ecnológico
Monterrey

NLP 2

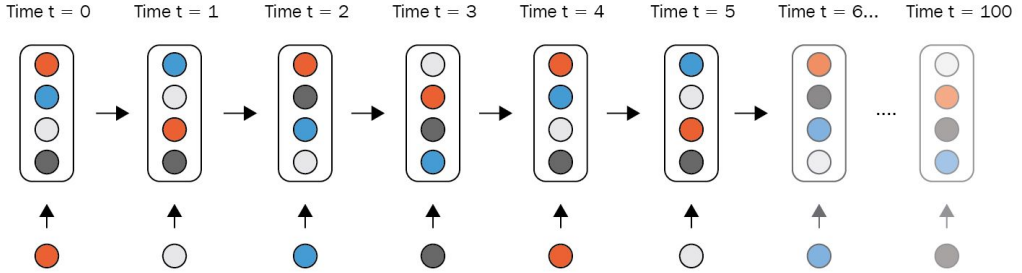
Inteligencia artificial avanzada para la ciencia de datos II Modulo 5 NLP 2

LSTM

Vanishing gradient

As we discuss in RNN, **vanishing problem** is an effect that happens because of hidden state and recursion.

Hidden state is also called the “memory component”, if sequence is too long then memory vanish or gets lost.

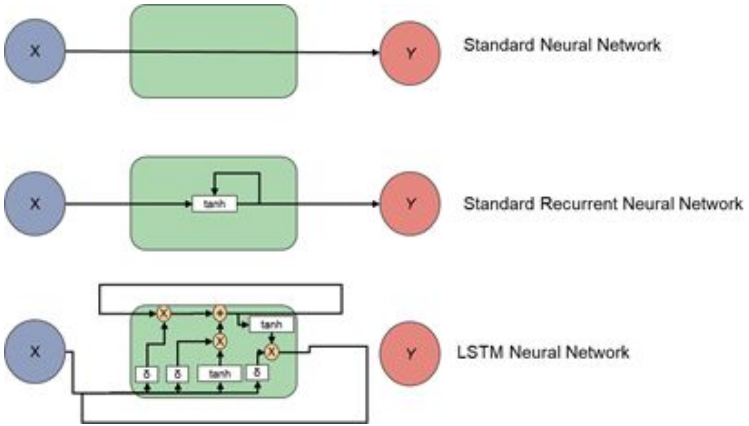


Long-short term memory

LSTM is a RNN network subtype, de architecture of the network includes several changes but if we ignore what happens inside the Node or “cell”, the only difference with traditional or “vanilla” RNN is the Cell state.

Cell state is a variable that travels in each iteration and is not suitable of vanishing problem, Cell state is the **long memory** component that avoids vanishing.

Cell state is independent of “at” or hidden state of the vanilla RNN model and it is also available here, the is called **short memory.**



Forget / Refresh

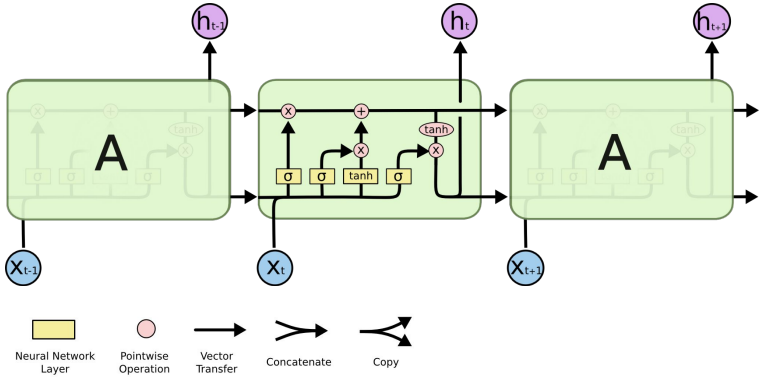
The “cell state” have two main changes during each event in the sequence:

1. The first one is to **forget** (multiply between 0 and 1)
2. The second one is to **learn/refresh** (multiply between -1 and 1)

