Práctica 7
Redes Neuronales Recurrentes

TÉCNICAS ESTADÍSTICAS PARA EL APRENDIZAJE II

Máster Universitario en Estadística Computacional
y Ciencia de Datos para la Toma de Decisiones

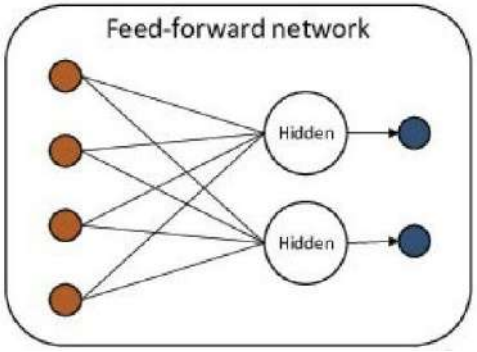
Redes Neuronales Recurrentes

Redes neuronales *feed-forward*

- Problemas de clasificación y regresión

Redes Neuronales Convolucionales

- Principalmente Imágenes

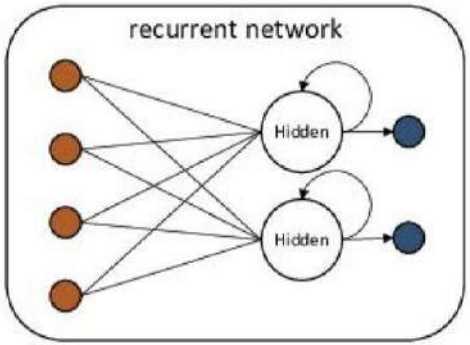


En las redes anteriores cada entrada que se les muestra se procesa de forma independiente, sin que se conserve ningún estado entre una entrada y otra.

¿Cómo procesamos datos secuenciales como, por ejemplo, una serie temporal de datos?

Una **red neuronal recurrente** (**RNN**) procesa secuencias iterando a través de los elementos de la secuencia y manteniendo un estado que contiene información relativa a lo que ha visto hasta el momento.

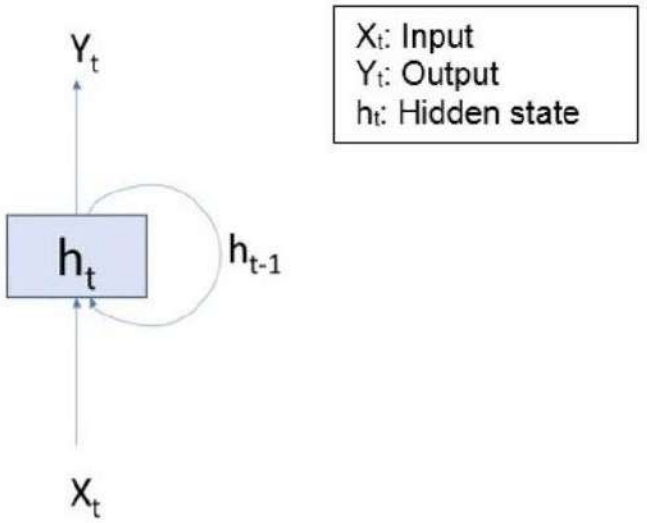
Una neurona de una red recurrente predice su salida a partir de la entrada y también a partir de estados anteriores. Se podría decir que tiene cierta forma de **memoria** (pueden mantener información del contexto).



Redes Neuronales Recurrentes

Las neuronas recurrentes contienen un **estado oculto y bucles**, que permiten que la red almacene información anterior en el estado oculto y funcione con secuencias.

Las RNNs tienen **dos conjuntos de pesos**: uno para el vector del estado oculto y otro para las entradas. Durante el entrenamiento, la red aprende los pesos de las entradas y del estado oculto. Cuando se implementa, **la salida se basa en la entrada actual y en el estado oculto, que a su vez se basa en entradas anteriores**.



