Redes Recurrentes en Keras

Gonzalo Uribarri • Gabriel B. Mindlin

Sistemas dinámicos e inteligencia artificial aplicados al modelado de datos

@gonzauri

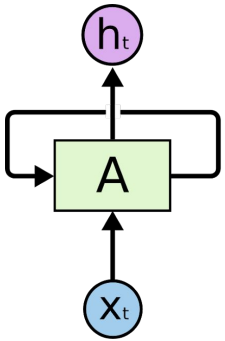
Objetivos de la práctica de hoy

- Conocer que existen distintos tipos de redes recurrentes.
- Implementar una red Recurrente en Keras.
- Visualizar y evaluar las predicciones de una red recurrente.
- Comparar esta nueva arquitectura con la de una red densa para predicción en series temporales

Redes Recurrentes

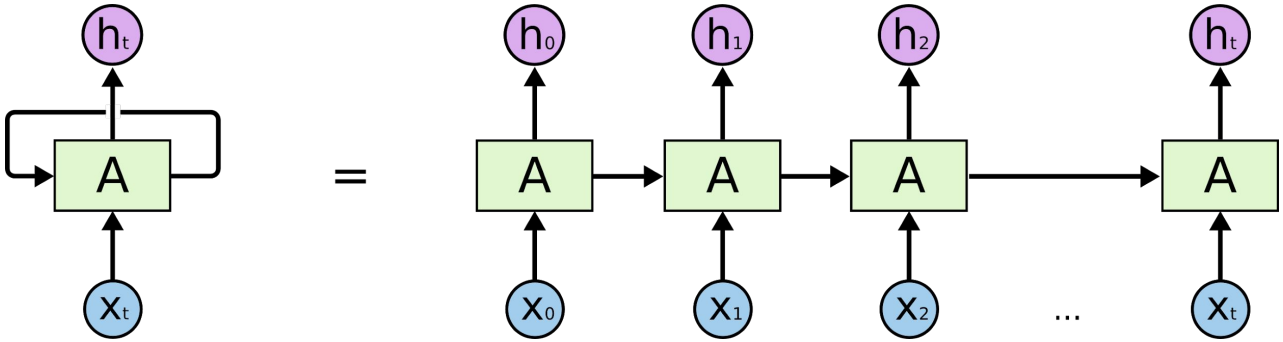
Redes Recurrentes

Son redes donde la salida de la red para un dado paso temporal se utiliza como entrada de la red para el siguiente paso temporal.



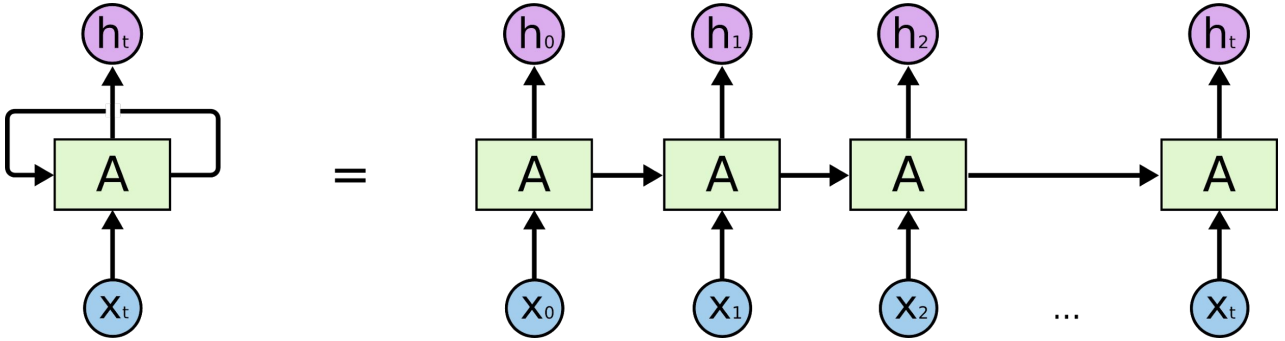
Redes Recurrentes

Son redes donde la salida de la red para un dado paso temporal se utiliza como entrada de la red para el siguiente paso temporal.

