Recurrent Neural Networks
RNN

Motivation, Architecture, Problems and Modifications

AUA, DS330 Deep Learning 2021, Hovhannes Tamoyan

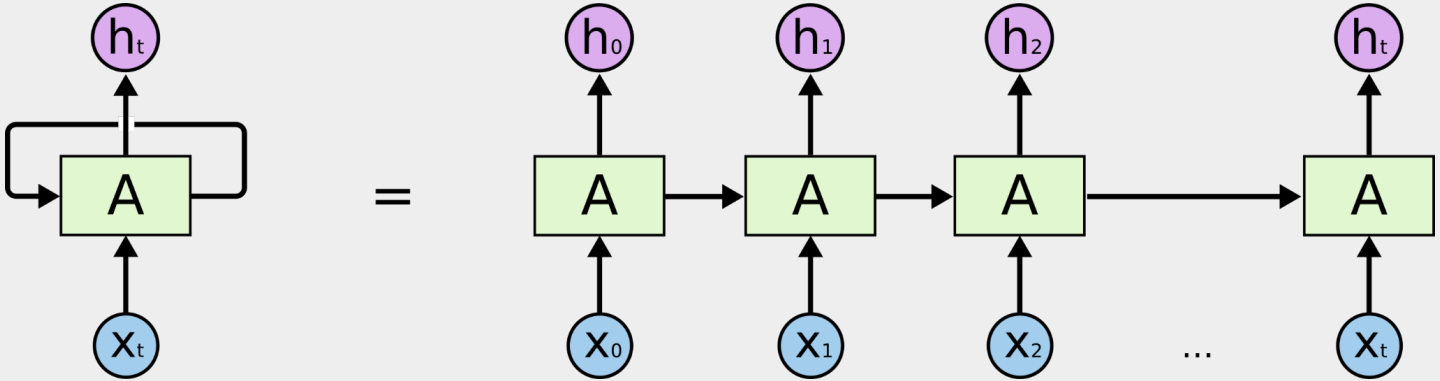
Motivation

We need to input a sequence (sentence, equation etc) of tokens in our NN.

We need an NN that can process **any length input**.

Humans don’t start their thinking from scratch every second. As you read this slide, you understand each word based on your understanding of previous words. Your **thoughts have persistence**.

The solution is in RNNs!


h_t
A
x_t
=
h_0
h_1
h_2
h_t
A
A
A
A
x_0
x_1
x_2
...
x_t

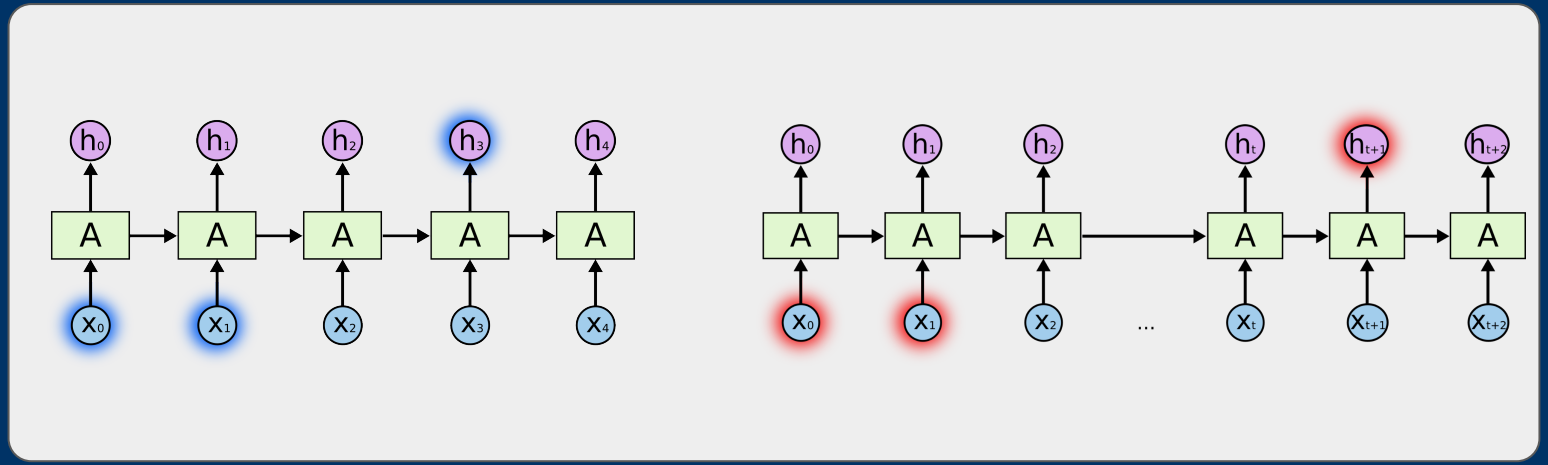
Definition

A Recurrent Neural Network (RNN) is a type of NN which uses sequential data and for every state (t) it has two inputs: the previous state's output (y_{t-1}) and the regular input (x_t).

