Sistemas de Inteligencia Artificial - 2019 1ºC

TPE Nº2: Redes Neuronales

Integrantes

Nombre	*Correo*	*Legajo*
Dammiano, Agustín	adammiano@itba.edu.ar	57702
Donoso Naumczuk, Alan	adonoso@itba.edu.ar	57583
Sanz Gorostiaga, Lucas	lsanz@itba.edu.ar	56312
Torreguitar, José	jtorreguitar@itba.edu.ar	57519

1. Introducción

Este informe detalla la implementación de una red neuronal, la cual fue entrenada para poder aproximar un terreno. Se exploran distintas variaciones sobre el algoritmo backpropagation y mediante pruebas empíricas se desea sacar conclusiones sobre las mismas. Además se relata sobre las distintas decisiones que fueron tomadas para mejorar el rendimiento de la red neuronal.

2. Implementación de la red neuronal

2.1. Algoritmo

Se implementó el algoritmo de backpropagation visto en clase. A este mismo se realizaron dos variaciones configurables, el coeficiente de aprendizaje adaptativo y momentum (como fueron vistas en clase). El trabajo cuenta con otra opción configurable, la cual decide cuando se corrigen los pesos de la red neuronal. Una de estas es la opción incremental, donde por cada época se agarran los patrones de aprendizaje al azar y se corrigen los pesos por cada patrón. En cambio la otra opción, llamada batch, acumula todas las correcciones de los patrones de una época y las mismas son aplicadas al final de la época. Además, se brinda la opción de utilizar una función de activación distinta en la última capa con respecto a las otras capas. La razones por dicha decisión se darán más adelante en este informe.

2.2. Inicialización de los pesos

La inicialización de los pesos puede perjudicar o beneficiar el aprendizaje de la red neuronal. Si dichos pesos son cercanos a los valores de una red que aprendió el problema entonces se necesitaría pocas épocas para llegar a dicha configuración y sucede lo opuesto si los valores son lejanos. Además, existe la posibilidad de que si estos valores son muy grandes la red neuronal sature fácilmente y quede estancada en un mínimo local.

La primer solución que se planteó era inicializar todos los pesos con una constante que arbitrariamente se eligió como 1 (uno). Pero rápidamente se notó que al tener un gran número de entradas las mismas podrían saturar la red neuronal. Debido a esto, se propuso otra solución a dicho problema que era elegir a dicha constante usando el valor de las entradas de las capas, se usó a uno sobre la raíz cuadrada de las entradas como la constante. Por último, se planteó inicializar los pesos aleatorios (en el orden de uno sobre la raíz cuadrada de las entradas) con el fin de tener la posibilidad de obtener una mejor red al realizar varios aprendizajes con la misma arquitectura. Esto se debe, a que algunas queden en mínimos locales debido a los pesos iniciales.

2.3. Normalización de los datos de entrada

Al utilizar como función de activación a la tangente hiperbólica y la exponencial, las cuales saturan cuando un número es muy grande o muy chico, es necesaria trasladar las entradas de la red a un rango en el cual no saturen. Por lo tanto se utilizó la siguiente función para normalizar los datos:

$$f(x) = a + (x - A) * \frac{b - a}{B - A}$$

Siendo el rango [a,b] donde la función de activación no satura y el rango [A,B] donde existen los datos de entrada. Una limitación de esta función es que los valores fueran del rango [A,B] podrían saturar la red y esta no podría aproximar su salida.

Se propuso una función alternativa que no tenía dicho problema ya que normaliza el vector de entradas. Pero dicha función genera que dos patrones de entrada distintos entren a la red siendo iguales (por

ejemplo (0,1) y (0,2)). Esto genera que la red no pueda distinguir entre los patrones y no pueda aprender correctamente. Es por esto que se optó por la primer función de normalización.

2.4. Desnormalización de los datos de salida

El problema que se intenta resolver en este trabajo consta de aproximar la altura de un terreno. Como los valores de altura existen en los reales y las funciones de activación tangente hiperbólica y exponencial devuelven valores en un rango entre dos constantes, es necesario desnormalizar la salida. A pesar de esto se optó por no utilizar una función normalizadora y en cambio usar a la función lineal como función de activación en la última capa. De esta forma se puede evitar todos los problemas relacionados a desnormalizar datos, los cuales son similares a lo que surgieron en la sección anterior.

2.5. Elección de los conjuntos de aprendizaje

Para elegir el conjunto de aprendizaje se decidió tomar una muestra que tuviera datos representativos para la estimación del terreno. Es por esto que se eligió tomar, en partes iguales, un conjunto de puntos altos, bajos e intermedios. La configuración del programa permite definir el porcentaje total de la muestra de aprendizaje con respecto a los datos totales, dividiendo esta misma en tres partes iguales.

3. Procedimiento

3.1. Arquitectura

Para el diseño de la arquitectura se utilizó el siguiente procedimiento: Se comenzó trabajando en incremental, con una red de 4 capas ocultas de 30 neuronas cada una de ellas. Luego, comenzamos a decrementar la cantidad de capas y de neuronas por capa, de tal forma que se mantenga el mismo error, o se obtenga uno menor. Seguimos con este procedimiento hasta que el error aumentara, en ese caso sabiamos que habiamos tomado una decisión de cambio de arquitectura posiblemente incorrecta.

De esta manera, pasamos de tener 4 capas ocultas a solo tener 2. Y de tener 30 neuronas por capa oculta, a tener 20 en la primera y 10 en la segunda.

Dado que no logramos reducir el error de entre 0.4 y 0.2 aproximadamente, se optó por probar utilizar batch en vez de incrementar. Con este cambio, logramos reducir nuestro error a 0.08, lo cual consideramos significativo. Y luego, variando entre la activacion y desactivacion de mejoras como momentum y factor de aprendizaje adaptativo, o variando sus parámetros, logramos bajar el error a 0.06.

La configuración final entonces resulta ser 2 capas ocultas, la primera con 20 neuronas y la segunda con 10. La capa de entrada con 2 neuronas, y la de salida con 1. Aprendizaje en modo batch, con learning factor adaptativo activo y momentum desactivado. Para más detalles en los parámetros, ver la configuración de la Corrida 1 en el anexo. Ver la Figura 1 en el anexo, para ver una representación del grafo de la red neuronal recientemente detallada.

Nota: siempre que se habla de error, se refiere al error cuadrático medio.

3.2. Comparaciones de distintos parámetros

Batch vs Incremental

Se realizó dos corridas, una en batch (ver anexo corrida 1) y otra en incremental (ver anexo corrida 2) dejando el resto de la configuración igual excepto por el número de épocas. En vista de que el error cuadrático medio era aproximadamente 3, se decidió modificar el eta y el valor k del eta adaptativo. Si no se

realizaban esto cambios, al correr incremental con lo valores de batch, la red no lograba aprender. Una vez hechas las modificaciones, el error cuadrático medio se redujo a 0,54764.

