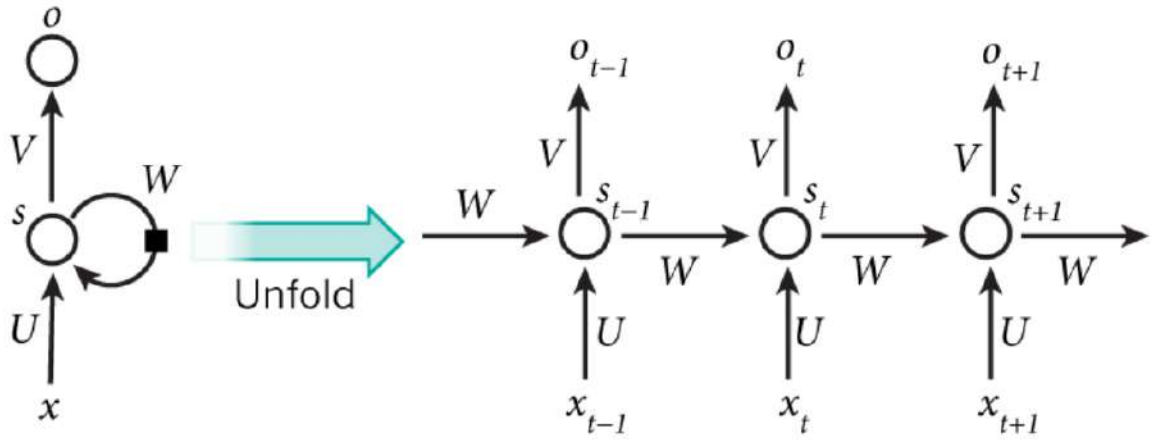
UNIT 4
RNN AND ITS CONCEPTS

What is a Recurrent Neural Network (RNN)?

- Recurrent neural network is a type of neural network in which the output form the previous step fed as input to the current step
- In traditional neural networks, all the inputs and outputs are independent of each other, but this not a good idea if we want to predict the next word in a sentence
- We need to remember the previous word in order to generate the next word in a sentence, hen traditional neural networks are not efficient for NLP applications
- RNNs also have a hidden stage which used to capture information about a sentence
- RNNs have a 'memory', which is used to capture information about the calculations made so far
- In theory, RNNs can use information in arbitrary long sequences, but practically they are limited look back only a few steps

Diagrammatic Representation



Here, x_t: input at time t, s_t: hidden state at time t, and O_t: output at time t

SENTIMENTAL ANALYSIS APPLICATION :

Unfolding means writing the network for the complete sequence, for example, if a sequence has 4 words then the network will be unfolded into a 4 layered neural network

We can think of s_t as the memory of the network as it captures information about what happened in all the previous steps

A traditional neural network uses different parameter at each layer while an RNN shares the same parameter across all the layers, in the diagram we could see that the same parameters (U, V, W) were being used across all the layers

Using the same parameters across all layers shown that we are performing the same task with different inputs, thus reducing the total number of parameters to learn

The tree set of parameter (U, V, and W) are used to apply linear transformation over their respective inputs

Parameter U transformation the input x_t to the state s_t

Parameter W transforms the previous state s_{t-1} to the current state s_t

And, parameter V maps the computed internal state s_t to the output O_t

Formula to calculate current state:

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

Here, h_t is the current state, h_{t-1} is the previous state and x_t is the current input

The equation applying after activation function (tanh) is:

$$h_t = \tanh(w_{hh} h_{t-1} + w_{xh} x_t)$$

Here, whh : weight at recurrent neuron, Wxh : weight at input neuron

- After calculating the final state, we can then produce the output
- The output state can be calculated as:

$$O_t = W_{hy} h_t$$

Here, O_t is the output state, why: weight at output layer, h_t: current state

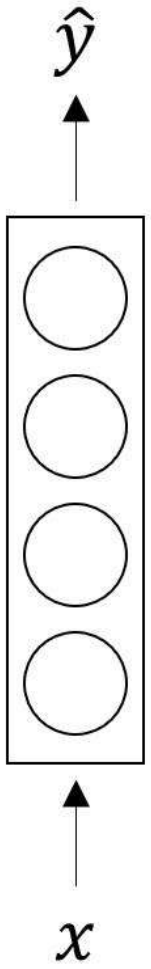
Types of RNN Architectures

The common architectures which are used for sequence learning are:

- One to one
- One to many
- Many to one
- Many to many

One to one

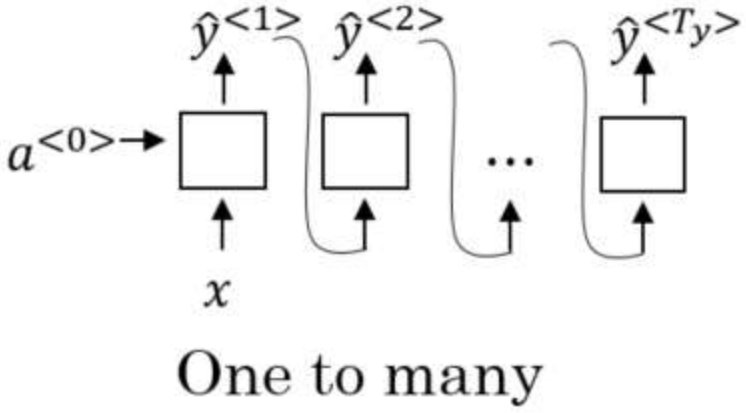
- This model is similar to a single layer neural network as it only provides linear predictions
- It is mostly used fixed-size input ‘x’ and fixed-size output ‘y’ (example: image classification)



One-to-one

One to many

- This consist of a single input ‘x’, activation ‘a’, and multiple outputs ‘y’
- Example: generating an audio stream. It takes a single audio stream as input and generates new tones or new music based on that stream
- In some cases, it propagates the output ‘y’ to the next RNN units



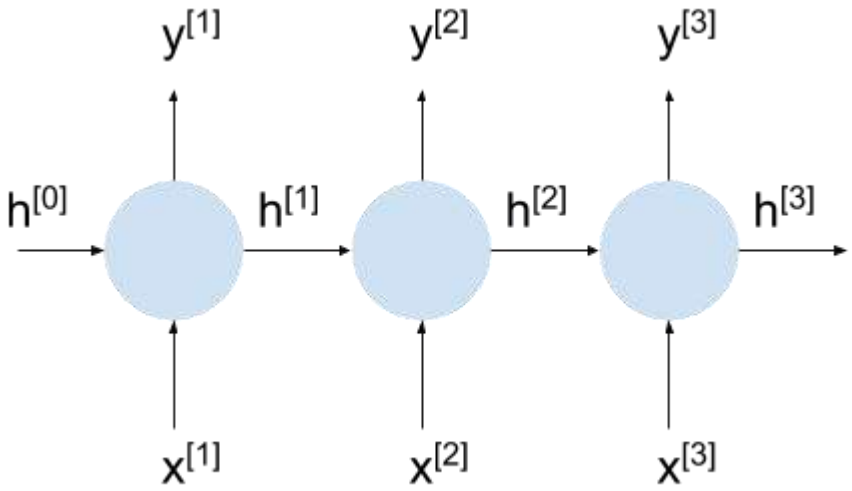
One to many

Many to one

- This consist of multiple inputs ‘x’ (such as words or sentences), activation ‘a’ and produce a single output ‘y’ at the end
- This type of architecture is mostly used to perform sentiment analysis as it processes the entire input (collection of words sentences) to produce a single output (positive, negative, or neutral sentiment)

