Argentina programa 4.0

Universidad Nacional de San Martín

Módulo 3
Aprendizaje Automático

Argentina programa 4.0

Universidad Nacional de San Martín

Módulo 3
Aprendizaje Automático

Semana 9.
Redes Neuronales Recurrentes

Contenidos del módulo

ML Clásico

- Árboles de Decisión
- Métodos de Ensemble
  - Bagging / Pasting —> Random Forests
  - Boosting
- Support Vector Machines

Deep Learning

- Redes Neuronales
- Redes Neuronales Convolucionales
- Auto-Encoders / Auto-Encoders Variacionales
- Redes Neuronales Recurrentes (LSTM, otras)
- Extras:
  - Generative Adversarial Networks (GAN)
  - Reinforcement Learning

Autoencoders variacionales

2013 → 2023

Auto-Encoding Variational Bayes

Diederik P. Kingma
Machine Learning Group
Universiteit van Amsterdam
dpkingma@gmail.com

Max Welling
Machine Learning Group
Universiteit van Amsterdam
welling.max@gmail.com

Abstract

How can we perform efficient inference and learning in directed probabilistic models, in the presence of continuous latent variables with intractable posterior distributions, and large datasets? We introduce a stochastic variational inference and learning algorithm that scales to large datasets and, under some mild differentiability conditions, even works in the intractable case. Our contributions are two-fold. First, we show that a reparameterization of the variational lower bound yields a lower bound estimator that can be straightforwardly optimized using standard stochastic gradient methods. Second, we show that for i.i.d. datasets with continuous latent variables per datapoint, posterior inference can be made especially efficient by fitting an approximate inference model (also called a recognition model) to the intractable posterior using the proposed lower bound estimator. Theoretical advantages are reflected in experimental results.

2013 → 2023

GAN, Atención — 2014
2015 — DCGAN
Transformers — 2017
2018 — StyleGAN
GPT-3 — 2020
2021 — DALL-E
ChatGPT — 2022
Llama, Bard — multi-modal

Redes Neuronales Recurrentes

Hasta ahora, solo hemos visto datos *estáticos*

Los datos secuenciales tienen una dimensión temporal (discreta): El mismo conjunto de características se repite en pasos ordenados

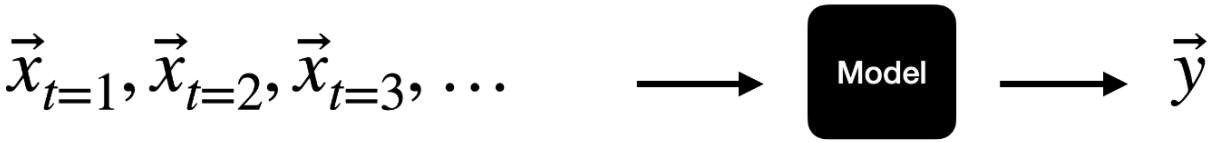
$$\text{input} : \vec{x}_{t=1}, \vec{x}_{t=2}, \vec{x}_{t=3}, \ldots$$

La salida, o las predicciones, pueden ser un solo vector o incluso una secuencia:

$$\text{vector output} : \vec{y} \qquad \text{sequence output} : \vec{y}_{t=1}, \vec{y}_{t=2}, \vec{y}_{t=3}, \ldots$$

Tipos de modelos

Modelos secuencia a vector:



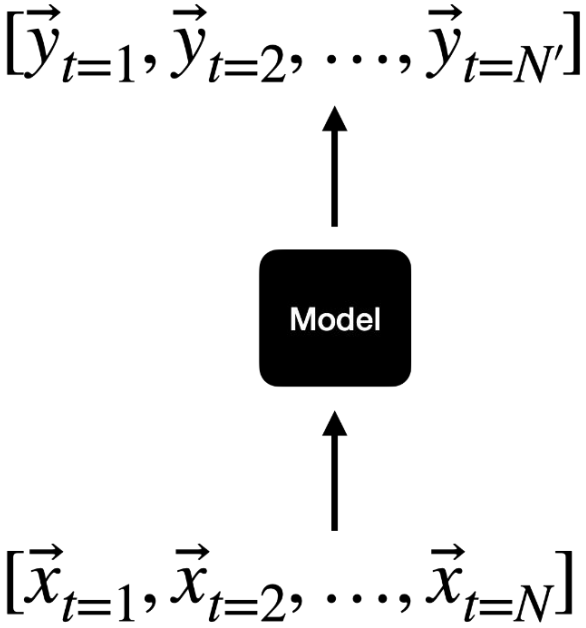
P.ej.: Analisis de sentimientos, ...