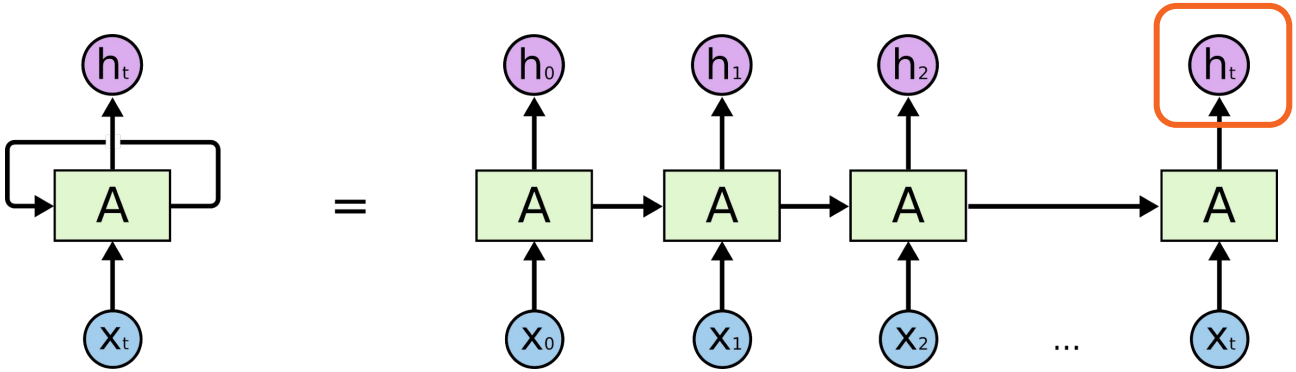


La red posee un estado interno (hidden state). Esta información es la que se pasa al siguiente paso y de donde se lee la salida.

Redes Recurrentes

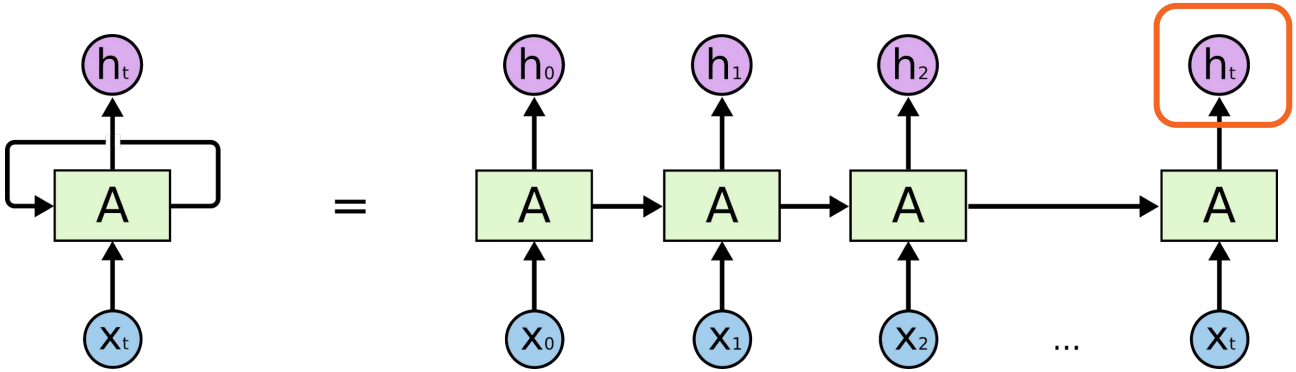


Redes Recurrentes



Normalmente tomamos como salida el último hidden state, pero también se puede consultar el hidden state en cada paso.

Redes Recurrentes



Normalmente tomamos como salida el último hidden state, pero también se puede consultar el hidden state en cada paso.

RNNs, once unfolded in time, can be seen as very deep feedforward networks in which all the layers share the same weights.

— Deep learning, Nature, 2015

Redes Recurrentes

Existen muchos tipo distintos de redes recurrentes:

RNNs, LSTMs, GRUs, Echo State Networks, etc...

