

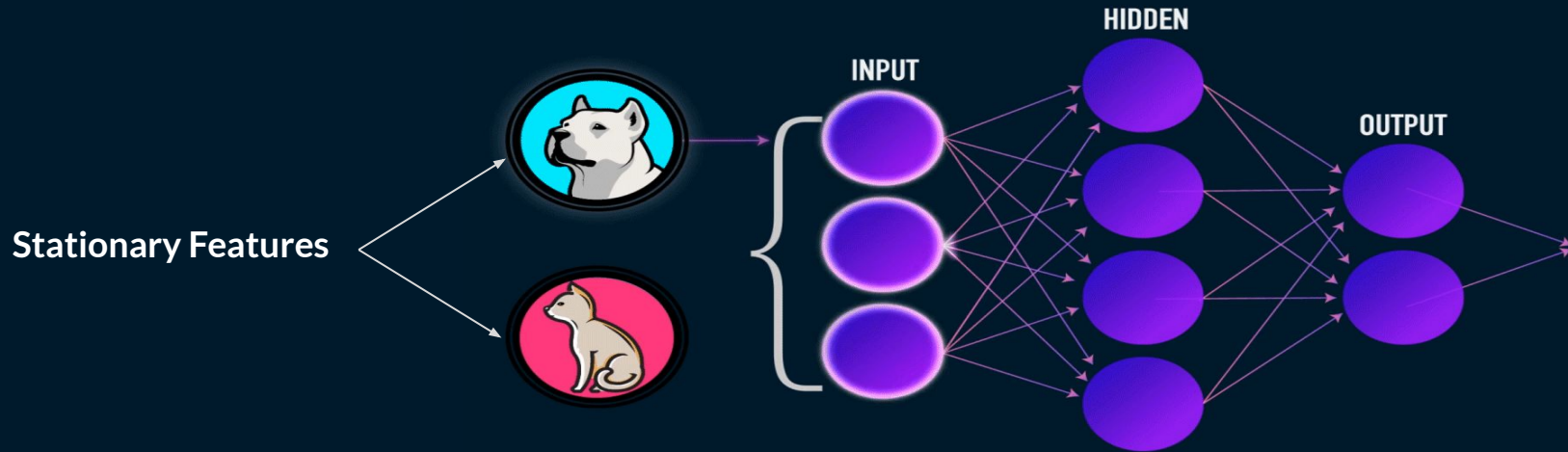
A decorative graphic on the left side of the slide consisting of overlapping geometric shapes. It includes a blue parallelogram, a light green parallelogram, and a dark blue parallelogram, all oriented diagonally.

# Sequence Models

Recurrent Neural Networks

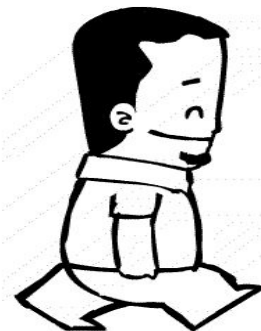
# Why Sequence Models ?

## Vanilla Network



# What if our Features are Non-Stationary ?

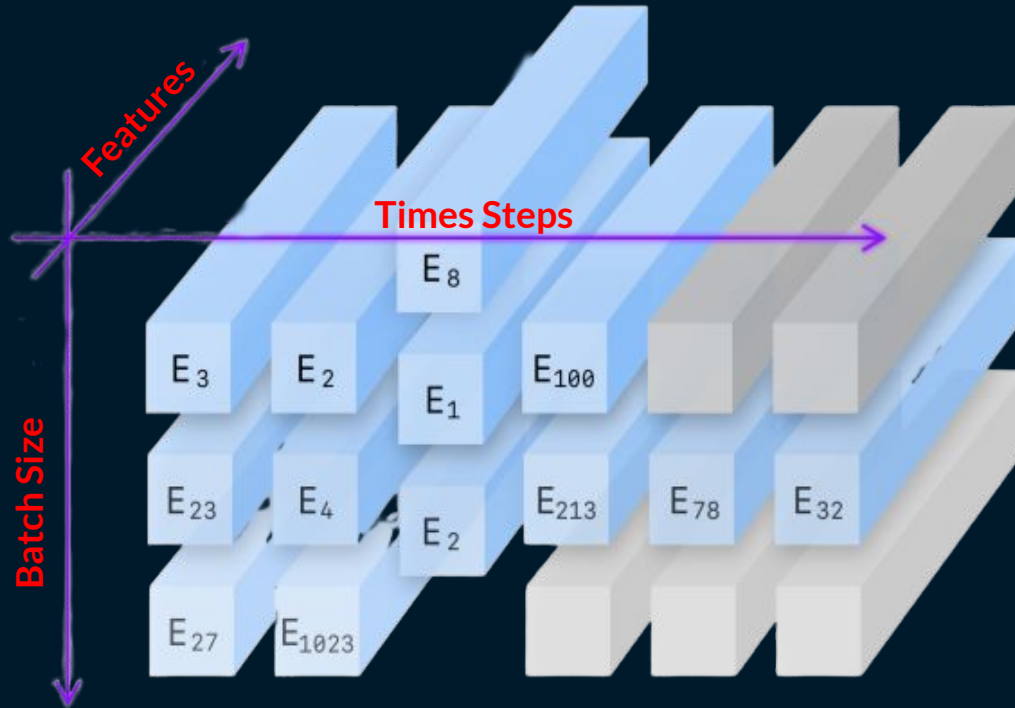
## This Man is Walking or Dancing ?