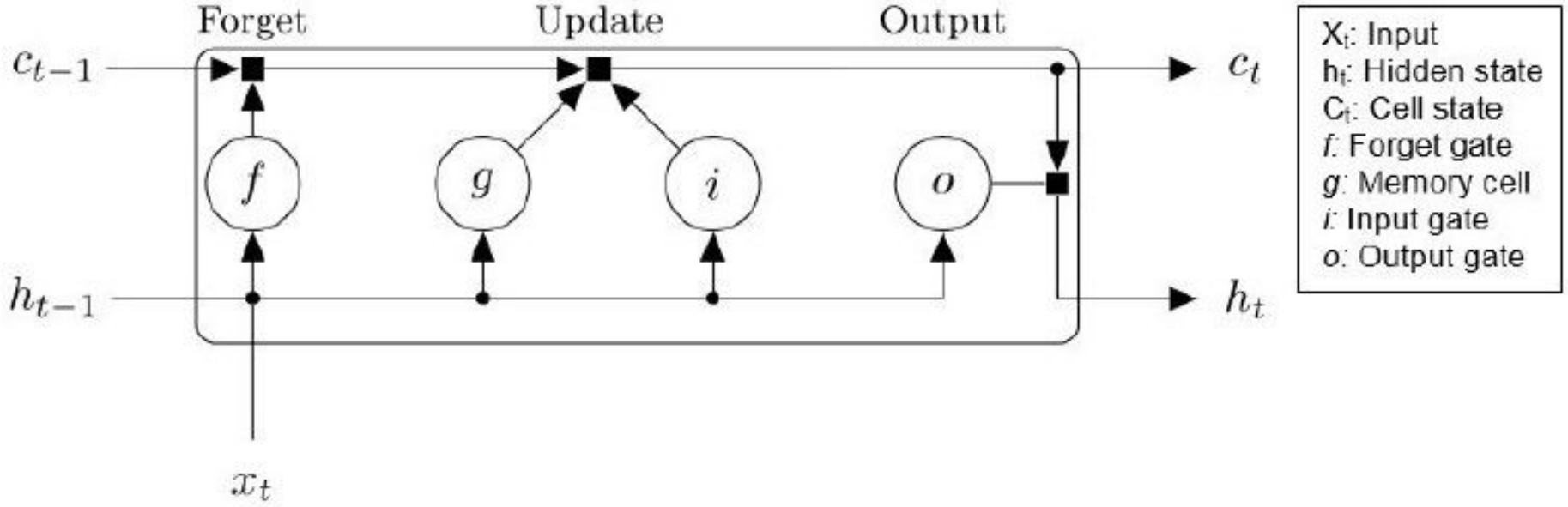
En RNNs , **el calculo del gradiente para un instante dado considera los instantes anteriores**. Si tenemos muchos estados anteriores el gradiente puede ser demasiado pequeño, lo que se conoce como el problema de desvanecimiento del gradiente (*vanishing gradient*) y la red puede que sea incapaz de aprender correctamente los pesos. En el mejor de los casos, aprenderá muy lentamente.

Existen diferentes capas recurrentes. Una de las más utilizadas es la **capa LSTM (*Long Short-Term Memory*)**. El algoritmo subyacente de memoria a largo plazo fue desarrollado por Hochreiter y Schmidhuber en 1993.

Una capa LSTM permite que la información pasada se reinyecte en un momento posterior, luchando así contra el problema de desvanecimiento del gradiente, es decir, es capaz de recordar un dato relevante en la secuencia y de preservarlo por varios instantes de tiempo.

