SOFT-COMPUTING

Práctica Final

Juan José Córdoba Zamora
Juan Casado Ballesteros

Universidad de Alcalá

Índice

1. Introducción al problema

Una decisión será mejor cuanta más información se tenga para poderla tomar. Si pudiéramos tener información de cómo será el futuro las decisiones que tomemos serían mejores ya que tendríamos más información y además sería información que otros no podrían conocer. Conocer con exactitud el futuro resulta ciertamente complicado, no obstante realizar predicciones sobre él se antoja más asequible.

Deseamos poder estimar la demanda de energía que se producirá en años venideros a partir de variables macroeconómicas. Dichas variables suponemos contienen información suficiente como para poder realizar la predicción. Es muy importante intentar predecir con poco error cuanta energía se va a demandar en un futuro próximo, para así poder adelantarse y producir la cantidad de energía necesaria.

El problema concreto que resolveremos no es exactamente el de la predicción a futuro si no el de interpolación de la evolución del consumo de energía en Francia. Utilizaremos series de tiempo de las variables macroeconómicas y del consumo de energía para predecir a partir de las variables de los años pares.

Partición de los datos del problema:

Años	90	91	92	93	94	95	96	98	99	…	08	09	10	11	12	13	14	15	16
Variables																			
Energía																			

Entrenamiento -> Objetivo: entrenamos utilizando las variables macroeconómicas de los años pares de modo que se obtenga la energía de los impares.
Test -> Predicción: utilizamos las variables macroeconómicas de los años impares de modo que se prediga la energía de los pares, lo cual consiste en el test que nos indica si nuestro algoritmo está siendo capaz de predecir o solo se está aprendiendo los datos de entrenamiento.
No utilizamos: la primera energía par no se utiliza pues no tenemos variables macroeconómicas del año anterior para predecirla o aprenderla, las últimas variables macroeconómicas tampoco se utilizan pues no tenemos energía para poder comprobar nuestra predicción o para entrenar con ella.

Se han propuesto varias soluciones que se comentarán más adelante, únicamente comentar que con la red neuronal se ha entrenado de dos formas, utilizando validación cruzada, el test será solo una de las parejas variable-energía. Las otras que no se estén utilizando para test se incorporarán en el entrenamiento. La pareja utilizada como test será una distinta en cada entrenamiento.
Y la otra forma es sin utilizar validación cruzada, los datos se dividen de la misma forma que se ha comentado en la tabla, una vez divididos los datos de entrenamiento se dividen a su vez en 70% para entrenar y 30% para validar y que no sobreentrene.

Como aclaración, cuando indicamos fenotipo se refiere a los individuos de una población, es decir, a las filas y cuando decimos genotipo se refiere a los valores que tiene el individuo, es decir, a las columnas del fenotipo. Cuando nos referimos a población es una matriz con varios individuos, es decir, fenotipos.

2. Descripción del problema y datos utilizados para resolverlo

Para la resolución de este problema se han utilizado variables macroeconómicas que describen cómo es la economía de un país.
Las variables económicas que se han utilizado para resolver este problema están comprendidas entre los años 1990 y 2016:

- **Target**: Demanda total de la energía: Variable objetivo medida en toneladas equivalentes de petróleo.
- **V1**:Población: Número de habitantes de Francia.
- **V2:**GDP: Gross domestic product (producto interior bruto), medido en trillones de dólares estadounidenses.
- **V3** :CO2: Emisiones de dióxido de carbono medidas en toneladas.
- **V4**: Electricidad producida: Medida en teravatio hora.
- **V5**: Electricidad consumida: Medida en teravatio hora.
- **V6**: Ratio de desempleo: Porcentaje de habitantes desempleados en Francia.

2.1. Función a optimizar

Una vez explicadas las variables utilizadas en el problema, se procede a explicar el problema de forma matemática.
El problema se reduce a optimizar una función, para este problema se ha utilizado la función del valor cuadrático medio entre la predicción realizada y los valores reales.

$$f(M) = \frac{1}{n*}\sum_{j=1}^{n*}(E(j) - \hat{E}(j))^2$$

Donde:
M significa el modelo.
$n*$ significa número de variables del problema
E significa la energía real.
$\hat{E}$ significa energía predicha.

2.2. Modelo

El modelo consiste en una función de parámetros ajustables por el algoritmo optimizador. Cuando el modelo es evaluado con unas entradas, las variables macroeconómicas, produce una salida, la energía predicha. Dicha energía predicha es la que posteriormente se utiliza en la función a optimizar para calcular el error con la energía real.
En definitiva el modelo enlaza los parámetros ajustados por el optimizador con la función que de se debe optimizar.

Para predecir la energía en este problema vamos a utilizar 4 diferentes modelos.

M1: Modelo exponencial (propuesto por el profesor):

$$\hat{E}(t+1) = \sum_{i=1}^{T} w_i * \ X_i(t)^{w_{i+1}} + w_{13}$$

Donde:
$\hat{E}(t+1)$ significa energía predicha del año siguiente.
T significa el número de años, en este caso del 1990 al 2016.
$X_i(t)$ significa el valor de la variable (normalizada).
w_i significa el peso de la variable con el dominio del -1 al 1.
w_{13} significa el parámetro bias con el dominio del 0 al 5.

M2: Modelo exponencial (propuesto por nosotros):

$$\hat{E}(t+1) = \sum_{i=1}^{T} (w_i * X_i(t))^{w_{i+m'}} + w_{13}$$

Donde:
$\hat{E}(t+1)$ significa energía predicha del año siguiente.
T significa el número de años, en este caso del 1990 al 2016.
$X_i(t)$ significa el valor de la variable (normalizada).
w_i significa el peso de la variable con el dominio del 0 al 1.
w_{13} significa el parámetro bias con el dominio del 0 al 5.

M3: Modelo lineal:

$$\hat{E}(t+1) = \sum_{i=1}^{T} w_i * X_i(t) + w_7$$

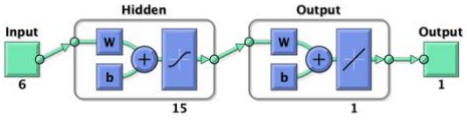
$\hat{E}(t+1)$ significa energía predicha del año siguiente.
T significa el número de años, en este caso del 1990 al 2016.
$X_i(t)$ significa el valor de la variable (normalizada).
w_i significa el peso de la variable con el dominio del -1 al 1.
w_7 significa el parámetro bias con el dominio del 0 al 5.

