Generalización de Terrenos con Redes Neuronales Multicapa

Sistemas de Inteligencia Artificial
Instituto Tecnológico de Buenos Aires

Garrigó, Mariano 54393
Raies, Tomás A. 56099
Saqués, M. Alejo 56047

Resumen

Se ha analizado el potencial de una red neuronal multicapa supervisada a los efectos de aprender la forma de un terreno, y, posteriormente, generalizar sobre la misma.

Se ha evaluado una serie de variaciones sobre el algoritmo de *backpropagation*, particularmente *adaptive eta* y *momentum*, como así también combinaciones de ambas técnicas. Asimismo, se han comparado los resultados obtenidos actualizando los pesos de la red tanto de manera incremental como en *batch*.

Asimismo, se han entrenado diversas redes con capacidad de generalizar terrenos de diferentes ciudades del mundo, tomando como referencia puntos tomados de mapas de altura de dichas ciudades.

Por último, se han comparado estas técnicas con el estado del arte en algoritmos de generación aleatoria de terrenos, como el *Diamond-Square Algorithm* (DSA). El objetivo ha sido probar las capacidades de una red neuronal de aprender terrenos de la vida real, a los efectos de evaluar la viabilidad de las redes neuronales a los efectos de generar terrenos de manera procedural.

Palabras clave: Redes multicapa supervisadas, *feature scaling*, *backpropagation*, *adaptive eta*, *momentum*, actualización de pesos incremental / en *batch*, funciones de activación, *Diamond-Square Algorithm*.

1. Descripción del entrenamiento

En esta sección, se discutirán las decisiones tomadas por el Equipo para entrenar redes neuronales. Esto incluye tanto toda la instancia de pre-procesamiento, como así también el entrenamiento propiamente dicho.

1.1. Pre-procesamiento

1.1.1. *Feature scaling*

Dado que las imágenes de las funciones de activación son intervalos en $\mathbb{R}^2$ ($tanh : \mathbb{R} \to [-1, 1]$ y $logistic : \mathbb{R} \to [0, 1]$), si los datos del *set* de entrenamiento no se encuentran dentro de dichos intervalos, cierto ajuste es necesario.

Para ello, se utiliza *feature scaling* para estandarizar muestras en algún intervalo deseado. De todas las variaciones de dicho método, se utiliza el *rescaling* de datos:

$$x_i^{'} = \frac{x_i - min(X)}{max(X) - min(X)}(b - a) + a \qquad (1)$$

Donde X es el espacio de muestras a estandarizar, a y b son las cotas inferior y superior del intervalo de llegada, respectivamente.

1.1.2. Selección de muestras de entrenamiento

Dado que una de las características que se quiere probar es la capacidad de generalización de la red, es razonable que parte de las

muestras del terreno se utilicen para verificar que la red, efectivamente, aprendió el problema. Por ende, se ha decidido utilizar el 90 % de las muestras para el entrenamiento, y un 10 % para testeo. Los patrones de cada conjunto se toman de manera no determinística. Se debe notar que la proporción es un dato parametrizable.

1.2. Implementaciones de *backpropagation*

A continuación, se elaborará sobre los diferentes algoritmos utilizados para entrenar las redes neuronales. Las descripciones atacarán fundamentalmente aquellos rasgos distintivos de los mismos, que emanan de decisiones tomadas por el Equipo durante el proceso de desarrollo.

Nota: El error cuadrático medio sobre el conjunto de datos de entrenamiento se calcula, para cada versión del algoritmo, después de ejecutar cada *epoch*.

1.2.1. Algoritmo incremental con *adaptive eta* y *momentum*

Tal como su nombre lo indica, las actualizaciones de pesos en este algoritmo se realizan patrón a patrón. Esto lleva consigo la necesidad de iterar sobre todos los patrones en cada *epoch*, desaprovechándose así las mejoras que realiza *Octave/Matlab* sobre el producto matricial. Esto se verá reflejado en la lentitud de ejecución de cada *epoch* por parte de este algoritmo, relativa a la versión en *batch* que se mencionará a continuación.

En lo que respecta al orden de las muestras de entrenamiento en cada *epoch*, se ha tomado la decisión de presentarlas en un orden no determinístico, dado que se ha observado que, de esta forma, es menos probable que el algoritmo se atasque en mínimos locales.

Por último, se han considerado dos variaciones sobre el proceso de eta adaptativo, las cuales versan sobre si después de k iteraciones en las que el error cuadrático medio descendió consistentemente, dicho contador se restablece tras incrementar el eta o si se prosigue incrementando *epoch* tras *epoch* hasta que haya un incremento en el error. Se ha notado que ambas variaciones presentan comportamientos similares si para la primera versión se establecen valores de k *pequeños* y, para la segunda versión, *grandes*. Para el caso, *pequeños* y *grandes* valores se refieren a cifras en el orden de 10^1 y 10^2 respectivamente. Se ha decidido utilizar la versión que restablece k tras cada incremento.

En lo que respecta a las condiciones de corte, el algoritmo retorna cuando:

1. Se ha llegado al máximo de iteraciones,
2. Se ha llegado al valor del error cuadrático medio deseado,
3. El eta se ha hecho ~ 0.
4. La diferencia entre el error cuadrático medio anterior y el actual es 0.

Nota: El ítem 3. refleja la situación en la que el error ha crecido consistentemente, haciendo que tras sucesivas disminuciones del eta dicho valor se acerque a 0.

