Inspire...Educate...Transform.

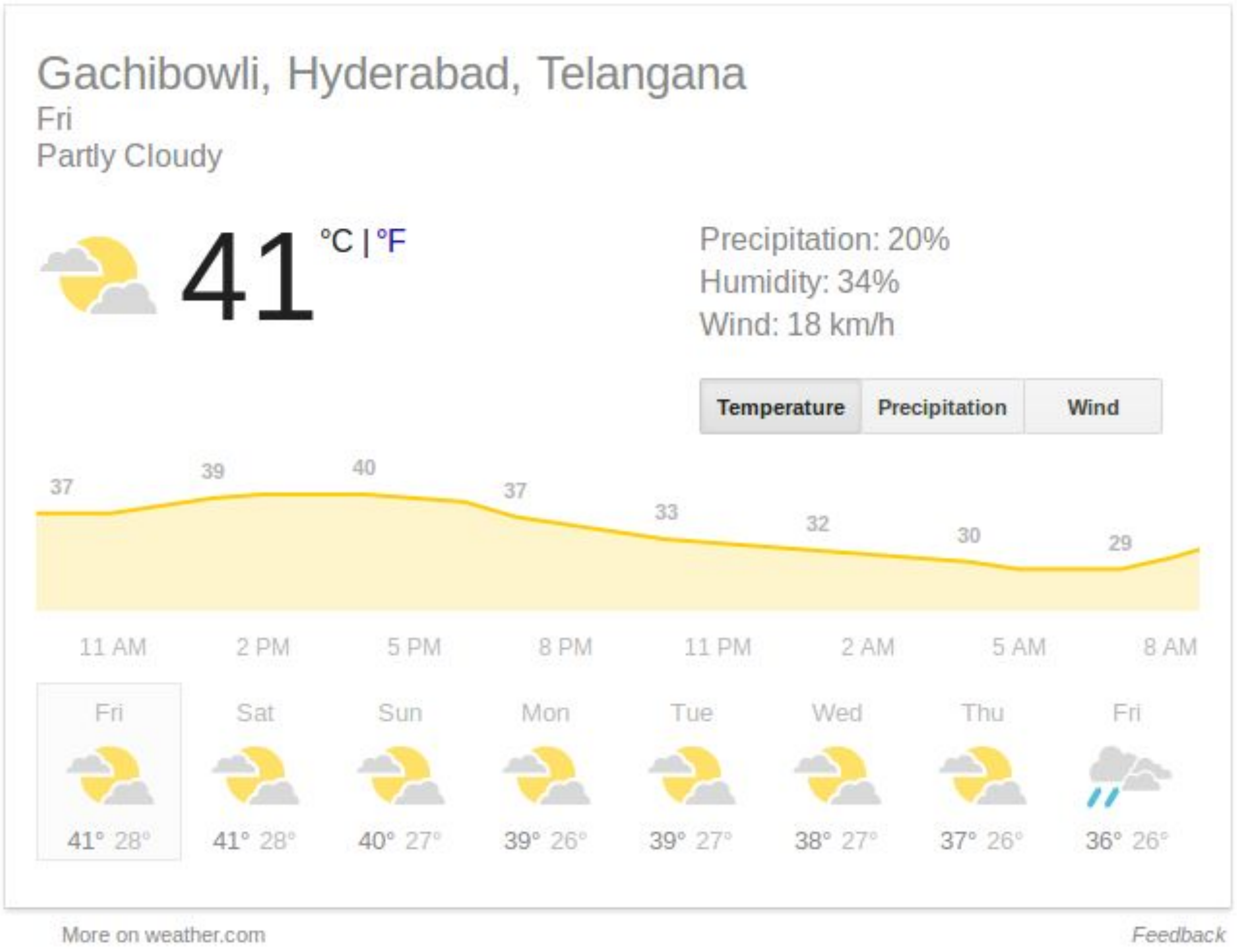
Inspire…Educate…Transform.

Recurrent Neural Networks

Dr. Kishore Reddy Konda

Mentor, International School of Engineering

Weather report



Gachibowli, Hyderabad, Telangana
Fri
Partly Cloudy

41 °C | °F

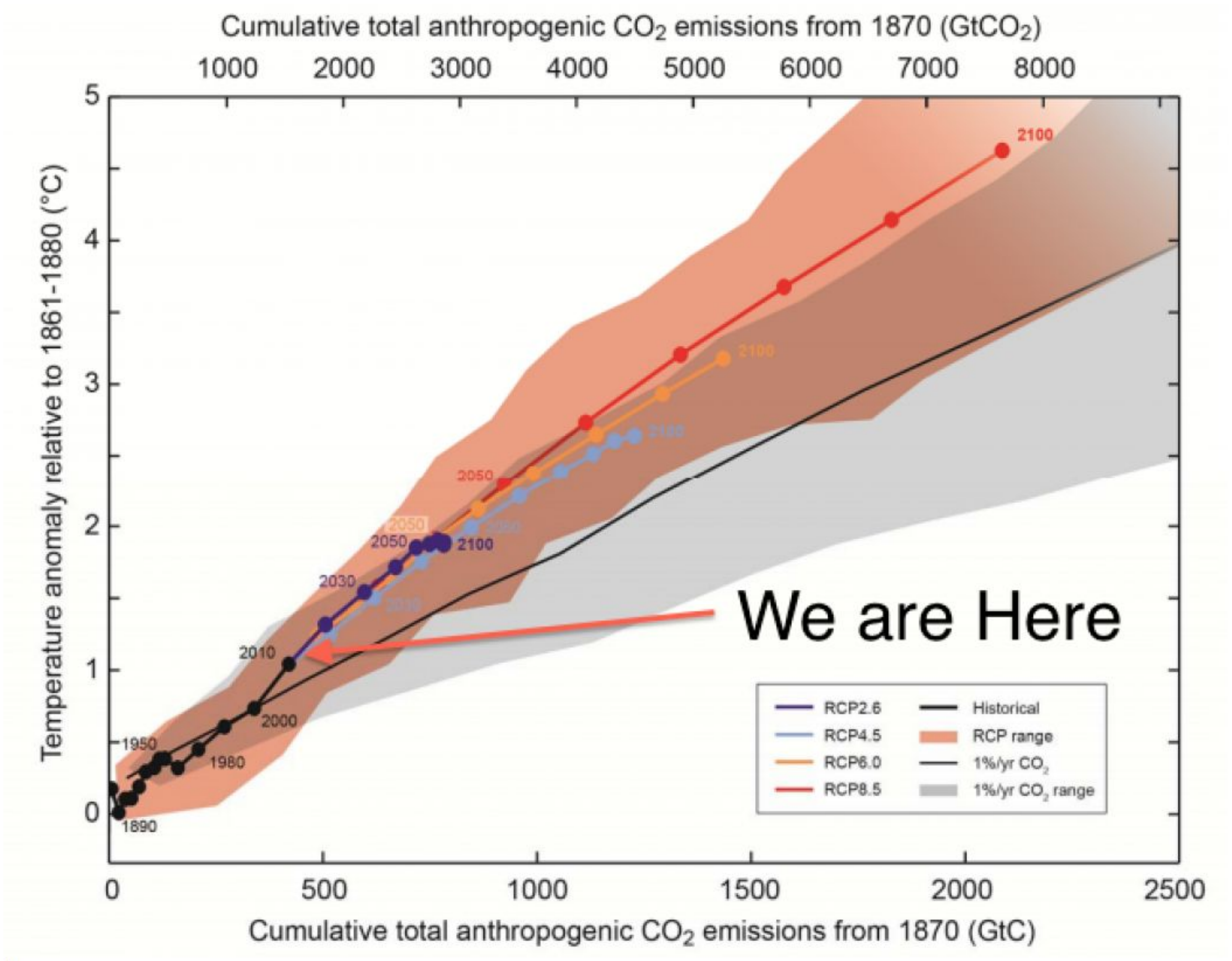
Precipitation: 20%
Humidity: 34%
Wind: 18 km/h

Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri
41° 28°	41° 28°	40° 27°	39° 26°	39° 27°	38° 27°	37° 26°	36° 26°

More on weather.com

Feedback

Global warming



Prediction of increase/decrease in temperature for the future years.

Stock market prediction

Predicting which stocks might go up or down in future based on the past data

Little on the Input Data.

How is time-series/sequence data different compared to other static data?

For example: Weather prediction,

$x_1,x_2,x_3,x_4,x_5,x_6,....x_{n-1}$ is the given data and the task is to predict x_n

Each x is a set of attributes (measurements such as humidity, temperature, wind, could-cover etc.)

How can solve this task?

Little on the Input Data.

What other types of sequence do we know of?

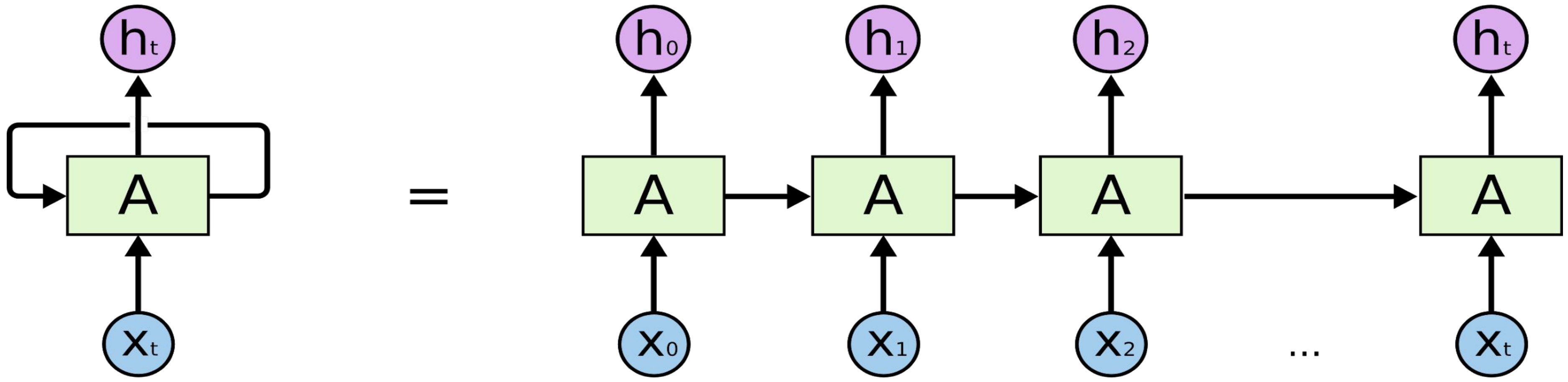
Simple example is “text”, any others you can think of?

The dataset now is set of sequences,

Sequence i: $x^i_{1,} x^i_2 x^i_3 x^i_4 \ldots x^i_n$

Given this sequence as input we predict some target such as **sentiment**.

Recurrent Neural Network



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Intuition

Are we humans constrained by length or dimensions of input?

No

For example consider text understanding:

-
- We can read and understand arbitrary length sequences.
-

Joe waited for the train.

Mary and Samantha took the bus

I looked for Mary and Samantha at the bus station.

What we need is a internal memory state to remember past input.

Intuition

I

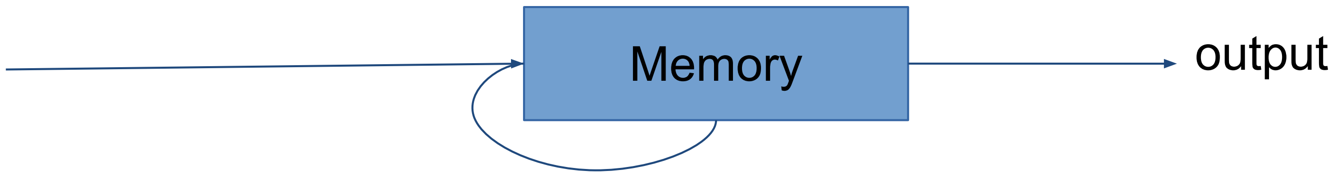
looked

for

Mary

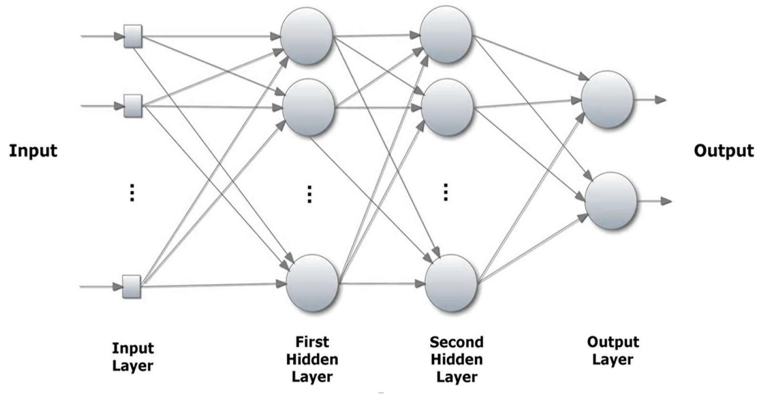
and

Samantha

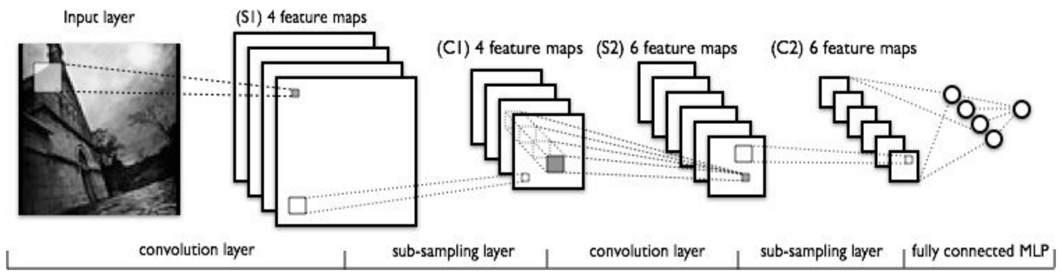


Intuition

Do feed forward architectures, we saw so far, have this property to read one element of a sequence at a time and predict based on current element together with past memory?



