Neurocomputación

Otras aplicaciones de las
Redes Neuronales Artificiales

Francisco Andreu Sanz
Francisco Lobo García
Aitor Muñoz Cuña

Grupo 2461

Índice de contenido

1. Auto-Encoder

1.1 Modificaciones realizadas

Los cambios a realizar en nuestro código han sido los siguientes:

- Ha sido necesario definir una **función de parseo del alfabeto** para que el formato con el que se trabaje sea el mismo que con la práctica anterior. Además, ha sido necesario transformar el conjunto de datos a bipolar (1 si el píxel está encendido y -1 en caso contrario).
- La deducción de la clase a clasificar ha cambiado debido a que ya no activamos únicamente la neurona de salida con mayor valor:

```
for l in range(0, self.num_neuronas_salida):
    if (self.vector_yout[l] == maximo):
        vector_prediccion = np.append(vector_prediccion, 1)
    else:
        vector_prediccion = np.append(vector_prediccion, 0)
```

```
for l in range(0, self.num_neuronas_salida):
    if (self.vector_yout[l] >= 0):
        vector_prediccion = np.append(vector_prediccion, 1)
    else:
        vector_prediccion = np.append(vector_prediccion, -1)
```

- La función de error se ha modificado de tal forma que ahora se computa en explotación el error promedio en términos de número de píxeles errados por letra.

1.2 Estudio realizado

1.2.1 Training y Test sin ruido

El entrenamiento ha sido realizado durante **500 épocas con tasa de aprendizaje 0.2.** El número de neuronas en la capa oculta mínimas que han sido necesarias para clasificar todo el conjunto correctamente ha sido de 9.
A continuación mostramos una tabla con los resultados obtenidos:

Nº de neuronas en capa oculta	Nº de letras falladas	Letras falladas	Nº de píxeles erróneos (por letra) promedio	%Acierto en *Training*
1	26	'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'	8.231 píxeles	0.0%
2	26	'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'	6.461 píxeles	0.0%
3	25	'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'Y', 'Z'	4.421 píxeles	3,426%
4	23	'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'Y', 'Z'	2.807 píxeles	6,077%
5	16	'A', 'C', 'D', 'E', 'F', 'G', 'J', 'L', 'M', 'P', 'Q', 'S', 'U', 'W', 'Y', 'Z'	1.307 píxeles	27,897%
6	14	'A', 'B', 'C', 'E', 'F', 'G', 'K', 'L', 'M', 'O', 'P', 'R', 'T', 'Z'	1.038 píxeles	38.430%

7	7	'A', 'B', 'D', 'E', 'F', 'I', 'J', 'M', 'O', 'S'	0.884 píxeles	53.138%
8	1	'M'	0.038 píxeles	93.256%
9	0	-	0 píxeles	95.254%
10	0	-	0 píxeles	98.507%

Pese a que con más de 8 neuronas en la capa oculta la clasificación se realiza correctamente, vemos que la letra que más problemas ocasiona es la **'M'.**

1.2.2 Training sin ruido y Test con ruido

En este apartado hemos realizado modificaciones a nivel de bit mediante un método implementado por nosotros que cambia *n* valores de cada letra al azar para la partición de **Test**.

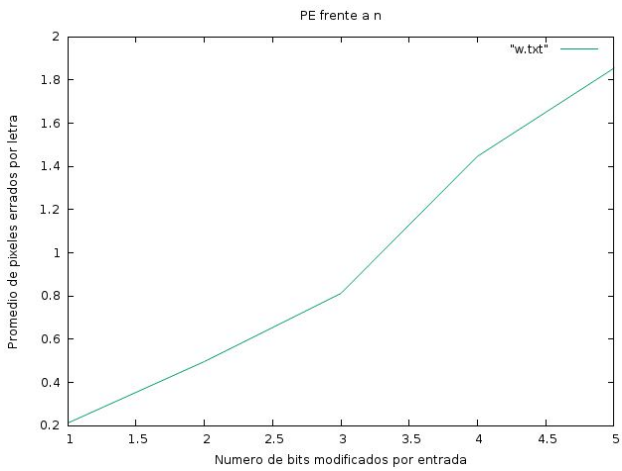
Para este apartado hemos diseñado otra versión del método *parsearAlfabeto(n)* que, además de parsear el fichero que se nos proporciona, cambia el valor de *n* bits de cada letra pero eso sí, dejando su salida intacta **puesto que es para la partición de test.**
Además, crea 10 versiones de ruido para cada letra. El formato de *alfabeto_ruido.txt* (representativo, realmente se muestran bits, no letras) es:

```
(A con ruido)₁ - A
(A con ruido)₂ - A
...
(A con ruido)₁₀ - A

(B con ruido)₁ - B
....
```

Los resultados, con **500 épocas y tasa de aprendizaje 0.2** son:

n	PE	Letras erradas	Letras acertadas
1	0.211538461538	36	224
2	0.496153846154	70	190
3	0.811538461538	108	152
4	1.44615384615	150	110
5	1.85384615385	173	87



Los resultados obtenidos variando la n son acordes a lo esperado. A mayor *n,* mayor tasa de error.

1.2.3 Training y Test con el mismo ruido

Para ello creamos un alfabeto de *training* con *n = 5* ruido en las letras. Posteriormente, en la partición de test insertamos **el mismo ruido.**
Realizamos varias pruebas y en todas ellas la red AutoEncoder se comportó de la misma manera que en el apartado sin ruido. Ésto es lógico puesto que al final realmente lo que estás haciendo es que la red aprenda patrones con ruido. Si en la partición de test clasificas con **exactamente** el mismo ruido, se va a comportar de manera óptima.

1.2.4 Training y Test con ruido diferente

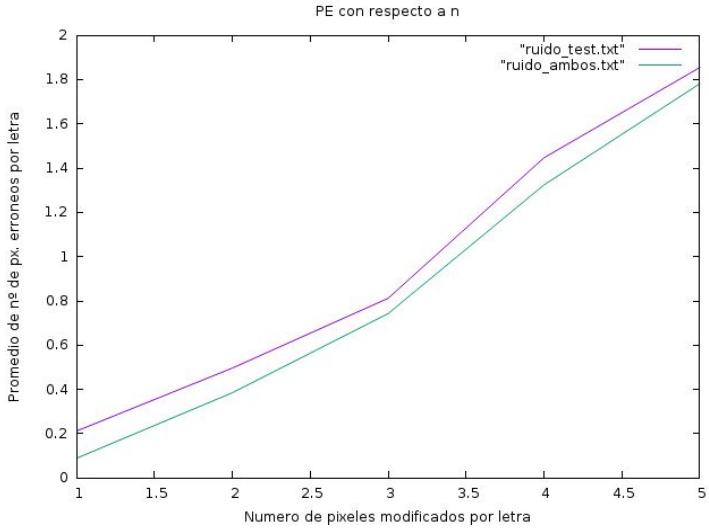
Por último hemos modificado el alfabeto de *training* añadiendo, además de las versiones "óptimas" de la letra, **10** versiones **con ruido** pero que predicen la letra original. Es decir, el formato representativo de *alfabeto_training_ruido.txt* es:

```
(A sin ruido) - A
(A con ruido)_1 - A
(A con ruido)_2 - A
...
(A con ruido)_10 - A

(B sin ruido) - B
(B con ruido)_1 - B
....
```

Mostramos aquí la tabla con resultados:

n	PE	Letras erradas	Letras acertadas	**%Acierto en Training**
1	0.088461538461	13	247	99.6195804196 %
2	0.384615384615	48	212	99.2146853147%
3	1.04230769231	96	164	98.1307692308 %
4	1.32307692308	110	150	97.7328671329 %
5	1.78076923077	140	120	90.8237762238 %



Podemos comprobar que para todas las *n,* **los resultados han sido notablemente mejores** pero, eso sí, a costa de un tiempo de ejecución 10 veces mayor (había que entrenar nºLetras x nº Variantes·letra -> 26*10).

2. Series Temporales

2.1 Función de paseo de fichero de entrada

La función **adaptaficheroserie**(entrada, salida, Na, Ns) se encarga del *parseo* del fichero de serie temporal proporcionado para que pueda ser tratado por nuestro motor de clasificación.

Recibe **4** argumentos de entrada:

- **entrada**: El fichero a parsear.
- **salida**: El archivo donde se escribirá la entrada parseada.
- **Na**: El número de atributos de la red.
- **Ns**: El número de salidas (o número de puntos a predecir) de la red.

2.2 Cambios en la red

En este apartado seguimos al pie de la letra los pasos indicados por el enunciado para hacer funcionar nuestra red neuronal como motor de predicción **(*regresión*).** El número de cambios fueron mayores que para el *Auto-Encoder.*
Los cambios fueron los siguientes:

- El número de neuronas en capa de salida a activar **ya no es uno** (como sí pasaba con la clase PerceptronMulticapa).
- La **función de activación** en la capa de salida ya no es la sigmoidal, sino la función $f(x)=x$. Obviamente, la derivada de la misma será 1.
- Se ha añadido un parámetro a la función creaParticiones que, al establecerlo a *False*, deshabilita la randomización de los datos para crear las particiones(así lo requiere el problema).

2.3 Cómputo del ECM

En este caso, tanto la función **error()** como la función **errorEntrena()** ya no devuelve un porcentaje o tasa de acierto, sino que **calcula el ECM (**o error cuadrático medio**)** de la siguiente manera:

$$ECM = \frac{\sum_{t=1}^{n}\left(x_t - \hat{x}_t\right)^2}{n}$$

Siendo x_t el dato real, $\hat{x}_t$ la predicción y n el número de datos a evaluar.

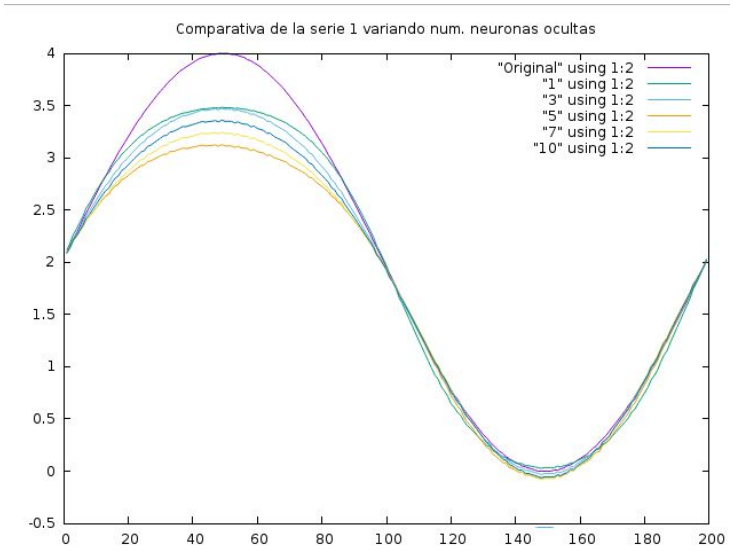
2.4 Serie temporal 1

Con todo lo anterior, procedemos al análisis de la primera serie temporal proporcionada, con **200 datos para entrenar y 200 para clasificar**. A la hora de la puesta en marcha de nuestro clasificador (o predictor, mejor dicho) el único dato que mantuvimos intacto fue el **nº de épocas,** a **250** (número que consideramos suficiente para estudiar esta serie temporal sencilla como veremos a continuación). Tanto el nº de neuronas en capa oculta y la tasa de aprendizaje, como Na y Ns, fueron variando para comprobar los resultados.

Para empezar, mostraremos **el ECM y una representación gráfica** de la serie variando *num_neuronas_oculta,* y dejando fijo el resto de parámetros de la siguiente manera:

- Na = 2
- Ns = 1
- Tasa de aprendizaje = 0.15

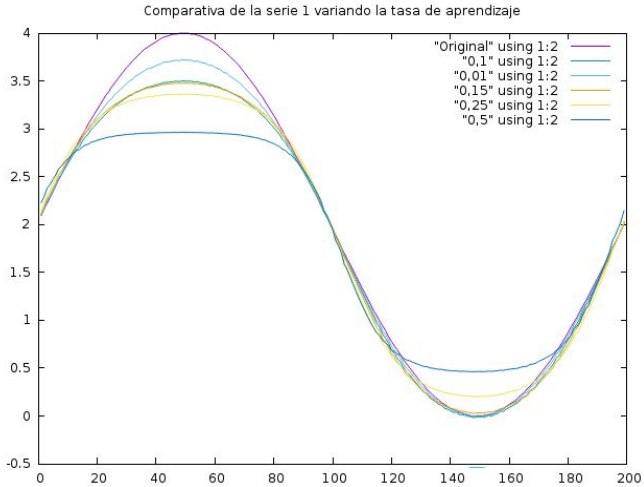
Núm. neuronas oculta	ECM en *train*	ECM en *test*
1	0.00135105305455	0.0478190676236
3	0.000771736598384	0.0565807085729
5	0.000649980184627	0.133219028626
7	0.000642604860043	0.126868653215
10	0.000579009402471	0.0892499730298
40	0.000902235125439	0.0032308098116



Como podemos comprobar,el mejor resultado es **num_neuronas_oculta = 40.** Sin embargo, tras este y por extraño que parezca, el segundo resultado es con **num_neuronas_oculta = 1.** Tiene sentido puesto que **Ns = 1** y al haber una única neurona

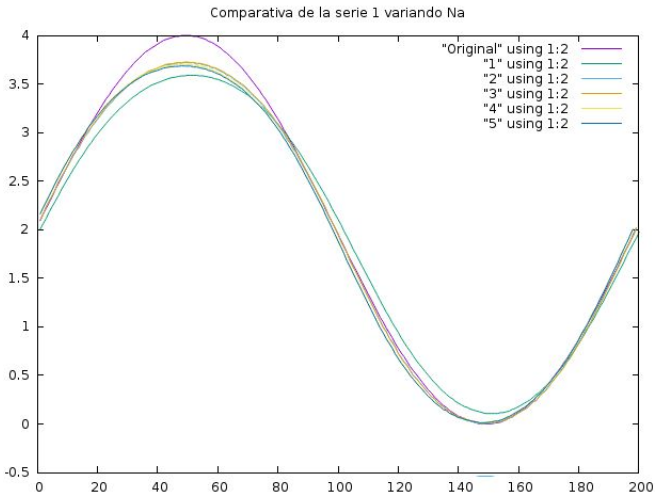
de salida **no es necesaria tanta información en la capa oculta.** Por tanto, en realidad el parámetro Ns y el número de neuronas ocultas van bastante de la mano.
En segundo lugar, vamos a hacer lo mismo pero esta vez **variando la tasa de aprendizaje**. Dejaremos por tanto los valores anteriores fijos y el número de neuronas ocultas a 1.

Tasa de aprendizaje	ECM en *train*	ECM en *test*
0.01	0.00653880583103	0.0137077899663
0.1	0.000778553704535	0.0451936931022
0.15	0.00144233226811	0.0473954017523
0.25	0.00194498823081	0.073165516267
0.5	0.00206465276776	0.2310681890078



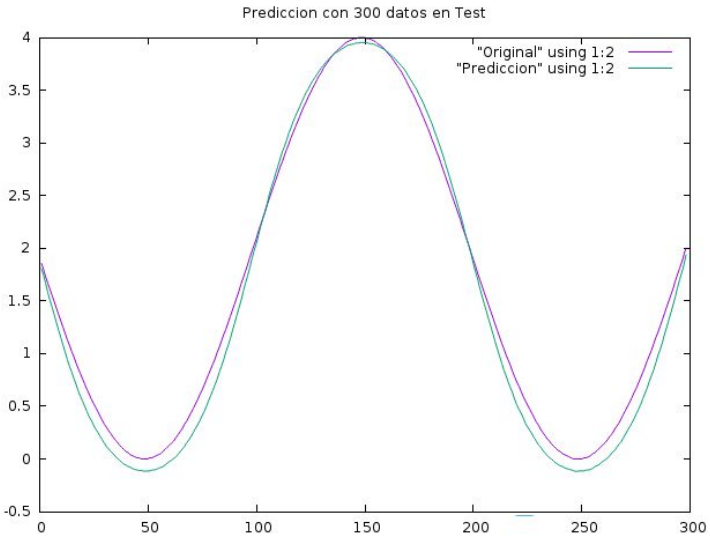
Observamos que la predicción mejor parada es aquella con tasa de aprendizaje **0.01.** De hecho conforme vamos incrementando la tasa la predicción va fallando más.
Veamos otro estudio, esta vez variando **Na.**

Na	ECM en *train*	ECM en *test*
1	0.00618593468145	0.0354000754632
2	0.00605359356428	0.0134117100681
3	0.00511318792506	0.0126425188197
4	0.00354671236486	0.0150907725074
5	0.00412775743871	0.016843618603



En este caso no se aprecia tan notablemente pero si nos fijamos en el ECM de test vemos que para **Na = 3** es donde aparece un mejor resultado (por muy poco).
Como conclusión, y viendo los resultados obtenidos en el entrenamiento, podemos concluir que la serie temporal 1 es **muy sencilla** debido a varios factores, entre ellos que contiene pocos datos y sobre todo **que es periódica.** De hecho, se repite cada 200 uds. de tiempo. Casualmente, al haber 400 datos en total y al dividir el %Train y %Test en **200 y 200,** la modelización es bastante similar al primer apartado del Auto-Encoder, donde **entrenábamos y clasificábamos lo mismo.** Como hemos podido comprobar, sin un número de neuronas y de épocas demasiado alto, la predicción es satisfactoria. **Cabe destacar que en las gráficas únicamente hemos representado el conjunto de *Test,* puesto que el enunciado no decía lo contrario y lo vimos así oportuno para el estudio.**

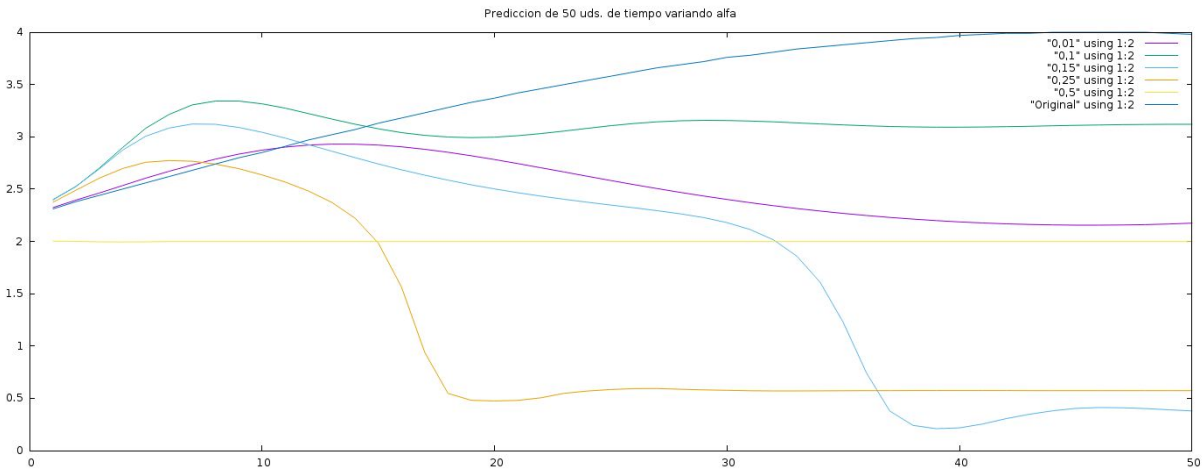
Por último, vamos a dejar fijos todos los valores pero vamos a variar el %Train y %Test en **100** datos y **300** respectivamente. Observemos la predicción:



Realmente esperábamos un peor resultado de nuestra red a la hora de predecir los valores. Sin embargo también debemos tener en cuenta de que se trata de una serie "simétrica" verticalmente (**función seno**). Aun así podemos apreciar que **en las zonas "bajas", las cuales no se han visto en el entrenamiento, obtiene peores resultados que en la "alta"**, como es lógico.

En este apartado se nos pedía también emplear recursión para predecir 50 unidades de tiempo utilizando como entradas las "salidas" del dato anterior. Hemos ido variando el parámetro de la tasa de aprendizaje dejando fijos el resto de valores a **1** neurona oculta, partición de entrenamiento **100%, Na = 5, Ns = 1 y Nf = 50.** Esta predicción recursiva se realiza gracias a la función `clasifica_recursivo(...)`. Dicha función es muy similar a la función `clasifica(...)`, pero estableciendo un caso base(prediciendo el primer dato), y a partir de éste, generando el resto de datos a partir del anterior, teniendo como parámetros:

- **Na**: Número de atributos a utilizar para predecir el siguiente dato.
- **Ns**: Número de datos a predecir.



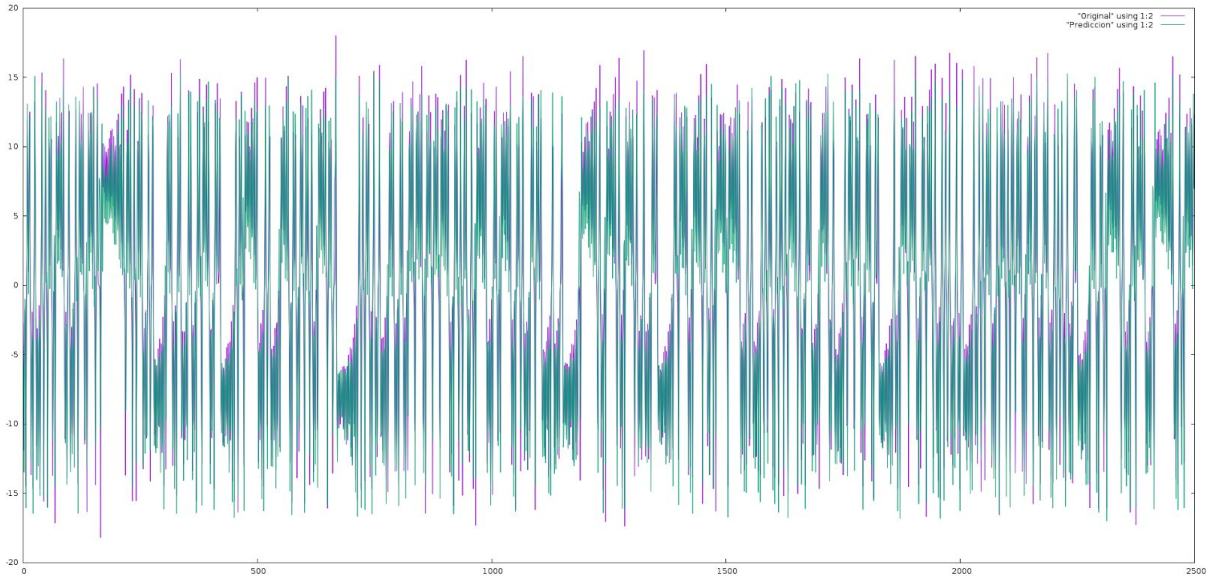
Podemos observar que pese a que la primera subida la cumplen todos(antes o después), a partir de ahí, dependiendo del parámetro de la tasa de aprendizaje, la serie comete una bajada o mínima o **muy grande,** como el caso de **alfa = 0.25**.
La mejor aproximación la hemos conseguido con **alfa = 0.01**.Aun así, la predicción es relativamente mala debido a que **vamos acumulando el error en cada dato.**

2.5 Serie Temporal 2

Para este apartado nos hemos limitado a probar nuestro motor de predicción utilizando los parámetros que mejor nos han funcionado en el apartado anterior. Esto es:

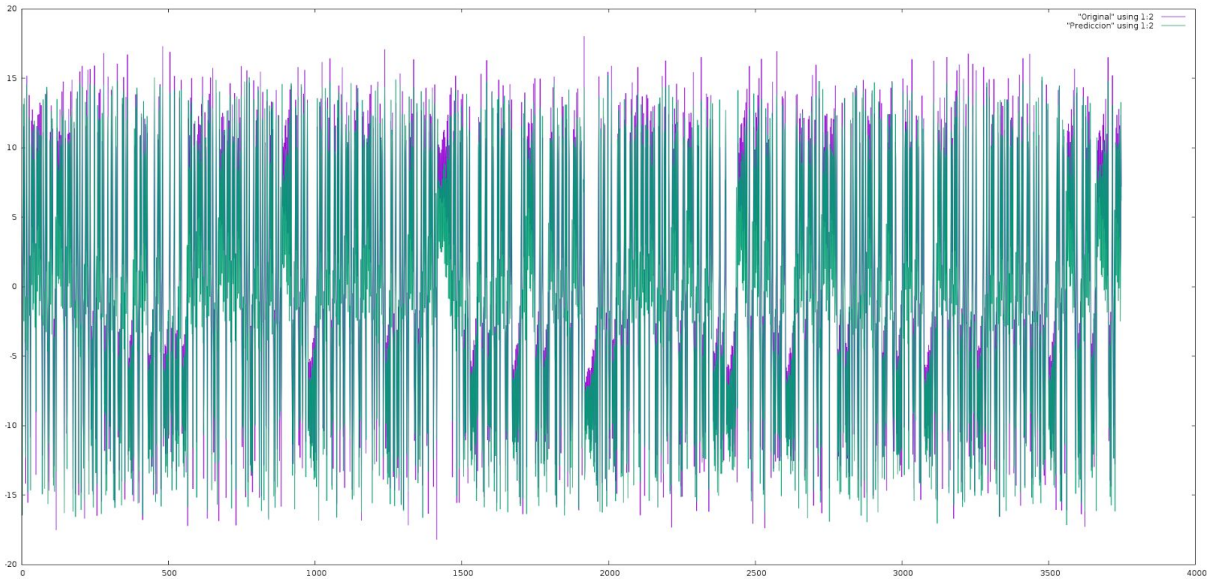
- Tasa de aprendizaje: 0.01
- Número de neuronas: 10 (éste lo hemos cambiado, pues pensamos que al ser la serie más compleja, requeriríamos un número de neuronas ocultas mayor).
- Na: 3
- Ns: 1

El particionado *Train-Test* ha sido de 50%-50%:



ECM en *Train*	0.597819659074	ECM en *Test*	4.99932391187

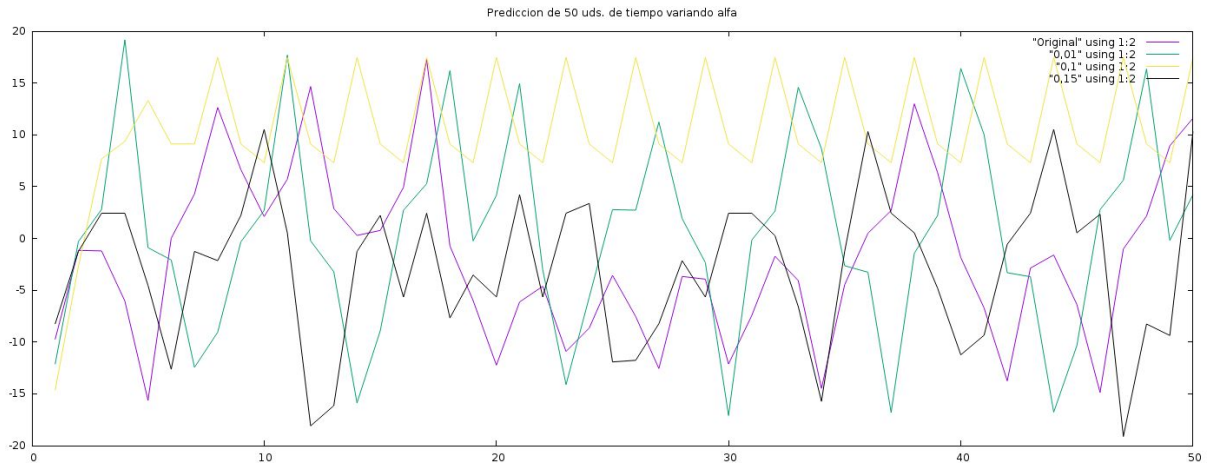
Como podemos observar, la predicción **se aproxima en gran medida a la real.** Ésto podía haberse mejorado aún más incrementando el número de neuronas ocultas. Sin embargo, el tiempo de ejecución se veía muy comprometido y decidimos dejarlo en 10. Probemos ahora con un particionado *Train-Test* 25%-75%:



ECM en *Train*	0.521849031304	ECM en *Test*	5.17906987705

Gráficamente es difícil concluir cómo se comporta ahora la predicción respecto al caso anterior. Es cierto que vemos quizá más “rango por dato” erróneo, sin embargo es más fácil apreciar el comportamiento al ver el ECM en *Test*, el cual se ve bastante afectado **doblando** su valor con respecto al caso 50%-50%.

Al igual que en la primera serie temporal, vamos a representar la predicción con los mismos parámetros que los utilizados en la primera serie temporal, **entrenando sólamente los 50 primeros datos del entrenamiento** y prediciendo 50 datos (es decir Nf = 50):



Como vemos en el resultado obtenido, la predicción *a ciegas* no es nada buena. Sin embargo, realmente ésto es más realista puesto que normalmente en una serie temporal no tenemos los resultados del “dato anterior” para así no arrastrar error.
En este caso, como dijimos anteriormente, hemos representado únicamente un segmento de la serie. Concretamente, del dato 0 al dato 50. Creemos que el mejor resultado se obtiene con la **tasa de aprendizaje = 0.15.**