1.2.2. Algoritmo en *batch* con *momentum*

En este algoritmo, las actualizaciones de pesos se realizan tras cada *epoch*. A diferencia del caso anterior, esto permite tomar las muestras del set de entrenamiento de manera conjunta, sin tener que iterar por cada patrón. De esta forma, se aprovechan las mejoras que *Octave/Matlab* realizan sobre el producto matricial, haciendo que la ejecución de cada *epoch* sea notoriamente más rápida que en el algoritmo anterior.

En lo que respecta a las condiciones de corte, el algoritmo retorna cuando:

1. Se ha llegado al máximo de iteraciones,
2. Se ha llegado al valor del error cuadrático medio deseado,

3. La diferencia entre el error cuadrático medio anterior y el actual es ~ 0.

1.2.3. Algoritmo en *batches* parametrizables

La idea de esta versión de *batch* es idéntica a la anterior, salvo que los *batches* se toman de a tamaños parametrizables. Esta implementación busca aprovechar las mejoras en eficiencia del producto de matrices de menores dimensiones.

2. Resultados del entrenamiento

En esta sección, se discutirán los mejores resultados logrados con ambos algoritmos, exhibiéndose la progresión del error y analizándose la precisión de la red obtenida al generalizar el terreno.

Glosario de terminología:

- f: Función de activación
- H: Capas ocultas (`array`)
- η: *Learning rate*
- *momentum*: factor del *momentum*
- α: constante de crecimiento del eta adaptativo
- β: factor de decrecimiento del eta adaptativo
- k: constante de decrecimiento consistente
- ϵ: cota inferior para η (condición de corte)
- ϵ_e: condición de corte del error cuadrático medio
- max_{iters}: número máximo de *epochs*.

2.1. Algoritmo incremental con *adaptive eta*

A continuación siguen los resultados obtenidos para el entrenamiento con el algoritmo incremental utilizando eta adaptativo.

2.1.1. Parámetros y resultados generales

- $f = \tanh(x)$
- $H = [10, 10]$
- $\eta = 0{,}1$
- $momentum = 0$
- $\alpha = 0{,}05$
- $\beta = 0{,}1$

- $k = 5$
- $\epsilon = 0{,}00001$
- $\epsilon_e = 0{,}0001$
- $max_{iters} = 10000$

En este primer caso de análisis, se ha logrado obtener una red neuronal que aproxima el terreno con un error cuadrático medio sobre el *set* de entrenamiento de $6{,}5 * 10^{-4}$. Es decir, la ejecución del algoritmo finalizó por haberse alcanzado el máximo de iteraciones. El *set* de pesos que menor error cuadrático medio logró se correspondió con la iteración 9954.

Un aspecto a destacar es la elección de no utilizar *momentum*. Por lo general, al combinar *momentum* con *adaptive eta*, se ha encontrado que el comportamiento del error a lo largo del tiempo es mucho más errático que sin utilizar el primero. Por ende, se ha optado por no usar *momentum* en este caso, optándose por usarlo como mecanismo de aceleración de *backpropagation* en el algoritmo por *batch*.

2.1.2. Tasas de éxito

Un indicador de qué tan acertado fue el entrenamiento de la red se puede encontrar en las tasas de éxito, es decir, en qué proporción la red neuronal generalizó correctamente un nuevo dato. Para ello, se está tomando como punto de comparación el *set* de muestras de prueba extraído tal como se indicó más arriba. Este conjunto representa el 10 % del conjunto original. Los intervalos dentro de los cuales se considera que el valor fue correctamente generalizado están definidos por el Épsilon (ϵ): $[S_i - \epsilon, S_i + \epsilon]$.

Épsilon	Éxito (%)
0.05	75.6
0.06	80
0.07	84.5
0.08	91.2
0.09	91.2
0.1	97.8
0.11	100

Cuadro 1: Tasas de éxito

Como se podrá ver, con valores relativamente pequeños del Épsilon se logra tasas altas de éxito, considerando que el espacio de llegada (la coordenada Z) toma valores en el intervalo $[-1, 1]$.

2.1.3. Errores cometidos en el set de comparación

En estrecha vinculación con el punto anterior, se puede analizar la diferencia entre el valor esperado y el valor emitido por la red. Esto está definido por:

$$D_i = abs(S_{c_i} - E_i) \tag{1}$$

Donde E_i es la evaluación del i-ésimo patrón, S_{c_i} es su salida esperada.

Este análisis arroja los siguientes resultados:

- $max(D) = 0{,}10006$
- $min(D) = 1{,}7035 * 10^{-4}$
- $mean(D) = 0{,}030684$

Cómo se puede ver, los valores están en línea con los resultados de la sección anterior. El valor más significativo en este punto es la media de las diferencias, que, como podrá observarse, es pequeña en relación a la longitud del espacio de llegada.

A continuación se presenta la gráfica de cada una de las diferencias entre los valores esperado y calculado.

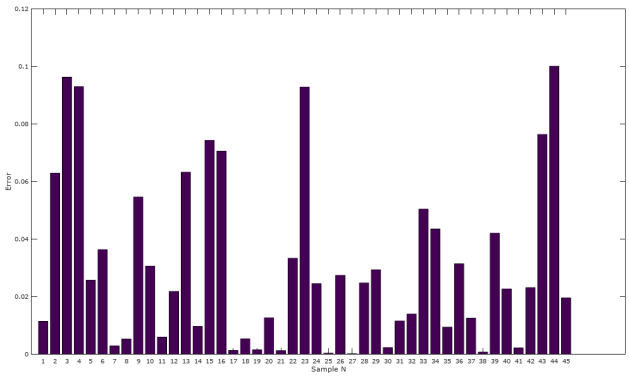

Figura 1: D_i

Como puede verse en la figura 1, existen picos en los cuales la diferencia es mucho mayor a la del resto. Analizar a qué puntos corresponden dichos valores permitirá comprender el comportamiento de la red.

