clusterAI 2020
ciencia de datos en ingeniería industrial
UTN BA
curso I5521

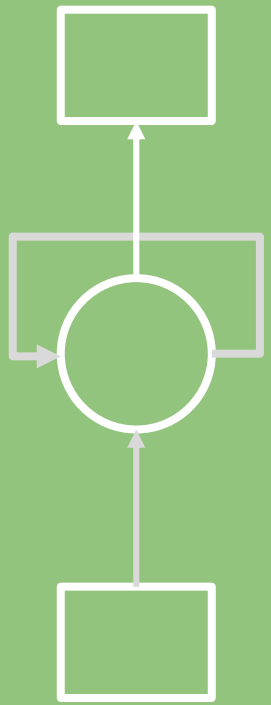
clase_12: Redes Neuronales Recurrentes

Docente: Nicolás Aguirre

Agenda

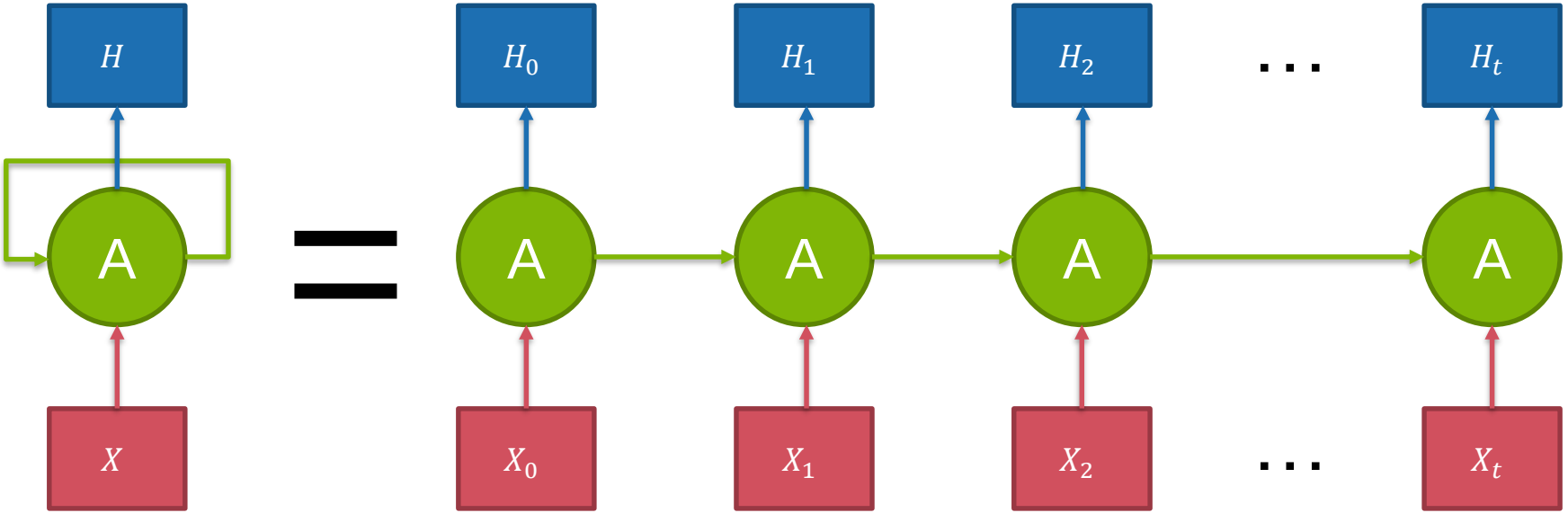
Redes Neuronales Recurrentes (RNN)

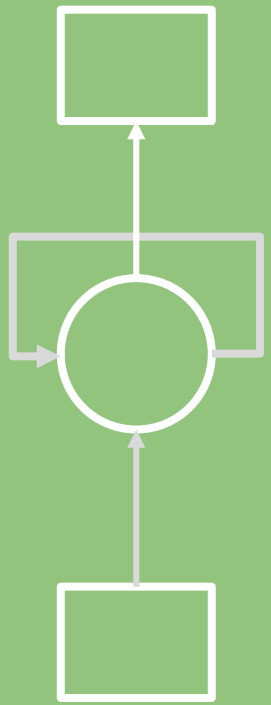
- Recurrencia
- Tipos de RNN
- Arquitecturas
- Entrenamiento
- Práctica