O modelos secuencia a secuencia

P.ej.: traducciones, predicciones de series temporales, ...

Tratamiento ingenuo (I)

Para una longitud de secuencia fija, podemos tratar el problema como un problema estático, como cualquier otro que hayamos visto:

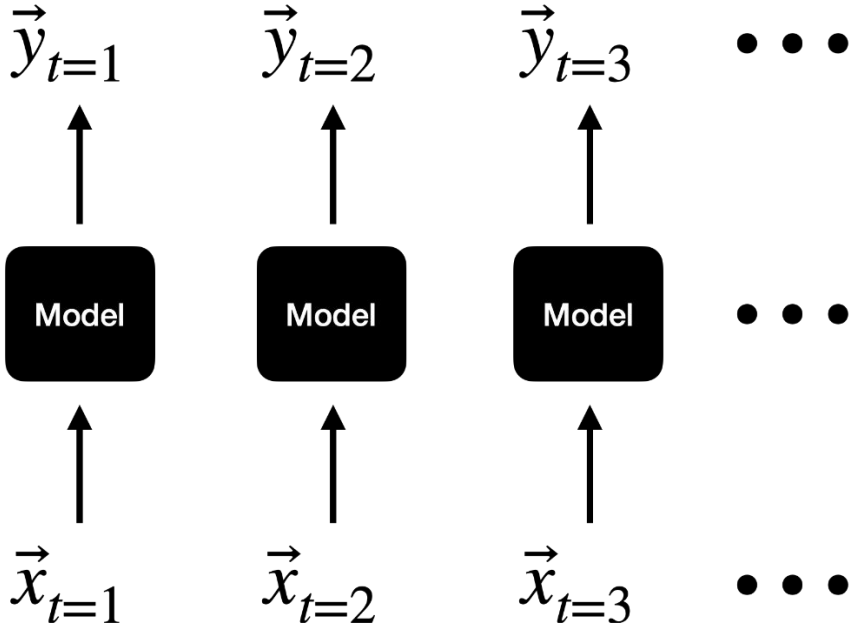


Pero el tamaño de la red aumenta rápidamente con el tamaño de la secuencia.

Además, este modelo pasa por alto la estructura temporal de los datos.

Tratamiento ingenuo (II)

Podemos intentar utilizar la simetría temporal y construir un modelo que funcione en cada paso de tiempo:



Pero el suyo funcionaría terriblemente mal. Estamos tratando cada paso de tiempo como una muestra separada, desechando cualquier correlación temporal.

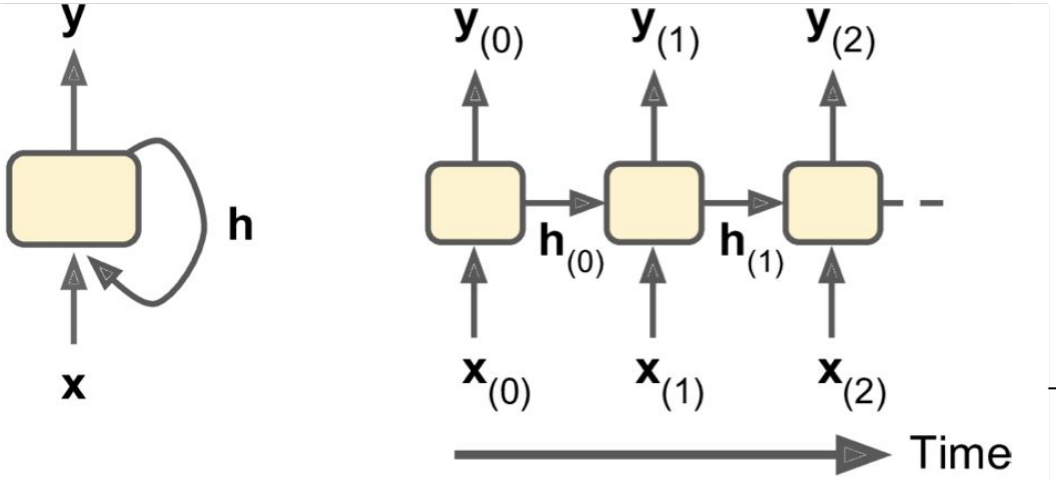
Es un modelo sin memoria.