Inside the cell

Inside the cell **long memory (**c state) and **short memory** (a state) interacts by using 3 phases or gates.
First of all a-t state and xt input merge to bring “new vs previous context”

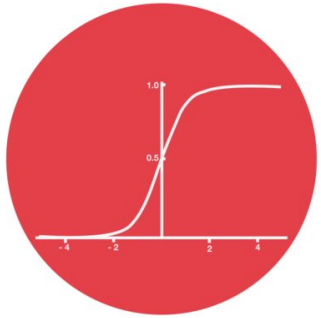
1. **Forget gate.-** given at-1 ,xt and wf (weights) decides how much to forget from ct -1
2. **Update gate .-** given at-1 ,xt and wi (weights) decides how much to update (learn of update) in ct -1
3. **Output gate .-** given at-1 ,xt ,new ct and wo (weights) decides the next at+1 value



Activation functions

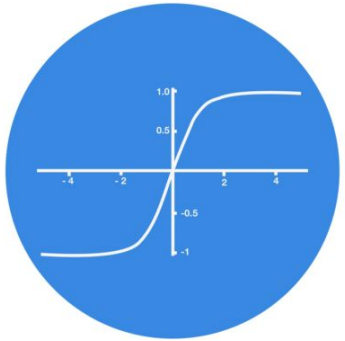
Sigmoid.- we have used sigmoid for classification models but in LSTM is used as a vanishing (forget) function, if the sigmoid function equals 1 means nothing to forget, if 0 means forget everything.

5
0.1
-0.5



Tanh.- In RNN we discuss the problem of fading (keep big values or small values), tanh function keeps values between -1 to 1. This helps to avoid vanishing and exploding gradient problem by increasing or decreasing values through the network.

5
0.1
-0.5



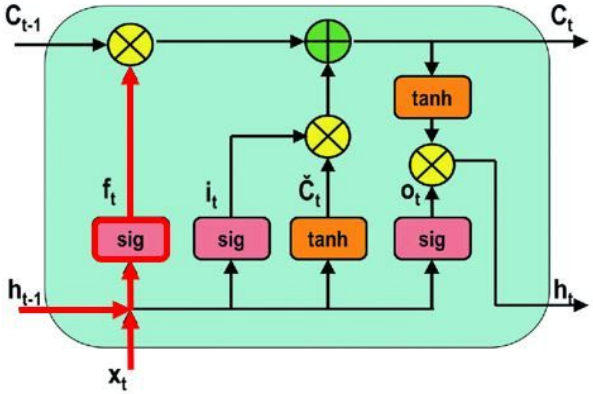
Forget gate

In the first step of the cell gates, Ct-1 contains the information from **previous sequence events** in one vector.

By having a dot product resulting vector from xt and ht-1 shor memory state, the Wf matrix and a bias decides how **much about this information should by deleted from long memory**.

Then by using the sigmoid function we decide how much form the vector **will be used to forget**.

Finally by multiplying the Ct-1 state vector and ft the long memory is updated to forget and a **new CT value is born**.



Forget Gate Operation

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

$t = timestep$

$f_t = forget\ gate\ at\ t$

$x_t = input$

$h_{t-1} = Previous\ hidden\ state$

$W_f = Weight\ matrix\ between\ forget\ gate\ and\ input\ gate$

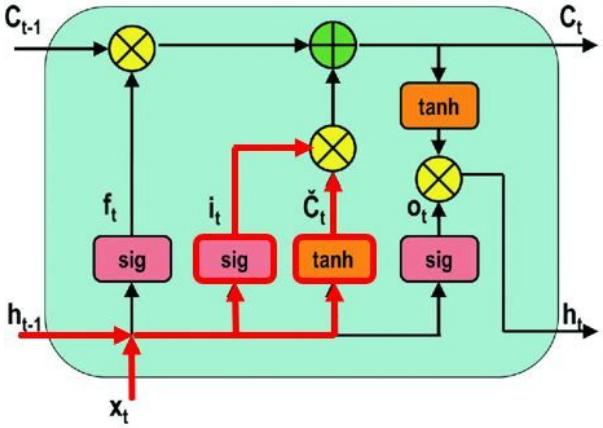
$b_t = connection\ bias\ at\ t$

Update gate

This is probably the most complex gate, first a it vector is produced using a Wi vector dot product with ht-1 and xt (Note ht-1 and xt didn't change after forget gate) **this vector represents what information is valuable to include or update** 1 means and 0 means nothing important.