Redes Recurrentes

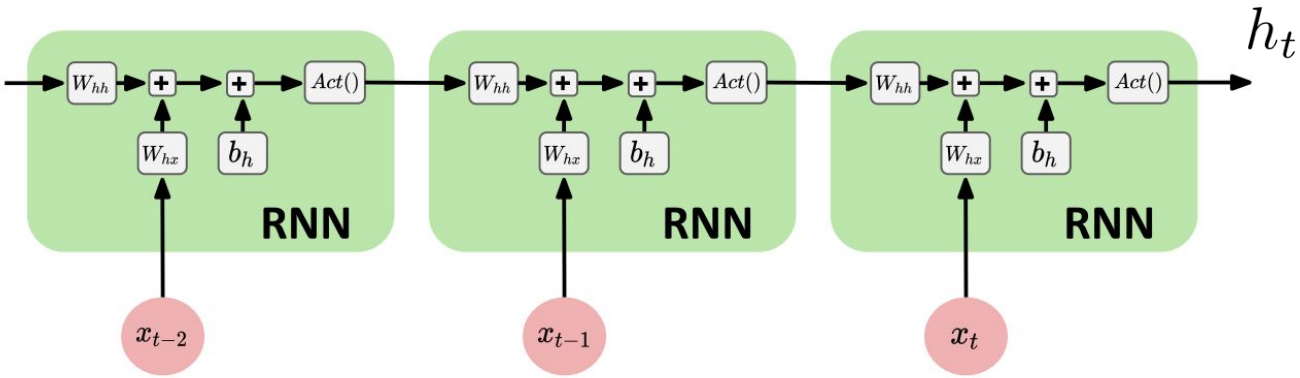
Existen muchos tipo distintos de redes recurrentes:

RNNs, LSTMs, GRUs, Echo State Networks, etc...

Vamos a comentar 2 tipos de redes recurrentes:

- 1: RNN Simple
- 2: LSTM

1) RNN Simple



Una matriz multiplica la entrada, una matriz multiplica el *hidden state* del paso anterior, se suman, y se aplica una activación de tipo *tanh*.

1) RNN Simple

Keras

```
model = Sequential()
model.add(SimpleRNN(5, activation='tanh', input_shape=(look_back, n_features)))
model.add(Dense(1,activation='linear'))
model.compile(optimizer='adam', loss='mse')
```

Importante: Los datos de entrada deben tener la forma:

X.shape = [num_samples, look_back , n_features]

1) RNN Simple

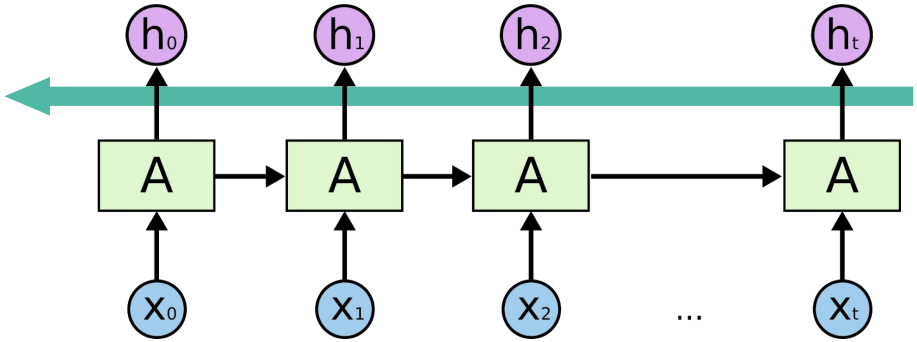
Vanishing gradient problem

Problema! Les cuesta aprender dependencias de largo plazo.

1) RNN Simple

Vanishing gradient problem

Problema! Les cuesta aprender dependencias de largo plazo.

