Trabajo Práctico Especial 1

Redes Neuronales

Integrantes del Grupo 11:

Alonso, Juan Manuel
Cavallin, Florencia
Krammer, Esteban
Scomazzon, Martina

Profesores a cargo:

Parpaglione, María Cristina
Luciani, Ignacio

SISTEMAS DE INTELIGENCIA ARTIFICIAL I INSTITUTO TECNOLOGICO DE BUENOS AIRES I MARZO 2018

ÍNDICE

INTRODUCCIÓN

Con el propósito de aplicar los conocimientos aprendidos en clase, desarrollamos una red neuronal multicapa con aprendizaje supervisado que a partir de mediciones de altura, latitud y longitud puede simular terrenos de diferentes partes del mundo.

Para obtener una arquitectura óptima, utilizamos diferentes configuraciones de red y modificaciones en el algoritmo de aprendizaje de *Backpropagation*.

¿A que llamamos que una red sea optima? Consideraremos que una red neuronal es mas optima que otra si no solo tarda menos tiempo en encontrar una solución de pesos adecuados tal que el error cuadrático medio sea menor o igual al establecido por parámetro, épsilon, si no que también buscamos que la red aprenda y no memorice.

A continuación, se detalla qué tipos de arquitecturas evaluamos y qué efectos tuvieron en la performance de la red neuronal. Por ultimo, detallaremos las pruebas realizadas y el porque de las mismas que nos ayudaran luego a generar una conclusión.

ARQUITECTURA DE LA RED

No existe forma de establecer con exactitud la cantidad de capas ocultas y neuronas necesarias en ellas para obtener un 'buen' resultado.

Ahora bien, ¿A qué nos referimos cuando hablamos de un buen resultado? Como en todo programa, el tiempo importa, pero no es el principal problema a abordar. Debemos tener en cuenta que nuestro objetivo es que la red aprenda, nos provea de los pesos apropiados para poder luego llevar a cabo los testeos correspondientes y obtener un buen porcentaje de puntos acertados.

Luego de varios intentos consideramos que una buena arquitectura podría estar compuesta de dos capas internas donde la cantidad de neuronas en cada una puede variar entre 3 y 10 neuronas, dos neuronas de entrada y una de salida. Las dos neuronas de entrada corresponden a la latitud y longitud respectivamente y ademas solo necesitamos obtener un único resultado, la altitud, por ello la neurona de la última capa, la capa de salida.

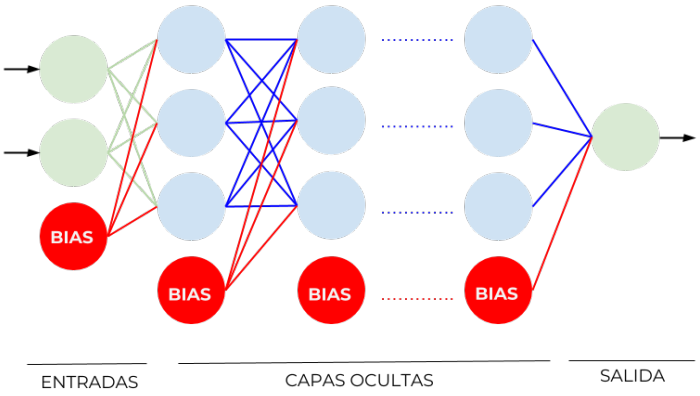


Figura 1. Red neuronal multicapa implementada

Además de la estructura planteada, debemos destacar la existencia de una unidad adicional tanto en la capa de entrada como en las capas ocultas dicha unidad se la conoce como *bias* y su valor de entrada es siempre -1.

Por último, las unidades de una capa están unidas a través de un enlace pesado, representando los pesos, a todas las unidades de la capa siguiente conservando las características de la implementación feed-forward, es decir, no existe retroalimentación en la red (ciclos o lazos).

DETALLES DE IMPLEMENTACIÓN

A la hora de desarrollar el algoritmo que garantizará el correcto funcionamiento de aprendizaje de la red, se tomaron las siguientes decisiones basadas en la funcionalidad de cada parámetro.

Elección de patrones

Para este proyecto se nos brindó, una lista de puntos que describen un terreno. Es de gran importancia de que manera se eligen los puntos para entrenar la red neuronal. Una mala elección de los mismo causaría un aprendizaje incorrecto ya que tomar puntos consecutivos, demasiado altos o demasiado bajos llevaría a un entrenamiento imparcial por lo que decidimos optar por una selección de puntos totalmente aleatoria.

Normalización de patrones

En casos en donde los patrones tienen valores muy diferentes, es importante normalizarlos ya que los valores de mayor magnitud afectarían mucho más a los pesos en comparación a los valores de menor magnitud.

Decidimos normalizar las entradas para poder crear una red que es compatible con diferentes terrenos y no solo con el terreno con el que entrenamos a la red.

Pesos Iniciales

Para definir los pesos iniciales de cada neurona se implementó un algoritmo que toma en cuenta la cantidad de entradas y salidas de cada una de ellas dependiendo de la capa en la que se encuentran y si es interna o externa.

