RNN - Recurrent neural networks

- Traditional neural networks struggle with maintaining context over time.
- **Problem:** Classifying actions in a movie frame by frame is challenging because traditional networks do not retain information from previous frames.
- Recurrent Neural Networks (**RNNs**) address this issue by incorporating memory, allowing past information to influence future decisions.

Usage examples

1. Natural Language Processing (NLP)

- **Text generation** (e.g., chatbots, story generation)
- **Speech recognition** (e.g., voice assistants like Siri, Alexa)
- **Machine translation** (e.g., Google Translate)
- **Sentiment analysis** (e.g., detecting emotions in reviews)
- **Named entity recognition (NER)** (e.g., identifying names, places in text)
- **Autocorrect and text prediction** (e.g., smartphone keyboards)

2. Time Series Analysis

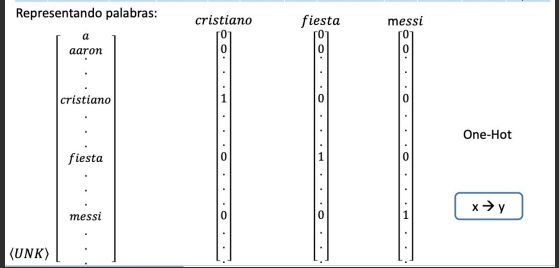
- **Stock price prediction**
- **Weather forecasting**
- **Anomaly detection in IoT and cybersecurity**
- **Demand forecasting for supply chain management**

Ejemplo: NER

Reconocimiento de entidades nombradas (NER)

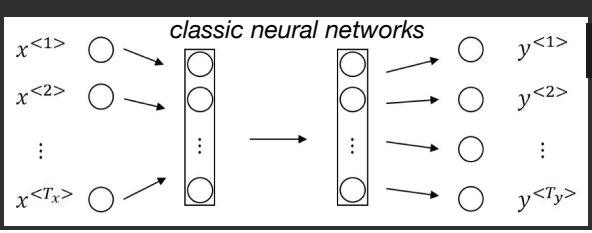
	Cristiano	Ronaldo	Sabotea	La	Fiesta	De	Cumpleaños	De	Lionel	Messi	T_x=10
y <t>	1	1	0	0	0	0	0	0	1	1	T_y=10

Representando palabras:

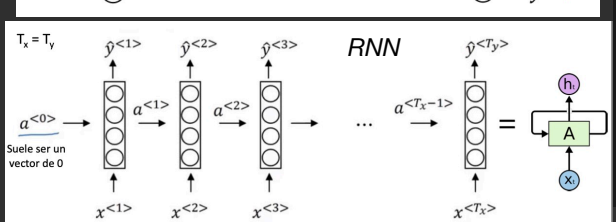


- Words are represented using **one-hot encoding**, meaning each word is transformed into a binary vector where only one position is `1` and all others are `0`.
- For example:
  - "Cristiano" → A vector with `1` at the index corresponding to "Cristiano".
  - "Fiesta" → A vector with `1` at the index corresponding to "Fiesta".
  - "Messi" → A vector with `1` at the index corresponding to "Messi".

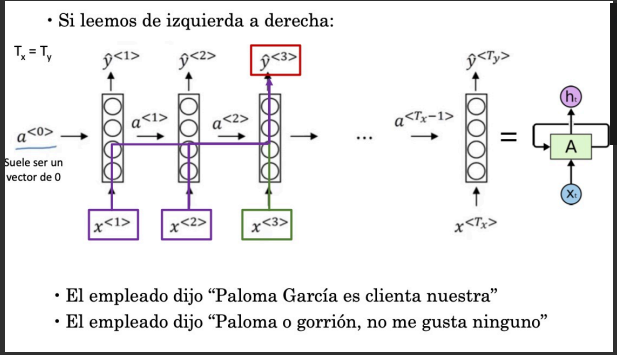
3. **NER Prediction (`x → y` Mapping):**
   - The model learns to classify whether a word is a named entity based on its context in the sentence.
   - Words like "Cristiano", "Ronaldo", and "Lionel", and "Messi" are identified as named entities (`1` in `y<t>`).
   - Other words like "sabotea", "fiesta", and "cumpleaños" are not entities (`0` in `y<t>`).