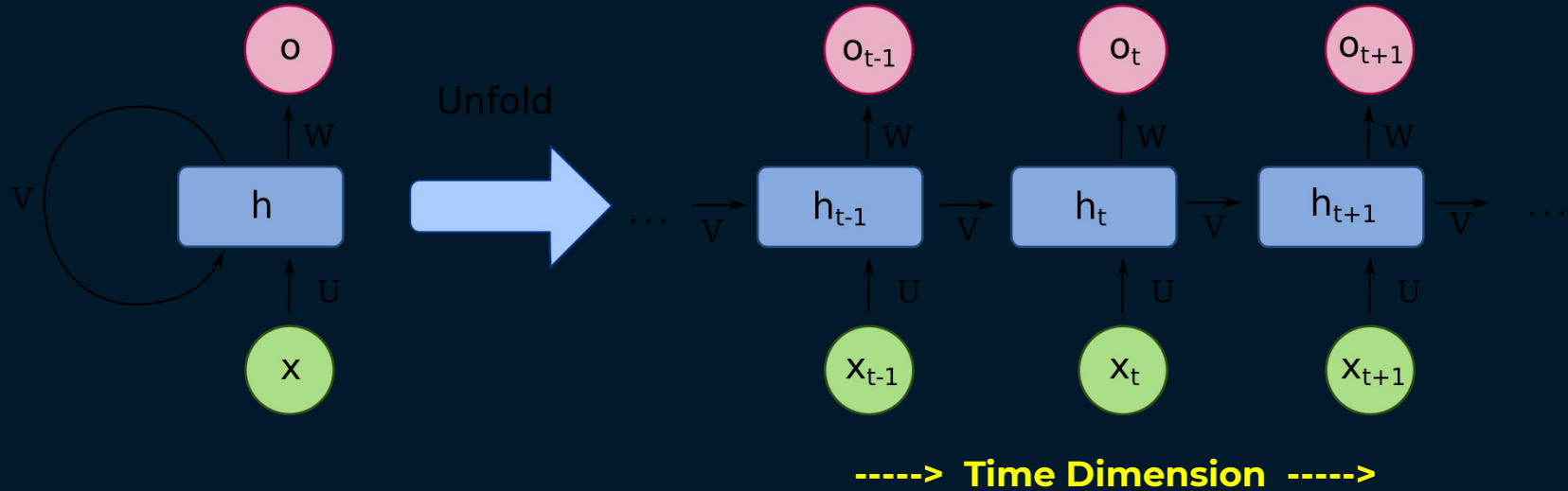
PROFILE WALKCYCLE

T1 T2 T3 T4 T5 T6 T7 T8 T9 T10 T11 ..... Tn

What are the dimensions of the previous example ?



# Recurrent Neural Networks





# When the memory element get updated ?

At  $t=1$

- $\text{input}(t=1) + \text{memory}(t=0)$
- $\text{memory}(t=1) = \text{output}(t=1)$
- Keep memory( $t=1$ )

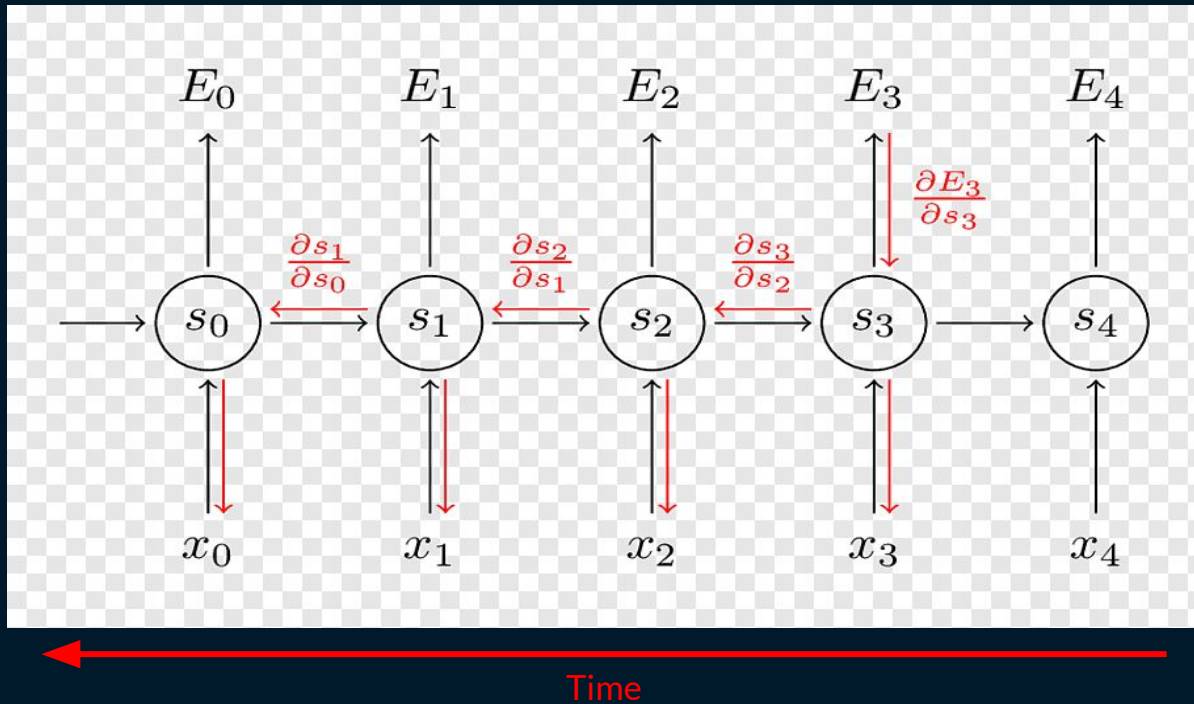
Memory element == hidden state

At  $t=2$

- $\text{input}(t=2) + \text{memory}(t=1)$
- $\text{memory}(t=2) = \text{output}(t=2)$
- Keep memory ( $t=2$ )