Adaptive LR vs Constant LR

Se realizó dos corridas con el resto de parámetros constantes, solo modificando si estaba o no activo el factor de aprendizaje adaptativo: activado (ver anexo corrida 1) y desactivado (ver anexo corrida 3).

Momentum vs No-Momentum

Se realizó dos corridas con el resto de parámetros constantes, solo modificando el flag de activación del momentum (ver anexo corridas 3 y 4).

Adaptive LR & Momentum vs Constant LR & No-Momentum

Se realizó dos corridas con el resto de parámetros constantes. En una se activaba el factor de aprendizaje adaptativo y momentum (ver anexo corrida 5). En cambio en la otras se desactivaba esto parámetros (ver anexo corrida 3).

Porcentaje del conjunto de pruebas

Se realizó varias corridas cambiando el porcentaje del conjunto de aprendizaje. De esta forma podemos concluir sobre el aprendizaje nivel de generalización de la red. Los porcentajes usados fueron 50% (ver anexo corrida 7), 70% (ver anexo corrida 1) y 90% (ver anexo corrida 6).

4. Análisis de los resultados

Batch vs Incremental

En los gráficos de error de las corridas 1 y 2 se puede notar que uno es más suave que otro. El más suave (ignorando cuando deja de converger) corresponde al 3 el cual utiliza batch. Esto se debe a que los pesos son corregidos de golpe con todos los cambios aculados en una época. En cambio en incremental el siguiente paso es dado solo por un solo patrón específico que no se sabe cómo puede repercutir en el promedio y por eso vemos esos saltos más bruscos. En general, para superficie que nos fue asignada obtuvimos un mejor error cuando utilizamos batch, las corridas 1 y 3 son solo un caso particular de esto.

En la corrida 2 se obtuvo un error de 0,54764 que es mucho mayor al error de 0.067393 obtenido en la corrida 1.

Adaptive LR vs Constant LR

En los gráficos de error de las corridas 1 y 3 se puede notar que uno diverge más que otro. Esto se debe al tener el coeficiente de aprendizaje adaptativo este aumenta si el error decrece lo cual en batch sucede comúnmente y por lo tanto termina teniendo un eta grande que hace que se aleje del mínimo.

Los resultados fueron de 0.067393 para el eta adaptativo y 0.068420 para el eta constante.

Momentum vs No-Momentum

En los gráfico de las corridas 3 y 4 se puede destacar que el gráfico 4 (el que usa momentum) disminuye su error más rápido que el otro. Inclusive hay momentos donde se pasa y deja de converger (los picos en la función de error). Esto se debe al funcionamiento de momentum: a la corrección actual se le suma la corrección anterior multiplicada por una constante. De esta forma, las correcciones anteriores de alguna manera le dan mayor “fuerza” a las correcciones del momento, es como que vinieran con cierta inercia a tender hacia algún lado.

Adaptive LR & Momentum vs Constant LR & No-Momentum

Como se vio en los dos casos anteriores, el momentum genera que cuando el error esta decreciendo, lo haga con más velocidad. El eta adaptativo en batch hace que el error diverja varias veces. Probablemente por estos frecuentes picos de la función de error, los resultados fueron peores: 0.076247 para el que tiene momentum y eta adaptativo y 0.068420 para el que no.

Porcentaje del conjunto de pruebas

A medida que se disminuye el porcentaje de patrones para el grupo de aprendizaje el error del conjunto de testeo sube. Pero a pesar de esto, el error de testeo, cuando el porcentaje es 50%, es de 0.13105, lo cual es un error relativamente bajo para los que obtuvimos en la resolución de este problema.

5. Conclusión

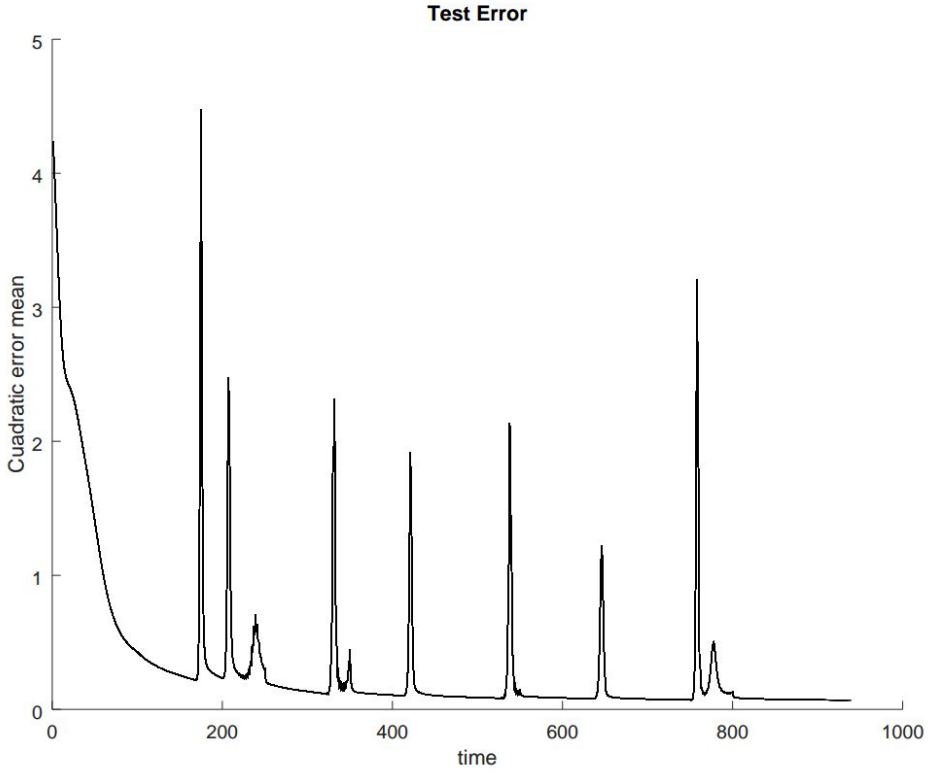
Luego de haber estudiado exhaustivamente y de manera empírica los resultados de los distintos diseños de la red neuronal para estimar una superficie se puede llegar a concluir que no hay una única regla o fórmula que cumpla con los valores del problema a resolver. Los problemas que constituyen estimaciones por redes neuronales supervisadas van a tener una resolución que dependerá completamente de la información provista para entrenar a la red así como de su arquitectura, con esta última siendo dependiente de la primera. El impacto que ocasiona un pequeño ajuste o desajuste de parámetros o variables puede ser lo que lleve a que la red pase largos minutos o hasta horas intentando resolver el problema, así como también puede llegar a simplificar enormemente el panorama.