Recurrencia

Recurrencia

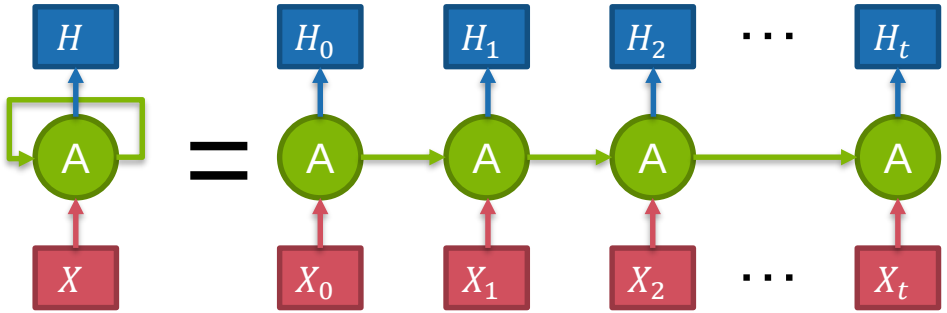




Tipos de RNN

Tipos de RNN

- Vanilla RNN
- Long-Short Term Memory (LSTM)
- Gated Recurrent Unit (GRU)
- Bidireccionales

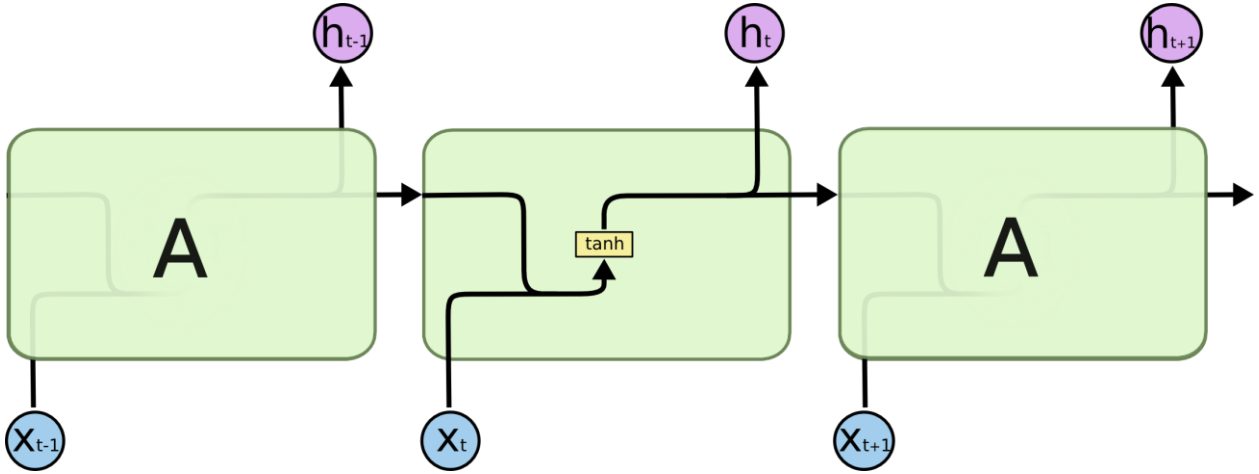


El primer tipo de RNN que se sugirio es llamado hoy en dia ‘Vanilla RNN’. Se identificaron los dos problemas mas dificiles al momento de entrenar una RNN. Uno compartido por las redes densamente conectadas, ‘**vanishing gradients**’, y el segundo particular de las RNN, ‘**exploding gradients**’.

A finales de los 90’ se propuso el tipo de celda LSTM. Tiene la ventaja que permite disminuir el ‘vanishing gradients’ al agregar compuertas y conexiones paralelas que permiten que el gradiente del error pueda fluir hacia atrás. Como desventaja encontramos que la cantidad de parámetros a entrenar es mucho mas grande.

Si bien han surgido muchos otros tipos de celdas recurrentes, la alternativa mas conocida a las LSTM, es la GRU. Tiene la ventaja de tener menos cantidad de parámetros a aprender y suele alcanzar rendimiento muy similar (a veces incluso superior) a las LSTM. Como desventaja en ciertos escenarios puede no tener capacidad suficiente.