Many to many

- In this, a single frame is taken as input for each RNN unit. A-frame represents multiple inputs ‘x’, activations ‘a’ which are propagated through the network to produce output ‘y’ which are the classification result for each frame
- It used mostly in video classification, where we try to classify each frame of the video



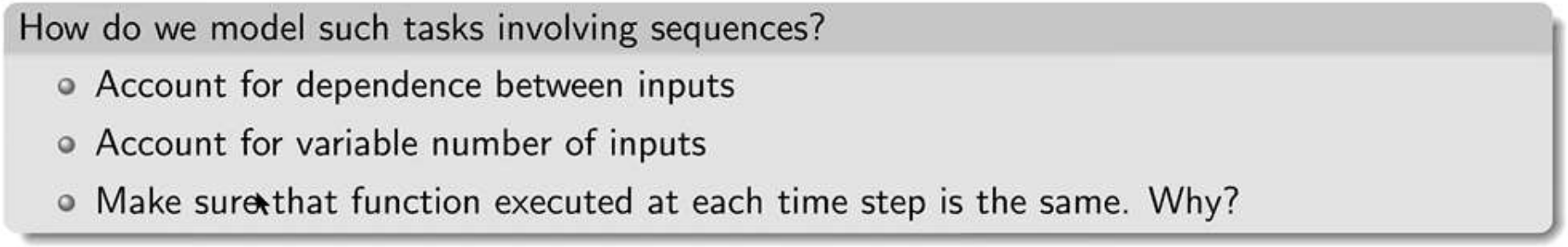
Application of RNN :

Text summarization: Summarizing the text from any literature, for example, if a news website wants to display brief summary of important news from each and every news article on the website, then text summarization will be helpful

Text recommendation: Text autofill or sentence generation in data every work by making use of RNNs can help in automating the processes and make it less time consuming

Image recognition: RNNs can be combined with CNN in order to recognize an image and give its description

Music generation: RNNs can be used to generate new music or tunes, by feeding a single tune as an input we can generate new notes or tunes of music.



26/09/2023 :

DEEP RNN :

Deep RNN (Recurrent Neural Network) refers to a recurrent neural network architecture that consists of multiple layers of recurrent units. Recurrent neural networks are designed to process sequential data by maintaining and utilizing hidden states that capture information from previous time steps.

In a standard RNN, the hidden state from the previous time step is used as input along with the current input to compute the hidden state at the current time step. However, in a deep RNN, multiple layers of recurrent units are stacked on top of each

other, allowing for the capture of more complex dependencies and patterns within the sequential data.

Each layer in a deep RNN has its own set of recurrent units, and the hidden state of each layer is passed as input to the next layer. This creates a hierarchical representation of the sequential data, where higher layers can learn higher-level abstractions while lower layers capture more fine-grained details.

APPLICATIONS :

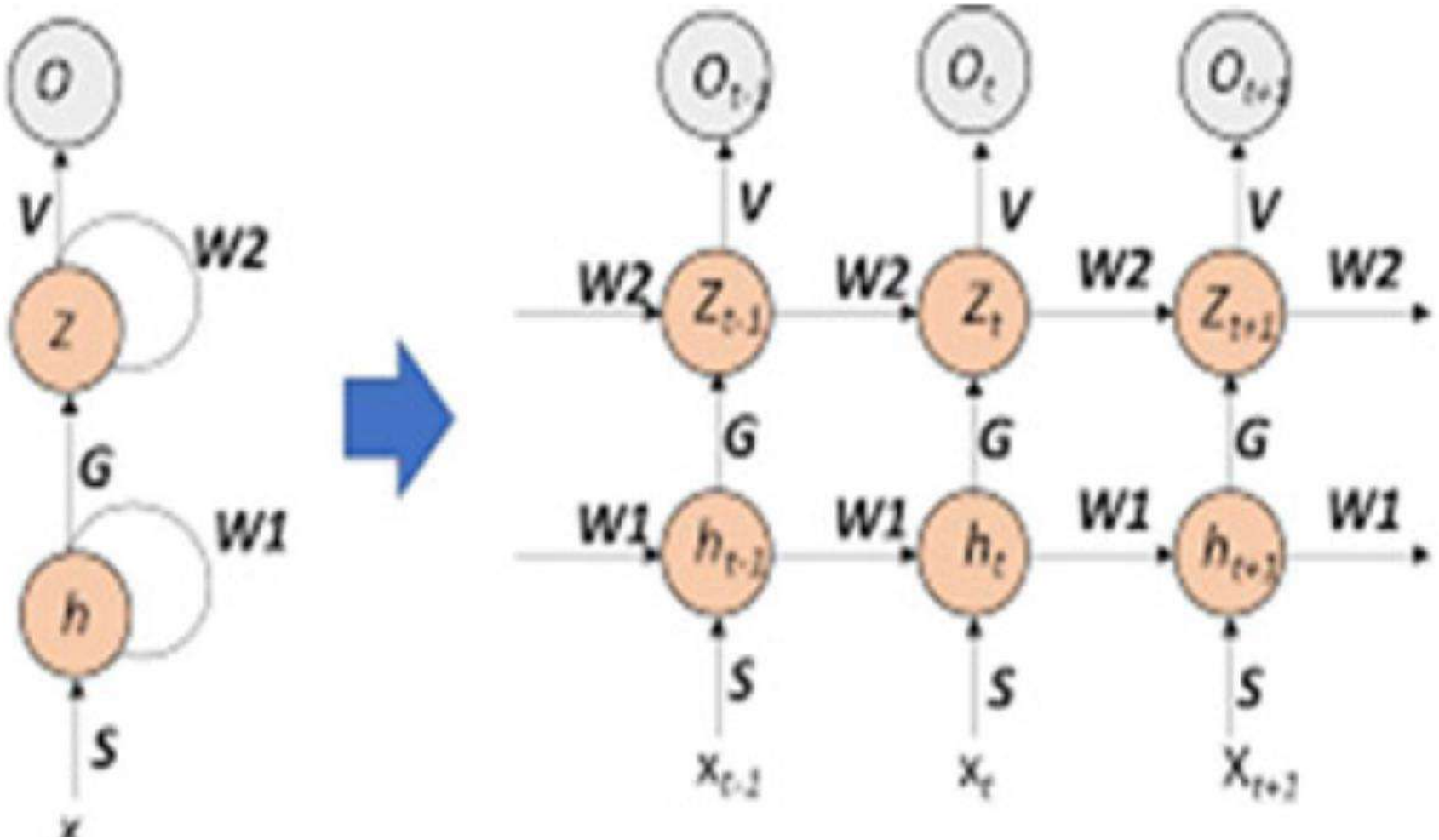
Natural language processing, speech recognition, and time series analysis. They have the ability to model long-term dependencies and capture complex sequential patterns, making them suitable for tasks that involve understanding and generating sequential data.

IMPLEMENTATION :

One popular variant of deep RNNs is the Long Short-Term Memory (LSTM) network, which introduces memory cells and gating mechanisms to better capture and control the flow of information through the network. Another variant is the Gated Recurrent Unit (GRU), which simplifies the LSTM architecture by combining the memory and hidden state into a single unit.

ADVANTAGES :

Overall, deep RNNs provide a powerful framework for modeling sequential data by leveraging multiple layers of recurrent units to learn hierarchical representations and capture long-term dependencies.