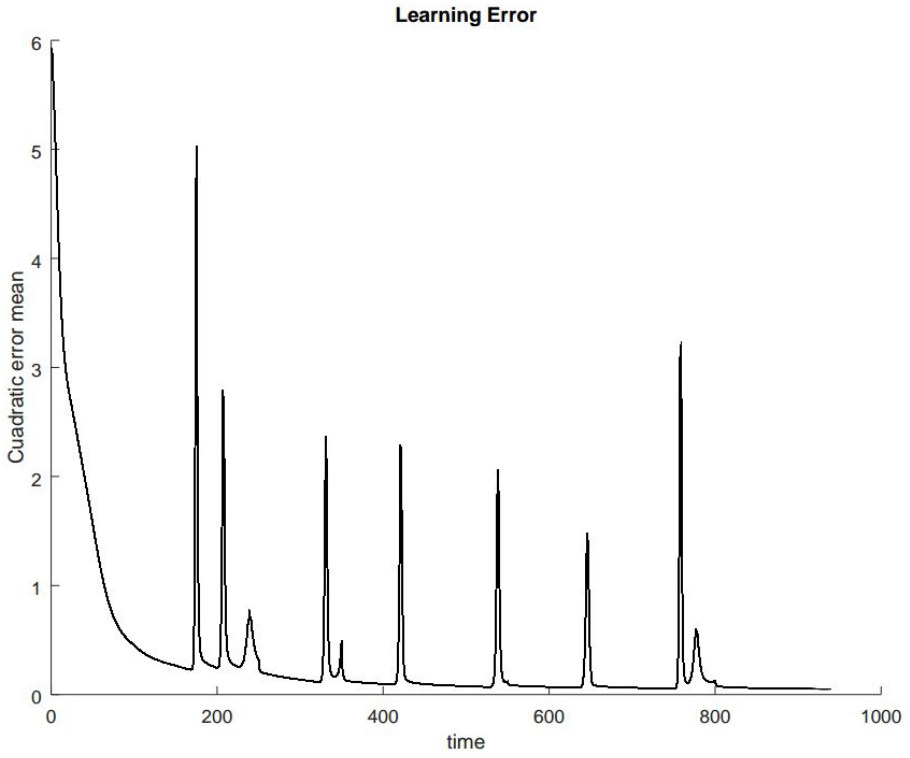
Las mejoras de momentum y eta adaptativo pueden reducir el tiempo de entrenamiento si son usadas correctamente, de lo contrario, pueden perjudicar los resultados. Es de mayor utilidad utilizar eta adaptativo combinado con aprendizaje incremental.

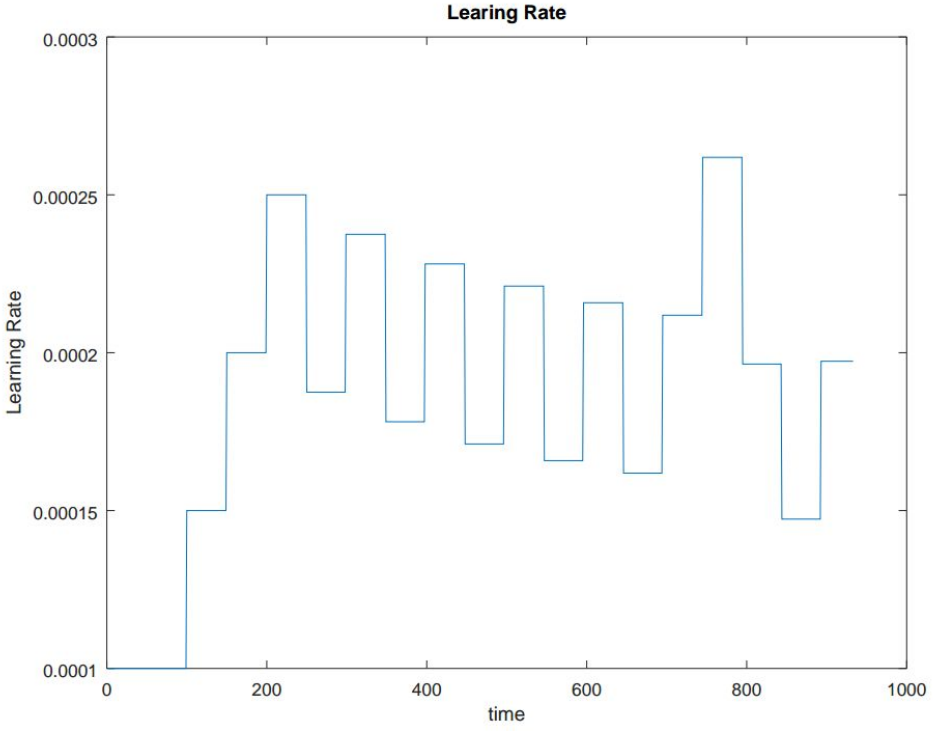
6. Anexo

Corrida 1:

learningRate	0.0001
adaptiveLearningRate	1
timesLR	50
incLR	0.00005
decLR	0.25
momentum	0
momentumRate	0.9
maxError	0.05
epoch	1000
debugTimes	Inf
deltaWCalculation	batch
beta	1
gamma	0
function	tanh
betaLast	1
gammaLast	0
functionLast	linear
learningSamplePercentage	0.7
testingSample	terrain02.data
qtyNeuronsInLayer1	2
qtyNeuronsInLayer2	20
qtyNeuronsInLayer3	10
qtyNeuronsInLayer4	1