The drawback of vanilla RNNs

"the clouds are in the ***sky***."

"I grew up in France… I speak fluent ***French***."



The drawback of standart RNNs

Unfortunately, as the gap between the relevant information and the point where it is needed grows, RNNs become unable to learn to connect the information.

In theory, RNNs are absolutely capable of handling such “long-term dependencies.”

Sadly, in practice, RNNs don’t seem to be able to learn them.

Vanishing gradients problem.

Thankfully, LSTMs don’t have this problem!

Long Short-Term Memory Networks (LSTM)

LSTMs are capable of learning long-term dependencies.

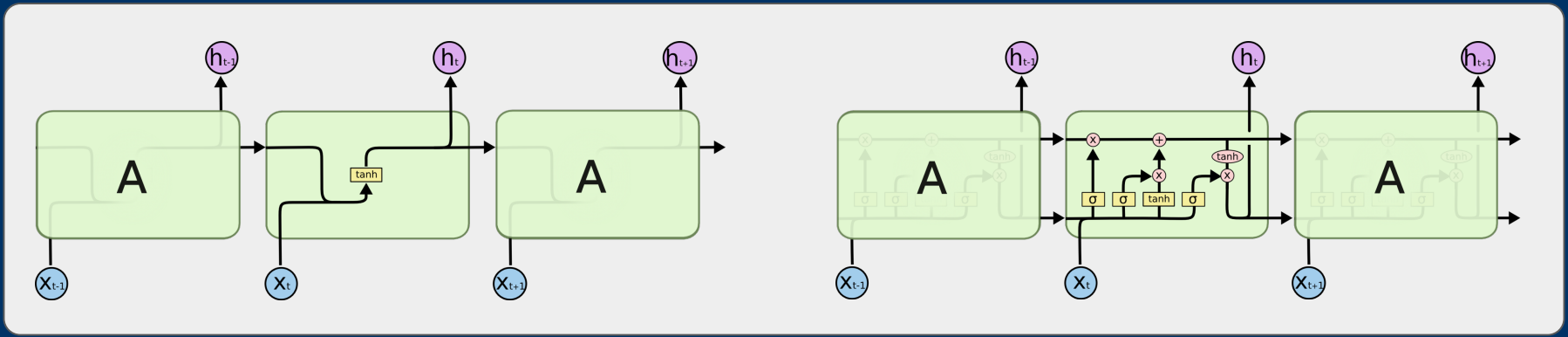
LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

by Hochreiter & Schmidhuber (1997)

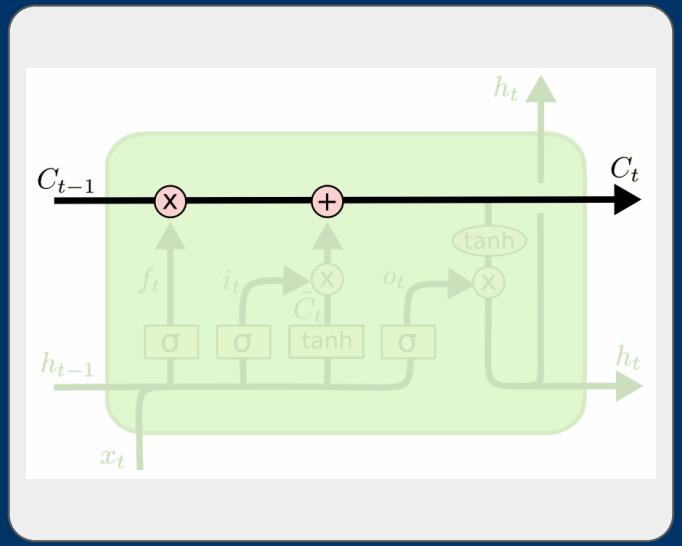
LSTM Architecture

In standard RNNs, the repeating module will have a very simple structure, such as a single tanh layer.

LSTMs also have this chain like structure, but the repeating module has a different structure. It has four layers, interacting in a very special way.