Se comienza calculando los límites de un intervalo del cual se extraerá un número aleatorio que será el peso asignado. El límite izquierdo y derecho del intervalo tienen el mismo módulo y distinto signo, se calculan mediante el inverso del cuadrado de la cantidad de entradas a cada neurona de la capa ya sea interna o externa.

Mejoras

Diversos factores se deben tener en cuenta con el fin de obtener mejor eficiencia en el aprendizaje de una red neuronal. Entre los agentes de mayor importancia se encuentran el momentum y learning rate adaptativo. Estos buscan acelerar el proceso de entrenamiento y aprendizaje.

Learning Rate Adaptativo

Con respecto al learning rate, implementamos un algoritmo adaptativo. El mismo consiste en una corrección continua de este factor. A medida que cambian los pesos, se calcula un vector de error cuadrático que resulta de hacer la diferencia entre los resultados deseados y los obtenidos (Esta representación del error, se hará en todas las épocas en el caso de la implementación incremental y cada cierta cantidad de épocas en la de Batch). Una vez obtenido este error, se lo comparará con el anterior y en el caso de que el actual sea menor que el anterior, asumimos que vamos por el camino correcto. Luego de cierta cantidad de aciertos, se incrementará el learning rate para avanzar más rápido, es decir consumir menos tiempo.

En caso contrario, si el error actual es mayor al anterior, estamos en el mal camino, y lo correcto es deshacerse de todo y volver a los pesos anteriores obtenidos correctamente. En este segundo caso, el learning rate es decrementado para priorizar correctitud a tiempo.

Momentum

En un problema ideal, la función de error se vería de esta manera, **Figura 2.**

Entonces nunca caería en un mínimo local ya que no tiene ninguno. Sin embargo, la función de error real es mucho más compleja y puede tener varios mínimos locales. Los mínimos locales hacen que la optimización se trabe.

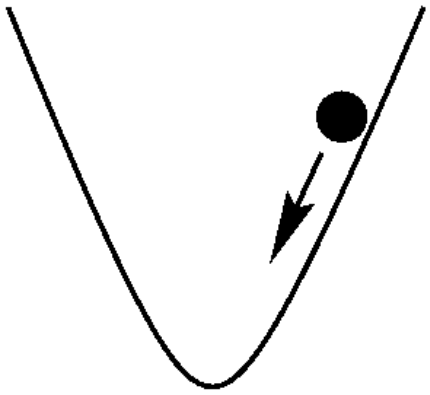


Figura 2. Función de error sin mínimos locales

En este caso, es probable que el algoritmo se encuentre con un mínimo local y piense que llegó a un mínimo global generando resultados subóptimos. Para evitar situaciones como esta, utilizamos momentum en la función delta (ΔW). El momentum es un valor entre 0 y 1 que incrementa el tamaño de los pasos hacia el mínimo. Un valor de momentum muy grande quiere decir que la convergencia hacia un mínimo va a ser mas rapida.

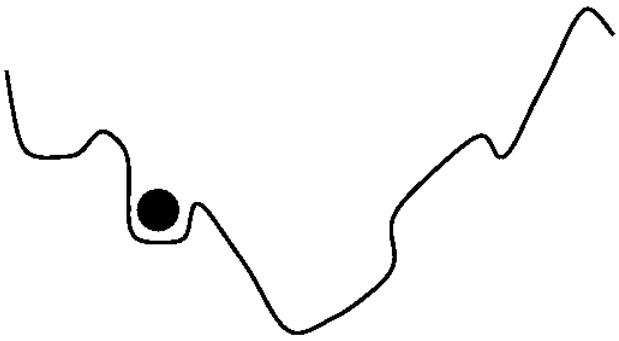


Figura 3. Funcion de error con minimos locales

El valor del momentum se debe elegir teniendo en cuenta que:

- Si tomo un valor de momentum y de η demasiado grande, podría causar que se saltee el mínimo.
- Si tomo un valor de momentum demasiado chico, no tendría suficiente efecto en la función delta (ΔW) como para evitar mínimos locales y haría más lento el proceso de aprendizaje.

También, el momentum es útil para 'suavizar' la función de error.

Variables pasadas por parámetro

Decidimos hacer parametrizables una serie de variables para asemejarnos, lo más posible, a un problema real. Dentro de los mismos se encuentran:

- **Terrenos:**

  Con el objetivo de que la red neuronal implementada no sea solo útil para el aprendizaje de un terreno en particular, no solo se puede intercambiar con cualquier terreno, sino con cualquier archivo de conjunto de puntos, sin importar lo que representan.

  Un ejemplo concreto es el archivo de xor, también incluido en el material de entrega. El archivo de entrada deberá constar de una lista de patrones (entradas) y una columna de resultados por cada patrón (salidas).

  Del mismo se separará una matriz de patrones y una de objetivos para ingresar a la red neuronal.

- **Learning Rate:**

El usuario podrá decidir con qué learning rate desea comenzar el proceso, asimismo podrá elegir si ese learning rate se mantendrá durante todo el proceso (si elige que el mismo sea 'fixed') o si irá variando de forma adaptativa (si elige la opción 'adaptive').