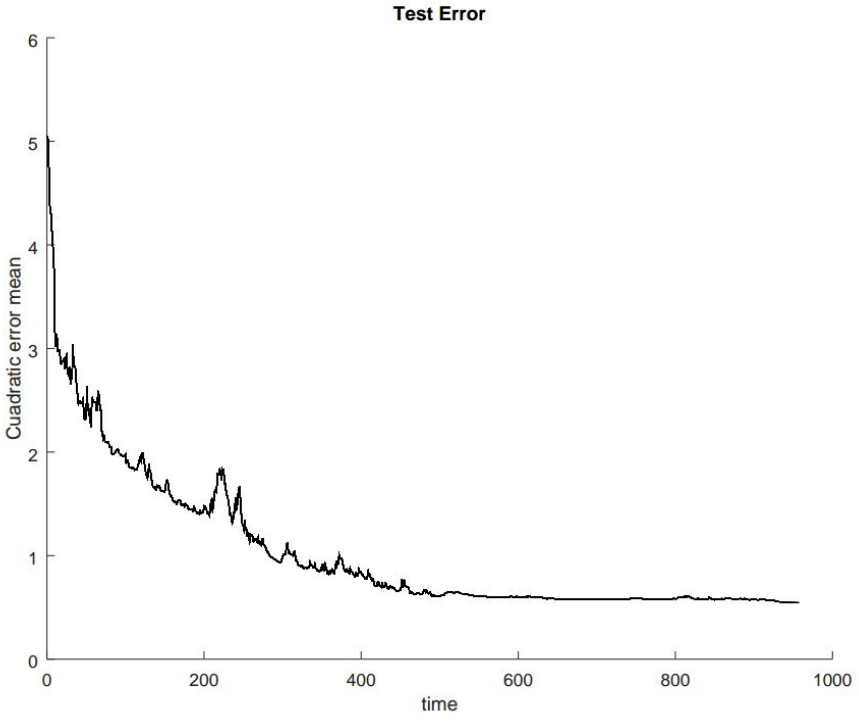


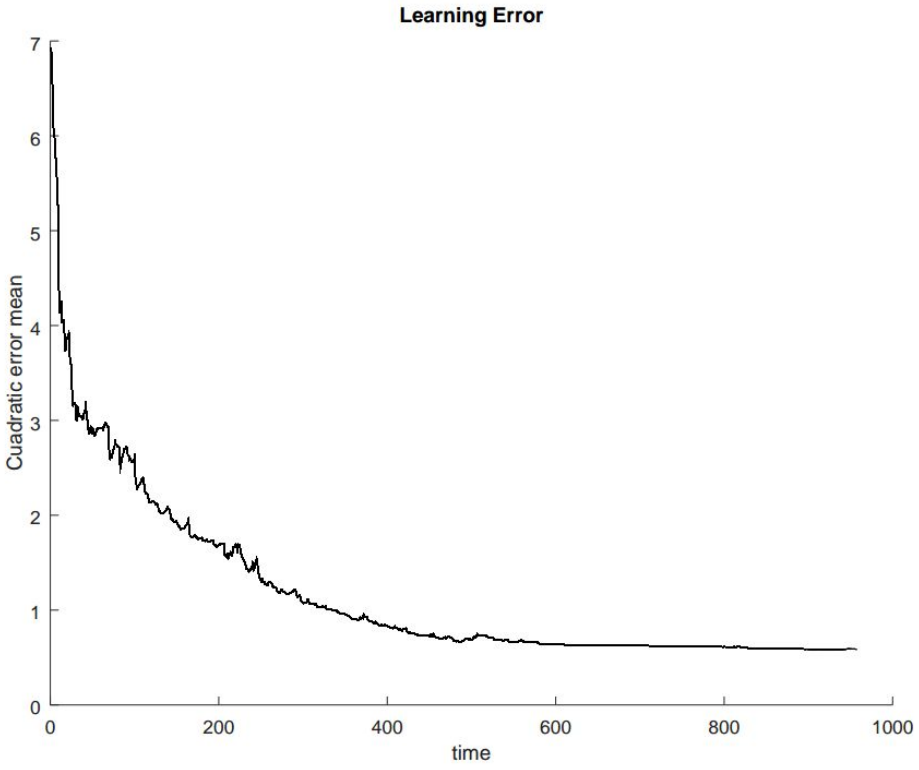

Learning Error
Cuadratic error mean
time
0
1
2
3
4
5
6
0
200
400
600
800
1000

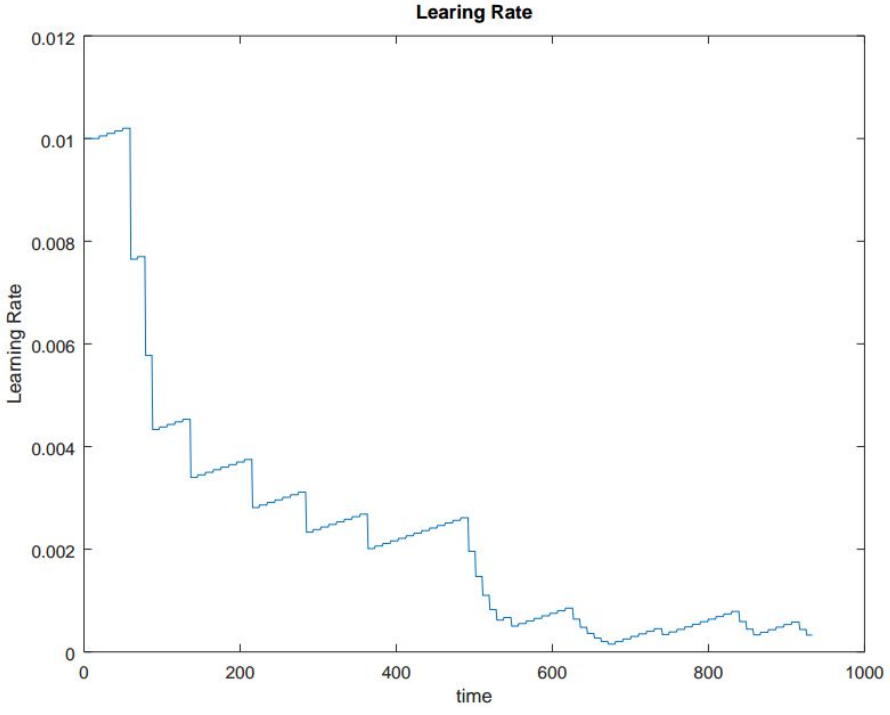

Learing Rate
Learning Rate
time
0.0001
0.00015
0.0002
0.00025
0.0003
0
200
400
600
800
1000

Corrida 2 (Incremental):

learningRate	0.01
adaptiveLearningRate	1
timesLR	10
incLR	0.00005
decLR	0.25
momentum	0
momentumRate	0.9
maxError	0.05
epoch	3
debugTimes	Inf
deltaWCalculation	incremental
beta	1
gamma	0
function	tanh
betaLast	1
gammaLast	0
functionLast	linear
learningSamplePercentage	0.7
testingSample	terrain02.data
qtyNeuronsInLayer1	2
qtyNeuronsInLayer2	20
qtyNeuronsInLayer3	10
qtyNeuronsInLayer4	1