M4: Red neuronal:

Aunque este es un caso especial, porque no utiliza ninguna función, sino el modelo es la propia red neuronal ajustada con los pesos, lo que ponemos como modelo es la arquitectura de la red neuronal.

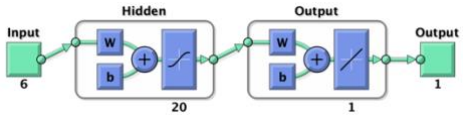
Con validación cruzada:

Para el modelo de red neuronal con validación cruzada se ha probado varias arquitecturas y con la que mejor resultados hemos conseguido ha sido 1 capa oculta con 15 neuronas más la capa de salida. La red neuronal utilizada ha sido *fitnet* utilizada para problemas de regresión.



Sin validación cruzada:

Después de varias pruebas realizadas hemos llegado a la conclusión de que con 1 capa oculta con 20 neuronas funciona muy bien para este problemas con estos datos. La red neuronal utilizada ha sido *fitnet* utilizada para problemas de regresión.



2.3. Optimizadores

Los optimizadores se encargan de ajustar los parámetros del modelo de modo que se minimice o maximice la función a optimizar. Para ello se utilizan diferentes técnicas de las que exploraremos 5. En realidad, se han implementado más técnicas de optimización, todas las presentadas en la práctica anterior pues el nuevo modelo y la nueva función a optimizar se han integrado con el código que ya se realizó para esa práctica.

Cuando nos referimos a fenotipo son los individuos de una población, es decir, a las filas y cuando decimos genotipo se refiere a los valores que tiene el individuo, es decir, a las columnas del fenotipo. Cuando nos referimos a población es una matriz con varios individuos, es decir, fenotipos.

O1: Temple simulado

En este algoritmo tendremos un único individuo. Dicho individuo recibirá mutaciones sobre sus genotipos con un ruido de tipo gaussiano. Si el individuo resultante tras la mutación es mejor que el individuo que se tenía anteriormente el nuevo sustituirá al anterior.
Como mecanismo de descubrimiento habrá una cierta probabilidad de que el nuevo individuo resultado de la mutación sustituya al anterior, aunque no sea mejor que él.
El algoritmo implementa un mecanismo de reducción de la probabilidad de aceptar individuos que no sean mejor que el anterior, así como de reducir el ancho de la gaussiana del ruido con el que se muta al individuo.

O2: Algoritmo genético completo con cruce

Este optimizador utiliza un conjunto de individuos, la población sobre los que realiza tres acciones consecutivas. Primero toma parejas de individuos y los cruza generando otros nuevos, luego mutaciones sobre algunos de ellos generando más individuos nuevos, finalmente selecciona los mejores individuos.
Para cada una de las acciones existen distintas implementaciones, en nuestro caso para el cruce realizamos dos cortes en los fenotipos y mezclamos las partes correspondientes, para la mutación mutamos todos los genotipos con distinto ruido gaussiano para cada uno y para seleccionar los mejores los ordenamos y nos quedamos con los mejores según el tamaño de la población.

O3: Algoritmo genético con temple

Para implementar este algoritmo hemos mezclado ideas procedentes de los dos anteriores. Hemos partido del algoritmo genético completo con cruce y le hemos añadido que el ruido de las mutaciones se reduzca con el tiempo. De este modo pretendemos poder caer con mayor precisión sobre los mínimos que gracias al cruce esperemos que sean globales.

O4: Optimizador harmónico

Este optimizador es una actualización de las estrategias evolutivas, para implementarlo se tiene una matriz de harmonías donde cada fila es una harmonía distinta, sobre esa matriz de harmonía se realizan 3 cambios, aunque nosotros solo hemos implementado las dos primeras:

Cambio 1 - HMCR consiste en generar una harmonía, cada genotipo se genera de forma independiente. HMCR es una probabilidad de que el genotipo sea añadido a la matriz de harmonías.

Cambio 2 - PAR cada genotipo se ajusta con una probabilidad PAR, sino no se ajusta el genotipo. Para ello se define un ancho de banda bw y un número aleatorio épsilon con una distribución u(-1,1).

O5: Levenberg-Marquardt

Para entrenar a esta red neuronal vamos a utilizar el método de Levenberg-Marquardt, que consiste en optimizar la red utilizando el método de mínimos cuadrados, es una mejora del método del descenso por el gradiente. Gracias a esta mejora el método es más rápido, más eficiente y además es más difícil que pueda quedarse en mínimos locales.

3. Resultados obtenidos

Exponemos a continuación los resultados obtenidos con cada uno de los algoritmos y modelos probados. Proporcionaremos una tabla comparativa entre los algoritmos y los modelos. Recordamos que para leer la tabla podremos hacerlo por filas o por columnas. Es decir, se podrán comparar distintos modelos con el mismo optimizador o, el mismo modelo con distintos optimizadores. No tiene mucho sentido comparar un modelo con un optimizador con otro caso en el que ambos hayan cambiado.

Tabla resumen error medio

Los valores que se comparan se corresponden con el error medio absoluto.

	O1	**O2**	**O3**	**O4**	**Media**	**Media sin O1**
M1	642.4783	51.1372	61.6414	57.5888	203.2114	56.7891
M2	361.2378	59.6019	44.2313	51.7036	129.1936	51.8456
M3	265.0212	71.6373	60.0556	48.6401	111.3386	60.1110
Media	422.9124	60.7921	55.3094	52.6442		

Tabla resumen red neuronal

Se utilizará otra tabla para comparar la red neuronal con validación cruzada y sin validación cruzada:

	O5	
	Validación Cruzada	*Sin Validación Cruzada*
M4	70.9166	49.7254

Tabla resumen pesos y bias

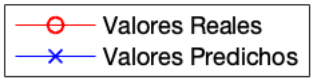
Presentamos además otra tabla en la que se comparan los pesos de cada variable a lo largo de cada uno de los modelos en el mejor individuo obtenido en cada optimizador. Cuando estos pesos sean elevados se nos estará indicando que dicha variable macroeconómica tiene gran relevancia, por el contrario, si los pesos de la variable son bajos lo que se nos estará indicando es que su importancia es reducida.