En el primer caso, la eficiencia del sistema dependerá del learning rate que el cliente le brinda, si el mismo es muy bajo, entonces se tomará demasiado tiempo en obtener un resultado, en caso contrario, se producirá un sobreaprendizaje y un entrenamiento incorrecto del sistema.

Mientras que de forma adaptativa, el learning rate irá variando dependiendo de las salidas obtenidas. De esta forma mientras más respuestas correctas obtenga, más rápido llegaremos a la solución final.

Por último, en caso de utilizar la evaluación dinámica del learning rate, el sistema brinda la posibilidad de establecer cada cuantas épocas se modificará este parametro con el objetivo de no obtener un learning rate demasiado variable.

- **Epsilon:**

Esta variable es de suma importancia ya que representa el límite para decidir si una respuesta es correcta o incorrecta. Cuando se entrena la red neuronal, se considerará aprendido un conjunto de patrones si para cada uno de ellos el error cuadrático entre el valor deseado y el calculado es menor a epsilon.

Si el valor de esta variables es demasaido grande, la red neuronal aceptará como correcto un resultado que quizas esta demasiado lejos del resultado deseado realmente. Mientras que si es demasiado pequeño, se tomará mucho tiempo en obtener los resultados, ya que deberá encontrar exactamente los deseados.

En muchos casos, esto no es siquiera posible debido a que la red tratará de aproximar puntos que realmente no aprendio.

- **Épocas:**

Las épocas serán un límite de corte para el sistema. Al llegar a la máxima cantidad de iteraciones establecidas por las épocas, se obtendrán los resultados calculados hasta el momento para los pesos.

Si este parámetro es demasiado grande, probablemente se obtendrá el valor deseado porque la condición de corte será que el error sea menor al epsilon establecido. En caso contrario, los pesos seguramente distarán mucho de ser los correctos por el poco tratamiento que recibieron.

- **Función de activación:**

  Es referida en el programa como 'training type' y será tratada en profundidad en breve.

- **Momentum:**

  Está variable, ya mencionada anteriormente, es también parametrizable. El programa ofrece la opción de incluirla o no en el algoritmo.

- **Capas ocultas y neuronas:**

  El sistema ofrece la posibilidad de elegir la cantidad de capas ocultas al igual que la cantidad de neuronas en cada una de ellas.

  Si se usan demasiadas neuronas en una capa, esto podría llevar a un 'overfitting', es decir, cuando la red funciona bien con los patrones de entrenamiento pero no cuando se lo prueba con otros patrones. Esto se debe a que el modelo termina memorizando y no realmente aprendiendo. Por otro lado, pocas neuronas por capas causarían 'underfitting' que es cuando la red produce resultados subóptimos con los patrones de entrenamiento ya que no puede establecer una relación entre los las entradas y las salidas.

Función de Activación

Tal como el enunciado lo solicita se implementó la posibilidad de usar como función de activación tanto la función exponencial como la tangente hiperbólica.

La función de activación cumple un rol fundamental a la hora de actualizar los pesos de la red ya que se le aplica la derivada de esta a la suma pesada de los enlaces entre neuronas para poder obtener una cota de error.

En caso de elegir como función de activación la tangente hiperbólica debemos tener en cuenta que su imagen varía entre -1 y 1 por lo tanto si dentro de nuestros 'targets', salidas deseadas, tenemos valores mayores a uno la red no los aprenderá correctamente.

Si bien nuestro terreno no posee valores mayores a uno si quisiéramos que la red aprenda otro terreno que posea dicha característica no lo aprendería, arrojaría valores incorrectos para el testeo del mismo.

¿Cómo lo solucionamos? Decidimos especificar una función de activación para las unidades de las capas internas y una función de activación para la capa de salida, esta última puede ser cualquier función que no acote la imagen de la misma, por ejemplo, una función lineal.

Ahora si, la red no solo será capaz de aprender nuestro terreno si no que también estará capacitado para solucionar cualquier tipo de terreno que se le presente aportandole mayor portabilidad al proyecto.

Si bien Octave posee un paquete para calcular la derivada de las funciones de activación decidimos y consideramos apropiado incluirlas en un archivo dentro del proyecto para evitar la descarga del mismo.

¿Incremental o Batch?

Para poder obtener los pesos deseados se implementó lo que se conoce como algoritmo de backpropagation. Este consiste en avanzar a través de todas las capas con el método de feed-forward, almacenar el error y una vez llegado a la salida 'volver' sobre los pesos realizando las modificaciones pertinentes.

Dentro de las variaciones de dicho algoritmo se destacan el método incremental y el método batch, la principal diferencia entre estos se presenta al momento de actualizar los pesos.

En el método incremental, los pesos se modifican en cada 'pattern', es decir, en cada época los pesos se habrán modificado tantas veces como pares de entradas allá. Por el contrario, en el método batch, modificaremos los pesos al finalizar cada época, es decir, luego de pasar con todos los pares de entradas posibles se acumulará el error y modificará los pesos respectivamente.

RESULTADOS

A continuación, detallaremos a través de tablas y gráficos los resultados obtenidos teniendo en cuenta el tipo de entrenamiento utilizado, condición de corte, función de activación, cantidad de capas ocultas y cantidad de neuronas en cada una de ellas, porcentaje de patterns utilizados para el entrenamiento de la red, entre otros.