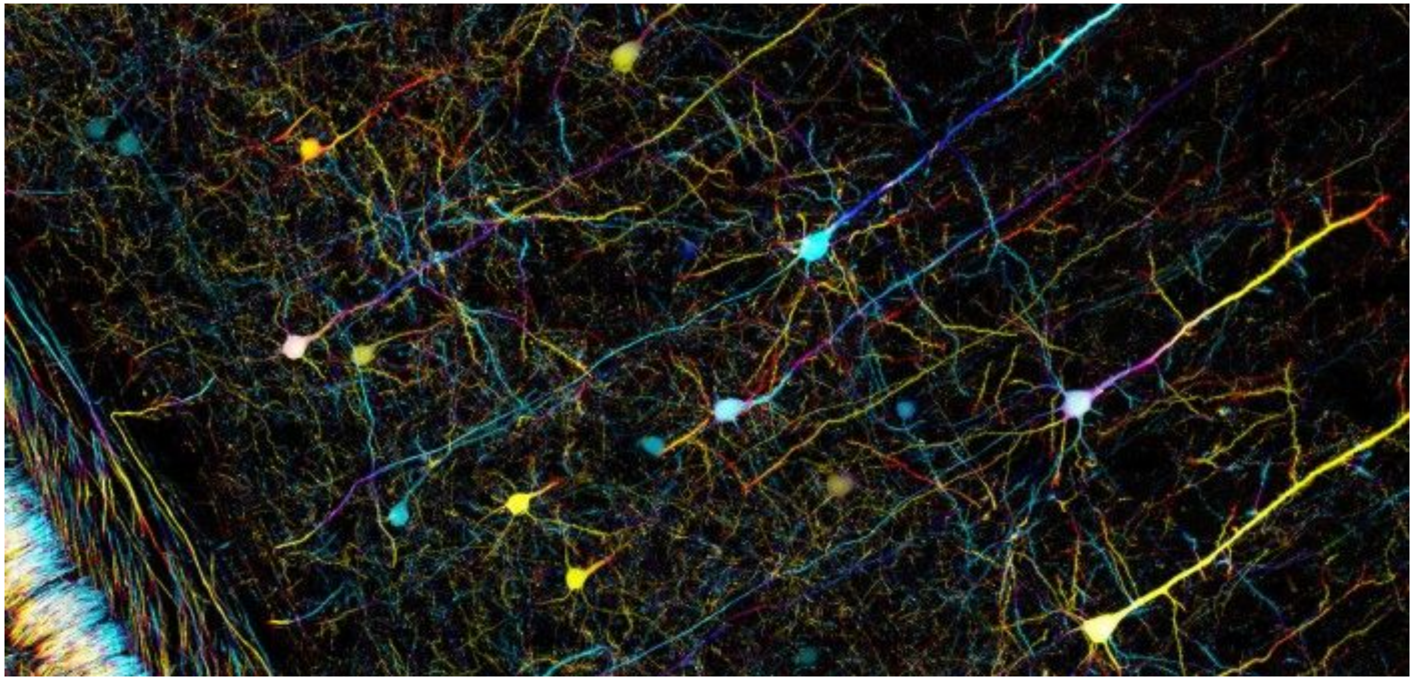
MLP



CNN

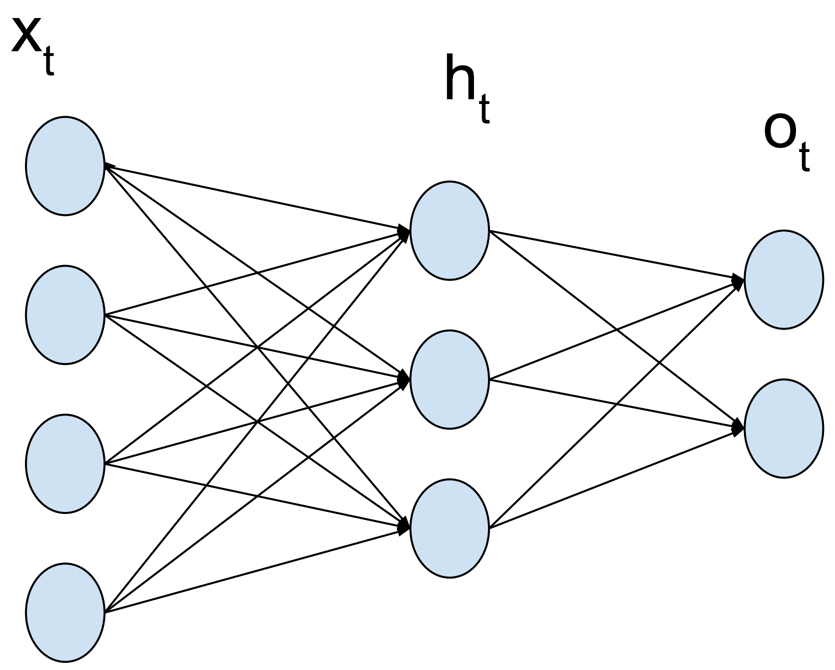
Let's draw inspiration from biology

Are all the connections in brain feed-forward?



Are there feedback connections?

Let's build with what we know!



Where 't' is the index in a given input sequence. One input gives one output no concept of past.

Lets build from what we know!

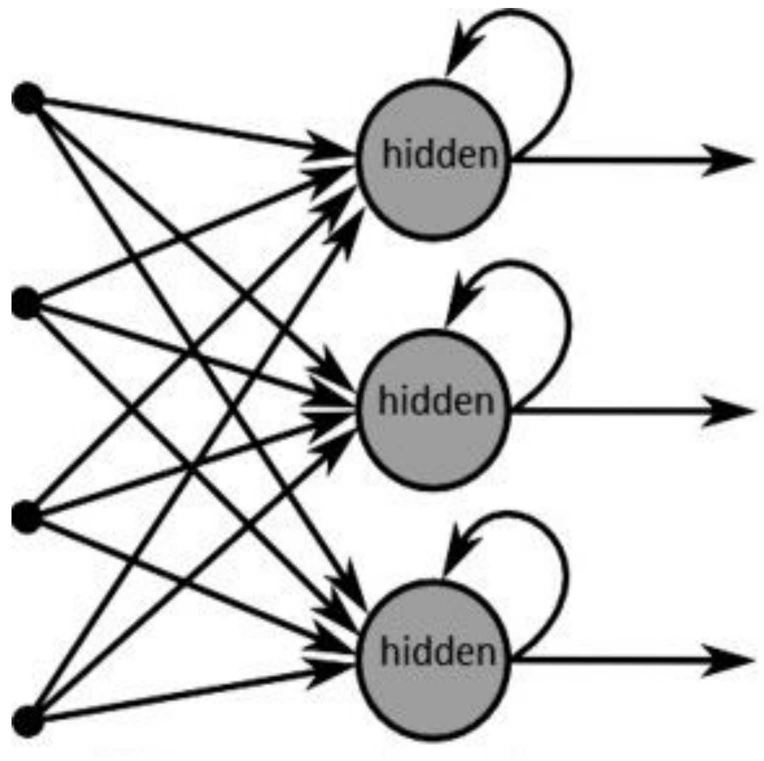
Lets first write the equation for output o_t also considering also the past h_{t-1}

$s_t = \tanh(Ws_{t-1} + Ux_t + b)$
$o_t = \mathit{sigma}(Vs_t + c)$

How do we implement these equations in terms of a network architecture.

Lets build from what we know!

x_t



$$s_t = \tanh(Ws_{t-1} + Ux_t + b)$$

A hidden layer is fully connected to itself.

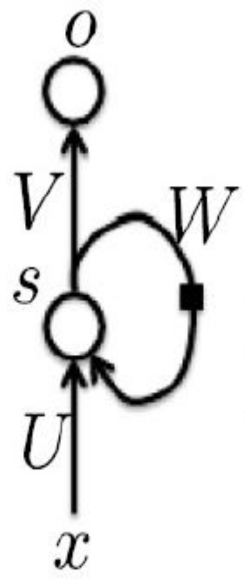
What does it even mean?

RNN

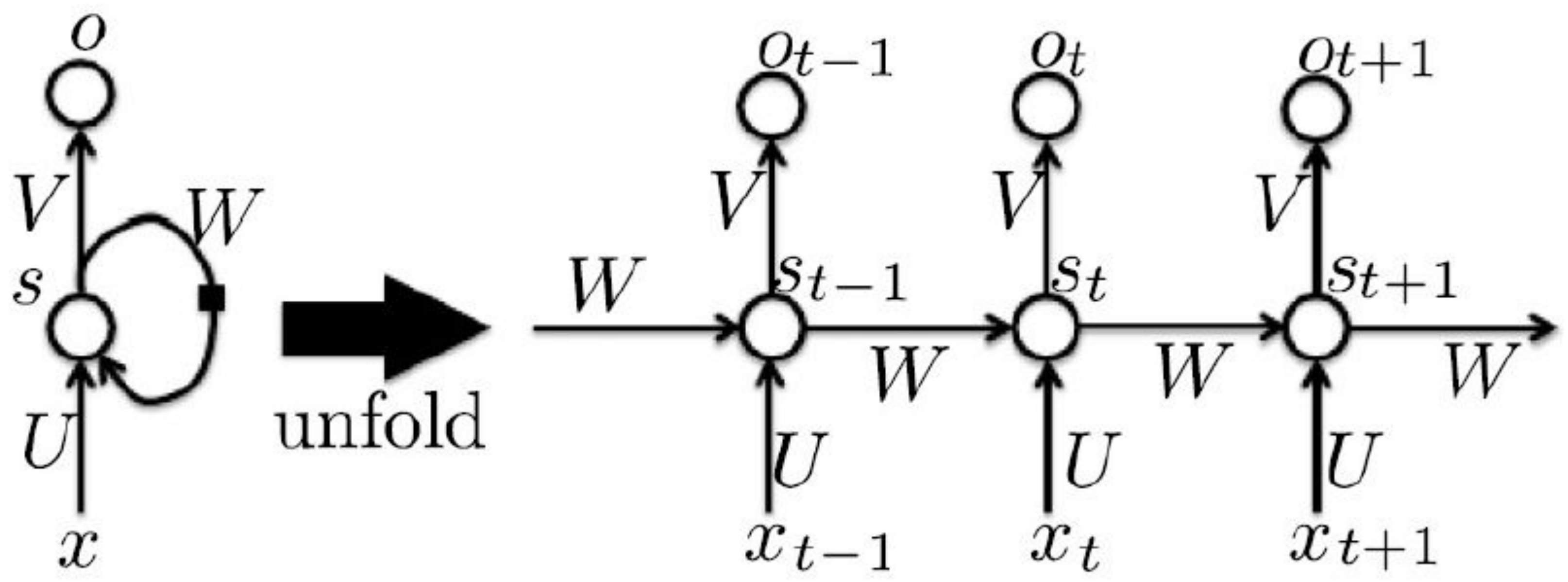
A Recurrent neural network is created by applying the same set of weights over a differentiable graph-like structure

Introduced first in 1986 (Rumelhart et al 1986)

Mostly used to handle sequential data of arbitrary input lengths.

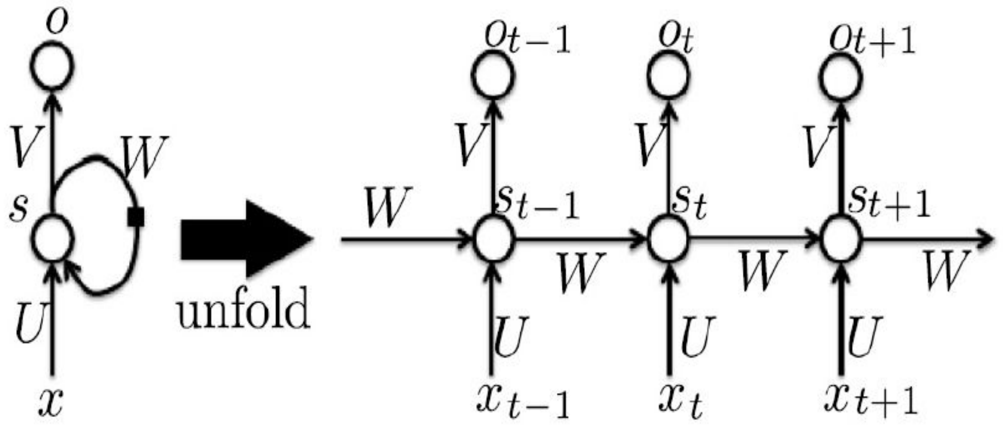


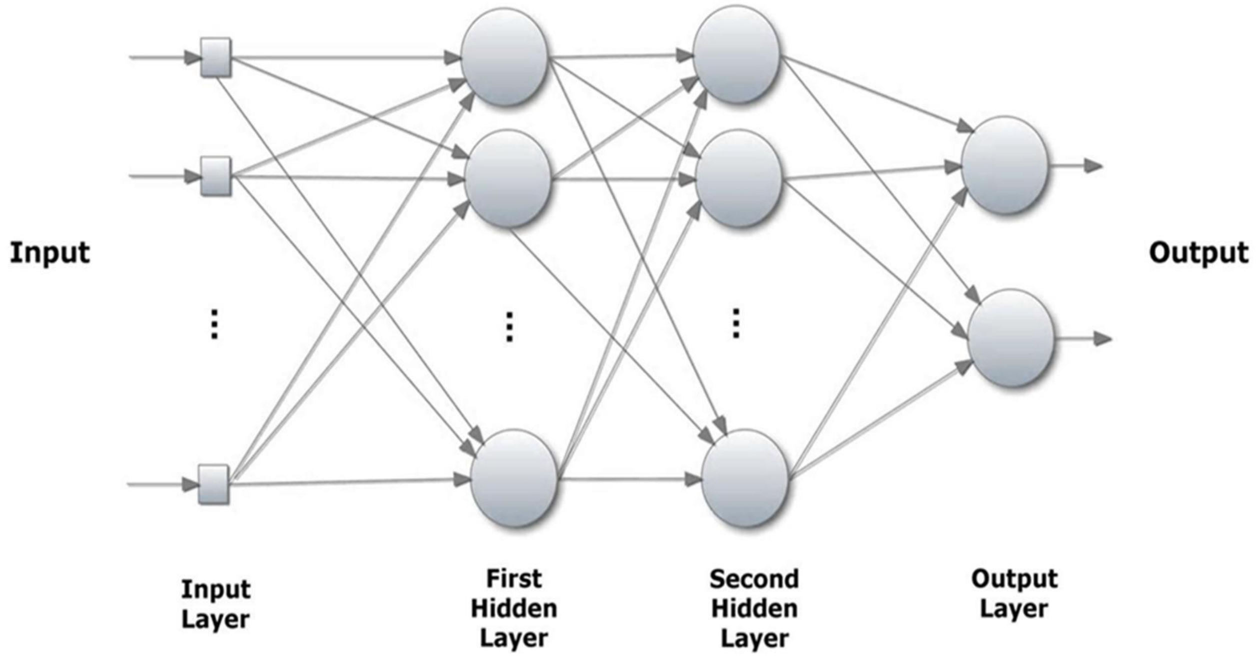
RNN



s_t = tanh(Ws_{t-1} + Ux_t + b)
o_t = *sigma*(Vs_t + c)

