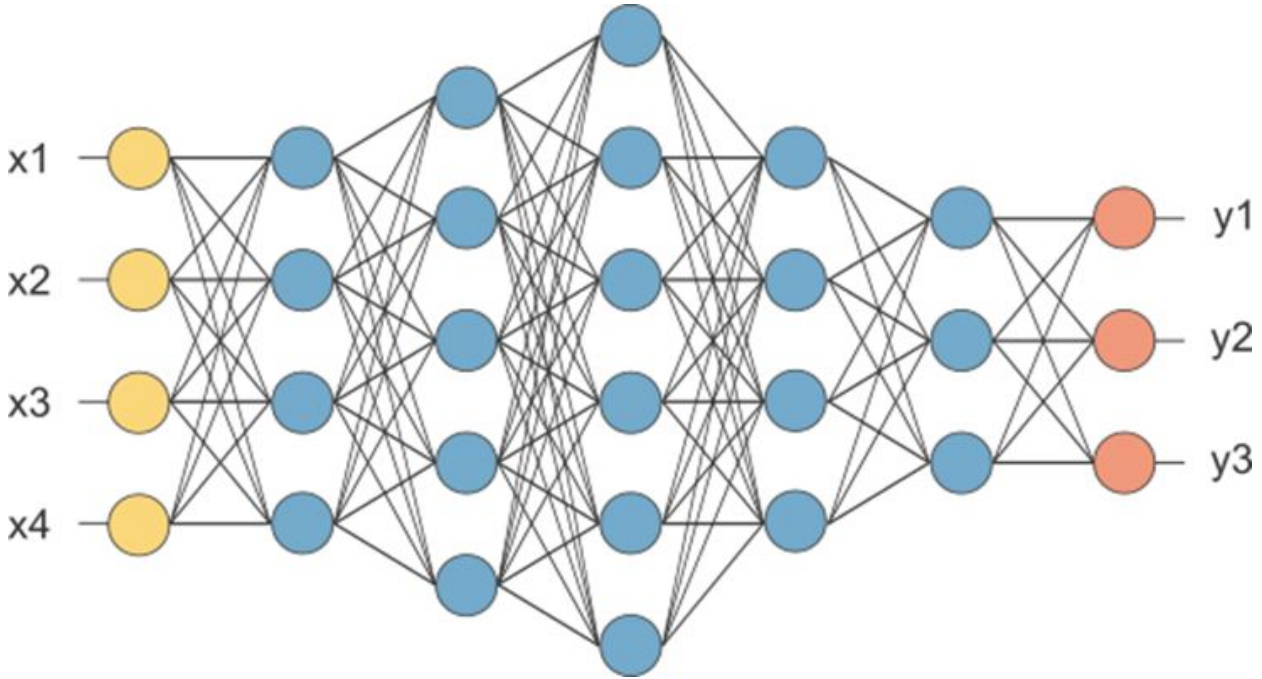
Hour of CODE

Redes Neuronales Recurrentes

Usando Keras

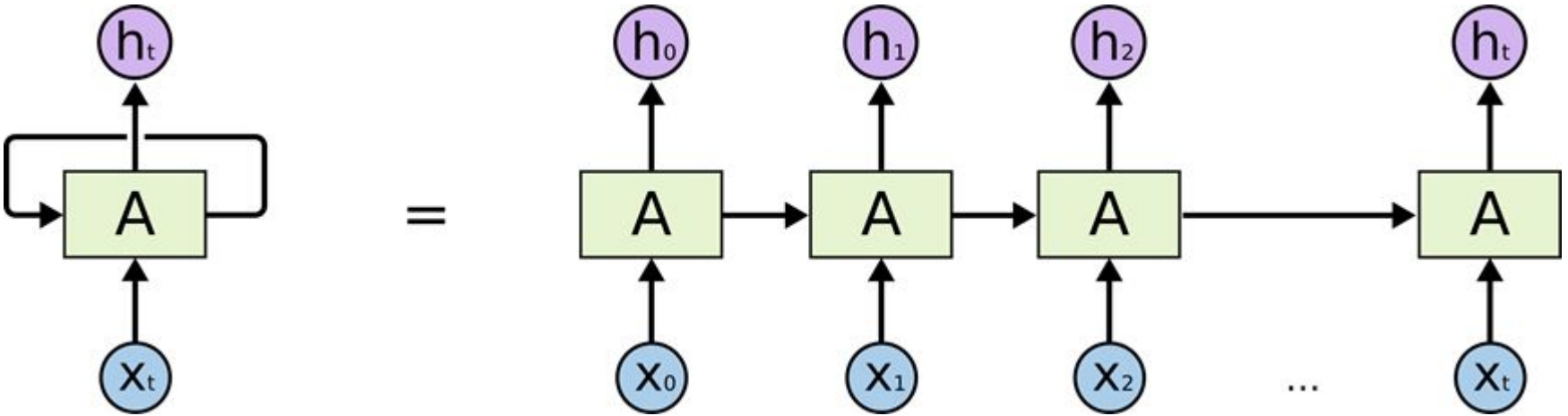
Ángel Moreno Calvo + César Hernández Rodríguez

Red Neuronal



¿Qué es una RNN?

retroalimentaciones entre las neuronas dentro de las capas.



RNN

```
tf.keras.layers.SimpleRNN(
    units,
    activation="tanh",
    use_bias=True,
    kernel_initializer="glorot_uniform",
    recurrent_initializer="orthogonal",
    bias_initializer="zeros",
    kernel_regularizer=None,
    recurrent_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    recurrent_constraint=None,
    bias_constraint=None,
    return_sequences=False,
    return_state=False,
    go_backwards=False,
    **kwargs
)
```

RNN

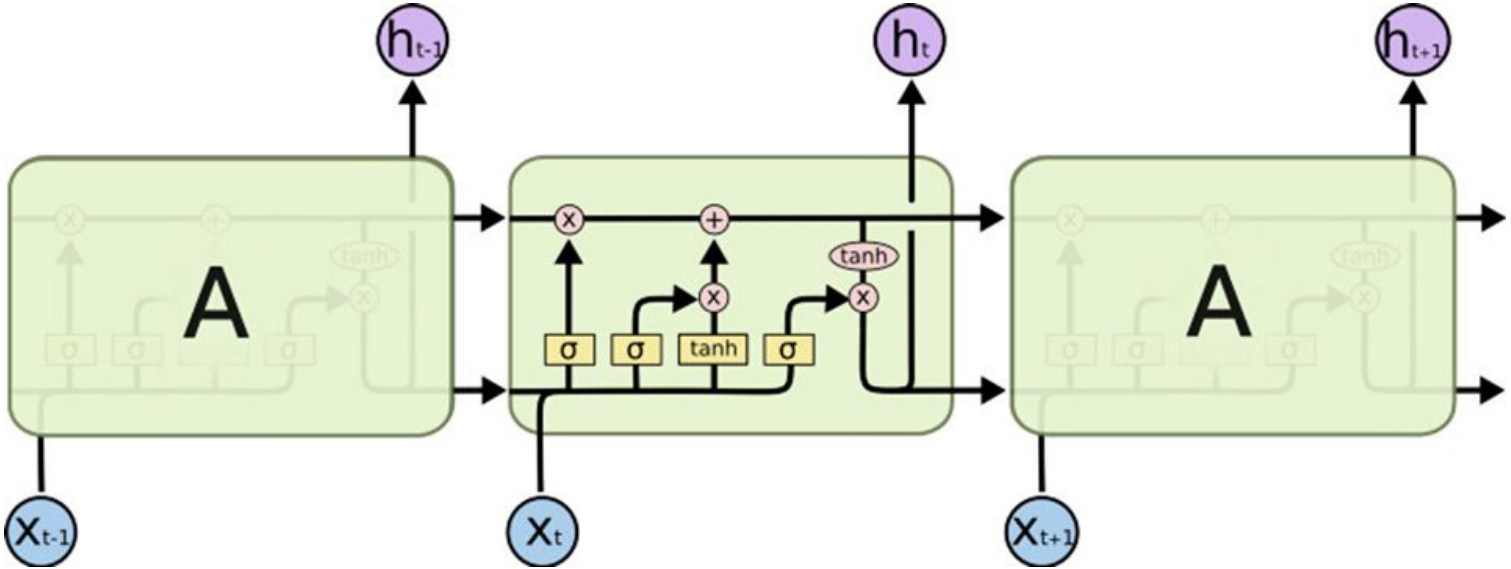
```
inputs = np.random.random([32, 10, 8]).astype(np.float32)
simple_rnn = tf.keras.layers.SimpleRNN(4)

output = simple_rnn(inputs)  # The output has shape `[32, 4]`.

simple_rnn = tf.keras.layers.SimpleRNN(
    4, return_sequences=True, return_state=True)

# whole_sequence_output has shape `[32, 10, 4]`.
# final_state has shape `[32, 4]`.
whole_sequence_output, final_state = simple_rnn(inputs)
```

¿LSTM?



