# Perceptrón multicapa regresión

Para esta implementación vamos a utilizar la función

```
sklearn.neural_network.MLPRegressor(
hidden_layer_sizes=100, activation='relu', *, solver='adam',
alpha=0.0001, batch_size='auto', learning_rate='constant',
learning_rate_init=0.001, power_t=0.5, max_iter=200,
shuffle=True, random_state=None, tol=0.0001,
verbose=False, warm_start=False, momentum=0.9,
nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1,
beta_1=0.9, beta_2=0.999, epsilon=1e-08,
n_iter_no_change=10, max_fun=15000)
```

de la biblioteca de `sklearn`

https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html

(Añadir enlace a la bibliografía más adelantes )

Además utilizaremos los siguientes argumentos:

- `hidden_layer_sizes` número de unidades por capa en el rango 50-100, que afinaremos por validación cruzada. Usaremos tres capas, ya que es lo que se nos pide, esto además nos parece coherente ya que con una capa oculta ya sabemos que es un aproximador universal ( Hornik, Kurt; Tinchcombe, Maxwell; White, Halbert (1989). Multilayer Feedforward Networks are Universal Approximators (PDF). Neural Networks, Vol. 2, pp. 359-366, 1989. Pergamon Press plc.).

- `activation`: `logistic` sería indiferente usar una `tanh`, ninguna presenta ninguna ventaja frente a otra.

- `solver` la técnica para minimizar `adam` ya que según la documentación este método es el que funciona mejor con miles datos como es nuestro caso, profundizaremos en este método más adelante.

- `alpha` método de regularización.

- `learning_rate: {'constant', 'invscaling', 'adaptative'}`.

- `learning_rate_init` aquí si hay que utilizarlo.

## Explicación del método de minimización de `adam`

Bibliografía:


Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

Es un método basado en la optimización de gradiente descendiente. Requiere de gradiente de primer orden.

Las ventajas que supone frente al sgd clásico, según el propio artículo de publicación son las siguientes:

- Computacionalemente eficiente.

- En memoria también es eficiente.

- Invariante a reescaladodos de la diagonal del gradiente. (Aunque esto no nos afecta en nuestro problema).

- Apropiado para objetivos no estacionarios, en nuestro caso la documentación ofrecida por UCI, no nos proporciona información explícita de alguno de nuestros datos se correspondan a este tipo. Sin embargo al leer los variables que se han analizado, no sería descabellado. De todas formas haremos una compatativa con el sgd tradicional.
- Apropiado para problemas con mucho ruído.

Además como heurística en la documentación del sklearn se recomendaba para tamaños de entrenamiento de miles, como el nuestro.

Información consultada sobre datos no estacionarios: Bibliografía: - https://boostedml.com/2020/05/stationarity-and-non-stationary-time-series-with-applications-in-r.html
- https://www.investopedia.com/articles/trading/07/stationary.asp

El algoritmo indicado en el artículo de 2015 donde se publicó es el siguiente:

.

## Exploración inicial de los datos

Comenzaremos un estudio preliminar ajustando tan solo el tamaño de dos capas ocultas y el método de minimización, si Adam o SGD.

Los valores fijados han sido:
- `max_iter = 500` El número de iteraciones máximas.
- `shuffle = True` Desordena los datos en cada iteración, es una heurísitica para mejorar la convergencia.
- `activation = 'logistic'` Función logística que devuelve $f(x) = \frac{1}{1+exp(-x)}$. Vista en teoría justo a `tahn`, no hay ningún motivo para el que preferir una sobre otra.
- `alpha = 0.0001` Por defecto la función aplica regularización a ese valor, como es una primera aproximación lo dejamos fijo.

Los resultados obtenidos son los siquientes:

**Algorithm 1:** *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. $g_t^2$ indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With $\beta_1^t$ and $\beta_2^t$ we denote $\beta_1$ and $\beta_2$ to the power $t$.

**Require:** $\alpha$: Stepsize
**Require:** $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
**Require:** $f(\theta)$: Stochastic objective function with parameters $\theta$
**Require:** $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  **while** $\theta_t$ not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  **end while**
  **return** $\theta_t$ (Resulting parameters)

Figura 1: Descripción del algoritmo de minimización estocástico de Adam

Mejores parámetros: {‘hidden_layer_sizes’: (100, 50), ‘solver’: ‘adam’} Con una $R^2$ de: 0.6205065254947334