		V1		**V2**		**V3**		**V4**		**V5**		**V6**		**Bias**
O1	**M1**	-0.8729	-0.2642	0.1714	-0.8948	0.1702	-0.2751	0.7620	-0.5719	0.7971	0.4336	0.5136	-0.4949	-0.1497
	M2	0.0282	0.6825	0.1976	0.7742	0.0570	0.9987	0.3843	0.9712	0	0.4166	0.0836	0.5686	0.0024
	M3	-0.8466	-0.3297	0.1214	0.1137	-0.1402	-0.2354	0.6179	-0.3635	-0.3327	0.9913	-0.1057	-0.0002	0.3061
O2	**M1**	0.3071	1.0000	1.0000	0	0.6904	0.5095	-1.0000	-0.0105	-1.0000	-0.1126	1.0000	0	0
	M2	0	1.0000	0	1.0000	0.0993	0.4697	0.3237	0.4151	0	0.0119	0	0.1567	0
	M3	0.5195	-0.0006	-0.1530	-0.0335	0.1454	0.0096	0.4869	0.0146	-0.0615	-0.0178	-0.0872	-0.0060	0.0191
O3	**M1**	0.4108	0.4025	-0.1362	1.0000	-0.7348	-0.0824	-0.3805	-0.8567	-0.3234	-0.2264	-0.1977	0.5619	0.4577
	M2	0.0046	0.7504	0	0.0366	0.3036	0.9909	0.3995	0.5294	0	0.3150	0	0.0246	0.0081
	M3	0.1859	1.0000	0.0152	1.0000	0.6658	1.0000	1.0000	1.0000	-1.0000	-1.0000	0	-1.0000	-0.1342
O4	**M1**	0.2080	-1.0000	-0.1254	1.0000	0.4897	-1.0000	0.1020	1.0000	0.4586	-0.7096	0	0	-0.0212
	M2	0	0.9670	0	1.0000	0.1297	0.2847	0.4124	1.0000	0	0.0216	0	0.3828	0
	M3	-0.6604	-0.8968	-0.0535	0.0182	-0.0384	0.1211	0.3814	0.1052	0.2827	-0.3025	-0.0645	0.3998	0.2137

Para obtener los resultados hemos ejecutado cada algoritmo un total de 30 veces. Como error hemos proporcionamos el mínimo obtenido en todas las ejecuciones y como valores del modelo los que se corresponden con los de esa mejor optimización.

Leyenda Común

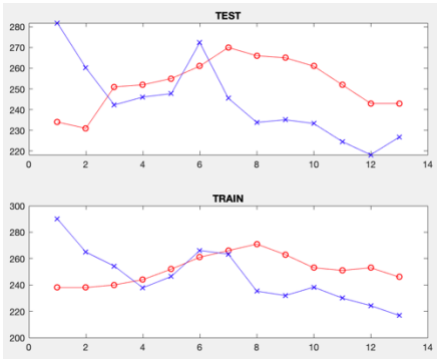
En todos los gráficos aunque no este indicado se utiliza la misma leyenda. La línea roja son los valores reales, es decir, es la variable demanda total de la energía.
La línea azul son los valores que el modelo ha predicho.



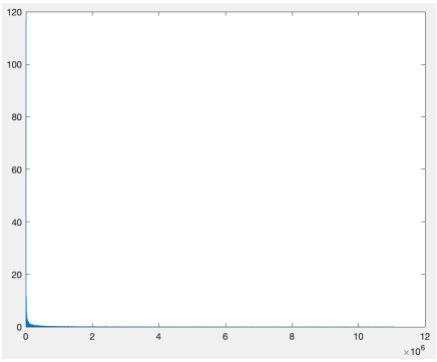
Modelo 1: Modelo exponencial (propuesto por el profesor)

Optimizador 1: Temple simulado

Predicción de energía



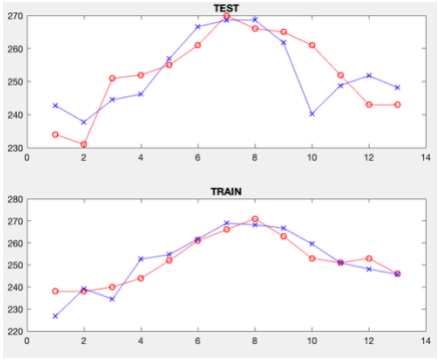
Evolución del fitness



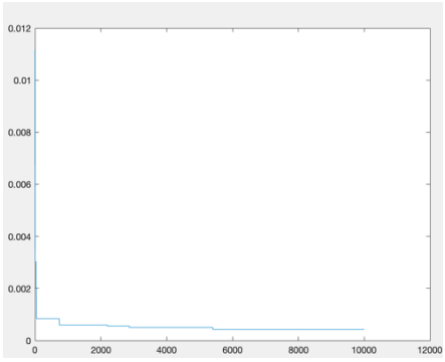
Optimizador 2: Algoritmo genético completo (cruce)

M1: Modelo exponencial (propuesto por el profesor):

Predicción de energía



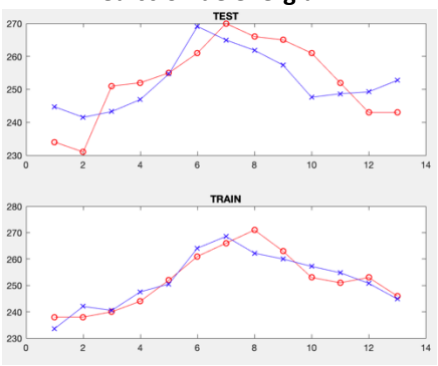
Evolución del fitness



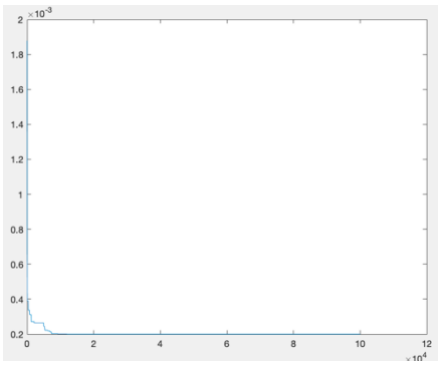
Optimizador 3: Algoritmo genético con temple

M1: Modelo exponencial (propuesto por el profesor):