Comparison with feed forward networks

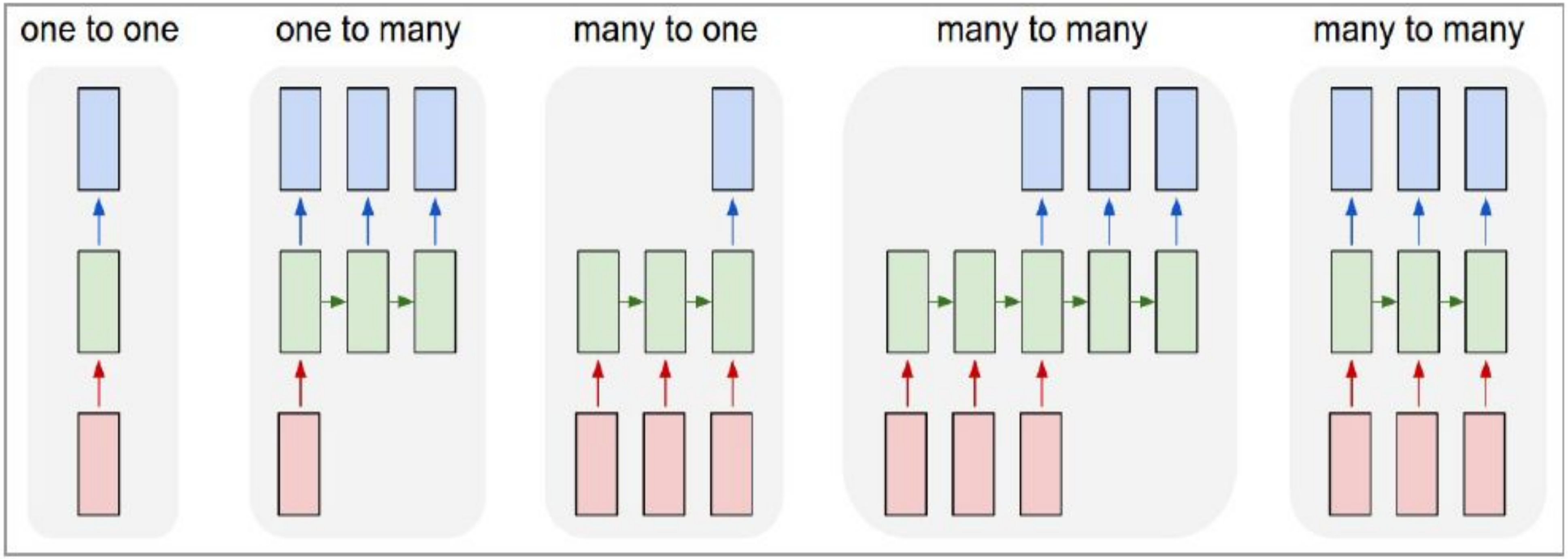




Parameter sharing across sequence

Multiple inputs at different time instance

Self or recurrent connections


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

Input are in red, output vectors are in blue and green vectors hold the RNN's state. From left to right:

(1) Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification).

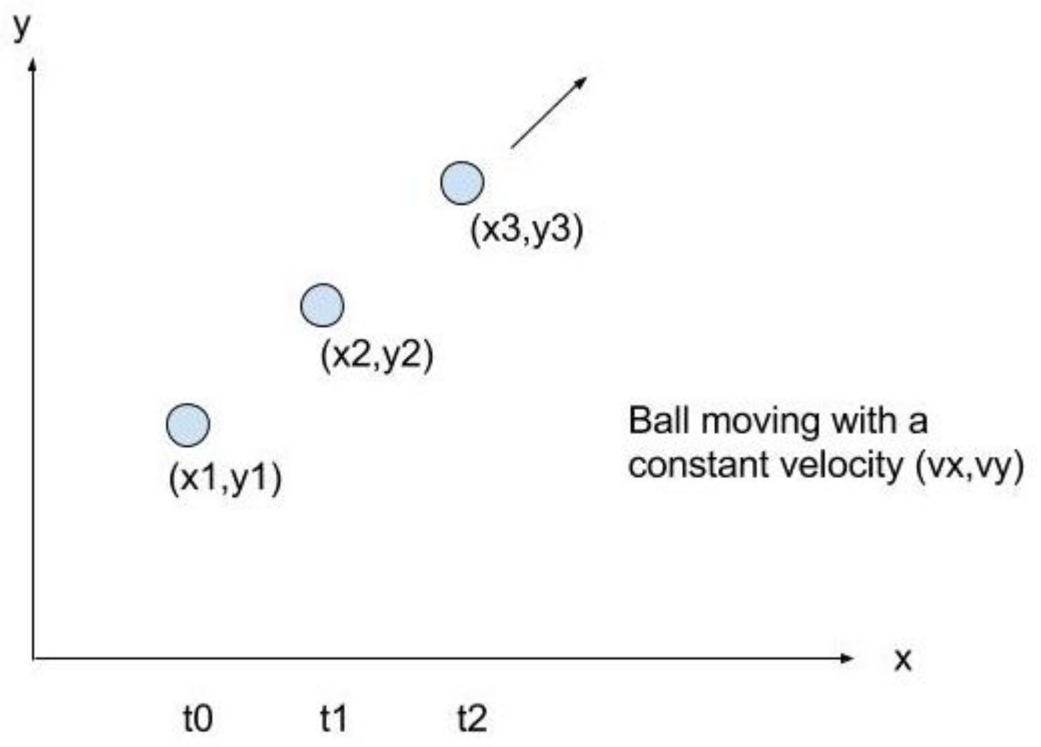
(2) Sequence output (e.g. image captioning takes an image and outputs a sentence of words).

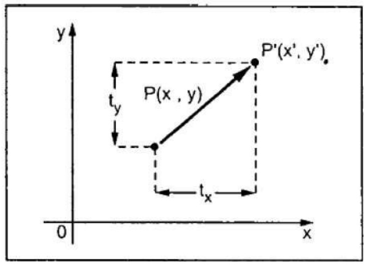
(3) Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment).

(4) Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French).

(5) Synced sequence input and output (e.g. video classification where we wish to label each frame of the video).

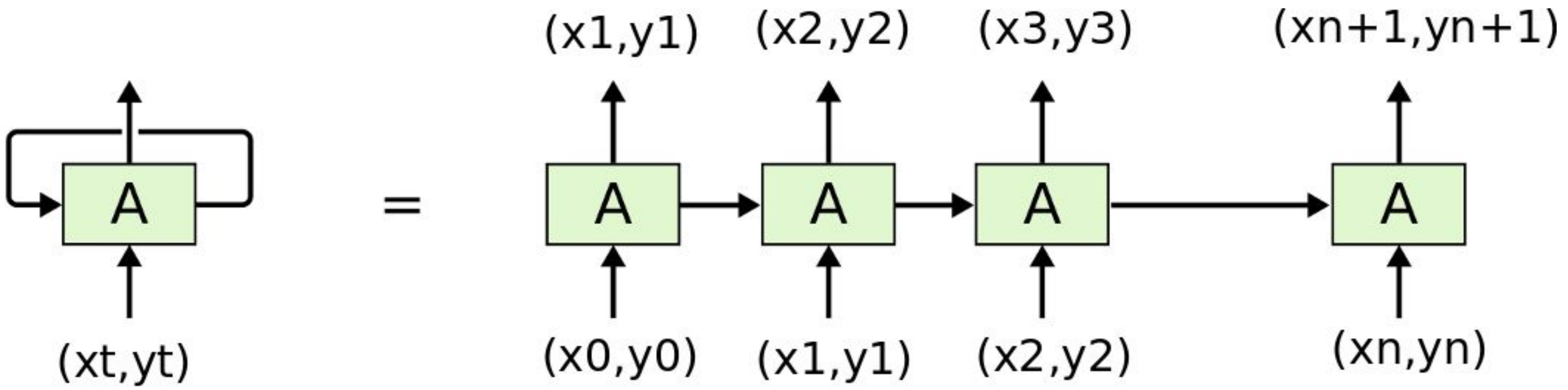
Simple physics use case



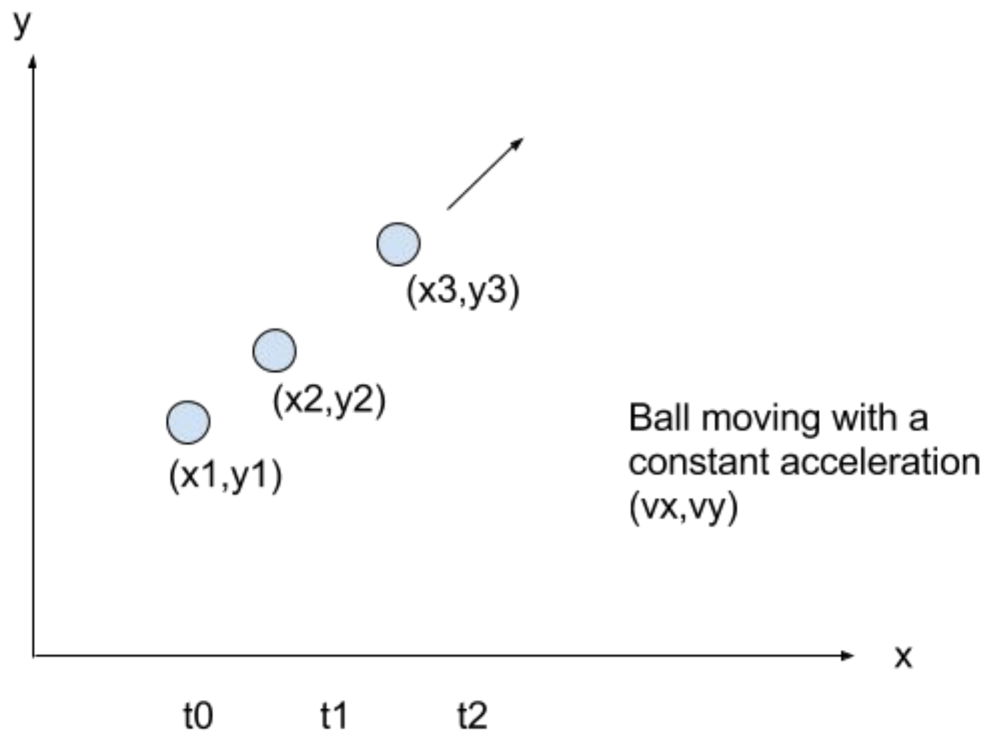


The input sequence in this case is a series x,y positions of the ball and the target would be to predict the future positions of the ball.

