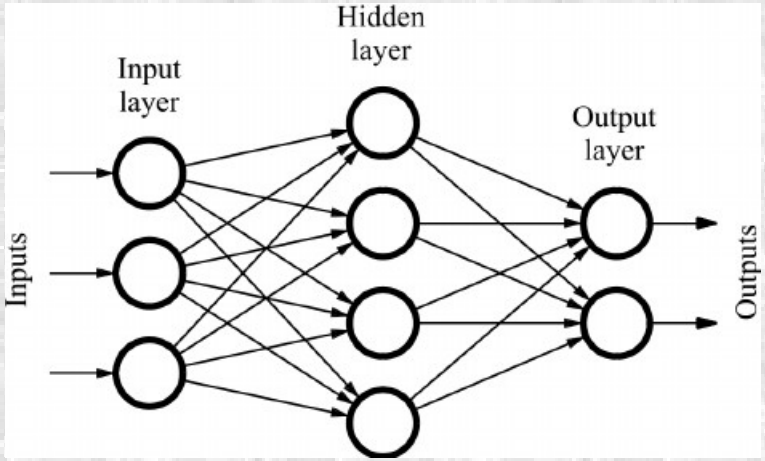
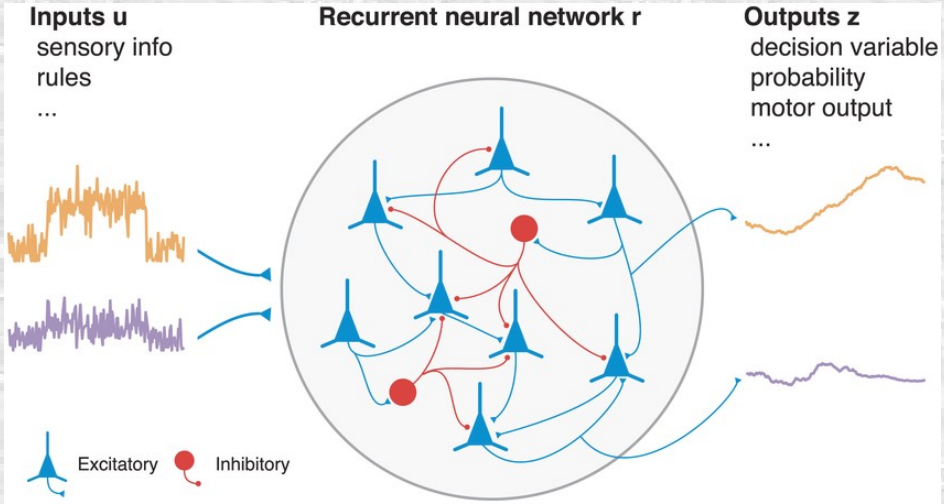
Redes Neuronales Recurrentes

Las neuronas pueden estar conectadas de diferentes maneras (arquitecturas)

Feed-forward



Recurrente



Redes Neuronales Recurrentes

Las redes neuronales *feed-forward* son utilizadas para implementar relaciones entrada-salida fijas pero no capturan fenómenos dinámicos

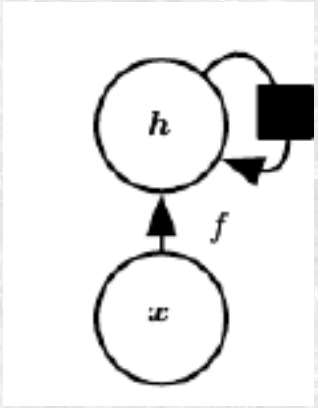
Muchos fenómenos están caracterizados mejor por sistemas dinámicos tanto deterministas como estocásticos

- Series temporales, naturales o artificiales
  - Registros EEG o ECG
  - Datos económicos
- Producción y análisis de lenguaje
- Secuencias genéticas
- Estructura de proteínas

Redes Neuronales Recurrentes

La manera de incorporar dinámica es usando *recurrencia*, esto es el estado de la neurona depende del estado de la misma neurona o de otras neuronas de la misma capa a tiempo anterior.

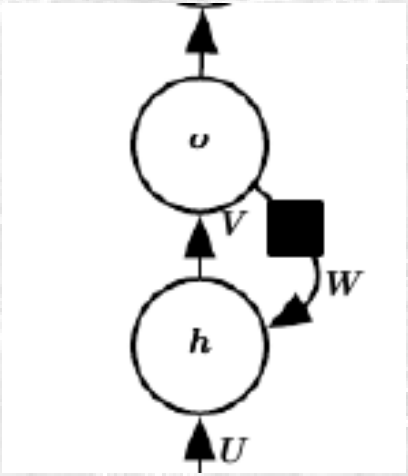
En caso mas simple es:



Donde $h^{(t)}=f(h^{(t-1)},x^{(t)};\theta)$
h es la salida, x es la entrada, θ son los parámetros y f es alguna función entrada-salida

Redes Neuronales Recurrentes

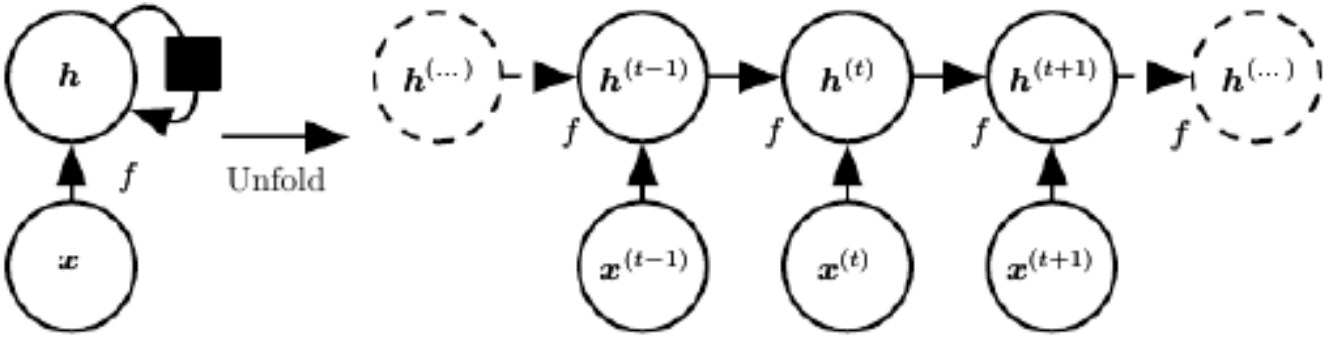
La recurrencia también puede ir a otra neurona (o a otras capas). Por ejemplo:



En sistemas que tienen este tipo de conexiones no se puede aplicar *back-propagation* en la manera usual.
De hecho el concepto de relación entrada-salida y función de costo a minimizar no está del todo bien definido.