O
V
W2
z
G
W1
h
S
x
O_{t-1}
O_t
O_{t+1}
V
V
V
W2
Z_{t-1}
W2
Z_t
W2
Z_{t+1}
W2
G
G
G
W1
h_{t-1}
W1
h_t
W1
h_{t+1}
W1
S
S
S
x_{t-1}
x_t
x_{t+1}

RECURSIVE NEURAL NETWORKS :

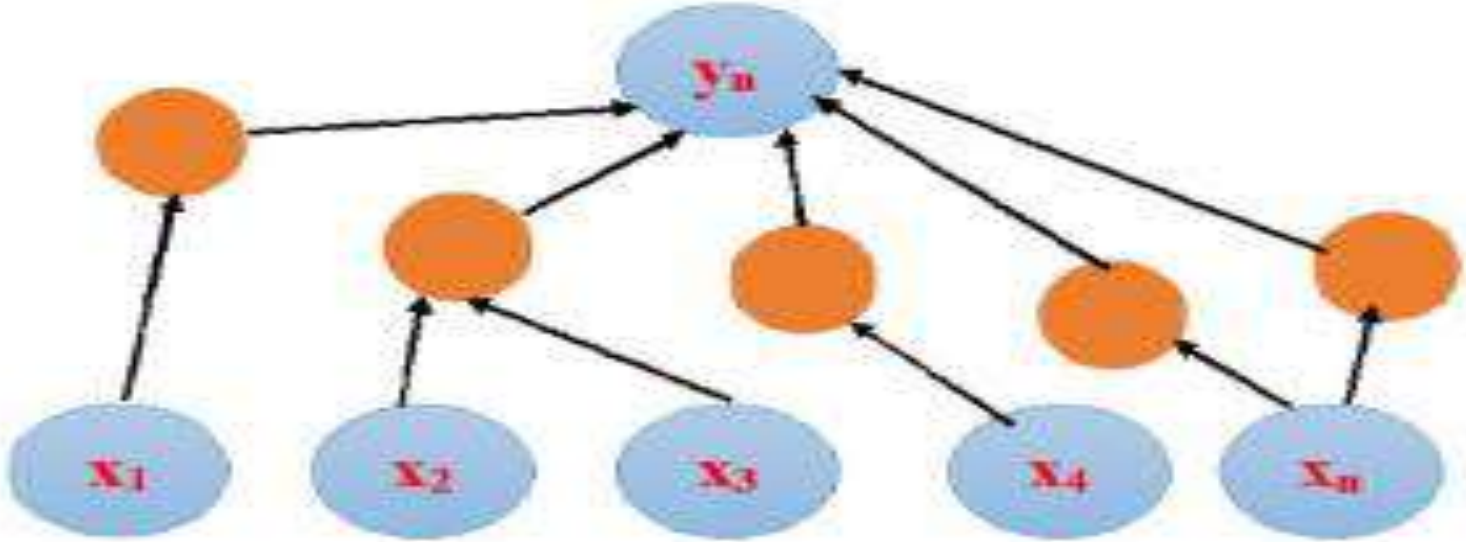
RNNs are different from recurrent neural networks (also abbreviated as RNNs), which are a type of neural network designed for sequential data processing, such as time series or text sequences. The term "recursive neural network" specifically refers to the architecture designed for structured data with recursive or hierarchical relationships.

STRUCTURE OF RECURSIVE NN :

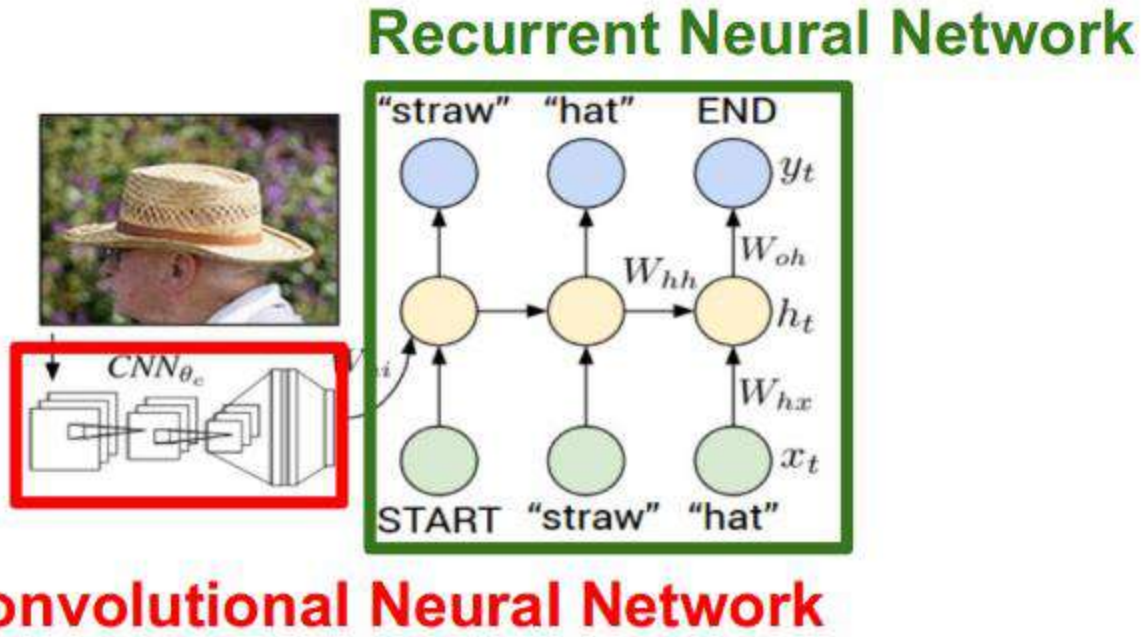
The key idea behind RNNs is the recursive application of the same neural network module to different parts of the input structure. At each level of the hierarchy, the network takes as input the representations of its child nodes and produces a representation of the current node. This process

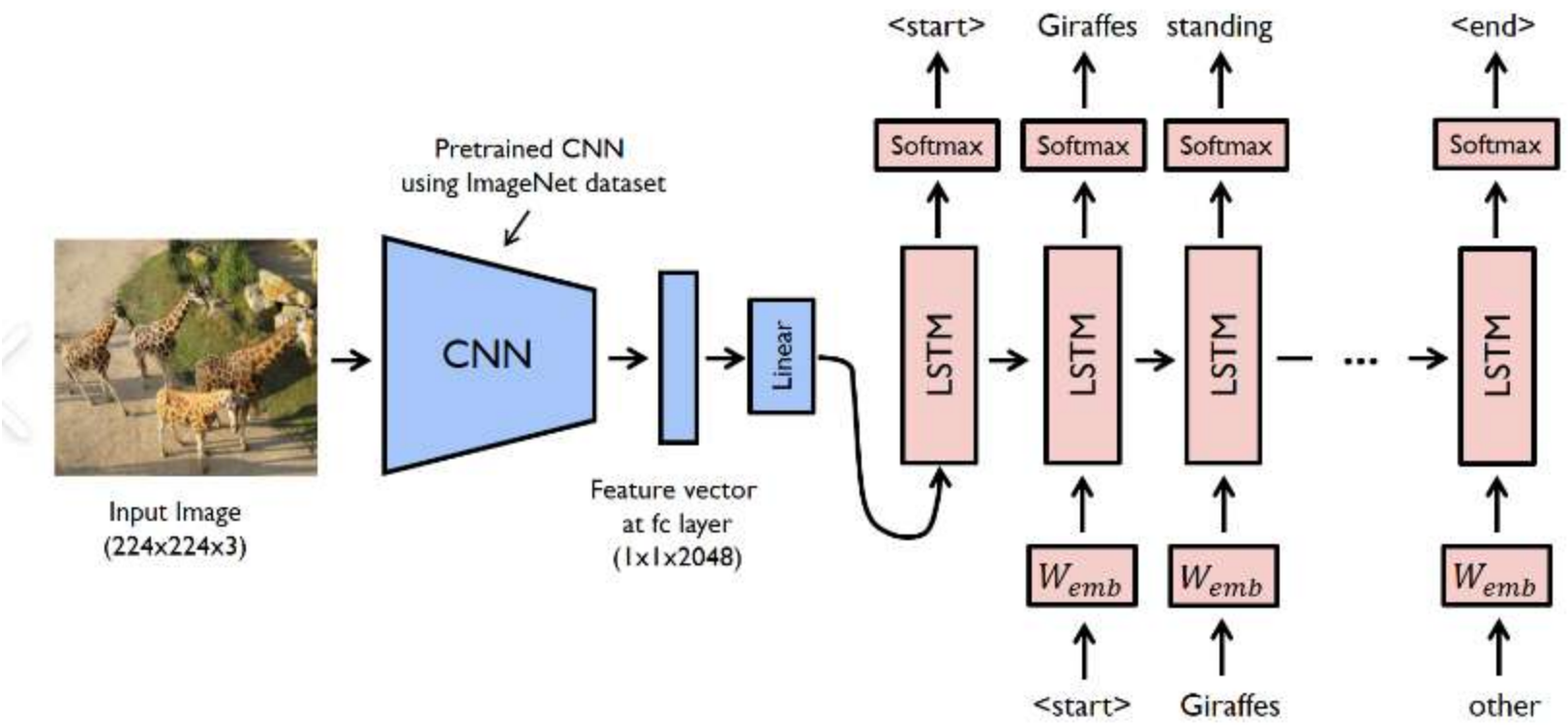
is repeated recursively until a single representation, often referred to as the root representation, is obtained.