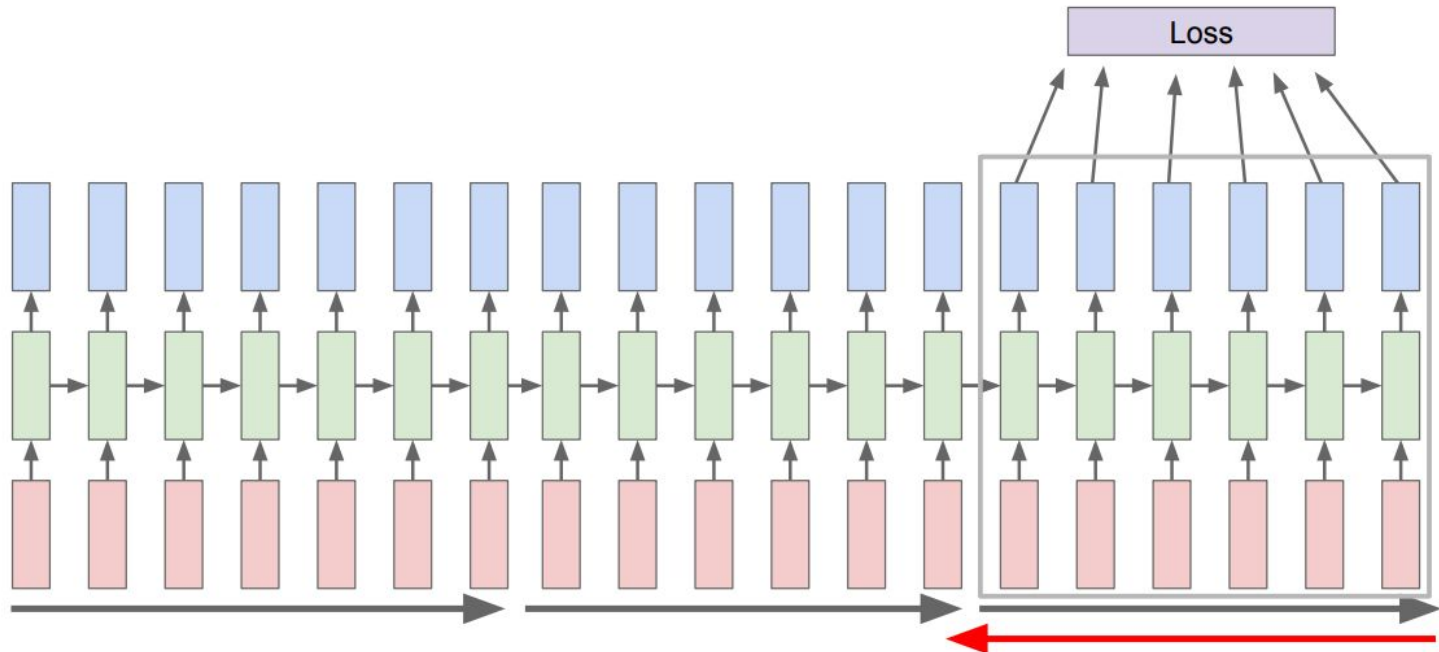
# How to teach the network ?

## Backpropagation Through Time



# Truncated Backpropagation through time

## Truncated Backpropagation through time







## General Notes :-

- 1- All Samples within one batch are handled in PARALLEL (Vectorization)
- 2- All Time-Steps within each sample are handled SEQUENTIALLY
- 3- The memory keeps only the knowledge from each independent sample

→ Is it always the case???

# Stateful vs Stateless RNN

Stateless : Memory resets after each sample

Stateful : the last state for each sample at index  $i$  in a batch will be used as initial state for the sample of index  $i$  in the following batch. (Keras definition)

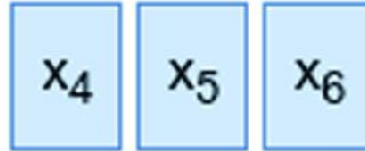
stateful = False

Memory Cell = [0,0,0]

batch 1



Memory Cell = [0,0,0] batch 2



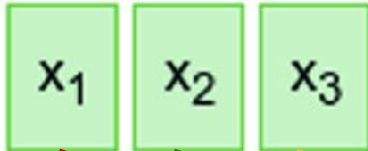
Memory Cell = [0,0,0] batch 3



stateful = True

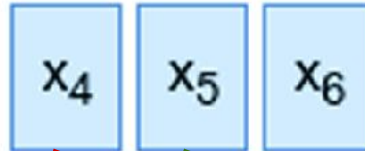
Memory Cell = [0,0,0]

batch 1



Memory Cell = [x1, x2, x3]

batch 2



Memory Cell = [x4, x5, x6]