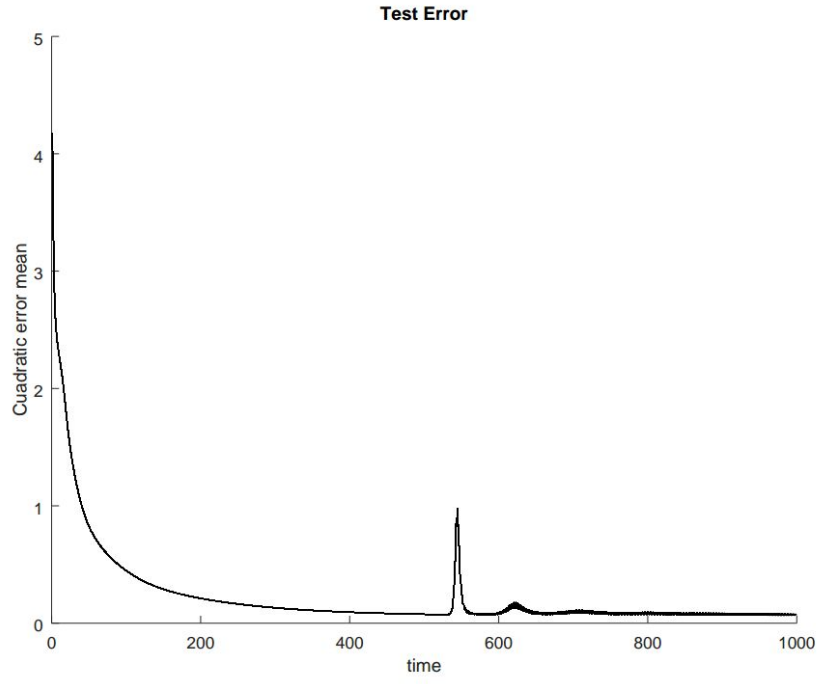


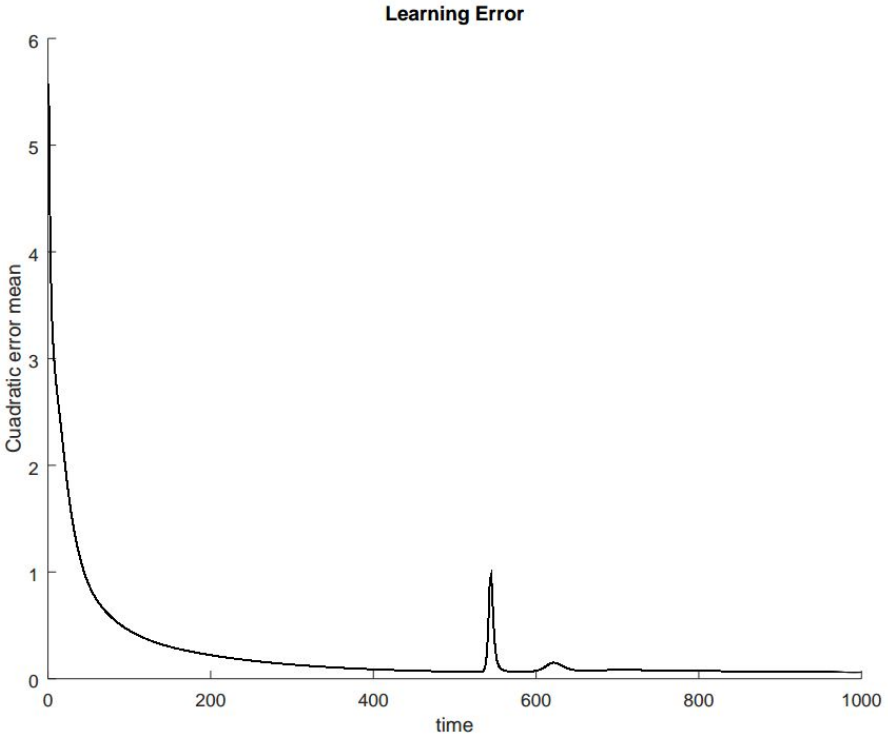

Learning Error
Cuadratic error mean
time
0
1
2
3
4
5
6
7
0
200
400
600
800
1000

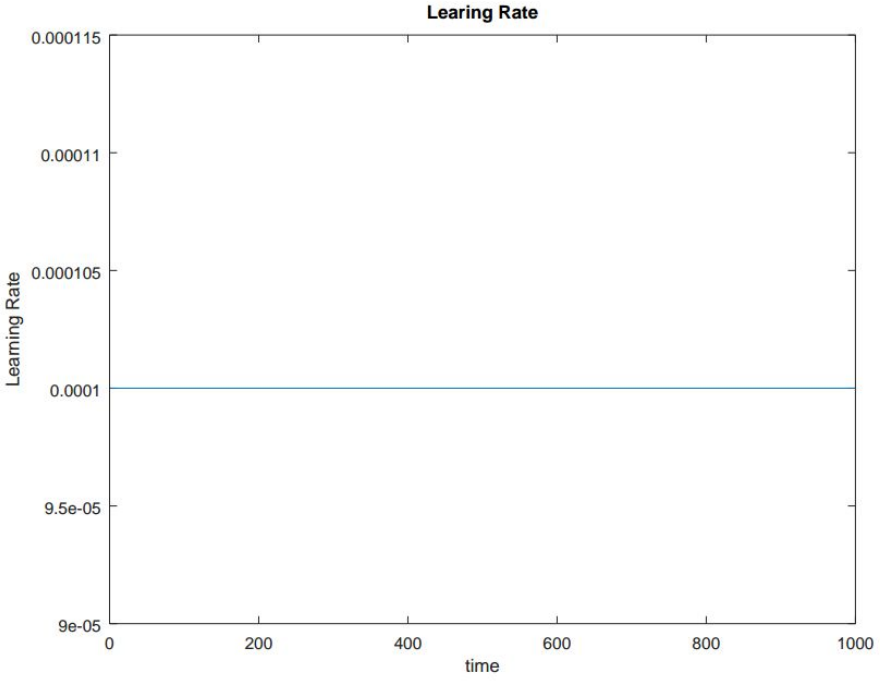

Learing Rate
Learning Rate
time
0
0.002
0.004
0.006
0.008
0.01
0.012
0
200
400
600
800
1000

Corrida 3 (Constant Learning Rate & No momentum):

learningRate	0.0001
adaptiveLearningRate	0
timesLR	50
incLR	0.00005
decLR	0.25
momentum	0
momentumRate	0.9
maxError	0.05
epoch	1000
debugTimes	Inf
deltaWCalculation	batch
beta	1
gamma	0
function	tanh
betaLast	1
gammaLast	0
functionLast	linear
learningSamplePercentage	0.7
testingSample	terrain02.data
qtyNeuronsInLayer1	2
qtyNeuronsInLayer2	20
qtyNeuronsInLayer3	10
qtyNeuronsInLayer4	1