Muestra	X	Y
44	-0.14806	0.31066
3	-1	-0.93007
23	1	-0.11425
4	0.061232	-0.240449

Cuadro 2: Puntos de mayor D_i

En la tabla 2 se pueden observar algunos datos interesantes. Las muestras 3 y 23 se localizan aproximadamente en dos diferentes puntas del terreno. Esto podría estar indicando que la red neuronal tiene mayores dificultades en aprender las puntas por ciertas características particulares que las mismas podrían tener.

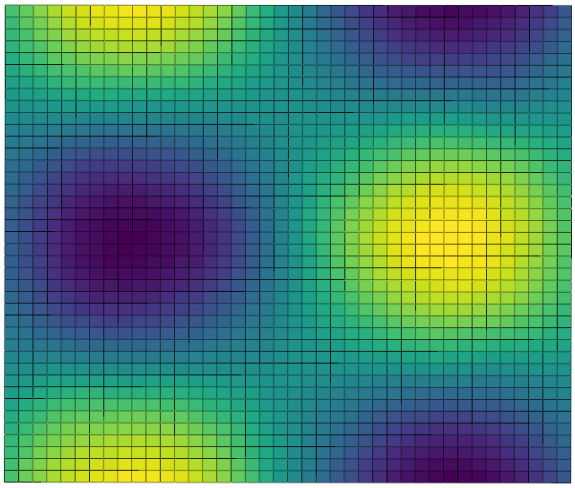

Figura 2: Mapa de altura

Como se podrá ver en el mapa de altura de la figura 2, realizado mediante una generalización realizada por la red, en las puntas hay o bien depresiones muy acentuadas o elevaciones pronunciadas. Nótese que colores mas azulados reflejan puntos más bajos.

Observando los valores de la muestra 4, puede afirmarse que este se encuentra en las inmediaciones de uno de los extremos del terreno, por lo que es de esperar que el comportamiento sea similar al de los casos anteriores.

2.1.4. Análisis del error cuadrático medio

Para analizar la evolución del error cuadrático medio en el tiempo, la gráfica a continuación puede ayudar a entender su comportamiento.

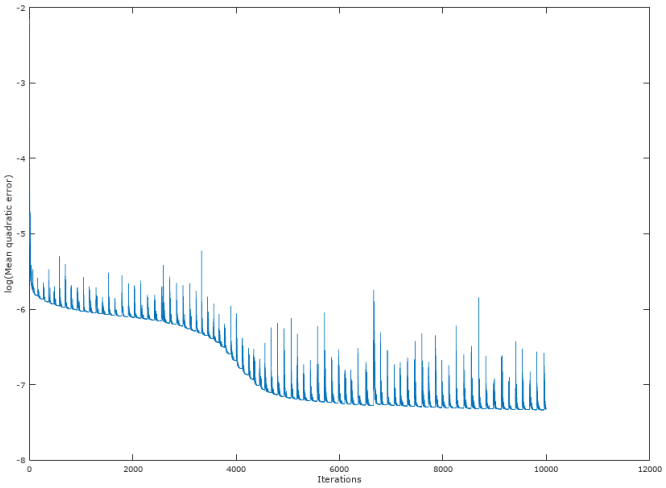

Figura 3: Error cuadrático medio

La figura 3 es una gráfica en escala logarítmica del error cuadrático medio en función del número de iteración. El aspecto quizás más interesante que se puede observar es la periodicidad con la que el error muestra picos de aumento, rápidamente corrigiéndose para mostrar el error una tendencia decreciente. Dicha periodicidad está relacionada con el valor k, es decir, el número de *epochs* que se considera que expresa un decrecimiento consistente en el error. Es probable que lo que se observe sea que, al aumentar el eta, el error inicialmente aumenta para luego corregirse.

Otro aspecto interesante es el abrupto descenso del error en torno a la iteración 4000, sobre todo teniendo en cuenta que la pendiente de la curva, hasta ese momento, era similar a lo que se encuentra en torno a la iteración 6000, es decir, después de dicho suceso. Queda como interrogante el analizar si un descenso como tal podría haber ocurrido de haberse proseguido la ejecución. De todas formas, se considera que el error alcanzado resuelve con creces el objetivo del problema con errores mínimos, tal como se ha discutido en secciones anteriores.

2.2. Algoritmo en *batch* con *momentum*

En este apartado se discutirán los resultados obtenidos al entrenar la red neuronal realizando las actualizaciones de pesos en *batch*.

2.2.1. Parámetros y resultados generales

- $f = \tanh(x)$
- $H = [10, 10]$
- $\eta = 0{,}001$
- $momentum = 0{,}9$
- $\epsilon_e = 0{,}0001$
- $max_{iters} = 100000$

Si se compara el *set* de parámetros iniciales con el del caso anterior, se podrá notar que el η es considerablemente más pequeño. Esta decisión se puede asociar con el factor de que, a valores de η *grandes* (mayores a 10^{-2}), el algoritmo tiende a atascarse en mínimos locales.

Este último comentario está estrechamente vinculado con la decisión de utilizar *momentum*. Dado que *momentum* busca acelerar el proceso de entrenamiento adicionando una proporción de ΔW_{prev} (en particular, 0,9), valores de η relativamente *grandes* pueden hacer desviar al algoritmo del camino correcto, así atascándose en mínimos locales o, incluso, aumentando el error indefinidamente −esto último se ha notado en algunos casos utilizando *momentum* en el algoritmo incremental con *adaptive eta*. Visto esto, se puede decir que la decisión tomada con respecto a los valores iniciales, además de − como se verá − funcionar en la práctica, tiene sus fundamentos teóricos.

