Redes Neuronales

RECURRENTES

UNIVERSIDAD DE VALLADOLID

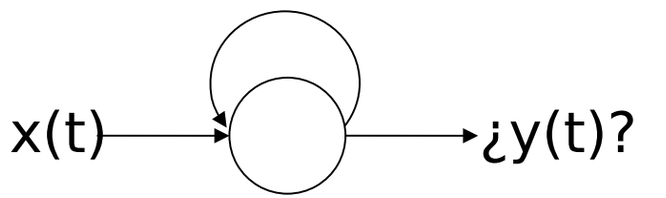
MINERÍA DE DATOS

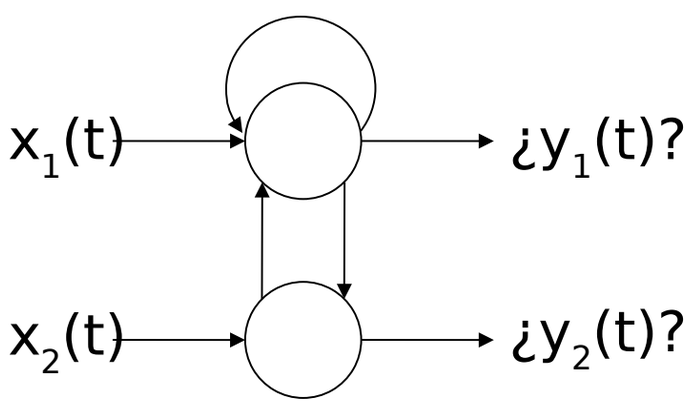
4º Curso. Grado en Ingeniería Informática
5º Curso. Doble Grado Informática/Estadística (INDAT)

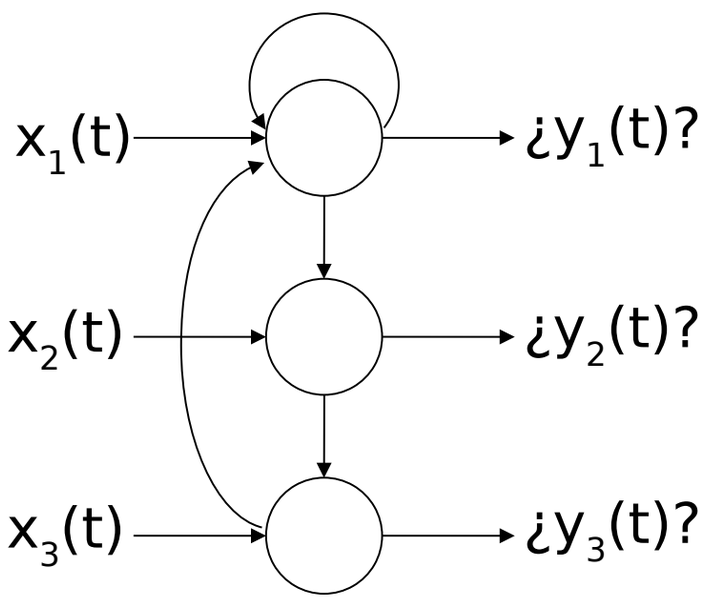
Departamento de Informática (ATC, CCIA y LSI)

Definición de recurrencia (RNA)

- **Conexiones** de una neurona **consigo** misma directa o indirectamente.







- **Más** conexiones ⇒ **capacidad** de representación
- Recurrencia ⇒ uso de la variable **tiempo**

Aplicaciones RNN

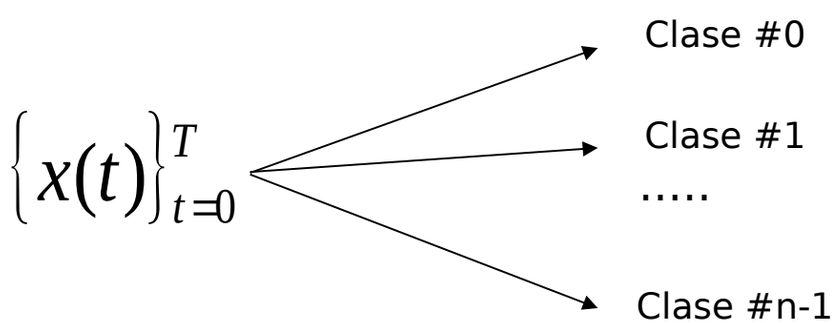
- Predicción temporal:

$$\{x(t)\}_{t=0}^{T} \rightarrow x(T+n)$$

Ejemplo: predicción de consumos, logística, bolsa, etc.

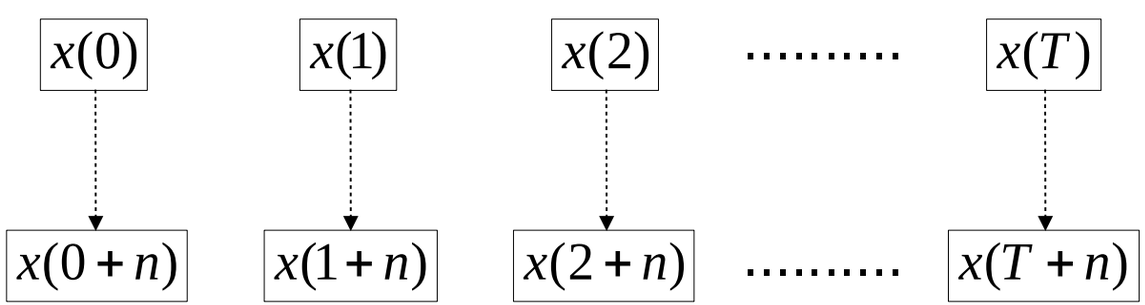
- Clasificación de secuencias (longitud variable):
  - Ejemplo: escenas de vídeo

(Aprendizaje Profundo)



Predicción temporal

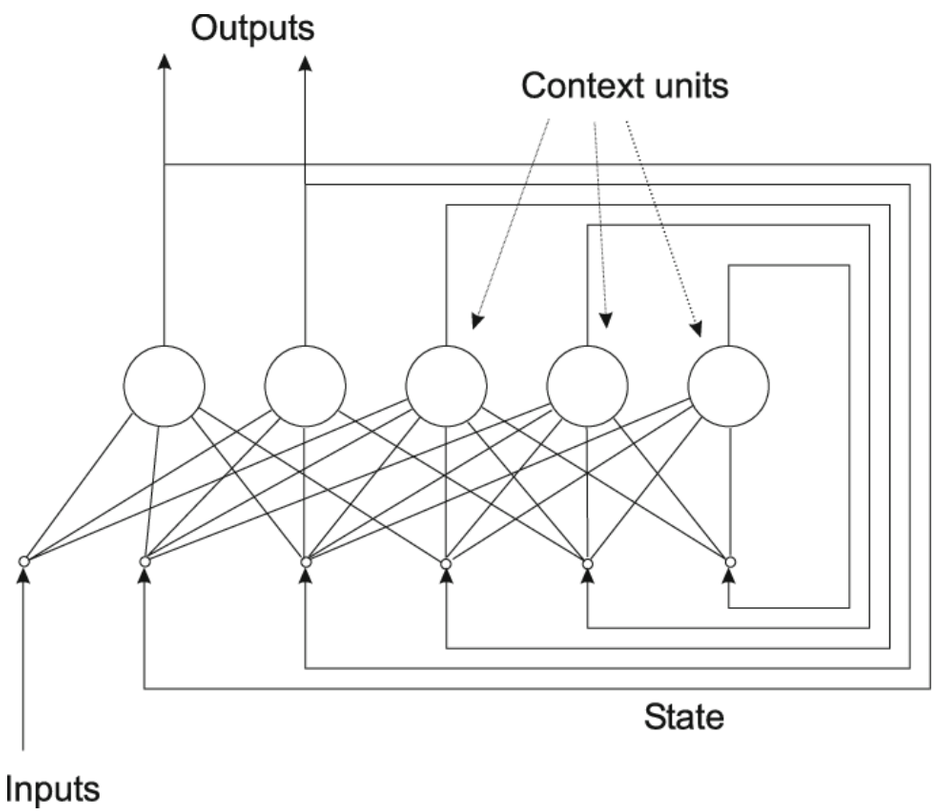
- Aprendizaje **por épocas**:
  - Sólo al final de una serie, aparece la salida deseada.
  - Es entonces cuando se produce la modificación de los pesos
- Aprendizaje en **tiempo real:**
  - Se dispone de la salida deseada para cada entrada de la secuencia
  - Modificación de pesos no espera al final de la secuencia: es en cada entrada