Tabla 1: Comparativa preliminar número de capas método

| Parámetros | $R^2$ medio | Des típ $R^2$ | Ranking | t medio |
|---|---|---|---|---|
| hidden_layer_sizes (100, 50) solver adam | 0.6205 | 0.0135 | 1 | 2.2745 |
| hidden_layer_sizes (100, 75) solver adam | 0.6188 | 0.0158 | 2 | 2.5420 |
| hidden_layer_sizes (50, 100) solver adam | 0.6182 | 0.0134 | 3 | 1.7802 |
| hidden_layer_sizes (75, 75) solver adam | 0.6175 | 0.0133 | 4 | 2.5021 |
| hidden_layer_sizes (100, 100) solver adam | 0.6170 | 0.0174 | 5 | 3.4487 |
| hidden_layer_sizes (75, 50) solver adam | 0.6168 | 0.0166 | 6 | 2.0171 |
| hidden_layer_sizes (75, 100) solver adam | 0.6137 | 0.0146 | 7 | 2.6748 |
| hidden_layer_sizes (50, 50) solver adam | 0.6131 | 0.0117 | 8 | 1.2846 |
| hidden_layer_sizes (50, 75) solver adam | 0.6087 | 0.0202 | 9 | 1.8376 |
| hidden_layer_sizes (100, 50) solver sgd | 0.0731 | 0.0142 | 10 | 1.5981 |
| hidden_layer_sizes (50, 75) solver sgd | 0.0724 | 0.0067 | 11 | 0.9025 |
| hidden_layer_sizes (75, 75) solver sgd | 0.0287 | 0.0103 | 12 | 1.5344 |
| hidden_layer_sizes (50, 50) solver sgd | 0.0129 | 0.0075 | 13 | 0.8827 |
| hidden_layer_sizes (75, 100) solver sgd | 0.0121 | 0.0046 | 14 | 1.8657 |
| hidden_layer_sizes (100, 75) solver sgd | 0.0091 | 0.0055 | 15 | 2.0079 |
| hidden_layer_sizes (75, 50) solver sgd | -0.0277 | 0.0060 | 16 | 0.9771 |
| hidden_layer_sizes (50, 100) solver sgd | -0.0383 | 0.0095 | 17 | 1.2715 |
| hidden_layer_sizes (100, 100) solver sgd | -0.0448 | 0.0092 | 18 | 2.1985 |

Es notoria la diferencia entre utilizar un método de minimización u otro, de hecho esto nos hace plantearnos qué puede estar pasando con el sgd ¿Son insuficientes el número de iteraciones? Sea como fuere, el métodod de minimización de `adam` es mejor así continuaremos trabajando con él para comprobar si podemos refinarlo.

### Estudiaremos ahora el learning rate

Los argumentos fijos son los anteriores, cambiando el máximo número de iteraciones a `200` y añadiento el mejor resultado anterior `hidden_layer_sizes = (100, 50)` y `solver = 'adam'`.

Los resultados son los siguientes:

Tabla 2: Error cambiando la tasa de aprendizaje

| Parámetros | $R^2$ medio | Desviación tipica $R^2$ | Ranking | tiempo medio ajuste |
|---|---|---|---|---|
| learning_rate_init 0.001 | 0.6205 | 0.0135 | 1 | 2.2736 |
| learning_rate_init 0.0001 | 0.5418 | 0.0210 | 2 | 3.5609 |
| learning_rate_init 0.1 | 0.4921 | 0.0603 | 3 | 7.0441 |
| learning_rate_init 0.01 | 0.4705 | 0.0327 | 4 | 7.6932 |
| learning_rate_init 1 | -0.4716 | 0.7183 | 5 | 3.8819 |

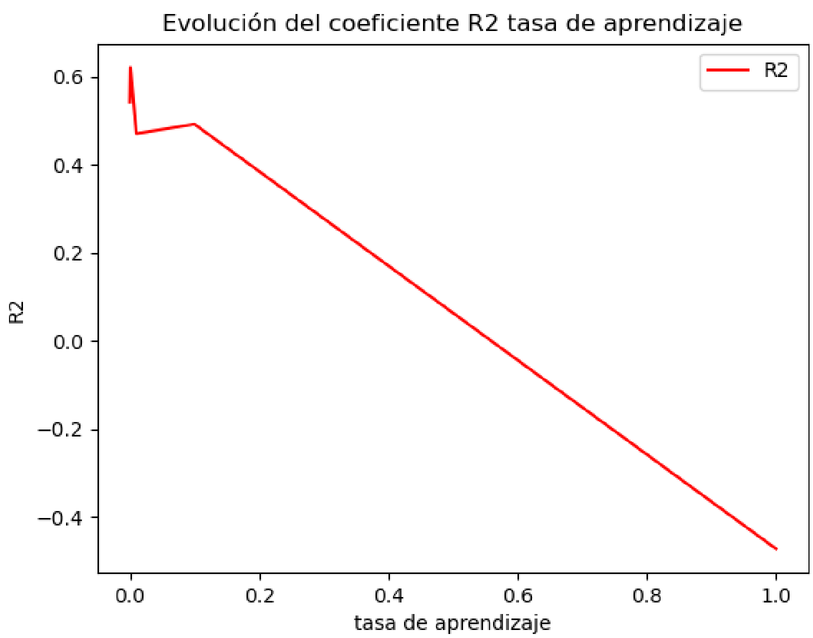


Figura 2: Variación de $R^2$ conforme aumenta el learning rate

.

El `learning_rate_init 0.0001` tiene un $R^2$ probablemente por el número de iteraciones, no vamos a estudiar su comportamiento aumentando el número de iteraciones porque con un `learning_rate_init` de `0.001` se ha conseguido el mismo $R^2$ que en el mejor de los casos anteriores y en menos iteraciones, luego resulta más interesante su exploración.

Dicho esto observamos que en un intervalo entre (0.0001, 0.01) se encuentra el mejor valor asociado a la tase de variación.

Si bien en los experimentos realizados hasta ahora mantenemos constante el método de adaptación del `learing rate`. Podría contemplarse un estudio utilizando un criterio adaptativo: `adaptative`.
Éste consiste en un método de actualización de la tasa de aprendizaje: la mantiene constante siempre que el error de entrenamiento siga decreciendo, si se dan dos épocas consecutivas en el que la variación es menor que la tolerancia entonces es dividida entre 5.

Sin embargo no es compatible con `Adam` según la documentación oficial, así pues estudiaremos la regularización.

Para ello mantenemos condiciones iniciales anteriores y variamos los valores de regularización, obteniendo con ellos los siguietne errores:

Tabla 3: Evolución $R^2$ con regularización.

| Parámetros | $R^2$ medio | Desviación tipica $R^2$ | Ranking | tiempo medio ajuste |
|---|---|---|---|---|
| alpha 0.01 | 0.6228 | 0.0129 | 1 | 1.8559 |
| alpha 0.001 | 0.6205 | 0.0135 | 2 | 1.7910 |
| alpha 0.0001 | 0.6205 | 0.0135 | 3 | 1.8227 |
| alpha 0 | 0.6205 | 0.0135 | 4 | 1.8057 |
| alpha 1 | 0.5338 | 0.0139 | 5 | 3.9193 |

Obtenemos por tanto que la regularización mejor está entre $(0{,}0001, 0{,}001)$ da resultados similares y la diferencia entra dentro de la desviación típica, así que cualquiera de las tres sería aceptada, a partir de ahora trabajaremos con la de 0.01.

.

Finalmente veamos que aumentar las iteraciones no mejora el error.

Los valores son

La tolerancia por defento es de 0.0001, la cual nos parece razonable y por eso la mantenemose

| Parámetros | $R^2$ medio | Desviación tipica $R^2$ | Ranking | tiempo medio ajuste |
|---|---|---|---|---|
| max_iter 50 | 0.6205 | 0.0135 | 1 | 1.8014 |
| max_iter 100 | 0.6205 | 0.0135 | 1 | 1.8023 |
| max_iter 200 | 0.6205 | 0.0135 | 1 | 1.7872 |

| Parámetros | $R^2$ medio | Desviación tipica $R^2$ | Ranking | tiempo medio ajuste |
|---|---|---|---|---|
| max_iter 350 | 0.6205 | 0.0135 | 1 | 1.7923 |
| max_iter 10 | 0.5985 | 0.0173 | 5 | 0.7059 |

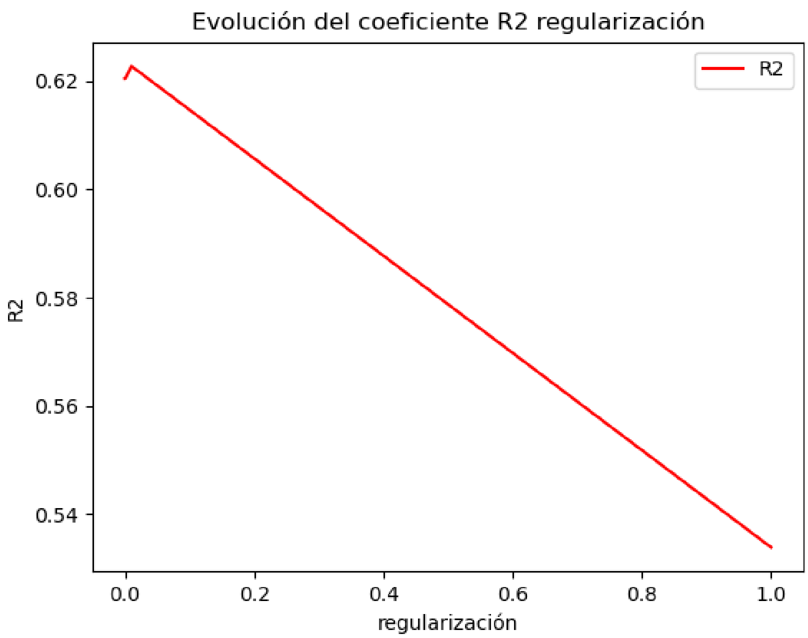


Figura 3: Variación de $R^2$ conforme aumenta el learning rate