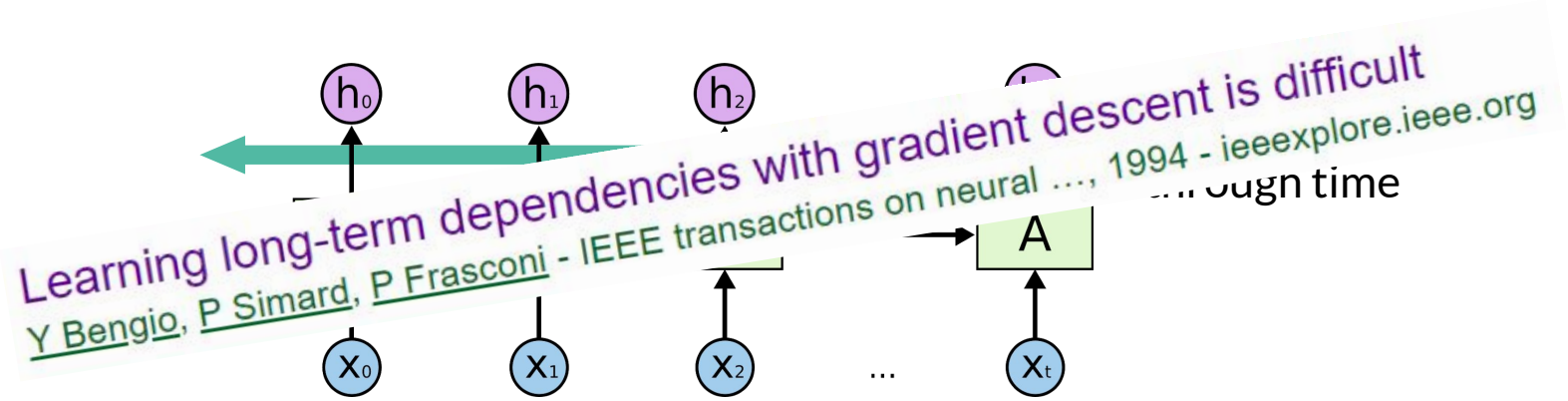


Backpropagation through time

1) RNN Simple

Vanishing gradient problem

Problema! Les cuesta aprender dependencias de largo plazo.



Learning long-term dependencies with gradient descent is difficult
Y Bengio, P Simard, P Frasconi - IEEE transactions on neural ..., 1994 - ieeexplore.ieee.org

2) LSTM

La idea es poder captar dependencias a largo plazo, para esto se agrega un nuevo estado interno de la red llamado “cell state”.

Long short-term memory

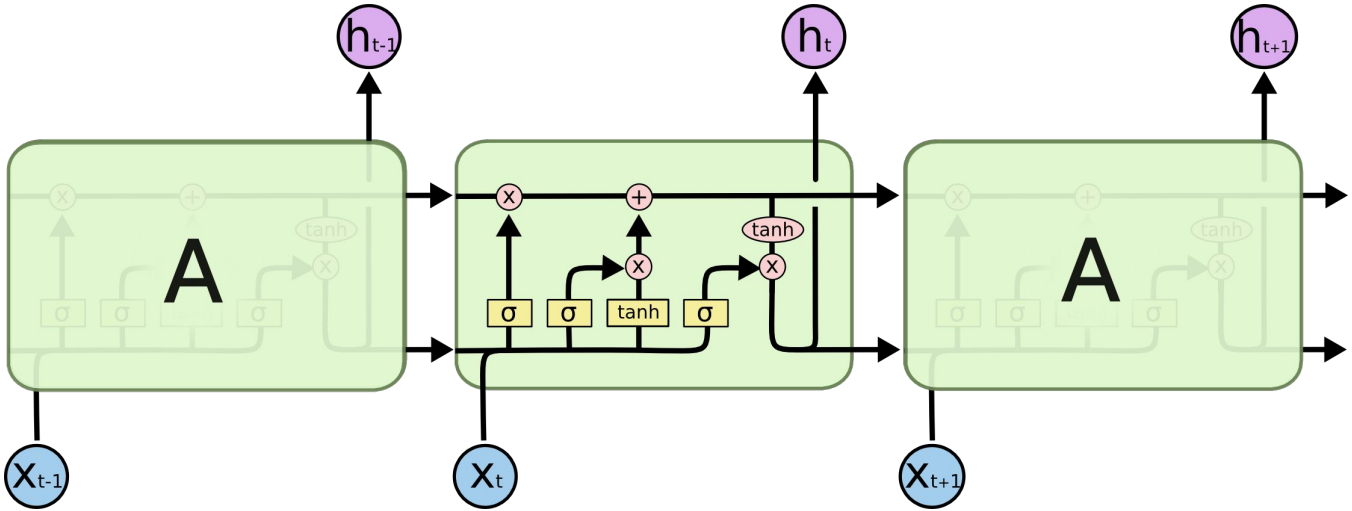
S Hochreiter, J Schmidhuber - Neural computation, 1997 - MIT Press

Learning to store information over extended time intervals by recurrent backpropagation takes a very long time, mostly because of insufficient, decaying error backflow. We briefly review Hochreiter's (1991) analysis of this problem, then address it by introducing a novel …