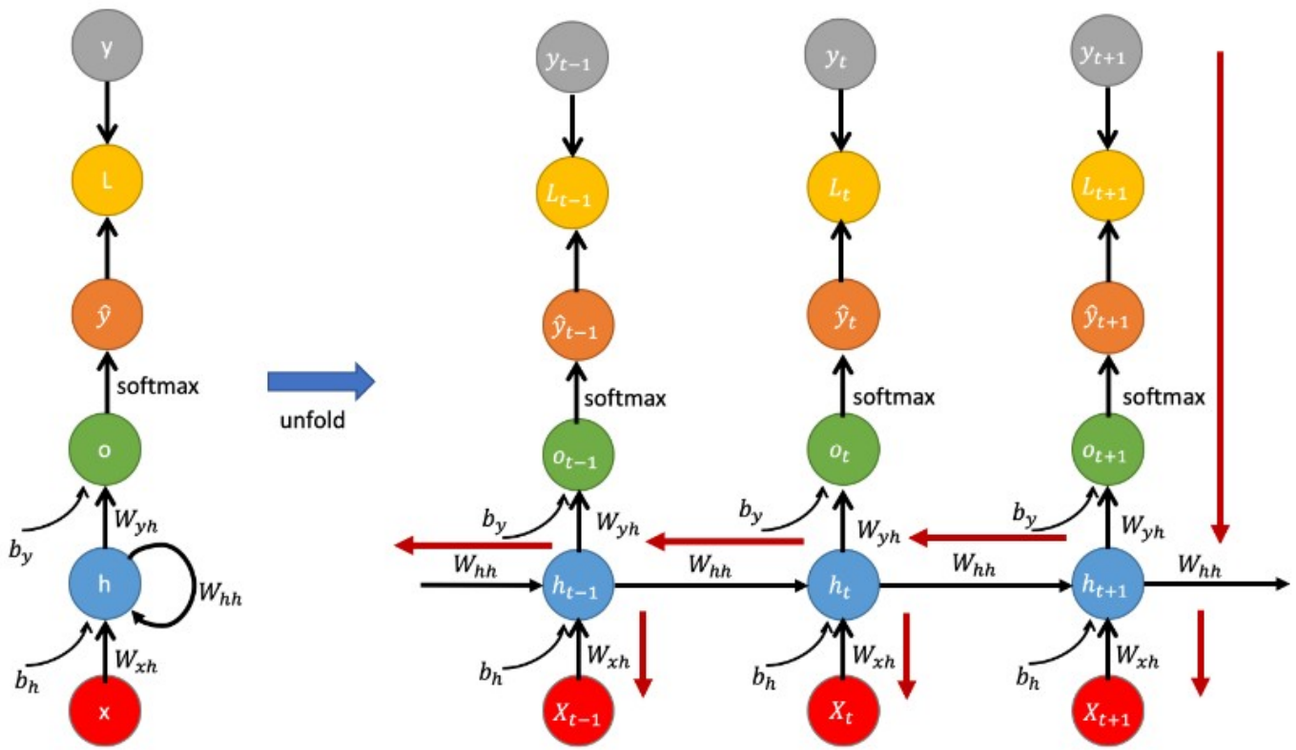


Ejemplos de redes

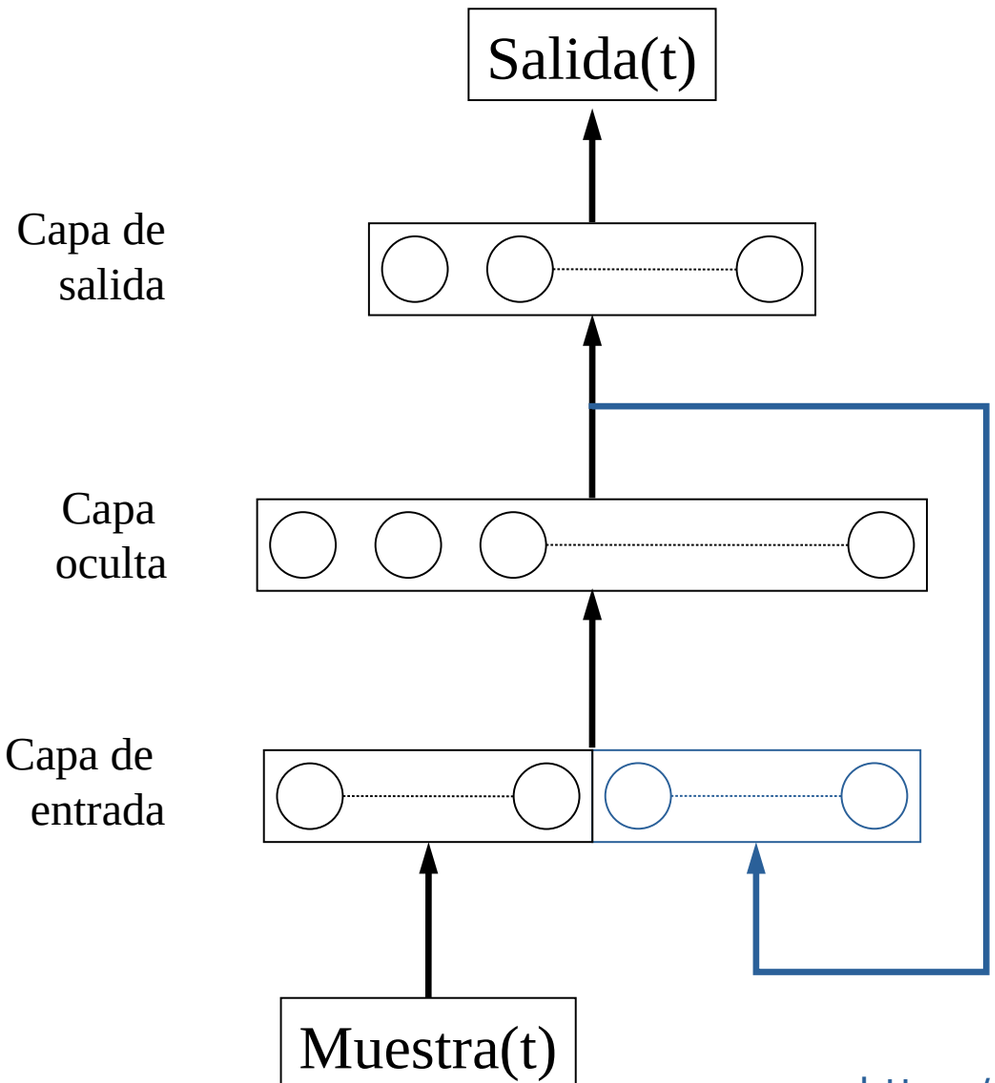
- Inspiradas en teorías de la **Física**:
  - Red de **Hopfield** (Cuántica)
  - Máquina de **Boltzmann** (Termodinámica: enfriamiento del vidrio)
- **Retropropagación** en el tiempo:
  - Consta de **una** sola **capa**
  - Se **desenrollan** en un diagrama temporal (mismas neuronas en diferentes instantes)
  - Da lugar a una **capa replicada** en el tiempo
  - Minimización de una **función de coste**
  - Da la imagen de un **MLP (misma capa)**
- Redes **parcialmente recurrentes**: Elman y Jordan

Totalmente Recurrentes: Retropropagación en el tiempo





Red de Elman

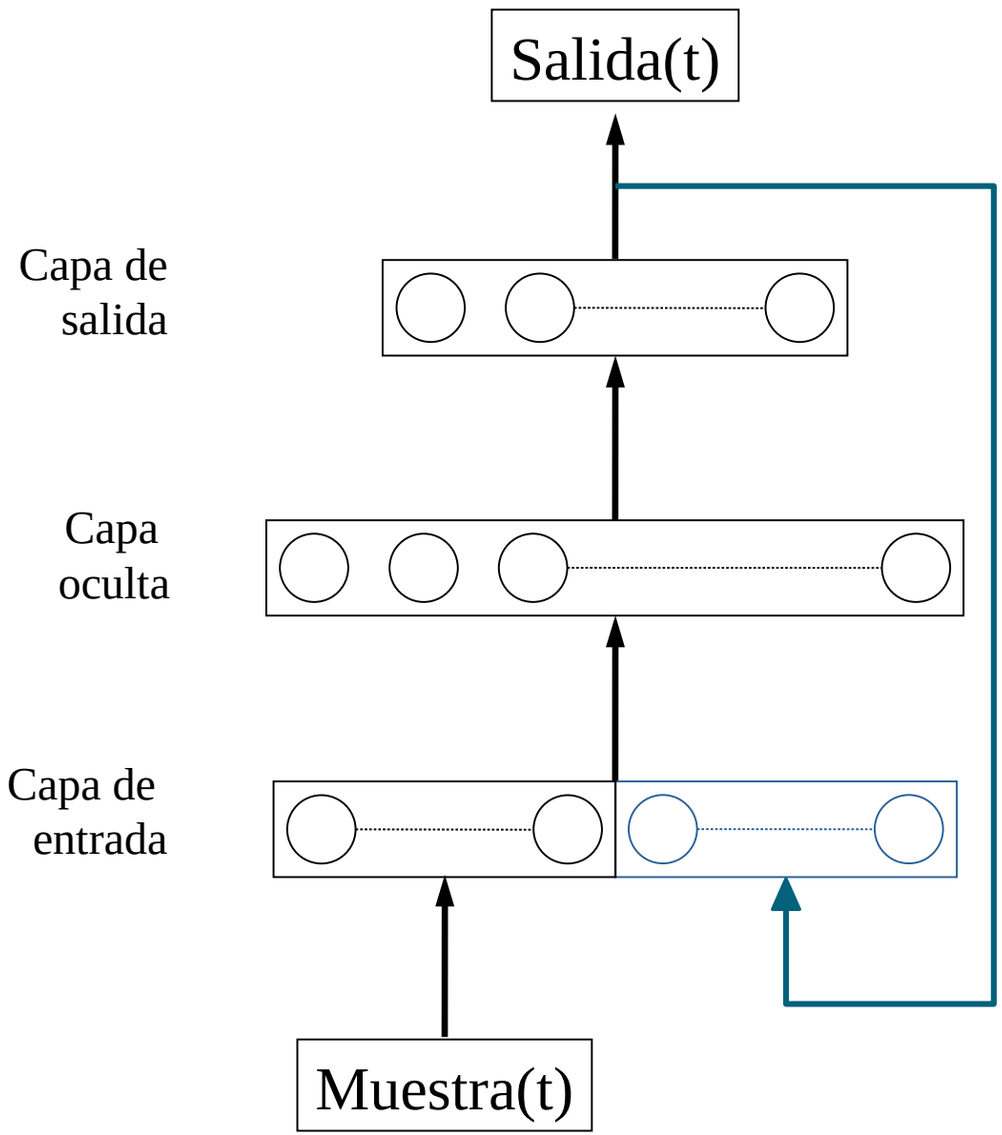


$$y_i^1(t+1) = F\left(\sum_{j=0}^{N[0]} w_{ij}^1 y_j^0(t+1)\right)$$

$$y_j^0(t+1) = F\left(\sum_{k=0}^{N0} w_{jk}^0 x_k(t+1) + \sum_{i=1}^{N[0]} w_{j,N0+i}^0 y_i^0(t)\right)$$