Overall, recursive neural networks are a powerful tool for processing structured data with hierarchical relationships, and they have been successfully applied to various NLP tasks where such structures are present.


y_n
x_1
x_2
x_3
x_4
x_n

Describing images




Pretrained CNN
using ImageNet dataset
CNN
Input Image
(224x224x3)
Feature vector
at fc layer
(1x1x2048)
Linear
LSTM
LSTM
LSTM
LSTM
Softmax
Softmax
Softmax
Softmax
<start>
Giraffes
standing
<end>
W_{emb}
W_{emb}
W_{emb}
<start>
Giraffes
other

Implementing Time Series forecasting Concept using RNN :

Lab Exercise – 7 DOS : 3/10/2023

Preparation dataset for RNN model by collecting 2D data and convert the them as 3D data.

Chapter 6 : Do it and print the output

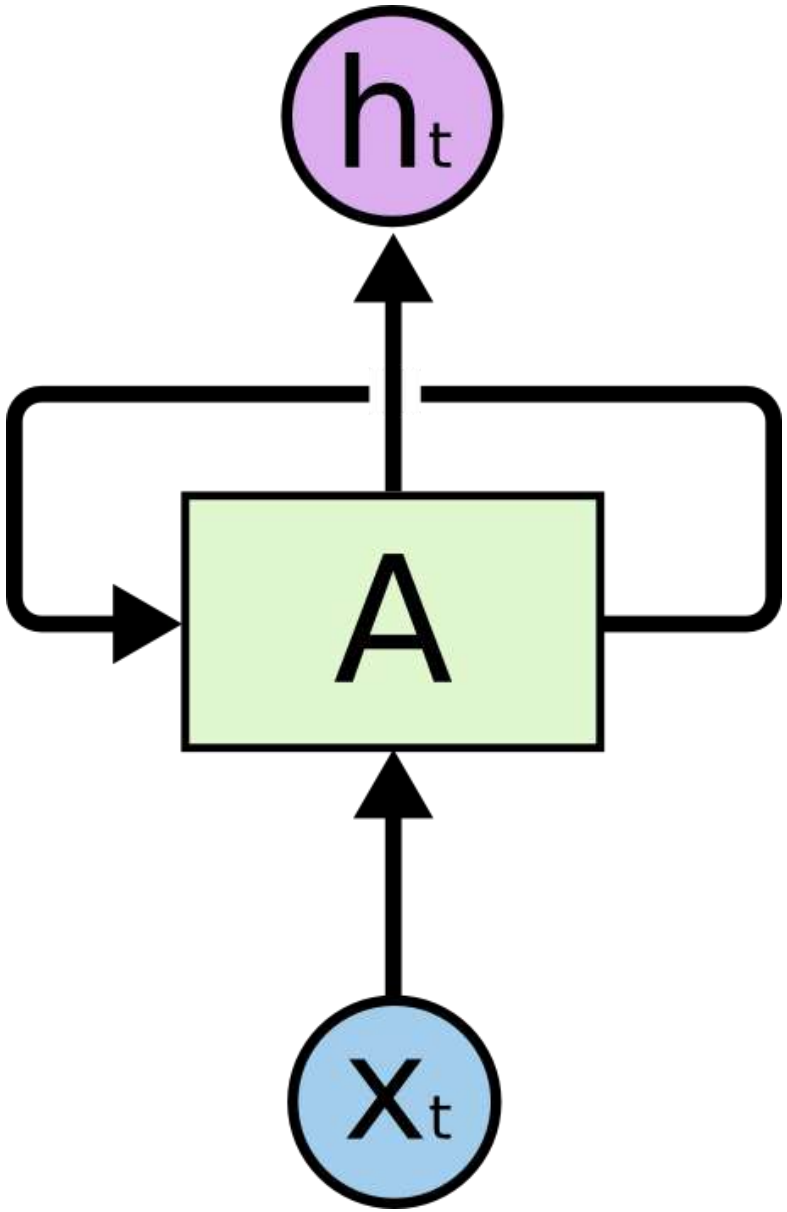
Creating a CRNN model to recognize text in an image (Part-2) | TheAILearner – Lab exercise 8

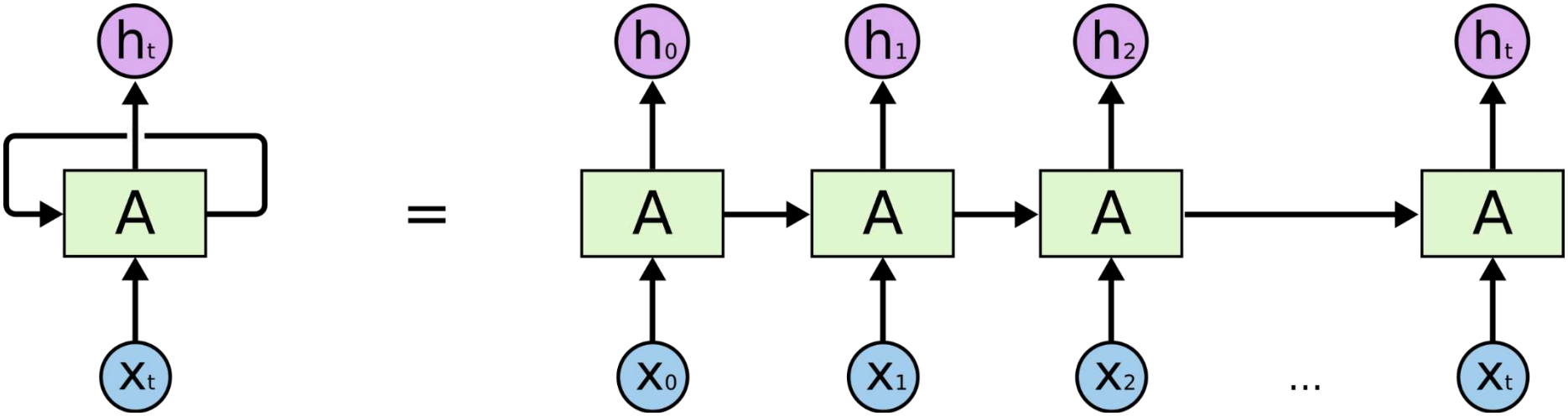
Assignment 4 : Chapter -2 :DOS : 30/09/2023.