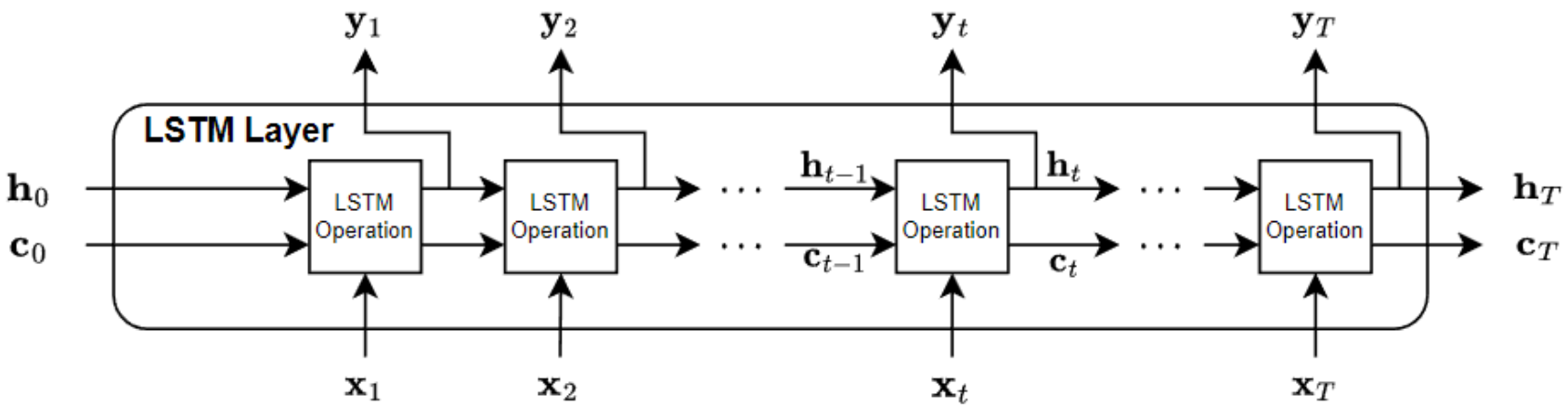
Las capas LSTM utilizan **puertas adicionales** para controlar qué información del estado oculto se exporta como salida y al siguiente estado oculto.

Además del estado oculto de las RNNs tradicionales, la arquitectura de un bloque LSTM generalmente tiene una **celda de memoria, puerta de entrada, puerta de salida y puerta de olvido**. Las puertas adicionales permiten que la red aprenda relaciones de datos a largo plazo de manera más eficaz.



Los pesos y sesgos de la **puerta de entrada** controlan cuánto fluye un nuevo valor en la unidad LSTM. Del mismo modo, los pesos y sesgos de la **puerta de olvido** y la **puerta de salida** controlan cuánto permanece un valor en la unidad, y hasta qué punto se utiliza ese valor para calcular la activación de la salida del bloque LSTM, respectivamente.

Flujo de datos a través de una **capa LSTM con múltiples unidades de tiempo**:



Cada operación de la capa LSTM recibe el estado oculto y el estado de celda de la operación anterior, y pasa un estado oculto y estado de celda actualizados a la siguiente operación.

Las RNNs son **difíciles de entrenar** y en muchos casos no resulta práctico su uso ante problemas que puedan resolverse con redes *feed-forward*.

Se recomiendan únicamente para problemas que requieran esa potencia, donde la secuencia de datos y la dinámica temporal que los conecta es importante. Por ejemplo, en tareas de **procesamiento de señales**, en problemas complejos de procesamiento del lenguaje natural como la **respuesta a preguntas** o la **traducción automática**, etc.

En **Keras** hay disponibles diferentes capas recurrentes, siendo una de las más utilizadas la **capa LSTM:** https://keras.io/api/layers/recurrent_layers/lstm/

Normalmente, sólo se especifica la dimensionalidad de la capa LSTM y se dejan todos los demás argumentos (hay muchos) en los valores por defecto de Keras. Keras tiene buenos valores por defecto, y las cosas casi siempre "simplemente funcionan" sin tener que pasar tiempo ajustando los parámetros a mano.

Keras ofrece también otras implementaciones de RNN como es la ***Gated Recurrent Unit* (GRU).**

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

Las capas GRU aparecieron en el 2014, y usan el mismo principio que LSTM, pero están simplificadas de manera que su rendimiento está a la par con LSTM pero computacionalmente son más eficientes.

Redes Neuronales Recurrentes

En esta práctica trabajaremos con series temporales y RNNs.

Una **serie temporal** puede ser cualquier dato obtenido mediante **mediciones a intervalos regulares**, como el precio diario de una acción, el consumo eléctrico horario de una ciudad o las ventas semanales de una tienda.