Cabe destacar que incluiremos el tiempo que tardo la red en obtener un resultado, pero esto no refleja necesariamente cuan optimizada esta o no la red dado que muchas veces la velocidad de procesamiento depende del dispositivo utilizado. [1]

Sección 1: [2]

Razón de corte: Error [*epsilon = 0.01*]
Funcion de activacion: tanh(x)
Tipo de entrenamiento: Incremental

Cantidad de capas ocultas: 2
Cantidad de neuronas en cada capa: [13 14]
Porcentaje de entrenamiento: 50%

Tiempo (s)	Épocas	Mejoras	$\eta_{inicial}$	η_{final}	m	Error	Success Rate[3]
211.677	**25**	**No**	**0.1**	**0.1**	**0**	**0.01**	**71.9457 %**
272.035[A]	**41**	**η**	**0.1**	**0.122**	**0**	**0.01**	**73.3032 %**
447.806	57	η	0.01	0.0338	0	0.01	71.0407 %

Tabla 1

[1] Se adjuntan gráficos en el Anexo. Referencias en los tiempos
[2] m = valor de momentum
[3] Testeo

Cantidad de capas ocultas: 3 Cantidad de neuronas en cada capa: [15 15 15] Porcentaje de entrenamiento: 50%							
Tiempo	Épocas	Mejoras	$\eta_{inicial}$	η_{final}	m	Error	Success Rate[4]
323.246[B]	23	η + m	0.01	0.015604	0.9	0.01	62.8959 %
426.942	29	η + m	0.01	0.0196	0.9	0.01	65.6109 %
499.417	33	η + m	0.01	0.019604	0.9	0.01	61.991 %
478.083	**36**	**η + m**	**0.01**	**0.021604**	**0.9**	**0.01**	**70.1357 %**
1500.01	78	η + m	0.01	0.0187	0.9	0.01	70.1357%

Tabla 2 – Variables constantes. Efectos de Random

Cantidad de capas ocultas: 2 Cantidad de neuronas en cada capa: [3 3] Porcentaje de entrenamiento: 50%							
Tiempo	Épocas	Mejoras	$\eta_{inicial}$	η_{final}	m	Error	Success Rate[5]
21.8814	26	No	0.1	0.1	0	0.01	74.2081 %
30.7502	32	No	0.1	0.1	0	0.01	64.2534 %
72.9128[C]	**84**	**No**	**0.2**	**0.2**	**0**	**0.01**	**75.5656 %**
17.6731	**21**	**η**	**0.1**	**0.104**	**0**	**0.01**	**67.4208 %**
41.1221	**50**	**η + m**	**0.1**	**0.116**	**0.5**	**0.01**	**74.2081 %**
101.18	122	η + m	0.1	0.04960	0.9	0.01	65.6109 %

Tabla 3 – Variabilidad en eta & momentum

[4] Testeo
[5] Testeo

Sección 2: [6]

Razón de corte: Error [*epsilon = 0.001*]
Funcion de activacion: tanh(x)
Tipo de entrenamiento: Incremental

Cantidad de capas ocultas: 2
Cantidad de neuronas en cada capa: [10 10]
Porcentaje de entrenamiento: 60%

Tiempo	Épocas	Mejoras	$\eta_{inicial}$	η_{final}	m	Error	Success Rate[7]
261.242	48	η + m	0.01	0.02560	0.5	0.001	62.6415 %
121.769	22	η + m	0.05	0.05700	0.5	0.001	71.6981 %
67.9344	**14**	**η + m**	**0.1**	**0.10200**	**0.5**	**0.001**	**73.962 %**
247.885	50	η + m	0.01	0.02760	0.6	0.001	67.9245 %
109.695	**22**	**η + m**	**0.05**	**0.05500**	**0.6**	**0.001**	**70.9434 %**
90.2761	18	η + m	0.1	0.10400	0.6	0.001	60.3774 %
344.602	**68**	**η + m**	**0.01**	**0.033604**	**0.7**	**0.001**	**74.3396 %**
81.6888	16	η + m	0.05	0.10400	0.7	0.001	71.6981 %
102.798	21	η + m	0.1	0.10600	0.7	0.001	68.8113 %

Tabla 4 – Cambios incrementales en momentum & eta

[6] m = valor de momentum
[7] Testeo

Cantidad de capas ocultas: 2 Porcentaje de entrenamiento: 60%							
Tiempo	Épocas	Mejoras	$\eta_{inicial}$	η_{final}	m	Capas	Success Rate[8]
67.9344	17	η + m	0.1	0.10200	0.5	[10 10]	73.962 %
344.602	13	η + m	0.01	0.033604	0.7	[10 10]	74.3396 %
26.5412	**14**	**η + m**	**0.1**	**0.10200**	**0.5**	**[5 5]**	**78.1132 %**
199.138	10	η + m	0.01	0.033604	0.7	[5 5]	69.434 %
47.9289	24	η + m	0.1	0.11200	0.5	[3 3]	55.8491 %
136.863	96	η + m	0.01	0.04560	0.7	[3 3]	69.434 %