Redes Neuronales Recurrentes

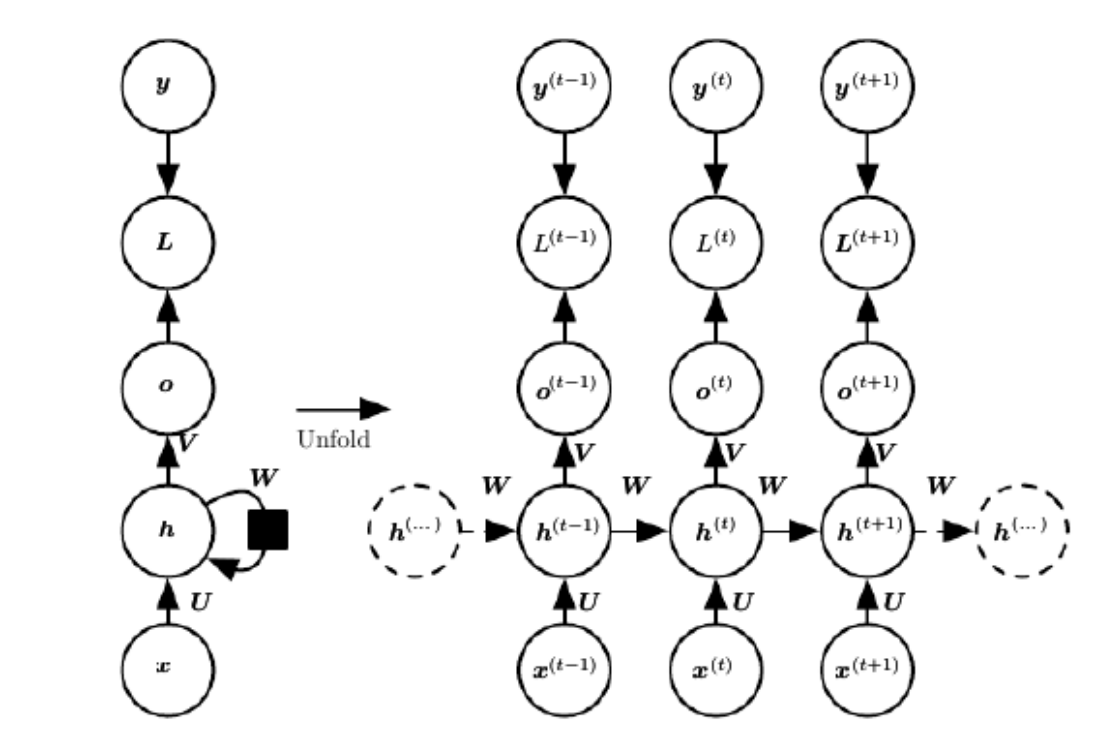
Para definir que hace la red de manera precisa la *desenrollamos (unfolding):*



Es decir representamos cada paso temporal como si ocurriera en una red diferente, pero que comparten los valores de las conexiones (comparar con redes convolucionales)

Redes Neuronales Recurrentes

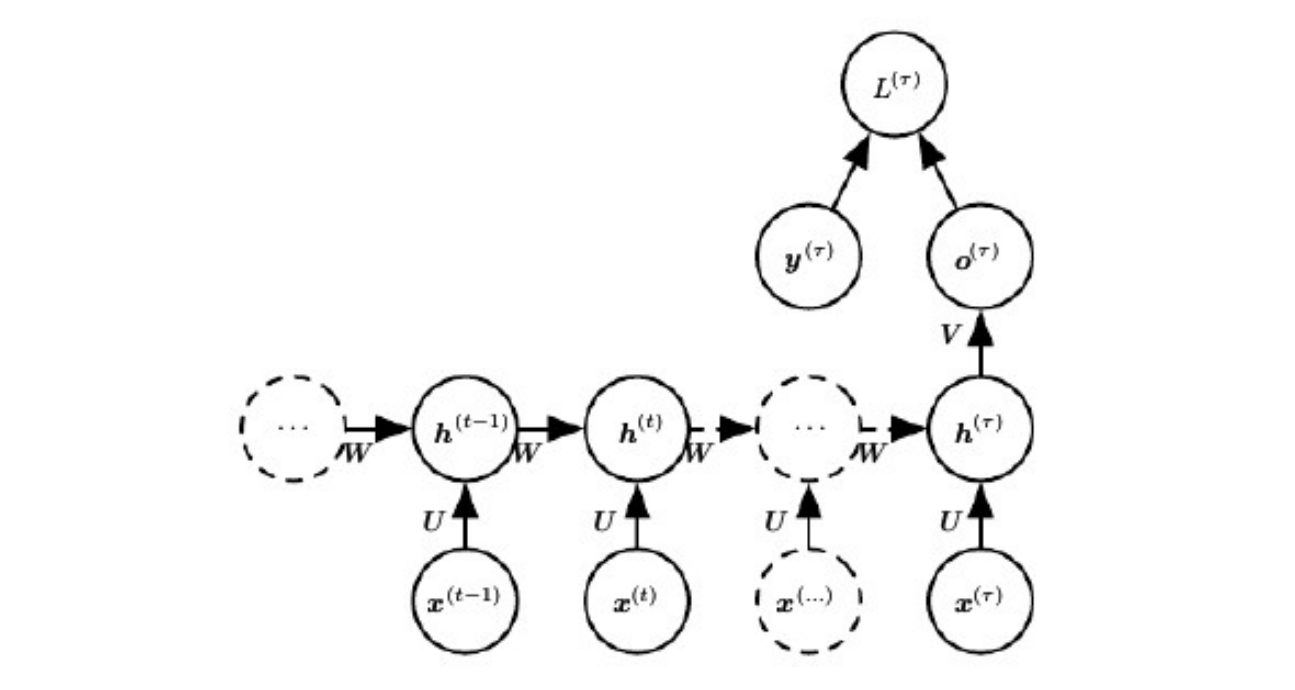
Estos da lugar a diferentes tipos de arquitecturas posibles, que resuelven diferentes tipos de problemas:



Entrada: secuencia; salida: secuencia (traducción)

