Anuario Jurídico y Económico Escurialense, **LII (2019) 257-276 / ISSN: 1133-3677**

Análisis de las series temporales a la luz de Deep Learning

Time Series Analysis and Deep Learning

Dr. Agustín ALONSO RODRÍGUEZ
Real Centro Universitario
"Escorial-María Cristina"
San Lorenzo del Escorial

Resumen: Las nuevas tecnologías dejan sentir su impacto tanto en el ámbito de la estadística, como en el del análisis de las series temporales. *Machine Learning* y *Deep Learning* acercan la potencia de las redes neuronales artificiales para la modelización de las series temporales. En este trabajo se presentan conceptos básicos de estas tecnologías, ilustrando lo expuesto con un ejemplo relativo a la modelización de la producción de energía eléctrica fotovoltaica en España.

Abstract: The new technologies are leaving their impact in statistics as well as in time series analysis. *Machine Learning* and *Deep Learning* show the power of artificial neural networks for modeling time series. In this paper after the presentation of some of the fundamental ideas of these technologies, a model for the production of solar electricity in Spain is presented.

Palabras clave: Redes neuronales artificiales, *Machine Learning, Deep Learning*, aprendizaje supervisado, training data, test data, Keras model, programa **R**.

Keywords: Artificial neural networks, *Machine Learning*, *Deep Learning*, supervised learning, train data, test data, Keras model, program **R**.

Sumario:

I. Introducción.

II. Terminología básica.

III. Redes neuronales artificiales y *Deep Learning.*

IV. Series temporales y Redes neuronales.

V. Modelos para series temporales.

VI. Multilayer-Perceptron.

VII. Producción de energía eléctrica fotovoltaica en España.

VIII. Conclusiones.

IX. Bibliografía.

Recibido: octubre 2018.
Aceptado: enero 2019.

I. INTRODUCCIÓN

El análisis de las series temporales ha experimentado recientemente notables avances. En este sentido, merece citarse la obra de los profesores Robert Hyndman y George Athanasopoulos *Forecasting: Principles and Practice*, segunda edición, 2018, presentada como un resumen de los últimos métodos de predicción, utilizando el programa **R**. Son muchos los años dedicados al tema de las series temporales que preceden a la publicación citada, y que constituyen todo un cúmulo de conocimientos sobre el análisis de las series temporales.

Dicho lo anterior, ¿qué necesidad hay de estudiar el tema a la luz del nuevo enfoque que proporcionan las llamadas *redes neuronales artificiales*?

La razón reside en el hecho de que los modelos de redes neuronales artificiales son muy eficientes a la hora de detectar patrones de comportamiento en las secuencias de datos, tema central de *Machine Learning*, y las series de datos temporales son verdaderas secuencias de observaciones o datos.

Es imposible entrar en excesivos detalles sobre la temática del *Machine Learning* y del *Deep Learning* , sin embargo, parece obligado presentar algunas ideas básicas sobre este tema, ya que nos encontramos ante lo que el profesor Jordi Torres denomina *una nueva tecnología disruptiva*. (Cf. p. 45, *Deep Learning, Introducción práctica con Keras, Primera parte,* 2018).

II. TERMINOLOGÍA BÁSICA

Central en la temática del *Machine Learning* es el concepto de secuencia o sucesión. En su expresión más simple, cabe presentarla como

$$input \rightarrow modelo \rightarrow output$$

En *input* vienen los datos a estudiar. En *modelo* se encuentran las redes neuronales que modifican el *input* para generar el *output*.

Dejando de lado las posibles analogías con las redes neuronales del cerebro humano, en pocas palabras cabe decir que las redes neuronales, en el ámbito que

nos ocupa, son funciones matemáticas que transforman el *input* en el deseado *output*.

Siendo un poco más preciso, y siguiendo a Jordi Torres, en la obra citada, la terminología básica pertinente permite distinguir los siguientes conceptos.

1.- *label*, la variable dependiente, *y*, la que se pretende predecir con el modelo
2.- *feature*, variable de entrada: la variable *x*, con los datos a utilizar por el modelo
3.- *model*, modelo, es decir, el conjunto definitorio de las relaciones entre *label* y *feature*, generadoras del *output*.

El modelo presenta dos fases de ejecución:

training, fase de aprendizaje, a partir de los datos de entrada

inference, fase de inferencia (predicción) al aplicar el modelo a los datos de validación.