☆ 99 Citado por 40180 Artículos relacionados Las 47 versiones

2) LSTM

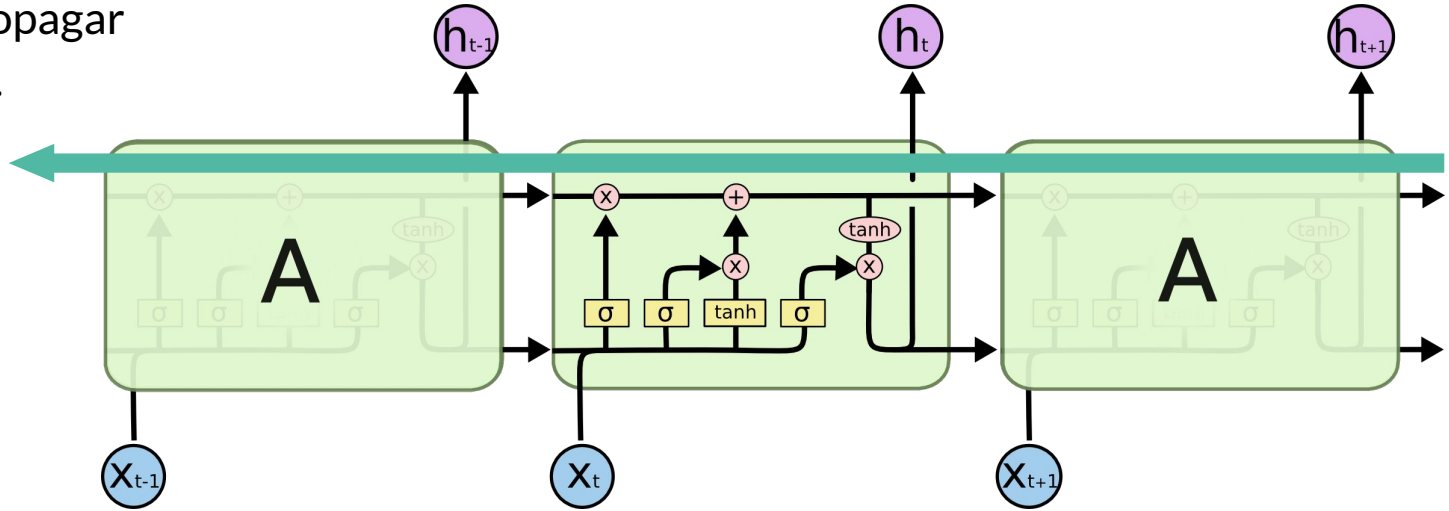
La idea es poder captar dependencias a largo plazo, para esto se agrega un nuevo estado interno de la red llamado “cell state”.



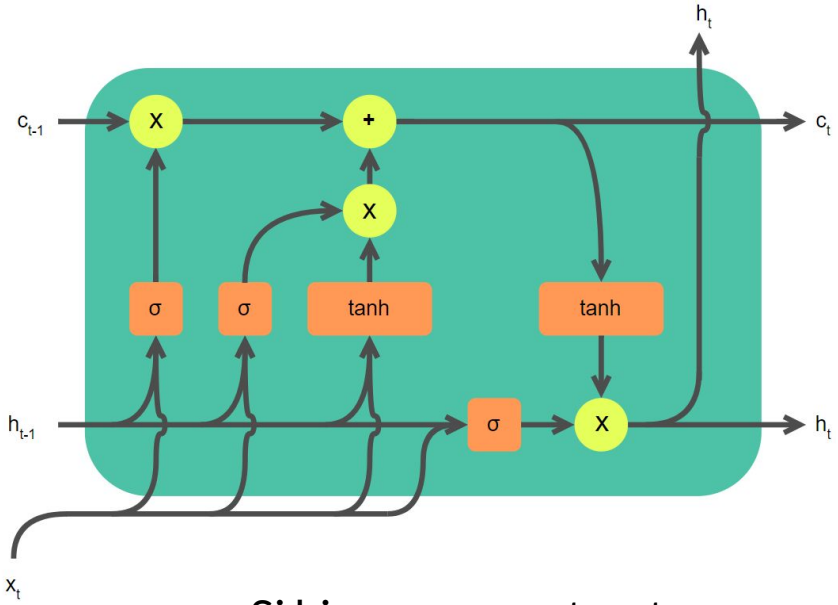
2) LSTM

La idea es poder captar dependencias a largo plazo, para esto se agrega un nuevo estado interno de la red llamado “cell state”.