Redes Recurrentes

"If training vanilla neural nets is optimization over functions, training recurrent nets is optimization over programs."

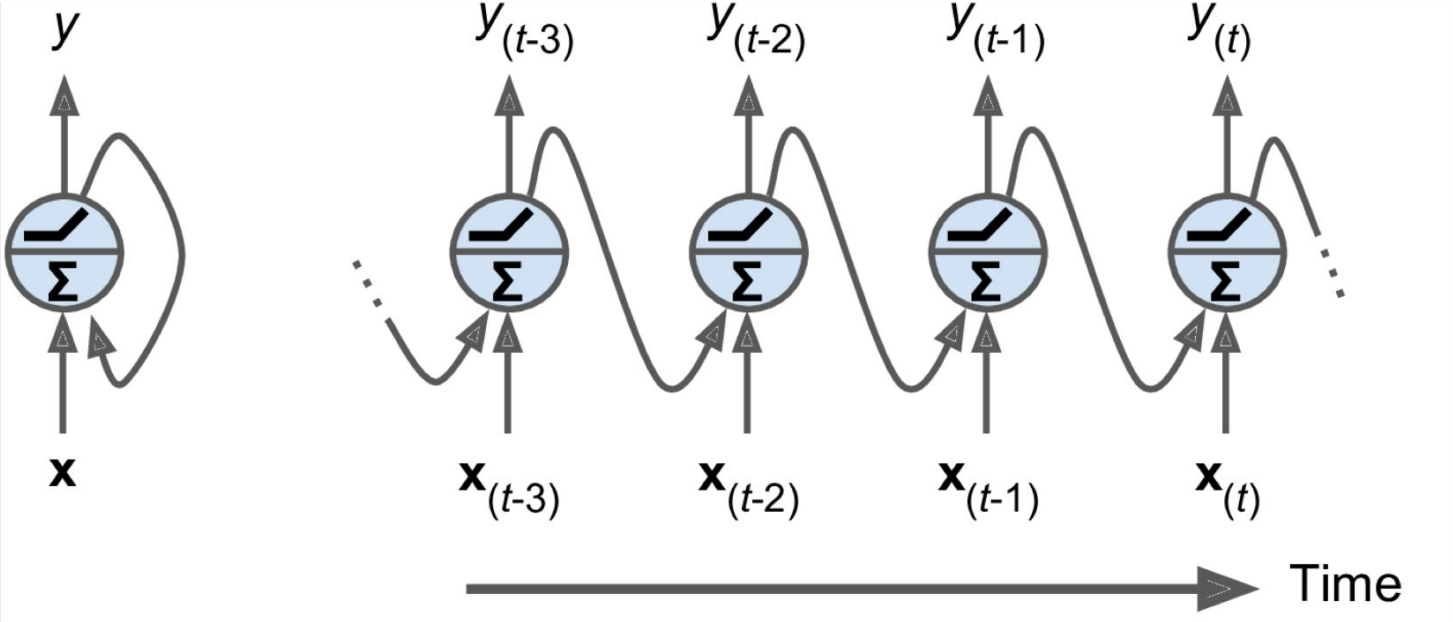
https://karpathy.github.io/2015/05/21/rnn-effectiveness/

```
class RNN:
  # ...
  def step(self, x):
    # update the hidden state
    self.h = np.tanh(np.dot(self.W_hh, self.h) + np.dot(self.W_xh, x))
    # compute the output vector
    y = np.dot(self.W_hy, self.h)
    return y
```



