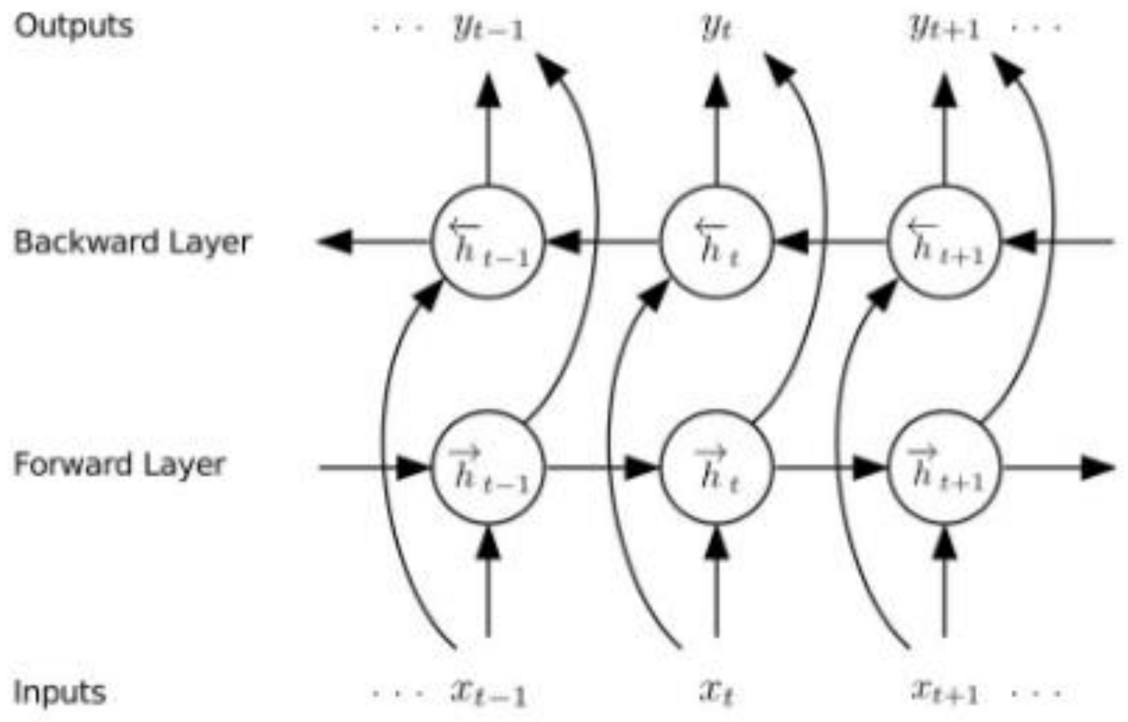
- Deep LSTMs can be created by stacking multiple LSTM layers vertically, with the output sequence of one layer forming the input sequence of the next (in addition to recurrent connections within the same layer)

- Increases the number of parameters - but given sufficient data, performs significantly better than single-layer LSTMs (Graves et al. 2013)

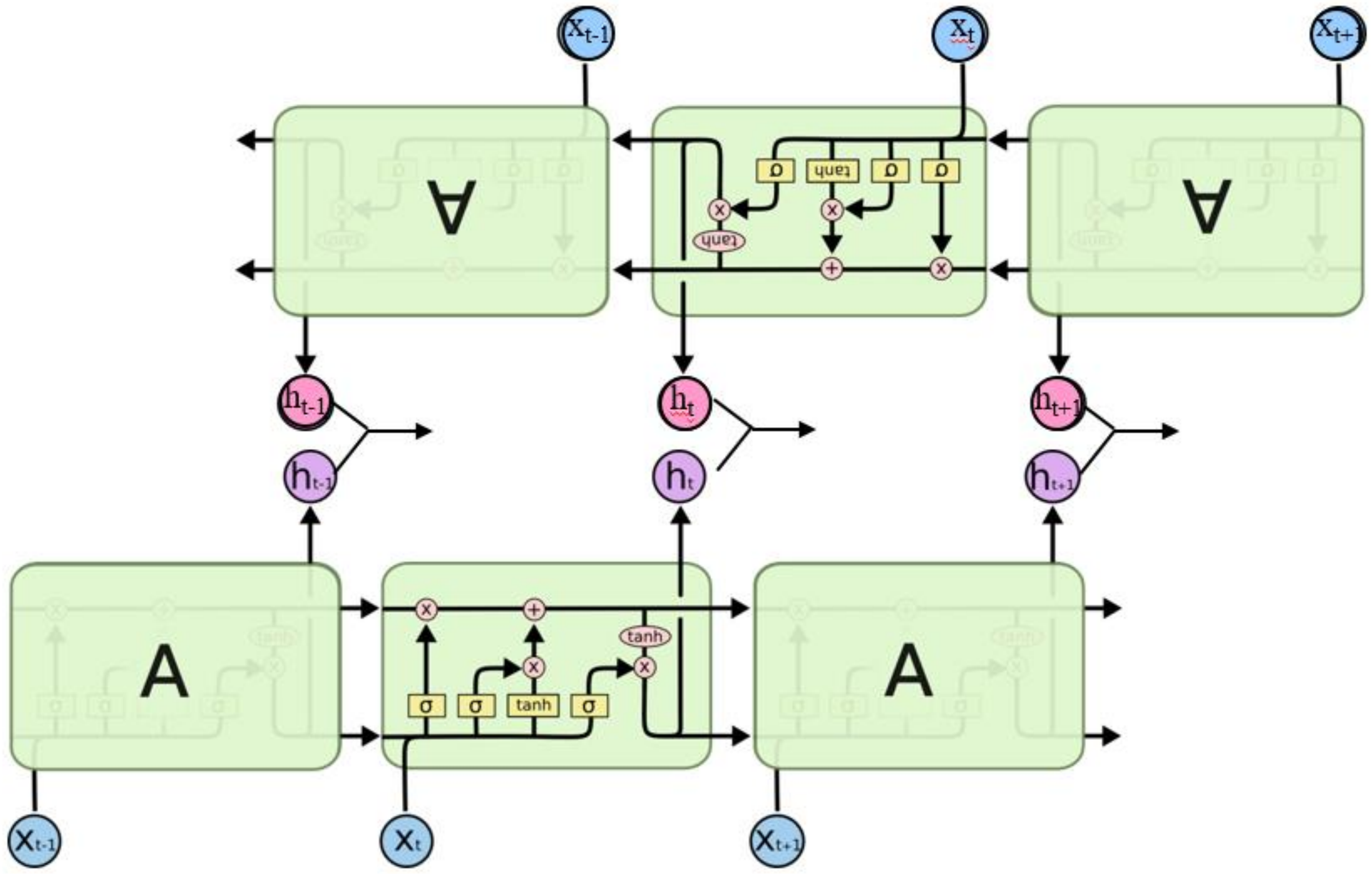
- Dropout usually applied only to non-recurrent edges, including between layers