Con respecto a los resultados de la ejecución, se ha obtenido un error cuadrático medio sobre el conjunto de datos de entrenamiento de $3{,}26 * 10^{-4}$, es decir, prácticamente la mitad del error obtenido en el caso anterior. Sin embargo, es de notar que dicho valor fue conseguido en 100000 *epochs*, lo cual supuso un tiempo de entrenamiento de 01 : 15 horas en una computadora con un procesador **Intel Core i7-4700MQ**. Este tiempo supera con creces al del caso anterior, que no superó la media hora. El análisis de la generalización de esta red ayudará a discernir entre qué red presenta una mejor relación resultados/esfuerzo computacional.

2.2.2. Tasas de éxito

Al igual que el caso anterior, se buscará analizar dentro de qué intervalos la red obtenida generaliza correctamente datos de prueba, es decir, no vistos durante el entrenamiento.

Épsilon	Éxito (%)
0.05	88.9
0.06	93.4
0.07	95.6
0.08	97.8
0.09	100
0.1	100

Cuadro 3: Tasas de éxito

Como podrá verse, a diferencia del caso anterior, todas las muestras de prueba son generalizadas correctamente en un intervalo de $\pm 0{,}1$, teniendo en torno a un 89 % de éxito incluso con intervalos el doble de pequeños. Analizando el último caso mencionado, este valor es alrededor de un 14 % mejor que su equivalente en los resultados del algoritmo anterior.

2.2.3. Errores cometidos en el set de comparación

Algunas de las cifras que describen a los resultados del *set* de comparación son las siguientes:

- $max(D) = 0{,}085488$
- $min(D) = 9{,}3254 * 10^{-4}$
- $mean(D) = 0{,}021261$

El valor que más dista del obtenido en la ejecución anterior es el $min(D)$, que se aproxima a ser 1 orden de magnitud inferior a su equivalente del caso anterior. Por lo demás, los otros valores son, razonablemente inferiores, siendo la media de la diferencia entre los valores esperado y calculado inferior en torno a $\frac{1}{3}$.

El siguiente gráfico muestra los errores de cada coordenada:

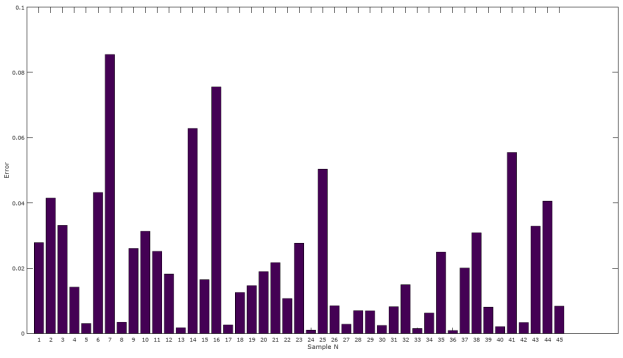

Figura 4: D_i

Mediante una simple inspección de la gráfica en 4, comparándola con su equivalente anterior (1), se puede percibir que la cantidad de *picos* − barras cuya longitud es más del doble que el resto − no solamente es inferior, sino que son de menor magnitud.

De todas formas, se procederá a realizar el mismo análisis con respecto a la ubicación de los puntos con generalización más deficiente:

Muestra	X	Y
7	1	0.58233
16	1	0.82133
14	-0.14806	-0.35267
41	0.94275	-0.24045

Cuadro 4: Puntos de mayor D_i

Como podrá notarse en el cuadro 4, las conclusiones que se pueden extraer son similares a las encontradas en (2). Las muestras 16 y 41 se encuentra aproximadamente en dos diferentes puntas, el caso 7 sobre la mitad de un borde, y el caso 14 en alguna región de alto gradiente. La conclusión que se puede extraer a propósito de esto es que más puntos pueden ser necesarios en estas regiones si se precisa un muy buen nivel de detalle, y que las dificultades son independientes de los algoritmos utilizados. Esto no quita que no exista una estructura de red neuronal que tenga un mejor potencial para aprender dichas regiones, pero se afirma que, de las decenas de configuraciones probadas, la que se presenta es la que mejores resultados ha exhibido.

2.2.4. Análisis del error cuadrático medio

Para concluir el análisis de los resultados para esta ejecución, a continuación se discutirá el comportamiento del error cuadrático medio en función del tiempo.

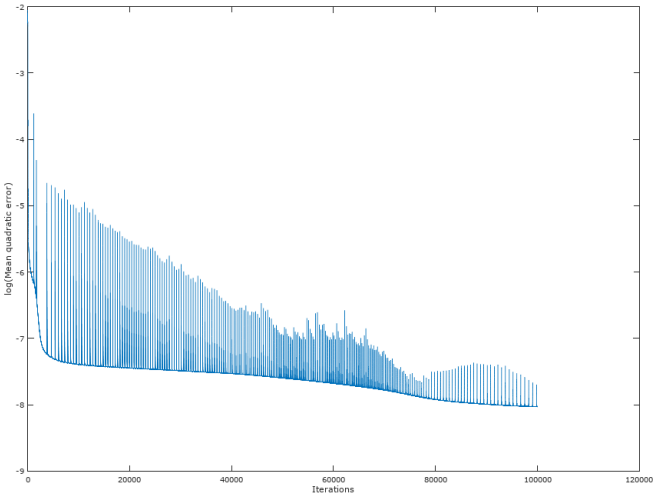

Figura 5: Error cuadrático medio

Lo que sin lugar a dudas impacta al observar esta gráfica es la periodicidad en con la que ocurren picos muy significativos en el error. Sin embargo, es de notar que la magnitud de dichos saltos tiende a decrecer en el tiempo, para lograr que el error tenga una tendencia generalmente decreciente.