2.1 to 2.10. – What are the Steps involved in preparing Time Series Forecasting model using RNN concepts.

30/09/2023:

What is LSTM and why would we need it ?


h_t
A
x_t

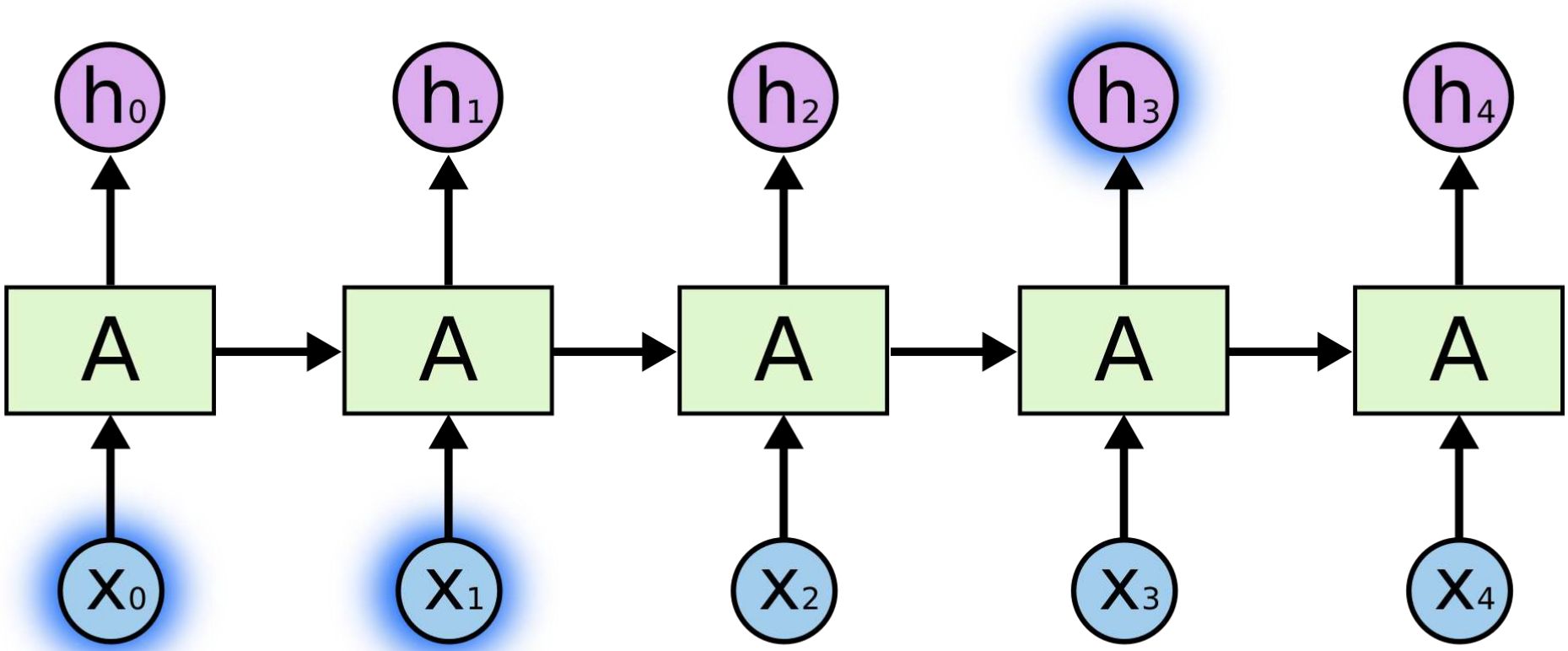


Sometimes, we only need to look at recent information to perform the present task.

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 ne xt 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.



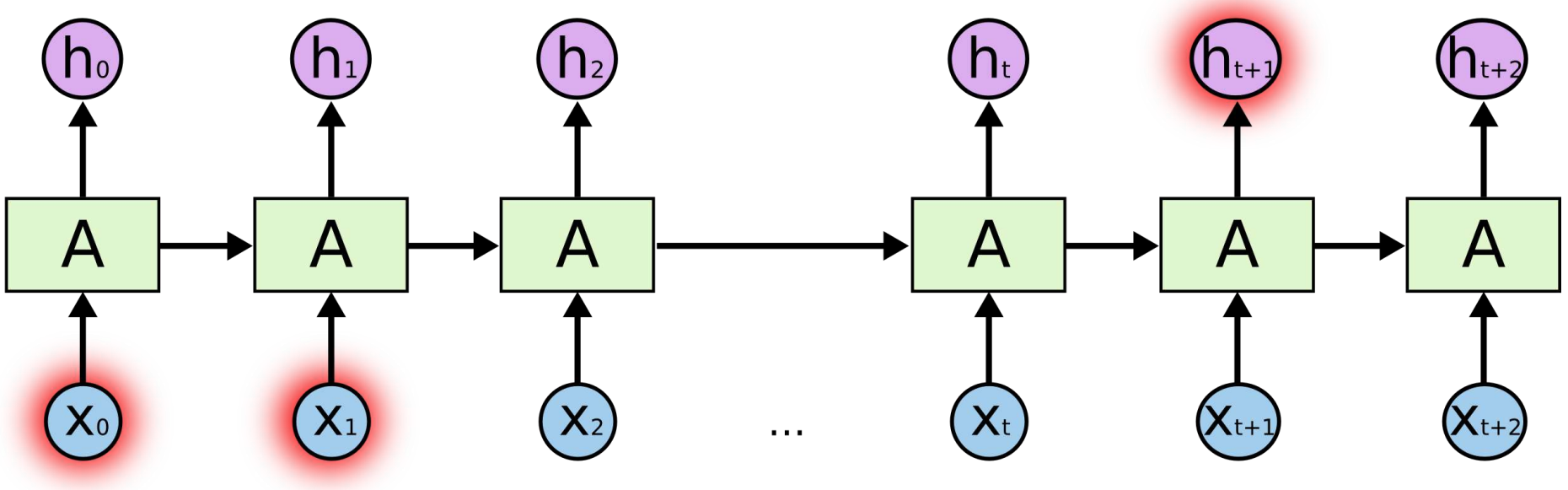
But there are also cases where we need more context.

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.