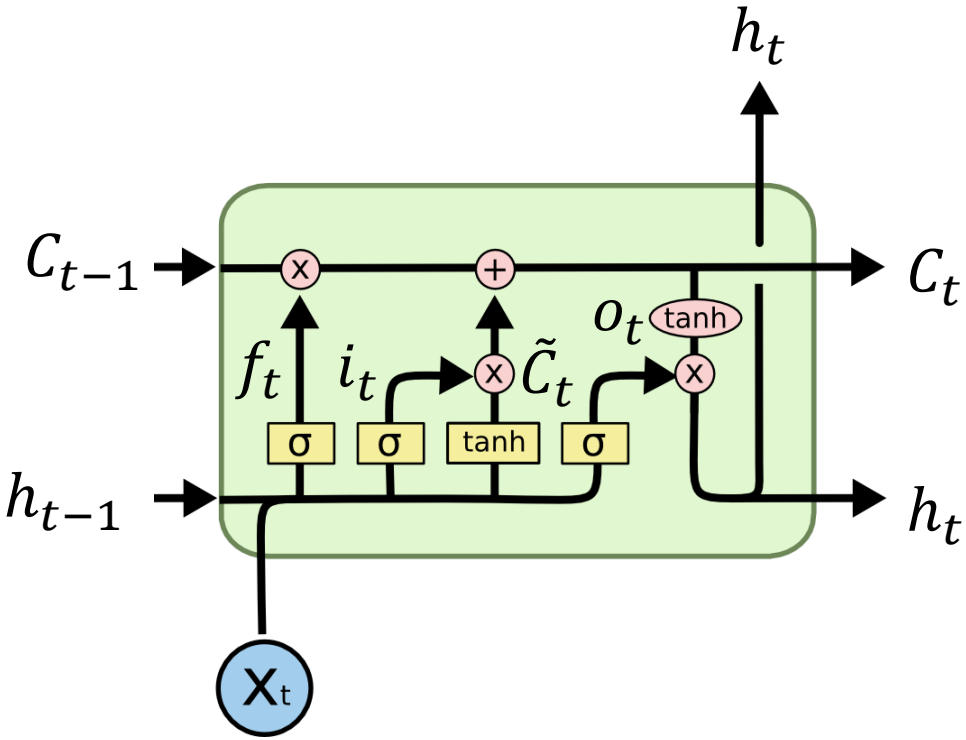
Tipos de RNN: Vanilla RNN



$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Fuente: Understanding LSTM Networks

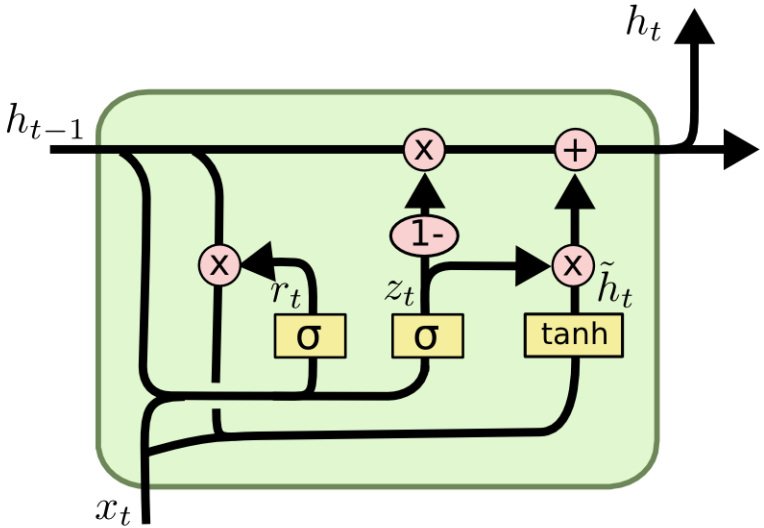
Tipos de RNN: LSTM



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_C . [h_{t-1}, x_t] + b_C)$$
$$C_t = f_t * C_t + i_t * \tilde{C}_t$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * tanh(C_t)$$