¿LSTM?

Long-Short Term Memory

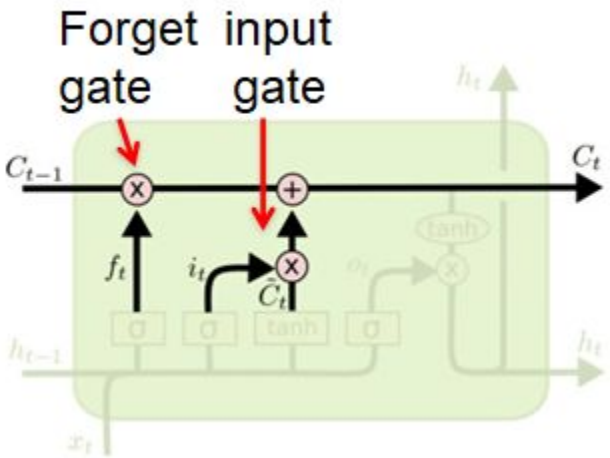
- son una extensión de las redes neuronales recurrentes.
- amplían su memoria para **aprender de experiencias importantes que han pasado hace mucho tiempo**.
- LSTM contiene su información en la memoria, similar a la memoria de un ordenador.
- puede leer, escribir y borrar información de su memoria.

¿LSTM?

Long-Short Term Memory

- Esta memoria se puede ver como una “celda” bloqueada
    - “bloqueada” → almacenar o eliminar información dentro (abriendo la puerta o no para almacenar)
    - en función de la importancia que asigna a la información que está recibiendo.
    - Asignación de importancia en función de los pesos.

¿LSTM?



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

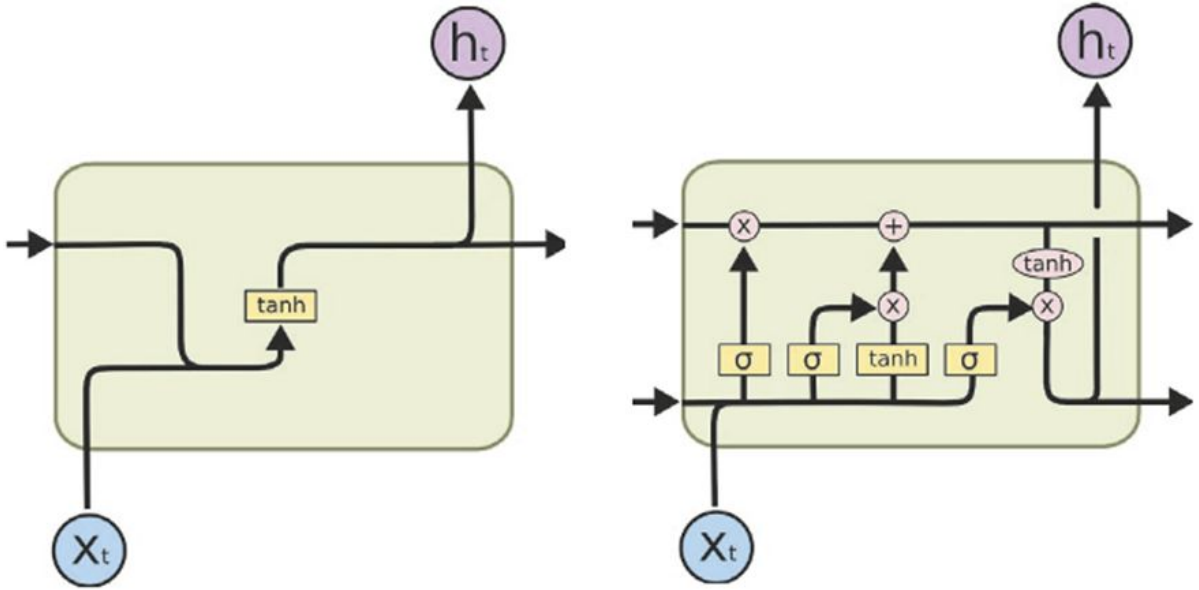
LSTM

```
tf.keras.layers.LSTM(
    units,
    activation="tanh",
    recurrent_activation="sigmoid",
    use_bias=True,
    kernel_initializer="glorot_uniform",
    recurrent_initializer="orthogonal",
    bias_initializer="zeros",
    unit_forget_bias=True,
    kernel_regularizer=None,
    recurrent_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    recurrent_constraint=None,
    bias_constraint=None,
    return_sequences=False,
    return_state=False,
    time_major=False,
    unroll=False,
    **kwargs
)
```

LSTM

```
>>> inputs = tf.random.normal([32, 10, 8])
>>> lstm = tf.keras.layers.LSTM(4)
>>> output = lstm(inputs)
>>> print(output.shape)
(32, 4)
>>> lstm = tf.keras.layers.LSTM(4, return_sequences=True, return_state=True)
>>> whole_seq_output, final_memory_state, final_carry_state = lstm(inputs)
>>> print(whole_seq_output.shape)
(32, 10, 4)
>>> print(final_memory_state.shape)
(32, 4)
>>> print(final_carry_state.shape)
(32, 4)
```

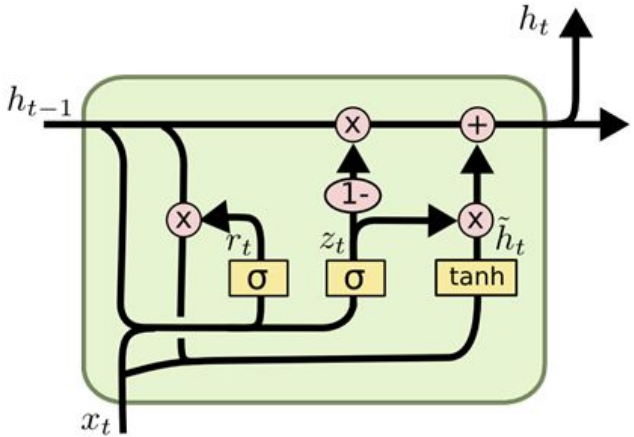
RNN vs LSTM



(a) RNN

(b) LSTM

GRU



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

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

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

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

GRU

Usan el mismo principio que LSTM, pero están simplificadas de manera que su rendimiento está a la par con LSTM pero computacionalmente son más eficiente.

- Combina el forget y el input en una única update gate.
- También fusiona el estado de la celda y el estado oculto. Esto es más simple que LSTM.
- También hay muchas otras variantes.

GRU

```
tf.keras.layers.GRU(
    units,
    activation="tanh",
    recurrent_activation="sigmoid",
    use_bias=True,
    kernel_initializer="glorot_uniform",
    recurrent_initializer="orthogonal",
    bias_initializer="zeros",
    kernel_regularizer=None,
    recurrent_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    recurrent_constraint=None,
    bias_constraint=None,
    return_sequences=False,
    return_state=False,
    time_major=False,
    reset_after=True,
    **kwargs
)
```

GRU

```
>>> inputs = tf.random.normal([32, 10, 8])
>>> gru = tf.keras.layers.GRU(4)
>>> output = gru(inputs)
>>> print(output.shape)
(32, 4)
>>> gru = tf.keras.layers.GRU(4, return_sequences=True, return_state=True)
>>> whole_sequence_output, final_state = gru(inputs)
>>> print(whole_sequence_output.shape)
(32, 10, 4)
>>> print(final_state.shape)
(32, 4)
```

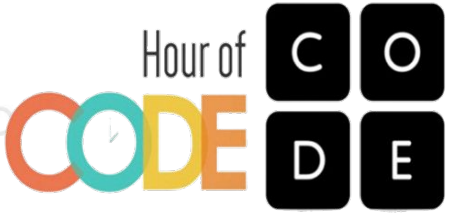
LSTM y GRU

- Los GRU también toman xt y ht-1 como entradas. Realizan algunos cálculos y luego pasan ht.
- GRU no necesitan la capa de celda para transmitir valores.
- Aseguran que los valores de ht retengan una gran cantidad de información antigua.
- Aseguran creación de una gran cantidad de información nueva.

Uso de las RNN

- Generación de texto.
- Generador de musica.
- Imagenes.
- Clasificador de sentimientos.
- Reconocimiento de entrada.

Gracias por asistir



Angel Moreno Calvo
github.com/angelmorenocalvo
@__angelmoreno

Hour of CODE