Se puede observar que, dentro de las primeras 20000 iteraciones, los errores oscilan entre valores de $9 * 10^{-4}$ $(\exp(-7))$ y $6{,}7 * 10^{-3}$ $(\exp(-5))$, lo cual representa saltos de 1 orden de magnitud.

Adicionalmente, puede notarse como hacia la iteración 80000 hubo un marcado incremento en la tasa de decrecimiento del error cuadrático medio, para luego asentarse en niveles comparados a los anteriores.

Por último, observando que en ciertos puntos de la curva − principalmente dentro de las primeras 20000 iteraciones − existen puntos con mayor tasa de descenso del error, podría suponerse que tomar valores de η más grandes dicha tasa se incrementaría de manera positiva. En los casos que se ha probado esto − siendo una suerte de variante del *adaptive eta* −, los resultados han sido negativos.

2.3. Algoritmo en *batches* parametrizables

Por último, se analizarán algunas diferencias relevantes entre la implementación de *batches* de tamaño fijo contra los parametrizables.

2.3.1. Parámetros y resultados generales

- $f = \tanh(x)$
- $H = [10, 10]$
- $\eta = 0{,}001$
- $momentum = 0{,}9$
- $\epsilon_e = 0{,}0001$
- $max_{iters} = 15000$

El objetivo de esta ejecución fue observar la evolución del error cuadrático medio, a los efectos de compararla con los resultados anteriores. Por ende, se realizó únicamente el 15 % de las iteraciones que el caso anterior. Por lo demás, todos los demás valores iniciales permanecieron inalterados.

En 15000 *epochs*, se ha logrado un error de $4{,}22 * 10^{-4}$, es decir, un punto medio entre las dos ejecuciones anteriores. Este resultado se la logrado en 00 : 10 : 30 minutos, en la misma computadora que el caso anterior.

2.3.2. Tasas de éxito

Las tasas de éxito para este caso fueron las siguientes:

Épsilon	Éxito (%)
0.05	77.8
0.06	88.9
0.07	95.6
0.08	97.8
0.09	100
0.1	100

Cuadro 5: Tasas de éxito

Es decir, si bien resultaron ser sensiblemente inferiores al caso anterior, esta red logra 100 % de aciertos con un intervalo de error idéntico, de ±0,09. Esto en una fracción del tiempo de ejecución.

2.3.3. Datos significativos del *set* de comparación

Las cifras significativas del *set* de comparación son las siguientes:

- $max(D) = 0{,}084863$
- $min(D) = 0{,}0011897$
- $mean(D) = 0{,}028660$

Peculiarmente, el máximo de las diferencias es muy sensiblemente inferior al del caso anterior, mientras que el mínimo es un orden de magnitud superior. La media, si bien es superior a la observada en el caso anterior, sigue siendo considerablemente mejor que el primer caso analizado.

2.3.4. Análisis del error cuadrático medio

Por último, lo más interesante para este caso se encuentra en la gráfica del error cuadrático medio.

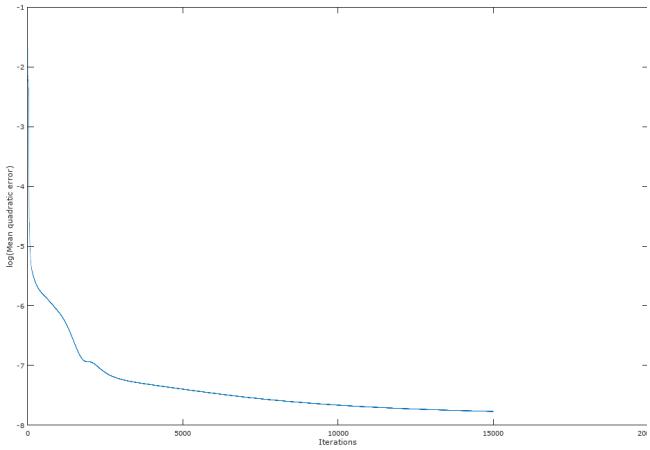

Figura 6: Error cuadrático medio

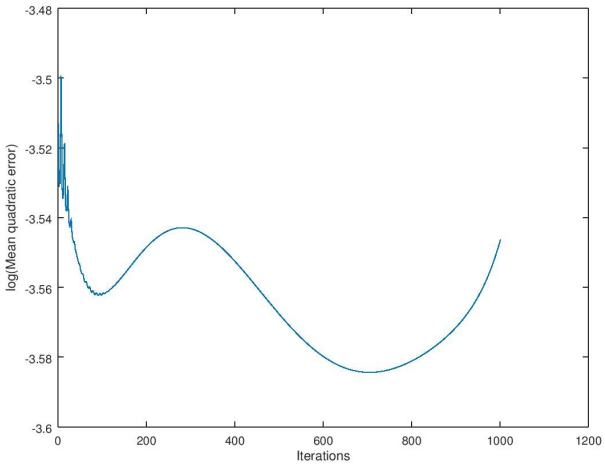

Figura 7: Error cuadrático medio

Como podrá verse, a diferencia de los casos anteriores, no se observan saltos en los valores del error cuadrático medio. Esto puede estar vinculado con una elección acertada de los pesos iniciales que, si bien son inicializados de manera no determinística, ligeros cambios pueden guiar a resultados más precisos o, en este caso, con un descenso del error más suave.