A diferencia de los tipos de datos que hemos visto hasta ahora, trabajar con series temporales implica **comprender la dinámica de un sistema**: sus ciclos periódicos, su evolución en el tiempo, su régimen regular y sus picos repentinos.

La **tarea más común** relacionada con las series temporales es, con diferencia, la **previsión**: **predecir qué ocurrirá a continuación en una serie**. Por ejemplo, predecir el consumo de electricidad con unas horas de antelación para poder anticipar la demanda; predecir los ingresos con unos meses de antelación para poder planificar el presupuesto; predecir el tiempo con unos días de antelación para poder planificar el horario. No obstante, **hay muchas otras cosas que se pueden hacer con las series temporales** como clasificación, detección de eventos, detección de anomalías, etc.

Redes Neuronales Recurrentes

En esta práctica nos centraremos en el problema de **predecir la temperatura 24 horas en el futuro**, dada una serie temporal de mediciones horarias de magnitudes como la presión atmosférica y la humedad, registradas en el pasado reciente por un conjunto de sensores situados en el tejado de un edificio.

Trabajaremos con un conjunto de datos de series temporales meteorológicas registradas en la estación meteorológica del Instituto Max Planck de Biogeoquímica de Jena, Alemania. En este conjunto de datos, se registraron 14 cantidades diferentes (temperatura, presión, humedad, dirección del viento, etc.) cada 10 minutos durante varios años. Los datos originales se remontan a 2003, pero el subconjunto de datos que vamos a usar se limita a 2009-2016.

Descripción de la base de datos:

Date Time: fecha y hora
p (mbar): presión atmosférica en milibares
T (degC): temperatura en Celsius
Tpot (K): temperatura en Kelvin
Tdew (degC): temperatura en Celsius relativa a la humedad.
rh (%): humedad relativa. Medida de qué tan saturado está el aire con vapor de agua
VPmax (mbar): presión de vapor de saturación
VPact (mbar): presión de vapor
VPdef (mbar): déficit de presión de vapor
sh (g/kg): humedad específica
H2OC (mmol/mol): concentración de vapor de agua
rho (g/m**3): hermético
wv (m/s): velocidad del viento
max. wv (m/s): velocidad máxima del viento
wd (deg): dirección del viento en grados

En todos nuestros experimentos, utilizaremos el primer 50% de los datos para el entrenamiento, el siguiente 25% para la validación y el último 25% para las pruebas.

Cuando se trabaja con datos de series temporales, es importante utilizar **datos de validación y prueba** que sean **más recientes** que los datos de entrenamiento, ya que lo que se pretende es **predecir el futuro a partir del pasado**, y no al revés, y la división de los datos debe reflejarlo (algunos problemas resultan mucho más sencillos si se invierte el eje temporal).

La formulación exacta del problema será la siguiente: **dados unos datos que cubren los cinco días anteriores y muestreados una vez por hora, ¿podemos predecir la temperatura en 24 horas?**

Antes de empezar a utilizar modelos de aprendizaje profundo para resolver el problema de la predicción de la temperatura, pensemos en un **enfoque sencillo y de sentido común** que nos servirá para **establecer una base** que tendremos que superar para demostrar la utilidad de modelos de aprendizaje automático más avanzados.

En este caso, se puede suponer que las series temporales de temperatura son continuas (es probable que las temperaturas de mañana sean similares a las de hoy) y periódicas con un periodo diario. Por lo tanto, un enfoque de sentido común podría ser **predecir siempre que la temperatura dentro de 24 horas será igual a la temperatura actual.**

Con este enfoque se logra un MAE de validación de 2.44 grados Celsius y un MAE de prueba de 2.62 grados Celsius. Por lo tanto, si siempre suponemos que la temperatura dentro de 24 horas será la misma que la actual, **nos equivocaremos en dos grados y medio en promedio**. No está tan mal, pero probablemente no lanzaríamos un servicio de pronóstico del tiempo basado en esta heurística. Ahora el juego consiste en intentar hacerlo mejor.