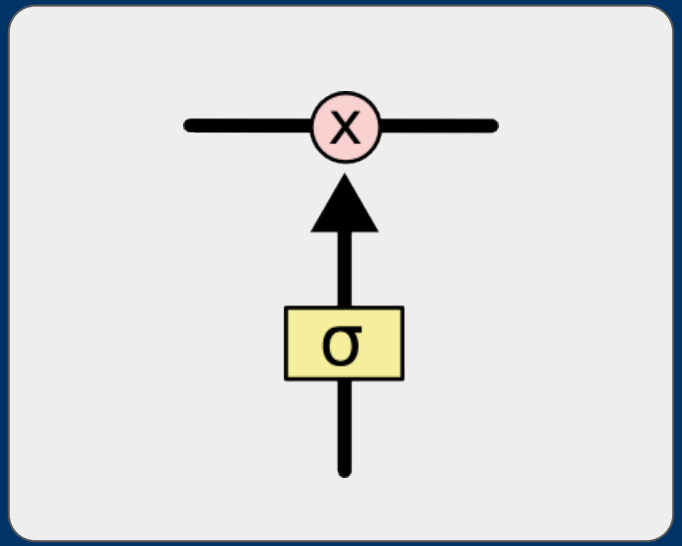
The main idea behind LSTMs



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

The main idea behind LSTMs contd.

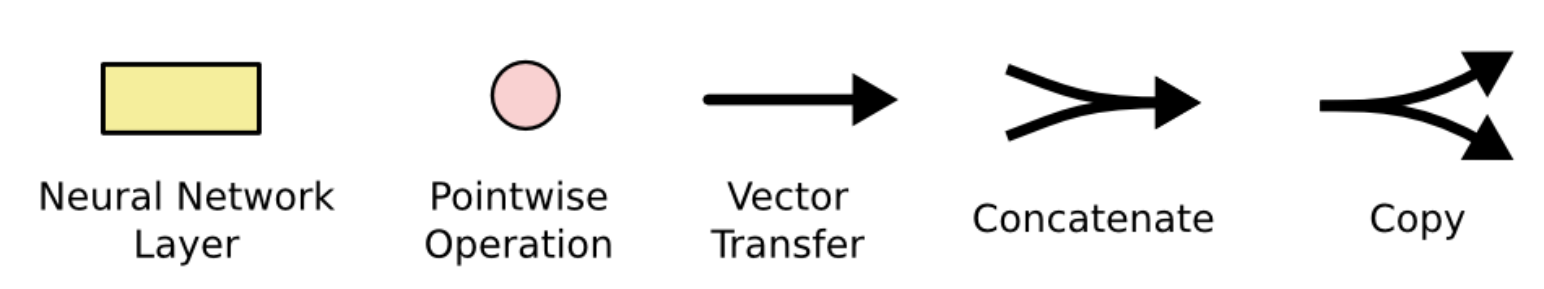


Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.

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.

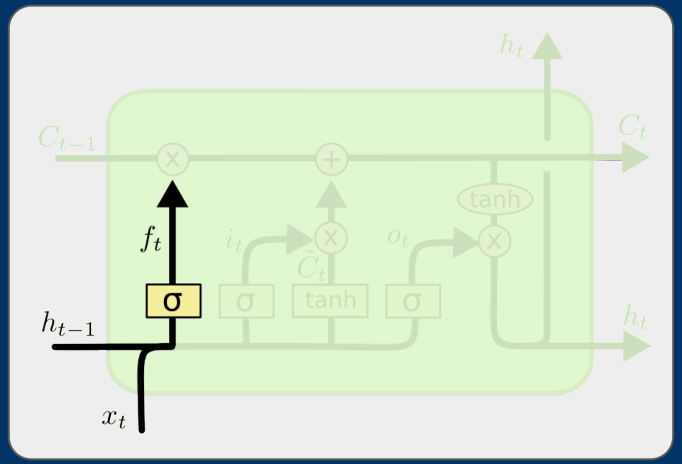
Some notations before we continue



h_t - hidden state at time t

C_t - Cell state at time t

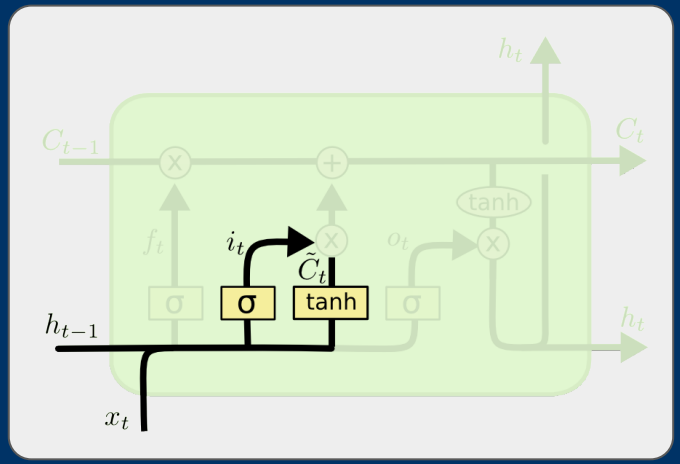
LSTM - Forget Gate



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**."

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

LSTM - Input Gate



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

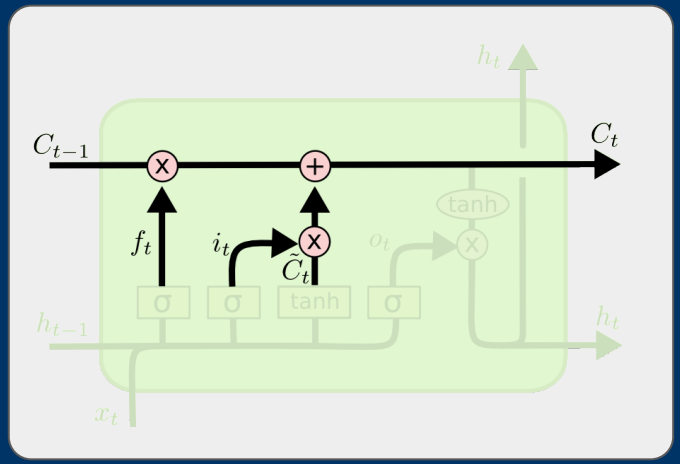
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

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, $\tilde{C}_t$, that could be added to the state.

In the next step, we'll combine these two to create an update to the state.

LSTM - Update Mechanism

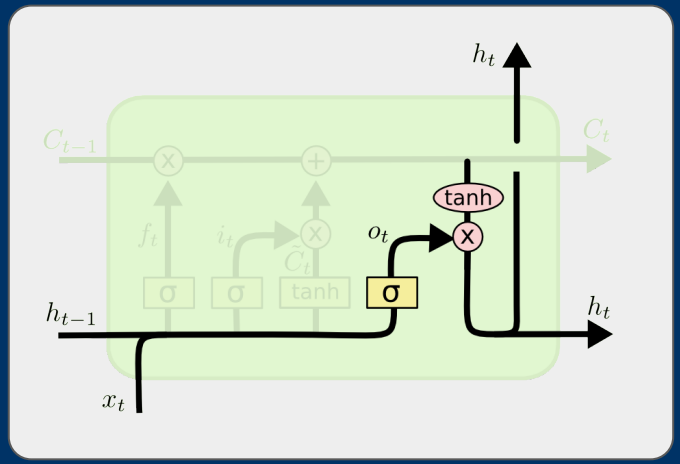


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

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 f_t , 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.

LSTM - Output Gate



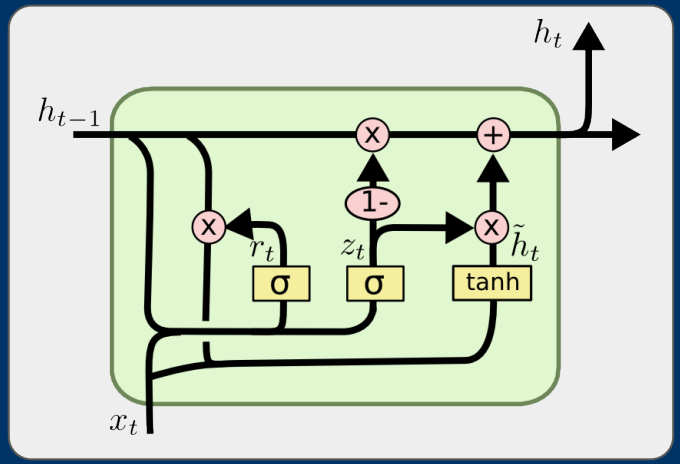
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

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

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 *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.

Gated Recurrent Unit (GRU)



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

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

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

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

A slightly modified variant of LSTM.

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.

by Cho, et al. (2014)

LSTM vs GRU

Now we have seen the operation of both the layers to combat the problem of vanishing gradient. So you might wonder which one to use?

According to empirical evaluation, there is not a clear winner.

GRU uses less training parameter and therefore uses less memory and executes faster than LSTM whereas LSTM is more accurate on a larger dataset. One can choose LSTM if you are dealing with large sequences and accuracy is concerned, GRU is used when you have less memory consumption and want faster results.

Thank You