Predicción de energía

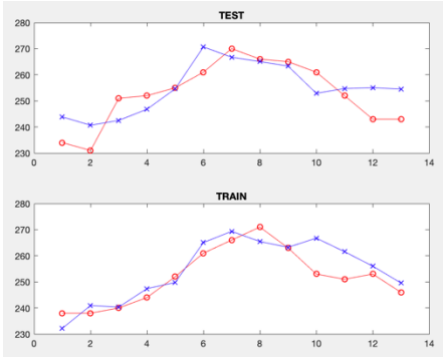


Evolución del fitness

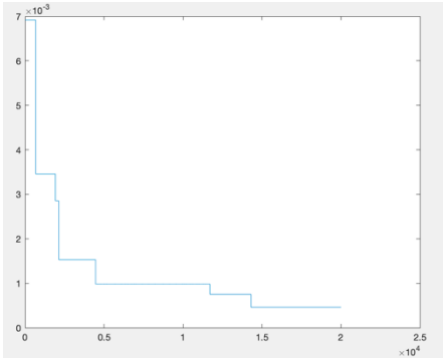


Optimizador 4: Optimizador harmónico

Predicción de energía



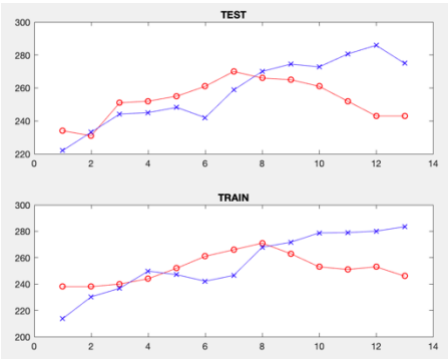
Evolución del fitness



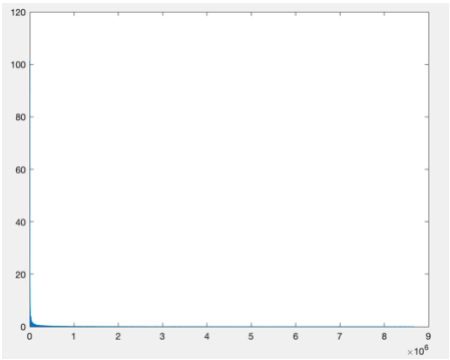
Modelo 2: Modelo exponencial (propuesto por nosotros)

Optimizador 1: Temple simulado

Predicción de energía

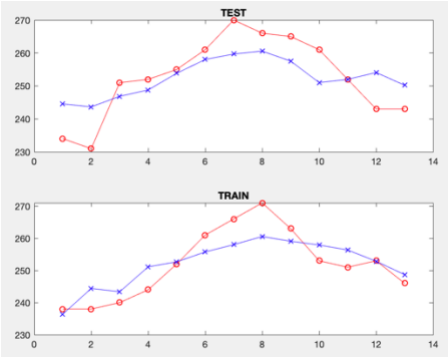


Evolución del fitness

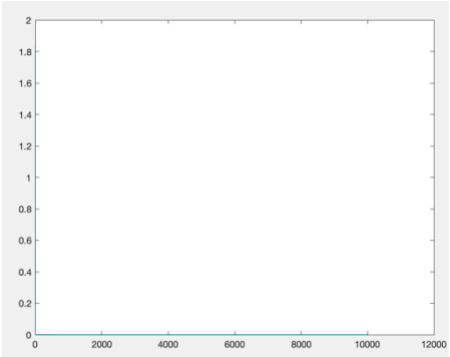


Optimizador 2: Algoritmo genético completo (cruce)