En la fase de aprendizaje, hay que distinguir dos tipos del mismo: *aprendizaje supervisado* y *aprendizaje no supervisado*.

En el *aprendizaje supervisado*, los datos de entrada, incluyen la *x*, junto con la variable, *label*. En el no-supervisado, no se incluye el *label*.

Para ilustrar lo dicho, consideremos, la siguiente relación lineal entre *feature* y *label*

$$y = wx + b$$

y: es el *label*, la etiqueta de entrada, el objetivo a predecir por el modelo
x: es el *feature*, la variable de este ejemplo
w: en este caso, es la pendiente de la recta, es el símbolo de las ponderaciones o *weights*, en *Machine Learning*. Es uno de los parámetros que tiene que aprender o descubrir el modelo durante la fase de entrenamiento.
b: aquí, es la ordenada en el origen de la recta. En el ámbito del *Machine Learning* recibe el nombre de sesgo, *bias*. Es otro de los parámetros a descubrir por el modelo.

En la práctica podemos tener muchas *features* o variables de entrada, cada una con su propia ponderación, por lo que el anterior modelo pasaría a ser

$$y = w_1x_1 + w_2x_2 + w_3x_3 + b$$

y, en general,

$$y = \sum_i w_i \, x_i + b$$

utilizando los vectores *X* e *W*.

En la fase de entrenamiento del modelo, se descubren y aplican los valore de los *pesos* y *sesgos*.

En el aprendizaje supervisado, se aplican algoritmos de aprendizaje, para obtener los valores de estos parámetros, examinando muchos ejemplos etiquetados, es decir, con *x* e *y*, minimizando el error, o *loss*.

loss, representa la penalización de una mala predicción, de un mal ajuste. De momento, es una función matemática que suma los errores individuales de los ejemplos de entrada al modelo.

Tomando en consideración a *loss* el proceso de aprendizaje consiste basicamente en ajustar las *ponderaciones* y *sesgos* de manera que sea mímo el valor del *loss*.

Otro concepto a mencionar es *overfitting*, que se produce cuando el modelo obtenido se ajusta tanto a los datos de entrada que no puede realizar predicciones correctas al utilizar nuevos datos, los datos de validación. Esta última afirmación requiere mencionar otra idea fundamental: la *partición* de los datos a analizar en dos bloques, el bloque de aprendizaje o entrenamiento, *train* y el bloque de validación, el bloque *test*.

III. REDES NEURONALES ARTIFICIALES Y DEEP LEARNING

Las redes neuronales artificiales son estructuras algorítmicas que permiten establecer modelos compuestos de múltiples capas de procesamiento, para realizar las transformaciones lineales y no lineales, que a partir de los datos de entrada en una capa, generan el output para la capa siguiente.

Una red neuronal muy simple es la representada en la figura 1.

Esta red neuronal tiene tres capas o *layers*. La capa de entrada, o *layer* input, recibe los datos de entrada y genera una salida o *layer* output. En medio quedan las capas ocultas *hidden layers*, aquí sólo una. En la práctica puden ser muchas las capas, de ahí que se hable de capas apiladas. Es esta posible colección de capas lo que permite hablar de *Deep learning*, *apredizaje profundo* aplicado a la red.

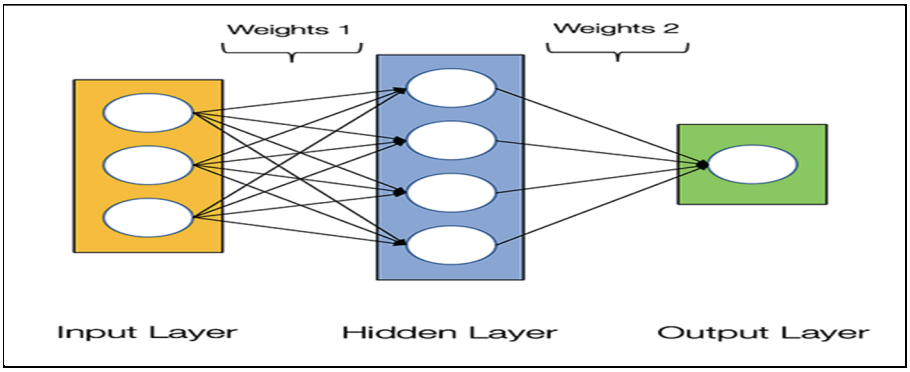

Figura 1. Red neuronal simple

Las neuronas, los círculos, de las distintas capas, están entrelazadas por las ponderaciones o *weights* y los sesgos respectivos, transformaciones que ejecutan las neuronas de la capa anterior para pasarlas a la capa siguiente.

La reunión de todas la capas permiten descubrir los patrones de comportamiento, y conviene señalar lo arduo que puede ser la programación de la red. (Cf. J. Torres, o.c., p. 56).

Hay que poner de relieve que el auge actual del *Deep Learning* no se debe sólo al apoyo de las grandes empresas usuarias como Apple, Amazon, Google, Facebook, etc., sino, y sobre todo, a la aparición de entornos de software, de código abierto, que permiten la creación y entrenamiento de estos modelos. Como ejemplos tenemos: *Keras*, *TensorFlow*, *PyTorch*, y otros. Originariamente, *Keras* era una biblioteca de algoritmos desarrollada en Python, y actualmente, desde 2017, también se encuentra en el entorno ***R***.

IV. SERIES TEMPORALES Y REDES NEURONALES

En la modelización de las series temporales se utilizan las redes neuronales capaces de aprender, descubrir, la regla que relaciona el futuro del sistema con su pasado

$$x_{t+1} = f(x_1, x_2, \dots, x_t)$$

En f se codifican las relaciones dinámicas del sistema para predecir el futuro.

En la práctica, es muy difícil establecer una función que dependa de todos las observaciones pasadas. Por ello, la red asume que la información sobre las observaciones

$$x_1, x_2, \dots, x_{t-1}$$

se puede codificar en un vector h_t, en cuyo caso, la ecuación actualizadora se simplifica, pasando a ser

$$x_{t+1}, h_{t+1} = f(x_t, h_t)$$

Señalar que la función f se aplica a todos los momentos t. Es decir, se supone que la serie temporal es estacionaria, o, en otras palabras, que la arquitectura dinámica no cambia en el tiempo.

Hay que destacar la importancia de las redes neuronales en el análisis de las series temporales, y, en concreto, de las redes neuronales recurrentes. Su importancia se debe al *teorema de la convergencia universal* que establece que la redes neuronales conexionadas en profundidad son poderosos medios de aproximación a cualquier función arbitraria (Cf. Ramsundar, B. y Zadeh, R. B. *TensorFlow for Deep Learning*, p. 87, 2018).

Las redes neuronales diseñadas para permitir el apilamiento profundo de múltiples capas, se denominan *Redes Neuronales Recurrentes* (RNN, en inglés). Estas redes permiten aplicar la transformación establecida a todas las capas de la red.

Un inconveniente de las RNN es que las señales alejadas en el tiempo se debilitan con rapidez. Una solución a esta eventualidad, ha sido la arquitectura *long-short term memory*: LSTM, que permite el paso de los estados, a través del momento presente, con mínimos cambios (Cf. Ramsundar, B. y Zadeh, R. B., o. c., p. 152).

Las redes *LSTM* conllevan ecuaciones con múltiples términos, lo que ha llevado al diseño de *hardware* apropiado, como, por ejemplo, Entel GPU, Nvidia GPU, etc.

V. MODELOS PARA SERIES TEMPORALES

En la formulación *AR(p)* de un modelo de series temporales,

$$x_t = w_0 + \Sigma_{i=1}^{p} w_i x_{t-i} + \epsilon_t$$

la serie x_t en el momento *t* es una regresión lineal con las *p* observaciones precedentes, más un término de error: ϵ_t.

Cabe generalizar la idea del modelo lineal para establecer una función *f* que relacione x_t con las observaciones precedentes, es decir

$$x_t = f(x_{t-1}, x_{t-2} \dots x_{t-p})$$

Desde el punto de vista del *Deep Learning* cabe desarrollar la función *f* como una red neuronal, que sometida al proceso de aprendizaje, permita transformar el input en el output.

Keras es una aplicación que permite diversos tipos de arquitecturas para *f* y, en definitiva, para las redes neuronales, así como su entrenamiento, utilizando diversos optimizadores. En el nivel más bajo, la aplicación utiliza *C*, *C++*, FORTRAN, etc.