Redes Neuronales Recurrentes

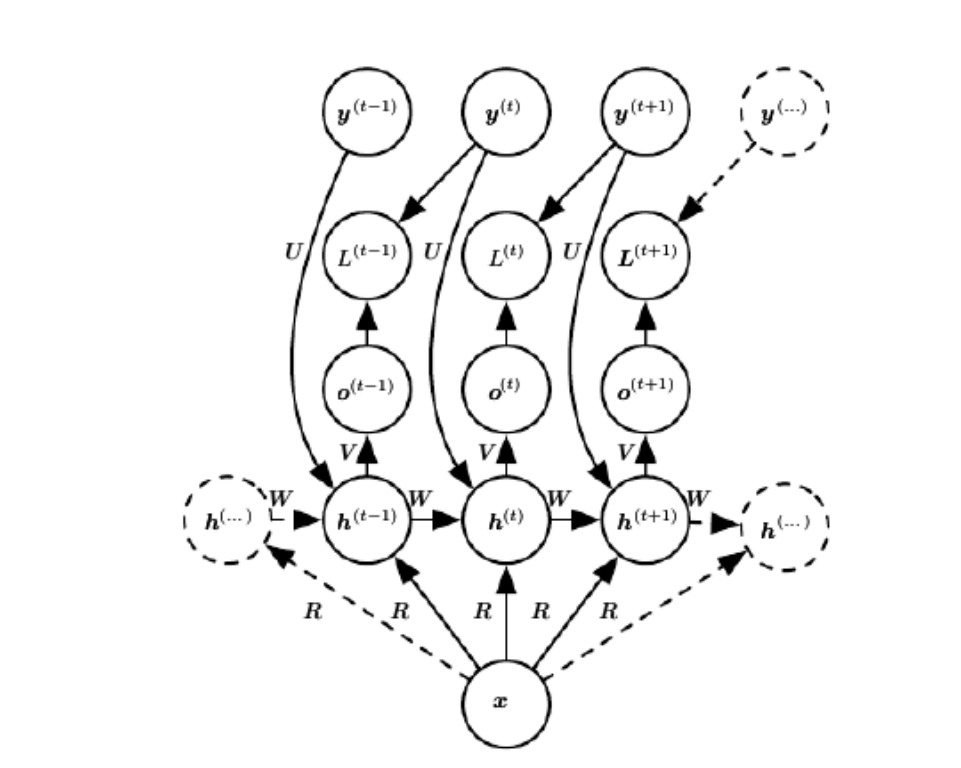
Estos da lugar a diferentes tipos de arquitecturas posibles, que resuelven diferentes tipos de problemas:



Entrada: secuencia; salida: valor constante
(clasificación series temporales)

Redes Neuronales Recurrentes

Estos da lugar a diferentes tipos de arquitecturas posibles, que resuelven diferentes tipos de problemas:

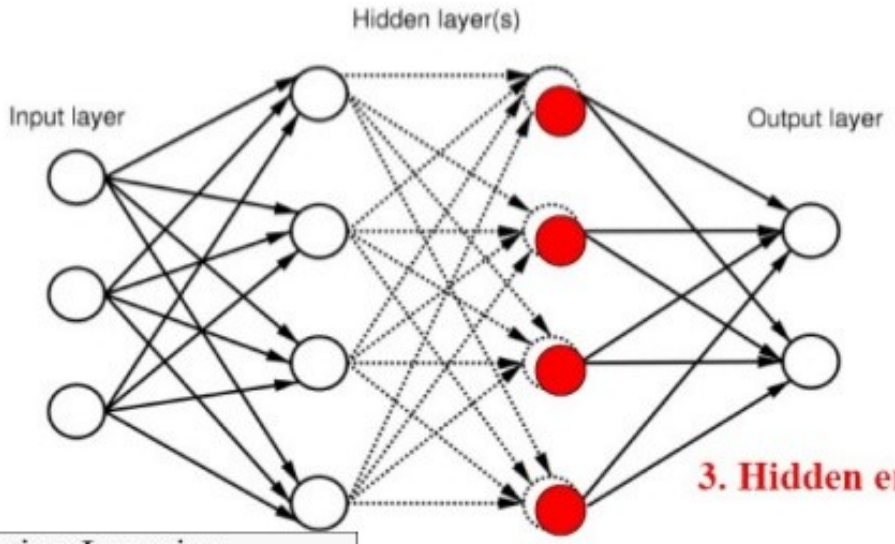


Entrada: valor constante; salida: secuencia
(captioning de imágenes*)*

Redes Neuronales Recurrentes

Método de retropropagación de errores (*back-propagation)*

Backpropagation Learning



10

Backpropagation Learning

$$E_{out\,i} = d_{out\,i} - out_i$$

$$E_{total} = \sum_{i=0}^{num(n_{out})} E_{out\,i}^2$$

$$E_{hid\,i} = \sum_{k=1}^{num(n_{out})} E_{out\,k} \cdot w_{out\,i,k}$$

$$diff_{hid\,i} = E_{hid\,i} \cdot (1 - o(n_{hid\,i})) \cdot o(n_{hid\,i})$$

Redes Neuronales Recurrentes

Método de retropropagación de errores (*back-propagation)*

1-Conjunto de datos entrada salida: $X=\{(\mathbf{x}_i,\mathbf{y}_i)\}$ $(i=1,\ldots,P)$

2-Conjunto de parámetros de la red: Θ_k (pesos *w* y umbrales)

3-Función error: $E(X,\Theta)$

Por ejemplo: $E(X,\Theta) = (1/2P)\ \Sigma_i\ |\mathbf{y'}_i(\mathbf{x}_i,\Theta)-\mathbf{y}_i|^2$

donde $y'_i(x_i,\Theta)$ es la salida **observada** cuando la entrada es $\mathbf{x}_i$ y $\mathbf{y}_i$ es la salida **deseada**

El error se puede minimizar haciendo

$\Delta\theta_k = -\eta\ \partial E/\partial\theta_k$

Redes Neuronales Recurrentes

Método de retropropagación de errores (*back-propagation)*

La salida de una neurona esta dada por

$$y'_l = f(h_l) = f(\Sigma_j w_{lj}\, y'_j)$$

Para las conexiones que llegan a la capa de salida

$$\partial E/\partial w_{lj} \approx (y'_l - y_l)\, f'(h_l)\, y'_j$$

$$= \delta_l\, y'_j$$

Se puede probar, utilizando la regla de la cadena, que para cualquier parámetro de pesos se tiene que