Bi-directional recurrent model

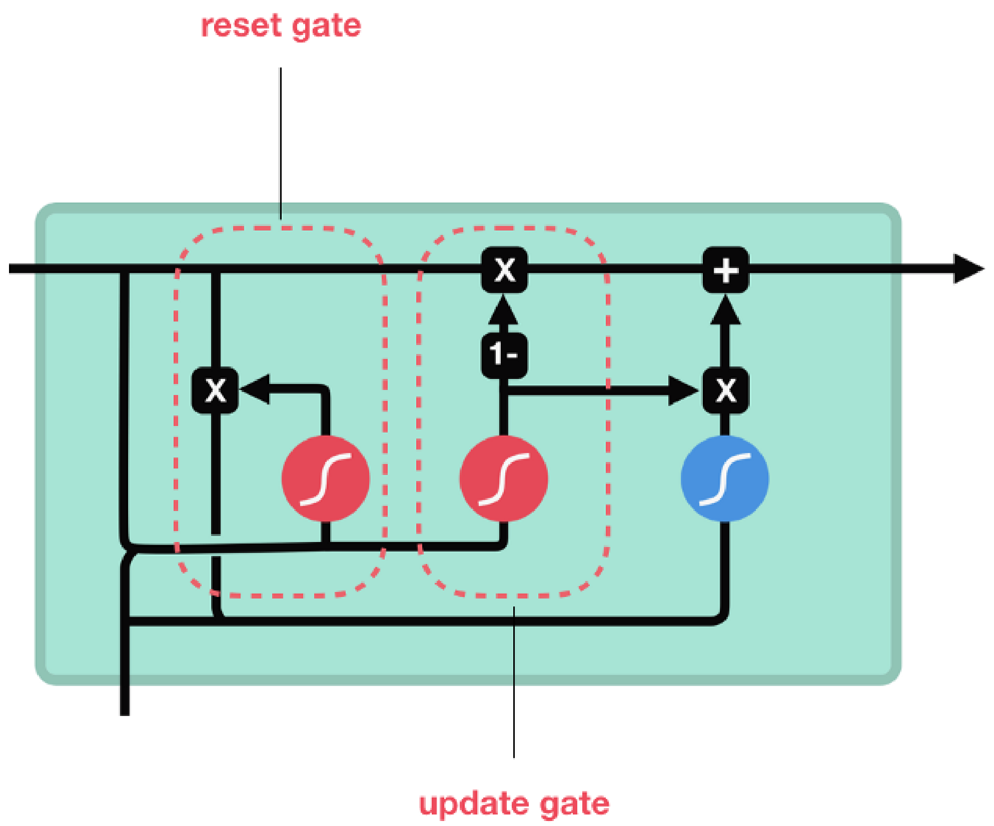


Bi-LSTM



x_{t-1}
x_t
x_{t+1}
A
A
A
A
σ
σ
tanh
σ
tanh
h_{t-1}
h_t
h_{t+1}
h_{t-1}
h_t
h_{t+1}
σ
σ
tanh
σ
tanh
x_{t-1}
x_t
x_{t+1}

Gated recurrent units



GRU’s got rid of the cell state and used the hidden state to transfer information. It also only has two gates, a reset gate and update gate.

Update Gate
The update gate acts similar to the forget and input gate of an LSTM. It decides what information to throw away and what new information to add.
Reset Gate
The reset gate is another gate is used to decide how much past information to forget.