Como detalle peculiar, llama la atención lo que puede observarse en torno a la iteración 2000: un pequeño estancamiento en un mínimo local que fue subsanado rápidamente por la suma de pesos tras cierta *epoch*. Es razonable pensar que, sin el accionar del *momentum*, dicho cambio pudo no haber existido.

2.4. Función logística

Como se ha observado, en ninguno de los casos anteriores se ha utilizado la función logística como función de activación. A continuación se exhibe un gráfico del comportamiento observado en la gran mayoría de los casos con los que se ha probado dicha función.

Para esta ejecución, se han utilizado los mismos valores iniciales de la sección 2. de este apartado, con la diferencia que la función de activación es la logística.

La gráfica muestra claramente un comportamiento errático, mostrando picos y valles de manera alternada. Visto esto, y considerando el tiempo que pueden demorar las redes neuronales en entrenar, se consideró que este comportamiento no es de fiar para ejecuciones que pueden durar horas. Por ende, se ha decidido estandarizar el uso de la función de activación tangente hiperbólica.

2.5. Otras arquitecturas

Como se puede haber notado, todas las arquitecturas hasta ahora exhibidas son idénticas: dos capas con sendas 10 neuronas. A continuación se mostraran algunos de los comportamientos observados en otras arquitecturas que fueron descartadas.

Nota: Todas las redes de esta sección se han entrenado con el algoritmo de *batch* con *momentum*, con idénticas configuraciones a las de la sección 2 de este apartado.

2.5.1. Mínimos locales

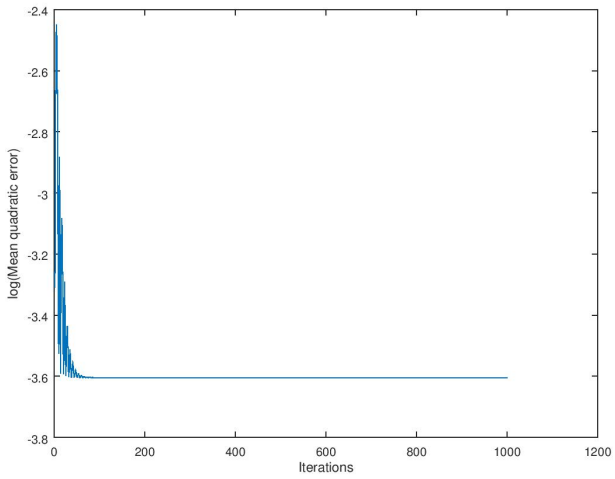

Figura 8: *Batch* con *momentum*, $H = [5, 5]$

En la figura 8, se puede ver como la pendiente de la curva del error cuadrático medio tiende a 0, luego de haber oscilado dentro de las primeras 50 iteraciones. Este comportamiento permitió, en muchos casos, descartar muchas arquitecturas candidatas en el proceso de entrenamiento.

2.5.2. Incremento del error

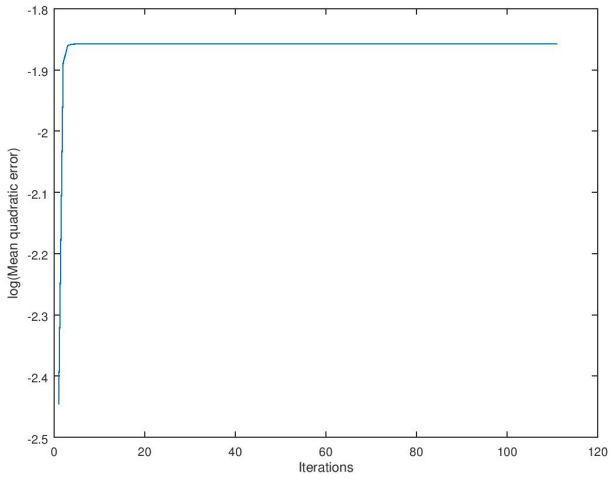

Figura 9: *Batch* con *momentum*, $H = [80]$

La gráfica 9 exhibe otro de los comportamientos encontrados: en vez de decrecer, la función del error cuadrático medio crece rápidamente para luego atascarse en un mínimo local.

2.6. Conclusiones

Como conclusión de esta sección, se puede afirmar que, para los alcances del problema, las tres redes neuronales propuestas los resuelven con creces. En el peor de los casos, se ha conseguido generalizar correctamente en torno al 75 % de los casos de prueba.

La discusión, por ende, se centrará más en la relación que existe entre la precisión de la generalización y el esfuerzo computacional requerido para obtener los pesos de la red. Con respecto a este punto, se rescatan las fortalezas exhibidas por el último caso, que combina errores comparables a los de la segunda ejecución, insumiendo su entrenamiento un tiempo moderado. Por ende, al hacer un balance entre estas características, la tercera red sería la red escogida para una aplicación en producción.

Si bien el algoritmo incremental con *adaptive eta* presenta una buena performance, su rasgo característico − que η se adapte en función al error − hace que la elección de los parámetros de crecimiento y decrecimiento correctos sea crítica para la estabilidad del algoritmo. Valores muy altos de la constante de crecimiento y bajos del factor de decrecimiento pueden hacer que el algoritmo recaiga en una situación en la que, tras un incremento del η, el error vaya en aumento y el factor de decrecimiento no pueda contener tal situación. Valores muy bajos de la constante de crecimiento hacen que el proceso de aceleración del entrenamiento sea muy lento.

En una implementación sin *adaptive eta*, pero utilizando *momentum* para acelerar el entrenamiento, estos parámetros se eliminan, haciendo que, para los criterios del Equipo, el entrenamiento sea más estable. Este es uno de los argumentos detrás de no haber permitido que el primero de los algoritmos ejecutara una cantidad de *epochs* en el orden del segundo − considérese asimismo que la ejecución de cada *epoch* del algoritmo incremental es inherentemente computacionalmente más costosa.