Fuente: Understanding LSTM Networks

Tipos de RNN: GRU



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

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

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

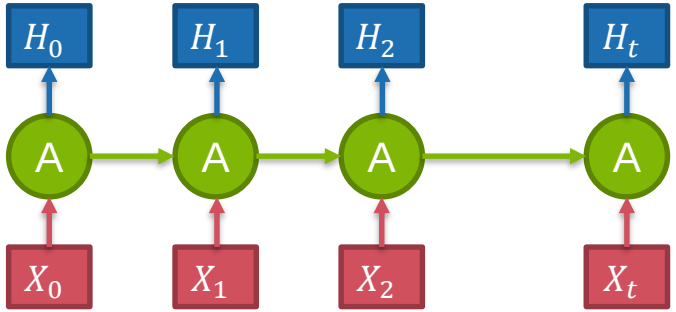
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Fuente: Understanding LSTM Networks

Tipos de RNN: Bidireccionales

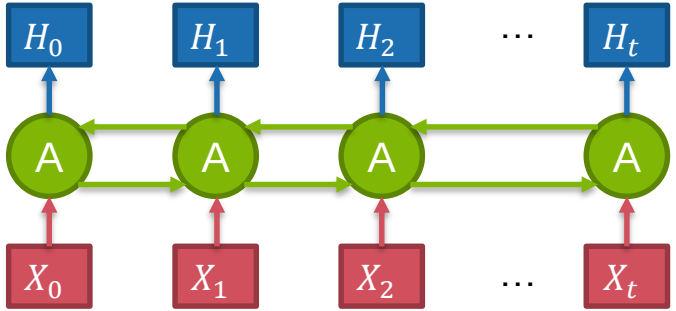
Causales

RNN ‘Foreward’

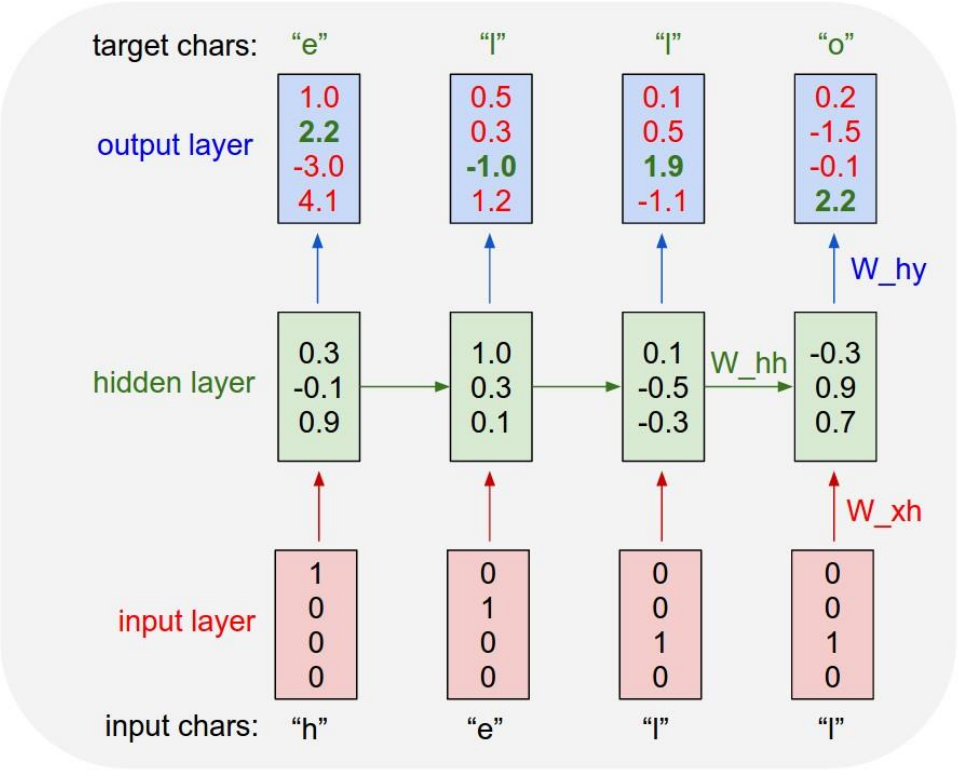


No-Causales

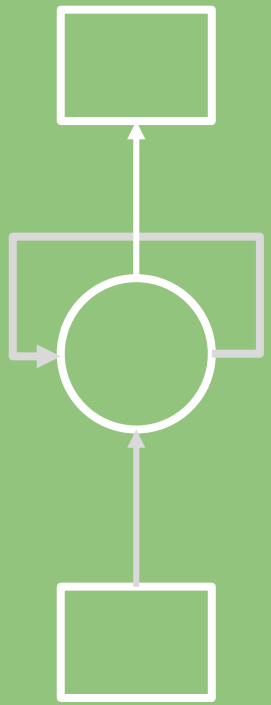
RNN ‘Bidireccionales’