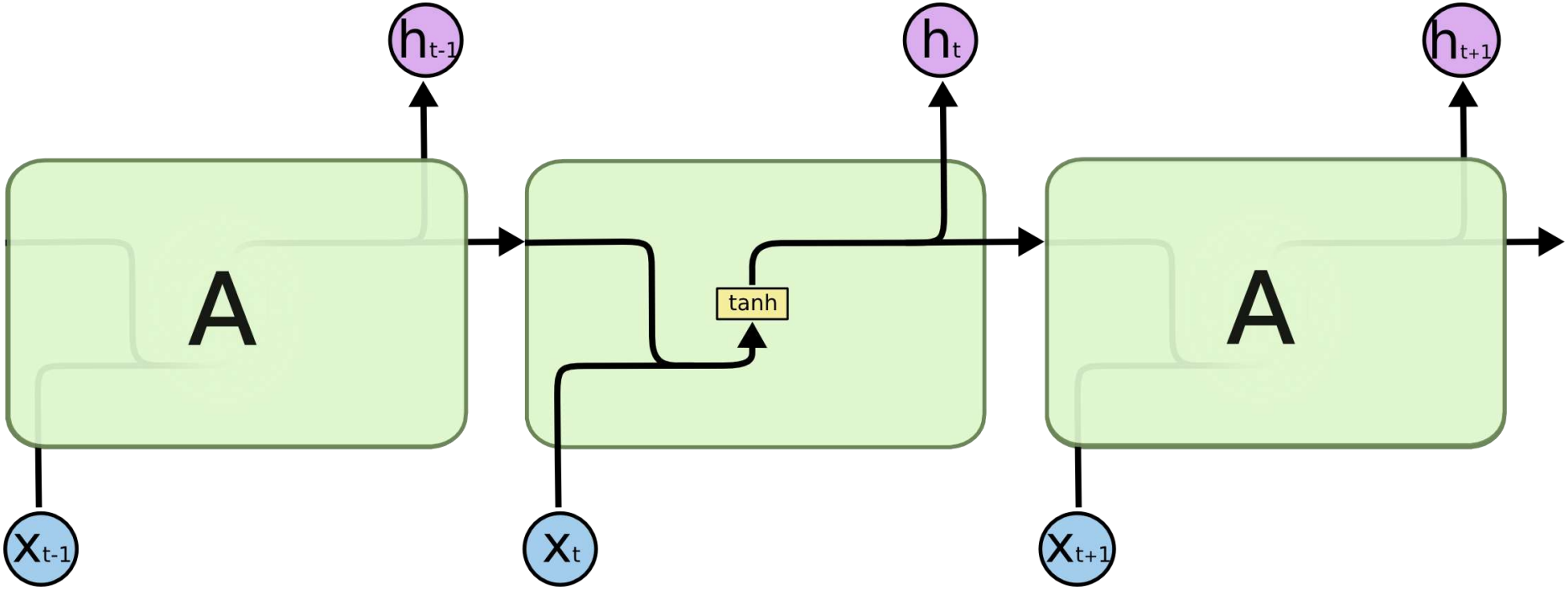
LSTM Networks :

Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies.

.

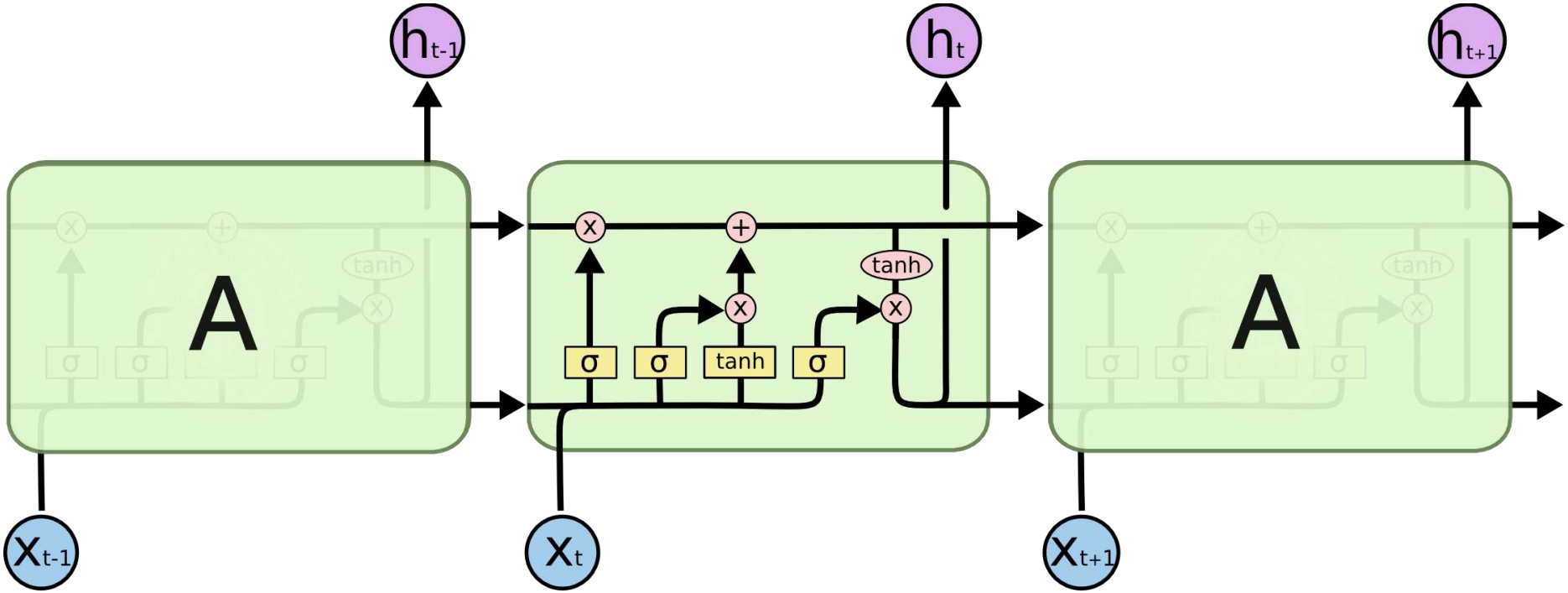
All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

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



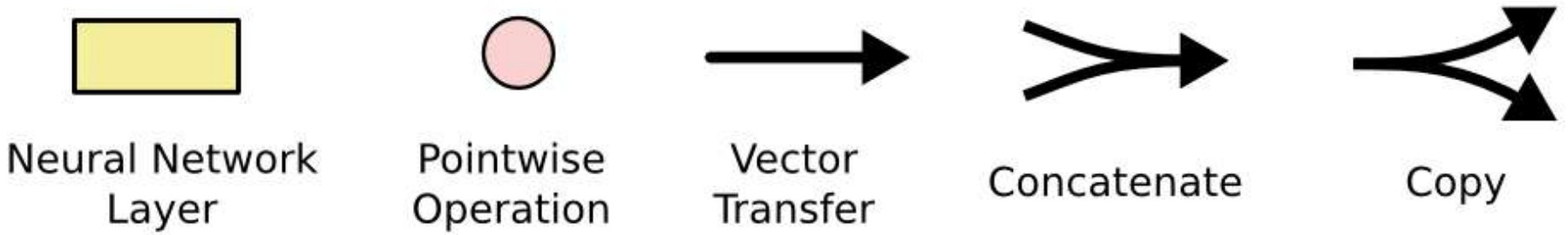
The repeating module in a standard RNN contains a single layer.

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.