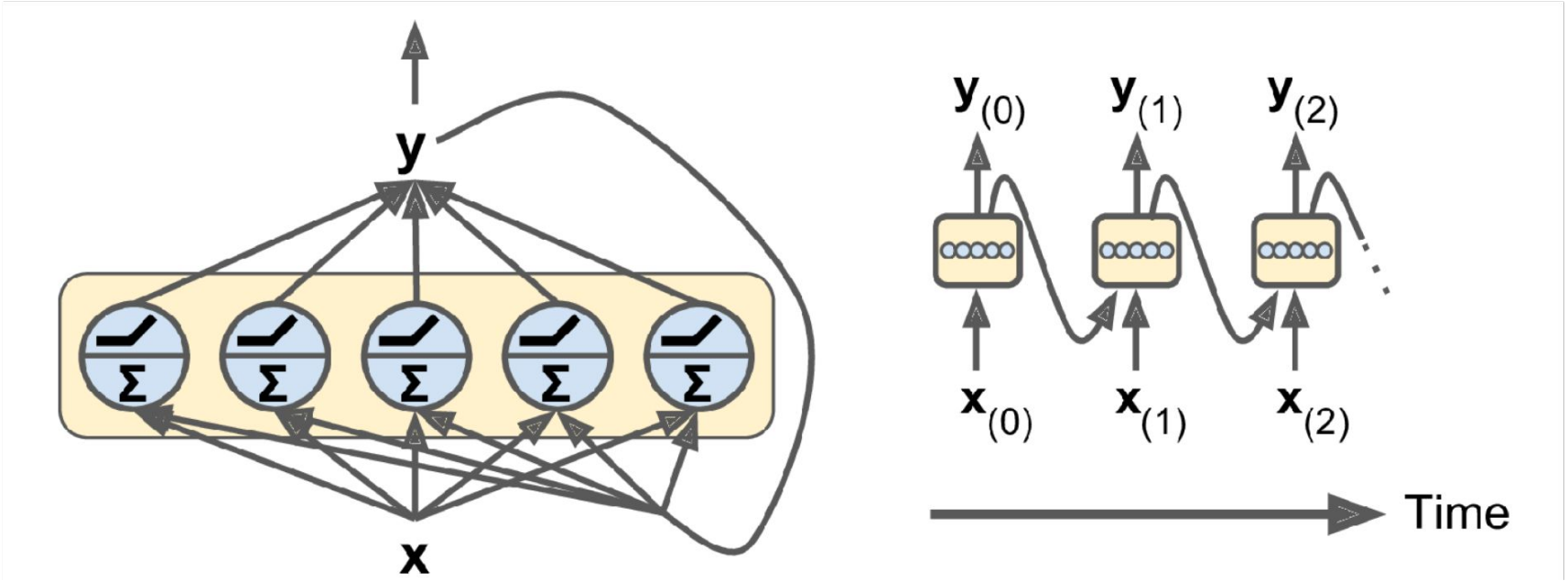
Redes Recurrentes Sencillas

En el caso más sencillo, el estado que se transmite a la próxima neurona es exactamente la salida y



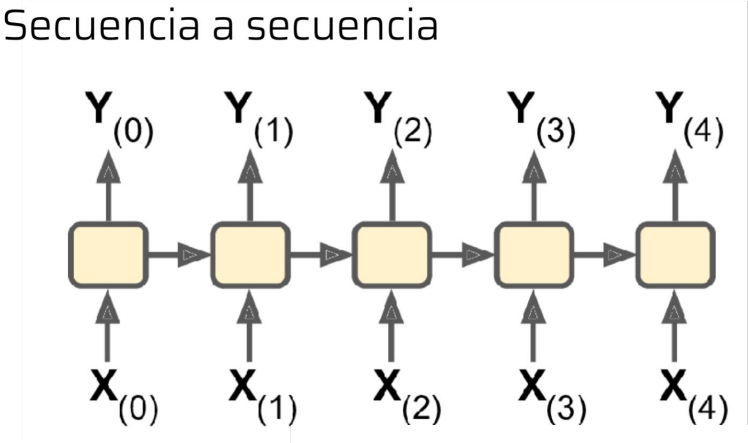
RNNs profundas

Capa de muchas neuronas, desenrollada en el tiempo

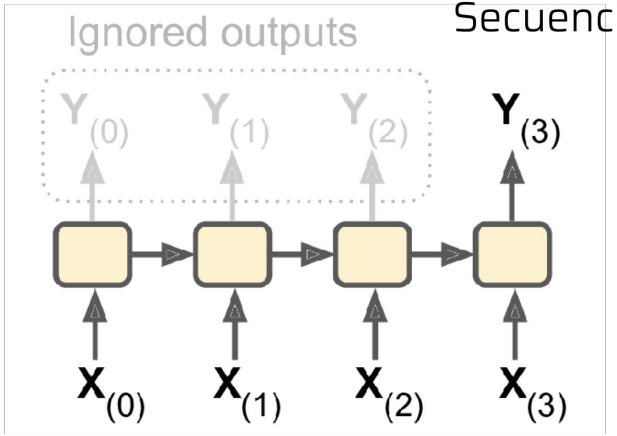


Distintas arquitecturas

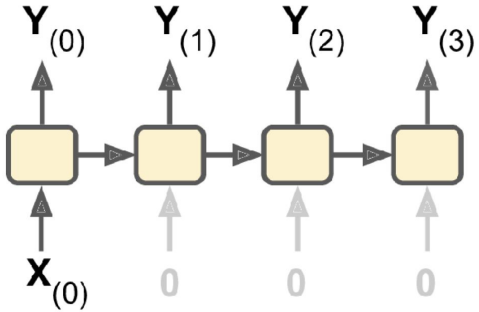
Secuencia a secuencia

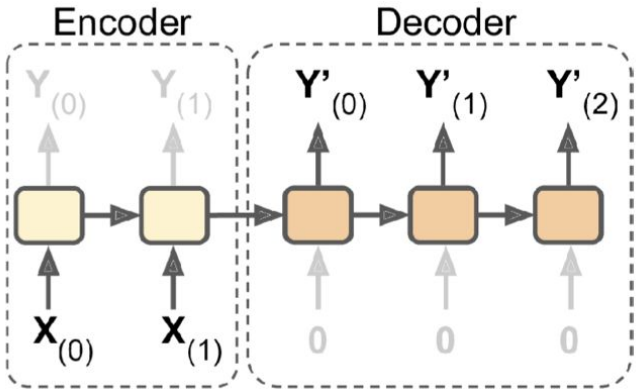


Secuencia a vector



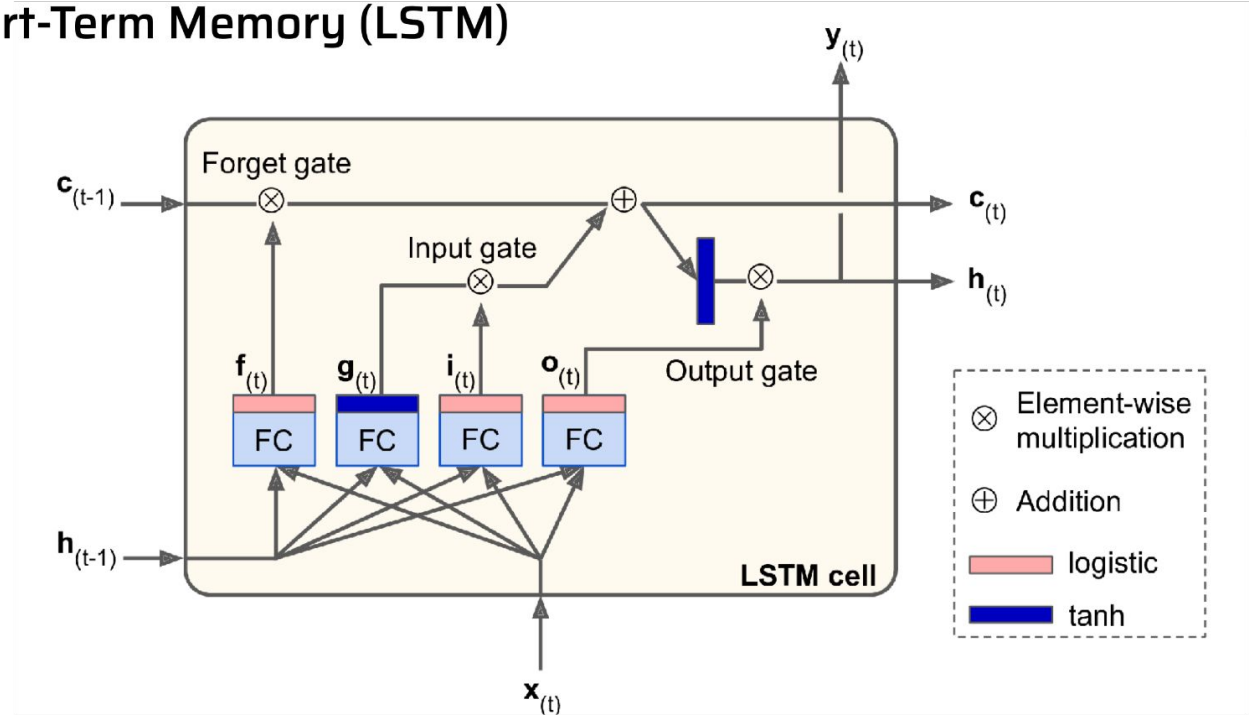
Vector a secuencia





Celdas más complejas

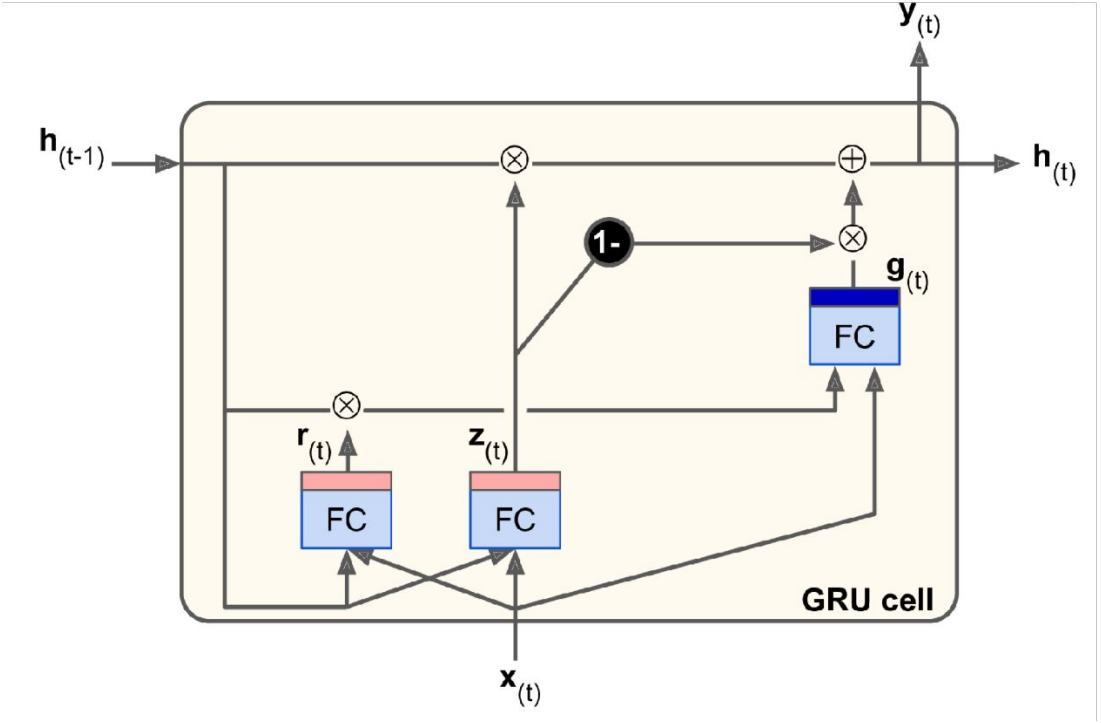
Long Short-Term Memory (LSTM)



Compuertas (~binarias): Forget (f), Input(i), Output(i)

Celdas más complejas

Gated Recurrent Unit (GRU)