Más fácil propagar el gradiente.



2) LSTM



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

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

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

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

$$h_t = o_t * \tanh\left(C_t\right)$$

Si bien es una estructura un poco más complicada, no dejan de ser matrices de pesos (a aprender) y activaciones no lineales.

2) LSTM

Keras

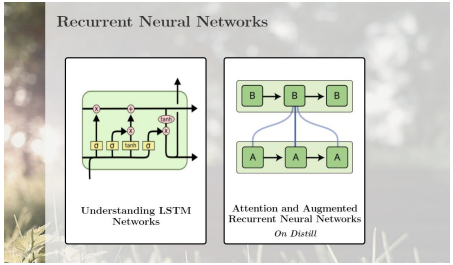
```
model = Sequential()
model.add(LSTM(5, activation='tanh', input_shape=(look_back, n_features)))
model.add(Dense(1,activation='linear'))
model.compile(optimizer='adam', loss='mse')
```

Recuerden siempre recurrir a la documentación:

https://keras.io/api/layers/recurrent_layers/lstm/

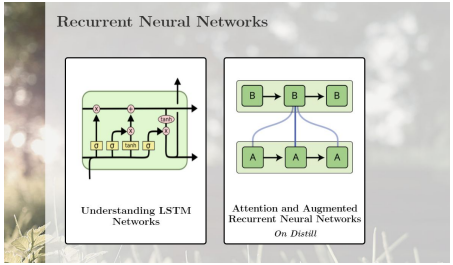
Para leer sobre RNNs

- Colah's Blog: http://colah.github.io/posts/2015-08-Understanding-LSTMs/



Para leer sobre RNNs

- Colah's Blog: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

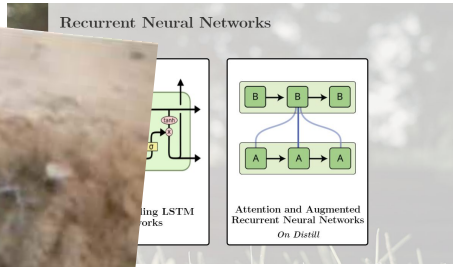


- Videos de Andrew NG: https://www.youtube.com/watch?v=2E65LDnM2cA&list=PL1w8k37X_6L_s4ncq-swTBvKDWnRSrinI&index=4&ab_channel=KnowledgeCenter

Para leer sobre RNNs

- Colah's Blog: http://colah.gitl 5-08-Understa

- Videos de An https://www.youtu nM2cA&list=PL1v DWnRSrinI&index geCenter



Trabajo en el Notebook

Link

https://drive.google.com/file/d/1rk2LXMadVXT3-yfnLq-eruveKNtsuDUU/view?usp=sharing

Breakout Rooms

Trabajo en forma grupal, uno comparte pantalla. Pueden llamarme a la sala en cualquier momento.

Puesta en común y Conclusiones

(15 minutos) Comentarios sobre el trabajo realizado.

RNNs en comparación a redes densas

RNNs vs Densas

Hay dos ventajas importantes a la hora de trabajar series temporales con RNNs:

- 1: Número de Parámetros
- 2: Forma de los datos (Inputs y outputs)

1) Menos parámetros

- En una red densa, la cantidad de pesos crece con el tamaño de la secuencia de entrada. En una RNN no.
- Se intenta explotar cierta invariancia temporal en las operaciones.
- Se vuelve más importante si la secuencia de entrada x tiene una dimensión no despreciable.

1) Forma de los Datos

- Una RNN puede procesar inputs de distinto largo.
- Las RNN también se pueden entrenar de manera continua (statefull). **Ojo!** Muchas veces dificulta la convergencia del entrenamiento.
- Si fuese necesario, podemos obtener la salida para cada tiempo:

```
model.add(LSTM(5, activation = 'tanh', input_shape = (look_back,n_features),
                                return_sequences = True))
```

RNNs en arquitecturas más complejas

RNNs con densas a la salida

Permitimos una lectura más compleja del estado final de la red.

Keras

```
model = Sequential()
model.add(SimpleRNN(10, activation='tanh', input_shape=(look_back, n_features)))
model.add(Dense(8,activation='relu'))
model.add(Dense(4,activation='relu'))
model.add(Dense(1,activation='relu'))
model.compile(optimizer='adam', loss='mse')
```

Stacked RNNs