$$\partial E/\partial w_{ij} \approx \delta_i\, y'_j$$

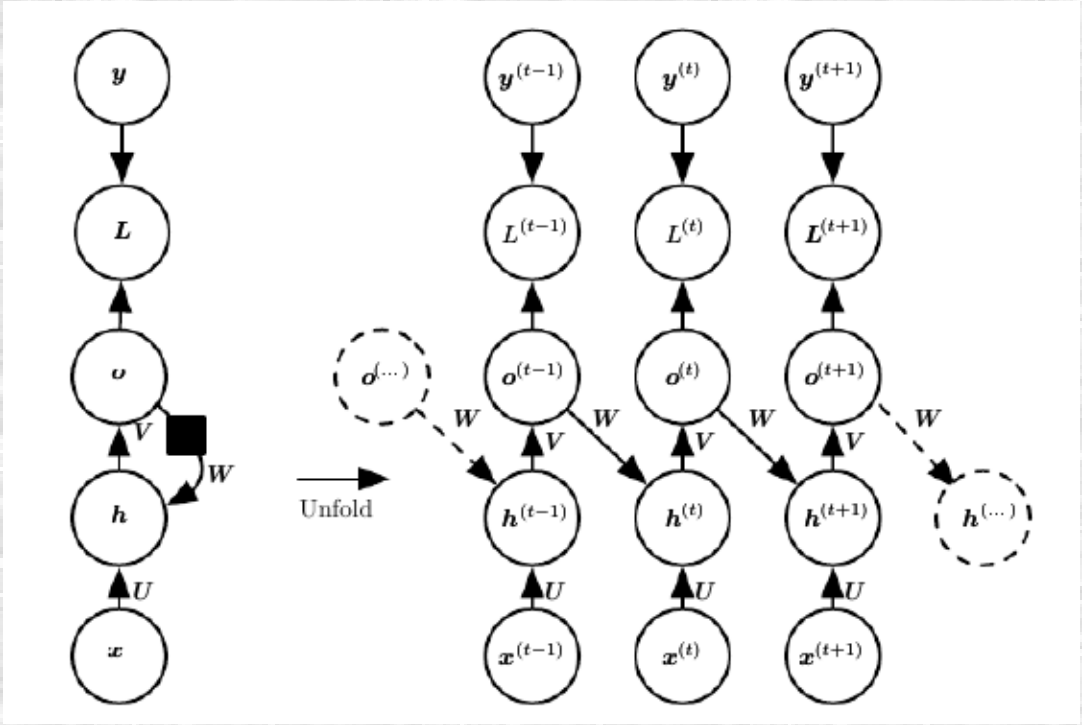
donde

$$\delta_i = f'(h_i)\, \Sigma_k\, \delta_k\, w_{ki}$$

Redes Neuronales Recurrentes

Aplicando esta idea a redes recurrentes desenrolladas se obtiene el método de *back-propagation in time:*

Consideremos la red



Redes Neuronales Recurrentes

La función de costo total será

$L = \Sigma_t L^{(t)}$ (o $L=L^{(\tau)}$ donde τ es el tiempo final)

Tenemos que tomar la derivada de L respecto a cada uno de los pesos (y bias), es decir U, V, W, etc.

Si las ecuaciones de *update* son

$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}, \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}), \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}, \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)}),
\end{aligned}
$$

Redes Neuronales Recurrentes

Las derivadas de la función de costo en respecto a los pesos se puede escribir primero en función de las derivadas respecto a los estados de las neuronas:

$$
\begin{aligned}
\nabla_{\boldsymbol{V}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}} \right) \nabla_{\boldsymbol{V}^{(t)}} o_i^{(t)} = \sum_t \left(\nabla_{\boldsymbol{o}^{(t)}} L \right) \boldsymbol{h}^{(t)\top}, \\
\nabla_{\boldsymbol{W}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}} \right) \nabla_{\boldsymbol{W}^{(t)}} h_i^{(t)} \\
&= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)} \right)^2 \right) \left(\nabla_{\boldsymbol{h}^{(t)}} L \right) \boldsymbol{h}^{(t-1)\top}, \\
\nabla_{\boldsymbol{U}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}} \right) \nabla_{\boldsymbol{U}^{(t)}} h_i^{(t)} \\
&= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)} \right)^2 \right) \left(\nabla_{\boldsymbol{h}^{(t)}} L \right) \boldsymbol{x}^{(t)\top},
\end{aligned}
$$

Lo mismo para los bias:

Redes Neuronales Recurrentes

Lo mismo para los bias:

$$\nabla_{\boldsymbol{c}} L = \sum_t \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{c}} \right) \nabla_{\boldsymbol{o}^{(t)}} L = \sum_t \nabla_{\boldsymbol{o}^{(t)}} L,$$

$$\nabla_{\boldsymbol{b}} L = \sum_t \left(\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{b}^{(t)}} \right)^{\top} \nabla_{\boldsymbol{h}^{(t)}} L = \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)} \right)^2 \right) \nabla_{\boldsymbol{h}^{(t)}} L$$

Redes Neuronales Recurrentes

En el cálculo de las derivadas respecto a la salida de las neuronas entra la recurrencia.
Por ejemplo, la derivada respecto a $h^{(t)}$:

$$\nabla_{\boldsymbol{h}^{(t)}} L = \left(\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{h}^{(t)}} \right)^{\top} (\nabla_{\boldsymbol{h}^{(t+1)}} L) + \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}} \right)^{\top} (\nabla_{\boldsymbol{o}^{(t)}} L)$$
$$= \boldsymbol{W}^{\top} \text{diag} \left(1 - \left(\boldsymbol{h}^{(t+1)} \right)^2 \right) (\nabla_{\boldsymbol{h}^{(t+1)}} L) + \boldsymbol{V}^{\top} (\nabla_{\boldsymbol{o}^{(t)}} L) ,$$

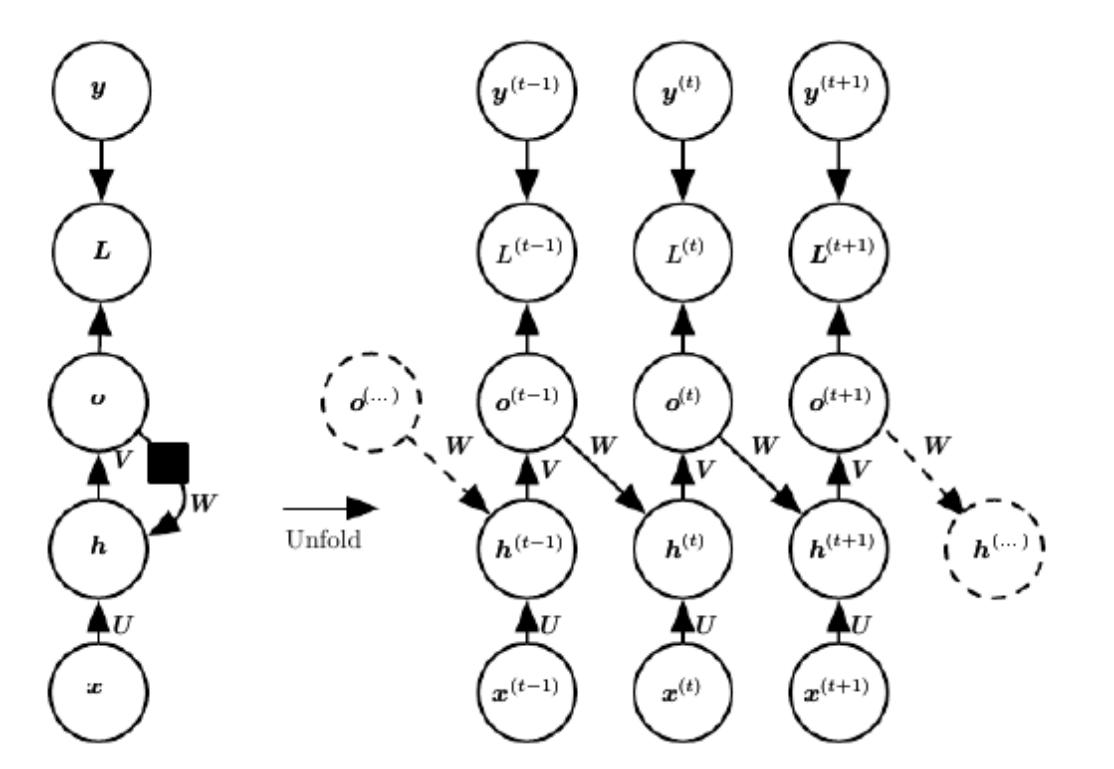
depende de la derivada respecto a $h^{(t+1)}$

Si la dinámica solo “corre” hasta un tiempo final τ, la derivada respecto a $h^{(\tau)}$ solo incluye en segundo término, que es conocido. Esto permite iniciar la recurrencia hacia atrás en tiempo.

Redes Neuronales Recurrentes

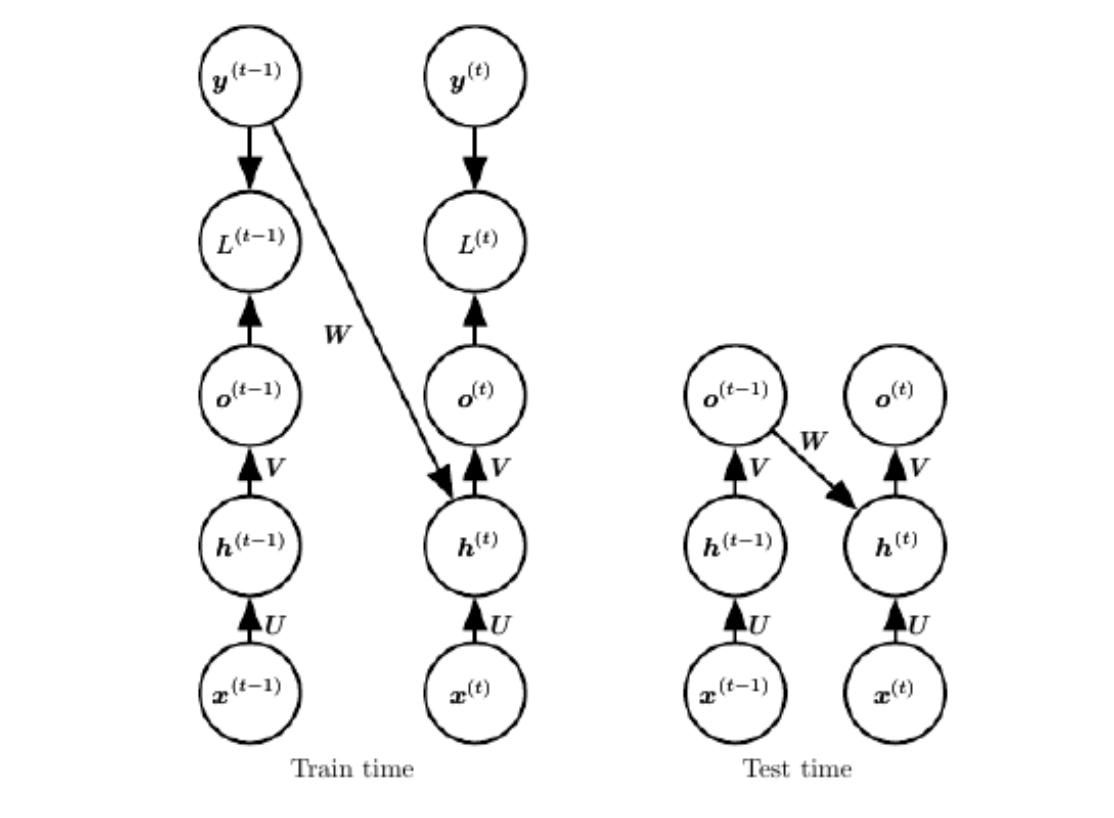
El tiempo de cálculo es O(τ) y no se puede paralelizar en tiempo.

Si el feedback solo sale de la unidad de salida



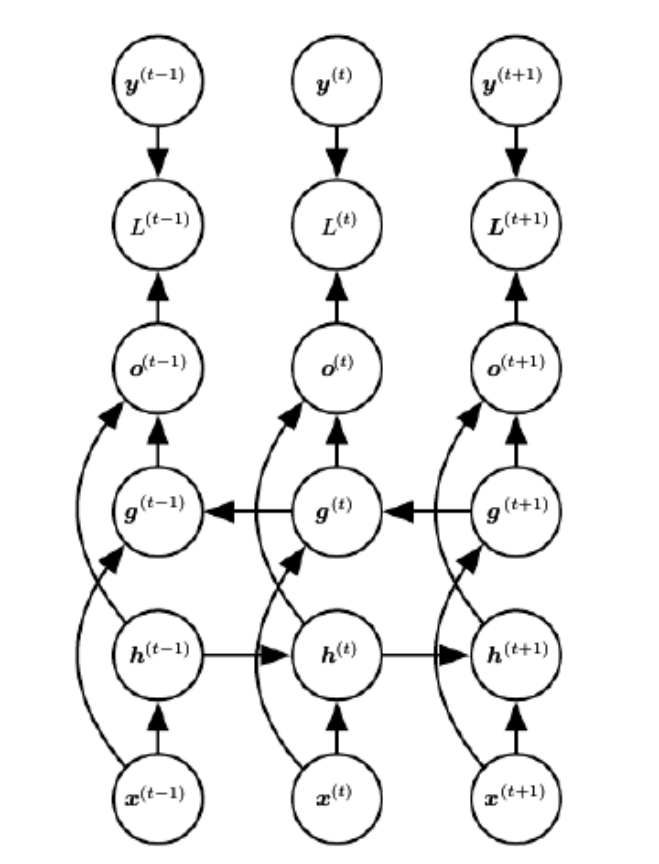
Redes Neuronales Recurrentes

Se puede utilizar *teacher forcing* (es decir reemplazar el feedback real $\sigma^{(t)}$ por la salida deseada $y^{(t)}$)