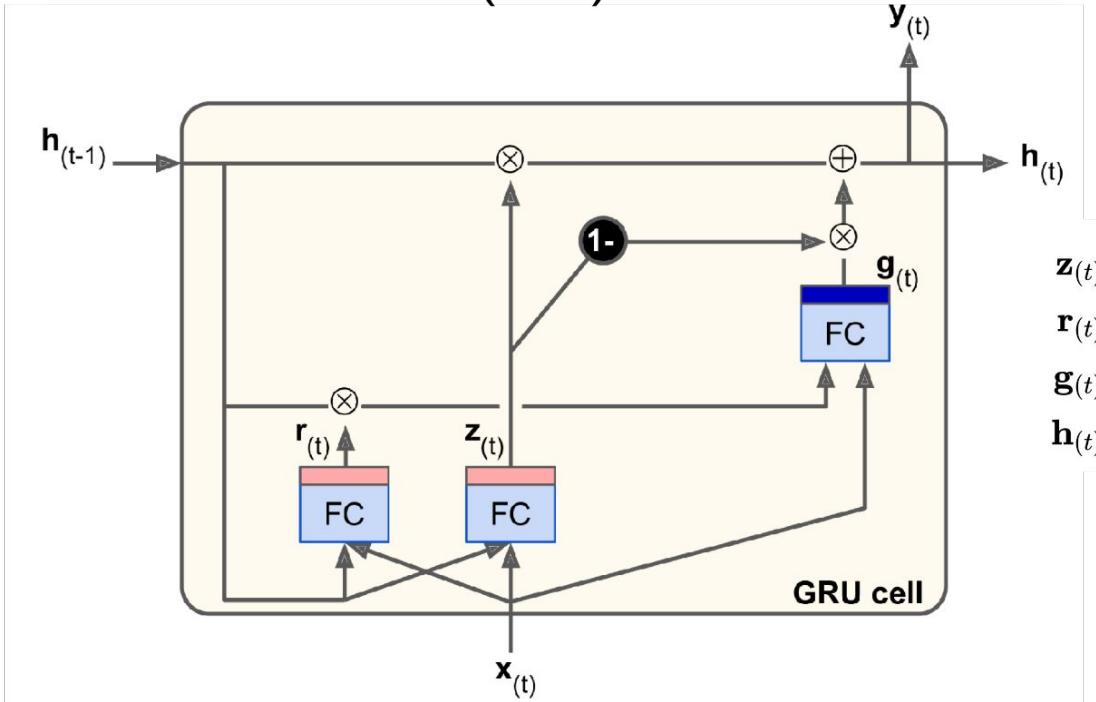


LSTM simplificada:

- Único estado: h
- Compuertas de entrada y de salida únicas: z

Celdas más complejas

Gated Recurrent Unit (GRU)



$$
\begin{aligned}
\mathbf{z}_{(t)} &= \sigma\left(\mathbf{W}_{xz}{}^{\intercal}\mathbf{x}_{(t)} + \mathbf{W}_{hz}{}^{\intercal}\mathbf{h}_{(t-1)} + \mathbf{b}_z\right) \\
\mathbf{r}_{(t)} &= \sigma\left(\mathbf{W}_{xr}{}^{\intercal}\mathbf{x}_{(t)} + \mathbf{W}_{hr}{}^{\intercal}\mathbf{h}_{(t-1)} + \mathbf{b}_r\right) \\
\mathbf{g}_{(t)} &= \tanh\left(\mathbf{W}_{xg}{}^{\intercal}\mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^{\intercal}\left(\mathbf{r}_{(t)} \otimes \mathbf{h}_{(t-1)}\right) + \mathbf{b}_g\right) \\
\mathbf{h}_{(t)} &= \mathbf{z}_{(t)} \otimes \mathbf{h}_{(t-1)} + \left(1 - \mathbf{z}_{(t)}\right) \otimes \mathbf{g}_{(t)}
\end{aligned}
$$

Celdas más complejas

RNNs
Como regla general, una RNN simple tiene una memoria de ~10 pasos de tiempo.
Podemos conseguir una memoria ~10 veces mejor con algunas arquitecturas de red más avanzadas.

LSTM
Consiste en dos estados ocultos (memorias a largo y corto plazo), y entrena puertas para saber cuándo recordar u olvidar algo, y cuánto utilizar para la salida.

GRU
Es una versión simplificada de la LSTM, tiene un rendimiento más rápido y normalmente mejor que la LSTM.

Notebook

Revisión en

`Notebook_Semana_9_RNN.ipynb`