Keras opera sobre *TensorFlow*, desarrollado por Google, *CNTK*, desarrollado por Microsoft y *Theano*, originalmente desarrollado por la Universidad de Montreal, Canadá.

VI. MULTILAYER-PERCEPTRON

Es la red básica de la arquitectura de las redes neuronales. Tres son sus componentes básicos, (cf. figura 2)

- input layer
- conjunto de layers ocultos
- output layer.

El input layer consiste en un vector de regresores o *features*. Por ejemplo, las anteriores observaciones precedentes: $(x_{t-1}, x_{t-2}, \dots, x_{t-p})$

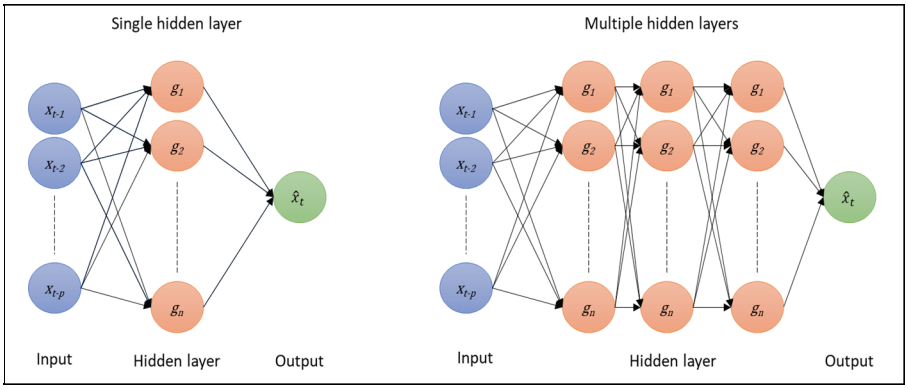

Figura 2. Dos tipos de redes neuronales.

En la representación de la izquierda, la red sólo tiene un *hidden layer*, en la figura de la derecha la red tiene múltiples *hidden layers*. En ambos casos, las diferentes capas reciben como input el output de las capas anteriores, en las que tras el ajuste de poderaciones y sesgos, generan el output $\hat{x}_t$

Las variables, *feature* input, acceden al *hidden layer* con *n* neuronas, y cada una aplica una transformación activadora, lineal o no lineal, que genera el *output* g_i

$$g_i = h(w_i x + b_i)$$

siendo w_i y b_i los pesos y sesgos de la transformación activada por *h.*

La función activadora *h* permite a la red modelizar las complejas relaciones entre los regresores, *features* y la variable objetivo. Las dos funciones *h* más utilizadas son la *sigmoide* y la *tanh*.

La función *sigmoide*

$$h = \frac{1}{1 - e^{-x}}$$

reduce los resultados de las transformaciones a un número real en el rango [0,1], indicando el *cero* el bloqueo de la transformación, y el *uno* el paso de la misma. Por esta propiedad, este activador *h* es más utilizado en los modelos de clasificación.

La función *tanh*

$$h = \frac{1 - e^{-x}}{1 + e^{-x}}$$

reduce cualquier número real al intervalo $[-1,1]$. En determinadas circustacias, la función *h* puede ser lineal o incluso la función identidad.

El caso más simple de una red neuronal, viene representado en la figura 3. Es una red con un sólo *hidden* layer, con dos neuronas, cada una, con su función activadora *h: g1* y *g2*.

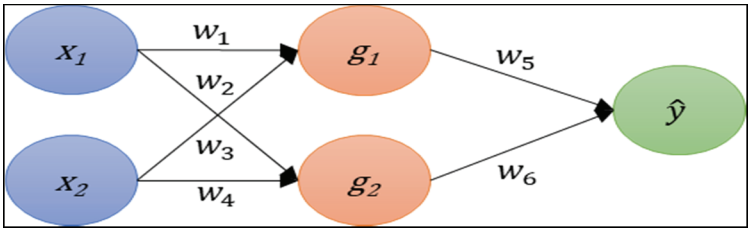

Figura 3. Red neuronal sencilla.

La red recibe dos variables input: (x_1, x_2). La red realiza una serie de sumas y multiplicaciones y con las funciones *h* activadoras, transforma el input en la predicción $\hat{y}$.

Las transformaciones del *input* en la predicción $\hat{y}$ constituye el paso *forward* de la fase de entrenamiento o aprendizaje. Concluida la cual, el algoritmo *backpropagation* entra en acción en lo que se denomina la pasada *backforward pass*.

En el caso de múltiples *hidden layers* las neuronas generan sus respectivos *outputs* que constituyen el *input* para la siguiente capa oculta. La última capa trasmite su transformación al layer *output*.

Las capas ocultas reciben el adjetivo de *layers* plenamente conectados: *dense layers*, siendo conveniente destacar el enorme número de ponderaciones y sesgos a calcular. Si *p* es el número de *features* input de la red, y el número de neuronas: n_1, n_2, n_3, el número de ponderaciones viene dado por $p \times n_1 + n_1 \times n_2 + n_2 \times n_3 + n_3$. (Cf. Pal, A. y Prakash, PKS., *Practical Time Series Analysis*, p.168, 2017).

En el proceso de aprendizaje, las ponderaciones son obtenidas mediante algoritmos de optimización, que minimizan el error o *loss* al recorrer los datos del *bloque train*. El error cuadrático medio, MSE en inglés, o el error mínimo absoluto, MAE en inglés, son las funciones utilizadas para tareas de regresión, y predicción en las series temporales, mientras que *logloss* se utiliza para tareas de clasificación.

Durante la fase de aprendizaje, las ponderaciones son iniciadas aleatoriamente, bien mediante el uso de la distribución uniforme, o mediante la distribución normal, con media cero y varianza uno.

El cálculo de las ponderaciones se repite múltiples veces. El nombre con que se indica su número es el de *epochs*, número de pasadas, *forward* lo largo del bloque de datos *training*.

Afortunadamente existe Keras, Tensorflow, Theano y CNTK que realizan estas operaciones y sus correspondientes representaciones gráficas orientadoras de los cálculos a realizar.

Tras la larga exposición anterior, pasamos a su aplicación, utilizando los datos de la producción de energía eléctrica fotovoltaica en España, y siguiendo los pasos del *script* en **R** de Richard Wanjohi, mencionado en la bibliografía.

VII. LA PRODUCCIÓN DE ENERGÍA ELÉCTRICA FOTOVOLTAICA EN ESPAÑA

Se trata de establecer un modelo de redes neuronales para la predicción, o mejor, para la explicación de la producción de energía eléctrica fotovoltaica, utilizando los datos de Red Eléctrica Española (REE).

Los primeros valores de la serie, en GWh,, con sus fechas, se muestran en la siguiente tabla 1:

Fecha	GWh
Enero-07	0.0
Febrero-07	0.1
Marzo-07	0.1
Abril-07l	0.1
Mayo-07	0.2
Junio-07	0.2
Julio-07	0.2
Agosto-07	0.2

Tabla 1. Primeros valores de la serie

Hasta el año 2007, la generación de electricidad fotovoltaica era insignificante. Desde este año, y hasta septiembre de 2018, los datos mensuales vienen representados en la figura 4. Es una *Serie* de 147 observaciones mensuales.

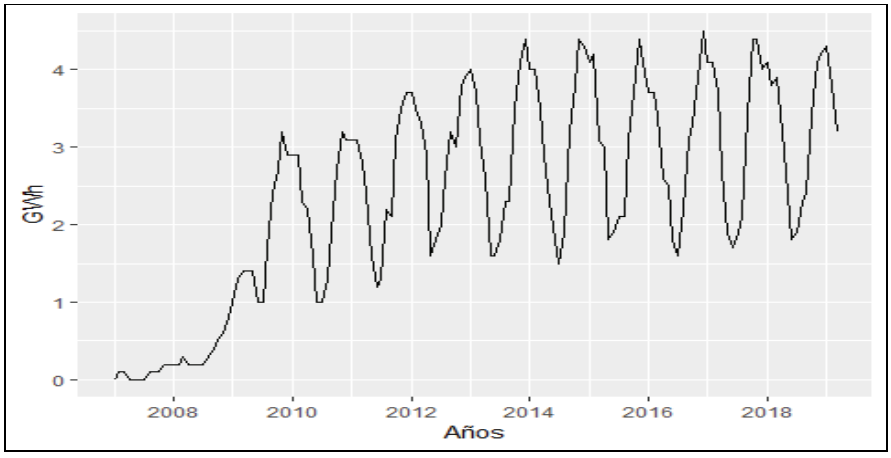

Figura 4. Producción de energía eléctrica fotovoltaica: 2007 – 2018.

Primer paso

A la hora de establecer un modelo para esta serie, cabe distinguir los siguientes pasos.

En este primer paso se preparan los datos en el formato apropiado para su tratamiento por Keras. Es el momento en que se particionan los datos en dos bloques: el bloque de entrenamiento o aprendizaje, *train*, y el bloque de validación, *test*.

Segundo paso

Aquí se define el modelo según Keras, y se ajusta el modelo establecido a los datos de entrenamiento, los del bloque *train*.

Tercer paso

Es el momento de presentación de los resultados

Primer paso

Se comienza cargado en **R** las biblioteca Keras.

library(keras)

En pos de lograr la estacionariedad de la serie, se toman primeras diferencias:

diffed = **diff**(Series, differences = 1)

Para utilizar LSTM los datos tienen que presentarse en modo *supervisado*. Es decir, junto a la variable objetivo, la y, el *label*, tiene que estar la serie x, el predictor. Para lograr el objetivo se retrasan las series al momento $t - k$.

Retardando los datos $k = 1$ tenemos el conjunto de datos en el formato propiado para el modo de *procesamiento supervisado*.

```
lag_transform <- function(x, k){
   lagged =  c(rep(NA, k), x[1:(length(x)-k)])
   DF = as.data.frame(cbind(lagged, x))
   colnames(DF) <- c( paste0('x-', k), 'x')
   DF[is.na(DF)] <- 0
   return(DF)
}
```

Llamamos *supervised* a los datos transformados, y con *head (supervised)* podemos ver sus seis primeros valores

```
head(supervised)
##    x-1    x
## 1  0.0  0.1
## 2  0.1  0.0
## 3  0.0 -0.1
## 4 -0.1  0.0
## 5  0.0  0.0
## 6  0.0  0.0
```

El paso siguiente consiste en particionar los datos en dos bloques: el bloque de aprendizaje, *train* y el bloque de validación, *test*. Dado que el ordenamiento de las observaciones en una serie temporal es intocable, la partición no se puede hacer de forma aleatoria, como es lo normal, en *Deep Learning*. En esta ocasión, el 70% de los datos pasan a integrar el bloque de apredizaje, quedando el 30% restante en el bloque de validación.

```
N = nrow(supervised)
n = round(N * 0.7, digits = 0)
train = supervised[1:n, ]
test  = supervised[(n+1):N,  ]
```

El conjunto de datos *train* consta de 102 filas y dos columnas.

Las seis primeras observaciones del conjunto de entrenamiento, *train,* son:

```
head(train)
##   x-1    x
##   0.0  0.1
##   0.1  0.0
##   0.0 -0.1
##  -0.1  0.0
##   0.0  0.0
##   0.0  0.0
```

El bloque de datos *test* tiene 44 filas y dos columnas.

Las seis primeras observaciones del bloque de validación, *test,* son:

```
head(test)
##   x-1   x
##  0.2  0.0
##  0.0  1.0
##  1.0  0.6
##  0.6  0.7
##  0.7 -0.3
## -0.3 -0.4
```

A continuación se normalizan los datos en los dos pasos siguientes. Con la función *scale_data* se logra el objetivo. Hay que señalar que el mínimo y máximo de los datos se utilizan como coeficientes para ambos bloques de datos, *train* y *test*, así como para la predicción. Ello nos asegura que ambos valores no influyen en el modelo.

Primer paso

```
scale_data <- function(train, test, feature_range = c(0, 1)) {
    x = train
    fr_min = feature_range[1]
    fr_max = feature_range[2]

    std_train = ((x - min(x) ) / (max(x) - min(x)  ))
    std_test  = ((test - min(x) ) / (max(x) - min(x)  ))

    scaled_train = std_train *(fr_max - fr_min) + fr_min

    scaled_test = std_test *(fr_max - fr_min) + fr_min

    return( list(scaled_train = as.vector(scaled_train), scaled_test =
    as.vector(scaled_test) ,scaler= c(min =min(x), max = max(x))) )
    }
```

Segundo paso

```
Scaled = scale_data(train, test,c(-1,1))

y_train = Scaled$scaled_train[,2]
x_train = Scaled$scaled_train[,1]

y_test = Scaled$scaled_test[,2]
x_test = Scaled$scaled_test[,1]
```

Como el objetivo del análisis son los datos en su escala original, la transformación oportuna se logra con la función *invert_scaling*.

```
invert_scaling = function(scaled, scaler, feature_range = c(0, 1)){
  min = scaler[1]
  max = scaler[2]

  n = length(scaled)

  mins = feature_range[1]
  maxs = feature_range[2]

  inverted_dfs = numeric(n)

  for( i in 1:n){
    X = (scaled[i]- mins)/(maxs - mins)
    rawValues = X *(max - min) + min
    inverted_dfs[i] <- rawValues
  }
  return(inverted_dfs)
}
```

Paso segundo

Tras los pasos anteriores, llega el momento de establecer el modelo.

El input del modelo debe ser tridimensional, indicando el número de observaciones a utilizar, el número de etapas, y el número de *features*. Aquí: 102,1,1. Además, el tamaño del batch: *batch_size*, por ser una serie temporal, procesa uno a uno sus valores: $batch\_size = 1$, por tanto la dimensión del x_train se establece mediante

```
dim(x_train) <- c(length(x_train), 1, 1)

 dim(x_train)

##  102   1   1

# argumentos
 X_shape2 = dim(x_train)[2]
 X_shape3 = dim(x_train)[3]
 batch_size = 1
 units = 1
```

Hecho lo anterior, se pasa a la definición del modelo. Con ayuda de Keras. La instrucción *keras_model_sequential* establece la arquitectura que permite agregar las capas que se consideren necesarias. Aquí la capa o *layer_lstm*, y la capa *layer_dense*. En este paso, también se establece *loss*, el algoritmo optimizador *adam,* y la métrica del proceso: *accuracy*. Las instrucciones oportunas son:

```
model <- keras_model_sequential()
model%>%
  layer_lstm(units, batch_input_shape = c(batch_size, X_shape2, X_shape3),
stateful= TRUE)%>%
  layer_dense(units = 1)
model %>% compile(
  loss = 'mean_squared_error',
  optimizer = optimizer_adam( lr= 0.02, decay = 1e-6 ),
  metrics = c('accuracy')
)
```

Con *summary(model)* obtenemos el resumen del modelo:

```
summary(model)
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## lstm_1 (LSTM)                (1, 1)                    12
## _________________________________________________________________
## dense_1 (Dense)              (1, 1)                    2
## =================================================================
## Total params: 14
## Trainable params: 14
## Non-trainable params: 0
## _________________________________________________________________
```

Dentro de este segundo paso, llega el momento de ajustar el modelo a los datos de entrenamiento. La instrucción es *fit.*

```
Epochs = 50
nb_epoch = Epochs

for(i in 1:nb_epoch ){
```

```
  model %>% fit(x_train, y_train, epochs=1, batch_size=batch_size, verbose
=1,   shuffle=FALSE)

model %>% reset_states()
}
```

Epochs: indica el número de veces que se utilizan todos los datos del bloque train en la fase de aprendizaje. (Cf. Torres, o. c., p. 128).

fit es la instrucción que arranca el procedimiento de aprendizaje.

accuracy: indica la fracción de veces que la predicción coincide con el dato de entrenamiento. (Cf. Torres, o. c., p. 101).

Tercer paso: Resultados

Con el nombre de *predicciones* se presentan los datos ajustados

```
L = length(x_test)
dim(x_test) = c(length(x_test), 1, 1)
scaler = Scaled$scaler
predictions = numeric(L)

 for(i in 1:L){
 X = x_test[i , , ]
 dim(X) = c(1,1,1)
  # forecast
 yhat = model %>% predict(X, batch_size=batch_size)

  # invert scaling
  yhat = invert_scaling(yhat, scaler,  c(-1, 1))

  # invert differencing
  yhat  = yhat + Series[(n+i)]

  # save prediction
  predictions[i] <- yhat
 }
```

Gráficamente, el resultado del ajuste se presenta en la figura 5.

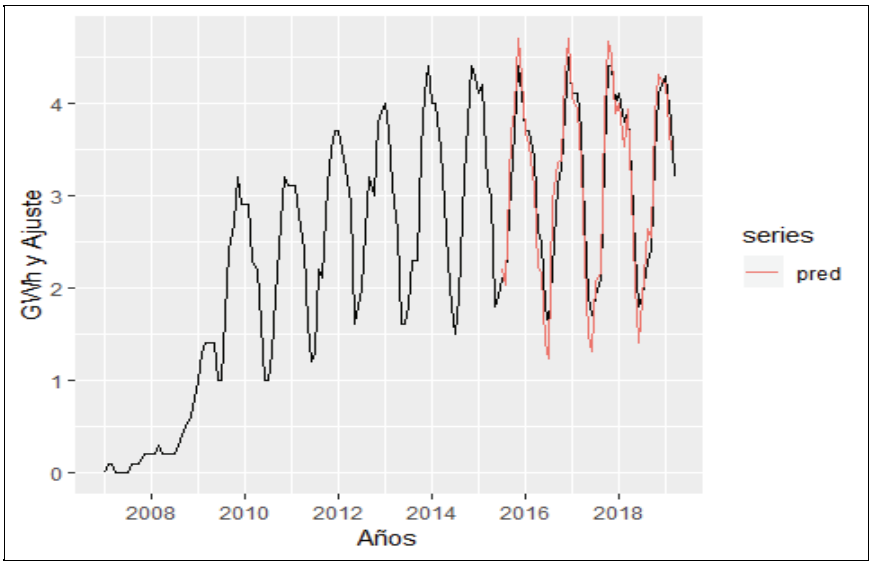

Figura 5. Serie original y serie ajustada

Las 44 observaciones del bloque *test* sirve para que el modelo genere unas predicciones o ajustes, que al representarlas junto con los datos de la serie original, muestran un ajuste que podríamos decir casi perfecto. El error cuadrático medio de esta ejecución muestra el valor $ECM = 0.1031$.

VIII. CONCLUSIÓN

Esta representación gráfica de la figura 5, permite augurar un futuro prometedor para el análisis de las series temporales con ayuda del *Deep Learning*.

En la WEB aparecen ya numerosos ejemplos de utilización de *Deep Learning* para la predicción de series temporales; en la bibliografía se mencionan algunos. Limitaciones de espacio no permiten ahondar más en el tema.

IX. BIBLIOGRAFÍA

- ALONSO RODRÍGUEZ, A., Forecasting Economic Magnitudes with Neural Networks Models, en *International Advances in Economic Research*, 5 (1999) 496-511.

- AUNGIERS, J., *Time Series Prediction using LSTM Deep Neural Networks*, https://jakob.aungiers.com/articles, 1 de septiembre, 2018.

- BROWNLEE, J., *Deep Learning for Time Series Forecasting*, eBook, 2018.

- CHOLLET, F., *Deep Learning with Python*, Manning Publications, Shelter Island, NY., 2018.

- CHOLLET, F., y ALLAIRE, J. J., *Deep Learning with R*, Manning Publications, Shelter Island, NY., 2018.

- COLAH'S BLOG, *Understanding LSTM Networks*, http://colah.github.io/posts/2015-08-Understanding-LSTMs/ , 27 de agosto, 2015.

- HYNDMAN, R. J., y ATHANASOPOULOS, G., *Forecasting: Principles and Practice* segunda edición, OTexts, 2018.

- MERLINO, A., "How to build your own Neural Network from scratch", in *R, Posts on Tychobra*, 8 de octubre, 2018.

- PAL, A., y PRAKASH, PKS., *Practical Time Series Analysis*, Packt Publishing, Birminham, 2017.

- PANT, N., *A Guide for Time Series Prediction Using Recurrent Neural Networks (LSTMs)*, https://blog.statsbot.co/time-series-prediction-using-recurrent-neural-networks-lstms-807fa6ca7f, 7 de septiembre, 2017.

- RAMSUNDAR, B. , y ZADEH, R. B., *TensorFlow for Deep Learning* , O'Reilly Media, Sebastopol, CA., 2018.

- SINGH, A., *Stock Prices Prediction Using Machine Learning and Deep Learning Techniques (with Python codes)*, https://www.analyticsvidhya.com/blog/, 25 de octubre, 2018.

- TORRES, J., *Deep Learning, Introducción práctica con Keras Primera Parte*, tercera edición, colección Watch this Space, Kindle Direct Publishing, Amazon, 2018.

- WANJOHI, R., *Time Series Forecasting using LSTM in R*, http://rwanjohi.rbind.io/2018/04/05/time-series-forecasting-using-lstm-in-R/ , 5 de abril, 2018.