Universidad Autónoma de Madrid

Escuela Politécnica Superior

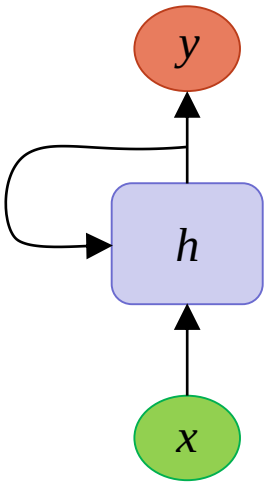
Redes recurrentes II

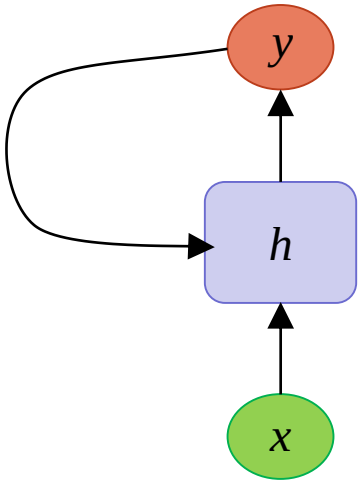
Redes recurrentes vistas como redes profundas

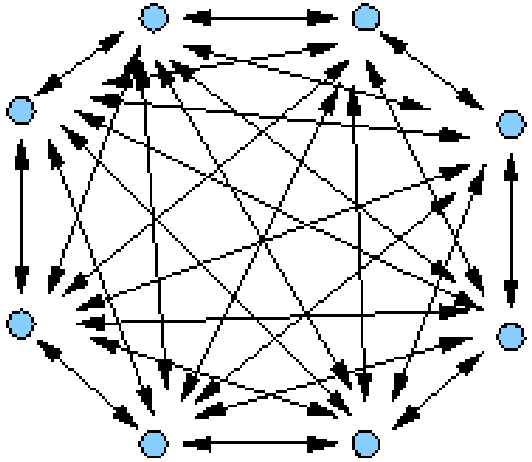
- Las redes recurrentes (RNN) se pueden caracterizar como redes profundas cuya arquitectura contempla **conexiones recurrentes (en ciclo cerrado) de una capa a sí misma o de otra capa a una capa anterior.**

- **La información circula de forma recurrente** en estos ciclos cerrados, en vez de propagarse hacia adelante como en las redes feedforward.

- Por su diseño estas redes **pueden procesar mejor entradas secuenciales o con una codificación temporal** que preserva el orden temporal.





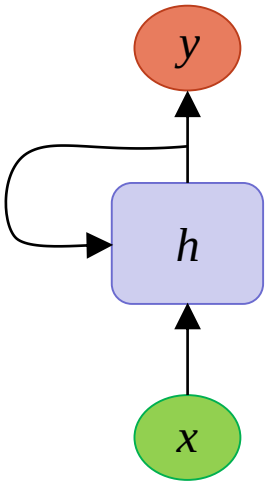


Formulaciones clásicas

Formulación de Elman (1990):

- La recurrencia es de una capa oculta a sí misma:

$$h_t = f\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right)$$
$$y_t = f\left(W_{hy}h_t + b_y\right)$$



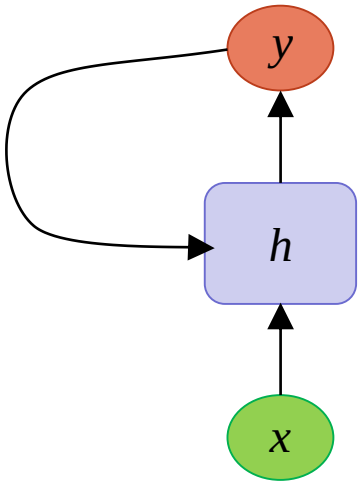
Formulaciones clásicas

Formulación de Jordan (1997):

- La recurrencia es de una capa de salida a la capa oculta:

$$h_t = f\left(W_{xh}x_t + W_{yh}y_{t-1} + b_h\right)$$

$$y_t = f\left(W_{hy}h_t + b_y\right)$$



Implementación en Keras

Definición del modelo:

•stateful=False: indica que no es necesario pasar el último estado al siguiente batch.