- However, **there is no connection between past and future words**, meaning the model treats each word **independently**.
- Different text samples have different sentence lengths.
- A classical neural network **cannot handle variable-length inputs and outputs** efficiently.

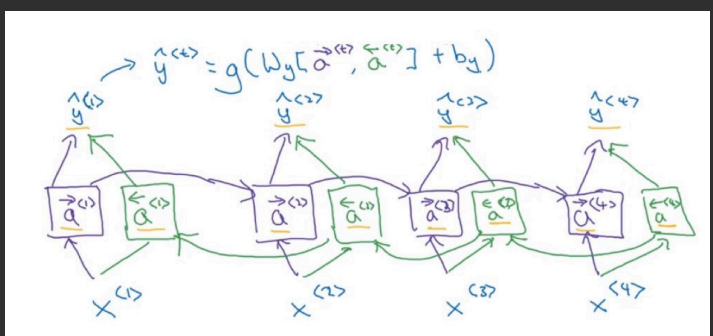


2. **Initialization of the Hidden State (`a<0>`)**
   - The initial hidden state (`a<0>`) is **usually a zero vector**.
   - This serves as the starting point, allowing the network to gradually build up memory as it processes each input `x<t>`.
3. **Processing Each Input Step (`x<t>`)**
   - Each input (`x<t>`) is processed together with the previous hidden state (`a<t-1>`).
   - The network produces an **updated hidden state** (`a<t>`) and an **output prediction** (`ŷ<t>`).
   - This structure allows the model to maintain memory of past words, making it useful for sequential tasks like **Named Entity Recognition (NER)**.

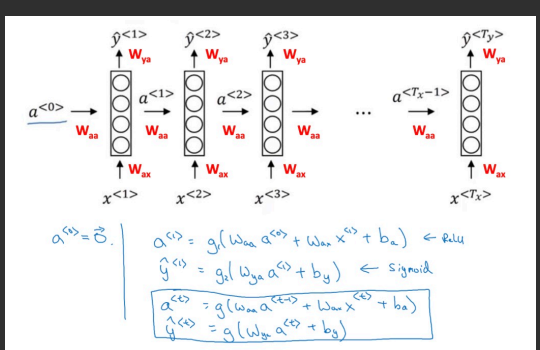


- The two sentences at the bottom illustrate the **importance of context**:
  - **Sentence 1:** "Paloma García es clienta nuestra" (Paloma García is our client) → "Paloma" is likely a **person's name**.
  - **Sentence 2:** "Paloma o gorrión, no me gusta ninguno" (Pigeon or sparrow, I don't like either) → "Paloma" refers to a **bird**.

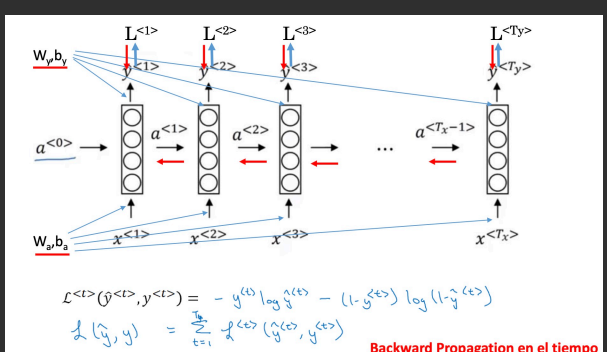
Bidirectional RNN



Forward propagation



Backward propagation



Language model with RNN

1. **What is a Language Model?**
   - A **language model** assigns a probability to sequences of words, helping determine how **natural or likely** a sentence is.

1. **"Elena no quiere salir con Juan."** → Probability: 6×10^{-3}
2. **"El enano quiere salir con Juan."** → Probability: 6×10^{-5}

- The model assigns **higher probability** to the **more common phrase** ("Elena no quiere salir con Juan").
- This suggests that the first sentence is more likely to appear in natural text or speech.

2. **What is Tokenization?**
   - **Breaking a sentence into individual words (tokens).**

En el máster de Data Science aprendemos mucho<EOS>

y<1> y<2> y<3> y<4> y<5> y<6> y<7> y<8> y<9>

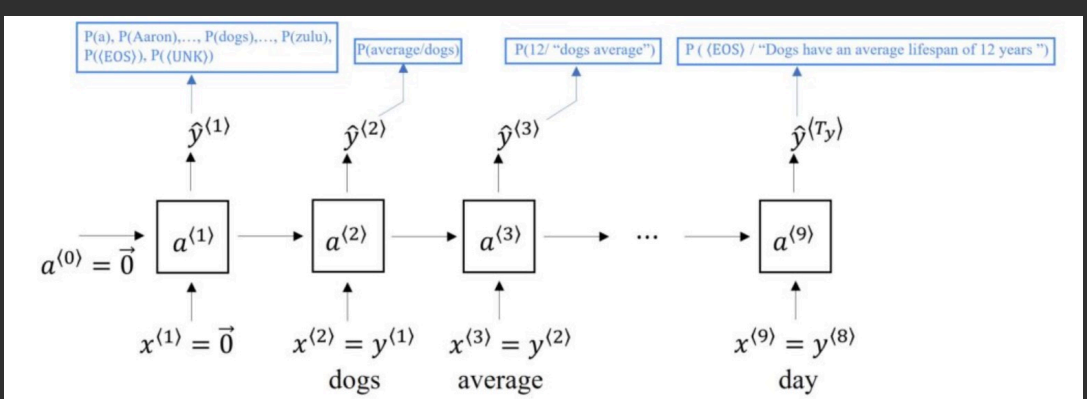
3. **Why Use `<EOS>`?**
   - Helps the model learn **when a sentence ends**, improving **text generation** and **sequence-based tasks**.

if the vocabulary doesn't have a specific word, we use the token <UNK>

Example

"*Dogs have an average lifespan of 12 years<EOS>*"

$x^{<t>} = y^{<t-1>}$



as activation function we use softmax

The loss is:

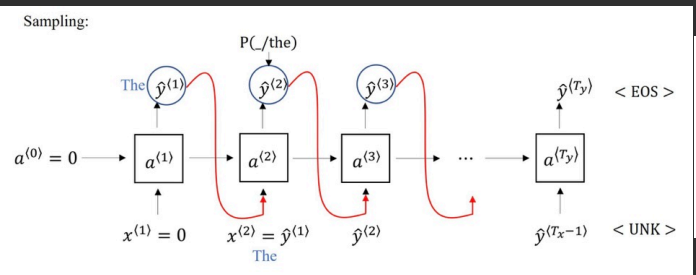
$$\mathcal{L}(\hat{y}^{<t>}, y^{<t>}) = -\sum_i y_i^{<t>} \log \hat{y}_i^{<t>}$$

$$\mathcal{L} = \sum_t \mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>})$$

After training the network we can compute the probability of a sentence: $y^{<1>}\ y^{<2>}\ y^{<3>}$

In this case his probability is: $P(y^{<1>})P(y^{<2>}|y^{<1>})P(y^{<3>}|y^{<1>}y^{<2>})$

Sample new sequences from a RNN



The sampling begins after we have initiated the first input $x^{\langle 1\rangle} = 0$ and set the initial activation value $a^{\langle o\rangle} = 0$. Softmax distribution calculates the probabilities of random samples. The vector formed by all these probability values of each time step is then inputted to a NumPy command, *np.random.choice*, which samples the first words according to these calculated probabilities.

→ P(a)P(Aaron) ... P(zulu)P(< UNK >) np.random.choice

After the first word has been sampled, we move on the second time step which is expecting $y^{\langle 1\rangle} = 0$ as input. However, we pass on the output we sampled above, $\hat{y}^{\langle 1\rangle}$, and pass as input to the second time step. Softmax distribution will, then, make a prediction for the second output $\hat{y}^{\langle 2\rangle}$ based on this input, which is nothing but the sampled output of the previous time step.

Let's say the first word we sample, $\hat{y}^{\langle 1\rangle}$, happens to be "*The*", which is a fairly common choice for the first word in the English language. We pass this "*The*" as the input to the second time step, represented by $x^{\langle 2\rangle}$. Softmax distribution will now determine what is the most probable second word, given that the first word is "The".

In the second time step as well, we shall use the sampling function method to sample the next word. This process keeps continuing until we reach the last time step.

But how would you know when the sequence ends? One solution, as we learned earlier, is to add an EOS (End Of Sentence) token as part of our dictionary. The second solution is that we don't include EOS in our dictionary but set a pre-decided number of time steps.

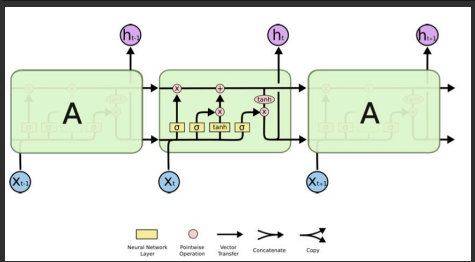
Language model at characters level

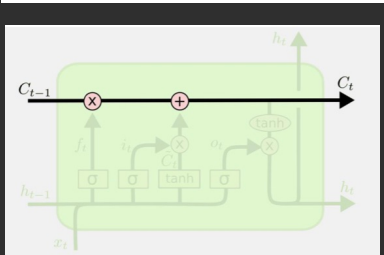
Instead of using vocabulary made up with words, it is made with letter and/or numbers

[a,b,c,d,...., A, B, C, D, ...,0,1,2,3.....]

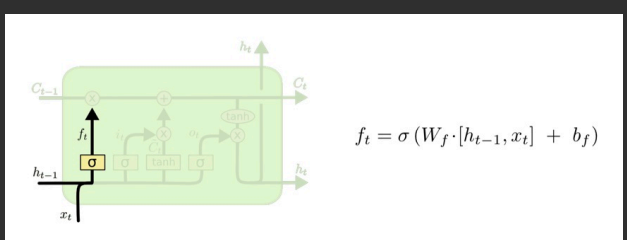
Long Short Term Memory - LSTM

- LSTMs are a special type of **RNN** designed to **retain information over long distances** in a sequence.
- They overcome the **vanishing gradient problem**, which affects standard RNNs when processing long sequences.

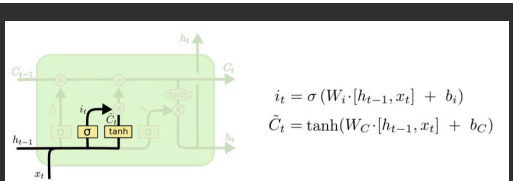




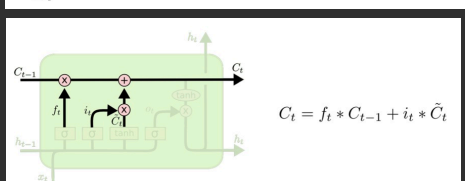
C is the long term memory, it is modified based on the input and the short term memory



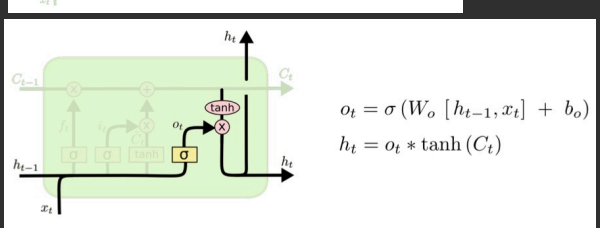
This is the forget gate, it chose which information keep or discard from C. The sigmoid output numbers between 0 and 1 that multiplied with C modify the content of it.



This is the save gate, it chose the new information to save in the long memory. The sigmoid chose the values to be updated, while the tanh chose the new set of values to be added to C



To remove useless memory we multiply the forgot gate output with C, and to add new info we add the information of the save gate.



The value to output are chosen filtering the one in the new long term memory. The same output is also used for the short term memory