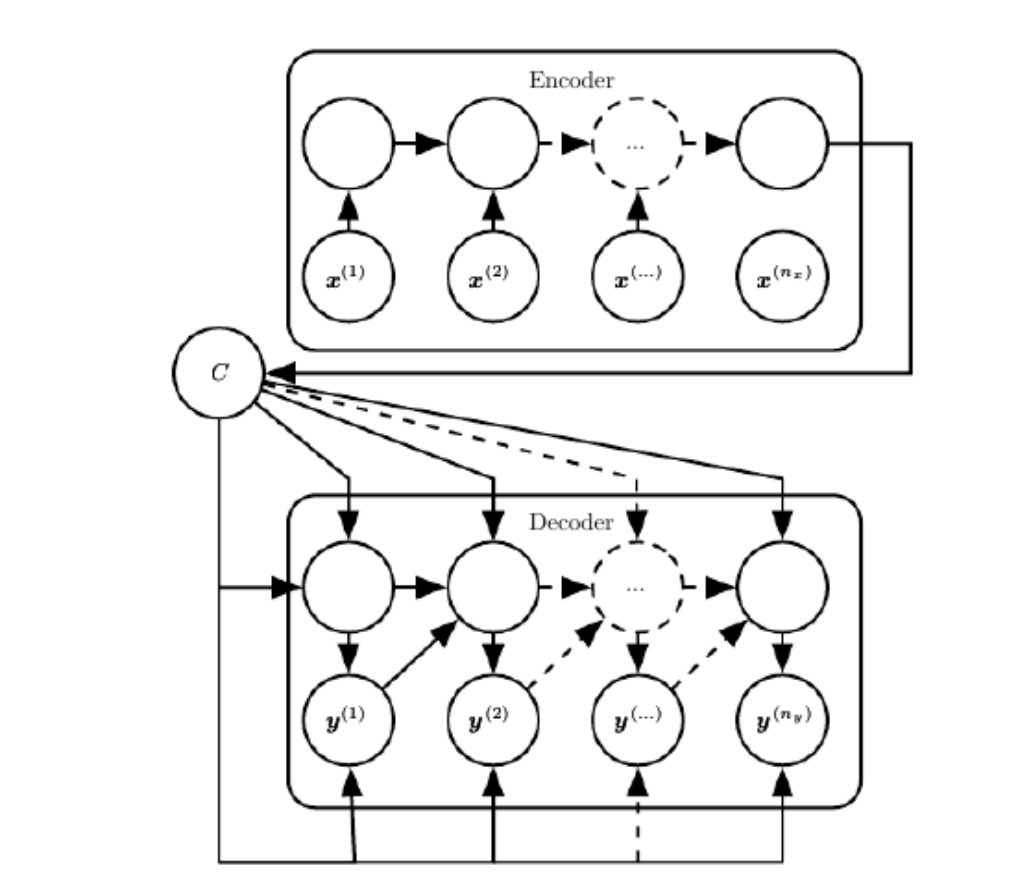
Redes Neuronales Recurrentes

Otras arquitecturas: Redes bidireccionales



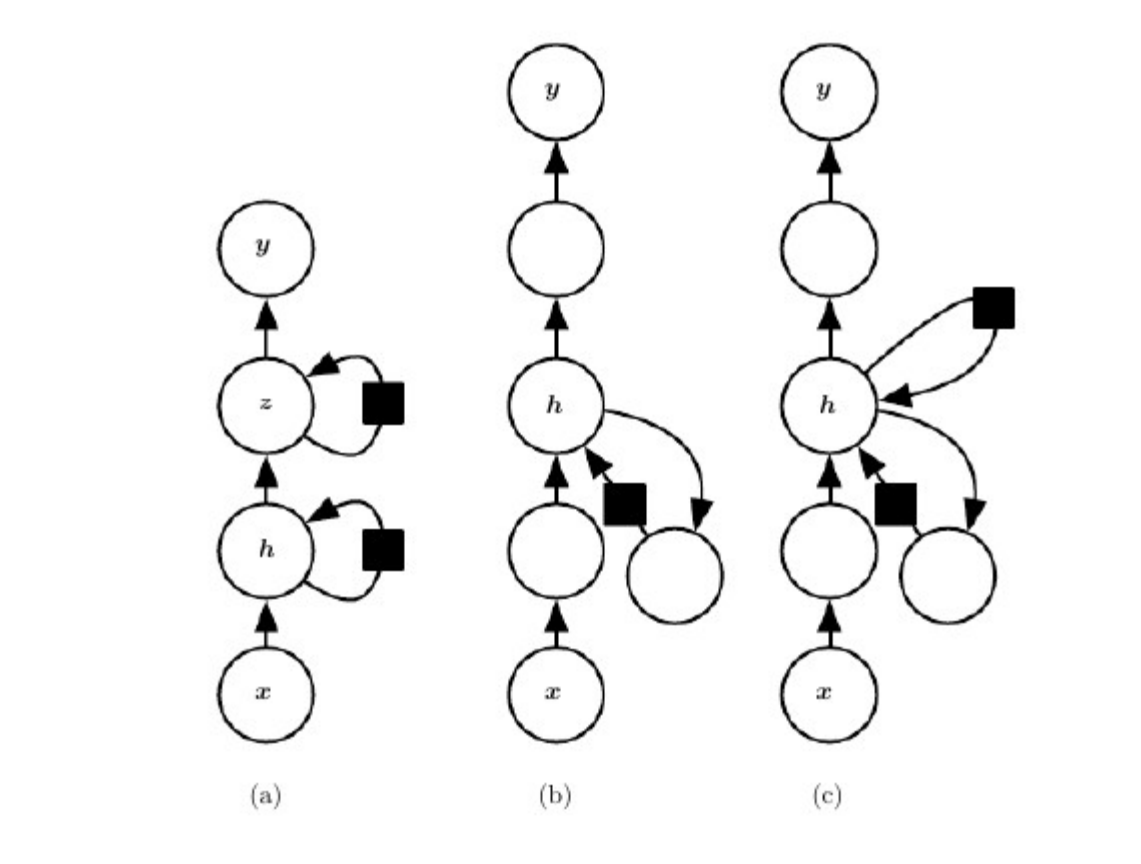
Redes Neuronales Recurrentes

Otras arquitecturas: *Encoder-Decoder*



Redes Neuronales Recurrentes

Otras arquitecturas: *Deep Recurrent NNs*



Redes Neuronales Recurrentes

La aplicación de *back-propagation* en una desenrollada inducirá el mismo problema de gradientes que se anulan (o explotan) que en una red profunda

Esto es un problema cuando hay múltiples escalas de tiempo involucradas

Soluciones propuestas:

-Saltear conexiones

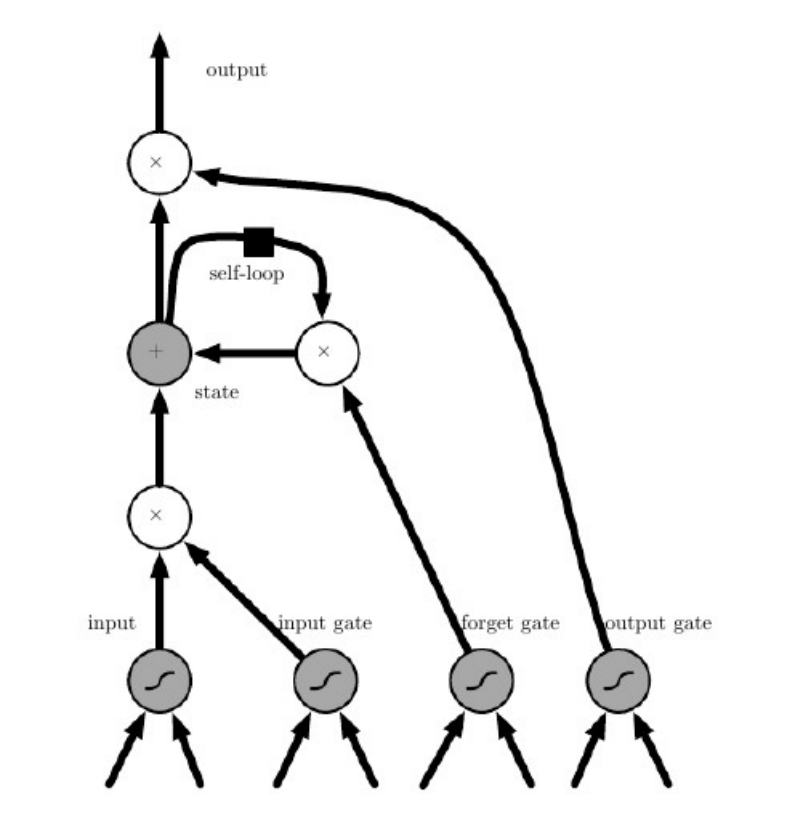
-Neuronas con fuga (*leaky)*

-*Gating*

Redes Neuronales Recurrentes

LSTM (Long Short Term Memory)

Esta es una capa “gatillada” y que autocontrola su constante de tiempo efectiva



Redes Neuronales Recurrentes

LSTM (Long Short Term Memory)

Mecanismo de “olvido”

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma \left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)} \right),$$

El valor de $f_i^{(t)}$ controla la escala de tiempo del loop de feedback

Si $f_i^{(t)} << 1$ se olvida rápidamente, la historia no interesa.

Si $f_i^{(t)} = O(1)$ la historia influye un tiempo muy largo

El valor de $f_i^{(t)}$ se calcular usando:

$$f_i^{(t)} = \sigma \left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)} \right),$$

Redes Neuronales Recurrentes

LSTM (Long Short Term Memory)

Tanto las matrices U, W como U^f, W^f se pueden aprendar usando *back-propagation.*

Similarmente se puede controlar el flujo de información de entrada usando las variables de gatillado:

$$g_i^{(t)} = \sigma\left(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}\right).$$

Redes Neuronales Recurrentes

LSTM (Long Short Term Memory)

La salida también se puede gatillar:

$$h_i^{(t)} = \tanh\left(s_i^{(t)}\right) q_i^{(t)},$$

$$q_i^{(t)} = \sigma\left(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)}\right),$$

Redes Neuronales Recurrentes

LSTM (Long Short Term Memory)

Ahora veamos las aplicaciones y un ejemplo de código

example-lstm.py

Redes Neuronales Recurrentes

Classification of Cardiac Arrhythmias from Single Lead ECG
with a Convolutional Recurrent Neural Network

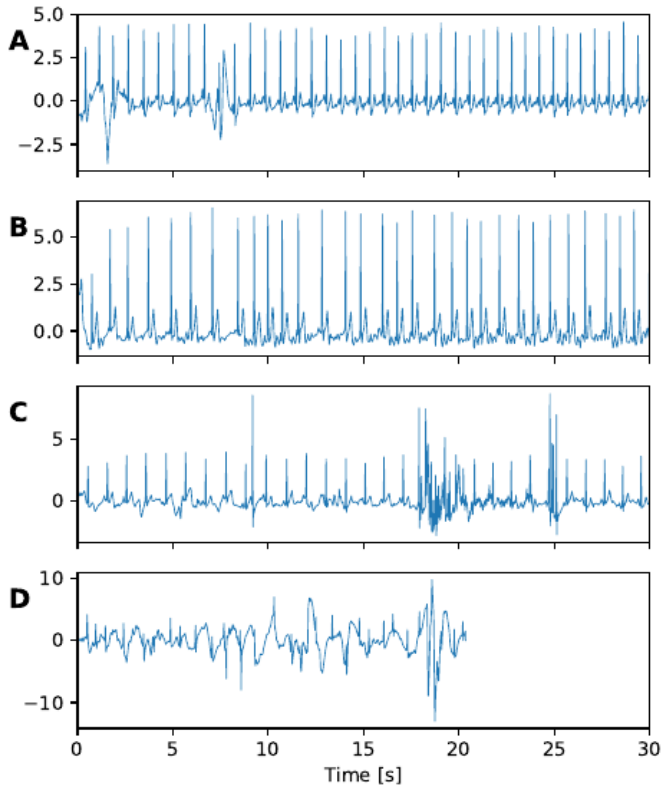


Figure 2: Examples of ECG records: (A) normal rhythm from record A00001, (B) atrial fibrillation from record A00004, (C) normal rhythm from record A00002, (D) atrial fibrillation from record A00015.

Table 1: Number of signals and mean duration for each class.

Class	Count	Proportion	Mean duration [s]
Normal rhythm	5076	59.52%	32.11
Atrial fibrillation	758	8.89%	32.34
Other rhythm	2415	28.32%	34.30
Noise	279	3.27%	24.38
Total	8528	100%	32.50

Redes Neuronales Recurrentes

Classification of Cardiac Arrhythmias from Single Lead ECG
with a Convolutional Recurrent Neural Network (van Zaen et al 2019)

Table 2: Neural network architecture. The output size is given as $N_w \times N_s \times N_c$ where N_w denotes the variable number of windows, N_s the number of samples, and N_c the number of channels.