También es útil probar **modelos de aprendizaje automático sencillos** (como redes pequeñas densamente conectadas) antes de buscar modelos complicados y costosos desde el punto de vista informático, como las RNNs. Esta es la mejor manera de asegurarse de que cualquier complejidad adicional que se añada al problema es legítima y aporta beneficios reales.

En el **enfoque densamente conectado** primero aplanaremos las series temporales, lo que elimina la noción de tiempo de los datos de entrada.

Después analizaremos los datos como lo que son, una secuencia, donde la causalidad y el orden importan, haciendo uso de **RNNs**.

Para resolver nuestro problema, crearemos un objeto *Dataset* que proporcione lotes de datos de los últimos cinco días junto con una temperatura objetivo para dentro de 24 horas.

Dado que las muestras del conjunto de datos son muy redundantes (la muestra N y la muestra N + 1 tendrán en común la mayoría de sus pasos temporales), sería un derroche asignar explícitamente memoria para cada muestra. En su lugar, generaremos las muestras sobre la marcha mientras solo mantenemos en la memoria las estructuras originales *data* y *temperature*, y nada más.

Hay una utilidad de conjunto de datos incorporada en Keras que hace precisamente eso (***timeseries_dataset_from_array()***) que se puede utilizar para cualquier tipo de tarea de previsión de series temporales.

La idea general de ***timeseries_dataset_from_array()*** es que le proporcionamos una matriz de datos de series temporales (el argumento *data*), y nos proporciona ventanas extraídas de la serie temporal original (que llamaremos "secuencias")

Veamos un ejemplo sencillo:
data = [0 1 2 3 4 5 6]
sequence_length=3
timeseries_dataset_from_array() generará las siguientes muestras:
[0 1 2], [1 2 3], [2 3 4], [3 4 5], [4 5 6].

También podemos pasar el argumento *targets* a *timeseries_dataset_from_array()*. La primera entrada de *targets* debe coincidir con el objetivo deseado para la primera secuencia que se generará a partir de la matriz *data*. Por lo tanto, si estamos haciendo pronósticos de series temporales, *targets* debe ser la misma matriz que data, con un desplazamiento por una cierta cantidad.

Por ejemplo, con *data* y *sequence_length* anteriores, podemos crear un conjunto de datos para predecir el siguiente paso en la serie pasando *targets* = [3 4 5 6 ...].

Utilizaremos *timeseries_dataset_from_array()* para crear tres conjuntos de datos: uno para el entrenamiento, otro para la validación y otro para las pruebas, con los siguientes valores de parámetros:

sampling_rate = 6. Las observaciones se muestrearán a razón de un punto de datos por hora: sólo conservaremos un punto de datos de cada 6.

sequence_length = 120. Las observaciones se remontan a 5 días (120 horas).

delay = *sampling_rate* * (*sequence_length* + 24 - 1) = 858 (6 días de mediciones). El objetivo de una secuencia será la temperatura 24 horas después del final de la secuencia.