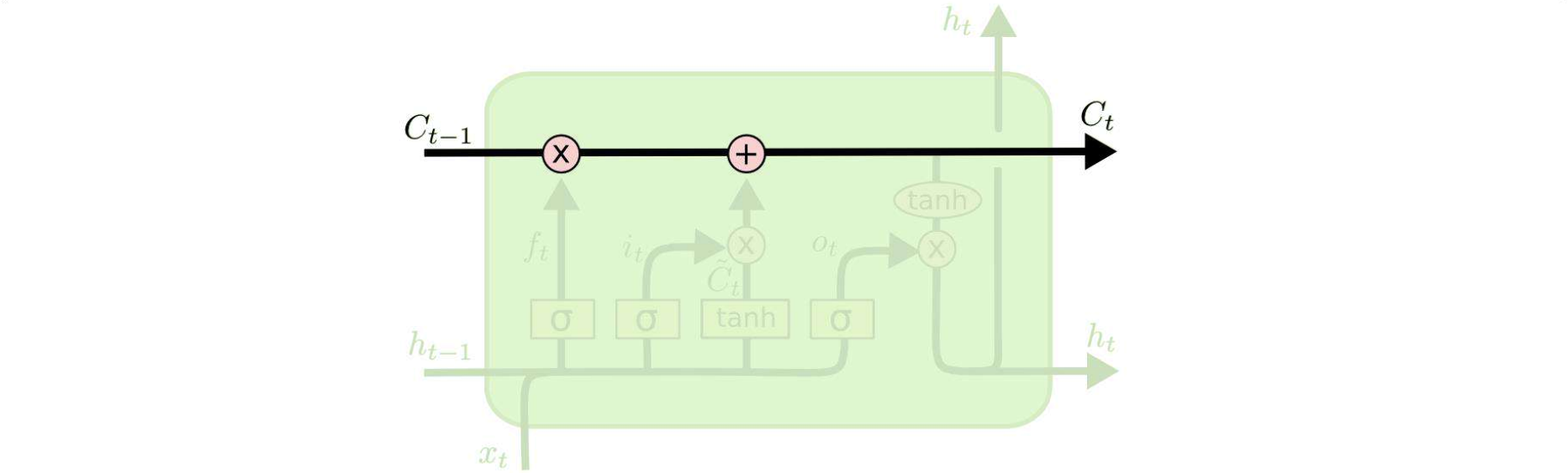
The repeating module in an LSTM contains four interacting layers.

STEP BY STEP OPERATION HAPPENS INSIDE LSTM



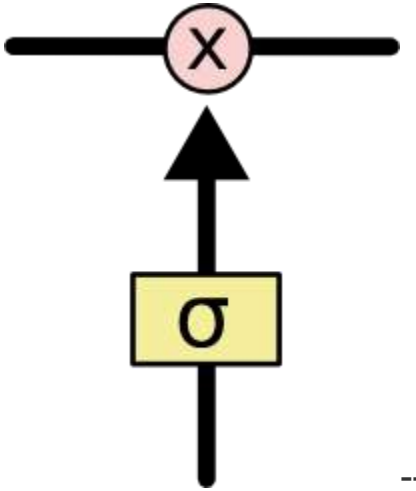
The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.

The Core Idea Behind LSTMs :



- 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 pointwise multiplication operation.



------ > FORGETGATE

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.

- INPUT GATE
- OUTPUT GATE
- FORGET GATE

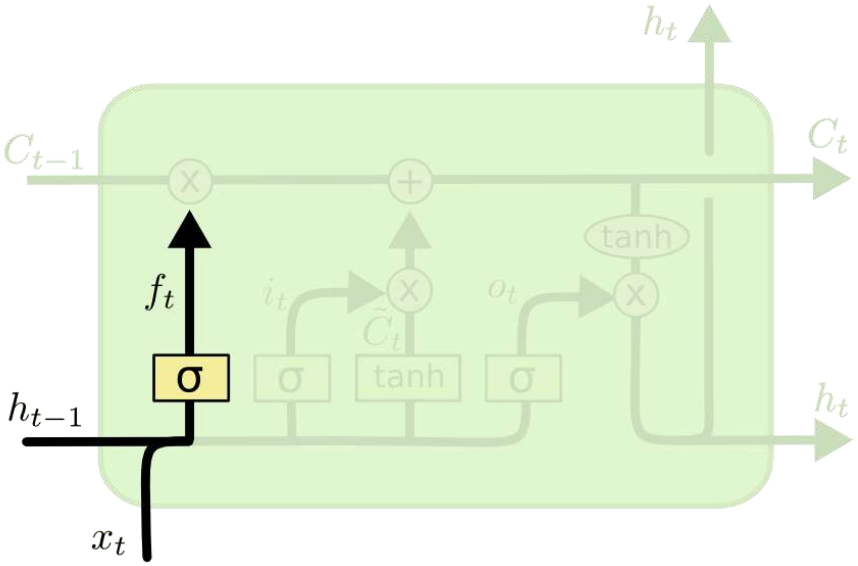
Step-by-Step LSTM Process:

STEP 1 :

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*h*−1 and xt, and outputs a number between 00 and 11 for each number in the cell state Ct−1−1. A 11 represents "completely keep this" while a 00 represents "completely get rid of this."

Example :

Let's go back to our 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.

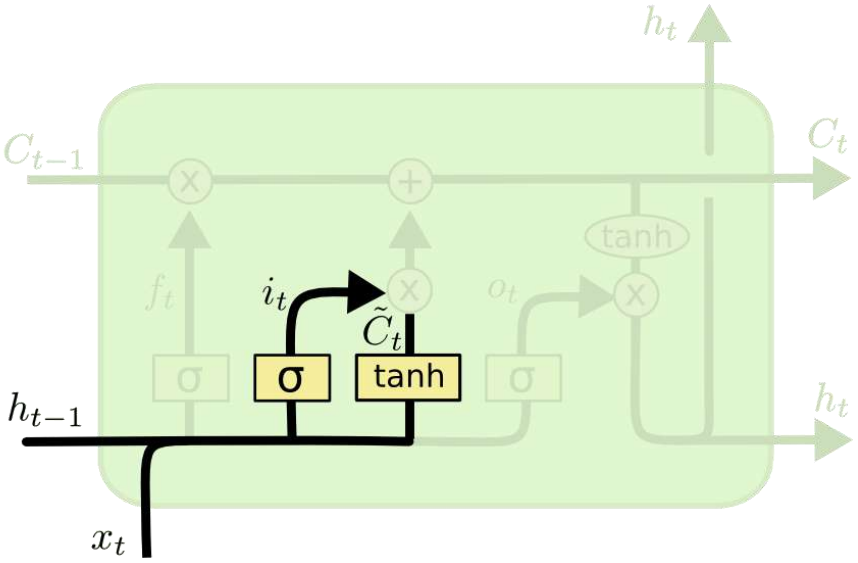


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

STEP 2 : INPUT LAYER :

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, 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.

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.



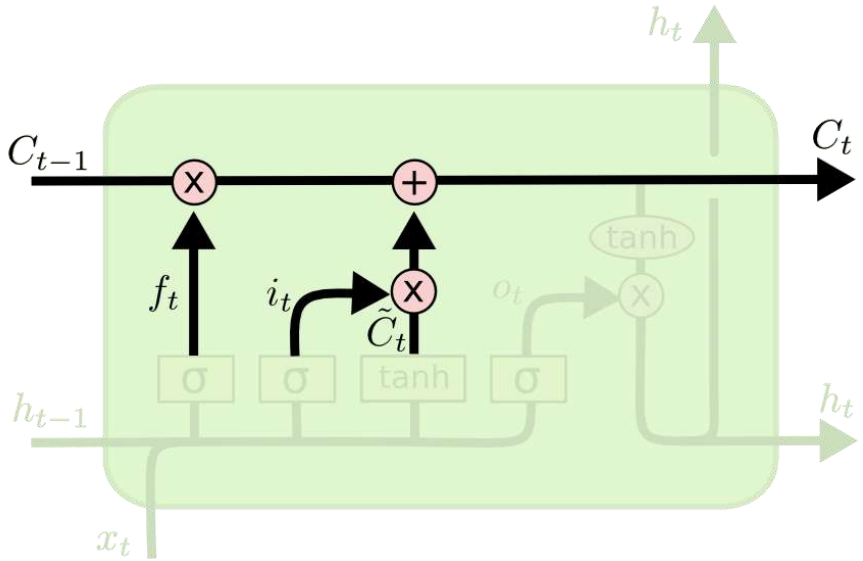
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

It’s now time to update the old cell state, Ct−1��−1, into the new cell state Ct��. 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 it∗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.