Simultaneously the same process is performed by using a Wc vector and a tanh function, this resulting vector indicates **how the long memory will change.** Both vector multiply and new update is ready.

Finally notice that this update **vector will be an adding operation** (not multiplication) this is why long memory doesn't have vanishing or exploding gradient, cell memory is ready for the new network iteration.



Input Gate Operation

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

t = timestep

i_t = input gate at t

W_i = Weight matrix of sigmoid operator between input gate and output gate

b_t = bias vector at t

$C\sim_t$ = value genrated by tanh

W_c = Weight matrix of tanh operator between cell state information and network output

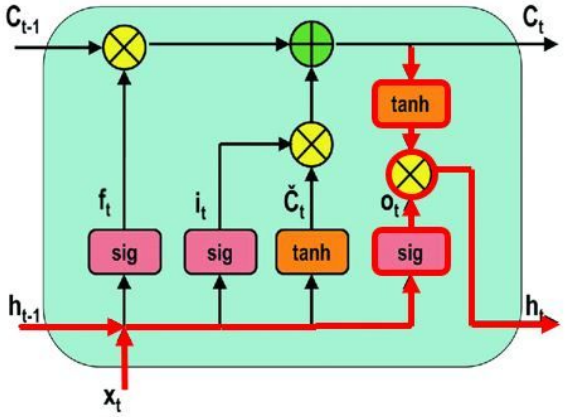
b_c = bias vector at $t, w.r.t\ W_c$

Output gate

In the last step, again the ht-1, xt and an Wo vector will perform a dot product operation. This vector represent **how much of the new information should remain in the short memory.**

Simultaneously the new Ct state (long memory) is transformed so we will include long memory information so **short memory will have the long term information benefit.**

Finally both vector are multiplied and resulting vector is stored in h state so next iteration **short memory is ready.**



Output Gate Operation

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

t = timestep

O_t = output gate at t

W_o = Weight matrix of output gate

b_o = bias vector, w.r.t W_o

h_t = LSTM output

LSTM benefits vs limitations

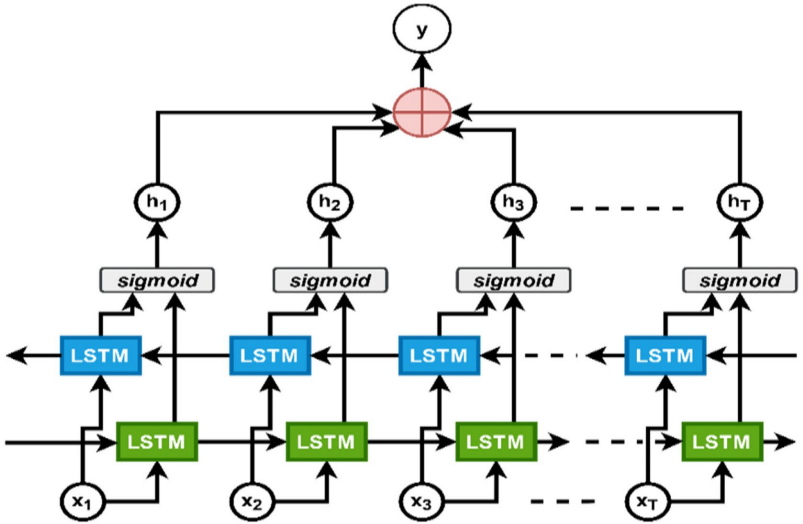
Benefits of LSTM	Limitations of LSTM
1. Long-Term Dependency Handling	1. Computational Complexity: LSTMs are computationally more intensive than simpler models, which can lead to longer training times and higher resource requirements.
2. Vanishing Gradient Mitigation	2. Gating Mechanisms: While the gating mechanisms are a strength, they also make LSTMs more complex and harder to interpret compared to simpler models.
3. Gating Mechanisms: Improved Information Flow Control	3. Data Efficiency: LSTMs require a large amount of training data to perform effectively, and they may struggle with smaller datasets.
4. Variable Sequence Length Handling	4. Difficulty with Parallelization: Although LSTMs can be parallelized, they are not as parallel-friendly as some other architectures like Convolutional Neural Networks (CNNs).
5. State Preservation: Context Maintenance	5. Model Architecture: The architecture of LSTMs can be more difficult to design and tune compared to feedforward neural networks. Proper hyperparameter selection is crucial.