3. Comparación con otros algoritmos de generación de terreno

La generación de terrenos procedural es un tema de estudio de suma importancia en determinadas áreas, siendo su aplicación de sumo interés en el desarrollo de videojuegos. Existe un gran número de videojuegos de mundo abierto que buscan proveer al jugador de experiencias nuevas partida tras partida, y uno de los mecanismos para lograr esto es ofrecerle constantemente paisajes desconocidos o exóticos.

Existe una variedad de algoritmos que buscan ofrecer estas características, entre los cuales se puede nombrar el **Diamond-Square Algorithm (DSA)** [1] y **Perlin noise** [2]. En la siguiente sección se mostrarán algunos resultados que ofrece el primer algoritmo.

3.1. Diamond-Square Algorithm

El objetivo de **Diamond-Square Algorithm (DSA)** es simple: se parte de una matriz de dimensión 2^N+1 cuyas puntas están inicializadas en valores determinados por el usuario (*seeds*). Para mayor aleatoriedad, es deseable que dichas *seeds* sean provistas por un generador pseudoaleatorio.

El objetivo, luego, es rellenar toda la matriz, utilizando estas *seeds* como base, sumando ruido en cada paso − idealmente de magnitud decreciente en el tiempo. La explicación completa detrás del algoritmo se puede encontrar en [1].

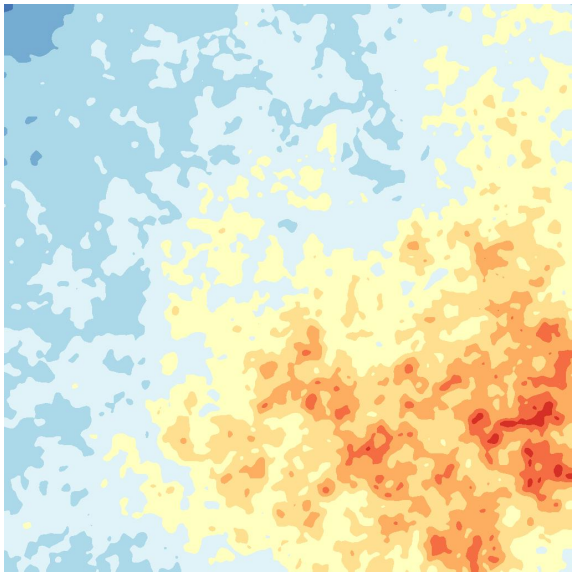

Figura 10: Ejemplo de generación de terreno con **DSA**

Como puede verse en el mapa de altura en 10, **DSA** es capaz de generar terrenos sumamente interesantes, tanto con océanos como zonas costeras y montañas. Las características del terreno generado son parametrizables. Una implementación en `Java` puede encontrarse en [4].

3.2. Potencial de las redes neuronales para generación de terreno aleatorio

El objetivo de esta sección es, basándose en el estado del arte en generación procedural de terrenos, analizar el potencial de las redes neuronales multicapa para generar similares resultados.

Dado que 4 *seeds* son insuficientes para entrenar las redes neuronales, y dado que se quiere generar un terreno de apariencia aleatoria pero preservando armonía, se ha optado por tomar como base de datos de entrenamiento mapas de altura de diversas ciudades del mundo. La herramienta utilizada para tal fin se puede encontrar en la siguiente referencia [3].

Nota: Para las siguientes generalizaciones, se ha utilizado el algoritmo en *batch* con *momentum* de la sección 2.2.

3.2.1. Punta del Este

Con la herramienta mencionada, se ha tomado una captura del mapa de altura de la península de Maldonado, incluyendo la Isla Gorriti. Se han tomado 512 muestras equiespaciadas de una imagen de $1024x1024$ píxeles, utilizándose el 90 % para el conjunto de entrenamiento, y se ha utilizado el algoritmo mencionado con $max_{iters} = 10000$, lográndose un error cuadrático medio del orden de 10^{-2}.

Figura 11: Vista cartográfica

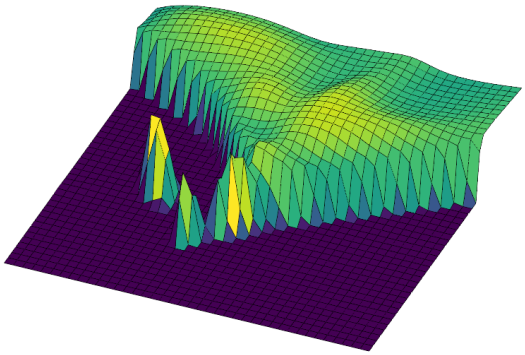

Figura 12: Generalización lograda

Como puede observarse en la gráfica tridimensional en 12, la generalización del terreno sigue las líneas que se pueden observar en el mapa, incluso con los niveles de error detallados. Se puede notar una pérdida de definición en el área de la península y en la Isla Gorriti, lo cual puede tener que ver con la baja cantidad de muestras que se han tomado. Esto podría ser subsanado con un mayor nivel de muestras en dichas zonas.

3.2.2. San Petersburgo

Se ha tomado un mapa de altura de la bahía de San Petersburgo, Rusia. Se han extraído 1024 muestras equiespaciadas de una imagen de $1024x1024$ píxeles, utilizándose el 90 % para el conjunto de entrenamiento, y se ha utilizado el algoritmo mencionado con $max_{iters} = 5000$, lográndose un error cuadrático medio de $8{,}5 * 10^{-3}$.