Predicción de energía

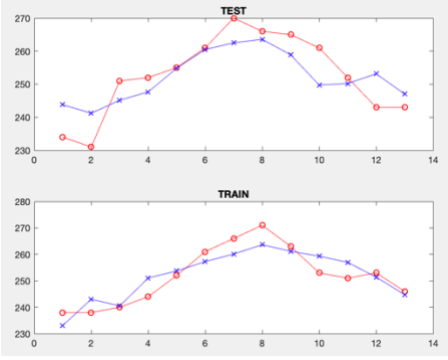


Evolución del fitness

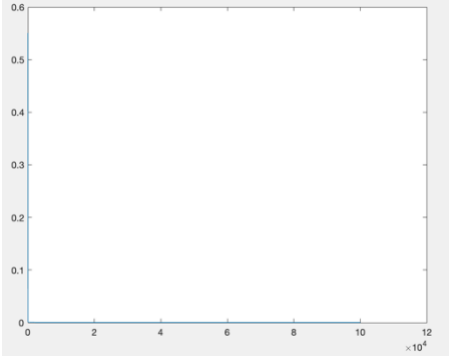


Optimizador 3: Algoritmo genético con temple

Predicción de energía

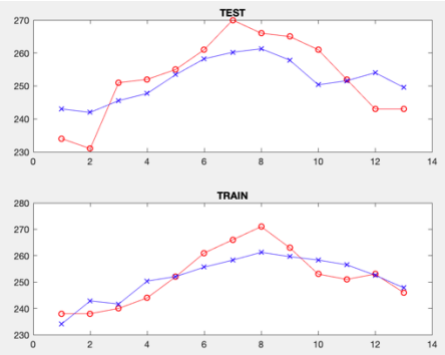


Evolución del fitness

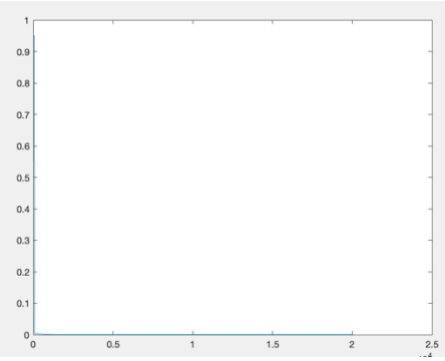


Optimizador 4: Optimizador harmónico

Predicción de energía



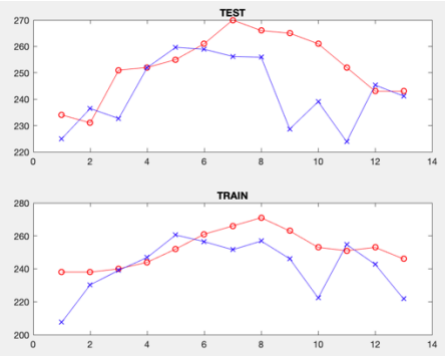
Evolución del fitness



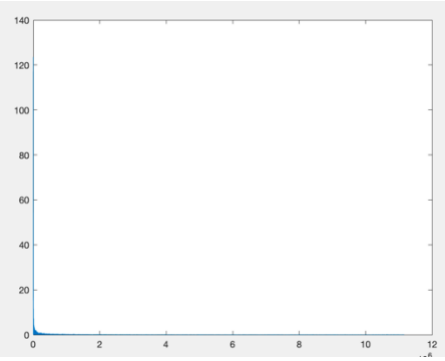
Modelo 3: Modelo lineal

Optimizador 1: Temple simulado

Predicción de energía

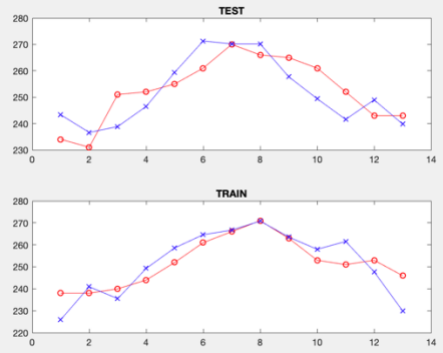


Evolución del fitness

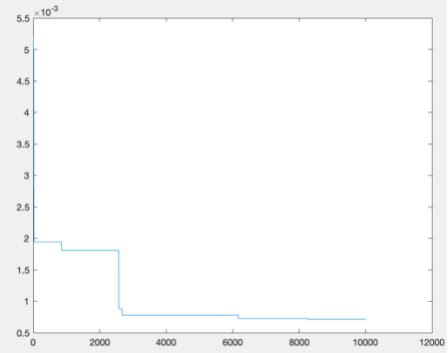


Optimizador 2: Algoritmo genético completo (cruce)

Predicción de energía

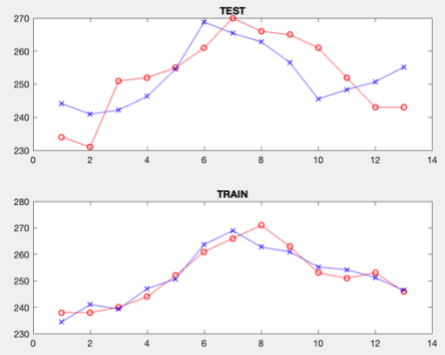


Evolución del fitness

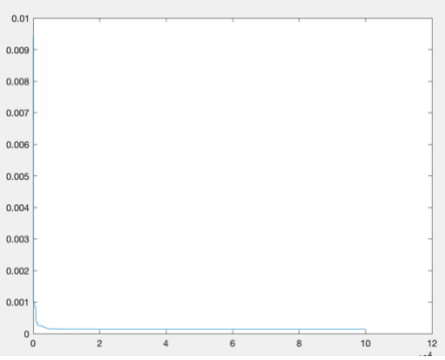


Optimizador 3: Algoritmo genético con temple

Predicción de energía

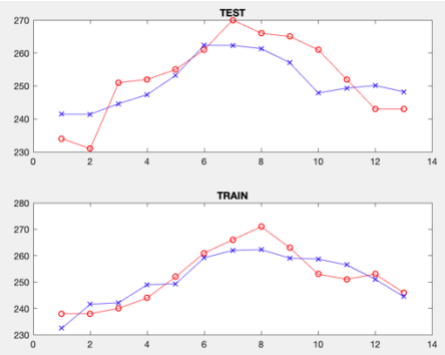


Evolución del fitness

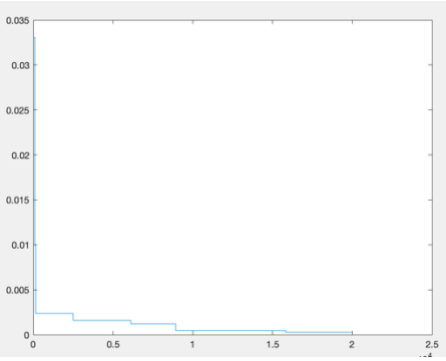


Optimizador 4: Optimizador harmónico

Predicción de energía



Evolución del fitness



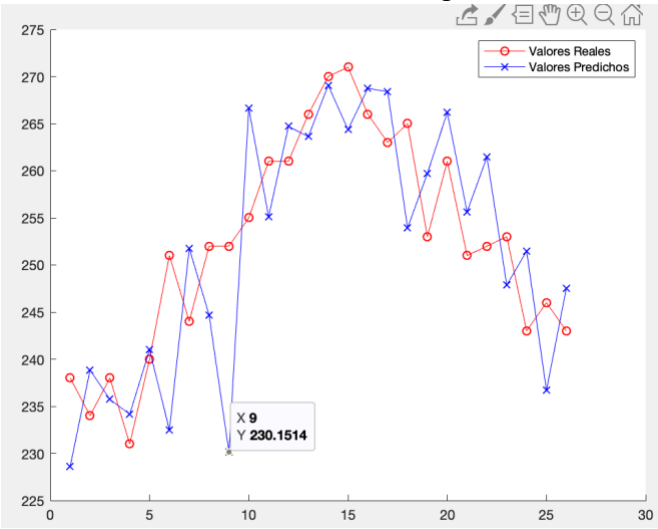
Modelo 4: Red neuronal

Optimizador 5: Levenberg-Marquardt

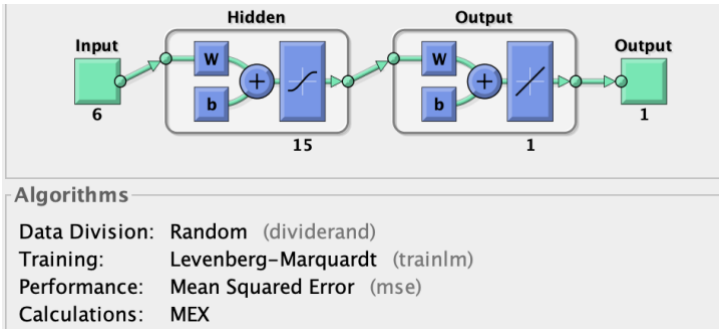
Con validación cruzada

Para este caso, como se ha comentado anteriormente es un caso especial, porque utilizamos únicamente un valor para predecir, mientras que los demás valores se utilizan para entrenar la red. Cada iteración desplazamos el valor sobre el que hacemos test, para al final sumar todos esos valores y poder tener una predicción completa de todos los años. Por lo que para hacer el gráfico se va a ver todos los valores predichos para el test.