Layer	Output size
Input windows	$N_w \times 512 \times 1$
Convolutional layer 1	$N_w \times 256 \times 8$
Convolutional layer 2	$N_w \times 128 \times 16$
Convolutional layer 3	$N_w \times 64 \times 32$
Convolutional layer 4	$N_w \times 32 \times 64$
Convolutional layer 5	$N_w \times 16 \times 128$
Convolutional layer 6	$N_w \times 8 \times 256$
Convolutional layer 7	$N_w \times 4 \times 512$
Global average pooling	$N_w \times 512$
LSTM layer	128
Softmax layer	4

Redes Neuronales Recurrentes

Classification of Cardiac Arrhythmias from Single Lead ECG
with a Convolutional Recurrent Neural Network

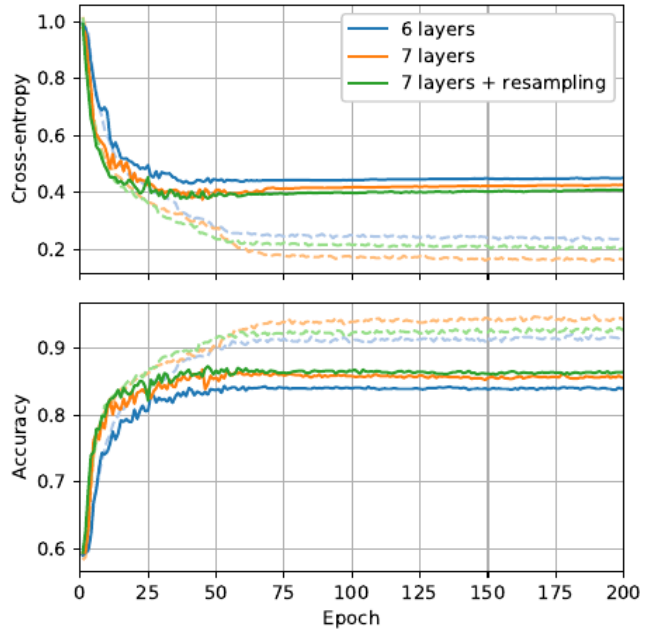


Figure 5: Cross-entropy loss (top) and accuracy (bottom) evaluated during training for a network with 6 convolutional layers without resampling (blue), a network with 7 convolutional layers without resampling (orange), and a network with 7 convolutional layers with resampling (green). Dashed lines denote results obtained on the training set and solid lines results obtained on the test set.

Redes Neuronales Recurrentes

Show and Tell: A Neural Image Caption Generator (Vinyals et al 2015)

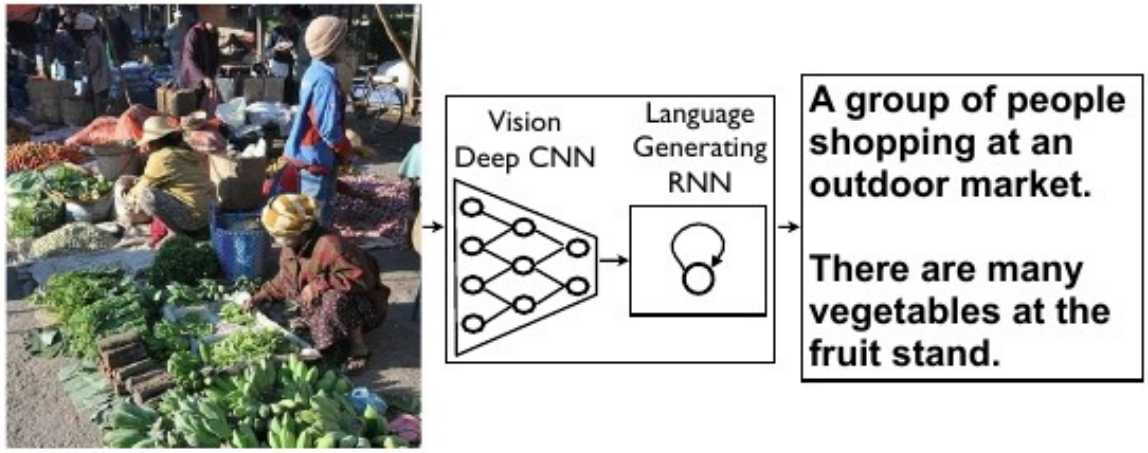


Figure 1. NIC, our model, is based end-to-end on a neural network consisting of a vision CNN followed by a language generating RNN. It generates complete sentences in natural language from an input image, as shown on the example above.

Redes Neuronales Recurrentes

Show and Tell: A Neural Image Caption Generator

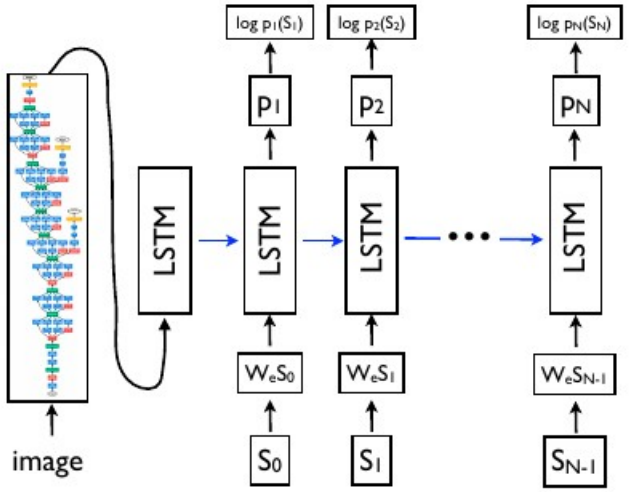


Figure 3. LSTM model combined with a CNN image embedder (as defined in [12]) and word embeddings. The unrolled connections between the LSTM memories are in blue and they correspond to the recurrent connections in Figure 2. All LSTMs share the same parameters.

Redes Neuronales Recurrentes

Show and Tell: A Neural Image Caption Generator

Figure 5. A selection of evaluation results, grouped by human rating.

Redes Neuronales Recurrentes

Movie Reviews Sentiment Analysis

https://www.kaggle.com/c/movie-review-sentiment-analysis-kernels-only/overview

"There's a thin line between likably old-fashioned and fuddy-duddy, and The Count of Monte Cristo ... never quite settles on either side."

The Rotten Tomatoes movie review dataset is a corpus of movie reviews used for sentiment analysis, originally collected by Pang and Lee [1]. In their work on sentiment treebanks, Socher et al. [2] used Amazon's Mechanical Turk to create fine-grained labels for all parsed phrases in the corpus. This competition presents a chance to benchmark your sentiment-analysis ideas on the Rotten Tomatoes dataset. You are asked to label phrases on a scale of five values: negative, somewhat negative, neutral, somewhat positive, positive. Obstacles like sentence negation, sarcasm, terseness, language ambiguity, and many others make this task very challenging.