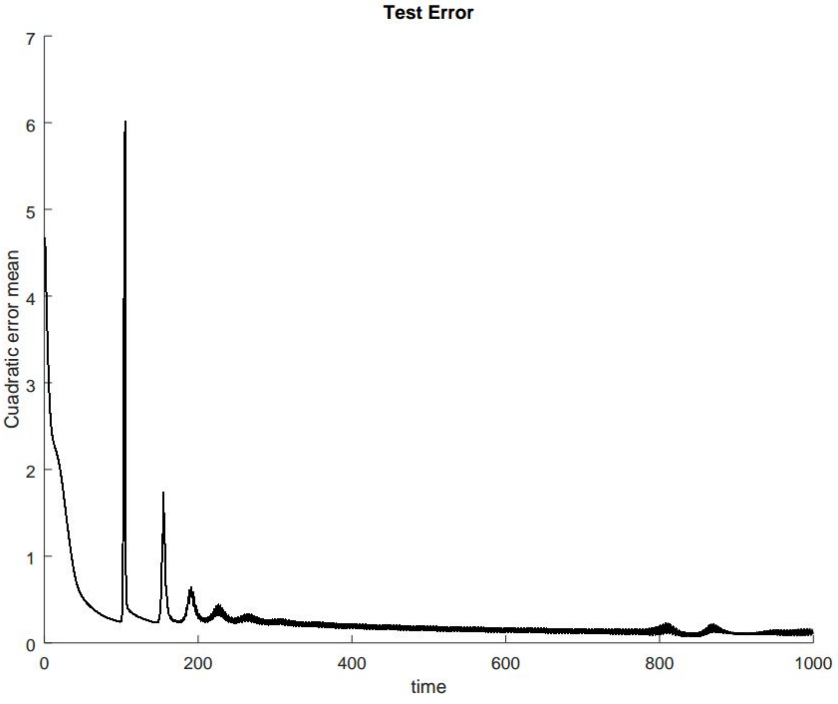


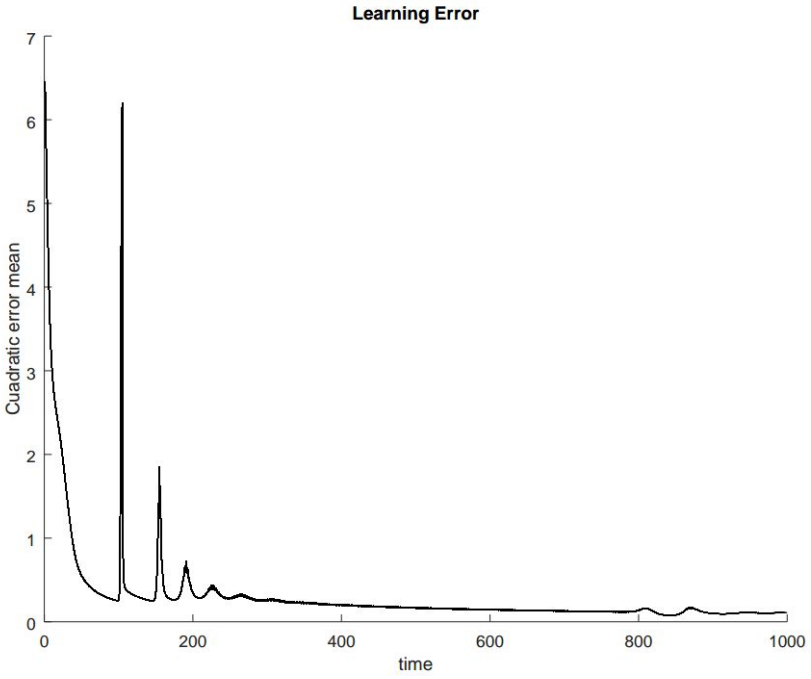

Learning Error
Cuadratic error mean
time
0
1
2
3
4
5
6
0
200
400
600
800
1000

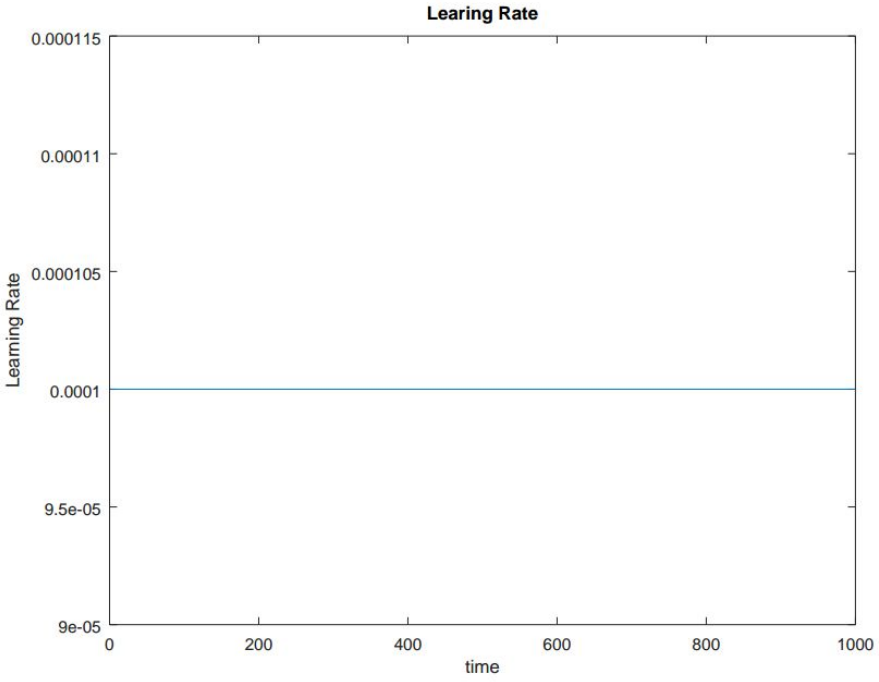

Learing Rate
Learning Rate
time
0.000115
0.00011
0.000105
0.0001
9.5e-05
9e-05
0
200
400
600
800
1000

Corrida 4 (Momentum & Constant Learning rate):

learningRate	0.0001
adaptiveLearningRate	0
timesLR	50
incLR	0.00005
decLR	0.25
momentum	1
momentumRate	0.9
maxError	0.05
epoch	1000
debugTimes	Inf
deltaWCalculation	batch
beta	1
gamma	0
function	tanh
betaLast	1
gammaLast	0
functionLast	linear
learningSamplePercentage	0.7
testingSample	terrain02.data
qtyNeuronsInLayer1	2
qtyNeuronsInLayer2	20
qtyNeuronsInLayer3	10
qtyNeuronsInLayer4	1