Predicción de energía

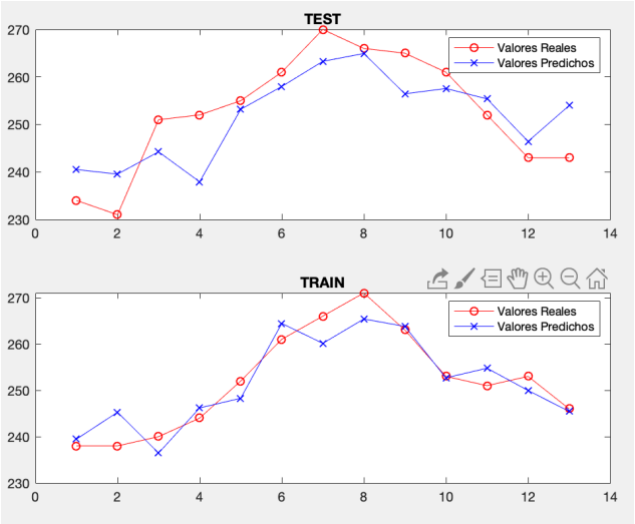


Parámetros de la red

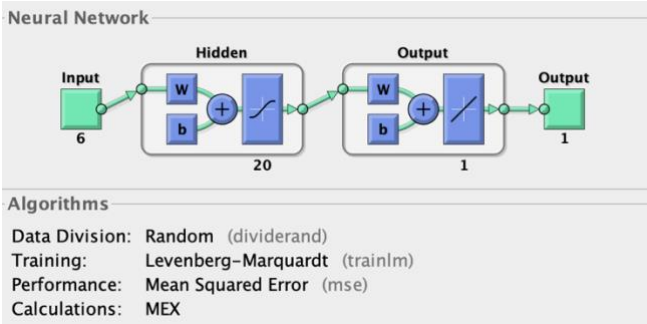


Sin validación cruzada

Predicción de energía



Parámetros de la red



4. Conclusiones

Hemos descubierto que el algoritmo de temple simulado no es un algoritmo adecuado para resolver este problema. Debido a que el resto de los algoritmos probados ofrecen muchos mejores resultados, es decir, el error que generan es mucho menor que el temple con cualquier modelo.
Si tenemos en cuenta la relación entre tiempos de optimización y los resultados obtenidos el mejor algoritmo sería la red neural sin validación cruzada. Ya que los datos son pocos el entrenamiento es instantáneo y es la que menos tarda en dar los resultados los cuales son suficientemente buenos. Los siguientes algoritmos más rápidos son el genético sin temple y después el harmónico.
Si no nos importara el tiempo y nos centráramos solo en el error mínimo el ranking obtenido para los algoritmos sería el siguiente:

- Red neural sin validación cruzada
- Optimizador harmónico
- Algoritmo genético completo con cruce
- Algoritmo genético con temple
- Red neural con validación cruzada
- Temple simulado

Las grandes sorpresas del ranking son haber obtenido peores resultados con validación cruzada que sin ella, así como que el algoritmo genético con temple haya obtenido peores resultados que el algoritmo genético sin temple.

Si ordenamos el error medio obtenido por modelos, en general los resultados son muy parecidos y no hay prácticamente un modelo muy superior que los demás. Con los resultados obtenidos el ranking es el siguiente:

1. Modelo 4 Red neuronal sin validación cruzada y después con validación cruzada.
2. Modelo 2 (exponencial) propuesto por nosotros.
3. Modelo 3 (lineal).
4. Modelo 1 (exponencial) propuesto por el profesor.

Aunque esto no es del todo correcto porque el modelo exponencial propuesto por el profesor da mejores resultados que el modelo lineal si no tenemos en cuenta el algoritmo de temple simulado que puede verse como un outlaier.

Con respecto a la calidad de las variables macroeconómicas y la importancia de cada una a la hora de predecir la demanda energética es difícil proporcionar una conclusión. Comparando los valores obtenidos para los parámetros de cada modelo hay mucha variación. Modelos iguales con optimizadores distintos llegan a soluciones muy diferentes. Incluso el mismo optimizador con el mismo modelo llega a soluciones con error similar con parámetros del modelo muy diferentes. Esto nos indica que la función que estamos tarando de optimizar es muy compleja y tiene múltiples mínimos locales de profundidad similar.
No obstante, se puede observar que solo hay dos variables cuyos pesos nunca han llegado a valer 0, dichas variables son las emisiones de dióxido de carbono y la electricidad producida. El resto de las variables en algún momento no han estado ponderando nada en el modelo pudiendo llegar a errores bajos.

5. Bibliografía

1. S. Salcedo-Sanz, J. Muñoz-Bulnes, J.A. Portilla-Figueras, J. Del Ser,
One-year-ahead energy demand estimation from macroeconomic variables using computational intelligence algorithms,
Energy Conversion and Management,Volume 99,2015,Pages 62-71,ISSN 0196-8904.
2. José Banda, Las variables macroeconómicas, July 6, 2011,
https://www.economiasimple.net/las-variables-macroeconomicas.html
Accessed: December 20, 2019
3. Sancho Salcedo-Sanz, Jesús Muñoz-Bulnes, Mark J.A. Vermeij,
New coral reefs-based approaches for the model type selection problem: a novel method to predict a nation's future energy demand,
Int. J. Bio-Inspired Computation, Vol. 10, No. 3, 2017.
4. K. Huang and Y. Hsieh, "Very fast simulated annealing for pattern detection and seismic applications," 2011 IEEE International Geoscience and Remote Sensing Symposium, Vancouver, BC, 2011, pp. 499-502.
doi: 10.1109/IGARSS.2011.6049174
5. M. M. Keikha, "Improved Simulated Annealing Using Momentum Terms," 2011 Second International Conference on Intelligent Systems, Modelling and Simulation, Kuala Lumpur, 2011, pp. 44-48.
doi: 10.1109/ISMS.2011.18
6. Jason Brownlee, A Gentle Introduction to k-fold Cross-Validation, May 23, 2018,
https://machinelearningmastery.com/k-fold-cross-validation/
Accessed: December 21, 2019
7. Alberto Quesada, Five Algorithms to Train a Neural Network
https://www.neuraldesigner.com/blog/5_algorithms_to_train_a_neural_network
Accessed: December 21, 2019
8. Sancho Salcedo Sanz, Una introducción a los algoritmos de computación evolutiva
(Diapositivas de teoría)
Accessed: December 21, 2019

6. Anexo

En este apartado está el código de los algoritmos utilizados para resolver este problema. Aunque también adjuntamos todo el código necesario para poder ejecutar todos los algoritmos implementados.