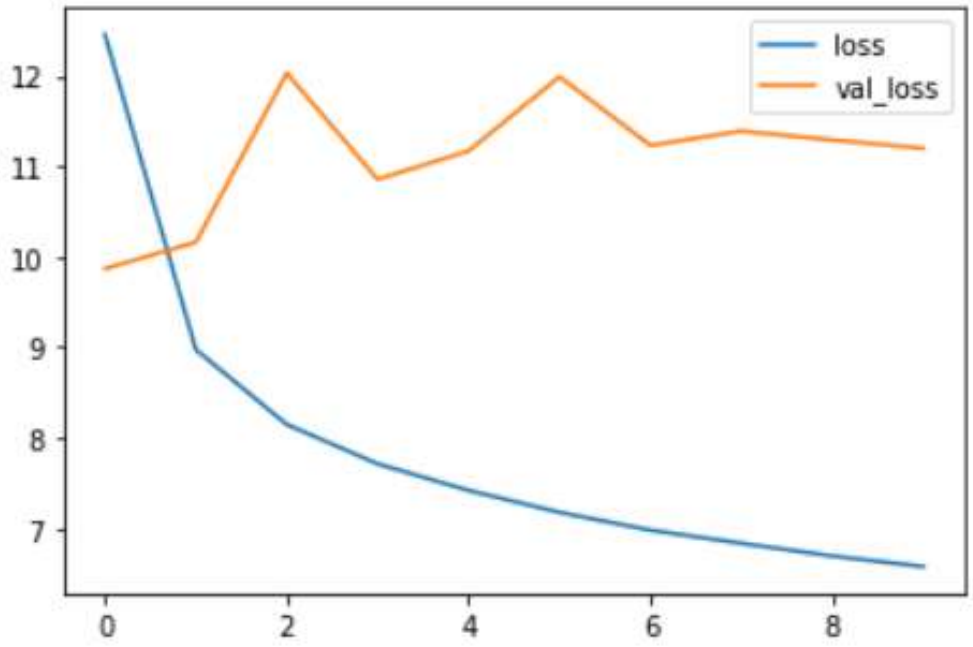
Cada conjunto de datos producirá una tupla (*muestras*, *objetivos*), donde *muestras* será un lote de 256 muestras, cada una de las cuales contiene 120 horas consecutivas de datos de entrada, y *objetivos* es la correspondiente matriz de 256 temperaturas objetivo.

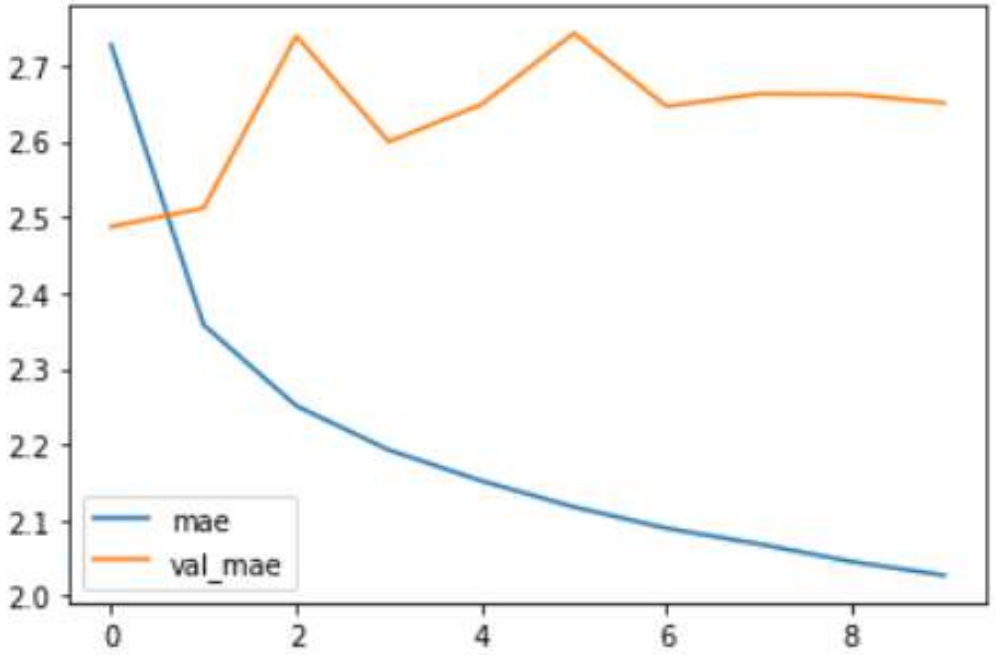
Además, las muestras se barajan aleatoriamente, por lo que dos secuencias consecutivas de un lote (como *muestras[0]* y *muestras[1]*) no están necesariamente próximas en el tiempo.