Ejemplo



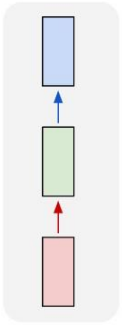
Fuente:The Unreasonable Effectiveness of Recurrent Neural Networks



Arquitecturas

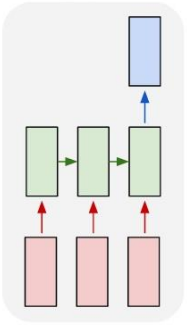
Arquitecturas

one to one



NN Fully Conected

many to one



Sequence to Vector (seq2vec)

one to many

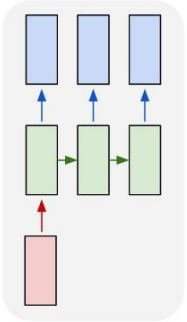
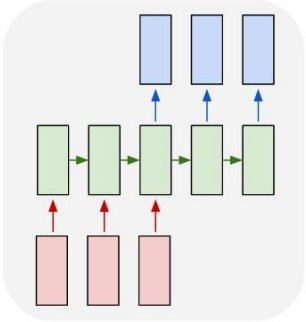
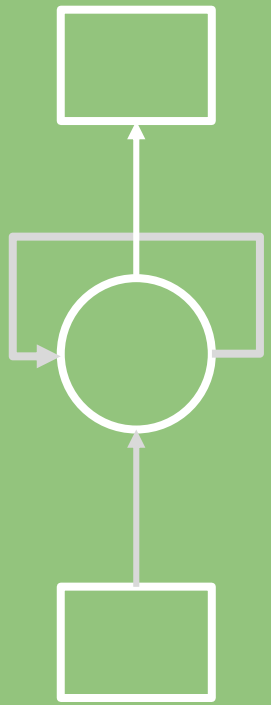


Image to Text

many to many



Sequence to Sequence (seq2seq)



Entrenamiento

Entrenamiento

El entrenamiento de las RNN se realiza calculando el gradiente del error. El algoritmo que se utiliza para propagar el gradiente se llama ‘Back-propagation through time’ (BPTT).

Para ello, primero se ‘desenrolla’ la serie temporal, se realiza un foreward pass de inicio a fin de la serie temporal. Una vez obtenida la salida de la red, se calcula el error y se realiza el backward pass. El tiempo de estas dos etapas es de orden $O(T)$, y no puede paralelizarse, debido a la naturaleza secuencial del algoritmo.

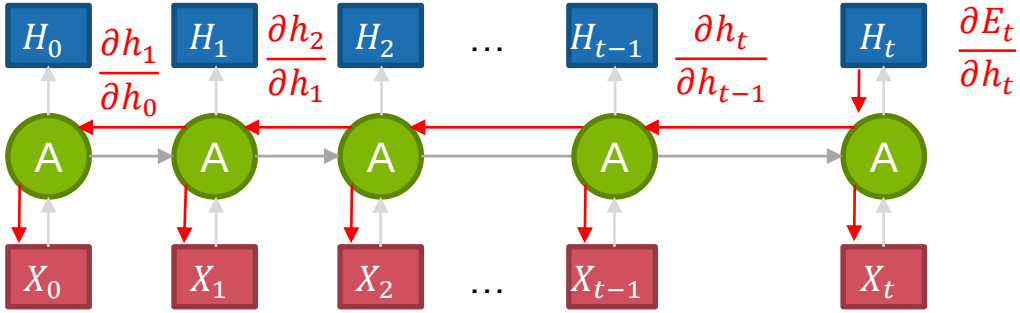
A cada instante ‘t’ se debe guardar los estados intermedios de la red para usarlos durante el backward pass. Por tal motivo, entrenar RNN requiere un uso extensivo de la memoria (GPU) y de tiempo.