Código del algoritmo temple simulado

```
function [best_fenotype, fitness] = Temple (evaluator,comparator, stopper, newFenotype, sigma, lambda,
domain)
    best_fenotype = newFenotype();
    best_fitness = evaluator(best_fenotype);
    fitness = best_fitness;
    sigma_inicial = sigma;
    current_iteration = 1;
    while ~stopper()
        for i = 1:lambda
            new_fenotype = totalMutation(sigma, domain, best_fenotype);
            new_fitness = evaluator(new_fenotype);
            if comparator(new_fitness, best_fitness)
                best_fenotype = new_fenotype;
                best_fitness = new_fitness;
                fitness = [fitness best_fitness];
            else
                if rand(1,1) < exp(-(abs(best_fitness - new_fitness))/sigma)
                    best_fenotype = new_fenotype;
                    best_fitness = new_fitness;
                    fitness = [fitness best_fitness];
                end
            end
        end
        %sigma = sigma*0.95;
        sigma = sigma_inicial/(0.5*current_iteration);
        current_iteration = current_iteration + 1;
    end
end
```

Código del algoritmo genético completo (cruce)

```
function [best_fenotype, best_fitness] = FullGenetic (evaluator, comparator, stopper, mutator, crosser,
selector, newPopulation)
    population = newPopulation();
    getBest = @(population) bestInPopulation (evaluator, comparator, population);
    [best_fenotype, best_fitness] = getBest(population);
    while ~stopper()
        childs = zeros(size(population,1)*2,size(population,2));
        for i = 1:(size(population,1))
            if mod(i,2)==0
                index1 = randi(size(population,1), 1);
                index2 = randi(size(population,1), 1);
                [child1, child2] = crosser(population(index1,:), population(index2,:));
                childs(i,:) = child1;
                childs(i+1,:) = child2;
            end
        end
        mutations = [];
        for i = 1:size(population,1)
            if randi(100, 1) <= 5
                mutations = [mutations; mutator(population(i,:))];
            end
        end
        population = selector ([population; childs; mutations]);
        [best_fenotype, new_fitness] = getBest(population);
        best_fitness = [best_fitness new_fitness];
    end
end
```

Código del algoritmo genético con temple

```
function [best_fenotype, best_fitness] = GeneticTemple (evaluator, comparator, stopper, mutator, crosser,
selector, newPopulation, sigma, lambda)
    population = newPopulation();
    getBest = @(population) bestInPopulation (evaluator, comparator, population);
    [best_fenotype, best_fitness] = getBest(population);
    sigma_inicial = sigma;
    current_iteration = 1;
    while ~stopper()
        for l = 1:lambda
            childs = zeros(size(population,1)*2,size(population,2));
            for i = 1:(size(population,1))
                if mod(i,2)==0
                    index1 = randi(size(population,1), 1);
                    index2 = randi(size(population,1), 1);
                    [child1, child2] = crosser(population(index1,:), population(index2,:));
                    childs(i,:) = child1;
                    childs(i+1,:) = child2;
                end
            end
            mutations = [];
            for i = 1:size(population,1)
                mutations = [mutations; mutator(population(i,:), sigma)];
            end
            population = selector ([population; childs; mutations]);
            [best_fenotype, new_fitness] = getBest(population);
            best_fitness = [best_fitness new_fitness];
        end
        %sigma = sigma*0.95;
        sigma = sigma_inicial/(0.5*current_iteration);
        current_iteration = current_iteration + 1;
```

```
    end
end
```

Código del algoritmo harmonic search

```
function [best_fenotype, best_fitness] = Harmonic (evaluator, comparator, stopper, newGenotype,
newPopulation, mutatorFenotype, mutatorelement)
    population = newPopulation();
    population = sortPopulation(evaluator, population);
    worst_fitness = evaluator(population(end,:));
    getBest = @(population) bestInPopulation (evaluator, comparator, population);
    [best_fenotype, best_fitness] = getBest(population);
    while ~stopper()
        new_fenotype = zeros(1, size(population,2));
        for i = 1:size(population, 2)
            if randi(100, 1) > 20
                row_index = randi(size(population, 1),1);
                new_fenotype(i) = population(row_index, i);
            else
                new_fenotype(i) = newGenotype();
            end
            if randi(100,1) < 30
                new_fenotype(i) = mutatorelement(new_fenotype(i));
            end
        end
        if randi(100,1) < 30
            new_fenotype = mutatorFenotype(new_fenotype);
        end
        new_fitness = evaluator(new_fenotype);
        if comparator(new_fitness, worst_fitness)
            population = sortPopulation(evaluator, population);
```

```
            population(end,:) = new_fenotype;
            worst_fitness = evaluator(population(end-1,:));
            population = population(randperm(size(population,1)),:);
        end
        [best_fenotype, best_fitness_pop] = getBest(population);
        best_fitness = [best_fitness best_fitness_pop];
    end
end
```