BI-LSTM

One inconvenient until this point is the future prediction, we only focus on **giving past context to predict future.**

Sometimes it is desirable to give future context to predict what is missing in a more efficient way.

BI - LSTM is a easy implementation to solve this task, simply uses **two traditional LSTM models (forward and reverse)** and at the end the output of the two models is calculated by getting the **mean value.**

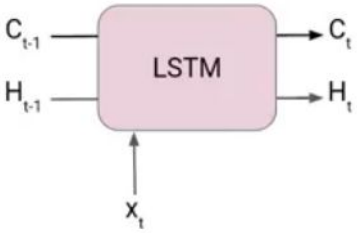


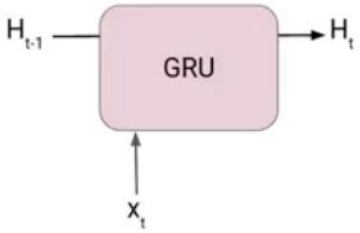
Gated Recurrent Unit (GRU)

In 2014 an optimization of LSTM was proposed, this model simplifies the gate concept inside the cell.

As a result GRU was adopted for simplicity architecture and also because in practical evidence brings long memory as efficiently as LSTM

Also GRU involves less weight (Tensors) operations so it is so much fast to train than LSTM.

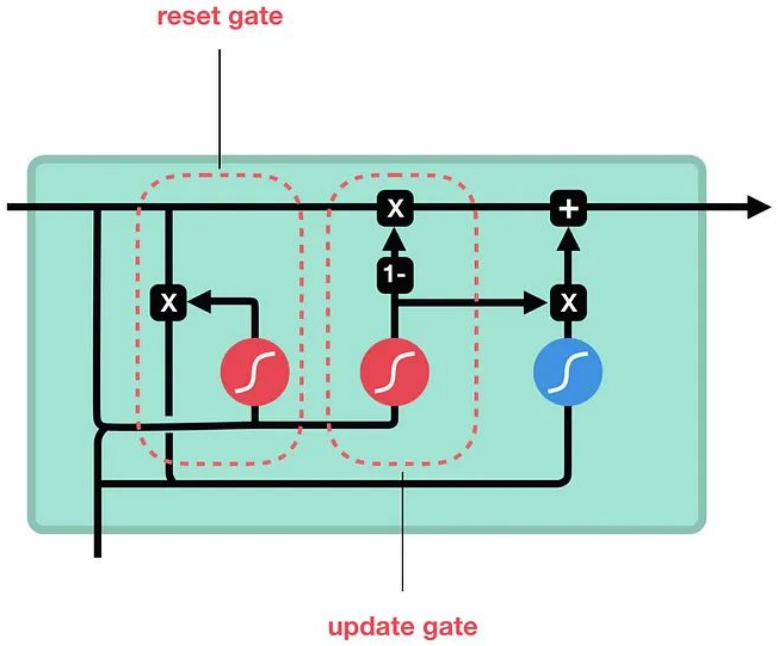




Improvement

GRU simplifies LSTM by removing one control gate, now there are only a reset gate which works in a similar way to the forget gate but now it only needs to decide w**hat information needs to be deleted in memory**.

The update gate works in two phases , a phase that decides what **new information needs to be updated** and finally includes a learning component to avoid vanishing.



Thanks

Do you have any questions?

emmanuel.paez@tec.mx
Slack #module-5-nlp-1