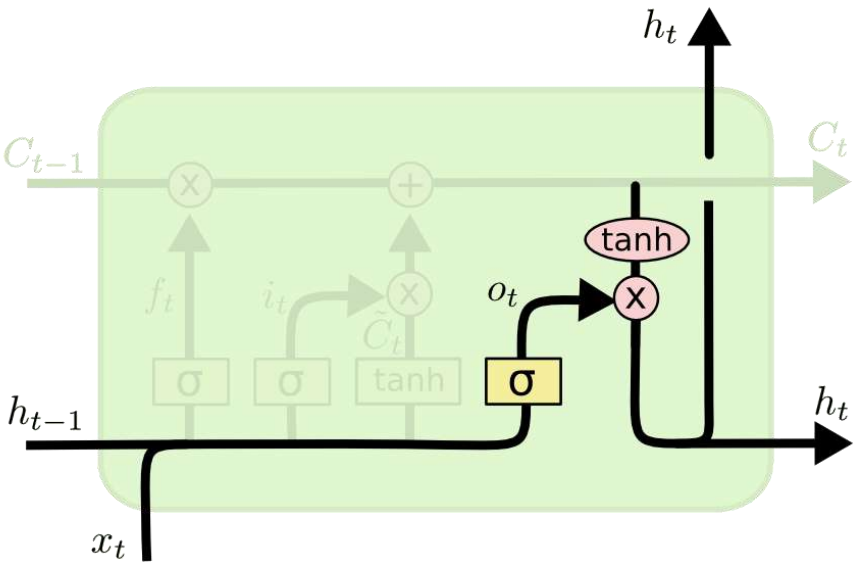


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

STEP 3 : OUTPUT GATE

Finally, we need to decide what we're going to output. This 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 tanhtanh (to push the values to be between −1−1 and 11) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case that's what is coming next. For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that's what follows next.



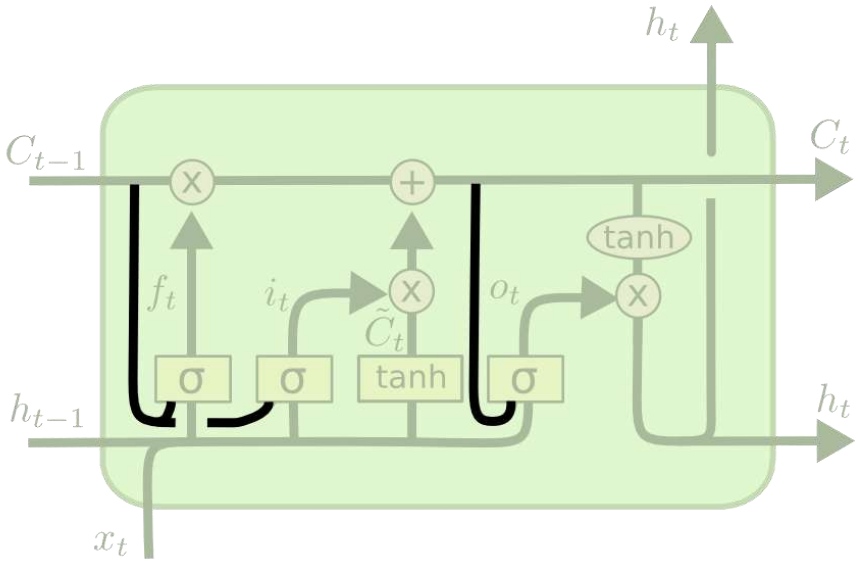
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

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

Variants on Long Short Term Memory

What I've described so far is a pretty normal LSTM. But not all LSTMs are the same as the above. In fact, it seems like almost every paper involving LSTMs uses a slightly different version. The differences are minor, but it's worth mentioning some of them.

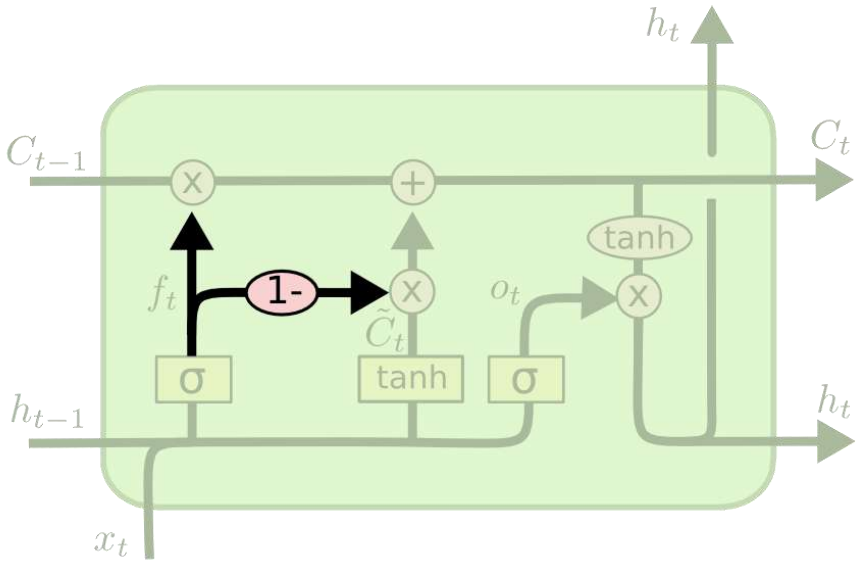
One popular LSTM variant, introduced by Gers & Schmidhuber (2000), is adding "peephole connections." This means that we let the gate layers look at the cell state.



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

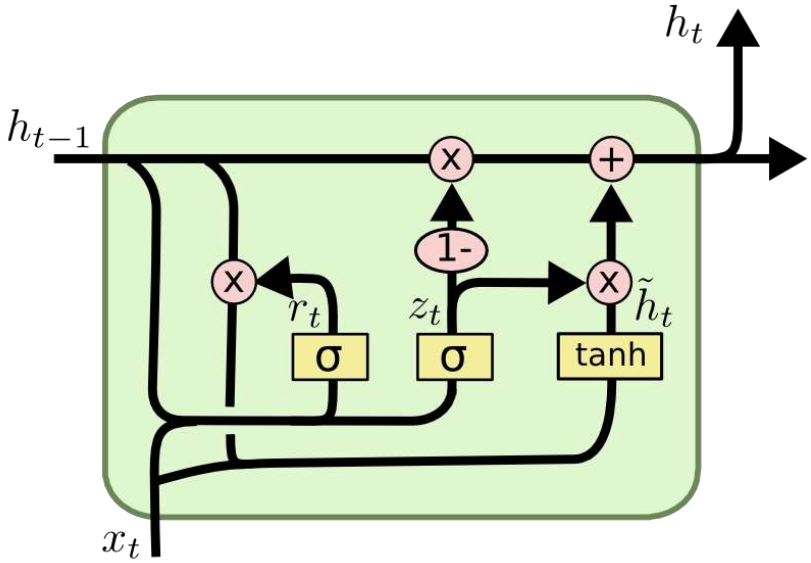
The above diagram adds peepholes to all the gates, but many papers will give some peepholes and not others.

Another 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.



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

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.



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

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

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

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

These are only a few of the most notable LSTM variants. There are lots of others, like Depth Gated RNNs by Yao, et al. (2015). There's also some completely different approach to tackling long-term dependencies, like Clockwork RNNs by Koutnik, et al. (2014).

Which of these variants is best? Do the differences matter? Greff, et al. (2015) do a nice comparison of popular variants, finding that they're all about the same. Jozefowicz, et al. (2015) tested more than ten thousand RNN architectures, finding some that worked better than LSTMs on certain tasks.