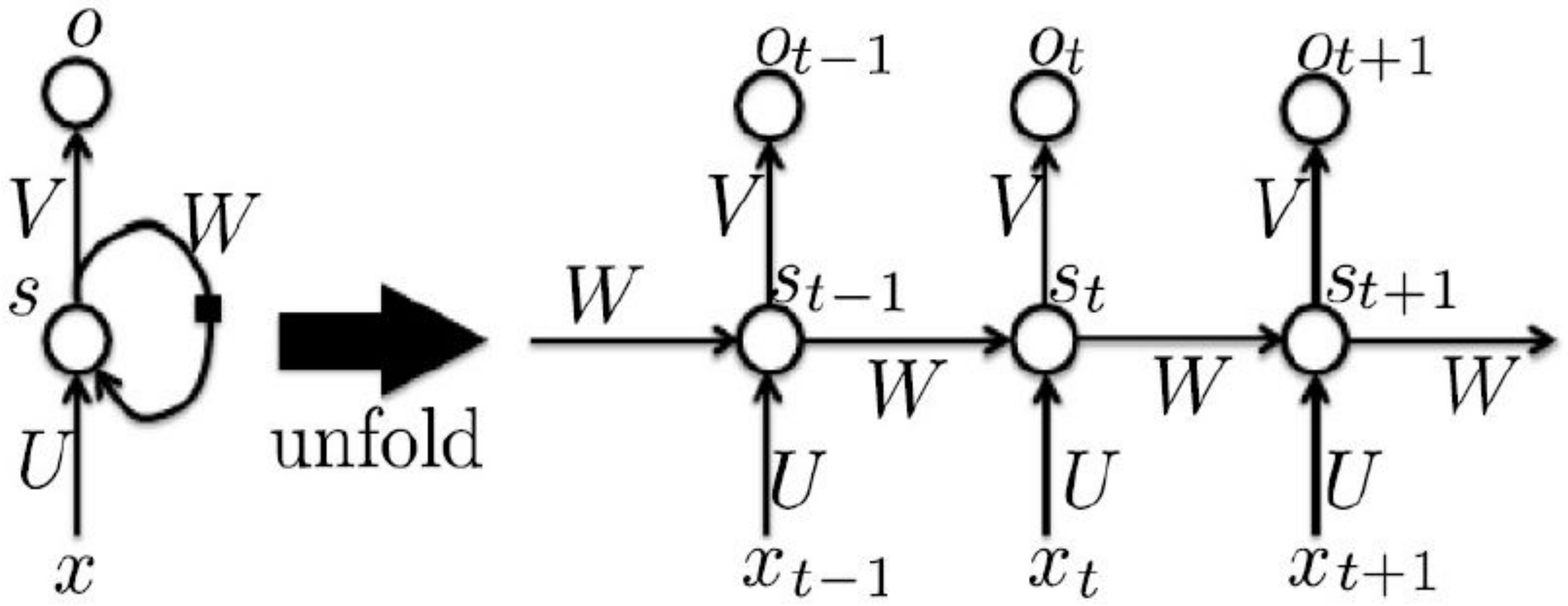
Simple physics use case



Slightly complex physics use case



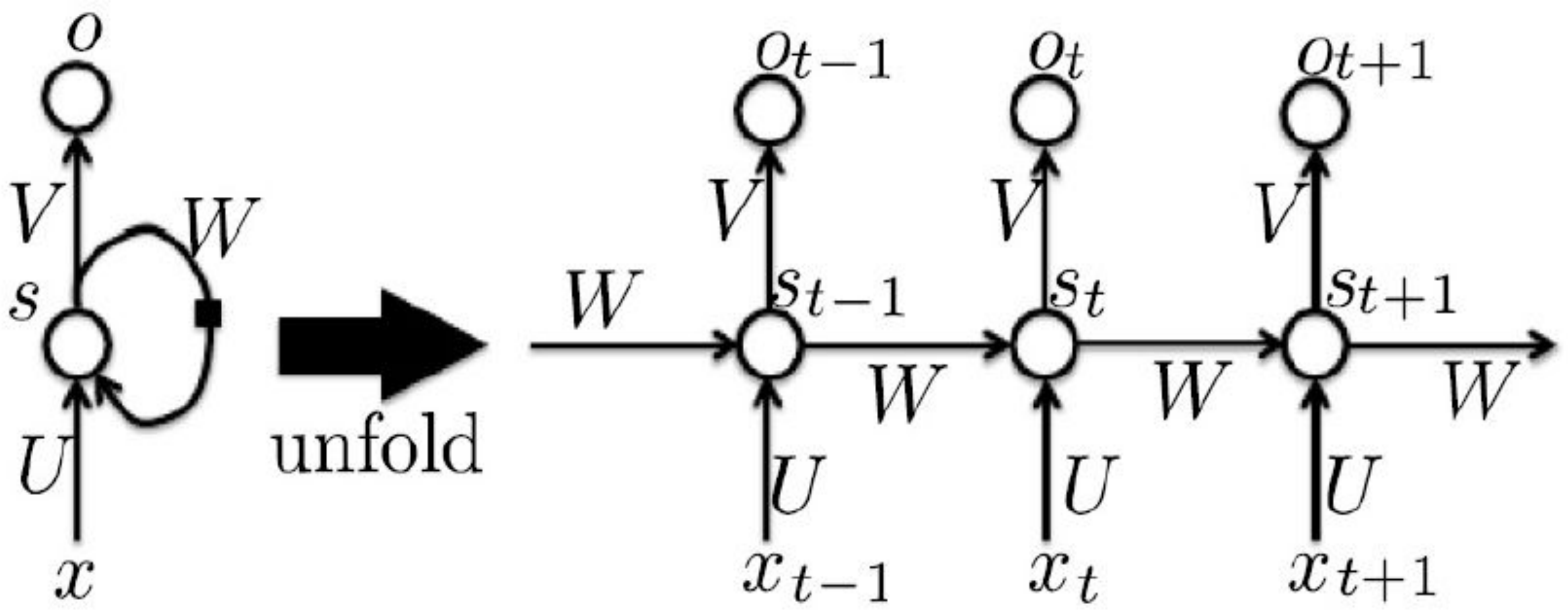
Backpropagation Through Time (BPTT)



BPTT begins by unfolding a recurrent neural network through time .

Training then proceeds in a manner similar to training a feed-forward neural network with backpropagation.

Back Propagation Through Time (BPTT)



After each pattern is presented, weight updates are computed for time instance. All weight updates are averaged together so that they all have the same weights.

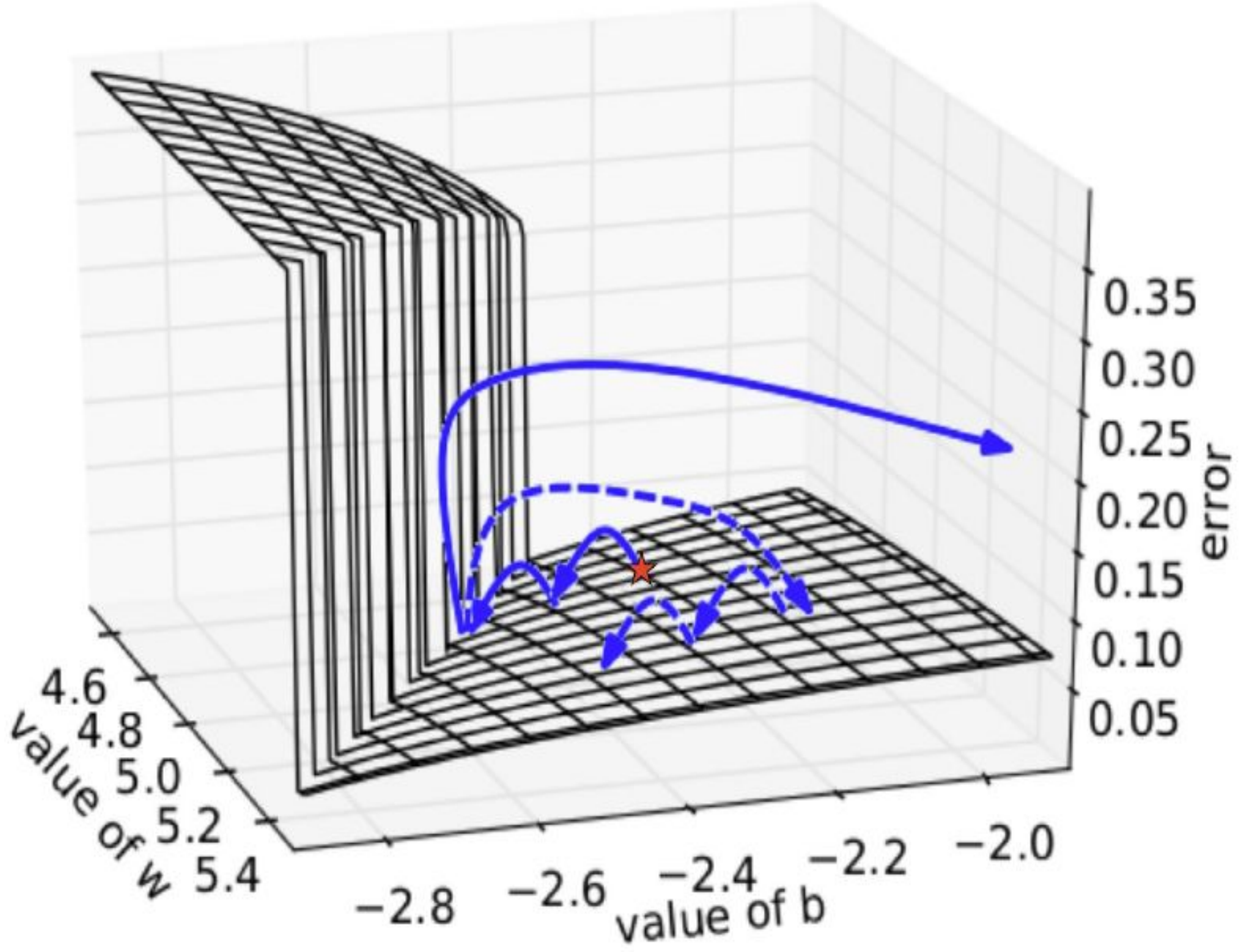
Issue with plain vanilla RNNs

Exploding or vanishing products of Jacobian

In recurrent nets (also in very deep nets), the final output is the composition of a large number of non-linear transformations.

Even if each of these non-linear transformations is smooth. Their composition might not be.

The derivatives through the whole composition will tend to be either very small or very large.



0.35
0.30
0.25
0.20
0.15
0.10
0.05
error
4.6
4.8
5.0
5.2
5.4
value of w
−2.8
−2.6
−2.4
−2.2
−2.0
value of b

Exploding gradients

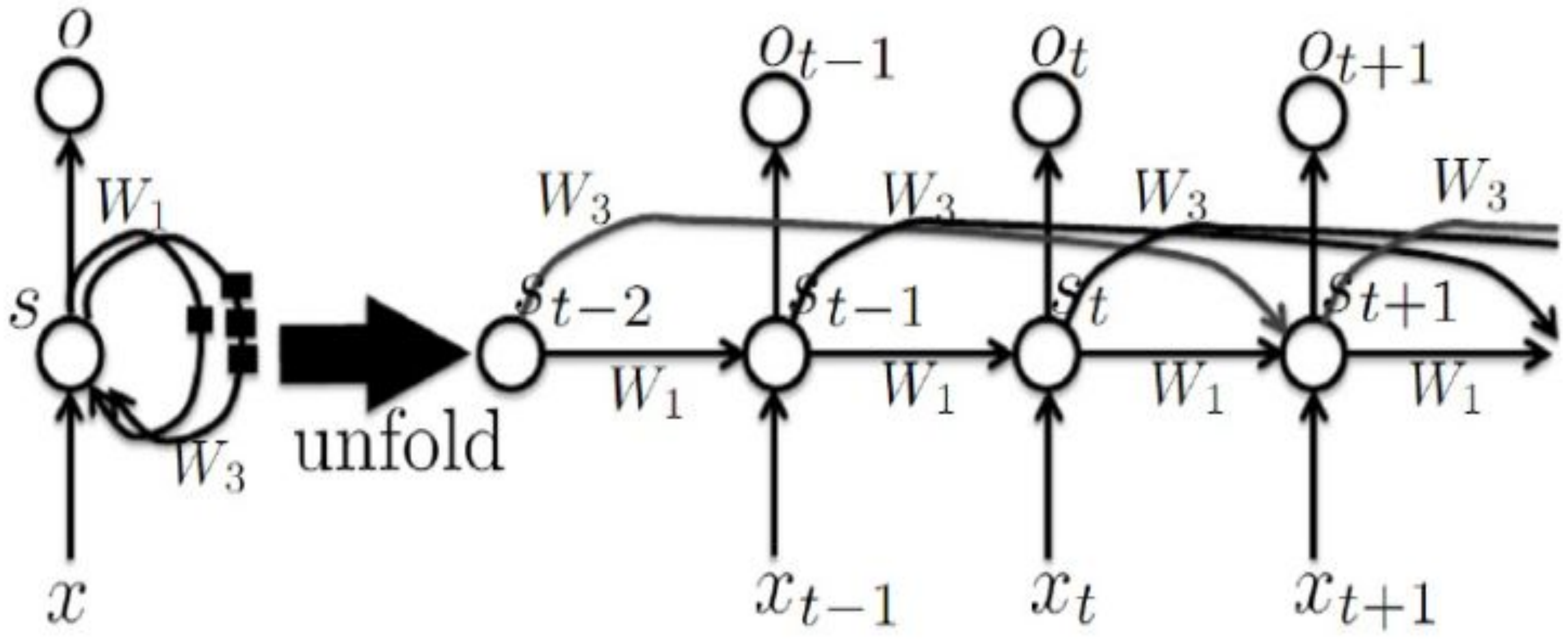
- Simple solution can be to clip the gradient (Mikolov, 2012; Pascanu et al., 2013)
- Clip the parameter gradient from a mini batch element-wise (Mikolov, 2012) just before the parameter update.
- Clip the norm g of the gradient (Pascanu et al., 2013) just before the parameter update.
- You can imagine perhaps controlling this issue by rescaling gradients to never exceed a maximal magnitude (see the dotted path after hitting the cliff), but this approach still doesn't perform spectacularly well, especially in more complex RNNs.

- Using BPTT results in a vanishing gradient problem.
- Error is propagated back in time where each time step is exactly equivalent to propagating through an additional layer of a feed forward network.
- The importance of network state at times which lie far back in the past.
- Typically, gradient based networks cannot reliably use information which lies more than about 10 time steps in the past.
- If you now imagine an attempt to use a recurrent neural network in a real life situation

Solutions for vanishing gradients

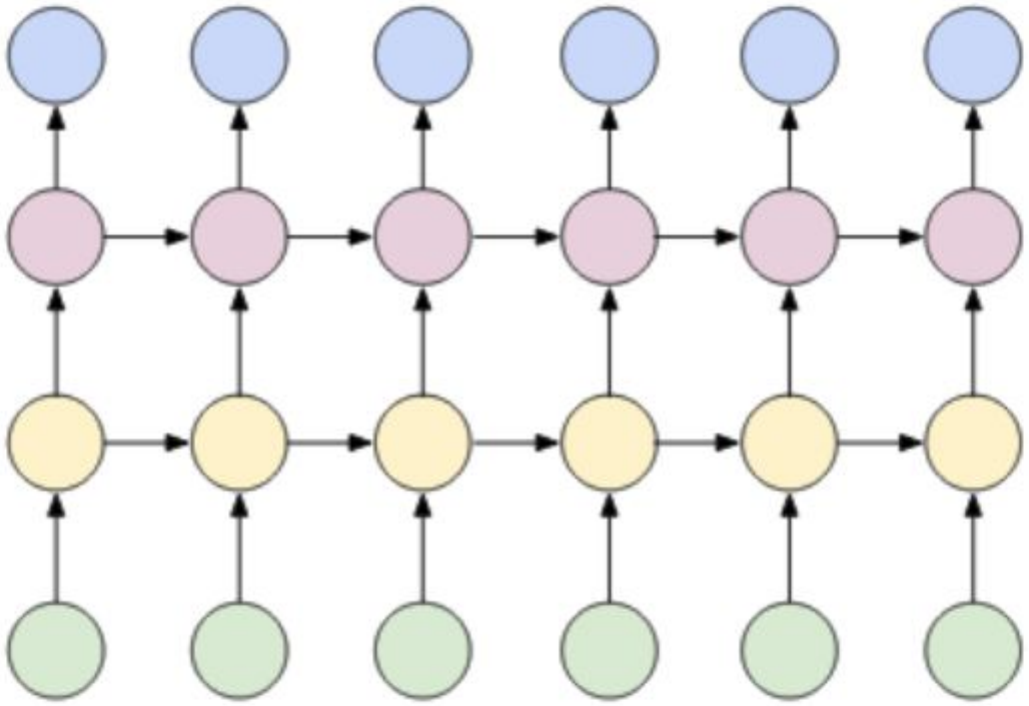
Long delay RNNs

Use recurrent connections with long delays.



Applications

Deep RNN



Learning formal grammars

Given a set of strings S, each composed of a series of symbols, identify the strings which belong to a language L.

A simple example: L = {a,b} is the language composed of strings of any number of a's, followed by the same number of b's.

Strings belonging to the language include aaabbb, ab, aaaaaabbbbbb.

Strings not belonging to the language include aabbb, abb, etc.

Strings which belong to a language L are said to be grammatical and are ungrammatical otherwise.

Speech recognition

In some of the best speech recognition systems built so far are based on RNNs

Speech is first presented as a series of spectral slices to a recurrent network.

Each output of the network represents the probability of a specific word given both present and recent input.

The probabilities are then interpreted by a Hidden Markov Model which tries to recognize the whole utterance.

HMM can be replaced by RNN too.

Music composition

- A recurrent network can be trained by presenting it with the notes of a musical score. It's task is to predict the next note. Obviously this is impossible to do perfectly, but the network learns that some notes are more likely to occur in one context than another.

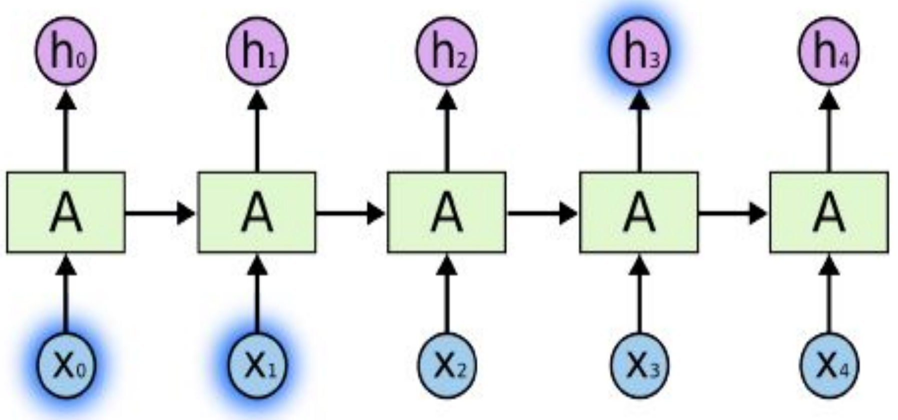
- Training, for example, on a lot of music by J. S. Bach, we can then seed the network with a musical phrase, let it predict the next note, feed this back in as input, and repeat, generating new music. Music generated in this fashion typically sounds fairly convincing at a very local scale, i.e. within a short phrase. At a larger scale, however, the compositions wander randomly from key to key, and no global coherence arises. This is an interesting area for further work.
-

https://www.youtube.com/watch?v=0VTI1BBLydE

Gated RNNs

How much of past?

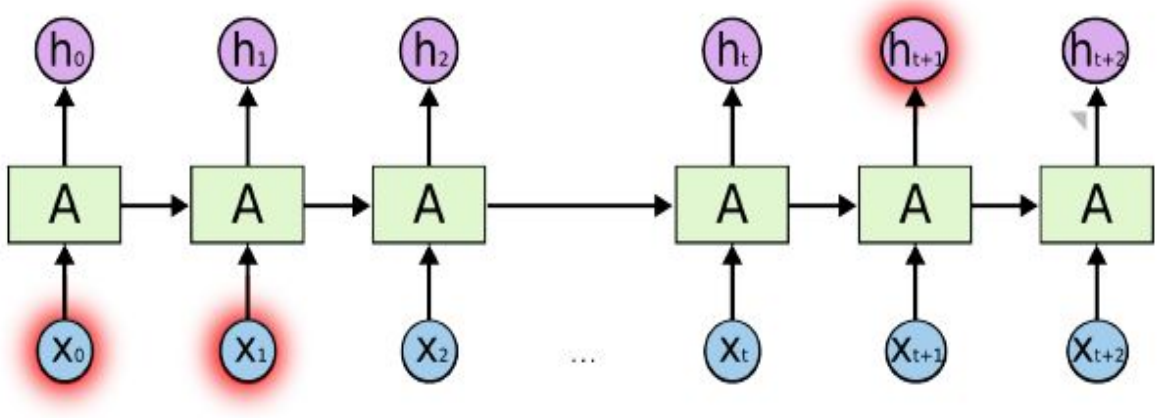
Sometimes, we only need to look at recent information to perform the present task. For example, consider a language model trying to predict the next word based on the previous ones. If we are trying to predict the last word in "the clouds are in the *sky*," we don't need any further context – it's pretty obvious the next word is going to be sky. In such cases, where the gap between the relevant information and the place that it's needed is small, RNNs can learn to use the past information.



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

But there are also cases where we need more context. Consider trying to predict the last word in the text 'I grew up in France... I speak fluent *French*." Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back. It's entirely possible for the gap between the relevant information and the point where it is needed to become very large.

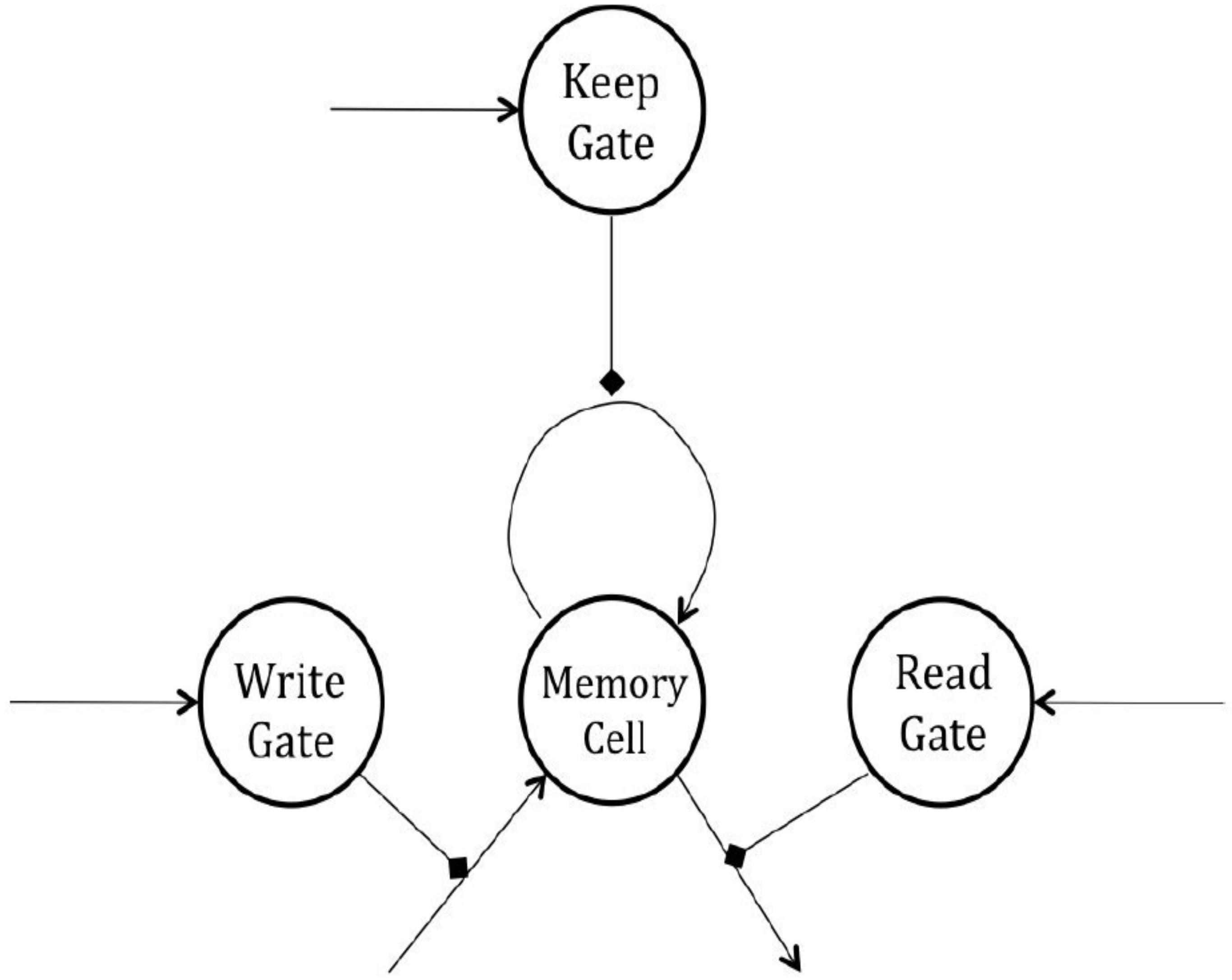
Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.



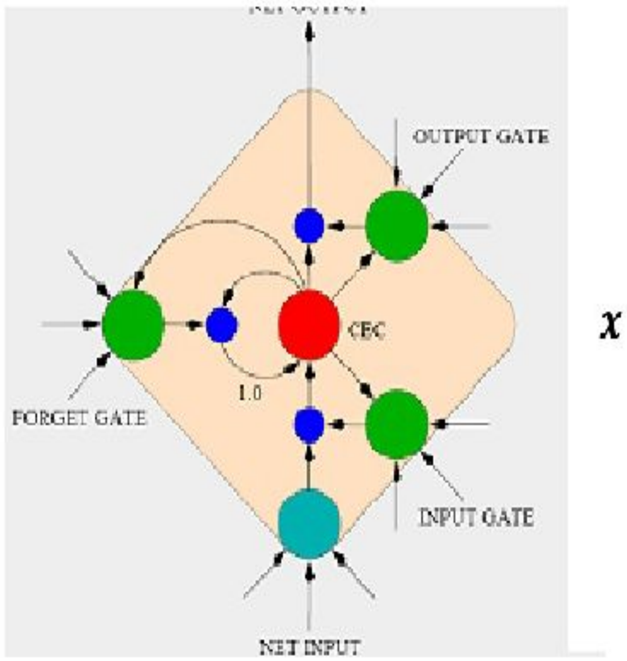
In theory, RNNs are absolutely capable of handling such 'long-term dependencies." A human could carefully pick parameters for them to solve toy problems of this form. Sadly, in practice, RNNs don't seem to be able to learn them. The problem was explored in depth by Hochreiter (1991) [German]

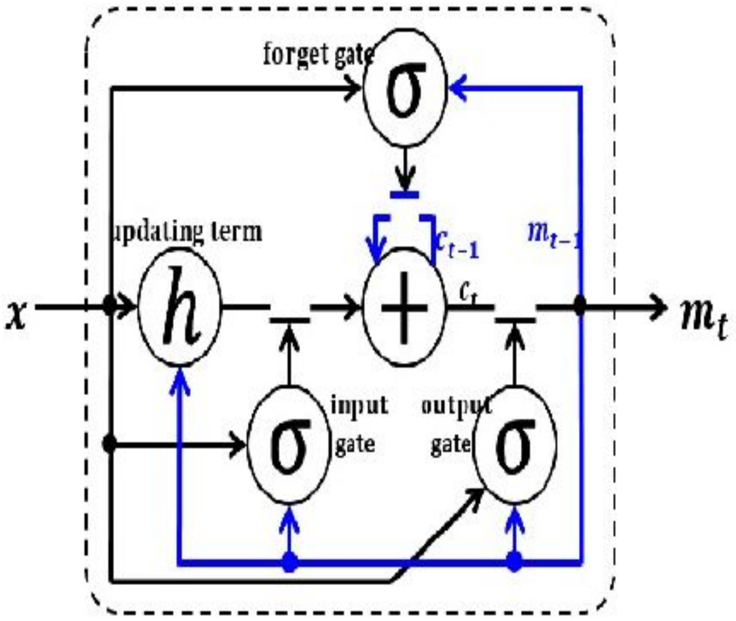
How to solve

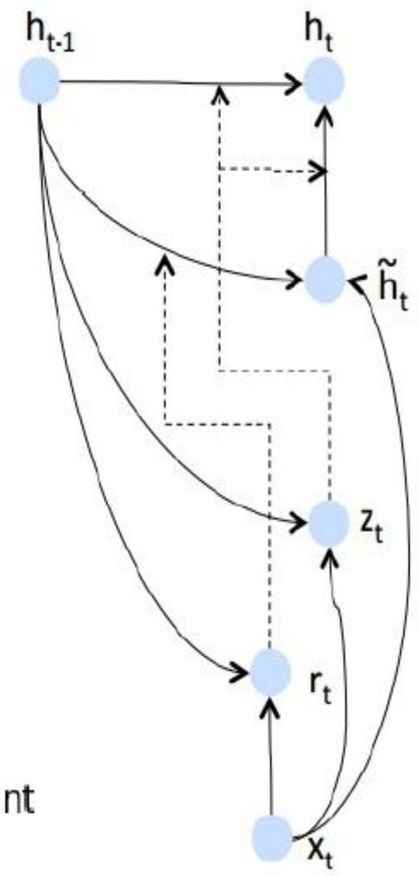
- A human carefully analyzes the passage and picks the right features to put in the memory.
- A natural architecture that remembers the relevant past information.
- In an artificial system we need a mechanism which can facilitate update of memory only when required.

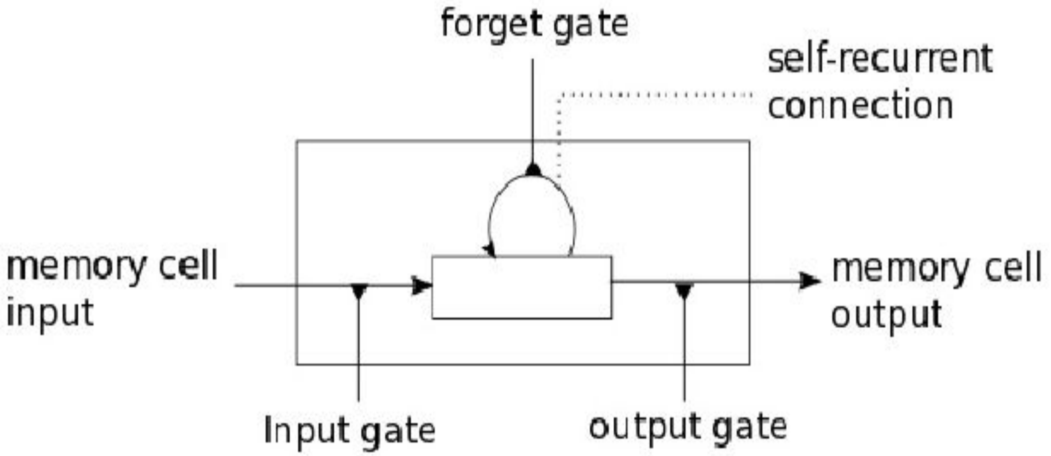

Keep Gate
Write Gate
Memory Cell
Read Gate

LSTMs and GRUs



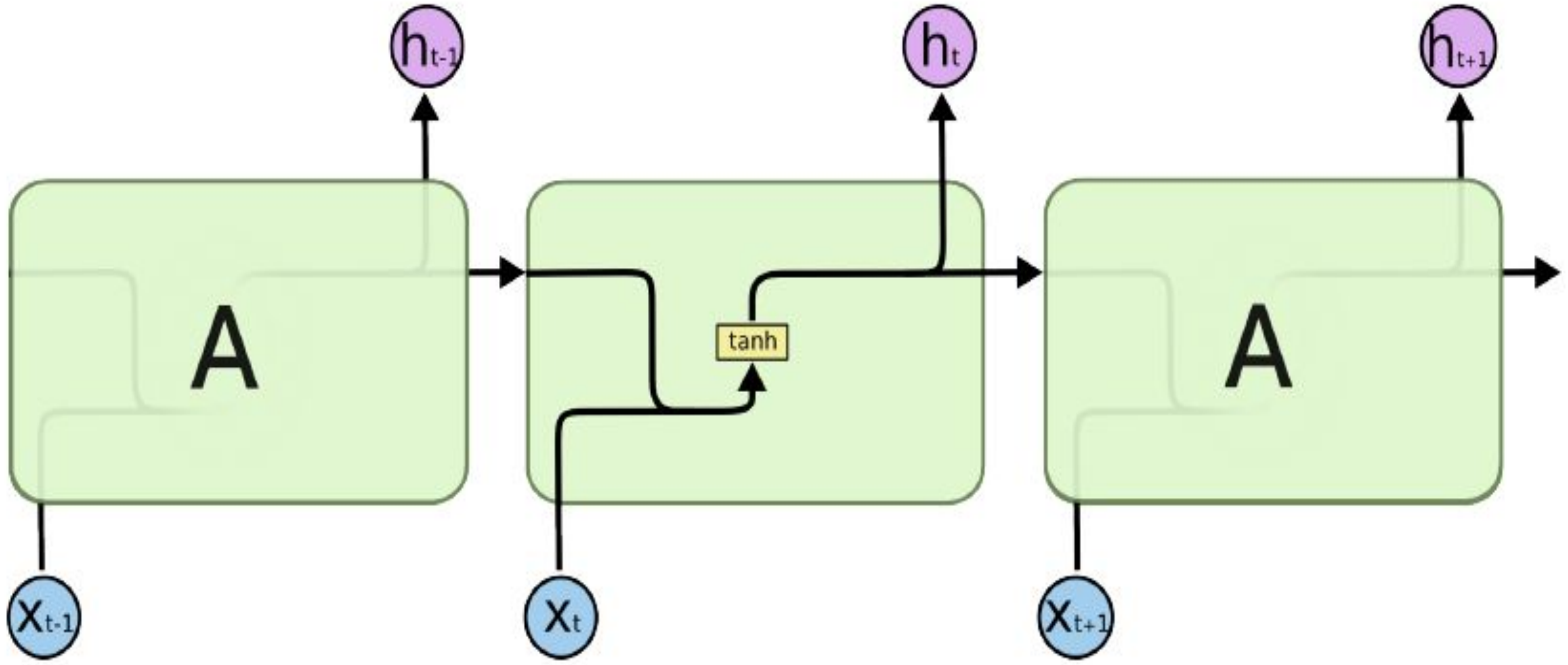






Long Short Term Memory RNNs (LSTM)

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

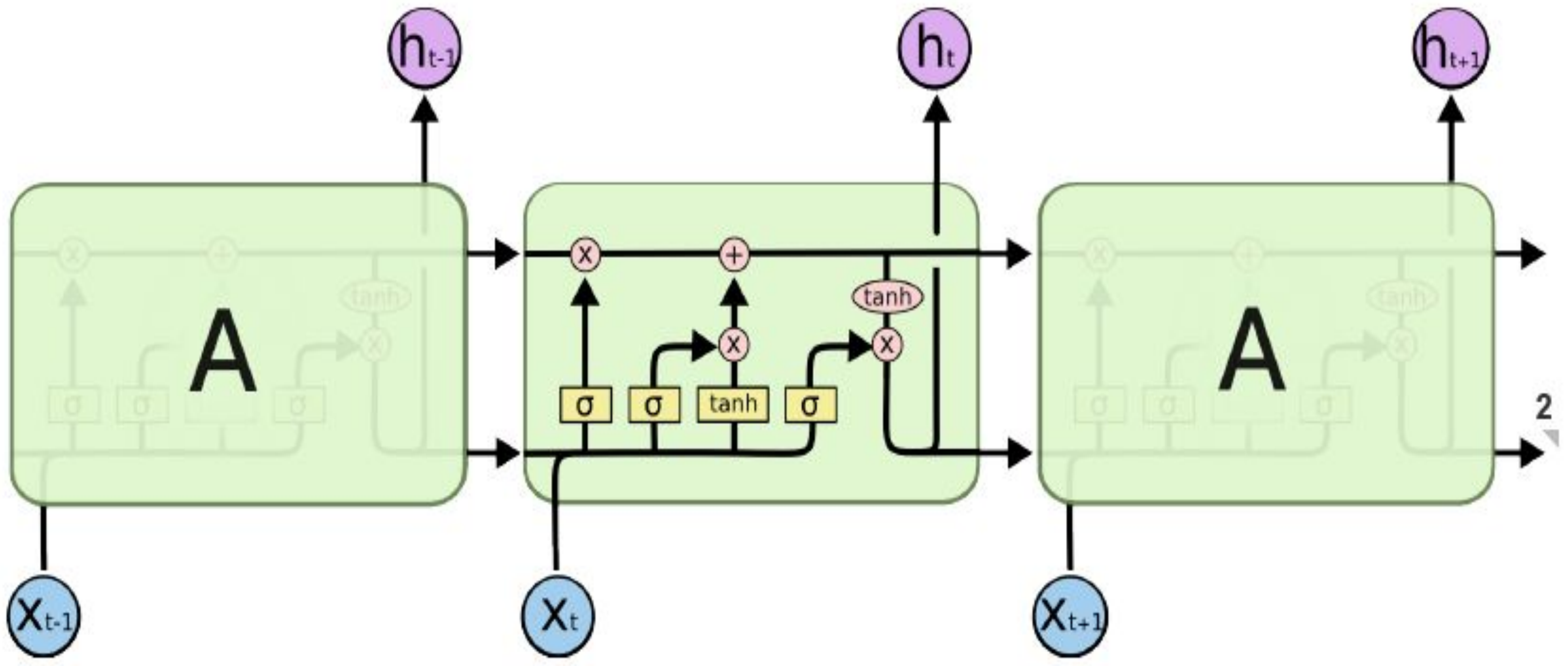


4

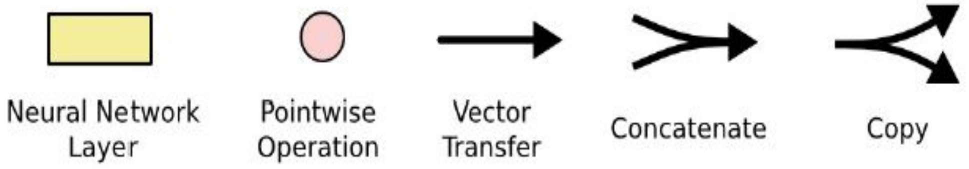
The repeating module in a standard RNN contains a single layer.

LSTMs

- LSTMs help preserve the error that can be backpropagated through time and layers.
- They allow recurrent nets to continue to learn over many time steps (over 1000)
- This opens a channel to link causes and effects remotely.
- LSTMs contain information outside the normal flow of the recurrent network in a gated cell.
- Information can be stored in, written to, or read from a cell, much like data in a computer's memory.
- The cell makes decisions about what to store, and when to allow reads, writes and erases, via gates that open and close.

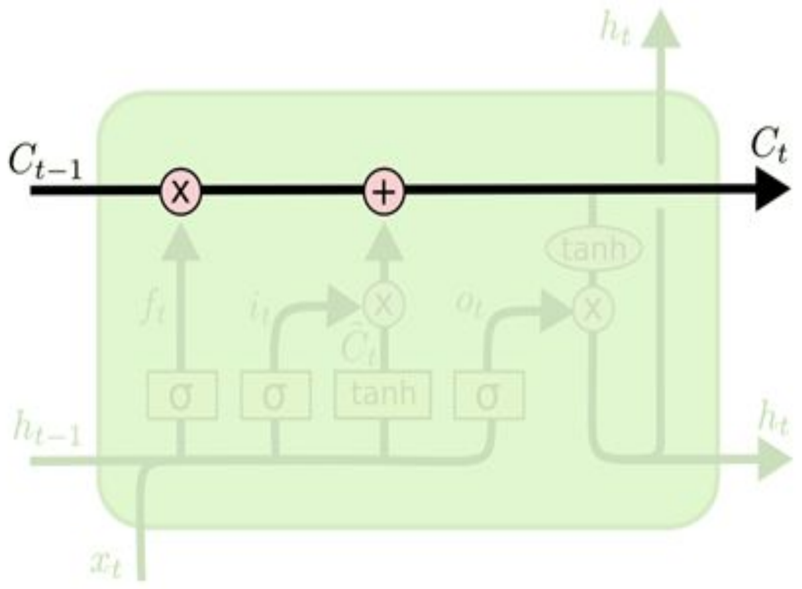


The repeating module in an LSTM contains four interacting layers.



Each line carries an entire vector, from the output of one node to the inputs of others.
The pink circles are point-wise operations, like vector addition,
The yellow boxes are learned neural network layers.
Lines merging denote concatenation,
Line forking denote its content being copied and the copies going to different locations.

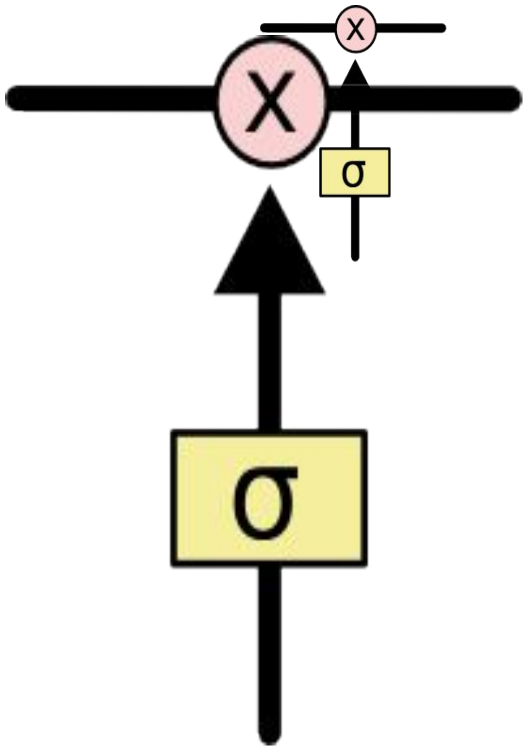
Cell state



The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.

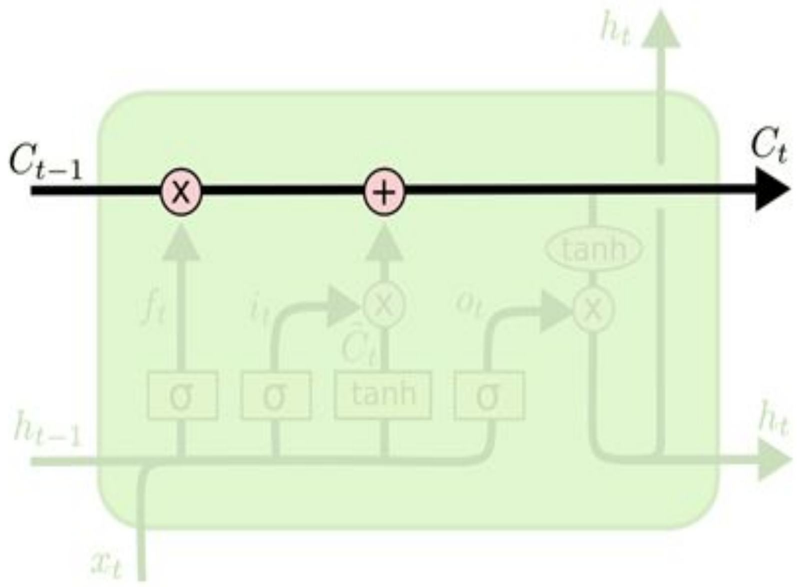
The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.

The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates. Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a point-wise multiplication operation.



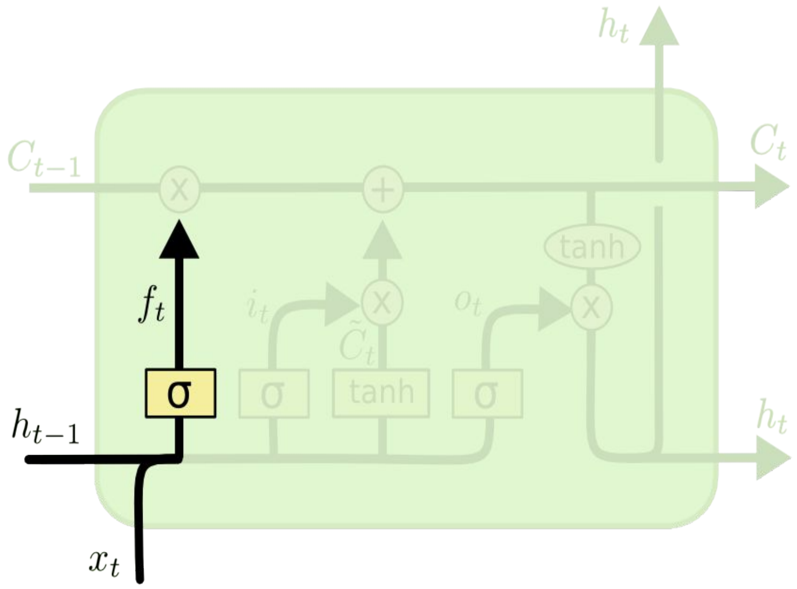
The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means “let nothing through,” while a value of one means “let everything through!”

An LSTM has three of these gates, to protect and control the cell state.



The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer." It looks at ht−1 and xt, and outputs a number between 0 and 1 for each number in the cell state Ct−1.

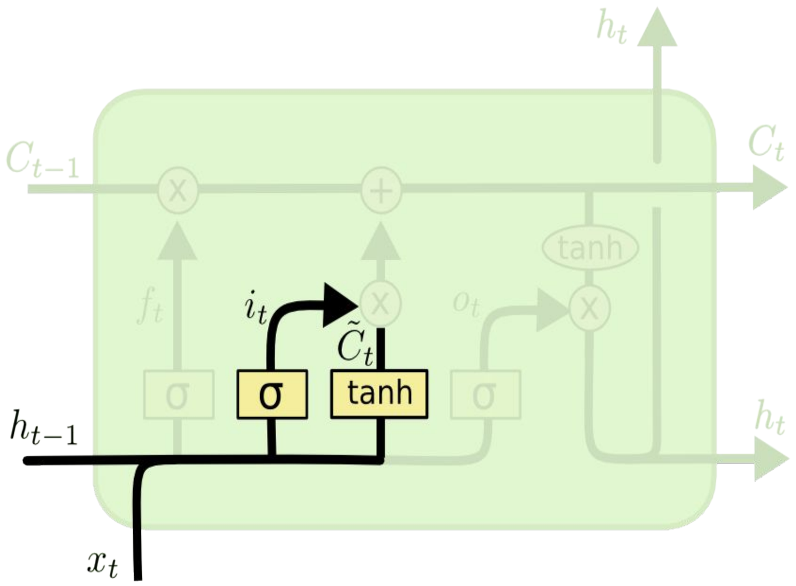
1 represents "completely keep this" while a 0 represents "completely get rid of this."



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

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.

- The next step is to decide what new information we're going to store in the cell state. This 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, Ct, that could be added to the state. In the next step, we'll combine these two to create an update to the state.
-
- In the example of our language model, we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting.



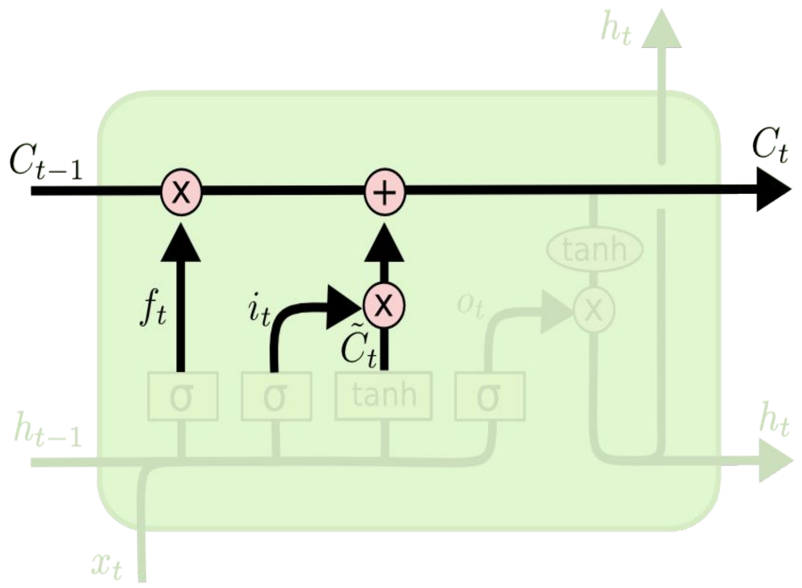
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

- It’s now time to update the old cell state, C_{t-1}, into the new cell state C_t. The previous steps already decided what to do, we just need to actually do it.

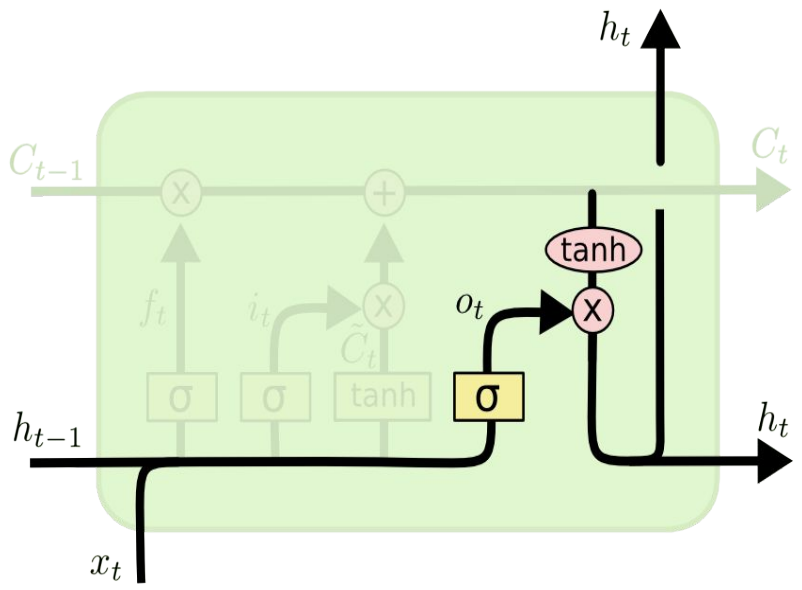
- We multiply the old state by ft, forgetting the things we decided to forget earlier. Then we add $i_t * \tilde{C}_t$. This is the new candidate values, scaled by how much we decided to update each state value.

- In the case of the language model, this is where we’d actually drop the information about the old subject’s gender and add the new information, as we decided in the previous steps.



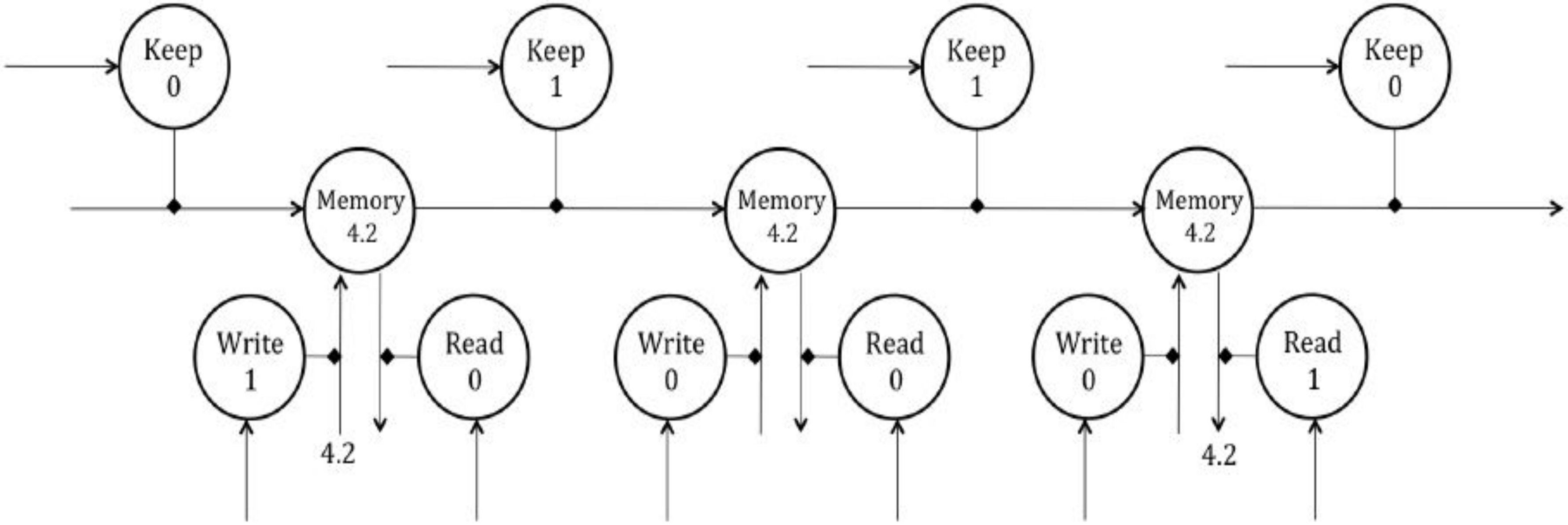
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- Now we need to decide what we’re going to output.
- The output will be based on our cell state, but will be a filtered version.
- First, we run a sigmoid layer which decides what parts of the cell state we’re going to output.
- Then, we put the cell state through tanh (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.



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

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


Keep 0
Keep 1
Keep 1
Keep 0
Memory 4.2
Memory 4.2
Memory 4.2
Write 1
Read 0
Write 0
Read 0
Write 0
Read 1
4.2
4.2

- At first, the keep gate is set to 0 and the write gate is set to 1, which places 4.2 into the memory cell.

- This value is retained in the memory cell by a subsequent keep value of 1 and protected from read/write by values of 0.

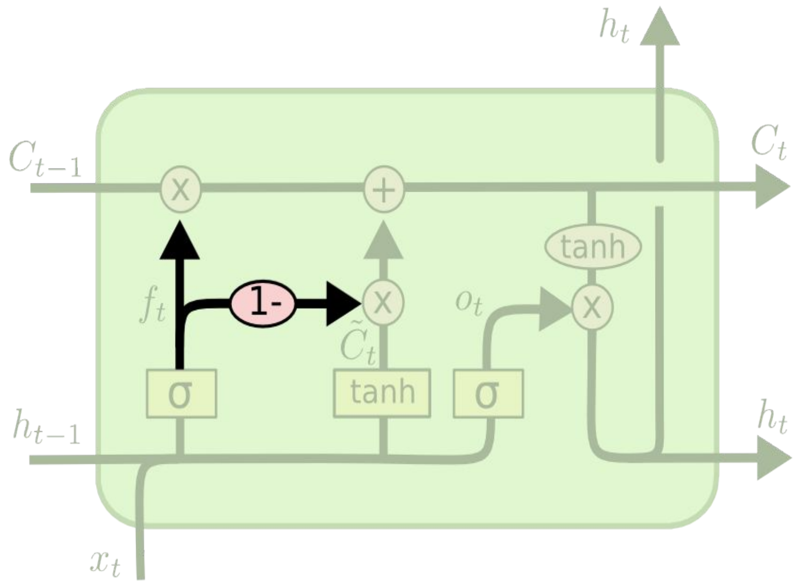
- Finally, the cell is read and then cleared. Now we try to follow the backpropagation from the point of loading 4.2 into the memory cell to the point of reading 4.2 from the cell and its subsequent clearing.

- We realize that due to the linear nature of the memory neuron, the error derivative that we receive from the read point backpropagates with negligible change until the write point because the weights of the connections connecting the memory cell through all the time layers have weights approximately equal to 1 (approximate because of the logistic output of the keep gate).

- As a result, we can locally preserve the error derivatives over hundreds of steps without having to worry about exploding or vanishing gradients.

Many variants are available

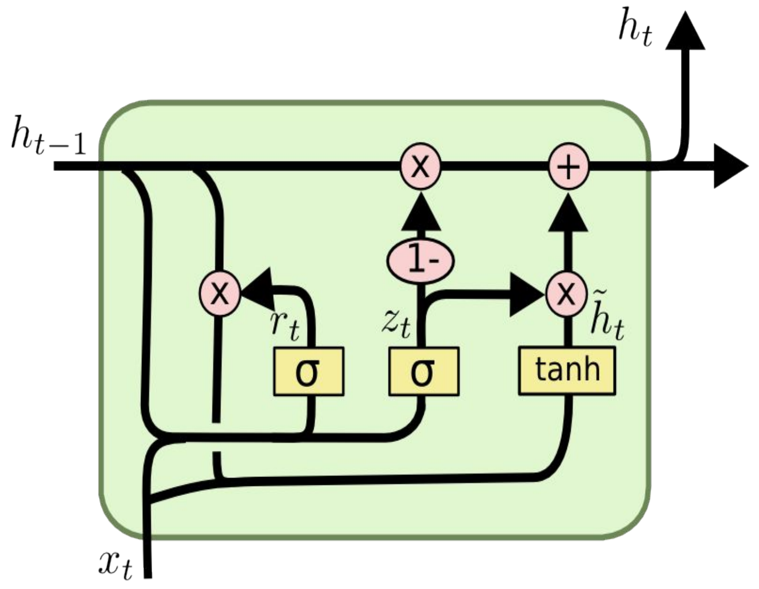
One variation is to use coupled forget and input gates. Instead of separately deciding what to forget and what we should add new information to, we make those decisions together. We only forget when we're going to input something in its place. We only input new values to the state when we forget something older.



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

Gated Recurrent Unit

A slightly more dramatic variation on the LSTM is the Gated Recurrent Unit, or GRU, introduced by Cho, et al. (2014). It combines the forget and input gates into a single “update gate.” It also merges the cell state and hidden state, and makes some other changes. The resulting model is simpler than standard LSTM models, and has been growing increasingly popular.



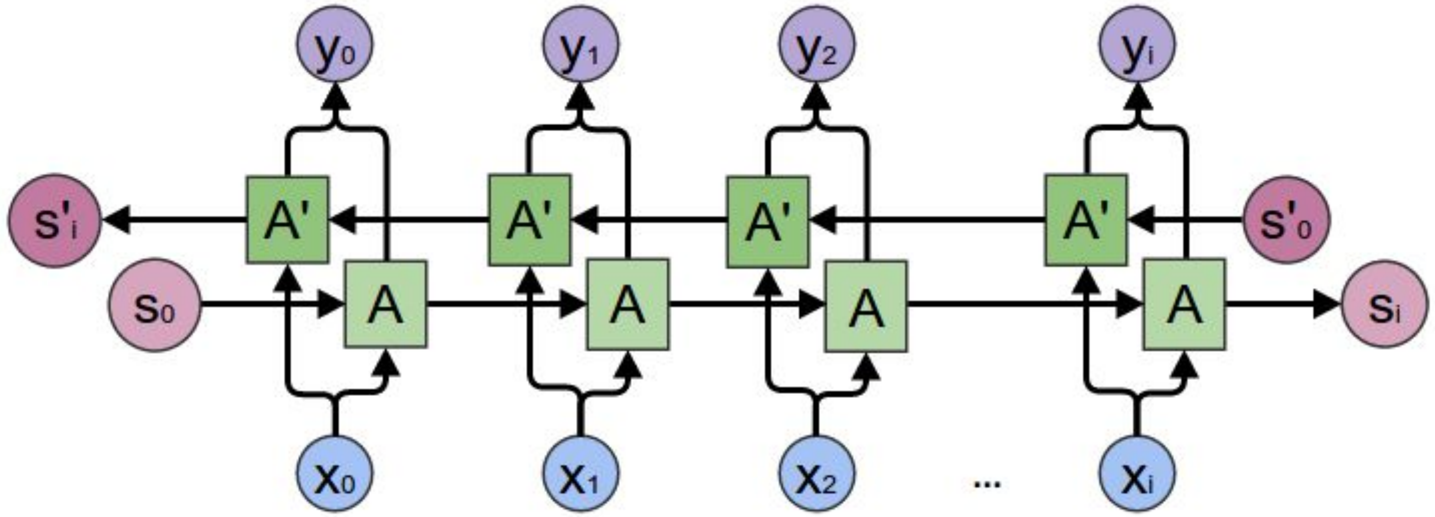
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

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

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

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

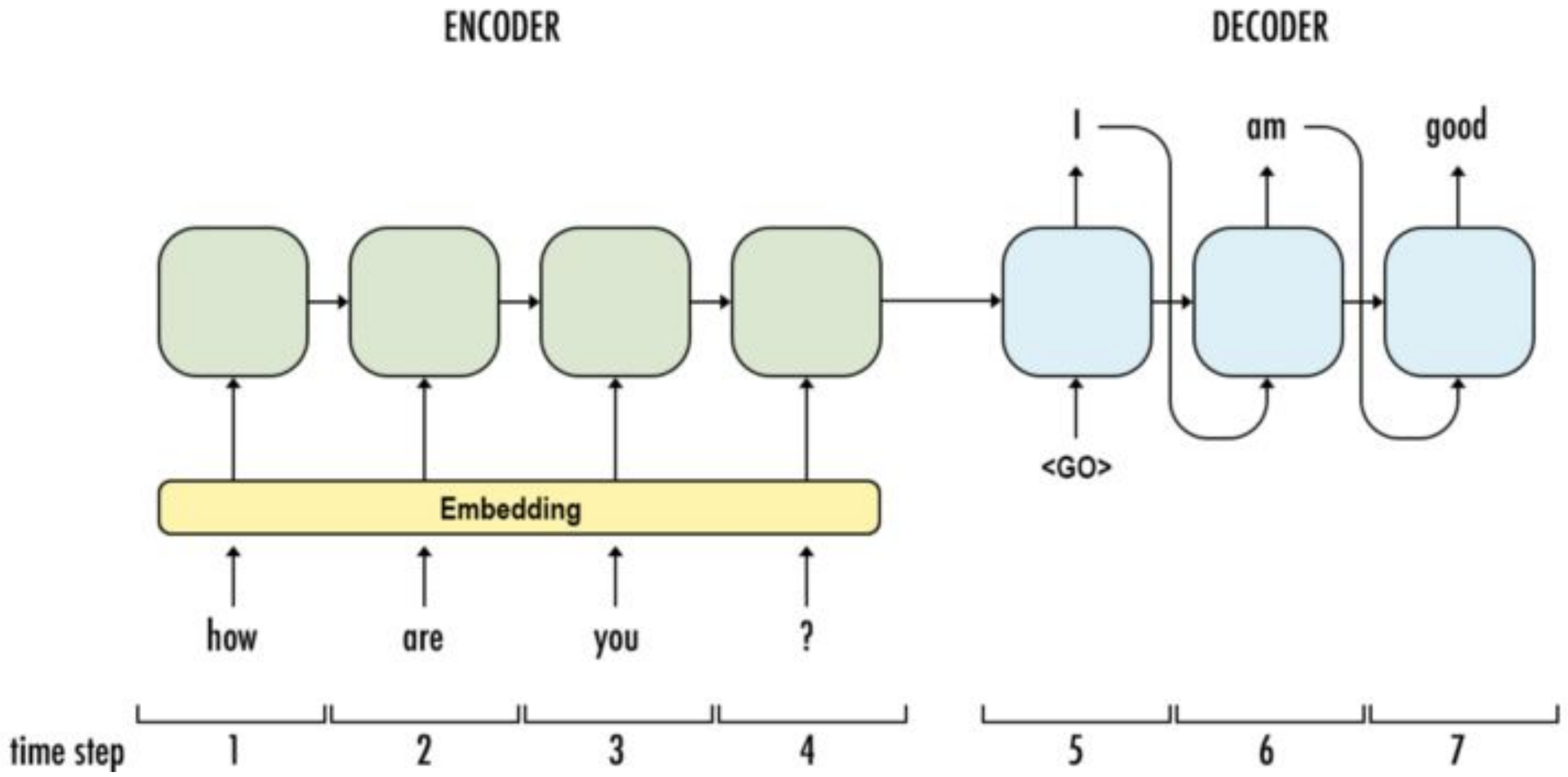
Bi-directional models



Sometimes, you might have to learn representations from future time steps to better understand the context and eliminate ambiguity.

“He said, Teddy bears are on sale”
“He said, Teddy Roosevelt was a great President”

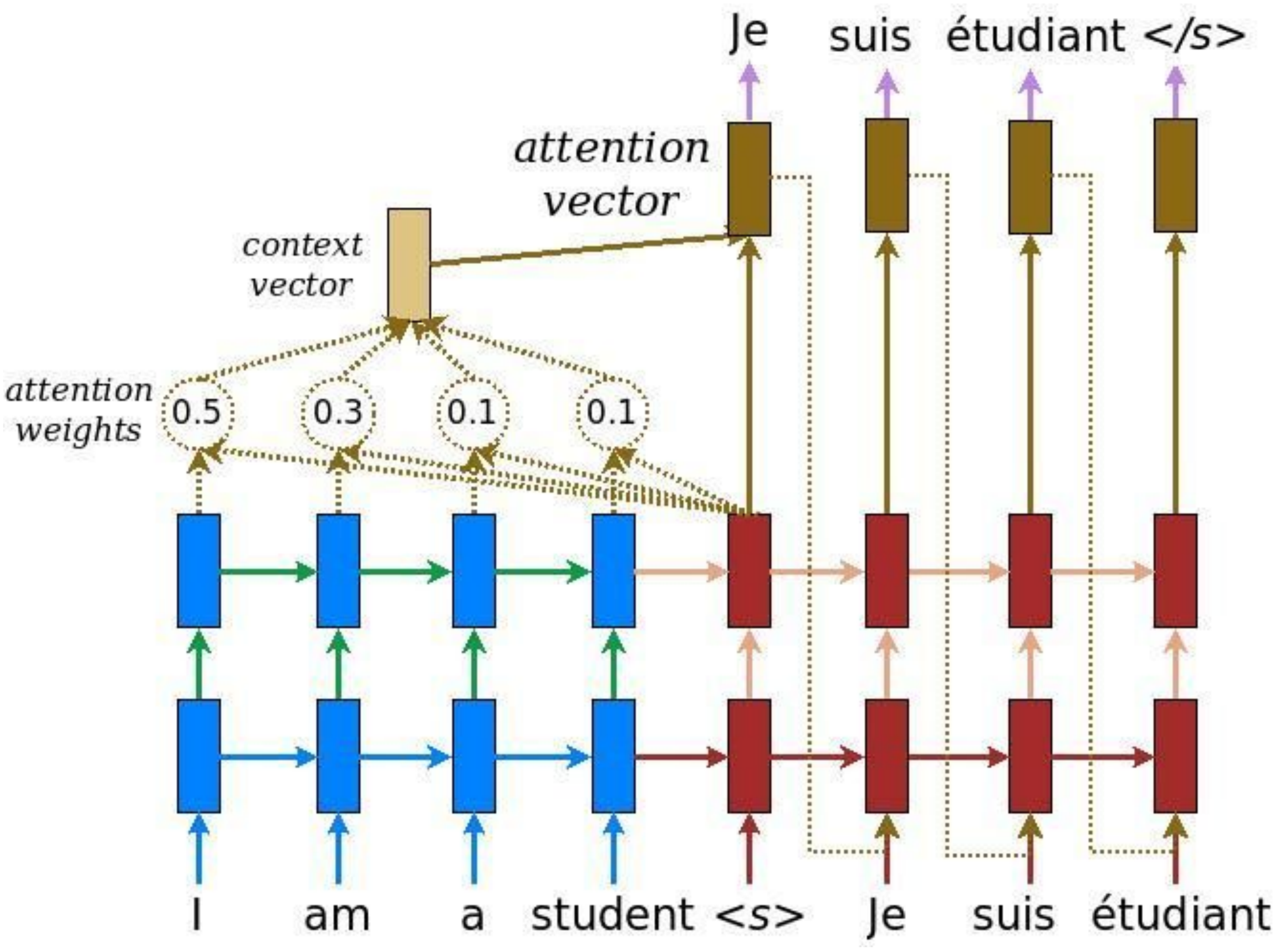
Encoder-decoder models



Encoder-decoder models with attention

- The limitation of the encode-decoder architecture and the fixed-length internal representation.
- The attention mechanism to overcome the limitation that allows the network to learn where to pay attention in the input sequence for each item in the output sequence.
- 5 applications of the attention mechanism with recurrent neural networks in domains such as text translation, speech recognition, and more.

Encoder-decoder models with attention



https://medium.com/syncedreview/a-brief-overview-of-attention-mechanism-13c578ba9129