Tabla 5 – Cambios incrementales en momentum & eta para distintas arquitecturas

Cantidad de capas ocultas: 2 Cantidad de neuronas en cada capa: [3 3] Porcentaje de entrenamiento: 50%							
Tiempo	Épocas	Mejoras	$\eta_{inicial}$	η_{final}	m	Error	Success Rate[9]
112.784	17	m	0.05	0.05	0.5	0.001	66.7925 %
100.564	**13**	**m**	**0.05**	**0.05**	**0.6**	**0.001**	**75.0943 %**
49.7843	10	m	0.05	0.05	0.7	0.001	72.4528 %
188.428	24	m	0.05	0.05	0.8	0.001	54.3396 %

Tabla 6 – Cambios incrementales en momentum

[8] Testeo
[9] Testeo

Sección 3: Neuronas vs. Success Rate

Razón de corte: Error [*epsilon = 0.01*]
Funcion de activacion: tanh(x)
Tipo de entrenamiento: Incremental

Cantidad de capas ocultas: 2 Porcentaje de entrenamiento: 50%						
Tiempo	Épocas	Mejoras	$\eta_{inicial}$	Capas	Error	Success Rate
35.0707	39	No	0.1	[5 5]	0.01	61.6541 %
32.6875	26	No	0.1	[5 8]	0.01	51.8797 %
79.7794	**42**	**No**	**0.1**	**[15 10]**	**0.01**	**74.4361 %**
148.946	34	No	0.1	[15 15]	0.01	72.9323 %

Tabla 7 – Neuronas vs. Success Rate

Sección 4: % de Entrenamiento vs. Success Rate

Razón de corte: Error [*epsilon = 0.01*]
Funcion de activacion: tanh(x)
Tipo de entrenamiento: Incremental

Cantidad de capas ocultas: 2 Cantidad de neuronas en cada capa: [10 10]						
Tiempo	Épocas	Mejoras	$\eta_{inicial}$	% Entrenamiento	Error	Success Rate
19.9676	26	No	0.1	10	0.01	51.1111 %
67.4614	50	No	0.1	20	0.01	62.9213 %
79.7794	50	No	0.1	30	0.01	67.6692 %
128.794	**50**	**No**	**0.1**	**40**	**0.01**	**69.9998 %**

Tabla 8 – Entrenamiento vs. Success Rate

Análisis de Resultados:

Analicemos los resultados obtenidos en función de las tablas mostradas.

Podemos observar que en la **tabla 2** se mantuvieron las variables constantes con el fin de reflejar la importancia de una buena elección de patrones de entrenamiento ya que si no se elijen puntos lo suficientemente dispersos donde la función por ejemplo represente picos, la red no podrá luego aproximar de forma adecuada los valores. Además, la tabla deja al descubierto que, dependiendo del *rand* generado para los pesos, será la velocidad y capacidad de resolución de la misma.

Considerando la **tabla 4** donde se aplicaron dos modificaciones simultaneas; eta adaptativo & momentum. Se analizaron tres valores distintos de momentum y a su vez para cada uno de ellos probamos otros tres valores de eta inicial. ¿Qué resultados obtuvimos? La red posee un mejor porcentaje de success rate cuando se combina un valor alto de momentum con un valor bajo de eta inicial. En este caso, el mejor resultado se obtuvo cuando el momentum valía 0.7 y el eta inicial 0.001.

La mayoría de las veces el algoritmo de entrenamiento finaliza por la cota aplicada al error cuadrático medio y no por épocas.

La cantidad de épocas necesarias para obtener los pesos deseados depende de la cantidad de patrones de entrenamiento y los randoms realizados para la inicialización de los pesos.

No siempre la combinación de eta adaptativo y momentum nos da mejores resultados. Por ejemplo, en la **tabla 3** queda en evidencia que el hecho de no aplicarle ninguna mejora al algoritmo para dicha arquitectura es mejor que aplicarle implementar tanto eta adaptativo como momentum, pero a su vez solo este es mejor que aplicarle solo eta adaptativo.

Pero, por otro lado, en la **tabla 1**donde la arquitectura no es mas [3 3] si no que [13 14], cada elemento indica la cantidad de neuronas en las respectivas capas, se puede observar que aplicarle eta adaptativo brinda un mayor porcentaje de success rate que no hacerlo.

Además, en las **tablas 7 & 8** se busca mostrar como el success rate cambia dependiendo de la cantidad de neuronas en cada capa y del porcentaje de patrones de entrenamiento respectivamente.

Cuanto mayor es el porcentaje de patrones de entrenamiento, mayor es el success rate.

En **la tabla 7** no pudimos sacar una tendencia ya que más neuronas por capas no siempre se traduce en un mejor resultado. Por ejemplo, dos capas de 5 neuronas cada una produjeron un mejor resultado que una variante de la red con una capa de 5 neuronas y otra de 8. Agregando, una red con una capa de 15 neuronas y otra de 10 produjo un mejor resultado que las mencionadas anteriormente e incluso mejor que una red con dos capas de 15 neuronas cada una.

La función sigmoide es de gran utilidad para representar probabilidades ya que devuelve valores entre 0 y 1. Sin embargo, estos valores pequeños se convierten en un problema cuando se juntan con backpropagation ya que el gradiente rápidamente se vuelve demasiado pequeño, imposibilitando el cambio del peso. Este problema es conocido como 'Vanishing Gradient Problem'.