Código de la red neuronal con validación cruzada

```
clear all;
close all;
data = ...
    [
        [58512000 58559000 58851000 59206000 59327000 59541000 59753000 59964000 60186000
60496000 60912000  61357000 61805000 62244000 62704000 63179000 63621000 64016000
64374000 64707000 65027000 65342000 65659000 65998000 66331000 66624000 66892000];
        [1.275 1.276 1.409 1.333 1.402 1.610 1.614 1.461 1.511 1.500 1.368 1.382 1.500 1.848 2.124 2.204
2.325 2.663 2.923 2.694 2.647 2.863 2.681 2.809 2.849 2.434 2.465];
        [347 374 363 346 339 349 364 353 380 373 374 383 373 379 379 377 365 358 356 338 348 320 322
326 292 299 306];
        [421 456 464 473 477 494 513 505 511 526 540 550 559 567 574 576 575 569 574 536 569 561 566
572 563 568 553];
        [323 347 356 356 363 368 384 382 393 401 410 421 419 437 449 451 446 448 461 448 472 443 454
457 432 440 448];
        [7.8 8.0 9.0 10.0 10.4 10.1 10.5 10.7 10.3 10.0 8.5 7.8 7.9 8.5 8.9 8.9 8.8 8.0 7.5 9.1 9.3 9.2 9.8 10.3
10.2 9.9 9.7];
        [1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006
2007 2008 2009 2010 2011 2012 2013 2014 2015 2016];
```

```
    ];
targets = [225 238 234 238 231 240 251 244 252 252 255 261 261 266 270 271 266 263 265 253 261 251
252 253 243 246 243];
variables_to_use = [1 1 1 1 1 1 0];
number_of_variables = sum(variables_to_use);

normatization_data = max(data,[],2);
normatization_targets = max(targets);

trainD = zeros(number_of_variables, floor(size(data,2)/2));
targets_train = zeros(1, floor(size(data,2)/2));
testD = zeros(number_of_variables, floor(size(data,2)/2));
targets_test = zeros(1, floor(size(data,2)/2));
offset = 0;

for i = 1:size(data, 1)
    if variables_to_use(i)
        for j = 1:(size(data, 2)-1)
            if mod(j ,2)
                trainD(i-offset,(j+1)/2) = data(i,j)/normatization_data(i);
            else
                testD(i-offset,j/2) = data(i,j)/normatization_data(i);
            end
        end
    else
        offset = offset + 1;
    end
end

for j = 2:size(targets, 2)
    if mod(j ,2)
        targets_test((j-1)/2) = targets(j)/normatization_targets;
    else
        targets_train(j/2) = targets(j)/normatization_targets;
    end
end

dataD = zeros((size(data,1)-1),(size(data,2)-1));
targetsD = zeros(1,(size(targets,2)-1));
for i = 2:(size(dataD,2)+1)
```

```
    if ~mod(i,2)
        dataD(:,i-1) = trainD(:,i/2);
        targetsD(i-1) = targets_train(i/2);
    else
        dataD(:,i-1) = testD(:,(i-1)/2);
        targetsD(i-1) = targets_test((i-1)/2);
    end
end


errors = [];
targetsDraw = [];
outputsDraw = [];
for i = 1:size(dataD,2)
   if i < size(dataD,2)
    trainCross = [dataD(:,1:i-1) dataD(:,i+1:end)];
    targetsTrain = [targetsD(:,1:i-1) targetsD(:,i+1:end)];
   else
    trainCross = [dataD(:,1:i-1)];
    targetsTrain = [targetsD(:,1:i-1)];
   end
   %disp(size(trainCross))
   %disp(size(targetsTrain))

   testCross = dataD(:,i);
   targetsTest = targetsD(:,i);
   targetsDraw = [targetsDraw targetsTest];



    hiddenLayerSize = [15];
    net = fitnet(hiddenLayerSize);
    net.divideParam.trainRatio = 100/100;
    net.divideParam.valRatio = 0/100;
    net.divideParam.testRatio = 0/100;
    net.trainParam.epochs = 10;
    net.trainParam.goal = 0;
    % Entrenamiento de la red
    [net,tr] = train(net,trainCross,targetsTrain);
    outputs = net(testCross);
    outputsDraw = [outputsDraw outputs];
```

```
    errors = [errors ((outputs-targetsTest)^2)*normatization_targets^2];
end

view(net)
disp(errors);
figure;
hold on;
view(net)
plot(targetsDraw*normatization_targets, '-or')
plot(outputsDraw*normatization_targets, '-xb')
legend('Valores Reales','Valores Predichos')
hold off;
disp(mean(errors));
```

Código de la red neuronal sin validación cruzada

```
clear all;
close all;
data = ...
   [
      [58512000 58559000 58851000 59206000 59327000 59541000 59753000 59964000 60186000
60496000 60912000  61357000 61805000 62244000 62704000 63179000 63621000 64016000
64374000 64707000 65027000 65342000 65659000 65998000 66331000 66624000 66892000];
      [1.275 1.276 1.409 1.333 1.402 1.610 1.614 1.461 1.511 1.500 1.368 1.382 1.500 1.848 2.124 2.204
2.325 2.663 2.923 2.694 2.647 2.863 2.681 2.809 2.849 2.434 2.465];
      [347 374 363 346 339 349 364 353 380 373 374 383 373 379 379 377 365 358 356 338 348 320 322
326 292 299 306];
      [421 456 464 473 477 494 513 505 511 526 540 550 559 567 574 576 575 569 574 536 569 561 566
572 563 568 553];
      [323 347 356 356 363 368 384 382 393 401 410 421 419 437 449 451 446 448 461 448 472 443 454
457 432 440 448];
      [7.8 8.0 9.0 10.0 10.4 10.1 10.5 10.7 10.3 10.0 8.5 7.8 7.9 8.5 8.9 8.9 8.8 8.0 7.5 9.1 9.3 9.2 9.8 10.3
10.2 9.9 9.7];
      [1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006
2007 2008 2009 2010 2011 2012 2013 2014 2015 2016];

   ];
targets = [225 238 234 238 231 240 251 244 252 252 255 261 261 266 270 271 266 263 265 253 261 251
252 253 243 246 243];
variables_to_use = [1 1 1 1 1 1 0];
number_of_variables = sum(variables_to_use);

normatization_data = max(data,[],2);
normatization_targets = max(targets);

trainD = zeros(number_of_variables, floor(size(data,2)/2));
targets_train = zeros(1, floor(size(data,2)/2));
testD = zeros(number_of_variables, floor(size(data,2)/2));
targets_test = zeros(1, floor(size(data,2)/2));
offset = 0;

for i = 1:size(data, 1)
   if variables_to_use(i)
```

```
        for j = 1:(size(data, 2)-1)
            if mod(j ,2)
                trainD(i-offset,(j+1)/2) = data(i,j)/normatization_data(i);
            else
                testD(i-offset,j/2) = data(i,j)/normatization_data(i);
            end
        end
    else
        offset = offset + 1;
    end
end

for j = 2:size(targets, 2)
    if mod(j ,2)
        targets_test((j-1)/2) = targets(j)/normatization_targets;
    else
        targets_train(j/2) = targets(j)/normatization_targets;
    end
end

%dataT = (trainD+testD);
%targetsT = (targets_train+targets_test);

hiddenLayerSize = [20];
net = fitnet(hiddenLayerSize);
% División del conjunto de datos para entrenamiento, validación y test
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 0/100;

% Entrenamiento de la red
[net,tr] = train(net,trainD,targets_train);
outputs = net(testD);

error = 0;
numValoresTest = length(outputs);
for i = 1:numValoresTest
    error = error + ((targets_test(i)*normatization_targets - outputs(i)*normatization_targets)^2);
end
outputs_train = net(trainD);
```

```
%%
% Visualización
view(net)
disp(error/numValoresTest)
figure;subplot(2,1,1);
plot(targets_test*normatization_targets, '-or')
hold on;
plot(outputs*normatization_targets, '-xb')
legend('Valores Reales','Valores Predichos')
title("TEST");
hold off
subplot(2,1,2);
plot(targets_train*normatization_targets, '-or')
hold on;
plot(outputs_train*normatization_targets, '-xb')
legend('Valores Reales','Valores Predichos')
title("TRAIN");
```