Entrenamiento

Back-propagation through time (BPTT)

$$\frac{\partial E_t}{\partial h_1} = \frac{\partial E_t}{\partial h_t} \times \frac{\partial h_t}{\partial h_{t-1}} \times \cdots \times \frac{\partial h_2}{\partial h_1} \times \frac{\partial h_1}{\partial h_0}$$

$$\frac{\partial E_t}{\partial h_1} = \frac{\partial E_t}{\partial h_t} \prod_{i=1}^{i=t} \frac{\partial h_t}{\partial h_{t-1}}$$



Entrenamiento

Vanishing / Exploding Gradients

$$\frac{\partial E_t}{\partial h_1} = \frac{\partial E_t}{\partial h_t} \prod_{i=1}^{i=t} \frac{\partial h_t}{\partial h_{t-1}}$$

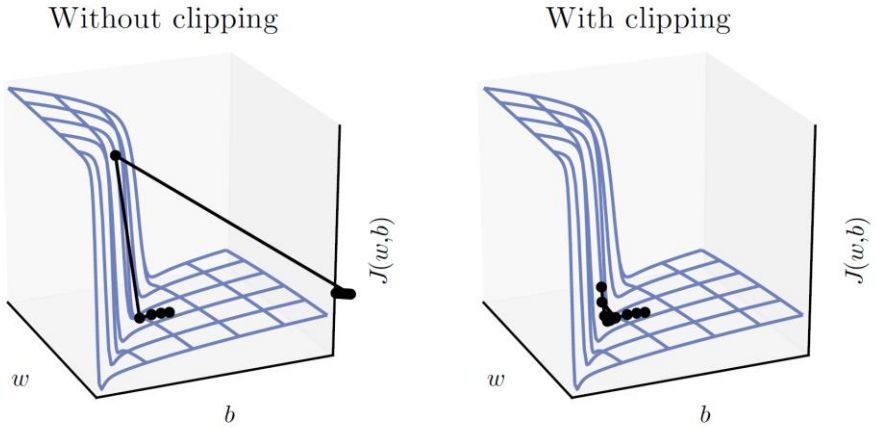
$$W_h^t = (V\ diag\ (\lambda) V^{-1})^t = V\ diag\ (\lambda)^t V^{-1}$$

$$\lambda < 1 : Vanishing$$

$$\lambda > 1 : Exploding$$

Clipping Gradients

Una técnica muy común para moderar el 'exploding gradients' es limitar el valor o la norma del gradiente. Se evitan 'saltos' muy grandes que nos lleven a regiones donde la función de costo es muy alta y perdamos el 'camino aprendido'



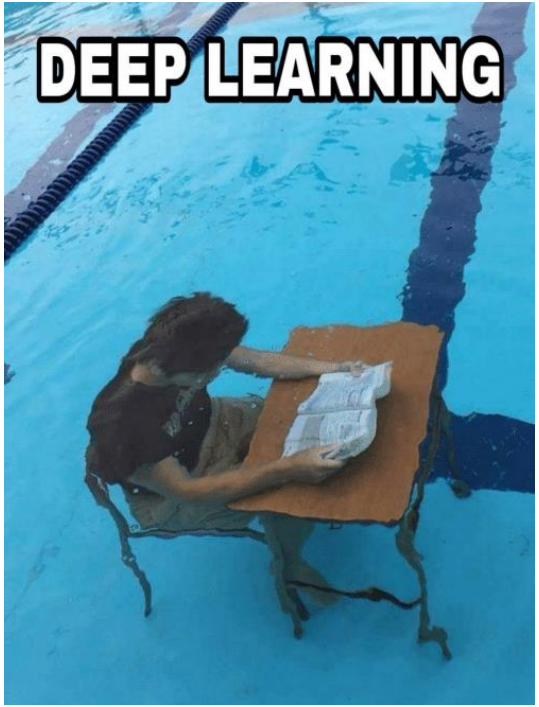
Fuente: Deep Learning book

Practica

Keras

TensorFlow

PyTorch



Windowing