Red neuronal densamente conectada:

	loss	mae	val_loss	val_mae
0	12.454168	2.727469	9.867157	2.487300
1	8.974234	2.357505	10.160137	2.511800
2	8.147337	2.250639	12.028915	2.739122
3	7.711767	2.192345	10.855097	2.599455
4	7.418180	2.151590	11.168503	2.648406
5	7.176542	2.117082	11.989826	2.742723
6	6.979064	2.088968	11.229738	2.646136
7	6.834569	2.068130	11.387469	2.662618
8	6.693655	2.044904	11.289682	2.661829
9	6.577634	2.026908	11.198376	2.650585

Predecir siempre que la temperatura dentro de 24 horas será igual a la temperatura actual: MAE de validación de 2.44 grados Celsius y un MAE de prueba de 2.62 grados Celsius



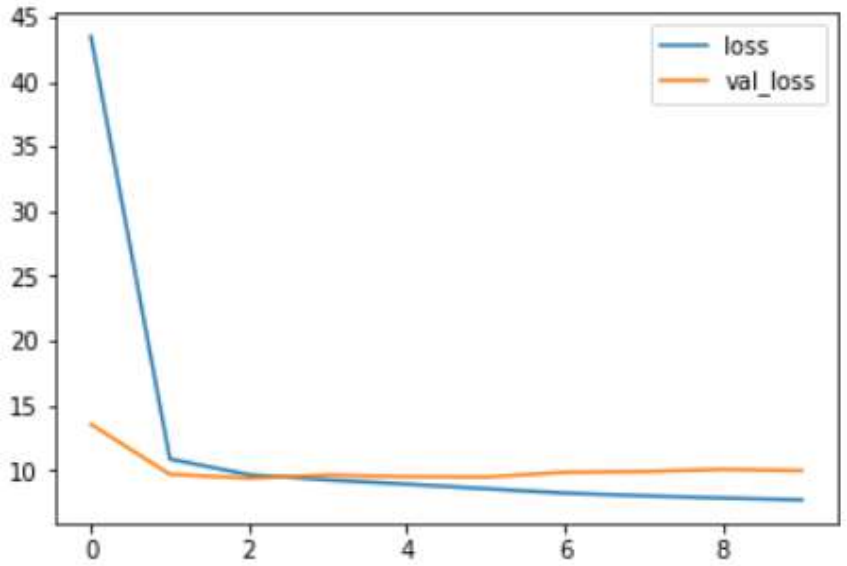


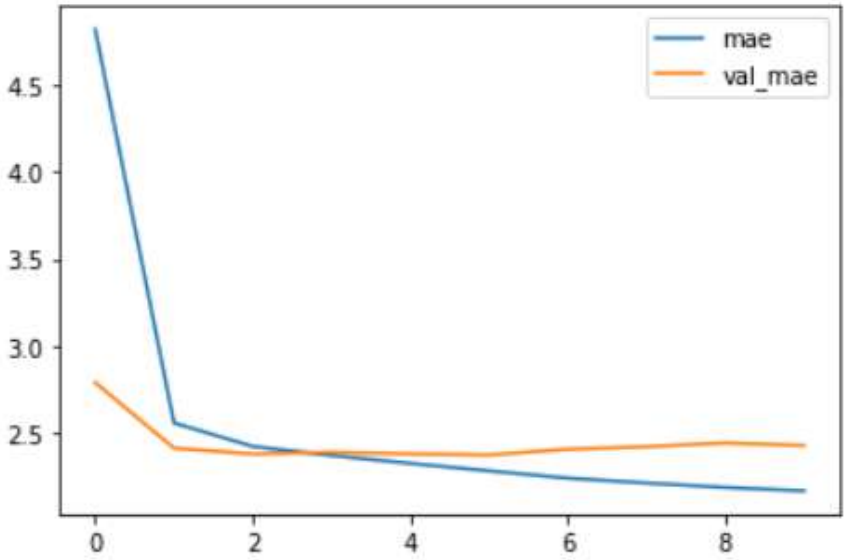
Conjunto test: loss: 12.8800 - mae: 2.8198