•return_sequences=False: indica que no es necesario tener como salida los estados en todos los pasos de tiempo, solo en el último.

```
model = Sequential()

model.add(SimpleRNN(10, activation="tanh",
input_shape=(seq_len, 1), return_sequences=False,
stateful=False, unroll=True))

model.add(Dense(1, activation='sigmoid'))

print(model.summary())
plot_model(model, show_shapes=True, show_layer_names=True)
```

Implementación en Keras

Compilación del modelo:

```
model.compile(loss='binary_crossentropy',
optimizer=keras.optimizers.Adam(), metrics=['accuracy'])
```

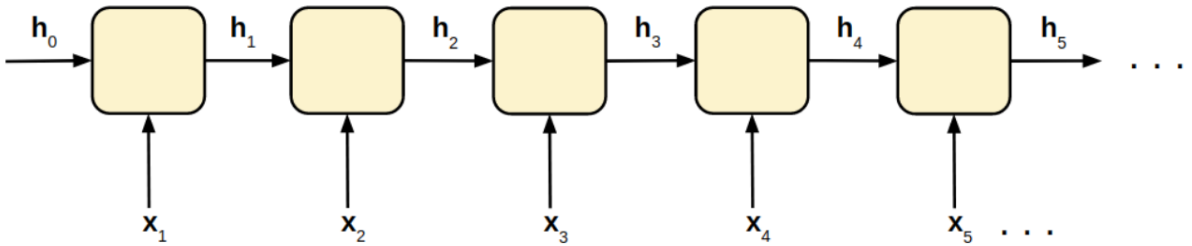
Entrenamiento:

```
history = model.fit(x[:, :, None], t[:, None], epochs=200,
batch_size=1000, validation_data=(xval[:, :, None],
tval[:, None]))
```

Retropropagación a lo largo del tiempo

- Si consideramos el modelo de Elman, El estado de la red h_t depende del estado en el paso de tiempo anterior, h_{t-1}, y a su vez este depende de h_{t-2}, etc.

- Por tanto podemos representar la recurrencia de forma que tenemos una copia diferente de la red en cada paso del tiempo y el estado pasa de un paso de tiempo al siguiente:



- El estado de la red h_t y por tanto y_t depende de la historia previa (todas las entradas y estados anteriores). Como todas las "copias" de la red comparten los mismos parámetros, cuando se calcula los gradientes de la función de error con respecto a los pesos de las capas ocultas, tenemos que **retropropagar el error para un número arbitrario de pasos de tiempo**.
- Ya que esto no es práctico, normalmente se fija una ventana temporal de pasos de tiempo y no se propaga el error más (**retropropagación truncada en el tiempo**). El estado de la red sí se puede pasar de forma continua de un batch al siguiente y, por tanto, la red lleva la cuenta de lo que ha ocurrido hace mucho.

Redes recurrentes sin problemas con el gradiente

- Para solucionar el problema de los gradientes decrecientes (*vanishing gradients*) se han propuesto distintas arquitecturas: redes Long short-term memory (LSTM), Gated recurrent unit (GRU), etc.

$$
\begin{aligned}
\mathbf{f}_t &= \sigma(W_f[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{b}_f), \\
\mathbf{i}_t &= \sigma(W_i[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{b}_i), \\
\tilde{\mathbf{c}}_t &= \tanh(W_c[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{b}_c), \\
\mathbf{c}_t &= \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \tilde{\mathbf{c}}_t, \\
\mathbf{o}_t &= \sigma(W_o[\mathbf{x}_t; \mathbf{h}_{t-1}] + \mathbf{b}_o), \\
\mathbf{h}_t &= \mathbf{o}_t \circ \tanh(\mathbf{c}_t).
\end{aligned}
$$

$[\mathbf{x}_t; \mathbf{h}_{t-1}]$ representa la concatenación de vectores en las redes LSTM