CONCLUSIONES

Al comenzar el proyecto el grupo se planteó como principal objetivo lograr encontrar una arquitectura óptima, tal como explicamos al inicio del informe, consideramos que una arquitectura óptima es aquella que encuentre los pesos adecuados de modo que luego nos provea el mejor success rate posible a la hora de generalizar la red.

Recordemos que denominamos success rate a la capacidad de aprendizaje de la red, es decir, luego de ser entrenada, cuan capaz es de aproximar el terreno.

Es de suma importancia aclarar hay quienes consideran que una arquitectura óptima es aquella que te arroja los resultados del algoritmo en menor tiempo posible sin considerar el nivel de aprendizaje o testeo.

Luego de varios testeos donde variamos algunos parámetros como por ejemplo; el eta, el momentum, la función de activación, la propia arquitectura de la red, agregando y quitando capas, modificando la cantidad de neuronas en cada una de ellas, entre otras, concluimos que si bien existe determinada tendencia no existe una arquitectura en particular que satisfaga todas nuestras expectativas en su totalidad ya que dependiendo de las mejoras o no, implementadas en el algoritmo existen varias arquitecturas que dan como resultado un success rate similar.

En base a los resultados obtenidos y analizados consideramos que una buena arquitectura no necesita más de 2 capas ocultas, ya que si ponemos 3 no logramos un mayor success rate, simplemente, tardamos más.

En cuanto a la cantidad de neuronas en cada una de ellas notamos que cuanto mayor es la cota de error, el epsilon, y mayor es el eta - constante- mayor es la cantidad de neuronas necesarias. Recomendamos usar entre 3 y 10 neuronas en cada una de ellas dependiendo del tamaño del conjunto de patrones de entrenamiento.

A mayor patrón de entrenamiento, mayor cantidad de neuronas son necesarias para poder aprender y considerar con mayor exactitud los puntos que se encuentran en los picos y valles de la función.

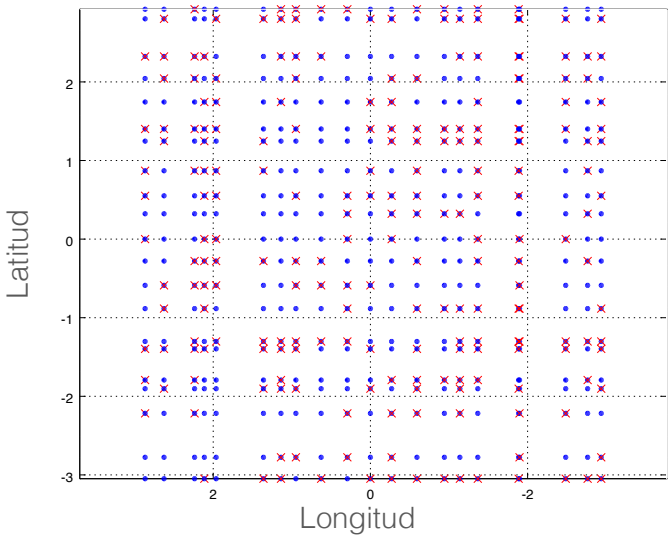
Con respecto a las mejoras implementadas en el algoritmo, eta adaptativo & momentum notamos que cuando aplicamos ambos efectivamente las posibilidades de caer en un mínimo disminuyen. No solo eso, si no que, además, obtenemos altos porcentajes de success rate.

Un detalle no menor es que cuanto más grande sea el momentum más pequeño debe ser el learning rate inicial. El valor para momentum, que en general, brindó un mejor resultado fue de 0.9.

La tangente hiperbólica fue la funcion de activacion que nos dio los resultados más óptimos ya que la función exponencial nos causó el problema del 'Vanishing Gradient' el cual explicamos anteriormente.

En conclusión, la red ha logrado aprender el terreno indicado con un porcentaje de acierto de entre 73 % y 75 % de confiabilidad considerando entre un 40 % y 50 % de los patrones dados.

Tabla 1. A. Learning Rate Adaptativo



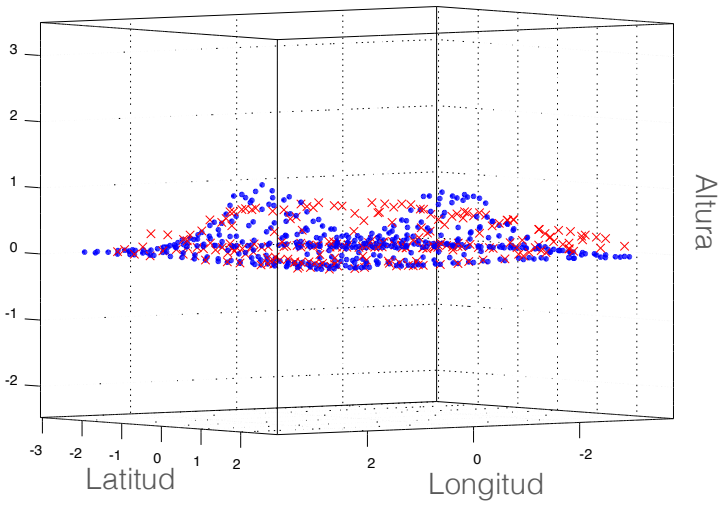


Figura 4 [izquierda] & Figura 5 [derecha]. Aciertos de testeo 73.3032 %. Donde:

Reseña:

Learning Rate adaptativo
Error Cuadratico Medio
DeltaError

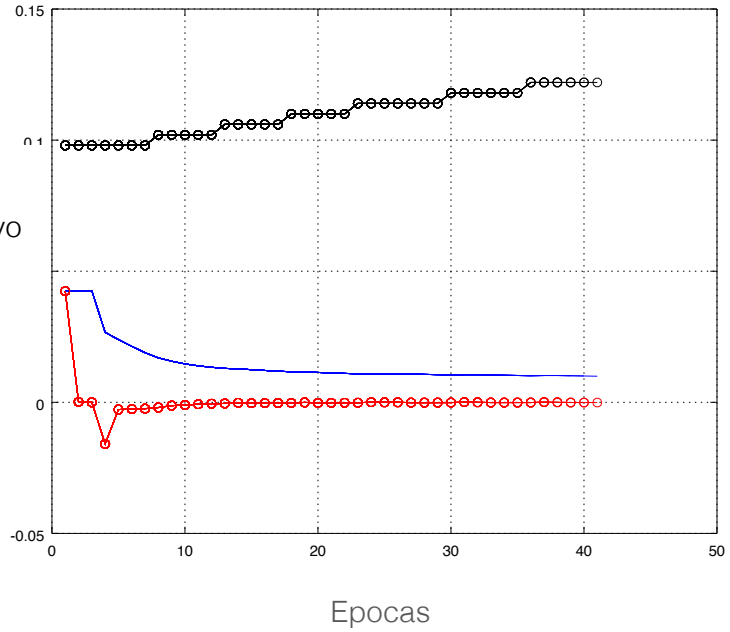
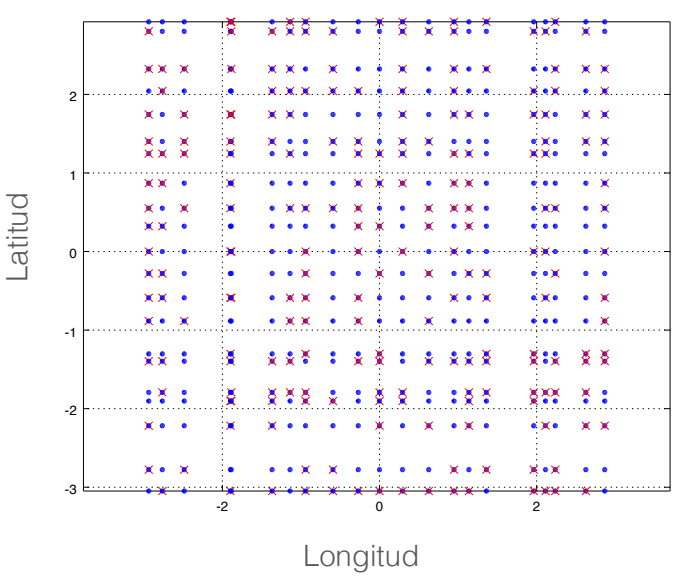


Figura 6 [derecha]. Gráficos de error y modificación del *learning rate*

Tabla 2. B. Learning Rate Adaptativo + Momentum



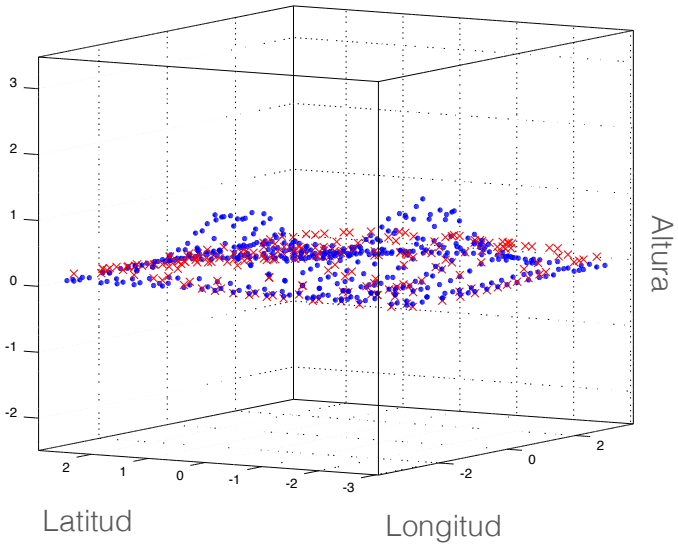


Figura 7 [izquierda] & Figura 8 [derecha]. Aciertos de testeo 62.8959 %. Donde: Real | Aproximado

Reseña:

Learning Rate adaptativo
Error Cuadratico Medio
DeltaError

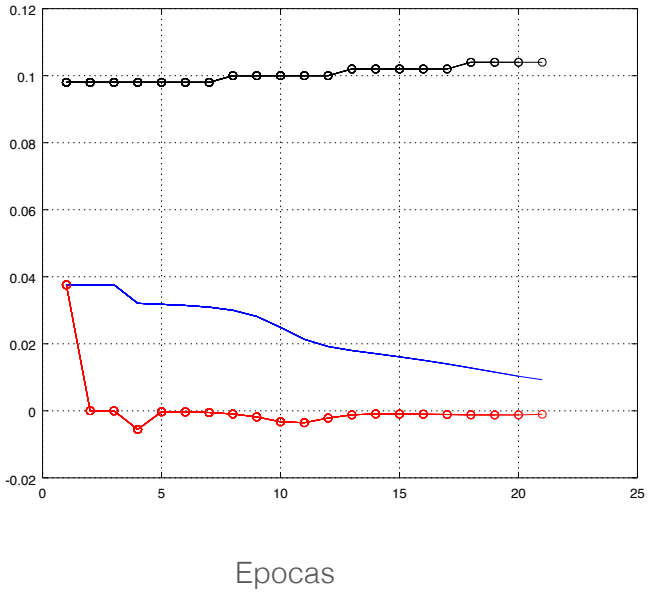
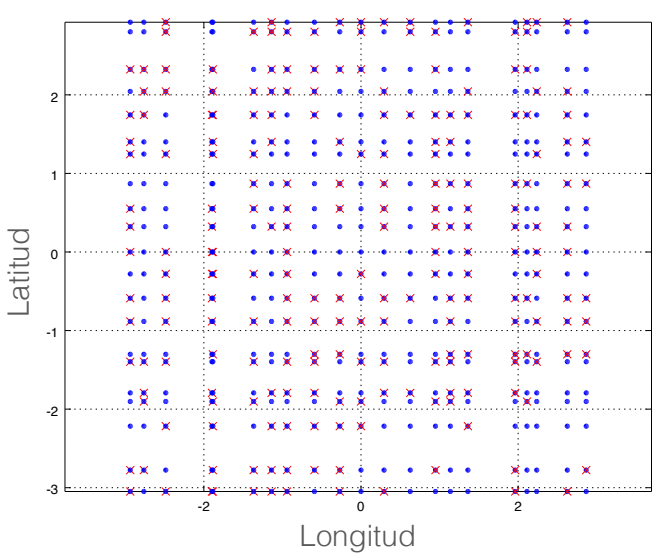


Figura 9 [derecha]. Gráficos de error y modificación del *learning rate & implementación momentum*

Tabla 3. C. Sin mejoras en el Algoritmo



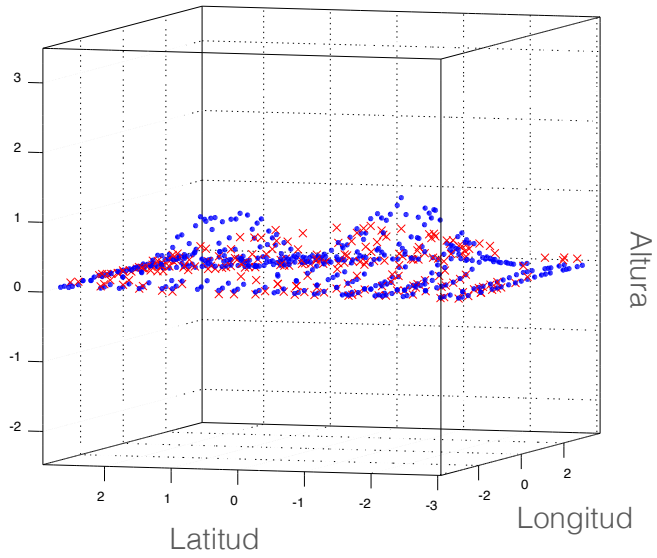


Figura 10 [izquierda] & Figura 11 [derecha]. Aciertos de testeo 75.5656 %. Donde: Real | Aproximado

Reseña:

Learning Rate adaptativo
Error Cuadratico Medio

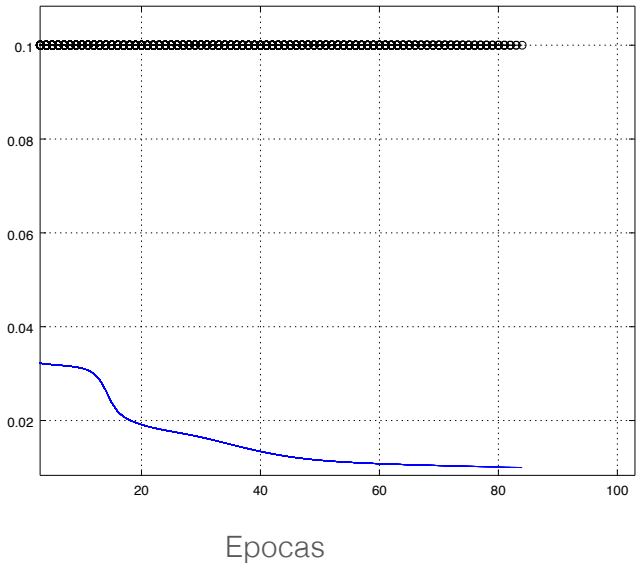


Figura 12 [derecha]. Gráficos de error y learning rate sin mejoras

Implementación del Algoritmo con Batch