Red neuronal recurrente:

	loss	mae	val_loss	val_mae
0	43.504196	4.815332	13.554334	2.791520
1	10.866060	2.560505	9.689913	2.416552
2	9.658811	2.425993	9.389549	2.382643
3	9.258682	2.374158	9.635159	2.392448
4	8.931483	2.328682	9.513363	2.382849
5	8.575382	2.285868	9.477625	2.377295
6	8.238463	2.244312	9.841351	2.410214
7	8.030267	2.216438	9.908163	2.424403
8	7.854950	2.191018	10.073985	2.445941
9	7.715842	2.171483	9.972598	2.431124

Predecir siempre que la temperatura dentro de 24 horas será igual a la temperatura actual: MAE de validación de 2.44 grados Celsius y un MAE de prueba de 2.62 grados Celsius





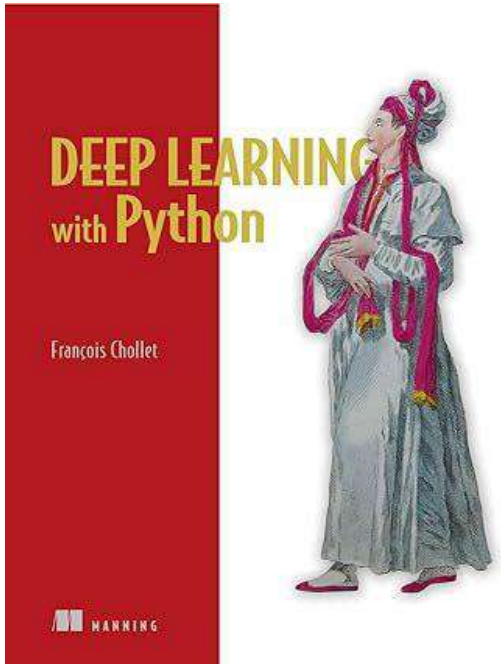
Conjunto test: loss: 11.2143 - mae: 2.6176

Mejoramos sensiblemente el problema de *overfitting* pero el MAE es todavía comparable al del enfoque de sentido común.

Redes Neuronales Recurrentes

Las redes neuronales recurrentes están desarrolladas con ejemplos prácticos en el libro ***Deep Learning with Python,*** en los capítulos 10 "Deep learning for timeseries" y 11 "Deep learning for text".

Otro tipo de redes que han generado una gran expectación en la comunidad de investigación porque permiten generar nuevos datos sintéticos que resultan indistinguibles de datos reales son las **redes generativas antogónicas** (*Generative Adversarial Networks,* **GANs**) introducidas por primera vez por Ian Goodfellow en 2014. Estas redes se desarrollan en el capítulo 12 "*Generative deep learning*" del libro *Deep Learning with Python.*



Redes generativas antogónicas (GANs)

Una GAN **genera nuevos datos** a partir de un conjunto de datos de entrenamiento determinado. Por ejemplo, puede generar nuevas imágenes a partir de una base de datos de imágenes existente, música original a partir de una base de datos de canciones, etc.

Una GAN entrena dos redes neuronales diferentes y las enfrenta entre sí:

- Una red, el **generador**, genera nuevos datos al tomar una muestra de datos de entrada modificándolos en la medida de lo posible.
- La otra red, el **discriminador**, intenta predecir si la salida de datos generada pertenece al conjunto de datos original, es decir, determina si los datos generados son falsos o reales. El sistema genera versiones nuevas y mejoradas de valores de datos falsos hasta que el discriminador ya no puede distinguir el falso del original.

Aplicaciones de las Redes generativas antogónicas (GANs)

- Convertir una imagen de baja resolución en una de alta resolución o una imagen en blanco y negro en color.
- Crear rostros, personajes y animales realistas para animación y video.
- Pueden ayudar a crear experiencias visuales realistas e inmersivas en videojuegos.
- Generar datos de entrenamiento para otros modelos.
- …

Una preocupación en torno a los modelos generativos es su uso indebido debido a que distinguir entre, por ejemplo, un video original y uno manipulado puede ser un desafío para los humanos e incluso para los propios computadores.

Actualmente se está trabajando en herramientas, basadas a su vez en GANs, para detectar imágenes falsas, discursos de audio *fake*…