batch 3





# Vanishing Gradient Problem

Recurrent Neural Networks

# Vanishing Gradient Problem

The **people** who sailed the sea,

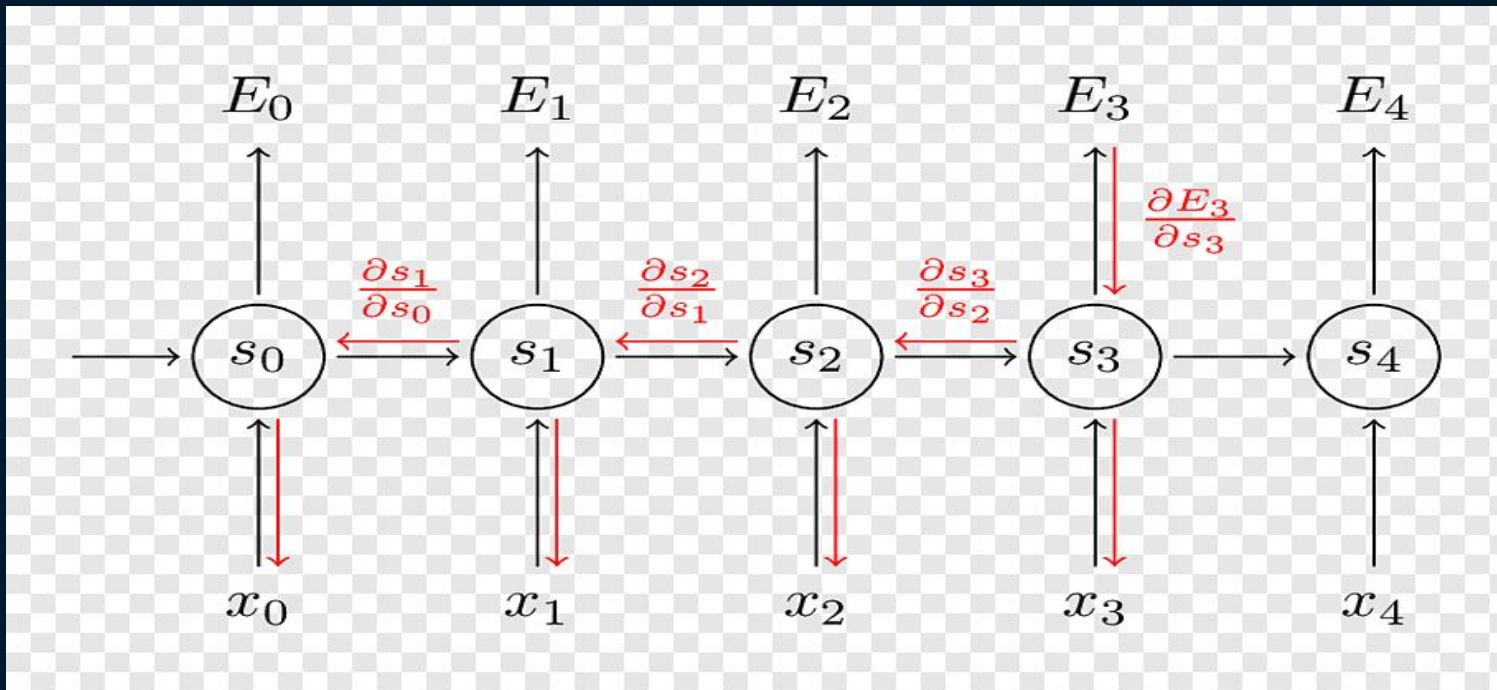
rode the desert,

And hit this **guy**,

**Are** bad !



# Vanishing Gradient Problem





## Vanishing Gradient Problem

- 

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



## Vanishing Gradient Problem

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



## Vanishing Gradient Problem

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



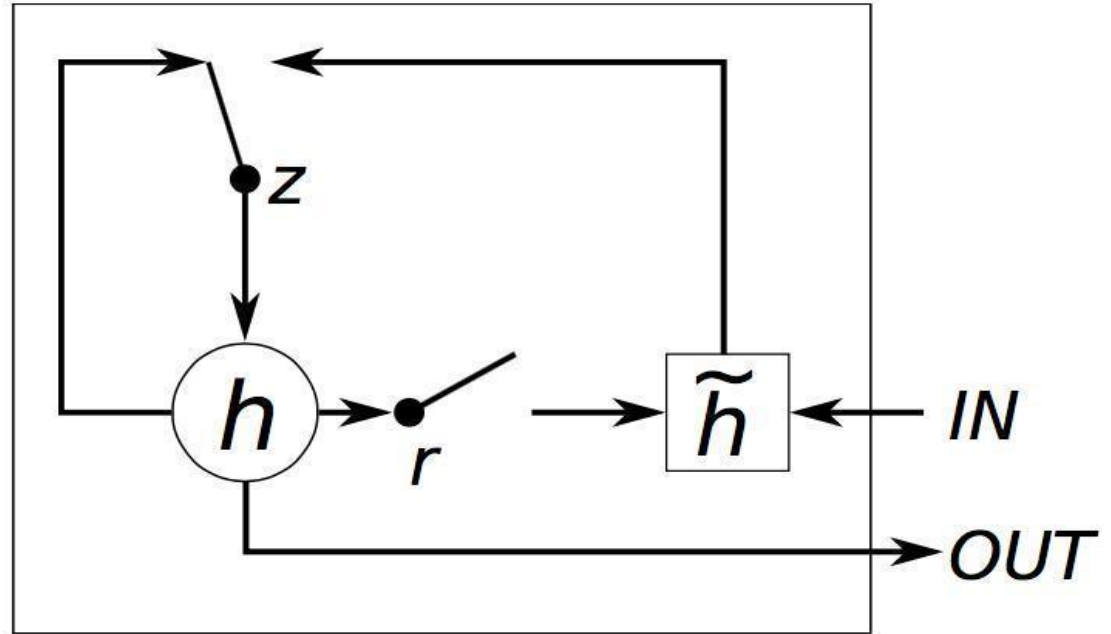
A decorative graphic on the left side of the slide consisting of several overlapping geometric shapes. There is a blue parallelogram, a light green parallelogram, and a dark blue parallelogram, all oriented diagonally. The background is a dark navy blue with subtle diagonal lines.

# Gated Recurrent Units

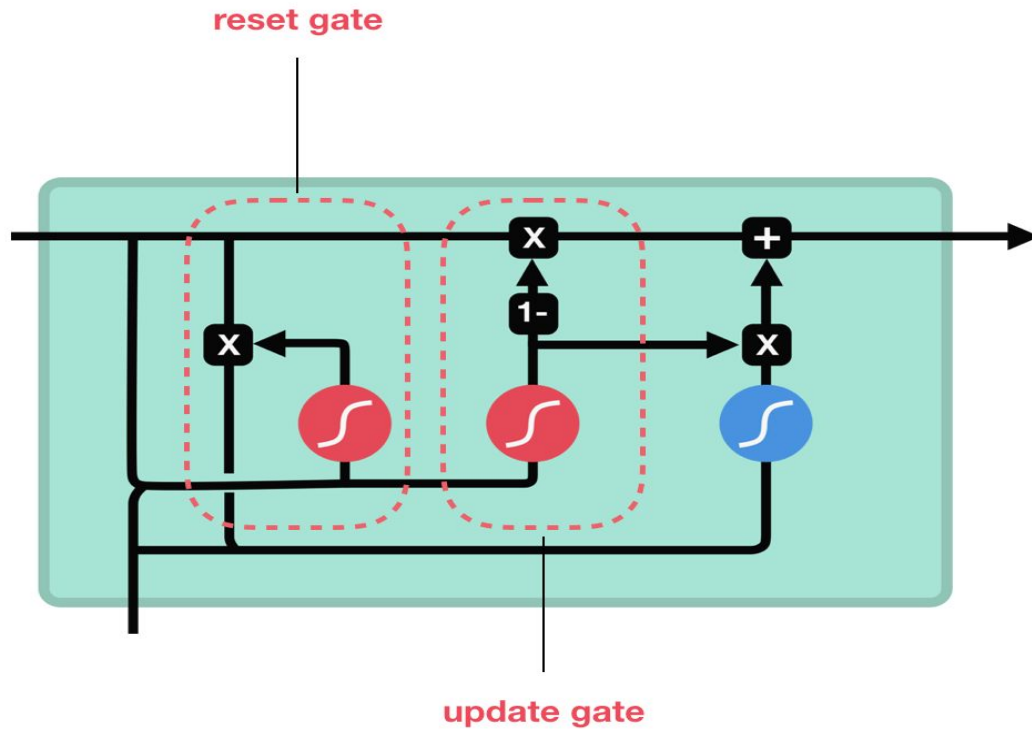
Modification on Vanilla RNN

# Gated Recurrent Units

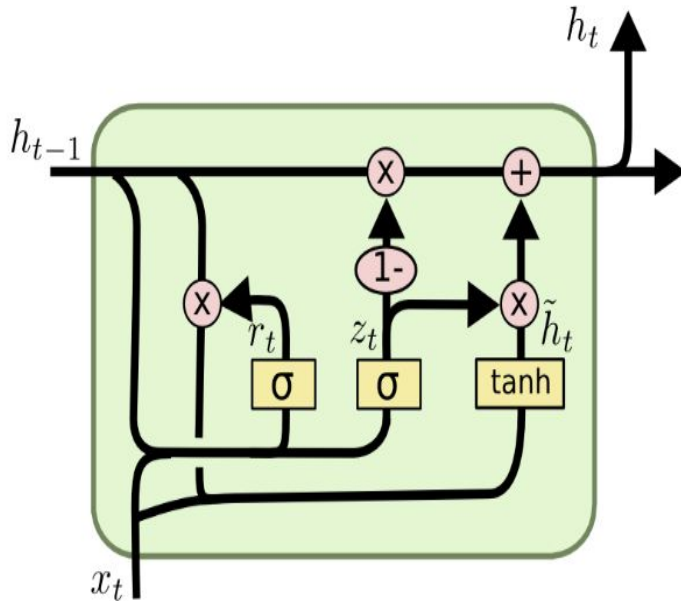
- $R \Rightarrow$  Reset Gate
- $z \Rightarrow$  Update Gate
- $h \Rightarrow$  Previous activations
- $\tilde{h} \Rightarrow$  Candidate value



# Gated Recurrent Units



# Gated Recurrent Units



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

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

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

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



# Gated Recurrent Units


The **people** ,who sailed the sea, rode the desert, And hit this **guy** ,**are** bad!

- $T = 15$  (sequence length)
- To remember the subject, we need to save the hidden state activation
- At  $t = 2$ ,  $r = 1$  (update gate)
- Then  $r = 0$  to prevent updating the hidden state activation corresponding to the index of people in the word output vector
- At  $T = 14$  (are), hopefully the network remembers that the subject people now got its verb.



# Useful Resources

- <https://distill.pub/2019/memorization-in-rnns/>
- <http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/>
- <https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be>
- <https://towardsdatascience.com/gate-recurrent-units-explained-using-matrices-part-1-3c781469fc18>



“Humans don’t start their thinking from scratch every second. As you read this, you understand each word based on your understanding of previous words. You don’t throw everything away and start thinking from scratch again. Your thoughts have persistence.”

Source: [Understanding LSTM Networks](#)

# LSTM:

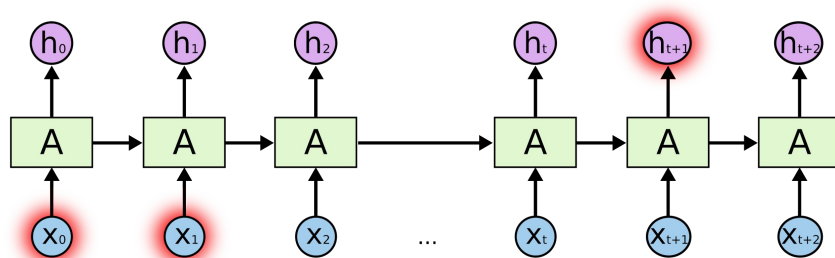
## The problem of long term dependencies

Notice the difference between those two sentences:

The **Clouds** are in the

I grew up in **France**... I speak fluent

Unfortunately, as that gap grows, RNNs become unable to learn to connect the information. LSTMs don't have this problem!







## LSTM:

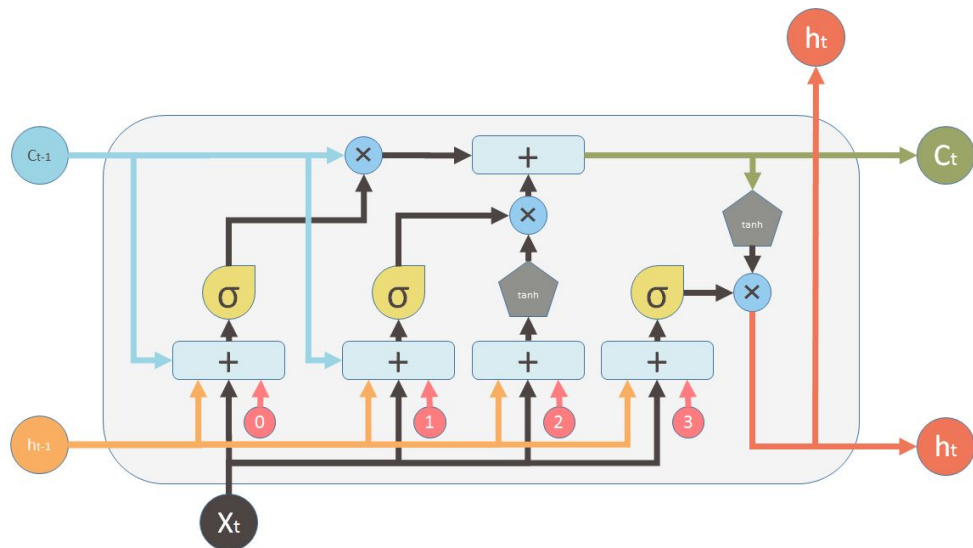
The problem of long term dependencies

Layers that get a **small gradient update** stops learning. Those are usually the **earlier layers**. So because these layers don't learn, RNN's can forget what it seen in longer sequences, thus having a **short-term memory**.

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!, overcome the problem of **vanishing gradient**.

# LSTM:

## Diagram



Inputs:



Input vector



Memory from previous block



Output of previous block

outputs:



Memory from current block



Output of current block

Nonlinearities:



Sigmoid



Hyperbolic tangent

Bias:



Vector operations:



Element-wise multiplication

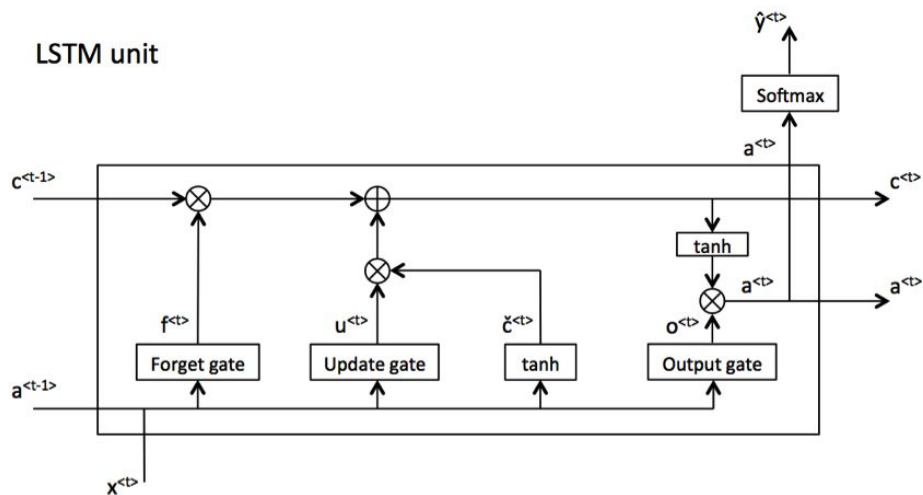


Element-wise Summation / Concatenation

# LSTM:

## Unit

LSTM unit



$$\check{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$

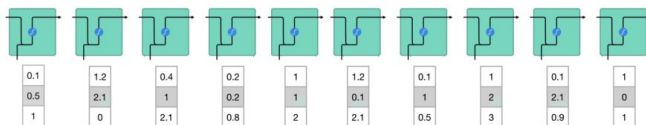
$$\text{update gate: } \Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$

$$\text{forget gate: } \Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$$

$$\text{output gate: } \Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$$

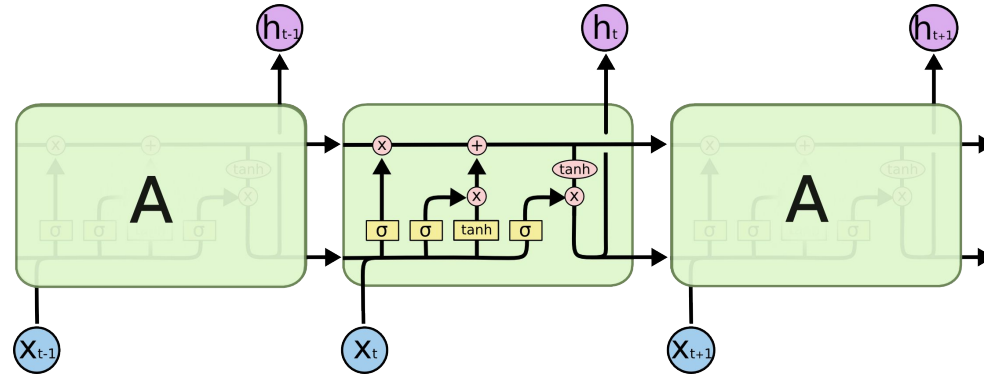
$$c^{<t>} = \Gamma_u * \check{c}^{<t>} + \Gamma_f * c^{<t-1>}$$

$$a^{<t>} = \Gamma_o * \tanh(c^{<t>})$$



# LSTM:

## The Repeating Module (Chain)

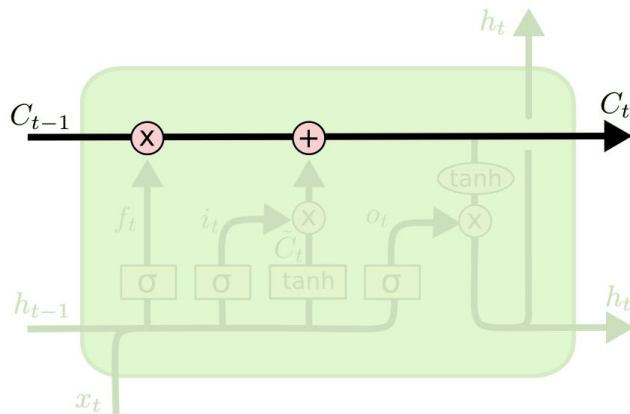


# LSTM:

## Concept

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

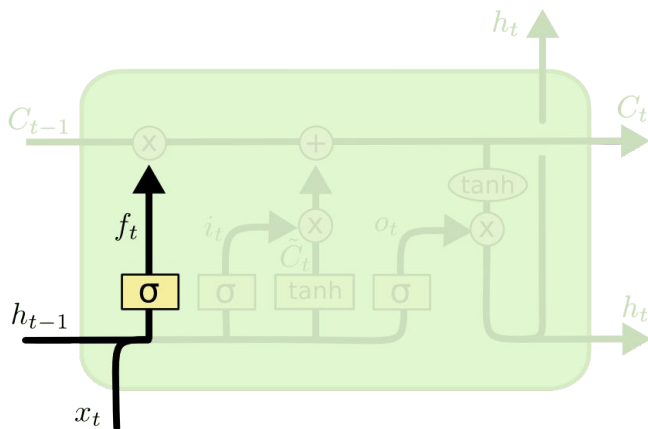
Gates are a way to optionally let **information** through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.



# LSTM:

## How it works

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.”

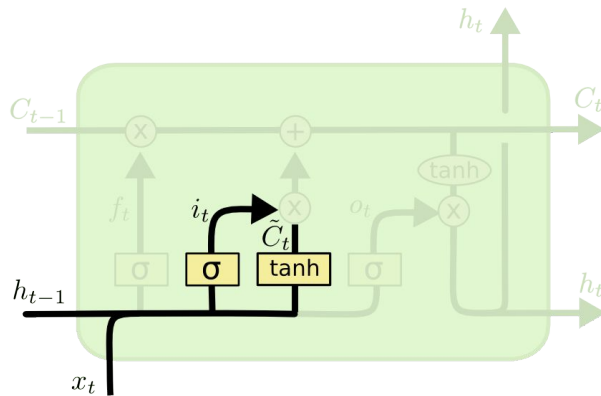


$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

# LSTM:

## How it works

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 “**update gate** layer” decides which values we'll update. Next, a **tanh layer creates a vector of new candidate values**, that could be added to the state. In the next step, we'll combine these two to create an update to the state.



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

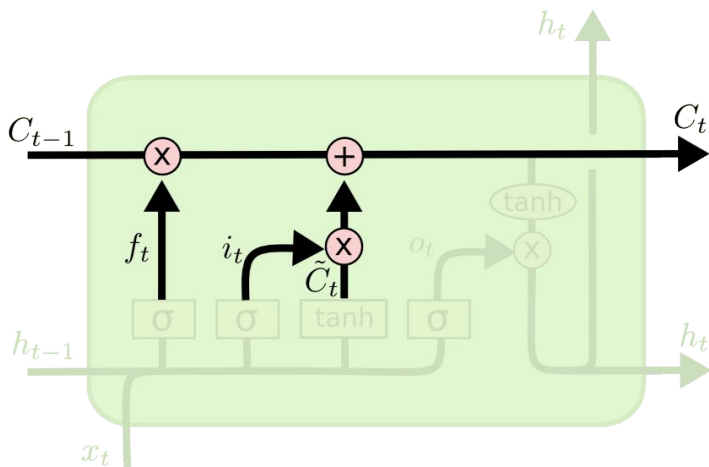
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

# LSTM:

## How it works

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.



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

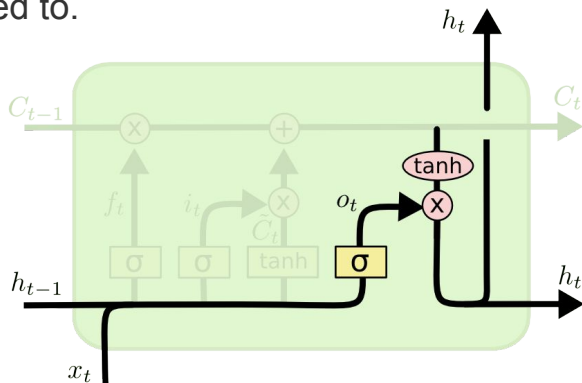


# LSTM:

## How it works

Finally, we need to decide what we're going to output.

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.



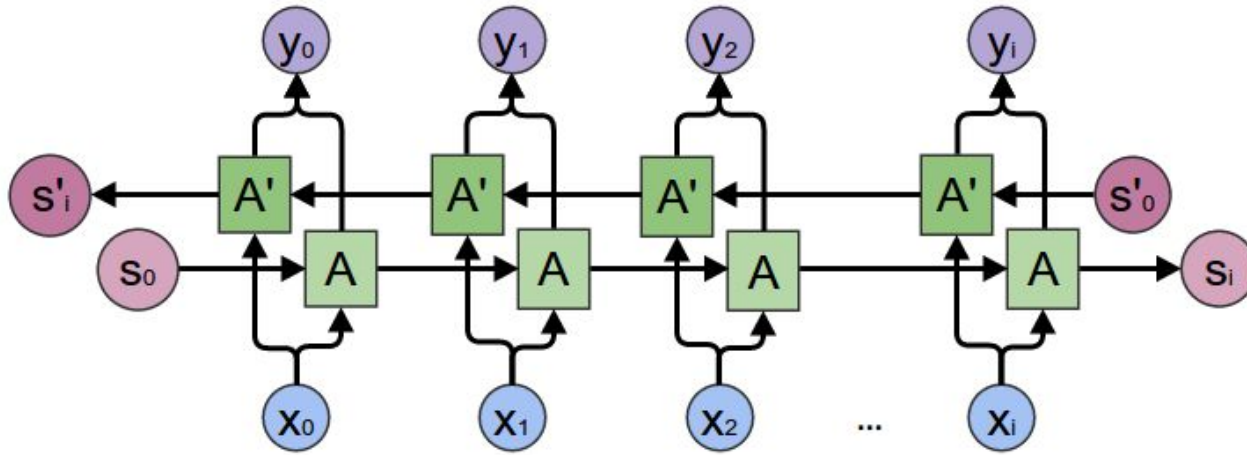
$$o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$$

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

# BRNN:

## Bidirectional Recurrent Neural Network

In a bidirectional RNN, we consider **2 separate sequences**. One from right to left and the other in the reverse order.

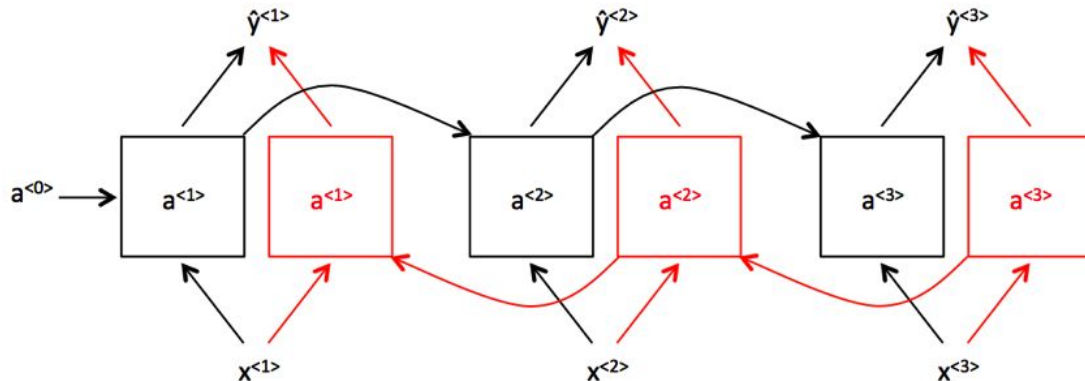


# BRNN:

## Bidirectional Recurrent Neural Networks

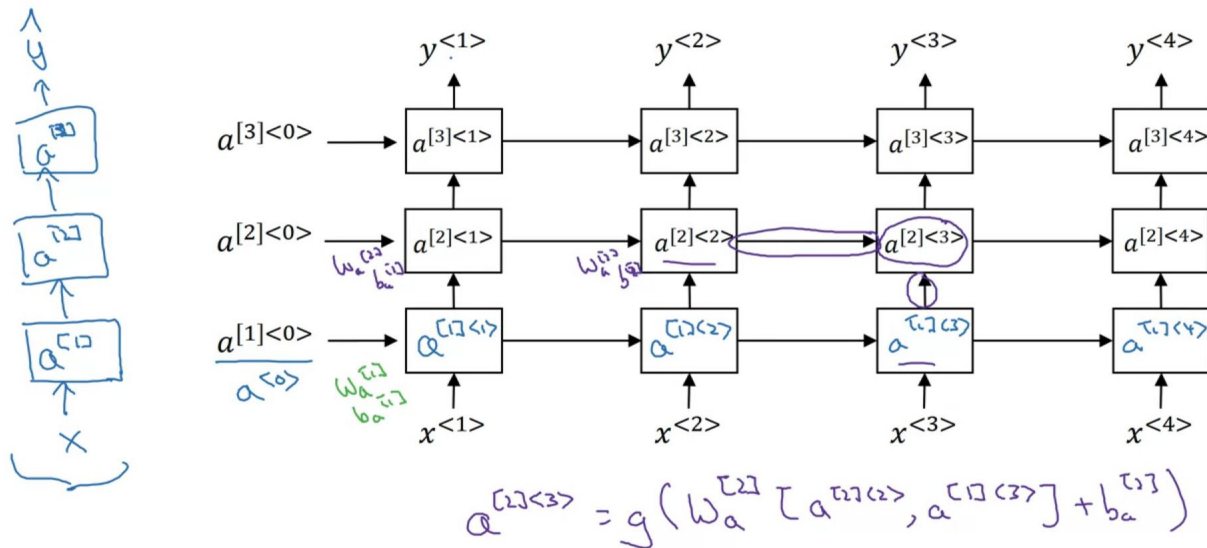
These networks take not only information from earlier in the sequence, but also from later on. They are basically like a patient listener. They “**listen**” to the **whole sentence before making a prediction**.

Using bidirectional will run inputs in two ways, one from past to future and one from future to past



# Deep RNN

## Deep RNN example





# Deep RNN

It adds **extra nonlinearity** to the gating mechanisms.

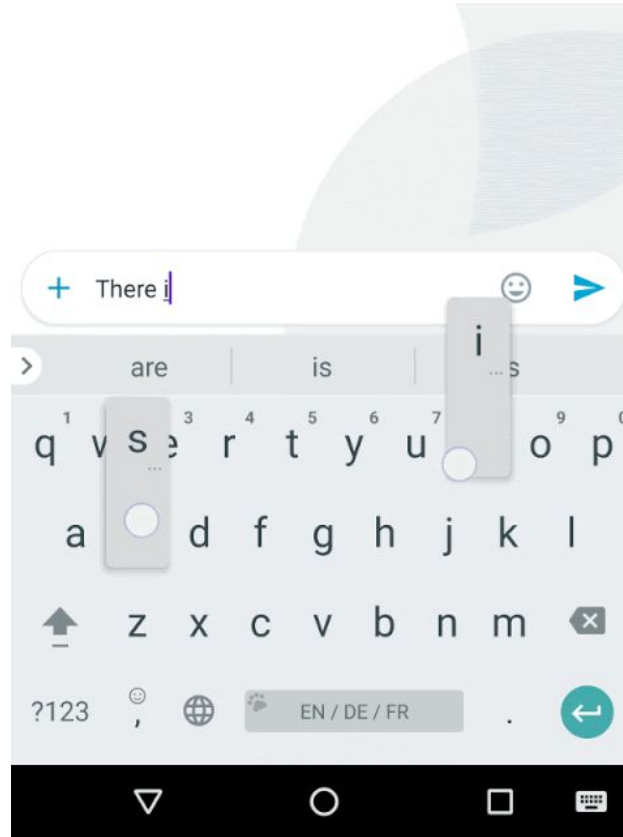
It makes the model **more sophisticated**.

We could stack multiple layers of RNN units on top of each other. This results in a mechanism that is more flexible, due to the combination of several simple layers.

In particular, data might be relevant at different levels of the stack. For instance, we might want to **keep high-level data about financial market conditions (bear or bull market) available, whereas at a lower level we only record shorter-term temporal dynamics**.

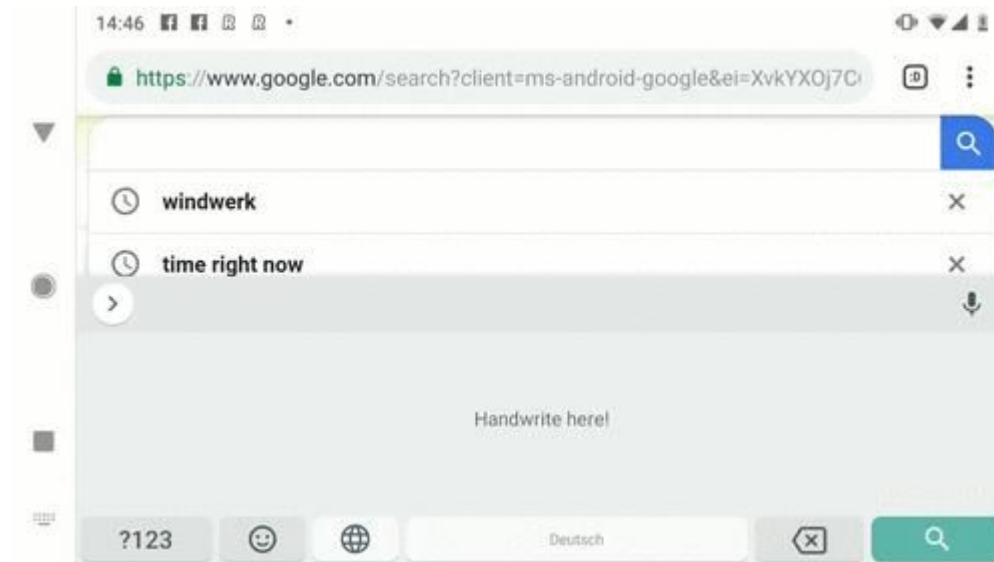
# Some Popular Applications

Trilingual input typing in **Gboard**



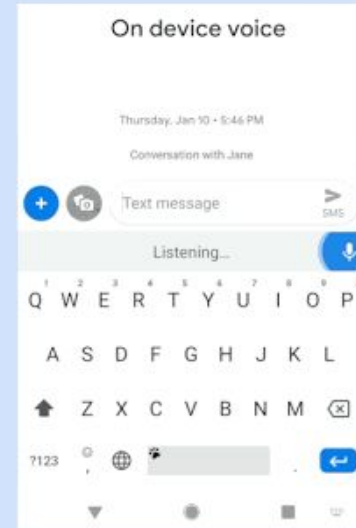
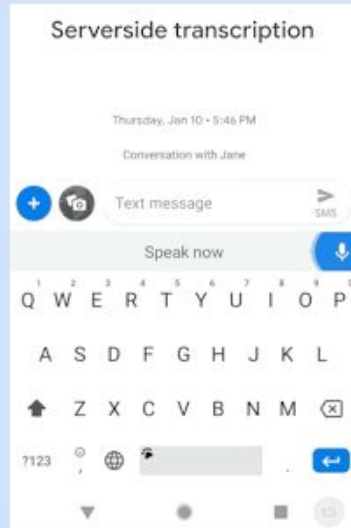
# Some Popular Applications

Fast Multi-language LSTM-based Online Handwriting Recognition



# Some Popular Applications

An All-Neural On-Device Speech Recognizer







# References:

[5 Types of LSTM Recurrent Neural Networks and What to Do With Them](#)

[Understanding LSTM Networks](#)

[Illustrated Guide to LSTM's and GRU's: A step by step explanation](#)

[DeepLearning series: Sequence Models - Machine Learning bites](#)

[Understanding Bidirectional RNN in PyTorch](#)

<https://arxiv.org/pdf/1909.09586.pdf>

[What's the difference between a bidirectional LSTM and an LSTM?](#)

[LONG SHORT-TERM MEMORY 1 INTRODUCTION](#)



# References:

[9.3. Deep Recurrent Neural Networks — Dive into Deep Learning 0.7.1 documentation](#)

[The Machine Intelligence Behind Gboard](#)

[RNN-Based Handwriting Recognition in Gboard](#)

[Google voice search: faster and more accurate](#)

[An All-Neural On-Device Speech Recognizer](#)

[Text-to-Speech for Low-Resource Languages \(Episode 2\): Building a Parametric Voice](#)