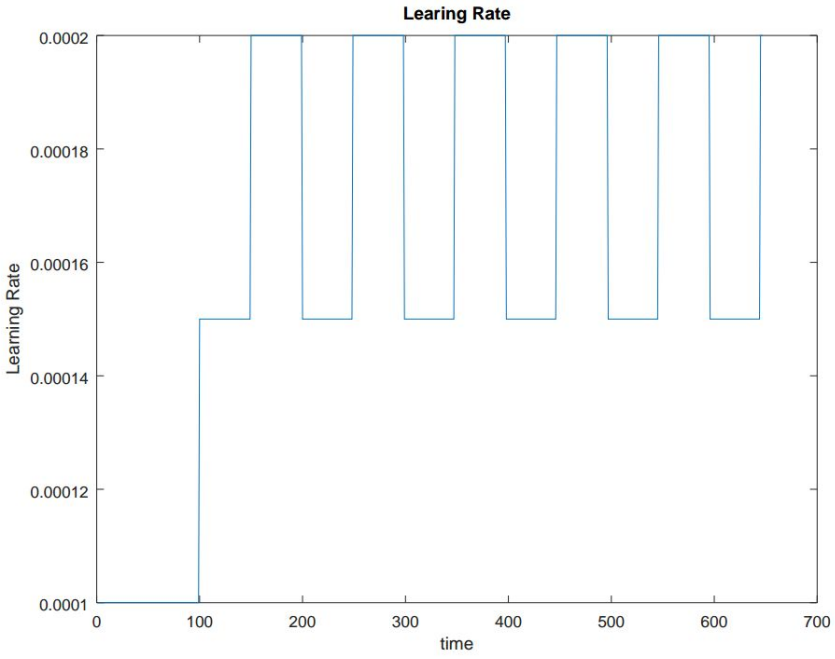


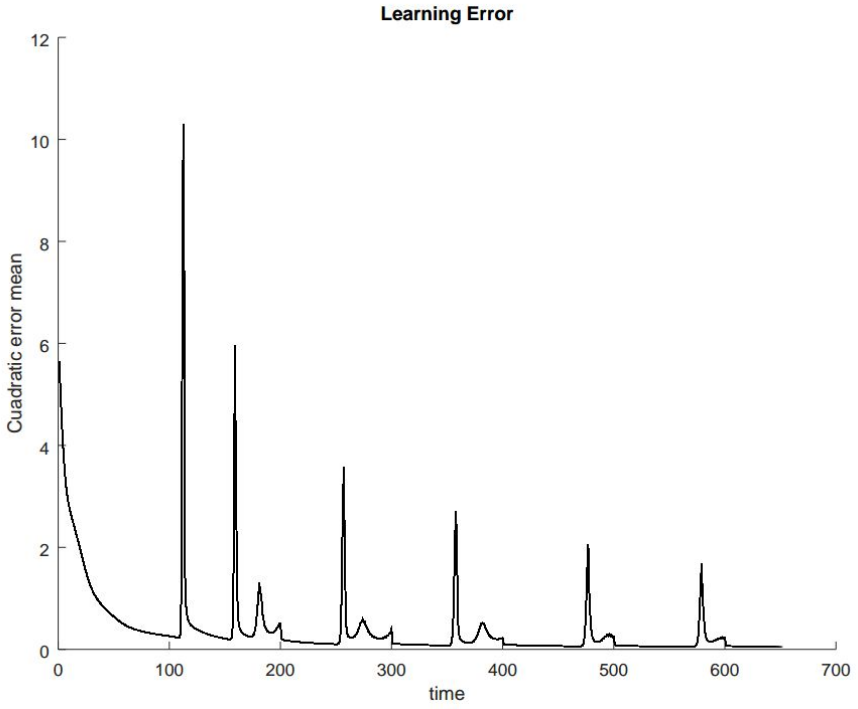

Learning Error
Cuadratic error mean
time
0
1
2
3
4
5
6
7
0
200
400
600
800
1000

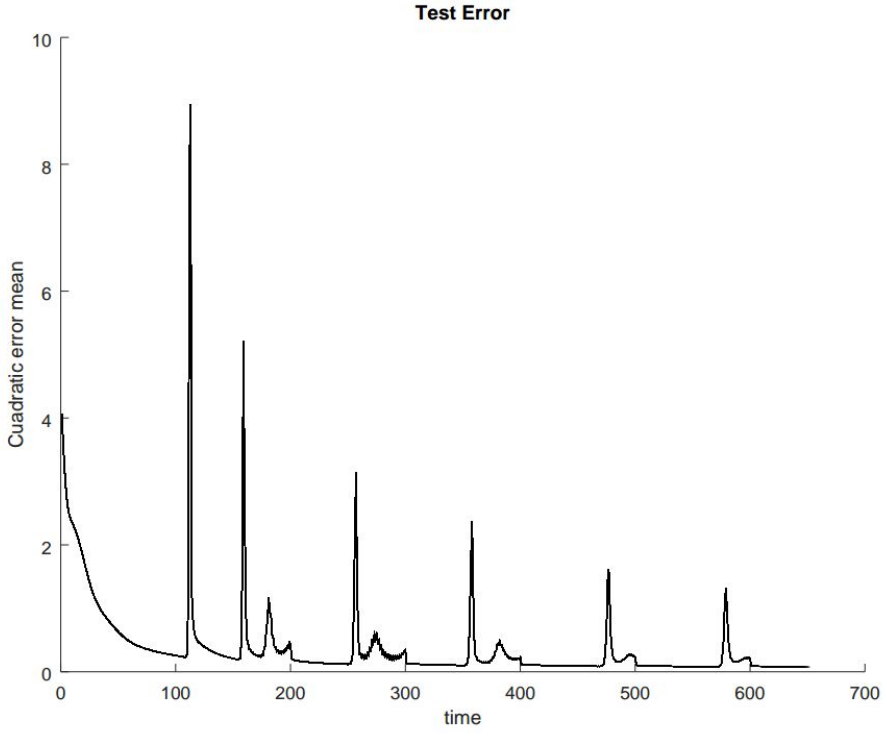

Learing Rate
Learning Rate
time
0.000115
0.00011
0.000105
0.0001
9.5e-05
9e-05
0
200
400
600
800
1000

Corrida 5 (Momentum & Adaptive learning rate):

learningRate	0.0001
adaptiveLearningRate	1
timesLR	50
incLR	0.00005
decLR	0.25
momentum	1
momentumRate	0.9
maxError	0.05
epoch	1000
debugTimes	Inf
deltaWCalculation	batch
beta	1
gamma	0
function	tanh
betaLast	1
gammaLast	0
functionLast	linear
learningSamplePercentage	0.7
testingSample	terrain02.data
qtyNeuronsInLayer1	2
qtyNeuronsInLayer2	20
qtyNeuronsInLayer3	10
qtyNeuronsInLayer4	1