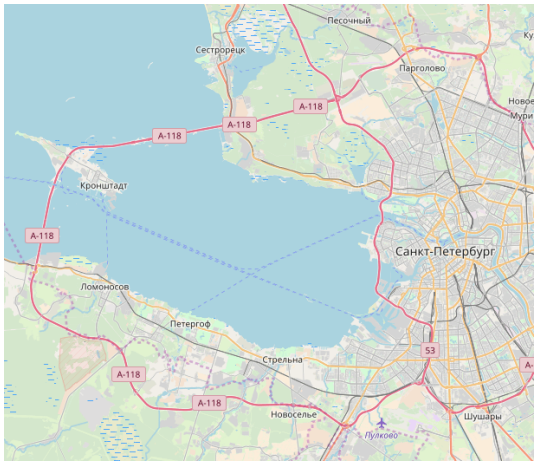

Figura 13: Vista cartográfica

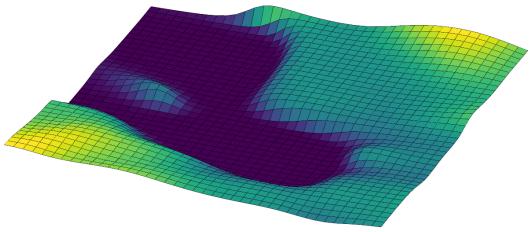

Figura 14: Generalización lograda

Como se podrá observar, en líneas generales la generalización ha sido correcta. Dado el bajo nivel de muestras, en la ciudad propiamente dicha se pierde precisión, pudiéndose observar un ligero detalle de la desembocadura del río Neva en el golfo de Finlandia. Además, puede notarse que al norte y al sur hay dos elevaciones, las cuales constituyen un error, dado que el área es perfectamente plana (pantano).

3.2.3. Dover

Se ha tomado un mapa de altura de la ciudad de Dover, Inglaterra. Se han extraído 1024 muestras equiespaciadas de una imagen de $1024x1024$ píxeles, utilizándose el 90 % para el conjunto de entrenamiento, y se ha utilizado el algoritmo mencionado con $max_{iters} = 5000$, lográndose un error cuadrático medio de $8{,}25 * 10^{-3}$.

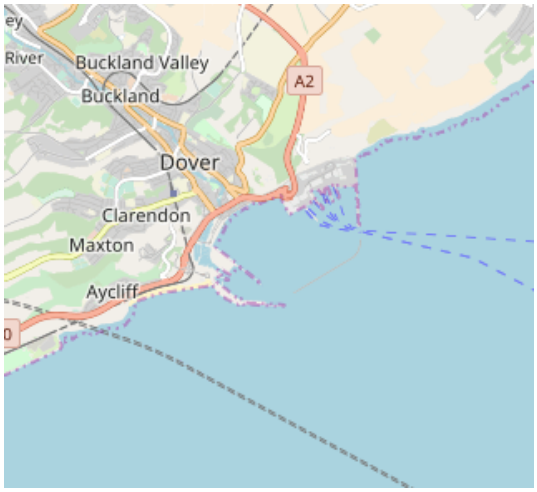

Figura 15: Vista cartográfica

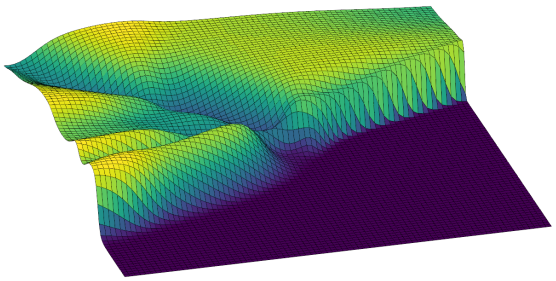

Figura 16: Generalización lograda

Las mayores dificultades se observan muy claramente en la zona del puerto. Sin embargo, las colinas al sur de la ciudad han sido generalizadas satisfactoriamente.

3.3. Conclusiones

Las redes neuronales tienen cierto potencial para generalizar terrenos reales, aunque debe notarse que los órdenes de magnitud de los errores cuadráticos medios exhibidos en esta sección son uno o dos veces superiores que los mostrados en la sección 2. El terreno asignado al equipo, si bien abunda en colinas y valles, es eminentemente curvado, sin haber picos de altura muy marcados en comparación a sus alrededores. En las ciudades reales probadas puede no existir tal comportamiento, a excepción que la ciudad se encuentre sobre una llanura perfectamente plana.

A pesar de todos estos elementos, se rescata la capacidad de las redes neuronales de realizar generalizaciones aceptables. Naturalmente un algoritmo como **DSA** será más capaz al generar terreno aleatorio por el simple hecho de que ha sido creado para tal fin, destacando el hecho que dentro de las capacidades de **DSA** no está la replicación de terrenos reales.

Se deja como propuesta la idea de producir un algoritmo enteramente autónomo de generación de terreno basado en redes neuronales, utilizando como valores de entrenamiento *seeds* generadas por un algoritmo pseudoaleatorio con una distribución con reducido desvío estándar, a los efectos de generar terrenos armoniosos.

Referencias

[1] WIKIPEDIA.ORG, **Diamond-Square Algorithm**, https://en.wikipedia.org/wiki/Diamond-square_algorithm

[2] WIKIPEDIA.ORG, **Perlin noise**, https://en.wikipedia.org/wiki/Perlin_noise

[3] TERRAIN.PARTY, **The easiest way to get real-world height maps for Cities: Skylines**, https://terrain.party/

[4] GITHUB, **Diamond-Square Algorithm implementation**, https://github.com/saques/DiamondSquare