https://pytorch.org/docs/stable/generated/torch.nn.RNN.html

Red de Jordan



$$y_i^1(t+1) = F\left(\sum_{j=0}^{N[0]} w_{ij}^1 y_j^0(t+1)\right)$$

$$y_j^0(t+1) = F\left(\sum_{k=0}^{N0} w_{jk}^0 x_k(t+1) + \sum_{i=1}^{N[1]} w_{j,N0+i}^0 y_i^1(t)\right)$$

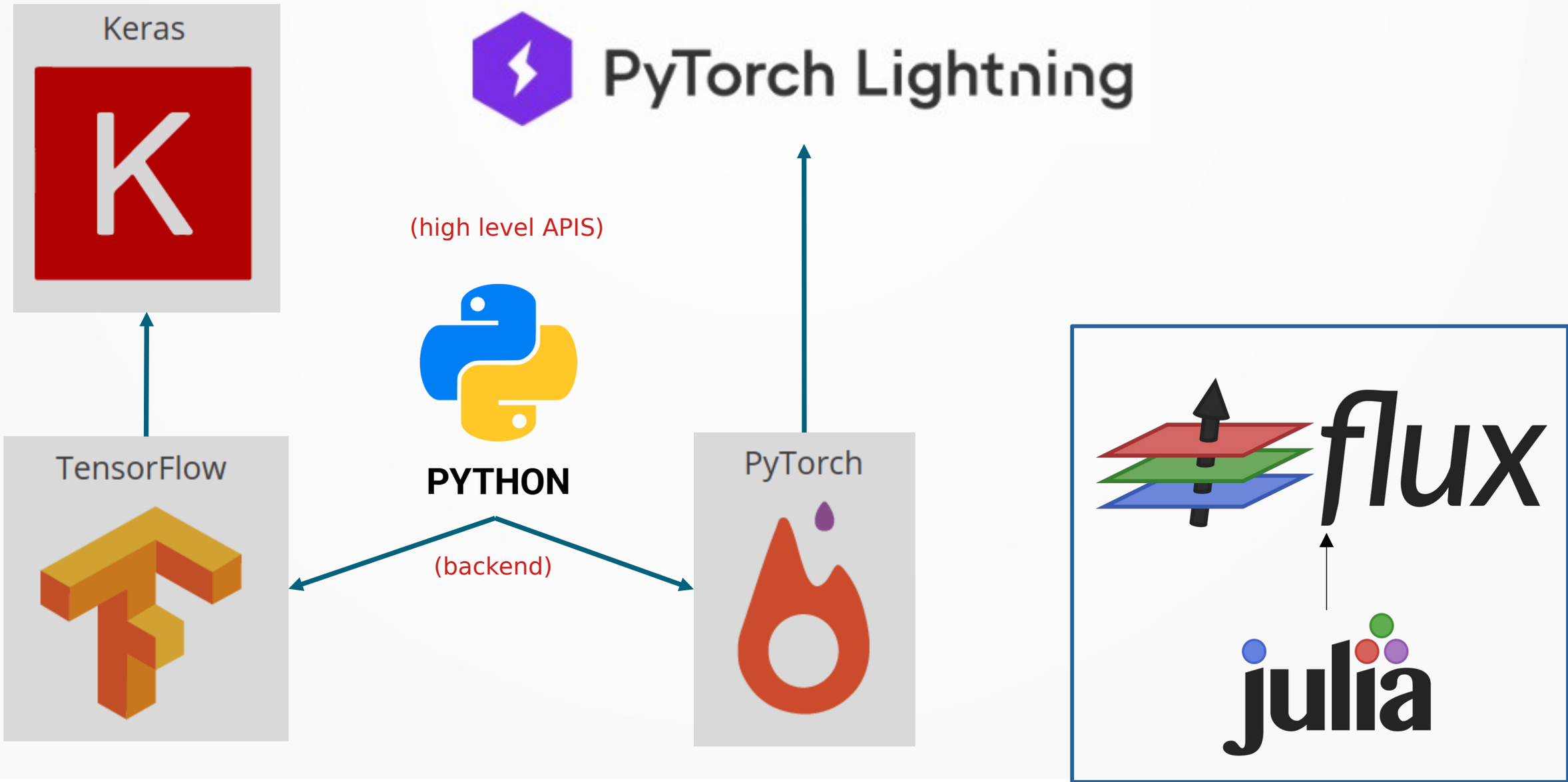
Aprendizaje Elman y Jordan

- Por **secuencias**:
  - Se aplica el algoritmo del **gradiente** al error cuadrático medio **acumulado** en toda la secuencia
  - Para hacerlo más independiente de la **longitud** de la serie, este error se suele **dividir** por su número de entradas
  - Compatible con el procesamiento por lotes (batch)
- En **tiempo real** precisa salida deseada para cada muestra
  - El gradiente se aplica **muestra a muestra**
- Término **momento** para evitar caer en mínimos locales

Ejercicio:

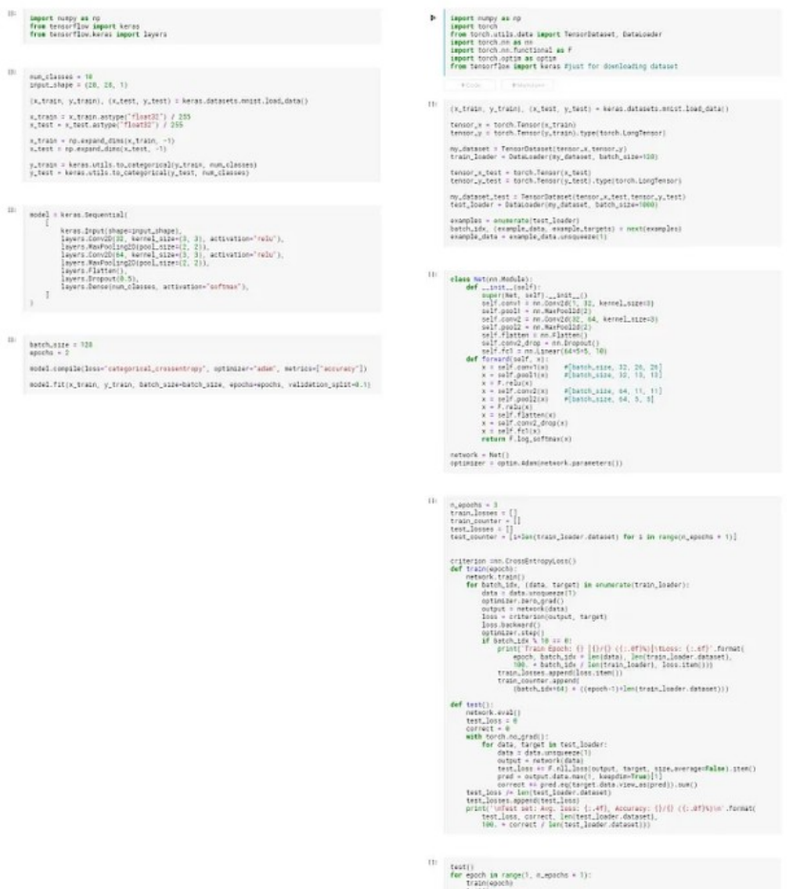
- Predicción temporal de valores bursátiles:
  - Red de Elman
  - Cada secuencia será del último mes: 20 valores
  - La salida deseada será el siguiente valor en la serie global
  - Fichero de datos: cotización Iberdrola en los años (2010-24)
    - Descarga del campus virtual
  - Hay que normalizar [0,15]. Usad la tanh como función de activación
  - Tasa de aciertos en función del error relativo entre predicción y la salida (margen = 2% ~ **5**%).

Deep Learning

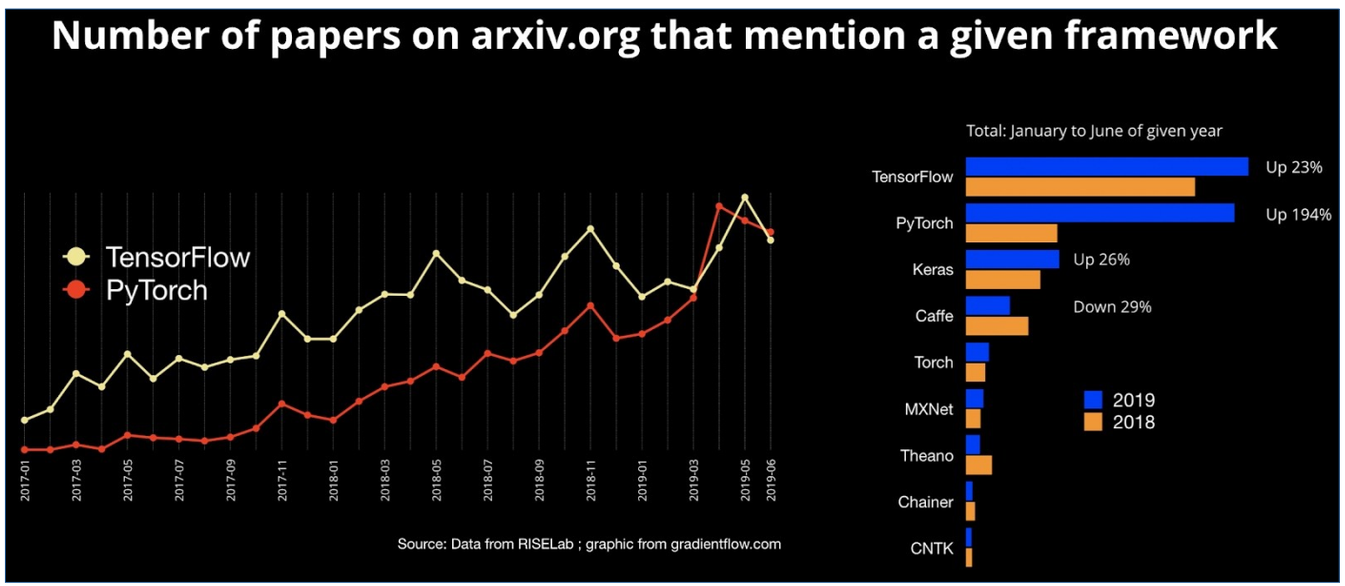


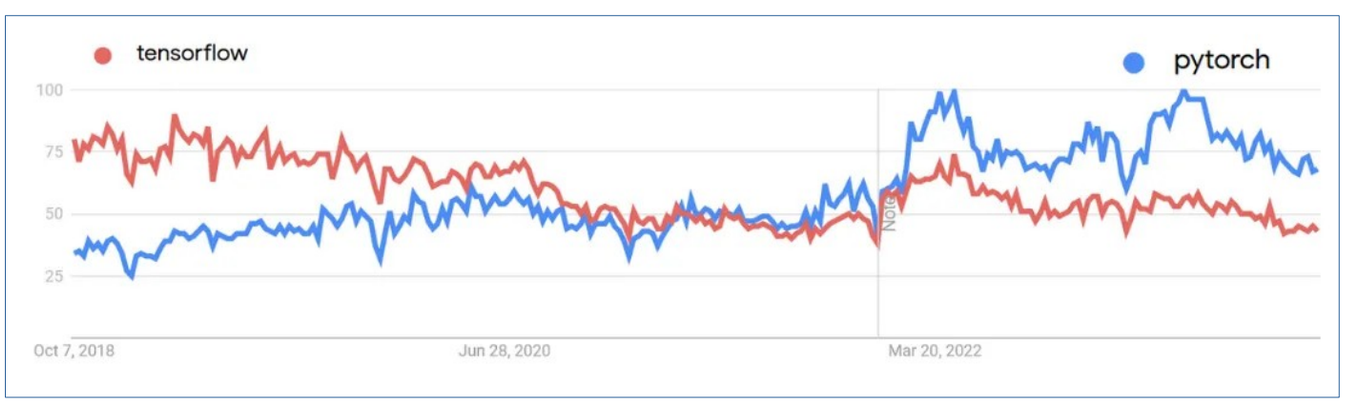
Keras vs Pytorch

High vs Low Level



Simple CNN in both frameworks.





Datos suministrados por Google

Predicción de una serie temporal

Dataset: Ventas de Amazon

	Date	Open	High	Low	Close	Adj Close	Volume
0	1997-05-15	0.121875	0.125000	0.096354	0.097917	0.097917	1443120000
1	1997-05-16	0.098438	0.098958	0.085417	0.086458	0.086458	294000000
2	1997-05-19	0.088021	0.088542	0.081250	0.085417	0.085417	122136000
3	1997-05-20	0.086458	0.087500	0.081771	0.081771	0.081771	109344000
4	1997-05-21	0.081771	0.082292	0.068750	0.071354	0.071354	377064000
...	...	...	...	...	...	...	...
6511	2023-03-30	101.550003	103.040001	101.010002	102.000000	102.000000	53633400
6512	2023-03-31	102.160004	103.489998	101.949997	103.290001	103.290001	56704300
6513	2023-04-03	102.300003	103.290001	101.430000	102.410004	102.410004	41135700
6514	2023-04-04	102.750000	104.199997	102.110001	103.949997	103.949997	48662500
6515	2023-04-05	103.910004	103.910004	100.750000	101.099998	101.099998	45103000

6516 rows × 7 columns

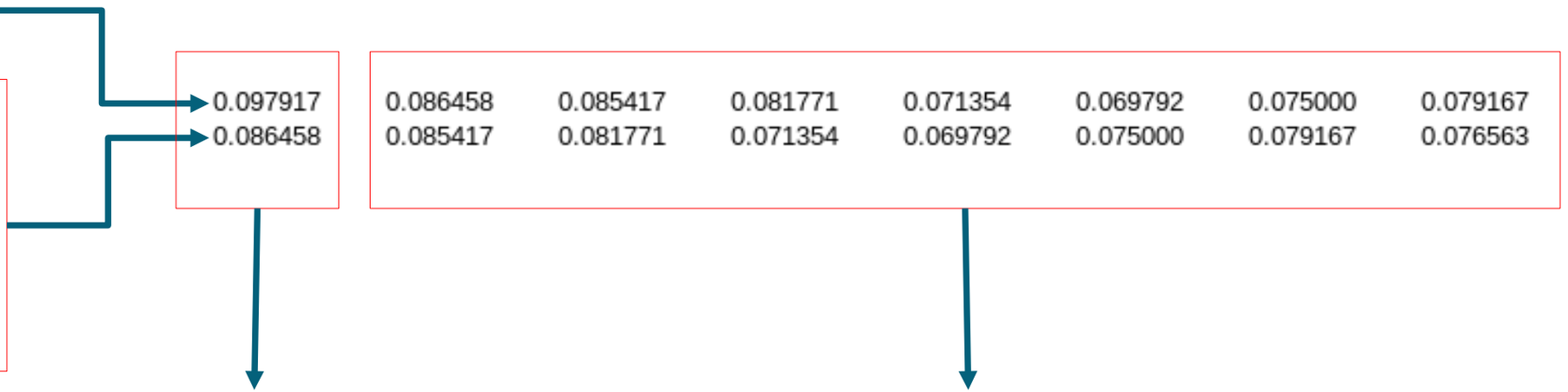
	Date	Close
0	1997-05-15	0.097917
1	1997-05-16	0.086458
2	1997-05-19	0.085417
3	1997-05-20	0.081771
4	1997-05-21	0.071354
...	...	...
6511	2023-03-30	102.000000
6512	2023-03-31	103.290001
6513	2023-04-03	102.410004
6514	2023-04-04	103.949997
6515	2023-04-05	101.099998

6516 rows × 2 columns

HIPÓTESIS: cada valor de venta al cierre dependerá de los siete anteriores (una semana)

Creación Dataset Serie Temporal

Date	Close
1997-05-15	0.097917
1997-05-16	0.086458
1997-05-19	0.085417
1997-05-20	0.081771
1997-05-21	0.071354
1997-05-22	0.069792
1997-05-23	0.075000
1997-05-27	0.079167
1997-05-28	0.076563
1997-05-29	0.075260
1997-05-30	0.075000
1997-06-02	0.075521
1997-06-03	0.073958
1997-06-04	0.070833
1997-06-05	0.077083
1997-06-06	0.082813
1997-06-09	0.084375
1997-06-10	0.079167
1997-06-11	0.077083
1997-06-12	0.080208
1997-06-13	0.079167
1997-06-16	0.078646
1997-06-17	0.075260
1997-06-18	0.075521
1997-06-19	0.075521
1997-06-20	0.076302
1997-06-23	0.075000
1997-06-24	0.075521




```
df.set_index('Date', inplace=True)

for i in range(1, n_steps+1):
    df[f'Close(t-{i})'] = df['Close'].shift(i)
df.dropna(inplace=True)
```

Escalado y HoldOut

- La matriz de entradas se organiza como: X[batch, secuencia-i,input]
- La de salida: y[batch, ouput]
- A los DataLoaders se le pasa **batch=1 (aconsejan que sea 2^n)**
- Como la función de activación es **tanh** (-, +), las entradas se suelen escalar a sus valores asintóticos: [-1,1]
- El método de reserva no se estratifica, pero hay que procurar que lleven asociada la **misma casuística** ambos (train y test).
- Pero se elegirán las **N primeras** para **aprendizaje** y el **resto** para **test**. El motivo es poder representar gráficamente una métrica

DataSets y DataLoaders

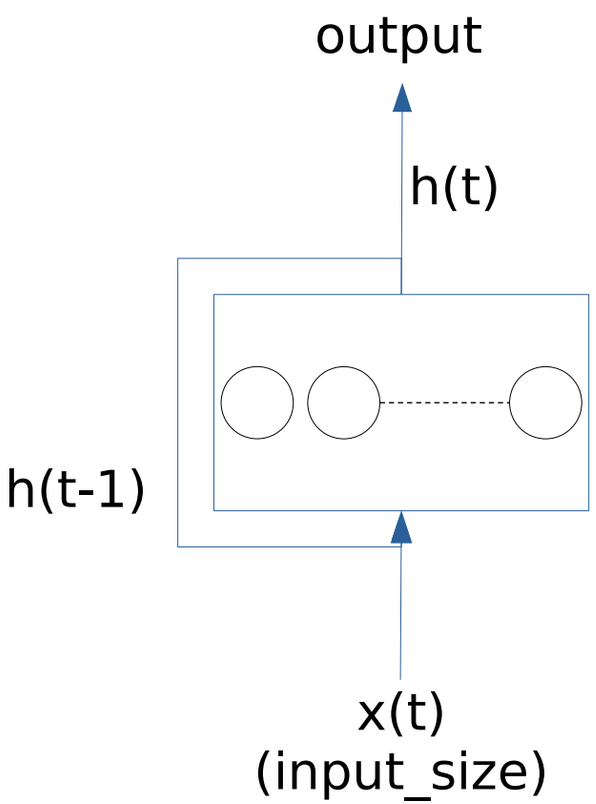
```
from torch.utils.data import Dataset

class TimeSeriesDataset(Dataset):
    def __init__(self, X, y):
        self.X = X
        self.y = y

    def __len__(self):
        return len(self.X)

    def __getitem__(self, i):
        return self.X[i], self.y[i]
```

```
train_dataset = TimeSeriesDataset(X_train, y_train)
test_dataset = TimeSeriesDataset(X_test, y_test)
```

```
from torch.utils.data import DataLoader

batch_size = 16

train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

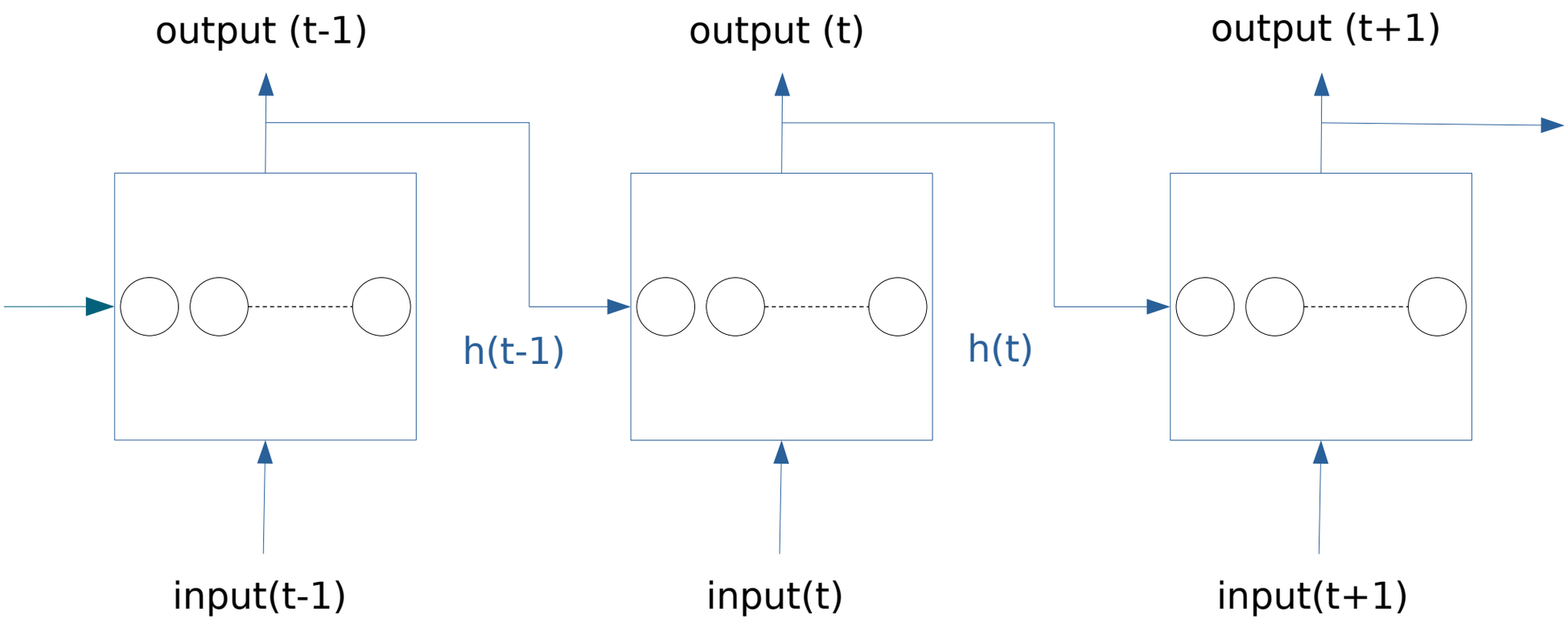
Celda RNN - Elman (PyTorch)

https://pytorch.org/docs/stable/generated/torch.nn.RNN.html#torch.nn.RNN




```
for x_batch, y_batch in train_loader:
    hidden_layer_size = 5
    h = torch.zeros(1, x_batch.shape[1], hidden_layer_size)
    modelo = nn.RNN(input_size=1,
                    hidden_size=hidden_layer_size, num_layers=1)
    output, h = modelo(x_batch, h)
    break
output, h, output.shape, h.shape
```

Desenrolle en el tiempo (pytorch)



Red de Elman (PyTorch)

```
class Elman(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super().__init__()
        self.hidden_size = hidden_size
        self.hidden_layer = nn.RNN(input_size, self.hidden_size, batch_first=True)
        self.output_layer = nn.Linear(self.hidden_size, 1)

    def forward(self, x):
        batch_size = x.size(0)
        h0 = torch.zeros(1, batch_size, self.hidden_size)
        out, h = self.hidden_layer(x, h0)
        output = self.output_layer(out[:, -1, :])
        return output
```

```
learning_rate = 0.001
num_epochs = 200
loss_function = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

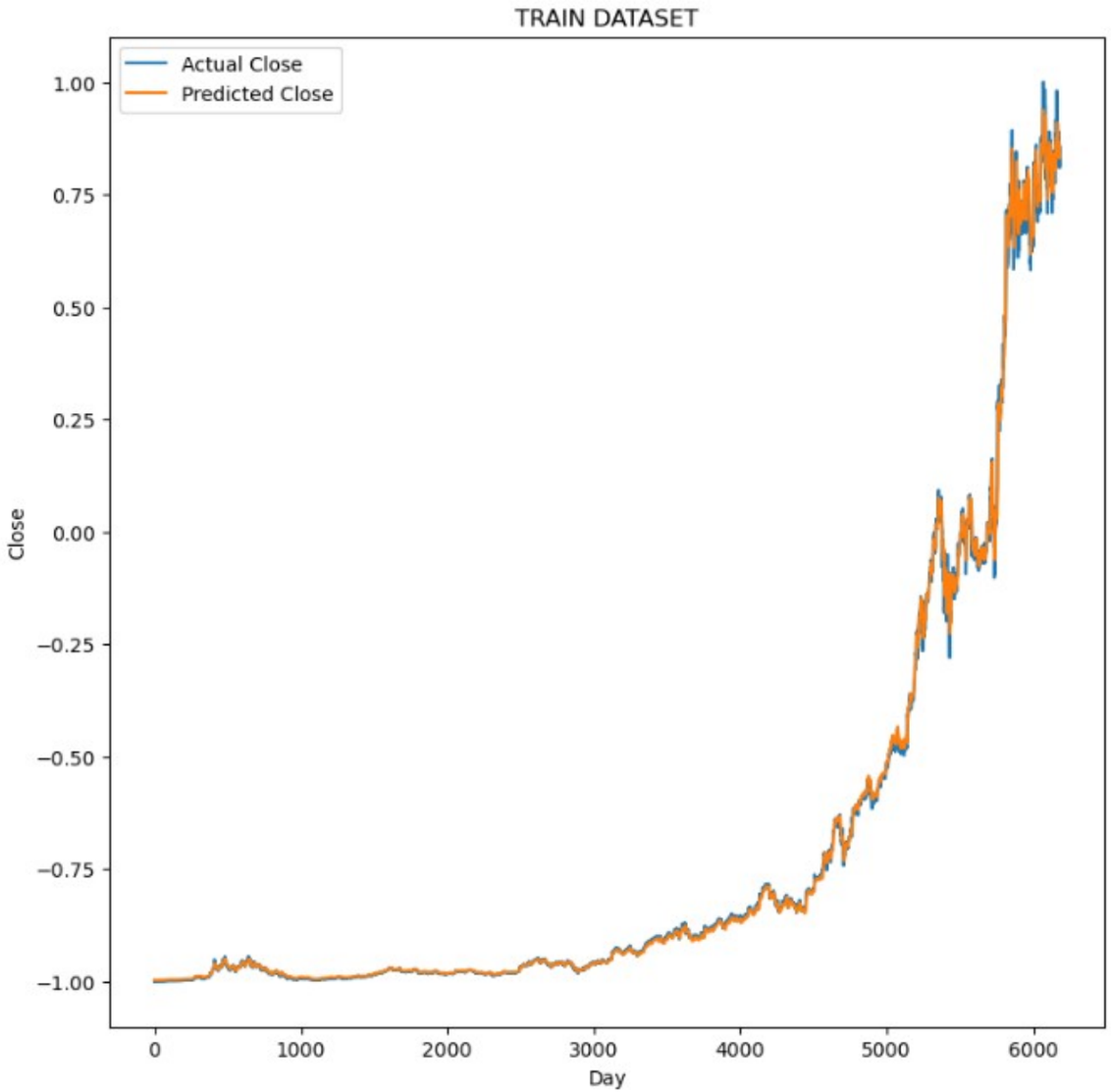
Entrenamiento

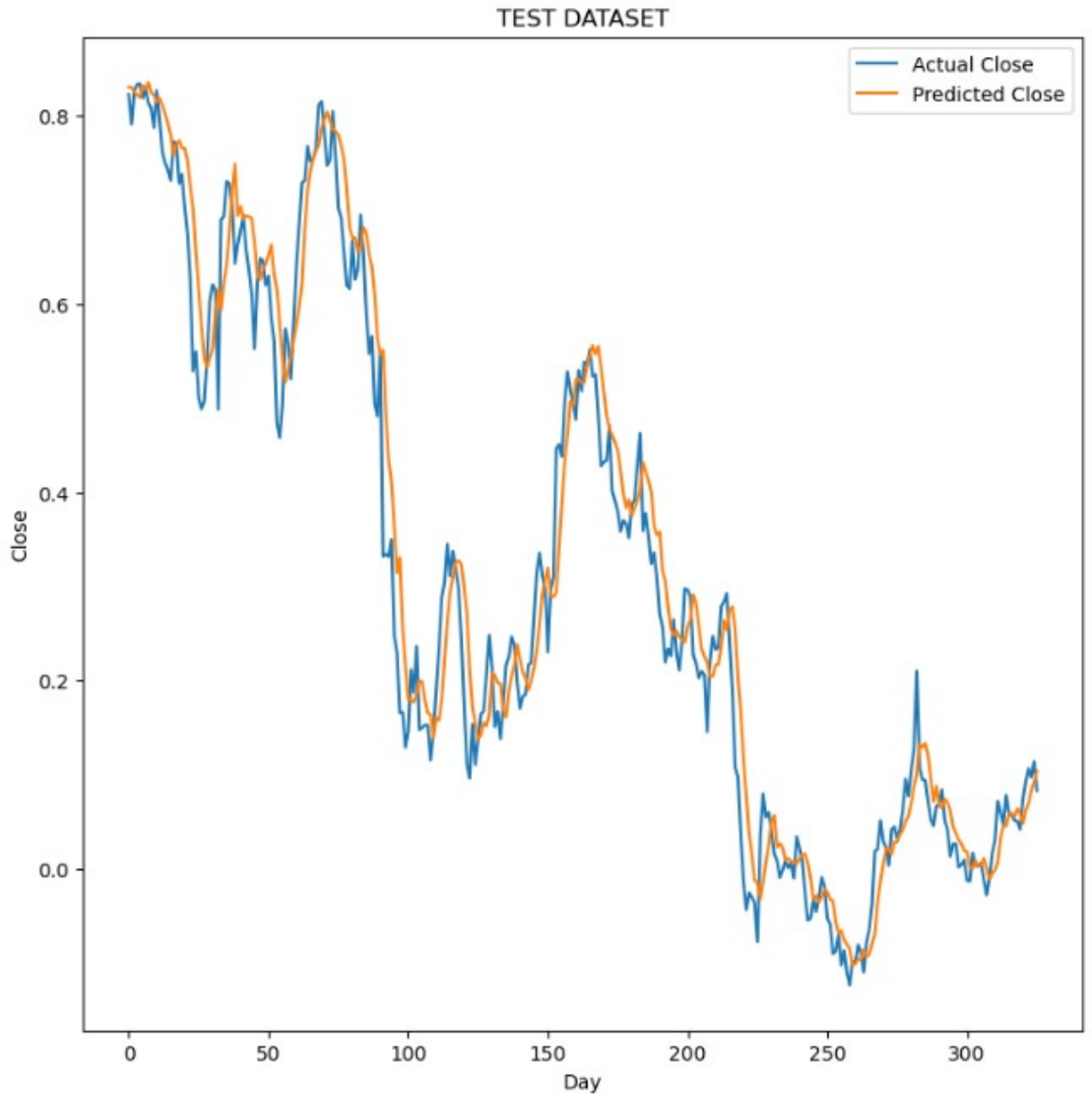
```
def train_one_epoch():
    model.train(True)
    print(f'Epoch: {epoch + 1}')
    running_loss = 0.0

    for x_batch, y_batch in tqdm(train_loader):

        output = model(x_batch)
        loss = loss_function(output, y_batch)
        running_loss += loss.item()

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    return running_loss/len(train_loader)
```

Test

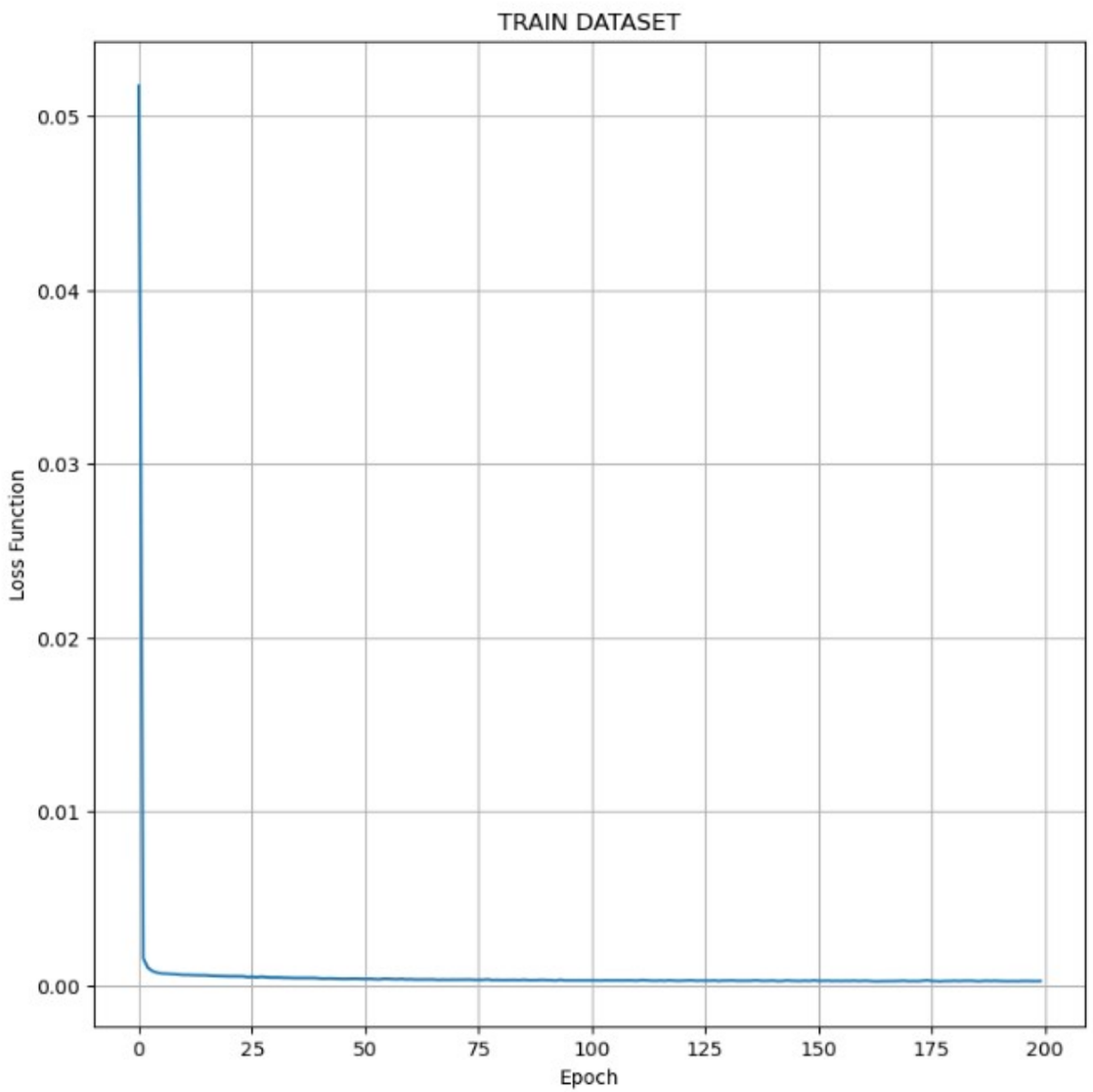
```
def validate_one_epoch():
    model.train(False)
    running_loss = 0.0

    for x_batch, y_batch in test_loader:

        with torch.no_grad():
            output = model(x_batch)
            loss = loss_function(output, y_batch)
            running_loss += loss.item()

    return running_loss / len(test_loader)
```

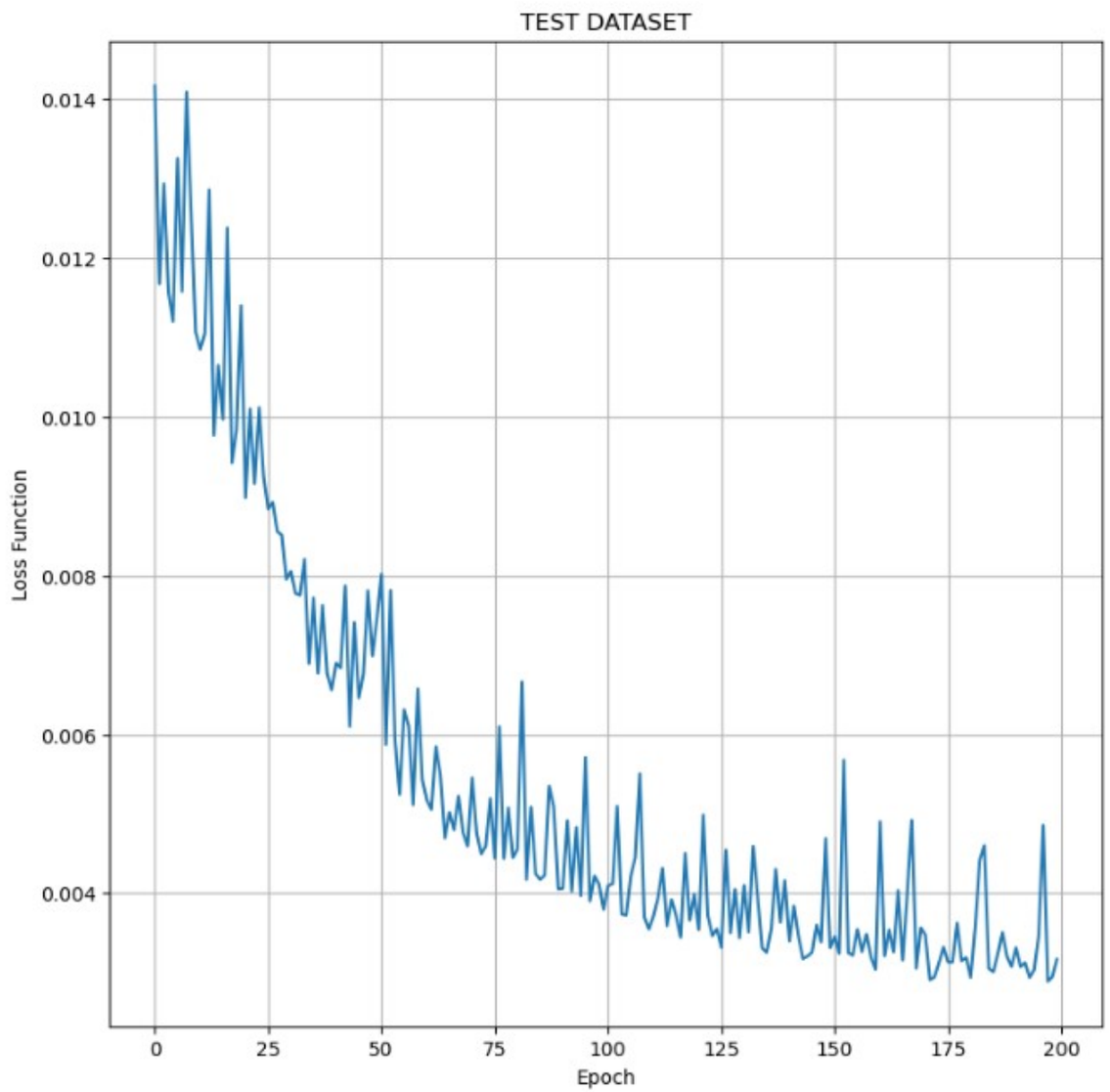
Predicción vs Salida Real





Función de Coste / Pérdida



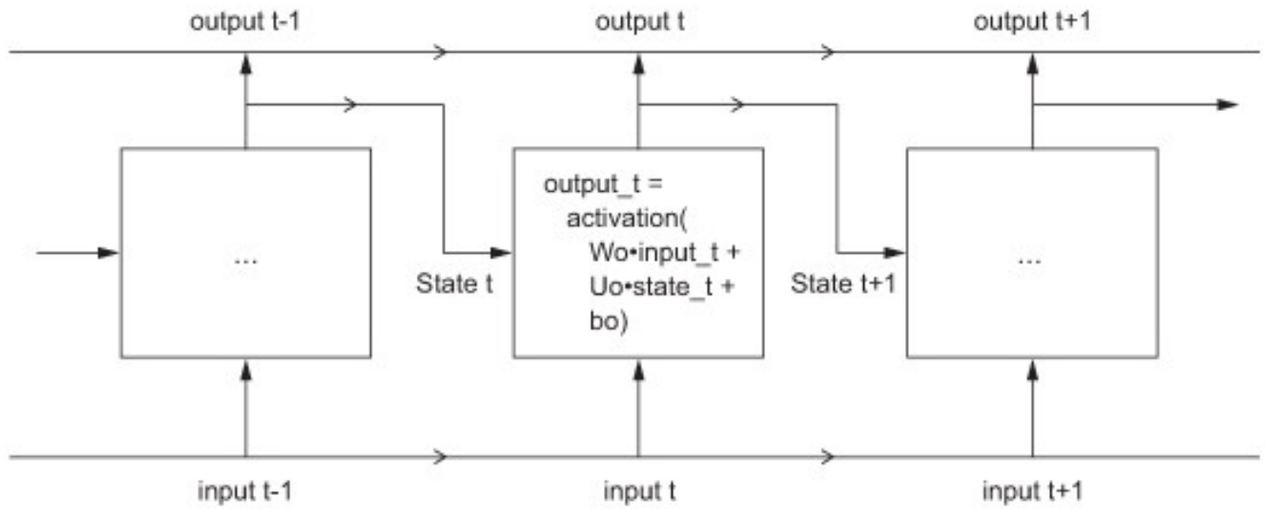


Motivación LSTM

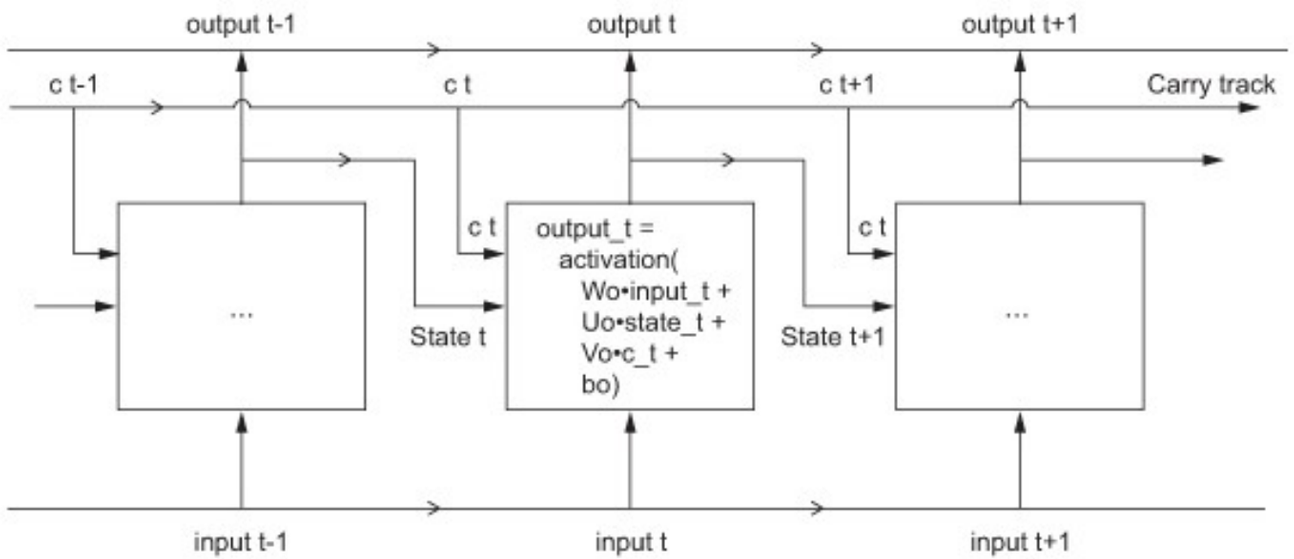
- Cuando las **secuencias** son relativamente **grandes**, tanto Elman o Jordan, no da buenos resultados.
- Problema de la **Evanescencia del Gradiente**.
- Aparece también en las RN, en general, al aumentar el número de capas:
  - La magnitud del gradiente se va reduciendo de una capa a otra, a medida que se aleja de la salida
  - Llega a ser **imperceptible** en **capas** muy **profundas**

Réplica en el tiempo nn.RNN y nn.LSTM

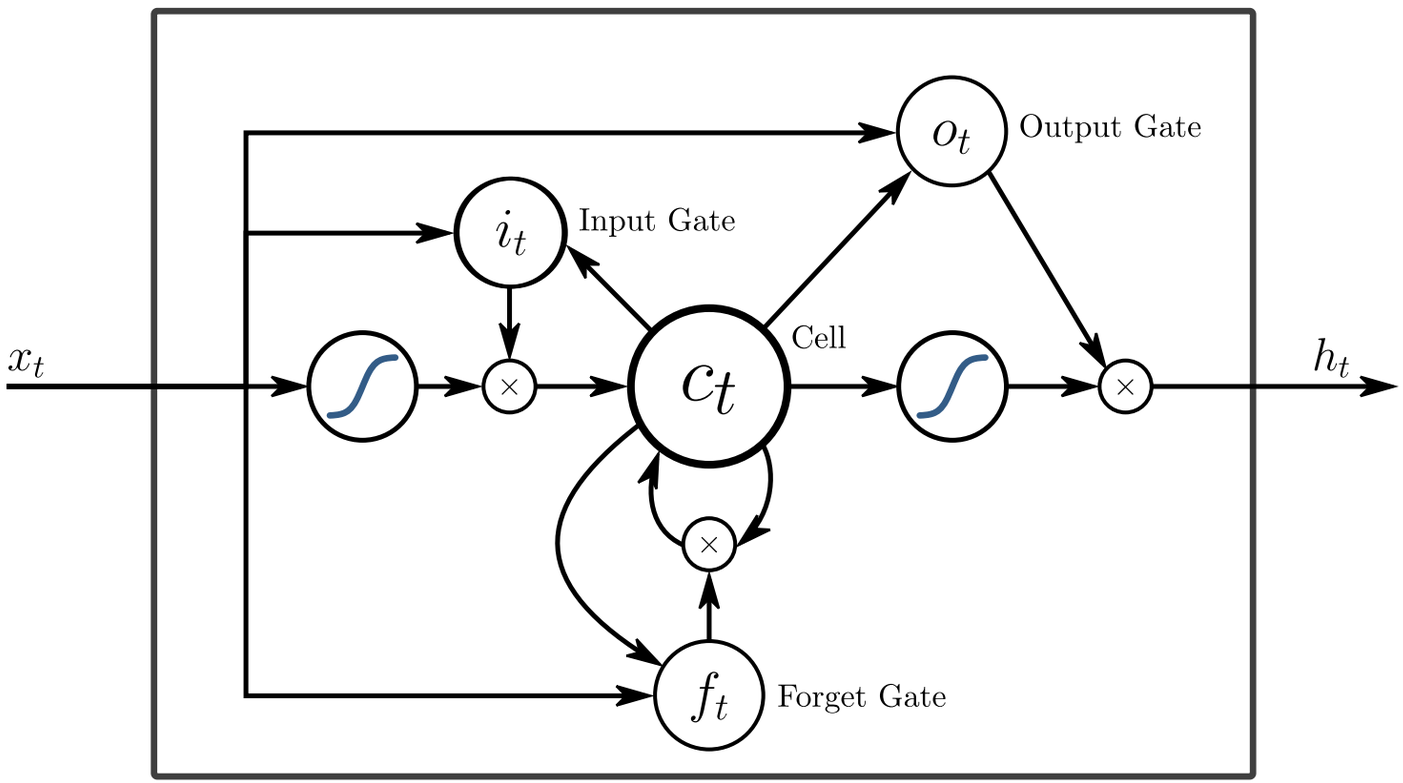
nn.RNN:
- Sólo recuerda la última salida
- El resto, a través del estado
- Poca influencia muy atrás en tiempo



nn.LSTM:
- Añade una “cinta transportadora”
- Recordar entradas anteriores
- Incluso alejadas en el tiempo



Modelo LSTM (Long-Short Time Memory)



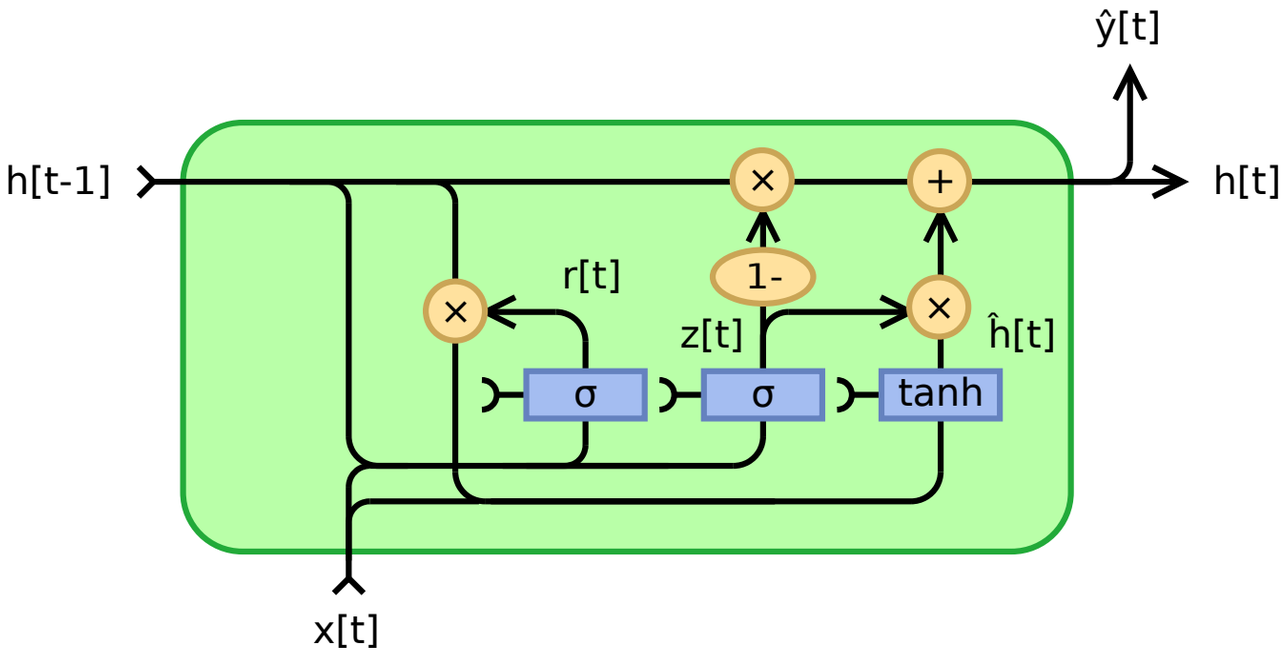
$$f_t = \sigma_g(W_f x_t + U_f c_{t-1} + b_f)$$
$$i_t = \sigma_g(W_i x_t + U_i c_{t-1} + b_i)$$
$$o_t = \sigma_g(W_o x_t + U_o c_{t-1} + b_o)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + b_c)$$
$$h_t = \sigma_h(o_t \circ c_t)$$

- σ_g: sigmoid function.
- σ_c: hyperbolic tangent function.
- σ_h: hyperbolic tangent function

- $x_t \in \mathbb{R}^d$: input vector to the LSTM unit
- $f_t \in \mathbb{R}^h$: forget gate's activation vector
- $i_t \in \mathbb{R}^h$: input/update gate's activation vector
- $o_t \in \mathbb{R}^h$: output gate's activation vector
- $h_t \in \mathbb{R}^h$: hidden state vector also known as output vector of the LSTM unit
- $c_t \in \mathbb{R}^h$: cell state vector
- $W \in \mathbb{R}^{h \times d}$, $U \in \mathbb{R}^{h \times h}$ and $b \in \mathbb{R}^h$: weight matrices and bias vector parameters

Fuente: wikipedia

GRU (Gated Recurrent Unit)



$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \phi_h(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h)$$

- x_t: input vector
- h_t: output vector
- z_t: update gate vector
- r_t: reset gate vector
- W, U and b: parameter matrices and vector

- σ_g: The original is a sigmoid function.
- ϕ_h: The original is a hyperbolic tangent.

Fuente: wikipedia

Práctica: predicción de secuencias temporales – Acciones Iberdrola 2012-24

- En un sólo notebook de jupyter (pytorch) entrega un sólo programa que entrene:
  - Una red de Elman con dos capas ocultas de 5 neuronas
  - La hipótesis es que el valor al cierre va a depender sólo de los 20 anteriores (mes).
  - Entrenarlo con 100 – 200 épocas con una valor del lote de 1
  - Se tomarán como función de pérdida el Error Cuadrático Medio (MSE)
  - El optimizador será Adam con un coeficiente de aprendizaje de 0,001
  - Se representará la evolución del MSE para aprendizaje y test a lo largo de las épocas
  - Finalmente, se contruirá una gráfica con una precisión que cuente las secuencias cuyo valor predicho y el deseado difieran menos de un 5%
  - Repetir todo esto para una LSTM y una GRU, cada una con 2 capas ocultas de 5 neuronas
  - El histórico de datos de normalizará tomando como valor mínimo el de una acción (0,00€) y un máximo, en este caso de 15€, asumiendo que estas redes no pueden predecir más allá de este valor.
  - En todas las redes de esta práctica, se tomará la tanh como activación de la capa oculta y la de salida igual a la lineal.
  - Tasa de aciertos en función del error relativo entre predicción y salida (margen = 2% ~ **5**%).

Gráficas de Resultados