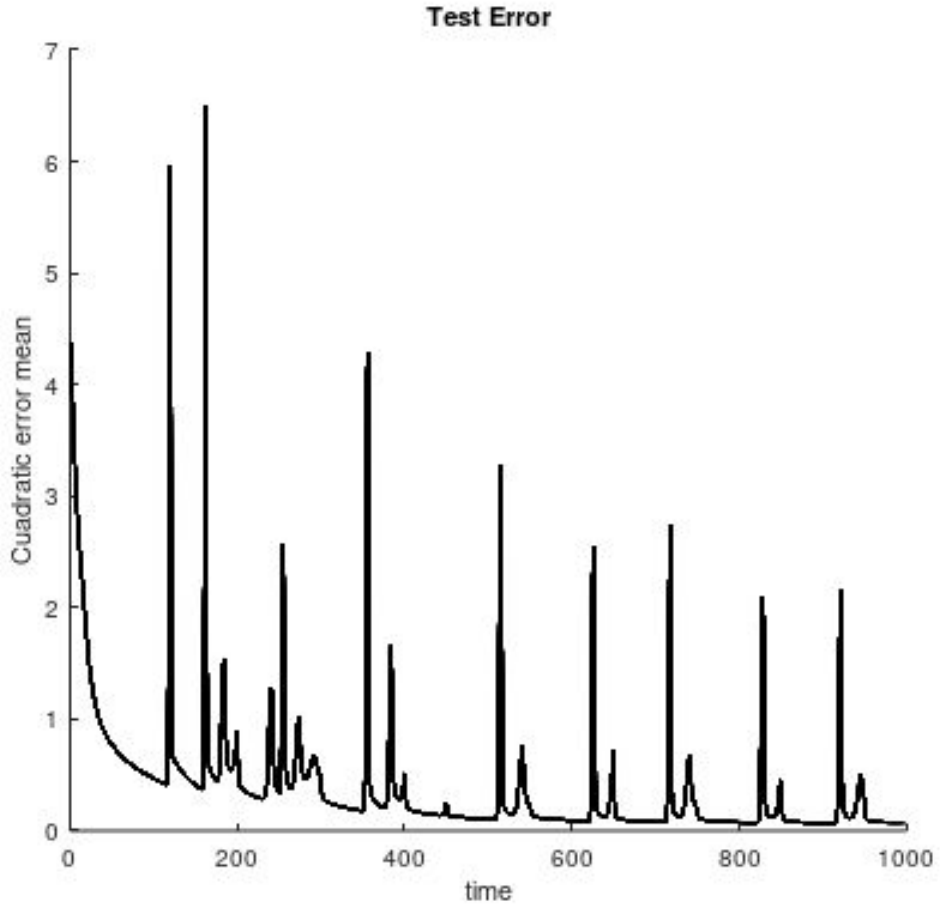



Learning Error
Cuadratic error mean
time
0
2
4
6
8
10
12
0
100
200
300
400
500
600
700


Test Error
Cuadratic error mean
time
0
2
4
6
8
10
0
100
200
300
400
500
600
700

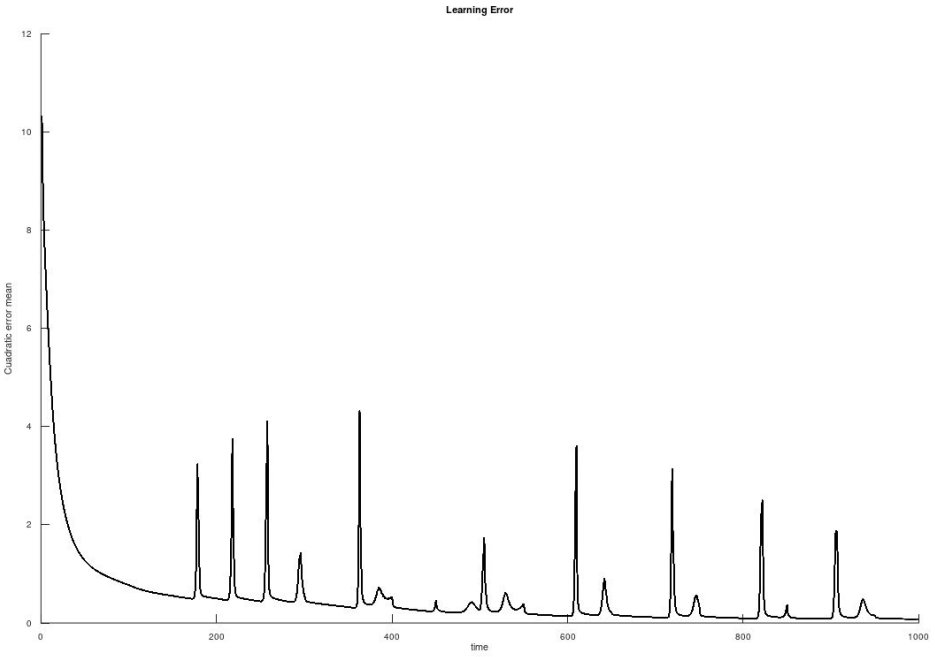
Corrida 6 (Learning sample 90%):

learningRate	0.0001
adaptiveLearningRate	1
timesLR	50
incLR	0.00005
decLR	0.25
momentum	0
momentumRate	0.9
maxError	0.05
epoch	1000
debugTimes	Inf
deltaWCalculation	batch
beta	1
gamma	0
function	tanh
betaLast	1
gammaLast	0
functionLast	linear
learningSamplePercentage	0.9
testingSample	terrain02.data
qtyNeuronsInLayer1	2
qtyNeuronsInLayer2	20
qtyNeuronsInLayer3	10
qtyNeuronsInLayer4	1



Corrida 7 (Learning sample 50%):

learningRate	0.0001
adaptiveLearningRate	1
timesLR	50
incLR	0.00005
decLR	0.25
momentum	0
momentumRate	0.9
maxError	0.05
epoch	1000
debugTimes	Inf
deltaWCalculation	batch
beta	1
gamma	0
function	tanh
betaLast	1
gammaLast	0
functionLast	linear
learningSamplePercentage	0.5
testingSample	terrain02.data
qtyNeuronsInLayer1	2
qtyNeuronsInLayer2	20
qtyNeuronsInLayer3	10
qtyNeuronsInLayer4	1


Learning Error
Cuadratic error mean
time
0
2
4
6
8
10
12
0
200
400
600
800
1000

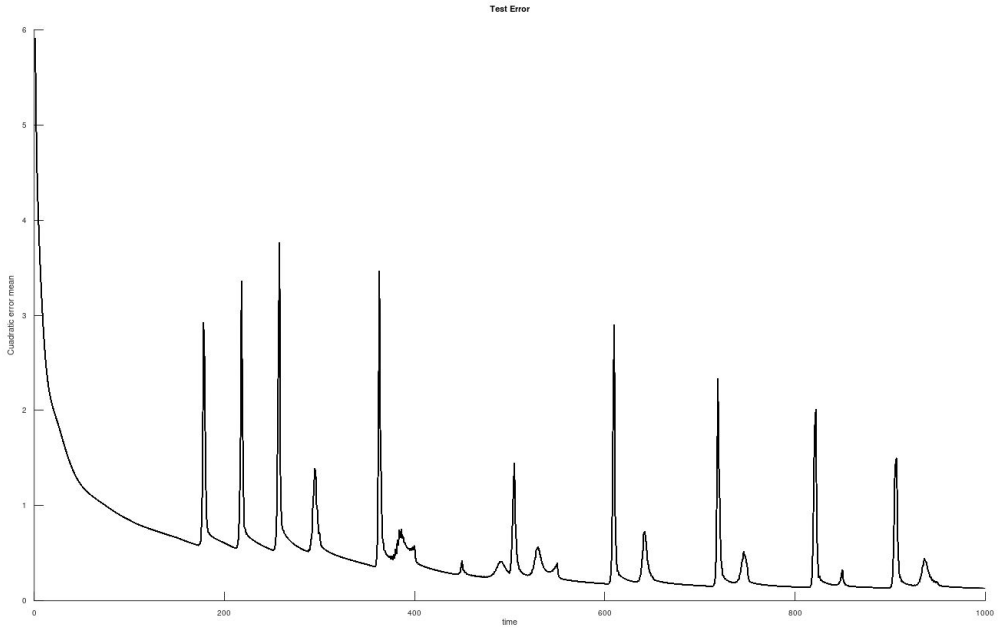

Test Error
Cuadratic error mean
time
0
1
2
3
4
5
6
0
200
400
600
800
1000

Figura 1: